Informatics Institute of Technology

Collaboration with University of Westminster, UK

Machine Learning & Data Mining

5DATA001C.2

Name - Ranasinghe Perera
Uow Number – w1838855
IIT Number - 20200905

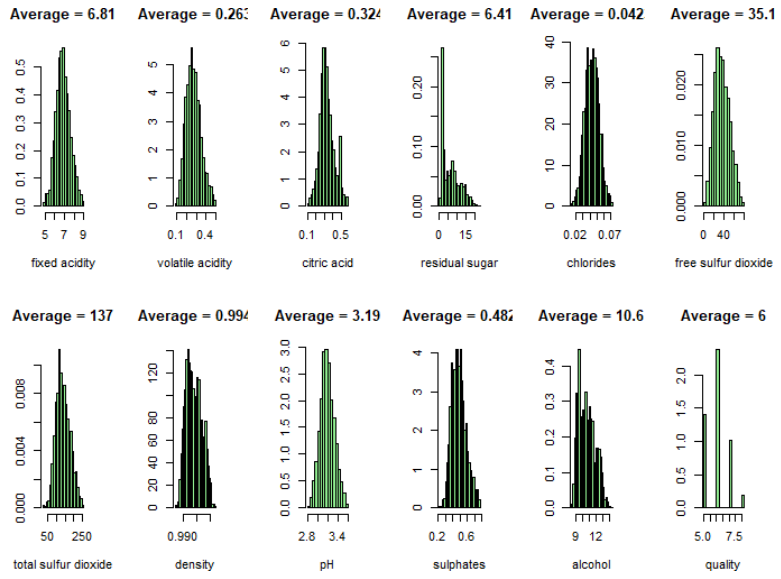
Pre-processing tasks

Two pre-processing mothers are used in the attempt of clustering in this section

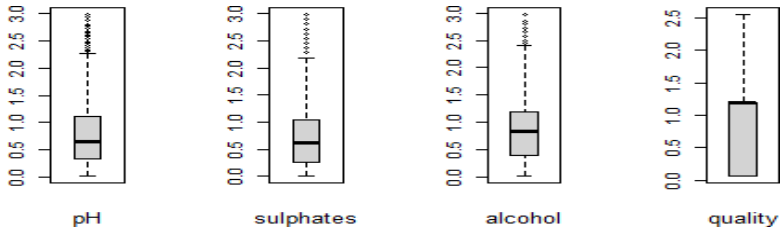
Outliers removal

```
outliers = c()
for (i in 1:11 ) {# for loop for 1 to 11
stats = boxplot.stats(wine_data_set[[i]])$stats#satatus
rows_of_bottom_outlier = which(wine_data_set[[i]] < stats[1]) #rows for bottom outlier
rows_of_top_outlier = which(wine_data_set[[i]] > stats[5]) #rows for top outlier outliers = c(outliers ,rows_of_top_outlier[ !rows_of_top_o
#outliers
outliers = c(outliers , rows_of_bottom_outlier[ !rows_of_bottom_outlier %in% outliers
] )#outliers
}


clean_wine_data_set = wine_data_set[-outliers, ]# wine data set oldpar = par(mfrow = c(2,6))#older path detecting
for (i in 1:12 ) {# for loop 1 to 12
truehist(clean_wine_data_set[[i]], xlab = names(clean_wine_data_set)[i], col = 'red', #clearing
}

main = paste("Average =", signif(mean(clean_wine_data_set[[i]]),3)))

par(oldpar)
```

Bellow diagram shows result of the outliners process



Scalling



Scaling data

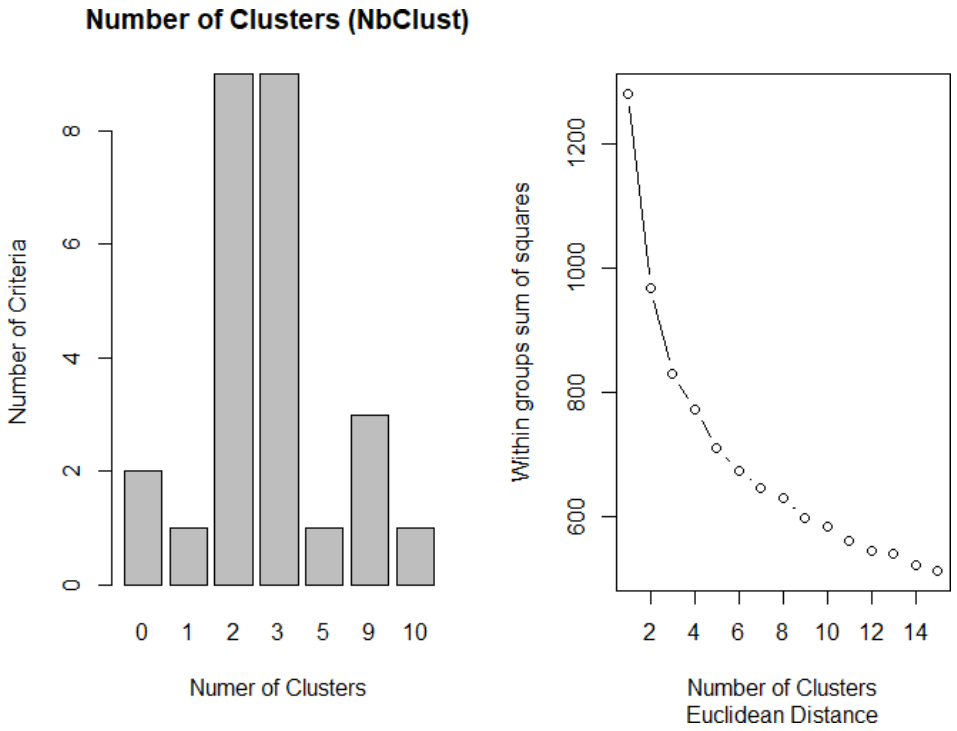
```
#scale  PCA dataset
#main data scalling process
pca_data_scalled= as.data.frame(pca_dset_with_removed_outliers)

#Dispaly the scalled data
pca_data_scalled
dim(pca_data_scalled)
```

