Block 1, Lab 2 Report

Mohammed Bakheet

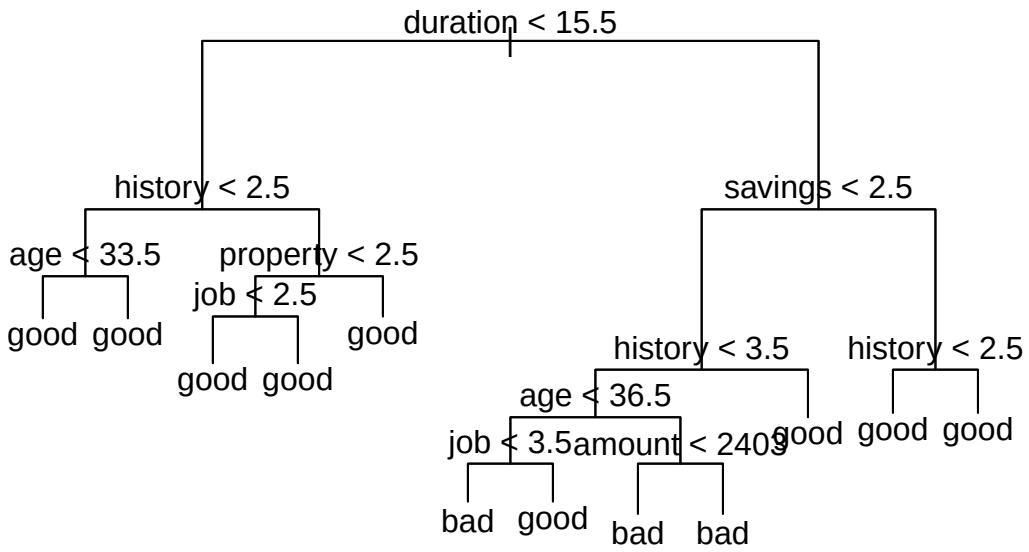
12/07/2019

Assignment 2. Analysis of credit scoring

```
summary(treeModelDeviance)
```

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = "deviance")
## Variables actually used in tree construction:
## [1] "duration" "history"  "age"      "property" "job"      "savings"  "amount"
## Number of terminal nodes:  12
## Residual mean deviance:  0.9879 = 476.2 / 482
## Misclassification error rate: 0.247 = 122 / 494
```

```
plot(treeModelDeviance)
text(treeModelDeviance,pretty = 0)
```

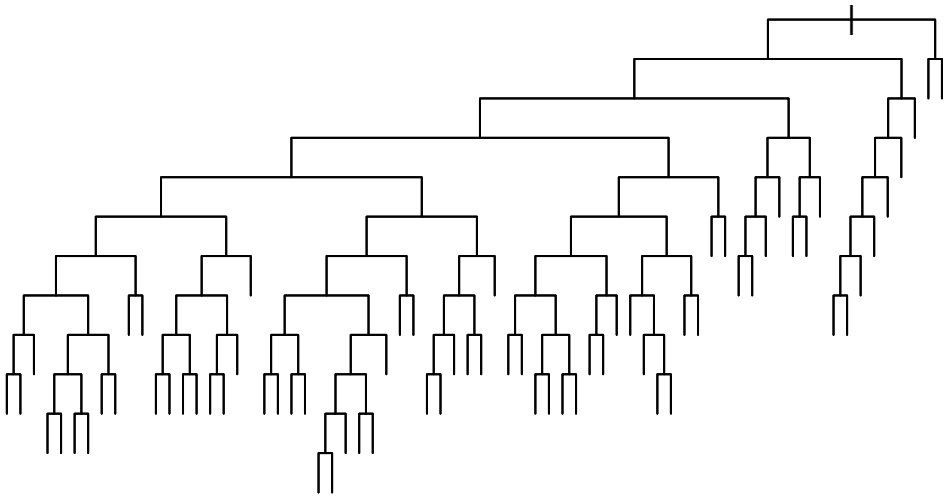


#Missclassification error rate for gini index model is 0.2368, whereas the missclassification rate for the model when using deviance is 0.2105

```
summary(treeModelGini)
```

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = "gini")
## Variables actually used in tree construction:
##  [1] "foreign"  "coapp"    "depends"  "existcr"  "telephon" "savings"
##  [7] "history"  "property" "employed" "resident" "marital"  "purpose"
## [13] "duration" "housing"  "installp" "amount"   "job"      "age"
## Number of terminal nodes:  70
## Residual mean deviance:  1.059 = 449.1 / 424
## Misclassification error rate: 0.247 = 122 / 494
```

```
plot(treeModelGini, type = "uniform")
```



```
#text(treeModelGini,pretty = 0)
```

```
#Missclassification rate for training data for deviance
print(1 - mean(treePredTraining == train$good_bad))
```

```
## [1] 0.252
```

```r
table(treePredTraining,train$good_bad)
```

```
##
## treePredTraining bad good
##             bad   71   49
##             good  77  303
```

```r
#Missclassification rate for validation data for deviance
print(1 - mean(treePredValid == validationData$good_bad))
```

```
## [1] 0.32
```

```r
table(treePredTest,validationData$good_bad)
```

```
##
## treePredTest bad good
##         bad   25   41
##         good  62  122
```

```r
#Missclassification rate for the testing data for deviance
print(1 - mean(treePredTest == testingData$good_bad))
```

```
## [1] 0.348
```

```r
table(treePredTest,testingData$good_bad)
```

```
##
## treePredTest bad good
##         bad   22   44
##         good  43  141
```

```r
#Missclassification rate for training data for Gini measurment
print(1 - mean(treePredTrainingG == train$good_bad))
```

```
## [1] 0.252
```

```r
table(treePredTrainingG,train$good_bad)
```

```
##
## treePredTrainingG bad good
##              bad   58   36
##              good  90  316
```

```r
#Missclassification rate for validation data for Gini measurment
print(1 - mean(treePredValidG == validationData$good_bad))
```

```
## [1] 0.344
```

```
table(treePredValidG,validationData$good_bad)
```

```
##
## treePredValidG bad good
##           bad   28   27
##           good  59  136
```

```
#Missclassification rate for the testing data for Ginin measurment
print(1 - mean(treePredTestG == testingData$good_bad))
```

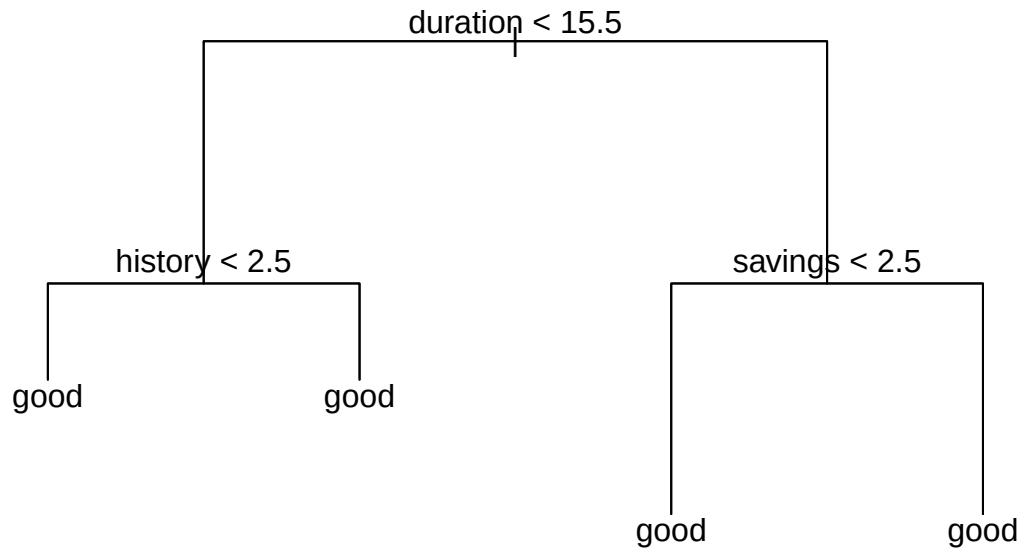
```
## [1] 0.348
```

```
table(treePredTestG,testingData$good_bad)
```

```
##
## treePredTestG bad good
##           bad   19   41
##           good  46  144
```


```
##
## Classification tree:
## snip.tree(tree = treeModelBest, nodes = c(4L, 7L, 5L, 6L))
## Variables actually used in tree construction:
```

```
## [1] "duration" "history"  "savings"
## Number of terminal nodes:  4
## Residual mean deviance:  1.089 = 533.7 / 490
## Misclassification error rate: 0.2935 = 145 / 494
```




```
##       Yfit
##        bad good
##   bad    0   87
##   good   0  163
```

```
## [1] 0.348
```

```
##       YfitTest
##        bad good
##   bad    0   65
##   good   0  185
```

```
## [1] 0.26
```

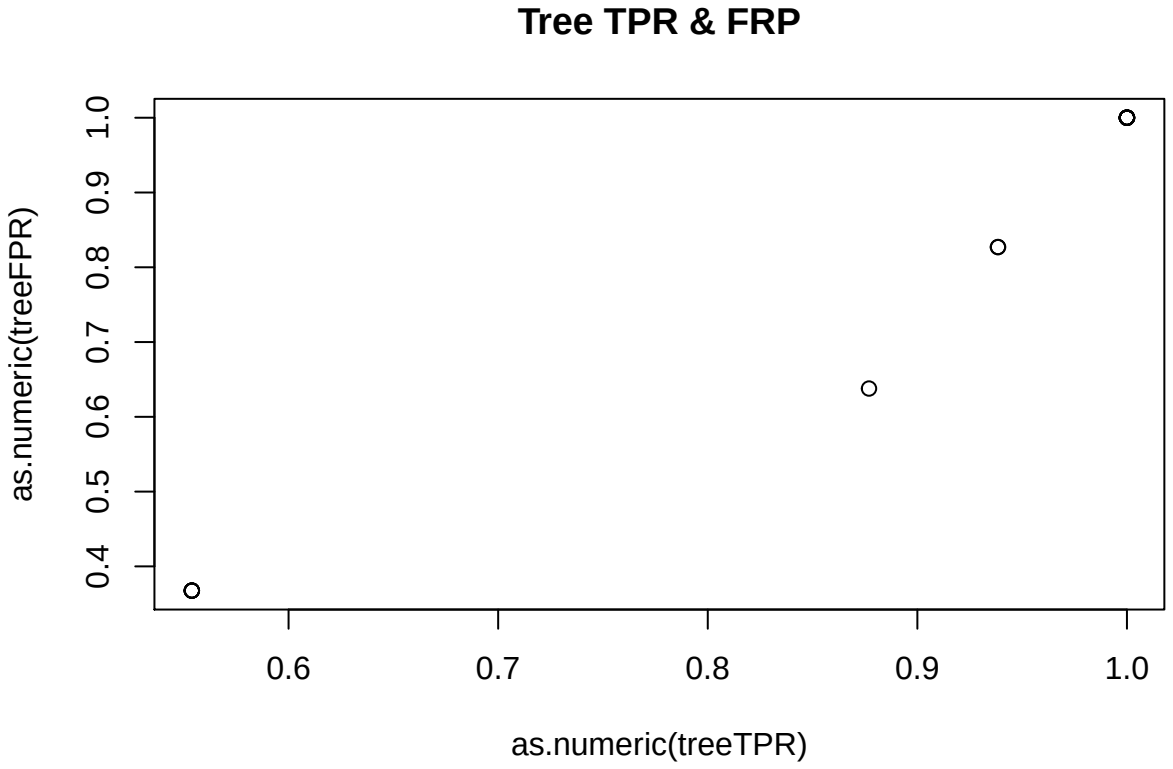
```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    19     -none- list
## levels     2     -none- character
## isnumeric 19     -none- logical
## call       4     -none- call
```

