Analytics of Business Intelligence

Chapter # 5 - Data Mining with Cluster Analysis

Dr. Wajahat Gilani

Rutgers Business School

April 7, 2021

Introduction

Data mining is the process of working with large amounts of data to gather insights and find patterns. The data does not need to include a responsive variable, the belief is that a relationship or the information about the structure of the relationship(s) lie within the data. This section will cover 3 introductory topics on data mining:

- Explaining cluster analysis
- Partitioning using k-means clustering
- Clustering using hierarchical techniques

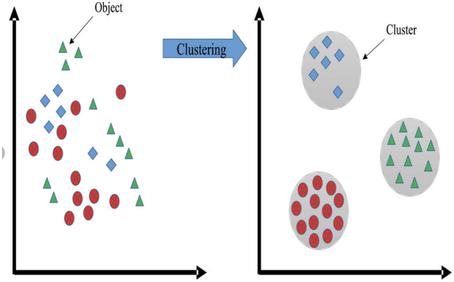
We will be using the text file Ch5_bike_station_locations.csv.

Explaining Cluster Analysis

Clustering

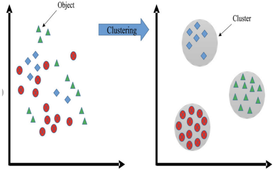
Clustering is the process of grouping a set of data objects into multiple groups or clusters so that objects within a cluster have high similarity, but are very dissimilar to objects in other clusters.

Imagine a dinner party in a rectangular room. All the guests don't congregate around the center to collectively socialize, eventually people break into different groups around the room based on various reasons.



Explaining Cluster Analysis

Clusters are collections of points from a multidimensional set of data such that they minimize the distance between each cluster central point and its members. In this section, we will focus on the error between the data points and their central point within each group for cluster. When we were working with linear regression, you will recall that the regression had a response variable. That is referred to **supervised learning**, i.e. when there is a response variable. Cluster analysis is **unsupervised learning**, because it does not have a response variable. In unsupervised learning other ways are used to explain relationships in the data. In the dinner party example, the method would be to find a central point within each identifiable cluster. We explore two types of cluster analysis methods: partioning and clustering.



Partitioning using k-means Clustering

The objective of partitioning, requires setting points within the data and minimizing the distance (or error) from each data to one of those specific points. The partitioning method **K− means** places centers at K locations inside the observations space. That is where the K in $K\!-\!means$ comes from. For example, if you were performing $K\!-\!means$ clustering with K = 3, you would be creating three clusters in the data space.

K – means iteratively steps through 3 steps:

1. Specified the number of clusters, K. Assign their initial locations randomly or in specific locations.
2. The algorithm assigns all observations in the data set to the nearest cluster.
3. The location of each cluster center is recalculated by calculating the mean of all members of the cluster across all dimensions.

Steps 2 and 3 repeat (reassigning points to clusters and then reposition cluster centers) until there is no further movement of the clusters.

Partitioning using k-means Clustering

This brings up an interesting problem problem, how do you know **how many clusters** to choose? Usually the business case drives that or the data itself will help make that apparent.

Customer Service Kiosk Placement Today's lab involves choosing the location for 3 kiosks strategically placed among the bike sharing stations. The idea is that people will work in these kiosks throughout the day and will help rider with routes and sell small products and services to the riders, with the possibility of converting them to annual members.

Our job as data scientists is to use the locations of the bike sharing stations provided in the data ands recommend where to put the 3 kiosks. We want to make sure that no bike sharing station is too far from at least 1 kiosk.

This is a prime example of the project itself dictating what K should be in our clustering analysis.

Exploring the Data

```r
library(data.table)
bike = copy(Ch5_bike_station_locations)
setDT(bike)
str(bike)
```

```
> str(bike)
Classes ‘data.table’ and 'data.frame':	244 obs. of  2 variables:
 $ latitude : num  39 38.9 39 38.9 38.9 ...
 $ longitude: num  -77 -77 -77.1 -76.9 -77.1 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

The data just has 2 columns, latitude and longitude. If you recall longitude measures on a map the positions from east to west and latitude measures the positions from north to south. This data is supposed to be for Washington D.C., which is why longitude is negative (west of prime meridian) and the latitude is positive (north of equator).

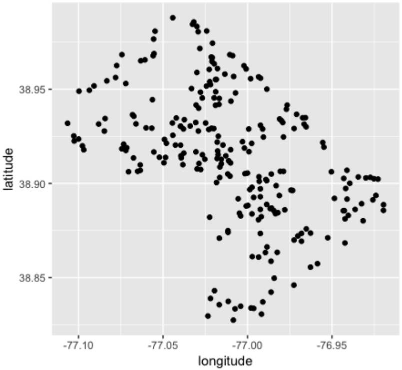
There are no categorical columns but lets check for nulls.

Exploring the Data

```
grep('NA',bike)
```

Recall that the grep() function will give us the column numbers of the table bike that have NA values. Clearly there are no NA's.

```
ggplot(bike,aes(x=longitude,y=latitude)) + geom_point
    ()
```



We can see a general outline of Washington D.C.

Running kmeans() Function

The kmeans() function will generate points that are centrally placed among the data. It'll give the mean positions for each of the k points, and provide vector with the number stating which the point belongs to which cluster group. The kmeans() function starts with randomly placing the center k points, and then starts the interative approach of calculating the distance of each point from the closest of the **k points**, and then moving the points until there is no more room for reducing the average distance. Since it starts with randomly placing the points, we can use set.seed() to get the same answer:

```
set.seed(123)
k3=kmeans(bike,3)
k3
```

Interpreting the Model Output

```
> k3
K-means clustering with 3 clusters of sizes 78, 40, 126

Cluster means:
  latitude longitude
1 38.90412 -76.96869
2 38.85987 -76.99616
3 38.93786 -77.04018

Clustering vector:
  [1] 3 1 3 1 3 3 1 2 1 1 3 1 1 1 3 3 3 3 3 1 2 3 3 1 3 1 3 3 3 2 1 1 1 3 1 3 3 1 3 3 2 3 3 3 1 3 1 3 1 3 2 3 2 2 1 1 1 2 3 1
 [61] 3 3 3 3 2 2 2 3 1 3 1 1 3 3 1 1 1 3 1 3 1 3 3 3 1 3 2 3 2 2 3 1 3 2 2 3 3 2 3 3 3 2 3 3 1 3 3 3 3 3 3 3 1 3 3 2 3 3 2 1
[121] 1 1 3 3 3 1 1 1 2 2 3 3 3 3 3 2 1 1 3 1 3 3 3 3 3 3 3 1 3 2 3 3 1 3 3 1 3 1 1 1 1 1 2 2 3 1 3 3 3 2 3 3 3 1 3 3 1 3 3 3
[181] 2 1 1 3 1 2 2 3 2 3 1 2 3 3 2 1 1 3 1 3 3 3 3 3 3 1 1 3 3 3 3 1 2 3 3 1 3 1 2 3 1 3 1 2 3 1 1 3 2 1 3 1 3 3 1 2 1 3 1 3
[241] 2 1 2 1

