Market Segmentation Using K-Means Cluster Analysis

Harry B. Rowe

March 26, 2012

A “market segment” refers to a group of potential buyers for a category of product or service whose needs are similar. A member of a market segment has needs that are more similar to the needs of another member of the same segment than to the needs of a member of a different market segment. An example of market segments for cars might be “luxury car buyers”, “performance car buyers”, and “economy car buyers”.

Clearly, knowledge of market segments allows producers to target their offerings exactly for the members of specific segments, resulting in higher sales and greater customer satisfaction.

The problem for organizations trying to understand the needs of their customers is that they do not know in advance how many market segments there are, and how the needs of those segments differ from one another. Although some market segments may align with specific demographic groups (luxury buyers tend to come from upper income groups), others may not (“environmentally conscious buyers” may come from all income groups).

K-means cluster analysis is a technique for taking a mass of raw data and dividing it into groups that are more similar within groups than between groups. An explanation of how it works is beyond the scope of this article (and beyond the capability of this author). But software libraries and statistical packages exist which allow k-means analysis to be carried out without full knowledge of its implementation.

The package used for this demonstration is R, an open-source statistical environment available for download from www.r-project.org. It is available for Windows, Apple Macintosh, and Linux.

Figure 1 shows an x-y scatter plot of a data set containing 100 (x,y) pairs. It is easy to see that the data points group into four clusters.

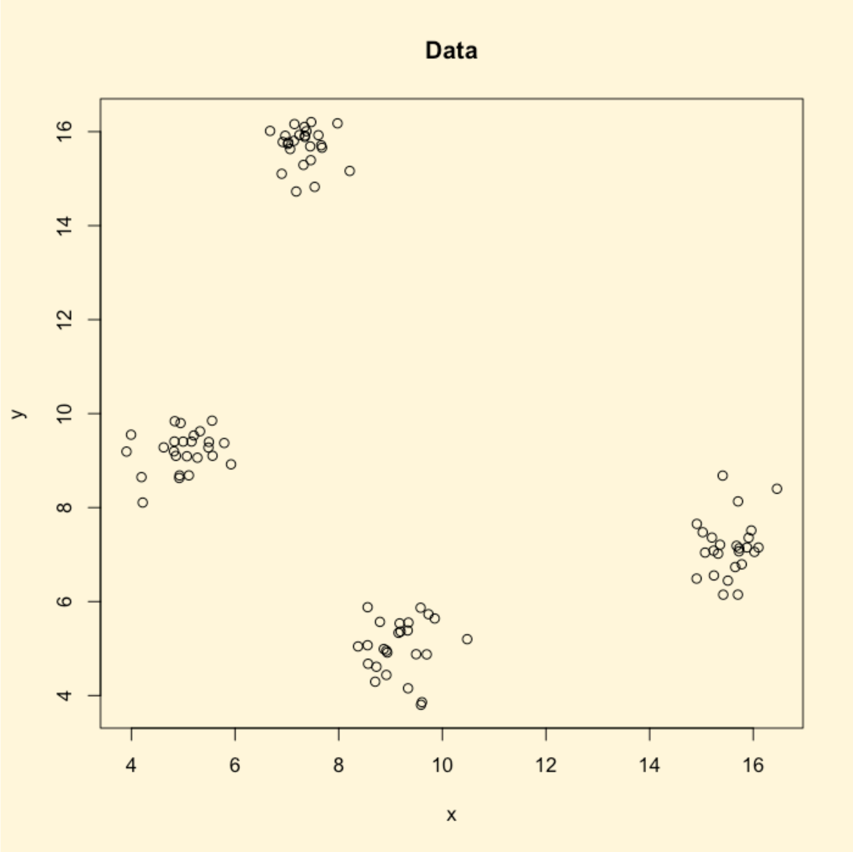

Figure 1

While it is easy to see the clusters, it is at best tedious, and at worst impossible, to determine which data values fall into which clusters. It is also tedious to locate the center of the clusters.

With R, however, this can be done relatively easily.

With the data for the x and y values stored in the (vector) variables x and y, we issue the following commands. (The lines with "#" are comments and may be omitted.)

```
#
# Create a 2-d data matrix
#
   d = cbind( x, y )
#
# Find four cluster centers in the data set d, using the
# Hartigan-Wong algorithm, selecting the best fit from
# among 10 random starts.
#
   km = kmeans( d, centers=4, algorithm="Har", nstart=10 )
#
# Print the returned object
#
   print( km )
```

```
#
```

The program thinks for a fraction of a second, and returns the following information:

```
K-means clustering with 4 clusters of sizes 25, 25, 25, 25

Cluster means:
          x         y
1  9.180482  5.025350
2  7.322979 15.700404
3  5.001532  9.206401
4 15.556767  7.159617

Clustering vector:
  [1] 1 2 1 2 4 3 2 3 2 2 1 4 1 1 2 3 3 4 4 1 1 3 4 3 3 2 4 2 1 2 1 4 1
3
 [35] 4 3 4 1 3 1 2 2 3 4 1 4 4 2 2 3 2 4 1 1 2 2 2 1 4 4 4 1 3 2 3 2 1
4
 [69] 3 4 2 4 3 1 3 1 2 3 1 3 2 1 3 3 1 1 2 3 4 3 4 4 3 4 1 4 3 4 2 2

Within cluster sum of squares by cluster:
[1] 14.434937  6.801739 10.880267 12.817525

Available components:
[1] "cluster"  "centers"  "withinss" "size"
```

It is probably obvious that the four “Cluster means” are the locations of the centers of each of the four clusters.

The “Clustering vector” is a list of numbers corresponding to the rows of the input data, specifying the cluster into which that row was assigned.

