Assignment 7: K-means Clustering using EM algorithm

Prepared by:

- Josh Levine (jl2108)
- Harsh Patel (hkp49)
- Jaini Patel (jp1891)
- Yifan Liao (yl1463)
- Aayush Shah (avs93)

EM Algorithm

The EM algorithm is used for obtaining maximum likelihood estimates of parameters when some of the data is missing. More generally, however, the EM algorithm can also be applied when there is unobserved data, which was never intended to be observed in the first place. In that case, we simply assume that the latent data is missing and proceed to apply the EM algorithm. The EM algorithm has many applications throughout statistics. It is often used for example, in machine learning and data mining applications, and in Bayesian statistics where it is often used to obtain the mode of the posterior marginal distributions of parameters.

EM is an iterative algorithm with two linked steps:

- E-step: fill-in hidden values using inference and
- M-step: apply standard MLE method to completed data.

EM algorithm always converges to a local optimum of the likelihood. It is mostly used for the multivariate distribution like Gaussian Mixture Models(GMM). A simple example of EM algorithm is K-means Clustering.

K-Means Clustering

K-means is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid. The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.

Steps to perform k-means clustering

1) Choose the number of clusters k. (no. of means k)
2) Select k random points from the data as centroids/means.
3) Assign all the points in the data to the closest cluster centroid.
4) Compute the centroids of the newly formed clusters.
5) Repeat step 3 and 4.

Stopping Criteria for K-Means Clustering

There are essentially three stopping criteria that can be adopted to stop the K-means algorithm:

1) Centroids/mean of newly formed clusters do not change.
2) Points remain in the same cluster.
3) Maximum number of iterations are reached.

We can stop the algorithm if the centroids of newly formed clusters are not changing. Even after multiple iterations, if we are getting the same centroids for all the clusters, we can say that the algorithm is not learning any new pattern and it is a sign to stop the training.

Another clear sign that we should stop the training process if the points remain in the same cluster even after training the algorithm for multiple iterations.

DUNN Index

Dunn Index shows the goodness of fitness of clustering. Dunn index is the ratio of the minimum of inter-cluster distances and maximum of intracluster distances.

- Minimun Inter-cluster Distances, the distance between even the closest clusters should be more which will eventually make sure that the clusters are far away from each other.
- Maximum Intra-cluster Distances, the maximum distance between the cluster centroids and the points should be minimum which will eventually make sure that the clusters are compact.

$$DunnIndex = min(Inter - ClusterDistance)/max(Intra - clusterDistance)$$

The values of the Dunn index have to be maximized because we want to maximize minimum the distance between two clusters and minimize the maximum distance (or spread) within the cluster. Once the algorithm converges, the cluster assignment remains constant and hence the Dunn value gets stable.With each iteration, the value of the Dunn index should increase as we are making our clusters better and better. This can be observed from the Dunn index calculated at each iteration for the k-means algorithm.

Below are the code chunks and corresponding outputs of the functions used to perform k means clustering

"init" function assigns cluster value to each data value.

```
library(MASS)
library(ggplot2)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::select() masks MASS::select()
```

```
library(matlib) #for matrix function


#  assign clusters to each data point.
init <- function(data, k){
  n = nrow(data)
  clusters = rep(1:k, length.out = n, replace = TRUE)
  data = cbind(data, clusters)
  return(data)
}
```

"parameters" function calculates the parameters (mean and variance) of all the clusters, based on the data points assigned to each cluster.

```
parameters <- function(data, k){
  features = ncol(data)-1
  #taking mean feature wise
  mean = aggregate(data, by = list(data$clusters), mean, na.rm = TRUE)
  cov_list <- lapply(sort(unique(data$clusters)),
                     function(x)
                       cov(data[data$clusters==x,-(features+1)],use="na.or.complete"))

  return(list(mean = mean[-1], cov_list = cov_list ))
}
```

"euclidean_distance" finds the Euclidean distance between two points.

```
euclidean_distance <- function(x1, x2){
  return(sqrt(sum((x1 - x2) ^ 2)))
}
```

"assign_euclidean" function re-assigns the clusters to each data point based on the euclidean distance of the point from the cluster centroid.

```
#re-assign cluster based on the euclidean distance from the cluster mean
assign_euclidean <- function(data, clustered_list){

  n = nrow(data)
  features = ncol(data)-1

  for(i in c(1:n)){
    min_dist = +Inf
    for( j in  unique(data$clusters) ){
        mean = clustered_list$mean[j,-(features+1)]
        euc_distance = euclidean_distance(data[i,-ncol(data)], mean )
        if(euc_distance < min_dist){
          min_dist = euc_distance
          #assign the cluster with min distance from the data point
          data[i,]$clusters = j
         }
      }
    }
  return(data)
}
```

"plot" function plots a scatter plot showing relation between any two features of data.

```
plot<- function(data){
  print(data %>%
          ggplot(aes(data[,1],data[,2], color = as.factor(clusters))) +
          geom_point() +
          scale_color_manual(
            values=c("red", "blue", "green")))
}
```

