Homework Assignment for Chapter 12

12.1 The Hepatic Injury

a) Training and Testing set.

Given the classification imbalance, stratified random sampling method would be the best approach in splitting the data into training and testing set.

b) Classification statistic

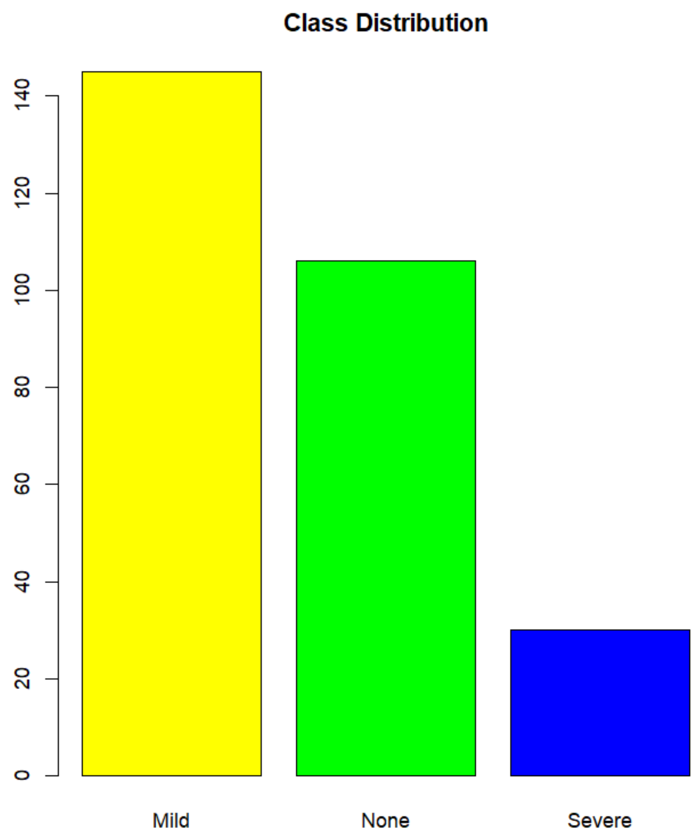


Fig 1: Distribution of the different classes in the response variable of the dataset

Given the distribution has more than 2 classes, kappa and accuracy are the best classification statistics. I would choose Kappa as a classification statistic to optimize for this exercise.

c) Training and Testing

Logistic Regression

```
225 samples
 96 predictor
  3 classes: 'Mild', 'None', 'Severe'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 169, 169, 169, 169, 169, 169, ...
Resampling results across tuning parameters:

  decay  logLoss    AUC        prAUC      Accuracy   Kappa        Mean_F1    Mean_Sensitivity  Mean_Specificity  Mean_Pos_Pred_Value
  0e+00  19.849982  0.5436728  0.2386227  0.3921429  0.037535713  0.3522554  0.3660427         0.6790519         0.3595268
  1e-04  11.352295  0.5473951  0.3362843  0.4000000  0.038452093  0.3523776  0.3673892         0.6784508         0.3571626
  1e-01   2.071993  0.5385078  0.3499925  0.4278571  0.003155277  0.3747908  0.3541434         0.6652134         0.3638287
  Mean_Neg_Pred_Value  Mean_Precision  Mean_Recall  Mean_Detection_Rate  Mean_Balanced_Accuracy
  0.6783829            0.3595268       0.3660427    0.1307143            0.5225473
  0.6789135            0.3571626       0.3673892    0.1333333            0.5229200
  0.6639770            0.3638287       0.3541434    0.1426190            0.5096784

Kappa was used to select the optimal model using the largest value.
The final value used for the model was decay = 1e-04.
```

```
Confusion Matrix and Statistics

          Reference
Prediction Mild None Severe
    Mild     12   11      5
    None     12    5      0
    Severe    5    5      1

Overall Statistics

               Accuracy : 0.3214
                 95% CI : (0.2029, 0.4596)
    No Information Rate : 0.5179
    P-Value [Acc > NIR] : 0.9990

                  Kappa : -0.1194

 Mcnemar's Test P-Value : 0.1686

Statistics by Class:

                     Class: Mild Class: None Class: Severe
Sensitivity               0.4138     0.23810       0.16667
Specificity               0.4074     0.65714       0.80000
Pos Pred Value            0.4286     0.29412       0.09091
Neg Pred Value            0.3929     0.58974       0.88889
Prevalence                0.5179     0.37500       0.10714
Detection Rate            0.2143     0.08929       0.01786
Detection Prevalence      0.5000     0.30357       0.19643
Balanced Accuracy         0.4106     0.44762       0.48333
> 
```

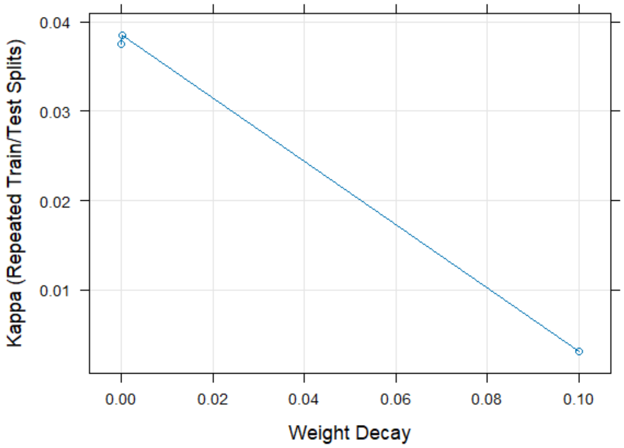


Fig 2: Plot of Kappa vs Weigh Decay for Logistic Regression

LDA

```
Linear Discriminant Analysis

225 samples
 96 predictor
  3 classes: 'Mild', 'None', 'Severe'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 169, 169, 169, 169, 169, 169, ...
Resampling results:

  logLoss   AUC        prAUC      Accuracy   Kappa       Mean_F1    Mean_Sensitivity  Mean_Specificity  Mean_Pos_Pred_Value
  4.646426  0.5413375  0.3591677  0.4214286  0.04851528  0.3594966  0.3681554         0.6838434         0.3624741
  Mean_Neg_Pred_Value  Mean_Precision  Mean_Recall  Mean_Detection_Rate  Mean_Balanced_Accuracy
  0.6833069            0.3624741       0.3681554    0.1404762            0.5259994
```

```
Confusion Matrix and Statistics

          Reference
Prediction Mild None Severe
    Mild     18    7      2
    None     10   12      1
    Severe    1    2      3

Overall Statistics

               Accuracy : 0.5893
                 95% CI : (0.4498, 0.719)
    No Information Rate : 0.5179
    P-Value [Acc > NIR] : 0.1747

                  Kappa : 0.2977

 Mcnemar's Test P-Value : 0.7539

Statistics by Class:

                     Class: Mild Class: None Class: Severe
Sensitivity               0.6207      0.5714       0.50000
Specificity               0.6667      0.6857       0.94000
Pos Pred Value            0.6667      0.5217       0.50000
Neg Pred Value            0.6207      0.7273       0.94000
Prevalence                0.5179      0.3750       0.10714
Detection Rate            0.3214      0.2143       0.05357
Detection Prevalence      0.4821      0.4107       0.10714
Balanced Accuracy         0.6437      0.6286       0.72000
```

PLSDA

```
Partial Least Squares

225 samples
 96 predictor
  3 classes: 'Mild', 'None', 'Severe'

Pre-processing: centered (96), scaled (96)
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 169, 169, 169, 169, 169, 169, ...
Resampling results across tuning parameters:

  ncomp  logLoss   AUC        prAUC      Accuracy   Kappa        Mean_F1    Mean_Sensitivity  Mean_Specificity  Mean_Pos_Pred_Value
   1     1.029974  0.5314927  0.3520355  0.5100000  0.052719936        NaN  0.3548987         0.6827937               NaN
   2     1.051596  0.5233578  0.3448435  0.4864286  0.019030137        NaN  0.3414778         0.6723668               NaN
   3     1.064066  0.5300979  0.3462342  0.4878571  0.035304728        NaN  0.3469513         0.6785467               NaN
   4     1.070227  0.5384948  0.3513280  0.4750000  0.019669134  0.3377176  0.3411385         0.6735041         0.3459017
   5     1.078944  0.5461276  0.3515789  0.4700000  0.012874258  0.4297998  0.3407444         0.6710688         0.3967828
   6     1.088574  0.5437108  0.3514548  0.4864286  0.046249714  0.4160606  0.3601204         0.6816353         0.4654380
   7     1.108331  0.5351090  0.3452783  0.4628571  0.006963565  0.3772711  0.3395074         0.6693079         0.3774511
   8     1.120990  0.5352470  0.3453930  0.4678571  0.024051798  0.4025025  0.3460536         0.6750293         0.3587227
   9     1.126809  0.5363608  0.3494551  0.4678571  0.030788391  0.3868731  0.3497537         0.6776395         0.3586208
  10     1.137619  0.5391484  0.3520954  0.4735714  0.043787416  0.3886049  0.3536070         0.6821630         0.3645169
  11     1.146572  0.5381211  0.3494566  0.4671429  0.038185786  0.3908116  0.3510564         0.6803302         0.3571913
  12     1.153089  0.5402534  0.3524448  0.4685714  0.045767663  0.3879850  0.3530268         0.6829108         0.3454018
  13     1.156581  0.5411761  0.3536293  0.4707143  0.051486885  0.3793724  0.3547564         0.6850765         0.3484363
  14     1.157705  0.5453142  0.3588207  0.4792857  0.070475368  0.3979530  0.3651998         0.6914610         0.3532554
  15     1.161970  0.5464935  0.3587136  0.4757143  0.069357772  0.3935199  0.3643021         0.6908063         0.3515199
  Mean_Neg_Pred_Value  Mean_Precision  Mean_Recall  Mean_Detection_Rate  Mean_Balanced_Accuracy
  0.6900099                  NaN       0.3548987    0.1700000            0.5188462
  0.6744536                  NaN       0.3414778    0.1621429            0.5069223
  0.6829147                  NaN       0.3469513    0.1626190            0.5127490
  0.6758845            0.3459017       0.3411385    0.1583333            0.5073213
  0.6729549            0.3967828       0.3407444    0.1566667            0.5059066
  0.6851454            0.4654380       0.3601204    0.1621429            0.5208778
  0.6713721            0.3774511       0.3395074    0.1542857            0.5044077
  0.6770840            0.3587227       0.3460536    0.1559524            0.5105415
  0.6795845            0.3586208       0.3497537    0.1559524            0.5136966
  0.6854422            0.3645169       0.3536070    0.1578571            0.5178850
  0.6827195            0.3571913       0.3510564    0.1557143            0.5156933
  0.6853472            0.3454018       0.3530268    0.1561905            0.5179688
  0.6875420            0.3484363       0.3547564    0.1569048            0.5199165
  0.6942946            0.3532554       0.3651998    0.1597619            0.5283304
  0.6929204            0.3515199       0.3643021    0.1585714            0.5275542

Kappa was used to select the optimal model using the largest value.
The final value used for the model was ncomp = 14.
```

```
Confusion Matrix and Statistics

          Reference
Prediction Mild None Severe
    Mild     17   11      3
    None     12    9      2
    Severe    0    1      1

Overall Statistics

               Accuracy : 0.4821
                 95% CI : (0.3466, 0.6197)
    No Information Rate : 0.5179
    P-Value [Acc > NIR] : 0.7482

                  Kappa : 0.0677

 Mcnemar's Test P-Value : 0.3371

Statistics by Class:

                     Class: Mild Class: None Class: Severe
Sensitivity               0.5862      0.4286       0.16667
Specificity               0.4815      0.6000       0.98000
Pos Pred Value            0.5484      0.3913       0.50000
Neg Pred Value            0.5200      0.6364       0.90741
Prevalence                0.5179      0.3750       0.10714
Detection Rate            0.3036      0.1607       0.01786
Detection Prevalence      0.5536      0.4107       0.03571
Balanced Accuracy         0.5338      0.5143       0.57333
```

