Homework 5 - PCA, SVM & Clustering

Harvard CS109B, Spring 2017

Paul Harris

Mar 2017

Problem 1: Face recoginition

In this problem, the task is to build a facial recognition system using Principal Components Analysis (PCA) and a Support Vector Machine (SVM). We provide you with a collection of grayscale face images of three political personalities "George W. Bush", "Hugo Chavez" and "Ariel Sharon", divided into training and test data sets. Each face image is of size \(250 \times 250\), and is flattened into a vector of length 62500. All the data for this problem is located in the file `CS109b-hw5-dataset_1.Rdata`. You can read this file using the `load()` function, which will load four new variables into your environment. The vectorized images are available as rows in the arrays `imgs_train` and `imgs_test`. The identity of the person in each image is provided in the vectors `labels_train` and `labels_test`. The goal is to fit a face detection model to the training set, and evaluate its classification accuracy (i.e. fraction of face images which were recognized correctly) on the test set.

One way to perform face recognition is to treat each pixel in an image as a predictor, and fit a classifier to predict the identity of the person in the image. Do you foresee a problem with this approach?

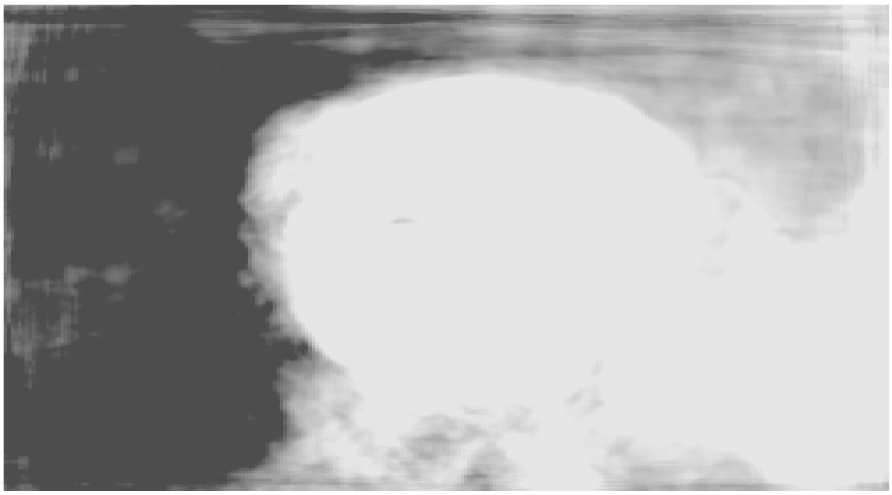
Instead we recommend working with low-dimensional representations of the face images computed using PCA. This can be done by calculating the top \(K\) principal components (PCs) for the vectorized face images in the training set, projecting each training and test image onto the space spanned by the PC vectors, and represent each image using the \(K\) projected scores. The PC scores then serve as predictors for fitting a classification model. Why might this approach of fitting a classification model to lower dimensional representations of the images be more beneficial?

The following function takes a vectorized version of an image and plots the image in its original form:

```
rot90 <- function(x, n = 1){
  #Rotates 90 degrees (counterclockwise)
  r90 <- function(x){
    y <- matrix(rep(NA, prod(dim(x))), nrow = nrow(x))
    for(i in seq_len(nrow(x))) y[, i] <- rev(x[i, ])
    y
  }
  for(i in seq_len(n)) x <- r90(x)
  return(x)
}
plot.face = function(x,zlim=c(-1,1)) {
  #Plots Face given image vector x
  x = pmin(pmax(x,zlim[1]),zlim[2])
  cols = gray.colors(100)[100:1]
  image(rot90(matrix(x,nrow=250)[,250:1],3),col=cols,
        zlim=zlim,axes=FALSE)
}
```

- Apply PCA to the face images in `imgs_train`, and identify the top 5 principal components. Each PC has the same dimensions as a vectorized face image in the training set, and can be reshaped into a 250 x 250 image, referred to as an *Eigenface*. Use the code above to visualize the Eigenfaces, and comment on what they convey. (*Hint*: for better visualization, we recommend that you re-scale the PC vectors before applying the above code; e.g. multiplying the PC vectors by 500 results in good visualization)

```
# Apply pca
faces.pca = prcomp(imgs_train, scale=TRUE)
#summary(faces.pca)

#Find the first five principal components
PC5 <- faces.pca$rotation[ ,1:5]
PC5.rescaled = 500*PC5

# PLot the eigenfaces
for(i in 1:5){
  plot.face(PC5.rescaled[,i])
}
```



#The eigenfaceas convey the

amount of condensed pixel information about a face for these top 5 principal components.

- Retain the top PCs that contribute to 90% of the variation in the training data. How does the number of identified PCs compare with the total number of pixels in an image? Compute the PC scores for each image in the training and test set, by projecting it onto the space spanned by the PC vectors.

```
#Find the PCs that contribute to 90% of the variation.
#Could do this programmatically... but let's just look at he summary
#summary(faces.pca)
#109 PCs accounts for 90% of the variation

#Compute the PC scores for each image in the training and test set
#For this problem, I could multiply the matrices, but this info is already contained in faces.pca$x
#TRaining
#faces.pca$x[,1:109]

#But for good form lets do it anyway
PC109 <- faces.pca$rotation[ ,1:109]
train_pca_scores = imgs_train%*%PC109
#Test
#We actually have to do this for the test set :/
test_pca_scores = imgs_test%*%PC109
```

The number of identified PCs increase with the total number of pixels in an image.

- Treating the PC scores as predictors, fit a SVM model to the the training set, and report the classification accuracy of the model on the test set. How does the accuracy of the fitted model compare to a naÃ¯ve classifier that predicts a random label for each image?

```
#For the svm
labels_train = factor(labels_train)
labels_test = factor(labels_test)
model1_svm = svm(x=train_pca_scores, y=labels_train, kernal="linear", gamma=.001, cost=.1)


#Tune for best cost and gamma
svm_tune_linear <- tune(svm, train.x=train_pca_scores, train.y=labels_train,
             kernel="linear",tunecontrol = tune.control(sampling = "cross", cross = 5),ranges=list(cost=
c(.1,1,5,10,20,30), gamma=c(.001,.01,.1,.5,1,2,2.5)))

print(svm_tune_linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 5-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.1 0.001
##
## - best performance: 0.1300702
```

```
#Best cost = .1, gamma = .001

test_preds_linear = predict(svm_tune_linear$best.model,test_pca_scores)
linear_accuracy = confusionMatrix(test_preds_linear, labels_test)$overall[1]
print("Svm classification accuracy")
```

```
## [1] "Svm classification accuracy"
```

```
linear_accuracy
```

```
##  Accuracy
## 0.9056047
```

```
#for the naive
naive = sample(unique(labels_test), length(labels_test), replace = TRUE)
naive_accuracy = sum(naive == labels_test)/length(labels_test)
print("naive classification accuracy")
```

```
## [1] "naive classification accuracy"
```

```
naive_accuracy
```

```
## [1] 0.3067847
```

Hint: You may use the function `prcomp` to perform PCA and `pr$rotation` attribute to obtain the loading vectors. The variance captured by each principal component can be computed using the `pr$sdev` attribute.

Problem 2: Analyzing Voting Patterns of US States

In this problem, we shall use unsupervised learning techniques to analyze voting patterns of US states in six presidential elections. The data set for the problem is provided in the file `CS109b-hw5-dataset_2.txt`. Each row represents a state in the US, and contains the logit of the relative fraction of votes cast by the states for Democratic presidential candidates (against the Republican candidates) in elections from 1960 to 1980. The logit transformation was used to expand the scale of proportions (which stay between 0 and 1) to an unrestricted scale which has more reliable behavior when finding pairwise Euclidean distances. Each state is therefore described by 6 features (years). The goal is to find subgroups of states with similar voting patterns.

You will need the `cluster`, `factoextra`, `mclust`, `corrplot`, `dbscan`, `MASS`, `ggplot2`, `ggfortify` and `NbClust` libraries for this problem.

```
## Package 'mclust' version 5.2.2
```

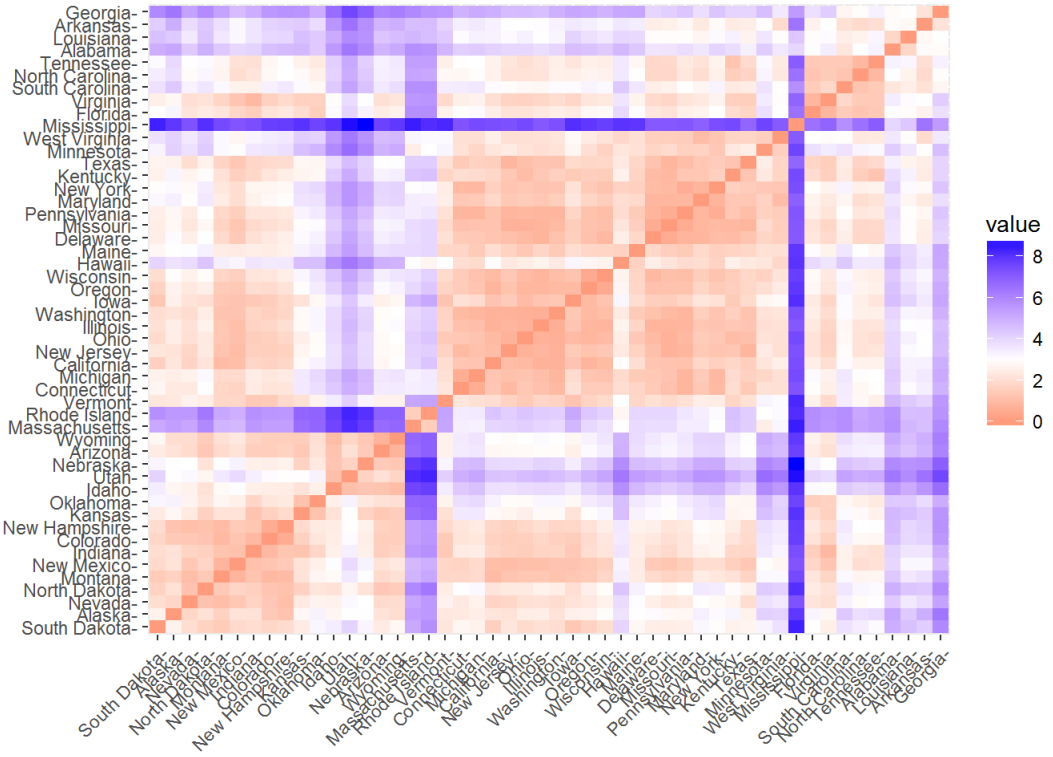
```
## Type 'citation("mclust")' for citing this R package in publications.
```

Part 2a: Visualize the data

Generate the following visualizations to analyze important characteristics of the data set:

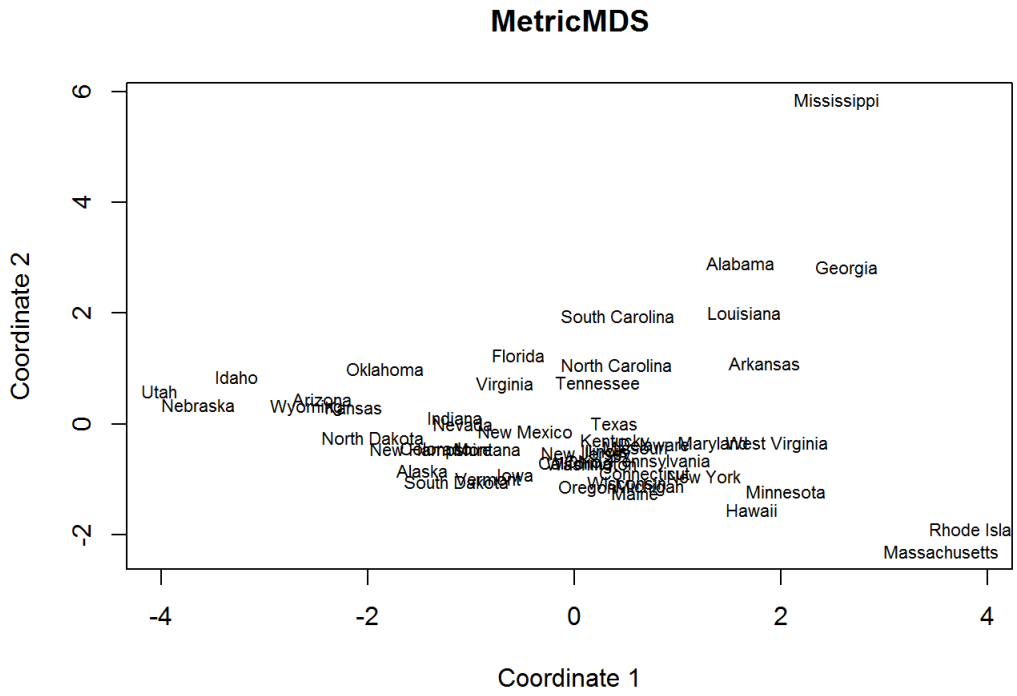
- Rescale the data, and compute the Euclidean distance between each pair of states. Generate a heat map of the pair-wise distances (*Hint:* use the `daisy` and `fviz_dist` functions).

```
#rescale the data
states_rescaled = scale(states)
#compute the euclidean distance
d.states = daisy(states_rescaled, metric="euclidean")
#generate a heatmap of the pairwise distances
fviz_dist(d.states,order = TRUE, show_labels = TRUE, lab_size = NULL,
          gradient = list(low = "red", mid = "white", high = "blue"))
```



