Homework 2: Unsupervised Learning and Clustering

Harvard CS 109B, Spring 2018

Jan 23, 2018

Contents

Homework 2 is due February 19, 2018

"Handy" Algorithms

In this assignment, you will be working with data collected from a motion capture camera system. The system was used to record 12 different users performing 5 distinct hand postures with markers attached to a left-handed glove. A set of markers on the back of the glove was used to establish a local coordinate system for the hand, and 11 additional markers were attached to the thumb and fingers of the glove. Three markers were attached to the thumb with one above the thumbnail and the other two on the knuckles. Finally, 2 markers were attached to each finger with one above the fingernail and the other in the middle of the finger. A total of 36 features were collected resulting from the camera system. Two other variables in the dataset were the ID of the user and the posture that the user made.

The data were partially preprocessed. First, all markers were transformed to the local coordinate system of the record containing them. Second, each transformed marker with a norm greater than 200 millimeters was eliminated. Finally, any record that contained fewer than three markers was removed.

A few issues with the data are worth noting. Based on the manner in which data were captured, it is likely that, for a given record and user, there exists a near duplicate record originating from the same user. Additionally, There are many instances of missing data in the feature set. These instances are denoted with a ? in the dataset. Finally, there is the potential for imbalanced classes, as there is no guarantee that each user and/or posture is represented with equal frequency in the dataset.

The dataset, provided in CSV format, contains 78,095 rows and 38 columns. Each row corresponds to a single instant or frame as recorded by the camera system. The data are represented in the following manner:

`Class - Integer. The hand posture of the given obervation, with 1=Fist (with thumb out), 2=Stop (hand flat), 3=Point1 (point with index finger), 4=Point2 (point with index and middle fingers), 5=Grab (fingers curled as if to grab).`

`User - Integer. The ID of the user that contributed the record.`

`X0, Y0, Z0, X1, Y1, Z1, ..., X11, Y11, Z11 - Real. The x-coordinate, y-coordinate and z-coordinate of the twelve unlabeled marker positions.`

1 Missing Data Imputation

The data contain many missing values. Before attempting to perform any statistical procedures, we will need to address the missing data. One way to address missing data is to impute it.

Given the knowledge of how the data was collected, we can hypothesize that there are two ways in which the data might cluster together: by user and by posture. Perhaps the users have significantly different heights and/or hand sizes, resulting in the data generated by each user to be distinct from each other. Or, perhaps the hand postures are sufficiently unique such that the markers on the glove tend to be grouped together by the posture, regardless of who the user is. We will examine these hypotheses to see if either one provides a reasonable way to impute the data.

For this part of the assignment, you will want to use the following libraries:

```
# import libraries
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(cluster)
library(mclust)
```

```
## Package 'mclust' version 5.4
## Type 'citation("mclust")' for citing this R package in publications.
```

```
library(factoextra)
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```
library(NbClust)
library(dbscan)
library(reshape2)
library(devtools)
library(ggfortify)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
# read in data
data <- read.csv("postures.csv")
```

Write code to impute the missing data values with the mean of their respective feature (column), grouped by both users and postures. That is, you should create two new dataframes: one where the missing values are replaced by the mean of the user's feature, and one where the missing values are replaced by the mean of the posture's feature.

Hint: when loaded into R, the raw CSV might list the observations as factors. You will want to change that. One way to convert factors to numeric is to cast the columns of the dataframe like so:

```r
# convert factors to numeric
for (i in 1:ncol(data)) {
      data[,i] <- as.numeric(as.character(data[,i]))
}
```

The "dplyr" package might also be useful for data cleaning.

Helper Function: Col means data imputation

Hint: function to impute means for one column, you will want to adapt this for all the necessary columns.

```r
# function to impute means for one column
impute_column <- function(column) {
  missing_indices <- which(is.na(column))
  column_mean <- mean(as.numeric(column[-missing_indices]))
  column[missing_indices] <- column_mean
  return(column)
}

# function to impute means for one df
impute_means <- function(df){
  for (j in 3:ncol(df)){
    missing_indices_j <- which(is.na(df[,j]))
    if (length(missing_indices_j > 0 )){ # impute only if column has missing values
      if(length(missing_indices_j) != nrow(df)){ # if column has non-NA values
        df[,j] <- impute_column(df[,j])
      }
      else{ # column has all NA values, impute with global df mean
        df[,j] <- mean(colMeans(df[,-c(1,2)]), na.rm = TRUE)
      }
    }
  }
  return(df)
}
```

For data imputation, you can use this code after you make necessary adjustment above

First, create each imputed posture data frame

```r
for (posture in unique(data$Class)) {
  df <- data %>% filter(Class == posture)
  df_imputed_means <- impute_means(df)
  assign(paste0("imputed_posture_means", posture), df_imputed_means)
}
```

Second, create each imputed user data frame

```r
for (user in unique(data$User)) {
  df <- data %>% filter(User == user)
  df_imputed_means <- impute_means(df)
  assign(paste0("imputed_user_means", user), df_imputed_means)
}
```

Then, stack data frames back together (inserting appropriate variable names instead of ellipsis).

```
imputed_posture_all <- rbind(imputed_posture_means1, imputed_posture_means2,
                             imputed_posture_means3, imputed_posture_means4,
                             imputed_posture_means5)
imputed_user_all <- rbind(imputed_user_means0, imputed_user_means1, imputed_user_means2,
                          imputed_user_means4, imputed_user_means5, imputed_user_means6,
                          imputed_user_means7, imputed_user_means8, imputed_user_means9,
                          imputed_user_means10, imputed_user_means11, imputed_user_means12,
                          imputed_user_means13, imputed_user_means14)
```

Finally manage your stack and remove clutter (inserting appropriate variable names instead of ellipsis).

```
rm(imputed_posture_means1, imputed_posture_means2, imputed_posture_means3,
   imputed_posture_means4, imputed_posture_means5)

rm(imputed_user_means0, imputed_user_means1, imputed_user_means2, imputed_user_means4,
   imputed_user_means5, imputed_user_means6, imputed_user_means7, imputed_user_means8,
   imputed_user_means9, imputed_user_means10, imputed_user_means11, imputed_user_means12,
   imputed_user_means13, imputed_user_means14)

rm(posture)
rm(user)
rm(df)
rm(df_imputed_means)

unique((imputed_posture_all %>% filter(Class==5))[,38]) # should return 35.67459
```

```
## [1] 35.67459
```

2 Clustering with k-means

Now that we have imputed the missing values, we can investigate our hypotheses by examining how well the data clusters by user and by posture. In the following problem, we will explore a wider choice of options for the number of centroids.

We will first use the k-means algorithm to carry out the clustering.

(a) Using the "kmeans" function in R, run kmeans on the features using 14 centroids (representing the 14 users). Do not run the algorithm on the entire dataset, as the eventual visualization can become unwieldy. Instead, obtain a random sample of 2,000 observations without replacement, and run the algorithm on the sampled values. Set a seed at '42' and set the 'nstart' parameter in the kmeans function to '46' to ensure that we can check your results. Hint: You can take a random sample of the dataframe's indices using R's "sample" function:

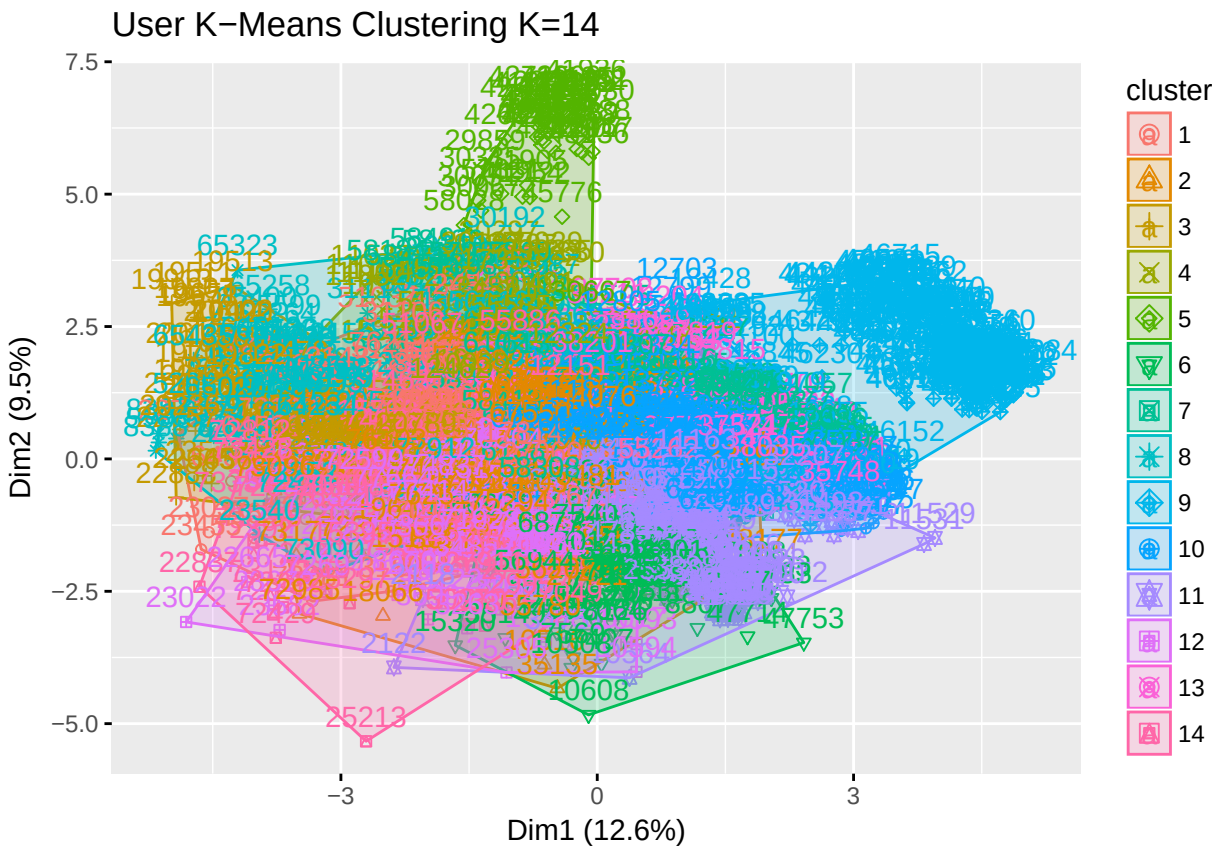
```
# scale the features
imputed_user_all[, 3:ncol(imputed_user_all)] <- scale(imputed_user_all[, 3:ncol(imputed_user_all)])
imputed_posture_all[, 3:ncol(imputed_posture_all)] <- scale(imputed_posture_all[, 3:ncol(imputed_posture_all)

# take a random sample of 2000 observations
set.seed(42)
samp <- sample(x=1:nrow(imputed_user_all), size=2000, replace=F)
user_cluster_target <- imputed_user_all$User[samp]
user_cluster_features <- imputed_user_all[samp, 3:ncol(imputed_user_all)]
```

```
# kmeans clustering by user
user.kmeans <- kmeans(user_cluster_features, 14, iter.max=100, nstart=46)
```

(b) Use the "fviz-cluster" function to visualize the results of your clustering algorithm (you will probably want to press the "Zoom" button in the plots section of R Studio so that you can see the results on a larger plot). How much of the variance in the data is explained by the first two principal components? Does it look like the data separate into 14 distinct clusters?

