FIT3152
ASSIGNMENT 2
CLASSIFICATION MODELS REPORT

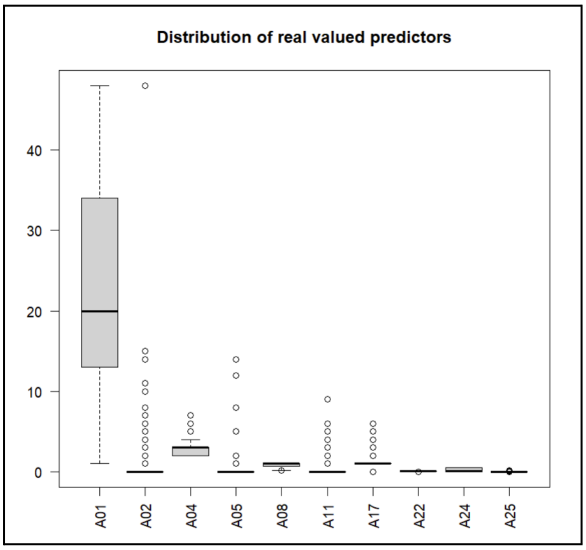
BY:
ELKE IE | 33067902

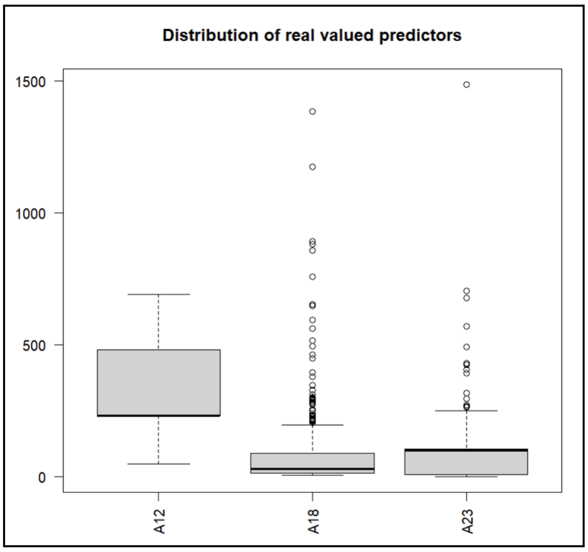
Introduction

This analysis is conducted to test how different classification models in R perform when predicting whether a website is legitimate or designed for phishing from the data provided. Further optimisation and exploration of new classification models are also conducted in this analysis.

Question 1

There are 718 phishing data while there are 1282 legitimate data(output 1). Output 2 says that all attributes are all numeric and as we can see in output 3, the real valued attributes are columns 1, 2, 4, 5, 8, 11, 12, 17, 18, 22, 23, 24, and 25 because other columns only contain 0/1/NA which mean they are most probably a categorical attributes with yes/no answer.





Above is the distribution of real valued attributes splitted into 2 plots to make it more visible due to the different range of values. As we can see, columns 12, 18 and 23 have a relatively large range of values compared to the others. Columns 2, 4, 5, 8, 11, 17, 22, 24, and 25 are dominated by a relatively small value of just 5 and below. Column 1 is dominated by values around 12 to 34 and columns 18 and 23 are dominated by values from 0 to 100.

After observing the distribution of all columns(output 4), we can see that the values columns 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15, 16, 17, 19, 20, 21, 24, and 25 are dominated by just 1 value, these columns can be considered to be omitted from our data since they might not give valuable information for the model to be able to differentiate the dependent variable. But we will not omit any variables before fitting it to the models in the following questions to reduce the risk of underfitting, we will do this only when we are improving the initial models later.

Question 2

Before fitting the data into the model, we need to omit rows with NA values, this is because NA values are actually incomplete data and this is considered as noise that might make the training process of classification models inaccurate and reduce the model's performance. Next we also need to change the data type of the dependent variable into factor, because most of the models we use expect the dependent variable to be in factor data type and this will also make it clear that our dependent variable is a class.

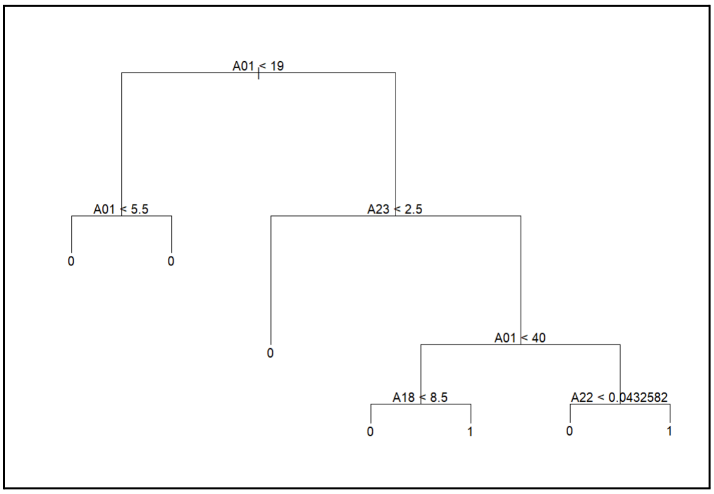
Question 3

(In R code appendix below)

Question 4

(In R code appendix below)

Decision tree plot:



Question 5

- Decision tree

```
> tree.table
       predicted
actual    0   1
     0  244  57
     1   53 114
```

```
> tree.acc
[1] 76.49573
```

Accuracy: 76.50%

- Naive Bayes

```
> naive.table
      predicted
actual   0   1
     0  93 208
     1  18 149
```

```
> naive.acc
[1] 51.7094
```

Accuracy: 51.71%

- Bagging

```
> bag.predict$confusion
               Observed Class
Predicted Class   0   1
              0 269  78
              1  32  89
```

```
> bagging.acc
[1] 76.49573
```

Accuracy: 76.50%

- Boosting

```
> boost.predict$confusion
               Observed Class
Predicted Class   0   1
              0 258  77
              1  43  90
```

```
> boost.acc
[1] 74.35897
```

Accuracy: 74.36%

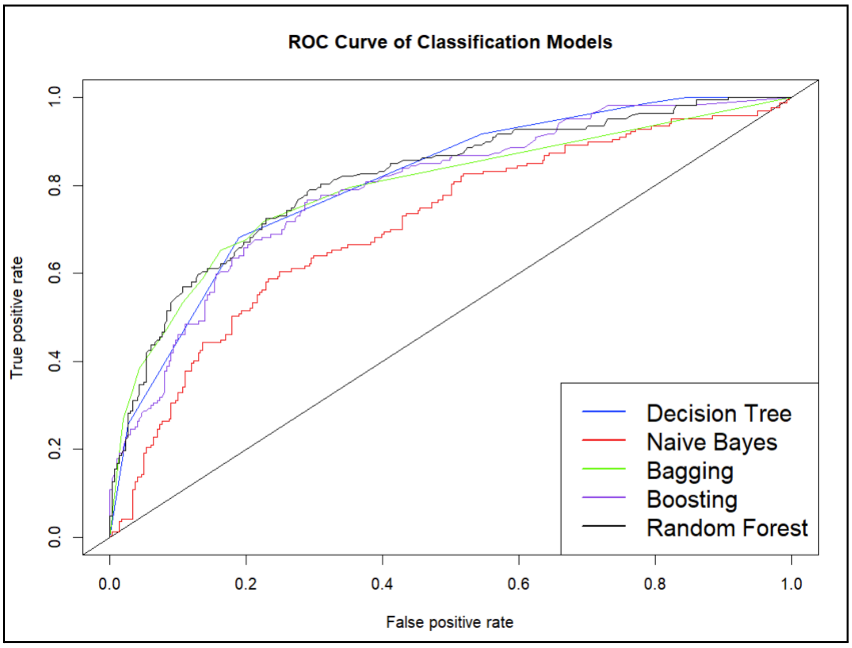
- Random Forest

```
> rf.table
        predicted
observed   0   1
       0 269  32
       1  73  94
```

```
> rf.acc
[1] 77.5641
```

Accuracy: 77.56%

Question 6



- Decision tree AUC value

```
> tree_auc
[1] 0.8071498
```

- Naive Bayes AUC value

```
> naive_auc
[1] 0.7100782
```

- Bagging AUC value

```
> bagging_auc
[1] 0.7926174
```

- Boosting AUC value

```
> boosting_auc
[1] 0.7927268
```

- Random Forest AUC value

```
> rf_auc
[1] 0.8128096
```

Question 7

```
> performance
          Model Accuracy       AUC
1          Tree 76.49573 0.8071498
2   Naive Bayes 51.70940 0.7100782
3       Bagging 76.49573 0.7926174
4      Boosting 74.35897 0.7927268
5 Random Forest 77.56410 0.8128096
```

The above output shows us that Random Forest is the single best classifier we have made out of the data provided since it has the highest accuracy and AUC value, meaning it has the highest performance. And naive bayes is the classifier with the worst performance.

