Homework 5 - PCA, SVM & Clustering

Harvard CS109B, Spring 2017

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 250×250, 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)
}

# modified plot.face function for better contrast
plot.face = function(x,zlim=c(0,1)) {
  #Plots Face given image vector x
  x = pmin(pmax(x,zlim[1]),zlim[2])
  cols = gray.colors(100)[1:100]
  image(rot90(matrix(x,nrow=250)[,250:1],3),col=cols,
        zlim=zlim,axes=FALSE, asp=1)
}
```

- 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)

Answer

```
data <- load('CS109b-hw5-dataset_1.Rdata')

dim(imgs_train)
```

```
## [1]   339 62500
```

```
dim(imgs_test)
```

```
## [1]   339 62500
```

```
# example of an image (image 1)
plot.face(imgs_train[1,])
```

```
# Perform principal component analysis
imgs_train.prc <- prcomp(imgs_train,
                         center = TRUE,
                         scale = TRUE)
summary(imgs_train.prc)
```

```
## Importance of components:
##                             PC1      PC2      PC3      PC4      PC5
## Standard deviation     113.4501 62.65555 60.23879 58.51709 50.31605
## Proportion of Variance   0.2059  0.06281  0.05806  0.05479  0.04051
## Cumulative Proportion    0.2059  0.26875  0.32681  0.38159  0.42210
##                             PC6      PC7      PC8      PC9     PC10
## Standard deviation     49.63817 44.69014 36.64647 36.01672 34.33743
## Proportion of Variance  0.03942  0.03196  0.02149  0.02076  0.01886
## Cumulative Proportion   0.46152  0.49348  0.51497  0.53572  0.55459
##                            PC11     PC12     PC13    PC14     PC15
## Standard deviation     32.25369 30.84751 30.33609 28.5066 27.54532
## Proportion of Variance  0.01664  0.01523  0.01472  0.0130  0.01214
## Cumulative Proportion   0.57123  0.58646  0.60118  0.6142  0.62632
##                            PC16     PC17     PC18     PC19     PC20
## Standard deviation     25.21336 24.59068 23.14006 22.66151 22.19603
## Proportion of Variance  0.01017  0.00968  0.00857  0.00822  0.00788
```

```
## Cumulative Proportion   0.63649  0.64617  0.65474  0.66295  0.67084
##                            PC21     PC22     PC23     PC24     PC25
## Standard deviation     21.3647 20.70499 20.49456 19.81975 19.44335
## Proportion of Variance  0.0073  0.00686  0.00672  0.00629  0.00605
## Cumulative Proportion   0.6781  0.68500  0.69172  0.69800  0.70405
##                            PC26    PC27     PC28     PC29    PC30    PC31
## Standard deviation     18.66807 18.3697 18.27945 17.93717 17.4989 17.3193
## Proportion of Variance  0.00558  0.0054  0.00535  0.00515  0.0049  0.0048
## Cumulative Proportion   0.70963  0.7150  0.72037  0.72552  0.7304  0.7352
##                            PC32     PC33     PC34     PC35     PC36
## Standard deviation     17.07042 16.72703 16.33078 15.95963 15.91429
## Proportion of Variance  0.00466  0.00448  0.00427  0.00408  0.00405
## Cumulative Proportion   0.73988  0.74436  0.74863  0.75270  0.75676
##                            PC37     PC38     PC39     PC40    PC41
## Standard deviation     15.71510 15.42719 15.21859 14.90360 14.7959
## Proportion of Variance  0.00395  0.00381  0.00371  0.00355  0.0035
## Cumulative Proportion   0.76071  0.76451  0.76822  0.77177  0.7753
##                            PC42     PC43     PC44    PC45     PC46
## Standard deviation     14.64152 14.41278 14.17363 13.9280 13.59042
## Proportion of Variance  0.00343  0.00332  0.00321  0.0031  0.00296
## Cumulative Proportion   0.77871  0.78203  0.78524  0.7883  0.79130
##                            PC47     PC48     PC49    PC50     PC51
## Standard deviation     13.49256 13.29167 13.11525 12.9922 12.97226
## Proportion of Variance  0.00291  0.00283  0.00275  0.0027  0.00269
## Cumulative Proportion   0.79422  0.79704  0.79980  0.8025  0.80519
##                            PC52     PC53     PC54     PC55     PC56
## Standard deviation     12.79502 12.54690 12.46800 12.33548 12.31936
## Proportion of Variance  0.00262  0.00252  0.00249  0.00243  0.00243
## Cumulative Proportion   0.80781  0.81033  0.81281  0.81525  0.81768
##                            PC57     PC58     PC59    PC60     PC61     PC62
## Standard deviation     12.2563 12.05229 11.88824 11.7197 11.55954 11.53791
## Proportion of Variance  0.0024  0.00232  0.00226  0.0022  0.00214  0.00213
## Cumulative Proportion   0.8201  0.82240  0.82467  0.8269  0.82900  0.83113
##                            PC63     PC64     PC65     PC66    PC67
## Standard deviation     11.48537 11.30683 11.08911 11.03347 10.8992
## Proportion of Variance  0.00211  0.00205  0.00197  0.00195  0.0019
## Cumulative Proportion   0.83324  0.83529  0.83725  0.83920  0.8411
##                            PC68     PC69     PC70     PC71     PC72
## Standard deviation     10.76940 10.74636 10.68776 10.64826 10.57388
## Proportion of Variance  0.00186  0.00185  0.00183  0.00181  0.00179
## Cumulative Proportion   0.84296  0.84481  0.84663  0.84845  0.85024
##                            PC73     PC74     PC75     PC76     PC77
## Standard deviation     10.43659 10.41219 10.33909 10.28531 10.16626
## Proportion of Variance  0.00174  0.00173  0.00171  0.00169  0.00165
## Cumulative Proportion   0.85198  0.85371  0.85543  0.85712  0.85877
##                            PC78     PC79    PC80    PC81    PC82    PC83
## Standard deviation     10.10831 10.07277 9.91739 9.89610 9.76575 9.71860
## Proportion of Variance  0.00163  0.00162 0.00157 0.00157 0.00153 0.00151
## Cumulative Proportion   0.86041  0.86203 0.86360 0.86517 0.86670 0.86821
##                            PC84    PC85    PC86    PC87   PC88    PC89
## Standard deviation     9.65610 9.62426 9.53143 9.44722 9.3537 9.33582
## Proportion of Variance 0.00149 0.00148 0.00145 0.00143 0.0014 0.00139
## Cumulative Proportion  0.86970 0.87118 0.87263 0.87406 0.8755 0.87686
##                            PC90    PC91    PC92    PC93    PC94    PC95
```

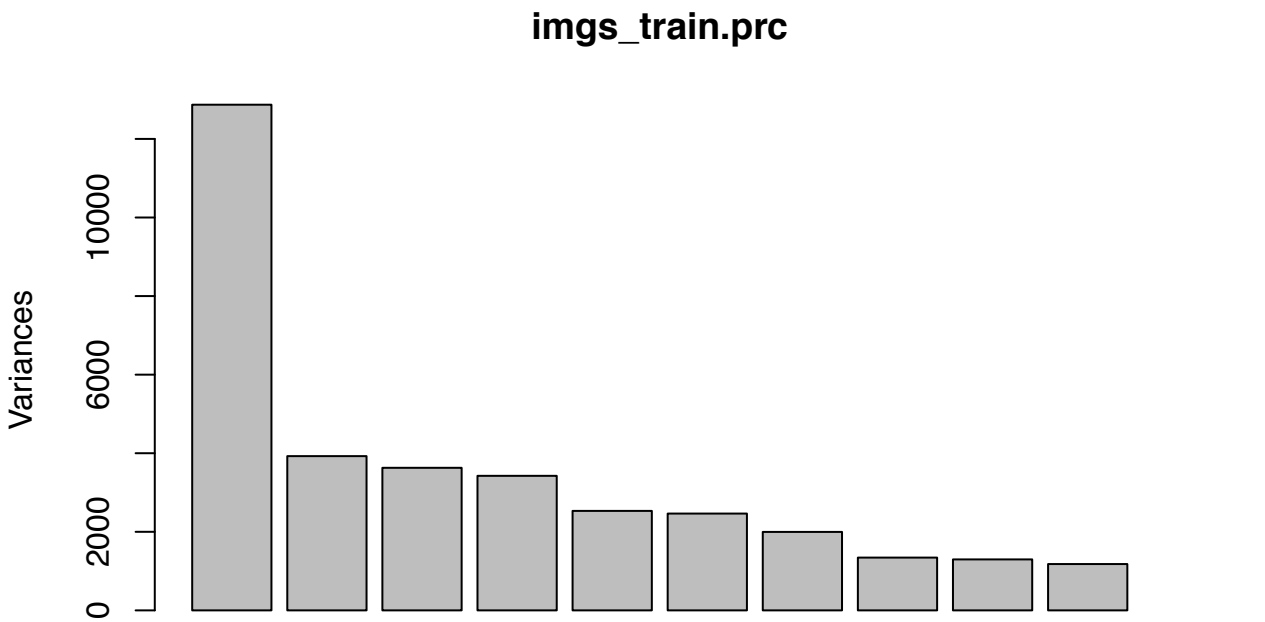
```
## Standard deviation     9.28107 9.17310 9.09191 8.95964 8.93784 8.88899
## Proportion of Variance 0.00138 0.00135 0.00132 0.00128 0.00128 0.00126
## Cumulative Proportion  0.87824 0.87958 0.88090 0.88219 0.88347 0.88473
##                           PC96    PC97    PC98    PC99   PC100   PC101
## Standard deviation     8.87472 8.84081 8.74372 8.63771 8.62449 8.53609
## Proportion of Variance 0.00126 0.00125 0.00122 0.00119 0.00119 0.00117
## Cumulative Proportion  0.88599 0.88724 0.88847 0.88966 0.89085 0.89201
##                           PC102   PC103   PC104   PC105   PC106   PC107
## Standard deviation     8.51179 8.44225 8.36277 8.34506 8.32943 8.26622
## Proportion of Variance 0.00116 0.00114 0.00112 0.00111 0.00111 0.00109
## Cumulative Proportion  0.89317 0.89431 0.89543 0.89655 0.89766 0.89875
##                           PC108   PC109   PC110   PC111   PC112   PC113
## Standard deviation     8.16580 8.10749 8.08025 8.01101 8.00553 7.94726
## Proportion of Variance 0.00107 0.00105 0.00104 0.00103 0.00103 0.00101
## Cumulative Proportion  0.89982 0.90087 0.90191 0.90294 0.90397 0.90498
##                           PC114 PC115   PC116   PC117   PC118   PC119
## Standard deviation     7.92570 7.891 7.83128 7.81300 7.80042 7.74065
## Proportion of Variance 0.00101 0.001 0.00098 0.00098 0.00097 0.00096
## Cumulative Proportion  0.90598 0.907 0.90796 0.90894 0.90991 0.91087
##                           PC120   PC121   PC122   PC123   PC124  PC125
## Standard deviation     7.69781 7.63352 7.59428 7.56955 7.54192 7.5058
## Proportion of Variance 0.00095 0.00093 0.00092 0.00092 0.00091 0.0009
## Cumulative Proportion  0.91182 0.91275 0.91367 0.91459 0.91550 0.9164
##                           PC126   PC127   PC128   PC129   PC130   PC131
## Standard deviation     7.41369 7.40076 7.35487 7.29076 7.27326 7.24931
## Proportion of Variance 0.00088 0.00088 0.00087 0.00085 0.00085 0.00084
## Cumulative Proportion  0.91728 0.91816 0.91902 0.91987 0.92072 0.92156
##                           PC132   PC133   PC134  PC135   PC136   PC137
## Standard deviation     7.22674 7.18377 7.12101 7.0775 7.02867 7.00508
## Proportion of Variance 0.00084 0.00083 0.00081 0.0008 0.00079 0.00079
## Cumulative Proportion  0.92239 0.92322 0.92403 0.9248 0.92562 0.92641
##                           PC138   PC139   PC140   PC141   PC142   PC143
## Standard deviation     6.96167 6.91585 6.86731 6.82833 6.81116 6.76993
## Proportion of Variance 0.00078 0.00077 0.00075 0.00075 0.00074 0.00073
## Cumulative Proportion  0.92718 0.92795 0.92870 0.92945 0.93019 0.93093
##                           PC144   PC145   PC146  PC147  PC148   PC149
## Standard deviation     6.75585 6.69790 6.68039 6.6272 6.6016 6.57860
## Proportion of Variance 0.00073 0.00072 0.00071 0.0007 0.0007 0.00069
## Cumulative Proportion  0.93166 0.93237 0.93309 0.9338 0.9345 0.93518
##                           PC150   PC151   PC152   PC153   PC154   PC155
## Standard deviation     6.55178 6.51758 6.48400 6.45819 6.40264 6.36999
## Proportion of Variance 0.00069 0.00068 0.00067 0.00067 0.00066 0.00065
## Cumulative Proportion  0.93587 0.93655 0.93722 0.93789 0.93854 0.93919
##                           PC156   PC157   PC158   PC159   PC160   PC161
## Standard deviation     6.36394 6.33045 6.32829 6.31574 6.28868 6.23895
## Proportion of Variance 0.00065 0.00064 0.00064 0.00064 0.00063 0.00062
## Cumulative Proportion  0.93984 0.94048 0.94112 0.94176 0.94239 0.94302
##                           PC162   PC163   PC164  PC165  PC166   PC167
## Standard deviation     6.19997 6.16058 6.15026 6.1238 6.1059 6.07935
## Proportion of Variance 0.00062 0.00061 0.00061 0.0006 0.0006 0.00059
## Cumulative Proportion  0.94363 0.94424 0.94484 0.9454 0.9460 0.94663
##                           PC168   PC169   PC170   PC171   PC172   PC173
## Standard deviation     6.03542 6.02029 5.99839 5.97332 5.94822 5.92177
## Proportion of Variance 0.00058 0.00058 0.00058 0.00057 0.00057 0.00056
```

```
## Cumulative Proportion  0.94721 0.94779 0.94837 0.94894 0.94951 0.95007
##                           PC174   PC175   PC176   PC177   PC178   PC179
## Standard deviation     5.89746 5.88283 5.84559 5.81533 5.79325 5.75834
## Proportion of Variance 0.00056 0.00055 0.00055 0.00054 0.00054 0.00053
## Cumulative Proportion  0.95062 0.95118 0.95172 0.95227 0.95280 0.95333
##                           PC180   PC181   PC182   PC183   PC184  PC185
## Standard deviation     5.72571 5.71677 5.70407 5.66427 5.64376 5.6067
## Proportion of Variance 0.00052 0.00052 0.00052 0.00051 0.00051 0.0005
## Cumulative Proportion  0.95386 0.95438 0.95490 0.95541 0.95592 0.9564
##                          PC186  PC187  PC188   PC189   PC190   PC191
## Standard deviation     5.5904 5.5715 5.5669 5.54225 5.51931 5.49010
## Proportion of Variance 0.0005 0.0005 0.0005 0.00049 0.00049 0.00048
## Cumulative Proportion  0.9569 0.9574 0.9579 0.95841 0.95890 0.95938
##                           PC192   PC193   PC194   PC195   PC196   PC197
## Standard deviation     5.47677 5.44546 5.43356 5.41016 5.39149 5.36058
## Proportion of Variance 0.00048 0.00047 0.00047 0.00047 0.00047 0.00046
## Cumulative Proportion  0.95986 0.96033 0.96081 0.96128 0.96174 0.96220
##                           PC198   PC199   PC200   PC201   PC202   PC203
## Standard deviation     5.35486 5.33404 5.30522 5.29861 5.21852 5.18793
## Proportion of Variance 0.00046 0.00046 0.00045 0.00045 0.00044 0.00043
## Cumulative Proportion  0.96266 0.96311 0.96356 0.96401 0.96445 0.96488
##                           PC204   PC205   PC206   PC207   PC208   PC209
## Standard deviation     5.18262 5.15522 5.14961 5.13198 5.09700 5.09171
## Proportion of Variance 0.00043 0.00043 0.00042 0.00042 0.00042 0.00041
## Cumulative Proportion  0.96531 0.96574 0.96616 0.96658 0.96700 0.96741
##                           PC210   PC211  PC212  PC213  PC214   PC215
## Standard deviation     5.06436 5.04431 5.0241 5.0176 5.0022 4.96760
## Proportion of Variance 0.00041 0.00041 0.0004 0.0004 0.0004 0.00039
## Cumulative Proportion  0.96782 0.96823 0.9686 0.9690 0.9694 0.96983
##                           PC216   PC217   PC218   PC219   PC220   PC221
## Standard deviation     4.95305 4.94262 4.90123 4.89457 4.88214 4.86063
## Proportion of Variance 0.00039 0.00039 0.00038 0.00038 0.00038 0.00038
## Cumulative Proportion  0.97022 0.97061 0.97100 0.97138 0.97176 0.97214
##                           PC222   PC223   PC224   PC225   PC226   PC227
## Standard deviation     4.84556 4.81831 4.78765 4.77271 4.76226 4.73999
## Proportion of Variance 0.00038 0.00037 0.00037 0.00036 0.00036 0.00036
## Cumulative Proportion  0.97252 0.97289 0.97326 0.97362 0.97398 0.97434
##                           PC228   PC229   PC230   PC231   PC232   PC233
## Standard deviation     4.72398 4.71007 4.69905 4.68428 4.65696 4.64783
## Proportion of Variance 0.00036 0.00035 0.00035 0.00035 0.00035 0.00035
## Cumulative Proportion  0.97470 0.97505 0.97541 0.97576 0.97611 0.97645
##                           PC234   PC235   PC236   PC237   PC238   PC239
## Standard deviation     4.62381 4.62168 4.60656 4.56682 4.55614 4.52125
## Proportion of Variance 0.00034 0.00034 0.00034 0.00033 0.00033 0.00033
## Cumulative Proportion  0.97679 0.97713 0.97747 0.97781 0.97814 0.97847
##                           PC240   PC241   PC242   PC243   PC244   PC245
## Standard deviation     4.51758 4.49805 4.47404 4.44846 4.43418 4.42760
## Proportion of Variance 0.00033 0.00032 0.00032 0.00032 0.00031 0.00031
## Cumulative Proportion  0.97879 0.97912 0.97944 0.97975 0.98007 0.98038
##                           PC246   PC247  PC248  PC249  PC250   PC251
## Standard deviation     4.40444 4.37638 4.3498 4.3421 4.3188 4.28373
## Proportion of Variance 0.00031 0.00031 0.0003 0.0003 0.0003 0.00029
## Cumulative Proportion  0.98069 0.98100 0.9813 0.9816 0.9819 0.98220
##                           PC252   PC253   PC254   PC255   PC256   PC257
```

