Clustering Project: Partitioning and Hierarchical Methods

by A Smith

April 2020

Part 1: Partitioning Methods

1 Abstract

In this project, we discuss partitioning and hierarchical clustering techniques and how we can apply these techniques in the statistical programme *R*. We will discuss the advantages and disadvantages of each method of clustering and compare the similarities and differences between partitioning and hierarchical clustering methods.

2 Introduction: Partitioning Clustering

Clustering is a method of data mining which attempts to group objects based on similarities and patterns. When clustering, we wish to group objects in such a way that similar objects are closer together than dissimilar objects. Such a grouping can aid in statistical data analysis and help us identify an underlying pattern within our data. In the Dictionary of Statistical Terms, Kendall and Buckland define a cluster as 'a group of contiguous elements of a statistical population'[1]. Many other definitions exist, most of which identify objects within the same cluster as being 'similar' or 'alike' and distinct from objects in other clusters. Interestingly, there exists no universally accepted definition on what a cluster is, but the concepts of closeness and alikeness appear in all proposed definitions of what clustering is[2].

Cluster analysis may be performed by various different algorithms and methods, each of which use different metrics to identify clusters and measure distances between sets of points. Again, the analysis tends to focus on identifying small distances between members of the same cluster and denseness of certain areas of the data. We imagine cluster analysis as a multi-objective optimisation problem, where the solution is dependent on the data being analysed or the algorithm being used. We can identify two broad types of clustering: *Hard clustering*, in which each point belongs to a distinct cluster or not, and *soft clustering*, in which each data point has an associated closeness with each cluster.

Since no universally accepted definition of clustering exists, there are many different algorithms to solve clustering problems. As such, we often need to experiment with the dataset to find the optimal clustering algorithm and its associated parameters. The choice of clustering algorithm may depend on the structure of the data and the distance measure being used. This means that a clustering technique which works well for one dataset may perform badly on a dataset with a fundamentally different structure. In this project, we will identify some of the most common clustering techniques and examine how we can use R to perform these techniques on a dataset. We will also discuss the advantages and disadvantages of each clustering algorithm and explore the circumstances in which we would use each one.

The first part of this project will examine different methods of partitioning clustering, perform these techniques in R using a dataset and finally we will discuss our findings. The second part of this project will examine hierarchical clustering techniques and again apply these to a dataset in R. We will then examine similarities and differences between partitioning and hierarchical methods and the results they give.

3 Distance Measures

Considering that clustering is the process of grouping similar objects, and that the distance between objects is inversely proportional to their similarity, deciding which distance we use for measuring is an important step in finding clusters. Here, we briefly discuss different measurement methods and their advantages and disadvantages.

Euclidean ($L2$ norm): The Euclidean distance is probably the most used distance measure. For two vectors $\mathbf{x} = (x_1\,, x_2\,, \ldots\,, x_n)$ and $\mathbf{y} = (y_1\,, y_2\,, \ldots\,, y_n)$, we define the Euclidean distance as:

$$\begin{aligned} dist(\mathbf{x},\ \mathbf{y}) &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2} \\ &= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \end{aligned}$$

Euclidean distance is often undesirable for use in raw data as it is vulnerable to changes in scale. As such, data may need to be normalised before we take Euclidean measurements. One advantage of the Euclidean distance is that it is fairly easy to implement, but its main disadvantage is that it requires normalisation for meaningful results.

The analysis in this project will use Euclidean distance on scaled data as the primary distance measure, as this metric is effective when used in lower dimensional data. We will be calculating distances using the Euclidean metric unless otherwise stated.

Manhattan ($L1$ norm): The Manhattan distance is another commonly used distance measure. For two vectors $\mathbf{x} = (x_1\,, x_2\,, \ldots\,, x_n)$ and $\mathbf{y} = (y_1\,, y_2\,, \ldots\,, y_n)$, we define the Manhattan distance as:

$$\begin{aligned} dist(\mathbf{x}, \mathbf{y}) &= \| \mathbf{x} - \mathbf{y} \|_1 \\ &= \sum_{i=1}^{n} |x_i - y_i| \end{aligned}$$

This measure is also referred to as taxicab geometry and is inspired by the grid layout of the streets of Manhattan Island. The Manhattan distance can be defined as the sum of the horizontal and vertical components of two or more objects, or the absolute difference of their coordinates[3]. The Manhattan distance is often favoured in high dimensional data. This is because, in high dimensional space, data becomes more sparse and the L_k norm is sensitive to the largeness of k. Therefore, we often prefer to use the $L1$ norm instead of the $L2$ norm, that is, we often prefer to use the Manhattan distance in data mining for large dimensions. Again, the Manhattan distance is easy to calculate, but it requires data to be normalised.

Mahalanobis: The final distance measure we will discuss is the Mahalanobis distance. The Mahalanobis distance is a multivariate version of the Euclidean distance. This metric measures distances relative to centroids, this means that it is an effective method for discovering multivariate outliers. For two vectors $\mathbf{x} = (x_1\,, x_2\,, \ldots\,, x_n)$ and $\mathbf{y} = (y_1\,, y_2\,, \ldots\,, y_n)$, we define the Mahalanobis distance as:

$$Mahalanobis(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})\Sigma^{-1}(\mathbf{x} - \mathbf{y})^T$$

Where Σ^{-1} represents the inverse of the variance-covariance matrix. One main disadvantage of using the Mahalanobis distance is that it requires the inversion of the variance-covariance matrix and therefore is more computationally expensive than simpler distance measures.

4 Dataset: *Motor Trend Car Road Tests*

The dataset I chose for performing cluster analysis is Motor Trend Car Road Tests (*MTCars*), which provides data for 32 models of motorcars produced between 1973-1974[4]. The dataset gives observations on 11 variables concerning a car's performance, including *mpg (miles per US Gallon), cyl (number of cylinders), disp (displacement: a measure of engine size), hp (gross horsepower), weight (in 1000s of lbs)*, and $\frac{1}{4}$ *mile time*. The dataset is readily available in *R*, and can be summoned:

```
datasets::mtcars
```

For my analysis, I examined the first 7 variables of the $MTCars$ dataset, this is because these variables have a larger variability, compared with the 8^{th}, 9^{th}, 10^{th} and 11^{th} variables. The 8^{th} variable is a Bernoulli variable for *engine*

shape (0 = *V-shaped*, 1 = *straight*), the 9^{th} variable is a Bernoulli variable for *transmission* (0 = *automatic*, 1 = *manual*). The 10^{th} and 11^{th} variables are measurements of the *number of forward gears* and *carburettors*, respectively[4].

4.1 Rudimentary Analysis

To visualise our data, we produce a scatterplot comparing all variables to quickly examine correlations or relationships between them. We may also examine some variables in a larger plot using the *plot* function in *R*:

```
plot(cars,main="Scatterplot of MTCars Data")
```

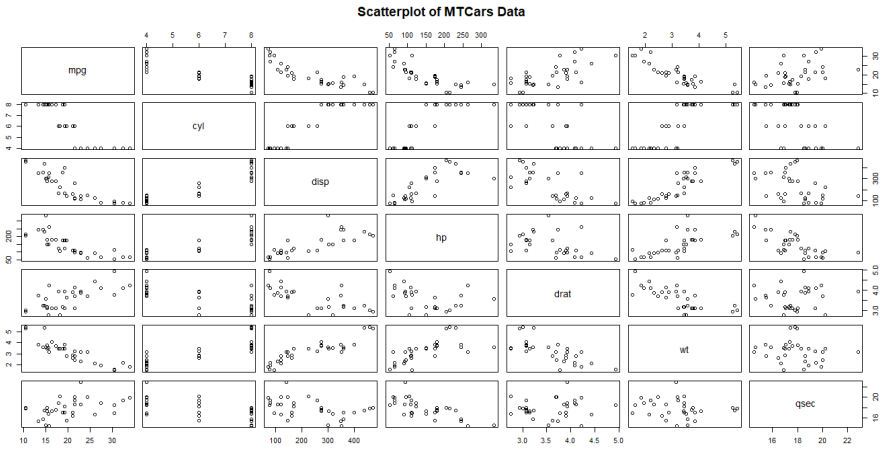


Figure 1: *A scatterplot of all variables within our dataset.*

We may examine a couple more plots using the *plot* function in *R*:

```
plot(cars[,3],cars[,1],xlab="Displacement (Cubic inches)",
ylab="MPG (Miles Per Gallon)",
     main="Plot of Engine Size vs Mileage")
```

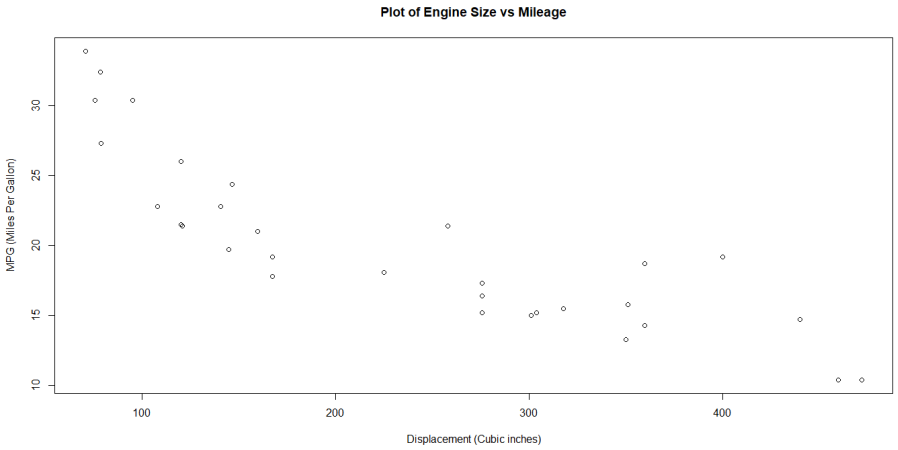


Figure 2: *Plot of Engine Size vs Mileage.*

```
plot(cars[,3],cars[,4],xlab="Displacement (Cubic inches)",
ylab="Horsepower (hp)",
     main="Plot of Engine Size vs Horsepower")
```

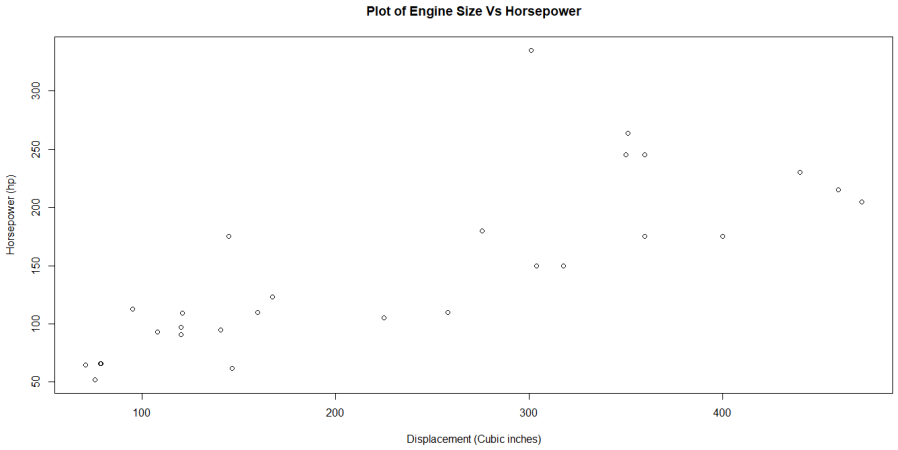


Figure 3: *Plot of Engine Size vs Horsepower.*

4.2 Initial analysis: Distance matrices

Since cluster analysis judges alikeness according to distance, it is useful to calculate and then visualise distances between points using *R*. Using our *MTCars* dataset, we calculate distances between clustered points. Before any distance calculation, I scaled our data so that variables with a larger variability would not dominate the data. This is important because the different variables are measured using different units.

```
#Ensure all the neccessary packages are loaded in R for this and
```

all future analysis.

```
library(tidyverse)
library(cluster)
library(factoextra)
library(stats)

ScaledCars<-scale(cars)
EuclideanDistance<-dist(ScaledCars,method="euclidean")
round(as.matrix(EuclideanDistance)[1:5,1:5],1)
```

	Mazda RX4	Mazda RX4 Wag	Datsun 710	Hornet 4 Drive	Hornet Sportabout
Mazda RX4	0.0	0.4	1.8	2.5	2.8
Mazda RX4 Wag	0.4	0.0	1.6	2.2	2.7
Datsun 710	1.8	1.6	0.0	2.4	3.9
Hornet 4 Drive	2.5	2.2	2.4	0.0	2.2
Hornet Sportabout	2.8	2.7	3.9	2.2	0.0

Figure 4: *Submatrix of the entire distance matrix returned by R.*

Throughout our analysis, we shall also examine the effects of our clustering on a subset on the *MTCars* dataset, this will often help to produce clearer plots to help us examine what is happening within the *R* script. The entire *MTCars* dataset contains 32 elements. For our subset, *carsreduced*, we take the first 15 observations:

```
carsreduced<-mtcars[1:15,1:7]
ScaledCars.reduced<-scale(carsreduced)
EuclideanDistance.subset<-dist(ScaledCars.reduced,method="euclidean")
```

4.3 Visualising distances

Using the *R* package *factoextra*, we can visualise the distances between points using a heat map. We do this using the *R* command *fviz_dist*, the output is shown below:

```
fviz_dist(EuclideanDistance,gradient = list(low = "black", mid
= "grey",high ="white"))
```

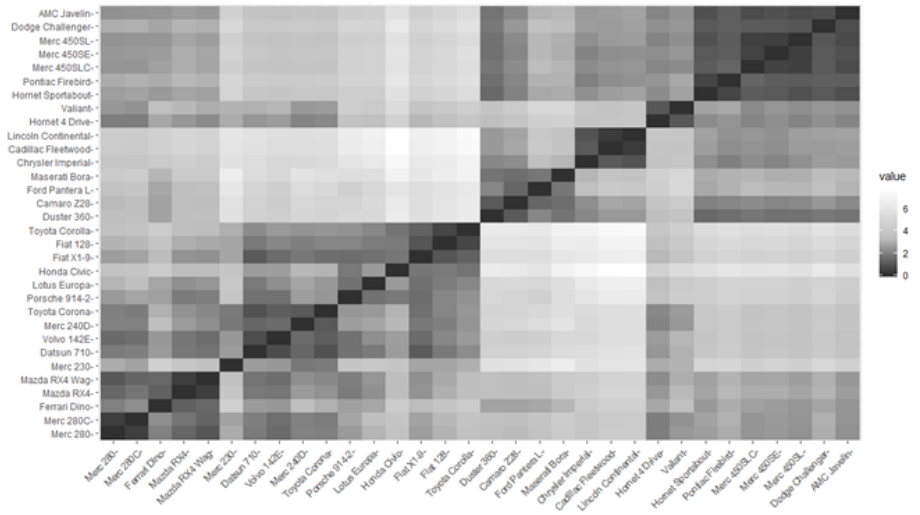


Figure 5: *Heatmap for distances between observations in our MTCars dataset returned by R.*

Here, the lighter colours correspond to a larger Euclidean distance between the points. Obviously, the black squares running diagonally through the heat map represent the distance from a point to itself with value 0. From this, we imagine which points will be close together when plotted. We also produce a heatmap for our subset, *carsreduced*:

```
fviz_dist(EuclideanDistance.subset,gradient = list(low = "black",
mid = "grey", high ="white"))
```

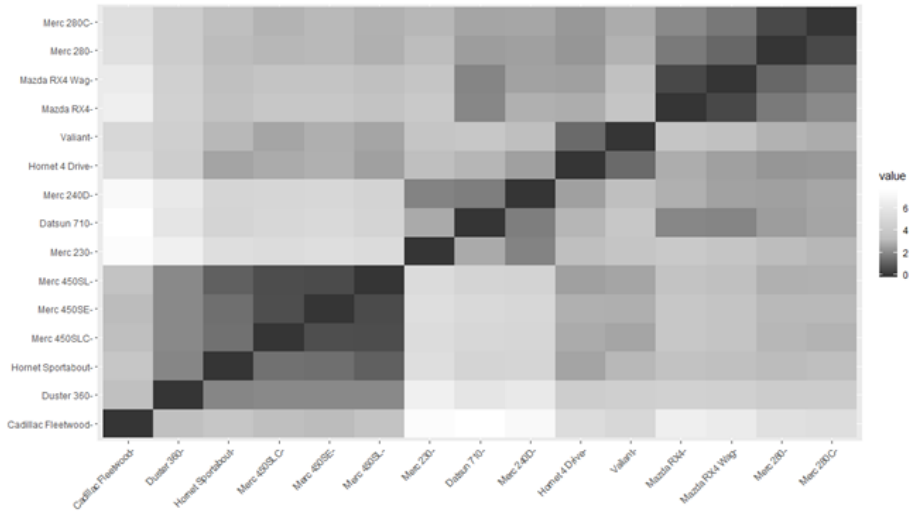


Figure 6: *Heatmap for distances between observations in our carsreduced dataset returned by R.*

Unsurprisingly, pairs of cars which we may imagine to have a similar performance (for example the *Mercedes 280* and *Mercedes 280C* as well as the *Mazda RX4* and *Mazda RX4 Wag*) have a small distance value. We imagine that these pairs of cars will be plotted near each other and will probably be part of the same cluster.