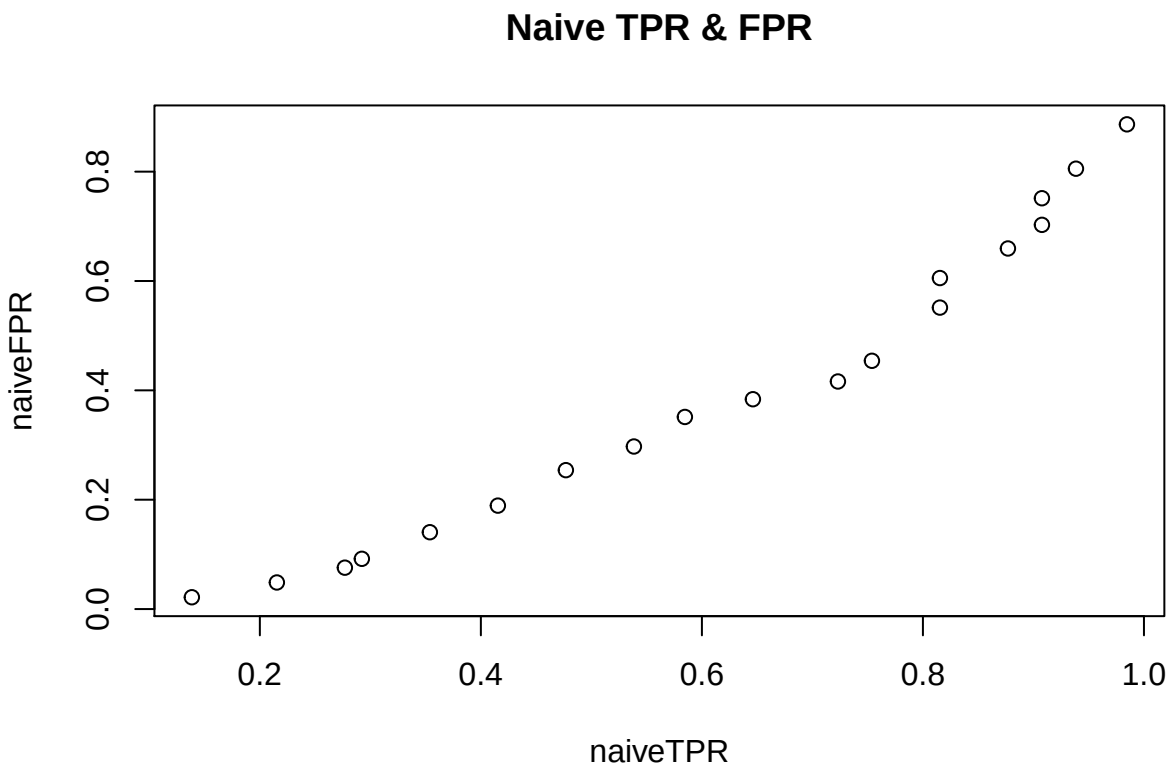
```
##
## YfitNaiveTrain bad good
##           bad  103  111
##           good  45  241
```

```
## [1] 0.312
```

```
##
## YfitNaiveTest bad good
##          bad   42   71
##          good  23  114
```

```
## [1] 0.376
```





#The tree model did very well, by looking at the values of true positive and false positive, The true positive rates are very high (sometimes all values are predicted correctly), and the same is correct for false positive rates. Whereas, in the naive bayes model the values of true positive and false positive rates increase as we increase the value of Pi.

#6)loss matrix

```
lossModel <- naiveBayes(good_bad~., train)

lossModelPred <- predict(lossModel, newdata = train, type = "raw")
condPrediction <- ifelse(lossModelPred[,2] > 0.1, "good", "bad")

#Confusion matrix for the training data
table(train$good_bad, condPrediction)
```

```
##       condPrediction
##        bad good
##   bad   24  124
##   good  20  332
```

```
#Missclassification rate for the training data
lossMisClassification <- 1-mean(train$good_bad == condPrediction)
lossMisClassification
```

```
## [1] 0.288
```

```r
lossModelTest <- naiveBayes(good_bad~., train)
lossModelTesting <- predict(lossModelTest, newdata = testingData, type = "raw")
condPredTesting <- ifelse(lossModelTesting[,2] > .1, "1", "0")

#Confusion matrix for the testing data
table(testingData$good_bad, condPredTesting)
```

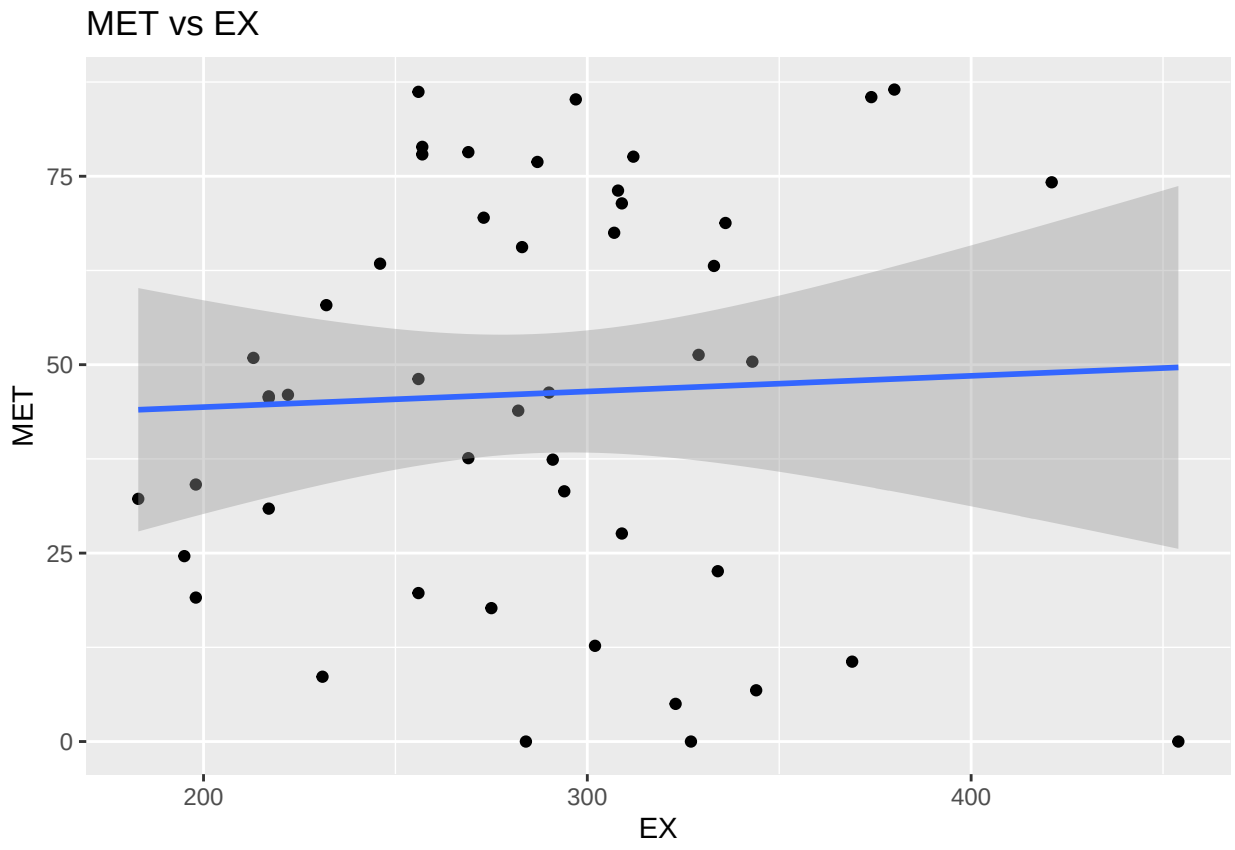
```
##       condPredTesting
##          0   1
##   bad   14  51
##   good   9 176
```

```r
#Missclassification rate for the testing data
lossMisClassificationTest <- 1-mean(testingData$good_bad == condPredTesting)
lossMisClassificationTest
```

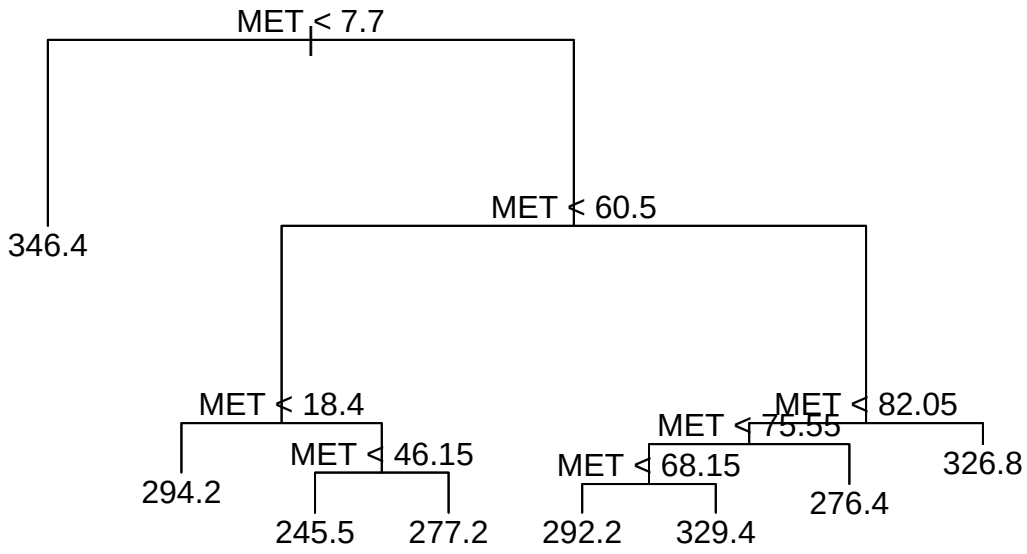
```
## [1] 1
```

Assignment 3. Uncertainty estimation

```r
ggplot(data2,aes(x=EX,y =MET )) +geom_point() +geom_smooth(method = "lm")+ggtitle("MET vs EX")
```



