**Partitioning Clustering**

**1st Subtask Objectives**

**(A)**

Before conducting the K-means it is important to pre-process, in this case scaling and removing outliers. Outliers must be removed first otherwise they could influence the scaling results and skew the data for future analysis and methodologies. If objects in the data set have a large variance, but only due to one data value, for example, it could mean that results deviate too far in a specific direction and allow a singular point to have too much influence, for example on the mean. To remove outliers, I initially chose to use quartiles however, this was removing too many data points, so I decided to use boxplots instead as they give more accurate results. Furthermore, it gives the ability for the outliers to be visualised. I used to code below for implementation:

A screenshot of a computer code

Description automatically generated with low confidence

A picture containing diagram, line, technical drawing, white

Description automatically generatedThe library installed (boxplotdbl) allows the visualisation to be implemented. Each variable of the dataset is listed and then analysed, and all outliers are removed giving the following results:

The graph above shows the outliers defined by the circular shape. It is clear these are outliers as they lay beyond the median quartiles (shown by the boxes). These outliers can be numerically accounted for by the new data frame created ‘no outliers. As shown below, compared to the original data frame ‘new\_dataframe’, there is now only 833 observations instead of 846. It is important to note the variables ‘sample’ was not included as it is not necessary for analysis, and ‘class as it is nonnumerical.



Now the outliers are removed, it is necessary to scale the data. It is important that all data is represented on an even as possible scale in order to give the most accurate results later. I decided to use the Z-Score instead of min-max normalisation as data can become more orientated towards the mean and Z-Score generally gives more accurate results. Again, the columns scaled did not include the sample number or the class of vehicle as this would be futile. The code and results are shown below:



The Scale() function used produced a new data frame ‘no\_outliers\_NormZ’ and all data scaled as shown below:

A screenshot of a calculator

Description automatically generated with medium confidence

As we can see the data has transitioned from integers to long decimal numbers, which shows evidence they have been scaled. Now that the variables are all on a similar range, it will be much easier to analyse them.

**(B)**

Now the data can be used for analysis and learning, it’s important to find how many cluster centres are required. Knowing the correct number of clusters will give more accurate results, if we have too many or too few clusters, then results can become less accurate. According to the **NBclust** method and using the following the code:

A black text on a white background

Description automatically generated with low confidence

The majority rule dictates and shows via the results below that 2 is the best number of clusters.

A picture containing text, screenshot, font, algebra

Description automatically generated

On the other hand, we can also interpret the necessary number of clusters via the **Elbow** method. This provides a more visual approach where the required number of clusters can be shown via an ‘elbow’ shape on the graph. To get these results the following code was implemented:

A picture containing text, screenshot, font, white

Description automatically generated

Which gave the following result:

A picture containing text, diagram, line, plot

Description automatically generated

Interpreting these results, we can see that the WSS begins to level off and show a significantly slower rate of decrease at k=7, therefore that is the recommended number of clusters. The next automated method to determine the necessary number of clusters is the **Gap Statistic Algorithm**, which uses a comparison of the total intra-cluster variation through different K. The required k is the one that it maximised. To gain this we use the below code:

A white background with black text

Description automatically generated with low confidence

Fortunately for the gap statistic algorithm, R has a built-in function as seen above ‘fviz\_nbclust’ which allows quick efficient results as shown below:

A picture containing text, line, diagram, plot

Description automatically generated

The dotted blue line intersecting at point 3 represents where the gap statistic is maximised. Through initial visitation, one could be swayed to choose K =10 however, considering the number of clusters to the gap statistic, we can tell this is not the maximal.

