

FOOKAIYAN33085625

Foo Kai Yan

2024-05-05

Student Name: Foo Kai Yan

Student ID: 33085625

Student Email: kfoo0012@student.monash.edu

Remove/Clean the environment

```
rm(list=ls())
```

Set working directory

```
setwd("C:/Monash/FIT3152")
```

Install and load the libraries used

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
##
```

```
## Attaching package: 'dplyr'
```

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

```
##
```

```
## filter, lag
```

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

```
##
```

```
## intersect, setdiff, setequal, union
```

```
## Loading required package: foreach
```

```

## Loading required package: doParallel

## Loading required package: iterators

## Loading required package: parallel

##
## Attaching package: 'adabag'

## The following object is masked from 'package:ipred':
##
##     bagging

##
## Attaching package: 'reshape'

## The following objects are masked from 'package:tidyr':
##
##     expand, smiths

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

## corplot 0.92 loaded

## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

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

## The following object is masked from 'package:ggplot2':
##
##     margin

```

Load data in PhishingData.csv

```

rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(33085625)
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

Phishing dataset dimensions after seed was set

```
dim(PD)
```

```
## [1] 2000  26
```

There is a total of 2,000 rows and 26 columns present in the dataset.

Phishing dataset column names

```
names(PD)
```

```
## [1] "A01" "A02" "A03" "A04" "A05" "A06" "A07" "A08" "A09"  
## [10] "A10" "A11" "A12" "A13" "A14" "A15" "A16" "A17" "A18"  
## [19] "A19" "A20" "A21" "A22" "A23" "A24" "A25" "Class"
```

The colnames are A01, A02, A03, A04, A05, A06, A07, A08 , A09, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19, A20, A21, A22, A23, A24, A25 and Class

Obtain the proportion of phishing sites to legitimate sites

```
as.data.frame(table(PD["Class"]))
```

```
##   Class Freq  
## 1      0 1459  
## 2      1  541
```

In binary classification, 0 is often the label for negative class and 1 for positive class. So, it can be seen from the output above that the ratio of phishing sites compared to genuine legitimate sites is observed to be 541 to 1,459 where 541 is the legitimate sites and 1,459 is the phishing sites.

Phishing dataset predictor description

```
## 'data.frame':  2000 obs. of  26 variables:  
## $ A01 : int  11 10 25 31 3 3 3 11 10 25 ...  
## $ A02 : int  0 0 0 0 0 0 1 0 0 NA ...  
## $ A03 : int  0 NA 0 0 0 0 0 0 0 0 ...  
## $ A04 : int  3 3 2 3 2 3 3 2 2 3 ...  
## $ A05 : int  0 0 0 0 0 0 0 0 0 0 ...  
## $ A06 : int  1 0 1 0 0 0 0 0 0 0 ...  
## $ A07 : int  0 0 0 0 0 0 0 0 0 0 ...  
## $ A08 : num  1 1 1 0.441 0.714 ...  
## $ A09 : int  1 0 0 0 0 0 0 0 0 0 ...  
## $ A10 : int  0 0 0 0 0 0 0 0 0 0 ...  
## $ A11 : int  0 0 0 0 0 0 0 0 0 0 ...
```

```

## $ A12 : int 232 232 365 232 648 232 232 388 648 232 ...
## $ A13 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A14 : int 0 0 1 0 0 0 0 0 0 0 ...
## $ A15 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A16 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A17 : int 1 1 1 1 2 1 1 2 2 1 ...
## $ A18 : int 96 30 89 17 42 29 39 27 46 44 ...
## $ A19 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A20 : int 1 1 1 0 0 1 0 0 1 0 ...
## $ A21 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A22 : num 0.0605 0.0617 0.0601 0.0633 0.0626 ...
## $ A23 : int 15 137 107 1 110 111 116 165 107 16 ...
## $ A24 : num 0.52291 0.52291 0.00159 0.52291 0.02856 ...
## $ A25 : num 0 0 0 0 0 0 0 0 0 0 ...
## $ Class: int 0 0 0 0 0 0 1 0 0 1 ...

```

	A01	A02	A03	A04
## Min.	: 3.00	Min. : 0.0000	Min. :0.000000	Min. :2.000
## 1st Qu.:	10.00	1st Qu.: 0.0000	1st Qu.:0.000000	1st Qu.:2.000
## Median :	15.00	Median : 0.0000	Median :0.000000	Median :3.000
## Mean :	16.72	Mean : 0.1602	Mean :0.001521	Mean :2.728
## 3rd Qu.:	25.00	3rd Qu.: 0.0000	3rd Qu.:0.000000	3rd Qu.:3.000
## Max. :	35.00	Max. :20.0000	Max. :1.000000	Max. :7.000
## NA's	:21	NA's :27	NA's :27	NA's :27

	A05	A06	A07	A08
## Min.	: 0.00000	Min. :0.0000	Min. :0.000000	Min. :0.1429
## 1st Qu.:	0.00000	1st Qu.:0.0000	1st Qu.:0.000000	1st Qu.:0.6774
## Median :	0.00000	Median :0.0000	Median :0.000000	Median :1.0000
## Mean :	0.02375	Mean :0.1351	Mean :0.001515	Mean :0.8434
## 3rd Qu.:	0.00000	3rd Qu.:0.0000	3rd Qu.:0.000000	3rd Qu.:1.0000
## Max. :	18.00000	Max. :1.0000	Max. :1.000000	Max. :1.0000
## NA's	:21	NA's :24	NA's :20	NA's :26

	A09	A10	A11	A12
## Min.	:0.00000	Min. :0.00000	Min. :0.00000	Min. : 3.0
## 1st Qu.:	0.00000	1st Qu.:0.00000	1st Qu.:0.00000	1st Qu.:232.0
## Median :	0.00000	Median :0.00000	Median :0.00000	Median :232.0
## Mean :	0.02274	Mean :0.04447	Mean :0.05136	Mean :314.3
## 3rd Qu.:	0.00000	3rd Qu.:0.00000	3rd Qu.:0.00000	3rd Qu.:388.0
## Max. :	1.00000	Max. :1.00000	Max. :9.00000	Max. :692.0
## NA's	:21	NA's :21	NA's :14	NA's :21

	A13	A14	A15	A16
## Min.	: 0.00000	Min. :0.0000	Min. :0.00	Min. :0.00000
## 1st Qu.:	0.00000	1st Qu.:0.0000	1st Qu.:0.00	1st Qu.:0.00000
## Median :	0.00000	Median :0.0000	Median :0.00	Median :0.00000
## Mean :	0.01523	Mean :0.0854	Mean :0.15	Mean :0.03794
## 3rd Qu.:	0.00000	3rd Qu.:0.0000	3rd Qu.:0.00	3rd Qu.:0.00000
## Max. :	12.00000	Max. :1.0000	Max. :1.00	Max. :1.00000
## NA's	:30	NA's :21	NA's :20	NA's :23

	A17	A18	A19	A20
## Min.	:0.000	Min. : 4.00	Min. :0.0000	Min. :0.0000
## 1st Qu.:	1.000	1st Qu.: 14.00	1st Qu.:0.0000	1st Qu.:0.0000
## Median :	1.000	Median : 33.00	Median :0.0000	Median :0.0000
## Mean :	1.178	Mean : 61.08	Mean :0.1112	Mean :0.2376
## 3rd Qu.:	1.000	3rd Qu.: 89.00	3rd Qu.:0.0000	3rd Qu.:0.0000

```
## Max. :5.000 Max. :2082.00 Max. :1.0000 Max. :1.0000
## NA's :19 NA's :21 NA's :22 NA's :18
## A21 A22 A23 A24
## Min. :0.00000 Min. :0.006982 Min. : 0.00 Min. :0.000000
## 1st Qu.:0.00000 1st Qu.:0.050545 1st Qu.: 30.00 1st Qu.:0.007505
## Median :0.00000 Median :0.057638 Median :100.00 Median :0.079963
## Mean :0.02724 Mean :0.055725 Mean : 83.95 Mean :0.267362
## 3rd Qu.:0.00000 3rd Qu.:0.062879 3rd Qu.:111.00 3rd Qu.:0.522907
## Max. :3.00000 Max. :0.083627 Max. :985.00 Max. :0.522907
## NA's :18 NA's :17 NA's :18 NA's :19
## A25 Class
## Min. :0.000000 Min. :0.0000
## 1st Qu.:0.000000 1st Qu.:0.0000
## Median :0.000000 Median :0.0000
## Mean :0.000132 Mean :0.2705
## 3rd Qu.:0.000000 3rd Qu.:1.0000
## Max. :0.111000 Max. :1.0000
## NA's :15

## A01 A02 A03 A04 A05 A06 A07
## 1 10.08057 0.9944798 0.03897416 0.5313891 0.5828193 0.3419398 0.03890527
## A08 A09 A10 A11 A12 A13 A14
## 1 0.2191602 0.1491071 0.2061821 0.4470791 141.0333 0.3939249 0.2795417
## A15 A16 A17 A18 A19 A20 A21
## 1 0.3571616 0.1910905 0.6080678 100.063 0.3144882 0.4257441 0.1804822
## A22 A23 A24 A25 Class
## 1 0.01073047 60.04321 0.2521005 0.003523979 0.4443292
```

For the code chunk above, `echo = FALSE` parameter was added to prevent printing of the R code output as the output is lengthy.

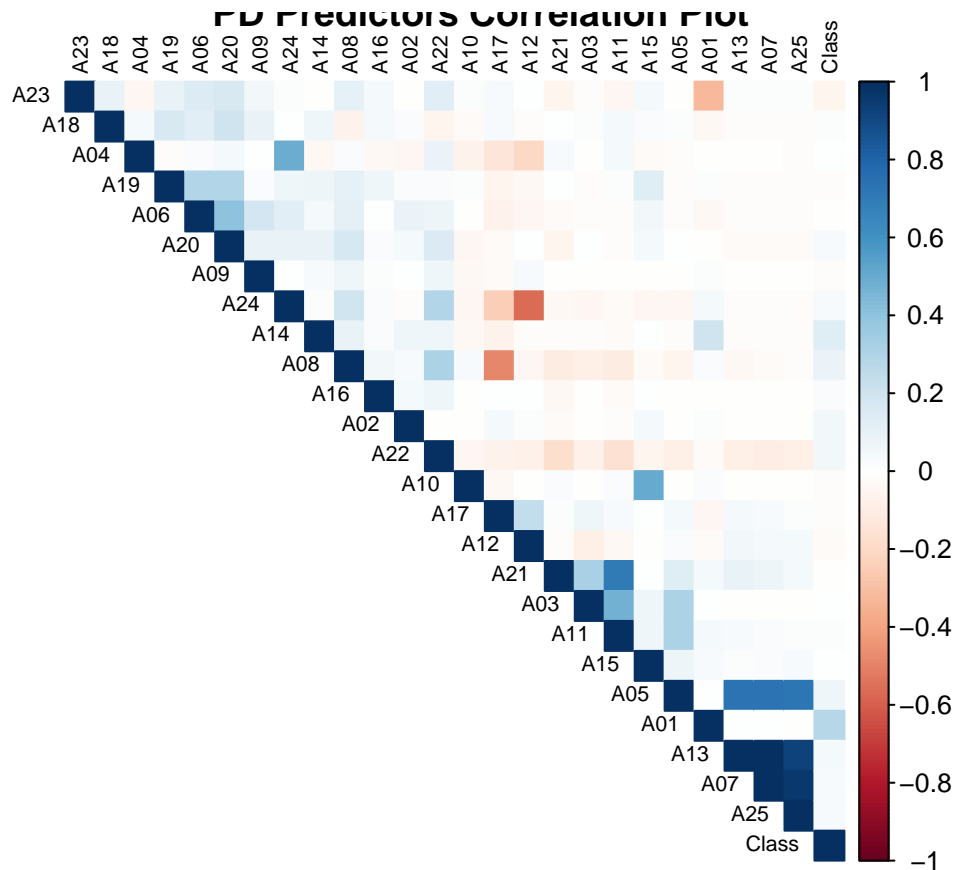
`str()` method was used to obtain information on which column is to be used for `summary()` method. From `str()`, it was known that all columns datatype were either integer or number so all columns were included in the `summary()` method to obtain the **Min, Max, Median, Mean, and the Number of NAs** within the columns. It is found that A01, A12, A18, and A23 have high standard deviation but any 4 of these attributes might be an important predictor for the models below so none of them will be removed.

The output from the code chunk above shows that all columns have NA values between the range of **14 to 30**, other than **column A01 and Class** which have **no NA values at all**. Column **A13** contains the highest number of missing values, totaling **30**. The output also shows that column **A01, A12, A18 and A23** have standard deviation more than 1 which means that their variability is high which indicates that the data in these columns have a wider spread around the mean. This then further suggest that the values in these columns are more dispersed and less consistent compared to columns with a standard deviation less than 1. Even with these high standard deviation, column **A01, A12, A18 and A23** are not removed as any one of these column could be important predictor on whether the class is legitimate or phishing.