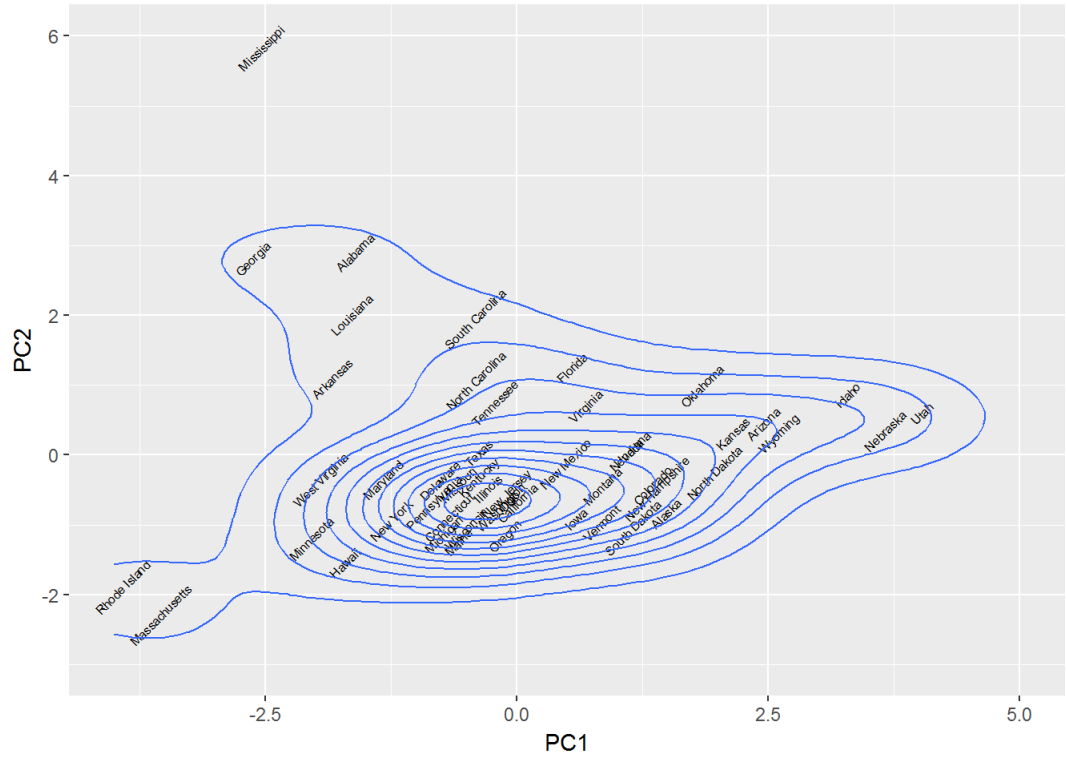
- Apply multi-dimensional scaling to the pair-wise distances, and generate a scatter plot of the states in two dimension (*Hint*: use the `cmdscale` function).

```
fit = cmdscale(d.states,eig=TRUE,k=2)
x = fit$points[,1]
y = fit$points[,2]
plot(x, y, xlab="Coordinate 1", ylab="Coordinate 2",
  main="Metric  MDS",   type="n")
text(x, y, labels = row.names(states), cex=.7)
```



- Apply PCA to the data, and generate a scatter plot of the states using the first two principal components (*Hint:* use the `prcomp` function). Add a 2d-density estimation overlay to the plot via the `geom_density2d` function.

```
#apply pca to the states
states.pca = prcomp(states_rescaled,scale=TRUE)

#plot a scatterplot of the states using the first two principal components
#Find the first two principal components
PC2_states <- states.pca$rotation[ ,1:2]
states_scores = states_rescaled%*%PC2_states

m = ggplot(states_scores, aes(x=PC1, y=PC2, label=rownames(states)))+
  geom_text(check_overlap = FALSE, size=2, angle=45) +
  xlim(-4,5) +
  ylim(-3,6)
m + geom_density2d()
```



Summarize the results of these visualizations. What can you say about the similarities and differences among the states with regard to

voting patterns? By visual inspection, into how many groups do the states cluster?

The result of these visualizations indicate the presence of variable voting patterns between states, but with the evidence of multiple, localized clusters. We see from the MDS and the PCA plots that MIssissippi displays the most different voting pattern between all of the states. We also see from the MDS plot that many states that are closer in region exhibit closer voting patterns to each other than to other states further in geographical dstance (e.g., closeness of north and south carolina, and Massachusetts and Rhode Island, yet those two groups being far from each other). Hard to say how many groups are necessary, but I would guess around 8-9.

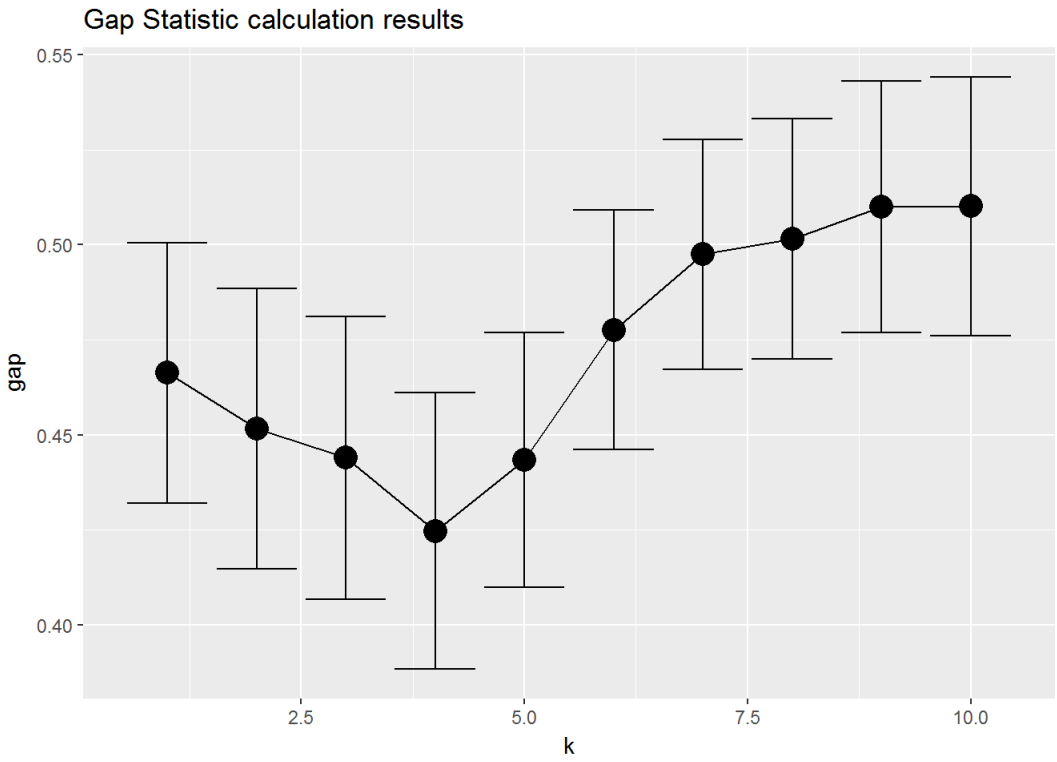
Part 2b: Partitioning clustering

Apply the following partitioning clustering algorithms to the data:

- **K-means clustering** (*Hint:* use the `kmeans` function)
- **Partitioning around medoids (PAM)** (*Hint:* use the `pam` function)

In each case, determine the optimal number of clusters based on the Gap statistic, considering 2 to 10 clusters (*Hint:* use the `clusGap` function). Also determine the choice of the optimal number of clusters by producing elbow plots (*Hint:* use `fviz_nbclust`). Finally, determine the optimal number of clusters using the method of average silhouette widths (*Hint:* use `fviz_nbclust` with argument `method="silhouette"`). Do the choices of these three methods agree? If not, why do you think you are obtaining different suggested numbers of clusters?

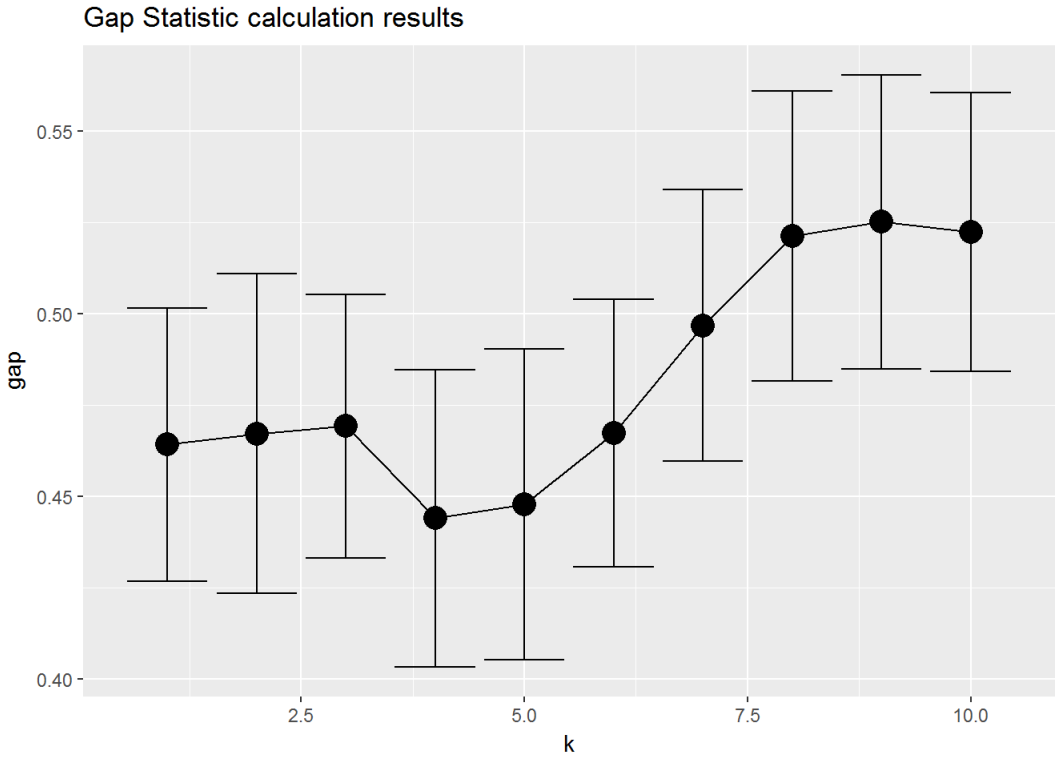
```
#By Method of Gap Statistic

#kmeans
gskmn <- clusGap(x=states_rescaled, FUN = kmeans, nstart = 20, K.max = 10, B = 60)

plot_clusgap = function(clusgap, title="Gap Statistic calculation results"){
    require("ggplot2")
    gstab = data.frame(clusgap$Tab, k=1:nrow(clusgap$Tab))
    p = ggplot(gstab, aes(k, gap)) + geom_line() + geom_point(size=5)
    p = p + geom_errorbar(aes(ymax=gap+SE.sim, ymin=gap-SE.sim))
    p = p + ggtitle(title)
    return(p)
}

plot_clusgap(gskmn)
```




```
print("GapStat Best k = 9")
```

```
## [1] "GapStat Best k = 9"
```

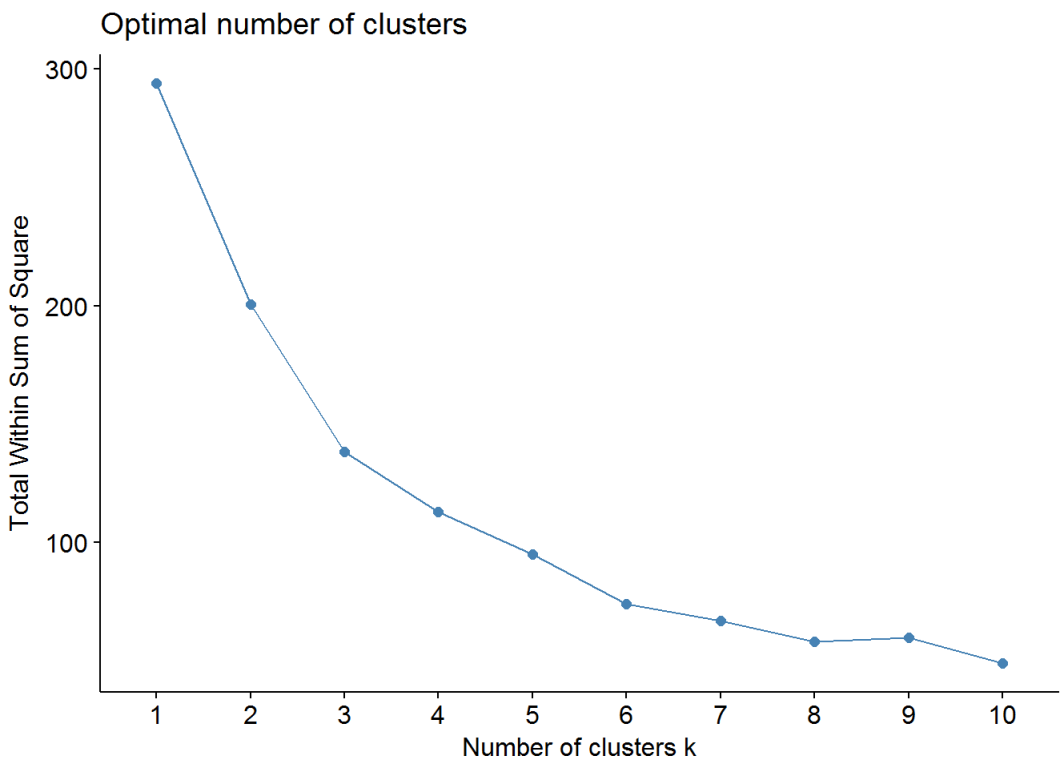
```
#pam
pam_fun = function(x,k){list(cluster = pam(x,k, cluster.only=TRUE))}
gspam = clusGap(x=states_rescaled, FUN=pam_fun, K.max=10, B=60)

plot_clusgap(gspam)
```




```
print("GapStat best k=9")
```

```
## [1] "GapStat best k=9"
```

