INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

5DATA001C.2 Machine Learning & Data Mining

Module leader : Aponso Achala Chathuranga

Informatics Institute of Technology
Department of Computing (BSc)
in Software Engineering

Academic details of Student

Name : Kaliyugavarathan Sean Jesuharun

UOW ID : W1833520

Student ID : 20200696

Table of Content

List of Tables

List of Figures

Clustering Part

1. Data preprocessing tasks

This part consists of 2 tasks one is outlier removal and the other one is scaling. So here outlier removal will be done first because if we scale the data with outliers even if the output shows the values between 0 and 1(min-max normalization) still the range of these values will be affected by these outliers if some large or even small value is present in each feature. So removing the outliers before the scaling is always the best option.

1. Outlier removal

By plotting boxplots for all the features we can view the outliers present in the dataset.

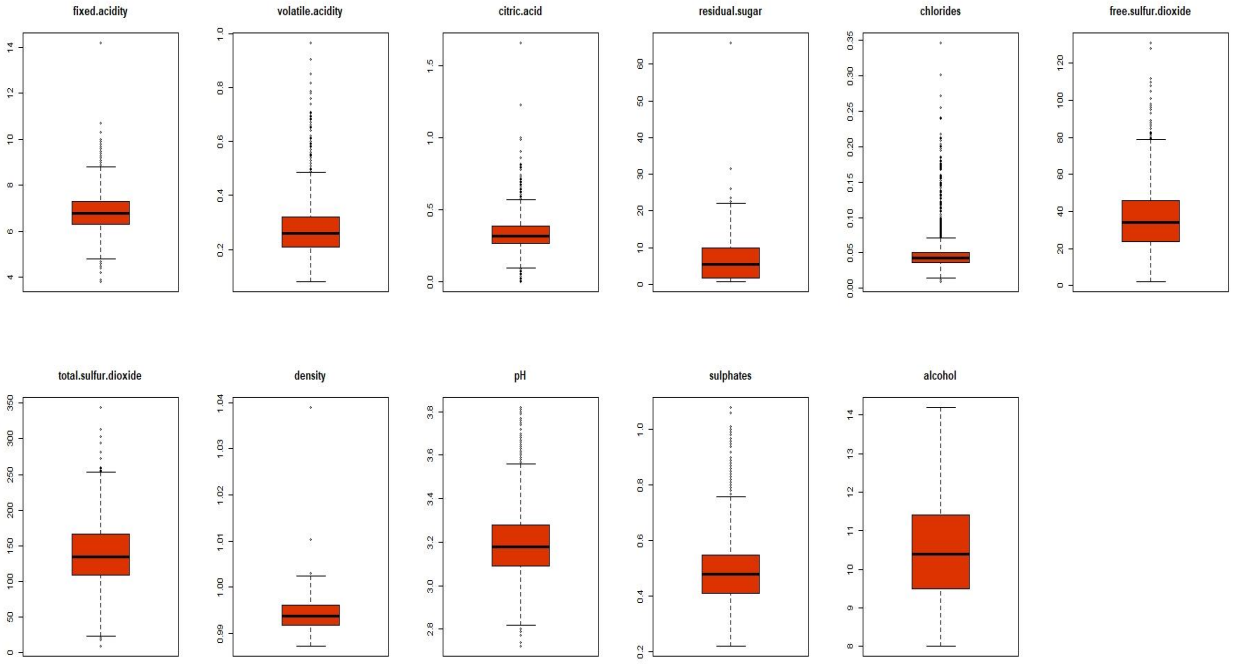

Figure 1: Viewing outliers

You can clearly see that all the features have outliers except alcohol. So they need to be removed from the dataset. In order to remove them we will use the extreme of the lower whisker (1/4th quartile - 1.4 * IQR) and the extreme of the upper whisker (3/4th quartile + 1.4 * IQR) values. Anything that is lower than the extreme of the lower whisker value and anything that is higher than the extreme

of the upper whisker values of each feature will be removed from the dataset. Initially in the dataset we had 4710 observations and after extreme outlier removal we now had 3747 observations. By plotting the boxplot again for all the features you can verify that most of the outliers have been removed.

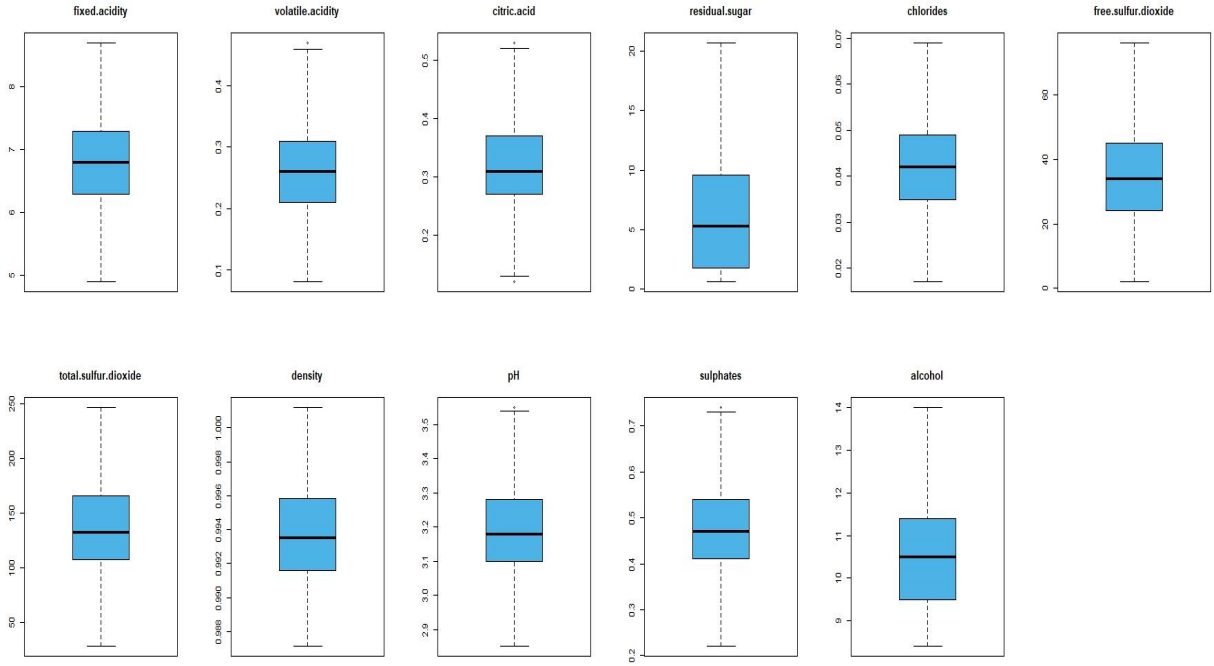

Figure 2: Viewing after removing outliers

And now you can see that most of the outliers have been removed from the dataset. And now you are good to go with data normalization with your dataset.

2. Scaling

The main goal of scaling is to make every feature in your dataset to have the same scale so each feature is equally important. To scale the data again we have 2 options. One is Normalization (min-max) and the other one is Standardization.

In order to select which scaling is best for the dataset, we will plot a histogram to see how our data is being distributed for each feature. The following picture of the histogram shows how the data in each feature is being distributed in the wine dataset.

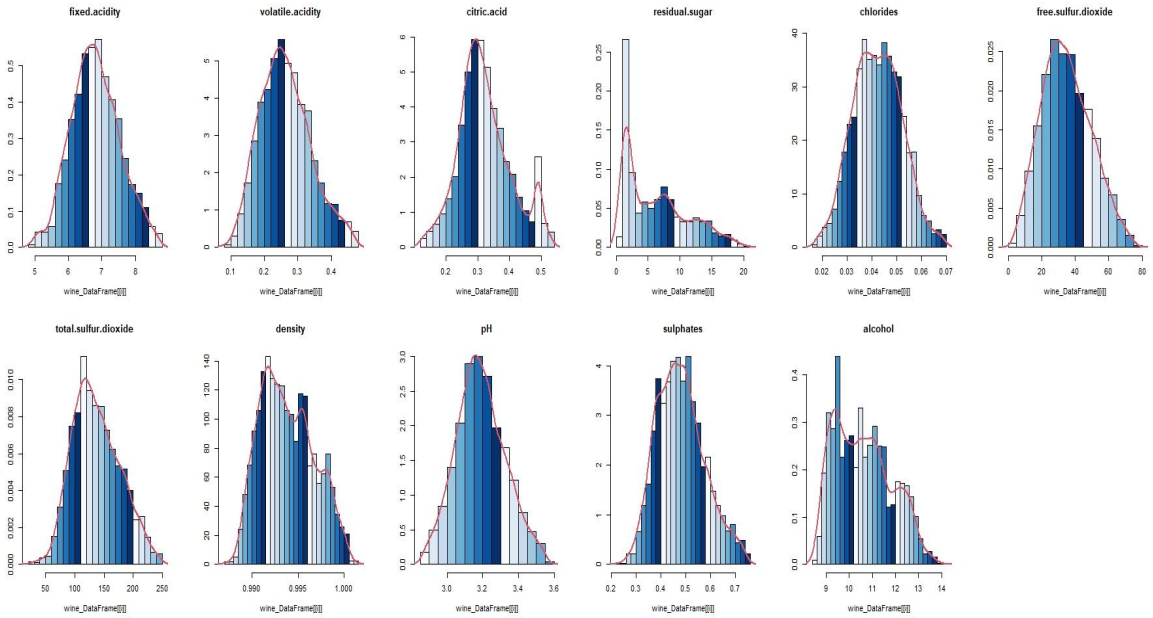

Figure 3: Histogram for distribution

As you can see here that the distribution of some features falls under Normal/Gaussian distribution and others falls under skewed distribution. Standardization is preferable if we have Gaussian distribution for all the features. But here we will be using Normalization (min-max) because some of the features fall under skewed distribution.

Min-max Normalization is basically rescaled each feature value so that feature values will be ended up in the ranging between 0 and 1.

```
> summary(wine_DataFrame_without_quality)
 fixed.acidity    volatile.acidity  citric.acid     residual.sugar     chlorides      free.sulfur.dioxide
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.3684   1st Qu.:0.3333   1st Qu.:0.3659   1st Qu.:0.0597   1st Qu.:0.3462   1st Qu.:0.2973
 Median :0.5000   Median :0.4615   Median :0.4634   Median :0.2338   Median :0.4808   Median :0.4324
 Mean   :0.5006   Mean   :0.4680   Mean   :0.4913   Mean   :0.2870   Mean   :0.4855   Mean   :0.4447
 3rd Qu.:0.6316   3rd Qu.:0.5897   3rd Qu.:0.6098   3rd Qu.:0.4478   3rd Qu.:0.6154   3rd Qu.:0.5811
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
 total.sulfur.dioxide    density             pH           sulphates         alcohol
 Min.   :0.0000       Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.3630       1st Qu.:0.3180   1st Qu.:0.3571   1st Qu.:0.3654   1st Qu.:0.1964
 Median :0.4795       Median :0.4535   Median :0.4714   Median :0.4808   Median :0.3750
 Mean   :0.4979       Mean   :0.4768   Mean   :0.4841   Mean   :0.4988   Mean   :0.3927
 3rd Qu.:0.6301       3rd Qu.:0.6182   3rd Qu.:0.6143   3rd Qu.:0.6154   3rd Qu.:0.5357
 Max.   :1.0000       Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
> |
```

Figure 4: Summary of wine dataset

As you can see, the minimum and maximum value for all the features are set to 0 and 1. So now all the features will be treated equally important in your training phase.

2. Defining the number of cluster center

In order to define the number of clusters we will use some built-in methods like the Elbow method, Silhouette method, Gap statistic Method and NBclust method.

1. Elbow method

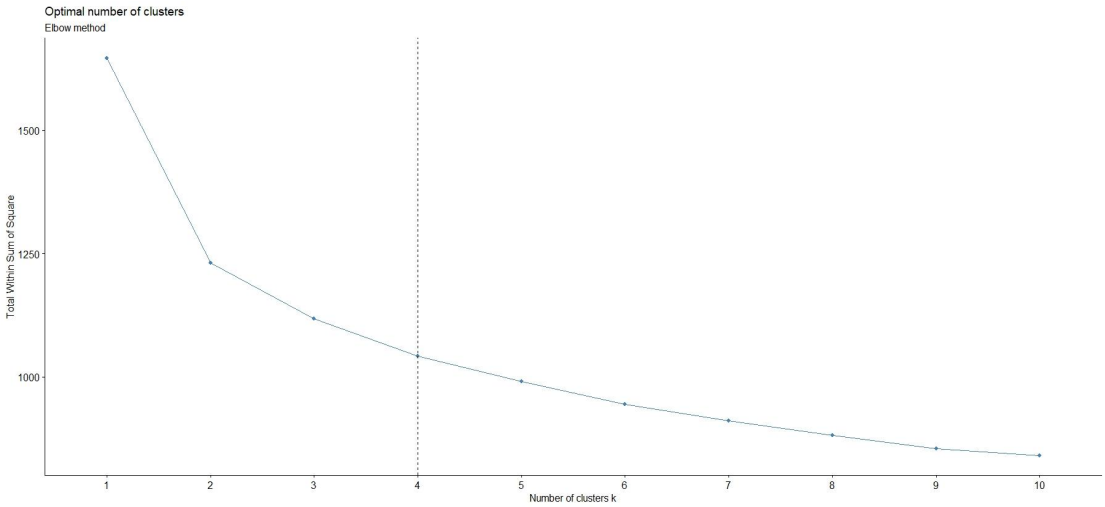

Figure 5: Elbow plot

As you can see here the elbow plot suggests that 4 as the best cluster.