Phishing dataset predictor correlation

```
# Compute the correlation matrix
cor_matrix <- cor(PD, use = "complete.obs") # Omit NA values

# Create a correlation plot
corrplot(cor_matrix, method = "color", order = "AOE", tl.cex = 0.7, tl.col = "black",
         title = "PD Predictors Correlation Plot", type = "upper")
```



Question 2

Remove rows with NA values

```
PD = data.frame(PD)
PD$Class = factor(PD$Class)
PD_na_free = PD[complete.cases(PD),]
PD_na_free = na.omit(PD)
dim(PD_na_free)
```

```
## [1] 1541 26
```

```
str(PD_na_free) # Confirm after removal of NA values, the predictor 'Class' is still a factor
```

```
## 'data.frame': 1541 obs. of 26 variables:
## $ A01 : int 11 25 31 3 3 3 11 10 6 13 ...
## $ A02 : int 0 0 0 0 0 1 0 0 0 0 ...
## $ A03 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A04 : int 3 2 3 2 3 3 2 2 2 3 ...
## $ A05 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A06 : int 1 1 0 0 0 0 0 0 0 0 ...
## $ A07 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A08 : num 1 1 0.441 0.714 1 ...
```

```
## $ A09 : int 1 0 0 0 0 0 0 0 0 0 ...
## $ A10 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A11 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A12 : int 232 365 232 648 232 232 388 648 226 232 ...
## $ A13 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A14 : int 0 1 0 0 0 0 0 0 0 0 ...
## $ A15 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A16 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A17 : int 1 1 1 2 1 1 2 2 1 1 ...
## $ A18 : int 96 89 17 42 29 39 27 46 5 97 ...
## $ A19 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A20 : int 1 1 0 0 1 0 0 1 0 1 ...
## $ A21 : int 0 0 0 0 0 0 0 0 0 0 ...
## $ A22 : num 0.0605 0.0601 0.0633 0.0626 0.0461 ...
## $ A23 : int 15 107 1 110 111 116 165 107 100 124 ...
## $ A24 : num 0.52291 0.00159 0.52291 0.02856 0.52291 ...
## $ A25 : num 0 0 0 0 0 0 0 0 0 0 ...
## $ Class: Factor w/ 2 levels "0","1": 1 1 1 1 1 2 1 1 1 1 ...
## - attr(*, "na.action")= 'omit' Named int [1:459] 2 10 11 16 23 25 28 29 41 43 ...
## ..- attr(*, "names")= chr [1:459] "69509" "65009" "61141" "91683" ...
```

Any occurrence of NA values, no matter how small, is eliminated since they can create uncertainty and lower the model's prediction accuracy, potentially resulting in biased or incorrect statistical conclusions. By making sure that the dataset is free of NA values, the algorithms can then work with fully populated data, which will lead to a more transparent understanding of the patterns and relationships within the data and its predictors.

After removing all rows that contain NA values within the Phishing dataset, there remain a total of 1541 rows within the Phishing dataset sample. `PD_na_free` contains the 1541 rows of the Phishing dataset that does not contain any NA values as shown below with `summary()` method.

```
##           A01           A02           A03           A04
## Min.      : 3.00   Min.      : 0.0000   Min.      :0.000000   Min.      :2.000
## 1st Qu.:10.00   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.:2.000
## Median :15.00   Median : 0.0000   Median :0.000000   Median :3.000
## Mean      :16.48   Mean      : 0.1635   Mean      :0.001947   Mean      :2.746
## 3rd Qu.:25.00   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.:3.000
## Max.      :35.00   Max.      :20.0000   Max.      :1.000000   Max.      :7.000
##           A05           A06           A07           A08
## Min.      : 0.0000   Min.      :0.0000   Min.      :0.000000   Min.      :0.1429
## 1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.000000   1st Qu.:0.6818
## Median : 0.0000   Median :0.0000   Median :0.000000   Median :1.0000
## Mean      : 0.0305   Mean      :0.1369   Mean      :0.001947   Mean      :0.8429
## 3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.:0.000000   3rd Qu.:1.0000
## Max.      :18.0000   Max.      :1.0000   Max.      :1.000000   Max.      :1.0000
##           A09           A10           A11           A12
## Min.      :0.00000   Min.      :0.00000   Min.      :0.00000   Min.      : 3.0
## 1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:232.0
## Median :0.00000   Median :0.00000   Median :0.00000   Median :232.0
## Mean      :0.02401   Mean      :0.04088   Mean      :0.04932   Mean      :314.4
## 3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:388.0
## Max.      :1.00000   Max.      :1.00000   Max.      :9.00000   Max.      :692.0
##           A13           A14           A15           A16
## Min.      : 0.00000   Min.      :0.00000   Min.      :0.00000   Min.      :0.00000
```

```
## 1st Qu.: 0.00000 1st Qu.:0.00000 1st Qu.:0.0000 1st Qu.:0.00000
## Median : 0.00000 Median :0.00000 Median :0.0000 Median :0.00000
## Mean : 0.01947 Mean :0.08177 Mean :0.1441 Mean :0.03829
## 3rd Qu.: 0.00000 3rd Qu.:0.00000 3rd Qu.:0.0000 3rd Qu.:0.00000
## Max. :12.00000 Max. :1.00000 Max. :1.0000 Max. :1.00000
## A17 A18 A19 A20
## Min. :0.000 Min. : 4.0 Min. :0.0000 Min. :0.0000
## 1st Qu.:1.000 1st Qu.: 14.0 1st Qu.:0.0000 1st Qu.:0.0000
## Median :1.000 Median : 31.0 Median :0.0000 Median :0.0000
## Mean :1.162 Mean : 58.9 Mean :0.1181 Mean :0.2375
## 3rd Qu.:1.000 3rd Qu.: 89.0 3rd Qu.:0.0000 3rd Qu.:0.0000
## Max. :5.000 Max. :2082.0 Max. :1.0000 Max. :1.0000
## A21 A22 A23 A24
## Min. :0.00000 Min. :0.009351 Min. : 0.00 Min. :0.00000
## 1st Qu.:0.00000 1st Qu.:0.050773 1st Qu.: 28.00 1st Qu.:0.00820
## Median :0.00000 Median :0.057820 Median :100.00 Median :0.07996
## Mean :0.02726 Mean :0.055790 Mean : 83.05 Mean :0.27178
## 3rd Qu.:0.00000 3rd Qu.:0.062943 3rd Qu.:110.00 3rd Qu.:0.52291
## Max. :3.00000 Max. :0.083627 Max. :985.00 Max. :0.52291
## A25 Class
## Min. :0.0000000 0:1115
## 1st Qu.:0.0000000 1: 426
## Median :0.0000000
## Mean :0.0001707
## 3rd Qu.:0.0000000
## Max. :0.1110000
```

Question 3

```
set.seed(33085625)
train.row = sample(1:nrow(PD_na_free), 0.7*nrow(PD_na_free))
PD_na_free.train = PD_na_free[train.row,]
PD_na_free.test = PD_na_free[-train.row,]
```

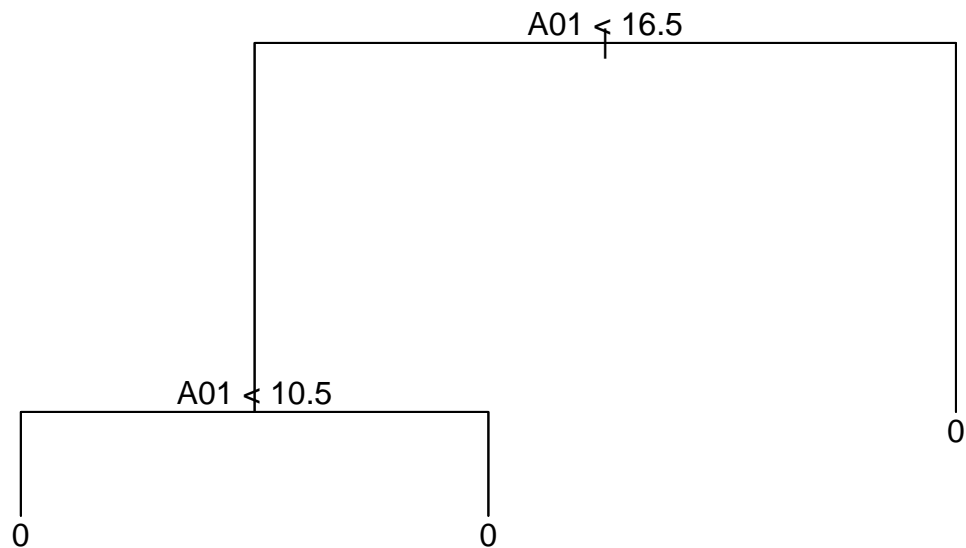
Question 4

```
decision_tree_model = tree(Class ~ ., data = PD_na_free.train)
summary(decision_tree_model)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = PD_na_free.train)
## Variables actually used in tree construction:
## [1] "A01"
## Number of terminal nodes: 3
## Residual mean deviance: 1.099 = 1182 / 1075
## Misclassification error rate: 0.2774 = 299 / 1078
```



```
plot(decision_tree_model, main = "Decision Tree")
text(decision_tree_model, pretty = 0)
```



```
naive_bayes_model = naiveBayes(Class ~., data = PD_na_free.train)
summary(naive_bayes_model)
```

```
##           Length Class  Mode
## apriori      2      table numeric
## tables      25     -none- list
## levels       2     -none- character
## isnumeric   25     -none- logical
## call        4     -none- call
```

```
# Check for NA values because Bagging algorithm cannot process NA values directly
sum(is.na(PD_na_free.train))
```

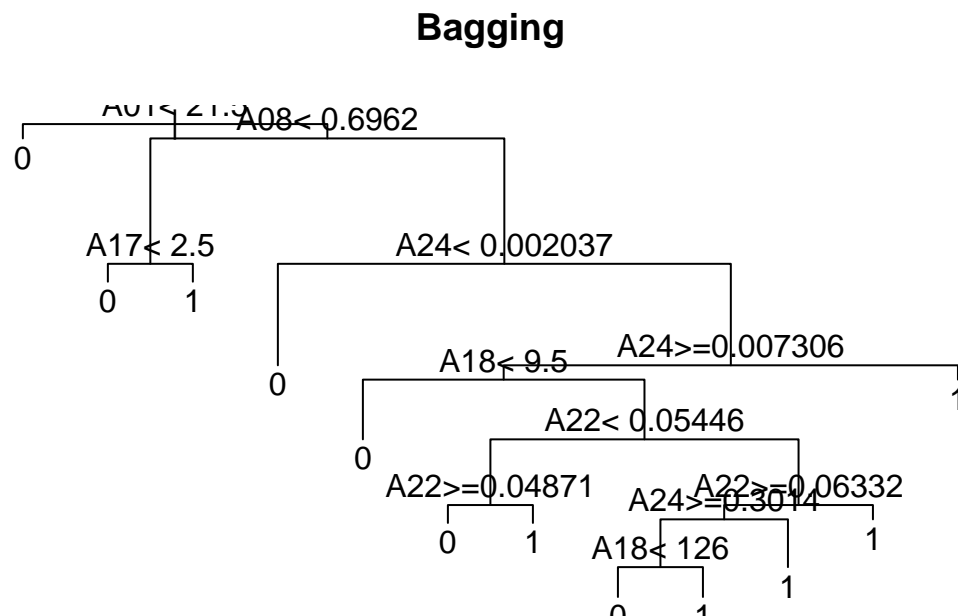
```
## [1] 0
```

```
bagging_model <- bagging(Class ~., data = PD_na_free.train, coob = TRUE, resampling = "bootstrap")
summary(bagging_model)
```

```
##           Length Class  Mode
## formula           3 formula call
```

```
## trees      100 -none- list
## votes      2156 -none- numeric
## prob       2156 -none- numeric
## class      1078 -none- character
## samples    107800 -none- numeric
## importance  25 -none- numeric
## terms       3 terms call
## call        5 -none- call
```

```
plot(bagging_model$trees[[1]], main = "Bagging")
text(bagging_model$trees[[1]], pretty = 0)
```



```
# Check for NA values because Boosting algorithm cannot process NA values directly
sum(is.na(PD_na_free.train))
```

