Homework 3

Anna Tigranyan

05/06/2022

Problem 1 (30 ptoints)

(a) What are the differences among exclusive, overlapping and fuzzy clustering? Bring (create your own) an example of fuzzy clustering with k = 2. Use the function funny() from library cluster and data visualization techniques from package factoextra to show your results. Show the membership matrix. Which of your observations belongs to both clusters?

(b) Suppose we have an example of a data set with 20 observations. We need to cluster the data using the K-means algorithm. After clustering using k = 1, 2, 3, 4 and 5 we obtained only one non-empty cluster. How is it possible?

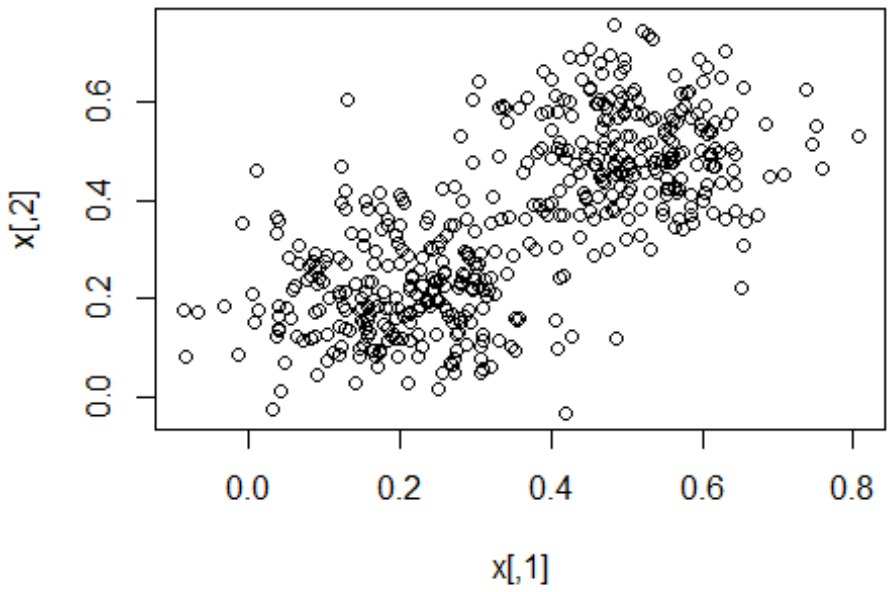
(c) Suppose we have an example of a data set consisting of three natural circular clusters. These clusters have the same number of points and have the same distribution. The centers of clusters lie on a line, the clusters are located such that the center of the middle cluster is equally distant from the other two. Why will not bisecting K-means find the correct cluster?

(a) What are the differences among exclusive, overlapping and fuzzy clustering?Bring (create your own) an example of fuzzy clustering with k = 2. Use the function funny() from library cluster and data visualization techniques from package factoextra to show your results. Show the membership matrix. Which of your observations belongs to both clusters?

exclusive clustering - This separation is based on the characteristic that allows a data object to exist 1 or more than 1 clusters. Exclusive clustering is as the name suggests and stipulates that each data object can only exist in one cluster. **overlapping clustering** - Overlapping allows data objects to be grouped in 2 or more clusters.For example in university overlapping clustering would allow a student to also be grouped as an employee while exclusive clustering would demand that the person must choose the one that is more important. **Fuzzy clustering**-In Fuzzy clustering every data object belongs to every cluster,we can describe fuzzy clustering as an extreme version of overlapping, the major difference is that the data objects has a membership weight that is between 0 to 1 where 0

