machine learning(732A99) lab2

Anubhav Dikshit(anudi287)

10 December 2018

Contents

Loading The Libraries

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(xlsx, ggplot2, MASS, tidyr, dplyr, reshape2, gridExtra,
               tree, caret, e1071, pROC, boot, factoextra, fastICA)

set.seed(12345)
options("jtools-digits" = 2, scipen = 999)

# colours (colour blind friendly)
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
```

Assignment 2

2.1 Import the data to R and divide into training/validation/test as 50/25/25: use data partitioning code specified in Lecture 1e.

```
set.seed(12345)
credit_data <- read.xlsx("creditscoring.xls", sheetName = "credit")
credit_data$good_bad <- as.factor(credit_data$good_bad)


n=NROW(credit_data)
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=credit_data[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=credit_data[id2,]

id3=setdiff(id1,id2)
test=credit_data[id3,]
```

2.2 Fit a decision tree to the training data by using the following measures of impurity: a. Deviance b. Gini index and report the misclassification rates for the training and test data. Choose the measure providing the better results for the following steps.

```
set.seed(12345)

# Create a decision tree model
credit_tree_deviance <- tree(good_bad~., data=train, split = c("deviance"))
credit_tree_gini <- tree(good_bad~., data=train, split = c("gini"))

# Visualize the decision tree with rpart.plot
summary(credit_tree_deviance)
```

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = c("deviance"))
## Variables actually used in tree construction:
## [1] "savings"  "duration" "history"  "age"      "purpose"  "amount"
## [7] "resident" "other"
## Number of terminal nodes:  15
## Residual mean deviance:  0.9569 = 458.3 / 479
## Misclassification error rate: 0.2105 = 104 / 494
```

```
summary(credit_tree_gini)
```

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = c("gini"))
## Variables actually used in tree construction:
##  [1] "foreign"  "coapp"    "depends"  "telephon" "existcr"  "savings"
##  [7] "history"  "property" "marital"  "duration" "employed" "age"
## [13] "housing"  "amount"   "purpose"  "resident" "job"      "installp"
## Number of terminal nodes:  72
## Residual mean deviance:  1.015 = 428.5 / 422
## Misclassification error rate: 0.2368 = 117 / 494
```

```
# predicting on the test dataset to get the misclassification rate.
predict_tree_deviance <- predict(credit_tree_deviance, newdata = test, type = "class")
predict_tree_gini <- predict(credit_tree_gini, newdata = test, type = "class")

conf_tree_deviance <- table(test$good_bad, predict_tree_deviance)
names(dimnames(conf_tree_deviance)) <- c("Actual Test", "P2redicted Test")
caret::confusionMatrix(conf_tree_deviance)
```

```
## Confusion Matrix and Statistics
##
##            P2redicted Test
## Actual Test bad good
##        bad   28   48
##        good  19  155
##
##                Accuracy : 0.732
##                  95% CI : (0.6725, 0.7859)
##     No Information Rate : 0.812
##     P-Value [Acc > NIR] : 0.9992613
##
##                   Kappa : 0.2904
##  Mcnemar's Test P-Value : 0.0006245
##
##             Sensitivity : 0.5957
##             Specificity : 0.7635
##          Pos Pred Value : 0.3684
##          Neg Pred Value : 0.8908
##              Prevalence : 0.1880
##          Detection Rate : 0.1120
##    Detection Prevalence : 0.3040
##       Balanced Accuracy : 0.6796
##
```

```
##        'Positive' Class : bad
##
```

```
conf_tree_gini <- table(test$good_bad, predict_tree_gini)
names(dimnames(conf_tree_gini)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_tree_gini)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test bad good
##        bad   18   58
##        good  33  141
##
##                Accuracy : 0.636
##                  95% CI : (0.573, 0.6957)
##     No Information Rate : 0.796
##     P-Value [Acc > NIR] : 1.00000
##
##                   Kappa : 0.052
##  Mcnemar's Test P-Value : 0.01187
##
##             Sensitivity : 0.3529
##             Specificity : 0.7085
##          Pos Pred Value : 0.2368
##          Neg Pred Value : 0.8103
##              Prevalence : 0.2040
##          Detection Rate : 0.0720
##    Detection Prevalence : 0.3040
##       Balanced Accuracy : 0.5307
##
##        'Positive' Class : bad
##
```

Analysis: On the Training dataset model with 'deviance' had a misclassfication rate of 21.05% while the model with 'gini' split had the misclassification rate of 23.68%.

For the test dataset we see that the model with 'deviance' type of split has a accuracy of 73.2% or misclassifiaction rate of 26.8%, we see that to predict 'good' the accuracy is 89.08% but for predicting bad its just 36.84%. Thus our our model is heavily baised towards predicting cases as 'good'.

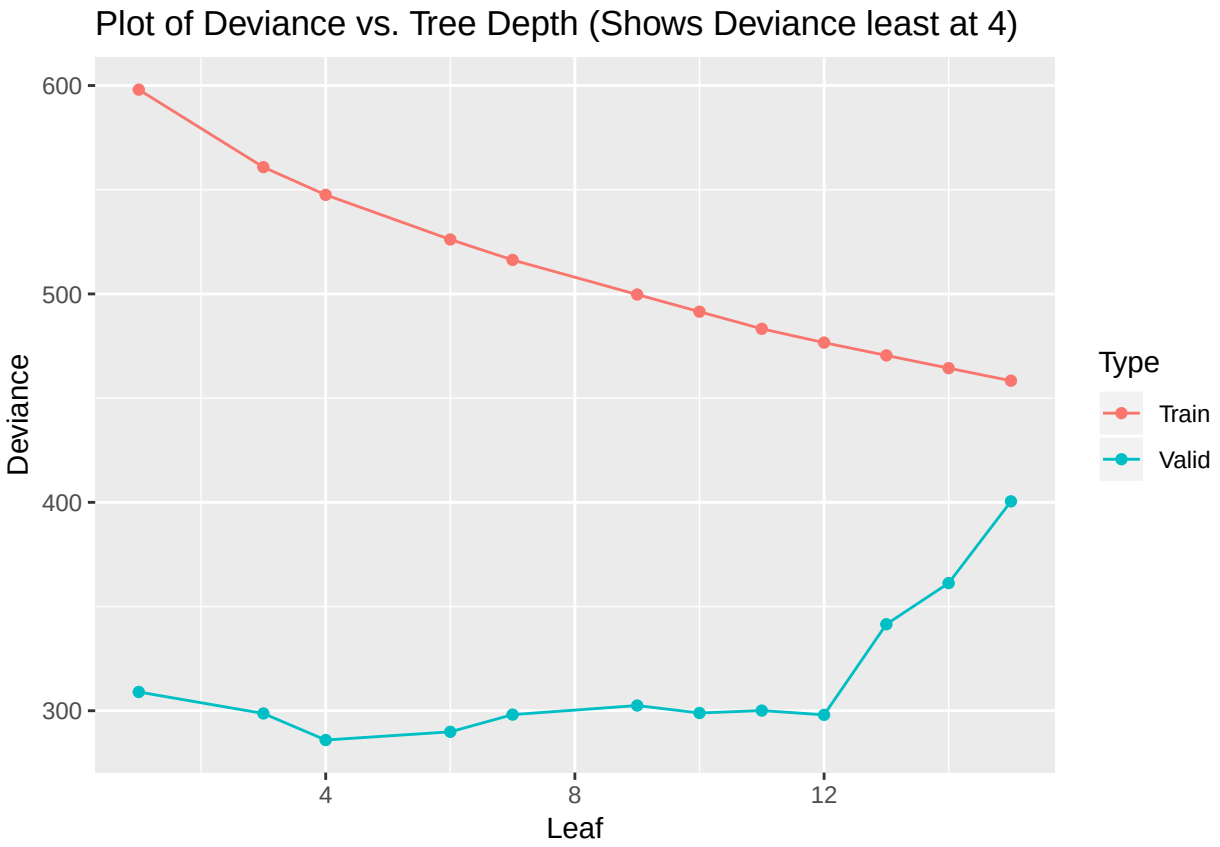
For the test dataset we see that the model with 'gini' type of split has a accuracy of 63.6% or misclassifiaction rate of 36.4%, we also see that to predict 'good' the accuracy is 81.03% but for predicting bad its just 18.97%. Thus our model is heavily baised towards predicting cases as 'good' even more than the model which uses 'deviance' to split variable.

Both our models would lead to many bad loan applicants to be given loans which is never a good thing, however among the model the one using 'deviance' mode for split is better by 9.6%.

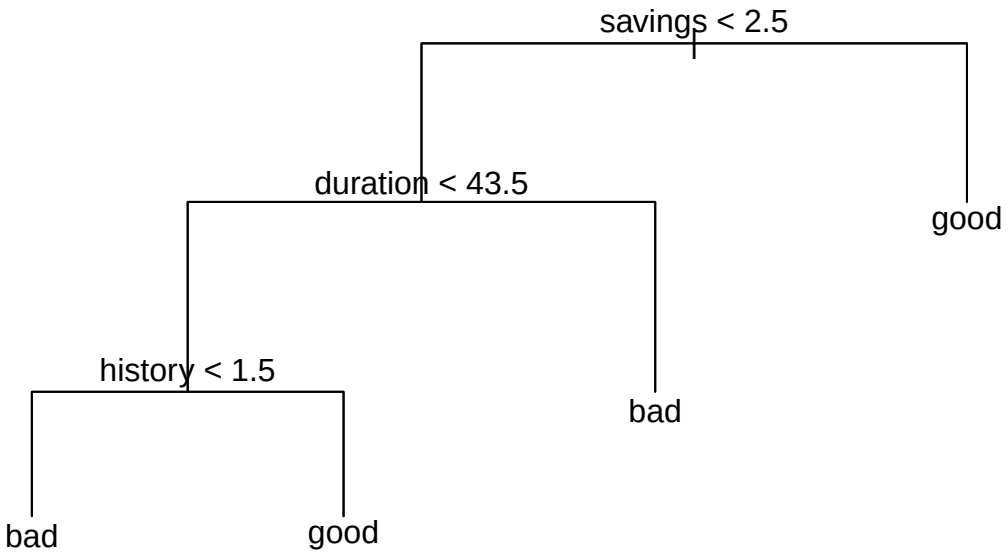
Thus we will select model using 'deviance' for further model building.

3. Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by the tree structure. Estimate the misclassification rate for the test data.

```
set.seed(12345)

credit_tree <- tree(good_bad~., data=train, split = c("deviance"))

credit_tree_purned_train <- prune.tree(credit_tree, method = c("deviance"))
credit_tree_purned_valid <- prune.tree(credit_tree, newdata = valid ,method = c("deviance"))

result_train <- cbind(credit_tree_purned_train$size,
                      credit_tree_purned_train$dev, "Train")
result_valid <- cbind(credit_tree_purned_valid$size,
                      credit_tree_purned_valid$dev, "Valid")

result <- as.data.frame(rbind(result_valid, result_train))
colnames(result) <- c("Leaf", "Deviance", "Type")

result$Leaf <- as.numeric(as.character(result$Leaf))
result$Deviance <- as.numeric(as.character(result$Deviance))

# plot of deviance vs. number of leafs
ggplot(data = result, aes(x = Leaf, y = Deviance, colour = Type)) +
  geom_point() + geom_line() +
  ggtitle("Plot of Deviance vs. Tree Depth (Shows Deviance least at 4)")
```




```
# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=4)

plot(credit_tree_sniped)
text(credit_tree_sniped)
```




```
# misclassification rate for best pruned tree

result_prune_test <- predict(credit_tree_sniped, newdata = test, type = "class")

conf_prune_tree_test <- table(test$good_bad, result_prune_test)
names(dimnames(conf_prune_tree_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_prune_tree_test)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test bad good
##        bad   18   58
##        good   6  168
##
##                Accuracy : 0.744
##                  95% CI : (0.6852, 0.7969)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2507
##  Mcnemar's Test P-Value : 0.000000000183
##
##             Sensitivity : 0.7500
##             Specificity : 0.7434
##          Pos Pred Value : 0.2368
```

```
##          Neg Pred Value : 0.9655
##              Prevalence : 0.0960
##          Detection Rate : 0.0720
##    Detection Prevalence : 0.3040
##       Balanced Accuracy : 0.7467
##
##        'Positive' Class : bad
##
```

Analysis: Choosing optimal depth tree we get that '4' as the best depth. The variables used in the best tree are- duration, history, savings.

From the tree structure we can see that the following variables are best to split on, 'savings' < 2.5 then 'duration' < 43.5 and 'history' < 1.5.

The accuracy on the model trained on 'train' dataset is accuracy 73.2% and misclassifiaction 26.8% while on the 'test' dataset accuracy is 74.4%, thus the misclassification rate is 25.6%. We see that model predicts 'good' applicants very well (accuracy of 96.55%) while it classifies 'bad' applicant way badly (accuracy is 23.68%).

Thus this model would be very bad for the business and would likely to run the business bankrupt.

4. Use training data to perform classification using Naïve Bayes and report the confusion matrices and misclassification rates for the training and for the test data. Compare the results with those from step 3.

