Name: **Ong Jing Wei**
Student ID: **32909764**

Generative AI was used in this assignment

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)**

In this dataset, there are 1261 legitimate sites and 739 phishing sites, hence the proportion of phishing sites to legitimate sites is 0.3695.

```
> phishing_site <- sum(PD$Class == 1)
> legitimate_site <- sum(PD$Class == 0)
> proportion <- phishing_site / (phishing_site + legitimate_site)
> proportion
[1] 0.3695
```

All the independent variables in this dataset are numerical, hence the mean, 1st quartile, median, 3rd quartile, min and max is listed using summary function:

```
> summary(PD)
      A01             A02                A03                A04             A05                A06
 Min.   : 3.00   Min.   : 0.0000   Min.   :0.000000   Min.   :2.000   Min.   : 0.00000   Min.   :0.000
 1st Qu.:17.00   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.:2.000   1st Qu.: 0.00000   1st Qu.:0.000
 Median :29.00   Median : 0.0000   Median :0.000000   Median :3.000   Median : 0.00000   Median :0.000
 Mean   :27.28   Mean   : 0.2735   Mean   :0.001514   Mean   :2.775   Mean   : 0.02066   Mean   :0.129
 3rd Qu.:40.00   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.:3.000   3rd Qu.: 0.00000   3rd Qu.:0.000
 Max.   :49.00   Max.   :74.0000   Max.   :1.000000   Max.   :8.000   Max.   :15.00000   Max.   :1.000
                 NA's   :18        NA's   :18         NA's   :21      NA's   :16         NA's   :16
      A07                A08              A09               A10               A11
 Min.   :0.000000   Min.   :0.1739   Min.   :0.00000   Min.   :0.00000   Min.   : 0.00000
 1st Qu.:0.000000   1st Qu.:0.6842   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.: 0.00000
 Median :0.000000   Median :1.0000   Median :0.00000   Median :0.00000   Median : 0.00000
 Mean   :0.002021   Mean   :0.8475   Mean   :0.02523   Mean   :0.03539   Mean   : 0.04934
 3rd Qu.:0.000000   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.: 0.00000
 Max.   :1.000000   Max.   :1.0000   Max.   :1.00000   Max.   :1.00000   Max.   :10.00000
 NA's   :21         NA's   :22       NA's   :18        NA's   :22        NA's   :14
      A12             A13                A14              A15              A16               A17
 Min.   : 19.0   Min.   : 0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.000
 1st Qu.:232.0   1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:1.000
 Median :232.0   Median : 0.00000   Median :0.0000   Median :0.0000   Median :0.00000   Median :1.000
 Mean   :318.3   Mean   : 0.01662   Mean   :0.1484   Mean   :0.1264   Mean   :0.04527   Mean   :1.168
 3rd Qu.:419.0   3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:1.000
 Max.   :692.0   Max.   :24.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :5.000
 NA's   :18      NA's   :14         NA's   :19       NA's   :14       NA's   :12        NA's   :16
      A18              A19              A20              A21               A22
 Min.   :   4.00   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.002836
 1st Qu.:  12.00   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.051037
 Median :  31.00   Median :0.0000   Median :0.0000   Median :0.00000   Median :0.058048
 Mean   :  55.62   Mean   :0.1077   Mean   :0.2213   Mean   :0.02482   Mean   :0.055832
 3rd Qu.:  89.00   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.062840
 Max.   :1888.00   Max.   :1.0000   Max.   :1.0000   Max.   :2.00000   Max.   :0.083258
 NA's   :25        NA's   :22       NA's   :16       NA's   :26        NA's   :26
      A23              A24              A25                Class
 Min.   :   0.00   Min.   :0.0000   Min.   :0.000000   Min.   :0.0000
 1st Qu.:   3.00   1st Qu.:0.0082   1st Qu.:0.000000   1st Qu.:0.0000
 Median :  60.50   Median :0.5229   Median :0.000000   Median :0.0000
 Mean   :  63.76   Mean   :0.2748   Mean   :0.000161   Mean   :0.3695
 3rd Qu.: 104.00   3rd Qu.:0.5229   3rd Qu.:0.000000   3rd Qu.:1.0000
 Max.   :1683.00   Max.   :0.5229   Max.   :0.211000   Max.   :1.0000
 NA's   :18        NA's   :33       NA's   :24
```

It is noteworthy that there are missing values in each of the variables except for A01, hence to calculate the standard deviation, we need to omit the NA values.

```
> std_dev <- sapply(numeric_columns, function(x) sd(x, na.rm = TRUE))
> std_dev
         A01          A02          A03          A04          A05          A06          A07          A08 
1.445754e+01 2.044781e+00 3.690133e-02 5.598153e-01 8.472855e-01 3.313275e-01 4.865703e-02 2.146528e-01 
         A09          A10          A11          A12          A13          A14          A15          A16 
1.497188e-01 1.962201e-01 1.349904e+00 1.439059e+02 2.117538e+00 3.519340e-01 3.385772e-01 2.005395e-01 
         A17          A18          A19          A20          A21          A22          A23          A24 
5.930480e-01 9.223330e+01 3.031197e-01 4.246784e-01 1.898714e-01 1.044803e-02 7.104474e+01 2.515965e-01 
         A25        Class 
4.128447e-03 4.861785e-01 
```

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

```
> total_na_rows <- sum(apply(PD, 1, function(x) any(is.na(x))))
> total_na_rows
[1] 419
```

The total number of rows containing at least one NA is 419, however our datasets only contains 2000 rows, hence removing NA rows will reduce dramatically to our dataset, hence NA values are handled with median imputation.

```
# handle missing values by median imputation
PD_imputed <- PD %>%
  mutate(across(where(is.numeric), ~ifelse(is.na(.), median(., na.rm = TRUE), .)))
```

Median is less sensitive to extreme values and outliers compared to mean, hence imputing with median helps in maintaining the original distribution of the data.

```
# convert Class variable to factor
PD_imputed$Class <- factor(PD_imputed$Class)
str(PD_imputed$Class)
```

Class variable is converted to factor as classification models in R require categorical variables to be in factor.

