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: Nblust, Elbow method, Gap statistics and sillhoutte methods . 2

1.2.1 Nblust . . . 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: Nblust, Elbow method, Gap statistics and sillhoutte methods . . . 9

1.5.1 Nblust . . . 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: Nblust, Elbow method, Gap statistics and sillhoutte methods

1.2.1 Nblust

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

```
* Among all indices:
* 6 proposed 2 as the best number of clusters
* 12 proposed 3 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 1 proposed 11 as the best number of clusters
* 1 proposed 12 as the best number of clusters
* 2 proposed 15 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  3
```

1.2.2 Elbow Method

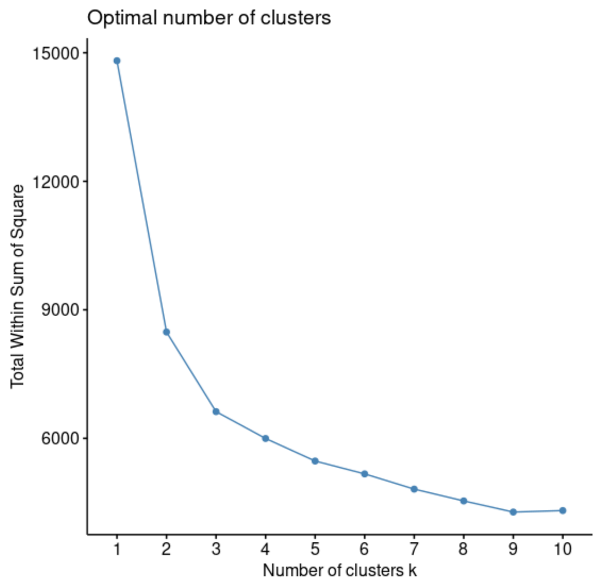

Figure 1: Elbow method plot

The Elbow method uses the **WCSS(within-cluster sums of squares)** which measueres how close data points are in respect of their cluster centers. Based on the plot above, the reccomended 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

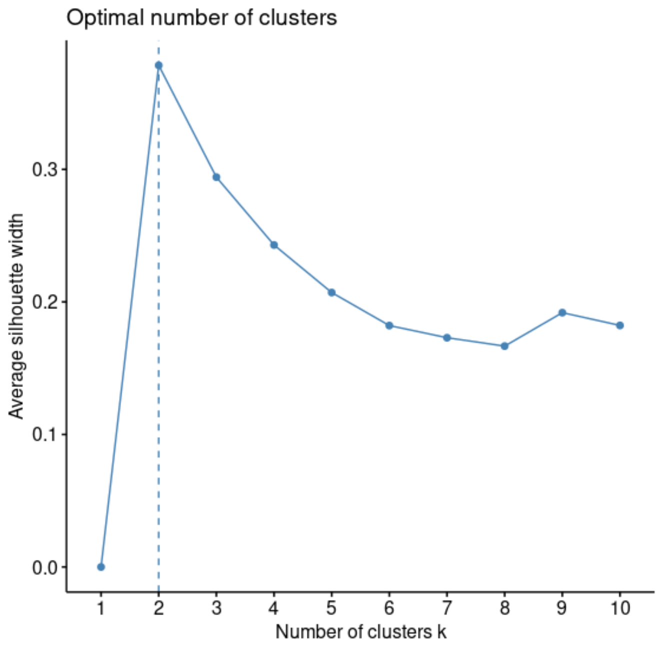

Figure 2: Gap statistics plot

The Gap statistics also uses the **WCSS** to calculate the best number of clusters to use. However, the reccomended number of clusters in this case is **2**, knowing that the orignal 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 Sillhoutte Method

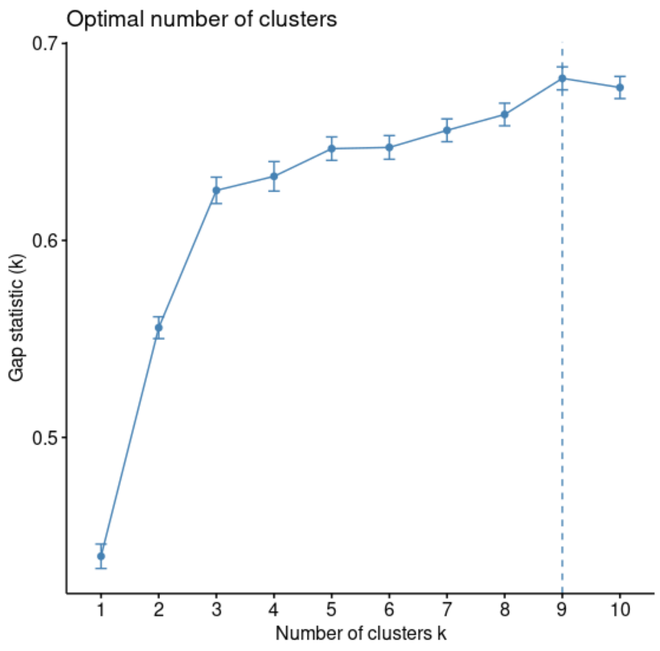

Figure 3: Sillhoutte method plot

The sillhoutte plot shows how similar a data point is to its own cluster using the **sillhuotte 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 reccomended 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 ouput everytime. This is by far the worst result as the orignal 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 sillhoutte 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-demensional led to an extreme result for the sillhoutte 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

