Car Pricing: Partitioning and hierarchical clustering

Code

Loading the Data

Hide

```
Cars_df <- read.csv(
  file = "C:/Users/Suma Marri/Documents/GitHub/USCars/USA_cars_datasets.csv",
  colClasses = "character"
  )
Cars_df
```

X <chr>	price <chr>	brand <chr>	model <chr>	y... <chr>	title_status <chr>	mileage <chr>	color <chr>	vin <chr>
0	6300	toyota	cruiser	2008	clean vehicle	274117.0	black	jtezu11f88k00776
1	2899	ford	se	2011	clean vehicle	190552.0	silver	2fmdk3gc4bbb02
2	5350	dodge	mpv	2018	clean vehicle	39590.0	silver	3c4pdcgg5jt3464
3	25000	ford	door	2014	clean vehicle	64146.0	blue	1ftfw1et4efc2374
4	27700	chevrolet	1500	2018	clean vehicle	6654.0	red	3gcpcrec2jg4739
5	5700	dodge	mpv	2018	clean vehicle	45561.0	white	2c4rdgeg9jr2379
6	7300	chevrolet	pk	2010	clean vehicle	149050.0	black	1gcsksea1az1211
7	13350	gmc	door	2017	clean vehicle	23525.0	gray	1gks2gkc3hr3267
8	14600	chevrolet	malibu	2018	clean vehicle	9371.0	silver	1g1zd5st5jf19186
9	5250	ford	mpv	2017	clean vehicle	63418.0	black	2fmpk3j92hbc125

1-10 of 2,499 rows | 1-9 of 13 columns Previous 1 2 3 4 5 6 ... 100 Next

This US Cars Dataset data was scraped from AUCTION EXPORT.com. The dataset includes information about 28 brands of clean and used vehicles for sale in US.

Hide

```
price <- as.integer(Cars_df$price)
mileage <- as.double(Cars_df$mileage)
year <- as.factor(Cars_df$year)
d <- data.frame(year, mileage, price)
d
```

year	mileage	price
<fctr>	<dbl>	<int>
2008	274117	6300
2011	190552	2899
2018	39590	5350
2014	64146	25000
2018	6654	27700
2018	45561	5700
2010	149050	7300
2017	23525	13350
2018	9371	14600
2017	63418	5250

1-10 of 2,499 rows Previous **1** 2 3 4 5 6 ... 100 Next

Partitioning Unsupervised Learning

K-Means Clustering

Hide

```
#Scaling the data to remove influence caused by large variance
library(dplyr)
```

```


Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union
```

Hide

```
scaled_d <- d%>%mutate_if(is.numeric,scale)
scaled_d
```

year	mileage	price
<fctr>	<dbl>	<dbl>

year <fctr>	mileage <dbl>	price <dbl>
2008	3.715206367	-1.029017314
2011	2.315586950	-1.309718317
2018	-0.212856135	-1.107425416
2014	0.198429144	0.514384260
2018	-0.764496955	0.737228338
2018	-0.112848626	-1.078538220
2010	1.620475300	-0.946482471
2017	-0.481926750	-0.447146667
2018	-0.718990272	-0.343978113
2017	0.186235966	-1.115678900

1-10 of 2,499 rows Previous 1 2 3 4 5 6 ... 100 Next

Hide

```
#Dropping the character variable while clustering
#Performing k-means clustering with 2 clusters initially
kmeans_scaled_d_2 <- kmeans(
  x = scaled_d[-1],
  centers = 2
)
kmeans_scaled_d_2
```

```
K-means clustering with 2 clusters of sizes 1376, 1123

Cluster means:
     mileage      price
1  0.3377841 -0.6971274
2 -0.4138833  0.8541829

Clustering vector:
   [1] 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 2
1 1 2 1 2 1 1 2 1 2 1 2 1 2 1
  [61] 1 2 1 1 2 1 2 2 1 2 1 1 2 1 2 1 1 2 1 1 1 1 2 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 2 1 1 2 1 1 2
1 1 2 1 2 2 1 1 2 1 2 2 1 2 2
 [121] 1 2 2 2 2 1 2 2 1 1 2 1 2 1 2 2 1 2 2 1 1 1 2 1 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 1 1 2 2 1 2
2 1 2 2 1 2 2 1 1 2 1 1 2 1 2
 [181] 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1
2 1 1 1 1 1 1 1 1 1 1 1 2 1 1
 [241] 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 2 2 1 2 2 1 2 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [301] 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1
1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 [361] 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1
1 1 1 1 1 1 1 2 1 2 1 1 1 1 1
 [421] 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 2 1 1
1 1 1 1 2 1 2 1 1 1 2 2 1 1 1
 [481] 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 [541] 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 2 2 1 2 1 2 2 2 1
 [601] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 1 2 2 2 1 2 1 2 1 2 2 2 2 2 1
2 2 1 2 2 1 1 2 2 2 1 2 2 2 1
 [661] 2 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 2 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 1 1 1 1
2 2 1 2 2 1 2 2 2 1 2 1 1 2 2
 [721] 1 2 1 1 2 1 1 1 2 2 2 2 1 1 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 1 1 2
2 2 2 1 2 1 1 1 1 1 1 1 1 2 2
 [781] 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 2 2 1 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 1 1 2 2
1 1 2 2 2 2 2 1 1 1 1 1 2 1 1
 [841] 1 1 1 2 1 2 1 1 1 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 1 1 1 1 1 2 2 1
1 1 2 2 2 2 1 2 1 2 1 1 2 1 1
 [901] 1 1 1 2 2 1 1 1 2 1 2 2 2 1 2 2 2 2 2 1 2 1 2 2 1 2 1 1 1 2 2 2 2 1 1 1 1 1 1 1 2 2 1 2 2
1 2 2 1 2 1 1 1 1 1 2 1 1 1 2
 [961] 2 1 1 1 2 2 2 1 1 2 2 1 2 1 2 2 2 2 1 2 1 1 1 2 2 2 2 1 2 1 2 1 2 1 2 1 1 2 2 1
 [ reached getOption("max.print") -- omitted 1499 entries ]

Within cluster sum of squares by cluster:
[1] 2332.8242  825.7173
 (between_SS / total_SS =  36.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "s
ize"         "iter"
[9] "ifault"
```

At this point some sort of conclusion can be drawn by looking at the centers, which look somewhat similar. We can see a relation between low mileage and high price along with high mileage with low price. Next, we can try increasing the number of clusters.

Hide

```
#Performing k-means clustering with 3 clusters
kmeans_scaled_d_3 <- kmeans(
  x = scaled_d[-1],
  centers = 3
)
kmeans_scaled_d_3
```

```
K-means clustering with 3 clusters of sizes 334, 863, 1302

Cluster means:
     mileage      price
1  1.7440687 -1.0813259
2 -0.4216979  1.0825755
3 -0.1678906 -0.4401688

Clustering vector:
   [1] 1 1 3 2 2 3 1 3 3 3 1 3 2 1 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 1 3 3 2 3 2
3 3 3 3 2 3 3 2 3 3 3 3 3 2 3
  [61] 3 3 3 3 2 3 3 3 3 3 3 3 2 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 1 3 2 1 2 2 3 2 2 3 3 2 3 3 2
3 3 3 3 3 2 3 3 2 3 2 3 3 2 2
 [121] 3 2 2 2 2 3 2 2 3 3 2 3 3 3 2 2 3 3 2 3 3 1 2 3 1 2 3 2 3 1 3 3 3 2 3 3 2 3 3 3 3 3 2 3 2
2 3 3 3 3 2 2 3 3 2 3 3 2 3 2
 [181] 2 3 1 2 3 1 3 3 1 2 3 1 3 3 1 2 3 1 2 1 1 2 3 1 3 1 3 3 1 1 2 3 3 3 1 1 2 3 3 2 3 3 3 3 3
2 3 1 3 3 1 1 3 1 3 3 3 2 3 1
 [241] 2 3 1 2 3 1 3 3 2 3 3 1 3 3 1 2 3 1 1 3 1 2 1 1 1 3 3 3 3 2 3 3 2 3 3 1 3 2 3 3 2 3 3 2 1
1 3 3 1 3 1 3 3 1 3 3 3 1 3 3
 [301] 3 1 3 3 3 1 3 3 3 3 1 1 3 1 1 3 3 3 1 1 1 2 3 1 3 3 3 1 1 2 1 1 3 3 3 1 2 1 2 1 1 3 3 3 1
1 3 1 3 3 3 1 1 3 1 1 1 3 3 1
 [361] 1 2 1 1 3 3 3 1 1 2 1 1 3 3 1 3 3 2 1 3 2 1 3 1 3 2 1 1 3 3 1 1 3 2 1 1 3 1 1 2 1 3 3 2 1
3 1 3 1 1 1 3 2 3 2 3 1 1 1 1
 [421] 1 1 1 3 3 2 2 3 3 3 1 3 3 2 3 1 1 3 3 2 3 3 3 1 2 1 2 1 1 3 3 2 2 3 3 1 1 2 3 1 3 3 2 3 3
3 1 3 3 2 1 3 1 3 3 3 2 3 3 1
 [481] 1 2 1 2 3 3 3 2 2 3 1 3 1 1 1 2 1 2 3 3 3 2 2 3 3 3 1 1 3 2 1 3 3 3 1 3 1 3 1 1 1 2 1 3 3
3 3 3 1 3 1 1 1 2 1 3 3 3 3 3
 [541] 3 1 1 3 2 3 2 3 3 1 3 1 3 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 1 2 3 2 3 2 1 2 3 2
3 2 3 2 3 2 2 2 3 2 3 2 2 2 3
 [601] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 3 3 3 2 2 3 3 2 3 3 3 2 3 2 3 3 3 2 2 2 3 2 2 3
2 2 3 2 2 3 3 2 2 3 3 2 2 2 3
 [661] 2 2 3 2 2 3 3 2 2 3 2 2 2 2 2 2 2 3 2 3 1 3 3 3 1 2 3 3 3 3 3 3 3 3 3 3 3 1 2 2 2 1 3 1 1
2 3 1 3 2 3 2 2 2 3 2 3 3 3 2
 [721] 3 2 3 3 3 3 3 3 2 3 2 2 3 1 2 2 3 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 3 1 3 3 3 3 1 2 3 3 3 1 2
2 2 2 3 2 1 3 3 1 1 1 3 3 2 2
 [781] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 1 2 2 3 2 3 2 2 1 3 3 2 2 2 2 2 2 3 2 3 1 2 2
3 3 2 3 3 2 2 3 3 3 3 3 3 3 3
 [841] 3 3 3 3 3 2 3 3 1 3 2 2 2 1 3 2 1 2 2 2 3 3 3 2 3 3 2 1 3 2 2 3 2 2 3 2 3 3 3 3 1 3 2 2 3
3 3 3 2 2 2 3 3 1 2 3 3 2 3 3
 [901] 3 1 1 2 3 3 3 1 2 1 2 3 3 3 3 2 2 2 2 3 2 3 2 2 1 2 3 3 3 3 3 3 2 3 3 3 3 3 3 1 2 3 1 3 3
3 2 3 3 3 1 3 3 3 3 2 3 3 3 2
 [961] 2 3 3 1 2 3 3 3 1 3 2 3 3 3 2 2 2 2 3 2 3 1 1 3 3 3 3 3 3 1 3 1 3 3 3 1 3 2 2 3
 [ reached getOption("max.print") -- omitted 1499 entries ]

Within cluster sum of squares by cluster:
[1] 1087.0463  600.5236  448.1056
 (between_SS / total_SS =  57.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "s
```

```
ize"         "iter"
[9] "ifault"
```

Increasing the number of clusters does not improve the centers for clustering. We will stick with the initial 2 clusters.

Analysis of Optimal number of clusters

The difference between the centers using 2 and 3 clusters is visible, however, we don't know the optimal number of clusters. For this, we can visualize the results using fviz_cluster from the factoextra package in R. This uses Principal Component analysis and plot the data points according to the first two principal components.

Hide

```
library(factoextra)
```

```
Loading required package: ggplot2
Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

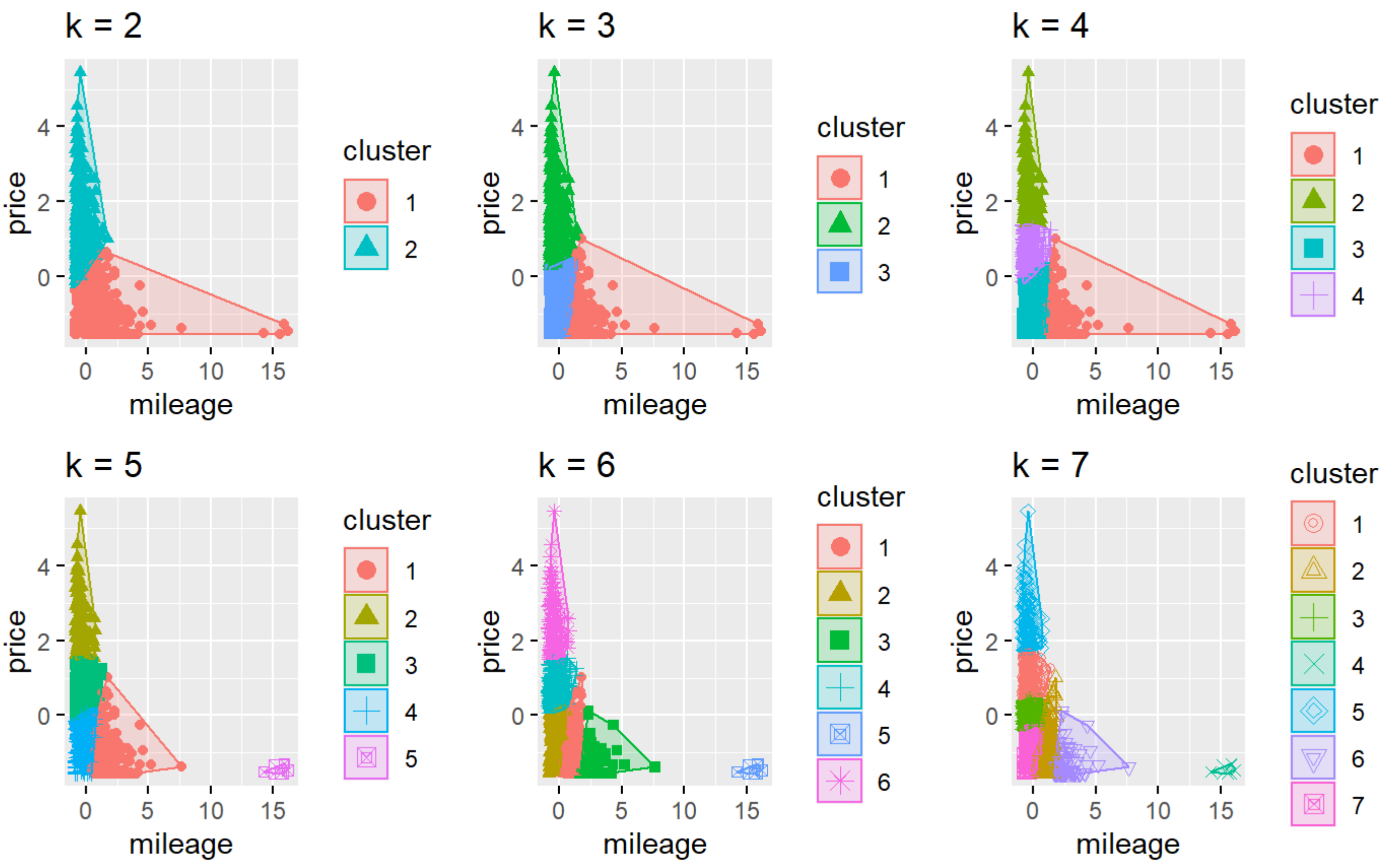
Hide

```
kmeans_scaled_d_4 <- kmeans(scaled_d[-1], centers = 4)
kmeans_scaled_d_5 <- kmeans(scaled_d[-1], centers = 5)
kmeans_scaled_d_6 <- kmeans(scaled_d[-1], centers = 6)
kmeans_scaled_d_7 <- kmeans(scaled_d[-1], centers = 7)

#Plots for comparison
p1 <- fviz_cluster(kmeans_scaled_d_2, geom = "point", data = scaled_d[-1]) + ggtitle("k = 2")
p2 <- fviz_cluster(kmeans_scaled_d_3, geom = "point",  data = scaled_d[-1]) + ggtitle("k = 3")
p3 <- fviz_cluster(kmeans_scaled_d_4, geom = "point",  data = scaled_d[-1]) + ggtitle("k = 4")
p4 <- fviz_cluster(kmeans_scaled_d_5, geom = "point",  data = scaled_d[-1]) + ggtitle("k = 5")
p5 <- fviz_cluster(kmeans_scaled_d_6, geom = "point",  data = scaled_d[-1]) + ggtitle("k = 6")
p6 <- fviz_cluster(kmeans_scaled_d_7, geom = "point",  data = scaled_d[-1]) + ggtitle("k = 7")

library(gridExtra)
```

```


Attaching package: ‘gridExtra’

The following object is masked from ‘package:dplyr’:

    combine
```

Hide

```
grid.arrange(p1, p2, p3, p4, p5, p6, nrow = 2)
```

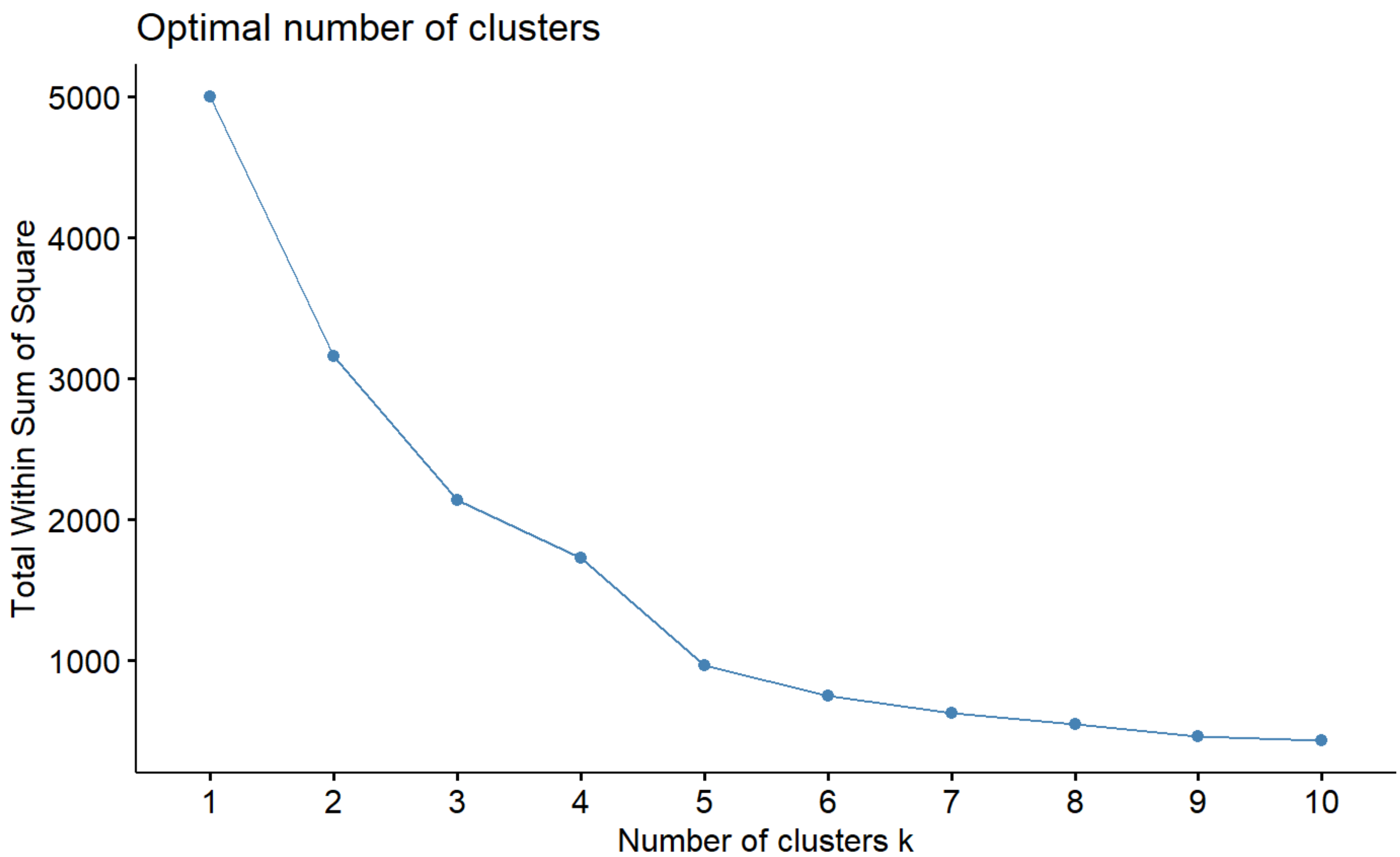


From the plot above, we can see that k=2 clearly differentiates between high priced low mileage cars and low priced high mileage cars. However, it does not group the low priced low mileage cars. This can be caused by various reasons but one of the main reasons is the cars' value being depreciated as the years pass and the car is an older year model. Overall, k=3 is better than k=4 or k=5, since it takes in to account high priced low mileage, low priced high mileage, and low price low mileage. The data points far to the right make more sense to be in the high mileage cluster in k=3.

Next, we can determine the optimum number of clusters using the Elbow method.

Hide

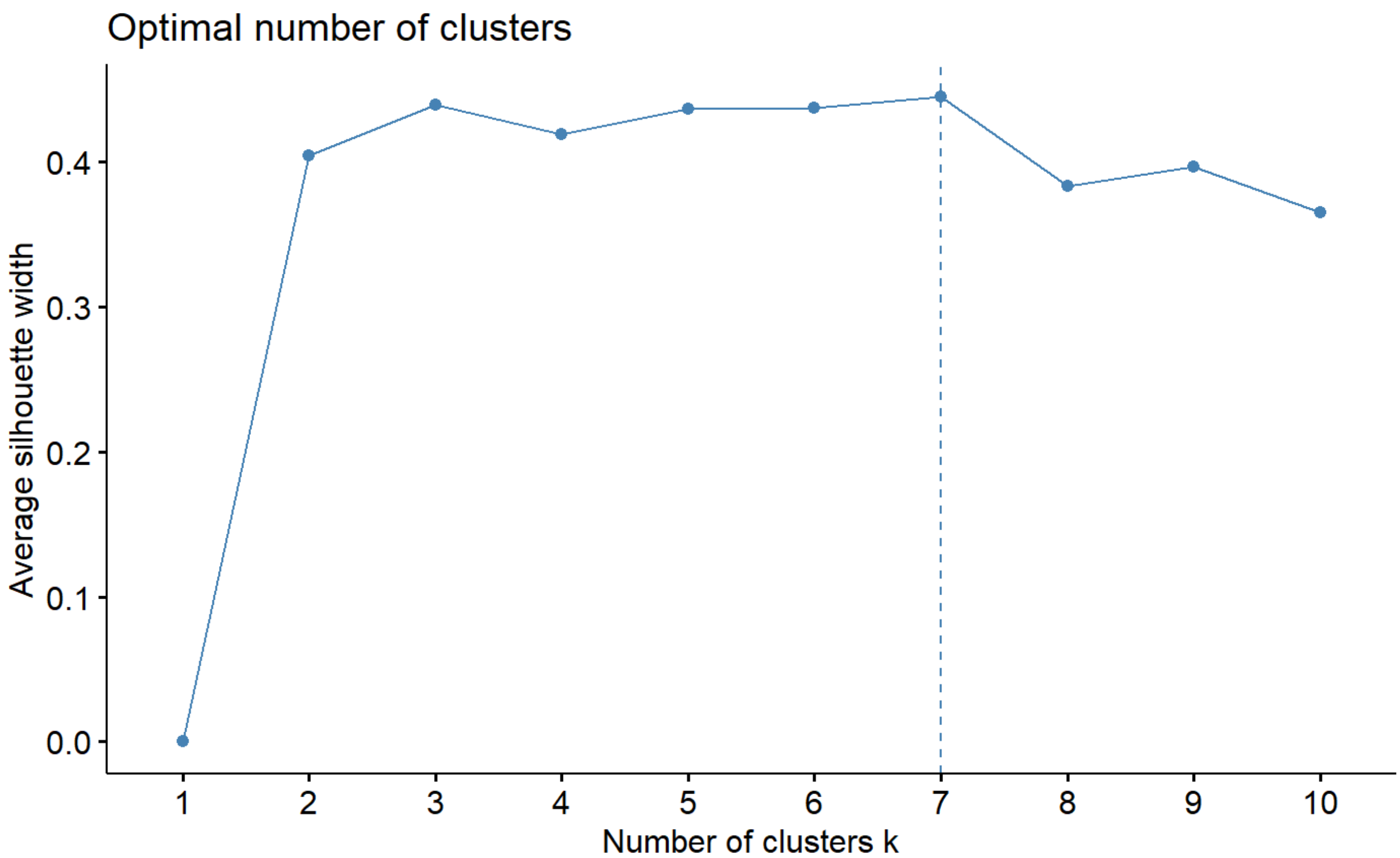
```
set.seed(1)
factoextra::fviz_nbclust(
  x = scaled_d[-1],
  FUNcluster = kmeans,
  method = "wss")
```



Next, we can determine the optimum number of clusters using the Silhouette method.

Hide