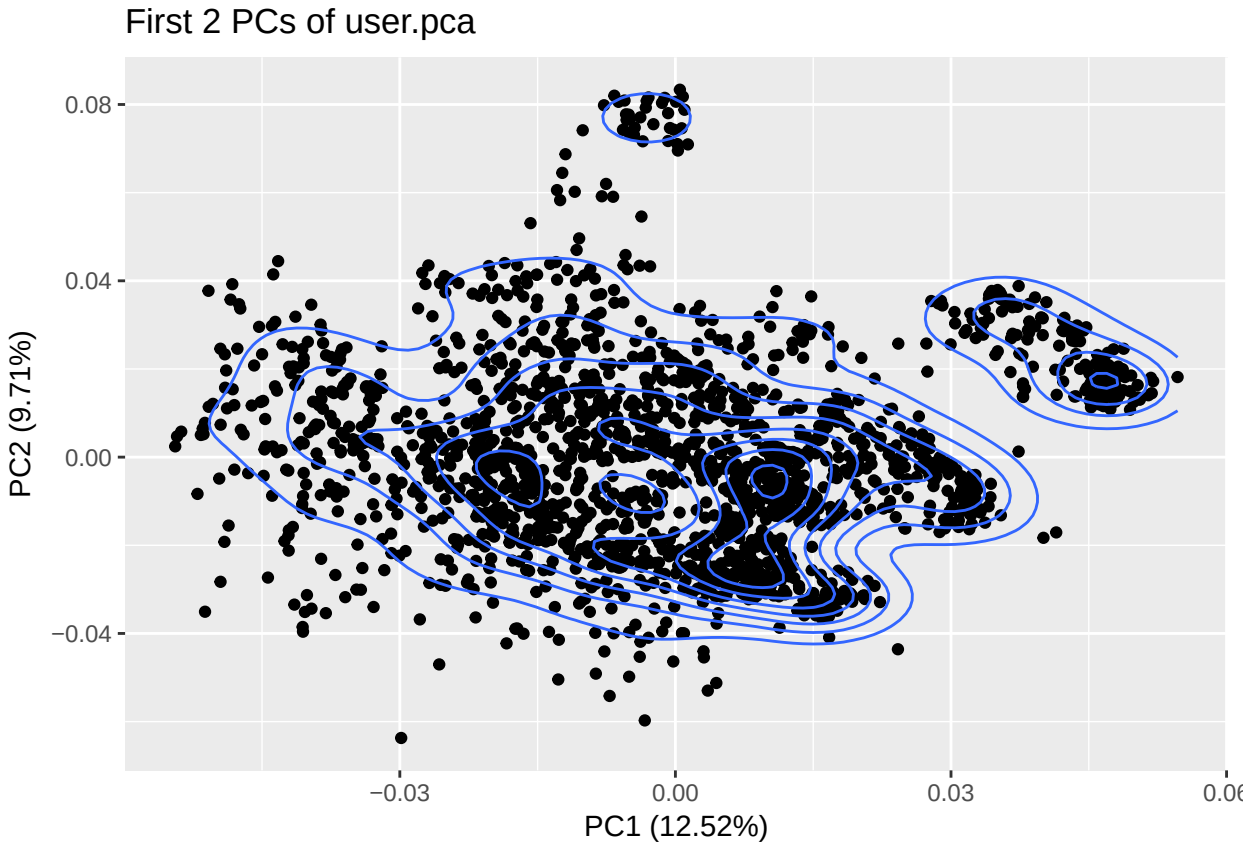
```
fviz_cluster(user.kmeans, data=user_cluster_features, main='User K-Means Clustering K=14')
```




```
# PCA
user.pca <- prcomp(user_cluster_features, center=T)
user.pc.vars <- cumsum(user.pca$sdev^2/sum(user.pca$sdev^2))
print(paste0("Variance Explained by the first 2 principal components = ", user.pc.vars[2]))
```

```
## [1] "Variance Explained by the first 2 principal components = 0.222325604955223"
```

```
autoplot(user.pca) + geom_density2d() + ggtitle('First 2 PCs of user.pca')
```



ANSWER (2. Clustering with k-means (b)):
The amount of variance explained by the first 2 principla components is 0.2223256. The clusters do not seem to be distinct. They have a lot of overlap with each other.

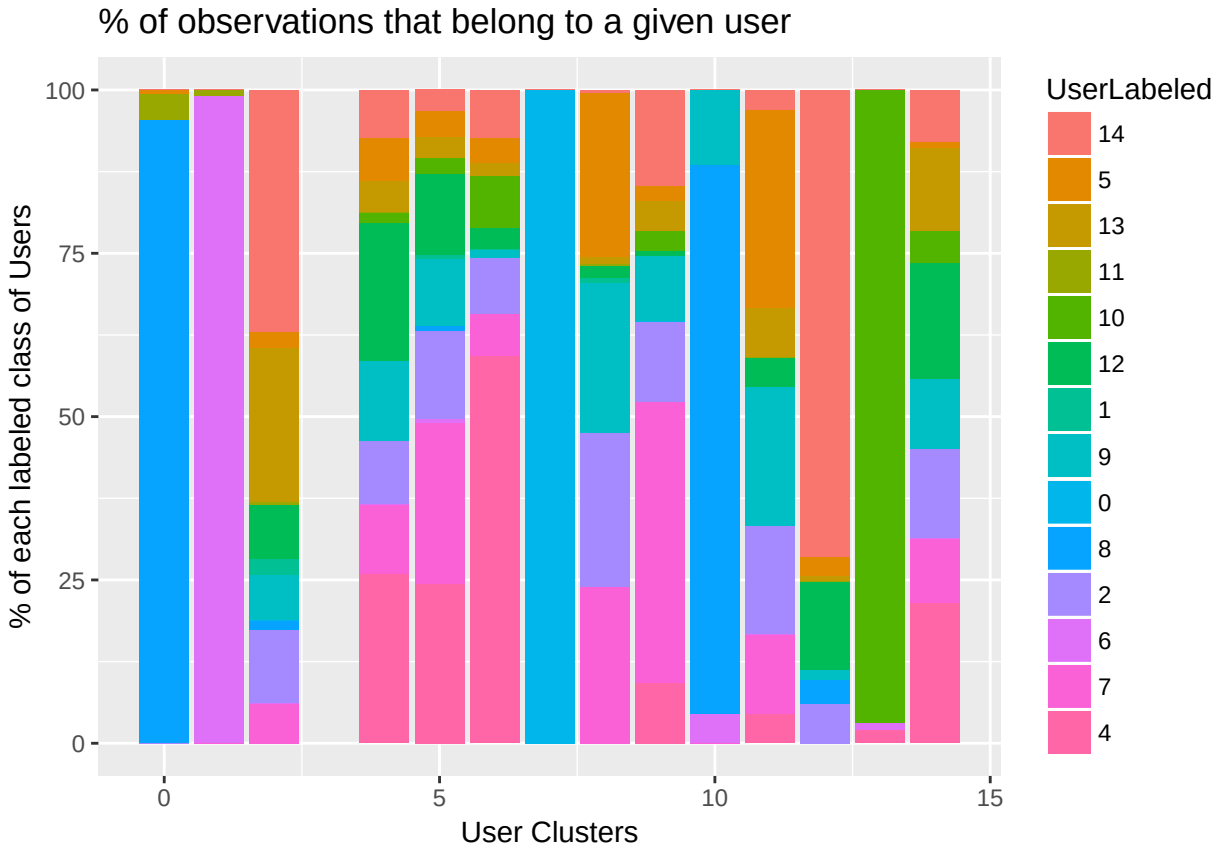
(c) Compare the results from your clustering algorithm to the actual users. Specifically, make a bar plot showing the assigned cluster from kmeans against the actual user of the observation. Have the area of each bar correspond to the the percentage of observations that belong to a given user. Based on this graph, does it look like the data clusters well by user?

Hint: You will probably want to make use the "geom-bar" function in ggplot2 to do this.

```
# calculate the percentage of observations that belong to a given user
user_kmeans_percent <- matrix(rep(0, 14*14), ncol = 14)
for (i in 1:14) {
  cluster_i <- which(user.kmeans$cluster==i) # indices of users clustered as i
  clustered_users <- user_cluster_target[cluster_i] # users clustered as i
  for (j in 1:14) {
    if (j < 4) {
      type <- j-1
    }
    else {
      type <- j
    }
    users_j <- clustered_users[clustered_users==type]
    user_kmeans_percent[i, j] <- length(users_j) / length(cluster_i) * 100
  }
}
# print(rowSums(user_kmeans_percent))
rownames(user_kmeans_percent) <- unique(user_cluster_target)
```

```r
colnames(user_kmeans_percent) <- unique(user_cluster_target)

# plot
df_user_kmeans_percent <- as.data.frame(user_kmeans_percent)
df_user_kmeans_percent$UserId <- unique(user_cluster_target)
df_user_kmeans_percent_melted <- melt(df_user_kmeans_percent, id.var="UserId")
colnames(df_user_kmeans_percent_melted) <- c("UserClustered", "UserLabeled", "value")
ggplot(df_user_kmeans_percent_melted, aes(x = UserClustered, y = value, fill = UserLabeled)) +
  geom_bar(stat = "identity") + xlab("User Clusters") +
  ylab("% of each labeled class of Users") +
  ggtitle("% of observations that belong to a given user")
```



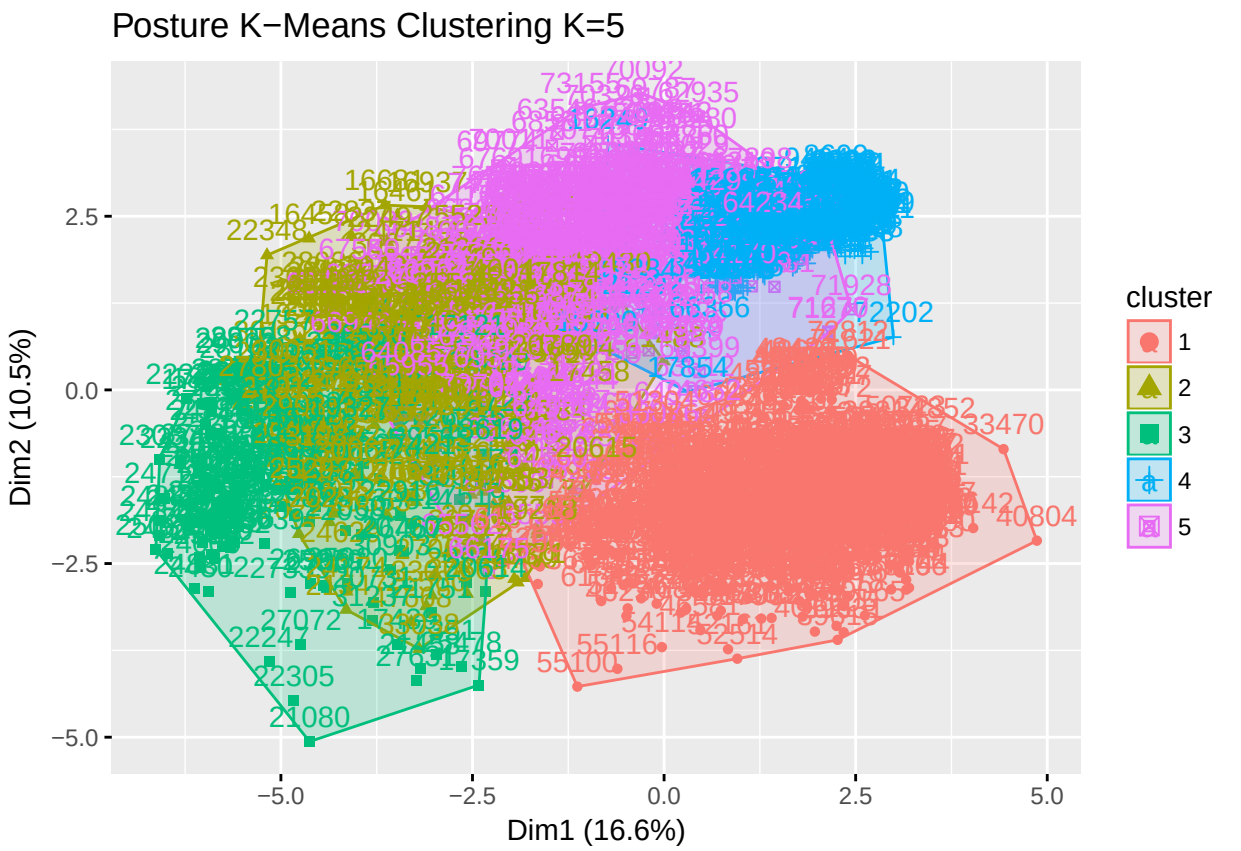
ANSWER (2. Clustering with k-means (c)):
Based on the percentage bar plot, only the cluster for user 0, 1, 7, 10, 13 are pure. Other clusters contain multiple labels of users. Therefore, the data does not cluster well by user.