There are 3 methods are used to define the clusters

1) Euclidean Distance
2) Manhattan Distance
3) Elbow method

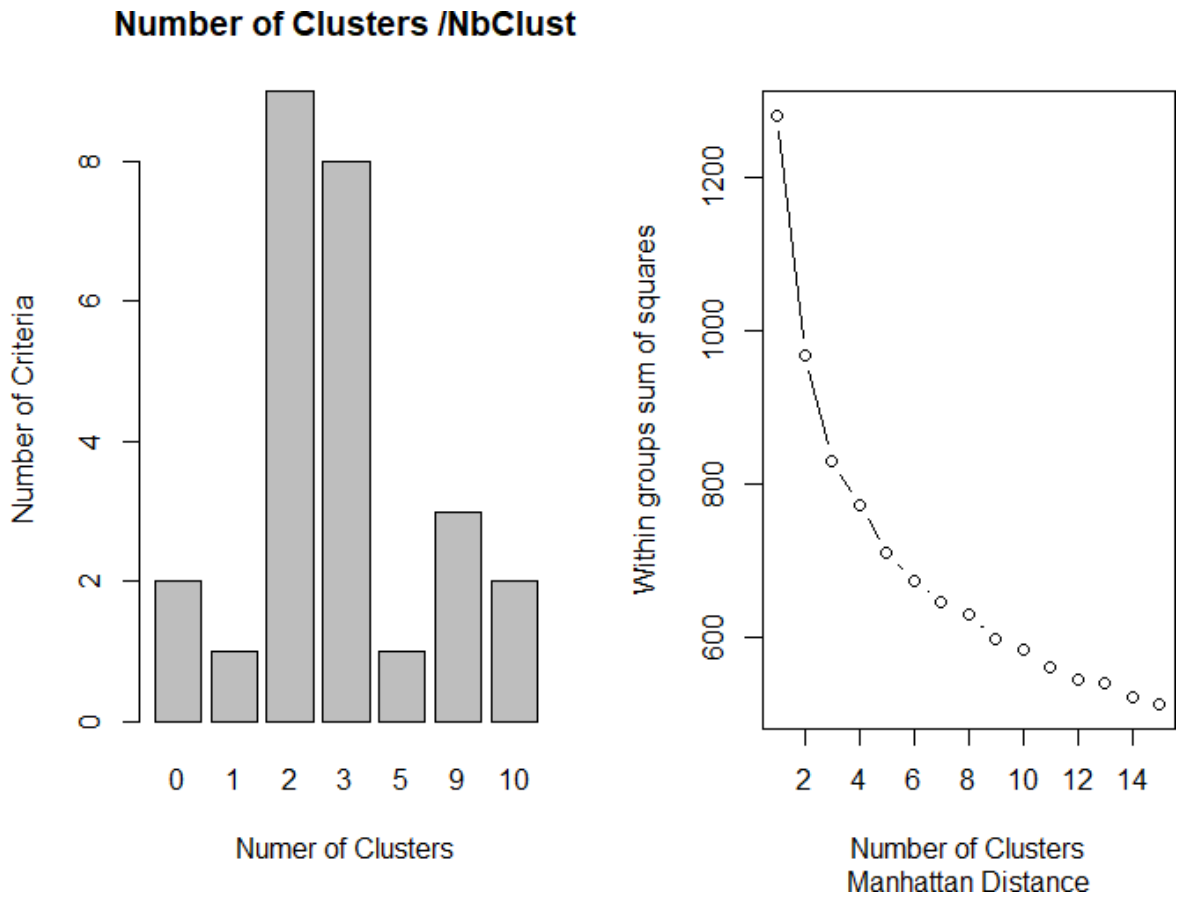
1. Euclidean Distance

```
# 1. Euclidean Distance cluster_Available=NbClust(completly_normalized__data,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")#available clustering table(cluster_Available$Best.n[1,])#creating table barplot(table(clust
xlab="Numer of Clusters", ylab="Number of Criteria", main="Number of Clusters (NbClust)")
cluster_ws <- 0
for (i in 1:15){# for loop for 1 to 15
cluster_ws[i] <- sum(kmeans(completly_normalized data, centers=i)$withinss)

}

plot(1:15,# plotting cluster_ws, type="b",
xlab="Number of Clusters",
ylab="Within groups sum of squares",sub = "Euclidean Distance")
```



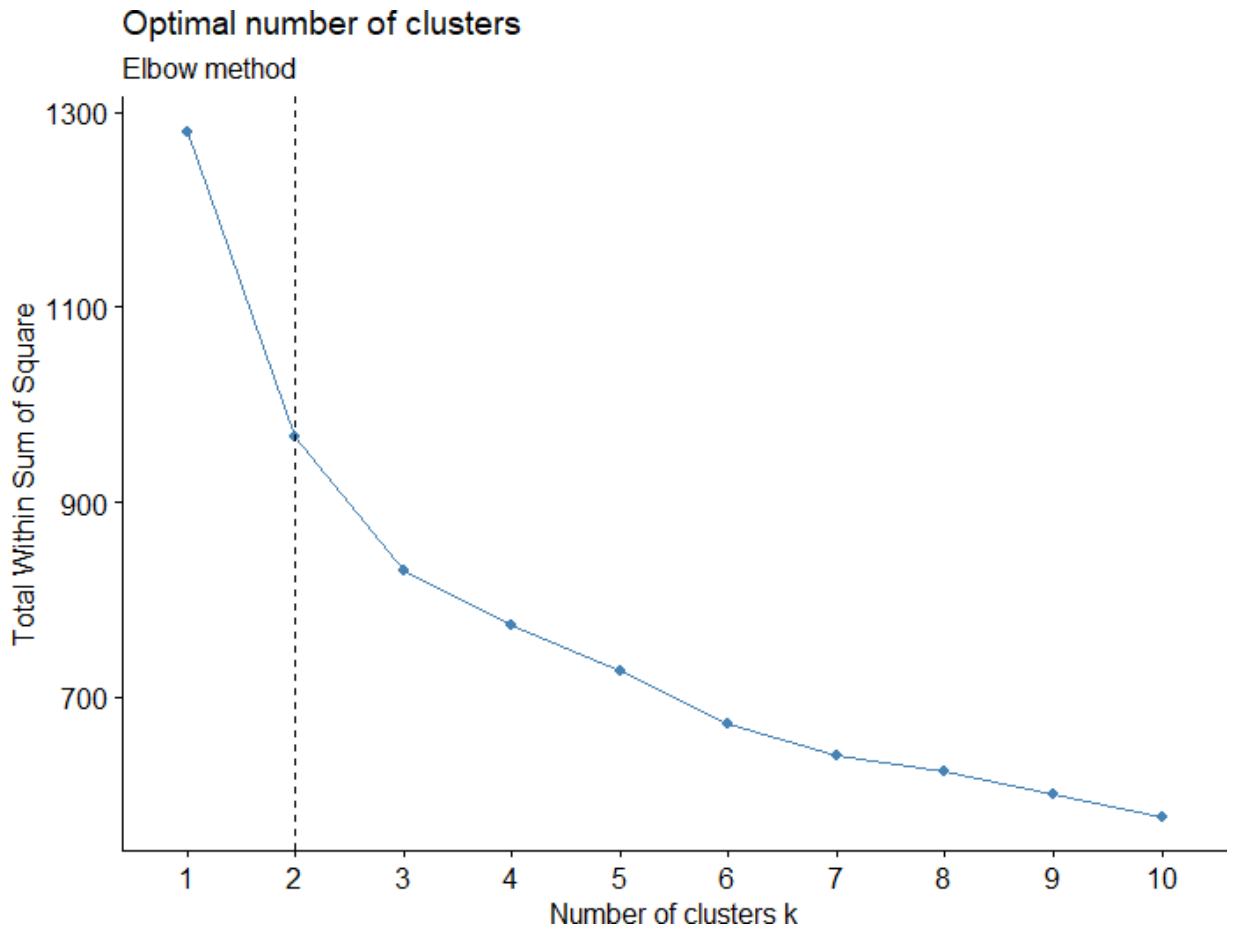
Using Manhattan Distance

```
# 2. Manhattan Distance\
cluster_Available=NbClust(completly_normalized data,distance="manhattan", min.nc=2,max.nc=10,method="kmeans",index="all")#clustering
table(cluster_Available$Best.n[1,])#creating table barplot(table(cluster_Available$Best.n[1,]), #barploting
xlab="Numer of Clusters", ylab="Number of Criteria", main="Number of Clusters /NbClust")
cluster_ws <- 0
for (i in 1:15){#for 1 to 15
cluster_ws[i] <- sum(kmeans(completly_normalized data, centers=i)$withinss)

}
plot(1:15, cluster_ws, type="b",
xlab="Number of Clusters",
ylab="Within groups sum of squares",sub = "Manhattan Distance")
```



Using Elbow method

```
# 3. Elbow method
fviz_nbclust(completly_normalized data, kmeans, method = "wss") + geom_vline(xintercept = 2, linetype = 2) + # adding line for better visua

labs(subtitle = "Elbow method") # adding subtitle for code
```



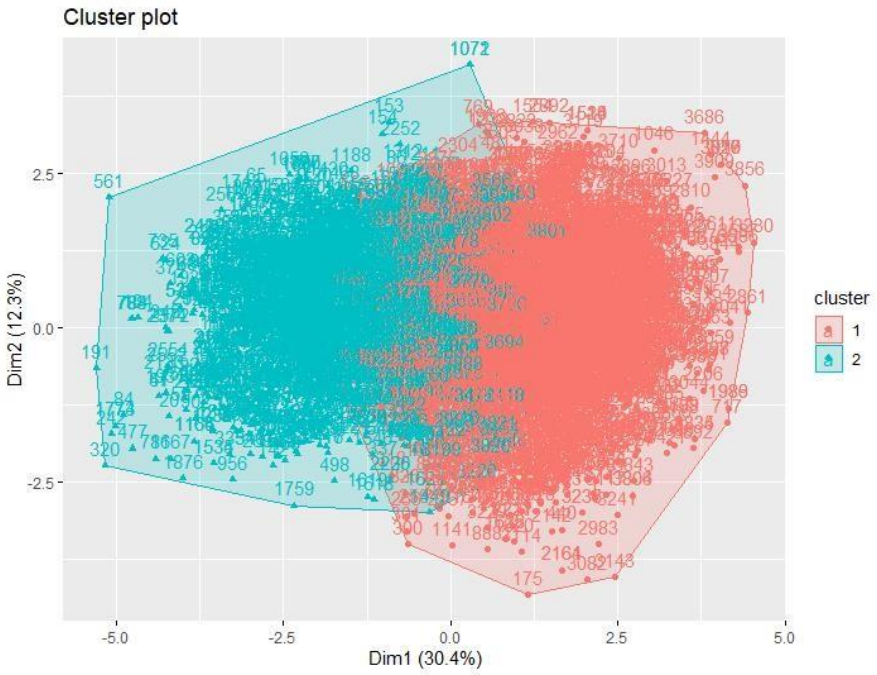
K-means analysis

K-means clustering is one of the most frequent unsupervised learning algorithms that is majorly utilized for clustering problems in the field of machine learning and data mining. The basic clustering problem is to find the groups of which the data could be divided so that a testing data point could be arranged into a correct data cluster according to the groups that are specified by the given data. In partition clustering, the main aim is to divide the given set of data into disjoint subsets so that the precise clustering measures are enhanced.

