Business Intelligence

Coursework 1

Lukas Kopecky

5th November 2020

Objective 1: Partitioning Clustering

Pre-porcessing tasks

Usually, every dataset are affected by some kind of noise. Noisy values are either missing records, mistaken or outliers.

We can deal with missing values by simply deleting those records or entering a mean value of the column, as such won't affect the variable. [1]

Finding Outliers

Statistics defines outliers as data points that defers significantly from other observations. They might be caused by wrong measurement such as sensor failure or caused by inconsistent data entry. For example in case in data set {1.80, 1.75, 1.50, 1.66, 174, 1.79} representing human height in meters the value 174 is clearly an outlier. However, with further investigation we will realise that the value was entered in centimetres instead of meters and as such can be fixed easily. Other solution to fix outliers is simply deleting the whole record from our dataset.

Nevertheless, outliers aren't always erroneous values. In some cases, outliers are desired values that we are aiming to detect [1].

There are various methods to identify them. Simple statistical method is the most common on. First we need to find IQR (Inter Quartile Range) which is area between 25th and 75th percentiles (*IQR = Q3 - Q1).* The next step is to find upper range and lower range of the dataset *(UP = IQR * 1.5 + Q3; LOW = IQR * 1.5 - Q1)*. Values higher than upper range or smaller than lower range are outliers. *(OUTLIERS = value > UP || value < LOW)*.

Following code shows applying statistical method in R:

1. Identifying Quartiles: *Q1 and Q3*

```
Q <- quantile(data$column, probs=c(.25, .75), na.rm = FALSE)
```

2. Finding Inter Quartile Range: *IQR = Q3 - Q1*

```
iqr <- IQR(data$column)
```

3. Identify Outliers: *Outliers < Q1 - (1.5 * IQR) OR Outliers > Q3 + (1.5 * IQR)*

```
up <-  Q[2]+1.5*iqr   # Upper Range


low<- Q[1]-1.5*iqr   # Lower Range
```

4. Deleting Outliers from the dataset *(Variable 'eliminated' is a data frame that doesn't contain outliers.)*

```
eliminated <- subset(data, data$column > (Q[1] - 1.5*iqr) & data$column < (Q[2]+1.5*iqr))
```

However, R language offers a **box-plot** method for displaying quartiles that is capable to identify outliers graphically and can return outlier values [2].

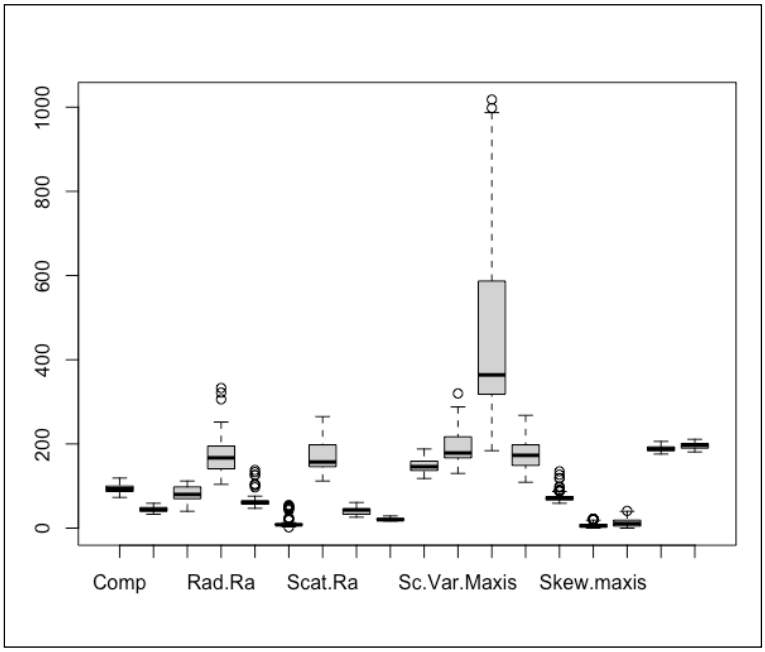


Figure 1: Displays all variables in one chart. Circles represent outliers

'boxplot' function with value $out retrieves all outlier values. The function 'which' identifies indices of those values and saves them into a list 'col4'.

```
col4 <- which(vehicles.raw$Rad.Ra %in% boxplot(vehicles.raw[4], plot=FALSE)$out)
```

Running this process against all columns will produces several list of indices. Appending those list together and **searching for unique** values identifies **33 unique outliers.**

```
#append all columns together
allColumns <- append(col4, c(col5, col6, col11, col12, col14, col15, col16))


#check for unique outlier indices
outliers <- unique(allColumns) # = 33 Unique Outliers
 [1]  38 136 389   5 101 292 524 707 128 392 545 656 816  86 836  48  80 231 382 499  45
[22] 114 124 191 347 401 506 517 624 762 797 798 133


#delete outliers
vehicles.clear <- vehicles.raw[-outliers,]
```

The last steps **removes** outliers and saves data into **'vehicles.clear'** data frame.

Scaling data

Majority of the machine learning algorithms using Euclidian distance to measure space between two data points, thus a high variety of range, and units may cause a problem. For example, one variable represented in Meters and the other one in millimetres [3].

There are various methods to solve this issue such as Standardisation, Mean Normalisation and Scaling.

Function 'scale' in R distributes values equally with mean value equal to 0 in range -4:4.

Optionally, data could be normalised our dataset in rage 0:1 instead. However, scaled data were proposing better results.

```
vehicles.scaled <- scale(vehicles.clear)
summary(vehicles.scaled)
```

Comp	Circ	D.Circ	Rad.Ra
Min. :-2.5575	Min. :-1.9254	Min. :-2.7065	Min. :-2.02198
1st Qu.:-0.8204	1st Qu.:-0.7869	1st Qu.:-0.7798	1st Qu.:-0.85233
Median :-0.0760	Median :-0.1362	Median :-0.1376	Median :-0.06202
Mean : 0.0000	Mean : 0.0000	Mean : 0.0000	Mean : 0.00000
3rd Qu.: 0.7925	3rd Qu.: 0.6770	3rd Qu.: 1.0184	3rd Qu.: 0.82312
Max. : 2.7777	Max. : 2.3035	Max. : 1.9175	Max. : 2.46696

Finding the ideal number of clusters

As k-Means is unsupervised learning method, we normally do not know number of classes. Thus, we need to determine the number of clusters ourselves. If we have some knowledge of the data we working with, we can estimate the number of clusters manually, run k-means algorithm several times and pick the most reasonable one.

However, there are some automated tools as well

The Elbow method:

The elbow method is a heuristic (discover) method proposing the best number of clusters by plotting explained variation as a function of the number of clusters and picking the elbow of the curve as the number of clusters.

```
set.seed(26)
wss <- 0
for (i in 1:15){
  wss[i] <- sum(kmeans(vehicles.scaled, centers=i)$withinss)
}

plot(1:15, wss, type="b", xlab="Number of Clusters",ylab="Within groups sum of squares")
```