(d) Repeat all of the above steps, but group by posture rather than by user. That is, run the kmeans algorithm with 5 centroids instead of 14. Construct the same plots and answer the same questions.

```r
# take a random sample of 2000 observations
set.seed(42)
samp <- sample(x=1:nrow(imputed_posture_all), size=2000, replace=F)
posture_cluster_target <- imputed_posture_all$Class[samp]
posture_cluster_features <- imputed_posture_all[samp, 3:ncol(imputed_posture_all)]

# kmeans clustering by posture
posture.kmeans <- kmeans(posture_cluster_features, 5, iter.max=20, nstart=46)
fviz_cluster(posture.kmeans, data=posture_cluster_features,
```

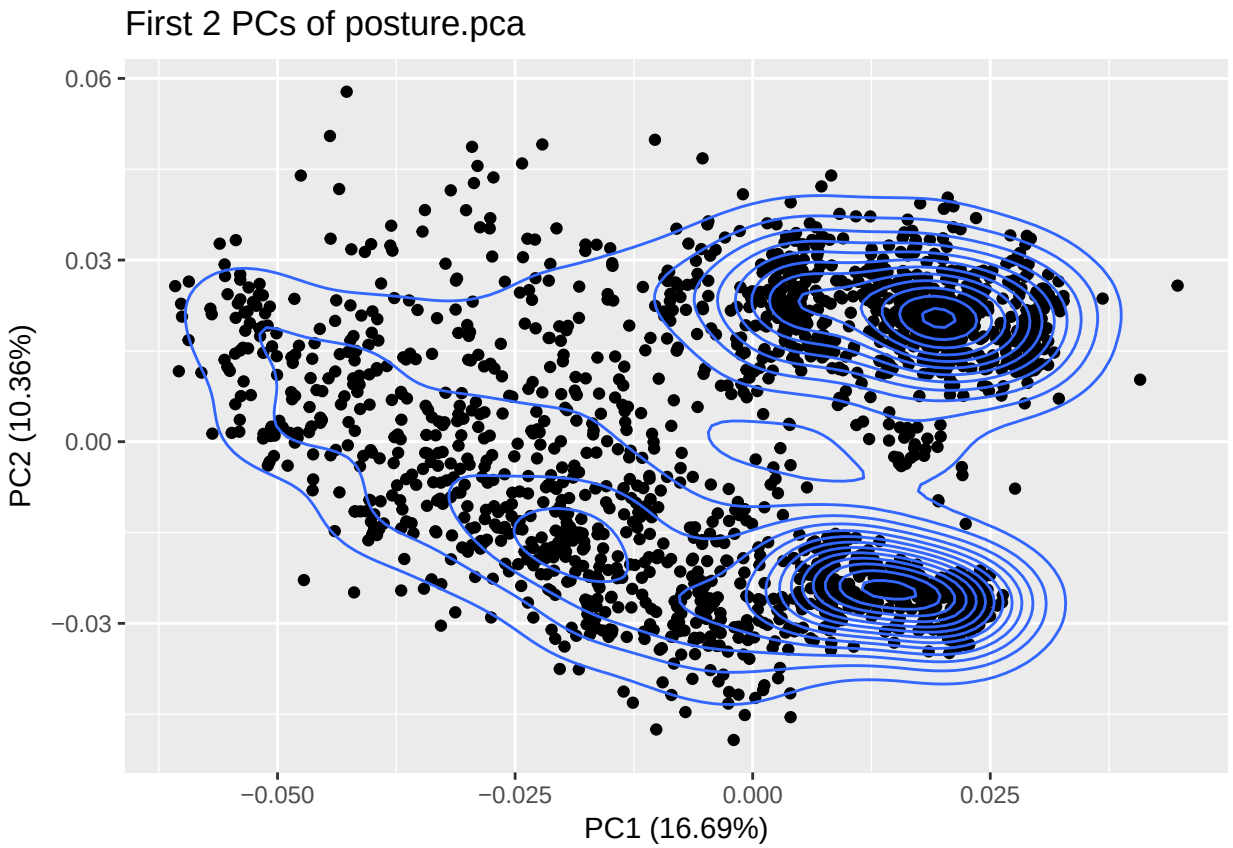
```
          main='Posture K-Means Clustering K=5')
```




```
# PCA
posture.pca <- prcomp(posture_cluster_features, center=T)
posture.pc.vars <- cumsum(posture.pca$sdev^2/sum(posture.pca$sdev^2))
print(paste0("Variance Explained by the first 2 principal components = ", posture.pc.vars[2]))
```

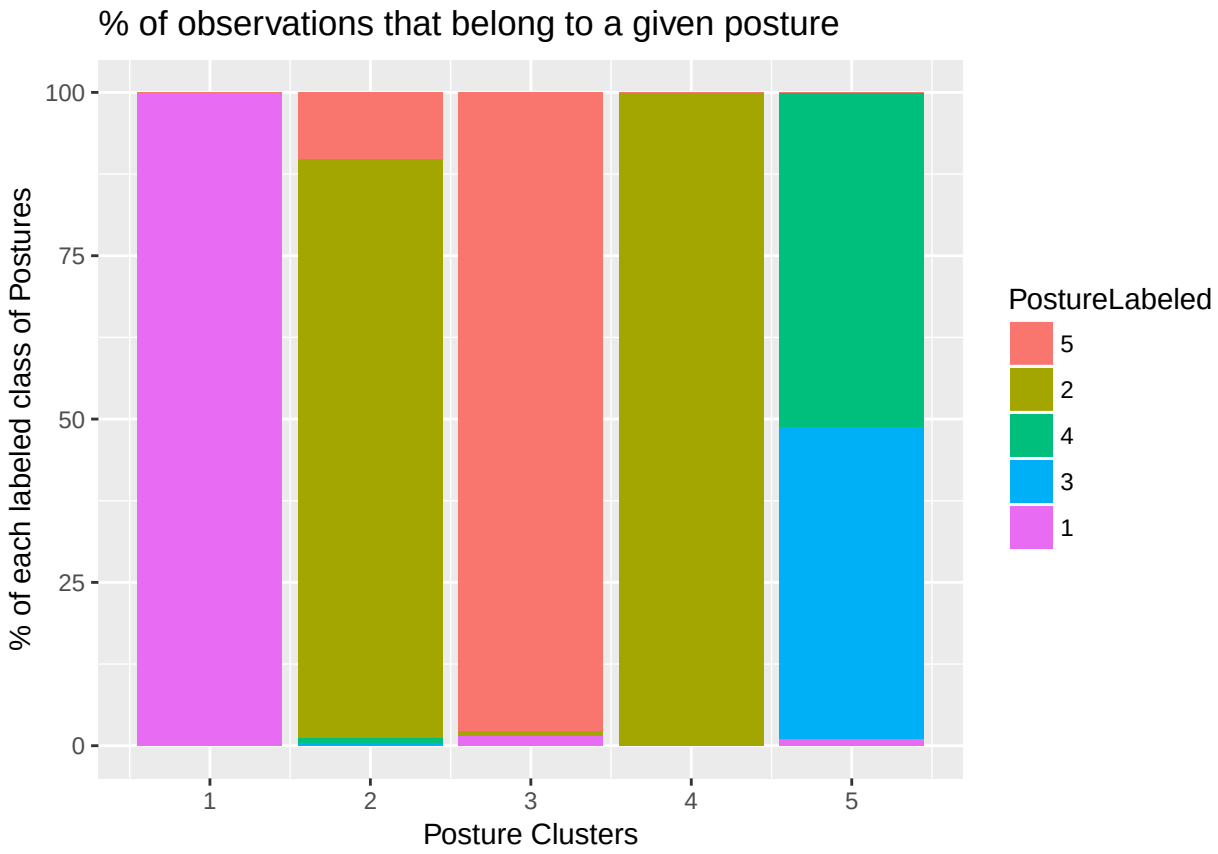
```
## [1] "Variance Explained by the first 2 principal components = 0.270498295750811"
```

```
autoplot(posture.pca) + geom_density2d() + ggtitle('First 2 PCs of posture.pca')
```




```
# calculate the percentage of observations that belong to a given posture
posture_kmeans_percent <- matrix(rep(0, 5*5), ncol = 5)
for (i in 1:5) {
  cluster_i <- which(posture.kmeans$cluster==i) # indices of postures clustered as i
  clustered_postures <- posture_cluster_target[cluster_i] # postures clustered as i
  for (j in 1:5) {
    type <- j
    postures_j <- clustered_postures[clustered_postures==type]
    posture_kmeans_percent[i, j] <- length(postures_j) / length(cluster_i) * 100
  }
}
# print(rowSums(posture_kmeans_percent))
rownames(posture_kmeans_percent) <- unique(posture_cluster_target)
colnames(posture_kmeans_percent) <- unique(posture_cluster_target)

# plot
df_posture_kmeans_percent <- as.data.frame(posture_kmeans_percent)
df_posture_kmeans_percent$PostureId <- unique(posture_cluster_target)
df_posture_kmeans_percent_melted <- melt(df_posture_kmeans_percent, id.var="PostureId")
colnames(df_posture_kmeans_percent_melted) <- c("PostureClustered", "PostureLabeled", "value")
ggplot(df_posture_kmeans_percent_melted,
       aes(x = PostureClustered, y = value, fill = PostureLabeled)) +
  geom_bar(stat = "identity") + xlab("Posture Clusters") +
  ylab("% of each labeled class of Postures") +
  ggtitle("% of observations that belong to a given posture")
```



ANSWER (2. Clustering with k-means (d)):
The amount of variance explained by the first 2 principla components is 0.2704983. The clusters seem to be distinct.

Based on the percentage bar plot, all clusters except for cluster 5 are mostly pure. Therefore, we can say that the data clusters well by user.

(e) What do the results of the bar plot clustered by posture suggest about the data? Why does this make sense in the context of what we know about the problem? **ANSWER (2. Clustering with k-means (e)):**
The bar plot clustered by posture shows that 1) there are 2 different clusters (2 and 4) that have the same posture, while 2) there's another cluster (5) that consists of 2 distinct postures.

Based on the posture classes 1=Fist (thumb out), 2=Stop (hand flat), 3=Point1 (point with index finger), 4=Point2 (point with index and middle fingers), and 5=Grab (fingers curled as if to grab), the hand flat posture (label 2) could be different depending on whether they fingers together or spread. Pointing with index finger only (label 3) and pointing with both index and middle fingers (label 4) are similar postures, and thus they do not form distinct clusters. Thumb out (label 1) being well clustered could be explained by having one more markers on the thumb than on other fingers. Grab (label 5) being well clustered could be explained by it being very different posture compared to the other posture types.

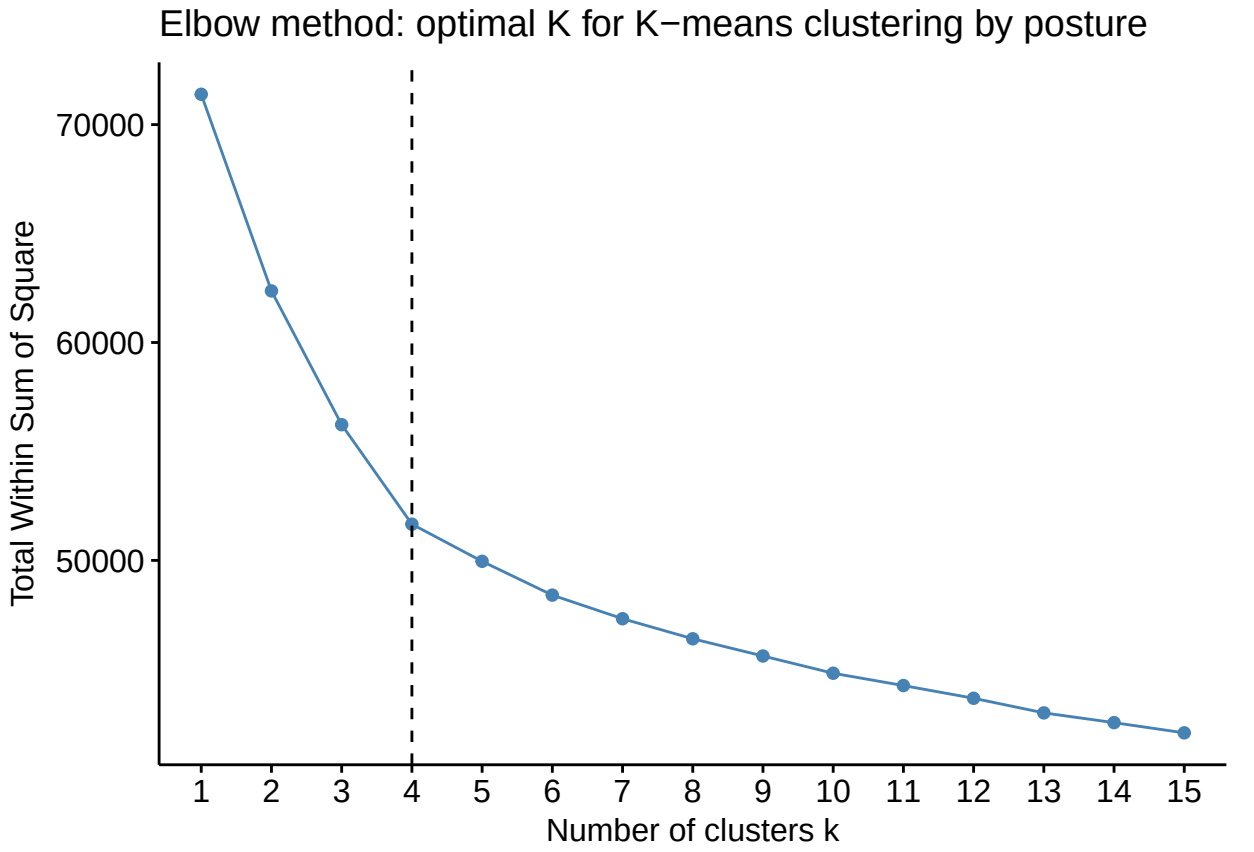
(f) Using all of the information gleaned from this problem, how do you recommend the missing data be imputed? Why? **ANSWER (2. Clustering with k-means (f)):**
Since 1) the portion of explained variance of the imputed data by posture is higher than that of the imputed data by user, and 2) the percentage bar plot shows that "by postures" can better cluster the data than "by user", we would recommend the missing data be imputed by posture.

3 Clustering Evaluation

In the previous problem, we used k-means with 5 and 14 centroids to decide how we should impute missing data. In this problem, we will investigate various ways of evaluating the quality of a clustering assignment.

