Northeastern University

Case Report on
Applied Science in Supply Chain

IE7275 Data Mining in Engineering
Sec 07 Fall 2019

Student : Shivahari Revathi Venkateswaran: venkateswaran.sh@husky.neu.edu

TABLE OF CONTENTS

,

1. INTRODUCTION

supply chain is the network of all the individuals, organizations, resources, activities and technology involved in the creation and sale of a product, from the delivery of raw materials from the supplier to manufacturing the products and delivering to the end user. In today's world, supply chain plays a vital role in the success of an organization.

Since supply chain is booming, it has become imperative for all companies to have efficient end to end supply chain that helps to focus on optimization. A recent study shows that 85% of companies focus more on their supply chain to understand what products the customers want and at the same time how they can internally improve their operations for critical decision making for company profitability.

But, dealing with millions of products and customers is not an easy task. Companies are facing difficulties to identify the right commodities for the right customers with right supplier combination that are of more importance. This has become a common challenge for supply chain companies so as to put more efforts to find opportuinities for improvement.

In order to improve supply chain efficiencies, handling large sets of data, data mining techniques can be used to perform segmentation where commodities are classified into three parts based on their trade value. With the help of machine learning, companies can concentrate on most important products with high service level, furthermore, segmentation helps to predict heavyweight commodities in each of the three segments to accordingly plan for future transportation needs. Accurately predicting the product categories using data mining techniques provide insights to know about customer preferences and product weightage which is the essence of supply chain management.

2. PROBLEM STATEMENT

The problem is to improve supply chain optimization by analyzing the data to perform segmentation to predict what are the commodities that fall under high trade value so that weightage of commodities mostly preferred by customers could be determined using data mining tools.
In reality…

- Firms operates multiple supply chain
- Firms segments in order to match the right methods to the right product/customer/supplier combination
- Firms can segment products, customers, suppliers etc.

Segmentation only makes sense if you do something different in how you buy, make, move, store or sell!

In the “Global Commodity Trade Statistics”, this project motivates how to segment the commodities into three different segments based on the trade values of those commodities. This would greatly help the firms to concentrate more on Segment A and B, as most of the trade values relies on these segments. Based on the Pareto principle (80-20 rule), 20 percent of the commodities will cover 80 percent of the trade values. Thus, the segmentation concept greatly helps us to improve the supply chain process by concentrating on the most important commodities with high service level.

3. SOLUTION DESIGN AND DATA COLLECTION

To perform segmentation to identify the commodities so that they can be much focused because of their trade value, the following operations are performed :

Explore, clean and pre-process the data to ensure the data at hand is complete and relevant.

- Conduct data reduction to reduce the impact of outliers on our model.
- Finding a Calculated column called “segments” that is used as a label for the Classification technique.
- Use data visualization techniques to have a better understanding of the spread of data and the correlation between variables.
- Determine the appropriate data mining techniques to be used.
- Partition the data into two sections, a training set and a validation set to conduct supervised learning and calibrate it in the later steps.
- Use the algorithm to perform the predictions and interpret the results.

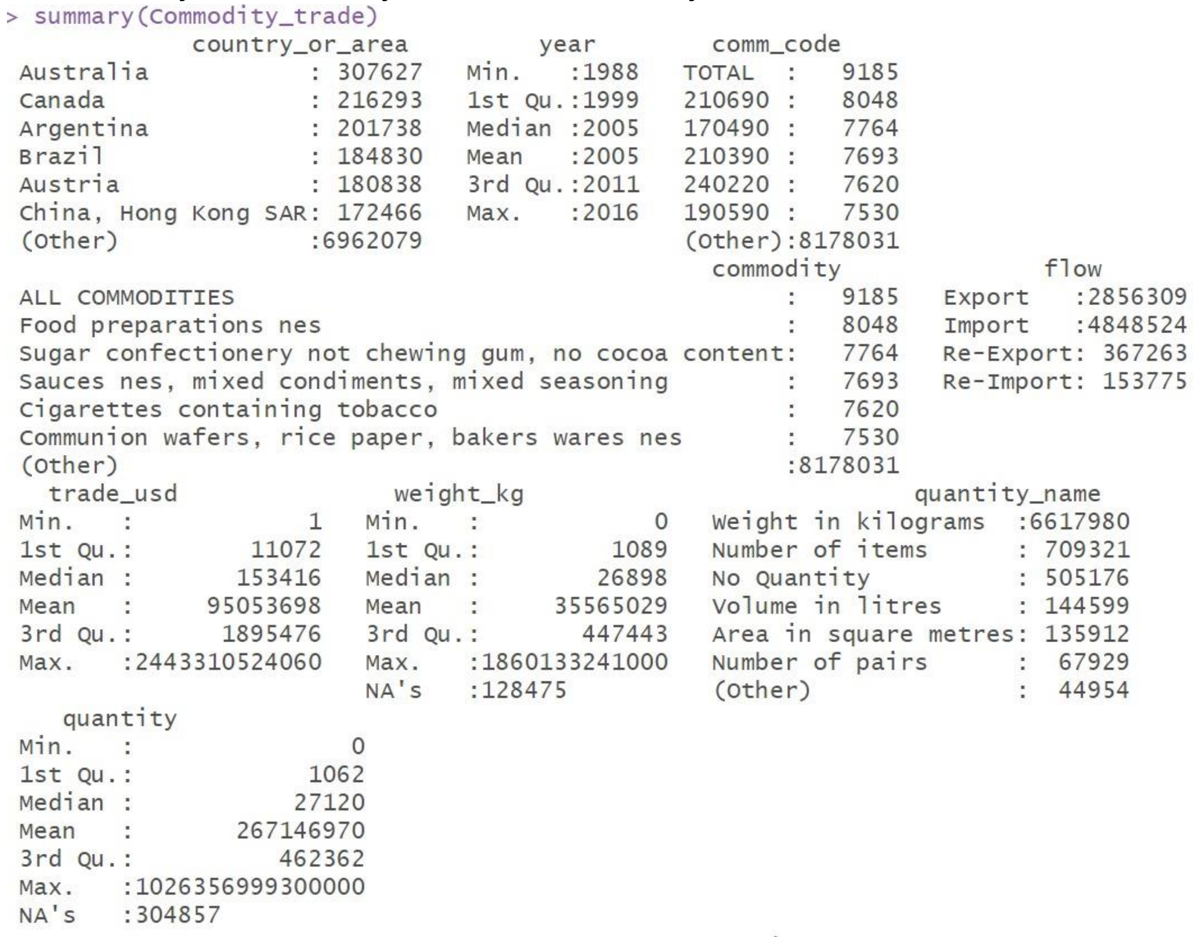
The dataset structure is as shown below. For this project we collected the dataset, “Global Commodity Trade Statistics” from Kaggle.com. The dataset has 10 variables and 825871 records.

No.	Variable Name	Description	Data Type
1	country_or_area	Country name of record	String
2	year	Year in which the trade has taken place	Numeric
3	comm_code	Unique code for each of the Commodities	Numeric
4	commodity	Description of the Commodities	String
5	flow	Flow of trade. E.g. Import and Export	String
6	trade_usd	Value of trade in USD	Numeric
7	weight_kg	Weight of commodity in kg.	Numeric

8	quantity name	Description of quantity measurement	String
9	quantity	Count of quantity of given items based on quantity name	Numeric
10	category	Category to identify commodity	String

4. DATA VISUALIZATION/PROCESSING

Firstly, we understand our data and its structure. We create a data frame – commodity_trade. Then we analyze the summary statistics of commodity_trade.

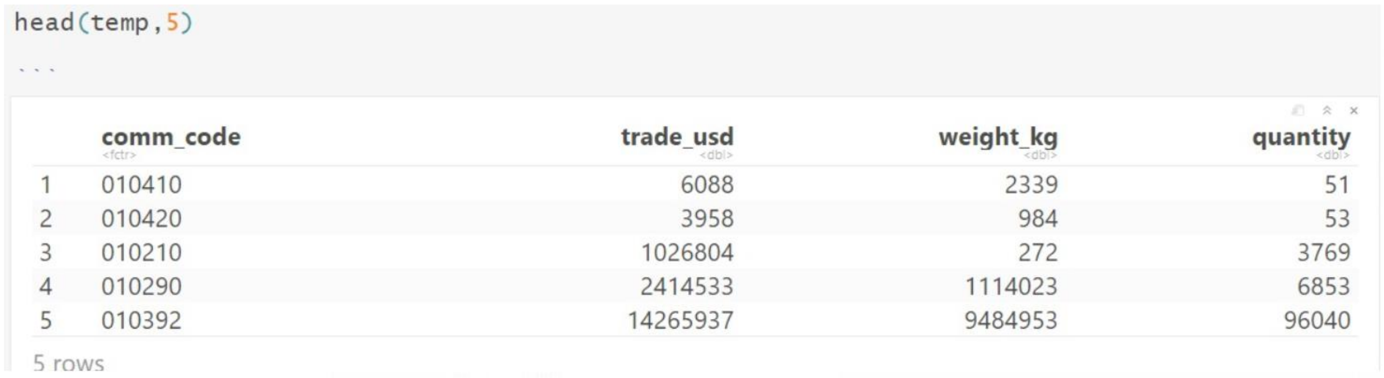


```
> summary(Commodity_trade)
             country_or_area        year          comm_code
 Australia           : 307627   Min.   :1988   TOTAL   :   9185
 Canada              : 216293   1st Qu.:1999   210690 :   8048
 Argentina           : 201738   Median :2005   170490 :   7764
 Brazil              : 184830   Mean   :2005   210390 :   7693
 Austria             : 180838   3rd Qu.:2011   240220 :   7620
 China, Hong Kong SAR: 172466   Max.   :2016   190590 :   7530
 (Other)             :6962079                  (Other):8178031
                                                 commodity             flow
 ALL COMMODITIES                                     :   9185   Export   :2856309
 Food preparations nes                               :   8048   Import   :4848524
 Sugar confectionery not chewing gum, no cocoa content:   7764   Re-Export: 367263
 Sauces nes, mixed condiments, mixed seasoning       :   7693   Re-Import: 153775
 Cigarettes containing tobacco                       :   7620
 Communion wafers, rice paper, bakers wares nes      :   7530
 (Other)                                             :8178031
   trade_usd                weight_kg                       quantity_name
 Min.   :             1   Min.   :             0   Weight in kilograms   :6617980
 1st Qu.:         11072   1st Qu.:          1089   Number of items       : 709321
 Median :        153416   Median :         26898   No Quantity           : 505176
 Mean   :      95053698   Mean   :      35565029   Volume in litres      : 144599
 3rd Qu.:       1895476   3rd Qu.:        447443   Area in square metres: 135912
 Max.   :2443310524060   Max.   :1860133241000   Number of pairs       :  67929
                          NA's   :128475           (Other)               :  44954
    quantity
 Min.   :              0
 1st Qu.:           1062
 Median :          27120
 Mean   :      267146970
 3rd Qu.:         462362
 Max.   :1026356999300000
 NA's   :304857
```

As per the summary statistics we can clearly see that the dataset consists of many null values which needs to be processed before building the models. It can be observed that there 5 numerical and 5 string variables. In order to apply the required data mining techniques, we need to select predictor variables that must be numerical and response variable as categorical. we also need to perform feature selection for classifying the segments.