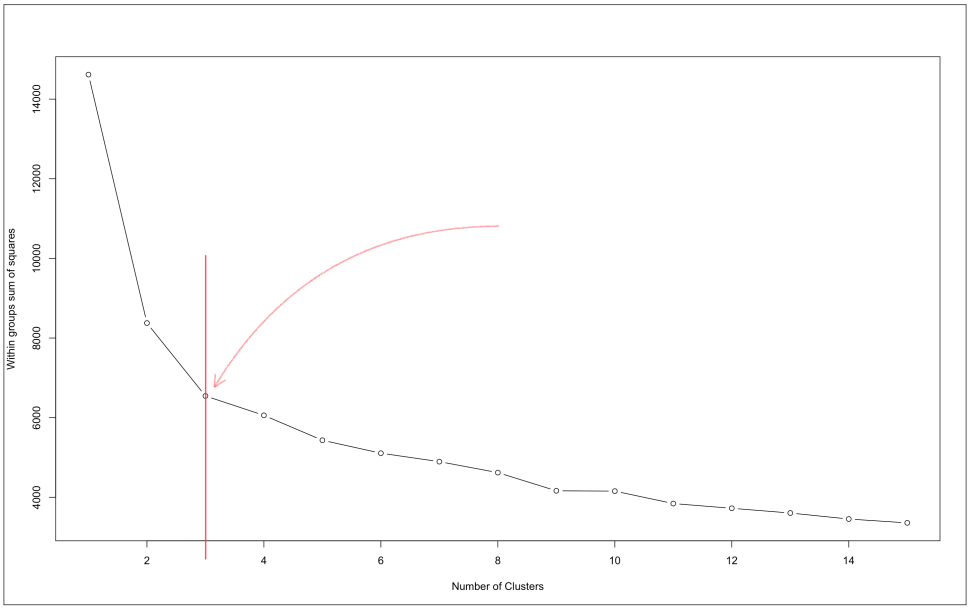


Figure 2: Arrow is pointing to the knee of the the curve

The Method clearly proposes 3 as the best number of clusters using 15 WSS (within-cluster sum of square).

NbClust Function

An an automatic method which is part of the NbClust library. It provides 30 indices to determine the best number of clusters in a dataset allowing a user to perform analysis with different distance measures.

Euclidean distance

Perform NbClust function using euclidian distance with min 2 and max 10 number of clusters.

```
clusterNo=NbClust(vehicles.scaled,distance="euclidean",min.nc=2,max.nc=10,method="kmeans",index="all")
```

```
*******************************************************************
* Among all indices:
* 9 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 7 as the best number of clusters
* 2 proposed 9 as the best number of clusters

                   ***** Conclusion *****

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


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

Figure 3: NbClust using Euclidian distance proposing 3 as the best number of clusters

Manhattan Distance

Perform NbClust function using city block (“Manhattan”) distance with min 2 and max 10 number of clusters.

```
clusterNo=NbClust(vehicles.scaled,distance="manhattan",min.nc=2,max.nc=10,method="kmeans",index="all")
```

```
*******************************************************************
* Among all indices:
* 10 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 7 as the best number of clusters
* 1 proposed 9 as the best number of clusters

                  ***** Conclusion *****

* According to the majority rule, the best number of clusters is  2


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

Figure 4: NbClust using Manhattan distance proposing 2 as the best number of clusters

Despite the algorithm concludes 2 as the best number of indices, among all indices 2 or ten is proposed the same number of times. Considering two previous outcomes, 3 still seems to be the best number of clusters.

Maximum Distance

Perform NbClust function using maximum distance with min 2 and max 10 number of clusters.

```
clusterNo=NbClust(vehicles.scaled,distance="maximum",min.nc=2,max.nc=10,method="kmeans",index="all")
```

```
*******************************************************************
* Among all indices:
* 8 proposed 2 as the best number of clusters
* 11 proposed 3 as the best number of clusters
* 2 proposed 7 as the best number of clusters
* 2 proposed 9 as the best number of clusters

                  ***** Conclusion *****

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


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

Figure 5: NbClust using Maximum distance proposing 3 as the best number of clusters

Performing k-Means

From all proposed results is clear, that 2 and 3 are the most likely the correct number clusters. (Despite we know our dataset contains 4 classes). Let's make a few test to prove this estimation.

Maximum Distance k=2

```
fit.km <- Kmeans(vehicles.scaled, 2, iter.max = 30, nstart = 10,method = "maximum")
```

Maximum distance k=2

	1	2
Bus	77	131
Opel	32	176
Saab	38	170
Van	64	125

This clustering isn't valid as the majority of items has been assigned to the cluster 2. No further investigation needed.

Euclidean distance and Manhattan Distance k=2

```
fit.km <- kmeans(vehicles.scaled, 2, nstart=20)
```

```
fit.km <- Kmeans(vehicles.scaled, 2, iter.max = 30, nstart = 10,method = “manhattan")
```

	1	3		Recall		%
Bus	155	53		155/(155+53)	0.745192307692308	75
Opel	92	116		116/(116+92)	0.557692307692308	56
Saab	99	109		109/(109+99)	0.524038461538462	53
Van	189	0		1	1	100
		Precision				
			C1	(155+189)/ (155+189+99+92)	0.642990654205607	64
			C2	(116+109)/ (116+109+53)	0.809352517985612	81
		Accuracy		(155+189+116+109) /All	0.699876998769988	**70**
Macro-Averaging		**Recall**		284		
			/	4	71	71
		Precision		145		
			/	2	72.5	73
Macro F1				10366		
			/	144	71.9861111111111	**72**

Both algorithms are proposing the same result.

Buses are dominant in the cluster one, Opel cars in the cluster 2, Saab in the cluster 2 and Vans in the cluster 1. Hence we can assume that cluster 1 contains large vehicles. Cluster 2 contains predominantly automobiles.

Accuracy 70% and Macro - F1 Score 72%

Maximum Distance k=3

```
fit.km <- Kmeans(vehicles.scaled, 3, iter.max = 30, nstart = 10,method = "maximum")
```

	1	2	3		Recall	%
Bus	102	28	78		0.865384615384615	87
Opel	63	110	35		0.528846153846154	53
Saab	68	104	36		0.5	50
Van	83	47	59		0.751322751322751	75
		Precision				
			C1	(102+83)÷(102+83+63+68)	0.585443037974684	58
			C2	(116+109)÷(116+109+53)	0.809352517985612	81
			C3	(78+59)÷(78+59+35+36)	0.658653846153846	66
		Accuracy		(102+83+110+104+78+59) ÷ all	0.659286592865929	**66**
Macro-Averaging		**Recall**		265		
			/	4	66.25	66
		Precision		205		
			/	3	68.3333333333333	68
Macro F1				8976		
			/	134	66.9850746268657	**67**

From an observation we can determine that cluster 2 contains small automobiles i.e. Opel and Saab. Clusters 1 and 2 are both containing Buses and Vans and do not clearly distinguish between them

Accuracy 68% and Macro - F1 Score 67%

Euclidean distance k=3

```
fit.km <- kmeans(vehicles.scaled, 2, nstart=20)
```

	1	2	3		Recall	%
Bus	80	46	82	(80+82)÷(80+82+46)	0.778846153846154	78
Opel	35	110	63	(110)÷(110+63+35)	0.528846153846154	53
Saab	38	97	73	97÷(97+73+38)	0.466346153846154	47
Van	80	0	109		100	100
		Precision				
			C1	160÷(160+35+38)	0.686695278969957	69
			C2	(110+97)÷(110+97+46)	0.818181818181818	82
			C3	(82+109)÷(109+82+63+73)	0.584097859327217	58
		Accuracy		(102+83+110+104+78+59) ÷ all	0.686346863468635	**69**
Macro-Averaging		**Recall**		278		
			/	4	69.5	70
		Precision		209		
			/	3	69.6666666666667	68
Macro F1				9520		
			/	138	68.9855072463768	**69**

From an observation we can determine that cluster 2 contains small automobiles i.e. Opel and Saab. Clusters 1 and 2 are both containing Buses and Vans and do not clearly distinguish between them

Accuracy 69% and Macro - F1 Score 69%

Manhattan distance k=3