2. Silhouette method

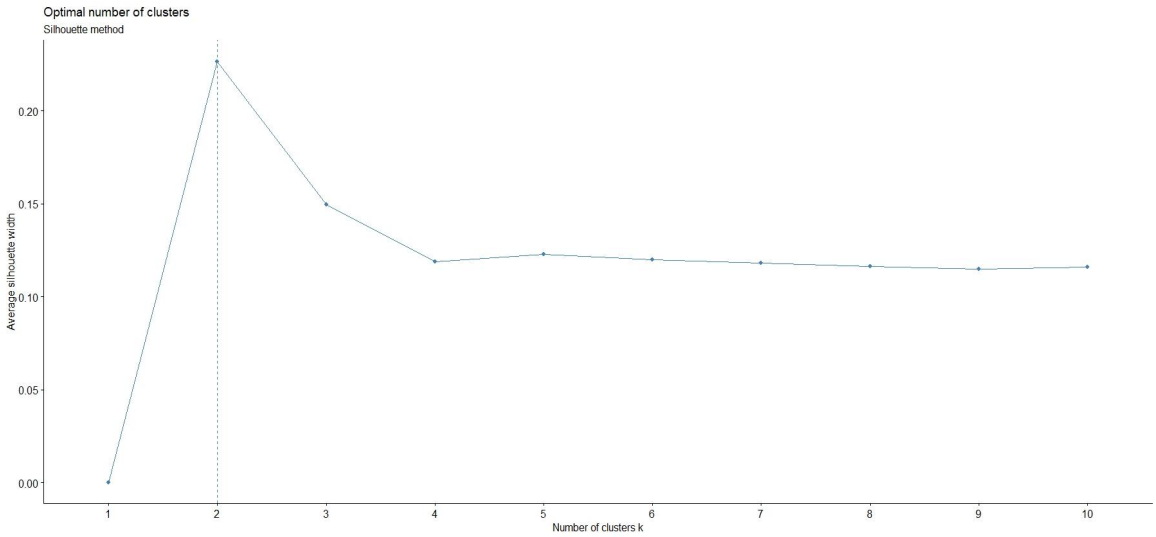

Figure 6: Silhouette plot

As you can see here the silhouette plot suggests that 2 as the best cluster.

3. Gap statistic method

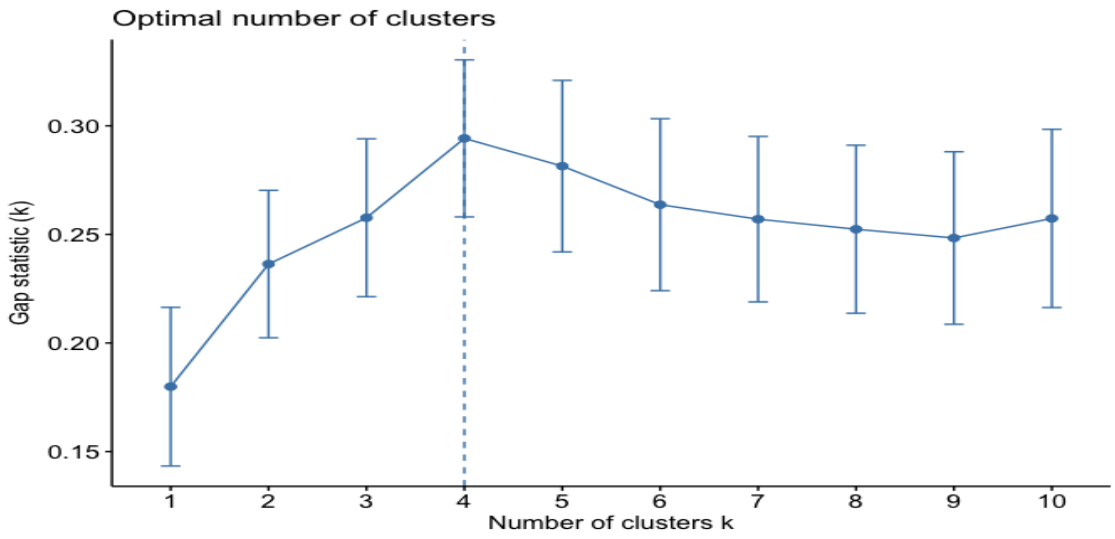

Figure 7: Gap statistic plot

As you can see here the Gap statistic method suggests that 4 as the best cluster.

4. NBclust method

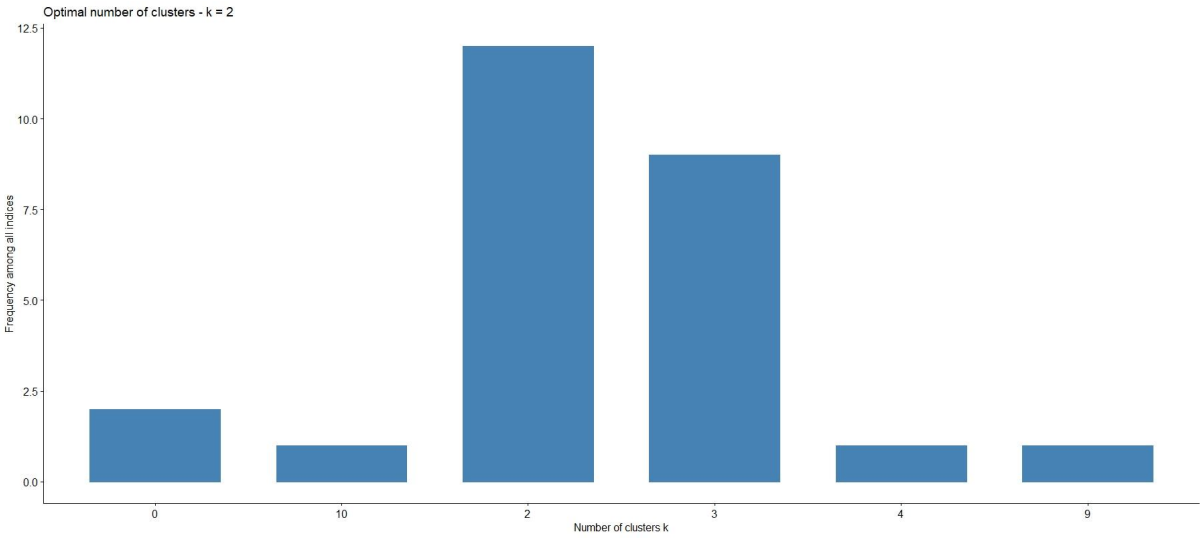

Figure 8: NBclust plot

```
> fviz_nbclust(nb)
Among all indices:
===================
* 2 proposed  0 as the best number of clusters
* 12 proposed  2 as the best number of clusters
* 9 proposed  3 as the best number of clusters
* 1 proposed  4 as the best number of clusters
* 1 proposed  9 as the best number of clusters
* 1 proposed  10 as the best number of clusters

Conclusion
=========================
* According to the majority rule, the best number of clusters is  2 .
```

Figure 9: NBclust results

As you can see clearly, the NBclust method suggests that 2 as the best cluster.

So here 2 of the methods (Elbow & Gap statistic) suggest 4 as the best cluster cluster and other 2 methods (Silhouette & NBclust) suggest 2 as the best cluster. Since we have four quality labels namly 5, 6, 7 & 8 as our output labels in our wine dataset it's obvious that we should have 4 clusters in our dataset. So k = 4 is the optimum number of clusters that we should have in our wine dataset.

3. K-means clustering analysis

Here we will perform k-means analysis for k = 2, k = 3 & k = 4 and then we will evaluate the produced cluster output against the 12th (quality) column for each k values by producing confusion matrices with actual quality labels and predicted cluster results. And with the confusion matrices we will calculate the accuracy, recall (sensitivity) and specificities.

- Accuracy = (TP + TN) / (TP + FN + TN + FP)
- Recall (sensitivity) = TP / (TP + FN)
- Specificity = TN / (TN + FP)

(TP - True Positive, TN - True Negative, FP - False Positive, FN - False Negative)

1. K-means analysis when (k = 2)

```
> set.seed(5005)
> KM <- kmeans(wine_DataFrame_norm[,-12], centers = 2, nstart = 25)
> KM
K-means clustering with 2 clusters of sizes 1450, 2297

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.5228312        0.4826879   0.4955088      0.5158929 0.5979310           0.5615797
2     0.4864870        0.4586640   0.4885694      0.1425285 0.4145457           0.3708803
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.6353125 0.6851012 0.4514089 0.5084218 0.2112562
2            0.4112183 0.3452341 0.5046645 0.4926576 0.5072698

Clustering vector:
   [1] 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 1 1 1
  [52] 1 1 1 2 1 1 2 1 1 2 2 1 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 2 2 2 2 2 1 2 2 1 2 2 2 1 2 1 1 2 2 2 1 1 2 1 1
 [103] 2 2 2 1 1 1 1 1 1 1 1 1 2 2 1 2 2 2 2 2 1 1 2 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 2
 [154] 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 2 1 1 2 2 2 2 1 1
 [205] 1 1 2 1 2 2 2 2 2 2 2 1 1 2 2 1 2 1 1 2 2 2 2 1 2 2 2 1 2 2 2 1 1 1 2 1 1 2 2 2 2 1 2 2 1 2 2 2 2 2 1 1
 [256] 1 1 2 2 2 2 2 1 2 2 2 2 2 1 1 1 1 1 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2 2 1
 [307] 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 2 1 1 1 1 1 1 1 2 2 1 2 2 1 2
 [358] 1 2 2 1 2 1 2 1 1 2 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 2 2 1 1 2
 [409] 2 2 2 1 2 1 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 2 2 2 2 2 2 2 1 2 1 1 1 1 1 2 1 1 1 2 2 1 1 1 1 2 1 2 1 1 1 2
 [460] 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 2 1 2 2 2 2 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 1
 [511] 1 2 1 1 1 2 1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 1 2 2 1 1 1 1 2 1 2 1 1 1 2 1 1
 [562] 1 1 1 1 2 2 2 1 2 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 2 2 1 2 2
 [613] 2 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 1 2 2 1 2 2 2 2 2 1 1 1 2 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 2 2 2
 [664] 2 2 2 2 1 1 2 2 1 2 2 1 2 1 2 1 2 2 1 2 2 2 1 2 1 2 2 1 1 1 2 2 2 2 1 2 1 1 2 2 1 2 2 2 2 2 2 1 2 2 1 2
 [715] 1 2 1 1 1 1 1 2 2 1 2 2 2 2 2 1 1 1 1 2 1 1 1 2 2 1 1 2 1 1 2 2 1 1 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 2 2
 [766] 1 1 2 1 1 1 1 1 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 1 2 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 1 1 2 1 2 1 1 1 1 1 1 2
 [817] 1 1 1 1 1 2 1 2 2 2 2 1 1 2 2 2 1 1 1 1 2 2 1 2 1 1 2 2 1 1 2 2 2 1 2 2 1 2 2 1 1 2 2 1 1 1 1 2 1 1 2
 [868] 1 2 2 2 1 1 2 1 1 1 1 1 1 1 2 1 2 1 2 2 2 2 2 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 1 1 1 2 1 2 2
 [919] 1 1 1 1 2 1 1 1 2 2 1 2 1 1 1 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2 1 1 1 1 1 2 2
 [970] 1 2 1 1 1 2 2 1 2 2 1 1 1 2 2 2 2 1 2 2 1 2 2 2 2 2 1 1 2 1 2 1
 [ reached getOption("max.print") -- omitted 2747 entries ]

Within cluster sum of squares by cluster:
[1] 463.9606 766.9651
 (between_SS / total_SS =  25.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
```

Figure 10: k-means results (k=2)

- Each cluster sizes => 1450, 2297
- Total Within Cluster Sum of Squares (WSS) => 1230.926
- Between Cluster Sum of Squares (BSS) => 415.8025
- Total Sum of Squares (TSS) => 1646.728
- (BSS / TSS) => 25.3 %

The following code snippet and the related output shows which quality label is represented by which cluster.

```
> table(KM.clusters, wine_dataFrame_labels)
           wine_dataFrame_labels
KM.clusters    5    6    7    8
          1  600  684  144   22
          2  434 1119  628  116
>
```

As you can see that the quality 5 is being described more by cluster 1 and the rest of the qualities (6, 7 & 8) are being described by cluster 2. So with this information we will make our Confusion Matrix including the accuracy, recall and precision for the clusters.

```
> confusion_Matrix
Confusion Matrix and Statistics

          Reference
Prediction    1    2
         1  600  850
         2  434 1863

               Accuracy : 0.6573
                 95% CI : (0.6419, 0.6725)
    No Information Rate : 0.724
    P-Value [Acc > NIR] : 1

                  Kappa : 0.2374

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.5803
            Specificity : 0.6867
         Pos Pred Value : 0.4138
         Neg Pred Value : 0.8111
             Prevalence : 0.2760
         Detection Rate : 0.1601
   Detection Prevalence : 0.3870
      Balanced Accuracy : 0.6335

       'Positive' Class : 1

>
```