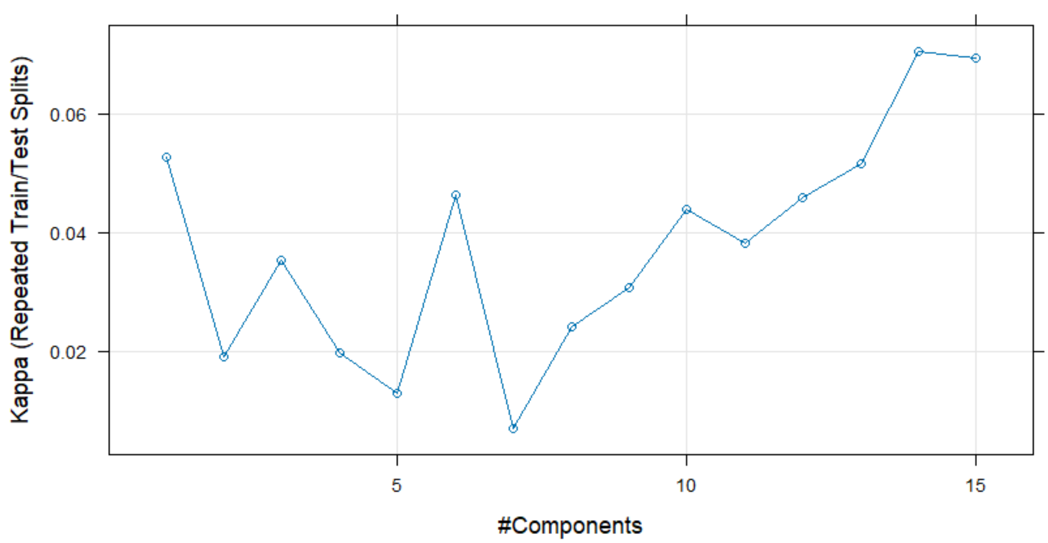


Fig 3: Plot of tuning parameter for PLSDA model

Glmnet

```
glmnet

225 samples
 96 predictor
  3 classes: 'Mild', 'None', 'Severe'

Pre-processing: centered (96), scaled (96)
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 169, 169, 169, 169, 169, 169, ...
Resampling results across tuning parameters:

  alpha  lambda      logLoss    AUC        prAUC        Accuracy   Kappa        Mean_F1
  0.0    0.01000000  2.1117474  0.5503897  0.357011044  0.4614286  0.058379737  0.3930794
  0.0    0.03111111  1.5789332  0.5490661  0.352513538  0.4778571  0.062190735  0.4072934
  0.0    0.05222222  1.4012425  0.5493445  0.352489622  0.4778571  0.048091939  0.3916180
  0.0    0.07333333  1.3065767  0.5498002  0.352194090  0.4792857  0.045051622  0.3899835
  0.0    0.09444444  1.2460239  0.5497011  0.351767249  0.4821429  0.043447276  0.3920211
  0.0    0.11555556  1.2034180  0.5501954  0.352261830  0.4871429  0.046545530  0.3916944
  0.0    0.13666667  1.1715110  0.5507062  0.352427990  0.4892857  0.046390871  0.3893633
  0.0    0.15777778  1.1467007  0.5509706  0.353066580  0.4935714  0.049037172  0.3674336
  0.0    0.17888889  1.1266409  0.5507307  0.353860542  0.4892857  0.038228036  0.3663536
  0.0    0.20000000  1.1101390  0.5506610  0.355263963  0.4885714  0.032932254  0.3750842
  0.1    0.01000000  2.0180586  0.5510356  0.355882434  0.4671429  0.062880758  0.4029170
  0.1    0.03111111  1.4439662  0.5471923  0.351539680  0.4721429  0.042018618  0.3886079
  0.1    0.05222222  1.2662955  0.5472742  0.352780964  0.4785714  0.038513542  0.3853954
  0.1    0.07333333  1.1757593  0.5493696  0.355813675  0.4842857  0.038512670  0.4074613
  0.1    0.09444444  1.1213088  0.5503700  0.356646898  0.4828571  0.024652295  0.3946197
  0.1    0.11555556  1.0855167  0.5479795  0.355169091  0.4864286  0.022796884        NaN
  0.1    0.13666667  1.0601313  0.5464738  0.354263181  0.4878571  0.019598612        NaN
  0.1    0.15777778  1.0407978  0.5459843  0.354547163  0.4942857  0.025373068        NaN
  0.1    0.17888889  1.0258990  0.5446728  0.353850814  0.4928571  0.017601509        NaN
  0.1    0.20000000  1.0139374  0.5430350  0.352752720  0.5000000  0.028572519        NaN
  0.2    0.01000000  1.9175937  0.5492235  0.355895550  0.4635714  0.053700776  0.4003238
  0.2    0.03111111  1.3431075  0.5475500  0.353914575  0.4771429  0.042084080  0.3974343
  0.2    0.05222222  1.1794419  0.5490482  0.355374039  0.4800000  0.029021478  0.4081553
```

```
        NaN        0.3333333     0.1726190                 0.5000000
  0.3621103        0.3667980     0.1414286                 0.5247098
        NaN        0.3333333     0.1726190                 0.5000000
        NaN        0.3333333     0.1726190                 0.5000000
        NaN        0.3333333     0.1726190                 0.5000000
        NaN        0.3333333     0.1726190                 0.5000000
        NaN        0.3333333     0.1726190                 0.5000000
        NaN        0.3333333     0.1726190                 0.5000000
        NaN        0.3333333     0.1726190                 0.5000000
        NaN        0.3333333     0.1726190                 0.5000000
        NaN        0.3333333     0.1726190                 0.5000000
  0.3432257        0.3482978     0.1347619                 0.5100762
        NaN        0.3333333     0.1726190                 0.5000000
 [ reached getOption("max.print") -- omitted 8 rows ]

Kappa was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0 and lambda = 0.
```

```
Confusion Matrix and Statistics

          Reference
Prediction Mild None Severe
    Mild     17    9      3
    None     12   11      2
    Severe    0    1      1

Overall Statistics

               Accuracy : 0.5179
                 95% CI : (0.3803, 0.6534)
    No Information Rate : 0.5179
    P-Value [Acc > NIR] : 0.5537

                  Kappa : 0.1399

 Mcnemar's Test P-Value : 0.2883

Statistics by Class:

                     Class: Mild Class: None Class: Severe
Sensitivity               0.5862      0.5238       0.16667
Specificity               0.5556      0.6000       0.98000
Pos Pred Value            0.5862      0.4400       0.50000
Neg Pred Value            0.5556      0.6774       0.90741
Prevalence                0.5179      0.3750       0.10714
Detection Rate            0.3036      0.1964       0.01786
Detection Prevalence      0.5179      0.4464       0.03571
Balanced Accuracy         0.5709      0.5619       0.57333
```

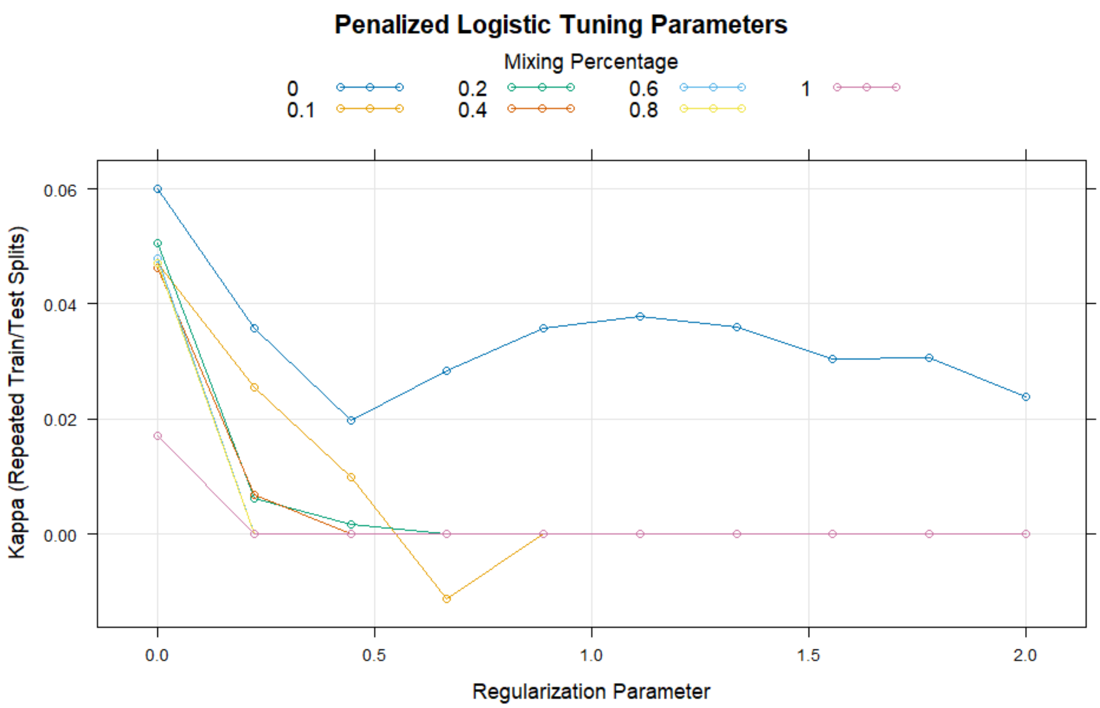


Fig 4: Plot of tuning parameter for glmnet model

Model	Best Tuning Parameter	Training		Testing	
		Kappa	Accuracy	Kappa	Accuracy
Logistic Regression	Decay=1e-04	0.03845	0.4000	-0.1194	0.3214
Linear Discriminant Analysis	N/A	0.04851528	0.4214286	0.2977	0.5893
PLSDA	ncomp=14	0.0704754	0.4792857	0.0677	0.4821
glmnet	alpha=0, lambda=0.0	0.06000	0.4621429	0.1399	0.5179

