UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

7BUIS008C.2

Data Mining and Machine Learning

Coursework 1

Name: Asiri Mevan Senanayake
IIT ID: 20221919
UOW ID: w1986425

Table of Content

Objective 1 - Clustering

1.1 Pre-processing tasks

Pre-processing of data is critical. We must recognize the characteristics in the dataset and process the values that may be used in the analysis. Outlier removal is also an essential aspect of data preparation. The code below shows how the "Whitewine_v2.xlsx" file was read in R, and the screenshot below shows the summary created for the dataset.

```
> # Read in the initial excel datafile
> whitewine_initial <- read_excel("Whitewine_v2.xlsx") %>%
+   janitor::clean_names() %>%
+   mutate(class = as_factor(quality))
> whitewine_initial
```

By choosing quality as the dataset's class, we are indicating that the dataset is labeled according to the level of quality, with each data point belonging to one of the quality classes.

```
> # Get an overview of what the dataset looks like from a high-level perspective.
> summary(whitewine_initial)
 fixed_acidity    volatile_acidity  citric_acid     residual_sugar     chlorides       free_sulfur_dioxide
 Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600   Min.   :0.00900   Min.   :  2.00
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700   1st Qu.:0.03600   1st Qu.: 24.00
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.300   Median :0.04300   Median : 34.00
 Mean   : 6.842   Mean   :0.2744   Mean   :0.3352   Mean   : 6.455   Mean   :0.04561   Mean   : 35.65
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.:10.000   3rd Qu.:0.05000   3rd Qu.: 46.00
 Max.   :14.200   Max.   :0.9650   Max.   :1.6600   Max.   :65.800   Max.   :0.34600   Max.   :131.00
 total_sulfur_dioxide    density             p_h           sulphates         alcohol          quality      class
 Min.   :  9.0        Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :5.000   5:1457
 1st Qu.:109.0        1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50   1st Qu.:5.000   6:2198
 Median :134.0        Median :0.9937   Median :3.180   Median :0.4800   Median :10.40   Median :6.000   7: 880
 Mean   :138.7        Mean   :0.9940   Mean   :3.188   Mean   :0.4904   Mean   :10.53   Mean   :5.952   8: 175
 3rd Qu.:167.0        3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.000
 Max.   :344.0        Max.   :1.0390   Max.   :3.820   Max.   :1.0800   Max.   :14.20   Max.   :8.000
```

Structure of dataset

```
> #Structure of the dataset
> str(whitewine_initial)
tibble [4,710 × 13] (S3: tbl_df/tbl/data.frame)
 $ fixed_acidity       : num [1:4710] 8.1 8.6 7.9 8.3 6.5 7.6 5.8 7.3 6.5 7.3 ...
 $ volatile_acidity    : num [1:4710] 0.27 0.23 0.18 0.42 0.31 0.67 0.27 0.28 0.39 0.24 ...
 $ citric_acid         : num [1:4710] 0.41 0.4 0.37 0.62 0.14 0.14 0.2 0.43 0.23 0.39 ...
 $ residual_sugar      : num [1:4710] 1.45 4.2 1.2 19.25 7.5 ...
 $ chlorides           : num [1:4710] 0.033 0.035 0.04 0.04 0.044 0.074 0.044 0.08 0.051 0.057 ..
.
 $ free_sulfur_dioxide : num [1:4710] 11 17 16 41 34 25 22 21 25 45 ...
 $ total_sulfur_dioxide: num [1:4710] 63 109 75 172 133 168 179 123 149 149 ...
 $ density             : num [1:4710] 0.991 0.995 0.992 1 0.996 ...
 $ p_h                 : num [1:4710] 2.99 3.14 3.18 2.98 3.22 3.05 3.37 3.19 3.24 3.21 ...
 $ sulphates           : num [1:4710] 0.56 0.53 0.63 0.67 0.5 0.51 0.37 0.42 0.35 0.36 ...
 $ alcohol             : num [1:4710] 12 9.7 10.8 9.7 9.5 9.3 10.2 12.8 10 8.6 ...
 $ quality             : num [1:4710] 5 5 5 5 5 5 5 5 5 5 ...
 $ class               : Factor w/ 4 levels "5","6","7","8": 1 1 1 1 1 1 1 1 1 1 ...
```

After summarizing the above picture shows the dataset.

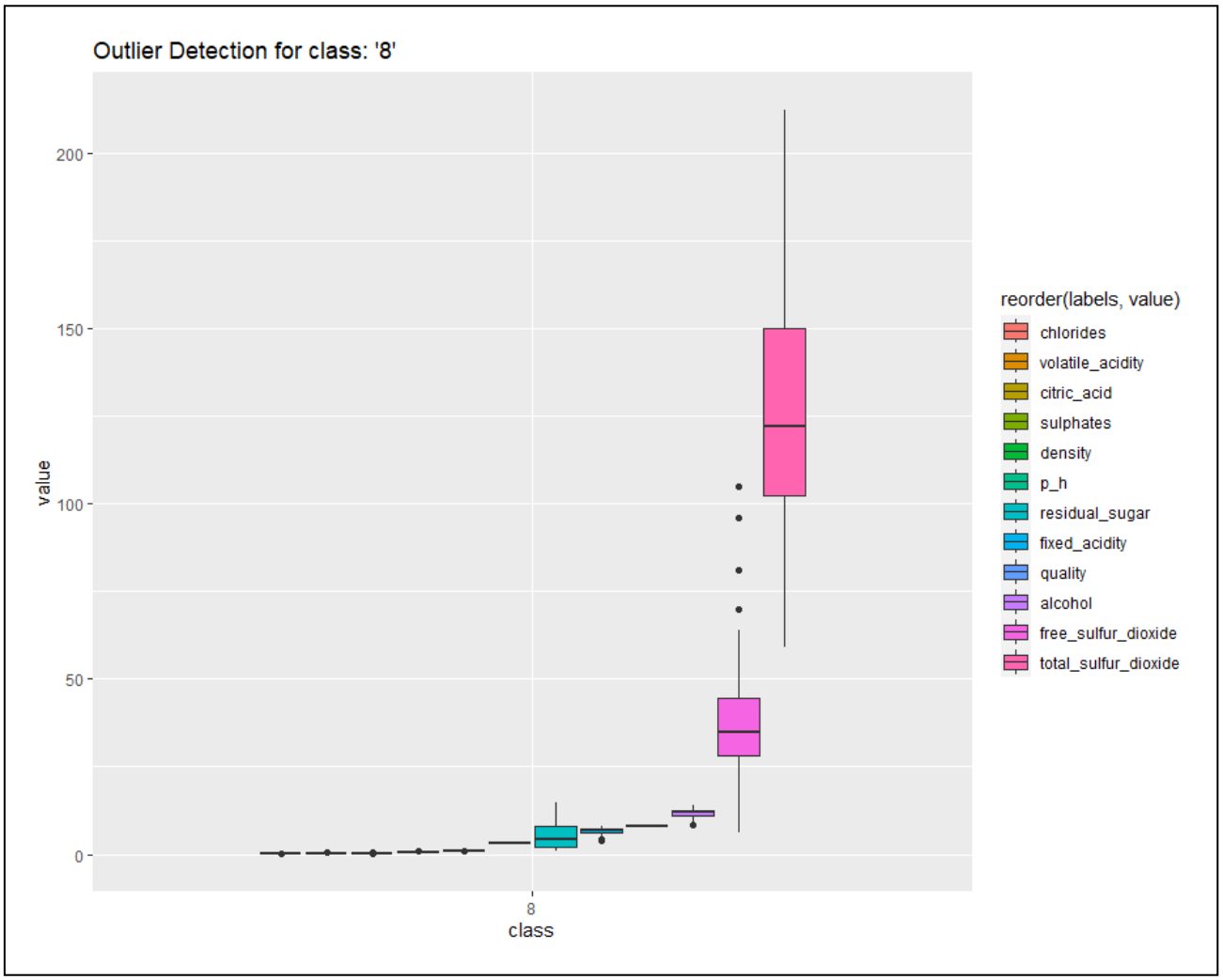
```
> whitewine_new <- whitewine_initial %>% mutate(class = as_factor(
+   case_when(
+     quality == 5 ~ 5,
+     quality == 6 ~ 6,
+     quality == 7 ~ 7,
+     quality == 8 ~ 8)
+ ))
>
> summary(whitewine_new)
 fixed_acidity    volatile_acidity  citric_acid     residual_sugar     chlorides       free_sulfur_dioxide
 Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600   Min.   :0.00900   Min.   :  2.00
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700   1st Qu.:0.03600   1st Qu.: 24.00
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.300   Median :0.04300   Median : 34.00
 Mean   : 6.842   Mean   :0.2744   Mean   :0.3352   Mean   : 6.455   Mean   :0.04561   Mean   : 35.65
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.:10.000   3rd Qu.:0.05000   3rd Qu.: 46.00
 Max.   :14.200   Max.   :0.9650   Max.   :1.6600   Max.   :65.800   Max.   :0.34600   Max.   :131.00
 total_sulfur_dioxide    density             p_h           sulphates         alcohol         quality      class
 Min.   :  9.0        Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :5.000   5:1457
 1st Qu.:109.0        1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50   1st Qu.:5.000   6:2198
 Median :134.0        Median :0.9937   Median :3.180   Median :0.4800   Median :10.40   Median :6.000   7: 880
 Mean   :138.7        Mean   :0.9940   Mean   :3.188   Mean   :0.4904   Mean   :10.53   Mean   :5.952   8: 175
 3rd Qu.:167.0        3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.000
 Max.   :344.0        Max.   :1.0390   Max.   :3.820   Max.   :1.0800   Max.   :14.20   Max.   :8.000
```

We used switch-case to replace the classes and summaries of the new dataset presented above.

Outlier Detection

The following R code was used to display the dataset filtered by class "8" in order to highlight potential outliers discovered in the dataset.

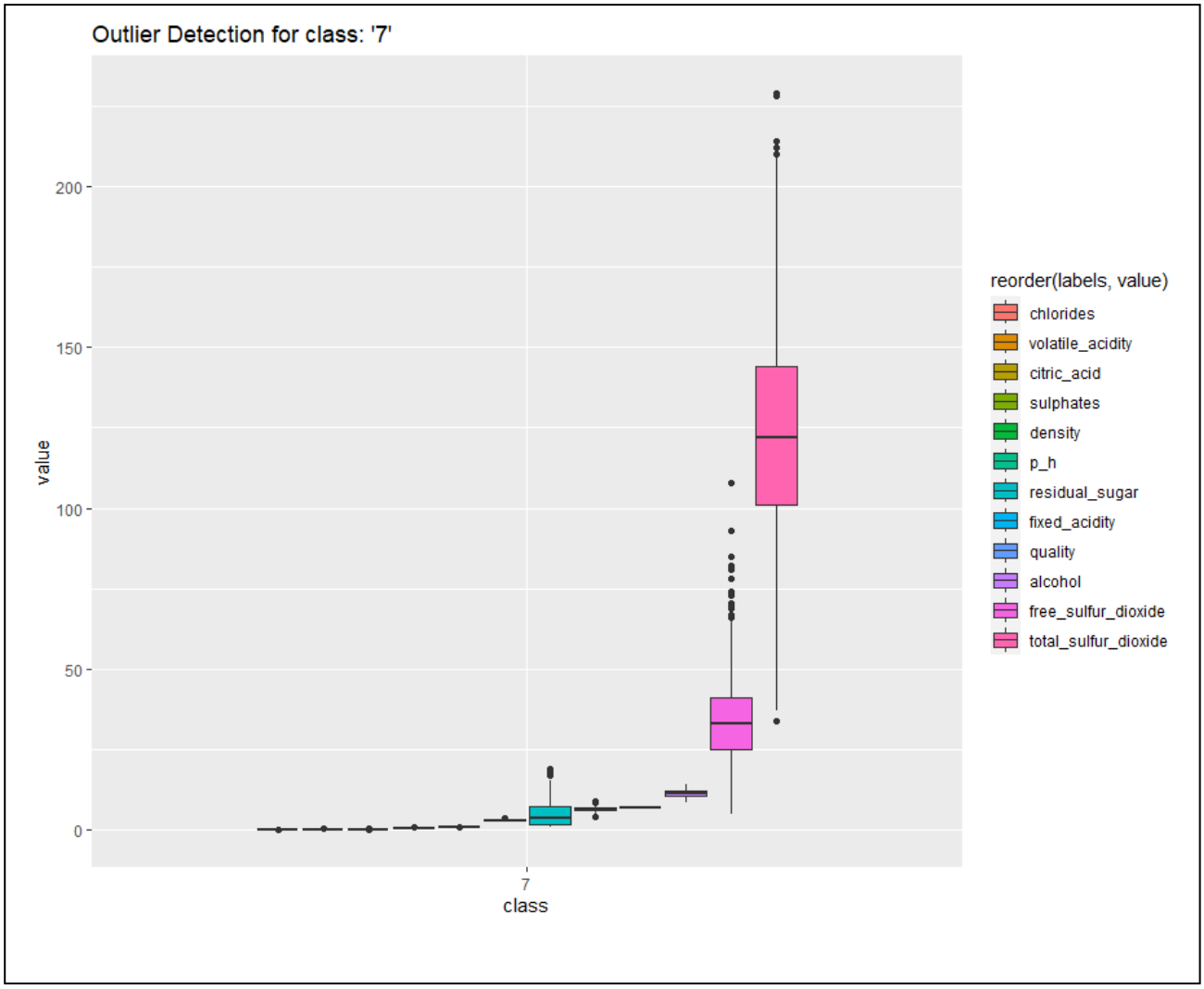
```
> whitewine_new %>%
+   pivot_longer(1:12,names_to = "labels") %>%
+   filter(class == 8) %>%
+   mutate(class = fct_reorder(class,value,median)) %>%
+   ggplot(aes(class, value, fill = reorder(labels,value))) +
+   geom_boxplot() +
+   labs(title = "Outlier Detection for class: '8'")
```



Based on the visualization, we can identify that the attribute free_sulfur_dioxide has more outliers than the rest of the attributes.

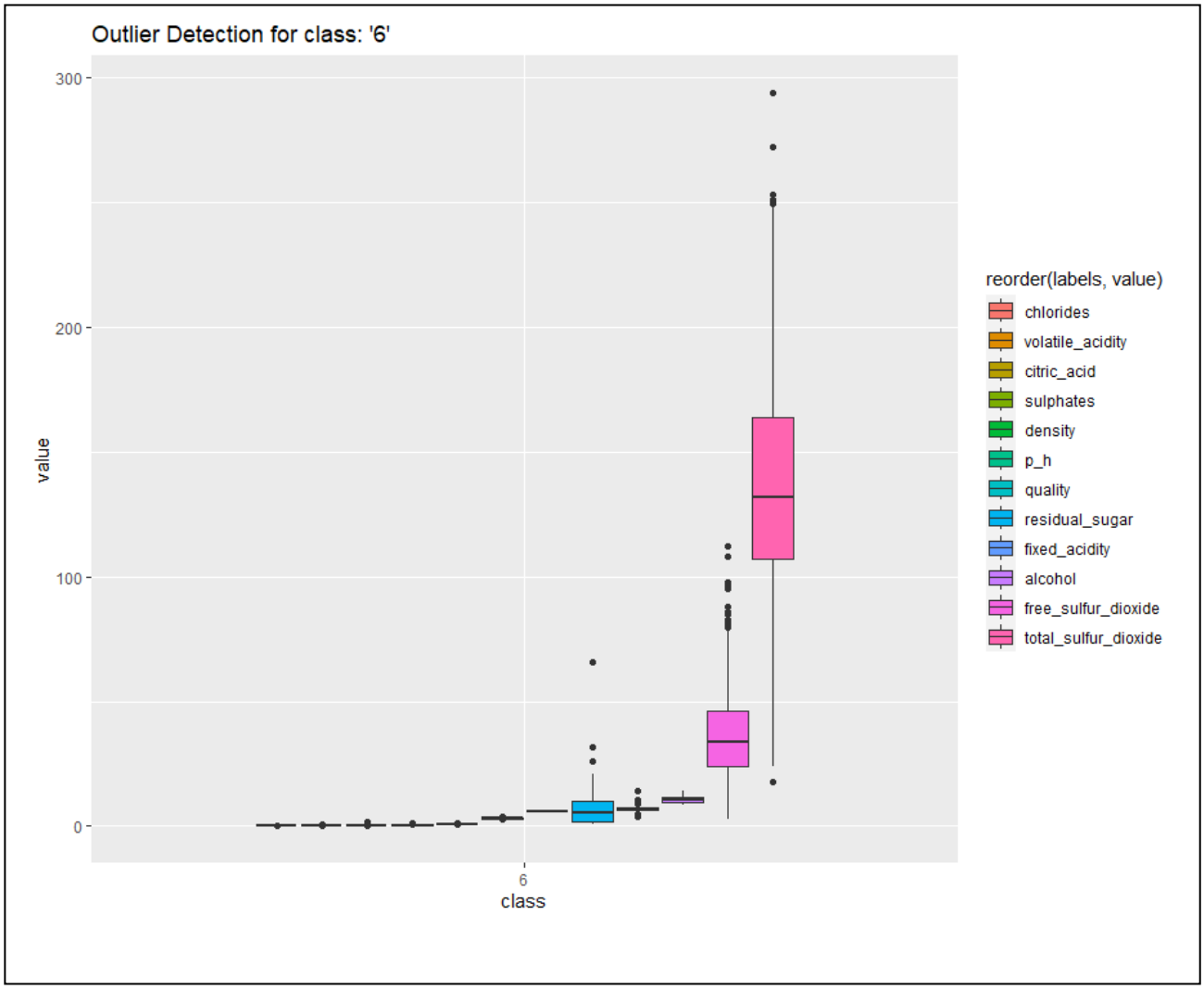
The following R code was used to display the dataset filtered by class "7" in order to highlight potential outliers discovered in the dataset.

```
> whitewine_new %>%
+   pivot_longer(1:12,names_to = "labels") %>%
+   filter(class == 7) %>%
+   mutate(class = fct_reorder(class,value,median)) %>%
+   ggplot(aes(class, value, fill = reorder(labels,value))) +
+   geom_boxplot() +
+   labs(title = "Outlier Detection for class: '7'")
```