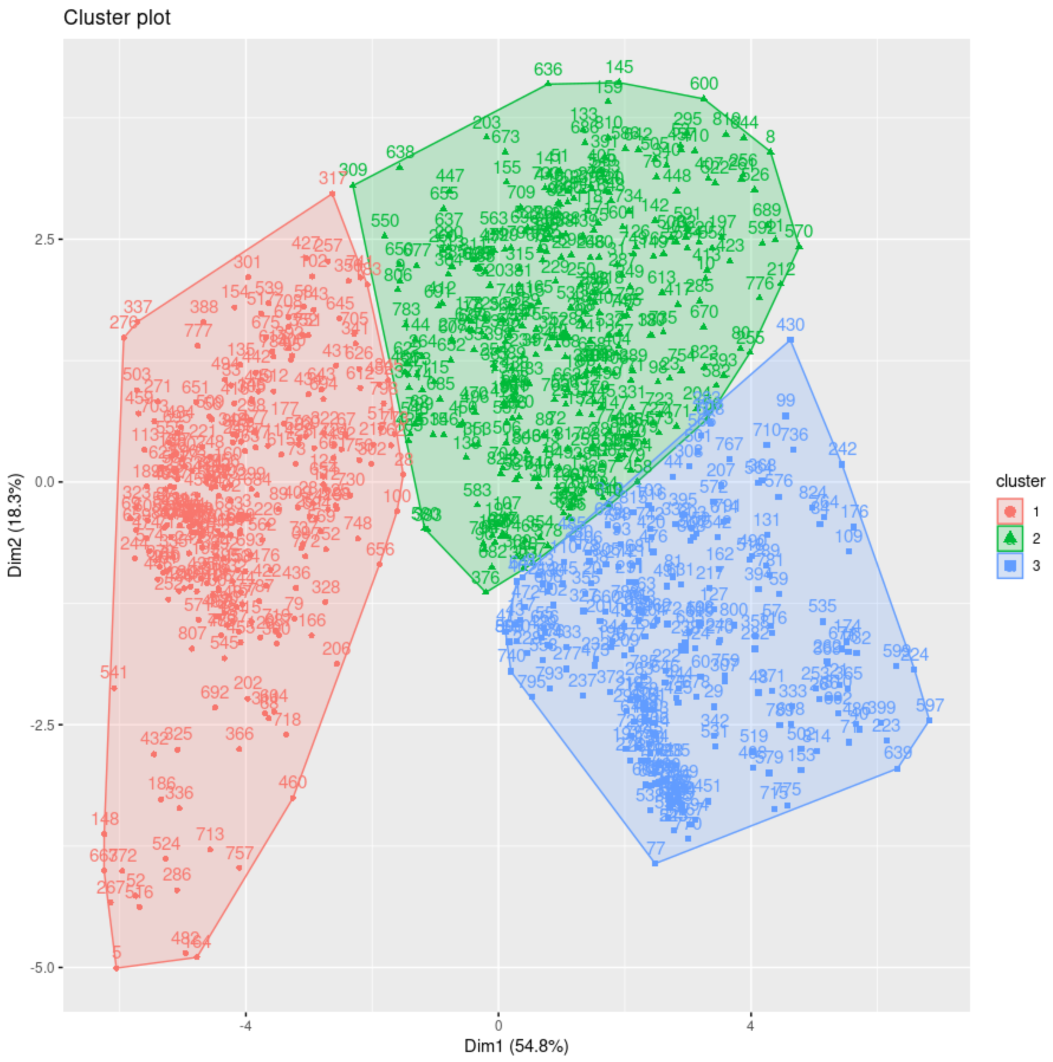

Figure 4: Clustering plot

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 ingnore 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 seperated from each other where a higher value means that clusters are well seperated 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 reccomended 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 the **WSS** was lower in the clustering attempt with **k=2**.

```
K-means clustering with 3 clusters of sizes 256, 331, 237

> kmeans_data$centers
        Comp       Circ     D.Circ       Rad.Ra Pr.Axis.Ra   Max.L.Ra    Scat.Ra
1  1.1672551  1.1913560  1.2226654  1.061855474  0.2398399  0.6675158  1.3141094
2 -0.2324797 -0.5226347 -0.2851558 -0.002041173  0.3625937 -0.1440161 -0.4446806
3 -0.9361458 -0.5569412 -0.9224294 -1.144132376 -0.7654747 -0.5198934 -0.7984081
       Elong Pr.Axis.Rect Max.L.Rect Sc.Var.Maxis Sc.Var.maxis     Ra.Gyr  Skew.Maxis
1 -1.2220251    1.3199740  1.1102132    1.2689258    1.3291991  1.0980640 -0.08461041
2  0.3064563   -0.4736786 -0.4874626   -0.3936680   -0.4533218 -0.5482611 -0.66286263
3  0.8919890   -0.7642435 -0.5184154   -0.8208477   -0.8026391 -0.4203797  1.01716369
   Skew.maxis  Kurt.maxis  Kurt.Maxis    Holl.Ra
1  0.16667482  0.27331007  0.01515673  0.2044549
2 -0.06083852 -0.01874875  0.75780956  0.6641968
3 -0.09506837 -0.26903603 -1.07474720 -1.1484793

> kmeans_data$cluster
  [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
 [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
 [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
[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 1 2 3 2 2 3 1 2 2 1 1 2 1 3 3 1 1
[165] 2 1 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
[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
[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
[288] 1 3 2 2 1 2 2 2 3 2 1 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
[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
[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
[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
[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
[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
[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
[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
[616] 3 1 3 3 2 3 2 1 1 2 1 2 2 3 2 3 1 2 1 3 2 2 2 3 2 3 2 1 2 1 3 2 2 2 2 1 2 3 1 2 1
[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
[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
[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
[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
[821] 2 1 2 3

> kmeans_data$tot.withinss
[1] 6624.09

> kmeans_data$betweenss
[1] 8189.91

Within cluster sum of squares by cluster:
[1] 2191.909 2735.763 1696.418
 (between_SS / total_SS =  55.3 %)
```