Below are few functions used to find intracluster distance, intercluster distance and dunn index.

```
intracluster_dist <- function(cluster_mean, cluster_data){
  n = nrow(cluster_data)
```

```
  max = 0
  for (i in c(1:n)){
    euc_distance = euclidean_distance(cluster_mean, cluster_data[i,])
    if(euc_distance>max){
      max = euc_distance
    }
  }
  return(max)
}

get_max_intracluster_distance <- function(data, mean){
  features = ncol(data)-1
  for (i in c(1:k)){
    t = lapply(sort(unique(data$clusters)),
               function(x)
                 intracluster_dist(mean[x,],data[data$clusters==x,-(features+1)]))
  }

  return(max(unlist(t)))
}

get_min_intercluster_distance <- function(mean, k){
  min = +Inf
  for (i in c(1:(k-1))){
    for (j in c((i+1):k)){
      euc_distance = euclidean_distance(mean[i,], mean[j,])
      if(euc_distance<min){
        min = euc_distance
      }
    }
  }
  return(min)
}

dunn_index <- function(data, mean, k){
  features = ncol(data)

  min_intercluster_distance = get_min_intercluster_distance(mean[-features], k)
  max_intracluster_distance = get_max_intracluster_distance(data, mean[-features])
  return(min_intercluster_distance/max_intracluster_distance)
}
```

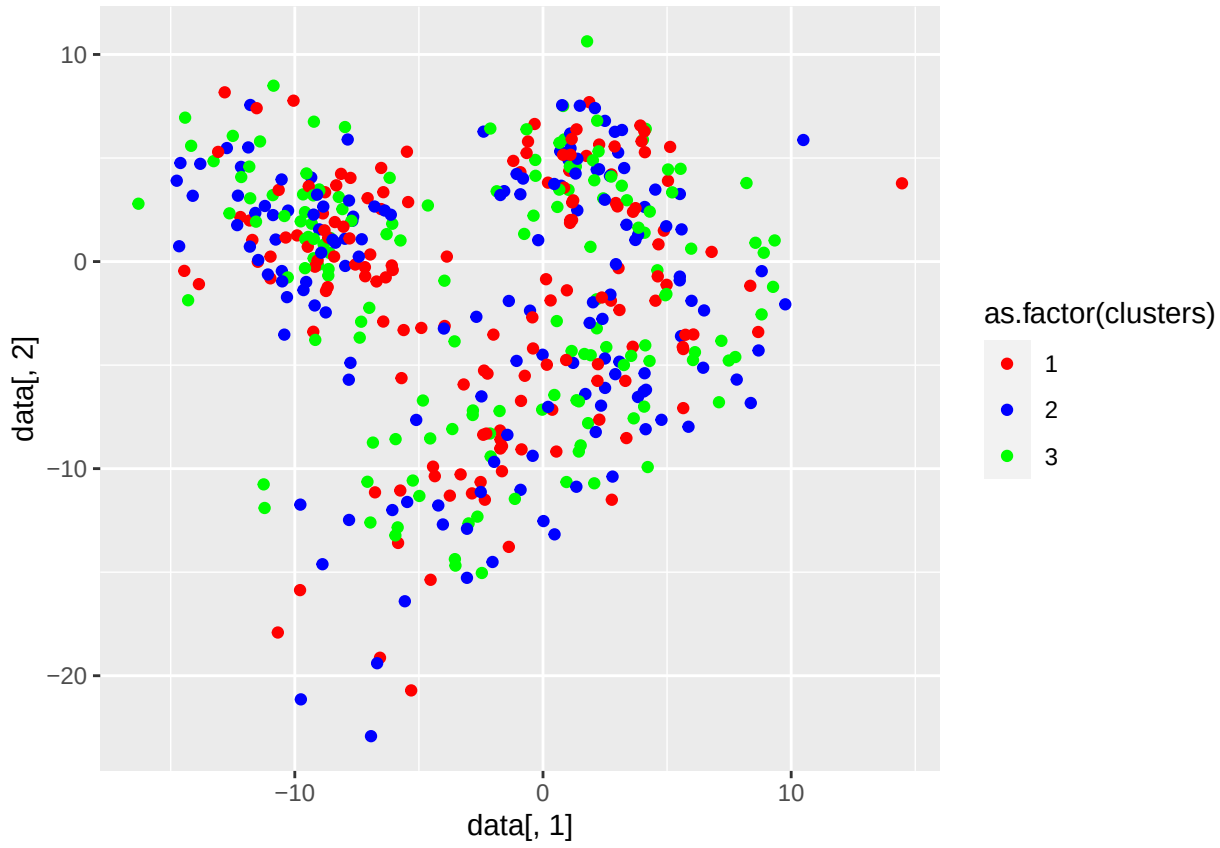
"k_means" function is the function that computes the k_means clustering on the data points.

```
k_means <- function(data, k){
  data = init(data, k)
  data_cluster = data
  plot(data)
  features = ncol(data_cluster)

  #convergence of dunn index
  prev_dunn_index = 0
  difference = 1
  n_steps = 1
```