```
fit.km <- Kmeans(vehicles.scaled, 3, iter.max = 30, nstart = 10,method = “manhattan")
```

	1	2	3		Recall	%
Bus	89	44	75	(89+75)÷(89+75+44)	0.788461538461538	79
Opel	45	109	54	(109)÷(109+45+54)	0.524038461538462	52
Saab	44	97	67	97÷(97+67+44)	0.466346153846154	47
Van	102	0	87		100	100
		Precision				
			C1	(89+102)÷(89+102+45+44)	0.682142857142857	69
			C2	(110+97)÷(110+97+44)	0.824701195219124	82
			C3	(75+87)÷(75+87+54+67)	0.57243816254417	57
		Accuracy		(89+102+109+97+75+87)÷ all	0.687576875768758	**69**
Macro-Averaging		**Recall**		278		
			/	4	69.5	70
		Precision		208		
			/	3	69.3333333333333	69
Macro F1 score				9660		
			/	139	69.4964028776978	**69**

From an observation we can determine that cluster 2 contains small automobiles i.e. Opel and Saab. Clusters 1 and 2 are both containing Buses and Vans and do not clearly distinguish between them.

Accuracy 69% and Macro - F1 Score 69%

Best 2 clusters

Accuracy and Macro-F1 Score determine as the best cluster k=2 using both Euclidean or Manhattan distance. For this algorithm Accuracy was 70% and Macro-F1 Score 72%.

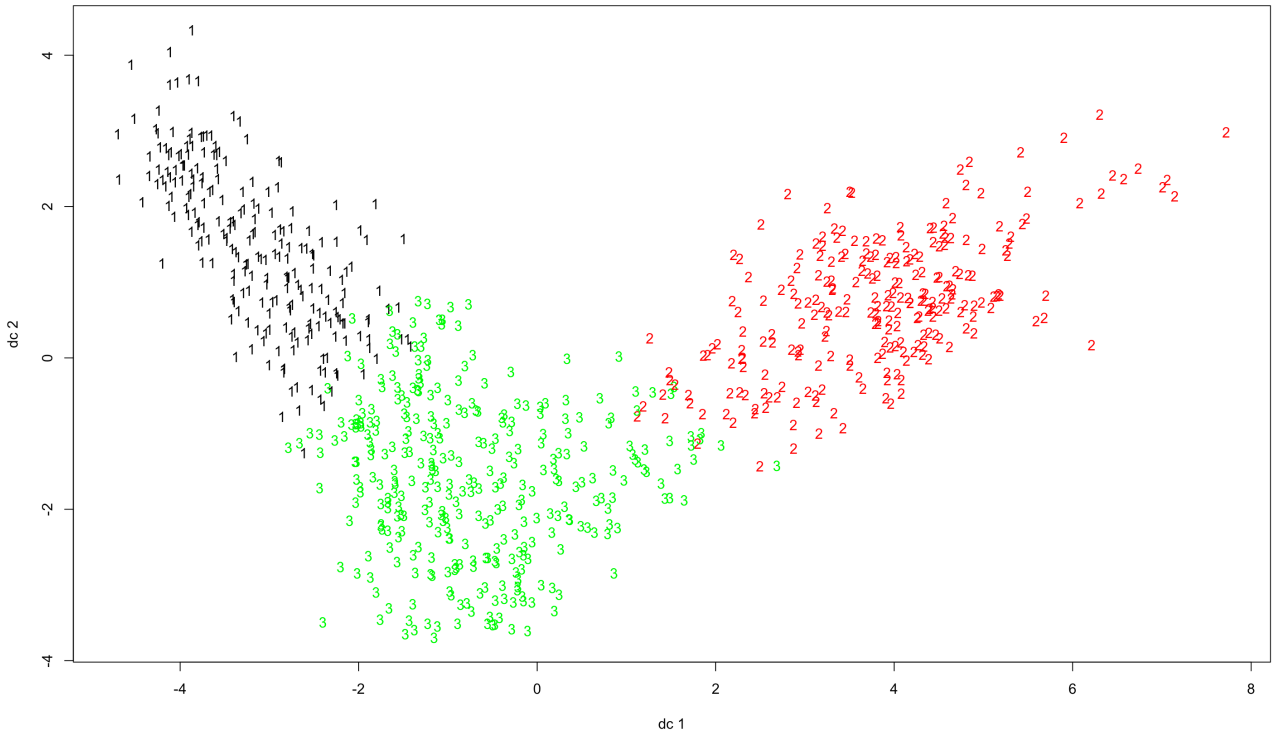
Despite Manhattan k=3 and Euclidian k=3 have the same Accuracy (69%) and even Macro-F1 Score (69%), algorithm using Euclidean distance is the winner here, as it has better Precision for the cluster 3.

k-Means k=3 Euclidean distance: Means of each attribute

```
fit.km <- kmeans(vehicles.scaled, 3, nstart=20)
```

```
Cluster means:
        Comp       Circ     D.Circ       Rad.Ra Pr.Axis.Ra   Max.L.Ra    Scat.Ra
1 -0.9423859 -0.5474136 -0.9181953 -1.145655221 -0.7525338 -0.5350742 -0.7945601
2  1.1632638  1.1894078  1.2242562  1.053281740  0.2198741  0.7124052  1.3122570
3 -0.2285316 -0.5301921 -0.2929582  0.001398735  0.3660925 -0.1699273 -0.4491392
       Elong Pr.Axis.Rect Max.L.Rect Sc.Var.Maxis Sc.Var.maxis     Ra.Gyr Skew.Maxis
1  0.8899092   -0.7607372 -0.5059132   -0.8128739   -0.7977349 -0.4155658  0.9920330
2 -1.2248909    1.3170750  1.1105028    1.2659807    1.3242876  1.0962089 -0.0306539
3  0.3136041   -0.4769669 -0.4987139   -0.4002859   -0.4561852 -0.5520307 -0.6831445
   Skew.maxis  Kurt.maxis  Kurt.Maxis    Holl.Ra
1 -0.10149637 -0.26085537 -1.06280490 -1.1285562
2  0.13863734  0.27527837 -0.02367658  0.1594321
3 -0.03494371 -0.02711354  0.77560770  0.6807868
```

```
> fit.km$size
[1] 233 253 327
```

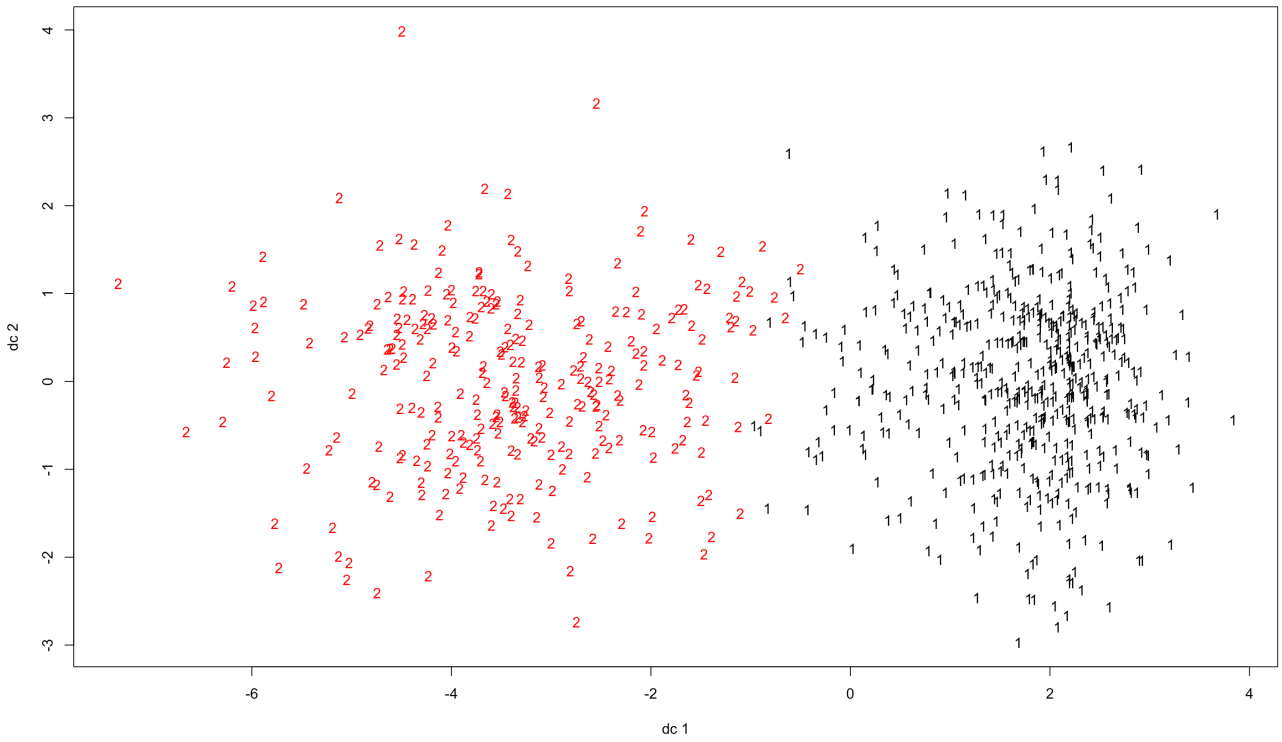


k-Means k=2 Manhattan distance: Means of each attribute