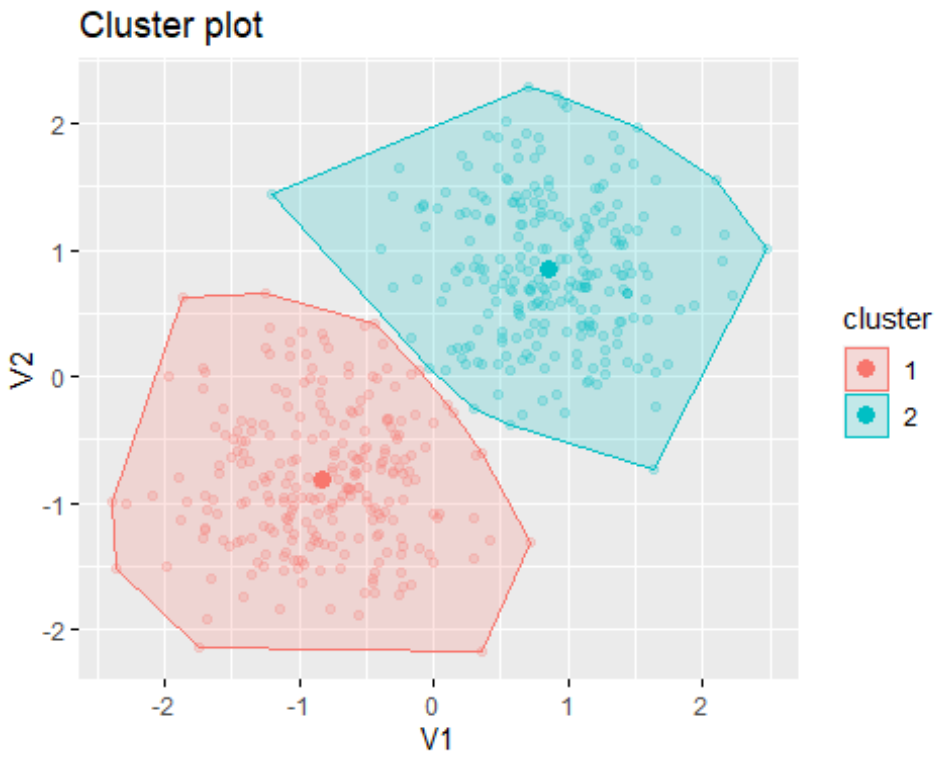
means it does not belong to a given cluster and 1 means it absolutely belongs to the cluster. Fuzzy clustering is also known as probabilistic clustering.

```
library(cluster)
library(factoextra)
library(ggplot2)
library(tidyverse)

x <- rbind(matrix(rnorm(500, mean=0.5, sd=0.1), ncol=2), matrix(rnorm(500,
mean=0.2, sd=0.1), ncol=2))
plot(x)
```




```
f = fanny(x=x, k=2, metric = "euclidean", stand = FALSE)
head(f$membership, 10) # Membership coefficients

##            [,1]      [,2]
##  [1,] 0.8867679 0.1132321
##  [2,] 0.8247024 0.1752976
##  [3,] 0.8826883 0.1173117
##  [4,] 0.8374652 0.1625348
##  [5,] 0.8722620 0.1277380
##  [6,] 0.8932025 0.1067975
##  [7,] 0.8468582 0.1531418
##  [8,] 0.7149018 0.2850982
##  [9,] 0.8870820 0.1129180
## [10,] 0.8211857 0.1788143

x = as.data.frame(x)
```

```r
kms<-kmeans(x,2,nstart=100)

fviz_cluster(kms, data = x, alpha=0.2,shape=19,geom = "point")
```



From plot we can see that we don't have observations belong to both clusters.

(b) Suppose we have an example of a data set with 20 observations. We need to cluster the data using the K-means algorithm. After clustering using k = 1, 2, 3, 4 and 5 we obtained only one non-empty cluster. How is it possible?

Empty clusters can happen when using K-means clustering algorithm, if the random initialization is bad, the number of K is inappropriate, the number of K is more than the number of data points in the data set. The original K-means algorithms is not designed to handle this situation. However,if we find empty clusters while running K-means, it will drop those clusters in the next iteration. So we may end up with fewer final clusters than we initially gave to the algorithm. To avoid this situation, we may want to try different K or improve initialization of the initial cluster centers. In this case if we have 20 observations, I think the points are probably very close to each other, and there are actually 1 cluster.

???So if we choose k values 2,3,4 and 5 K-means algorithm separates only 1 non-empty cluster.

(c) Suppose we have an example of a data set consisting of three natural circular clusters. These clusters have the same number of points and have the same distribution. The centers of clusters lie on a line, the clusters are located such that the center of the middle cluster is equally distant from the other two. Why will not bisecting K-means find the correct cluster?

Bisecting K-means clustering technique is a little modification to the regular K-Means algorithm, wherein you fix the procedure of dividing the data into clusters. So, similar to K-means, we first initialize K centroids. After which we apply regular K-means with K=2 (that's why the word bisecting). We keep repeating this bisection step until the desired number of clusters are reached. After the first bisection (when we have 2 clusters) is done, one can think of multiple strategies for selecting one of the clusters and re-iterating the entire process of bisection and assignment within that cluster - for example: choose cluster with maximum variance or the spread-out cluster, choose cluster with the maximum number of data points, etc. As we mentioned in the first step (bisection) we will divide our points into 2 clusters.If this bisection is 'incorrect' and each of 2 parts contains points from 2 cluster , then in further bisections we will deepen the problem and we will not be able to perform the correct clustering.