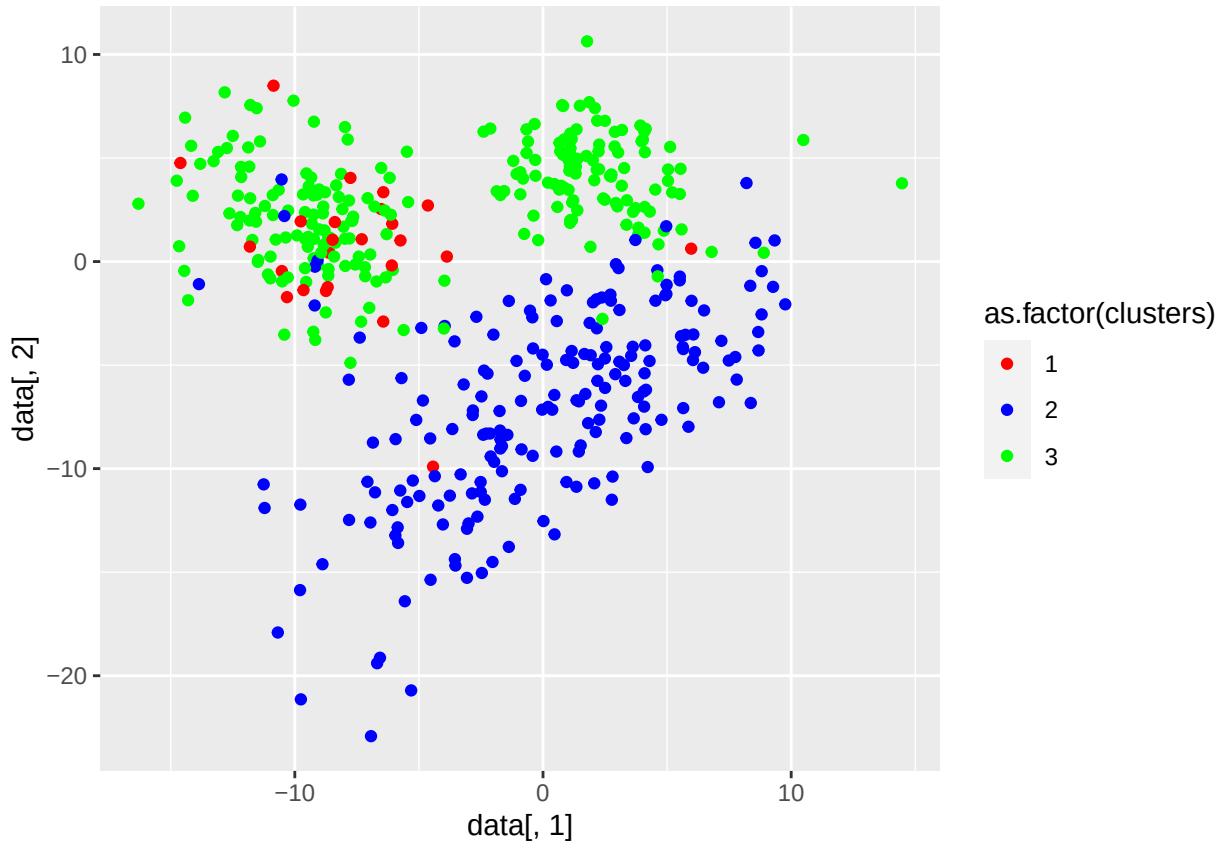
```
  # stop when dunn index becomes constant
  while(difference!=0){
    print(paste(n_steps))
    #clustered_list contains the cluster parameters of mean and covariance for each cluster
    clustered_list = parameters(data, k)
    #new column of 'clusters' gets appended
    data = assign_euclidean(data,clustered_list )

    new_dunn_index = dunn_index(data, clustered_list$mean[-(features)], k)
    print(new_dunn_index)
    plot(data)
    difference = abs(new_dunn_index - prev_dunn_index)
    prev_dunn_index = new_dunn_index
    n_steps = n_steps + 1
  }

  return(list(data = data, clustered_list = clustered_list))
}
```

Below are the codes to simulate the above mentioned functions and the plots used to visualize the pair-pair plots.

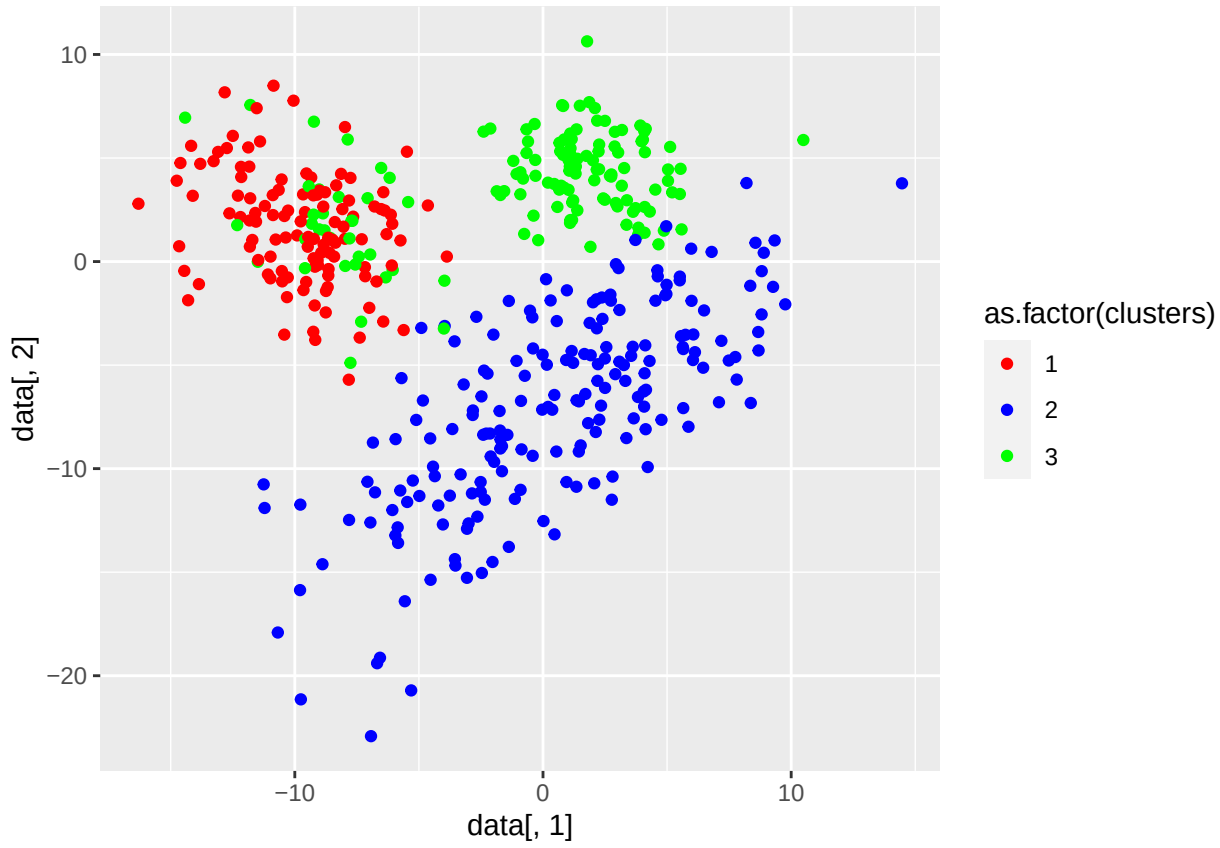
```
df <- read.csv("kmeans.csv")
data = data.frame(df)

#cols = c(1:5)
#rows = 1:nrow(df)
#data = df[rows,cols] #to convert csv into dataframe

k = 3 #number of clusters to be made.number of means.

output = k_means(data, k)#gives dunn index after each iteration and the iterated clusters.
```



