

Report on Partitioning Clustering and Energy Forecasting

Name: Hasanur Rahman Mohammad

Student ID: w1780941

Module Code: 5DATA002W

Tutor: Mahmoud Aldraimli

Seminar Group: 5CS01

Contents

1	Partitioning Clustering	2
1.1	Pre-Processing the data	2
1.2	Finding the best k using: Nblast, Elbow method, Gap statistics and sillhoutte methods .	2
1.2.1	Nblast	2
1.2.2	Elbow Method	2
1.2.3	Gap Statistics	3
1.2.4	Sillhoutte Method	4
1.3	K-means Clustering investigation	5
1.3.1	Discussing the K-means outputs	5
1.3.2	Sillhoutte Plot	7
1.4	K-means Clustering with PCA	7
1.4.1	Creating the new dataset with PCA	7
1.5	Finding the best k for PCA dataset using: Nblast, Elbow method, Gap statistics and sillhoutte methods	9
1.5.1	Nblast	9
1.5.2	Elbow Method	9
1.5.3	Gap Statistics	10
1.5.4	Sillhoutte Method	10
1.6	K-means Clustering Investigation with PCA	11
1.6.1	Discussing the K-means outputs	11
1.6.2	Sillhoutte Plot	13
1.6.3	Calinski Harabasz index	13
2	Energy Forecasting	13
2.1	Type of input variables used for electricity load forecasting	13
2.2	Normalising the data	13
2.3	Evaluating the models using: RMSE, MAE, MAPE. sMAPE	13
2.4	Evaluating the results	14
2.5	NN training with the NARX approach	16
2.5.1	Evaluating the results in the NARX approach	16
A	Code for Partitioning clustering	16
B	Code for Energy forecasting	18

1 Partitioning Clustering

1.1 Pre-Processing the data

For this task we were given a vehicle.xmls file containing **846** samples, with **19** different attributes including the '**Class**'. However, as the goal is to perform k-means clustering on the data, an unsupervised learning algorithm, it is required to remove the '**Class**' column as the model will classify the data on its own. I also removed the 'Sample' column as it will affect the next pre-processing tasks, scaling and outlier removal.

When it comes to the order, I chose to remove the outliers first as they seemed to negatively affect the clustering results if I scaled the data before removing them. To find the outliers I found the **z-score** for each of the samples and then removed any samples with a **z-score** than **3** and less than **-3**.

1.2 Finding the best k using: Nblast, Elbow method, Gap statistics and sillhoutte methods

1.2.1 Nblast

As shown below, Nblast says the best number of clusters is 3. Considering the original number of classes is 4 I believe that this is a good result.

```
1 * Among all indices:
2 * 6 proposed 2 as the best number of clusters
3 * 12 proposed 3 as the best number of clusters
4 * 1 proposed 6 as the best number of clusters
5 * 1 proposed 8 as the best number of clusters
6 * 1 proposed 11 as the best number of clusters
7 * 1 proposed 12 as the best number of clusters
8 * 2 proposed 15 as the best number of clusters
9
10      ***** Conclusion *****
11
12 * According to the majority rule, the best number of clusters is 3
```

1.2.2 Elbow Method

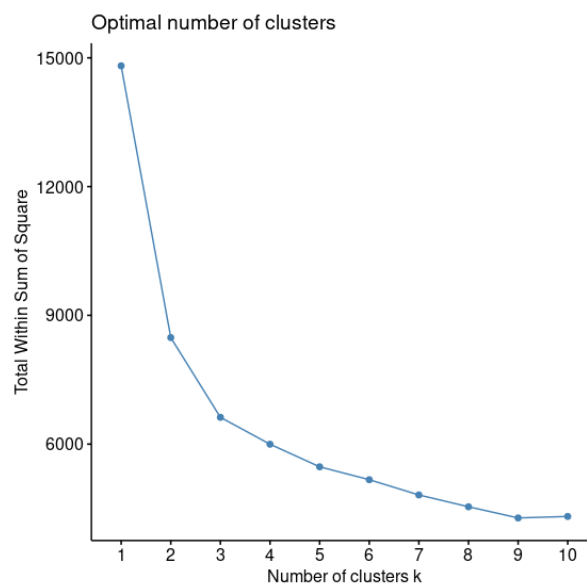


Figure 1: Elbow method plot

The Elbow method uses the **WCSS(within-cluster sums of squares)** which measures how close data points are in respect of their cluster centers. Based on the plot above, the recommended number of clusters is **3** as that is where the results begin to flatten out slowly indicating that increasing the clusters anymore will not result in any increase in performance.

1.2.3 Gap Statistics

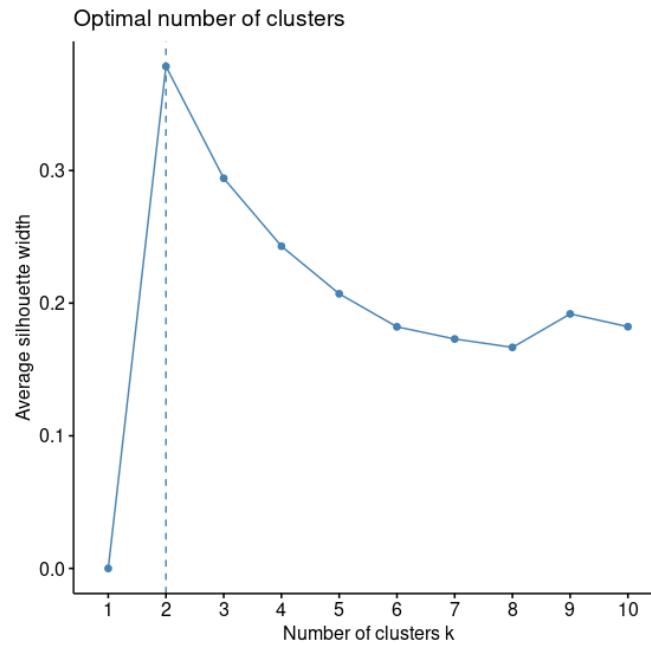


Figure 2: Gap statistics plot

The Gap statistics also uses the **WCSS** to calculate the best number of clusters to use. However, the recommended number of clusters in this case is **2**, knowing that the original data set has **4** possible classes, we can conclude that this result is worse than what we got with the elbow method which was **3**.

1.2.4 Silhouette Method

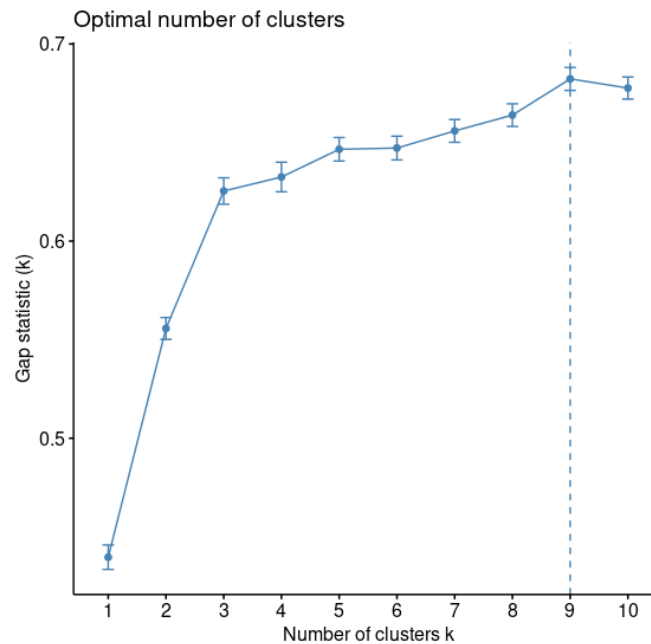


Figure 3: Silhouette method plot

The silhouette plot shows how similar a data point is to its own cluster using the **silhouette score**, this is a value that ranges from -1 and 1, with values closer to -1 meaning the data point should be in another cluster and the closer the value is to 1 meaning the current cluster is a good fit for the data point. This is where things get interesting, based on the plot above **9** is the recommended number of clusters. This is significantly higher than any of the other results from the other evaluation methods, I made to sure to run the model several times checking if there were errors with the code, but it gave **9** as the output everytime. This is by far the worst result as the original data set has **4** classes.

However as shown later in the report, after running the evaluation tools for the data that had **PCA** done on it. The results for the silhouette plot were a lot more controlled and matched the other evaluation methods as well. This led me to believe that having a data set that is too multi-dimensional led to an extreme result for the silhouette plot.