The “Within cluster sum of squares by cluster” is a measure of dispersion calculated for each cluster by summing the squares of the distances of each point in a cluster from the center of the cluster. The more points in the cluster, and the more spread out they are, the larger the value will be. These numbers are also called the “sum squared errors” or “SSE”.

Figure 2 shows the data with the calculated cluster centers plotted in red. As you can see, the calculation does a pretty good job of locating the centers.

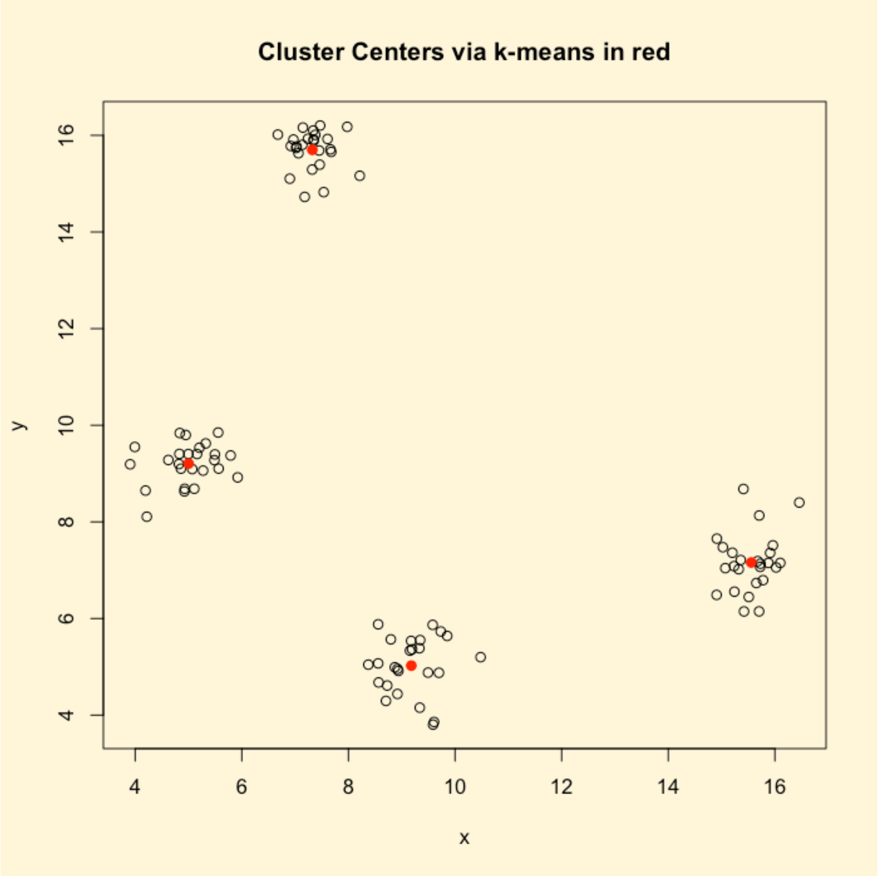

Figure 2

The algorithm is not limited to two dimensions, however. K-means can be used with data with an arbitrary number of dimensions. To keep the example to something we can easily visualize, we add a z dimension to our dataset and plot the data as in figure 3.

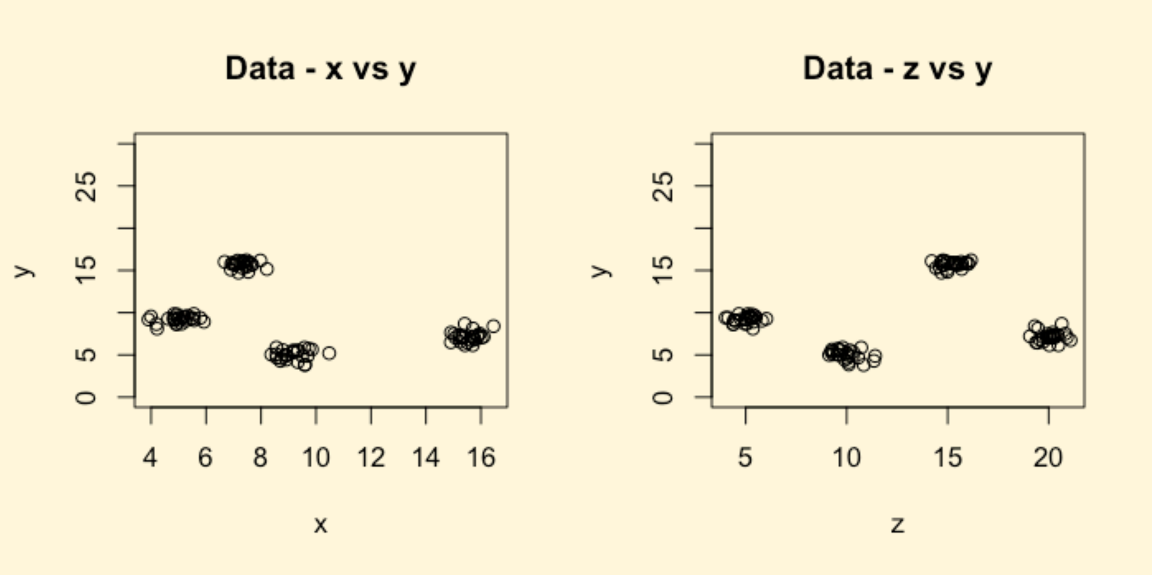

Figure 3

In R, we use a very similar set of commands to create and analyze the three-dimensional data set.

```
#
# Now add a third dimension
#
   d2 = cbind( x, y, z )
#
#
# Find four cluster centers for 3-d object
#
   km = kmeans( d2, centers=4, algorithm="Har", nstart=10 )
#
# Print the returned object
#
   print( km )
#
```