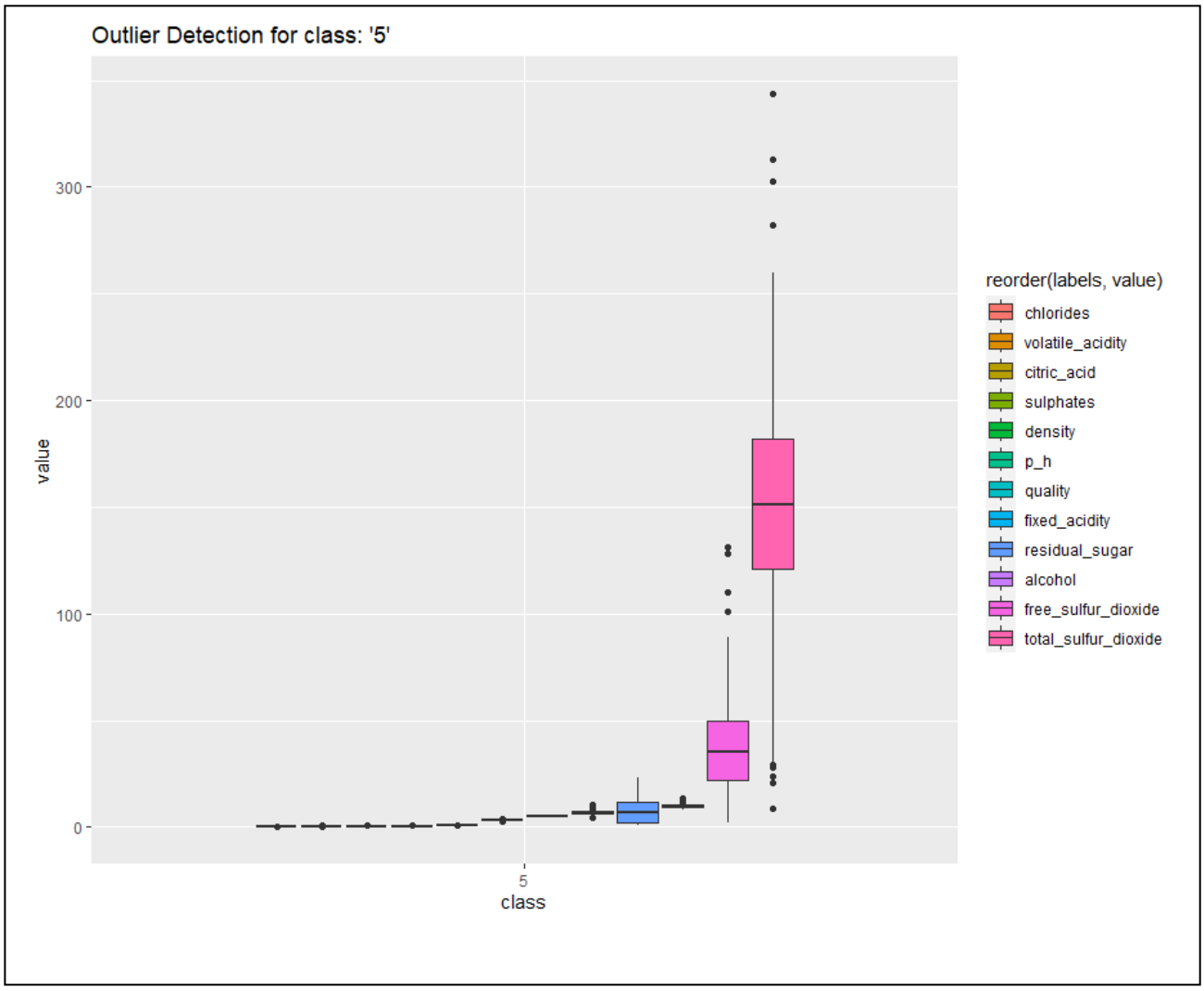
The following R code was used to display the dataset filtered by class "6" in order to highlight potential outliers discovered in the dataset.

```
> whitewine_new %>%
+   pivot_longer(1:12,names_to = "labels") %>%
+   filter(class == 6) %>%
+   mutate(class = fct_reorder(class,value,median)) %>%
+   ggplot(aes(class, value, fill = reorder(labels,value))) +
+   geom_boxplot() +
+   labs(title = "Outlier Detection for class: '6'")
```



The following R code was used to display the dataset filtered by class "5" in order to highlight potential outliers discovered in the dataset.

```
> whitewine_new %>%
+   pivot_longer(1:12,names_to = "labels") %>%
+   filter(class == 5) %>%
+   mutate(class = fct_reorder(class,value,median)) %>%
+   ggplot(aes(class, value, fill = reorder(labels,value))) +
+   geom_boxplot() +
+   labs(title = "Outlier Detection for class: '5'")
```

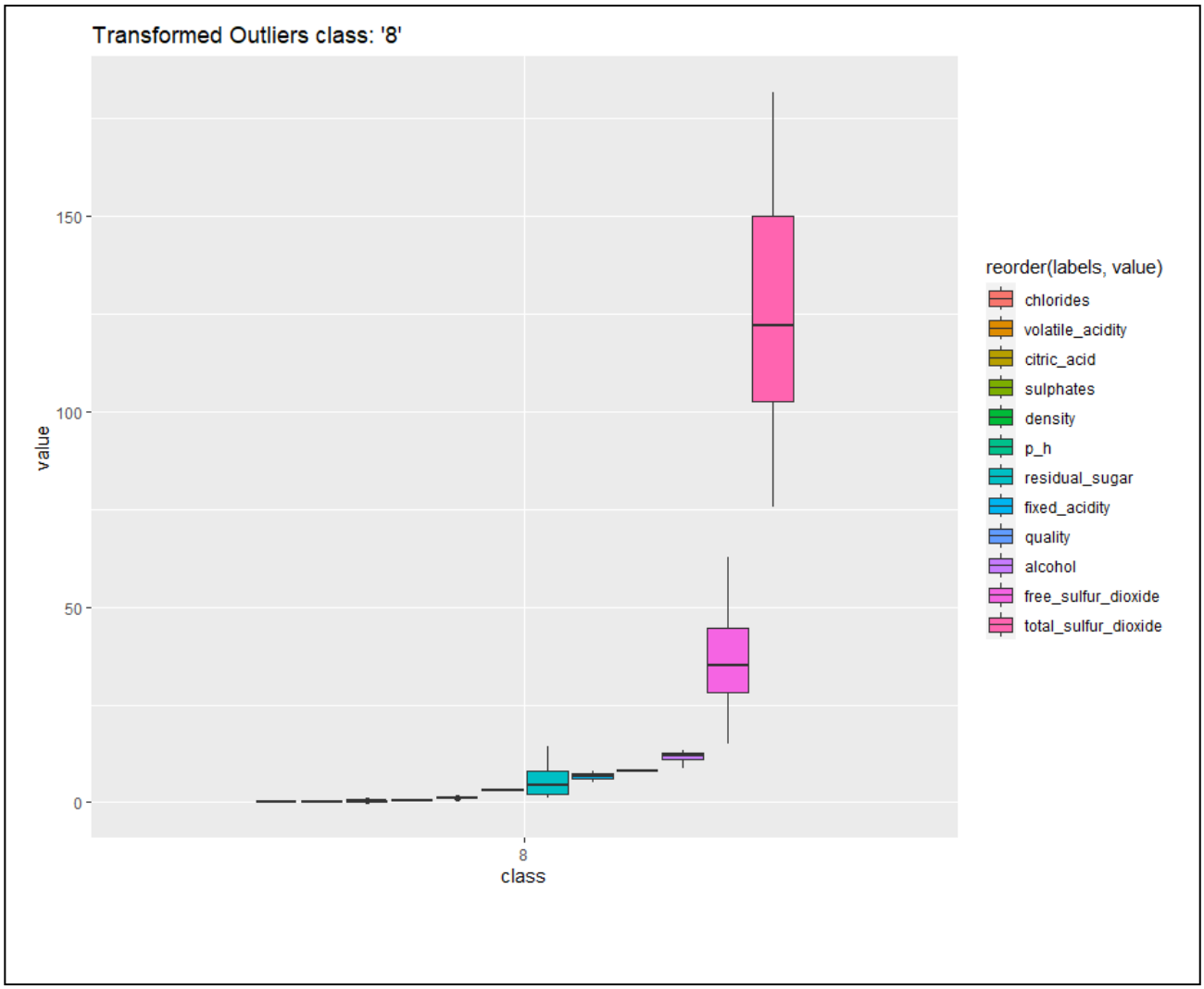


The data was then enhanced by using filters to minimize outliers before being aggregated by combining different datasets into one.

```
> quality_8 = whitewine_new %>%
+   filter(class == 8) %>%
+   mutate(across(1:12, ~squish(.x, quantile(.x, c(.05, .95)))))
> quality_7 = whitewine_new %>%
+   filter(class == 7) %>%
+   mutate(across(1:12, ~squish(.x, quantile(.x, c(.05, .95)))))
> quality_6 = whitewine_new %>%
+   filter(class == 6) %>%
+   mutate(across(1:12, ~squish(.x, quantile(.x, c(.05, .95)))))
> quality_5 = whitewine_new %>%
+   filter(class == 5) %>%
+   mutate(across(1:12, ~squish(.x, quantile(.x, c(.05, .95)))))
>
> combined = bind_rows(list(quality_8,quality_7,quality_6,quality_5))
> print(combined)
# A tibble: 4,710 × 13
   fixed_acidity volatile_acidity citric_acid residual_sugar chlorides free_sulfur_dioxide total_sulfur_dioxide density   p_h sulphates alcohol quality class
           <dbl>            <dbl>       <dbl>          <dbl>     <dbl>              <dbl>               <dbl>   <dbl> <dbl>     <dbl>   <dbl>   <dbl> <fct>
 1         6.2              0.47        0.48            1.2     0.029                 29                75.7   0.989  3.33      0.39    12.8       8 8
 2         6.2              0.47        0.48            1.2     0.029                 29                75.7   0.989  3.33      0.39    12.8       8 8
 3         6.8              0.26        0.42            1.7     0.049                 41               122     0.993  3.45      0.48    10.5       8 8
 4         6.7              0.23        0.31            2.1     0.046                 30                96     0.993  3.33      0.64    10.7       8 8
 5         6.7              0.23        0.31            2.1     0.046                 30                96     0.993  3.33      0.64    10.7       8 8
 6         5.2              0.44        0.222           1.4     0.036                 43               119     0.989  3.36      0.33    12.1       8 8
 7         5.2              0.44        0.222           1.4     0.036                 43               119     0.989  3.36      0.33    12.1       8 8
 8         6.8              0.47        0.35            3.8     0.034                 26               109     0.991  3.26      0.57    12.7       8 8
 9         6.7              0.26        0.39            1.17    0.04                  45               147     0.994  3.32      0.58     9.6       8 8
10         7                0.24        0.36            2.8     0.034                 22               112     0.99   3.19      0.38    12.6       8 8
```

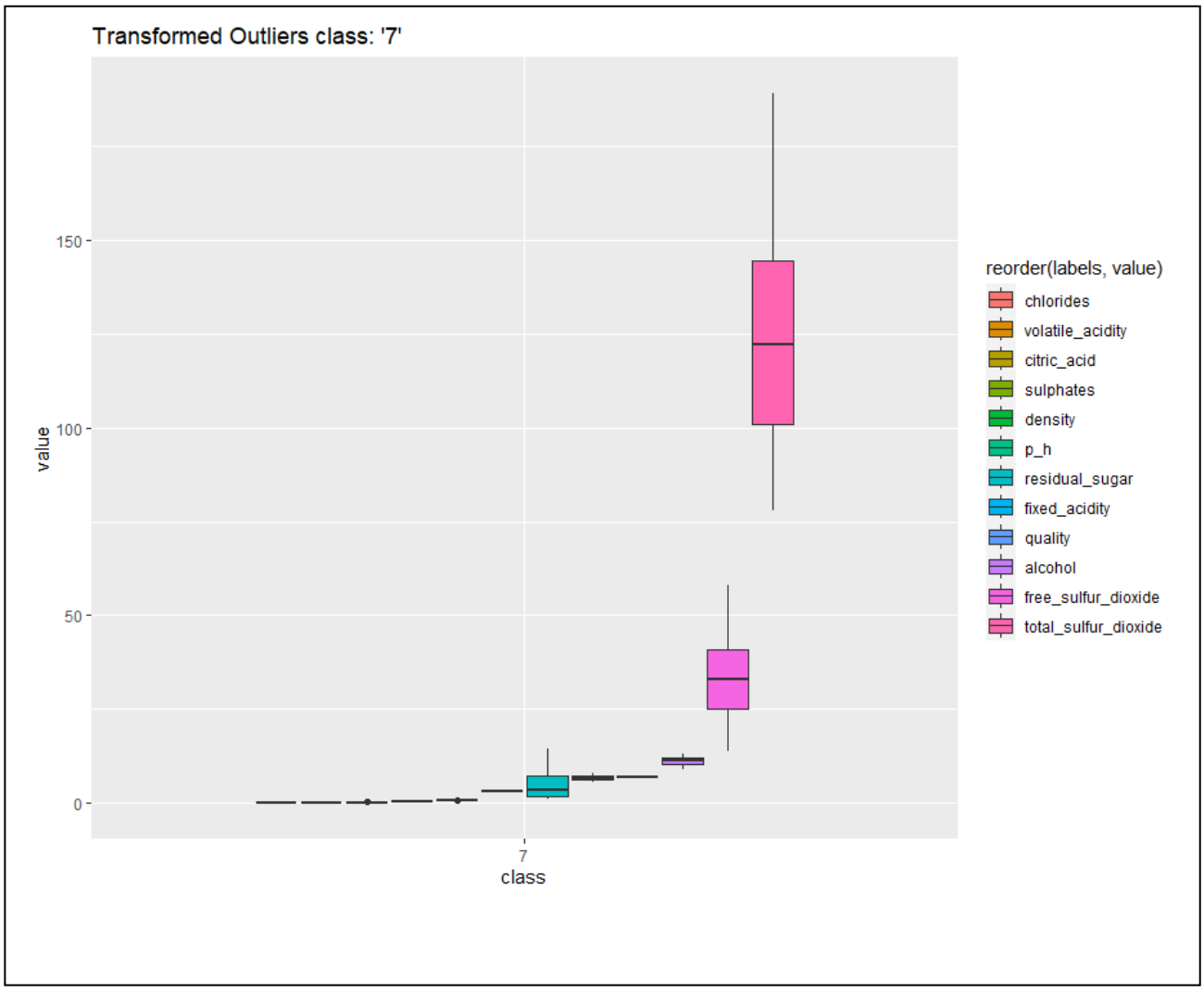
The R code used to display the dataset filtered by class "8" after eliminating any outliers is shown below.

```
> combined %>%
+   pivot_longer(1:12,names_to = "labels") %>%
+   filter(class == 8) %>%
+   mutate(class = fct_reorder(class,value,median)) %>%
+   ggplot(aes(class, value, fill = reorder(labels,value))) +
+   geom_boxplot() +
+   labs(title = "Transformed Outliers class: '8'")
```



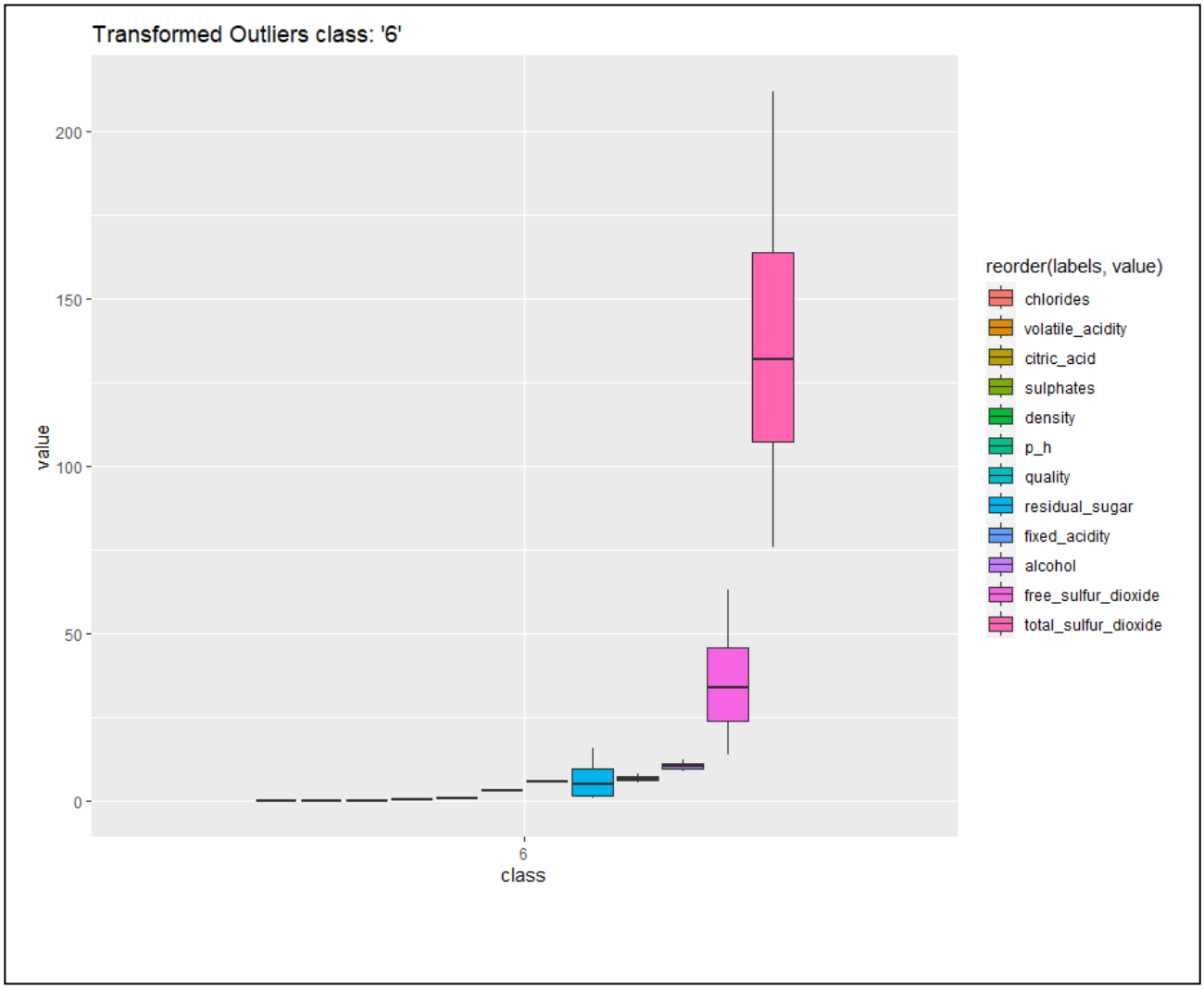
The R code used to display the dataset filtered by class "7" after eliminating any outliers is shown below.

```
> combined %>%
+   pivot_longer(1:12,names_to = "labels") %>%
+   filter(class == 7) %>%
+   mutate(class = fct_reorder(class,value,median)) %>%
+   ggplot(aes(class, value, fill = reorder(labels,value))) +
+   geom_boxplot() +
+   labs(title = "Transformed Outliers class: '7'")
```