Similarly, pairs of observables which have very different performance statistics will have a much larger distance value. For example we may compare the big, large-engined, luxury *Cadillac Fleetwood* with the lightweight, smaller-engined *Datsun 710*. We imagine that these two cars will probably not be in the same cluster.

Figure 7: *1973-1974 Mercedes 280*[5].

Figure 8: *1973-1974 Mercedes 280 Coupe*[6].

Above shows the comparison of two models of the *Mercedes 280*. We imagine that these cars will be clustered together.

Figure 9: *1973-1974 Cadillac Fleetwood*[7].

Figure 10: *1973-1974 Datsun 710*[8].

Above shows comparison of the large luxury car *Cadillac Fleetwood* with the small family car *Datsun 710*. We imagine that these cars will not be clustered together.

4.4 Types Of Partitioning Clustering

4.4.1 *K*-means

We first explore *K-means* clustering, proposed by MacQueen in 1967[9]. *K*-means is the most commonly used method of partition clustering. We begin by selecting K points as *centroids*, we then assign each point to its closest centroid.

Once each point has been assigned to a cluster based on its proximity to its nearest centroid, we update each centroid until we converge to a solution.

We may therefore summarise K-means clustering as an algorithm:

Algorithm *K*-Means Clustering

1: Select K points as initial centroids.
2: **repeat**
3: Form K clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: **until** convergence criterion is met.

Figure 11: *Instructions for performing the K-means algorithm*[10].

To decide how many clusters would be suitable for our dataset, we use the R command *fviz_nbclust* from the $factoextra$ package to create a plot which calculates the *total within sum of squares* for each potential number of clusters. We wish to minimise the *total within sum of squares* while keeping the algorithm as computationally inexpensive as possible, therefore we examine where the plot bends sharply after an initial sharp decrease.

```
fviz_nbclust(cars,kmeans,method="wss")+geom_vline
(xintercept = 2, linetype = 2)
```

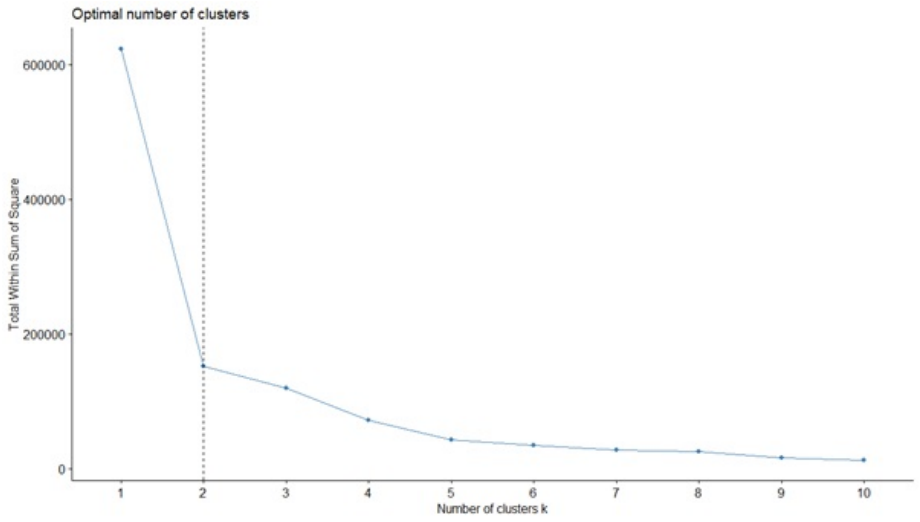


Figure 12: *Optimal number of clusters recommended by the K-means algorithm for the MTCars dataset.*

Creating this plot demonstrates that the optimal number of clusters for our K-means algorithm is 2. The clusters beyond cluster 2 do not have a huge effect on lowering the residual distances within our data.

We also aim to estimate the number of recommended clusters for our *carsreduced* dataset:

```
fviz_nbclust(carsreduced,kmeans,method="wss")+geom_vline
(xintercept = 3, linetype = 2)
```

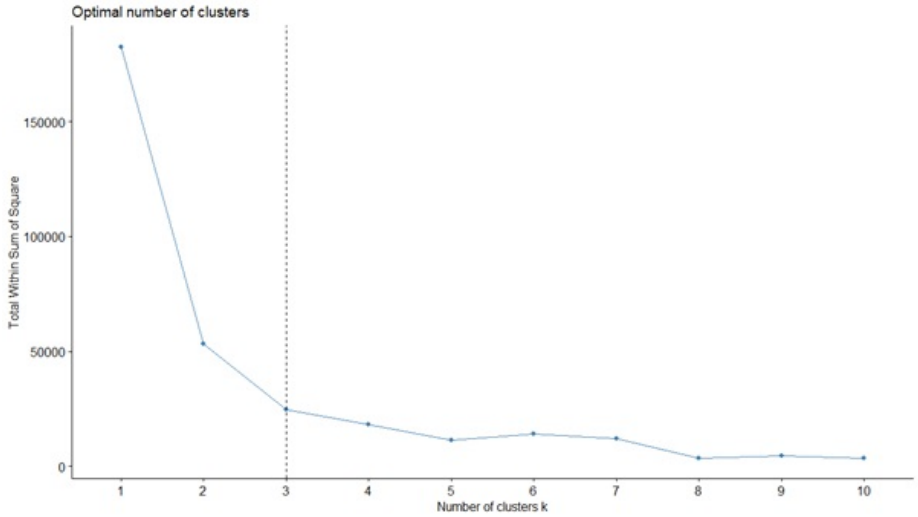


Figure 13: *Optimal number of clusters recommended by the K-means algorithm for carsreduced dataset.*

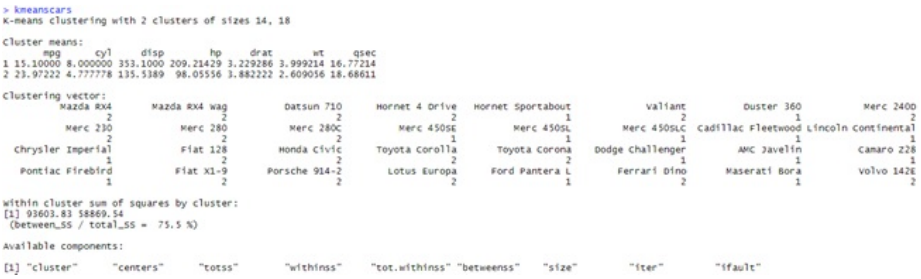
The gradient of this plot is strongly decreasing until we reach K=3 clusters, then the plot levels off, suggesting that we should choose K=3 for the optimal number of clusters for our *carsreduced* dataset.

Now we have an estimate for the optimal number of clusters we should choose, we can implement the K-means clustering algorithm. For simplicity, we can use the R command *kmeans*, which is available in the *stats* package, to perform the above algorithm on a dataset. We may specify the number of centroids in this function and therefore the number of clusters that will be plotted.

```
kmeanscars<-kmeans(cars,2)
kmeanscars.reduced<-kmeans(carsreduced,3)
```

The output of this function gives us the cluster means for each of our variables and a vector which displays which cluster we should assign each observable to. The command also informs us of the within cluster sum of squares for each cluster.

```
kmeanscars
```

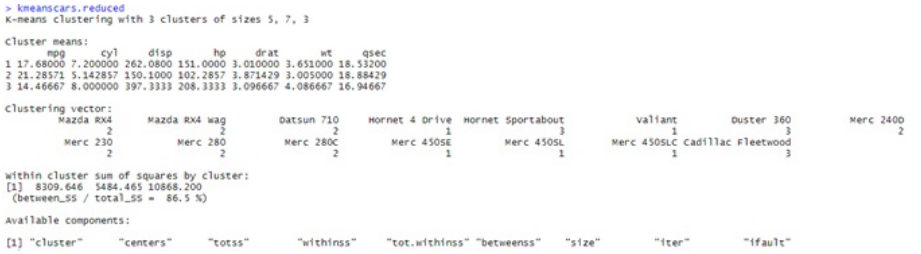


```
> kmeanscars
K-means clustering with 2 clusters of sizes 14, 18

Cluster means:
      mpg      cyl     disp       hp     drat       wt     qsec
1 15.10000 8.000000 353.1000 209.21429 3.229286 3.999214 16.77214
2 23.97222 4.777778 135.5389  98.05556 3.882222 2.609056 18.68611

Clustering vector:
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet Sportabout             Valiant          Duster 360           Merc 240D
                  2                   2                   2                   2                   1                   2                   1                   2
           Merc 230            Merc 280            Merc 280C           Merc 450SE          Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental
                  2                   2                   2                   1                   1                   1                   1                   1
  Chrysler Imperial             Fiat 128          Honda Civic       Toyota Corolla       Toyota Corona      Dodge Challenger           AMC Javelin            Camaro Z28
                  1                   2                   2                   2                   2                   1                   1                   1
   Pontiac Firebird            Fiat X1-9       Porsche 914-2          Lotus Europa      Ford Pantera L         Ferrari Dino        Maserati Bora           Volvo 142E
                  1                   2                   2                   2                   1                   2                   1                   2

Within cluster sum of squares by cluster:
[1] 93603.83 58869.54
 (between_SS / total_SS =  75.5 %)

Available components:

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

Figure 14: *Output for K-means algorithm on MTCars.*

And the output for the *carsreduced* dataset:

```
kmeanscars.reduced
```



```
> kmeanscars.reduced
K-means clustering with 3 clusters of sizes 5, 7, 3

Cluster means:
      mpg      cyl     disp       hp     drat       wt     qsec
1 17.68000 7.200000 262.0800 151.0000 3.010000 3.651000 18.53200
2 21.28571 5.142857 150.1000 102.2857 3.871429 3.005000 18.88429
3 14.46667 8.000000 397.3333 208.3333 3.096667 4.086667 16.94667

Clustering vector:
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet Sportabout             Valiant          Duster 360           Merc 240D
                  2                   2                   2                   1                   3                   1                   3                   2
           Merc 230            Merc 280            Merc 280C           Merc 450SE          Merc 450SL         Merc 450SLC  Cadillac Fleetwood
                  2                   2                   2                   1                   1                   1                   3

Within cluster sum of squares by cluster:
[1]  8309.646  5484.465 10868.200
 (between_SS / total_SS =  86.5 %)

Available components:

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

Figure 15: *Output for K-means algorithm on carsreduced.*

When plotting the results of the K-means clustering algorithm, we use the *fviz_cluster* function in *factoextra*. Since we cannot plot in 7 dimensions, our function uses *principal component analysis* (PCA) to amalgamate the variance from all 7 variables into 2 variables for effective plotting:

```
fviz_cluster(kmeanscars, data = cars,
             ellipse.type = "euclid",
             star.plot = TRUE,
             repel = TRUE,
             ggtheme = theme_minimal())
```

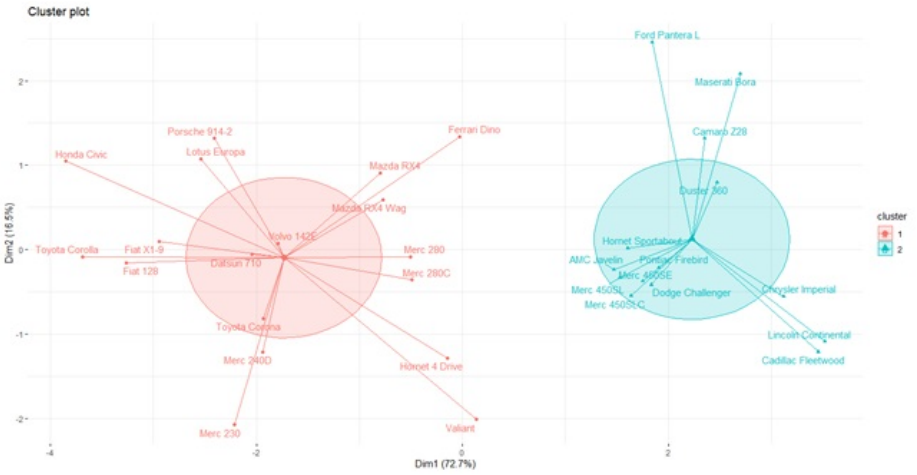


Figure 16: *Results of the K-means cluster method on our $MTCars$ dataset.*

The above plot shows the K-means clustering applied to the $MTCars$ dataset. As we can observe, the K-means algorithm has effectively partitioned the set of all observables into two distinct groups based on principal component analysis. As previously predicted, we see that cars of the same model, such as the *Mazda RX4* and *RX4 Wag* as well as the *Mercedes 450SL* and *Mercedes 450SLC* have been plotted very close together and are contained within the same cluster. Pairs of cars that we identified as having very different characteristics are in different clusters. From this analysis, we may begin to draw conclusions about these cars and their classification. Interestingly, cluster 1 is mainly populated by cars from European and Japanese manufacturers whereas cluster 2 is mainly populated by cars from US-based manufacturers. This may imply that American cars tend to be generally dissimilar from European and Japanese cars.

We shall now plot the results of our K-means clustering for the subset of $MTCars$, $carsreduced$:

```
fviz_cluster(kmeanscars.reduced, data = carsreduced,
         ellipse.type = "euclid",
         star.plot = TRUE,
         repel = TRUE,
         ggtheme = theme_minimal())
```

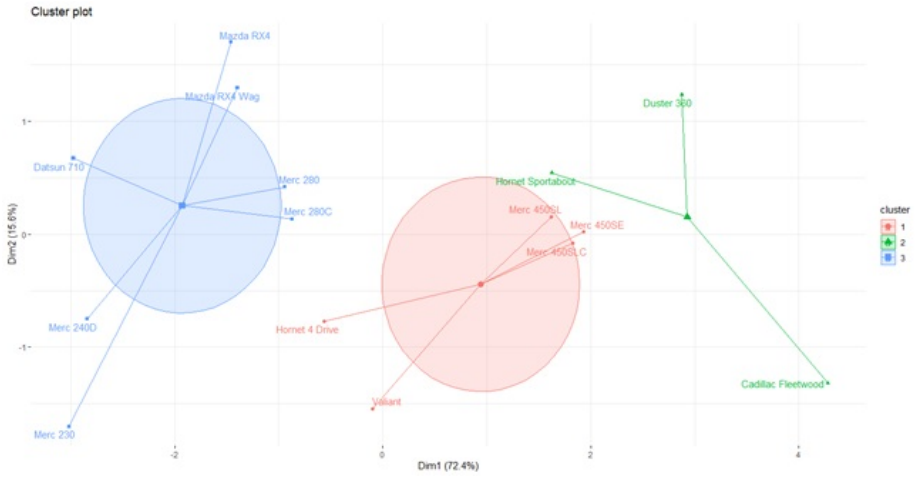


Figure 17: *Results of the K-means cluster method on our carsreduced dataset. We see that the third cluster does not contain enough observations for an ellipse to be created in R.*

Here we discuss the main benefits and limitations of the K-means algorithm method for clustering. The main benefit of K-means clustering is that the algorithm is fast and simple to perform, even when confronted with large datasets. It also adapts effectively to new examples and is effective in creating clusters of different shapes and sizes[11].