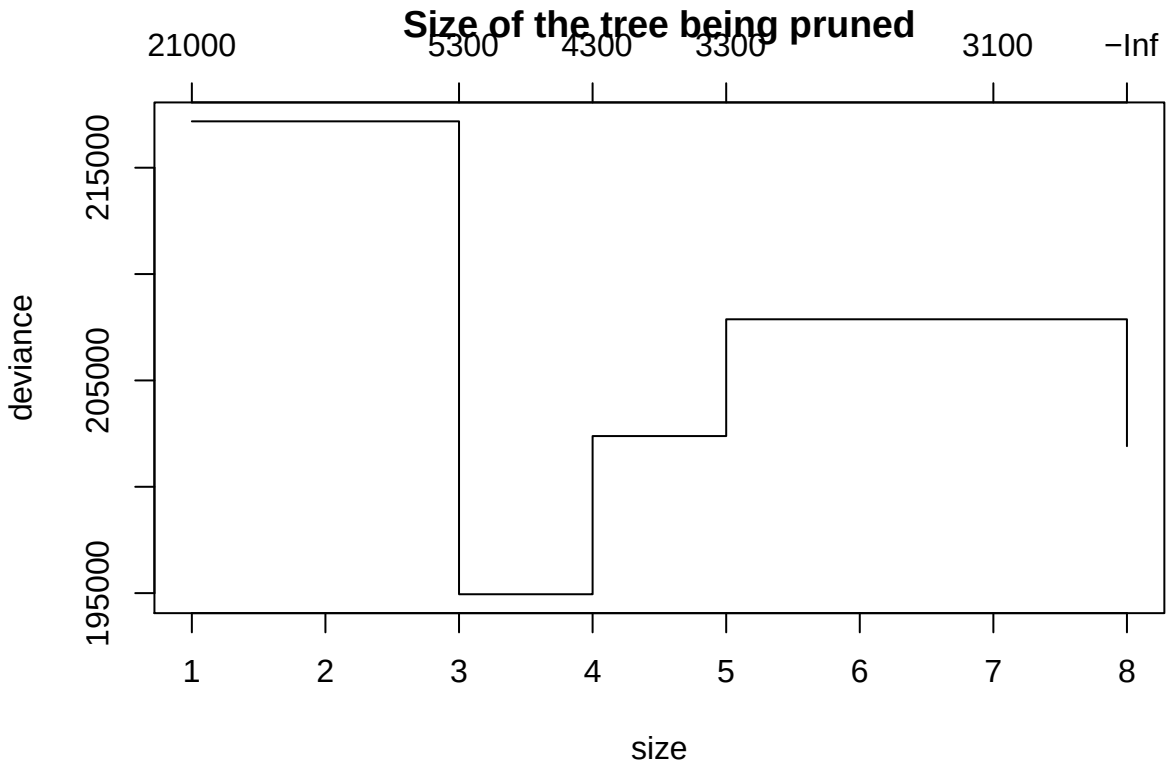
```r
#3.2) Regression tree model
      plot(regTreeModel)
      text(regTreeModel, pretty = 0)
```



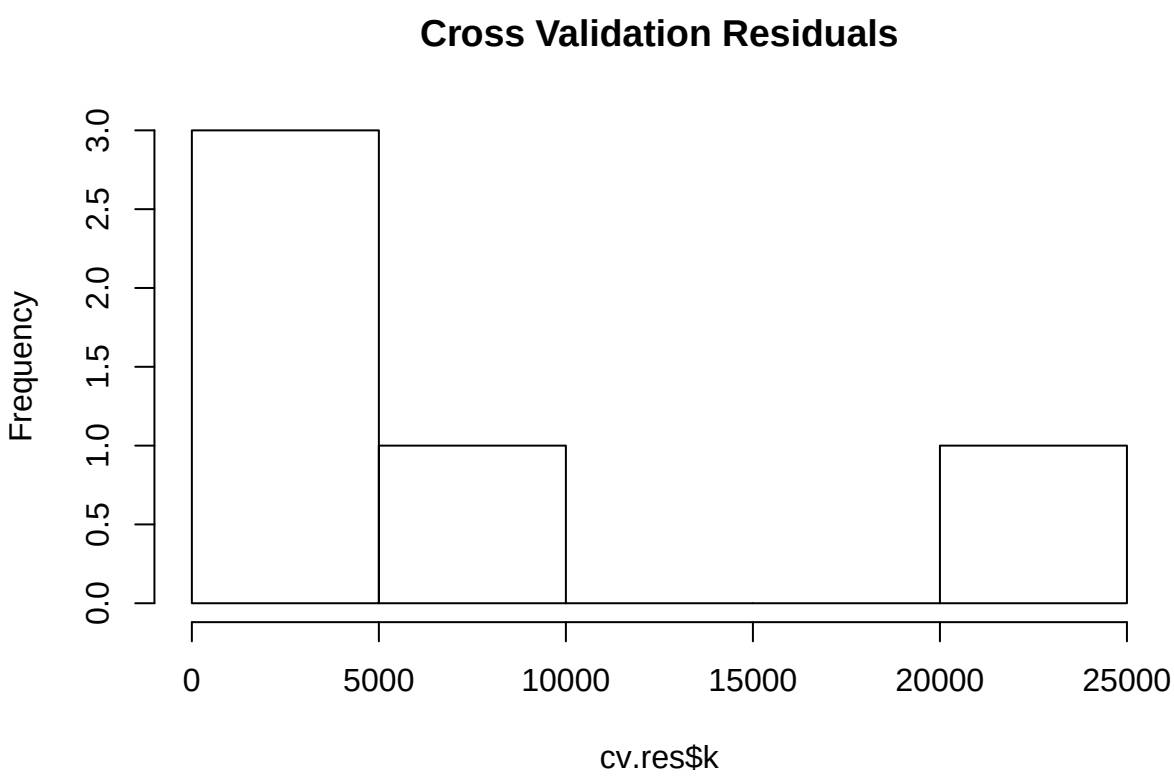

```r
      summary(regTreeModel)
```

```
##
## Regression tree:
## tree(formula = EX ~ MET, data = data2, control = setup)
## Number of terminal nodes:  8
## Residual mean deviance:  2555 = 102200 / 40
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -70.75  -28.79  -14.25    0.00   37.59  107.60
```

```r
      #applying cross validation for the tree
      cv.res=cv.tree(regTreeModel)
      plot(cv.res,main="Size of the tree being pruned")
```



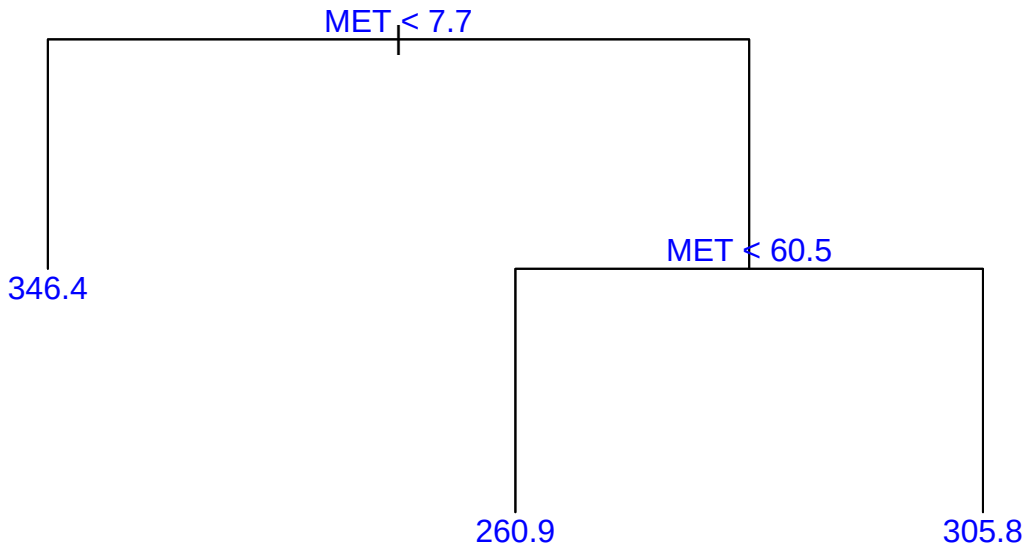

```
minDeviance <- which.min(cv.res$dev)
hist(cv.res$k, main = "Cross Validation Residuals")
```



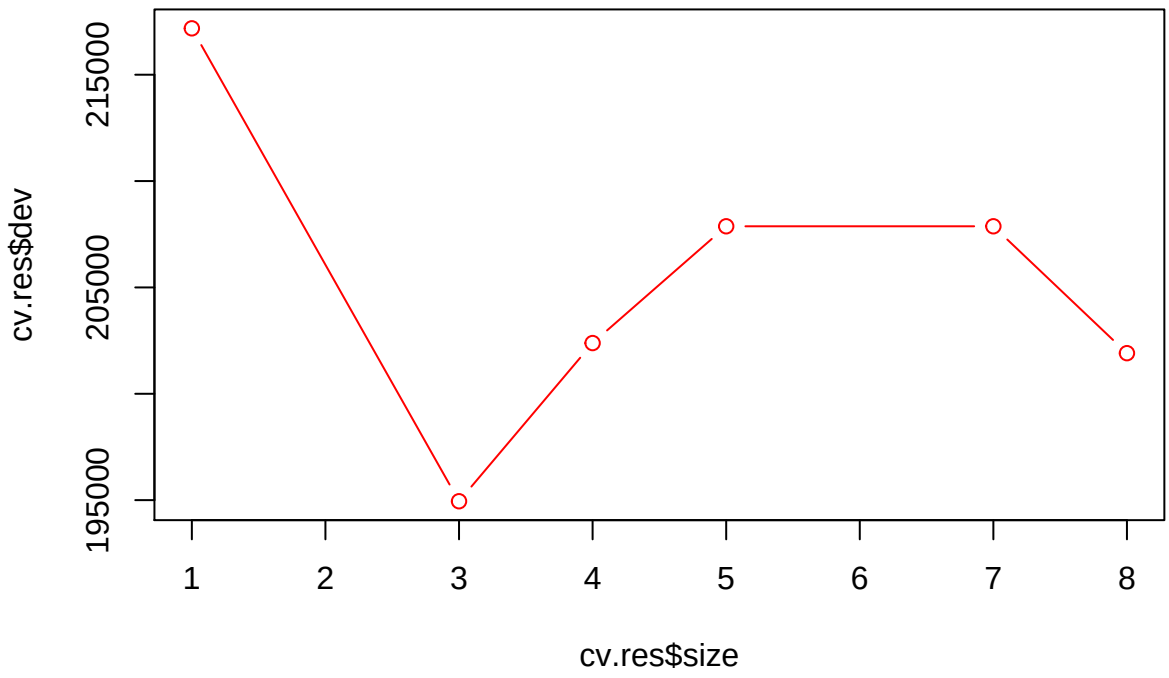

```
      #minumum number of leaves
      minLeaves <- cv.res$size[minDeviance]
      minLeaves
```

```
## [1] 3
```