Within cluster sum of squares by cluster:
[1] 0.07973031 0.02673172 0.15673804
 (between_SS / total_SS =  63.2 %)

Available components:

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

- Three clusters, sizes 78, 40, 126
- The means of the center points
- Shows the cluster assignment of each data point
- The sum of square error in each cluster, as well as the percentage showing how well the model accounted for error
- All the components you can tyaccess for computation

Explaining The Error

The 63.2% error describes how well the model fits. The error is the ratio of the **AVERAGE sum of squares** between each **cluster** and the mean, divided by **sum of squares** for all the data.

$$\text{Error Rate} = \frac{\text{Between Cluster Sum of Squares}}{\text{All Sum of Squares}}$$

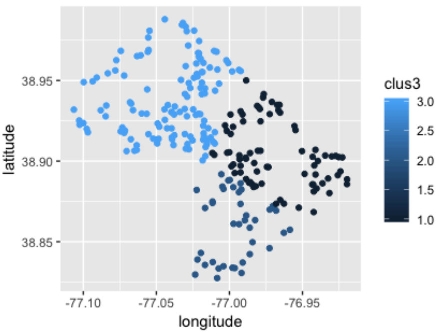
$$\text{Sum of Squares} = \sum_{i=1}^{Points} (x_i - mean_x)^2 + (y_i - mean_y)^2$$

In cluster analysis, the idea is to have a lot of similarity within each cluster (a small sum of squares) and have very different characteristics between the clusters. This is the reason we want to maximize the error of the numerator. If each individual point was its own cluster, then the **error rate** would be 100%.

Visualize The Clusters

Lets visualize the different clusters in a visual plot. First, add a column to our bike table, that has the cluster assignments for each station, and then visualize it, with each cluster being a different color:

```
bike[,clus3:=k3$cluster]
ggplot(bike,aes(x=longitude,y=latitude,color=clus3)) +
    geom_point()
```

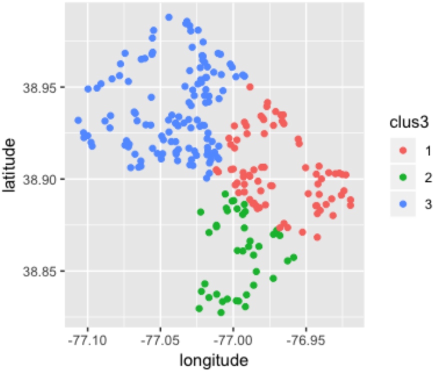


This is not the result we wanted, what happened?

Visualize The Clusters

Is the column, clus3, a numerical column or a categorical column? Is it nominal or ordinal? This is the result of not properly identifying the data:

```
bike[,clus3:=factor(clus3)]
ggplot(bike,aes(x=longitude,y=latitude,color=clus3)) +
    geom_point()
```



Now we get a clear distinction of the clusters.

Visualize The Stations

The original business questions is where do we place the kiosks. We have the locations:

```
k3$centers
```

But its being given to use in a data container type that we haven't discussed, called a matrix:

```
class(k3$centers)
```

A a matrix is just a multi-dimensional vector. Just like a table, instead of the just 1 position, you have 2 positions separated by a comma, for example:

```
k3$centers[1,1]
```

Is the value in the first row and first column. We really don't have to deal with matrices because we are using data.tables.

Visualize The Stations

Convert the matrix into a data.table:

```
centdt=data.table(k3$centers)
centdt
```

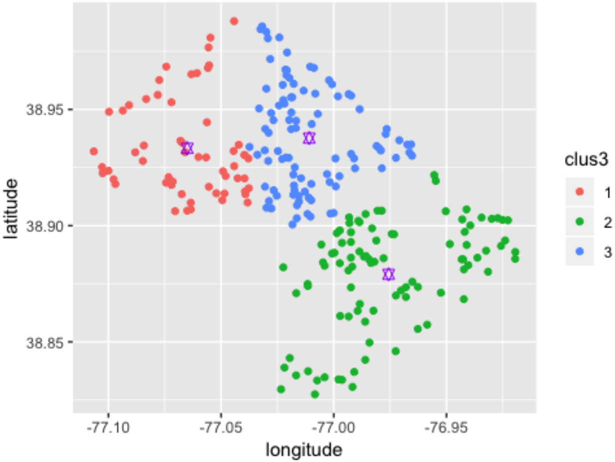
```
> centdt
   latitude longitude
1: 38.90412 -76.96869
2: 38.85987 -76.99616
3: 38.93786 -77.04018
```

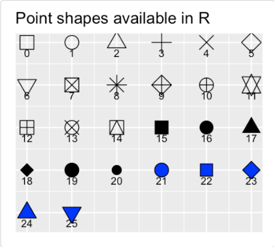
Now add the points on our graph, as additional points:

```
ggplot(bike,aes(x=longitude,y=latitude,color=clus3)) +
    geom_point() + geom_point(data=centdt,aes(x=
   longitude, y=latitude), colour="purple", shape=11,
   size=2)
```

Visualize The Stations

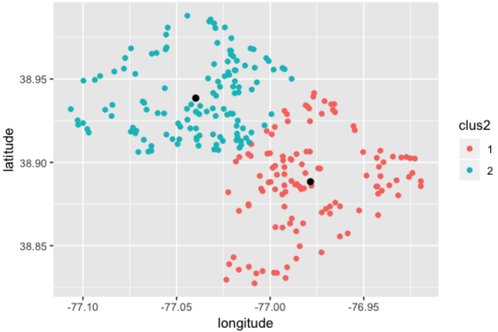
```
ggplot(bike,aes(x=longitude,y=latitude,color=clus3)) +
    geom_point() + geom_point(data=centdt,aes(x=
   longitude, y=latitude), colour="purple", shape=11,
   size=2)
```

