Lab2_Block1

Sreenand.S

08/12/2019

Lab 2 Block 1

Data Setup

A2.1-Deriving the dataset

```
RNGversion("3.5.1")
```

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```

```
creditscoring<-read_excel("creditscoring.xls")
n = dim(creditscoring)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = creditscoring[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = creditscoring[id2,]
id3 = setdiff(id1,id2)
test = creditscoring[id3,]
creditscoring$good_bad=as.factor(creditscoring$good_bad)
```

The dataset is derived and split into 50% as training set and the rest as validation and test set. ## A2.2:Gini and Deviance Index

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```

```
##
## predict_tree_test bad good
##              bad   28   19
##              good  48  155
```

```
##
## predict_tree_train bad good
##               bad   61   20
##               good  86  333
```

```
##
##  The Deviance index train data missclassification rate is  0.212
```

```
##
##  The Deviance index test data missclassification rate is  0.268
```

```
##
## predict_tree_test1 bad good
##               bad   19   33
##               good  57  141
```

```
##
## predict_tree_train1 bad good
##                bad   62   42
##                good  85  311
```

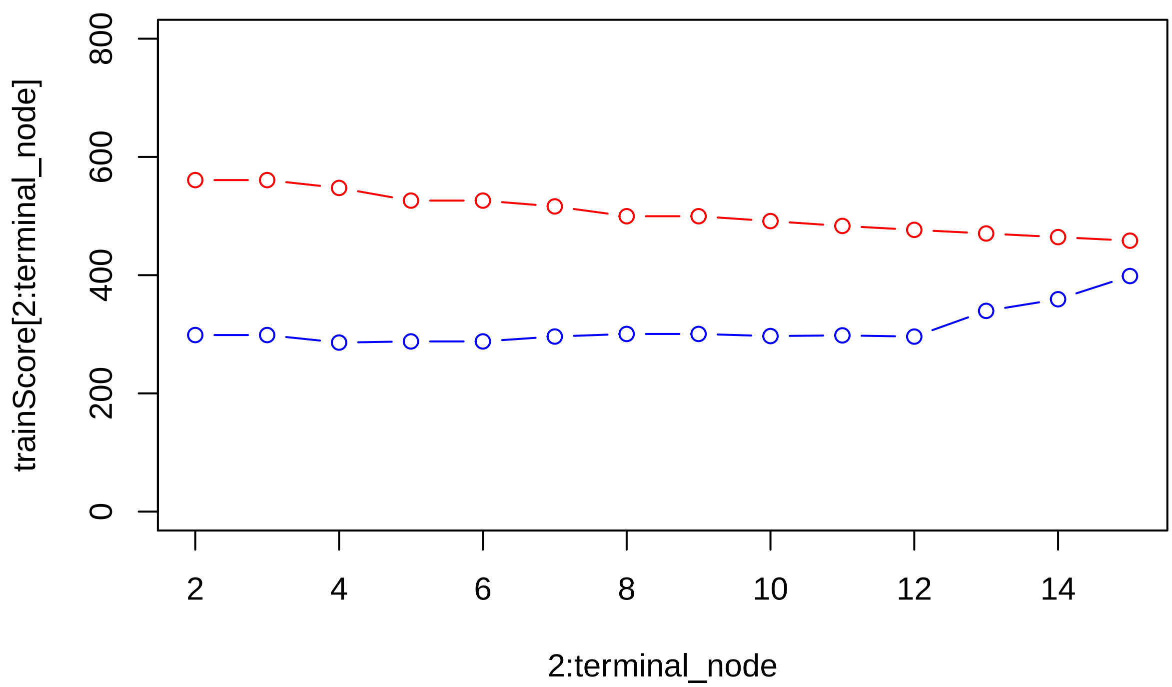
```
##
##  The Gini index train data missclassification rate is  0.254
```

```
##
##  The Gini index test data missclassification rate is  0.36
```

When the deviance index is used there is a misclassification rate of 21.2% when the confusion matrix is built around the train data and when it is used on the test data there is only a slight difference. When the gini index increase of the misclassification rate when it is built on the test data. In this case the tree model seems to give a better fit when deviance is used.

A2.3-Optimal tree

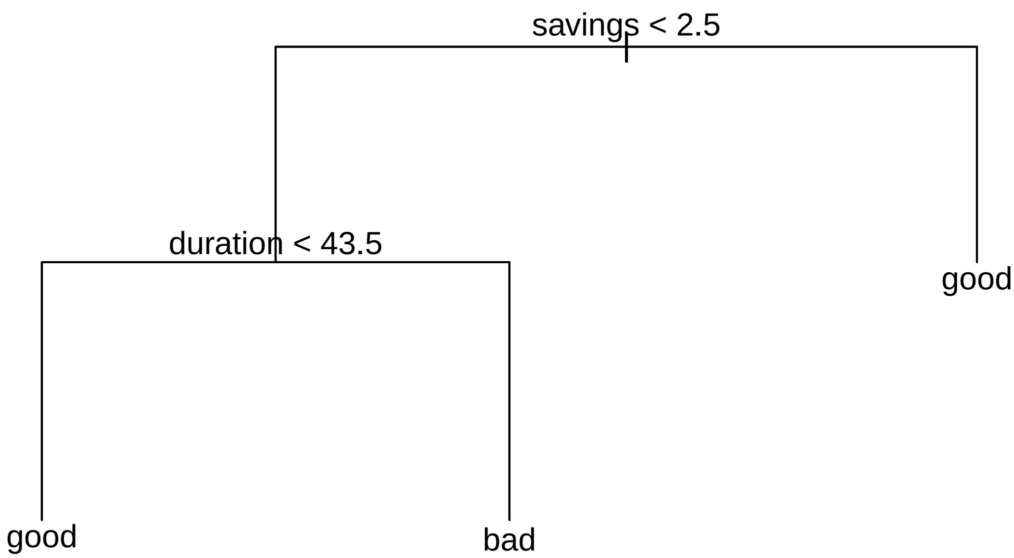
```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```




```
##
##  The minimum deviance is found out to be when the depth of the tree is  3
```

```
##
## Yfit   bad good
##   bad    9    6
##   good  68  167
```

```
## [1] 0.296
```



The optimal tree depth is found out to be 3 which has the least deviance and when it is used the misclassification rate is found out to be 29.6%.The optimal tree structure is displayed above.

A2.4-Naive Bayes Model

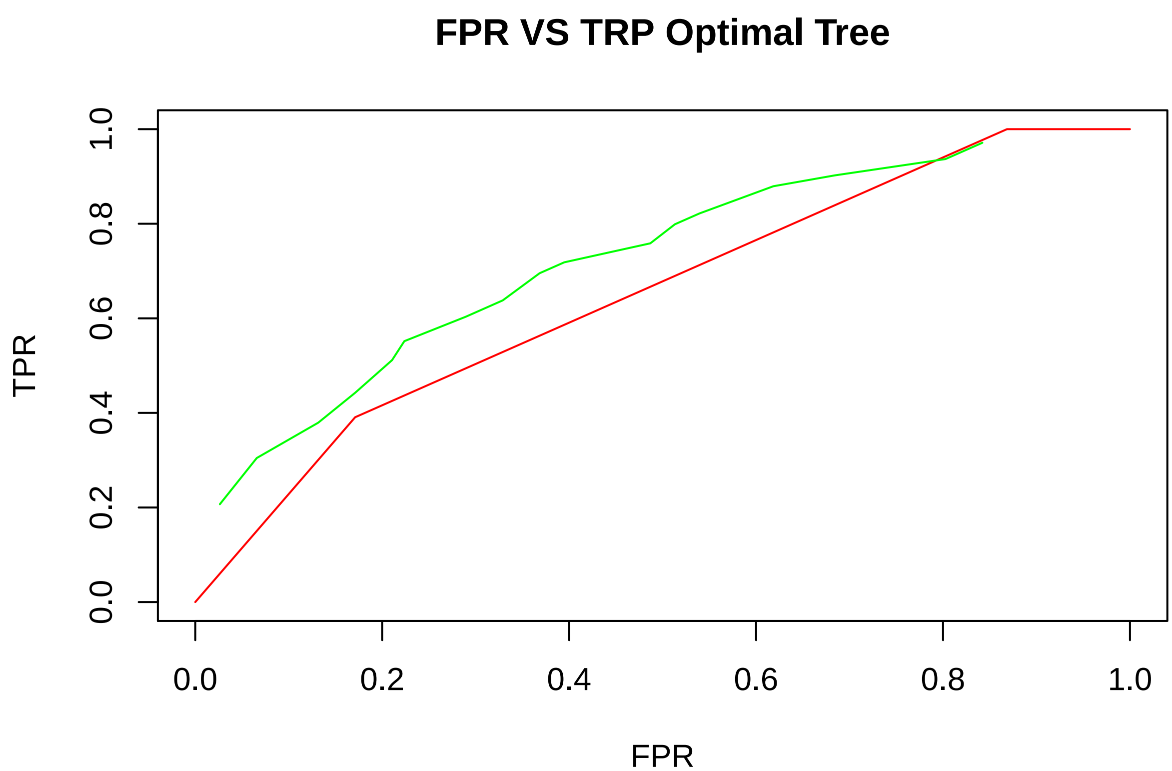
```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used

## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.

##
## predict_naive_bayes_train bad good
##                      bad   95   98
##                      good  52  255

##
##  The misclassification rate when the Naive bayes models is used on train data is  0.3

## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.

##
## predict_naive_bayes_test bad good
##                     bad   46   49
```

```
##                    good  30  125
##
##  The misclassification rate when the Naive bayes models is used on test data is  0.316
```

When Naive bayes Classifier is used on the test data the misclassification rate is found out to be 30% and when predicted on train data it is observed as 31.6%. In conclusion it is seen that it has a higher misclassification rate than compared to the Tree with the best fit of the tree.

A2.5- ROC curve

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used

## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.
```



From the Roc curve above it is seen that there is only a slight change between the naive bayes and the Optimal tree model. Also, naive bayes more area under the curve so hence naive bayes can be thought of as the better classifier.

