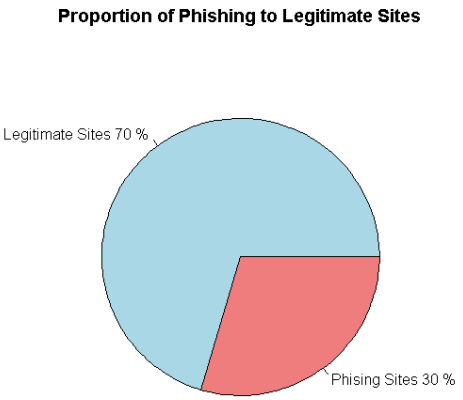
Phishing Website

Data Analytics

Fullname: Hansel Liebrata

Data Exploration

The dataset consists of 26 columns and 2000 rows, with all data types represented by integers and numbers. After sampling, the dataset contains 1407 legitimate sites and 593 phishing sites, indicating a proportion of phishing sites to legitimate sites at approximately 29.7%. Since we cannot really know if the attributes are real-valued attributes I did the descriptive analysis for all the columns except class. The descriptive statistics reveal interesting insights into the predictor variables (A01 to A25). Attributes such as A12 exhibit high variability, with maximum values reaching up to 692 and a mean of 317.68, suggesting the presence of outliers. In contrast, attributes like A13, A16, and A21 show low variability with minimal means which indicate that the variable does not vary significantly.



Several attributes, including A02, A19, A22, A23, A24, and A25, have a high number of missing values that might need to be removed to ensure the accuracy of the model. There aren't any attributes that are not included in the analysis.However some of the row containing the missing or null will be removed in the data manipulation.

	Min	X1st.Quartile	Median	Mean	X3rd.Quartile	Max	Std	num_na
A01	3.00000000	5.00000000	15.00000000	2.063250e+01	33.00000000	4.700000e+01	1.518291e+01	0
A02	0.00000000	0.00000000	0.00000000	2.018256e-01	0.00000000	6.000000e+01	2.046848e+00	28
A03	0.00000000	0.00000000	0.00000000	1.010611e-03	0.00000000	1.000000e+00	3.178208e-02	21
A04	2.00000000	2.00000000	3.00000000	2.781754e+00	3.00000000	6.000000e+00	5.419839e-01	16
A05	0.00000000	0.00000000	0.00000000	1.365706e-02	0.00000000	1.200000e+01	3.154517e-01	23
A06	0.00000000	0.00000000	0.00000000	1.334681e-01	0.00000000	1.000000e+00	3.401660e-01	22
A07	0.00000000	0.00000000	0.00000000	2.530364e-03	0.00000000	1.000000e+00	5.025176e-02	24
A08	0.17391304	0.68000000	1.00000000	8.517092e-01	1.00000000	1.000000e+00	2.119285e-01	18
A09	0.00000000	0.00000000	0.00000000	2.225594e-02	0.00000000	1.000000e+00	1.475521e-01	23
A10	0.00000000	0.00000000	0.00000000	3.585859e-02	0.00000000	1.000000e+00	1.859845e-01	20
A11	0.00000000	0.00000000	0.00000000	7.777778e-02	0.00000000	3.100000e+01	8.706146e-01	20
A12	3.00000000	232.00000000	232.00000000	3.176813e+02	427.00000000	6.920000e+02	1.431821e+02	23
A13	0.00000000	0.00000000	0.00000000	9.095503e-03	0.00000000	1.200000e+01	2.860387e-01	21
A14	0.00000000	0.00000000	0.00000000	1.160444e-01	0.00000000	1.000000e+00	3.203590e-01	18
A15	0.00000000	0.00000000	0.00000000	1.304129e-01	0.00000000	1.000000e+00	3.368420e-01	14
A16	0.00000000	0.00000000	0.00000000	4.246714e-02	0.00000000	1.000000e+00	2.017034e-01	22
A17	0.00000000	1.00000000	1.00000000	1.166751e+00	1.00000000	5.000000e+00	6.235082e-01	21
A18	4.00000000	13.00000000	32.00000000	5.799544e+01	89.00000000	1.825000e+03	8.785023e+01	25
A19	0.00000000	0.00000000	0.00000000	9.868421e-02	0.00000000	1.000000e+00	2.983130e-01	24
A20	0.00000000	0.00000000	0.00000000	2.464503e-01	0.00000000	1.000000e+00	4.310531e-01	28
A21	0.00000000	0.00000000	0.00000000	3.388973e-02	0.00000000	3.000000e+00	2.095027e-01	23
A22	0.01100637	0.05111965	0.05808033	5.591690e-02	0.06291761	8.876583e-02	1.049902e-02	16
A23	0.00000000	18.00000000	100.00000000	7.783005e+01	108.00000000	1.700000e+03	7.167932e+01	23
A24	0.00000000	0.00820030	0.52290710	2.757147e-01	0.52290710	5.229071e-01	2.518872e-01	21
A25	0.00000000	0.00000000	0.00000000	1.105502e-04	0.00000000	7.700000e-02	2.398104e-03	19