However, there are a few disadvantages of K-means clustering. In order to use K-means effectively, one must choose the appropriate number of clusters to be created, therefore we assume that we have knowledge of the data before performing the clustering. Also, since the algorithm is affected by the initial random selection of K centroids, the algorithm may give different results and therefore may be considered inconsistent. Lastly, since the algorithm involves the calculation of means, it is sensitive to any outliers or extreme values which may distort the locations of the centroids. We will now examine a method which is less sensitive to extreme values, Partitioning Around Medoids (PAM).

4.4.2 Partitioning Around Medoids (PAM)

In this next section, we will consider K-medoids algorithms and focus on one particular modification, the PAM method. The K-medoids methods are similar to K-means methods but we choose actual observed data points as initial centroids, called *medoids*.

We may summarise K-medoids clustering as an algorithm:

Algorithm 14 K-Medoids Clustering

1: Select K points as the initial representative objects.
2: **repeat**
3: Assign each point to the cluster with the nearest representative object.
4: Randomly select a nonrepresentative object x_i.
5: Compute the total cost S of swapping the representative object m with x_i.
6: If $S < 0$, then swap m with x_i to form the new set of K representative objects.
7: **until** Convergence criterion is met.

Figure 18: *Instructions for performing K-medoids algorithms*[12].

PAM is one of the algorithms based on K-medoids clustering available to us in R. R allows us to perform PAM as a clustering algorithm. The PAM algorithm uses the distance matrix of a dataset and minimises this by iteratively swapping all medoids and non-medoids iteratively until we reach convergence on a solution. The PAM algorithm was proposed by Kaufman and Rousseeuw (1990)[13].

Because we use medoids instead of means to select cluster centroids, the PAM algorithm is less affected by extreme values and outliers than the K-means algorithms. Therefore, if our dataset contains many extreme values, we may wish to implement the PAM algorithm over K-means.

As with the K-means method, we can use the calculation of *average silhouette width* to estimate the optimal number of clusters for our data. The silhouette width refers to the measure of how similar an object is to its own cluster, compared to other clusters. In this sense, we may imagine that the silhouette width is proportional to the effectiveness of our clustering. Again, we may use the $fviz\_nbclust$ function in R to determine the optimal number of clusters. Here we change the second argument in our function, corresponding to the partitioning function, to 'PAM':

```
  fviz_nbclust(cars, pam, method = "silhouette")+
theme_classic()
```

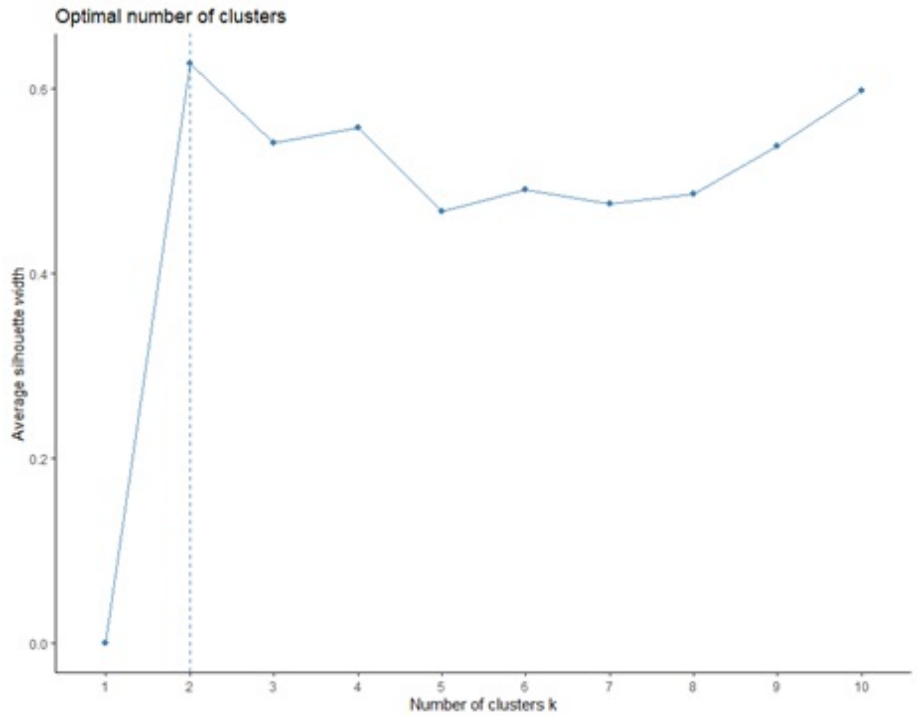


Figure 19: *Optimal number of clusters recommended by the PAM algorithm for the MTCars dataset. R recommends using K=2 clusters since this value optimises the average silhouette width.*

We examine the optimal number of clusters on our *carsreduced* dataset:

```
  fviz_nbclust(carsreduced, pam, method = "silhouette")+
theme_classic()
```

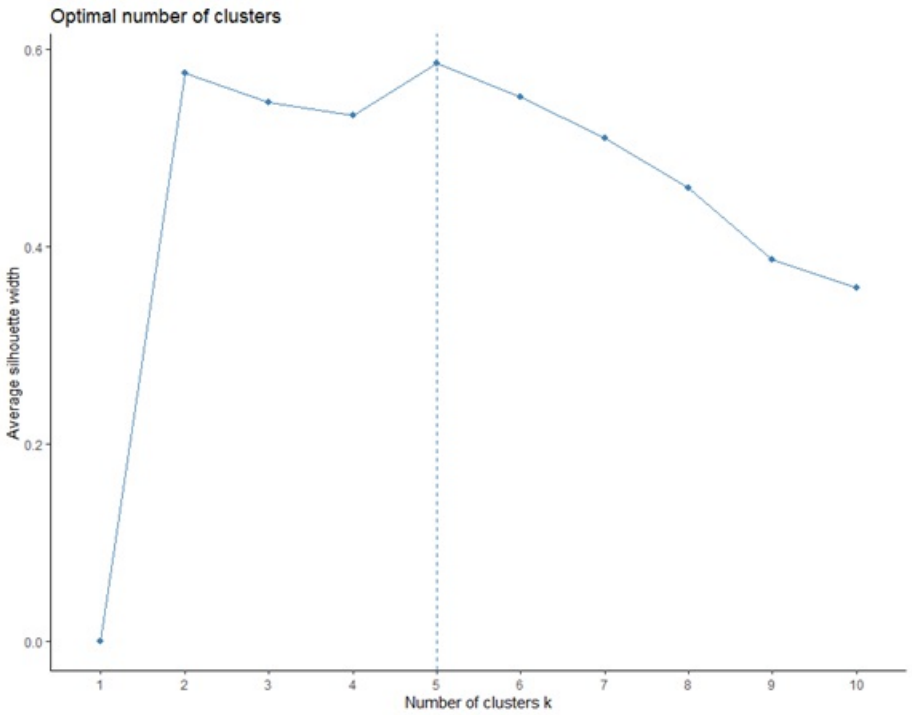


Figure 20: *Optimal number of clusters recommended by the PAM algorithm for the carsreduced dataset. R recommends using K=5 clusters since this value optimises the average silhouette width.*

As before, we use this estimate for the optimal number of clusters in the algorithm. We use the *pam* function which is available in the *stats* package to perform the PAM algorithm on our dataset. We will use our recommended number of clusters as an argument in the *pam* function.

```
pam.alg <- pam(cars, 2)
pam.alg
```

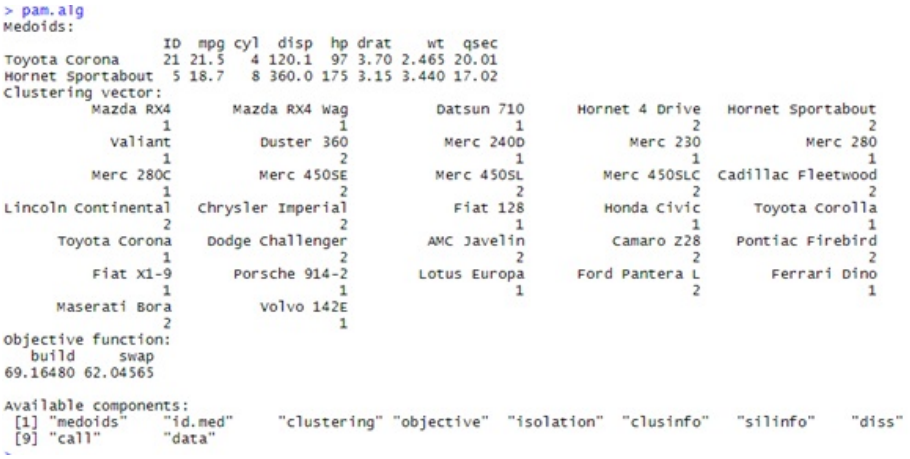


Figure 21: *Output for PAM algorithm on MTCars. Note that the medoids were selected as points in the dataset (Toyota Corona and Hornet Sportabout).*

We also perform the PAM algorithm on our *carsreduced* dataset, using the recommended 5 clusters:

```
pam.alg.reduced <- pam(carsreduced, 5)
pam.alg.reduced
```

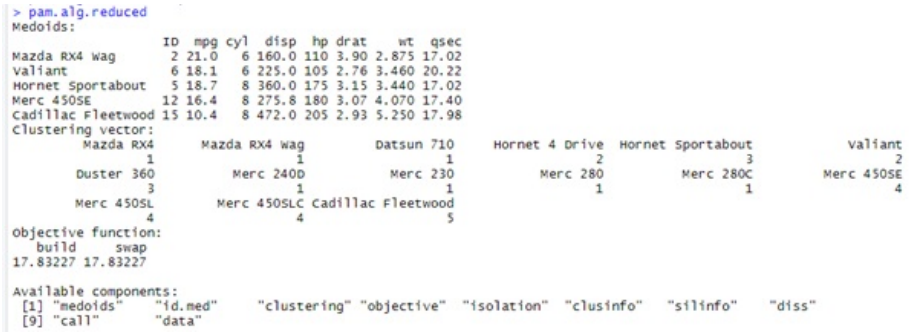


Figure 22: *Output for PAM algorithm on carsreduced. Here we see 5 cars as medoids: Mazda RX4 Wag, Valient, Hornet Sportabout, Merc 450SE and Cadillac Fleetwood. Interestingly, cluster 5 contains only 1 observation.*

Again, we may visualise the clusters from the PAM method. Since our data consists of several dimensions, our function again uses principal component analysis to amalgamate the variance from all 7 variables into 2 variables for effective plotting:

```
fviz_cluster(pam.alg,
             ellipse.type = "t", repel = TRUE,
             ggtheme = theme_classic(),main="Cluster plot (PAM)")
```

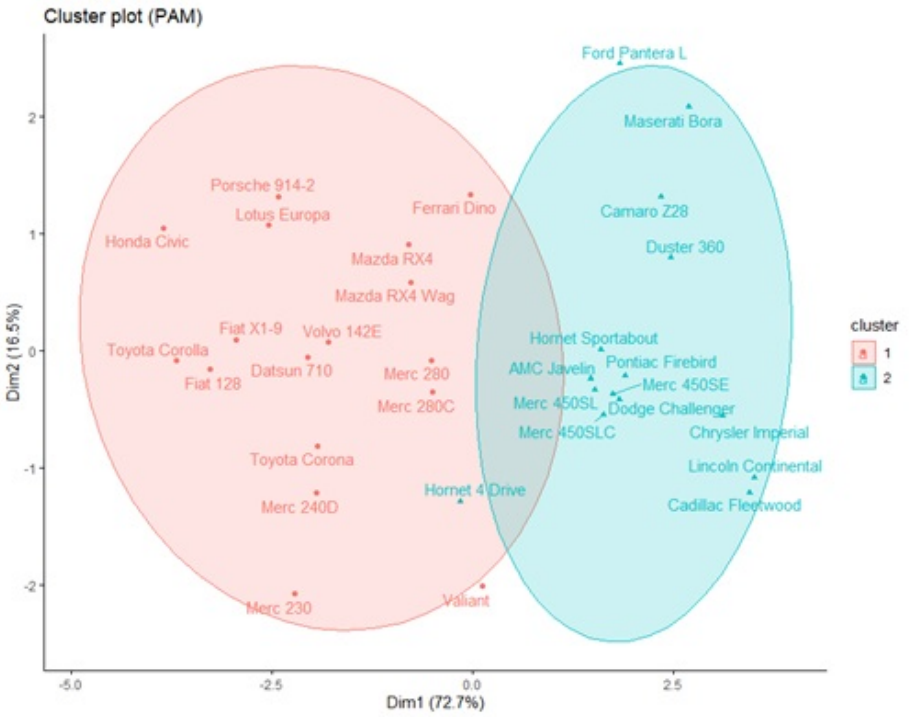


Figure 23: *Results of the PAM cluster method on our MTCars dataset.*

The above plot shows the output of the PAM algorithm. This output is extremely similar to the results from the K-means algorithm, although the observable *Hornet 4 Drive* has shifted from the first cluster in the K-means method to the second cluster in the PAM method. Also, *Valiant* is a member of the first cluster but falls outside of its ellipse. From both algorithms, we imagine that *Valiant* and *Hornet 4 Drive* represent extreme values within their clusters and these observations will be sensitive to the method of clustering chosen. We next examine the plot of PAM clustering on the *carsreduced* dataset, with K=5 clusters:

```
fviz_cluster(pam.alg.reduced,
             ellipse.type = "t", repel = TRUE,
             ggtheme = theme_classic(),main="Cluster plot (PAM)")
```

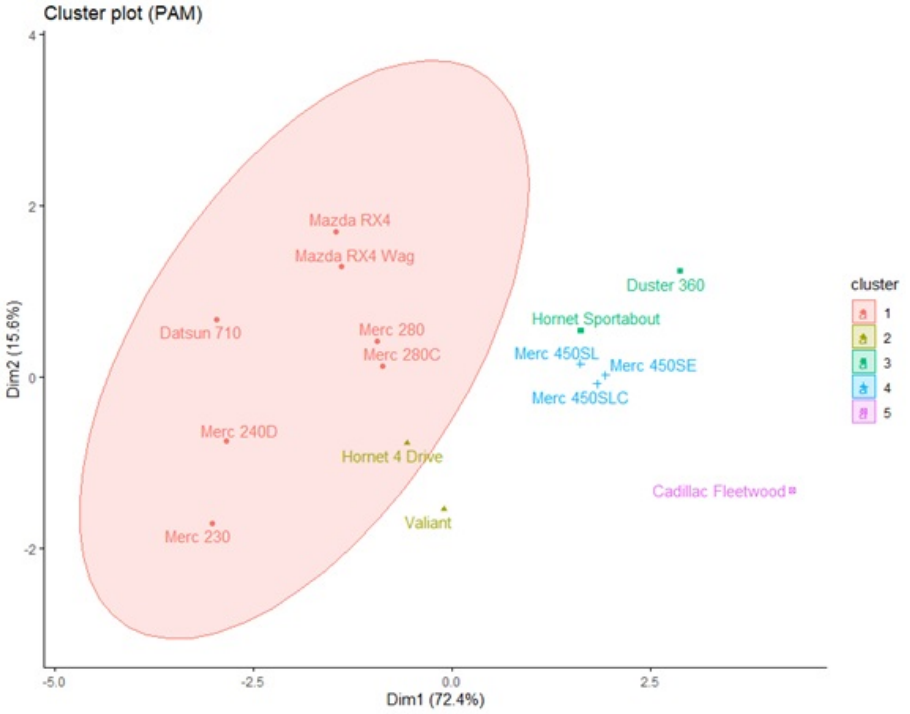


Figure 24: *Results of the PAM cluster method on our carsreduced dataset.*

Here we see one large cluster, with the other 4 only containing between 1-3 observations. Unsurprisingly, all 3 variations of the *Mercedes 450* series are in one cluster. This time, the *Hornet 4 Drive* and *Valiant* have been clustered together even though, again, the *Hornet 4 Drive* is enclosed in the ellipse of cluster 1. Similarly, the large and heavy *Cadillac Fleetwood* is in its own cluster.

