1 Hierarchial Clustering with R

Contents

1 Hierarchial Clustering with R **1**

1.1 Preliminaries : The Cars Data Set 2

1.2 Distance Matrices 4

1.3 Euclidean Distance 7

1.3.1 Example 7

1.4 Squared Euclidean Distance 8

1.4.1 Standardized Euclidean distance 9

1.5 Manhattan (City Block) Distance 12

1.5.1 Example 12

1.6 Cluster Analysis : Proximity Matrices 13

1.7 Standardization - The `scale` Function 15

1.8 Implementation of the Clustering Analysis 19

1.9 Agglomeration Methods used by `hclust` 20

1.10 Dendrograms 21

1.11 Using the `rect.hclust` Command 26

1.12 Using the `cutree` Command 28

1.1 Preliminaries : The Cars Data Set

To get an idea of what information we have in the ***cars*** data set, let's look at the first few records;

```
> tail(cars,3)
   Country                          Car  MPG Weight Drive_Ratio
36    U.S.              Ford Mustang 4 26.5  2.585        3.08
37   Japan            Honda Accord LX 29.5  2.135        3.05
38    U.S. Ford Country Squire Wagon 15.5  4.054        2.26

   Horsepower Displacement Cylinders
36         88          140         4
37         68           98         4
38        142          351         8
>
```

```
> summary(cars)
    Country                        Car          MPG            Weight
        : 0   AMC Concord D/L       : 1   Min.   :15.50   Min.   :1.915
 France : 1   AMC Spirit            : 1   1st Qu.:18.52   1st Qu.:2.208
 Germany: 5   Audi 5000             : 1   Median :24.25   Median :2.685
 Italy  : 1   BMW 320i              : 1   Mean   :24.76   Mean   :2.863
 Japan  : 7   Buick Century Special: 1   3rd Qu.:30.38   3rd Qu.:3.410
 Sweden : 2   Buick Estate Wagon    : 1   Max.   :37.30   Max.   :4.360
 U.S.   :22   (Other)               :32

  Drive_Ratio      Horsepower     Displacement     Cylinders
 Min.   :2.260   Min.   : 65.0   Min.   : 85.0   Min.   :4.000
 1st Qu.:2.695   1st Qu.: 78.5   1st Qu.:105.0   1st Qu.:4.000
 Median :3.080   Median :100.0   Median :148.5   Median :4.500
 Mean   :3.093   Mean   :101.7   Mean   :177.3   Mean   :5.395
 3rd Qu.:3.625   3rd Qu.:123.8   3rd Qu.:229.5   3rd Qu.:6.000
 Max.   :3.900   Max.   :155.0   Max.   :360.0   Max.   :8.000
```

1.2 Distance Matrices

In clustering analysis, an important step is calculating a **distance matrix**.
For a data set with n observations, the distance matrix will have n rows and n columns; the (i, j)th element of the distance matrix will be the "distance" between observation i and observation j.

- There are two functions that can be used to calculate distance matrices in `R`;
  - the `dist` function,
  - the `daisy` function, which is part of the ***cluster*** library.
- We'll use the `dist` function from now on in this example.
- Each function provides a choice of distance metrics; in this example, we'll use the default of ***Euclidean*** distance, but you may find that using other metrics will give different insights into the structure of your data.

```
cars.dist = dist(cars)
```

```
1          2          3          4          5          6
2  7.67122038
3  7.83892209 1.22909204
4  7.93329268 1.17056831 0.32148633
5  7.79786438 1.24693030 0.09318086 0.35340761
6  7.84972498 1.19468213 0.24363662 0.27248979 0.21846778
7  7.78735704 0.52646180 1.16777805 0.98623148 1.17782394 1.0
...........
```

Remarks on the Distance Matrix

- If you display the distance matrix in `R` (for example, by typing its name), you'll notice that only the lower triangle of the matrix is displayed.
- This is to remind us that the distance matrix is symmetric, since it doesn't matter which observation we consider first when we calculate a distance. `R` takes advantage of this fact by only storing the lower triangle of the distance matrix.
- All of the clustering functions will recognize this and have no problems, but if you try to access the distance matrix in the usual way (for example, with subscripting), you'll see an error message.
- If you need to use the distance matrix with anything other than the clustering functions, you'll need to use `as.matrix()` to convert it to a regular matrix.