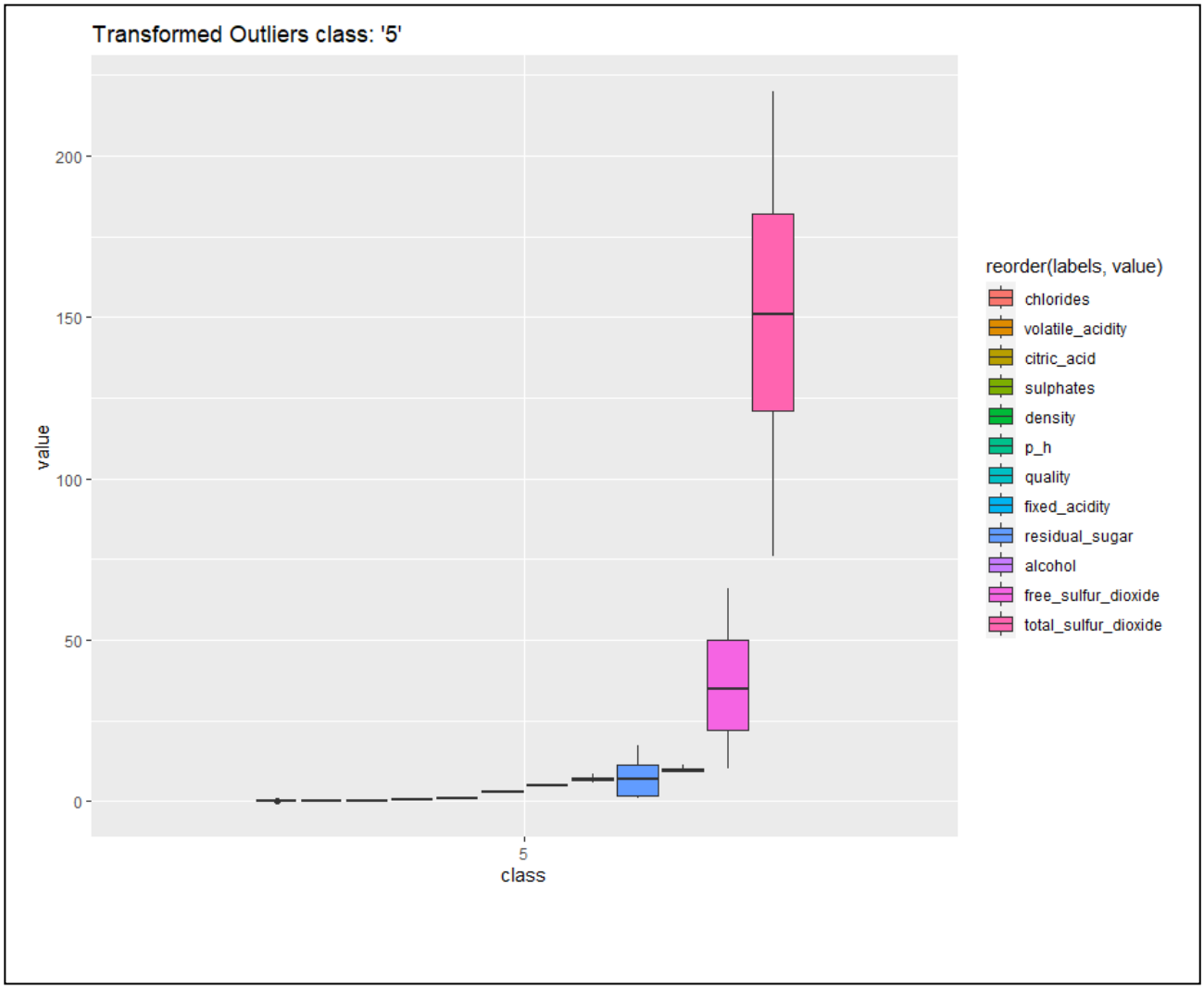
The R code used to display the dataset filtered by class "6" after eliminating any outliers is shown below.

```
> combined %>%
+   pivot_longer(1:12,names_to = "labels") %>%
+   filter(class == 6) %>%
+   mutate(class = fct_reorder(class,value,median)) %>%
+   ggplot(aes(class, value, fill = reorder(labels,value))) +
+   geom_boxplot() +
+   labs(title = "Transformed Outliers class: '6'")
```



The R code used to display the dataset filtered by class "5" after eliminating any outliers is shown below.

```
> combined %>%
+   pivot_longer(1:12,names_to = "labels") %>%
+   filter(class == 5) %>%
+   mutate(class = fct_reorder(class,value,median)) %>%
+   ggplot(aes(class, value, fill = reorder(labels,value))) +
+   geom_boxplot() +
+   labs(title = "Transformed Outliers class: '5'")
```



1.2 Define the number of cluster centres

We now retain numerical data for the algorithm while removing quality and class.

Following code snippet was used to scale the data after removing ourliners and analyze the dataset after all the pre-processing tasks

```
> # Remove the quality and the class name. Both of these will be remove so that only
> # numerical data is left for the algorithm.
> whitewine_data_points = combined %>%
+   select(-quality, -class)

> # Now that we have the "whitewine_data_points" dataset, scaling is performed
> whitewine_scaled = whitewine_data_points %>%
+   mutate(across(everything(), scale))

# Analyze the dataset after outlier removal and scaling
> summary(whitewine_scaled)
```

```
  fixed_acidity.V1   volatile_acidity.V1     citric_acid.V1     residual_sugar.V1
 Min.   :-2.2474076   Min.   :-1.6115915   Min.   :-1.9953114   Min.   :-1.0934632
 1st Qu.:-0.7328926   1st Qu.:-0.7529338   1st Qu.:-0.6480349   1st Qu.:-0.9689704
 Median :-0.0444766   Median :-0.1396068   Median :-0.1298517   Median :-0.2220135
 Mean   : 0.0000000   Mean   : 0.0000000   Mean   : 0.0000000   Mean   : 0.0000000
 3rd Qu.: 0.6439393   3rd Qu.: 0.5963856   3rd Qu.: 0.5956049   3rd Qu.: 0.7531802
 Max.   : 2.1584543   Max.   : 2.6816974   Max.   : 2.7927019   Max.   : 2.2470939
    chlorides.V1     free_sulfur_dioxide.V1 total_sulfur_dioxide.V1      density.V1
 Min.   :-1.728632   Min.   :-1.7395121     Min.   :-1.6357979      Min.   :-1.7949335
 1st Qu.:-0.675934   1st Qu.:-0.7775489     1st Qu.:-0.7685812      1st Qu.:-0.8231434
 Median :-0.061860   Median :-0.0904324     Median :-0.1175177      Median :-0.1064839
 Mean   : 0.000000   Mean   : 0.0000000     Mean   : 0.0000000      Mean   : 0.0000000
 3rd Qu.: 0.552214   3rd Qu.: 0.7341074     3rd Qu.: 0.7418862      3rd Qu.: 0.7535073
 Max.   : 3.710307   Max.   : 2.1083405     Max.   : 2.1221410      Max.   : 2.0076613
       p_h.V1           sulphates.V1          alcohol.V1
 Min.   :-1.7224355   Min.   :-1.6604153   Min.   :-1.4404495
 1st Qu.:-0.7148739   1st Qu.:-0.7689186   1st Qu.:-0.8516262
 Median :-0.0481052   Median :-0.0755322   Median :-0.0945677
 Mean   : 0.0000000   Mean   : 0.0000000   Mean   : 0.0000000
 3rd Qu.: 0.6927489   3rd Qu.: 0.6178541   3rd Qu.: 0.7466084
 Max.   : 2.1744571   Max.   : 2.8267849   Max.   : 2.2607254
```

The whitewine_scaled dataset is a pre-processed dispersed dataset that we may utilize for the procedure. It is scaled to obtain the required range. We set the seed value to retain the same train and test datasets.

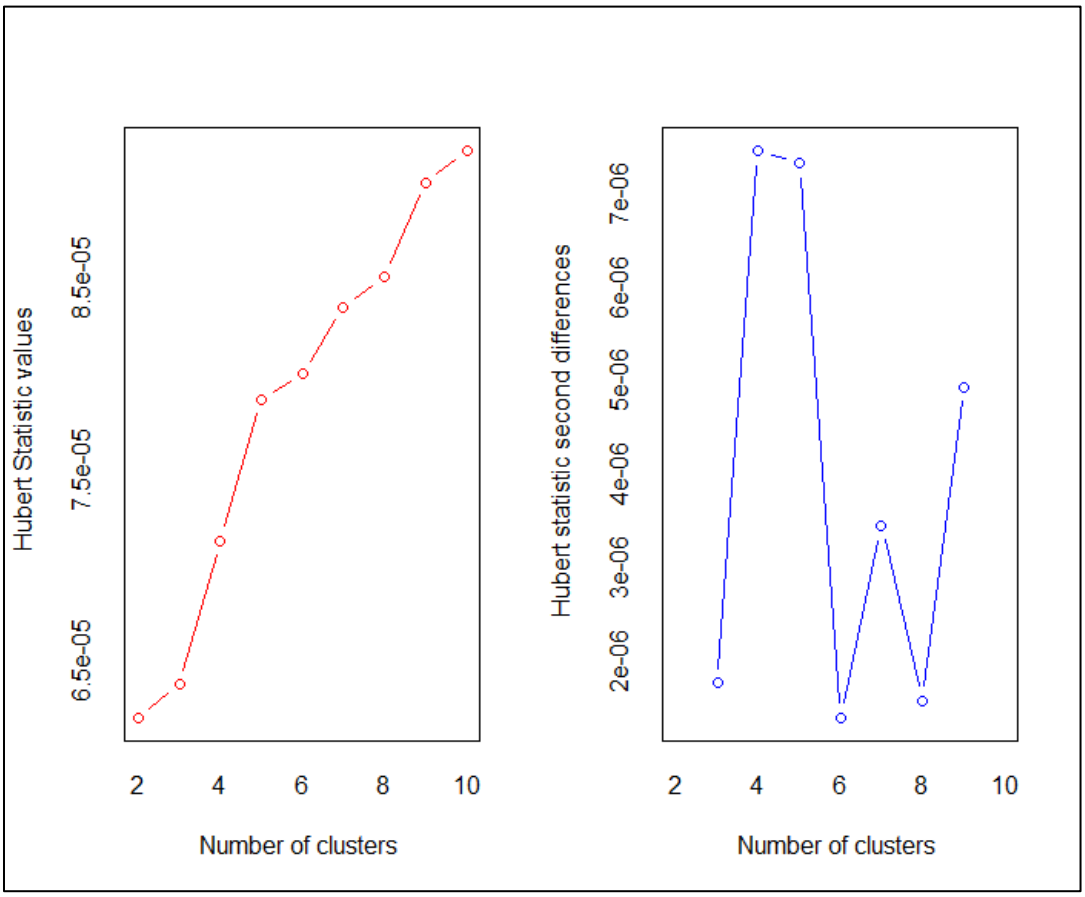
```
> set.seed(1234)
```

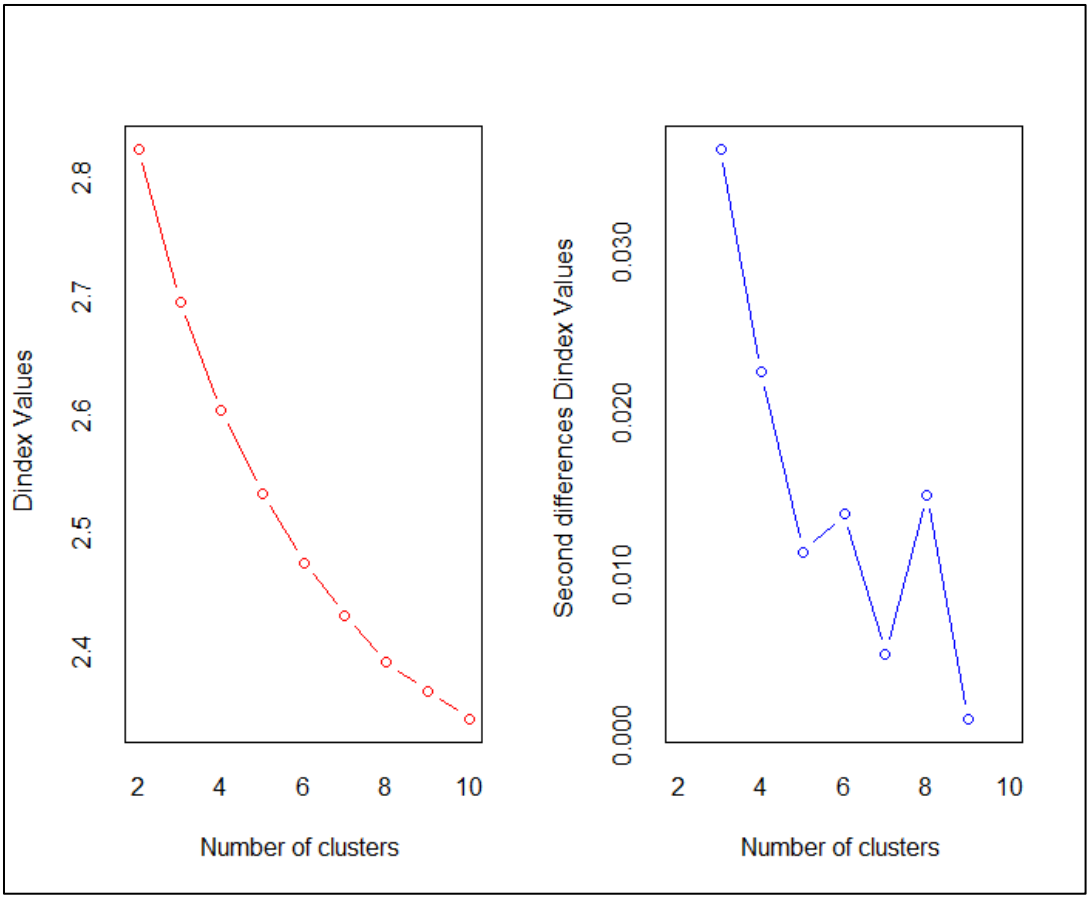
Perform the k-means using the NbClust function with euclidean for distance. Min=2 and Max=10

```
> # Perform the kmeans using the NbClust function
> # Use Euclidean for distance
> cluster_euclidean = NbClust(whitewine_scaled,distance="euclidean",
+                             min.nc=2,max.nc=10,method="kmeans",index="all")
```

```
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hube
rt
                index second differences plot.

*** : The D index is a graphical method of determining the number of clusters.
                In the plot of D index, we seek a significant knee (the significant peak in Dinde
x
                second differences plot) that corresponds to a significant increase of the value
of
                the measure.

*******************************************************************
* Among all indices:
* 12 proposed 2 as the best number of clusters
* 7 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 1 proposed 8 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


*******************************************************************
```

Analysis suggests that the ideal number of clusters be two. 3,4,6,7,8, and 10 have also been proposed.


Hubert Statistic values
6.5e-05
7.5e-05
8.5e-05
2
4
6
8
10
Number of clusters
Hubert statistic second differences
2e-06
3e-06
4e-06
5e-06
6e-06
7e-06
2
4
6
8
10
Number of clusters

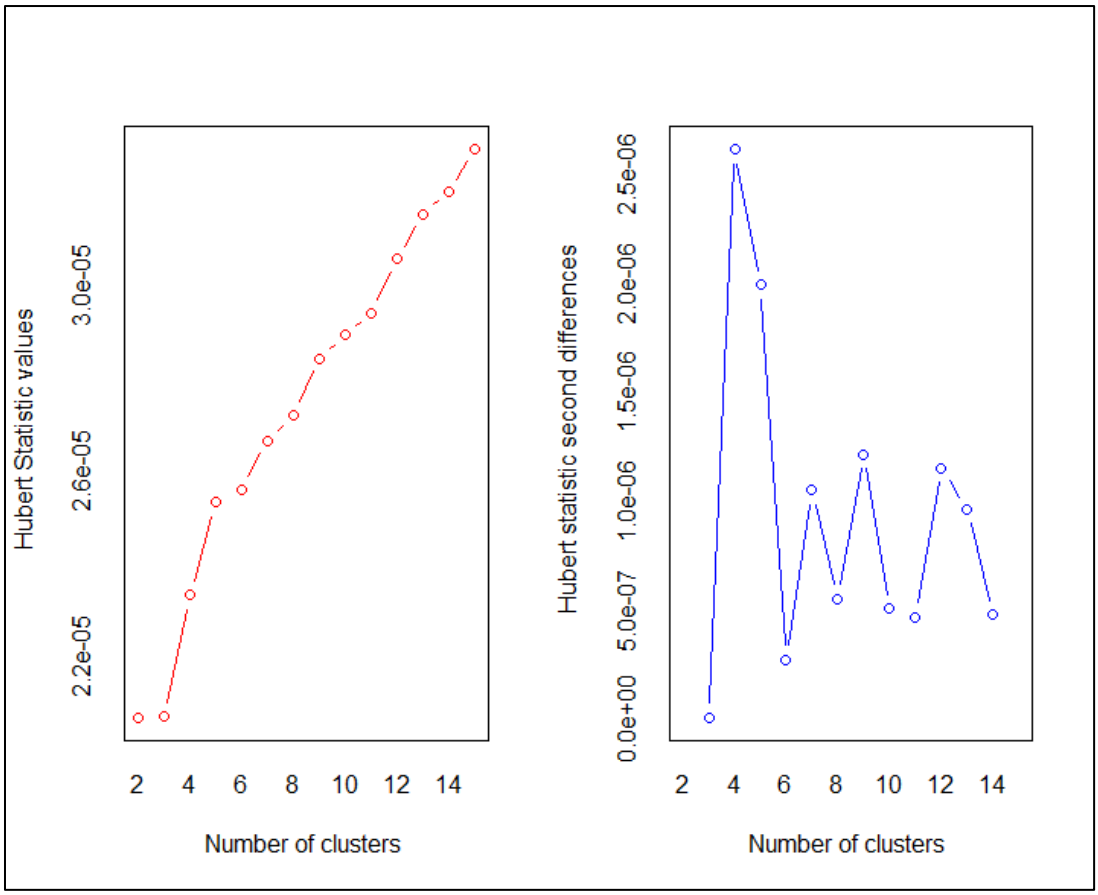

Dindex Values
2.4
2.5
2.6
2.7
2.8
2
4
6
8
10
Number of clusters
Second differences Dindex Values
0.000
0.010
0.020
0.030
2
4
6
8
10
Number of clusters

