Exercise 2 Trashaj Alberto 1075402

alberto.trashaj

October 2023

1 Point 1 a

In the first point the request was to use the gap statistic method to estimate the number of clusters for the olive oil data, and for Artificial Data Set 2.

Here we import the usual libraries needed.

```
library(cluster)
library(ggplot2)
library(sn)
library(mvtnorm)
```

Now we import the oliveoil dataset provided in Virtuale, check the structure and print the summary.

```
oliveoil <- read.csv("~/Desktop/Universita /Unsupervised/
------------oliveoil.dat", sep="")
str(oliveoil)
summary(oliveoil)
attach(oliveoil)
```

No particular comment needed for this dataset that we already discussed in class.
There are some categorical variable: let's take the subset of the non-categorical variable and standardize it.

```
data_olive <- oliveoil[,
                       c("palmitic", "palmitoleic", "stearic", "oleic", "linoleic",
stand_olive <- scale(data_olive)
head(stand_olive)
```

To perform the Gap statistic method we import the function provided in Virtuale called "gapnc":

```
gapnc <- function(data, FUNcluster=kmeans,
                  K.max=10, B = 100, d.power = 2,
                  spaceH0 ="scaledPCA",
                  method ="globalSEmax", SE.factor = 2,...){
# As in original clusGap function the ... arguments are passed on
```

```
  # to the clustering method FUNcluster (kmeans).
  # Run clusGap
  gap1 <- clusGap(data,kmeans,K.max, B, d.power,spaceH0,...)

# Values of log(S_k) and its expectation under uniform distribution
# The latter are in cg1$Tab[,2].
  # Find optimal number of clusters; note that the method for
  # finding the optimum and the SE.factor q need to
  # be specified here.
  nc <- maxSE(gap1$Tab[,3],gap1$Tab[,4],method, SE.factor)
  # Re-run kmeans with optimal nc.
  kmopt <- kmeans(data,nc, nstart = 100)
  out <- list()
  out$gapout <- gap1
  out$nc <- nc
  out$kmopt <- kmopt
  out
}
```

We are ready now to apply the function to the standardized dataset:

```
set.seed(123456)

cg_olive <- gapnc(stand_olive)

cg_olive
```

The output provided from the script tell us that the optimal number of clusters is 8: the function therefore computes the k-means with the optimal number of clusters of sizes 154, 59, 98, 56, 52, 79, 36, 38.
The $(\frac{between_{SS}}{total_{SS}}) = 0.773$

Now we perform the same analysis for the Artifical data set 2 provided also in Virtuale: let's import the dataset.

```
artificial <- read.csv("/Users/albertotrashaj/Desktop/
----------------Universita /Unsupervised/artificial2.dat", sep="-")
str(artificial)
summary(artificial)
attach(artificial)
```

Now we can apply the gapnc function:

```
set.seed(123456)

artificial <- scale(artificial)

cg_artificial2 <- gapnc(artificial)
```

```
cg_artificial2
```

In this case, the output suggest us that the optimal number of clusters in this dataset will be 10. Then , by computing the k-means algorithm with $K = 10$, the $(\frac{between_{SS}}{total_{SS}})$ is 0.969, much higher than the k-Means on the olive dataset.

1.1 Point 1 b

In the second point the request was to generate a specific dataset based on the artifical data set 2:
since the name of the variables of the original dataset were not so meaningful we changed them

```
names(artificial)[1] <- "x1"
names(artificial)[2] <- "x2"

min_x1 <- min(artificial$x1)
max_x1 <- max(artificial$x1)
min_x2 <- min(artificial$x2)
max_x2 <- max(artificial$x2)
```

So, after we defined the minimum and maximum of the variables of the dataset, we can generate a new dataset from a two-dimensional uniform distribution