K-means is a widely used clustering algorithm because it is known to reduce the clustering error that occurs during data clustering. But it also possesses a few drawbacks, as the clustering is completely dependent on the initial stages as to which the K value that is the number of cluster centers should be partitioned.

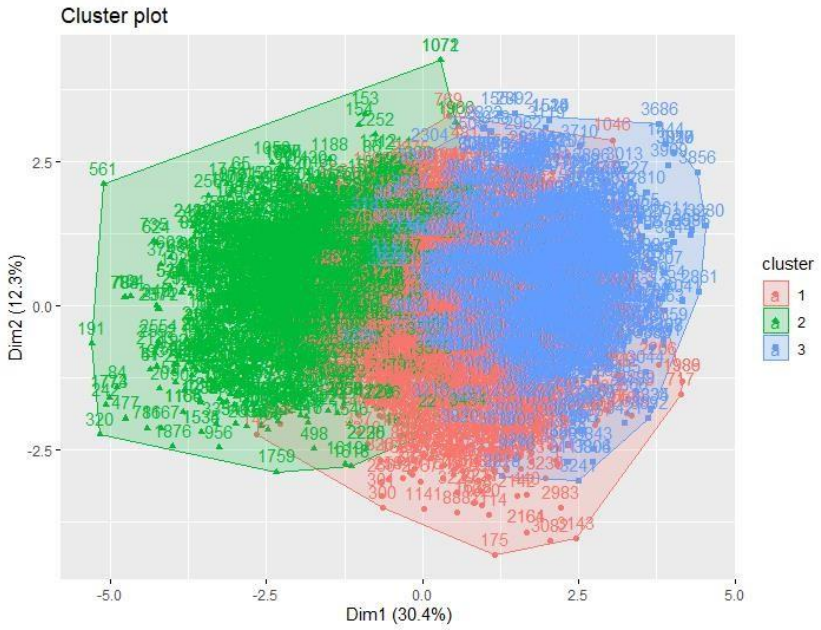
k= 2

```
# k equals 2
custer_2 <- kmeans(completly_normalized data, 2, iter.max = 140 , algorithm="Lloyd", nstart=100)# clustering for k2
custer_2
p.r <- pam(completly_normalized  data, 2, metric = c("euclidean"))# res data # Visualize
fviz_cluster(p.r)# clusterr
```



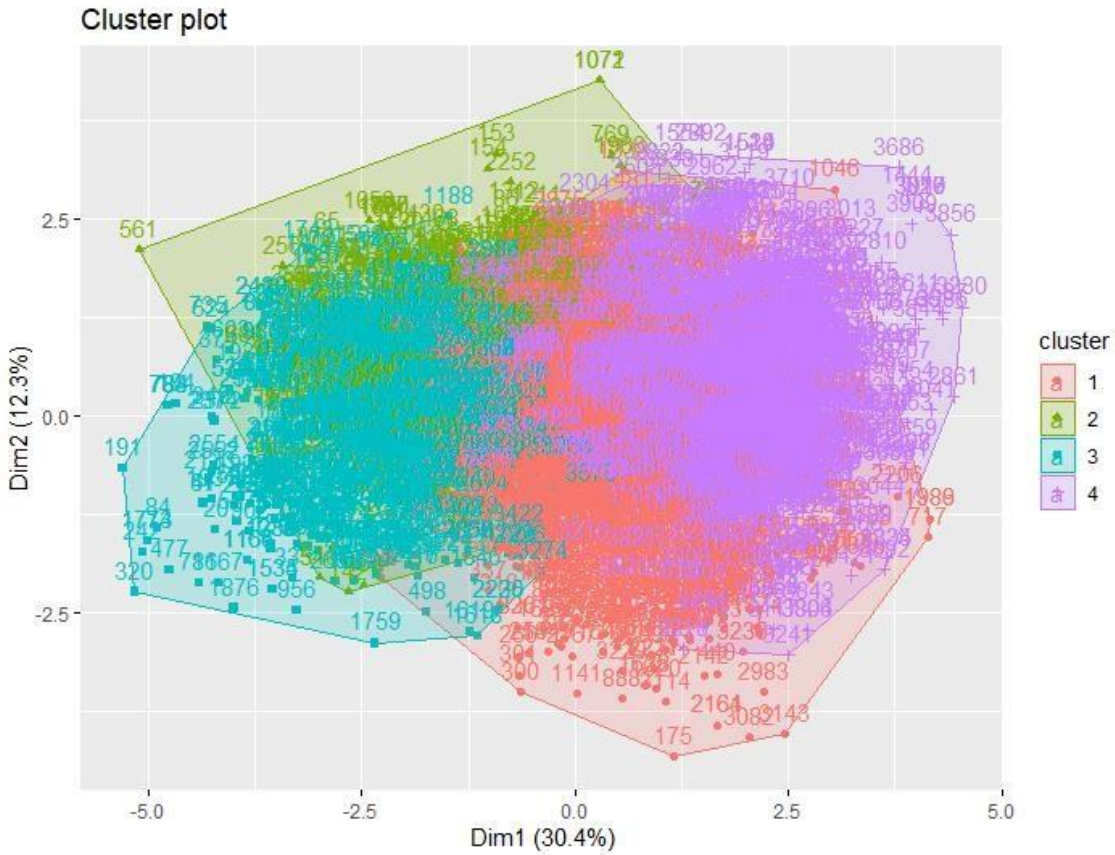
k=3

```
# k equals 3
custer_3 <- kmeans(completly_normalized data, 3, iter.max = 140 , algorithm="Lloyd", nstart=100)# clustering for k3
custer_3
p.r <- pam(completly_normalized  data, 3, metric = c("euclidean"))# res data # Visualize
fviz_cluster(p.r)# clusterr
```



k=4

```
# k equals 4
custer_4 <- kmeans(completly_normalized data, 4, iter.max = 140 , algorithm="Lloyd", nstart=100)# clustering for k4
custer_4
p.r <- pam(completly_normalized data, 4, metric = c("euclidean"))#res data
```



Evaluvtion

Confusion matrix

```
confusion_matrix = table(pca_data_scalled$quality,pca_k2$cluster)#confustion matrix confusion_matrix
#visualization of matrix
```

Confusion matrix and the accuracy result

```
> confusion_matrix = table(pca_data_scalled$quality,pca_k_2_number$cluster)
> confusion_matrix

                        1    2
  0.0601216031010796 2163    0
  1.18733544802705      0 1443
  1.30757865422921      0  858
  2.55503570535733      0  169
> pca_quality = as.numeric(factor(pca_data_scalled$quality ))
> confusionMatrix(as.factor(c(as.factor(pca_k_2_number$cluster))) , as.factor(pca_quality))
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4
         1 2163    0    0    0
         2    0 1443  858  169
         3    0    0    0    0
         4    0    0    0    0

Overall Statistics

               Accuracy : 0.7783
                 95% CI : (0.7661, 0.7902)
    No Information Rate : 0.4669
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6401

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            1.0000   1.0000   0.0000  0.00000
Specificity            1.0000   0.6781   1.0000  1.00000
Pos Pred Value         1.0000   0.5842      NaN      NaN
Neg Pred Value         1.0000   1.0000   0.8148  0.96352
Prevalence             0.4669   0.3115   0.1852  0.03648
Detection Rate         0.4669   0.3115   0.0000  0.00000
Detection Prevalence   0.4669   0.5331   0.0000  0.00000
Balanced Accuracy      1.0000   0.8390   0.5000  0.50000
```