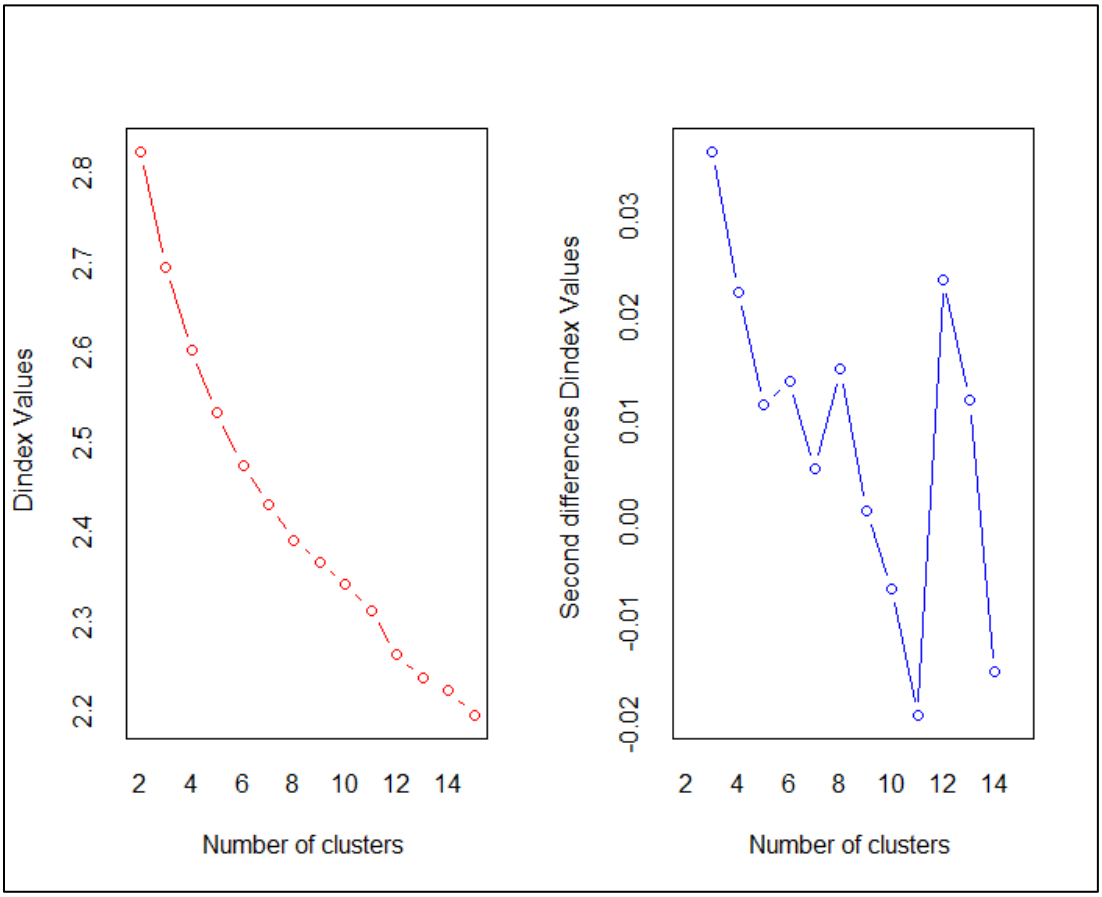
Perform the k-means using the NbClust function with manhattan for distance. Min=2 and Max=15 clusters

```
> # Use manhattan for distance
> cluster_manhattan = NbClust(whitewine_scaled,distance="manhattan",
+                             min.nc=2,max.nc=15,method="kmeans",index="all")
```

```
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hube
rt
                index second differences plot.

*** : The D index is a graphical method of determining the number of clusters.
                In the plot of D index, we seek a significant knee (the significant peak in Dinde
x
                second differences plot) that corresponds to a significant increase of the value
of
                the measure.

*******************************************************************
* Among all indices:
* 9 proposed 2 as the best number of clusters
* 7 proposed 3 as the best number of clusters
* 2 proposed 4 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 2 proposed 12 as the best number of clusters
* 1 proposed 13 as the best number of clusters
* 1 proposed 14 as the best number of clusters
* 1 proposed 15 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  2


*******************************************************************
```

Analysis suggests that the ideal number of clusters be two. 3,4,8,12,13,14 and 15 have also been proposed.


Hubert Statistic values
2.2e-05
2.6e-05
3.0e-05
2 4 6 8 10 12 14
Number of clusters
Hubert statistic second differences
0.0e+00
5.0e-07
1.0e-06
1.5e-06
2.0e-06
2.5e-06
2 4 6 8 10 12 14
Number of clusters


Dindex Values
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2 4 6 8 10 12 14
Number of clusters
Second differences Dindex Values
-0.02
-0.01
0.00
0.01
0.02
0.03
2 4 6 8 10 12 14
Number of clusters

1.3 K-means analysis

- **The Elbow Method**
- **The Silhouette Method**

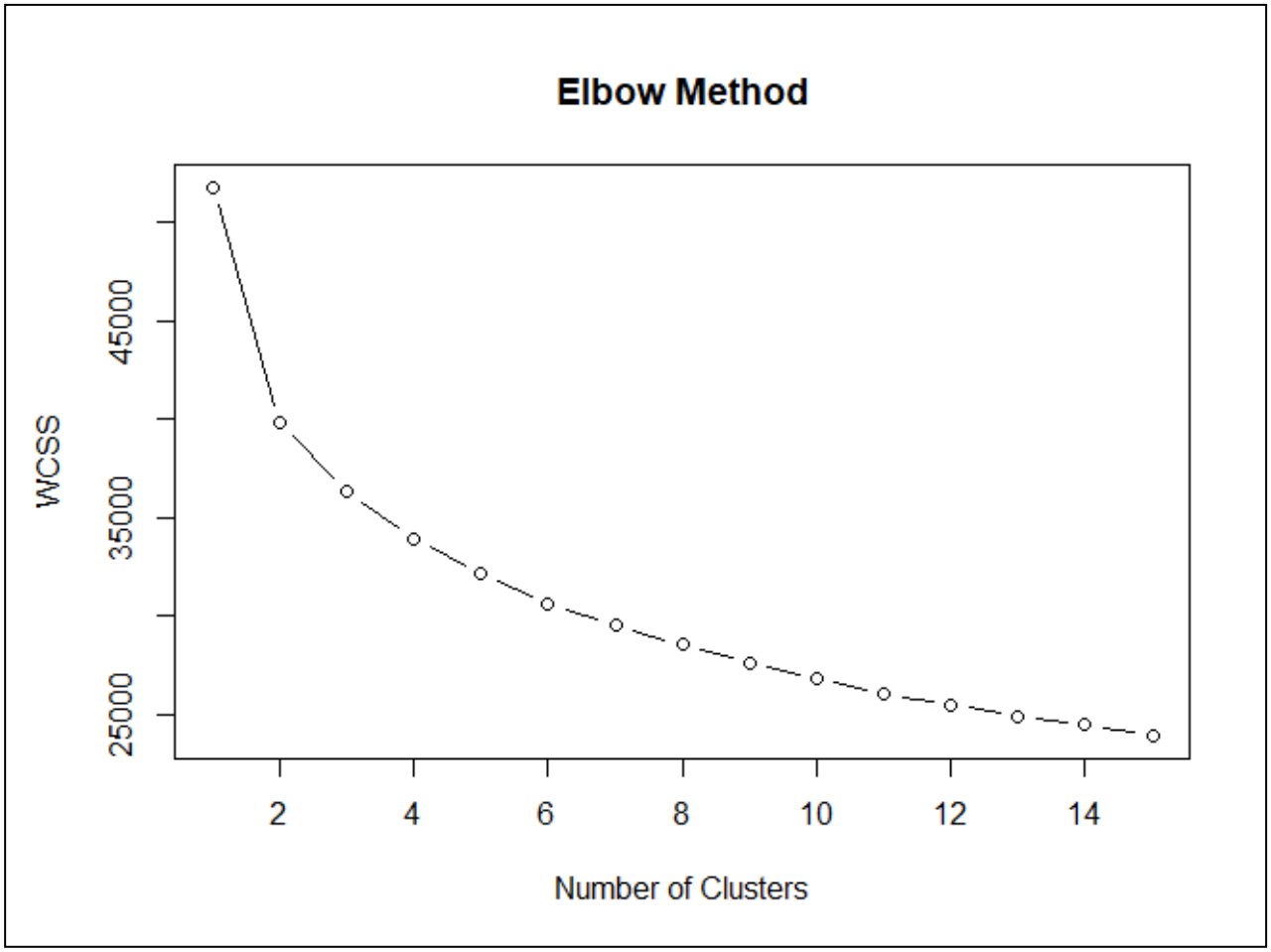
The Elbow Method

The ideal number of clusters may be determined using the Elbow method. We want the clustering's overall within-cluster sum of squares (wss), which measures how compact the clustering is, to be as little as feasible.

The steps to determine the ideal number of clusters using the elbow approach are as follows:

- Compute the k-means for various k values. Measure the total within-cluster sum of squares (wss) for each cluster k (by varying the number of clusters from 1 to 12) and plot the curve dependent on the cluster count k.
- The ideal number of clusters is often determined by where a bend (knee) appears in the graph.

Following in the plot created using wss, we can observe that the plot bends when the number of clusters is 2, hence we can infer that the elbow method's recommended number of clusters is 2.



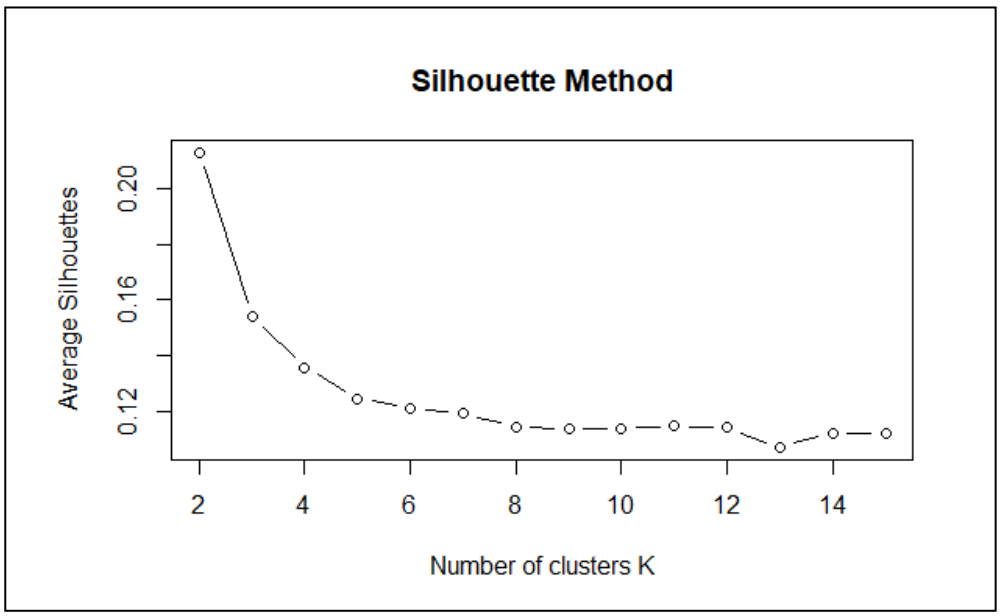
Following is the code snippet used to calculate the optimal number of clusters using the elbow method.

```
> # Finding the Optimal number of clusters using was
> # function that computes total within-cluster sum of square
> fn_kemans_clust <- function(data, cluster_count) {
+   kmeans(data, cluster_count, iter.max = 300, nstart = 7)
+ }
>
> # Use elbow method to find optimal number of clusters
> wcss <- vector()
> arr_clusters <- 1: 15
>
> for (i in arr_clusters) wcss[i] <- sum(fn_kemans_clust(whitewine_scaled, i)
$withinss)
>
> plot(arr_clusters, wcss, type ="b",
+      main="Elbow Method",
+      xlab="Number of Clusters",
+      ylab="WCSS")
```

The Silhouette Method

The silhouette value compares how similar a point is to its own cluster to other clusters (separation). The Silhouette value has a range of +1 to -1. A high value is preferable since it shows that the point is in the proper cluster. If several points have a negative Silhouette value, we may have built too many or too few clusters.

The output shows as below. According to that, 2 and 3 clusters are the only capable cluster count for k-means.



Following is the code snippet used to calculate the optimal number of clusters using the silhouette method.

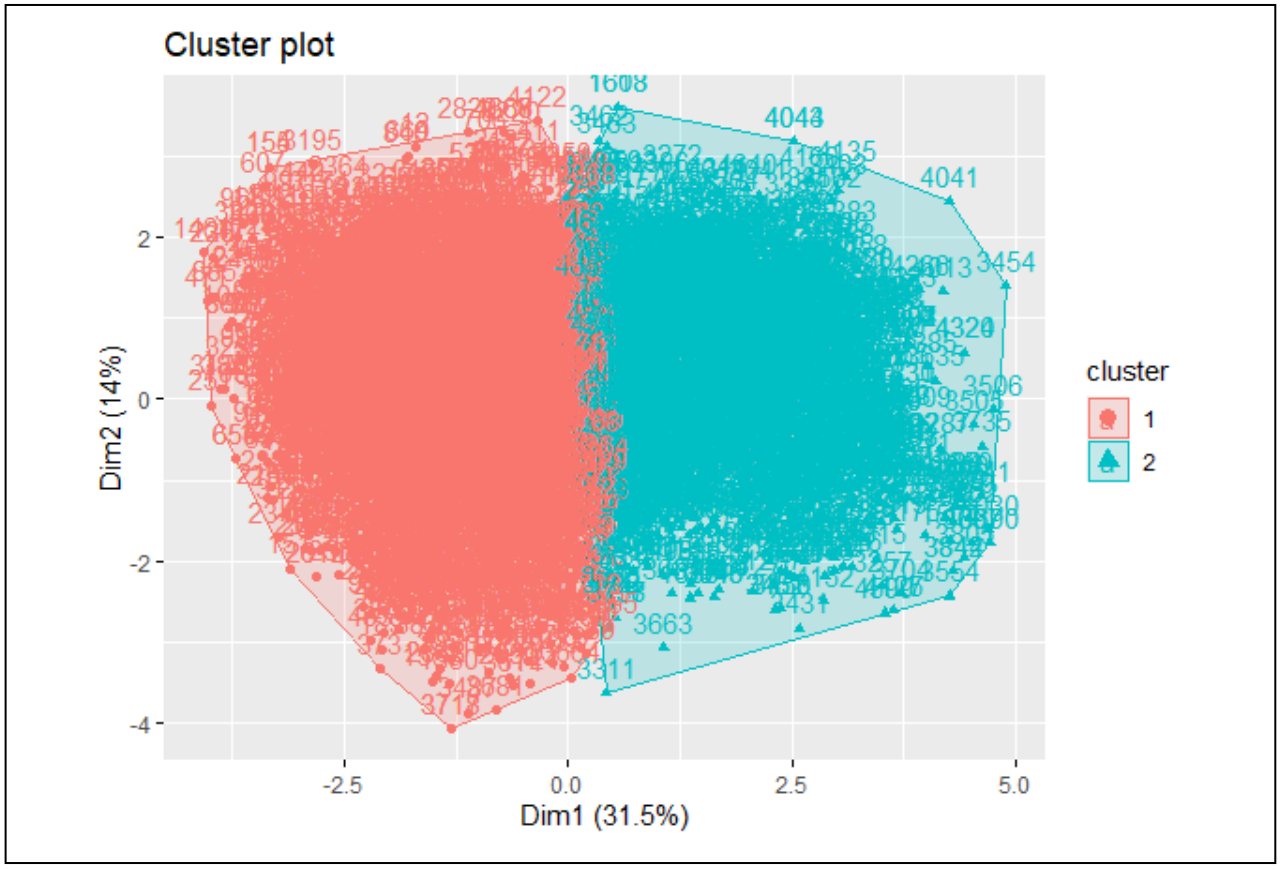
```
> # Use elbow Silhouette to find optimal number of clusters
> avg_sils <- vector()
> arr_clusters <- 2: 15
>
> fn_avg_sil <- function(data_matrix, cluster_count) {
+   k.temp <- fn_kemans_clust(data_matrix, cluster_count)
+   sil_values <- silhouette(k.temp$cluster, dist(whitewine_scaled))
+   mean(sil_values[,3])
+ }
> for (i in arr_clusters) avg_sils[i - 1] <- fn_avg_sil(whitewine_scaled, i)
>
> plot(arr_clusters, avg_sils, type ="b",
+      main="Silhouette Method",
+      xlab = "Number of clusters K",
+      ylab="Average Silhouettes")
```

In the previous step, we identified 2 best no of clusters. 3,4,6,7,8 and 10 also proposed. K-mean analysis for each cluster as follows.

```
> # kmeans results
> result_c2<-kmeans(whitewine_scaled, 2)
> result_c3<-kmeans(whitewine_scaled, 3)
> result_c4<-kmeans(whitewine_scaled, 4)
> result_c6<-kmeans(whitewine_scaled, 6)
> result_c7<-kmeans(whitewine_scaled, 7)
> result_c8<-kmeans(whitewine_scaled, 8)
> result_c10<-kmeans(whitewine_scaled, 10)
```

Following is the code snippet used to generate a confusion matrix for k-means of 2 clusters and the corresponding table generated.

```
> table(whitewine_new$class, result_c2$cluster)
```



Co-relation matrix

```
> #Co-relation matrix
> whitewine_initial.features <- whitewine_initial
> whitewine_initial.features$quality <- NULL
> cor(whitewine_initial.features)

                      fixed acidity volatile acidity citric acid residual sugar    ch
lorides free sulfur dioxide
fixed acidity           1.00000000      -0.03712969  0.28013101     0.09353639  0.02
3958005         -0.04082000
volatile acidity       -0.03712969       1.00000000 -0.13261918     0.08466140  0.05
3732427         -0.07838715
citric acid             0.28013101      -0.13261918  1.00000000     0.08665264  0.12
6229492          0.09523070
residual sugar          0.09353639       0.08466140  0.08665264     1.00000000  0.08
5086913          0.30800013
chlorides               0.02395801       0.05373243  0.12622949     0.08508691  1.00
0000000          0.11593517
free sulfur dioxide    -0.04082000      -0.07838715  0.09523070     0.30800013  0.11
5935168          1.00000000
total sulfur dioxide    0.09819163       0.11047571  0.11623658     0.40379988  0.19
4929710          0.60700661
density                 0.26365328       0.02294248  0.14728385     0.84472041  0.25
2989086          0.31015384
pH                     -0.42259047      -0.03931476 -0.15592455    -0.19481518 -0.09
2994759         -0.01107036
sulphates              -0.01565558      -0.03281480  0.05538476    -0.03391986  0.00
5871057          0.05064902
alcohol                -0.12174031       0.08370870 -0.08301621    -0.45941216 -0.35
9102035         -0.26696222
                     total sulfur dioxide     density           pH    sulphates
alcohol
fixed acidity                0.098191625  0.26365328 -0.422590473 -0.015655584 -0.1
2174031
volatile acidity             0.110475711  0.02294248 -0.039314756 -0.032814799  0.0
8370870
citric acid                  0.116236582  0.14728385 -0.155924547  0.055384760 -0.0
8301621
residual sugar               0.403799880  0.84472041 -0.194815183 -0.033919861 -0.4
5941216
chlorides                    0.194929710  0.25298909 -0.092994759  0.005871057 -0.3
5910203
free sulfur dioxide          0.607006611  0.31015384 -0.011070365  0.050649017 -0.2
6696222
total sulfur dioxide         1.000000000  0.53471756 -0.007178929  0.119745623 -0.4
5806275
density                      0.534717559  1.00000000 -0.096319776  0.067934004 -0.7
8266368
pH                          -0.007178929 -0.09631978  1.000000000  0.156034102  0.1
2575378
sulphates                    0.119745623  0.06793400  0.156034102  1.000000000 -0.0
1760224
alcohol                     -0.458062753 -0.78266368  0.125753775 -0.017602238  1.0
```

According to the result there are very low co-relations between variables.