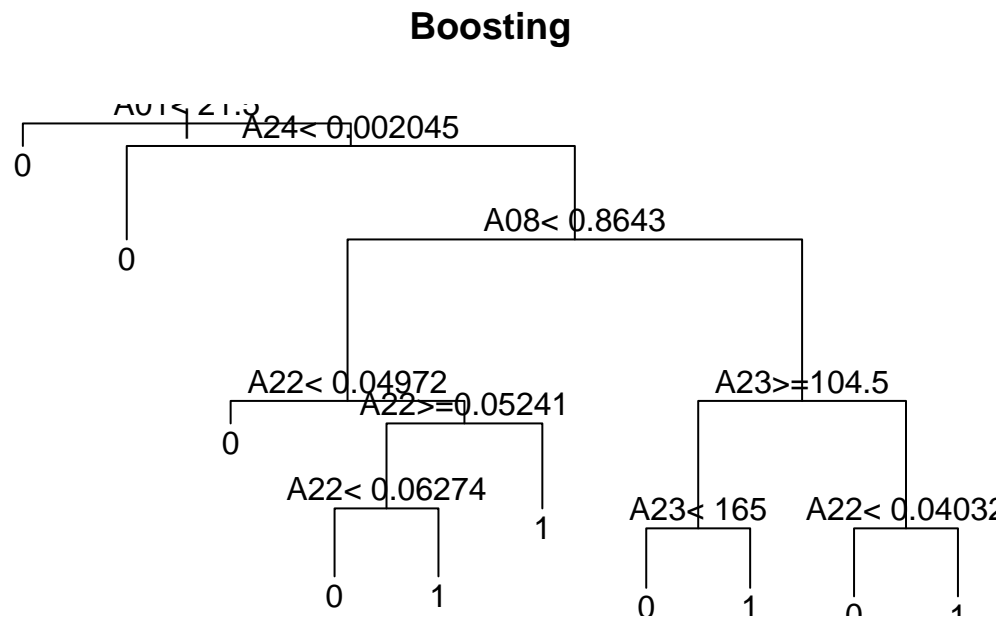
```
## [1] 0
```

```
boosting_model = boosting(Class ~., data = PD_na_free.train)
summary(boosting_model)
```

```
##          Length Class  Mode
## formula      3  formula call
## trees        100  -none- list
## weights       100  -none- numeric
```

```
## votes      2156  -none-  numeric
## prob      2156  -none-  numeric
## class     1078  -none-  character
## importance  25   -none-  numeric
## terms       3   terms   call
## call        3   -none-  call
```

```
plot(boosting_model$trees[[1]], main = "Boosting")
text(boosting_model$trees[[1]], pretty = 0)
```

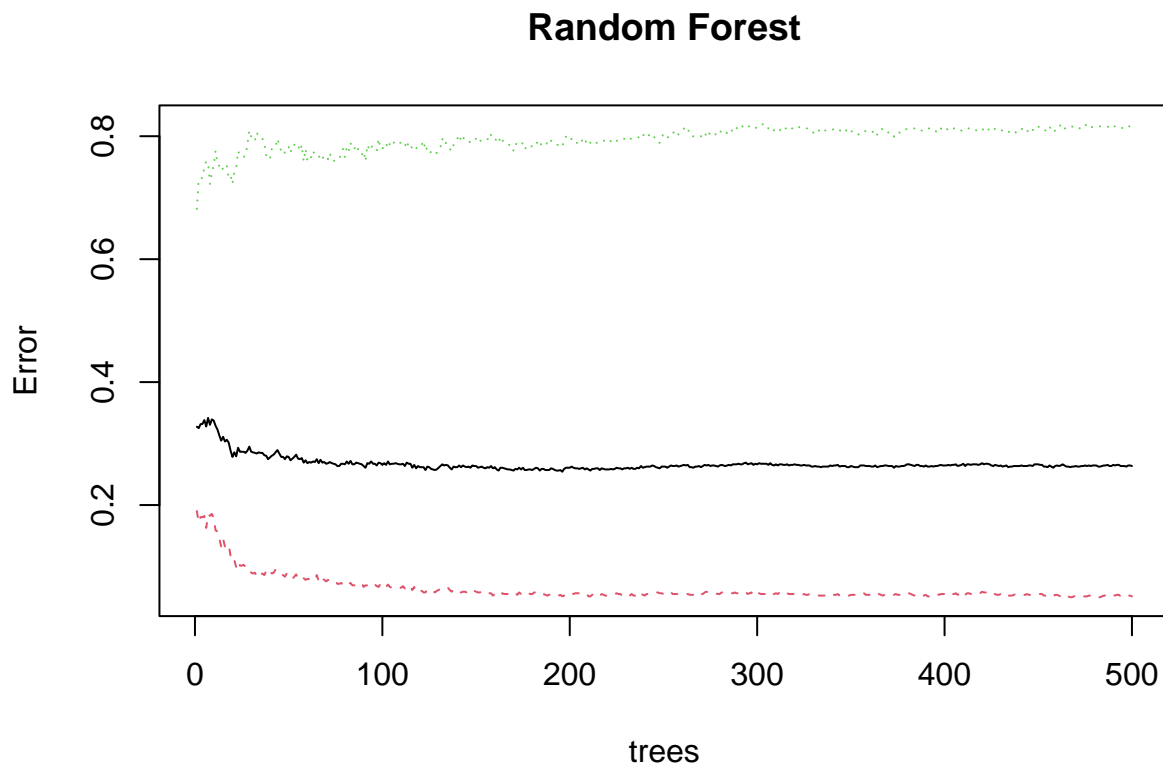


```
random_forest_model = randomForest(Class ~., data = PD_na_free.train)
summary(random_forest_model)
```

```
##           Length Class Mode
## call           3  -none- call
## type            1  -none- character
## predicted      1078 factor numeric
## err.rate       1500 -none- numeric
## confusion        6  -none- numeric
## votes          2156 matrix numeric
## oob.times       1078 -none- numeric
## classes         2   -none- character
## importance       25  -none- numeric
## importanceSD      0  -none- NULL
## localImportance  0  -none- NULL
```

```
## proximity      0  -none- NULL
## ntree          1  -none- numeric
## mtry           1  -none- numeric
## forest        14  -none- list
## y             1078 factor numeric
## test           0  -none- NULL
## inbag           0  -none- NULL
## terms          3  terms  call
```

```
plot(random_forest_model, main = "Random Forest")
```



Question 5

Phishing is labelled as 1

Legitimate is labelled as 0

In the confusion matrix obtained below, there is True Positives, True Negatives, False Positives and False Negatives given in values.

- True Positives (TP) are the cases where the model correctly predicted the positive class (phishing). In the confusion matrix, this would be in the bottom right cell.
- True Negatives (TN) are the cases where the model correctly predicted the negative class (legitimate). This is the top left cell of the confusion matrix.

- False Positives (FP) are the cases where the model incorrectly predicted the positive class. In the confusion matrix, this would be the top right cell.
- False Negatives (FN) are the cases where the model incorrectly predicted the negative class. This is the bottom left cell of the confusion matrix.

It should also be known that the accuracy of the model is calculated as: $\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{FP} + \text{FN} + \text{TP} + \text{TN})$ where $(\text{TP} + \text{TN}) / (\text{FP} + \text{FN} + \text{TP} + \text{TN})$ is the formula to calculate the accuracy of the model.

```
# Predict using the testing dataset after training the model
predict_decision_tree = predict(decision_tree_model, newdata = PD_na_free.test, type = "class")

# Evaluating the model performance
# Decision Tree Confusion Matrix
result_decision_tree = table(actual = PD_na_free.test$Class, prediction = predict_decision_tree)
colnames(result_decision_tree) = c("legitimate", "phishing")
rownames(result_decision_tree) = c("legitimate", "phishing")
result_decision_tree

##           prediction
## actual      legitimate phishing
## legitimate      336         0
## phishing        127         0

# Evaluating the model accuracy
decision_tree_accuracy = (result_decision_tree[2, 2] + result_decision_tree[1, 1]) / (result_decision_t.
decision_tree_accuracy

## [1] 0.7257019
```

From the confusion matrix displayed above in the output, it is displayed that:

- The TP for the confusion matrix for decision tree is 0, indicating that the model did not correctly predict any phishing cases.
- The TN for the confusion matrix for decision tree is 336, indicating that the model correctly identified 336 legitimate cases.
- The FP for the confusion matrix for decision tree is 0, indicating that the model did not incorrectly predict any legitimate cases as phishing.
- The FN for the confusion matrix for decision tree is 127, indicating that the model incorrectly identified 127 phishing cases as legitimate.

Given the values from the confusion matrix above, the accuracy of the decision tree model is 0.7257019 which is approximately 72.57%.

This indicates that the decision tree model accurately recognized around 72.57% of the instances. Nevertheless, there is a major problem with the model as it failed to accurately detect any instances of phishing, as shown by the TP cell displaying zero. Every phishing incident was labeled as legitimate, posing a significant problem for a phishing detection algorithm. This might result from a few factors like overfitting or there is a lacking representative features for the phishing class, or even an uneven distribution in the training data.

```

# Predict using the testing dataset after training the model
predict_naive_bayes = predict(naive_bayes_model, newdata = PD_na_free.test, type = "class")

# Evaluating the model performance
# Naïve-Bayes Confusion Matrix
result_naive_bayes = table(actual = PD_na_free.test$Class, predicted = predict_naive_bayes)
colnames(result_naive_bayes) = c("legitimate", "phishing")
rownames(result_naive_bayes) = c("legitimate", "phishing")
result_naive_bayes

```

```

##           predicted
## actual      legitimate phishing
## legitimate      335         1
## phishing       126         1

```

```

# Evaluating the model accuracy
naive_bayes_accuracy = (result_naive_bayes[2, 2] + result_naive_bayes[1, 1]) / (result_naive_bayes[2, 2] + result_naive_bayes[2, 1] + result_naive_bayes[1, 2] + result_naive_bayes[1, 1])
naive_bayes_accuracy

```

```
## [1] 0.7257019
```

From the confusion matrix displayed above in the output, it is shown that:

- The TP for the confusion matrix for Naïve-Bayes is 1, indicating that the model only correctly predict 1 phishing case.
- The TN for the confusion matrix for Naïve-Bayes is 335, indicating that the model correctly identified 335 legitimate cases.
- The FP for the confusion matrix for Naïve-Bayes is 1, indicating that the model did only predict 1 legitimate case as phishing.
- The FN for the confusion matrix for Naïve-Bayes is 126, indicating that the model incorrectly identified 126 phishing cases as legitimate.

Given the values from the confusion matrix above, the accuracy of the Naïve-Bayes model is same as the decision tree model accuracy of 0.7257019 which is approximately 72.57%.

Eventhough both decision tree model and Naïve-Bayes model have the same accuracy, the Naïve-Bayes model has managed to correctly identify 1 phishing case. Yet even with the only correct identification of the 1 phishing case, the Naïve-Bayes model still shows a significant bias towards predicting legitimate cases, with a high number of 126 false negatives which are the phishing cases misclassified as legitimate.

```

# Predict using the testing dataset after training the model
predict_bagging = predict(bagging_model, newdata = PD_na_free.test, type = "class")

# Evaluating the model performance
# Bagging Confusion Matrix
result_bagging = predict_bagging$confusion
colnames(result_bagging) = c("legitimate", "phishing")
rownames(result_bagging) = c("legitimate", "phishing")
result_bagging

```

```
##               Observed Class
## Predicted Class legitimate phishing
##      legitimate      315      101
##      phishing       21       26

# Evaluating the model accuracy
bagging_accuracy = (result_bagging[2, 2] + result_bagging[1, 1]) / (result_bagging[2, 2] + result_bagging[1, 1] + result_bagging[2, 1] + result_bagging[1, 2])
bagging_accuracy

## [1] 0.7365011
```

From the confusion matrix displayed above in the output, it is known that:

- The TP for the confusion matrix for Bagging is 26, indicating that the model only correctly predict 26 phishing cases.
- The TN for the confusion matrix for Bagging is 315, indicating that the model correctly identified 315 legitimate cases.
- The FP for the confusion matrix for Bagging is 101, indicating that the model did only predict 101 legitimate cases as phishing.
- The FN for the confusion matrix for Bagging is 21, indicating that the model incorrectly identified 21 phishing cases as legitimate.

Given the values from the confusion matrix above, the accuracy of the Bagging model is 0.7365011 which is approximately 73.65%. The accuracy of the Bagging model is slightly higher than the accuracy for Naïve-Bayes and decision tree model.