We will now discuss the advantages and limitations of the PAM method of clustering. As we have already discussed, taking medoids as our initial K centres results in an algorithm that is less sensitive to outliers since our centroids are less affected by extreme values. As with the K-means algorithm, the PAM clustering method requires us to select an optimal number of clusters (although we may use R to help us calculate silhouette widths). The main disadvantage of the PAM method is that calculating the dissimilarity matrix and performing swaps of points may be computationally expensive and therefore PAM is unsuitable for large datasets. If we are dealing with datasets with a large number of observations or datasets with a high dimensionality, we prefer to use a different method of clustering, *clustering large applications* (CLARA), also proposed by Kaufman and Rousseeuw (1990), which simplifies the PAM method by using sampling.

4.4.3 Clustering Large Applications (CLARA)

The CLARA method of clustering aims to deal with data of thousands or more observations. The CLARA method examines a small sample of the dataset

which it then performs PAM on. This creates a set of optimal medoids. The algorithm then repeats this sampling until a maximum number of iterations is reached. This method is effective for clustering large datasets. Performing CLARA instead of PAM is less computationally expensive when confronted with big data.

We again examine the optimal number of clusters recommended by R, using the *silhouette width* measure method:

```
fviz_nbclust(cars, clara, method = "silhouette")+
  theme_classic()
```

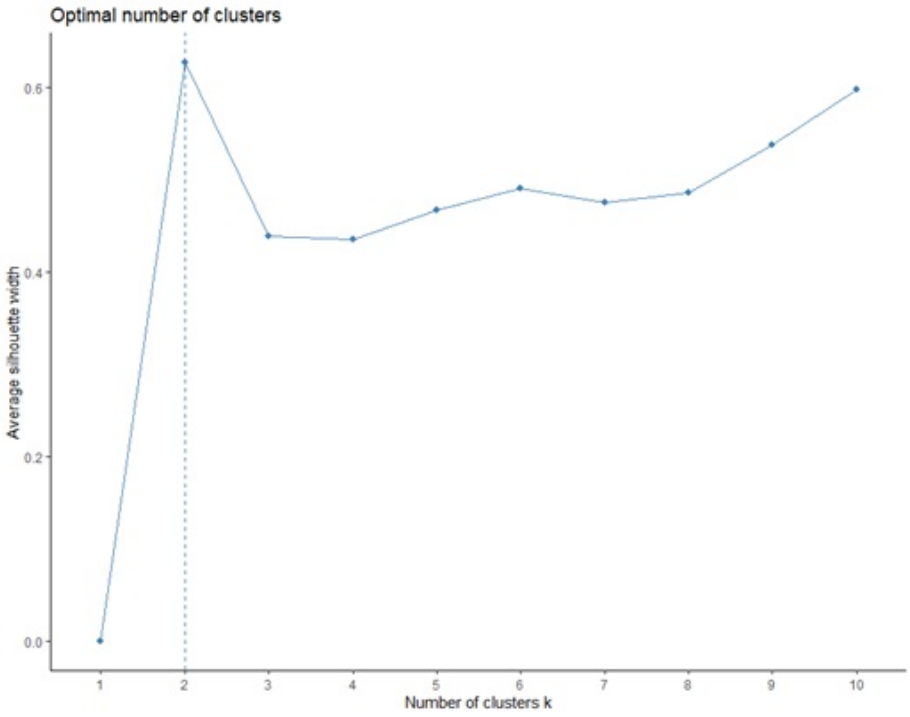


Figure 25: *Optimal number of clusters recommended by the CLARA algorithm for the MTCars dataset.*

R again recommends 2 clusters for applying CLARA to the dataset. This is consistent with the number of clusters recommended by the K-means and PAM algorithms.

We check the recommendation for the *carsreduced* dataset:

```
fviz_nbclust(carsreduced, clara, method = "silhouette")+
  theme_classic()
```

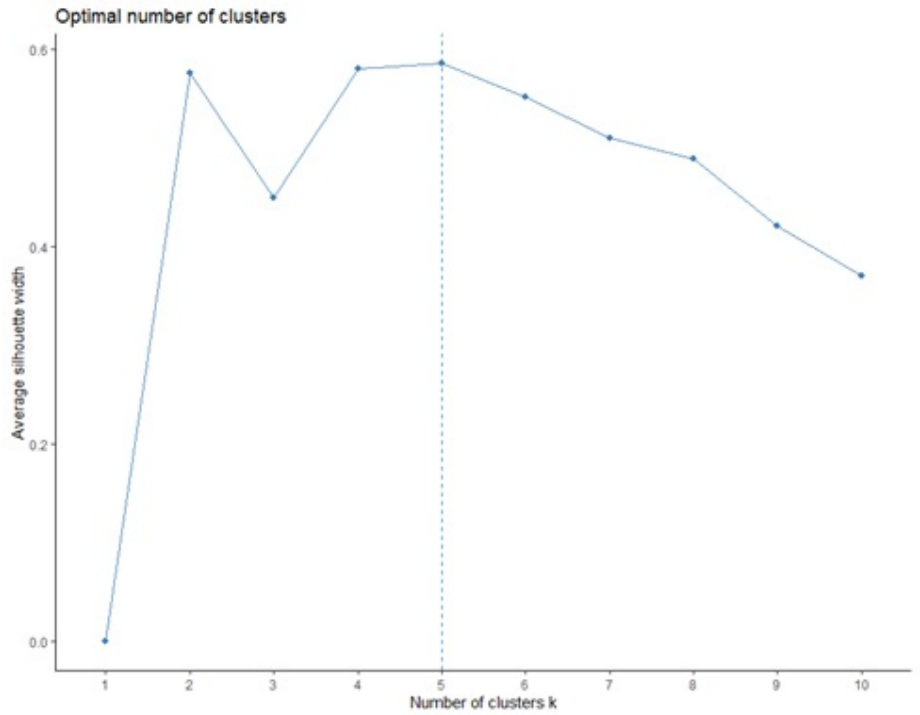


Figure 26: *Optimal number of clusters recommended by the CLARA algorithm for the carsreduced dataset.*

R again recommends K=5 clusters for performing CLARA on the *carsreduced* dataset.

We once more examine the recommended clustering:

```
clara.alg <- clara(cars, 2, pamLike = TRUE,samples = 5)
clara.alg
```

```
Call:     clara(x = cars, k = 2, pamLike = TRUE)
Medoids:
                  mpg cyl  disp  hp drat    wt  qsec
Toyota Corona    21.5   4 120.1  97 3.70 2.465 20.01
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02
Objective function:      62.04565
Clustering vector:       Named int [1:32] 1 1 1 2 2 1 2 1 1 1 1 2 2 2 2 2 2 1 ...
 - attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" "Hornet Sportabout" "Valian
t" "Duster 360" ...
Cluster sizes:           17 15
Best sample:
 [1] Mazda RX4           Mazda RX4 Wag       Datsun 710          Hornet 4 Drive      Hornet Sportabout
 [6] Valiant             Duster 360          Merc 240D           Merc 230            Merc 280
[11] Merc 280C           Merc 450SE          Merc 450SL          Merc 450SLC         Cadillac Fleetwood
[16] Lincoln Continental Chrysler Imperial   Fiat 128            Honda Civic         Toyota Corolla
[21] Toyota Corona       Dodge Challenger    AMC Javelin         Camaro Z28          Pontiac Firebird
[26] Fiat X1-9           Porsche 914-2       Lotus Europa        Ford Pantera L      Ferrari Dino
[31] Maserati Bora       Volvo 142E

Available components:
 [1] "sample"     "medoids"    "i.med"      "clustering" "objective"  "clusinfo"   "diss"       "call"
 [9] "silinfo"    "data"
```

Figure 27: *Output for CLARA algorithm on MTCars. Here the suggested clusters are not so clear.*

We also compute the clustering for the *carsreduced* dataset:

```
clara.alg.reduced<-clara(carsreduced, 5, pamLike = TRUE,samples = 5)
clara.alg.reduced
```

```
> clara.alg.reduced
Call:     clara(x = carsreduced, k = 5, samples = 5, pamLike = TRUE)
Medoids:
                   mpg cyl  disp  hp drat    wt  qsec
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02
Valiant           18.1   6 225.0 105 2.76 3.460 20.22
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40
Cadillac Fleetwood 10.4  8 472.0 205 2.93 5.250 17.98
Objective function:      17.83227
Clustering vector:       Named int [1:15] 1 1 1 2 3 2 3 1 1 1 1 4 4 4 5
 - attr(*, "names")= chr [1:15] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" "Hornet Sportabout"
 "Valiant" "Duster 360" ...
Cluster sizes:           7 2 2 3 1
Best sample:
 [1] Mazda RX4          Mazda RX4 Wag      Datsun 710         Hornet 4 Drive     Hornet Sportabout
 [6] Valiant            Duster 360         Merc 240D          Merc 230           Merc 280
[11] Merc 280C          Merc 450SE         Merc 450SL         Merc 450SLC        Cadillac Fleetwood

Available components:
 [1] "sample"     "medoids"    "i.med"      "clustering" "objective"  "clusinfo"   "diss"
 [8] "call"       "silinfo"    "data"
```

Figure 28: *Output for CLARA algorithm on carsreduced.*

The output does not make it clear which observations should be placed into each cluster; therefore we plot the clustering according to the CLARA method:

```
fviz_cluster(clara.alg,
             ellipse.type = "t", # Concentration ellipse
              pointsize = 1,
             ggtheme = theme_classic(),repel = TRUE,main="Cluster
             plot (CLARA)")
```

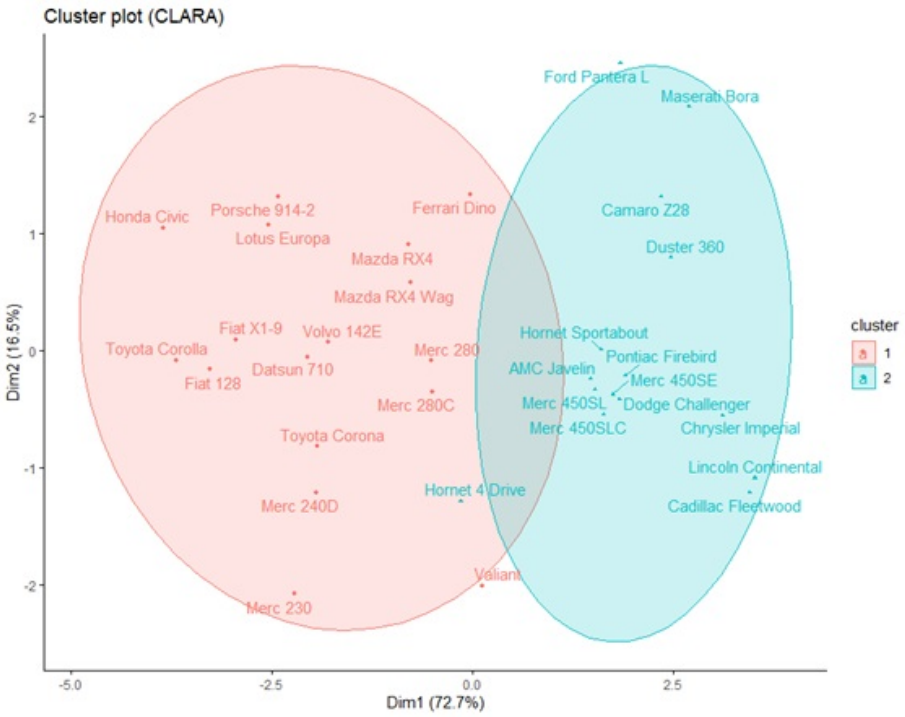


Figure 29: *Results of the CLARA cluster method on our MTCars dataset. The partitioning of our data into 2 clusters based on the CLARA method. It is worth noting that CLARA gives an identical clustering to the PAM method.*

For K=2 partitions, PAM and CLARA give identical results on the *MTCars* dataset. We also compare how CLARA performs against PAM for the *carsreduced* dataset.

```
fviz_cluster(clara.alg.reduced,
             ellipse.type = "t", # Concentration ellipse
             pointsize = 1,
             ggtheme = theme_classic(),repel = TRUE,main="Cluster plot
             (CLARA)"
```

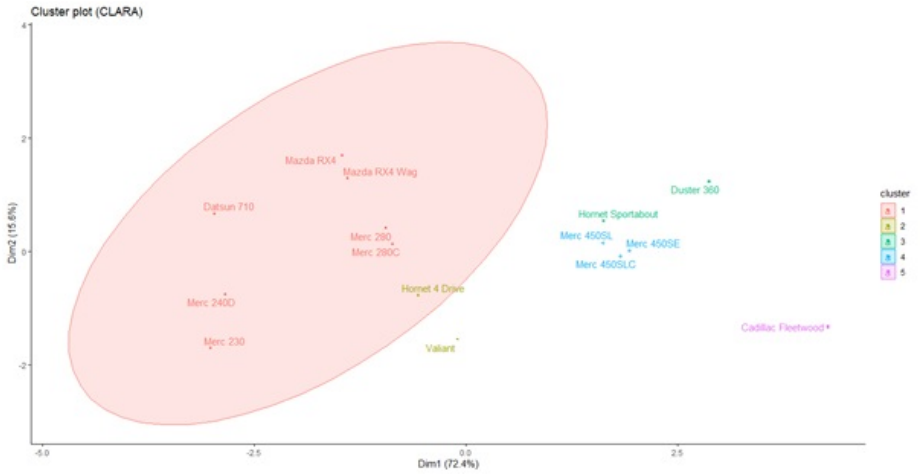


Figure 30: *Results of the CLARA cluster method on our carsreduced dataset. The clustering recommended by CLARA on K=5 clusters. Again, this is the same as the recommended clustering given by the PAM method.*

We imagine that, since we are dealing with a relatively small dataset, there would be no reason to favour CLARA over PAM. If, however, we had thousands of observations, we may prefer to use the sampling approach provided by the CLARA method. Naturally, the effectiveness of the CLARA method is affected by the size of the sample taken. For a small sample size, CLARA will perform quickly but at the expense of the quality of clustering.

4.5 Comparison of Clustering Algorithms

Following our discussion of each clustering algorithm, we here compare the advantages and disadvantages of each method. All methods relied on the selection of an optimal number of clusters, which may be achieved by examining a plot of the *silhouette width* or the *within-cluster sum of squares*. K-means clustering is the simplest of the methods we have explored; it calculates the mean value of all points within each cluster and places the centroid of each cluster based on this calculation. It then iterates to minimise the total within sum of squares distance. As previously discussed, K-means is simple to employ but its sensitivity to outliers may make it unsuitable for certain datasets. Before employing the K-means method, we may wish to plot the data and examine whether our data contains outliers which will skew our centroids. For datasets without many extreme values, K-means is a robust and effective method of clustering.

As previously mentioned, the PAM clustering algorithm partitions around medoids instead of means. As such, the PAM algorithm is much more effective on datasets containing outliers. The PAM algorithm assigns each object to its nearest medoid (these are existing datapoints instead of calculated means). As such, PAM is an effective alternative for clustering. When applied to our $MTCars$ dataset, K-means and PAM gave similar results. A couple of datapoints were identified as elements of different clusters depending on the algorithm chosen, but mostly the results were identical. We may imagine, however, that a dataset

with many outliers would give different results based on whether K-means or PAM was chosen as the clustering method.