Question 8

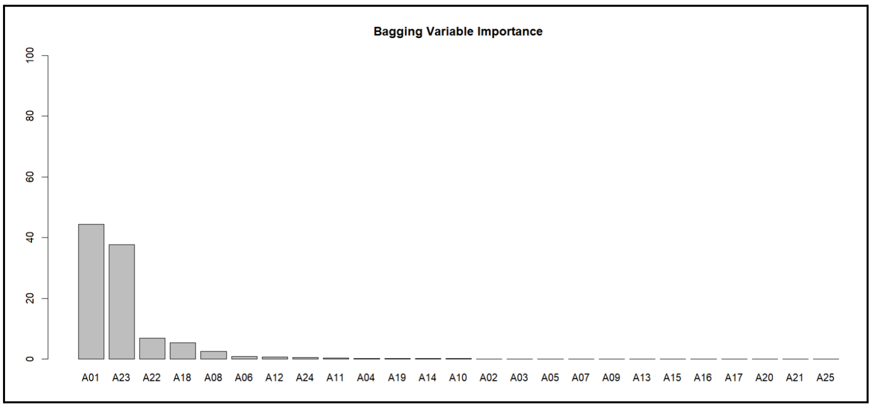
We will be analysing the most important attributes in the decision tree, bagging, boosting, and random forest. We will not check for naive bayes because variable importance is not a concept in naive bayes, as each variable contributes independently.

Decision tree variables:

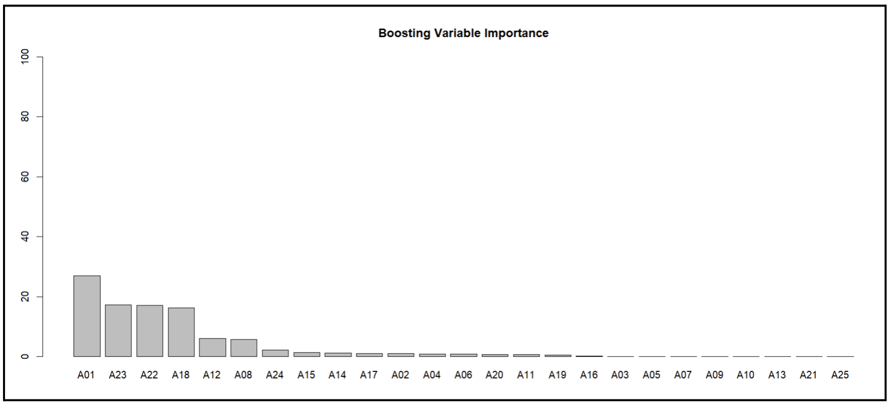
```
> summary(tree.fit)

Classification tree:
tree(formula = Class ~ ., data = PD.train)
Variables actually used in tree construction:
[1] "A01" "A23" "A18" "A22"
Number of terminal nodes:  7
Residual mean deviance:  0.9484 = 1028 / 1084
Misclassification error rate: 0.2236 = 244 / 1091
```

The above output shows that the decision tree uses A01, A23, A18, and A22.

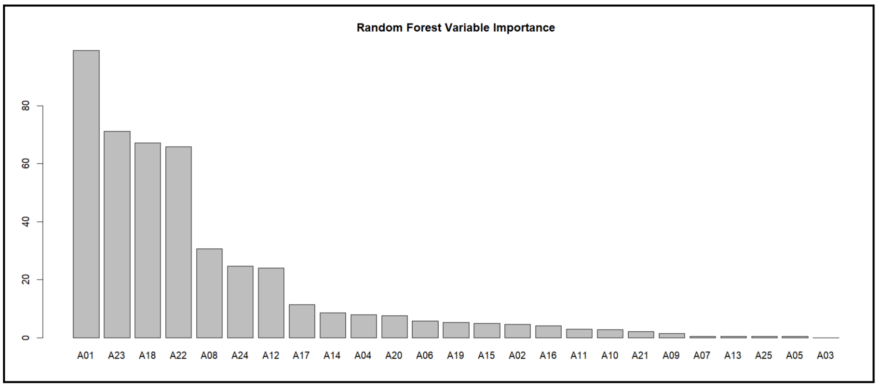
Bagging variable importance:



Boosting variable importance:



Random Forest variable importance:



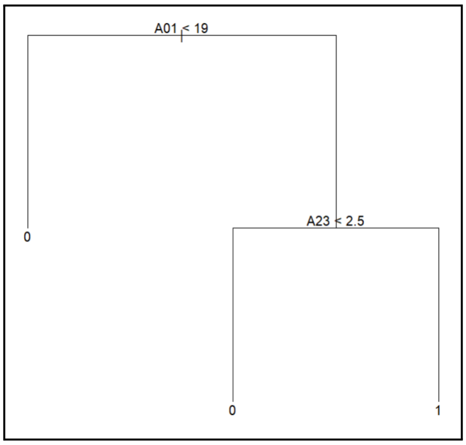
Based on the output above, we can see that variables A01, A23, A18, and A22 are on the top 4 of all the models, so we can say that they are the important attributes, but the most important will be A01 since it always has the highest importance in all models.
We can also see that A08 is always on the top 6 most important attributes, so we can still say that it might have an effect on the models performance but not significant, in the other hand, all other attributes other than that (A02, A03, A04, A05, A06, A07, A09, A10, A11, A12, A13, A14, A15, A16, A17, A19, A20, A21, A24, A25) are considered not really important and can be omitted from the data with little effect on the models performance because they have little effect on all the models we have created so far and we can consider them as not significant. And it seems that the variables that can be omitted mentioned in question 1 are in the list of not important attributes here, so our observation in question 1 is quite accurate.

Question 9

We will be using our decision tree to make the simple model, because a single tree with little branches will be easy for people to track the outcome by hand rather than many trees such as bagging/boosting/random forest, or calculation such as naive bayes. The important factor we must note is that the attributes used must be in the list of important attributes in question 8, and the tree must only have around 2 branches to make it simple. We will use the initial tree we have created in question 4 because all the attributes used in this tree are the important attributes we mentioned in question 8, therefore we can use it and just need to prune them into just having around 2 branches.

```
> cvtest = cv.tree(tree.fit, FUN = prune.misclass)
> cvtest
$size
[1] 7 6 5 3 1

$dev
[1] 257 257 258 269 392

$k
[1] -Inf    0    4   10   62

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

The output above shows that the tree with 3 terminal nodes which will make a simple tree doesn't increase the number of misclassifications significantly, so we will prune the tree to this number of terminal nodes.



Simple tree plot

As we can see in the tree plot above, attributes A01, and A23 are used in this simple decision tree and these attributes are the top 2 most important attributes mentioned in question 8, furthermore the tree is a smaller part of the original tree and it only has 2 branches which means it is simple.

