

Let us now look at implementing the k-means algorithm.

Implementation of K-Means Clustering in R II

- `kmeans()` function is already implemented in R-base (i.e. no additional package is needed)
- The function takes a `data.frame`/matrix as well as the number of clusters, k , and then returns an object of class "kmeans" which has a `print` and a `fitted` method.
- One can alternatively input the initial centroids. Moreover, the maximum number of algorithm iterations can be defined by the user.
- Finally, the user can define a number of start sets and choose the best clustering outcome. Note that k-means is a stochastic algorithm.

As is now common, R provides several options for implementing k-means. But, the basic `kmeans` package is available in the R-base package, so no further libraries are needed to use that, though we will install further packages to get some added output.

There are several user-defined options. These include specifying k , the initial centroid values, the number of start sets, which specifies how many times an initial solution is generated, and the best clustering outcome.

Implementation of K-Means Clustering in R

- In addition to base R, there are a number of packages that can provide additional functionalities.
- One package is “factoextra” which contains a number of useful functions:
 - **fviz_cluster()**: to show the results of clustering
 - **fviz_dist()**: to show the distance between the observations, and
 - **fviz_nbclust()**: Determines and visualize the optimal number of clusters using different methods

Code for this module is available on our github account at <https://github.com/KSU-MSBA/64060.git>

We will extend the use of the base package through some additional packages. These will help us understand the results better. Install the “factoextra” package. Note that code for this module is available in your github account.

Example: Clustering of Cars I

```
library(tidyverse) # data manipulation
library(factoextra) # clustering algorithms & visualization
library(ISLR)
set.seed(123)
df<-Auto[,c(1,6)]
summary(df)
```

```
##      mpg      acceleration
##  Min.   : 9.00   Min.      : 8.00
##  1st Qu.:17.00   1st Qu.:13.78
##  Median :22.75   Median :15.50
##  Mean   :23.45   Mean    :15.54
##  3rd Qu.:29.00   3rd Qu.:17.02
##  Max.   :46.60   Max.    :24.80
```

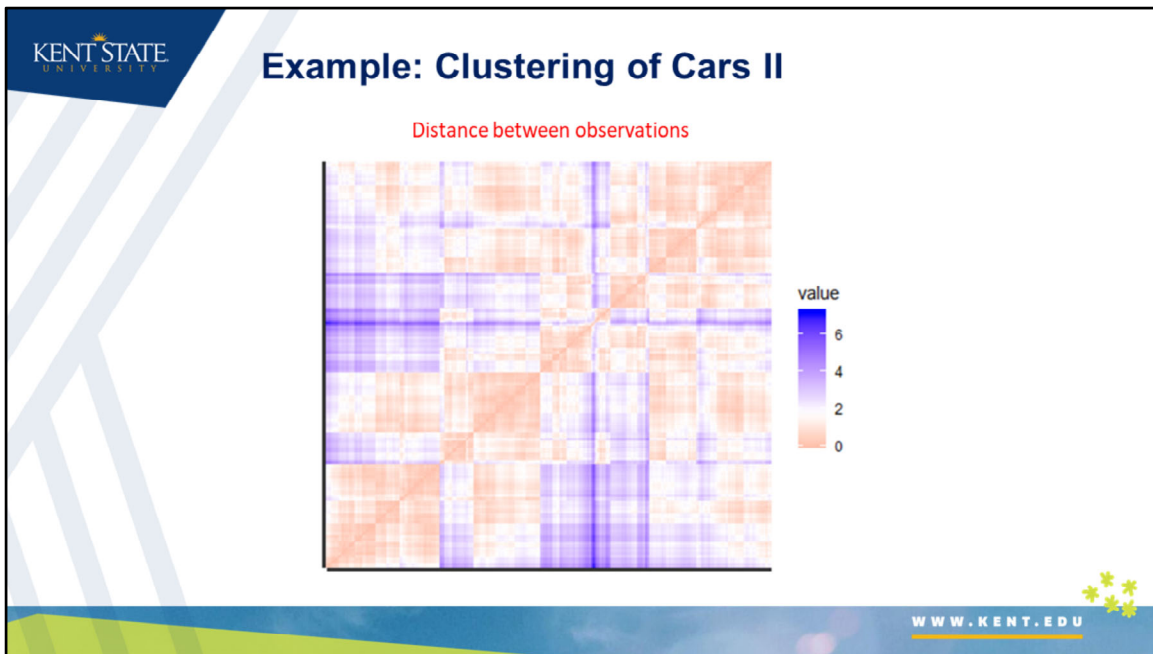
```
# Scaling the data frame (z-score)
```

```
df <- scale(df)
distance <- get_dist(df)
fviz_dist(distance)
```

Clustering Cars based on their fuel efficiency
(Mile Per Gallon) and Speed (Acceleration)

We could alternatively scale using range with
the caret package

We will cluster the cars based on mpg and acceleration. By default, we use the Euclidean distance. As the distance measure is sensitive to scale, we normalize it.



