K-Means Clustering Algorithm R Notebook

```
# 1. LOAD DATA

# Import package
library(ggplot2)
# Load Data
kdata = read.csv("/home/ktd2001/Downloads/KMeansData_Group3.csv", header=FALSE)
# summary of data and checking for missing values
summary(kdata)
```

```
##        V1                V2
##  Min.   :-0.3793   Min.   : 0.02133
##  1st Qu.: 0.8656   1st Qu.: 5.21858
##  Median : 2.3535   Median : 6.83760
##  Mean   : 3.5005   Mean   : 6.24060
##  3rd Qu.: 6.6170   3rd Qu.: 7.97631
##  Max.   :10.3086   Max.   :10.45902
```

```
# Dimension of data
dim(kdata)
```

```
## [1] 873   2
```

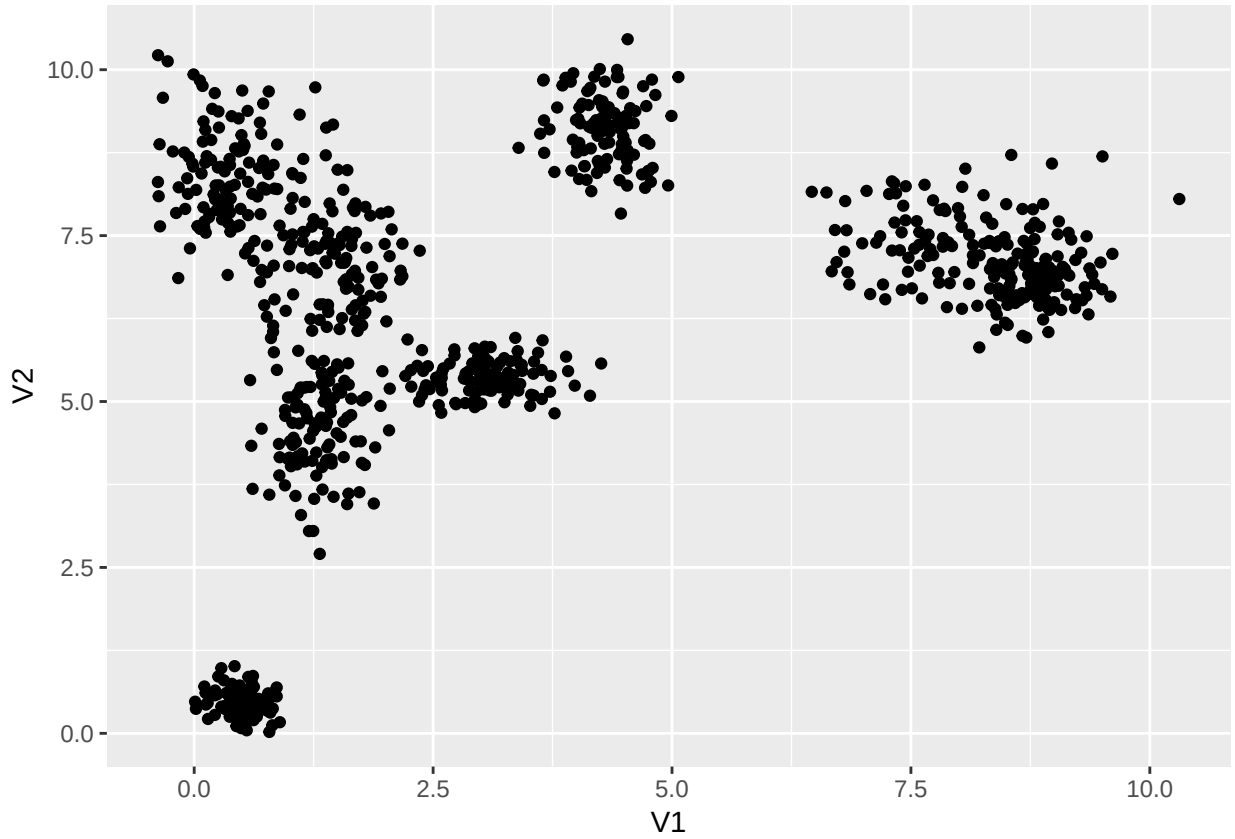
```
# Look at the first several rows
head(kdata)
```

```
##           V1        V2
## 1 0.59087880 0.6699168
## 2 0.81788914 0.1212022
## 3 0.00958507 0.4764786
## 4 0.12246082 0.4334967
## 5 0.75625048 0.5213313
## 6 0.44507971 0.4299888
```

```
# Column names
colnames(kdata)
```

```
## [1] "V1" "V2"
```