```
> treeSimple.table
       predicted
actual   0   1
     0 227  74
     1  51 116
```

Confusion matrix

```
> treeSimple.acc
[1] 73.2906
```

Accuracy

```
> tree_auc_s
[1] 0.7139376
```

AUC

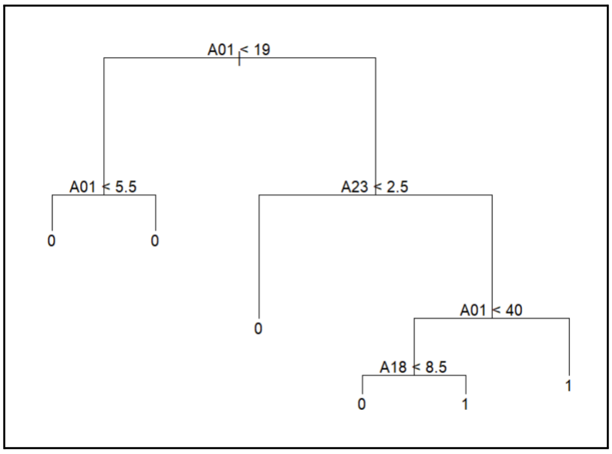
After calculating the accuracy and AUC, we can see, the accuracy and AUC is not bad compared to the models in question 4, it performs better than naive bayes, but it does perform less than the original decision tree but not extreme (output 5). In conclusion, we succeed in making a simple tree with decent performance.

Question 10

The tree based classification we will be improving is the decision tree, because there will be more room for improvement in a single decision tree, since a single decision tree technically did not learn the train data complexly compared to bagging/boosting/random forest therefore it can be improved further. The important factor we need to consider is that the attributes used must be in the list of important attributes from question 8 and we will try to reduce the rate of overfitting so that it can predict unknown data well.

For Improving the decision tree, we will adjust the “mincut” parameter. This parameter basically indicates the minimum number of observations to create a branch in the tree. By adjusting this to 15, we ensure that each branch/node has a minimum of 15 observations and this will prevent overfitting with the training data and hence might improve our decision tree for the unknown data.

```
> summary(treeImproved.fit)

Classification tree:
tree(formula = Class ~ ., data = PD.train, mincut = 15)
Variables actually used in tree construction:
[1] "A01" "A23" "A18"
Number of terminal nodes:  6
Residual mean deviance:  0.964 = 1046 / 1085
Misclassification error rate: 0.2273 = 248 / 1091
```



Improved Decision Tree Plot

The output above shows us that A01, A23, and A18 are used in this improved decision tree which means it uses the important attributes mentioned in question 8.

```
> treeImproved.table
        predicted
actual    0   1
     0 243  58
     1  51 116
```

Confusion matrix

```
> treeImproved.acc
[1] 76.7094
```

Accuracy

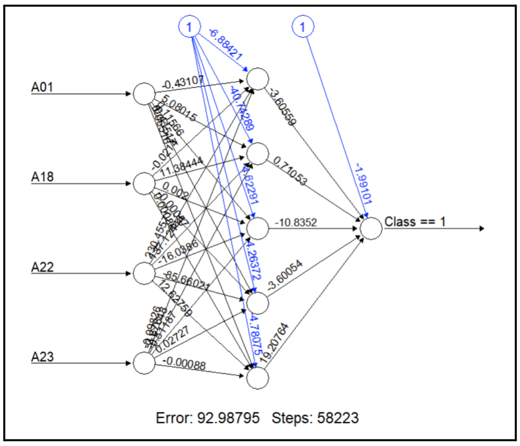
```
> tree_auc_i
[1] 0.8139236
```

AUC

After analysing the performance measure, we can see that both accuracy and AUC of the decision tree improved from the original tree which was previously have an accuracy of 76.49 and AUC of 0.8071, and the AUC is now the highest out of all the classifiers in question 4, but the accuracy is still slightly lower than the random forest (output 5). We can conclude that either the random forest or this single improved decision tree is the best tree based classifier we have now.

Question 11

As we have discussed in question 8, the important attributes are A01, A23, A18, and A22, so that will be the attributes we use to make the neural network model. The only preprocessing required is to make a 8:2 training to testing data ratio instead of 7:3, because neural networks can learn complex data in detail, therefore providing more training data will prevent overfitting and increase the performance. Then we also set the hidden parameter to 8 and stepmax to 1e7, this means that there will be 8 hidden layers and a maximum of 1e7(10 million) iterations in the ANN. We set the stepmax at a high value so that the model can converge within this stepmax, else it might not reach convergence in the stepmax default value and the model doesn't work. Below is what the ANN model looks like.



ANN plot

```
> net.table
        predicted
observed   0   1
       0 171  30
       1  49  62
```

Confusion matrix

```
> net.acc
[1] 74.67949
```

Accuracy

```
> net_auc
[1] 0.7840527
```

AUC

After analysing the performance, we can see that compared to the models in question 4, the ANN classifier is better than naive bayes and comparable to boosting, but not better than decision tree, bagging, and random forest (output 5). This performance result may be due to the data we are dealing with, the data might not be very complex for ANN to perform better than other tree based classifier because if the data is relatively simple, tree based classifier which can capture simple decision boundary well might perform better than classifier that is made for complex data such as ANN.

Question 12

We will be creating a Support Vector Machine(SVM) as our new classifier. This command can be found under the e1071 R-package (note: using SVM under e1071 package is permitted by the lecturer). SVM can be used for both regression and classification, and it works by creating a hyperplane that will separate each class. SVM can be used for linear and non-linear classification which makes it flexible, we adjust this by changing the 'kernel' parameter into 'linear' for linear classification or 'poly', 'radial', etc for non-linear classification. Since I have tried using 'linear', 'poly' and 'radial' for the kernel and 'radial' gives the best performance, we will use this kernel and that means our classification problem is non-linear. And to note, the attributes used in this model are only the important attributes we found in question 8 which are A01, A18, A22, and A23. And those attributes are preprocessed by being scaled because SVM is sensitive to the scale of features and it might prioritise features with higher scale if done without scaling.

```
> svm.table
   svm.pred
      0   1
  0 232  69
  1  69  98
```
Confusion matrix

```
> svm.acc
[1] 70.51282
```
Accuracy

```
> svm_auc
[1] 0.7360097
```
AUC

After analysing the performance of the radial SVM classifier, we can see that the performance is comparable with the models we have in question 4 and 11, it exceeds naive bayes but not better than a single decision tree, bagging, boosting, random forest, and ANN (output 5).

Package detail:
https://cran.r-project.org/web/packages/e1071/index.html
https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf

Conclusion

In conclusion, the order of classifiers from best to worst out of all the classifiers we have made in this analysis is :

1. Optimised decision tree
2. Random forest
3. Original decision tree
4. Bagging
5. Boosting
6. ANN
7. SVM
8. Naive bayes

We managed to improve a decision tree by adjusting the “mincut” parameter, and made a simpler decision tree by pruning it into just 2 branches. We also found out the important variables needed to classify the data into legitimate and phishing then using it for the ANN and SVM models to give them a better performance.

APPENDIX

OUTPUT 1

```
> phishing = sum(PD$Class == 1)
> phishing
[1] 718
> legit = sum(PD$Class == 0)
> legit
[1] 1282
```

OUTPUT 2

```
> str(PD)
'data.frame':	1559 obs. of  26 variables:
 $ A01  : int  13 18 20 46 34 18 13 48 18 20 ...
 $ A02  : int  0 0 0 0 0 0 0 0 0 1 ...
 $ A03  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A04  : int  2 3 3 3 3 3 3 3 3 2 ...
 $ A05  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A06  : int  0 0 1 0 0 0 0 0 0 1 ...
 $ A07  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A08  : num  0.867 0.524 0.643 1 1 ...
 $ A09  : int  0 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  648 232 232 232 232 232 232 232 633 648 ...
 $ A13  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A14  : int  0 0 0 1 1 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  2 1 1 1 1 1 1 2 1 2 ...
 $ A18  : int  17 63 96 99 55 36 31 8 5 20 ...
 $ A19  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A20  : int  0 0 0 0 0 1 0 0 0 1 ...
 $ A21  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A22  : num  0.0583 0.0463 0.0477 0.0655 0.0565 ...
 $ A23  : int  112 100 1 41 28 102 111 0 100 6 ...
 $ A24  : num  0.0286 0.5229 0.5229 0.5229 0.5229 ...
 $ A25  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Class: Factor w/ 2 levels "0","1": 1 1 2 1 2 1 2 1 1 1 ...
 - attr(*, "na.action")= 'omit' Named int [1:441] 2 5 9 11 13 18 27 34 37 39
...
  ..- attr(*, "names")= chr [1:441] "61104" "25817" "61874" "84707" ...
```

OUTPUT 3

```
> # See the unique values in each columns
> unique_values <- lapply(PD, unique)
> unique_values
$A01
 [1] 13 48 18 20  1 46 34 30 31 10