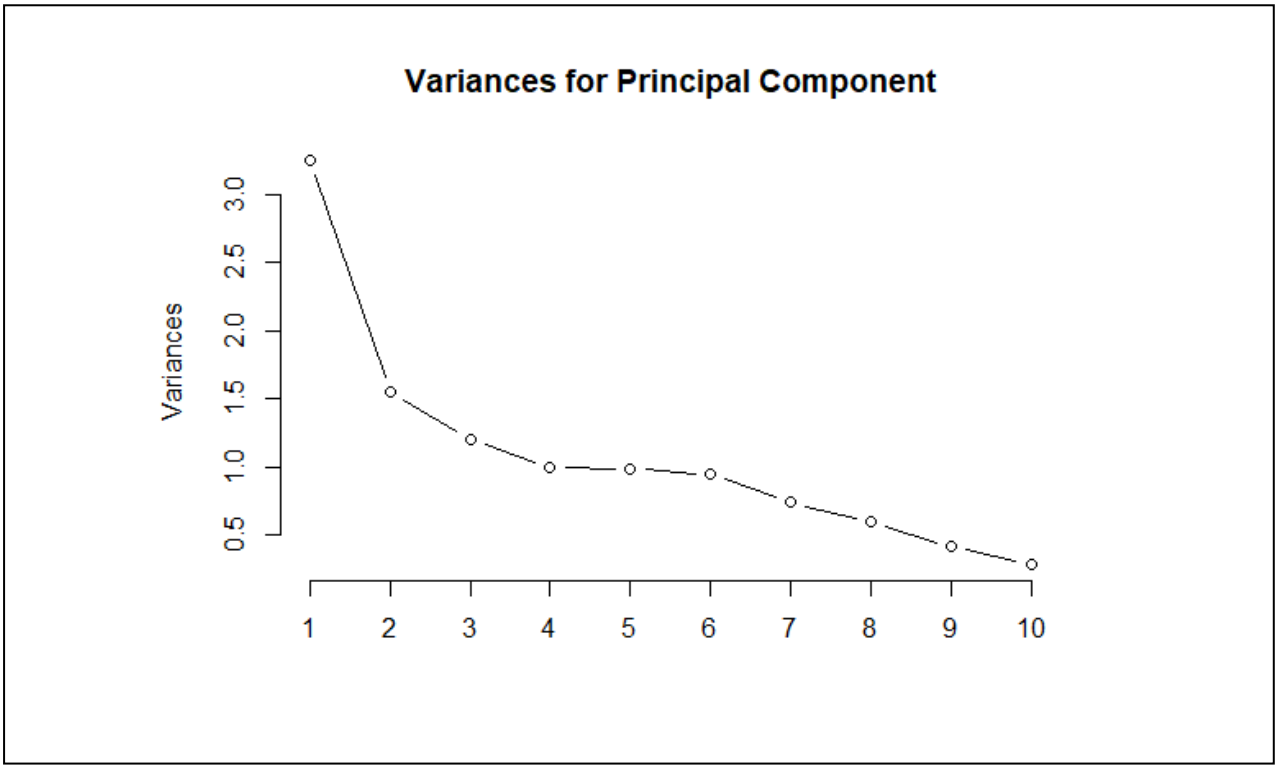
Principal Component Analysis (PCA)

Method for highlighting variance and highlighting prominent patterns in a dataset. It is frequently applied to make data exploration and visualization simple. In essence, PCA reduces the complexity of high dimensional data by condensing the data into a small number of dimensions that may be utilized to condense the features.

```
> #Principal Component Analysis (PCA)
> prc <- prcomp(whitewine_initial.features, scale = TRUE)
> summary(prc)
Importance of components:
                          PC1    PC2    PC3    PC4     PC5    PC6     PC7
PC8     PC9    PC10    PC11
Standard deviation     1.8020 1.2467 1.0961 1.0011 0.99439 0.9726 0.86008 0.
7714 0.64404 0.53919 0.14123
Proportion of Variance 0.2952 0.1413 0.1092 0.0911 0.08989 0.0860 0.06725 0.
0541 0.03771 0.02643 0.00181
Cumulative Proportion  0.2952 0.4365 0.5457 0.6368 0.72670 0.8127 0.87995 0.
9341 0.97176 0.99819 1.00000
```

How proportion of variance changed with principle components.

```
> # Variances for each principal component
> plot(prc, type = "lines", main="Variances for Principal Component")
```



1.4 Evaluation of the produced outputs

Following is the code snippet used to generate a confusion matrix for k-means of clusters and the corresponding tables generated.

```
> # Confusion matrix for k-means with 2 cluster
> table(whitewine_new$class, result_c2$cluster)

      1    2
  5 1069  388
  6 1254  944
  7  359  521
  8   67  108
```

```
> # Confusion matrix for k-means with 3 cluster
> table(whitewine_new$class, result_c3$cluster)

      1   2   3
  5 678 472 307
  6 705 720 773
  7 140 289 451
  8  19  64  92
```

```
> # Confusion matrix for k-means with 4 cluster
> table(whitewine_new$class, result_c4$cluster)

      1   2   3   4
  5 262 323 555 317
  6 653 538 461 546
  7 391 239  49 201
  8  67  72  16  20
```

```
> # Confusion matrix for k-means with 6 cluster
> table(whitewine_new$class, result_c6$cluster)

      1   2   3   4   5   6
  5 281 487 172 169 108 240
  6 325 397 370 360 327 419
  7  90  46 170 200 218 156
  8   9   9  39  66  34  18
```

```
> # Confusion matrix for k-means with 7 cluster
> table(whitewine_new$class, result_c7$cluster)

      1   2   3   4   5   6   7
  5 213  69 203 175 267 420 110
  6 261 180 355 380 380 322 320
  7  67 178 141 178  90  26 200
  8  11  49  15  49  18   5  28
```

```
> # Confusion matrix for k-means with 8 cluster
> table(whitewine_new$class, result_c8$cluster)

      1   2   3   4   5   6   7   8
  5 196 170  27  78 230 265 101 390
  6 359 370 106 231 262 273 305 292
  7 132 179 122 113  53  81 175  25
  8  18  48  23  34   4  19  23   6
```

```
> # Confusion matrix for k-means with 10 cluster
> table(whitewine_new$class, result_c10$cluster)

     1   2   3   4   5   6   7   8   9  10
  5 208  10 205 119  94 197  80 339  57 148
  6 263  48 193 283 261 312 194 236 120 288
  7  65  85  34 129 156  79 111  11 103 107
  8   6  19   8  35  23   8  10   6  29  31
```

Confusion matrix

```
> #confusion matrix function
> confusion_matrix <- function(k_data) {
+   whitewineNcluster <- cbind(whitewine_new, cluster = k_data$cluster)
+   whitewineNcluster_df <- union(whitewineNcluster$cluster, whitewineNcluste
r$class)
+   #get confusion matrix table
+   whitewineNcluster_df_table <- table(factor(whitewineNcluster$cluster, whi
tewineNcluster_df),
+                                       factor(whitewineNcluster$quality, whi
tewineNcluster_df))
+   confusionMatrix(whitewineNcluster_df_table)
+ }
> confusion_matrix(result_data_points)
```

```
Confusion Matrix and Statistics


     1    2    5    6    7    8
  1  0    0  933 1247  384   75
  2  0    0  524  951  496  100
  5  0    0    0    0    0    0
  6  0    0    0    0    0    0
  7  0    0    0    0    0    0
  8  0    0    0    0    0    0

Overall Statistics

               Accuracy : 0
                 95% CI : (0, 8e-04)
    No Information Rate : 0.4667
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity                NA       NA   0.0000   0.0000   0.0000  0.00000
Specificity            0.4397   0.5603   1.0000   1.0000   1.0000  1.00000
Pos Pred Value             NA       NA      NaN      NaN      NaN      NaN
Neg Pred Value             NA       NA   0.6907   0.5333   0.8132  0.96285
Prevalence             0.0000   0.0000   0.3093   0.4667   0.1868  0.03715
Detection Rate         0.0000   0.0000   0.0000   0.0000   0.0000  0.00000
Detection Prevalence   0.5603   0.4397   0.0000   0.0000   0.0000  0.00000
Balanced Accuracy          NA       NA   0.5000   0.5000   0.5000  0.50000
```

Evaluation Indices

Confusion matrix findings show how well the classes in our test dataset were predicted. True Positive (TP) and True Negative (TN) are accurately predicted positive and negative classes, respectively. The actual class that is false and anticipated true is called a false positive (FP). False Negative (FN) is a true class result that was projected to be false.

	Prediction		
Actual		Class: Yes	Class: No
	Class : Yes	TP	FN
	Class: No	FP	TN

- Accuracy
  Accuracy gives how accurately predicted the result. It is a ratio of correct prediction and total results.

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$

- Precision
  Precision is the ratio of True Positive value according to the total Positive predictions

$$\text{Precision} = \frac{TP}{TP+FP}$$

- Recall
  Recall is the ratio of True Positive value according to the total True predictions

$$\text{Recall} = \frac{TP}{TP+TN}$$

1.5 Final "winner" and evaluation indices

The NbClust approach advises employing 2 clusters, considering both Manhattan distance and Euclidean distance, as the winning cluster.

Predicted result of 2 clusters

```
> result_data_points <- kmeans(whitewine_data_points,2)
> table(whitewine_new$class, result_data_points$cluster)

      1    2
  5  933  524
  6 1247  951
  7  384  496
  8   75  100
```

Confusion Matrix of prediction

```
> confusion_matrix(result_data_points)
Confusion Matrix and Statistics


      1    2    5    6    7    8
  1   0    0  933 1247  384   75
  2   0    0  524  951  496  100
  5   0    0    0    0    0    0
  6   0    0    0    0    0    0
  7   0    0    0    0    0    0
  8   0    0    0    0    0    0

Overall Statistics

               Accuracy : 0
                 95% CI : (0, 8e-04)
    No Information Rate : 0.4667
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 5 Class: 6 Class: 7 Class: 8
Sensitivity                NA       NA   0.0000   0.0000   0.0000  0.00000
Specificity            0.4397   0.5603   1.0000   1.0000   1.0000  1.00000
Pos Pred Value             NA       NA      NaN      NaN      NaN      NaN
Neg Pred Value             NA       NA   0.6907   0.5333   0.8132  0.96285
Prevalence             0.0000   0.0000   0.3093   0.4667   0.1868  0.03715
Detection Rate         0.0000   0.0000   0.0000   0.0000   0.0000  0.00000
Detection Prevalence   0.5603   0.4397   0.0000   0.0000   0.0000  0.00000
Balanced Accuracy          NA       NA   0.5000   0.5000   0.5000  0.50000
```

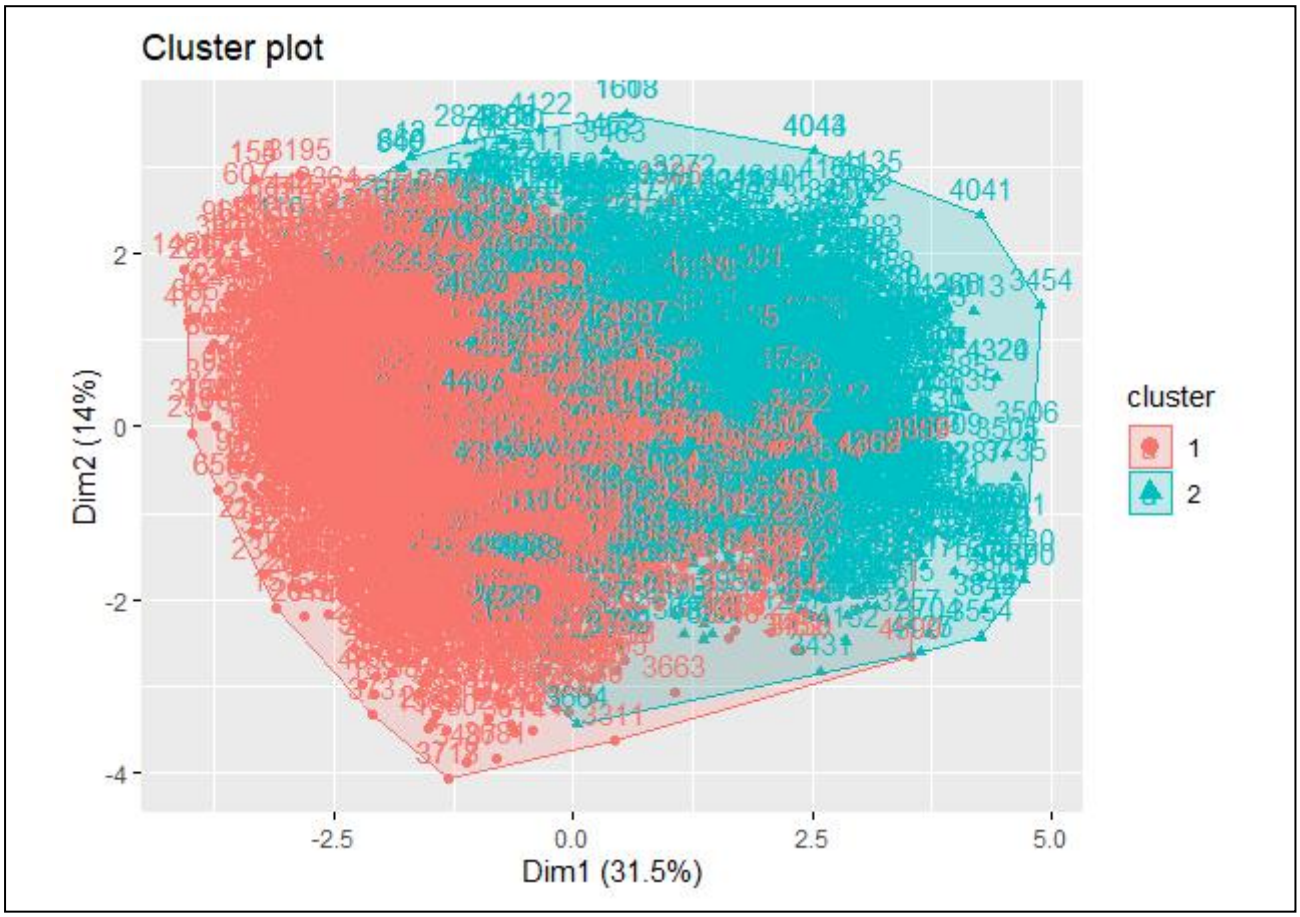
1.6 Illustrate the coordinates

```
> result_data_points <- kmeans(whitewine_data_points,2)
> table(whitewine_new$class, result_data_points$cluster)

      1    2
5   933  524
6  1247  951
7   384  496
8    75  100
```

```
> result_data_points
K-means clustering with 2 clusters of sizes 2639, 2071

Cluster means:
  fixed_acidity volatile_acidity citric_acid residual_sugar  chlorides free_sulfur_dioxide total_
sulfur_dioxide   density
1      6.762505        0.2656878   0.3227768       4.812111 0.04076199            27.86264
109.9738 0.9927863
2      6.921246        0.2786359   0.3449570       8.355176 0.04745553            44.81381
174.8784 0.9955402
       p_h sulphates   alcohol
1 3.188753 0.4782298 10.956934
2 3.183614 0.4995975  9.945999

Clustering vector:
   [1] 1 1 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 2 1 2 2 1 1 1 2 2 2 1 1 1 2 1 2 1 2 2 1 1 1 1 1 2 2
1 1 1 1 1 2 1 2 2 2 1 2 1 1
  [60] 1 1 2 2 2 2 2 1 1 1 1 2 2 2 2 1 2 2 2 2 2 2 1 1 1 1 1 1 1 2 2 1 2 2 1 1 1 1 1 1 1 1 2 1 1
2 1 1 1 1 2 1 1 1 1 1 1 1 1
 [119] 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
2 2 2 2 2 1 2 1 1 1 1 1 2 1
 [178] 1 1 2 1 1 1 1 1 1 1 2 1 2 2 2 1 1 2 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 2
1 2 1 1 1 1 1 1 1 1 1 1 1 1
 [237] 1 2 2 1 1 2 2 1 2 1 1 2 1 1 1 1 1 1 2 2 1 1 2 1 1 2 1 1 2 1 1 2 1 2 1 1 1 1 1 1 1 1 1 2
2 2 2 2 1 1 1 1 1 1 1 2 2 1
 [296] 1 1 1 2 1 1 1 1 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2 2 1 2 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 2 1
1 2 2 1 2 1 1 1 1 1 1 1 1 1
 [355] 1 1 1 2 2 1 2 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1
1 1 1 1 1 2 1 1 2 1 2 2 2
 [414] 2 2 2 2 2 2 1 1 1 1 1 2 2 2 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 2 2 2 2 1 1 1 1 2 1 2 2 1 1
1 1 1 1 2 2 2 2 2 2 1 1 1 1
 [473] 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 2 2 1 1 1 2 2 2 2 2 1 1 2 1 2
1 2 1 1 2 1 1 2 2 1 1 2 2 2
 [532] 1 1 1 1 2 2 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1 2 1 2 2 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1
1 1 1 1 2 2 1 1 1 2 1 1 1 1
 [591] 1 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 2 1 1 1 1 1 1
 [650] 1 2 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
1 1 1 1 1 1 1 2 1 2 2 2 1 2
 [709] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 2 2 2
1 1 1 1 1 1 1 1 1 1 1 2 1 2
 [768] 1 1 2 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1
1 1 1 1 1 1 2 1 1 2 1 1 1 1
 [827] 1 1 1 1 1 1 2 1 2 2 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 2 1 1 1 2 2 1
2 1 2 2 2 2 2 2 2 2 1 1 1 1
 [886] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 1 1 1 1 1 1 2 1 2 1 1 1 1 2 1
1 2 2 1 1 1 1 2 1 1 2 1 2 1
 [945] 2 2 2 2 2 2 1 2 1 2 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
 [ reached getOption("max.print") -- omitted 3710 entries ]

Within cluster sum of squares by cluster:
[1] 1369326 1452617
 (between_SS / total_SS =  65.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "si
ze"         "iter"
[9] "ifault"
```