```
set.seed(1)
factoextra::fviz_nbclust(
  x = scaled_d[-1],
  FUNcluster = kmeans,
  method = "silhouette")
```



Next, we can determine the optimum number of clusters using the Gap Statistic method.

Hide

```
set.seed(1)
clusGap_kmeans <- cluster::clusGap(
  x = scaled_d[-1],
  FUNcluster = kmeans,
  K.max = 12)
```

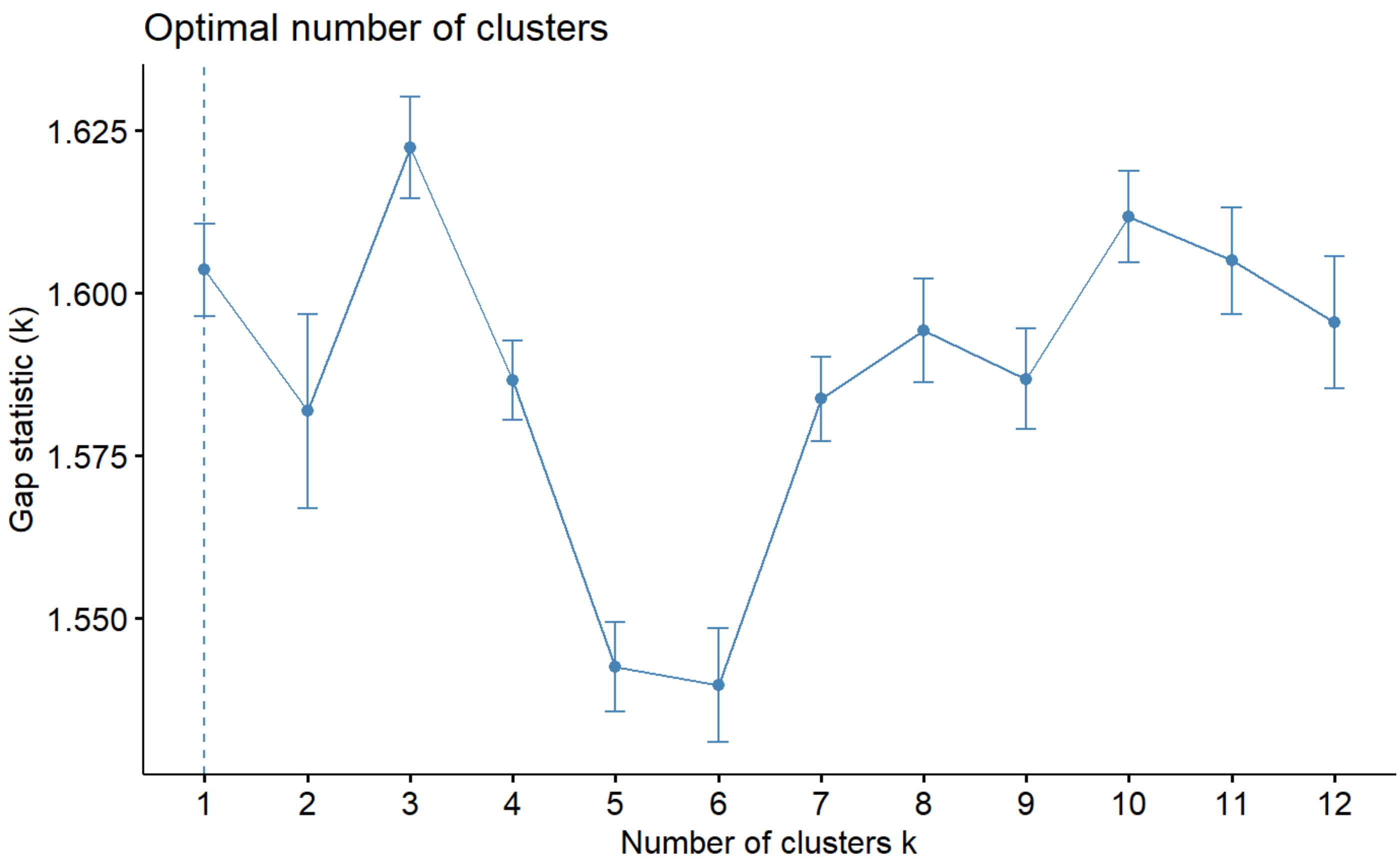
```
Clustering k = 1,2,..., K.max (= 12): .. done
Bootstrapping, b = 1,2,..., B (= 100)  [one "." per sample]:
.......
```

```
Warning: did not converge in 10 iterations
```

```
........................................... 50
.................................................. 100
```

Hide

```
fviz_gap_stat(clusGap_kmeans)
```



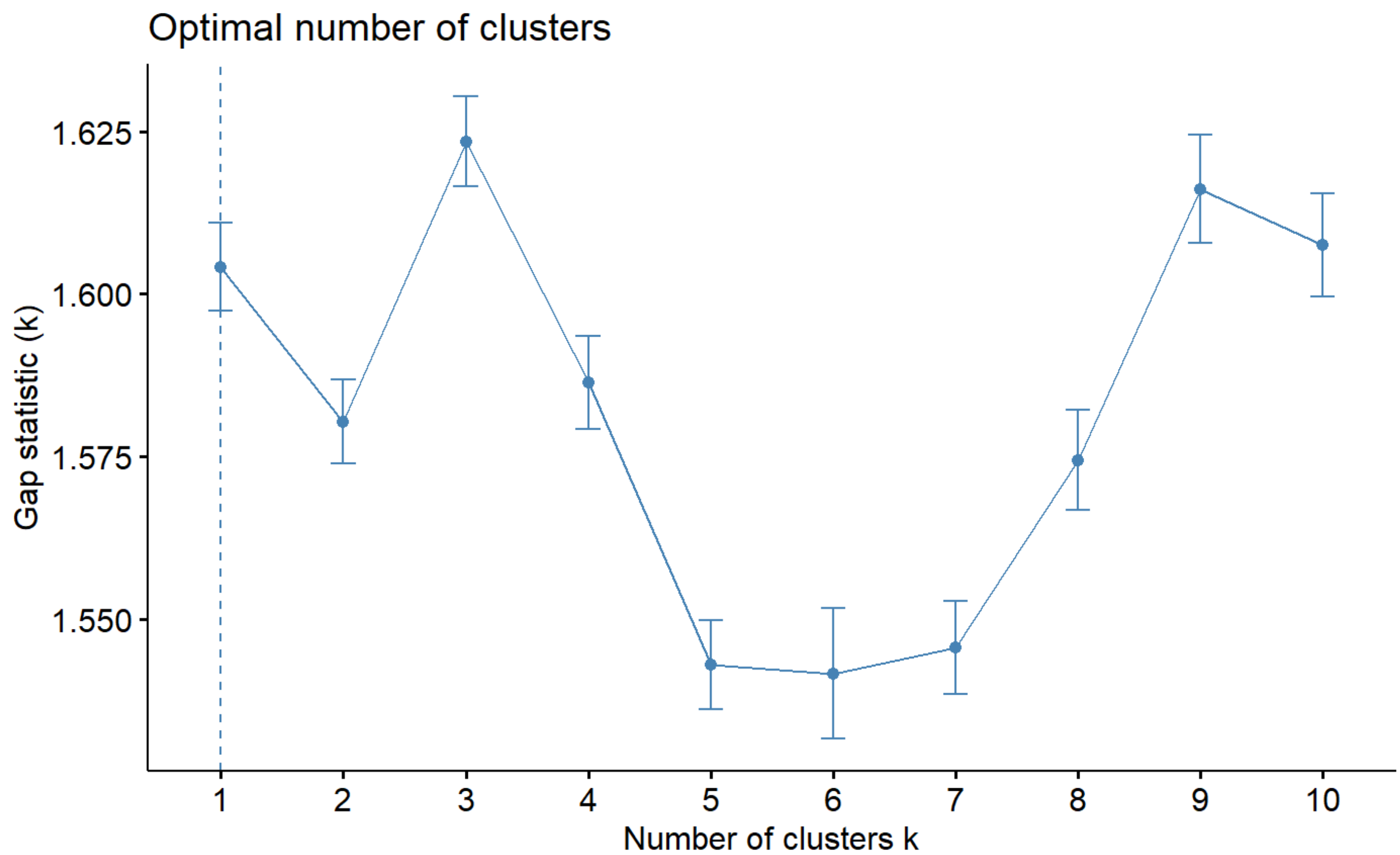
Hide

```
set.seed(1)
factoextra::fviz_nbclust(
  x = scaled_d[-1],
  FUNcluster = kmeans,
  method = "gap_stat")
```

```
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 100)  [one "." per sample]:
.................................................. 50
..........
```

```
Warning: did not converge in 10 iterations
```

```
....................................... 100
```



When looking for diminishing returns to determine optimum number of clusters, the Silhoutte method suggests optimal clusters to be k=7, the centers looked similar when knmeans clustering was performed using k=2, and k=3 made the most sense in our analysis.

Final Analysis of K-means clustering

For the final analysis we can compare the results/centers for k=2, k=3, and k=7.

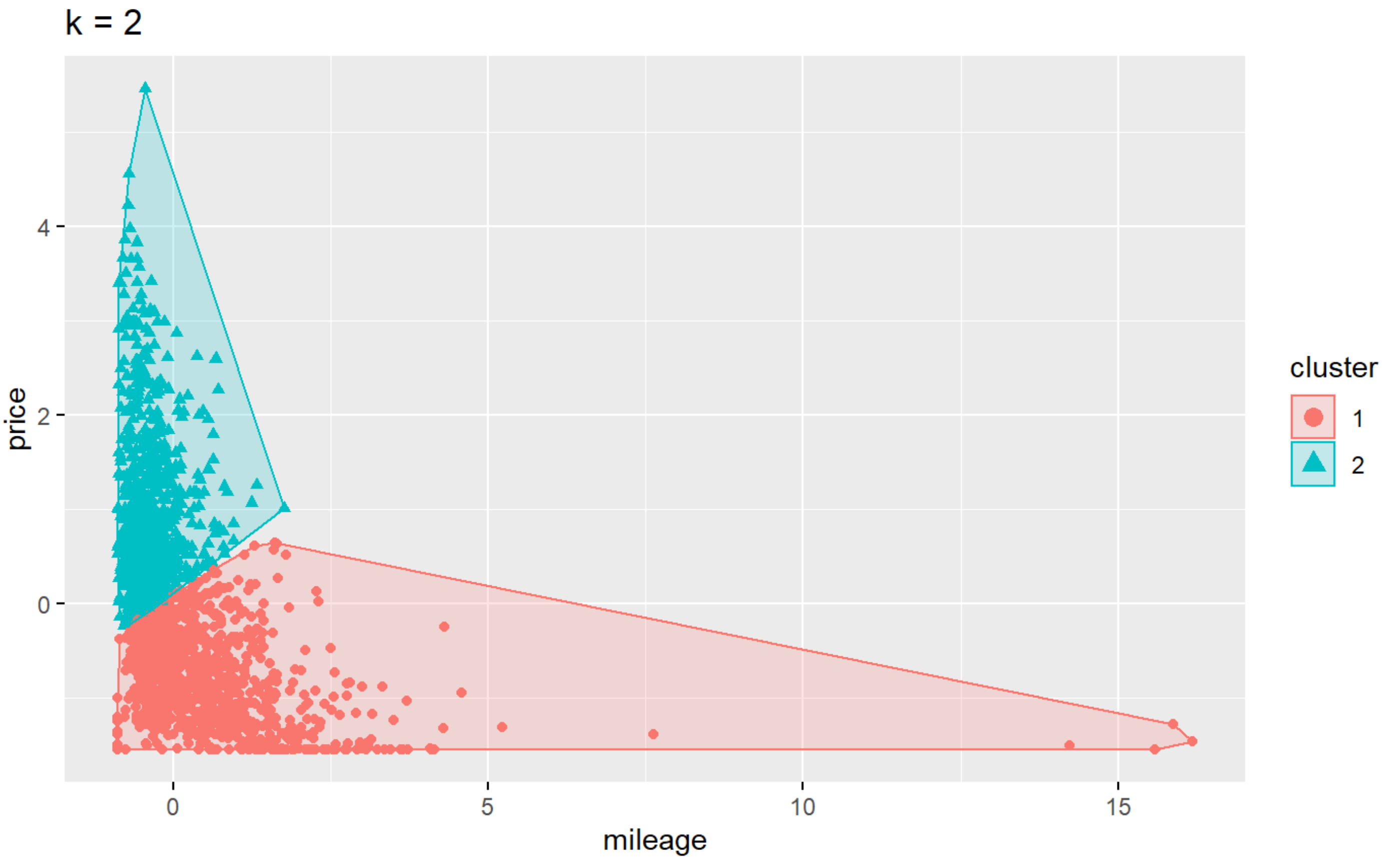
Hide

```
kmeans_scaled_d_2
```

```
K-means clustering with 2 clusters of sizes 1376, 1123

Cluster means:
     mileage      price
1  0.3377841 -0.6971274
2 -0.4138833  0.8541829

Clustering vector:
   [1] 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 2
1 1 2 1 2 1 1 2 1 2 1 2 1 2 1
  [61] 1 2 1 1 2 1 2 2 1 2 1 1 2 1 2 1 1 2 1 1 1 1 2 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 2 1 1 2 1 1 2
1 1 2 1 2 2 1 1 2 1 2 2 1 2 2
 [121] 1 2 2 2 2 1 2 2 1 1 2 1 2 1 2 2 1 2 2 1 1 1 2 1 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 1 1 2 2 1 2
2 1 2 2 1 2 2 1 1 2 1 1 2 1 2
 [181] 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1
2 1 1 1 1 1 1 1 1 1 1 1 2 1 1
 [241] 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 2 2 1 2 2 1 2 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [301] 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1
1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 [361] 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1
1 1 1 1 1 1 1 2 1 2 1 1 1 1 1
 [421] 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 2 1 1
1 1 1 1 2 1 2 1 1 1 2 2 1 1 1
 [481] 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 [541] 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 2 2 1 2 1 2 2 2 1
 [601] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 1 2 2 2 1 2 1 2 1 2 2 2 2 2 2 1
2 2 1 2 2 1 1 2 2 2 1 2 2 2 1
 [661] 2 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 2 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 1 1 1 1
2 2 1 2 2 1 2 2 2 1 2 1 1 2 2
 [721] 1 2 1 1 2 1 1 1 2 2 2 2 1 1 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 1 1 2
2 2 2 1 2 1 1 1 1 1 1 1 1 2 2
 [781] 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 1 1 2 2
1 1 2 2 2 2 2 1 1 1 1 1 2 1 1
 [841] 1 1 1 2 1 2 1 1 1 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 1 1 1 1 1 2 2 1
1 1 2 2 2 2 1 2 1 2 1 1 2 1 1
 [901] 1 1 1 2 2 1 1 1 2 1 2 2 2 1 2 2 2 2 2 1 2 1 2 2 1 2 1 1 1 2 2 2 2 1 1 1 1 1 1 1 2 2 1 2 2
1 2 2 1 2 1 1 1 1 1 2 1 1 1 2
 [961] 2 1 1 1 2 2 2 1 1 2 2 1 2 1 2 2 2 2 1 2 1 1 1 2 2 2 2 1 2 1 2 1 2 1 2 1 1 2 2 1
 [ reached getOption("max.print") -- omitted 1499 entries ]

Within cluster sum of squares by cluster:
[1] 2332.8242  825.7173
 (between_SS / total_SS =  36.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "s
ize"         "iter"
[9] "ifault"
```

Hide

```
fviz_cluster(kmeans_scaled_d_2, geom = "point", data = scaled_d[-1]) + ggtitle("k = 2")
```



Hide

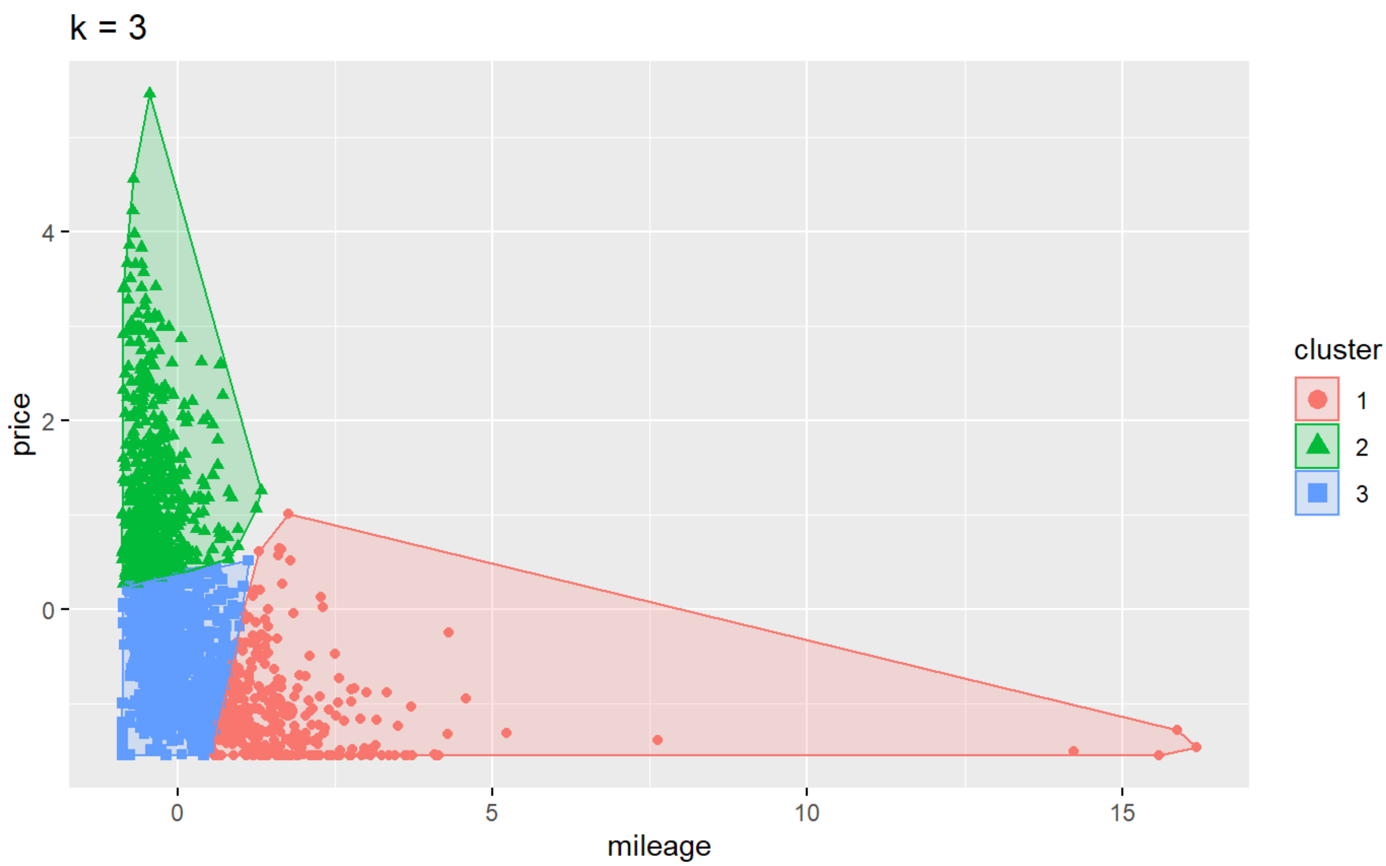
```
kmeans_scaled_d_3
```