1.3 Euclidean Distance

The Euclidean Distance between two points, x and y, with k dimensions is calculated as:

$$\sqrt{\sum_{j=1}^{k}(x_j - y_j)^2}$$

The Euclidean distance is always greater than or equal to zero. The measurement would be zero for identical points and high for points that show little similarity.

1.3.1 Example

Compute the Euclidean Distance between the following points: $X = \{1, 5, 4, 3\}$ and $Y = \{2, 1, 8, 7\}$

x_j	y_j	$x_j - y_j$	$(x_j - y_j)^2$
1	2	-1	1
5	1	4	16
4	8	-4	16
3	7	-4	16
			49

The Euclidean Distance between the two points is $\sqrt{49}$ i.e. 7.

1.4 Squared Euclidean Distance

The Squared Euclidean distance between two points, x and y, with k dimensions is calculated as:

$$\sum_{j=1}^{k}(x_j - y_j)^2$$

The Squared Euclidean distance may be preferred to the Euclidean distance as it is slightly less computational complex, without loss of any information.

1.4.1 Standardized Euclidean distance

Let us consider measuring the distances between between two points using the three continuous variables pollution, depth and temperature. Let us suppose that a difference of 4.1 in terms of pollution is considered quite large and unusual, while a difference of 48 in terms of depth is large, but not particularly unusual. What would happen if we applied the Euclidean distance formula to measure distance between two cases.

Variables	case 1	case 2
Pollution	6.0	1.9
Depth	51	99
Temp	3.0	2.9

Here is the calculation for Euclidean Distance:

$$d = \sqrt{(6.0 - 1.9)^2 + (51 - 99)^2 + (3.0 - 2.9)^2}$$

$$d = \sqrt{16.81 + 2304 + 0.01} = \sqrt{2320.82} = 48.17$$

The contribution of the second variable depth to this calculation is huge one could say that the distance is practically just the absolute difference in the depth values (equal to $|51 - 99| = 48$) with only tiny additional contributions from pollution and temperature.

These three variables are on completely different scales of measurement and the larger depth values have larger differences, so they will dominate in the calculation of Euclidean distances.

- The approach to take here is **standardization**, which is is necessary to balance out the contributions, and the conventional way to do this is to transform the variables so they all have the same variance of 1.
- At the same time we ***center*** the variables at their means this centering is not necessary for calculating distance, but it makes the variables all have mean zero and thus easier to compare.
- The transformation commonly called standardization is thus as follows:

$$\text{standardized value} = \frac{\text{observed value} \;\; \text{mean}}{\text{standard deviation}}$$

Variables	Case 1	Case 2	Mean	Std. Dev	Case 1 (std)	Case 2 (std)
Pollution	6.0	1.9	4.517	2.141	0.693	-1.222
Depth	51	99	74.433	15.615	-1.501	1.573
Temp	3.0	2.9	3.057	0.281	-0.201	-0.557

$$d_{std} = \sqrt{(0.693 - (-1.222))^2 + (-1.501 - 1.573)^2 + (-0.201 - (-0.557))^2}$$

$$d_{std} = \sqrt{3.667 + 9.449 + 0.127} = \sqrt{13.243} = 3.639$$

- Pollution and temperature have higher contributions than before but depth still plays the largest role in this particular example, even after standardization.
- But this contribution is justified now, since it does show the biggest standardized difference between the samples.

1.5 Manhattan (City Block) Distance

The City block distance between two points, x and y, with k dimensions is calculated as:

$$\sum_{j=1}^{k} |x_j - y_j|$$

The City block distance is always greater than or equal to zero. The measurement would be zero for identical points and high for points that show little similarity.

1.5.1 Example

Compute the Manhattan Distance between the following points: $X = \{1, 3, 4, 2\}$ and $Y = \{5, 2, 5, 2\}$

x_j	y_j	$x_j - y_j$	$\lvert x_j - y_j \rvert$
1	5	-4	4
3	2	1	1
4	5	-1	1
2	2	0	0
			6

The Manhattan Distance between the two points is 6.

