INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology
Department of Computing (BSc.) in Computer Science

Module: 5DATA001C.2

Machine Learning and Data Mining

Module Leader: Mr. Achala Aponso

Date of Submission	: 09/052022
Name	: K.A.D.S.T.Kumarapeli
UOW No	: W1790958
IIT No	: 2019952

Contents

Clustering Part

With the given problem description, we will be able to perform all tasks whit in below steps.

1. Preprocessing
    i. Remove Outliares
    ii. Normalization
2. Define Number of Clusters
    i. Manual Method
    ii. Automated Methods
3. k-means Clustering
    i. k=2
    ii. k=3
    iii. k=4
4. PCA
6. Clustering

EDA- Get High Level Idea about dataset

```
summary(df)
```
✓ 0.4s

```
fixed_acidity    volatile_acidity  citric_acid     residual_sugar
Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700
Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.300
Mean   : 6.842   Mean   :0.2744   Mean   :0.3352   Mean   : 6.455
3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.:10.000
Max.   :14.200   Max.   :0.9650   Max.   :1.6600   Max.   :65.800
  chlorides       free_sulfur_dioxide total_sulfur_dioxide    density
Min.   :0.00900   Min.   :  2.00      Min.   :  9.0        Min.   :0.9871
1st Qu.:0.03600   1st Qu.: 24.00      1st Qu.:109.0        1st Qu.:0.9917
Median :0.04300   Median : 34.00      Median :134.0        Median :0.9937
Mean   :0.04561   Mean   : 35.65      Mean   :138.7        Mean   :0.9940
3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:167.0        3rd Qu.:0.9961
Max.   :0.34600   Max.   :131.00      Max.   :344.0        Max.   :1.0390
     p_h          sulphates         alcohol         quality
Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :5.000
1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50   1st Qu.:5.000
Median :3.180   Median :0.4800   Median :10.40   Median :6.000
Mean   :3.188   Mean   :0.4904   Mean   :10.53   Mean   :5.952
3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.000
Max.   :3.820   Max.   :1.0800   Max.   :14.20   Max.   :8.000
```

From above figure we will be able to get an idea about the dataset and the data distribution with Min and Max Values and Some Statistical Measures of the Dataset.

There are 4710 total observations and 12 features in this dataset. There is a column called quality which is ordinal variable to represent quality of each red wine sample and it has four qualities such as 5,6,7,8 and in this project, I will map these values to 1,2,3,4 and it will be help in next stages.

replacing each quality measures into 1-4 range

1 wrost and 4 is best

```
df["quality"][df["quality"] == 5] <- 1
df["quality"][df["quality"] == 6] <- 2
df["quality"][df["quality"] == 7] <- 3
df["quality"][df["quality"] == 8] <- 4
```
75] ✓ 0.2s

Missing Values!

```
#check null values
sum(is.na(df))
```

✓ 0.4s

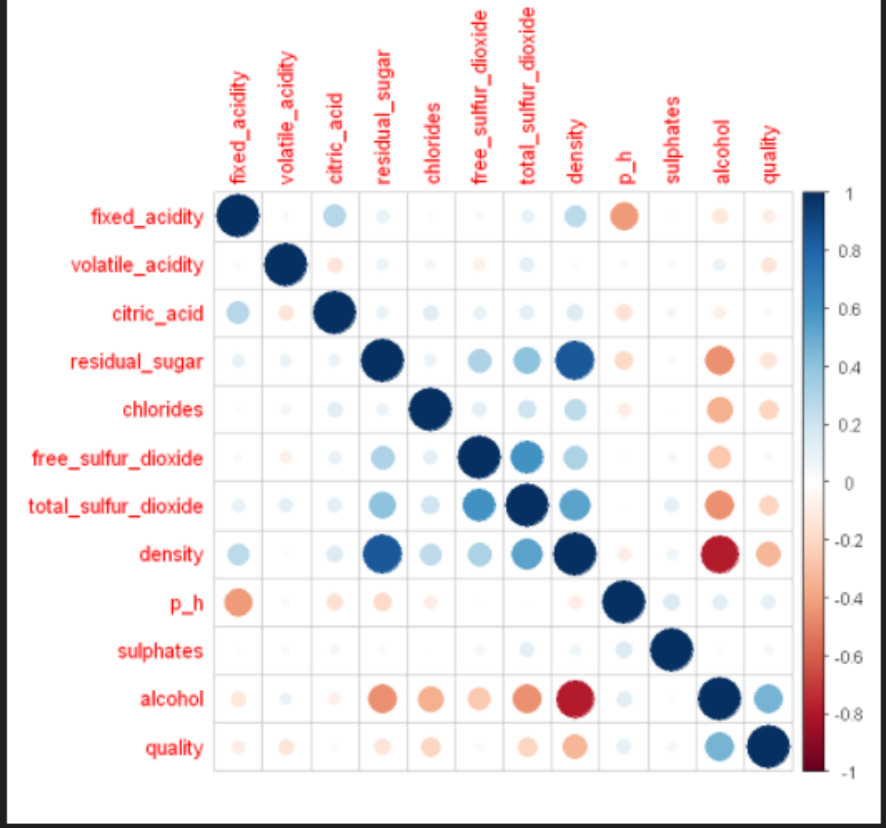
```
0
```

there are no null values in this dataset

There are not any missing values that we have to deal with in this dataset.

```
#Correlation Heatmap of Variables
corrplot(cor(df[1:12]))
```

✓ 0.3s



By using this figure, we can see alcohol level is the most correlated feature for the Quality.

Outlier Removal and Feature Scaling - Preprocessing

Outlier Removal

Outlier is a silent killer, an observation that is numerically distant from the rest of the data or in a simple word it is the value which is out of the range.

Reasons for outliers,
* Data Entry Errors: - Human errors such as errors caused during data collection, recording, or entry can cause outliers in data.
* Data Entry Errors: - Human errors such as errors caused during data collection, recording, or entry can cause outliers in data.
* Measurement Error: - It is the most common source of outliers. This is caused when the measurement instrument used turns out to be faulty.
* Natural Outlier: - When an outlier is not artificial (due to error), it is a natural outlier. Most of real-world data belong to this category.

For detect outliers we can use different techniques such as Hypothesis Testing, Z-score methos, Isolation Forest and Visualizing the Data. In this Project We use Z-score Method.

IQR method

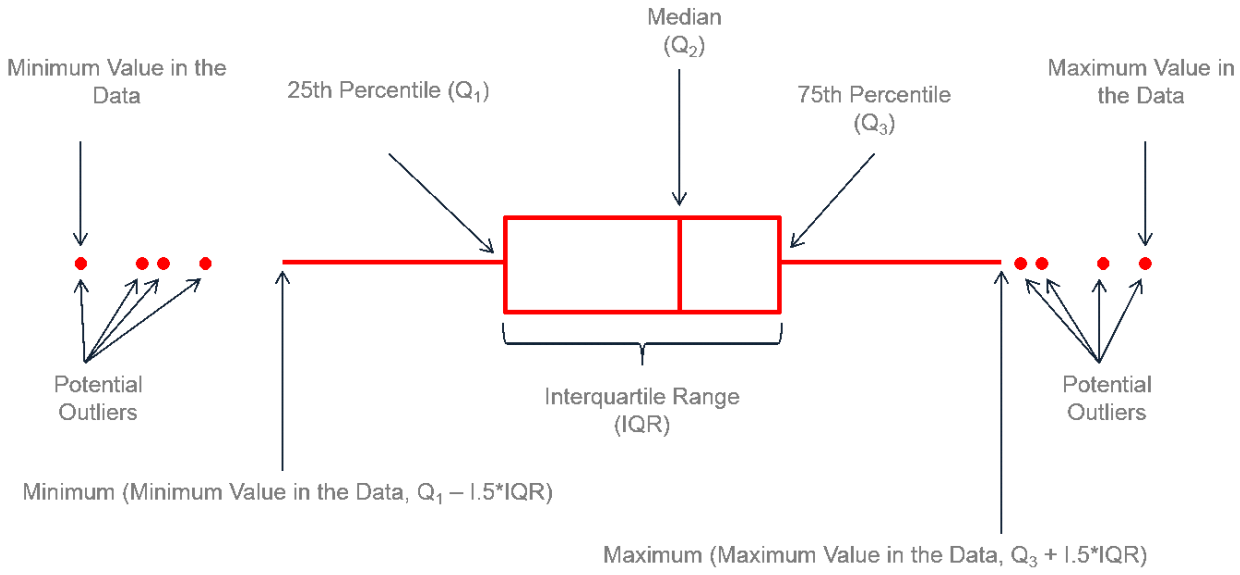


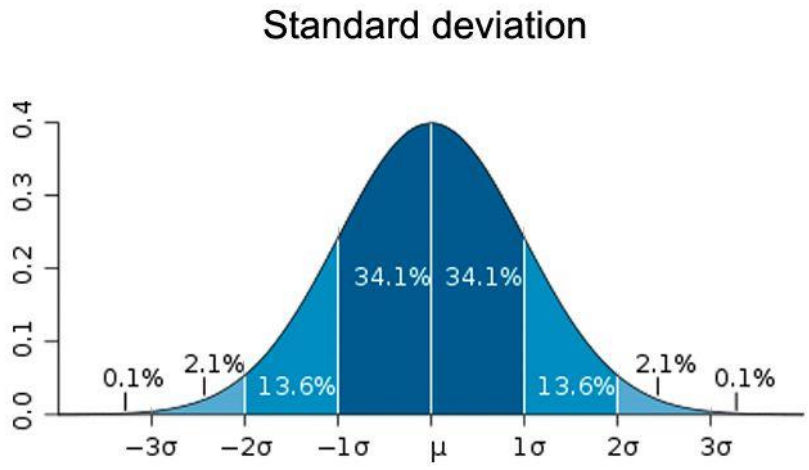
Figure 1 : IQR method

Any Value which is -1.5*IQR > x or 1.5*IQR > x treated as outliers.

* Q1 represents the 1st quartile/25th percentile of the data.
* Q2 represents the 2nd quartile/median/50th percentile of the data.
* Q3 represents the 3rd quartile/75th percentile of the data.
* (Q1–1.5*IQR) represent the smallest value in the data set and (Q3+1.5*IQR) represent the largest value in the data set.

Z-Score Method

Using this method, we will be able to find how many Slandered Deviations value away from the mean.



% of values expected to lie in the symmetric interval (−zσ, zσ)

1σ	68.2689492%
2σ	95.4499736%
3σ	99.7300204%
4σ	99.993666%
5σ	99.9999426697%

Figure shows area under normal curve and how much area that standard deviation covers.
* 68% of the data points lie between + or - 1 standard deviation.
* 95% of the data points lie between + or - 2 standard deviation
* 99.7% of the data points lie between + or - 3 standard deviation

By experimenting above mentioned IQR method I experienced most of the data points are eliminated with IQR method. Because of that reason we are using Z-score method to Scale and Remove Outliers in this Dataset.

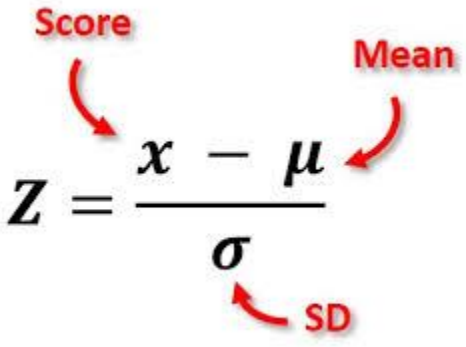


Figure 2 : z-score formula

```
z_score = function(x) {
  return((x - mean(x)) / sd(x))
}
```
✓ 0.1s

```
dfNorm <- as.data.frame(lapply(df[1:11], z_score))
dfNorm$index <- 1:nrow(dfNorm)
head(dfNorm)
```
✓ 0.1s

A data.frame: 6 × 12

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	p_h	sulphates	alcohol
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1.5238976	-0.04648274	0.6271255	-0.9831716	-0.58570079	-1.5272358	-1.8230268	-1.0701698	-1.31955843	0.61051063	1.1935810
2	2.1293483	-0.46690270	0.5433238	-0.4429502	-0.49277626	-1.1554516	-0.7152461	0.2268701	-0.32155169	0.34731453	-0.6685289
3	1.2817173	-0.99242766	0.2919184	-1.0322826	-0.26046491	-1.2174157	-1.5340406	-0.6710806	-0.05541657	1.22463485	0.2220454
4	1.7660778	1.53009213	2.3869628	2.5135339	-0.26046491	0.3316852	0.8019318	2.0560288	-1.38609221	1.57556298	-0.6685289
5	-0.4135447	0.37393723	-1.6355225	0.2053154	-0.07461584	-0.1020631	-0.1372736	0.4929295	0.21071856	0.08411844	-0.8304515
6	0.9184469	4.15771691	-1.6355225	-0.9733494	1.31925222	-0.6597394	0.7056030	-0.1057043	-0.92035573	0.17185047	-0.9923741

Figure 3 : applying z-score normalization

```
summary(dfNorm[1:11])
```
[1021] ✓ 0.1s

```
 fixed_acidity      volatile_acidity   citric_acid      residual_sugar
 Min.   :-3.68298   Min.   :-2.0435   Min.   :-2.8087   Min.   :-1.1501
 1st Qu.:-0.65573   1st Qu.:-0.6771   1st Qu.:-0.5461   1st Qu.:-0.9341
 Median :-0.05027   Median :-0.1516   Median :-0.1271   Median :-0.2269
 Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.: 0.55518   3rd Qu.: 0.4790   3rd Qu.: 0.4595   3rd Qu.: 0.6964
 Max.   : 8.91040   Max.   : 7.2583   Max.   :11.1023   Max.   :11.6580
   chlorides       free_sulfur_dioxide total_sulfur_dioxide    density
 Min.   :-1.7008   Min.   :-2.0849     Min.   :-3.1235      Min.   :-2.2974
 1st Qu.:-0.4463   1st Qu.:-0.7217     1st Qu.:-0.7152      1st Qu.:-0.7709
 Median :-0.1211   Median :-0.1021     Median :-0.1132      Median :-0.1057
 Mean   : 0.0000   Mean   : 0.0000     Mean   : 0.0000      Mean   : 0.0000
 3rd Qu.: 0.2042   3rd Qu.: 0.6415     3rd Qu.: 0.6815      3rd Qu.: 0.6925
 Max.   :13.9570   Max.   : 5.9084     Max.   : 4.9441      Max.   :14.9533
      p_h             sulphates          alcohol
 Min.   :-3.11597   Min.   :-2.37238   Min.   :-2.0449
 1st Qu.:-0.65422   1st Qu.:-0.70547   1st Qu.:-0.8305
 Median :-0.05542   Median :-0.09135   Median :-0.1018
 Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
 3rd Qu.: 0.60992   3rd Qu.: 0.52278   3rd Qu.: 0.7078
 Max.   : 4.20275   Max.   : 5.17258   Max.   : 2.9747
```

Figure 4 : summary of normalized data frame

Eliminating all outliers using z-score,

```
#only keep rows in dataframe with all z-scores less than absolute value of 3
no_outliers <- dfNorm[!rowSums(dfNorm[1:11]>3),]
```

✓ 0.5s

Figure 5 : eliminating outliers

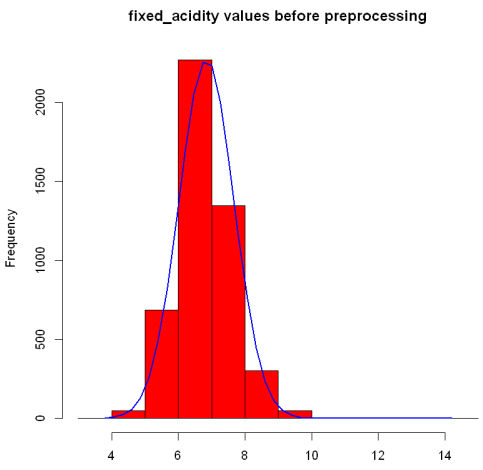
Below figure shows a summary of final dataset after scaling and outlier removal.

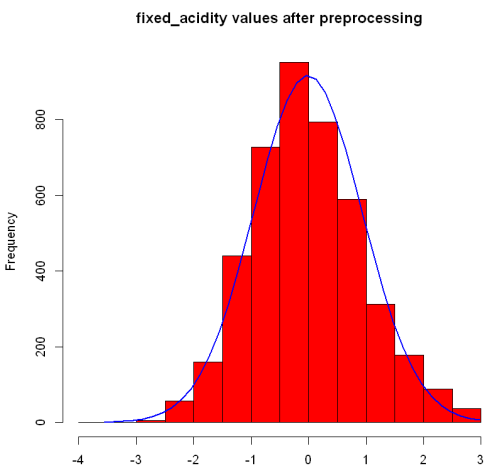
```
summary(df_final[2:12])
```

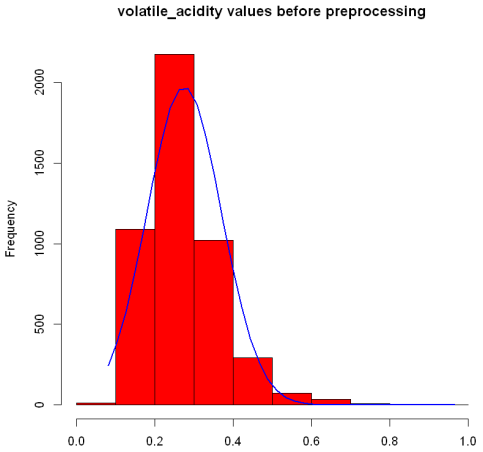
[1170] ✓ 0.2s

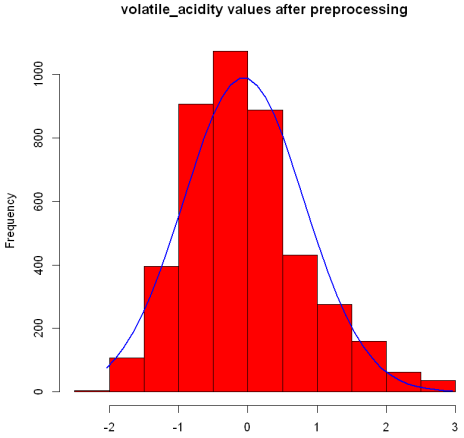
```
   fixed_acidity      volatile_acidity    citric_acid       residual_sugar
Min.   :-3.56189   Min.   :-2.04348   Min.   :-2.80875   Min.   :-1.1501491
1st Qu.:-0.65573   1st Qu.:-0.67711   1st Qu.:-0.54610   1st Qu.:-0.9144161
Median :-0.05027   Median :-0.15159   Median :-0.21089   Median :-0.2072173
Mean   :-0.01286   Mean   :-0.05957   Mean   :-0.06593   Mean   : 0.0007457
3rd Qu.: 0.55518   3rd Qu.: 0.47904   3rd Qu.: 0.37572   3rd Qu.: 0.6964257
Max.   : 2.97698   Max.   : 2.94901   Max.   : 2.97358   Max.   : 2.8180223
  chlorides       free_sulfur_dioxide total_sulfur_dioxide    density
Min.   :-1.5614   Min.   :-2.08491    Min.   :-2.83448     Min.   :-2.29737
1st Qu.:-0.4928   1st Qu.:-0.72170    1st Qu.:-0.73933     1st Qu.:-0.78748
Median :-0.1675   Median :-0.10206    Median :-0.11319     Median :-0.13231
Mean   :-0.1192   Mean   :-0.02871    Mean   :-0.01793     Mean   :-0.02102
3rd Qu.: 0.2042   3rd Qu.: 0.57954    3rd Qu.: 0.65744     3rd Qu.: 0.69247
Max.   : 2.9919   Max.   : 2.93417    Max.   : 2.92116     Max.   : 2.64136
     p_h              sulphates           alcohol
Min.   :-2.650234   Min.   :-2.37238   Min.   :-1.72103
1st Qu.:-0.654221   1st Qu.:-0.70547   1st Qu.:-0.83045
Median :-0.055417   Median :-0.17908   Median :-0.10180
Mean   :-0.001019   Mean   :-0.03857   Mean   : 0.02107
3rd Qu.: 0.609921   3rd Qu.: 0.43505   3rd Qu.: 0.70781
Max.   : 2.938604   Max.   : 2.97928   Max.   : 2.97473
```

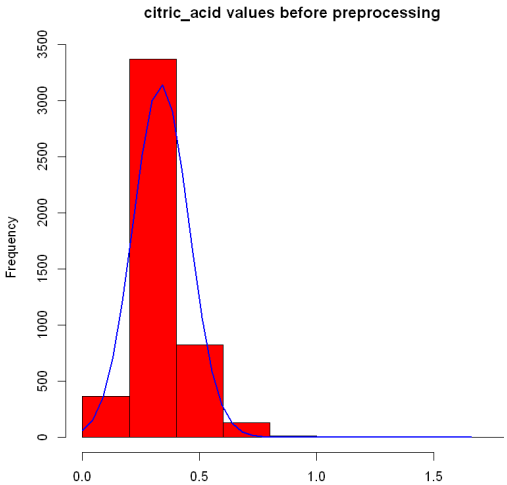
Figure 6 : summary of scaled and outlier removed data frame

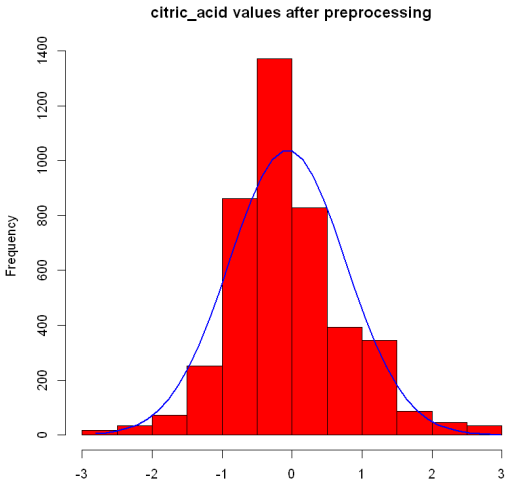

fixed_acidity values before preprocessing
Frequency
0
500
1000
1500
2000
4
6
8
10
12
14

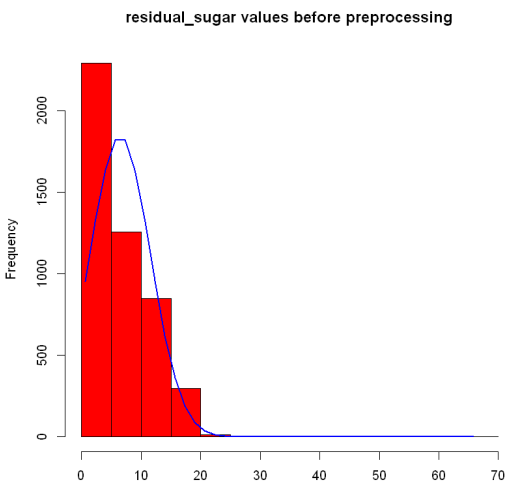

fixed_acidity values after preprocessing
Frequency
0
200
400
600
800
-4
-3
-2
-1
0
1
2
3

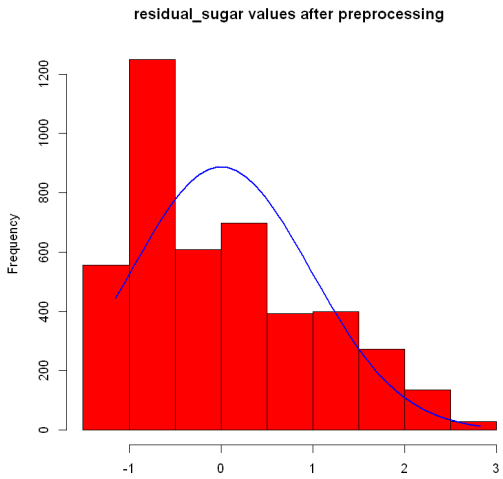

volatile_acidity values before preprocessing
Frequency
0
500
1000
1500
2000
0.0
0.2
0.4
0.6
0.8
1.0