```
## Standard deviation     4.28039 4.26038 4.24892 4.22156 4.20725 4.19682
## Proportion of Variance 0.00029 0.00029 0.00029 0.00029 0.00028 0.00028
## Cumulative Proportion  0.98249 0.98278 0.98307 0.98335 0.98364 0.98392
##                           PC258   PC259   PC260   PC261   PC262   PC263
## Standard deviation     4.17610 4.16706 4.15849 4.14547 4.12685 4.09693
## Proportion of Variance 0.00028 0.00028 0.00028 0.00027 0.00027 0.00027
## Cumulative Proportion  0.98420 0.98448 0.98475 0.98503 0.98530 0.98557
##                           PC264   PC265   PC266   PC267   PC268   PC269
## Standard deviation     4.07785 4.06447 4.05551 4.03311 4.02708 3.99197
## Proportion of Variance 0.00027 0.00026 0.00026 0.00026 0.00026 0.00025
## Cumulative Proportion  0.98583 0.98610 0.98636 0.98662 0.98688 0.98714
##                           PC270   PC271   PC272   PC273   PC274   PC275
## Standard deviation     3.97219 3.94448 3.92819 3.92506 3.91007 3.89673
## Proportion of Variance 0.00025 0.00025 0.00025 0.00025 0.00024 0.00024
## Cumulative Proportion  0.98739 0.98764 0.98788 0.98813 0.98838 0.98862
##                           PC276   PC277   PC278   PC279   PC280   PC281
## Standard deviation     3.88301 3.87169 3.86137 3.84969 3.83824 3.83044
## Proportion of Variance 0.00024 0.00024 0.00024 0.00024 0.00024 0.00023
## Cumulative Proportion  0.98886 0.98910 0.98934 0.98958 0.98981 0.99005
##                           PC282   PC283   PC284   PC285   PC286   PC287
## Standard deviation     3.78718 3.77863 3.75717 3.74821 3.73581 3.71860
## Proportion of Variance 0.00023 0.00023 0.00023 0.00022 0.00022 0.00022
## Cumulative Proportion  0.99028 0.99050 0.99073 0.99095 0.99118 0.99140
##                           PC288   PC289   PC290   PC291   PC292   PC293
## Standard deviation     3.71063 3.69444 3.65973 3.64943 3.62216 3.62016
## Proportion of Variance 0.00022 0.00022 0.00021 0.00021 0.00021 0.00021
## Cumulative Proportion  0.99162 0.99184 0.99205 0.99226 0.99247 0.99268
##                           PC294   PC295  PC296  PC297  PC298  PC299  PC300
## Standard deviation     3.60335 3.58359 3.5760 3.5577 3.5226 3.5015 3.4987
## Proportion of Variance 0.00021 0.00021 0.0002 0.0002 0.0002 0.0002 0.0002
## Cumulative Proportion  0.99289 0.99310 0.9933 0.9935 0.9937 0.9939 0.9941
##                           PC301   PC302   PC303   PC304   PC305   PC306
## Standard deviation     3.49036 3.47958 3.45836 3.43099 3.41162 3.40115
## Proportion of Variance 0.00019 0.00019 0.00019 0.00019 0.00019 0.00019
## Cumulative Proportion  0.99429 0.99448 0.99468 0.99486 0.99505 0.99524
##                           PC307   PC308   PC309   PC310   PC311   PC312
## Standard deviation     3.39015 3.36862 3.35866 3.33843 3.31855 3.30261
## Proportion of Variance 0.00018 0.00018 0.00018 0.00018 0.00018 0.00017
## Cumulative Proportion  0.99542 0.99560 0.99578 0.99596 0.99614 0.99631
##                           PC313   PC314   PC315   PC316   PC317   PC318
## Standard deviation     3.28102 3.26711 3.25663 3.24182 3.21530 3.17950
## Proportion of Variance 0.00017 0.00017 0.00017 0.00017 0.00017 0.00016
## Cumulative Proportion  0.99648 0.99665 0.99682 0.99699 0.99716 0.99732
##                           PC319   PC320   PC321   PC322   PC323   PC324
## Standard deviation     3.15969 3.14263 3.12243 3.10423 3.09277 3.04753
## Proportion of Variance 0.00016 0.00016 0.00016 0.00015 0.00015 0.00015
## Cumulative Proportion  0.99748 0.99764 0.99779 0.99795 0.99810 0.99825
##                           PC325   PC326   PC327   PC328   PC329   PC330
## Standard deviation     3.03612 3.01567 2.97443 2.95301 2.92683 2.88548
## Proportion of Variance 0.00015 0.00015 0.00014 0.00014 0.00014 0.00013
## Cumulative Proportion  0.99840 0.99854 0.99868 0.99882 0.99896 0.99909
##                           PC331   PC332   PC333   PC334   PC335   PC336
## Standard deviation     2.86277 2.82426 2.80275 2.77057 2.70583 2.67518
## Proportion of Variance 0.00013 0.00013 0.00013 0.00012 0.00012 0.00011
```

```
## Cumulative Proportion  0.99922 0.99935 0.99948 0.99960 0.99972 0.99983
##                           PC337   PC338     PC339
## Standard deviation     2.41469 2.17633 1.279e-13
## Proportion of Variance 0.00009 0.00008 0.000e+00
## Cumulative Proportion  0.99992 1.00000 1.000e+00
```

```
# Plotting the first ten principal components
plot(imgs_train.prc)
```



The first five principal components are PC1, PC2, PC3, PC4, and PC5, which are the first five variable loadings: `imgs_train.prc$rotation[, 1]` to `imgs_train.prc$rotation[, 5]`. We visualize them in the following:

First 5 principal components

```
# Top 5 eigenfaces
plot.face(imgs_train.prc$rotation[, 1]*500)
title("PC1")
```

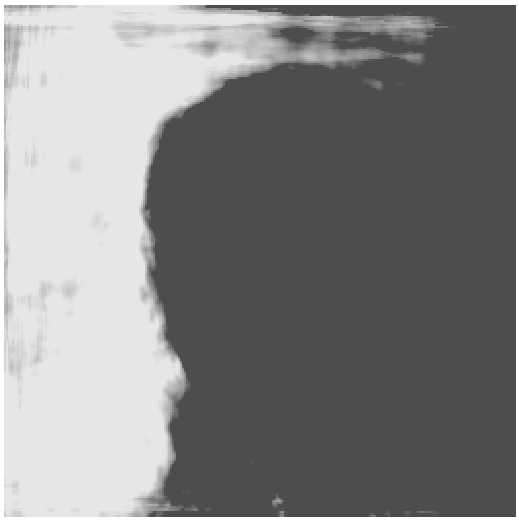
PC1

```
plot.face(imgs_train.prc$rotation[, 2]*500)
title("PC2")
```

PC2

```
plot.face(imgs_train.prc$rotation[, 3]*500)
title("PC3")
```

PC3



```
plot.face(imgs_train.prc$rotation[, 4]*500)
title("PC4")
```

PC4

```
plot.face(imgs_train.prc$rotation[, 5]*500)
title("PC5")
```

PC5

These eigenfaces select out the facial features. PC1 identifies the frontal facial contour, with details to the eys, nose and mouth, PC2 shows the edge of the chin and the right side of the face, PC3 identifies the shape of the left side of the head, PC4 identifies the contour of the top right side of the head, while PC5 identifies the countour and shadow of the face.

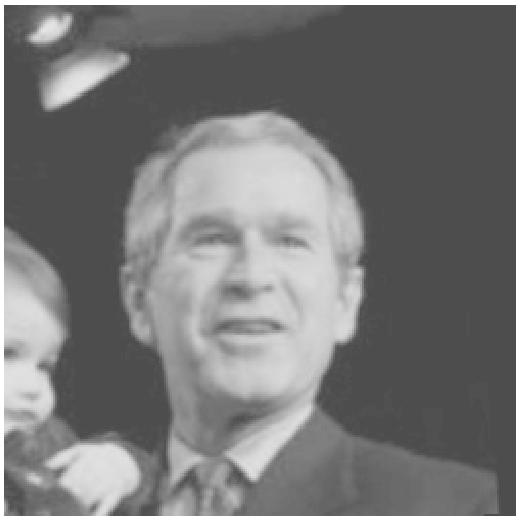
In the following, we reconstructs an image from the top 5 PC components and compare it to the original image.

```
## Reconstruction of the images from the top 5 PC components
imgs_train.pc1to5 <- imgs_train.prc$x[, 1:5, drop = FALSE] %*%
    t(imgs_train.prc$rotation[, 1:5, drop = FALSE]) # loading x eigenvectors

# reconstructed image
plot.face(imgs_train.pc1to5[2,])
title("Recontructed Image from top 5 PC")
```

Recontructed Image from top 5 PC

```r
# original image
plot.face(imgs_train[2,])
title("Original Image")
```



- 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.

Answer

```r
# Cumulative variance
cumvar <- summary(imgs_train.prc)$importance[3,]
cumvar[cumvar<=0.9]
```

```
##     PC1     PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9
## 0.20593 0.26875 0.32681 0.38159 0.42210 0.46152 0.49348 0.51497 0.53572
##    PC10    PC11    PC12    PC13    PC14    PC15    PC16    PC17    PC18
## 0.55459 0.57123 0.58646 0.60118 0.61418 0.62632 0.63649 0.64617 0.65474
##    PC19    PC20    PC21    PC22    PC23    PC24    PC25    PC26    PC27
## 0.66295 0.67084 0.67814 0.68500 0.69172 0.69800 0.70405 0.70963 0.71503
##    PC28    PC29    PC30    PC31    PC32    PC33    PC34    PC35    PC36
## 0.72037 0.72552 0.73042 0.73522 0.73988 0.74436 0.74863 0.75270 0.75676
##    PC37    PC38    PC39    PC40    PC41    PC42    PC43    PC44    PC45
## 0.76071 0.76451 0.76822 0.77177 0.77528 0.77871 0.78203 0.78524 0.78835
##    PC46    PC47    PC48    PC49    PC50    PC51    PC52    PC53    PC54
## 0.79130 0.79422 0.79704 0.79980 0.80250 0.80519 0.80781 0.81033 0.81281
##    PC55    PC56    PC57    PC58    PC59    PC60    PC61    PC62    PC63
## 0.81525 0.81768 0.82008 0.82240 0.82467 0.82686 0.82900 0.83113 0.83324
##    PC64    PC65    PC66    PC67    PC68    PC69    PC70    PC71    PC72
## 0.83529 0.83725 0.83920 0.84110 0.84296 0.84481 0.84663 0.84845 0.85024
##    PC73    PC74    PC75    PC76    PC77    PC78    PC79    PC80    PC81
## 0.85198 0.85371 0.85543 0.85712 0.85877 0.86041 0.86203 0.86360 0.86517
##    PC82    PC83    PC84    PC85    PC86    PC87    PC88    PC89    PC90
## 0.86670 0.86821 0.86970 0.87118 0.87263 0.87406 0.87546 0.87686 0.87824
##    PC91    PC92    PC93    PC94    PC95    PC96    PC97    PC98    PC99
## 0.87958 0.88090 0.88219 0.88347 0.88473 0.88599 0.88724 0.88847 0.88966
```

```
##   PC100   PC101   PC102   PC103   PC104   PC105   PC106   PC107   PC108
## 0.89085 0.89201 0.89317 0.89431 0.89543 0.89655 0.89766 0.89875 0.89982
```

Since the first 108 components give a cummulative varince of 0.89982. We shall use the first 109 principal components in the following which contributes to 90% of the variation in the training data. 109 is a much smaller number compared to the number of pixels of the images, which is 62,500. The number of principal components to keep is 2 orders of magnitudes smaller than the origianl dimension of 62,500. We have reduced the dimension significantly while capturing 90 percent of the variance.

```
# PC score for training set
train.score.90 <- imgs_train.prc$x[, 1:109] #dim: 339 x 109

## PC score for test set
test.score.90 <- scale(imgs_test) %*%
    imgs_train.prc$rotation[, 1:109, drop = FALSE] #dim: 339 x 109
```

- 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 naïve classifier that predicts a random label for each image?

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.

Answer

```
library("e1071")
train.df <- data.frame(train.score.90, label = labels_train)
test.df <- data.frame(test.score.90, label = labels_test)

# tuning for best parameters
rbf.tune <- tune(svm,
                 label ~ .,
                 data = train.df,
                 kernel = "radial",
                 ranges = list(gamma = 10^(-5:1), cost = 10^(-5:1)),
                 tunecontrol = tune.control(cross = 5))
rbf.tune
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 5-fold cross validation
##
## - best parameters:
##  gamma cost
##  0.001   10
##
## - best performance: 0.1032924
```

```
rbf.tune$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = label ~ ., data = train.df,
##     ranges = list(gamma = 10^(-5:1), cost = 10^(-5:1)), tunecontrol = tune.control(cross = 5),
##     kernel = "radial")
##
##
```

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.001
##
## Number of Support Vectors:  191
```

We tune the parameters (gamma, cost) of the rbf svm and find the best gamma is 0.001, and the best costis 10. In the following, we use these parameters and perform svm on the training set.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
# prediction on train
pred.rbf.train <- predict(rbf.tune$best.model, newdata = train.df)
confusionMatrix(pred.rbf.train, train.df$label)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Ariel_Sharon George_W_Bush Hugo_Chavez
##   Ariel_Sharon            37             0           0
##   George_W_Bush            1           266           1
##   Hugo_Chavez              1             0          33
##
## Overall Statistics
##
##                Accuracy : 0.9912
##                  95% CI : (0.9744, 0.9982)
##     No Information Rate : 0.7847
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9752
##  Mcnemar's Test P-Value : 0.3916
##
## Statistics by Class:
##
##                      Class: Ariel_Sharon Class: George_W_Bush
## Sensitivity                       0.9487               1.0000
## Specificity                       1.0000               0.9726
## Pos Pred Value                    1.0000               0.9925
## Neg Pred Value                    0.9934               1.0000
## Prevalence                        0.1150               0.7847
## Detection Rate                    0.1091               0.7847
## Detection Prevalence              0.1091               0.7906
## Balanced Accuracy                 0.9744               0.9863
##                      Class: Hugo_Chavez
## Sensitivity                     0.97059
## Specificity                     0.99672
## Pos Pred Value                  0.97059
## Neg Pred Value                  0.99672
## Prevalence                      0.10029
```

```
## Detection Rate                 0.09735
## Detection Prevalence           0.10029
## Balanced Accuracy              0.98365
```

The prediction accuracy on the train set is very high, at 99.12%. It is able to predict all three classes well with almost perfect accuracy.