$A02
 [1]  0  1 NA  2  6  5 48  3  8 10 15  7  4 11 14
```

```
$A03
[1]  0 NA  1

$A04
[1]  2  3 NA  4  5  6  7

$A05
[1]  0 NA  1 12  5  2  8 14

$A06
[1]  0 NA  1

$A07
[1]  0  1 NA

$A08
  [1] 0.8667 1.0000 0.5238 0.6429     NA 0.3913 0.5000 0.8182 0.6000 0.7857
0.8421 0.8095 0.8462
 [14] 0.5333 0.7273 0.6538 0.7000 0.6667 0.8333 0.5714 0.8800 0.5357 0.5833
0.6364 0.6471 0.6250
 [27] 0.8125 0.5758 0.2667 0.8000 0.6957 0.4118 0.3750 0.7778 0.8889 0.7727
0.6500 0.5455 0.7368
 [40] 0.4706 0.3396 0.7333 0.3636 0.8571 0.6923 0.5152 0.6410 0.3103 0.1944
0.7200 0.4286 0.9091
 [53] 0.4167 0.3824 0.8750 0.5897 0.7500 0.5385 0.5909 0.3077 0.8824 0.4783
0.5294 0.2727 0.4074
 [66] 0.7917 0.6071 0.5185 0.6061 0.6596 0.3333 0.5556 0.5172 0.7692 0.6842
0.6154 0.7179 0.7222
 [79] 0.4737 0.8947 0.8500 0.3448 0.4000 0.3438 0.3256 0.4615 0.3778 0.9062
0.5143 0.3111 0.7143
 [92] 0.4048 0.2895 0.4565 0.4242 0.2353 0.3000 0.6562 0.4091 0.7647 0.5625
0.4500 0.4348 0.4545
[105] 0.4667 0.6875 0.3226 0.4688 0.4194 0.3478 0.3600 0.6087 0.6452 0.9231
0.8235 0.3214 0.3947
[118] 0.7941 0.5312 0.7619 0.5417 0.5217 0.5667 0.7059 0.5600 0.4800 0.3846
0.6316 0.4762 0.1739
[131] 0.3714 0.4375 0.2222 0.6800 0.4054 0.3810 0.3030 0.7826 0.5652 0.8636
0.6400 0.4643 0.4821
[144] 0.4211 0.2683 0.6944 0.9000 0.8929 0.6818 0.3830 0.6111 0.5526 0.5200
0.8846 0.9167 0.4444
[157] 0.2154 0.4390 0.4138 0.8611 0.4815 0.5349 0.5862 0.9286 0.5429 0.6176
0.5778 0.5789 0.4878
[170] 0.2903 0.3529 0.3684 0.3514 0.3590 0.2439 0.8036 0.5405 0.6765 0.7083
0.4333 0.2778 0.6857
[183] 0.1750 0.7949 0.3182 0.1852 0.2542 0.2647 0.3019 0.9130 0.9048 0.4857
0.3784 0.4848 0.2319
[196] 0.5946 0.5882 0.2963 0.6585 0.4474 0.4839 0.6190 0.2899 0.8387 0.2045
0.2857 0.2128 0.2273
[209] 0.4412 0.7407 0.1471 0.4400 0.7297 0.6522 0.3043 0.5263 0.6129 0.3704
0.2973 0.5484 0.7027
[222] 0.6279 0.6774 0.5500
```

```
$A09
[1]  0  1 NA

$A10
[1]  0  1 NA

$A11
[1]  0  1 NA  3  4  2  5  6  9

$A12
  [1] 648 504 232 482 227 633 451 306 180 133 388 335 576 554 377 473 210 673
365 553 274 212 224
 [24] 190  NA 142 572 304 383 492 255 677 613 283 491 641 431 281 379 432 223
686 418 522 259 573
 [47] 369 293 171 449 579 599 317 135 444 337 483 551 272 362 497 310 501 578
692 366 637 371 507
 [70] 141 456 629 678 420 419 535 521 533 338 189 158 487 363 650 226 253 644
361 170 645 389 129
 [93] 372 474 675 360 643 499 364 433 219 646 130 139 625 352 422 647 398 443
595 664 122 374 252
[116] 123 149 278 205 515 204 502 615  48

$A13
[1]  0 NA  3 12 15  6 24

$A14
[1]  0  1 NA

$A15
[1]  0  1 NA

$A16
[1]  0  1 NA

$A17
[1]  2  1  0 NA  3  4  5  6

$A18
  [1]   17   NA   63   96   99   55   36   29   31   22    8   75    5   20
42  111   14    6
 [19]   21   10   89   87  140  449   37   69   19    7   11   15   62  110
13   16   44   90
 [37]   46  190   76  154    9   61   98   65   68  516   25  163   28   32
93  130  134   27
 [55]   92   88   33   80   50  653   12   57  119  157   24  275   23   48
53  173   72  114
 [73]  117  226   66  165  158   41   86  100  125   47  102   81   83   39
78   38   77  301
 [91]   91  116  129  126   40  295  159  191   43  141   59  857   26  213
79  121   30   97
[109]  109  170  113  101   70  164   95   60  138   85  136   34   52   35
278  142  153   58
```

```
[127]  106   18  515   73  216  179   64  120  181  162  155   51  131  112
84  107  166  233
[145]  137  182   45  250   82  122  118  145  379  283  104  161   54  219
56  115  168  195
[163]  105  394   94  172  151  293   49  124   74 1386  256   71  177  144
239  143  347  288
[181]  139  494  463  132  327  108  148  203  562  313  892   67  299  647
127  146 1176  758
[199]  135  103  594  149  147  156  329  123  221  192  171  133  211  882
230  188