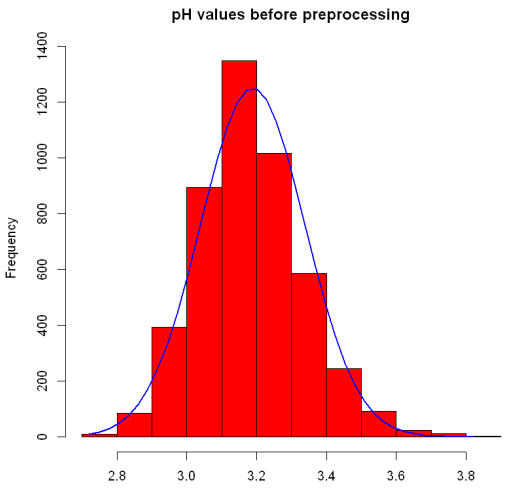

volatile_acidity values after preprocessing
Frequency
0
200
400
600
800
1000
-2
-1
0
1
2
3

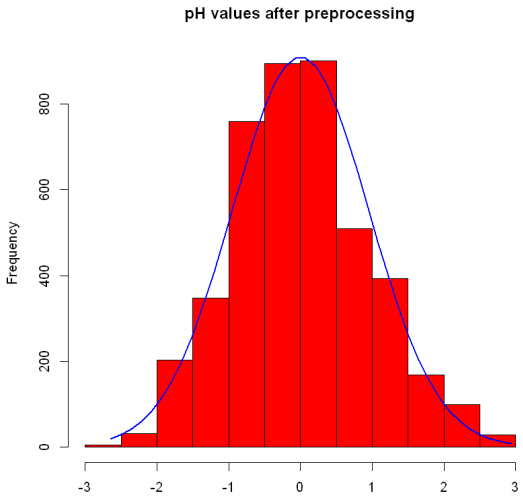

citric_acid values before preprocessing
Frequency
0
500
1000
1500
2000
2500
3000
3500
0.0
0.5
1.0
1.5

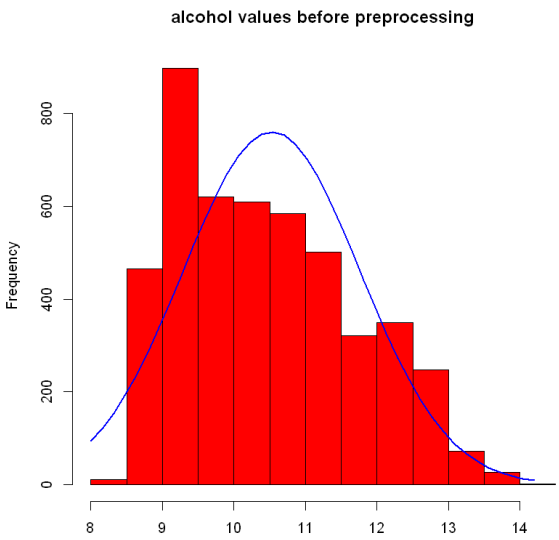

citric_acid values after preprocessing
Frequency
0
200
400
600
800
1000
1200
1400
-3
-2
-1
0
1
2
3

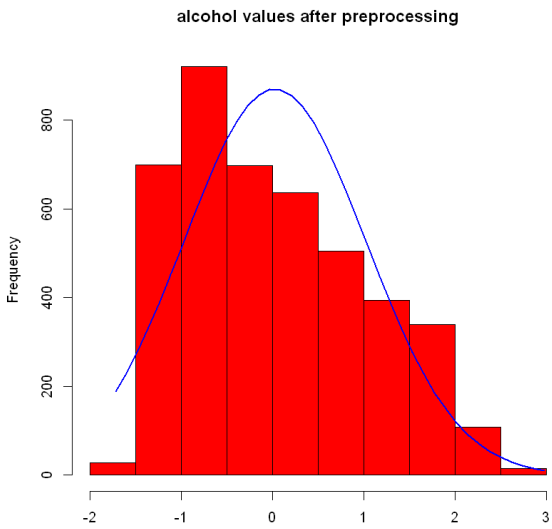




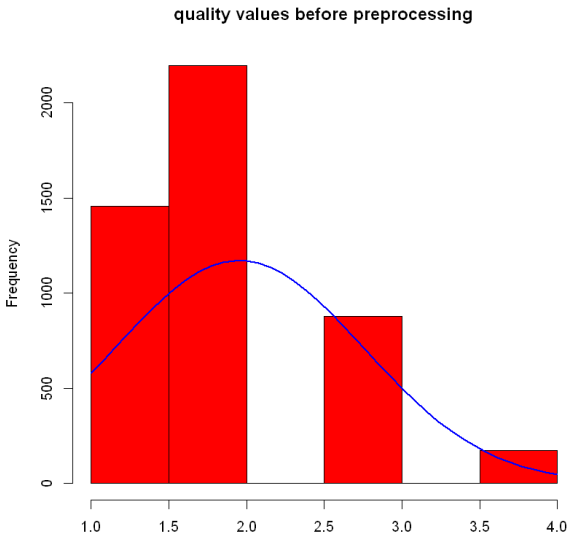


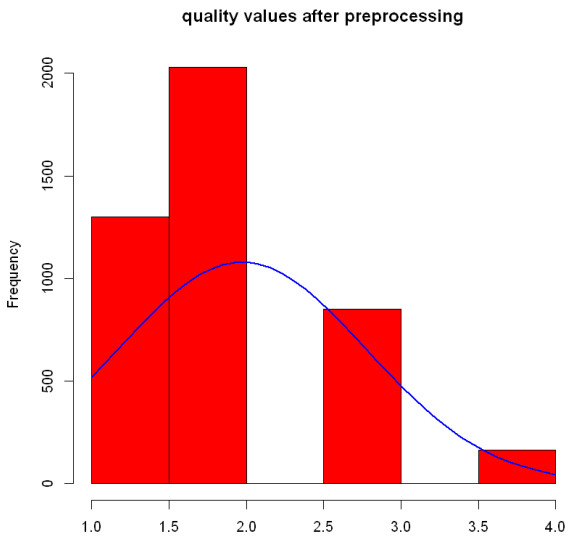






From below figure we will be able to figure it out it doesn't affect to Quality feature distribution to Scale and the Outlier removal task.





Scaling and outlier removal is very important task when doing ML use case, it will be helpful to increase performance and the accuracy of the Machine Learning Model.

Define the number of cluster centers

Manual Method

This red wine dataset originally consisted with 1-10 range qualities but in this dataset, we will be able to detect 4 qualities which are 5,6,7 and 8. So there will be four clusters in this data frame and further clarification we must process with different methods and techniques to evaluate number of clusters in this red wine dataset.

Automated Methods

1.NBclust

NbClust

```
# see the datasrt types
str(df_final)
```

✓ 0.7s

```
'data.frame':   4342 obs. of  13 variables:
 $ index               : int  1 2 3 4 5 7 8 9 10 11 ...
 $ fixed_acidity       : num  1.524 2.129 1.282 1.766 -0.414 ...
 $ volatile_acidity    : num  -0.0465 -0.4669 -0.9924 1.5301 0.3739 ...
 $ citric_acid         : num  0.627 0.543 0.292 2.387 -1.636 ...
 $ residual_sugar      : num  -0.983 -0.443 -1.032 2.514 0.205 ...
 $ chlorides           : num  -0.5857 -0.4928 -0.2605 -0.2605 -0.0746 ...
 $ free_sulfur_dioxide : num  -1.527 -1.155 -1.217 0.332 -0.102 ...
 $ total_sulfur_dioxide: num  -1.823 -0.715 -1.534 0.802 -0.137 ...
 $ density             : num  -1.07 0.227 -0.671 2.056 0.493 ...
 $ p_h                 : num  -1.3196 -0.3216 -0.0554 -1.3861 0.2107 ...
 $ sulphates           : num  0.6105 0.3473 1.2246 1.5756 0.0841 ...
 $ alcohol             : num  1.194 -0.669 0.222 -0.669 -0.83 ...
 $ quality             : num  5 5 5 5 5 5 5 5 5 5 ...
```

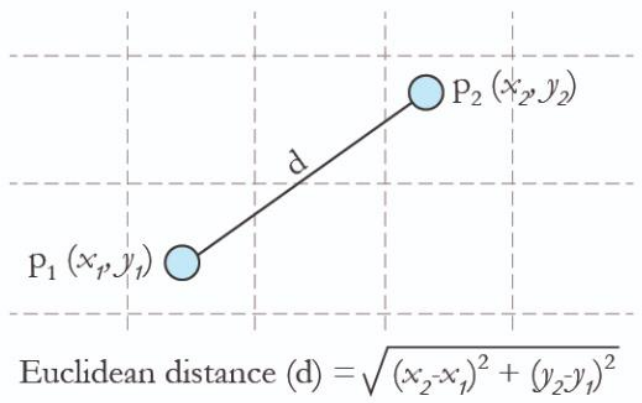
```
df_nb <- df_final[2:12]
```

✓ 0.1s

```
#setting seed point
set.seed(26)
```

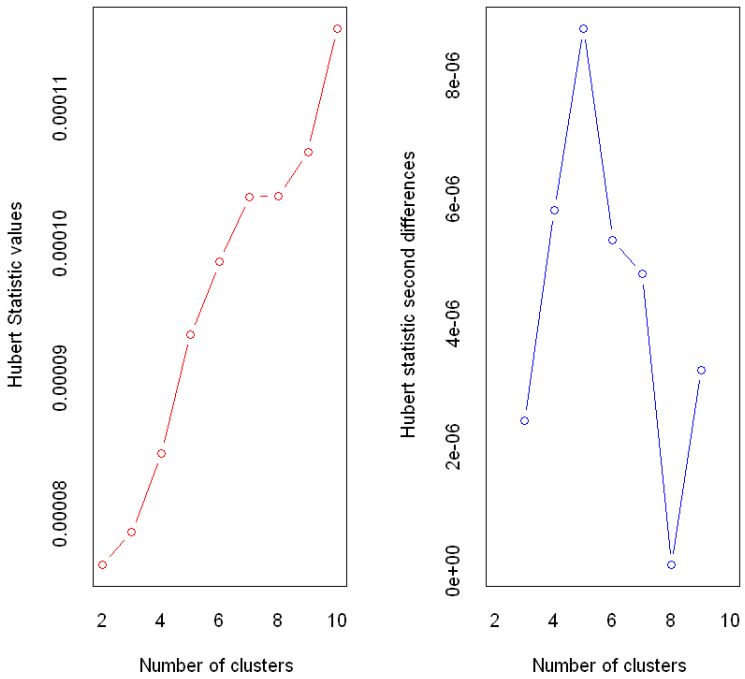
✓ 0.4s

i. Euclidian Distance




```
#euclidient distance
no_of_clusters_eud = NbClust(df_nb,distance="euclidean", min.nc=2,max.nc=10,method="kmeans",index="all")
```
✓ 7m 36.4s

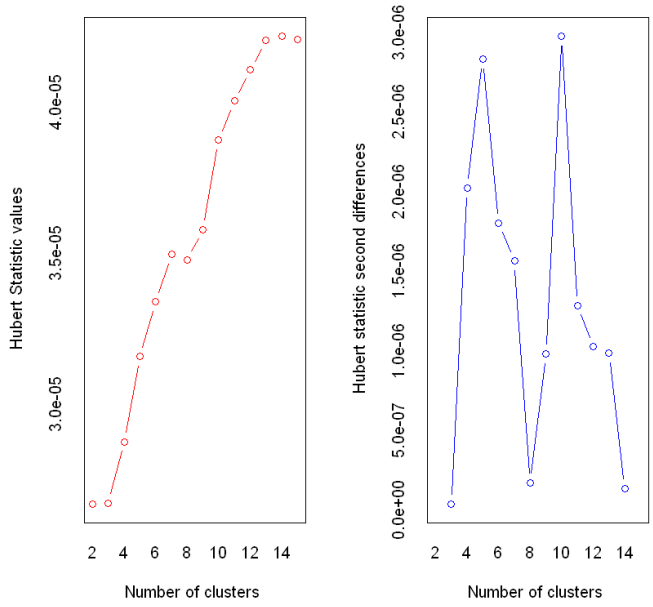
```
*** : The Hubert index is a graphical method of determining the number of clusters.
              In the plot of Hubert index, we seek a significant knee that corresponds to a
              significant increase of the value of the measure i.e the significant peak in Hubert
              index second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

*******************************************************************
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 9 proposed 3 as the best number of clusters
* 2 proposed 5 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


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

ii. Manhattan Distance

```
#manhatten
no_of_clusters_man = NbClust(df_nb,distance="manhattan", min.nc=2,max.nc=15,method="kmeans",index="all")
✓ 12m 32.6s

*** : 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 Hubert
                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 Dindex
                second differences plot) that corresponds to a significant increase of the value of
                the measure.

*****************************************************************
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 10 proposed 3 as the best number of clusters
* 2 proposed 5 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


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

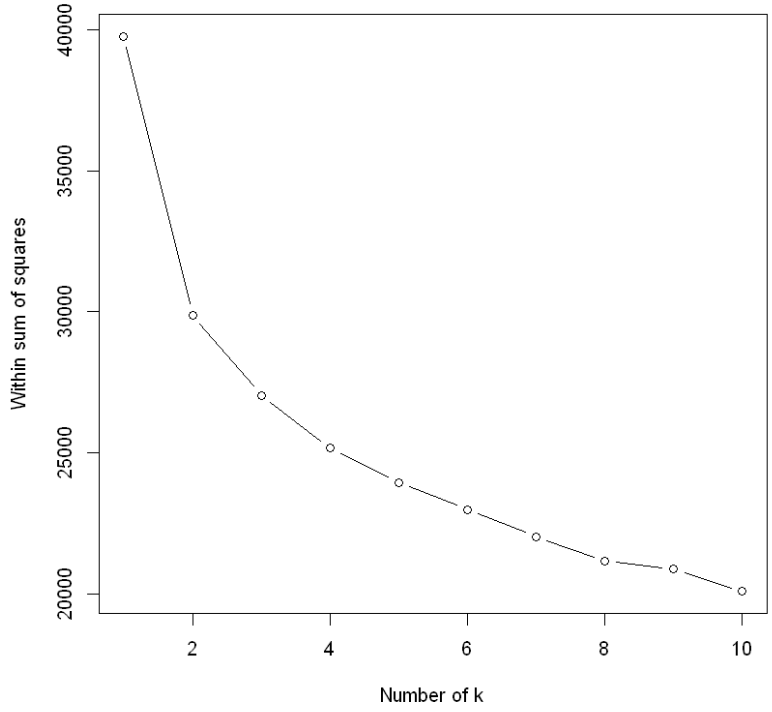
2.Elbow Method

```
[132]
k = 1:10
```
✓ 0.1s

```
[133]
set.seed(25)
WSS = sapply(k, function(k) {kmeans(df_final[2:12], centers=k)$tot.withinss})
```
✓ 0.5s

```
[134]
# You can then use a line plot to plot the within sum of squares with a different number of k
plot(k, WSS, type="b", xlab= "Number of k", ylab="Within sum of squares")
```
✓ 0.1s

In below figure k value is initializing 1 to 10 and it will calculate the “WSS” (With in cluster Sum of Square) and return. We need to plot this output to get a better idea about that. This elbow point is the optimal cluster value of k. in below figure it is k=4

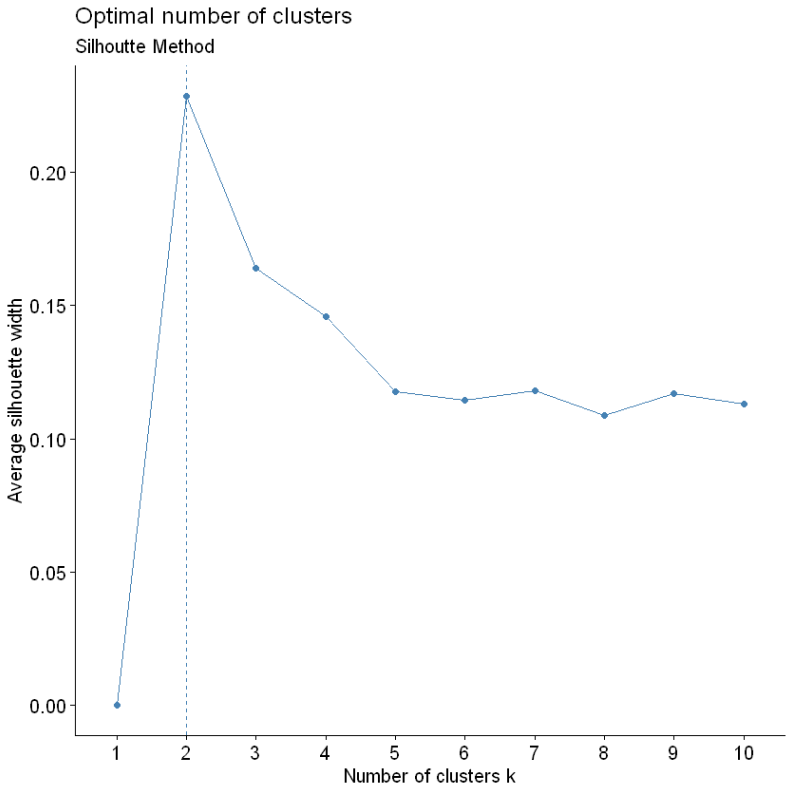


3.Silhouette Method

This will calculate average Silhouette with each k value and display the optimal solution.

```
fviz_nbclust(df_final[2:12], kmeans, method = "silhouette") +
labs(subtitle = "Silhoutte Method")
```

✓ 1.2s



Above figure silhouette method returned optimal number of clusters as 2.

As above mentioned, automated tools most of the tools suggested k=2 for optimal clustering and elbow method suggest k=4, from this idea and as form the instructions given in the problem we will perform k-means clustering from k=2, k=3 and k=4 in next chapter and evaluate each clusters using BSS/TSS and confusion matrix.

K-means Clustering

From above chapter we discussed some automated and manual methods to select optimal cluster amount for k-means algorithm, in this chapter we will experimenting k=2, k=3 and k=4 as number of clusters and evaluate which is the “winning” clustering approach.

To evaluate this process, we use WSS (Between cluster Sums of Square) over TSS (Total cluster Sums of Square) and Confusion Metrix calculations such as accuracy/precision and recall.

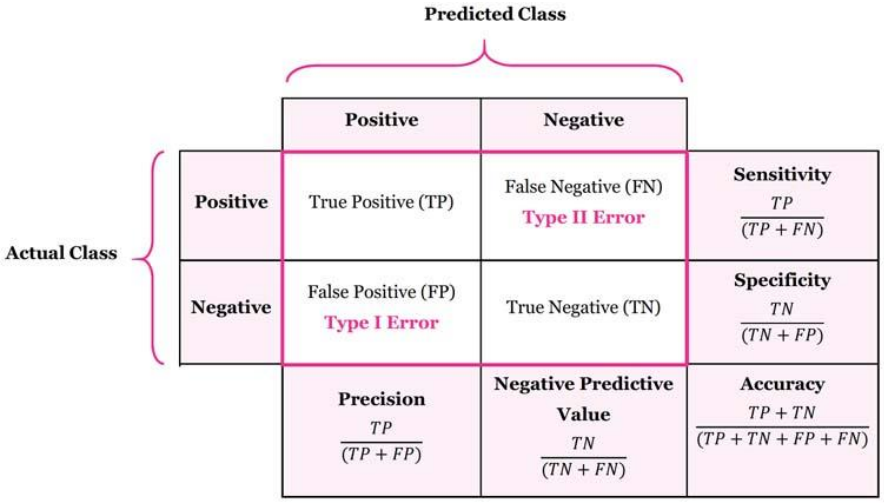


Figure 7 : confusion matrix

K=2

```
x=df_final[2:12]
y=df_final$quality
```
✓ 0.8s

For further analysis we will divide data frame into x and y, x: all numerical features / y: quality.

```
kc_2 <- kmeans(x,centers=2)
```
✓ 0.3s

```
kc_2
```
✓ 0.4s

Above figure shows executing k-means inbuilt function in r with data and number of clusters. Then we will see the results.

```
Cluster means:
  fixed_acidity volatile_acidity citric_acid residual_sugar  chlorides
1    -0.1155064      -0.08696850 -0.16850322     -0.5742046 -0.2830937
2     0.1496217      -0.01618924  0.09644741      0.9108840  0.1401720
  free_sulfur_dioxide total_sulfur_dioxide    density        p_h   sulphates
1          -0.4098224           -0.5150499 -0.6461456  0.1060547 -0.09263144
2           0.5745835            0.7689961  0.9685464 -0.1705149  0.04701108
    alcohol
1  0.5318484
2 -0.7874905

Clustering vector:
   [1] 1 1 1 2 2 2 1 1 2 2 2 2 1 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2
  [38] 1 2 2 1 1 2 2 2 2 2 1 2 1 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 1 2 2 2
  [75] 2 2 1 2 2 2 2 1 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 1 1 1 1 1 2 1 1 1 2 1 1 2 1
 [112] 2 1 2 2 1 2 1 1 1 1 2 2 1 2 2 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1
 [149] 2 1 1 1 1 2 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 1 2 1 1 2 2 2 2 2 2 2 2 2 1 1 2
 [186] 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 1 2 2 2 2 2 2 2 1 2 1 2 2 2 2 1 1 2 1
 [223] 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1 1 1 1 1 2 1 2 2 2 2 1 2 2 1 1 2 2 1
 [260] 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 1 1 1 1 1 1 1 2 1 1 2 1 2 1 2 1 1 1 1 2 1 1
 [297] 1 1 2 1 1 1 2 2 2 2 2 2 2 1 2 2 2 1 1 2 1 1 2 1 2 1 2 1 1 2 1 2 2 1 1 2 1
 [334] 2 2 2 2 2 1 1 1 1 2 1 1 2 1 1 2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 2 1 2 1
 [371] 1 2 2 1 2 2 2 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1
 [408] 1 1 2 1 1 2 1 1 1 2 1 1 1 2 1 2 1 2 1 1 2 2 1 2 1 2 2 2 2 2 2 2 2 1 1 1 1
 [445] 1 2 1 1 2 2 2 2 2 2 1 2 2 1 1 2 1 1 2 1 2 1 1 2 1 2 1 2 2 2 2 1 1 1 2 2 2
 [482] 1 2 2 2 2 2 1 1 2 2 1 2 2 2 2 2 1 1 1 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 2 2 2
 [519] 2 2 2 1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 2 2 2 2 1 1
 [556] 1 2 2 1 2 1 1 1 1 1 2 2 1 1 1 1 1 2 2 2 2 2 1 2 1 2 2 1 1 1 2 2 2 2 1 2 1
 [593] 2 2 2 2 2 2 1 1 2 2 1 2 2 2 2 2 2 2 2 2 1 1 1 2 1 2 2 2 1 2 1 2 2 2 1 1 2 1
 [630] 1 1 2 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2
 [667] 1 2 2 2 1 2 1 2 1 2 2 1 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
 [704] 1 1 1 2 1 2 2 2 1 1 2 2 2 2 2 1 2 2 1 2 2 2 1 2 2 2 1 2 1 2 2 1 2 1 2 1 1
 [741] 1 2 1 2 2 2 1 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 1 1 2 2 2 2 2 1 1 2 1 2 2
 [778] 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 2 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 2 1 2 2 2
 [815] 1 1 1 2 1 1 1 2 2 1 1 1 1 2 1 2 2 1 2 2 1 2 2 2 2 1 2 1 1 2 1 2 1 1 1 1 1
 [852] 1 1 1 1 2 2 1 1 2 1 1 1 2 2 2 2 1 2 1 2 1 1 2 1 2 1 1 1 1 2 1 2 1 1 2 2 1
 [889] 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 1 2 1 2 2 2 2 2 1 1 2 1 1 1 1 2 2
 [926] 2 1 1 2 2 2 2 1 2 2 2 1 1 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 2 1 1 2 1 1 2 2 2
 [963] 2 1 2 1 1 1 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2
[1000] 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1 1 1 1 1 2 1 1 1 1 2 2 1 2 1 1 1 2 1 2 1 2 2
[1037] 1 2 2 2 2 2 2 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 2 1 1 2 1 2 2 1 1 2 2 2 1
[1074] 1 1 1 1 2 1 1 2 1 1 1 2 2 1 1 1 2 2 2 2 1 1 1 1 2 2 1 2 1 1 1 2 2 1 1 2 1
[1111] 1 2 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2 1 1 1 2 2 1 2 2 2 2 2 2 2 1 2 2 2 1 1 1
[1148] 1 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 2 2 1 2 2 2 2 1 1 2 1 2 2 2 1 1 1 1 1 1 2
[1185] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 1 1 2 1 2 2 2 1 2 2 1 1 1 2
[1222] 2 1 2 1 1 1 2 1 1 2 2 2 2 2 1 1 1 1 2 1 2 1 2 1 1 1 1 2 2 2 1 1 1 1 2 1 2
```

```
121   [3923] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
122   [3960] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1
123   [3997] 1 1 1 1 1 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 1 1 1 1
124   [4034] 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1
125   [4071] 2 1 1 2 2 2 2 2 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
126   [4108] 1 1 1 1 1 1 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 2 2 1 1 1 1
127   [4145] 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
128   [4182] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1
129   [4219] 1 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 1 1 1 1 2 2 2 2
130   [4256] 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1
131   [4293] 1 1 1 1 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 2
132   [4330] 2 2 2 2 2 2 1 2 1 1 1 1 1
133
134   Within cluster sum of squares by cluster:
135   [1] 18741.16 11131.07
136    (between_SS / total_SS =  24.9 %)
137
138   Available components:
139
140   [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
141   [6] "betweenss"    "size"         "iter"         "ifault"
```

```
str(kc_2)
✓ 0.6s

List of 9
 $ cluster     : int [1:4342] 1 1 1 2 2 2 1 1 2 2 ...
 $ centers     : num [1:2, 1:11] -0.1155 0.1496 -0.087 -0.0162 -0.1685 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "1" "2"
  .. ..$ : chr [1:11] "fixed_acidity" "volatile_acidity" "citric_acid" "residual_sugar" ...
 $ totss       : num 39754
 $ withinss    : num [1:2] 18741 11131
 $ tot.withinss: num 29872
 $ betweenss   : num 9882
 $ size        : int [1:2] 2661 1681
 $ iter        : int 1
 $ ifault      : int 0
 - attr(*, "class")= chr "kmeans"
```