The best accuracy is taken by when k=4 . so **k=4 is the winning cluster.**

PCA Application

```
#PCA section

clean_wine_data_set_pca <- prcomp(clean_wine_data_set[,c(1:11)], center =
TRUE,scale. = TRUE)# getting value for the pca summary(clean_wine_data_set_pca)#summarying

str(clean_wine_data_set_pca)#str

#Removing outliers from the PCA Dataset pca_dset = wine_data_set[9:12]# pca dataset

# outliers for pca dataset is checking here
zscore_for_pca = as.data.frame(sapply(pca_dset, function(pca_dset)(abs(pca_dset - mean(pca_dset))/sd(pca_dset))))#fun
pca_dset_with_removed_outliers = zscore_for_pca[!rowSums(zscore_for_pca>3),]


#visualising pca dataset with outlines removal old_par0 = par(mfrow = c(2,6))
for ( i in 1:12 ) {#for loop for 1 to 12 boxplot(pca_dset_with_removed_outliers[[i]])#boxploting mtext(names(pca_dset
}
par(old_par0)

# Scale the PCA dataset
# main data scalling proccess

pca scalled_data= as.data.frame(pca_dset_with_removed_outliers)#data scalling
#Display the scalled data
pca scalled_data
dim(pca_data_scalled)
####Cluster proccess for the PCA dataset Dataset for K == 2
```

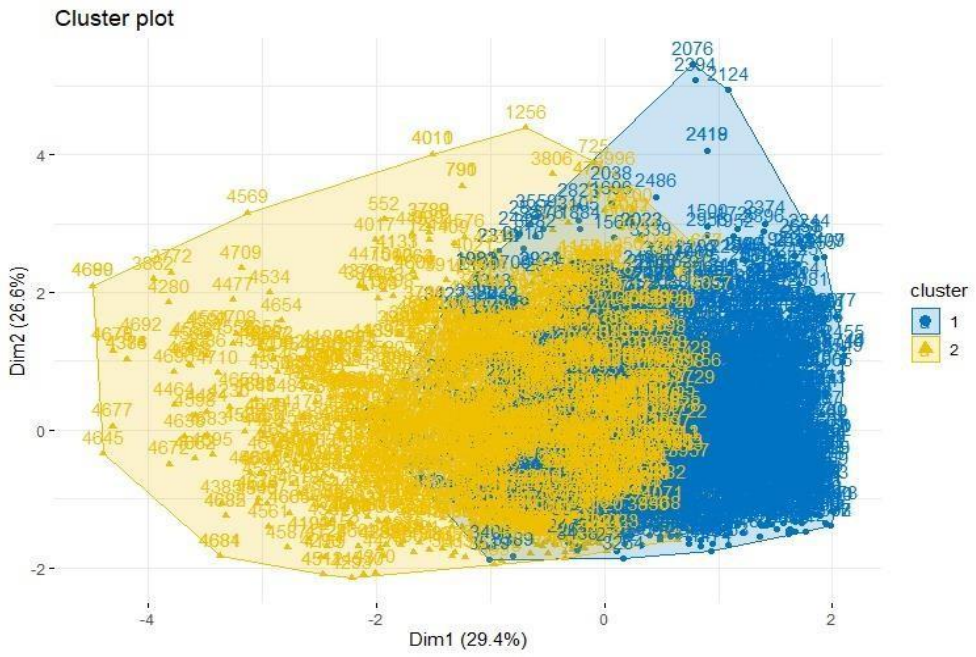
```
set.seed(123)
pca_k2 = kmeans(pca scalled_data, centers = 2 , nstart = 25) pca_k2$cluster

# Centroid Plot against 1st 2 discriminant functions clusplot
#(clean_wine_data_set, km$cluster, color=TRUE, shade=TRUE,labels=2, lines=0)

fviz_cluster(pca_k2, data = pca_data_scalled, ellipse.type = "convex",
```

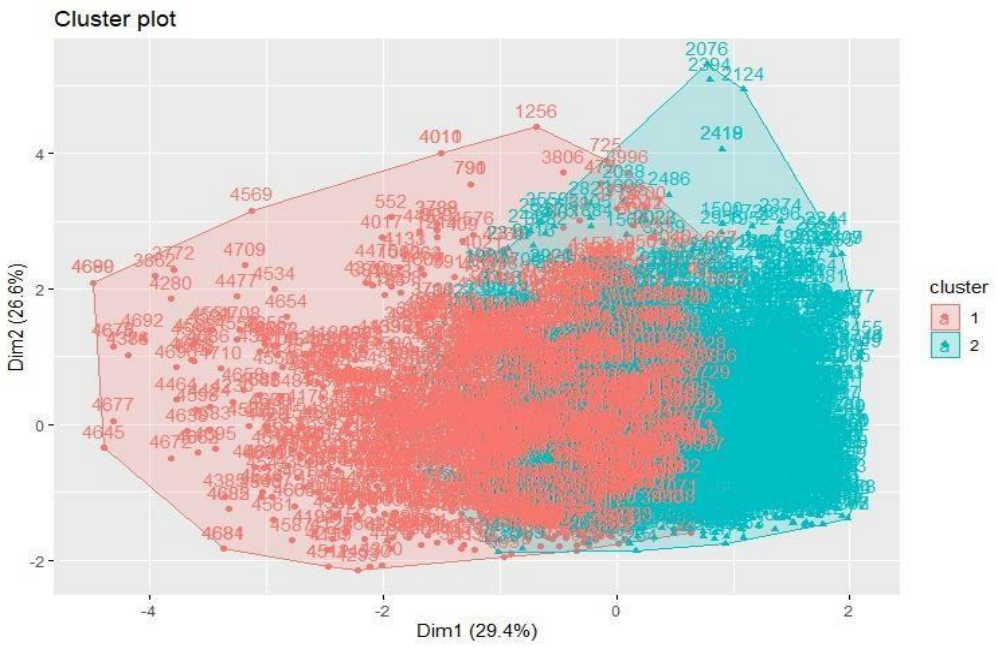
Appling k-means on this new "PCA-based" dataset

```
pca_quality = as.numeric(factor(pca_data_scalled$quality )) confusionMatrix(as.factor(c(as.factor(pca_k2$cluster))) ,
 as.factor(pca_quality))


pca_k2 <- kmeans(pca_data_scalled, 2, iter.max = 140 , algorithm="Lloyd", nstart=100) pca_k2
pca_k2$totss
```

Result ;

```
> #PCA
> clean_wine_dataset.pca <- prcomp(clean_wine_dataset[,c(1:11)], center = TRUE,scale. = TRUE)
> summary(clean_wine_dataset.pca)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11
Standard deviation     1.8716 1.2142 1.0964 1.04634 0.98597 0.87737 0.85471 0.76906 0.61278 0.52373 0.11167
Proportion of Variance 0.3184 0.1340 0.1093 0.09953 0.08838 0.06998 0.06641 0.05377 0.03414 0.02494 0.00113
Cumulative Proportion  0.3184 0.4525 0.5617 0.66126 0.74963 0.81961 0.88603 0.93979 0.97393 0.99887 1.00000
>
```





MPL Part

- The various methods used for defining the input vector in time-series Problems

A. Autoregression
B. Moving Average
C. Autoregressive Moving Average
D. Vector Autoregression
E. Autoregressive Integrated Moving Average

Various adopted input vectors and the related input/output matrices Data pre-processing