Looking at the results from the table, the models exhibit different performances. On the training, PLSDA has the best performance with an accuracy of 47.93% and kappa of 0.07048. On the testing, LDA has the best performance with an accuracy of 58.93% and kappa of 0.2977. However, LDA performs poorly on the training while PLSDA performs poorly on testing. The best model for predicting injury based on biological factors is glmnet as it performs better on both training and testing with Accuracy of 46.21% and kappa of 0.06 on training and accuracy of 51.79% and kappa of 0.1399 on testing. Glmnet was the second-best performing model on both training and testing while LDA and PLSDA fluctuated in performance.

d) Five important predictors

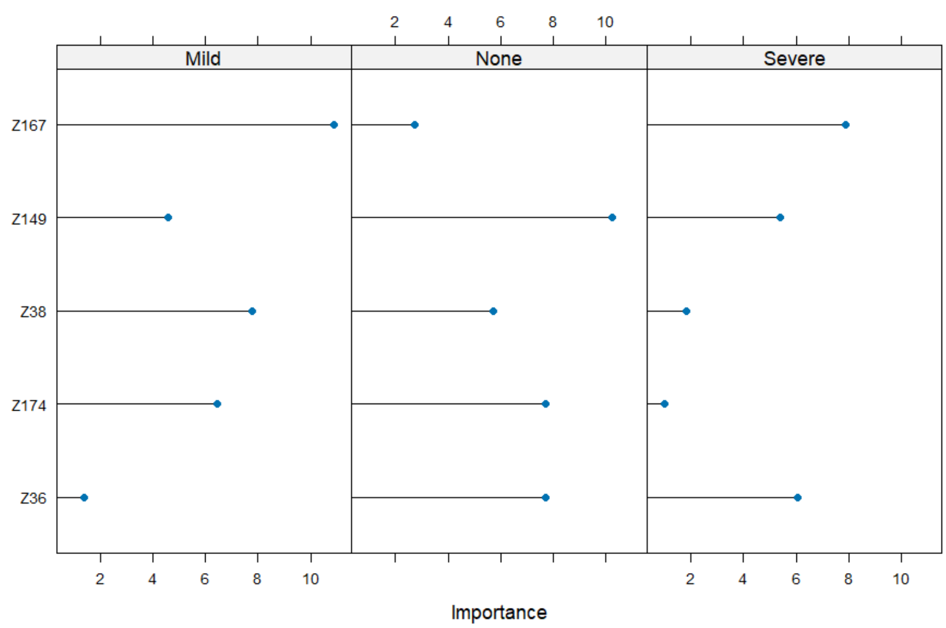


Fig 5: Important features for glmnet model

12.3 MLC++ software package

a) Explore the data

The ‘mlc_churn’ dataset has 5000 observations and 20 predictors

NearZeroVariance

```
> nZv <- nearZeroVar(mlc_churn)
> length(nZv)
[1] 1
>
```

From the dataset, one predictor was found to have near zero variance. The predictor was removed.

Distribution for response variable

```
> counts

 yes   no
 707 4293
> percentage <- prop.table(counts) * 100
> percentage

  yes    no
14.14 85.86
```

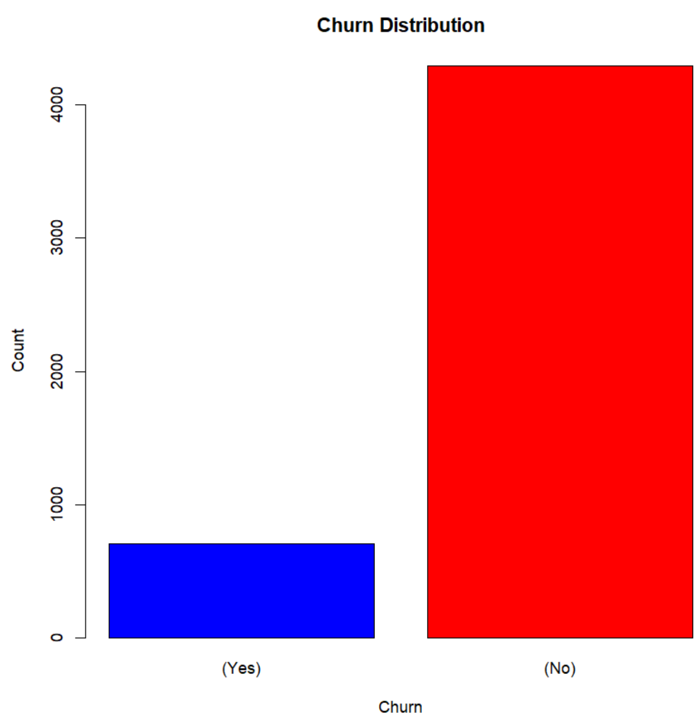


Fig 6: Churn distribution

The response variable was churn, with two classes(‘Yes’, ‘No’). 14.14% of the variables were under ‘Yes’ class and 85.86% were under the ‘No’ class.

Barplot for Categorical pred

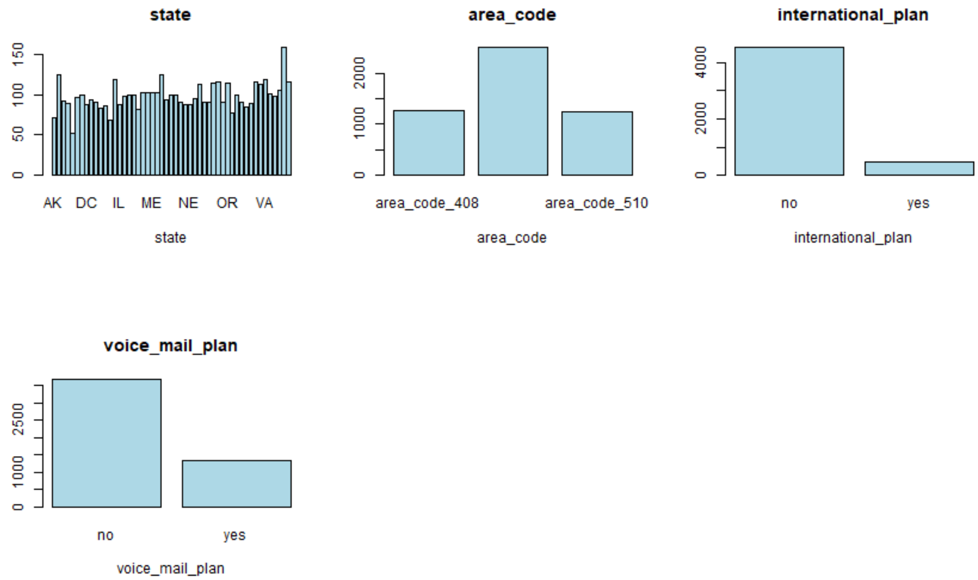
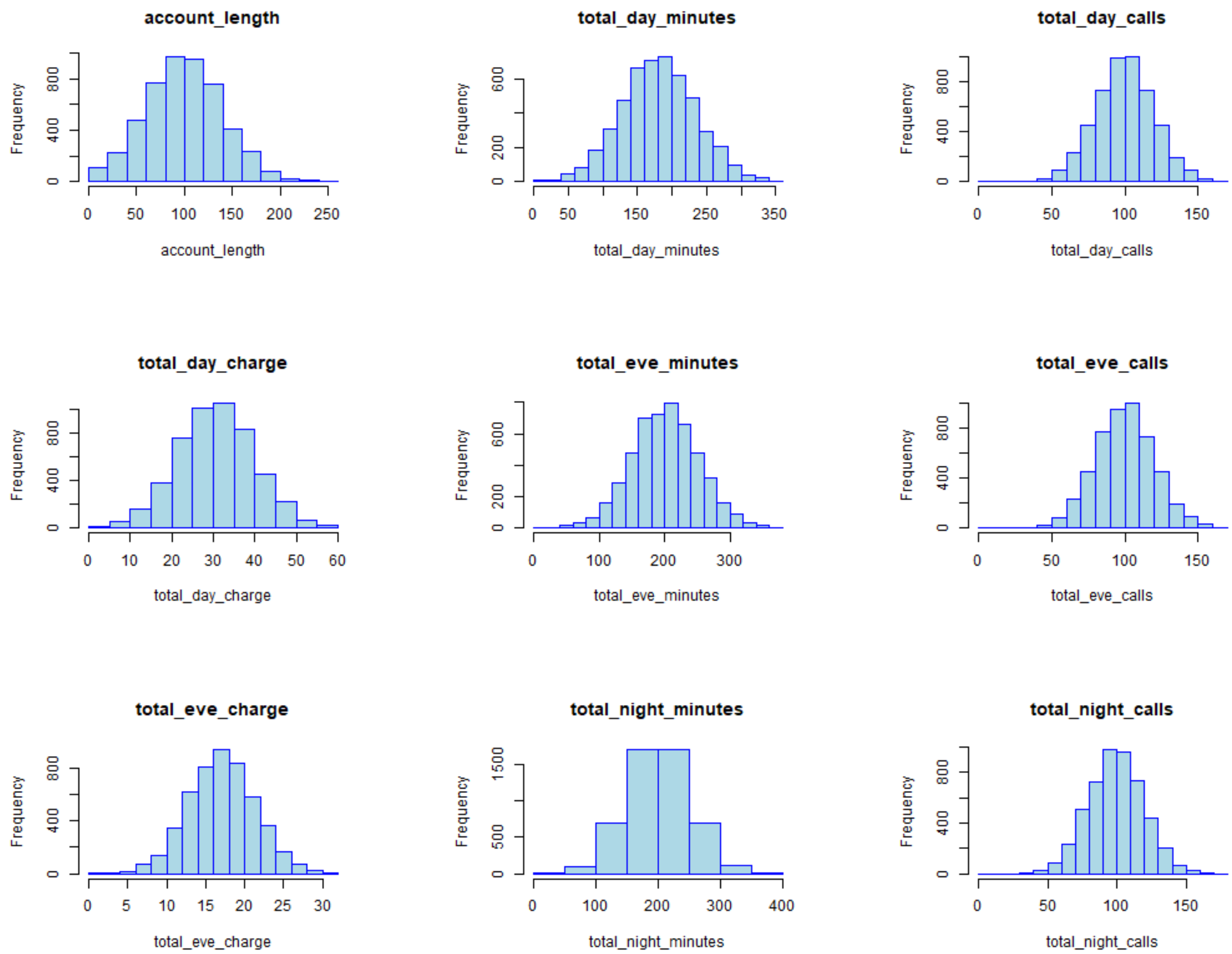