Plot the result cluster

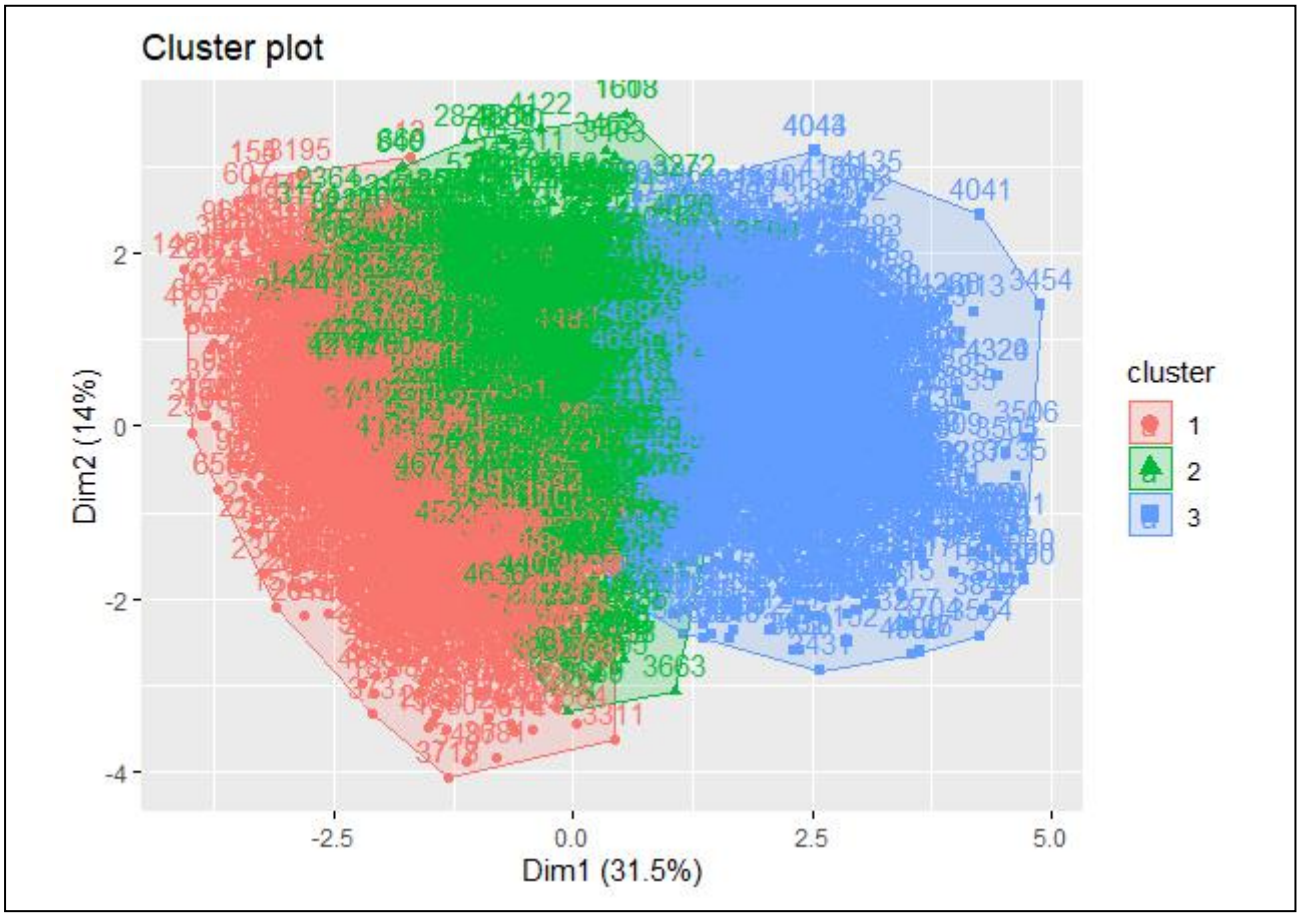
```
> #PLot cluster
> fviz_cluster(result_data_points, whitewine_scaled)
```



Plot next best 2 cluster

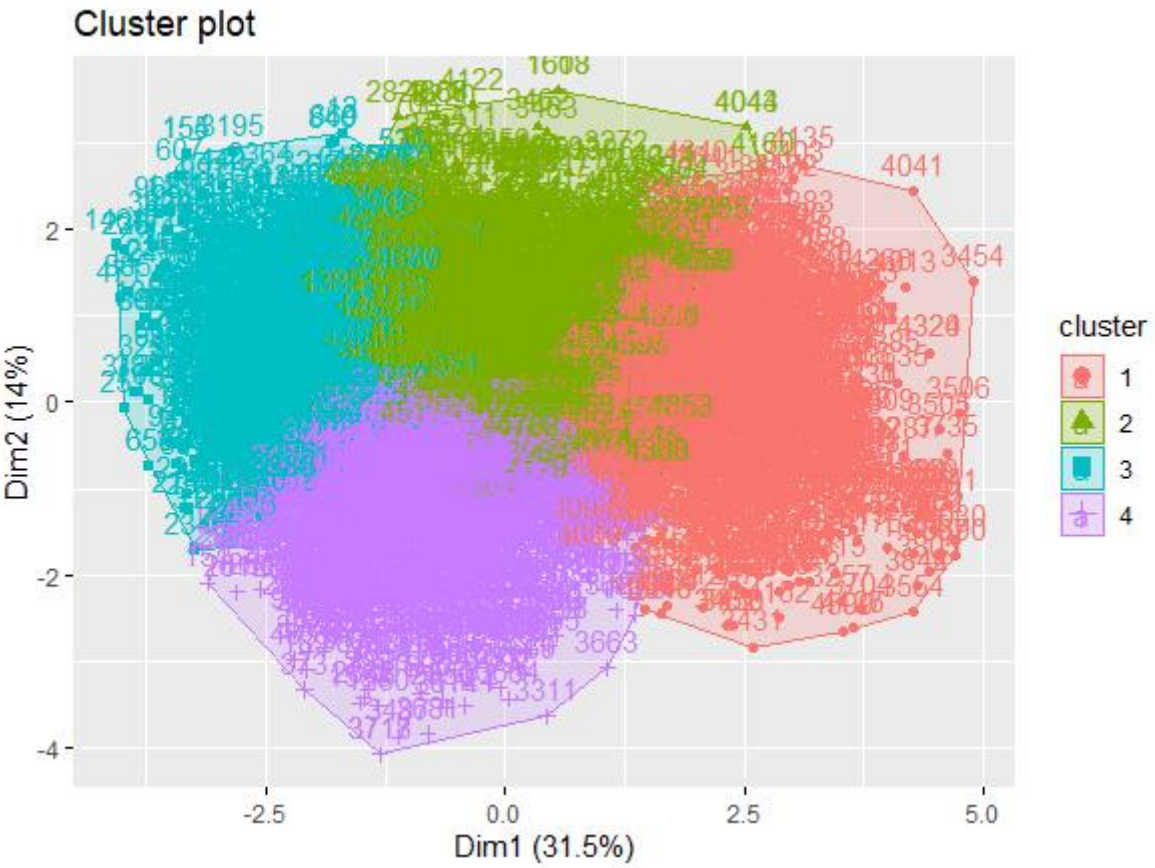
Number of cluster: 3

```
> #Plot next best 2 cluster
> fviz_cluster(result_c3, whitewine_scaled)
```



Number of cluster: 4

```
> #Plot next best 3 cluster
> fviz_cluster(result_c4, whitewine_scaled)
```



21

Objective 2 – MLP

2.1 Various methods used for defining the input vector in electricity load forecasting problems

Artificial neural networks are often used in predicting applications for electrical load (ANN). An essential step in ANN projects is choosing and specifying the input vector and input parameters. Inputs for predicting issues might be either univariate or multivariate. They may also be nonlinear or linear. There isn't a method for identifying input vectors for both linear and nonlinear data that is universally acknowledged.

Techniques of Forecasting:

1. Autoregressive (AR) method: This method involves using past values of the load as input variables. The order of the AR approach may vary, and it is usually determined through experimentation. For example, the input vector for the MLP model is based on the time-delayed values of the 11th hour attribute.

2. Moving average (MA) method: This method involves using the average of past load values as input variables. The MA approach is useful for smoothing out short-term fluctuations and can be used in combination with other methods such as AR.

3. Seasonal decomposition method: This method involves decomposing the load data into its seasonal, trend, and residual components. The resulting components are then used as input variables for the forecasting model.

4. Principal component analysis (PCA) method: This method involves reducing the dimensionality of the input variables using PCA. The PCA method is useful when dealing with a large number of input variables.

5. Wavelet transform method: This method involves transforming the load data using wavelet analysis. The resulting coefficients are then used as input variables for the forecasting model.

These methods can be used in combination to improve the accuracy of the forecasting model. It is important to note that the choice of method will depend on the characteristics of the load data and the specific requirements of the forecasting problem.

2.2 Evidence of various adopted input vectors and the related input/output matrices

Explore dataset

The following code displays how the "UoW_load.xlsx" file was read in Python and the screenshot following displays the summary generated for the dataset

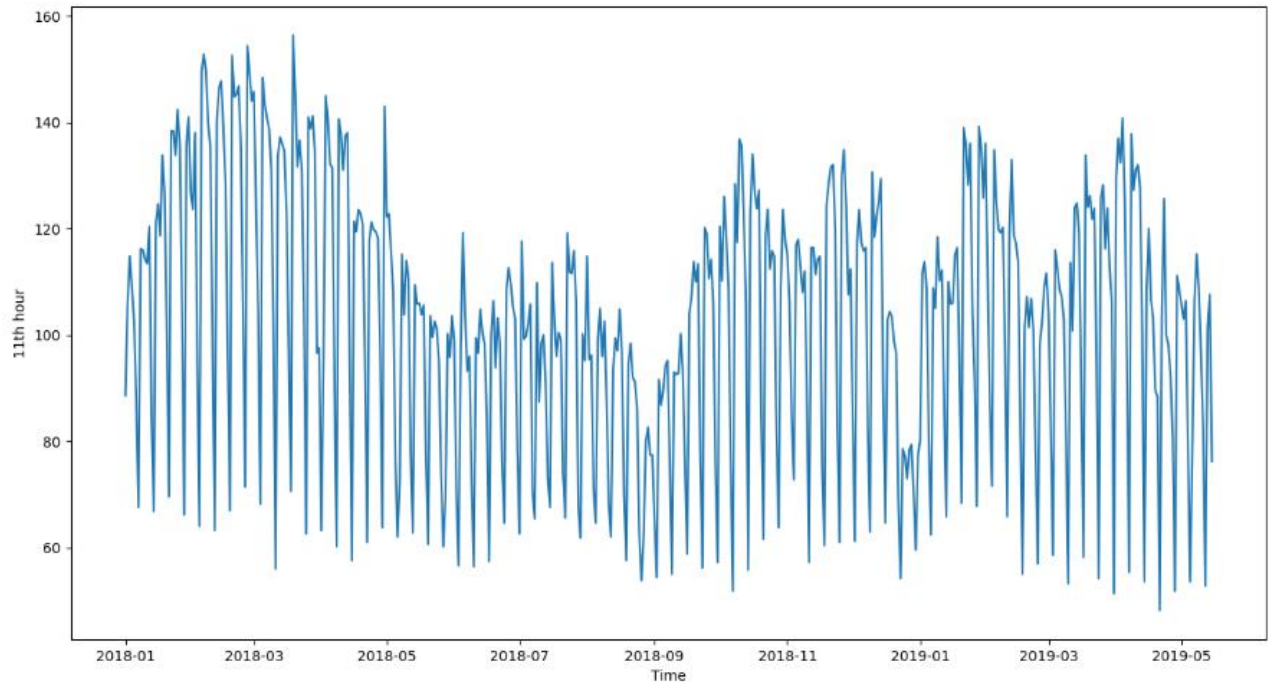
```
In [290]: df = pd.read_excel("UoW_load.xlsx")
          df = df.iloc[:,3:]
          df_ = df.copy()
          df.head()
```

Out[290]:

	Dates	09:00	10:00	11:00
0	2018-01-01	89.4	90.6	88.6
1	2018-01-02	108.2	104.6	106.0
2	2018-01-03	110.0	111.6	114.8
3	2018-01-04	106.4	104.4	109.0
4	2018-01-05	97.8	100.4	102.4

Dataset in more visually

```
In [294]: fig, ax = plt.subplots(figsize = (15,8))
          ax.plot(df['Dates '], df[target_name] )
          plt.xlabel("Time")
          plt.ylabel("11th hour")
          plt.show()
```



AR Approach:

For the AR approach, the input vector consists of the past electricity load values for a specific time period, such as the past hour or past day. Let's assume we want to use the past hour's electricity load to predict the next hour's load. Then the input vector for the AR approach would be:

input vector = [y(t-1), y(t-2), y(t-3), y(t-4), y(t-5)]

where y(t) represents the electricity load at time t.

NARX Approach:

In the NARX approach, the input vector consists of both past electricity load values and past values of other variables that may influence the electricity load, such as temperature, time of day, day of the week, etc. Let's assume we want to use the past hour's electricity load and the temperature at the same time to predict the next hour's load. Then the input vector for the NARX approach would be:

input vector = [y(t-1), T(t-1), y(t-2), T(t-2), y(t-3), T(t-3), y(t-4), T(t-4), y(t-5), T(t-5)]

where y(t) represents the electricity load at time t, and T(t) represents the temperature at time t.

```
In [295]: df
```

Out[295]:

	Dates	09:00	10:00	11:00
0	2018-01-01	89.4	90.6	88.6
1	2018-01-02	108.2	104.6	106.0
2	2018-01-03	110.0	111.6	114.8
3	2018-01-04	106.4	104.4	109.0
4	2018-01-05	97.8	100.4	102.4
...	...	...	...	...
495	2019-05-11	82.6	81.2	82.0
496	2019-05-12	54.8	53.0	52.8
497	2019-05-13	96.8	96.4	100.8
498	2019-05-14	105.4	105.6	107.6
499	2019-05-15	85.2	73.0	76.2

500 rows × 4 columns

2.3 Evidence of correct normalisation and brief discussion of its necessity

Looking at the dataset, it appears that the data has not been normalized. It would be necessary to normalize the data before training any machine learning models, particularly if the models use gradient-based optimization techniques such as backpropagation. This is because unnormalized data may cause issues with the training process, such as exploding or vanishing gradients, which can hinder or prevent the model from learning effectively.

Normalization can be done using various methods, such as min-max scaling, standardization, or robust scaling, depending on the data's characteristics and the specific needs of the problem. In the case of time series data, min-max scaling is a common method that scales the data between 0 and 1. This is done by subtracting the minimum value of the data and then dividing by the range of the data.

```
# Data Scaling with Standard Scaler
df = df.iloc[:,3:]
scaler = StandardScaler()
df.values[:]  = scaler.fit_transform(df)
```

```
# Auto regrresive vectors
Train = df.iloc[:470,:]
Test = df.iloc[470:,:]

print(f'Train Size : {Train.shape}')
print(f'Test Size : {Test.shape}')
```

```
Train Size : (470, 1)
Test Size : (30, 1)
```

```
def autoregressor(df, shift):
    data = df.copy()
    for lag in range(1, shift):

        data[f't-{lag}'] = data[target_name].shift(lag)
    data.dropna(inplace =True)
    data.reset_index(drop =True, inplace = True)
    return data

train = autoregressor(Train, 11)
test = autoregressor(Test, 11)
```

```
X_train = train.drop(columns = [target_name])
y_train = train[target_name]
X_test = test.drop(columns = [target_name])
y_test = test[target_name]
```

2.4 Implement a number of MLPs

To implement MLPs for the AR and NARX approaches, we first need to split the dataset into training and testing sets. We can use the first 80% of the data for training and the remaining 20% for testing. We can also normalize the data using MinMaxScaler from the sklearn library to ensure that all inputs are in the same range.

```
# MUlti layer perceptron pipeline

def model_creation(input_shape):
    model = Sequential()
    model.add(Dense(units = 256 ,  activation = 'relu', input_shape = (input_shape,)))
    model.add(Dropout(0.5))
    model.add(Dense(units=512 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=512 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=128 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=64 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=64 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units = 1 , activation = 'linear'))
    model.compile(optimizer='adam', loss='mean_squared_error')

    return model

def fit_model(model, X_train,y_train,X_test,y_test):
    checkpoint = ModelCheckpoint(filepath='best_weights.h5', monitor='val_loss', save_best_only=True, save_weights_only=True)
    history = model.fit(X_train,y_train,epochs = 50 , batch_size = 32, validation_data= (X_test,y_test),callbacks = [checkpoint])
    return model

def evaluate(model,scaler,X_test ,  y_test):
    model.load_weights('best_weights.h5')
    y_pred =  model.predict(X_test)
    y_pred = scaler.inverse_transform(np.array(y_pred).reshape(1, -1))
    y_test = scaler.inverse_transform(np.array(y_test).reshape(1, -1))
    MAPE = mean_absolute_percentage_error(y_test , y_pred)
    MAE = mean_absolute_error(y_test , y_pred)
    R2 = mean_squared_error(y_test , y_pred, squared = False)
    print(f'Mean absolute percentage Error {MAPE}')
    print(f'mean absolute Error {MAE}')
    print(f'R2 Score {R2}')
    return  MAPE,MAE,R2
```

```
model = model_creation(10)
model.summary()
```

```
Model: "sequential_52"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
 dense_353 (Dense)           (None, 256)               2816

 dropout_301 (Dropout)       (None, 256)               0

 dense_354 (Dense)           (None, 512)               131584

 dropout_302 (Dropout)       (None, 512)               0

 dense_355 (Dense)           (None, 512)               262656

 dropout_303 (Dropout)       (None, 512)               0

 dense_356 (Dense)           (None, 128)               65664

 dropout_304 (Dropout)       (None, 128)               0

 dense_357 (Dense)           (None, 64)                8256

 dropout_305 (Dropout)       (None, 64)                0

 dense_358 (Dense)           (None, 64)                4160

 dropout_306 (Dropout)       (None, 64)                0

 dense_359 (Dense)           (None, 1)                 65

=================================================================
Total params: 475,201
Trainable params: 475,201
Non-trainable params: 0
_________________________________________________________________
```