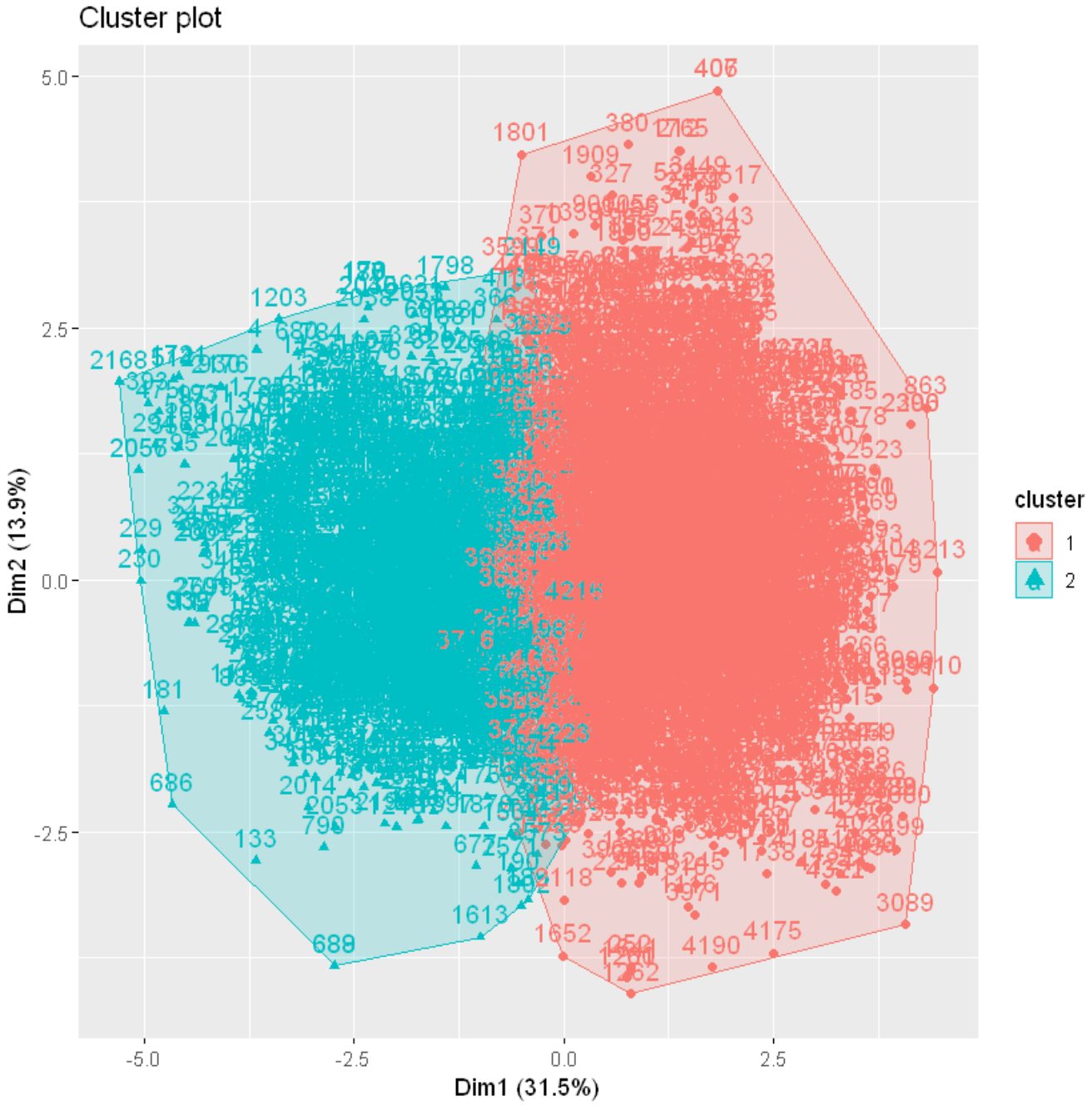
Then we will be able to create confusion matrix, below figure shows the confusion matrix,

```
# implement the metrix
confusionMatrix(

 factor(kc_2$cluster, levels = 1:4),

 factor(df_final$quality,levels=1:4)
)
```

✓ 0.3s

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4
         1  728  774  153   26
         2  571 1256  698  136
         3    0    0    0    0
         4    0    0    0    0

Overall Statistics

               Accuracy : 0.4569
                 95% CI : (0.442, 0.4719)
    No Information Rate : 0.4675
    P-Value [Acc > NIR] : 0.9214

                  Kappa : 0.0913

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.5604   0.6187    0.000  0.00000
Specificity            0.6868   0.3923    1.000  1.00000
Pos Pred Value         0.4331   0.4720      NaN      NaN
Neg Pred Value         0.7854   0.5396    0.804  0.96269
Prevalence             0.2992   0.4675    0.196  0.03731
Detection Rate         0.1677   0.2893    0.000  0.00000
Detection Prevalence   0.3871   0.6129    0.000  0.00000
Balanced Accuracy      0.6236   0.5055    0.500  0.50000
```

Then we visualize this result with the help of visualization libraries.



K=3

```
kc_3 <- kmeans(x,centers=3)
✓ 0.3s

kc_3
✓ 0.3s
```

```
[3405] 1 3 3 3 1 2 3 3 2 3 3 2 3 1 3 3 3 1 1 1 1 1 3 3 1 1 2 2 2 2 1 1 1 1 1
[3442] 1 1 1 1 1 3 1 3 3 3 2 2 1 2 2 1 1 1 2 1 3 3 3 3 2 2 1 2 1 1 2 2 2 3 3 1 1
[3479] 1 1 3 1 1 1 3 2 2 3 2 1 3 2 1 1 1 3 1 1 1 1 1 1 1 3 3 3 1 1 1 3 3 1 1 3 3
[3516] 3 3 3 3 1 1 1 3 3 1 1 1 1 1 1 1 1 1 3 2 2 3 1 2 1 2 3 1 1 3 3 3 3 3 1 3 3
[3553] 2 3 1 2 2 2 2 3 3 3 3 3 3 3 3 2 3 3 1 1 1 1 1 3 3 1 1 1 1 3 3 3 2 2 2 2 3
[3590] 3 3 2 3 2 3 3 3 1 3 3 3 3 2 1 1 2 1 3 3 1 1 2 3 3 3 3 3 2 2 2 2 3 3 3 3 1
[3627] 1 2 2 2 2 2 2 1 1 1 1 1 3 1 3 1 3 3 1 3 2 2 2 2 2 1 1 1 1 2 3 3 1 3 3 3 2
[3664] 3 1 3 2 1 1 3 3 1 3 2 1 3 3 3 1 1 2 2 2 2 2 2 2 1 2 1 3 1 3 3 2 2 3 1 1 1
[3701] 3 1 3 1 2 1 1 1 1 1 3 1 1 3 1 1 3 1 2 1 1 1 1 1 3 3 1 1 3 1 1 1 1 1 3 1 2
[3738] 2 2 2 2 2 2 1 1 1 1 1 2 2 2 1 1 2 1 1 1 1 3 1 2 1 1 1 1 1 2 3 3 3 1 3 3 1
[3775] 1 3 3 1 2 1 3 3 1 1 3 1 1 3 1 1 1 1 1 3 3 1 3 3 1 1 1 1 3 3 3 3 3 3 1 1 1
[3812] 3 3 3 1 3 2 3 3 3 3 3 3 3 3 3 1 1 3 3 3 1 3 3 3 1 3 1 1 1 2 3 3 1 1 3 3 1 1
[3849] 3 3 3 1 1 1 1 3 1 3 3 3 3 3 1 1 1 1 3 3 3 1 1 3 3 1 1 3 1 1 3 1 3 3 1 1 2
[3886] 1 3 1 1 1 3 3 3 3 3 1 1 3 1 3 1 1 1 3 1 1 1 3 1 1 2 1 1 3 3 3 3 1 3 1 3 3
[3923] 3 1 1 1 3 1 3 3 1 1 1 3 1 3 3 3 1 1 1 1 1 1 3 1 1 3 1 3 1 3 3 1 3 1 3 3 3
[3960] 3 3 3 3 1 1 1 1 3 3 3 1 1 3 2 2 3 2 1 3 1 1 3 3 3 3 1 2 1 3 1 1 3 1 3 1 3
[3997] 3 1 1 1 1 1 1 2 2 2 2 2 2 2 2 1 1 1 1 1 3 3 1 3 1 1 1 1 3 1 1 1 3 1 1 1 1
[4034] 1 3 3 1 3 1 2 1 3 1 2 1 3 3 1 1 1 1 3 1 2 2 2 2 2 2 2 2 2 1 1 3 3 3 1 3 3
[4071] 2 3 1 2 2 2 2 2 2 3 1 1 2 3 1 1 1 1 1 2 1 1 1 1 1 1 1 1 3 3 1 3 3 3 1 1 1
[4108] 3 1 3 1 1 1 1 3 1 1 1 1 1 1 3 1 1 1 3 1 3 2 1 1 1 1 1 1 3 2 2 2 2 1 1 1 1
[4145] 1 1 1 2 2 3 3 1 3 1 3 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
[4182] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 3 3 3 3 1 1 3 2 1 2 3 1 2 1 3
[4219] 1 1 3 1 1 1 3 3 3 3 3 3 3 3 3 1 3 1 3 1 3 1 3 3 3 1 1 1 1 1 1 1 2 2 2 2
[4256] 2 2 1 1 3 3 3 1 1 3 1 3 3 3 3 1 3 1 1 3 2 2 2 1 1 3 3 3 3 3 3 3 3 3 3 3 3
[4293] 1 1 1 1 1 1 1 3 1 3 1 1 3 1 3 1 1 3 3 2 1 1 1 1 1 3 3 3 1 1 1 1 3 3 1 3 2
[4330] 2 2 2 2 2 2 1 2 1 1 1 1 1

Within cluster sum of squares by cluster:
[1]  8815.095 10162.267  8071.899
 (between_SS / total_SS =  32.0 %)

Available components:

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

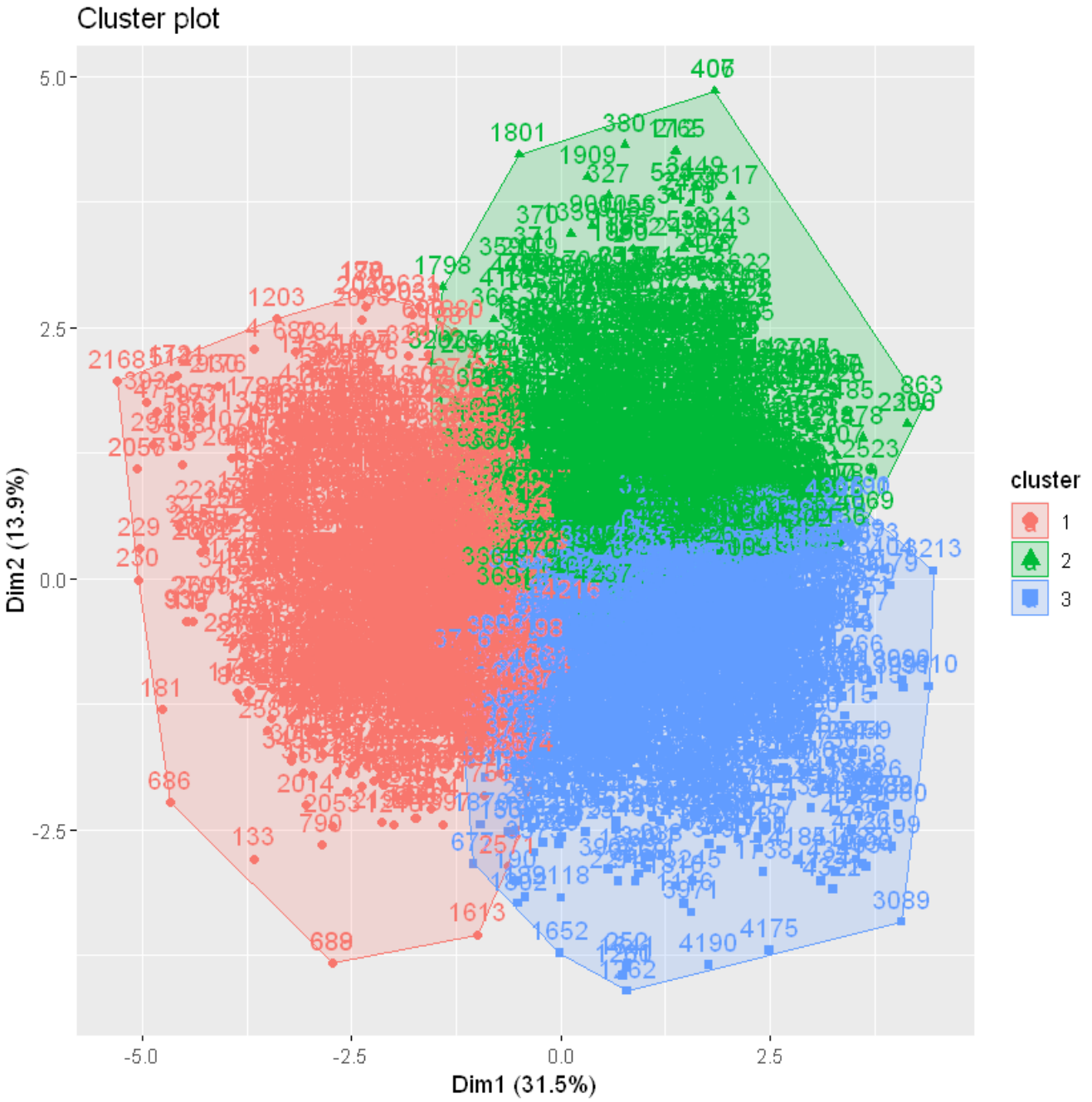
```
str(kc_3)
```
✓ 0.4s

```
List of 9
 $ cluster     : int [1:4342] 3 3 3 2 2 2 3 1 2 2 ...
 $ centers     : num [1:3, 1:11] -0.7516 0.1291 0.6065 -0.1313 -0.0196 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:3] "1" "2" "3"
  .. ..$ : chr [1:11] "fixed_acidity" "volatile_acidity" "citric_acid" "residual_sugar" ...
 $ totss       : num 39754
 $ withinss    : num [1:3] 8815 10162 8072
 $ tot.withinss: num 27049
 $ betweenss   : num 12705
 $ size        : int [1:3] 1422 1588 1332
 $ iter        : int 4
 $ ifault      : int 0
 - attr(*, "class")= chr "kmeans"
```

```
table(y,kc_3$cluster)
```
✓ 0.1s

```
y     1   2   3
  1 254 691 354
  2 669 730 631
  3 420 142 289
  4  79  25  58
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4
         1 690 721 141  25
         2 358 637 277  56
         3 251 672 433  81
         4   0   0   0   0

Overall Statistics

               Accuracy : 0.4053
                 95% CI : (0.3907, 0.4201)
    No Information Rate : 0.4675
    P-Value [Acc > NIR] : 1

                  Kappa : 0.13

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

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.5312   0.3138  0.50881  0.00000
Specificity            0.7085   0.7011  0.71240  1.00000
Pos Pred Value         0.4375   0.4797  0.30132      NaN
Neg Pred Value         0.7797   0.5378  0.85611  0.96269
Prevalence             0.2992   0.4675  0.19599  0.03731
Detection Rate         0.1589   0.1467  0.09972  0.00000
Detection Prevalence   0.3632   0.3058  0.33095  0.00000
Balanced Accuracy      0.6198   0.5075  0.61061  0.50000
```


Cluster plot
Dim2 (13.9%)
Dim1 (31.5%)
5.0
2.5
0.0
-2.5
-5.0
-2.5
0.0
2.5
cluster
1
2
3

K=4

```
kc_4 <- kmeans(x,centers=4)
```
✓ 0.2s

```
kc_4
```
✓ 0.3s

```
K-means clustering with 4 clusters of sizes 1058, 1016, 1393, 875

Cluster means:
  fixed_acidity volatile_acidity citric_acid residual_sugar   chlorides
1    -0.4984108       -0.3485354  -0.3139415     -0.5049657 -0.05419528
2     0.8383557       -0.2130369   0.1246448     -0.4656109 -0.21985616
3     0.1458344        0.0215752   0.1354443      1.0885242  0.16006365
4    -0.6667961        0.3388622  -0.3079108     -0.5780140 -0.52564499
  free_sulfur_dioxide total_sulfur_dioxide    density        p_h    sulphates
1         -0.05843053         -0.003376164 -0.2117886  0.7708475  0.340514888
2         -0.56722022         -0.531572036 -0.3896821 -0.6701992 -0.286238531
3          0.64277310          0.822997224  1.0954954 -0.2647616  0.008919556
4         -0.43649179         -0.777887350 -1.1397747  0.2625769 -0.284957683
     alcohol
1 -0.1275141
2  0.2976544
3 -0.8757755
4  1.3073432

Clustering vector:
   [1] 2 2 2 3 1 3 2 1 3 3 3 1 4 1 1 2 3 2 2 1 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3
  [38] 1 3 1 2 4 3 3 3 3 3 4 3 2 3 4 3 3 3 3 3 3 3 3 4 1 3 3 3 3 3 3 3 3 4 3 3 2
  [75] 3 3 2 2 3 3 3 2 3 1 3 3 2 3 3 3 2 3 3 3 3 3 1 1 1 2 2 2 3 4 1 1 3 2 2 3 4
 [112] 3 2 3 3 1 3 2 2 2 2 3 3 2 3 3 4 1 1 4 1 2 3 2 3 3 3 3 3 3 1 3 3 3 3 1 1 2
 [149] 1 2 2 2 4 3 3 1 3 3 3 3 1 3 1 3 3 3 1 3 3 4 4 3 4 4 1 1 3 3 3 3 3 3 3 2 1 3
 [186] 3 3 3 1 1 3 3 3 1 2 1 2 2 1 3 3 1 2 2 3 1 1 3 3 3 3 2 3 4 3 3 3 3 1 1 2 1
 [223] 3 3 3 2 1 3 3 3 3 3 3 3 1 2 3 3 3 3 3 3 1 2 1 1 2 3 2 3 3 3 3 1 3 3 2 1 3 3 1
 [260] 1 1 4 4 1 3 3 3 3 3 3 3 4 3 3 2 2 2 2 2 2 2 2 3 1 1 3 4 3 4 3 2 2 2 2 3 2 1
 [297] 2 2 3 2 2 2 3 3 1 1 3 3 2 2 3 3 3 1 2 3 2 4 3 4 3 1 3 2 2 2 2 2 2 1 1 3 2
 [334] 2 3 3 3 1 4 2 1 3 2 1 3 2 2 3 2 3 3 3 3 2 2 2 1 1 1 3 2 2 3 2 2 2 1 3 1 2
 [371] 2 3 3 2 3 1 2 2 2 2 4 1 2 1 1 2 3 3 3 2 2 2 3 2 2 2 2 3 2 2 2 2 2 2 1 2 2
 [408] 2 2 3 2 2 2 4 4 1 2 2 2 1 2 2 2 1 3 2 1 3 3 2 3 1 3 3 3 3 3 3 3 3 2 1 4 1
 [445] 2 3 2 2 3 3 3 3 3 1 1 3 2 2 1 3 1 4 3 2 3 2 2 3 1 3 1 3 3 3 3 2 2 2 3 3 3
 [482] 2 3 3 3 3 1 4 3 3 2 3 3 3 3 3 2 2 1 3 3 3 3 1 2 2 3 3 3 2 3 3 3 3 3 3 3 3
 [519] 3 3 3 2 2 2 2 3 3 3 3 1 3 1 1 1 1 1 1 2 2 3 2 3 2 2 2 1 2 1 2 3 3 3 3 3 2
 [556] 2 3 3 1 3 2 1 2 2 2 1 1 1 1 1 2 2 3 3 3 3 3 2 3 1 3 3 1 1 2 3 3 3 3 1 3 2
 [593] 3 3 3 3 3 1 4 3 3 1 3 3 3 3 3 3 3 3 3 1 1 1 3 1 3 3 1 2 3 2 3 1 3 2 1 3 2
 [630] 2 2 3 3 1 1 3 3 3 3 1 3 3 3 3 2 3 3 3 3 1 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 2
 [667] 2 3 3 3 1 3 2 3 2 3 1 2 3 3 1 3 3 2 4 3 3 1 1 3 3 3 3 3 3 1 3 1 1 3 3 3 2
```

```
[3405] 1 2 4 2 1 3 2 2 3 2 2 3 2 4 4 2 4 4 4 1 1 1 1 2 2 1 1 3 3 3 3 1 1 4 4 4 1
[3442] 1 4 1 1 1 2 1 2 2 4 1 1 1 3 3 1 1 1 3 1 2 2 2 2 3 3 1 3 1 1 3 3 3 1 1 4 1
[3479] 1 4 4 4 1 1 2 1 3 2 3 1 4 1 1 1 1 2 1 4 4 4 1 1 1 2 2 2 1 1 1 2 2 1 1 2 2
[3516] 2 2 2 2 4 1 1 2 2 1 4 1 4 4 4 4 4 4 2 1 3 4 1 3 1 3 2 2 4 4 4 2 2 4 1 2 4
[3553] 2 2 4 1 3 3 3 2 2 2 2 4 4 2 2 3 4 2 4 1 1 4 1 2 4 4 1 4 4 2 4 2 3 3 3 3 2
[3590] 4 2 3 2 3 2 4 2 4 2 2 2 2 3 4 1 3 1 4 2 4 1 3 2 2 2 2 2 3 3 3 3 2 2 2 2 4
[3627] 4 3 3 3 3 3 3 1 1 1 1 1 4 1 2 4 2 2 1 2 3 3 3 3 3 1 1 1 1 1 2 2 4 2 2 2 3
[3664] 1 1 2 3 1 1 4 2 4 2 1 1 4 4 4 1 4 3 3 3 3 3 3 3 1 2 1 1 4 2 2 3 3 2 1 1 1
[3701] 4 1 4 4 1 4 4 4 4 4 2 4 1 4 1 1 2 1 3 1 4 4 4 4 2 2 1 1 2 1 1 1 1 1 4 4 3
[3738] 3 3 3 3 3 3 1 1 1 1 1 3 3 3 4 4 3 1 1 4 1 4 1 3 4 4 1 4 4 3 2 2 2 4 2 4 4
[3775] 4 2 4 4 3 4 2 2 4 4 4 4 1 2 1 4 4 1 1 4 2 4 2 2 4 1 4 1 4 4 2 4 4 4 4 1 4
[3812] 4 2 2 4 4 3 4 4 4 2 2 2 2 2 4 4 2 2 2 4 2 2 2 4 2 4 4 4 3 4 4 4 4 4 2 1 4
[3849] 4 2 2 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 2 2 4 4 4 2 4 4 4 4 1 1 4 4 4 4 1 1 3
[3886] 4 2 4 1 1 1 2 1 2 4 4 4 4 4 4 4 1 4 2 4 4 4 4 1 4 3 1 4 2 2 2 4 4 2 4 4 2
[3923] 2 4 4 4 4 4 2 2 1 4 4 2 4 4 2 2 4 4 4 1 4 4 4 1 1 4 4 4 4 4 4 1 4 4 2 2 2
[3960] 2 4 2 4 1 4 1 1 2 2 4 1 1 4 3 3 4 3 4 2 4 4 2 4 4 2 1 3 4 4 4 4 2 1 4 4 2
[3997] 4 1 4 1 4 1 4 3 3 3 3 3 3 3 3 4 1 4 4 4 2 4 4 4 4 4 4 1 2 4 4 4 4 4 4 4 4
[4034] 4 4 4 4 2 4 3 1 4 1 1 1 4 2 1 1 1 4 4 4 3 3 3 3 3 3 3 3 3 4 4 4 2 4 4 4 1
[4071] 3 1 4 3 3 3 3 3 3 4 4 4 3 1 4 4 4 1 4 3 1 1 4 4 4 4 1 4 4 4 4 4 2 2 4 4 4
[4108] 2 4 2 4 1 1 1 2 4 4 4 4 1 4 4 4 4 4 2 4 4 1 1 4 4 1 4 4 4 3 3 3 3 1 1 1 1
[4145] 4 1 4 3 3 4 4 1 4 4 4 4 4 4 4 4 4 1 4 2 1 1 4 1 4 4 4 4 4 4 4 4 4 4 4 4 1
[4182] 1 1 4 4 4 1 4 4 4 4 4 1 4 1 4 1 1 4 3 3 4 4 4 2 2 2 1 1 2 3 1 3 4 1 1 1 2
[4219] 4 1 2 1 1 4 2 2 2 2 2 4 2 2 4 4 2 4 1 1 2 1 2 2 2 1 4 1 1 1 1 1 1 3 3 3 3
[4256] 3 3 4 4 4 2 2 4 1 2 4 2 2 2 2 4 2 4 4 4 2 2 2 4 4 4 2 2 2 4 4 4 2 4 4 2 4
[4293] 4 4 4 4 4 4 4 2 4 4 1 1 4 4 4 4 4 4 2 3 4 4 4 4 4 4 4 2 2 4 4 1 4 2 4 4 4 3
[4330] 3 3 3 3 3 3 1 3 4 4 4 4 4

Within cluster sum of squares by cluster:
[1] 6202.836 5842.081 8524.159 4599.996
 (between_SS / total_SS =  36.7 %)

Available components:

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