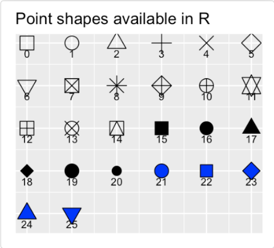




What-IF We Decide On 2 Stations

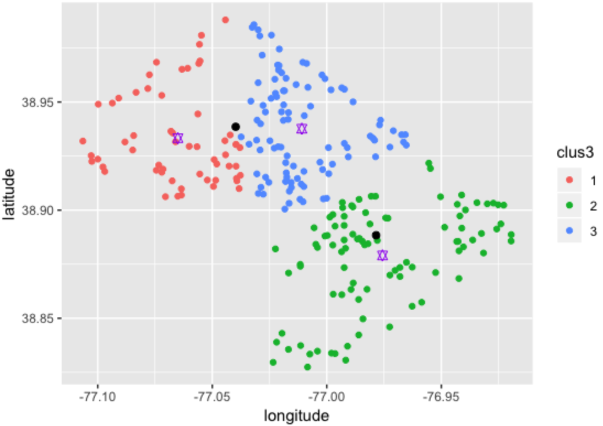
```
k2 = kmeans(bike[,.(latitude,longitude)],2)
bike[,clus2:=k2$cluster]
bike[,clus2:=factor(clus2)]
centdt2=data.table(k2$centers)
ggplot(bike,aes(x=longitude,y=latitude,color=clus2)) +
    geom_point() +
geom_point(data=centdt2,aes(x=longitude, y=latitude),
   colour="black", shape=19,size=2)5
```





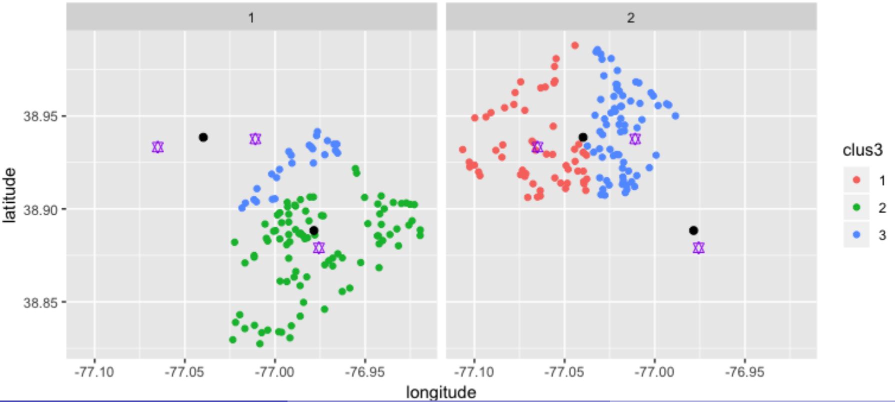
What-IF We Decide On 2 Stations

```
ggplot(bike,aes(x=longitude,y=latitude,color=clus3)) +
    geom_point()+geom_point(data=centdt,aes(x=
   longitude, y=latitude), colour="purple", shape=11,
   size=2)+geom_point(data=centdt2,aes(x=longitude, y=
   latitude), colour="black", shape=19,size=2)
```



Visualizing the 2 vs 3 Kiosk Locations

```
ggplot(bike,aes(x=longitude,y=latitude,color=clus3)) +
    geom_point() + geom_point(data=centdt,aes(x=
   longitude, y=latitude), colour="purple", shape=11,
   size=2)+geom_point(data=centdt2,aes(x=longitude, y=
   latitude), colour="black", shape=19,size=2) +facet_
   wrap(~clus2)
```



Business Case - 3 Scenarios

We want to analyze **3 scenarios**, the 3 kiosk plan, the 2 kiosk plan, and a 2-step process where 2 of the 3 kiosks are built first and then the 3rd kiosk is built. This leads to 2 questions:

- Which 2 kiosks should be built first?
- What is the average added distance for each bike station?

The answer to both questions rely on converting the longitude and latitude coordinate to actual distances. To this we are going to use the package, **geosphere**. Install the package **geosphere** using the packages tab at the bottom right window of RStudio, and then call the package into memory using the library command. We first do a simple example:

```
library(geosphere)
dd=distm(c(40.777250, -73.872610), c(40.6895,
    -74.1745), fun = distHaversine)
dd
class(dd)
```

Business Case - 3 Scenarios

```r
library(geosphere)
dd=distm(c(40.777250, -73.872610), c(40.6895,
    -74.1745), fun = distHaversine)
dd
class(dd)
```

```
> dd
         [,1]
[1,] 33713.61
> class(dd)
[1] "matrix"
>
```

There are 3 parameters, the first 2 being the locations we are measuring, the 3rd is a mathematical methodology we are using to find the shortest distance between 2 points on a spherical earth, ignoring ellipsoidal effects. This methodology was formally published by R.W. Sinnott in 1984, but was being used longer than that. Notice that the results come back as a matrix, which is hinting that we can get back a multi-dimensional result back.

There is a slight problem with the **33713.61** result, its coming back as kilometers!!! (This is murica!!!)

Business Case - 3 Scenarios (3 Location)

To address the kilometers debacle, we will apply a simple mathematical trick:

```
library(geosphere)
distm(c(40.777250, -73.872610), c(40.6895, -74.1745),
    fun = distHaversine) / 1609
```

Dividing by 1609, converts the distance to miles. Now, lets calculate the miles distance from each bike station to its prospective kiosk, in both the 2-kiosk plan and the 3-kiosk plan. For the 2-step plan, we need to calculate the distance from each bike station to each of the 3 bike locations, and choose the smallest of the distances.

```
res_matrix=distm(bike[,.(latitude,longitude)],centdt,
    fun=distHaversine)/1609
```

Business Case - 3 Scenarios (3 Location)

```
res_matrix=distm(bike[,.(latitude,longitude)],centdt,
    fun=distHaversine)/1609
head(res_matrix)
```

```
> head(res_matrix)
         [,1]     [,2]     [,3]
[1,] 4.965619 1.723989 1.243175
[2,] 4.416347 1.833187 0.822953
[3,] 1.025228 5.694133 3.105609
[4,] 9.359121 3.186997 5.628658
[5,] 1.994685 8.233870 5.726424
[6,] 1.366935 7.584293 5.111255
```

There are 3 columns, each column representing the kiosk location in the 3-kiosk plan. Each row represents the distance from the bike station in our bike table. Since we need all this data for our 2-step plan, we will save all 3 columns in our bike table.

```
bike[,c('k31','k32','k33'):=as.data.table(res_matrix)]
```

	latitude	longitude	clus3	clus2	k31	k32	k33
1	38.95659	-76.99344	3	2	4.9656192	1.7239888	1.2431755
2	38.90522	-77.00150	3	1	4.4163470	1.8331867	0.8229530
3	38.98086	-77.05472	1	2	1.0252280	5.6941327	3.1056088