4.1 Feature Selection

In order to perform segmentation, we need to select the features that can be used for classifying the commodities that fall into segment A,B and C. As per inventory management method followed in supply chain, we need the trade value, quantity and weight for classifying the products into three segments. Thus, we select only commodity trade_usd, weight_kg and quantity.



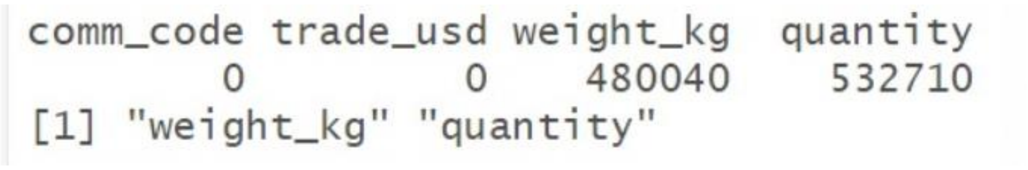
```
head(temp,5)
```

	comm_code <fctr>	trade_usd <dbl>	weight_kg <dbl>	quantity <dbl>
1	010410	6088	2339	51
2	010420	3958	984	53
3	010210	1026804	272	3769
4	010290	2414533	1114023	6853
5	010392	14265937	9484953	96040

5 rows

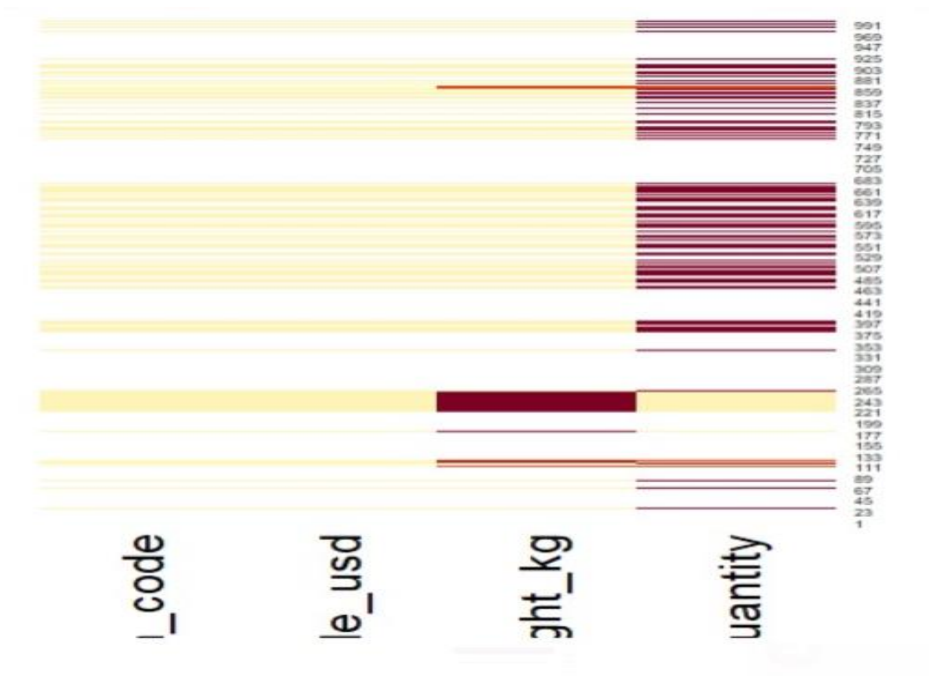
4.2 Handling the Missing Values

We need to first identify the missing values from the predictor variables in order to check if this will significantly affect our analysis. Below figure shows the count of missing values:



```
comm_code trade_usd weight_kg  quantity
        0         0    480040    532710
[1] "weight_kg" "quantity"
```

To check whether the number of missing values in weight_kg and quantity column, we can use heat map to see how these missing values significantly affect our analysis. Below is the heat map



From the heat map, we can infer that there's a high impact from the missing values. Thus, we cannot simply remove the missing values, instead we can replace the missing values with the mean value of that particular variable.

4.3 Data Manipulation

We still do not have the response/dependent variable (Y). for this we need to calculate the weighted average of trade_value, replace_mean_weight_kg and replace_mean_quantity. By using the formula:

Weighted_avg = trade_usd/(replace_mean_weight_kg)*(replace_mean_quantity)

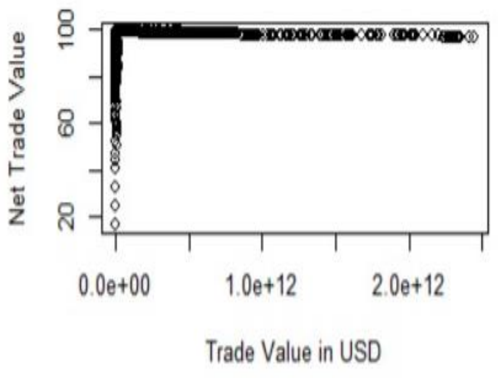
After calculating the weighted average for each of the rows in the dataset, we need to arrange the weighted_avg column in descending order and then find the cumulative sum for the same. After this we find the net value for each of the rows by dividing their corresponding cumulative value by overall value of the weighted_avg column and store this in a separate column called Net_value.

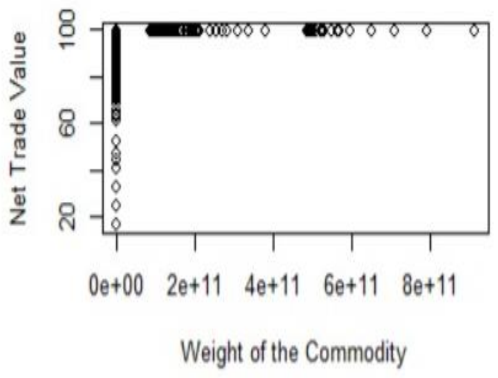
4.4 Creating the Y Label

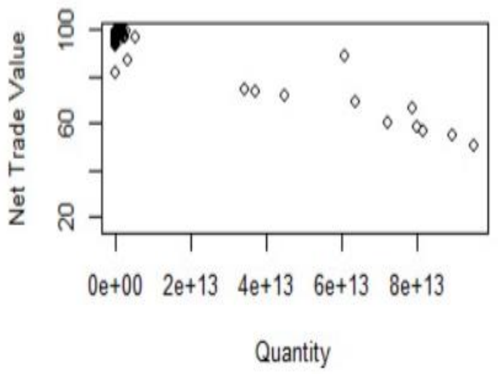
For supervised machine learning, we need to create Y label as a response variable for classification in order to train the machine and predict the output. We are creating a column "Segments" that will store the Y labels as A, B and C. For this we will assign 20% of the net value to segment A, next 30% to B and remaining 50% to C. since, segment C consists of low values we consider only segment A and B only.

4.5 Visual Representation of Data

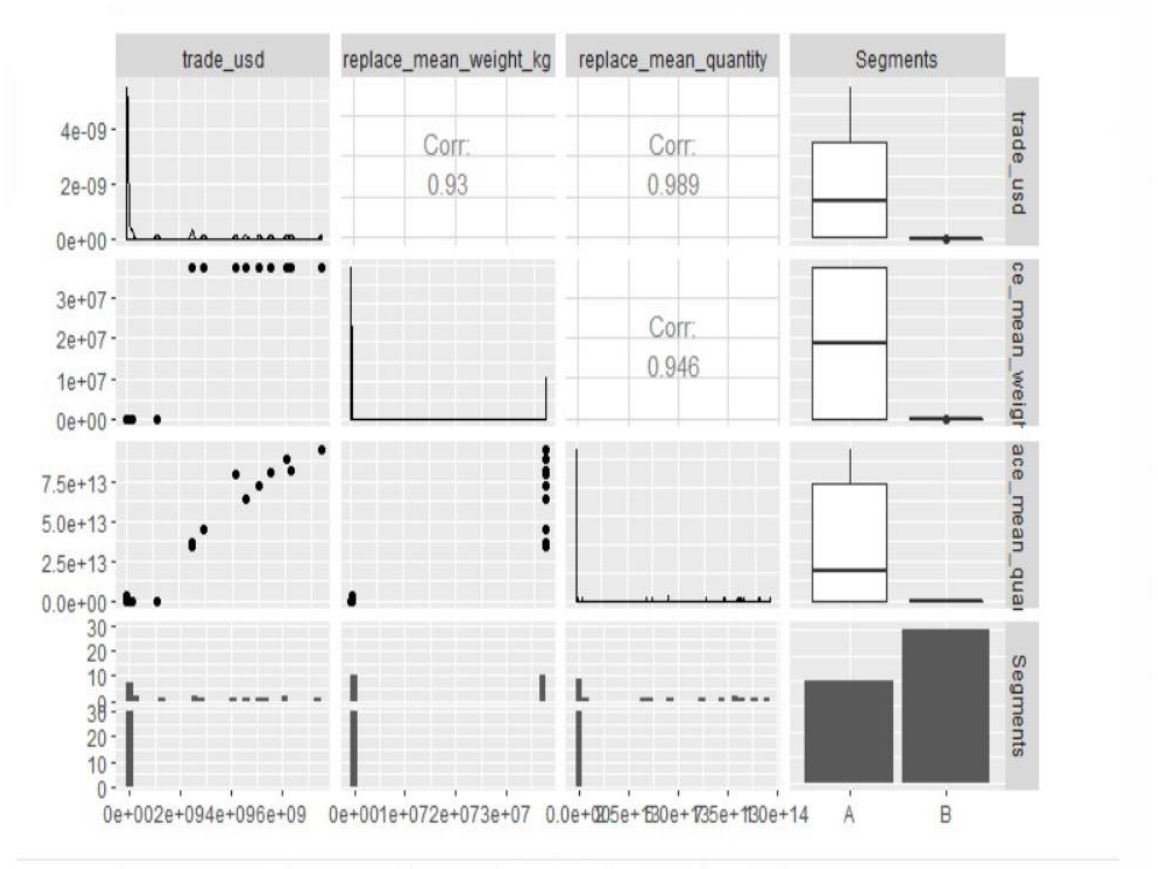
We can represent the predictor value with respect to the response value by the below plot to check how these data are distributed.