Figure 11: Confusion matrix (k=2)

2. K-means analysis when (k = 3)

```
> set.seed(5005)
> KM <- kmeans(wine_DataFrame_norm[,-12], centers = 3, nstart = 25)
> KM
K-means clustering with 3 clusters of sizes 1198, 1252, 1297

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4876329        0.5232332   0.4976994      0.1692975 0.3203898           0.3642163
2     0.5270094        0.4916134   0.5093899      0.5566656 0.5956470           0.5757815
3     0.4869436        0.3940751   0.4677962      0.1354427 0.5317152           0.3924389
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.3742348 0.2865380 0.4480444 0.4383749 0.6300655
2            0.6460658 0.7067857 0.4352008 0.5020890 0.2040307
3            0.4692089 0.4304020 0.5644785 0.5513166 0.3556325

Clustering vector:
   [1] 1 3 3 3 2 3 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 3 1 1 2 2 2 2 3 3 3 2 2 2 2 2 3 1 3 2 2 2 2
  [52] 2 2 2 1 2 2 1 2 2 1 1 2 2 2 2 1 3 2 1 2 2 1 2 2 2 2 2 3 1 1 1 3 1 2 1 3 2 3 1 1 2 1 2 2 1 3 3 2 2 3 2 2
 [103] 3 3 3 2 2 2 2 2 2 2 2 2 3 3 3 1 3 3 1 2 2 3 2 3 3 2 2 2 2 2 2 1 2 2 2 2 3 3 2 2 2 3 3 3 3 3 3 2 2 3 1
 [154] 2 3 3 2 2 2 2 1 2 3 2 2 2 2 3 3 3 3 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 3 1 3 3 1 2 2 2 2 3 2 2 3 3 3 3 2 2
 [205] 2 2 1 2 3 1 3 3 3 3 1 3 2 3 3 2 3 2 2 1 3 3 1 2 3 1 1 2 1 1 3 2 2 2 3 2 2 3 1 3 1 2 1 3 2 3 3 3 3 2 2
 [256] 2 2 3 1 1 3 2 3 3 3 3 2 2 2 2 2 3 3 1 3 3 3 2 2 1 1 3 2 3 3 3 1 1 3 3 3 3 3 3 2 2 2 3 3 3 2 3 3 1 1 2
 [307] 3 3 3 3 3 3 3 2 3 3 1 2 3 3 3 2 3 3 2 2 2 3 2 2 2 2 2 2 2 2 1 3 3 3 2 3 3 3 2 2 2 2 2 3 3 1 3 2 3 1 2 3
 [358] 2 3 1 2 3 2 3 2 2 3 2 2 2 1 2 2 2 2 3 2 2 2 2 2 2 2 1 3 3 2 2 2 2 3 1 3 3 2 2 2 2 2 2 2 2 3 3 2 2 3
 [409] 3 3 3 2 3 2 1 3 3 1 3 1 2 2 2 2 2 2 2 3 2 3 3 3 3 3 3 2 3 2 2 2 2 2 3 2 2 2 3 3 2 2 2 2 3 2 3 2 2 2 3
 [460] 3 2 2 3 2 2 2 2 2 2 2 3 3 3 3 2 3 3 3 2 2 2 3 3 2 1 1 3 3 2 2 2 2 2 2 2 2 3 2 2 3 2 2 2 2 2 2 2 2 3 2
 [511] 3 1 2 2 2 3 2 2 3 2 2 3 2 2 3 1 2 2 2 2 2 2 2 3 2 3 3 2 2 2 3 3 3 3 3 2 3 1 2 2 2 2 3 2 3 2 2 2 3 3 2
 [562] 3 2 2 2 2 1 1 3 3 3 2 2 2 3 2 3 3 2 2 2 2 2 1 2 2 1 2 2 2 2 2 3 3 2 2 3 2 2 3 3 2 2 3 2 2 2 2 2 2 1 3 3 3 3
 [613] 1 1 2 2 2 2 2 2 2 2 2 3 1 3 2 3 2 2 2 3 1 2 3 1 1 2 1 3 1 1 2 2 2 1 2 2 2 2 2 3 3 2 3 3 2 1 2 3 3 3 3 1
 [664] 3 1 1 3 3 2 3 3 2 1 3 2 1 3 3 2 1 3 2 3 3 1 2 3 2 3 3 2 2 3 3 3 3 3 2 3 2 3 1 3 3 1 3 3 1 1 2 3 3 2 3
 [715] 2 1 2 2 2 2 2 1 3 2 3 1 3 3 2 2 2 2 1 2 2 2 3 3 2 3 3 2 2 3 2 2 3 2 2 2 2 2 2 1 3 2 2 2 2 2 1 1 1 1 3 1 2
 [766] 2 2 1 2 2 3 3 3 3 2 1 2 3 3 3 3 3 3 1 2 2 2 2 1 2 2 2 2 2 2 3 3 3 3 3 3 3 3 2 2 2 1 2 3 2 3 2 3 2 2 3
 [817] 2 2 2 2 2 3 2 3 3 3 3 2 2 1 1 1 2 2 2 2 3 3 2 3 3 2 2 3 1 2 2 3 1 3 2 3 3 2 3 1 2 2 3 3 3 2 2 2 1 2 2 3
 [868] 2 3 3 3 2 2 3 2 3 3 2 2 2 2 2 3 2 2 3 3 1 3 3 2 2 3 2 2 2 2 2 2 2 3 2 2 2 2 3 3 1 2 2 3 2 2 3 2 3 1
 [919] 2 2 2 2 3 2 2 2 3 3 2 3 2 2 2 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 2 2 3 3 2 3 2 2 2 3 3
 [970] 3 3 2 2 2 3 3 3 1 3 2 3 3 2 3 1 1 1 2 1 3 2 1 3 1 1 2 3 1 3 3 2
 [ reached getOption("max.print") -- omitted 2747 entries ]

Within cluster sum of squares by cluster:
[1] 349.2902 377.1902 391.4500
 (between_SS / total_SS =  32.1 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> |
```

Figure 12: k-means results (k=3)

- Each cluster sizes => 1198, 1252, 1297
- Total Within Cluster Sum of Squares (WSS) => 1117.93
- Between Cluster Sum of Squares (BSS) => 528.7978
- Total Sum of Squares (TSS) => 1646.728
- (BSS / TSS) => 32.1 %

The following code snippet and the related output shows which quality label is represented by which cluster.

```
> table(KM.clusters, wine_dataFrame_labels)
            wine_dataFrame_labels
KM.clusters   5   6   7   8
          1 127 556 431  84
          2 535 575 122  20
          3 372 672 219  34
> |
```

As you can see that the quality 5 is being described more by cluster 2 and the quality 6 is being described more by cluster 3 the rest of the qualities (7 & 8) are being described by cluster 1. So with this information we will make our Confusion Matrix including the accuracy, recall and precision for the clusters.

```
> confusion_Matrix
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1 515 127 556
         2 142 535 575
         3 253 372 672

Overall Statistics

               Accuracy : 0.4596
                 95% CI : (0.4435, 0.4757)
    No Information Rate : 0.4812
    P-Value [Acc > NIR] : 0.9962

                  Kappa : 0.1856

 Mcnemar's Test P-Value : <2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.5659   0.5174   0.3727
Specificity            0.7593   0.7357   0.6785
Pos Pred Value         0.4299   0.4273   0.5181
Neg Pred Value         0.8450   0.8000   0.5384
Prevalence             0.2429   0.2760   0.4812
Detection Rate         0.1374   0.1428   0.1793
Detection Prevalence   0.3197   0.3341   0.3461
Balanced Accuracy      0.6626   0.6266   0.5256
>
```

Figure 13: Confusion matrix (k=3)

3. K-means analysis when (k = 4)

```
> set.seed(5005)
> KM <- kmeans(wine_DataFrame_norm[,-12], centers = 4, nstart = 25)
> KM
K-means clustering with 4 clusters of sizes 1207, 787, 949, 804

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.5259887        0.4915875   0.5130842      0.5622983 0.5995475           0.5779910
2     0.3841704        0.5672629   0.4528156      0.1569882 0.3104047           0.3687884
3     0.4179330        0.4074059   0.4167673      0.1532118 0.5311461           0.4210834
4     0.6738020        0.4067643   0.5840311      0.1589441 0.4318551           0.3466704
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.6466408 0.7112009 0.4320748 0.5027404 0.1990867
2            0.3655347 0.2524703 0.5310764 0.4570668 0.6725321
3            0.4977145 0.4418173 0.6245522 0.5647443 0.3441649
4            0.4045639 0.3855739 0.3502310 0.4557023 0.4668251

Clustering vector:
   [1] 4 4 4 3 1 3 1 1 1 3 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 3 4 2 1 1 1 1 3 4 3 1 1 1 1 1 3 2 3 1 1 1 1
  [52] 1 1 1 2 1 1 4 1 1 4 4 1 1 1 1 4 3 1 4 1 1 4 1 1 1 1 1 1 3 4 4 4 4 4 1 2 3 1 3 4 2 1 4 1 1 4 4 4 1 1 4 1 1
 [103] 3 3 3 1 1 1 1 1 1 1 1 1 1 3 3 3 4 4 4 2 1 1 3 1 3 3 1 1 1 3 3 1 4 1 1 1 1 1 3 3 1 1 1 3 3 3 4 4 3 1 1 3 4
 [154] 1 3 3 1 1 1 1 4 3 3 1 1 1 1 3 3 4 3 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 3 4 3 3 4 1 1 1 1 4 1 1 3 3 3 3 1 1
 [205] 1 1 2 1 4 4 4 4 4 4 4 3 1 3 3 1 3 1 1 4 3 4 4 1 4 4 4 1 4 4 4 1 1 4 4 1 1 3 4 4 2 1 2 3 1 4 4 3 3 1 1
 [256] 1 1 3 2 4 3 1 3 3 4 4 1 1 1 1 1 4 4 4 3 3 3 1 1 4 4 3 1 4 4 3 4 4 3 3 4 3 3 4 1 1 1 4 4 4 1 3 4 4 4 1
 [307] 4 3 4 4 4 3 4 1 4 4 2 4 3 4 3 1 4 3 1 1 1 3 1 1 1 1 1 1 1 1 4 3 3 4 1 4 4 1 1 1 1 1 3 4 4 3 1 3 2 1 4
 [358] 1 4 4 1 3 1 3 1 1 4 1 1 1 4 1 1 1 1 3 1 1 1 1 1 1 1 4 4 3 1 1 1 1 3 4 4 4 1 1 1 1 1 1 1 1 1 4 4 1 1 3
 [409] 3 3 3 1 4 1 4 4 3 4 3 4 1 1 1 1 1 1 1 3 1 3 4 4 4 3 3 1 4 1 1 1 1 1 3 1 1 1 3 3 1 1 1 1 3 1 4 1 1 1 2
 [460] 2 1 1 3 1 1 1 1 1 1 1 1 3 3 3 3 1 3 3 3 1 1 1 4 3 1 4 4 3 3 1 1 1 1 1 1 1 1 4 1 1 3 1 1 1 1 1 1 1 3 1
 [511] 3 4 1 1 1 3 1 1 4 1 1 3 1 1 4 2 1 1 1 1 1 1 1 3 1 3 3 1 1 1 4 4 3 3 3 1 3 4 1 1 1 1 4 1 4 1 1 1 3 3 1
 [562] 3 1 1 1 2 2 3 3 3 1 1 1 3 1 3 3 1 1 1 1 2 1 1 2 1 1 1 1 3 3 1 1 3 1 1 3 3 1 4 1 1 1 1 1 1 1 2 3 3 4 4
 [613] 2 2 1 1 1 1 1 1 1 1 4 2 4 1 3 1 1 1 4 2 1 4 4 2 1 4 3 2 2 1 1 1 4 1 3 1 1 3 4 3 1 2 3 1 4 1 3 3 3 3 4
 [664] 3 4 4 4 3 1 4 3 1 4 4 1 2 3 3 1 4 4 1 4 4 2 1 4 1 3 3 1 1 3 3 4 3 2 1 3 1 3 2 4 3 2 4 4 4 2 1 4 3 1 4
 [715] 1 4 1 1 1 1 4 3 1 3 2 3 3 1 1 1 1 4 1 1 1 4 4 1 4 3 1 1 3 1 1 4 1 1 1 1 1 1 4 3 1 1 1 1 2 2 4 2 3 3 1
 [766] 1 1 4 1 1 3 3 4 4 1 4 1 3 3 3 3 3 3 2 1 1 1 1 2 1 1 1 1 1 1 3 4 3 4 4 4 4 4 3 1 1 2 1 2 3 3 3 3 1 1 4
 [817] 1 1 1 1 1 4 1 4 4 3 3 1 1 4 4 4 1 1 1 1 3 3 1 3 1 1 3 4 1 1 4 4 3 1 3 4 1 3 4 1 1 3 3 3 3 3 1 2 3 3 3
 [868] 1 3 4 3 1 1 4 1 3 3 1 1 1 1 1 3 1 3 1 4 3 2 4 4 1 1 3 1 1 1 1 1 1 1 4 1 1 1 1 3 3 2 1 1 1 1 1 4 1 4 4
 [919] 1 1 1 1 3 1 1 1 4 3 1 4 1 1 1 3 3 3 3 3 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 3 3 1 1 3 3 1 3 1 1 1 1 3 3
 [970] 3 4 1 1 1 3 3 3 2 3 1 3 1 4 4 4 4 1 4 2 4 2 3 4 2 3 3 4 3 4 1
 [ reached getOption("max.print") -- omitted 2747 entries ]

Within cluster sum of squares by cluster:
[1] 356.2492 195.7300 270.9504 219.6557
 (between_SS / total_SS =  36.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> |
```