```
# prediction on test
pred.rbf.test <- predict(rbf.tune$best.model, newdata = test.df)
confusionMatrix(pred.rbf.test, test.df$label)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Ariel_Sharon George_W_Bush Hugo_Chavez
##   Ariel_Sharon            27             0           0
##   George_W_Bush           10           263           9
##   Hugo_Chavez              1             1          28
##
## Overall Statistics
##
##                Accuracy : 0.9381
##                  95% CI : (0.9069, 0.9612)
##     No Information Rate : 0.7788
##     P-Value [Acc > NIR] : 9.385e-16
##
##                   Kappa : 0.8143
##  Mcnemar's Test P-Value : 0.0005847
##
## Statistics by Class:
##
##                      Class: Ariel_Sharon Class: George_W_Bush
## Sensitivity                      0.71053               0.9962
## Specificity                      1.00000               0.7467
## Pos Pred Value                   1.00000               0.9326
## Neg Pred Value                   0.96474               0.9825
## Prevalence                       0.11209               0.7788
## Detection Rate                   0.07965               0.7758
## Detection Prevalence             0.07965               0.8319
## Balanced Accuracy                0.85526               0.8714
##                      Class: Hugo_Chavez
## Sensitivity                      0.7568
## Specificity                      0.9934
## Pos Pred Value                   0.9333
## Neg Pred Value                   0.9709
## Prevalence                       0.1091
## Detection Rate                   0.0826
## Detection Prevalence             0.0885
## Balanced Accuracy                0.8751
```

The prediction accuracy on the test set is fairly high, at an overall accuracy of 93.81%. It is able to predict George_W_Bush with a very high accuracy of 99.6%, Hugo_Chavez with an accuracy of 75.7%, and Ariel_Sharon with an accuracy of 71%.

```
# Naive classifier with random label
set.seed(123)
```

```r
random.vector <- runif(dim(test.df)[1])
naive <- rep('na', dim(test.df)[1])
naive[random.vector < 1/3] = 'Ariel_Sharon'
naive[random.vector >= 1/3 & random.vector < 2/3] = 'George_W_Bush'
naive[random.vector > 2/3] = 'Hugo_Chavez'

confusionMatrix(naive, test.df$label)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Ariel_Sharon George_W_Bush Hugo_Chavez
##   Ariel_Sharon            10            90          11
##   George_W_Bush           11            94          13
##   Hugo_Chavez             17            80          13
##
## Overall Statistics
##
##                Accuracy : 0.3451
##                  95% CI : (0.2946, 0.3984)
##     No Information Rate : 0.7788
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.003
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: Ariel_Sharon Class: George_W_Bush
## Sensitivity                      0.26316               0.3561
## Specificity                      0.66445               0.6800
## Pos Pred Value                   0.09009               0.7966
## Neg Pred Value                   0.87719               0.2308
## Prevalence                       0.11209               0.7788
## Detection Rate                   0.02950               0.2773
## Detection Prevalence             0.32743               0.3481
## Balanced Accuracy                0.46380               0.5180
##                      Class: Hugo_Chavez
## Sensitivity                     0.35135
## Specificity                     0.67881
## Pos Pred Value                  0.11818
## Neg Pred Value                  0.89520
## Prevalence                      0.10914
## Detection Rate                  0.03835
## Detection Prevalence            0.32448
## Balanced Accuracy               0.51508
```

The naive classifier predicts with an overall accuracy of 34.51%, and approximately 33% accuracy on each of the three classes. This is expected as we have set up the naive classifier to predict randomly one of the three classes. Both the overall and class-specific accuracy of prediction by the naive classifier is significantly lower than the prediction accuracy of the svm on the test set.

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.

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).
- 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).
- 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.

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?

```
df <- read.csv("CS109b-hw5-dataset_2.txt", sep = " ")
dim(df)
```

```
## [1] 50  6
```

```
df
```

```
##                    X1980       X1976      X1972       X1968
## Alabama       -0.027201677  0.268399734 -1.0423335  0.28946619
## Alaska        -0.721785159 -0.484881901 -0.5180856 -0.06145278
## Arizona       -0.763439407 -0.347866025 -0.7072806 -0.44834213
## Arkansas      -0.012800175  0.619918445 -0.8071413 -0.01307208
## California    -0.383429543 -0.036804154 -0.2806273 -0.06705214
## Colorado      -0.572327240 -0.238727346 -0.5932042 -0.20111083
## Connecticut   -0.223323553 -0.104494980 -0.3780388  0.11098799
## Delaware      -0.050810929  0.110111118 -0.4192343 -0.08078208
## Florida       -0.366032732  0.107302838 -0.9504218 -0.27054579
## Georgia        0.308831427  0.705472603 -1.1136019 -0.12977904
## Hawaii         0.043206720  0.051211189 -0.5099724  0.43516606
## Idaho         -0.969902181 -0.478103737 -0.9031676 -0.61527367
## Illinois      -0.174037963 -0.040005335 -0.3763812 -0.06354821
## Indiana       -0.397138665 -0.154305443 -0.6847589 -0.28041892
## Iowa          -0.284299553 -0.020800750 -0.3528148 -0.26160983
## Kansas        -0.552864840 -0.155512656 -0.8297402 -0.45695051
## Kentucky      -0.030002250  0.146260177 -0.6001910 -0.15262977
## Louisiana     -0.112518561  0.118538608 -0.8344735  0.18232155
## Maine         -0.076437195 -0.017200424 -0.4679566  0.24924991
```

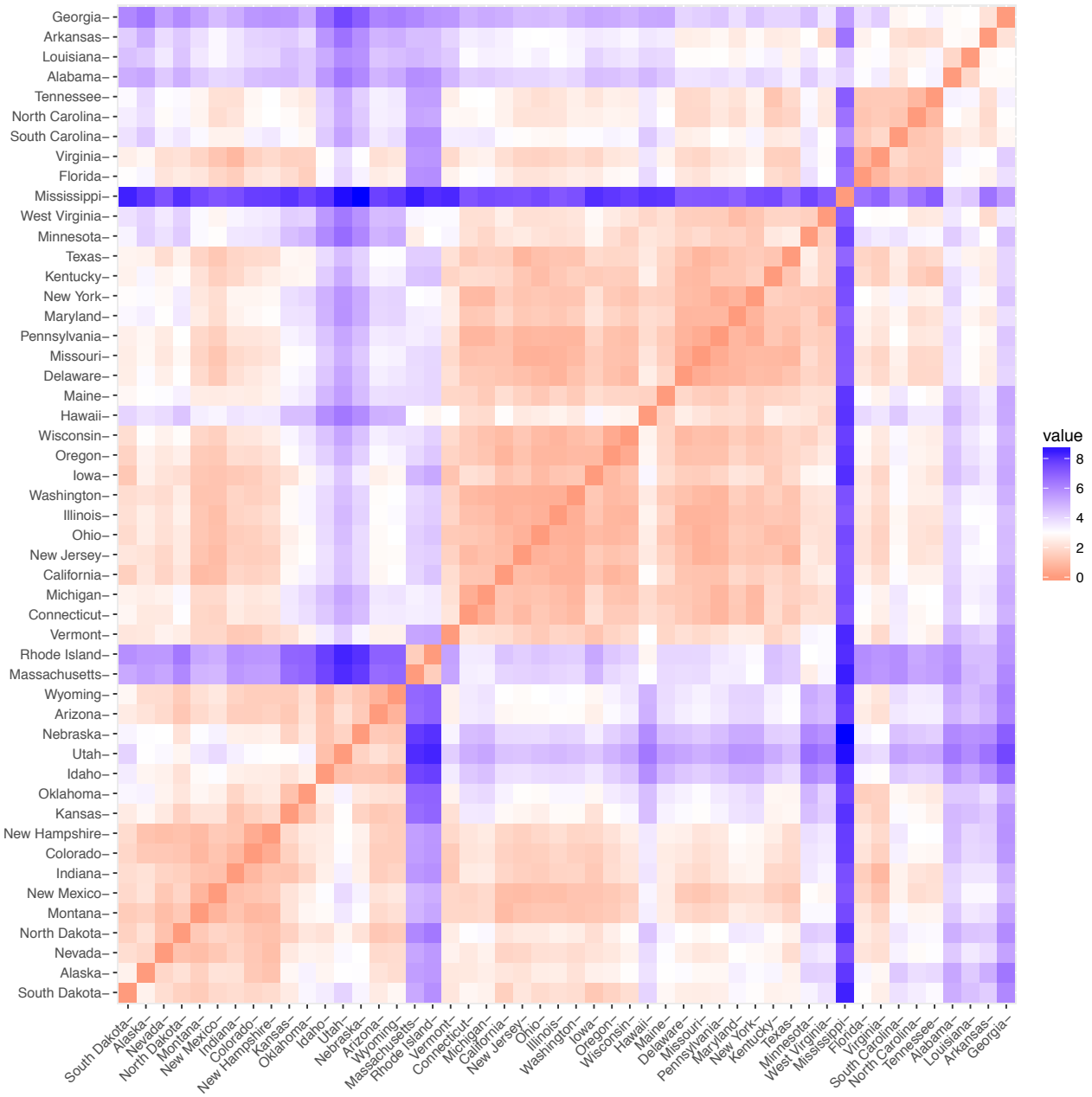
```
## Maryland        0.064822689  0.121750170 -0.4946447  0.03977132
## Massachusetts  -0.003600004  0.327290631  0.1808916  0.64966207
## Michigan       -0.142239332 -0.109709909 -0.2957367  0.14966559
## Minnesota       0.088457635  0.267178092 -0.1129198  0.26329062
## Mississippi    -0.027201677  0.038804869 -1.3825486  0.53280453
## Missouri       -0.143043408  0.073633251 -0.5018756 -0.02708969
## Montana        -0.561069648 -0.151489046 -0.4255059 -0.19585141
## Nebraska       -0.923231066 -0.431366798 -0.8712224 -0.63153937
## Nevada         -0.843966745 -0.090862462 -0.5615020 -0.18950519
## New Hampshire  -0.711352754 -0.230616687 -0.6071920 -0.17125063
## New Jersey     -0.298599020 -0.044007101 -0.5155217 -0.04662332
## New Mexico     -0.401716509 -0.049610172 -0.5129595 -0.26603896
## New York       -0.058816950  0.088858421 -0.3511648  0.11502026
## North Carolina -0.044007101  0.222918554 -0.8774812 -0.30213196
## North Dakota   -0.894412900 -0.120144312 -0.5507087 -0.38072886
## Ohio           -0.230616687  0.005600015 -0.4485734 -0.05222526
## Oklahoma       -0.548122987 -0.024801271 -1.1222185 -0.39919550
## Oregon         -0.222918554 -0.003600004 -0.2144177 -0.12838117
## Pennsylvania   -0.154707835  0.054413422 -0.4125535  0.07864313
## Rhode Island    0.248064292  0.227779753 -0.1241593  0.69941679
## South Carolina -0.031602630  0.264328363 -0.9271661 -0.25243992
## South Dakota   -0.646848818 -0.029602161 -0.1736349 -0.23826671
## Tennessee      -0.006000018  0.264328363 -0.8221852 -0.29653953
## Texas          -0.288790494  0.064021859 -0.6888003  0.02963180
## Utah           -1.263918977 -0.618160203 -0.9409920 -0.59703282
## Vermont        -0.144249609 -0.230616687 -0.5412364 -0.19375025
## Virginia       -0.274510951 -0.027601752 -0.8118332 -0.28921935
## Washington     -0.285932291 -0.080844003 -0.3871648  0.04551165
## West Virginia   0.094871093  0.325647618 -0.5584767  0.19530875
## Wisconsin      -0.103692811  0.034403393 -0.1998626 -0.07813083
## Wyoming        -0.806203951 -0.398386759 -0.8174746 -0.45224117
##                      X1964        X1960
## Alabama        -0.82124238  0.290832784
## Alaska          0.65928609 -0.037604431
## Arizona        -0.02000067 -0.224538660
## Arkansas        0.25578553  0.153098341
## California      0.37099755 -0.010800105
## Colorado        0.47260441 -0.196227231
## Connecticut     0.74826255  0.149477702
## Delaware        0.45193837  0.032802941
## Florida         0.04600811 -0.060418372
## Georgia        -0.16557725  0.513386445
## Hawaii          1.31051906  0.001200000
## Idaho           0.03680415 -0.151489046
## Illinois        0.38342954  0.003600004
## Indiana         0.25090809 -0.210372445
## Iowa            0.48997270 -0.271658423
## Kansas          0.18250489 -0.435557650
## Kentucky        0.58536084 -0.143847530
## Louisiana      -0.27410338  0.567127080
## Maine           0.79264975 -0.283891428
## Maryland        0.63976351  0.144651699
## Massachusetts   1.17862230  0.420487924
## Michigan        0.70095737  0.040405496
```

```
## Minnesota       0.57146003  0.028801991
## Mississippi    -1.91339430  1.116290531
## Missouri        0.57753494  0.010400094
## Montana         0.37389578 -0.050410673
## Nebraska        0.10449498 -0.492520425
## Nevada          0.34662949  0.046408327
## New Hampshire   0.57059302 -0.137013943
## New Jersey      0.66151219  0.016000341
## New Mexico      0.38633452  0.014800270
## New York        0.78380687  0.105698296
## North Carolina  0.24725196  0.084450155
## North Dakota    0.32523693 -0.218869560
## Ohio            0.52964367 -0.131388688
## Oklahoma        0.23102204 -0.364792227
## Oregon          0.57189361 -0.104896077
## Pennsylvania    0.62652016  0.046408327
## Rhode Island    1.44158514  0.559340798
## South Carolina -0.35941990  0.049610172
## South Dakota    0.22534882 -0.331400085
## Tennessee       0.22129874 -0.145053801
## Texas           0.55113983  0.040405496
## Utah            0.19501571 -0.193400580
## Vermont         0.67713966 -0.349515154
## Virginia        0.14786884 -0.110111118
## Washington      0.50570889 -0.048809688
## West Virginia   0.75101582  0.109308709
## Wisconsin       0.49804590 -0.074434348
## Wyoming         0.26392135 -0.201074744
```

```
## Rescale the data, and compute the Euclidean distance between each pair of states. Generate a heat ma
library(cluster)
library(factoextra)
library(ggplot2)

# Compute Euclidean distance
df.dist <- daisy(scale(df), metric = "euclidean")

# Heatmap
fviz_dist(df.dist)
```



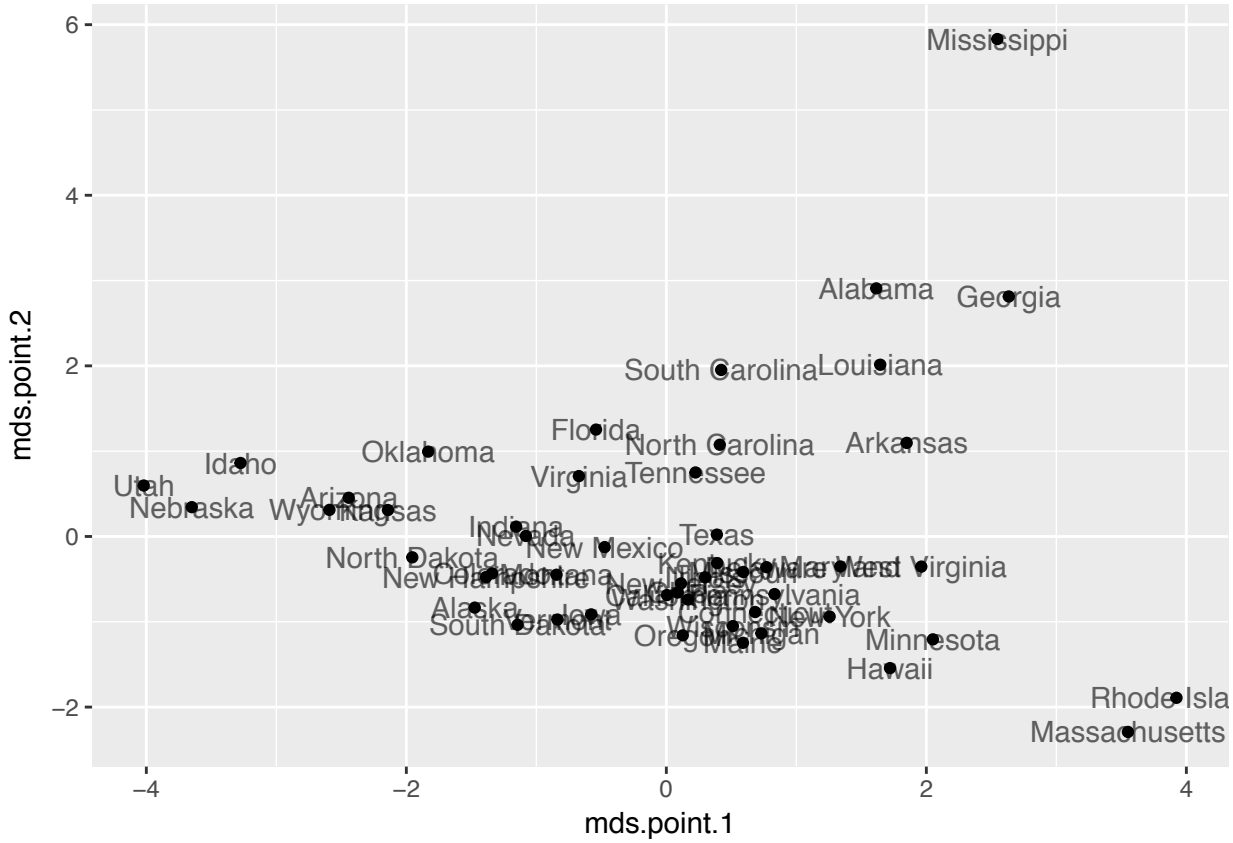
The heatmap color tells us the distances between different states. The further the distance, the more different a pair of states are from each other, vice versa. From the heatmap, we can see that pairs that are more blue are further from each other, while those pairs that are more red are similar to each other. For example, we see that Mississippi is far apart from all other states; Rhoad Island and Massachusetts are very different from states like South Dakota, Alaska, Utah, Nebraska, etc., but are similar to each other, States such as New Jersey, Ohio, Illinois, Washington, etc are similar to one another and are likely to be in one cluster.

From the heatmap, we can group the states into roughly three clusters. Perhaps it is most obvious to concentrate on the rows of Rhoad Island and Massachusetts - they make three different regions/colors with the rest of the states: 1. the blue, which are states like Utah, Nerbraska, indicating they belong to a different cluster; 2. the red, which are themselves, and 3. the purple (in between red and blue), which are states such as Kentucky and Missouri.