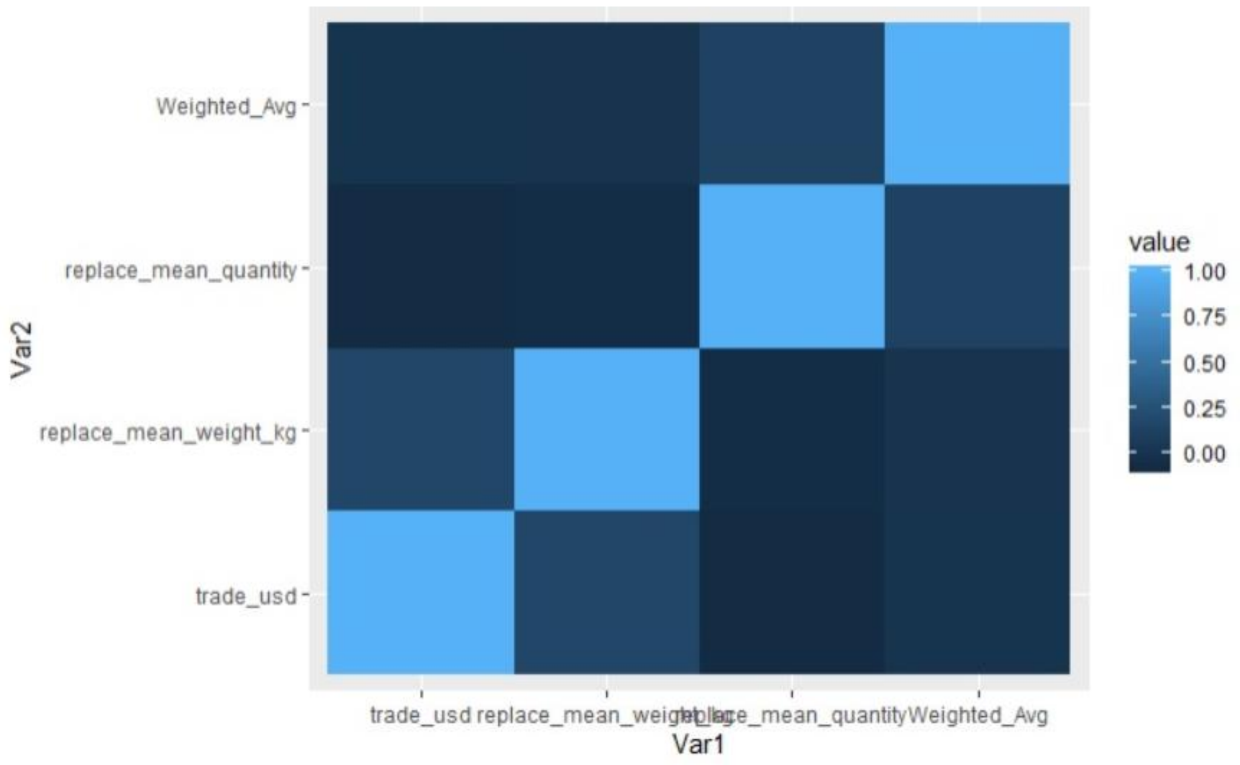


To get a visualization, correlation and linearity of the data, a scatter plot matrix is created.



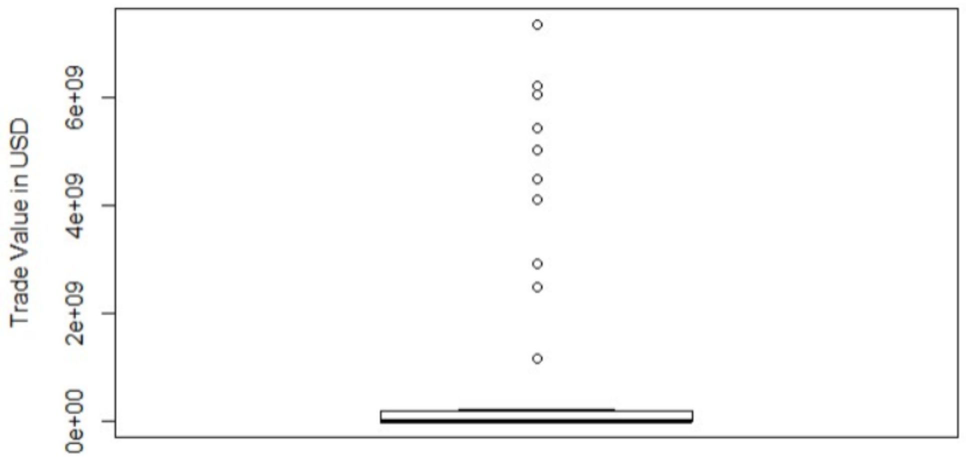
From the above plots it can be inferred that not many of the variables are highly correlated and that there happens to a large spread of data. This implies that when prediction models are being developed the normalization of the data set should be considered.

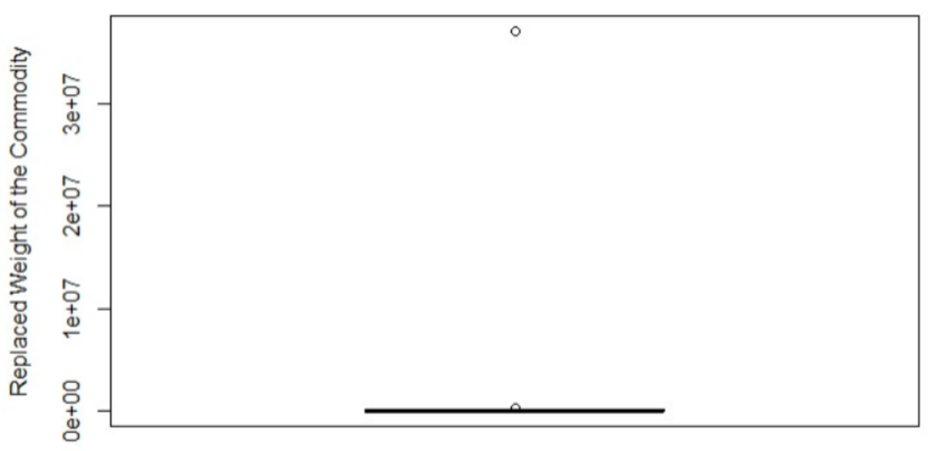
Another tool for a visual representation of the correlation would be the correlation plot. The color legend and the size of the cube describes the extent of correlation.

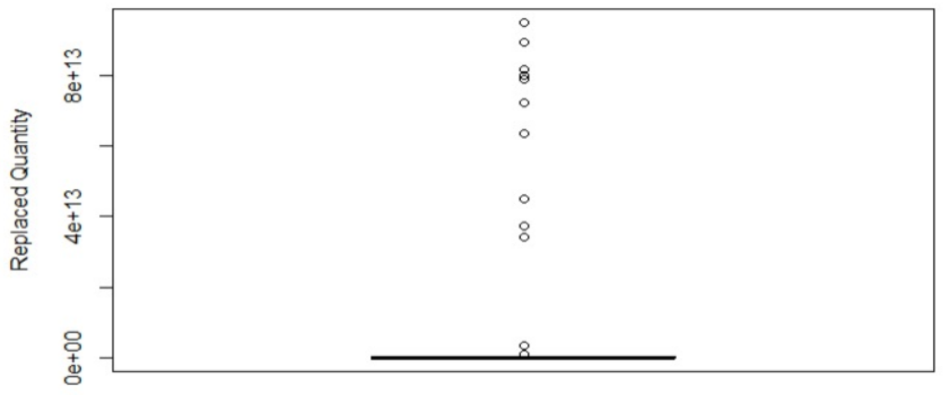


The diagonal blue cubes must correspond to value (1.00) and be ignored since they represent the correlation of a variable with itself. There seems to be several variables that have moderately significant correlation amongst themselves. These data points will have to be normalized in order to eliminate multicollinearity. Multicollinearity needs to be eliminated so that there isn't any redundancy of data.

As we removed the missing values in the data set, we can use boxplot to identify and eliminate the outliers for all the numerical predictors, if needed the box plot for the three predictor variables are shown below.







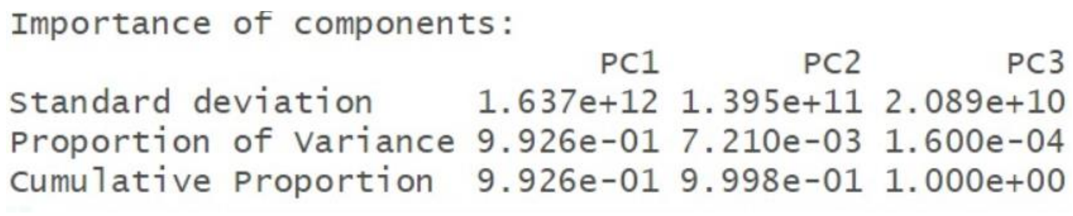
Since our dataset is very large, we have used a slice/ portion of our dataset to plot the above box plot to show the outliers in each of the predictor variables even though there are considerable number of outliers present in the data, we are not removing the outliers since all these data are real and removing them would impact the overall result.

5. DIMESION REDUCTION

In a dataset when several variables exist, it is not necessary that all the predictor variables will have a significant effect on the response variable. Large number of predictors can cause challenges with the "curse of dimensionality" where multivariate models can be highly complex that no sensible pattern or information can be derived from it.

We attempted Principal Component Analysis (PCA) on our data. Since we clearly we have three predictor variables which will significantly impact the response variable, we are not using Scree plot to extract the principle components.

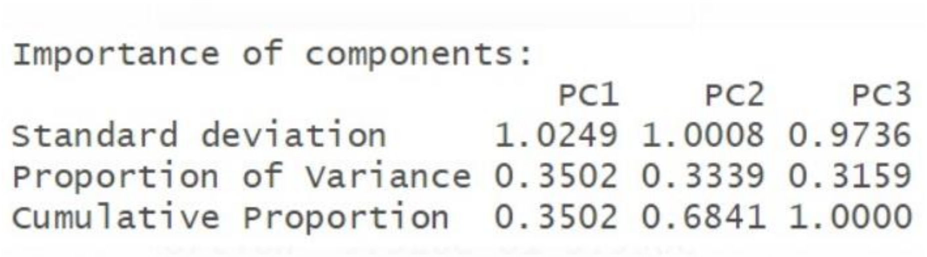
Below is the principle component course.



```
Importance of components:
                             PC1       PC2       PC3
Standard deviation     1.637e+12 1.395e+11 2.089e+10
Proportion of Variance 9.926e-01 7.210e-03 1.600e-04
Cumulative Proportion  9.926e-01 9.998e-01 1.000e+00
```

Here, we can easily see that, the first principal component is dominated by replace_mean_quantity, second principal component is dominated by trade_usd and so on. The variation of these two variables are much larger than the other variables and thus, the total variance is dominated by these two variables. To avoid, this we have to do Normalization.