$A19
[1]  0  1 NA

$A20
[1]  0  1 NA

$A21
[1]  0  1 NA  2

$A22
   [1] 0.05826 0.06407 0.04629 0.04767 0.06962 0.06549 0.05645 0.05891
0.02919 0.05438 0.07046
  [12] 0.05932 0.06534 0.04234 0.05587 0.06170 0.07270 0.05708 0.03194
0.06108 0.06003 0.05723
  [23] 0.05852 0.06310 0.06421 0.06200 0.04408 0.06548 0.04096 0.07054
0.06119 0.06582 0.04829
  [34] 0.04661 0.05592 0.05991 0.06439 0.03133 0.02855 0.04802 0.05949
0.05990 0.05768 0.04753
  [45] 0.04030 0.05148 0.05745 0.06233 0.05877 0.02298      NA 0.06798
0.03755 0.05314 0.06161
  [56] 0.05721 0.05594 0.05546 0.05665 0.06149 0.06986 0.06161 0.07626
0.04682 0.06183 0.05742
  [67] 0.05552 0.05561 0.06356 0.04212 0.03276 0.03671 0.05607 0.05630
0.06851 0.06805 0.06010
  [78] 0.04110 0.05620 0.03864 0.06106 0.06446 0.04300 0.05340 0.05592
0.07142 0.05117 0.05895
  [89] 0.06057 0.03903 0.05557 0.04533 0.06412 0.05533 0.06490 0.03868
0.06002 0.05687 0.05811
 [100] 0.03505 0.06185 0.05615 0.06808 0.06966 0.04587 0.03631 0.02312
0.05739 0.06092 0.06095
 [111] 0.04408 0.06665 0.03780 0.05461 0.07122 0.06181 0.05740 0.05204
0.05868 0.04413 0.06526
 [122] 0.05876 0.03171 0.07018 0.03598 0.06380 0.07438 0.06730 0.05430
0.06330 0.06184 0.05697
 [133] 0.04523 0.05931 0.05689 0.04526 0.06516 0.02903 0.05658 0.06462
0.05938 0.07095 0.05274
 [144] 0.04042 0.05797 0.06004 0.06354 0.04076 0.05447 0.06511 0.03177
0.04152 0.04509 0.05561
 [155] 0.04321 0.06028 0.05347 0.06198 0.04505 0.05846 0.06721 0.06293
0.05749 0.04071 0.05931
 [166] 0.05745 0.06439 0.06399 0.05966 0.02966 0.06141 0.07258 0.06114
0.06122 0.06411 0.05421
```

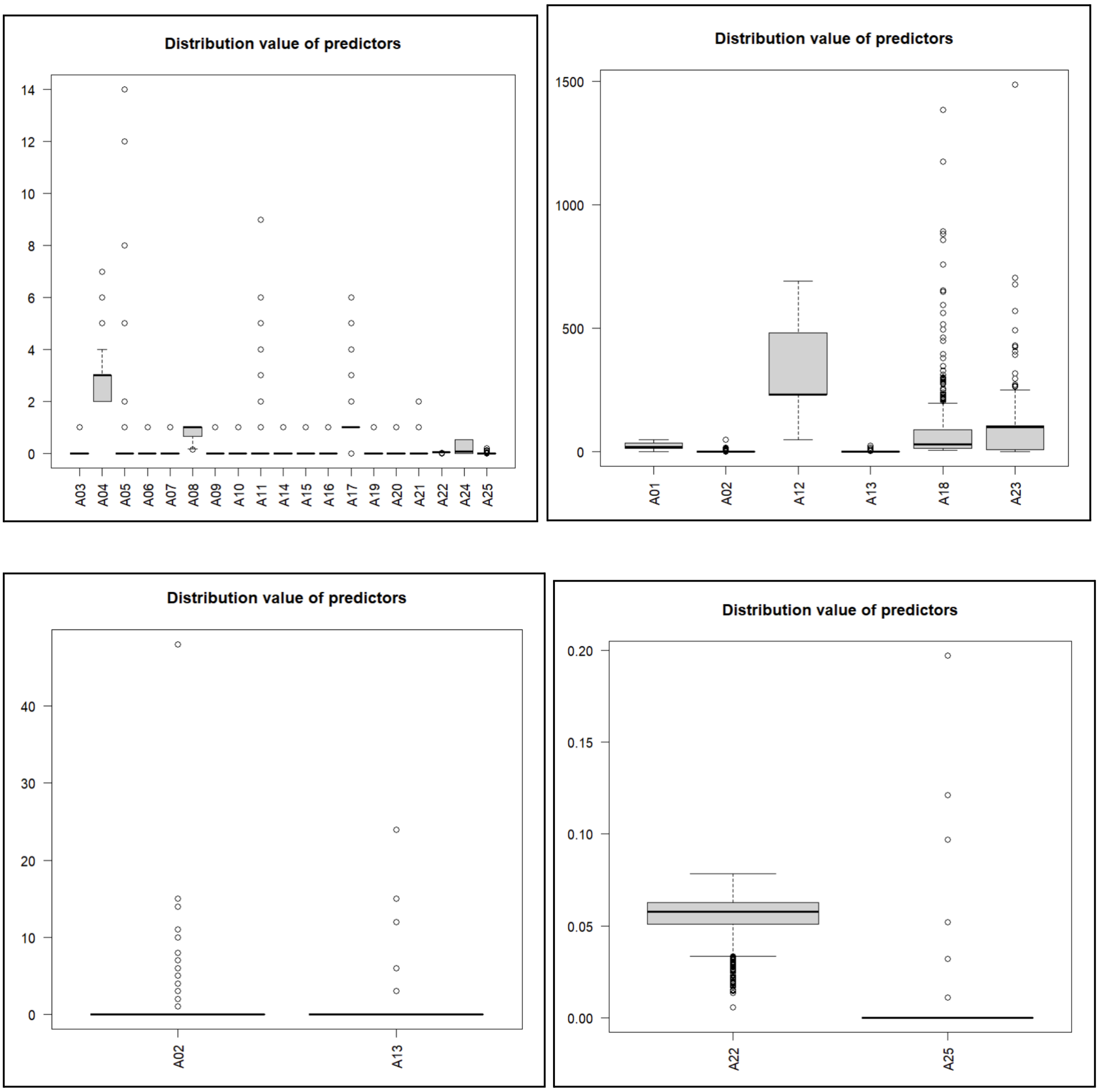
```
 [177] 0.05851 0.06142 0.06204 0.03266 0.06207 0.05618 0.04484 0.05529
0.05459 0.05557 0.05439
 [188] 0.05977 0.06571 0.05555 0.04474 0.04437 0.06857 0.06242 0.05929
0.01878 0.05087 0.05559
 [199] 0.05774 0.06011 0.05844 0.04048 0.05425 0.06148 0.03492 0.04300
0.04851 0.06249 0.04693
 [210] 0.05770 0.06840 0.06384 0.06272 0.06079 0.06887 0.05380 0.06082
0.04607 0.04446 0.05950
 [221] 0.06842 0.05428 0.05612 0.06550 0.04795 0.02846 0.05888 0.06816
0.03389 0.05665 0.03387
 [232] 0.03236 0.06153 0.05329 0.05839 0.06045 0.05317 0.06599 0.02953
0.05159 0.03574 0.06299
 [243] 0.05427 0.06664 0.06151 0.05331 0.04814 0.05434 0.05291 0.06102
0.06645 0.05824 0.04907
 [254] 0.05742 0.06528 0.06123 0.04272 0.06295 0.03846 0.03976 0.05950
0.07094 0.04951 0.06459
 [265] 0.04330 0.06777 0.05912 0.03735 0.06007 0.06359 0.04939 0.06486
0.06287 0.04673 0.06022
 [276] 0.06071 0.06234 0.05541 0.06719 0.06225 0.05969 0.04956 0.04974
0.06181 0.07760 0.05896
 [287] 0.06494 0.05516 0.05105 0.05925 0.04897 0.06317 0.04252 0.06577
0.07367 0.04661 0.06096
 [298] 0.05884 0.06901 0.04716 0.05787 0.05796 0.05674 0.06249 0.06813
0.05459 0.06482 0.07145
 [309] 0.06536 0.04687 0.05388 0.06059 0.05769 0.05540 0.06800 0.04171
0.02958 0.06966 0.05256
 [320] 0.05262 0.06219 0.07123 0.06125 0.05798 0.06167 0.05940 0.07769
0.06180 0.05980 0.06803
 [331] 0.04283 0.04765 0.05610 0.05008 0.06793 0.04833 0.05648 0.04055
0.05679 0.05326 0.04422
 [342] 0.04812 0.06274 0.04301 0.05733 0.05519 0.06280 0.04440 0.05776
0.07038 0.04666 0.05095
 [353] 0.05120 0.05957 0.06368 0.03685 0.03560 0.06606 0.03896 0.05590
0.06768 0.04499 0.06203
 [364] 0.04842 0.05470 0.05420 0.05417 0.05230 0.02145 0.06973 0.06355
0.05870 0.07079 0.05919
 [375] 0.04744 0.06419 0.03597 0.05571 0.03622 0.04919 0.06142 0.06122
0.06342 0.04704 0.01938
 [386] 0.06262 0.06515 0.05121 0.06636 0.06368 0.03851 0.07358 0.05639
0.05050 0.03175 0.05718
 [397] 0.05110 0.05865 0.06361 0.07043 0.05650 0.05381 0.05273 0.05484
0.05795 0.06852 0.05654
 [408] 0.04706 0.06192 0.07217 0.05991 0.04650 0.05898 0.06143 0.06480
0.05847 0.05383 0.05773
 [419] 0.05529 0.06577 0.06865 0.06942 0.06469 0.04574 0.05920 0.05619
0.05764 0.04922 0.04420
 [430] 0.06966 0.06555 0.05856 0.06028 0.06362 0.05763 0.05619 0.05670
0.05142 0.05206 0.06188
 [441] 0.04684 0.06807 0.03373 0.04901 0.06299 0.05485 0.05772 0.05935
0.06729 0.06686 0.05855
 [452] 0.05953 0.06443 0.05328 0.05781 0.05324 0.04226 0.05934 0.06014
0.06165 0.06471 0.05795
```

[463] 0.06338 0.06403 0.05321 0.07075 0.05822 0.06608 0.06369 0.06309 0.06683 0.05926 0.05238
[474] 0.05499 0.06511 0.05551 0.05086 0.04405 0.04478 0.06779 0.06041 0.05939 0.05505 0.05762
[485] 0.06098 0.04698 0.06171 0.05824 0.04543 0.05171 0.06125 0.05705 0.05870 0.07207 0.03899
[496] 0.01836 0.06465 0.06000 0.05901 0.06517 0.02865 0.04939 0.06706 0.06110 0.05336 0.03838
[507] 0.05947 0.05476 0.05686 0.05500 0.05918 0.06470 0.05910 0.05434 0.04600 0.03932 0.03973
[518] 0.05187 0.06606 0.05716 0.06492 0.06514 0.05360 0.03038 0.06118 0.06076 0.04858 0.06059
[529] 0.03850 0.05615 0.06145 0.03944 0.04149 0.06010 0.04508 0.05475 0.04371 0.06532 0.05605
[540] 0.06622 0.06361 0.05297 0.05835 0.05149 0.02944 0.05904 0.06418 0.06246 0.06432 0.04016
[551] 0.05026 0.04832 0.06254 0.06783 0.05795 0.06000 0.05216 0.05912 0.06354 0.04990 0.05634
[562] 0.06068 0.06458 0.05749 0.06257 0.05570 0.05228 0.04121 0.06209 0.04300 0.05466 0.04352
[573] 0.06839 0.06435 0.05664 0.05755 0.05956 0.05497 0.06422 0.05556 0.07673 0.07122 0.06488
[584] 0.06641 0.06950 0.04993 0.06023 0.05274 0.04772 0.06046 0.06166 0.07543 0.06364 0.05510
[595] 0.06408 0.04909 0.04508 0.06513 0.05575 0.05998 0.05462 0.05682 0.05975 0.05714 0.07110
[606] 0.03978 0.06134 0.05548 0.06027 0.06582 0.05789 0.06861 0.04276 0.07134 0.06378 0.06395
[617] 0.05824 0.04201 0.06600 0.03073 0.02645 0.05327 0.06081 0.06012 0.06345 0.06649 0.06284
[628] 0.04772 0.06408 0.05583 0.05855 0.05508 0.05862 0.04828 0.06718 0.06065 0.04236 0.06589
[639] 0.07140 0.06052 0.04685 0.06222 0.06210 0.05640 0.05239 0.05883 0.05351 0.05537 0.05594
[650] 0.07787 0.07275 0.06725 0.05661 0.05802 0.06982 0.06112 0.07207 0.07177 0.06187 0.07556
[661] 0.05923 0.03300 0.04609 0.06301 0.05321 0.07358 0.06579 0.05973 0.06228 0.05829 0.07105
[672] 0.06369 0.06244 0.07005 0.05449 0.05390 0.05148 0.05833 0.06056 0.06480 0.05978 0.03595
[683] 0.04922 0.06815 0.05247 0.05751 0.03247 0.06289 0.05383 0.06912 0.05274 0.05622 0.05403
[694] 0.05316 0.05921 0.05964 0.05954 0.06015 0.05779 0.06024 0.06268 0.04801 0.06272 0.06471
[705] 0.03767 0.05878 0.05349 0.06059 0.06557 0.06780 0.06595 0.06901 0.06471 0.05334 0.04987
[716] 0.06334 0.04328 0.06612 0.06099 0.05980 0.03912 0.03212 0.05686 0.06288 0.04852 0.05445
[727] 0.06490 0.06318 0.03669 0.04765 0.05570 0.06275 0.07704 0.03997 0.05804 0.05655 0.06055
[738] 0.07012 0.07082 0.05220 0.05627 0.06010 0.05179 0.05883 0.05150 0.05330 0.05593 0.06197