Data Pre-processing

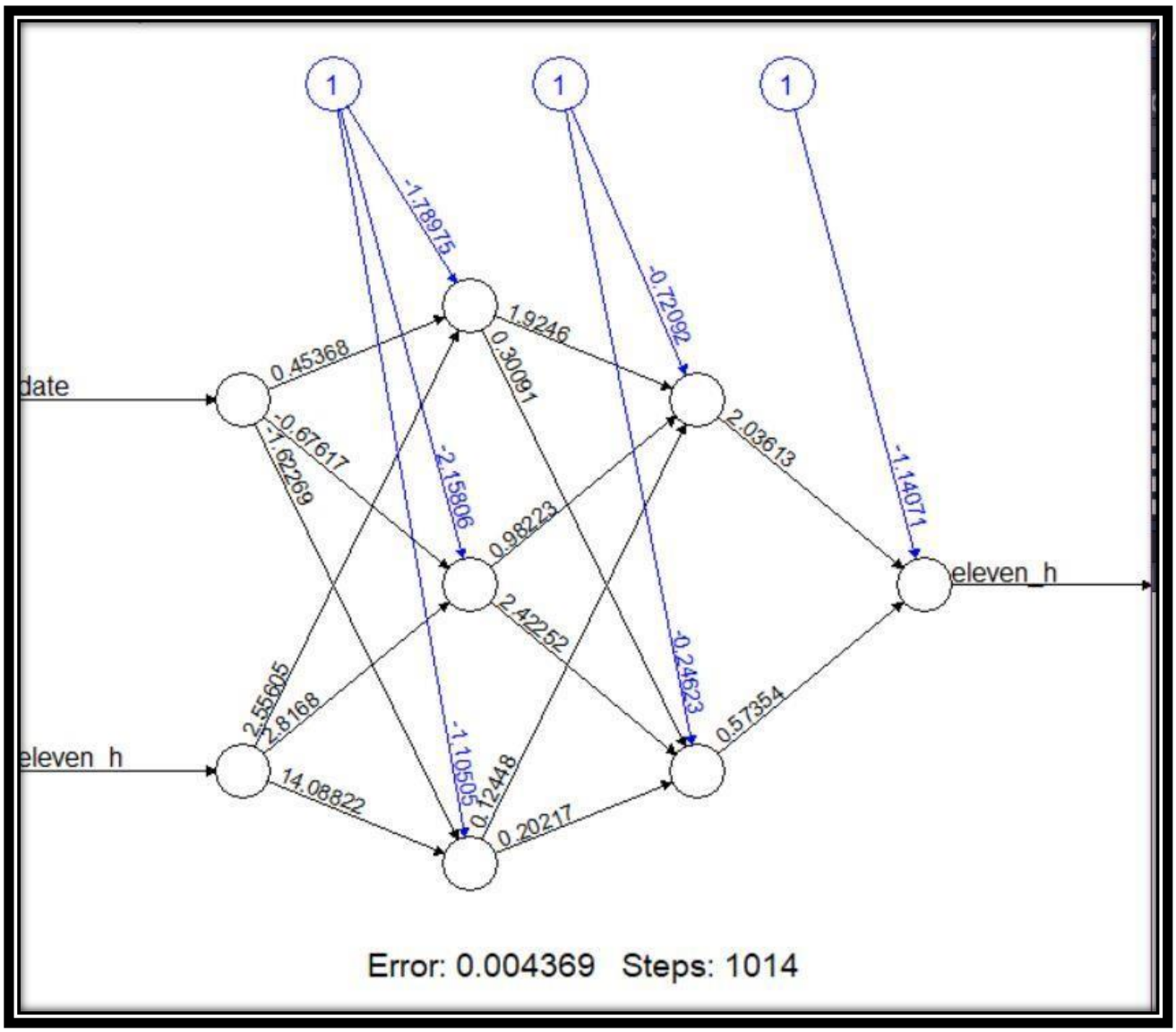
```r
#Data pre-processing
names(uow_data)[2] <- 'seventh'
names(uow_data)[3] <- 'eighthh'
names(uow_data)[4] <- 'nineth'


dates <-factor(uow_load_data_set$Dates) #factoding data
dates <-as.numeric(dates) #numbaric
dates


uow_df <- data.frame(dates, uow_data$'seventh', uow_data$'eightth', uow_data$'nineth')
uow_df

#normalization
uow_norm <- as.data.frame(lapply(uow_df, function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}))

names(uow_norm)[2] <- 'seventh'
names(uow_norm)[3] <- 'eightth'
names(uow_norm)[4] <- 'nineth'


#dividing data into training and testing data set
set.seed(123)
uow_norm_train <- uow_norm[1:430,]
uow_norm_test <- uow_norm[431:500,]
```

Approch of the AR

```r
#Generate Neural Network in AR
UoW_load_data_NNAR <- neuralnet(eleventh ~ dates + eleventh, hidden = c(3,2), data = uow_norm_train, linear.output = TRUE, threshold = 0.01
plot(UoW_load_data_NNAR)

#model performance evaluation
UoW_load_modelPerform<- predict(UoW_load_data_NNAR, UoW_load_testData_normalization)
UoW_load_modelPerform

#Get trained data test dataset without normalization
UoW_load_trainData <- UoW_load_data[1:430,"nineth"]
UoW_load_testData <- UoW_load_data[431:500,"nineth"]

#find minimum and maximum values of the train data set
trained_min <- min(UoW_load_trainData)
trained_max <- max(UoW_load_testData)

#un-normalizing the data

unNormalized <- function(x, min, max) {
  return ((max - min)*x + min)
}

predUnnorm <- unNormalized(UoW_load_modelPerform, trained_min, trained_max)
predUnnorm

#Testing performance with RMSE
RMSE(exp(predUnnorm), UoW_load_testData$nineth)

#testing performance with MSE
MSE(exp(predUnnorm), UoW_load_testData$nineth)
```

```r
#testing performance with MAPE
MAPE(exp(predUnnorm), UoW_load_testData$nineth)

#correlation between predicted and actual values
cor(predUnnorm, UoW_load_testData$nineth)

#generate nineth hour plot

par(mfrow = c(1,1))

plot(UoW_load_testData$nineth, UoW_load_predicted_unNormalized, col = "red", main = "Unnormalized Prediction Graph AR", pch = 18, cex = 0.7
abline(0, 1, lwd = 2)
legend("bottomright", legend = "NN", pch = 18, col = "red", bty = "m")

UoW_load_finalResult <- cbind(UoW_load_testData, UoW_load_predicted_unNormalized)
UoW_load_finalResult

plot(UoW_load_testData$eleventh, ylab = "Predicted vs Expected AR", type = "l", col = "yellow")
par(new = TRUE)

plot(UoW_load_predicted_unNormalized, ylab = "", yaxt = "m", type = "l", col = "red", main = "Predicted val vs Expected val AR")
legend("lopleft", c("Expected", "Predicted"), fill = c("yellow", "red")) #ERROR

#Calculate Accuracy
UowDataset_predicted = UoW_load_modelPerform * abs(diff(range(UoW_load_testData_normalization$eleventh))) + min(UoW_load_testData_normaliza

UowDataset_actual = UoW_load_testData_normalization$eleventh * abs(diff(range(UoW_load_testData_normalization$eleventh))) + min(UoW_load_te

predict_actual_comparison = data.frame(UowDataset_predicted, UowDataset_actual) #Remove

Data_deviation = ((UowDataset_actual - UowDataset_predicted) / UowDataset_actual)
```