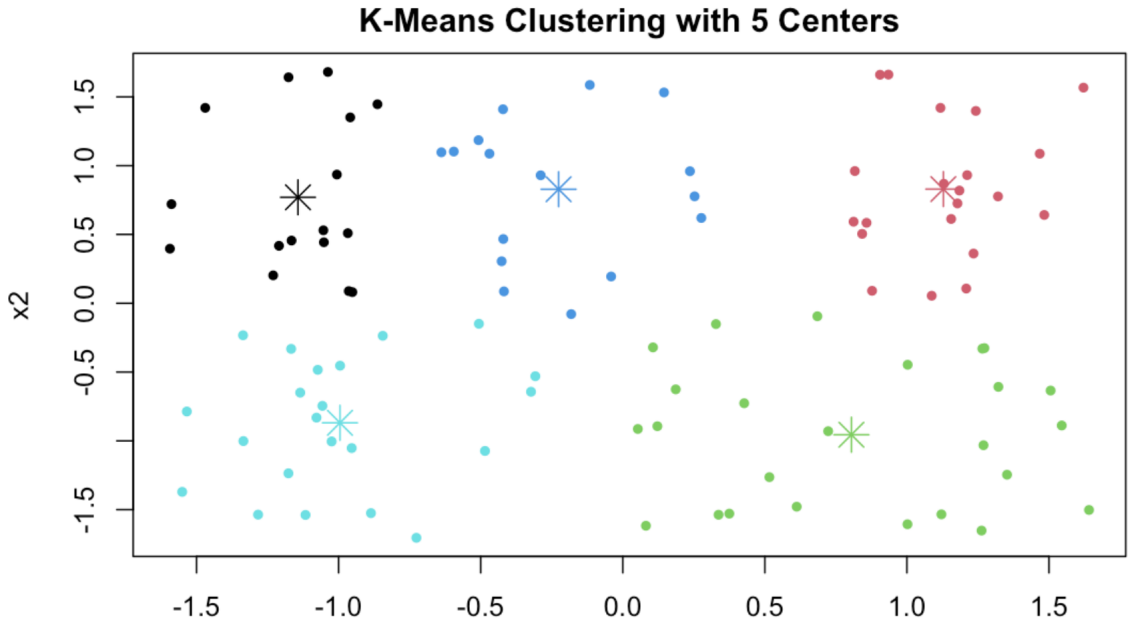
```
n <- 100

set.seed(123456)
# Generate random values for x1 and x2
x1 <- runif(n, min_x1, max_x1)
x2 <- runif(n, min_x2, max_x2)

# Create a data frame with x1 and x2
generated_data <- data.frame(x1 = x1, x2 = x2)
plot(generated_data)
```

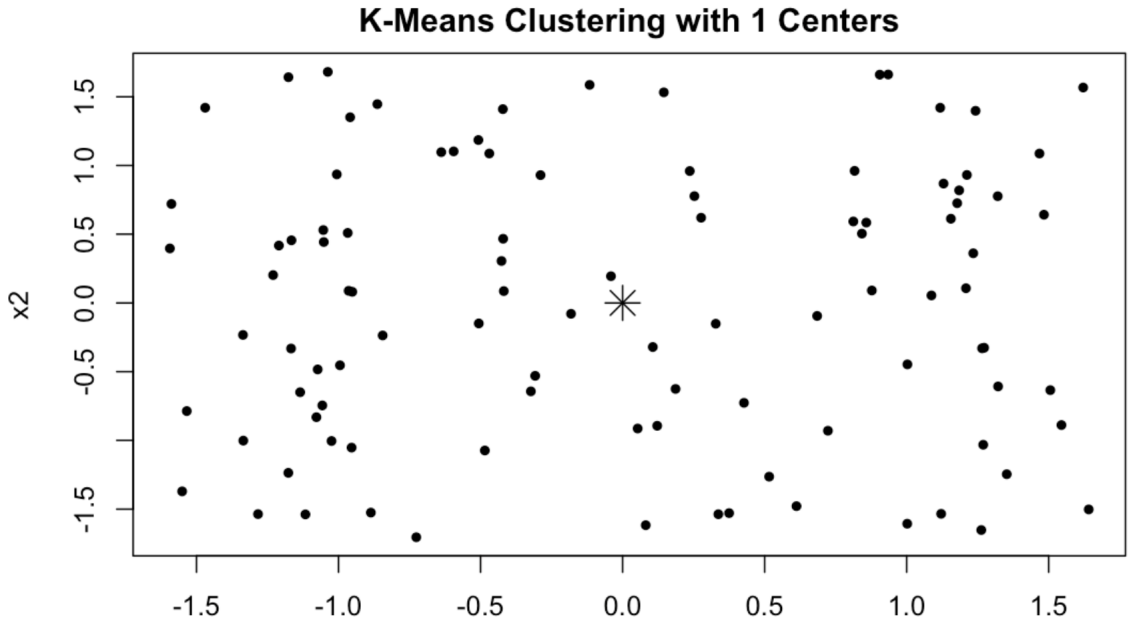
As we can see from the plot the data are well spread into the rectangle defined from the min and max of x_1 and x_2.