```
fit.km <- Kmeans(vehicles.scaled, 2, iter.max = 30, nstart = 10, method = “manhattan")
```

```
Cluster means:
        Comp       Circ    D.Circ     Rad.Ra Pr.Axis.Ra   Max.L.Ra   Scat.Ra     Elong
1 -0.5581484 -0.5746465 -0.598326 -0.5409894 -0.1354837 -0.3368540 -0.636507  0.606645
2  1.0741346  1.1058845  1.151455  1.0411126  0.2607329  0.6482622  1.224932 -1.167464
  Pr.Axis.Rect Max.L.Rect Sc.Var.Maxis Sc.Var.maxis     Ra.Gyr  Skew.Maxis  Skew.maxis
1   -0.6385458 -0.5309785   -0.6175342   -0.6400189 -0.5270382  0.04804773 -0.05799499
2    1.2288561  1.0218471    1.1884201    1.2316911  1.0142641 -0.09246595  0.11160907
  Kurt.maxis  Kurt.Maxis    Holl.Ra
1 -0.1198347 -0.02909619 -0.1073133
2  0.2306172  0.05599447  0.2065202
```

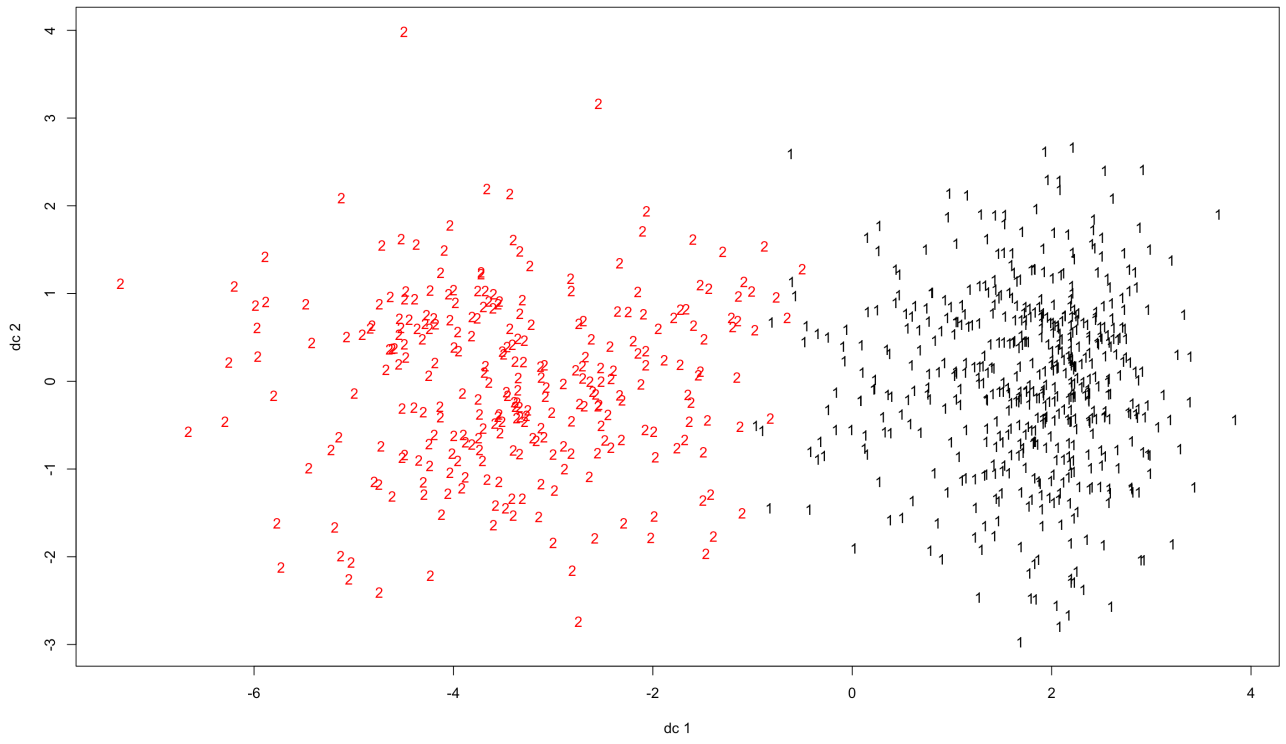
```
> fit.km$size
[1] 535 278
```


dc 2
dc 1
4
3
2
1
0
-1
-2
-3
-6
-4
-2
0
2
4

k-Means k=2 Euclidean distance: Means of each attribute