Pre-processing

To make the dataset suitable for the model fitting, several pre-processing steps were undertaken. First of all, I removed all rows containing NA Value to prevent any error from the model fitting process. Which resulted in a total of 513 rows being removed from the sampled data. Moreover, the class attribute which when 1 indicates a phishing site and 0 indicates a legitimate site was converted into a factor to ensure proper handling by the modelling algorithm. The pre-processing is essential to ensure data integrity and enhance the model's performance. Then the data is splitted to 70% as the training set and 30% as the test set. The training set is used to create the model and test set to check the accuracy and performance of the model.

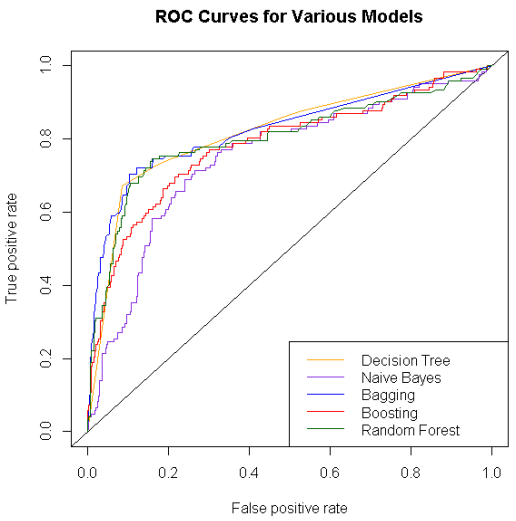
Confusion Matrix & Accuracy of Models

I have implemented 5 different classification models to predict whether a site is legitimate. The models are Decision Tree, Naive Bayes, Bagging, Boosting and Random Forest. After testing all the models on the test set and creating confusion matrices for each shown in Appendix 1.0, the accuracy of each model was evaluated. The Decision Tree model achieved the highest accuracy at 85.06%, demonstrating its effectiveness in classifying the sites correctly. Bagging with second highest accuracy of 84.20%.Followed by Bagging with an accuracy of 84.20%, showing a potential in its model. The random forest is another well performing model with an accuracy of 82.29%. While boosting has a lower accuracy it still shows a good model with an accuracy of 79.43%. The Naive Bayes model, with an accuracy of 33.12%, had the lowest performance, suggesting that the model doesn't suit well for this dataset. These results highlight the superior performance towards Decision Tree, Bagging and Random Forest, in predicting the legitimacy of sites.

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest
Accuracy	0.8506494	0.3311688	0.8419913	0.7943723	0.8290043

ROC Curve & AUC of Models

Using the model and testing it on the test data, I computed the confidence in predicting phishing for each model. Then, the ROC curves for all the models were plotted on the same axis. Each model is represented by a different colour to help users differentiate between the lines. The decision tree is plotted using a yellow line, Naive Bayes using purple, bagging using blue, boosting using red, and random forest using green. The plot allows for easy comparison of the ROC of each model. However, in some cases such as decision trees, random forests, and bagging, it is difficult to compare since the values are similar.



Therefore, I decided to calculate the AUC. The bagging model achieved the highest AUC of 0.8254, indicating the best performance in distinguishing between phishing and legitimate sites. The decision tree model followed closely with an AUC of 0.8194, and the random forest model also performed well with an AUC of 0.8031. The boosting model had a moderate AUC of 0.7837, while the Naive Bayes model had the lowest AUC of 0.7475.

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest
ROC AUC	0.8190453	0.7474807	0.8253616	0.7836789	0.8031220

Model Comparison and Best Classifier

Based on both the accuracy and ROC AUC of the models it seems that Decision Tree and bagging perform the best. The Decision Tree classifier shows the highest accuracy of 85.06% and an AUC-ROC of 81.19%, while Bagging has the highest ROC AUC of 82.25% with 84.20%. The value of the accuracy and AUC of both models are very similar. Since they are similar, choosing the best classifier depends if ROC or accuracy is needed. Higher AUC-ROC model performs better in imbalanced data, while accuracy performs better in balanced dataset. (Singh, 2023). According to Singh (2023) AUC generally indicates how good the model is. Based on that statement I believe that the Bagging model is the best based on the training model.