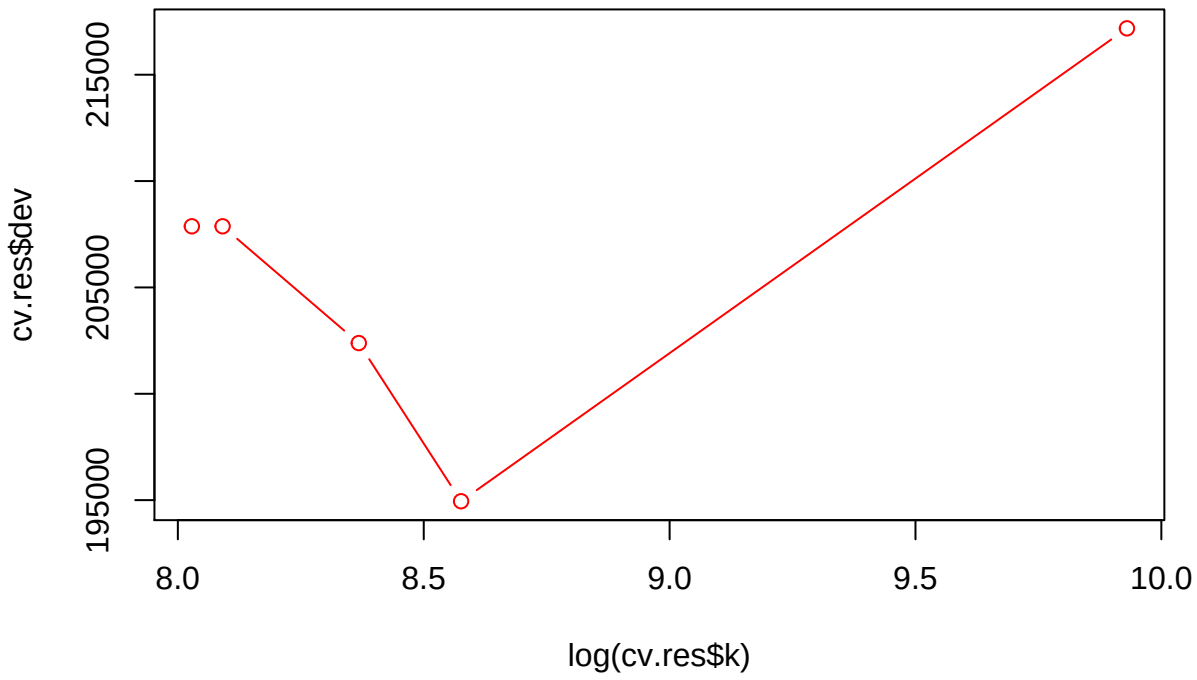
```
      #The optimal tree
      optimalTree=prune.tree(regTreeModel, best=3)
      plot(optimalTree, main="Optimal Tree")
      text(optimalTree,pretty = 0, col="blue")
```




```
#Residuals
plot(cv.res$size, cv.res$dev, type="b", col="red")
```

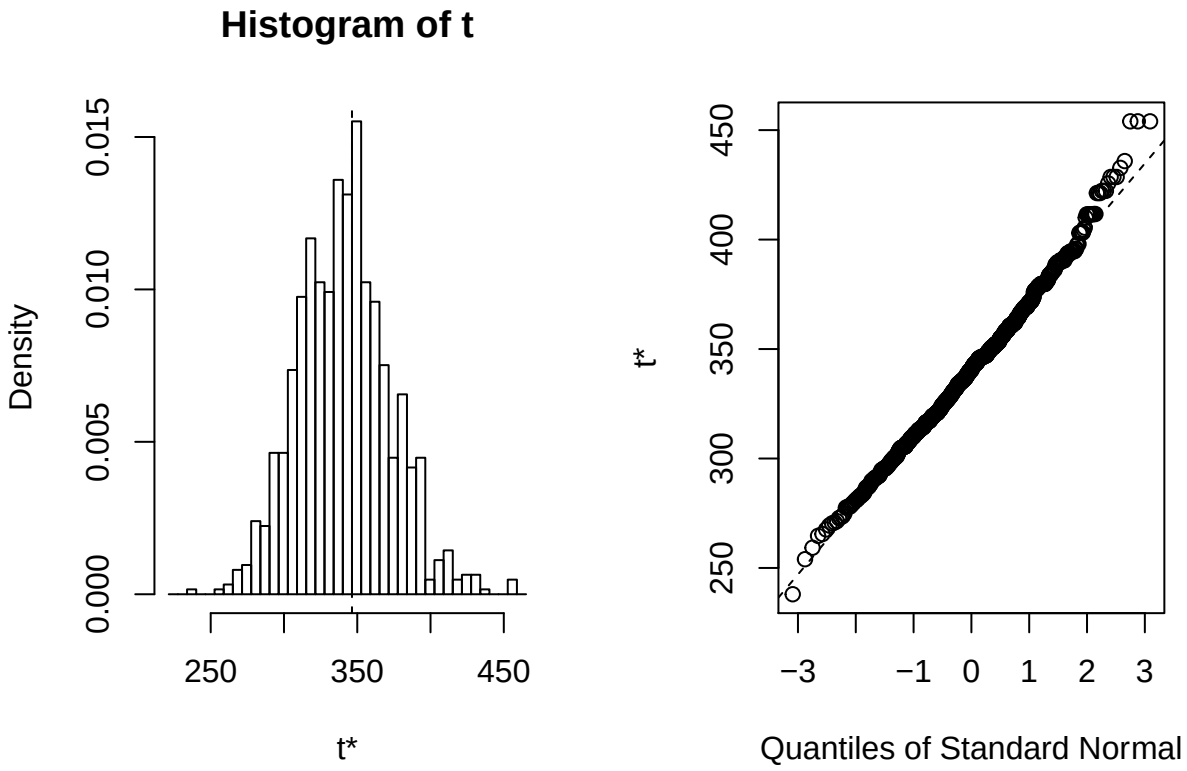



```
plot(log(cv.res$k), cv.res$dev,type="b", col="red")
```



#The deviance is least when the number of leaves is equal to three, as for the quality of fit, the deviance is higher when two or one leave, but it's well distributed closed to linearly when using 3,4,5,7, or 8 leaves.

```
#Make a bootstrap
bootResult <- boot(data2,statistic = f,R=1000)
plot(bootResult)
```



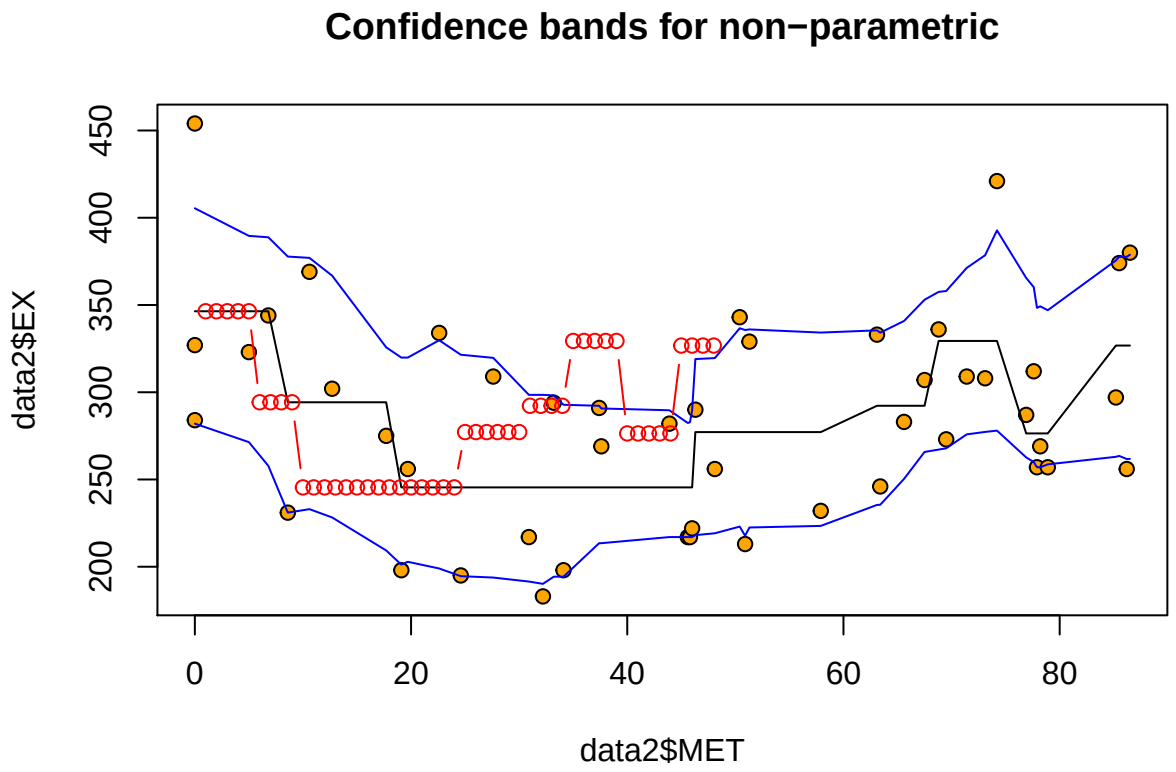

```
      e=envelope(bootResult) #compute confidence bands
      summary(e)
```

```
##         Length Class  Mode
## point   96     -none- numeric
## overall 96     -none- numeric
## k.pt     2     -none- numeric
## err.pt   2     -none- numeric
## k.ov     2     -none- numeric
## err.ov   2     -none- numeric
## err.nom  2     -none- numeric
```

```
      #Calculating the boot confidence interval
      bootCi <- boot.ci(boot.out = bootResult, type = "norm")
      bootCi
```