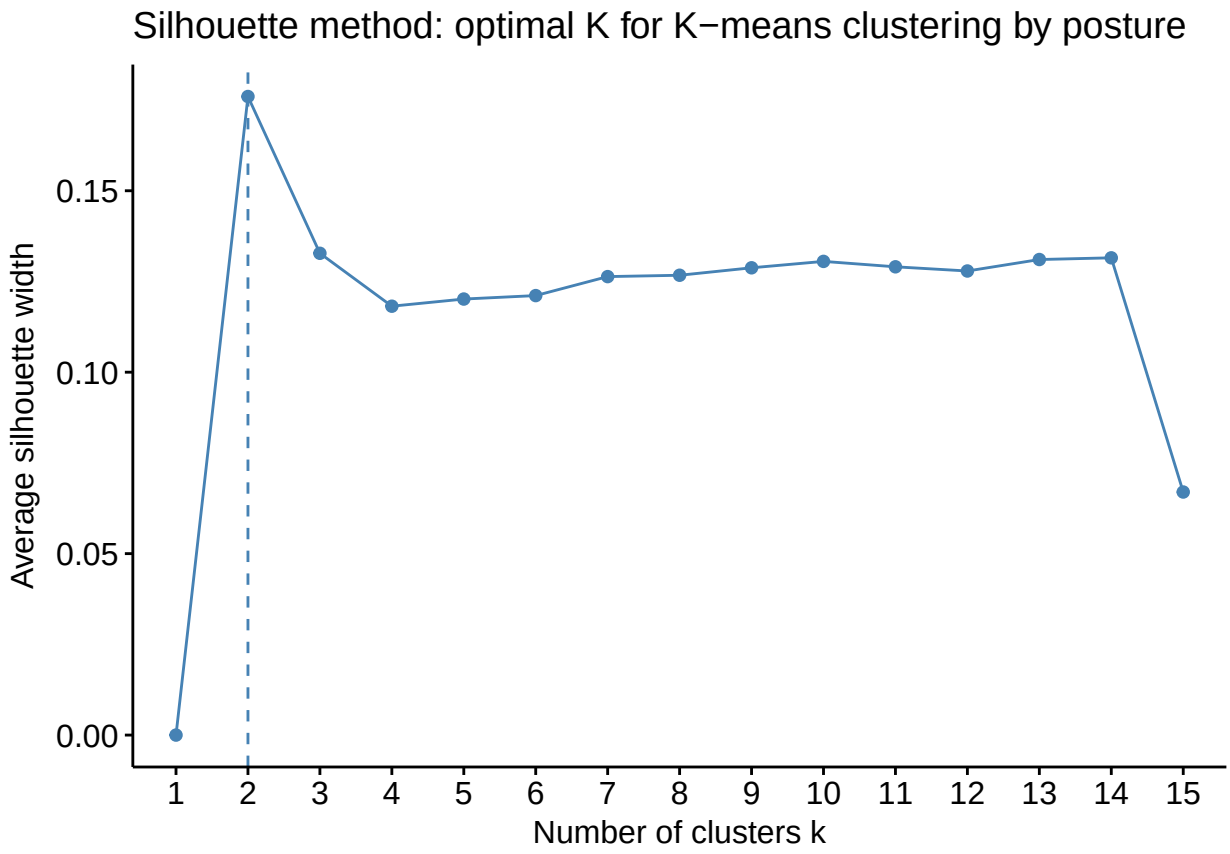
(a) Use the elbow method to evaluate the best choice of the number of clusters, plotting the total within-cluster variation against the number of clusters for k-means clustering with $k \in (1, 2, \ldots 15)$.

```
# elbow - method = "wss"
fviz_nbclust(x = posture_cluster_features, FUNcluster = kmeans,
             method = "wss", k.max = 15, nstart=46) +
  ggtitle("Elbow method: optimal K for K-means clustering by posture") +
  geom_vline(xintercept=4,linetype=2)
```



(b) Use the average silhouette to evaluate the choice of the number of clusters for k-means clustering with $k \in (1, 2, \ldots 15)$. Plot the results.

```
# average silhouette - method = "silhouette"
fviz_nbclust(x = posture_cluster_features, FUNcluster = kmeans,
             method = "silhouette", k.max = 15, nstart=46) +
  ggtitle("Silhouette method: optimal K for K-means clustering by posture")
```



(c) Use the gap statistic to evaluate the choice of the number of clusters for k-means clustering with k ∈ (1, 2, . . . 15). Plot the results. Be patient – this might take a few minutes.

```
gapstat = clusGap(posture_cluster_features, FUN=kmeans, iter.max=30, nstart=46, d.power=2, K.max=15, B=20)
```

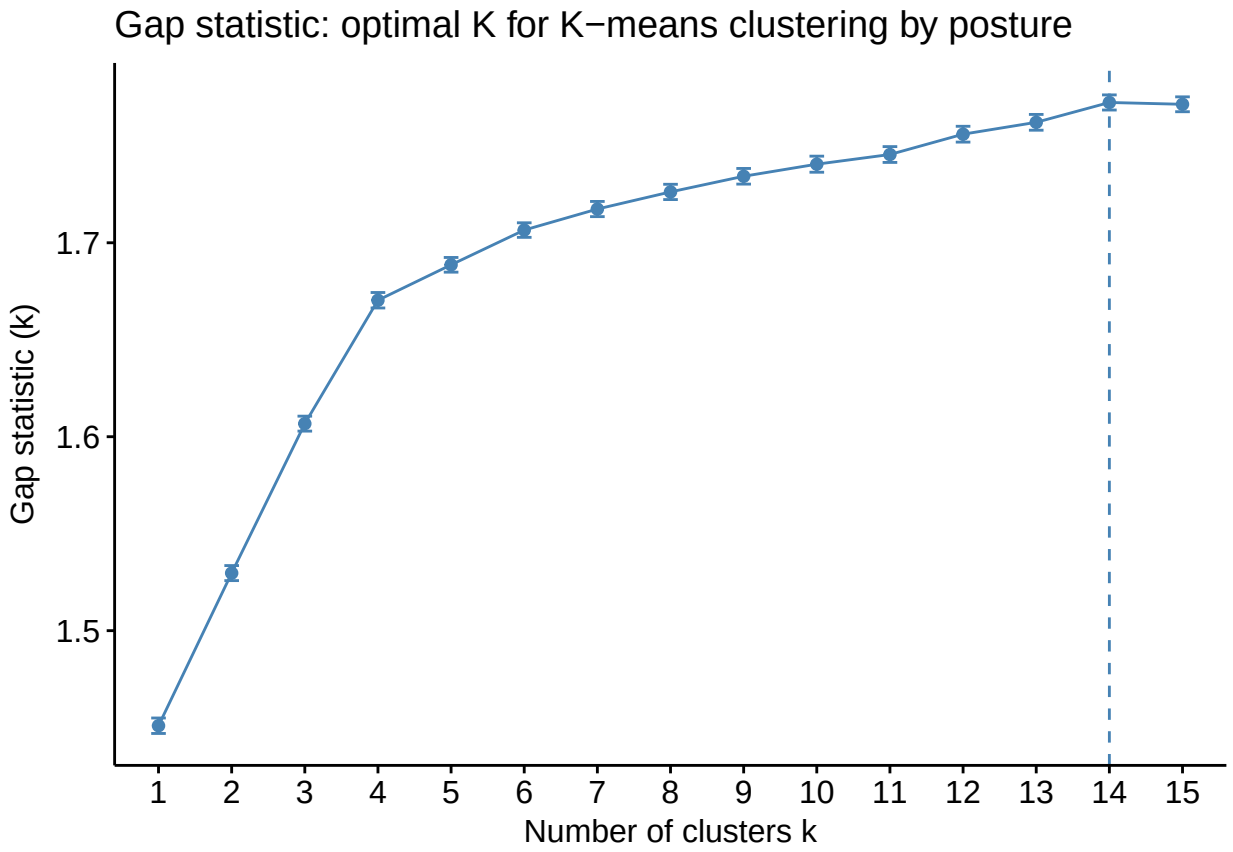
```
## Warning: Quick-TRANSfer stage steps exceeded maximum (= 100000)
```

```
print(gapstat, method="Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = posture_cluster_features, FUNcluster = kmeans, K.max = 15,    B = 20, d.power = 2, iter.max = 30, ns
## B=20 simulated reference sets, k = 1..15; spaceH0="scaledPCA"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 14
##            logW   E.logW      gap      SE.sim
##  [1,] 10.482790 11.93377 1.450982 0.003985654
##  [2,] 10.347646 11.87733 1.529687 0.003873091
##  [3,] 10.244013 11.85076 1.606743 0.003857718
##  [4,] 10.159314 11.82969 1.670376 0.003992468
##  [5,] 10.125772 11.81441 1.688640 0.003783779
##  [6,] 10.094231 11.80078 1.706553 0.003754764
##  [7,] 10.071594 11.78900 1.717401 0.003923555
##  [8,] 10.051920 11.77814 1.726222 0.003935439
##  [9,] 10.034770 11.76906 1.734295 0.004034780
## [10,] 10.020147 11.76067 1.740522 0.004141617
## [11,] 10.007355 11.75287 1.745513 0.004076183
## [12,]  9.989589 11.74558 1.755990 0.004074519
## [13,]  9.976727 11.73884 1.762113 0.004066634
## [14,]  9.959932 11.73228 1.772350 0.003904230
```

```
## [15,]  9.954948 11.72636 1.771416 0.003836039
```

```
fviz_gap_stat(gapstat,
  maxSE=list(method="Tibs2001SEmax", SE.factor=1)) +
  ggtitle("Gap statistic: optimal K for K-means clustering by posture")
```



(d) After analyzing the plots produced by all three of these measures, discuss the number of clusters that you feel is the best fit for this dataset. Defend your answer with evidence from the previous parts of this assignment, the three graphs produced here, and what you surmise about this dataset.

ANSWER (3. Clustering evaluation (d)):
By previous parts of analysis, especially the percentage bar plot by user and posture, we found that posture can better cluster the data than user can. However, elbow method picks an optimal K to be 4; average silhouette picks an optimal to be 2 while the gap statistic picks an optimal K to be 14. (Number of clusters????)

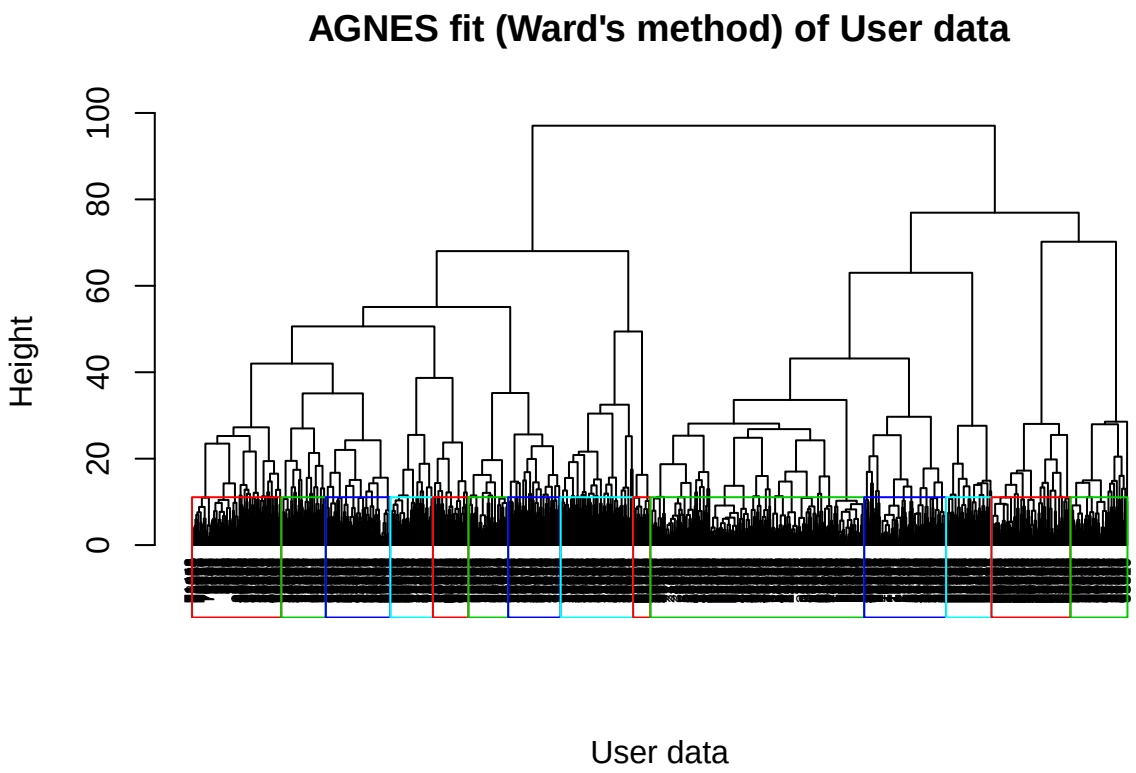
4 Other Clustering Algorithms

Up until now, we have used the k-means algorithm to cluster the data. In this problem, we will explore other methods used to create clusters.