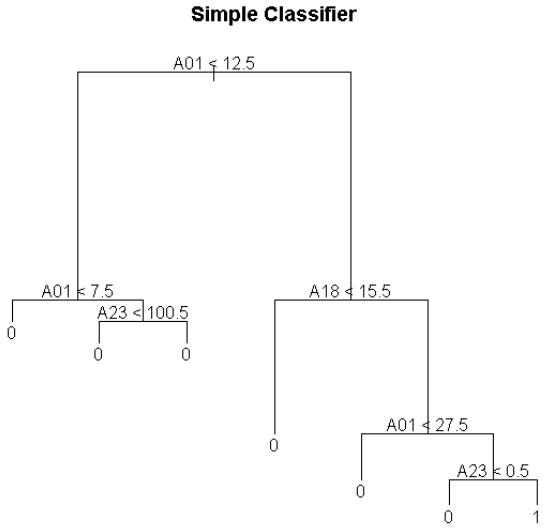
	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest
Accuracy	0.8506494	0.3311688	0.8419913	0.7943723	0.8290043
ROC AUC	0.8190453	0.7474807	0.8253616	0.7836789	0.8031220

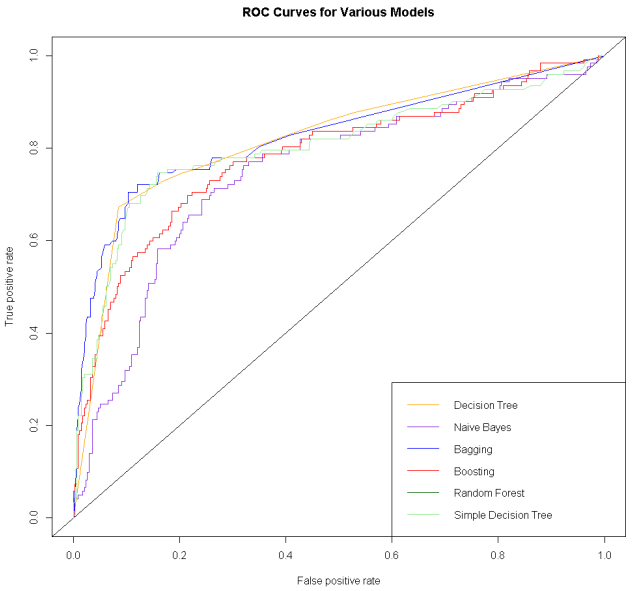
Variable Importance

As shown in Image 2.1 Variable Importance in the Appendix it highlights the key variables used in building each model. Based on the result I have determined that A01, A18, and A23 are the most important variables as they are utilised by all the models. From Image 2.1 for the Decision Tree model the variables A01, A18, A23, and A08 are variables that are used. Analysing the importance variable from other models we can see that A01, A18, and A23 are the variables that are consistently important in generating these models. Conversely, variables A03, A05, A07, A11, A13, and A21 could be omitted with very little effect on the performance of the model. The value of importance is constantly lower than 1 and most of the time their importance is 0. Removing these variables will have a negligible effect on the model's performance.

Model Simplification

The most important factor used in building the simple model is choosing the right features. From the previous analysis on the importance variable it shows that A01, A18 and A23 are the most important predictors. Therefore, I use those variables in building the model using a decision tree. The selected attributes chosen are the attributes that likely have strong correlation with the class attributes that can help separate data into different classes effectively. As we want a simple model choosing a minimal amount of variables is preferred.



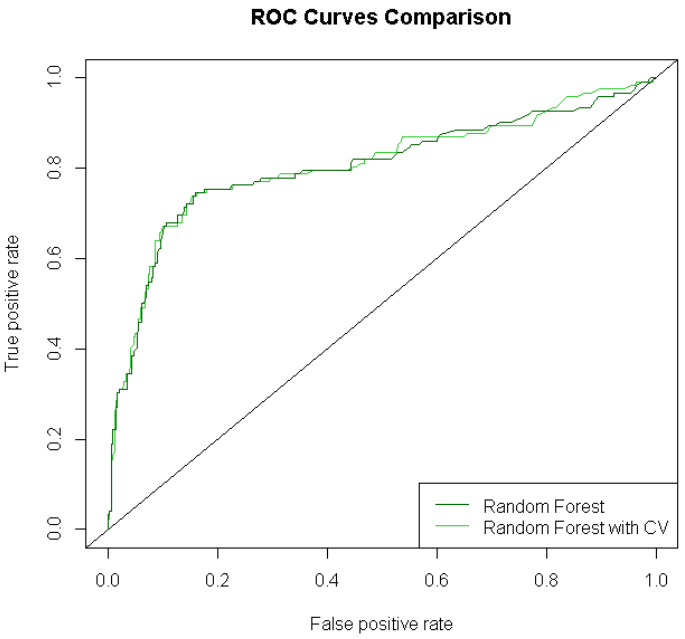


I have added the simplified decision tree curve to the ROC chart. We can see that the ROC of the model shows a good performance of the simplified decision tree even though it only uses a minimal amount of predictors. The simplified model demonstrates a high accuracy of 85.06% and an AUC of 81.84%, showcasing its effectiveness. In terms of AUC it outperformed Naive Bayes, Boosting and Random Forest models. For its accuracy it matches with the full decision tree model with highest accuracy compared to the other. This indicates that a simple model using key attributes can achieve similar or even superior performance compared to more complex models.