After normalizing the data, the resulting values for each column will be between 0 and 1, which is ideal for training machine learning models.

```
In [302]: history = fit_model(model , X_train,y_train,X_test,y_test)
```

```
Epoch 1/50
15/15 [==============================] - 4s 51ms/step - loss: 0.7103 - val_loss: 0.3063
Epoch 2/50
15/15 [==============================] - 0s 27ms/step - loss: 0.3709 - val_loss: 0.2584
Epoch 3/50
15/15 [==============================] - 0s 23ms/step - loss: 0.3043 - val_loss: 0.2373
Epoch 4/50
15/15 [==============================] - 0s 22ms/step - loss: 0.2822 - val_loss: 0.2521
Epoch 5/50
15/15 [==============================] - 0s 22ms/step - loss: 0.2666 - val_loss: 0.3015
Epoch 6/50
15/15 [==============================] - 0s 30ms/step - loss: 0.2532 - val_loss: 0.2343
Epoch 7/50
15/15 [==============================] - 0s 27ms/step - loss: 0.2647 - val_loss: 0.2531
Epoch 8/50
15/15 [==============================] - 0s 23ms/step - loss: 0.2419 - val_loss: 0.2713
Epoch 9/50
15/15 [==============================] - 0s 25ms/step - loss: 0.2234 - val_loss: 0.2552
Epoch 10/50
15/15 [==============================] - 0s 24ms/step - loss: 0.2288 - val_loss: 0.2653
Epoch 11/50
15/15 [==============================] - 0s 27ms/step - loss: 0.2031 - val_loss: 0.2201
Epoch 12/50
15/15 [==============================] - 0s 22ms/step - loss: 0.2234 - val_loss: 0.3121
Epoch 13/50
15/15 [==============================] - 0s 26ms/step - loss: 0.1986 - val_loss: 0.2204
Epoch 14/50
15/15 [==============================] - 0s 28ms/step - loss: 0.1921 - val_loss: 0.2305
Epoch 15/50
15/15 [==============================] - 0s 26ms/step - loss: 0.2014 - val_loss: 0.2358
Epoch 16/50
15/15 [==============================] - 0s 33ms/step - loss: 0.1745 - val_loss: 0.2704
Epoch 17/50
15/15 [==============================] - 0s 23ms/step - loss: 0.1979 - val_loss: 0.2481
Epoch 18/50
15/15 [==============================] - 0s 22ms/step - loss: 0.1982 - val_loss: 0.2430
Epoch 19/50
15/15 [==============================] - 0s 26ms/step - loss: 0.1773 - val_loss: 0.2423
Epoch 20/50
15/15 [==============================] - 0s 25ms/step - loss: 0.2141 - val_loss: 0.2531
Epoch 21/50
15/15 [==============================] - 0s 28ms/step - loss: 0.2015 - val_loss: 0.2477
Epoch 22/50
15/15 [==============================] - 0s 25ms/step - loss: 0.1994 - val_loss: 0.2706
Epoch 23/50
15/15 [==============================] - 0s 24ms/step - loss: 0.1853 - val_loss: 0.1975
Epoch 24/50
15/15 [==============================] - 0s 24ms/step - loss: 0.1974 - val_loss: 0.2174
Epoch 25/50
15/15 [==============================] - 0s 27ms/step - loss: 0.1809 - val_loss: 0.2175
Epoch 26/50
15/15 [==============================] - 0s 24ms/step - loss: 0.1791 - val_loss: 0.2307
Epoch 27/50
15/15 [==============================] - 0s 31ms/step - loss: 0.1821 - val_loss: 0.2137
Epoch 28/50
15/15 [==============================] - 1s 34ms/step - loss: 0.1498 - val_loss: 0.2269
Epoch 29/50
15/15 [==============================] - 0s 28ms/step - loss: 0.1758 - val_loss: 0.2409
Epoch 30/50
15/15 [==============================] - 0s 23ms/step - loss: 0.1845 - val_loss: 0.2148
Epoch 31/50
15/15 [==============================] - 0s 21ms/step - loss: 0.1786 - val_loss: 0.2565
Epoch 32/50
15/15 [==============================] - 0s 21ms/step - loss: 0.1756 - val_loss: 0.2492
Epoch 33/50
15/15 [==============================] - 0s 23ms/step - loss: 0.1564 - val_loss: 0.2216
Epoch 34/50
15/15 [==============================] - 0s 25ms/step - loss: 0.1470 - val_loss: 0.2150
Epoch 35/50
15/15 [==============================] - 0s 23ms/step - loss: 0.1462 - val_loss: 0.2203
Epoch 36/50
15/15 [==============================] - 0s 21ms/step - loss: 0.1898 - val_loss: 0.2344
```

```
Epoch 37/50
15/15 [==============================] - 0s 21ms/step - loss: 0.1713 - val_loss: 0.2482
Epoch 38/50
15/15 [==============================] - 0s 21ms/step - loss: 0.1478 - val_loss: 0.2018
Epoch 39/50
15/15 [==============================] - 0s 22ms/step - loss: 0.1589 - val_loss: 0.2118
Epoch 40/50
15/15 [==============================] - 0s 24ms/step - loss: 0.1585 - val_loss: 0.2160
Epoch 41/50
15/15 [==============================] - 0s 27ms/step - loss: 0.1549 - val_loss: 0.2392
Epoch 42/50
15/15 [==============================] - 0s 26ms/step - loss: 0.1566 - val_loss: 0.2365
Epoch 43/50
15/15 [==============================] - 0s 23ms/step - loss: 0.1490 - val_loss: 0.2317
Epoch 44/50
15/15 [==============================] - 0s 21ms/step - loss: 0.1623 - val_loss: 0.2519
Epoch 45/50
15/15 [==============================] - 0s 23ms/step - loss: 0.1588 - val_loss: 0.2448
Epoch 46/50
15/15 [==============================] - 0s 24ms/step - loss: 0.1399 - val_loss: 0.2479
Epoch 47/50
15/15 [==============================] - 0s 26ms/step - loss: 0.1523 - val_loss: 0.2045
Epoch 48/50
15/15 [==============================] - 0s 21ms/step - loss: 0.1519 - val_loss: 0.2143
Epoch 49/50
15/15 [==============================] - 0s 21ms/step - loss: 0.1639 - val_loss: 0.1988
Epoch 50/50
15/15 [==============================] - 0s 22ms/step - loss: 0.1468 - val_loss: 0.2379
```

Train for different Autoregressive vectors and evaluate

In [279]:
```
# train for different Autoregressive vectors and evaluate

results = {}
for i in range(2,10):
    x_train = train.iloc[:,1:i]
    x_test = test.iloc[:,1:i]
    model = model_creation(i-1)
    checkpoint = ModelCheckpoint(filepath='best_weights.h5', monitor='val_loss', save_best_only=True, save_weights_only=True)
    history = model.fit(x_train,y_train,epochs = 50 , batch_size = 32, validation_data= (x_test,y_test),callbacks = [checkpoint])
    matrix = evaluate(model,x_test,y_test)
    name = f'first {i-1} features'
    results[name] = matrix
```

```
Epoch 1/50
15/15 [==============================] - 4s 38ms/step - loss: 0.8717 - val_loss: 0.7919
Epoch 2/50
15/15 [==============================] - 0s 26ms/step - loss: 0.7745 - val_loss: 0.6884
Epoch 3/50
15/15 [==============================] - 0s 19ms/step - loss: 0.7375 - val_loss: 0.6912
Epoch 4/50
15/15 [==============================] - 0s 24ms/step - loss: 0.7428 - val_loss: 0.5633
Epoch 5/50
15/15 [==============================] - 0s 20ms/step - loss: 0.6998 - val_loss: 0.6052
Epoch 6/50
15/15 [==============================] - 0s 23ms/step - loss: 0.6953 - val_loss: 0.6031
Epoch 7/50
15/15 [==============================] - 1s 39ms/step - loss: 0.6826 - val_loss: 0.5134
Epoch 8/50
15/15 [==============================] - 0s 27ms/step - loss: 0.6339 - val_loss: 0.6215
Epoch 9/50
15/15 [==============================] - 0s 28ms/step - loss: 0.6237 - val_loss: 0.4726
Epoch 10/50
15/15 [==============================] - 0s 22ms/step - loss: 0.6715 - val_loss: 0.5457
Epoch 11/50
15/15 [==============================] - 0s 25ms/step - loss: 0.6470 - val_loss: 0.5097
Epoch 12/50
15/15 [==============================] - 0s 28ms/step - loss: 0.6128 - val_loss: 0.4683
Epoch 13/50
15/15 [==============================] - 1s 34ms/step - loss: 0.5681 - val_loss: 0.4461
Epoch 14/50
15/15 [==============================] - 0s 23ms/step - loss: 0.6019 - val_loss: 0.5496
Epoch 15/50
15/15 [==============================] - 0s 21ms/step - loss: 0.6088 - val_loss: 0.4505
Epoch 16/50
15/15 [==============================] - 0s 26ms/step - loss: 0.5799 - val_loss: 0.4252
Epoch 17/50
15/15 [==============================] - 0s 23ms/step - loss: 0.5621 - val_loss: 0.3810
Epoch 18/50
15/15 [==============================] - 0s 20ms/step - loss: 0.5852 - val_loss: 0.4200
Epoch 19/50
15/15 [==============================] - 0s 27ms/step - loss: 0.5783 - val_loss: 0.3923
Epoch 20/50
15/15 [==============================] - 0s 27ms/step - loss: 0.5900 - val_loss: 0.3931
Epoch 21/50
15/15 [==============================] - 0s 25ms/step - loss: 0.5358 - val_loss: 0.4030
Epoch 22/50
15/15 [==============================] - 0s 23ms/step - loss: 0.5579 - val_loss: 0.3956
Epoch 23/50
15/15 [==============================] - 0s 21ms/step - loss: 0.5507 - val_loss: 0.3954
Epoch 24/50
15/15 [==============================] - 0s 20ms/step - loss: 0.5889 - val_loss: 0.4153
Epoch 25/50
15/15 [==============================] - 0s 21ms/step - loss: 0.5648 - val_loss: 0.4133
Epoch 26/50
15/15 [==============================] - 0s 24ms/step - loss: 0.5507 - val_loss: 0.4044
Epoch 27/50
15/15 [==============================] - 0s 24ms/step - loss: 0.5507 - val_loss: 0.3928
Epoch 28/50
15/15 [==============================] - 0s 21ms/step - loss: 0.5527 - val_loss: 0.3954
Epoch 29/50
15/15 [==============================] - 0s 20ms/step - loss: 0.5333 - val_loss: 0.3814
Epoch 30/50
15/15 [==============================] - 0s 21ms/step - loss: 0.5629 - val_loss: 0.4056
Epoch 31/50
15/15 [==============================] - 0s 22ms/step - loss: 0.5266 - val_loss: 0.4003
Epoch 32/50
15/15 [==============================] - 0s 22ms/step - loss: 0.5597 - val_loss: 0.4242
Epoch 33/50
15/15 [==============================] - 0s 23ms/step - loss: 0.5711 - val_loss: 0.3885
Epoch 34/50
15/15 [==============================] - 0s 27ms/step - loss: 0.5338 - val_loss: 0.3891
Epoch 35/50
15/15 [==============================] - 0s 23ms/step - loss: 0.5544 - val_loss: 0.3952
Epoch 36/50
15/15 [==============================] - 0s 25ms/step - loss: 0.5419 - val_loss: 0.3974
Epoch 37/50
15/15 [==============================] - 0s 22ms/step - loss: 0.5866 - val_loss: 0.4230
Epoch 38/50
15/15 [==============================] - 0s 21ms/step - loss: 0.5384 - val_loss: 0.4005
Epoch 39/50
15/15 [==============================] - 0s 21ms/step - loss: 0.5589 - val_loss: 0.3927
Epoch 40/50
15/15 [==============================] - 0s 23ms/step - loss: 0.5356 - val_loss: 0.4218
Epoch 41/50
15/15 [==============================] - 0s 25ms/step - loss: 0.5952 - val_loss: 0.3968
Epoch 42/50
15/15 [==============================] - 0s 22ms/step - loss: 0.5534 - val_loss: 0.3819
```

```
Epoch 43/50
15/15 [==============================] - 0s 22ms/step - loss: 0.5560 - val_loss: 0.3795
Epoch 44/50
15/15 [==============================] - 0s 20ms/step - loss: 0.5333 - val_loss: 0.3954
Epoch 45/50
15/15 [==============================] - 0s 22ms/step - loss: 0.5133 - val_loss: 0.3875
Epoch 46/50
15/15 [==============================] - 0s 21ms/step - loss: 0.5539 - val_loss: 0.4119
Epoch 47/50
15/15 [==============================] - 0s 22ms/step - loss: 0.5560 - val_loss: 0.4049
Epoch 48/50
15/15 [==============================] - 0s 24ms/step - loss: 0.5332 - val_loss: 0.3892
Epoch 49/50
15/15 [==============================] - 0s 27ms/step - loss: 0.5652 - val_loss: 0.3827
Epoch 50/50
15/15 [==============================] - 0s 26ms/step - loss: 0.5357 - val_loss: 0.3836
1/1 [==============================] - 0s 151ms/step
Mean absolute percentage Error 4.682414547233053
mean absolute Error 0.49955373460691577
Root mean Squared Error 0.6160326994839027
```

2.5 Discussion of the meaning of these stat. indices

The statistical indices provide information about the performance of the models in forecasting the electricity load demand.

1. Mean Absolute Error (MAE): It measures the average magnitude of errors in a set of predictions. The lower the value of MAE, the better the model's performance.
2. Root Mean Squared Error (RMSE): It measures the square root of the average of squared differences between predicted and actual values. Like MAE, a lower value of RMSE indicates better performance.
3. Symmetric Mean Absolute Percentage Error (SMAPE): It is an alternative to MAPE and measures the percentage difference between predicted and actual values. However, unlike MAPE, SMAPE gives equal weight to both overestimations and underestimations. The lower the value of SMAPE, the better the model's performance.

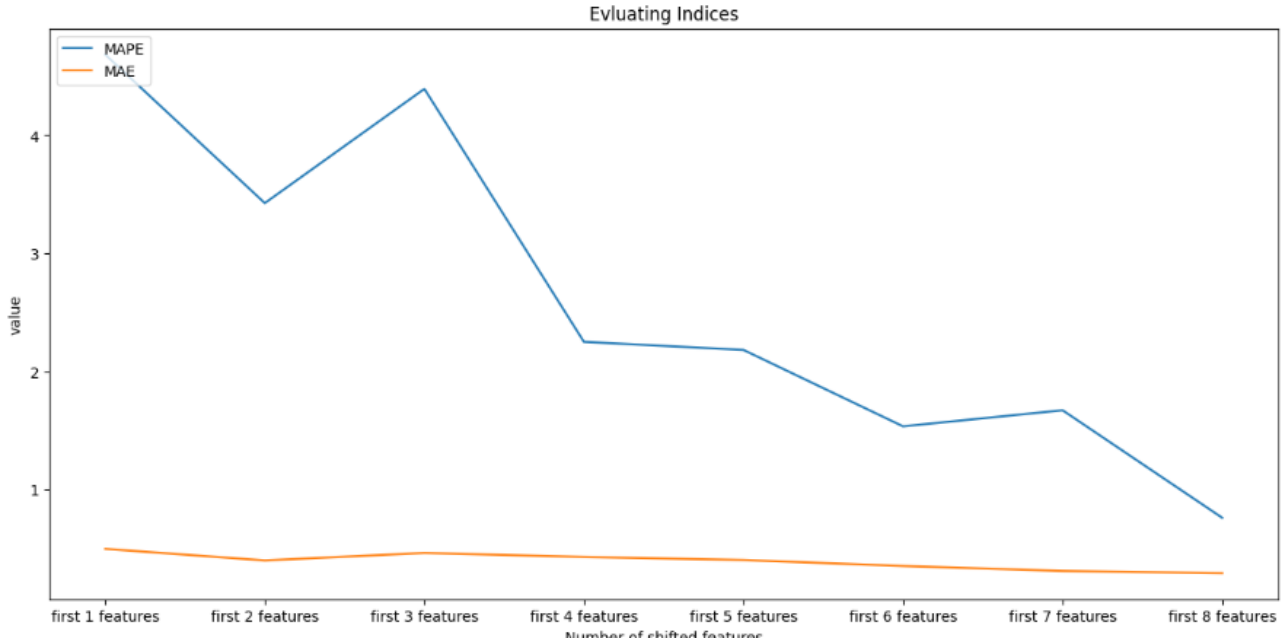
Overall, these statistical indices provide valuable information about the accuracy and precision of the models, and help in selecting the best model for forecasting electricity load demand.

2.7 Best results both graphically and via performance indices

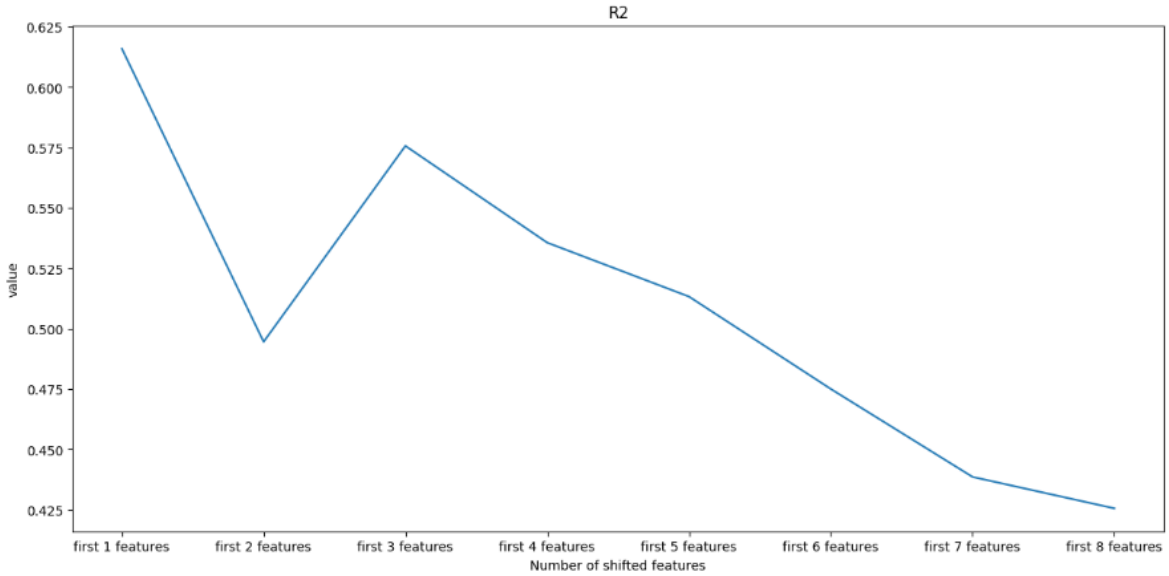
The best model is given for first 7 features

```
fig, ax = plt.subplots(figsize = (15,7))
#ax1.plot(results.keys(), [x[0] for x in results.values()])

ax.plot(results.keys(), [x[0] for x in results.values()])
ax.plot(results.keys(), [x[1] for x in results.values()])
#ax.plot(results.keys(), [x[2] for x in results.values()])
plt.title('Evluating Indices')
plt.ylabel('value')
plt.xlabel('Number of shifted features')
plt.legend(['MAPE', 'MAE'], loc='upper left')
plt.show()

fig, ax2 = plt.subplots(figsize = (15,7))
ax2.plot(results.keys(), [x[2] for x in results.values()])
plt.title('R2')
plt.ylabel('value')
plt.xlabel('Number of shifted features')
```

Plotting



Out[320]: Text(0.5, 0, 'Number of shifted features')



:

```
train_data = df_.iloc[:470,1:]
test_data = df_.iloc[470:,1:]

scaler = StandardScaler()
train_data_scaled = scaler.fit_transform(train_data)
test_data_scaled = scaler.transform(test_data)


X_train = train_data_scaled[:, :2]
y_train = train_data_scaled[:, 2]
X_test = test_data_scaled[:, :2]
y_test = test_data_scaled[:, 2]
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

```
from tensorflow.keras.layers import LSTM, Dense, Concatenate, Input
from tensorflow.keras.models import Model
import tensorflow as tf