The clustering large applications (CLARA) algorithm is an extension of the PAM method, but uses sampling to apply PAM to larger datasets. As with PAM and K-means, the CLARA algorithm involves the selection of several clusters into which to partition the dataset; this requires some prior knowledge of the structure of the data. Because it uses sampling, CLARA is less memory intensive than PAM when applied to large datasets, however this efficiency may come at the expense of effectiveness. If the sample size is too small, CLARA may not be as effective. Therefore, we wish to optimise the sample size chosen for the CLARA algorithm such that the sample size is large enough for effective clustering without being too large and therefore inefficient. When applied to the $MTCars$ dataset, the CLARA method gave identical results to the PAM method. This is because our dataset is relatively small, containing only 32 observations on 7 variables. If our dataset contained thousands or millions of datapoints, the PAM method would be inefficient, and we would rely on CLARA to sample the data and perform clustering based on these samples.

4.6 Conclusion: Partitioning Clustering

We conclude that the most effective method of clustering is dependant on the size, dimensionality and structure of the data. While more basic methods such as K-means clustering may be effective on smaller sets without extreme values, the evolving complexity of modern datasets has resulted in a requirement for sampling approaches to perform clustering effectively. As we have seen from our datasets, each method usually gives similar results and therefore it is the decision of the statistician to choose the most suitable method of partitioning clustering based on the complexity and structure of the dataset.

We now turn our attention to a different method of clustering entirely: hierarchical clustering

Part 2: Hierarchical Methods

5 Introduction: Hierarchical Clustering

Hierarchical clustering is a type of clustering method which uses a hierarchical tree-like structure. Hierarchical methods rely on the calculation of a similarity matrix to organise the data into a tree-like structure, called a *dendrogram*. This method is often preferred to partitioning clustering because it circumvents the requirement for a statistician to select the optimal number of clusters. Instead we may decide, upon the creation of the hierarchical structure, where to cut the tree into separate clusters. In this project, we will examine the two main types of hierarchical clustering, *agglomerative* and *divisive*, examine the dendrograms which provide a visual representation of the clustering and compare dendrograms created through different hierarchical clustering methods.

The hierarchy of the clustering corresponds to the structure of the dendrogram. The root node represents all points in the dataset. Each level of the dendrogram represents a different level of clustering. One major advantage of the hierarchical clustering methods is that we can easily alter the nature of the clustering based on how we cut our dendrogram.

As previously mentioned, the two main types of hierarchical clustering are agglomerative and divisive:

Agglomerative Clustering: Agglomerative clustering begins with assigning each datapoint to its own cluster (called a leaf) and then iteratively merges similar points into clusters until we reach one large cluster of all points (called the root). In this section, this is the main type of hierarchical clustering we will use.

Divisive Clustering: Divisive clustering is the opposite process of agglomerative clustering. We begin with the root and iteratively divide dissimilar points until each point is assigned to its own cluster (or leaf).

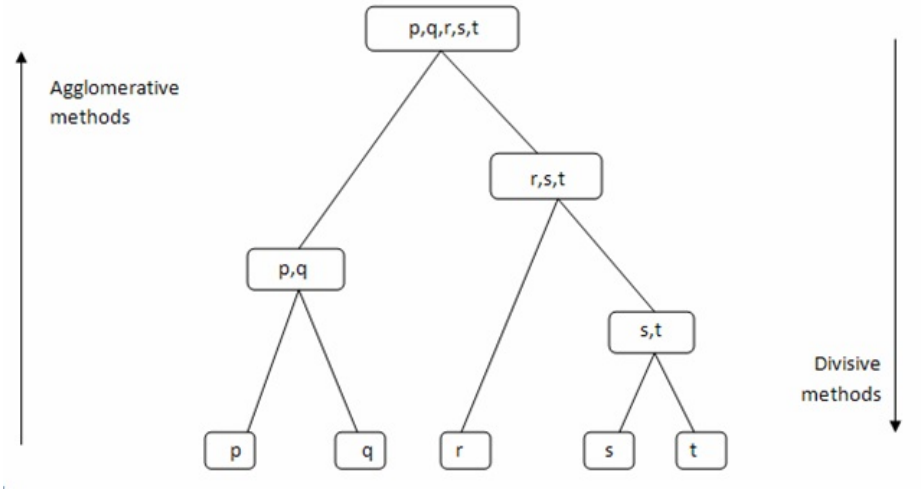


Figure 31: *A visual representation of the two hierarchical clustering methods*[14]. *Agglomerative methods begin with the leaves of the tree and iteratively combine groups based on their similarity until we reach the root. Divisive methods begin with the root of the tree and iteratively divide groups based on their dissimilarity until we finish at the leaves.*

We can use the statistical software R to produce the dendrograms which provide a graphical representation of clusters in a tree structure. Since there are different levels of clustering, we may choose, based on this plot, into how many clusters we wish to group the data. For our analysis, we will perform different hierarchical clustering methods on the Motor Trend Car Road Tests dataset (*MTCars*), which provides data for 32 models of motorcars produced between 1973-1974[4]. This is the same dataset we used to examine partitioning clustering methods.

6 Agglomerative Clustering

Since clustering is broadly defined as the process of grouping items based on their similarity, the bottom-up approach of agglomerative clustering is by far the most commonly used method of hierarchical clustering. At each iteration, the agglomerative clustering algorithm groups the two most similar clusters until all datapoints are included in one large cluster (the root). Performing agglomerative clustering first requires the calculation of the distance matrix between pairs of points. The algorithm then uses a linkage function to group the datapoints into a dendrogram, which we may visualise using R. Finally, after the dendrogram has been created, we decide where to cut it. This decision will determine the number of clusters produced. As discussed previously, we may decide between different metrics (such as Euclidean, Manhattan or Mahalanobis distances) to create our distance matrix. To provide a direct comparison between the hierarchical and partitioning clustering methods, we will again use the Euclidean distance as our main distance measure.

Again, because our dataset contains variables measured using different units, it

is important to scale the data so that the distances are not skewed by variables with a larger variability:

```
ScaledCars<-scale(cars)
EuclideanDistance<-dist(ScaledCars,method="euclidean")
round(as.matrix(EuclideanDistance)[1:5,1:5],1)
```

	Mazda RX4	Mazda RX4 Wag	Datsun 710	Hornet 4 Drive	Hornet Sportabout
Mazda RX4	0.0	0.4	1.8	2.5	2.8
Mazda RX4 Wag	0.4	0.0	1.6	2.2	2.7
Datsun 710	1.8	1.6	0.0	2.4	3.9
Hornet 4 Drive	2.5	2.2	2.4	0.0	2.2
Hornet Sportabout	2.8	2.7	3.9	2.2	0.0

Figure 32: *Distance matrix for the first 5 observations in the MTCars dataset.*

We will now use a linkage function to pair (or link) groups of datapoints based on their distance measures. These objects are then combined into larger objects until all the datapoints are combined into a tree. We may use the *hclust* function in *R*, available through the *stats* package. The function takes form *hclust(d, method="complete")* where the d argument represents some distance structure and the *method* argument represents the agglomeration method to be used. Since we are calculating distances between clusters and there are many different methods of doing this, we will briefly discuss the main agglomeration methods available, based on the distance matrix.

6.1 Linkage Functions

Complete Linkage: This method takes the distance between two clusters as the maximum pairwise distance between all datapoints in cluster 1 and cluster 2. This is the default linkage method used by the *hclust* function. We may imagine this as a "worst case scenario" for distances. Because it relies on the maximum length, it is sensitive to outliers. For clusters c_1,c_2, we define the complete linkage as:

$$dist(c_1,\, c_2) = \max_{x_1 \in\, c_1,\, x_2 \in\, c_2} dist(x_1,\, x_2)$$

Single Linkage: This method takes the distance between two clusters as the minimum pairwise distance between all datapoints in cluster 1 and cluster 2. We may imagine this as a "best case scenario" for distances. Because it relies on the minimum length, it is sensitive to outliers. For clusters c_1,c_2, we define the single linkage as:

$$dist(c_1,\, c_2) = \min_{x_1 \in\, c_1,\, x_2 \in\, c_2} dist(x_1,\, x_2)$$

Average Linkage: As its name suggests, this method takes the mean pairwise distance measurement between cluster 1 and cluster 2. This method is less sensitive to outliers as it calculates an average. For clusters c_1,c_2, we define the average linkage as:

$$dist(c_1,\, c_2) = \frac{1}{|c_2|} \sum_{x_1 \in\, c_1} \sum_{x_2 \in\, c_2} dist(x_1,\, x_2)$$

Centroid Linkage: This method takes the distance between two clusters as the distance between their centroids. For clusters c_1, c_2, and centroids $\widehat{x_1}$, $\widehat{x_2}$, we define the centroid linkage as:

$$dist(c_1,\, c_2) = \left(\left(\frac{1}{|c_1|} \sum_{x_1 \in\, c_1} \widehat{x_1} \right), \left(\frac{1}{|c_2|} \sum_{x_2 \in\, c_2} \widehat{x_2} \right) \right)$$

Ward's Method Linkage: This method merges the pair of clusters with the minimum between-cluster distances at each step. According to Ward, the distance between c_1 and c_2 is equal to the increase in the sum of squares when they are merged. For clusters c_1, c_2 with centroids $\widehat{x_1}$, $\widehat{x_2}$, we define the Ward's method linkage as:

$$dist(c_1,\, c_2) = \sum_{x \,\in\, c_1 \cup c_2} (x - \widehat{x_{1\cup 2}})^2 - \sum_{x \,\in\, c_1 \cup c_2} (x - \widehat{x_1})^2 - \sum_{x \,\in\, c_1 \cup c_2} (x - \widehat{x_2})^2$$

The structure of our dendrogram depends on the linkage method we decide to use. We will examine each dendrogram on the *carsreduced* dataset and discuss our findings:

```
distancereducedhccomplete<-hclust(d=distancereduced,method="complete")
fviz_dend(distancereducedhccomplete,cex=0.55,main = "Cluster Dendrogram
(Complete Method)")
```

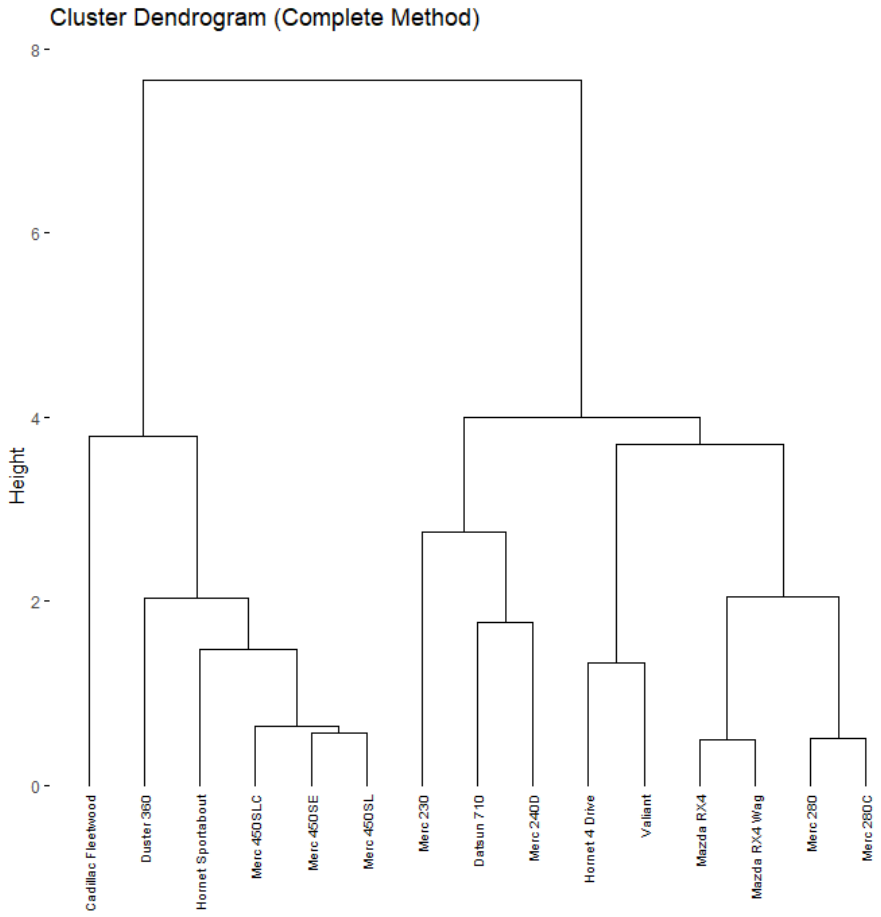


Figure 33: *The dendrogram resulting from the "complete method" on the carsreduced dataset.*

```
distancereducedhcsingle<-hclust(d=distancereduced,method="single")
fviz_dend(distancereducedhcsingle,cex=0.50,main = "Cluster Dendrogram
(Single Method)")
```

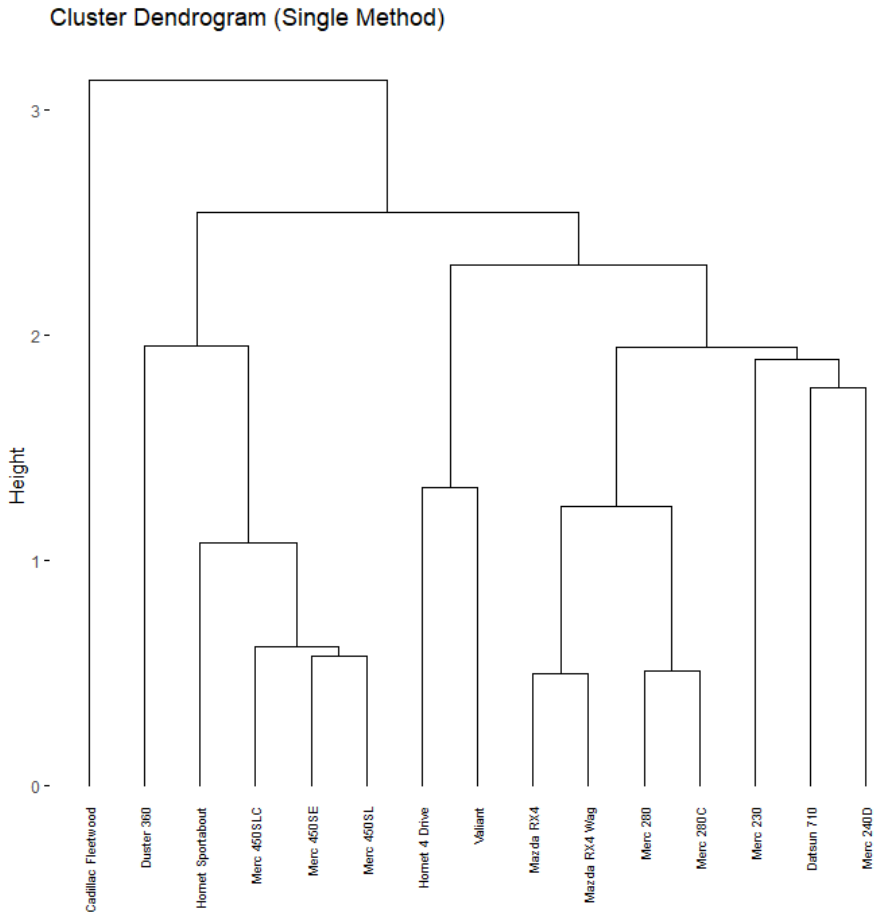


Figure 34: *The dendrogram resulting from the "single method" on the carsreduced dataset.*

```
distancereducedhcaverage<-hclust(d=distancereduced,method="average")
fviz_dend(distancereducedhcaverage,cex=0.5, main = "Cluster Dendrogram
(Average Method)")
```