```
#Fitting the Naive Bayes model
credit_naive_model = naiveBayes(good_bad ~., data=train)
credit_naive_model
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   bad  good
## 0.294 0.706
##
## Conditional probabilities:
##       duration
## Y          [,1]     [,2]
##   bad  24.03401 14.00998
##   good 19.00283 11.12851
##
##       history
## Y          [,1]     [,2]
##   bad  2.278912 1.084020
##   good 2.742210 1.038384
##
##       purpose
## Y          [,1]     [,2]
##   bad  2.662069 2.865554
```

```
##   good 2.581662 2.364077
##
##       amount
## Y          [,1]     [,2]
##   bad  3550.041 3321.899
##   good 2987.490 2339.068
##
##       savings
## Y          [,1]     [,2]
##   bad  1.707483 1.304423
##   good 2.274788 1.625739
##
##       employed
## Y          [,1]     [,2]
##   bad  3.129252 1.251148
##   good 3.430595 1.148801
##
##       installp
## Y          [,1]     [,2]
##   bad  3.183673 1.000140
##   good 2.875354 1.131315
##
##       marital
## Y          [,1]      [,2]
##   bad  2.619048 0.7524693
##   good 2.728045 0.6654893
##
##       coapp
## Y          [,1]      [,2]
##   bad  1.088435 0.3687292
##   good 1.167139 0.5198853
##
##       resident
## Y          [,1]     [,2]
##   bad  2.802721 1.063977
##   good 2.790368 1.111008
##
##       property
## Y          [,1]     [,2]
##   bad  2.448980 1.041493
##   good 2.288952 1.069365
##
##       age
## Y          [,1]     [,2]
##   bad  33.07483 10.74810
##   good 36.10198 11.01372
##
##       other
## Y          [,1]      [,2]
##   bad  2.564626 0.7769631
##   good 2.753541 0.6341558
##
##       housing
## Y          [,1]      [,2]
```

```
##   bad  1.918367 0.5911471
##   good 1.957507 0.4953265
##
##       existcr
## Y          [,1]      [,2]
##   bad  1.380952 0.5773503
##   good 1.439093 0.5812910
##
##       job
## Y          [,1]      [,2]
##   bad  2.863946 0.6987215
##   good 2.858357 0.6416373
##
##       depends
## Y          [,1]      [,2]
##   bad  1.136054 0.3440185
##   good 1.164306 0.3710790
##
##       telephon
## Y          [,1]      [,2]
##   bad  1.346939 0.4776234
##   good 1.382436 0.4866721
##
##       foreign
## Y          [,1]      [,2]
##   bad  1.013605 0.1162422
##   good 1.050992 0.2202926
```

```r
#Prediction on the dataset
predict_naive_train = predict(credit_naive_model, newdata=train, type = "class")
predict_naive_test = predict(credit_naive_model, newdata=test, type = "class")

conf_naive_train <- table(train$good_bad, predict_naive_train)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)
```

```
## Confusion Matrix and Statistics
##
##              Predicted Train
## Actual Train bad good
##         bad   95   52
##         good  98  255
##
##                Accuracy : 0.7
##                  95% CI : (0.6577, 0.7399)
##     No Information Rate : 0.614
##     P-Value [Acc > NIR] : 0.00003655
##
##                   Kappa : 0.3378
##  Mcnemar's Test P-Value : 0.0002386
##
##             Sensitivity : 0.4922
##             Specificity : 0.8306
##          Pos Pred Value : 0.6463
##          Neg Pred Value : 0.7224
```

```
##              Prevalence : 0.3860
##          Detection Rate : 0.1900
##    Detection Prevalence : 0.2940
##       Balanced Accuracy : 0.6614
##
##        'Positive' Class : bad
##
```

```
conf_naive_test <- table(test$good_bad, predict_naive_test)
names(dimnames(conf_naive_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_naive_test)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test bad good
##        bad   46   30
##        good  49  125
##
##                Accuracy : 0.684
##                  95% CI : (0.6224, 0.7411)
##     No Information Rate : 0.62
##     P-Value [Acc > NIR] : 0.02075
##
##                   Kappa : 0.3024
##  Mcnemar's Test P-Value : 0.04285
##
##             Sensitivity : 0.4842
##             Specificity : 0.8065
##          Pos Pred Value : 0.6053
##          Neg Pred Value : 0.7184
##              Prevalence : 0.3800
##          Detection Rate : 0.1840
##    Detection Prevalence : 0.3040
##       Balanced Accuracy : 0.6453
##
##        'Positive' Class : bad
##
```

Analysis:

For the train dataset using NaiveBayes method we get accuracy 70% or misclassification of 30%, here we also notice that the accuracy of class 'bad' is 64.63% while for class 'good' is 72.24%, thus the model is more balanced in predicting, thus its still baised in predict one class over the other.

For the test dataset using NaiveBayes method we get accuracy 68.4% or misclassification of 31.6%, here we also notice that the accuracy of class 'bad' is 60.53% while for class 'good' is 71.84%, thus the model is almost the same compared to train.

Compared to step3, we see that for the 'train' dataset the optimal tree has accuracy of 73.2% while it is 74.4% on the 'test' dataset. For the NaiveBayes model, accuracy on the 'train' dataset is 70% and while it is 68.4% on the 'test' datatset.

Accuracy is only part of the story what we see is better here is that this model classifies 'bad' customers better better for both train and test dataset than decision tree (60+% for both train and test for naive compared 36.84% train, 23.68% test for decision tree).

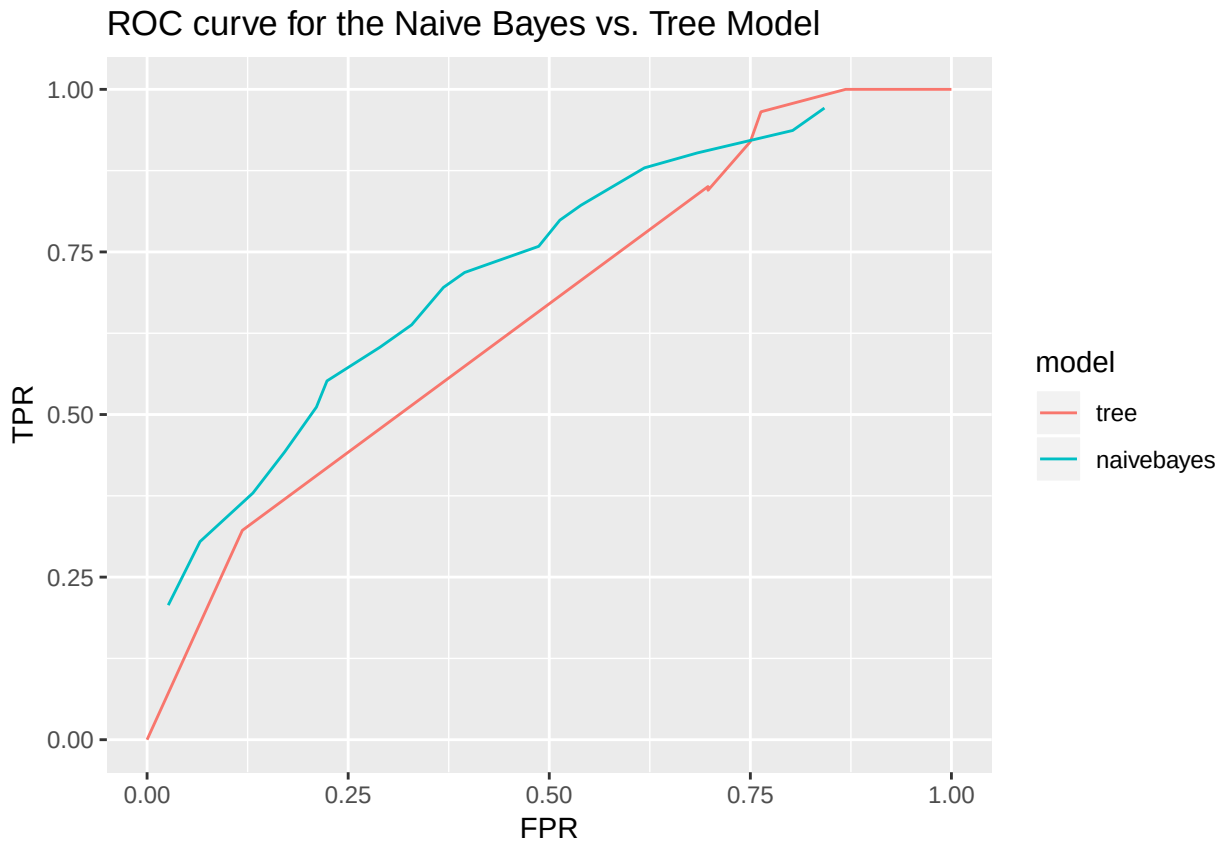
Thus the model is better to be used for the business than the one in the step3, the risk of providing loans to

bad applicant is lesser than the previous model but its still not good enough!

5. Use the optimal tree and the Naïve Bayes model to classify the test data by using the following principle: where prob(Y|'good')=A, where A=0.05,0.10,. . . ..0.95. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion?

```
set.seed(12345)

credit_tree <- tree(good_bad~., data=train, split = c("deviance"))
credit_naive_model = naiveBayes(good_bad ~., data=train)

# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=7)

# predicting class, getting probability
predict_prune_test_prob <- predict(credit_tree_sniped, newdata = test)
predict_naive_test_prob <- predict(credit_naive_model, newdata=test, type = "raw")

# data mugging
probability_data_naive <- as.data.frame(cbind(predict_naive_test_prob,
                                              as.character(test$good_bad), "naivebayes"))
probability_data_tree <- as.data.frame(cbind(predict_prune_test_prob,
                                             as.character(test$good_bad), "tree"))

probability_data_combined <- rbind(probability_data_tree, probability_data_naive)
colnames(probability_data_combined) <- c("prob_bad", "prob_good",
                                         "actual_test_class", "model")

# final dataset
probability_data_combined$prob_good <- as.numeric(as.character(probability_data_combined$prob_good))

# changing the threshold and printing the probability

tree_list <- NULL
naive_list <- NULL
final <- NULL
for(threshold in seq(from = 0.05, to = 0.95, by = 0.05)){
  probability_data_combined$predicted_class <- ifelse(probability_data_combined$prob_good > threshold,

  df2 <- probability_data_combined[,c("model", "actual_test_class", "predicted_class")]
  df2$threshold <- threshold
  df2$match <- ifelse(df2$actual_test_class == df2$predicted_class, 1, 0)

  final <- rbind(df2, final)
}

# Creating the FRP and TRP for each model and threshold
final$temp <- 1
final_summary <- final %>%
group_by(model, threshold) %>%
summarise(total_positive = sum(temp[actual_test_class == "good"]),
```

```
          total_negative = sum(temp[actual_test_class == "bad"]),
          correct_positive = sum(temp[actual_test_class == "good" & predicted_class == "good"]),
          false_positive = sum(temp[actual_test_class == "bad" & predicted_class == "good"])) %>%
    mutate(TPR = correct_positive/total_positive, FPR = false_positive/total_negative) %>%
  select(model, threshold, TPR, FPR)

ggplot(data = final_summary, aes(x = FPR, y=TPR)) + geom_line(aes(colour = model)) +
  ggtitle("ROC curve for the Naive Bayes vs. Tree Model")
```



Analysis: We find that 'naivebayes' model is better than 'tree' model for almost across varying threshold values, except the higher values of threshold(0.75+). Thus the decision of using NaiveBayes model here is further reinforced using the ROC curve.

6. Repeat Naïve Bayes classification as it was in step 4 but use the following loss matrix (good loss 1, bad loss 10) and report the confusion matrix for the training and test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```
set.seed(12345)

credit_naive_model = naiveBayes(good_bad ~., data=train)

# predicting class, getting probability
predict_naive_train_prob <- predict(credit_naive_model, newdata=train, type = "raw")
predict_naive_test_prob <- predict(credit_naive_model, newdata=test, type = "raw")
```

```r
train <- cbind(predict_naive_train_prob, train)
test <- cbind(predict_naive_test_prob, test)

# class based on the loss matrix
train$predicted_class <- ifelse(train$good > 10*train$bad, "good", "bad")
test$predicted_class <- ifelse(test$good > 10*test$bad, "good", "bad")

# confusion matrix
conf_naive_train <- table(train$good_bad, train$predicted_class)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)
```

```
## Confusion Matrix and Statistics
##
##              Predicted Train
## Actual Train bad good
##         bad  137   10
##         good 263   90
##
##                Accuracy : 0.454
##                  95% CI : (0.4097, 0.4988)
##     No Information Rate : 0.8
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1244
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.3425
##             Specificity : 0.9000
##          Pos Pred Value : 0.9320
##          Neg Pred Value : 0.2550
##              Prevalence : 0.8000
##          Detection Rate : 0.2740
##    Detection Prevalence : 0.2940
##       Balanced Accuracy : 0.6213
##
##        'Positive' Class : bad
##
```

```r
conf_naive_test <- table(test$good_bad, test$predicted_class)
names(dimnames(conf_naive_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_naive_test)
```

```
## Confusion Matrix and Statistics
##
##             Predicted Test
## Actual Test bad good
##        bad   71    5
##        good 122   52
##
##                Accuracy : 0.492
##                  95% CI : (0.4284, 0.5557)
##     No Information Rate : 0.772
##     P-Value [Acc > NIR] : 1
```

```
##
##                   Kappa : 0.1626
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.3679
##             Specificity : 0.9123
##          Pos Pred Value : 0.9342
##          Neg Pred Value : 0.2989
##              Prevalence : 0.7720
##          Detection Rate : 0.2840
##    Detection Prevalence : 0.3040
##       Balanced Accuracy : 0.6401
##
##        'Positive' Class : bad
##
```

Analysis: We see a major drop in accuracy for both train and test cases for using the new loss matrix, the accuracy for train is 45.4% and for test cases its 49.2%, this was 70% and 68.4% respectively.