```
K-means clustering with 3 clusters of sizes 334, 863, 1302

Cluster means:
     mileage      price
1  1.7440687 -1.0813259
2 -0.4216979  1.0825755
3 -0.1678906 -0.4401688

Clustering vector:
   [1] 1 1 3 2 2 3 1 3 3 3 1 3 2 1 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 1 3 3 2 3 2
3 3 3 3 2 3 3 2 3 3 3 3 3 2 3
  [61] 3 3 3 3 2 3 3 3 3 3 3 3 2 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 1 3 2 1 2 2 3 2 2 3 3 2 3 3 2
3 3 3 3 3 2 3 3 2 3 2 3 3 2 2
 [121] 3 2 2 2 2 3 2 2 3 3 2 3 3 3 2 2 3 3 2 3 3 1 2 3 1 2 3 2 3 1 3 3 3 2 3 3 2 3 3 3 3 3 2 3 2
2 3 3 3 3 2 2 3 3 2 3 3 2 3 2
 [181] 2 3 1 2 3 1 3 3 1 2 3 1 3 3 1 2 3 1 2 1 1 2 3 1 3 1 3 3 1 1 2 3 3 3 1 1 2 3 3 2 3 3 3 3 3
2 3 1 3 3 1 1 3 1 3 3 3 2 3 1
 [241] 2 3 1 2 3 1 3 3 2 3 3 1 3 3 1 2 3 1 1 3 1 2 1 1 1 3 3 3 3 2 3 3 2 3 3 1 3 2 3 3 2 3 3 2 1
1 3 3 1 3 1 3 3 1 3 3 3 1 3 3
 [301] 3 1 3 3 3 1 3 3 3 3 1 1 3 1 1 3 3 3 1 1 1 2 3 1 3 3 3 1 1 2 1 1 3 3 3 1 2 1 2 1 1 3 3 3 1
1 3 1 3 3 3 1 1 3 1 1 1 3 3 1
 [361] 1 2 1 1 3 3 3 1 1 2 1 1 3 3 1 3 3 2 1 3 2 1 3 1 3 2 1 1 3 3 1 1 3 2 1 1 3 1 1 2 1 3 3 2 1
3 1 3 1 1 1 3 2 3 2 3 1 1 1 1
 [421] 1 1 1 3 3 2 2 3 3 3 1 3 3 2 3 1 1 3 3 2 3 3 3 1 2 1 2 1 1 3 3 2 2 3 3 1 1 2 3 1 3 3 2 3 3
3 1 3 3 2 1 3 1 3 3 3 2 3 3 1
 [481] 1 2 1 2 3 3 3 2 2 3 1 3 1 1 1 2 1 2 3 3 3 2 2 3 3 3 1 1 3 2 1 3 3 3 1 3 1 3 1 1 1 2 1 3 3
3 3 3 1 3 1 1 1 2 1 3 3 3 3 3
 [541] 3 1 1 3 2 3 2 3 3 1 3 1 3 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 1 2 3 2 3 2 1 2 3 2
3 2 3 2 3 2 2 2 3 2 3 2 2 2 3
 [601] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 3 3 3 2 2 3 3 2 3 3 3 2 3 2 3 3 3 2 2 2 3 2 2 3
2 2 3 2 2 3 3 2 2 3 3 2 2 2 3
 [661] 2 2 3 2 2 3 3 2 2 3 2 2 2 2 2 2 2 3 2 3 1 3 3 3 1 2 3 3 3 3 3 3 3 3 3 3 3 1 2 2 2 1 3 1 1
2 3 1 3 2 3 2 2 2 3 2 3 3 3 2
 [721] 3 2 3 3 3 3 3 3 2 3 2 2 3 1 2 2 3 2 2 2 3 3 2 2 2 2 2 2 2 2 2 3 1 3 3 3 3 1 2 3 3 3 1 2
2 2 2 3 2 1 3 3 1 1 1 3 3 2 2
 [781] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 1 2 2 3 2 3 2 2 1 3 3 2 2 2 2 2 2 3 2 3 1 2 2
3 3 2 3 3 2 2 3 3 3 3 3 3 3 3
 [841] 3 3 3 3 3 2 3 3 1 3 2 2 2 1 3 2 1 2 2 2 3 3 3 2 3 3 2 1 3 2 2 3 2 2 3 2 3 3 3 3 1 3 2 2 3
3 3 3 2 2 2 3 3 1 2 3 3 2 3 3
 [901] 3 1 1 2 3 3 3 1 2 1 2 3 3 3 3 2 2 2 2 3 2 3 2 2 1 2 3 3 3 3 3 3 2 3 3 3 3 3 3 1 2 3 1 3 3
3 2 3 3 3 1 3 3 3 3 2 3 3 3 2
 [961] 2 3 3 1 2 3 3 3 1 3 2 3 3 3 2 2 2 2 3 2 3 1 1 3 3 3 3 3 3 1 3 1 3 3 3 1 3 2 2 3
 [ reached getOption("max.print") -- omitted 1499 entries ]

Within cluster sum of squares by cluster:
[1] 1087.0463  600.5236  448.1056
 (between_SS / total_SS =  57.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "s
```

```
ize"          "iter"
[9] "ifault"
```

Hide

```
fviz_cluster(kmeans_scaled_d_3, geom = "point", data = scaled_d[-1]) + ggtitle("k = 3")
```



Hide

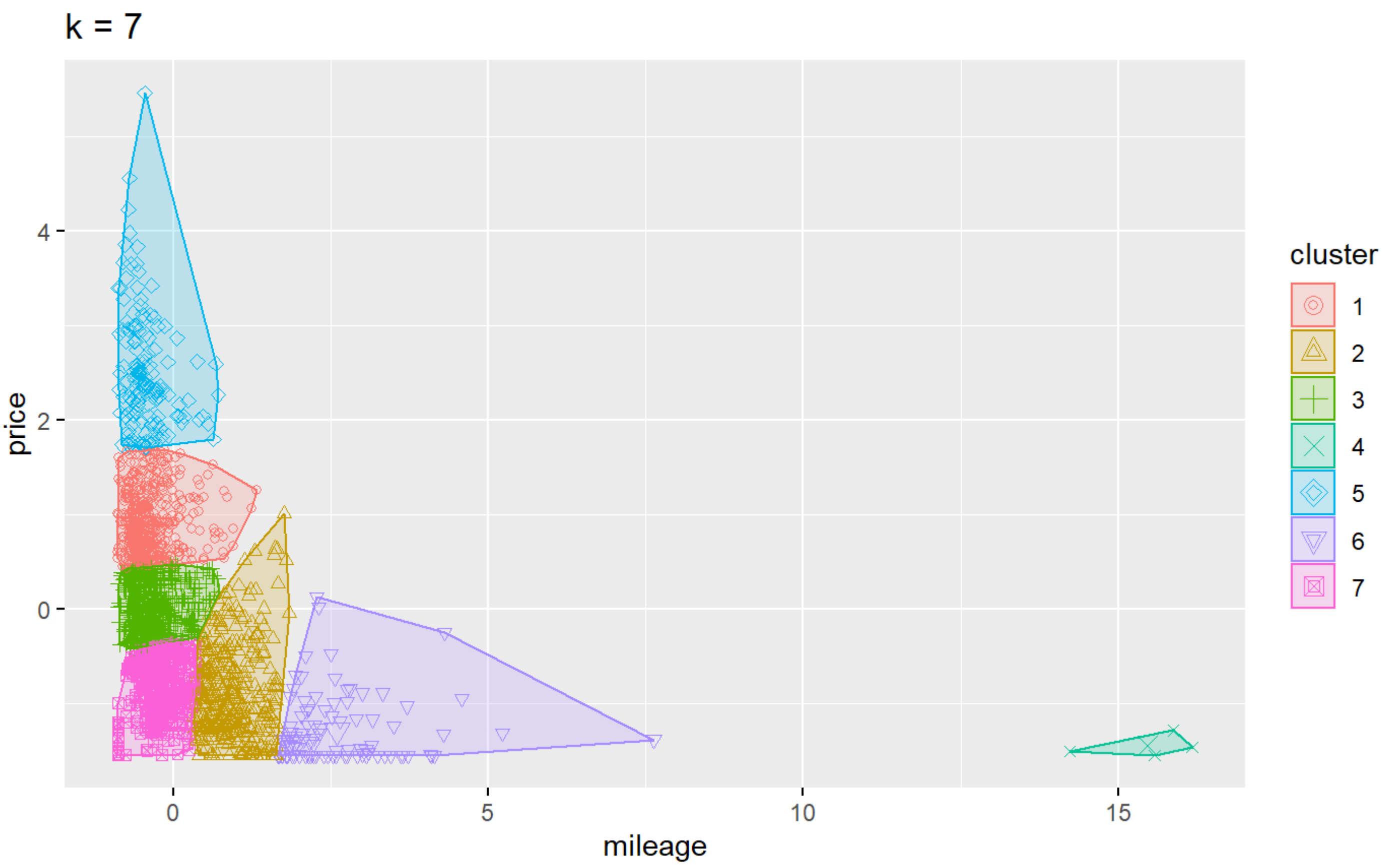
```
kmeans_scaled_d_7
```

```
K-means clustering with 7 clusters of sizes 566, 376, 770, 4, 150, 100, 533

Cluster means:
     mileage        price
1 -0.3886627  0.894231192
2  0.9190947 -0.885981577
3 -0.3362276  0.007097886
4 15.4645144 -1.452008387
5 -0.4468455  2.495725619
6  2.5967642 -1.288173422
7 -0.2274089 -0.784624055

Clustering vector:
   [1] 6 6 7 1 1 7 2 7 3 7 2 7 1 2 2 7 7 3 7 1 7 7 3 7 2 7 3 7 7 2 7 3 1 7 7 7 2 3 7 6 7 2 1 7 5
7 3 3 7 5 7 3 1 7 3 7 3 3 3 3
  [61] 7 3 3 7 1 7 3 3 7 3 7 3 1 7 3 7 3 1 7 3 7 3 3 7 3 7 3 3 7 7 2 2 1 2 5 5 3 1 1 7 3 3 7 3 1
7 7 3 7 3 1 7 3 1 7 1 3 7 1 1
 [121] 7 1 5 3 3 2 1 5 7 7 1 7 3 7 1 3 7 3 1 7 7 6 1 7 2 1 7 1 7 6 2 7 3 1 7 3 1 7 3 3 7 3 1 7 1
5 7 3 3 7 1 1 7 2 1 3 2 1 3 1
 [181] 1 7 6 1 7 6 3 7 6 1 3 2 3 7 2 1 2 6 1 2 2 1 7 6 7 2 7 7 6 6 3 7 7 3 2 2 5 7 3 5 7 3 3 7 3
5 7 2 7 7 6 6 7 2 2 7 2 5 2 2
 [241] 1 7 6 5 7 6 3 2 1 7 2 2 3 7 6 1 7 2 2 2 2 1 2 2 2 3 3 7 7 3 7 7 3 7 7 6 7 5 3 7 1 3 7 1 6
2 7 7 2 7 2 7 2 6 7 7 7 6 7 7
 [301] 7 6 7 3 2 6 7 7 7 7 6 6 3 6 2 7 7 7 2 2 2 3 7 6 7 7 3 2 6 1 6 6 7 7 7 2 1 2 1 6 6 7 7 7 2
2 3 6 7 7 7 6 2 3 6 6 2 7 7 2
 [361] 2 5 6 6 7 7 7 6 2 5 2 2 7 7 2 7 7 5 6 3 5 6 7 6 2 5 2 6 2 7 6 2 2 5 6 6 2 2 6 5 2 7 2 5 2
7 2 2 2 6 6 2 5 7 5 7 2 2 6 2
 [421] 6 6 2 3 7 1 5 7 7 3 2 3 2 1 7 6 2 3 7 5 7 7 7 6 1 2 1 6 2 3 7 3 5 7 7 6 2 1 7 2 3 7 3 2 7
7 6 3 2 3 6 3 2 7 7 3 1 7 7 2
 [481] 2 1 2 5 2 7 7 1 1 7 4 7 6 2 2 1 6 1 7 7 7 1 5 7 7 7 2 2 7 1 2 7 7 7 2 7 4 7 6 6 2 1 2 7 7
7 3 7 4 7 6 6 6 1 6 7 7 7 3 2
 [541] 7 2 2 7 1 7 1 7 7 2 3 2 7 7 1 7 1 2 1 2 1 2 1 7 1 7 1 7 1 7 1 3 1 2 1 2 1 2 1 2 1 2 1 7 1
7 1 3 1 2 1 1 1 7 1 3 1 1 1 7
 [601] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 3 3 3 7 1 3 7 3 3 3 3 3 1 3 1 7 3 7 1 1 1 3 1 1 7
5 5 7 1 1 3 7 5 3 3 2 1 3 1 3
 [661] 1 1 7 1 1 3 3 5 1 3 1 1 1 1 3 3 3 7 1 3 2 2 2 3 2 1 7 3 7 7 3 7 3 7 3 3 3 2 1 1 1 2 2 2 6
5 3 2 3 1 3 1 1 1 2 1 7 2 3 1
 [721] 3 1 7 7 3 2 2 2 1 3 1 1 7 2 1 1 3 1 1 1 3 3 1 1 1 1 1 1 3 1 1 1 2 2 2 2 2 2 2 1 2 7 2 2 3
3 3 3 7 5 2 3 3 2 2 2 2 3 1 1
 [781] 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 7 2 2 2 2 3 2 1 1 2 1 3 3 3 2 3 3 1 1 1 1 1 1 3 3 3 6 1 3
3 7 1 3 3 5 5 7 3 7 7 2 3 2 2
 [841] 7 2 2 3 2 1 2 7 2 3 1 1 1 2 3 3 2 1 3 1 3 3 3 3 7 3 5 2 3 3 5 3 1 1 3 1 3 3 7 3 2 2 1 3 3
3 3 3 3 3 1 3 3 2 1 3 7 1 3 7
 [901] 2 2 6 1 3 3 7 2 1 2 1 3 3 2 3 3 1 1 1 2 3 2 3 3 2 1 7 3 3 3 3 3 1 7 7 3 7 7 3 2 1 3 2 3 3
3 3 3 3 3 6 2 3 3 7 3 3 3 3 1
 [961] 3 3 3 2 1 3 3 3 2 3 1 7 3 7 1 1 1 1 2 3 7 2 6 3 3 3 3 2 3 2 3 2 3 7 3 2 3 1 3 3
 [ reached getOption("max.print") -- omitted 1499 entries ]

Within cluster sum of squares by cluster:
[1] 122.648271 136.812597 117.828621   2.241823  73.588733  91.962718  87.722712
 (between_SS / total_SS =  87.3 %)
```

```
Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "s
ize"         "iter"
[9] "ifault"
```

Hide

```
fviz_cluster(kmeans_scaled_d_7, geom = "point", data = scaled_d[-1]) + ggtitle("k = 7")
```



Even though k=7, as suggested from the silhoutte plot, does a substantial job of clustering data points separately, k=2 would make more sense for the purpose of the analysis. This is because the comparison is between high priced low mileage vehicles and low priced high mileage vehicles. Looking at the centers, they are more similar for k=2, however k=3 once again, also takes in to account low mileage low price vehicles which are older. For a more in depth analysis that would also take in to account the year and how old or new a car is, k=7 would be a more suitable choice which would factor in the conditions of the car, the brand of the car, and the state it is being sold in. Hence, for the purpose of this analysis k=2 would be the best choice.

Cluster package in R

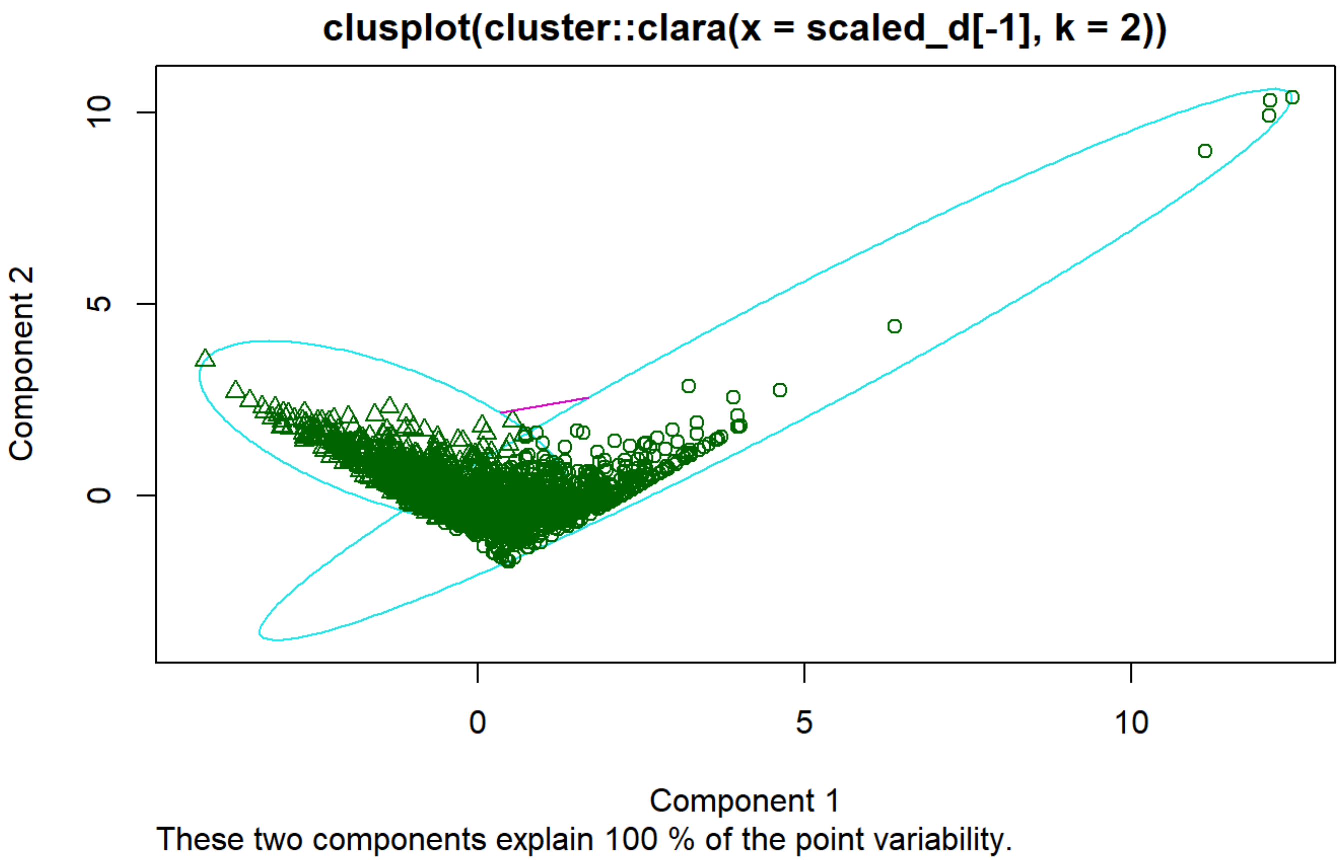
For the next part of the analysis of the problem, k=2 will be used.

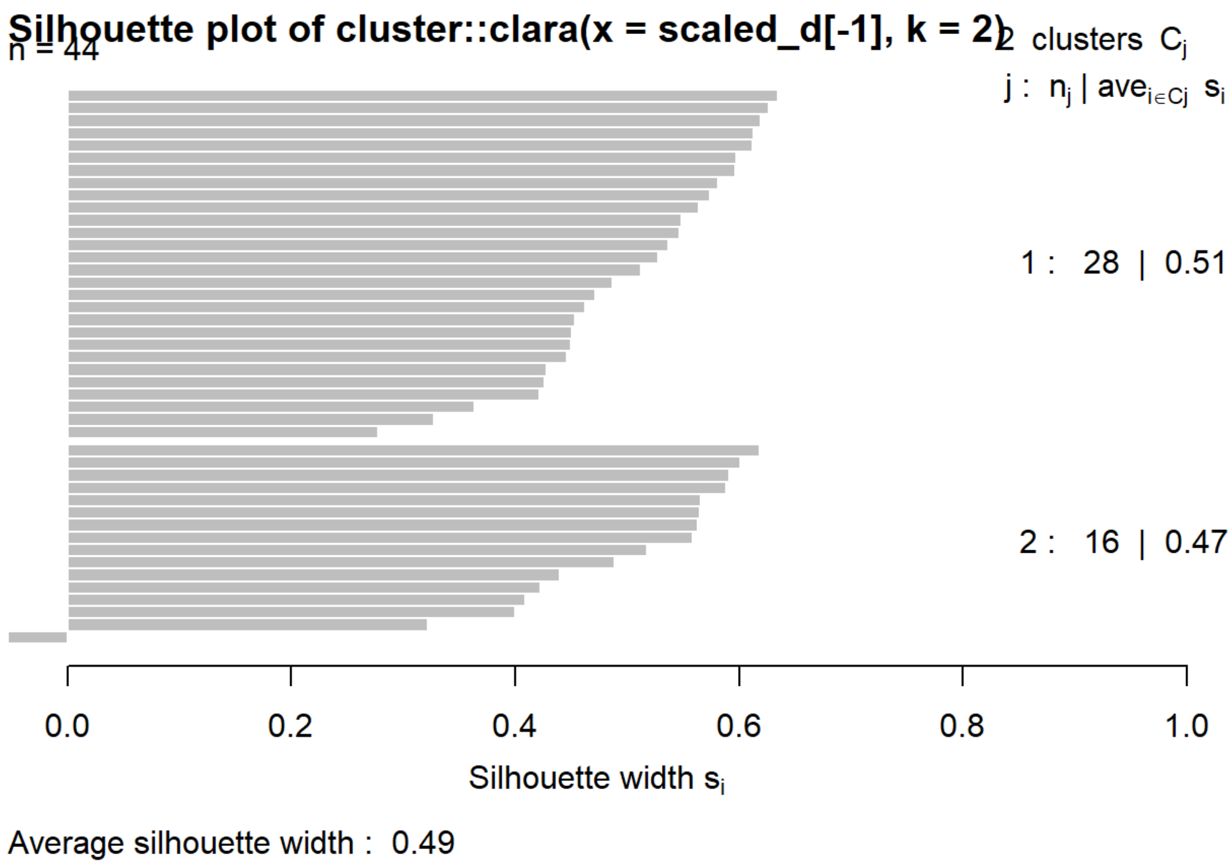
cluster::clara()

The cluster::clara() function is used when robustness is not needed.

Hide