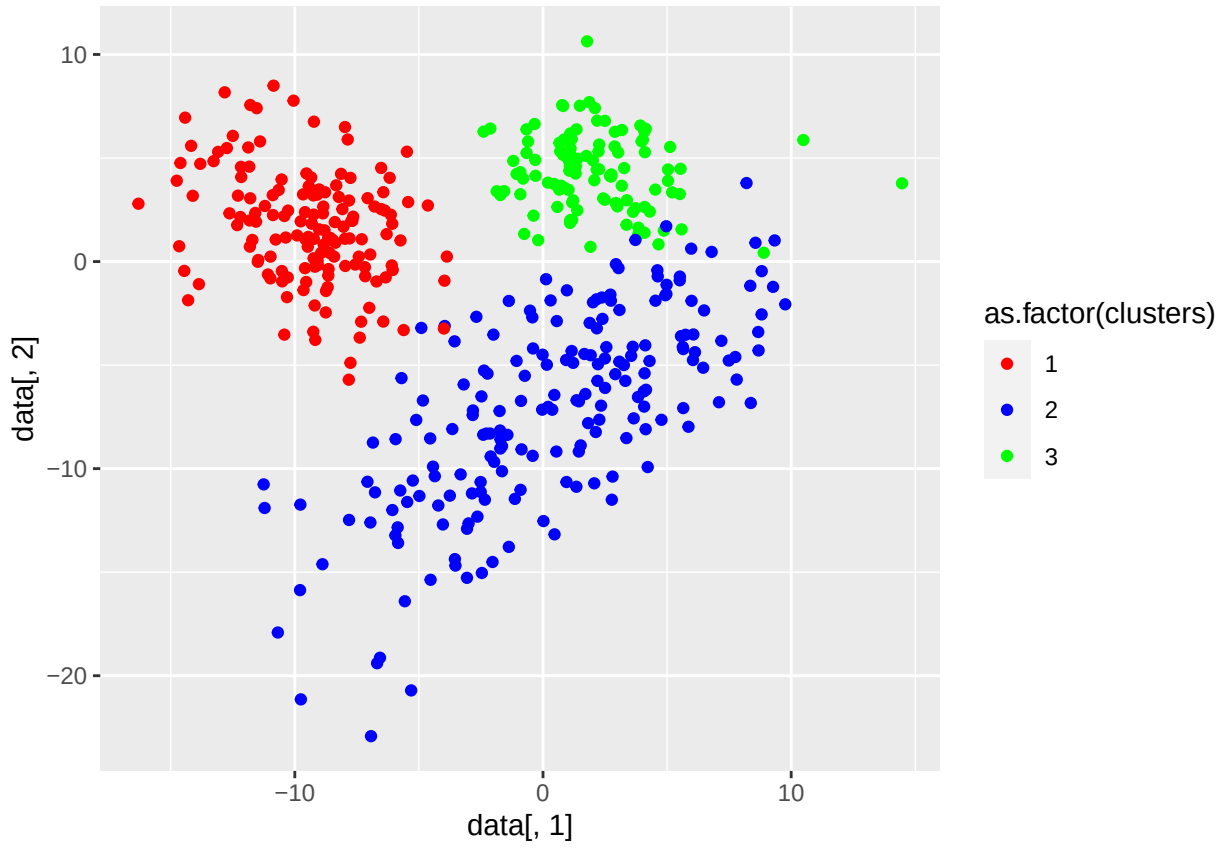

```
## [1] "1"
## [1] 0.01946066
```



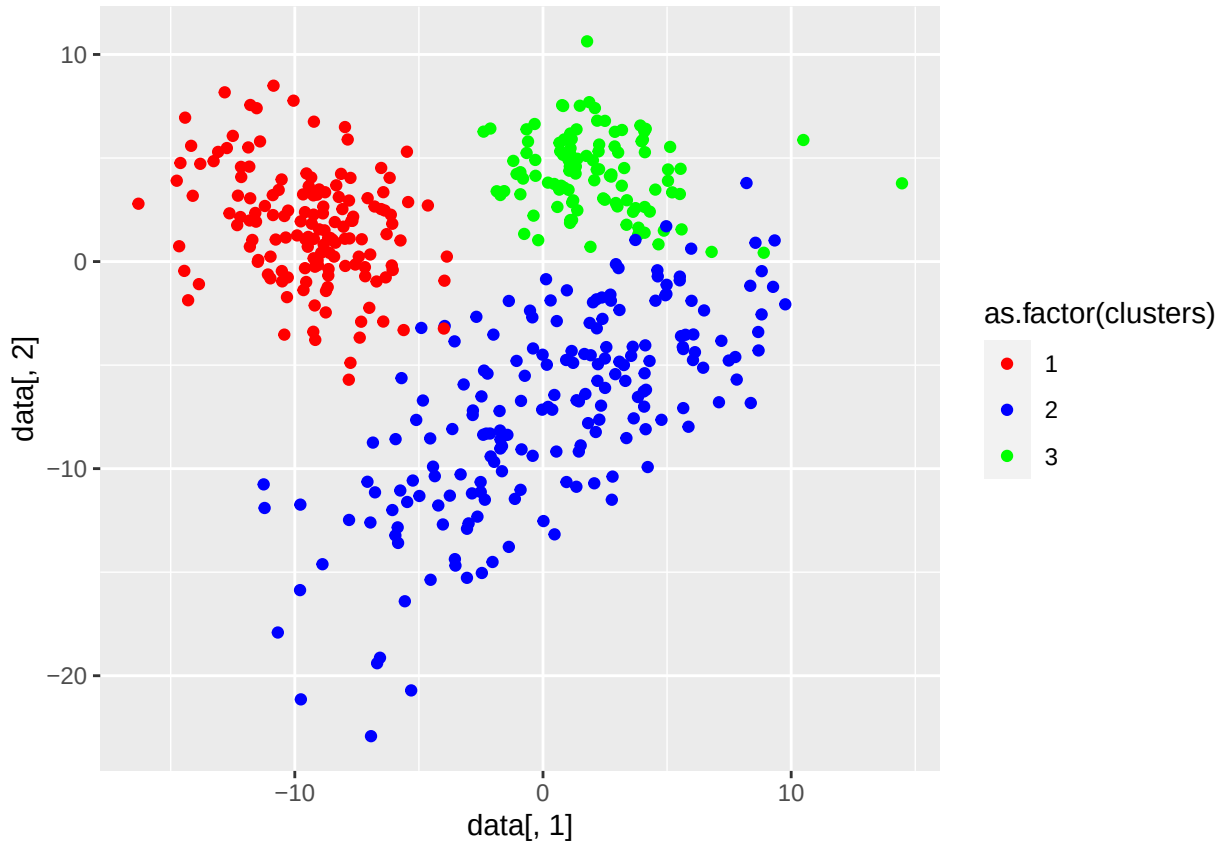

```
## [1] "2"
## [1] 0.2814332
```