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.

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

The cleaned and imputed dataset is split into 70% training and 30% test set by using the given code and student ID as the random seed.

```
> set.seed(32909764) #Student ID as random seed
> train.row = sample(1:nrow(PD_imputed), 0.7*nrow(PD_imputed))
> PD_imputed.train = PD_imputed[train.row,]
> PD_imputed.test = PD_imputed[-train.row,]
> cat("Number of rows in the training set:", nrow(PD_imputed.train), "\n")
Number of rows in the training set: 1400
> cat("Number of rows in the test set:", nrow(PD_imputed.test), "\n")
Number of rows in the test set: 600
```

From the screenshot above, we can see that 2000 rows of dataset has been split into 1400 of training set and 600 of test set.

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

This five classification models is used to predict the target variable "Class" using the imputed training dataset as shown below:

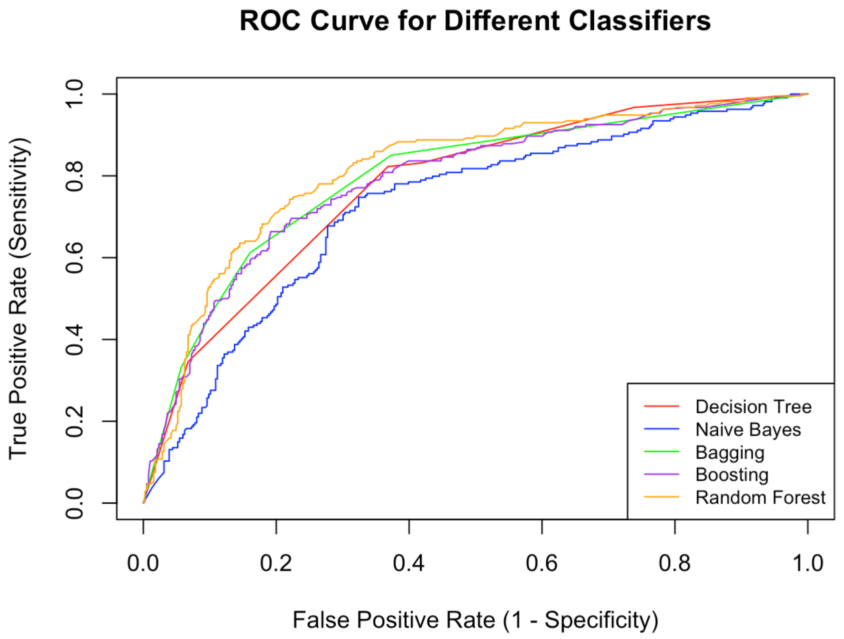
```
> PD.decisionTree <- tree(Class~., data = PD_imputed.train)
> PD.nBayes <- naiveBayes(Class~., data = PD_imputed.train)
> PD.bagging <- bagging(Class~., data = PD_imputed.train)
> PD.boosting <- boosting(Class~., data = PD_imputed.train)
> PD.randomForest <- randomForest(Class~., data = PD_imputed.train)
```

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)**

```
> accuracy_randomForest <- prediction_func(PD.randomForest)
               Actual_Class
Predicted_Class   0   1
              0 329  84
              1  57 130
[1] 0.765
> accuracy_decisionTree <- prediction_func(PD.decisionTree)
               Actual_Class
Predicted_Class   0   1
              0 360 140
              1  26  74
[1] 0.7233333
> accuracy_naiveBayes <- prediction_func(PD.nBayes)
               Actual_Class
Predicted_Class   0   1
              0  15   2
              1 371 212
[1] 0.3783333
> accuracy_bagging <- prediction_func(PD.bagging)
               Actual_Class
Predicted_Class   0   1
              0 324  70
              1  62 144
[1] 0.78
> accuracy_boosting <- prediction_func(PD.boosting)
               Actual_Class
Predicted_Class   0   1
              0 315  81
              1  71 133
[1] 0.7466667
> accuracy_randomForest <- prediction_func(PD.randomForest)
               Actual_Class
Predicted_Class   0   1
              0 329  83
              1  57 131
[1] 0.7666667
```

The codes above predict and evaluate the performance of different classification models including decision tree, naïve bayes, bagging, boosting and random forest to classify the test cases as phishing (1) or legitimate (0).