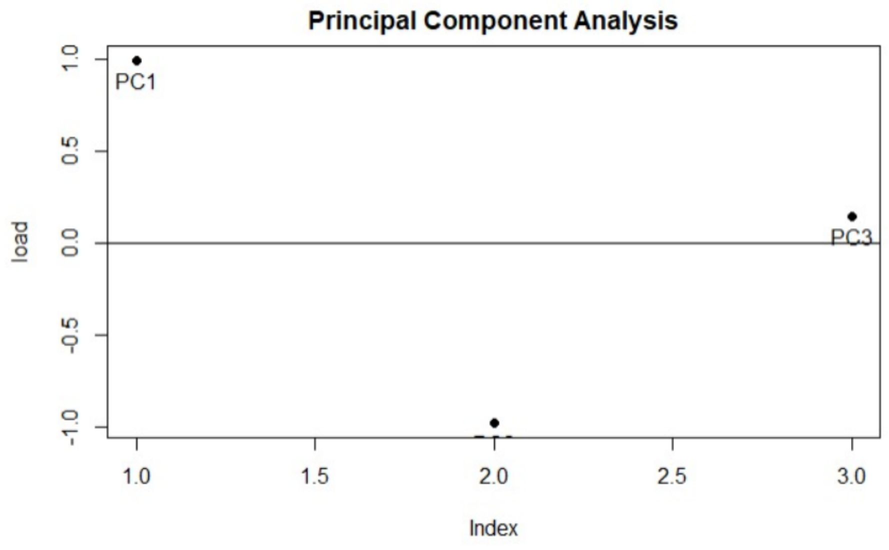
The principle component score after normalization is shown below:



```
Importance of components:
                          PC1    PC2    PC3
Standard deviation     1.0249 1.0008 0.9736
Proportion of Variance 0.3502 0.3339 0.3159
Cumulative Proportion  0.3502 0.6841 1.0000
```

From the above result we can see the dominance of the single variable from each of the principle components is eliminated. Now the principle components are orthogonally rotated to normalize the data and eliminate multi collinearity.

The below graph shows the orthogonal solution using factor.plot()

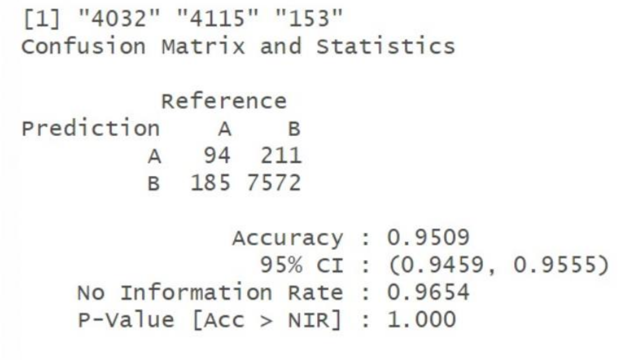


6. CLASSIFICATION METHODS

6.1 K- Nearest Neighbors (k-NN)

k-nearest-neighbors algorithm can be used for classification (of a categorical outcome). To classify a new record, the method relies on finding "similar" records in the training data. These "neighbors" are then used to derive a classification or prediction for the new record by voting (for classification)

For building the k-NN classifier , we are splitting the datasets into training and validation sets and then normalizing it. Now the normalized sets are used for building the model. The model will produce different results for different K values, initially we are assuming k= 1 and building the model. The confusion matrix for this model is as shown below.

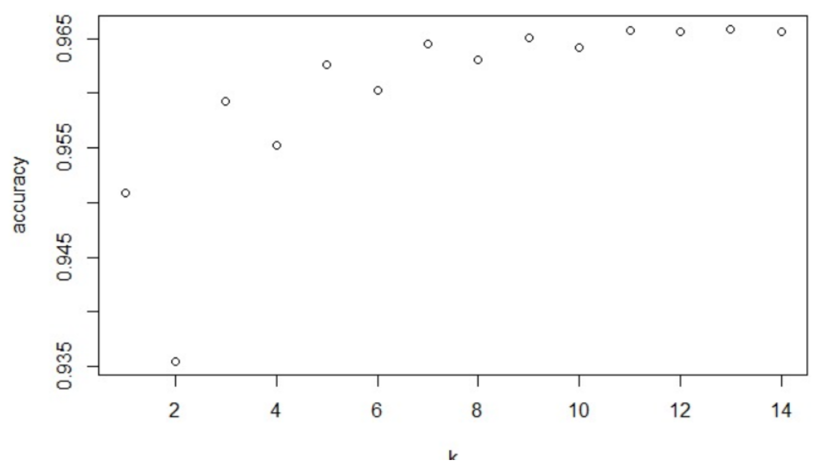


```
[1] "4032" "4115" "153"
Confusion Matrix and Statistics

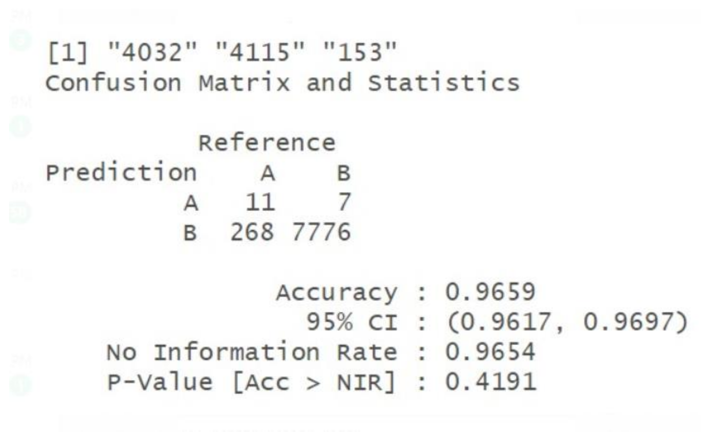
          Reference
Prediction    A    B
         A   94  211
         B  185 7572

               Accuracy : 0.9509
                 95% CI : (0.9459, 0.9555)
    No Information Rate : 0.9654
    P-Value [Acc > NIR] : 1.000
```

Now we are measuring the accuracy of K values that can be used for building the classifier. The below plot shows the accuracy of different K values.



From the above plot it is evident that K value = 13 is more accurate. Thus, we are building the K-NN classifier with K value =13. The confusion matrix for this model is shown below.



```
[1] "4032" "4115" "153"
Confusion Matrix and Statistics

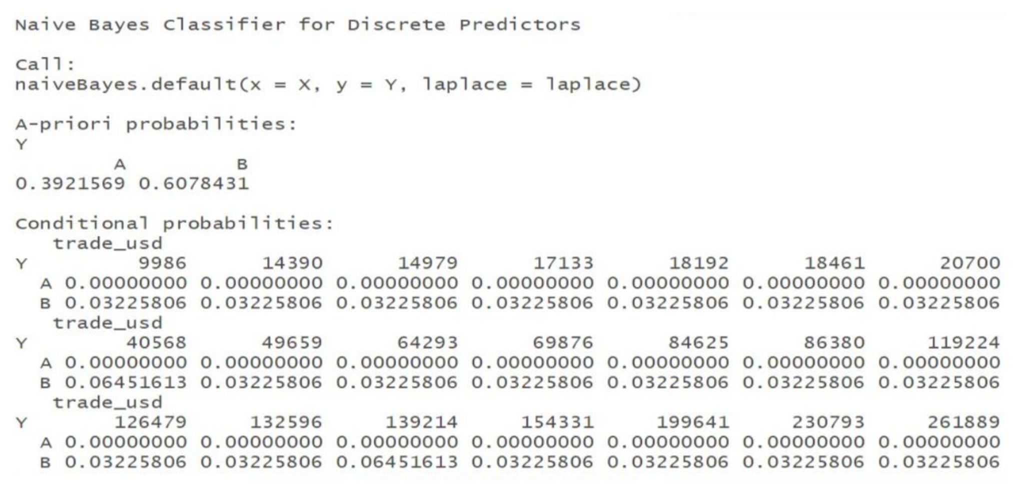
          Reference
Prediction    A    B
         A   11    7
         B  268 7776

               Accuracy : 0.9659
                 95% CI : (0.9617, 0.9697)
    No Information Rate : 0.9654
    P-Value [Acc > NIR] : 0.4191
```

Now we can see the accuracy is increased from 95.09% to 96.59% for K value = 1 and 13 respectively.

6.2 Naïve Bayer's Classifier

Naïve Bayer's is another tool for classification technique. For this we need to find all the other records with the same predictor profile (i.e., where the predictor values are the same). Determine what classes the records belong to and which class is most prevalent and then assign that class to the new record.
Below is a partial output for Naïve Bayer's classifier.



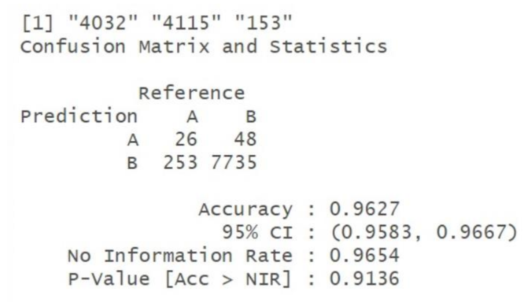
```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        A         B 
0.3921569 0.6078431 

Conditional probabilities:
   trade_usd
Y         9986      14390      14979      17133      18192      18461      20700
  A 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
  B 0.03225806 0.03225806 0.03225806 0.03225806 0.03225806 0.03225806 0.03225806
   trade_usd
Y        40568      49659      64293      69876      84625      86380     119224
  A 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
  B 0.06451613 0.03225806 0.03225806 0.03225806 0.03225806 0.03225806 0.03225806
   trade_usd
Y       126479     132596     139214     154331     199641     230793     261889
  A 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
  B 0.03225806 0.03225806 0.06451613 0.03225806 0.03225806 0.03225806 0.03225806
```

The confusion matrix for Naïve Bayer’s classification is shown below:



```
[1] "4032" "4115" "153"
Confusion Matrix and Statistics

          Reference
Prediction    A    B
         A   26   48
         B  253 7735

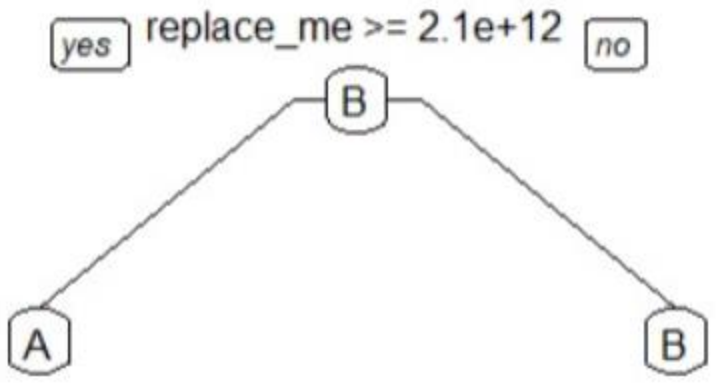
               Accuracy : 0.9627
                 95% CI : (0.9583, 0.9667)
    No Information Rate : 0.9654
    P-Value [Acc > NIR] : 0.9136
```