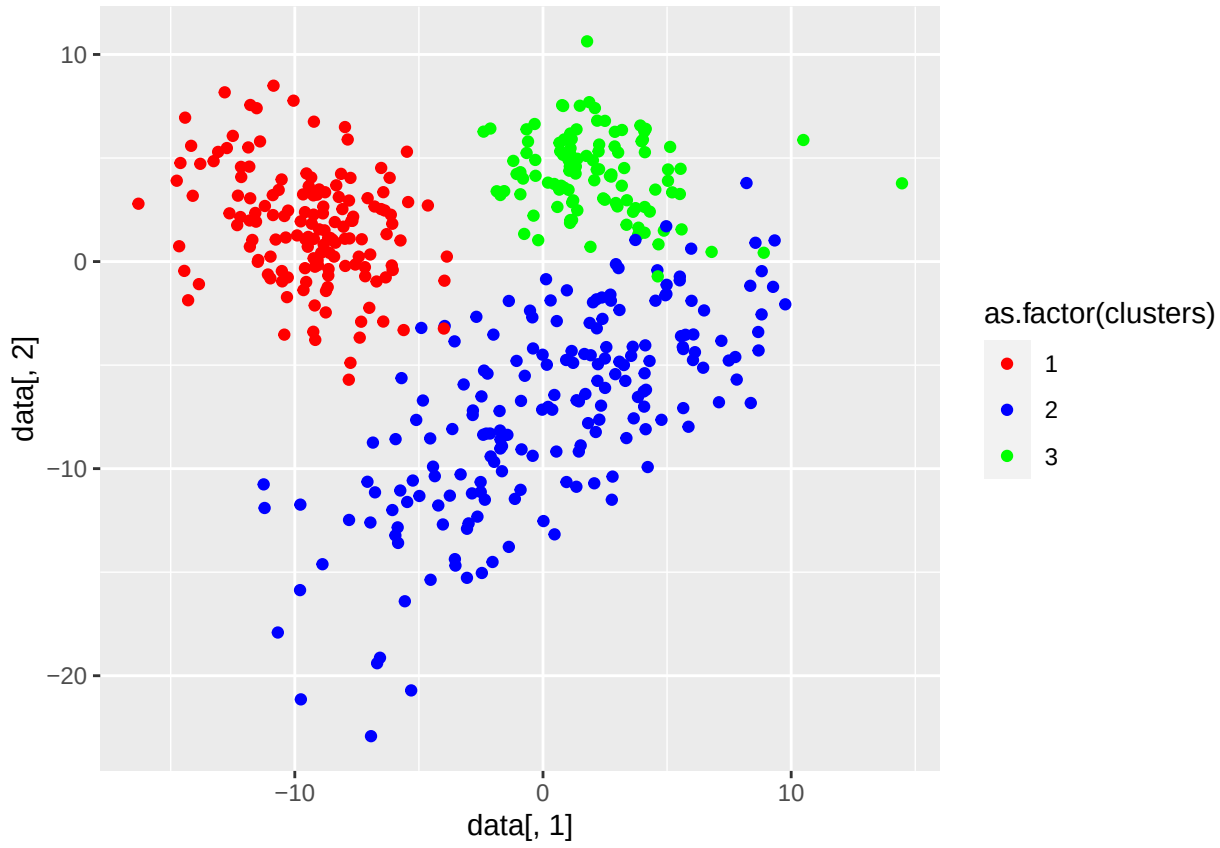

```
## [1] "3"
## [1] 0.418629
```



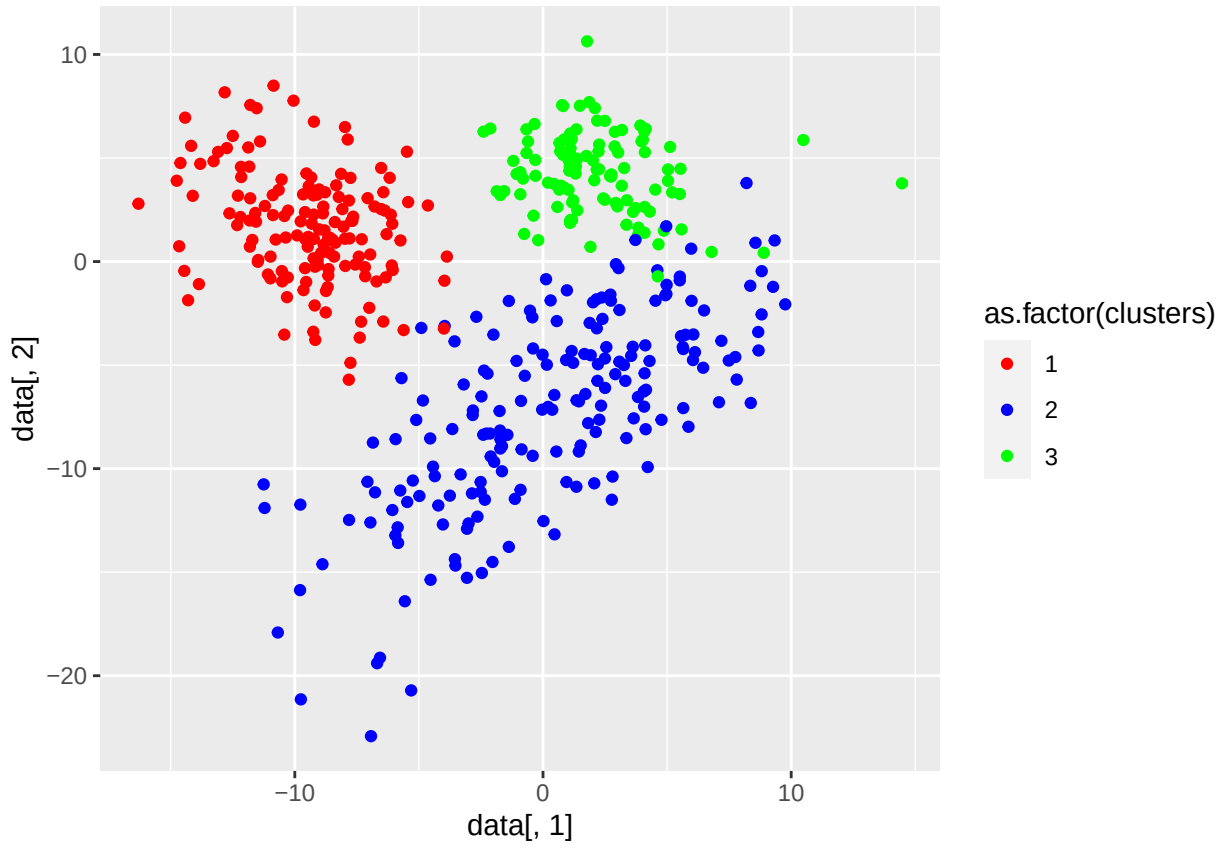

```
## [1] "4"
## [1] 0.5137268
```