```
str(kc_4)
```

✓ 1.1s

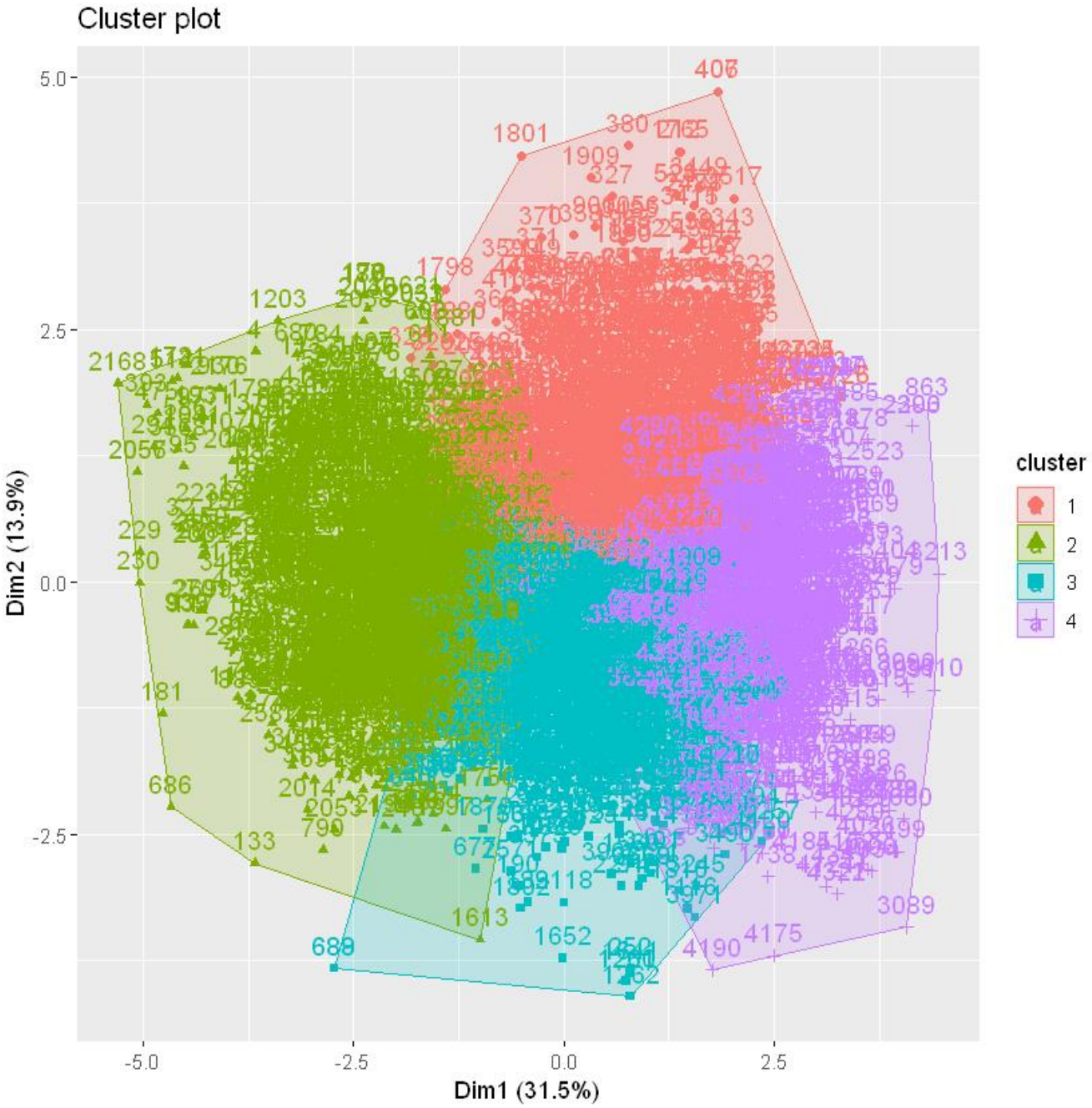
```
List of 9
 $ cluster     : int [1:4342] 2 2 2 3 1 3 2 1 3 3 ...
 $ centers     : num [1:4, 1:11] -0.498 0.838 0.146 -0.667 -0.349 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:4] "1" "2" "3" "4"
  .. ..$ : chr [1:11] "fixed_acidity" "volatile_acidity" "citric_acid" "residual_sugar" ...
 $ totss       : num 39754
 $ withinss    : num [1:4] 6203 5842 8524 4600
 $ tot.withinss: num 25169
 $ betweenss   : num 14585
 $ size        : int [1:4] 1058 1016 1393 875
 $ iter        : int 5
 $ ifault      : int 0
 - attr(*, "class")= chr "kmeans"
```

```
table(y,kc_4$cluster)
```

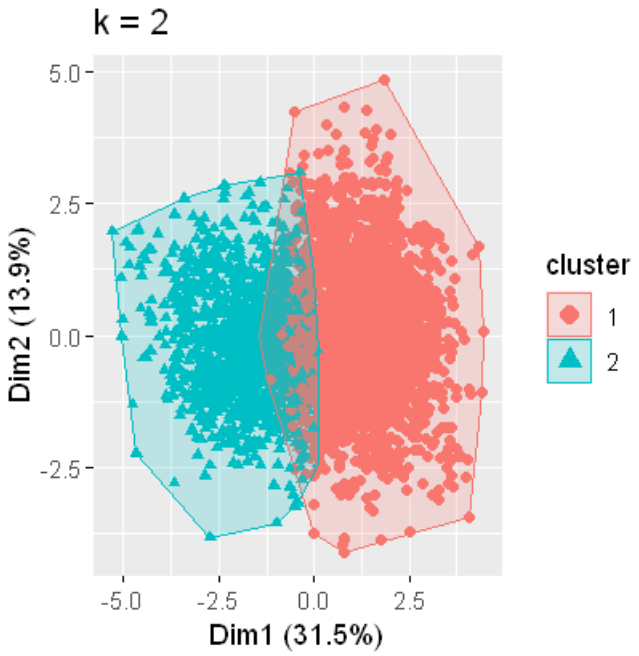
✓ 0.2s

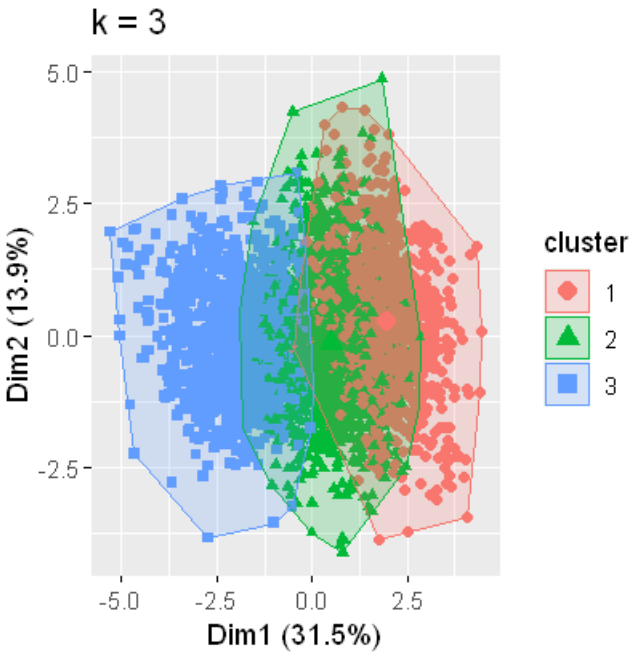
```
y     1   2   3   4
  1 276 316 632  75
  2 547 488 617 378
  3 203 173 125 350
  4  32  39  19  72
```

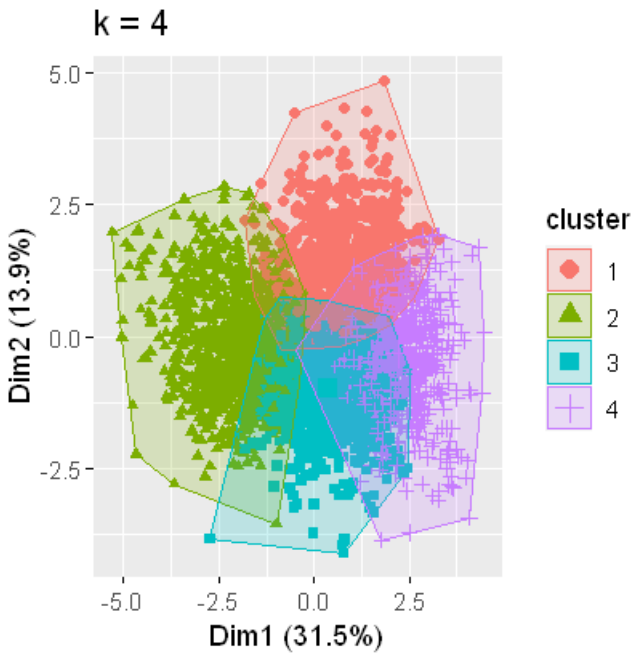
```
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4
         1 632 617 125  19
         2 316 488 173  39
         3  75 378 350  72
         4 276 547 203  32

Overall Statistics

               Accuracy : 0.3459
                 95% CI : (0.3318, 0.3603)
    No Information Rate : 0.4675
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1233

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

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity            0.4865   0.2404  0.41128  0.19753
Specificity            0.7499   0.7716  0.84961  0.75455
Pos Pred Value         0.4537   0.4803  0.40000  0.03025
Neg Pred Value         0.7738   0.5364  0.85549  0.96041
Prevalence             0.2992   0.4675  0.19599  0.03731
Detection Rate         0.1456   0.1124  0.08061  0.00737
Detection Prevalence   0.3208   0.2340  0.20152  0.24367
Balanced Accuracy      0.6182   0.5060  0.63045  0.47604
```


Cluster plot
Dim2 (13.9%)
Dim1 (31.5%)
5.0
2.5
0.0
-2.5
-5.0
-2.5
0.0
2.5
cluster
1
2
3
4

Visualizing all together,







Compression Table

Class1 = quality 5
Class2 = quality 6
Class3 = quality 7
Class4 = quality 8

Cluster amount	BSS/TSS	Accuracy	Recall/Sensitivity	Specificity
k=2	24.9%	45.69%	class1:56.04% class2:61.86%	class1:68.68% class2:39.23%
k=3	32.0%	40.53%	class1:53.12% class2:31.38% class3:50.88%	class1:70.85% class2:70.11% class3:71.24%
k=4	36.7%	34.59%	class1:48.65% class2:24.04% class3:41.12% class4:19.75%	class1:74.95% class2:77.16% class3:84.96% class4:75.45%

Define "winning" cluster

From above automated tools form previous chapter gives most optimal cluster is k=2 and after experimenting with different k values form this chapter, we will be able to see accuracy was changing time to time when executing the code with different times, but BSS/TSS percentage didn't changed. For considering about BSS/TSS percentage reflect good fit of the clusters. The highest BSS/TSS percentage is k=4 and good to have as 4 clusters because we are dealing with some quality factor and if we consider as k=2 there are lots of different variations of wine samples may be together as same cluster. Because of those reasons I will consider k=4 is the "winning" cluster for this dataset and clustering is subjective and it can be use as different purposes.

Evaluation Metrics,

After we are doing Machine Learning Task, we must evaluate our results, so for that reason we use some evaluation metrics,

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

When we have labeled dataset to perform our task after the prediction, we will be able to define above matrix this is call confusion matrix,

True Positives: these values correctly predicted positive by model and actual class of predicted value is also yes

True Negatives: these values correctly predicted negative by model and actual class of predicted value is also no

False Positives: when actual class is no, and predicted class is yes

False Negative: when actual class in yes, and predicted class is no

Accuracy:

Accuracy is the most common performance measure, and it is simply a ratio of correctly predicted values vs total observations.

$$\textbf{Accuracy} \quad \frac{TP + TN}{(TP + TN + FP + FN)}$$

Precision:

Ratio of correctly predicted positive values to the total predicted positive observations

$$\textbf{Precision} \quad \frac{TP}{(TP + FP)}$$

Recall (Sensitivity):

Ratio of correctly predicted positive observations to all observation in actual class.

$$\textbf{Sensitivity} \quad \frac{TP}{(TP + FN)}$$

Dimensionality Reduction (PCA)

In this chapter we will reduce the intentionality of our dataset and evaluate which features are mostly affect to the quality of the wine sample and perform k-means “winning” clustering number for new dataset which is reduced dimensionality by using PCA.

```
head(df_final)
```
✓ 0.5s R

A data.frame: 6 × 13

	index	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	p_h	sulphates	
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
1	1	1.5238976	-0.04648274	0.6271255	-0.9831716	-0.58570079	-1.5272358	-1.8230268	-1.0701698	-1.31955843	0.61051063	1.1
2	2	2.1293483	-0.46690270	0.5433238	-0.4429502	-0.49277626	-1.1554516	-0.7152461	0.2268701	-0.32155169	0.34731453	-0.6
3	3	1.2817173	-0.99242766	0.2919184	-1.0322826	-0.26046491	-1.2174157	-1.5340406	-0.6710806	-0.05541657	1.22463485	0.2
4	4	1.7660778	1.53009213	2.3869628	2.5135339	-0.26046491	0.3316852	0.8019318	2.0560288	-1.38609221	1.57556298	-0.6
5	5	-0.4135447	0.37393723	-1.6355225	0.2053154	-0.07461584	-0.1020631	-0.1372736	0.4929295	0.21071856	0.08411844	-0.8
6	7	-1.2611757	-0.04648274	-1.1327118	1.6688241	-0.07461584	-0.8456315	0.9705071	0.7257315	1.20872529	-1.05639797	-0.2

```
principle_components <- prcomp(df_final[2:12])
```
✓ 0.7s R

Above figure shows the dataset used for PCA analysis and the PCA function with numerical features.

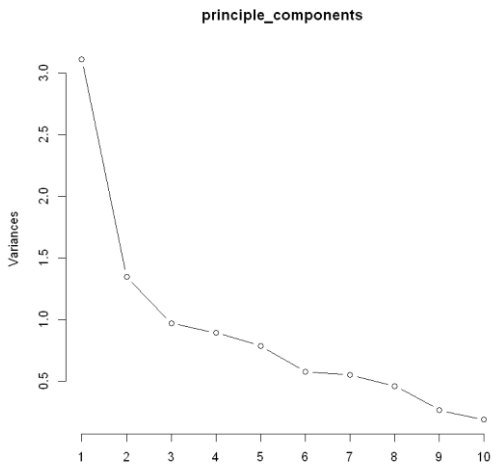
```
Standard deviations (1, .., p=11):
 [1] 1.7637111 1.1597870 0.9848076 0.9445540 0.8866711 0.7602305 0.7422495
 [8] 0.6783070 0.5146607 0.4335380 0.1092516

Rotation (n x k) = (11 x 11):
                             PC1         PC2         PC3         PC4
fixed_acidity        -0.137581852 -0.59801192  0.32565474  0.04710833
volatile_acidity      0.003614621  0.03454142 -0.37313686  0.48562984
citric_acid          -0.096724297 -0.23355876  0.44578417  0.05255540
residual_sugar       -0.441145600 -0.04157095 -0.30318924 -0.05783970
chlorides            -0.145855652  0.01309497 -0.01897213 -0.11293925
free_sulfur_dioxide  -0.305204270  0.24747938  0.20906901  0.50565029
total_sulfur_dioxide -0.416041688  0.23275917  0.15670945  0.42166843
density              -0.518684644 -0.02011572 -0.11437235 -0.25615517
p_h                   0.104149769  0.63234583  0.08438014 -0.24233680
sulphates            -0.055352542  0.26168678  0.60375550 -0.16411479
alcohol               0.454570232 -0.02388673  0.10354344  0.39789314
                             PC5         PC6         PC7         PC8
fixed_acidity        -0.161274238  0.231218596 -0.34679043  0.522400466
volatile_acidity     -0.652611049  0.292972116  0.01607555 -0.160497325
citric_acid           0.142672358  0.525802181  0.47980868 -0.454670043
residual_sugar       -0.042171665 -0.079895016  0.58610689  0.325514328
chlorides            -0.055214871  0.038022725 -0.20320557 -0.215145067
free_sulfur_dioxide   0.390893567 -0.214201110  0.02126846  0.127175970
total_sulfur_dioxide -0.006873509  0.094160372 -0.30313468 -0.075041156
density              -0.099038852  0.108052216  0.08527173  0.146946032
p_h                   0.028370132  0.602133826 -0.05648120  0.361294602
sulphates            -0.599483282 -0.383922024  0.16299366 -0.006149795
alcohol              -0.008253307 -0.004856779  0.36752756  0.410361899
```

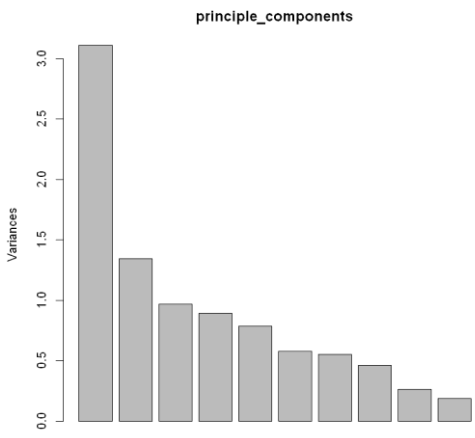
```
                              PC9         PC10         PC11
fixed_acidity         0.12399667 -0.015785105  0.164076688
volatile_acidity      0.29088029  0.036655365  0.004444604
citric_acid           0.02466066  0.010582145  0.011155033
residual_sugar       -0.14326955 -0.101338561  0.468993686
chlorides             0.01176473 -0.933407495  0.044465279
free_sulfur_dioxide   0.56987028 -0.074103672 -0.025744681
total_sulfur_dioxide -0.67300852  0.095212839  0.043922423
density               0.04244632  0.035595680 -0.773594188
p_h                   0.09672641 -0.020575177  0.128729108
sulphates             0.04579256  0.008058951  0.038193900
alcohol              -0.29629970 -0.316960567 -0.363225163
```

Below figure shows summary details of principle components in our data frame

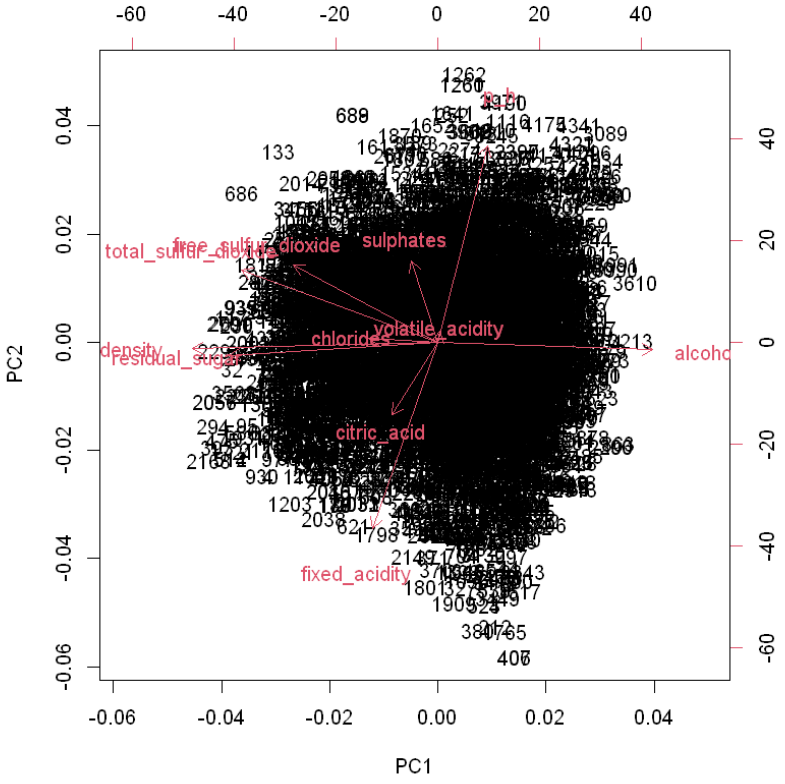
```
summary(principle_components)
```
✓ 0.3s