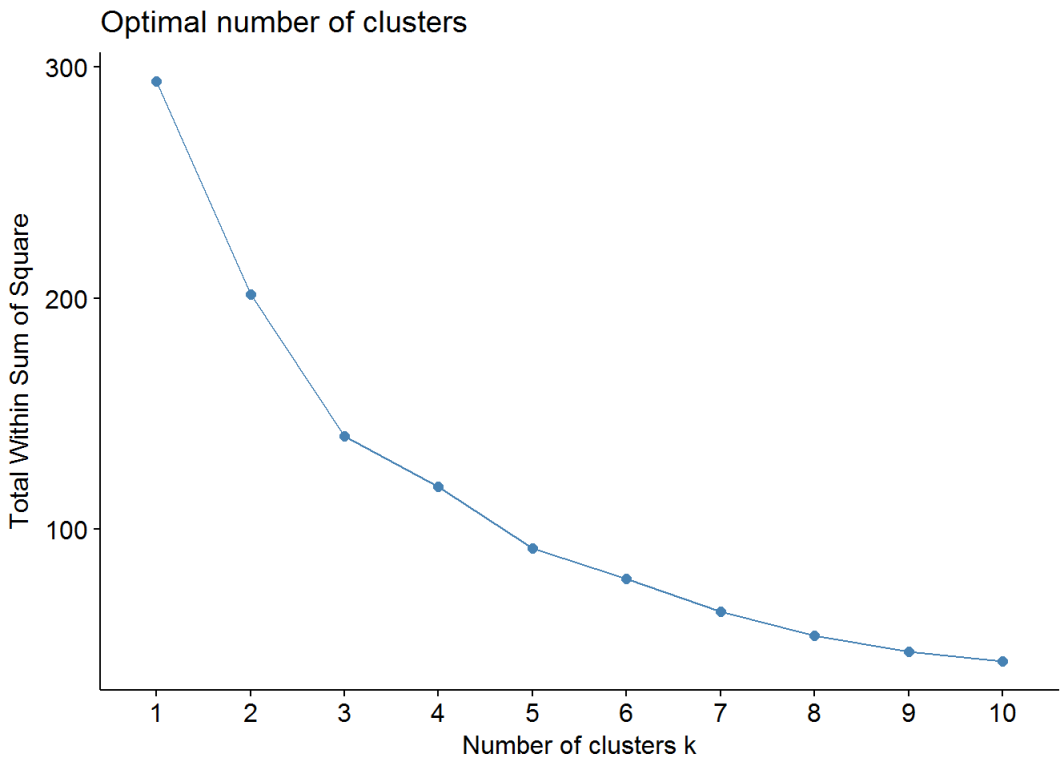
```
#By Elbow PLots

#kmeans
fviz_nbclust(states_rescaled, kmeans, method="wss")
```




```
print("best k = 6")
```

```
## [1] "best k = 6"
```

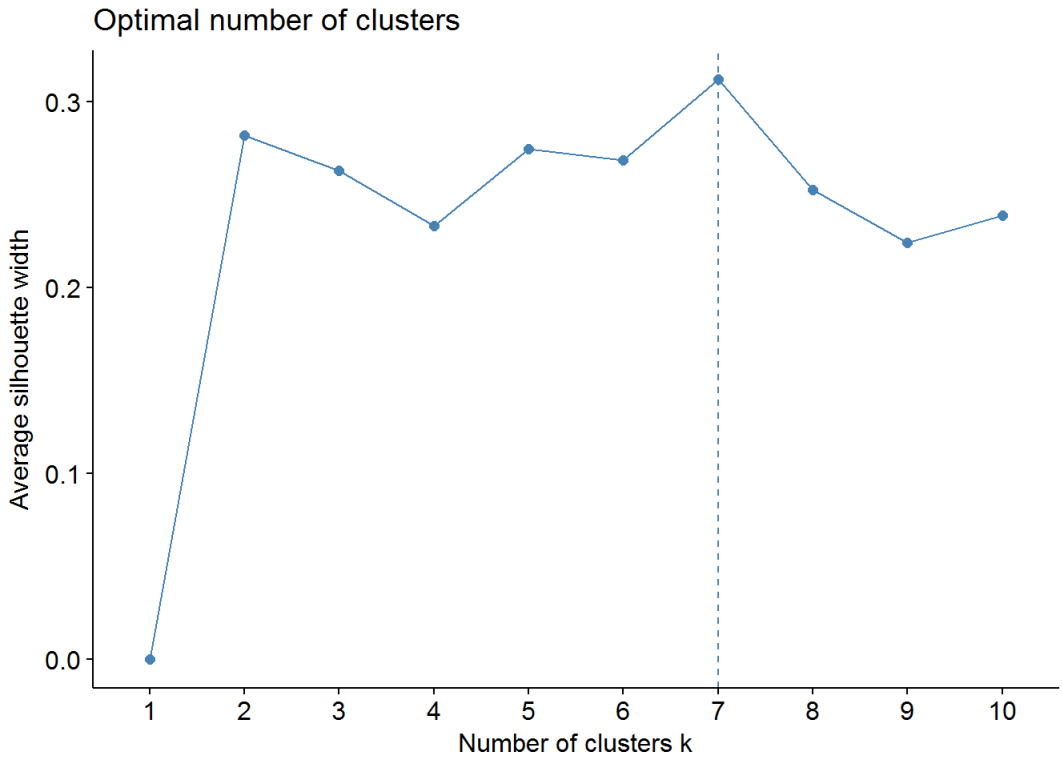
```
#pam
fviz_nbclust(states_rescaled,pam, method="wss")
```




```
print("bestk=7")
```

```
## [1] "bestk=7"
```

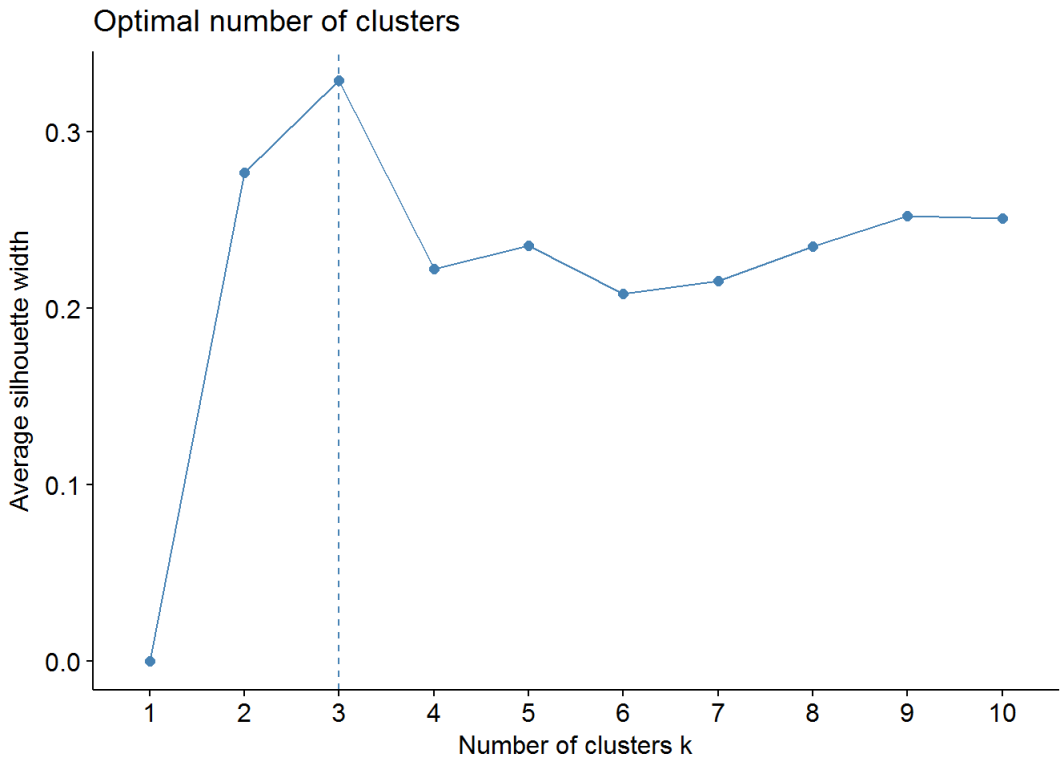
```
#By Method of average silhouette widths

#kmeans
fviz_nbclust(states_rescaled, kmeans, method="silhouette")
```




```
print("best k=7")
```

```
## [1] "best k=7"
```

```
#pam
fviz_nbclust(states_rescaled,pam, method="silhouette")
```



```
print("best k = 3")
```

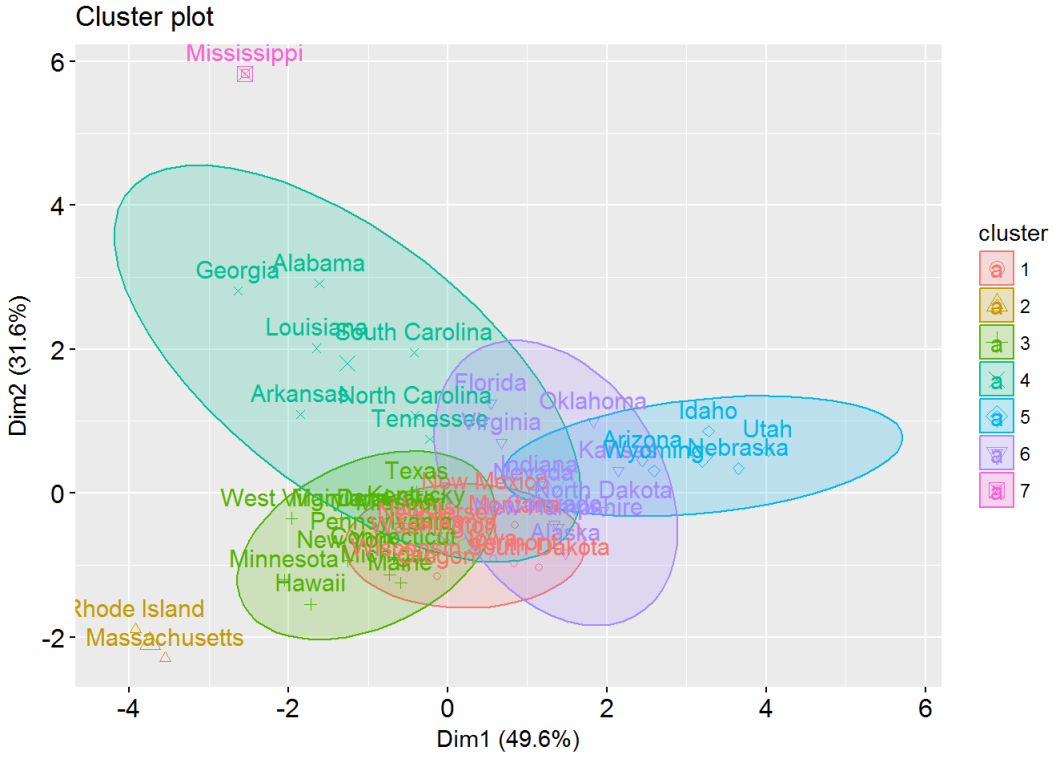
```
## [1] "best k = 3"
```

The number of clusters do not exactly agree. This may be because they are all using different evaluation metrics for the optimal choice of k. And these evaluation metrics penalize different things.

With your choice of the number of clusters, construct a principal components plot the clusters for *K-means* and *PAM* using the `fviz_cluster` function. Are the clusterings the same? Summarize the results of the clustering including any striking features of the clusterings.