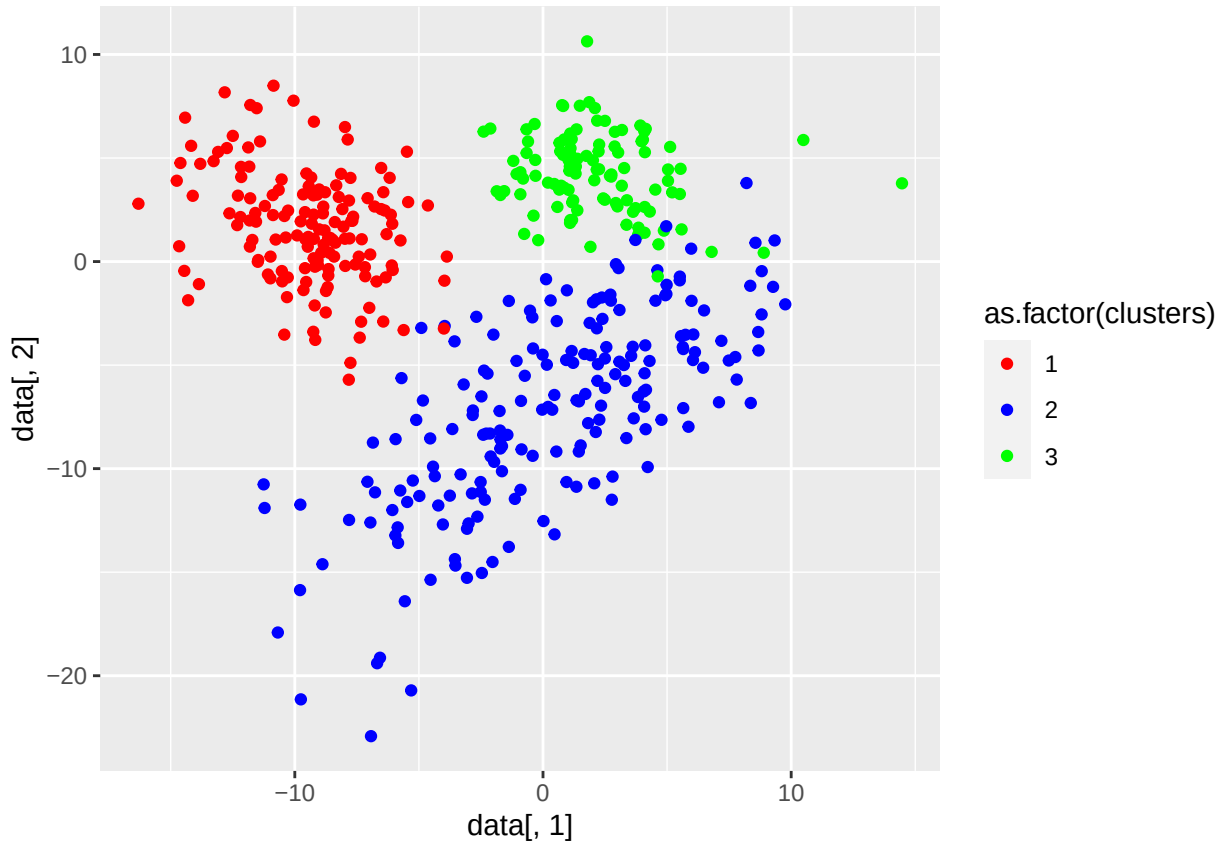

```
## [1] "5"
## [1] 0.5160091
```




```
## [1] "6"
## [1] 0.5165099
```




```
## [1] "7"
## [1] 0.5165099
```