```
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7
Standard deviation     1.7637 1.1598 0.9848 0.94455 0.88667 0.76023 0.74225
Proportion of Variance 0.3397 0.1469 0.1059 0.09742 0.08585 0.06311 0.06016
Cumulative Proportion  0.3397 0.4866 0.5925 0.68989 0.77574 0.83885 0.89901
                          PC8     PC9    PC10   PC11
Standard deviation     0.67831 0.51466 0.43354 0.1093
Proportion of Variance 0.05024 0.02892 0.02052 0.0013
Cumulative Proportion  0.94925 0.97817 0.99870 1.0000
```

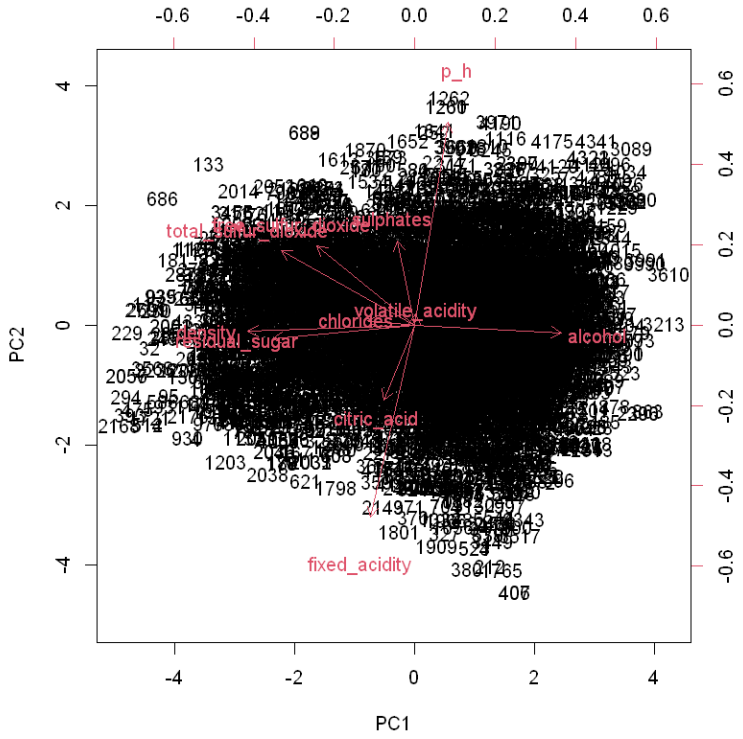




We can pot PCA's to get some idea about it, but it's difficult to get exact idea about that because we are plotting 2d graph and this PCA's in various 12 dimensions.



Changing the Scale,



Merging new columns with data frame,

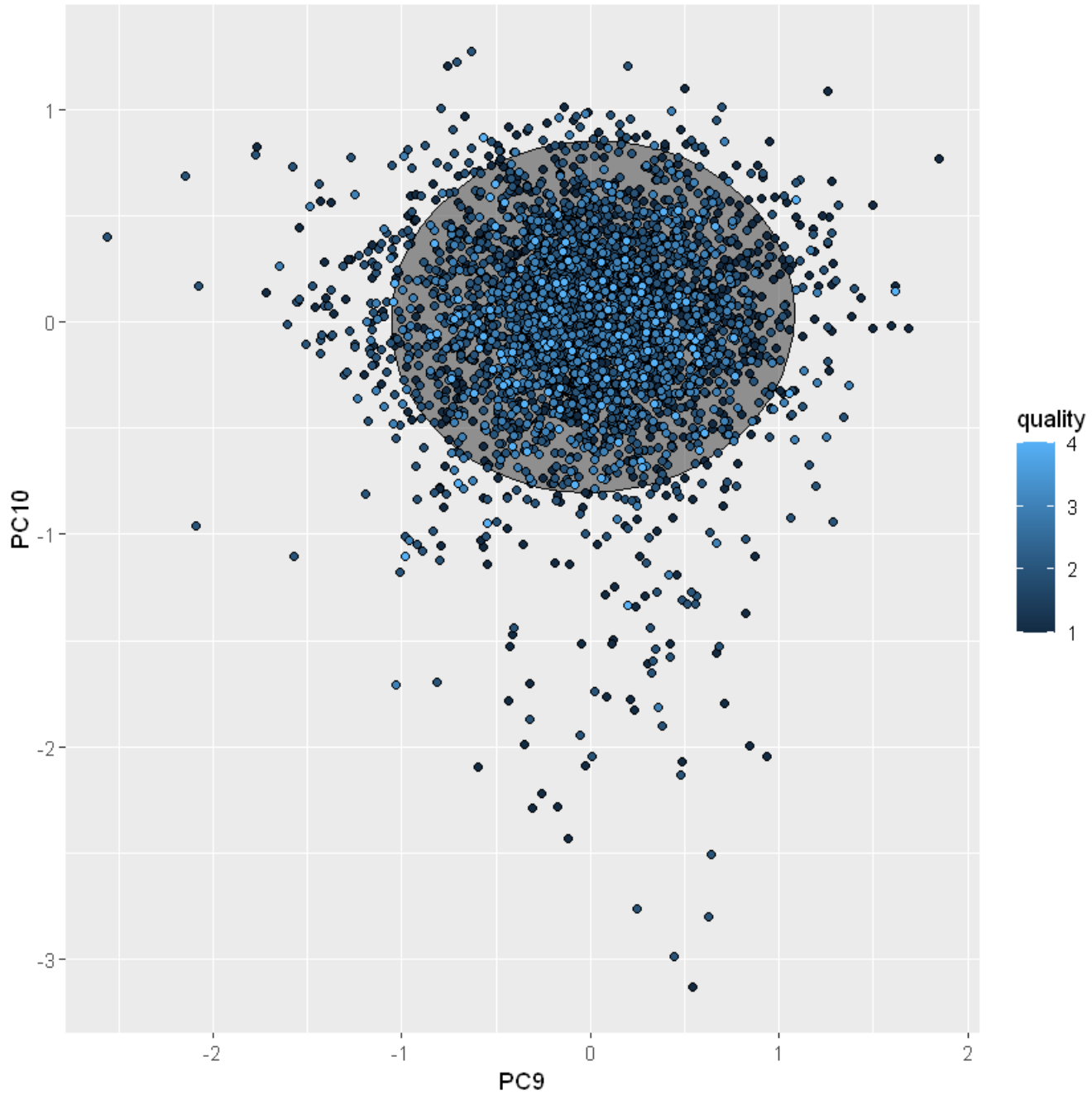
```
df_final_with_pcs <- cbind(df_final, principle_components$x)
head(df_final_with_pcs)
```

✓ 0.3s R

A data.frame: 6 × 24

al_sulfur_dioxide	density	p_h	...	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
<dbl>	<dbl>	<dbl>	...	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
-1.8230268	-1.0701698	-1.31955843	...	-2.507450	1.037543	-0.3458290	-0.9958955	-0.2267304	0.3549709	0.05716721	0.2180437	0.08122723	-0.02217554
-0.7152461	0.2268701	-0.32155169	...	-1.955253	1.024154	-1.1843357	-0.6483172	0.5822436	-0.6051595	0.38769275	0.2500563	0.60554450	0.16234138
-1.5340406	-0.6710806	-0.05541657	...	-1.191507	1.612201	-1.4855391	-0.6478954	-0.1857300	-0.2073085	0.27955324	0.3527243	0.05386811	0.15663117
0.8019318	2.0560288	-1.38609221	...	-1.886857	1.072986	0.6475778	-2.1442520	0.7347356	2.1111434	-0.09654371	0.2678407	0.31870617	0.05228100
-0.1372736	0.4929295	0.21071856	...	0.743327	-1.143859	-0.5370382	-0.5925853	-0.6610490	-0.7228263	0.29321089	0.3476583	0.22213184	-0.04310457
0.9705071	0.7257315	1.20872529	...	1.445659	-2.036778	-0.6370440	0.2145349	0.4920310	0.2976442	0.62233304	-1.3600306	0.03661862	0.27470212

Plotting cumulative score > 96% features (PC9, PC10, PC11),



K-means Clustering

```
x_2 = df_final_with_pcs[22:24]
y_2 = df_final_with_pcs$quality
```
✓ 0.7s

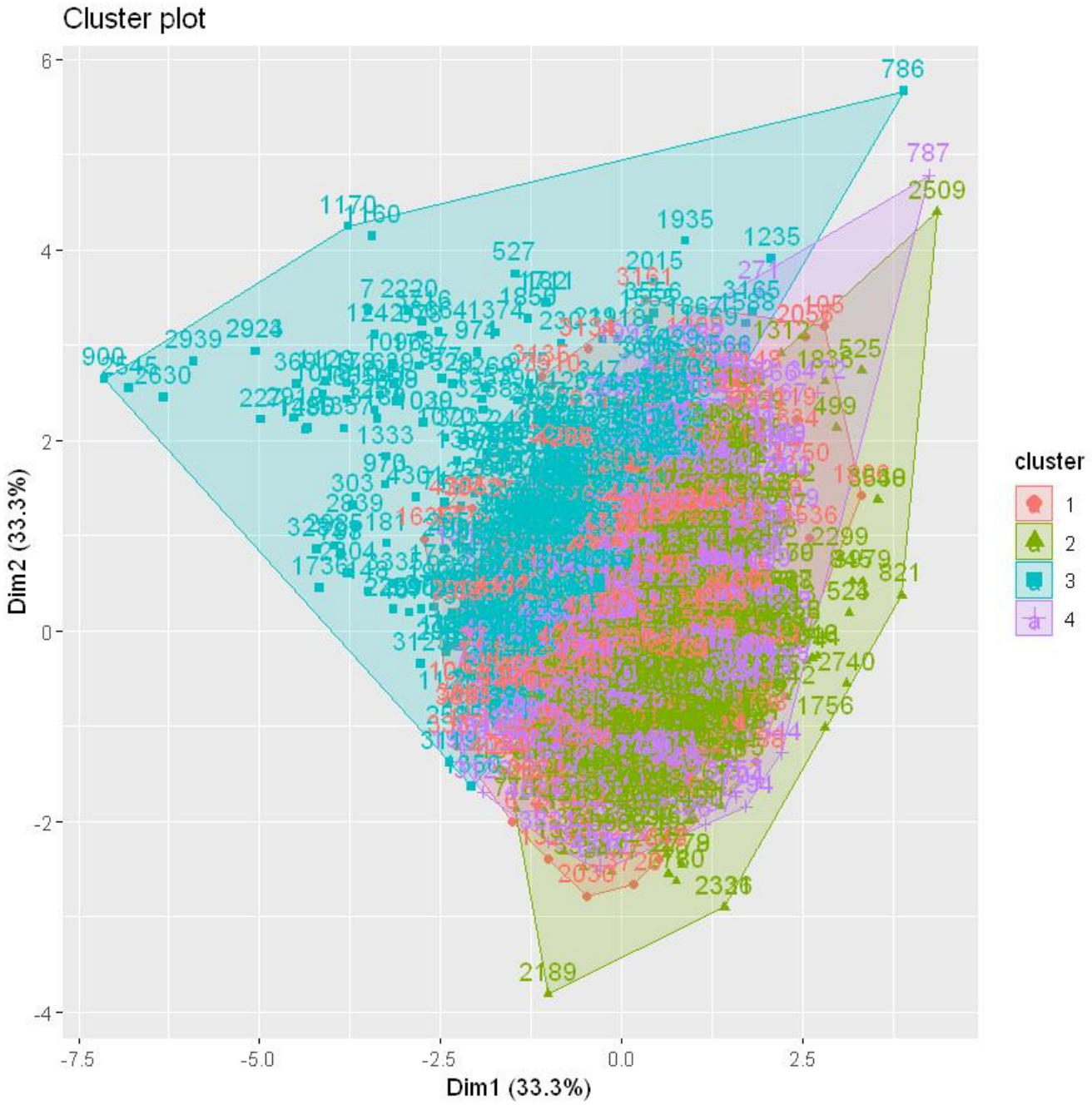
```
kcpca <- kmeans(x_2,4)
```
✓ 0.8s

Above figure shows getting data and labels to do clustering and perform clustering with k=4

```
K-means clustering with 4 clusters of sizes 763, 1604, 784, 1191

Cluster means:
        PC9         PC10         PC11
1 -0.76750362 -0.01597494  0.001815782
2 -0.08194952  0.22506439  0.002494829
3  0.03895665 -0.59107899 -0.002498812
4  0.57641501  0.09621454 -0.002878319

Clustering vector:
   [1] 2 2 4 2 4 1 3 2 3 3 3 3 2 4 4 2 1 2 3 4 3 4 4 2 3 4 3 3 3 4 2 1 4 4 4 4 3
  [38] 4 4 3 4 2 2 2 2 2 4 4 4 4 1 4 2 4 1 1 3 4 2 2 4 3 2 2 2 2 2 2 2 2 2 4 2
  [75] 4 4 2 2 2 2 2 4 4 2 4 2 2 2 2 2 4 3 2 4 3 3 4 2 2 1 4 2 2 2 1 4 3 4 4 2 4
 [112] 4 4 2 4 3 1 4 4 4 4 2 2 4 3 2 4 1 4 1 2 3 4 3 2 2 4 3 4 4 2 2 2 2 2 1 1 3
 [149] 4 3 1 1 2 4 4 4 4 4 2 4 4 4 4 2 4 2 1 2 2 2 2 2 2 2 2 2 4 4 4 4 3 3 3 3 2
 [186] 2 2 2 2 2 1 1 3 1 2 1 4 4 4 2 2 4 1 3 2 2 2 2 2 2 2 4 1 4 2 2 2 4 1 4 3 2
 [223] 2 2 2 2 3 3 1 3 2 1 1 4 2 2 2 2 2 2 2 2 2 4 2 4 4 4 4 4 4 2 3 4 4 4 2 1 2
 [260] 2 2 4 4 2 3 4 2 2 2 2 4 4 2 1 2 2 4 2 2 2 4 2 4 4 3 4 2 4 2 2 4 2 4 1 3 2
 [297] 4 4 2 2 4 4 3 3 3 3 3 3 1 4 3 4 2 4 1 4 3 4 2 4 4 2 2 4 2 4 4 4 4 2 2 4 2
 [334] 1 2 4 4 2 4 4 4 2 4 4 3 1 3 1 1 1 1 2 2 2 2 2 2 2 4 2 4 4 2 1 1 1 1 2 3 2
 [371] 4 1 1 2 1 2 1 1 4 3 4 2 2 4 4 2 3 2 2 3 4 2 1 4 2 3 2 2 2 2 1 1 2 2 1 4 4
 [408] 3 3 1 3 2 4 4 4 4 4 4 4 1 4 4 4 4 2 2 2 2 2 4 2 2 2 4 2 1 3 3 3 3 4 2 4 2
 [445] 3 1 4 4 2 2 2 2 2 4 4 3 4 4 4 3 4 2 3 2 3 2 2 2 4 2 1 4 4 4 1 3 1 1 2 1 1
 [482] 2 1 2 2 2 2 4 2 2 2 2 2 4 4 2 1 4 2 3 2 2 3 3 4 3 2 3 2 4 4 1 4 1 2 4 4 2
 [519] 1 1 4 4 2 2 2 4 3 3 3 1 4 3 4 4 1 1 3 4 4 4 2 2 2 4 2 2 4 2 4 4 2 2 2 2 4
 [556] 4 2 1 2 2 3 4 2 2 2 2 4 4 2 2 4 4 2 2 1 1 4 2 2 2 2 1 4 2 2 1 4 4 4 4 4 2
 [593] 3 3 3 2 2 2 2 1 1 3 3 3 3 4 1 3 3 1 2 3 3 2 4 2 4 2 4 1 2 1 2 3 2 1 1 3 1
 [630] 1 4 4 4 1 3 2 2 2 3 3 1 2 4 2 2 4 4 3 3 4 4 4 4 4 4 3 2 4 2 1 1 4 2 2 2 2
 [667] 2 4 4 4 2 2 2 2 4 3 3 3 1 4 1 1 2 4 4 3 2 3 3 2 2 2 2 2 1 1 1 1 1 2 2 2 4
 [704] 4 4 4 2 4 2 3 3 2 2 2 2 2 2 2 2 2 2 3 2 2 3 2 2 1 2 4 2 4 2 4 3 4 3 2 2 2
 [741] 4 2 4 2 2 1 2 3 2 1 1 2 2 2 2 2 2 2 2 4 2 2 1 3 2 4 3 4 2 4 2 2 2 2 2 4 4
```

```
[3664] 3 4 1 2 4 4 2 4 2 1 2 1 2 4 4 2 4 1 1 1 1 1 1 1 3 3 3 4 2 3 3 4 4 2 4 4 4
[3701] 3 4 3 4 2 4 3 3 2 2 2 4 4 2 2 3 1 2 1 1 2 2 2 2 2 2 2 2 2 4 4 4 4 4 2 4 1
[3738] 1 1 1 1 1 1 3 3 4 4 4 4 4 4 4 4 4 4 2 4 3 4 3 3 2 2 1 2 1 4 2 2 2 4 2 4 4
[3775] 4 4 2 2 1 1 4 4 3 3 4 3 1 2 1 1 1 2 2 3 2 4 2 2 4 1 4 1 2 2 4 3 1 3 4 4 4
[3812] 3 2 2 4 2 3 2 2 2 4 1 1 2 1 1 2 1 2 2 1 2 1 1 2 1 1 1 4 4 2 3 2 2 2 3 2 2
[3849] 2 4 3 4 4 4 2 3 3 2 2 2 2 2 2 2 2 2 1 1 2 1 4 1 1 2 4 3 4 2 2 1 2 1 2 2 1
[3886] 3 3 4 1 2 3 3 3 3 2 3 2 2 2 4 3 4 2 3 1 3 3 3 2 1 2 1 2 2 2 1 3 3 4 3 1 2
[3923] 2 2 2 3 1 2 2 2 2 1 3 2 3 1 3 2 1 1 3 4 1 1 3 2 2 3 3 2 3 3 3 4 1 4 4 2 2
[3960] 2 1 2 4 4 2 2 2 3 2 4 2 2 2 4 4 1 4 2 2 4 1 2 3 3 1 2 4 4 2 1 1 2 3 2 1 2
[3997] 2 3 2 2 1 2 1 1 1 1 1 1 1 1 1 2 4 2 1 3 4 3 1 4 1 2 3 4 4 3 4 4 3 4 4 4 4
[4034] 3 3 3 3 4 1 2 4 2 2 4 4 2 4 3 3 3 1 3 1 1 1 1 1 1 1 1 4 2 2 4 4 1 3 2 3
[4071] 3 3 3 1 1 1 1 1 1 3 1 3 1 1 2 4 3 4 2 3 2 4 1 1 1 1 4 2 3 4 2 3 4 4 1 1 3
[4108] 1 3 1 1 2 1 2 2 2 2 1 2 3 2 2 4 3 4 4 3 2 2 4 2 1 2 4 4 3 3 3 3 3 2 2 2 2
[4145] 4 2 4 1 1 2 2 4 1 2 1 1 4 3 3 4 4 1 3 4 2 2 2 2 4 4 3 2 4 4 4 3 2 4 2 1 4
[4182] 4 4 4 4 4 4 1 3 2 4 2 1 4 2 2 4 3 4 4 4 2 3 3 1 1 1 2 2 4 4 1 4 2 4 2 2 2
[4219] 2 2 4 2 4 4 4 2 4 2 4 3 2 2 3 3 2 3 4 2 2 1 2 1 1 2 3 4 2 2 2 2 2 1 1 1 1
[4256] 1 1 2 4 4 2 2 4 4 2 2 3 3 3 2 1 2 4 3 2 4 4 4 1 2 2 1 2 2 2 2 2 3 3 3 3 2
[4293] 2 4 1 1 1 2 2 2 3 1 4 4 1 3 1 3 1 1 1 2 3 3 1 3 3 4 1 1 2 2 3 4 3 1 3 3 3
[4330] 3 3 3 3 3 3 2 3 3 3 1 1 1

Within cluster sum of squares by cluster:
[1] 167.5135 196.1997 229.2974 201.0049
 (between_SS / total_SS =  60.6 %)

Available components:

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

We can see BSS/TSS is improved than previous chapter and it tells this clusters are fitted well.


Cluster plot
Dim2 (33.3%)
Dim1 (33.3%)
6
4
2
0
-2
-4
-7.5
-5.0
-2.5
0.0
2.5
786
787
2509
1935
1235
527
2015
1170
1160
2939
2924
900
2630
1333
303
2189
2740
1756
821
525
499
2299
cluster
1
2
3
4

```
table(y_2,kcpca$cluster)
```
✓ 1.8s

```
y_2   1   2   3   4
  1 184 493 220 402
  2 363 756 386 525
  3 182 303 140 226
  4  34  52  38  38
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4
         1 417 535 229  40
         2 485 740 302  52
         3 179 362 163  30
         4 218 393 157  40

Overall Statistics

               Accuracy : 0.3132
                 95% CI : (0.2994, 0.3273)
    No Information Rate : 0.4675
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0269

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

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity           0.32102   0.3645  0.19154 0.246914
Specificity           0.73579   0.6371  0.83644 0.816268
Pos Pred Value        0.34152   0.4687  0.22207 0.049505
Neg Pred Value        0.71740   0.5331  0.80931 0.965478
Prevalence            0.29917   0.4675  0.19599 0.037310
Detection Rate        0.09604   0.1704  0.03754 0.009212
Detection Prevalence  0.28121   0.3637  0.16905 0.186089
Balanced Accuracy     0.52840   0.5008  0.51399 0.531591
```

In previous chapter we perform k-means clustering with the original features of the dataset and at 'winning" cluster value we got BSS/TSS as 36.7% then after we performed PCA to reduce the dimensionality of the dataset and we was able to reduce the dimension of the dataset. Them after we performed k-means clustering and got BSS/TSS=60.6% so we can see in 2nd stage we got more fitted clusters with the new dataset after clustering, while k-means is an unsupervised machine learning algorithm and reduce the dimensionality of the dataset affected to the improvement of the machine learning algorithm's performance.

Conclusion

When we are dealing with Machine Learning first, we must clean the dataset and must do some preprocessing before dealing with machine learning. Then it will be very important to increase the performance of the Machine Learning model and after the machine learning stage we need to evaluate our outputs with some methods. Following these steps in sequency help to smooth the machine learning journey.

Time Series Forecasting

As the given instructions we must perform time series forecasting on energy data. We must introduce new input features from given data and crate some previous time period data to perform this task by changing the network architecture and validate by using RMSE, MSE and MAPE.

Time Series Forecasting

Time series forecasting is the Machine Learning technique which is used to make scientific predictions using Historical time stamped data. Time Series Forecasting involves building models through historical analysis and using them to make observations and drive future strategic decision-making.

Energy Forecasting Analysis

Basically, Energy Forecasting Analysis is a technique to predict future energy needs to archive demand and Supply equilibrium. In this problem we have historical electricity consumptions data in three hours and we Must predict next day electricity consumption for 11th hour.

In this domain several techniques were used by researchers which includes some traditional methods such as Regression, Time Series, Artificial Neural Networks (ANNs), Support Vector Machines (SVM), fuzzy logic and Gray Prediction. In this use case we need to construct Multi-Layer Perceptron Neural Network with Autoregressive approach.

Auto Regressive Models

When we use multiple regression model, we forecast the variable of interest using a linear combination of predictors, but in Autoregressive models we forecast the variable of interest using a linear combination of past values of the variable and the error term. AR models are named with the number of previous time period terms. Thus, an autoregressive model of order of p can be written as,

$$X_t = \sum_{r=1}^{p} \phi_r X_{t-r} + \epsilon_t$$

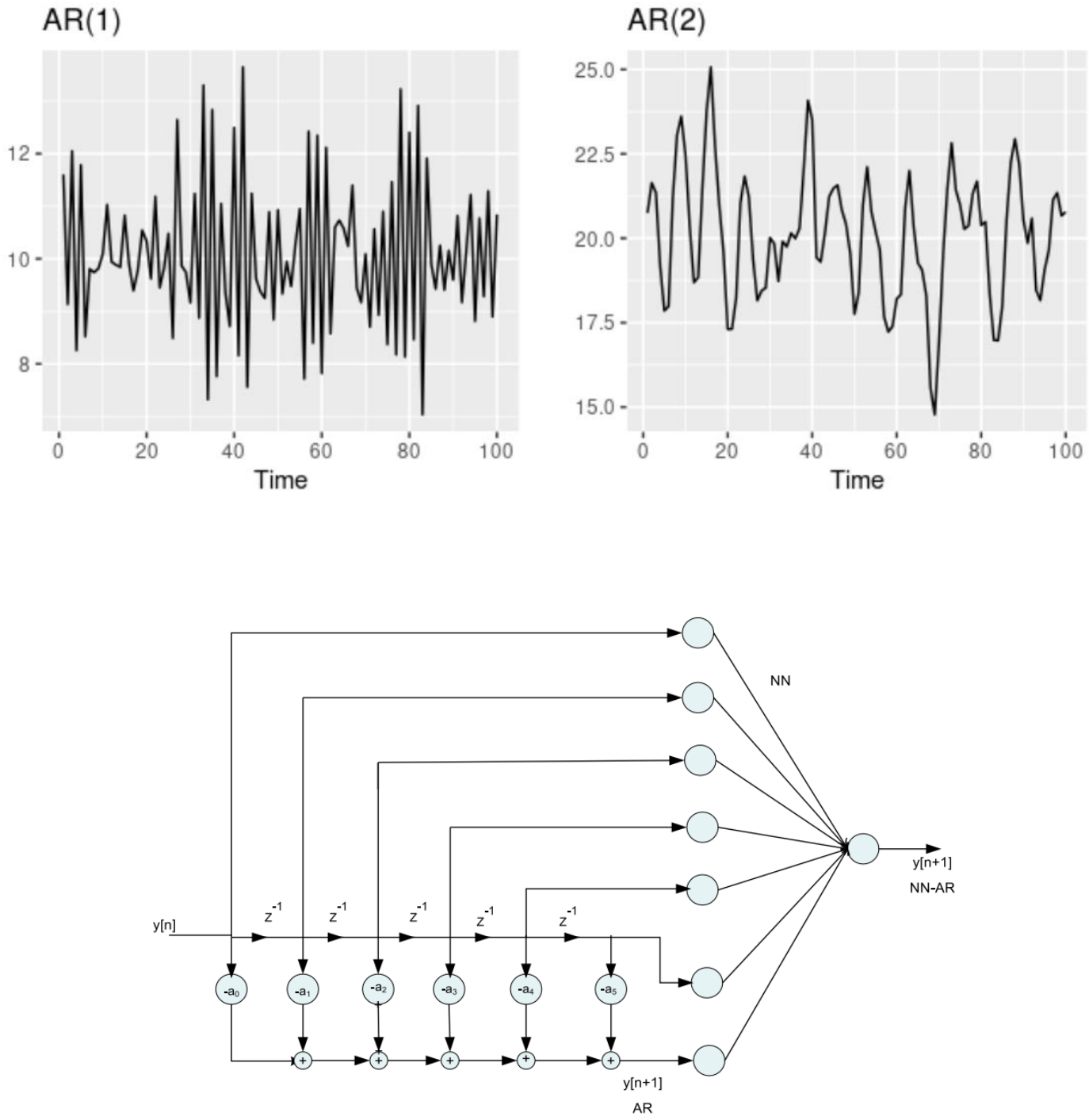


Figure. 1: The structure of AR-input ANN

When we use this autoregressive approach to this problem, we need to construct ANN's with time delayed values of the electricity loads. As per the instructions we will be able to use (t-4) level for experiments and week before values (t-7) also can be use specifically for electricity consumption forecast.