1.6 Cluster Analysis : Proximity Matrices

Using ***nearest neighbour*** linkage, describe how the agglomeration schedule based on the following proximity matrix. With nearest neighbour, a case is assigned to the cluster of the case with which it has the shortest distance. Cluster are also joined on this basis.

Case	1	2	3	4	5	6	7	8	9	10
1	0.00	**4.82**	89.39	85.97	46.26	71.87	56.42	23.75	31.57	11.70
2	**4.82**	0.00	94.24	38.96	**5.55**	35.07	74.52	71.27	61.84	**4.84**
3	89.39	94.24	0.00	57.65	27.27	25.31	20.89	**2.84**	63.50	89.39
4	85.97	38.96	57.65	0.00	**22.94**	**7.13**	70.49	23.09	**12.75**	85.97
5	46.26	**5.55**	27.27	**22.94**	0.00	39.44	17.43	79.22	14.47	46.26
6	71.87	35.07	25.31	**7.13**	39.44	0.00	27.50	30.65	13.34	71.87
7	56.42	74.52	20.89	70.49	17.43	27.50	0.00	91.16	44.92	**6.42**
8	23.75	71.27	**2.84**	23.09	79.22	30.65	91.16	0.00	**3.18**	23.75
9	31.57	61.84	63.50	**12.75**	14.47	13.34	44.92	**3.18**	0.00	31.57
10	11.70	**4.84**	89.39	85.97	46.26	71.87	**6.42**	23.75	31.57	0.00

- The closest pair in terms of distance (2.84) are cases 3 and 8. So this is the first linkage.
- The next closest pair (3.18) are 8 and 9. The next linkage joins case 9 to 3 and 8.
- The next closest pair (4.82) are 1 and 2. So this is the next linkage. [So far (3,8,9) and (2,10)]
- The next closest pair (4.84) are 2 and 10. The next linkage joins case 1 to 2 and 10.

- The next closest pair (5.55) are 2 and 5. The next linkage joins case 5 to 1, 2 and 10. [So far (3,8,9) and (1,2,5,10)]
- The next closest pair (6.42) are 7 and 10. The next linkage joins case 7 to 1, 2, 5 and 10.
- The next closest pair (7.13) are 4 and 6. The next linkage joins case 4 to 6. [So far (3,8,9), (4,6) and (1,2,5,10) All cases are in clusters. This is a 3 cluster solution.]
- The next closest pair (11.70) are 1 and 10. Disregard, because they are already clustered together.
- The next closest pair (19.44) are 4 and 9. This joins cluster (4,6) to cluster (3,8,9) [So far (3,4,6,8,9) and (1,2,5,10). This is a 2 cluster solution.]
- The next closest pairing is 4 and 5. This linkage joins all cases together in one cluster.

1.7 Standardization - The `scale` Function

- It is clear that the variables are measured on different scales, so we will likely want to standardize the data before proceeding.
- The `daisy` function in the ***cluster*** library will automatically perform standardization, but it doesn't allow the user complete control.

- If you have a particular method of standardization in mind, you can use the `scale` function. You pass to the scale function a matrix or data frame to be standardized, and two optional vectors.
- The first, called `center`, is a vector of values, one for each column of the matrix or data frame to be standardized, which will be subtracted from every entry in that column.
- The second, called `scale`, is similar to `center`, but is used to divide the values in each column. In order to get z-scores, you could pass scale a vector of means for center, and a vector of standard deviations for scale.
- **Defaults**: The default settings for `center` and `scale` are the mean and standard deviation respectively.
  However, implicit in this choice is the ***assumption of normally distributed data***.

- These vectors can be created with the `apply` function, that performs the same operation on each row or column of a matrix.

- *(Remark, for a column-wise operation with the* ***`apply`*** *function, specify the addition argument "2", Otherwise it would work on a row-wise basis)*

- Non-numeric variables, such as the country of origin and name of the car, will not be useful in the cluster analysis, so they have been removed.
  The number of cylinders has been retained as a variable.