```
 [749] 0.06267 0.04202 0.07166 0.05955 0.06553 0.04007 0.05770 0.06730
0.05072 0.05932 0.06522
 [760] 0.05780 0.06592 0.05486 0.04860 0.05944 0.06837 0.06126 0.06270
0.06291 0.05962 0.04646
 [771] 0.05602 0.04245 0.06080 0.06048 0.03858 0.04348 0.02329 0.06193
0.04162 0.04325 0.03333
 [782] 0.05262 0.05887 0.06611 0.05400 0.05603 0.06048 0.05448 0.04862
0.05293 0.04718 0.04042
 [793] 0.06002 0.04407 0.05103 0.05788 0.02083 0.05309 0.05017 0.06436
0.05709 0.06237 0.06339
 [804] 0.06568 0.06283 0.05966 0.06294 0.05006 0.05122 0.03852 0.06034
0.04898 0.07106 0.05545
 [815] 0.05190 0.04141 0.06532 0.07849 0.05486 0.05430 0.04887 0.05503
0.06708 0.04524 0.05658
 [826] 0.06664 0.06629 0.06565 0.06161 0.06095 0.05586 0.06363 0.05475
0.05589 0.03992 0.05622
 [837] 0.05494 0.06236 0.05835 0.02225 0.05622 0.05088 0.04929 0.05252
0.04159 0.05967 0.05825
 [848] 0.05577 0.05322 0.05300 0.05927 0.05264 0.05167 0.06119 0.04947
0.05405 0.04941 0.02924
 [859] 0.07560 0.05460 0.05292 0.05519 0.06088 0.05245 0.05961 0.06452
0.04892 0.06228 0.04159
 [870] 0.05671 0.04448 0.05249 0.04904 0.05908 0.06173 0.04672 0.05916
0.06588 0.06674 0.06579
 [881] 0.04498 0.06168 0.05983 0.05784 0.06244 0.06086 0.07338 0.05501
0.06396 0.03381 0.06719
 [892] 0.07057 0.05141 0.06901 0.03287 0.06812 0.03922 0.04332 0.04954
0.07780 0.05428 0.06220
 [903] 0.07063 0.05716 0.05580 0.04087 0.03690 0.05620 0.06485 0.04477
0.05025 0.06211 0.06009
 [914] 0.06227 0.06764 0.06012 0.04405 0.06251 0.06332 0.05391 0.04377
0.04123 0.06226 0.03171
 [925] 0.07350 0.05533 0.06109 0.06099 0.06170 0.05518 0.05284 0.05646
0.04773 0.06620 0.05438
 [936] 0.05644 0.05919 0.04634 0.05103 0.05949 0.05778 0.07013 0.04825
0.06315 0.06078 0.06507
 [947] 0.05665 0.03975 0.05698 0.06596 0.05480 0.05974 0.04345 0.04669
0.05755 0.05206 0.05738
 [958] 0.04853 0.06155 0.06473 0.06419 0.06272 0.06982 0.06992 0.06795
0.05349 0.03729 0.06341
 [969] 0.06117 0.04852 0.06275 0.06319 0.06735 0.06302 0.04711 0.04769
0.04197 0.05762 0.06714
 [980] 0.06109 0.07149 0.05653 0.06721 0.03845 0.06162 0.05946 0.04526
0.06938 0.05526 0.05989
 [991] 0.07134 0.03521 0.05881 0.05395 0.05962 0.04339 0.05786 0.04175
0.06410 0.06869
 [ reached getOption("max.print") -- omitted 984 entries ]

$A23
  [1]  112   11  100    1  263   41   28  102    0  111   84  146    6  108
NA   39   24  131
 [19]  192  106   25   26  116  105  104   13    8   48  101  110  120   45
33  113    2   85
```

```
 [37]   50  114   80   88   17  186   15   32    4  103   49  141   12  107
14   66  140   81
 [55]   10  316  115  148   18   21  128   35    7  134  130   20   47  142
109   72  117   31
 [73]  127    3   30  124  169   19    5   38  136  152   68   51  126   23
122   98   61  153
 [91]  159   62   36  147   43  125   44  137   42   34   16   53   40  206
431   79  119  123
[109]  118   91    9   69   65  149   46  138   57   60  162   71   58   55
249   74  129   97
[127]  144   27  161   29  158 1487  493   89  132  677  171  271   75  121
168  199  145  407
[145]  163   52  392   22   99  265  133  182   96  154  248   37  143  211
170   73  705  135
[163]  317  157   63  218  226   56   70   67   54  167  173  156   78  139
86   76  151  424
[181]  160  189  231  297  184  570   59  165