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest	Simple Classifier
Accuracy	0.8506494	0.3311688	0.8419913	0.7943723	0.8290043	0.8506494
ROC AUC	0.8190453	0.7474807	0.8253616	0.7836789	0.8031220	0.8183582

Optimising Tree-Based Classifier

Initially I decided to apply cross validation towards the decision tree model. However, The decision tree has already used 4 variables and reducing the variables into 3 will be the previously simple created model. Moreover, cross validation on the decision tree does not affect the accuracy and ROC of the model. Consequently, I chose to focus on the Random Forest model. The Random Forest model is similar to bagging but reduces overfitting due to random feature selection. This inherent feature of Random Forest makes it more robust and improves generalisation performance.



In creating the best tree based classifier I did not prune or select the attributes manually to utilise all available features. This allows Random Forest's capability to automatically assess and prioritise features during training. For the cross validation choosing 5 folds for the cross validation is the best for this scenario for this data as it has lots of data entries and it provides a higher accuracy and ROC. The chart shown on the side is the ROC curve comparison between random forest and the optimal random forest. It's very similar as the difference between the two models is around 0.003. The random forest with cross validation shows an improvement in both accuracy and ROC compared to the model without cross validation. From the table, the tuned Random Forest model had the same accuracy as the Bagging model and was the second highest overall. Its ROC AUC was lower than the Decision Tree and Bagging models but showed improvement due to cross validation.

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest	Tuned Random Forest
Accuracy	0.8506494	0.3311688	0.8419913	0.7943723	0.8290043	0.8419913
ROC AUC	0.8190453	0.7474807	0.8253616	0.7836789	0.8031220	0.8062801

Implementation of Neural Network

To implement a neural network model for this dataset choosing the right predictor is crucial in producing the best model. Based on previous analyses, A01, A18, and A23 have been identified as the most important predictors and will be used here. Additionally, A22 and A08 have been included based on the most important predictor in the Tuned Random Forest shown in Appendix 2.2. Manual selection on the number of predictors and choosing these five attributes shows the highest accuracy and ROC with three hidden layers. To create the neural network model additional data preprocessing is required. The selected attributes should be converted into numeric form, and normalisation, while optional, can improve training efficiency.

The Neural Network classifier shows the best ROC AUC of 84.32% and a high accuracy of 83.77%. Although the accuracy is slightly lower than that of the Decision Tree and Bagging models, the AUC is the highest among all models. This higher performance in AUC can be attributed to the ANN's ability to capture complex, non-linear relationships between the data points. Moreover, ANN can handle large volumes of data and learn patterns better which might be missed by simpler models. Which further increases its effectiveness in identifying phishing sites.

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest	Neural Network
Accuracy	0.8506494	0.3311688	0.8419913	0.7943723	0.8290043	0.8376623
ROC AUC	0.8190453	0.7474807	0.8253616	0.7836789	0.8031220	0.8432498

Exploration of a New Classifier

The new classifier that I am going to test out is the gradient boosting algorithm from XGBoost library. The algorithm is a boosting algorithm but the difference is that it implements a machine learning algorithm that when each tree helps in correcting errors made by previously trained trees. This process continues until no further improvement can be made.(Wohlwend, 2023) To use this library, the data must be preprocessed by converting all attributes to numeric format. For this model, all provided attributes were used. The model achieved an accuracy of 80.52%, which is high but lower than some simpler models like the Decision Tree. However, it shows an improvement over the basic boosting algorithm. The ROC AUC of the Gradient Boosting model, while not the highest compared to other models, is still an improvement over standard boosting, indicating better performance in distinguishing between phishing and legitimate sites.

	Decision Tree	Naive Bayes	Bagging	Boosting	Random Forest	Gradient Boosting
Accuracy	0.8506494	0.3311688	0.8419913	0.7943723	0.8290043	0.8051948
ROC AUC	0.8190453	0.7474807	0.8253616	0.7836789	0.8031220	0.7862584

Reference

Singh, V. (2023, October 17). *ROC-AUC vs Accuracy: Which Metric Is More Important? - Shiksha Online*. Shiksha.com. https://www.shiksha.com/online-courses/articles/roc-auc-vs-accuracy/#:~:text=Accuracy%20is%20one%20of%20the%20most%20used%20and%20easy%20to,Accuracy%20metrics%20perform%20very%20well.&text=AUC%20measures%20the%20model%20sensitivity,does%20not%20distinguish%20between%20these.