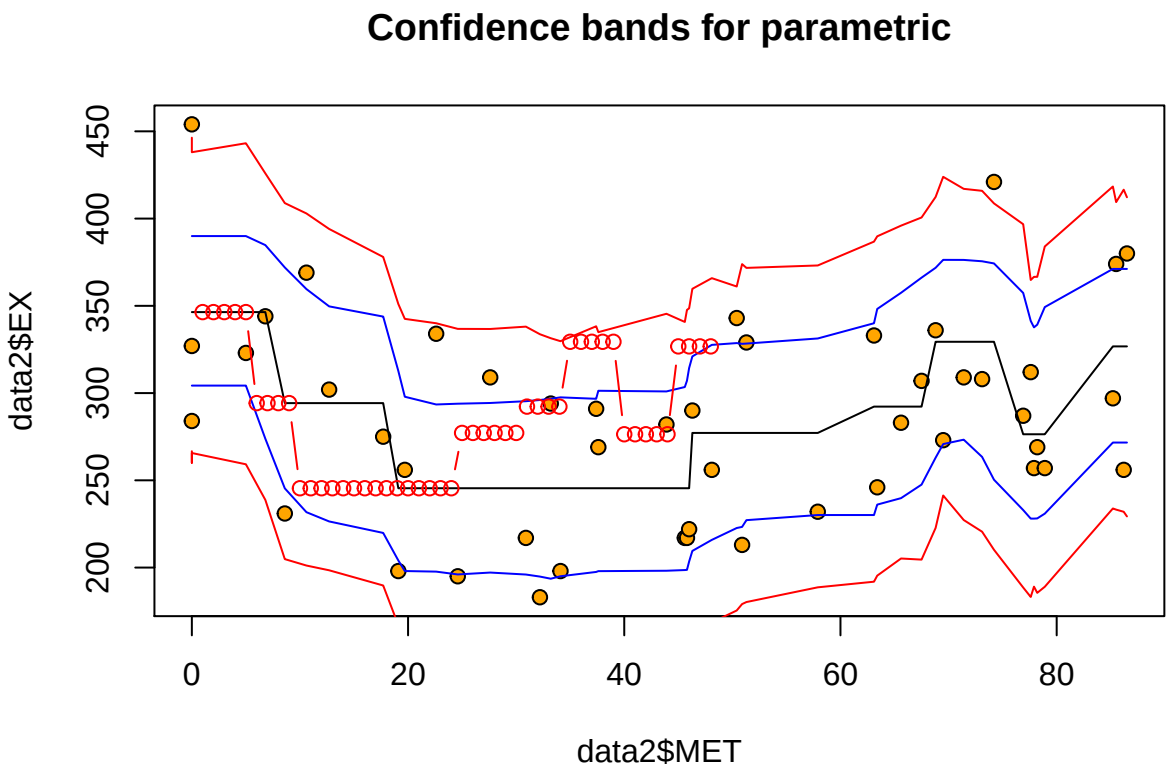
```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootResult, type = "norm")
##
## Intervals :
## Level      Normal
## 95%   (290.6, 413.2 )
## Calculations and Intervals on Original Scale
```

```
#Plotting confidence bands
confsetup<-tree.control(nrow(data2), minsize = 8)
fitPred <- tree(EX~MET, data=data2, control = confsetup)
plotPred <- predict(fitPred,data2)
#plot cofidence bands
plot(data2$MET, data2$EX, pch=21, bg="orange", main = "Confidence bands for non-parametric")
points(data2$MET,plotPred,type="l") #plot fitted line
points(data2$MET,e$point[2,], type="l", col="blue")
points(data2$MET,e$point[1,], type="l", col="blue")
points(predict(regTreeModel, newdata=data2), type="b", col="red")
```



#from the plot it's clear that the band is bumpy, and the model is very good nonetheless, a few data is out of the prediction band.

```
plotParametric <- predict(fitParametric,data2)
plot(data2$MET, data2$EX, pch=21, bg="orange", main = "Confidence bands for parametric")
points(data2$MET,plotParametric,type="l") #plot fitted line
points(data2$MET,eParamteric$point[2,], type="l", col="blue")
points(data2$MET,eParamteric$point[1,], type="l", col="blue")

points(data2$MET,predParametric$point[1,], type="l", col="red")
points(data2$MET,predParametric$point[2,], type="l", col="red")

points(predict(regTreeModel, newdata=data2), type="b", col="red")
```

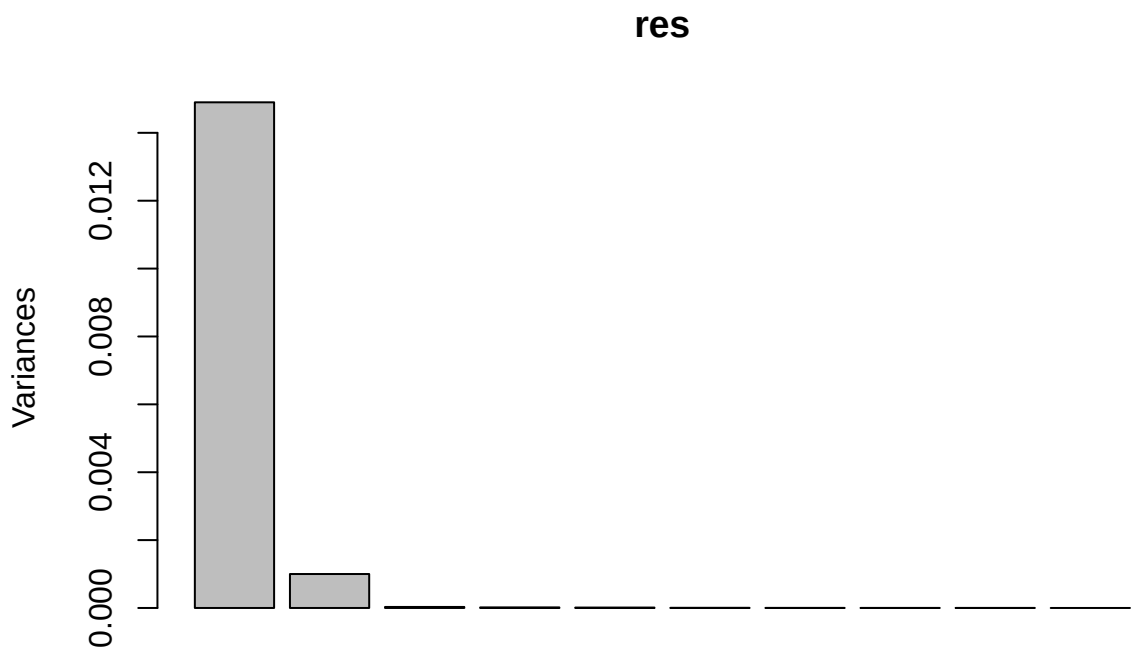


#from the plot it's clear that the band is bumpy, and the model is very good at representing the data, that is to say, all data lies within the prediction confidence bands, When using parametric bootstrapping the model gives a wider confidence interval than the case of non-parametric.

Assignment 4. Principal components

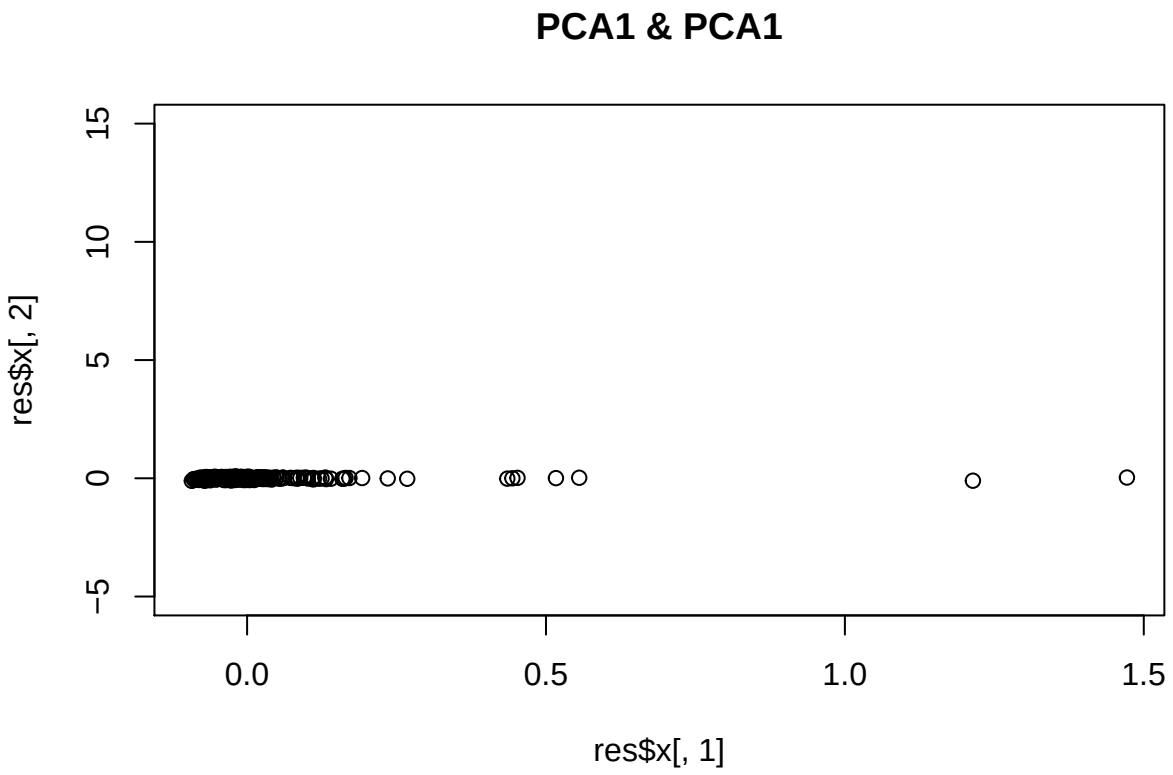
```
##   [1] 1.489914e-02 9.998545e-04 2.954195e-05 1.608532e-05 1.091077e-05
##   [6] 3.939315e-06 1.414911e-06 4.981545e-07 4.262849e-07 2.577774e-07
##  [11] 2.080001e-07 1.587511e-07 1.425823e-07 1.126727e-07 7.232246e-08
##  [16] 6.878939e-08 5.307373e-08 4.373598e-08 3.975200e-08 3.627181e-08
##  [21] 3.473207e-08 2.838554e-08 2.750156e-08 2.356802e-08 2.057859e-08
##  [26] 1.921151e-08 1.772579e-08 1.719151e-08 1.546958e-08 1.450458e-08
##  [31] 1.349010e-08 1.229577e-08 1.210005e-08 1.144210e-08 1.068630e-08
##  [36] 1.046807e-08 9.148433e-09 8.884085e-09 8.567593e-09 8.126130e-09
##  [41] 7.768325e-09 7.271742e-09 7.005011e-09 6.462762e-09 6.415715e-09
##  [46] 6.123419e-09 5.705293e-09 5.634860e-09 5.489343e-09 5.237779e-09
##  [51] 5.146764e-09 4.927885e-09 4.683481e-09 4.541157e-09 4.483382e-09
##  [56] 4.334378e-09 4.102898e-09 3.859924e-09 3.754631e-09 3.735784e-09
##  [61] 3.569046e-09 3.458093e-09 3.357414e-09 3.280258e-09 3.117910e-09
##  [66] 3.077594e-09 2.965870e-09 2.877830e-09 2.821708e-09 2.689767e-09
##  [71] 2.553543e-09 2.451608e-09 2.443009e-09 2.351475e-09 2.323987e-09
##  [76] 2.261444e-09 2.210244e-09 2.146417e-09 2.044353e-09 1.957622e-09
##  [81] 1.932558e-09 1.871133e-09 1.864625e-09 1.752181e-09 1.698602e-09
##  [86] 1.676920e-09 1.656262e-09 1.581932e-09 1.557666e-09 1.467634e-09
##  [91] 1.410370e-09 1.380420e-09 1.347822e-09 1.336314e-09 1.236098e-09
```

```
##  [96] 1.209134e-09 1.179186e-09 1.123574e-09 1.068961e-09 9.985917e-10
## [101] 9.858015e-10 9.526850e-10 8.980246e-10 8.728980e-10 8.288593e-10
## [106] 8.144341e-10 7.545504e-10 7.293165e-10 7.199373e-10 6.733103e-10
## [111] 6.580809e-10 6.168986e-10 6.099031e-10 5.763810e-10 5.525311e-10
## [116] 5.421167e-10 5.266189e-10 5.093443e-10 4.827224e-10 4.463711e-10
## [121] 4.127067e-10 3.895925e-10 3.786334e-10 3.495125e-10 3.033627e-10
## [126] 2.675822e-10

##   [1] "93.332" "6.263"  "0.185"  "0.101"  "0.068"  "0.025"  "0.009"  "0.003"
##   [9] "0.003"  "0.002"  "0.001"  "0.001"  "0.001"  "0.001"  "0.000"  "0.000"
##  [17] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [25] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [33] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [41] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [49] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [57] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [65] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [73] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [81] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [89] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
##  [97] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [105] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [113] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
## [121] "0.000"  "0.000"  "0.000"  "0.000"  "0.000"  "0.000"
```