The accuracy for the Naïve Bayer’s classifier is 96.27%

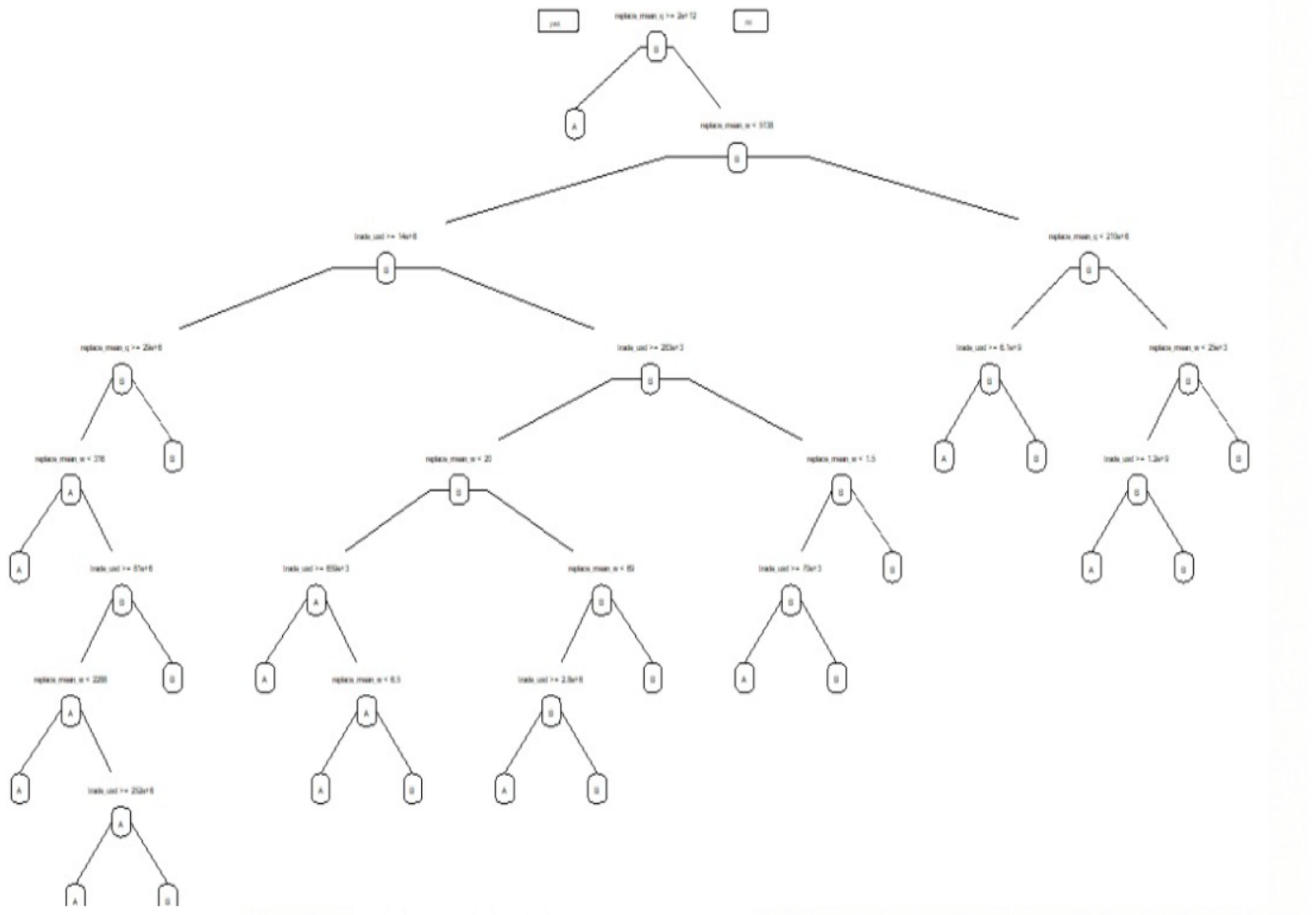
6.3 Classification Tree

Among the data-driven methods, trees are the most transparent and easy to interpret. Trees are based on separating records into subgroups by creating splits on predictors. These splits create logical rules that are transparent and easily understandable, for example, if replace_mean_quantity >= 2.1*e^12 then class = A. The resulting subgroups should be more homogeneous in terms of the outcome variable, thereby creating useful prediction or classification rules.

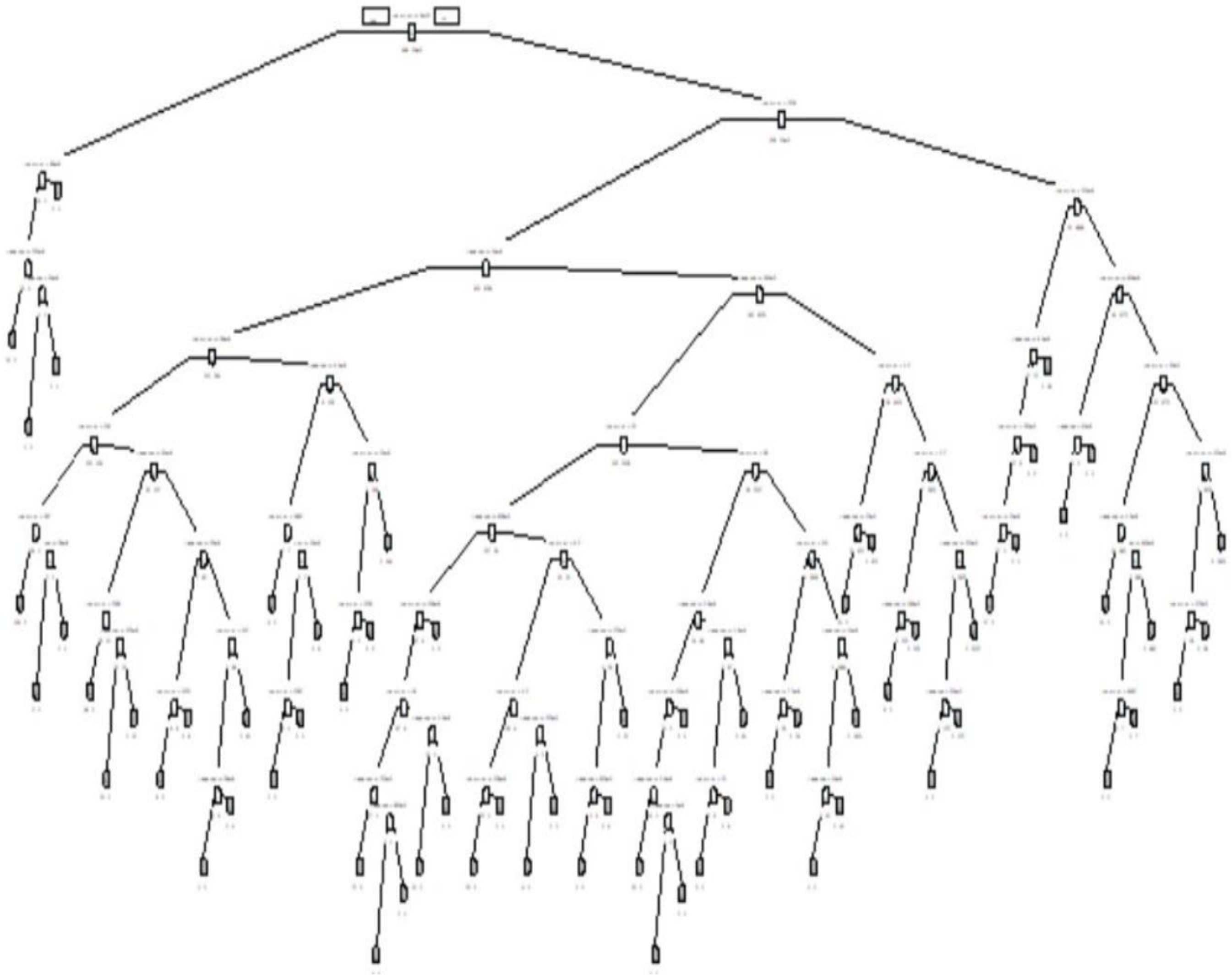
Two key ideas underlying trees: recursive partitioning (for constructing the tree) and pruning (for cutting the tree back). Below is the tree representation of 1st split



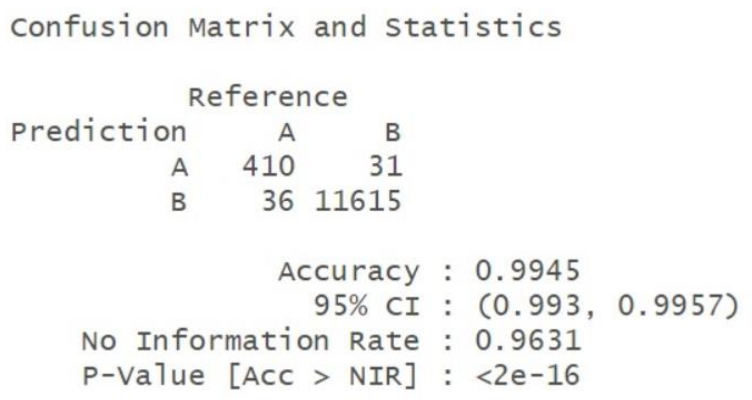
The default classification tree is given below.



A full/deeper classifier tree for segmentation of commodity_trade dataset obtained as:



Now we have to classify the validation data using a tree and compute confusion matrices and accuracy for the training and validation data. The confusion matrix is given below :



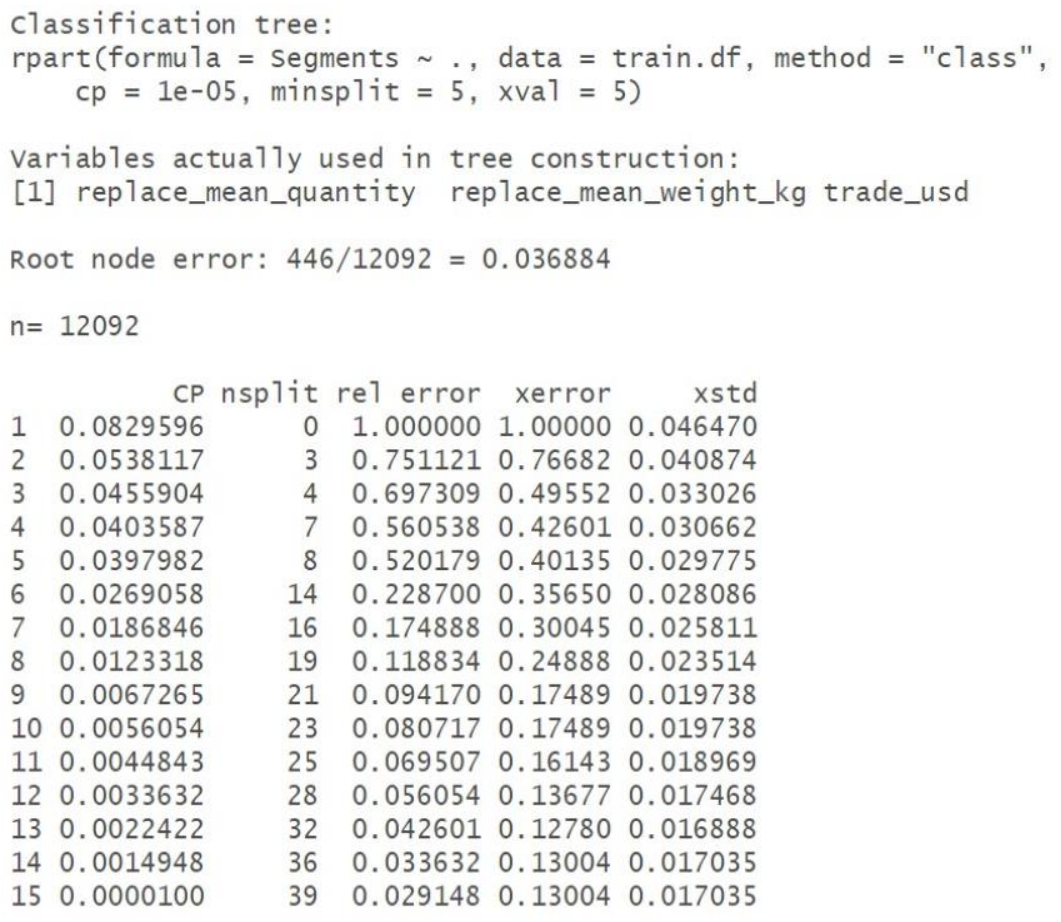
```
Confusion Matrix and Statistics

          Reference
Prediction     A     B
         A   410    31
         B    36 11615

               Accuracy : 0.9945
                 95% CI : (0.993, 0.9957)
    No Information Rate : 0.9631
    P-Value [Acc > NIR] : <2e-16
```