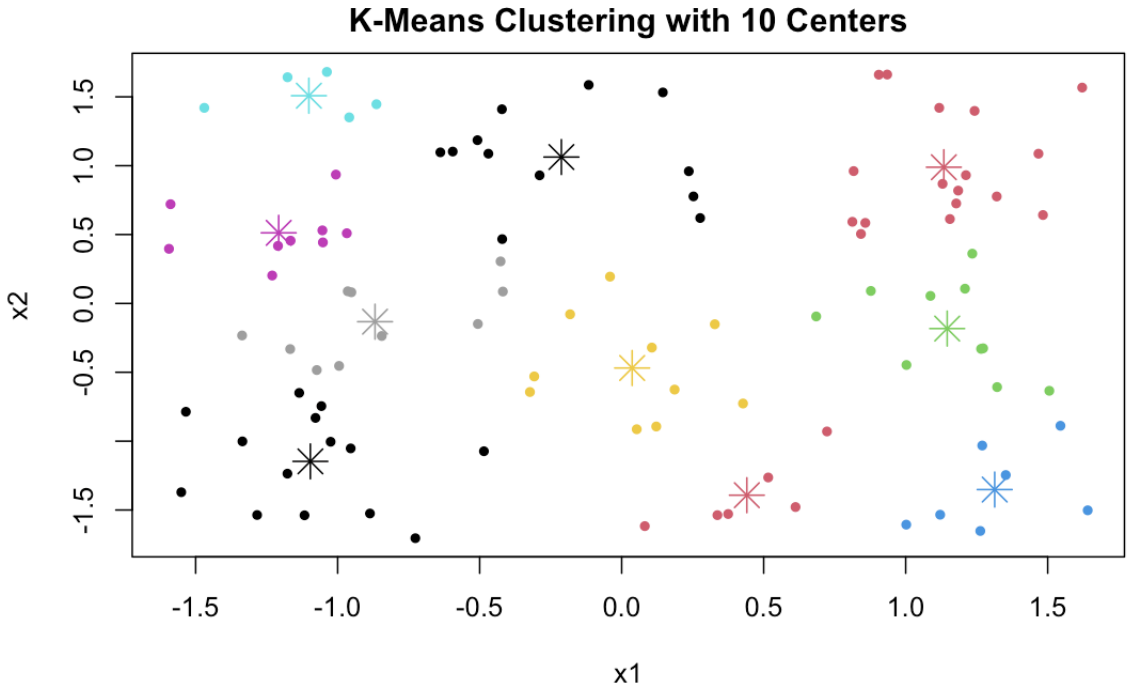
Now we perform a k-Means clustering with K from 1 to 10 and we produce a scatter plot for every K in the iteration process.

```
  for (k in 1:10){
k_means <- kmeans(generated_data, centers = k)
clusters <- k_means$cluster

plot(generated_data, col=clusters, pch=20,
       main=paste("K-Means Clustering with", k, "Centers"))
points(k_means$centers, col=1:k, pch=8, cex=2)
```

}


K-Means Clustering with 5 Centers
x2
1.5
1.0
0.5
0.0
-0.5
-1.5
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5