The bagging model has a better balance between predicting legitimate and phishing cases compared to the decision tree and Naïve-Bayes models as the bagging model has correctly identified more phishing cases of 26 compared to the previous models, indicating a better performance in detecting phishing attempts than the previous prediction models.

```
# Predict using the testing dataset after training the model
predict_boosting = predict(boosting_model, newdata = PD_na_free.test, type = "class")

# Evaluating the model performance
# Boosting Confusion Matrix
result_boosting = predict_boosting$confusion
colnames(result_boosting) = c("legitimate", "phishing")
rownames(result_boosting) = c("legitimate", "phishing")
result_boosting

##               Observed Class
## Predicted Class legitimate phishing
##      legitimate      281      97
##      phishing       55      30

# Evaluating the model accuracy
boosting_accuracy = (result_boosting[2, 2] + result_boosting[1, 1]) / (result_boosting[2, 2] + result_boosting[1, 1] + result_boosting[2, 1] + result_boosting[1, 2])
boosting_accuracy
```

```
## [1] 0.6717063
```

From the confusion matrix displayed above in the output, it is presented that:

- The TP for the confusion matrix for Boosting is 30, indicating that the model only correctly predict 30 phishing cases.
- The TN for the confusion matrix for Boosting is 281, indicating that the model correctly identified 281 legitimate cases.
- The FP for the confusion matrix for Boosting is 97, indicating that the model did only predict 97 legitimate cases as phishing.
- The FN for the confusion matrix for Boosting is 55, indicating that the model incorrectly identified 55 phishing cases as legitimate.

Given the values from the confusion matrix above, the accuracy of the Boosting model is 0.6717063 which is approximately 67.17%. The accuracy of 67.17% is the lowest prediction model accuracy compared to the Bagging, decision tree, Naïve-Bayes model above and Random forest model below.

From the confusion matrix above, it is shown that the Boosting model has identified more phishing cases of 30 compared to the decision tree and Naïve-Bayes models but fewer than the bagging model. Other than that, the number of false positives is 97 which is high, indicating that many legitimate cases were incorrectly classified as phishing.

```
# Predict using the testing dataset after training the model
predict_random_forest = predict(random_forest_model, newdata = PD_na_free.test, type = "class")

# Evaluating the model performance
# Random Forest Confusion Matrix
result_random_forest = table(actual = PD_na_free.test$Class, predicted = predict_random_forest)
colnames(result_random_forest) = c("legitimate", "phishing")
rownames(result_random_forest) = c("legitimate", "phishing")
result_random_forest
```

```
##           predicted
## actual    legitimate phishing
## legitimate      314       22
## phishing        103       24
```

```
# Evaluating the model accuracy
random_forest_accuracy = (result_random_forest[2, 2] + result_random_forest[1, 1]) / (result_random_forest[2, 2] + result_random_forest[2, 1] + result_random_forest[1, 2] + result_random_forest[1, 1])
random_forest_accuracy
```

```
## [1] 0.7300216
```

From the confusion matrix displayed above in the output, it is presented that:

- The TP for the confusion matrix for Random Forest is 24, indicating that the model only correctly predict 24 phishing cases.
- The TN for the confusion matrix for Random Forest is 314, indicating that the model correctly identified 314 legitimate cases.

- The FP for the confusion matrix for Random Forest is 22, indicating that the model did only predict 22 legitimate cases as phishing.
- The FN for the confusion matrix for Random Forest is 103, indicating that the model incorrectly identified 103 phishing cases as legitimate.

Given the values from the confusion matrix above, the accuracy of the Random Forest model is 0.7300216 which is approximately 73.00%. This accuracy is comparable to the accuracy of the Bagging model which is approximately 73.65%.

The Random Forest model has a moderate number of both false positives and false negatives, indicating that this Random Forest model has a more balanced approach to classifying legitimate and phishing cases compared to some of the other models. Nonetheless, the Bagging model has a higher accuracy than this Random Forest model by approximately 0.65%. This Random Forest model has correctly identified 24 phishing cases, which is fewer than the bagging model but more than the decision tree and Naïve-Bayes models.

Question 6

Calculate Confidence Level, Prediction and Performance of model

```
confidence_decision_tree = predict(decision_tree_model, newdata = PD_na_free.test, type = "vector")

prediction_decision_tree = prediction(confidence_decision_tree[, 2], PD_na_free.test$Class)
prediction_decision_tree
```

```
## A prediction instance
##   with 463 data points
```

```
performance_decision_tree = performance(prediction_decision_tree, "tpr", "fpr")
performance_decision_tree
```

```
## A performance instance
##   'False positive rate' vs. 'True positive rate' (alpha: 'Cutoff')
##   with 4 data points
```

```
confidence_naive_bayes = predict(naive_bayes_model, newdata = PD_na_free.test, type = "raw")

prediction_naive_bayes = prediction(confidence_naive_bayes[, 2], PD_na_free.test$Class)
performance_naive_bayes = performance(prediction_naive_bayes, "tpr", "fpr")
```

```
confidence_bagging = predict(bagging_model, newdata = PD_na_free.test, type = "prob")
confidence_bagging = confidence_bagging$prob
confidence_bagging = confidence_bagging[, 1:2]

prediction_bagging = prediction(confidence_bagging[, 2], PD_na_free.test$Class)
performance_bagging = performance(prediction_bagging, "tpr", "fpr")
```

```
confidence_boosting = predict(boosting_model, newdata = PD_na_free.test, type = "prob")
confidence_boosting = confidence_boosting$prob
```

```
prediction_boosting = prediction(confidence_boosting[, 2], PD_na_free.test$Class)
performance_boosting = performance(prediction_boosting, "tpr", "fpr")
```

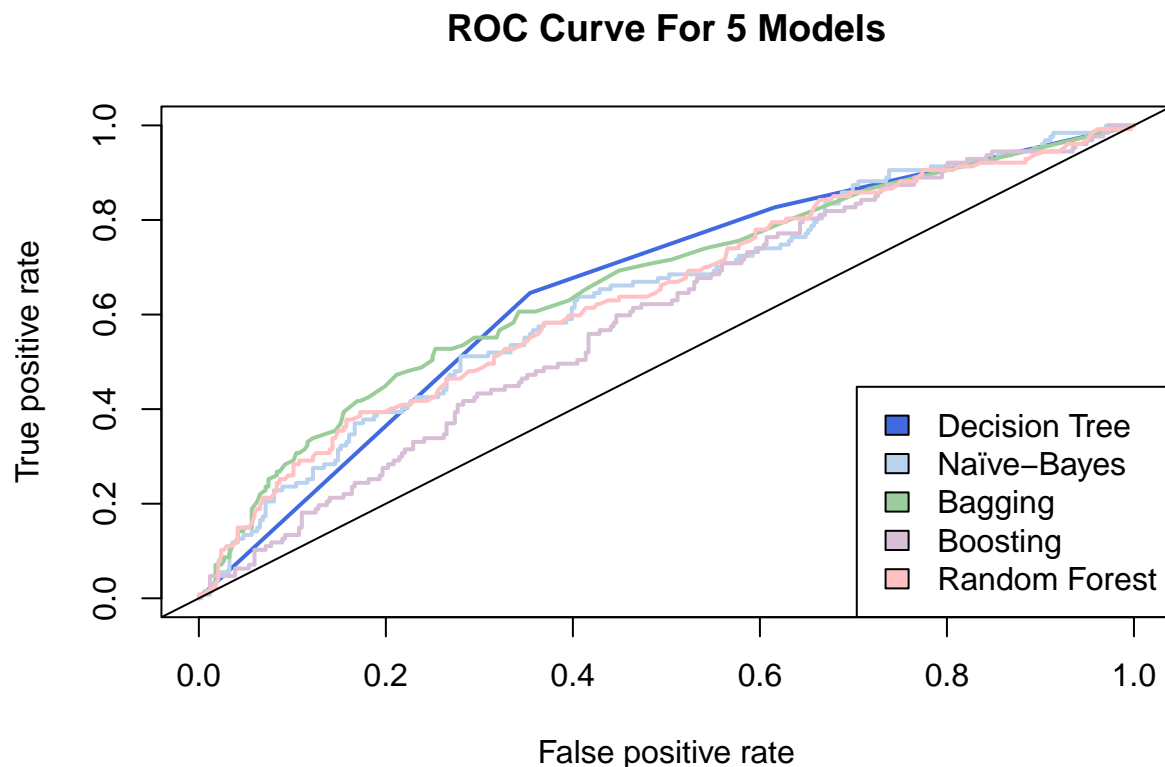
```
confidence_random_forest = predict(random_forest_model, newdata = PD_na_free.test, type = "prob")
```

```
prediction_random_forest = prediction(confidence_random_forest[, 2], PD_na_free.test$Class)
performance_random_forest = performance(prediction_random_forest, "tpr", "fpr")
```

Plot ROC Curve For 5 Models

```
plot(performance_decision_tree, col = "royalblue", main = "ROC Curve For 5 Models", lwd = 2)
plot(performance_naive_bayes, col = "slategray2", add = TRUE, lwd = 2)
plot(performance_bagging, col = "darkseagreen3", add = TRUE, lwd = 2)
plot(performance_boosting, col = "thistle", add = TRUE, lwd = 2)
plot(performance_random_forest, col = "rosybrown1", add = TRUE, lwd = 2)
# lwd was used to increase the thickness of the line in the plot
abline(0, 1)
```

```
legend("bottomright", legend = c("Decision Tree", "Naïve-Bayes", "Bagging", "Boosting", "Random Forest"))
```



The ROC Curve is a graphical method used to evaluate the performance of binary classifiers, specifically focusing on the 'Class' column in this scenario. A curve located near the upper-left corner suggests a better model. If the curve is closer to the top-left corner, it indicates a better model. By examining the curves, it is possible to assess on which model achieves optimal balance between TPR and FPR.

In the ROC Curve shown, the y-axis represents the True Positive Rate (TPR), which indicates the percentage of true positive cases correctly recognized by the model, while the x-axis represents the False Positive Rate (FPR), showing the percentage of true negative cases mistakenly classified as positive.

To identify the best model for prediction, we will use the Area Under Curve (AUC) value as the model with the greatest area under its ROC curve is typically seen as the best model.

Calculate AUC Value for each models

```
auc_decision_tree = performance(prediction_decision_tree, "auc")
auc_decision_tree_num = as.numeric(auc_decision_tree@y.values)
auc_decision_tree_num
```

```
## [1] 0.6578318
```

```
auc_naive_bayes = performance(prediction_naive_bayes, "auc")
auc_naive_bayes_num = as.numeric(auc_naive_bayes@y.values)
auc_naive_bayes_num
```

```
## [1] 0.63967
```

```
auc_bagging = performance(prediction_bagging, "auc")
auc_bagging_num = as.numeric(auc_bagging@y.values)
auc_bagging_num
```

```
## [1] 0.6660222
```

```
auc_boosting = performance(prediction_boosting, "auc")
auc_boosting_num = as.numeric(auc_boosting@y.values)
auc_boosting_num
```

```
## [1] 0.5934102
```

```
auc_random_forest = performance(prediction_random_forest, "auc")
auc_random_forest_num = as.numeric(auc_random_forest@y.values)
auc_random_forest_num
```

```
## [1] 0.6404199
```

Based on the Area Under the Curve (AUC) values calculated above, below is the ranking of these values from large to small:

1. Bagging: AUC value = 0.6660222

2. Decision Tree: AUC value = 0.6578318
3. Random Forest: AUC value = 0.6404199
4. Naïve-Bayes: AUC value = 0.63967
5. Boosting: AUC value = 0.5934102

The Bagging model stands out with the highest AUC value in the ranking, establishing itself as the top performer in ROC curve analysis among the listed models. The closer the AUC is to 1, the better the model is at distinguishing between positive and negative classes.