1.3 K-means Clustering investigation

1.3.1 Discussing the K-means outputs

Using the results from the evaluation methods, I decided to go for **k=3**, as both **Nbclust** and the **Elbow Method** gave a result of the best **k** being **3**. Below you can see the plot made from the clustering, without looking at the output data you can see a clear distinction between the clusters where there is no overlapping

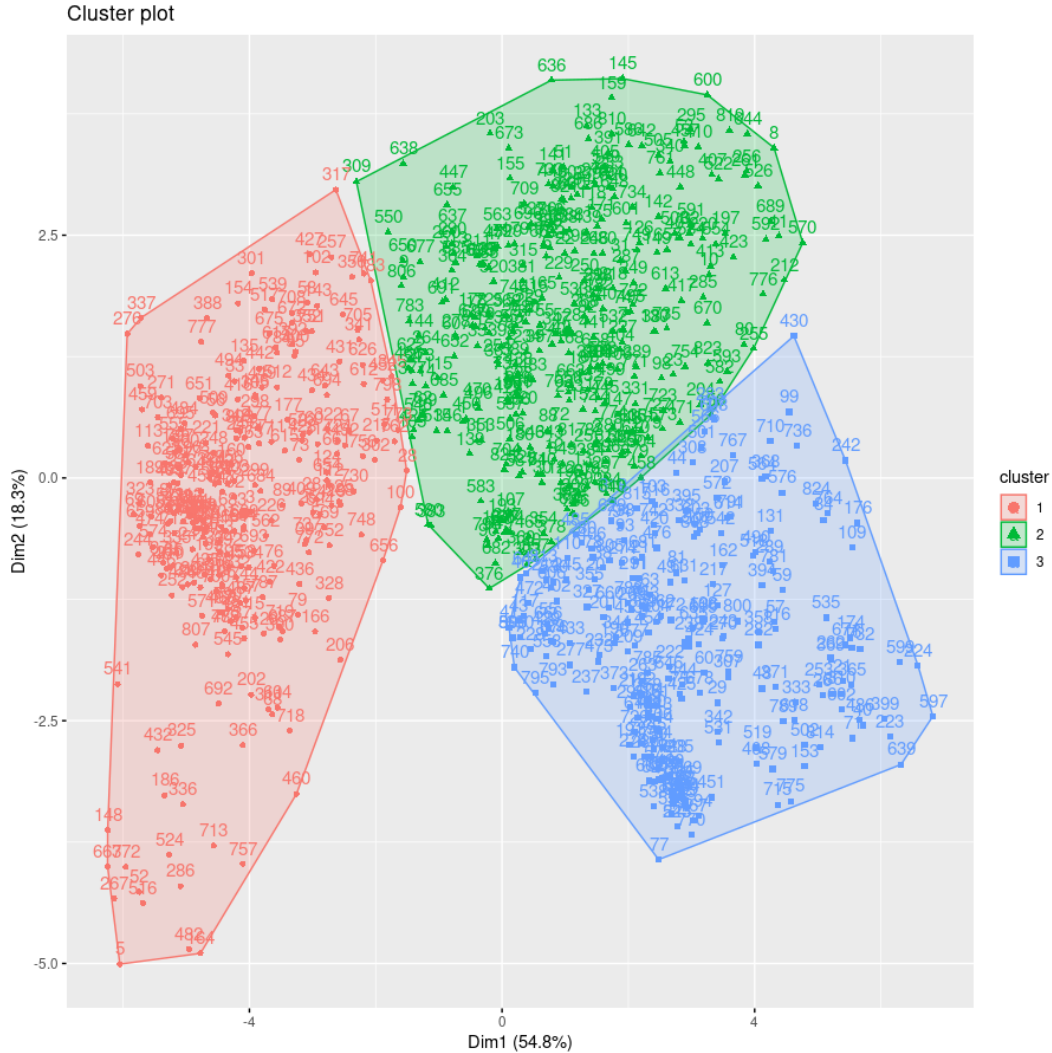


Figure 4: Clustering plot

Below are the kmeans output for the clustering attempt with **k=3**. You can see that the sizes of each cluster is evenly distributed which implies that the clustering did not favour or ignore any specific cluster. The **BSS(between sums of squares)** in this clustering is **8189.91** while the **WSS(within cluster sums of squares)** is **6624.09**. The ratio of the **BSS** and the **TSS(total sums of squares)** is **55.3%**, this number shows how well the clusters are separated from each other where a higher value means that clusters are well separated and a lower value means the clusters are not well defined.

To further investigate whether it was possible to get a lower **WSS** and a higher **BSS**, I run the clustering with **k=2** which was the second most recommended value of **k** by the automated tools, I concluded that **k=3** was indeed the best number of clusters as the **BSS** was higher while the **WSS** was lower in the clustering attempt with **k=2**.

```

1 K-means clustering with 3 clusters of sizes 256, 331, 237
2
3 > kmeans_data$centers
4      Comp      Circ      D.Circ      Rad.Ra Pr.Axis.Ra      Max.L.Ra      Scat.Ra
5 1  1.1672551  1.1913560  1.2226654  1.061855474  0.2398399  0.6675158  1.3141094
6 2 -0.2324797 -0.5226347 -0.2851558 -0.002041173  0.3625937 -0.1440161 -0.4446806
7 3 -0.9361458 -0.5569412 -0.9224294 -1.144132376 -0.7654747 -0.5198934 -0.7984081
8      Elong Pr.Axis.Rect Max.L.Rect Sc.Var.Maxis Sc.Var.maxis      Ra.Gyr      Skew.Maxis
9 1 -1.2220251  1.3199740  1.1102132  1.2689258  1.3291991  1.0980640 -0.08461041
10 2  0.3064563 -0.4736786 -0.4874626 -0.3936680 -0.4533218 -0.5482611 -0.66286263
11 3  0.8919890 -0.7642435 -0.5184154 -0.8208477 -0.8026391 -0.4203797  1.01716369
12      Skew.maxis      Kurt.maxis      Kurt.Maxis      Holl.Ra
13 1  0.16667482  0.27331007  0.01515673  0.2044549
14 2 -0.06083852 -0.01874875  0.75780956  0.6641968
15 3 -0.09506837 -0.26903603 -1.07474720 -1.1484793
16
17 > kmeans_data$cluster
18 [1] 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 3 2 1 1 3 3 2 2 1 2 3 1 1 3 2 2 2 1 2 2 3 1 3 1 3 3
19 [42] 2 3 3 3 3 2 3 2 1 2 1 2 2 3 1 3 1 3 3 3 2 3 3 1 2 1 1 1 2 3 2 1 2 3 1 3 3 1 2 3 2
20 [83] 2 3 2 3 1 2 1 2 3 1 3 3 1 3 2 2 3 1 1 1 3 3 2 2 2 3 3 3 2 1 1 3 2 3 3 2 2 2 3 2 2
21 [124] 1 1 2 3 1 3 2 3 2 2 3 1 3 2 1 2 2 2 2 1 2 2 1 2 2 3 2 2 3 1 2 2 1 1 2 1 3 3 1 1
22 [165] 2 1 2 2 2 2 2 2 3 1 3 2 3 1 2 2 2 1 2 1 2 2 1 2 3 1 3 3 3 2 2 1 1 2 2 2 3 3 1 2 2 2
23 [206] 1 3 2 3 1 3 2 1 3 1 3 3 2 1 2 1 3 3 3 3 1 2 3 2 3 1 3 2 2 3 1 3 3 2 2 1 3 3 1 3 2
24 [247] 2 1 2 2 1 1 3 2 2 2 1 3 3 2 2 3 3 2 2 2 1 2 3 3 1 2 2 3 3 1 3 2 2 3 1 3 3 2 2 1 2
25 [288] 1 3 2 2 1 2 2 2 3 2 1 1 1 1 2 2 1 3 3 3 2 3 1 1 3 1 2 3 1 3 2 2 2 1 1 3 1 1 3 1
26 [329] 2 2 2 3 3 1 1 1 1 2 2 2 1 3 2 3 1 2 2 1 2 1 1 1 2 2 3 1 2 3 3 2 2 2 2 2 3 1 1 3 3
27 [370] 1 3 1 3 1 2 2 2 2 1 3 2 2 2 2 2 2 2 1 2 1 2 1 2 3 3 2 2 2 3 3 2 3 1 2 2 3 2 3 1 2
28 [411] 3 2 2 1 2 1 2 1 1 3 3 1 2 3 3 2 1 1 3 3 1 1 3 1 1 1 2 2 2 2 2 1 3 3 2 1 2 2 1 2 3
29 [452] 1 3 3 1 1 2 2 1 1 1 3 1 1 2 2 3 1 1 2 2 3 3 1 2 3 1 1 2 3 1 1 2 1 3 3 1 1 1 3 3 1
30 [493] 1 1 2 2 1 3 2 1 3 3 1 3 2 2 3 2 1 2 1 1 2 3 2 1 1 3 3 2 1 2 1 1 2 2 2 2 2 3 3 2 2
31 [534] 1 3 3 2 3 1 2 1 3 3 1 1 2 1 2 2 2 1 2 3 2 1 2 2 3 1 1 1 1 2 3 3 3 1 1 1 2 1 3 2 1
32 [575] 3 3 3 2 3 2 2 2 2 2 2 2 1 2 2 1 2 2 2 3 1 3 3 2 3 2 2 3 3 1 1 3 2 3 1 2 2 1 2 3 1
33 [616] 3 1 3 3 2 3 2 1 1 2 1 2 2 3 2 3 1 2 1 3 2 2 3 2 3 2 1 2 1 3 2 2 2 2 1 2 3 1 2 1
34 [657] 2 2 1 3 1 3 2 2 2 3 1 2 3 2 3 1 2 2 1 3 2 3 2 2 3 2 1 1 2 2 1 1 2 3 2 1 1 1 1 2 1
35 [698] 2 2 1 1 2 1 2 1 2 3 1 2 3 1 1 1 2 3 3 1 1 1 2 1 2 2 1 2 3 2 3 2 1 2 3 2 2 2 3 1 3
36 [739] 3 3 1 3 1 1 3 2 2 1 2 3 1 1 3 2 2 1 1 1 3 1 2 1 1 3 3 1 3 1 2 3 2 1 1 2 3 2 1 1 2
37 [780] 2 3 2 2 1 3 2 1 3 3 1 3 2 3 3 3 2 1 1 2 3 1 2 1 1 3 2 1 3 3 2 2 1 3 3 3 2 2 2 2 2
38 [821] 2 1 2 3
39
40 > kmeans_data$tot.withinss
41 [1] 6624.09
42
43 > kmeans_data$betweenss
44 [1] 8189.91
45
46 Within cluster sum of squares by cluster:
47 [1] 2191.909 2735.763 1696.418
48 (between_SS / total_SS = 55.3 %)