#The plot shows that two PCAs should be extracted, and the PCAs that capture at least 99% of the toal variance are PCA1 and PCA2.

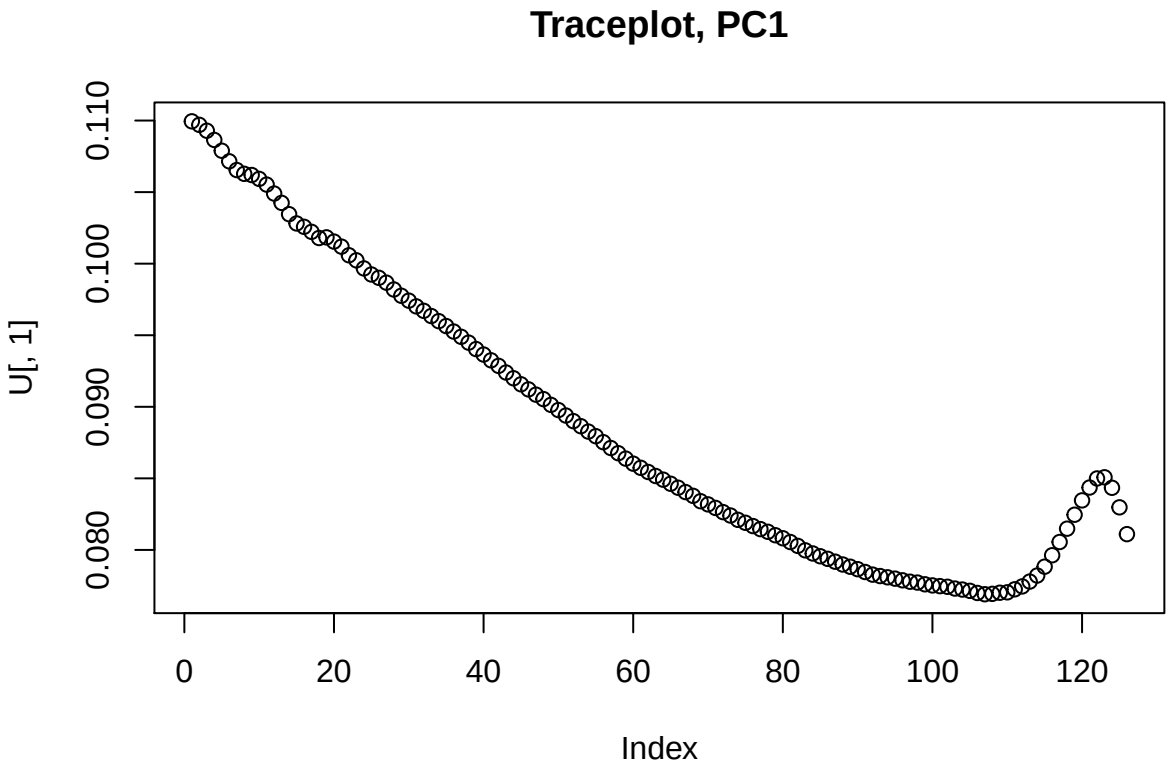
```r
U=res$rotation
#Plotting PCA1 & PCA2 in the coordinates
plot(res$x[,1], res$x[,2], ylim=c(-5,15), main = "PCA1 & PCA1")
```



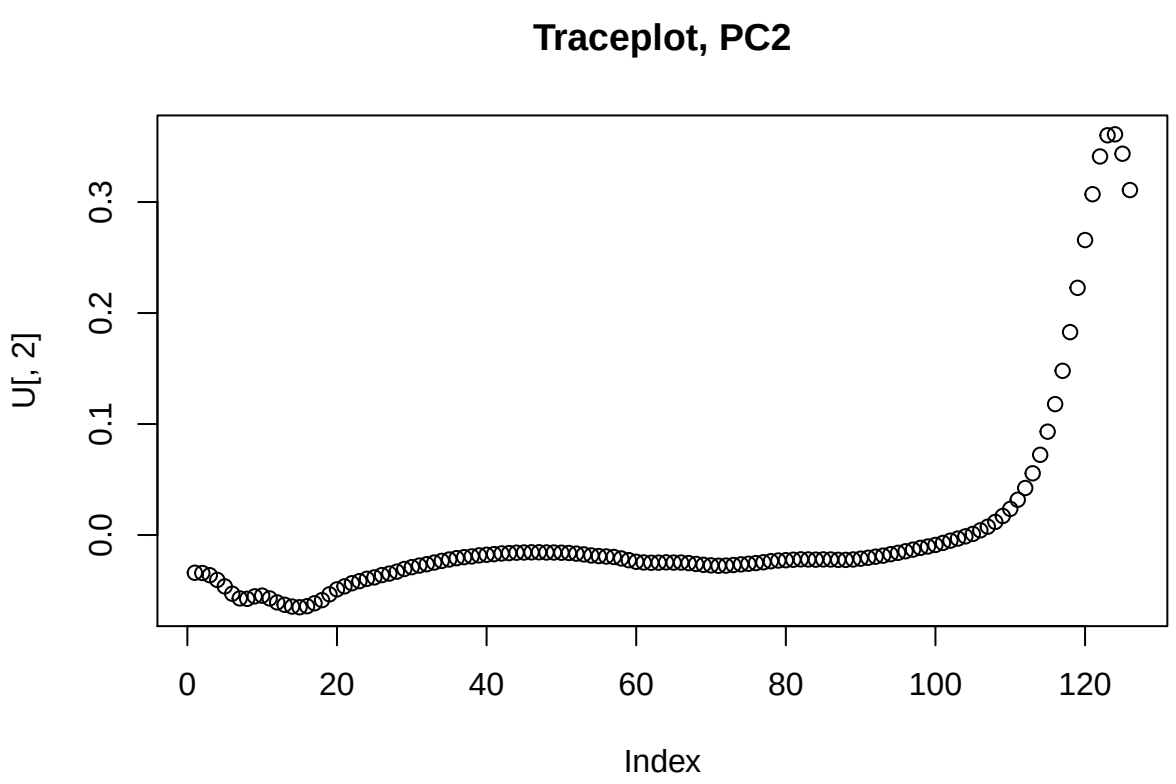

```r
#According to this plot, there are two unusual diesel fuels on the x axes.
head(U)[,1:2]
```

```
##             PC1         PC2
## X750 0.1099439 -0.03399551
## X752 0.1096993 -0.03423031
## X754 0.1092886 -0.03620695
## X756 0.1086458 -0.04045827
## X758 0.1078912 -0.04631170
## X760 0.1071514 -0.05288925
```

```r
#4.2) Trace plot
plot(U[,1], main="Traceplot, PC1")
```