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)**




```
> auc_decisionTree
[1] 0.7740545
> auc_naiveBayes
[1] 0.7233003
> auc_bagging
[1] 0.7942654
> auc_boosting
[1] 0.7868808
> auc_randomForest
[1] 0.8145066
```

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

	Decision Tree	Naïve Bayes	Bagging	Boosting	Random Forest
Accuracy	0.7233333	0.3783333	0.7583333	0.7466667	0.7766667
AUC	0.7740545	0.7233003	0.7942654	0.7868808	0.8145066

From the table above, we can see that Random Forest has the highest accuracy and highest AUC. This indicates that it is the best performing classifier in terms of both accuracy and the ability to distinguish between phishing and legitimate classes. As high accuracy indicates that the classifier correctly identifies a high percentage of phishing and legitimate websites, while high AUC indicates that the model has a good measure of separability and is better at distinguishing between the two classes.

```
> classifier_table
     Classifier  Accuracy       AUC
1 Decision Tree 0.7233333 0.7740545
2   Naive Bayes 0.3783333 0.7233003
3       Bagging 0.7583333 0.7942654
4      Boosting 0.7466667 0.7868808
5 Random Forest 0.7766667 0.8145066
```

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)**

```
> imptVar_decisionTree <- summary(PD.decisionTree)
> imptVar_decisionTree

Classification tree:
tree(formula = Class ~ ., data = PD_imputed.train)
Variables actually used in tree construction:
[1] "A01" "A18" "A23"
Number of terminal nodes:  5
Residual mean deviance:  1.052 = 1467 / 1395
Misclassification error rate: 0.2664 = 373 / 1400
```

```
> imptVar_bagging <- sort(PD.bagging$importance, decreasing = TRUE)
> imptVar_bagging
       A01        A18        A23        A22        A08        A17        A06        A24        A16
40.7949779 24.4501186 16.3396412  8.3314794  4.6553693  2.1546314  1.1612667  1.0528148  0.6489468
       A14        A02        A03        A04        A05        A07        A09        A10        A11
 0.4107540  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
       A12        A13        A15        A19        A20        A21        A25
 0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
```

```
> imptVar_boosting <- sort(PD.boosting$importance, decreasing = TRUE)
> imptVar_boosting
       A01        A22        A18        A23        A08        A24        A12        A02        A06
28.5494181 20.2592608 16.1208907 11.2581271  5.7041837  4.1468276  3.0404717  1.9875727  1.5452873
       A17        A14        A19        A15        A09        A10        A16        A04        A21
 1.5446474  1.0630909  1.0479887  0.7684302  0.5539354  0.5095741  0.4496668  0.4341180  0.4027135
       A20        A11        A03        A05        A07        A13        A25
 0.3264160  0.2873792  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
```

```
> imptVar_randomForest <- PD.randomForest$importance[order(-PD.randomForest$importance),]
> imptVar_randomForest
         A01          A18          A22          A23          A08          A24          A12          A14
109.50660750  91.06306733  86.26787679  80.96294346  43.15701745  31.54998364  29.79147296  14.73650940
         A17          A20          A04          A02          A06          A15          A19          A16
 13.69151398  12.42380409  11.30847845   9.57434193   8.39708422   8.01506127   7.70395292   4.63539919
         A09          A10          A11          A21          A05          A07          A13          A25
  4.38339190   3.41920864   2.21589285   2.07369735   0.11333038   0.09421338   0.09189386   0.06809346
         A03
  0.02206543
```

	1	2	3	4	5	6	7	8	9	10
Decision Tree	A01	A18	A23							
Bagging	A01	A18	A23	A22	A08	A17	A06	A24	A16	A14
Boosting	A01	A22	A18	A23	A08	A24	A12	A02	A06	A17
Random Forest	A01	A18	A22	A23	A08	A24	A12	A14	A17	A20

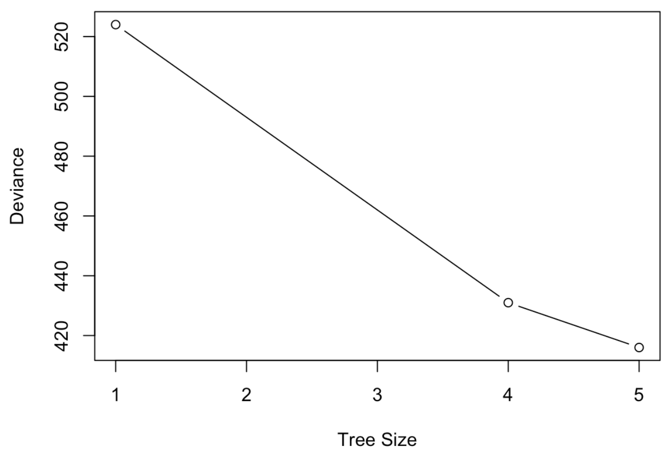
Naïve Bayes does not inherently provide variable importance like other models because it is a probabilistic classifier and does not produce an internal metric for assessing the importance of each feature in the same way that models build an explicit structure like trees do.

From the table above, we can see that A01 is the most important variables among the four classifiers in predicting phishing and legitimate website as it has the highest importance value compared to other variables. Followed by variables A18, A23 and A22. These variables appear frequently across the models and have the highest importance scores. The remaining variables in the dataset have comparatively less impact to the classifier's prediction.