Fig 7: Distribution for categorical predictors

Looking at the categorical predictor, State was a degenerate predictor. It shows little to no variability across different observations. This predictor was removed before modelling.

Histogram for Numerical predictors



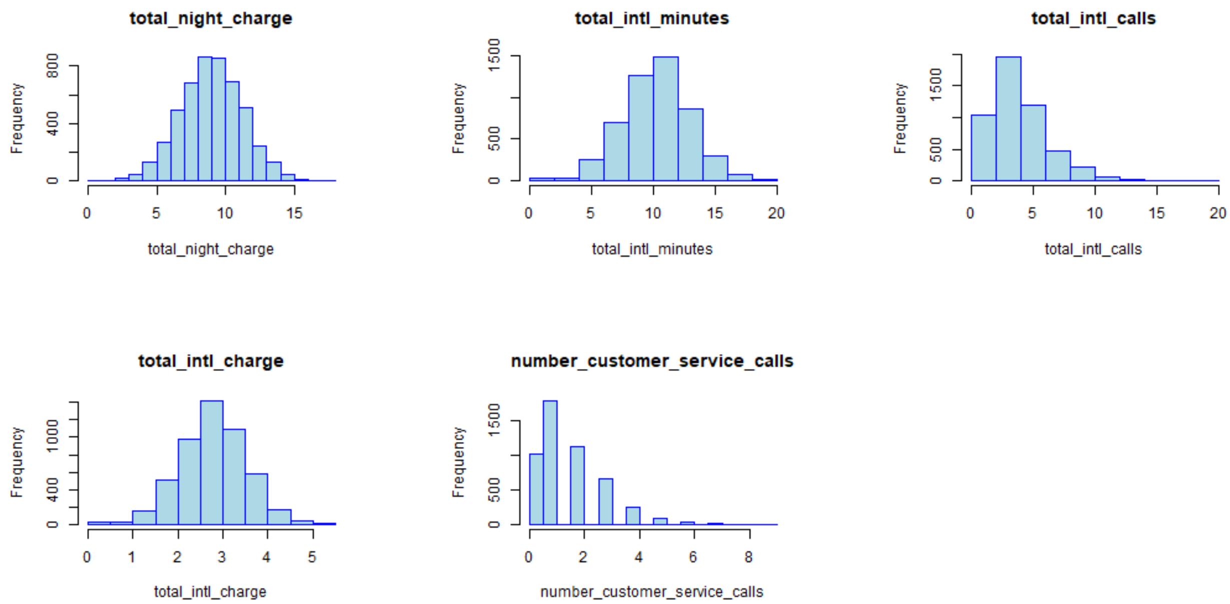
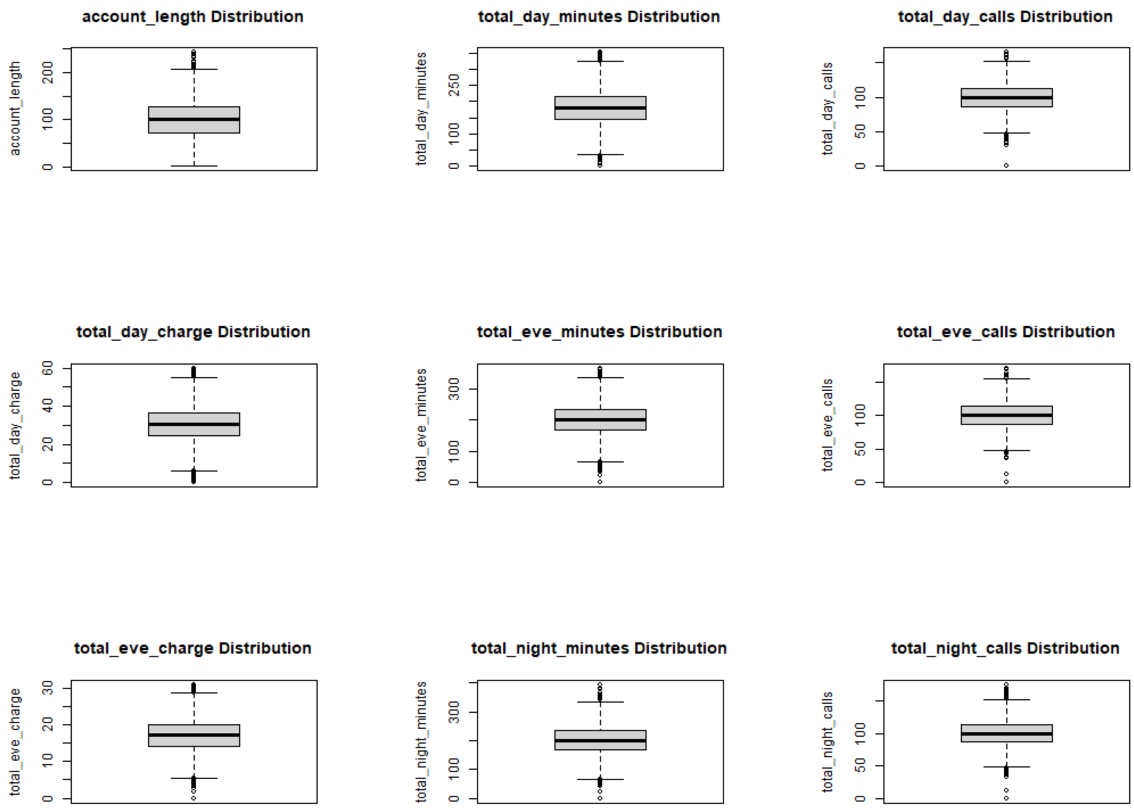


Fig 8: Histogram plots for numerical predictors

The histogram plots show the distribution of the numerical predictors. From the plots, some numerical predictors had skewness and the need to remove the skewness was evident.

Boxplot distribution for Numerical predictors to check on outliers



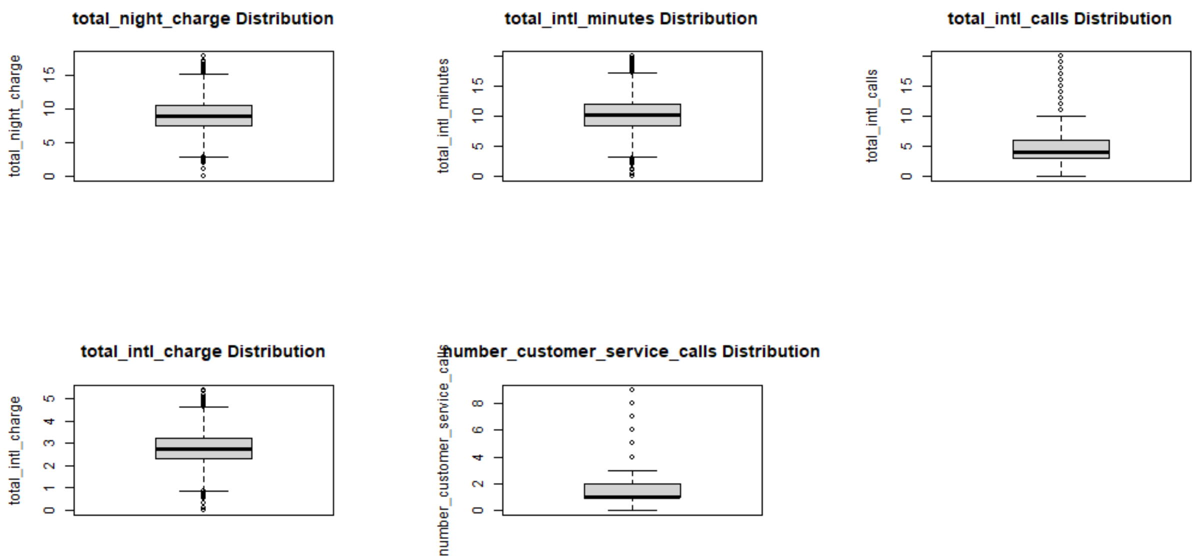


Fig 9: Boxplot distribution for numerical predictors

From the boxplots, some numerical predictors had outliers.