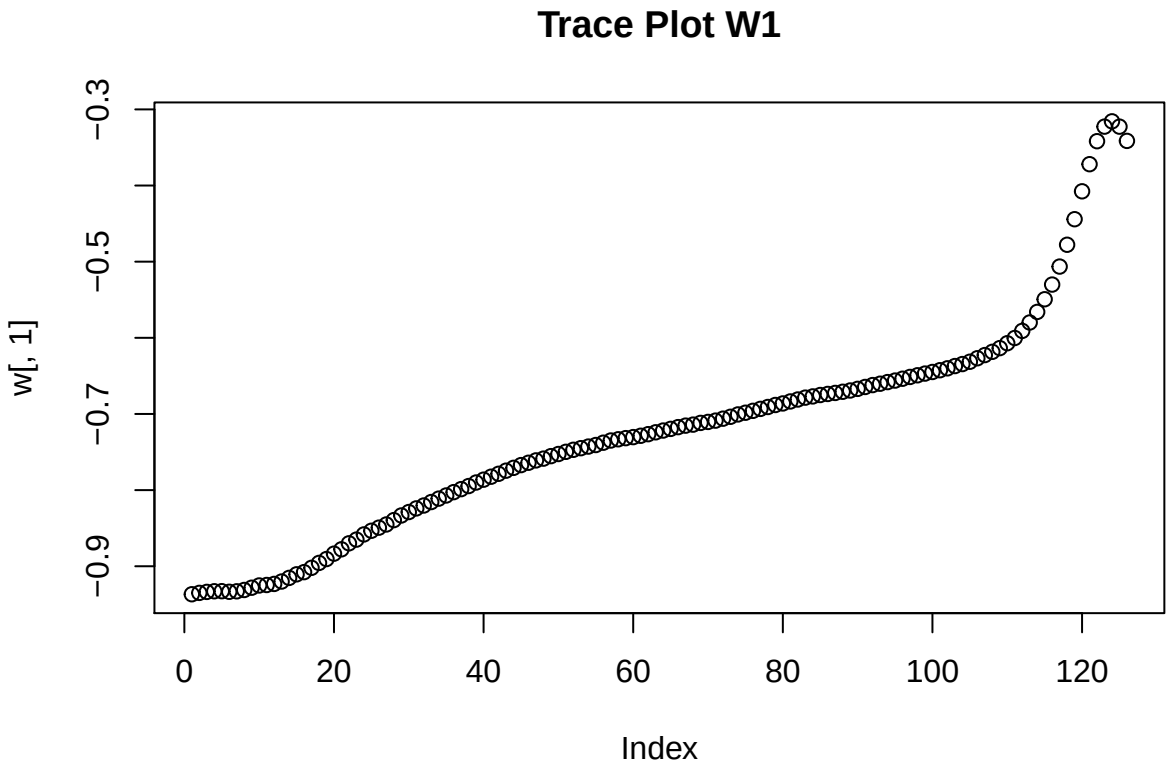


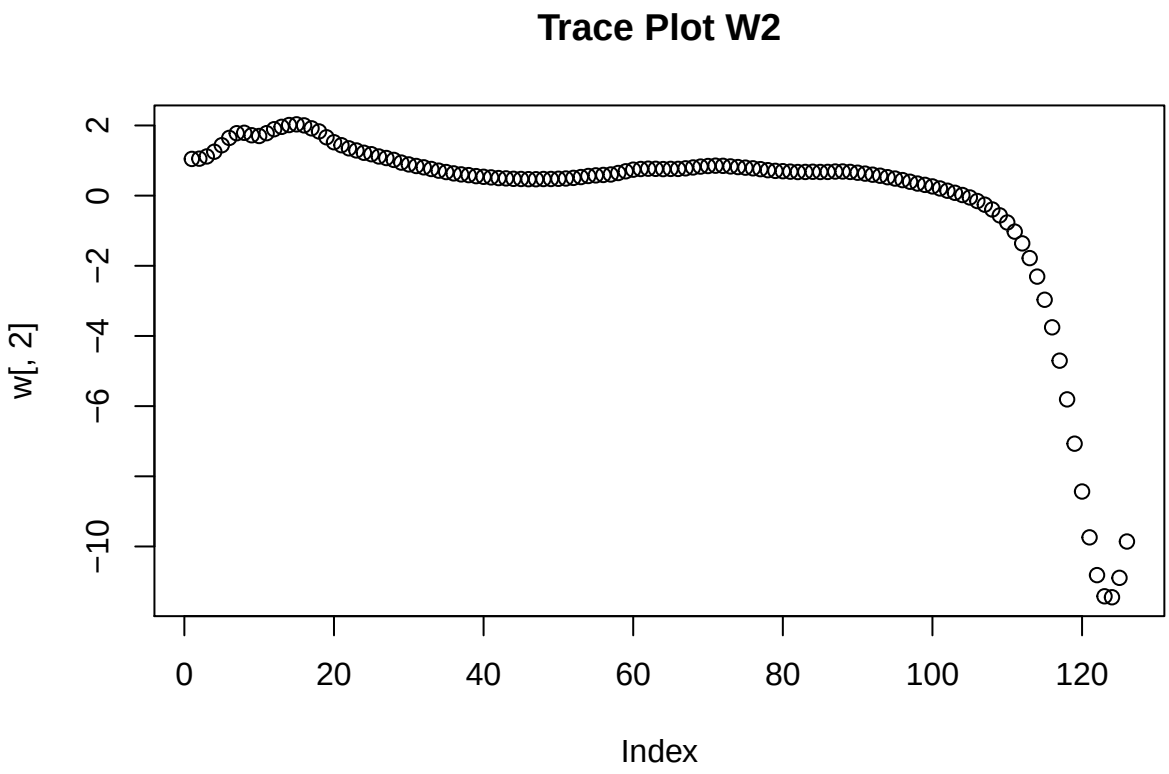

```
plot(U[,2],main="Traceplot, PC2")
```



#For the first PC (PC1), from the plot (Traceplot, PC1) it's clear that the range of data is from 0.08 to 0.110 which is less compared to PCA2. Whereas in PCA2 the range is from 0.0 to 0.3, which is larger compared to PC1, and most of the data is linearly centered arount 0.0

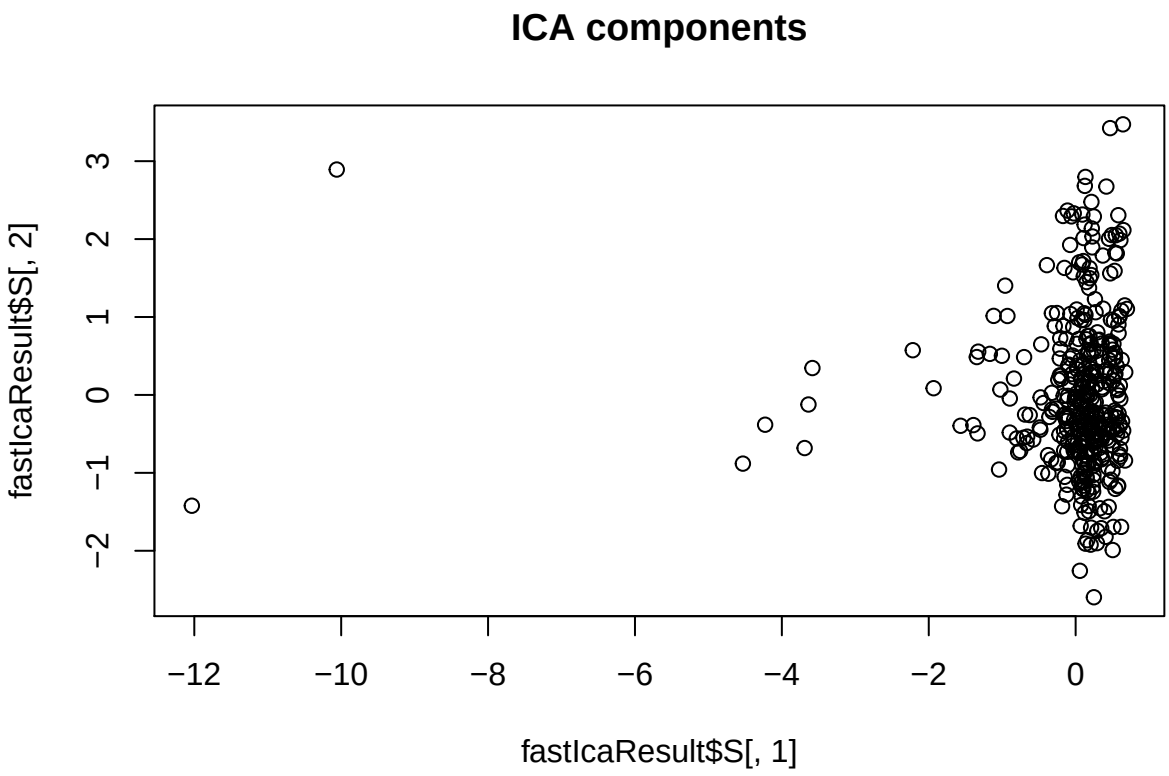
```
## Centering

## Whitening

## Symmetric FastICA using logcosh approx. to neg-entropy function

## Iteration 1 tol = 0.01930239

## Iteration 2 tol = 0.01303959

## Iteration 3 tol = 0.002393582

## Iteration 4 tol = 0.0006708454

## Iteration 5 tol = 0.0001661602

## Iteration 6 tol = 3.521604e-05
```


Trace Plot W1
w[, 1]
−0.3
−0.5
−0.7
−0.9
0
20
40
60
80
100
120
Index



#When comparing the second plot (Trace Plot W2), it's clear that PC2 ranges between -1.0 and -0.2 which is almost the same as when using step 2 W is a matrix estimated by the ICA in an attempt to un-mix the data.

```
plot(fastIcaResult$S[,1],fastIcaResult$S[,2], main = "ICA components")
```