And we get a result in exactly the same form, but with cluster centers in three dimensions.

```
K-means clustering with 4 clusters of sizes 25, 25, 25, 25

Cluster means:
          x         y        z
1  9.180482  5.025350 10.00090
2  7.322979 15.700404 15.20269
3  5.001532  9.206401  4.98175
4 15.556767  7.159617 20.05480

Clustering vector:
  [1] 1 2 1 2 4 3 2 3 2 2 1 4 1 1 2 3 3 4 4 1 1 3 4 3 3 2 4 2 1 2 1 4 1
3
 [35] 4 3 4 1 3 1 2 2 3 4 1 4 4 2 2 3 2 4 1 1 2 2 2 1 4 4 4 1 3 2 3 2 1
4
 [69] 3 4 2 4 3 1 3 1 2 3 1 3 2 1 3 3 1 1 2 3 4 3 4 4 3 4 1 4 3 4 2 2

Within cluster sum of squares by cluster:
[1] 23.67429 13.89660 17.23296 19.24907

Available components:
[1] "cluster"  "centers"  "withinss" "size"
```

And the locations of the centers are shown in figure 4.

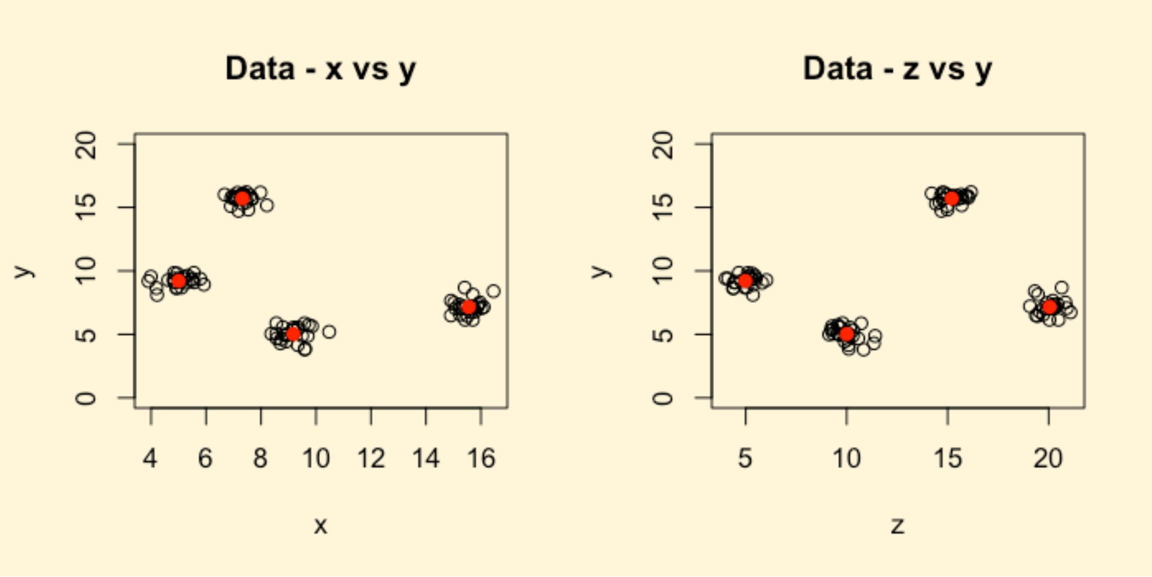

Figure 4

While it is somewhat more difficult, even in three dimensions we can still plot the data in a way that clearly shows the number and general location of the clusters. But what about four, five, or more dimensions?

While it is very difficult for humans to visualize data in more than three dimensions, added dimensions present no such difficulty for the k-means algorithm. Simply add the data as additional columns and execute the algorithm.

One problem with k-means analysis however is that the user must specify the number of clusters into which to divide the data. What if we don't know how many clusters there are? We usually won't.

Figure 5 shows what happens if we ask the k-means algorithm to find three clusters in our 2-D dataset.

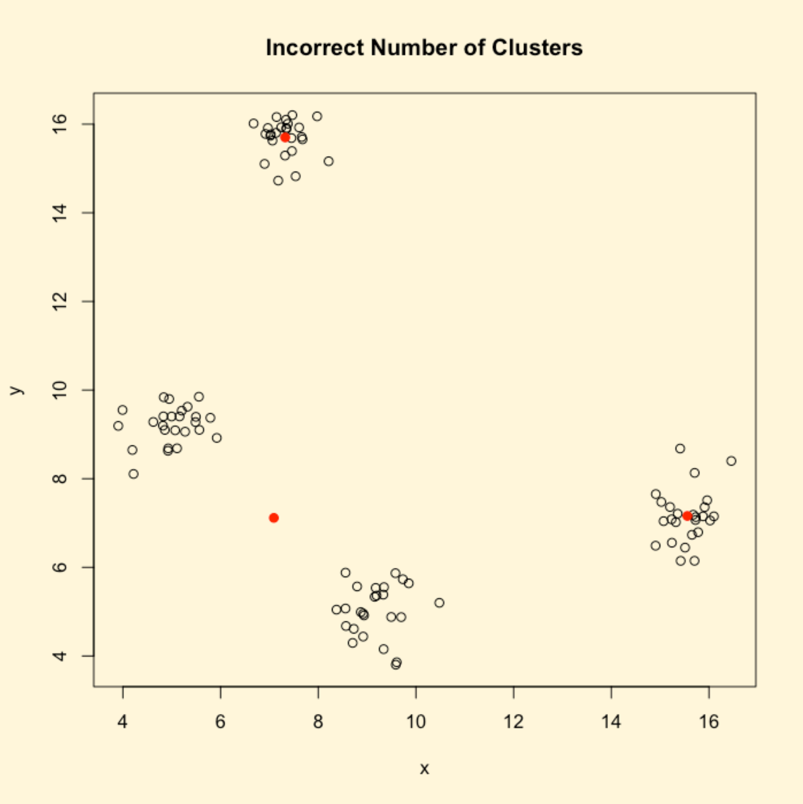

Figure 5

As you can see, the algorithm obligingly finds three clusters when our eyes easily distinguish four. Fortunately, there is a technique to reduce, but not eliminate, the difficulty.