```
fit.km <- kmeans(vehicles.scaled, 2, nstart=20)
```

```
Cluster means:
        Comp       Circ    D.Circ     Rad.Ra Pr.Axis.Ra   Max.L.Ra   Scat.Ra     Elong
1 -0.5581484 -0.5746465 -0.598326 -0.5409894 -0.1354837 -0.3368540 -0.636507  0.606645
2  1.0741346  1.1058845  1.151455  1.0411126  0.2607329  0.6482622  1.224932 -1.167464
  Pr.Axis.Rect Max.L.Rect Sc.Var.Maxis Sc.Var.maxis     Ra.Gyr  Skew.Maxis  Skew.maxis
1   -0.6385458 -0.5309785   -0.6175342   -0.6400189 -0.5270382  0.04804773 -0.05799499
2    1.2288561  1.0218471    1.1884201    1.2316911  1.0142641 -0.09246595  0.11160907
  Kurt.maxis  Kurt.Maxis    Holl.Ra
1 -0.1198347 -0.02909619 -0.1073133
2  0.2306172  0.05599447  0.2065202
```



Objective 2: MLP

Input Configuration

Appropriate selection of inputs for time series forecasting models is important because it not only has the potential to improve performance of forecasting models, but also helps reducing cost in data collection.

If input subset is under-specified, the model performance becomes poor since the selected variables do not fully describe the behaviour of the modelled system [4].

On the other hand, when input subset is over-specified, the model will show a good performance on the training dataset, but very poor on the testing dataset. This phenomenon is called over-fitting data. [2] Overfitted model contains of more parameters than can be justified by data. It means the model remembers data instead of understanding the trend. (Similar as a person memorise something instead of understating it).

Tran [4] is proposing two approaches, model-based and model free. The model-free approach doesn't depend on the model structure and model calibration as the model-based. However, the model-free approach considers the the statistical relationship in the form of linear and non-linear correlation.

For the sake of this coursework we are going to use model-free approach when we use time series vector to create time series matrix. Each input represents one time step T, is current data, T-1 one step back. T+1 is the value we are seeking to predict.

Pre-processing

The first step to create a neural network for forecasting: $y_{t+1} = f(y_t, y_{t-1}, \ldots y_{t-m})$ we need to decide on *m* value and transform time series to the following matrix:

Tm2	Tm1	T	Tp1
0.89513	0.89966	0.89934	0.89963
0.89966	0.89934	0.89963	0.89771
0.89934	0.89963	0.89771	0.90022
0.89963	0.89771	0.90022	0.89754
0.89771	0.90022	0.89754	0.88977
0.90022	0.89754	0.88977	0.88718
0.89754	0.88977	0.88718	0.88536
0.88977	0.88718	0.88536	0.88135
0.88718	0.88536	0.88135	0.8717
0.88536	0.88135	0.8717	0.86539
0.88135	0.8717	0.86539	0.87012
0.8717	0.86539	0.87012	0.87802
0.86539	0.87012	0.87802	0.8713
0.87012	0.87802	0.8713	0.87147
0.87802	0.8713	0.87147	0.86721

Tp1 means T+1 and it is the value we desire to predict

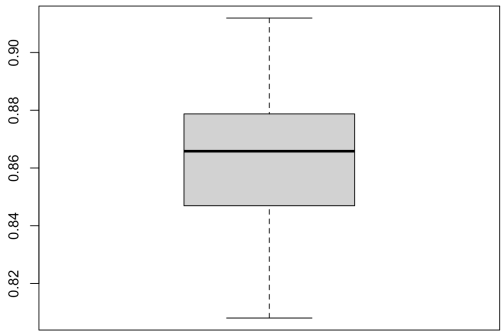
T is current time

Tm1 means T-1

Tm2 means T-2

<- this matrix is used for a neural network with 3 inputs.

Optionally, we can search for any outlier, however deleting them would destroy our time series. In case any outliers detected, we would have prices it further.



No outliers detected

Normalisation

Normalisation is a type of scaling that is used for normalising data that are not distributed in bell shape.

See more detail in part 1.

Normalised dataset dataT1 (in range 0:1) with two inputs:

Tm1	T	Tp1
Min. :0.0000	Min. :0.0000	Min. :0.0000
1st Qu.:0.3753	1st Qu.:0.3744	1st Qu.:0.3731
Median :0.5588	Median :0.5562	Median :0.5545
Mean :0.5316	Mean :0.5304	Mean :0.5293
3rd Qu.:0.6812	3rd Qu.:0.6805	3rd Qu.:0.6804
Max. :1.0000	Max. :1.0000	Max. :1.0000

While working on the Objective 2 a hypothesis was brought up! Because data are normally distributed between 0.8080 and 0.9119. normalisation is not necessary. Moreover, all input columns are 'equal' hence standardisation is also unnecessary. I decided to prove this hypothesis by making the same trials on both normalised and raw data!!

Building a neural network

```
dataT3.MODEL <- neuralnet(Tp1 ~ T + Tm1 + Tm2 + Tm3, hidden = c(3,4,2), data = dataT3.TRAIN)
```

Code above builds a prediction model with 3 time-series input vectors, it has 3 hidden layers with 3, 4 and 2 nodes.

Both tests (using normalised data and raw data) were performed on the same datasets using the same neural network configuration. 'Hidden layers' column indicates the number of hidden layers (number of items in the vector) and number of nodes (scalars in the vector).

NORMALISED INPUT DATA

Inputs	Hidden layers	Corelation	RMSE	MAPE	MAE	Indicies Average
2 (t,t-1)	c(3)	0.8716788	0.006595049	0.005879531	0.005085592	0.005853390666666
3 (t,t-1,t-2)	c(3)	0.8438359	0.0072788	0.006763991	0.005847953	0.006630248
4 (t,t-1,t-2,t-3)	c(3)	0.8073025	0.007384032	0.006816416	0.005895866	0.006698771333333
2 (t,t-1)	c(10)	0.8764246	0.00645214	0.005743571	0.004968493	0.005721401333333
3 (t,t-1,t-2)	c(10)	0.8436615	0.007285059	0.006760291	0.00584625	0.006630533333333
4 (t,t-1,t-2,t-3)	c(10)	0.8065492	0.007416531	0.006829304	0.005909102	0.006718312333333
2 (t,t-1)	c(4)	0.8768022	0.006464374	0.00579945	0.005015001	0.005759608333333
3 (t,t-1,t-2)	c(4)	0.8411789	0.007336999	0.006816409	0.005894274	0.006682560666666
4 (t,t-1,t-2,t-3)	c(4)	0.81119	0.007292238	0.006737781	0.005828151	0.00661939
2 (t,t-1)	c(2)	0.874627	0.006548764	0.005821241	0.005035564	0.005801856333333
3 (t,t-1,t-2)	c(2)	0.8466433	0.007248085	0.006727819	0.005816708	0.006597537333333
4 (t,t-1,t-2,t-3)	c(2)	0.8113524	0.007276553	0.006733174	0.00582311	0.006610945666666
2 (t,t-1)	c(2,5)	0.8707431	0.006630973	0.005907129	0.005109524	0.005882542
3 (t,t-1,t-2)	c(2,5)	0.8388645	0.007403794	0.006877786	0.005947193	0.006742924333333
4 (t,t-1,t-2,t-3)	c(2,5)	0.807624	0.007425512	0.006813195	0.005894911	0.006711206
2 (t,t-1)	c(4,8)	0.8766235	0.006498923	0.005775238	0.00499578	0.005756647
3 (t,t-1,t-2)	c(4,8)	0.8409522	0.007390513	0.006843003	0.005917183	0.006716899666666
4 (t,t-1,t-2,t-3)	c(4,8)	0.8092545	0.007374749	0.006778071	0.005863276	0.006672032
2 (t,t-1)	c(10,4)	0.8727355	0.006581249	0.005848518	0.005059385	0.005829717333333
3 (t,t-1,t-2)	c(10,4)	0.8466431	0.007204924	0.006668896	0.005765979	0.006546599666666
4 (t,t-1,t-2,t-3)	c(10,4)	0.8071538	0.007428504	0.006831154	0.00591005	0.006723236
2 (t,t-1)	c(3,4,2)	0.8702074	0.006687615	0.005928355	0.005128757	0.005914909
3 (t,t-1,t-2)	c(3,4,2)	0.8355339	0.007450173	0.006896932	0.005963172	0.006770092333333
4 (t,t-1,t-2,t-3)	c(3,4,2)	0.8051415	0.007459868	0.006889922	0.005961032	0.006770274
	Max Correlation =	0.8768022			MIN Average =	0.005721401333333
Correlation AVG =	0.841363470833333		Indices Average =	0.006390068111111	2nd MIN Average	0.005756647

Observing the comparison table of normalised data, we can see that the best result in terms of correlation (0.8768022) is a network with 2 inputs, 1 hidden layer together with 4 nodes (Green).

The best result in therms of error indices was proposed by network consisting of 2inputs, 1 hidden layer and 10 nodes (YELLOW).

Average correlation of all neural networks performing on normalised data was 0.8413635, which indicates very strong correlation between predicted and original data.

RAW INPUT DATA

Inputs	Hidden layers	Corelation	RMSE	MAPE	MAE	Indices Average
2 (t,t-1)	c(3)	0.8671676	0.006598793	0.005981211	0.005173361	0.005917788333333
3 (t,t-1,t-2)	c(3)	0.8957457	0.006071019	0.005690634	0.004917842	0.005559831666666
4 (t,t-1,t-2,t-3)	c(3)	0.8361255	0.006617417	0.006021052	0.005205434	0.005947967666666
2 (t,t-1)	c(10)	0.888928	0.006122491	0.005563423	0.004810042	0.005498652
3 (t,t-1,t-2)	c(10)	0.9207791	0.005424844	0.004886178	0.00423331	0.004848110666666
4 (t,t-1,t-2,t-3)	c(10)	0.5986715	0.01154477	0.01073484	0.009281606	0.010520405333333
2 (t,t-1)	c(4)	0.9368197	0.00495637	0.004498553	0.003888503	0.004447808666666
3 (t,t-1,t-2)	c(4)	0.9340525	0.005233645	0.004764575	0.00411766	0.004705293333333
4 (t,t-1,t-2,t-3)	c(4)	0.9318746	0.003466915	0.00324274	0.002801836	0.003170497
2 (t,t-1)	c(2)	0.9316779	0.004965788	0.004488146	0.003879565	0.004444499666666
3 (t,t-1,t-2)	c(2)	0.923943	0.005313272	0.004809026	0.004156332	0.004759543333333
4 (t,t-1,t-2,t-3)	c(2)	0.9248951	0.004129644	0.003861213	0.003335221	0.003775359333333
2 (t,t-1)	c(2,5)	0.9306387	0.004990293	0.004503656	0.003893018	0.004462322333333
3 (t,t-1,t-2)	c(2,5)	0.8654232	0.007332035	0.006803087	0.005874792	0.006669971333333
4 (t,t-1,t-2,t-3)	c(2,5)	-0.862096	0.01360622	0.01290439	0.01114243	0.012551013333333
2 (t,t-1)	c(4,8)	0.9398695	0.004838468	0.004357924	0.00376631	0.004320900666666
3 (t,t-1,t-2)	c(4,8)	0.8979961	0.006091181	0.005680932	0.004908424	0.005560179
4 (t,t-1,t-2,t-3)	c(4,8)	0.8905232	0.005157355	0.004790144	0.00414091	0.004696136333333
2 (t,t-1)	c(10,4)	0.9117582	0.005443927	0.004867736	0.004212321	0.004841328
3 (t,t-1,t-2)	c(10,4)	0.9130634	0.005479729	0.004963327	0.004292144	0.004911733333333
4 (t,t-1,t-2,t-3)	c(10,4)	0.886284	0.005008204	0.004468424	0.003865953	0.004447527
2 (t,t-1)	c(3,4,2)	0.9394719	0.01332432	0.01267575	0.01093891	0.012312993333333
3 (t,t-1,t-2)	c(3,4,2)	-0.9151387	0.01351543	0.01285223	0.01109176	0.012486473333333
4 (t,t-1,t-2,t-3)	c(3,4,2)	-0.8395015	0.01335771	0.01267369	0.01094338	0.012324926666666
Max Correlation =	0.9398695	Positive COR AVG	AVG without deviation	0.00551518325396825	MIN Average	0.003170497
Correlation AVG =	0.672873841666667	0.893605161904762	Indices Average =	0.00638255256944444	2n MIN Average	0.003775359333333

Observing the comparison table of neural networks using ‘raw’ data, we can see that the best result in terms of correlation (0.9398695) is a network with 2 inputs, 2 hidden layers together with 4 and 8 nodes (GREEN). It is much better result compare to normalised dataset. Moreover, it indicates even stronger average correlation! Average correlation of all neural networks performing on raw data was 0.672874. Although, It is much Lower correlation compare to neural networks trained on normalise dataset, further investigation relieves that some correlation indices are negative. (Negative correlation is a relationship between two variables in which an increase in one variable is associated with decrease in other.) If we filter out those out, the average correlation is 0.893605162. It is a better result compare to normalise dataset.

The best result in therms of error indices for NNs using raw dataset was proposed by network consisting of 4inputs, 1 hidden layer and 4 nodes (YELLOW).

CONCLUSION

Neural network performing on normalised data are showing best results on models with two inputs with one or two hidden layers.

Neural networks performing on raw dataset are showing best results on models with 4 inputs with one hidden layer.

In some cases NNs using unscaled dataset are performing better compare to the same configuration using normalised data. Nevertheless, in some cases the performance decreases rapidly and overall performance thus lower compare normalised data.

I am proposing to use carefully selected model performing on raw data!

Best results

For Normalised dataset