A2.6-Naive bayes with loss matrix

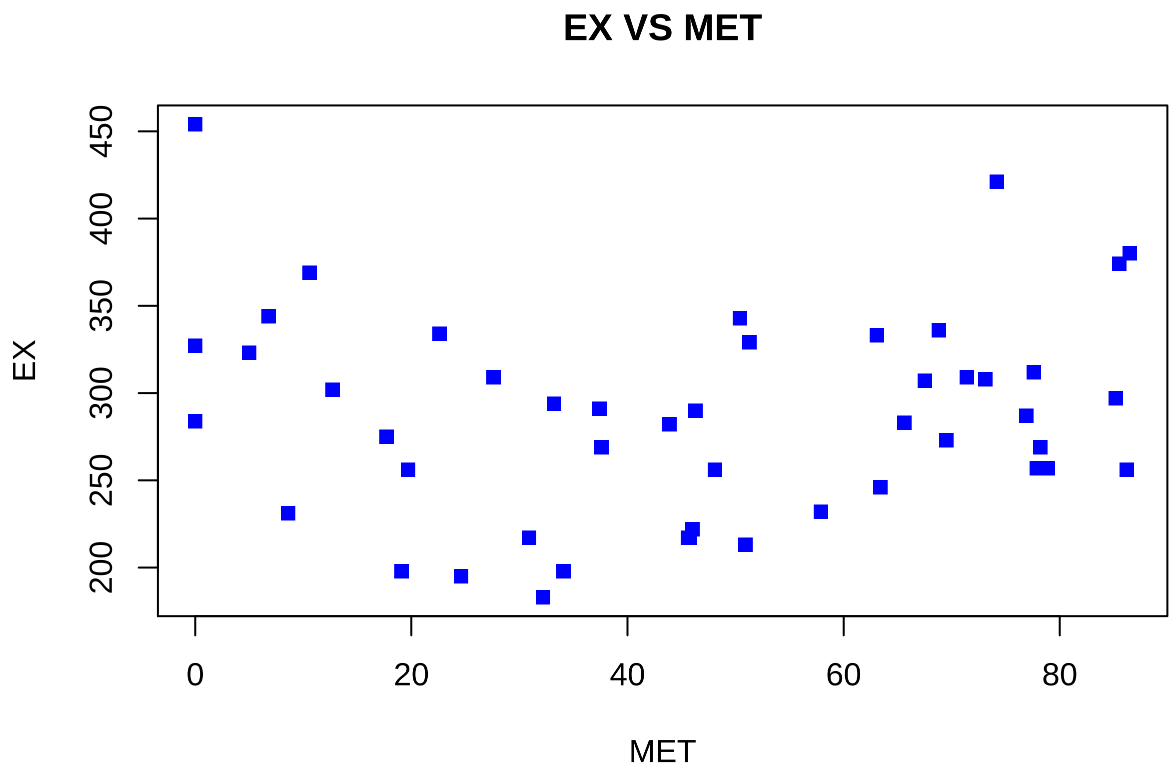
```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used

## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
```

```
## based on features to be found in the tables.

## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.

##
## naive_bayes_train_loss bad good
##                   bad   30   17
##                   good 117  336

##
## naive_bayes_test_loss bad good
##                  bad   15   11
##                  good  61  163

##
##  The misclassification rate when the train data is used is 0.268

##
##  The misclasification rate when the test data is used is  0.288
```

The misclassification rate using the train data is somewhere around 26% whereas the error rate when uesing test data is around 28%.Thus it is much less than when the loss function is not used.

A3.1-Reorder data

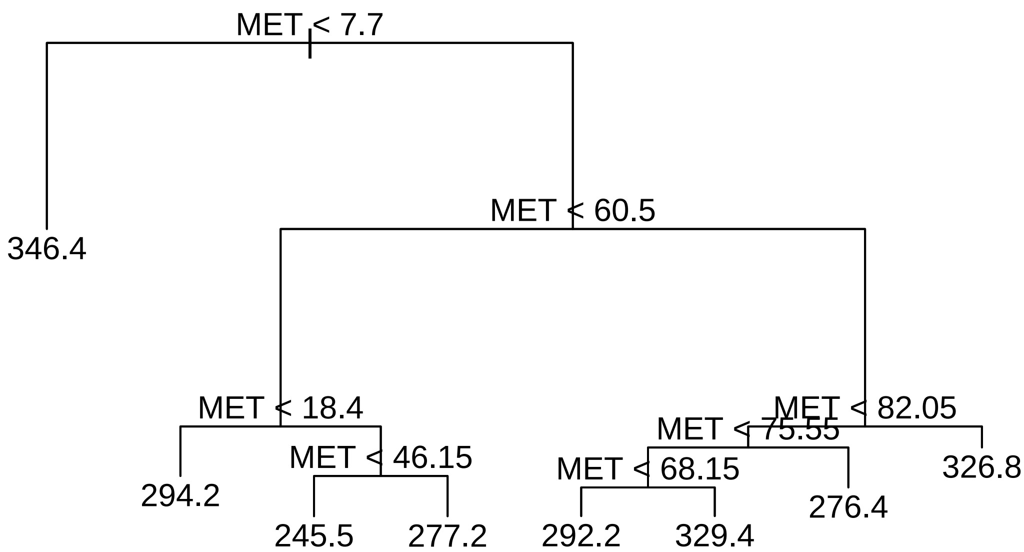
```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used

## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.

## Parsed with column specification:
## cols(
##   EX = col_double(),
##   ECAB = col_double(),
##   MET = col_double(),
##   GROW = col_double(),
##   YOUNG = col_double(),
##   OLD = col_double(),
##   WEST = col_double(),
##   STATE = col_character()
## )
```

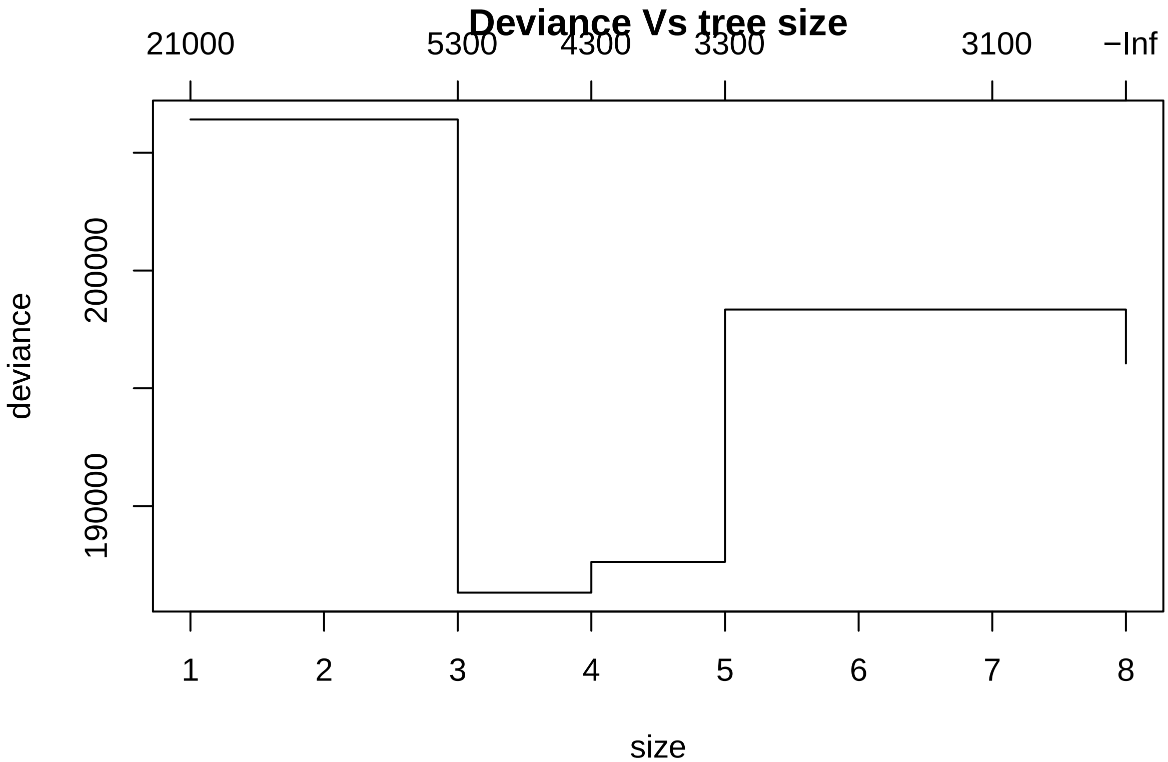


As noted in the above Figure the data is not correlated and hence you cant fit a straight line onto this.The data seems very dispersed and theredore a tree would be an ideal fit for this.

