* R Data Extraction from CS, Excel, MySQL, MS SQL Server using ODBC
* Data mining and cleaning
* Reshape data frame
* Create linear regression, and logistics regression model
* Plot graph

K Means Clustering in R

## What is K Means Clustering?

K Means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity.. In k means clustering, we have the specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster. Then, the algorithm iterates through two steps:

* Reassign data points to the cluster whose centroid is closest.
* Calculate new centroid of each cluster.

These two steps are repeated till the within cluster variation cannot be reduced any further. The within cluster variation is calculated as the sum of the euclidean distance between the data points and their respective cluster centroids.

## Exploring the data

The iris dataset contains data about sepal length, sepal width, petal length, and petal width of flowers of different species. Let us see what it looks like:

library(datasets)

head(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

1 5.1 3.5 1.4 0.2 setosa

2 4.9 3.0 1.4 0.2 setosa

3 4.7 3.2 1.3 0.2 setosa

4 4.6 3.1 1.5 0.2 setosa

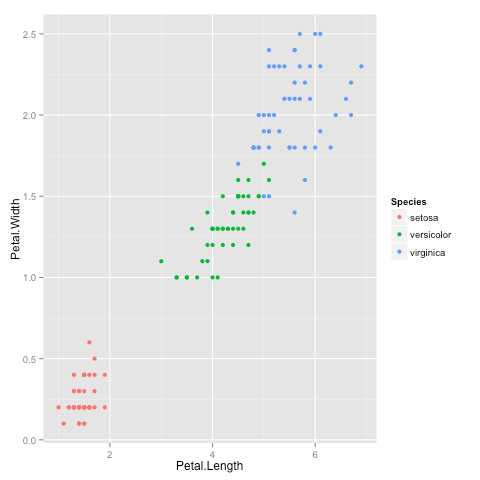
5 5.0 3.6 1.4 0.2 setosa

6 5.4 3.9 1.7 0.4 setosa

After a little bit of exploration, I found that Petal.Length and Petal.Widthwere similar among the same species but varied considerably between different species, as demonstrated below:

library(ggplot2)

ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom\_point()

Here is the plot:  
[](https://i2.wp.com/datascienceplus.com/wp-content/uploads/2015/12/plot1.png)

## Clustering

Okay, now that we have seen the data, let us try to cluster it. Since the initial cluster assignments are random, let us set the seed to ensure reproducibility.

set.seed(20)

irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)

irisCluster

K-means clustering with 3 clusters of sizes 46, 54, 50

Cluster means:

Petal.Length Petal.Width

1 5.626087 2.047826

2 4.292593 1.359259

3 1.462000 0.246000

Clustering vector:

[1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

[35] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

[69] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1

[103] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1

[137] 1 1 2 1 1 1 1 1 1 1 1 1 1 1

Within cluster sum of squares by cluster:

[1] 15.16348 14.22741 2.02200

(between\_SS / total\_SS = 94.3 %)

Available components:

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size" "iter"

[9] "ifault"

Since we know that there are 3 species involved, we ask the algorithm to group the data into 3 clusters, and since the starting assignments are random, we specify nstart = 20. This means that R will try 20 different random starting assignments and then select the one with the lowest within cluster variation.  
We can see the cluster centroids, the clusters that each data point was assigned to, and the within cluster variation.

Let us compare the clusters with the species.

table(irisCluster$cluster, iris$Species)

setosa versicolor virginica

1 0 2 44

2 0 48 6

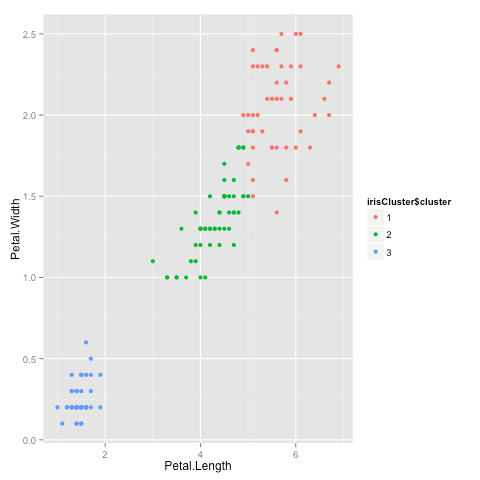
3 50 0 0

As we can see, the data belonging to the setosa species got grouped into cluster 3, versicolor into cluster 2, and virginica into cluster 1. The algorithm wrongly classified two data points belonging to versicolor and six data points belonging to virginica.

