INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER (UOW)

BEng (Hons) in Software Engineering

-5DATA001C -

Machine Learning and Data Mining

Assignment title: Machine Learning and Data Mining – Coursework (2021/22)

Module Leader: Dr. V.S. Kontogiannis

Date of submission: 09/05/2022, 13:00

Student Name: Achintha Jayatilake

IIT Student ID: 2019530

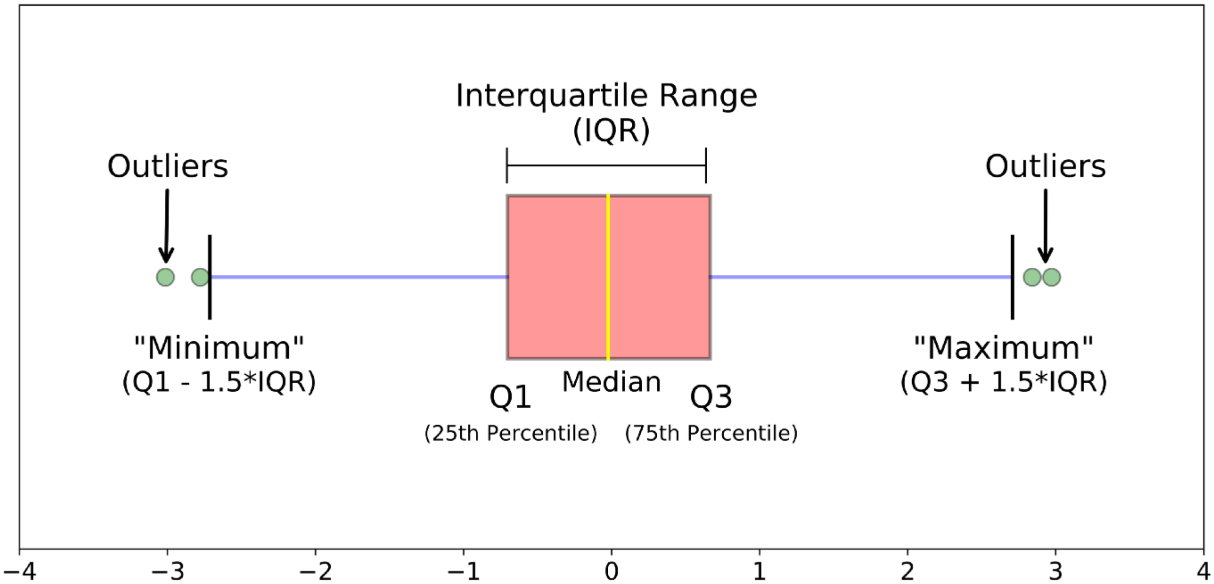
UOW No: w1761374

1st Objective (partitioning clustering)

• Pre-processing tasks

For the Pre- processing tasks I have used in here is Boxplot

So, for some distributions/datasets, you will discover that you require more information than the measures of central tendency (median, mean, and mode).



A boxplot is depicted in the figure above. A boxplot is a standardized method of depicting data distribution based on a five-number summary ("minimum," first quartile (Q1), median, third quartile (Q3), and "maximum"). It can provide information about your outliers and their values. It can also tell you if your data is symmetrical, how densely your data is clustered, and whether or not your data is skewed.

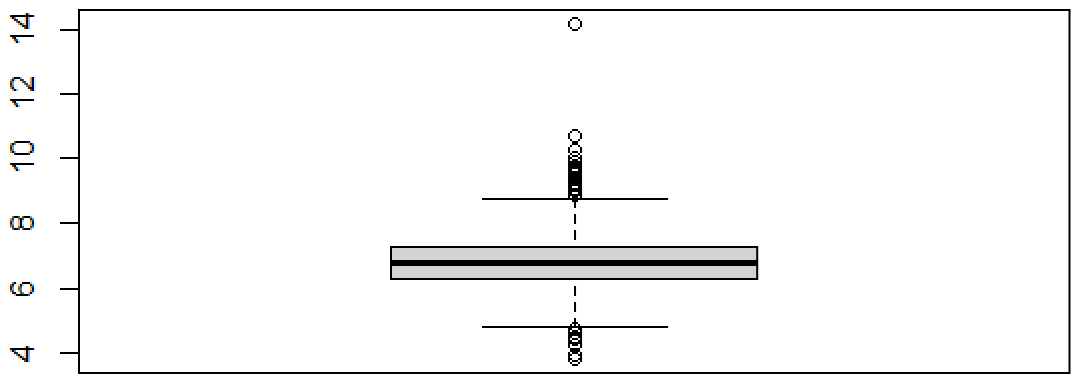
Comparing a boxplot to the probability density function for a normal distribution can help you understand its anatomy.

Scaling and Outliers removal

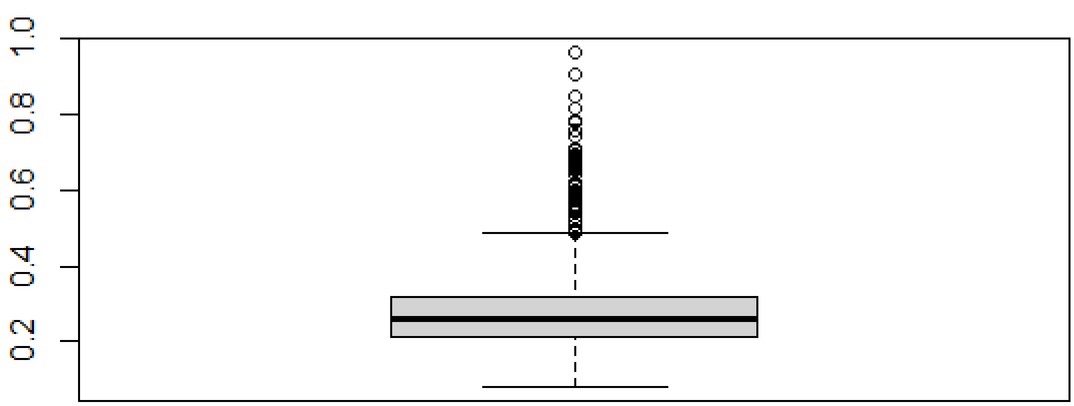
Standardizing is a popular scaling approach that subtracts the mean from values and divides by the standard deviation, resulting in a *conventional Gaussian probability distribution for an input variable (zero mean and unit variance). If the input variable contains outlier values, standardization can become skewed or prejudiced.*

To solve this, while standardizing numerical input variables, the median and interquartile range, also known as robust scaling, can be utilized.

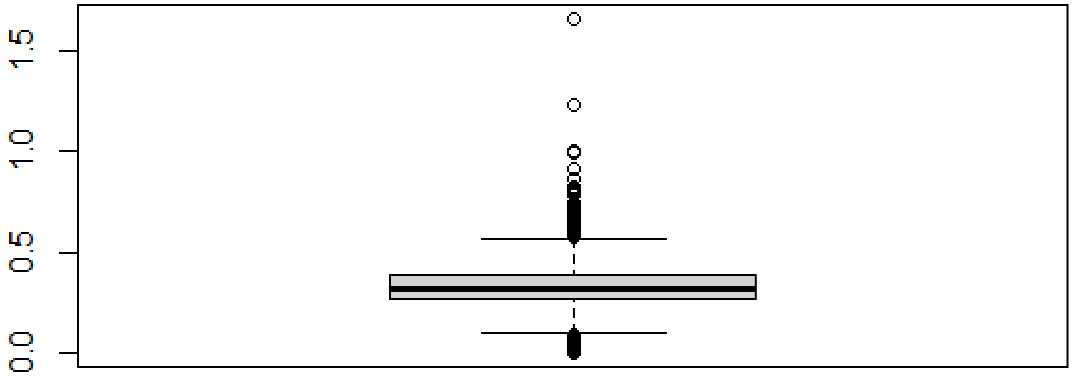
boxplot(Whitewine_v2$`fixed acidity`)



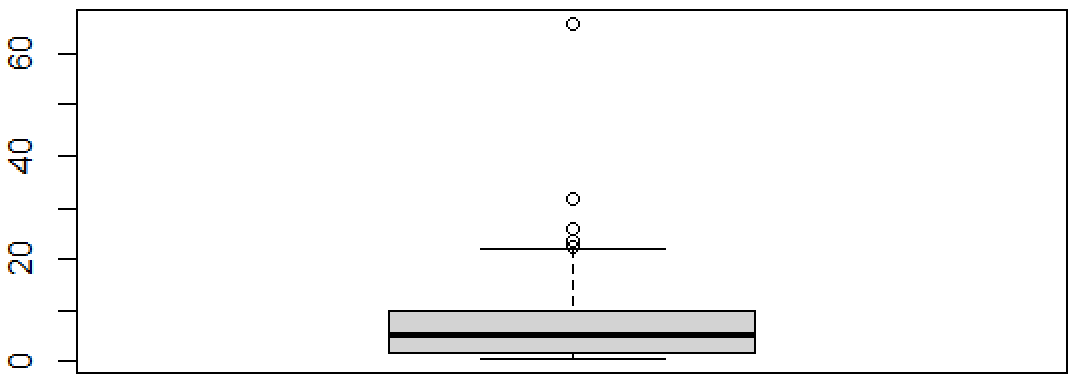
boxplot(Whitewine_v2$`volatile acidity`)



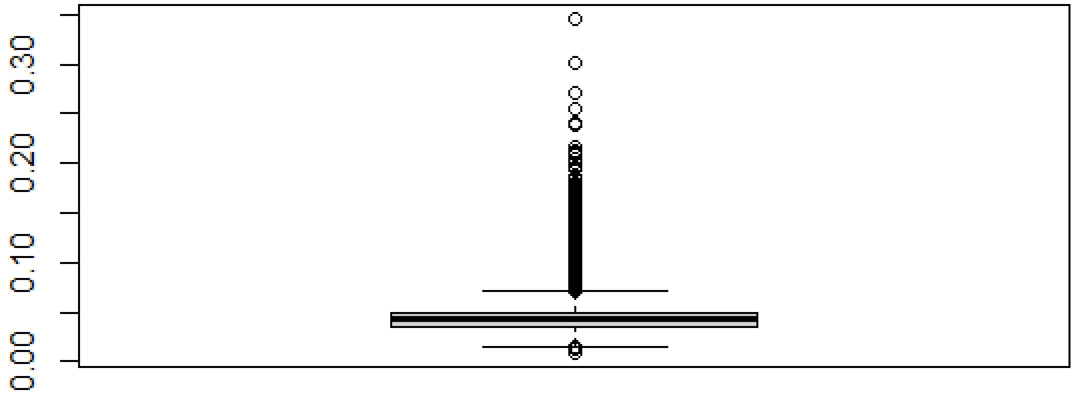
boxplot(Whitewine_v2$`citric acid`)



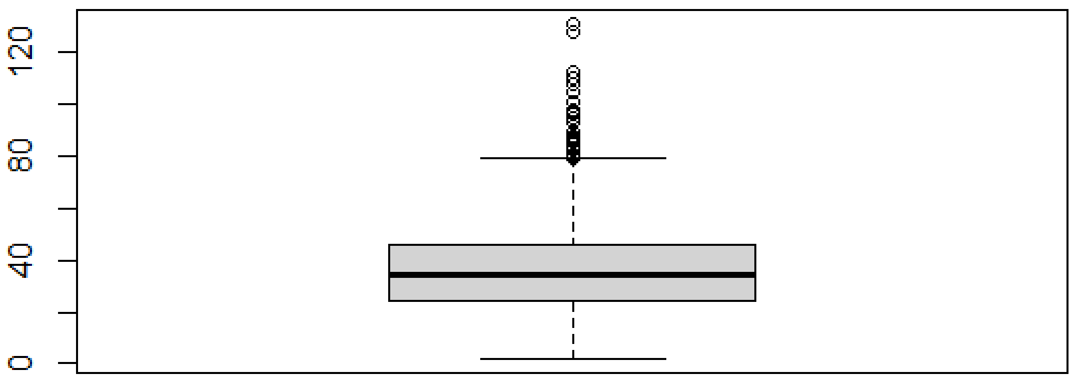
boxplot(Whitewine_v2$`residual sugar`)



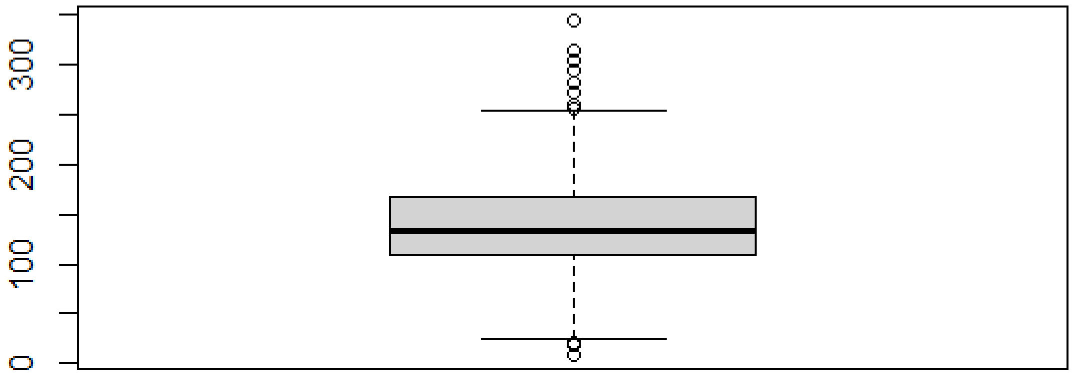
boxplot(Whitewine_v2$chlorides)



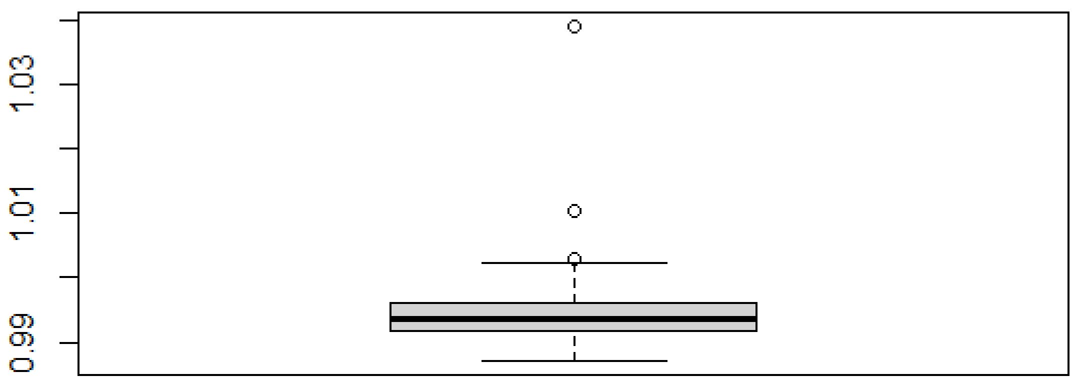
boxplot(Whitewine_v2$`free sulfur dioxide`)



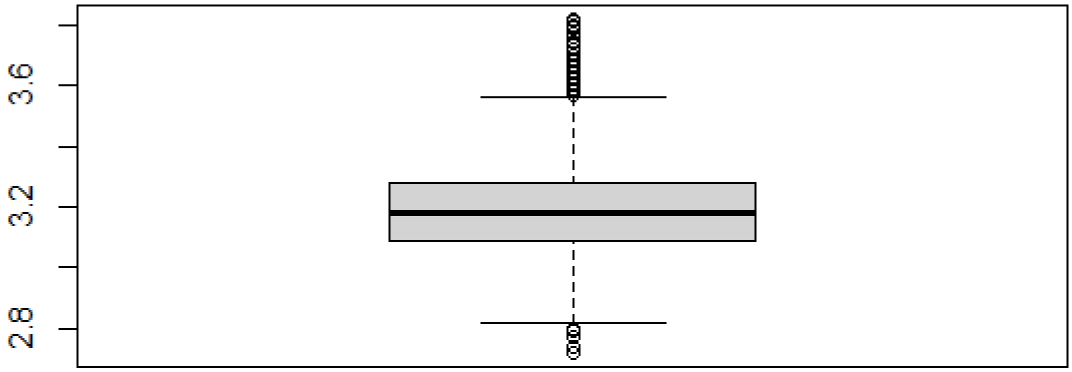
boxplot(Whitewine_v2$`total sulfur dioxide`)



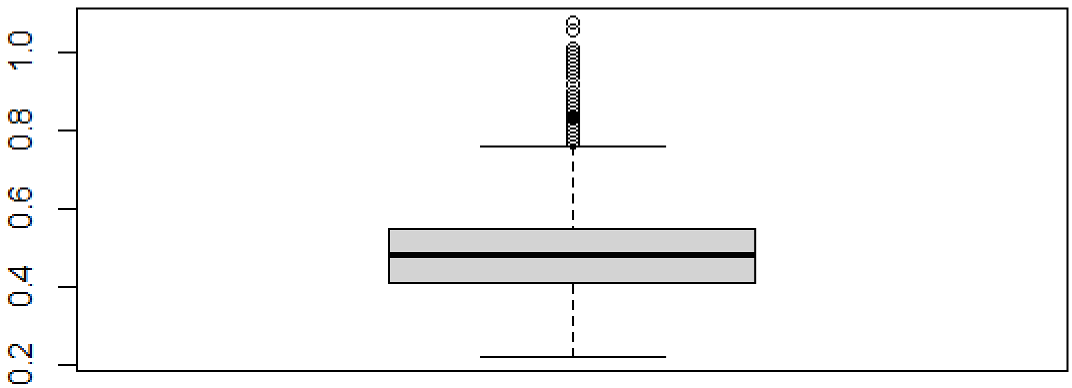
boxplot(Whitewine_v2$density)



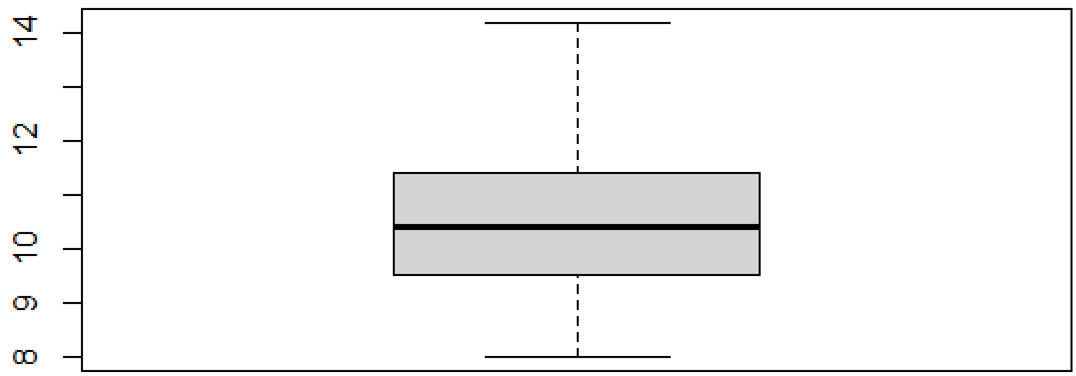
boxplot(Whitewine_v2$pH)



boxplot(Whitewine_v2$sulphates)



boxplot(Whitewine_v2$alcohol)