```
clara_d <- cluster::clara(
  x = scaled_d[-1],
  k = 2
)
plot(clara_d)
```





Hide

```
print(clara_d)
```

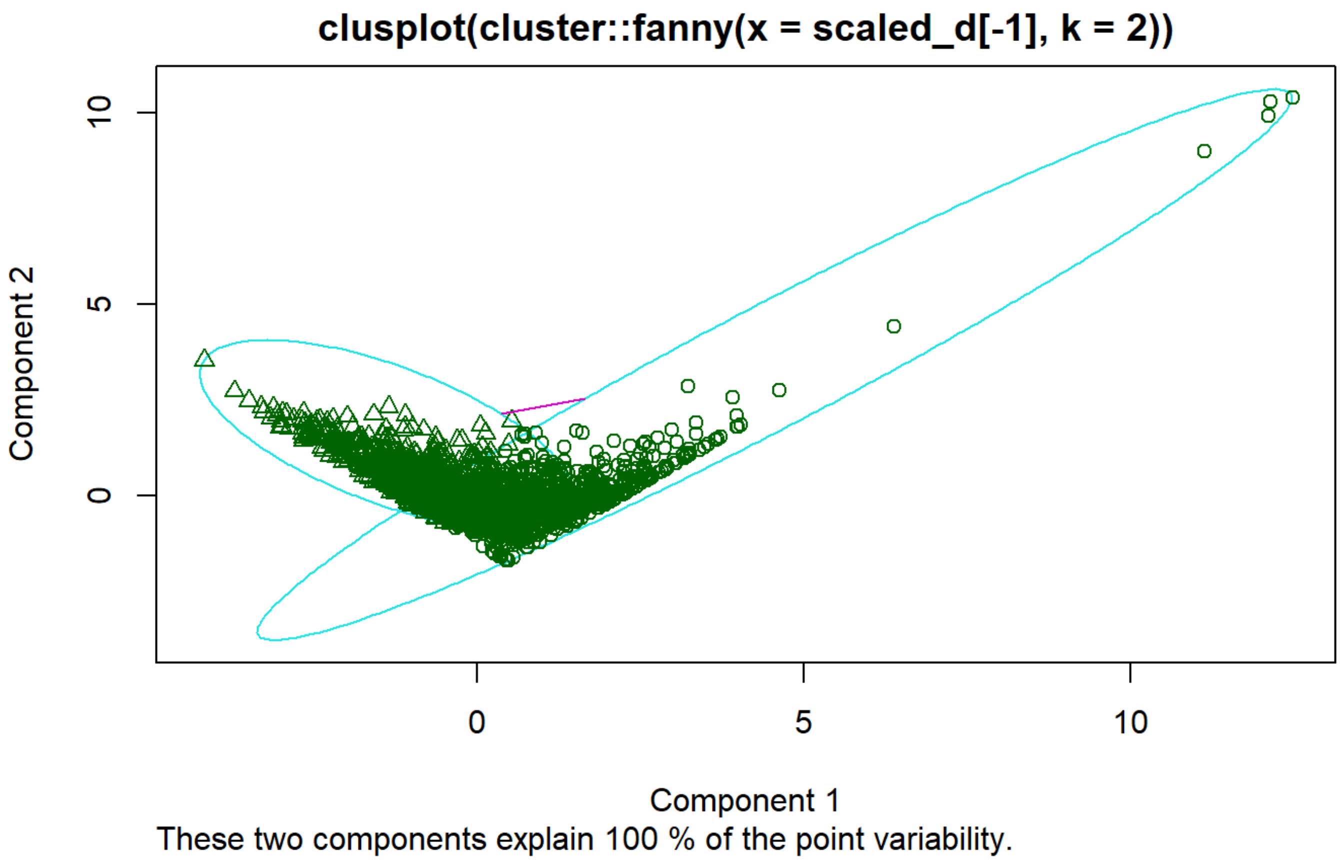
```
Call:    cluster::clara(x = scaled_d[-1], k = 2)
Medoids:
        mileage      price
[1,] -0.2118512 -0.4826366
[2,] -0.5208511  0.6466051
Objective function:  0.7735325
Clustering vector:   int [1:2499] 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 ...
Cluster sizes:           1496 1003
Best sample:
 [1]   12   63  222  232  470  472  510  518  548  604  769  827  898  953  974 1051 1091 1108 1
157 1263 1319 1366 1371 1519
[25] 1530 1602 1867 1900 1942 1980 1986 2001 2005 2099 2160 2161 2191 2200 2209 2368 2383 2385 2
437 2479

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

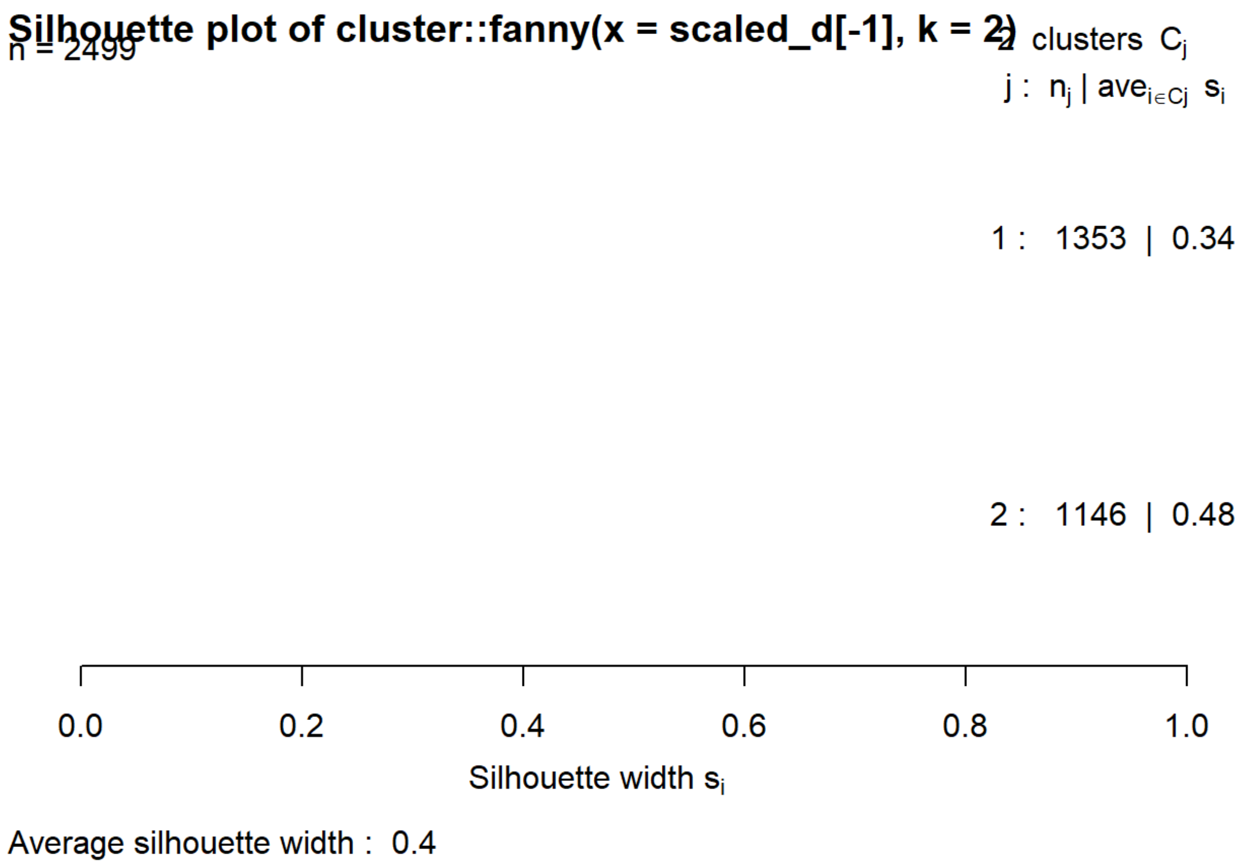
cluster::fanny()

The cluster::fanny() function gives a likelihood of a point belonging to a cluster.

Hide

```
fanny_d <- cluster::fanny(
  x = scaled_d[-1],
  k = 2
)
plot(fanny_d)
```





Hide

```
print(fanny_d)
```

```
Fuzzy Clustering object of class 'fanny' :
m.ship.expon.        2
objective     927.5075
tolerance        1e-15
iterations          40
converged            1
maxit              500
n                 2499
Membership coefficients (in %, rounded):
        [,1] [,2]
   [1,]   55   45
   [2,]   60   40
   [3,]   74   26
   [4,]   36   64
   [5,]   23   77
   [6,]   75   25
   [7,]   64   36
   [8,]   64   36
   [9,]   54   46
  [10,]   76   24
  [11,]   69   31
  [12,]   73   27
  [13,]   23   77
  [14,]   65   35
  [15,]   55   45
  [16,]   67   33
  [17,]   75   25
  [18,]   51   49
  [19,]   73   27
  [20,]   32   68
  [21,]   70   30
  [22,]   75   25
  [23,]   70   30
  [24,]   76   24
  [25,]   55   45
  [26,]   70   30
  [27,]   30   70
  [28,]   71   29
  [29,]   73   27
  [30,]   70   30
  [31,]   67   33
  [32,]   58   42
  [33,]   25   75
  [34,]   73   27
  [35,]   75   25
  [36,]   70   30
  [37,]   62   38
  [38,]   39   61
  [39,]   73   27
  [40,]   60   40
  [41,]   69   31
  [42,]   63   37
```

```
[43,]   21   79
[44,]   74   26
[45,]   39   61
[46,]   69   31
[47,]   57   43
[48,]   40   60
[49,]   75   25
[50,]   39   61
[51,]   74   26
[52,]   59   41
[53,]   19   81
[54,]   77   23
[55,]   44   56
[56,]   73   27
[57,]   40   60
[58,]   54   46
[59,]   29   71
[60,]   48   52
[61,]   66   34
[62,]   34   66
[63,]   57   43
[64,]   72   28
[65,]   19   81
[66,]   71   29
[67,]   31   69
[68,]   46   54
[69,]   69   31
[70,]   42   58
[71,]   70   30
[72,]   62   38
[73,]   20   80
[74,]   71   29
[75,]   41   59
[76,]   70   30
[77,]   61   39
[78,]   18   82
[79,]   72   28
[80,]   51   49
[81,]   71   29
[82,]   49   51
[83,]   45   55
[84,]   72   28
[85,]   45   55
[86,]   70   30
[87,]   47   53
[88,]   55   45
[89,]   67   33
[90,]   74   26
[91,]   70   30
[92,]   75   25
[93,]   21   79
[94,]   68   32
```

```
 [95,]   32   68
 [96,]   39   61
 [97,]   66   34
 [98,]   27   73
 [99,]   18   82
[100,]   75   25
[101,]   60   40
[102,]   24   76
[103,]   76   24
[104,]   63   37
[105,]   20   80
[106,]   76   24
[107,]   69   31
[108,]   44   56
[109,]   77   23
[110,]   33   67
[111,]   21   79
[112,]   72   28
[113,]   55   45
[114,]   26   74
[115,]   72   28
[116,]   20   80
[117,]   34   66
[118,]   69   31
[119,]   19   81
[120,]   21   79
[121,]   72   28
[122,]   25   75
[123,]   36   64
[124,]   36   64
[125,]   30   70
[126,]   74   26
[127,]   23   77
[128,]   39   61
[129,]   74   26
[130,]   77   23
[131,]   29   71
[132,]   71   29
[133,]   44   56
[134,]   75   25
[135,]   19   81
[136,]   31   69
[137,]   74   26
[138,]   40   60
[139,]   26   74
[140,]   77   23
[141,]   71   29
[142,]   61   39
[143,]   23   77
[144,]   75   25
[145,]   66   34
[146,]   20   80
```

```
[147,]   74   26
[148,]   30   70
[149,]   67   33
[150,]   58   42
[151,]   56   44
[152,]   69   31
[153,]   55   45
[154,]   30   70
[155,]   74   26
[156,]   42   58
[157,]   35   65
[158,]   72   28
[159,]   44   56
[160,]   49   51
[161,]   64   36
[162,]   46   54
[163,]   31   69
[164,]   72   28
[165,]   37   63
[166,]   31   69
[167,]   76   24
[168,]   31   69
[169,]   45   55
[170,]   74   26
[171,]   26   74
[172,]   31   69
[173,]   74   26
[174,]   70   30
[175,]   22   78
[176,]   67   33
[177,]   76   24
[178,]   19   81
[179,]   69   31
[180,]   32   68
[181,]   32   68
[182,]   78   22
[183,]   56   44
[184,]   21   79
[185,]   73   27
[186,]   60   40
[187,]   44   56
[188,]   72   28
[189,]   57   43
[190,]   21   79
[191,]   67   33
[192,]   56   44
[193,]   38   62
[194,]   75   25
[195,]   59   41
[196,]   23   77
[197,]   70   30
[198,]   53   47
```

```
[199,]   22   78
[200,]   72   28
[201,]   70   30
[202,]   34   66
[203,]   70   30
[204,]   57   43
[205,]   75   25
[206,]   64   36
[207,]   63   37
[208,]   75   25
[209,]   58   42
[210,]   63   37
[211,]   26   74
[212,]   78   22
[213,]   73   27
[214,]   54   46
[215,]   69   31
[216,]   60   40
[217,]   35   65
[218,]   71   29
[219,]   67   33
[220,]   35   65
[221,]   73   27
[222,]   68   32
[223,]   52   48
[224,]   73   27
[225,]   62   38
[226,]   35   65
[227,]   72   28
[228,]   57   43
[229,]   78   22
[230,]   75   25
[231,]   61   39
[232,]   60   40
[233,]   73   27
[234,]   68   32
[235,]   75   25
[236,]   74   26
[237,]   77   23
[238,]   36   64
[239,]   73   27
[240,]   72   28
[241,]   30   70
[242,]   77   23
[243,]   60   40
[244,]   36   64
[245,]   71   29
[246,]   59   41
[247,]   51   49
[248,]   74   26
[249,]   33   67
[250,]   70   30
```

```
[251,]   77   23
[252,]   66   34
[253,]   64   36
[254,]   73   27
[255,]   57   43
[256,]   24   76
[257,]   73   27
[258,]   65   35
[259,]   70   30
[260,]   71   29
[261,]   63   37
[262,]   28   72
[263,]   71   29
[264,]   65   35
[265,]   70   30
[266,]   55   45
[267,]   48   52
[268,]   73   27
[269,]   67   33
[270,]   27   73
[271,]   78   22
[272,]   67   33
[273,]   28   72
[274,]   69   31
[275,]   69   31
[276,]   60   40
[277,]   75   25
[278,]   42   58
[279,]   42   58
[280,]   62   38
[281,]   24   76
[282,]   45   55
[283,]   71   29
[284,]   22   78
[285,]   62   38
[286,]   66   34
[287,]   67   33
[288,]   78   22
[289,]   71   29
[290,]   65   35
[291,]   67   33
[292,]   66   34
[293,]   70   30
[294,]   56   44
[295,]   71   29
[296,]   67   33
[297,]   75   25
[298,]   63   37
[299,]   67   33
[300,]   68   32
[301,]   71   29
[302,]   61   39
```

```
[303,]   67   33
[304,]   42   58
[305,]   73   27
[306,]   57   43
[307,]   77   23
[308,]   64   36
[309,]   73   27
[310,]   62   38
[311,]   60   40
[312,]   54   46
[313,]   34   66
[314,]   60   40
[315,]   69   31
[316,]   76   24
[317,]   64   36
[318,]   75   25
[319,]   67   33
[320,]   64   36
[321,]   69   31
[322,]   31   69
[323,]   62   38
[324,]   59   41
[325,]   73   27
[326,]   62   38
[327,]   65   35
[328,]   68   32
[329,]   62   38
[330,]   27   73
[331,]   55   45
[332,]   59   41
[333,]   76   24
[334,]   62   38
[335,]   67   33
[336,]   70   30
[337,]   23   77
[338,]   69   31
[339,]   30   70
[340,]   58   42
[341,]   57   43
[342,]   76   24
[343,]   62   38
[344,]   68   32
[345,]   65   35
[346,]   71   29
[347,]   35   65
[348,]   57   43
[349,]   70   30
[350,]   62   38
[351,]   67   33
[352,]   63   37
[353,]   64   36
[354,]   56   44
```

```
[355,]   61   39
[356,]   60   40
[357,]   69   31
[358,]   62   38
[359,]   77   23
[360,]   66   34
[361,]   67   33
[362,]   38   62
[363,]   59   41
[364,]   57   43
[365,]   68   32
[366,]   62   38
[367,]   74   26
[368,]   55   45
[369,]   71   29
[370,]   39   61
[371,]   64   36
[372,]   66   34
[373,]   73   27
[374,]   70   30
[375,]   67   33
[376,]   72   28
[377,]   74   26
[378,]   36   64
[379,]   62   38
[380,]   62   38
[381,]   39   61
[382,]   58   42
[383,]   62   38
[384,]   60   40
[385,]   75   25
[386,]   38   62
[387,]   65   35
[388,]   57   43
[389,]   75   25
[390,]   70   30
[391,]   61   39
[392,]   68   32
[393,]   73   27
[394,]   40   60
[395,]   59   41
[396,]   55   45
[397,]   75   25
[398,]   70   30
[399,]   58   42
[400,]   41   59
[401,]   65   35
[402,]   69   31
[403,]   76   24
[404,]   38   62
[405,]   67   33
[406,]   69   31
```

```
[407,]   70   30
[408,]   77   23
[409,]   66   34
[410,]   61   39
[411,]   56   44
[412,]   72   28
[413,]   31   69
[414,]   71   29
[415,]   40   60
[416,]   70   30
[417,]   72   28
[418,]   68   32
[419,]   62   38
[420,]   64   36
[421,]   61   39
[422,]   58   42
[423,]   65   35
[424,]   61   39
[425,]   71   29
[426,]   28   72
[427,]   40   60
[428,]   70   30
[429,]   74   26
[430,]   53   47
[431,]   67   33
[432,]   37   63
[433,]   73   27
[434,]   25   75
[435,]   75   25
[436,]   62   38
[437,]   71   29
[438,]   55   45
[439,]   70   30
[440,]   31   69
[441,]   74   26
[442,]   70   30
[443,]   71   29
[444,]   56   44
[445,]   30   70
[446,]   69   31
[447,]   34   66
[448,]   55   45
[449,]   63   37
[450,]   55   45
[451,]   72   28
[452,]   24   76
[453,]   36   64
[454,]   78   22
[455,]   71   29
[456,]   56   44
[457,]   69   31
[458,]   20   80
```

```
 [459,]   67   33
 [460,]   64   36
 [461,]   54   46
 [462,]   72   28
 [463,]   23   77
 [464,]   75   25
 [465,]   71   29
 [466,]   70   30
 [467,]   56   44
 [468,]   68   32
 [469,]   73   27
 [470,]   21   79
 [471,]   63   37
 [472,]   41   59
 [473,]   68   32
 [474,]   62   38
 [475,]   74   26
 [476,]   26   74
 [477,]   22   78
 [478,]   71   29
 [479,]   70   30
 [480,]   72   28
 [481,]   71   29
 [482,]   19   81
 [483,]   67   33
 [484,]   37   63
 [485,]   73   27
 [486,]   61   39
 [487,]   71   29
 [488,]   21   79
 [489,]   22   78
 [490,]   73   27
 [491,]   51   49
 [492,]   68   32
 [493,]   62   38
 [494,]   69   31
 [495,]   69   31
 [496,]   19   81
 [497,]   57   43
 [498,]   20   80
 [499,]   73   27
 [500,]   63   37
 [ reached getOption("max.print") -- omitted 1999 rows ]
Fuzzyness coefficients:
dunn_coeff normalized
 0.5816543  0.1633085
Closest hard clustering:
   [1] 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 2
1 1 2 1 2 1 1 2 1 2 1 2 1 2 2
  [61] 1 2 1 1 2 1 2 2 1 2 1 1 2 1 2 1 1 2 1 1 1 2 2 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 2 1 1 2 1 1 2
1 1 2 1 2 2 1 1 2 1 2 2 1 2 2
 [121] 1 2 2 2 2 1 2 2 1 1 2 1 2 1 2 2 1 2 2 1 1 1 2 1 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 2 1 2 2 1 2
```