Figure 14: k-means results (k=4)

- Each cluster sizes => 1207, 787, 949, 804
- Total Within Cluster Sum of Squares (WSS) => 1042.585
- Between Cluster Sum of Squares (BSS) => 604.1429
- Total Sum of Squares (TSS) => 1646.728
- (BSS / TSS) => 36.7 %

The following code snippet and the related output shows which quality label is represented by which cluster.

```
> table(KM.clusters, wine_dataFrame_labels)
            wine_dataFrame_labels
KM.clusters   5   6   7   8
          1 520 549 117  21
          2  51 348 326  62
          3 253 502 168  26
          4 210 404 161  29
> |
```

As you can see that the quality 5 is being described more by cluster 1, quality 6 is being described more by cluster 3, quality 7 is being described more by cluster 2 and the quality 8 being described more by cluster 4. So with this information we will make our Confusion Matrix including the accuracy, recall and precision for the clusters.

```
> confusion_Matrix
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4
         1 520 117 549  21
         2  51 326 348  62
         3 253 168 502  26
         4 210 161 404  29

Overall Statistics

               Accuracy : 0.3675
                 95% CI : (0.352, 0.3832)
    No Information Rate : 0.4812
    P-Value [Acc > NIR] : 1

                  Kappa : 0.143

 Mcnemar's Test P-Value : <2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.5029   0.4223   0.2784  0.21014
Specificity            0.7468   0.8450   0.7701  0.78526
Pos Pred Value         0.4308   0.4142   0.5290  0.03607
Neg Pred Value         0.7976   0.8493   0.5350  0.96296
Prevalence             0.2760   0.2060   0.4812  0.03683
Detection Rate         0.1388   0.0870   0.1340  0.00774
Detection Prevalence   0.3221   0.2100   0.2533  0.21457
Balanced Accuracy      0.6248   0.6337   0.5242  0.49770
>
```

Figure 15: Confusion matrix (k=4)

(Cluster Plots when k = 2, k = 3 & k =4)

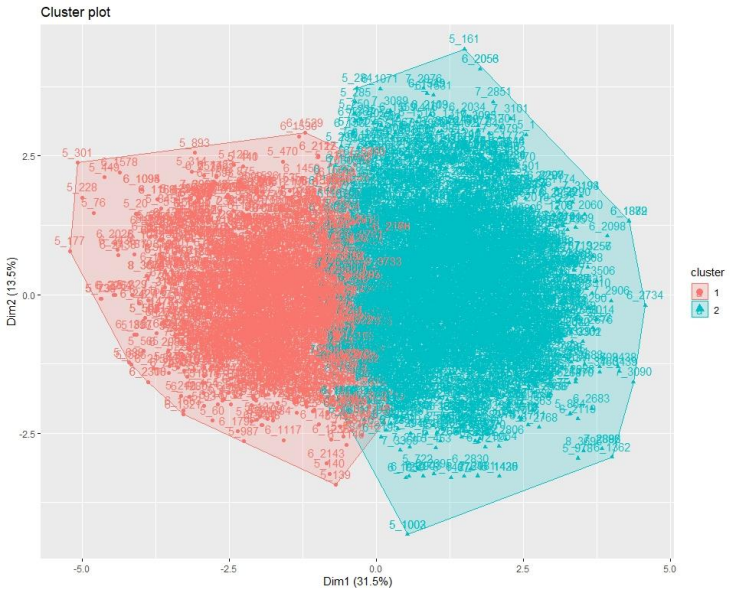

Figure 16: Cluster plot (k=2)

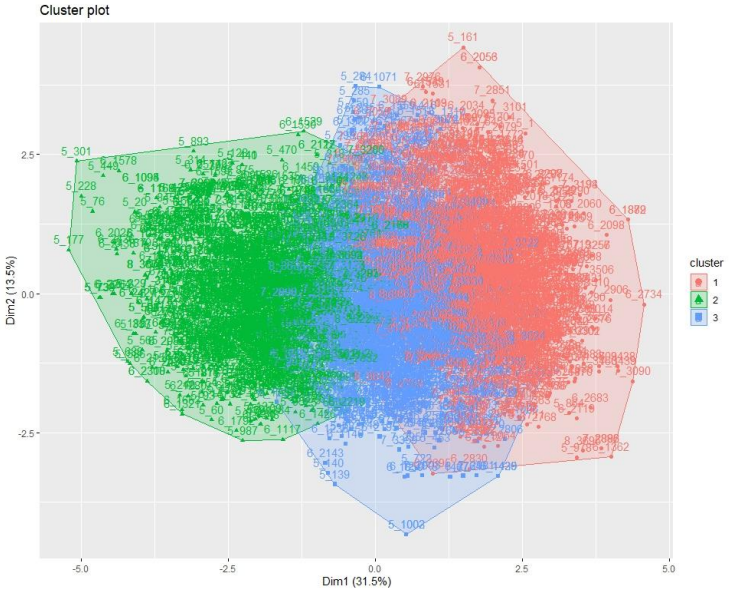

Figure 17: Cluster plot (k=3)

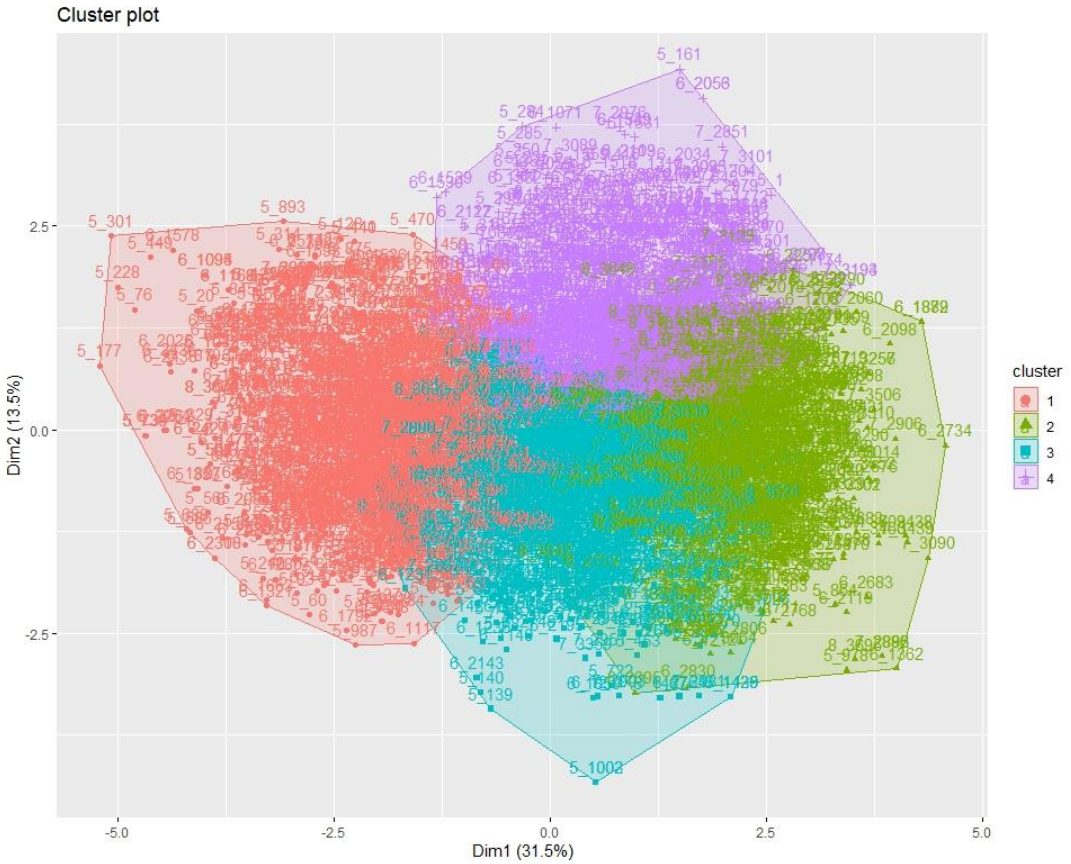

Figure 18: Cluster plot (k=4)

4. Defining the winner cluster.

In order to define the best cluster out of the 3 clusters mentioned above we will consider the total within sums of square (WSS) and the between sums of square (BSS) values.

Within sums of square (WSS) values will show us how well the separation is being done in each cluster. The lower the within sums of square value the higher the separation. So we should get the k value which gives the lower value for the within sums of squares.

Between sums of square (BSS) values will show us how well the cohesion is being done within the clusters. The higher the between sums of square value the higher the cohesion. So we should get the k value which gives the higher value for between sums of squares.

And we should calculate ((Between sums of square / Total sums of square) * 100) for all the k values and we should choose the k value which gives the higher value for ((BSS) / (TSS) * 100). And the Total sums of squares will be calculated by adding Total within sums of square value with the Between sums of square value (TSS = WSS + BSS).

The following table shows the summary for all the values of each cluster (k = 2, k = 3 & k = 4).

Value for k (no of cluster centers)	Total within sums of square (WSS)	Between sums of square (BSS)	Total sums of square (TSS)	(BSS)/ (TSS)
k = 2	1230.926	415.8025	1646.728	25.3 %
k = 3	1117.93	528.7978	1646.728	32.1 %
k = 4	1042.585	604.1429	1646.728	36.7 %

Tabel 1: Summary of clustering results (k = 2, 3, 4)

As you see when k = 4 only it gives lower value for WSS (1042.585) and higher value for BSS (604.1429) and also it gives higher percentage for ((BSS) / (TSS) * 100). So it's obvious that the optimum value for k is 4. So the winner cluster value for the wine dataset is 4.

The following picture is the cluster plot for the winner cluster which is k = 4 in our case.

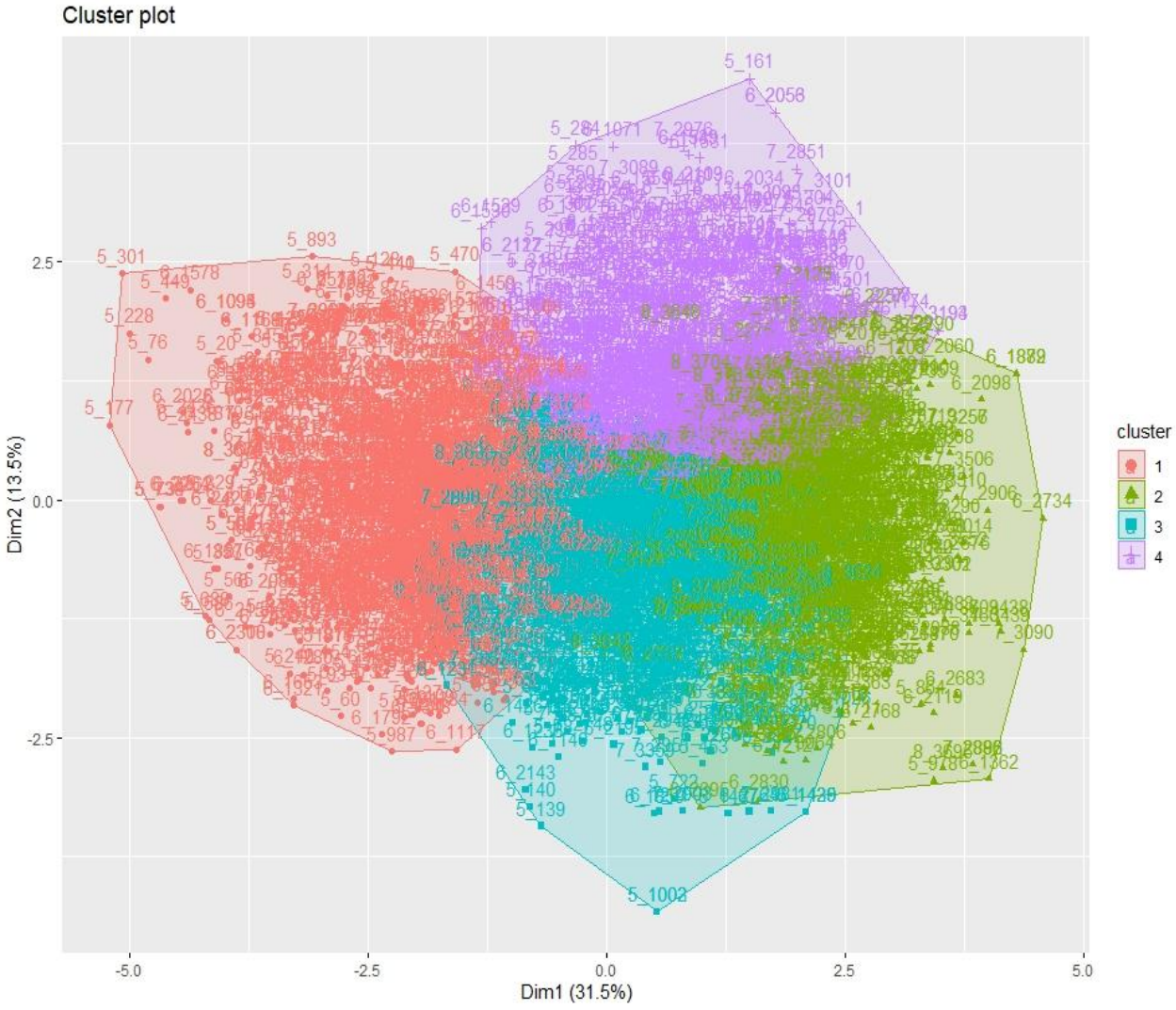

Figure 19: Winner Cluster plot (k=4)

5. Applying Principal Component Analysis (PCA)

Principal component analysis (PCA) is a method of obtaining important variables (in the form of components) from a large set of variables available in a data set. It extracts a low dimensional set of features by taking a projection of irrelevant dimensions from a high dimensional data set with a motive to capture as much information as possible.