Wohlwend, B. (2023, August 12). Decision Tree, Random Forest, and XGBoost: An Exploration into the Heart of Machine Learning. *Medium*. https://medium.com/@brandon93.w/decision-tree-random-forest-and-xgboost-an-exploration-into-the-heart-of-machine-learning-90dc212f4948

XGBoost Documentation — xgboost 2.0.3 documentation. (n.d.). https://xgboost.readthedocs.io/en/stable/

Appendix

1.0 Confusion Matrix

1.1 Decision Tree Confusion Matrix

```
> print(TreeMatrix)
                Actual_Class
Predicted_Class   0   1
              0 311  40
              1  29  82
> accuracy_tree <- sum(diag(as.matrix(TreeMatrix)))/ nrow(PD.test) #Accuracy Decision Tree
> accuracy_tree
[1] 0.8506494
```

1.1 Naive Bayes Confusion Matrix

```
> NaiveMatrix =table(Predicted_Class = PD.predbayes, Actual_Class = PD.test$Class)
> print(NaiveMatrix)
                Actual_Class
Predicted_Class   0   1
              0  37   6
              1 303 116
> accuracy_naive <- sum(diag(as.matrix(NaiveMatrix)))/ nrow(PD.test) #Accuracy Naive Matrix
> print(accuracy_naive)
[1] 0.3311688
```

1.1 Bagging Confusion Matrix

```
> print(PDpred.bagging$confusion)
                Observed Class
Predicted Class   0   1
              0 310  43
              1  30  79
> accuracy_bagging <- sum(diag(as.matrix(PDpred.bagging$confusion)))/ nrow(PD.test) #Accuracy Bagging
> accuracy_bagging
[1] 0.8419913
```

1.1 Boosting Confusion Matrix

```
> #Boosting
> print(PDpred.boost$confusion)
                Observed Class
Predicted Class   0   1
              0 297  52
              1  43  70
> accuracy_boosting <- sum(diag(as.matrix(PDpred.boost$confusion)))/ nrow(PD.test) #Accuracy Boosting
> accuracy_boosting
[1] 0.7943723
```

1.1 Random Forest Confusion Matrix

```
> ForestMatrix = table(Predicted_Class = PDpredrf, Actual_Class = PD.test$Class)
> print(ForestMatrix)
                Actual_Class
Predicted_Class   0   1
              0 309  48
              1  31  74
> accuracy_forest <- sum(diag(as.matrix(ForestMatrix)))/ nrow(PD.test) #Accuracy Random Forest
> accuracy_forest
[1] 0.8290043
```

2.1 Image Variable Importance Simple Model

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

Classification tree:
tree(formula = Class ~ ., data = PD.train)
Variables actually used in tree construction:
[1] "A01" "A23" "A18" "A08"
Number of terminal nodes:  8
Residual mean deviance:  0.904 = 966.3 / 1069
Misclassification error rate: 0.2033 = 219 / 1077
> cat("\n#Baging Attribute Importance\n")
 #Baging Attribute Importance
 > print(PD.bagging$importance)
        A01        A02        A03        A04        A05        A06        A07        A08        A09        A10        A11        A12        A13        A14
42.80632159  0.17386692  0.00000000  0.33543759  0.00000000  0.19841052  0.00000000  2.53798130  0.05215108  0.07811910  0.00000000  1.96080667  0.00000000  2.35707135
        A15        A16        A17        A18        A19        A20        A21        A22        A23        A24        A25
 0.21963673  0.02055811  0.36266054 28.66627496  0.07492085  0.41700923  0.00000000 10.12219556  8.18525307  1.43132484  0.00000000
 > cat("\n#Boosting Attribute Importance\n")

 #Boosting Attribute Importance
 > print(PD.boost$importance)
        A01        A02        A03        A04        A05        A06        A07        A08        A09        A10        A11        A12        A13        A14        A15
11.1246202  0.9273380  0.0000000  0.3388440  0.1470190  0.7932731  0.0000000  7.8695161  0.1191643  0.4456292  0.0000000  6.5913538  0.0000000  1.0729728  1.0611107
        A16        A17        A18        A19        A20        A21        A22        A23        A24        A25
 0.2507878  2.3344764 18.0321552  0.9209666  1.2411773  0.2695170 27.7008013 13.5285782  5.2306989  0.0000000
 > cat("\n#Random Forest Attribute Importance\n")