The **Average Silhouette Model** aims to show the ideal number of clusters through determining how well each data point fits within its cluster. The best cluster is the one that maximizes the average silhouette over a range of possible values for k, it is also important to note the next best option. To get the average silhouette I used the following code:

A black text on a white background

Description automatically generated with low confidenceWhich gave the following results:

A picture containing line, diagram, plot, slope

Description automatically generated

Based on these results the optimal number of clusters is K=2 and the next best option would be 3. Overall, 2/4 of the autonomous results recommend clusters = 2 whilst the others recommend 3 and 7 respectively. Therefore, moving forward I will use k=2 as my number of clusters

**(C)**

Now we have chosen the number of clusters, we can conduct relevant analysis. The first analysis completed was finding the ratio of BSS to TSS as this will help to show how much variance there is within our two clusters. To do this we use the code below:

A picture containing text, screenshot, font, white

Description automatically generated

Which gives the below output:

A picture containing text, font, white, receipt

Description automatically generated

The ratio of BSS to TSS as represented above is 42.8% which represents that K=2 is a below average fit but also that there may be an alternative number of clusters that can produce a better score. However, using the following code we can visualise the clusters as demonstrated below:

A picture containing text, font, screenshot, algebra

Description automatically generated

A picture containing text, map, diagram, screenshot

Description automatically generated

As we can see the data is largely placed into two groups via the right and left-hand side. Out of curiosity I tried alternative numbers of clusters, however, anything more than 2 and the clusters begin to overlap, and the data becomes part of numerous clusters, therefore this is evidence k=2 is strong (as shown above)

Next it was important to find the eigenvalues and eigen vectors as these represent the area and relationship between the two clusters. specifically, the WSS, to show the average distance within a cluster, and the BSS the average distance between clusters were calculated with the following code:

A picture containing text, font, screenshot, white

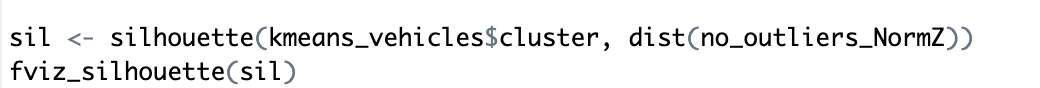
Description automatically generated

Which gave the following output:

The BSS results represents the distance both clusters are from one another, therefore the greater the distance between them then the more accurate the cluster are in their ability to group together data with specific patterns. As shown above the BSS is 6412.99 which is a very good and high score, however, the WSS which represents the widthof each cluster is 8563.01 which is a very high score. This is due to the fact that there are only two clusters, and the dataset has a large quantity of objects meaning that they are widely spread amongst the two clusters.

**(D)**

The silhouette plot represents how close data points in one cluster are, to the data points in the next cluster. In order to find the average silhouette width, score it’s important to use the average silhouette model. In order to obtain the model, the following code was used:



Which produced the following results:

A picture containing text, screenshot, diagram, font

Description automatically generated

A black text on a white background

Description automatically generated with low confidence

The results above show the accuracy and fit of the clusters where k=2. In this instance the results show the individual clusters width. The silhouette coefficient is measured between -1 and 1. A score of 1 represents the perfect score and -1 the worst. The closer to 1, the more compact the data point is within its belonging cluster, and it also shows its far away from other clusters. In this example the average for both clusters are:

1 = 0.40

2 = 0.37

Which represents a very good score and emphasises that the clusters are accurate and acceptable for pre-processing.

**2nd Subtask Objectives**

**(E)**

Now we have scaled and cleaned our data we can reduce its dimensionality whilst still retaining a high level of accuracy by completing the PCA (principal component analysis). Below represents the code used and the following outputs:

A computer code with text

Description automatically generated

A number and numbers on a white background

Description automatically generated

The PCA gives us 3 results to interpret for each of the 18 attributes: standard deviation, proportion of variance, and cumulative proportion. The standard deviation explains the dataset in the context of the PCA itself and essentially represents the eigenvalues as our data is already standardized. The proportion of variance indicates the level of variability. The cumulative proportion is accumulation of explained variance, in this case we are only interested in results >92%.