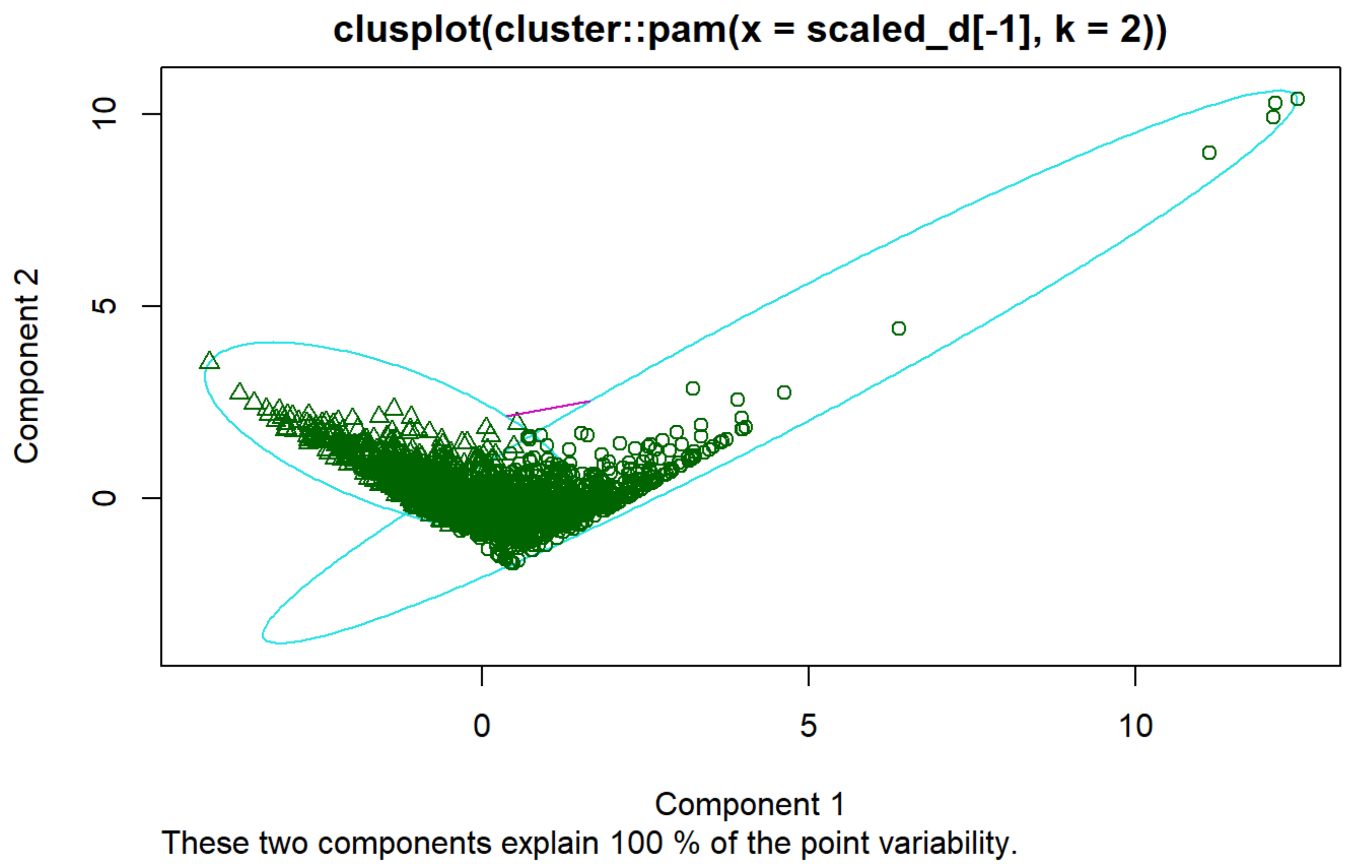
```
2 1 2 2 1 2 2 1 1 2 1 1 2 1 2
 [181] 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1
2 1 1 1 1 1 1 1 1 1 1 1 2 1 1
 [241] 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 2 2 1 2 2 1 2 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [301] 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1
1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 [361] 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1
1 1 1 1 1 1 1 2 1 2 1 1 1 1 1
 [421] 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 2 1 1
1 1 1 1 2 1 2 1 1 1 2 2 1 1 1
 [481] 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 [541] 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 2 2 1 2 1 2 2 2 1
 [601] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 1 2 2 2 1 2 1 2 1 2 2 2 2 2 1
2 2 1 2 2 1 1 2 2 2 1 2 2 2 1
 [661] 2 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 2 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 1 1 1 1
2 2 1 2 2 1 2 2 2 1 2 1 1 2 2
 [721] 1 2 1 1 2 1 1 1 2 2 2 2 1 1 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 1 1 2
2 2 2 1 2 1 2 2 1 1 1 1 1 2 2
 [781] 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 1 1 2 2
1 1 2 2 2 2 2 1 1 1 1 1 2 1 1
 [841] 1 1 1 2 1 2 1 1 1 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 1 1 1 1 1 2 2 1
1 2 2 2 2 2 1 2 1 2 1 1 2 1 1
 [901] 1 1 1 2 2 1 1 1 2 1 2 2 2 1 2 2 2 2 2 1 2 1 2 2 1 2 1 1 1 2 2 2 2 1 1 1 1 1 1 1 2 2 1 2 2
1 2 2 1 2 1 1 1 1 1 2 1 1 1 2
 [961] 2 1 1 1 2 2 2 1 1 2 2 1 2 1 2 2 2 2 1 2 1 1 1 2 2 2 2 1 2 1 2 1 2 1 2 1 1 2 2 1
 [ reached getOption("max.print") -- omitted 1499 entries ]

Available components:
 [1] "membership"  "coeff"       "memb.exp"    "clustering"  "k.crisp"     "objective"   "conver
gence" "diss"
 [9] "call"        "silinfo"     "data"
```

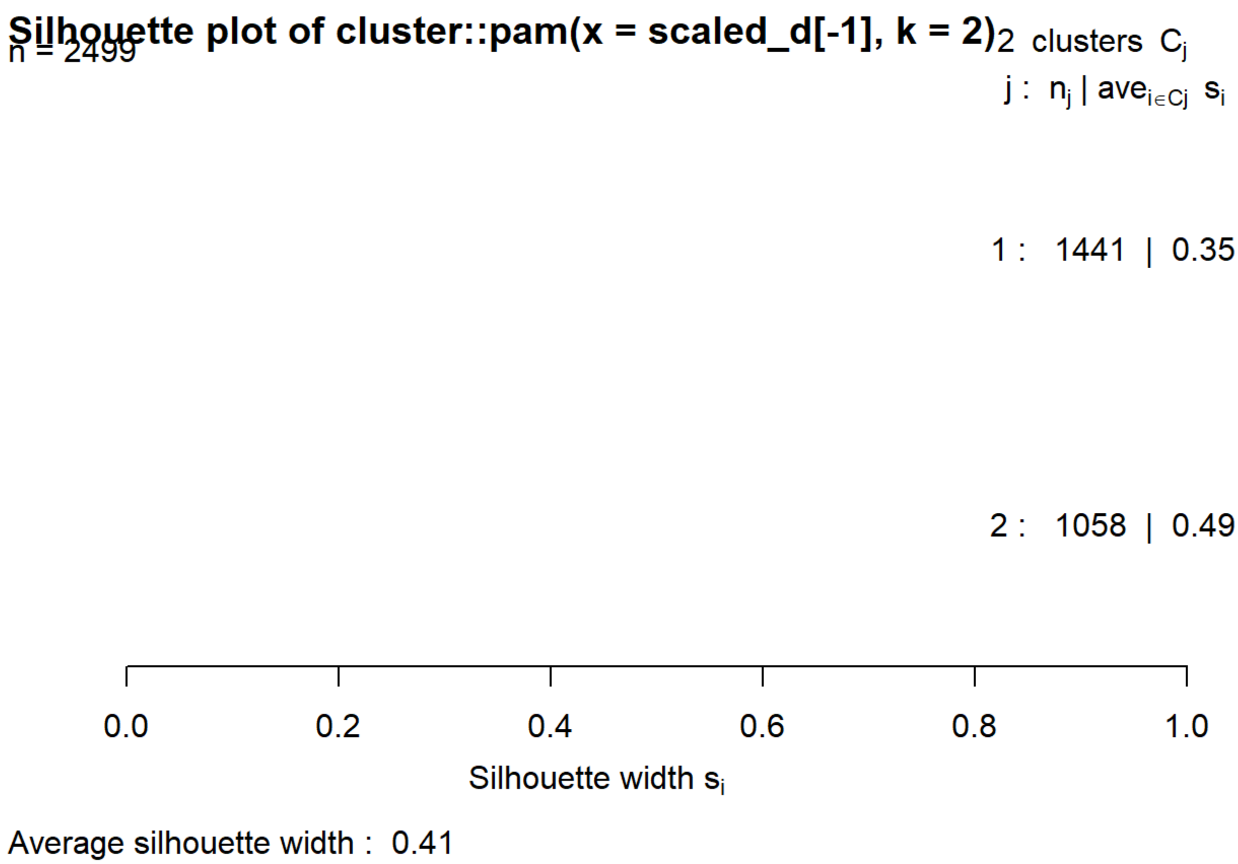
cluster::pam()

The cluster::pam() function, also a robust version of k-means, uses medoids and centers the observations in the dataset. Usually a good choice when the dataset contains outliers. This is time consuming, so other options of clustering might be better.

Hide

```
pam_d <- cluster::pam(scaled_d[-1],
                      k=2)
plot(pam_d)
```





Hide

```
print(pam_d)
```

```
Medoids:
       ID       mileage      price
[1,] 2117  0.0006919717 -0.5833292
[2,]  613 -0.4617443606  0.6961260
Clustering vector:
   [1] 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 2
1 1 2 1 2 1 1 2 1 1 1 2 1 2 1
  [61] 1 2 1 1 2 1 2 1 1 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 2 1 2 2 1 1 2 1 1 2
1 1 1 1 2 2 1 1 2 1 2 2 1 2 2
 [121] 1 2 2 2 2 1 2 2 1 1 2 1 1 1 2 2 1 2 2 1 1 1 2 1 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 1 1 1 2 1 2
2 1 2 1 1 2 2 1 1 2 1 1 2 1 2
 [181] 2 1 1 2 1 1 1 1 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1
2 1 1 1 1 1 1 1 1 1 1 1 2 1 1
 [241] 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 2 1 2 1 1 2 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [301] 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1
1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 [361] 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1
1 1 1 1 1 1 1 2 1 2 1 1 1 1 1
 [421] 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 2 1 1
1 1 1 1 2 1 2 1 1 1 2 2 1 1 1
 [481] 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 [541] 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 2 1 2 2 2 1 2 1 2 2 2 1
 [601] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 1 2 2 2 1 2 1 2 1 2 2 2 2 2 1
2 2 1 2 2 1 1 2 2 1 1 2 2 2 1
 [661] 2 2 1 2 2 1 1 2 2 1 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 2 1 1 1 1 2 1 2 1 2 1 1 1 2 2 2 1 1 1 1
2 2 1 2 2 1 2 2 2 1 2 1 1 1 2
 [721] 1 2 1 1 2 1 1 1 2 1 2 2 1 1 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 1 1 2
2 2 2 1 2 1 1 1 1 1 1 1 1 2 2
 [781] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 1 1 1 2 2 2 2 2 2 2 2 1 1 2 2
1 1 2 2 2 2 2 1 1 1 1 1 1 1 1
 [841] 1 1 1 1 1 2 1 1 1 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 1 1 1 1 1 2 2 1
1 1 2 2 2 2 1 2 1 2 1 1 2 1 1
 [901] 1 1 1 2 1 1 1 1 2 1 2 2 2 1 2 2 2 2 2 1 2 1 2 2 1 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 2 2 1 2 2
1 2 1 1 1 1 1 1 1 1 2 1 1 1 2
 [961] 2 1 1 1 2 1 2 1 1 1 2 1 2 1 2 2 2 2 1 2 1 1 1 2 2 2 2 1 2 1 2 1 2 1 2 1 1 2 2 1
 [ reached getOption("max.print") -- omitted 1499 entries ]
Objective function:
    build      swap 
0.8330044 0.7611082 

Available components:
 [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"  "clusinfo"   "silinfo"   
"diss"       "call"      
[10] "data"      
```

As mentioned earlier, in the analysis for this problem, k=2 would make the most sense since the comparison for groups is based on high-mileage low-price and low-price high-mileage vehicles. For a more in depth analysis that takes in to account the condition, brand, and the state the vehicle is sold in, k=7 would make more sense. With that in consideration, k=2 was then tested with the K-Means, and Clustering Large Application, Fuzzy Analysis Clustering, and Partitioning Around Medoids using the cluster package in R. Looking at the Cluster means for the algorithms, k-means(k=2) showed a clear comparison of groups for low-mileage high-price vehicles and high-mileage and low-price vehicles. The cluster::clara() function showed low-mileage low-price and low-mileage high price vehicle clusters. The cluster::fanny() function brought the centers closer in the comparison of high-mileage low-price vehicles and low-mileage high-price vehicles. The cluster::pam() function, even though being a robust version of k-means clustering, had the centers far off than k-means. The partitioning algorithm that would be best suited for this analysis is Fuzzy Analysis Clustering with k=2 since it brings the centers closer than k-means and gives us a likelihood of data points belonging to the cluster.

Hierarchical Clustering

hclust()

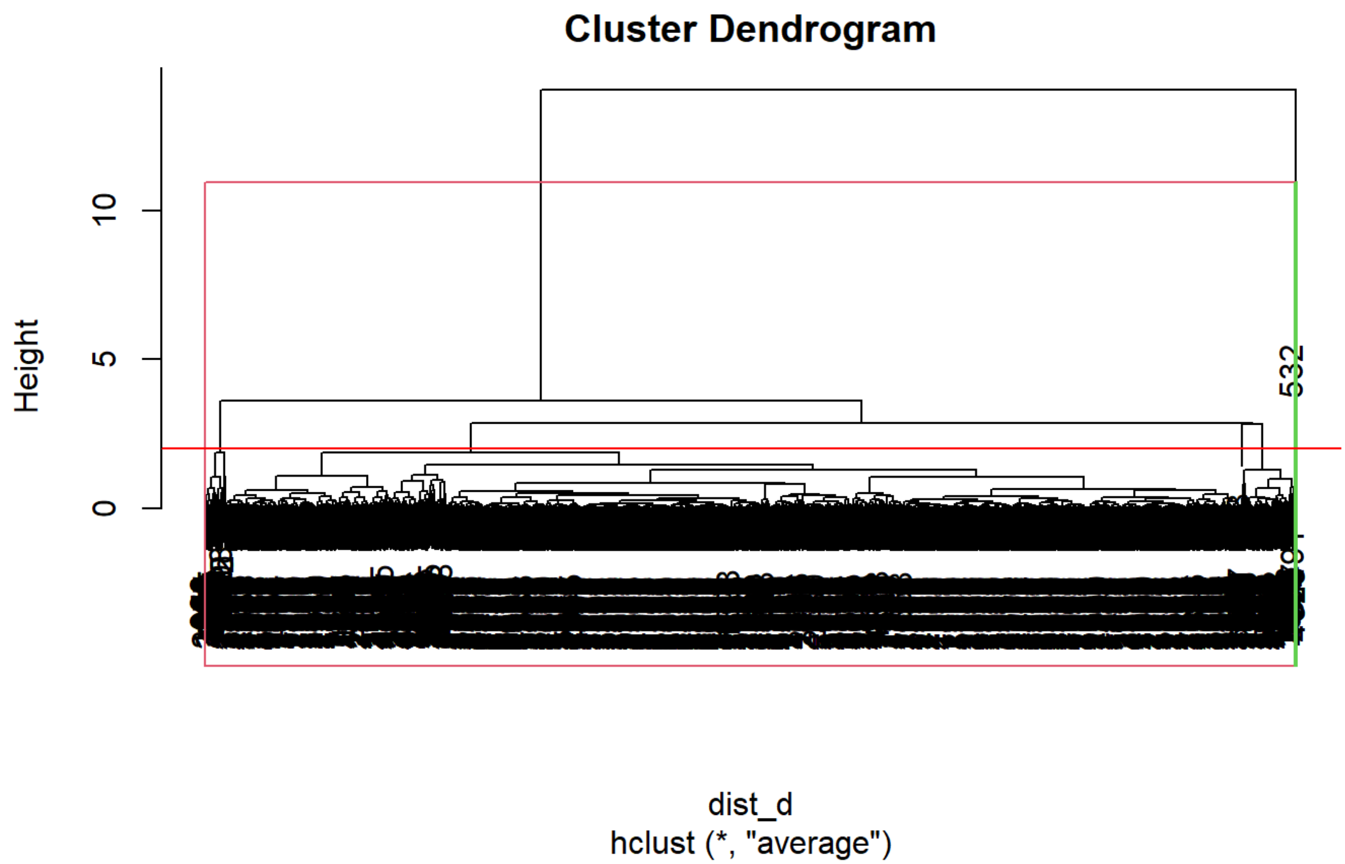
Hierarchical clustering requires a distance matrix to perform divisive clustering. Setting the number of clusters to 2 and performing hierarchical clustering with original scaled data

Hide

```
#Creating a distance matrix for the scaled dataset used in the analysis
dist_d <- dist(
  x=scaled_d[-1],
  method = 'euclidean'
)

#Performing hierarchical clustering
hclust_d <- hclust(
  d = dist_d,
  method = 'average'
)

plot(hclust_d)
rect.hclust(hclust_d , k = 2, border = 2:6)
```

Hide

```
abline(h = 2, col = 'red')
```



Hide

```
#coefficient for clustering
coef(hclust_d)
```

```
[1] 0.9972347
```

Not much can be visualized by the above plot due to the size of the data.

Have to reduce the size of the data to visualize better.