 #Random Forest Attribute Importance
 > print(PD.rf$importance)
     MeanDecreaseGini
A01      93.904002531
A02       5.595847915
A03       0.001463655
A04       6.445922220
A05       0.722980461
A06       5.539918065
A07       0.327836634
A08      24.602253483
A09       1.855320582
A10       2.373014890
A11       1.013840912
A12      20.663500875
A13       0.353683707
A14      14.760034457
A15       5.233255729
A16       2.549651161
A17       9.829194313
A18      64.367840289
A19       4.384863454
A20       6.803053484
A21       0.894771607
A22      57.134417206
A23      56.882319964
A24      21.119873712
A25       0.221450243
```

2.2 Image Variable Importance Tuned Random Forest

```
> print(importance)
rf variable importance

  only 20 most important variables shown (out of 25)

    Overall
A01 94.5686
A18 64.4358
A22 59.1865
A23 56.1568
A08 24.1847
A24 20.8670
A12 20.3650
A14 14.9075
A17 10.0873
A20  6.3607
A04  6.2592
A15  5.4682
A02  5.3590
A06  5.2180
A19  4.6438
A16  2.6242
A10  2.4262
A09  1.9294
A21  0.9356
A11  0.9096
```

R-Code

```
library(tree)
library(e1071)
library(ROCR)
library(randomForest)
library(adabag)
library(rpart)
library(dplyr)

rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(32578342) # 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

#1
str(PD)
ncol(PD)
nrow(PD)

phishing_proportion <- table(PD$Class)
phishing_proportion

labels <- c("Legitimate Sites", "Phising Sites")
percentages <- round(phishing_proportion / sum(phishing_proportion) * 100)
percent_labels <- paste(labels, percentages, "%", sep=" ")
pie_positions <- pie(phishing_proportion, labels = percent_labels, col = c("lightblue",
"lightcoral"), main = "Proportion of Phishing to Legitimate Sites")

descriptions <- data.frame(
  Min = sapply(PD[, !names(PD) %in% "Class"], min, na.rm = TRUE),
  `1st Quartile` = sapply(PD[, !names(PD) %in% "Class"], function(x) quantile(x, 0.25, na.rm
= TRUE)),
  Median = sapply(PD[, !names(PD) %in% "Class"], median, na.rm = TRUE),
  Mean = sapply(PD[, !names(PD) %in% "Class"], mean, na.rm = TRUE),
  `3rd Quartile` = sapply(PD[, !names(PD) %in% "Class"], function(x) quantile(x, 0.75, na.rm
= TRUE)),
  Max = sapply(PD[, !names(PD) %in% "Class"], max, na.rm = TRUE),
  Std = sapply(PD[, !names(PD) %in% "Class"], sd, na.rm = TRUE),
  num_na = sapply(PD[, !names(PD) %in% "Class"], function(x) sum(is.na(x)))
)
print(descriptions)

#2
```

```
sum(is.na(PD))
PD <- na.omit(PD)
nrow(PD)
PD$Class <- as.factor(PD$Class)


#3
set.seed(32578342) #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,]

#4
#Decision Tree
PD.tree = tree(Class ~.,data = PD.train)
#Naive Bayes
PD.bayes = naiveBayes(Class ~. , data = PD.train)
PD.predbayes = predict(PD.bayes, PD.test)
#Bagging
PD.bagging <- bagging(Class ~. , data = PD.train)
PDpred.bagging <- predict.bagging(PD.bagging, PD.test)
#Boosting
PD.boost <- boosting(Class ~. , data = PD.train)
PDpred.boost <- predict.boosting(PD.boost, PD.test)
#Random Forest
PD.rf <- randomForest(Class ~. , data = PD.train, na.action = na.exclude)
PDpredrf <- predict(PD.rf, PD.test)


#5
#Decision Tree
PD.predtree = predict(PD.tree, PD.test, type = "class")
TreeMatrix = table(Predicted_Class = PD.predtree, Actual_Class = PD.test$Class)
print(TreeMatrix)

accuracy_tree <- sum(diag(as.matrix(TreeMatrix)))/ nrow(PD.test) #Accuracy Decision Tree
accuracy_tree
#Naive Bayes
NaiveMatrix =table(Predicted_Class = PD.predbayes, Actual_Class = PD.test$Class)
print(NaiveMatrix)

accuracy_naive <- sum(diag(as.matrix(NaiveMatrix)))/ nrow(PD.test) #Accuracy Naive Matrix
print(accuracy_naive)
#Bagging
print(PDpred.bagging$confusion)

accuracy_bagging <- sum(diag(as.matrix(PDpred.bagging$confusion)))/ nrow(PD.test)
#Accuracy Bagging
```

```
accuracy_bagging
#Boosting
print(PDpred.boost$confusion)

accuracy_boosting <- sum(diag(as.matrix(PDpred.boost$confusion)))/ nrow(PD.test)
#Accuracy Boosting
accuracy_boosting
#Random Forest
ForestMatrix = table(Predicted_Class = PDpredrf, Actual_Class = PD.test$Class)
print(ForestMatrix)
accuracy_forest <- sum(diag(as.matrix(ForestMatrix)))/ nrow(PD.test) #Accuracy Random Forest
accuracy_forest