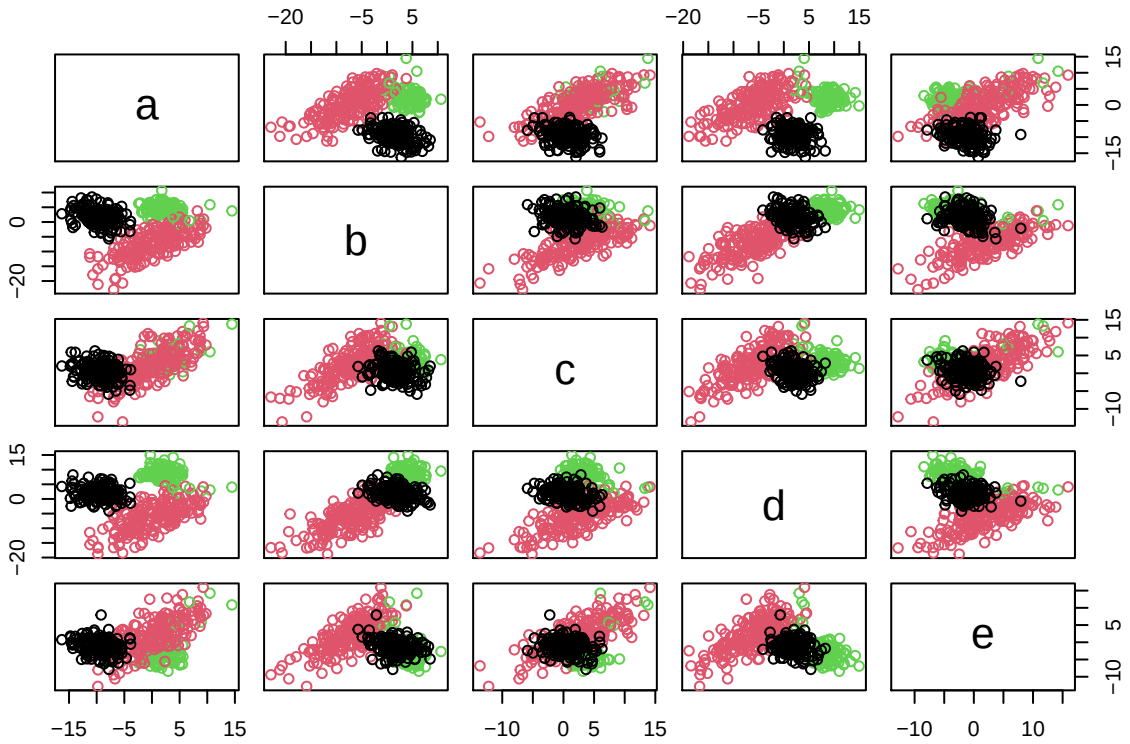



```
clustered_data = output$data
parameters = output$clustered_list

print(parameters) #prints mean and covariance matrix for clusters.
```