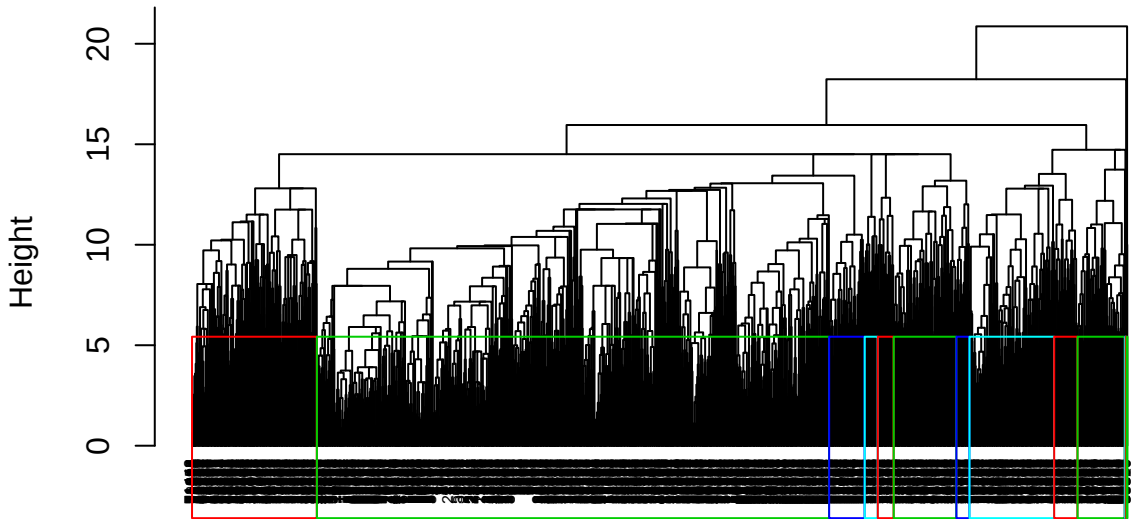
(a) Hierarchical clustering: Implement agglomerative clustering (using Ward's method) and divisive clustering. Plot the results of these algorithms using a dendrogram and interpret the results. Hint: Use the "agnes" and "diana" functions, respectively.

```
# agglomerative clustering (using Ward's method)
user.agnes <- agnes(user_cluster_features, method="ward")
```

```r
pltree(user.agnes, cex=0.5, hang= -1,
  main="AGNES fit (Ward's method) of User data",
  xlab="User data",sub="")
rect.hclust(user.agnes, k=14, border=2:5)
```



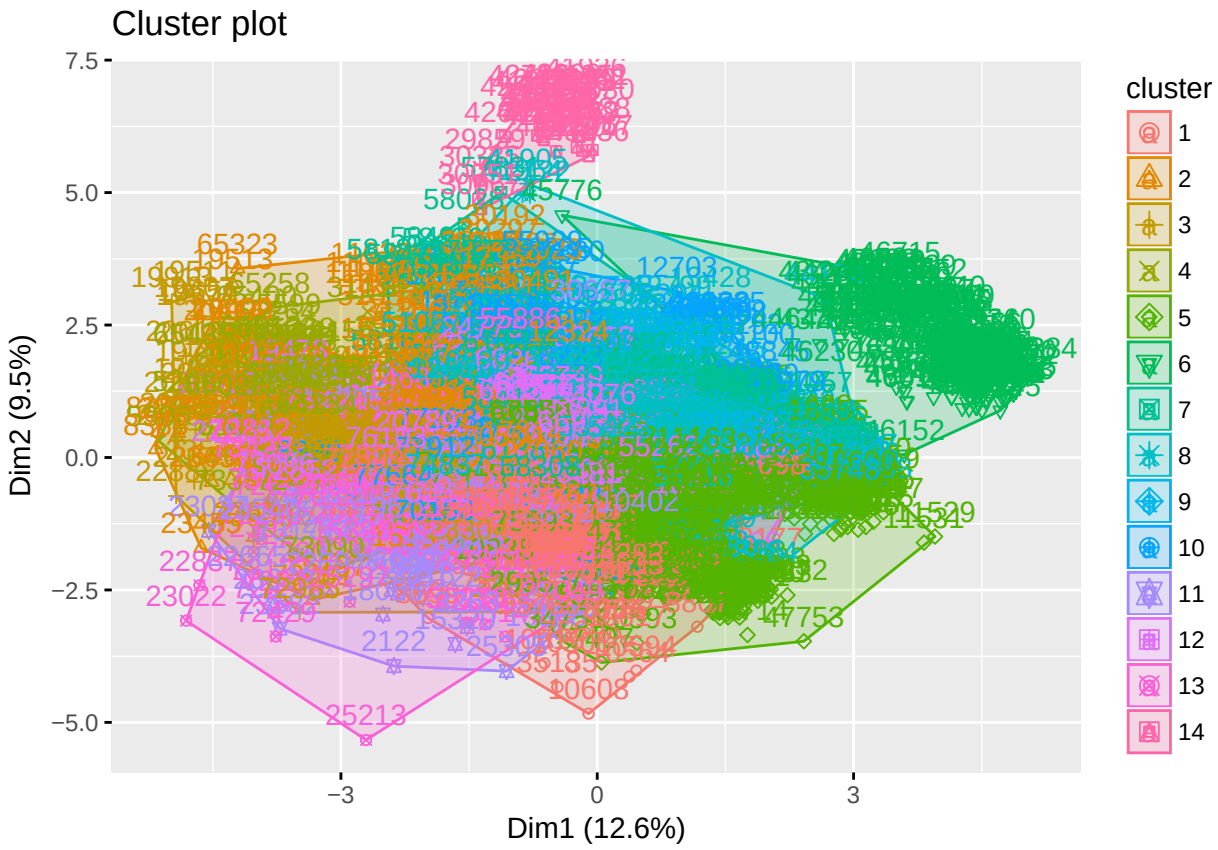

```r
# divisive clustering - diana
user.diana = diana(user_cluster_features)
pltree(user.diana, cex=0.5, hang= -1,
  main="DIANA fit of User data", xlab="User data", sub="")
rect.hclust(user.diana, k=14, border=2:5)
```



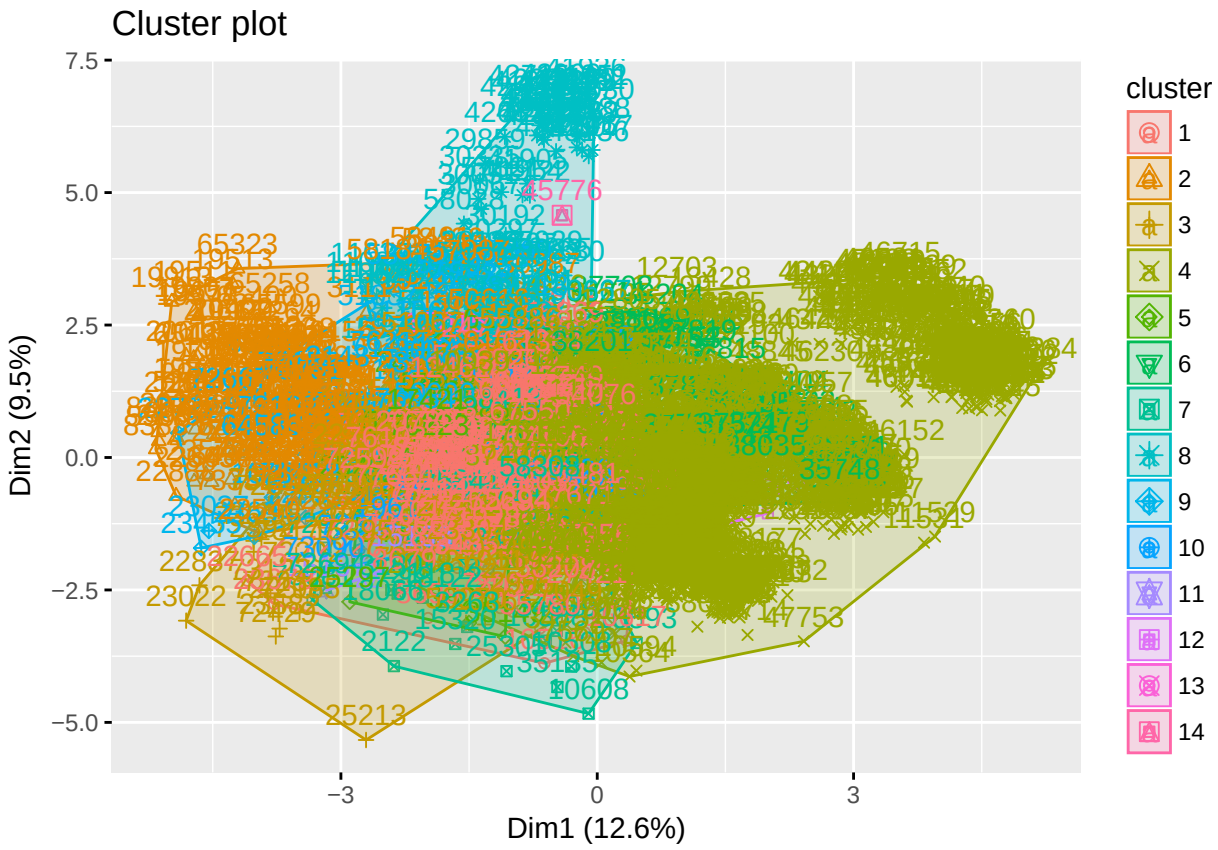
SWER (4. Other Clustering Algorithms (a)): AN-

In a dendrogram, the height of a cluster represents the largest diameter of the cluster, i.e., the largest dissimilarity between two members points within this cluster. We found that the maximum height of aggolomerative clustering is larger than the maximum height of divisive clustering. The resulting 14 clusters from aggolomerative clustering have more dissimilar heights than the from divisive clustering. This is because aggolomerative clustering is greedy picking the most similar pair first with individual data points and then with larger groups, while devisive clustering is greedy with first the entire group and then smaller ones. (Interprete results - Agnes VS. Diana ????)

```
user.grp.agnes <- cutree(user.agnes, k=14)
fviz_cluster(list(data=user_cluster_features, cluster=user.grp.agnes))
```




```
user.grp.diana <- cutree(user.diana, k=14)
fviz_cluster(list(data=user_cluster_features, cluster=user.grp.diana))
```



(b) Soft clustering: Run fuzzy clustering and a Gaussian mixture model on the scaled features. For the fuzzy clustering, run the algorithm with 5 and 14 clusters and plot the results using the "fviz-silhouette" function. For the Gaussian mixture model, the "Mclust" algorithm chooses the optimal number of clusters internally; report the number of clusters it selects. Also display the membership probabilities for the first 10 observations in your sample.

Hint: Use the "fanny" and "Mclust" functions, respectively. You might need to adjust the "memb.exp" parameter to something between 1 and 2 to get the function to run correctly. Make sure to include analysis for trial-and-error of fanny parameters. Justify your results.

```
# fuzzy clustering - 5 clusters
param <- seq(1.1, 2, 0.1)
for (i in param){
  print(i)
  user.fanny.k5 = fanny(user_cluster_features, k=5, memb.exp = i)
}
```

```
## [1] 1.1
## [1] 1.2
```

```
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
## [1] 1.3
```

```
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

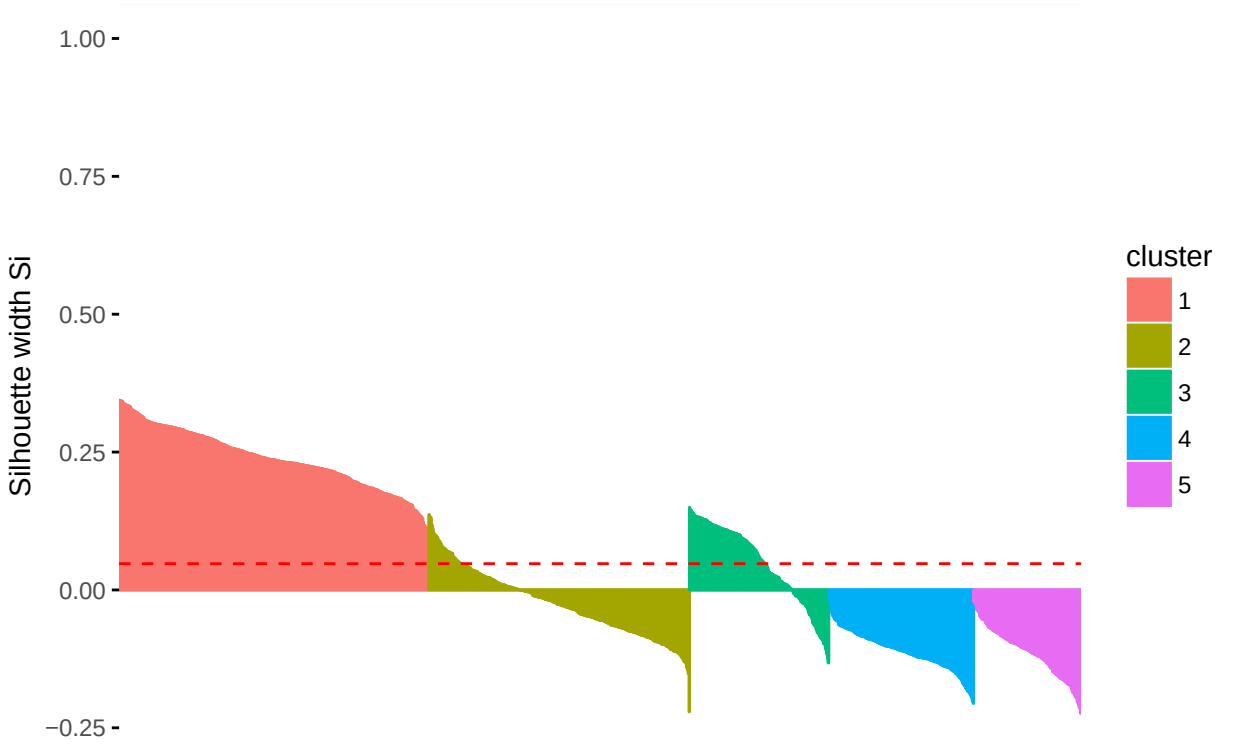
```
## [1] 1.4
```