The `scale()` function - Example:
Suppose we want to standardize by subtracting the median (`median`) and dividing by the mean average deviation (`mad`):

```
# First, remove the first two (non-numeric) columns.
cars.use = cars[,-c(1,2)]

# Compute the medians of each column
medians = apply(cars.use,2,median)

# Compute the MADs of each column
mads = apply(cars.use,2,mad)

# Use the scale function
cars.use = scale(cars.use,center=medians,scale=mads)
```

```
> head(cars.use)
         MPG     Weight Drive_Ratio Horsepower Displacement
1 -0.8402554  2.0921704  -0.4918162  1.5785954    2.6912849
2  1.2403771 -0.9617739  -0.1545708 -0.5740347   -0.6744908
3  0.8745516 -0.9492833   0.9836324 -0.8323503   -0.7946970
4  1.1260566 -0.8868304   0.9133729 -1.0045607   -0.8347658
5  0.8288234 -0.8680946   0.9836324 -0.8323503   -0.7946970
6  0.8631195 -0.8306229   0.8712172 -1.0045607   -0.8481220
   Cylinders
1  4.7214353
2 -0.6744908
3 -0.6744908
4 -0.6744908
5 -0.6744908
6 -0.6744908
```

- We will continue the workshop, using both scaled and unscaled data, for the sake of simplicity.

1.8 Implementation of the Clustering Analysis

To get started, we'll use the `hclust` method. The argument to this function is the distance matrix that we have created earlier.

```
cars.hclust = hclust(cars.dist)

cars.hclust

names(cars.hclust)
```

- We are going to use the default method of `hclust`, which is to update the distance matrix using what `R` calls ***complete linkage***.
- Using this method, when a cluster is formed, its distance to other objects is computed as the maximum distance between any object in the cluster and the other object.
- Other linkage methods will provide different solutions, and **should not** be ignored.
- For example, using ***Ward's Linkage*** (`method=ward`) tends to produce clusters of fairly equal size, and can be useful when other methods find clusters that contain just a few observations.

1.9 Agglomeration Methods used by `hclust`

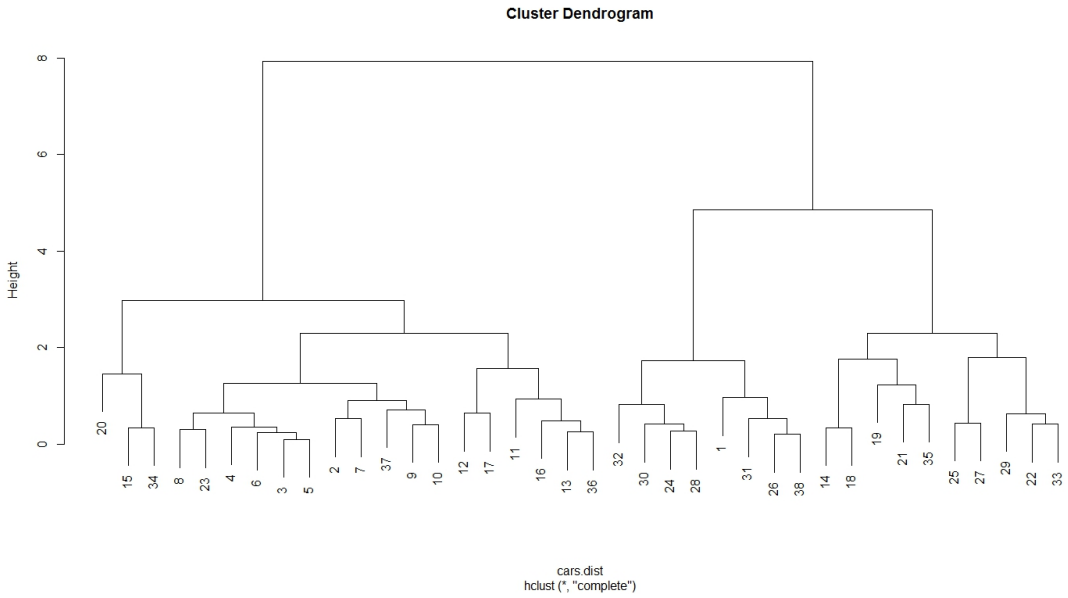
- The agglomeration method used by **`hclust`** can be specified using the **`method=`** argument.
- The default method is **`complete linkage`**.
- The other agglomeration methods are