For this PCA we need to find >92% for the cumulative proportion so we use the first 6 PCA’s as their results fit the requirements, even though PC6 is above the threshold it is only minimal and more PCA’s will give more accurate results. Now we need to create a new dataset with the first 6 PCA’s as the attributes using the code shown below:



Which gives the following results below:

A table of numbers on a white sheet

Description automatically generated

We have now created a new data frame called ‘no\_outliers\_NormZ\_pca’ (shown above) which represents all 6 attributes and their values.

**(F)**

With the new dataset it is now appropriate to run similar tests to that used in the original dataset to see the recommended number of Ks. Below represents the code and their findings:

**NBclust**

A close-up of a computer code

Description automatically generated

A black text on a white background

Description automatically generated

**Elbow**

A close-up of a computer code

Description automatically generated

A graph with numbers and a line

Description automatically generated

**Gap statistic**

****

**A graph with blue dots and a line

Description automatically generated**

**Average silhouette model**

**A graph with a line and a blue line

Description automatically generated**

The table below represents the findings from both the original and new data frame and shows the differences (if any) between the recommended number of K’s.

|  |  |  |
| --- | --- | --- |
|  | **Original Dataset** | **PCA dataset** |
| **NBclust** | 2 | 2 |
| **Elbow** | 7 | 7 |
| **Gap Statistic** | 3 | 3 |
| **Average Silhouette** | 2 | 2 |

As we can see from the table above, none of the results differ from either data frame. This shows that the PCA analysis and results are very accurate as we have managed to reduce the dimensionality of the data whilst still maintaining the same results. In this instance we have achieved a very credible set of results.

**(G)**

As the cluster results are the same as the original dataset, I have concluded to use the same number of clusters before, which was 2. First, we need to find the ratio of BSS to TSS as this will show the amount of variance between the two clusters. To do this we use the code below which gave the following results:

A close up of a text

Description automatically generated

A black text on a white background

Description automatically generated

As we can see the ratio is 45% which is still below average but represents a 2.2% increase in accuracy compared to the original data frame. We can now visualise these clusters with the code below which gives the following results:



A graph showing different colored squares

Description automatically generated

As we can see the dimensionality of the clusters has considerably reduced. On the original dataset, dim1 = 18.9% and dim2 = 54.7% compared to the new clusters above where dim1 = 16.7% and dim2 =16.7%. Next, we have to find the eigenvalues and eigen vectors which represent the area and relationship between the two clusters. To do this we use the code below which gives the following results:

A close-up of a text

Description automatically generated

As we can see the new BSS and WSS in comparison to the results from the original data frame are lower. Reducing the dimensionality has essentially removed certain data points which means those that were originally close together may now not be, thus creating a greater gap between clusters or in their own cluster.

**(H)**

We can now analyse the efficiency of each cluster using the silhouette plot to represent how close data points in one cluster are, to the data points in the next cluster. In order to obtain the model, the following code was used, which gave the following results:



A graph with a red line

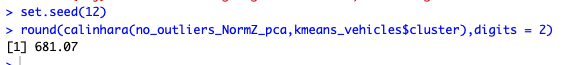
Description automatically generatedA black text on a white background

Description automatically generated

As we already know an average silhouette width score close to 1 represents a good, compact cluster, whereas a score close to -1 indicates a very poor cluster formation. Using the new data frame we have obtained a score of 0.4 which is adequate and also represents an increase in accuracy compared to the original data frames score of 0.38.

**(I)**

The Calinski-Harabasz index is essentially a metric to evaluate the quality of clustering results where it assesses the separation between different clusters. A good result will indicate well separated clusters with high intra-cluster similarity. Obviously, this will require the use of BSS and WSS where a comparison is made between the ratio dispersion. To obtain this metric we use the following code which gives the following results:



As we can see from the code above, our Calinski-Harabasz index = 681.07. As the CH index is ultimately heuristic, we act on the assumption a high score is indicative of good clustering, therefore in this example, our results are very positive.