```
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
```

```
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
## [1] 1.5
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
## [1] 1.6
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
## [1] 1.7
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
## [1] 1.8
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
## [1] 1.9
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
## [1] 2
## Warning in fanny(user_cluster_features, k = 5, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
user.fanny.k5.opt = fanny(user_cluster_features, k=5, memb.exp = 1.1)
fviz_silhouette(silhouette(user.fanny.k5.opt),
    main=paste0("Silhouette plot for FUZZY clustering - 5 clusters, optimal memb.exp = ", 1.1))
```

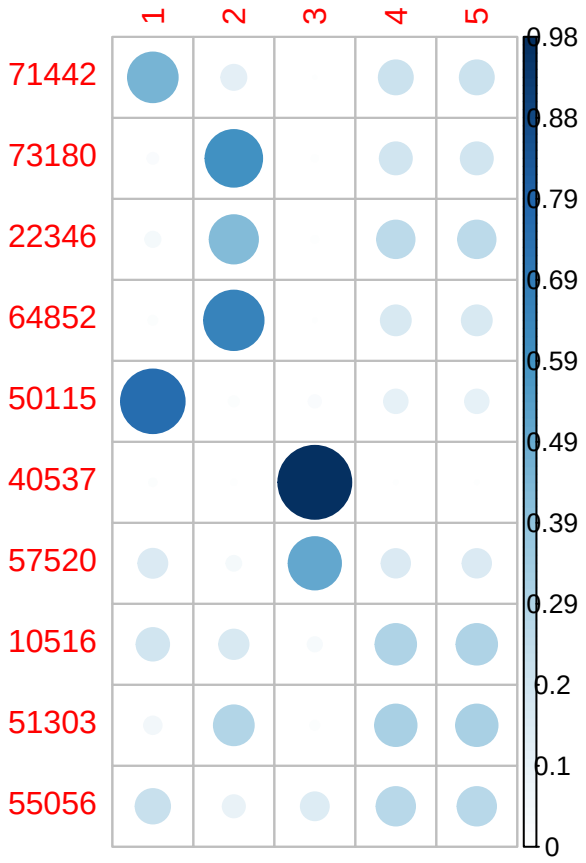
```
##   cluster size ave.sil.width
## 1       1  644          0.24
## 2       2  542         -0.02
## 3       3  290          0.05
## 4       4  301         -0.11
## 5       5  223         -0.11
```




```
# the membership probabilities for the first 10 observations in the sample
print(head(round(user.fanny.k5.opt$membership,3), 10))
```

```
##        [,1]  [,2]  [,3]  [,4]  [,5]
## 71442 0.455 0.116 0.002 0.213 0.213
## 73180 0.023 0.591 0.008 0.189 0.189
## 22346 0.043 0.429 0.009 0.259 0.259
## 64852 0.015 0.651 0.003 0.165 0.165
## 50115 0.746 0.020 0.029 0.103 0.103
## 40537 0.011 0.001 0.981 0.003 0.003
## 57520 0.155 0.041 0.505 0.150 0.150
## 10516 0.195 0.159 0.038 0.304 0.304
## 51303 0.057 0.293 0.016 0.317 0.317
## 55056 0.219 0.091 0.144 0.273 0.273
```

```
corrplot(user.fanny.k5.opt$membership[1:10,], is.corr=F)
```




```
# fuzzy clustering - 14 clusters
param <- seq(1.1, 2, 0.1)
for (i in param){
  print(i)
  user.fanny.k14 = fanny(user_cluster_features, k=14, memb.exp = i)
}
```

```
## [1] 1.1
## [1] 1.2
```

```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
## [1] 1.3
```

```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
## [1] 1.4
```

```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
## [1] 1.5
```

```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
## [1] 1.6
```

```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
```

```
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
## [1] 1.7
```

```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
## [1] 1.8
```

```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
## [1] 1.9
```

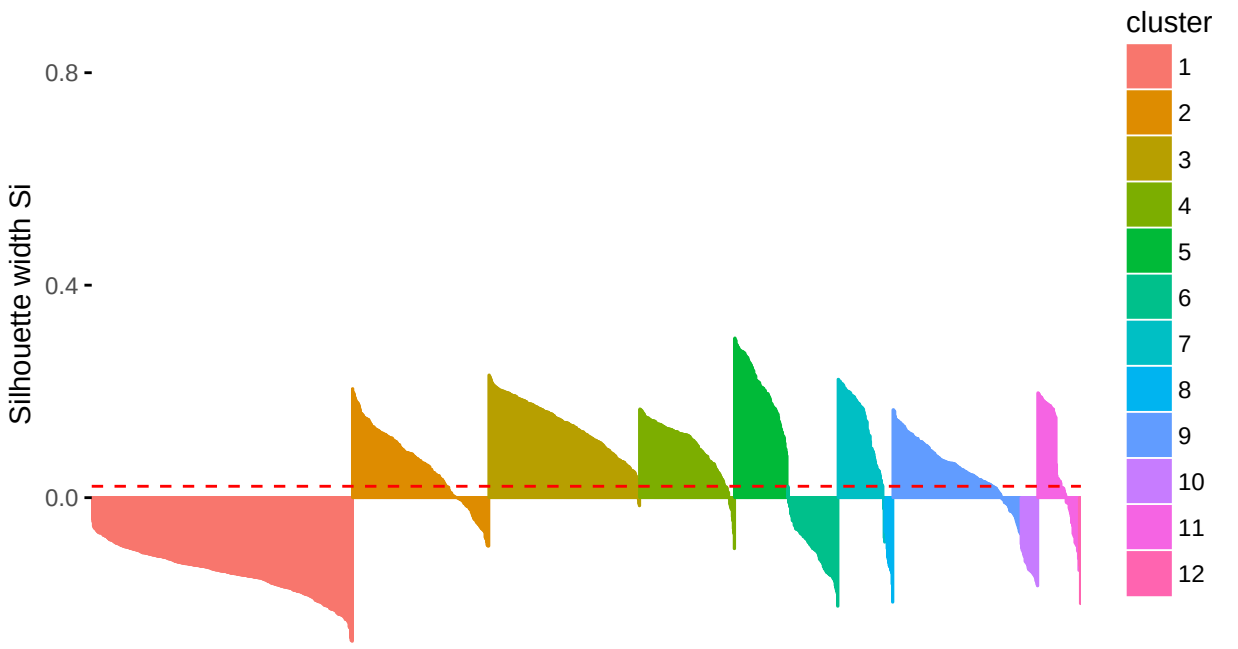
```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

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

```
## Warning in fanny(user_cluster_features, k = 14, memb.exp = i): the
## memberships are all very close to 1/k. Maybe decrease 'memb.exp' ?
```

```
user.fanny.k14.opt = fanny(user_cluster_features, k=14, memb.exp = 1.1)
fviz_silhouette(silhouette(user.fanny.k14.opt),
    main=paste0("Silhouette plot for FUZZY clustering - 14 clusters, optimal memb.exp = ", 1.1))
```

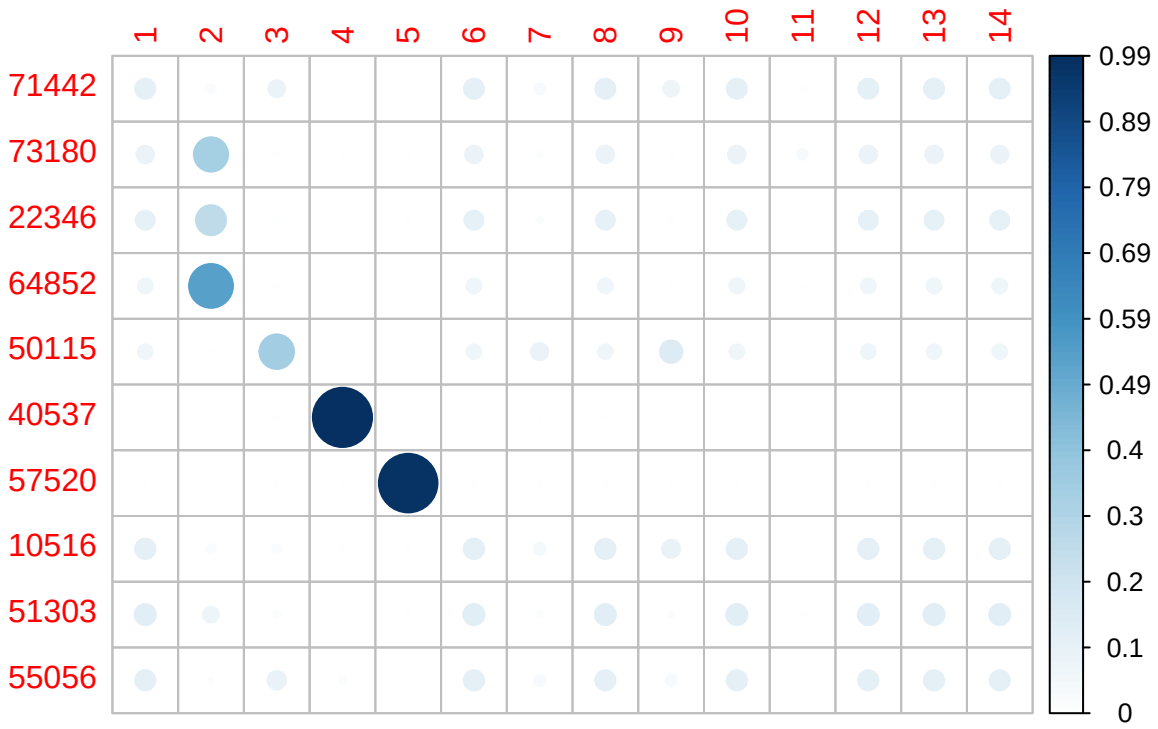
```
##    cluster size ave.sil.width
## 1        1  527         -0.14
## 2        2  276          0.06
## 3        3  305          0.14
## 4        4  192          0.10
## 5        5  107          0.21
## 6        6  102         -0.10
## 7        7   95          0.14
## 8        8   16         -0.12
## 9        9  259          0.06
## 10      10   34         -0.13
## 11      11   84          0.07
## 12      12    3         -0.15
```




```
# the membership probabilities for the first 10 observations in the sample
print(head(round(user.fanny.k14.opt$membership,3), 10))
```

```
##        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11]
## 71442 0.113 0.025 0.081 0.000 0.000 0.113 0.034 0.113 0.071 0.113 0.000
## 73180 0.087 0.332 0.007 0.001 0.003 0.087 0.011 0.087 0.006 0.087 0.031
## 22346 0.101 0.255 0.009 0.001 0.002 0.101 0.014 0.101 0.009 0.101 0.001
## 64852 0.064 0.542 0.004 0.000 0.001 0.064 0.003 0.064 0.002 0.064 0.002
## 50115 0.062 0.002 0.337 0.002 0.001 0.062 0.085 0.062 0.142 0.062 0.000
## 40537 0.001 0.000 0.004 0.989 0.000 0.001 0.001 0.001 0.001 0.001 0.000
## 57520 0.003 0.000 0.003 0.003 0.969 0.003 0.002 0.003 0.002 0.003 0.000
## 10516 0.116 0.024 0.023 0.004 0.005 0.116 0.040 0.116 0.092 0.116 0.001
## 51303 0.125 0.070 0.018 0.001 0.005 0.125 0.017 0.125 0.011 0.125 0.005
## 55056 0.115 0.011 0.097 0.017 0.003 0.115 0.035 0.115 0.034 0.115 0.001
##       [,12] [,13] [,14]
## 71442 0.113 0.113 0.113
## 73180 0.087 0.087 0.087
## 22346 0.101 0.101 0.101
## 64852 0.064 0.064 0.064
## 50115 0.062 0.062 0.062
## 40537 0.001 0.001 0.001
## 57520 0.003 0.003 0.003
## 10516 0.116 0.116 0.116
## 51303 0.125 0.125 0.125
## 55056 0.115 0.115 0.115
```

```
corrplot(user.fanny.k14.opt$membership[1:10,], is.corr=F)
```




```
# Gaussian mixture model on the scaled features
user.mc = Mclust(user_cluster_features)
print(summary(user.mc))
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEV (ellipsoidal, equal shape) model with 8 components:
##
##  log.likelihood    n   df       BIC       ICL
##       -57832.37 2000 5378 -156542.4 -156542.5
##
## Clustering table:
##   1   2   3   4   5   6   7   8
## 733  93 584 219 120  96 118  37
```

```
# optimal number of clusters
print(paste0("Optimal number of clusters by Mclust = ", user.mc$G))
```