The accuracy for the classifier tree is 99.45%. One danger in growing deeper trees on the training data is overfitting. overfitting will lead to poor performance on new data. If we look at the overall error at the various sizes of the tree, it is expected to decrease as the number of terminal nodes grows until the point of overfitting

Below is the output for the code for tabulating tree errors as a function of the complexity parameter (CP)



```
Classification tree:
rpart(formula = Segments ~ ., data = train.df, method = "class",
    cp = 1e-05, minsplit = 5, xval = 5)

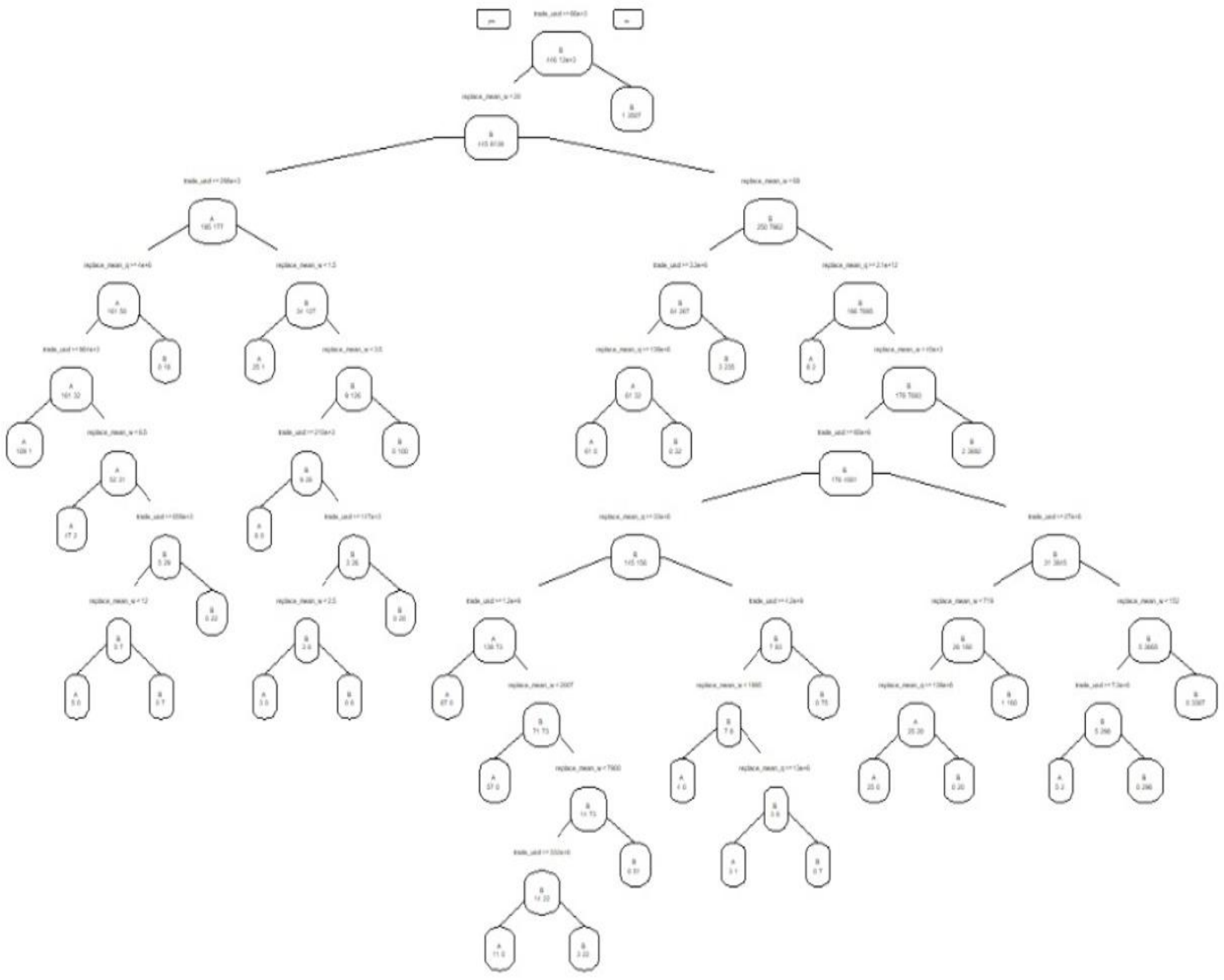
Variables actually used in tree construction:
[1] replace_mean_quantity  replace_mean_weight_kg trade_usd

Root node error: 446/12092 = 0.036884

n= 12092

          CP nsplit rel error  xerror     xstd
1  0.0829596      0  1.000000 1.00000 0.046470
2  0.0538117      3  0.751121 0.76682 0.040874
3  0.0455904      4  0.697309 0.49552 0.033026
4  0.0403587      7  0.560538 0.42601 0.030662
5  0.0397982      8  0.520179 0.40135 0.029775
6  0.0269058     14  0.228700 0.35650 0.028086
7  0.0186846     16  0.174888 0.30045 0.025811
8  0.0123318     19  0.118834 0.24888 0.023514
9  0.0067265     21  0.094170 0.17489 0.019738
10 0.0056054     23  0.080717 0.17489 0.019738
11 0.0044843     25  0.069507 0.16143 0.018969
12 0.0033632     28  0.056054 0.13677 0.017468
13 0.0022422     32  0.042601 0.12780 0.016888
14 0.0014948     36  0.033632 0.13004 0.017035
15 0.0000100     39  0.029148 0.13004 0.017035
```

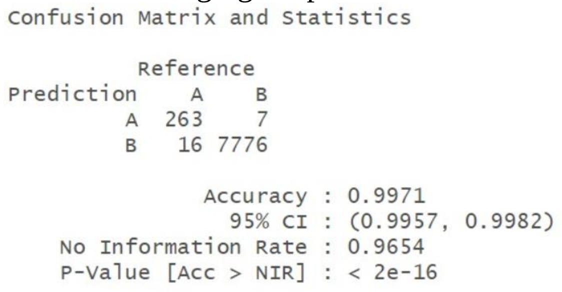
Pruning the tree with the validation data solves the problem of overfitting. The pruned classification tree is given below:



6.4 Random Forest

The basic idea in random forests is to:

Draw multiple random samples, with replacement, from the data (this sampling approach is called the bootstrap). Using a random subset of predictors at each stage, fit a classification (or regression) tree to each sample (and thus obtain a “forest”). Then combine the predictions/classifications from the individual trees to obtain improved predictions. Use voting for classification and averaging for prediction.



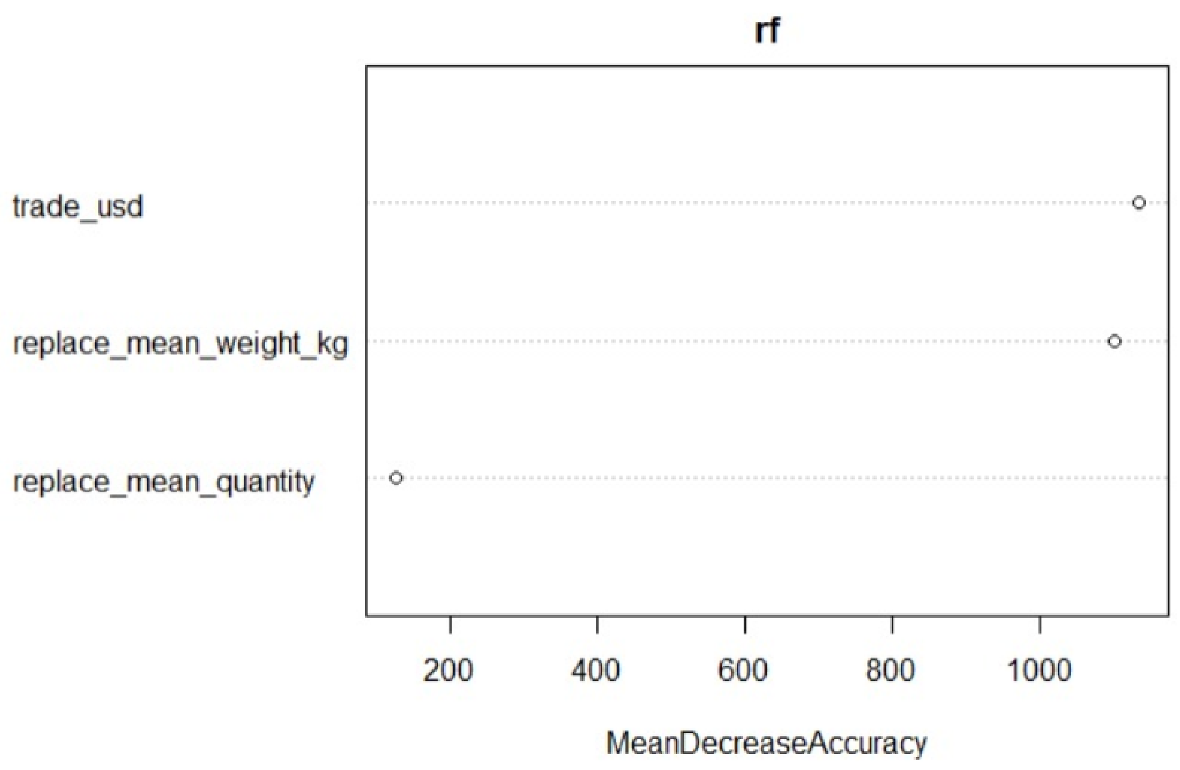
```
Confusion Matrix and Statistics

          Reference
Prediction    A    B
         A  263    7
         B   16 7776

               Accuracy : 0.9971
                 95% CI : (0.9957, 0.9982)
    No Information Rate : 0.9654
    P-Value [Acc > NIR] : < 2e-16
```

The output illustrates applying random forest. The accuracy of the random forest is slightly higher than the single default tree that we fit earlier (compare to the validation performance in 6.3).