Hide

```
set.seed(1)
#Reducing the size of the randomly to visualize the dendogram
reduced_scaled_d <- scaled_d[sample(nrow(scaled_d), 200), ]
reduced_scaled_d
```

	year	mileage	price
	<fctr>	<dbl>	<dbl>
1017	2019	-0.57598841	1.26545134
679	2019	-0.29861036	1.51305587
2177	2019	-0.20639107	0.01917520
930	2019	-0.43427621	-0.10462707

	year	mileage	price
	<fctr>	<dbl>	<dbl>
1533	2016	1.07561777	-0.35223160
471	2010	1.72418431	-1.54692346
2347	2019	-0.60380829	0.38232851
270	2019	-0.71337940	0.31630064
1211	2015	-0.39128186	1.00769502
597	2019	-0.48579574	0.56390517

1-10 of 200 rows Previous **1** 2 3 4 5 6 ... 20 Next

Hide

```
#Creating a distance matrix for the scaled dataset used in the analysis using euclidean distance
dist_d <- dist(
  x=reduced_scaled_d[-1],
  method = 'euclidian'
)
```

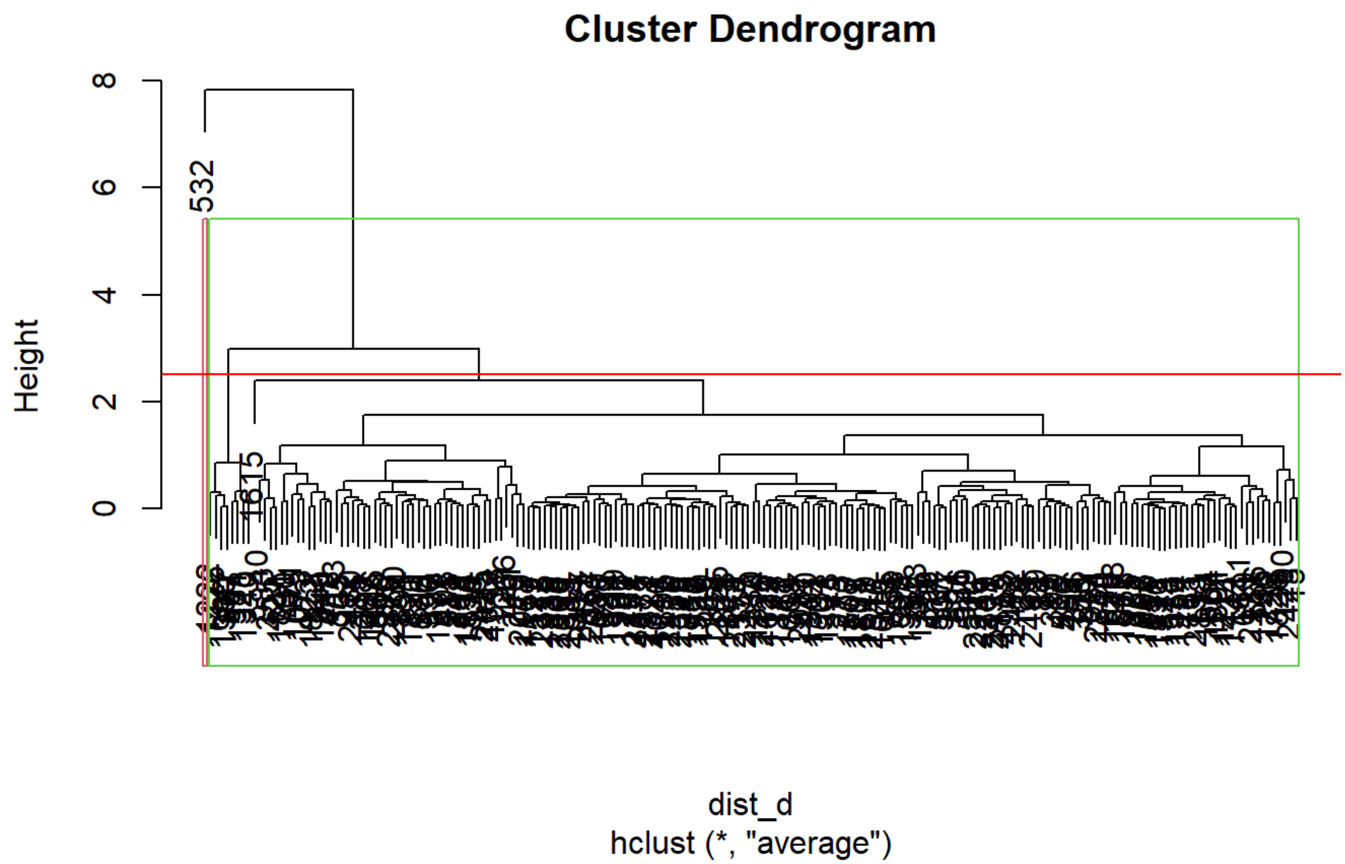
Performing hierarchical clustering using ‘average’ method. (Using the reduced size data)

Hide

```
hclust_d <- hclust(
  d = dist_d,
  method = 'average'
)
plot(hclust_d)
rect.hclust(hclust_d , k = 2, border = 2:6)
```

Hide

```
abline(h = 2.5, col = 'red')
```



Hide

```
#coefficient for clustering
coef(hclust_d)
```

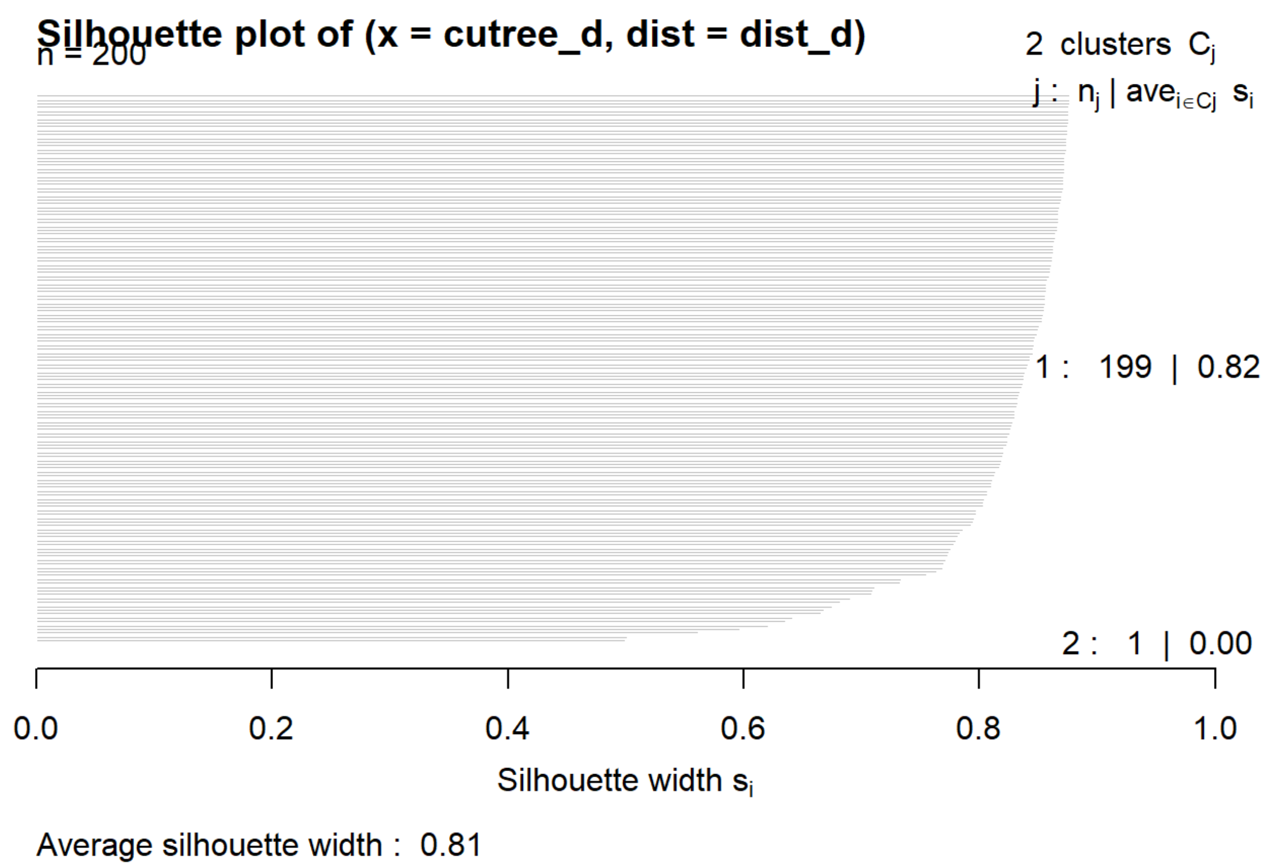
```
[1] 0.9810651
```

In the plot above, we can see that every data point finally merges into a single cluster with the height shown on the y-axis about 2.5, however, in this analysis we already know that the groups for comparison are high-price low-mileage vehicles and low-price high-mileage vehicles.

Using Silhouette plots to evaluate how well each point fits with the rest of its cluster. (Using the reduced sized data)

Hide

```
cutree_d <- cutree(
  tree = hclust_d,
  k = 2
)

silhouette_d <- cluster::silhouette(
  x = cutree_d,
  dist = dist_d
)
plot(
  x = silhouette_d
)
```



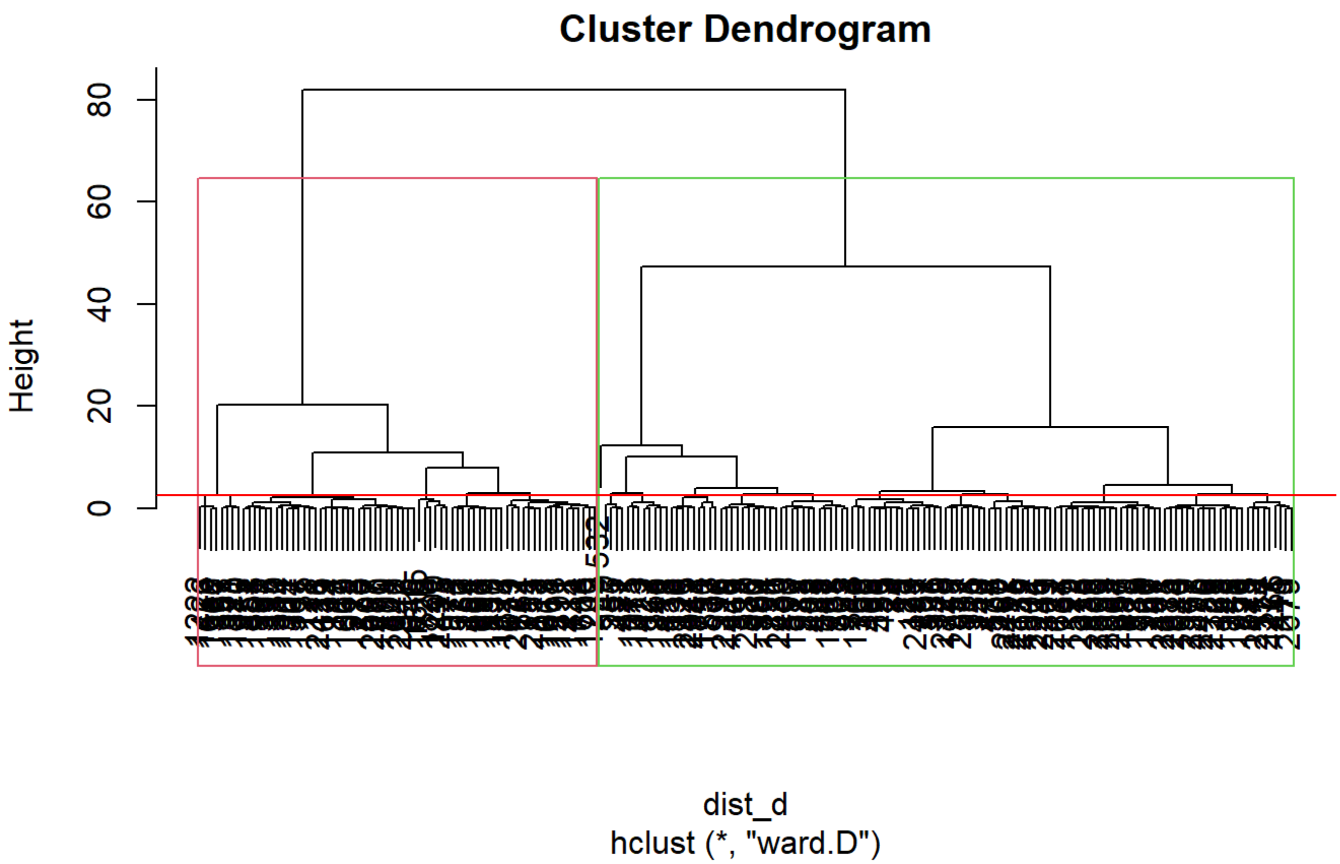
Performing hierarchical clustering using ‘ward.D’ method. (Using the reduced size data)

Hide

```
hclust_d <- hclust(
  d = dist_d,
  method = 'ward.D'
)
plot(hclust_d)
rect.hclust(hclust_d , k = 2, border = 2:6)
```

Hide

```
abline(h = 2.5, col = 'red')
```



Hide

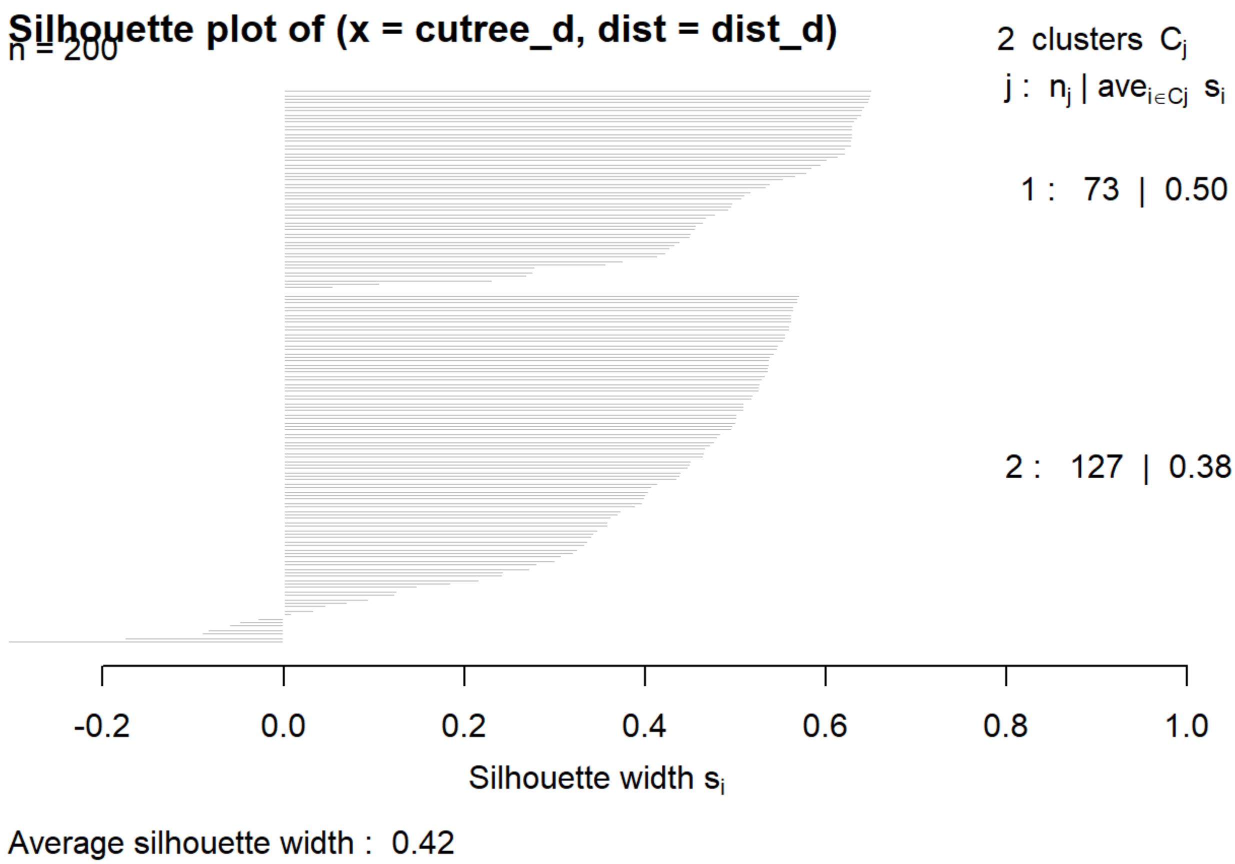
```
#coefficient for clustering
coef(hclust_d)
```

```
[1] 0.9978797
```

Using Silhouette plots to evaluate how well each point fits with the rest of its cluster. (Using the reduced sized data)

Hide

```
cutree_d <- cutree(
  tree = hclust_d,
  k = 2
)

silhouette_d <- cluster::silhouette(
  x = cutree_d,
  dist = dist_d
)
plot(
  x = silhouette_d
)
```



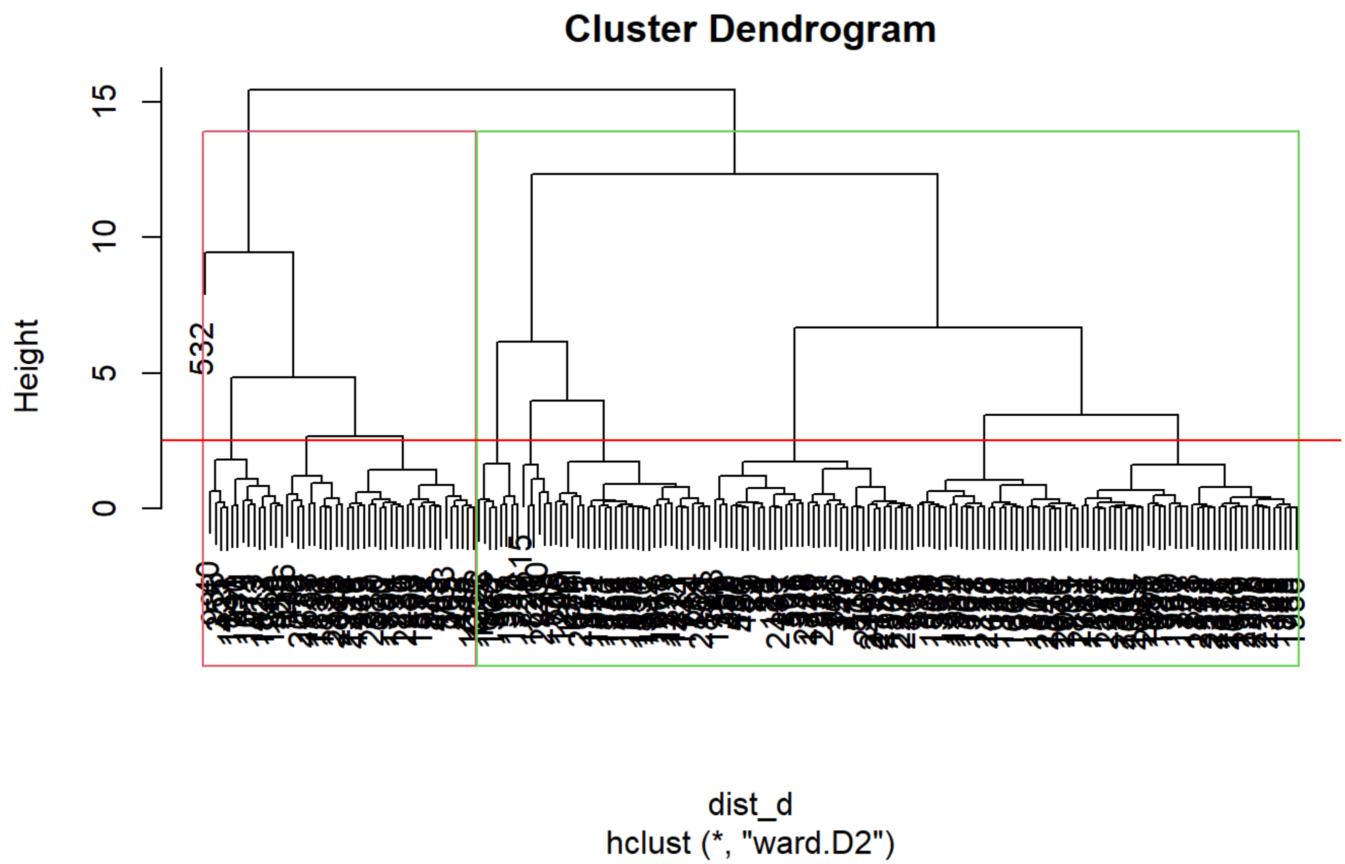
Performing hierarchical clustering using ‘ward.D2’ method. (Using the reduced size data)

Hide

```
hclust_d <- hclust(
  d = dist_d,
  method = 'ward.D2'
)
plot(hclust_d)
rect.hclust(hclust_d , k = 2, border = 2:6)
```

Hide

```
abline(h = 2.5, col = 'red')
```



Hide

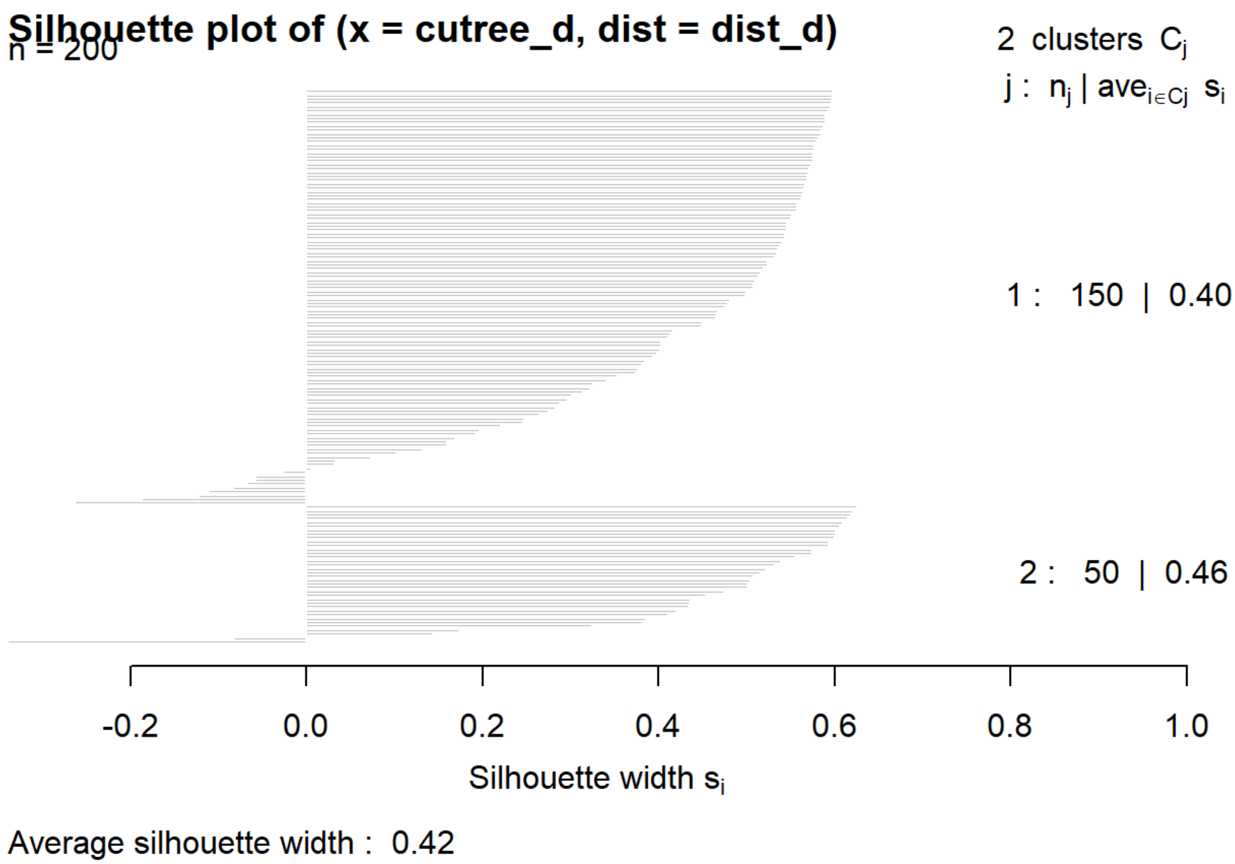
```
#coefficient for clustering
coef(hclust_d)
```

```
[1] 0.9898605
```