We can also plot the data to see the clusters:

irisCluster$cluster <- as.factor(irisCluster$cluster)

ggplot(iris, aes(Petal.Length, Petal.Width, color = iris$cluster)) + geom\_point()

Here is the plot:  
[](https://i2.wp.com/datascienceplus.com/wp-content/uploads/2015/12/plot2.png)

That brings us to the end of the article. I hope you enjoyed it! If you have any questions or feedback, feel free to leave a comment or reach out to me on Twitter.

# K Means Clustering in R Example

Summary: The **kmeans()** function in R requires, at a minimum, numeric data and a number of centers (or clusters).  The cluster centers are pulled out by using **$centers**.  The cluster assignments are pulled by using **$cluster.**  You can evaluate the clusters by looking at **$totss** and **$betweenss**.

**Tutorial Time**: 30 Minutes

R comes with a default K Means function, kmeans().  It only requires two inputs: a matrix or data frame of all numeric values and a number of centers (i.e. your number of clusters or the K of k means).

|  |  |
| --- | --- |
| 1  2  3 | kmeans(x, centers, iter.max = 10, nstart = 1,   algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",   "MacQueen"), trace=FALSE) |

* X is your data frame or matrix.  All values must be numeric.
  + If you have an ID field make sure you drop it or it will be included as part of the centroids.
* Centers is the K of K Means.  centers = 5 would results in 5 clusters being created.
  + You have to determine the appropriate number for K.
* iter.max is the number of times the algorithm will repeat the cluster assignment and moving of centroids.
* nstart is the number of times the initial starting points are re-sampled.
  + In the code, it looks for the initial starting points that have the lowest within sum of squares (withinss).
  + That means it tries “nstart” samples, does the cluster assignment for each data point “nstart” times, and picks the centers that have the lowest distance from the data points to the centroids.
* trace gives a verbose output showing the progress of the algorithm.

## K Means Algorithms in R

The out-of-the-box K Means implementation in R offers three algorithms (Lloyd and Forgy are the same algorithm just named differently).

The default is the Hartigan-Wong algorithm which is often the fastest.  This StackOverflow answer is the closest I can find to showing some of the differences between the algorithms.

Research Paper References:

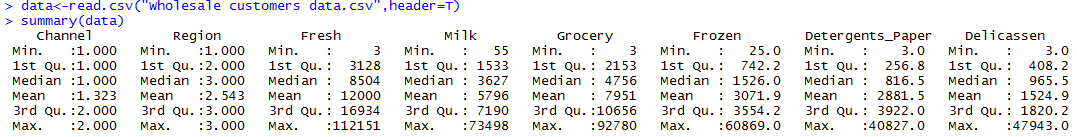
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | Forgey, E. (1965). “Cluster Analysis of Multivariate Data:  Efficiency vs. Interpretability of Classification”. In: Biometrics.    Lloyd, S. (1982). “Least Squares Quantization in PCM”.  In: IEEE Trans. Information Theory.    Hartigan, J. A. and M. A. Wong (1979). “Algorithm AS  136: A k-means clustering algorithm”. In: Applied Statistics  28.1, pp. 100–108.    MacQueen, J. B. (1967). “Some Methods for classification  and Analysis of Multivariate Observations”. In: Berkeley  Symposium on Mathematical Statistics and Probability |

## kmeans() R Example

Let’s take an example of clustering customers from a wholesale customer database.  You can download the data I’m using from the Berkley UCI Machine Learning Repository here.