```
## Apply multi-dimensional scaling to the pair-wise distances, and generate a scatter plot of the states
```

```
# multi-dimensional scaling
df.mds <- cmdscale(df.dist)
df_mds <- data.frame(df, mds.point = df.mds)

# scatter plot
ggplot(df_mds, mapping = aes(x = mds.point.1, y = mds.point.2)) +
  geom_point()+
  geom_text(aes(label = rownames(df.mds)), alpha = 0.6)
```



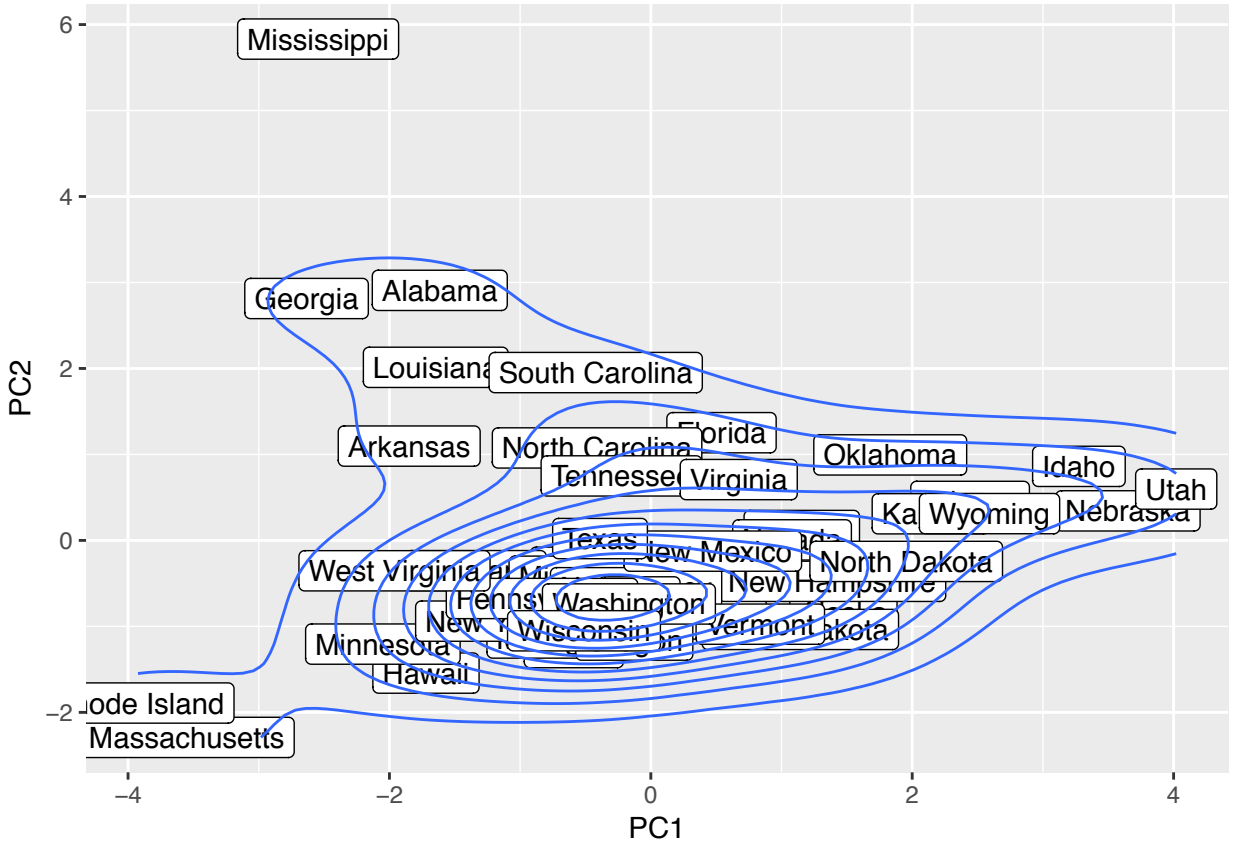
From the scatter plot, we can see that Mississippi is a clear outlier that is different from everyone else. Rhode Island and Massachusetts are also further away from the rest of the states and may belong to one cluster. Some states such as Oregon, New York, Pennsylvania, etc. are clustered tightly together in the center of the scatter plot and likely form one cluster. Other states such as Idaho, Oklahoma, Louisianna, Arkansas are away from the center cluster, and likely form another cluster.

```
## Apply PCA to the data, and generate a scatter plot of the states using the first two principal compor

# PCA of data
df.pca <- prcomp(df, scale = TRUE)
summary(df.pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     1.7245 1.3762 0.8130 0.46787 0.40220 0.30091
## Proportion of Variance 0.4956 0.3157 0.1102 0.03648 0.02696 0.01509
## Cumulative Proportion  0.4956 0.8113 0.9215 0.95795 0.98491 1.00000
```

```
df.pca.asdf <- data.frame(df.pca$x)
ggplot(df.pca.asdf, aes(x = PC1, y = PC2))+
  geom_point()+
  geom_label(aes(label = rownames(df.pca.asdf)))+
  geom_density_2d()
```




```
# Scatter plot of first two PC, and 2d-density estimation overlay
```

The contour plot here conveys that states that are one the same contour lines are similar. States like Washington, Wisconsin, New York, Pennsylvania, etc are closely clustered together, and likely form one cluster. Other states that sit on contour lines further away from the center likely form another cluster. Rhode Island and Massachusetts likely form another cluster, whereas we see that Mississippi looks like an outlier that is far away from every other state.

From all the data visualizations above, it looks reasonable that we may find three clusters in this data set.

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?

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.

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).

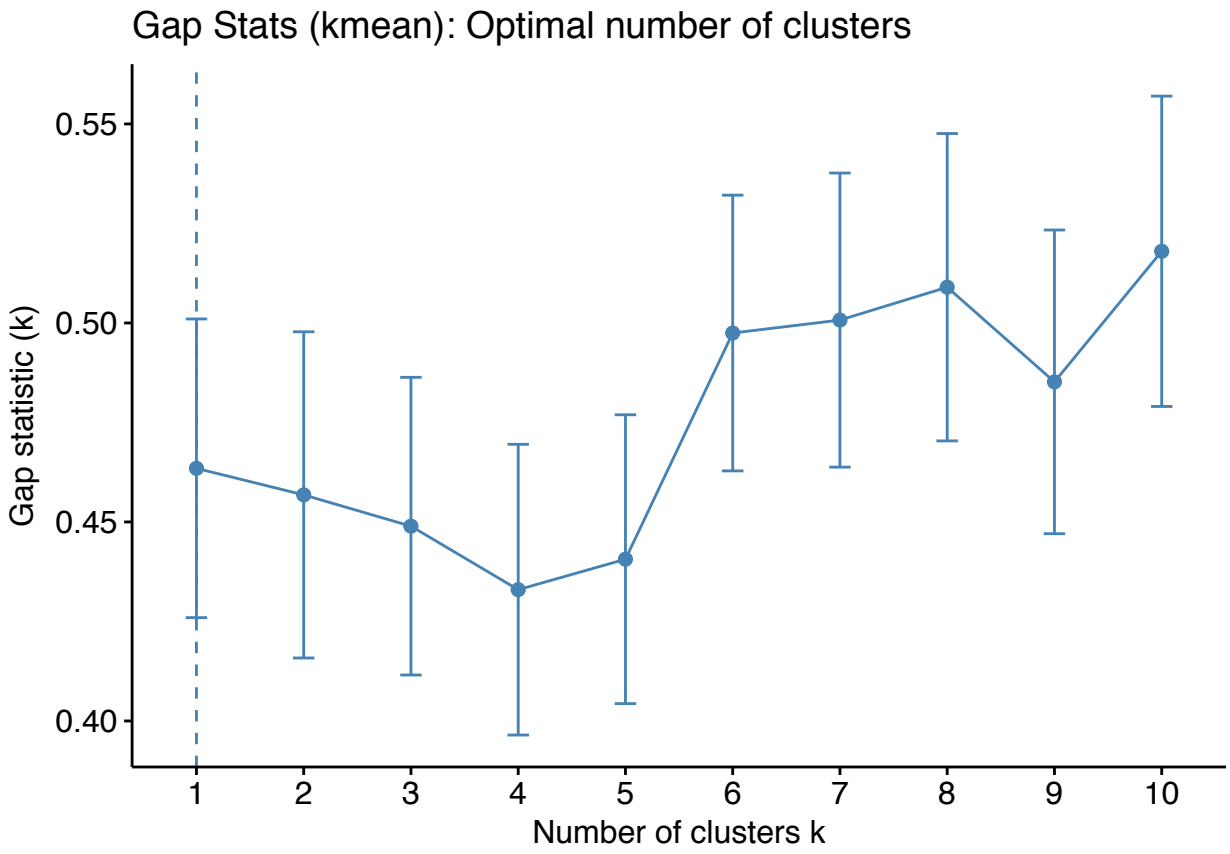
Answer

(1) Identidy the optimal number of clusters

```
## Choice of number of clusters using Gap statistics
gapstat <- clusGap(scale(df), FUN=kmeans, K.max=10, B=500)
print(gapstat, method="Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = scale(df), FUNcluster = kmeans, K.max = 10, B = 500)
## B=500 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 1
##           logW   E.logW       gap     SE.sim
##  [1,] 3.620266 4.083742 0.4634758 0.03752363
##  [2,] 3.402462 3.859266 0.4568043 0.04097650
##  [3,] 3.235942 3.684886 0.4489445 0.03739458
##  [4,] 3.129812 3.562799 0.4329866 0.03652712
##  [5,] 3.038095 3.478737 0.4406418 0.03628760
##  [6,] 2.909188 3.406661 0.4974733 0.03463772
##  [7,] 2.837302 3.338026 0.5007245 0.03694968
##  [8,] 2.765288 3.274270 0.5089820 0.03861255
##  [9,] 2.729238 3.214437 0.4851988 0.03815378
## [10,] 2.640875 3.158882 0.5180077 0.03898948
```

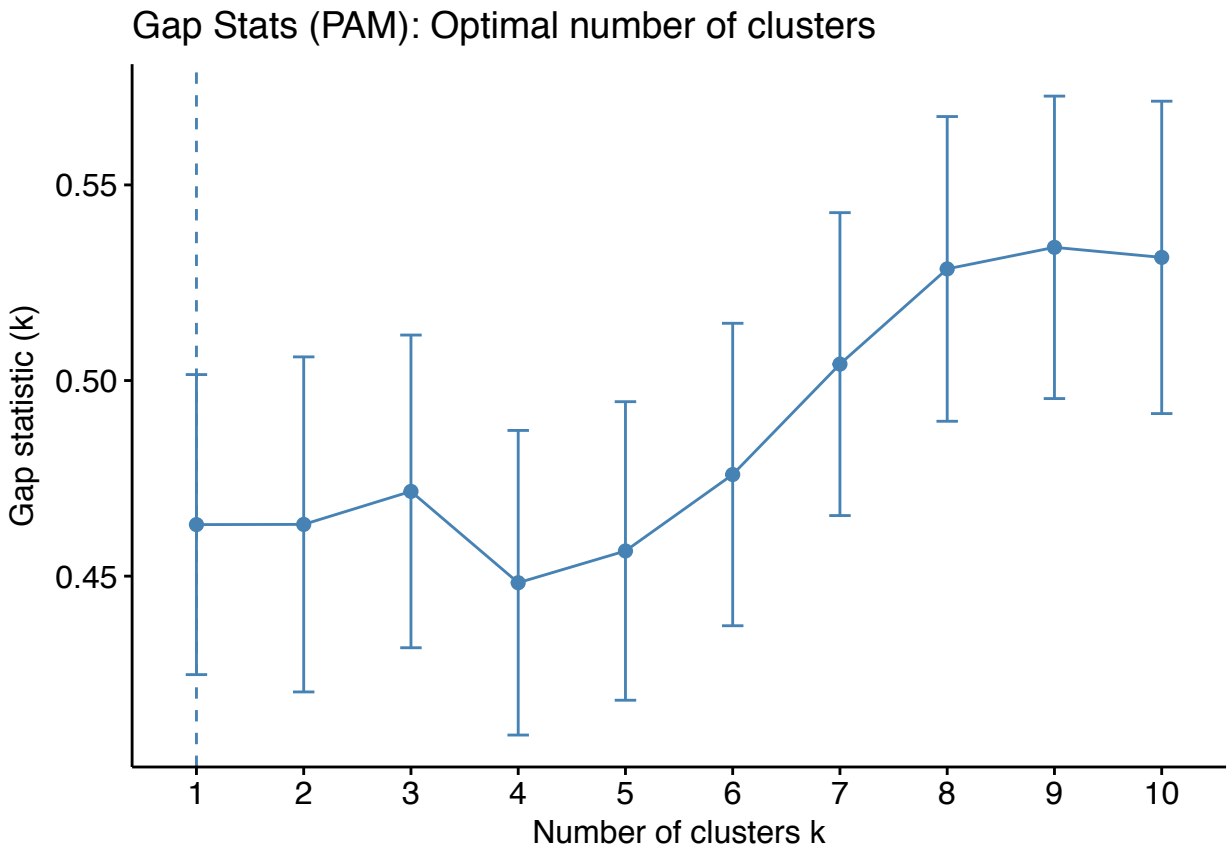
```
fviz_gap_stat(gapstat,
  maxSE=list(method="Tibs2001SEmax",SE.factor=1)) +
  ggtitle("Gap Stats (kmean): Optimal number of clusters")
```



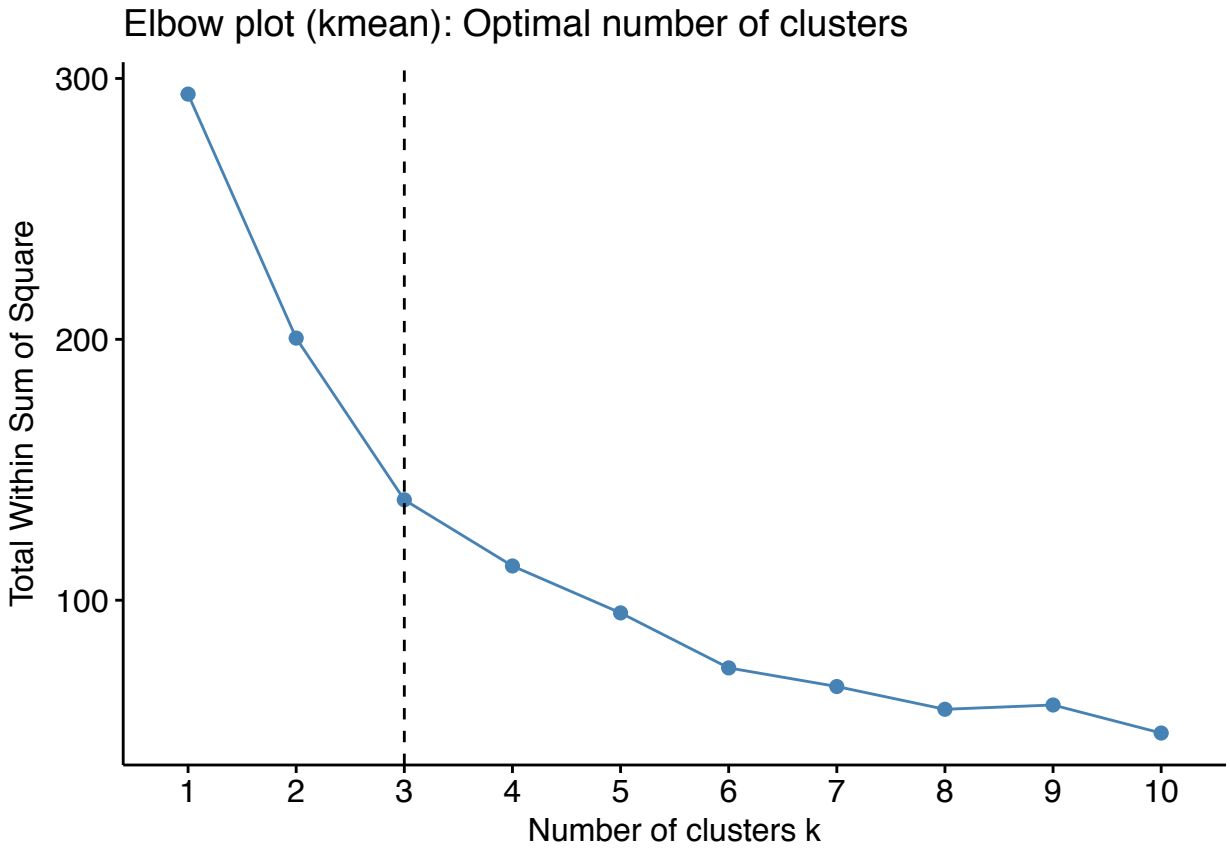