Problem 2 (30 points)

Consider the following dataset:

```
var1 <- c(2,2,8,0,7,0,1,7,3,9)
var2 <- c(5,3,3,4,5,7,5,3,7,5)
df <- data.frame(var1,var2)
df
```

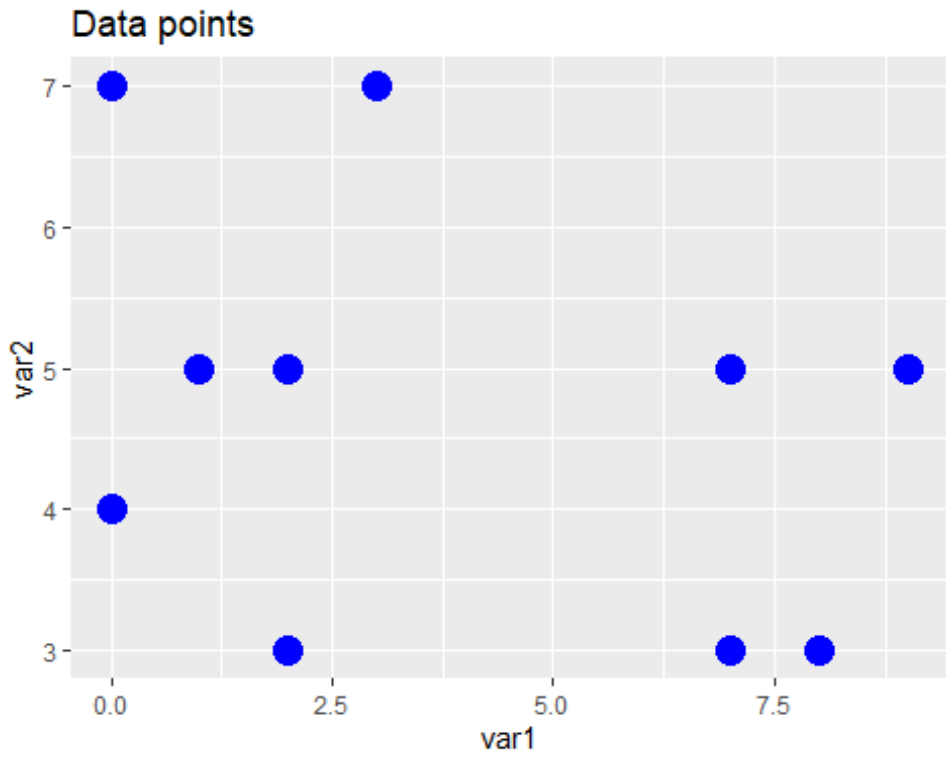
```
##    var1 var2
## 1     2    5
## 2     2    3
## 3     8    3
## 4     0    4
## 5     7    5
## 6     0    7
## 7     1    5
## 8     7    3
## 9     3    7
## 10    9    5
```

The goal of this task is to perform K-means clustering via R (manually), with k = 2, using data with 2 features from the table above. Follow the step above: **(a)** Neatly plot the observations using ggplot. **(b)** Randomly assign a cluster label to each observation. You can use the sample() command in R to do it. Report the cluster labels for each observation. **(c)** Define the coordinates of the centroids for each cluster. Show your results. **(d)** Assign each observation to the cluster using the closeness of each observation to the centroids, in terms of Euclidean distance. Report the cluster labels for each observation. **(e)** Repeat (c) and (d) until the centroids remain the same. You can use loops for this task. **(f)** Show the observations on the plot by coloring them according to the clusters obtained. Show centroids on the plot.

(a) Neatly plot the observations using ggplot.

Let's plot the data points and visualize them using ggplot:

```
ggplot(data=df,aes(x=var1,y=var2))+geom_point(size=5,color='blue')+ggtitle('D
ata points')
```