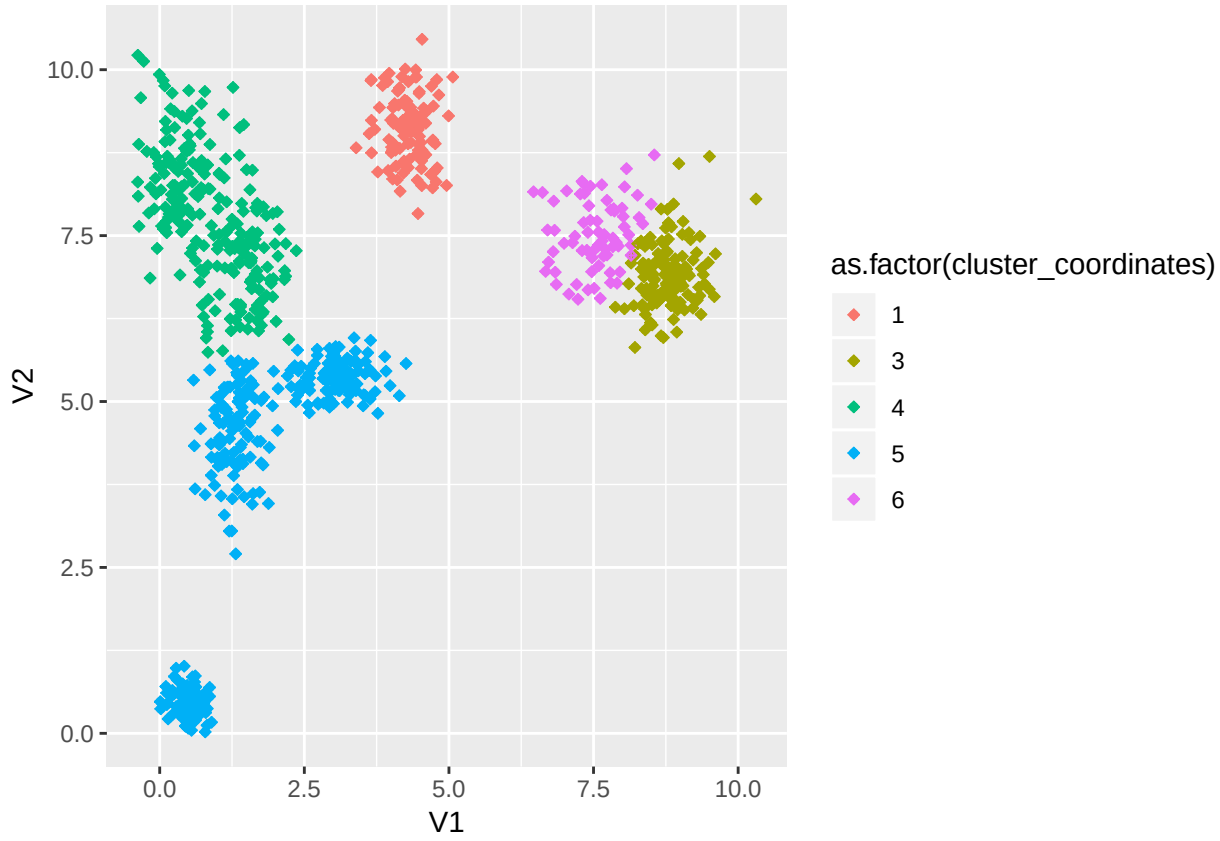
```
# plot data
ggplot(kdata, aes(V1, V2), colours(TRUE)) + geom_point()
```




```r
# 2. CREATE INITIAL CENTROIDS

# Use K-means function with for initialization
kcluster <- function(V1, V2, nclus) {

  # Create random cluster centers
  xcen <- runif(n = nclus, min = min(V1), max = max(V1))
  ycen <- runif(n = nclus, min = min(V2), max = max(V2))

  # Create data frame where data points and cluster assignment are
  kdata <- data.frame(V1, V2, cluster_coordinates = 1)
  cluster_coordinates <- data.frame(name = 1:nclus, xcen = xcen, ycen = ycen)

  finish <- FALSE

  while(finish == FALSE) {

# 3. ASSIGN DATA POINTS TO NEAREST CENTROID USING EUCLIDEAN DISTANCE FORMULA

   # Assign random clusters with minimum distance to each data point
    for (i in 1:length(V1)) {
      dist <- sqrt((V1[i]- cluster_coordinates$xcen)^2 + (V2[i]-cluster_coordinates$ycen)^2)
      kdata$cluster_coordinates[i] <- which.min(dist)
    }

    xcen_old <- cluster_coordinates$xcen
```

```
    ycen_old <- cluster_coordinates$ycen


# 4. RE-CALCULATE DATA TO CENTROIDS
    # Calculate a set of new cluster centers
    for(i in 1:nclus) {
      cluster_coordinates[i,2] <- mean(subset(kdata$V1, kdata$cluster_coordinates == i))
      cluster_coordinates[i,3] <- mean(subset(kdata$V2, kdata$cluster_coordinates == i))
    }

    # Interupt the loop if there is no change in cluster coordinates
    if(identical(xcen_old, cluster_coordinates$xcen) & identical(ycen_old, cluster_coordinates$ycen))
      finish <- TRUE
  }
  kdata
}

# 5. RE-aSSIGN CENTROIDS TO IMPROVE ON CLOSENESS TO CENTROIDS

# Compute kmeans function to sample data for a k = 6
cluster <- kcluster(kdata$V1, kdata$V2, nclus=6)
cluster.centers <- aggregate(.~cluster_coordinates, cluster, mean)

# Plot results and have algorithm created clusters
ggplot(cluster, aes(V1, V2, color= as.factor(cluster_coordinates))) + geom_point(size=1) +
  geom_point(kdata=cluster.centers, aes(V1, V2, col=as.factor(cluster_coordinates)), pch=9, size=1)
```