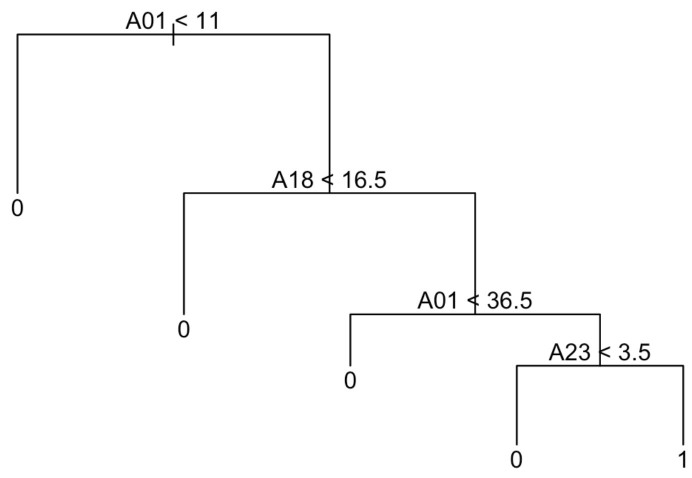
Variables that consistently show very low or 0 importance scores across multiple models could be omitted from the data. The variables could be omitted from the data will be A03, A05, A07, A13, A25 as they have relatively low in all the classifiers. Omitting these less important variables can simplify the model and reduce overfitting without significantly affecting the performance of the model.

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**)

Decision tree model in question 4 is selected and cross-validation is performed to find the optimal complexity of the tree and prune it as pruned decision tree model is simpler and easier for a person to use. The tree splits the data based on the most important variables identified in question 8.



From this graph, we can see that the size 5 has the lowest deviance. This indicates that a tree with 5 terminal nodes provides the best balance between complexity and predictive accuracy. Therefore, a tree with size of 5 is chosen for the pruned model.



A01, A18 and A23 were chosen based on their importance score as determined from the summary of decision tree in question 8. These variables contribute significantly to the model's ability to distinguish between phishing and legitimate website. A01 is used at the root as it has the highest importance score indicating that it is a strong predictor. A18 and A23 also highly important and used to further split the data to improve classification accuracy.

A person can manually classify between phishing and legitimate by looking at the graph. They need to measure the attribute A01 of the website and if it is less than 11 then we can immediately classify the site as legitimate, while if it is more than 11, they need to measure the attribute A18 of that website. If it is less than 16.5, they can classify it as legitimate, while if it is more than 16.5 then proceed to measure A01. If A01 is less than 36.5 then the website will be legitimate, while if it is more than 36.5 then measure for A23. If A23 is smaller than 3.5 then legitimate, or else it will be phishing.

This step is simple to follow and only need to know the thresholds for these attributes to classify a site without needing to understand or implement the underlying the machine learning model.

```
> cm_simpleTree_pruned
      Predicted
Actual   0   1
     0 360  26
     1 140  74
> accuracy_simpleTree_pruned <- sum(diag(cm_simpleTree_pruned)) / sum(cm_simpleTree_pruned)
> accuracy_simpleTree_pruned
[1] 0.7233333
> auc_simpleTree_pruned
[1] 0.7740545
```

The pruned decision tree has an accuracy of 0.723333 and AUC of 0.7740545, it has the same accuracy and AUC as the original decision tree. This demonstrates that the simplified model retains the effectiveness while being easier to interpret and use manually.

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)**

Random Forest classifier is chosen for this question as it demonstrated the highest accuracy and AUC among all classifiers in question 4. The random forest model showed robust performance, making it the best choice for further improvement.

To optimize the Random Forest model, I performed cross-validation and tuned the hyperparameters.

```
set.seed(32909764)
tuneGrid <- expand.grid(
  mtry = c(2, 3, 4, 5) # Number of variables randomly sampled as candidates at each split
)
trainControl <- trainControl(method = "cv", number = 5)

rf_tuned <- train(
  Class ~ ., data = PD_imputed.train, method = "rf",
  tuneGrid = tuneGrid,
  trControl = trainControl,
  ntree = 500, # Number of trees in the forest
  nodesize = 5 # Minimum size of terminal nodes
)

best_model <- rf_tuned$finalModel
rf_tuned$bestTune
```

Several factors were considered to improve the model's performance where the 'mtry' parameter determines the number of variables randomly sampled as candidates at each split. A grid search approach is used to evaluate different values of 'mtry' and identified the optimal configuration ranging from 2 to 5. By testing different values of mtry, the model's sensitivity to variable selection was assessed which aim to find the optimal balance between model complexity and accuracy.

The number of trees in the Random Forest was set to 500 so that this could improve the stability of the model by reducing the risk of overfitting. The minimum node size is 5 so that larger node size helps prevent the model from splitting nodes too finely which can lead to overfitting.

A 5-fold cross-validation was set up to evaluate different combinations of hyperparameters. This is to help to estimate how the model will perform on unseen data and ensures that the model not overly rely on specific subset of data. The model is then trained on the training data. The cross-validation process tested different values of 'mtry' to find the best one.

```
> rf_tuned$bestTune
  mtry
3    4
```

As we can see from the output, the best value of 'mtry' is 4.

The best tuned Random Forest was then evaluated on the test set, and the accuracy and the AUC of this model are 0.78 and 0.8201.

	Tuned Random Forest	Original Random Forest
Accuracy	0.78	0.7766667
AUC	0.8201	0.8145066

From the comparison table above, we can see that the tuned random forest's performance metrics show a slight improvement compared to the original Random Forest model. The tuning process improved the model's predictive capability, confirming that random forest remains the best classifier for this dataset.