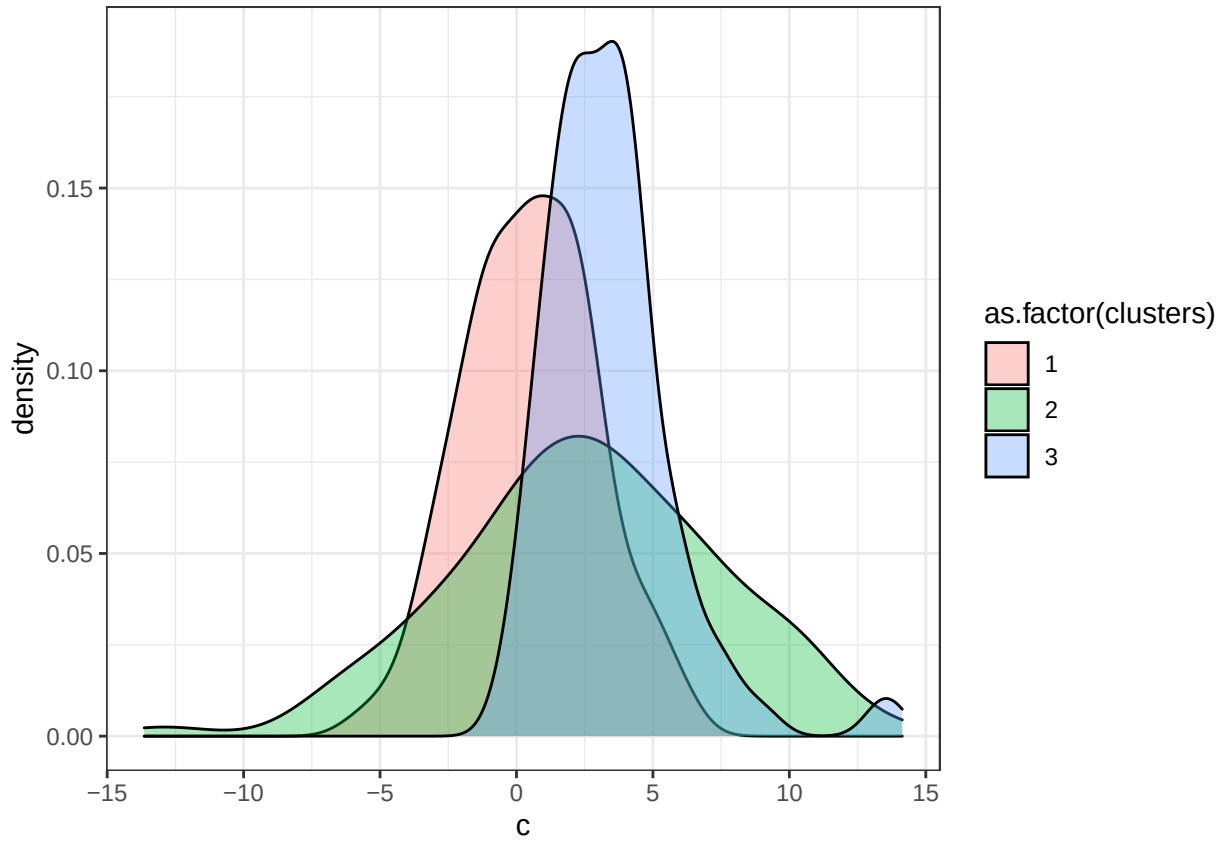
```
## $mean
##            a         b         c         d         e clusters
## 1 -9.4204480  1.672011 0.5089758  1.919512 -1.232519        1
## 2  0.3144581 -7.201651 2.5441233 -6.065219  2.163828        2
## 3  2.1741308  4.229824 3.4080843  8.514269 -2.839595        3
##
## $cov_list
## $cov_list[[1]]
##            a         b          c          d          e
## a  6.0258522 -2.103989 -1.5924905 -0.6766731 -1.2684408
## b -2.1039894  7.354028 -1.4223185 -1.9793705 -1.6959598
## c -1.5924905 -1.422319  5.8006265 -1.5018690 -0.9890357
## d -0.6766731 -1.979370 -1.5018690  5.1324572 -0.8965703
## e -1.2684408 -1.695960 -0.9890357 -0.8965703  5.0772760
##
## $cov_list[[2]]
##          a        b        c        d        e
## a 22.23066 15.78934 17.03207 14.71740 16.34372
## b 15.78934 22.68029 17.36112 15.27510 17.02358
## c 17.03207 17.36112 23.71020 17.10150 19.16054
## d 14.71740 15.27510 17.10150 22.88242 16.59680
```

```
## e 16.34372 17.02358 19.16054 16.59680 24.47796
##
## $cov_list[[3]]
##            a          b         c          d         e
## a  6.4036049 -0.9770626  2.328004 -1.9622532  4.608219
## b -0.9770626  3.7213929 -1.109333  0.3019885 -1.535218
## c  2.3280044 -1.1093327  5.541228 -2.2649431  3.555287
## d -1.9622532  0.3019885 -2.264943  4.9181470 -4.158362
## e  4.6082188 -1.5352180  3.555287 -4.1583615 13.003833
```

```r
#visualization with pair-pair plot.
features = ncol(clustered_data)
with(clustered_data[-(features)],
     pairs(clustered_data[-(features)], col=clustered_data$clusters))
```




```r
#Density comparison of each feature
print(clustered_data %>%
  group_by(clusters) %>%
  ggplot(aes(x = c,  fill = as.factor(clusters))) +
  geom_density( alpha = 0.35) +
  theme_bw())
```


density
0.15
0.10
0.05
0.00
−15
−10
−5
0
5
10
15
c
as.factor(clusters)
1
2
3