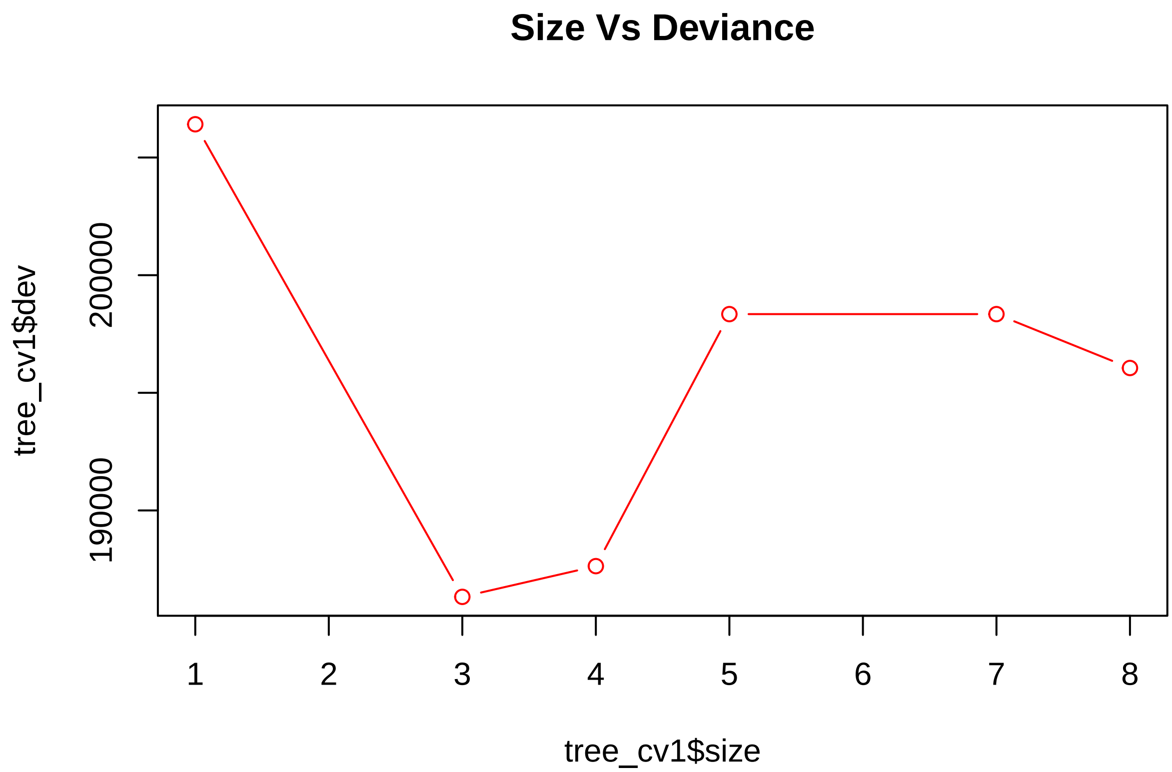
A3.2-Cross Validation

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```

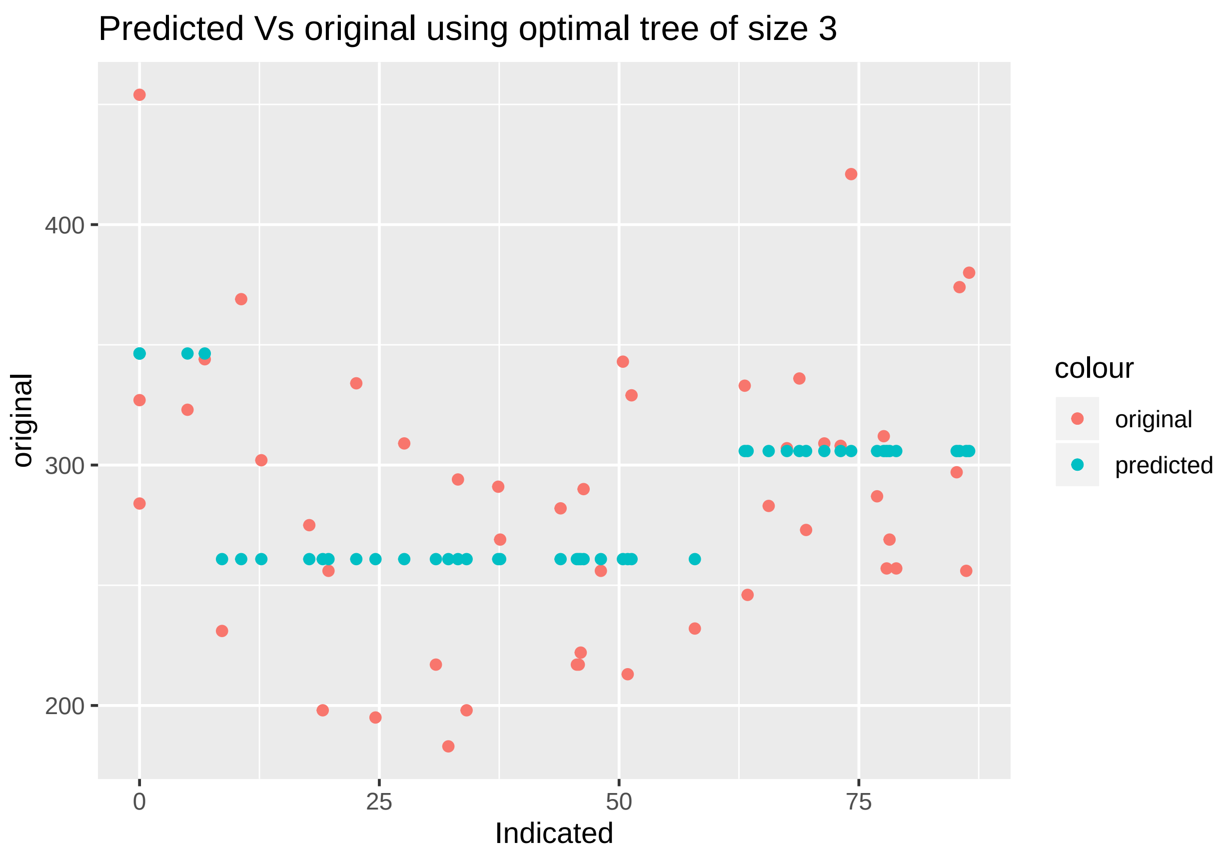

MET < 7.7
MET < 60.5
346.4
MET < 18.4
MET < 82.05
MET < 75.55
MET < 46.15
MET < 68.15
326.8
294.2
276.4
245.5
277.2
292.2
329.4



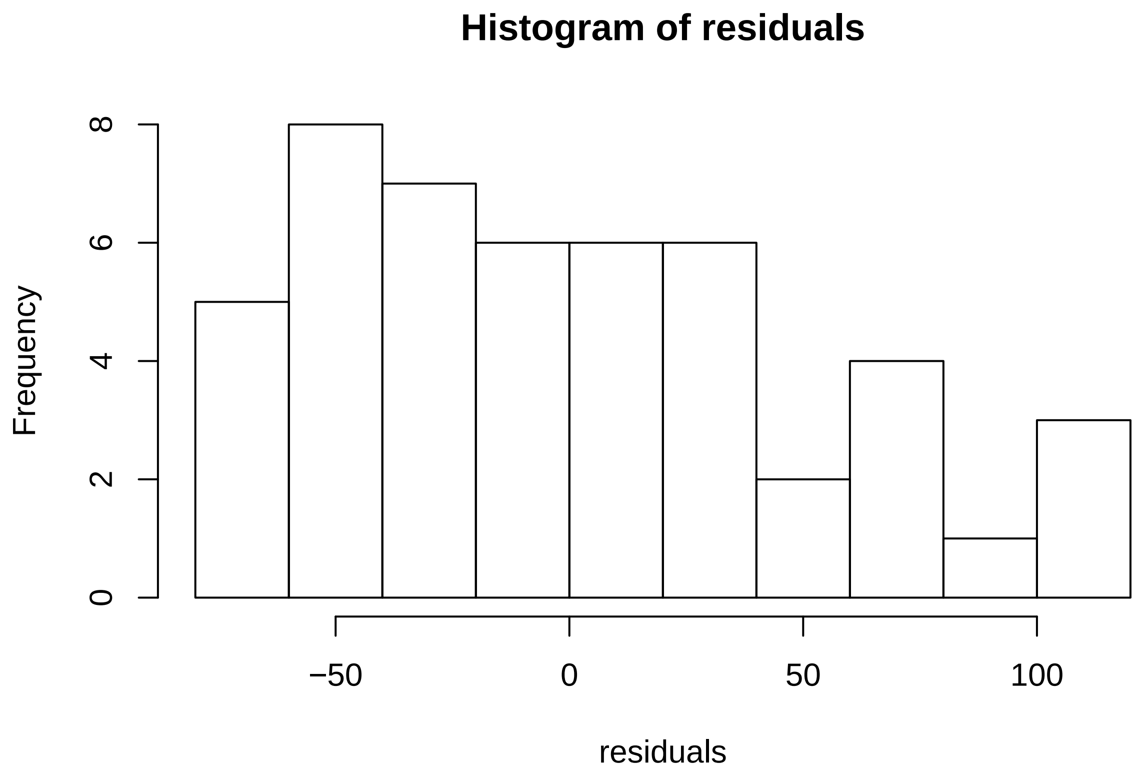
Deviance Vs tree size
21000
5300
4300
3300
3100
−Inf
deviance
200000
190000
1
2
3
4
5
6
7
8
size




```
##
## Optimal tree: 3
```



Predicted Vs original using optimal tree of size 3
original
400
300
200
0
25
50
75
Indicated
colour
original
predicted



The graph shows the least deviance is at point 3 having the deviance as 178460.7.From the histogram it is observed that is tail is towards the right so it is not a good fit. The optimal tree chosn here is 3 since it has the least deviance. The graph given above shows the original and the predicted data when the optimal size of the tree is used. The histogram shows that the residuals are spread across the dataset.

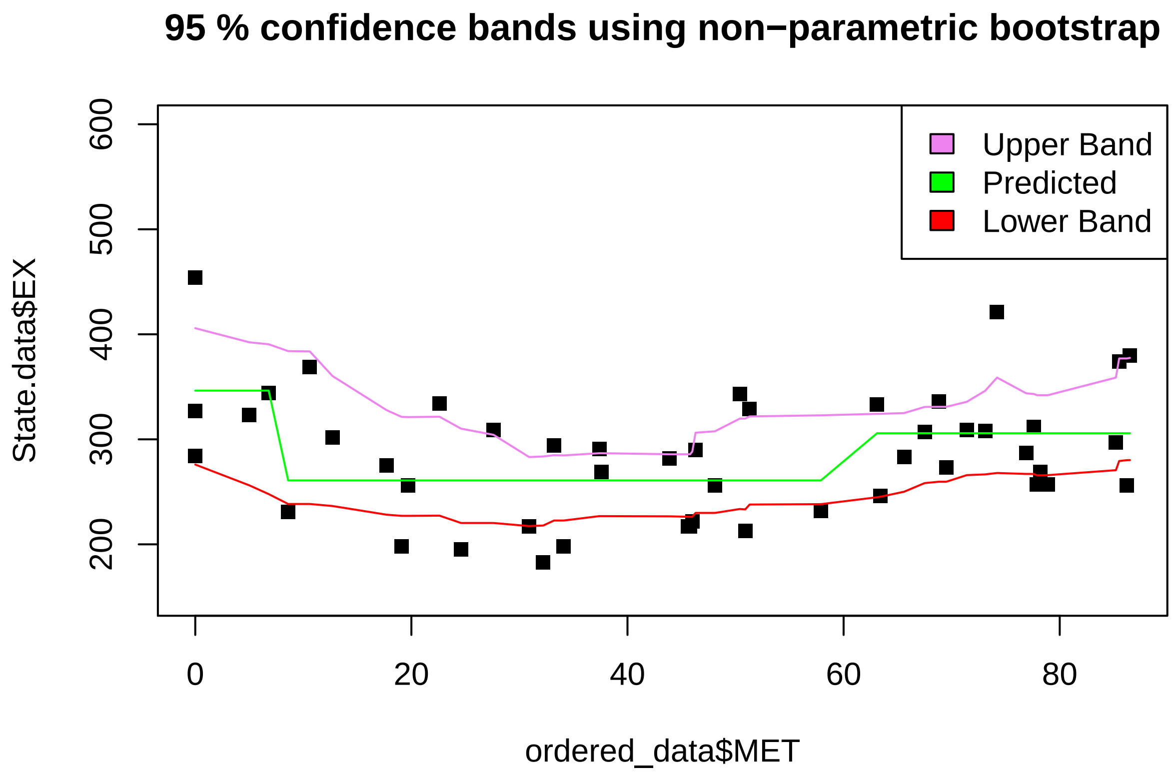
A3.3- Non-Parametric Bootstrap

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```

```
##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:lattice':
##
##     melanoma
```

```
## Warning in prune.tree(trainedmodel, best = Optimal_Size): best is bigger
## than tree size
```