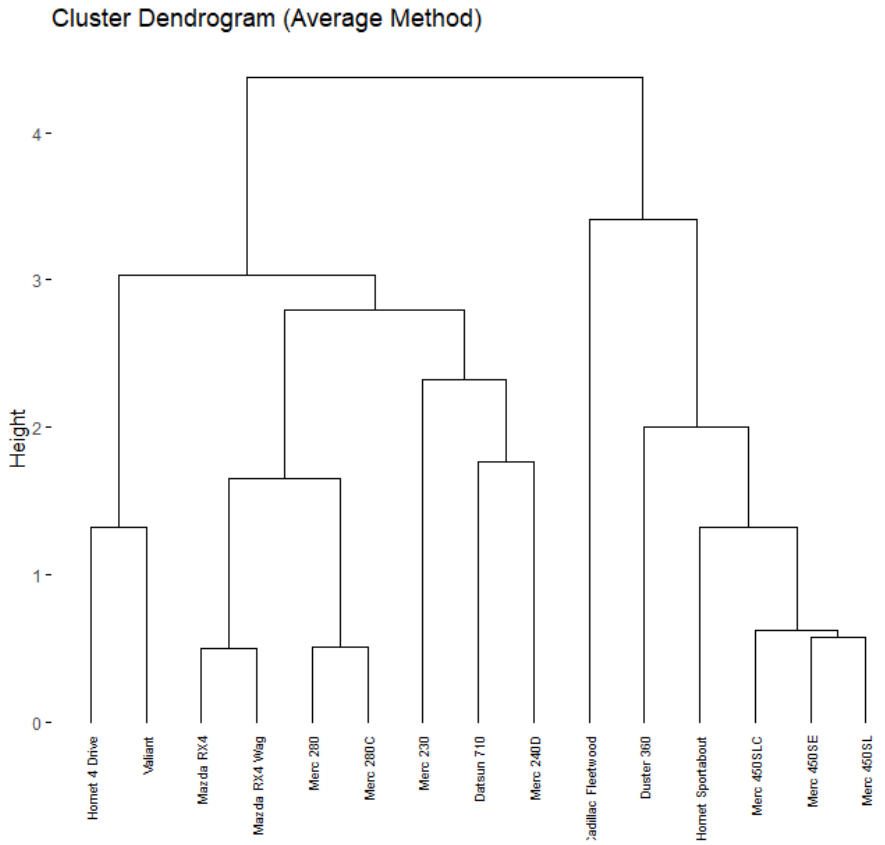


Figure 35: *The dendrogram resulting from the "average method" on the carsreduced dataset.*

```
distancereducedhccentroid<-hclust(d=distancereduced,method="centroid")
fviz_dend(distancereducedhccentroid,cex=0.40, main = "Cluster Dendrogram
(Centroid Method)")
```

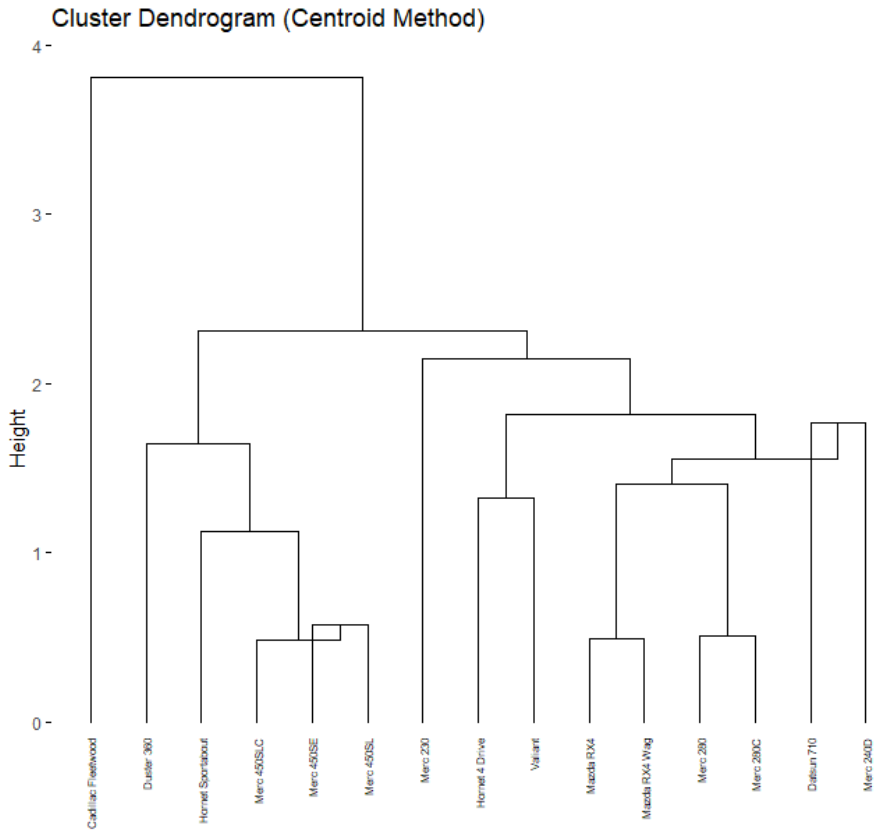


Figure 36: *The dendrogram resulting from the "centroid method" on the carsreduced dataset.*

```
distancereducedhcward<-hclust(d=distancereduced,method="ward.D2")
fviz_dend(distancereducedhcward,cex=0.5, main = "Cluster Dendrogram
(Ward Method)")
```

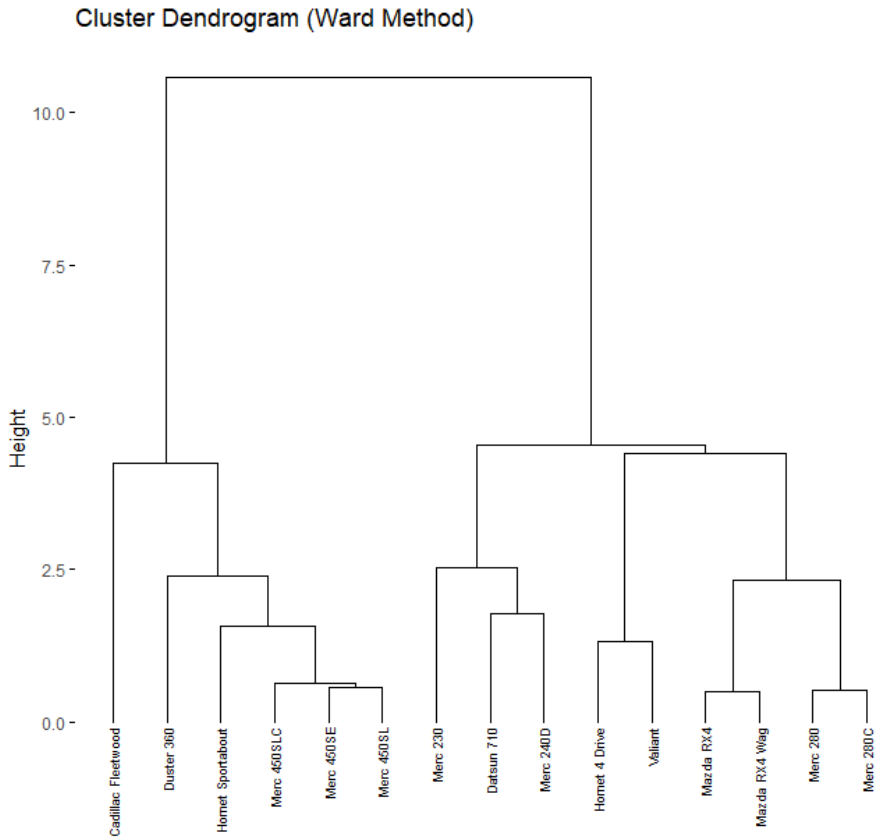


Figure 37: *The dendrogram resulting from the "Ward method" on the carsreduced dataset. It is interesting to note that the complete method and Ward method gave identical dendrograms.*

From the above dendrograms, we observe that the single method and centroid method both identify the *Cadillac Fleetwood* datapoint as an outlier because this datapoint's leaf comes directly from the root. This is problematic if we chose to partition the dataset into 2 clusters as these two clusters would be the *Cadillac Fleetwood* in cluster 1 and all other cars in cluster 2. The complete method and Ward method create identical dendrograms. The dendrograms from the complete and Ward methods appear to be more effective if we are attempting to avoid outliers being assigned to their own cluster.

The point on the y-axis at which leaves combine corresponds to their distance. Datapoints which are combined at a higher level have a larger pairwise distance and therefore are more dissimilar. We may, from the above plots, assume that, unsurprisingly, the *Mazda RX4 & Mazda RX4 Wag* are extremely similar. We also assume similarities exist between the *Mercedes 280 & Mercedes 280C*, *Mercedes 450SL & Mercedes 450SE* and, to a lesser extent, the *Hornet 4 Drive & Valiant*. Since the *Cadillac Fleetwood* only combines with other leaves at a large height or at the root (depending on the linkage method chosen), we may assume that the *Cadillac Fleetwood* is dissimilar to all other cars. We also see consistent

dissimilarities between certain pairs of cars, such as the *Duster 360* & *Mercedes 280*, *Hornet Sportabout* & *Datsun 710* and *Mercedes 230* & *Mercedes 450SLC*.

6.2 Assessing Each Link Function

To examine which of these linkage methods best reflects the pairwise distances of our data, we calculate the correlation between the *cophenetic distance* (the height on the dendrogram at which the branches of two separate objects merge to form a single branch) and the original distance data. This correlation takes values between 0 and 1. The closer the correlation is to 1, the better the dendrogram reflects the distances observed in the data.

We calculate the cophenetic score for each of our linkage methods for the *carsreduced* dataset:

Complete Method:

```
copheneticdistcomplete<-cophenetic(distancereducedhccomplete)
cor(copheneticdistcomplete,distancereduced)
[1] 0.7339261
```

Single Method:

```
copheneticdistsingle<-cophenetic(distancereducedhcsingle)
cor(copheneticdistsingle,distancereduced)
[1] 0.765752
```

Average Method:

```
copheneticdistaverage<-cophenetic(distancereducedhcaverage)
cor(copheneticdistaverage,distancereduced)
[1] 0.7527959
```

Centroid Method:

```
copheneticdistcentroid<-cophenetic(distancereducedhccentroid)
cor(copheneticdistcentroid,distancereduced)
[1] 0.7429086
```

Ward Method:

```
copheneticdistward<-cophenetic(distancereducedhcward)
cor(copheneticdistward,distancereduced)
[1] 0.720653
```

From the above output, we see that the single method of linkage best reflects the pairwise distances from the dataset. Therefore, we endorse this linkage method.

7 Cutting The Dendrogram

Once we have decided on the linkage method to use, we perform our clustering. Unlike partitioning methods, hierarchical clustering does not tell us how many clusters there are or at which level we should set the dendrogram for the optimal clustering. Instead, we may use the *cutree* function available in the *stats*

package in *R*. This function will cut the dendrogram according to a desired number of clusters (the K argument in the function), or the function will cut the dendrogram at a specified height. The *cutree* function produces a vector which specifies which observation belongs to each cluster.

Suppose we wish to cut the dendrogram from the *carsreduced* dendrogram with single linkage method into 3 clusters:

```
cutsingle<-cutree(distancereducedhcsingle,k=3)
```

```
> cutsingle
        Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive  Hornet Sportabout
                1                   1                   1                   1                   2
          Valiant          Duster 360           Merc 240D           Merc 230           Merc 280
                1                   2                   1                   1                   1
         Merc 280C          Merc 450SE          Merc 450SL         Merc 450SLC Cadillac Fleetwood
                1                   2                   2                   2                   3
```

Figure 38: *Result from cutting the tree into K=3 subtrees.*

From the above output we can clearly see which cluster the hierarchical method with single linkage recommends. If we had a large number of observations and/or a large number of desired clusters, we could use the *R* command:

```
rownames(ScaledCars.reduced)[cutsingle==1]
```

To extract all the observations in cluster 1, we could then change this to 2,3,4 etc. While the output above reflects the dendrogram clustering, it does not provide a visual representation. To create a plot of the cut dendrogram, we again use *R* function *fviz_dend* but specify an argument for K clusters:

```
fviz_dend(distancereducedhcsingle, k = 3,cex = 0.6, rect = TRUE,
main="Cut Cluster Dendrogram (Single Linkage)",
color_labels_by_k =TRUE,k_colors = c("red","darkgreen","blue"))
```

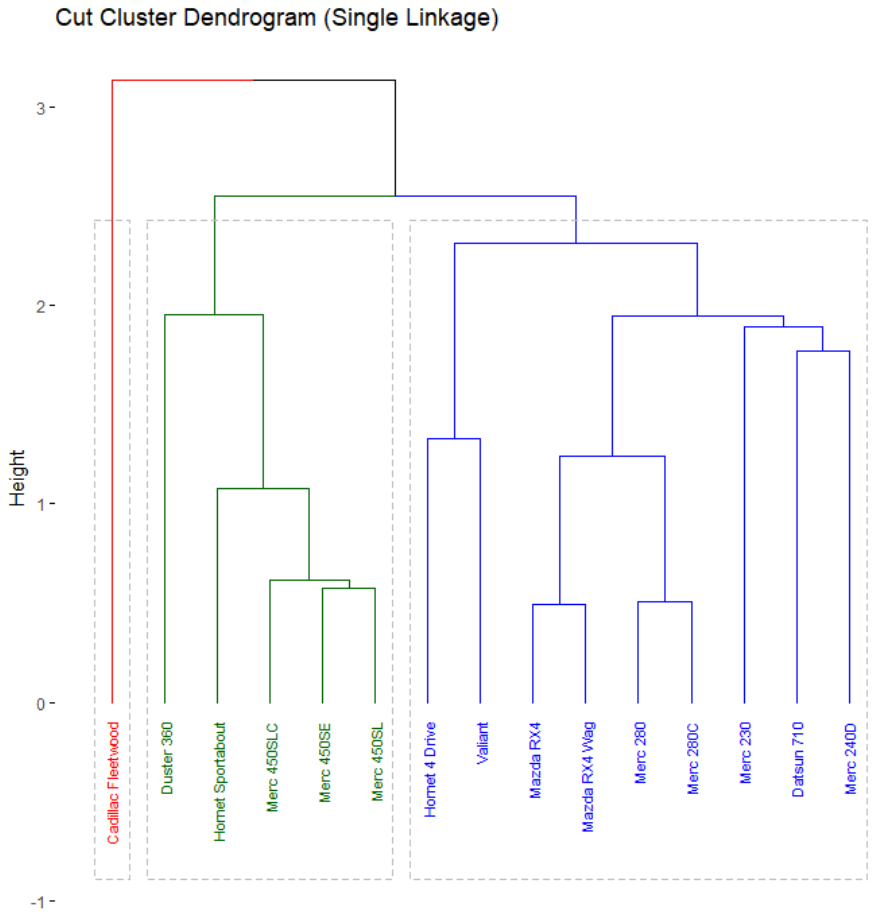


Figure 39: *The dendrogram for the carsreduced dataset, divided into 3 clusters. As previously mentioned, this method allocates the Cadillac Fleetwood to its own cluster.*

When cut into 3 clusters, one of which contains only a single observation, the dendrogram is drastically different from the partitioning clustering methods we have seen on this dataset. When we applied the K-means method of partitioning on our *carsreduced* dataset, we saw three clusters, each with several elements. We may again use the *fviz_cluster* command to create a plot of our hierarchical clustering:

```
fviz_cluster(list(data = carsreduced, cluster = cutsingle),
repel =TRUE,show.clust.cent = FALSE, ellipse.type = "t",
ggtheme = theme_minimal(),main="Visual Representation
of Dendrogram Cut",palette=c("blue","darkgreen","red"))
```