Question 7

```
decision_tree_accuracy_q7 = performance(prediction_decision_tree, "acc")
decision_tree_accuracy_q7_num = as.numeric(max(decision_tree_accuracy_q7@y.values[[1]]))
decision_tree_accuracy_q7_num
```

```
## [1] 0.7257019
```

```
naive_bayes_accuracy_q7 = performance(prediction_naive_bayes, "acc")
naive_bayes_accuracy_q7_num = as.numeric(max(naive_bayes_accuracy_q7@y.values[[1]]))
naive_bayes_accuracy_q7_num
```

```
## [1] 0.7321814
```

```
bagging_accuracy_q7 = performance(prediction_bagging, "acc")
bagging_accuracy_q7_num = as.numeric(max(bagging_accuracy_q7@y.values[[1]]))
bagging_accuracy_q7_num
```

```
## [1] 0.7408207
```

```
boosting_accuracy_q7 = performance(prediction_boosting, "acc")
boosting_accuracy_q7_num = as.numeric(max(boosting_accuracy_q7@y.values[[1]]))
boosting_accuracy_q7_num
```

```
## [1] 0.7300216
```

```
random_forest_accuracy_q7 = performance(prediction_random_forest, "acc")
random_forest_accuracy_q7_num = as.numeric(max(random_forest_accuracy_q7@y.values[[1]]))
random_forest_accuracy_q7_num
```

```
## [1] 0.7365011
```

```
q5_model_accuracy = c(decision_tree_accuracy, naive_bayes_accuracy, bagging_accuracy, boosting_accuracy)
q6_model_accuracy = c(decision_tree_accuracy_q7_num, naive_bayes_accuracy_q7_num, bagging_accuracy_q7_num, random_forest_accuracy_q7_num)
average_model_accuracy = c((decision_tree_accuracy + decision_tree_accuracy_q7_num) / 2, (naive_bayes_accuracy + naive_bayes_accuracy_q7_num) / 2, (bagging_accuracy + bagging_accuracy_q7_num) / 2, (boosting_accuracy + boosting_accuracy_q7_num) / 2)
```

```
comparison_tbl = data.frame(q5_model_accuracy, q6_model_accuracy, average_model_accuracy)
rownames(comparison_tbl) = c("Decision Tree", "Naïve Bayes", "Bagging", "Boosting", "Random Forest")
colnames(comparison_tbl) = c("Question 5 Model Accuracy", "Question 6 Model Accuracy", "Average Model Accuracy")
comparison_tbl
```

```
##           Question 5 Model Accuracy Question 6 Model Accuracy
## Decision Tree           0.7257019           0.7257019
## Naïve Bayes             0.7257019           0.7321814
## Bagging                 0.7365011           0.7408207
## Boosting                0.6717063           0.7300216
## Random Forest           0.7300216           0.7365011
##           Average Model Accuracy
## Decision Tree           0.7257019
## Naïve Bayes             0.7289417
## Bagging                 0.7386609
## Boosting                0.7008639
## Random Forest           0.7332613
```

From looking at the AUC values alone, it is shown that the Bagging model appears to be the best classifier among all the other models listed. Even without referring to the AUC values, Bagging model still shows the highest accuracy in both Question 5 and Question 6, as well as the highest average accuracy. Therefore, based on these accuracy metrics alone, Bagging model can be seen as a single best classifier.

Question 8

```
summary(decision_tree_model)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = PD_na_free.train)
## Variables actually used in tree construction:
## [1] "A01"
## Number of terminal nodes: 3
## Residual mean deviance: 1.099 = 1182 / 1075
## Misclassification error rate: 0.2774 = 299 / 1078
```

For the decision tree model, the most important variables in predicting whether a web site will be phishing or legitimate is A01.

```
acc = c()
for(tab in naive_bayes_model[["tables"]]) {
  acc = c(acc, ((tab[2, 2] + tab[1, 1]) / (tab[2, 2] + tab[1, 1] + tab[2, 1] + tab[1, 2])))
}

naive_bayes_model_accuracy = data.frame(Attributes = colnames(PD_na_free[, 1:25]), Accuracy = acc)
naive_bayes_model_accuracy = naive_bayes_model_accuracy[order(naive_bayes_model_accuracy$Accuracy, decreasing = TRUE), ]
naive_bayes_model_accuracy
```

```
##      Attributes Accuracy
```

```
## 25      A25 0.9278274
## 13      A13 0.9249882
## 7       A07 0.9242796
## 5       A05 0.9098618
## 3       A03 0.6014463
## 11      A11 0.5558329
## 2       A02 0.5314883
## 23      A23 0.5311506
## 14      A14 0.5223521
## 21      A21 0.5056554
## 16      A16 0.5044564
## 15      A15 0.5001736
## 6       A06 0.5000609
## 17      A17 0.4996931
## 12      A12 0.4982039
## 4       A04 0.4971695
## 19      A19 0.4970307
## 20      A20 0.4962195
## 22      A22 0.4920418
## 24      A24 0.4879557
## 9       A09 0.4858980
## 8       A08 0.4847184
## 10      A10 0.4846875
## 1       A01 0.4469351
## 18      A18 0.4465845
```

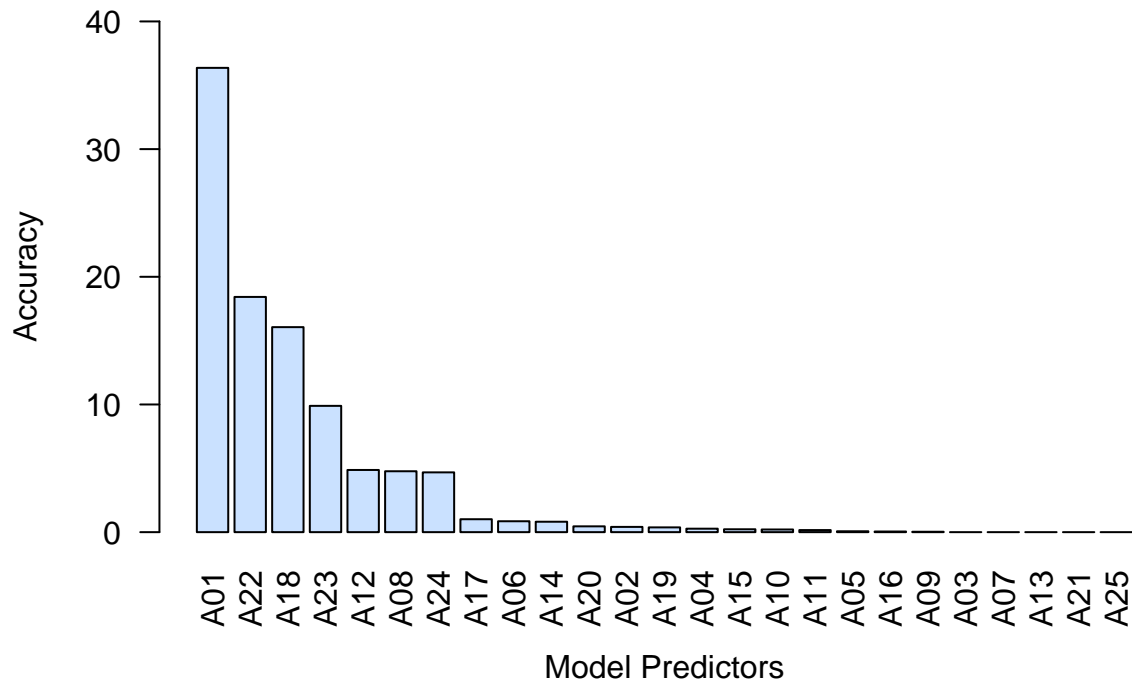
For the naive bayes model, the three most important variables in predicting whether a web site will be phishing or legitimate is A25, A13 and A07 but A25 would be the most important variables in predicting whether a web site will be phishing or legitimate as it has the highest accuracy value of 0.9278274, indicating that it correctly predicts the outcome more often than the other attributes.

```
bagging_model$importance
```

```
##      A01      A02      A03      A04      A05      A06
## 36.36463533 0.41138344 0.00000000 0.27093448 0.06862954 0.85418473
##      A07      A08      A09      A10      A11      A12
## 0.00000000 4.77056071 0.03070511 0.20903342 0.16311832 4.87027002
##      A13      A14      A15      A16      A17      A18
## 0.00000000 0.81613788 0.22520005 0.05043278 1.01303572 16.05690360
##      A19      A20      A21      A22      A23      A24
## 0.37296691 0.45473774 0.00000000 18.42330859 9.89217207 4.68164955
##      A25
## 0.00000000
```

```
barplot(bagging_model$importance[order(bagging_model$importance, decreasing = TRUE)], ylim = c(0, 40), l
```

Bagging Model Important Predictors



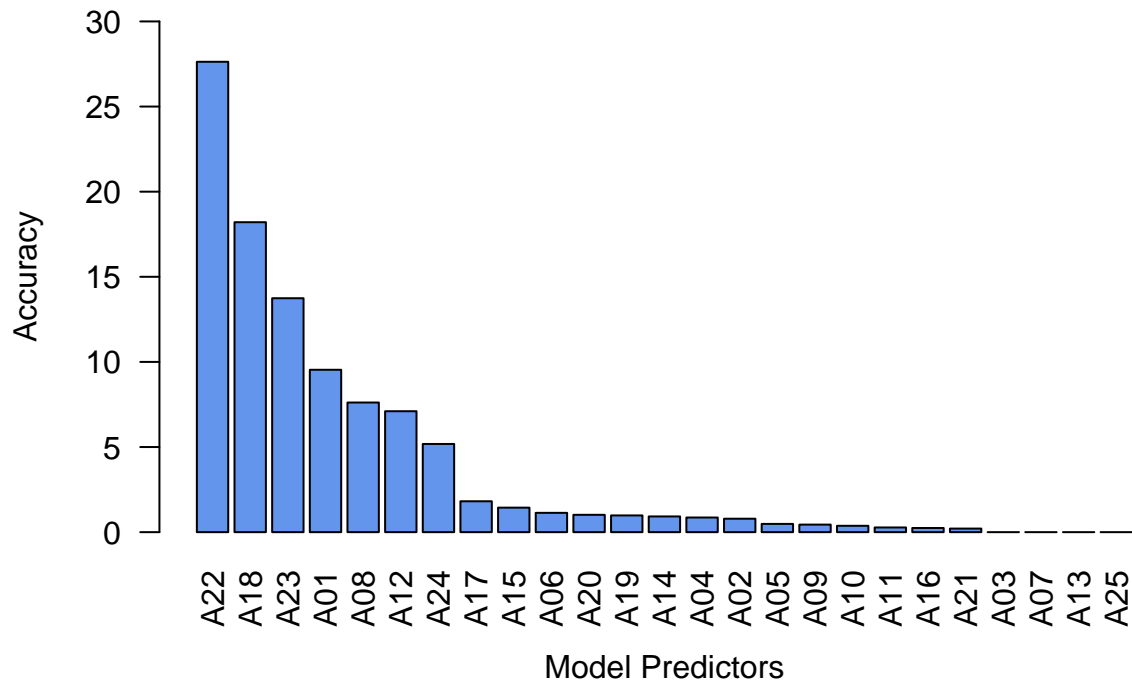
It is slightly difficult to identify which model predictor for the Bagging Model is the most important variables in predicting whether a web site will be phishing or legitimate so a bar plot is used to visualise it. From the bar plot, it is known that A01 is most important variables in predicting whether a web site will be phishing or legitimate for the Bagging Model as it has the highest accuracy of 36.36463533.

```
boosting_model$importance
```

```
##      A01      A02      A03      A04      A05      A06      A07
## 9.5373975 0.7898486 0.0000000 0.8596421 0.4837477 1.1336361 0.0000000
##      A08      A09      A10      A11      A12      A13      A14
## 7.6146331 0.4438161 0.3728328 0.2754940 7.1035051 0.0000000 0.9219427
##      A15      A16      A17      A18      A19      A20      A21
## 1.4388227 0.2437279 1.8133112 18.2057291 0.9815843 1.0184811 0.2099097
##      A22      A23      A24      A25
## 27.6298532 13.7412666 5.1808184 0.0000000
```

```
barplot(boosting_model$importance[order(boosting_model$importance, decreasing = TRUE)], ylim = c(0, 30))
```

Boosting Model Important Predictors



Determining the most significant predictor for the Boosting Model in distinguishing between a phishing website and a legitimate one can be somewhat challenging, hence the use of a bar plot for visualization. The bar plot reveals that A22 is the most crucial variable for determining if a website is phishing or legitimate in the Boosting Model due to its high accuracy of 27.6298532.