```
gapstat <- clusGap(scale(df), FUN=pam, K.max=10, B=500)
print(gapstat, method="Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = scale(df), FUNcluster = pam, K.max = 10, B = 500)
## B=500 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 1
##           logW   E.logW       gap     SE.sim
##  [1,] 3.620266 4.083443 0.4631769 0.03834946
##  [2,] 3.405836 3.869061 0.4632250 0.04281975
##  [3,] 3.225686 3.697350 0.4716641 0.03996395
##  [4,] 3.130698 3.579011 0.4483128 0.03893140
##  [5,] 3.036721 3.493160 0.4564393 0.03815619
##  [6,] 2.940884 3.416859 0.4759753 0.03865398
##  [7,] 2.843135 3.347333 0.5041981 0.03871212
##  [8,] 2.754444 3.282978 0.5285342 0.03894172
##  [9,] 2.687968 3.222013 0.5340455 0.03865635
## [10,] 2.631513 3.162988 0.5314751 0.03992107
```

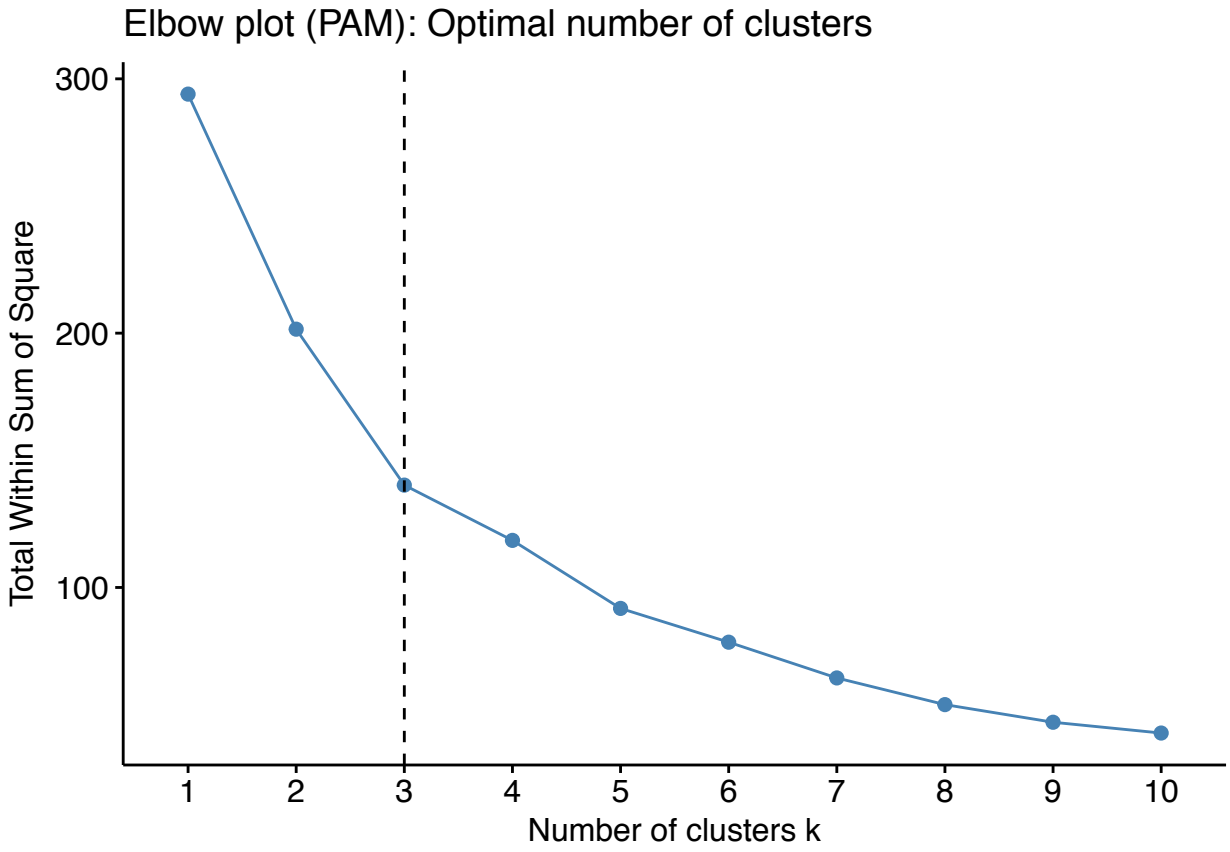
```
fviz_gap_stat(gapstat,
  maxSE=list(method="Tibs2001SEmax",SE.factor=1)) +
  ggtitle("Gap Stats (PAM): Optimal number of clusters")
```



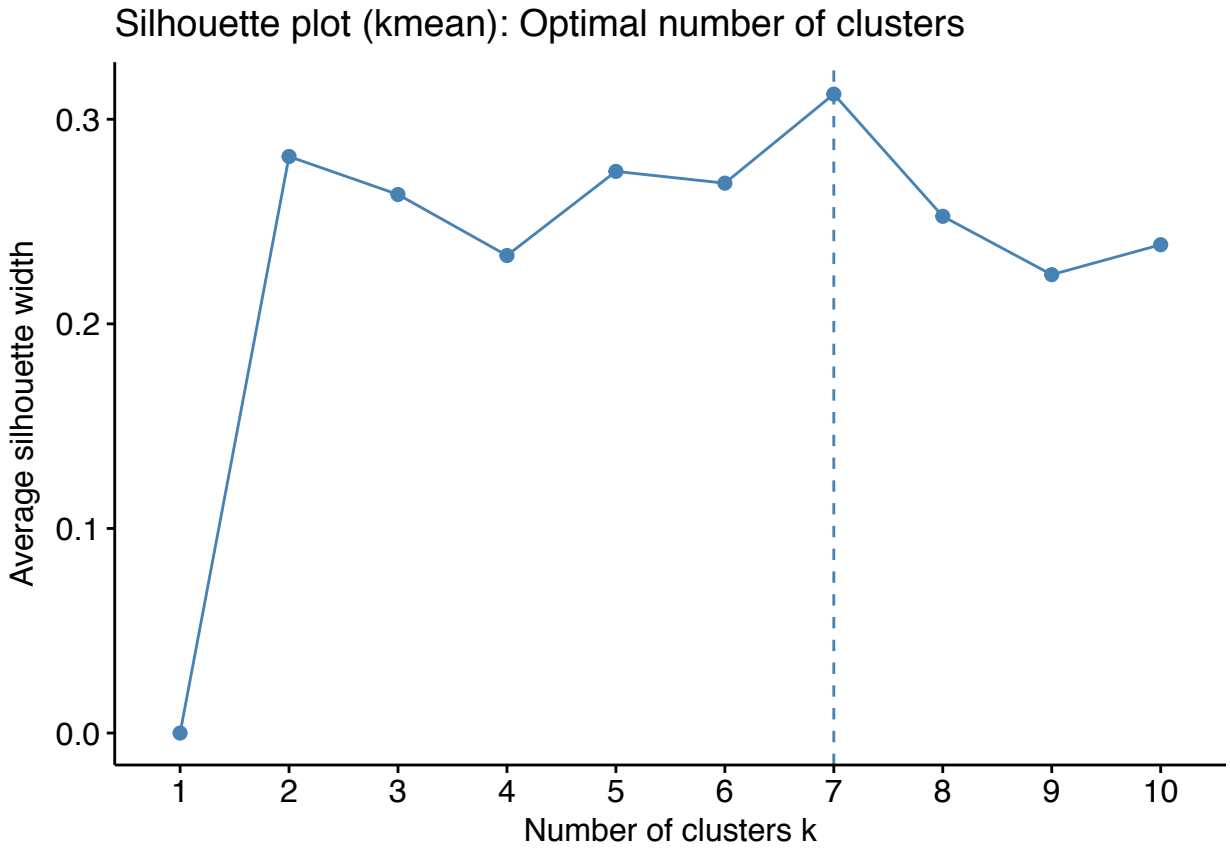

```
## Choice of number of clusters with elbow plots (fviz_nbclust)
fviz_nbclust(scale(df), kmeans, method="wss") +
  ggtitle("Elbow plot (kmean): Optimal number of clusters")+
 geom_vline(xintercept=3,linetype=2)
```



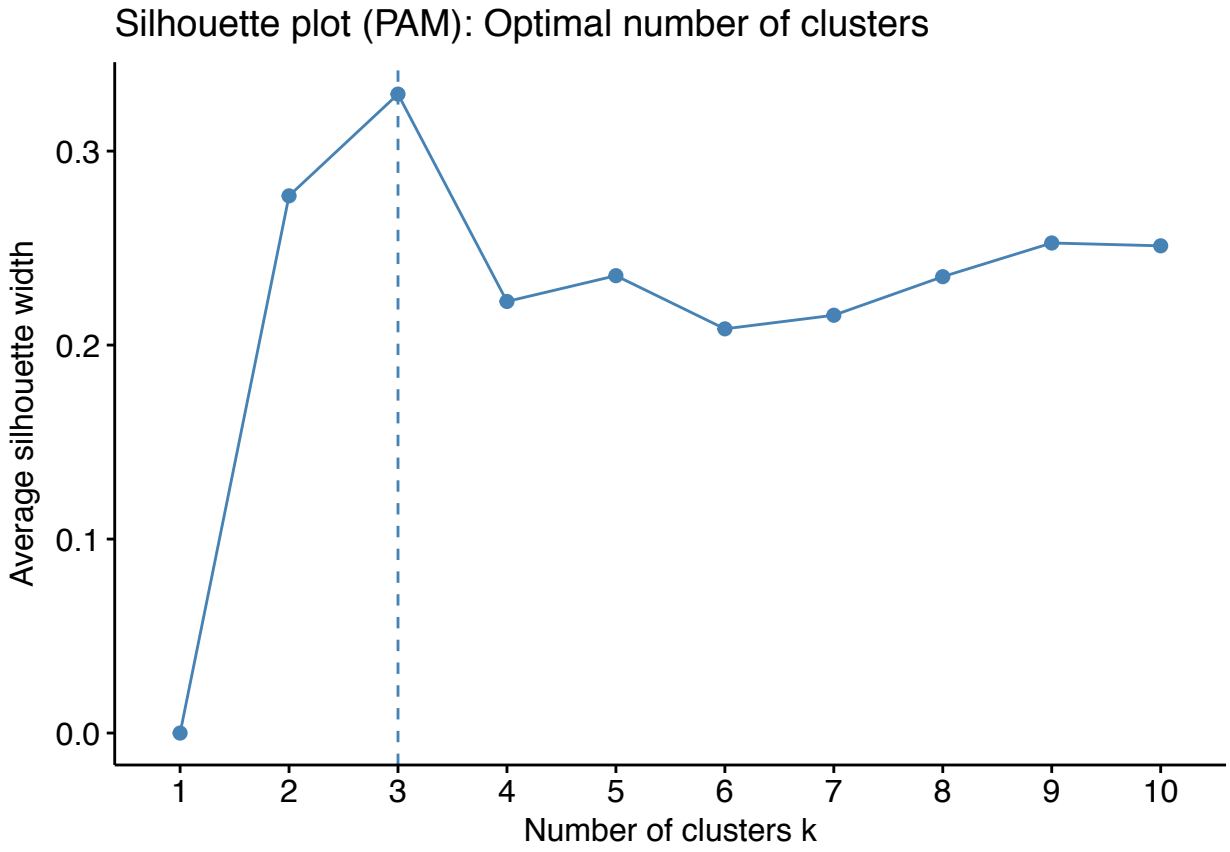

```
fviz_nbclust(scale(df), pam, method="wss") +
  ggtitle("Elbow plot (PAM): Optimal number of clusters")+
 geom_vline(xintercept=3,linetype=2)
```




```
## Choice of number of clusters with method of average silhouette widths (fviz_nbclust, method="silhoue
fviz_nbclust(scale(df),kmeans,method="silhouette") +
  ggtitle("Silhouette plot (kmean): Optimal number of clusters")
```

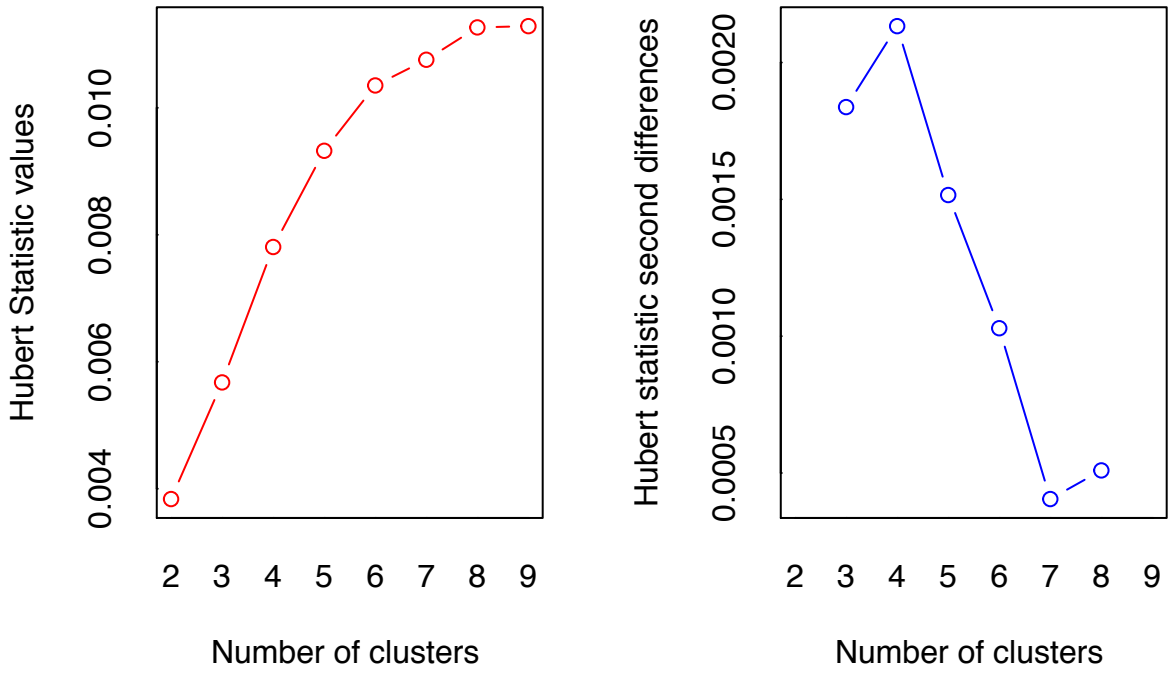