1.3.2 Sillhoutte Plot

The sillhouette plot shows how well the clustering is taking place and it will calucalte the average distance between the clusters. In practice the plot displays how close each point in one cluster is to poins 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 clsoe to **0** means the samples are more ambiguously placed and could be in another cluster.

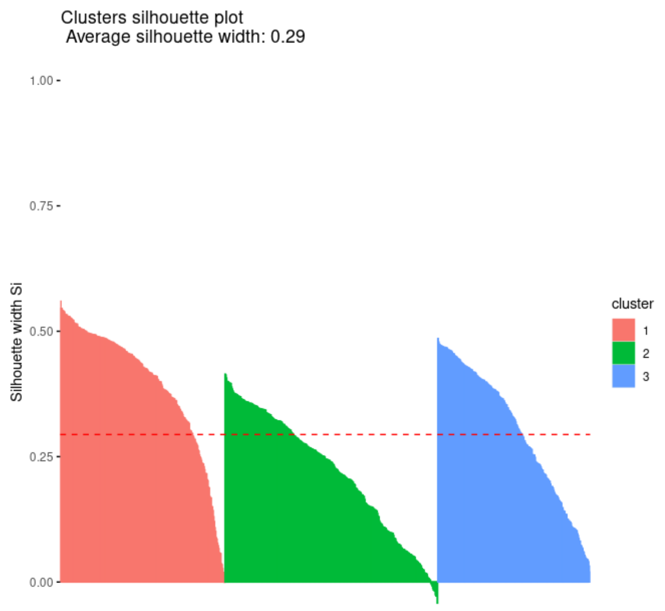

Figure 5: Sillhoutte plot

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 maxmimum 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 khown 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%**

```
> eigenvalues
 [1] 9.8655415144 3.3026315672 1.2050866140 1.1255984677 0.8773731809 0.6636174794
 [7] 0.3374343341 0.2274918343 0.1176165632 0.0871789864 0.0607683889 0.0450646831
[13] 0.0292070985 0.0213994038 0.0150961795 0.0123913361 0.0061400710 0.0003622977

> eigenvectors
                    PC1         PC2         PC3          PC4         PC5          PC6
Comp        -0.27099550  0.08819711 -0.03979285 -0.142474274 -0.15979926  0.219704493
Circ        -0.28538005 -0.14799378 -0.19761320  0.023348077  0.12602923 -0.019390179
D.Circ      -0.30078375  0.04064437  0.07450874 -0.104513476  0.07338676  0.000941066
Rad.Ra      -0.27595481  0.19284625  0.04085638  0.244080006 -0.12620414 -0.153234232
Pr.Axis.Ra  -0.10790106  0.24598582 -0.10092681  0.611908838 -0.05646656 -0.599471567
Max.L.Ra    -0.18693783  0.06836380 -0.10600156 -0.255241647  0.70801896 -0.255529947
Scat.Ra     -0.30925633 -0.07715243  0.10748098  0.001027495 -0.09117998  0.078463678
```

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


> summary(pca_data)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7     PC8
Standard deviation     3.1409 1.8173 1.09776 1.06094 0.93668 0.81463 0.58089 0.47696
Proportion of Variance 0.5481 0.1835 0.06695 0.06253 0.04874 0.03687 0.01875 0.01264
Cumulative Proportion  0.5481 0.7316 0.79851 0.86105 0.90979 0.94666 0.96540 0.97804
                           PC9    PC10    PC11   PC12    PC13    PC14    PC15    PC16
Standard deviation     0.34295 0.29526 0.24651 0.2123 0.17090 0.14629 0.12287 0.11132
Proportion of Variance 0.00653 0.00484 0.00338 0.0025 0.00162 0.00119 0.00084 0.00069
Cumulative Proportion  0.98458 0.98942 0.99280 0.9953 0.99692 0.99811 0.99895 0.99964
                          PC17    PC18
Standard deviation     0.07836 0.01903
Proportion of Variance 0.00034 0.00002
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 orignal k-means clustering attempt using the orignal dataset. Nbclust still says the best number of clusters is **3**.

```
* Among all indices:
* 7 proposed 2 as the best number of clusters
* 10 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 2 proposed 6 as the best number of clusters
* 2 proposed 9 as the best number of clusters
* 2 proposed 15 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  3
```