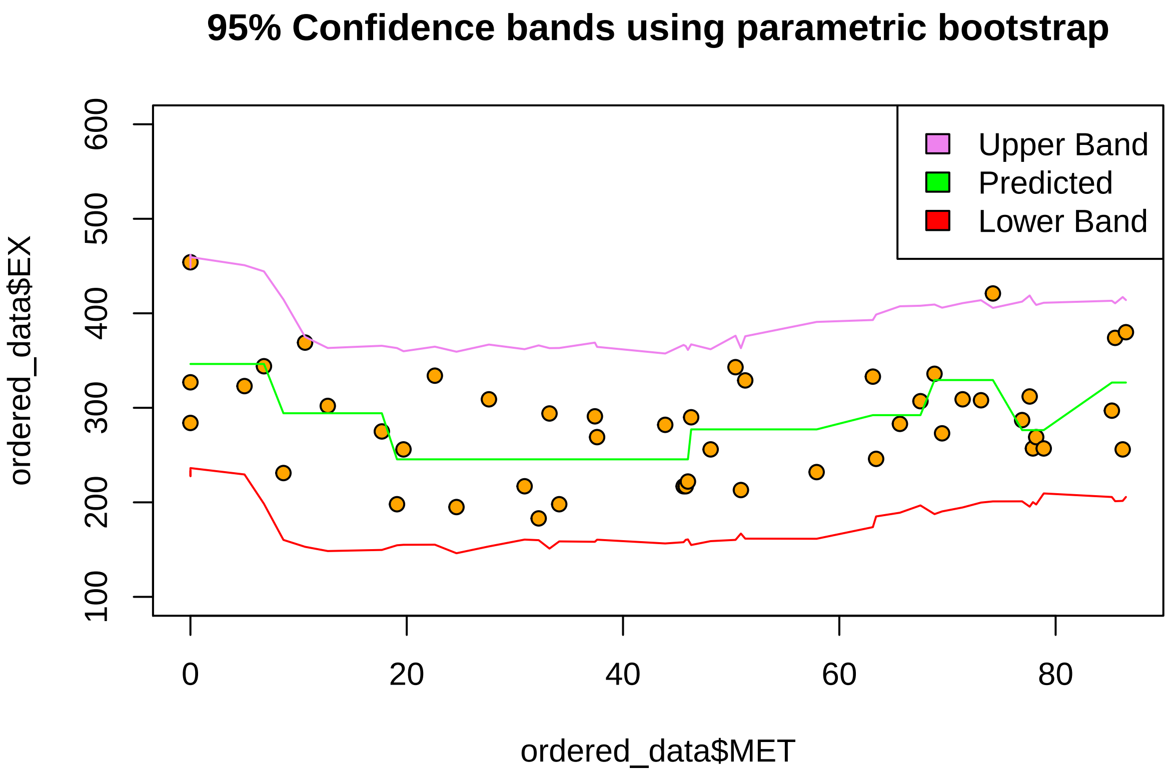


The confidence interval bands are close to each other and the fit is not as good as shown in step 2. The confidence band leaves out most of the data.The confidence bands are bumpy and not smooth.This is because of the bias.

A3.4-Parametric Bootstrap

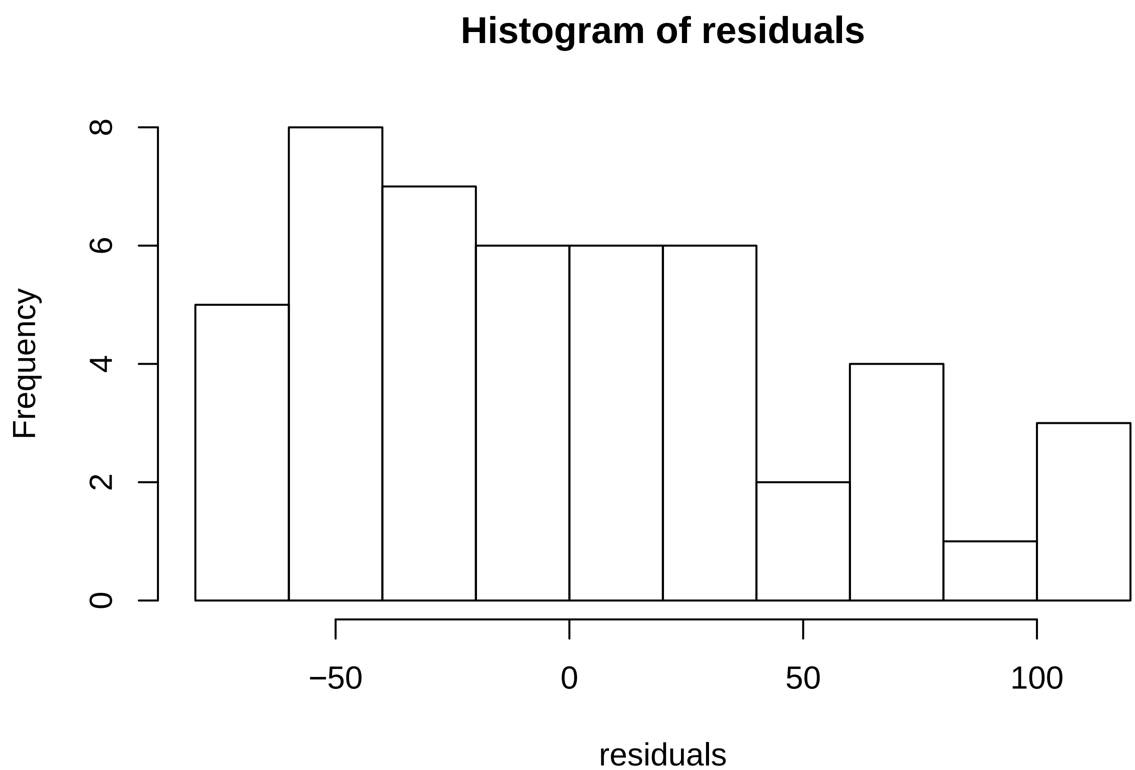
```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```