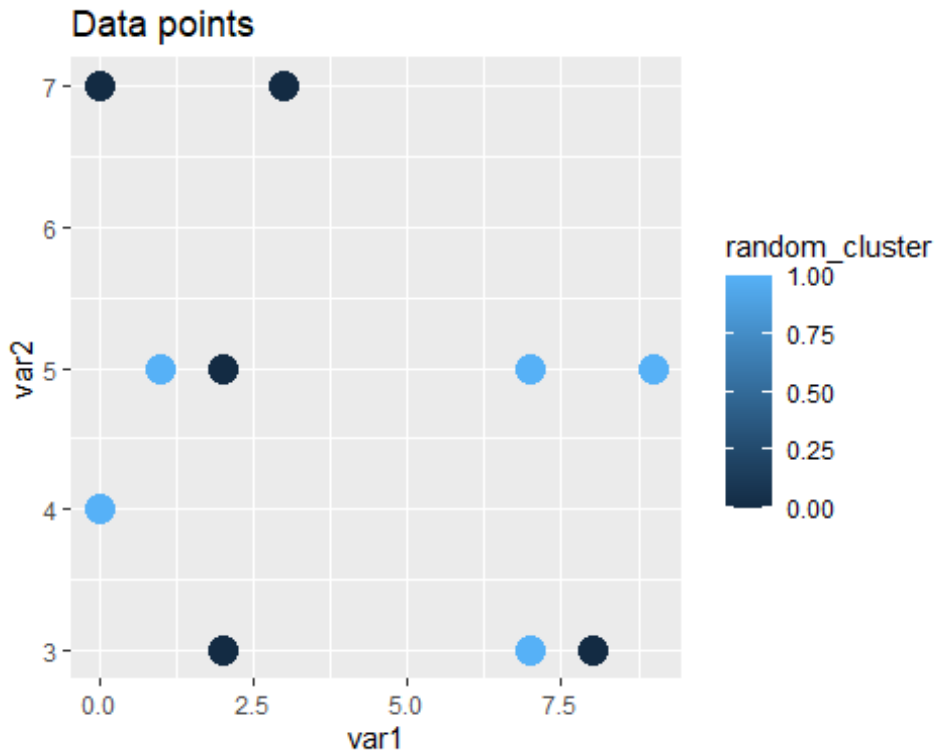
(b) Randomly assign a cluster label to each observation. You can use the sample() command in R to do it. Report the cluster labels for each observation.

```
df['random_cluster'] = sample(c(0,1), size = nrow(df), replace = TRUE)
df
```

```
##    var1 var2 random_cluster
## 1     2    5              0
## 2     2    3              0
## 3     8    3              0
## 4     0    4              1
## 5     7    5              1
## 6     0    7              0
## 7     1    5              1
## 8     7    3              1
## 9     3    7              0
## 10    9    5              1
```

Let's plot and see randomly chosen clusters.

```
ggplot(data=df,aes(x=var1,y=var2,color=random_cluster))+geom_point(size=5)+gg
title('Data points')
```

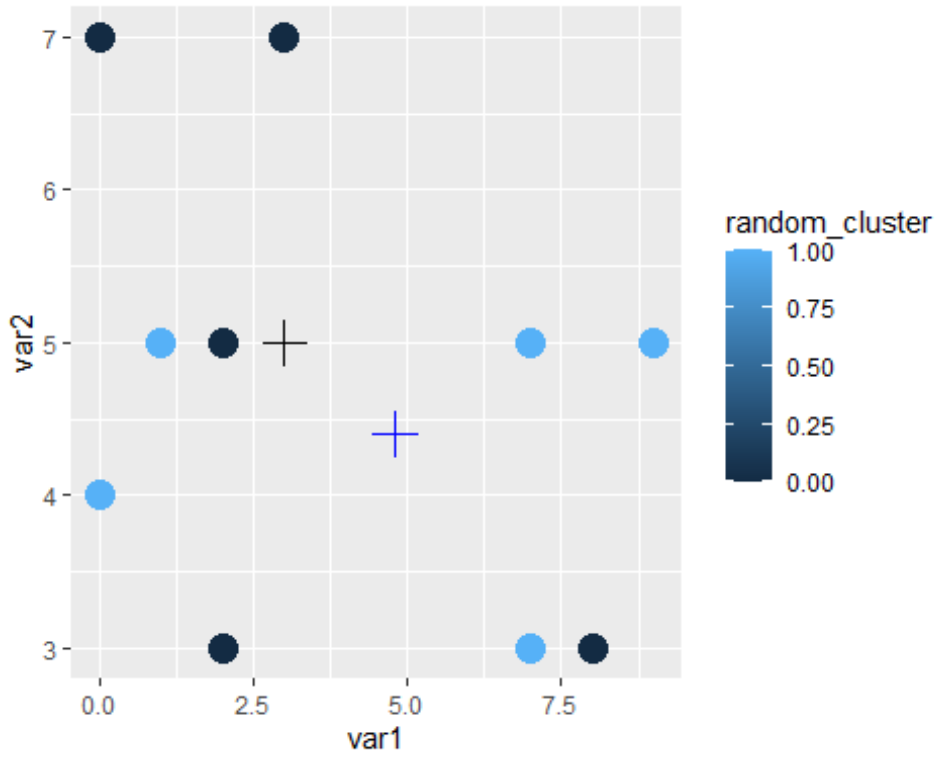


(c) Define the coordinates of the centroids for each cluster. Show your results.

```
# Centroid of 0s
c_0_x <- mean(df$var1[df$random_cluster==0])
c_0_y <- mean(df$var2[df$random_cluster==0])
# Centroid of 1s
c_1_x <- mean(df$var1[df$random_cluster==1])
c_1_y <- mean(df$var2[df$random_cluster==1])
paste('centroid of 0s: (',c_0_x,',',c_0_y,')')

## [1] "centroid of 0s: ( 3 , 5 )"

paste('centroid of 1s: (',c_1_x,',',c_1_y,')')

## [1] "centroid of 1s: ( 4.8 , 4.4 )"
```