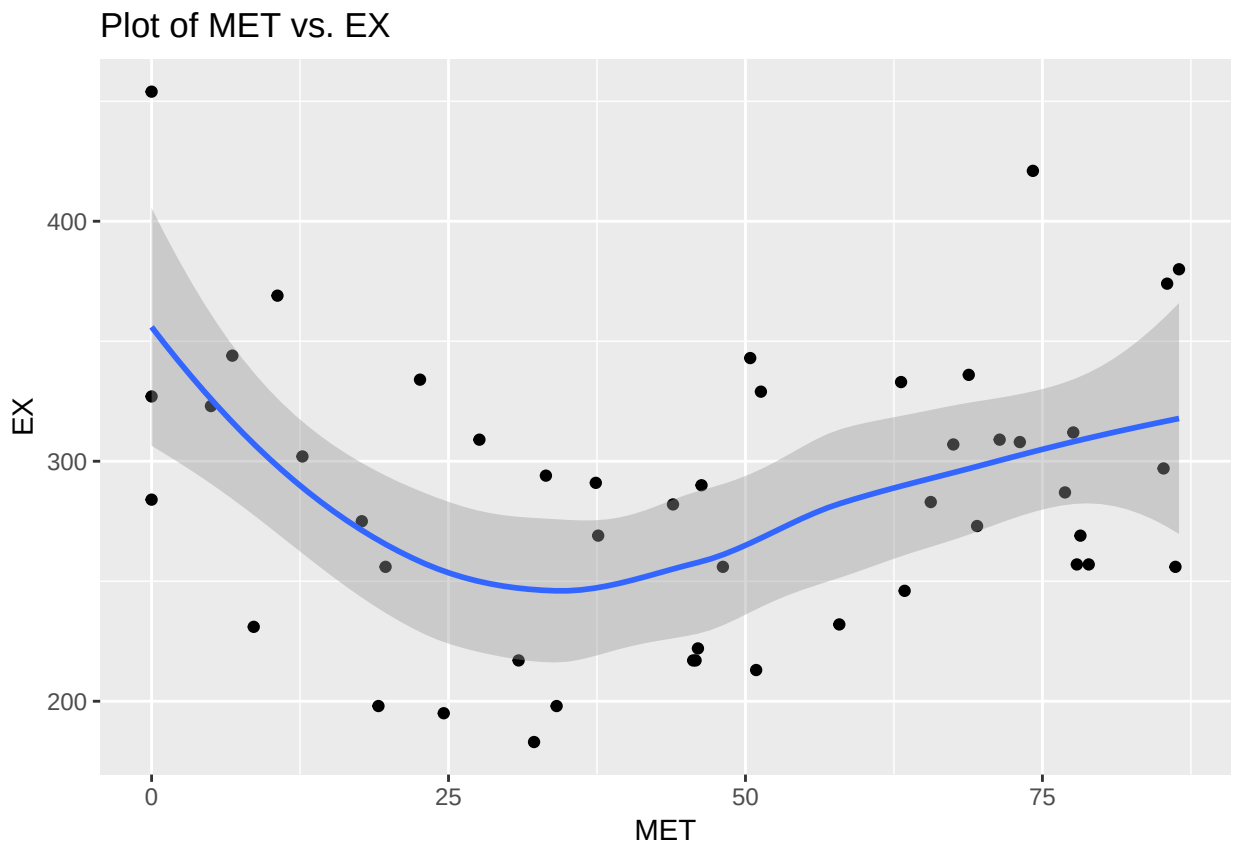
The biggest difference is in recognizing the bad customers, where the accuracy is 93.20% and 93.42% for the train and test cases respectively. This implies our model identifies 'bad' customers with a very high accuracy while it suffers to detect 'good' customers (accuracy of 25% and 29%). This is actually good for business because its many times more important to identify the bad customers instead of good customer(minimize risk).

The change in the accuracy is due to loss matrix where we have ensured that a customer is considered 'good' customer only if the probability of them being a 'good'customer is 10 times greater than them being a 'bad' customer.

Assignment 3

1. Reorder your data with respect to the increase of MET and plot EX versus MET. Discuss what kind of model can be appropriate here. Use the reordered data in steps 2-5.

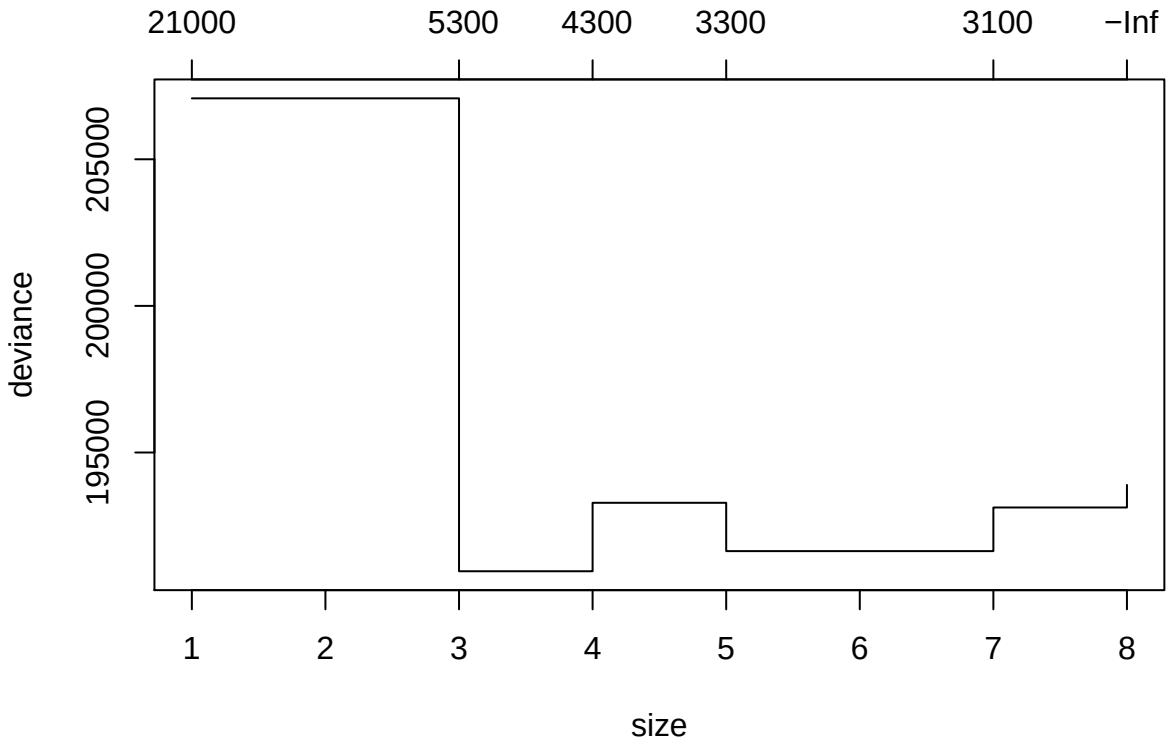
```
rm(list=ls())

set.seed(12345)
state_data <- read.csv2("state.csv")

state_data <- state_data %>% arrange(MET)

ggplot(data = state_data, aes(x=MET, y = EX)) +
  geom_point() +
  geom_smooth(method = 'loess') +
  ggtitle("Plot of MET vs. EX")
```



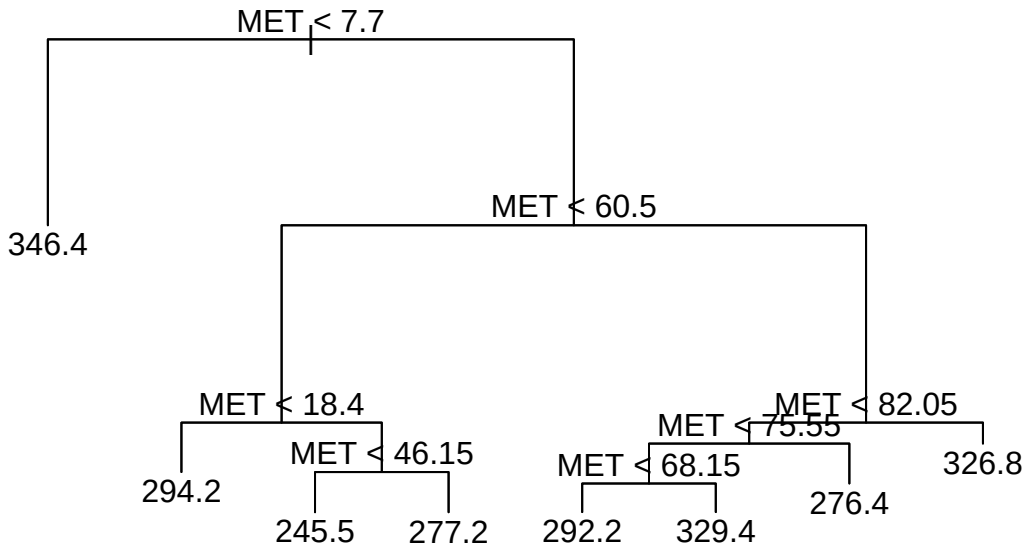
Analysis: As evident from the graph the best model, linear regression will not be a good fit, even the trend is non linear. Piece wise linear model(spline) might be a good one, thus regression per group/cluster will be a good approach.

2. Use package tree and fit a regression tree model with target EX and feature MET in which the number of the leaves is selected by cross-validation, use the entire data set and set minimum number of observations in a leaf equal to 8 (setting minsize in tree.control). Report the selected tree. Plot the original and the fitted data and histogram of residuals. Comment on the distribution of the residuals and the quality of the fit.

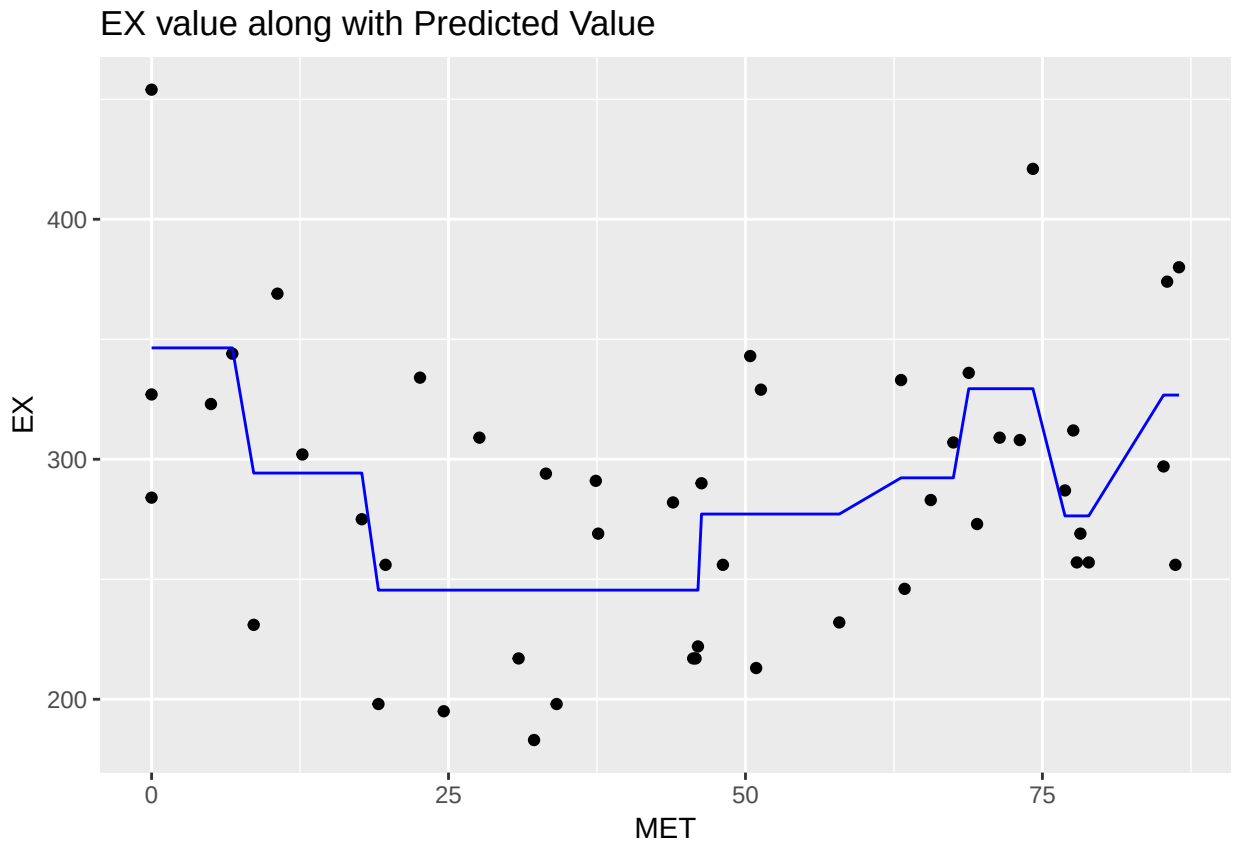
```
set.seed(12345)

state_tree_regression <- tree(data = state_data, EX~MET,
                              control = tree.control(nobs=NROW(state_data),
                                                     minsize = 8))

state_cv_tree <- cv.tree(state_tree_regression, FUN  = prune.tree)
plot(state_cv_tree)
```