```
dataT1.MODEL <- neuralnet(Tp1 ~ T + Tm1, hidden = c(10), data = dataT1.TRAIN)
```

Correlation

```
        [,1]
[1,] 0.8764246
```

Original and predicted data (head)

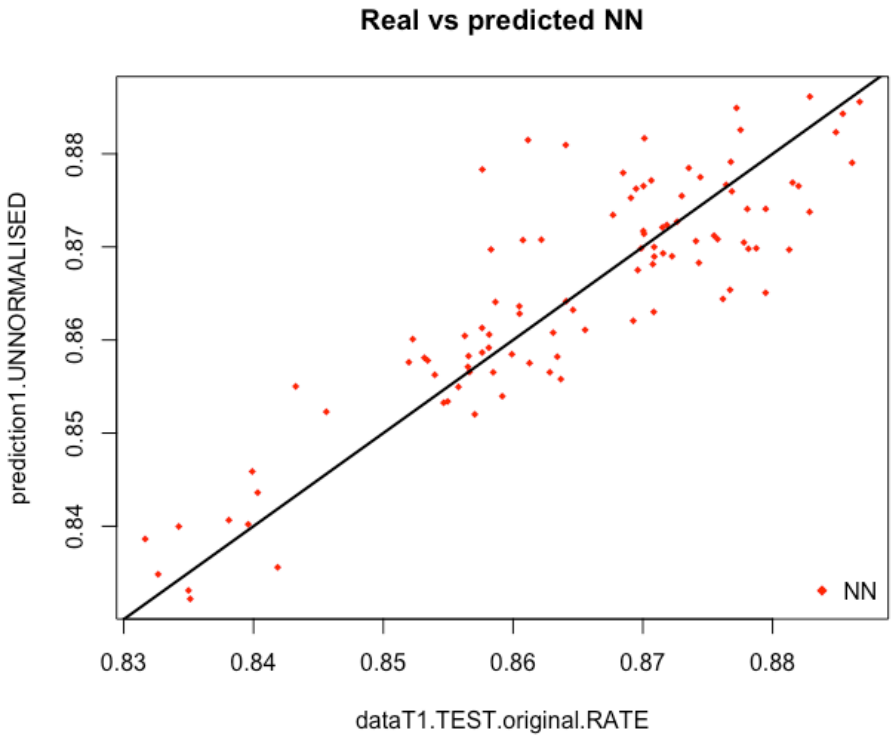
	dataT1.TEST.original.RATE	
400	0.87077	0.8681469
401	0.87946	0.8650753
402	0.87779	0.8704773
403	0.87678	0.8791369
404	0.87443	0.8774927
405	0.88200	0.8765404

Error Indices:

RMSE: 0.00645214

MAPE: 0.005743571

MAE: 0.004968493



For Raw dataset