1.5.2 Elbow Method

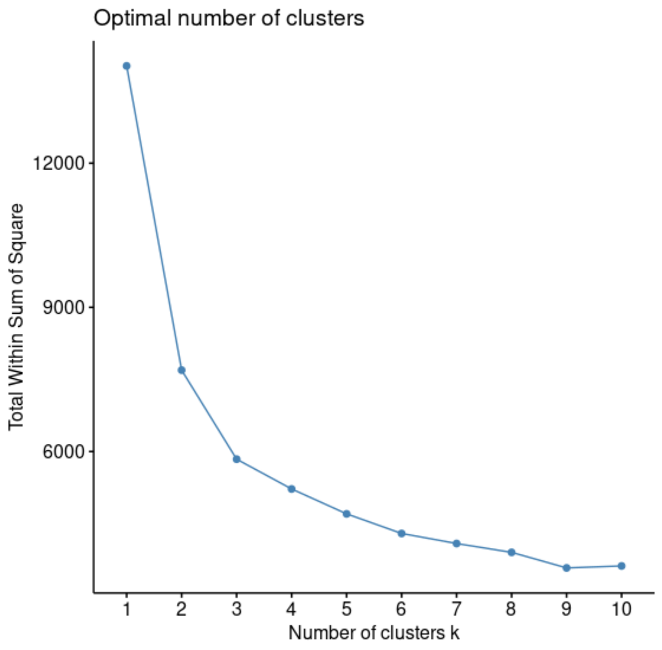

Figure 6: Elbow method plot

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

1.5.3 Gap Statistics

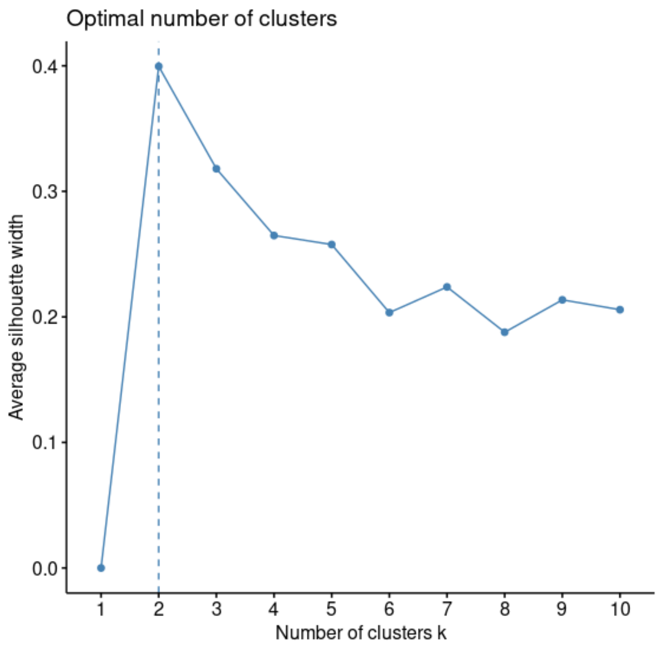

Figure 7: Gap statistics plot

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

1.5.4 Sillhoutte Method

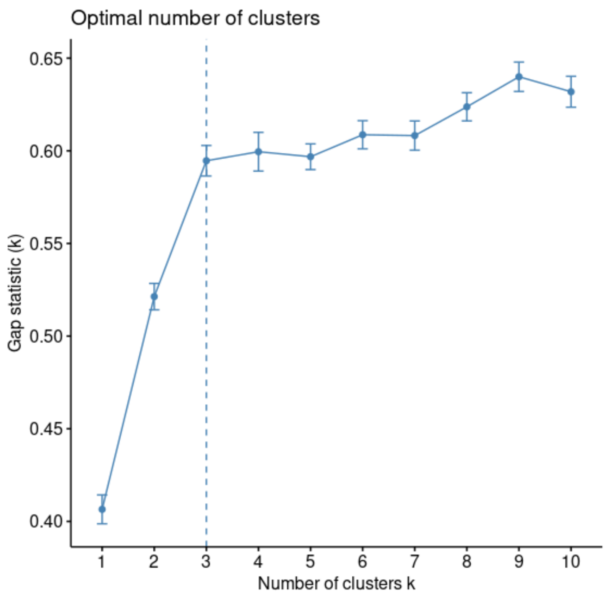

Figure 8: Sillhoutte method plot

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.