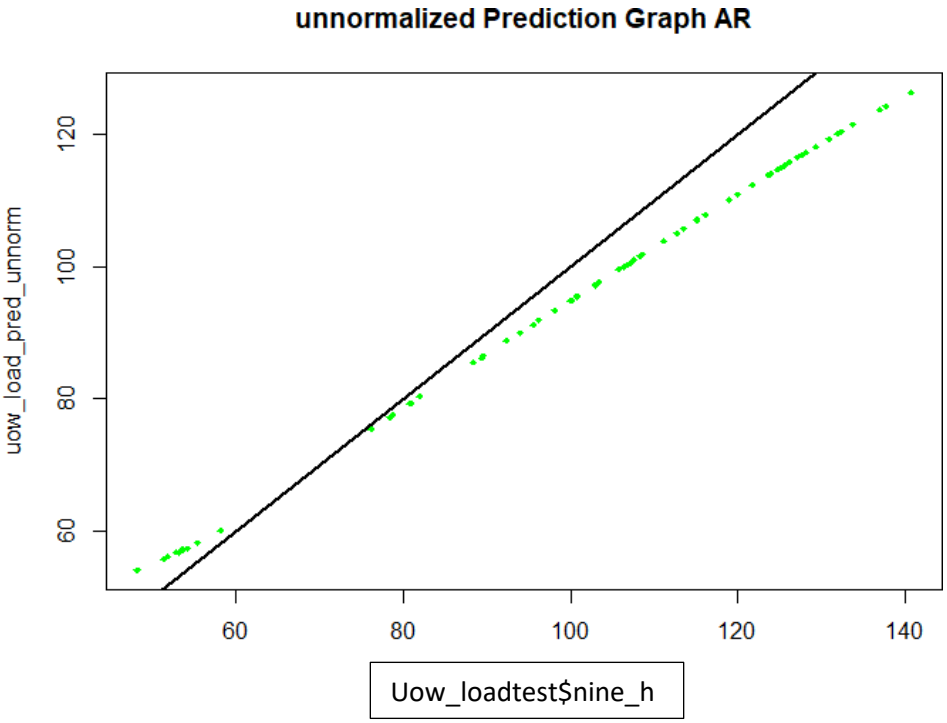
```r
Data_deviation_OmitNA <- na.omit(Data_deviation)
Data_deviation_OmitNA

comparison = data.frame(UowDataset_predicted, UowDataset_actual, Data_deviation)
Accuracy_level = 1 - abs(mean(Data_deviation_OmitNA))
Accuracy_level
```

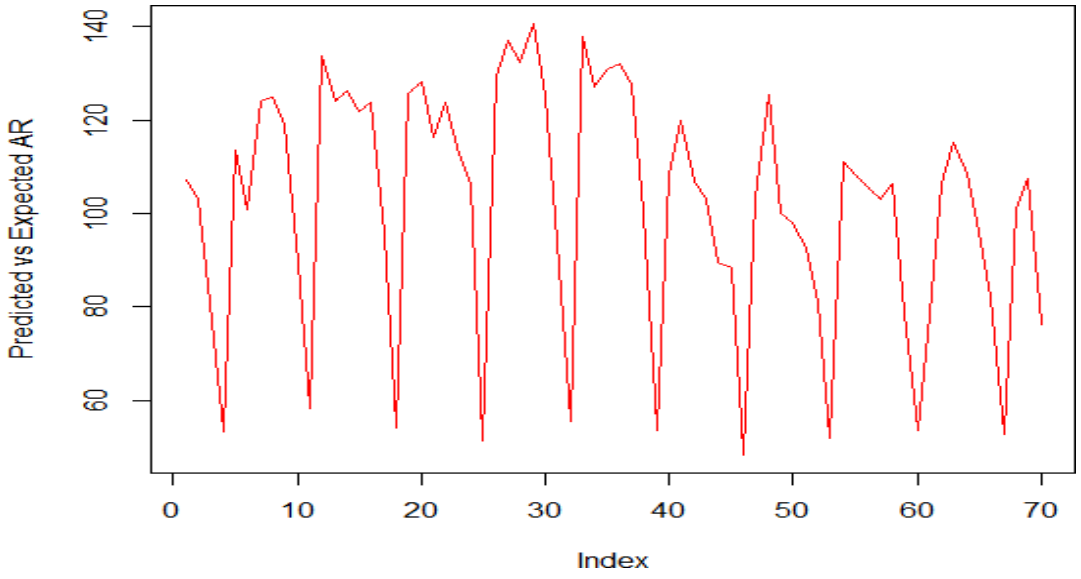

1
1
1
-1.78975
-0.72092
1.9246
0.30091
0.45368
date
-0.67617
-1.62269
-2.15806
0.98223
2.03613
-1.14071
eleven_h
2.42252
-0.24623
2.55605
2.8168
0.57354
eleven_h
-1.10505
0.12448
14.08822
0.20217
Error: 0.004369 Steps: 1014

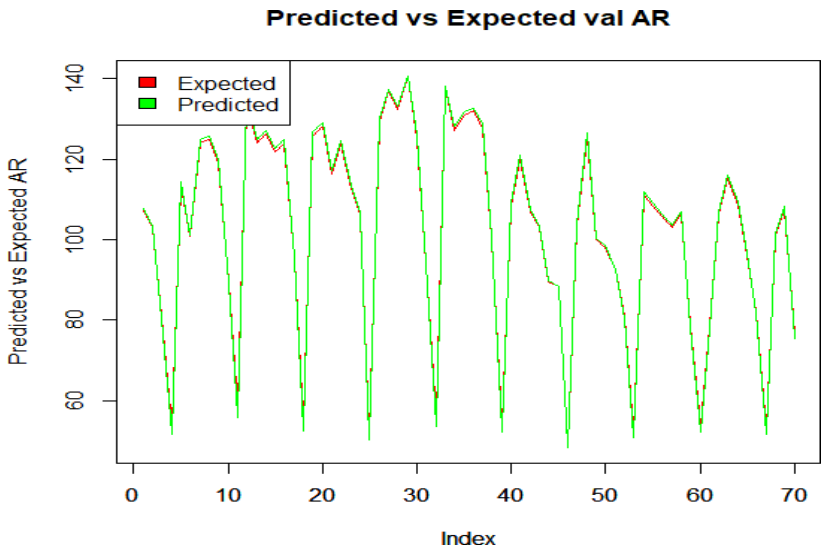
Approch of the NARX

```
#_____________________
#NARX Approach
#_____________________

#Generate Neural Network in AR

#UoW_load_NNAR <- neuralnet(t_eleven ~ day + t_eleven,hidden = c(3,2), data = UoW_load_trainData_normalization, linear.output = TRUE, thres
UoW_load_narx <- neuralnet(eleventh ~ dates + nineth + tenth + eleventh , hidden = c(3,2), data = UoW_load_trainData_normalization, linear.
plot(UoW_load_narx)

#model performance evaluation
UoW_load_RX_modelPerform <- predict(UoW_load_narx, UoW_load_testData_normalization)
UoW_load_RX_modelPerform

UoW_load_predicted_NARXunNormalized <- unNormalized(UoW_load_RX_modelPerform, trained_min, trained_max)
UoW_load_predicted_NARXunNormalized

#testing performance with RMSE
RMSE(exp(UoW_load_predicted_NARXunNormalized), UoW_load_testData$eleventh)

#testing performance with MSE
MSE(exp(UoW_load_predicted_NARXunNormalized), UoW_load_testData$eleventh)

#testing performance with MAPE
MAPE(exp(UoW_load_predicted_NARXunNormalized), UoW_load_testData$eleventh)


#correlation between predicted and actual values
cor(UoW_load_predicted_NARXunNormalized, UoW_load_testData$eleventh)
```

```
#Generate 11th hour plot in NARX
par(mfrow = c(1,11))

plot(UoW_load_testData$eleventh, UoW_load_predicted_NARXunNormalized, col = "red", main = "Unnormalized Prediction Grpah", pch = 18, cex =
abline(0, 1, lwd = 2)

UoW_load_finalNARX <- cbind(UoW_load_testData, UoW_load_predicted_NARXunNormalized)
UoW_load_finalNARX

plot(UoW_load_testData$t_eleven, ylab = '', yaxt = 'm', type = 'l', col = "red", main = 'Predict val vs Ecpected val')
legend("lopleft", c("Expected", "Predicted"), fill = c("red", "green")) #ERROR

#calculate accuracy
RX_predicted_values = UoW_load_RX_modelPerform * abs(diff(range(UoW_load_testData_normalization$eleventh))) + min(UoW_load_testData_normali
RX_actual_values = UoW_load_testData_normalization$eleventh * abs(diff(range(UoW_load_testData_normalization$eleventh))) + min(UoW_load_tes

RX_comparison = data.frame(RX_predicted_values, RX_actual_values)

RX_deviation = ((RX_actual_values - RX_predicted_values) / RX_actual_values)
RX_deviation
is.na(RX_deviation) <- sapply(RX_deviation, is.infinite)
RX_deviation

RX_deviation_omitNA <- na.omit(RX_deviation)
RX_deviation_omitNA

RX_comparison = data.frame(RX_predicted_values, RX_actual_values, RX_deviation)
RX_accuray = 1 - abs(mean(RX_deviation_omitNA))
RX_accuray
```


unnormalized Prediction Graph AR
uow_load_pred_unnorm
60
80
100
120
60
80
100
120
140
Uow_loadtest$nine_h


Predicted vs Expected AR
60
80
100
120
140
0
10
20
30
40
50
60
70
Index



Normalization.