The answer lies in the within cluster sum of squares values reported by the k-means algorithm. If we define an objective function that is the sum of the SSE's over all the clusters in a particular set of n clusters, we can investigate how that function changes with number of clusters.

Figure 6 is a plot of this function for our 2-D data for between two and seven clusters. The values on the plot are obtained by executing the k-means algorithm once for two clusters, three clusters, four clusters, and so on. Then the reported SSE values for each is summed and plotted.

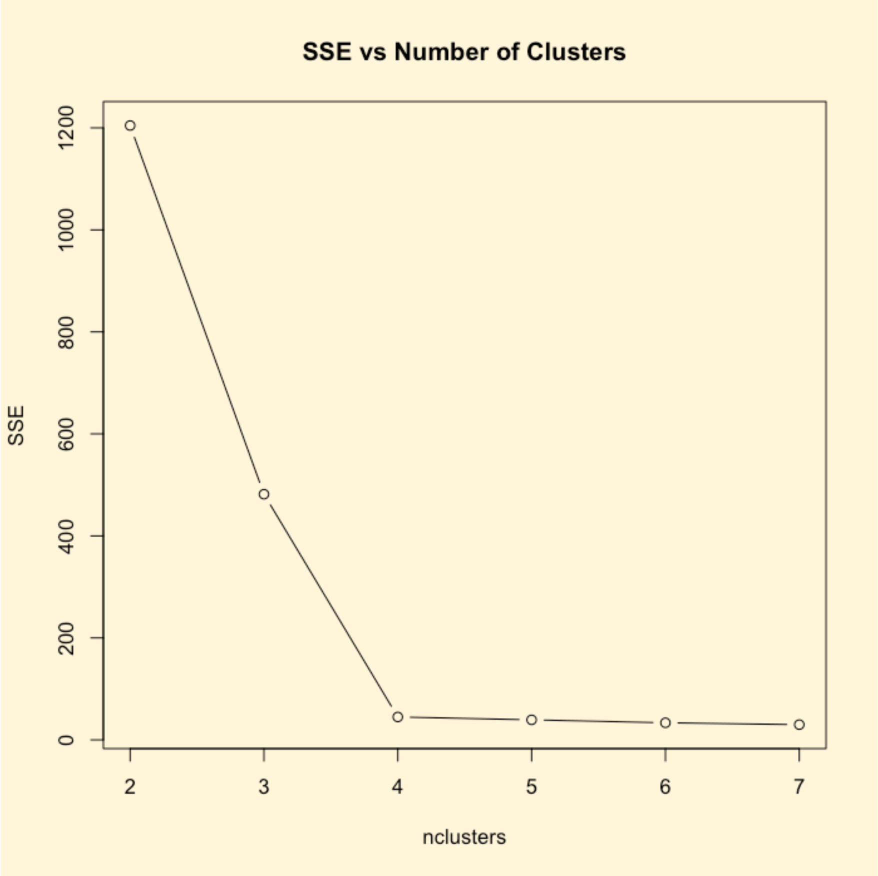

Figure 6

As you can see, the objective function decreases rapidly up to four clusters, then much more slowly as the number of clusters is increased further. (Theoretically, the SSE will continue to decline until the number of clusters reaches the number of data points.) The abrupt change in slope at four clusters indicates that this is a viable solution for this data set.

In practical application, however, there are still difficulties. The k-means algorithm only works if the number of clusters is two or greater. Thus plotting the SSE values allows us to see a sharp change in slope only if the actual number of clusters is three or greater.

To illustrate, figure 7 shows a data set with two clusters, while figure 8 shows a data set with no distinct clusters. Figure 9 and 10 show the plots of SSE values for the data sets of figures 7 and 8 respectively.