Code Appendix

```
packages <- c("ggplot2", "plotly","readxl","tree", "MASS", "e1071", "boot", "fastICA")
options(tinytex.verbose = TRUE)
knitr::opts_chunk$set(echo = TRUE)
packages <- c("ggplot2", "plotly","readxl","tree", "MASS", "e1071", "boot", "fastICA")
options(tinytex.verbose = TRUE)
library(tree)
#1)Importing Data
data <- readxl::read_excel("D:/Desktop/Machine Learning/Machine Learning/lab02 block 1/creditscoring.xls
data$good_bad <- as.factor(data$good_bad)

#Dividing into training, validation, testing
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
remaining=data[-id,]
s=dim(remaining)[1]
validationId = sample(1:s, floor(s*0.5))
validationData = remaining[validationId,]
testingData = remaining[-validationId,]
```

```r
#2) Fitting the data into the tree model with Gini and deviance measurments

treeModelDeviance <- tree(good_bad ~.,data = train,split = "deviance")
summary(treeModelDeviance)
plot(treeModelDeviance)
text(treeModelDeviance,pretty = 0)
library(e1071)
treeModelGini <- tree(good_bad ~.,data = train,split = "gini")
summary(treeModelGini)
plot(treeModelGini, type = "uniform")
#text(treeModelGini,pretty = 0)
#Predicting for validation and testing data for deviance measurement
treePredTraining = predict(treeModelDeviance, newdata = train, type="class")
treePredValid = predict(treeModelDeviance, newdata = validationData, type="class")
treePredTest = predict(treeModelDeviance, newdata = testingData, type="class")

#Missclassification rate for training data for deviance
print(1 - mean(treePredTraining == train$good_bad))
table(treePredTraining,train$good_bad)

#Missclassification rate for validation data for deviance
print(1 - mean(treePredValid == validationData$good_bad))
table(treePredTest,validationData$good_bad)

#Missclassification rate for the testing data for deviance
print(1 - mean(treePredTest == testingData$good_bad))
table(treePredTest,testingData$good_bad)
#Predicting for validation and testing data for gini measurement
treePredTrainingG = predict(treeModelGini, newdata = train, type="class")
treePredValidG = predict(treeModelGini, newdata = validationData, type="class")
treePredTestG = predict(treeModelGini, newdata = testingData, type="class")
#Missclassification rate for training data for Gini measurment
print(1 - mean(treePredTrainingG == train$good_bad))
table(treePredTrainingG,train$good_bad)

#Missclassification rate for validation data for Gini measurment
print(1 - mean(treePredValidG == validationData$good_bad))
table(treePredValidG,validationData$good_bad)

#Missclassification rate for the testing data for Ginin measurment
print(1 - mean(treePredTestG == testingData$good_bad))
table(treePredTestG,testingData$good_bad)

#3) Finding the optimal tree by using training and validation datasets
treeModelBest <- tree(good_bad ~.,data = train)
trainScore=rep(0,9)
testScore=rep(0,9)
for(i in 2:9) {
  prunedTree=prune.tree(treeModelBest,best=i)
  pred=predict(prunedTree, newdata=validationData, type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
```

```r
plot(2:9, trainScore[2:9], type="b", col="red", ylim=c(0,700), main = "Training and Testing Scores")
points(2:9, testScore[2:9], type="b", col="blue")
#As the number of leaves increases the training and testing scores both decreases.

#The optimal tree is when the number of leaves is equal to four
finalTree=prune.tree(treeModelBest, best=4)
summary(finalTree)
plot(finalTree, main = "Best Tree")
text(finalTree,pretty = 0)
#The tree goes to the left side in one direction, the depth in the right direction is
#only one, whereas in the left direction it's three.

#Variables actually used in tree construction are:
#"savings"  "duration" "history"

#Number of terminal nodes:  4
#Misclassification error rate: 0.251

Yfit=predict(finalTree, newdata=validationData, type="class")

#Confusion matrix for the optimal tree
table(validationData$good_bad,Yfit)

#Missclassification rate for the optimal tree
print(1 - mean(Yfit == validationData$good_bad))

#The missclassification rate for the test data is: 0.26
YfitTest=predict(finalTree, newdata=testingData, type="class")
table(testingData$good_bad,YfitTest)
print(1 - mean(YfitTest == testingData$good_bad))


#4) Training data for classification using Naive Bayes

naiveModel <- naiveBayes(good_bad~., data=train)
summary(naiveModel)
YfitNaiveTrain=predict(naiveModel, newdata=train, type = "class")
YfitNaiveTest=predict(naiveModel, newdata=testingData, type = "class")

#Confusion matrix for the training data
table(YfitNaiveTrain,train$good_bad)

#Missclassification for the training data (0.3)
print(1 - mean(YfitNaiveTrain == train$good_bad))

#Confusion matrix for the testing data
table(YfitNaiveTest,testingData$good_bad)

#Missclassification for the test data (0.32)
print(1 - mean(YfitNaiveTest == testingData$good_bad))

#Comparing this result with the result from step 3, the missclassification rate is higher
#when using Naive Bayes at 0.32, whereas when using the optimal dicision tree the missclassiciation
```

```r
#rate is 0.26


#5) The optimal tree and Naive Bayes

set.seed(12345)
naiveModelFactored <- naiveBayes(good_bad~., data=train)
piValue <- seq(from = 0.05, to = 0.95, by = 0.05)

treeTPR <- c()
naiveTPR <- c()
naiveFPR <- c()
treeFPR <- c()
for(i in piValue){
  naive.probs <- predict(naiveModelFactored, newdata = testingData, type = "raw")
  optimal.probs <- predict(finalTree, newdata = testingData)

  naive.pred <- ifelse(naive.probs[,2] > i, 1, 0)
  naiveConfusion <- table(testingData$good_bad, naive.pred)

  optimal.tree.pred <- ifelse(optimal.probs[,2] > i,1,0)
  optimalConfusion <- table(testingData$good_bad,optimal.tree.pred)

  #Calculating naive TPR & FPR
  naiveTPR <- cbind(naiveTPR,naiveConfusion[1,1]/sum(naiveConfusion[1,]) )
  naiveFPR <- cbind(naiveFPR,naiveConfusion[2,1]/sum(naiveConfusion[2,]))

  #Calculating tree TPR & FPR
  treeTPR <- cbind(treeTPR,optimalConfusion[1,1]/sum(optimalConfusion[1,]) )
  treeFPR <- cbind(treeFPR,optimalConfusion[2,1]/sum(optimalConfusion[2,]) )
}

#Plotting TPR & FPR for tree model
plot(as.numeric(treeTPR), as.numeric(treeFPR), main = "Tree TPR & FRP")
#Plotting TPR & FPR for naive model
plot(naiveTPR, naiveFPR, main = "Naive TPR & FPR")

lossModel <- naiveBayes(good_bad~., train)

lossModelPred <- predict(lossModel, newdata = train, type = "raw")
condPrediction <- ifelse(lossModelPred[,2] > 0.1, "good", "bad")

#Confusion matrix for the training data
table(train$good_bad, condPrediction)

#Missclassification rate for the training data
lossMisClassification <- 1-mean(train$good_bad == condPrediction)
lossMisClassification

lossModelTest <- naiveBayes(good_bad~., train)
lossModelTesting <- predict(lossModelTest, newdata = testingData, type = "raw")
condPredTesting <- ifelse(lossModelTesting[,2] > .1, "1", "0")
```

```r
#Confusion matrix for the testing data
table(testingData$good_bad, condPredTesting)

#Missclassification rate for the testing data
lossMisClassificationTest <- 1-mean(testingData$good_bad == condPredTesting)
lossMisClassificationTest



library(ggplot2)
library(boot)
data <- read.csv2("D:/Desktop/Machine Learning/Machine Learning/lab02 block 1/State.csv")
data2=data[order(data$MET),]
# dev.off()
ggplot(data2,aes(x=EX,y =MET )) +geom_point() +geom_smooth(method = "lm")+ggtitle("MET vs EX")
#3.2) Regression tree model

      #Fitting the regression tree model with minimize 8 leaves, and plotting the tree
      setup<-tree.control(nrow(data2), minsize = 8)
      regTreeModel=tree(EX~MET, data2, control = setup)
#3.2) Regression tree model
      plot(regTreeModel)
      text(regTreeModel, pretty = 0)
      summary(regTreeModel)

      #applying cross validation for the tree
      cv.res=cv.tree(regTreeModel)
      plot(cv.res,main="Size of the tree being pruned")
      minDeviance <- which.min(cv.res$dev)
      hist(cv.res$k, main = "Cross Validation Residuals")
      #minumum number of leaves
      minLeaves <- cv.res$size[minDeviance]
      minLeaves
      #The optimal tree
      optimalTree=prune.tree(regTreeModel, best=3)
      plot(optimalTree, main="Optimal Tree")
      text(optimalTree,pretty = 0, col="blue")
      #Residuals
      plot(cv.res$size, cv.res$dev, type="b", col="red")
      plot(log(cv.res$k), cv.res$dev,type="b", col="red")
#3.3) 95% confidence band
      f=function(data, ind){
        data1=data[ind,]# extract bootstrap sample
        confsetup<-tree.control(nrow(data1), minsize = 8)
        res=tree(EX~MET, data=data1, control = confsetup) #fit linear model
        #predict values for all Area values from the original data
        priceP=predict(res,newdata=data2)
        return(priceP)
      }
      #Make a bootstrap
      bootResult <- boot(data2,statistic = f,R=1000)
      plot(bootResult)
      e=envelope(bootResult) #compute confidence bands
```

```
      summary(e)

      #Calculating the boot confidence interval
      bootCi <- boot.ci(boot.out = bootResult, type = "norm")
      bootCi
      #Plotting confidence bands
      confsetup<-tree.control(nrow(data2), minsize = 8)
      fitPred <- tree(EX~MET, data=data2, control = confsetup)
      plotPred <- predict(fitPred,data2)
      #plot cofidence bands
      plot(data2$MET, data2$EX, pch=21, bg="orange", main = "Confidence bands for non-parametric")
      points(data2$MET,plotPred,type="l") #plot fitted line
      points(data2$MET,e$point[2,], type="l", col="blue")
      points(data2$MET,e$point[1,], type="l", col="blue")
      points(predict(regTreeModel, newdata=data2), type="b", col="red")

#3.4) 95% parametric bootstrap
      treeSetup <- tree.control(nrow(data2), minsize = 8)
      fitParametric <- tree(EX~MET, data=data2, control = treeSetup)

      rng=function(data, fitParametric) {
        data1=data.frame(EX=data$EX, MET=data$MET)
        n=length(data$EX)
        #generate new Price
        data1$EX=rnorm(n,predict(fitParametric, newdata=data1),sd(residuals(fitParametric)))
        return(data1)
      }
      # predict(fitParametric, newdata=data2)
      f1=function(data1){
        treeSetupf <- tree.control(nrow(data1), minsize = 8)
        fitParametricModel <- tree(EX~MET, data=data1, control = treeSetupf) #fit tree model
        #predict values for all Area values from the original data
        priceP=predict(fitParametricModel,newdata=data2)
        return(priceP)
      }
      f2 <- function(data1){
        treeSetupf <- tree.control(nrow(data1), minsize = 8)
        fitParametricModel <- tree(EX~MET, data=data1, control = treeSetupf) #fit tree model

        n = length(data1$EX)
        predictPred <- predict(fitParametricModel, newdata = data2)
        pred <- rnorm(n,predictPred, sd(residuals(fitParametricModel)))
        return(pred)

      }

      resParametric=boot(data2, statistic=f1, R=1000, mle=fitParametric,ran.gen=rng, sim="parametric")
      resParametricPred=boot(data2, statistic=f2, R=1000, mle=fitParametric,ran.gen=rng, sim="parametri
      eParamteric <- envelope(resParametric) #compute confidence bands
      predParametric <- envelope(resParametricPred) #compute confidence for the prediction

      #Confidence band for the data
          #eParamteric
```

```
      #Prediction confidence
          #predParametric
      #fitting and plotting the model
      plotParametric <- predict(fitParametric,data2)
      plot(data2$MET, data2$EX, pch=21, bg="orange", main = "Confidence bands for parametric")
      points(data2$MET,plotParametric,type="l") #plot fitted line
      points(data2$MET,eParamteric$point[2,], type="l", col="blue")
      points(data2$MET,eParamteric$point[1,], type="l", col="blue")

      points(data2$MET,predParametric$point[1,], type="l", col="red")
      points(data2$MET,predParametric$point[2,], type="l", col="red")

      points(predict(regTreeModel, newdata=data2), type="b", col="red")
library(fastICA)
#4.1) standard PCA
    pcaData <- read.csv2("D:/Desktop/Machine Learning/Machine Learning/lab02 block 1/NIRSpectra.csv")

    data1=pcaData
    data1$Viscosity = c()
    data1$Fat=c()
    res=prcomp(data1)
    lambda=res$sdev^2
    #eigenvalues
    lambda
          #proportion of variation
          sprintf("%2.3f",lambda/sum(lambda)*100)
          screeplot(res)
          U=res$rotation
          #Plotting PCA1 & PCA2 in the coordinates
          plot(res$x[,1], res$x[,2], ylim=c(-5,15), main = "PCA1 & PCA1")
          #According to this plot, there are two unusual diesel fuels on the x axes.
          head(U)[,1:2]

#4.2) Trace plot
          plot(U[,1], main="Traceplot, PC1")
          plot(U[,2],main="Traceplot, PC2")
#4.3) Independent Component Analysis

          set.seed(12345)
          #Fitting fastICA function
          icaData <- as.matrix(data1)
          fastIcaResult <- fastICA(icaData, n.comp = 2,
                                   fun = "logcosh",
                                   method = "R",
                                   alpha = 1,
                                   maxit = 200,
                                   tol = 0.0001,
                                   verbose = TRUE)

          #computing the W
          w <- fastIcaResult$K %*% fastIcaResult$W

          #Plotting w as traceplot
```

```
        plot(w[,1], main = "Trace Plot W1")
        plot(w[,2], main = "Trace Plot W2")
        plot(fastIcaResult$S[,1],fastIcaResult$S[,2], main = "ICA components")
```