```

1.3.2 Silhouette Plot

The silhouette plot shows how well the clustering is taking place and it will calculate the average distance between the clusters. In practice the plot displays how close each point in one cluster is to points in the neighbouring clusters. The **average width score** indicates how well the samples are well clustered, it ranges from **1** to **-1** where a score close to **1** means the samples are well matched to their own cluster while a score closer to **-1** means the samples are poorly matched to their own clusters, and a score close to **0** means the samples are more ambiguously placed and could be in another cluster.

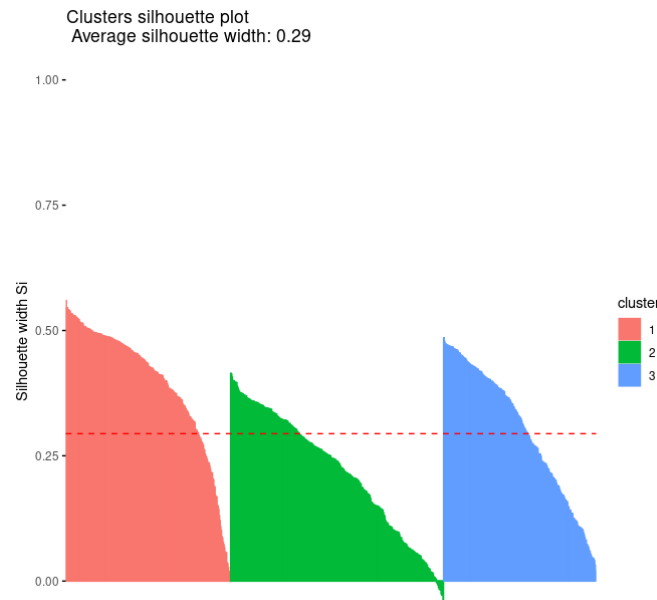


Figure 5: Silhouette plot

From the plot above you can see that the **average width score** is **0.29**, being a positive value we can say that the clusters are moderately accurate, but as the maximum score is **1** there can still be some improvements in the clustering to achieve a better **average width score**

1.4 K-means Clustering with PCA

1.4.1 Creating the new dataset with PCA

Below I have known the **eigenvalues**, the **eigenvectors**, and the **cumulative score** per principle component. To make the new transformed dataset we want to use the PCs with at least a **cumulative score** of **>92%** I decided to use the first **6 PCs** as they gave a total score of **0.94%**

```
1 > eigenvalues
2 [1] 9.8655415144 3.3026315672 1.2050866140 1.1255984677 0.8773731809 0.6636174794
3 [7] 0.3374343341 0.2274918343 0.1176165632 0.0871789864 0.0607683889 0.0450646831
4 [13] 0.0292070985 0.0213994038 0.0150961795 0.0123913361 0.0061400710 0.0003622977
5
6 > eigenvectors
7
8 Comp          PC1          PC2          PC3          PC4          PC5          PC6
9 Circ          -0.27099550  0.08819711 -0.03979285 -0.142474274 -0.15979926  0.219704493
10 D.Circ        -0.30078375  0.04064437  0.07450874 -0.104513476  0.07338676  0.000941066
11 Rad.Ra        -0.27595481  0.19284625  0.04085638  0.244080006 -0.12620414 -0.153234232
12 Pr.Axis.Ra    -0.10790106  0.24598582 -0.10092681  0.611908838 -0.05646656 -0.599471567
13 Max.L.Ra      -0.18693783  0.06836380 -0.10600156 -0.255241647  0.70801896 -0.255529947
14 Scat.Ra       -0.30925633 -0.07715243  0.10748098  0.001027495 -0.09117998  0.078463678
```


15	Elong	0.30718493	0.01853683	-0.09109269	-0.071391309	0.08547550	-0.061072204
16	Pr.Axis.Rect	-0.30618660	-0.09004872	0.10605368	-0.025003047	-0.08566679	0.087748728
17	Max.L.Rect	-0.27419519	-0.13582051	-0.20286313	-0.052151262	0.25259264	-0.012583332
18	Sc.Var.Maxis	-0.30244511	-0.07264590	0.13477043	0.057153180	-0.15616630	0.103440122
19	Sc.Var.maxis	-0.30676191	-0.08004640	0.10787776	0.004398469	-0.12508727	0.106371087
20	Ra.Gyr	-0.25860012	-0.21823056	-0.21386460	0.068595328	0.01184258	-0.063754044
21	Skew.Maxis	0.06158617	-0.50300209	0.06768991	0.125377302	-0.13879190	-0.159605991
22	Skew.maxis	-0.03877299	0.02950349	-0.55339412	-0.517610761	-0.48274633	-0.382718195
23	Kurt.maxis	-0.05921378	0.09616696	0.68221125	-0.400234808	-0.09248124	-0.471711647
24	Kurt.Maxis	-0.04751059	0.50763917	-0.07208105	0.027069843	-0.17449701	0.240919678
25	Holl.Ra	-0.09728514	0.50329529	-0.03870066	-0.089901222	0.12059247	0.082978199
26		PC7	PC8	PC9	PC10	PC11	
27	Comp	0.25075003	-0.762917498	0.336727260	-0.17080380	0.06059915	
28	Circ	-0.38184560	-0.084996844	0.048161956	0.14521912	-0.06103582	
29	D.Circ	0.10924250	0.307560350	0.369297550	0.09330027	0.74865950	
30	Rad.Ra	0.13812347	0.062362314	0.159039213	-0.02487175	-0.17932432	
31	Pr.Axis.Ra	0.06368508	-0.146618654	0.033075197	0.08677308	0.04900671	
32	Max.L.Ra	0.40902849	-0.032642651	-0.227739119	-0.25103768	-0.10840290	
33	Scat.Ra	0.09891112	0.092046874	-0.128654451	0.10439303	-0.14948976	
34	Elong	-0.10476915	-0.225039791	0.263923313	0.02991565	-0.09895280	
35	Pr.Axis.Rect	0.09681861	0.043426157	-0.071150433	0.18287483	-0.26837448	
36	Max.L.Rect	-0.36733465	-0.241378159	-0.121107876	0.50017751	0.09455120	
37	Sc.Var.Maxis	0.11234218	0.149165987	-0.129154922	-0.16972307	0.03485767	
38	Sc.Var.maxis	0.08604684	0.045421860	-0.102778876	0.11442449	-0.24325771	
39	Ra.Gyr	-0.45586499	0.112011651	0.148879282	-0.69204782	-0.05867687	
40	Skew.Maxis	0.11079493	-0.298664862	-0.505836049	-0.11269997	0.40780344	
41	Skew.maxis	0.12381756	0.128361642	-0.070226350	0.07170181	-0.02036668	
42	Kurt.maxis	-0.31638913	-0.134628700	0.005249849	-0.04532918	-0.03440818	
43	Kurt.Maxis	-0.18582252	-0.098767436	-0.460060564	-0.18269431	0.16137526	
44	Holl.Ra	-0.18385202	-0.002257517	-0.204524578	0.01863833	0.12215130	
45		PC12	PC13	PC14	PC15	PC16	
46	Comp	0.016236215	-0.15538799	-0.084941797	-0.009893937	0.014731452	
47	Circ	-0.108002512	-0.02379761	0.200359434	-0.411699600	0.633197650	
48	D.Circ	0.027236923	0.23107314	-0.032038645	-0.128176485	-0.032941366	
49	Rad.Ra	-0.148278795	0.02028449	0.782962301	-0.002680653	-0.262185162	
50	Pr.Axis.Ra	0.061511729	0.03176897	-0.360686574	0.022171363	0.091207086	
51	Max.L.Ra	-0.103284857	0.09369549	-0.004005999	-0.047699349	0.023924621	
52	Scat.Ra	0.114239242	-0.02130040	-0.070286104	-0.106776285	0.005784063	
53	Elong	0.155162263	0.75286275	0.157069120	0.228858568	0.132825704	
54	Pr.Axis.Rect	0.272369519	0.30540490	-0.201563990	-0.167283715	-0.290260077	
55	Max.L.Rect	-0.201414962	-0.03380300	-0.013996509	0.370105120	-0.376075706	
56	Sc.Var.Maxis	-0.228758252	0.06412845	-0.026516516	0.694529334	0.411479615	
57	Sc.Var.maxis	0.177853616	0.28399252	-0.085543400	-0.046373620	0.145074045	
58	Ra.Gyr	0.153739469	0.03885792	-0.107040664	0.039303314	-0.249704673	
59	Skew.Maxis	0.220683275	0.09807688	0.277643886	-0.073168084	-0.021263745	
60	Skew.maxis	0.001322677	-0.01515260	0.002919287	0.032651291	0.017641831	
61	Kurt.maxis	-0.087563183	-0.01891627	-0.022107399	-0.022609384	0.002337621	
62	Kurt.Maxis	-0.385070902	0.34206081	-0.073483280	-0.224113793	-0.089715206	
63	Holl.Ra	0.697685477	-0.19114605	0.188619261	0.198088827	0.123427696	
64		PC17	PC18				
65	Comp	0.0022871138	-0.0001888306				
66	Circ	0.1935606420	0.0189798203				
67	D.Circ	-0.0338082604	-0.0095717960				
68	Rad.Ra	0.0060687302	-0.0275176970				
69	Pr.Axis.Ra	-0.0091405437	0.0177603416				
70	Max.L.Ra	-0.0048404910	-0.0083581152				
71	Scat.Ra	-0.3857289906	0.7909901632				
72	Elong	-0.0570308312	0.2237899607				
73	Pr.Axis.Rect	0.6548967453	-0.0151607986				
74	Max.L.Rect	-0.1025905067	-0.0266527364				
75	Sc.Var.Maxis	0.2336432149	0.0416825198				
76	Sc.Var.maxis	-0.5542304543	-0.5644496563				
77	Ra.Gyr	-0.0737884732	0.0031786081				
78	Skew.Maxis	0.0234042984	-0.0068080750				
79	Skew.maxis	0.0046042860	-0.0030669415				
80	Kurt.maxis	-0.0007617607	-0.0075237217				
81	Kurt.Maxis	-0.0118304320	0.0341259519				
82	Holl.Ra	0.0432842111	-0.0094922864				
83							
84							
85	> summary(pca_data)						

```

86 Importance of components:
87
88      PC1      PC2      PC3      PC4      PC5      PC6      PC7      PC8
89 Standard deviation  3.1409  1.8173  1.09776  1.06094  0.93668  0.81463  0.58089  0.47696
90 Proportion of Variance  0.5481  0.1835  0.06695  0.06253  0.04874  0.03687  0.01875  0.01264
91 Cumulative Proportion  0.5481  0.7316  0.79851  0.86105  0.90979  0.94666  0.96540  0.97804
92      PC9      PC10      PC11      PC12      PC13      PC14      PC15      PC16
93 Standard deviation  0.34295  0.29526  0.24651  0.2123  0.17090  0.14629  0.12287  0.11132
94 Proportion of Variance  0.00653  0.00484  0.00338  0.0025  0.00162  0.00119  0.00084  0.00069
95 Cumulative Proportion  0.98458  0.98942  0.99280  0.9953  0.99692  0.99811  0.99895  0.99964
96      PC17      PC18
97 Standard deviation  0.07836  0.01903
98 Proportion of Variance  0.00034  0.00002
99 Cumulative Proportion  0.99998  1.00000