```
dataT3.MODEL <- neuralnet(Tp1 ~ T + Tm1 + Tm2 + Tm3, hidden = c(4), data = dataT3.TRAIN)
```

Correlation

[,1]

[1,] 0.9318746

Desired and Predicted data (Head)

dataT3.TEST.original.RATE

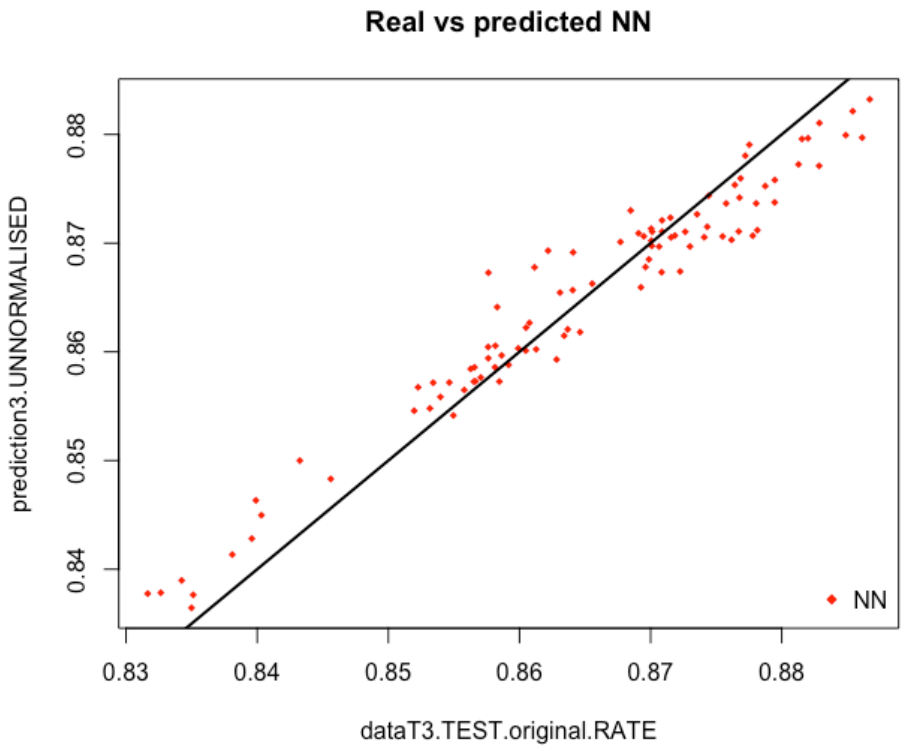
400	0.87946	0.8737495
401	0.87779	0.8706821
402	0.87678	0.8741836
403	0.87443	0.8743603
404	0.88200	0.8796480
405	0.87805	0.8736564

Error Indices:

RMSE: 0.003466915

MAPE: 0.00324274

MAE: 0.002801836



References:

[1] A. Géron, H*ands-On Machine Learning with Scikit-Learn, Keras, and TenserFlow,* 2nd ed. Newton, MA, USA: O'Reilly Media, Inc., 2019.

[2] D. Spiegelhalter, *The Art of Statistics: Learning from Data.* London, UK: Penguin Books, 2020.

[3] E. Alpaydin, *Machine Learning: The new AI*. Cambridge, MA, USA: MIT Press, 2016.

[4] H.D. Tran, N. Muttil, B.J.C Perera, "Selection of significant input variables for time series forecasting," *Environmental Modelling & Software,* 2015, Vol. 64, pp. 156-163. [Online] Available at: https://doi.org/10.1016/j.envsoft.2014.11.018 [Accessed Nov. 4, 2020]

Appendix A: Objective 1 (clustering code)

```
######## LOAD DATA #########

#Load Data
library(readxl)
vehicles <- read_excel("~/Desktop/BI/vehicles.xlsx")

#Create Working set
vehicles.raw <- vehicles[2:20]


#view Structure
str(vehicles.raw)
head(vehicles.raw)
summary(vehicles.raw)

classes.name <- vehicles$Class
set.seed(20)

######### Data Preparation ########

#Look for outliers
boxplot(vehicles.raw[1:18], plot=TRUE)

#Outlier detected at columns index: 4, 5, 6, 11, 12, 14, 15, 16

#Retrieve list of Outliers
col4 <- which(vehicles.raw$Rad.Ra %in% boxplot(vehicles.raw[4], plot=FALSE)$out)
col5 <- which(vehicles.raw$Pr.Axis.Ra %in% boxplot(vehicles.raw[5], plot=FALSE)$out)
col6 <- which(vehicles.raw$Max.L.Ra %in% boxplot(vehicles.raw[6], plot=FALSE)$out)
col11 <- which(vehicles.raw$Sc.Var.Maxis %in% boxplot(vehicles.raw[11], plot=FALSE)$out)
col12 <- which(vehicles.raw$Sc.Var.maxis %in% boxplot(vehicles.raw[12], plot=FALSE)$out)
col14 <- which(vehicles.raw$Skew.Maxis %in% boxplot(vehicles.raw[14], plot=FALSE)$out)
col15 <- which(vehicles.raw$Skew.maxis %in% boxplot(vehicles.raw[15], plot=FALSE)$out)
col16 <- which(vehicles.raw$Kurt.maxis %in% boxplot(vehicles.raw[16], plot=FALSE)$out)

#append all columns together
allColumns <- append(col4, c(col5,col6,col11,col12,col14,col15,col16))

#check for unique otliers
outliers <- unique(allColumns) # = 33 Unique Outliers

#delete outlires
vehicles.clear <- vehicles.raw[-outliers,]

#plot clear dataset
boxplot(vehicles.clear[1:18], plot=TRUE)

#copy Class names for evaluation
classes.name <- vehicles.clear$Class

#delete class column!!!!!!!!!!!!!!!! it is important to do this step after removing
#outliers, so the last colum will be the same length as the dataset
vehicles.clear <- vehicles.clear[1:18]


set.seed(20)
######## SCALE DATA #########
vehicles.scaled <- scale(vehicles.clear)
summary(vehicles.scaled)

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

```
vehicles.norm <- as.data.frame(lapply(vehicles.clear, normalise))

######### NB CLUST ##########
library("NbClust")

set.seed(26)
clusterNo=NbClust(vehicles.scaled,distance="euclidean", min.nc=2,max.nc=10,method="kmeans",index="all")
clusterNo=NbClust(vehicles.scaled,distance="manhattan", min.nc=2,max.nc=10,method="kmeans",index="all")
clusterNo=NbClust(vehicles.scaled,distance="maximum", min.nc=2,max.nc=10,method="kmeans",index="all")

table(clusterNo$Best.n[1,])



barplot(table(clusterNo$Best.n[1,]),
        xlab="Numer of Clusters",
        ylab="Number of Criteria",
        main="Number of Clusters Chosen by 30 Criteria")

set.seed(26)
wss <- 0
for (i in 1:15){
  wss[i] <- sum(kmeans(vehicles.scaled, centers=i)$withinss)
}

plot(1:15,
     wss,
     type="b",
     xlab="Number of Clusters",
     ylab="Within groups sum of squares")




################### APPLY K-MEANS ###################

set.seed(20)

set.seed(20)
fit.km <- kmeans(vehicles.scaled, 2, nstart=20)

library(amap)
set.seed(20)
#fit.km <- Kmeans(vehicles.scaled, 3, iter.max = 30, nstart = 10,
#       method = "maximum")

#fit.km <- Kmeans(vehicles.scaled, 2, iter.max = 30, nstart = 10,method = "manhattan")

fit.km

fit.km$centers
fit.km$size


library(fpc)
plotcluster(vehicles.scaled, fit.km$cluster)


#Evaluation
confuseTable.km <- table(classes.name, fit.km$cluster)
confuseTable.km

table(classes.name,fit.km$cluster)
table(fit.km$cluster,classes.name)
```

Appendix B: Objective 2 (NNs code)