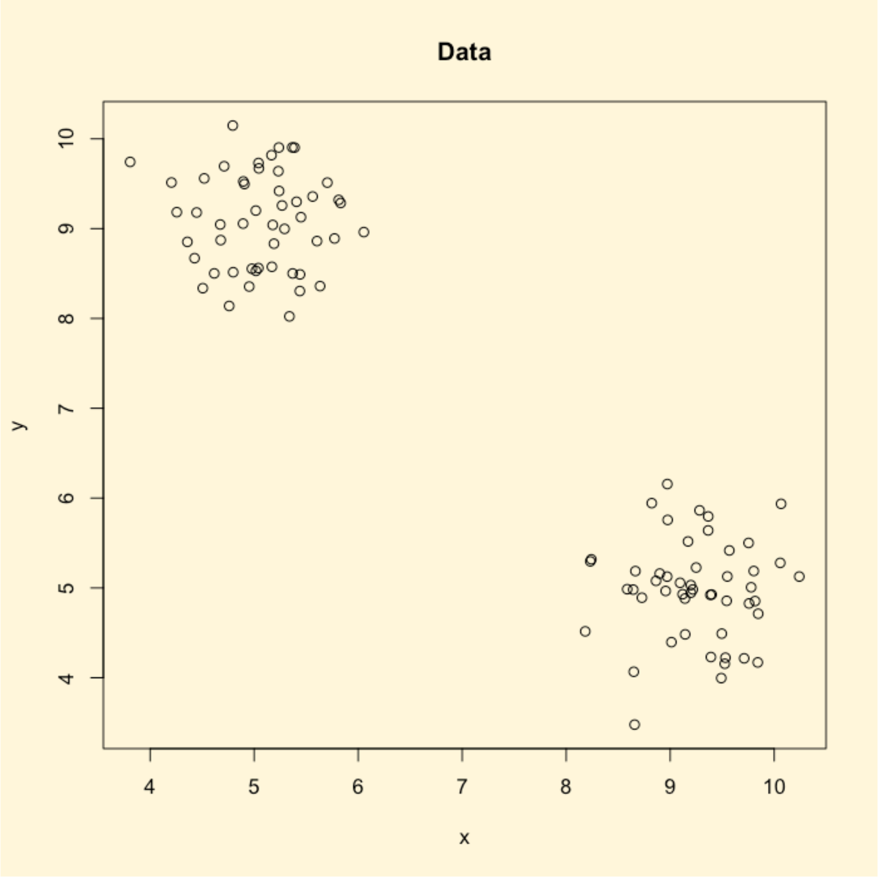

Figure 7

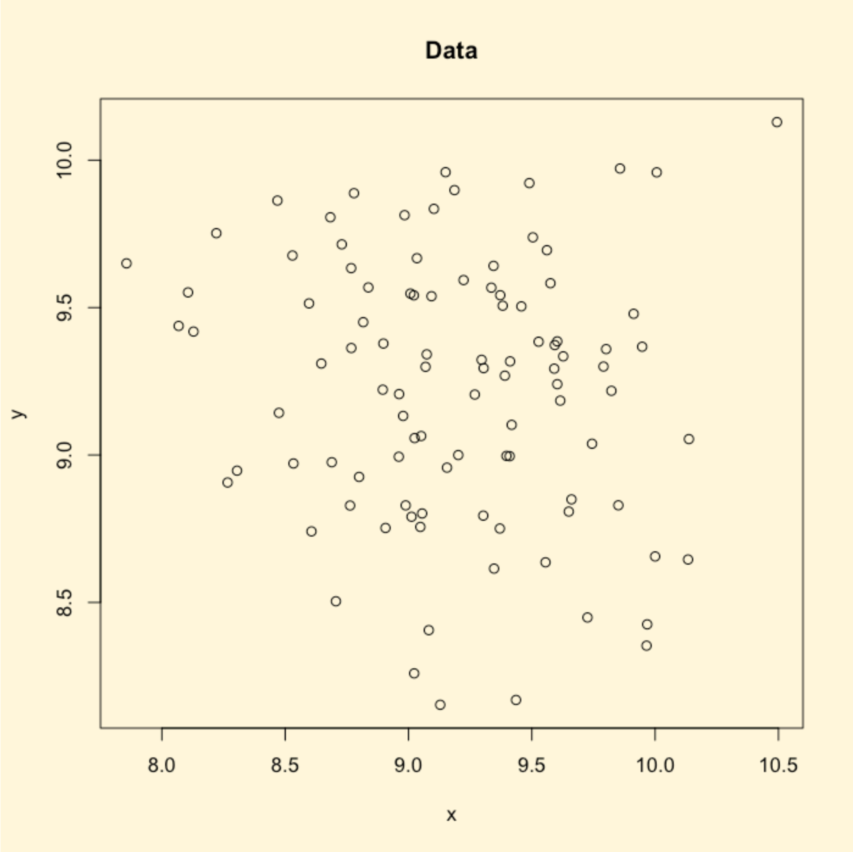

Figure 8

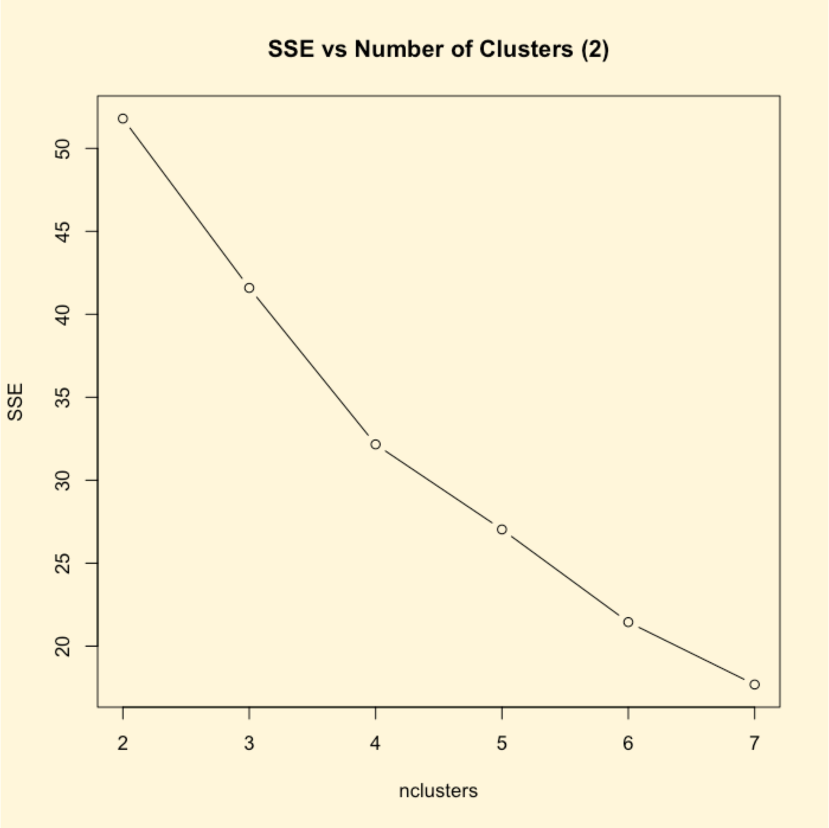
SSE vs Number of Clusters (2)
SSE
50
45
40
35
30
25
20
2
3
4
5
6
7
nclusters