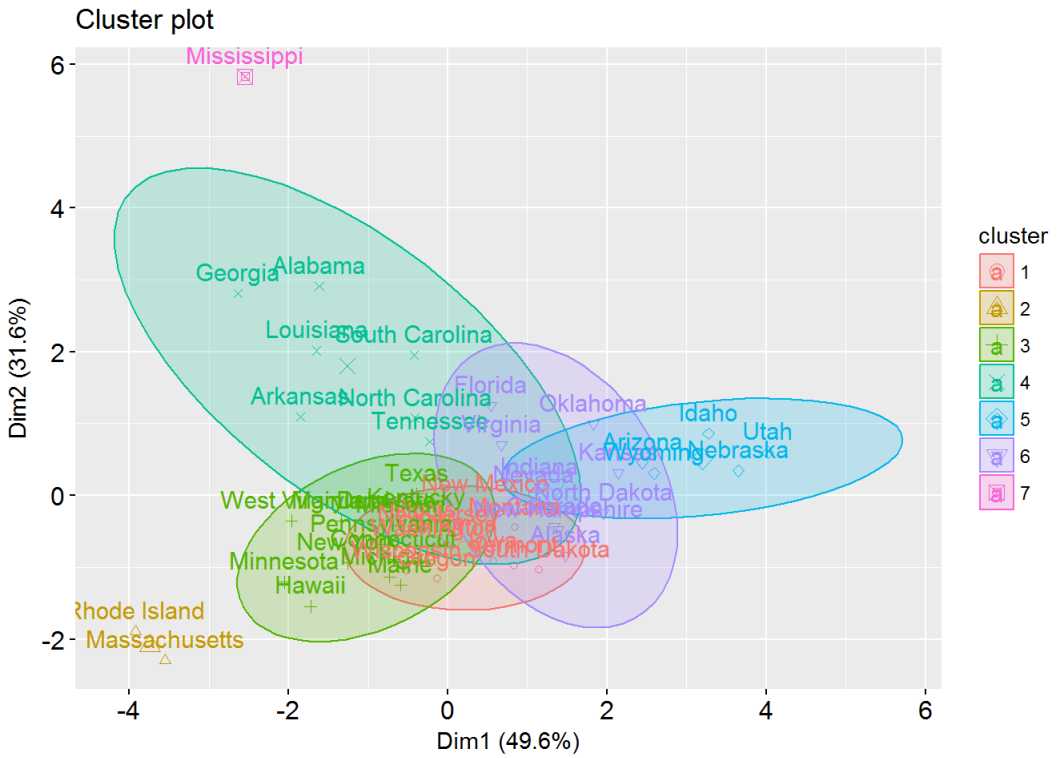
```
#kmeans
km.res <- kmeans(states_rescaled, 7, nstart = 10)
fviz_cluster(km.res, states_rescaled, ellipse.type = "norm")
```

```
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
```




```
#pam
pm.res <- pam(states_rescaled, 7)
fviz_cluster(km.res, states_rescaled, ellipse.type = "norm")
```

```
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
```

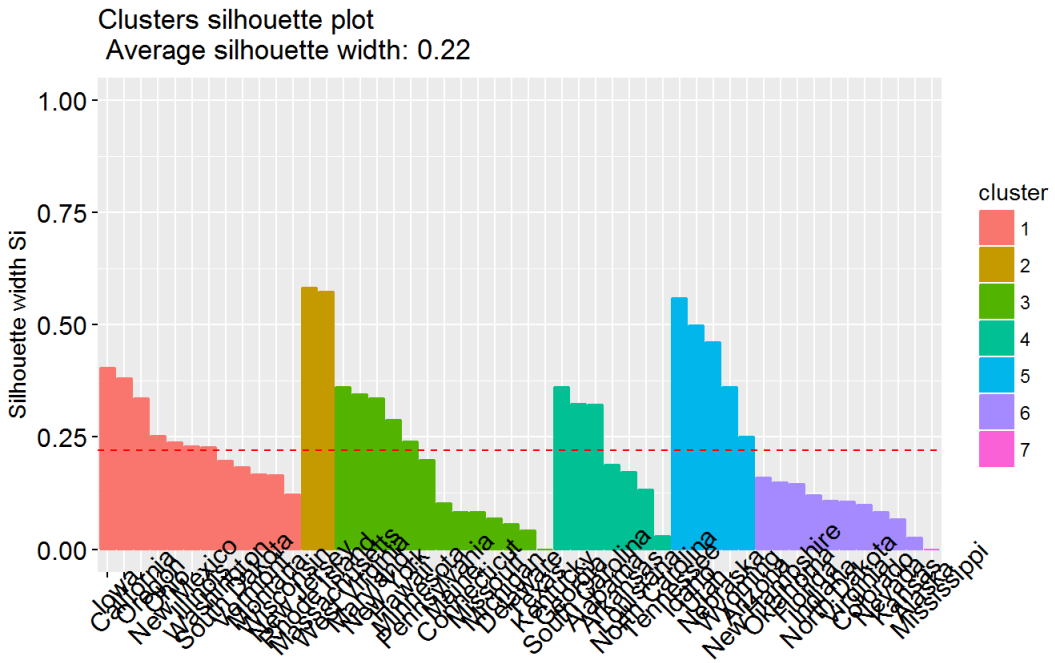


The clusters are the same in this case. The only thing I find striking is that generally, for the states that share cluster borders, I feel as if the assignment has the potential to be slightly arbitrary.

Generate silhouette plots for the *K-means* and *PAM* clusterings with the optimal number of clusters. Identify states that may have been placed in the wrong cluster (*Hint:* use the `fviz_silhouette` function).

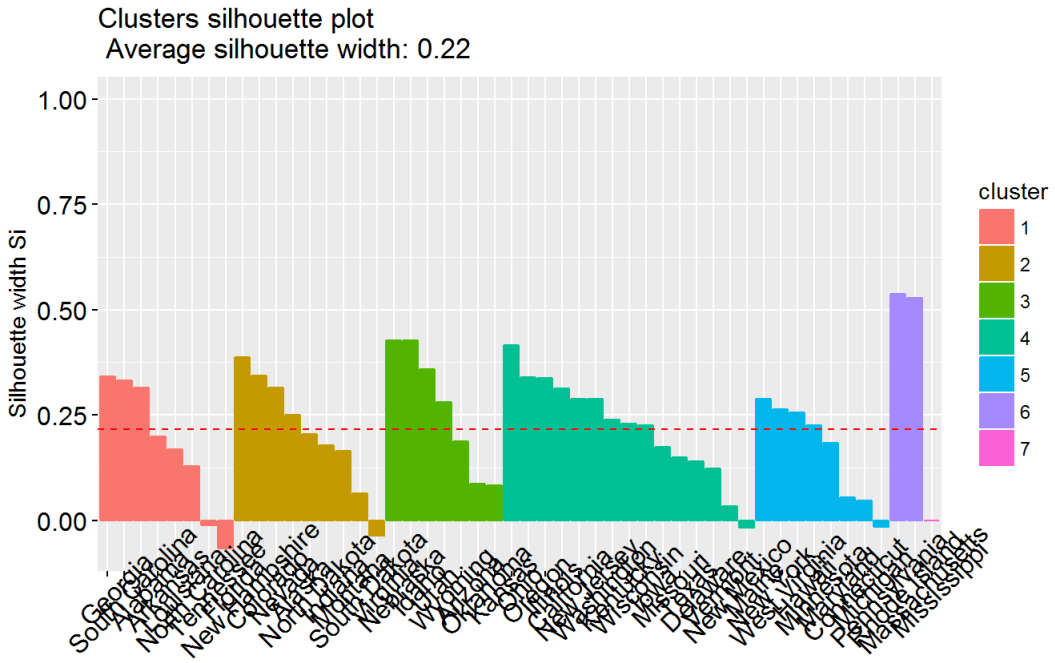
```
#kmeans
sil_k = silhouette(km.res$cluster, dist(states_rescaled))
rownames(sil_k) =rownames(states)
fviz_silhouette(sil_k,label=TRUE)
```

```
##   cluster size ave.sil.width
## 1       1   12          0.24
## 2       2    2          0.58
## 3       3   13          0.17
## 4       4    7          0.22
## 5       5    5          0.43
## 6       6   10          0.11
## 7       7    1          0.00
```




```
#pam
sil_p = silhouette(pm.res$cluster, dist(states_rescaled))
rownames(sil_p) =rownames(states)
fviz_silhouette(sil_p, label=TRUE)
```

```
##   cluster size ave.sil.width
## 1       1    8          0.18
## 2       2    9          0.21
## 3       3    7          0.26
## 4       4   15          0.22
## 5       5    8          0.16
## 6       6    2          0.53
## 7       7    1          0.00
```



According to the silhouette

plots, we see that clusters 3,4,and 7 have lots of observations near 0. the worst offenders seem to be Mississippi, Delaware, and Lousiana, among others.

Part 2c: Hierarchical clustering

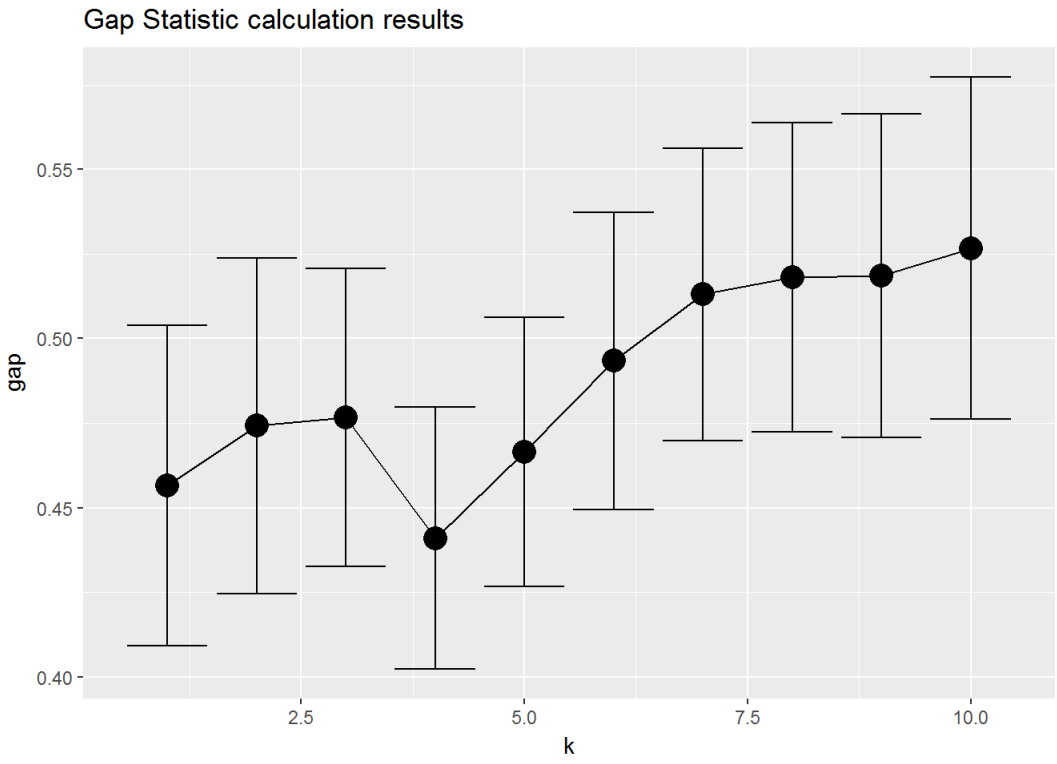
Apply the following hierarchical clustering algorithms to the data:

- **Agglomerative clustering** with Ward's method (*Hint*: use the `agnes` function)
- **Divisive clustering** (*Hint*: use the `diana` function)

In each case, summarize the results using a dendogram. (*Hint:* use the `pltree` function in the `cluster` library to plot the dendograms, and the `cutree` function to derive cluster groups from hierarchical clustering model). Determine the optimal number of clusters using Gap statistic, and add rectangles to the dendrograms sectioning off clusters (*Hint:* use `rect.hclust`). Do you find that states that predominantly vote for Republicans (e.g., Wyoming, Idaho, Alaska, Utah, Alabama) are closer together in the hierarchy? What can you say about states that usually lean towards Democrats (e.g. Maryland, New York, Vermont, California, Massachusetts)? Comment on the quality of clustering using Silhouette diagnostic plots.

```
agnes.reformat<-function(x, k){
# x: Data matrix or frame, k: Number of clusters
  x.agnes = agnes(x,method="ward",stand=T)
  x.cluster = list(cluster=cutree(x.agnes,k=k))
  return(x.cluster)
}

diana.reformat<-function(x, k){
# x: Data matrix or frame, k: Number of clusters
  x.diana = diana(x,stand=T)
  x.cluster = list(cluster=cutree(x.diana,k=k))
  return(x.cluster)
}
```