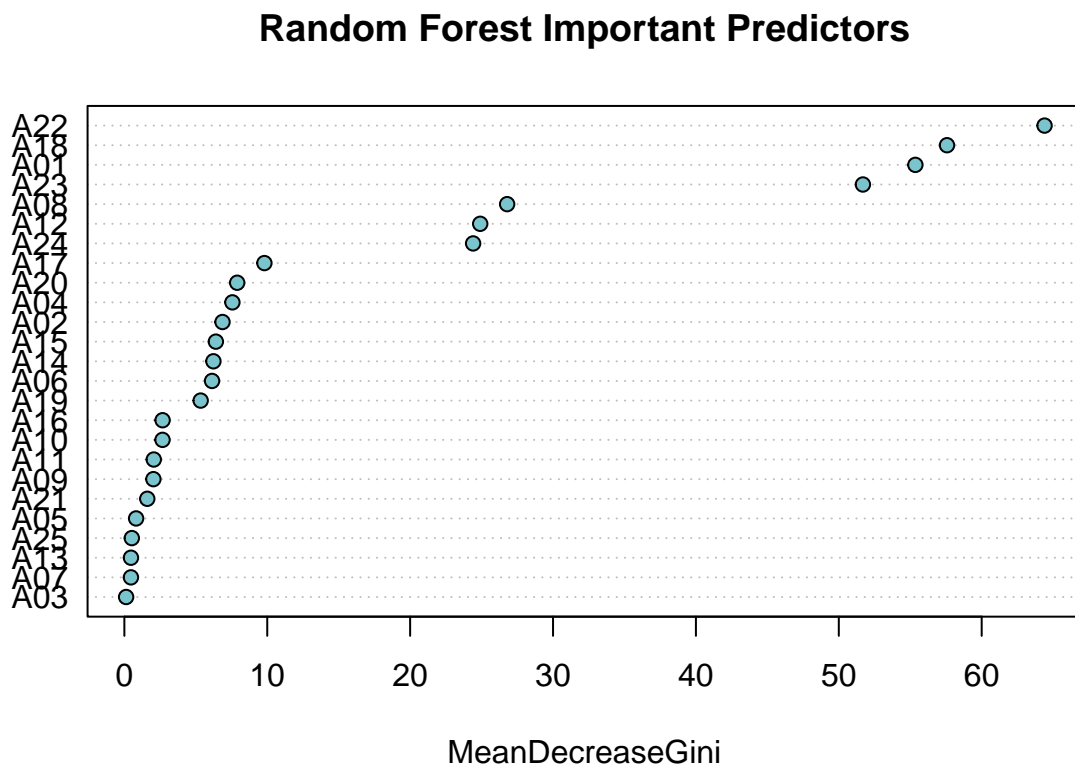
```
random_forest_model$importance
```

```
##      MeanDecreaseGini
## A01      55.3583242
## A02       6.8675525
## A03       0.1225678
## A04       7.5604378
## A05       0.8210876
## A06       6.1359170
## A07       0.4597608
## A08      26.7886785
## A09       2.0297776
## A10       2.6695832
## A11       2.0573564
## A12      24.9033090
## A13       0.4610878
## A14       6.2308689
## A15       6.4040187
## A16       2.6742955
## A17       9.7991893
## A18      57.5730872
```



```
## A19      5.3395807
## A20      7.8959540
## A21      1.6025464
## A22     64.3995504
## A23     51.6875572
## A24     24.4078212
## A25      0.5204239
```

```
varImpPlot(random_forest_model, main = "Random Forest Important Predictors", bg = "cadetblue3", cex=1)
```



It is somewhat challenging to determine the most important variables in predicting if a website is phishing or legitimate using the Random Forest Model, thus the varImpPlot library is utilized to visualize the model. According to the visualization, A22 is the most crucial variable for predicting if a website is phishing or legitimate in the Random Forest Model, with the highest accuracy of 64.3995504.

Variables that could be omitted from the data with very little effect on performance are variables with the lowest importance and/or accuracy scores. First, the few lowest predictors will be evaluated from model to model before deciding on which variables are the lowest predictors for each of the models.

For the Decision Tree model, only one variable, A01, was actually used in tree construction which suggested that A01 is a significant predictor for the Decision Tree model.

For the Naïve-Bayes Model, these are the 3 predictors with the lowest score:

1. A18: 0.4465845
2. A01: 0.4469351

3. A10: 0.4846875

For the Bagging Model, these are the 3 predictors with the lowest score:

1. A03, A07, A13, A21, A25: 0.00000000
2. A09: 0.03070511
3. A16: 0.05043278

For the Boosting Model, these are the 3 predictors with the lowest score:

1. A03, A07, A13, A25: 0.00000000
2. A21: 0.2099097
3. A10: 0.3728328

For the Random Forest Model, these are the 3 predictors with the lowest score:

1. A03: 0.1225678
2. A07: 0.4597608
3. A13: 0.4610878

According to the information gain displayed above, variables that consistently exhibit minimal importance or accuracy across the 5 models could possibly be excluded from the data without significantly impacting performance. The variables include A03, A07, A10, and A13. A03 showed poor performance in the Random Forest, Boosting, and Bagging models, and did not contribute at all in the Boosting and Bagging models, resulting in its exclusion. A07 is comparable to A03, with low scores in the Random Forest model and no contribution in either the Boosting or Bagging models, hence, its excluded. A13 is of minimal significance in the Random Forest model and makes no impact in the Boosting and Bagging models, hence it is omitted. A10 scored poorly in the Naïve Bayes model and ranks near the bottom in the Boosting model, therefore it is also excluded. While A25 does not contribute to Boosting and Bagging models, it is a significant predictor in the Naïve Bayes model, so it is not excluded as it could be valuable in specific contexts or scenarios.

Hence, in conclusion, the variables that could be omitted from the data includes A03, A07, A10, and A13 as these variables does not provide any important information when classifying.

Question 9

From Question 8, it is known that the important predictors are basically A01, A18, A22, A23, and A25, hence these predictors are also chosen to be used here. Despite the low score of A01 in the Naïve Bayes model, it was the only variable used in the Decision Tree model's construction which indicates its potential significance. A25 is the best predictor in the Naïve Bayes model while A22 has the highest MeanDecreaseGini score in the Random Forest model so both A25 and A22 are chosen. Furthermore, A18 and A23 also have high MeanDecreaseGini scores in the Random Forest model which strongly suggest that they both are also important predictors.

```
new_PD = data.frame(PD)
new_PD$Class = factor(new_PD$Class)
new_PD = subset(new_PD, select = c(1, 18, 22, 23, 25, 26))
new_PD = new_PD[complete.cases(new_PD),]
new_PD = na.omit(new_PD)

str(new_PD) # Confirm after removal of NA values, the predictor 'Class' is still a factor
```

```
## 'data.frame': 1929 obs. of 6 variables:
## $ A01 : int 11 10 25 31 3 3 3 11 10 25 ...
## $ A18 : int 96 30 89 17 42 29 39 27 46 44 ...
## $ A22 : num 0.0605 0.0617 0.0601 0.0633 0.0626 ...
## $ A23 : int 15 137 107 1 110 111 116 165 107 16 ...
## $ A25 : num 0 0 0 0 0 0 0 0 0 0 ...
## $ Class: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 2 1 1 2 ...
```

```
set.seed(33085625)
new_train.row = sample(1:nrow(new_PD), 0.7*nrow(new_PD))
new_PD.train = new_PD[new_train.row,]
new_PD.test = new_PD[-new_train.row,]
```

The simple classifier that was chosen to be used here is the decision tree. Even though the decision tree does not have the best accuracy, decision trees are simple which means that they are easy to understand and interpret. It is also important to know that decision trees are able to capture non-linear correlations between features and the target variable without any additional work.

```
simple_classifier = tree(Class ~., data = new_PD.train)
predict_simple_classifier = predict(simple_classifier, newdata = new_PD.test, type = "class")

tbl_result = table(actual = new_PD.test$Class, predicted = predict_simple_classifier)
colnames(tbl_result) = c("legitimate", "phishing")
rownames(tbl_result) = c("legitimate", "phishing")
tbl_result
```

```
##           predicted
## actual    legitimate phishing
## legitimate    427         0
## phishing      152         0
```

```
simple_classifier_accuracy = (tbl_result[2, 2] + tbl_result[1, 1]) / (tbl_result[2, 2] + tbl_result[1, 1] + tbl_result[1, 2] + tbl_result[2, 1])
simple_classifier_accuracy
```

```
## [1] 0.7374784
```

From the confusion matrix displayed above in the output, it is presented that:

- The TP for the confusion matrix for decision tree is 0, indicating that the model did not correctly predict any phishing cases.
- The TN for the confusion matrix for Random Forest is 427, indicating that the model correctly identified 427 legitimate cases.
- The FP for the confusion matrix for decision tree is 0, indicating that the model did not incorrectly predict any legitimate cases as phishing.
- The FN for the confusion matrix for Random Forest is 152, indicating that the model incorrectly identified 152 phishing cases as legitimate.

Given the values from the confusion matrix above, the accuracy of the decision tree model is 0.7374784 which is approximately 73.74%. Previously, the decision tree model results from Question 4 and 5 have an accuracy of 0.7257019 which is approximately 72.57% which implies that this current decision tree have a better accuracy than the decision tree before with an accuracy increase of 0.0117765 which is around 1.17% increase in accuracy.

```

# The test data used in Question 3 is PD_na_free.test
predict_simple_classifier_q3test = predict(simple_classifier, newdata = PD_na_free.test, type = "class")

# Confusion Matrix
tbl_result_q3test = table(actual = PD_na_free.test$Class, predicted = predict_simple_classifier_q3test)
colnames(tbl_result_q3test) = c("legitimate", "phishing")
rownames(tbl_result_q3test) = c("legitimate", "phishing")
tbl_result_q3test

##           predicted
## actual    legitimate phishing
## legitimate    336         0
## phishing     127         0

simple_classifier_accuracy_q3test = (tbl_result_q3test[2, 2] + tbl_result_q3test[1, 1]) / (tbl_result_q3test[2, 2] + tbl_result_q3test[1, 1] + tbl_result_q3test[2, 1] + tbl_result_q3test[1, 2])
simple_classifier_accuracy_q3test

## [1] 0.7257019

```

From the confusion matrix displayed above in the output, it is presented that:

- The TP for the confusion matrix for decision tree is 0, indicating that the model did not correctly predict any phishing cases.
- The TN for the confusion matrix for Random Forest is 336, indicating that the model correctly identified 336 legitimate cases.
- The FP for the confusion matrix for decision tree is 0, indicating that the model did not incorrectly predict any legitimate cases as phishing.
- The FN for the confusion matrix for Random Forest is 127, indicating that the model incorrectly identified 127 phishing cases as legitimate.

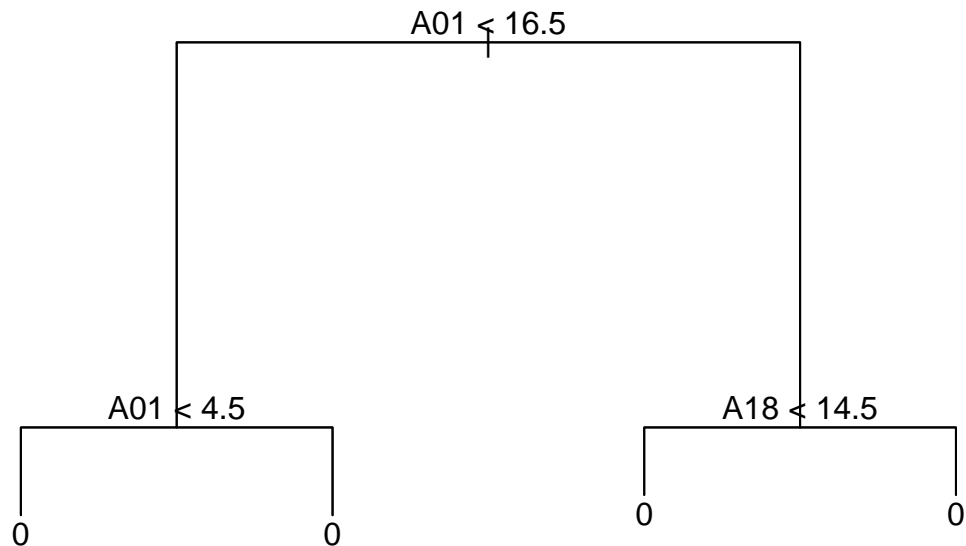
Given the values from the confusion matrix above, the accuracy of the decision tree model is 0.7257019 which is approximately 72.57%.

```

# Plot Decision tree
plot(simple_classifier)
text(simple_classifier, pretty = 0)
title(main = "Decision Tree Model Diagram")

```

Decision Tree Model Diagram



Confidence level on the test data used in Question 3

```
confidence_lvl_q3test = predict(simple_classifier, newdata = PD_na_free.test, type = "vector")
summary(confidence_lvl_q3test)
```

```
##           0           1
##  Min.    :0.5152   Min.    :0.04861
## 1st Qu.:0.5152   1st Qu.:0.20061
##  Median :0.7994   Median :0.20061
##   Mean  :0.7156   Mean    :0.28436
## 3rd Qu.:0.7994   3rd Qu.:0.48477
##   Max.  :0.9514   Max.    :0.48477
```