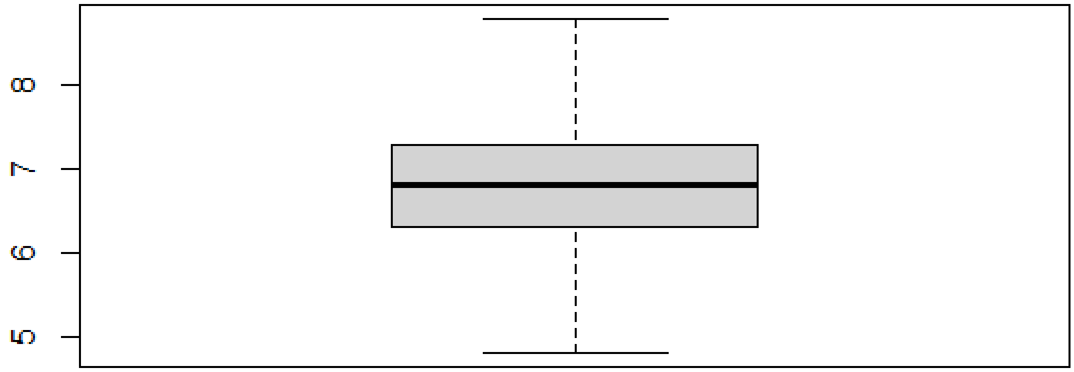



```
detect_outlier <- function(x) {
  # calculate first quantile
  Quantile1 <- quantile(x, probs=.25)
  # calculate third quantile
  Quantile3 <- quantile(x, probs=.75)
  # calculate inter quartile range
  IQR = Quantile3-Quantile1
  # return true or false
  x > Quantile3 + (IQR*1.5) | x < Quantile1 - (IQR*1.5)
}
# create remove outlier function
remove_outlier <- function(dataframe,
                           columns=names(dataframe)) {
  # for loop to traverse in columns vector
```

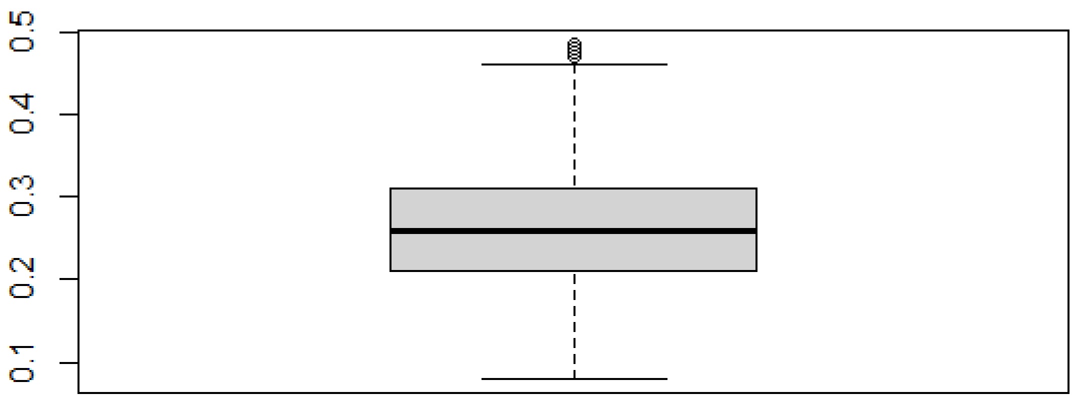
```
  for (col in columns) {
    # remove observation if it satisfies outlier function
    dataframe <- dataframe[!detect_outlier(dataframe[[col]]), ]
  }
  # return dataframe
  print("Remove outliers")
  print(dataframe)
}
outlier_remove_data <- remove_outlier(Whitewine_v2, c('fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dioxide','total sulfur dioxide','density', 'pH','sulphates','alcohol'))
print(outlier_remove_data)
```

```
[1] "Remove outliers"
# A tibble: 3,872 x 12
   `fixed acidity` `volatile acidity` `citric acid` `residual sugar` chlorides
             <dbl>              <dbl>         <dbl>            <dbl>     <dbl>
 1             8.1               0.27          0.41             1.45     0.033
 2             8.6               0.23          0.4              4.2      0.035
 3             7.9               0.18          0.37             1.2      0.04 
 4             6.5               0.31          0.14             7.5      0.044
 5             5.8               0.27          0.2             15.0      0.044
 6             6.5               0.39          0.23             5.4      0.051
 7             7.3               0.24          0.39            18.0      0.057
 8             7.3               0.24          0.39            18.0      0.057
 9             6.2               0.46          0.25             4.4      0.066
10             6.9               0.19          0.35             5        0.067
# ... with 3,862 more rows, and 7 more variables: `free sulfur dioxide` <dbl>,
#   `total sulfur dioxide` <dbl>, density <dbl>, pH <dbl>, sulphates <dbl>,
#   alcohol <dbl>, quality <dbl>
```

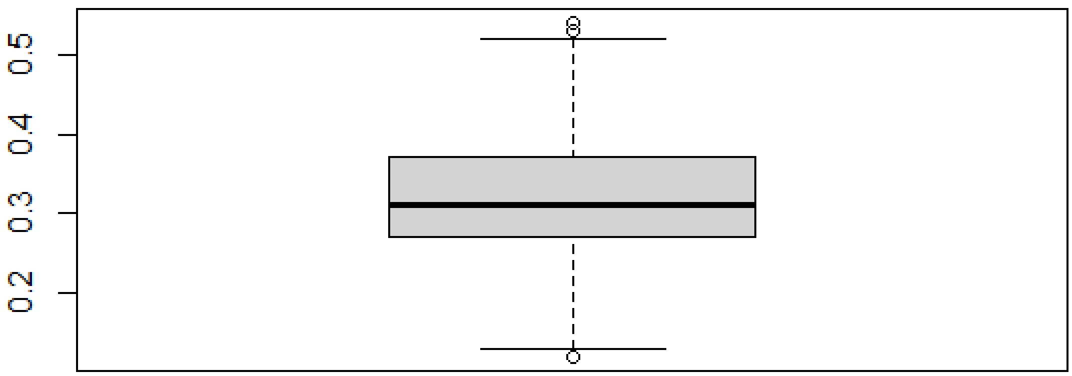
```
boxplot(outlier_remove_data$`fixed acidity`)
```



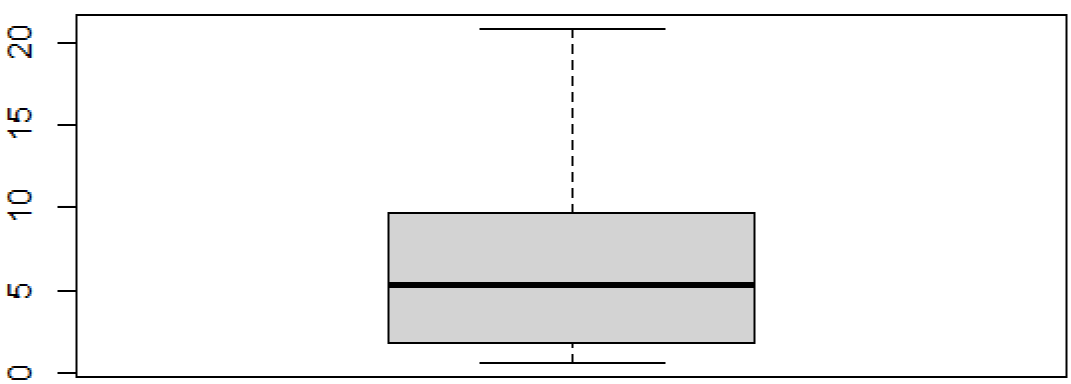
boxplot(outlier_remove_data$`volatile acidity`)



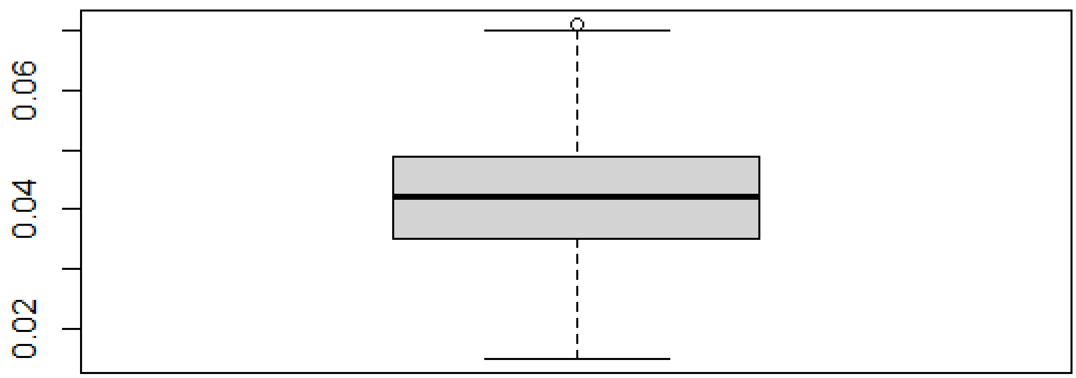
boxplot(outlier_remove_data$`citric acid`)



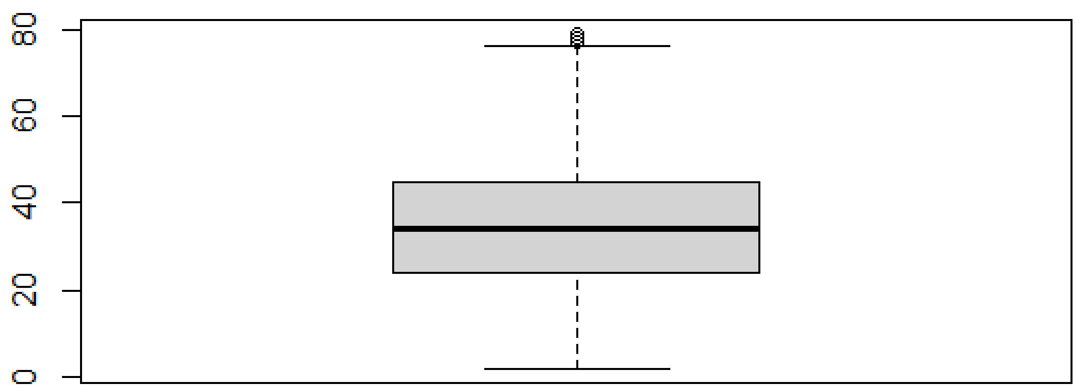
boxplot(outlier_remove_data$`residual sugar`)



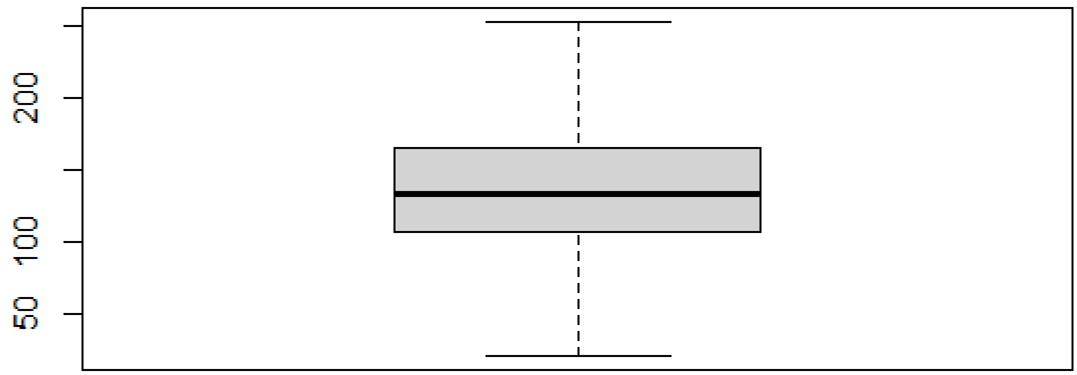
boxplot(outlier_remove_data$chlorides)



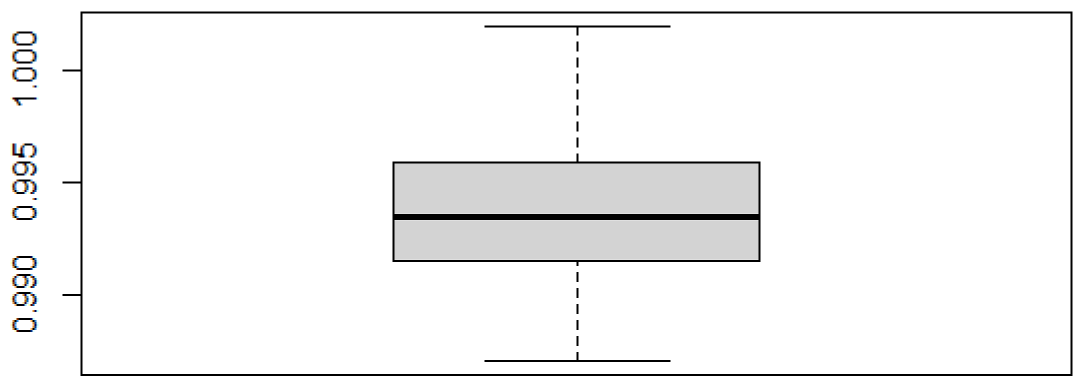

```
boxplot(outlier_remove_data$`free sulfur dioxide`)
```



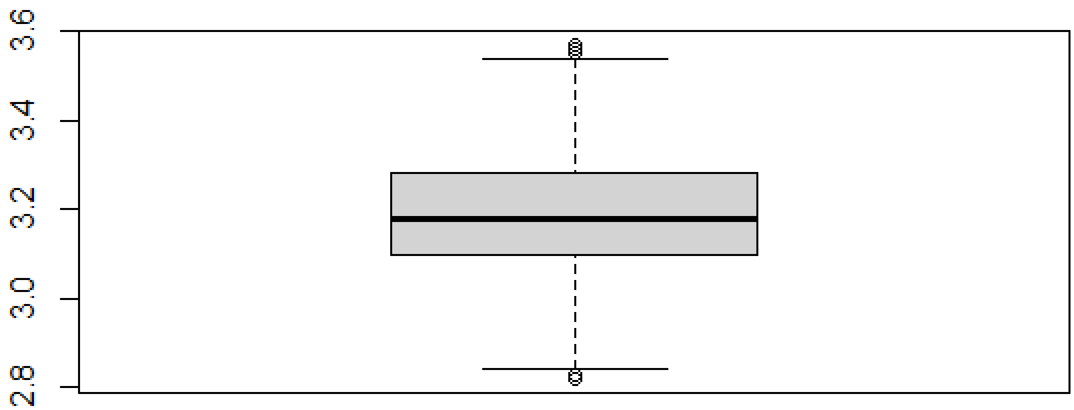

```
boxplot(outlier_remove_data$`total sulfur dioxide`)
```



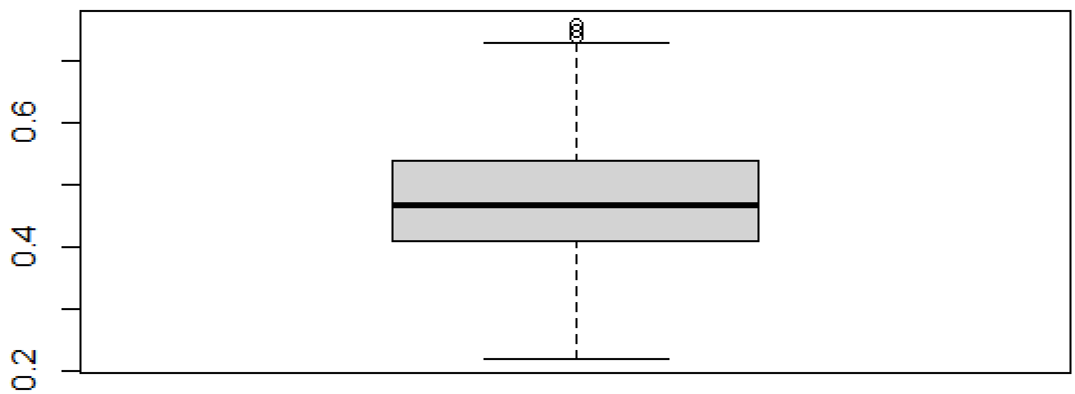
boxplot(outlier_remove_data$density)