```

1.5 Finding the best k for PCA dataset using: Nblust, Elbow method, Gap statistics and sillhoutte methods

1.5.1 Nblust

The results for **Nbclust** while using the newly made dataset using **PCA** were not different from the original k-means clustering attempt using the original dataset. Nbclust still says the best number of clusters is **3**.

```

1 * Among all indices:
2 * 7 proposed 2 as the best number of clusters
3 * 10 proposed 3 as the best number of clusters
4 * 1 proposed 4 as the best number of clusters
5 * 2 proposed 6 as the best number of clusters
6 * 2 proposed 9 as the best number of clusters
7 * 2 proposed 15 as the best number of clusters
8
9      ***** Conclusion *****
10
11 * According to the majority rule, the best number of clusters is 3

```

1.5.2 Elbow Method

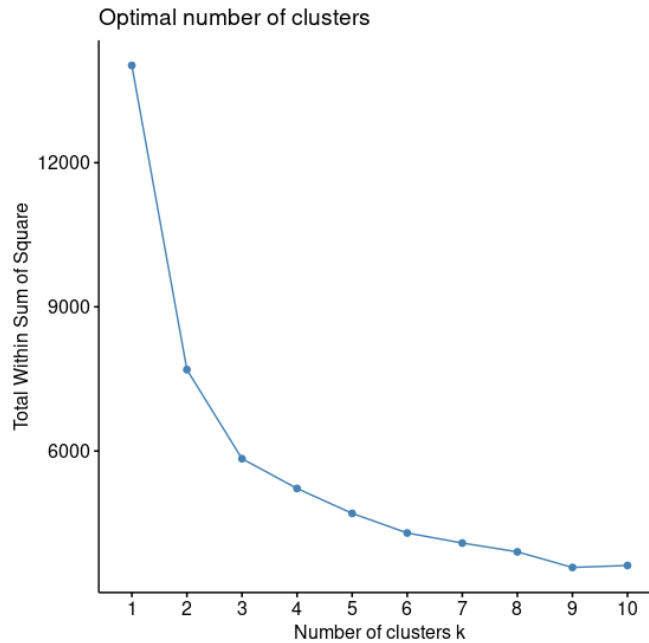


Figure 6: Elbow method plot

The **elbow method** is not showing new results and also says the recommended number of clusters is still **3**.

1.5.3 Gap Statistics

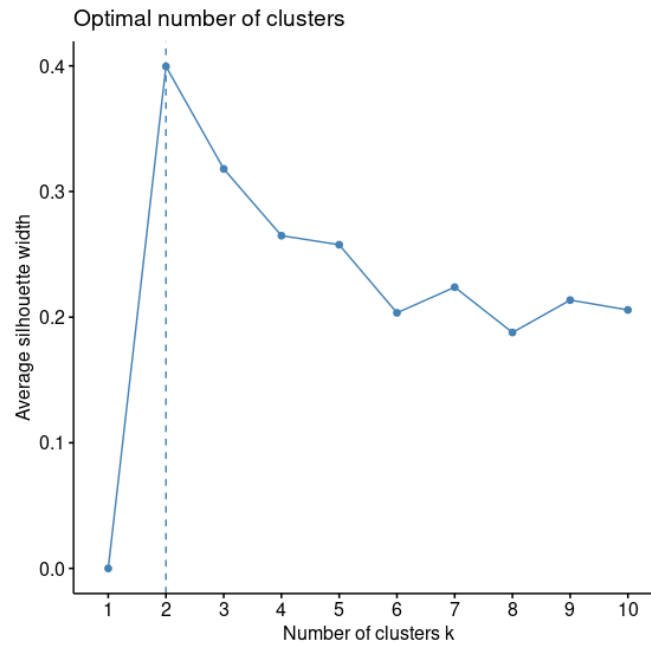


Figure 7: Gap statistics plot

The **gap statistics** also still says the recommended number of clusters is still **2**.

1.5.4 Silhouette Method

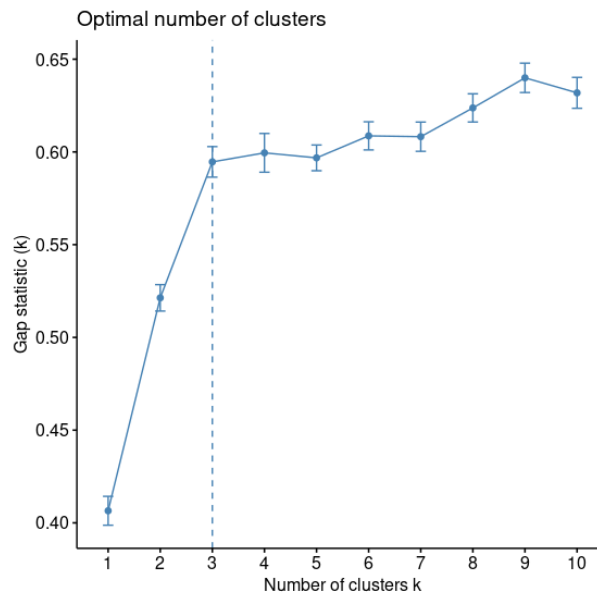


Figure 8: Silhouette method plot

As stated previously, the **sillhoutte plot** for the data that had **PCA** done to it gives a more reasonable result for the reccomended number of clusters, this being **3**.

1.6 K-means Clustering Investigation with PCA

1.6.1 Discussing the K-means outputs

The most reccomended number of clusters is still **3**, however this time as the data has been passed through **PCA**, the clustering is significantly different compared to the attempt done with the original data. As shown in the plot below, this time there is a lot more of overlapping especially with cluster 1 and 2. This is not ideal as we want each cluster to have a clear distance from the other ones.