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 pre-processing required. How does this classifier compare with the others? Can you give any reasons? **(4 Marks)**

To implement an artificial neural network classifier to predict whether a website is phishing or legitimate, we need to first preprocess the data. The 'Class' attribute in the training and testing datasets were converted to numeric form as this conversion is crucial because neural network require numeric input.

```
PD_imputed.train$Class <- as.numeric(recode(PD_imputed.train$Class, "1" = 1, "0" = 0))
PD_imputed.test$Class <- as.numeric(recode(PD_imputed.test$Class, "1" = 1, "0" = 0))
```

Next, we also need to ensure that there were no missing values in the datasets to ensure that neural network receives complete data.

```
# check if there's any NA
anyNA(PD_imputed.train)
anyNA(PD_imputed.test)
PD_imputed.test <- na.omit(PD_imputed.test)
PD_imputed.train <- na.omit(PD_imputed.train)
```

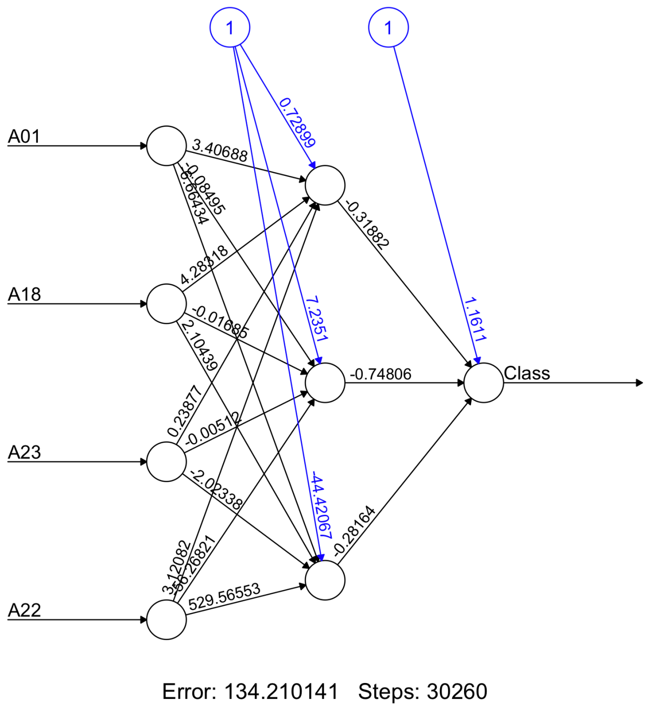
After data preprocessing, we trained a neural network using the 'neuralnet' package with the selected attributes A01, A18, A23, A22 as these attributes have the highest importance score based on previous models as these attributes consistently showed high importance and contribution to the model's performance. Initially, we trained the ANN model using all available attributes. However, the model did not perform optimally, indicating that some attributes might be introducing noise or redundancy. The neural network had one hidden layer with 3 neurons and a linear output.

For the performance evaluation of the trained neural network, predictions are made on the test set, while confusion matrix and AUC are used to calculate the accuracy and evaluate its performance.

```
> cm_PD.nn <- table(observed = PD_imputed.test$Class, predicted = PD.nn_predr$V1)
> cm_PD.nn
        predicted
observed   0   1
       0 315  71
       1  81 133
> accuracy_PD.nn <- sum(diag(cm_PD.nn)) / sum(cm_PD.nn)
> accuracy_PD.nn
[1] 0.7466667
> PD.nn_probs <- PD.nn_predictions$net.result
> PD.nn_pred <- prediction(PD.nn_probs, PD_imputed.test$Class)
> PD.nn_perf <- performance(PD.nn_pred, "tpr", "fpr")
> auc_nn <- performance(PD.nn_pred, "auc")
> auc_nn <- round(as.numeric(auc_nn@y.values), 4)
> auc_nn
[1] 0.7934
```

	Tuned Random Forest	Pruned decision tree	Neural Network
Accuracy	0.78	0.7233	0.7466667
AUC	0.8201	0.7741	0.7934

From the performance metrics of the neural network, we can see that the accuracy of 0.7466667 and AUC of 0.7934 performs better than the pruned decision tree in terms of both accuracy and AUC, but it is slightly outperformed by the tuned random forest. The outcome is expected as random forests typically provide robust performance due to their ensemble nature, which reduces overfitting and variance. Overall, the neural network classifier showed good performance, especially in comparison to the decision tree. However, the random forest remains the best model due to its higher accuracy and AUC. The choice of attributes and careful data preprocessing were crucial in achieving these results.



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)**

For this question, we implemented support vector machine (SVM) classifier using package 'e1071'. SVM is chosen as it performs well on smaller datasets, and it can handle high-dimensional data which is useful as our datasets have many features. By using the right kernel and regularization parameters, SVM can manage overfitting to ensure a better generalization on unseen data.

Before training the SVM model, we need to preprocess that data. We used median imputation to handle missing values in the numeric columns of the dataset. This approach is robust to outliers and ensures that the missing values are replaced with the median value of the respective column. Next, the dataset was split into training (70%) and test (30%) sets using a random seed based on the student ID for reproducibility. For compatibility with the SVM implementation, the class variable was converted to numeric. The numeric columns were scaled to ensure that all features contribute equally to the model. Properly scaled data enhances the stability of the SVM model, leading to more consistent and robust performance across different datasets and cross-validation folds. I also created a function to identify and remove columns with all null values from both the training and test datasets. The SVM model was trained using the e1071 package with the training data.

```
svm_model <- svm(Class ~ ., data = PD_imputed.train, probability = TRUE)
summary(svm_model)
```

Predictions were made on the test dataset, and the results were evaluated using a confusion matrix and accuracy.

```
> cm_svm <- table(Actual = PD_imputed.test$Class, Predicted = as.numeric(svm_predictions > 0.5))
> cm_svm
                    Predicted
Actual                 0   1
  -0.743962457499878 365  21
  1.3419135915652    142  72
> accuracy_svm <- sum(diag(cm_svm)) / sum(cm_svm)
> accuracy_svm
[1] 0.7283333
```

The SVM classifier achieved an accuracy of 72.83%, indicating a moderate performance in distinguishing between the two classes. When compared to the neural network and random forest models, the SVM's performance is slightly lower, particularly in terms of accuracy. The random forest model achieved the highest performance, likely due to its ensemble nature which effectively reduces overfitting and variance.