$A24
  [1] 0.0285550 0.0799628 0.5229071 0.0384199 0.0059772 0.0002752 0.0001306
0.0001194 0.0094423
 [10] 0.0015020 0.0230451 0.0033282 0.0004041 0.0180132 0.0129268 0.0017218
0.0001858 0.0000800
 [19] 0.0015878 0.0004547 0.0115013 0.0049832 0.0006227 0.0101825 0.0100856
0.0008953 0.0141483
 [28] 0.0121780 0.0023289 0.0326503 0.0002282 0.0000464 0.0056158 0.0082003
0.0001144 0.0004411
 [37] 0.0064109 0.0012232 0.0001384 0.0001819 0.0017496 0.0003306 0.0069695
0.0009613 0.0003027
 [46] 0.0050842 0.0027596 0.0017975 0.0002984 0.0017656 0.0004020 0.0000502
0.0020445 0.0036375
 [55] 0.0003682 0.0000530 0.0002818 0.0015230 0.0002328 0.0022172        NA
0.0004584 0.0002256
 [64] 0.0002052 0.0004946 0.0018398 0.0013543 0.0011882 0.0001446 0.0038776
0.0002559 0.0008962
 [73] 0.0003477 0.0001507 0.0003718 0.0075051 0.0001665 0.0015361 0.0016589
0.0000967 0.0033189
 [82] 0.0000398 0.0045585 0.0033218 0.0036307 0.0021745 0.0000247 0.0000727
0.0000747 0.0002005
 [91] 0.0001088 0.0000467 0.0007497 0.0002358 0.0005731 0.0019696 0.0019933
0.0014653 0.0001699
[100] 0.0012193 0.0023239 0.0002356 0.0002355 0.0003860 0.0008316 0.0000333
0.0000706 0.0022733
[109] 0.0001612 0.0003212 0.0013635 0.0004267 0.0001586 0.0000418 0.0000881
0.0001532 0.0000338
[118] 0.0010349 0.0004928 0.0008079 0.0008457 0.0005052 0.0000000

$A25
[1] 0.000    NA 0.032 0.052 0.011 0.121 0.097 0.197

$Class
[1] 0 1
```

OUTPUT 4



OUTPUT 5

```
> performance
          Model Accuracy       AUC
1          Tree 76.49573 0.8071498
2   Naive Bayes 51.70940 0.7100782
3       Bagging 76.49573 0.7926174
4      Boosting 74.35897 0.7927268
5 Random Forest 77.56410 0.8128096
```

R CODE

```
setwd("C:/Users/USER/Desktop/FIT3152/Assignments/A2")

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


rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(33067902) # Your Student ID is the random seed
L <- as.data.frame(c(1:50))
L <- L[sample(nrow(L), 10, replace = FALSE),]
Phish <- Phish[(Phish$A01 %in% L),]
PD <- Phish[sample(nrow(Phish), 2000, replace = FALSE),] # sample of 2000 rows


#===================================
#QUESTION 1
#===================================
# Getting to know the data
View(PD)
dim(PD)

#Label 0 corresponds to a legitimate URL, label 1 to a phishing URL
#Proportion of phishing to legitimate
phishing = sum(PD$Class == 1)
phishing
legit = sum(PD$Class == 0)
legit

# Seeing the data types of each variables
str(PD)

# See the unique values in each columns
unique_values <- lapply(PD, unique)
unique_values

# Get the real valued attributes
real_valued = c(1, 2, 4, 5, 8, 11, 12, 17, 18, 22, 23, 24, 25)

# Create a data frame for real valued attributes
real_valued_PD = subset(PD, select = real_valued)
```

```
# Create visualization of distribution of real valued attributes
boxplot(PD[,c(1, 2, 4, 5, 8, 11, 17, 22, 24, 25)], las = 2, main = "Distribution of real valued
predictors")
boxplot(PD[,c(12, 18, 23)], las = 2, main = "Distribution of real valued predictors")


# Determining which variable can be omitted (seeing the distribution of values)
boxplot(PD[,c(3:11, 14:17, 19:22, 24:25)], las = 2, main = "Distribution value of predictors")
boxplot(PD[,c(1:2, 12:13, 18, 23)], las = 2, main = "Distribution value of predictors")
boxplot(PD[,c(2,13)], las = 2, main = "Distribution value of predictors")
boxplot(PD[,c(22, 25)], las = 2, main = "Distribution value of predictors")


#====================================
#QUESTION 2
#====================================
# Remove NAs
PD = na.omit(PD)
# Change dependent variable to factor type
PD$Class = factor(PD$Class)


#====================================
#QUESTION 3
#====================================
set.seed(33067902) #Student ID as random seed
# Split into 70% training data and 30% testing data
train.row = sample(1:nrow(PD), 0.7*nrow(PD))
PD.train = PD[train.row,]
PD.test = PD[-train.row,]

#====================================
#QUESTION 4
#====================================
# Decision Tree
tree.fit = tree(Class ~., data = PD.train)
# Plot the decision tree
plot(tree.fit, main = "Decision Tree")
text(tree.fit, pretty = 0)

# Naïve Bayes
naive.fit = naiveBayes(Class~., data = PD.train)

# Bagging
bag.fit = bagging(Class~., data = PD.train, mfinal = 10)

# Boosting
```

```
boost.fit = boosting(Class~., data = PD.train, mfinal = 10)

# Random Forest
rf.fit <- randomForest(Class ~ ., data = PD.train)


#==================================
#QUESTION 5
#==================================
# Decision Tree
# Make Prediction
tree.predict = predict(tree.fit, PD.test, type = "class")
# Create confusion matrix
tree.table = table(actual = PD.test$Class, predicted = tree.predict)
tree.table
# Calculate accuracy
tree.acc = (sum(diag(as.matrix(tree.table))) / nrow(PD.test))*100
tree.acc


# ====================================================
# Naïve Bayes
# Make Prediction
naive.predict = predict(naive.fit, PD.test, type = "class")
# Create confusion matrix
naive.table = table(actual = PD.test$Class, predicted = naive.predict)
naive.table
# Calculate accuracy
naive.acc = (sum(diag(as.matrix(naive.table))) / nrow(PD.test))*100
naive.acc


# ====================================================
# Bagging
# Make Prediction
bag.predict = predict.bagging(bag.fit, newdata = PD.test)
# Create confusion matrix
bag.predict$confusion
# Calculate accuracy
bagging.acc = (sum(diag(as.matrix(bag.predict$confusion))) / nrow(PD.test))*100
bagging.acc


# ====================================================
# Boosting
# Make Prediction
boost.predict = predict.boosting(boost.fit, newdata = PD.test)
# Create confusion matrix
```

```
boost.predict$confusion
# Calculate accuracy
boost.acc = (sum(diag(as.matrix(boost.predict$confusion))) / nrow(PD.test))*100
boost.acc


# ========================================================
# Random Forest
# Make Prediction
rf.predict <- predict(rf.fit, PD.test)
# Create confusion matrix
rf.table = table(observed = PD.test$Class, predicted = rf.predict)
rf.table
# Calculate accuracy
rf.acc = (sum(diag(as.matrix(rf.table))) / nrow(PD.test))*100
rf.acc


#=====================================
#QUESTION 6
#=====================================
# Decision Tree
# Calculate confidence
tree_conf = predict(tree.fit, PD.test, type = "vector")
# Create prediction object (choose 2nd bcs we only use the probability of yes)
tree_prediction = prediction(tree_conf[,2], PD.test$Class)
# Calculate TPR AND FPR
tree_ROC = performance(tree_prediction, "tpr", "fpr")
# Plot ROC
plot(tree_ROC, col = "blue", main = "ROC Curve of Classification Models")
abline(0,1)

# Calculate AUC
tree_auc = performance(tree_prediction, "auc")
tree_auc = as.numeric(tree_auc@y.values)
tree_auc

#================================================
# Naïve Bayes
# Calculate confidence
naive_conf = predict(naive.fit, PD.test, type = "raw")
# Create prediction object (choose 2nd bcs we only use the probability of yes)
naive_prediction = prediction(naive_conf[,2], PD.test$Class)
# Calculate TPR AND FPR
naive_ROC = performance(naive_prediction, "tpr", "fpr")
# Plot ROC
plot(naive_ROC, add = TRUE,  col = "red")
```

```
# Calculate AUC
naive_auc = performance(naive_prediction, "auc")
naive_auc = as.numeric(naive_auc@y.values)
naive_auc

#==============================================
# Bagging
# Create prediction object (choose 2nd bcs we only use the probability of yes)
bagging_prediction = prediction(bag.predict$prob[,2], PD.test$Class)
# Calculate TPR AND FPR
bagging_ROC = performance(bagging_prediction, "tpr", "fpr")
# Plot ROC
plot(bagging_ROC, add = TRUE,  col = "green")


# Calculate AUC
bagging_auc = performance(bagging_prediction, "auc")
bagging_auc = as.numeric(bagging_auc@y.values)
bagging_auc

#==============================================
# Boosting
# Create prediction object (choose 2nd bcs we only use the probability of yes)
boosting_prediction = prediction(boost.predict$prob[,2], PD.test$Class)
# Calculate TPR AND FPR
boosting_ROC = performance(boosting_prediction, "tpr", "fpr")
# Plot ROC
plot(boosting_ROC, add = TRUE,  col = "blueviolet")