Using Silhouette plots to evaluate how well each point fits with the rest of its cluster. (Using the reduced sized data)

Hide

```
cutree_d <- cutree(
  tree = hclust_d,
  k = 2
)

silhouette_d <- cluster::silhouette(
  x = cutree_d,
  dist = dist_d
)
plot(
  x = silhouette_d
)
```



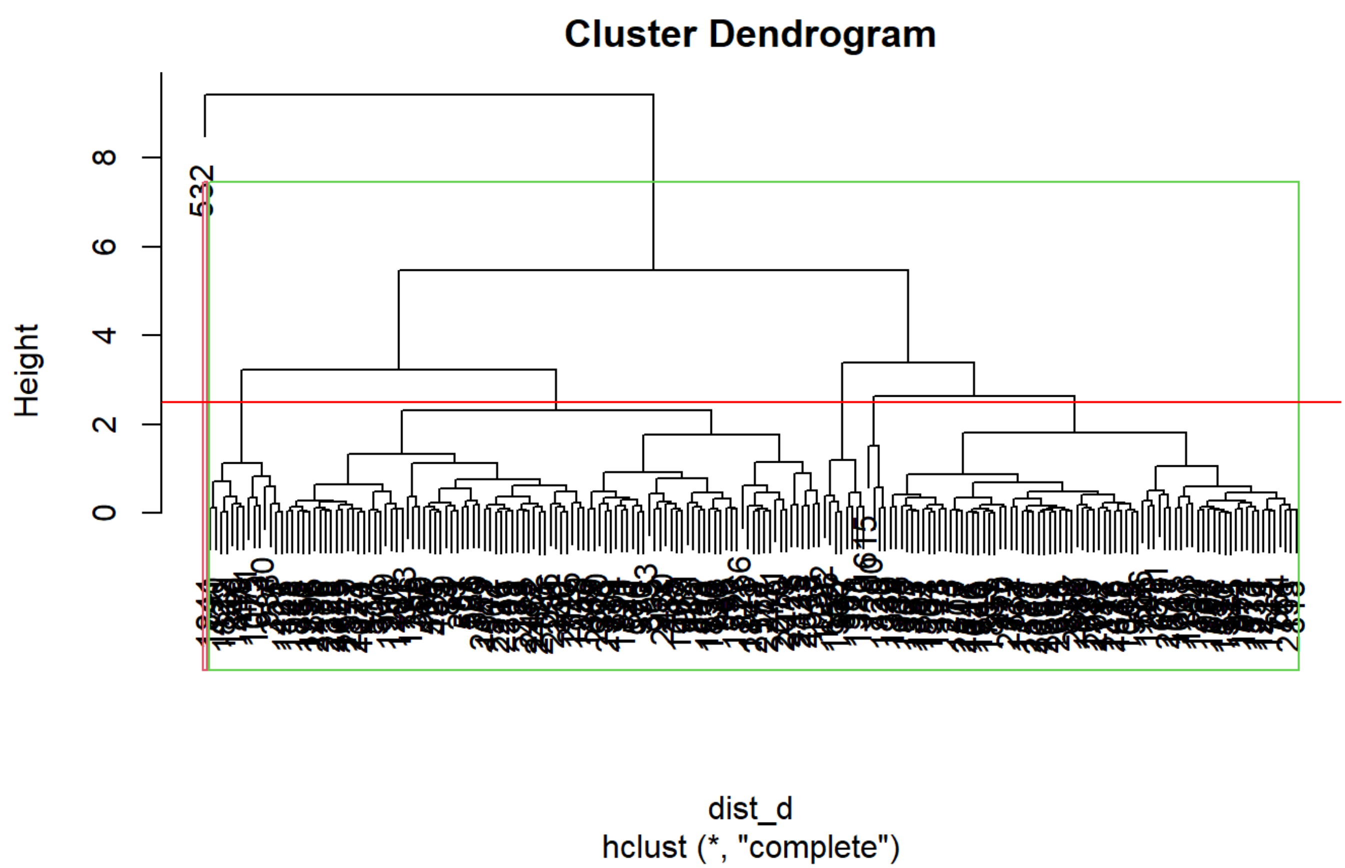
Performing hierarchical clustering using 'complete' method. (Using the reduced size data)

Hide

```
hclust_d <- hclust(
  d = dist_d,
  method = 'complete'
)
plot(hclust_d)
rect.hclust(hclust_d , k = 2, border = 2:6)
```

Hide

```
abline(h = 2.5, col = 'red')
```



Hide

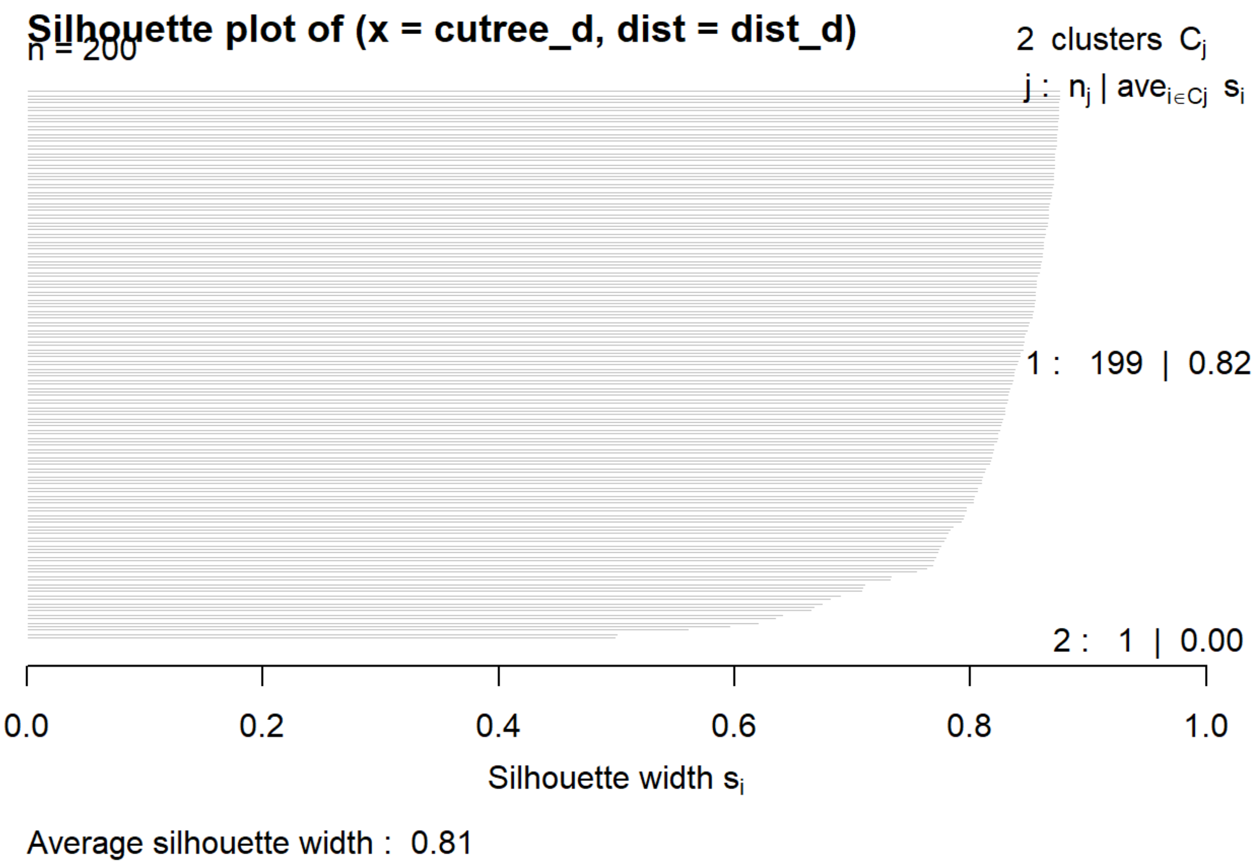
```
#coefficient for clustering
coef(hclust_d)
```

```
[1] 0.9833109
```

Using Silhouette plots to evaluate how well each point fits with the rest of its cluster. (Using the reduced sized data)

Hide

```
cutree_d <- cutree(
  tree = hclust_d,
  k = 2
)

silhouette_d <- cluster::silhouette(
  x = cutree_d,
  dist = dist_d
)
plot(
  x = silhouette_d
)
```



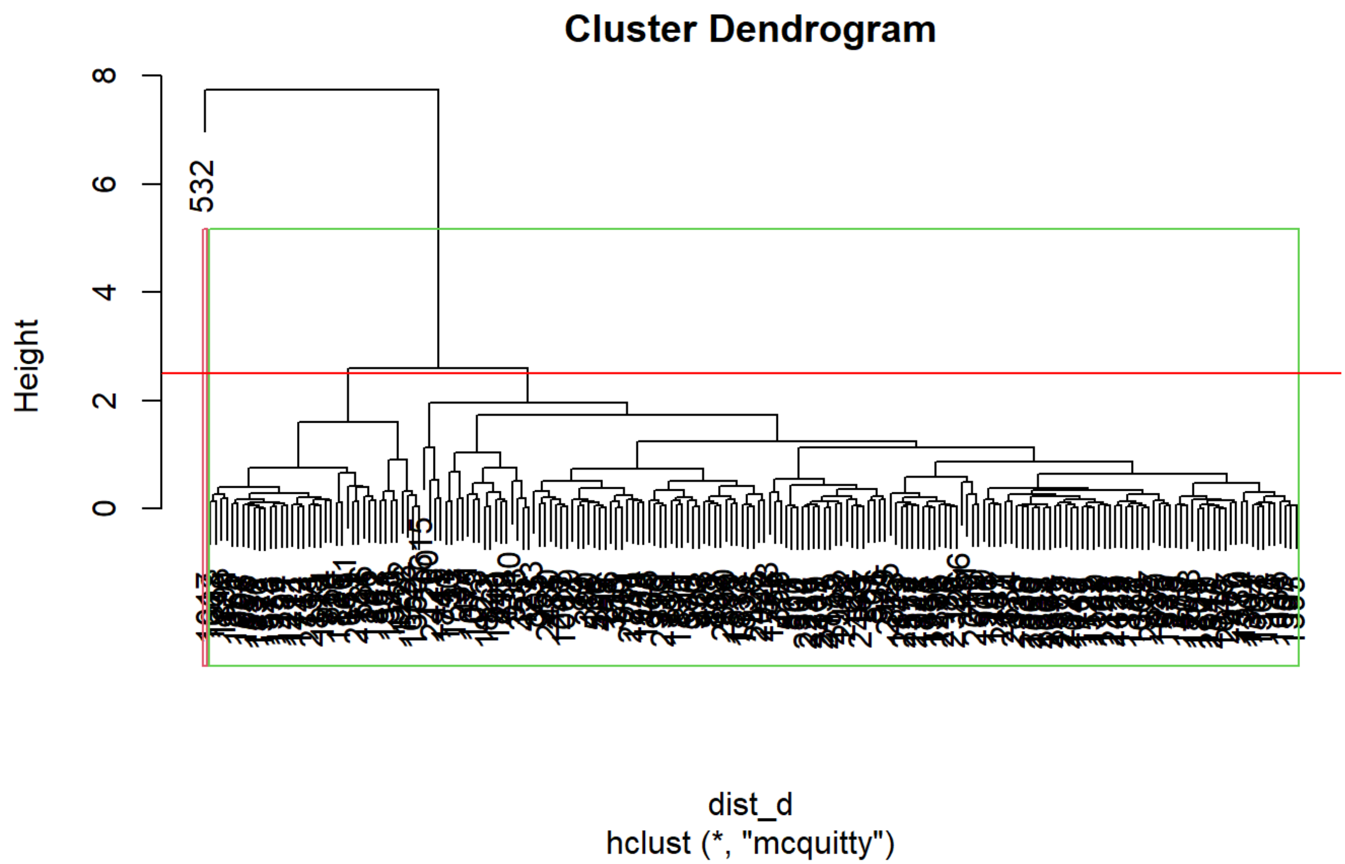
Performing hierarchical clustering using ‘mcquitty’ method. (Using the reduced size data)

Hide

```
hclust_d <- hclust(
  d = dist_d,
  method = 'mcquitty'
)
plot(hclust_d)
rect.hclust(hclust_d , k = 2, border = 2:6)
```

Hide

```
abline(h = 2.5, col = 'red')
```



Hide

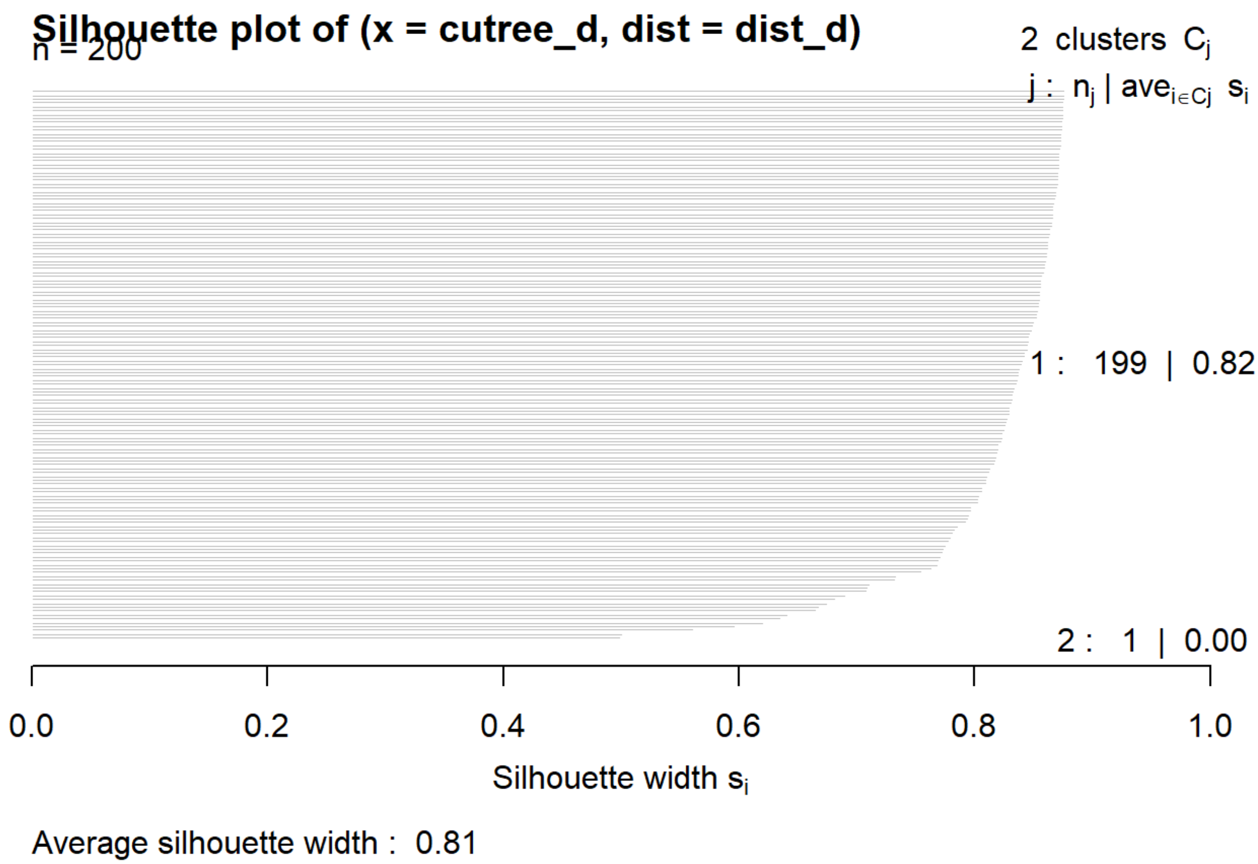
```
#coefficient for clustering
coef(hclust_d)
```

```
[1] 0.9817874
```

Using Silhouette plots to evaluate how well each point fits with the rest of its cluster. (Using the reduced sized data)

Hide

```
cutree_d <- cutree(
  tree = hclust_d,
  k = 2
)

silhouette_d <- cluster::silhouette(
  x = cutree_d,
  dist = dist_d
)
plot(
  x = silhouette_d
)
```



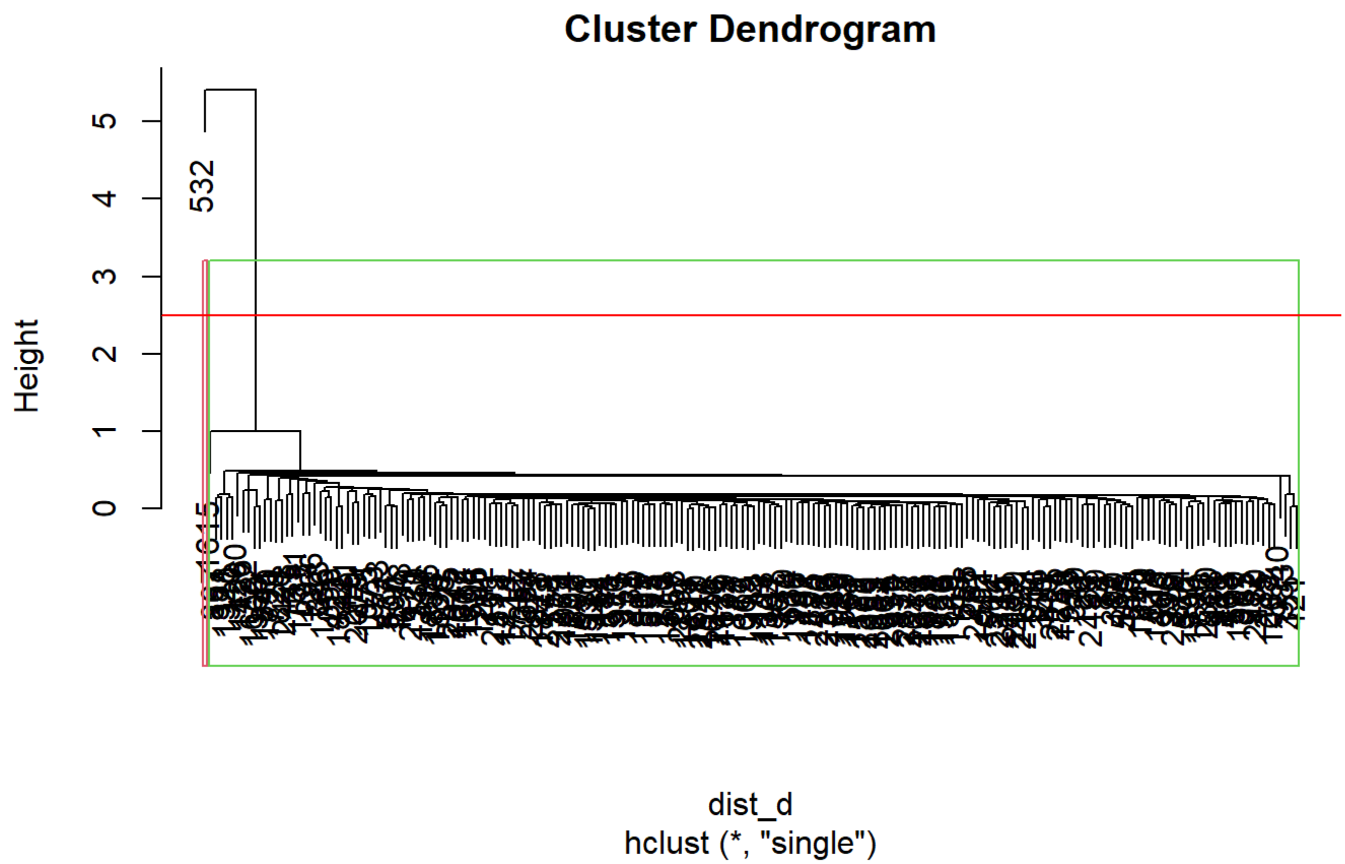
Performing hierarchical clustering using 'single' method. (Using the reduced size data)

Hide

```
hclust_d <- hclust(
  d = dist_d,
  method = 'single'
)
plot(hclust_d)
rect.hclust(hclust_d , k = 2, border = 2:6)
```