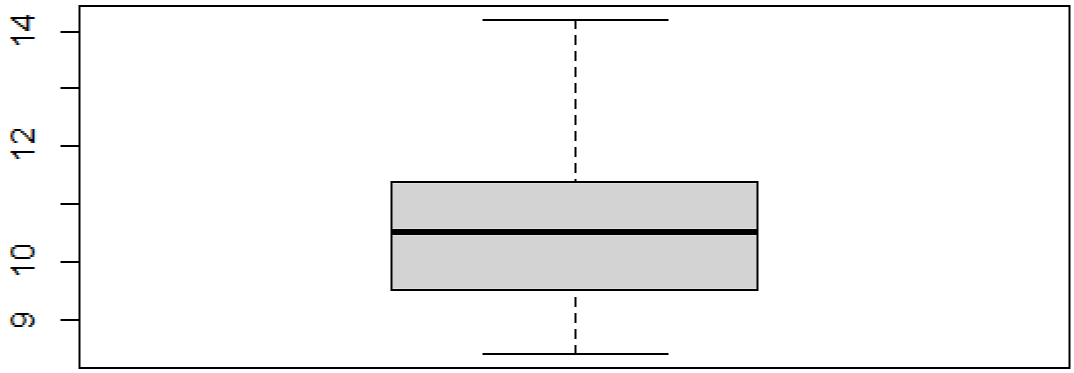
boxplot(outlier_remove_data$pH)



boxplot(outlier_remove_data$sulphates)



boxplot(outlier_remove_data$alcohol)




```
scale_data <- as.data.frame(scale(outlier_remove_data)) #z score scale
scale_data$quality <- outlier_remove_data$quality
print(scale_data)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides
1	1.7627223	0.09315308	1.05668988	-1.00650208	-0.92804768
2	2.4447506	-0.43224930	0.93554272	-0.44616777	-0.72583611
3	1.4899111	-1.08900227	0.57210122	-1.05744156	-0.22030720
4	-0.4197679	0.61855546	-2.21428359	0.22623340	0.18411593
5	-1.3746074	0.09315308	-1.48740060	1.74422998	0.18411593
6	-0.4197679	1.66936022	-1.12395910	-0.20165826	0.89185640
7	0.6714772	-0.30089870	0.81439555	2.35550377	1.49849109
8	0.6714772	-0.30089870	0.81439555	2.35550377	1.49849109
9	-0.8289849	2.58881439	-0.88166477	-0.40541619	2.40844313
10	0.1258546	-0.95765168	0.32980689	-0.28316143	2.50954891
11	0.8078828	-0.30089870	-0.39707611	0.75600402	0.79075062
12	-0.0105510	0.48720487	-1.12395910	-0.36466460	1.90291422
13	0.8078828	-0.30089870	-0.39707611	0.75600402	0.79075062
14	0.8078828	-1.08900227	-0.27592894	0.49111871	2.20623157
15	0.3986659	1.01260725	-1.48740060	-0.05902770	2.10512579
16	-0.9653905	0.09315308	1.29898421	0.22623340	0.68964484
17	0.8078828	-0.16954811	0.57210122	1.44878098	1.80180844
18	0.3986659	-1.87710584	-0.03363461	0.65412505	1.19517375
19	-1.1017961	-0.69495049	-1.00281193	1.16351988	0.79075062
20	0.9442885	0.55288016	0.93554272	2.54907380	1.70070266
21	0.8078828	-0.16954811	0.57210122	1.44878098	1.80180844
22	0.3986659	-1.87710584	-0.03363461	0.65412505	1.19517375
23	0.1258546	0.88125665	-0.51822327	-1.03706577	0.89185640
24	0.5350716	0.61855546	2.14701437	1.40802939	1.39738531
25	-0.1469566	1.93206141	0.20865972	0.57262188	0.68964484
26	-0.1469566	1.93206141	0.20865972	0.57262188	0.68964484
27	0.5350716	0.61855546	2.14701437	1.40802939	1.39738531
28	0.6714772	0.74990606	1.90472004	1.40802939	1.80180844
29	1.3535054	-0.30089870	-0.03363461	1.18389567	1.19517375
30	0.8078828	-1.35170346	-0.15478177	0.09379074	1.70070266
31	0.8078828	1.66936022	-1.12395910	0.12435443	-0.92804768
32	-1.5110131	-0.03819751	-0.88166477	0.81713139	-2.24242284
33	-0.4197679	2.12908730	0.93554272	1.36727781	-0.42251877
34	-0.2833623	-0.30089870	-0.63937044	1.91742422	-0.72583611
35	-0.0105510	0.09315308	-1.24510626	0.34848816	-0.82694189
36	0.1469566	0.09315308	0.15478177	1.89704842	0.62473033

36	-0.1469566	0.09315308	-0.15478177	1.89704842	-0.62473033
37	-0.6925792	0.61855546	0.20865972	-0.85368363	0.28522171
38	0.5350716	-1.08900227	1.05668988	-1.05744156	0.58853906
39	-0.6925792	1.27530844	-0.27592894	-0.32391301	0.68964484
40	-0.1469566	-0.30089870	0.32980689	1.36727781	0.79075062
41	0.2622603	-0.43224930	0.45095405	1.34690201	0.89185640
42	0.9442885	0.35585427	-0.15478177	0.52168240	1.29627953
43	0.9442885	0.09315308	-0.15478177	2.30456428	0.89185640
44	-1.2382018	-0.43224930	-0.27592894	1.32652622	1.19517375
45	0.3986659	1.01260725	-2.09313642	-1.05744156	1.09406797
46	-1.2382018	0.48720487	-1.12395910	-0.44616777	-0.42251877
47	-0.2833623	-1.48305406	0.20865972	-0.26278563	1.29627953
48	-0.0105510	-0.03819751	-1.00281193	0.28736078	0.99296218
49	0.3986659	0.74990606	-1.00281193	1.36727781	0.79075062
50	-0.0105510	-0.03819751	-1.00281193	0.28736078	0.99296218
51	-0.0105510	0.09315308	-0.76051760	1.97855160	0.68964484
52	0.3986659	0.74990606	-1.00281193	1.36727781	0.79075062
53	-0.5561736	-0.69495049	2.14701437	1.06164091	-0.01809564
54	0.2622603	-1.35170346	-0.03363461	0.38923974	0.28522171
55	-1.6474187	-0.23522341	-0.88166477	0.67450084	-1.02915346
56	0.2622603	-1.35170346	-0.03363461	0.38923974	0.28522171
57	-1.3746074	-0.03819751	-1.00281193	0.57262188	0.18411593
58	0.9442885	0.09315308	0.45095405	0.12435443	-0.62473033
59	-1.3746074	-0.03819751	-1.00281193	0.57262188	0.18411593
60	-1.5110131	0.22450368	-1.00281193	2.26381270	0.18411593
61	0.9442885	-0.43224930	0.45095405	0.12435443	-0.62473033
62	0.9442885	0.09315308	0.45095405	0.12435443	-0.62473033
63	0.3986659	-0.30089870	1.05668988	2.32494008	0.38632749
64	0.3986659	-0.30089870	1.05668988	2.32494008	0.38632749
65	0.6714772	0.74990606	-1.12395910	1.48953256	0.79075062
66	1.2170998	0.61855546	-0.76051760	0.28736078	-1.13025924
67	-0.9653905	1.93206141	-2.21428359	0.81713139	-0.52362455
68	0.1258546	0.35585427	0.93554272	2.66114066	0.08301014
69	0.1258546	0.22450368	0.93554272	0.36886395	-0.62473033
70	0.1258546	0.35585427	0.93554272	2.66114066	0.08301014
71	-1.3746074	-0.16954811	-0.76051760	1.36727781	0.89185640
72	0.2622603	0.35585427	2.63160304	0.87825877	0.38632749

73	1.2170998	-0.03819751	0.93554272	-1.07781735	-0.01809564
74	-0.1469566	1.01260725	-0.27592894	1.87667263	1.19517375
75	0.3986659	1.14395784	-1.00281193	1.83592104	1.29627953
76	0.6714772	0.74990606	-0.88166477	0.16510602	1.39738531
77	0.8078828	0.22450368	1.17783705	2.73245594	2.40844313
78	0.5350716	2.19476260	-1.00281193	0.06322705	1.59959688
79	-1.3746074	1.01260725	-1.36625343	0.04285126	-0.22030720
80	-0.2833623	-0.30089870	0.32980689	0.26698498	-1.13025924
81	-0.2833623	-0.30089870	0.32980689	0.26698498	-1.13025924
82	2.0355336	1.93206141	2.26816154	-0.89443522	0.38632749
83	0.5350716	-0.69495049	1.05668988	-1.03706577	-0.62473033

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
1	-1.63045941	-1.82399197	-1.05457662	-1.439034140	0.78817491
2	-1.22386402	-0.69543310	0.29760931	-0.353111748	0.48542567
3	-1.29162992	-1.52958531	-0.63851941	-0.063532443	1.49458980
4	-0.07184375	-0.10661977	0.57498078	0.226046862	0.18267643
5	-0.88503453	1.02193911	0.81768082	1.311969254	-1.12923694
6	-0.68173684	0.28592245	-0.15311933	0.370836514	-1.33106977
7	0.67358113	0.28592245	2.10052388	0.153652035	-1.23015335
8	0.67358113	0.28592245	2.10052388	0.153652035	-1.23015335
9	1.82560140	1.70888799	0.02023784	0.443231340	0.38450926
10	-0.20737555	0.31045634	0.40162361	1.239574428	-0.01915640
11	-0.95280043	-0.79356865	0.81768082	-0.425506574	-1.33106977
12	1.04629357	2.48170548	0.67899508	0.949995123	1.19184056
13	-0.95280043	-0.79356865	0.81768082	-0.425506574	-1.33106977
14	-0.61397094	-0.84263643	0.78300939	-1.801008271	0.78817491
15	0.80911292	0.65393078	0.26293788	-0.135927269	-0.62465487
16	2.02889909	2.59210798	0.64432365	-0.497901400	-0.12007281
17	1.14794241	1.34087966	1.26840946	-1.366639314	-0.42282205
18	1.96113319	0.60486300	0.81768082	1.529153733	-0.72557129
19	1.35124011	0.65393078	1.09505229	1.094784776	-0.92740411
20	0.60581523	0.80113411	2.13519531	-1.439034140	-0.22098922
21	1.14794241	1.34087966	1.26840946	-1.366639314	-0.42282205
22	1.96113319	0.60486300	0.81768082	1.529153733	-0.72557129
23	0.13145394	1.21821022	-0.39581937	0.588020992	1.19184056
24	2.23219678	1.41448133	1.51110950	-1.294244488	-0.12007281
25	-0.41067324	0.31045634	1.02570942	0.226046862	0.28359284

26	-0.41067324	0.31045634	1.02570942	0.226046862	0.28359284
27	2.23219678	1.41448133	1.51110950	-1.294244488	-0.12007281
28	1.48677190	1.43901522	1.51110950	-1.077060009	0.18267643
29	0.47028343	0.01604967	1.58045237	-1.294244488	0.58634208
30	-0.27514145	-0.15568755	0.47096648	0.732810645	-1.43198618
31	-0.41067324	-0.27835699	0.05490927	-0.353111748	-0.62465487
32	-1.90152300	-1.97119531	0.05490927	1.456758906	-1.12923694
33	1.62230370	2.54304021	1.40709520	0.298441688	0.88909132
34	0.74134702	1.24274411	1.51110950	0.370836514	0.28359284
35	1.35124011	1.61075244	0.78300939	0.008862383	0.38450926
36	0.60581523	1.02193911	1.40709520	0.515626166	0.78817491
37	-1.02056633	-1.48051753	-0.39581937	0.805205471	-0.52373846
38	0.40251753	-0.98983976	-0.67319084	-0.353111748	-0.32190563
39	-1.42716172	-1.28424642	-0.22246220	0.660415819	-0.92740411
40	1.96113319	1.65982021	1.09505229	-0.280716921	0.18267643
41	2.50326038	0.97287133	1.16439516	-0.208322095	0.08176002
42	-1.02056633	0.33499023	1.02570942	-0.787480705	0.58634208
43	-0.13960965	0.87473578	1.78848097	-0.715085878	1.59550621
44	1.48677190	0.80113411	1.16439516	0.660415819	-0.92740411
45	1.75783550	1.12007466	-0.08377646	-0.715085878	-0.52373846
46	0.47028343	-0.45009421	-0.49983367	-0.280716921	0.18267643
47	-0.07184375	-0.30289088	0.12425214	1.239574428	-0.62465487
48	1.28347421	1.88062521	0.78300939	-0.425506574	-0.12007281
49	1.14794241	1.63528633	1.44176663	-0.642691052	0.08176002
50	1.28347421	1.88062521	0.78300939	-0.425506574	-0.12007281
51	1.35124011	1.43901522	1.58045237	-0.280716921	0.18267643
52	1.14794241	1.63528633	1.44176663	-0.642691052	0.08176002
53	0.67358113	0.38405800	1.16439516	-0.280716921	-0.52373846
54	0.19921984	-0.27835699	0.67899508	0.153652035	-1.43198618
55	-1.56269351	-1.70132253	0.05490927	0.877600297	-1.43198618
56	0.19921984	-0.27835699	0.67899508	0.153652035	-1.43198618
57	1.35124011	0.35952412	0.78300939	0.877600297	-1.02832053
58	0.67358113	0.65393078	0.02023784	-1.149454835	-1.53290259
59	1.35124011	0.35952412	0.78300939	0.877600297	-1.02832053
60	1.69006960	0.72753245	1.75380954	0.877600297	-0.42282205
61	0.53804933	0.58032911	-0.01443360	-1.077060009	-1.63381901
62	0.67358113	0.65393078	0.02023784	-1.149454835	-1.53290259

```
63          0.26698574          0.18778690  2.06585244  0.949995123 -0.92740411
64          0.26698574          0.18778690  2.06585244  0.949995123 -0.92740411
65          0.94464472          1.46354910  1.61512380  0.081257209 -0.22098922
66         -0.81726863         -1.16157698  0.19359501 -0.425506574  0.18267643
67         -1.15609812         -0.45009421  0.74833795  1.384364080 -0.32190563
68          0.06368804          0.45765967  1.99650958 -1.873403097 -0.12007281
69         -1.35939582         -1.03890754  0.19359501 -0.135927269 -1.53290259
70          0.06368804          0.45765967  1.99650958 -1.873403097 -0.12007281
71          0.60581523          0.26138856  1.16439516  0.732810645 -1.02832053
72          1.62230370          2.37130298  0.95636656 -1.004665183  1.29275697
73         -1.76599121         -1.89759364 -0.81187658 -2.162982402  0.18267643
74          1.08017652          1.43901522  1.51110950  0.008862383  0.08176002
75          0.74134702          1.48808299  1.71913810 -0.497901400  0.08176002
76          0.80911292          1.04647300  0.78300939 -0.787480705 -0.12007281
77          1.21570831          1.41448133  2.13519531 -1.656218619 -0.42282205
78          0.33475164          0.62939689  0.40162361  0.081257209 -0.72557129
79          1.01241062          0.72753245  0.08958071  0.732810645  1.39367339
80          0.06368804         -0.05755199 -0.01443360  0.008862383 -1.12923694
81          0.06368804         -0.05755199 -0.01443360  0.008862383 -1.12923694
82         -1.63045941          1.70888799 -0.29180507 -1.221849662  0.68725850
83         -0.13960965         -1.28424642 -0.63851941 -0.135927269  0.28359284
        alcohol quality
1   1.1516562188       5
2  -0.7395235934       5
3   0.1649537081       5
4  -0.9039740118       5
5  -0.3283975473       5
6  -0.4928479657       5
7  -1.6440008948       5
8  -1.6440008948       5
9  -0.6572983841       5
10 -0.6572983841       5
11 -0.9039740118       5
12 -0.9039740118       5
13 -0.9039740118       5
14 -1.0684244303       5
15 -0.4928479657       5
```

```
16 -1.3151000579        5
17 -1.2328748487        5
18 -0.9861992210        5
19 -0.9861992210        5
20 -1.3151000579        5
21 -1.2328748487        5
22 -0.9861992210        5
23 -0.2461723380        5
24 -1.1506496395        5
25 -1.2328748487        5
26 -1.2328748487        5
27 -1.1506496395        5
28 -1.1506496395        5
29 -1.4795504764        5
30 -0.7395235934        5
31 -0.0817219196        5
32  0.0005032896        5
33 -1.3151000579        5
34 -1.1506496395        5
35 -1.3973252672        5
36 -0.8217488026        5
37 -0.3283975473        5
38 -0.1639471288        5
39  0.0005032896        5
40 -0.9039740118        5
41 -0.6572983841        5
42 -1.0684244303        5
43 -0.3283975473        5
44 -0.9861992210        5
45 -1.1506496395        5
46  0.3294041265        5
47 -0.8217488026        5
48 -1.3973252672        5
49 -1.4795504764        5
50 -1.3973252672        5
51 -1.0684244303        5
52 -1.4795504764        5
```

```
53 -1.4795504764        5
54 -1.1506496395        5
55 -0.0817219196        5
56 -1.1506496395        5
57 -0.9861992210        5
58  0.3294041265        5
59 -0.9861992210        5
60 -0.9861992210        5
61  0.3294041265        5
62  0.3294041265        5
63 -1.5617756856        5
64 -1.5617756856        5
65 -1.5617756856        5
66 -0.1639471288        5
67 -0.4928479657        5
68 -1.3973252672        5
69 -0.3283975473        5
70 -1.3973252672        5
71 -1.0684244303        5
72 -0.9039740118        5
73  0.0005032896        5
74 -1.0684244303        5
75 -1.4795504764        5
76 -1.4795504764        5
77 -1.2328748487        5
78 -0.5750731749        5
79 -0.4928479657        5
80 -0.0817219196        5
81 -0.0817219196        5
82  0.6583049634        5
83 -0.1639471288        5
 [ reached 'max' / getOption("max.print") -- omitted 3789 rows ]
```

• Defining the number of cluster centres

Partitioning clustering, such as k-means clustering, requires the user to choose the number of clusters k to be formed.

The method used to measure similarities and the criteria utilized for partitioning are both quite subjective.

```
# Installing Packages

install.packages("ClusterR")

install.packages("cluster")



# Loading package
```

library(ClusterR)

library(cluster)

Fitting K-Means clustering Model

to training dataset

set.seed(240) # Setting seed

kmeans.re <- kmeans(scale_data, centers = 2, nstart = 100) #scale_data

kmeans.re

```
Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides
1    0.13980810       0.06884549  0.03606444      0.8025412  0.6287181
2   -0.09982756      -0.04915793 -0.02575119     -0.5730407 -0.4489253
  free sulfur dioxide total sulfur dioxide    density         pH   sulphates
1           0.5715542            0.7513591  0.9429859 -0.1444100  0.09033134
2          -0.4081084           -0.5364950 -0.6733228  0.1031135 -0.06449954
     alcohol  quality
1 -0.8229994 5.681959
2  0.5876486 6.229748

Clustering vector:
   [1] 2 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 2 2 1 1 1 1 2 2 2
  [40] 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 1 2 1 1 1 1 1
  [79] 1 2 2 2 2 2 1 2 2 1 1 2 2 2 1 2 1 1 1 2 2 2 1 1 2 1 1 1 2 2 1 1 1 1 1 1 1 1 1
 [118] 1 2 2 1 2 2 2 2 1 1 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 1 1
 [157] 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1
 [196] 2 1 1 1 1 2 1 1 2 2 2 2 1 1 1 1 2 1 2 2 2 2 2 2 2 1 1 2 2 1 2 1 1 2 2 2 2 1 2
 [235] 2 2 1 2 2 2 1 1 1 1 2 1 1 2 2 2 2 1 2 2 1 1 2 2 2 1 1 1 1 2 2 2 2 1 2 2 1 2 1
 [274] 1 1 1 1 2 1 2 2 2 2 1 1 2 2 2 1 2 1 1 1 1 1 2 2 1 2 2 2 2 2 1 1 1 2 1 2 1 2 2
 [313] 2 2 1 1 2 2 1 1 2 2 1 2 2 2 1 2 2 2 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 2 1 2
 [352] 2 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 1 2 2 1 2 1 2 1 1 2 1 1 1 2 1 1 1 1 2 1 1 2 1
 [391] 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 1 1 1 2
 [430] 2 1 2 2 2 2 1 1 1 1 1 2 2 1 1 1 1 2 2 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 [469] 1 1 2 1 2 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 2 2 2 1
 [508] 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 2 1 1 2 1 1 1 1 1 2 2
 [547] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 2 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1
 [586] 1 2 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 2 2 1 2 1 1 1
 [625] 1 1 1 1 1 1 1 2 2 1 2 2 2 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 1 2 2 1 2 1
 [664] 2 2 1 1 1 2 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 1 2 1 1 2 2 2 1 1 1
 [703] 1 2 1 2 1 2 1 2 1 2 2 2 2 1 2 1 2 2 1 1 1 2 2 2 2 1 2 1 1 2 2 1 2 1 1 1 2 1 2
 [742] 2 1 2 1 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1 2 2 1 1 2 1 1 1 2 2 1 1 2 1 1 2 1 1 2 1
 [781] 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 2 2 2 1 1 1 2 1 1 1 1 2 2 1 2 1 2 2 2 2 2 2 2
 [820] 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1
 [859] 2 1 2 2 2 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 1 1 2 2 1 1 1 2 2 1 2 1 1 1 2 1 1 1 1
 [898] 1 1 1 1 2 1 1 2 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 1 1 1
```

```
 [937] 1 1 1 1 2 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1
 [976] 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1
 [ reached getOption("max.print") -- omitted 2872 entries ]

Within cluster sum of squares by cluster:
[1] 14183.44 20745.59
 (between_SS / total_SS =  22.4 %)

Available components:

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

```
# Cluster identification for

# each observation

kmeans.re$cluster
```

In R, a confusion matrix is a table that categorizes predictions based on their actual values. It has two dimensions, one of which will show the anticipated values and the other will show the actual values.