Steps followed in this problem.

1. Construct time delayed inputs
2. Normalization
3. One layer model experimenting with various layer nodes and input sets
4. Two-layer model experimenting with various layer nodes and input sets
5. Choosing best models from above stages
6. Experimenting with learning rate with best models
7. Choose best model
8. Prediction with best model and conclusion

Construct time-delayed values

```
df <- read_excel("data/UOW_load.xlsx")
head(df)
```

✓ 0.2s

A tibble: 6 × 4

Dates	09:00	10:00	11:00
<dttm>	<dbl>	<dbl>	<dbl>
2018-01-01	89.4	90.6	88.6
2018-01-02	108.2	104.6	106.0
2018-01-03	110.0	111.6	114.8
2018-01-04	106.4	104.4	109.0
2018-01-05	97.8	100.4	102.4
2018-01-06	87.0	90.8	87.2

```r
df$Dates<-as.Date(df$Dates)
```

✓ 0.5s

```r
names(df)[2] <- paste("ninth_hour")
names(df)[3] <- paste("tenth_hour")
names(df)[4] <- paste("eleventh_hour")
head(df)
```

✓ 0.7s

A tibble: 6 × 4

Dates	ninth_hour	tenth_hour	eleventh_hour
<date>	<dbl>	<dbl>	<dbl>
2018-01-01	89.4	90.6	88.6
2018-01-02	108.2	104.6	106.0
2018-01-03	110.0	111.6	114.8
2018-01-04	106.4	104.4	109.0
2018-01-05	97.8	100.4	102.4
2018-01-06	87.0	90.8	87.2

```r
summary(df[2:4])
```

✓ 0.7s

```
  ninth_hour       tenth_hour     eleventh_hour
Min.   : 50.40   Min.   : 49.20   Min.   : 48.20
1st Qu.: 81.55   1st Qu.: 83.00   1st Qu.: 84.75
Median : 97.50   Median :102.80   Median :105.40
Mean   : 95.35   Mean   : 99.63   Mean   :102.83
3rd Qu.:110.60   3rd Qu.:116.05   3rd Qu.:120.90
Max.   :141.20   Max.   :148.80   Max.   :156.40
```

```r
# insert all the inputs to the one data frame
# creating revious time frame values for autoregressive model
df_full = df %>%
 mutate(
    nine_lag_1 = lag(df$ninth_hour,1),
    ten_lag_1 = lag(df$tenth_hour,1),
    nine_lag_2 = lag(df$ninth_hour,2),
    ten_lag_2 = lag(df$tenth_hour,2),
    eleven_lag_1 = lag(df$eleventh_hour,1),
    eleven_lag_2 = lag(df$eleventh_hour,2),
    eleven_lag_3 = lag(df$eleventh_hour,3),
    eleven_lag_4 = lag(df$eleventh_hour,4),
    #roll mean
    nine_rollmean_4 = rollmean(ninth_hour,4, fill = NA),
    nine_rollmean_7 = rollmean(ninth_hour,7, fill = NA),
    ten_rollmean_4 = rollmean(tenth_hour,4, fill = NA),
    ten_rollmean_7 = rollmean(tenth_hour,7, fill = NA),
    eleven_rollmean_4 = rollmean(eleventh_hour,4, fill = NA),
    eleven_rollmean_7 = rollmean(eleventh_hour,7, fill = NA)) %>%
#Drop null coulumns
drop_na()
```

✓ 0.2s

Above figure shows creating time-delayed values using lag.

Defining various input vector sets for further experiments.

```r
#1st lag
df_full %>%
pivot_longer(cols = c(2, 3, 5, 6, 9), names_to = "kind", values_to = "energy_level") %>%
ggplot(aes(Dates, energy_level, color = kind)) +
geom_line() +
facet_wrap(~kind) + theme(axis.text.x = element_text(angle = 45, vjust =
0.5, hjust = 1
)) +
labs(x = "",
title = "First Set of Input Variables") +
theme(legend.position = "none")
```

✓ 0.5s

```r
#2nd lag
df_full %>%
pivot_longer(cols = c(2, 3, 5, 6, 9, 7, 8, 10), names_to = "kind", values_to = "energy_level") %>%
ggplot(aes(Dates, energy_level, color = kind)) +
geom_line() +
facet_wrap(~kind) + theme(axis.text.x = element_text(angle = 45, vjust =
0.5, hjust = 1
)) +
labs(x = "",
title = "Second Set of Input Variables") +
theme(legend.position = "none")
```

✓ 0.6s

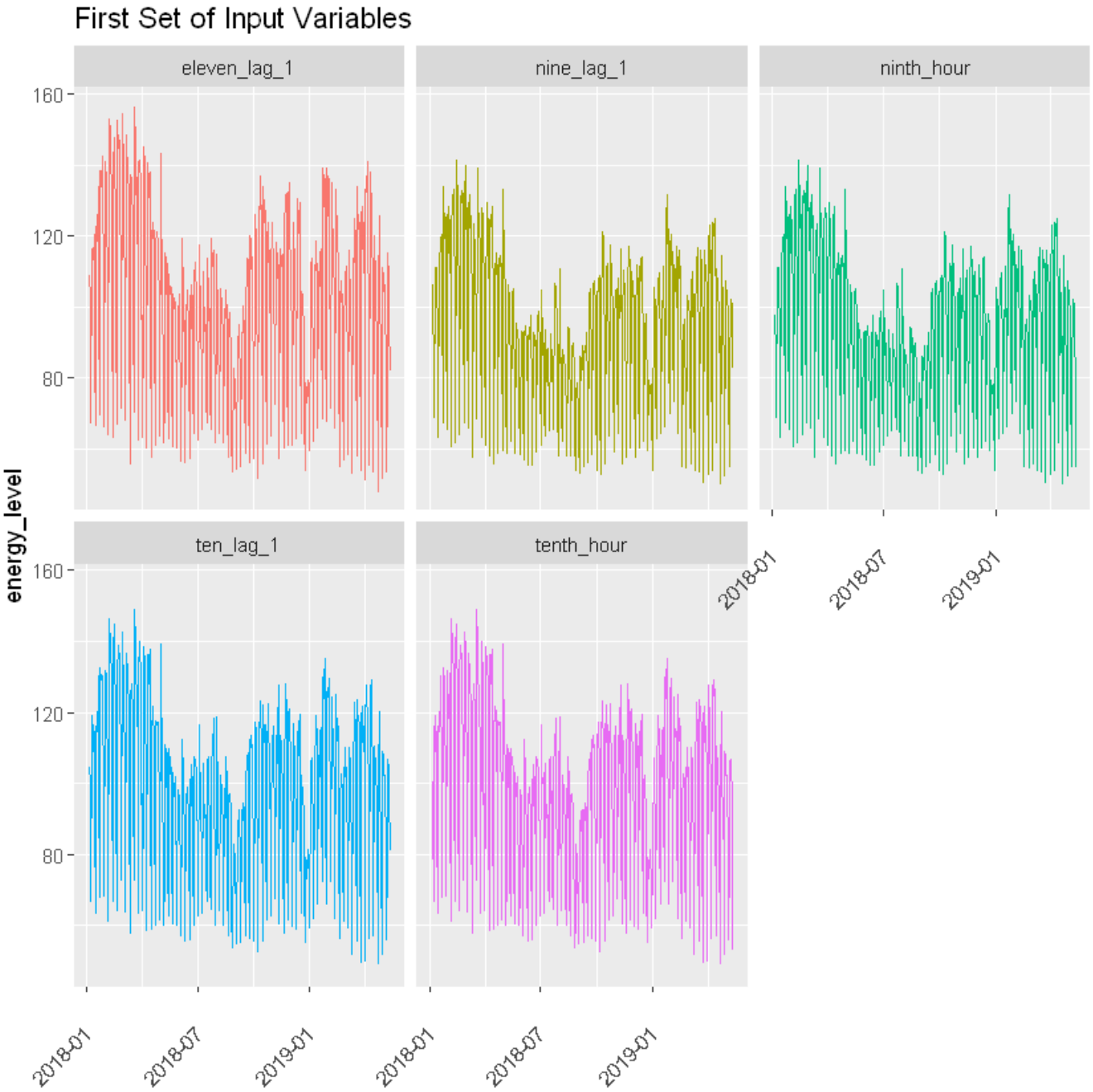
```r
#3rd lag
df_full %>%
pivot_longer(cols = c(2, 3, 5, 6, 9, 7, 8, 10, 11, 13, 15, 17), names_to = "kind", values_to = "energy_level") %>%
ggplot(aes(Dates, energy_level, color = kind)) +
geom_line() +
facet_wrap(~kind) + theme(axis.text.x = element_text(angle = 45, vjust =
0.5, hjust = 1
)) +
labs(x = "",
title = "Third Set of Input Variables") +
theme(legend.position = "none")
```

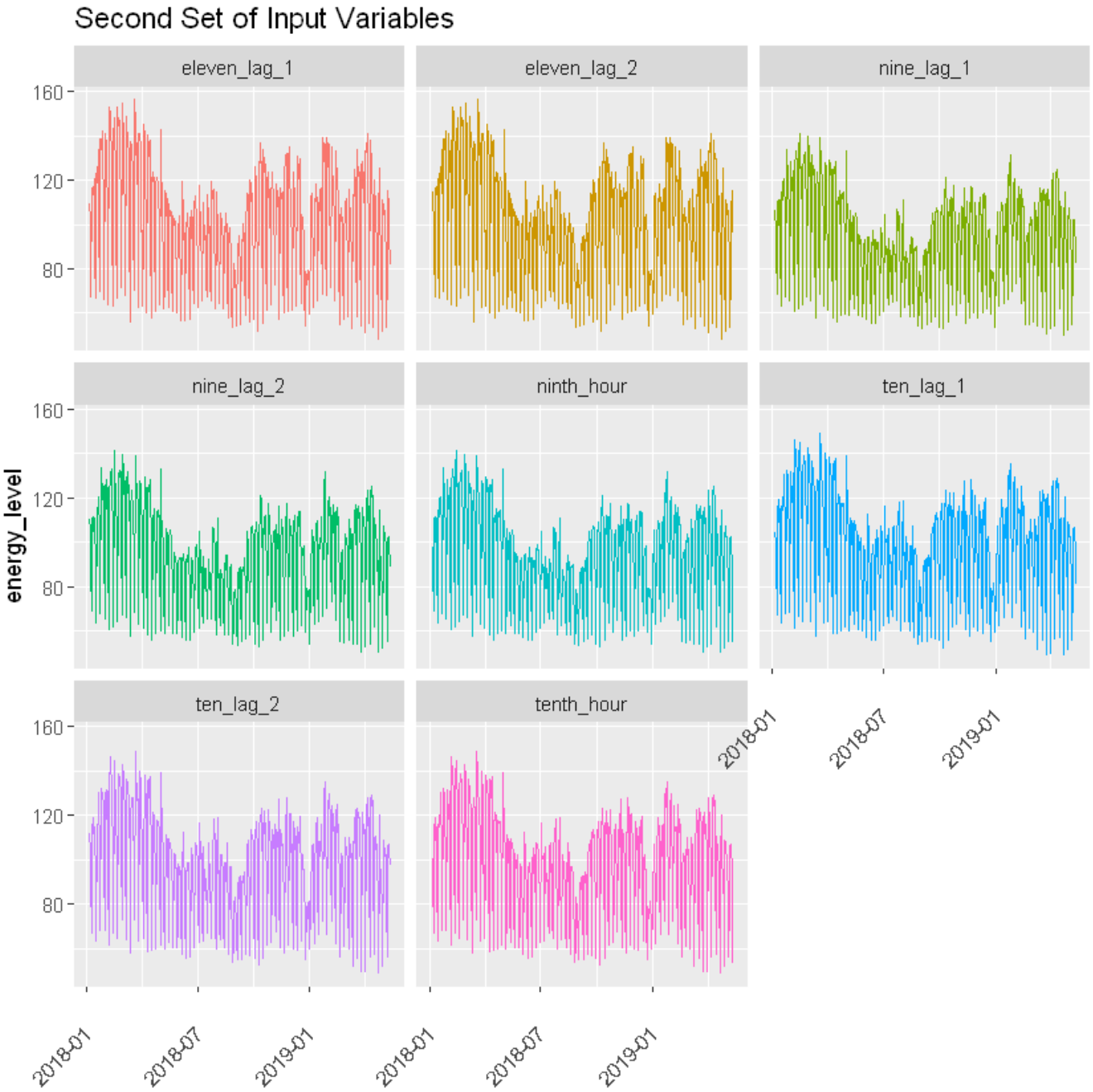
✓ 0.8s

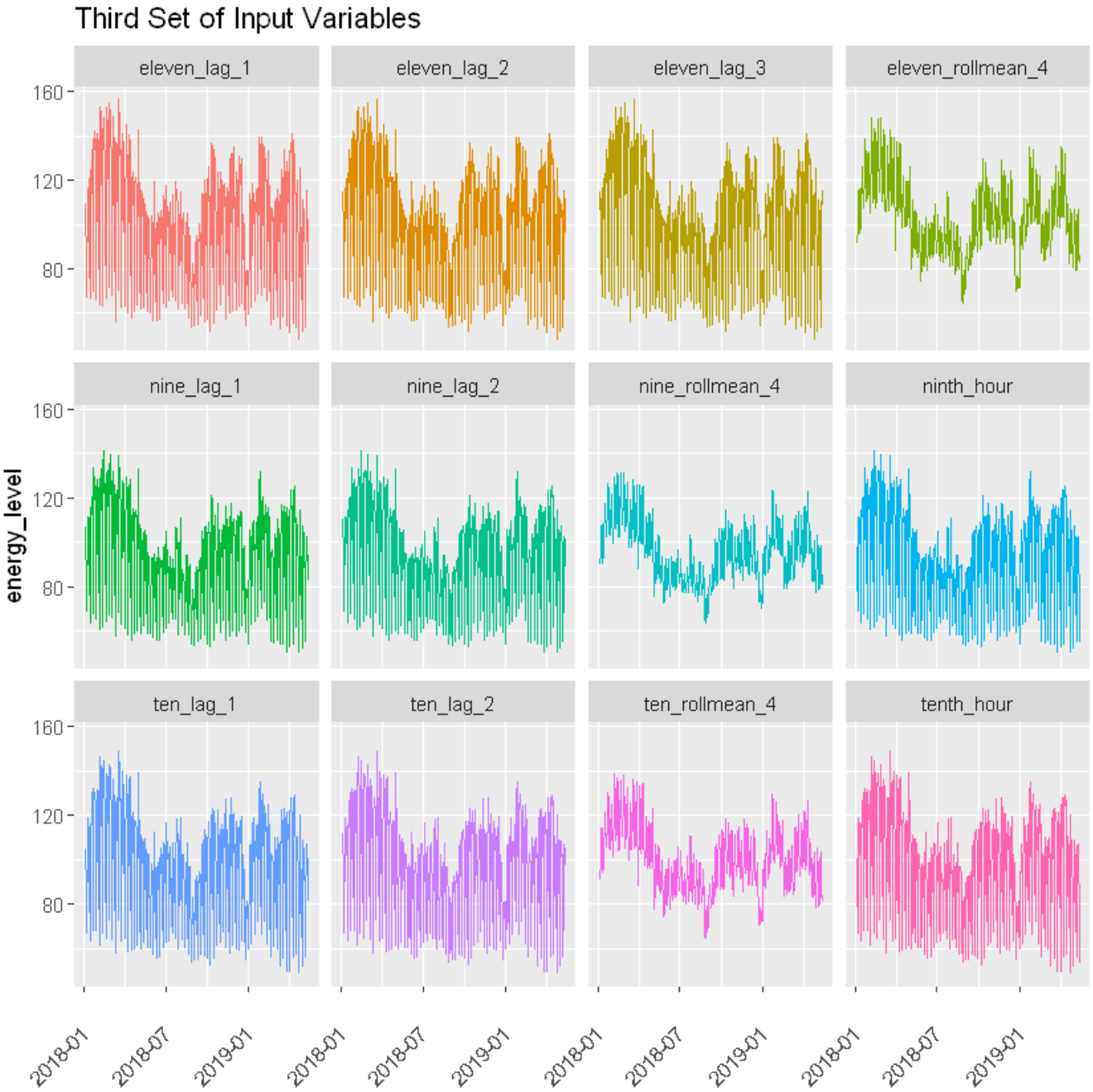
```r
#4thd lag
df_full %>%
pivot_longer(cols = c(13, 14, 15, 16, 17, 18), names_to = "kind", values_to = "energy_level") %>%
ggplot(aes(Dates, energy_level, color = kind)) +
geom_line() +
facet_wrap(~kind) + theme(axis.text.x = element_text(angle = 45, vjust =
0.5, hjust = 1
)) +
labs(x = "",
title = "Forth Set of Input Variables") +
theme(legend.position = "none")
```

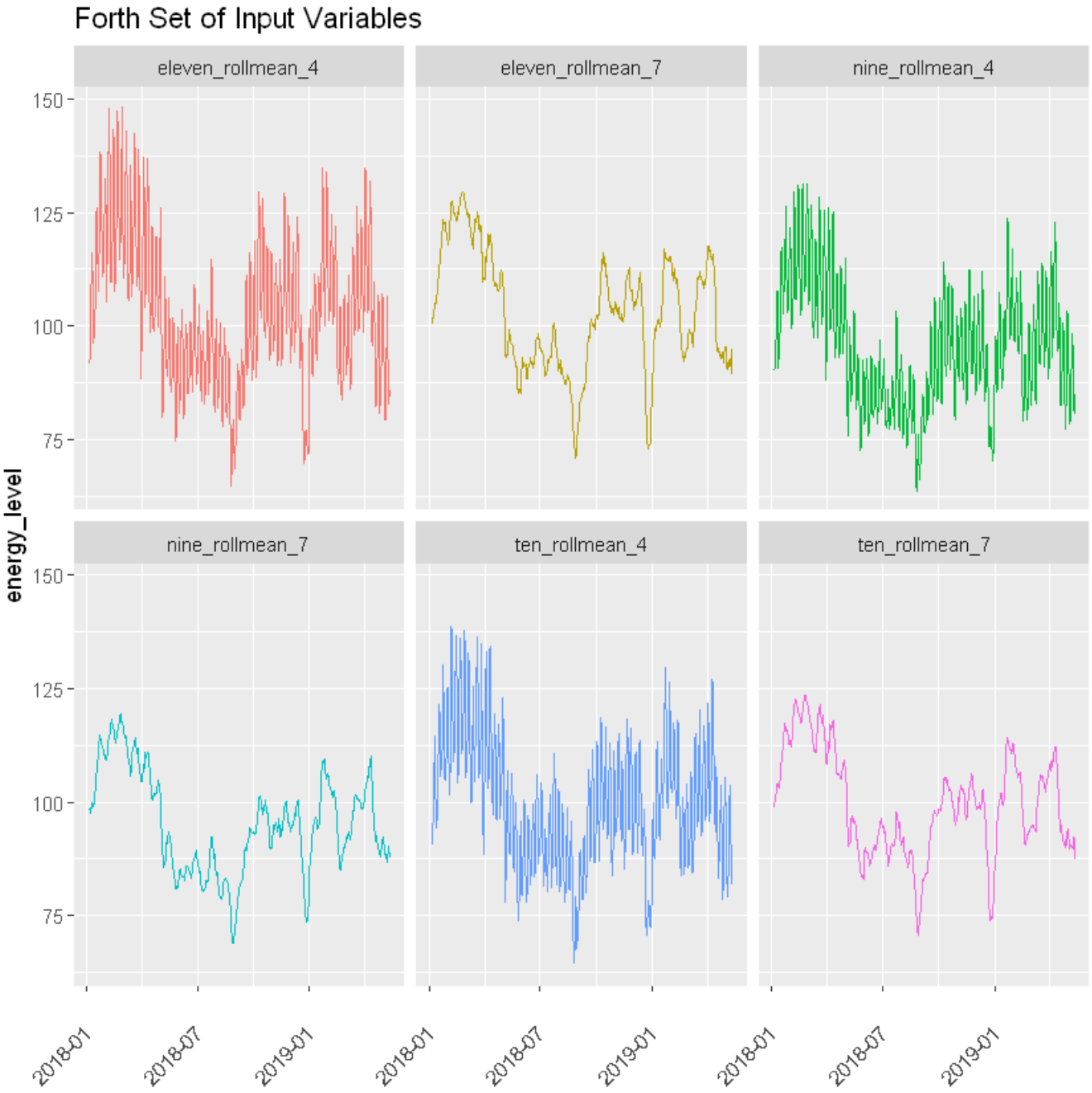
✓ 0.4s

Above code snippets used to visualize four different input sets used to experiments.