Hide

```
abline(h = 2.5, col = 'red')
```



Hide

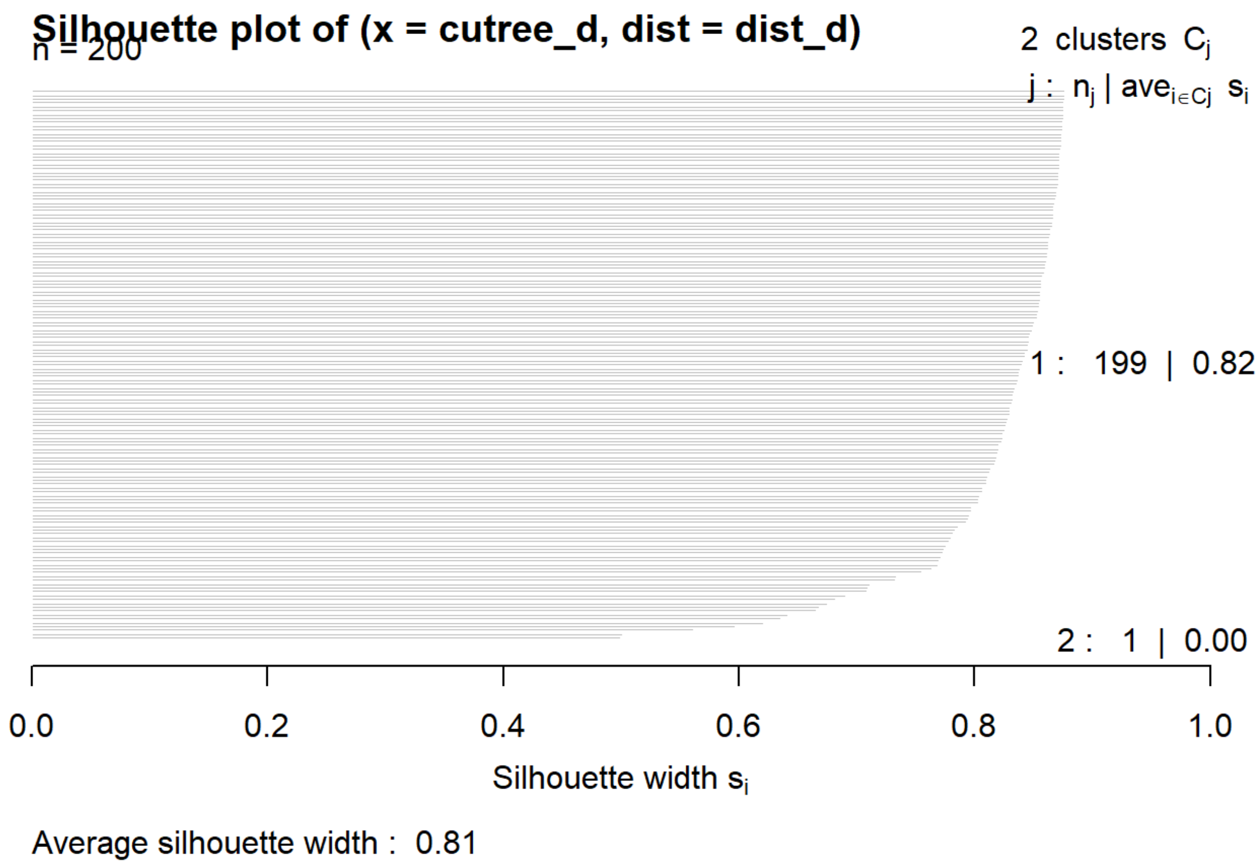
```
#coefficient for clustering
coef(hclust_d)
```

```
[1] 0.9780878
```

Using Silhouette plots to evaluate how well each point fits with the rest of its cluster. (Using the reduced sized data)

Hide

```
cutree_d <- cutree(
  tree = hclust_d,
  k = 2
)

silhouette_d <- cluster::silhouette(
  x = cutree_d,
  dist = dist_d
)
plot(
  x = silhouette_d
)
```



Coefficients for hierarchical clustering using hclust() with the following methods:

average = 0.9619 ward.D = 0.9983 ward.D2 = 0.99256 complete = 0.9815 mcquitty = 0.9669013 single = 0.902107

Even though the mcquitty method has the smaller coefficient compared to other methods for clustering, it still does a fairly decent job of clustering based on approximately equal number of observations in both clusters and might detect outliers somewhat effectively. However, ward.D will do a better job at creating clusters based on equal number of observations since it has the highest coefficient. The smallest coefficient for hclust() hierarchical clustering using the ‘single’ method means that this method will do the best job of detecting outliers.

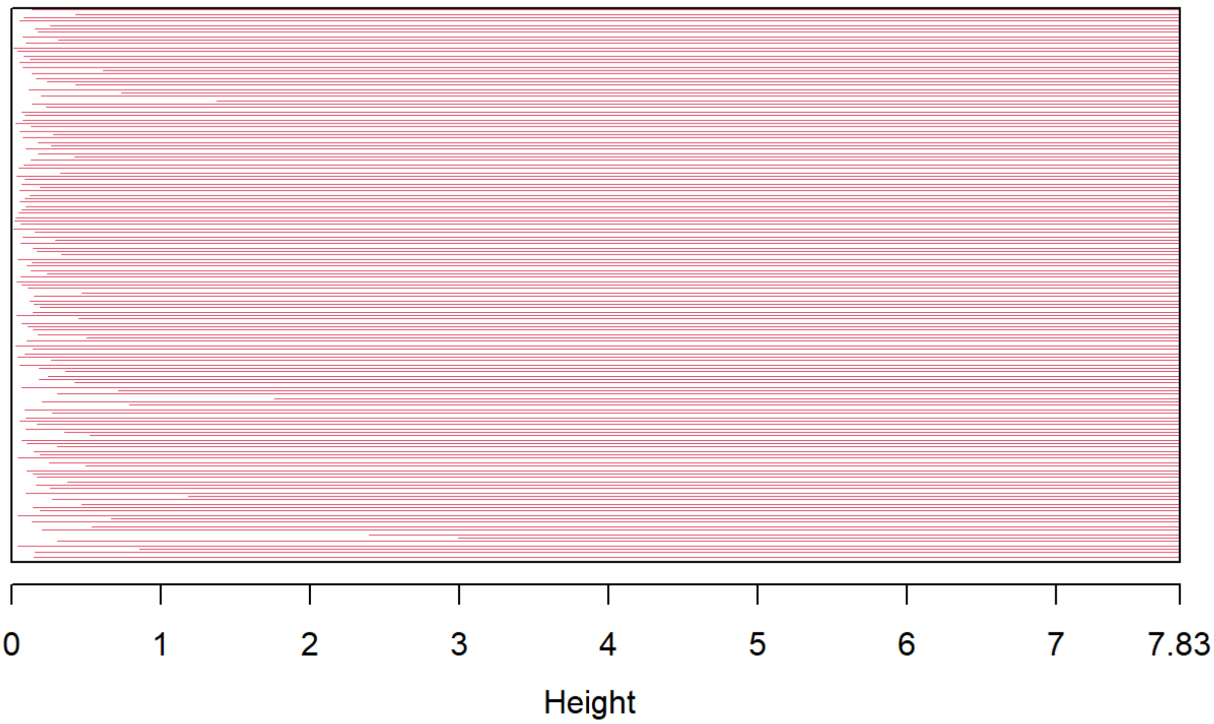
cluster::agnes()

Next in the analysis of the problem, agnes() function from the package cluster in R will be used for hierarchical clustering. The same reduced size data will be used for consistency of the analysis of hierarchical clustering algorithms.

Hide

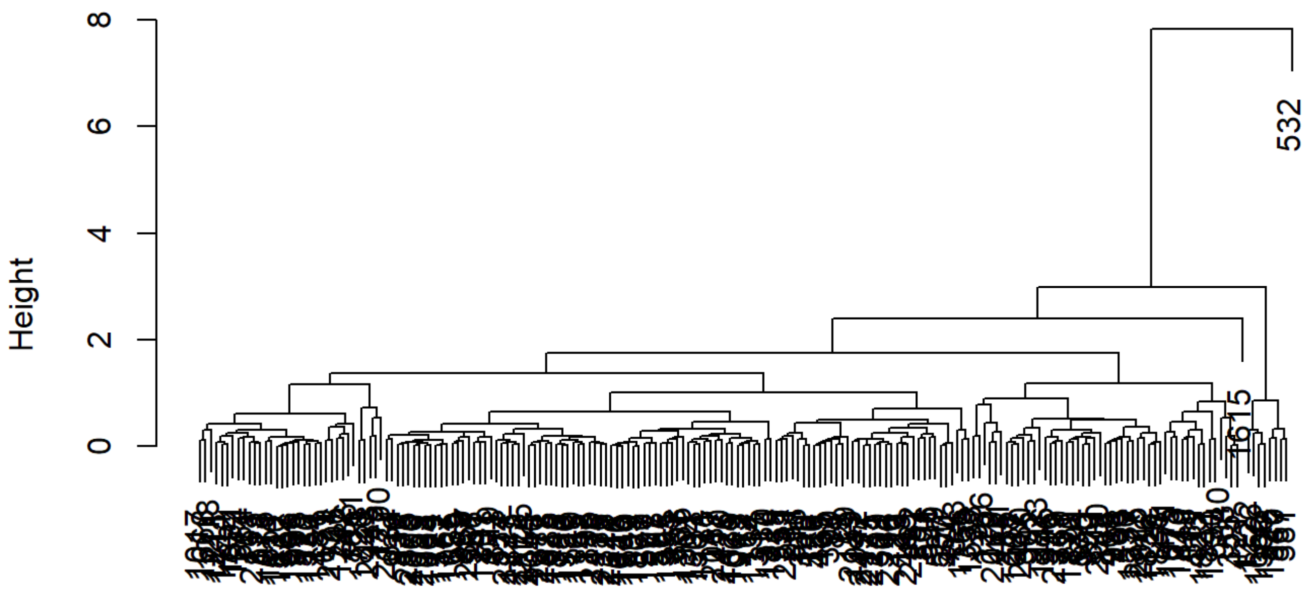
```
agnes_d <- cluster::agnes(reduced_scaled_d[-1])
plot(agnes_d)
```

Banner of cluster::agnes(x = reduced_scaled_d[-1])



Agglomerative Coefficient = 0.98

Dendrogram of cluster::agnes(x = reduced_scaled_d[-1])



reduced_scaled_d[-1]
Agglomerative Coefficient = 0.98

Hide

```
#coefficient for clustering
coef(agnes_d)
```

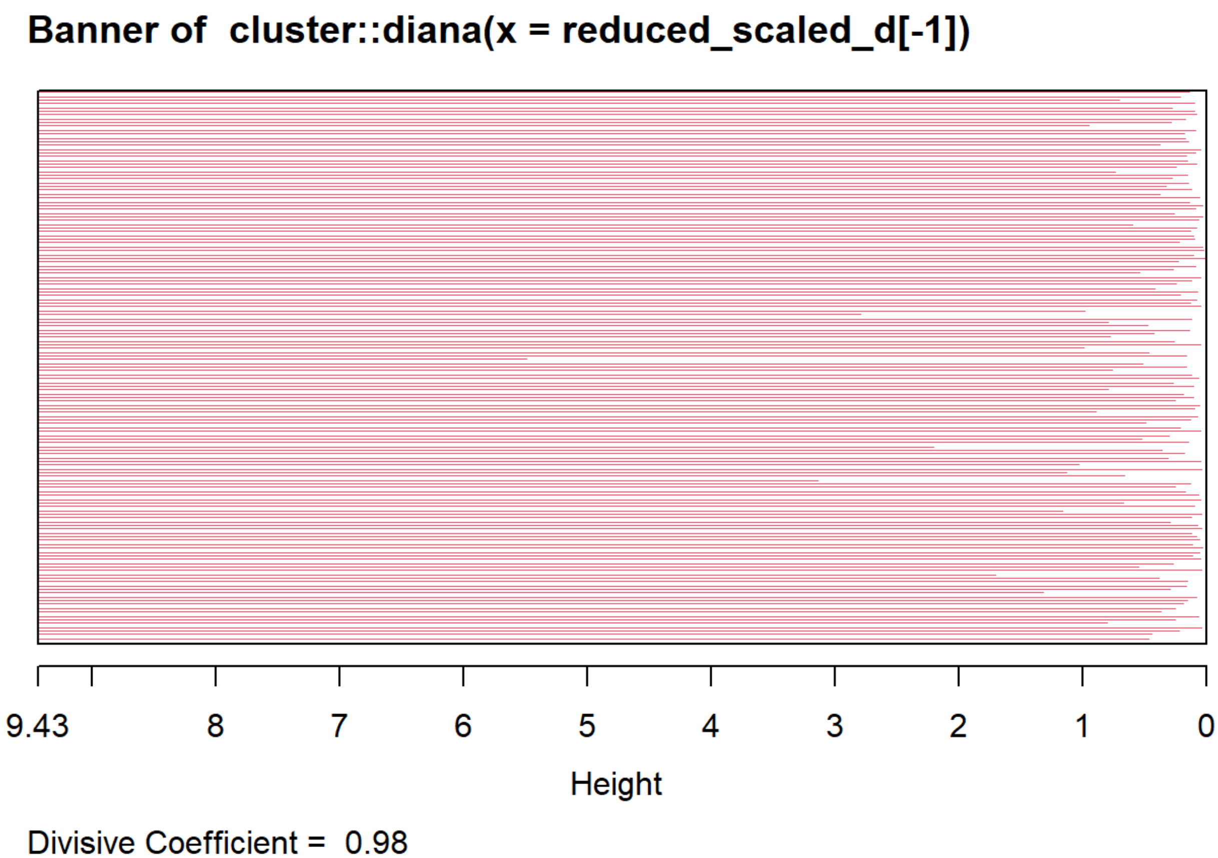
```
[1] 0.9810651
```

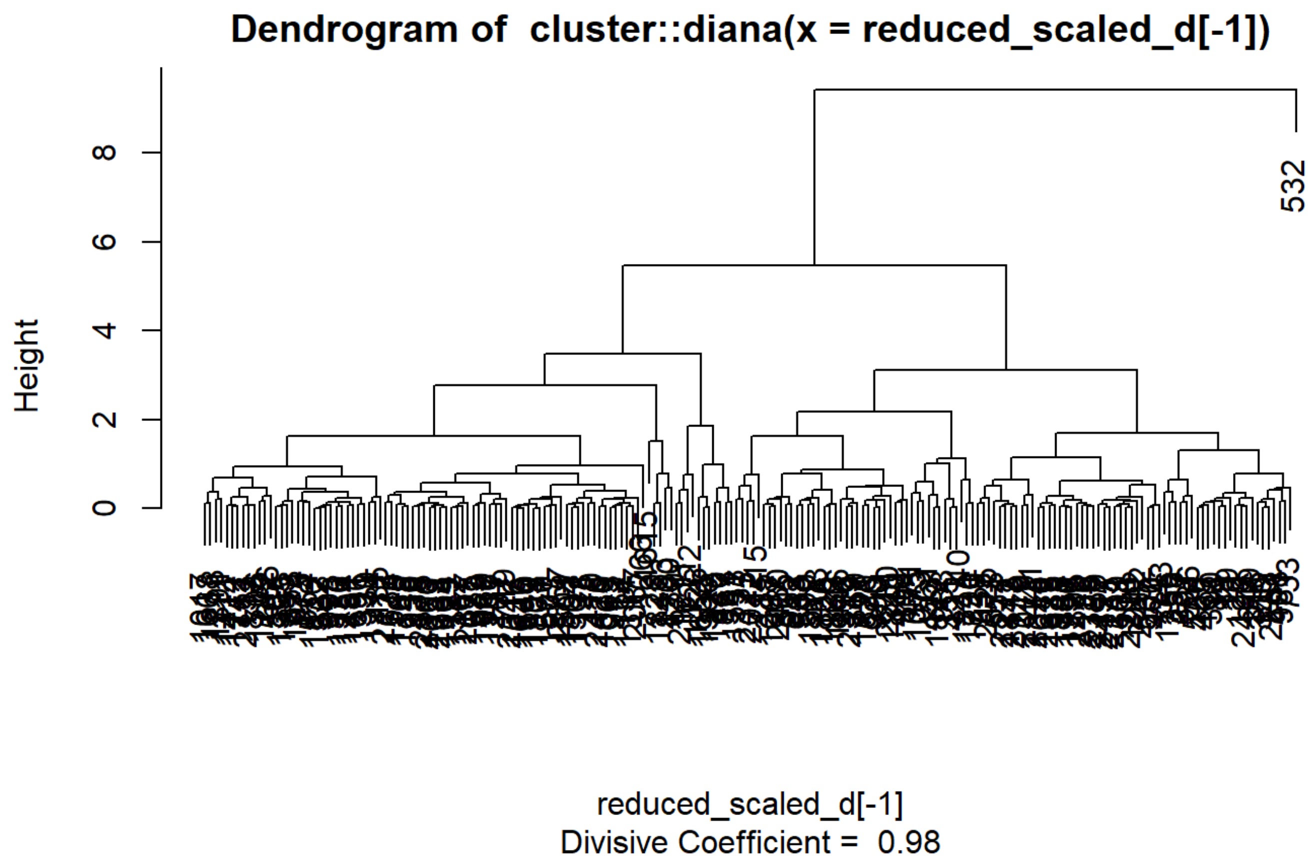
cluster::diana()

Next, we will use divisive method diana() that divides data in to smaller subsets

Hide

```
diana_d <- cluster::diana(reduced_scaled_d[-1])
plot(diana_d)
```





Hide

```
#coefficient for clustering
coef(diana_d)
```

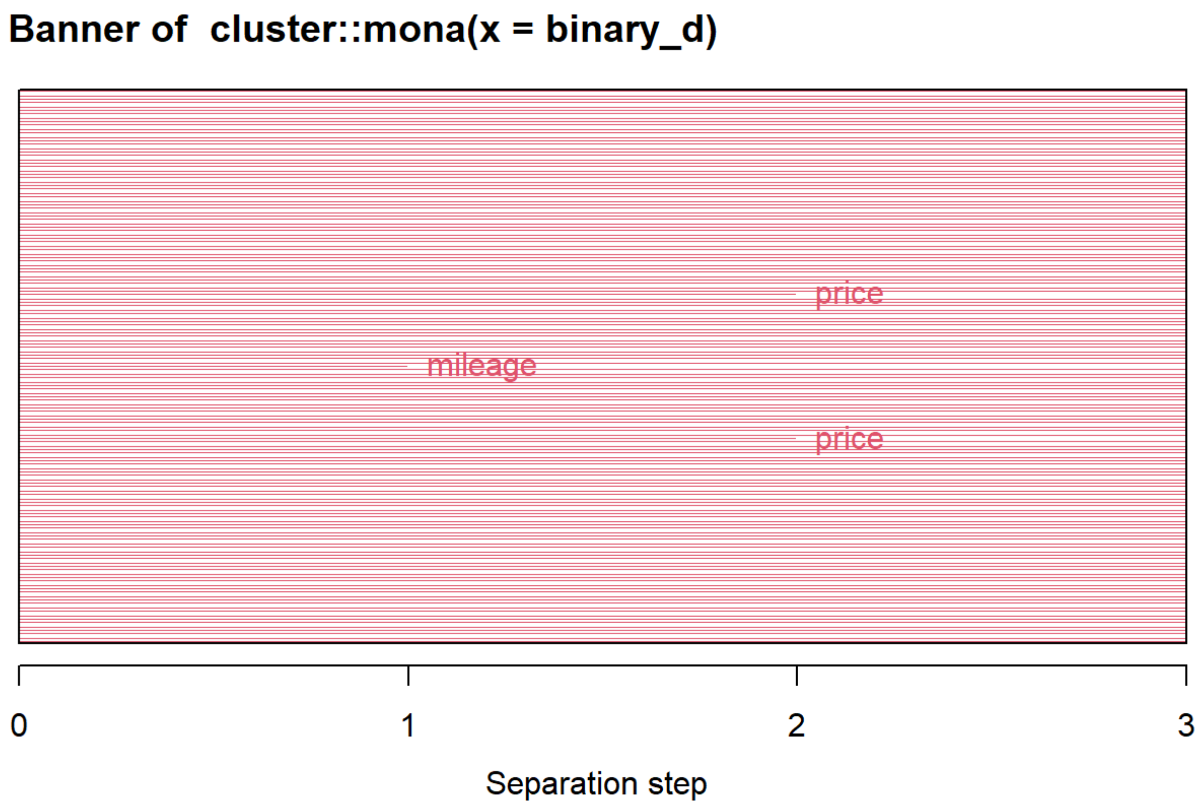
```
[1] 0.9818087
```

cluster::mona()

Next, we will perform clustering using mona() method which is specialized for binary datasets

Hide

```
binary_d <- reduced_scaled_d[-1]
for(j in 1:ncol(binary_d)) binary_d[,j] <- as.numeric(
  binary_d[,j] > median(binary_d[,j])
)
mona_d <- cluster::mona(binary_d)
plot(mona_d)
```



After performing hierarchical clustering using the methods from the cluster package in R, a good model for outlier detection can be hclust() using the ‘single’ method. This can be seen from the silhouette plot and the coefficient being the highest for the ‘single’ method. A good model for partitioning the data into approximately equal sized groups can be the hclust() using the ‘ward.D’ method. This can be seen from the silhoutte plot and the coefficient value being the highest for ‘ward.D’. However, hclust() using the ‘mcquitty’ method can also be a good model which does both the jobs, which can be seen from the silhouette method and a high coefficient value.