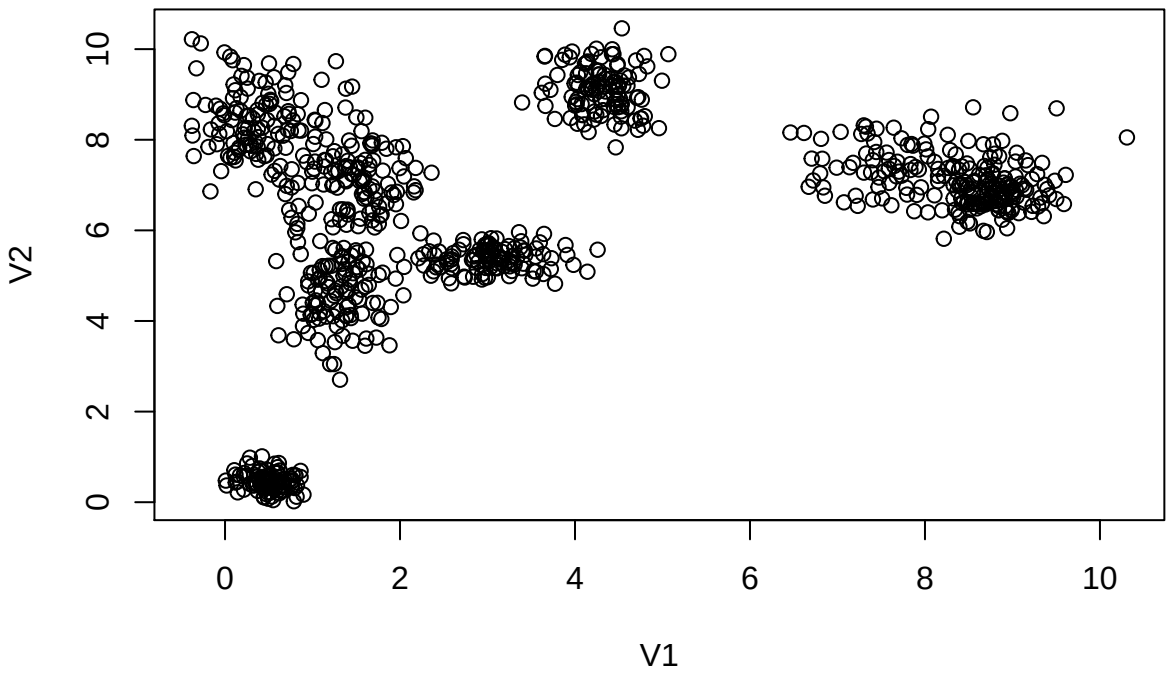
```
## Warning: Ignoring unknown parameters: kdata
```




```
### After several runs of the algorithm, the centroids of some of the clusters changed
### from their intial centers and the cluster numbers also changed from 4, 5 and 6.
### 6 was the final number of clusters.

# 6. USE KMEANS FUNCTION IN R AND 'ELBOW METHOD' TO FIND OPTIMAL NUMBER OF CLUSTERS

# Change dataset name when using K-means function in R
kdata -> kdata1
head(kdata1)
```

```
##           V1        V2
## 1 0.59087880 0.6699168
## 2 0.81788914 0.1212022
## 3 0.00958507 0.4764786
## 4 0.12246082 0.4334967
## 5 0.75625048 0.5213313
## 6 0.44507971 0.4299888
```

```
# Plot kdata1
plot(kdata1)
```




```
# We can see 4, 5 or even 6 naturally occuring clusters. I will settle on 6 clusters

# K-means clustering function needs dataset and # of clusters declared.
kdata1.results <- kmeans( kdata1, centers = 6)

# Show results
kdata1.results
```

```
## K-means clustering with 6 clusters of sizes 115, 112, 127, 100, 220, 199
##
## Cluster means:
##          V1        V2
## 1 1.2775979 4.7022519
## 2 4.3032836 9.1063747
## 3 2.8819448 5.5166619
## 4 0.5018009 0.4565123
## 5 8.3966141 7.0825943
## 6 0.8220758 7.9544464
##
## Clustering vector:
##   [1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [36] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [71] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5
## [106] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [141] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [176] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6
```