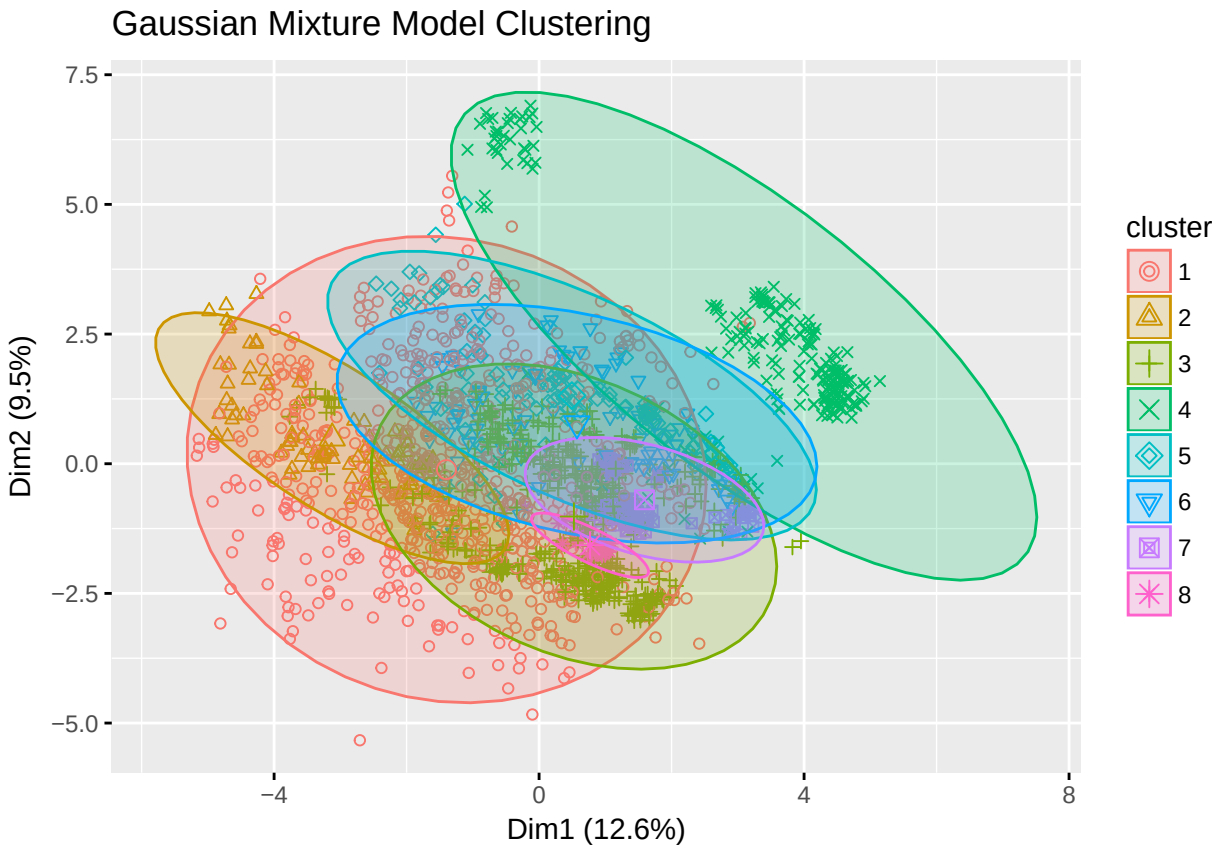
```
## [1] "Optimal number of clusters by Mclust = 8"
```

```
# the membership probabilities for the first 10 observations in the sample
print(head(round(user.mc$z, 3), 10))
```

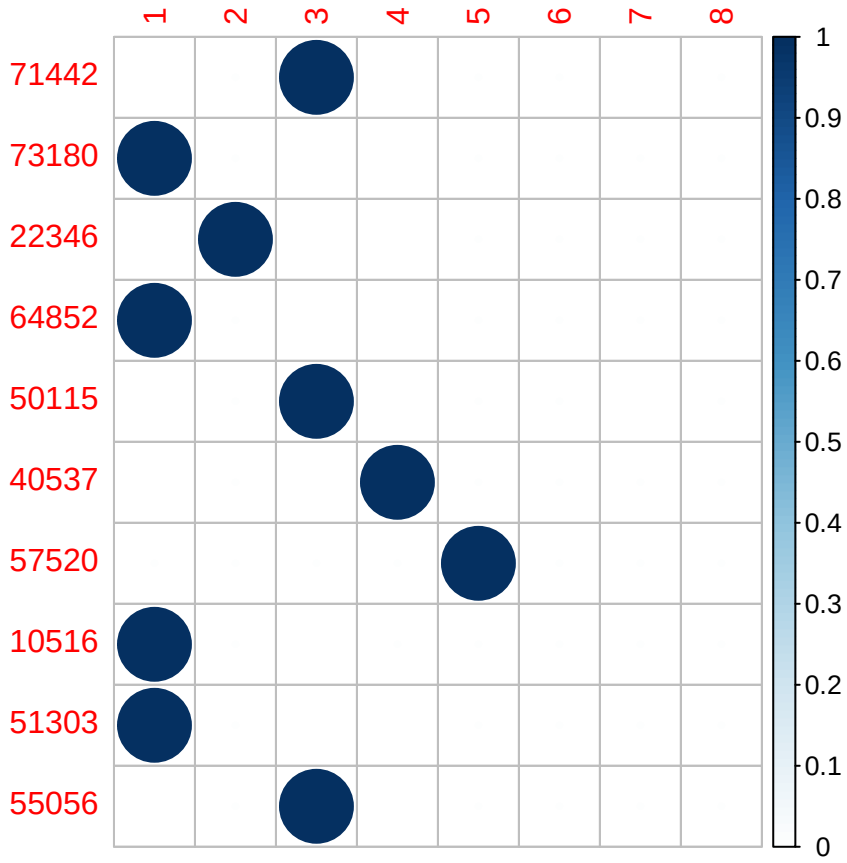
```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## 71442    0    0    1    0    0    0    0    0
## 73180    1    0    0    0    0    0    0    0
## 22346    0    1    0    0    0    0    0    0
## 64852    1    0    0    0    0    0    0    0
## 50115    0    0    1    0    0    0    0    0
## 40537    0    0    0    1    0    0    0    0
## 57520    0    0    0    0    1    0    0    0
```

```
## 10516    1    0    0    0    0    0    0    0
## 51303    1    0    0    0    0    0    0    0
## 55056    0    0    1    0    0    0    0    0
```

```
fviz_cluster(user.mc, ellipse.type='norm', geom="point") +
  ggtitle("Gaussian Mixture Model Clustering")
```




```
corrplot(user.mc$z[1:10,], is.corr=F)
```



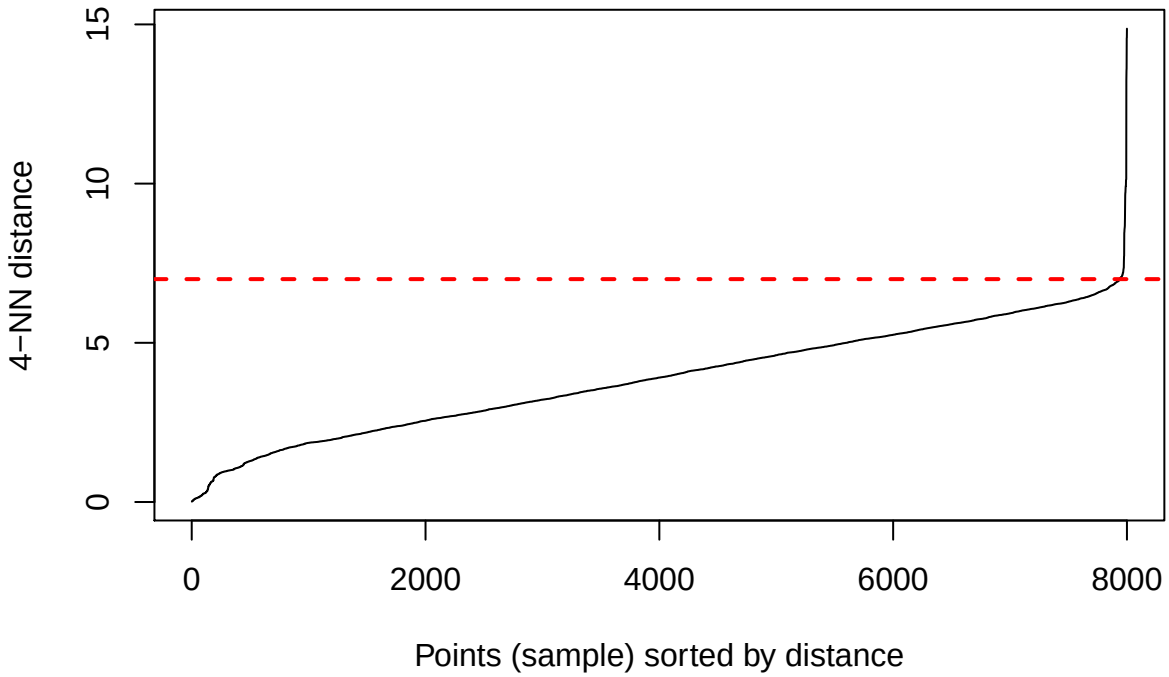
ANSWER (4. Other Clustering Algorithms (b)):
To find the optimal 'memb.exp', we tried values between 1.1 and 2 (step by 0.1). The only value that did not give the error "the memberships are all very close to 1/k" was 1.1. The 'memb.exp' parameter indicates membership ambiguity between clusters. Values larger than 1.1 appear to be too fuzzy to pick out the largest proabaility of belonging to certain group.

The optimal number of clusters by Mclust is 8. The membership probability of the Gaussian mixture model shows thaat the model has very high confidence in the clustering results.

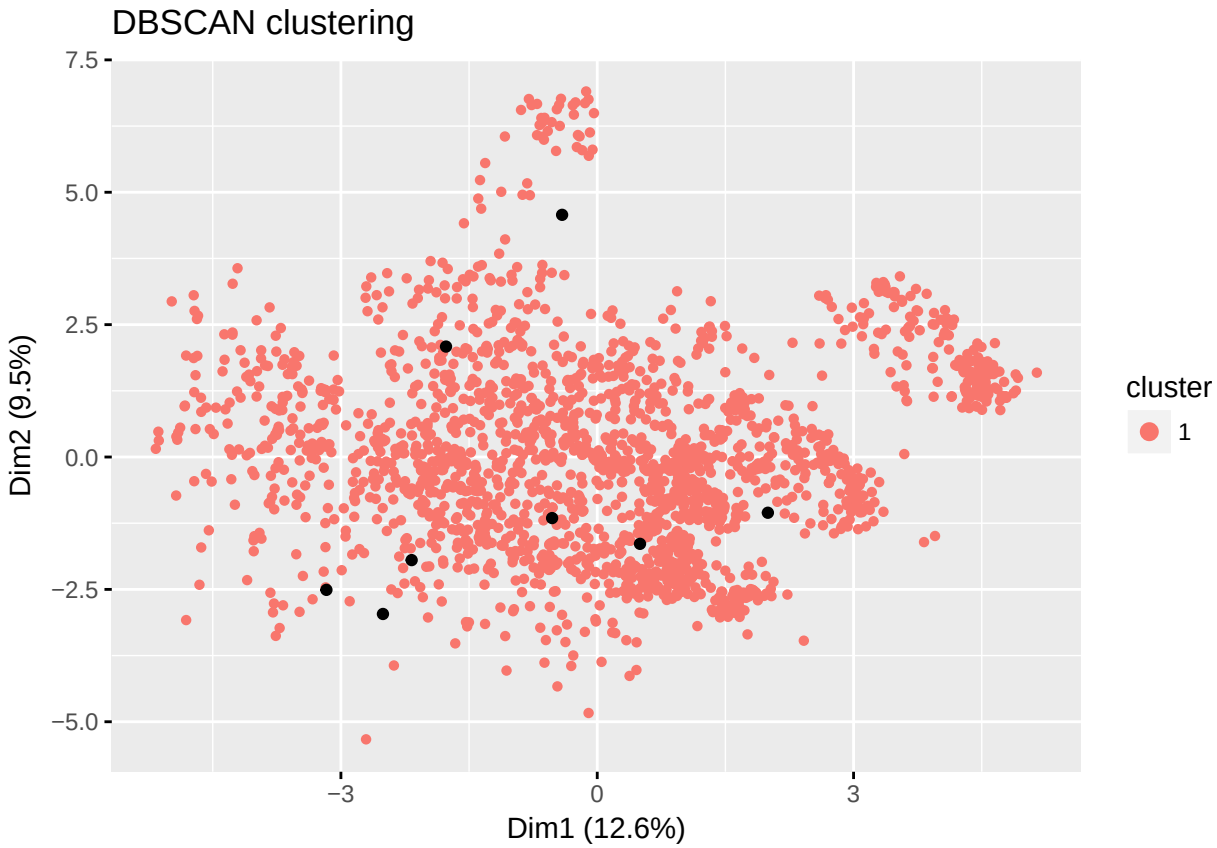
(c) (AC 209b students only) Density-based clustering: Apply DBSCAN to the data. Determine a reasonable combination of ϵ, the radius of the neighborhood around an observation, and the number of nearest neighbors within the ϵ-neighborhood to be considered a core point. You should construct a knee plot to determine the choice of ϵ. Summarize the results using a principal components plot, and comment on the clusters and outliers identified. How does the clustering produced by DBSCAN compare to the previous methods? Read Section 2.2 of the R vignette on DBSCAN

https://cran.r-project.org/web/packages/dbscan/vignettes/dbscan.pdf to learn about the OPTICS algorithm.

```
# knee plot
kNNdistplot(scale(user_cluster_features))
abline(7, 0, lty=2, lwd=2, col="red")
```




```
user.db = dbscan(user_cluster_features, eps=7, minPts=5)
fviz_cluster(user.db, user_cluster_features, ellipse=F, geom = "point") +
  ggtitle("DBSCAN clustering")
```