Correlation

```
> highCorr
[1]  8 13  7  4
> length(highCorr)
[1] 4
```

Correlation plot for numerical predictors

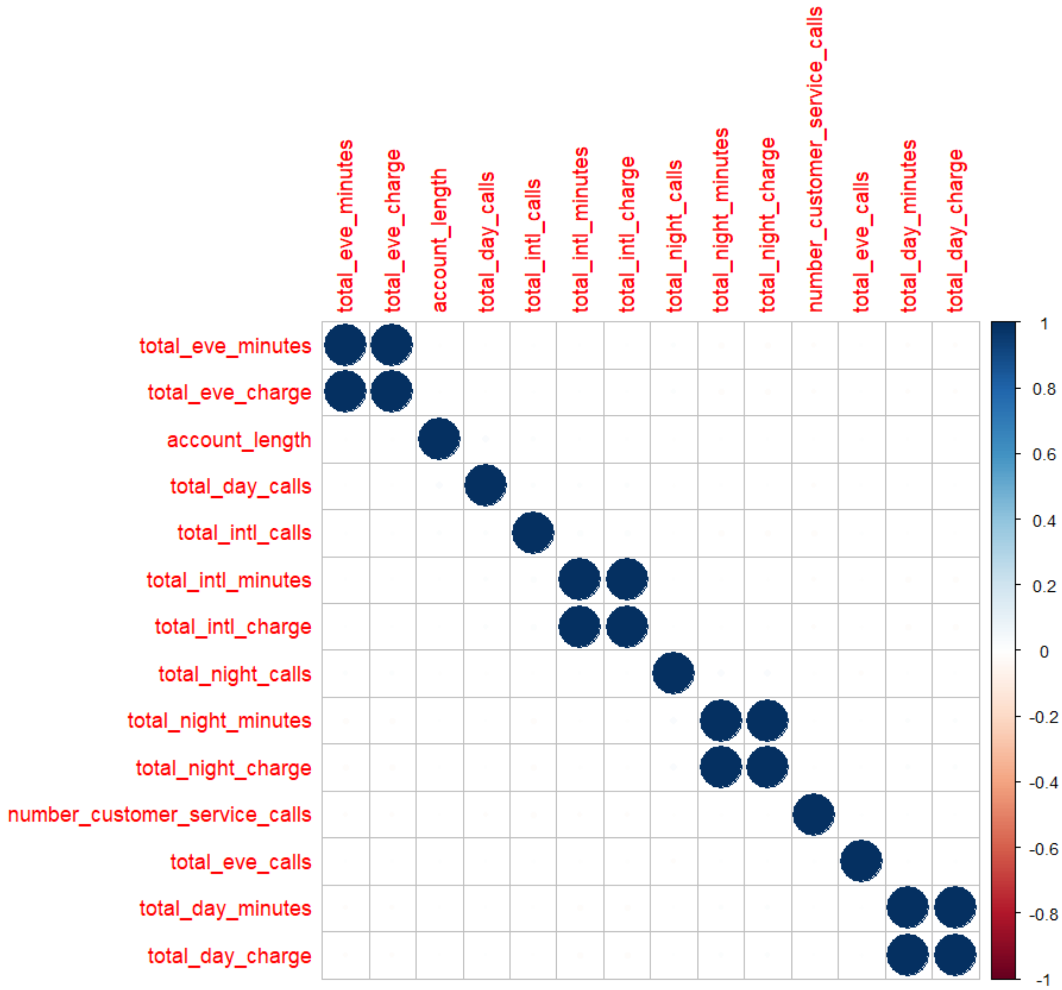


Fig 10: Correlation plots for numerical predictors

Four predictors were highly correlated and the need to remove them was evident.

Correlation plot after removing highly correlated predictors

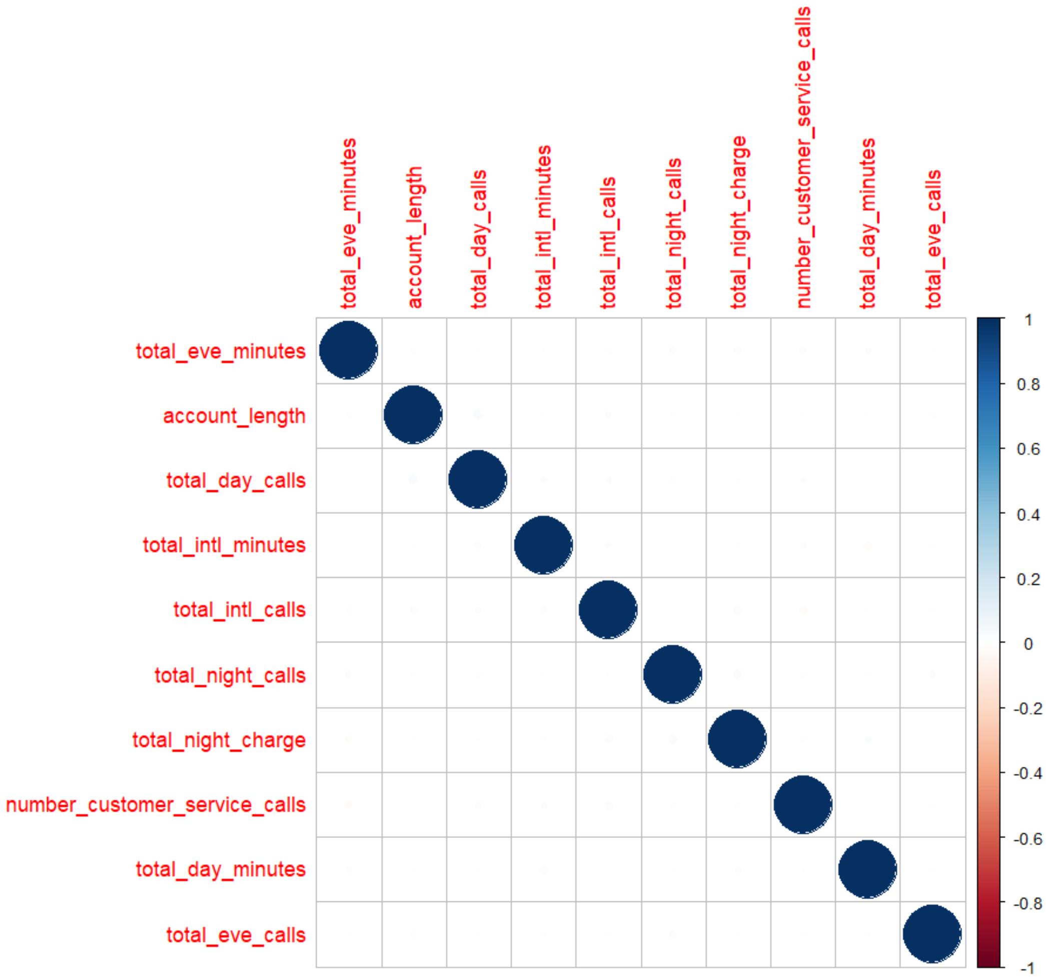


Fig 11: Correlation plot after removing highly correlated predictors

Preprocessing and transformation

```
Created from 5000 samples and 10 variables

Pre-processing:
  - Box-Cox transformation (1)
  - centered (10)
  - ignored (0)
  - scaled (10)
  - spatial sign transformation (10)

Lambda estimates for Box-Cox transformation:
0.9
```

Numerical predictors after transformation

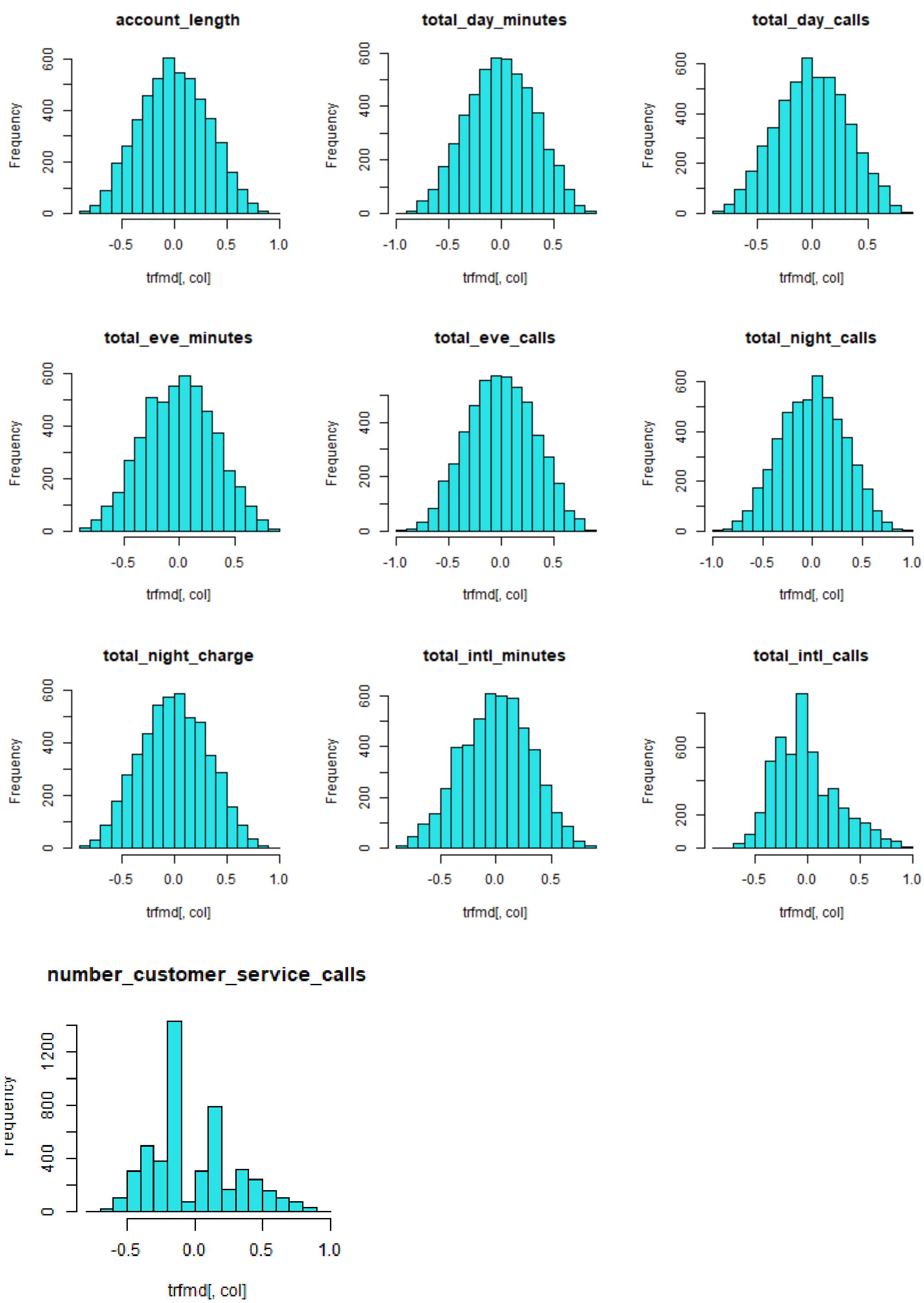


Fig 12: Distribution of numerical predictors after transformation

b) Evaluation of model

Area under the curve(AUC-ROC) was used to evaluate the effectiveness of the models

c) Training and testing models

Logistic regression

```
Generalized Linear Model

4001 samples
  14 predictor
   2 classes: 'yes', 'no'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 3002, 3002, 3002, 3002, 3002, 3002, ...
Resampling results:

  ROC        Sens      Spec
  0.8085635  0.177305  0.9765035
```