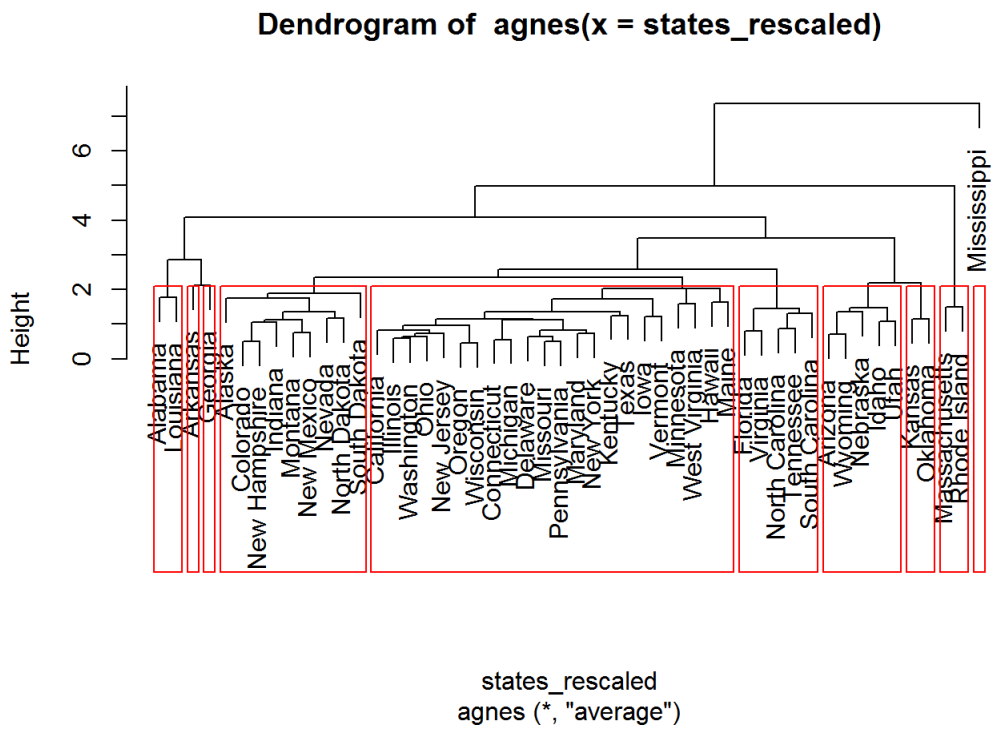
```
#agnes
gsagg <- clusGap(x=states_rescaled, FUN = agnes.reformat, K.max = 10, B = 60)
plot_clusgap(gsagg)
```




```
print("optimal number is 10")
```

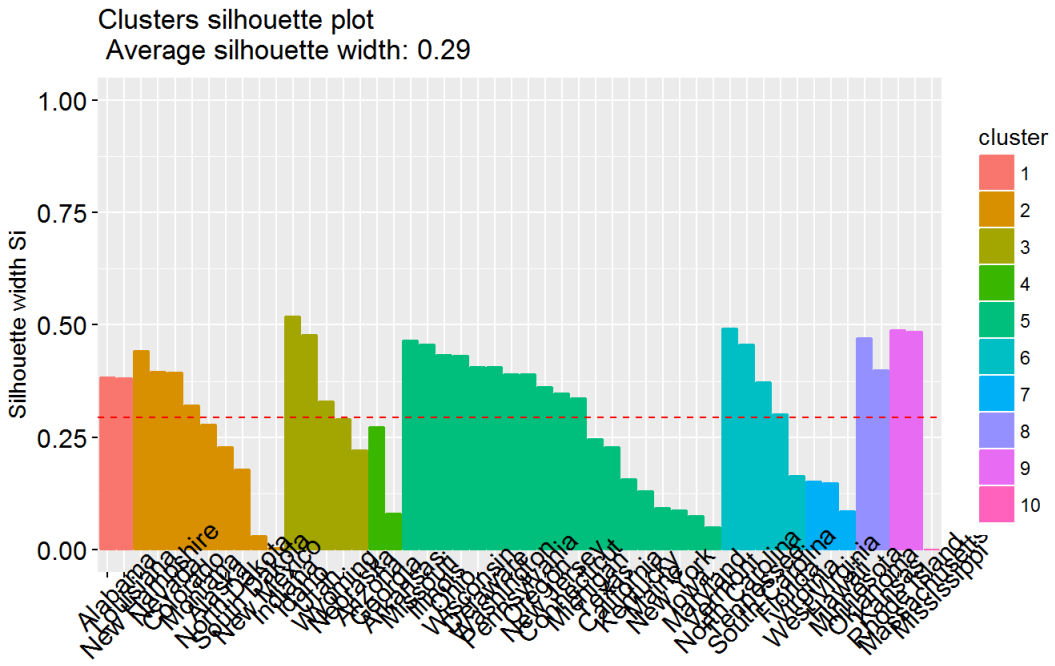
```
## [1] "optimal number is 10"
```

```
states.ag = agnes(states_rescaled)
pltree(states.ag)
rect.hclust(states.ag,k=10,border="red")
```

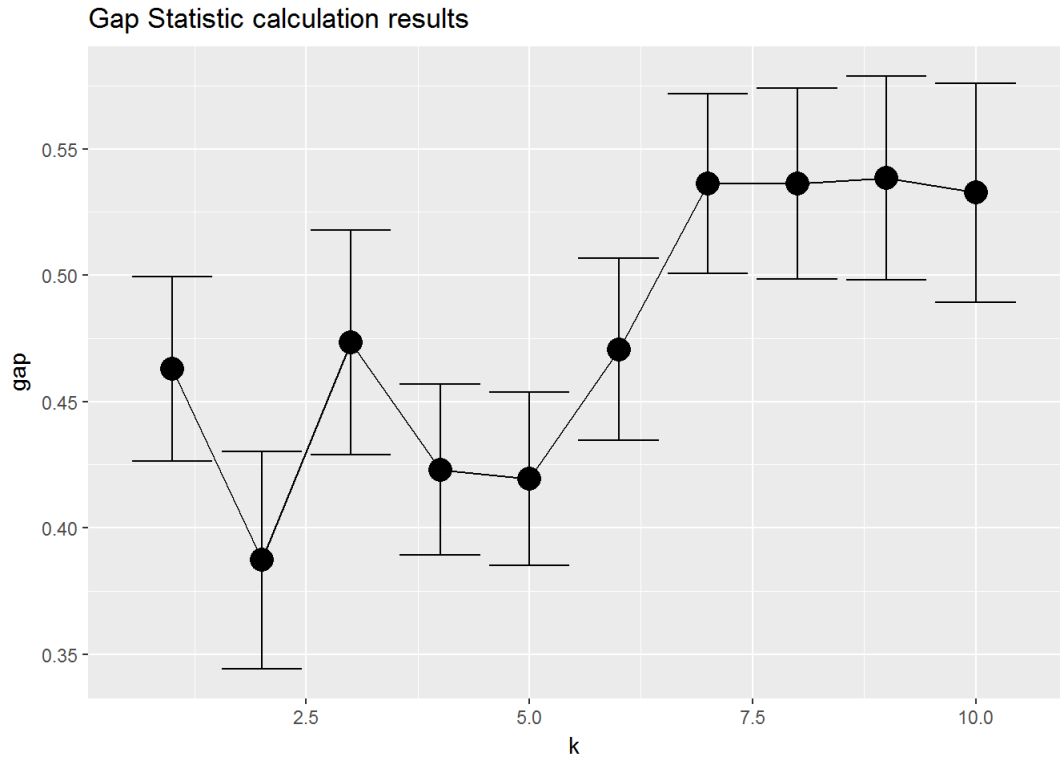



```
ck10 = cutree(states.ag,k=10)

agnes.res = agnes.reformat(states_rescaled,10)
sil_agg = silhouette(agnes.res$cluster, dist(states_rescaled))
rownames(sil_agg) =rownames(states)
fviz_silhouette(sil_agg,label=TRUE)
```

```
##    cluster size ave.sil.width
## 1        1    2          0.38
## 2        2    9          0.25
## 3        3    5          0.37
## 4        4    2          0.18
## 5        5   19          0.29
## 6        6    5          0.36
## 7        7    3          0.13
## 8        8    2          0.43
## 9        9    2          0.49
## 10      10    1          0.00
```



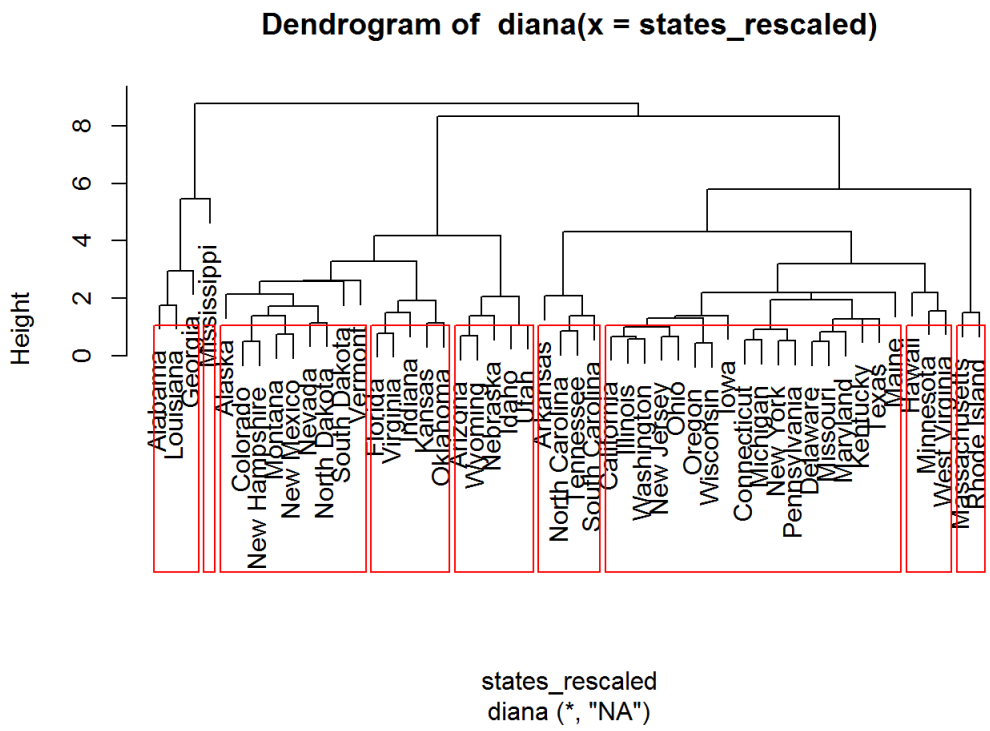

```
#diana
gsdia <- clusGap(x=states_rescaled, FUN = diana.reformat, K.max = 10, B = 60)
plot_clusgap(gsdia)
```




```
print("optimal number is 9")
```

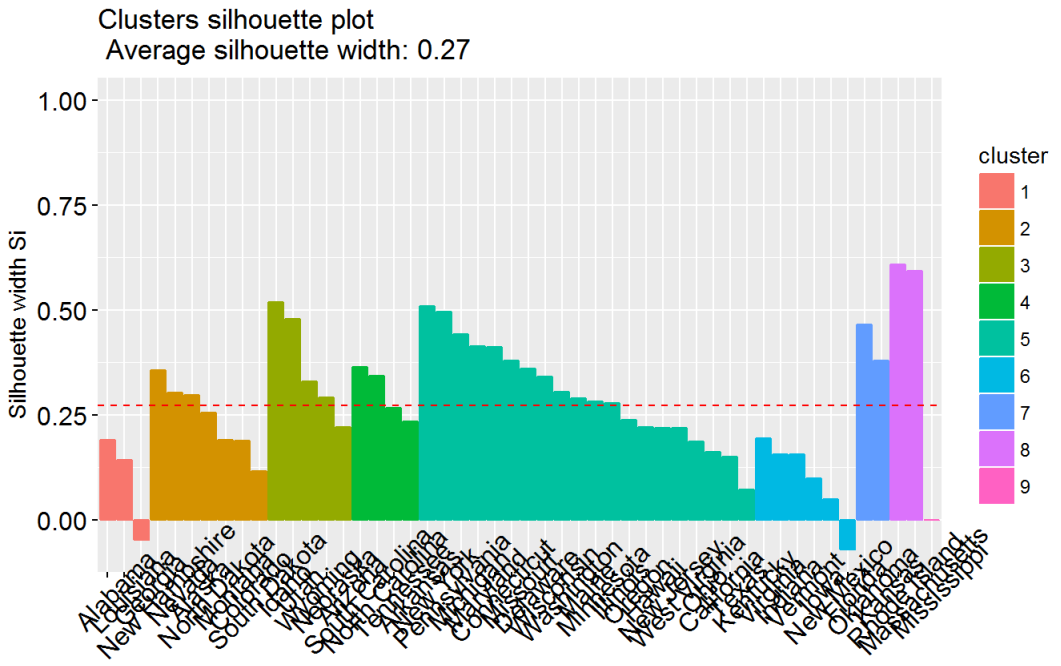
```
## [1] "optimal number is 9"
```

```
states.di = diana(states_rescaled)
pltree(states.di)
rect.hclust(states.di,k=9,border="red")
```




```
ck9 = cutree(states.di,k=9)

diana.res = diana.reformat(states_rescaled,9)
sil_dia = silhouette(diana.res$cluster, dist(states_rescaled))
rownames(sil_dia) =rownames(states)
fviz_silhouette(sil_dia,label=TRUE)
```

```
##   cluster size ave.sil.width
## 1       1    3          0.10
## 2       2    7          0.24
## 3       3    5          0.37
## 4       4    4          0.30
## 5       5   20          0.30
## 6       6    6          0.10
## 7       7    2          0.42
## 8       8    2          0.60
## 9       9    1          0.00
```