I evaluated four different kernels: linear, polynomial, radial basis function (RBF), and sigmoid. The performance metrics (accuracy and confusion matrix) for each kernel are as follows:

```
Kernel: linear
Accuracy: 0.67
                    Predicted
Actual                0   1
  -0.743962457499878 351  35
  1.3419135915652    163  51

Kernel: polynomial
Accuracy: 0.6833333
                    Predicted
Actual                0   1
  -0.743962457499878 368  18
  1.3419135915652    172  42

Kernel: radial
Accuracy: 0.7283333
                    Predicted
Actual                0   1
  -0.743962457499878 365  21
  1.3419135915652    142  72

Kernel: sigmoid
Accuracy: 0.63
                    Predicted
Actual                0   1
  -0.743962457499878 322  64
  1.3419135915652    158  56
```

Among the kernels tested, the Radial Kernel (RBF) performed the best with an accuracy of 0.7283. This suggests that the RBF kernel is well-suited for our dataset, likely because it can capture the non-linear relationships between features more effectively than linear or polynomial kernels.

References

I acknowledge the use of ChatGPT (https://chat.openai.com/) to generate R codes and refine the language for my own work.

Appendix

```
# Name: Ong Jing Wei
# Student ID: 32909764
library(dplyr)
#library(rpart)
library(ipred)
library(e1071)
library(adabag)
library(randomForest)
library(tree)
library(pROC)
library(ROCR)
library(neuralnet)

rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(32909764) # 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

#______________________________________________________________________________#
# Question 1
dim(PD)

class_distribution <- table(PD$Class)
class_distribution
prop.table(class_distribution)

phishing_site <- sum(PD$Class == 1)
legitimate_site <- sum(PD$Class == 0)

proportion <- phishing_site / (phishing_site + legitimate_site)
proportion

summary(PD)

numeric_columns <- Phish %>% select_if(is.numeric)
std_dev <- sapply(numeric_columns, function(x) sd(x, na.rm = TRUE))
std_dev

#______________________________________________________________________________#
# Question 2

# Total number of rows containing at least one NA
total_na_rows <- sum(apply(PD, 1, function(x) any(is.na(x))))
total_na_rows
```

```
# handle missing values by median imputation
PD_imputed <- PD %>%
  mutate(across(where(is.numeric), ~ifelse(is.na(.), median(., na.rm = TRUE), .)))

# convert Class variable to factor
PD_imputed$Class <- factor(PD_imputed$Class)
str(PD_imputed$Class)

#____________________________________________________________________________#
# Question 3

set.seed(32909764) #Student ID as random seed
train.row = sample(1:nrow(PD_imputed), 0.7*nrow(PD_imputed))
PD_imputed.train = PD_imputed[train.row,]
PD_imputed.test = PD_imputed[-train.row,]

# checking
cat("Number of rows in the training set:", nrow(PD_imputed.train), "\n")
cat("Number of rows in the test set:", nrow(PD_imputed.test), "\n")

#____________________________________________________________________________#
# Question 4

PD.decisionTree <- tree(Class~., data = PD_imputed.train)
PD.nBayes <- naiveBayes(Class~., data = PD_imputed.train)
PD.bagging <- bagging(Class~., data = PD_imputed.train, mfinal = 5)
PD.boosting <- boosting(Class~., data = PD_imputed.train, mfinal = 10)
PD.randomForest <- randomForest(Class~., data = PD_imputed.train)

#____________________________________________________________________________#
# Question 5

prediction_func <- function(model){


  if (inherits(model, "tree")){
    pred <- predict(model, PD_imputed.test, type = "class")
  } else if (inherits(model, "naiveBayes")){
    pred <- predict(model, PD_imputed.test)
  } else if (inherits(model, "bagging")){
    pred <- predict.bagging(model, PD_imputed.test)$class
  } else if (inherits(model, "boosting")){
    pred <- predict(model, PD_imputed.test)$class
  } else if (inherits(model, "randomForest")){
    pred <- predict(model, PD_imputed.test, type = "class")
  }

  confusion_matrix = table(Predicted_Class = pred, Actual_Class = PD_imputed.test$Class)
  accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
  print(confusion_matrix)
```

```
  print(accuracy)
  return (accuracy)
}

accuracy_decisionTree <- prediction_func(PD.decisionTree)
accuracy_naiveBayes <- prediction_func(PD.nBayes)
accuracy_bagging <- prediction_func(PD.bagging)
accuracy_boosting <- prediction_func(PD.boosting)
accuracy_randomForest <- prediction_func(PD.randomForest)

#______________________________________________________________________________#
# Question 6

roc_func <- function(model, colour, label, bool){