With the use of Principal component analysis (PCA) we can reduce the dimensionality of the dataset. In order to apply the PCA to a dataset it is a must to scale the data first. We can get the maximum use of PCA only if we apply PCA for a scaled dataset.

The following is the code snippet and the related summary output for applying PCA to our normalized wine dataset.

```
> #PCA APPLYING.....
> PCA_components <- prcomp(~fixed.acidity + volatile.acidity + citric.acid
+                          + residual.sugar + chlorides + free.sulfur.dioxide
+                          + total.sulfur.dioxide + density + pH + sulphates +
+                            alcohol, data = wine_DataFrame_norm,
+                          scale. = T)
> summary(PCA_components)
Importance of components:
                          PC1    PC2    PC3    PC4     PC5     PC6     PC7     PC8    PC9    PC10    PC11
Standard deviation     1.8627 1.2208 1.0917 1.0505 0.99697 0.87666 0.85730 0.76251 0.6170 0.52192 0.11276
Proportion of Variance 0.3154 0.1355 0.1084 0.1003 0.09036 0.06987 0.06681 0.05286 0.0346 0.02476 0.00116
Cumulative Proportion  0.3154 0.4509 0.5593 0.6596 0.74994 0.81980 0.88662 0.93948 0.9741 0.99884 1.00000
> |
```

Figure 20: PCA application

As you can see the summary of the principal components it shows that the first 8 principal components achieve the cumulative score > 96%. So now we will transform our dataset into which we have only the first 8 principal components. The following code snippet shows the code and the output of the transformed dataset.

```
> trnfrm_PCA_components <- as.data.frame(PCA_components$x[,1:8])
> head(trnfrm_PCA_components)
         PC1        PC2        PC3         PC4        PC5         PC6        PC7          PC8
1  2.54585895  2.8826716 -0.4357777 0.007385251  1.5053353  0.41091506  0.4484454  0.008014836
2  0.18688467  2.4153025 -0.9567648 0.605963803  1.0671895  0.01806569  0.4802707 -1.309010536
3  1.61494065  1.5545729 -1.8854779 0.937242107  1.3056175  0.61628142  0.1278765 -0.513822030
4 -0.62594099 -1.4782284  1.2154589 0.962230833  0.7514572  0.79021212 -0.4605018 -0.464103249
5 -1.09005515 -2.0978627  1.4801774 1.253275981 -0.5629674 -0.62552601  0.6883195 -0.541439942
6 -0.09345576 -0.9366681  1.8242048 0.463288700  0.5480815 -1.38161454 -1.0542516  0.003285588
> |
```

Figure 21: Viewing principal components

As you can see, the transformed dataset only has the first 8 principal components. And Now we need to provide this transformed dataset for our k-means algorithm.

6. K-means clustering analysis for PCA

Now we will perform k-means clustering for this newly transformed dataset with the winner k value, which is 4 that we got previously.

```
> set.seed(1600) #Setting seed
> KM.PCA <- kmeans(trnfrm_PCA_components, centers = 4, nstart = 25)
> KM.PCA
K-means clustering with 4 clusters of sizes 1179, 867, 782, 919

Cluster means:
        PC1         PC2        PC3         PC4         PC5         PC6         PC7         PC8
1 -2.2656845 -0.06476213  0.1863376 -0.08491831 -0.05872567  0.02637979  0.1818482  0.07578242
2  0.7321132  1.37716526 -0.0158793  0.08707282  0.07971244 -0.01026658 -0.2148729 -0.14313497
3  2.1911789 -0.37809047  0.5464889 -0.42683563 -0.09677130 -0.04241151  0.2898052  0.10074141
4  0.3514669 -0.89442980 -0.6890957  0.39000219  0.08248318  0.01193161 -0.2771837 -0.04790995

Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21   22
   2    2    2    4    1    4    1    1    1    4    2    1    2    1    1    1    1    1    1    1    1    1
  23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38   39   40   41   42   43   44
   4    1    1    1    1    1    1    4    2    3    1    1    1    1    4    2    3    1    1    1    1    1
  45   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60   61   62   63   64   65   66
   1    3    4    1    1    1    1    1    1    1    3    1    1    2    1    1    2    2    1    1    1    2
  67   68   69   70   71   72   73   74   75   76   77   78   79   80   81   82   83   84   85   86   87   88
   4    1    2    1    1    2    1    1    1    1    1    4    2    2    2    2    2    1    3    4    1    4
  89   90   91   92   93   94   95   96   97   98   99  100  101  102  103  104  105  106  107  108  109  110
   2    3    1    2    1    1    2    2    2    1    1    2    1    1    4    4    4    1    1    1    1    1
 111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132
   1    1    1    1    4    4    4    2    2    2    3    1    1    4    1    4    4    1    1    1    1    1
 133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154
   1    2    1    1    1    1    4    4    1    1    1    4    4    4    2    2    4    1    1    4    2    1
 155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176
   4    4    1    1    1    1    2    1    3    1    1    1    1    4    3    2    4    1    1    1    1    1
 177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198
   1    1    1    1    2    1    1    1    1    1    4    2    4    4    2    1    1    1    1    2    1    1
```

Figure 22: k-means results PCA 1

```
   1    1    1    2    4    4    3    1    1    1    1    1    2    1    2    2    1    1    1    1    4    1
 925  926  927  928  929  930  931  932  933  934  935  936  937  938  939  940  941  942  943  944  945  946
   1    1    2    4    1    2    1    1    1    4    4    4    2    4    1    1    1    1    1    1    1    1
 947  948  949  950  951  952  953  954  955  956  957  958  959  960  961  962  963  964  965  966  967  968
   1    1    1    1    1    1    1    1    1    1    4    2    1    1    2    4    1    4    1    1    1    4
 969  970  971  972  973  974  975  976  977  978  979  980  981  982  983  984  985  986  987  988  989  990
   4    4    3    1    1    1    4    2    4    3    4    1    4    1    2    2    2    2    1    2    4    2
 991  992  993  994  995  996  997  998  999 1000
   3    4    2    3    4    4    2    1    2    1
 [ reached getOption("max.print") -- omitted 2747 entries ]

Within cluster sum of squares by cluster:
[1] 8009.623 5863.153 4599.807 6074.283
 (between_SS / total_SS =  36.6 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
>
```

Figure 23: k-means results PCA 2

- Each cluster sizes => 1179, 867, 782, 919
- Total Within Cluster Sum of Squares (WSS) => 24546.87
- Between Cluster Sum of Squares (BSS) => 14165.18
- Total Sum of Squares (TSS) => 38712.05
- (BSS / TSS) => 36.6 %

The following picture is the cluster plot for the k-means clustering with the first 8 principal components.

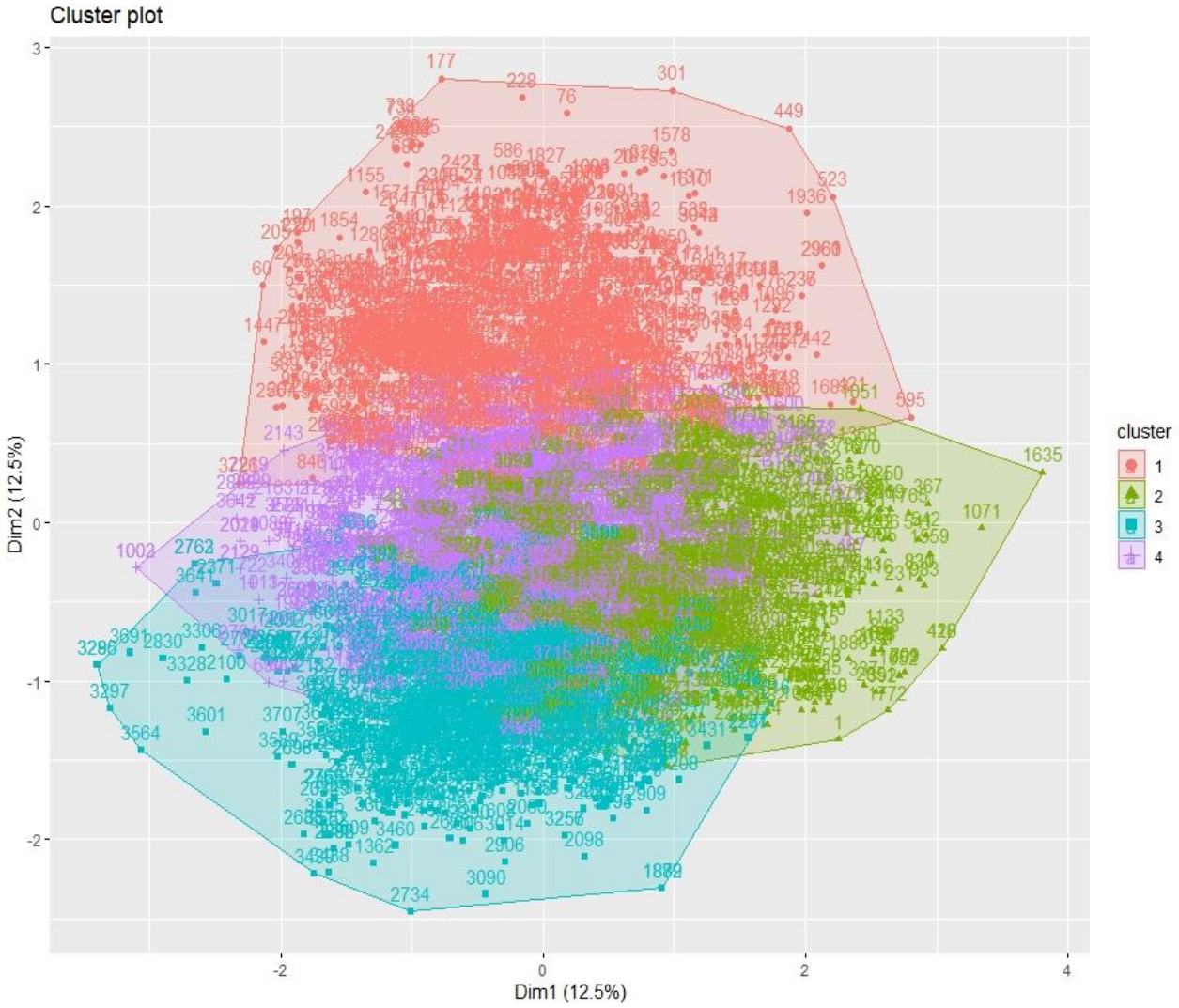

Figure 24: Cluster plot (PCA)

7. K-means clustering with PCA and without PCA results analysis and comparison

Winner cluster value (k = 4)	Total within sums of square (WSS)	Between sums of square (BSS)	Total sums of square (TSS)	(BSS)/ (TSS)
K-means without PCA	1042.585	604.1429	1646.728	36.7 %
K-means with PCA	24546.87	14165.18	38712.05	36.6 %

Tabel 2: Summary of clustering with PCA and without PCA

As you can see even though we reduced the dimensions by applying PCA to our dataset the percentage of ((BSS) / (TSS) * 100) still it gives approximately the same percentage (36.6%) as the percentage we got from the k-means clustering without applying PCA (36.7%).

It shows that applying PCA is a good way to reduce the dimension of your dataset. Which leads to faster computing for your clustering analysis.