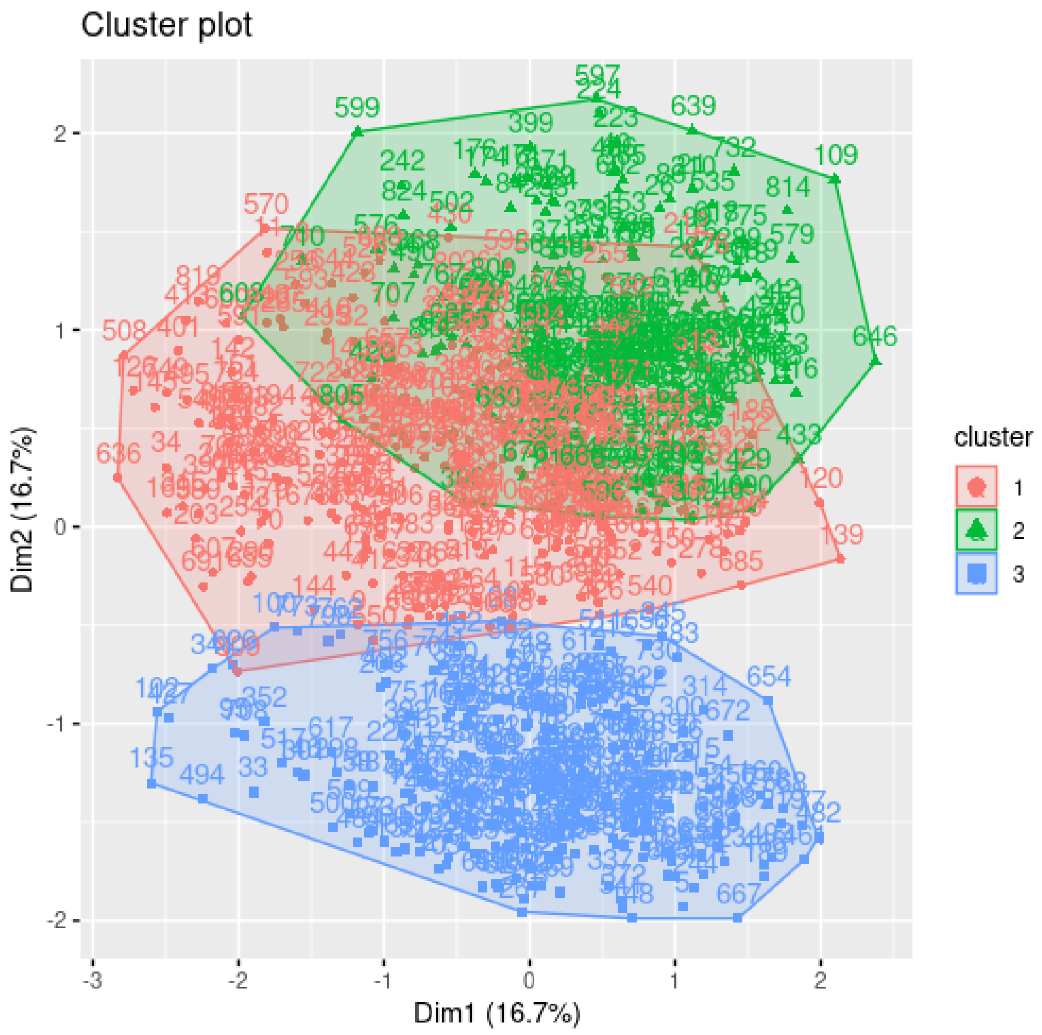

Figure 9: Clustering plot

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.

```
K-means clustering with 3 clusters of sizes 332, 236, 256

> kmeans_pca_data$centers
```

```
        PC1        PC2         PC3        PC4          PC5         PC6
1 -1.078026 -1.5110695  0.04723636 -0.1754898  0.009380699 -0.01659038
2 -2.911460  1.7332691  0.02469198  0.1345272 -0.123755941  0.11129772
3  4.082068  0.3618108 -0.08402257  0.1035711  0.101921914 -0.08108694

> kmeans_pca_data$cluster
  [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
2
 [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
1
 [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
1
[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
3
[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
1
[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
1
[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
1
[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
3
[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
2
[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
1
[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
2
[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
3
[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
1
[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
3
[575] 2 2 2 1 2 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
3
[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
3
[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
3
[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
2
[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
1
[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
1
[821] 1 3 1 2

> kmeans_pca_data$tot.withinss
[1] 5840.179

> kmeans_pca_data$betweenss
[1] 8183.617

Within cluster sum of squares by cluster:
[1] 2415.343 1461.091 1963.745
 (between_SS / total_SS =  58.4 %)
```

1.6.2 Sillhoutte 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

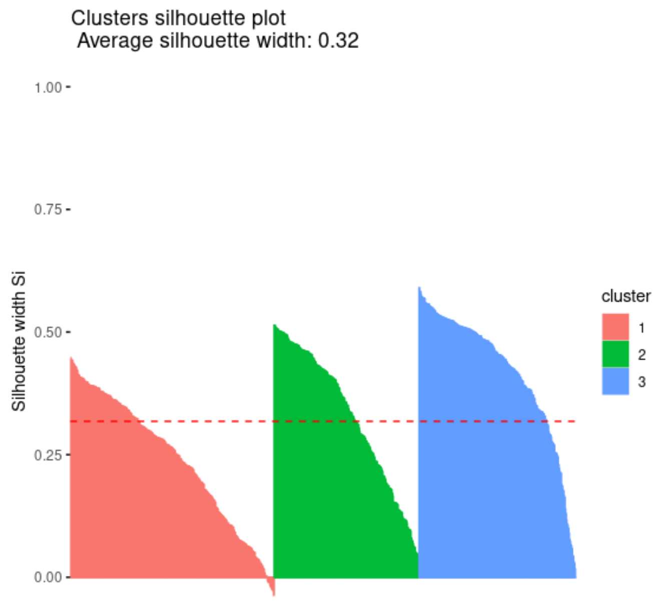

Figure 10: Sillhoutte plot

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 differeneces 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 ouput was negligeble 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 $\mathbf{NN_3}$ while the best **2-layer NN** is $\mathbf{NN_1}$**1**. 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 $\mathbf{NN_3}$ and $\mathbf{NN_1}$1 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 **NN_1**1 which had inputs from the periods **18(t-4), 19(t), 20(t+1)**.