Unlike a single tree, results from a random forest cannot be displayed in a tree-like diagram, thereby losing the interpretability that a single tree provides

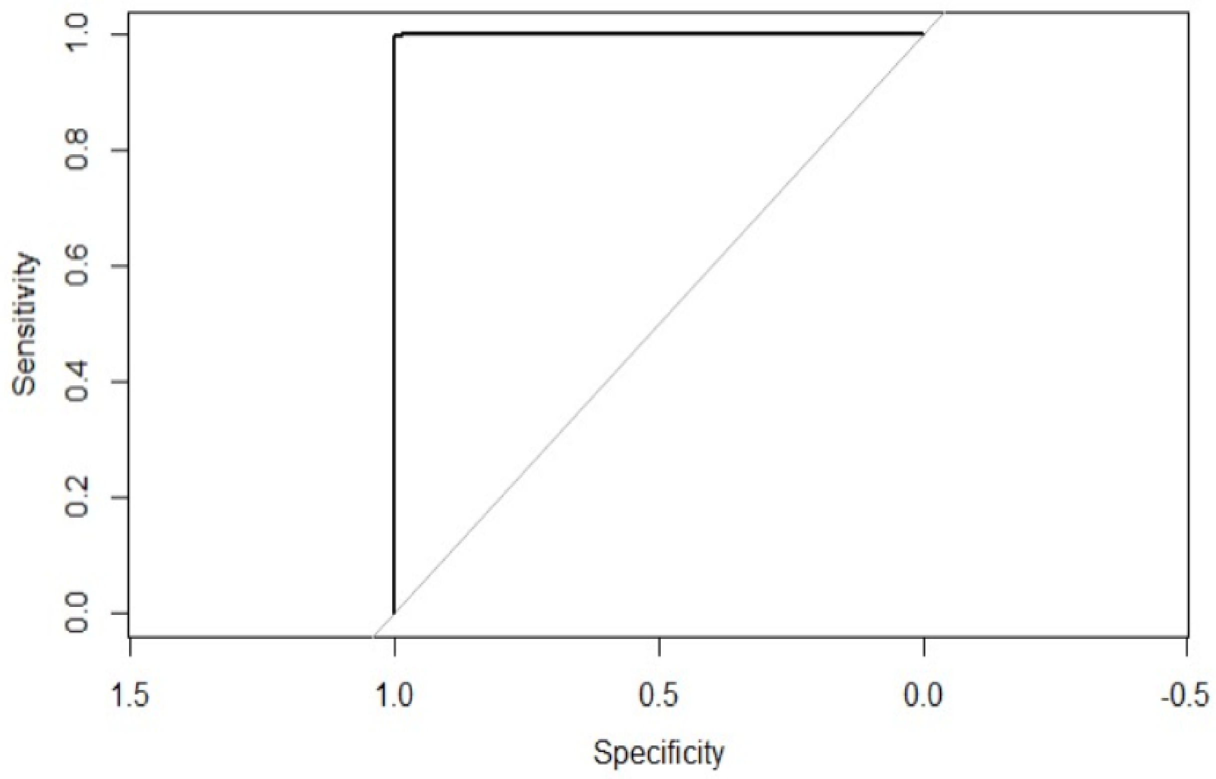


However, random forests can produce "variable importance" scores, which measure the relative contribution of the different predictors. The importance score for a particular predictor is computed by summing up the decrease in the Gini index for that predictor over all the trees in the forest. The above figure shows the variable importance plots generated from the random forest model

7. PERFORMANCE EVALUATION

So far to check the performance of the algorithms the accuracy from the classification matrix has been calculated. Which is an indication of how well the model fits the training data and how accurately can it predict new cases. Amongst all the models used so far, the random forests method seemed to be the most successful with an accuracy of 99.75% and hence will be scrutinized under performance evaluation.

There are however other methods that can be employed. Since our project is keener on the segments A and B, it is appropriate to use the ROC curve as a performance evaluation criterion. The straight line in plot, also known as the random classification line, is the benchmark for the algorithm. It shows the accuracy of a random classification of records. The higher the curve is on the y axis is relation to the random classification line, the more accurate it is. It is visually represented by the ROC curve and quantitatively represented by the AUC index.



The AUC or the area under the curve can reach a maximum of 1. For the random forest classification, the AUC is 99.75% which is a very accurate model. Hence it can be used with 99.75% of confidence to distinctively classify the 2 classes of the response variable from each other.

8. CONCLUSION

Four different classification methods were used to predict the segments A and B out of which the Random Forests gave a better prediction performance. Since this method uses multiple classification trees and votes based maximum likelihood, it gives a more accurate classification of categories and can more efficiently distinguish between segments A and B.

9. APPENDICIES

~~~
---
title: "DM Project"
author: "Shivahari Revathi Venkateswaran, Niranjan Ramesh Babu"
date: "11/24/2019"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

# Installing packages:

```{r}
install.packages("e1071")

```

# Loading the Libraries:

```{r}
library(dplyr)
library(tidyr)
library(ggplot2)
library(gplots)
library(reshape2)
library(caret)
library(FNN)
library(sqldf)
library(dplyr)
library(e1071)
library(rpart)
library(rpart.plot)
~~~

~~~
library(randomForest)
library(rpart)
library(adabag)
library(psych)

library(GPArotation)
```

# Loading the Data:

```{r}

Commodity_trade <- read.csv("C:/Users/venka/OneDrive - Northeastern University/sem
3/DM/Project/global-commodity-trade-statistics/commodity_trade_statistics_data.csv")

head(Commodity_trade)


```


```{r}
summary(Commodity_trade)
```


# Feature Selection:

```{r}
temp <- Commodity_trade %>% select(comm_code, trade_usd, weight_kg, quantity)


head(temp,5)

```


# Replacing all the Zeros with NULL values for Statistical Analysis:

```{r}

temp[temp == 0] <- NA

```


# Handling the Missing Values:

```{r}

colSums(is.na(temp))
~~~

~~~~
list_na <-  colnames(temp)[ apply(temp, 2, anyNA) ]

list_na

```

# Heat Map to show the Missing values:

```{r}
missing_slice <- temp[c(1:1000),]

missing_slice # Slicing out 1000 values since the dataset is too huge

heatmap(1 * is.na(missing_slice), Rowv = NA, Colv = NA)
# This shows that the weight_kg and quantity columns only have missing values

```




# Recoding the Missing Values:

```{r}

average_missing <- apply(temp[,colnames(temp) %in% list_na],
      2,
      mean,
      na.rm =  TRUE)
average_missing

```

# Replacing the Missing values with Mean values:

```{r}
temp_replace <- temp %>%
   mutate(replace_mean_weight_kg  = ifelse(is.na(weight_kg), average_missing[1], weight_kg),
   replace_mean_quantity = ifelse(is.na(quantity), average_missing[2], quantity ))

head(temp_replace)


colSums(is.na(temp_replace))

```


# Creating the Weighted Average Column:

```{r}
~~~~

```
temp1 <- mutate(temp_replace, Weighted_Avg = ((temp_replace$trade_usd/(temp_replace$replace_mean_weight_kg)*(temp_replace$replace_mean_quantity))))


head(temp1,1000)

```
```

# Visualizing the Correlation between Dependent(Y) and Independent(X) variables:

```
```{r}

Correlation_slice <- temp1 %>% select(comm_code,trade_usd,replace_mean_weight_kg,replace_mean_quantity,Weighted_Avg)

Correlation_slice <- Correlation_slice[c(1:1000),]

Correlation_slice

typeof(Correlation_slice)

Correlation_slice <- as.data.frame(Correlation_slice)

Correlation_slice <- Correlation_slice[,c(2:5)]

cormat <- round(cor(Correlation_slice),2)

cormat

head(cormat)

melted_cormat <- melt(cormat)
head(melted_cormat)

ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile()

```
```

# Creating a new column for Cumulative Sum:

```
```{r}
temp2 <- temp1 %>% arrange(desc(Weighted_Avg)) %>% mutate(cum_value = cumsum(Weighted_Avg))

head(temp2)
```

~~~
```


# Creating a Column for Net_value:

```{r}
Overall_Value <- sum(temp2$Weighted_Avg)


Overall_Value
```



```{r}
temp3 <- temp2 %>%  mutate(Net_value = ((temp2$cum_value/Overall_Value)*100))

tail(temp3,20)

```


# Creating the Y lable by Segmentation Technique:

```{r}

temp4 <- temp3 %>% mutate(Segments = ifelse(Net_value<97, "A", ifelse(Net_value < 99.7, "B", "C")))
temp4

```


```{r}
new <- sqldf("select * from temp4 where Segments like 'A' or Segments like 'B' ")

head(new)

```




# Scatter Plot Matrix with Net_value and three predictors:

```{r}
## simple plot
# use plot() to generate a matrix of 4X4 panels with variable name on the diagonal,
# and scatter plots in the remaining panels.

scatter_matrix <- new
~~~

```
#plot(scatter_matrix[, c(1,2,3,4)])
# alternative, nicer plot (displayed)
library(GGally)
ggpairs(scatter_matrix[, c(2,5,6,7)])

```
```

```
```{r}

temp5 <- new %>% arrange(comm_code) %>% select(trade_usd, replace_mean_weight_kg,
replace_mean_quantity, Segments)

temp5

```
```

```
```{r}
library(ggplot2)
par(mfrow=c(2,2))
plot(x = new$trade_usd, y = new$Net_value,
xlab = "Trade Value in USD",
ylab = "Net Trade Value")
plot(x = new$weight_kg, y = new$Net_value,
xlab = "Weight of the Commodity",
ylab = "Net Trade Value")
plot(x = new$quantity, y = new$Net_value,
xlab = "Quantity",
ylab = "Net Trade Value")

```
```

```
```{r}
temp6 <- sqldf("select * from temp5 where Segments like 'A'")

temp7 <- head(temp6, 20)

temp8 <- sqldf("select * from temp5 where Segments like 'B'")

temp9 <- head(temp8, 30)

temp10 <- rbind(temp7, temp9)

temp10

```
```

# Box Plot:
```

~~~~
```{r}
outlier1 <- boxplot(temp10$trade_usd, ylab = "Trade Value in USD")
outlier2 <- boxplot(temp10$replace_mean_weight_kg, ylab = "Replaced Weight of the Commodity")
outlier3 <- boxplot(temp10$replace_mean_quantity, ylab = "Replaced Quantity")


```

Storing Outliers into a vector:

```{r}
boxplot(temp10$trade_usd)$out
```

```{r}
outlier1 <- boxplot(temp10$trade_usd, plot = FALSE)$out

print(outlier1)
```

Removing or Not Removing the Outliers:

```{r}

# Find which rows the outliers are

temp10[which(temp10$trade_usd %in% outlier1), ]

```


PCA

```{r}
pcs <- prcomp(na.omit(temp5[,c(1:3)]))
summary(pcs)


```

```{r}
pcs$rot[,1:3]
```
~~~~

Here, we can easily see that, the first principal component is dominated by replace_mean_quantity, Second principal component is dominated by trade_usd and so on. The variation of these two variables are much larger than the other variables and thus, the total variance is dominated by these two variables. To avoid, this we have to do Normalization.