Energy Forecasting Part

1. Preparing various input vectors for both (AR) and (NARX) approaches.

   For the AR approach we have constructed 4 datasets using various time-delayed input vectors of the 11th hour attribute with the help of R programming language. The following is the code snippet that has been used to create the 4 datasets for the AR approach.

```
library(readxl)

#Loading the data set and creating the data Frame.
dataFrame <- read_excel("C:\\Users\\SEAN\\Desktop\\MachineLearningandDataMining\\MLCW\\UoW_load.xlsx")
str(dataFrame)


columns= c("Day1","Day2","Day3","Day4", "NextDay")

# pass this vector length to ncol parameter
# and nrow with 0
DataFrame_days = data.frame(matrix(nrow = 0, ncol = length(columns)))

# assign column names
colnames(DataFrame_days) = columns


i <- 1
while(i<497){
  values <- data.frame(dataFrame$Time11[i], dataFrame$Time11[i+1], dataFrame$Time11[i+2], dataFrame$Time11[i+3], dataFrame$Time11[i+4])
  #Naming the Data Frame - Step 2
  names(values) <- c("Day1","Day2","Day3","Day4", "NextDay")
  DataFrame_days <- rbind(DataFrame_days, values)
  i = i + 1
}

str(DataFrame_days)
write.csv(DataFrame_days, "C:\\Users\\SEAN\\Desktop\\MachineLearningandDataMining\\MLCW\\test_DataFrame_AR(4).csv")
```

Figure 25: For producing AR input vectors

The names of the constructed dataset files and the details of those 4 dataset files is as follows.

- DataFrame_AR(1) => Dataset which is responsible for predicting the next day's (t) electricity consumption based on the (t-1) day electricity consumption data.
- DataFrame_AR(2) => Dataset which is responsible for predicting the next day's (t) electricity consumption based on the (t-1) and (t-2) days electricity consumption data.
- DataFrame_AR(3) => Dataset which is responsible for predicting the next day's (t) electricity consumption based on the (t-1), (t-2) and (t-3) days electricity consumption data.
- DataFrame_AR(4) => Dataset which is responsible for predicting the next day's (t) electricity consumption based on the (t-1), (t-2), (t-3) and (t-4) days electricity consumption data.

	A	B	C	D	E	F	G
1		Day1	NextDay				
2	1	88.6	106				
3	2	106	114.8				
4	3	114.8	109				
5	4	109	102.4				
6	5	102.4	87.2				
7	6	87.2	67.6				
8	7	67.6	116.2				
9	8	116.2	116				
10	9	116	114.2				
11	10	114.2	113.4				
12	11	113.4	120.4				
13	12	120.4	84.8				
14	13	84.8	66.8				
15	14	66.8	121.2				
16	15	121.2	124.6				
17	16	124.6	118.6				
18	17	118.6	133.8				

Figure 26: Dataframe_AR(1)

	A	B	C	D	E	F	G
1		Day1	Day2	NextDay			
2	1	88.6	106	114.8			
3	2	106	114.8	109			
4	3	114.8	109	102.4			
5	4	109	102.4	87.2			
6	5	102.4	87.2	67.6			
7	6	87.2	67.6	116.2			
8	7	67.6	116.2	116			
9	8	116.2	116	114.2			
10	9	116	114.2	113.4			
11	10	114.2	113.4	120.4			
12	11	113.4	120.4	84.8			
13	12	120.4	84.8	66.8			
14	13	84.8	66.8	121.2			
15	14	66.8	121.2	124.6			
16	15	121.2	124.6	118.6			
17	16	124.6	118.6	133.8			
18	17	118.6	133.8	126.8			

Figure 27: Dataframe_AR(2)

	A	B	C	D	E	F	G
1		Day1	Day2	Day3	NextDay		
2	1	88.6	106	114.8	109		
3	2	106	114.8	109	102.4		
4	3	114.8	109	102.4	87.2		
5	4	109	102.4	87.2	67.6		
6	5	102.4	87.2	67.6	116.2		
7	6	87.2	67.6	116.2	116		
8	7	67.6	116.2	116	114.2		
9	8	116.2	116	114.2	113.4		
10	9	116	114.2	113.4	120.4		
11	10	114.2	113.4	120.4	84.8		
12	11	113.4	120.4	84.8	66.8		
13	12	120.4	84.8	66.8	121.2		
14	13	84.8	66.8	121.2	124.6		
15	14	66.8	121.2	124.6	118.6		
16	15	121.2	124.6	118.6	133.8		
17	16	124.6	118.6	133.8	126.8		
18	17	118.6	133.8	126.8	100.8		

Figure 28: Dataframe_AR(3)

	A	B	C	D	E	F	G
1		Day1	Day2	Day3	Day4	NextDay	
2	1	88.6	106	114.8	109	102.4	
3	2	106	114.8	109	102.4	87.2	
4	3	114.8	109	102.4	87.2	67.6	
5	4	109	102.4	87.2	67.6	116.2	
6	5	102.4	87.2	67.6	116.2	116	
7	6	87.2	67.6	116.2	116	114.2	
8	7	67.6	116.2	116	114.2	113.4	
9	8	116.2	116	114.2	113.4	120.4	
10	9	116	114.2	113.4	120.4	84.8	
11	10	114.2	113.4	120.4	84.8	66.8	
12	11	113.4	120.4	84.8	66.8	121.2	
13	12	120.4	84.8	66.8	121.2	124.6	
14	13	84.8	66.8	121.2	124.6	118.6	
15	14	66.8	121.2	124.6	118.6	133.8	
16	15	121.2	124.6	118.6	133.8	126.8	
17	16	124.6	118.6	133.8	126.8	100.8	
18	17	118.6	133.8	126.8	100.8	69.6	

Figure 29: Dataframe_AR(4)

For the NARX approach we have constructed 4 datasets using various time-delayed input vectors of the 9th, 10th, and 11th hour attributes with the help of R programming language. The following is the code snippet that has been used to create the 4 datasets for the NARX approach.

```
library(readxl)

#Loading the data set and creating the data Frame.
dataFrame <- read_excel("C:\\Users\\SEAN\\Desktop\\MachineLearningandDataMining\\MLCW\\UoW_load.xlsx")
str(dataFrame)

#"Day2_9","Day2_10","Day2_11","Day3_9","Day3_10","Day3_11","Day4_9","Day4_10","Day4_11",
columns= c("Day1_9","Day1_10","Day1_11","NextDay_9","NextDay_10","NextDay_11")

# pass this vector length to ncol parameter
# and nrow with 0
DataFrame_days = data.frame(matrix(nrow = 0, ncol = length(columns)))

# assign column names
colnames(DataFrame_days) = columns

#, dataFrame$Time9[i+2], dataFrame$Time10[i+2], dataFrame$Time11[i+2], dataFrame$Time9[i+3], dataFrame$Time10[i+3], dataFrame$Time11[i+3], dataFrame$Time
i <- 1
while(i<500){
  values <- data.frame(dataFrame$Time9[i], dataFrame$Time10[i], dataFrame$Time11[i], dataFrame$Time9[i+1], dataFrame$Time10[i+1], dataFrame$Time11[i+1])
  #Naming the Data Frame - Step 2
  names(values) <- c("Day1_9","Day1_10","Day1_11","NextDay_9","NextDay_10","NextDay_11")
  DataFrame_days <- rbind(DataFrame_days, values)
  i = i +1
}

str(DataFrame_days)
write.csv(DataFrame_days, "C:\\Users\\SEAN\\Desktop\\MachineLearningandDataMining\\MLCW\\test_DataFrame_NARX(1).csv")
```

Figure 30: For producing NARX input vectors

The names of the constructed dataset files and the details of those 4 dataset files is as follows.

- DataFrame_NARX(1) => Dataset which is responsible for predicting the next day's (t) 11th hour electricity consumption based on the (t-1) day's 9th, 10th, and 11th hour electricity consumption data and with (t) day's 9th, 10th hour electricity consumption data.
- DataFrame_NARX(2) => Dataset which is responsible for predicting the next day's (t) 11th hour electricity consumption based on the (t-1) and (t-2) day's 9th, 10th, and 11th hour electricity consumption data and with (t) day's 9th, 10th hour electricity consumption data.
- DataFrame_NARX(3) => Dataset which is responsible for predicting the next day's (t) 11th hour electricity consumption based on the (t-1), (t-2) and (t-3) day's 9th, 10th, and 11th hour electricity consumption data and with (t) day's 9th, 10th hour electricity consumption data.
- DataFrame_NARX(4) => Dataset which is responsible for predicting the next day's (t) 11th hour electricity consumption based on the (t-1), (t-2), (t-3) and (t-4) day's 9th, 10th, and 11th hour electricity consumption data and with (t) day's 9th, 10th hour electricity consumption data.

	A	B	C	D	E	F	G	H
1		Day1_9	Day1_10	Day1_11	NextDay_9	NextDay_10	NextDay_11	
2	1	89.4	90.6	88.6	108.2	104.6	106	
3	2	108.2	104.6	106	110	111.6	114.8	
4	3	110	111.6	114.8	106.4	104.4	109	
5	4	106.4	104.4	109	97.8	100.4	102.4	
6	5	97.8	100.4	102.4	87	90.8	87.2	
7	6	87	90.8	87.2	68.8	67	67.6	
8	7	68.8	67	67.6	110.2	113	116.2	
9	8	110.2	113	116.2	111.2	119.2	116	
10	9	111.2	119.2	116	109	114.6	114.2	
11	10	109	114.6	114.2	99.8	109	113.4	
12	11	99.8	109	113.4	111	114.8	120.4	
13	12	111	114.8	120.4	88.4	89.6	84.8	
14	13	88.4	89.6	84.8	63.4	63.6	66.8	
15	14	63.4	63.6	66.8	114.4	116.2	121.2	
16	15	114.4	116.2	121.2	111.2	114.6	124.6	
17	16	111.2	114.6	124.6	119.6	117.4	118.6	
18	17	119.6	117.4	118.6	117.6	123.6	133.8	
19	18	117.6	123.6	133.8	120.4	130.4	126.8	
20	19	120.4	130.4	126.8	105	101.4	100.8	

Figure 31: Dataframe_NARX(1)

	A	B	C	D	E	F	G	H	I	J	K
1		Day1_9	Day1_10	Day1_11	Day2_9	Day2_10	Day2_11	NextDay_9	NextDay_10	NextDay_11	
2	1	89.4	90.6	88.6	108.2	104.6	106	110	111.6	114.8	
3	2	108.2	104.6	106	110	111.6	114.8	106.4	104.4	109	
4	3	110	111.6	114.8	106.4	104.4	109	97.8	100.4	102.4	
5	4	106.4	104.4	109	97.8	100.4	102.4	87	90.8	87.2	
6	5	97.8	100.4	102.4	87	90.8	87.2	68.8	67	67.6	
7	6	87	90.8	87.2	68.8	67	67.6	110.2	113	116.2	
8	7	68.8	67	67.6	110.2	113	116.2	111.2	119.2	116	
9	8	110.2	113	116.2	111.2	119.2	116	109	114.6	114.2	
10	9	111.2	119.2	116	109	114.6	114.2	99.8	109	113.4	
11	10	109	114.6	114.2	99.8	109	113.4	111	114.8	120.4	
12	11	99.8	109	113.4	111	114.8	120.4	88.4	89.6	84.8	
13	12	111	114.8	120.4	88.4	89.6	84.8	63.4	63.6	66.8	
14	13	88.4	89.6	84.8	63.4	63.6	66.8	114.4	116.2	121.2	
15	14	63.4	63.6	66.8	114.4	116.2	121.2	111.2	114.6	124.6	
16	15	114.4	116.2	121.2	111.2	114.6	124.6	119.6	117.4	118.6	
17	16	111.2	114.6	124.6	119.6	117.4	118.6	117.6	123.6	133.8	
18	17	119.6	117.4	118.6	117.6	123.6	133.8	120.4	130.4	126.8	
19	18	117.6	123.6	133.8	120.4	130.4	126.8	105	101.4	100.8	
20	19	120.4	130.4	126.8	105	101.4	100.8	67.6	68	69.6	
21	20	105	101.4	100.8	67.6	68	69.6	133.8	132.4	138.4	
22	21	67.6	68	69.6	133.8	132.4	138.4	126	129	138.4	
23	22	133.8	132.4	138.4	126	129	138.4	119.2	129.4	133.8	
24	23	126	129	138.4	119.2	129.4	133.8	126.4	130.4	142.4	
25	24	119.2	129.4	133.8	126.4	130.4	142.4	124.6	124	135.6	

Figure 32: Dataframe_NARX(2)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		Day1_9	Day1_10	Day1_11	Day2_9	Day2_10	Day2_11	Day3_9	Day3_10	Day3_11	NextDay_	NextDay_	NextDay_11	
2	1	89.4	90.6	88.6	108.2	104.6	106	110	111.6	114.8	106.4	104.4	109	
3	2	108.2	104.6	106	110	111.6	114.8	106.4	104.4	109	97.8	100.4	102.4	
4	3	110	111.6	114.8	106.4	104.4	109	97.8	100.4	102.4	87	90.8	87.2	
5	4	106.4	104.4	109	97.8	100.4	102.4	87	90.8	87.2	68.8	67	67.6	
6	5	97.8	100.4	102.4	87	90.8	87.2	68.8	67	67.6	110.2	113	116.2	

Figure 33 Dataframe_NARX(3)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1		Day1_9	Day1_10	Day1_11	Day2_9	Day2_10	Day2_11	Day3_9	Day3_10	Day3_11	Day4_9	Day4_10	Day4_11	NextDay_9	NextDay_10	NextDay_11	
2	1	89.4	90.6	88.6	108.2	104.6	106	110	111.6	114.8	106.4	104.4	109	97.8	100.4	102.4	
3	2	108.2	104.6	106	110	111.6	114.8	106.4	104.4	109	97.8	100.4	102.4	87	90.8	87.2	
4	3	110	111.6	114.8	106.4	104.4	109	97.8	100.4	102.4	87	90.8	87.2	68.8	67	67.6	
5	4	106.4	104.4	109	97.8	100.4	102.4	87	90.8	87.2	68.8	67	67.6	110.2	113	116.2	
6	5	97.8	100.4	102.4	87	90.8	87.2	68.8	67	67.6	110.2	113	116.2	111.2	119.2	116	
7	6	87	90.8	87.2	68.8	67	67.6	110.2	113	116.2	111.2	119.2	116	109	114.6	114.2	
8	7	68.8	67	67.6	110.2	113	116.2	111.2	119.2	116	109	114.6	114.2	99.8	109	113.4	
9	8	110.2	113	116.2	111.2	119.2	116	109	114.6	114.2	99.8	109	113.4	111	114.8	120.4	
10	9	111.2	119.2	116	109	114.6	114.2	99.8	109	113.4	111	114.8	120.4	88.4	89.6	84.8	

Figure 34: Dataframe_NARX(4)

2. Data Normalization

Since we know that there are 2 possible ways that we can use to scale our data. One way is Normalization and the other way is Standardization. If the dataset does not have a normal or more or less normal distribution for some feature, Standardization may not be the most suitable method. So it's always best to Normalize the dataset for neural network problems. So here we will use the min-max Normalization to scale each feature in the range between 0 and 1.