Now lets plot our points and centroids.

```
ggplot(data=df,aes(x=var1,y=var2,color=random_cluster)) + geom_point(size=5)
+
      geom_point(aes(x=c_0_x,y=c_0_y),size=5,colour="black",shape=3) +
      geom_point(aes(x=c_1_x,y=c_1_y),size=5,colour="blue",shape=3)
```



(d)Assign each observation to the cluster using the closeness of each observation to the centroids, in terms of Euclidean distance. Report the cluster labels for each observation

There are three steps in computing the Euclidean distance:

- Compute the difference between the corresponding X and Y coordinates of the data-points and the centroid.
- Compute the sum of the square of the differences computed in Step 1.
- Find the square root of the sum of squares of differences computed in Step 2.

Let's do it for each centroid:

```
distance_from_cluster_0 = (df[,c(1,2)] - c(c_0_x,c_0_y))^2
distance_from_cluster_0 = sqrt(distance_from_cluster_0[,1] +
distance_from_cluster_0[,2])
distance_from_cluster_0
```

```
##  [1] 2.236068 3.605551 5.000000 5.099020 4.472136 5.385165 2.828427
2.828427
##  [9] 4.000000 4.000000
```

```
distance_from_cluster_1 = (df[,c(1,2)] - c(c_1_x,c_1_y))^2
distance_from_cluster_1 = sqrt(distance_from_cluster_1[,1] +
distance_from_cluster_1[,2])
distance_from_cluster_1
```

```
##  [1] 2.807134 2.778489 3.671512 4.418144 2.209072 5.110773 3.805260
2.952965
##  [9] 2.842534 4.638965
```

Combining distance_from_cluster_0 and distance_from_cluster_1, we get the total distance array called total_distance. This array contains the distances between points and the centroids. We use this array to determine which cluster a given point belongs to.

```
total_distance = array(c(distance_from_cluster_0, distance_from_cluster_1),
dim = c(10, 2))
total_distance
```

```
##           [,1]     [,2]
##  [1,] 2.236068 2.807134
##  [2,] 3.605551 2.778489
##  [3,] 5.000000 3.671512
##  [4,] 5.099020 4.418144
##  [5,] 4.472136 2.209072
##  [6,] 5.385165 5.110773
##  [7,] 2.828427 3.805260
##  [8,] 2.828427 2.952965
##  [9,] 4.000000 2.842534
## [10,] 4.000000 4.638965
```

Let's create a logical vector comparing distance_from_cluster_0 and distance_from_cluster_1. This vector will be comprised of the Boolean values TRUE and FALSE.

The condition would be as follows: distance to the first cluster is less than the second cluster's distance. Points that satisfy this condition belong to cluster 0. The remaining points belong to cluster 1.

```
library(dplyr)
df = df %>% mutate(new_cluster = case_when(c(total_distance[,1] <=
total_distance[,2])==TRUE ~ 0,
                           c(total_distance[,1] <=
total_distance[,2])==FALSE ~ 1))
df
```

```
##    var1 var2 random_cluster new_cluster
## 1     2    5              0           0
## 2     2    3              0           1
## 3     8    3              0           1
## 4     0    4              1           1
## 5     7    5              1           1
## 6     0    7              0           1
## 7     1    5              1           0
## 8     7    3              1           0
## 9     3    7              0           1
## 10    9    5              1           0
```

```
c(total_distance[,1] <= total_distance[,2])
```

```
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE
```

Elements that satisfy this condition in the df are printed bilow:

```
df[,1][c(total_distance[,1] <= total_distance[,2])]
```

```
## [1] 2 1 7 9
```

To find the centroid of the newly formed cluster, we take the mean of all the points obtained above.

```
mean(df[,1][c(total_distance[,1] <= total_distance[,2])])
```

```
## [1] 4.75
```