~~~
# Normalization: Here all the x componets are having different units of measurement and thus we require
Normalization

```{r}
pcs.cor <- prcomp(na.omit(temp5[,c(1:3)]), scale. = T)
summary(pcs.cor)
```

```{r}
pcs.cor$rot[,1:3]


```

```{r}
Pc1 <- principal(pcs.cor$rot[,1:3], nfactors = 1, scores = TRUE, rotate = "none")
Pc1
```

# Rotate the components:

```{r}
Pc2 <- principal(pcs.cor$rot[,1:3], nfactors = 1, rotate = "varimax")
Pc2
```

# Computing the components score:

```{r}
Pc3 <- principal(pcs.cor$rot[,1:3], nfactors = 1, scores = TRUE, rotate = "none")
Pc3$scores
```

# Graph an orthogonal solution using factor.plot():

```{r}
factor.plot(Pc1, labels = rownames(Pc1$loadings))
```






```{r}
scatter_matrix <- temp10

#plot(scatter_matrix[, c(1,2,3,4)])
# alternative, nicer plot (displayed)
library(GGally)
~~~

~~~~
ggpairs(scatter_matrix[, c(1,2,3,4)])
```




# Naive Bayes Classifier:

```{r}
naive.df <- temp10
naive.df$trade_usd <- factor(naive.df$trade_usd)
naive.df$replace_mean_weight_kg <- factor(naive.df$replace_mean_weight_kg)
naive.df$replace_mean_quantity <- factor(naive.df$replace_mean_quantity)
naive.df$Segments <- factor(naive.df$Segments)

# Creating Training and Validation set
selected.var <- c(1,2,3,4)
train.index <- sample(c(1:dim(naive.df)[1], dim(naive.df)[1]*0.6))
train.df <- naive.df[train.index, selected.var]
valid.df <- naive.df[-train.index, selected.var]
# Run naive bayes

naive.nb <- naiveBayes(Segments ~ ., data = train.df)
naive.nb


```

```{r}

```


# Pivot table for Commodity Segments by trade_usd values (Training Data)

```{r}

pivot_table  <- prop.table(table(train.df$Segments, train.df$trade_usd), margin = 1)

pivot_table


```

# Code for scoring data using Naive Bayes

```{r}
pred.prob <- predict(naive.nb, newdata = as.factor(valid.df), type = "raw")
~~~~

# predict class membership

pred.class <- predict(naive.nb, newdata = as.factor(valid.df))

df <- data.frame(actual = as.factor(valid.df$Segments), predicted = pred.class, pred.prob)

df[valid.df$trade_usd == 340000 & valid.df$replace_mean_weight_kg == 2 , valid.df$replace_mean_quantity == 275059223,]

# Code for Confusion Matrices:

```{r}

pred.class <- predict(naive.nb, newdata = train.df)

confusionMatrix(pred.class, as.factor(train.df$Segments))

```

# Knn Classifier:

```{r}
set.seed(111)
train.index <- sample(row.names(temp5), 0.6*dim(temp5)[1])
valid.index <- setdiff(row.names(temp5), train.index)
train.df <- temp5[train.index, ]
valid.df <- temp5[valid.index, ]

```

Inserting the new data

```{r}
new.df <- data.frame(trade_usd = 340000, replace_mean_weight_kg = 2, replace_mean_quantity = 275059223)
```

##Scatter plot

```{r}
plot(trade_usd ~ replace_mean_weight_kg, data=train.df, pch=ifelse(train.df$Segments=="A", 1, 3))
```

~~~
text(train.df$trade_usd, train.df$replace_mean_weight_kg, rownames(train.df), pos=4)
text(60, 20, "X")
legend("topright", c("A", "B", "New"), pch = c(1, 3, 4))
?plot()
```

# Initialize normalized training, validation data frames to originals

```{r}

train.norm.df <- train.df
valid.norm.df <- valid.df
temp5.norm.df <- temp5
new.norm.df <- new.df
```

# Use preProcess() from caret package to normalize the date

```{r}

norm.values <- preProcess(train.df[,1:3], method = c("center", "scale"))
train.norm.df[,1:3] <- predict(norm.values, train.df[,1:3])
valid.norm.df[,1:3] <- predict(norm.values, valid.df[,1:3])
temp5.norm.df[,1:3] <- predict(norm.values, temp5[,1:3])
new.norm.df[,1:3] <- predict(norm.values, new.df)

```



# Use knn() to compute knn. This is available from the library FNN

```{r}

nn <- knn(train = train.norm.df[,1:3], test = new.norm.df, cl = train.norm.df[,4], k = 3)

row.names(train.df)[attr(nn, "nn.index")]

nn

#confusionMatrix(nn, as.factor(train.df$Segments))

```

```{r}
#confusionMatrix(knn.pred.new, as.factor(train.df$Segments))
knn.pred.new <- knn(train.norm.df[,1:3], valid.norm.df[,1:3],
                    cl = train.norm.df[,4], k =5)
row.names(train.df)[attr(nn, "nn.index")]

#knn.pred.new
~~~

~~~~
confusionMatrix(knn.pred.new, as.factor(valid.df$Segments))

```




# Measuring the accuracy of different k values:

```{r}

# Initializing a data frame with two columns: k, accuracy.
accuracy.df <- data.frame(k = seq(1,14,1), accuracy = rep(0,14))


# compute knn for different k on validation.

for(i in 1:14) {
  knn.pred <- knn(train.norm.df[,1:3], valid.norm.df[,1:3], cl = train.norm.df[,4], k = i)
  accuracy.df[i,2] <- confusionMatrix(knn.pred, factor(valid.norm.df[,4]))$overall[1]
}

accuracy.df

plot(accuracy.df)


```

# Classifying the new data using the best k=13

```{r}
knn.pred.new <- knn(train.norm.df[,1:3], new.norm.df,
                    cl = train.norm.df[,4], k =13)
row.names(train.df)[attr(nn, "nn.index")]

knn.pred.new

```



```{r}

#confusionMatrix(knn.pred.new, as.factor(train.df$Segments))
knn.pred.new <- knn(train.norm.df[,1:3], valid.norm.df[,1:3],
                    cl = train.norm.df[,4], k =13)
row.names(train.df)[attr(nn, "nn.index")]

#knn.pred.new
~~~~

~~~~
confusionMatrix(knn.pred.new, as.factor(valid.df$Segments))


```


# Classifier tree:

```{r}
class.tree <- rpart(Segments ~ ., data = temp5,
                    control = rpart.control(maxdepth = 2), method = "class")

prp(class.tree, type = 1, split.font = 1, varlen = -10)


```


# Code for creating a default Classification Tree:

```{r}

#Partition
set.seed(1)
#temp5 <- as.factor(temp5)

train.index <- sample(c(1:dim(temp5)[1]), dim(temp5)[1]*0.6)
train.df <- temp5[train.index, ]
valid.df <- temp5[-train.index, ]

#train.df$Segments


#Classification tree
default.ct <- rpart(Segments ~ ., data = train.df, method = "class")

#Plot tree
prp(default.ct, type = 1, under = TRUE, split.font = -10, varlen = -10)

```


# COde for creating a deeper Claasification Tree:

```{r}
deeper.ct <- rpart(Segments ~ ., data = train.df, method = "class", cp = 0, minsplit = 1)

#Count number of levels
length(deeper.ct$frame$var [deeper.ct$frame$var == "<leaf>"])
~~~~

~~~~
#plot tre
prp(deeper.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = 10,
   box.col = ifelse(deeper.ct$frame$var == "<leaf>", 'gray', 'white'))


```


# Confusion matrices and accuracy for the default (small) and deeper (full) classification trees, on the training and validation sets of the commodity trade data

```{r}

#classify records in the validation set

default.ct.point.pred.train <- predict(default.ct, train.df, type = "class")

#generate confusion matrix for training data
confusionMatrix(default.ct.point.pred.train, as.factor(train.df$Segments))



```

# Code for tabulating tree errors as a function of the complexinty parameter (CP)

```{r}

cv.ct <- rpart(Segments ~., data = train.df, method = "class", cp = 0.00001, minsplit = 5 , xval = 5)
printcp(cv.ct)

```

# Code for pruning the tree:

```{r}
pruned.ct <- prune(cv.ct,
                   cp = cv.ct$cptable[which.min(cv.ct$cptable[,"xerror"]), "CP"])
length(pruned.ct$frame$var[pruned.ct$frame$var == "<leaf>"])
prp(pruned.ct, type = 1, extra = 1, split.font = 1, varlen = -10)
```


# Random Forest:

```{r}


rf <- randomForest(as.factor(Segments) ~., data = train.df, ntree = 500,
                   mtry = 4, nodesize = 5, importance = TRUE)
~~~~

~~~~
# Variable importance plot
varImpPlot(rf, type = 1)

# Confusion matrix

rf.pred <- predict(rf, valid.df)



#rf.pred <- as.factor(rf.pred)
confusionMatrix(rf.pred, as.factor(valid.df$Segments))

```

```{r}
install.packages("ROCR")
library(ROCR)
```

```{r}
ROCRpred <- prediction(rf.pred, as.factor(valid.df$Segments))
prediction()
```





# Logistics Regression:

```{r}
head(temp5)
```

```{r}
logistics.df <- temp5[,c(1:4)]
#temp5$Segments <- factor(temp5$Segments, levels = c(1,2,3), labels = c("A", "B"))
set.seed(2)
train.index <- sample(c(1:dim(logistics.df)[1]), dim(logistics.df)[1]*0.6)
train.df <- logistics.df[train.index, ]
valid.df <- logistics.df[-train.index, ]
# run logiatics regression

logit.reg <- glm(as.factor(Segments) ~., data = train.df, family = "binomial")
options(scipen= 99)
summary(logit.reg)


```
~~~~

~~~
```{r}
head(logit.reg,1)
```


# Evaluating the Performance:

```{r}
logit.reg.pred <- predict(logit.reg, valid.df[, -4], type = "response")

# first 5 actual and predicted records

data.frame(actual = valid.df$Segments[1:5], predicted = logit.reg.pred[1:5])
```


```{r}
class(as.factor(logit.reg$residuals))
class(as.factor(valid.df$Segments))
length(levels(as.factor(logit.reg.pred)))
length(levels(as.factor(valid.df$Segments)))
confusionMatrix(as.factor(logit.reg.pred[1:5]), as.factor(valid.df$Segments[1:5]))
```


# Validation:


```{r}
predwithprob = predict(rf, valid.df, type = 'prob')
auc = auc(valid.df$Segments, predwithprob[,2])
plot(roc(valid.df$Segments, predwithprob[,2]))
```
~~~