```
## [211] 3 1 6 6 6 1 1 6 6 6 6 6 6 6 6 3 1 3 6 6 6 6 6 3 6 6 3 3 6 6 6 6
## [246] 6 3 6 1 6 6 6 6 6 6 3 6 6 3 6 1 6 6 1 6 6 6 1 6 6 6 6 6 6 3 1 1 6 3 6
## [281] 6 6 6 6 6 3 3 6 1 6 6 6 6 6 6 6 6 3 6 3 6 6 3 6 3 6 1 6 6 6 6 6 6 6 6
## [316] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 1 6 6
## [351] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
## [386] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
## [421] 6 6 6 6 6 6 6 6 6 6 6 6 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [456] 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
## [491] 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
## [526] 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
## [561] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [596] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [631] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 5 5 5 5 5 5 5 5 5 5
## [666] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [701] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [736] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1
## [771] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 6 1 1 1 1 1 1 1 1 1 1 6
## [806] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [841] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1]  79.475094  40.059144  67.699014   7.866494 177.134816 226.212519
##  (between_SS / total_SS =  95.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

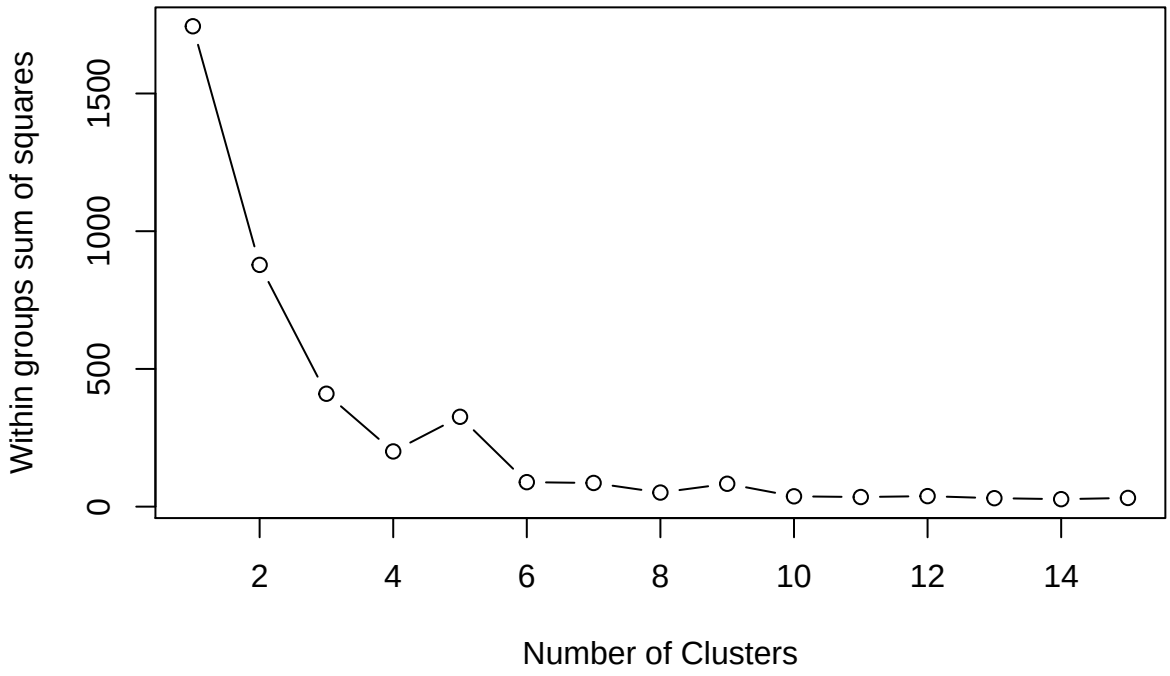
```
### Results reveal how many clusters, the size of the clusters, mean of each cluster center,
### cluster vectors and sum of squares for clusters.