```
fviz_nbclust(scale(df),pam,method="silhouette") +
  ggtitle("Silhouette plot (PAM): Optimal number of clusters")
```

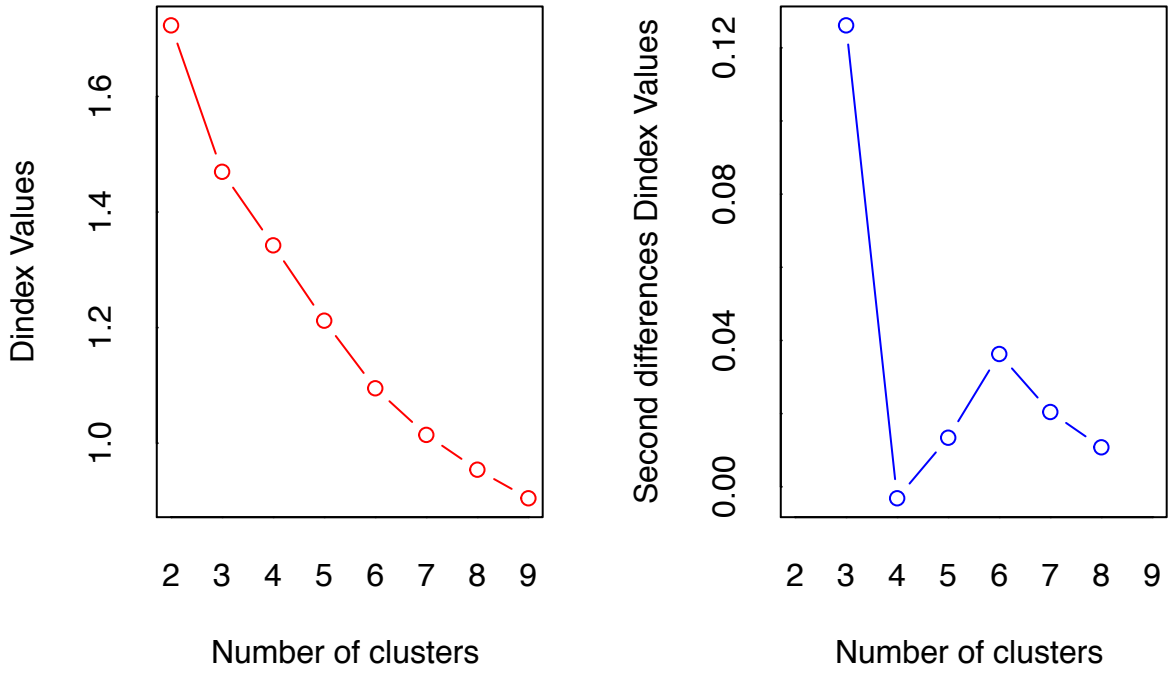



```
# as a cross-check (not required in question), we use NBClust to check the most popular number of clust
library(NbClust)

nb.df = NbClust(scale(df),distance="euclidean",
  min.nc=2, max.nc=9, method="ward.D2",index="all")
```



```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```




```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 3 proposed 2 as the best number of clusters
## * 9 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 4 proposed 5 as the best number of clusters
## * 3 proposed 7 as the best number of clusters
## * 3 proposed 9 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  3
##
##
## *******************************************************************
```

```
print(nb.df)
```

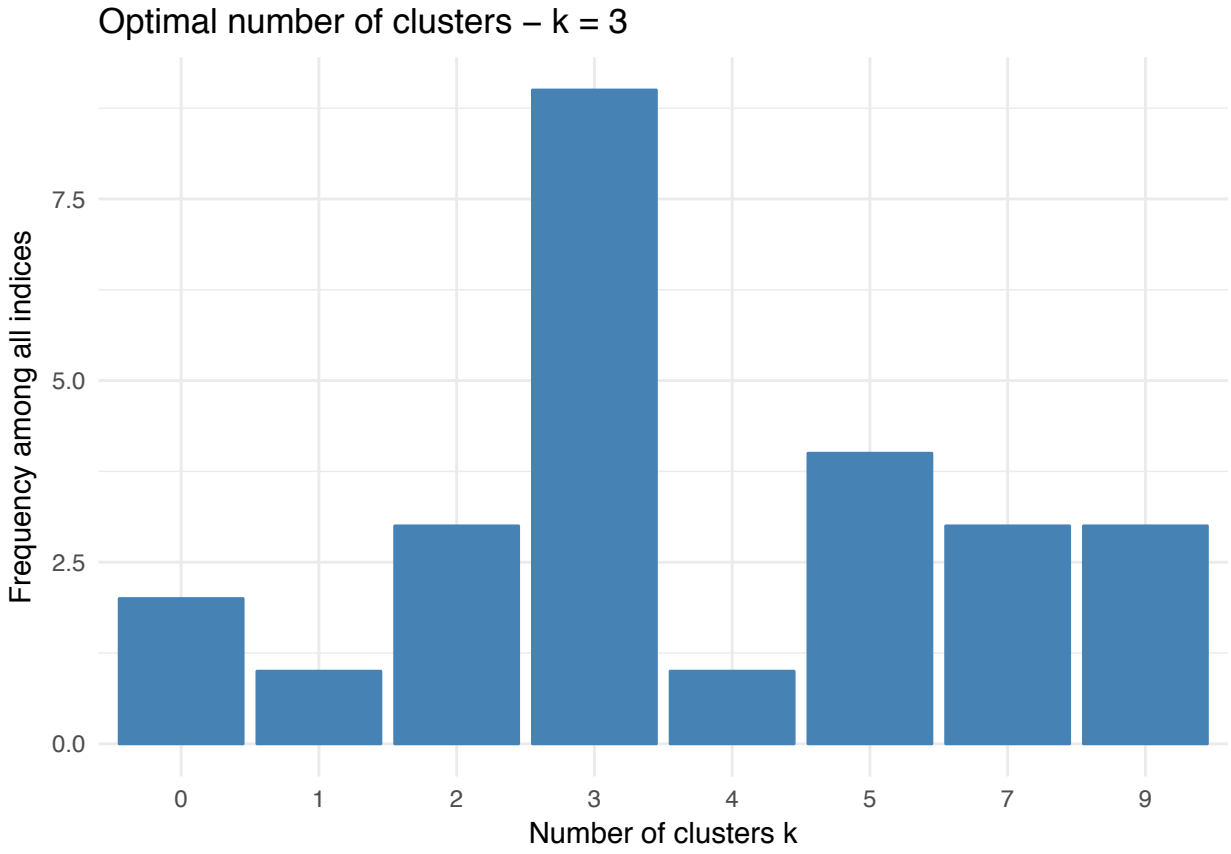
```
## $All.index
##        KL      CH Hartigan     CCC    Scott   Marriot    TrCovW   TraceW
## 2 0.7006 20.9116  20.6046 -2.0829  66.1782 175926712 1855.2832 204.7841
## 3 1.9481 24.7204  12.2962 -1.9987 129.8122 110865534  562.9388 143.2796
## 4 0.9159 24.3610  12.9693 -1.6954 191.2690  57658913  201.6520 113.5678
```

```
## 5 1.5629 26.0847   9.4397 -0.4211 225.3112  45603703  170.9732  88.5905
## 6 1.1355 26.5302   8.7510  0.2409 267.0849  28478722  166.5826  73.2292
## 7 1.8449 27.3285   5.6158  0.9148 305.2750  18059253  113.8151  61.0811
## 8 0.9977 26.6513   5.5610  1.0024 330.7105  14182518   91.7242  54.0254
## 9 1.0068 26.4574   5.5736  1.1942 382.7297   6341990   70.0949  47.7085
##   Friedman  Rubin Cindex     DB Silhouette   Duda Pseudot2   Beale
## 2  11.2624 1.4357 0.2722 1.2954     0.2778 0.6383  18.1293  2.1136
## 3  16.3400 2.0519 0.2403 1.2196     0.3315 1.3285  -1.9782 -0.8456
## 4  21.0715 2.5888 0.3103 0.9307     0.3529 0.5293  19.5661  3.2729
## 5  24.8230 3.3186 0.3428 0.7984     0.3666 0.5575  11.1101  2.8496
## 6  28.8526 4.0148 0.3067 0.8791     0.3146 0.5289   6.2346  2.9983
## 7  33.0764 4.8133 0.3752 0.8899     0.3178 0.7383   7.0881  1.2986
## 8  36.2964 5.4419 0.3439 1.0752     0.2436 0.3763   3.3154  4.2518
## 9  48.3308 6.1624 0.3327 0.9770     0.2577 0.4400   6.3643  4.0809
##   Ratkowsky     Ball Ptbiserial    Frey McClain   Dunn Hubert SDindex
## 2    0.3433 102.3920     0.2874 -0.0679  0.5864 0.1004 0.0038  2.0211
## 3    0.4101  47.7599     0.4611 -0.3219  1.0011 0.1203 0.0057  1.8482
## 4    0.3906  28.3920     0.4899 -0.0559  0.9858 0.1599 0.0078  1.5005
## 5    0.3734  17.7181     0.5419  0.6285  1.0263 0.2015 0.0093  1.3296
## 6    0.3534  12.2049     0.5278  0.1745  1.2438 0.2015 0.0104  1.5218
## 7    0.3362   8.7259     0.5353  2.4812  1.2887 0.2572 0.0108  1.5160
## 8    0.3193   6.7532     0.4136  0.0562  2.4262 0.2006 0.0113  2.4139
## 9    0.3050   5.3009     0.4163  0.2385  2.4354 0.2006 0.0113  2.3178
##   Dindex   SDbw
## 2 1.7230 1.2661
## 3 1.4697 0.8594
## 4 1.3424 0.3368
## 5 1.2120 0.2568
## 6 1.0951 0.2308
## 7 1.0144 0.2118
## 8 0.9542 0.2002
## 9 0.9048 0.1672
##
## $All.CriticalValues
##   CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
## 2         0.5992            21.4021       0.0535
## 3         0.3506            14.8210       1.0000
## 4         0.5432            18.5029       0.0049
## 5         0.4643            16.1499       0.0142
## 6         0.3212            14.7957       0.0157
## 7         0.5276            17.9093       0.2630
## 8         0.0348            55.4763       0.0159
## 9         0.2445            15.4517       0.0041
##
## $Best.nc
##                       KL      CH Hartigan    CCC  Scott  Marriot   TrCovW
## Number_clusters 3.0000  7.0000   3.0000 9.0000  3.000        4    3.000
## Value_Index     1.9481 27.3285   8.3083 1.1942 63.634 41151410 1292.344
##                   TraceW Friedman   Rubin Cindex     DB Silhouette   Duda
## Number_clusters  3.0000   9.0000  7.0000 3.0000 5.0000     5.0000 2.0000
## Value_Index     31.7926  12.0344 -0.1699 0.2403 0.7984     0.3666 0.6383
##                 PseudoT2   Beale Ratkowsky    Ball PtBiserial Frey McClain
## Number_clusters   2.0000  3.0000    3.0000  3.0000     5.0000    1  2.0000
## Value_Index      18.1293 -0.8456    0.4101 54.6322     0.5419   NA  0.5864
```

```
##                   Dunn Hubert SDindex Dindex   SDbw
## Number_clusters 7.0000      0  5.0000      0 9.0000
## Value_Index     0.2572      0  1.3296      0 0.1672
##
## $Best.partition
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              2              2              1              3
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              2              3              3              1              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              3              2              3              2              3
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              2              3              1              3              3
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              3              3              3              1              3
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              2              2              2              2              3
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              2              3              1              2              3
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              3              3              3              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              2              1              3              2              3
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              1              3              3              3              2
```

```
fviz_nbclust(nb.df) + theme_minimal()
```

```
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 1 proposed  1 as the best number of clusters
## * 3 proposed  2 as the best number of clusters
## * 9 proposed  3 as the best number of clusters
## * 1 proposed  4 as the best number of clusters
## * 4 proposed  5 as the best number of clusters
## * 3 proposed  7 as the best number of clusters
## * 3 proposed  9 as the best number of clusters
##
## Conclusion
## =========================
## * According to the majority rule, the best number of clusters is  3 .
```



- Gap statistics: for both kmeans and pam, gap statistics shows that 1 cluster is the optimal number of clusters. This is because we have set `spaceH0` to the default, which compresses the data space into a hypercube - this results in a preference for in a smaller number of clusters since the algorithm ignores small differences in certain dimensions and consider possible different clusters as one cluster. If we were to set `spaceH0` to `original`, which considers all dimentions of the data space - this will result in a prefernce for a larger number of clusters, as the algorithm is able to differentiate between clusters that are slightly in and out of planes from one another.
- Knee plot: For both kemans and pam, we read the knee plots and identify the number of clusters where there is a first change in gradient. In both cases, the method has identified 3 clusters to be the optimal number of clusters.
- Silhouette plot: The kmeans method idenfities the optimal cluster as 7, this is different from the rest of the predictions. The pam method identifies the optimal cluster as 3, which is consistent with the prediction from the knee plot. Even though the kmeans method identifies the optimal number of clusters as k = 7, the average silhouette width for k = 7 is not much higher than the for k = 3, and k = 3 is also a reasonable choice. The reason that k = 7 here is because the method is making a differentiation between very small differences between the clusters.

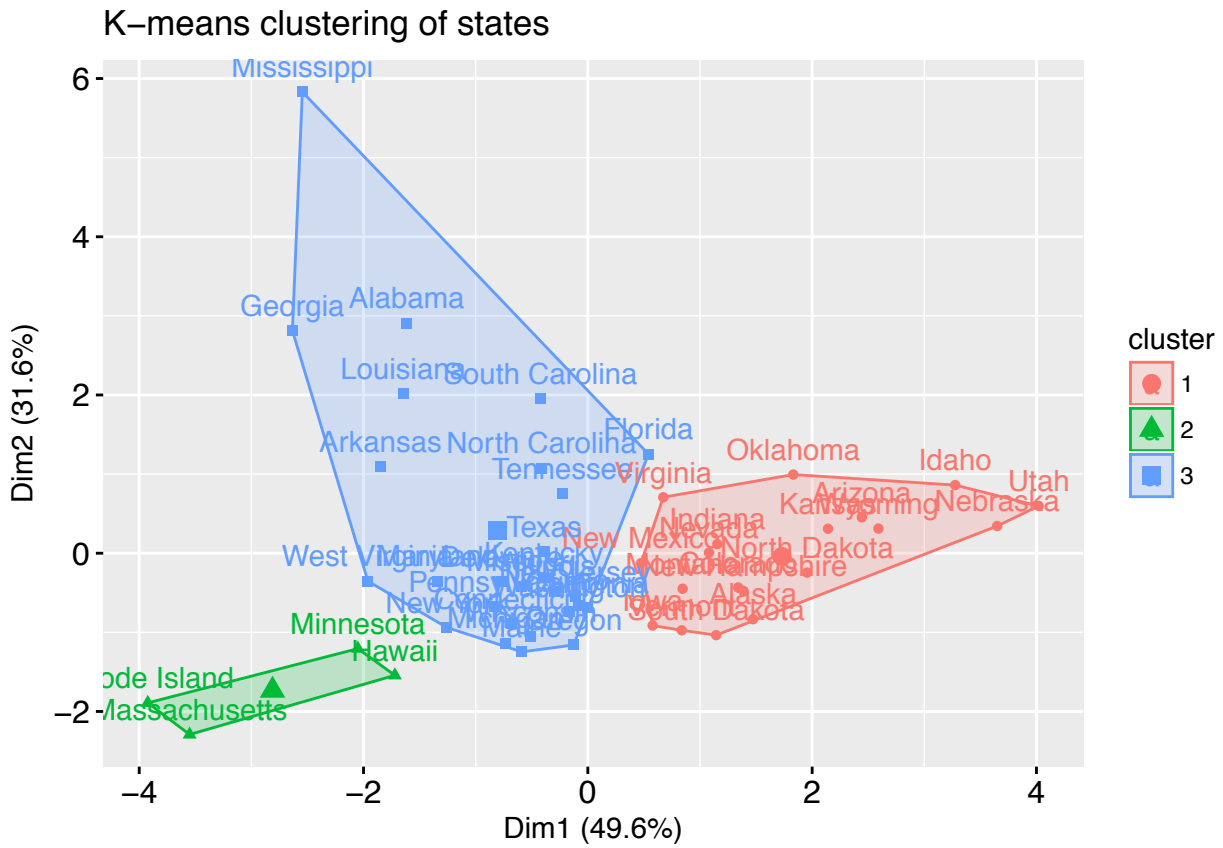
Using the `NBClust` function, we are able to optain a histogram which shows the best clustering scheme from the different results obtained by varying all combinations of number of clusters, distance measures, and clustering methods. The best number of clusters is clearly 3 in this case. Considering the context of the data set, this makes sense, too, as there will be one cluster of states that prefers to vote for the democratics, one cluster that prefers to vote for the republicans, and one cluster that consists of swing states.