Figure 9

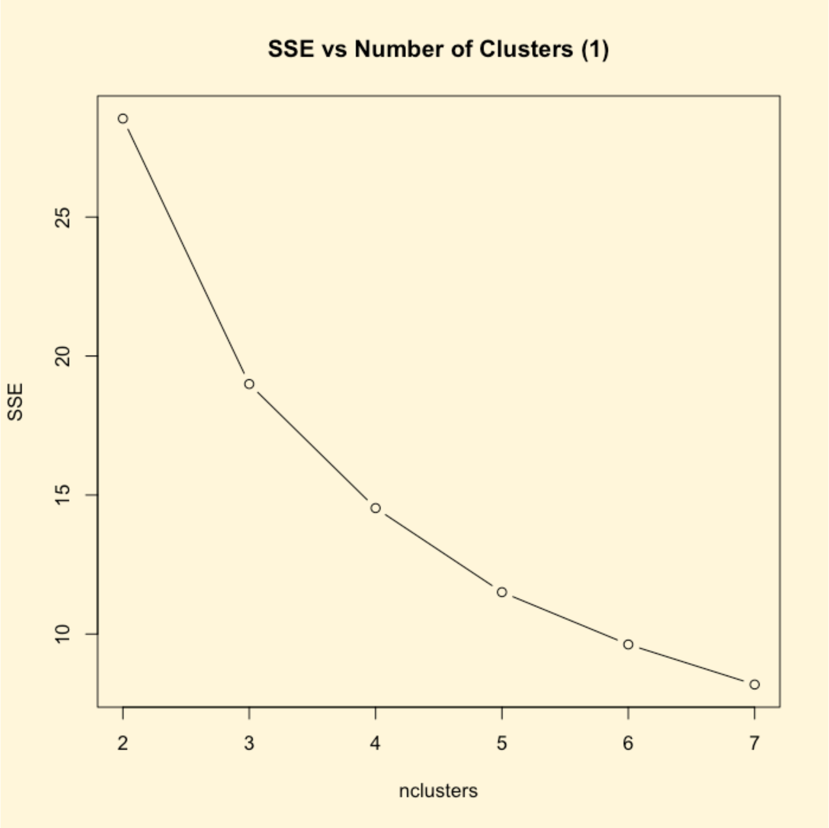

Figure 10

It is not clear from the SSE plot that there are identifiable clusters in one data set and not in the other.

The solution to this difficulty is to include in the SSE plot the SSE for the data set as a whole.

It can be shown that for a data set with individual observations in rows and the value of each dimension in columns, the SSE value is the sum of the variances of each of the columns times one less than the number of rows. That value can be calculated relatively easily:

1. Take the variance of each column
2. Sum the variances
3. Count the rows and subtract one
4. Multiply by the sum of the variances

Figures 11 and 12 show the resulting modified SSE plots for the data sets of figures 7 and 8. In figure 11, the characteristic abrupt change in slope at two indicates there are two clusters in the data. In figure 12, there is no such abrupt change, indicating there are no distinct clusters in the data, at least up through the largest number of clusters tested.

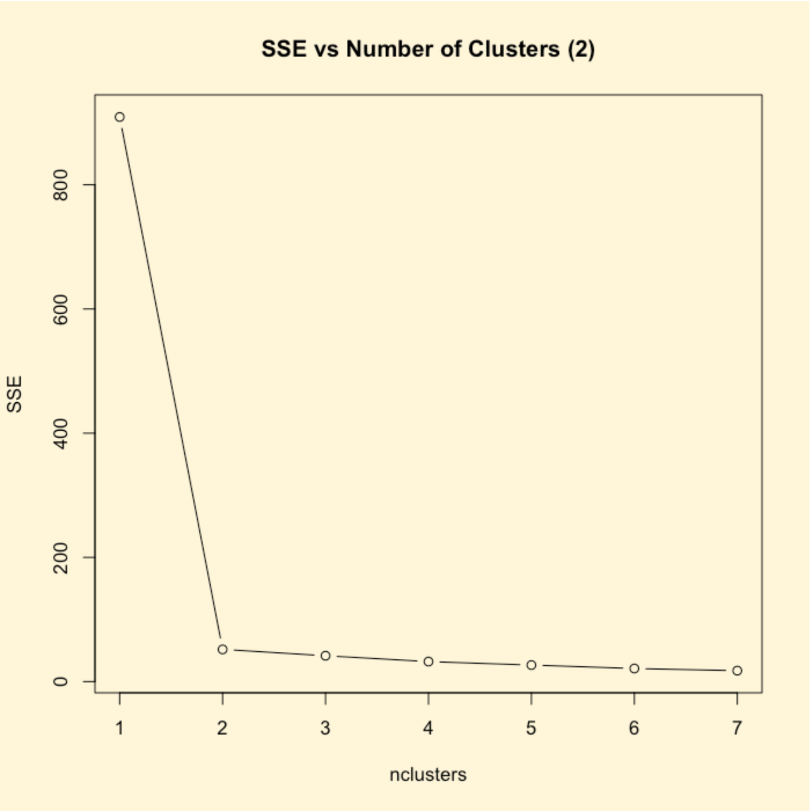

Figure 11

In figure 11 we can clearly see a sudden change in slope at two clusters, while in figure 12 there is no clearly defined slope change.

The difference between figure 11 and 12 is stark. Clearly there are two clusters in the data set represented by figure 11.