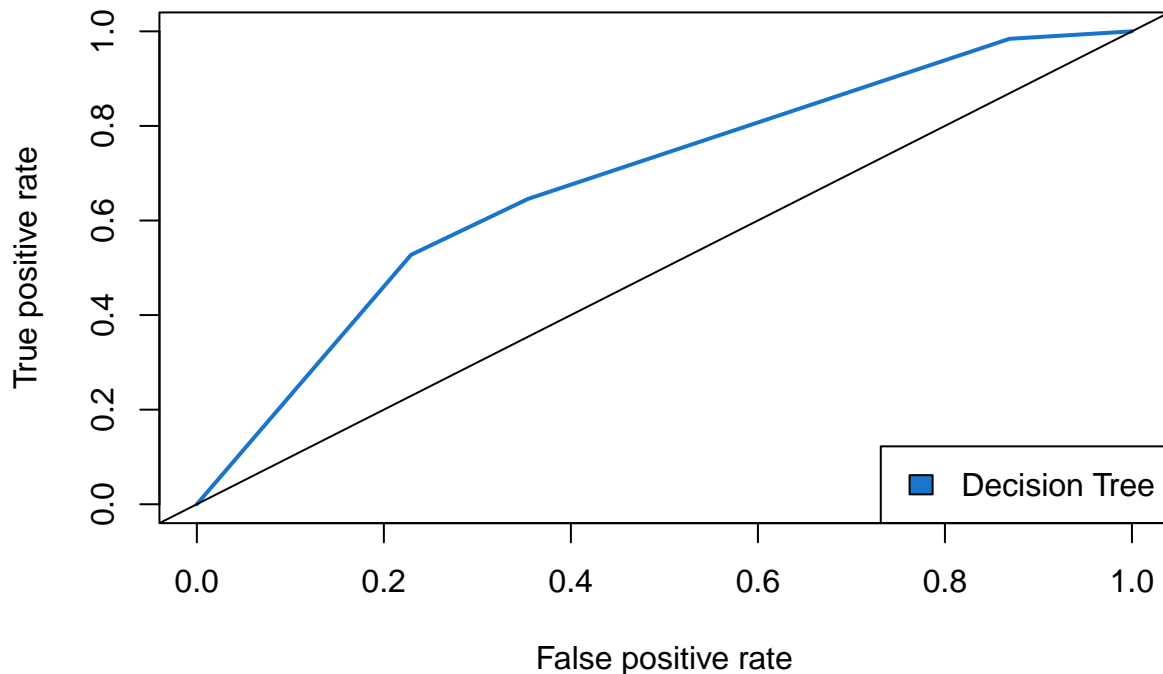
ROC curve

```
new_prediction_roc = prediction(confidence_lvl_q3test[, 2], PD_na_free.test$Class)
new_performance_roc = performance(new_prediction_roc, "tpr", "fpr")
plot(new_performance_roc, col = "dodgerblue3", main = "ROC Curve For Decision Tree Model", lwd = 2) + al
```

```
## integer(0)
```

```
legend("bottomright", legend = c("Decision Tree"), fill = c("dodgerblue3"))
```

ROC Curve For Decision Tree Model



The root node of the decision tree tests if A01 is indeed less than 16.5 then the outcome is Class 0 but if it is not then it further tests if A18 is less than 14.5. Depending on the outcome of A18 test, if A18 is less than 14.5 then the data is Class 0 but if more than 14.5 then the data is Class 1.

```
new_auc_decision_tree = performance(new_prediction_roc, "auc")
new_auc_decision_tree_num = as.numeric(new_auc_decision_tree@y.values)
new_auc_decision_tree_num
```

```
## [1] 0.6833052
```

Based on the Area Under the Curve (AUC) values calculated for the new decision tree, it is 0.6833052. The AUC value for the previous decision tree is 0.6578318. There is an increase of 0.0254734. This implies that the decision tree model have improved on distinguishing between Phishing and Legitimate classes.

Previously, the Bagging model stands out with the highest AUC value in the ranking with an AUC value of 0.6660222 now the new decision tree model have a higher auc value by 0.017283 which means that the variables selection did indeed improve the decision tree model.

Question 10

From Question 9, it is mentioned that the important predictors are basically A01, A18, A22, A23, and A25, hence these predictors are also chosen to be used here. Why A01, A18, A22, A23, and A25 are chosen as important predictors that can not be omitted is written at the start of Question 9.

In summary on why A01, A18, A22, A23, and A25 are chosen as important predictors is that although A01 didn't perform well in the Naïve Bayes model, its exclusive use in the Decision Tree model highlights its

importance. A25 and A22 are key predictors in the Naïve Bayes and Random Forest models, respectively, due to their high scores. Additionally, A18 and A23 are considered important in the Random Forest model because of their significant MeanDecreaseGini scores.

```
new_q10_PD = data.frame(PD)
new_q10_PD$Class = factor(new_q10_PD$Class)
new_q10_PD = subset(new_q10_PD, select = c(1, 18, 22, 23, 25, 26))
new_q10_PD = new_q10_PD[complete.cases(new_q10_PD),]
new_q10_PD = na.omit(new_q10_PD)

str(new_q10_PD) # Confirm after removal of NA values, the predictor 'Class' is still a factor
```

```
## 'data.frame': 1929 obs. of 6 variables:
## $ A01 : int 11 10 25 31 3 3 3 11 10 25 ...
## $ A18 : int 96 30 89 17 42 29 39 27 46 44 ...
## $ A22 : num 0.0605 0.0617 0.0601 0.0633 0.0626 ...
## $ A23 : int 15 137 107 1 110 111 116 165 107 16 ...
## $ A25 : num 0 0 0 0 0 0 0 0 0 0 ...
## $ Class: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 2 1 1 2 ...
```

```
set.seed(33085625)
new_q10_train.row = sample(1:nrow(new_q10_PD), 0.7*nrow(new_q10_PD))
new_q10_PD.train = new_q10_PD[new_q10_train.row,]
new_q10_PD.test = new_q10_PD[-new_q10_train.row,]
```

Since in Question 9, decision tree is used, for Question 10, the next best tree-based classifier would be random forest. Random Forest is a tree classification model that is particularly useful when there is complex relationships within the dataset and a model that can provide insights into the importance of different features is required. Random Forest is a method that combines the predictions from multiple decision trees to make more accurate and stable predictions than a single decision tree, so since the single decision tree in Question 9 has the highest model accuracy, this random forest tree-based classifier model would be an adequate choice for Question 10.

In addition to that, one more parameter was added to the random forest classifier, which is the parameter **importance**. The importance parameter refers to the feature importance and it was set to **TRUE**, which means that the importance of the predictors in the dataset should and would be assessed. This importance parameter help in understanding which features of the dataset are most important and influential in making predictions which in turn, can guide feature selection and model interpretation within the random forest.

```
simple_tree_classifier = randomForest(Class ~., data = new_q10_PD.train, importance = TRUE)
predict_simple_tree_classifier = predict(simple_tree_classifier, newdata = new_q10_PD.test, type = "class")

# Confusion Matrix
tree_tbl_result = table(actual = new_q10_PD.test$Class, predicted = predict_simple_tree_classifier)
colnames(tree_tbl_result) = c("legitimate", "phishing")
rownames(tree_tbl_result) = c("legitimate", "phishing")
tree_tbl_result
```

```
##           predicted
## actual    legitimate phishing
## legitimate    376      51
## phishing      113      39
```

```
simple_tree_classifier_accuracy = (tree_tbl_result[2, 2] + tree_tbl_result[1, 1]) / (tree_tbl_result[2, 2] + tree_tbl_result[1, 1] + tree_tbl_result[2, 1] + tree_tbl_result[1, 2])
```

```
## [1] 0.716753
```

From the confusion matrix displayed above in the output, it is presented that:

- The TP for the confusion matrix for decision tree is 39, indicating that the model did only correctly predict 39 phishing cases.
- The TN for the confusion matrix for Random Forest is 376, indicating that the model correctly identified 376 legitimate cases.
- The FP for the confusion matrix for decision tree is 51, indicating that the model predict only 51 legitimate cases as phishing.
- The FN for the confusion matrix for Random Forest is 113, indicating that the model incorrectly identified 113 phishing cases as legitimate.

Given the values from the confusion matrix above, the accuracy of the decision tree model is 0.716753 which is approximately 71.67%.

```
# Confidence level on the test data used in Question 3
```

```
confidence_lvl_q3testingdataset = predict(simple_tree_classifier, newdata = PD_na_free.test, type = "probabilities")
summary(confidence_lvl_q3testingdataset)
```

```
##           0           1
## Min.      :0.0620   Min.      :0.0000
## 1st Qu.:0.5510   1st Qu.:0.0680
## Median :0.8440   Median :0.1560
## Mean      :0.7299   Mean      :0.2701
## 3rd Qu.:0.9320   3rd Qu.:0.4490
## Max.      :1.0000   Max.      :0.9380
```

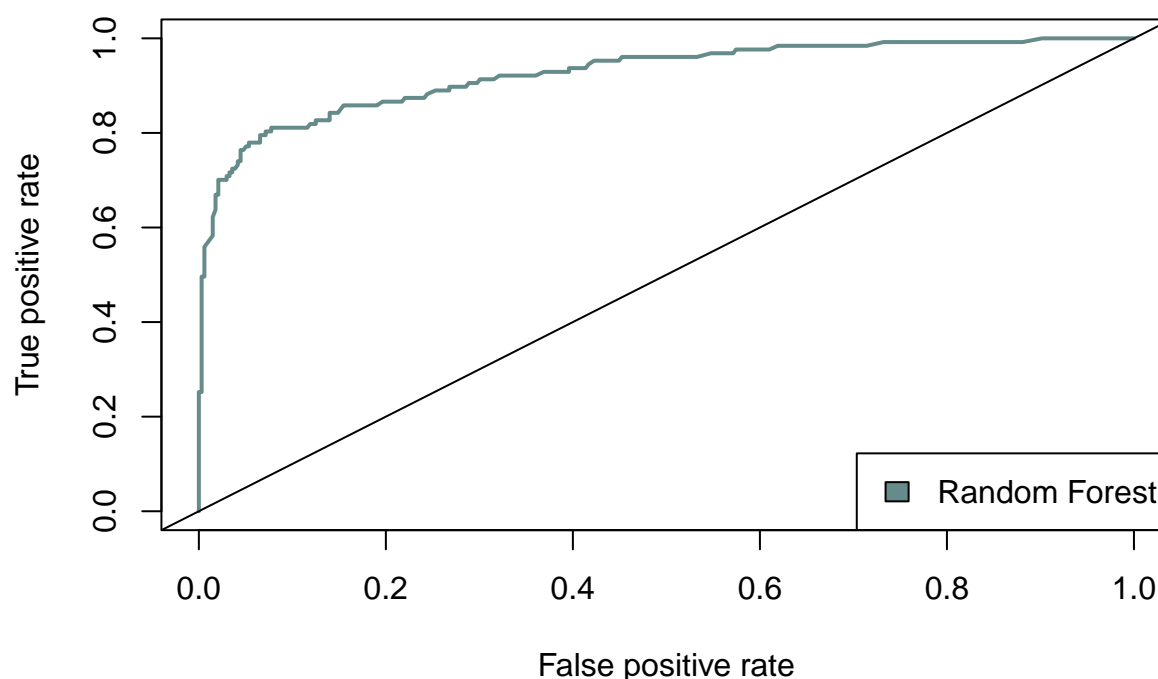
```
# ROC curve
```

```
tree_new_prediction_roc = prediction(confidence_lvl_q3testingdataset[, 2], PD_na_free.test$Class)
tree_new_performance_roc = performance(tree_new_prediction_roc, "tpr", "fpr")
plot(tree_new_performance_roc, col = "paleturquoise4", main = "ROC Curve For Random Forest Model", lwd = 2)
```

```
## integer(0)
```

```
legend("bottomright", legend = c("Random Forest"), fill = c("paleturquoise4"))
```


ROC Curve For Random Forest Model



```
new_auc_random_forest = performance(tree_new_prediction_roc, "auc")
new_auc_random_forest_num = as.numeric(new_auc_random_forest@y.values)
new_auc_random_forest_num
```

```
## [1] 0.9258296
```

Based on the Area Under the Curve (AUC) values calculated for the new random forest, it is 0.9258296. The AUC value for the previous random forest is 0.6404199. There is an increase of 0.2854097. This implies that the random forest model have improved on distinguishing between Phishing and Legitimate classes.

Previously, the new Decision tree model stands out with the highest AUC value in the ranking with an AUC value of 0.6833052 now the new random forest model have a higher auc value by 0.2425244 which means that the variables selection did indeed improve the random forest model significantly.