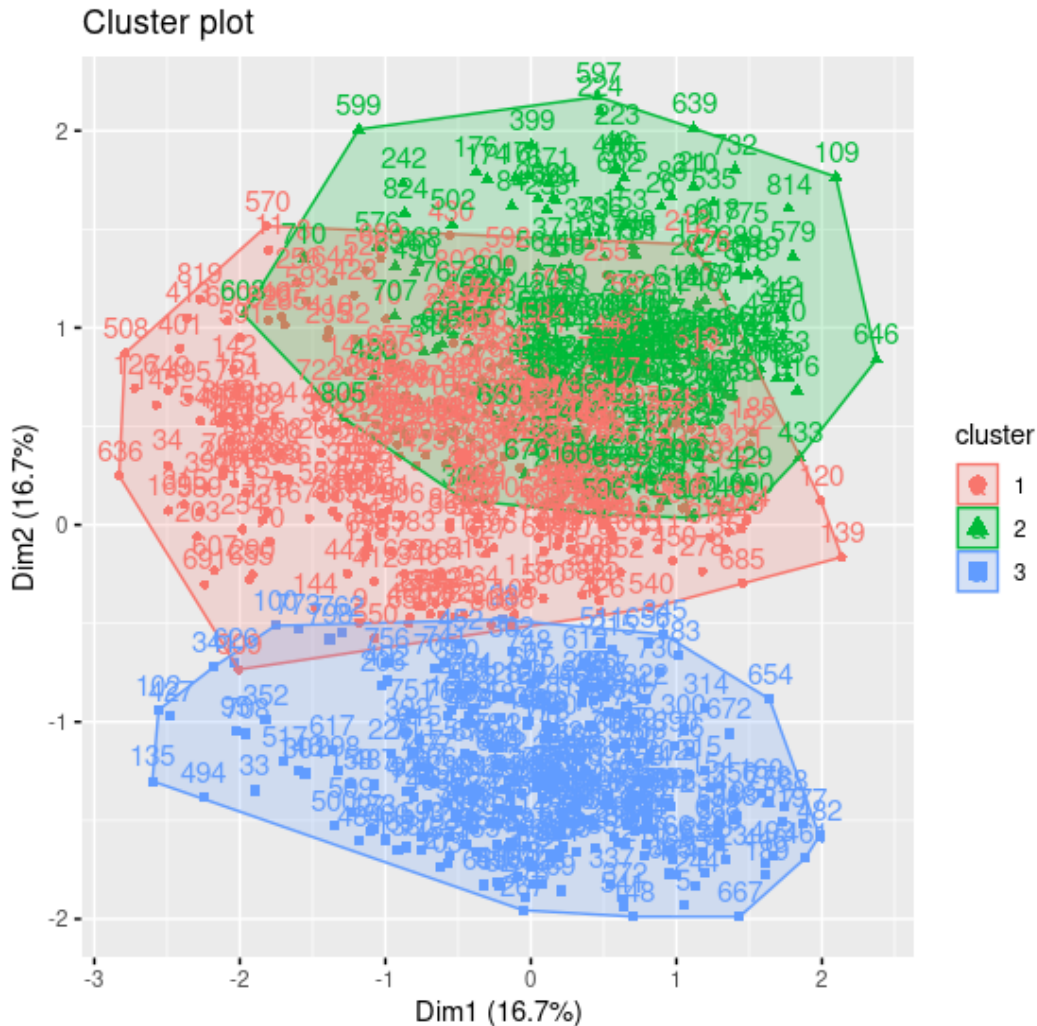


Figure 9: Clustering plot

I have put below the kmeans output for the clustering attempt using the data passed through **PCA**. I am still using **k=3** as Nbclust, elbow method and the sillhoutte plot all had an output of 3 as the best number of clusters. The **BSS** in this clustering is **8183.617** while the **WSS** is **5840.179**. The ratio of the **BSS** and the **TSS** is **58.4%**. By running **PCA** on the data, we were able to improve our results especially for the **WSS** as this time it is a lot lower than the original attempt.

```
1 K-means clustering with 3 clusters of sizes 332, 236, 256
2
3 > kmeans_pca_data$centers
```

```

4      PC1      PC2      PC3      PC4      PC5      PC6
5 1 -1.078026 -1.5110695 0.04723636 -0.1754898 0.009380699 -0.01659038
6 2 -2.911460 1.7332691 0.02469198 0.1345272 -0.123755941 0.11129772
7 3 4.082068 0.3618108 -0.08402257 0.1035711 0.101921914 -0.08108694
8
9 > kmeans_pca_data$cluster
10 [1] 1 1 3 1 3 1 1 1 1 1 1 1 1 1 3 2 1 3 3 2 2 1 1 3 1 2 3 3 2 1 1 1 3 1 1 2 3 2 3 2
11 2
12 [42] 1 2 2 2 2 1 2 1 3 1 3 1 1 2 3 2 3 2 2 2 1 2 2 3 1 3 3 3 1 2 1 3 1 2 3 2 2 3 1 2
13 1
14 [83] 1 2 1 2 3 1 3 1 2 3 2 2 3 2 1 1 2 3 3 3 2 2 1 1 1 2 2 2 1 3 3 2 1 2 2 1 1 1 2 1
15 1
16 [124] 3 3 1 2 3 2 1 2 1 1 2 3 2 1 3 1 1 1 1 3 1 1 3 1 3 1 2 1 1 2 3 1 1 3 3 1 3 2 2 3
17 3
18 [165] 1 3 1 1 1 1 1 2 3 2 1 2 3 1 1 1 3 1 3 1 1 3 1 2 3 2 2 2 1 1 3 3 1 1 1 2 2 3 1 1
19 1
20 [206] 3 2 1 2 3 2 1 3 2 3 2 2 1 3 1 3 2 2 2 2 3 1 2 1 2 3 2 1 1 2 3 2 2 1 1 3 2 2 3 2
21 1
22 [247] 1 3 1 1 3 3 2 1 1 1 3 2 2 1 1 2 2 1 1 1 3 1 2 2 3 1 1 2 2 3 2 1 1 2 3 2 1 1 1 3
23 1
24 [288] 3 2 1 1 3 1 1 1 2 1 3 3 3 3 3 2 1 3 2 2 2 1 2 3 3 2 3 1 2 3 2 1 1 1 3 3 2 3 3 2
25 3
26 [329] 1 1 1 2 2 3 3 3 3 1 1 1 3 2 1 2 3 1 1 3 1 3 3 3 1 1 2 3 1 2 2 1 1 1 1 1 2 3 3 2
27 2
28 [370] 3 2 3 2 3 1 1 1 1 3 2 1 1 1 1 1 1 1 3 1 3 1 3 1 2 2 1 1 1 2 2 1 2 3 1 1 2 1 2 3
29 1
30 [411] 2 1 1 3 1 3 1 3 3 2 2 3 1 2 2 1 3 3 2 1 3 3 2 3 3 3 1 1 1 1 1 3 2 2 1 3 1 1 3 1
31 2
32 [452] 3 2 2 3 3 1 1 3 3 3 2 3 3 1 1 2 3 3 1 1 2 2 3 1 2 3 3 1 2 3 3 1 3 2 2 3 3 3 2 2
33 3
34 [493] 3 3 1 1 3 2 1 3 2 2 3 2 1 1 2 1 3 1 3 3 1 2 1 3 3 2 2 1 3 1 3 3 1 1 1 1 1 2 2 1
35 1
36 [534] 3 2 2 1 2 3 1 3 2 2 3 3 1 3 1 1 1 3 1 2 1 3 1 1 2 3 3 3 3 1 2 2 2 3 3 3 1 3 2 1
37 3
38 [575] 2 2 2 1 2 1 1 1 1 1 1 1 1 3 1 1 3 1 1 1 2 3 2 2 1 2 1 1 2 2 3 3 2 1 2 3 1 1 3 1 2
39 3
40 [616] 2 3 2 2 1 2 1 3 3 1 3 1 1 2 1 2 3 1 3 2 1 1 1 2 1 2 1 3 1 3 2 1 1 1 1 3 1 2 3 1
41 3
42 [657] 1 1 3 2 3 2 1 1 1 2 3 1 2 1 2 3 1 1 3 2 1 2 1 1 2 1 3 3 1 1 3 3 1 2 1 3 3 3 3 1
43 3
44 [698] 1 1 3 3 1 3 1 3 1 2 3 1 2 3 3 3 1 2 2 3 3 3 1 3 1 1 3 1 2 1 2 1 3 1 2 1 1 1 2 3
45 2
46 [739] 2 2 3 2 3 3 2 1 1 3 1 2 3 3 2 1 1 3 3 3 2 3 1 3 3 2 2 3 2 3 1 2 1 3 3 1 2 1 3 3
47 1
48 [780] 1 2 1 1 3 2 1 3 2 2 3 2 1 2 2 2 1 3 3 1 2 3 1 3 3 2 1 3 2 2 1 1 3 2 2 2 1 1 1 1
49 1
50 [821] 1 3 1 2
51
52 > kmeans_pca_data$tot.withinss
53 [1] 5840.179
54
55 > kmeans_pca_data$betweenss
56 [1] 8183.617
57
58 Within cluster sum of squares by cluster:
59 [1] 2415.343 1461.091 1963.745
60 (between_SS / total_SS = 58.4 %)

```

1.6.2 Silhouette Plot

From the plot below you can see that the **average width score** this time is **0.32**, this is an increase of **0.03** as in the original attempt the score was **0.29**. Again, this is not the best result as the maximum is score **1**, but