Business Case - 3 Scenarios (3 Location)

	latitude	longitude	clus3	clus2	k31	k32	k33
1	38.95659	-76.99344	3	2	4.9656192	1.7239888	1.2431755
2	38.90522	-77.00150	3	1	4.4163470	1.8331867	0.8229530
3	38.98086	-77.05472	1	2	1.0252280	5.6941327	3.1056088

From here we need to create a column called, **c3dist**, where the value will be gotten from either **k31**, **k32**, or **k33**, according to what the cluster value is in **clus3**. For example, for the first and second row it will be from **k33**, where as from the 3rd row it will be from **k31**.

```
bike[clus3==1,c3dist:=k31]
bike[clus3==2,c3dist:=k32]
bike[clus3==3,c3dist:=k33]
```

	latitude	longitude	clus3	clus2	k31	k32	k33	c3dist
1	38.95659	-76.99344	3	2	4.9656192	1.7239888	1.2431755	1.2431755
2	38.90522	-77.00150	3	1	4.4163470	1.8331867	0.8229530	0.8229530
3	38.98086	-77.05472	1	2	1.0252280	5.6941327	3.1056088	1.0252280

Business Case - 3 Scenarios (2 Location)

```
res_matrix2=distm(bike[,.(latitude,longitude)],centdt2
    ,fun=distHaversine)/1609
head(res_matrix2)
```

```
> head(res_matrix2)
         [,1]     [,2]
[1,] 1.484370 3.216607
[2,] 1.615410 2.696986
[3,] 5.468738 1.226030
[4,] 3.366044 7.619402
[5,] 8.023900 3.730711
[6,] 7.378553 3.116799
```

There are just 2 columns now

Business Case - 3 Scenarios (2 Location)

```
bike[,c('k21','k22'):=as.data.table(res_matrix2)]
bike[clus2==1,c2dist:=k21]
bike[clus2==2,c2dist:=k22]
```

This will create the **c2dist** column for us, but we don't need columns **k21** and **k22**, so we can delete them:

```
bike[,c('k21','k22'):=NULL]
```

Which leaves us with a **bike** table:

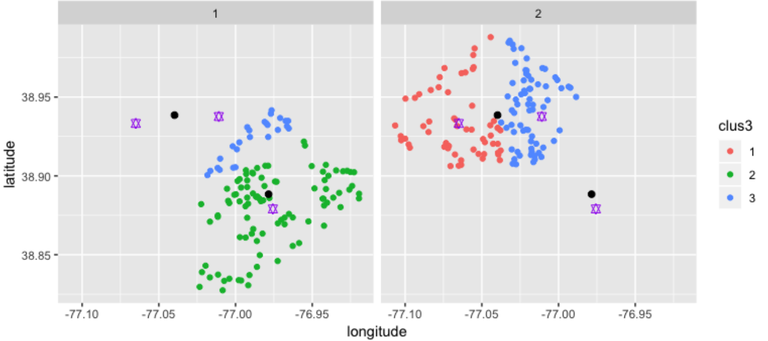
	latitude	longitude	clus3	clus2	k31	k32	k33	c3dist	c2dist
1	38.95659	-76.99344	3	2	4.9656192	1.7239888	1.2431755	1.2431755	3.2166065
2	38.90522	-77.00150	3	1	4.4163470	1.8331867	0.8229530	0.8229530	1.6154098
3	38.98086	-77.05472	1	2	1.0252280	5.6941327	3.1056088	1.0252280	1.2260298

Business Case - 3 Scenarios (2-Step)

We now need to answer the question for the 2-step process, which 2 of the stations from the **clus3** column do we develop in our initial stage?

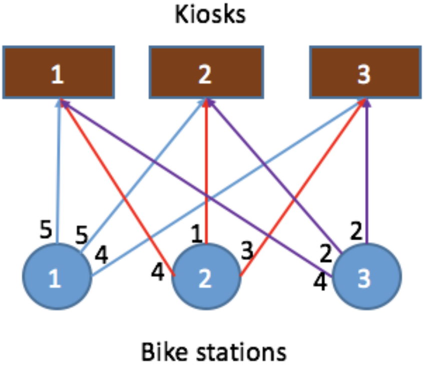
	latitude	longitude	clus3	clus2	k31	k32	k33	c3dist	c2dist
1	38.95659	-76.99344	3	2	4.9656192	1.7239888	1.2431755	1.2431755	3.2166065
2	38.90522	-77.00150	3	1	4.4163470	1.8331867	0.8229530	0.8229530	1.6154098
3	38.98086	-77.05472	1	2	1.0252280	5.6941327	3.1056088	1.0252280	1.2260298

Visually we know that kiosk 2 should be 1 of the kiosks built in the first phase, but which of the kiosks 1 or 3 should be built in the first phase?



Business Case - 3 Scenarios (2-Step)

The solution is to **not build** the kiosk that will add the least average distance to the other bike stations. Consider the example below:



Bike Station	1	2	3
Min Dist	4	1	2

The average minimum distance is **2.3**

Remove 1 kiosk at a time and re-calculate the minimum distance

Station **(no kiosk 1)**	1	2	3
Min Dist	4	1	2

average minimum distance is **2.3**

Station **(no kiosk 2)**	1	2	3
Min Dist	4	3	2

average minimum distance is **3**

Station **(no kiosk 3)**	1	2	3
Min Dist	5	1	2

average minimum distance is **2.67**

Business Case - 3 Scenarios (2-Step)

We create another table with the columns we need for the calculation:

```
StaDist=bike[,.(clus3,k31,k32,k33)]
```

We now want to find 3 scenarios:

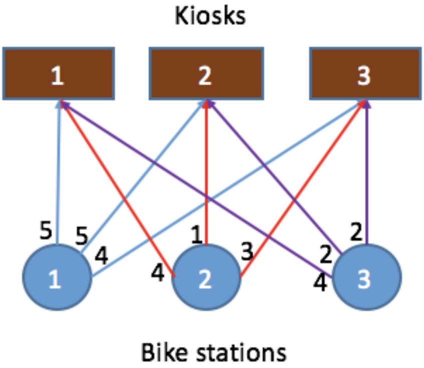
- For kiosk 1 bike stations, find the smaller distance between kiosk 2 or 3
- For kiosk 2 bike stations, find the smaller distance between kiosk 1 or 3
- For kiosk 3 bike stations, find the smaller distance between kiosk 1 or 2

We will need to apply the min function **across** columns. There are several ways to do this but there are a set functions can be applied across rows.