# Calculate AUC
boosting_auc = performance(boosting_prediction, "auc")
boosting_auc = as.numeric(boosting_auc@y.values)
boosting_auc

#==============================================
# Random Forest
rf_conf = predict(rf.fit, PD.test, type = "prob")
# Create prediction object (choose 2nd bcs we only use the probability of yes)
rf_prediction = prediction(rf_conf[,2], PD.test$Class)
# Calculate TPR AND FPR
rf_ROC = performance(rf_prediction, "tpr", "fpr")
# Plot ROC
plot(rf_ROC, add = TRUE,  col = "black")
# Give legend to plot
legend("bottomright", legend =c("Decision Tree", "Naive Bayes", "Bagging"
                              , "Boosting", "Random Forest"),
```

```
      col = c("blue", "red", "green", "blueviolet", "black"), lty = c(1,1),
      lwd = 2, cex = 1.5)


# Calculate AUC
rf_auc = performance(rf_prediction, "auc")
rf_auc = as.numeric(rf_auc@y.values)
rf_auc


#===================================
#QUESTION 7
#===================================
# Create data frame for the performance of each models
performance = data.frame(Model = c("Tree", "Naive Bayes", "Bagging", "Boosting"
                          , "Random Forest"),
                  Accuracy = c(tree.acc, naive.acc, bagging.acc, boost.acc, rf.acc),
                  AUC = c(tree_auc, naive_auc, bagging_auc, boosting_auc, rf_auc))
performance


#===================================
#QUESTION 8
#===================================
# Decision Tree
summary(tree.fit)

#============================================
# Bagging
bag.fit$importance
# Make Bar Plot
barplot(bag.fit$importance[order(bag.fit$importance, decreasing = TRUE)], ylim = c(0, 100),
       main = "Bagging Variable Importance")

#============================================
# Boosting
boost.fit$importance
# Make Bar Plot
barplot(boost.fit$importance[order(boost.fit$importance, decreasing = TRUE)], ylim = c(0,
100),
       main = "Boosting Variable Importance")

#============================================
# Random forest
rf.importance = rf.fit$importance
# Sort in order from most improtant to least
rf.importance = rf.importance[order(rf.importance[,1], decreasing = TRUE), , drop = FALSE]
# Make Bar Plot
```

```
barplot(rf.importance[,1], ylim = c(0, max(rf.importance[,1])), names.arg =
rownames(rf.importance),
        main = "Random Forest Variable Importance")


#===================================
#QUESTION 9
#===================================
# Perform cross validation
cvtest = cv.tree(tree.fit, FUN = prune.misclass)
cvtest
# Prune the decision tree model
pruned.Dfit = prune.misclass(tree.fit, best = 3)
summary(pruned.Dfit)

# Make prediction for simple Decision Tree
treeSimple.predict = predict(pruned.Dfit, PD.test, type = "class")
# Calculate accuracy by confusion matrix
treeSimple.table = table(actual = PD.test$Class, predicted = treeSimple.predict)
treeSimple.table
treeSimple.acc = (sum(diag(as.matrix(treeSimple.table))) / nrow(PD.test))*100
treeSimple.acc


# Calculate confidence
tree_conf_s = predict(pruned.Dfit, PD.test, type = "vector")

# Create prediction object (choose 2nd bcs we only use the probability of yes)
tree_prediction_s = prediction(tree_conf_s[,2], PD.test$Class)

# Calculate TPR AND FPR
tree_ROC_s= performance(tree_prediction_s, "tpr", "fpr")

# Calculate AUC
tree_auc_s= performance(tree_prediction_s, "auc")
tree_auc_s = as.numeric(tree_auc_s@y.values)
tree_auc_s

# Comparison of the simple tree and the original tree
summary(pruned.Dfit)
summary(tree.fit)

# Plot the simple tree
plot(pruned.Dfit, main = "Simple Decision Tree")
text(pruned.Dfit, pretty = 0)

#===================================
#QUESTION 10
```

```
#===================================
# Making the improved decision tree (changing the mincut)
treeImproved.fit = tree(Class ~., PD.train, mincut = 15)

# Make prediction for imrpved Decision Tree
treeImproved.predict = predict(treeImproved.fit, PD.test, type = "class")
# Calculate accuracy by confusion matrix
treeImproved.table = table(actual = PD.test$Class, predicted = treeImproved.predict)
treeImproved.table
treeImproved.acc = (sum(diag(as.matrix(treeImproved.table))) / nrow(PD.test))*100
treeImproved.acc

# Check the summary of the improved tree
summary(treeImproved.fit)

# Calculate confidence
tree_conf_i = predict(treeImproved.fit, PD.test, type = "vector")

# Create prediction object (choose 2nd bcs we only use the probability of yes)
tree_prediction_i = prediction(tree_conf_i[,2], PD.test$Class)

# Calculate TPR AND FPR
tree_ROC_i= performance(tree_prediction_i, "tpr", "fpr")

# Calculate AUC
tree_auc_i= performance(tree_prediction_i, "auc")
tree_auc_i = as.numeric(tree_auc_i@y.values)
tree_auc_i

# Plot the improved tree
plot(treeImproved.fit, main = "Improved Decision Tree")
text(treeImproved.fit, pretty = 0)

#===================================
#QUESTION 11
#===================================
# Import needed library
library(neuralnet)

# Make 80% training data and 20% testing data
set.seed(33067902) #Student ID as random seed
train.row.net = sample(1:nrow(PD), 0.8*nrow(PD))
PD.train.net = PD[train.row.net,]
PD.test.net = PD[-train.row.net,]


# Create Artificial Neural Network
```

```
net.PD <- neuralnet(Class == 1~ A01 + A18 + A22 + A23, PD.train.net, hidden=5,
linear.output = FALSE, stepmax = 1e7)

# Make prediction to the testing data
net.pred = compute(net.PD, PD.test.net[c(1, 18, 22, 23)])

# Round the result
net.predr = round(net.pred$net.result,0)
# Change to dataframe
net.predrdf = as.data.frame(net.predr)

# Create a confusion matrix
net.table = table(observed = PD.test.net$Class, predicted =
      net.predrdf$V1)
net.table

# Calculate Accuracy
net.acc = (sum(diag(as.matrix(net.table))) / nrow(PD.test.net))*100
net.acc

# Calculate confidence
net_conf = predict(net.PD, PD.test.net, type = "response")

# Create prediction object (choose 2nd bcs we only use the probability of yes)
net_prediction = ROCR::prediction(net.pred$net.result[,1], PD.test.net$Class)

# Calculate AUC
net_auc = performance(net_prediction, "auc")
net_auc = as.numeric(net_auc@y.values)
net_auc

# Plot ANN
plot(net.PD, rep="best")


#===================================
#QUESTION 12
#===================================
# MAKING SVM MODEL
# Only choose the important variables
PD.train.svm = PD.train[, c(1,  18, 22, 23, 26)]
PD.test.svm = PD.test[, c(1, 18, 22, 23, 26)]

# Scale independent variables
PD.train.svm[-5] = scale(PD.train.svm[-5])
PD.test.svm[-5] = scale(PD.test.svm[-5])

# Fitting SVM to the Training set
```

```
svm.model = svm(formula = Class ~ .,
                data = PD.train.svm,
                type = 'C-classification',
                kernel = 'radial',
               probability = TRUE)

# Make prediction to the test data
svm.pred = predict(svm.model, newdata = PD.test.svm)

# Create Confusion Matrix
svm.table = table(PD.test.svm$Class, svm.pred)
svm.table

# Calculate Accuracy
svm.acc = (sum(diag(as.matrix(svm.table))) / nrow(PD.test.svm))*100
svm.acc

# Calculate confidence
svm.pred.prob = predict(svm.model, newdata = PD.test.svm, probability = TRUE)
svm.pred.prob = attr(svm.pred.prob, "probabilities")

# Create prediction object (choose 2nd bcs we only use the probability of yes)
net_prediction = ROCR::prediction(svm.pred.prob[, 2], PD.test.svm$Class)

# Calculate AUC
svm_auc = performance(net_prediction, "auc")
svm_auc = as.numeric(svm_auc@y.values)
svm_auc
```