A Code for Partitioning clustering

Listing 1: My R script

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

```
sum(is.na(data))

#using the boxplot to visually show the outliers
boxplot(data)
#use the sapply function to find the z-score for all the samples to then determine the outliers.
#I am creating a new column for the z-score for each of the samples,
#this makes it easier to remove the outliers.
data$zscore <- sapply(data, function(data) (data-mean(data))/sd(data))
#just double checking the dimensions of the original data set before removing outliers.
dim(data)
head(data$zscore)

#creating a new table that does not contain the outliers,
#anything that has a z-score lower than 3 and more than -3.
no_outliers <- data[!rowSums(abs(data$zscore) > 3), ]

#checking the maximum and minimum z-score to make sure the previous step worked as planned.
max(no_outliers$zscore)
min(no_outliers$zscore)
#new dimension of the new table, it is now 824 x 19 instead of 846 x 19
dim(no_outliers)

#this is just visually compare the original data set with the newly made one
boxplot(data)
boxplot(no_outliers)

#now removing the z-score column as it will change the result of the clustering
no_outliers <- subset(no_outliers, select = -c(zscore))
#scaling the data using the built-in scale function in R,
#I am also making a new dataframe to make comparisons easier.
no_outliers_normalised <- as.data.frame(scale(no_outliers))
#making sure the data was normalised properly
no_outliers$Rad.Ra
no_outliers_normalised$Rad.Ra

#using automated tools to determine the best number of cluster centers
library("NbClust")
library("factoextra")
library("cluster")
library("fpc")

#Nbclust
NbClust(no_outliers_normalised, distance = "euclidean", method = "kmeans", index="all")
#Elbow method
fviz_nbclust(no_outliers_normalised, kmeans, method='wss')
#Silhouette method
fviz_nbclust(no_outliers_normalised, kmeans, method='silhouette')
#Gap Stastics
fviz_nbclust(no_outliers_normalised, kmeans, method='gap_stat')

#performing kmeans clustering using the most favoured "k" from the automated methods used above.
kmeans_data <- kmeans(no_outliers_normalised, centers = 3, nstart = 10)
fviz_cluster(kmeans_data, data = no_outliers_normalised)

data_cluster <- data.frame(no_outliers_normalised, cluster = as.factor(kmeans_data$cluster))
head(data_cluster)

#showing the kmeans output
kmeans_data
#showing the within cluser summs of squares(WSS) and the between cluster sums of squares (BSS)
kmeans_data$centers
kmeans_data$tot.withinss
kmeans_data$betweenss
#showing the silhouette plot for the clustering
kmeans_data$cluster
silhouette <- silhouette(kmeans_data$cluster, dist(no_outliers_normalised))
fviz_silhouette(silhouette)

#applying PCA to the dataset to reduce dimensionality.
#this is the scaled dataset without the outliers
head(no_outliers_normalised)
```

```
pca_data <- prcomp(no_outliers_normalised, center = TRUE, scale = TRUE)
eigenvalues <- pca_data$sdev^2
eigenvectors <- pca_data$rotation
eigenvalues
eigenvectors
summary(pca_data)

#making a new dataset with only the first 6 PCA as the cumulative proportion for them is > 92%
transformed_data <- as.data.frame(-pca_data$x[,1:6])
head(transformed_data)
#new dimension is 824 x 6, thus reducing the attributes by 3 times.
dim(pca_data$x)
dim(transformed_data)

#applying the same 4 automated tools to find the new favoured k for the clustring
#Nbclust
NbClust(transformed_data, distance = "euclidean", method = "kmeans", index="all")
#Elbow method
fviz_nbclust(transformed_data, kmeans, method='wss')
#Silhouette method
fviz_nbclust(transformed_data, kmeans, method='silhouette')
#Gap Stastics
fviz_nbclust(transformed_data, kmeans, method='gap_stat')

#performing kmeans clustering using the new dataset
#performing kmeans clustering using the most favoured "k" from the automated methods used above.
kmeans_pca_data <- kmeans(transformed_data, centers = 3, nstart = 10)
fviz_cluster(kmeans_pca_data, data = transformed_data)

data_cluster <- data.frame(transformed_data, cluster = as.factor(kmeans_pca_data$cluster))
head(data_cluster)

#showing the kmeans output
kmeans_pca_data
#showing the within cluser summs of squares(WSS) and the between cluster sums of squares (BSS)
kmeans_pca_data$centers
kmeans_pca_data$tot.withinss
kmeans_pca_data$betweenss
#showing the silhouette plot for the clustering
kmeans_pca_data$cluster
silhouette <- silhouette(kmeans_pca_data $cluster, dist(transformed_data))
fviz_silhouette(silhouette)

#using the calinski-harabasz index
calinhara(transformed_data, kmeans_pca_data$cluster)
```