Business Case - 3 Scenarios (2-Step)

StaDist

clus3	k31	k32	k33
1	5	5	4
2	4	1	3
3	4	2	2



clus3	k31	k32	k33	nok1	nok2	nok3
1	5	5	4	4	4	5
2	4	1	3	1	3	1
3	4	2	2	2	2	2
AVG				2.3	3	2.67

Business Case - 3 Scenarios (2-Step)

- rowMeans()
- rowSums()
- pmin()

These functions require 2 data.table functions, .SD and .SDcols. .SD stands for **S**ubset of **D**ata. It is a way to apply functions to all the columns, for example, if i wanted to see the first 5 rows of each cluster in my table:

```
StaDist[,head(.SD,5),by = clus3]

StaDist[,rowMeans(.SD),.SDcols
    =2:4]
StaDist[,rowMeans(.SD),.SDcols=c('
    k31','k32','k33')]
```

```
> StaDist[,head(.SD,5),by = clus3]
    clus3      k31       k32        k33
 1:     2 1.723989 1.2431755  4.9656192
 2:     2 1.833187 0.8229530  4.4163470
 3:     2 2.999618 0.5447025  3.2624987
 4:     2 2.557712 0.3501244  3.9459909
 5:     2 3.189288 0.5902038  3.2393898
 6:     3 5.694133 3.1056088  1.0252280
 7:     3 8.233870 5.7264238  1.9946850
 8:     3 7.584293 5.1112550  1.3669348
 9:     3 6.861336 4.4066723  0.6912196
10:     3 4.332771 1.8868047  1.9065133
11:     1 3.186997 5.6286584  9.3591213
12:     1 3.890662 6.3711358 10.0937281
13:     1 0.527249 2.2839947  5.8167687
14:     1 3.885809 6.3736137 10.0937301
15:     1 1.576961 3.8946558  7.6318720
```

Business Case - 3 Scenarios (2-Step)

The function pmin() is easier to use:

```
StaDist[,pmin(k31,k32,k33)]
```

That will give you the minimum value from those 3 columns, for each individual row. Now, we can find the alternative minimum distances:

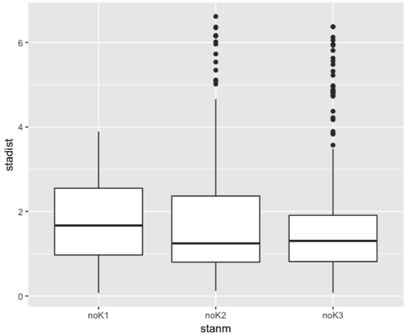
```
StaDist[,c('noK1','noK2','noK3'):=pmin(k31,k32,k33)]
StaDist[clus3==1,noK1:=pmin(k32,k33)]
StaDist[clus3==2,noK2:=pmin(k31,k33)]
StaDist[clus3==3,noK3:=pmin(k31,k32)]
StaDist[,.(mean(noK1),mean(noK2),mean(noK3))]
```

```
> StaDist[,.(mean(noK1),mean(noK2),mean(noK3))]
         V1       V2       V3
1: 1.758267 1.787666 1.711052
```

Business Case - 3 Scenarios (2-Step)

Restructure the table, to graph the distribution as boxplots.

```
stadist = c(StaDist[,noK1],StaDist[,noK2],StaDist[,noK3])
stanm = c(rep('noK1',StaDist[,.N]),rep('noK2',StaDist[,.N]),rep('noK3',StaDist[,.N]))
StaPlot = data.table(stadist,stanm)
ggplot(StaPlot,aes(x=stanm,y=stadist)) + geom_boxplot()
```



Business Case - Summarize Results

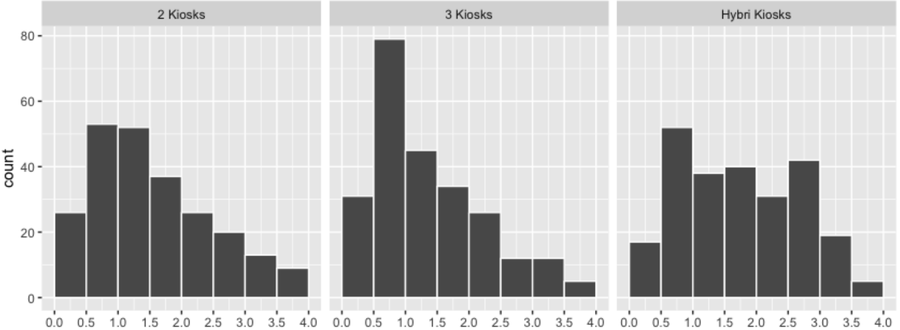
Now using the, .SD and .SDcols functions, take a summary of all 3 possible scenarios:

```
bike[,hdist:=StaDist$nok2]
bike[,summary(.SD),.SDcols=c('c3dist','c2dist','hdist'
    )]
```

```
> bike[,summary(.SD),.SDcols=c('c3dist','c2dist','hdist')]
     c3dist              c2dist              hdist
 Min.   :0.07396   Min.   :0.05196   Min.   :0.07396
 1st Qu.:0.62444   1st Qu.:0.86581   1st Qu.:0.97073
 Median :1.18240   Median :1.42258   Median :1.66877
 Mean   :1.36584   Mean   :1.63611   Mean   :1.75827
 3rd Qu.:1.84511   3rd Qu.:2.26765   3rd Qu.:2.54995
 Max.   :3.89066   Max.   :4.61923   Max.   :3.89066
```