The anticipated values will be represented by each row in the confusion matrix, while the actual values will be represented by the columns. This can also be reversed. Even though the matrixes are simple, the terminology behind them appears to be complicated. There is always the possibility of becoming confused regarding the classes. As a result, the phrase – **Confusion matrix** – was coined.

```
# Confusion Matrix

cm <- table(scale_data$quality, kmeans.re$cluster)

cm

kmeans.re <- kmeans(scale_data, centers = 3, nstart = 100)

kmeans.re
```

```
> # Confusion Matrix
> cm <- table(scale_data$quality, kmeans.re$cluster)
> cm

       1    2
  5  694  383
  6  757 1100
  7  143  650
  8   19  126
> kmeans.re <- kmeans(scale_data, centers = 3, nstart = 100)
> kmeans.re
K-means clustering with 3 clusters of sizes 1308, 1195, 1369

Cluster means:
  fixed acidity volatile acidity  citric acid residual sugar  chlorides
1     0.1566189        0.1137394  0.118818171      1.0566638  0.6015251
2    -0.1076901        0.3106785 -0.003322472     -0.4762711 -0.9064728
3    -0.0556376       -0.3798626 -0.110623676     -0.5938439  0.2165377
  free sulfur dioxide total sulfur dioxide    density         pH   sulphates
1           0.7037857            0.8524996  1.1129129 -0.2439660  0.03724261
2          -0.3864306           -0.6748205 -0.9520536 -0.1594329 -0.34681805
3          -0.3351110           -0.2254631 -0.2322761  0.3722643  0.26715430
     alcohol  quality
1 -0.8822862 5.675841
2  1.1219542 6.497908
3 -0.1363806 5.879474

Clustering vector:
   [1] 2 3 3 3 1 3 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 3 2 2 1 1 1 1 3 3 3
  [40] 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 3 3 1 3 1 1 1 2 1 1 1 1 1
  [79] 3 3 3 3 3 3 1 2 3 3 1 3 2 2 1 2 1 1 1 2 3 3 1 1 3 1 1 3 3 3 1 1 1 1 1 3 1 1 1
 [118] 1 3 3 3 2 3 3 2 1 1 3 1 3 1 3 1 1 1 1 1 1 3 1 1 1 1 3 3 1 1 1 3 3 3 3 3 3 1 1
 [157] 3 1 1 3 3 1 1 1 1 2 1 3 1 1 1 1 3 3 3 3 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 3 3 3 3
 [196] 3 1 1 1 1 3 1 1 3 3 3 3 1 1 1 1 2 1 3 2 3 3 3 3 2 3 1 3 3 1 3 1 1 2 3 3 3 1 3
 [235] 2 3 1 2 2 3 1 1 1 1 3 1 1 3 3 3 3 1 3 3 1 3 3 3 3 1 1 1 1 3 2 3 3 1 3 3 3 3 1
 [274] 1 1 1 1 3 3 2 3 3 3 1 1 3 1 3 1 3 3 1 1 1 3 2 2 3 3 3 3 3 3 1 1 1 3 3 3 1 3 3
 [313] 2 3 1 3 3 3 3 3 3 3 1 3 3 2 1 3 3 3 1 3 3 1 1 1 3 1 1 1 1 1 1 1 1 2 3 3 3 1 3
```

```
 [352] 3 1 1 1 1 1 1 1 1 2 3 1 3 2 1 3 1 3 2 1 3 1 3 1 1 3 1 1 1 2 1 1 1 1 3 1 1 3 1
 [391] 1 1 1 1 3 3 3 1 1 1 1 3 2 3 1 1 3 1 1 1 1 1 1 1 1 1 3 3 1 1 1 3 3 3 3 1 3 1 3
 [430] 2 3 3 3 3 3 1 1 1 1 1 2 2 1 1 3 1 3 3 3 3 3 3 3 3 3 1 1 1 1 1 3 1 1 1 3 3 1 1
 [469] 1 1 3 1 3 1 1 1 3 3 1 1 3 1 1 1 1 1 1 1 3 3 3 3 1 3 3 3 1 1 1 3 3 1 2 2 3 3 1
 [508] 1 1 1 1 1 1 1 3 1 1 1 1 3 1 1 1 1 1 1 1 1 3 1 3 2 1 1 1 3 1 1 3 1 1 3 1 1 3 2
 [547] 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 3 3 3 3 3 1 3 3 1 1 1 1 3 1 3 1 1 1 3 3 1 3 1 1
 [586] 1 3 3 3 3 3 1 1 1 3 1 3 3 1 1 1 1 2 1 1 2 1 1 1 1 3 3 1 1 1 1 1 3 3 1 3 1 1 1
 [625] 1 1 1 1 1 1 1 2 3 3 3 3 2 2 1 1 1 1 1 1 1 1 3 2 3 1 3 1 1 1 3 3 1 3 3 2 1 3 3
 [664] 2 3 1 1 1 2 1 1 1 1 1 1 3 1 3 3 1 3 1 3 3 3 3 3 3 2 2 3 3 1 3 3 1 3 3 2 1 1 1
 [703] 1 2 3 3 1 2 1 3 1 3 2 3 3 1 3 1 3 3 1 1 3 3 3 3 3 1 3 1 3 2 3 3 2 3 3 1 2 1 3
 [742] 3 1 3 1 2 1 1 1 1 2 3 1 3 3 3 3 1 1 1 2 2 1 1 3 1 1 1 3 3 1 3 3 1 1 3 1 1 3 1
 [781] 1 1 1 1 3 3 1 2 3 1 1 1 1 3 3 2 2 3 3 1 1 1 3 1 1 3 3 3 3 1 2 1 3 3 3 3 3 3 2
 [820] 1 1 1 1 1 2 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 1 1 2 1 3 1 3 1 3 1 1 3 1 1 1 1 1
 [859] 3 1 3 3 3 3 1 1 3 3 2 2 1 1 1 1 3 3 1 3 1 1 3 2 1 1 3 2 3 1 3 3 1 3 2 1 1 3 3
 [898] 3 1 1 1 2 1 1 3 1 3 3 3 1 1 3 1 3 3 1 1 1 1 1 1 3 1 3 1 3 3 3 3 3 1 1 3 1 1 1
 [937] 1 1 1 1 3 1 1 1 3 3 3 2 1 1 1 1 1 3 1 1 1 3 1 2 1 1 1 1 3 1 1 1 3 3 1 3 1 1 1
 [976] 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3
 [ reached getOption("max.print") -- omitted 2872 entries ]

Within cluster sum of squares by cluster:
[1] 10679.703  9705.474 11632.759
 (between_SS / total_SS =  28.9 %)

Available components:

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

Cluster identification for

each observation

kmeans.re$cluster

Confusion Matrix

cm <- table(scale_data$quality, kmeans.re$cluster)

cm

```
# Confusion Matrix
cm <- table(scale_data$quality, kmeans.re$cluster)
cm

    1   2   3
5 579  80 418
6 593 544 720
7 117 467 209
8  19 104  22
```

kmeans.re <- kmeans(scale_data, centers = 4, nstart = 100)

kmeans.re

```
> kmeans.re <- kmeans(scale_data, centers = 4, nstart = 100)
> kmeans.re
K-means clustering with 4 clusters of sizes 819, 841, 1259, 953

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides
1    -0.5523429        0.5271393  -0.1615861     -0.5074942 -0.9496515
2     0.9395848       -0.1975831   0.3724461     -0.4689300 -0.2173017
3     0.1295383        0.1076549   0.1114726      1.0661563  0.6417822
4    -0.5256147       -0.4208784  -0.3370746     -0.5585340  0.1600330
  free sulfur dioxide total sulfur dioxide    density         pH   sulphates
1          -0.3666957           -0.7050190 -1.0908271  0.1982911 -0.30120483
2          -0.5290823           -0.4864137 -0.3646357 -0.7031205 -0.24582834
3           0.7120661            0.8805916  1.1360880 -0.2313228  0.07150005
4          -0.1586666           -0.1282060 -0.2416461  0.7556762  0.38133245
     alcohol  quality
1  1.3025363 6.636142
2  0.2601661 5.890606
3 -0.9206934 5.661636
4 -0.1326590 6.003148

Clustering vector:
   [1] 2 2 2 4 3 4 3 3 3 4 2 3 2 3 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 4 2 1 3 3 3 3 4 2 4
  [40] 3 3 3 3 3 3 4 4 3 3 3 3 3 3 3 4 3 3 2 3 3 2 2 3 3 3 2 4 3 2 3 3 3 2 3 3 3 3 3
  [79] 4 2 2 2 2 2 3 1 4 4 3 4 2 1 3 2 3 3 3 2 2 2 3 3 2 3 3 4 4 4 3 3 3 3 3 4 3 3 3
 [118] 3 4 4 3 2 2 2 1 3 3 4 3 4 3 4 3 3 3 3 3 3 2 3 3 3 3 4 4 3 3 3 4 4 4 2 2 4 3 3
 [157] 4 2 3 4 4 3 3 3 3 2 3 4 3 3 3 3 4 4 2 4 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 4 2 4 4
 [196] 2 3 3 3 3 2 3 3 4 4 4 4 3 3 3 3 1 3 2 2 2 2 2 2 2 2 3 4 4 3 4 3 3 2 2 2 2 3 2
 [235] 2 2 3 2 2 2 3 3 3 2 2 3 3 4 2 2 1 3 1 4 3 2 2 4 4 3 3 3 3 4 1 2 4 3 4 4 2 2 3
 [274] 3 3 3 3 2 2 2 4 4 4 3 3 2 2 4 3 2 2 3 3 3 4 2 2 2 4 2 4 4 2 3 3 3 2 2 2 3 4 2
 [313] 2 2 3 2 2 2 2 2 2 4 3 2 2 1 2 4 2 4 3 2 4 3 3 3 4 3 3 3 3 3 3 3 3 2 4 4 2 3 2
 [352] 2 3 3 3 3 3 3 3 3 2 4 3 4 1 3 2 3 2 2 3 4 3 4 3 3 2 3 3 3 2 3 3 3 3 4 3 3 2 3
 [391] 3 3 3 3 2 2 4 3 3 3 3 4 2 2 3 3 2 3 3 3 3 3 3 3 3 3 2 2 3 3 3 4 4 4 4 3 2 3 2
 [430] 2 2 4 2 4 2 3 3 3 3 3 2 2 3 3 4 3 4 2 2 2 4 4 4 2 2 3 3 3 3 3 2 3 3 3 4 2 3 3
 [469] 3 3 4 3 2 3 3 3 4 4 3 3 4 3 3 3 3 3 3 3 4 4 4 4 3 4 4 4 3 3 3 2 4 3 2 2 2 4 3
 [508] 3 3 3 3 3 3 3 2 3 3 3 3 4 3 3 3 3 3 3 3 3 4 3 2 2 3 3 3 4 3 3 2 3 3 4 3 3 2 1
 [547] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 4 2 4 3 4 2 3 3 3 3 2 3 2 3 3 3 4 4 3 4 2 3
```

```
 [586] 3 4 4 4 3 4 3 3 3 2 3 4 4 3 3 3 3 1 3 3 1 3 3 3 3 2 4 3 3 3 3 3 2 2 3 2 3 3 3
 [625] 3 3 3 3 3 3 3 1 4 4 2 2 2 1 3 3 3 3 3 3 3 3 2 1 2 3 4 3 3 3 2 4 3 2 2 1 3 2 2
 [664] 2 4 3 3 3 2 3 3 3 3 3 3 4 3 4 4 3 2 3 4 4 4 4 2 4 2 2 2 3 3 4 4 3 2 2 1 3 3 3
 [703] 3 1 4 2 3 2 3 2 3 2 1 2 4 3 2 3 4 4 3 3 4 4 2 4 4 3 4 3 4 1 2 4 2 2 2 2 1 3 2
 [742] 4 3 2 3 2 3 3 3 3 2 4 3 4 4 4 4 3 3 3 2 2 3 3 2 3 3 3 4 2 3 2 4 3 3 4 3 3 2 3
 [781] 3 3 3 3 2 2 3 2 4 3 3 3 3 2 2 2 2 4 4 3 3 3 2 3 3 4 4 2 2 3 2 3 4 4 4 4 4 4 1
 [820] 3 3 3 3 3 2 3 3 3 3 3 3 3 4 2 4 2 2 2 2 2 4 3 3 2 3 4 3 4 3 4 3 3 2 3 3 3 3 3
 [859] 2 3 2 2 4 4 3 3 2 2 2 2 3 3 3 3 4 4 3 4 3 3 4 2 3 3 2 2 4 3 4 2 3 4 2 3 3 4 4
 [898] 4 3 3 3 1 3 3 4 3 4 2 4 3 3 2 3 4 4 3 3 3 3 3 3 3 3 3 3 2 4 4 2 2 3 3 4 3 3 3
 [937] 3 3 3 3 2 3 3 3 2 4 4 1 3 3 3 3 3 2 3 3 3 2 3 2 3 3 3 3 4 3 3 3 2 4 3 2 3 3 3
 [976] 4 4 4 2 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 2
 [ reached getOption("max.print") -- omitted 2872 entries ]

Within cluster sum of squares by cluster:
[1]  5800.185  6652.386 10067.884  7363.821
 (between_SS / total_SS =  33.6 %)

Available components:

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

Cluster identification for

each observation

kmeans.re$cluster

Confusion Matrix

cm <- table(scale_data$quality, kmeans.re$cluster)

cm

```
# Confusion Matrix
cm <- table(scale_data$quality, kmeans.re$cluster)
cm

    1   2   3   4
5  29 250 572 226
6 328 447 559 523
7 374 130 110 179
8  88  14  18  25
```

Installing Packages

install.packages("NbClust")

library("NbClust")

library(cluster)

library(factoextra)

result <- NbClust (data = scale_data, distance = "euclidean", min.nc = 2, max.nc = 15, method = 'complete', index = "ch") #result

print(result$Best.nc)

```
> result <- NbClust(data = scale_data, distance = "euclidean", min.nc = 2, max.nc = 15,
 method = 'complete', index = "ch") #result
> print(result$Best.nc)
Number_clusters     Value_Index
         2.0000        753.5564
```

I used the elbow approach. Remember, the primary principle underlying partitioning methods like k-means clustering is to construct clusters in such a way that the total intra-cluster variation [or total within-cluster sum of square (WSS)] is minimized. The total WSS assesses the clustering's compactness, and we want it to be as little as feasible.

The Elbow technique calculates total WSS as a function of cluster count: One should select a number of clusters such that adding another cluster does not significantly increase the total WSS.

1. Run a clustering algorithm (e.g., k-means clustering) for various k values. For example, changing k from 1 to 10 clusters.

2. Determine the total within-cluster sum of squares for each k. (wss).

3. Draw the wss curve based on the number of clusters k.

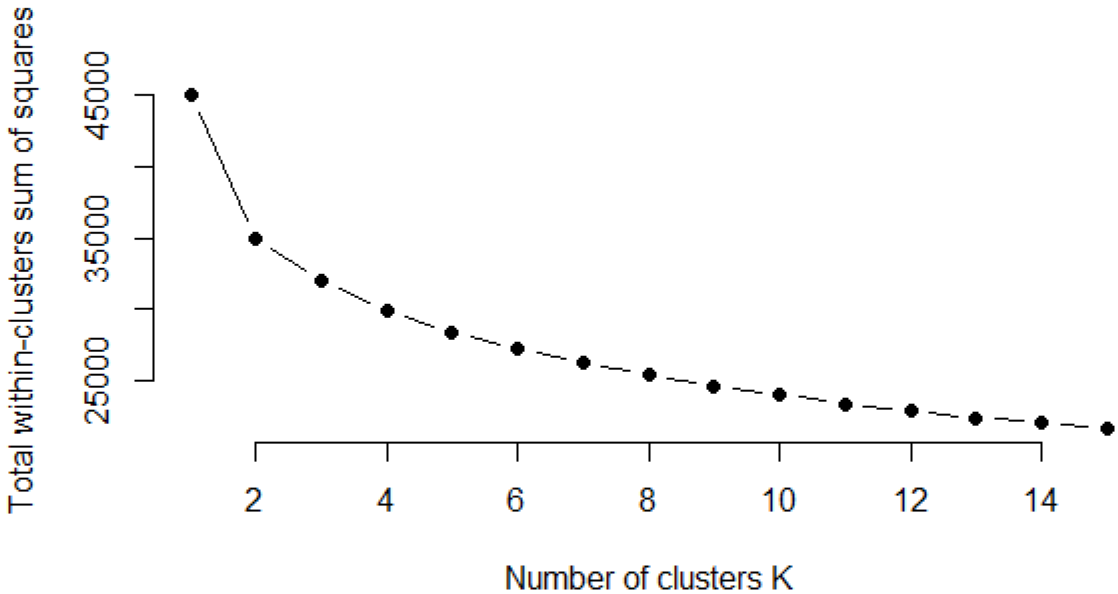
4. The location of a bend (knee) in the plot is commonly used to determine the proper number of clusters.