```
## Warning in envelope(para_boot, level = 0.95): unable to achieve requested
## overall error rate
```



The Graph above shows a higher prediction bandwidth covering most of the data points.Only 5 % of the data is out of the prediction band and this is how it should be as the data should cover 95 % of the data.

A3.5



From looking at the histogram residuals it can said that the non parametric bootstrap model would be more ideal for this type of distribution since it is scattered. The prediction bands for the non-parametric estimation is much lesser than that of the parametric and for the non - prarametric it seems to overfit the data as it covers 99 percent of the data.

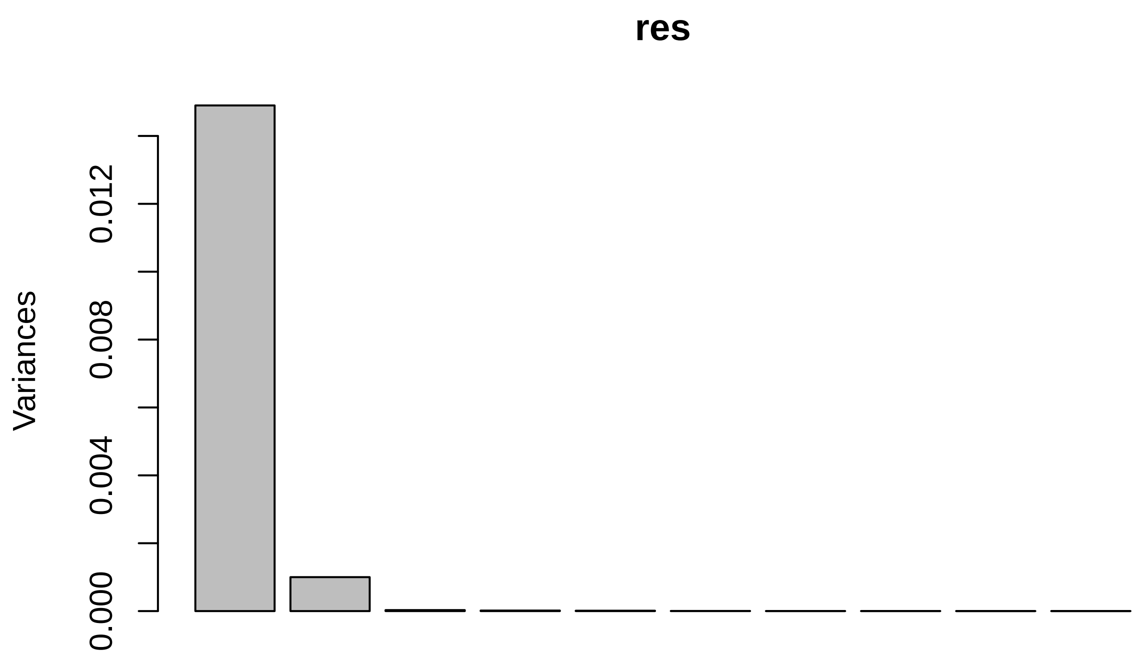
A4.1-Standard PCA

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```

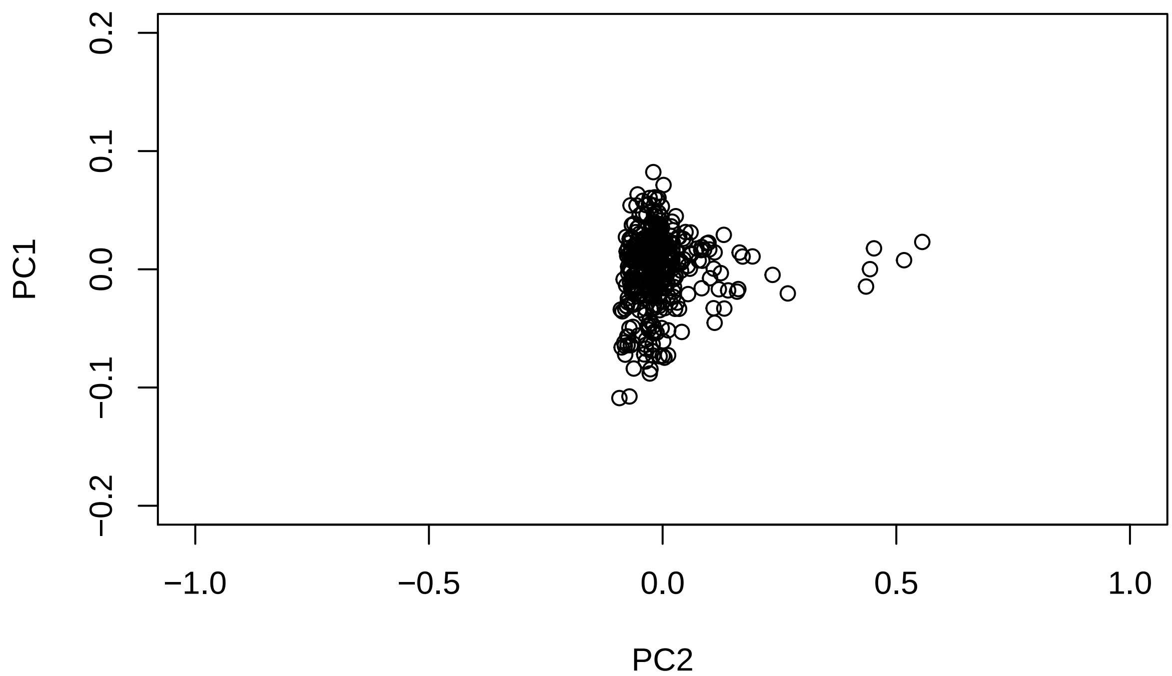
```
## Importance of components:
##                           PC1     PC2      PC3      PC4      PC5      PC6
## Standard deviation     0.1221 0.03163 0.005424 0.004009 0.003304 0.001985
## Proportion of Variance 0.9333 0.06265 0.001840 0.001010 0.000680 0.000250
## Cumulative Proportion  0.9333 0.99590 0.997740 0.998750 0.999430 0.999680
##                             PC7      PC8       PC9      PC10      PC11
## Standard deviation     0.001193 0.001091 0.0006925 0.0006473 0.0004833
## Proportion of Variance 0.000090 0.000070 0.0000300 0.0000300 0.0000100
## Cumulative Proportion  0.999770 0.999840 0.9998700 0.9999000 0.9999100
##                             PC12      PC13      PC14      PC15      PC16
## Standard deviation     0.0004553 0.0003903 0.0003741 0.0003333 0.0002629
## Proportion of Variance 0.0000100 0.0000100 0.0000100 0.0000100 0.0000000
## Cumulative Proportion  0.9999200 0.9999300 0.9999400 0.9999500 0.9999500
##                             PC17     PC18      PC19      PC20      PC21
## Standard deviation     0.0002614 0.000221 0.0001991 0.0001939 0.0001879
```

```
## Proportion of Variance 0.0000000 0.000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  0.9999600 0.999960 0.9999600 0.9999700 0.9999700
##                             PC22      PC23      PC24     PC25      PC26
## Standard deviation     0.0001826 0.0001676 0.0001553 0.000145 0.0001386
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.000000 0.0000000
## Cumulative Proportion  0.9999700 0.9999700 0.9999700 0.999970 0.9999800
##                             PC27      PC28      PC29     PC30      PC31
## Standard deviation     0.0001374 0.0001308 0.0001292 0.000122 0.0001202
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.000000 0.0000000
## Cumulative Proportion  0.9999800 0.9999800 0.9999800 0.999980 0.9999800
##                             PC32      PC33      PC34      PC35     PC36
## Standard deviation     0.0001158 0.0001103 0.0001096 0.0001068 0.000103
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.000000
## Cumulative Proportion  0.9999800 0.9999800 0.9999800 0.9999800 0.999980
##                            PC37      PC38      PC39      PC40      PC41
## Standard deviation     0.000101 9.471e-05 9.303e-05 9.235e-05 8.883e-05
## Proportion of Variance 0.000000 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  0.999990 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC42      PC43      PC44      PC45     PC46
## Standard deviation     8.532e-05 8.372e-05 8.033e-05 8.001e-05 7.85e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.00e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.00e+00
##                             PC47      PC48      PC49      PC50      PC51
## Standard deviation     7.549e-05 7.502e-05 7.407e-05 7.237e-05 7.227e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC52      PC53      PC54      PC55      PC56
## Standard deviation     7.026e-05 6.846e-05 6.772e-05 6.718e-05 6.596e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC57      PC58     PC59      PC60      PC61
## Standard deviation     6.403e-05 6.214e-05 6.15e-05 6.124e-05 5.973e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.00e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.00e+00 1.000e+00 1.000e+00
##                             PC62      PC63      PC64      PC65      PC66
## Standard deviation     5.881e-05 5.794e-05 5.724e-05 5.595e-05 5.557e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC67      PC68      PC69      PC70     PC71
## Standard deviation     5.446e-05 5.368e-05 5.318e-05 5.275e-05 5.17e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.00e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.00e+00
##                             PC72      PC73      PC74      PC75      PC76
## Standard deviation     5.042e-05 4.949e-05 4.903e-05 4.829e-05 4.758e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC77      PC78      PC79      PC80      PC81
## Standard deviation     4.697e-05 4.658e-05 4.519e-05 4.427e-05 4.409e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC82      PC83      PC84      PC85      PC86
## Standard deviation     4.348e-05 4.317e-05 4.247e-05 4.146e-05 4.111e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
```

```
##                              PC87      PC88      PC89      PC90      PC91
## Standard deviation     4.062e-05 3.984e-05 3.966e-05 3.819e-05 3.774e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                              PC92      PC93      PC94      PC95      PC96
## Standard deviation     3.721e-05 3.673e-05 3.659e-05 3.525e-05 3.501e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                              PC97     PC98      PC99     PC100     PC101
## Standard deviation     3.437e-05 3.35e-05 3.291e-05 3.162e-05 3.148e-05
## Proportion of Variance 0.000e+00 0.00e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.00e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC102     PC103     PC104     PC105     PC106
## Standard deviation     3.109e-05 3.009e-05 2.959e-05 2.909e-05 2.877e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC107     PC108     PC109     PC110     PC111
## Standard deviation     2.754e-05 2.708e-05 2.699e-05 2.607e-05 2.566e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC112     PC113     PC114     PC115     PC116
## Standard deviation     2.482e-05 2.465e-05 2.395e-05 2.341e-05 2.329e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC117     PC118     PC119     PC120     PC121
## Standard deviation     2.319e-05 2.265e-05 2.209e-05 2.116e-05 2.036e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                             PC122     PC123     PC124    PC125     PC126
## Standard deviation     1.975e-05 1.968e-05 1.882e-05 1.74e-05 1.631e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.00e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.00e+00 1.000e+00
```




```
## [1] 0.1099449
## [1] 0.1099449
## [1] 15876
```



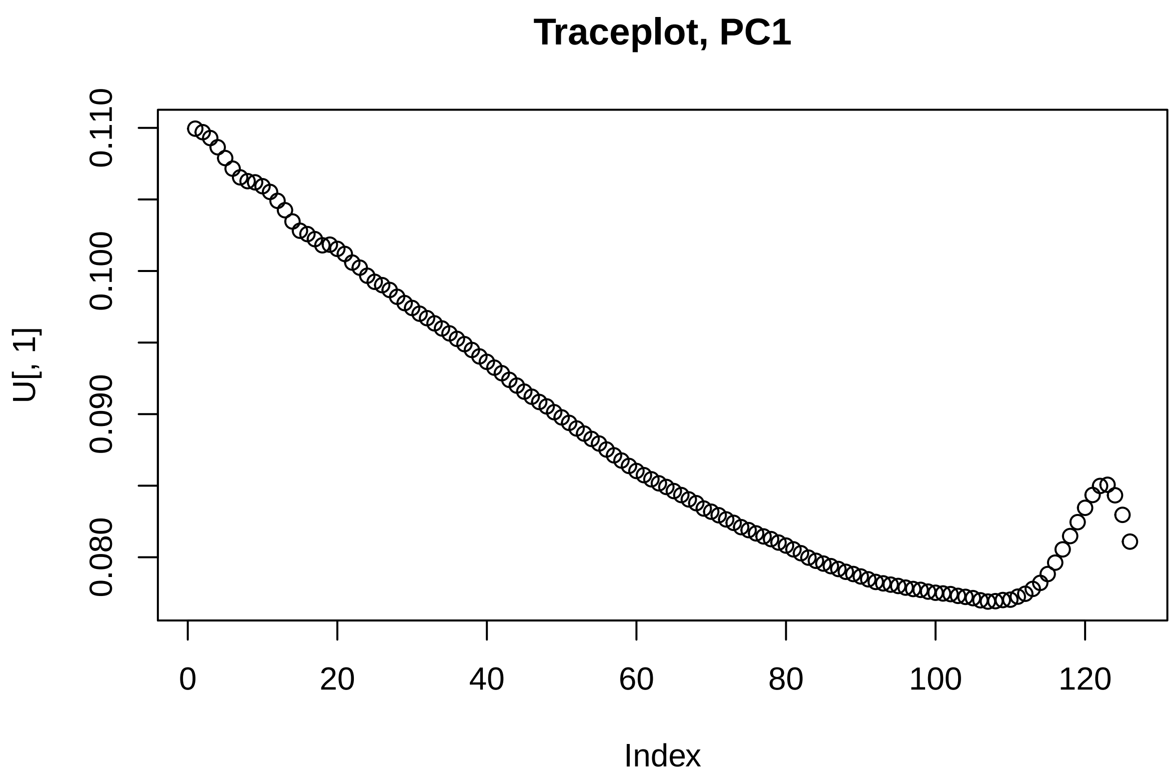