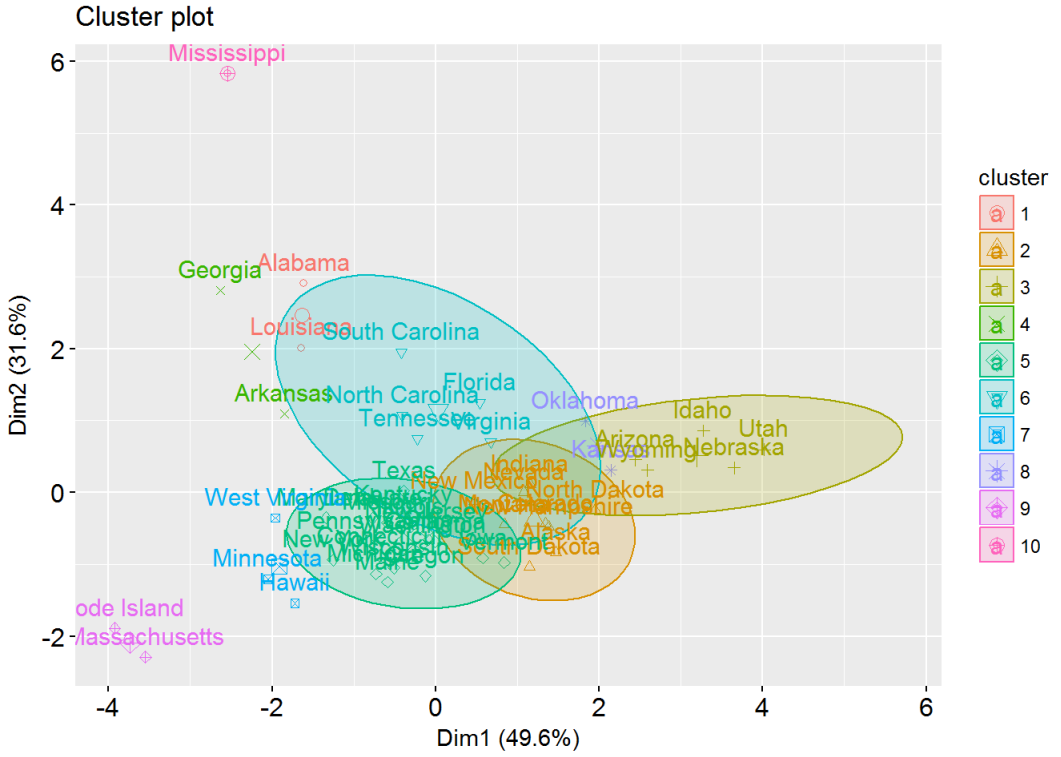
We do see that states that

predominantely vote republican or democrat are closer together in the hierarchy. Additionally, although there are some negative silhouette values, overall the clusters seem to do a good job. Over intended degree of state-by-state accuaracy obviously would change this opinion.

Based on your choice of the optimal number of clusters in each case, visualize the clusters using a principal components plot, and compare them with the clustering results in Part 2b.

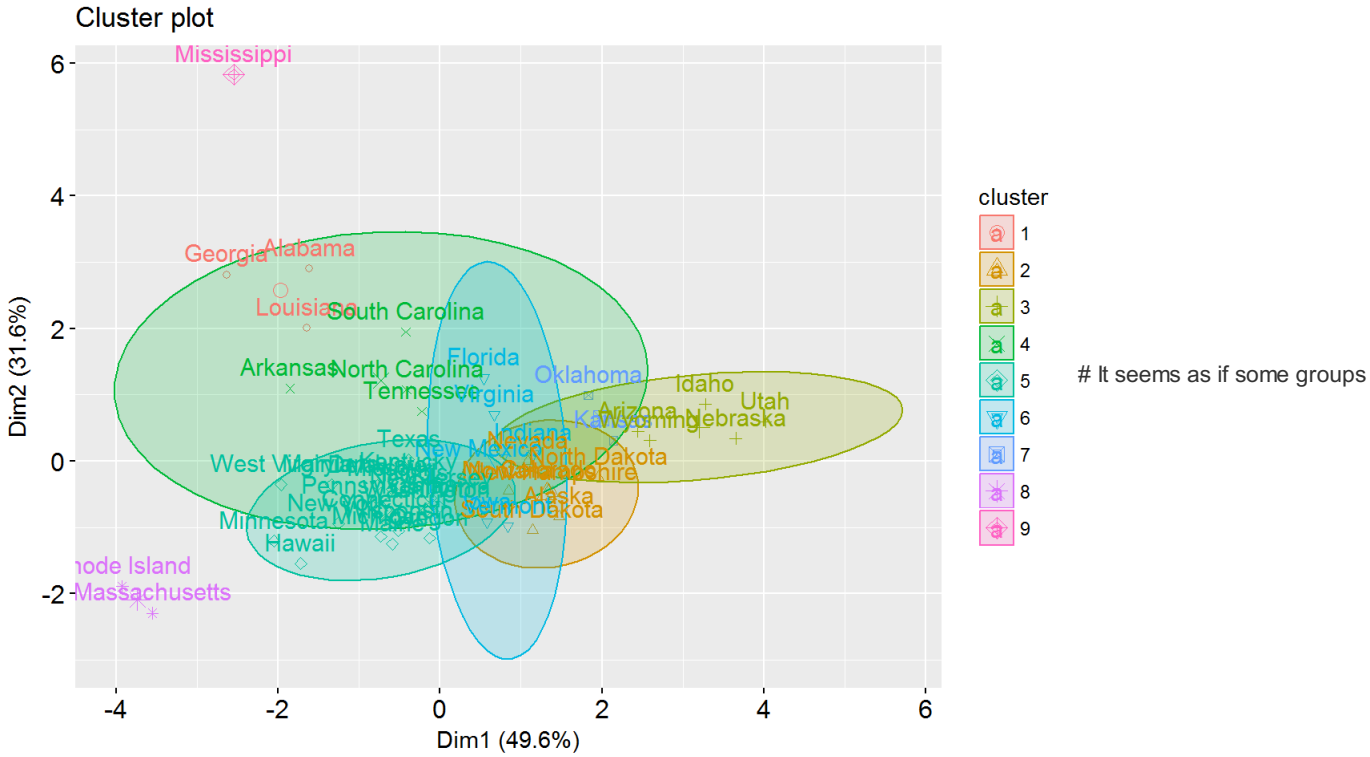
```
agnes.res = agnes.reformat(states_rescaled,10)
fviz_cluster(list(data=states_rescaled,cluster=agnes.res$cluster), ellipse.type = "norm")
```

```
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
```



```
diana.res = diana.reformat(states_rescaled,9)
fviz_cluster(list(data=states_rescaled,cluster=diana.res$cluster), ellipse.type = "norm")
```

```
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
## Too few points to calculate an ellipse
```



It seems as if some groups

around the lower left, where states and clusters tended to overlap in 2b, have been divided greater in this method. Based on the silhouette plots from 2b and those from this problem, this seems to be beneficial.

Part 2d: Soft clustering

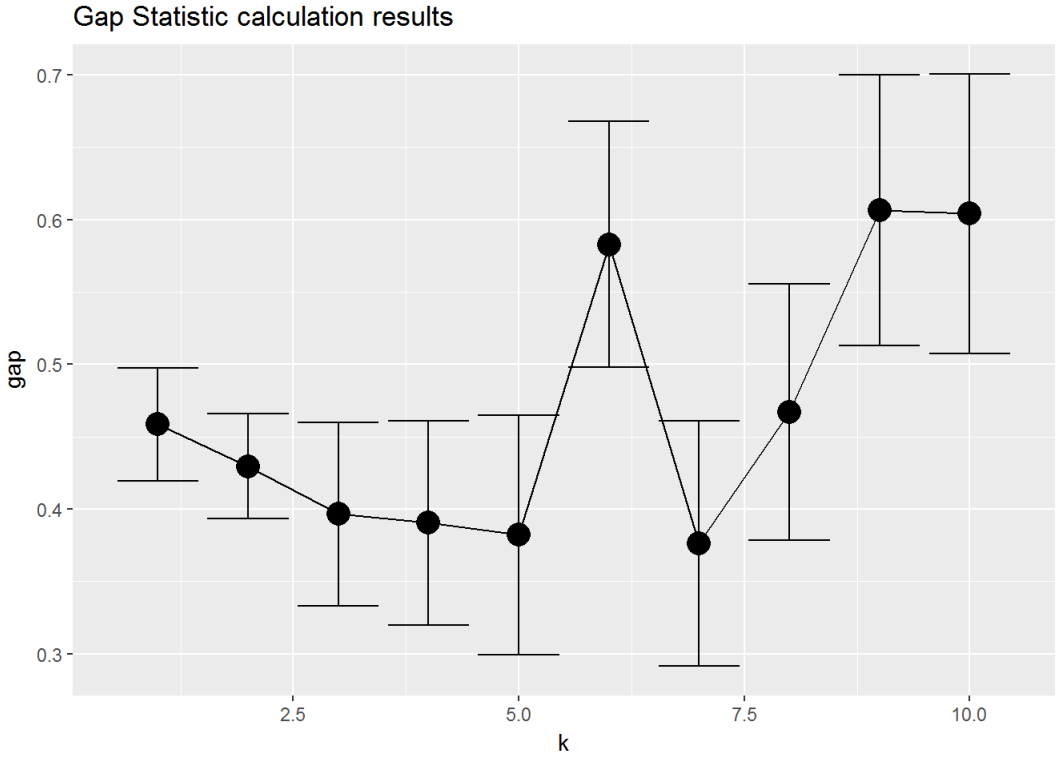
We now explore if soft clustering techniques can produce intuitive grouping. Apply the following methods to the data:

- **Fuzzy clustering** (*Hint:* use the `fanny` function)
- **Gaussian mixture model** (*Hint:* use the `Mclust` function)

For the fuzzy clustering, use the Gap statistic to choose the optimal number of clusters. For the Gaussian mixture model, use the internal tuning feature in `Mclust` to choose the optimal number of mixture components.

Summarize both sets of results using both a principal components plot, and a correlation plot of the cluster membership probabilities. Compare the results of the clusterings. Comment on the membership probabilities of the states. Do any states have membership probabilities approximately equal between clusters? For the fuzzy clustering, generate a silhouette diagnostic plot, and comment on the quality of clustering.

```
#fuzzy
fuzzy.reformat = function(x,k){
  # x: Data matrix or frame, k: Number of clusters
  x.fuzzy = fanny(x,k=k,memb.exp = 1)
  x.cluster = list(cluster=cutree(x.fuzzy,k=k))
  return(x.cluster)
}

gsfuz <- clusGap(x=states_rescaled, FUN = fanny, K.max = 10, B = 60)
plot_clusgap(gsfuz)
```




```
print("best k=9")
```

```
## [1] "best k=9"
```

```
fuzzy9 = fanny(x=states_rescaled,k=9, memb.exp=6)


#MIxture Model
states_mclust = Mclust(states_rescaled,G=1:10)
states_mclust$G
```

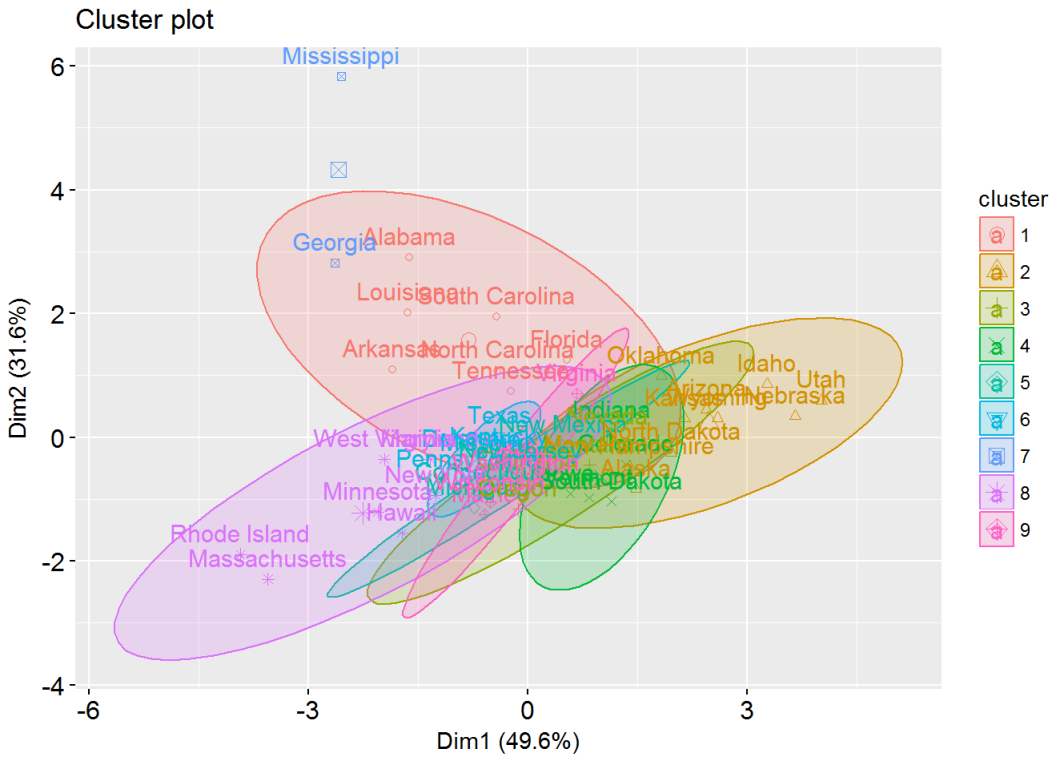
```
## [1] 2
```

```
print("optimal mixture components is 2")
```