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  25  19
       no  116 839

               Accuracy : 0.8649
                 95% CI : (0.8421, 0.8855)
    No Information Rate : 0.8589
    P-Value [Acc > NIR] : 0.3115

                  Kappa : 0.2178

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.17730
            Specificity : 0.97786
         Pos Pred Value : 0.56818
         Neg Pred Value : 0.87853
             Prevalence : 0.14114
         Detection Rate : 0.02503
   Detection Prevalence : 0.04404
      Balanced Accuracy : 0.57758

       'Positive' Class : yes
```

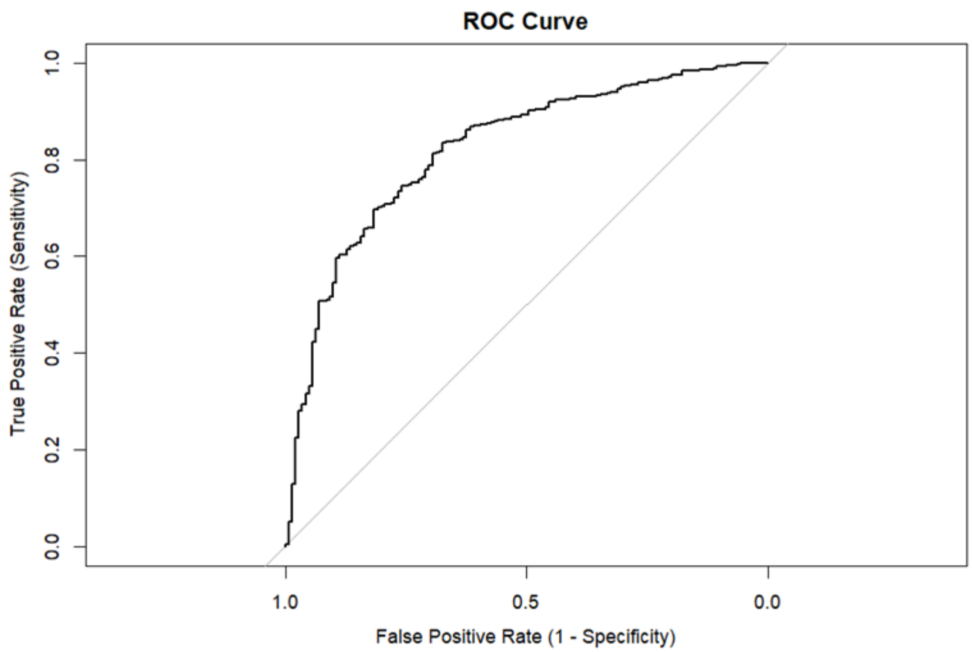


Fig 13: AUC-ROC curve for Logistic regression after prediction

Linear Discriminant Analysis

```
Linear Discriminant Analysis

4001 samples
  14 predictor
   2 classes: 'yes', 'no'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 3002, 3002, 3002, 3002, 3002, 3002, ...
Resampling results:

  ROC        Sens      Spec
  0.8110926  0.218156  0.9627506
```

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  34  25
       no  107 833

               Accuracy : 0.8679
                 95% CI : (0.8453, 0.8883)
    No Information Rate : 0.8589
    P-Value [Acc > NIR] : 0.2212

                  Kappa : 0.28

 Mcnemar's Test P-Value : 1.787e-12

            Sensitivity : 0.24113
            Specificity : 0.97086
         Pos Pred Value : 0.57627
         Neg Pred Value : 0.88617
             Prevalence : 0.14114
         Detection Rate : 0.03403
   Detection Prevalence : 0.05906
      Balanced Accuracy : 0.60600

       'Positive' Class : yes
```

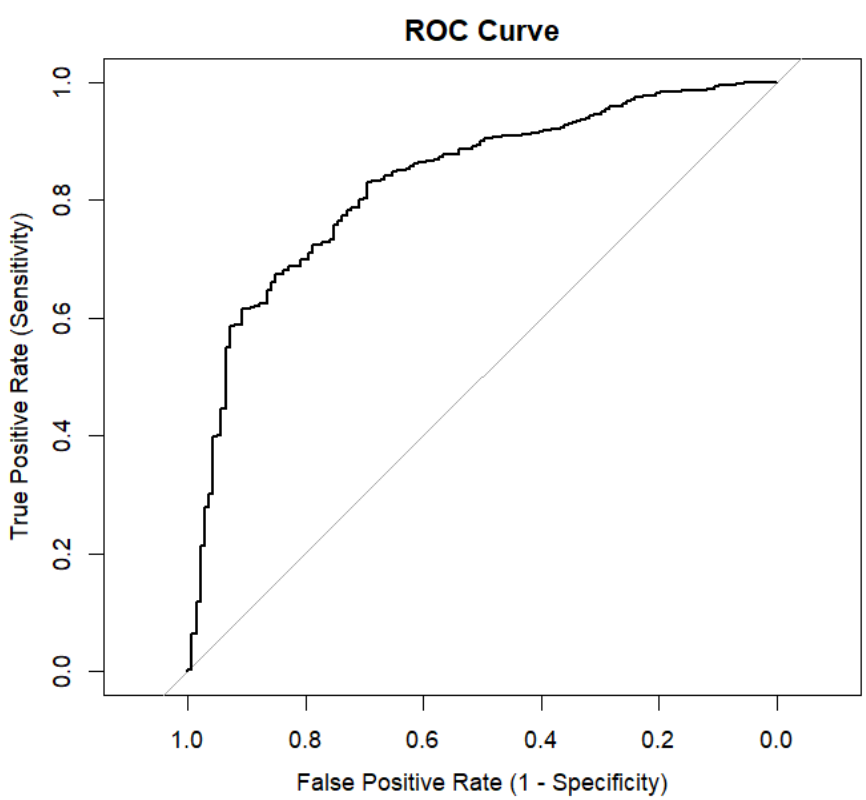


Fig 14: AUC-ROC curve for LDA after prediction

PLSDA

```
Partial Least Squares

4001 samples
  14 predictor
   2 classes: 'yes', 'no'

No pre-processing
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 3002, 3002, 3002, 3002, 3002, 3002, ...
Resampling results across tuning parameters:

  ncomp  ROC        Sens        Spec
   1     0.8014561  0.04283688  0.9950583
   2     0.8103629  0.06893617  0.9913287
   3     0.8109812  0.07347518  0.9902564
   4     0.8110840  0.08028369  0.9895105
   5     0.8110909  0.08028369  0.9895105
   6     0.8110916  0.08028369  0.9895105
   7     0.8110926  0.08028369  0.9895105
   8     0.8110926  0.08028369  0.9895105
   9     0.8110926  0.08028369  0.9895105
  10     0.8110926  0.08028369  0.9895105
  11     0.8110926  0.08028369  0.9895105
  12     0.8110926  0.08028369  0.9895105
  13     0.8110926  0.08028369  0.9895105
  14     0.8110926  0.08028369  0.9895105

ROC was used to select the optimal model using the largest value.
The final value used for the model was ncomp = 7.
```

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  15   9
       no  126 849

               Accuracy : 0.8649
                 95% CI : (0.8421, 0.8855)
    No Information Rate : 0.8589
    P-Value [Acc > NIR] : 0.3115

                  Kappa : 0.1468

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.10638
            Specificity : 0.98951
         Pos Pred Value : 0.62500
         Neg Pred Value : 0.87077
             Prevalence : 0.14114
         Detection Rate : 0.01502
   Detection Prevalence : 0.02402
      Balanced Accuracy : 0.54795

       'Positive' Class : yes
```

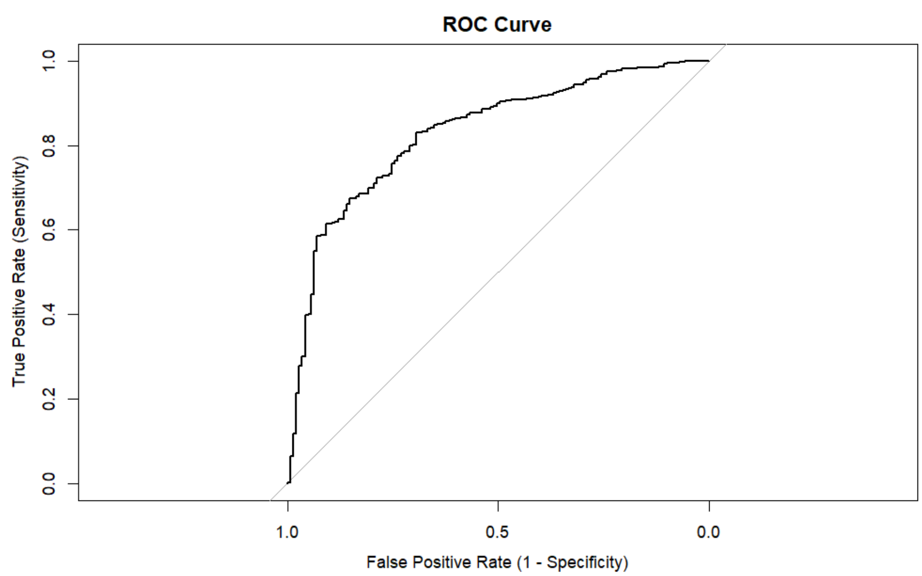


Fig 15: AUC-ROC curve for PLSDA after prediction

GLMNET

```
glmnet

4001 samples
  14 predictor
   2 classes: 'yes', 'no'