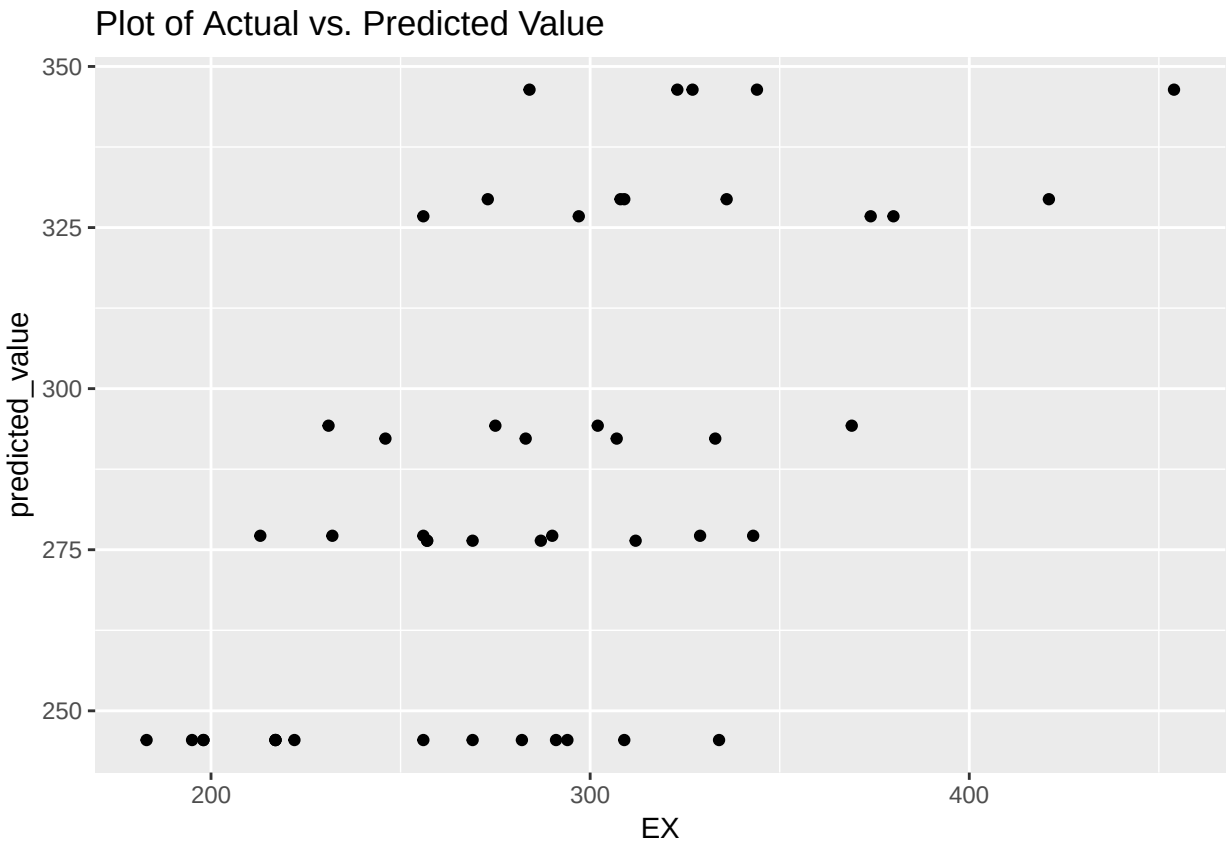

```
# The best size is either 3 or 4

# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, k = 3)
plot(state_cv_tree_purned, main="Pruned Tree for the given dataset")
text(state_cv_tree_purned)
```



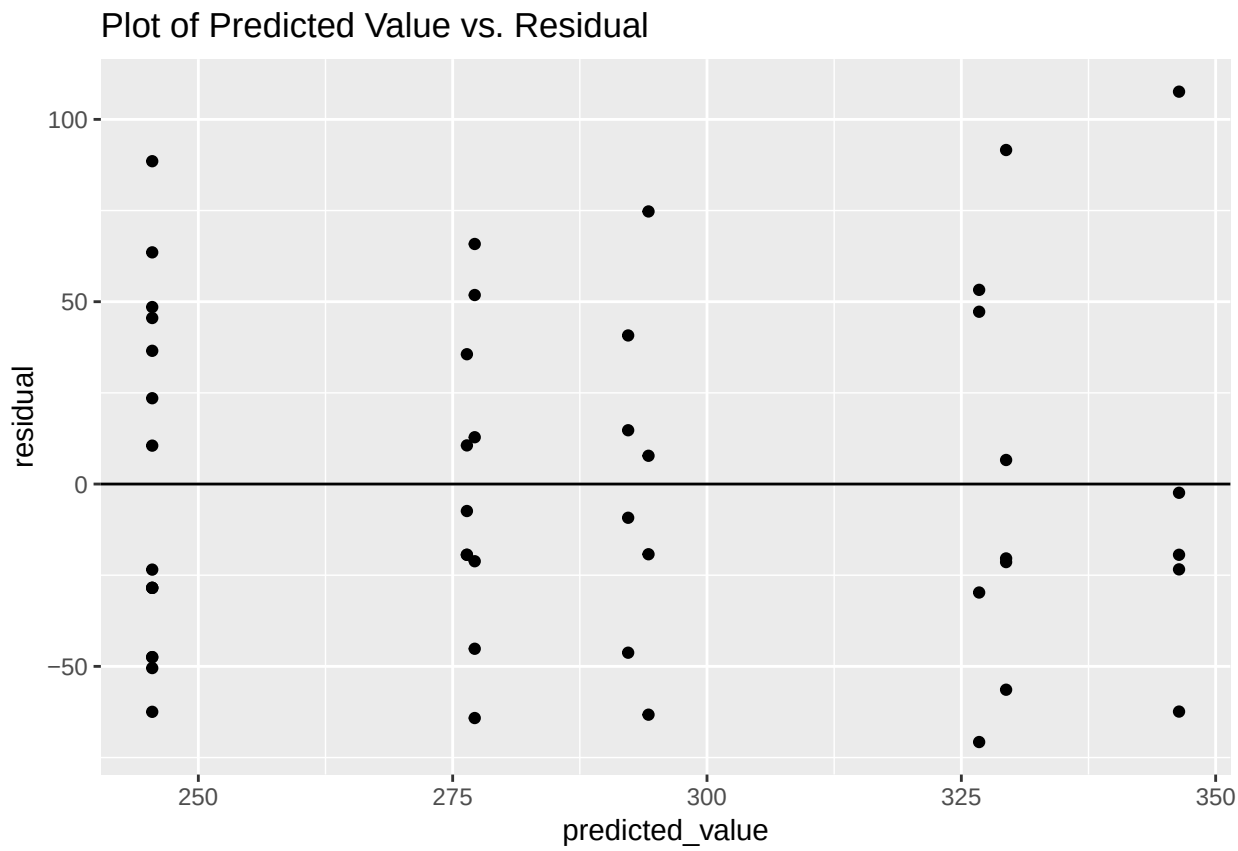

```r
# Original vs. Fitted values
compare_data <- predict(state_cv_tree_purned, newdata = state_data)
compare_data <- cbind(compare_data, state_data$EX, state_data$MET)
compare_data <- as.data.frame(compare_data)
colnames(compare_data) <- c("predicted_value", "EX", "MET")
compare_data$residual <- compare_data$EX -  compare_data$predicted_value

# plots
ggplot(data=compare_data, aes(x = MET, y = EX)) +
  geom_point(aes(x = MET,y=EX)) +
  geom_line(aes(x = MET, y=predicted_value), colour="blue") +
  ggtitle("EX value along with Predicted Value")
```



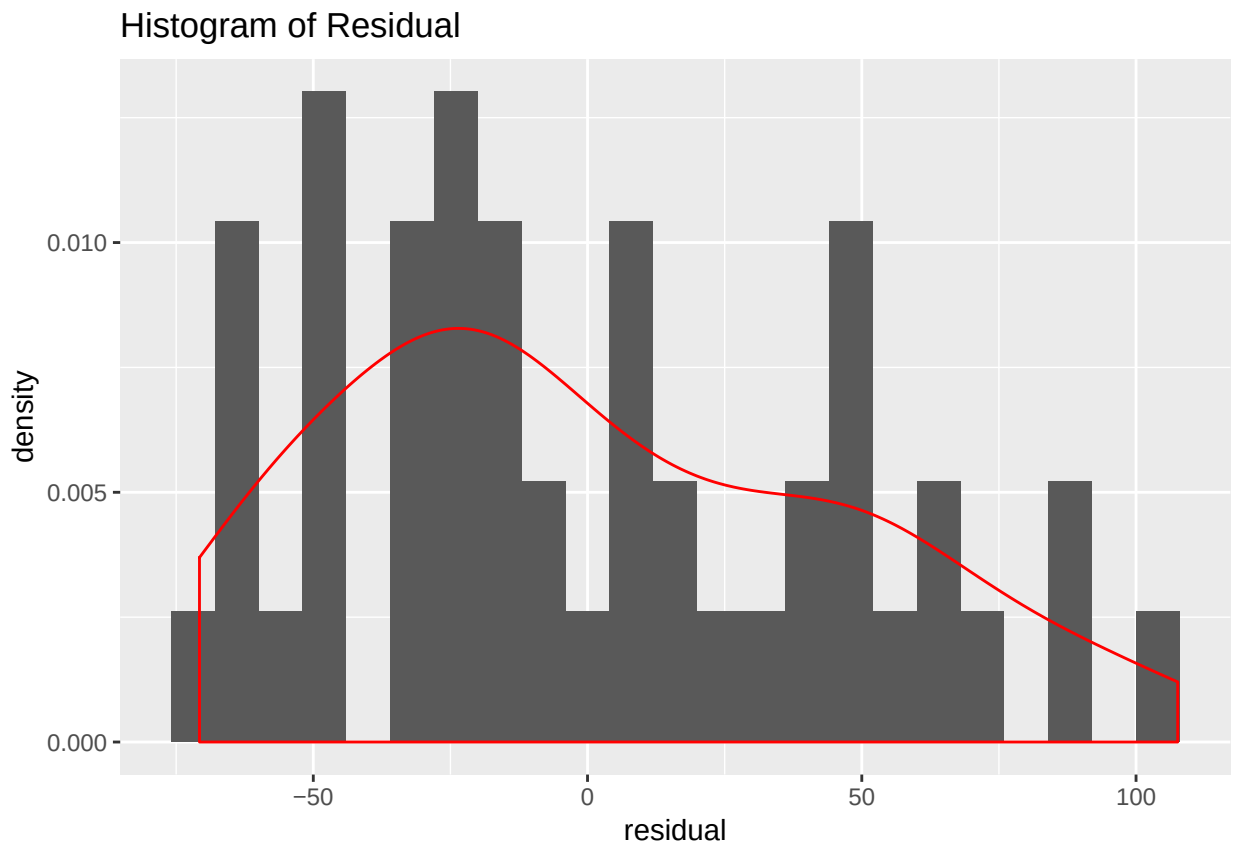

```
ggplot(compare_data, aes(x = EX, y =  predicted_value)) +
  geom_point() +
  ggtitle("Plot of Actual vs. Predicted Value")
```




```
ggplot(compare_data, aes(x = predicted_value, y = residual)) +
  geom_point() + geom_abline(slope=0, intercept=0) +
  ggtitle("Plot of Predicted Value vs. Residual")
```




```
ggplot(data = compare_data, aes(x = residual)) +
  geom_histogram(aes(y = ..density..), binwidth = 8) +
  geom_density(colour = "red") +
  ggtitle("Histogram of Residual")
```



Analysis:

The predicted vs. Actual provides us the insight that for lower values of variable, our model is over predicting (actual value ~200) while the predicted value is ~250. While for larger values (~400) our model is under predicting (~330). At around the mean value (~300) the predicted values are both under and over predicted thus no bias. Thus for values that are away from the mean our model is baised towards over/under predicted while values close to mean our model is not biased.

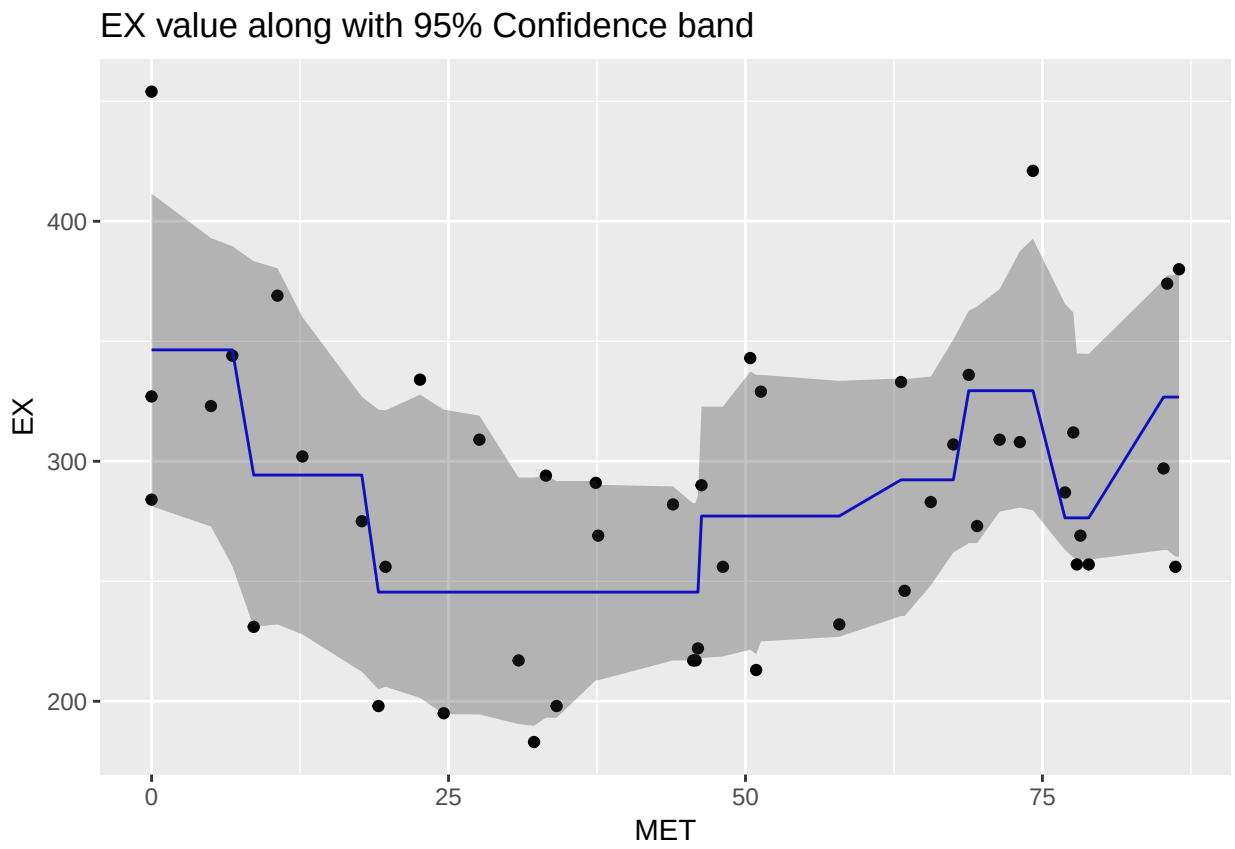
From the plot of Predicted vs. Residual values we can see that error appears random, neither is large bias/concentration of the error towards any value expect at lower/higher values(more points on one side of the line)

From the histogram we can see that the histogram has higher values on the left of zero, and a longer tail on the right. Thus from the above three points we can see that there is scope of improvement in the model especially in the extreme values of the predicted values.