#The Elbow Method is used to determine the ideal number of clusters.

set.seed(123)

```
# Compute and plot wss for k = 2 to k = 15.

k.max <- 15

data <- scale_data

wss <- sapply(1:k.max,

              function(k){kmeans(data, k, nstart=50,iter.max = 100
)$tot.withinss})

wss

plot(1:k.max, wss,

     type="b", pch = 19, frame = FALSE,

     xlab="Number of clusters K",

     ylab="Total within-clusters sum of squares")
```

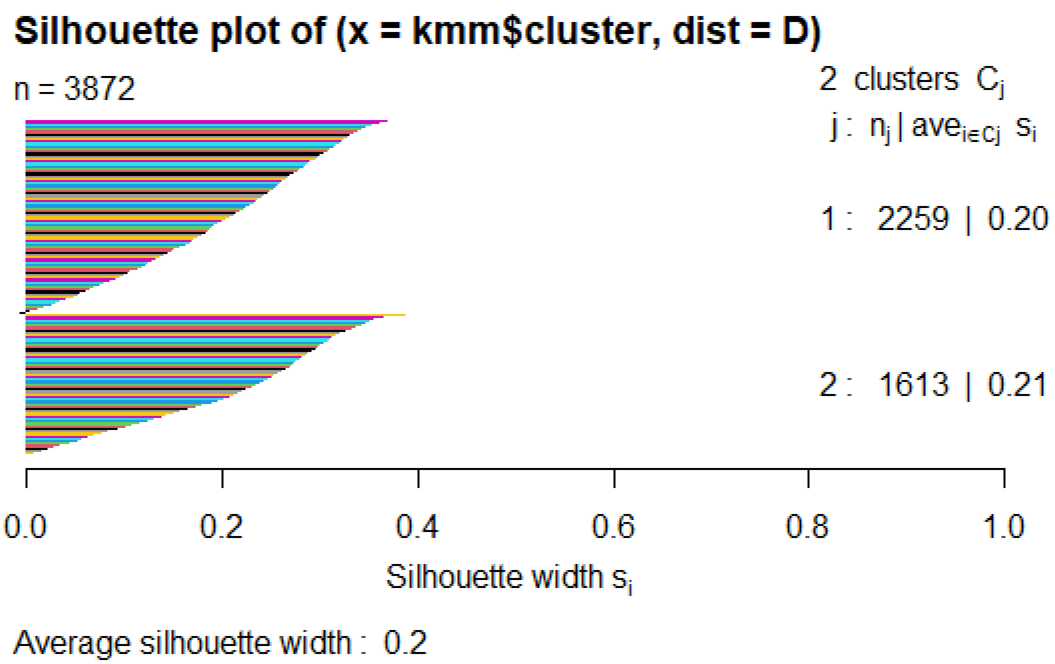
```
> #Elbow Method for finding the optimal number of clusters
> set.seed(123)
> # Compute and plot wss for k = 2 to k = 15.
> k.max <- 15
> data <- scale_data
> wss <- sapply(1:k.max,
+               function(k){kmeans(data, k, nstart=50,iter.max = 100 )$tot.withinss})
> wss
 [1] 45030.99 34929.03 32017.93 29884.28 28437.22 27213.01 26286.64 25456.82
 [9] 24679.33 24028.62 23405.92 22895.48 22436.16 22062.58 21690.06
> plot(1:k.max, wss,
+      type="b", pch = 19, frame = FALSE,
+      xlab="Number of clusters K",
+      ylab="Total within-clusters sum of squares")
> 
```



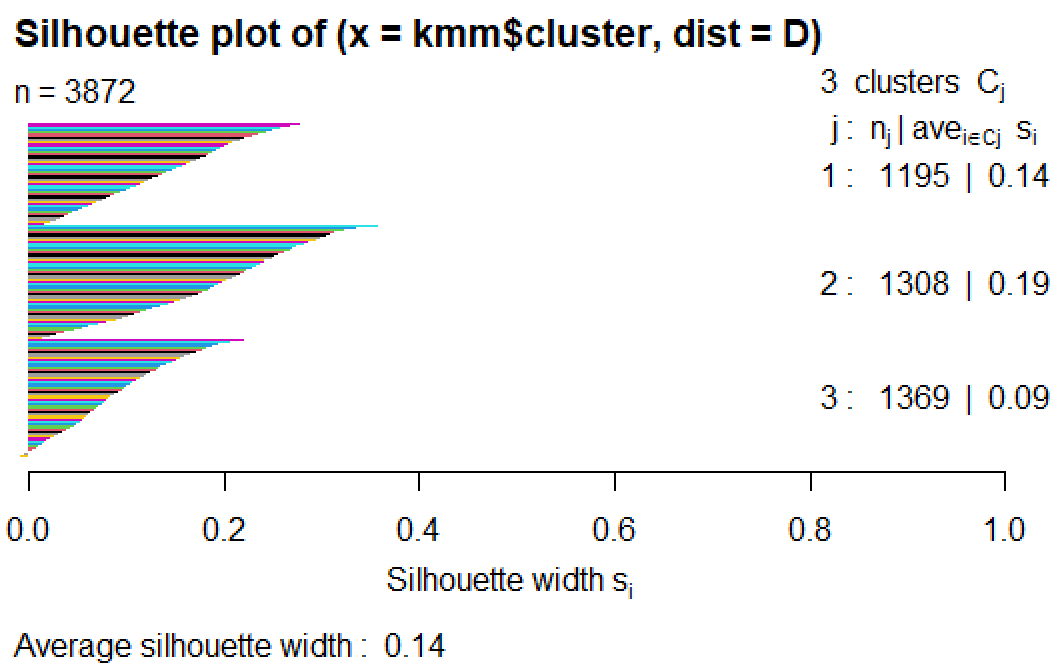
The silhouette approach will be thoroughly detailed in the chapter cluster validation statistics. In a nutshell, it assesses the quality of a grouping. In other words, it determines how well each object fits within its cluster. A large average silhouette width suggests that the clustering is effective.

The silhouette method calculates the average silhouette of observations for various k values. The ideal number of clusters k is the one that maximizes the average silhouette across a range of k values.

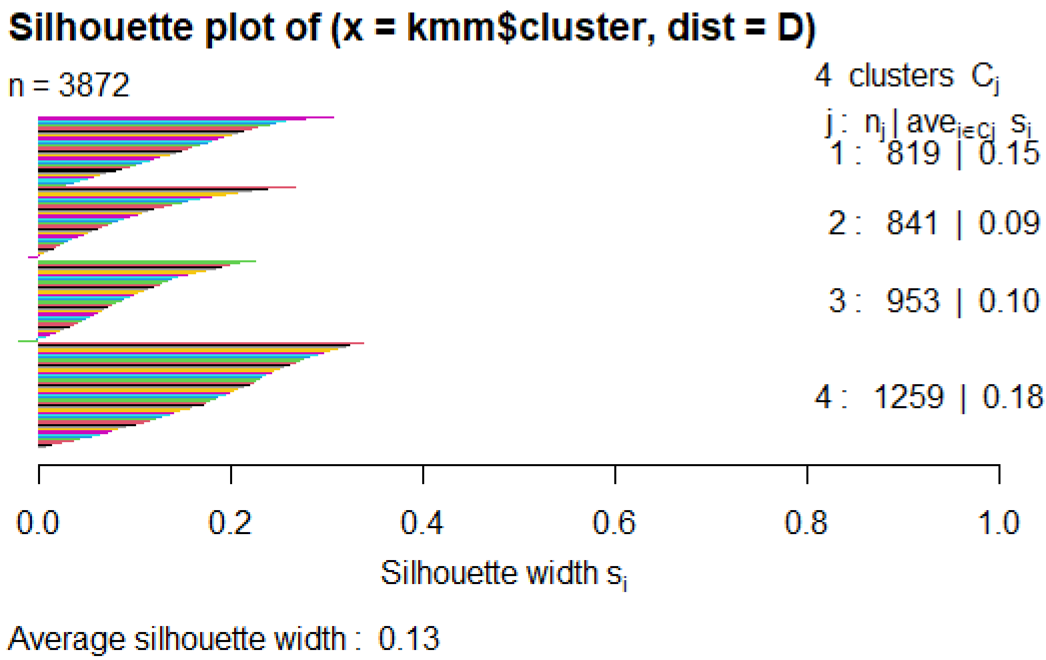
```
#silhouette 2

kmm <- kmeans(scale_data, centers = 2, nstart = 100)


D <- daisy(scale_data)


plot(silhouette(kmm$cluster, D), col=1:8, border=NA)
```




```
#silhouette 3
kmm <- kmeans(scale_data, centers = 3, nstart = 100)

D <- daisy(scale_data)

plot(silhouette(kmm$cluster, D), col=1:8, border=NA)
```




```
#silhouette 4

kmm <- kmeans(scale_data, centers = 4, nstart = 100)


D <- daisy(scale_data)


plot(silhouette(kmm$cluster, D), col=1:8, border=NA)
```



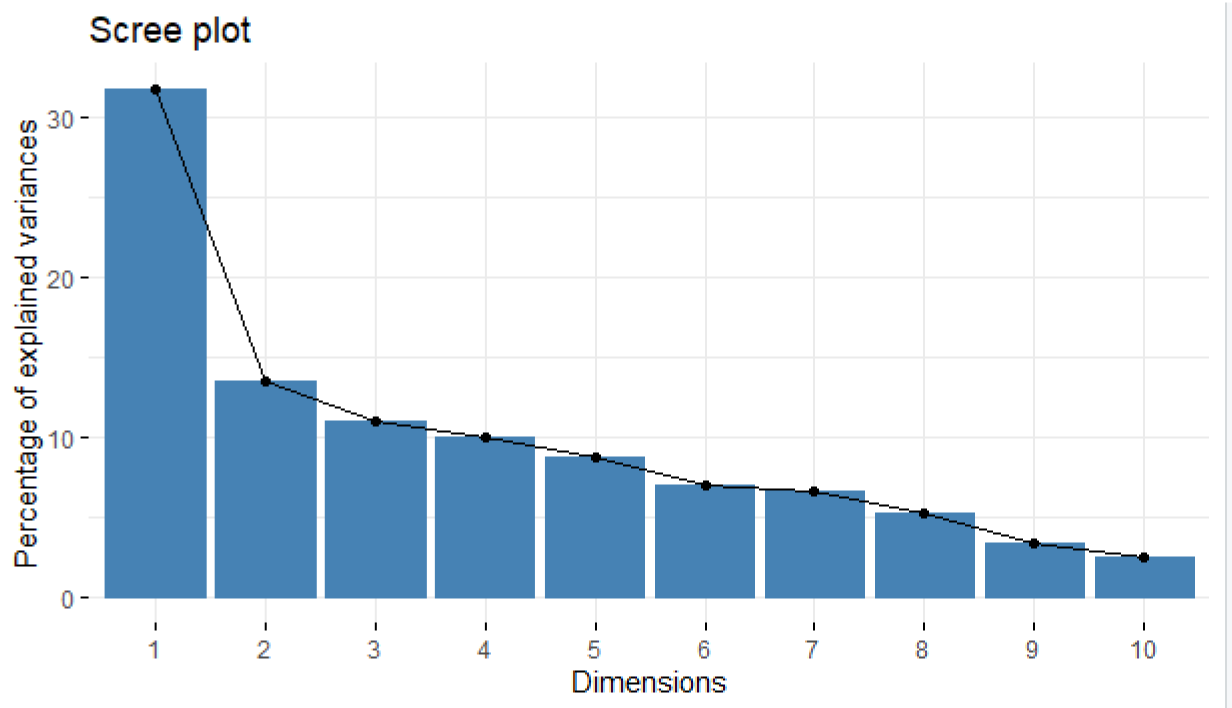
res.pca <- prcomp(outlier_remove_data[,c(1:11)], center = TRUE,scale. = TRUE)

summary(res.pca)

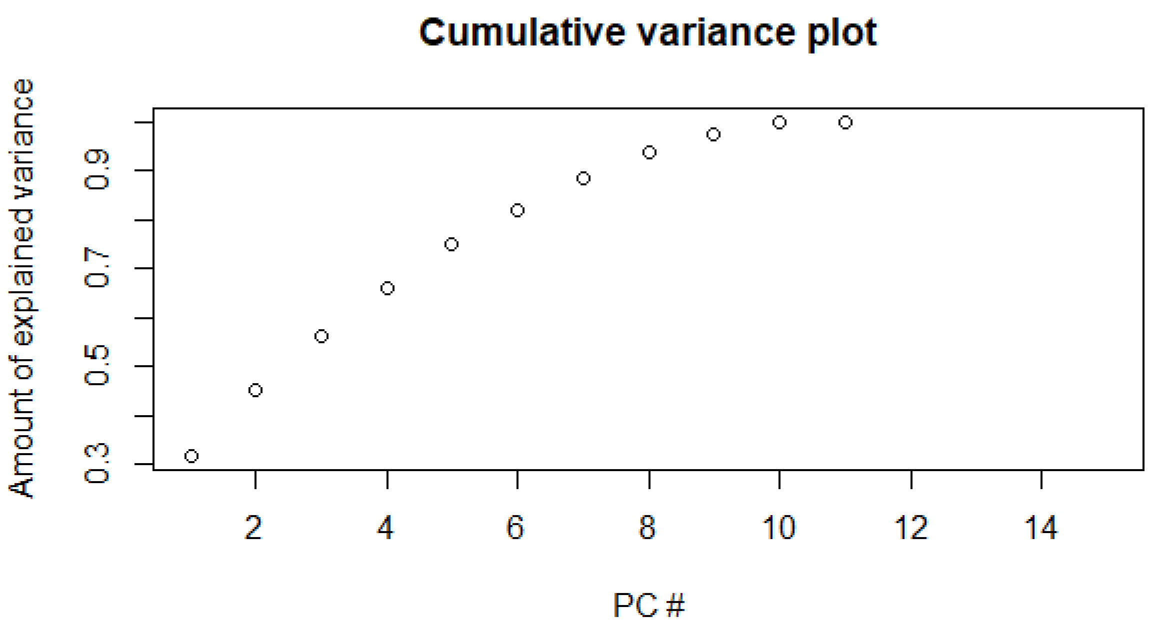
```
> #silhouette 4
> kmm <- kmeans(scale_data, centers = 4, nstart = 100)
> D <- daisy(scale_data)
> plot(silhouette(kmm$cluster, D), col=1:8, border=NA)
> res.pca <- prcomp(outlier_remove_data[,c(1:11)], center = TRUE,scale. = TRUE)
> summary(res.pca)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8
Standard deviation     1.8695 1.2169 1.1021 1.04712 0.98582 0.87547 0.85540 0.76144
Proportion of Variance 0.3177 0.1346 0.1104 0.09968 0.08835 0.06968 0.06652 0.05271
Cumulative Proportion  0.3177 0.4524 0.5628 0.66246 0.75081 0.82049 0.88701 0.93971
                           PC9    PC10    PC11
Standard deviation     0.61367 0.52357 0.11147
Proportion of Variance 0.03424 0.02492 0.00113
Cumulative Proportion  0.97395 0.99887 1.00000
> 
```

library("factoextra")

fviz_eig(res.pca)




```
cumpro <- cumsum(res.pca$sdev^2 / sum(res.pca$sdev^2))
plot(cumpro[0:15], xlab = "PC #", ylab = "Amount of explained variance", main = "Cumulative variance plot")
```



abline(v = 9, col="blue", lty=5)

abline(h = 0.97395, col="blue", lty=5)

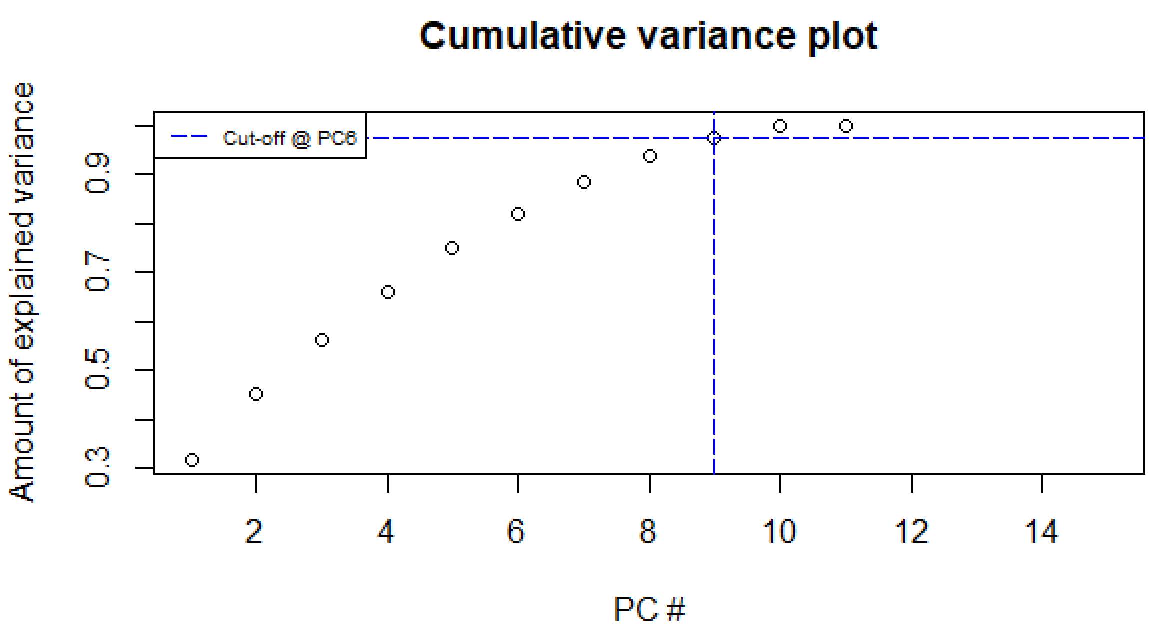
legend("topleft", legend=c("Cut-off @ PC6"),

col=c("blue"), lty=5, cex=0.6)

result <- NbClust (data = res.pca$x[,c(1:9)], distance = "euclidean", min.nc = 2, max.nc = 9, method = 'complete', index = "ch") #result

print(result$Best.nc)

```
> print(result$Best.nc)
Number_clusters     Value_Index
         2.0000        753.5564
> wss <- sapply(1:4,
+                function(k){kmeans(res.pca$x[,c(1:9)], k, nstart=50,iter.max = 100 )$to
t.withinss})
> |
```