B Code for Energy forecasting

Listing 2: My R script

```
library("readxl")
library("dplyr")
library("neuralnet")
library("Metrics")

data <- read_excel("~/Documents/rcw/data/uow_consumption.xlsx")

#renaming the columns as they have weird names by default
names(data)[2] <- '18:00'
names(data)[3] <- '19:00'
names(data)[4] <- '20:00'
head(data)

#getting the 20:00 column only as it is our focus
input_data <- subset(data, select = c('20:00'))
input_data

#normalising the data
normalize <- function(x) {
return((x - min(x)) / (max(x) - min(x)))
}

#normalizing data before creating the delayed input I/O matrices
normalized_input_data <- as.data.frame(lapply(input_data, normalize))
normalized_input_data
head(normalized_input_data)
#function to create the time delayed input variables I/O matrices
create_time_lagged_data <- function(data, lag_values, name) {
    #this lets me specify the time lag and it will bind the time delayed column
    #to the orginal dataset orginal
    #which acts as the column for the predicted column, as it has no lag
    time_lagged_data <- cbind(lapply(lag_values, function(x) lag(data, x)), data)
    # Remove rows with missing values
    time_lagged_data <- time_lagged_data[complete.cases(time_lagged_data),]
    #Raname the data with the relevant names
    names(time_lagged_data) <- name
    return(time_lagged_data)
}

# Create different versions of the time-lagged data
time_lagged_data_1 <- create_time_lagged_data(normalized_input_data,
                                            c(3,2,1),
                                            c("previous2", "previous1", "current", "predicted"))
time_lagged_data_2 <- create_time_lagged_data(normalized_input_data,
                                            c(8, 2),
                                            c("previous7", "previous1", "predicted"))
time_lagged_data_3 <- create_time_lagged_data(normalized_input_data,
                                            c(8, 5:1),
                                            c("previous7", "previous4", "previous3", "previous2",
                                                "previous1", "current", "predicted"))
time_lagged_data_4 <- create_time_lagged_data(normalized_input_data,
                                            c(8, 3, 1),
                                            c("previous7", "previous2", "current", "predicted"))
time_lagged_data_5 <- create_time_lagged_data(normalized_input_data,
                                            c(2, 1),
                                            c("previous1", "current", "predicted"))

# Define a function to make training the nn models easier
train_neuralnet <- function(data, layers, linear_output) {

    #creating the training and testing batches for the nn, keeping the training sample to 380.
    train_data <- data[1:380,]
    test_data <- data[381:nrow(data),]

    #making the nn using the neuralnet function.
    nn <- neuralnet(predicted ~ .,
                    data = train_data,
                    hidden = layers,
                    linear.output = linear_output)

    # calculating the total number of weights in the neural network
    #I was not able to get the weights to show properly as it was giving an
    #output of 2 and 3 for all models
    total_weights <- sum(sapply(nn$weights, length))

    predicted <- predict(nn, newdata = test_data)
    actual <- test_data$predicted
    #using the RMSE, MAE, MAPE and SMAPE to evaluate the models,
    #It is also saved into a list which makes it easy to view all results
    metrics <- c("RMSE" = rmse(actual, predicted),
               "MAE" = mae(actual, predicted),
               "MAPE" = mape(actual, predicted),
               "SMAPE" = smape(actual, predicted),
                "Total Weights" = total_weights)
    #plotting for visual representation
    # plot(nn)
    return(metrics)
}

#I am making a list that holds all the parameters I will be using to train each of the 15 models
#So for example any values at index 1 of the different keys of the nn_parameters list
```

```
#will be used to train nn_1, index 2 for nn_2 and so on.
nn_parameters <- list()
#specifying the data to be used to train the model
nn_parameters$data <- list(time_lagged_data_1,#nn1
                       time_lagged_data_2,    #nn2
                       time_lagged_data_4,    #nn3
                       time_lagged_data_3,    #nn4
                       time_lagged_data_5,    #nn5
                       time_lagged_data_4,    #nn6
                       time_lagged_data_2,    #nn7
                       time_lagged_data_1,    #nn8
                       time_lagged_data_3,    #nn9
                       time_lagged_data_5,    #nn10
                       time_lagged_data_4,    #nn11
                       time_lagged_data_3)    #nn12

#specifying the number of layers and nodes per layer
nn_parameters$layers <- list(c(8),    #nn1
                            c(6,4),   #nn2
                            c(5),     #nn3
                            c(12,6),  #nn4
                            c(4),     #nn5
                            c(6),     #nn6
                            c(4),     #nn7
                            c(7,3),   #nn8
                            c(10),    #nn9
                            c(4,2),   #nn10
                            c(5,3),   #nn11
                            c(6,2))   #nn12

#specifying if the output should be linear or not
nn_parameters$linear_output <- list(TRUE,  #nn1
                                    FALSE, #nn2
                                    TRUE,  #nn3
                                    FALSE, #nn4
                                    TRUE,  #nn5
                                    TRUE,  #nn6
                                    FALSE, #nn7
                                    FALSE, #nn8
                                    TRUE,  #nn9
                                    FALSE, #nn10
                                    FALSE, #nn11
                                    TRUE)  #nn12


nn_parameters

#this is the training loop, it makes use of the nn_parameters list made above,
#it makes it easy to train a variety of models and experimenting with parameters
#I made an empty list which I use to store the model outputs after each loop.
nn_output <- list()
for (i in 1:length(nn_parameters$data)) {
    #using my custom function to make the models and save the results for the evaluation metrics
    nn_output[[i]] <- train_neuralnet(nn_parameters$data[[i]],
                                      nn_parameters$layers[[i]],