During the normalization process, some characteristics are segregated into individual divisions. As a result, all attribute values must be transformed to a single unit, such as a percentage. This is called as normalization, and it is an important factor of a neural network's inputs.

```
#normalization
uow_norm <- as.data.frame(lapply(uow_df, function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}))

names(uow_norm)[2] <- 'seventh'
names(uow_norm)[3] <- 'eightth'
names(uow_norm)[4] <- 'nineth'


#dividing data into training and testing data set
set.seed(123)
uow_norm_train <- uow_norm[1:430,]
uow_norm_test <- uow_norm[431:500,]
```

Source Codes.

Appendix A.

```
library(caret) #for data preparation, model building, and model evaluation
library(leaps)# for ditacting most influential predictors for our model
library(tidyverse)# this is opinionated collection
library(reshape2)
library(ggplot2)# for visuaizations
library(ggcorrplot)# for visualizations library(MASS)
library(plotmo)# plot model surfaces library(corrplot)#detecting hidden patterns
among variables library(kableExtra)#building common complex tables library(keras)
library(psych)
library(modelr)#creating elegant pipelines when modelling library(gridExtra)#
working with the graphic objects library(Rmisc)#dtata anaizis and utility
operations library(rpart)# building classifications
library(scales)# scalling the data library(cluster)
library(yardstick)# estimate model prorformances library(factoextra)# visualizing
the output library(NbClust)#to determine relevent cluster in a data set
library(readxl) # reading file of the format with a xlsx  or xls  file into R


wine_data_set <- read_excel("d:/life/objective 1/Whitewine_v2.xlsx")

#step 1 =Scaling & Outlier Removal (free processing of the data)
#accoding to 1.5* rule outliering and remove those data
outliers = c()
for (i in 1:11 ) {# for loop for 1 to 11
stats = boxplot.stats(wine_data_set[[i]])$stats#satatus
rows_of_bottom_outlier = which(wine_data_set[[i]] < stats[1]) #rows for bottom
outlier

rows_of_top_outlier = which(wine_data_set[[i]] > stats[5]) #rows for top outlier
outliers = c(outliers ,rows_of_top_outlier[ !rows_of_top_outlier %in% outliers ]
)#

outliers
outliers = c(outliers , rows_of_bottom_outlier[ !rows_of_bottom_outlier %in%
outliers
] )#outliers
}


clean_wine_data_set = wine_data_set[-outliers, ]# wine data set oldpar =
par(mfrow = c(2,6))#older path detecting
```

```
for (i in 1:12 ) {# for loop 1 to 12
truehist(clean_wine_data_set[[i]], xlab = names(clean_wine_data_set)[i], col =
'red', #clearing
}

main = paste("Average =", signif(mean(clean_wine_data_set[[i]]),3)))

par(oldpar)

dim(clean_wine_data_set) #after the removel of outliers reduces the size of
dataset head(clean_wine_data_set)# heading




#dta type coverting

# normalize the large data
normalize_data <- sapply(clean_wine_data_set[,c(1,4,6,7,9,11,12)], function(x) (x
- min(x))/(max(x) - min(x)))#normalization
normalize_data <- data.frame(normalize_data)#normalization head(normalize_data)#
heading

completly_normalized data <-
cbind(clean_wine_data_set[,c(2,3,5,8,10)],normalize_data)# completed normalized
data
head(completly_normalized data)# heading


#number of cluster
# used NbClust for finding number of clusters

# 1. Euclidean Distance
cluster_Available=NbClust(completly_normalized__data,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")#available clustering
table(cluster_Available$Best.n[1,])#creating table
barplot(table(cluster_Available$Best.n[1,]), # baplot
xlab="Numer of Clusters", ylab="Number of Criteria", main="Number of Clusters
(NbClust)")
cluster_ws <- 0
for (i in 1:15){# for loop for 1 to 15
cluster_ws[i] <- sum(kmeans(completly_normalized data, centers=i)$withinss)

}
```

```
plot(1:15,# plotting cluster_ws, type="b",
xlab="Number of Clusters",
ylab="Within groups sum of squares",sub = "Euclidean Distance")


# 2. Manhattan Distance\
cluster_Available=NbClust(completly_normalized data,distance="manhattan",
min.nc=2,max.nc=10,method="kmeans",index="all")#clustering
table(cluster_Available$Best.n[1,])#creating table
barplot(table(cluster_Available$Best.n[1,]), #barploting
xlab="Numer of Clusters", ylab="Number of Criteria", main="Number of Clusters
/NbClust")
cluster_ws <- 0
for (i in 1:15){#for 1 to 15
cluster_ws[i] <- sum(kmeans(completly_normalized data, centers=i)$withinss)

}
plot(1:15, cluster_ws, type="b",
xlab="Number of Clusters",
ylab="Within groups sum of squares",sub = "Manhattan Distance")


# 3. Elbow method
fviz_nbclust(completly_normalized data, kmeans, method = "wss") +
geom_vline(xintercept = 2, linetype = 2) + # adding line for better visualization

labs(subtitle = "Elbow method") # adding subtitle for code


# 4. Silhouette method
fviz_nbclust(completly_normalized data, kmeans, method = "silhouette")+
labs(subtitle = "Silhouette method")# 4. Silhouette method

# clustering section

# k equals 2
custer_2 <- kmeans(completly_normalized data, 2, iter.max = 140 ,
algorithm="Lloyd", nstart=100)# clustering for k2
custer_2
p.r <- pam(completly_normalized  data, 2, metric = c("euclidean"))# res data #
Visualize
fviz_cluster(p.r)# clusterr


# k equals 3
```

```
custer_3 <- kmeans(completly_normalized data, 3, iter.max = 140 ,
algorithm="Lloyd", nstart=100)# clustering for k3
custer_3
p.r <- pam(completly_normalized  data, 3, metric = c("euclidean"))# res data #
Visualize
fviz_cluster(p.r)# clusterr


# k equals 4
custer_4 <- kmeans(completly_normalized data, 4, iter.max = 140 ,
algorithm="Lloyd", nstart=100)# clustering for k4
custer_4
p.r <- pam(completly_normalized data, 4, metric = c("euclidean"))#res data

# Visualization of the data fviz_cluster(p.r)

#PCA section

clean_wine_data_set_pca <- prcomp(clean_wine_data_set[,c(1:11)], center =
TRUE,scale. = TRUE)# getting value for the pca
summary(clean_wine_data_set_pca)#summarying

str(clean_wine_data_set_pca)#str

#Removing outliers from the PCA Dataset pca_dset = wine_data_set[9:12]# pca
dataset

# outliers for pca dataset is checking here
zscore_for_pca = as.data.frame(sapply(pca_dset, function(pca_dset)(abs(pca_dset -
mean(pca_dset))/sd(pca_dset))))#function for data frame
pca_dset_with_removed_outliers = zscore_for_pca[!rowSums(zscore_for_pca>3),]


#visualising pca dataset with outlines removal old_par0 = par(mfrow = c(2,6))
for ( i in 1:12 ) {#for loop for 1 to 12
boxplot(pca_dset_with_removed_outliers[[i]])#boxploting
mtext(names(pca_dset_with_removed_outliers)[i], cex = 0.8, side = 1, line = 2)
}
par(old_par0)

# Scale the PCA dataset
# main data scalling proccess

pca scalled_data= as.data.frame(pca_dset_with_removed_outliers)#data scalling
#Display the scalled data
```

```
pca scalled_data
dim(pca_data_scalled)
####Cluster proccess for the PCA dataset Dataset for K == 2


set.seed(123)
pca_k2 = kmeans(pca scalled_data, centers = 2 , nstart = 25) pca_k2$cluster

# Centroid Plot against 1st 2 discriminant functions clusplot
#(clean_wine_data_set, km$cluster, color=TRUE, shade=TRUE,labels=2, lines=0)

fviz_cluster(pca_k2, data = pca_data_scalled, ellipse.type = "convex",
palette = "jco",
ggtheme = theme_minimal())

fviz_cluster(list(data = pca_data_scalled, cluster = km$cluster),

ellipse.type = "norm", geom = "point", stand = FALSE, palette = "jco", ggtheme =
theme_classic())

pam.pca_res <- pam(pca_data_scalled, 2) # Visualize
fviz_cluster(pam.pca_res)

confusion_matrix = table(pca_data_scalled$quality,pca_k2$cluster)#confustion
matrix confusion_matrix
#visualization of matrix

pca_quality = as.numeric(factor(pca_data_scalled$quality ))
confusionMatrix(as.factor(c(as.factor(pca_k2$cluster))) ,
 as.factor(pca_quality))


pca_k2 <- kmeans(pca_data_scalled, 2, iter.max = 140 , algorithm="Lloyd",
nstart=100) pca_k2
pca_k2$totss




pca_dset_with_removed_outliers
```