  if (inherits(model, "tree")){
    predd <- predict(model, PD_imputed.test, type = "vector")
    PDpred <- prediction(predd[,2], PD_imputed.test$Class)

  } else if (inherits(model, "naiveBayes")){
    predd <- predict(model, PD_imputed.test, type = "raw")
    PDpred <- prediction(predd[,2], PD_imputed.test$Class)

  } else if (inherits(model, "bagging")){
    PDpred <- prediction(predict(model, PD_imputed.test, type = "prob")$votes[,2],
PD_imputed.test$Class)

  } else if (inherits(model, "boosting")){
    PDpred <- prediction(predict(model, PD_imputed.test)$prob[,2], PD_imputed.test$Class)

  } else if (inherits(model, "randomForest")){
    predd <- predict(model, PD_imputed.test, type = "prob")
    PDpred <- prediction(predd[,2], PD_imputed.test$Class)

  }

  perf <- performance(PDpred, "tpr", "fpr")
  plot(perf, col = colour, main = "ROC Curve for Different Classifiers",
       xlab = "False Positive Rate (1 - Specificity)",
       ylab = "True Positive Rate (Sensitivity)", add = bool)

  auc <- performance(PDpred, "auc")@y.values[[1]]
  return(auc)
}

plot.new()
auc_decisionTree <- roc_func(PD.decisionTree, "red", "Decision Tree", FALSE)
auc_naiveBayes <- roc_func(PD.nBayes, "blue", "Naive Bayes", TRUE)
auc_bagging <- roc_func(PD.bagging, "green", "Bagging", TRUE)
auc_boosting <- roc_func(PD.boosting, "purple", "Boosting", TRUE)
```

```
auc_randomForest <- roc_func(PD.randomForest, "orange", "Random Forest", TRUE)

legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting",
"Random Forest"),
      col = c("red", "blue", "green", "purple", "orange"), lty = 1, cex = 0.8)

auc_decisionTree
auc_naiveBayes
auc_bagging
auc_boosting
auc_randomForest

#____________________________________________________________________________#
# Question 7

classifier_table <- data.frame(
  Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
  Accuracy = c(accuracy_decisionTree, accuracy_naiveBayes, accuracy_bagging,
accuracy_boosting, accuracy_randomForest),
  AUC = c(auc_decisionTree, auc_naiveBayes, auc_bagging, auc_boosting,
auc_randomForest)
)

classifier_table

#____________________________________________________________________________#
# Question 8

imptVar_decisionTree <- summary(PD.decisionTree)
imptVar_decisionTree

imptVar_bagging <- sort(PD.bagging$importance, decreasing = TRUE)
imptVar_bagging

imptVar_boosting <- sort(PD.boosting$importance, decreasing = TRUE)
imptVar_boosting

imptVar_randomForest <- PD.randomForest$importance[order(-
PD.randomForest$importance),]
imptVar_randomForest

#____________________________________________________________________________#
# Question 9

PD.simpleTree <- tree(Class~., data = PD_imputed.train)
summary(PD.simpleTree)

cv.simpleTree <- cv.tree(PD.simpleTree, FUN = prune.misclass)
plot(cv.simpleTree$size, cv.simpleTree$dev, type="b", xlab="Tree Size", ylab="Deviance")
```

```
optimal_size <- which.min(cv.simpleTree$dev)
PD.simpleTree_pruned <- prune.misclass(PD.simpleTree,
best=cv.simpleTree$size[optimal_size])
plot(PD.simpleTree_pruned)
text(PD.simpleTree_pruned, pretty = 1)
summary(PD.simpleTree_pruned)

# confusion matrix and accuracy
PD.pred.simpleTree_pruned <- predict(PD.simpleTree_pruned, PD_imputed.test, type =
'class')
cm_simpleTree_pruned <- table(Actual=PD_imputed.test$Class,
Predicted=PD.pred.simpleTree_pruned)
cm_simpleTree_pruned
accuracy_simpleTree_pruned <- sum(diag(cm_simpleTree_pruned)) /
sum(cm_simpleTree_pruned)
accuracy_simpleTree_pruned

# AUC for pruned tree
PD.pred_simpleTree_prob <- predict(PD.simpleTree_pruned, PD_imputed.test, type =
'vector')
PD.pred_simpleTree_AUC <- prediction(PD.pred_simpleTree_prob[,2],
PD_imputed.test$Class)
auc_simpleTree_pruned <- performance(PD.pred_simpleTree_AUC, "auc")
auc_simpleTree_pruned <- as.numeric(auc_simpleTree_pruned@y.values)
auc_simpleTree_pruned

#______________________________________________________________________________#
# Question 10

set.seed(32909764)
tuneGrid <- expand.grid(
  mtry = c(2, 3, 4, 5) # Number of variables randomly sampled as candidates at each split
)
trainControl <- trainControl(method = "cv", number = 5)

rf_tuned <- train(
  Class ~ ., data = PD_imputed.train, method = "rf",
  tuneGrid = tuneGrid,
  trControl = trainControl,
  ntree = 500, # Number of trees in the forest
  nodesize = 5 # Minimum size of terminal nodes
)

best_model <- rf_tuned$finalModel
rf_tuned$bestTune

# Predict on the test set
pred_probs <- predict(best_model, PD_imputed.test, type = "prob")
pred_classes <- predict(best_model, PD_imputed.test)
```

```
# Create a confusion matrix
cm_improvedRF <- table(Actual = PD_imputed.test$Class, Predicted = pred_classes)