Pre-processing: centered (14), scaled (14)
Resampling: Repeated Train/Test Splits Estimated (25 reps, 75%)
Summary of sample sizes: 3002, 3002, 3002, 3002, 3002, 3002, ...
Resampling results across tuning parameters:

  alpha  lambda      ROC        Sens          Spec
  0.0    0.01000000  0.8094001  0.1407092199  0.9829371
  0.0    0.03111111  0.8102475  0.0856737589  0.9893240
  0.0    0.05222222  0.8106373  0.0581560284  0.9926807
  0.0    0.07333333  0.8108304  0.0419858156  0.9960373
  0.0    0.09444444  0.8109871  0.0255319149  0.9975758
  0.0    0.11555556  0.8109828  0.0130496454  0.9987879
  0.0    0.13666667  0.8110060  0.0073758865  0.9994872
  0.0    0.15777778  0.8109785  0.0011347518  0.9996270
  0.0    0.17888889  0.8109451  0.0000000000  0.9998135
  0.0    0.20000000  0.8109084  0.0000000000  1.0000000
  0.1    0.01000000  0.8097661  0.1370212766  0.9835431
  0.1    0.03111111  0.8109653  0.0765957447  0.9901166
  0.1    0.05222222  0.8113016  0.0462411348  0.9941725
  0.1    0.07333333  0.8111210  0.0241134752  0.9976690
  0.1    0.09444444  0.8105890  0.0070921986  0.9993473
  0.1    0.11555556  0.8094708  0.0014184397  0.9997203
  0.1    0.13666667  0.8080656  0.0000000000  1.0000000
  0.1    0.15777778  0.8064894  0.0000000000  1.0000000
  0.1    0.17888889  0.8049189  0.0000000000  1.0000000
  0.1    0.20000000  0.8035358  0.0000000000  1.0000000
  0.2    0.01000000  0.8100094  0.1336170213  0.9842890
  0.2    0.03111111  0.8109795  0.0700709220  0.9909557
  0.2    0.05222222  0.8102052  0.0346099291  0.9953380
  0.2    0.07333333  0.8075954  0.0096453901  0.9991608
  0.2    0.09444444  0.8044461  0.0014184397  0.9998135
  0.2    0.11555556  0.8012852  0.0000000000  1.0000000
  0.2    0.13666667  0.7975043  0.0000000000  1.0000000
  0.2    0.15777778  0.7945336  0.0000000000  1.0000000
  0.2    0.17888889  0.7911762  0.0000000000  1.0000000
  0.2    0.20000000  0.7870066  0.0000000000  1.0000000
  0.4    0.01000000  0.8102729  0.1248226950  0.9850350
  0.4    0.03111111  0.8090142  0.0519148936  0.9924476
  0.4    0.05222222  0.8031120  0.0110638298  0.9984615
```

```
  0.4     0.03111111  0.8090142  0.0519148936  0.9924476
  0.4     0.05222222  0.8031120  0.0110638298  0.9984615
  0.4     0.07333333  0.7961259  0.0002836879  1.0000000
  0.4     0.09444444  0.7892751  0.0000000000  1.0000000
  0.4     0.11555556  0.7836820  0.0000000000  1.0000000
  0.4     0.13666667  0.7767690  0.0000000000  1.0000000
  0.4     0.15777778  0.7461396  0.0000000000  1.0000000
  0.4     0.17888889  0.6495169  0.0000000000  1.0000000
  0.4     0.20000000  0.5909002  0.0000000000  1.0000000
  0.6     0.01000000  0.8102263  0.1129078014  0.9860606
  0.6     0.03111111  0.8044884  0.0365957447  0.9943124
  0.6     0.05222222  0.7947462  0.0011347518  0.9998135
  0.6     0.07333333  0.7845906  0.0000000000  1.0000000
  0.6     0.09444444  0.7727855  0.0000000000  1.0000000
  0.6     0.11555556  0.6681506  0.0000000000  1.0000000
  0.6     0.13666667  0.5817125  0.0000000000  1.0000000
  0.6     0.15777778  0.5000000  0.0000000000  1.0000000
  0.6     0.17888889  0.5000000  0.0000000000  1.0000000
  0.6     0.20000000  0.5000000  0.0000000000  1.0000000
  0.8     0.01000000  0.8100967  0.1043971631  0.9869464
  0.8     0.03111111  0.7996961  0.0224113475  0.9962704
  0.8     0.05222222  0.7857829  0.0000000000  1.0000000
  0.8     0.07333333  0.7671200  0.0000000000  1.0000000
  0.8     0.09444444  0.6135949  0.0000000000  1.0000000
  0.8     0.11555556  0.5000000  0.0000000000  1.0000000
  0.8     0.13666667  0.5000000  0.0000000000  1.0000000
  0.8     0.15777778  0.5000000  0.0000000000  1.0000000
  0.8     0.17888889  0.5000000  0.0000000000  1.0000000
  0.8     0.20000000  0.5000000  0.0000000000  1.0000000
  1.0     0.01000000  0.8096652  0.0941843972  0.9875991
  1.0     0.03111111  0.7947905  0.0099290780  0.9986946
  1.0     0.05222222  0.7802476  0.0000000000  1.0000000
  1.0     0.07333333  0.6333762  0.0000000000  1.0000000
  1.0     0.09444444  0.5000000  0.0000000000  1.0000000
  1.0     0.11555556  0.5000000  0.0000000000  1.0000000
  1.0     0.13666667  0.5000000  0.0000000000  1.0000000
  1.0     0.15777778  0.5000000  0.0000000000  1.0000000
  1.0     0.17888889  0.5000000  0.0000000000  1.0000000
  1.0     0.20000000  0.5000000  0.0000000000  1.0000000

ROC was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0.1 and lambda = 0.05222222.
```

```
Confusion Matrix and Statistics

          Reference
Prediction yes  no
       yes  10   3
       no  131 855

               Accuracy : 0.8659
                 95% CI : (0.8432, 0.8864)
    No Information Rate : 0.8589
    P-Value [Acc > NIR] : 0.2798

                  Kappa : 0.1086

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.07092
            Specificity : 0.99650
         Pos Pred Value : 0.76923
         Neg Pred Value : 0.86714
             Prevalence : 0.14114
         Detection Rate : 0.01001
   Detection Prevalence : 0.01301
      Balanced Accuracy : 0.53371

       'Positive' Class : yes
```

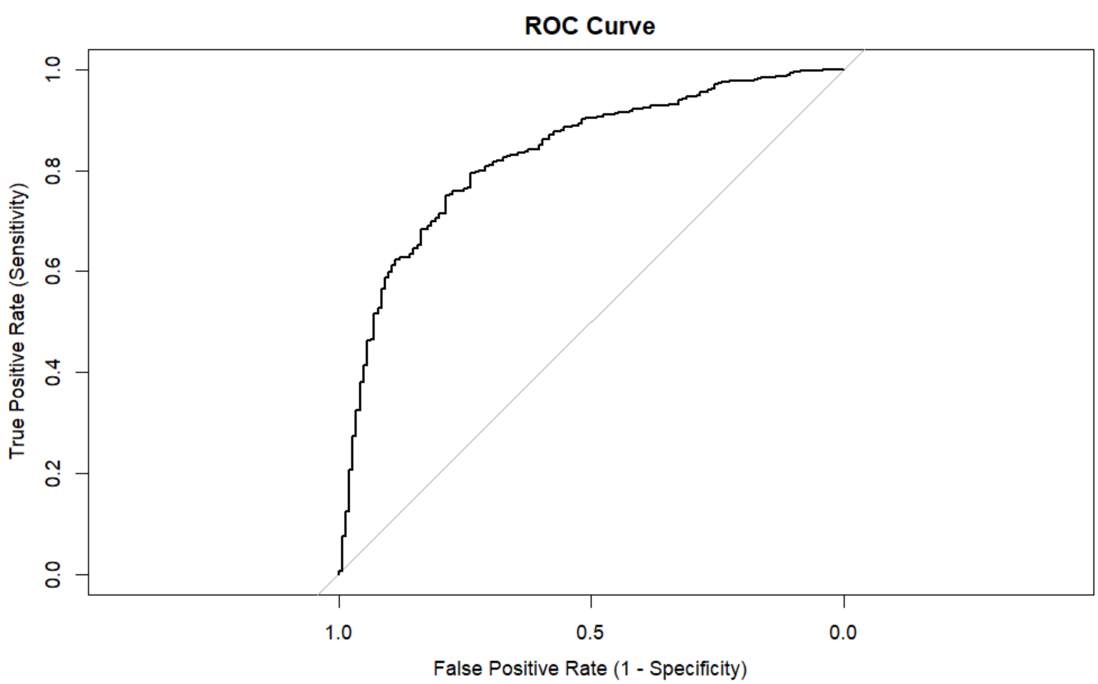


Fig 16: AUC-ROC curve for PLSDA after prediction

Model	Best Tuning Parameter	Training (AUC-ROC)	Testing (AUC-ROC)
Logistic Regression	N//A	0.8085635	0.8211
Linear Discriminant Analysis	N/A	0.8110926	0.8287
PLSDA	ncomp=7	0.8110926	0.8287
glmnet	alpha=0.1, lambda=0.05222	0.8110316	0.8279

From the summary in the table above, PLSDA and LDA have similar results in terms of their performance on the training and testing sets, using area under the curve. The two models have a performance of 81.11% on training and 82.87% on testing. Glmnet is the closest model to the two with a performance of 81.103% on training and 82.79% on testing. The best model would be PLSDA as it is the joint performing model with LDA, but it has a tuning parameter with ncomp=7 as the best tuning parameter for PLSDA. This parameter can be adjusted to improve the performance of the model.

Appendix

```
##Question 12.1

library(caret)
library(AppliedPredictiveModeling)
data(hepatic)
injury
table(injury)
barplot(table(injury), col=c('yellow','green','blue'),main='Class Distribution')

#part c: Nearzero & Corr
bio <- bio[, -nearZeroVar(bio)]
highCorrbio <- findCorrelation(cor(bio), cutoff=0.90)
bio<- bio[, -highCorrbio]

##split the data 80/20
install.packages("MLmetrics")

set.seed(100)
trainR <- createDataPartition(injury, p=0.8, list=FALSE)
X.train <- bio[trainR, ]
y.train <- injury[trainR]
X.test <- bio[-trainR, ]
y.test <- injury[-trainR]

ctrl<-trainControl(method='LGOCV',summaryFunction = multiClassSummary,classProbs =
TRUE)
```

```
##Running models
#install.packages(c("glmnet", "pamr", "rms", "sparseLDA", "subselect"))