Let’s start off by reading in the data (Note: You may have to use setwd() to change your directory to wherever you’re storing your data).  After reading in the data, let’s just get a quick summary.

|  |  |
| --- | --- |
| 1  2 | data <-read.csv("Wholesale customers data.csv",header=T)  summary(data) |

[](http://www.learnbymarketing.com/wp-content/uploads/2015/07/kmeans-r-summary.png)

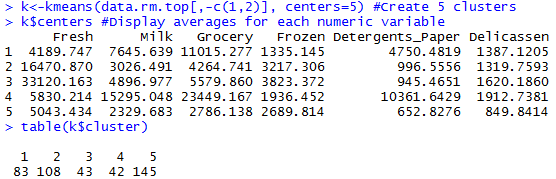
There’s obviously a big difference for the top customers in each category (e.g. Fresh goes from a min of 3 to a max of 112,151).  Normalizing / scaling the data won’t necessarily remove those outliers.  Doing a log transformation might help.   We could also remove those customers completely.  From a business perspective, you don’t really need a clustering algorithm to identify what your top customers are buying.  You usually need clustering and segmentation for your middle 50%.

With that being said, let’s try removing the top 5 customers from each category.  We’ll use a custom function and create a new data set called data.rm.top

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | top.n.custs <- function (data,cols,n=5) { #Requires some data frame and the top N to remove  idx.to.remove <-integer(0) #Initialize a vector to hold customers being removed  for (c in cols){ # For every column in the data we passed to this function  col.order <-order(data[,c],decreasing=T) #Sort column "c" in descending order (bigger on top)  #Order returns the sorted index (e.g. row 15, 3, 7, 1, ...) rather than the actual values sorted.  idx <-head(col.order, n) #Take the first n of the sorted column C to  idx.to.remove <-union(idx.to.remove,idx) #Combine and de-duplicate the row ids that need to be removed  }  return(idx.to.remove) #Return the indexes of customers to be removed  }  top.custs <-top.n.custs(data,cols=3:8,n=5)  length(top.custs) #How Many Customers to be Removed?  data[top.custs,] #Examine the customers  data.rm.top<-data[-c(top.custs),] #Remove the Customers |

Now, using data.rm.top, we can perform the cluster analysis.  Important note: We’ll still need to drop the Channel and Region variables.  These are two ID fields and are not useful in clustering.

|  |  |
| --- | --- |
| 1  2  3  4 | set.seed(76964057) #Set the seed for reproducibility  k <-kmeans(data.rm.top[,-c(1,2)], centers=5) #Create 5 clusters, Remove columns 1 and 2  k$centers #Display&nbsp;cluster centers  table(k$cluster) #Give a count of data points in each cluster |

[](http://www.learnbymarketing.com/wp-content/uploads/2015/07/kmeans-r-results.png)

Now we can start interpreting the cluster results:

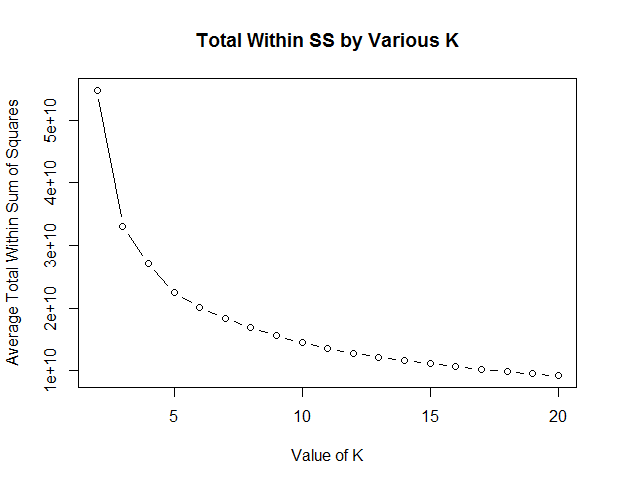
* Cluster 1 looks to be a heavy Grocery and above average Detergents\_Paper but low Fresh foods.
* Cluster 3 is dominant in the Fresh category.
* Cluster 5 might be either the “junk drawer” catch-all cluster or it might represent the small customers.