wss <- sapply(1:4,

function(k){kmeans(res.pca$x[,c(1:9)], k, nstart=50,iter.max = 100)$tot.withinss})

wss

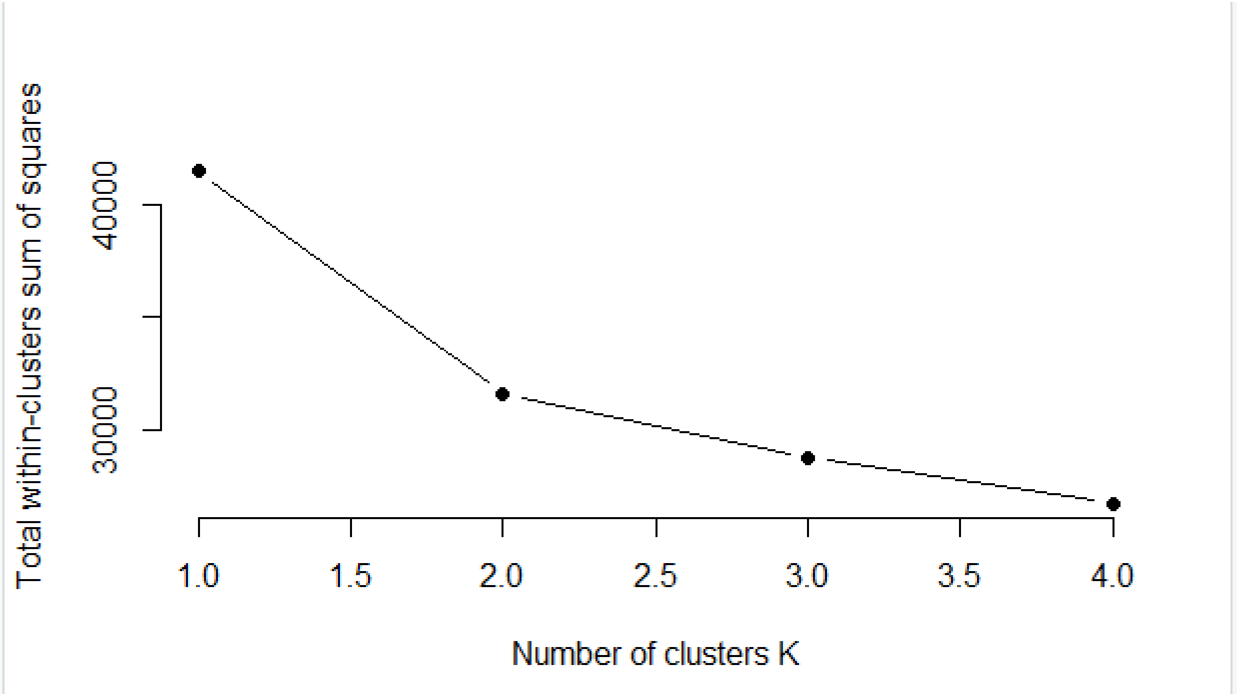
```
> wss
[1] 41471.75 31616.72 28782.25 26730.04
```

plot(1:4, wss,

type="b", pch = 19, frame = FALSE,

xlab="Number of clusters K",

ylab="Total within-clusters sum of squares")



• K-means analysis is performed for each k attempt.

K-means clustering is a fundamental unsupervised machine learning approach that is widely used. Unsupervised algorithms, on the other hand, infer from datasets exclusively only on input vectors, with no regard for known or labeled outcomes. The purpose of K-means, according to Andrey Bulezy, is simple: group comparable data points together to identify hidden patterns. To achieve this goal, K-means analyses a dataset for a predetermined number (k) of clusters.

set.seed(240) # Setting seed

kmeans.re <- kmeans(scale_data, centers = 2, nstart = 100) #scale_data

kmeans.re

 # Cluster identification for

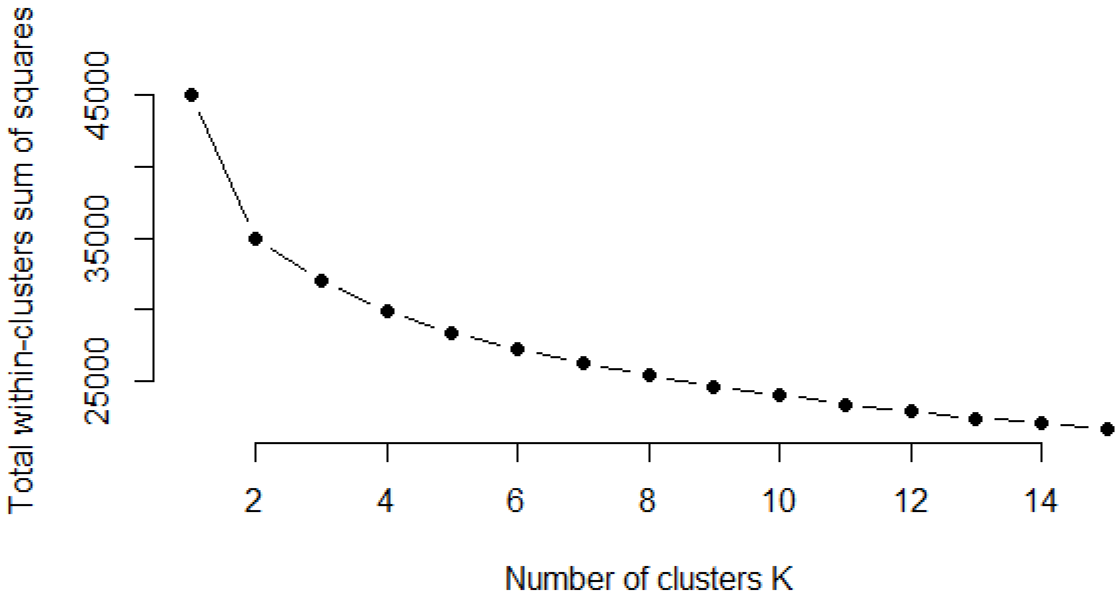
 # each observation

```
kmeans.re$cluster

# Confusion Matrix
cm <- table(scale_data$quality, kmeans.re$cluster)
cm

kmeans.re <- kmeans(scale_data, centers = 3, nstart = 100)
kmeans.re

# Cluster identification for
# each observation
kmeans.re$cluster

# Confusion Matrix
cm <- table(scale_data$quality, kmeans.re$cluster)
cm

kmeans.re <- kmeans(scale_data, centers = 4, nstart = 100)
kmeans.re

# Cluster identification for
# each observation
kmeans.re$cluster
```

 # Confusion Matrix

cm <- table(scale_data$quality, kmeans.re$cluster)

cm



• Evaluating the outputs against the 12th column and defining the final "winning" cluster scenario with a brief description of the evaluation indices PCA and its performance

cm <- table(scale_data$quality, kmeans.re$cluster)

cm

when evaluating the output against the 12th Colum is what we can see is when it comes to two clusters there is no significant any class Like classes which has name by 5 6 7 and 8 cannot be distinguished by the two clusters but there is one thing that we can say original class 8 is not much in the cluster one that means there is only a 19 instances are there in the cluster 1 but almost all the other instances in the cluster 2 therefore if instance is classified as class 1 there is a high probability would not be class 8 that is the only thing that we can say and similarly we can say the class 7 as well but the significant low compared to the previous previous class 8 but class 7 also like high probable to be in cluster 2 and when it comes to the other classes those are not significantly cluster. Those are the only two conclusions that we can take from 2 cluster model.

Similarly when we consider the model with three clusters, the original class 8 has a higher significant to be there in the cluster 1 and the original class 5 higher probability not to be in class 1. So those are 3 significant when compare in Cluster 3 in here but when it comes class 6 and 7

Its hard to predict which cluster it would be.

As we discussed for 2 and 3 cluster setup when we consider the 4 Cluster setup so the original class 8 has a lower probability to be in cluster 1 2 and 4 but the higher probability is there to be there at the cluster 3 similarly when like in the 3 cluster setup even in the caster 4 setup the original class 5 has a lower probability to be in the cluster 3 which means the original classes 8 and 5 has a distinguishable features compare to class 6 and 7 so that is only conclusion that we can take from the cluster setup and as the final conclusion we cannot take a proper decision just based on the confusion Matrix so it is better to consider NV clustering or else elbow curve to decide the Winning Cluster.

so when we considered NV cluster we insert a like we can define how many number of clusters we want to check and what is the distance Matrix which define and there are a couple of height parameters as well so those are the main two hyperparameters which we have to consider and there we have mentioned distance as the Euclidean distance and minimum number of clusters is 2 and the maximum number of clusters we going to check is 15. So there we can do is train the model itself will check for the best number of clusters based on the calculation method that defined from that we could have obtained the results as the best number of clusters, for this setup is 2 clusters that has been taken directly from the model itself and when we consider the elbow curve. There we have to do if we have to calculate the total distance when we are consider the elbow curve like the total cost of the method then we can do is we can plot it and see how the cost varies like the cost will eventually reduce when the number of clusters are height so that means the total Central to be done so that would reduce the number of classes with the problem is once we plot. If you can visualize the something like a folded elbow then we but we can say is that specific point where the graph dent is the maximum number of the best number of clusters when we consider the elbow curve what we can actually visualize at 2 clusters there is a slight bend in the curve even though it is not significant in curve but it is visible for us to take an elevation of the elbow curve

we have to consider PCA what we have to do is we have used PRCOMP function there we defined number of columns and the data set to do PCA - principal components Analyze so we can use scale value as true and it will be scaled the we use as PCA summary we can visualize the results of PCA so there we can see is we get standard deviation of each principle component then the proportion of variants and cumulative proportion as well so cumulation of the variants so there we visualized highest proportion is 0.7177 and followed by 0.1348 like it goes to 11th principle component 0.00113 what we hv to do is which we want to get cumulative value greater than 96% that means 0.96 so when we consider cumulative proportion at principle component 9th we get 0.9739 and in 8th 0.987 therefore we consider atleast 9 priniple components to achieve more than 96%, so when we train the model which we have used the 1-9 principal component because of that.

When we consider the principle components data and normal data. principal component data has a slight drop in the wss and the other measurements which means using principal component has a slight improvement compare to other data which we are using the row data.

02nd Objective – (MLP)

Neural networks are made up of simple input/output units known as neurons (inspired by neurons of the human brain). These input/output units are linked together, and each connection is assigned a weight. Neural networks are adaptable and can be used for classification as well as regression. In this post, we'll look at how neural networks can be used to solve regression problems.

Regression analysis aids in the establishment of a relationship between a dependent variable and one or more independent variables. Only when the regression equation is a good fit for the data do regression models operate successfully. Most regression models will not precisely fit the data. Although neural networks are sophisticated and computationally expensive, they are adaptable and can dynamically select the optimum form of regression; if that is insufficient, hidden layers can be added to improve prediction.

```
library(tsDyn)
```

```
test_data = tail(UoW_load, n =20)

train_data = head(UoW_load, n =480)
```

```
test_data

min_max_norm <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

#When the data is not in a specific range it would be hard optimize the moidel
scaled_data <- as.data.frame(lapply(UoW_load[2:4], min_max_norm))
scaled_data <- cbind(scaled_data, UoW_load[c(4)])

names(scaled_data)[1] <- "var1"
names(scaled_data)[2] <- "var2"
names(scaled_data)[3] <- "scaled_pred"
names(scaled_data)[4] <- "Pred"

scaled_test_data = tail(scaled_data, n =20)
scaled_train_data = head(scaled_data, n =480)

install.packages("neuralnet")
library(neuralnet)

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

min1 <- min(scaled_data[4])
max1 <- max(scaled_data[4])
```

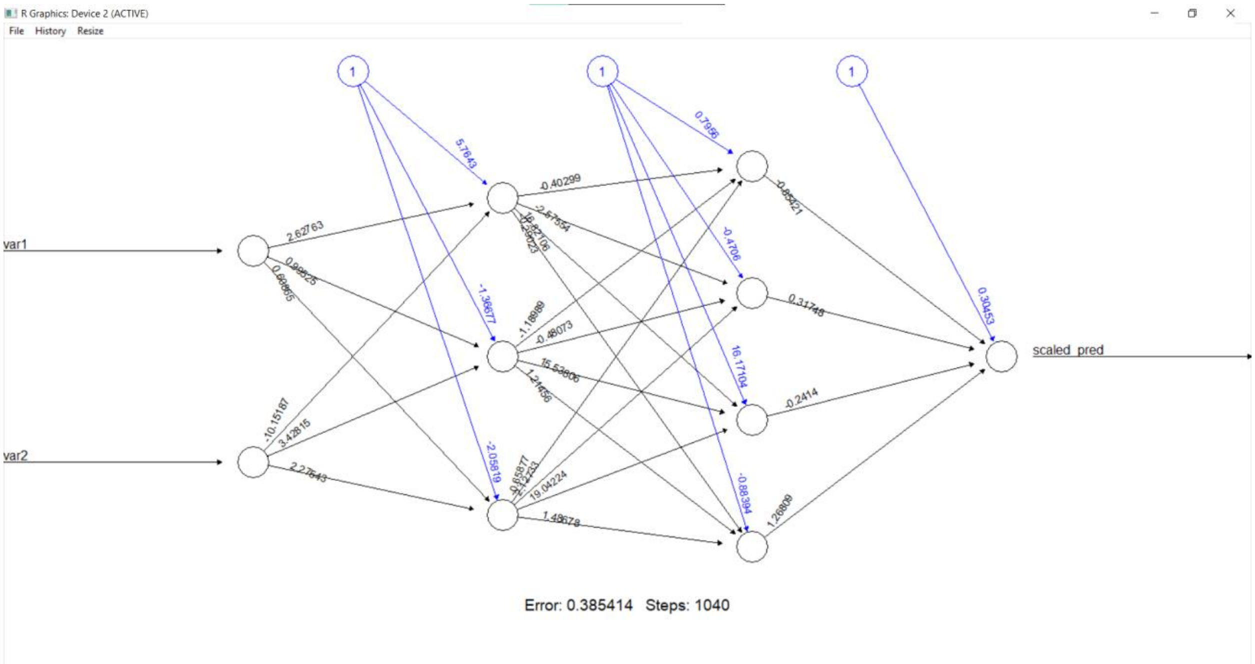
```
#Model 1
NN_model_1<- neuralnet (scaled_pred~var1+var2, hidden=c(3,4) , data = scaled_train_data
                        , linear. output=TRUE)
plot (NN_model_1)
#Evaluation model performance
model1Result <- predict (NN_model_1, scaled_test_data [1:2])
model1Result
renormormalized_prediction_value1 <- unnormalizing (model1Result, min1, max1)
renormormalized_prediction_value1 = unlist (as. list(renormormalized_prediction_value1), recursive=F)
renormormalized_prediction_value1

#Model 2
NN_model_2<- neuralnet(scaled_pred~var1+var2 ,hidden=c(10,30,10) , data = scaled_train_data
                      ,linear.output=TRUE)
plot(NN_model_2)
#Evaluation model performance
model2Result <- predict(NN_model_2, scaled_test_data[1:2])
model2Result
renormormalized_prediction_value2 <- unnormalizing(model2Result, min1, max1)
renormormalized_prediction_value2 = unlist(as.list(renormormalized_prediction_value2),recursive=F)
renormormalized_prediction_value2

#Model 3
NN_model_3<- neuralnet(scaled_pred~var1+var2 ,hidden=c(10,50,25,10) , data = scaled_train_data
```

```
            ,linear.output=TRUE)
plot(NN_model_3)
#Evaluation model performance
model3Result <- predict(NN_model_3, scaled_test_data[1:2])
model3Result
renormormalized_prediction_value3 <- unnormalizing(model3Result, min1, max1)
renormormalized_prediction_value3 = unlist(as.list(renormormalized_prediction_value3),recursive=F)
renormormalized_prediction_value3



mod1.nnet<- nnetTs(scaled_train_data[c(3)],m=5, size=3,steps=30)
mod1.nnet
renormormalized_prediction_value4 <- unnormalizing(predict(mod1.nnet,steps=5,n.ahead=20), min1,
max1)
renormormalized_prediction_value4 = unlist(as.list(renormormalized_prediction_value4),recursive=F)
renormormalized_prediction_value4
plot.ts(renormormalized_prediction_value4)
plot.ts(test_data[c(2)])

mod2.nnet<- nnetTs(scaled_train_data[c(3)], m = 4,  size=3,steps=20)
mod2.nnet
renormormalized_prediction_value5 <- unnormalizing(predict(mod2.nnet,steps=5,n.ahead=20), min1,
max1)
renormormalized_prediction_value5 = unlist(as.list(renormormalized_prediction_value5),recursive=F)
renormormalized_prediction_value5
plot.ts(renormormalized_prediction_value5)


mod3.nnet<- nnetTs(scaled_train_data[c(3)], m = 5, size=8,steps=10)
```

```
mod3.nnet

renormormalized_prediction_value6 <- unnormalizing(predict(mod3.nnet,steps=5,n.ahead=20), min1,
max1)

renormormalized_prediction_value6 = unlist(as.list(renormormalized_prediction_value6),recursive=F)

renormormalized_prediction_value6

plot.ts(renormormalized_prediction_value6)


install.packages("Metrics")

library("Metrics")



y_test = as.list(test_data[4])


#RMSE (Root_mean_Square_Deviation)

rmse(list(renormormalized_prediction_value1),as.list(test_data[4]))

#MSE

MSE(list(renormormalized_prediction_value1),as.list(test_data[4]))

#MAPE

MAPE(list(renormormalized_prediction_value1),as.list(test_data[4]))




# Plot regression line

plot(test_data[4], renormormalized_prediction_value1, col = "red",

    main = 'Real vs Predicted')

abline(0, 1, lwd = 2)



plot.ts(x = test_data[4] ,y = renormormalized_prediction_value1)

abline(0, 1, lwd = 2,col = "red")
```