```
##   [1] 1.489912e-02 1.000141e-03 2.942193e-05 1.606981e-05 1.091329e-05
##   [6] 3.940982e-06 1.422208e-06 1.190221e-06 4.795324e-07 4.189848e-07
##  [11] 2.336100e-07 2.072663e-07 1.523494e-07 1.399692e-07 1.111147e-07
##  [16] 6.910784e-08 6.833408e-08 4.883681e-08 3.963071e-08 3.758265e-08
##  [21] 3.530222e-08 3.334363e-08 2.810230e-08 2.411554e-08 2.101708e-08
##  [26] 1.920449e-08 1.889152e-08 1.710557e-08 1.668139e-08 1.488880e-08
##  [31] 1.445696e-08 1.341082e-08 1.215810e-08 1.201298e-08 1.141010e-08
##  [36] 1.061049e-08 1.019264e-08 8.969431e-09 8.654385e-09 8.527660e-09
##  [41] 7.890007e-09 7.279826e-09 7.008830e-09 6.453520e-09 6.401090e-09
##  [46] 6.163010e-09 5.698006e-09 5.627278e-09 5.487044e-09 5.237122e-09
##  [51] 5.223093e-09 4.936148e-09 4.686094e-09 4.586564e-09 4.513493e-09
##  [56] 4.351218e-09 4.099592e-09 3.861069e-09 3.782457e-09 3.750444e-09
##  [61] 3.567883e-09 3.458162e-09 3.356734e-09 3.276631e-09 3.130533e-09
##  [66] 3.088143e-09 2.965596e-09 2.881796e-09 2.828133e-09 2.782184e-09
##  [71] 2.673118e-09 2.542032e-09 2.449146e-09 2.403927e-09 2.331542e-09
##  [76] 2.264111e-09 2.206617e-09 2.169984e-09 2.042035e-09 1.959453e-09
##  [81] 1.943790e-09 1.890905e-09 1.863678e-09 1.803452e-09 1.718646e-09
##  [86] 1.689874e-09 1.649876e-09 1.587308e-09 1.572650e-09 1.458505e-09
##  [91] 1.424494e-09 1.384584e-09 1.349208e-09 1.339001e-09 1.242673e-09
##  [96] 1.225995e-09 1.181056e-09 1.122436e-09 1.083157e-09 9.996590e-10
## [101] 9.906932e-10 9.668856e-10 9.053256e-10 8.757602e-10 8.459812e-10
## [106] 8.279380e-10 7.585268e-10 7.330583e-10 7.284513e-10 6.797996e-10
## [111] 6.582387e-10 6.158420e-10 6.074936e-10 5.738200e-10 5.480468e-10
## [116] 5.422023e-10 5.377965e-10 5.131362e-10 4.879618e-10 4.476130e-10
## [121] 4.144698e-10 3.901895e-10 3.871228e-10 3.542004e-10 3.027533e-10
## [126] 2.661716e-10
```

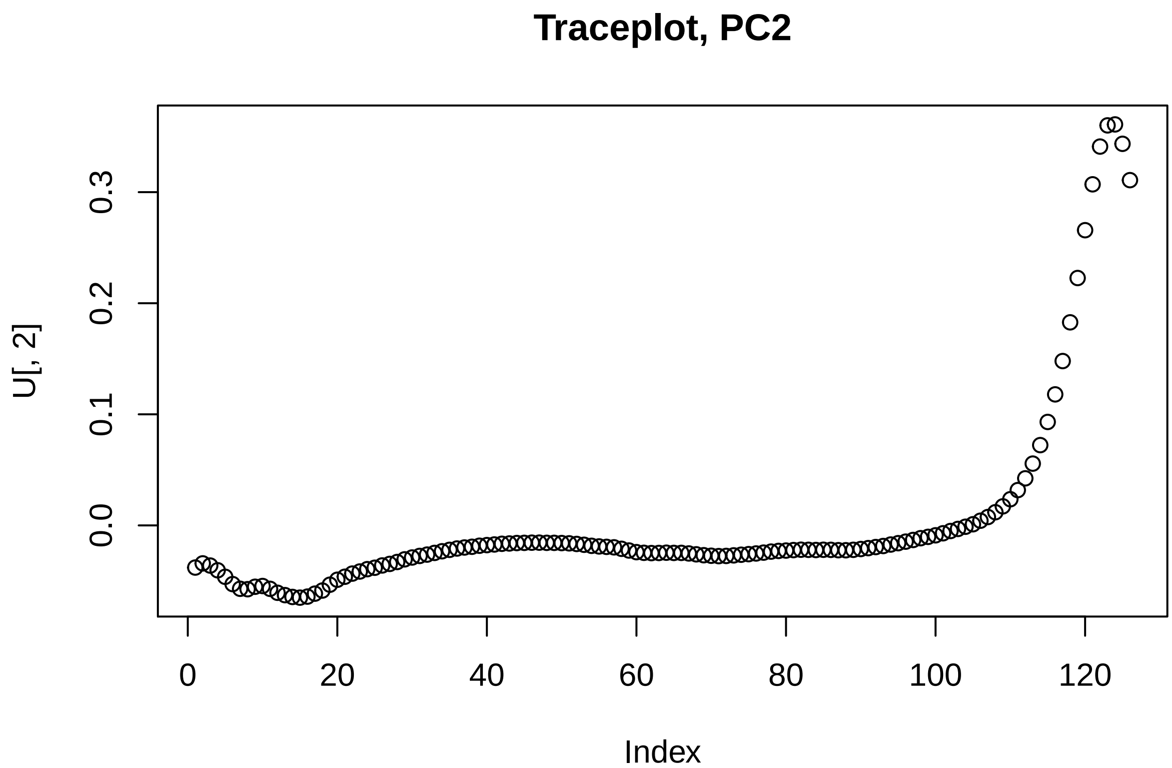
```
## [1] 99.59
```

Here two components are selected after seeing the scree graph as the first component covers almost 99 percent of the variances and the second component captures a signicant amount of variances as compared to the rest of the components. From the plot it is observed that there are a few outliers and hence there are unusual diesel fuels.

A4.2-Trace Plots

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```

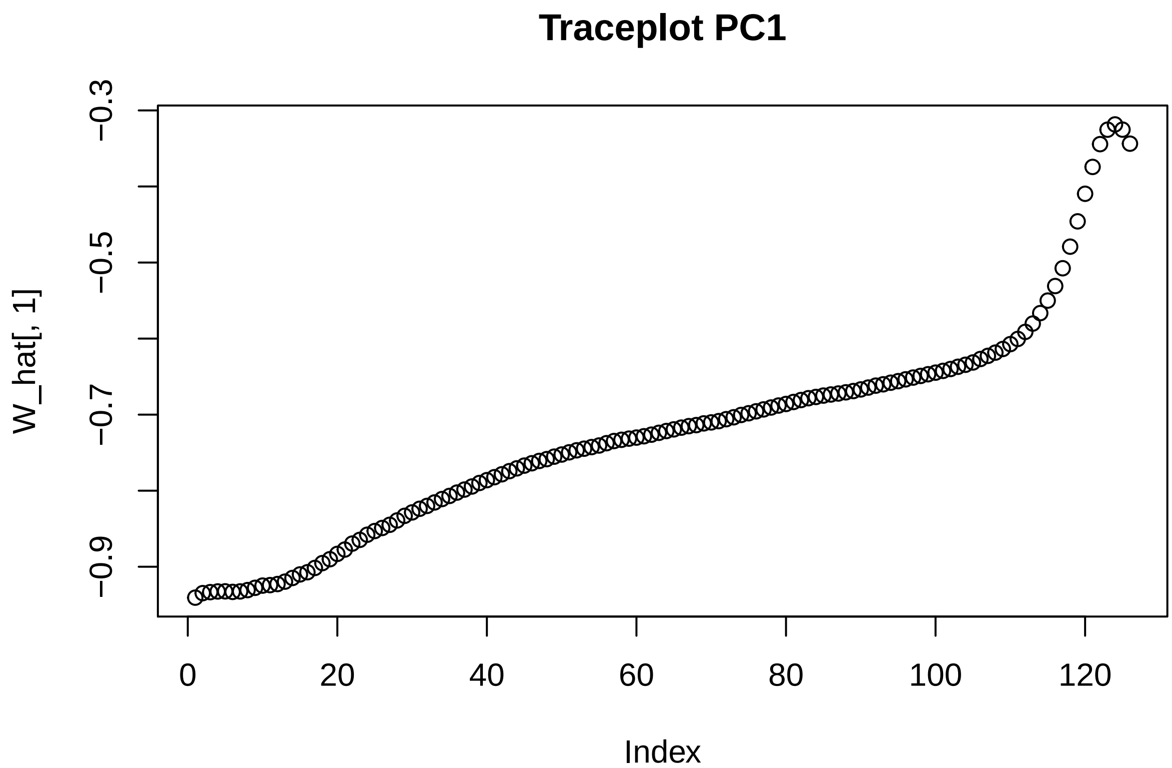


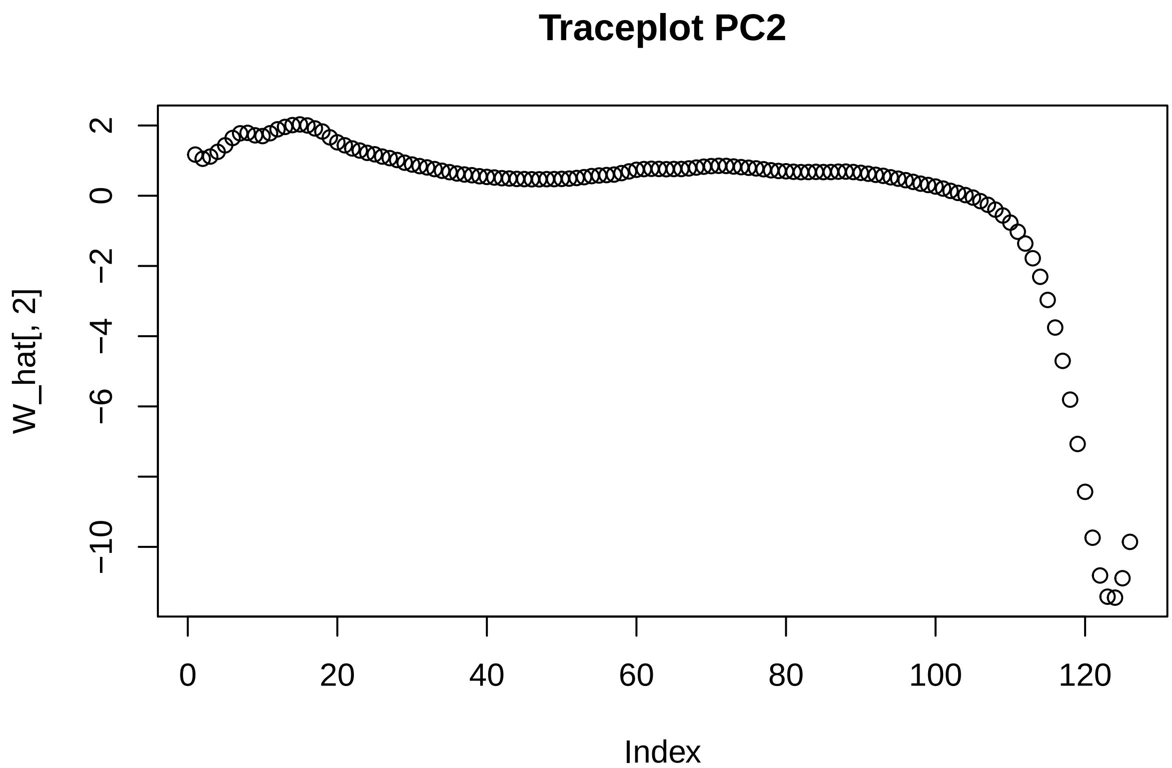


The variance reduces as the number of features increases.The variance for the PC1 after 100 features has reduced drastically whereas when u look at component 2 it starts off with a low variance and towards the end ahs high variance.Hence the first components which capture most of the variances can be explained by the original features.

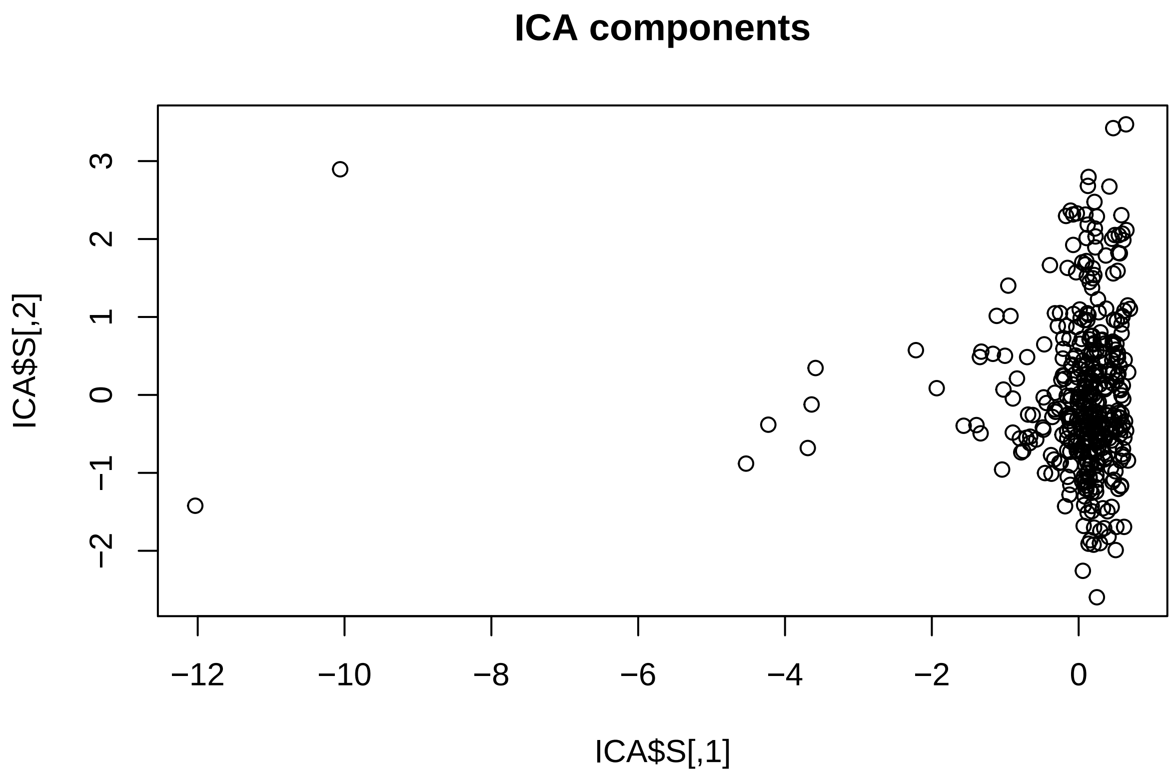
A4.3-Independant Component Analysis

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```


Traceplot PC1
W_hat[, 1]
−0.3
−0.5
−0.7
−0.9
0
20
40
60
80
100
120
Index



Traceplot PC2
W_hat[, 2]
2
0
−2
−4
−6
−10
0
20
40
60
80
100
120
Index



In both the cases the data points are clustered around the 0 point and hence can be concluded that both are highly correlated.

Appendix