K-Means Clustering with 1 Centers
x2
1.5
1.0
0.5
0.0
-0.5
-1.5
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5



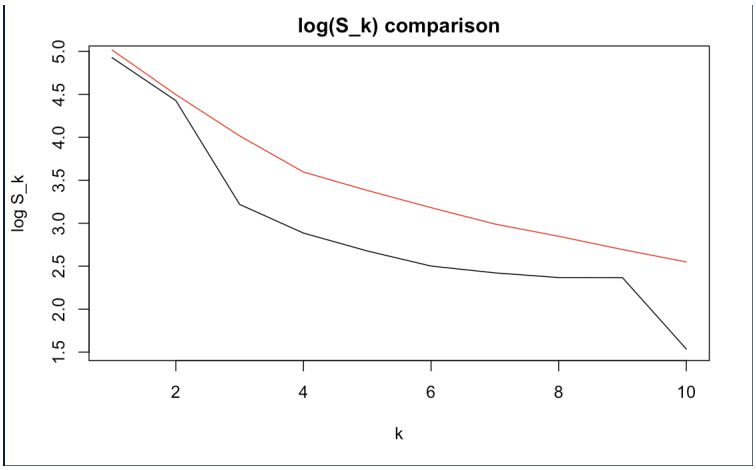
Now we compare the values of $logS_k$ from clustering Artificial Data Set 2 in the point (a) with those from clustering the uniformly distributed dataset: to do so, we apply the function gapnc to the generated data set.

```
set.seed(1234)

generated_data <- scale(generated_data)
cg_generated <- gapnc(generated_data)

cg_generated
```

As we would expect, the optimal number of clusters suggested by the output is 1, since we generated the data from a uniform distribution.
Now we compare the values in a plot

```
plot(1:10, cg_artificial2$gapout$Tab[,1], xlab="k",
     ylab="log S_k",type="l", main = "log(S_k) comparison")
lines(1:10, cg_generated$gapout$Tab[,2], col="red")
legend(legend(6,10, legend=
          c("Artificial data set 2","Generated artificial dataset"),
     col = c("black","red"), lwd = 1))
```

The red line shows the values of $logS_k$ of the generated data, the black line instead the values from the artifical data sets.



2 Point 2

In the secont point we compare various option of the clusGap function: let's breakdown the work.

We import the library needed to generate the Artificial data set 1 as shown in Virtuale

```
library(sn)
```

Initialize four empty vectors where we will store the number of optimal clusters

```
n_clusters_1 <- numeric()
n_clusters_2 <- numeric()
n_clusters_3 <- numeric()
n_clusters_4 <- numeric()
```

And now we can apply the gapnc function with the different combinations of "spaceH0" and "SE.factor"

```
for (i in 1:100){

  v1 <- c(rnorm(50,0,1), rsn(70,5,1,8), rnorm(30,6,1))
  v2 <- c(rnorm(50,0,1), rsn(70,0,1,8), 8+rt(30,5))
  clusterdata <- cbind(v1,v2)
  result_1 <- gapnc(clusterdata, nstart=3,
                    spaceH0 = "scaledPCA", SE.factor = 1)
  result_2 <- gapnc(clusterdata, nstart=3,
                    spaceH0 = "scaledPCA", SE.factor = 2)
  result_3 <- gapnc(clusterdata, nstart=3,
                    spaceH0 = "original", SE.factor = 1)
```

```
  result_4 <- gapnc(clusterdata , nstart=3,
                    spaceH0 = "original", SE.factor = 2)

  n_clusters_1 <- append(n_clusters_1, result_1$nc)
  n_clusters_2 <- append(n_clusters_2, result_2$nc)
  n_clusters_3 <- append(n_clusters_3, result_3$nc)
  n_clusters_4 <- append(n_clusters_4, result_4$nc)
}
```