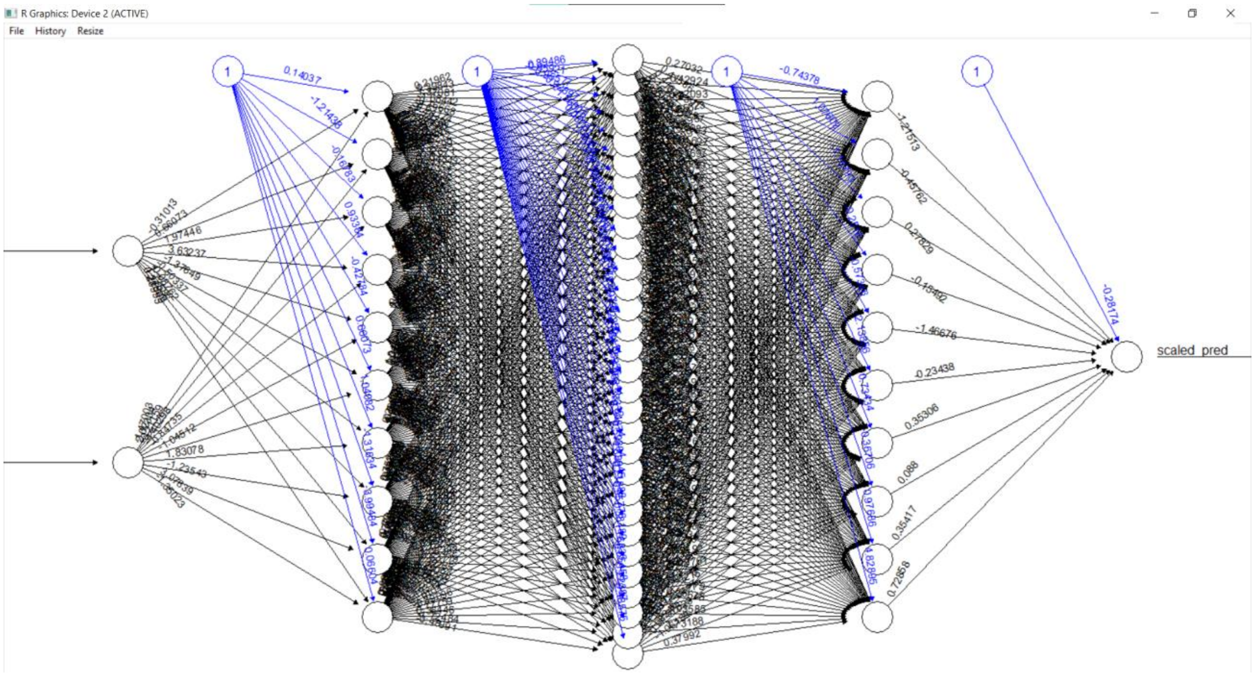
```
  as.zoo.data.frame zoo
> test_data = tail(UoW_load, n =20)
> train_data = head(UoW_load, n =480)
> test_data
# A tibble: 20 x 4
   Dates               `09:00` `10:00` `11:00`
   <dttm>                <dbl>   <dbl>   <dbl>
 1 2019-04-26 00:00:00    87.8    90.8    92.4
 2 2019-04-27 00:00:00    77.8    78.2    80.8
 3 2019-04-28 00:00:00    52.6    51.8    51.8
 4 2019-04-29 00:00:00   101     109     111.
 5 2019-04-30 00:00:00   107.    108     108.
 6 2019-05-01 00:00:00   102.    103.    106.
 7 2019-05-02 00:00:00   102.    101.    103
 8 2019-05-03 00:00:00    98.8   102.    106.
 9 2019-05-04 00:00:00    78.6    78.8    78.8
10 2019-05-05 00:00:00    55      56      53.6
11 2019-05-06 00:00:00    80.8    79.6    78.4
12 2019-05-07 00:00:00   102.    106     106.
13 2019-05-08 00:00:00   102.    107.    115.
14 2019-05-09 00:00:00   100.    104.    109.
15 2019-05-10 00:00:00    89      97.2    96.2
16 2019-05-11 00:00:00    82.6    81.2    82
17 2019-05-12 00:00:00    54.8    53      52.8
18 2019-05-13 00:00:00    96.8    96.4   101.
19 2019-05-14 00:00:00   105.    106.    108.
20 2019-05-15 00:00:00    85.2    73      76.2
> 
```

Trained Neural Network Modal 01



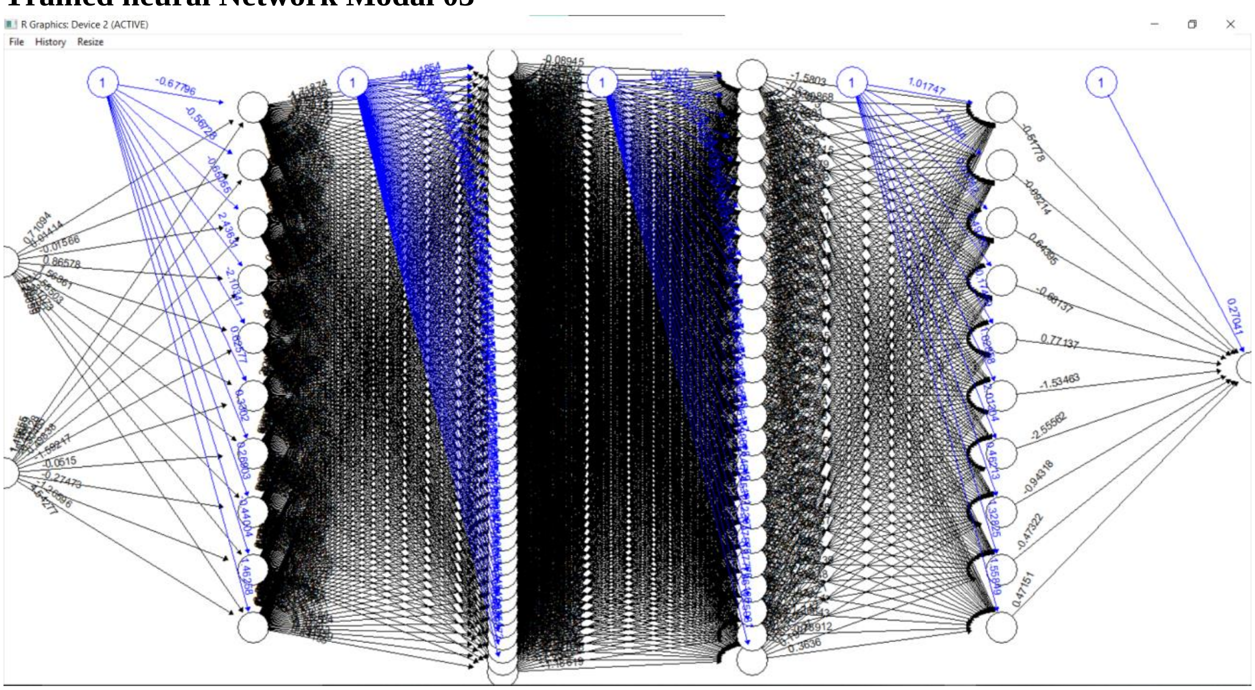

```
> #Evaluation model performance
> model1Result <- predict(NN_model_1, scaled_test_data[1:2])
> model1Result
          [,1]
481 0.41191965
482 0.28253852
483 0.05628259
484 0.59683934
485 0.59863599
486 0.54914960
487 0.53366152
488 0.53359778
489 0.28931095
490 0.08190456
491 0.30114460
492 0.57186862
493 0.57948735
494 0.55553548
495 0.46982793
496 0.31864126
497 0.06572784
498 0.48021741
499 0.57495173
500 0.25908309
```

Trained neural Network Modal 02




```
> renormormalized_prediction_value1 <- unnormalizing(model1Result, min1, max1)
> renormormalized_prediction_value1 = unlist(as.list(renormormalized_prediction_value1),recursive=F)
> renormormalized_prediction_value1
 [1]  92.76971  78.77067  54.28978 112.77802 112.97241 107.61799 105.94218 105.93528  79.50345  57.06207  80.78385 110.07618 110.90053 108.
[15]  99.03538  82.67698  55.31175 100.15952 110.40978  76.23279
> #Model 2
> NN_model_2<- neuralnet(scaled_pred~var1+var2 ,hidden=c(10,30,10) , data = scaled_train_data
+                        ,linear.output=TRUE)
> plot(NN_model_2)
> 
```