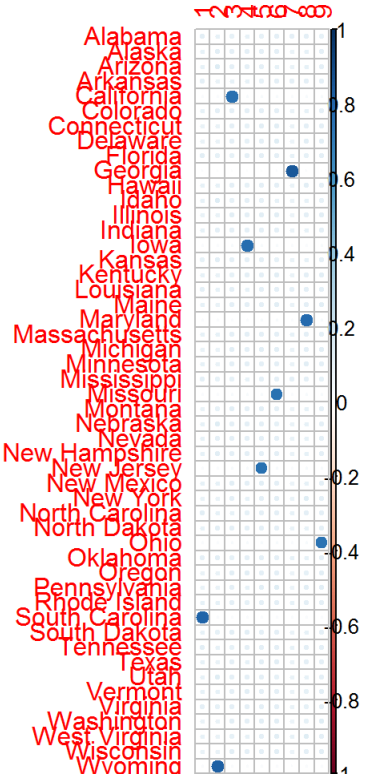
```
## [1] "optimal mixture components is 2"
```

```
#plots

#fuzzy
fviz_cluster(list(data=states_rescaled,cluster=fuzzy9$clustering), ellipse.type = "norm")
```

```
## Too few points to calculate an ellipse
```

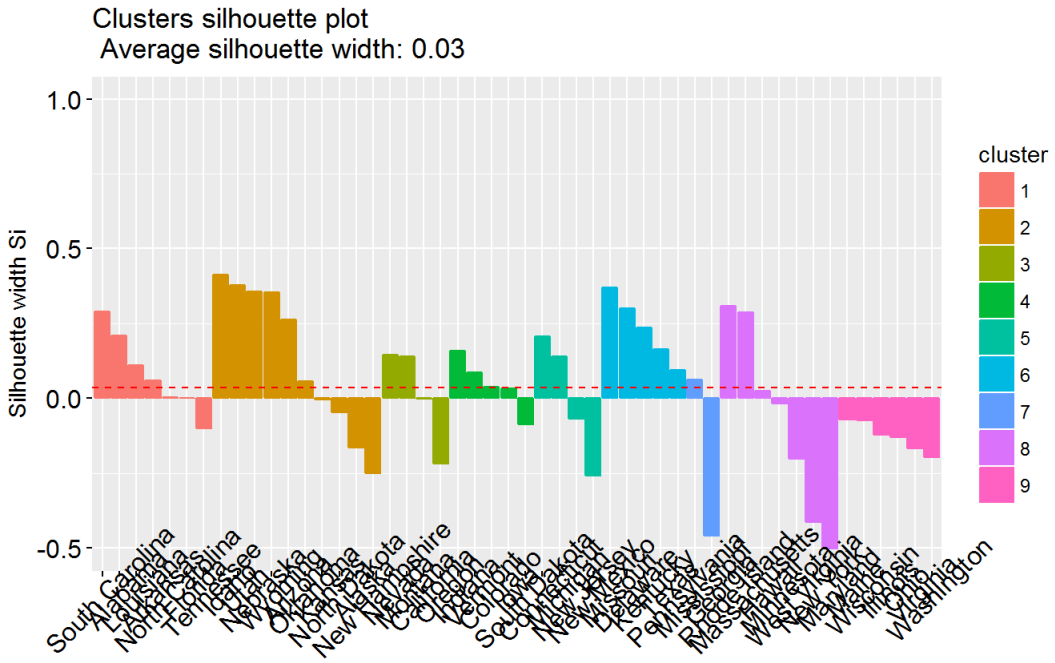



```
corrplot(fuzzy9$membership,method="circle")
```

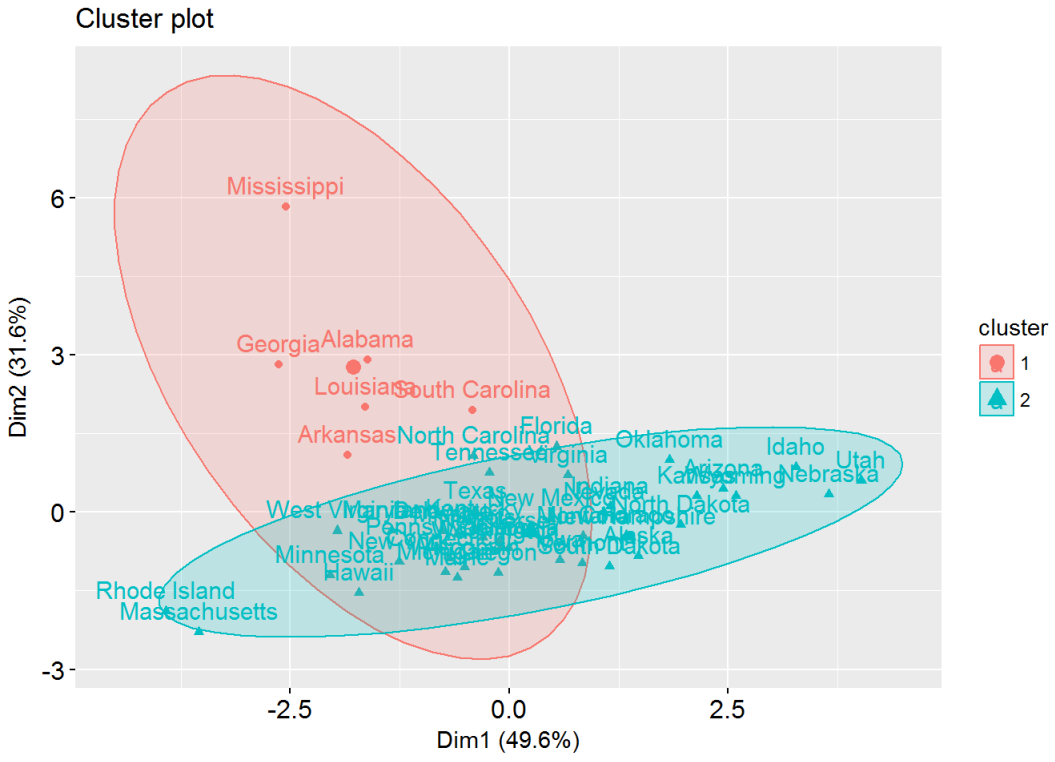



```
sil_fuz = silhouette(fuzzy9$clustering, dist(states_rescaled))
rownames(sil_fuz) =rownames(states)
fviz_silhouette(sil_fuz,label=TRUE)
```