3. Compute and plot the 95% confidence bands for the regression tree model from step 2 (fit a regression tree with the same settings and the same number of leaves as in step 2 to the resampled data) by using a non-parametric bootstrap. Comment whether the band is smooth or bumpy and try to explain why. Consider the width of the confidence band and comment whether results of the regression model in step 2 seem to be reliable.

```
set.seed(12345)
```

```r
# computing bootstrap samples
bootstrap <- function(data, indices){
  data <- state_data[indices,]

  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))

  model_purned <- prune.tree(model, k = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  return(final_fit_boot)
}

res <- boot(state_data, bootstrap, R=1000) #make bootstrap
e <- envelope(res, level = 0.95)
state_tree_regression <- tree(data = state_data, EX~MET,
                              control = tree.control(nobs=NROW(state_data),
                                                     minsize = 8))

# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, k = 3)
predict_for_ci <- predict(state_cv_tree_purned, state_data)
data_for_ci <- cbind(upper_bound = e$point[1,],
                     lower_bound = e$point[2,],
                     EX = state_data$EX,
                     MET = state_data$MET,
                     predicted_value = predict_for_ci) %>% as.data.frame()

#plot cofidence bands

ggplot(data=data_for_ci, aes(x = MET, y = EX)) +
  geom_point(aes(x = MET,y=EX)) +
  geom_line(aes(x = MET, y=predicted_value), colour="blue") +
  geom_ribbon(aes(x = MET, ymin=lower_bound, ymax=upper_bound),alpha = 0.3) +
  ggtitle("EX value along with 95% Confidence band")
```



Analysis:

The confidence bands certainly appear to be bumpy and not smooth, the confidence bands are bumpy because the predicted values shows large flucations. From the width of the confidence band we can assume that our model is not a good one. Ideally we want the confidence band to be narrow thus a wider band suggests we need further tuning to the model.

4.Compute and plot the 95% confidence and prediction bands the regression tree model from step 2 (fit a regression tree with the same settings and the same number of leaves as in step 2 to the resampled data) by using a parametric bootstrap, assume Normal distribution with mean as labels in the tree leaves, while varience is residual varience. Consider the width of the confidence band and comment whether results of the regression model in step 2 seem to be reliable. Does it look like only 5% of data are outside the prediction band? Should it be?

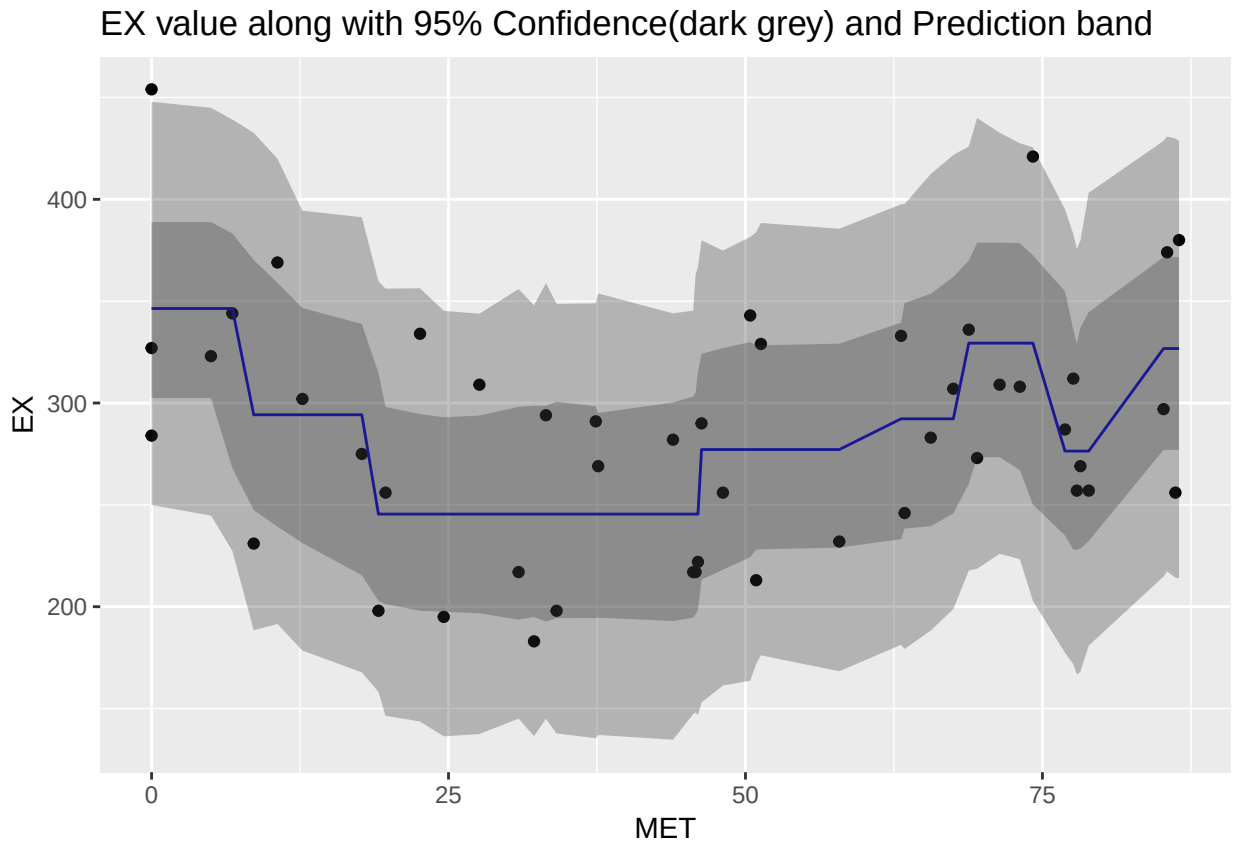
```
set.seed(12345)

mle=prune.tree(state_tree_regression, k = 3)
#data2 = state_data

rng=function(data, mle) {
  data1=data.frame(EX=data$EX, MET=data$MET)
  n=length(data$EX)
```

```
  pred <- predict(mle, newdata = data1)
  residual <- data1$EX - pred
#generate new Price
  data1$EX=rnorm(n, pred, sd(residual))
  return(data1)
}

# computing bootstrap samples
conf.fun <- function(data){
  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))

  model_purned <- prune.tree(model, k = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  return(final_fit_boot)
}


# computing bootstrap samples
pred.fun <- function(data){
  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))

  model_purned <- prune.tree(model, k = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  final_fit <- rnorm(n = length(final_fit_boot),  mean = final_fit_boot, sd=sd(residuals(mle)))
  return(final_fit)
}


conf_para = boot(state_data, statistic=conf.fun, R=1000, mle=mle, ran.gen=rng, sim="parametric")
pred_para = boot(state_data, statistic=pred.fun, R=1000, mle=mle, ran.gen=rng, sim="parametric")

e1 <- envelope(conf_para, level = 0.95)
e2 <- envelope(pred_para, level = 0.95)
```

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

```
# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, k = 3)
predict_for_ci <- predict(state_cv_tree_purned, state_data)


data_for_ci_para <- cbind(upper_bound = e1$point[1,],
                      lower_bound = e1$point[2,],
                      upper_bound_pred = e2$point[1,],
                      lower_bound_pred = e2$point[2,],
                      EX = state_data$EX,
                      MET = state_data$MET,
```

```
                    predicted_value = predict_for_ci) %>% as.data.frame()

ggplot(data=data_for_ci_para, aes(x = MET, y = EX)) +
  geom_point(aes(x = MET,y=EX)) +
  geom_line(aes(x = MET, y=predicted_value), colour="blue") +
  geom_ribbon(aes(x = MET, ymin=lower_bound, ymax=upper_bound),alpha = 0.3) +
  geom_ribbon(aes(x = MET, ymin=lower_bound_pred, ymax=upper_bound_pred), alpha = 0.3) +
  ggtitle("EX value along with 95% Confidence(dark grey) and Prediction band")
```



Analysis:

From the plot we see that the width of the 'confidence band'(darker grey) is narrower compared to non-parametric. We also notice that most of the data is at in the 'confidence band', thus I believe that our regression model does make sense.

We can defintely see that most of the data is under the 'prediction band' more than 95%, thus we can reasonably expect that less than 5% of the data is outside of the 'prediction band'. Based on our assumption of data is normally distributed it does makes sense for prediction band to contain 95% of the data.

5.Consider the histogram of residuals from step 2 and suggest what kind of bootstrap is actually more appropriate here.

```
set.seed(12345)
```

Assignment 4

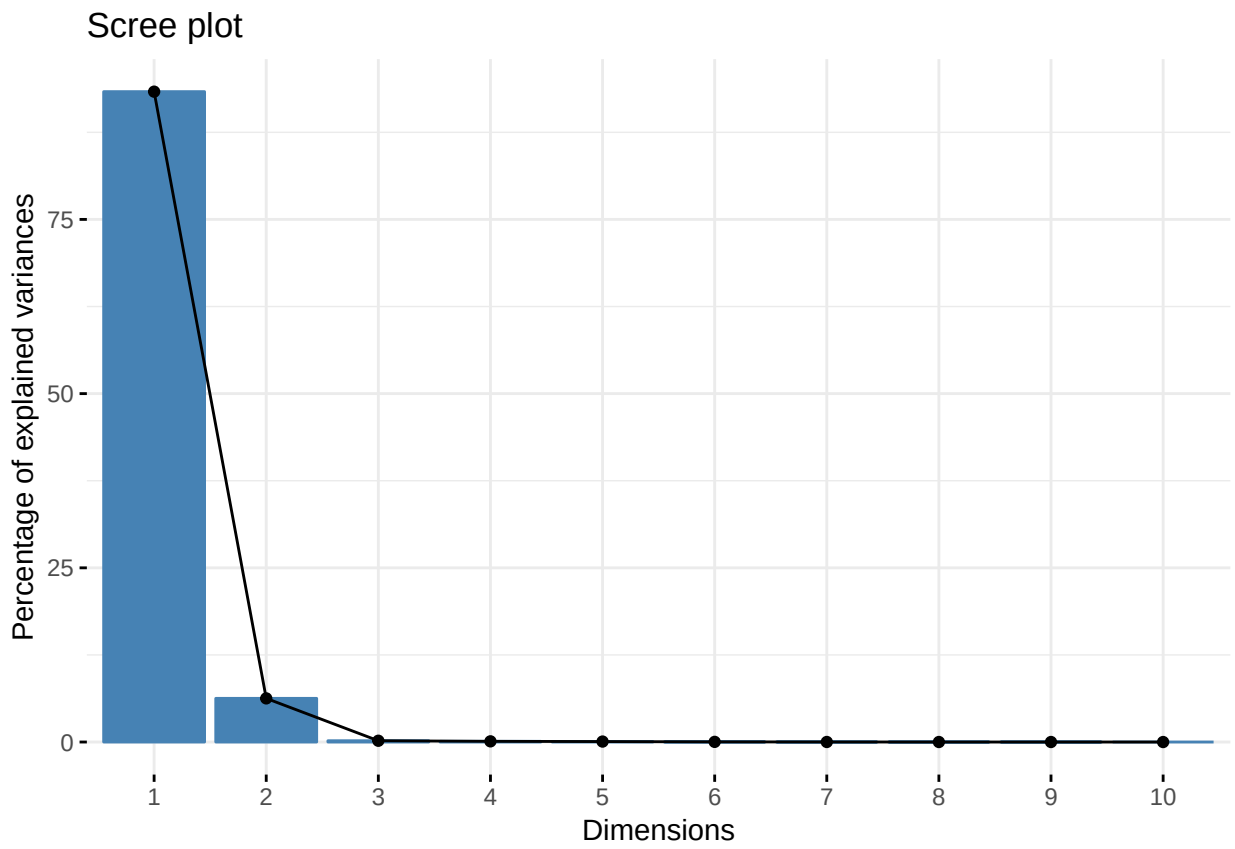
1.Conduct a standard PCA by using the feature space and provide a plot explaining how much variation is explained by each feature. Does the plot show how many PC should be extracted? Select the minimal number of components explaining at least 99% of the total variance. Provide also a plot of the scores in the coordinates (PC1, PC2). Are there unusual diesel fuels according to this plot?

```
rm(list=ls())

set.seed(12345)
NIR_data <- read.csv2("NIRSpectra.csv")

pca_data =  select(NIR_data,-c(Viscosity))
pca_result = prcomp(pca_data)

contribution <- summary(pca_result)$importance
knitr::kable(contribution[,1:5],
             caption = "Contribution of PCA axis towards varience explaination")
```