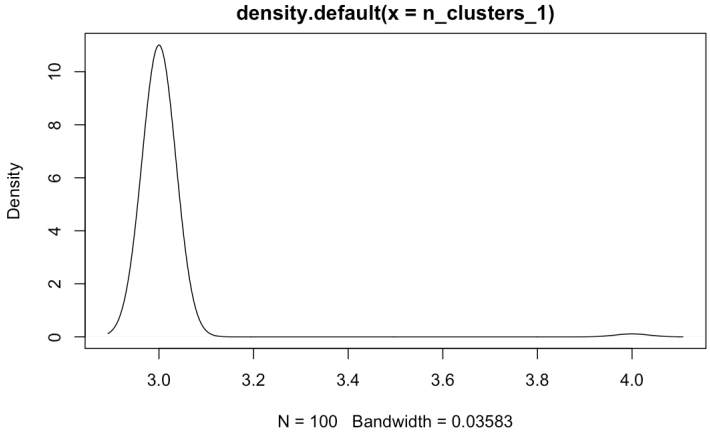
To better visualize the output of the different functions, we plot the density of the vectors

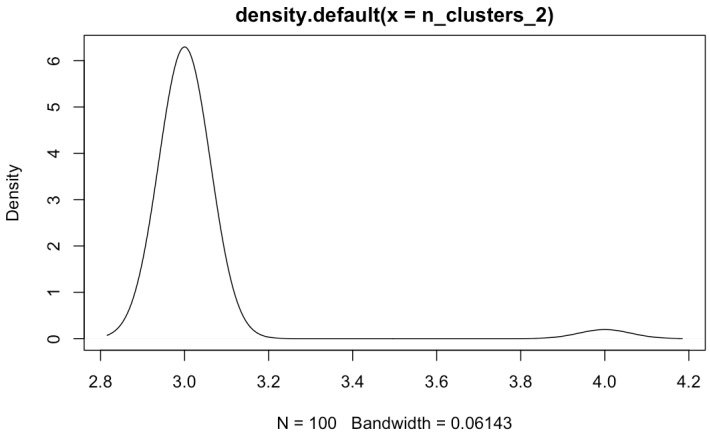
```
n_clusters_1_density <- density(n_clusters_1)
plot(n_clusters_1_density)

n_clusters_2_density <- density(n_clusters_2)
plot(n_clusters_2_density)

n_clusters_3_density <- density(n_clusters_3)
plot(n_clusters_3_density)

n_clusters_4_density <- density(n_clusters_4)
plot(n_clusters_4_density)
```

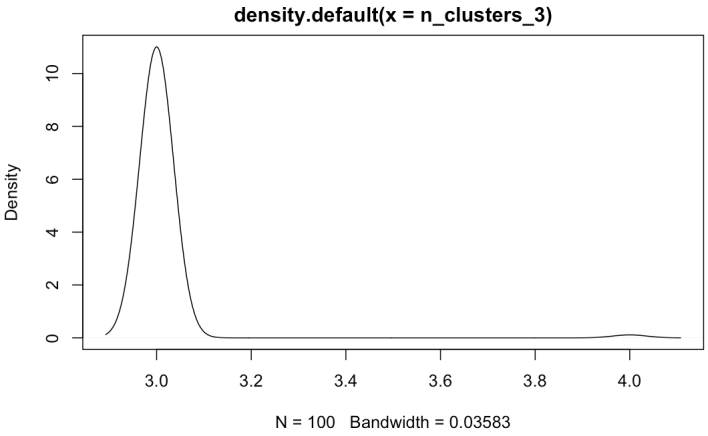
As we can see from the plots, almost all the methods suggest that the optimal number of clusters is 3.