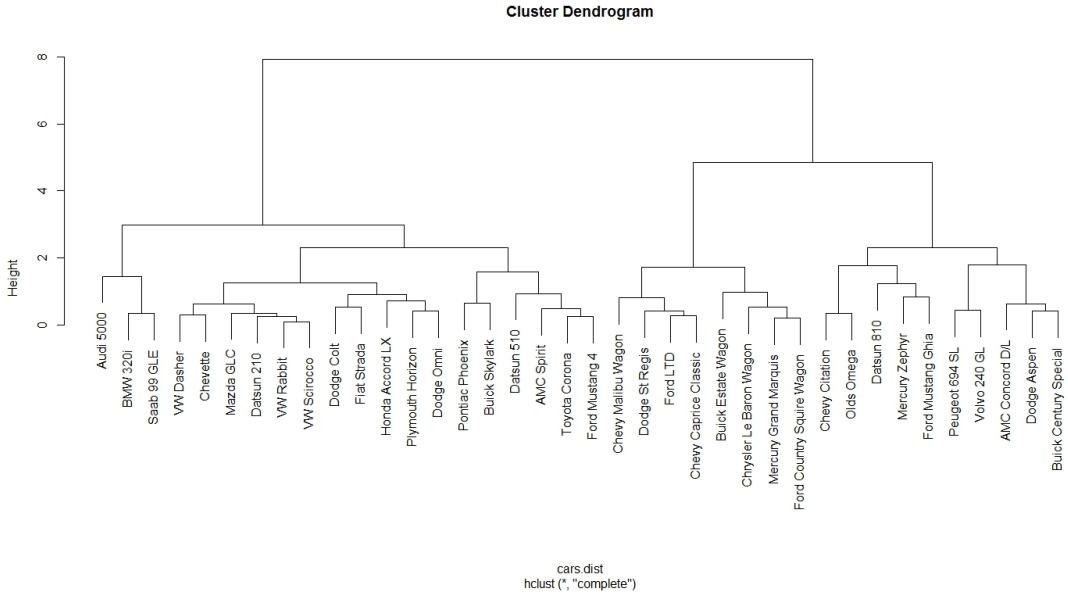
`single`	: nearest neighbor
`complete`	: furthest neighbor or compact (*default*)
`ward.D`	: Ward's minimum variance method
`ward.D2`	: Ward's minimum variance method (*alternative computation*)
`mcquitty`	: McQuitty's method
`average`	: average similarity
`median`	: median (as opposed to average) similarity
`centroid`	: geometric centroid
`flexible`	: flexible Beta

1.10 Dendrograms

- The main graphical tool for looking at a hierarchical cluster solution is known as a dendrogram.
- A dendrogram is a tree-structured graph used in heat maps to visualize the result of a hierarchical clustering calculation.
- The dendrogram lists the objects which are clustered along the x-axis, and the distance at which the cluster was formed along the y-axis.
  (Distances along the x-axis are not particularly meaningful in a dendrogram; the observations are often equally spaced to make the dendogram easier to read.)
- The result of a clustering is presented either as the distance or the similarity between the clustered rows or columns depending on the selected distance measure.
- When you use `hclust` (or `agnes`) to perform a cluster analysis, you can see the dendrogram by passing the result of the clustering to the `plot` function.
- To illustrate interpretation of the dendrogram, we'll look at a cluster analysis performed on the cars data set.

```
plot(cars.hclust)
plot(cars.hclust,labels=cars$Car)
```





- If you choose any height along the y-axis of the dendrogram, and move across the dendrogram counting the number of lines that

you cross, each line represents a group that was identified when objects were joined together into clusters.