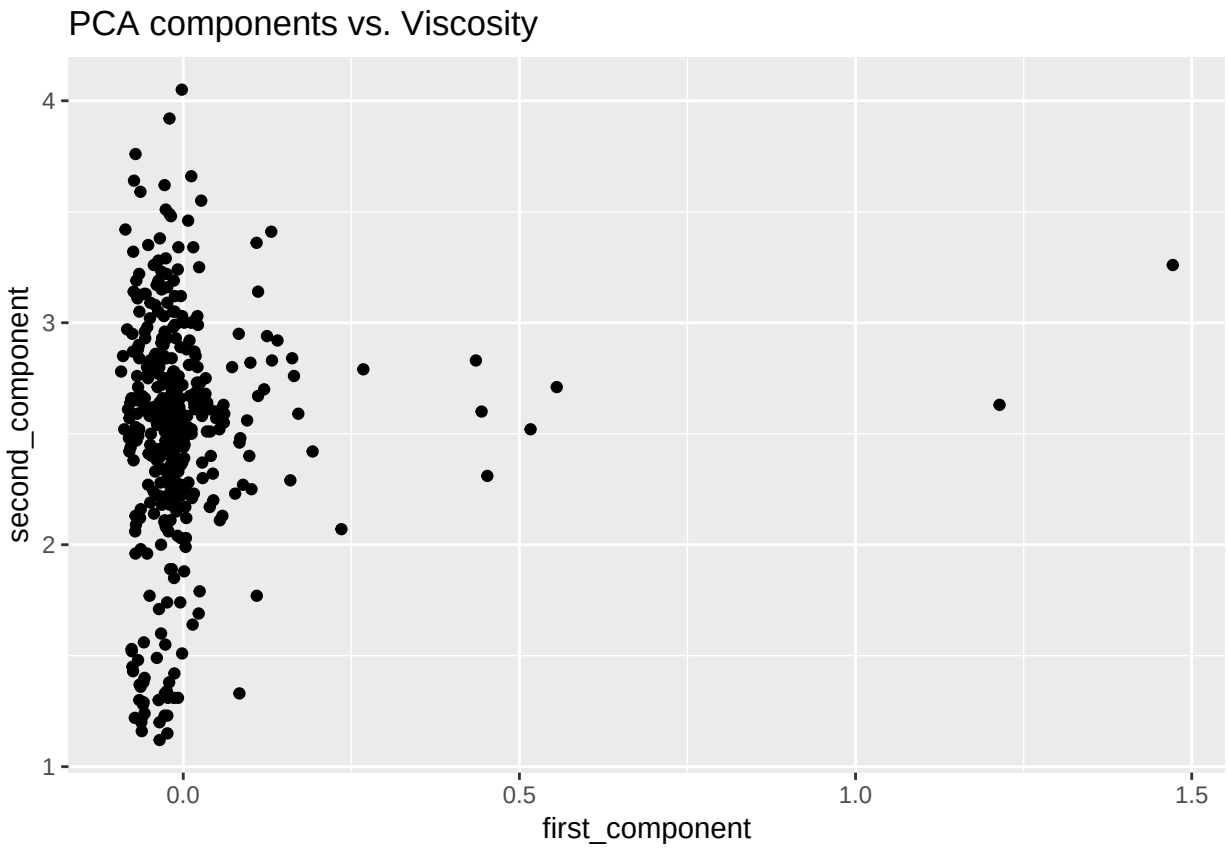
Table 1: Contribution of PCA axis towards varience explaination

	PC1	PC2	PC3	PC4	PC5
Standard deviation	0.122062	0.0316205	0.0054353	0.0040107	0.0033031
Proportion of Variance	0.933320	0.0626300	0.0018500	0.0010100	0.0006800
Cumulative Proportion	0.933320	0.9959600	0.9978100	0.9988200	0.9995000

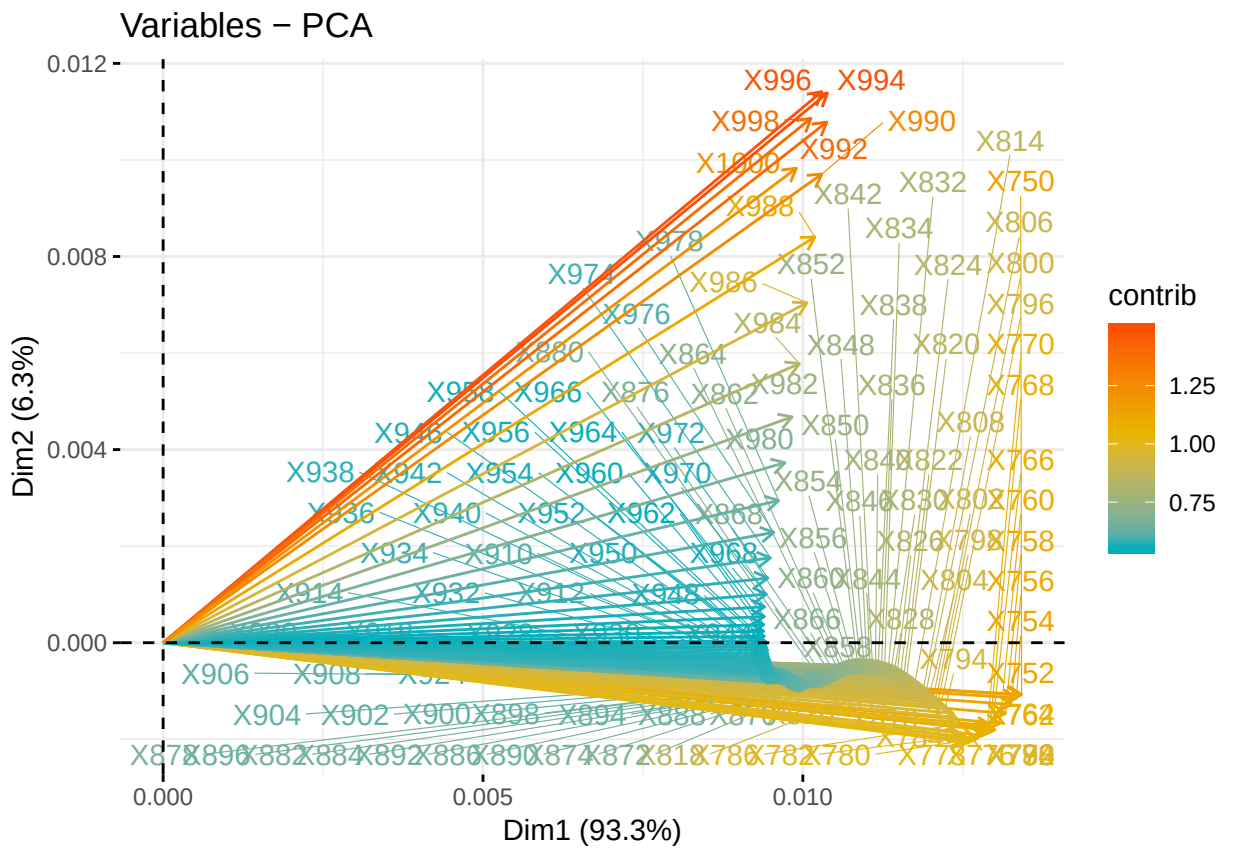
```
# plots PCA components and the eignen vectors
factoextra::fviz_eig(pca_result)
```




```
# pca components and the viscocity
pca_result_data = cbind(first_component = pca_result$x[,1],
                                second_component = pca_result$x[,2],
                                Viscosity = NIR_data$Viscosity) %>% as.data.frame()

# plotting the data variation and the viscocity
ggplot(data = pca_result_data, aes(x = first_component, y = second_component)) +
  geom_point(aes(y = Viscosity)) + ggtitle("PCA components vs. Viscosity")
```




```
# showing the score of PCA component
factoextra::fviz_pca_var(pca_result,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
             )
```



Analysis:

From the plot of PCA component vs. Viscoity, we can see that 2 components are needed (second component of PCA is needed), this is due to the fact the most of the data is vertically spread, removing this dimension would make it impossible to differenitate the types of diesel.

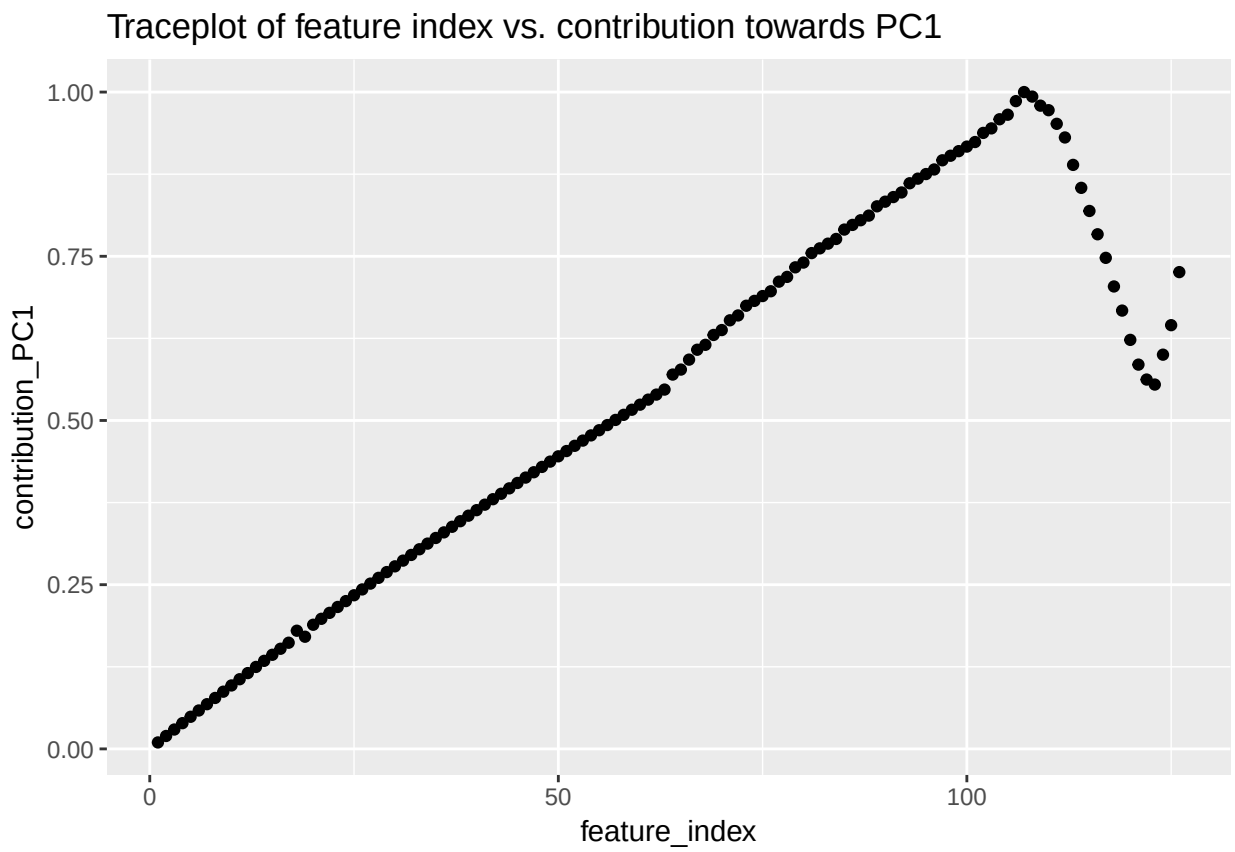
The minimum number of components that account for 99% of the total varience is 2, mainly PC1 and PC2.

From the plot we can also see that there are evidently some diesel that are outliers (diesel with first_component > 0.5 and second component~4).

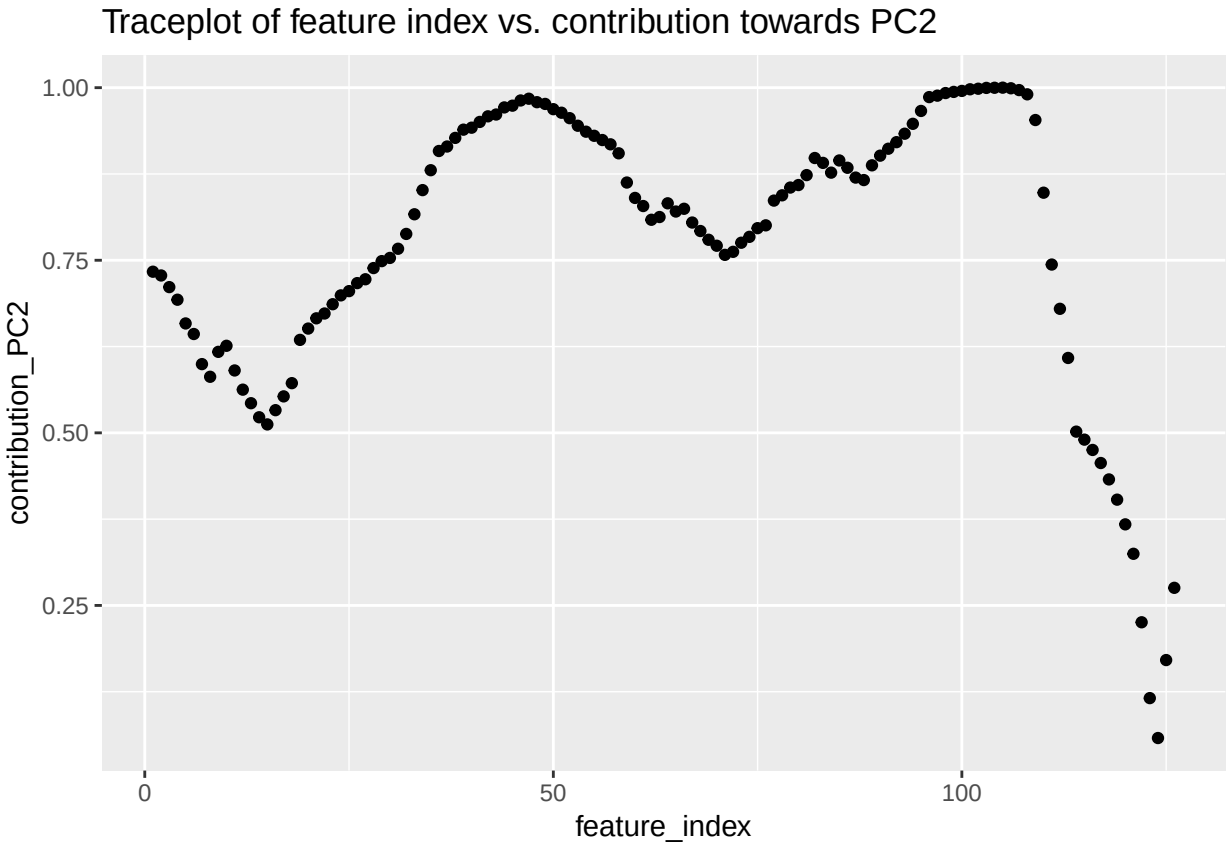
2.Make trace plots of the loadings of the components selected in step 1. Is there any principle component that is explained by mainly a few original features?

```
set.seed(12345)

# creating extra columns
aload <- abs(pca_result$rotation[,1:2])
components <- sweep(aload, 2, colSums(aload), "/")
components <- as.data.frame(components)
components$feature_name <- rownames(components)
components$feature_index <- 1:nrow(components)

components <- components %>% arrange(-PC1)
components$contribution_PC1 <- cumsum(components$PC1)

ggplot(data = components, aes(x = feature_index, y = contribution_PC1)) +
```

```
  geom_point() +
  ggtitle("Traceplot of feature index vs. contribution towards PC1")
```




```
components <- components %>% arrange(-PC2)
components$contribution_PC2 <- cumsum(components$PC2)

ggplot(data = components, aes(x = feature_index, y = contribution_PC2)) +
  geom_point() +
  ggtitle("Traceplot of feature index vs. contribution towards PC2")
```




```
knitr::kable(components[1:10,],
             caption = "Contribution of Features towards the Principle Components")
```