First Set of Input Variables
eleven_lag_1
nine_lag_1
ninth_hour
ten_lag_1
tenth_hour
energy_level
160
120
80
2018-01
2018-07
2019-01


Second Set of Input Variables
eleven_lag_1
eleven_lag_2
nine_lag_1
nine_lag_2
ninth_hour
ten_lag_1
ten_lag_2
tenth_hour
energy_level
160
120
80
2018-01
2018-07
2019-01


Third Set of Input Variables
eleven_lag_1
eleven_lag_2
eleven_lag_3
eleven_rollmean_4
nine_lag_1
nine_lag_2
nine_rollmean_4
ninth_hour
ten_lag_1
ten_lag_2
ten_rollmean_4
tenth_hour
energy_level
160
120
80
2018-01
2018-07
2019-01


Forth Set of Input Variables
eleven_rollmean_4
eleven_rollmean_7
nine_rollmean_4
nine_rollmean_7
ten_rollmean_4
ten_rollmean_7
energy_level
150
125
100
75
2018-01
2018-07
2019-01

Data Normalization

Data Normalization is a rescaling of data from original range to so that all values are within the 0 and 1. In this use case we used min-max normalization.

```
     Dates              ninth_hour       tenth_hour     eleventh_hour
Min.   :2018-01-05   Min.   : 50.40   Min.   : 49.20   Min.   : 48.2
1st Qu.:2018-05-08   1st Qu.: 81.20   1st Qu.: 83.00   1st Qu.: 84.6
Median :2018-09-08   Median : 97.40   Median :102.80   Median :105.2
Mean   :2018-09-08   Mean   : 95.28   Mean   : 99.65   Mean   :102.9
3rd Qu.:2019-01-09   3rd Qu.:110.60   3rd Qu.:116.20   3rd Qu.:121.2
Max.   :2019-05-12   Max.   :141.20   Max.   :148.80   Max.   :156.4
  nine_lag_1       ten_lag_1        nine_lag_2       ten_lag_2
Min.   : 50.40   Min.   : 49.20   Min.   : 50.40   Min.   : 49.20
1st Qu.: 81.40   1st Qu.: 83.00   1st Qu.: 81.40   1st Qu.: 83.20
Median : 97.60   Median :102.80   Median : 97.60   Median :103.00
Mean   : 95.38   Mean   : 99.75   Mean   : 95.44   Mean   : 99.82
3rd Qu.:110.60   3rd Qu.:116.20   3rd Qu.:110.60   3rd Qu.:116.20
Max.   :141.20   Max.   :148.80   Max.   :141.20   Max.   :148.80
 eleven_lag_1    eleven_lag_2    eleven_lag_3    eleven_lag_4
Min.   : 48.2   Min.   : 48.2   Min.   : 48.2   Min.   : 48.2
1st Qu.: 84.8   1st Qu.: 85.2   1st Qu.: 85.2   1st Qu.: 85.2
Median :105.6   Median :105.6   Median :105.8   Median :105.6
Mean   :103.0   Mean   :103.0   Mean   :103.1   Mean   :103.0
3rd Qu.:121.2   3rd Qu.:121.2   3rd Qu.:121.2   3rd Qu.:121.2
Max.   :156.4   Max.   :156.4   Max.   :156.4   Max.   :156.4
nine_rollmean_4  nine_rollmean_7  ten_rollmean_4   ten_rollmean_7
Min.   : 63.65   Min.   : 68.83   Min.   : 64.55   Min.   : 70.63
1st Qu.: 84.40   1st Qu.: 86.83   1st Qu.: 88.45   1st Qu.: 90.94
Median : 93.70   Median : 94.43   Median : 98.30   Median : 98.97
Mean   : 95.31   Mean   : 95.36   Mean   : 99.68   Mean   : 99.72
3rd Qu.:104.50   3rd Qu.:102.69   3rd Qu.:109.25   3rd Qu.:107.80
Max.   :131.35   Max.   :119.37   Max.   :138.75   Max.   :123.66
eleven_rollmean_4 eleven_rollmean_7
Min.   : 64.60    Min.   : 70.83
1st Qu.: 91.25    1st Qu.: 93.17
```

Above figure shows summary of the dataset before normalizing.

min-max normalization

```r
min_max_normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }


unnormalize_eleventh_hour <- function(x) {
return( (max(df_full$eleventh_hour) - min(df_full$eleventh_hour))*x + min(df_full$eleventh_hour)) }
```

✓ 0.4s

```r
df_normalized <- as.data.frame(lapply(df_full[2:18], min_max_normalize))
head(df_normalized)
```

✓ 0.5s

Above figure shows normalize and unnormalize functions and applying normalization for the dataset. And let's see some summary of the data.

```
  ninth_hour       tenth_hour     eleventh_hour      nine_lag_1
Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
1st Qu.:0.3392   1st Qu.:0.3394   1st Qu.:0.3364   1st Qu.:0.3414
Median :0.5176   Median :0.5382   Median :0.5268   Median :0.5198
Mean   :0.4943   Mean   :0.5065   Mean   :0.5052   Mean   :0.4954
3rd Qu.:0.6630   3rd Qu.:0.6727   3rd Qu.:0.6747   3rd Qu.:0.6630
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
  ten_lag_1        nine_lag_2       ten_lag_2       eleven_lag_1
Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
1st Qu.:0.3394   1st Qu.:0.3414   1st Qu.:0.3414   1st Qu.:0.3383
Median :0.5382   Median :0.5198   Median :0.5402   Median :0.5305
Mean   :0.5076   Mean   :0.4960   Mean   :0.5082   Mean   :0.5063
3rd Qu.:0.6727   3rd Qu.:0.6630   3rd Qu.:0.6727   3rd Qu.:0.6747
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
 eleven_lag_2     eleven_lag_3     eleven_lag_4    nine_rollmean_4
Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
1st Qu.:0.3420   1st Qu.:0.3420   1st Qu.:0.3420   1st Qu.:0.3065
Median :0.5305   Median :0.5323   Median :0.5305   Median :0.4439
Mean   :0.5069   Mean   :0.5071   Mean   :0.5067   Mean   :0.4677
3rd Qu.:0.6747   3rd Qu.:0.6747   3rd Qu.:0.6747   3rd Qu.:0.6034
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
nine_rollmean_7  ten_rollmean_4   ten_rollmean_7   eleven_rollmean_4
Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
1st Qu.:0.3561   1st Qu.:0.3221   1st Qu.:0.3831   1st Qu.:0.3182
Median :0.5065   Median :0.4549   Median :0.5345   Median :0.4388
Mean   :0.5250   Mean   :0.4734   Mean   :0.5486   Mean   :0.4573
3rd Qu.:0.6699   3rd Qu.:0.6024   3rd Qu.:0.7010   3rd Qu.:0.5761
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
eleven_rollmean_7
Min.   :0.0000
1st Qu.:0.3798
Median :0.5367
```

All the values scaled into 1 and 0 range.

```
dim(df_normalized)
```
✓ 0.4s

```
493 · 17
```

```
set.seed(123)
df_train <- df_normalized[1:430,]
df_test <- df_normalized[431:493,]
```
✓ 0.4s

Above figure shows splitting dataset into training and testing data.

Function to experiment one hidden layer Neural Network Architecture.

```
model_one_hidden_layer = function(arg, hidden_first, activation_fun, learning_rate, test_data, model_type) {
    one_layer_nurelnet = neuralnet(arg,
    data=df_train,
    hidden = c(hidden_first),
    linear.output = TRUE,
    act.fct = activation_fun,
    learningrate = learning_rate,
    algorithm = "rprop+"
    )
    results = compute(one_layer_nurelnet, test_data)
    truth_column = df_full[431:493, 4]$eleventh_hour
    predicted_column =  unnormalize_eleventh_hour(results$net.result)[,1]

    results <- data.frame(
        model_type = model_type,
        hidden_layers = hidden_first,
        RMSE = rmse(truth_column,predicted_column),
        MAE = mae(truth_column, predicted_column),
        MAPE =mape(truth_column, predicted_column),
        act_function = activation_fun,
        learning_rate = learning_rate)
}
```
✓ 0.3s

Function to experiment two hidden layer Neural Network Architecture.

```
model_two_hidden_layer = function(arg, hidden_first, hidden_second, activation_fun, learning_rate, test_data, model_type) {
    two_layer_nurelnet = neuralnet(arg,
    data=df_train,
    hidden = c(hidden_first, hidden_second),
    linear.output = TRUE,
    act.fct = activation_fun,
    learningrate = learning_rate,
    algorithm = "rprop+"
    )
    results = compute(two_layer_nurelnet, test_data)
    truth_column = df_full[431:493, 4]$eleventh_hour
    predicted_column =  unnormalize_eleventh_hour(results$net.result)[,1]

    results <- data.frame(
        model_type = model_type,
        hidden_layers = paste0(hidden_first," and ",hidden_second),
        RMSE = rmse(truth_column,predicted_column),
        MAE = mae(truth_column, predicted_column),
        MAPE =mape(truth_column, predicted_column),
        act_function = activation_fun,
        learning_rate = learning_rate)
}
```

✓ 0.3s

By using above functions, we will be able to experiment with input vectors, hidden layer nodes, activation function and learning rate in each model architecture.

Standard Statistical Indices

RMSE – Root Mean Squared Error

Standard deviation of the residuals (Predictions, Errors)

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

MAE – Mean Absolute Error

Measures average magnitude of the errors in a set of forecasts.

$$\mathrm{MAE} = \frac{1}{n} \sum_{j=1}^{n} |y_j - \hat{y}_j|$$

MAPE – Mean Absolute Percentage Error

Measures how accurate a forecast system is.

$$\mathrm{M} = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

One-hidden Layer Neural Network Architecture

In this step we will experiment single layer model architecture with different input vectors and different nodes and select the best single layer model with minimum error.

First Input set and results

1st lag

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

✓ 0.3s

```
one_layer_neuralnet_first_inset_result = lapply(1:10, function(n){
    model_one_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1",
        n,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8)],
        "First Set of Input Variables"
                )
    })
kable(one_layer_neuralnet_first_inset_result[])
```

✓ 3.1s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	1	4.5104	3.176271	0.0322746	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	2	4.39635	3.015967	0.0298124	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	3	4.36997	2.962272	0.0289265	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	4	4.379195	2.978004	0.0292111	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	5	4.58012	3.205595	0.0324052	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	6	4.427314	3.006772	0.0295682	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	7	4.504263	3.123514	0.031588	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	8	4.244565	2.925624	0.0279183	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	9	4.464161	3.055865	0.0301807	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	10	4.378542	3.032207	0.0295543	logistic	0.01

Second Input set and results

2nd lag

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

✓ 0.5s

+ Code + Markdown

```
one_layer_neuralnet_second_inset_result = lapply(1:10, function(n){
    model_one_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_2+eleven_lag_2",
        n,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9)],
        "Second Set of Input Variables"
                )
    })
kable(one_layer_neuralnet_second_inset_result[])
```

✓ 4.2s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	1	4.432914	3.095127	0.0317683	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	2	4.418926	3.103756	0.0313264	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	3	4.35427	3.020036	0.0297098	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	4	4.286195	2.865158	0.0283621	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	5	4.369226	3.075439	0.0308539	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	6	4.405174	3.027871	0.029944	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	7	4.470024	3.059079	0.029763	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	8	4.301791	2.997294	0.0302934	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	9	4.451674	3.12495	0.0310511	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	10	4.648676	3.144336	0.0311912	logistic	0.01

Third input set and results

```
set.seed(12345)
```
✓ 0.4s

```
one_layer_neuralnet_third_inset_result = lapply(1:10, function(n){
    model_one_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_2+eleven_lag_2+eleven_lag_3+nine_rollmean_
        n,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables"
                )
    })
kable(one_layer_neuralnet_third_inset_result[])
```
✓ 11.2s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	1	3.395186	2.334113	0.0235285	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	2	3.338284	2.329975	0.024222	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	3	3.125335	2.21461	0.0212598	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	4	3.387892	2.350998	0.0226527	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	5	3.043639	2.150009	0.021059	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	6	3.326778	2.519535	0.0257818	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	7	3.474811	2.438477	0.023838	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	8	3.362204	2.475763	0.0245321	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	9	3.11496	2.298332	0.0228325	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	10	3.265343	2.405955	0.024098	logistic	0.01

Forth input set and results

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

✓ 0.4s

```
one_layer_neuralnet_forth_inset_result = lapply(1:10, function(n){
    model_one_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4+nine_rollmean_7+ten_rollmean_7+eleven_rollmean_7",
        n,
        "logistic",
        0.01,
        df_test[, c(1, 2, 12, 14, 16, 13, 15, 17)],
        "Forth Set of Input Variables"
                )
    })
kable(one_layer_neuralnet_forth_inset_result[])
```

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	1	3.72896	2.586725	0.0263653	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	2	3.599836	2.430677	0.0235381	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	3	3.425955	2.387245	0.0232594	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	4	3.776237	2.646182	0.0262935	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	5	3.639989	2.509691	0.0245215	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	6	3.48725	2.352038	0.0222233	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	7	3.574555	2.415732	0.0234114	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	8	3.558247	2.505562	0.0236753	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	9	3.578875	2.490021	0.023764	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	10	3.514936	2.403229	0.0223679	logistic	0.01

Models with minimum error in each input set

```
kable(one_layer_neuralnet_first_inset_result[8])
```

✓ 0.5s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables	8	4.244565	2.925624	0.0279183	logistic	0.01

```
kable(one_layer_neuralnet_second_inset_result[4])
```

✓ 0.4s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables	4	4.286195	2.865158	0.0283621	logistic	0.01

```
kable(one_layer_neuralnet_third_inset_result[5])
```

✓ 0.4s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	5	3.043639	2.150009	0.021059	logistic	0.01

```
kable(one_layer_neuralnet_forth_inset_result[3])
```

✓ 0.1s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables	3	3.425955	2.387245	0.0232594	logistic	0.01

Among above network architectures network with 5 nodes in hidden layer and Third input set shows the minimum error, below figure shows stats of that model

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables	5	3.043639	2.150009	0.021059	logistic	0.01

Two-hidden Layer Neural Network Architecture

In this chapter we will experiment with two-hidden layer neural network architecture with different input sets and different node counts and select the best two-layer model with minimum error.

First input set and results

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

✓ 0.6s

```
two_layer_neuralnet_first_inset_result = lapply(1:10, function(n){
    lapply(1:10, function(m){
    model_two_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1",
        n,
        m,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8)],
        "First Set of Input Variables with 2 hidden layers"
                )
    })})
kable(two_layer_neuralnet_first_inset_result[])
```

✓ 45.3s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	1 and 1	4.495325	3.126883	0.0321828	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	1 and 2	25.54125	20.50023	0.2505597	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	1 and 3	4.399912	3.001625	0.0297157	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	1 and 4	4.386047	2.979411	0.0292981	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	1 and 5	4.428775	3.052829	0.0311269	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	1 and 6	4.471972	3.106588	0.0303767	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	1 and 7	4.509303	3.129918	0.0305834	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	2 and 2	4.524093	3.136029	0.0318566	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	2 and 3	4.379628	2.98057	0.0298481	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	2 and 4	4.651242	3.268468	0.0327373	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	2 and 5	4.466043	3.076936	0.0315397	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	2 and 6	4.565754	3.156658	0.0316099	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	2 and 7	4.469876	3.062968	0.0315926	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	2 and 8	4.487002	3.151517	0.0320446	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	2 and 9	4.437232	3.066956	0.0307039	logistic	0.01
First Set of Input Variables with 2 hidden layers	4 and 9	4.394357	3.007319	0.0297135	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	4 and 10	4.372432	2.969948	0.0296587	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	5 and 1	4.421312	3.037473	0.0307711	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	5 and 2	4.509233	3.148808	0.0313767	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	5 and 3	4.536846	3.189089	0.0324225	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	5 and 4	4.524325	3.134583	0.0321854	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	5 and 5	4.470579	3.089881	0.0316853	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	5 and 6	4.545698	3.209427	0.0311228	logistic	0.01

Second input set and results

```
set.seed(12345)
```
✓ 0.4s

```
two_layer_neuralnet_second_inset_result = lapply(1:10, function(n){
    lapply(1:10, function(m){
    model_two_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_2+eleven_lag_2",
        n,
        m,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9)],
        "Second Set of Input Variables with 2 hidden layers"
                )
    })})
kable(two_layer_neuralnet_second_inset_result[])
```
✓ 1m 4.3s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 1	4.442925	3.120988	0.0319387	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 2	4.408017	3.092861	0.0319644	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 3	4.365224	3.012984	0.0304585	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 4	4.399106	3.064805	0.0307981	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 5	4.431284	3.058461	0.0311178	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 6	4.535403	3.237048	0.0336659	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 7	4.530493	3.222691	0.0329467	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 8	4.407717	3.047318	0.0309791	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 9	4.44514	3.098449	0.0317513	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	1 and 10	4.551805	3.213519	0.0332632	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	2 and 1	4.499523	3.216915	0.0332864	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate

Third input set and results

```
set.seed(12345)
```
✓ 0.4s

```
two_layer_neuralnet_third_inset_result = lapply(1:10, function(n){
    lapply(1:10, function(m){
    model_two_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        n,
        m,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables with 2 hidden layers"
                )
    })})
kable(two_layer_neuralnet_third_inset_result[])
```
✓ 1m 51.8s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 1	3.446566	2.524328	0.0268599	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 2	3.408947	2.446085	0.0255018	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 3	3.275021	2.180412	0.0209026	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 4	3.504066	2.310956	0.0221102	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 5	3.471831	2.467659	0.0254582	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 6	3.25311	2.137238	0.0202343	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 7	3.399874	2.334103	0.0237902	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 8	3.338015	2.242546	0.0220136	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	1 and 9	3.498391	2.414193	0.024312	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	2 and 6	3.331727	2.264183	0.0221895	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	2 and 7	3.405624	2.307542	0.023045	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	2 and 8	3.217169	2.286906	0.0219925	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	2 and 9	3.496078	2.351956	0.0232312	logistic	0.01

| model_type | hidden_layers | RMSE | MAE | MAPE | act_function | learning_rate |

Forth input set and results

```
set.seed(12345)
```
✓ 0.6s

```
two_layer_neuralnet_forth_inset_result = lapply(1:10, function(n){
 lapply(1:10, function(m){
    model_two_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4+nine_rollmean_7+ten_rollmean_7+eleven_rollmean_7",
        n,
        m,
        "logistic",
        0.01,
        df_test[, c(1, 2, 12, 14, 16, 13, 15, 17)],
        "Forth Set of Input Variables with 2 hidden layers"
                )
    })})
kable(two_layer_neuralnet_forth_inset_result[])
```
✓ 1m 12.2s

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 1	3.726057	2.658716	0.027546	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 2	3.642796	2.467127	0.0240304	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 3	3.776601	2.640996	0.0272207	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 4	3.619335	2.507705	0.0240009	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 5	3.795984	2.692843	0.0277233	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 6	3.864565	2.702259	0.0277144	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 7	3.782165	2.779901	0.0291523	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 8	3.646151	2.512179	0.0250328	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	1 and 9	3.765839	2.602345	0.0262039	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	7 and 9	3.798739	2.786311	0.0271761	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	7 and 10	3.80266	2.540158	0.0245071	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	8 and 1	3.895097	2.838793	0.0294009	logistic	0.01

Models with minimum error in each input set

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
First Set of Input Variables with 2 hidden layers	8 and 2	4.311895	2.861946	0.0280006	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Second Set of Input Variables with 2 hidden layers	4 and 6	4.219101	2.824922	0.0290412	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	2.994381	2.26461	0.0222859	logistic	0.01

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Forth Set of Input Variables with 2 hidden layers	3 and 7	3.422755	2.311953	0.0216323	logistic	0.01

From above network architectures neural network with 10 and 3 in each hidden layer and third input set gave the minimum error and selected as the best architecture in two-hidden layer models.

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	2.994381	2.26461	0.0222859	logistic	0.01

Best Models in one-hidden layer and two-hidden layer architecture

One-hidden layer: Third input set and 5 nodes

Two-hidden layer: Third input set and 10 and 3 nodes in each hidden layer

Experimenting with Learning rate

In Machine learning learning rate is tuning parameter in an optimization algorithm that helps to minimize the loss in loss function.

```
l_rate=0.001
set.seed(12345)
for (i in 1:100) {
two_layer_neuralnet_third_inset_result = model_one_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        5,
        "logistic",
        l_rate,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables with 2 hidden layers"
                )

    print(kable(two_layer_neuralnet_third_inset_result[]))
    l_rate=l_rate+0.001
}
```

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	5	3.254631	2.330846	0.0225891	logistic	0.001

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	5	3.234071	2.442638	0.0240066	logistic	0.002

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	5	3.260642	2.242648	0.0216144	logistic	0.003

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	5	3.348682	2.344693	0.0233535	logistic	0.004

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	5	3.115083	2.11329	0.0204552	logistic	0.005

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	5	3.331232	2.400929	0.024372	logistic	0.006

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	5	3.384654	2.273253	0.0219598	logistic	0.007

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate

```
l_rate=0.001
set.seed(12345)
for (i in 1:100) {
two_layer_neuralnet_third_inset_result = model_two_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        10,
        3,
        "logistic",
        l_rate,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables with 2 hidden layers"
                )

    print(kable(two_layer_neuralnet_third_inset_result[]))
    l_rate=l_rate+0.001
}
```

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	3.926296	2.902561	0.0295836	logistic	0.001

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	3.396795	2.454123	0.024426	logistic	0.002

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	3.270112	2.346771	0.0231808	logistic	0.003

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	3.420688	2.506423	0.0238728	logistic	0.004

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	3.49989	2.587197	0.0247294	logistic	0.005

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	3.396263	2.660475	0.0271747	logistic	0.006

model_type	hidden_layers	RMSE	MAE	MAPE	act_function	learning_rate
Third Set of Input Variables with 2 hidden layers	10 and 3	3.330931	2.57906	0.0273486	logistic	0.007

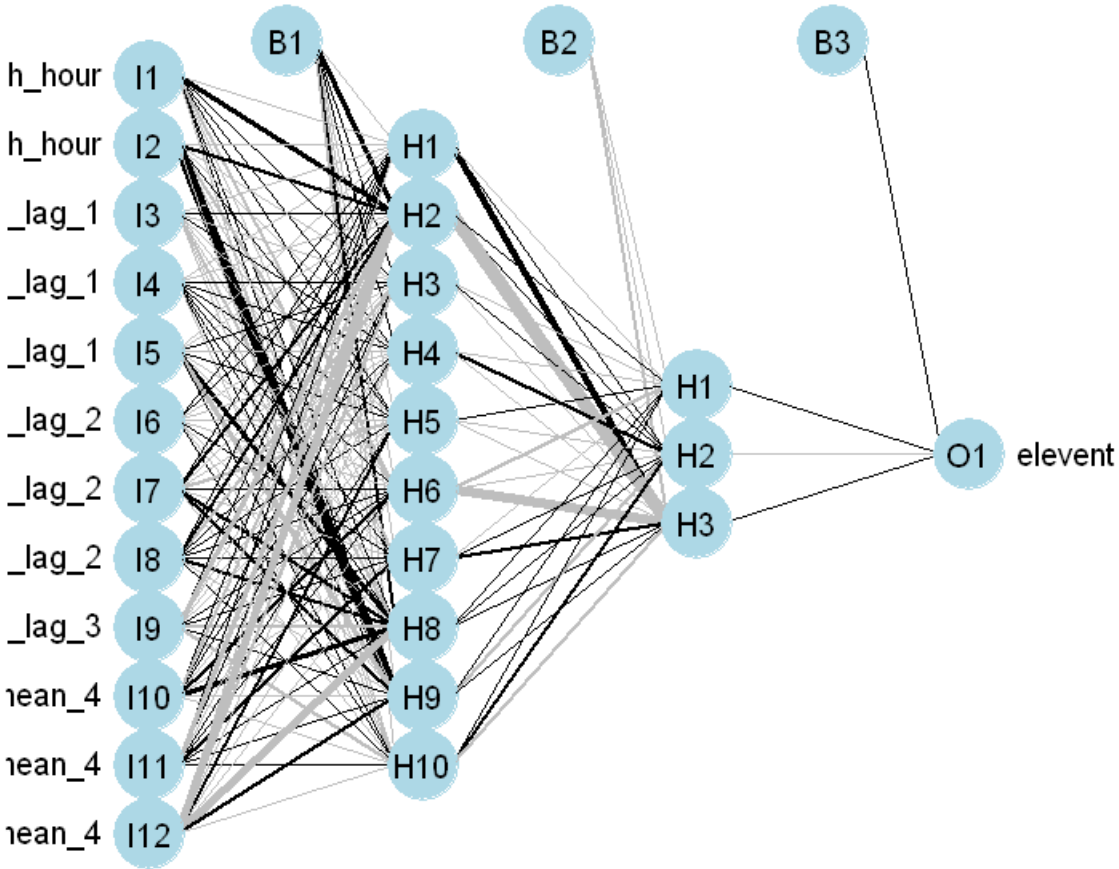
From above one-hidden layer and two-hidden layer approach minimum error given by two-hidden layer network architecture with learning rate = 0.037 and one-hidden layer minimum learning rate = 0.087 with comparing with these both models best model was two-hidden layer architecture which gave high performance with evaluation statistical indices (RMSE, MAE, MAPE)

Prediction with best MLP architecture

```r
best_nn = neuralnet(
    "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
    data=df_train,
    hidden = c(10, 3),
    linear.output = TRUE,
    act.fct = "logistic",
    learningrate = 0.037,
    algorithm = "rprop+"
    )
results = compute(best_nn, df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)])
truth_column = df_full[431:493, 4]$eleventh_hour
predicted_column =  unnormalize_eleventh_hour(results$net.result)[,1]
results_df <- data.frame(
    date = df_full[431:493,]$Dates,
    actual = truth_column,
    predicted = predicted_column)
```

Above figure shows best network construction code and predictions using test dataset.