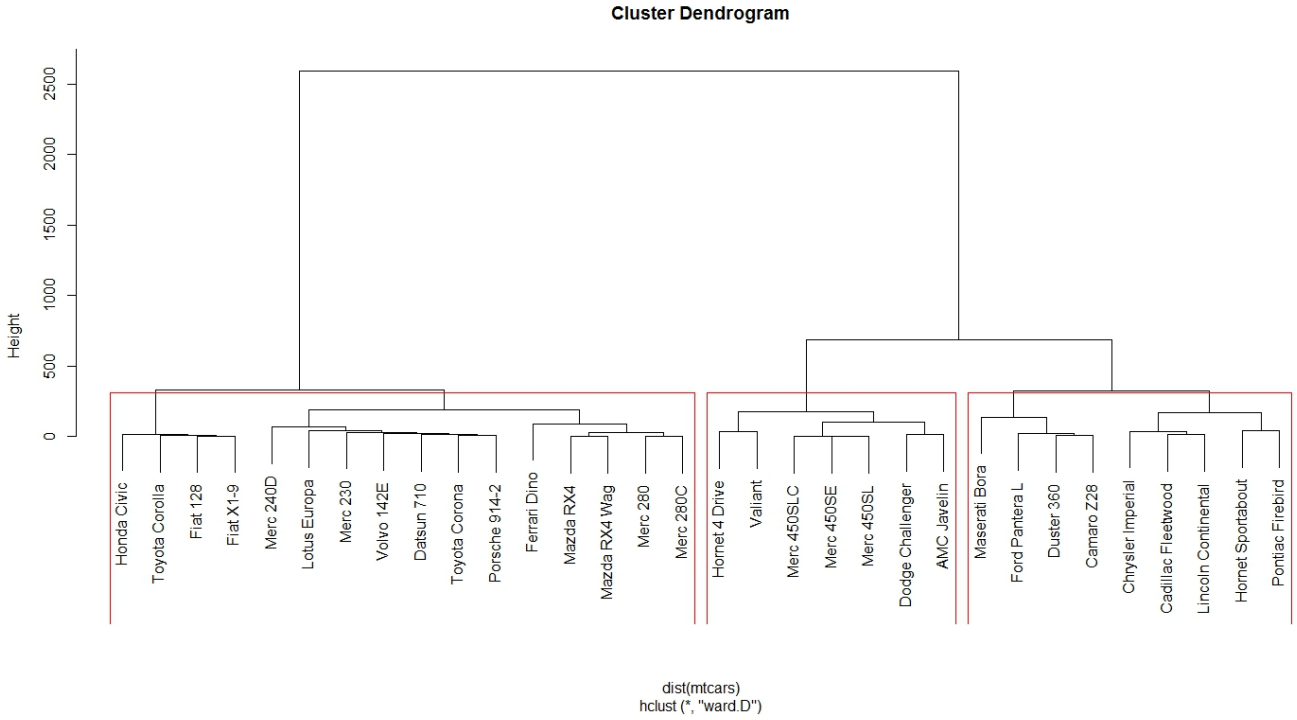
- The observations in that group are represented by the branches of the dendrogram that spread out below the line.
- For example, if we look at a height of 6, and move across the x-axis at that height, we'll cross two lines. That defines a two-cluster solution; by following the line down through all its branches, we can see the names of the cars that are included in these two clusters.
- Since the y-axis represents how close together observations were when they were merged into clusters, clusters whose branches are very close together (in terms of the heights at which they were merged) probably aren't very reliable. But if there's a big difference along the y-axis between the last merged cluster and the currently merged one, that indicates that the clusters formed are probably doing a good job in showing us the structure of the data.

- Looking at the dendrogram for the car data, there are clearly two very distinct groups; the right hand group seems to consist of two more distinct cluster, while most of the observations in the left hand group are clustering together at about the same height.
- For this data set, it looks like either two or three groups might be an interesting place to start investigating.
- This is not to imply that looking at solutions with more clusters would be meaningless, but the data seems to suggest that two or three clusters might be a good start.
- For a problem of this size, we can see the names of the cars, so we could start interpreting the results immediately from the dendrogram, but when there are larger numbers of observations, this won't be practical.