```
RNGversion("3.5.1")
knitr::opts_chunk$set(echo = TRUE)
library(readxl)
library(SDMTools)
library(party)
library(tree)
library(ineq)
library(rpart)
library(rpart.plot)
library(caTools)
library(caret)
library(class)
library(maptree)
library(naivebayes)
library(ggplot2)
library(dplyr)
library(MASS)
library(readr)
library(fastICA)
creditscoring<-read_excel("creditscoring.xls")
n = dim(creditscoring)[1]
```

```r
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = creditscoring[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = creditscoring[id2,]
id3 = setdiff(id1,id2)
test = creditscoring[id3,]
creditscoring$good_bad=as.factor(creditscoring$good_bad)
RNGversion("3.5.1")
creditscoring<-read_excel("creditscoring.xls")
n = dim(creditscoring)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = creditscoring[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
valid = creditscoring[id2,]
id3 = setdiff(id1,id2)
test = creditscoring[id3,]
creditscoring$good_bad=as.factor(creditscoring$good_bad)
RNGversion("3.5.1")
#Classifier with deviance
tree_dev<-tree(as.factor(good_bad)~.,data=train,method = "recursive.partition",split = c("deviance"))
#Prediction
predict_tree_test<-predict(tree_dev, test,type = "class")
predict_tree_train<-predict(tree_dev,train,type = "class")
#Confusion matrix for evaluating  the model(test)
confusionMatrix<-table(predict_tree_test,test$good_bad)
confusionMatrix
#Confusion matrix for evaluating the model(train)
confusionMatrix1<-table(predict_tree_train,train$good_bad)
confusionMatrix1
#Misclassification rate for deviance (train)
misclassification_rate_deviance_train<-1-(sum(diag(confusionMatrix1))/sum(confusionMatrix1))
cat("\n The Deviance index train data missclassification rate is ",misclassification_rate_deviance_trai
#Misclassification rate for deviance (test)
misclassification_rate_deviance_test<-1-(sum(diag(confusionMatrix))/sum(confusionMatrix))
cat("\n The Deviance index test data missclassification rate is ",misclassification_rate_deviance_test)

##Gini index
#Classifier with Gini
tree_dev2<-tree(as.factor(good_bad)~.,data=train,method = "recursive.partition",split = c("gini"))
#Prediction
predict_tree_test1<-predict(tree_dev2, test,type = "class")
predict_tree_train1<-predict(tree_dev2,train,type = "class")
#Confusion matrix for evaluating  the model(test)
confusionMatrix<-table(predict_tree_test1,test$good_bad)
confusionMatrix
#Confusion matrix for evaluating the model(train)
confusionMatrix1<-table(predict_tree_train1,train$good_bad)
```

```
confusionMatrix1

#Misclassification rate for deviance (train)
misclassification_rate_gini_train<-1-(sum(diag(confusionMatrix1))/sum(confusionMatrix1))
cat("\n The Gini index train data missclassification rate is ",misclassification_rate_gini_train)
#Misclassification rate for deviance (test)
misclassification_rate_gini_test<-1-(sum(diag(confusionMatrix))/sum(confusionMatrix))
cat("\n The Gini index test data missclassification rate is ",misclassification_rate_gini_test)
RNGversion("3.5.1")

fit=tree(as.factor(good_bad)~., data=train)
terminal_node = summary(fit)$size
set.seed(12345)
trainScore=rep(0,terminal_node)
testScore=rep(0,terminal_node)
for(i in 2:terminal_node) {
  prunedTree=prune.tree(fit,best=i)
  pred=predict(prunedTree, newdata=valid,
               type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:terminal_node, trainScore[2:terminal_node], type="b", col="red",
     ylim=c(0,800))
points(2:terminal_node, testScore[2:terminal_node], type="b", col="blue")

min_dev <- which(testScore[2:terminal_node] == min(testScore[2:terminal_node]) )
cat("\n The minimum deviance is found out to be when the depth of the tree is ",min_dev)

#The optimal depth of the tree is found out and fit to find the best tree

finalTree=prune.tree(fit, best=min_dev)
Yfit=predict(finalTree, newdata=valid,
             type="class")
confusionmatrix5<-table(Yfit,valid$good_bad)
confusionmatrix5
misclassification<-1-(sum(diag(confusionmatrix5))/sum(confusionmatrix5))
misclassification
plot(finalTree)
text(finalTree)