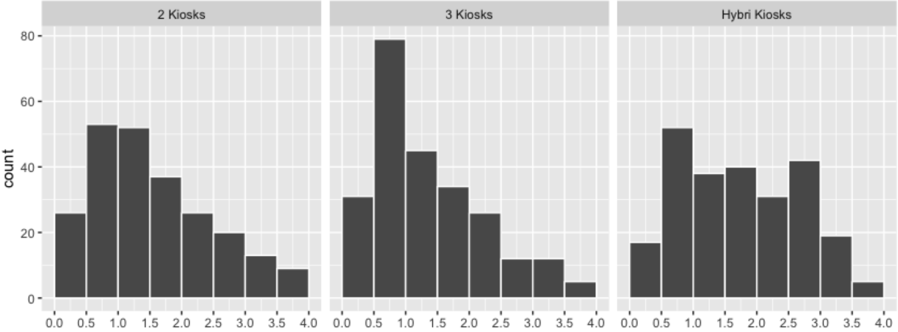
Business Case - Summarize Results

```
clusdist = c(bike[,c3dist],bike[,c2dist],bike[,hdist])
clusnm = c(rep('3 Kiosks',bike[,.N]),rep('2 Kiosks',bike[,.N]),rep('Hybri Kiosks',bike[,.N]))
clusPlot = data.table(clusdist,clusnm)
ggplot(clusPlot,aes(x=clusdist)) + geom_histogram(breaks=seq(0,4,.5),color='white')+scale_x_continuous(breaks=seq(0,4,.5)) + facet_wrap(~clusnm)
```



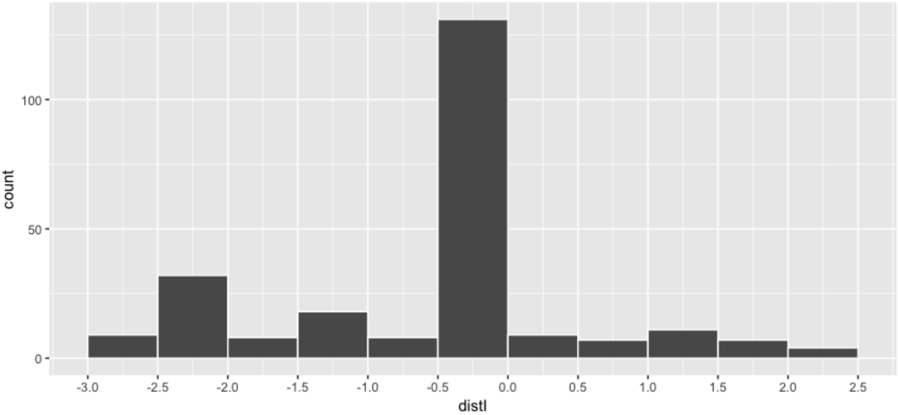
Business Case - Summarize Results

```
clusdist = c(bike[,c3dist],bike[,c2dist],bike[,hdist])
clusnm = c(rep('3 Kiosks',bike[,.N]),rep('2 Kiosks',
    bike[,.N]),rep('Hybri Kiosks',bike[,.N]))
clusPlot = data.table(clusdist,clusnm)
ggplot(clusPlot,aes(x=clusdist)) + geom_histogram(
    breaks=seq(0,4,.5),color='white')+scale_x_
    continuous(breaks=seq(0,4,.5)) + facet_wrap(~clusnm
    )
```



Business Case - Summarize Results

```
bike[,distI:=c3dist-hdist]
ggplot(bike,aes(x=distI)) + geom_histogram(breaks =
    seq(-3,2.5,.5),color='white') + scale_x_continuous(
    breaks=seq(-3,4,.5))
```



Clustering Using Hierarchical Techniques

Hierarchical techniques do not use a pre-determined set of clusters, instead they continually pair or split data into clusters based on the similarity (distance). There are 2 different approaches:

- Divisive clustering: All the data is in a single cluster and then splits it and all subsequent clusters until each data point is its own individual cluster.
- Agglomerative clustering: Each data point is its own cluster, and then they are paired together in a hierarchy until there is 1 cluster.

We will focus on using the agglomerative technique in this section, including the evaluation techniques to help us choose the number of clusters in the final model.

Targeted Marketing Segments

Using the data file Ch5_age_income_data.csv, we are given the age and income of 8,000 existing customers. We want to use that data to create customer segments.

```
setDT(marketing)
grep('NA',marketing)
str(marketing)
```

```
> str(marketing)
Classes 'data.table' and 'data.frame':  8105 obs. of  3 variables:
 $ bin   : chr  "60-69" "30-39" "20-29" "30-39" ...
 $ age   : int  64 33 24 33 78 62 88 54 54 31 ...
 $ income: num  87083 76808 12044 61972 60120 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

```
unique(marketing$bin)
```

```
> unique(marketing$bin)
[1] "60-69" "30-39" "20-29" "70-79" "80-"   "50-59" "10-19" "40-49"
```

Miller's Law?

Exploring Data

Convert the bin into categorical data, is it nominal or ordinal?

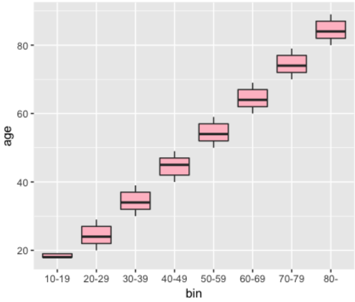
```
marketing[,bin:=factor(bin,ordered = T)]
str(marketing)
```

Why didn't we include levels?

```
> str(marketing)
Classes 'data.table' and 'data.frame':  8105 obs. of  3 variables:
 $ bin   : Ord.factor w/ 8 levels "10-19"<"20-29"<..: 6 3 2 3 7 6 8 5 5 3 ...
 $ age   : int  64 33 24 33 78 62 88 54 54 31 ...
 $ income: num  87083 76808 12044 61972 60120 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

Check to see if the data makes **logical** sense, we have the age AND a bin column that contains the age values in groups.

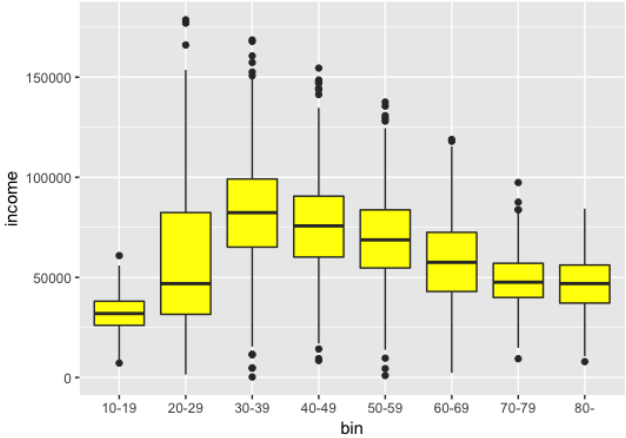
```
ggplot(marketing,aes(x=bin,y=age)) + geom_boxplot(fill='pink')
```



Exploring Data

Relationship between bin and Income?