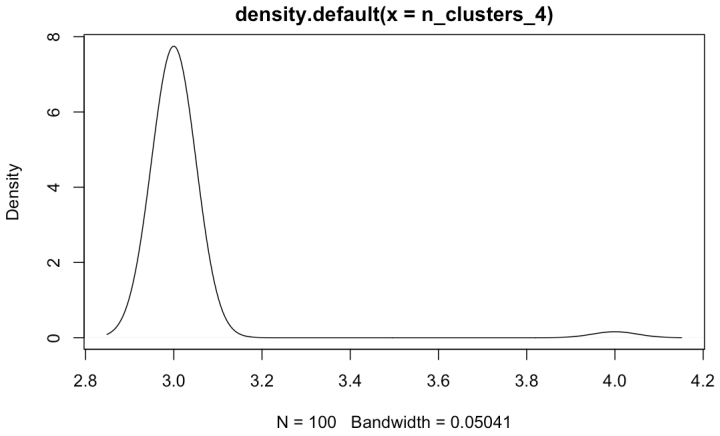
This result is expected since a visual inspection of the data generated suggest us that there are 3 clusters

```
v1 <- c(rnorm(50,0,1), rsn(70,5,1,8), rnorm(30,6,1))
v2 <- c(rnorm(50,0,1), rsn(70,0,1,8), 8+rt(30,5))
clusterdata <- cbind(v1,v2)
plot(clusterdata)
```


density.default(x = n_clusters_1)
Density
0
2
4
6
8
10
3.0
3.2
3.4
3.6
3.8
4.0
N = 100 Bandwidth = 0.03583


density.default(x = n_clusters_2)
Density
0
1
2
3
4
5
6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
4.2
N = 100 Bandwidth = 0.06143


density.default(x = n_clusters_3)
Density
0
2
4
6
8
10
3.0
3.2
3.4
3.6
3.8
4.0
N = 100 Bandwidth = 0.03583



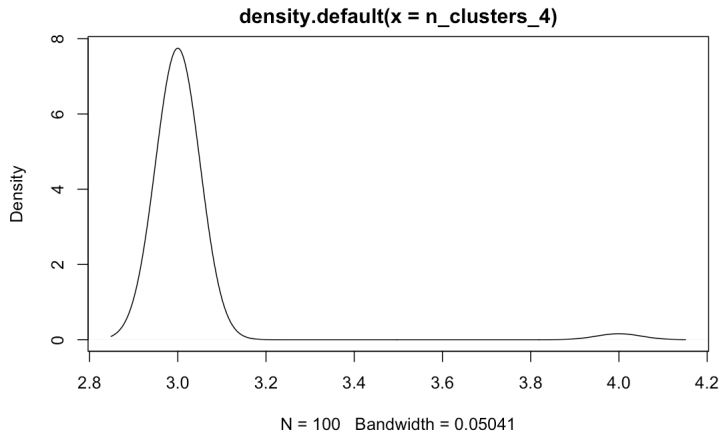
Now we do the same thing with another model generating data: a multivariate normal distribution.

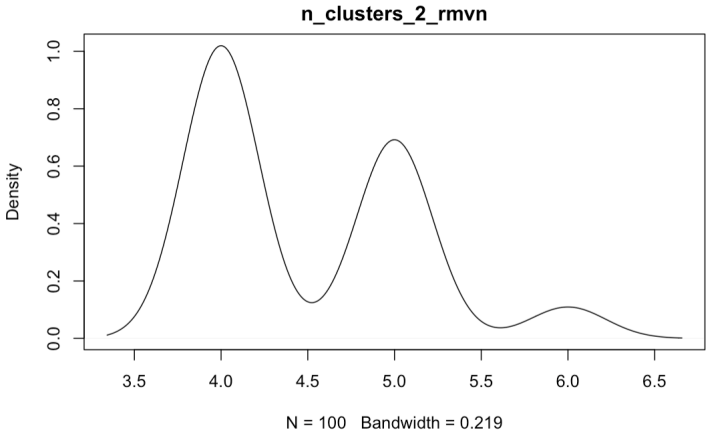
```
n_clusters_1_rmvn <- numeric()
n_clusters_2_rmvn <- numeric()
n_clusters_3_rmvn <- numeric()
n_clusters_4_rmvn <- numeric()

for (i in 1:100){

  n_obs <- c(50, 100, 150, 200) #number of observations for each
                                #multivariate normal distribution

  means <- matrix(c(1, 2, 3, 4, 150, 250, 350, 450, 800, 1800,
                    2800, 3800, 12000, 22000, 32000, 42000),
                    ncol = 4, byrow = TRUE)


  cov_matrices <- matrix(c(1, 0, 0, 0, 0, 1, 0, 0,
                           0, 0, 1, 0, 0, 0, 0, 1),
                           ncol = 4)
  data <- list()
  dataset <- list()

  for (i in 1:4){

    dataset[[i]] <- rmvnorm(n = n_obs[i],
                    mean = means[i,], sigma = cov_matrices)
    data[[i]] <- data.frame(dataset[[i]])
  }
```