A measurement that is more relative would be the withinss and betweenss.

* k$withinss would tell you the sum of the square of the distance from each data point to the cluster center.  Lower is better.  Seeing a high withinss would indicate either outliers are in your data or you need to create more clusters.
* k$betweenss tells you the sum of the squared distance between cluster centers.  Ideally you want cluster centers far apart from each other.

**It’s important to try other values for K**.  You can then compare withinss and betweenss.  This will help you select the best K.   For example, with this data set, what if you ran K from 2 through 20 and plotted the total within sum of squares?  You should find an “elbow” point.  Wherever the graph bends and stops making gains in withinss you call that your K.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | rng<-2:20 #K from 2 to 20  tries <-100 #Run the K Means algorithm 100 times  avg.totw.ss <-integer(length(rng)) #Set up an empty vector to hold all of points  for(v in rng){ # For each value of the range variable   v.totw.ss <-integer(tries) #Set up an empty vector to hold the 100 tries   for(i in 1:tries){   k.temp <-kmeans(data.rm.top,centers=v) #Run kmeans   v.totw.ss[i] <-k.temp$tot.withinss#Store the total withinss   }   avg.totw.ss[v-1] <-mean(v.totw.ss) #Average the 100 total withinss  }  plot(rng,avg.totw.ss,type="b", main="Total Within SS by Various K",   ylab="Average Total Within Sum of Squares",   xlab="Value of K") |



This plot doesn’t show a very strong elbow.  Somewhere around K = 5 we start losing dramatic gains.  So I’m satisfied with 5 clusters.

You now have all of the bare bones for using kmeans clustering in R.

Here’s the full code for this tutorial.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | data <-read.csv("Wholesale customers data.csv",header=T)  summary(data)  top.n.custs <- function (data,cols,n=5) { #Requires some data frame and the top N to remove  idx.to.remove <-integer(0) #Initialize a vector to hold customers being removed  for (c in cols){ # For every column in the data we passed to this function  col.order <-order(data[,c],decreasing=T) #Sort column "c" in descending order (bigger on top)  #Order returns the sorted index (e.g. row 15, 3, 7, 1, ...) rather than the actual values sorted.  idx <-head(col.order, n) #Take the first n of the sorted column C to  idx.to.remove <-union(idx.to.remove,idx) #Combine and de-duplicate the row ids that need to be removed  }  return(idx.to.remove) #Return the indexes of customers to be removed  }  top.custs <-top.n.custs(data,cols=3:8,n=5)  length(top.custs) #How Many Customers to be Removed?  data[top.custs,] #Examine the customers  data.rm.top <-data[-c(top.custs),] #Remove the Customers  set.seed(76964057) #Set the seed for reproducibility  k <-kmeans(data.rm.top[,-c(1,2)], centers=5) #Create 5 clusters, Remove columns 1 and 2  k$centers #Display cluster centers  table(k$cluster) #Give a count of data points in each cluster  rng<-2:20 #K from 2 to 20  tries<-100 #Run the K Means algorithm 100 times  avg.totw.ss<-integer(length(rng)) #Set up an empty vector to hold all of points  for(v in rng){ # For each value of the range variable  v.totw.ss<-integer(tries) #Set up an empty vector to hold the 100 tries  for(i in 1:tries){  k.temp<-kmeans(data.rm.top,centers=v) #Run kmeans  v.totw.ss[i]<-k.temp$tot.withinss#Store the total withinss  }  avg.totw.ss[v-1]<-mean(v.totw.ss) #Average the 100 total withinss  }  plot(rng,avg.totw.ss,type="b", main="Total Within SS by Various K",  ylab="Average Total Within Sum of Squares",  xlab="Value of K") |

· © 2018 Learn by Marketing · Designed by Themes & Co ·