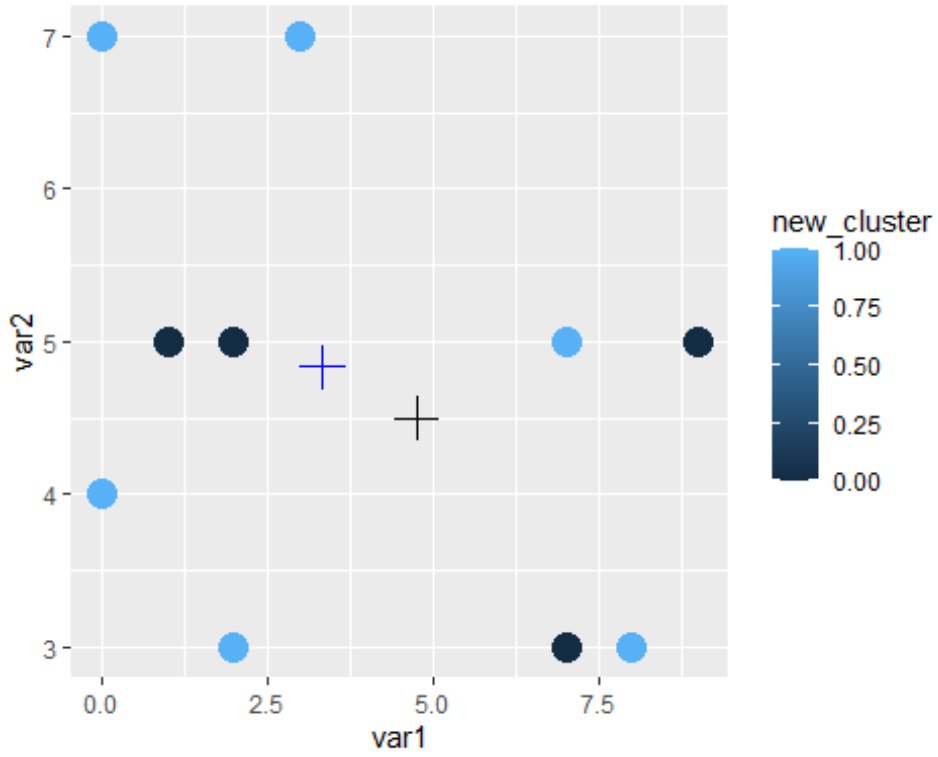
We compute the X and Y coordinates of the centroid using the code above. We store the X coordinate in c0 and y-coordinates in c1. We copy the data in these lists to a new array called new_centroid.

```
c0 = c(mean(df[,1][c(total_distance[,1] <= total_distance[,2])]),
mean(df[,2][c(total_distance[,1] <= total_distance[,2])]))

c1 = c(mean(df[,1][!c(total_distance[,1] <= total_distance[,2])]),
```

```
mean(df[,2][!c(total_distance[,1] <=total_distance[,2])]))
new_centroid = matrix( nrow = 2, ncol = 2)
new_centroid[1,] = c0
new_centroid[2,] = c1
new_centroid

##          [,1]     [,2]
## [1,] 4.750000 4.500000
## [2,] 3.333333 4.833333
```

The new_centroid contains the updated centroid of the formed clusters. Let's plot the new centroids :

```
ggplot(data=df,aes(x=var1,y=var2,color=new_cluster)) + geom_point(size=5) +

geom_point(aes(x=new_centroid[1,1],y=new_centroid[1,2]),size=5,shape=3,colour
="black") +

geom_point(aes(x=new_centroid[2,1],y=new_centroid[2,2]),size=5,shape=3,colour
="blue")
```



(e) Repeat (c) and (d) until the centroids remain the same. You can use loops for this task.

```
for (i in 1:5){
  distance_from_cluster_0 = (df[,c(1,2)] - new_centroid[1,])^2
  distance_from_cluster_0 = sqrt(distance_from_cluster_0[,1] +
```