#6
# 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, col = "orange", main = "ROC Curves for Various Models")
abline(0,1)

# 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")

# Bagging
PDBagpred <- ROCR::prediction(PDpred.bagging$prob[,2], PD.test$Class)
PDBagperf <- performance(PDBagpred, "tpr", "fpr")
plot(PDBagperf, add = TRUE, col = 'blue')

# Boosting
PDBoostpred <- ROCR::prediction(PDpred.boost$prob[,2], PD.test$Class)
PDBoostperf <- performance(PDBoostpred, "tpr", "fpr")
plot(PDBoostperf, add = TRUE, col= "red")

# 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")

# Add legend
legend("bottomright",
       legend = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
       col = c("orange", "blueviolet", "blue", "red", "darkgreen"),
```

```
      lty = 1)

#7
roc_tree <- performance(PDDpred, measure = "auc")@y.values[[1]]
roc_naive <- performance(PDBpred, measure = "auc")@y.values[[1]]
roc_bagging <- performance(PDBagpred, measure = "auc")@y.values[[1]]
roc_boosting <- performance(PDBoostpred, measure = "auc")@y.values[[1]]
roc_forest <- performance(PDFpred, measure = "auc")@y.values[[1]]

results_table <- matrix(c(
  accuracy_tree, accuracy_naive, accuracy_bagging, accuracy_boosting, accuracy_forest,
  roc_tree, roc_naive, roc_bagging, roc_boosting, roc_forest),
  nrow = 2, byrow = TRUE
)
rownames(results_table) <- c("Accuracy", "ROC AUC")
colnames(results_table) <- c("Decision Tree", "Naive Bayes", "Bagging", "Boosting",
"Random Forest")
results_table

#8
#Attribute importance
cat("\n#Decision Tree Attribute Importance\n")
print(summary(PD.tree))
cat("\n#Baging Attribute Importance\n")
print(PD.bagging$importance)
cat("\n#Boosting Attribute Importance\n")
print(PD.boost$importance)
cat("\n#Random Forest Attribute Importance\n")
print(PD.rf$importance)

#9
#A01, A18, A23
simple.model = tree(Class ~ A01 + A18 + A23 ,data = PD.train)
plot(simple.model, main = "")
text(simple.model, pretty=0)
title("Simple Classifier")

simple.predtree = predict(simple.model, PD.test, type = "class")
SimpleMatrix = table(Predicted_Class = simple.predtree, Actual_Class = PD.test$Class)
print(SimpleMatrix)

accuracy_simple <- sum(diag(as.matrix(SimpleMatrix)))/ nrow(PD.test) #Accuracy Decision
Tree
accuracy_simple

simple.pred.tree <- predict(simple.model, PD.test, type="vector")
Simplepred <- ROCR::prediction(simple.pred.tree[,2], PD.test$Class)
SimpleFperf <- performance(PDFpred,"tpr","fpr")
```

```
plot(PDFperf, add=TRUE, col = "lightgreen")
roc_simple <- performance(Simplepred, measure = "auc")@y.values[[1]]
roc_simple

results_table <- matrix(c(
   accuracy_tree, accuracy_naive, accuracy_bagging, accuracy_boosting, accuracy_forest,
accuracy_simple,
  roc_tree, roc_naive, roc_bagging, roc_boosting, roc_forest, roc_simple),
  nrow = 2, byrow = TRUE
)
rownames(results_table) <- c("Accuracy", "ROC AUC")
colnames(results_table) <- c("Decision Tree", "Naive Bayes", "Bagging", "Boosting",
"Random Forest", "Simple Classifier")
print(results_table)

#10
library(caret)
control <- trainControl(method="cv", number=5, search="grid")
tunegrid <- expand.grid(.mtry=c(1:15))
set.seed(32578342)
rf_gridsearch <- train(Class~., data=PD.train, method="rf", tuneGrid=tunegrid,
trControl=control)
best_rf_model <- rf_gridsearch$finalModel


best_rf_pred <- predict(best_rf_model, PD.test, type="response")
best_forest_matrix <- table(Predicted_Class = best_rf_pred, Actual_Class = PD.test$Class)
print(best_forest_matrix)
accuracy_best_rf <- sum(diag(as.matrix(best_forest_matrix))) / nrow(PD.test)
print(accuracy_best_rf)

# Calculate ROC AUC
best_rf_prob <- predict(best_rf_model, PD.test, type="prob")
best_rf_pred <- ROCR::prediction(best_rf_prob[,2], PD.test$Class)
best_rf_perf <- performance(best_rf_pred, "tpr", "fpr")
plot(best_rf_perf, col = "limegreen", main = "ROC Curves Comparison")
abline(0,1)
plot(PDFperf, add=TRUE, col = "darkgreen")