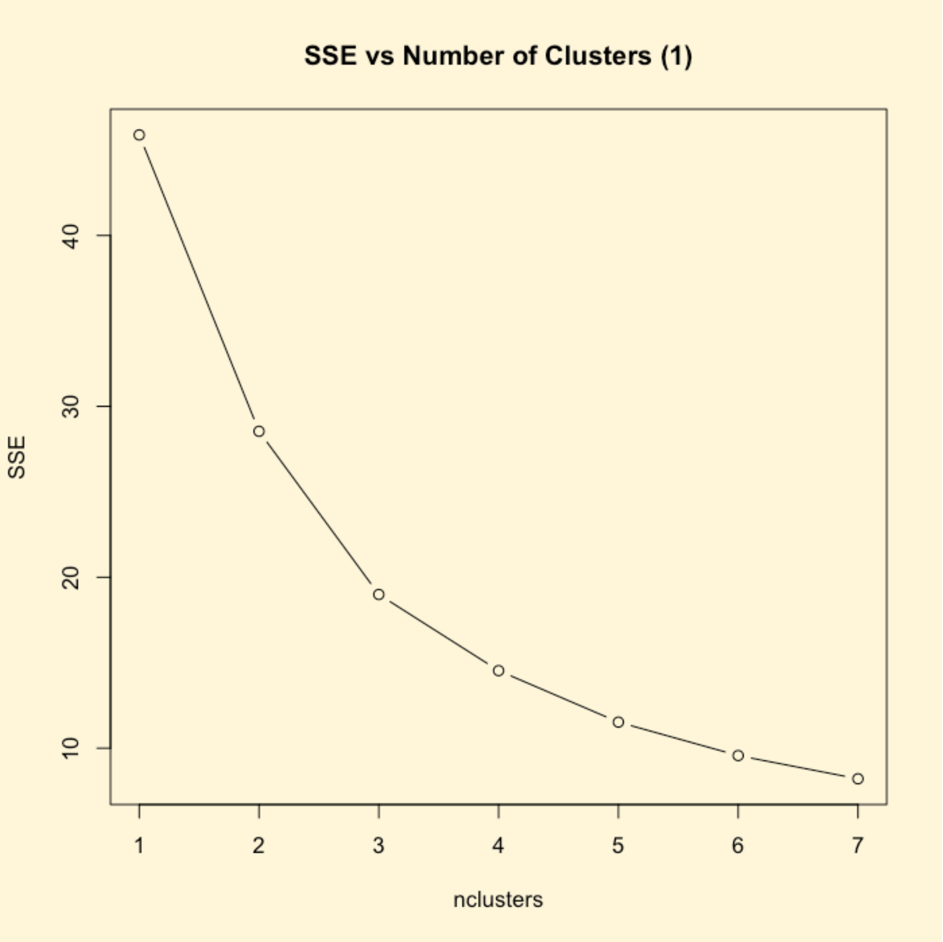

Figure 12

It should be noted that the effectiveness of this technique is dependent upon the degree of separation of the clusters in the data. If the clusters are distinct and well separated, the technique is very effective. As the data gets "messier", the technique produces results that are less clear-cut. As always, judgment and interpretation is necessary to do useful work.

The following example was constructed to simulate a survey of 600 car buyers. Each response was created to model a survey respondent in one of three categories (Economy, Luxury, and Performance) and rated each of the following characteristics on a scale of 1 to 5.

1. fuel economy
2. low price
3. low maintenance costs
4. reliability
5. comfortable ride
6. luxury appointments
7. spacious cabin
8. cornering ability
9. acceleration
10. braking performance

The data set was generated using a function that can generate random samples with specified probabilities. For example, the survey responses for "low price" were generated with the following probabilities by respondent type:

Value	Economy Buyer	Luxury Buyer	Performance Buyer
5	.60	0	.10
4	.30	0	.15
3	.10	.10	.25
2	0	.40	.40
1	0	.50	.10

Each of the other columns was populated in a similar fashion using probabilities considered appropriate for the three buyer types.

The k-means algorithm was applied and the SSE values are plotted in Figure 13.

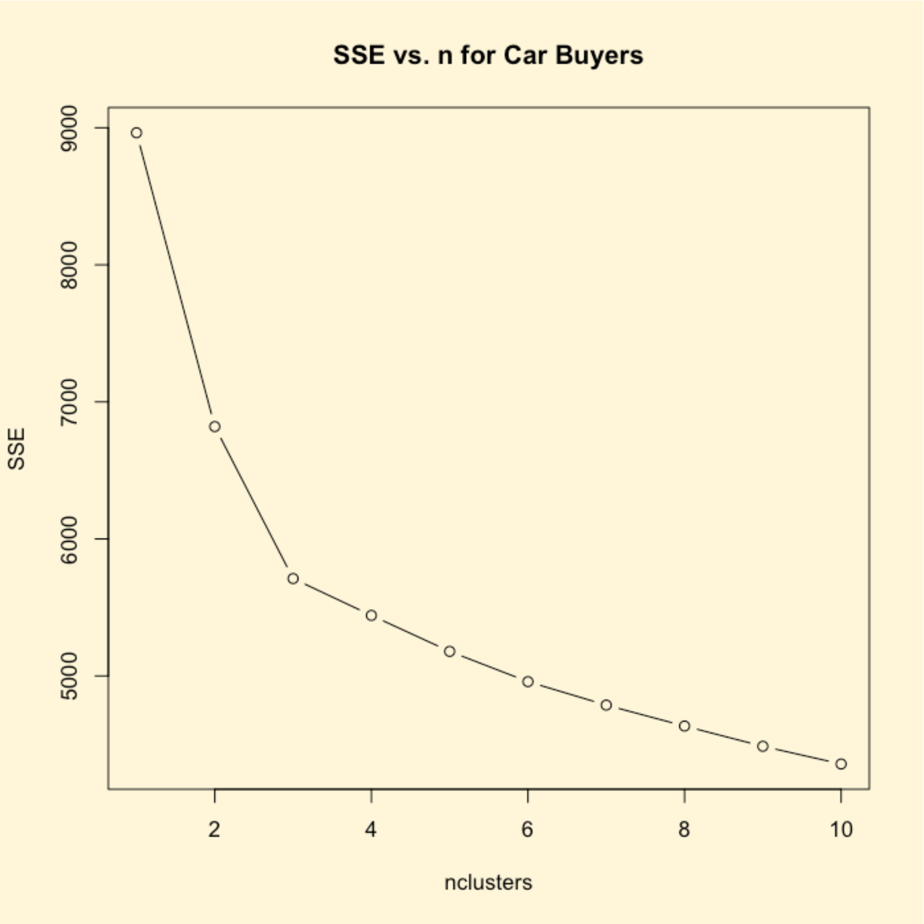

Figure 13

Although it is not dramatic, there is a significant change in slope at a value of 3 clusters.