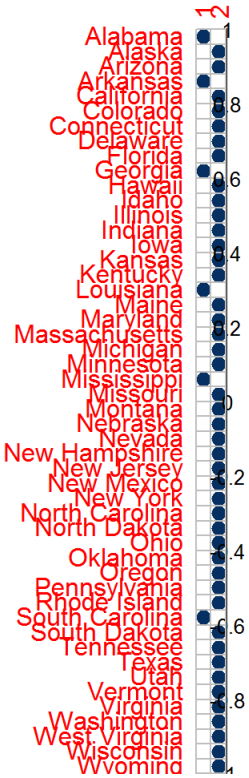
```
##   cluster size ave.sil.width
## 1       1    7          0.08
## 2       2   10          0.14
## 3       3    4          0.02
## 4       4    5          0.05
## 5       5    4          0.00
## 6       6    5          0.23
## 7       7    2         -0.20
## 8       8    7         -0.07
## 9       9    6         -0.13
```




```
#mclust
fviz_cluster(list(data=states_rescaled,cluster=states_mclust$classification), ellipse.type = "norm")
```




```
corrplot(states_mclust$z,method="circle")
```




```
#membership probs
#fuzzy
fuzzy9$membership
```

```
##                    [,1]       [,2]       [,3]       [,4]       [,5]
## Alabama        0.12274548 0.10399584 0.10866632 0.10628888 0.10936762
## Alaska         0.10224870 0.12003338 0.11684470 0.11386157 0.11800161
## Arizona        0.10535675 0.14531313 0.11097918 0.11280591 0.11030247
## Arkansas       0.11846750 0.09946235 0.10772105 0.10668319 0.10981020
## California     0.02715350 0.02735254 0.75632357 0.03268998 0.03399404
## Colorado       0.10124263 0.11854372 0.11665116 0.11910746 0.11867357
## Connecticut    0.09757257 0.09736339 0.11926119 0.10838666 0.12727855
## Delaware       0.09895565 0.09179941 0.11367306 0.10896804 0.11671519
## Florida        0.12202807 0.10896068 0.10966082 0.11077852 0.11297693
## Georgia        0.02182690 0.01850788 0.01934313 0.01916855 0.01961565
## Hawaii         0.10349707 0.10107508 0.11207973 0.10995359 0.11733014
## Idaho          0.10684514 0.14402743 0.10969272 0.11077874 0.10998614
## Illinois       0.09582538 0.09333457 0.12637096 0.11209354 0.12242938
## Indiana        0.10546976 0.11502485 0.11410902 0.11999461 0.11652141
## Iowa           0.02672759 0.02736430 0.03217473 0.76499718 0.03126070
## Kansas         0.10866270 0.12619086 0.11089477 0.11728364 0.11196043
## Kentucky       0.10390400 0.09631794 0.10965108 0.11469961 0.11660071
## Louisiana      0.11719292 0.10331008 0.11045797 0.10597368 0.11182447
## Maine          0.10107834 0.09976302 0.11383395 0.11425481 0.11847036
## Maryland       0.02677409 0.02471135 0.02903053 0.02791299 0.03078203
## Massachusetts  0.10626131 0.10291772 0.11426669 0.11040597 0.11392513
## Michigan       0.09750454 0.09697682 0.12040651 0.11057923 0.12306417
## Minnesota      0.10420146 0.09785615 0.11530008 0.11094678 0.11391670
## Mississippi    0.11538690 0.10844993 0.10993272 0.10789042 0.11015291
## Missouri       0.02881855 0.02735092 0.03367668 0.03217070 0.03772107
## Montana        0.09945591 0.10847517 0.12650687 0.11886373 0.12050518
## Nebraska       0.10723231 0.13740942 0.11019873 0.11332612 0.11032581
## Nevada         0.10429363 0.11598010 0.11907302 0.11370145 0.11801412
## New Hampshire  0.10151761 0.12037419 0.11671899 0.11623102 0.11909138
## New Jersey     0.02795204 0.02804942 0.03457553 0.03230436 0.74429897
## New Mexico     0.10099151 0.10340683 0.12155868 0.11736637 0.12316439
## New York       0.09788265 0.09371770 0.11378333 0.10713796 0.11914661
## North Carolina 0.12483028 0.10140975 0.10760208 0.10831836 0.11190379
## North Dakota   0.10446693 0.12524120 0.11549542 0.11752699 0.11390386
## Ohio           0.02858062 0.02820143 0.03581425 0.03553238 0.03785611
## Oklahoma       0.11333537 0.12071182 0.10946229 0.11340017 0.11220281
## Oregon         0.09553891 0.09546389 0.12777077 0.12357903 0.11755374
## Pennsylvania   0.09516212 0.09144245 0.11518853 0.10612220 0.12370751
## Rhode Island   0.10678176 0.10304334 0.11261394 0.10938168 0.11392336
```

```
## South Carolina 0.80633442 0.02295940 0.02397441 0.02397612 0.02426465
## South Dakota   0.10238014 0.10875844 0.12191558 0.12577910 0.11344527
## Tennessee      0.12247264 0.10166407 0.10812983 0.11178679 0.11138432
## Texas          0.10182766 0.09719786 0.11175641 0.10710461 0.12733938
## Utah           0.10731237 0.13452554 0.11087597 0.11154528 0.11116475
## Vermont        0.10131752 0.10727115 0.11339743 0.12448727 0.11741459
## Virginia       0.11384278 0.10929761 0.11142950 0.11448065 0.11545439
## Washington     0.09409361 0.09453126 0.12736156 0.11147862 0.12789804
## West Virginia  0.10580228 0.09628042 0.10933346 0.10679063 0.11380477
## Wisconsin      0.09757718 0.09495397 0.12393878 0.11841970 0.11638010
## Wyoming        0.02322761 0.81053832 0.02443232 0.02483409 0.02463368
##                      [,6]       [,7]       [,8]       [,9]
## Alabama        0.11109208 0.11683198 0.11165768 0.10935411
## Alaska         0.11194055 0.09463581 0.10761075 0.11482294
## Arizona        0.10731780 0.09469309 0.10306358 0.11016807
## Arkansas       0.11394318 0.11807186 0.11562440 0.11021626
## California     0.03289820 0.02467624 0.03020162 0.03471031
## Colorado       0.11189568 0.09124095 0.10490392 0.11774091
## Connecticut    0.12208167 0.09109433 0.11845826 0.11850339
## Delaware       0.13530406 0.08967590 0.12404383 0.12086488
## Florida        0.11323969 0.10086405 0.10854567 0.11294557
## Georgia        0.02003401 0.84153030 0.02040496 0.01956862
## Hawaii         0.11868131 0.09997940 0.12129940 0.11610429
## Idaho          0.10740420 0.09791787 0.10422259 0.10912517
## Illinois       0.12463126 0.08643456 0.11140554 0.12747482
## Indiana        0.11279426 0.09249128 0.10558939 0.11800540
## Iowa           0.03093181 0.02406822 0.02858141 0.03389406
## Kansas         0.10996703 0.09653504 0.10514325 0.11336228
## Kentucky       0.12624172 0.09304867 0.11674204 0.12279425
## Louisiana      0.11326271 0.11282138 0.11505398 0.11010280
## Maine          0.12004869 0.09385930 0.11555234 0.12313919
## Maryland       0.03392376 0.02502183 0.77144294 0.03040048
## Massachusetts  0.11521726 0.10571999 0.11780838 0.11347755
## Michigan       0.12160858 0.09086317 0.11795692 0.12104007
## Minnesota      0.11901438 0.09978826 0.12270500 0.11627118
## Mississippi    0.11058527 0.11678910 0.11123347 0.10957929
## Missouri       0.74015345 0.02616256 0.03612836 0.03781770
## Montana        0.11284305 0.08971625 0.10493207 0.11870177
## Nebraska       0.10812876 0.09815173 0.10471758 0.11050954
## Nevada         0.11256566 0.09488401 0.10667722 0.11481079
## New Hampshire  0.11192199 0.09234992 0.10546701 0.11632790
## New Jersey     0.03747939 0.02545174 0.03257158 0.03731696
## New Mexico     0.11673308 0.08988757 0.10693777 0.11995379
## New York       0.12660468 0.09213900 0.13214632 0.11744174
## North Carolina 0.11546143 0.10499605 0.11383389 0.11164437
## North Dakota   0.11014159 0.09471797 0.10470594 0.11380012
## Ohio           0.03811832 0.02575765 0.03263256 0.73750669
## Oklahoma       0.11092777 0.10052093 0.10684131 0.11259754
## Oregon         0.11775973 0.08753282 0.10889472 0.12590641
## Pennsylvania   0.13595147 0.08782358 0.12266716 0.12193497
## Rhode Island   0.11516753 0.10734776 0.11890542 0.11283523
## South Carolina 0.02485623 0.02458514 0.02459282 0.02445682
## South Dakota   0.11147809 0.09322632 0.10567035 0.11734671
## Tennessee      0.11555507 0.10277088 0.11242868 0.11380772
## Texas          0.12769266 0.09190683 0.11544138 0.11973321
## Utah           0.10864012 0.09988364 0.10582620 0.11022612
## Vermont        0.11454319 0.09191189 0.10829934 0.12135763
## Virginia       0.11478288 0.09617931 0.10835155 0.11618135
## Washington     0.12053378 0.08570897 0.10838190 0.13001226
## West Virginia  0.12101131 0.10208978 0.13105948 0.11382788
## Wisconsin      0.12084838 0.08956563 0.11382588 0.12449036
## Wyoming        0.02386625 0.02108985 0.02296348 0.02441440
```

```
#mclust
states_mclust$z
```

```
##                      [,1]          [,2]
## Alabama         1.000000e+00  1.164724e-47
## Alaska          1.466621e-33  1.000000e+00
## Arizona         1.039645e-11  1.000000e+00
## Arkansas        9.981010e-01  1.898970e-03
## California      2.930771e-17  1.000000e+00
## Colorado        6.138212e-17  1.000000e+00
## Connecticut     1.911938e-12  1.000000e+00
## Delaware        1.084740e-04  9.998915e-01
## Florida         2.968484e-09  1.000000e+00
## Georgia         1.000000e+00  7.924744e-10
## Hawaii         6.565548e-119  1.000000e+00
## Idaho           7.572377e-23  1.000000e+00
## Illinois        1.329522e-05  9.999867e-01
## Indiana         2.444373e-13  1.000000e+00
## Iowa            4.026786e-07  9.999996e-01
## Kansas          1.676793e-20  1.000000e+00
## Kentucky        1.054106e-11  1.000000e+00
## Louisiana       1.000000e+00  4.465657e-21
## Maine           1.886637e-46  1.000000e+00
## Maryland        2.097661e-10  1.000000e+00
## Massachusetts   1.249200e-28  1.000000e+00
## Michigan        3.467947e-10  1.000000e+00
## Minnesota       3.995504e-16  1.000000e+00
## Mississippi     1.000000e+00 1.904452e-179
## Missouri        3.156396e-07  9.999997e-01
## Montana         7.829112e-10  1.000000e+00
## Nebraska        4.294856e-29  1.000000e+00
## Nevada          7.404244e-10  1.000000e+00
## New Hampshire   2.926880e-17  1.000000e+00
## New Jersey      2.583603e-12  1.000000e+00
## New Mexico      1.518121e-06  9.999985e-01
## New York        1.294936e-07  9.999999e-01
## North Carolina  1.237913e-07  9.999999e-01
## North Dakota    5.637782e-14  1.000000e+00
## Ohio            2.703070e-06  9.999973e-01
## Oklahoma        1.068501e-57  1.000000e+00
## Oregon          9.063617e-13  1.000000e+00
## Pennsylvania    1.728226e-06  9.999983e-01
## Rhode Island    5.842272e-33  1.000000e+00
## South Carolina  9.999908e-01  9.233694e-06
## South Dakota    4.777319e-59  1.000000e+00
## Tennessee       1.529776e-06  9.999985e-01
## Texas           7.707097e-20  1.000000e+00
## Utah            5.509867e-39  1.000000e+00
## Vermont         2.819204e-43  1.000000e+00
## Virginia        1.132773e-12  1.000000e+00
## Washington      5.330520e-06  9.999947e-01
## West Virginia   7.108804e-16  1.000000e+00
## Wisconsin       3.065933e-14  1.000000e+00
## Wyoming         4.953411e-28  1.000000e+00
```

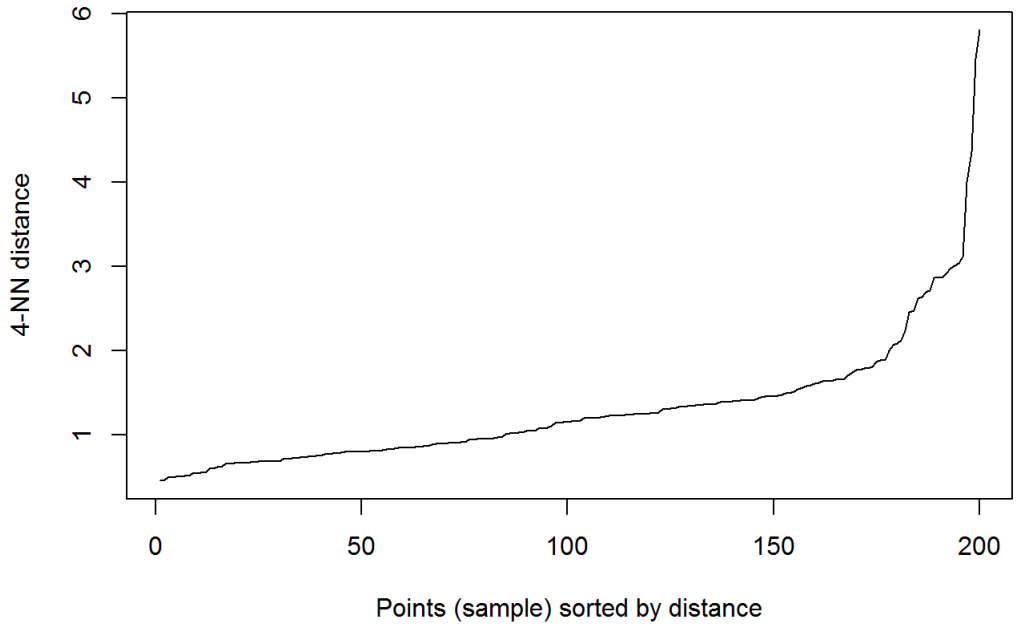
The fuzzy method has a few states that have extremely close probabilities for each cluster, but the Mclust method seems to have most of the states with either a 0 or 1 probability for the clusters. This does not seem right. Hmm. Additionally the silhouette plot for the fuzzy method looks… not good. Too many states are negative or near zero. What the heck?

Hint: use the `membership` attribute to obtain the cluster membership probabilties from the cluster model, and the `corrplot` function to generate a correlation plot.

Part 2e: Density-based clustering

Apply DBSCAN to the data with `minPts = 5` (*Hint:* use the `dbscan` function). Create a knee plot (*Hint:* use the `kNNdistplot` function) to estimate `eps`. 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?

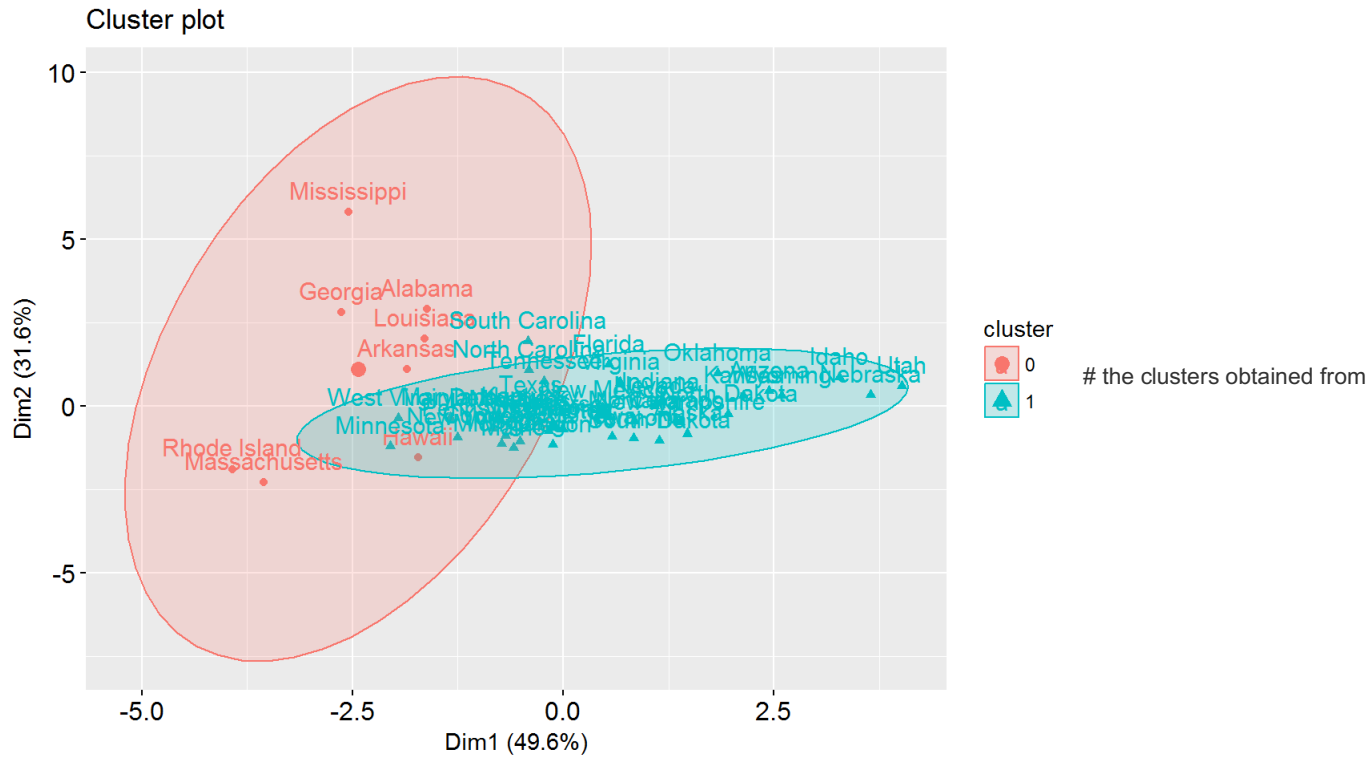
```
kNNdistplot(states_rescaled,k=4)
```




```
print("optimal eps around 1.5")
```

```
## [1] "optimal eps around 1.5"
```

```
states_dbscan = dbscan(states_rescaled,minPts = 5,eps=1.5)
fviz_cluster(list(data=states_rescaled,cluster=states_dbscan$cluster), ellipse.type = "norm")
```



the clusters obtained from

the dbscan method give only two clusters. Cluster 0 looks distorted for the purpose of including the outlier states, like Mississippi. This seems to be a form of overcompensating. Method seems non-ideal in this case.