(2) Clustering

```
## k-means
set.seed(123)
df.km = kmeans(scale(df), 3)
print(df.km)
```

```
## K-means clustering with 3 clusters of sizes 19, 4, 27
##
## Cluster means:
##        X1980      X1976       X1972      X1968      X1964      X1960
## 1 -1.0047281 -0.8673599 -0.23319895 -0.8327446 -0.1103716 -0.7030778
## 2  1.1344616  0.8728411  1.48086165  1.9973191  1.4878242  0.9303168
## 3  0.5389625  0.4810546 -0.05528395  0.2901063 -0.1427495  0.3569337
##
## Clustering vector:
##        Alabama         Alaska        Arizona       Arkansas     California
##              3              1              1              3              3
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              1              3              3              3              3
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              2              1              3              1              1
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              1              3              3              3              3
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              3              2              3              3
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              1              1              1              1              3
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              1              3              3              1              3
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              1              3              3              2              3
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              1              3              3              1              1
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              1              3              3              3              1
##
## Within cluster sum of squares by cluster:
## [1]  44.315817   9.656483 117.044229
##  (between_SS / total_SS =  41.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
fviz_cluster(df.km, data = scale(df),
  main="K-means clustering of states")
```



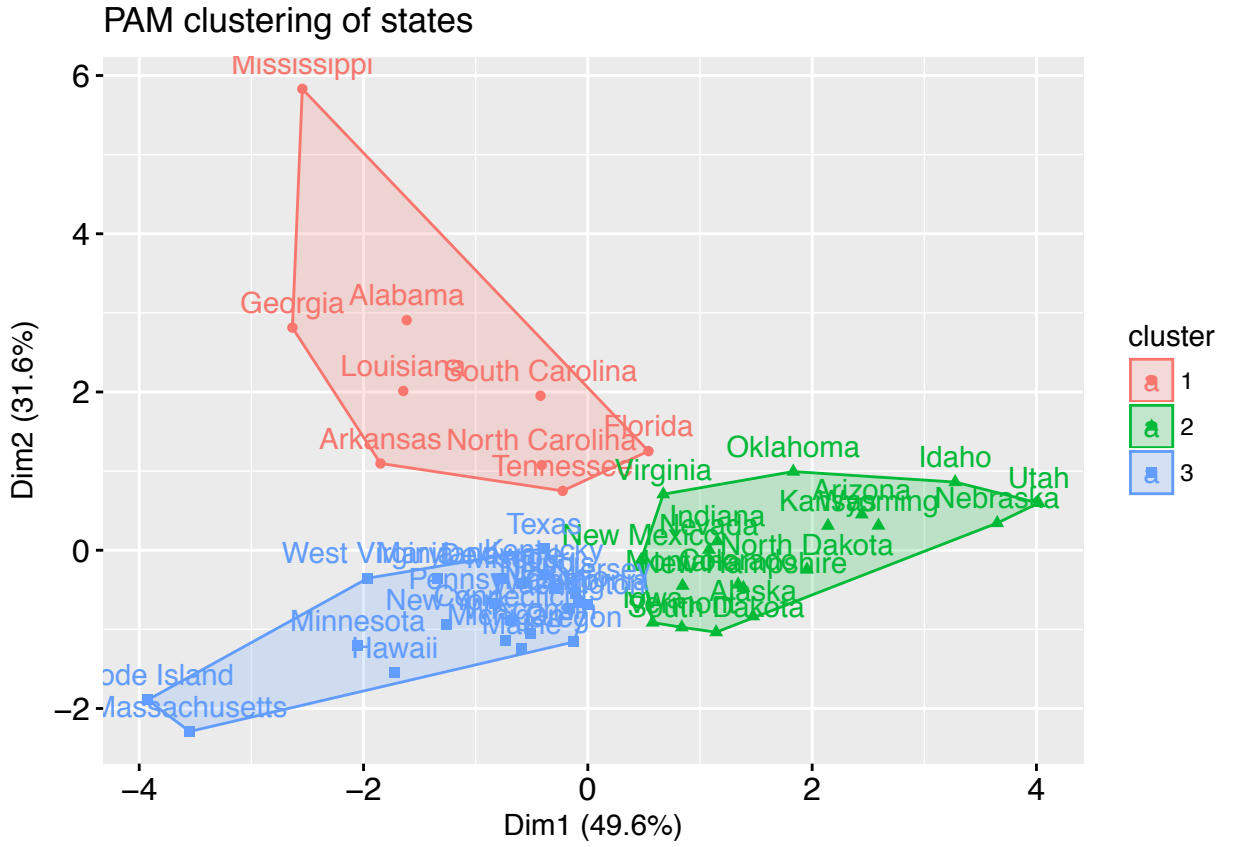
The kmeans method classifies Rhode Island, Massachusetts, Minnesota, and Hawaii as one cluster, states such as Kansas, North Dakota, Nebraska, etc as one cluster, and states such as Connecticut, New York, New Hampshire, etc. as one cluster. We can see that Connecticut, New York, New Hampshire are tightly together, and even though Missisippi, Alabama, Georgia are clustered into the same cluster, they are far away from Connecticut, New York, New Hampshire.

```
## Partitioning around medoids (PAM)** (*Hint:* use the `pam` function)
set.seed(123)
df.pam = pam(scale(df), k=3)
print(df.pam)
```

```
## Medoids:
##                 ID      X1980      X1976       X1972      X1968      X1964
## South Carolina 40  0.7714827  1.0530661 -1.12028683 -0.5624536 -1.4277743
## Colorado        6 -0.7907563 -0.9194404 -0.01456431 -0.3905489  0.2058308
## Pennsylvania   38  0.4158123  0.2299791  0.58355645  0.5463662  0.5080305
##                      X1960
## South Carolina  0.2077228
## Colorado       -0.6680120
## Pennsylvania    0.1963171
## Clustering vector:
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              2              2              1              3
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              2              3              3              1              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              3              2              3              2              2
##         Kansas       Kentucky      Louisiana          Maine       Maryland
```

```
##              2              3              1              3              3
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              3              3              3              1              3
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              2              2              2              2              3
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              2              3              1              2              3
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              3              3              3              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              2              1              3              2              2
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              3              3              3              2
## Objective function:
##    build     swap
## 1.532140 1.485936
##
## Available components:
##  [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"
##  [6] "clusinfo"   "silinfo"    "diss"       "call"       "data"
```

```
fviz_cluster(df.pam, # don't need to specify data for pam
  main="PAM clustering of states")
```

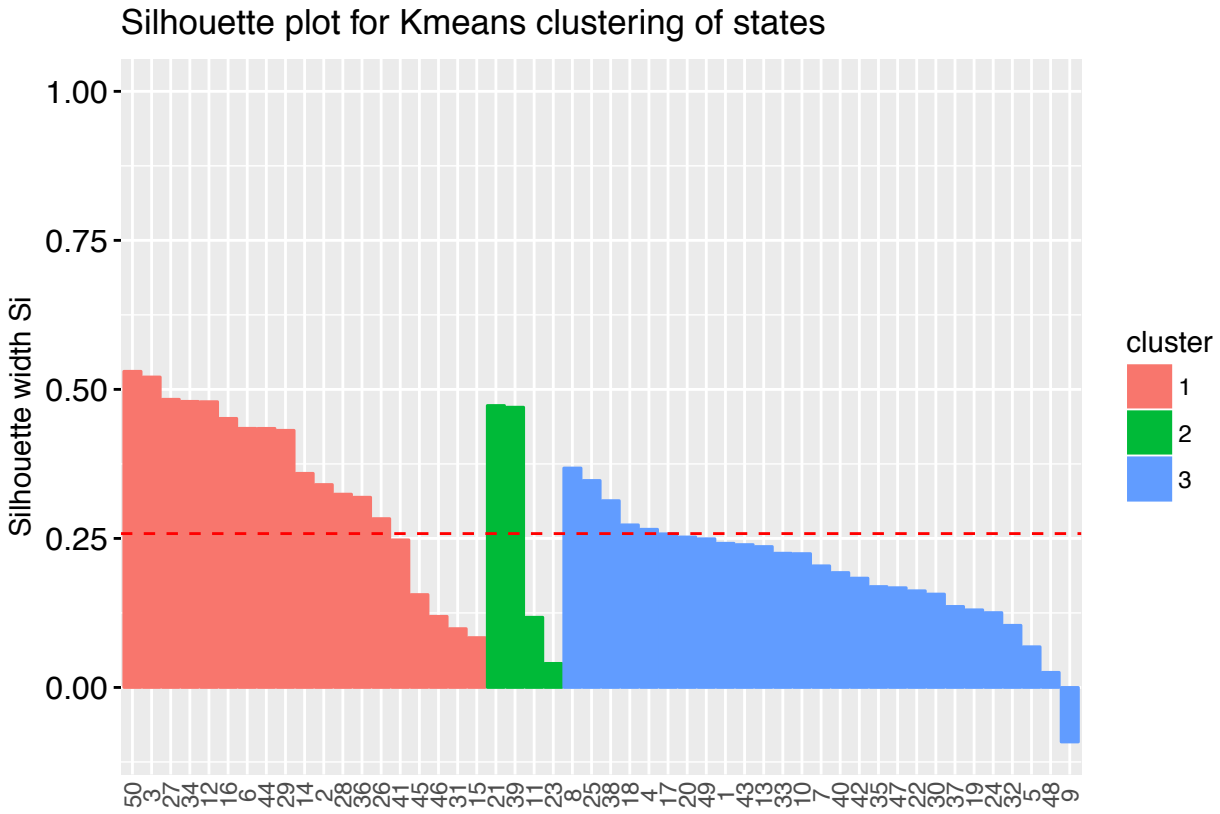


The pam method classifies Rhode Island, Massachusetts, New York, etc. as one cluster, states such as Kansas, Idaho, Nebraska, etc as one cluster, and states such as Alabama, Arkansas, South Carolina etc. as one cluster.

(3) Silhouette diagnostic plots

```
## K-means
fviz_silhouette(silhouette(df.km$cluster, dist(scale(df))),
  main="Silhouette plot for Kmeans clustering of states")+
  theme(axis.text.x = element_text(angle = 90))
```

```
##   cluster size ave.sil.width
## 1       1   19          0.35
## 2       2    4          0.28
## 3       3   27          0.19
```




```
# Compute silhouette
sil.km = silhouette(df.km$cluster, dist(scale(df)))[, 1:3]

# Objects with negative silhouette
neg_sil_index.km = which(sil.km[, 'sil_width'] < 0)
sil.km <- data.frame(states = rownames(df), sil.km)
print(sil.km[neg_sil_index.km, , drop = FALSE])
```

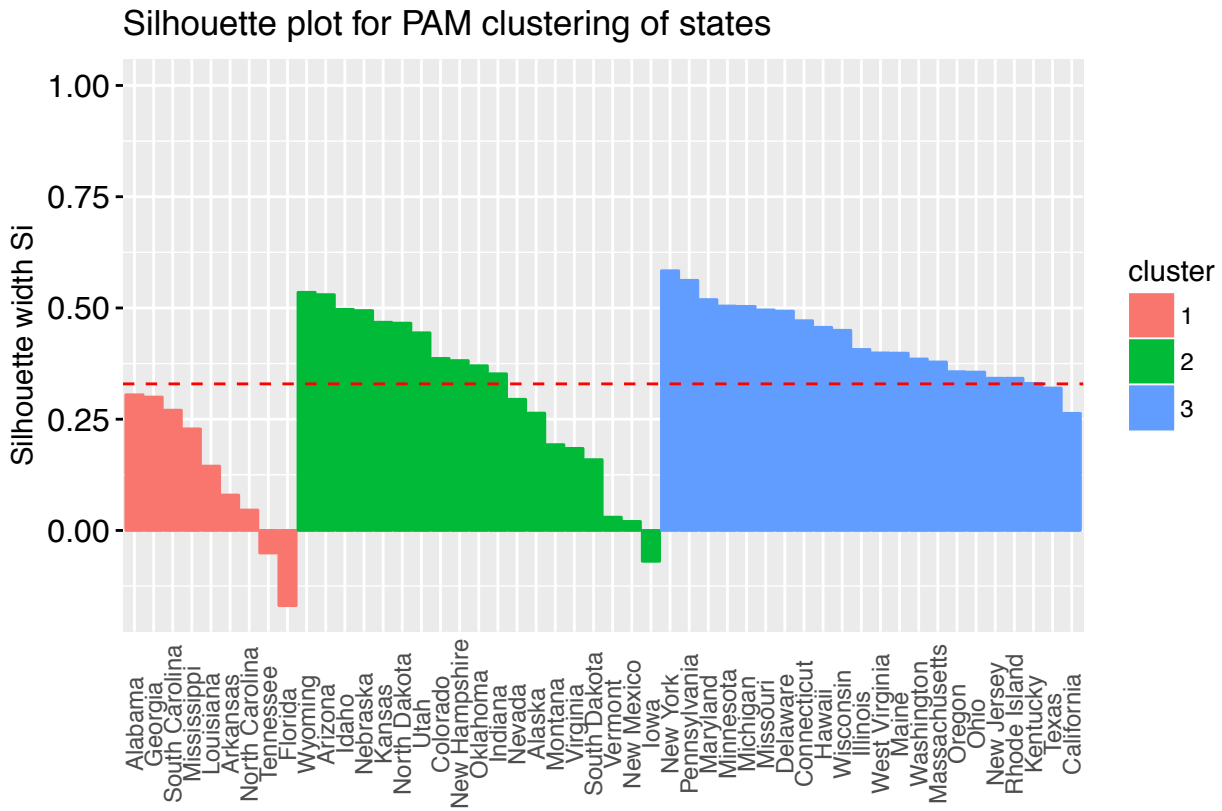
```
##    states cluster neighbor   sil_width
## 9 Florida       3        1 -0.09141436
```

When we look at the silhouette plot, we see that both clusters 1 and 2 are well clustered, with most of the states having widths above the average (dotted line), cluster three (blue) are not as well clustered, wich many states having silhouette widths below the average line. This is consistent with our earlier observation of the kmeans scatter plot, which has a number of states (Mississippi, Georgia etc.) in the clustr that are at a larger distance away from where most of the other states (New York, Connecticut, etc.) in the cluster are rightly clustered. In cluster 3, Florida has a negative sil_width, indicating this is very bad classification. A

negative sil_width means the dissimilarity between Florida and the other states in the next closest cluster is smaller than the dissimilarity between Florida and the other states in its own cluster. This means Florida is probably wrongly classified.

```
## PAM
fviz_silhouette(silhouette(df.pam),
  main="Silhouette plot for PAM clustering of states")+
    theme(axis.text.x = element_text(angle = 90))
```

```
##   cluster size ave.sil.width
## 1       1    9          0.13
## 2       2   19          0.32
## 3       3   22          0.42
```




```
# Compute silhouette
sil.pam = silhouette(df.pam)[, 1:3]
# Objects with negative silhouette
neg_sil_index.pam = which(sil.pam[, 'sil_width'] < 0)
print(sil.pam[neg_sil_index.pam, , drop = FALSE])
```

```
##           cluster neighbor   sil_width
## Tennessee       1        3 -0.05061202
## Florida         1        2 -0.16887690
## Iowa            2        3 -0.06924655
```

The silhouette plot shows that the clusters 2 and 3 are fairly well clustered, with most of the observations above the average sil_width. This means the two clusters are well separated from other clusters. However, cluster 1 is mostly below the average sil_width line and is not very well clustered. The smaller sil_width indicates that members of this cluster have very high dissimilarities among themselves, though not as high as compared to the next nearest cluster. States like Tennessee, Florida, Iowa are wrongly clustered as they have

negative sil_width, indicating they are more similar to their next nearest cluster than its currently classified cluster. The large negative value of sil_width = -0.169 for Florida indicates that it is very badly classified and is most probably wrong.

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.