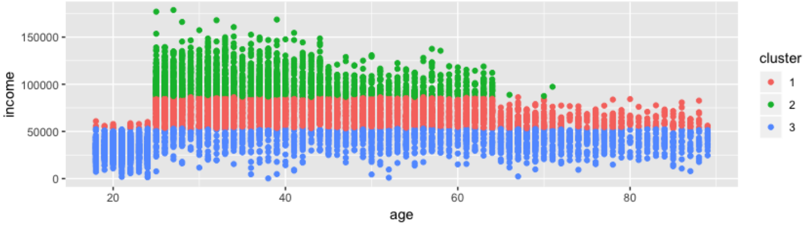
```
ggplot(marketing,aes(x=bin,y=income)) + geom_boxplot(
    fill='yellow')
```



Data Adaption

Check out the values of the columns. Do you notice something about the units? Age is tens and Income is in thousands. This matters in clustering, techniques are based on distance measures. Do a 3 cluster kmeans analysis.

```
mk3=kmeans(marketing[,.(age,income)],3)
mk3plot = data.table(age=marketing$age,income=
    marketing$income,cluster=mk3$cluster)
mk3plot[,cluster:=factor(cluster)]
ggplot(mk3plot,aes(x=age,y=income,color=cluster)) +
    geom_point()
```



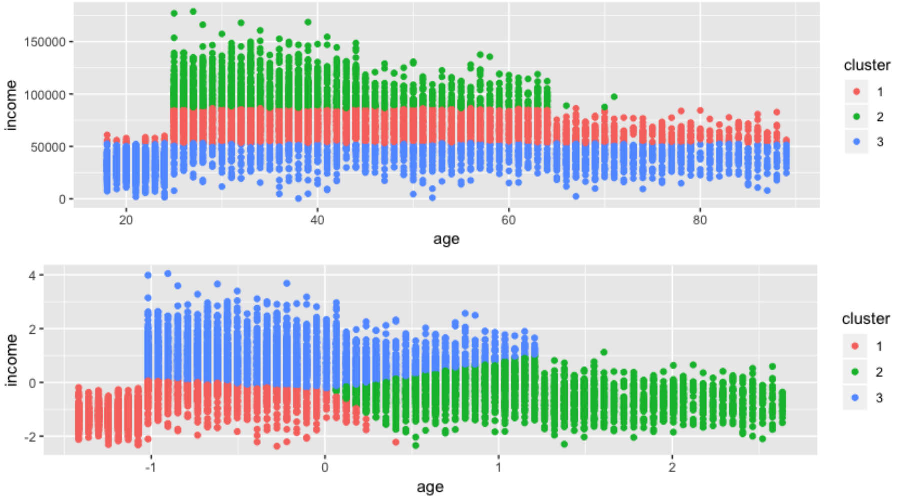
Data Adaption - Scale

A common way to normalize the data is to subtract the mean from each number and then divide by the standard deviation, i.e. the Z-score. There is a function in R called scale() that does it for you. The function scale() is versatile and even though the default step is in deviations, it can be the square root or some other mathematical transformation.

```
marketing[,age_s:=scale(age)]
marketing[,income_s:=scale(income)]

mk3=kmeans(marketing[,.(age_s,income_s)],3)
mk3plot = data.table(age=marketing$age_s,income=
    marketing$income_s,cluster=mk3$cluster)
mk3plot[,cluster:=factor(cluster)]
ggplot(mk3plot,aes(x=age,y=income,color=cluster)) +
    geom_point()
```

Data Adaption - Scale



Hierarchy Clustering

```r
hc = hclust(dist(marketing[,.(income_s,age_s)]),method
     = 'ward.D2')
hc
```

```
> hc

Call:
hclust(d = dist(marketing[, .(income_s, age_s)]), method = "ward.D2")

Cluster method   : ward.D2
Distance         : euclidean
Number of objects: 8105
```

The hclust() function creates the hierarchical clustering model. The dist() function creates a **distance matrix**:

	x	y
1	2	5
2	0	4

$$\sqrt{(2-0)^2+(5-4)^2} = 2.236068$$

Hierarchy Clustering - hclust() Function

```
> hc

Call:
hclust(d = dist(marketing[, .(income_s, age_s)]), method = "ward.D2")

Cluster method   : ward.D2
Distance         : euclidean
Number of objects: 8105
```

The hclust() function creates the hierarchical clustering model. The dist() function creates a **distance matrix**:

```
> x
     [,1] [,2]
[1,]    4    1
[2,]    4    1
[3,]    5    3
[4,]    4    1
[5,]    3    4
```

```
> dist(x)
         1        2        3        4
2 0.000000
3 2.236068 2.236068
4 0.000000 0.000000 2.236068
5 3.162278 3.162278 2.236068 3.162278
```

Clustering Algorithm - ward.D2

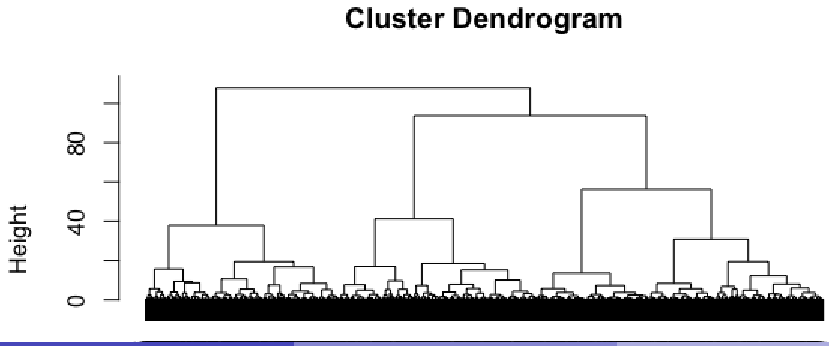
When using hclust() there are a few algorithms to choose from. Ward D2 is often used because of its speed and practicality.