Table 2: Contribution of Features towards the Principle Components

PC1	PC2	feature_name	feature_index	contribution_PC1	contribution_PC2
0.0075609	0.0579423	X996	124	0.6001894	0.0579423
0.0076268	0.0578055	X994	123	0.5546874	0.1157479
0.0074384	0.0551408	X998	125	0.6450879	0.1708887
0.0076192	0.0547439	X992	122	0.5623067	0.2256326
0.0072706	0.0498894	X1000	126	0.7258963	0.2755220
0.0075625	0.0492868	X990	121	0.5850672	0.3248088
0.0074820	0.0426587	X988	120	0.6227157	0.3674674
0.0073922	0.0357468	X986	119	0.6673233	0.4032143
0.0073049	0.0293427	X984	118	0.7040384	0.4325569
0.0072213	0.0237460	X982	117	0.7476261	0.4563030

Analysis:

From the above three plots we see that towards the first principle components axis (93.3% varience accounted) the feature index till 110 are the main contributers(positive contribution), while for the second component(6.3% varience accounted for) we have feature index 25-45 and 85-100 as the main components.The corresponding feature name can be accessed by viewing the table used for plot, a sample of the few columns is shown above.

From the 1st plot is evident that there are no few features which form the core essence of the PCA components, thus the PCA components is made up of many (20+) features atleast and they cannot be limited to few

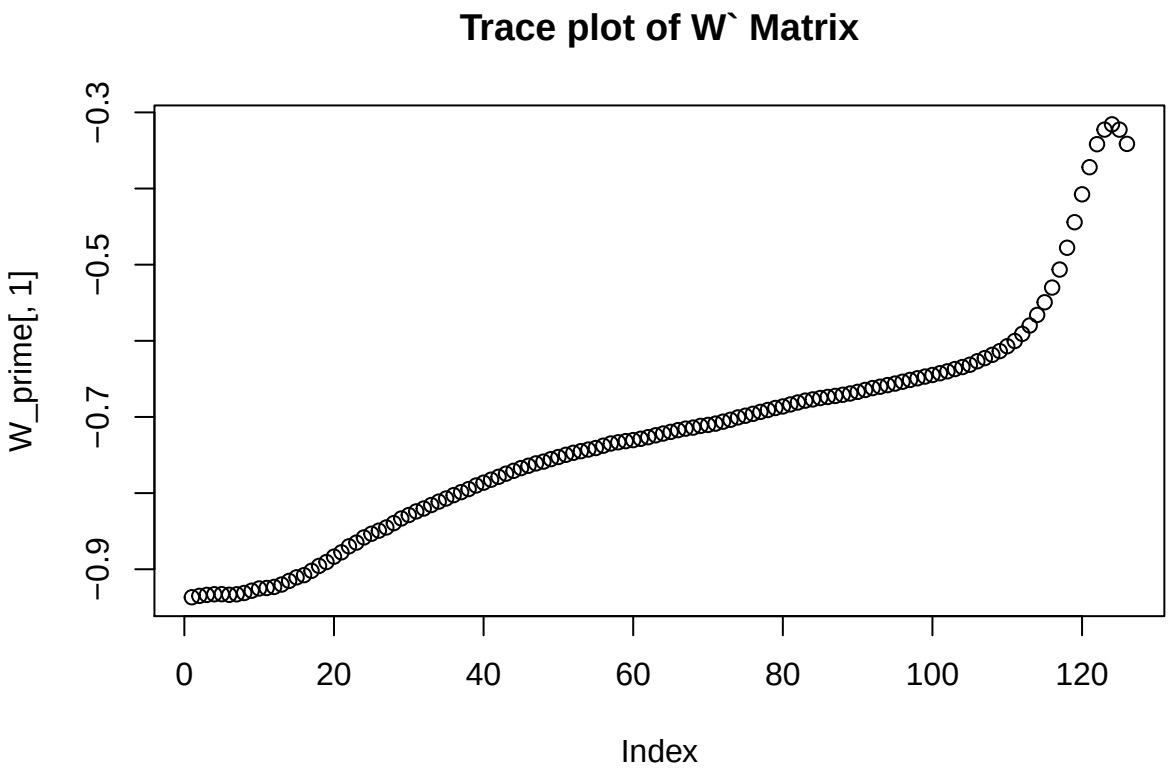
original features.

3. Perform Independent Component Analysis with the number of components selected in step 1 (set seed 12345). Check the documentation for the fastICA method in R and do the following: a.Compute W(prime) = K.W and present the columns of W(prime) in the form of trace plots. Compare with the trace plots in step2 and make conclusion.What kind of measure is represented by the matrix W(prime.), b. Make a plot of the scores of the first two latent features and compare it with the score plot from step 1.

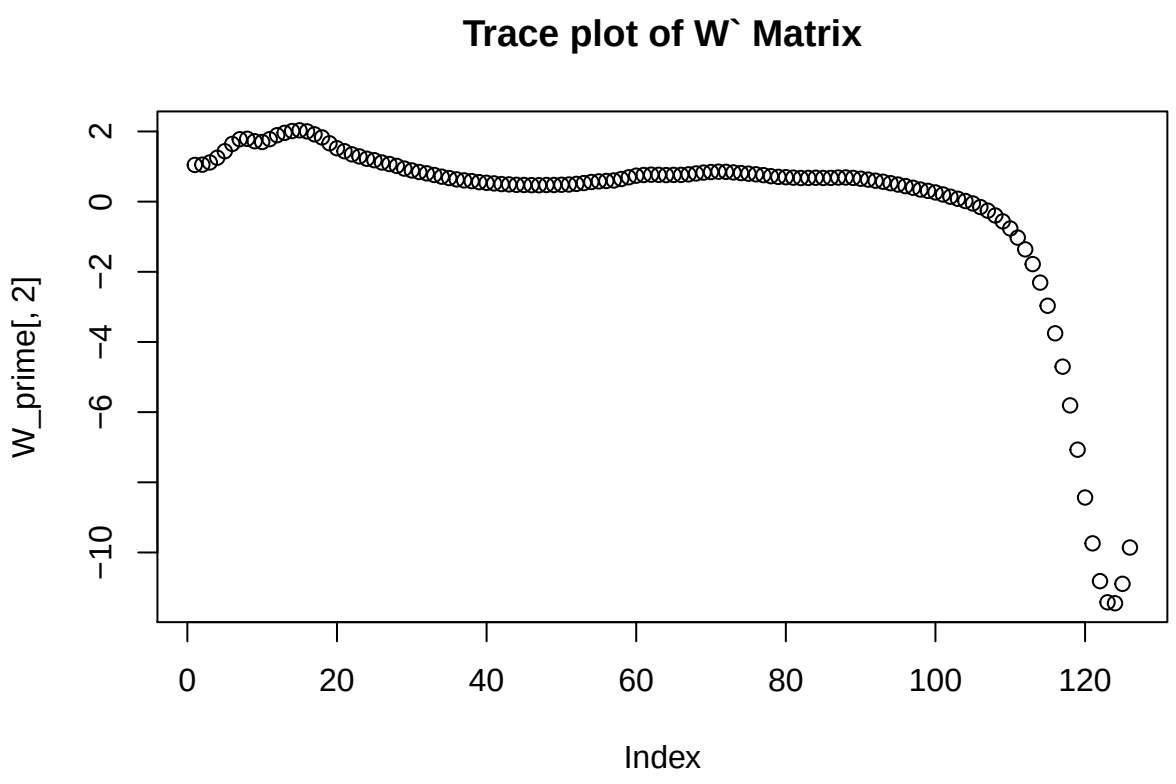
```
set.seed(12345)

# X -> pre-processed data matrix
# K -> pre-whitening matrix that projects data onto the first n.comppprincipal components.
# W -> estimated un-mixing matrix (see definition in details)
# A -> estimated mixing matrix
# S -> estimated source matrix

X <- as.matrix(pca_data)

ICA_extraction <- fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
method = "R", row.norm = FALSE, maxit = 200,
tol = 0.0001, verbose = TRUE)
```

```
## 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 = 0.00003521604
```

```
W_prime <- ICA_extraction$K %*% ICA_extraction$W

#trace plots
plot(W_prime[,1], main = "Trace plot of W` Matrix")
```



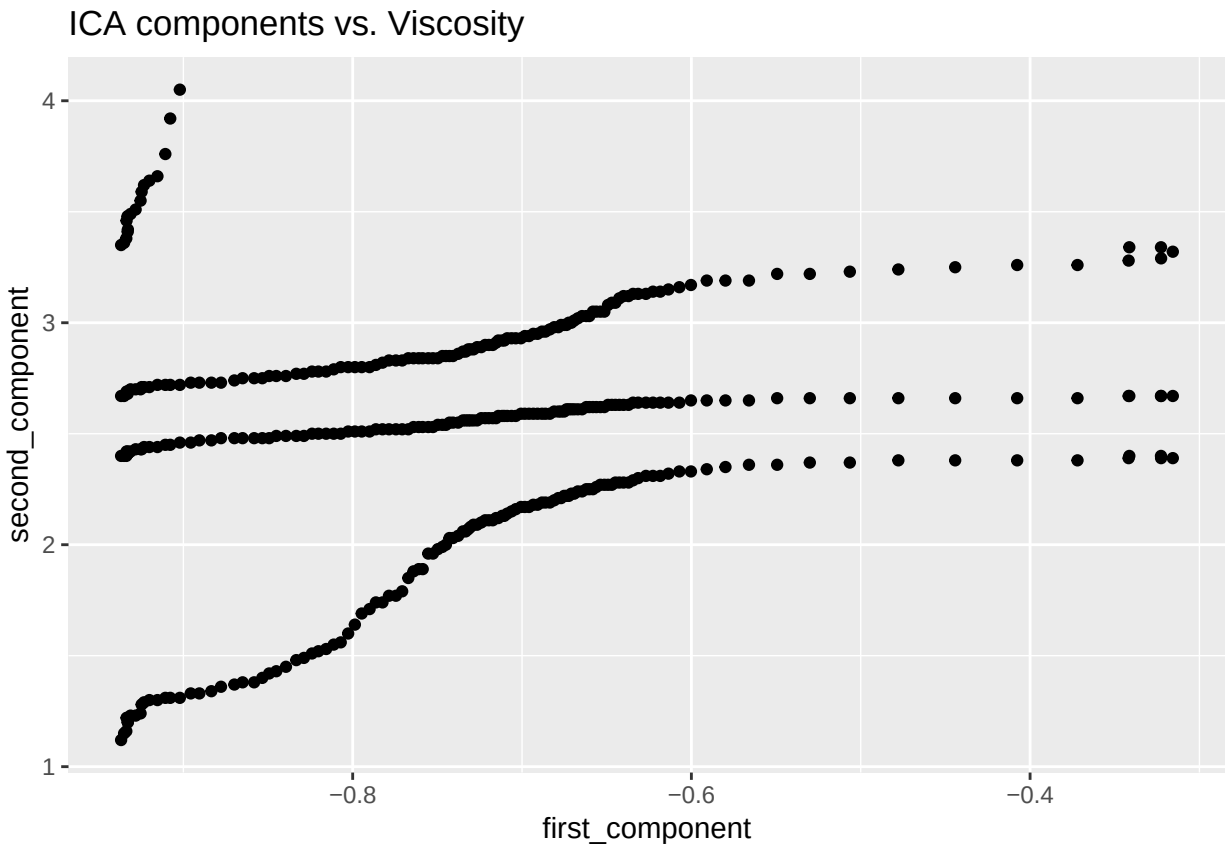

```
#trace plots
plot(W_prime[,2], main = "Trace plot of W` Matrix")
```




```
# pca components and the viscocity
ICA_result_data = cbind(first_component = W_prime[,1],
                                second_component = W_prime[,2],
                                Viscosity = NIR_data$Viscosity) %>% as.data.frame()
```

```
## Warning in cbind(first_component = W_prime[, 1], second_component =
## W_prime[, : number of rows of result is not a multiple of vector length
## (arg 1)
```

```
# plotting the data variation and the viscocity
ggplot(data = ICA_result_data, aes(x = first_component, y = second_component)) +
  geom_point(aes(y = Viscosity)) + ggtitle("ICA components vs. Viscosity")
```