# Add legend
legend("bottomright",
       legend = c("Random Forest", "Random Forest with CV"),
       col = c("darkgreen", "limegreen"),
       lty = 1)
roc_best_rf <- performance(best_rf_pred, measure = "auc")@y.values[[1]]
print(roc_best_rf)

results_table <- matrix(c(
```

```
  accuracy_tree, accuracy_naive, accuracy_bagging, accuracy_boosting, accuracy_forest, accuracy_best_rf,
  roc_tree, roc_naive, roc_bagging, roc_boosting, roc_forest, roc_best_rf),
  nrow = 2, byrow = TRUE
)
rownames(results_table) <- c("Accuracy", "ROC AUC")
colnames(results_table) <- c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest", "Tuned Random Forest")
print(results_table)

#11
library(neuralnet)
importance <- varImp(rf_gridsearch, scale = FALSE)
print(importance)

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

PDNN.train <- PD.train
PDNN.test <- PD.test
PDNN.train[] <- lapply(PDNN.train, as.numeric)
PDNN.test[] <- lapply(PDNN.test, as.numeric)

normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
PD.train.norm <- as.data.frame(lapply(PDNN.train, normalize))
PD.test.norm <- as.data.frame(lapply(PDNN.test, normalize))

predictors <- c("A01", "A18", "A23", "A22", "A08")
set.seed(32578342) #Student ID as random seed
PD.nn = neuralnet(Class ~ A01 + A18 + A23 + A22 + A08, PD.train.norm, hidden=3, linear.output = FALSE, stepmax = 1e6)
PDNN.pred = neuralnet::compute(PD.nn, PD.test.norm[,predictors])
PDNN.pred$net.result <- pmax(pmin(PDNN.pred$net.result, 1), 0)
PDNN.pred_df <- as.data.frame(PDNN.pred$net.result)
PDNN.pred_rounded <- as.data.frame(round(PDNN.pred$net.result, 0))

confusion_matrix <- table(observed = PD.test.norm$Class, predicted = PDNN.pred_rounded$V1)
print(confusion_matrix)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
accuracy

ann_pred <- ROCR::prediction(PDNN.pred_df, PD.test.norm$Class)
roc_ann <- performance(ann_pred, measure = "auc")@y.values[[1]]
roc_ann
```

```
results_table <- matrix(c(
   accuracy_tree, accuracy_naive, accuracy_bagging, accuracy_boosting, accuracy_forest,
accuracy,
  roc_tree, roc_naive, roc_bagging, roc_boosting, roc_forest, roc_ann),
  nrow = 2, byrow = TRUE
)
rownames(results_table) <- c("Accuracy", "ROC AUC")
colnames(results_table) <- c("Decision Tree", "Naive Bayes", "Bagging", "Boosting",
"Random Forest", "Neural Network")
print(results_table)

#12
library(xgboost)
PD.train[] <- lapply(PD.train, as.numeric)
PD.test[] <- lapply(PD.test, as.numeric)

train_labels <- PD.train$Class
train_matrix <- as.matrix(PD.train[, !names(PD.train) %in% "Class"])

test_labels <- PD.test$Class
test_matrix <- as.matrix(PD.test[, !names(PD.test) %in% "Class"])

dtrain <- xgb.DMatrix(data = train_matrix, label = train_labels)
dtest <- xgb.DMatrix(data = test_matrix, label = test_labels)

params <- list(
  objective = "binary:logistic",
  eval_metric = "error",
  max_depth = 6,
  eta = 0.3,
  nthread = 2
)

xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 100,
  watchlist = list(train = dtrain, eval = dtest),
  verbose = 1
)

pred <- predict(xgb_model, dtest)
pred_labels <- ifelse(pred > 0.5, 1, 0)

table(Predicted = pred_labels, Actual = test_labels)
accuracy <- sum(pred_labels == test_labels) / length(test_labels)
accuracy
```

```
pred_obj <- ROCR::prediction(pred, test_labels)
perf_obj <- performance(pred_obj, measure = "auc")
auc_value <- as.numeric(perf_obj@y.values)
auc_value

results_table <- matrix(c(
   accuracy_tree, accuracy_naive, accuracy_bagging, accuracy_boosting, accuracy_forest,
accuracy,
  roc_tree, roc_naive, roc_bagging, roc_boosting, roc_forest, auc_value),
  nrow = 2, byrow = TRUE
)
rownames(results_table) <- c("Accuracy", "ROC AUC")
colnames(results_table) <- c("Decision Tree", "Naive Bayes", "Bagging", "Boosting",
"Random Forest", "Gradient Boosting")
print(results_table)
```