Here is a visual representation of the normalized distance between cars.

Example: Clustering of Cars III

```
k4 <- kmeans(df, centers = 4, nstart = 25)
k4$centers
```

```
##      mpg acceleration
## 1 -0.4037992    0.4422649
## 2  1.0724455    1.5014548
## 3 -1.0417446   -1.0794729
## 4  0.9238892   -0.2028643
```

Centroids

```
k4$size
```

```
## [1] 122  55 106 109
```

Size of each cluster

```
k4$cluster[120]
```

Find the cluster of 120th observation in the dataset

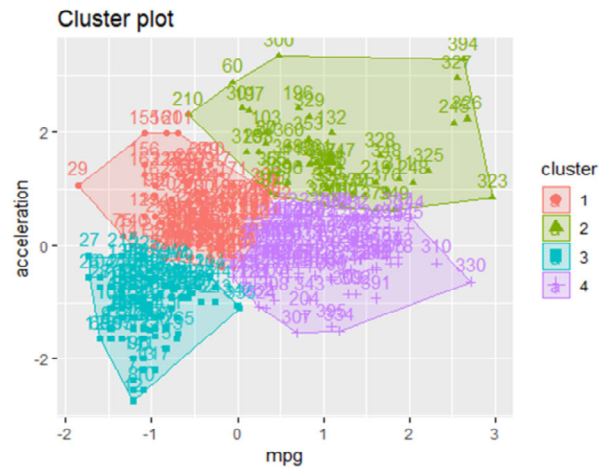
```
## 121
##    1
```

```
fviz_cluster(k4, data = df)
```

Visualize the clusters

We use the kmeans package with $k = 4$, and 25 restarts. The solution indicates 4 clusters with centroid values as specified in the output. Note again that the values have been normalized. We can now visualize the final clusters.

Example: Clustering of Cars IV




It is now easy to see that Cluster 1 represents cars with lower than average mpg (negative values), but higher than average acceleration (positive values). Similarly, Cluster 2 represents higher than average mpg and acceleration (positive values for the centroids).

'flexclust' Package


- 'flexclust' is another extensive package that implements a general framework for k-centroids cluster analysis supporting arbitrary distance measures and centroid computation
- The main function of the package is `kcca()` which returns objects of class "kcca".
- One can simply apply the `predict()` method to a `kcca` object to get the cluster association of new observations.
- If no additional data is passed to the `predict()` function, it will return the cluster association of the original dataset.

We now explore the use of other distances in calculating clusters. Install the "flexclust" package, which provides a general framework for k-means cluster analysis.



kcca() distance metrics

- `kccaFamily()` currently has the following pre-defined distance families:
 - `kmeans`: Euclidean distance
 - `kmedians`: Manhattan distance
 - `angle`: angle between observation and centroid
 - `jaccard`: Jaccard distance



WWW.KENT.EDU

The most common measure of distance, as we saw earlier, is the Euclidean distance.

The Manhattan distance, also known as the “city block” distance, looks at the absolute differences rather than squared differences like in the Euclidean distance calculation.

The Jaccard distance is a measure that is used primarily for categorical data.

Example: Clustering with Manhattan distance I

```
set.seed(123)
#kmeans clustering, using manhattan distance
k4 = kcca(df, k=4, kccaFamily("kmedians"))
k4

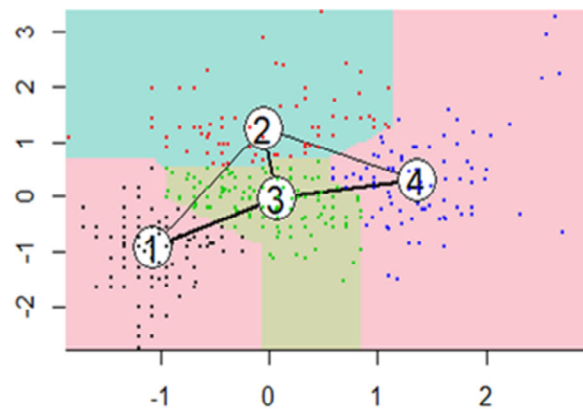
## kcca object of family 'kmedians'
## call:
## kcca(x = df, k = 4, family = kccaFamily("kmedians"))
##
## cluster sizes:
##      1      2      3      4
## 108    72   127    85

#Apply the predict() function
clusters_index <- predict(k4)

image(k4)
points(df, col=clusters_index, pch=19, cex=0.3)
```

Here we apply the Manhattan distance to our calculations.

Example: Clustering with Manhattan distance II



The output shows the four centroids for our clusters. Note that the clusters are different from when we used the Euclidean distances.

This concludes our discussion for this module. In the next module, we look at some practical considerations in applying k-means.