1.11 Using the `rect.hclust` Command

- This command is used to draw rectangles around the branches of a dendrogram highlighting the corresponding clusters. First the dendrogram is cut at a certain level, then a rectangle is drawn around selected branches.
- **Simple Demontration,**
  - using cars (unscaled)
  - Linkage Method : Ward's Linkage

```
plot(hclust(cars.dist,method="ward.D"))
rect.hclust(hclust(cars.dist,method="ward.D"),h=300)
```


Cluster Dendrogram
Height
0
500
1000
1500
2000
2500
Honda Civic
Toyota Corolla
Fiat 128
Fiat X1-9
Merc 240D
Lotus Europa
Merc 230
Volvo 142E
Datsun 710
Toyota Corona
Porsche 914-2
Ferrari Dino
Mazda RX4
Mazda RX4 Wag
Merc 280
Merc 280C
Hornet 4 Drive
Valiant
Merc 450SLC
Merc 450SE
Merc 450SL
Dodge Challenger
AMC Javelin
Maserati Bora
Ford Pantera L
Duster 360
Camaro Z28
Chrysler Imperial
Cadillac Fleetwood
Lincoln Continental
Hornet Sportabout
Pontiac Firebird
dist(mtcars)
hclust (*, "ward.D")

1.12 Using the `cutree` Command

- One of the first things we can look at is how many cars are in each of the groups. Suppose that we would like to do this for both the two cluster and three cluster solutions.
- You can create a vector showing the cluster membership of each observation by using the **`cutree`** function.
- Since the object returned by a hierarchical cluster analysis contains information about solutions with different numbers of clusters, we pass the **`cutree`** function the cluster object and the number of clusters we're interested in.
- So to get cluster memberships for the three cluster solution, we could use:

```
groups.3 = cutree(cars.hclust,3)
```

```
> groups.3 = cutree(cars.hclust,3)
>
> groups.3
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
 1  2  2  2  2  2  2  2  2  2  2  2  2  3  2
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 2  2  3  3  2  3  3  2  1  3  1  3  1  3  1
31 32 33 34 35 36 37 38
 1  1  3  2  3  2  2  1
>
> table(groups.3)
groups.3
 1  2  3
 8 20 10
>
> groups.4 = cutree(cars.hclust,4)
>
> table(groups.4)
groups.4
 1  2  3  4
 8 17 10  3
```

Automating the Process using `sapply`

- We'd like a solution where there aren't too many clusters with just a few observations, because it may make it difficult to interpret our results.
- For the three cluster solution, the distribution among the clusters looks satisfactory.
- You can get this information for many different groupings at once by combining the calls to `cutree()` and table in a call to `sapply()`.
- For example, to see the sizes of the clusters for solutions ranging from 2 to 6 clusters, we could use:

```
ClusSize=function(ncl=2) {
 table(cutree(cars.hclust,ncl))
 }
counts = sapply(2:6,ClusSize)
names(counts) = 2:6
counts
```

```
$"2"

 1  2
18 20

$"3"

 1  2  3
 8 20 10


......
```

Cluster Membership

- To see which cars are in which clusters, we can use subscripting on the vector of car names to choose just the observations from a particular cluster.
- Since we used all of the observations in the data set to form the distance matrix, the ordering of the names in the original data will coincide with the values returned by cutree.
- If observations were removed from the data before the distance matrix is computed, it's important to remember to make the same deletions in the vector from the original data set that will be used to identify observations.
- So, to see which cars were in the first cluster for the four cluster solution, we can use:

```
> cars$Car[groups.3 == 1]
[1] Buick Estate Wagon        Ford Country Squire Wagon
[3] Chevy Malibu Wagon        Chrysler LeBaron Wagon
[5] Chevy Caprice Classic     Ford LTD
[7] Mercury Grand Marquis     Dodge St Regis
```

Cross-Tabulations

- Cluster analysis is often performed to see if observations naturally group themselves in accord with some already measured variable.
- For the ***cars*** data set, we could ask whether the clusters reflect the country of origin of the cars, stored in the variable `Country` in the original data set.
- The `table` function can be used, this time passing two arguments, to produce a cross-tabulation of cluster group membership and country of origin:

```
> table(groups.3,cars$Country)

groups.3 France Germany Italy Japan Sweden U.S.
       1      0       0     0     0      0    8
       2      0       5     1     6      1    7
       3      1       0     0     1      1    7
>
```