- ward.D - creates groups that are roughly equal in size, but distance matrix is squared
- ward.D2 - creates groups that are roughly equal in size, standard distance matrix is used, compact spherical
- single - **minimal** distance of any object within a cluster and another object
- complete - distance is measured between the **maximum** distance of any object within a cluster and another object
- average - use the **average** distances between points
- mcquitty - an algorithm that uses mean distances, but does not re-calculate them, making it faster
- median - use the **median** distance between points
- centroid - difference between **centroids** between clusters

Clustering Algorithm - They Make a difference

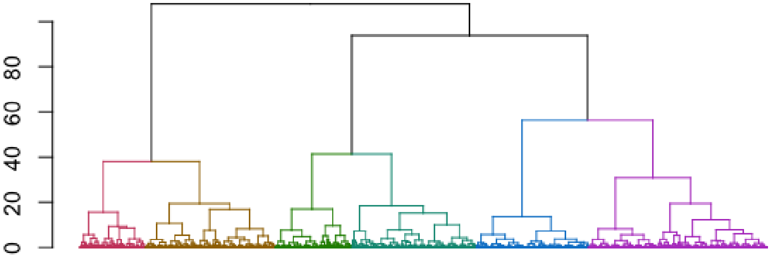
The reason for our purposes why ward.D2 makes sense is because we are doing customer segmentation. That implies that we want to have a few well defined groups, meaning roughly even amount. The complete linkage algorithm would create clusters with varying sizes and single linkage algorithm would cause long snake like cluster.

```
plot(hc)
```

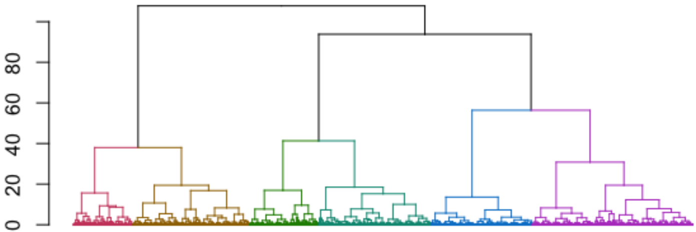


Clustering Algorithm - Visualizing

```
library(dendextend)
dend = as.dendrogram(hc)
dend_six_color = color_branches(dend,k=6)
plot(dend_six_color,leaflab = 'none')
```



Clustering Algorithm - Height and Strength

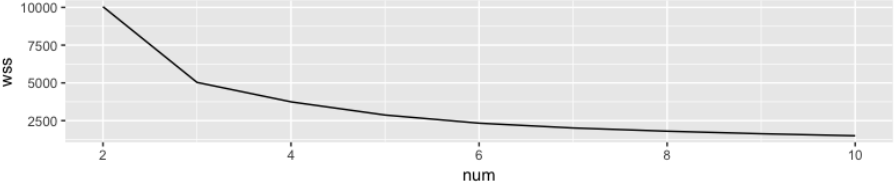


Height is the indication of strength of difference among the branches. The pairing of 2 points next to one another would create a low line. The greater the difference in between the branches implies the stronger the clustering (low variance within groups)

Evaluate Models - Elbow Method

From our previous graph, visually it seemed that we should stop at 6 clusters, but we can also you use kmeans() and the variance within groups to graph how much added clusters reduce the inner variance (tot.withinss)

```
wss=c()
for(i in 2:10)
{
  mk2=kmeans(marketing[,.(age_s,income_s)],i)
  wss[i-1]=mk2$tot.withinss
}
elbowdt = data.table(k=2:10,wss)
ggplot(elbowdt,aes(x=k,y=wss)) + geom_line()
```



Evaluate Models - Comparing Kmeans/Hierarchical

Lets compare 5 to 6 clustering using **both** kmeans and hierarchical.

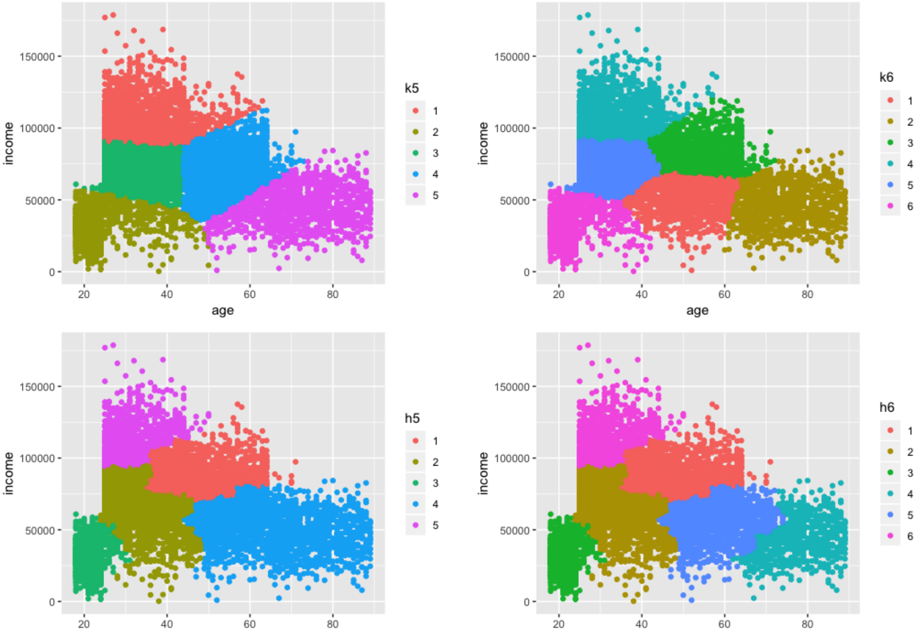
```
k5=kmeans(marketing[,.(income_s,age_s)],5)
k6=kmeans(marketing[,.(income_s,age_s)],6)
marketing[,k5:=k5$cluster]
marketing[,k6:=k6$cluster]
marketing[,h5:=cutree(dend,k=5)]
marketing[,h6:=cutree(dend,k=6)]
marketing[,k5:=factor(k5)]
marketing[,k6:=factor(k6)]
marketing[,h5:=factor(h5)]
marketing[,h6:=factor(h6)]
```

cutree() function basically shows the groups that each observation belongs to **IF** there were a certain number of clusters.

```
cutree(dend,k=5)
```

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

Evaluate Models - Comparing Kmeans/Hierarchical



Evaluate Models - Summary of Hierarchical 6

```r
mm=marketing[,.(mean(age),median(age),max(age),min(age
    ),mean(income),median(income),max(income),min(
    income)),by=h6]
names(mm)=c('group','mean_a','med_a','max_a','min_a','
    mean_i','med_i','max_i','min_i')
```

```
> mm
   group   mean_a med_a max_a min_a    mean_i     med_i     max_i      min_i
1:     1 47.54447    47    71    35  89367.53  88170.32 137557.18 69491.7763
2:     2 33.56966    33    48    24  66591.70  67957.66  94708.92   233.6338
3:     3 21.83069    22    31    18  32431.45  32329.49  60887.37  1484.8486
4:     4 77.19437    77    89    62  43541.17  43044.21  84300.56  2319.2740
5:     5 58.16972    58    74    44  56523.60  57806.34  81988.14   973.4146
6:     6 31.89945    31    50    25 113056.83 111124.93 178676.37 93826.6611
```