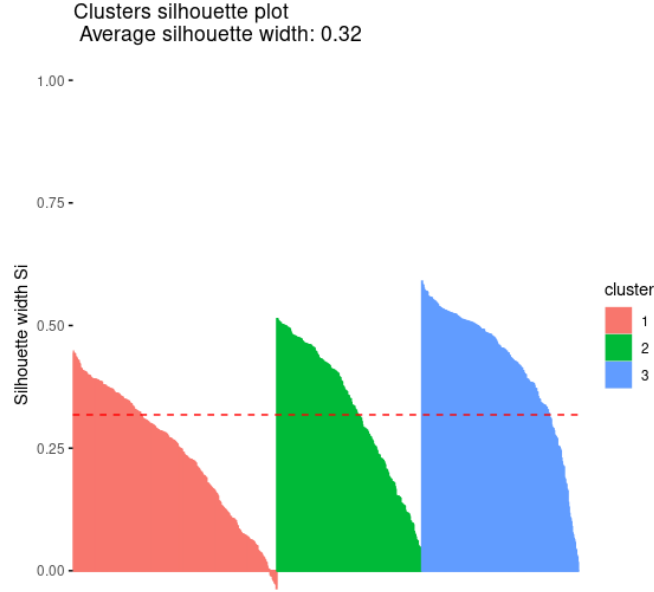


Figure 10: Silhouette plot

1.6.3 Calinski Harabasz index

This is a measure of the optimal number of clusters, the value can range between 0 and infinity where higher values means a better clustering. For the k-means model with a k of 3 the score received is 575.2178.

2 Energy Forecasting

2.1 Type of input variables used for electricity load forecasting

2.2 Normalising the data

Before beginning the training of the neural networks I normalised the I/O matrices. This is an essential step because it helps to improve the performance and training speed of the model. It also improves the stability of the model as outliers will not have as heavy of an influence in the models output, it also increases the rate of convergence as the model will be working with values in a smaller range allowing it to find the optimal weights faster.

2.3 Evaluating the models using: RMSE, MAE, MAPE. sMAPE

RMSE(Root Mean Square Error) is calculated using the following formula:

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

The formula uses the square root of the mean of the squared differences between the predicted and the actual values to tell us how far the predictions are on average from the actual values where a smaller **RMSE** means the model is predicting well and a bigger **RMSE** means the model is not predicting the data well. The downside here is that it is more sensitive to outliers as they can influence sum of squared errors. Another issue is that it can be affected by differences in scale between variables, if one has values in a much larger range than another it will be emphasised more.

MAE(Mean Absolute Error) is calculated using the following formula:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

This uses the average of the absolute difference between each predicted and actual value to tell us how far the predictions are on average from the actual values where a smaller **MAE** means the model is predicting well and a bigger **MAE** means the model is not predicting the data well. The downside here is that as it takes in the absolute error it will treat all errors equally so whether they are positive or negative it will not matter. Unlike **RMSE**, **MAE** is not as sensitive to outliers so it will not reflect the effect outliers have in the model prediction, which makes it a less accurate evaluation tool if the data still has outliers.

MAPE(Mean Absolute Percentage Error) is calculated using the following formula:

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100\%$$

This uses the absolute difference between predicted and actual value, it then divides the difference by the actual value and the result is expressed as a percentage in the end. This tells us how far the predictions are on average from the actual values where a smaller **MAPE** means the model is predicting well and a bigger **MAPE** means the model is not predicting the data well. This is useful when the scale of the data varies a lot, it is also commonly used as it provides the result in percentages which is easier to understand. The problem with this metric is that it can put more emphasis on large errors and less emphasis on smaller errors, it is also does not work as well when the actual value is equal to or close to **0**. In such cases it can be expected to receive a very large **MAPE** result or even for it to be undefined.

sMAPE(Symmetric Mean Absolute Percentage Error) is calculated using the following formula:

$$SMAPE = \frac{1}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2} \times 100\%$$

This measures the average of the absolute percentage difference between predicted and actual value, it then divides the difference by the sum of the absolute predicted and actual values. In situations where the actual values are equal to or close to **0**, **sMAPE** is preferred as it handles them better compared to **MAPE**.

2.4 Evaluating the results

When it came to the actual training of the neural network models I focused on a few parameters to change the output of the models, these being: the input I/O matrices, the layers and number of nodes per layer and whether the output is linear or not. I did not change the activation function as the difference in output was negligible in my attempts at training the models, so I chose to not include it as a parameter and just used the default one that comes with the **neuralnet()** function, the logistic activation function. I made **5** different input delayed I/O matrices and made sure to train at least **2** models with each matrix, the first with **1** hidden layer while the second with **2** hidden layers.

Table 1: Comparison table

Model	RMSE	MAE	MAPE	sMAPE	Description
NN ₁	0.1152572	0.0921373	0.1859476	0.1818735	I/O matrix: t-2, t-1, t, t+1 layers: 1x8 linear output: TRUE
NN ₂	0.11671855	0.09510258	0.18712046	0.18152017	I/O matrix: t-7, t-1, t+1 layers: 2x6-4 linear output: FALSE
NN ₃	0.11274105	0.08804056	0.16946062	0.17188169	I/O matrix: t-7, t-2, t-1, t+1 layers: 1x5 linear output: TRUE
NN ₄	0.1368248	0.1034995	0.2027808	0.2093880	I/O matrix: t-7, t-4, t-3, t-2, t-1, t, t+1 layers: 2x12-6 linear output: FALSE
NN ₅	0.11566742	0.09362933	0.19071920	0.18508842	I/O matrix: t-1, t, t+1 layers: 1x4 linear output: TRUE
NN ₆	0.11586446	0.08959127	0.17309226	0.17616521	I/O matrix: t-7, t-2, t-1, t+1 layers: 1x6 linear output: TRUE
NN ₇	0.11679370	0.09541529	0.18768615	0.18188354	I/O matrix: t-7, t-1, t+1 layers: 1x4 linear output: FALSE
NN ₈	0.1111209	0.0878771	0.1782428	0.1735545	I/O matrix: t-2, t-1, t, t+1 layers: 2x7-3 linear output: FALSE
NN ₉	0.11686106	0.09053119	0.17770444	0.17620528	I/O matrix: t-7, t-4, t-3, t-2, t-1, t, t+1 layers: 1x10 linear output: TRUE
NN ₁₀	0.11076446	0.09087368	0.18499380	0.17967582	I/O matrix: t-1, t, t+1 layers: 2x4-2 linear output: FALSE
NN ₁₁	0.10991593	0.08468949	0.16530730	0.16681759	I/O matrix: t-7, t-2, t-1, t+1 layers: 2x5-3 linear output: FALSE
NN ₁₂	0.1324693	0.1046402	0.1981125	0.2063737	I/O matrix: t-7, t-4, t-3, t-2, t-1, t, t+1 layers: 2x6-2 linear output: TRUE

From the table above you can see that the results for the different models are not too different from each other. After looking at the results I found that the best **1-layer NN** is NN₃ while the best **2-layer NN** is NN₁₁. I am basing this decision on the scores achieved for the evaluation metrics, I tried to find the total number of weights but it did not calculate properly as it showed 2 and 3 for all models. Another observation made from the table above is that both NN₃ and NN₁₁ have the period **t-7** in their input matrix. As suggested by the specification it appears that the **t-7** period does indeed hold more influence on the prediction made by the model.

2.5 NN training with the NARX approach

With the **NARX** approach we also have to consider the input vectors from the **18th** and **19th** hour columns. My expectations were that with a more varied amount of data it would lead to better results when it comes to NN training.

I made 4 different I/O matrices which ranged from **t** to **t-4** and included **t-7** for some of the I/O matrices.

2.5.1 Evaluating the results in the NARX approach

Below you can see the results from this training attempt, as you can see this models performed better than the ones trained with the AR approach.

Table 2: Comparison table for NARX approach

Model	RMSE	MAE	MAPE	sMAPE	Description
NN ₁	0.09954752	0.08211452	0.16860201	0.15971510	I/O matrix: 18(t-2), 19(t-1), 19(t), 20(t+1) layers: 1x8 linear output: TRUE
NN ₂	0.10163645	0.08423843	0.15945130	0.16691218	I/O matrix: 20(t-7), 19(t-7), 18(t-4), 20(t-2), 19(t), 20(t+1) layers: 2x10-4 linear output: FALSE
NN ₃	0.10463934	0.08856077	0.18183779	0.17094784	I/O matrix: 18(t-7), 20(t-4), 19(t), 20(t), 20(t+1) layers: 2x8-3 linear output: TRUE
NN ₄	0.09084888	0.07564465	0.15531772	0.14718595	I/O matrix: 18(t-4), 19(t), 20(t+1) layers: 1x6 linear output: FALSE
NN ₅	0.11171708	0.09121835	0.17266331	0.18359165	I/O matrix: 20(t-7), 19(t-7), 18(t-4), 20(t-2), 19(t), 20(t+1) layers: 1x12 linear output: TRUE

It seems that by using the other hours to train the model, it was able to make a better prediction across all the evaluation metrics. What seems to affect the accuracy of the model is the I/O matrix used as the input, in the **AR** approach training the **t-7** period led to the models performing better than others, but in this case models with I/O matrices with that period did not perform the best. Instead it was **NN₁** which had inputs from the periods **18(t-4)**, **19(t)**, **20(t+1)**.