ANSWER (4. Other Clustering Algorithms (209 Question)

Based on the knee plot, we selected $\epsilon = 7$. Density-based clustering gives one cluster. It gives very different

clustering result compared to the previous clustering algorithms. K-means and hierarchical clustering find clusters by minimizing data variance within pre-specified number of clusters. Gaussian mixture model assumes the data comes from some distribution of convex shapes. By contrast, density-based clustering does not assume parametric distributions or number of clusters before clustering. It is more capable of finding arbitrarily-shaped clusters without prior knowledge of the number of clusters.

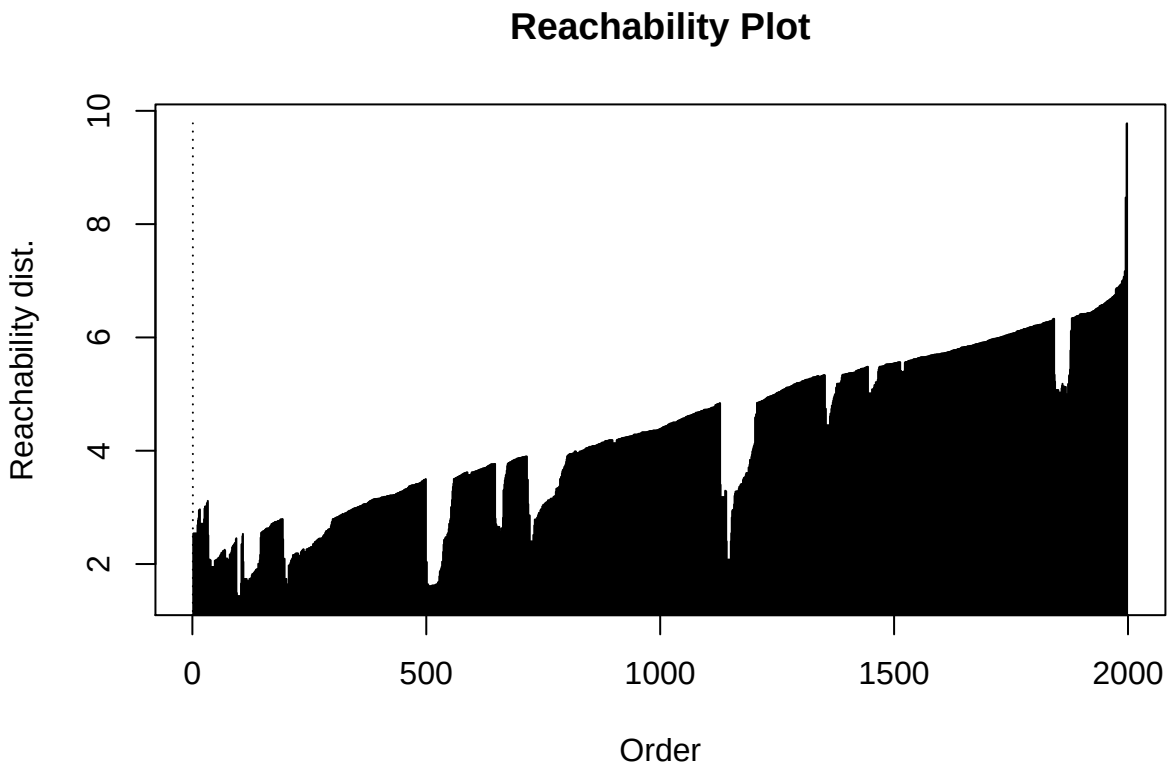
1. Describe the difference in goal between the DBSCAN and OPTICS algorithm. You may need to refer to the references cited within. **ANSWER (4. Other Clustering Algorithms (209 Question) - 1.:** DBSCAN has only one value for the ϵ parameter, which is a measure of density neighborhood size. It detects clusters consisting of data points that are at most ϵ distant. With the single valued neighborhood size, DBSCAN cannot find clusters of varying densities.

   By contrast, OPTICS's ϵ does not represent a single value. Instead, it is the upper threshold for different ϵ's applied to different clusters. Therefore, OPTICS can detect clusters of varying densities. By exploring neighbors of each point along its core- and reachability-distance (from lowest to highest), OPTICS outputs a sequence of data points with their reachability-distances. Low reachability-distances shown as valleys represent clusters separated by peaks represent- ing points with larger distances.

2. Run the OPTICS algorithm on the data within the dbscan package. Choose (and justify) an appropriate value of ϵ and the minimum number of points in the ϵ-neighborhood. Interpret the results of the clustering.

Hint: Make sure to also use and plot `extractXi()` with parameter `xi=0.5` to properly visualize your results. See documentation suggested above.

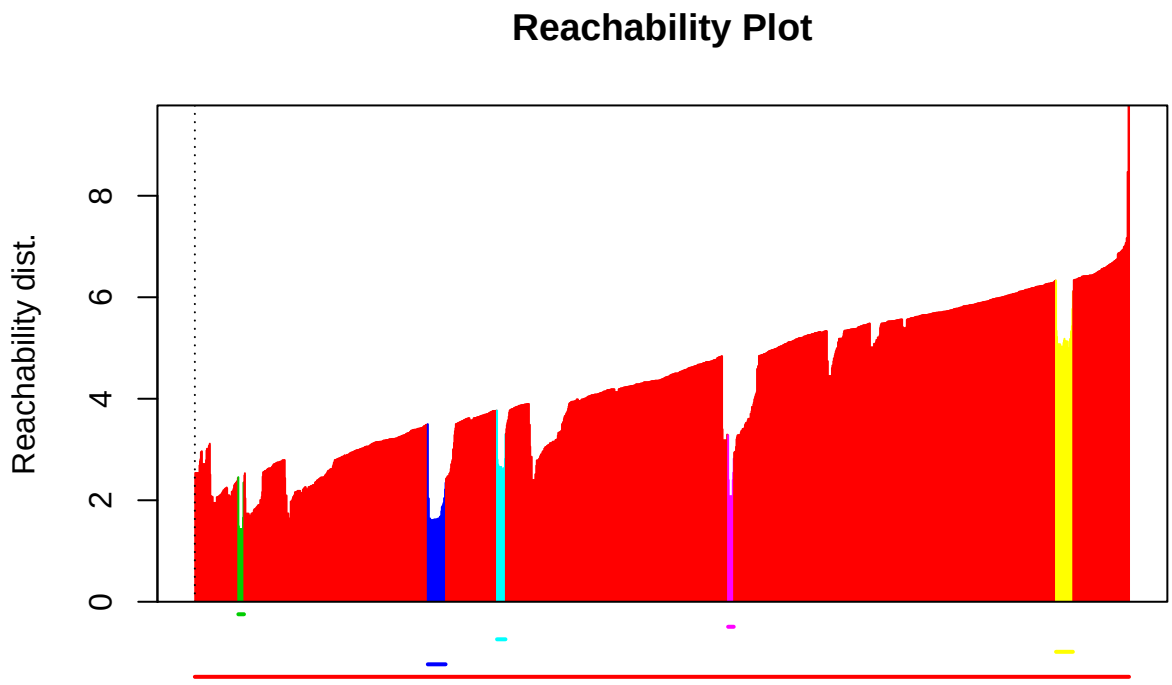
```
user.op <- optics(user_cluster_features, eps = 10, minPts = 10)
plot(user.op)
```




```
user.opXi <- extractXi(user.op, xi = 0.05)
user.opXi # result of extracted clusters from optics
```

```
## OPTICS ordering/clustering for 2000 objects.
## Parameters: minPts = 10, eps = 10, eps_cl = NA, xi = 0.05
## The clustering contains 6 cluster(s) and 2 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts,
##                   eps, eps_cl, xi, clusters_xi, cluster
```

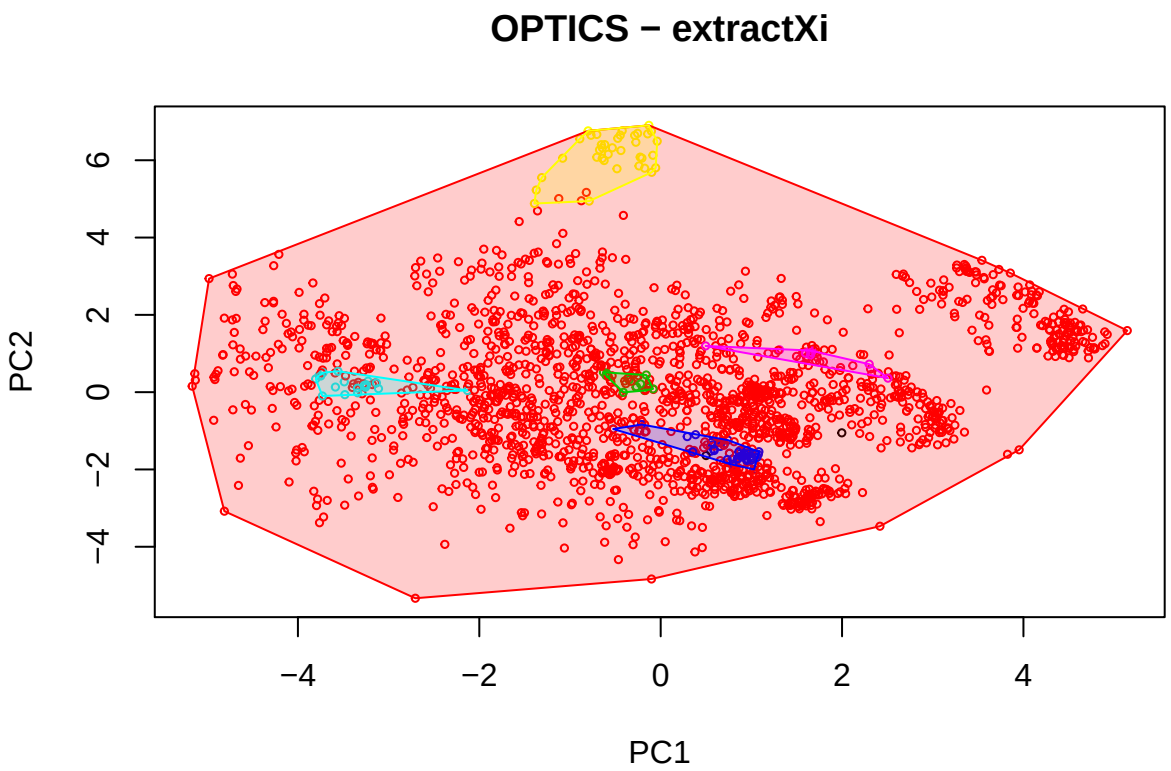
```
plot(user.opXi)
```




```
user.opXi$clusters_xi # cluster id's
```

```
##   start  end cluster_id
## 1     1 1998          1
## 2    94  106          2
## 3   499  537          3
## 4   647  665          4
## 5  1141 1153          5
## 6  1842 1878          6
```

```
hullplot(scale(user_cluster_features), user.opXi, main = "OPTICS - extractXi")
```



ANSWER (4. Other Clustering Algorithms (209 Question) - 2.:
OPTICS gives 6 clusters. Based on the clusters visualization, data points that do not cluster on the principle component plot could be denstiy reacheable/connected. This suggests that the largest part of the data variance could be explained by posture difference. At the same time, different users may have different habits of giving the same postures, there is additional data variance caused by differenct users.