**Financial Forecasting**

**(A)**

The exchange rate is one of the most influential markets in the financial industry, its influence can be felt throughout whole populations and therefore the ability to predict its movement is significant. In this part of the report we will attempt to use neural networks to predict the changes in the exchange rate for the coming day, therefore we will need to use adequate input variables. Input variables are what the mlp uses to make the prediction, therefore they are normally column names from the used data frame.

Input vectors are also highly important for financial forecasting and for our report we will use the autoregressive approach. This method uses past data from a time-series in order to make future predictions, this is only suitable when data has low variance, or when data seems to follow a pattern. Another method which could be used however, is exponential smoothing. In this instance a prediction would be a weighted linear sum of past observations where you assign exponentially decreasing weights to past observations.

**(B)**

A screenshot of a computer program

Description automatically generatedUsing the time series data from the data frame created (‘Exchange\_Rate’) we have to create an input/output matrix and test different time delayed input factors. The code below represents the methodology used for this:

Firstly we install all suitable packages and then label our data frame with accurate column headings to make the following processes easier. Then we create the vectors and use different time delays to see which one is most appropriate. In this case I used a delay time of 5 to create the prediction, meaning that it interprets the last 4 data values before making its prediction, this is shown by the lag function. Then we create a new data frame with suitable headings to show the results which can be shown below:

A screenshot of a table

Description automatically generated

This data frame is called ‘Exchange\_Vectors’ and as we can see the N/A sections represent where the previous data points are being used for the prediction column.

**(C)**

Next, we need to normalise our data. This is important because under the assumption the data has a constant mean and variance, by normalising the data we ensure all data points share a similar scale thus preventing a single variable from dominating the model’s behaviour due its specific size. We are unable to use the code to normalise from the partition clustering part as this does not remove out N/A values which will prevent us from having a working neural network, instead to normalise the matrix we use the lapply and function, so each value is between 0 and 1, as shown below:

A computer code with black text

Description automatically generated

We have now created a new matrix called ‘Exchange\_Vectors\_NOR where all the data values have now been normalised. We can see the change in values below where all data values are now between 0 and 1:

A table of numbers on a white background

Description automatically generated

**(D)**

A computer code with green and blue text

Description automatically generatedNow we have the data ready we can begin training on appropriate models for the neural network. It is important to experiment with different hidden layers, hidden neurons and input vectors to find the most accurate neural network for the forecasting. To build the neural network I used the following code:

This can be visualised as seen below:

A diagram of a network

Description automatically generated

Using this information we are then able to create our final neural network later. It is important to note this is not the final neural network, only an example using random levels of hidden layers etc to represent how we can calculate our final neural network and save different parameters to.

Now, it is important be able to able to calculate our indices (RMSE, MAE, MAPE, and sMAPE) in order to get the most accurate neural network

**(E)**

Before calculating these indices it’s important we fully understand what each one represents. Below shows their definitions and what they are indicative of:

RMSE is the root mean square error, which in its simplest, is a way to measure differences between values. It does this by quantifying the average difference between the predicted values of a model and the actual values in the dataset and it is normally used in regression models.

MAE is the mean absolute error and is taken by the average size of mistakes from a number of predictions, it represents the difference between the measured value and the ‘true’ value.

MAPE is the mean absolute percentage error and defines the accuracy of a forecasting method. It represents the average absolute percentage errors from each data point to calculate how accurate the forecasted predictions were in comparison with the actual predictions. MAPE is normally more accurate in larger datasets.

sMAPE is the symmetric mean absolute percentage error. sMAPE normalises relative errors by dividing by both actual and predicted values. However, sMAPE can also be figured using the same premise but by making certain parameters divisible by 2. It represents an accuracy measurement based on errors.

With all of the above indices we are looking for a result as close to zero as possible, the smaller the result the more accurate our neural network will be.