Appendix

```
knitr::opts_chunk$set(echo = TRUE)
if (!require("pacman")) install.packages("pacman")
pacman::p_load(xlsx, ggplot2, MASS, tidyr, dplyr, reshape2, gridExtra,
               tree, caret, e1071, pROC, boot, factoextra, fastICA)

set.seed(12345)
options("jtools-digits" = 2, scipen = 999)

# colours (colour blind friendly)
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

set.seed(12345)
credit_data <- read.xlsx("creditscoring.xls", sheetName = "credit")
credit_data$good_bad <- as.factor(credit_data$good_bad)


n=NROW(credit_data)
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=credit_data[id,]
```

```r
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=credit_data[id2,]

id3=setdiff(id1,id2)
test=credit_data[id3,]
set.seed(12345)

# Create a decision tree model
credit_tree_deviance <- tree(good_bad~., data=train, split = c("deviance"))
credit_tree_gini <- tree(good_bad~., data=train, split = c("gini"))

# Visualize the decision tree with rpart.plot
summary(credit_tree_deviance)
summary(credit_tree_gini)

# predicting on the test dataset to get the misclassification rate.
predict_tree_deviance <- predict(credit_tree_deviance, newdata = test, type = "class")
predict_tree_gini <- predict(credit_tree_gini, newdata = test, type = "class")

conf_tree_deviance <- table(test$good_bad, predict_tree_deviance)
names(dimnames(conf_tree_deviance)) <- c("Actual Test", "P2redicted Test")
caret::confusionMatrix(conf_tree_deviance)

conf_tree_gini <- table(test$good_bad, predict_tree_gini)
names(dimnames(conf_tree_gini)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_tree_gini)
set.seed(12345)

credit_tree <- tree(good_bad~., data=train, split = c("deviance"))

credit_tree_purned_train <- prune.tree(credit_tree, method = c("deviance"))
credit_tree_purned_valid <- prune.tree(credit_tree, newdata = valid ,method = c("deviance"))

result_train <- cbind(credit_tree_purned_train$size,
                      credit_tree_purned_train$dev, "Train")
result_valid <- cbind(credit_tree_purned_valid$size,
                      credit_tree_purned_valid$dev, "Valid")

result <- as.data.frame(rbind(result_valid, result_train))
colnames(result) <- c("Leaf", "Deviance", "Type")

result$Leaf <- as.numeric(as.character(result$Leaf))
result$Deviance <- as.numeric(as.character(result$Deviance))

# plot of deviance vs. number of leafs
ggplot(data = result, aes(x = Leaf, y = Deviance, colour = Type)) +
  geom_point() + geom_line() +
  ggtitle("Plot of Deviance vs. Tree Depth (Shows Deviance least at 4)")

# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=4)
```

```
plot(credit_tree_sniped)
text(credit_tree_sniped)

# misclassification rate for best pruned tree

result_prune_test <- predict(credit_tree_sniped, newdata = test, type = "class")

conf_prune_tree_test <- table(test$good_bad, result_prune_test)
names(dimnames(conf_prune_tree_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_prune_tree_test)
#Fitting the Naive Bayes model
credit_naive_model = naiveBayes(good_bad ~., data=train)
credit_naive_model

#Prediction on the dataset
predict_naive_train = predict(credit_naive_model, newdata=train, type = "class")
predict_naive_test = predict(credit_naive_model, newdata=test, type = "class")

conf_naive_train <- table(train$good_bad, predict_naive_train)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)

conf_naive_test <- table(test$good_bad, predict_naive_test)
names(dimnames(conf_naive_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_naive_test)
set.seed(12345)

credit_tree <- tree(good_bad~., data=train, split = c("deviance"))
credit_naive_model = naiveBayes(good_bad ~., data=train)

# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=7)

# predicting class, getting probability
predict_prune_test_prob <- predict(credit_tree_sniped, newdata = test)
predict_naive_test_prob <- predict(credit_naive_model, newdata=test, type = "raw")

# data mugging
probability_data_naive <- as.data.frame(cbind(predict_naive_test_prob,
                                              as.character(test$good_bad), "naivebayes"))
probability_data_tree <- as.data.frame(cbind(predict_prune_test_prob,
                                             as.character(test$good_bad), "tree"))

probability_data_combined <- rbind(probability_data_tree, probability_data_naive)
colnames(probability_data_combined) <- c("prob_bad", "prob_good",
                                         "actual_test_class", "model")

# final dataset
probability_data_combined$prob_good <- as.numeric(as.character(probability_data_combined$prob_good))

# changing the threshold and printing the probability

tree_list <- NULL
```

```r
naive_list <- NULL
final <- NULL
for(threshold in seq(from = 0.05, to = 0.95, by = 0.05)){
  probability_data_combined$predicted_class <- ifelse(probability_data_combined$prob_good > threshold,

  df2 <- probability_data_combined[,c("model", "actual_test_class", "predicted_class")]
  df2$threshold <- threshold
  df2$match <- ifelse(df2$actual_test_class == df2$predicted_class, 1, 0)

  final <- rbind(df2, final)
}

# Creating the FRP and TRP for each model and threshold
final$temp <- 1
final_summary <- final %>%
group_by(model, threshold) %>%
summarise(total_positive = sum(temp[actual_test_class == "good"]),
          total_negative = sum(temp[actual_test_class == "bad"]),
          correct_positive = sum(temp[actual_test_class == "good" & predicted_class == "good"]),
          false_positive = sum(temp[actual_test_class == "bad" & predicted_class == "good"])) %>%
    mutate(TPR = correct_positive/total_positive, FPR = false_positive/total_negative) %>%
  select(model, threshold, TPR, FPR)

ggplot(data = final_summary, aes(x = FPR, y=TPR)) + geom_line(aes(colour = model)) +
  ggtitle("ROC curve for the Naive Bayes vs. Tree Model")


set.seed(12345)

credit_naive_model = naiveBayes(good_bad ~., data=train)

# predicting class, getting probability
predict_naive_train_prob <- predict(credit_naive_model, newdata=train, type = "raw")
predict_naive_test_prob <- predict(credit_naive_model, newdata=test, type = "raw")

train <- cbind(predict_naive_train_prob, train)
test <- cbind(predict_naive_test_prob, test)

# class based on the loss matrix
train$predicted_class <- ifelse(train$good > 10*train$bad, "good", "bad")
test$predicted_class <- ifelse(test$good > 10*test$bad, "good", "bad")

# confusion matrix
conf_naive_train <- table(train$good_bad, train$predicted_class)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)

conf_naive_test <- table(test$good_bad, test$predicted_class)
names(dimnames(conf_naive_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_naive_test)
rm(list=ls())

set.seed(12345)
```

```r
state_data <- read.csv2("state.csv")

state_data <- state_data %>% arrange(MET)

ggplot(data = state_data, aes(x=MET, y = EX)) +
  geom_point() +
  geom_smooth(method = 'loess') +
  ggtitle("Plot of MET vs. EX")
set.seed(12345)

state_tree_regression <- tree(data = state_data, EX~MET,
                              control = tree.control(nobs=NROW(state_data),
                                                     minsize = 8))

state_cv_tree <- cv.tree(state_tree_regression, FUN  = prune.tree)
plot(state_cv_tree)
# The best size is either 3 or 4

# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, k = 3)
plot(state_cv_tree_purned, main="Pruned Tree for the given dataset")
text(state_cv_tree_purned)

# Original vs. Fitted values
compare_data <- predict(state_cv_tree_purned, newdata = state_data)
compare_data <- cbind(compare_data, state_data$EX, state_data$MET)
compare_data <- as.data.frame(compare_data)
colnames(compare_data) <- c("predicted_value", "EX", "MET")
compare_data$residual <- compare_data$EX -  compare_data$predicted_value

# plots
ggplot(data=compare_data, aes(x = MET, y = EX)) +
  geom_point(aes(x = MET,y=EX)) +
  geom_line(aes(x = MET, y=predicted_value), colour="blue") +
  ggtitle("EX value along with Predicted Value")

ggplot(compare_data, aes(x = EX, y =  predicted_value)) +
  geom_point() +
  ggtitle("Plot of Actual vs. Predicted Value")

ggplot(compare_data, aes(x = predicted_value, y = residual)) +
  geom_point() + geom_abline(slope=0, intercept=0) +
  ggtitle("Plot of Predicted Value vs. Residual")

ggplot(data = compare_data, aes(x = residual)) +
  geom_histogram(aes(y = ..density..), binwidth = 8) +
  geom_density(colour = "red") +
  ggtitle("Histogram of Residual")
set.seed(12345)

# computing bootstrap samples
bootstrap <- function(data, indices){
  data <- state_data[indices,]
```

```
  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))

  model_purned <- prune.tree(model, k = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  return(final_fit_boot)
}

res <- boot(state_data, bootstrap, R=1000) #make bootstrap
e <- envelope(res, level = 0.95)
state_tree_regression <- tree(data = state_data, EX~MET,
                              control = tree.control(nobs=NROW(state_data),
                                                     minsize = 8))

# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, k = 3)
predict_for_ci <- predict(state_cv_tree_purned, state_data)
data_for_ci <- cbind(upper_bound = e$point[1,],
                     lower_bound = e$point[2,],
                     EX = state_data$EX,
                     MET = state_data$MET,
                     predicted_value = predict_for_ci) %>% as.data.frame()

#plot cofidence bands

ggplot(data=data_for_ci, aes(x = MET, y = EX)) +
  geom_point(aes(x = MET,y=EX)) +
  geom_line(aes(x = MET, y=predicted_value), colour="blue") +
  geom_ribbon(aes(x = MET, ymin=lower_bound, ymax=upper_bound),alpha = 0.3) +
  ggtitle("EX value along with 95% Confidence band")
set.seed(12345)

mle=prune.tree(state_tree_regression, k = 3)
#data2 = state_data

rng=function(data, mle) {
  data1=data.frame(EX=data$EX, MET=data$MET)
  n=length(data$EX)
  pred <- predict(mle, newdata = data1)
  residual <- data1$EX - pred
#generate new Price
  data1$EX=rnorm(n, pred, sd(residual))
  return(data1)
}

# computing bootstrap samples
conf.fun <- function(data){
  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))
```

```
  model_purned <- prune.tree(model, k = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  return(final_fit_boot)
}


# computing bootstrap samples
pred.fun <- function(data){
  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))

  model_purned <- prune.tree(model, k = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  final_fit <- rnorm(n = length(final_fit_boot),  mean = final_fit_boot, sd=sd(residuals(mle)))
  return(final_fit)
}


conf_para = boot(state_data, statistic=conf.fun, R=1000, mle=mle, ran.gen=rng, sim="parametric")
pred_para = boot(state_data, statistic=pred.fun, R=1000, mle=mle, ran.gen=rng, sim="parametric")

e1 <- envelope(conf_para, level = 0.95)
e2 <- envelope(pred_para, level = 0.95)


# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, k = 3)
predict_for_ci <- predict(state_cv_tree_purned, state_data)


data_for_ci_para <- cbind(upper_bound = e1$point[1,],
                     lower_bound = e1$point[2,],
                     upper_bound_pred = e2$point[1,],
                     lower_bound_pred = e2$point[2,],
                     EX = state_data$EX,
                     MET = state_data$MET,
                     predicted_value = predict_for_ci) %>% as.data.frame()

ggplot(data=data_for_ci_para, aes(x = MET, y = EX)) +
  geom_point(aes(x = MET,y=EX)) +
  geom_line(aes(x = MET, y=predicted_value), colour="blue") +
  geom_ribbon(aes(x = MET, ymin=lower_bound, ymax=upper_bound),alpha = 0.3) +
  geom_ribbon(aes(x = MET, ymin=lower_bound_pred, ymax=upper_bound_pred), alpha = 0.3) +
  ggtitle("EX value along with 95% Confidence(dark grey) and Prediction band")
set.seed(12345)

rm(list=ls())

set.seed(12345)
NIR_data <- read.csv2("NIRSpectra.csv")
```

```r
pca_data =  select(NIR_data,-c(Viscosity))
pca_result = prcomp(pca_data)

contribution <- summary(pca_result)$importance
knitr::kable(contribution[,1:5],
             caption = "Contribution of PCA axis towards varience explaination")

# plots PCA components and the eignen vectors
factoextra::fviz_eig(pca_result)

# pca components and the viscocity
pca_result_data = cbind(first_component = pca_result$x[,1],
                                second_component = pca_result$x[,2],
                                Viscosity = NIR_data$Viscosity) %>% as.data.frame()

# plotting the data variation and the viscocity
ggplot(data = pca_result_data, aes(x = first_component, y = second_component)) +
  geom_point(aes(y = Viscosity)) + ggtitle("PCA components vs. Viscosity")


# showing the score of PCA component
factoextra::fviz_pca_var(pca_result,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
             )

set.seed(12345)

# creating extra columns
aload <- abs(pca_result$rotation[,1:2])
components <- sweep(aload, 2, colSums(aload), "/")
components <- as.data.frame(components)
components$feature_name <- rownames(components)
components$feature_index <- 1:nrow(components)

components <- components %>% arrange(-PC1)
components$contribution_PC1 <- cumsum(components$PC1)

ggplot(data = components, aes(x = feature_index, y = contribution_PC1)) +
  geom_point() +
  ggtitle("Traceplot of feature index vs. contribution towards PC1")

components <- components %>% arrange(-PC2)
components$contribution_PC2 <- cumsum(components$PC2)

ggplot(data = components, aes(x = feature_index, y = contribution_PC2)) +
  geom_point() +
  ggtitle("Traceplot of feature index vs. contribution towards PC2")


knitr::kable(components[1:10,],
             caption = "Contribution of Features towards the Principle Components")
```

```r
set.seed(12345)

# X -> pre-processed data matrix
# K -> pre-whitening matrix that projects data onto the first n.compprincipal components.
# W -> estimated un-mixing matrix (see definition in details)
# A -> estimated mixing matrix
# S -> estimated source matrix

X <- as.matrix(pca_data)

ICA_extraction <- fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
method = "R", row.norm = FALSE, maxit = 200,
tol = 0.0001, verbose = TRUE)

W_prime <- ICA_extraction$K %*% ICA_extraction$W

#trace plots
plot(W_prime[,1], main = "Trace plot of W` Matrix")
#trace plots
plot(W_prime[,2], main = "Trace plot of W` Matrix")

# pca components and the viscocity
ICA_result_data = cbind(first_component = W_prime[,1],
                                second_component = W_prime[,2],
                                Viscosity = NIR_data$Viscosity) %>% as.data.frame()

# plotting the data variation and the viscocity
ggplot(data = ICA_result_data, aes(x = first_component, y = second_component)) +
  geom_point(aes(y = Viscosity)) + ggtitle("ICA components vs. Viscosity")
```