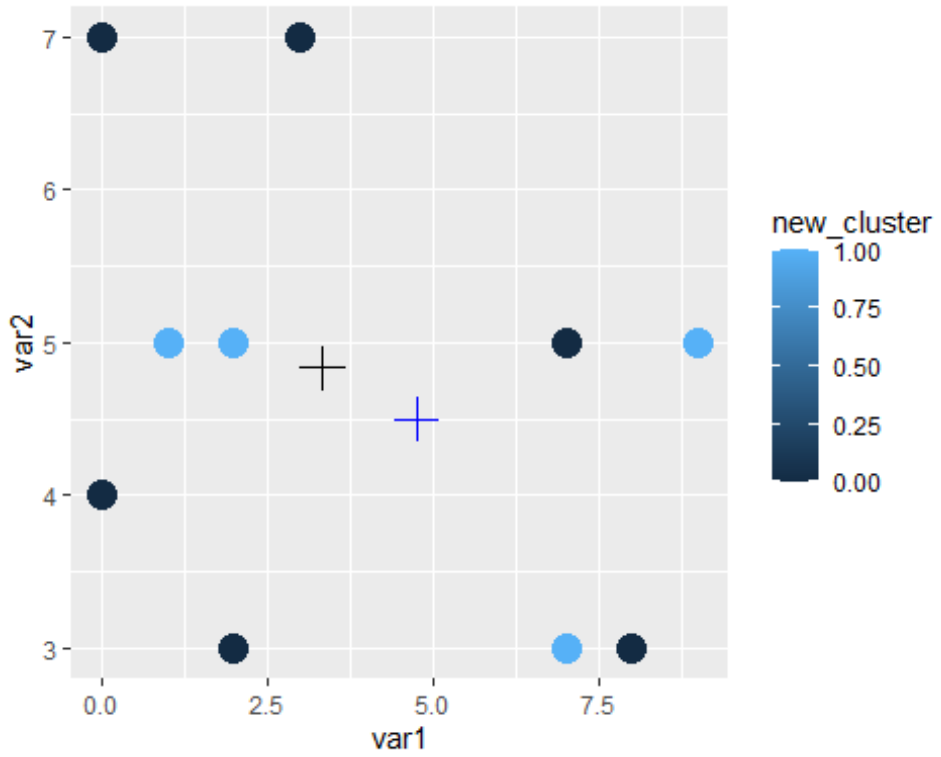
```
distance_from_cluster_0[,2])

  distance_from_cluster_1 = (df[,c(1,2)] - new_centroid[2,])^2
  distance_from_cluster_1 = sqrt(distance_from_cluster_1[,1] +
distance_from_cluster_1[,2])

  total_distance = array(c(distance_from_cluster_0, distance_from_cluster_1),
dim = c(10, 2))

  df$new_cluster = case_when(c(total_distance[,1] <=
total_distance[,2])==TRUE ~ 0,c(total_distance[,1] <=
total_distance[,2])==FALSE ~ 1)

  c0 = c(mean(df[,1][c(total_distance[,1] <= total_distance[,2])]),
mean(df[,2][c(total_distance[,1] <= total_distance[,2])]))

  c1 = c(mean(df[,1][!c(total_distance[,1] <= total_distance[,2])]),
mean(df[,2][!c(total_distance[,1] <=total_distance[,2])]))

  new_centroid[1,] = c0
  new_centroid[2,] = c1

  print(new_centroid)

}
```

```
##          [,1]     [,2]
## [1,] 3.333333 4.833333
## [2,] 4.750000 4.500000
##          [,1]     [,2]
## [1,] 4.750000 4.500000
## [2,] 3.333333 4.833333
##          [,1]     [,2]
## [1,] 3.333333 4.833333
## [2,] 4.750000 4.500000
##          [,1]     [,2]
## [1,] 4.750000 4.500000
## [2,] 3.333333 4.833333
##          [,1]     [,2]
## [1,] 3.333333 4.833333
## [2,] 4.750000 4.500000
```

(f)Show the observations on the plot by coloring them according to the clusters obtained. Show centroids on the plot.

```
ggplot(data=df,aes(x=var1,y=var2,color=new_cluster)) + geom_point(size=5) +

geom_point(aes(x=new_centroid[1,1],y=new_centroid[1,2]),size=5,shape=3,colour
="black") +
```

```
geom_point(aes(x=new_centroid[2,1],y=new_centroid[2,2]),size=5,shape=3,colour
="blue")
```



Problem 3 (40 points)

For this task you need to download World Value Survey (Wave 6) data and try to understand the disposition of our country among others based on some criterias. The description of the variables and the survey are given with a separate file. Here is the link to obtain more information: . Choose the subset1 fromWave 6 data to perform the cluster analysis. Note that you need to use meaningful selections both of variables based on some topic/problem2 and countries3. **(a)** Describe thoroughly how and why you choose your subset of variables and observations. What is your goal? Hint: You need to prepare data for the next step. **(b)** Use all (appropriate) tools/functions from our lecture to cluster the countries (both nested and untested algorithms). Interpret them. **(b1)** Is your hierarchical clustering stable regards to between clusters distance measures? **(b2)** Compare the results obtained from two different k-means. **(c)** Make the conclusion (also based on cluster centers).

```
library(readr)
dat <- read_csv('WaveData.csv')

head(dat)
```

```
## # A tibble: 6 x 443
##    ...1    V1    V2   V2A   cow C_COW_ALPHA B_COUNTRY_ALPHA    V3    V4
V5
##   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>       <chr>            <dbl> <dbl>
<dbl>
## 1     1     6    12    12   615 ALG         DZA                  1     1
1
## 2     2     6    12    12   615 ALG         DZA                  2     1
2
## 3     3     6    12    12   615 ALG         DZA                  3     1
3
## 4     4     6    12    12   615 ALG         DZA                  4     1
1
## 5     5     6    12    12   615 ALG         DZA                  5     1
1
## 6     6     6    12    12   615 ALG         DZA                  6     1
2
## # ... with 433 more variables: V6 <dbl>, V7 <dbl>, V8 <dbl>, V9 <dbl>,
## #   V10 <dbl>, V11 <dbl>, V12 <dbl>, V13 <dbl>, V14 <dbl>, V15 <dbl>,
## #   V16 <dbl>, V17 <dbl>, V18 <dbl>, V19 <dbl>, V20 <dbl>, V21 <dbl>,
## #   V22 <dbl>, V23 <dbl>, V24 <dbl>, V25 <dbl>, V26 <dbl>, V27 <dbl>,
## #   V28 <dbl>, V29 <dbl>, V30 <dbl>, V31 <dbl>, V32 <dbl>, V33 <dbl>,
## #   V34 <dbl>, V35 <dbl>, V36 <dbl>, V37 <dbl>, V38 <dbl>, V39 <dbl>,
## #   V40 <dbl>, V41 <dbl>, V42 <dbl>, V43 <dbl>, V44 <dbl>, V44_ES <dbl>,
...
```