# Calculate accuracy
accuracy_improvedRF <- sum(diag(cm_improvedRF)) / sum(cm_improvedRF)
accuracy_improvedRF

# Calculate AUC
pred <- prediction(pred_probs[,2], PD_imputed.test$Class)
perf <- performance(pred, "tpr", "fpr")
auc_improvedRF <- performance(pred, "auc")
auc_improvedRF <- round(as.numeric(auc_improvedRF@y.values), 4)
auc_improvedRF

#______________________________________________________________________________#
# Question 11

PD_imputed.train$Class <- as.numeric(recode(PD_imputed.train$Class, "1" = 1, "0" = 0))
PD_imputed.test$Class <- as.numeric(recode(PD_imputed.test$Class, "1" = 1, "0" = 0))

# check if there's any NA
anyNA(PD_imputed.train)
anyNA(PD_imputed.test)
PD_imputed.test <- na.omit(PD_imputed.test)
PD_imputed.train <- na.omit(PD_imputed.train)

# Train the neural network
PD.nn <- neuralnet(Class ~ A01 + A18 + A23 + A22, data=PD_imputed.train, hidden=3,
           linear.output = TRUE)

plot(PD.nn)

PD.nn_predictions <- compute(PD.nn, PD_imputed.test)
PD.nn_predr <- as.data.frame(round(PD.nn_predictions$net.result, 0))

cm_PD.nn <- table(observed = PD_imputed.test$Class, predicted = PD.nn_predr$V1)
cm_PD.nn

accuracy_PD.nn <- sum(diag(cm_PD.nn)) / sum(cm_PD.nn)
accuracy_PD.nn

PD.nn_probs <- PD.nn_predictions$net.result
PD.nn_pred <- prediction(PD.nn_probs, PD_imputed.test$Class)
PD.nn_perf <- performance(PD.nn_pred, "tpr", "fpr")
auc_nn <- performance(PD.nn_pred, "auc")
auc_nn <- round(as.numeric(auc_nn@y.values), 4)
auc_nn

#______________________________________________________________________________#
# Question 12
```

```
# handle missing values by median imputation
PD_imputed <- PD %>%
  mutate(across(where(is.numeric), ~ifelse(is.na(.), median(., na.rm = TRUE), .)))

# convert Class variable to factor
PD_imputed$Class <- factor(PD_imputed$Class)
str(PD_imputed$Class)

set.seed(32909764) #Student ID as random seed
train.row = sample(1:nrow(PD_imputed), 0.7*nrow(PD_imputed))
PD_imputed.train = PD_imputed[train.row,]
PD_imputed.test = PD_imputed[-train.row,]

PD_imputed.train$Class <- as.numeric(as.character(PD_imputed.train$Class))
PD_imputed.test$Class <- as.numeric(as.character(PD_imputed.test$Class))

scale_data <- function(df) {
  num_cols <- sapply(df, is.numeric)
  df[num_cols] <- scale(df[num_cols])
  return(df)
}

PD_imputed.train <- scale_data(PD_imputed.train)
PD_imputed.test <- scale_data(PD_imputed.test)

remove_all_na_columns <- function(train_data, test_data) {
  # Get column indices with all NaN values in test data
  na_cols_test <- which(colSums(is.na(test_data)) == nrow(test_data))

  if (length(na_cols_test) > 0) {
    # Remove columns with all NaN values from test data
    test_data <- test_data[, -na_cols_test]
    cat("Removed columns from test data:", paste(names(test_data)[na_cols_test]
                                              , collapse = ", "), "\n")

    # Remove the same columns from train data
    train_data <- train_data[, -na_cols_test]
    cat("Removed columns from train data:", paste(names(train_data)[na_cols_test]
                                               , collapse = ", "), "\n")
  } else {
    cat("No columns with all NaN values found in test data.\n")
  }

  return(list(train_data = train_data, test_data = test_data))
}

# Remove columns with all NaN values from training and test datasets
result <- remove_all_na_columns(PD_imputed.train, PD_imputed.test)
PD_imputed.train <- result$train_data
```

```
PD_imputed.test <- result$test_data

# Function to train and evaluate SVM with different kernels
evaluate_svm_kernel <- function(train_data, test_data, kernel_type) {
  # Train SVM model with specified kernel
  svm_model <- svm(Class ~ ., data = train_data, kernel = kernel_type, probability = TRUE)

  # Make predictions
  svm_predictions <- predict(svm_model, test_data, probability = TRUE)
  svm_probabilities <- attr(svm_predictions, "probabilities")[,2]

  # Calculate confusion matrix
  cm_svm <- table(Actual = test_data$Class, Predicted = as.numeric(svm_predictions > 0.5))

  # Calculate accuracy
  accuracy_svm <- sum(diag(cm_svm)) / sum(cm_svm)

  return(list(accuracy = accuracy_svm, cm = cm_svm))
}

# List of kernels to evaluate
kernels <- c("linear", "polynomial", "radial", "sigmoid")

# Store results
results <- list()

# Evaluate each kernel
for (kernel in kernels) {
  cat("Evaluating kernel:", kernel, "\n")
  results[[kernel]] <- evaluate_svm_kernel(PD_imputed.train, PD_imputed.test, kernel)
}

# Print results
for (kernel in kernels) {
  cat("\nKernel:", kernel, "\n")
  cat("Accuracy:", results[[kernel]]$accuracy, "\n")
  print(results[[kernel]]$cm)
}
```