The following code snippets shows how we normalize our given dataset in order to build neural networks for both AR and NARX approaches.

```
> #Normalizing the Data
> normalization <- function(x) {
+   (x - min(x))/(max(x) - min(x))
+ }
>
> #apply Min-Max normalization.
> DataFrame_days_scale <- data.frame(lapply(DataFrame_days[1:columnNumbers], normalization))
> head(DataFrame_days_scale)
       Day1      Day2      Day3      Day4    NextDay
1 0.3733826 0.5341959 0.6155268 0.5619224 0.5009242
2 0.5341959 0.6155268 0.5619224 0.5009242 0.3604436
3 0.6155268 0.5619224 0.5009242 0.3604436 0.1792976
4 0.5619224 0.5009242 0.3604436 0.1792976 0.6284658
5 0.5009242 0.3604436 0.1792976 0.6284658 0.6266174
6 0.3604436 0.1792976 0.6284658 0.6266174 0.6099815
>
```

Figure 35: min-max normalization for dataFrame_AR(4)

```
> #Normalizing the Data
> normalization <- function(x) {
+   (x - min(x))/(max(x) - min(x))
+ }
>
> #apply Min-Max normalization.
> DataFrame_days_scale <- data.frame(lapply(DataFrame_days[1:columnNumbers], normalization))
> head(DataFrame_days_scale)
    Day1_9   Day1_10   Day1_11    Day2_9   Day2_10   Day2_11    Day3_9   Day3_10   Day3_11    Day4_9   Day4_10   Day4_11 NextDay_9 NextDay_10 NextDay_11
1 0.4295154 0.4156627 0.3733826 0.6365639 0.5562249 0.5341959 0.6563877 0.6265060 0.6155268 0.6167401 0.5542169 0.5619224 0.5220264  0.5140562  0.5009242
2 0.6365639 0.5562249 0.5341959 0.6563877 0.6265060 0.6155268 0.6167401 0.5542169 0.5619224 0.5220264 0.5140562 0.5009242 0.4030837  0.4176707  0.3604436
3 0.6563877 0.6265060 0.6155268 0.6167401 0.5542169 0.5619224 0.5220264 0.5140562 0.5009242 0.4030837 0.4176707 0.3604436 0.2026432  0.1787149  0.1792976
4 0.6167401 0.5542169 0.5619224 0.5220264 0.5140562 0.5009242 0.4030837 0.4176707 0.3604436 0.2026432 0.1787149 0.1792976 0.6585903  0.6405622  0.6284658
5 0.5220264 0.5140562 0.5009242 0.4030837 0.4176707 0.3604436 0.2026432 0.1787149 0.1792976 0.6585903 0.6405622 0.6284658 0.6696035  0.7028112  0.6266174
6 0.4030837 0.4176707 0.3604436 0.2026432 0.1787149 0.1792976 0.6585903 0.6405622 0.6284658 0.6696035 0.7028112 0.6266174 0.6453744  0.6566265  0.6099815
>
```

Figure 36: min-max normalization for dataFrame_NARX(4)

Normalization/scaling is typically recommended and sometimes very important. Especially for neural networks, normalization can be very crucial because when you input unnormalised inputs to activation functions, you can get stuck in a very flat region in the domain and may not learn at all. Or worse, you can end up with numerical issues.

3. MLP Neural Network Analysis

1. MLP Neural Network Analysis for AR approach

Dataset Name (contains input vectors)	No of hidden layers and nodes in each layer	Learning rate	Activation Function	Testing Performance Analysis		
				RMSE	MAE	MAPE
DataFrame_AR(1)	1 (5)	0.001	logistic	19.69127	16.9423	0.179792
DataFrame_AR(1)	2 (8, 5)	0.001	tanh	19.63556	16.62699	0.1755653
DataFrame_AR(2)	1 (5)	0.001	tanh	16.54231	13.74638	0.1431922
DataFrame_AR(2)	2 (9, 6)	0.0001	logistic	15.98554	13.40043	0.1409552
DataFrame_AR(3)	1 (6)	0.0001	tanh	14.77517	12.44712	0.1304804
DataFrame_AR(3)	2 (8, 4)	0.001	logistic	14.77917	12.53184	0.128366
DataFrame_AR(4)	1 (7)	0.0005	logistic	14.48904	11.89507	0.1256197
DataFrame_AR(4)	2 (9, 5)	0.0001	logistic	14.06588	11.33218	0.119508

Tabel 3: MLP Neural Network Analysis for AR approach

2. MLP Neural Network Analysis for NARX approach

Dataset Name (contains input vectors)	No of hidden layers and nodes in each layer	Learning rate	Activation Function	Testing Performance Analysis		
				RMSE	MAE	MAPE
DataFrame_NARX(1)	1 (5)	0.001	logistic	9.251488	7.632887	0.06884571
DataFrame_NARX(1)	2 (8, 5)	0.001	tanh	9.137165	7.529608	0.06885017
DataFrame_NARX(2)	1 (10)	0.001	tanh	9.188893	7.633953	0.06958234
DataFrame_NARX(2)	2 (9, 6)	0.0001	logistic	8.961525	7.341603	0.06627507
DataFrame_NARX(3)	1 (6)	0.0001	tanh	8.97989	7.390405	0.06751549
DataFrame_NARX(3)	2 (9, 5)	0.001	logistic	8.681214	7.128098	0.0646964
DataFrame_NARX(4)	1 (13)	0.0005	logistic	8.802666	7.330058	0.0671484
DataFrame_NARX(4)	2 (14, 8)	0.0001	logistic	8.135052	6.666759	0.06102992

Tabel 4: MLP Neural Network Analysis for NARX approach

In the above tables there are three standard statistical indices,

RMSE - Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

MAE - Mean Absolute Error of your model refers to the mean of the absolute values of each prediction error on all instances of the test dataset. Prediction error is the difference between the actual value and the predicted value for that instance. Statistically, Mean Absolute Error (MAE) refers to the results of measuring the difference between two continuous variables. Let us assume variables M and N represent the same phenomenon but have recorded different observations.

MAPE - Mean Absolute Percentage Error (MAPE) is a statistical measure to define the accuracy of a machine learning algorithm on a particular dataset. MAPE can be considered as a loss function to define the error termed by the model evaluation. Using MAPE, we can estimate the accuracy in terms of the differences in the actual v/s estimated values.(MAPE - Mean Absolute Percentage Error in Python - AskPython, no date)

4. Choosing the best MLP Network

As you can see from both “AR” and “NARX” MLP Neural Network comparison tables the green color highlighted ones are the best 1 hidden layer MLP networks and the red color highlighted ones are the best 2 hidden layer MLP networks.

As you can see, that NARX approach gives relatively low values for the RMSE, MAE and MAPE. It means that the NARX approach is performing really well when compared to AR approach for this specific time series problem.

And the above-mentioned 2 hidden layer best MLP network from the NARX Network Network comparison table gives relatively small values for the RMSE, MAE and MAPE. which means that this MLP Neural Network was the best MLP Neural Network out of all the networks from both tables.

The following picture shows the best MLP Neural Network.

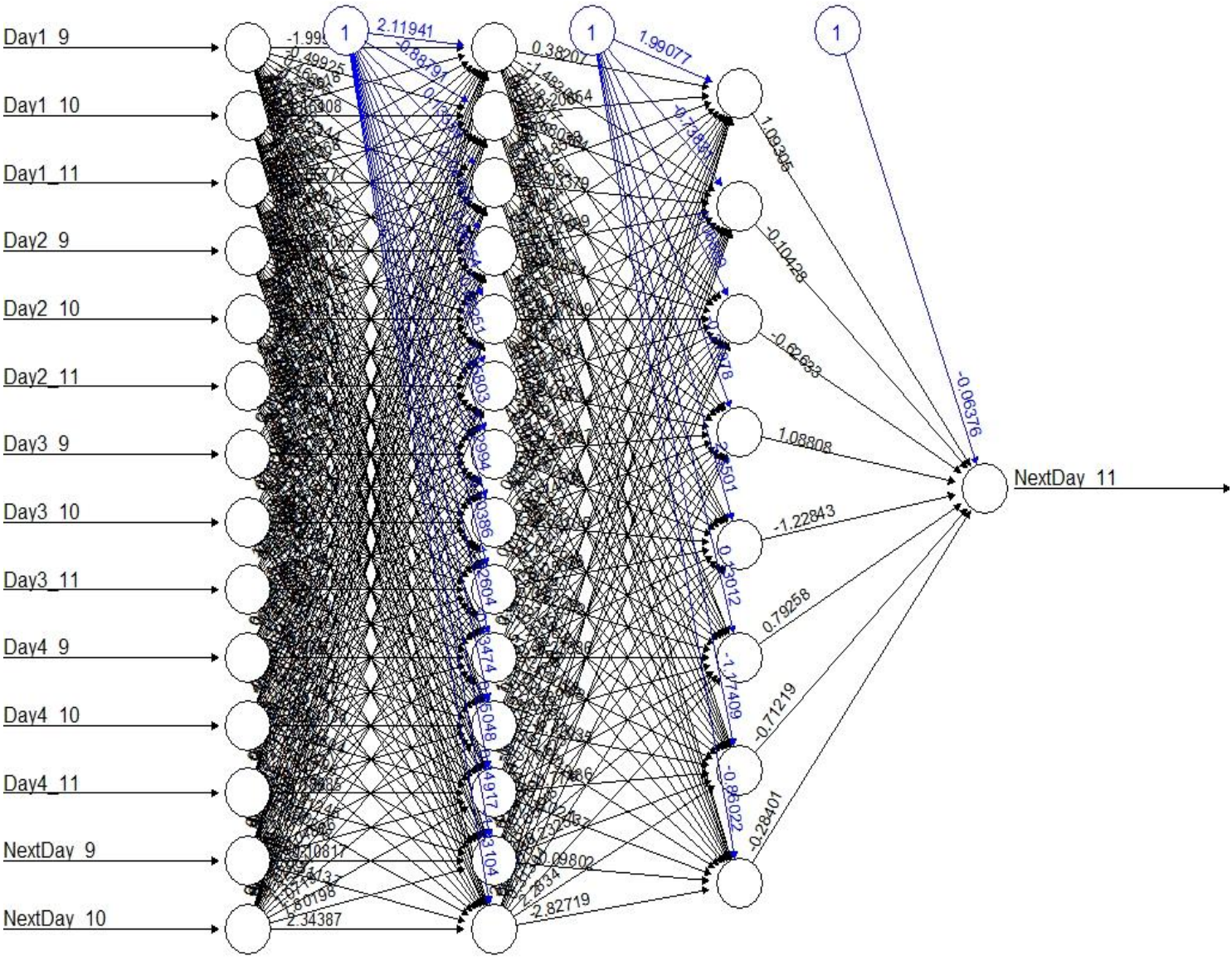

Figure 37: Best MLP Neural Network

Best MLP Neural Network Analysis

- RMSE => 8.135052
- MAE => 6.666759
- MAPE => 0.06102992

And the following picture shows the details about the (desired output vs predicted output) from the testing dataset.

	desired_Output	predicted_Output
1	113.6	105.78411
2	100.8	103.99796
3	124.0	116.31028
4	124.8	116.92487
5	119.0	108.96205
6	89.4	83.36382
7	58.2	55.30111
8	133.8	121.51773
9	124.0	116.03632
10	126.2	124.01003
11	121.8	115.70094
12	123.8	109.16038
13	95.6	83.92985
14	54.2	53.24203
15	125.8	118.54160
16	128.2	118.74230
17	116.2	118.74050
18	123.8	112.92986
19	112.8	104.89179
20	106.4	96.02626
21	51.4	48.64689
22	129.4	116.39928
23	137.0	118.40428
24	132.4	132.14659
25	140.8	115.04468
26	125.2	116.57004
27	94.0	87.87206
28	55.4	55.56579
29	137.8	125.50484
30	127.2	121.26555
31	131.0	127.77405
32	132.0	123.68893
33	127.8	117.33286
34	100.2	91.69613
35	53.6	59.26443
36	108.4	109.66037
37	120.0	108.75235
38	106.8	98.87220
39	103.0	102.02377
40	89.6	85.29034
41	88.4	83.06310
42	48.2	50.21511
43	103.4	97.03598
44	125.6	114.60018
45	100.0	104.23727
46	98.2	93.26594
47	92.4	85.81420
48	80.8	74.41106