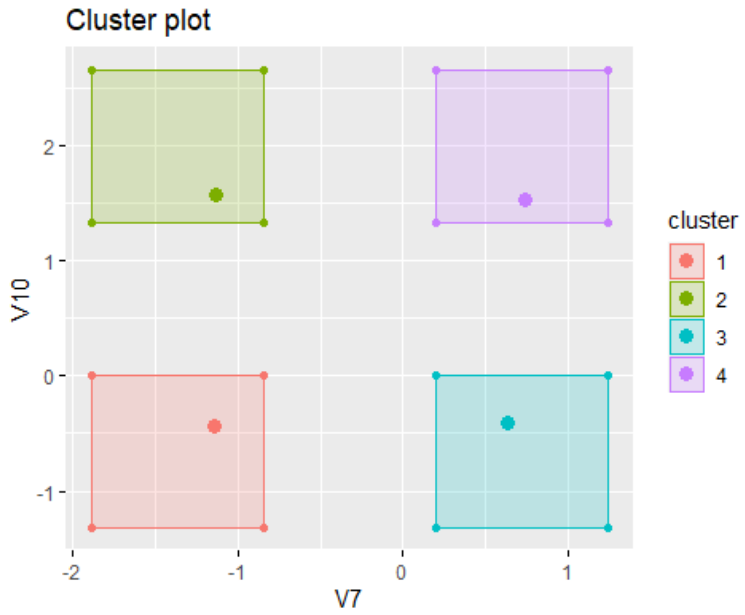
Variables: I have selected the following variables from the given data set?? - V2 - country code number - V7 -Politics(how important it is in your life.1-4(1-very important,4-Not at all important)) - V10 - Taking all things together, would you say you are (read out and code one answer): 1 Very happy 2 Rather happy 3 Not very happy 4 Not at all happy **Countries**I have taken Armenia and neighboring countries . **Problem:** How does the importance of politics affect happiness for different countries?

```
dat1 <- subset(dat, select=c(V2,V7,V10) )
head(dat1)
```

```
## # A tibble: 6 x 3
##      V2    V7   V10
##   <dbl> <dbl> <dbl>
## 1    12    NA     2
## 2    12     4     2
## 3    12     4     2
## 4    12     4     2
## 5    12     2     1
## 6    12     2     2
```

```
# 51 Armenia
# 31 Azerbaijan
# 268 Georgia
# 364 Iran
```

```
# 792 Turkey
# 643 Russia

# Choosing rows containing above countries
dat2 =
dat1[dat1$V2==51|dat1$V2==31|dat1$V2==268|dat1$V2==364|dat1$V2==792|dat1$V2==
643,]


# changing data types of our variables to categorical (all variables are
categorical)
# dat2$V2 = as.factor(dat2$V2)
# dat2$V7 = factor(dat2$V7,order=TRUE)
# dat2$V10 =factor(dat2$V10,order=TRUE)

# cleaning NA values(dropping rows containing NA values)
dat2 <- dat2 %>% drop_na()

set.seed(123) # Setting seed
kms <- kmeans(dat2[,c(2,3)], centers = 4, nstart = 100)
kms[2]

## $centers
##          V7      V10
## 1 1.713115 1.662970
## 2 1.718861 3.179715
## 3 3.417736 1.687117
## 4 3.518782 3.147208

fviz_cluster(kms, data = dat2[,c(2,3)], alpha=0.2,shape=19,geom = "point")
```



From above we can see that we have 4 clusters for our countries.