Appendix B

```
library(readxl)
#install.packages("forecast")
```

```
library(forecast)
library(neuralnet)
library(caret)
(MLmetrics)

#importing uow_load dataset
uow_load_data_set <-read_excel('d:/life/objective2/UoW_load.xlsx')


#Data pre-processing
names(uow_data)[2] <- 'seventh'
names(uow_data)[3] <- 'eighthh'
names(uow_data)[4] <- 'nineth'


dates <-factor(uow_load_data_set$Dates) #factoding data
dates <-as.numeric(dates) #numbaric
dates


uow_df <- data.frame(dates, uow_data$'seventh', uow_data$'eightth',
uow_data$'nineth')
uow_df

#normalization
uow_norm <- as.data.frame(lapply(uow_df, function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}))

names(uow_norm)[2] <- 'seventh'
names(uow_norm)[3] <- 'eightth'
names(uow_norm)[4] <- 'nineth'


#dividing data into training and testing data set
set.seed(123)
uow_norm_train <- uow_norm[1:430,]
uow_norm_test <- uow_norm[431:500,]

#_____________________
#AR Approach
#_____________________

#Generate Neural Network in AR
UoW_load_data_NNAR <- neuralnet(eleventh ~ dates + eleventh, hidden = c(3,2),
data = uow_norm_train, linear.output = TRUE, threshold = 0.01)
```

```r
plot(UoW_load_data_NNAR)

#model performance evaluation
UoW_load_modelPerform<- predict(UoW_load_data_NNAR,
UoW_load_testData_normalization)
UoW_load_modelPerform

#Get trained data test dataset without normalization
UoW_load_trainData <- UoW_load_data[1:430,"nineth"]
UoW_load_testData <- UoW_load_data[431:500,"nineth"]

#find minimum and maximum values of the train data set
trained_min <- min(UoW_load_trainData)
trained_max <- max(UoW_load_testData)

#un-normalizing the data

unNormalized <- function(x, min, max) {
  return ((max - min)*x + min)
}

predUnnorm <- unNormalized(UoW_load_modelPerform, trained_min, trained_max)
predUnnorm

#Testing performance with RMSE
RMSE(exp(predUnnorm), UoW_load_testData$nineth)

#testing performance with MSE
MSE(exp(predUnnorm), UoW_load_testData$nineth)

#testing performance with MAPE
MAPE(exp(predUnnorm), UoW_load_testData$nineth)

#correlation between predicted and actual values
cor(predUnnorm, UoW_load_testData$nineth)

#generate nineth hour plot

par(mfrow = c(1,1))

plot(UoW_load_testData$nineth, UoW_load_predicted_unNormalized, col = "red", main
= "Unnormalized Prediction Graph AR", pch = 18, cex = 0.7)
abline(0, 1, lwd = 2)
legend("bottomright", legend = "NN", pch = 18, col = "red", bty = "m")
```

```
UoW_load_finalResult <- cbind(UoW_load_testData, UoW_load_predicted_unNormalized)
UoW_load_finalResult

plot(UoW_load_testData$eleventh, ylab = "Predicted vs Expected AR", type = "l",
col = "yellow")
par(new = TRUE)

plot(UoW_load_predicted_unNormalized, ylab = "", yaxt = "m", type = "l", col =
"red", main = "Predicted val vs Expected val AR")
legend("lopleft", c("Expected", "Predicted"), fill = c("yellow", "red")) #ERROR

#Calculate Accuracy
UowDataset_predicted = UoW_load_modelPerform *
abs(diff(range(UoW_load_testData_normalization$eleventh))) +
min(UoW_load_testData_normalization$eleventh)

UowDataset_actual = UoW_load_testData_normalization$eleventh *
abs(diff(range(UoW_load_testData_normalization$eleventh))) +
min(UoW_load_testData_normalization$eleventh)

predict_actual_comparison = data.frame(UowDataset_predicted, UowDataset_actual)
#Remove

Data_deviation = ((UowDataset_actual - UowDataset_predicted) / UowDataset_actual)
Data_deviation
is.na(Data_deviation) <- sapply(Data_deviation, is.infinite)
Data_deviation

Data_deviation_OmitNA <- na.omit(Data_deviation)
Data_deviation_OmitNA

comparison = data.frame(UowDataset_predicted, UowDataset_actual, Data_deviation)
Accuracy_level = 1 - abs(mean(Data_deviation_OmitNA))
Accuracy_level

#_____________________
#NARX Approach
#_____________________

#Generate Neural Network in AR

#UoW_load_NNAR <- neuralnet(t_eleven ~ day + t_eleven,hidden = c(3,2), data =
UoW_load_trainData_normalization, linear.output = TRUE, threshold = 0.01)
```

```
UoW_load_narx <- neuralnet(eleventh ~ dates + nineth + tenth + eleventh , hidden
= c(3,2), data = UoW_load_trainData_normalization, linear.output = TRUE,
threshold = 0.01)
plot(UoW_load_narx)

#model performance evaluation
UoW_load_RX_modelPerform <- predict(UoW_load_narx,
UoW_load_testData_normalization)
UoW_load_RX_modelPerform

UoW_load_predicted_NARXunNormalized <- unNormalized(UoW_load_RX_modelPerform,
trained_min, trained_max)
UoW_load_predicted_NARXunNormalized

#testing performance with RMSE
RMSE(exp(UoW_load_predicted_NARXunNormalized), UoW_load_testData$eleventh)

#testing performance with MSE
MSE(exp(UoW_load_predicted_NARXunNormalized), UoW_load_testData$eleventh)

#testing performance with MAPE
MAPE(exp(UoW_load_predicted_NARXunNormalized), UoW_load_testData$eleventh)


#correlation between predicted and actual values
cor(UoW_load_predicted_NARXunNormalized, UoW_load_testData$eleventh)

#Generate 11th hour plot in NARX
par(mfrow = c(1,11))

plot(UoW_load_testData$eleventh, UoW_load_predicted_NARXunNormalized, col =
"red", main = "Unnormalized Prediction Grpah", pch = 18, cex = 0.7)
abline(0, 1, lwd = 2)

UoW_load_finalNARX <- cbind(UoW_load_testData,
UoW_load_predicted_NARXunNormalized)
UoW_load_finalNARX

plot(UoW_load_testData$t_eleven, ylab = '', yaxt = 'm', type = 'l', col = "red",
main = 'Predict val vs Ecpected val')
legend("lopleft", c("Expected", "Predicted"), fill = c("red", "green")) #ERROR

#calculate accuracy
```

```
RX_predicted_values = UoW_load_RX_modelPerform *
abs(diff(range(UoW_load_testData_normalization$eleventh))) +
min(UoW_load_testData_normalization$eleventh)
RX_actual_values = UoW_load_testData_normalization$eleventh *
abs(diff(range(UoW_load_testData_normalization$eleventh))) +
min(UoW_load_testData_normalization$eleventh)

RX_comparison = data.frame(RX_predicted_values, RX_actual_values)

RX_deviation = ((RX_actual_values - RX_predicted_values) / RX_actual_values)
RX_deviation
is.na(RX_deviation) <- sapply(RX_deviation, is.infinite)
RX_deviation

RX_deviation_omitNA <- na.omit(RX_deviation)
RX_deviation_omitNA

RX_comparison = data.frame(RX_predicted_values, RX_actual_values, RX_deviation)
RX_accuray = 1 - abs(mean(RX_deviation_omitNA))
RX_accuray
```