# Isolate to display cluster size information
kdata1.results$size
```

```
## [1] 115 112 127 100 220 199
```

```
# Dataset name changed
kdata -> kdata1

# Review data
head(kdata1)
```

```
##           V1        V2
## 1 0.59087880 0.6699168
## 2 0.81788914 0.1212022
## 3 0.00958507 0.4764786
## 4 0.12246082 0.4334967
## 5 0.75625048 0.5213313
## 6 0.44507971 0.4299888
```

```r
# Seed set for reproducible results
set.seed(100)

# To have kdata return as numeric vector the length of x values
kdata1_scaled <- as.data.frame(lapply(kdata1, scale))

# Create loop to plot and determine the optimial number of clusters
mss <- (nrow(kdata1_scaled)-1)*sum(apply(kdata1_scaled,2,var))
  # Vary K parameter from 1-15
  for (i in 2:15) mss[i] <- sum(kmeans(kdata1_scaled,centers=i)$withinss)
  # Plot graph
plot(1:15, mss, type="b", xlab="Number of Clusters", ylab="Within groups sum of squares")
```




```
### The "Elbow Method" shows K = 4 - 6 as the optimial number of clusters. I think we can
### settle on 6 being the ideal # of clusters. In comparison, the "Elbow Method" approach is
### probably more accurate over the traditional method in determining the ideal number of
### clusters for this dataset.
```