A Code for Partitioning clustering

Listing 1: My R script

```
1 library("readxl")
2 library("ggplot2")
3 library("dplyr")
4 # Reading in the vehicle.xlsx file
5 data <- read_excel("~/Documents/rcw/data/vehicles.xlsx")
6 #removing the class and sample column as we do not need it for the k means clustering.
7 data <- subset(data, select = -c(Samples, Class))
8 str(data)
9 #making sure there are no null values in the data set before checking for outliers.
```

```

10 sum(is.na(data))
11
12 #using the boxplot to visually show the outliers
13 boxplot(data)
14 #use the sapply function to find the z-score for all the samples to then determine the outliers.
15 #I am creating a new column for the z-score for each of the samples,
16 #this makes it easier to remove the outliers.
17 data$zscore <- sapply(data, function(data) (data-mean(data))/sd(data))
18 #just double checking the dimensions of the original data set before removing outliers.
19 dim(data)
20 head(data$zscore)
21
22 #creating a new table that does not contain the outliers,
23 #anything that has a z-score lower than 3 and more than -3.
24 no_outliers <- data[!rowSums(abs(data$zscore) > 3), ]
25
26 #checking the maximum and minimum z-score to make sure the previous step worked as planned.
27 max(no_outliers$zscore)
28 min(no_outliers$zscore)
29 #new dimension of the new table, it is now 824 x 19 instead of 846 x 19
30 dim(no_outliers)
31
32 #this is just visually compare the original data set with the newly made one
33 boxplot(data)
34 boxplot(no_outliers)
35
36 #now removing the z-score column as it will change the result of the clustering
37 no_outliers <- subset(no_outliers, select = -c(zscore))
38 #scaling the data using the built-in scale function in R,
39 #I am also making a new dataframe to make comparisons easier.
40 no_outliers_normalised <- as.data.frame(scale(no_outliers))
41 #making sure the data was normalised properly
42 no_outliers$Rad.Ra
43 no_outliers_normalised$Rad.Ra
44
45 #using automated tools to determine the best number of cluster centers
46 library("NbClust")
47 library("factoextra")
48 library("cluster")
49 library("fpc")
50
51 #Nbclust
52 NbClust(no_outliers_normalised, distance = "euclidean", method = "kmeans", index="all")
53 #Elbow method
54 fviz_nbclust(no_outliers_normalised, kmeans, method='wss')
55 #Silhouette method
56 fviz_nbclust(no_outliers_normalised, kmeans, method='silhouette')
57 #Gap Stastics
58 fviz_nbclust(no_outliers_normalised, kmeans, method='gap_stat')
59
60 #performing kmeans clustering using the most favoured "k" from the automated methods used above.
61 kmeans_data <- kmeans(no_outliers_normalised, centers = 3, nstart = 10)
62 fviz_cluster(kmeans_data, data = no_outliers_normalised)
63
64 data_cluster <- data.frame(no_outliers_normalised, cluster = as.factor(kmeans_data$cluster))
65 head(data_cluster)
66
67 #showing the kmeans output
68 kmeans_data
69 #showing the within cluser summs of squares(WSS) and the between cluster sums of squares (BSS)
70 kmeans_data$centers
71 kmeans_data$tot.withinss
72 kmeans_data$betweenss
73 #showing the silhouette plot for the clustering
74 kmeans_data$cluster
75 silhouette <- silhouette(kmeans_data$cluster, dist(no_outliers_normalised))
76 fviz_silhouette(silhouette)
77
78 #applying PCA to the dataset to reduce dimensionality.
79 #this is the scaled dataset without the outliers
80 head(no_outliers_normalised)

```

```

81 pca_data <- prcomp(no_outliers_normalised, center = TRUE, scale = TRUE)
82 eigenvalues <- pca_data$sdev^2
83 eigenvectors <- pca_data$rotation
84 eigenvalues
85 eigenvectors
86 summary(pca_data)
87
88 #making a new dataset with only the first 6 PCA as the cumulative proportion for them is > 92%
89 transformed_data <- as.data.frame(-pca_data$x[,1:6])
90 head(transformed_data)
91 #new dimension is 824 x 6, thus reducing the attributes by 3 times.
92 dim(pca_data$x)
93 dim(transformed_data)
94
95 #applying the same 4 automated tools to find the new favoured k for the clustering
96 #Nbclust
97 NbClust(transformed_data, distance = "euclidean", method = "kmeans", index="all")
98 #Elbow method
99 fviz_nbclust(transformed_data, kmeans, method='wss')
100 #Silhouette method
101 fviz_nbclust(transformed_data, kmeans, method='silhouette')
102 #Gap Stastics
103 fviz_nbclust(transformed_data, kmeans, method='gap_stat')
104
105 #performing kmeans clustering using the new dataset
106 #performing kmeans clustering using the most favoured "k" from the automated methods used above.
107 kmeans_pca_data <- kmeans(transformed_data, centers = 3, nstart = 10)
108 fviz_cluster(kmeans_pca_data, data = transformed_data)
109
110 data_cluster <- data.frame(transformed_data, cluster = as.factor(kmeans_pca_data$cluster))
111 head(data_cluster)
112
113 #showing the kmeans output
114 kmeans_pca_data
115 #showing the within cluser summs of squares(WSS) and the between cluster sums of squares (BSS)
116 kmeans_pca_data$centers
117 kmeans_pca_data$tot.withinss
118 kmeans_pca_data$betweenss
119 #showing the silhouette plot for the clustering
120 kmeans_pca_data$cluster
121 silhouette <- silhouette(kmeans_pca_data $cluster, dist(transformed_data))
122 fviz_silhouette(silhouette)
123
124 #using the calinski-harabasz index
125 calinhara(transformed_data, kmeans_pca_data$cluster)

```

B Code for Energy forecasting

Listing 2: My R script

```

1 library("readxl")
2 library("dplyr")
3 library("neuralnet")
4 library("Metrics")
5
6 data <- read_excel("~/Documents/rcw/data/uow_consumption.xlsx")
7
8 #renaming the columns as they have weird names by default
9 names(data)[2] <- '18:00'
10 names(data)[3] <- '19:00'
11 names(data)[4] <- '20:00'
12 head(data)
13
14 #getting the 20:00 column only as it is our focus
15 input_data <- subset(data, select = c('20:00'))
16 input_data
17
18 #normalising the data
19 normalize <- function(x) {
20 return((x - min(x)) / (max(x) - min(x)))

```

```

21 }
22
23 #normalizing data before creating the delayed input I/O matrices
24 normalized_input_data <- as.data.frame(lapply(input_data, normalize))
25 normalized_input_data
26 head(normalized_input_data)
27 #function to create the time delayed input variables I/O matrices
28 create_time_lagged_data <- function(data, lag_values, name) {
29   #this lets me specify the time lag and it will bind the time delayed column
30   #to the original dataset original
31   #which acts as the column for the predicted column, as it has no lag
32   time_lagged_data <- cbind(lapply(lag_values, function(x) lag(data, x)), data)
33   # Remove rows with missing values
34   time_lagged_data <- time_lagged_data[complete.cases(time_lagged_data),]
35   #Rename the data with the relevant names
36   names(time_lagged_data) <- name
37   return(time_lagged_data)
38 }
39
40 # Create different versions of the time-lagged data
41 time_lagged_data_1 <- create_time_lagged_data(normalized_input_data,
42   c(3,2,1),
43   c("previous2", "previous1", "current", "predicted"))
44 time_lagged_data_2 <- create_time_lagged_data(normalized_input_data,
45   c(8, 2),
46   c("previous7", "previous1", "predicted"))
47 time_lagged_data_3 <- create_time_lagged_data(normalized_input_data,
48   c(8, 5:1),
49   c("previous7", "previous4", "previous3", "previous2",
50     "previous1", "current", "predicted"))
51 time_lagged_data_4 <- create_time_lagged_data(normalized_input_data,
52   c(8, 3, 1),
53   c("previous7", "previous2", "current", "predicted"))
54 time_lagged_data_5 <- create_time_lagged_data(normalized_input_data,
55   c(2, 1),
56   c("previous1", "current", "predicted"))
57
58 # Define a function to make training the nn models easier
59 train_neuralnet <- function(data, layers, linear_output) {
60
61   #creating the training and testing batches for the nn, keeping the training sample to 380.
62   train_data <- data[1:380,]
63   test_data <- data[381:nrow(data),]
64
65   #making the nn using the neuralnet function.
66   nn <- neuralnet(predicted ~ .,
67     data = train_data,
68     hidden = layers,
69     linear.output = linear_output)
70
71   # calculating the total number of weights in the neural network
72   #I was not able to get the weights to show properly as it was giving an
73   #output of 2 and 3 for all models
74   total_weights <- sum(sapply(nn$weights, length))
75
76   predicted <- predict(nn, newdata = test_data)
77   actual <- test_data$predicted
78   #using the RMSE, MAE, MAPE and SMAPE to evaluate the models,
79   #It is also saved into a list which makes it easy to view all results
80   metrics <- c("RMSE" = rmse(actual, predicted),
81     "MAE" = mae(actual, predicted),
82     "MAPE" = mape(actual, predicted),
83     "SMAPE" = smape(actual, predicted),
84     "Total Weights" = total_weights)
85   #plotting for visual representation
86   # plot(nn)
87   return(metrics)
88 }
89
90 #I am making a list that holds all the parameters I will be using to train each of the 15 models
91 #So for example any values at index 1 of the different keys of the nn_parameters list