##########Logistic Regression############
X.train <- as.data.frame(X.train)


set.seed(100)
bio_lr <- caret::train(X.train,
            y.train,
            method ='multinom',
            metric = "Kappa",
            trControl = ctrl,
            )
bio_lr
plot(bio_lr)
pred_bio<-predict(bio_lr,X.test)

confusionMatrix(data=pred_bio,
             reference=y.test)
reducedRoc <- roc(response = lrReduced$pred$obs,
                predictor = lrReduced$pred$successful,
                levels = rev(levels(lrReduced$pred$obs)))
plot(reducedRoc, legacy.axes = TRUE)
auc(reducedRoc)

######Linear Discriminant Analysis##########

library(MASS)
```

```
set.seed(100)


bio_lda <- caret::train(x = X.train,
          y = y.train,
          method = "lda",
          metric = "Kappa",
          trControl = ctrl)
bio_lda

pred_bio2 <- predict(bio_lda,X.test)
confusionMatrix(data =pred_bio2,
          reference = y.test)

######PLSDA####
set.seed(100)
plsBio <- caret::train(x = X.train,
          y = y.train,
          method = "pls",
          tuneGrid = expand.grid(.ncomp = 1:15),
          preProc = c("center","scale"),
          metric = "Kappa",
          trControl = ctrl)
plsBio
plot(plsBio)
predictionPLSBio <-predict(plsBio,X.test)
confusionMatrix(data =predictionPLSBio,
          reference =y.test)
```

```
##########Penalized Logistic Regression############
glmnGrid <- expand.grid(.alpha = c( .1, .2, .4, .6, .8, 1),
                        .lambda = seq(0, 2, length = 10))
require(caret)
set.seed(100)
bio_plr <- caret::train(X.train,
                         y.train,
                         method = "glmnet",
                         metric='Kappa',
                         tuneGrid = glmnGrid,
                         preProc = c("center", "scale"),
                         trControl = ctrl)
bio_plr
plot(bio_plr, main='Penalized Logistic Tuning Parameters')
pred_bio4<-predict(bio_plr,X.test)

confusionMatrix(data=pred_bio4,
                reference=y.test)

imp<-varImp(bio_plr, scale=FALSE)
plot(imp,top=5)

### Question 12.3 ####
#install.packages("caret")
#install.packages('e1071')
library(caret)
```

```
#install.packages("modeldata")
#install.packages("generics")
library(generics)
#install.packages("tidyselect")
library(tidyselect)
library(modeldata)
data("mlc_churn")

#####NearZeroVariance######

nZv <- nearZeroVar(mlc_churn)
length(nZv)

mlc_churn<-mlc_churn[,-nZv]
dim(mlc_churn)
###Separating predictors from the response####
churn1 <- mlc_churn[,-19]
str(churn1)

####Separating categorical from numerical variables
churn_cat <- churn1[,c(3,4,5)]
churn_num <- churn1[,-c(1,3,4,5)]
str(churn_cat)
str(churn_num)

# response... barplot
counts <- table(mlc_churn$churn)
counts
```

```
percentage <- prop.table(counts) * 100
percentage

bp <- barplot(counts,
              names.arg = c("(Yes)", "(No)"),
              col = c("blue", "red"),
              main = "Churn Distribution",
              xlab = "Churn",
              ylab = "Count",
              ylim = c(0, max(counts) + 10))  # Adjust y-axis limits to make space for labels

# cat... barplot
par(mfrow = c(3, 3), pin = c(2, 1))
for (col in 1:ncol(churn_cat)) {
  # Count the frequency of each category
  freq <- table(churn_cat[, col])
  barplot(freq,
          col='lightblue',
          main = colnames(churn_cat)[col],
          xlab = colnames(churn_cat)[col],
  )  # Rotates axis labels for readability
}


# num ... histogram
par(mfrow = c(3, 3), pin = c(2, 1))
for (col in 1:ncol(churn_num)) {
  col_data <- churn_num[[col]]  # Changed from churn_num[, col] to churn_num[[col]]
```

```
  if (all(is.finite(col_data))) {  # Check if all values are finite
    hist(col_data,
         col='lightblue',
         border='blue',
         main = colnames(churn_num)[col],
         xlab = colnames(churn_num)[col])
  } else {
    cat("Skipping column", colnames(churn_num)[col], "due to non-finite values\n")
  }
}

##outliers plot

par(mfrow = c(1, 1))

#Boxplots for all numeric variables
#numeric_data <- mlc_churn[sapply(mlc_churn, is.numeric)]
plots_per_page <- 9
for (i in seq(1, ncol(churn_num), by = plots_per_page)) {
  par(mfrow = c(3, 3))
  for (j in i:min(i+plots_per_page-1, ncol(churn_num))) {
    boxplot(churn_num[[j]],
            main = paste(names(churn_num)[j], "Distribution"),
            ylab = names(churn_num)[j])
  }
}
# corelation [cat + num]
# corr plot
```

```
correlation_matrix <- cor(churn_num)
correlation_matrix

library(corrplot)
corrplot(correlation_matrix, order = "hclust")
corrplot



#Highly correlated predictors
highCorr = findCorrelation( cor( churn_num), cutoff=0.9 )
length(highCorr)
churn_num1= churn_num[,-highCorr]
str(churn_num1)

#correlation after
cor_matrix_2 <- cor(churn_num1)
cor_matrix_2

library(corrplot)
par(mfrow = c(1,1), pin=c(2,1))
corrplot(cor_matrix_2, order = "hclust")

churn_num2 <- as.data.frame(churn_num1)

trans <- preProcess(churn_num2, method = c("BoxCox", "center", "scale", "spatialSign"))
trans

# boxplot
```

```
trfmd <- predict(trans, churn_num2)
head(trfmd)

par(mfrow = c(3, 3))
for (col in 1:ncol(trfmd)) {
  hist(trfmd[,col],
       main = colnames(trfmd)[col],
       col = 5)
}

dummRes <- dummyVars("~area_code + international_plan + voice_mail_plan",
                     data = churn_cat,
                     fullRank = TRUE)
dummy <- data.frame(predict(dummRes, newdata = churn_cat))
dummy
dim(dummy)



churn_c <- cbind(dummy, trfmd)
dim(churn_c)
str(churn_c)
###splitting####

set.seed(100)
trainR <- createDataPartition(mlc_churn$churn, p=0.8, list=FALSE)
X.train <- churn_c[trainR, ]
y.train <- mlc_churn$churn[trainR]
X.test <- churn_c[-trainR, ]
```

```
y.test <- mlc_churn$churn[-trainR]



ctrl<-trainControl(summaryFunction = twoClassSummary,classProbs = TRUE,
                   method='LGOCV',savePredictions = TRUE)

##########Logistic Regression############
require(caret)
set.seed(100)
churn_lr <- caret::train(X.train,
                  y.train,
                  method = "glm",
                  metric = "Kappa",
                  trControl = ctrl)


churn_lr
pred_churn<-predict(churn_lr,X.test)


confusionMatrix(data=pred_churn,
                reference=y.test)
# Predict probabilities on the test set
prob_predictions <- predict(churn_lr, X.test, type = "prob")[, 2]

# Compute the ROC curve
roc_obj <- roc(y.test, prob_predictions)

# Plot the ROC curve
plot(roc_obj, main = "ROC Curve", xlab = "False Positive Rate (1 - Specificity)", ylab = "True Positive Rate (Sensitivity)")
```

```
auc_value <- auc(roc_obj)

# Print the AUC value
print(auc_value)



######Linear Discriminant Analysis##########

library(MASS)
set.seed(100)


churn_lda <- caret::train(x = X.train,
               y = y.train,
               method = "lda",
               metric = "Kappa",
               trControl = ctrl)
churn_lda

pred_churn2 <- predict(churn_lda,X.test)
confusionMatrix(data =pred_churn2,
            reference = y.test)
# Predict probabilities on the test set
prob_predictions2 <- predict(churn_lda, X.test, type = "prob")[, 2]

# Compute the ROC curve
roc_obj2 <- roc(y.test, prob_predictions2)
```

```
# Plot the ROC curve

plot(roc_obj2, main = "ROC Curve", xlab = "False Positive Rate (1 - Specificity)", ylab = "True Positive Rate (Sensitivity)")

auc_value2 <- auc(roc_obj2)


# Print the AUC value

print(auc_value2)



######PLSDA##########


library(MASS)

set.seed(100)



churn_plsda <- caret::train(x = X.train,

                    y = y.train,

                    method = "pls",

                    tuneGrid = expand.grid(

                      .ncomp = 1:14),

                    metric = "Kappa",

                    trControl = ctrl)

churn_plsda

plot(churn_plsda, main='PLSDA tuning parameter')

pred_churn3 <- predict(churn_plsda,X.test)

confusionMatrix(data =pred_churn3,

            reference = y.test)
```

```
# Predict probabilities on the test set
prob_predictions3 <- predict(churn_plsda, X.test, type = "prob")[, 2]

# Compute the ROC curve
roc_obj3 <- roc(y.test, prob_predictions3)

# Plot the ROC curve
plot(roc_obj3, main = "ROC Curve", xlab = "False Positive Rate (1 - Specificity)", ylab = "True Positive Rate (Sensitivity)")
auc_value3 <- auc(roc_obj3)

# Print the AUC value
print(auc_value3)




##########Penalized Logistic Regression############
glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .4, .6, .8, 1),
                        .lambda = seq(.01, .2, length = 10))
require(caret)
set.seed(100)
churn_plr <- caret::train(X.train,
                          y.train,
                          method = "glmnet",
                          metric='ROC',
                          tuneGrid = glmnGrid,
                          preProc = c("center", "scale"),
                          trControl = ctrl)
churn_plr
```

```
plot(churn_plr, main='Penalized Logistic Tuning Parameters')
pred_churn4<-predict(churn_plr,X.test)

confusionMatrix(data=pred_churn4,
                reference=y.test)



library(pROC)

# Predict probabilities on the test set
prob_predictions4 <- predict(churn_plr, X.test, type = "prob")[, 2]

# Compute the ROC curve
roc_obj4 <- roc(y.test, prob_predictions4)

# Plot the ROC curve
plot(roc_obj4, main = "ROC Curve", xlab = "False Positive Rate (1 - Specificity)", ylab = "True Positive Rate (Sensitivity)")
auc_value4 <- auc(roc_obj4)

# Print the AUC value
print(auc_value4)
```