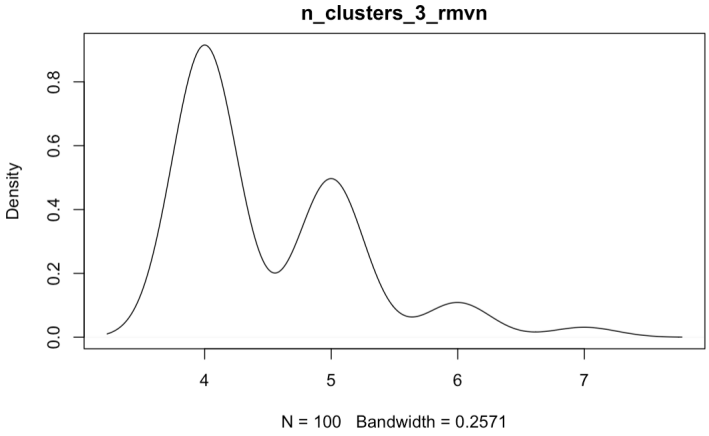
```
  combined_data <- do.call(rbind, data)

  result_1_rmvn <- gapnc(combined_data, nstart=3,
                         spaceH0 = "scaledPCA", SE.factor = 1)
  result_2_rmvn <- gapnc(combined_data, nstart=3,
                         spaceH0 = "scaledPCA", SE.factor = 2)
  result_3_rmvn <- gapnc(combined_data, nstart=3,
                         spaceH0 = "original", SE.factor = 1)
  result_4_rmvn <- gapnc(combined_data, nstart=3,
                         spaceH0 = "original", SE.factor = 2)

  n_clusters_1_rmvn <- append(n_clusters_1_rmv, result_1_rmvn$nc)
  n_clusters_2_rmvn <- append(n_clusters_2_rmv, result_2_rmvn$nc)
  n_clusters_3_rmvn <- append(n_clusters_3_rmv, result_3_rmvn$nc)
  n_clusters_4_rmvn <- append(n_clusters_4_rmv, result_4_rmvn$nc)
}
```

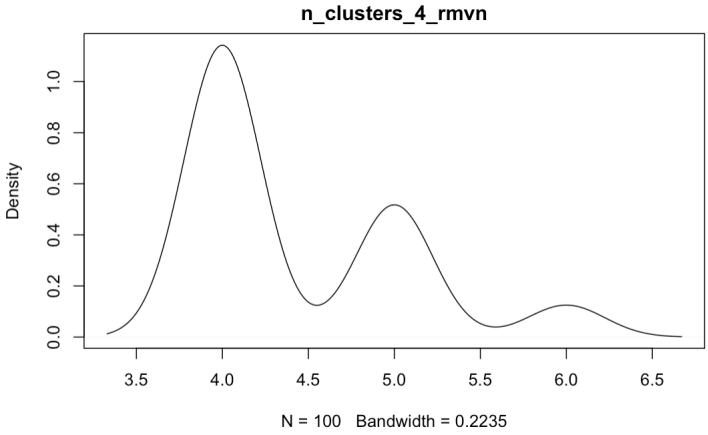
As before, we can check the optimal number of clusters by plotting the density of the values in the vector created

```
n_clusters_1_density_rmvn <- density(n_clusters_1_rmvn)
plot(n_clusters_1_density_rmvn, main = "n_clusters_1_rmvn")

n_clusters_2_density_rmvn <- density(n_clusters_2_rmvn)
plot(n_clusters_2_density_rmvn, main = "n_clusters_2_rmvn")

n_clusters_3_density_rmvn <- density(n_clusters_3_rmvn)
plot(n_clusters_3_density_rmvn, main = "n_clusters_3_rmvn")

n_clusters_4_density_rmvn <- density(n_clusters_4_rmvn)
plot(n_clusters_4_density_rmvn, main = "n_clusters_4_rmvn")
```









As we can see from the plot, almost all the four methods suggest that the optimal number of clusters is 4. Although we see more different values than in the previous point.