```

```

92 #will be used to train nn_1, index 2 for nn_2 and so on.
93 nn_parameters <- list()
94 #specifying the data to be used to train the model
95 nn_parameters$data <- list(time_lagged_data_1,#nn1
96                             time_lagged_data_2, #nn2
97                             time_lagged_data_4, #nn3
98                             time_lagged_data_3, #nn4
99                             time_lagged_data_5, #nn5
100                            time_lagged_data_4, #nn6
101                            time_lagged_data_2, #nn7
102                            time_lagged_data_1, #nn8
103                            time_lagged_data_3, #nn9
104                            time_lagged_data_5, #nn10
105                            time_lagged_data_4, #nn11
106                            time_lagged_data_3) #nn12
107
108 #specifying the number of layers and nodes per layer
109 nn_parameters$layers <- list(c(8), #nn1
110                              c(6,4), #nn2
111                              c(5), #nn3
112                              c(12,6), #nn4
113                              c(4), #nn5
114                              c(6), #nn6
115                              c(4), #nn7
116                              c(7,3), #nn8
117                              c(10), #nn9
118                              c(4,2), #nn10
119                              c(5,3), #nn11
120                              c(6,2)) #nn12
121
122 #specifying if the output should be linear or not
123 nn_parameters$linear_output <- list(TRUE, #nn1
124                                     FALSE, #nn2
125                                     TRUE, #nn3
126                                     FALSE, #nn4
127                                     TRUE, #nn5
128                                     TRUE, #nn6
129                                     FALSE, #nn7
130                                     FALSE, #nn8
131                                     TRUE, #nn9
132                                     FALSE, #nn10
133                                     FALSE, #nn11
134                                     TRUE) #nn12
135
136
137 nn_parameters
138
139 #this is the training loop, it makes use of the nn_parameters list made above,
140 #it makes it easy to train a variety of models and experimenting with parameters
141 #I made an empty list which I use to store the model outputs after each loop.
142 nn_output <- list()
143 for (i in 1:length(nn_parameters$data)) {
144     #using my custom function to make the models and save the results for the evaluation metrics
145     nn_output[[i]] <- train_neuralnet(nn_parameters$data[[i]],
146                                     nn_parameters$layers[[i]],
147                                     nn_parameters$linear_output[[i]])
148     #renaming the keys in the list to make it easier to extract results
149     names(nn_output)[i] <- paste0("nn_", i)
150 }
151
152 nn_output
153
154 #using the NARX approach to include the 18th and 19th hour attributes
155 narx_input_data <- subset(data, select = c("18:00", "19:00", "20:00"))
156 head(narx_input_data)
157
158 # narx_create_time_lagged_data <- function(target_data, data, lag_values) {
159 #     time_lagged_data <- cbind(lapply(lag_values, function(x) lag(data, x)), target_data)
160 #     # Remove rows with missing values
161 #     # time_lagged_data <- time_lagged_data[complete.cases(time_lagged_data),]
162 #     #Rename the data with the relevant names

```

```

163 # # names(time_lagged_data) <- name
164 # return(time_lagged_data)
165 # }
166
167 #Above I tried to make a similar function to make the time lagged data,
168 #as 18th and 19th hour need to be accounted for I struggled to find a solution.
169 #So instead I just manually made the time lagged data for the narx approach as the actual result
170 #I was at least able to use the nn training function and loop with some minor tweaking.
171
172 # normalizing data before creating the delayed input I/O matrices
173 narx_normalized_input_data <- as.data.frame(lapply(narx_input_data, normalize))
174 head(narx_normalized_input_data)
175 #I had to rename the columns as after normalizing the data they changed for some reason
176 names(narx_normalized_input_data) <- c("18:00", "19:00", "20:00")
177
178 #begging to manually make the I/O matrices
179 narx_time_lagged_data_1 <- cbind(previous2_18= lag(narx_normalized_input_data$"18:00", 3),
180 previous1_19 = lag(narx_normalized_input_data$"19:00", 2),
181 current_19 = lag(narx_normalized_input_data$"19:00", 1),
182 predicted = narx_normalized_input_data$"20:00")
183
184 narx_time_lagged_data_1 <- narx_time_lagged_data_1[complete.cases(narx_time_lagged_data_1),]
185
186 narx_time_lagged_data_2 <- cbind(previous7_20= lag(narx_normalized_input_data$"20:00", 8),
187 previous7_19 = lag(narx_normalized_input_data$"19:00", 8),
188 previous4_18 = lag(narx_normalized_input_data$"18:00", 5),
189 previous2_20 = lag(narx_normalized_input_data$"20:00", 3),
190 current_19 = lag(narx_normalized_input_data$"19:00", 1),
191 predicted = narx_normalized_input_data$"20:00")
192
193 narx_time_lagged_data_2 <- narx_time_lagged_data_2[complete.cases(narx_time_lagged_data_2),]
194
195 narx_time_lagged_data_3 <- cbind(previous4_18= lag(narx_normalized_input_data$"18:00", 5),
196 current_19 = lag(narx_normalized_input_data$"19:00", 1),
197 predicted = narx_normalized_input_data$"20:00")
198
199 narx_time_lagged_data_3 <- narx_time_lagged_data_3[complete.cases(narx_time_lagged_data_3),]
200
201 narx_time_lagged_data_4 <- cbind(previous7_18 = lag(narx_normalized_input_data$"18:00", 8),
202 previous4_20 = lag(narx_normalized_input_data$"20:00", 5),
203 current_19 = lag(narx_normalized_input_data$"19:00", 1),
204 current_20 = lag(narx_normalized_input_data$"20:00", 1),
205 predicted = narx_normalized_input_data$"20:00")
206
207 narx_time_lagged_data_4 <- narx_time_lagged_data_4[complete.cases(narx_time_lagged_data_4),]
208
209
210 #for the NARX approach I am only making 5 models to see if there is any difference in outputs
211 #So for example any values at index 1 of the different keys of the nn_parameters list will b
212 #e used to train nn_1, index 2 for nn_2 and so on.
213 narx_nn_parameters <- list()
214 #specifying the data to be used to train the model
215 narx_nn_parameters$data <- list(narx_time_lagged_data_1, #nn1
216 narx_time_lagged_data_2, #nn2
217 narx_time_lagged_data_4, #nn3
218 narx_time_lagged_data_3, #nn4
219 narx_time_lagged_data_2) #nn5
220
221 #specifying the number of layers and nodes per layer
222 narx_nn_parameters$layers <- list(c(8), #nn1
223 c(10,4), #nn2
224 c(8,3), #nn3
225 c(6), #nn4
226 c(12)) #nn5
227
228 #specifying if the output should be linear or not
229 narx_nn_parameters$linear_output <- list(TRUE, #nn1
230 FALSE, #nn2
231 TRUE, #nn3
232 FALSE, #nn4
233 TRUE) #nn5

```

```

234
235 #I was getting an error saying "$ is invalid for atomic vectors" when running this function.
236 #After quite a while I found out that the test data was being turned into an atomic vector
237 #and needed to be accessed using the [] instead of $.
238 #R error messages aren't the greatest which made finding the issue even more challenging.
239 train_neuralnet <- function(data, layers, linear_output) {
240
241   #creating the training and testing batches for the nn, keeping the training sample to 380.
242   train_data <- data[1:380,]
243   test_data <- data[381:nrow(data),]
244
245   #making the nn using the neuralnet function.
246   nn <- neuralnet(predicted ~ .,
247                   data = train_data,
248                   hidden = layers,
249                   linear.output = linear_output)
250
251   #code for the total weights does not seem to be working
252   total_weights <- sum(sapply(nn$weights, length))
253   predicted <- predict(nn, newdata = test_data)
254   #this is where the error was happening.
255   actual <- test_data[1]
256   #using the RMSE, MAE, MAPE and SMAPE to evaluate the models,
257   #It is also saved into a list which makes it easy to view all results
258   metrics <- c("RMSE" = rmse(actual, predicted),
259               "MAE" = mae(actual, predicted),
260               "MAPE" = mape(actual, predicted),
261               "SMAPE" = smape(actual, predicted),
262               "Total Weights" = total_weights)
263   #plotting for visual representation
264   # plot(nn)
265   return(metrics)
266 }
267
268 narx_nn_output <- list()
269 for (i in 1:length(narx_nn_parameters$data)) {
270   narx_nn_output[[i]] <- train_neuralnet(narx_nn_parameters$data[[i]],
271                                           narx_nn_parameters$layers[[i]],
272                                           narx_nn_parameters$linear_output[[i]])
273   #renaming the keys in the list to make it easier to compare results
274   names(narx_nn_output)[i] <- paste0("nn_", i)
275 }
276
277 narx_nn_output

```