RNGversion("3.5.1")
#predict with naive bayes (train data)
fit_naive_bayes<-naive_bayes(good_bad~., data=train,type = "prob")
predict_naive_bayes_train<-predict(fit_naive_bayes,train,type = "class")
confusionmatrix4<-table(predict_naive_bayes_train,train$good_bad)
confusionmatrix4
misclassification_naive_bayes_train<-1-(sum(diag(confusionmatrix4))/sum(confusionmatrix4))
cat("\n The misclassification rate when the Naive bayes models is used on train data is ",misclassificat
#Naive bayes clssifier (test data)
predict_naive_bayes_test<-predict(fit_naive_bayes,test,type = "class")
confusionmatrix3<-table(predict_naive_bayes_test,test$good_bad)
```

```r
confusionmatrix3
misclassification_naive_bayes<-1-(sum(diag(confusionmatrix3))/sum(confusionmatrix3))
cat("\n The misclassification rate when the Naive bayes models is used on test data is ",misclassificati

RNGversion("3.5.1")
#For optimal tree
pie = seq(0.05,0.95,by = 0.05)
fit_optimal_tree<-tree(as.factor(good_bad)~., data=train,split = "deviance")
finalTree=prune.tree(fit_optimal_tree, best=min_dev)
predict_optimal_tree<-predict(finalTree,test)

good_prob <- predict_optimal_tree[,which(colnames(predict_optimal_tree)=="good")]

test1 <- test
#temp_pred <- ifelse(predict_naive_bayes == "good",1,0)
new_y <- ifelse(test1$good_bad == "good",1,0)
newmatrix<-matrix(1,nrow = length(pie))
tpr_fpr<-matrix(nrow =length(pie),ncol = 2)

for (i in 1:length(pie)) {

  y_pred_test = ifelse(good_prob>pie[i],1,0)

  con_mat <- confusion.matrix(y_pred_test,new_y)

  tpr <- con_mat[2,2]/sum(con_mat[2,1]+con_mat[2,2])
  fpr <- con_mat[1,2]/sum(con_mat[1,1]+con_mat[1,2])
  tpr_fpr[i,] <- c(fpr,tpr)
  }
#For Naive bayes
predict_naive_bayes<-predict(fit_naive_bayes,test,type = "prob")

good_prob1 <- predict_naive_bayes[,which(colnames(predict_naive_bayes)=="good")]

test1 <- test
tpr_fpr_naive<-matrix(nrow =length(pie),ncol = 2)
#temp_pred <- ifelse(predict_naive_bayes == "good",1,0)
new_y <- ifelse(test1$good_bad == "good",1,0)
newmatrix<-matrix(1,nrow = length(pie))
tpr_fpr_naive<-matrix(nrow =length(pie),ncol = 2)

for (i in 1:length(pie)) {

  y_pred_test1 = ifelse(good_prob1>pie[i],1,0)

  con_mat <- confusion.matrix(y_pred_test1,new_y)

  tpr1 <- con_mat[2,2]/sum(con_mat[2,])
  fpr1 <- con_mat[1,2]/sum(con_mat[1,])
  tpr_fpr_naive[i,] <- c(fpr1,tpr1)
}

plot(tpr_fpr[,1],tpr_fpr[,2],type = "l",col = "red",pch = 19,xlab = "FPR",ylab = "TPR",main = "FPR VS TP
```

```r
points(tpr_fpr_naive[,1],tpr_fpr_naive[,2],type = "l", col = "green",pch= 19,xlab = "FPR",ylab = "TPR",m

RNGversion("3.5.1")
# the naive bayes model is fitted for the test and train data above and hence we can use these to deriv
predict_naive_bayes_test<-predict(fit_naive_bayes,test,type = "prob")
predict_naive_bayes_train<-predict(fit_naive_bayes,train,type = "prob")
good_prob2 <- predict_naive_bayes_train[,which(colnames(predict_naive_bayes_train)=="good")]
good_prob3 <- predict_naive_bayes_test[,which(colnames(predict_naive_bayes_test)=="good")]

naive_bayes_train_loss<-ifelse(good_prob2>0.1,'good','bad')
naive_bayes_test_loss<-ifelse(good_prob3>0.1,'good','bad')
conf_mat_train<-table(naive_bayes_train_loss,train$good_bad)
conf_mat_train
conf_mat_test<-table(naive_bayes_test_loss,test$good_bad)
conf_mat_test

# misclassification rate
miscal_naive_train<-1-(sum(diag(conf_mat_train))/sum(conf_mat_train))
miscal_naive_test<-1- (sum(diag(conf_mat_test))/sum(conf_mat_test))
cat("\n The misclassification rate when the train data is used is",miscal_naive_train)
cat("\n The misclasification rate when the test data is used is ",miscal_naive_test)
RNGversion("3.5.1")
State.data <- read_csv2("State.csv")
#Ordered data
State.data <-State.data[order(State.data$MET),]


plot(EX~MET, State.data,pch = 15,col = "blue",main ="EX VS MET")

RNGversion("3.5.1")
##Trainig the model
trainedmodel<-tree(formula=EX~MET,data=State.data,control = tree.control(nobs = nrow(State.data),minsize
#Fitted tree
plot(trainedmodel)
text(trainedmodel)

tree_cv1<-cv.tree(trainedmodel)
plot(tree_cv1,main = "Deviance Vs tree size")
##Plotting the  cv tree


plot(tree_cv1$size,tree_cv1$dev,type = "b",main = "Size Vs Deviance",col = "red")

Optimal_Size<- tree_cv1$size[which(tree_cv1$dev==min(tree_cv1$dev))]
cat("\n Optimal tree:",Optimal_Size)
##b original vs fitted data

OptimalTree=prune.tree(trainedmodel,best = Optimal_Size)
OptimalTreeFit=predict(OptimalTree, State.data)

resultant  <- data.frame(Indicated=State.data$MET,original=State.data$EX,predicted=OptimalTreeFit)
ggplot(resultant , aes(x = Indicated,color = "variable")) +
  geom_point(aes(y = original, col = "original")) +
```

```r
  geom_point(aes(y = predicted, col = "predicted"))+
  ggtitle("Predicted Vs original using optimal tree of size 3")
#c Histogram of the residuals
residuals <- State.data$EX - OptimalTreeFit

hist(residuals)
RNGversion("3.5.1")
#95% Confidence bands  for the regression tree model using non-parametric bootstrap
library(boot)
set.seed(12345)
ordered_data=State.data[order(State.data$MET),]
non_para_f=function(data,ind)
{
  sample_ext<-State.data[ind,]
  trainedmodel<-tree(formula=EX~MET,data=sample_ext,control = tree.control(nobs =nrow(sample_ext),minsi
  OptimalTree=prune.tree(trainedmodel,best = Optimal_Size)
  OptimalTreeFit=predict(OptimalTree, newdata=data)
  return(OptimalTreeFit)
}
non_para_boot = boot(State.data,non_para_f,R = 1000)
e = envelope(non_para_boot,level = 0.95)
plot(ordered_data$MET,State.data$EX,main = "95 % confidence bands using non-parametric bootstrap",bg =
points(ordered_data$MET,e$point[1,],col = 'violet',type = "l")
points(ordered_data$MET,e$point[2,],col = 'red',type = "l")
points(ordered_data$MET,OptimalTreeFit,type = "l",pch = 15,col = "green")
legend("topright",c("Upper Band","Predicted","Lower Band"),fill = c("violet","green","red"))




RNGversion("3.5.1")
ordered_data=State.data[order(State.data$MET),]
mle = prune.tree(tree(EX~MET,ordered_data,minsize = 8),best = Optimal_Size)

rng=function(data, mle) {
data1=data.frame(EX=ordered_data$EX, MET=ordered_data$MET)
n=length(ordered_data$EX)
#generate new EX
pred_val <- predict(mle, newdata=data)
residual <- data$EX - pred_val
data1$EX=rnorm(n,pred_val,sd(residuals))
return(data1)
}
para_bootstrap=function(data1){
tree_classification <- tree(EX~MET,data1,control = tree.control(nobs = nrow(data1),minsize = 8))
  res=prune.tree(tree_classification,best = Optimal_Size)#fit linear model
#predict values for all Area values from the original data
EXPred=predict(res,newdata=data1)#ordered_data)
EX_Pred_norm <- rnorm(length(ordered_data$EX),EXPred,sd(residuals(OptimalTree)))
return(EX_Pred_norm)
}

para_boot =boot(ordered_data,statistic = para_bootstrap,R = 1000,mle = mle,ran.gen = rng,sim = "paramet
```

```
e = envelope(para_boot,level = 0.95)
treefit<-tree(EX~MET,ordered_data,control = tree.control(nobs = nrow(ordered_data),minsize = 8))
parapred<-predict(treefit)
plot(ordered_data$MET,ordered_data$EX,pch = 21,bg = "orange",main = "95% Confidence bands using paramet
points(ordered_data$MET,parapred,type = "l", col = "green")
#confidence bands
points(ordered_data$MET, e$point[2,], type="l", col="red",pch = 19)
points(ordered_data$MET, e$point[1,], type="l", col="violet", pch = 19)
legend("topright",c("Upper Band","Predicted","Lower Band"),fill = c("violet","green","red"))

# Checking the best fit
hist(residuals)

RNGversion("3.5.1")
Assignment4 <- read.csv2("NIRspectra.csv",header = T, sep = ";",quote = "\"",fill = T)
# Assignment4
Assignment4$Viscosity=c()

res = prcomp(Assignment4)
summary(res)
screeplot(res)

print(min(res$rotation[1]))
print(max(res$rotation[1]))
print(length(res$rotation))
# head(U)
plot(res$x[,1],res$x[,2] , ylim=c(-0.2,0.2),xlim = c(-1,1),xlab = "PC2",ylab = "PC1")
lambda = res$sdev^2
#eigenvalues
lambda
#proportion of variation
sf<-as.numeric(sprintf("%2.3f",lambda/sum(lambda)*100))
total_per<-sf[1]+sf[2]
total_per


RNGversion("3.5.1")
U=res$rotation
plot(U[,1], main="Traceplot, PC1")
plot(U[,2],main="Traceplot, PC2")
RNGversion("3.5.1")
set.seed(12345)
ICA<-fastICA(Assignment4,2)
W_hat<-ICA$K %*% ICA$W
plot(W_hat[,1],main = "Traceplot PC1")
plot(W_hat[,2],main = "Traceplot PC2")
plot(ICA$S, main = "ICA components", col  = )
```