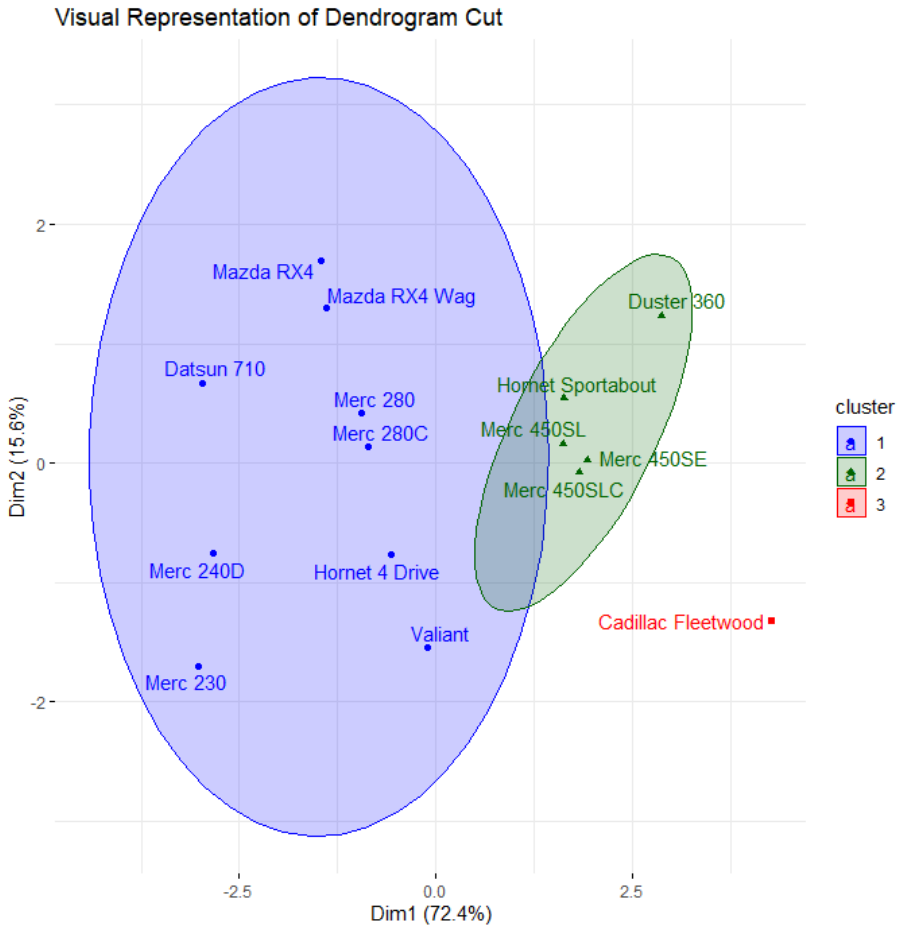


Figure 40: *A plot of our hierarchical clustering on the first 2 principal components. We compare against the similar plot of K-means. In this plot, the Valiant and Hornet 4 Drive have joined the* 1^{st} *cluster, the Hornet Sportabout and Duster 360 have joined the* 2^{nd} *cluster and the Cadillac Fleetwood is the only observation in the* 3^{rd} *cluster.*

We also examine a cut into K=5 clusters, to compare against PAM and CLARA from the partitioning clustering methods:

```
cutsingle5<-cutree(distancereducedhcsingle,k=5)
```

```
cutsingle5
       Mazda RX4        Mazda RX4 Wag           Datsun 710       Hornet 4 Drive    Hornet Sportabout              Valiant
               1                    1                    1                    2                    3                    2
      Duster 360            Merc 240D             Merc 230             Merc 280             Merc 280C            Merc 450SE
               4                    1                    1                    1                    1                    3
     Merc 450SL           Merc 450SLC      Cadillac Fleetwood
               3                    3                    5
```

Figure 41: *Result from cutting the tree into* K*=5 subtrees.*

We examine the correspondng dendrogram:

```
fviz_dend(distancereducedhcsingle, k = 5,cex = 0.6, rect = TRUE
,main="Cut Cluster Dendrogram (Single Linkage)",
color_labels_by_k =TRUE,k_colors = c("red","darkgreen","blue","gold3","purple"))
```

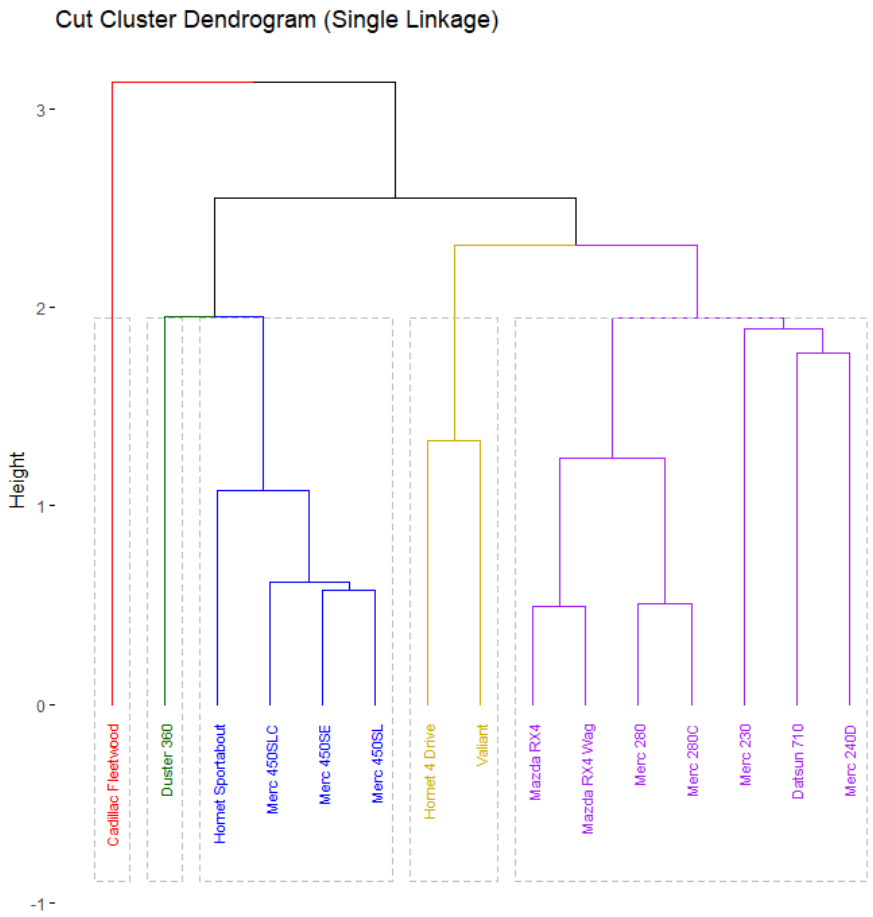


Figure 42: *The dendrogram for the carsreduced dataset, divided into 5 clusters. This method allocates the Cadillac Fleetwood and Duster 360 to their own respective clusters.*

To compare against the PAM and CLARA partitioning clustering algorithms (both of which recommended 5 clusters for the *carsreduced* dataset), we will divide our hierarchical cluster into 5 clusters. Again, the results plotted below take the 1^{st} and 2^{nd} principal components as the x-axis and y-axis respectively:

```
fviz_cluster(list(data = carsreduced, cluster = cutsingle5),
repel =TRUE,show.clust.cent = FALSE, ellipse.type = "t",
ggtheme = theme_minimal(),main="Visual Representation of Dendrogram
Cut",palette=c("purple","gold3","blue","darkgreen","red"))
```

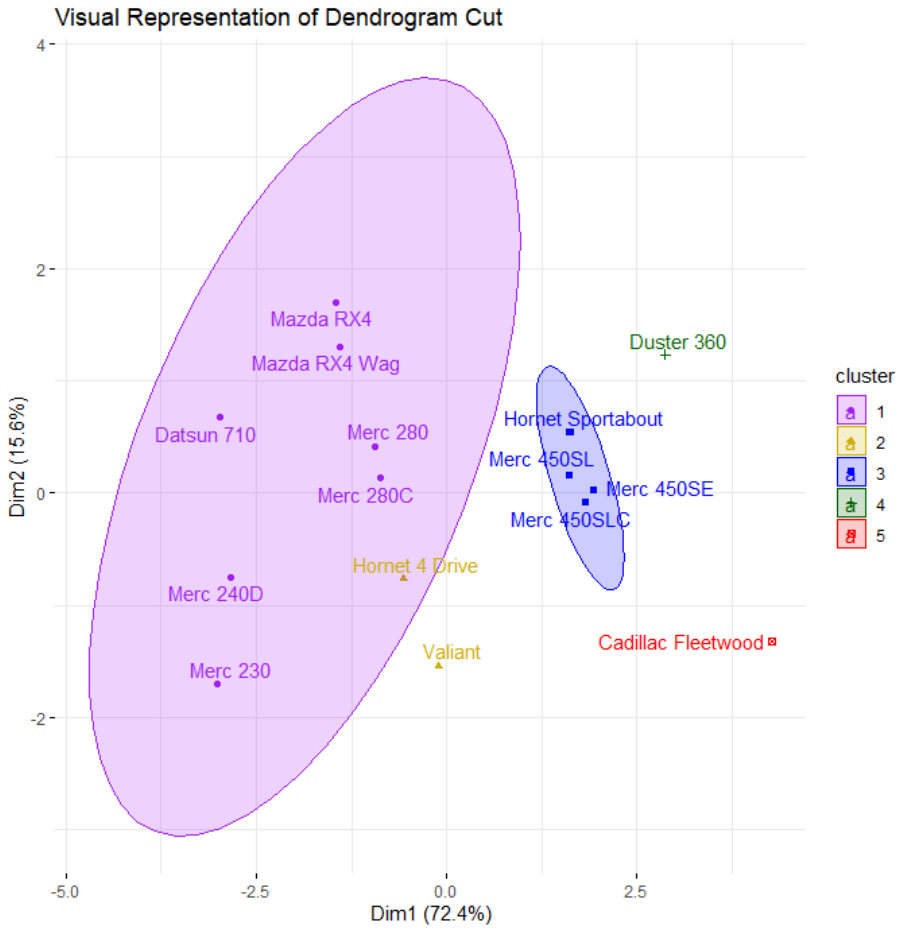


Figure 43: *A plot of the carsreduced dataset divided into 5 clusters based on the hierarchical clustering method with single linkage.*

From the above plot, we see that the hierarchical clustering technique produces different clusters to the PAM and CLARA partitioning cluster technique. In the partitioning techniques, the *Hornet Sportabout* was included in a cluster with the *Duster 360*. In the hierarchical method, however, the *Hornet Sportabout* is clustered with the *Mercedes 450* family and the *Duster 360* is allocated its own cluster. Unsurprisingly, the *Cadillac Fleetwood* is also allocated its own cluster.

8 Hierarchical Clustering on the Entire *MTCars* Dataset

Until now, we have performed hierarchical clustering techniques on the *carsreduced* dataset. Now we have discussed the process and examined a subset of the data, we examine the effect of performing hierarchical clustering on all 32 observations in the entire dataset. We imagine that, due to the larger number of datapoints, the dendrograms will be more convoluted and more difficult to read. We may

use the correlation between the cophenetic distance and original distances of the entire $MTCars$ dataset to choose a linkage method for hierarchical clustering:

Complete Method:

```
copheneticdistfullcomplete<-cophenetic(distancehccomplete)
cor(copheneticdistfullcomplete,distance)
[1] 0.7303783
```

Single Method:

```
copheneticdistfullsingle<-cophenetic(distancehcsingle)
cor(copheneticdistfullsingle,distance)
[1] 0.6909525
```

Average Method:

```
copheneticdistfullaverage<-cophenetic(distancehcaverage)
cor(copheneticdistfullaverage,distance)
[1] 0.745617
```

Centroid Method:

```
copheneticdistfullcentroid<-cophenetic(distancehccentroid)
cor(copheneticdistfullcentroid,distance)
[1] 0.7378652
```

Ward Method:

```
copheneticdistfullward<-cophenetic(distancehcward)
cor(copheneticdistfullward,distance)
[1] 0.724033
```

For the entire $MTCars$ dataset, we endorse the average linkage method:

```
fviz_dend(distancehcaverage,cex=0.5,main = "Cluster Dendrogram (Average Method)")
```

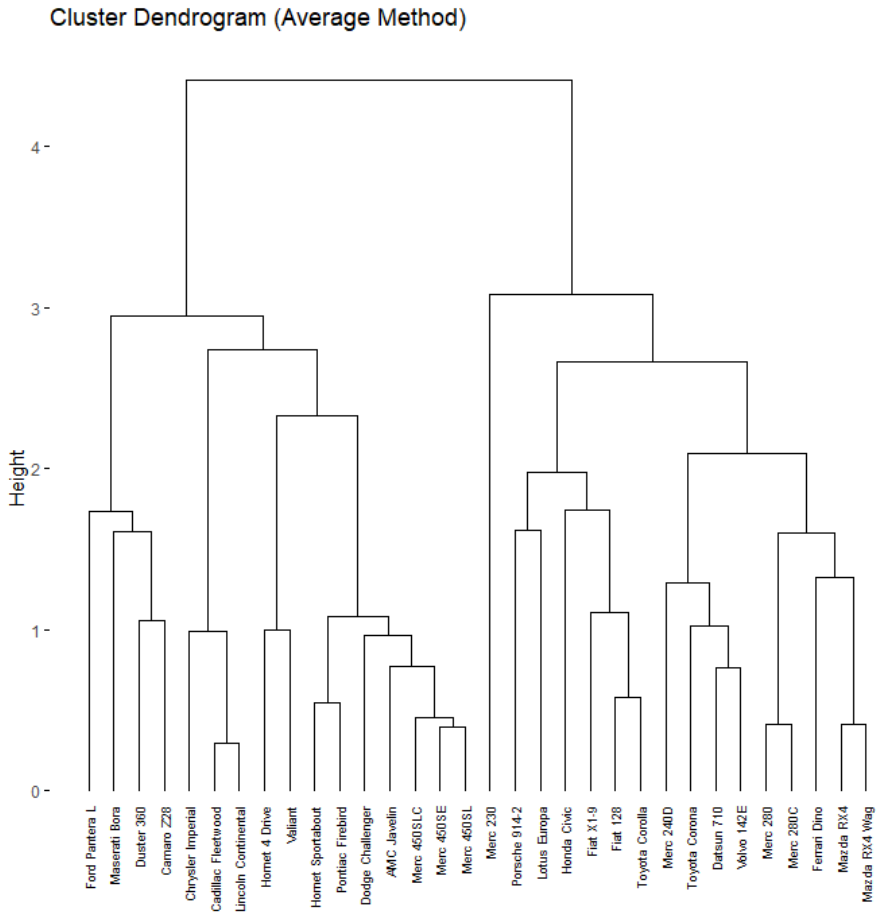


Figure 44: *The dendrogram of the entire MTCars dataset when using the average linkage method.*

From the above dendrogram of the entire *MTCars* dataset, we see that the data could be divided into 2 clusters of approximately equal size. This bears similarity to what we observed when the data was partitioned into 2 clusters using partitioning clustering techniques. To examine this further, we cut the dendrogram, this time into 2 clusters:

```
fviz_dend(distancehcaverage,k=2,cex=0.5,main = "Cluster Dendrogram
(Average Method)",rect=TRUE)
```