Plot MLP



Predicted vs actual from best MLP

```
results_df
```

✓ 0.1s

A data.frame: 63 × 3

	date	actual	predicted
	<date>	<dbl>	<dbl>
431	2019-03-11	113.6	110.22966
432	2019-03-12	100.8	111.74400
433	2019-03-13	124.0	119.69028
434	2019-03-14	124.8	124.87102
435	2019-03-15	119.0	118.09625
436	2019-03-16	89.4	90.07865
437	2019-03-17	58.2	57.76734
438	2019-03-18	133.8	132.00848
439	2019-03-19	124.0	126.24820
440	2019-03-20	126.2	125.18579
441	2019-03-21	121.8	124.28905
442	2019-03-22	123.8	121.52204
443	2019-03-23	95.6	92.45947
444	2019-03-24	54.2	54.77366
445	2019-03-25	125.8	125.77129
446	2019-03-26	128.2	127.99020
447	2019-03-27	116.2	118.60036
448	2019-03-28	123.8	122.21853

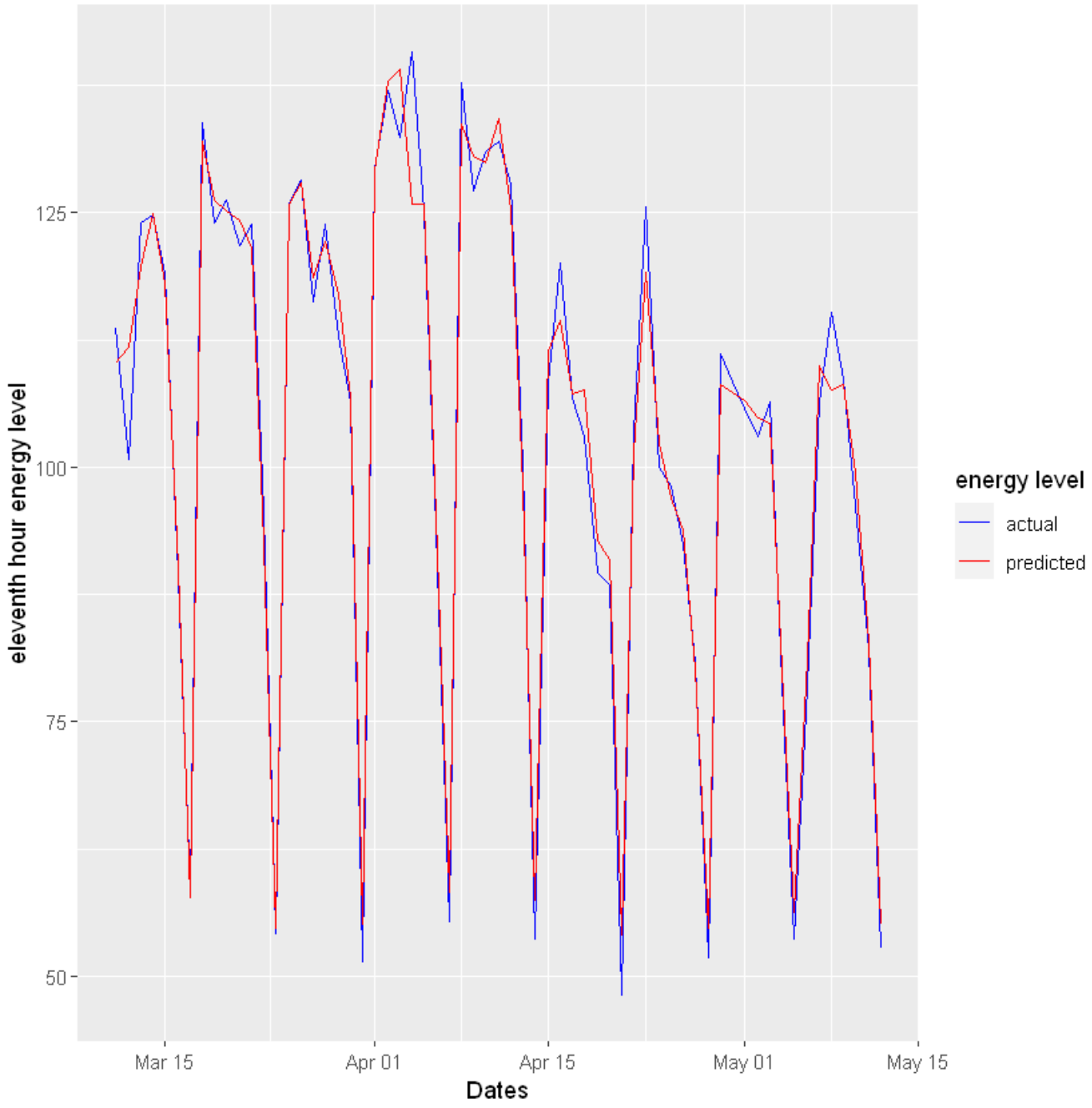
471	2019-04-20	88.4	90.97972
472	2019-04-21	48.2	54.03446
473	2019-04-22	103.4	101.38499
474	2019-04-23	125.6	119.13158
475	2019-04-24	100.0	102.40706
476	2019-04-25	98.2	96.98749
477	2019-04-26	92.4	93.83718
478	2019-04-27	80.8	80.11707
479	2019-04-28	51.8	54.68223
480	2019-04-29	111.2	108.05487
481	2019-04-30	108.4	107.38372
482	2019-05-01	105.8	106.55681
483	2019-05-02	103.0	104.78424
484	2019-05-03	106.4	104.34848
485	2019-05-04	78.8	80.15091
486	2019-05-05	53.6	56.29922
487	2019-05-06	78.4	81.21166
488	2019-05-07	106.4	109.94384
489	2019-05-08	115.2	107.55009
490	2019-05-09	108.6	108.06563
491	2019-05-10	96.2	99.39947
492	2019-05-11	82.0	82.74692
493	2019-05-12	52.8	55.09607

```
summary(results_df)
```
✓ 0.5s

```
      date                actual          predicted     
 Min.   :2019-03-11   Min.   : 48.2   Min.   : 54.03  
 1st Qu.:2019-03-26   1st Qu.: 91.0   1st Qu.: 92.58  
 Median :2019-04-11   Median :106.8   Median :107.59  
 Mean   :2019-04-11   Mean   :103.5   Mean   :103.83  
 3rd Qu.:2019-04-26   3rd Qu.:124.4   3rd Qu.:124.58  
 Max.   :2019-05-12   Max.   :140.8   Max.   :139.07  
```

Plotting predicted vs actual result data frame

```
ggplot() +
  geom_line(data = results_df, aes(y = actual, x = date, colour = "actual")) +
  geom_line(data = results_df, aes(y = predicted, x = date, colour = "predicted")) +
  scale_color_manual(name = "energy level", values = c("actual" = "blue", "predicted" = "red")) +
  xlab('Dates') +
  ylab('eleventh hour energy level')
```
✓ 0.2s



We can do some more optimizations to increase model forecasting accuracy.

Appendix

Clustering

```
# libraries
library("readxl")
options(warn = 0)
library(dplyr)
library(ggplot2)
library(caTools)
library(caret)
library(GGally)
library(janitor)
library(corrplot)
library(tidyverse)
library(hrbrthemes)
library(viridis)
library(NbClust)
library(factoextra)
library(gridExtra)

df <- read_excel("data/Whitewine_v2.xlsx")
#add index column to data frame
df$index <- 1:nrow(df)
head(df)

# replacing each quality measures into 1-4 range
# 1 wrost and 4 is best
df["quality"][df["quality"] == 5] <- 1
df["quality"][df["quality"] == 6] <- 2
df["quality"][df["quality"] == 7] <- 3
df["quality"][df["quality"] == 8] <- 4

head(df)

boxplot(df[1:11])

# format columns
df <- janitor::clean_names(df)

#EDA
summary(df)

#check null values
sum(is.na(df))

#Correlation Heatmap of Variables
corrplot(cor(df[1:12]))

z_score = function(x) {
  return((x - mean(x)) / sd(x))
}
```

```r
dfNorm <- as.data.frame(lapply(df[1:11], z_score))
dfNorm$index <- 1:nrow(dfNorm)
head(dfNorm)

summary(dfNorm[1:11])

#only keep rows in dataframe with all z-scores less than absolute value of 3
no_outliers <- dfNorm[!rowSums(dfNorm[1:11]>3),]

head(no_outliers)

# merge normalized one ane orginal quality
# merge two data frames by ID
df_final <- merge(no_outliers,df[12:13],by="index")

head(df_final)

summary(df_final[2:12])

histogram = function(x,title,x_label) {
  # Add a Normal Curve (Thanks to Peter Dalgaard)
  x <- x
  h<-hist(x, breaks=10, col="red", xlab=x_label,
    main=title)
  xfit<-seq(min(x),max(x),length=40)
  yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
  yfit <- yfit*diff(h$mids[1:2])*length(x)
  lines(xfit, yfit, col="blue", lwd=2)
}

# histogram(df$fixed_acidity,"fixed_acidity values before preprocessing","fixed_acidity
value")
# histogram(df_final$fixed_acidity,"fixed_acidity values after
preprocessing","fixed_acidity value")
histogram(df$quality,"quality values before preprocessing","fixed_acidity value")
histogram(df_final$quality,"quality values after preprocessing","fixed_acidity value")

head(df_final)

# see the datasrt types
str(df_final)

df_nb <- df_final[2:12]

#setting seed point
set.seed(26)

# #euclidient distance
no_of_clusters_eud = NbClust(df_nb,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")
# #manhatten
no_of_clusters_man = NbClust(df_nb,distance="manhattan",
min.nc=2,max.nc=15,method="kmeans",index="all")
# #maximum
```

```r
no_of_clusters_maximum = NbClust(df_nb,distance="maximum",
min.nc=2,max.nc=15,method="kmeans",index="all")

fviz_nbclust(df_final[2:12],kmeans,method="wss")+geom_vline(xintercept=4,linetype=1)


k = 1:10
set.seed(25)
WSS = sapply(k, function(k) {kmeans(df_final[2:12], centers=k)$tot.withinss})

# You can then use a line plot to plot the within sum of squares with a different number
of k
plot(k, WSS, type="b", xlab= "Number of k", ylab="Within sum of squares")

# silhouette
fviz_nbclust(df_final[2:12], kmeans, method = "silhouette") + labs(subtitle = "Silhoutte
Method")


# clustering
head(df_final)

x=df_final[2:12]
y=df_final$quality

kc_2 <- kmeans(x,centers=2)
kc_2
str(kc_2)
table(y,kc_2$cluster)
# implement the metrix
confusionMatrix(

 factor(kc_2$cluster, levels = 1:4),

 factor(df_final$quality,levels=1:4)
)
fviz_cluster(kc_2,data=x)


kc_3 <- kmeans(x,centers=3)
kc_3
str(kc_3)
table(y,kc_3$cluster)
# implement the metrix
confusionMatrix(

 factor(kc_3$cluster, levels = 1:4),

 factor(df_final$quality,levels=1:4)
)
fviz_cluster(kc_3,data=x)


kc_4 <- kmeans(x,centers=4)
```

```r
kc_4
str(kc_4)
table(y,kc_4$cluster)
# implement the metrix
confusionMatrix(

 factor(kc_4$cluster, levels = 1:4),

 factor(df_final$quality,levels=1:4)
)
fviz_cluster(kc_4,data=x)


# plots to compare
p1 <- fviz_cluster(kc_2, geom = "point", data = x) + ggtitle("k = 2")
p2 <- fviz_cluster(kc_3, geom = "point",  data = x) + ggtitle("k = 3")
p3 <- fviz_cluster(kc_4, geom = "point",  data = x) + ggtitle("k = 4")
grid.arrange(p1, p2, p3,nrow = 2)


head(df_final)

principle_components <- prcomp(df_final[2:12])
principle_components
summary(principle_components)

plot(principle_components)
plot(principle_components, type = "l")
biplot(principle_components)
biplot(principle_components, scale = 0)

str(principle_components)

df_final_with_pcs <- cbind(df_final, principle_components$x)

head(df_final_with_pcs)

library(ggplot2)

ggplot(df_final_with_pcs, aes(PC9,PC10,PC11,col = quality, fill = quality)) +
  stat_ellipse(geom = "polygon", col = "black", alpha = 0.5) +
  geom_point(shape = 21, col = "black")

x_2 = df_final_with_pcs[22:24]
y_2 = df_final_with_pcs$quality
kcpca <- kmeans(x_2,4)
kcpca
fviz_cluster(kcpca,data=x_2)
table(y_2,kcpca$cluster)
```

Forecasting

```
# libraries
library(readxl)
options(warn = 0)
library(tidyverse)
library(readxl)
library(lubridate)
library(zoo)
library(readxl)
library(neuralnet)
library(knitr)
library(Metrics)
library(NeuralNetTools)

df <- read_excel("data/UOW_load.xlsx")
head(df)

df$Dates<-as.Date(df$Dates)
names(df)[2] <- paste("ninth_hour")
names(df)[3] <- paste("tenth_hour")
names(df)[4] <- paste("eleventh_hour")
head(df)
summary(df[2:4])

# insert all the inputs to the one data frame
# creating revious time frame values for autoregressive model
df_full = df %>%
 mutate(
    nine_lag_1 = lag(df$ninth_hour,1),
    ten_lag_1 = lag(df$tenth_hour,1),
    nine_lag_2 = lag(df$ninth_hour,2),
    ten_lag_2 = lag(df$tenth_hour,2),
    eleven_lag_1 = lag(df$eleventh_hour,1),
    eleven_lag_2 = lag(df$eleventh_hour,2),
    eleven_lag_3 = lag(df$eleventh_hour,3),
    eleven_lag_4 = lag(df$eleventh_hour,4),
    #roll mean
    nine_rollmean_4 = rollmean(ninth_hour,4, fill = NA),
    nine_rollmean_7 = rollmean(ninth_hour,7, fill = NA),
    ten_rollmean_4 = rollmean(tenth_hour,4, fill = NA),
    ten_rollmean_7 = rollmean(tenth_hour,7, fill = NA),
    eleven_rollmean_4 = rollmean(eleventh_hour,4, fill = NA),
    eleven_rollmean_7 = rollmean(eleventh_hour,7, fill = NA)) %>%
#Drop null coulumns
drop_na()


head(df_full)
#1st lag
df_full %>%
pivot_longer(cols = c(2, 3, 5, 6, 9), names_to = "kind", values_to = "energy_level") %>%
ggplot(aes(Dates, energy_level, color = kind)) +
```

```
geom_line() +
facet_wrap(~kind) + theme(axis.text.x = element_text(angle = 45, vjust =
0.5, hjust = 1
)) +
labs(x = "",
title = "First Set of Input Variables") +
theme(legend.position = "none")

#2nd lag
df_full %>%
pivot_longer(cols = c(2, 3, 5, 6, 9, 7, 8, 10), names_to = "kind", values_to =
"energy_level") %>%
ggplot(aes(Dates, energy_level, color = kind)) +
geom_line() +
facet_wrap(~kind) + theme(axis.text.x = element_text(angle = 45, vjust =
0.5, hjust = 1
)) +
labs(x = "",
title = "Second Set of Input Variables") +
theme(legend.position = "none")

#3rd lag
df_full %>%
pivot_longer(cols = c(2, 3, 5, 6, 9, 7, 8, 10, 11, 13, 15, 17), names_to = "kind",
values_to = "energy_level") %>%
ggplot(aes(Dates, energy_level, color = kind)) +
geom_line() +
facet_wrap(~kind) + theme(axis.text.x = element_text(angle = 45, vjust =
0.5, hjust = 1
)) +
labs(x = "",
title = "Third Set of Input Variables") +
theme(legend.position = "none")

#4thd lag
df_full %>%
pivot_longer(cols = c(13, 14, 15, 16, 17, 18), names_to = "kind", values_to =
"energy_level") %>%
ggplot(aes(Dates, energy_level, color = kind)) +
geom_line() +
facet_wrap(~kind) + theme(axis.text.x = element_text(angle = 45, vjust =
0.5, hjust = 1
)) +
labs(x = "",
title = "Forth Set of Input Variables") +
theme(legend.position = "none")


#normalization
head(df_full)
summary(df_full)

# min max normalize
min_max_normalize <- function(x) {
```

```
return ((x - min(x)) / (max(x) - min(x))) }


unnormalize_eleventh_hour <- function(x) {
return( (max(df_full$eleventh_hour) - min(df_full$eleventh_hour))*x +
min(df_full$eleventh_hour)) }

df_normalized <- as.data.frame(lapply(df_full[2:18], min_max_normalize))
head(df_normalized)
summary(df_normalized)
dim(df_normalized)

# train test split
set.seed(123)
df_train <- df_normalized[1:430,]
df_test <- df_normalized[431:493,]

# one-hidden layer function
model_one_hidden_layer = function(arg, hidden_first, activation_fun, learning_rate,
test_data, model_type) {
    one_layer_nurelnet = neuralnet(arg,
    data=df_train,
    hidden = c(hidden_first),
    linear.output = TRUE,
    act.fct = activation_fun,
    learningrate = learning_rate,
    algorithm = "rprop+"
    )
    results = compute(one_layer_nurelnet, test_data)
    truth_column = df_full[431:493, 4]$eleventh_hour
    predicted_column =  unnormalize_eleventh_hour(results$net.result)[,1]

    results <- data.frame(
        model_type = model_type,
        hidden_layers = hidden_first,
        RMSE = rmse(truth_column,predicted_column),
        MAE = mae(truth_column, predicted_column),
        MAPE =mape(truth_column, predicted_column),
        act_function = activation_fun,
        learning_rate = learning_rate)
}

#two-hidden layer funcation
model_two_hidden_layer = function(arg, hidden_first, hidden_second, activation_fun,
learning_rate, test_data, model_type) {
    two_layer_nurelnet = neuralnet(arg,
    data=df_train,
    hidden = c(hidden_first, hidden_second),
    linear.output = TRUE,
    act.fct = activation_fun,
    learningrate = learning_rate,
    algorithm = "rprop+"
    )
    results = compute(two_layer_nurelnet, test_data)
```

```
    truth_column = df_full[431:493, 4]$eleventh_hour
    predicted_column =  unnormalize_eleventh_hour(results$net.result)[,1]

    results <- data.frame(
        model_type = model_type,
        hidden_layers = paste0(hidden_first," and ",hidden_second),
        RMSE = rmse(truth_column,predicted_column),
        MAE = mae(truth_column, predicted_column),
        MAPE =mape(truth_column, predicted_column),
        act_function = activation_fun,
        learning_rate = learning_rate)
}


# one-hidden layer experiments
# firse input set
set.seed(12345)
one_layer_neuralnet_first_inset_result = lapply(1:10, function(n){
    model_one_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1",
        n,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8)],
        "First Set of Input Variables"
                        )
    })
kable(one_layer_neuralnet_first_inset_result[])

# second input set
set.seed(12345)
one_layer_neuralnet_second_inset_result = lapply(1:10, function(n){
    model_one_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2",
        n,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9)],
        "Second Set of Input Variables"
                        )
    })
kable(one_layer_neuralnet_second_inset_result[])

# third input set
set.seed(12345)
one_layer_neuralnet_third_inset_result = lapply(1:10, function(n){
    model_one_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        n,
        "logistic",
```

```
        0.01,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables"
                        )
    })
kable(one_layer_neuralnet_third_inset_result[])

# forth input set
set.seed(12345)
one_layer_neuralnet_forth_inset_result = lapply(1:10, function(n){
    model_one_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4+nine
_rollmean_7+ten_rollmean_7+eleven_rollmean_7",
        n,
        "logistic",
        0.01,
        df_test[, c(1, 2, 12, 14, 16, 13, 15, 17)],
        "Forth Set of Input Variables"
                        )
    })
kable(one_layer_neuralnet_forth_inset_result[])
kable(one_layer_neuralnet_first_inset_result[8])
kable(one_layer_neuralnet_second_inset_result[4])
kable(one_layer_neuralnet_third_inset_result[5])
kable(one_layer_neuralnet_forth_inset_result[3])

# best one-hidden layer
kable(one_layer_neuralnet_third_inset_result[5])


# two-hidden layer experiments
# 1st input set
set.seed(12345)
two_layer_neuralnet_first_inset_result = lapply(1:10, function(n){
    lapply(1:10, function(m){
    model_two_hidden_layer(
        "eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1",
        n,
        m,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8)],
        "First Set of Input Variables with 2 hidden layers"
                        )
    })})
kable(two_layer_neuralnet_first_inset_result[])

# 2nd input set
set.seed(12345)
two_layer_neuralnet_second_inset_result = lapply(1:10, function(n){
    lapply(1:10, function(m){
    model_two_hidden_layer(
```

```
"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2",
        n,
        m,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9)],
        "Second Set of Input Variables with 2 hidden layers"
                        )
    })})
kable(two_layer_neuralnet_second_inset_result[])

# 3rd input set
set.seed(12345)
two_layer_neuralnet_third_inset_result = lapply(1:10, function(n){
    lapply(1:10, function(m){
    model_two_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        n,
        m,
        "logistic",
        0.01,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables with 2 hidden layers"
                        )
    })})
kable(two_layer_neuralnet_third_inset_result[])

# 4th input set
set.seed(12345)
two_layer_neuralnet_forth_inset_result = lapply(1:10, function(n){
 lapply(1:10, function(m){
    model_two_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4+nine
_rollmean_7+ten_rollmean_7+eleven_rollmean_7",
        n,
        m,
        "logistic",
        0.01,
        df_test[, c(1, 2, 12, 14, 16, 13, 15, 17)],
        "Forth Set of Input Variables with 2 hidden layers"
                        )
    })})
kable(two_layer_neuralnet_forth_inset_result[])
kable(two_layer_neuralnet_first_inset_result[8])
# 8 and 2
kable(two_layer_neuralnet_second_inset_result[4])
# 4 and 6
kable(two_layer_neuralnet_third_inset_result[10])
# 10 and 3
```

```
kable(two_layer_neuralnet_forth_inset_result[3])
# 3 and 7

# learning rate experiments
l_rate=0.001
set.seed(12345)
for (i in 1:100) {
two_layer_neuralnet_third_inset_result = model_two_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        10,
        3,
        "logistic",
        l_rate,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables with 2 hidden layers"
                        )

    print(kable(two_layer_neuralnet_third_inset_result[]))
    l_rate=l_rate+0.001
}
l_rate=0.001
set.seed(12345)
for (i in 1:100) {
two_layer_neuralnet_third_inset_result = model_one_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        5,
        "logistic",
        l_rate,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables with 2 hidden layers"
                        )

    print(kable(two_layer_neuralnet_third_inset_result[]))
    l_rate=l_rate+0.001
}

# final two-hidden layer
final_two_layer = model_two_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        10,
        3,
        "logistic",
        0.037,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables with 2 hidden layers"
                        )

    print(kable(final_two_layer[]))
```

```r
# final one-hidden layer
final_single_layer = model_one_hidden_layer(

"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
        5,
        "logistic",
        0.087,
        df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)],
        "Third Set of Input Variables with 2 hidden layers"
                        )

    print(kable(final_single_layer[]))


# construct best network and predict
best_nn = neuralnet(

"eleventh_hour~ninth_hour+tenth_hour+nine_lag_1+ten_lag_1+eleven_lag_1+nine_lag_2+ten_lag_
2+eleven_lag_2+eleven_lag_3+nine_rollmean_4+ten_rollmean_4+eleven_rollmean_4",
    data=df_train,
    hidden = c(10, 3),
    linear.output = TRUE,
    act.fct = "logistic",
    learningrate = 0.037,
    algorithm = "rprop+"
    )
results = compute(best_nn, df_test[, c(1, 2, 4, 5, 8, 6, 7, 9, 10, 12, 14, 16)])
truth_column = df_full[431:493, 4]$eleventh_hour
predicted_column =  unnormalize_eleventh_hour(results$net.result)[,1]
results_df <- data.frame(
    date = df_full[431:493,]$Dates,
    actual = truth_column,
    predicted = predicted_column)
plotnet(best_nn)
rmse(truth_column,predicted_column)
results_df
summary(results_df)


# plot actual vs predicted
ggplot() +
  geom_line(data = results_df, aes(y = actual, x = date, colour = "actual")) +
  geom_line(data = results_df, aes(y = predicted, x = date, colour = "predicted")) +
  scale_color_manual(name = "energy level", values = c("actual" = "blue", "predicted" =
"red")) +
  xlab('Dates') +
  ylab('eleventh hour energy level')
```

References

Basaran, U. and Kurban, M., 2007. A New Approach for the Short-Term Load Forecasting with Autoregressive and Artificial Neural Network Models. *International Journal of Computational Intelligence Research*, 3(1).

Datacamp. 2022. *Neural Network Models in R*. [online] Available at: <https://www.datacamp.com/tutorial/neural-network-models-r> [Accessed 9 May 2022].

Uc-r.github.io. 2022. *K-means Cluster Analysis · UC Business Analytics R Programming Guide*. [online] Available at: <https://uc-r.github.io/kmeans_clustering> [Accessed 9 May 2022].