NOTE: this code is processing normalised data in range 0:1, for processing raw data it needs to skip the normalisation/unnormalisation functions and make other minor adjustments!!!

```
library(ggplot2)
library(reshape2)
library(gridExtra)
library(MLmetrics)
library(neuralnet)
library(grid)
library(MASS)

#################### LOAD DATA ################
dataT1 <- read.csv("~/Desktop/BI/t-1.csv")
dataT2 <- read.csv("~/Desktop/BI/t-2.csv")
dataT3 <- read.csv("~/Desktop/BI/t-3.csv")

str(dataT1)
summary(dataT1)

str(dataT2)
summary(dataT2)

str(dataT3)
summary(dataT3)

#boxplot(dataT1$T, plot=TRUE) ## Detecting if any outliers


# creating functions
normalise <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
  #-1 + 2.*(data - min(data))/(max(data) - min(data))
}



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

rmse <- function(error)
{
  sqrt(mean(error^2))
}

#NORMAILISEDalise data
dataT1.NORMAILISED <- as.data.frame(lapply(dataT1, normalise))
dataT2.NORMAILISED <- as.data.frame(lapply(dataT2, normalise))
dataT3.NORMAILISED <- as.data.frame(lapply(dataT3, normalise))

summary(dataT1.NORMAILISED)
summary(dataT2.NORMAILISED)
summary(dataT3.NORMAILISED)

#Creating TRAINig datasets
dataT1.TRAIN <- dataT1.NORMAILISED[1:399, ]
dataT1.TEST <- dataT1.NORMAILISED[400:498, ]

dataT2.TRAIN <- dataT2.NORMAILISED[1:399, ]
dataT2.TEST <- dataT2.NORMAILISED[400:497, ]

dataT3.TRAIN <- dataT3.NORMAILISED[1:399, ]
```

```
dataT3.TEST <- dataT3.NORMAILISED[400:496, ]



######################    TRAINING    ##############################



set.seed(12345) # to guarantee repeatable results
######for t-1
dataT1.MODEL <- neuralnet(Tp1 ~ T + Tm1, hidden = c(10),  data = dataT1.TRAIN)
#plot(dataT1.MODEL)

set.seed(12345) # to guarantee repeatable results # to guarantee repeatable results
#####for t-2
dataT2.MODEL <- neuralnet(Tp1 ~ T + Tm1 + Tm2, hidden = c(10), data = dataT2.TRAIN)
#plot(dataT2.MODEL)

set.seed(12345) # to guarantee repeatable results # to guarantee repeatable results
####for t-3
dataT3.MODEL <- neuralnet(Tp1 ~ T + Tm1 + Tm2 + Tm3, hidden = c(10), data = dataT3.TRAIN)
#plot(dataT3.MODEL)




########################### EVALUATING PERFORMANCE ##############################

########## FOR T-1
#Number model result
MODEL1.results <- compute(dataT1.MODEL, dataT1.TEST[1:2])
#Obtaining predicted rate values
predicted.RATE1 <- MODEL1.results$net.result
cor(predicted.RATE1, dataT1.TEST$Tp1)
#head(predicted.RATE1)

#Retrieving original data
dataT1.TRAIN.original.RATE <- dataT1[1:399,3]  # the first 773 rows
dataT1.TEST.original.RATE <- dataT1[400:498,3] # the remaining rows

#Minimum and maximum
pred1.min <- min(dataT1.TRAIN.original.RATE)
pred1.max <- max(dataT1.TRAIN.original.RATE)
#head(dataT1.TRAIN.original.RATE)

#unuormalise values
prediction1.UNNORMALISED <- unnormalise(predicted.RATE1, pred1.min, pred1.max)
#prediction1.UNNORMALISED


final.result1 <- cbind(dataT1.TEST.original.RATE, prediction1.UNNORMALISED)
final.result1

#####Indexex

#MRSE INDEX
error <- (dataT1.TEST.original.RATE - prediction1.UNNORMALISED)
pred1_RMSE <- rmse(error)
pred1_RMSE

#MPE INDEX
pred1_MAPE <- MAPE(prediction1.UNNORMALISED, dataT1.TEST.original.RATE)
pred1_MAPE

#MAE INDEX
```

```
pred1_MAE <- mae(prediction1.UNNORMALISED, dataT1.TEST.original.RATE)
pred1_MAE

#PLOT
par(mfrow=c(1,1))
plot(dataT1.TEST.original.RATE, prediction1.UNNORMALISED ,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')



########## FOR T-2
#Number model result
MODEL2.results <- compute(dataT2.MODEL, dataT2.TEST[1:3])
#Obtaining predicted rate values
predicted.RATE2 <- MODEL2.results$net.result
cor(predicted.RATE2, dataT2.TEST$Tp1)
#head(predicted.RATE2)

#Retrieving original data
dataT2.TRAIN.original.RATE <- dataT2[1:399,4]  # the first 773 rows
dataT2.TEST.original.RATE <- dataT2[400:497,4] # the remaining rows

#Minimum and maximum
pred2.min <- min(dataT2.TRAIN.original.RATE)
pred2.max <- max(dataT2.TRAIN.original.RATE)
#head(dataT2.TRAIN.original.RATE)

#unuormalise values
prediction2.UNNORMALISED <- unnormalise(predicted.RATE2, pred2.min, pred2.max)
#prediction2.UNNORMALISED


final.result2 <- cbind(dataT2.TEST.original.RATE, prediction2.UNNORMALISED)
#final.result2

#####Indexex

#MRSE INDEX
error <- (dataT2.TEST.original.RATE - prediction2.UNNORMALISED)
pred2_RMSE <- rmse(error)
pred2_RMSE

#MPE INDEX
pred2_MAPE <- MAPE(prediction2.UNNORMALISED, dataT2.TEST.original.RATE)
pred2_MAPE

#MAE INDEX
pred2_MAE <- mae(prediction2.UNNORMALISED, dataT2.TEST.original.RATE)
pred2_MAE

#PLOT
par(mfrow=c(1,1))
plot(dataT2.TEST.original.RATE, prediction2.UNNORMALISED ,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')



########## FOR T-3
#Number model result
MODEL3.results <- compute(dataT3.MODEL, dataT3.TEST[1:4])
#Obtaining predicted rate values
```

```
predicted.RATE3 <- MODEL3.results$net.result
cor(predicted.RATE3, dataT3.TEST$Tp1)
#head(predicted.RATE3)

#Retrieving original data
dataT3.TRAIN.original.RATE <- dataT3[1:399,4]  # the first 773 rows
dataT3.TEST.original.RATE <- dataT3[400:496,4] # the remaining rows

#Minimum and maximum
pred3.min <- min(dataT3.TRAIN.original.RATE)
pred3.max <- max(dataT3.TRAIN.original.RATE)
#head(dataT3.TRAIN.original.RATE)

#unuormalise values
prediction3.UNNORMALISED <- unnormalise(predicted.RATE3, pred3.min, pred3.max)
#prediction3.UNNORMALISED


final.result3 <- cbind(dataT3.TEST.original.RATE, prediction3.UNNORMALISED)
#final.result3

#####Indexes

#MRSE INDEX
error <- (dataT3.TEST.original.RATE - prediction3.UNNORMALISED)
pred3_RMSE <- rmse(error)
pred3_RMSE

#MPE INDEX
pred3_MAPE <- MAPE(prediction3.UNNORMALISED, dataT3.TEST.original.RATE)
pred3_MAPE

#MAE INDEX
pred3_MAE <- mae(prediction3.UNNORMALISED, dataT3.TEST.original.RATE)
pred3_MAE

#PLOT
par(mfrow=c(1,1))
plot(dataT3.TEST.original.RATE, prediction3.UNNORMALISED ,col='red',main='Real vs predicted NN',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend='NN', pch=18,col='red', bty='n')


#OVERVIEW
pred1_RMSE
pred1_MAPE
pred1_MAE

pred2_RMSE
pred2_MAPE
pred2_MAE

pred3_RMSE
pred3_MAPE
pred3_MAE
```