Question 11

```
# Neuralnet library is loaded here as it's functionalities clashes with "prediction" function in R while
# They both seem to clash as error occurs when I tried to use "prediction" function in R while "neuraln
library("neuralnet") # Library for Artificial Neural Network classifier
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':  
##  
##      compute
```

```
## The following object is masked from 'package:ROCR':  
##  
##      prediction
```

```
# 1. Remove rows containing Missing Values (NA)
```

```
PD$Class = factor(PD$Class)  
PD_q11 = PD[complete.cases(PD),]  
PD_q11 = data.frame(PD_q11)
```

```
# 2. Recode output class as numeric
```

```
PD_q11$Class = as.numeric(PD_q11$Class)
```

```
# 3. Create training and testing dataset
```

```
new_q11_train.row = sample(1:nrow(PD_q11), 0.7*nrow(PD_q11))  
PD_q11.train = PD_q11[new_q11_train.row,]  
PD_q11.test = PD_q11[-new_q11_train.row,]
```

```
PD_q11.train$Class1 = PD_q11.train$Class == 1
```

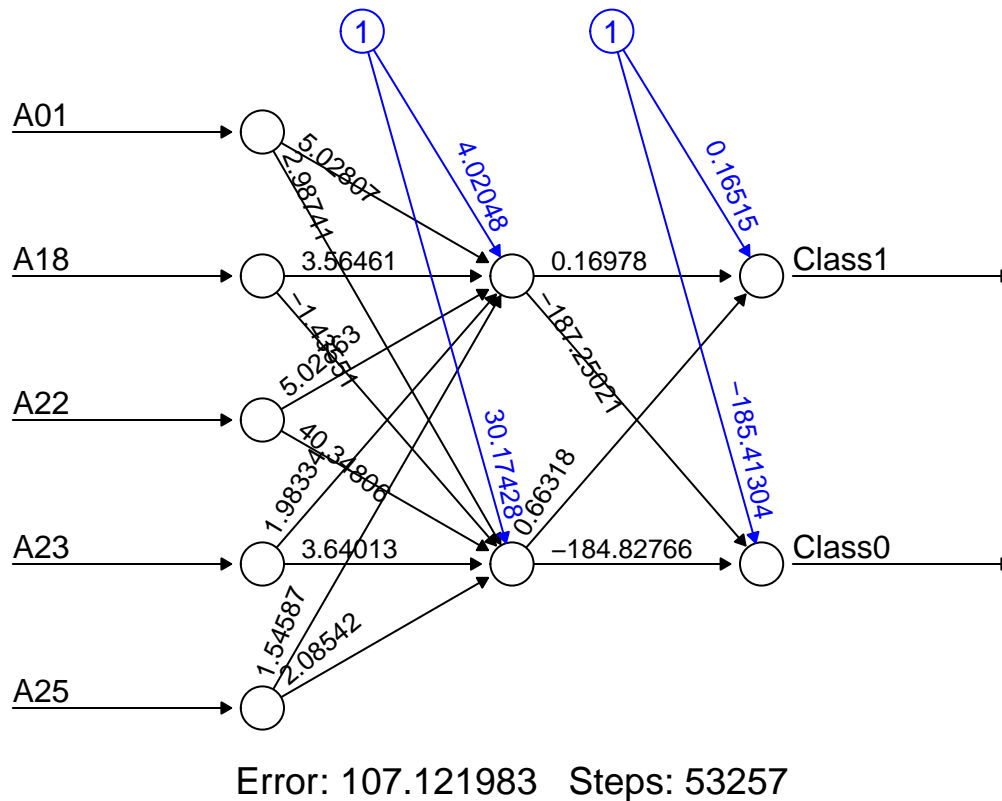
```
PD_q11.train$Class0 = PD_q11.train$Class == 0
```

```
# 4. Fit neural network and test accuracy
```

```
PD_q11.nn = neuralnet(Class1 + Class0 ~ A01 + A18 + A22 + A23 + A25, PD_q11.train, hidden = 2, linear.output = 1)
```

```
# Plot ANN
```

```
plot(PD_q11.nn, rep = "best")
```



```
# Plot Confusion Matrix
PD_q11.pred = compute(PD_q11.nn, PD_q11.test[c(1, 18, 22, 23, 25)])
PD_q11.pred_round = round(PD_q11.pred$net.result, 0)
df_PD_q11.pred_round = as.data.frame(as.table(PD_q11.pred_round))
df_PD_q11.pred_round_val = df_PD_q11.pred_round[!df_PD_q11.pred_round$Freq == 0,]

df_PD_q11.pred_round_val$Freq = NULL
colnames(df_PD_q11.pred_round_val) = c("Obs", "Class")

df_PD_q11.pred_round_val = df_PD_q11.pred_round_val[order(df_PD_q11.pred_round_val$Obs), ]

table(observed = PD_q11.test$Class, predicted = df_PD_q11.pred_round_val$Class)
```

```
##           predicted
## observed    A    B
##           1 334    0
##           2 129    0
```

Firstly, I made sure that 'Class' is a factor before remove rows containing Missing Values (NA) with `complete.cases` so it would not affect the output of the Artificial Neural Network (ANN). Once its done, the dataset will be transformed to a dataframe just in case it wasn't beforehand.

Previously, it was noted down that all attributes are num or int datatypes but just to make sure, `as.numeric` was used on Class attribute and the rest of the attributes within the dataset.

Once that was done, the training and testing dataset was created and was fit onto the ANN and the accuracy was tested. This classifier is pretty accurate compared to the other classifiers. ANNs are capable of

learning and modeling complex non-linear relationships which many traditional algorithms cannot capture as effectively which made it very adaptable and hence produce high accuracy results. ANNs can do feature extraction which allows it to automatically detect the important features without any human intervention.

Methods like decreasing or increasing the hidden layers values and decreasing or increasing important predictors and/or attributes to feed into the ANN couldn't reduce the error further than the one shown above.

Referring to the confusion matrix obtain above, Observed represents the actual class labels from the dataset whereas Predicted represent the class labels as predicted by the ANN. Class A and B are Class 0 and 1 which are the two classes that the ANN is trying to predict.

According to the confusion matrix:

- True Positives (TP) for Class A shows that 336 instances were correctly predicted as belonging to Class A.
- False Negatives (FN) for Class A shows that 0 instances were incorrectly predicted as not belonging to Class A when they actually did.
- False Positives (FP) for Class B shows that 127 instances were incorrectly predicted as not belonging to Class B when they actually did.
- True Negatives (TN) for Class B shows that The matrix does not provide this information, but it would be the number of instances correctly identified as not belonging to Class B.

From the results of the confusion matrix, it implies that the ANN appears to have classified all cases as Class A and none as Class B. This might suggest issues with the model, like a lack of recognition for Class B or Class B being very challenging to predict, or could be due to an uneven distribution of data where Class A is more dominant.

Question 12

```
# Neuralnet library is detached here as it's functionalities clashes with "prediction" function in R wh
detach("package:neuralnet", unload=TRUE)
```

```
# caret library is used here for k-nearest neighbors (KNN) model
knn = knn3(Class ~., data = PD_na_free.train, k = 10)
knn_prediction = predict(knn, newdata = PD_na_free.test, type = "class")

# Confusion Matrix
knn_result = table(actual = PD_na_free.test$Class, prediction = knn_prediction)
colnames(knn_result) = c("legitimate", "phishing")
rownames(knn_result) = c("legitimate", "phishing")
knn_result
```

```
##           prediction
## actual      legitimate phishing
## legitimate      314       22
## phishing        98       29
```

```
# Accuracy
knn_accuracy_q5 = (knn_result[2, 2] + knn_result[1, 1]) / (knn_result[2, 2] + knn_result[1, 1] + knn_res
knn_accuracy_q5
```

```
## [1] 0.7408207
```

From the knn confusion matrix displayed above in the output, it is presented that:

- The TP for the confusion matrix for knn is 312, indicating that the number of legitimate cases correctly predicted as legitimate, which is 312.
- The TN for the confusion matrix for knn is 28, indicating that the number of phishing cases correctly predicted as phishing is 28.
- The FP for the confusion matrix for knn is 24, indicating that the number of phishing cases incorrectly predicted as legitimate is 24.
- The FN for the confusion matrix for knn is 99, indicating that the number of legitimate cases incorrectly predicted as phishing is 99.

Given the values from the confusion matrix above, the accuracy of the decision tree model is 0.7343413 which is approximately 73.43%. This means that the KNN model correctly predicted the class is legitimate or phishing for about 73.43% of the cases in the test dataset.

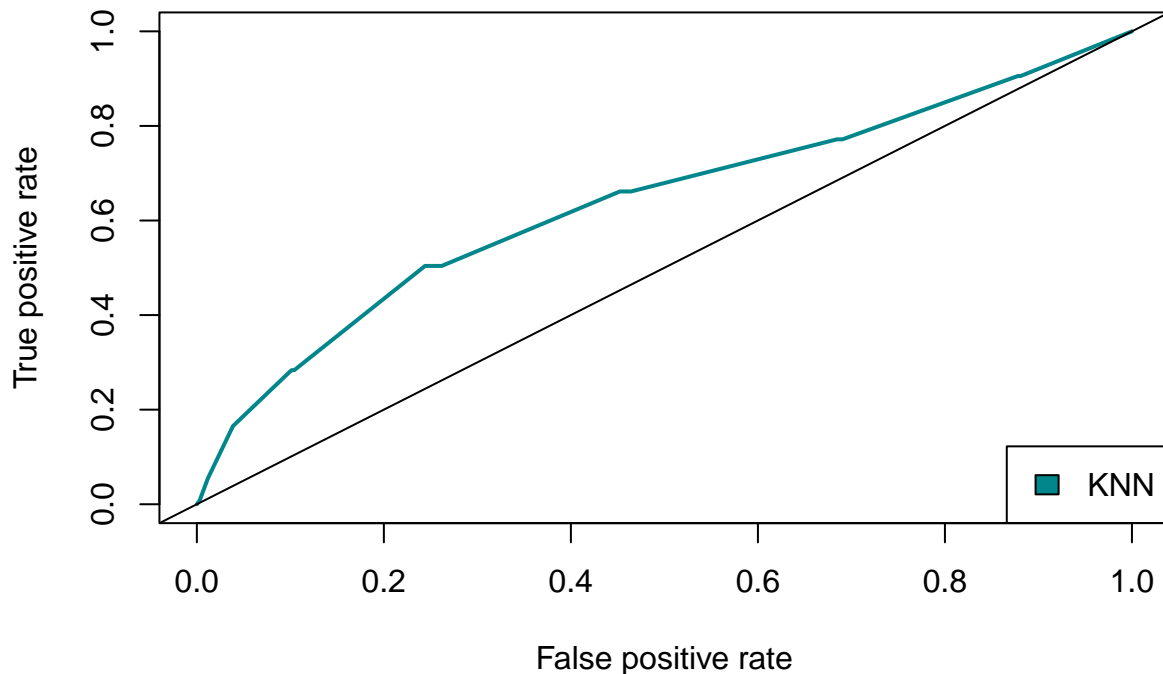
```
# Confidence level
knn_confidence = predict(knn, newdata = PD_na_free.test, type = "prob")

# ROC curve
prediction_knn_roc = prediction(knn_confidence[, 2], PD_na_free.test$Class)
performance_knn = performance(prediction_knn_roc, "tpr", "fpr")
plot(performance_knn, col = "turquoise4", main = "ROC Curve For KNN model", lwd = 2) + abline(0, 1)

## integer(0)

legend("bottomright", legend = c("KNN"), fill = c("turquoise4"))
```

ROC Curve For KNN model



The ROC Curve above shows that the **turquoise line** starts at the origin (0,0) and curves towards the top right corner (1,1), indicating an increasing TPR as the FPR increases.

```
auc_knn = performance(prediction_knn_roc, "auc")
auc_knn_num = as.numeric(auc_knn@y.values)
auc_knn_num
```

```
## [1] 0.6367993
```

The area under the curve (AUC) calculated above represents the knn model's ability to distinguish between the legitimate or phishing classes. A larger AUC that is close to 1 indicates that the model have a better performance. The AUC value of 0.6367993 suggests that there is a 63.68% chance that the model will be able to correctly distinguish a random positive case from a negative one.

```
knn_accuracy_q6 = performance(prediction_knn_roc, "acc")
knn_accuracy_q6_num = as.numeric(max(knn_accuracy_q6@y.values[[1]]))
knn_accuracy_q6_num
```

```
## [1] 0.7429806
```

The value 0.7429806 calculated above represents the highest accuracy achieved by the KNN model.

```
comparison_with_knn = rbind(comparison_tbl, list(knn_accuracy_q5, knn_accuracy_q6_num, (knn_accuracy_q5
rownames(comparison_with_knn)[rownames(comparison_with_knn) == "1"] = "KNN"
comparison_with_knn
```

##	Question 5 Model Accuracy	Question 6 Model Accuracy
## Decision Tree	0.7257019	0.7257019
## Naïve Bayes	0.7257019	0.7321814
## Bagging	0.7365011	0.7408207
## Boosting	0.6717063	0.7300216
## Random Forest	0.7300216	0.7365011
## KNN	0.7408207	0.7429806
##	Average Model Accuracy	
## Decision Tree	0.7257019	
## Naïve Bayes	0.7289417	
## Bagging	0.7386609	
## Boosting	0.7008639	
## Random Forest	0.7332613	
## KNN	0.7419006	

The KNN model shows a consistent performance with an accuracy of 0.7343413 in Question 5 and 0.7429806 in Question 6 in the table above. The average accuracy of the KNN model is 0.7386609, which is calculated by taking the average of the accuracy from Question 5 and 6. When compared to other models, the KNN model ranks among the top prediction models in average accuracy. The KNN model accuracy is equal to Bagging and have a higher accuracy than Decision Tree, Naïve Bayes, Boosting, and Random Forest models. The stability of the KNN model remains consistent from Question 5 to 6, indicating that the KNN model is reliable to perform under various data sets or conditions.