**(F)**

Before we conduct our final tests, we need to use a collection of code to ensure the accuracy of the neural network. Below represents the code I used and a description of their significance:

A computer screen shot of a program

Description automatically generated

Firstly we create new data frames for training and test data per the requirements of the report, the first 400 data points are used for training whilst the remaining is used for testing. We then also obtain our predicted values from the results of the initial NN before unnormalizing the data we have. We do this because it is not necessary for data to be normalised as we are no longer analysing and want to build a fully accurate NN. By Unnormalizing the data we get the following results:

A screenshot of a computer

Description automatically generated

Now we have both our actual data and predicted data via our NN we can begin to find our performances metrics as discussed above. To do this I used the following code which gave the following results:

A screenshot of a computer code

Description automatically generated

Now, using the table below we can test with different hidden layers, hidden neurons and different input vectors to discover the best NN to use.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Configuration** | RMSE | MAE | MAPE | sMAPE |
| 2 input vectors, 2 hidden layers, 20 hidden neurons | 0.07098693 | 0.05305636 | 0.03991027 | 0.04045127 |
| 2 input vectors, 3 hidden layers, 10 hidden neurons | 0.07100318 | 0.05314489 | 0.03997427 | 0.04051767 |
| 2 input vectors, 4 hidden layers, 20 hidden neurons | 0.07100318 | 0.05314489 | 0.03997427 | 0.04051767 |
| 2 input vectors, 5 hidden layers, 10 hidden neurons | 0.07100318 | 0.03346132 | 0.03896429 | 0.04051767 |
| 2 input vectors, 7 hidden layers, 20 hidden neurons | 0.07100318 | 0.05314489 | 0.03997427 | 0.04051767 |
| 2 input vectors, 2 hidden layers, 10 hidden neurons | 0.07100318 | 0.05314489 | 0.03997427 | 0.04051767 |
| 2 input vectors, 3 hidden layers, 20 hidden neurons | 0.07100318 | 0.05314489 | 0.03997427 | 0.04051767 |
| 2 input vectors, 4 hidden layers, 10 hidden neurons | 0.07100318 | 0.05314489 | 0.03997427 | 0.04051767 |
| 2 input vectors, 5 hidden layers, 20 hidden neurons | 0.07100318 | 0.05314489 | 0.03997427 | 0.04051767 |
| 2 input vectors, 7 hidden layers, 20 hidden neurons | 0.07100318 | 0.05314489 | 0.03997427 | 0.04051767 |

For some reason all but the first index’s give the same result, I am not sure if this is down to me trying to apply the code in the wrong way or something wrong with my NN itself.

**(G)**

No we have completed a variety of tests on our neural network and created our comparison table we can finally conclude on the best parameters to use to create the most efficient neural. Network. When doing this we need to consider specific results such as size, speed, and our index results. From the table above it is very evident that the configuration of 2 input vectors, 2 hidden layers, 20 hidden neurons. It gives us the following index results:

|  |  |  |  |
| --- | --- | --- | --- |
| RMSE | MAE | MAPE | sMAPE |
| 0.07098693 | 0.05305636 | 0.03991027 | 0.04045127 |

Furthermore, it takes only about 2 seconds to be built and give us results.

A screenshot of a computer

Description automatically generatedWe can also see this it has multiple weights which will increase the accuracy of the neural network. Therefore I conclude that this is the best option for our neural network.

**(H)**

Finally then, our best neural network gives the following result:

A computer generated image of a network

Description automatically generated

A red square with black text

Description automatically generatedWe can also use the following code to create a scatterplot to show the accuracy of our neural net. Ideally, we want our data to be ‘hugging’ the line.

A graph with red dots

Description automatically generated

As we can see our data is highly dispersed. This plot represents our predicted vs actual data and visually, only about 10% of the data lingers around the line. This represents a not so accurate neural network. I am unsure on where the neural network became skewed as I have run different tests using different code a variety of time but see now real accurate changes.