Figure 38: Desired Output vs Predicted Output

Appendix - Clustering Part Code

```r

library(MASS)
library(NbClust)
library(factoextra)
library(ClusterR)
library(cluster)
library(caret)
library(dplyr)
library(psych)
library(readxl)

#########################################Loading the Data set#################################################################
wine_DataFrame <- read_excel("C:\\Users\\SEAN\\Desktop\\MachineLearningandDataMining\\MLCW\\Whitewine_v2.xlsx")
wine_DataFrame <- data.frame(wine_DataFrame)
str(wine_DataFrame)

#####################################Visualization of Outliers in each features of the data set#########################
columnNames <- names(wine_DataFrame)
oldpar <- par(mfrow = c(2,6))
for (i in 1:11){
  boxplot(wine_DataFrame[i], main = columnNames[i], col = "#DD3300")
}
par(oldpar)

#############################################visualizing how the distribution is looks like#############################
oldpar <- par(mfrow = c(2,6))
for (i in 1:12){
  truehist(wine_DataFrame[[i]], main = columnNames[i], col = blues9)
  lines(density(wine_DataFrame[[i]]), col= 2, lwd=2)
}
par(oldpar)

#########################################################Removal of Outliers#################################################
# #The returned stats variable is a vector of length 5, containing the extreme of the lower whisker, the lower 'hinge', the median,
for (i in 1:11){
  stats <- boxplot.stats(wine_DataFrame[[i]], coef = 1.4)$stats
  #print(stats)
  lower_boundry <- stats[1]
  #print(lower_boundry)
  upper_boundry <- stats[5]
  #print(upper_boundry)
  wine_DataFrame <- wine_DataFrame[!(wine_DataFrame[i] < lower_boundry | wine_DataFrame[i] > upper_boundry),]
  #str(dataFrame1)
}
```

```r

##########################Visualization of each feature after outlier removal of the data set#############################
oldpar <- par(mfrow = c(2,6))
for (i in 1:11){
  boxplot(wine_DataFrame[i], main = columnNames[i], col = "#4DB3E6")
}
par(oldpar)

#################################visualizing how the distribution is looks like after Outlier Removal#####################
oldpar <- par(mfrow = c(2,6))
for (i in 1:11){
  truehist(wine_DataFrame[[i]], main = columnNames[i], col = blues9)
  lines(density(wine_DataFrame[[i]]), col= 2, lwd=2)
}
par(oldpar)

str(tail(wine_DataFrame))


#################################################Normalizing the Data#################################################

#define Min-Max normalization function
normalization <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}


#apply Min-Max normalization to first eleven columns in wine data-set.
wine_DataFrame_without_quality <- data.frame(lapply(wine_DataFrame[1:11], normalization))
str(wine_DataFrame_without_quality)
summary(wine_DataFrame_without_quality)

#combining the quality with the normalized data.
quality <- wine_DataFrame$quality
wine_DataFrame_norm <- cbind(wine_DataFrame_without_quality, quality)
str(wine_DataFrame_norm)


##########################Visualization of each feature after Normalization of the data set#############################
oldpar <- par(mfrow = c(2,6))
for (i in 1:11){
  boxplot(wine_DataFrame_norm[i], main = columnNames[i], col = "#4DC6E6")
}
par(oldpar)
```

```r
# Elbow method
fviz_nbclust(wine_DataFrame_norm[,-12], kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype = 2)+
  labs(subtitle = "Elbow method")

# Silhouette method
dataFrame3 <- sample_frac(wine_DataFrame_norm, 0.65)
str(dataFrame3)

fviz_nbclust(dataFrame3[,-12], kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method")

# Gap statistic
# nboot = 50 to keep the function speedy.
# recommended value: nboot= 500 for your analysis.
# Use verbose = FALSE to hide computing progression.
fviz_nbclust(wine_DataFrame_norm[,-12], kmeans, nstart = 25,  method = "gap_stat", nboot = 500)+
  labs(subtitle = "Gap statistic method")

# Nbclust Method
nb <- NbClust(wine_DataFrame_norm[,-12], distance = "euclidean", min.nc = 2,
              max.nc = 10, method = "kmeans")
fviz_nbclust(nb)

#################################################K-Means clustering Model#########################################
str(wine_DataFrame_norm)
set.seed(5005)
KM <- kmeans(wine_DataFrame_norm[,-12], centers = 4, nstart = 25)
KM
KM$tot.withinss
KM$betweenss
KM$totss

#############################################Visualizing the clustering algorithm results#############################
str(wine_DataFrame_norm)
temp_dataFrame <- wine_DataFrame_norm[,-12]
str(temp_dataFrame)

KM.clusters <- KM$cluster
rownames(temp_dataFrame)<- paste(wine_DataFrame_norm$quality, 1:dim(wine_DataFrame_norm)[1], sep = "_")
str(temp_dataFrame)
tail(temp_dataFrame)

fviz_cluster(list(data = temp_dataFrame, cluster = KM.clusters))
```

```r
############################################Setting the labels#####################################################
wine_dataFrame_labels <- wine_DataFrame_norm$quality
table(wine_dataFrame_labels)
tail(wine_dataFrame_labels)

#####################################to find which cluster is more describing the labels#############################
table(KM.clusters, wine_dataFrame_labels)


############################################ Confusion matrix ############################################################
cluster_wine_DataFrame_norm <- wine_DataFrame_norm
str(cluster_wine_DataFrame_norm)


cluster_wine_DataFrame_norm['quality'][cluster_wine_DataFrame_norm['quality'] == 5] <- 1
cluster_wine_DataFrame_norm['quality'][cluster_wine_DataFrame_norm['quality'] == 6] <- 3
cluster_wine_DataFrame_norm['quality'][cluster_wine_DataFrame_norm['quality'] == 7] <- 2
cluster_wine_DataFrame_norm['quality'][cluster_wine_DataFrame_norm['quality'] == 8] <- 4

head(cluster_wine_DataFrame_norm)

#add cluster assigment to original data.
final_data <- cbind(cluster_wine_DataFrame_norm, cluster = KM$cluster)

#view final data
head(final_data)
tail(final_data)


#Creates vectors having data points
expected_value <- factor(final_data$quality)
predicted_value <- factor(final_data$cluster)

#Creating confusion matrix
confusion_Matrix <- confusionMatrix(predicted_value, expected_value)
confusion_Matrix

################################################ PCA ###############################################
str(wine_DataFrame)
dataFrame1_scale <- as.data.frame(scale(wine_DataFrame[1 : 11]))
str(dataFrame1_scale)

str(wine_DataFrame_norm)
```

```r
#PCA APPLYING..........
PCA_components <- prcomp(~fixed.acidity + volatile.acidity + citric.acid
                         + residual.sugar + chlorides + free.sulfur.dioxide
                         + total.sulfur.dioxide + density + pH + sulphates +
                           alcohol, data = wine_DataFrame_norm,
                         scale. = T)
summary(PCA_components)

PCA_components$sdev

VE <- PCA_components$sdev^2
PVE <- VE / sum(VE)
round(PVE, 2)

screeplot(PCA_components, type='l', main="Screeplot for principle components")


trnfrm_PCA_components <- as.data.frame(PCA_components$x[,1:8])
head(trnfrm_PCA_components)

#Setting seed
set.seed(1600)
KM.PCA <- kmeans(trnfrm_PCA_components, centers = 4, nstart = 25)
KM.PCA
KM.PCA$tot.withinss
KM.PCA$betweenss
KM.PCA$totss

#############################################################################################
#Visualizing the clustering algorithm results
KM.PCA.clusters <- KM.PCA$cluster
fviz_cluster(list(data = trnfrm_PCA_components, cluster = KM.PCA.clusters))

#############################################################################################
```

Appendix - Energy Forecasting Part Code

(AR) approach

```r

library(readxl)
library(neuralnet)
library(Metrics)
library(MLmetrics)

DataFrame_days <- read.csv("C:\\Users\\SEAN\\Desktop\\MachineLearningandDataMining\\MLCW\\DataFrame_AR(4).csv")
DataFrame_days <- DataFrame_days[-(1)]
str(DataFrame_days)

length_DataFrame_days <- nrow(DataFrame_days)
length_DataFrame_days

columnNumbers <- length(DataFrame_days)
columnNumbers

#Testing Data set with No scale.
Data_test_NoScale = DataFrame_days[431:length_DataFrame_days,]
str(Data_test_NoScale)
head(Data_test_NoScale)

#Normalizing the Data
normalization <- function(x) {
  (x - min(x))/(max(x) - min(x))
}

#apply Min-Max normalization.
DataFrame_days_scale <- data.frame(lapply(DataFrame_days[1:columnNumbers], normalization))
head(DataFrame_days_scale)

#Training Data set
Data_train_scale = DataFrame_days_scale[1:430,]
str(Data_train_scale)

#Testing Data set
Data_test_scale = DataFrame_days_scale[431:length_DataFrame_days,]
str(Data_test_scale)
head(Data_test_scale)



```

```r

#Building the MLP neural network
set.seed(4444)
nn <- neuralnet(NextDay ~., data = Data_train_scale, hidden = c(9,5),
                learningrate = 0.0001, act.fct = "logistic",
                linear.output = TRUE)
plot(nn)
nn$result.matrix

#Test the resulting output
temp_Data_test_scale <- Data_test_scale[1:columnNumbers-1]
head(temp_Data_test_scale)
str(temp_Data_test_scale)

#Prediction using our model
nn.results <- compute(nn, temp_Data_test_scale)
nn.results$net.result

#validation
results <- data.frame(actual = Data_test_scale$NextDay, prediction = nn.results$net.result)
results

actual = Data_test_NoScale$NextDay
head(actual)

predicted = (results$prediction * (max(Data_test_NoScale$NextDay) - min(Data_test_NoScale$NextDay))) + min(Data_test_NoScale$NextDay)
head(predicted)
head(Data_test_NoScale$NextDay)

class(actual)
class(predicted)

#calculate RMSE (Root Mean Square Error)
rmse(actual, predicted)

#calculate MAE (Mean Absolute Error)
mae(actual, predicted)

#calculate MAPE (Mean Absolute Percentage Error)
MAPE(predicted, actual)

```

(NARX) approach

```r

library(readxl)
library(neuralnet)
library(Metrics)
library(MLmetrics)

DataFrame_days <- read.csv("C:\\Users\\SEAN\\Desktop\\MachineLearningandDataMining\\MLCW\\DataFrame_NARX(4).csv")
DataFrame_days <- DataFrame_days[-(1)]
str(DataFrame_days)

length_DataFrame_days <- nrow(DataFrame_days)
length_DataFrame_days

columnNumbers <- length(DataFrame_days)
columnNumbers

#Testing Data set with No scale.
Data_test_NoScale = DataFrame_days[431:length_DataFrame_days,]
str(Data_test_NoScale)
head(Data_test_NoScale)

#Normalizing the Data
normalization <- function(x) {
  (x - min(x))/(max(x) - min(x))
}

#apply Min-Max normalization.
DataFrame_days_scale <- data.frame(lapply(DataFrame_days[1:columnNumbers], normalization))
head(DataFrame_days_scale)

#Training Data set
Data_train_scale = DataFrame_days_scale[1:430,]
str(Data_train_scale)

#Testing Data set
Data_test_scale = DataFrame_days_scale[431:length_DataFrame_days,]
str(Data_test_scale)
head(Data_test_scale)



```

```r
#Building the MLP neural network
set.seed(4444)
nn <- neuralnet(NextDay_11 ~., data = Data_train_scale, hidden = c(14,8),
                learningrate = 0.0001, act.fct = "logistic",
                linear.output = TRUE)
plot(nn)
nn$result.matrix

#Test the resulting output
temp_Data_test_scale <- Data_test_scale[1:columnNumbers-1]
head(temp_Data_test_scale)
str(temp_Data_test_scale)

#Prediction using our model
nn.results <- compute(nn, temp_Data_test_scale)
nn.results$net.result

#validation
results <- data.frame(actual = Data_test_scale$NextDay_11, prediction = nn.results$net.result)
results

actual = Data_test_NoScale$NextDay_11
head(actual)

predicted = (results$prediction * (max(Data_test_NoScale$NextDay_11) - min(Data_test_NoScale$NextDay_11))) + min(Data_test_NoScale$NextDay_11)
head(predicted)
head(Data_test_NoScale$NextDay_11)

class(actual)
class(predicted)

#validation
results <- data.frame(desired_Output = actual, predicted_Output = predicted)
results

#calculate RMSE (Root Mean Square Error)
rmse(actual, predicted)

#calculate MAE (Mean Absolute Error)
mae(actual, predicted)

#calculate MAPE (Mean Absolute Percentage Error)
MAPE(predicted, actual)

```