3 Point 3

Let's assume that we have the same dataset to cluster and the same B datasets generated from a uniform as in the step 2 seen in class:

- if $q = 1$:
  $K_{0,1} = \min K : Gap(K) > Gap(K^* - s_{K^*})$
- if $q = 2$:
  $K_{0,1} = \min K : Gap(K) > Gap(K^* - 2s_{K^*})$

For the latter case, the threshold for selecting the optimal number of clusters is stricter compared to the case when $q = 1$: therefore, we can say that the set of K values that satisfy the criterion for $q = 2$ will be a subsets of the set of K values that satisfy the first realtionship.
Then we can say that $K_{0,1} \geq K_{0,2}$ for any data set.

4 Point 4

Here we need to define a dissimilarity measure that expresses the idea that two observations are similar if they tend to be relatively larger or smaller on the same variables, so that in particular $d^*(x_1, x_2) > d^*(x_1, x_3)$ where x_1 and x_3 are the vectors defined in the exercise statement.
Given this objective, I propose employing a dissimilarity metric grounded in cross-correlation. Cross-correlation measures the similarity of two sequences, accounting for the phase difference. By using cross-correlation, we leverage this ability to detect patterns of similarity, particularly in cases where the shape of the sequences align.

Let's introduce the cross-correlation coefficient: given two sequences $x_i = [x_{i,1}, x_{i,2}, ...x_{i,p}]$ and $x_j = [x_{j,1}, x_{j,2}, ..., x_{j,p}]$

$$cc = \frac{(\vec{x_i} \cdot \vec{x_j})}{\sqrt{(x_{i,1}^2 + x_{i,2}^2 + ... + x_{i,p}^2)(x_{j,1}^2 + x_{j,2}^2 + ... + x_{j,p}^2)}}$$

At the denominator we have the dot product of the two vectors, at the denominator the sum of the squares values of the vectors.
Values near 1 indicates a perfect match in shape, 0 indicates no correlation and -1 indicates perfect anti-correlation.
In order to have a distance measure we can substract to 1 the value of the coefficient:

$$D(cc) = 1 - cc$$

The range of values that takes the distance based on cross-correlation is the set [0, 2].

We implement this on R

```
x1 <- c(1, 4, 5, 4, 2,1,1,4)
x2 <- c(2, 3, 2, 2, 3,3,3,3)
x3 <- c(7, 11, 11, 12, 9,8,8,12)

# Define a function to compute the modified cross-correlation dissimilarity
cross_correlation_coefficient <- function(x, y) {
  sum_xy <- sum(x * y)
  sum_x_sq <- sum(x^2)
  sum_y_sq <- sum(y^2)
  return(sum_xy / sqrt(sum_x_sq * sum_y_sq))
}

# Calculate cross-correlation coefficients
correlation_coefficient_x1_x2 <- cross_correlation_coefficient(x1, x2)
correlation_coefficient_x1_x3 <- cross_correlation_coefficient(x1, x3)

# Print the results
print(paste("Cross-Correlation distance between x1 and x2:",
            1- correlation_coefficient_x1_x2))
print(paste("Cross-Correlation distance between x1 and x3:",
            1- correlation_coefficient_x1_x3))
```

As a result we have

```
[1] "Cross-Correlation distance between x1 and x2: 0.170711571137271" [1]
  "Cross-Correlation distance between x1 and x3: 0.0600518428811799"
```

We can see that

$$D(cc)^*(x_1, x_2) > D(cc)^*(x_1, x_3)$$