The output of the k-means algorithm for three clusters is:

```
K-means clustering with 3 clusters of sizes 204, 197, 199

Cluster means:
        c1       c2       c3       c4       c5       c6       c7
1 2.210784 1.637255 1.715686 3.676471 4.465686 3.901961 3.406863
2 2.624365 1.989848 3.137056 3.736041 2.944162 2.472081 2.888325
3 4.120603 4.618090 3.557789 3.658291 2.899497 2.472362 2.788945
        c8       c9      c10
1 3.338235 3.205882 3.333333
2 4.558376 4.461929 4.375635
3 3.000000 2.959799 2.829146
```

The cluster identified as cluster 1 has the highest weights associated with columns 5, 6, 4, and 7 in that order. Cluster 2 places the highest weights on columns 8, 9, 10, and 4. Cluster 3 favors columns 2, 1, 4, and 3. It is not difficult to identify cluster 1 as luxury buyers, cluster 2 as performance buyers, and cluster 3 as economy buyers.

Had we ben conflicted about whether there were actually three or four clusters, we could have examined the k-means output for four clusters.

```
K-means clustering with 4 clusters of sizes 122, 81, 203, 194

Cluster means:
        c1       c2       c3       c4       c5       c6       c7
1 4.147541 4.663934 3.500000 4.581967 2.918033 2.442623 2.786885
2 4.061728 4.469136 3.580247 2.148148 2.888889 2.481481 2.777778
3 2.206897 1.630542 1.714286 3.689655 4.467980 3.906404 3.408867
4 2.603093 1.974227 3.149485 3.773196 2.943299 2.489691 2.896907
        c8       c9      c10
1 2.991803 2.934426 2.868852
2 3.074074 3.037037 2.802469
3 3.339901 3.201970 3.334975
4 4.556701 4.474227 4.386598
```

This reveals that one cluster favors columns 2, 4, 1, and 3. This is the same as cluster 3 (economy) above. Another cluster favors 2, 1, 3, and 8. This cluster shares three of the top four columns with the economy cluster above. Another favors 5, 6, 4, and 7. This is the same as cluster 1 (luxury) above. The last cluster favors 8, 9, 10, and 4. This is the same as cluster 2 (performance) above. Apparently this arrangement subdivides the economy buyers into two clusters based on some of their less important needs.

So making an incorrect choice of number of clusters is not necessarily a disaster. Selecting one cluster too few will "split the difference" between two similar clusters, while one two many will subdivide a cluster into two.

A useful way of viewing the clusters is to plot them all on the same line graph, showing the values or weights assigned to each characteristic by each cluster. Figure 14 shows such a plot for the three clusters of our example.

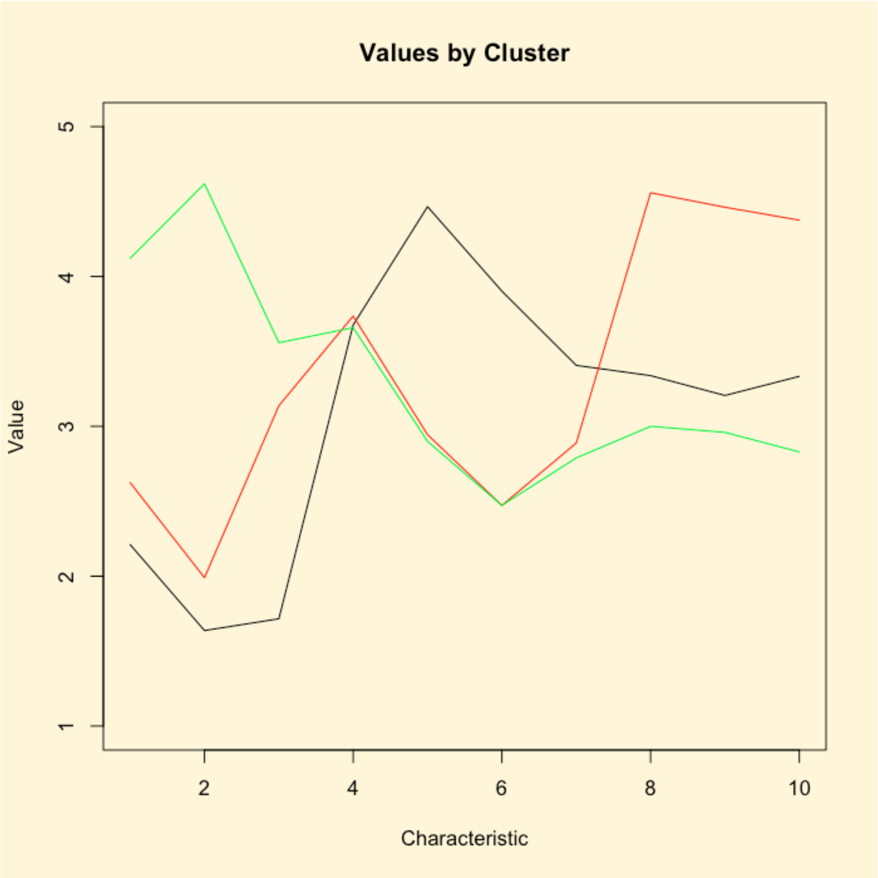

Figure 14

This plot shows clearly where the clusters differ most and where they are most similar.