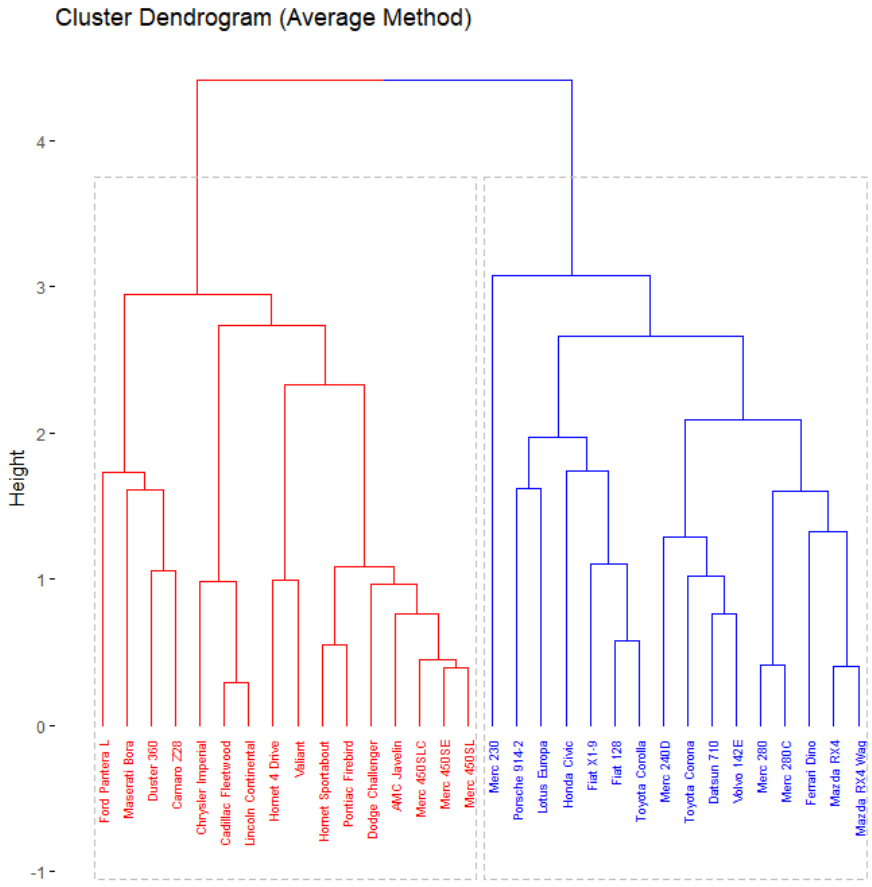


Figure 45: *The dendrogram for the entire MTCars dataset, divided into 2 clusters.*

We again plot our hierarchical clustering using the *fviz_cluster* command:

```
fviz_cluster(list(data = cars, cluster = cutreefullcars), repel =TRUE,
show.clust.cent = FALSE, ellipse.type = "t", ggtheme = theme_minimal()
,main="Visual Representation of Dendrogram Cut",palette = c("blue","red"))
```

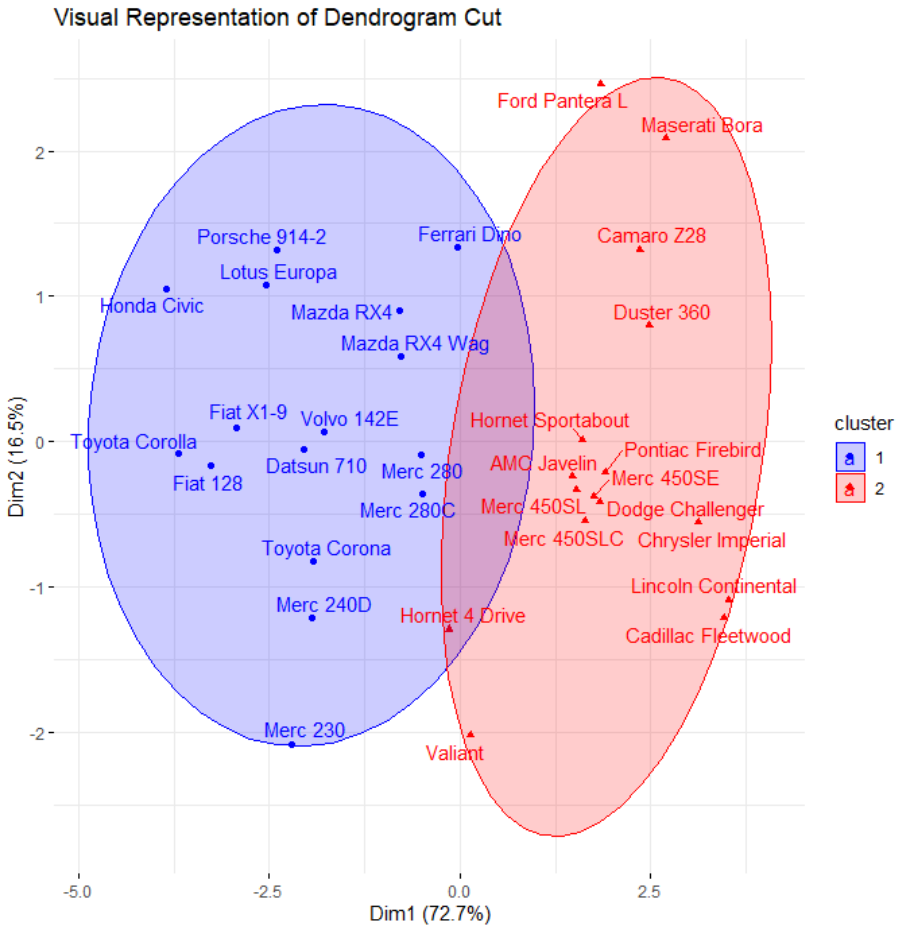


Figure 46: *A plot of the entire MTCars dataset divided into 2 clusters based on the hierarchical clustering method with average linkage.*

The above plot demonstrates that this method produces similar results to those seen in partitioning clustering. However, in this method, the *Valiant* and *Hornet 4 Drive* are both elements of the second cluster whereas in the K-means method of partitioning, both these cars were allocated to the first cluster. In the PAM and CLARA methods, the *Hornet 4 Drive* is allocated to the second cluster, however the *Valiant* is allocated to the first cluster, unlike in the hierarchical method where it is in the second. We conclude that this method produces different clustering results to each of the partitioning techniques.

9 Tanglegrams

R package *dendextend* provides functions for comparing the clustering results from various methods of hierarchical clustering. We will use the *tanglegram* function to create a plot which we use to compare dendrograms and the *cor.dendlist* function to calculate the correlations between different dendrograms[15]. Again, for simplicity we will use the *carsreduced* dataset to ensure that our plots are

easy to read.

```
dendreducedsingle<-as.dendrogram(distancereducedhcsingle)
dendreducedcomplete<-as.dendrogram(distancereducedhccomplete)
tanglegram(dendreducedsingle,dendreducedcomplete,lab.cex =0.8,
edge.lwd=1,main_left="Single Linkage Dendrogram",
main_right="Complete Linkage Dendrogram",
common_subtrees_color_branches = TRUE)
```

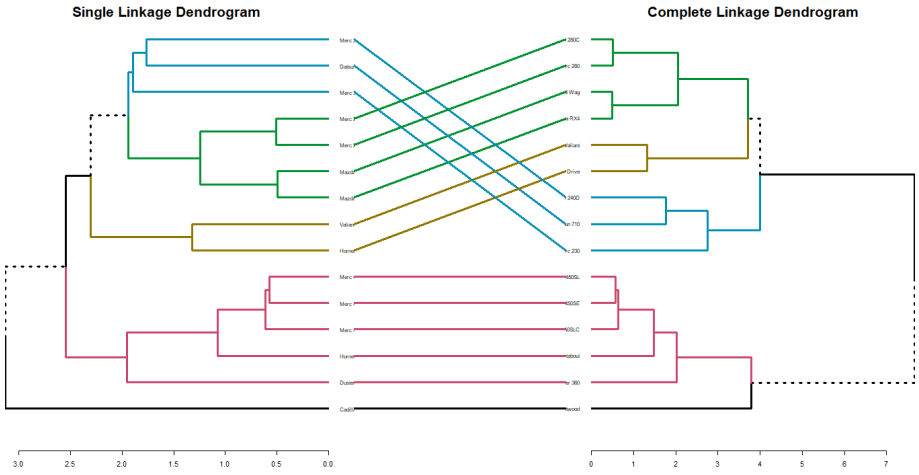


Figure 47: *Tanglegrams allow us to visually compare the results from different dendrograms based on different linkage functions. Here we see a comparison of the 'single linkage' function against the 'complete linkage' function. Nodes which contain a unique combination of items are represented with dashed lines. Similarly, subtrees which are present in both dendrograms share the same colour.*

From the above diagram, we compare the results of the single and complete linkage dendrogram. We see again that the single linkage method results in the *Cadillac Fleetwood* being assigned its own cluster. The horizontal straight lines between the two dendrograms towards the bottom of the diagram suggest that clusters will be extremely similar across the two methods. The diagonal lines indicate slight differences in the structure of many observations between the two dendrograms and display how the results of clustering will be different for the two methods. However, from the colouring of the tanglegram we see that choosing 5 clusters will result in identical clustering, despite which method we use. The scale underneath each dendrogram also displays the effect of cutting each dendrogram at a set height for clustering.

```
dendreducedaverage<-as.dendrogram(distancereducedhcaverage)
dendreducedward<-as.dendrogram(distancereducedhcward)
tanglegram(dendreducedaverage,dendreducedward,lab.cex =0.8,
edge.lwd=3,main_left="Average Linkage Dendrogram",
main_right="Ward Linkage Dendrogram",
common_subtrees_color_branches = TRUE)
```

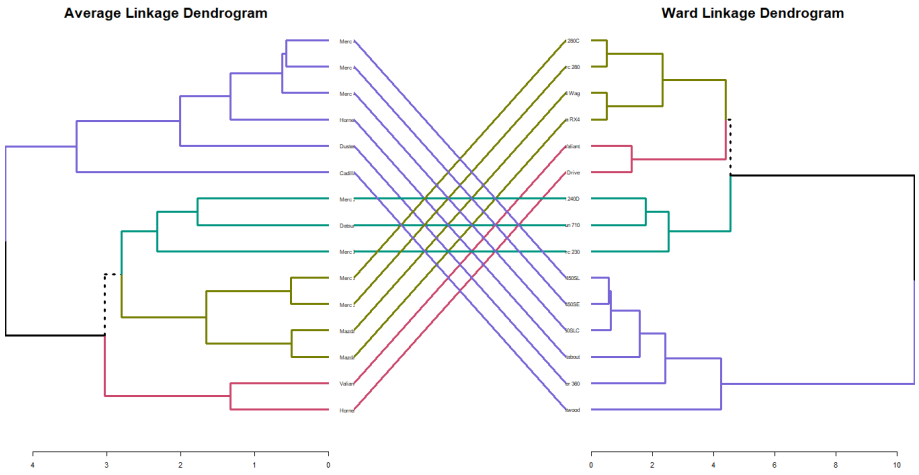


Figure 48: *The tanglegram showing the comparison between the dendrogram for the average linkage method and the Ward linkage method. These two methods appear much more entangled and therefore dissimilar.*

From the above comparison, we again compare the effects of selecting a certain number of clusters or cutting the tree at a certain height.

The *tanglegram* function provides a visual representation, but we may also use another command available through the *dendextend* package to measure the similarity of dendrograms, the *entanglement* function. This function takes, as its input, 2 dendrograms and returns a measure of dissimilarity (or entanglement) between the two dendrograms based on the $L - norm$ distances between the two vectors. The value returned by the function ranges from 1 (complete entanglement) to 0 (no entanglement). We examine the measure of entanglement on our tanglegrams:

```
entanglement(dendreducedsingle,dendreducedsingle)
[1] 0
```

Clearly the entanglement between two dendrograms created using the same method is equal to 0.

```
entanglement(dendreducedsingle,dendreducedcomplete)
[1] 0.2166252
```

The entanglement score between the single and complete linkage methods is approximately 0.22. This low score suggests that these two dendrograms are very similar.

```
entanglement(dendreducedaverage,dendreducedward)
[1] 0.9324923
```

The entanglement score between the average and Ward linkage methods is approximately 0.93. This high score suggests that these two dendrograms are very dissimilar.

10 Conclusion: Hierarchical Clustering

In this section, we examined hierarchical clustering techniques and applied these techniques to a subset of a dataset about car performance. We outlined the difference between the "bottom-up" agglomerative method and the "top-down" divisive method. We also discussed the linkage functions which underpin the techniques and determine the shapes of the dendrograms: tree-like structures which provide a visual representation of the clustering. The linkage functions we discussed were complete, single, average, centroid and Ward. Each of these linkage functions calculated the distance between objects and groups using different methods, hence the different results.

We then discussed the dendrogram produced by each of these linkage functions and calculated the cophenetic distance between each dendrogram and measured this value's correlation with the actual distance between points. We then assessed the effectiveness of hierarchical clustering using each method based on this value, which we desired to be close to 1. After calculating the correlation between the cophenetic distance and actual distance for each dendrogram, we decided that the single linkage method best represented the distances in our data.

After deciding which dendrogram to use, we cut the tree into a set number of clusters and discussed our findings. We found that, for some methods, an extreme outlier (namely the *Cadillac Fleetwood*) was designated to its own cluster and all other observations were grouped into another cluster. Therefore outliers will often skew the results of hierarchical clustering techniques and a statistician may need to consider how to treat outliers when performing hierarchical clustering.

We clustered the *carsreduced* dataset into 3 and then 5 clusters by cutting our preferred dendrogram into 3 and 5 subtrees. We discussed the results and plotted them against the first two principal components. Since these plots were reminiscent of the plots produced by using partitioning clustering, we compared the results from hierarchical clustering against the results from using the partitioning method. After this comparison, we performed this process of analysis on the entire $MTCars$ dataset and again compared the results with the clustering recommended by partitioning.

After this, we returned to the *carsreduced* dataset and examined the *tanglegram* and *entanglement* functions, both available through the *dendextend* package for *R*. The *tanglegram* function produces an illustration of how entangled two dendrograms are. We used this to compare two pairs of dendrograms (created using different linkage functions) and discussed their similarities and differences. Finally, to quantify the entanglement between pairs of dendrograms, we used the *entanglement* function . This function gave a measure of the dissimilarity between two dendrograms, this value being useful in comparing dendrograms from different hierarchical clustering measures.

References

[1] OECD, *OECD Glossary of Statistical Terms.* OECD, www.oecd.org/publishing/corrigenda, 2008, p. 78. Accessed March 2019.

[2] Brian Everitt, *Cluster Analysis: Second Edition.* Halsted Press, New York, 1980, p. 59. Computing in Geographic Information Systems

[3] Narayan Panigrahi, *Computing in Geographic Information Systems.* CRC Press, Boca Raton, FL, 2014, p. 212.

[4] R Documentation *Motor Trend Car Road Tests* from https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/mtcars.html, Accessed March 2020.

[5] CarGuru *Image of 1974 Mercedes 280* https://www.cargurus.com/Cars/1974-Mercedes-Benz-280-Pictures-c11004?picturesTabFilter= exteriorpictureId=35702230, Accessed March 2020.

[6] ArCar *Image of 1974 Mercedes 280 Coupe* https://www.arcar.org/video-5oxxkxrpmtm, Accessed March 2020.

[7] Pedigree Motors of the Palm Beaches *Image of 1974 Cadillac Fleetwood* https://www.pedigreemotorcars.com/vehicles /186/1974-cadillac-fleetwood-talisman, Accessed March 2020.

[8] WordDisk *Image of 1974 Datsun 710* https://worddisk.com/wiki/Nissan_Stanza/, Accessed March 2020.

[9] Bhattacharyya, Siddhartha, Bhaumik, Hrishikesh, De, Sourav, Klepac, Goran, *Intelligent Analysis of Multimedia Information.* IGI Global, Hershey, PA, 2017, p. 110.

[10] Charu C. Aggarwal, Chandan K. Reddy *Data Clustering: Algorithms and Applications.* CRC Press, Boca Raton, FL, 2014, p. 89.

[11] Google Developers *k-Means Advantages and Disadvantages* from https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages, Accessed March 2020.

[12] Charu C. Aggarwal, Chandan K. Reddy *Data Clustering: Algorithms and Applications.* CRC Press, Boca Raton, FL, 2014, p. 94.

[13] Boris Mirkin *Core Concepts in Data Analysis: Summarization, Correlation and Visualization.* Springer-Verlag, London, 2011, p. 245.

[14] Towards Data Science- Cory Maklin *Hierarchical Agglomerative Clustering Algorithm Example In Python* from https://towardsdatascience.com/machine-learning-algorithms-part-12-hierarchical-agglomerative-clustering-example-in-python-1e18e0075019, Accessed March 2020.

[15] Alboukadel Kassambara *Practical Guide To Cluster Analysis in R: Unsupervised Machine Learning.* STHDA, (http://www.sthda.com), 2017, p. 79.