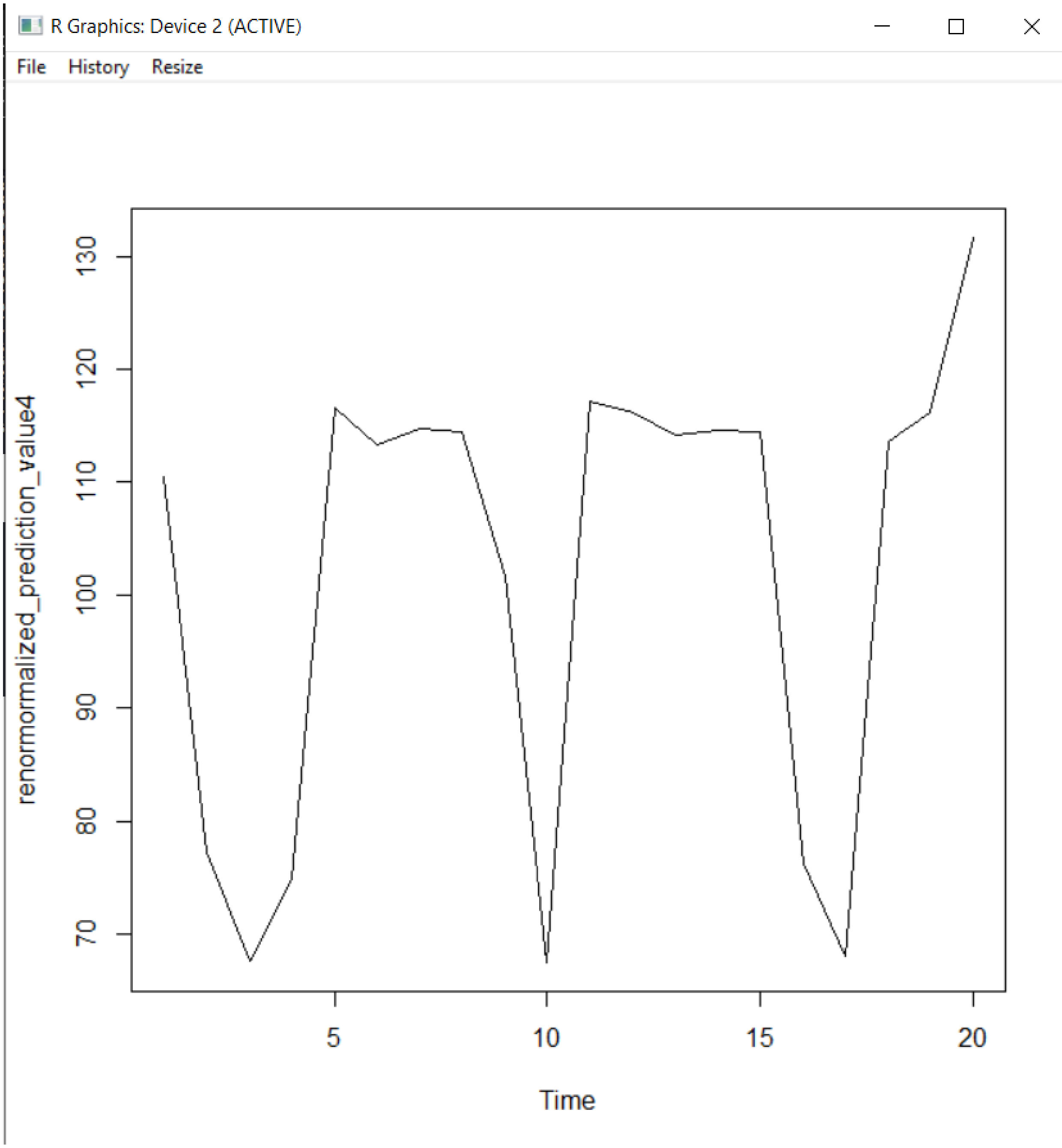
Trained neural Network Modal 03

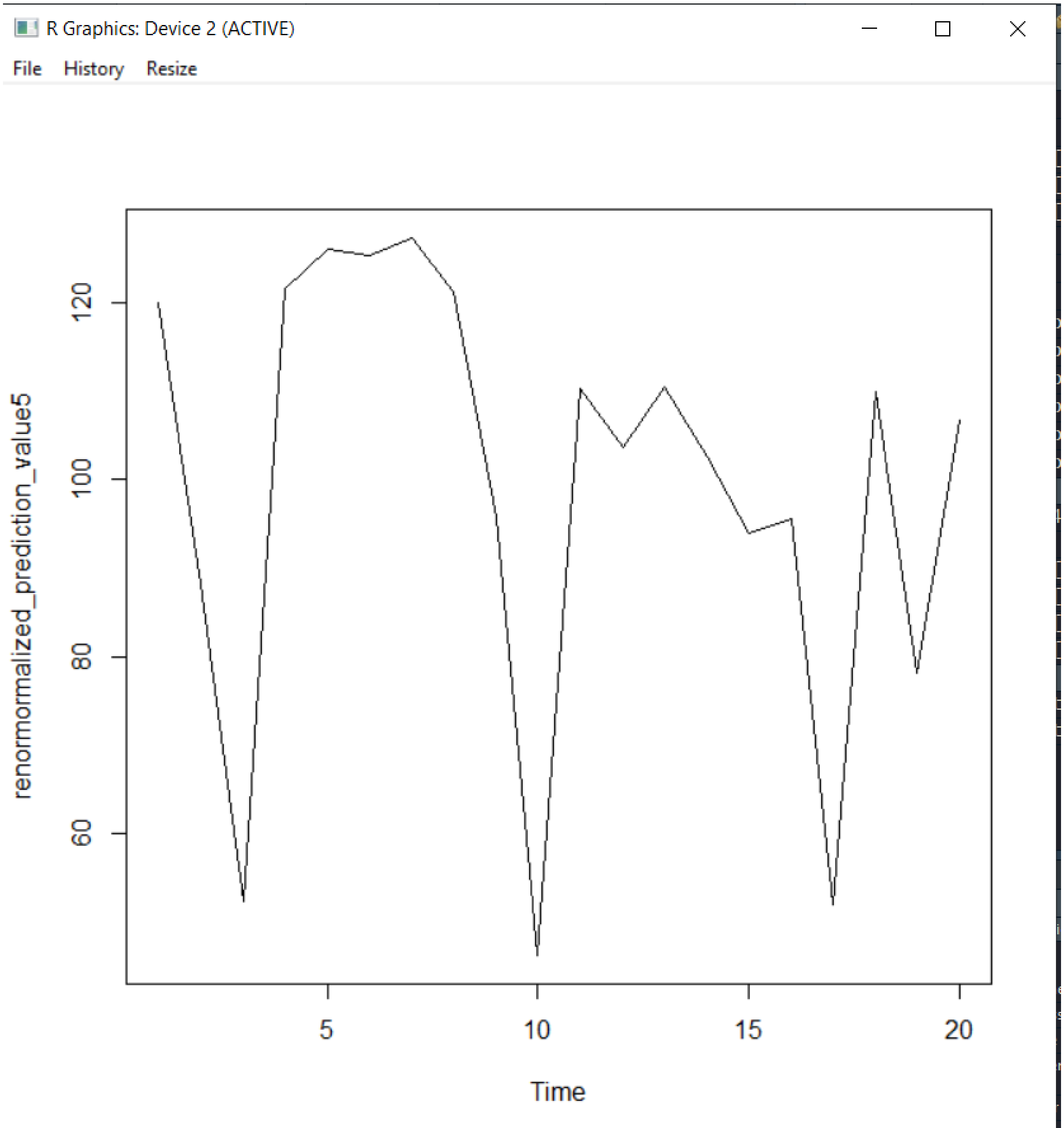


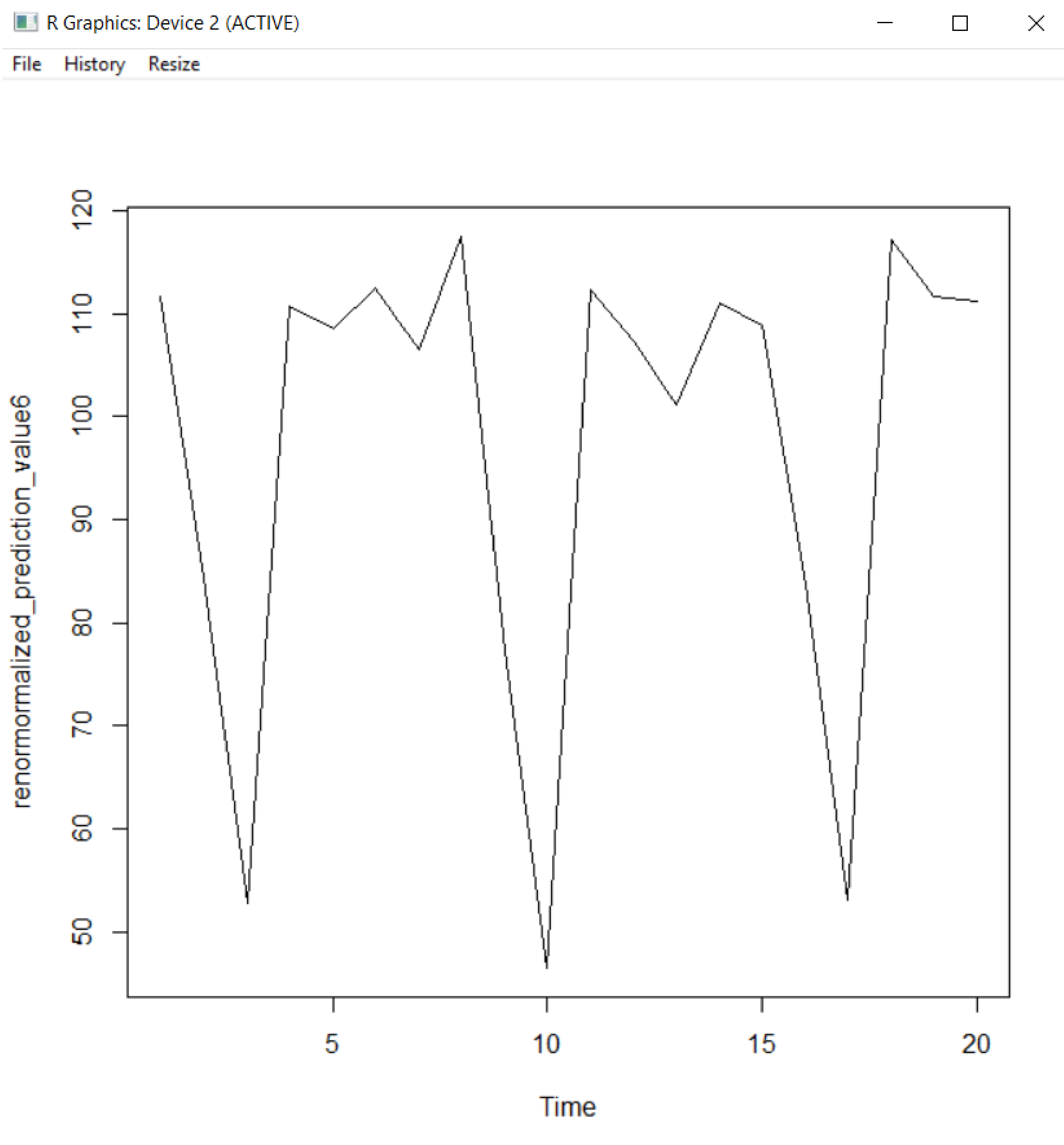
```
> renormormalized_prediction_value2 = unlist(as.list(renormormalized_prediction_value2),recursive=F)
> renormormalized_prediction_value2
 [1]  92.81786  79.34796  53.14788 112.57352 113.06154 107.53750 105.82662 105.80893  80.06348  56.43725  81.32450 109.99702 110.82681 108.20181
[15]  98.82659  83.14605  54.49595 100.03938 110.40624  77.01001
> #Model 3
> NN_model_3<- neuralnet(scaled_pred~var1+var2 ,hidden=c(10,50,25,10) , data = scaled_train_data
+                        ,linear.output=TRUE)
> #Model 3
> NN_model_3<- neuralnet(scaled_pred~var1+var2 ,hidden=c(10,50,25,10) , data = scaled_train_data
+                        ,linear.output=TRUE)
> plot(NN_model_3)
> #Evaluation model performance
> model3Result <- predict(NN_model_3, scaled_test_data[1:2])
> model3Result
          [,1]
481 0.41043411
482 0.28668542
483 0.03974234
484 0.59535511
485 0.60086306
486 0.54878107
487 0.53294490
488 0.53210819
489 0.29311675
490 0.07525467
491 0.30430372
492 0.57141929
493 0.57916712
494 0.55456715
495 0.46688161
496 0.32089534
497 0.05388895
498 0.47814923
499 0.57597690
500 0.26388596
> |
```

```
mod1.nnet<- nnetTs(scaled_train_data[c(3)],m=5, size=3,steps=30)
mod1.nnet
renormormalized_prediction_value4 <- unnormalizing(predict(mod1.nnet,steps=5,n.ahead=20), min1, max1)
renormormalized_prediction_value4 = unlist(as.list(renormormalized_prediction_value4),recursive=F)
renormormalized_prediction_value4
plot.ts(renormormalized_prediction_value4)
plot.ts(test_data[c(2)])

mod2.nnet<- nnetTs(scaled_train_data[c(3)], m = 4,  size=3,steps=20)
mod2.nnet
renormormalized_prediction_value5 <- unnormalizing(predict(mod2.nnet,steps=5,n.ahead=20), min1, max1)
renormormalized_prediction_value5 = unlist(as.list(renormormalized_prediction_value5),recursive=F)
renormormalized_prediction_value5
plot.ts(renormormalized_prediction_value5)


mod3.nnet<- nnetTs(scaled_train_data[c(3)], m = 5, size=8,steps=10)
mod3.nnet
renormormalized_prediction_value6 <- unnormalizing(predict(mod3.nnet,steps=5,n.ahead=20), min1, max1)
renormormalized_prediction_value6 = unlist(as.list(renormormalized_prediction_value6),recursive=F)
renormormalized_prediction_value6
plot.ts(renormormalized_prediction_value6)

```

According to the trained 03 modals this Neural Network is work for small neural network to large number of Neural networks.


R Graphics: Device 2 (ACTIVE)
File History Resize
renormormalized_prediction_value4
70
80
90
100
110
120
130
5
10
15
20
Time

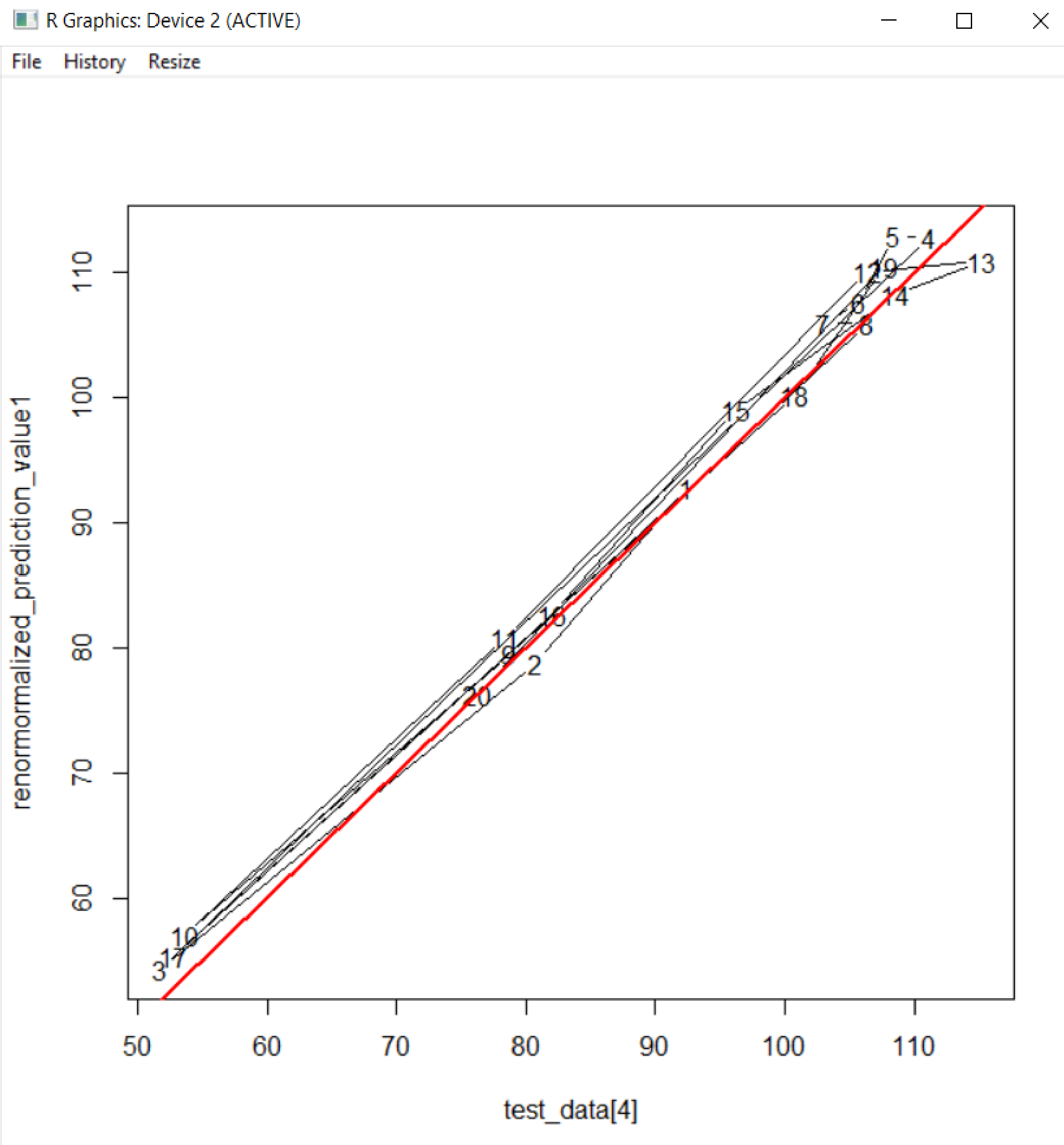

R Graphics: Device 2 (ACTIVE)
File History Resize
renormormalized_prediction_value5
60
80
100
120
5
10
15
20
Time


R Graphics: Device 2 (ACTIVE)
File History Resize
renormormalized_prediction_value6
120
110
100
90
80
70
60
50
5
10
15
20
Time

```r

install.packages("Metrics")
library("Metrics")


y_test = as.list(test_data[4])

#RMSE (Root_mean_Square_Deviation)
rmse(list(renormormalized_prediction_value1),as.list(test_data[4]))
#MSE
MSE(list(renormormalized_prediction_value1),as.list(test_data[4]))
#MAPE
MAPE(list(renormormalized_prediction_value1),as.list(test_data[4]))

```

- Below Display the Regression Line

```r

# Plot regression line
plot(test_data[4], renormormalized_prediction_value1, col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)


plot.ts(x = test_data[4] ,y = renormormalized_prediction_value1)
abline(0, 1, lwd = 2,col = "red")

```



Appendix – Objective 01

```r
boxplot(whitewine_v2$`fixed acidity`)
boxplot(whitewine_v2$`volatile acidity`)
boxplot(whitewine_v2$`citric acid`)
boxplot(whitewine_v2$`residual sugar`)
boxplot(whitewine_v2$chlorides)
boxplot(whitewine_v2$`free sulfur dioxide`)
boxplot(whitewine_v2$`total sulfur dioxide`)
boxplot(whitewine_v2$density)
boxplot(whitewine_v2$pH)
boxplot(whitewine_v2$sulphates)
boxplot(whitewine_v2$alcohol)


detect_outlier <- function(x) {

  # calculate first quantile
  Quantile1 <- quantile(x, probs=.25)

  # calculate third quantile
  Quantile3 <- quantile(x, probs=.75)

  # calculate inter quartile range
  IQR = Quantile3-Quantile1

  # return true or false
  x > Quantile3 + (IQR*1.5) | x < Quantile1 - (IQR*1.5)
}

# create remove outlier function
remove_outlier <- function(dataframe,
                           columns=names(dataframe)) {

  # for loop to traverse in columns vector
  for (col in columns) {
```

```
    # remove observation if it satisfies outlier function
    dataframe <- dataframe[!detect_outlier(dataframe[[col]]), ]
  }

  # return dataframe
  print("Remove outliers")
  print(dataframe)
}

outlier_remove_data <- remove_outlier(Whitewine_v2, c('fixed acidity',
'volatile acidity','citric acid','residual sugar','chlorides',
'free sulfur dioxide','total sulfur dioxide','density',
'pH','sulphates','alcohol'))
print(outlier_remove_data)

boxplot(outlier_remove_data$`fixed acidity`)
boxplot(outlier_remove_data$`volatile acidity`)
boxplot(outlier_remove_data$`citric acid`)
boxplot(outlier_remove_data$`residual sugar`)
boxplot(outlier_remove_data$chlorides)
boxplot(outlier_remove_data$`free sulfur dioxide`)
boxplot(outlier_remove_data$`total sulfur dioxide`)
boxplot(outlier_remove_data$density)
boxplot(outlier_remove_data$pH)
boxplot(outlier_remove_data$sulphates)
boxplot(outlier_remove_data$alcohol)

scale_data <- as.data.frame(scale(outlier_remove_data)) #z score scale
scale_data$quality <- outlier_remove_data$quality
print(scale_data)

# Installing Packages
install.packages("ClusterR")
install.packages("cluster")
```

```
# Loading package
library(ClusterR)
library(cluster)


# Fitting K-Means clustering Model
# to training dataset
set.seed(240) # Setting seed
kmeans.re <- kmeans(scale_data, centers = 2, nstart = 100) #scale_data
kmeans.re

# Cluster identification for
# each observation
kmeans.re$cluster

# Confusion Matrix
cm <- table(scale_data$quality, kmeans.re$cluster)
cm

kmeans.re <- kmeans(scale_data, centers = 3, nstart = 100)
kmeans.re

# Cluster identification for
# each observation
kmeans.re$cluster

# Confusion Matrix
cm <- table(scale_data$quality, kmeans.re$cluster)
cm

kmeans.re <- kmeans(scale_data, centers = 4, nstart = 100)
kmeans.re

# Cluster identification for
# each observation
```

```
kmeans.re$cluster

# Confusion Matrix
cm <- table(scale_data$quality, kmeans.re$cluster)
cm

# Installing Packages
install.packages("NbClust")
library("NbClust")

library(cluster)
library(factoextra)

result <- NbClust(data = scale_data, distance = "euclidean", min.nc = 2,
                  max.nc = 15, method = 'complete', index = "ch") #result
print(result$Best.nc)

#Elbow Method for finding the optimal number of clusters
set.seed(123)
# Compute and plot wss for k = 2 to k = 15.
k.max <- 15
data <- scale_data
wss <- sapply(1:k.max,
         function(k){kmeans(data, k,nstart=50,iter.max = 100)$tot.withinss})
wss
plot(1:k.max, wss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")

#silhouette 2
kmm <- kmeans(scale_data, centers = 2, nstart = 100)

D <- daisy(scale_data)
```

```
plot(silhouette(kmm$cluster, D), col=1:8, border=NA)

#silhouette 3
kmm <- kmeans(scale_data, centers = 3, nstart = 100)

D <- daisy(scale_data)

plot(silhouette(kmm$cluster, D), col=1:8, border=NA)

#silhouette 4
kmm <- kmeans(scale_data, centers = 4, nstart = 100)

D <- daisy(scale_data)

plot(silhouette(kmm$cluster, D), col=1:8, border=NA)



res.pca <- prcomp(outlier_remove_data[,c(1:11)], center = TRUE,scale. = TRUE)

summary(res.pca)

install.packages("factoextra")
library("factoextra")
fviz_eig(res.pca)



cumpro <- cumsum(res.pca$sdev^2 / sum(res.pca$sdev^2))
plot(cumpro[0:15], xlab = "PC #", ylab = "Amount of explained variance",
     main = "Cumulative variance plot")
abline(v = 9, col="blue", lty=5)
abline(h = 0.97395, col="blue", lty=5)
legend("topleft", legend=c("Cut-off @ PC6"),
       col=c("blue"), lty=5, cex=0.6)



result <- NbClust(data = res.pca$x[,c(1:9)], distance = "euclidean",
                  min.nc = 2, max.nc = 9, method = 'complete', index = "ch") #result
print(result$Best.nc)

wss <- sapply(1:4,
              function(k){kmeans(res.pca$x[,c(1:9)], k,
                                 nstart=50,iter.max = 100 )$tot.withinss})
wss
plot(1:4, wss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")

kmm <- kmeans(res.pca$x[,c(1:9)], centers = 2, nstart = 100)

D <- daisy(res.pca$x[,c(1:9)])
```

Appendix – Objective 02

```
install.packages("tsDyn")
library(tsDyn)


test_data = tail(UoW_load, n =20)
train_data = head(UoW_load, n =480)

test_data

min_max_norm <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

#when the data is not in a specific range it would be hard optimize the moidel
scaled_data <- as.data.frame(lapply(UoW_load[2:4], min_max_norm))
scaled_data <- cbind(scaled_data, UoW_load[c(4)])

names(scaled_data)[1] <- "var1"
names(scaled_data)[2] <- "var2"
names(scaled_data)[3] <- "scaled_pred"
names(scaled_data)[4] <- "Pred"

scaled_test_data = tail(scaled_data, n =20)
scaled_train_data = head(scaled_data, n =480)

install.packages("neuralnet")
library(neuralnet)

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

min1 <- min(scaled_data[4])
max1 <- max(scaled_data[4])

#Model 1
```

```
NN_model_1<- neuralnet(scaled_pred~var1+var2 ,hidden=c(3,4) , data = scaled_train_data
                         ,linear.output=TRUE)
plot(NN_model_1)
#Evaluation model performance
model1Result <- predict(NN_model_1, scaled_test_data[1:2])
model1Result
renormormalized_prediction_value1 <- unnormalizing(model1Result, min1, max1)
renormormalized_prediction_value1 = unlist(as.list(renormormalized_prediction_value1),
                                           recursive=F)
renormormalized_prediction_value1

#Model 2
NN_model_2<- neuralnet(scaled_pred~var1+var2 ,hidden=c(10,30,10) , data = scaled_train_data
                       ,linear.output=TRUE)
plot(NN_model_2)
#Evaluation model performance
model2Result <- predict(NN_model_2, scaled_test_data[1:2])
model2Result
renormormalized_prediction_value2 <- unnormalizing(model2Result, min1, max1)
renormormalized_prediction_value2 = unlist(as.list(renormormalized_prediction_value2),
                                           recursive=F)
renormormalized_prediction_value2



#Model 3
NN_model_3<- neuralnet(scaled_pred~var1+var2 ,hidden=c(10,50,25,10) , data = scaled_train_data
                       ,linear.output=TRUE)
plot(NN_model_3)
#Evaluation model performance
model3Result <- predict(NN_model_3, scaled_test_data[1:2])
model3Result
renormormalized_prediction_value3 <- unnormalizing(model3Result, min1, max1)
renormormalized_prediction_value3 = unlist(as.list(renormormalized_prediction_value3),
                                           recursive=F)
renormormalized_prediction_value3
```

```r
mod1.nnet<- nnetTs(scaled_train_data[c(3)],m=5, size=3,steps=30)
mod1.nnet
renormormalized_prediction_value4 <- unnormalizing(predict(mod1.nnet,steps=5,n.ahead=20),
                                                    min1, max1)
renormormalized_prediction_value4 = unlist(as.list(renormormalized_prediction_value4),
                                            recursive=F)
renormormalized_prediction_value4
plot.ts(renormormalized_prediction_value4)
plot.ts(test_data[c(2)])

mod2.nnet<- nnetTs(scaled_train_data[c(3)], m = 4,  size=3,steps=20)
mod2.nnet
renormormalized_prediction_value5 <- unnormalizing(predict(mod2.nnet,steps=5,n.ahead=20),
                                                    min1, max1)
renormormalized_prediction_value5 = unlist(as.list(renormormalized_prediction_value5),
                                            recursive=F)
renormormalized_prediction_value5
plot.ts(renormormalized_prediction_value5)


mod3.nnet<- nnetTs(scaled_train_data[c(3)], m = 5, size=8,steps=10)
mod3.nnet
renormormalized_prediction_value6 <- unnormalizing(predict(mod3.nnet,steps=5,n.ahead=20),
                                                    min1, max1)
renormormalized_prediction_value6 = unlist(as.list(renormormalized_prediction_value6),
                                            recursive=F)
renormormalized_prediction_value6
plot.ts(renormormalized_prediction_value6)

install.packages("Metrics")
library("Metrics")


y_test = as.list(test_data[4])

#RMSE
```

```r
rmse(list(renormormalized_prediction_value1),as.list(test_data[4]))
#MSE
MSE(list(renormormalized_prediction_value1),as.list(test_data[4]))
#MAPE
MAPE(list(renormormalized_prediction_value1),as.list(test_data[4]))



# Plot regression line
plot(test_data[4], renormormalized_prediction_value1, col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)


plot.ts(x = test_data[4] ,y = renormormalized_prediction_value1)
abline(0, 1, lwd = 2,col = "red")
```