# Define the NARX model
input_layer = Input(shape=(1, 2))
lstm_layer = LSTM(50, activation='relu')(input_layer)
dense_layer_1 = Dense(1)(lstm_layer)
reshaped_layer = tf.expand_dims(dense_layer_1, axis=-1)
concat_layer = Concatenate(axis=-1)([input_layer, reshaped_layer])
dense_layer_2 = Dense(1)(concat_layer)

model = Model(inputs=input_layer, outputs=dense_layer_2)
model.compile(optimizer='adam', loss='mse')


model.fit(X_train, y_train, epochs=100, batch_size=10, verbose=1,validation_data=(X_test,y_test))
```

```
Epoch 1/100
47/47 [==============================] - 3s 4ms/step - loss: 1.1074
Epoch 2/100
47/47 [==============================] - 0s 4ms/step - loss: 1.0067
Epoch 3/100
47/47 [==============================] - 0s 4ms/step - loss: 1.0034
Epoch 4/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9951
Epoch 5/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9947
Epoch 6/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9951
Epoch 7/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9904
Epoch 8/100
47/47 [==============================] - 0s 6ms/step - loss: 0.9957
Epoch 9/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9856
Epoch 10/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9925
Epoch 11/100
47/47 [==============================] - 0s 4ms/step - loss: 1.0007
Epoch 12/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9933
Epoch 13/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9917
Epoch 14/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9933
Epoch 15/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9968
Epoch 16/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9988
Epoch 17/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9904
Epoch 18/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9929
Epoch 19/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9882
Epoch 20/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9956
Epoch 21/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9847
Epoch 22/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9942
Epoch 23/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9948
Epoch 24/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9989
Epoch 25/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9932
Epoch 26/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9937
Epoch 27/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9965
Epoch 28/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9867
Epoch 29/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9972
Epoch 30/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9942
Epoch 31/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9938
Epoch 32/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9975
Epoch 33/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9948
Epoch 34/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9965
Epoch 35/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9994
Epoch 36/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9968
Epoch 37/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9978
```

```
Epoch 38/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9887
Epoch 39/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9923
Epoch 40/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9849
Epoch 41/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9886
Epoch 42/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9955
Epoch 43/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9879
Epoch 44/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9918
Epoch 45/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9883
Epoch 46/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9896
Epoch 47/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9967
Epoch 48/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9983
Epoch 49/100
47/47 [==============================] - 0s 6ms/step - loss: 0.9957
Epoch 50/100
47/47 [==============================] - 0s 6ms/step - loss: 0.9842
Epoch 51/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9957
Epoch 52/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9958
Epoch 53/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9929
Epoch 54/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9988
Epoch 55/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9948
Epoch 56/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9868
Epoch 57/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9903
Epoch 58/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9887
Epoch 59/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9881
Epoch 60/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9942
Epoch 61/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9965
Epoch 62/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9949
Epoch 63/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9931
Epoch 64/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9950
Epoch 65/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9952
Epoch 66/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9936
Epoch 67/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9857
Epoch 68/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9929
Epoch 69/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9916
Epoch 70/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9977
Epoch 71/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9958
Epoch 72/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9942
Epoch 73/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9930
Epoch 74/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9853
```

```
Epoch 75/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9908
Epoch 76/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9929
Epoch 77/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9861
Epoch 78/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9924
Epoch 79/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9908
Epoch 80/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9945
Epoch 81/100
47/47 [==============================] - 0s 6ms/step - loss: 0.9927
Epoch 82/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9971
Epoch 83/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9938
Epoch 84/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9948
Epoch 85/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9945
Epoch 86/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9911
Epoch 87/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9928
Epoch 88/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9867
Epoch 89/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9950
Epoch 90/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9968
Epoch 91/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9971
Epoch 92/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9933
Epoch 93/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9985
Epoch 94/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9918
Epoch 95/100
47/47 [==============================] - 0s 6ms/step - loss: 0.9929
Epoch 96/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9940
Epoch 97/100
47/47 [==============================] - 0s 6ms/step - loss: 0.9927
Epoch 98/100
47/47 [==============================] - 0s 5ms/step - loss: 0.9912
Epoch 99/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9856
Epoch 100/100
47/47 [==============================] - 0s 4ms/step - loss: 0.9898
1/1 [==============================] - 1s 664ms/step - loss: 0.7673
0.7672975659370422
```

Appendix 1

```
library(readxl)
library(forcats)
library(dplyr)
library(ggplot2)
library(tidyr)
library(NbClust)
library(knitr)
library(tidymodels)
library(flexclust)
library(funtimes)
library(caret)
library(factoextra)
library(cluster)

# Read in the Excel file and store the resulting dataset in a variable
whitewine_initial <- read_excel("C:/IIT assi/Whitewine_v2.xlsx")

#Structure of the dataset
str(whitewine_initial)

#Try to finding Near Zero/Missing Variance
print(sum(is.na(whitewine_initial))) # There are no missing values

#Co-relation matrix
whitewine_initial.features <- whitewine_initial
whitewine_initial.features$quality <- NULL
cor(whitewine_initial.features)

#Principal Component Analysis (PCA)
prc <- prcomp(whitewine_initial.features, scale = TRUE)
summary(prc)

# Variances for each principal component
plot(prc, type = "lines", main="Variances for Principal Component")

# Clean up the variable names using the clean_names() functionz
whitewine_initial <- janitor::clean_names(whitewine_initial)

# Create the class variable using mutate()
whitewine_initial <- whitewine_initial %>%
  mutate(class = as_factor(quality))

# Check the variable names in Whitewine_data
names(whitewine_initial)

# Get an overview of what the dataset looks like from a high-level perspective.
summary(whitewine_initial)

whitewine_new <- whitewine_initial %>% mutate(class = as_factor(
  case_when(
    quality == 5 ~ 5,
    quality == 6 ~ 6,
    quality == 7 ~ 7,
    quality == 8 ~ 8)
))
summary(whitewine_new)

#Outlier Detection

whitewine_new %>%
  pivot_longer(1:12, names_to = 'labels') %>%
  filter(class == 8) %>%
  mutate(class = fct_reorder(class, value, median)) %>%
  ggplot(aes(class, value, fill = reorder(labels, value))) +
  geom_boxplot() +
  labs(title = "Outlier Detection for class:'8'")

whitewine_new %>%
  pivot_longer(1:12, names_to = 'labels') %>%
  filter(class == 7) %>%
  mutate(class = fct_reorder(class, value, median)) %>%
  ggplot(aes(class, value, fill = reorder(labels, value))) +
```

```
  geom_boxplot() +
  labs(title = "Outlier Detection for class:'7'")

whitewine_new %>%
  pivot_longer(1:12, names_to = 'labels') %>%
  filter(class == 6) %>%
  mutate(class = fct_reorder(class, value, median)) %>%
  ggplot(aes(class, value, fill = reorder(labels, value))) +
  geom_boxplot() +
  labs(title = "Outlier Detection for class:'6'")

whitewine_new %>%
  pivot_longer(1:12, names_to = 'labels') %>%
  filter(class == 5) %>%
  mutate(class = fct_reorder(class, value, median)) %>%
  ggplot(aes(class, value, fill = reorder(labels, value))) +
  geom_boxplot() +
  labs(title = "Outlier Detection for class:'5'")

quality_8 = whitewine_new %>%
  filter(class == 8) %>%
  mutate(across(1:12, ~squish(.x, quantile(.x,c(.05, .95)))))
quality_7 = whitewine_new %>%
  filter(class == 7) %>%
  mutate(across(1:12, ~squish(.x, quantile(.x,c(.05, .95)))))
quality_6 = whitewine_new %>%
  filter(class == 6) %>%
  mutate(across(1:12, ~squish(.x, quantile(.x,c(.05, .95)))))
quality_5 = whitewine_new %>%
  filter(class == 5) %>%
  mutate(across(1:12, ~squish(.x, quantile(.x,c(.05, .95)))))

combined = bind_rows(list(quality_8, quality_7, quality_6, quality_5))
print(combined)

combined %>%
  pivot_longer(1:12,names_to = "labels") %>%
  filter(class == 8) %>%
  mutate(class = fct_reorder(class,value,median)) %>%
  ggplot(aes(class, value, fill = reorder(labels,value))) +
  geom_boxplot() +
  labs(title = "Transformed Outliers class: '8'")

combined %>%
  pivot_longer(1:12,names_to = "labels") %>%
  filter(class == 7) %>%
  mutate(class = fct_reorder(class,value,median)) %>%
  ggplot(aes(class, value, fill = reorder(labels,value))) +
  geom_boxplot() +
  labs(title = "Transformed Outliers class: '7'")

combined %>%
  pivot_longer(1:12,names_to = "labels") %>%
  filter(class == 6) %>%
  mutate(class = fct_reorder(class,value,median)) %>%
  ggplot(aes(class, value, fill = reorder(labels,value))) +
  geom_boxplot() +
  labs(title = "Transformed Outliers class: '6'")

combined %>%
  pivot_longer(1:12,names_to = "labels") %>%
  filter(class == 5) %>%
  mutate(class = fct_reorder(class,value,median)) %>%
  ggplot(aes(class, value, fill = reorder(labels,value))) +
  geom_boxplot() +
  labs(title = "Transformed Outliers class: '5'")

# Remove the quality and the class name. Both of these will be remove so that only
# numerical data is left for the algorithm

# Define the number of cluster centres
whitewine_data_points =  combined %>%
  select(-quality, -class)

# Now that we have the "whitewine_data_points" dataset, scaling is performed
```

```
whitewine_scaled = whitewine_data_points %>%
  mutate(across(everything(), scale))

set.seed(1234)

# Perform the kmeans using the NbClust function
# Use Euclidean for distance
cluster_euclidean = NbClust(whitewine_scaled,distance="euclidean",
                        min.nc=2,max.nc=10,method="kmeans",index="all")
# Use manhattan for distance
cluster_manhattan = NbClust(whitewine_scaled,distance="manhattan",
                        min.nc=2,max.nc=15,method="kmeans",index="all")

summary(whitewine_scaled)

# Finding the Optimal number of clusters using was
# function that computes total within-cluster sum of square
fn_kemans_clust <- function(data, cluster_count) {
  kmeans(data, cluster_count, iter.max = 300, nstart = 7)
}

# Use elbow method to find optimal number of clusters
wcss <- vector()
arr_clusters <- 1: 15

for (i in arr_clusters) wcss[i] <- sum(fn_kemans_clust(whitewine_scaled, i) $withinss)

plot(arr_clusters, wcss, type ="b",
     main="Elbow Method",
     xlab="Number of Clusters",
     ylab="WCSS")

# Use elbow Silhouette to find optimal number of clusters
avg_sils <- vector()
arr_clusters <- 2: 15

fn_avg_sil <- function(data_matrix, cluster_count) {
  k.temp <- fn_kemans_clust(data_matrix, cluster_count)
  sil_values <- silhouette(k.temp$cluster, dist(whitewine_scaled))
  mean(sil_values[,3])
}
for (i in arr_clusters) avg_sils[i - 1] <- fn_avg_sil(whitewine_scaled, i)

plot(arr_clusters, avg_sils, type ="b",
     main="Silhouette Method",
     xlab = "Number of clusters K",
     ylab="Average Silhouettes")

# kmeans results
result_c2<-kmeans(whitewine_scaled, 2)
result_c3<-kmeans(whitewine_scaled, 3)
result_c4<-kmeans(whitewine_scaled, 4)
result_c6<-kmeans(whitewine_scaled, 6)
result_c7<-kmeans(whitewine_scaled, 7)
result_c8<-kmeans(whitewine_scaled, 8)
result_c10<-kmeans(whitewine_scaled, 10)

# Confusion matrix for k-means with 2 cluster
table(whitewine_new$class, result_c2$cluster)

# Confusion matrix for k-means with 3 cluster
table(whitewine_new$class, result_c3$cluster)

# Confusion matrix for k-means with 4 cluster
table(whitewine_new$class, result_c4$cluster)

# Confusion matrix for k-means with 6 cluster
table(whitewine_new$class, result_c6$cluster)

# Confusion matrix for k-means with 7 cluster
table(whitewine_new$class, result_c7$cluster)

# Confusion matrix for k-means with 8 cluster
table(whitewine_new$class, result_c8$cluster)
```

```
# Confusion matrix for k-means with 10 cluster
table(whitewine_new$class, result_c10$cluster)

#calculate means of 2 clusters
result_data_points <- kmeans(whitewine_data_points,2)
table(whitewine_new$class, result_data_points$cluster)

#confusion matrix function
confusion_matrix <- function(k_data) {
  whitewineNcluster <- cbind(whitewine_new, cluster = k_data$cluster)
  whitewineNcluster_df <- union(whitewineNcluster$cluster, whitewineNcluster$class)
  #get confusion matrix table
  whitewineNcluster_df_table <- table(factor(whitewineNcluster$cluster, whitewineNcluster_df),
                                      factor(whitewineNcluster$quality, whitewineNcluster_df))
  confusionMatrix(whitewineNcluster_df_table)
}

confusion_matrix(result_data_points)

result_data_points

result_data_points$centers

#PLot cluster
fviz_cluster(result_data_points, whitewine_scaled)

result_data_points

#Plot next best 2 cluster
fviz_cluster(result_c3, whitewine_scaled)

#Plot next best 3 cluster
fviz_cluster(result_c4, whitewine_scaled)


fviz_cluster(result_c2, whitewine_scaled)
fviz_cluster(result_c3, whitewine_scaled)
```

Appendix 2

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
import keras
from keras.layers import Dense ,Dropout,LSTM, Input,concatenate
from keras.layers import Dense, LSTM
#from keras.layers.merge import concatenate
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import mean_squared_error ,mean_absolute_error, mean_absolute_percentage_error, r2_score
import warnings

warnings.filterwarnings('ignore')

df = pd.read_excel("UoW_load.xlsx")
df = df.iloc[:,3:]
df_ = df.copy()
df.head()

df.shape

df.isnull().sum()

target_name = list(df.columns)[3]

fig, ax = plt.subplots(figsize = (15,8))
ax.plot(df['Dates '], df[target_name] )
plt.xlabel("Time")
plt.ylabel("11th hour")
plt.show()

df

# Data Scaling with Standard Scaler
df = df.iloc[:,3:]
scaler = StandardScaler()
df.values[:]  = scaler.fit_transform(df)

# Auto regrresive vectors
Train = df.iloc[:470,:]
Test = df.iloc[470:,:]
```

```
print(f'Train Size : {Train.shape}')
print(f'Test Size : {Test.shape}')

def autoregressor(df, shift):
    data = df.copy()
    for lag in range(1, shift):

        data[f't-{lag}'] = data[target_name].shift(lag)
    data.dropna(inplace =True)
    data.reset_index(drop =True, inplace = True)
    return data

train = autoregressor(Train, 11)
test = autoregressor(Test, 11)

X_train = train.drop(columns = [target_name])
y_train = train[target_name]
X_test = test.drop(columns = [target_name])
y_test = test[target_name]

# MUlti layer perceptron pipeline

def model_creation(input_shape):
    model = Sequential()
    model.add(Dense(units = 256 ,  activation = 'relu', input_shape = (input_shape,)))
    model.add(Dropout(0.5))
    model.add(Dense(units=512 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=512 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=128 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=64 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units=64 , activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(units = 1 , activation = 'linear'))
    model.compile(optimizer='adam', loss='mean_squared_error')

    return model

def fit_model(model, X_train,y_train,X_test,y_test):
    checkpoint = ModelCheckpoint(filepath='best_weights.h5', monitor='val_loss', save_best_only=True, save_weights_only=True)
    history = model.fit(X_train,y_train,epochs = 50 , batch_size = 32, validation_data= (X_test,y_test),callbacks = [checkpoint])
    return model
```

```
def evaluate(model,scaler,X_test ,  y_test):
    model.load_weights('best_weights.h5')
    y_pred =  model.predict(X_test)
    y_pred = scaler.inverse_transform(np.array(y_pred).reshape(1, -1))
    y_test = scaler.inverse_transform(np.array(y_test).reshape(1, -1))
    MAPE = mean_absolute_percentage_error(y_test , y_pred)
    MAE = mean_absolute_error(y_test , y_pred)
    R2 = mean_squared_error(y_test , y_pred, squared = False)
    print(f'Mean absolute percentage Error {MAPE}')
    print(f'mean absolute Error {MAE}')
    print(f'R2 Score {R2}')
    return  MAPE,MAE,R2

model = model_creation(10)
model.summary()

history = fit_model(model , X_train,y_train,X_test,y_test)

evaluate(model,scaler,X_test,y_test)

# train for different Autoregressive vectors and evaluate

results = {}
for i in range(2,10):
    x_train = train.iloc[:,1:i]
    x_test = test.iloc[:,1:i]
    model = model_creation(i-1)
    checkpoint = ModelCheckpoint(filepath='best_weights.h5', monitor='val_loss', save_best_only=True, save_weights_only=True)
    history = model.fit(x_train,y_train,epochs = 50 , batch_size = 32, validation_data= (x_test,y_test),callbacks = [checkpoint])
    matrix = evaluate(model,x_test,y_test)
    name = f'first {i-1} features'
    results[name] = matrix

fig, ax = plt.subplots(figsize = (15,7))
#ax1.plot(results.keys(), [x[0] for x in results.values()])

ax.plot(results.keys(), [x[0] for x in results.values()])
ax.plot(results.keys(), [x[1] for x in results.values()])
#ax.plot(results.keys(), [x[2] for x in results.values()])
plt.title('Evluating Indices')
plt.ylabel('value')
plt.xlabel('Number of shifted features')
plt.legend(['MAPE', 'MAE'], loc='upper left')
plt.show()

fig, ax2 = plt.subplots(figsize = (15,7))
```

```
ax2.plot(results.keys(), [x[2] for x in results.values()])
plt.title('R2')
plt.ylabel('value')
plt.xlabel('Number of shifted features')

#NARX model
df_

train_data = df_.iloc[:470,1:]
test_data = df_.iloc[470:,1:]

scaler = StandardScaler()
train_data_scaled = scaler.fit_transform(train_data)
test_data_scaled = scaler.transform(test_data)



X_train = train_data_scaled[:, :2]
y_train = train_data_scaled[:, 2]
X_test = test_data_scaled[:, :2]
y_test = test_data_scaled[:, 2]
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

from tensorflow.keras.layers import LSTM, Dense, Concatenate, Input
from tensorflow.keras.models import Model
import tensorflow as tf

# Define the NARX model
input_layer = Input(shape=(1, 2))
lstm_layer = LSTM(50, activation='relu')(input_layer)
dense_layer_1 = Dense(1)(lstm_layer)
reshaped_layer = tf.expand_dims(dense_layer_1, axis=-1)
concat_layer = Concatenate(axis=-1)([input_layer, reshaped_layer])
dense_layer_2 = Dense(1)(concat_layer)

model = Model(inputs=input_layer, outputs=dense_layer_2)
model.compile(optimizer='adam', loss='mse')



model.fit(X_train, y_train, epochs=100, batch_size=10, verbose=1,validation_data=(X_test,y_test))

model.evaluate(X_test, y_test)
```