                                      nn_parameters$linear_output[[i]])
    #renaming the keys in the list to make it easier to extract results
    names(nn_output)[i] <- paste0("nn_", i)
}

nn_output

#using the NARX approach to include the 18th and 19th hour attributes
narx_input_data <- subset(data, select = c("18:00", "19:00", "20:00"))
head(narx_input_data)

# narx_create_time_lagged_data <- function(target_data, data, lag_values) {
#     time_lagged_data <- cbind(lapply(lag_values, function(x) lag(data, x)), target_data)
#     # Remove rows with missing values
#     # time_lagged_data <- time_lagged_data[complete.cases(time_lagged_data),]
#     #Raname the data with the relevant names
#     # names(time_lagged_data) <- name
#     return(time_lagged_data)
# }

#Above I tried to make a similar function to make the time lagged data,
#as 18th and 19th hour need to be accounted for I struggled to find a solution.
#So instead I just manually made the time lagged data for the narx approach as the actual result
#I was at least able to use the nn training function and loop with some minor tweaking.

# normalizing data before creating the delayed input I/O matrices
narx_normalized_input_data <- as.data.frame(lapply(narx_input_data, normalize))
head(narx_normalized_input_data)
#I had to rename the columns as after normalizing the data they changed for some reason
names(narx_normalized_input_data) <- c("18:00", "19:00", "20:00")

#begging to manually make the I/O matrices
narx_time_lagged_data_1 <- cbind(previous2_18= lag(narx_normalized_input_data$"18:00", 3),
                           previous1_19 = lag(narx_normalized_input_data$"19:00", 2),
                           current_19 = lag(narx_normalized_input_data$"19:00", 1),
                           predicted = narx_normalized_input_data$"20:00")

narx_time_lagged_data_1 <- narx_time_lagged_data_1[complete.cases(narx_time_lagged_data_1),]

narx_time_lagged_data_2 <- cbind(previous7_20= lag(narx_normalized_input_data$"20:00", 8),
                           previous7_19 = lag(narx_normalized_input_data$"19:00", 8),
                           previous4_18 = lag(narx_normalized_input_data$"18:00", 5),
                           previous2_20 = lag(narx_normalized_input_data$"20:00", 3),
                           current_19 = lag(narx_normalized_input_data$"19:00", 1),
                           predicted = narx_normalized_input_data$"20:00")

narx_time_lagged_data_2 <- narx_time_lagged_data_2[complete.cases(narx_time_lagged_data_2),]

narx_time_lagged_data_3 <- cbind(previous4_18= lag(narx_normalized_input_data$"18:00", 5),
                           current_19 = lag(narx_normalized_input_data$"19:00", 1),
                           predicted = narx_normalized_input_data$"20:00")

narx_time_lagged_data_3 <- narx_time_lagged_data_3[complete.cases(narx_time_lagged_data_3),]

narx_time_lagged_data_4 <- cbind(previous7_18 = lag(narx_normalized_input_data$"18:00", 8),
                           previous4_20 = lag(narx_normalized_input_data$"20:00", 5),
                           current_19 = lag(narx_normalized_input_data$"19:00", 1),
                           current_20 = lag(narx_normalized_input_data$"20:00", 1),
                           predicted = narx_normalized_input_data$"20:00")

narx_time_lagged_data_4 <- narx_time_lagged_data_4[complete.cases(narx_time_lagged_data_4),]


#for the NARX approach I am only making 5 models to see if there is any difference in outputs
#So for example any values at index 1 of the different keys of the nn_parameters list will b
#e used to train nn_1, index 2 for nn_2 and so on.
narx_nn_parameters <- list()
#specifying the data to be used to train the model
narx_nn_parameters$data <- list(narx_time_lagged_data_1,#nn1
                       narx_time_lagged_data_2,    #nn2
                       narx_time_lagged_data_4,    #nn3
                       narx_time_lagged_data_3,    #nn4
                       narx_time_lagged_data_2)    #nn5

#specifying the number of layers and nodes per layer
narx_nn_parameters$layers <- list(c(8), #nn1
                             c(10,4),    #nn2
                             c(8,3),     #nn3
                             c(6),       #nn4
                             c(12))      #nn5

#specifying if the output should be linear or not
narx_nn_parameters$linear_output <- list(TRUE,  #nn1
                                    FALSE, #nn2
                                    TRUE,  #nn3
                                    FALSE, #nn4
                                    TRUE)  #nn5
```

```r

#I was getting an error saying "$ is invalid for atmoic vectors" when running this function.
#After quite a while I found out that the test data was being turned into an atomic vector
#and needed to be accessed using the [] instead of $.
#R error messages aren't the greatest which made finding the issue even more challening.
train_neuralnet <- function(data, layers, linear_output) {

    #creating the training and testing batches for the nn, keeping the training sample to 380.
    train_data <- data[1:380,]
    test_data <- data[381:nrow(data),]

    #making the nn using the neuralnet function.
    nn <- neuralnet(predicted ~ .,
                    data = train_data,
                    hidden = layers,
                    linear.output = linear_output)

    #code for the total weights does not seem to be working
    total_weights <- sum(sapply(nn$weights, length))
    predicted <- predict(nn, newdata = test_data)
    #this is where the error was happening.
    actual <- test_data[1]
    #using the RMSE, MAE, MAPE and SMAPE to evaluate the models,
    #It is also saved into a list which makes it easy to view all results
    metrics <- c("RMSE" = rmse(actual, predicted),
               "MAE" = mae(actual, predicted),
               "MAPE" = mape(actual, predicted),
               "SMAPE" = smape(actual, predicted),
                "Total Weights" = total_weights)
    #plotting for visual representation
    # plot(nn)
    return(metrics)
}

narx_nn_output <- list()
for (i in 1:length(narx_nn_parameters$data)) {
    narx_nn_output[[i]] <- train_neuralnet(narx_nn_parameters$data[[i]],
                                           narx_nn_parameters$layers[[i]],
                                           narx_nn_parameters$linear_output[[i]])
    #renaming the keys in the list to make it easier to compare results
    names(narx_nn_output)[i] <- paste0("nn_", i)
}

narx_nn_output
```