Hint: The following code will help you reformat the output of the `agnes` and `diana` functions in order to apply the presented methods to find the optimal number of clusters:

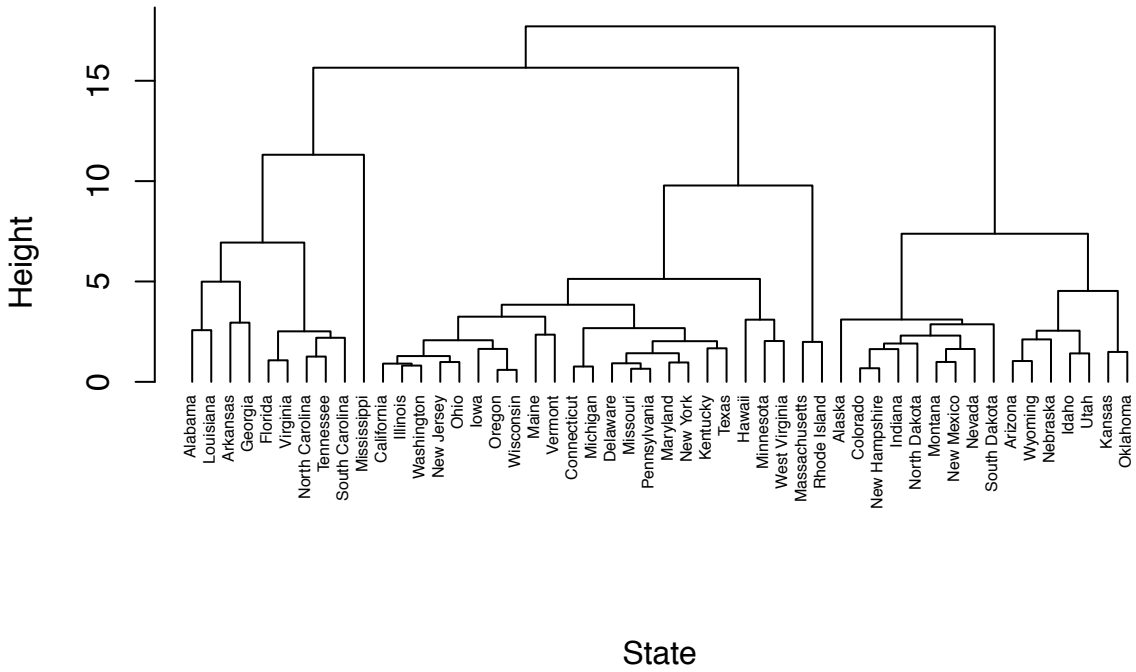
```
agnes.reformat<-function(x, k){
# x: Data matrix or frame, k: Number of clusters
  x.agnes = agnes(x,method="ward",stand=T)
  x.cluster = list(cluster=cutree(x.agnes,k=k))
  return(x.cluster)
}

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)
}
```

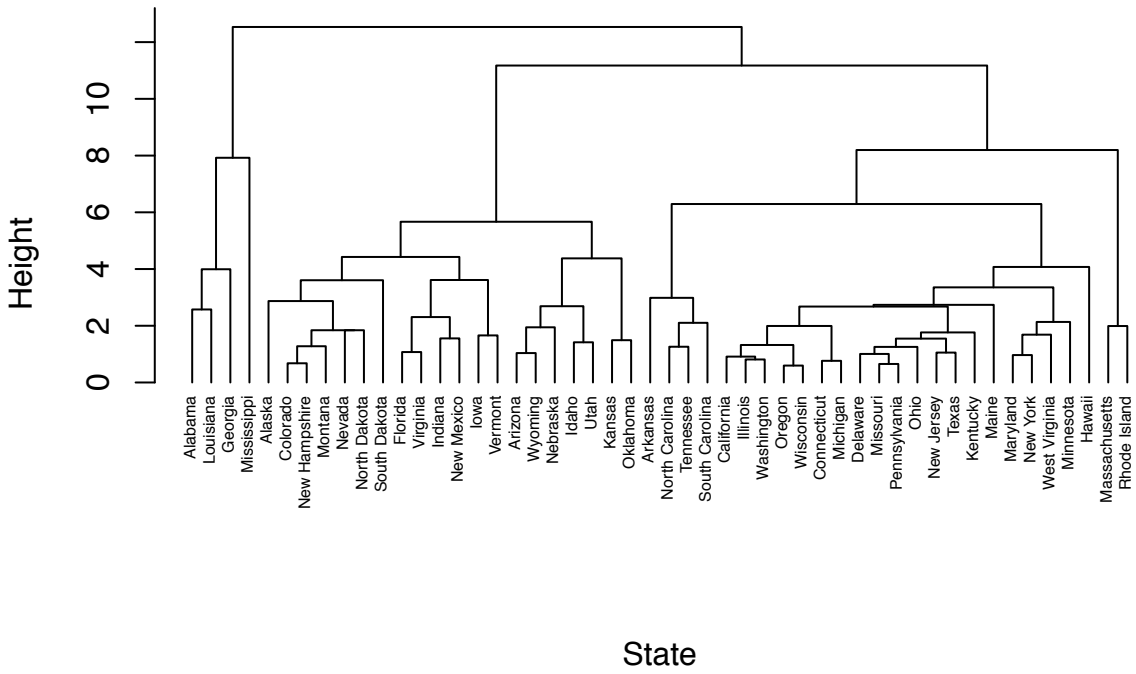
Based on your choice of the optimal number of clusters in each case, visualize the clusters using a principal components plot, and compare them with the clustering results in Part 2b.

Answer

```
## Agnes clustering
df.agnes = agnes(df, method="ward", stand=T)
pltree(df.agnes, cex=0.5, hang= -1,
  main="AGNES fit",
  xlab="State",sub="")
```




```
## Diana clustering
df.diana = diana(df, stand=T)
pltree(df.diana, cex=0.5, hang= -1,
  main="DIANA fit",
  xlab="State",sub="")
```

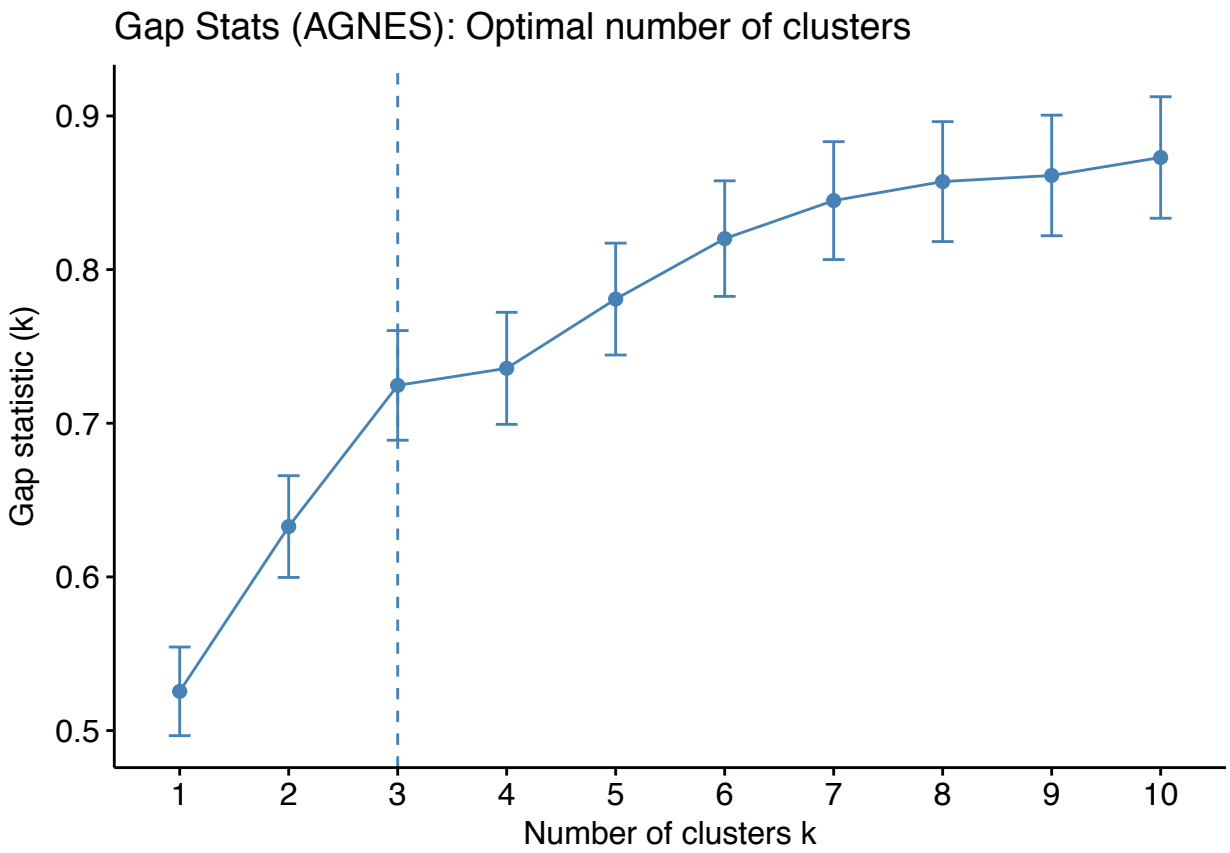


Optimal number of clusters using gap statistics

```
# Agglomerative clustering: AGNES
set.seed(123)
gapstat <- clusGap(scale(df), FUN=agnes.reformat, K.max=10, B=500,
                   spaceH0 = "original") #### here change spaceH0 to original to get >1 cluster
print(gapstat, method="Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = scale(df), FUNcluster = agnes.reformat, K.max = 10,     B = 500, spaceH0 = "original")
## B=500 simulated reference sets, k = 1..10; spaceH0="original"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 3
##           logW   E.logW       gap     SE.sim
##  [1,] 3.620266 4.145775 0.5255090 0.02884894
##  [2,] 3.411249 4.044009 0.6327599 0.03310461
##  [3,] 3.230774 3.955417 0.7246426 0.03569147
##  [4,] 3.140818 3.876569 0.7357509 0.03646912
##  [5,] 3.023726 3.804546 0.7808198 0.03639627
##  [6,] 2.917142 3.737305 0.8201632 0.03759517
##  [7,] 2.828306 3.673209 0.8449035 0.03835873
##  [8,] 2.755878 3.613151 0.8572728 0.03901670
##  [9,] 2.693792 3.555063 0.8612708 0.03923871
## [10,] 2.625889 3.498857 0.8729680 0.03954308
```

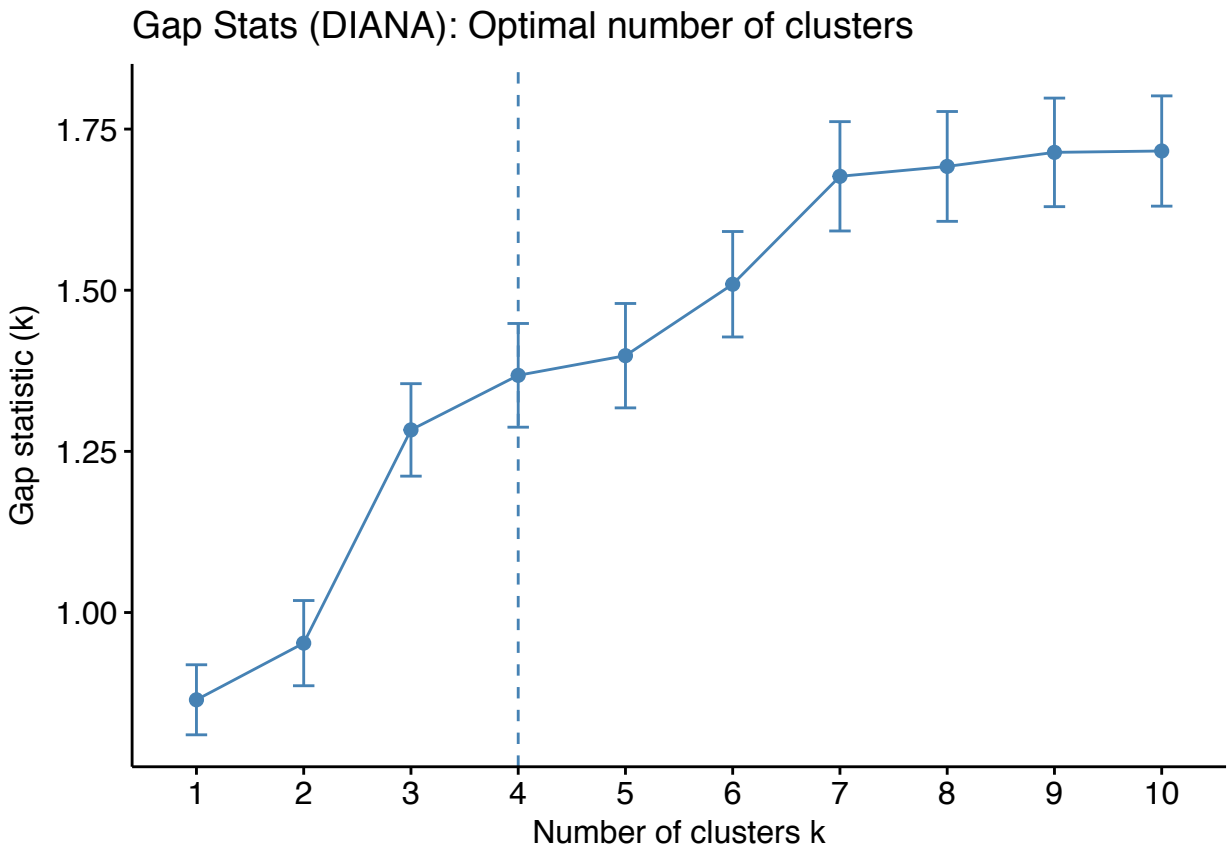
```
fviz_gap_stat(gapstat,
  maxSE=list(method="Tibs2001SEmax",SE.factor=1)) +
  ggtitle("Gap Stats (AGNES): Optimal number of clusters")
```



```r
# Divisive clustering: DIANA
gapstat <- clusGap(scale(df), FUN=diana.reformat, K.max=10, B=500,
                   d.power = 2, spaceH0 = "original")
print(gapstat, method="Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = scale(df), FUNcluster = diana.reformat, K.max = 10,     B = 500, d.power = 2, spaceH0 =
## B=500 simulated reference sets, k = 1..10; spaceH0="original"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 4
##           logW   E.logW       gap     SE.sim
##  [1,] 4.990433 5.855025 0.8645924 0.05434036
##  [2,] 4.715297 5.667805 0.9525080 0.06611329
##  [3,] 4.253012 5.536311 1.2832988 0.07173400
##  [4,] 4.047377 5.415333 1.3679558 0.08045150
##  [5,] 3.911527 5.309981 1.3984543 0.08099041
##  [6,] 3.705164 5.214451 1.5092866 0.08180348
##  [7,] 3.446163 5.122891 1.6767287 0.08482162
##  [8,] 3.343112 5.035165 1.6920527 0.08521260
##  [9,] 3.236808 4.950676 1.7138674 0.08421182
## [10,] 3.155082 4.871054 1.7159715 0.08553529
```

```r
fviz_gap_stat(gapstat,
  maxSE=list(method="Tibs2001SEmax",SE.factor=1)) +
  ggtitle("Gap Stats (DIANA): Optimal number of clusters")
```



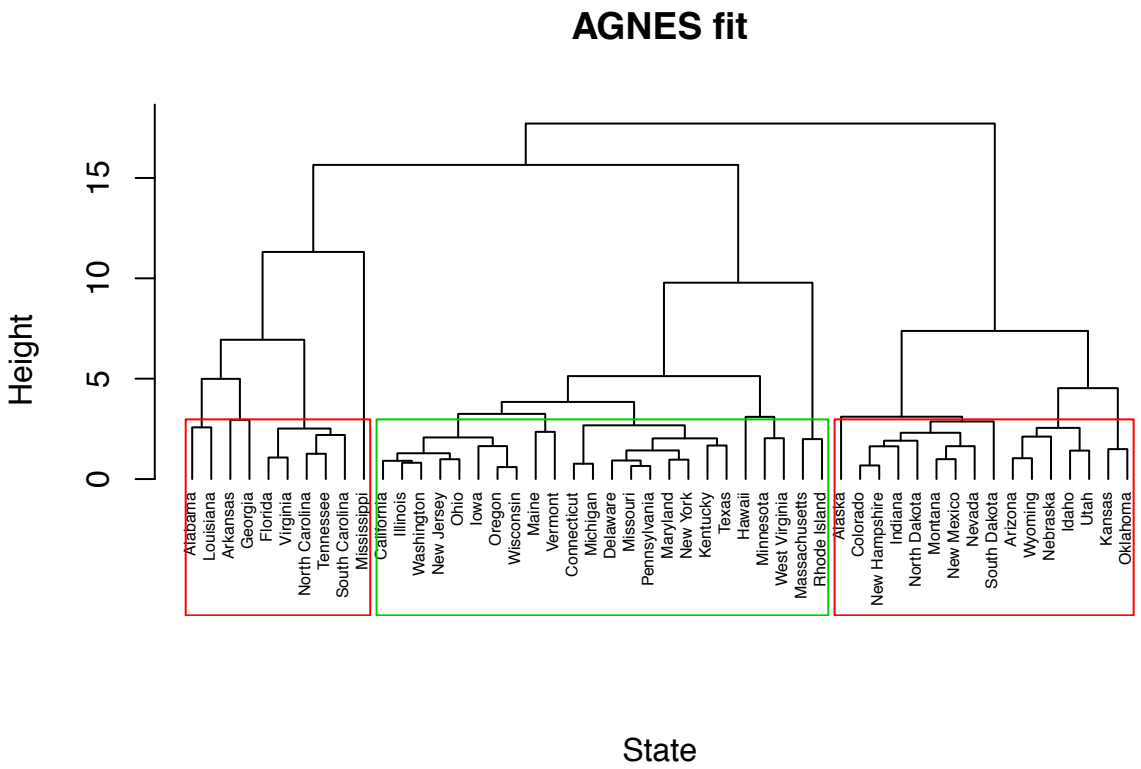
Here, gap statistics predicts 3 clusters as the optimal number of clusters to use for both agnes and diana.

We chose to use the `original` sapceH0 here as gap statistics tend to give us only one cluster if we use the

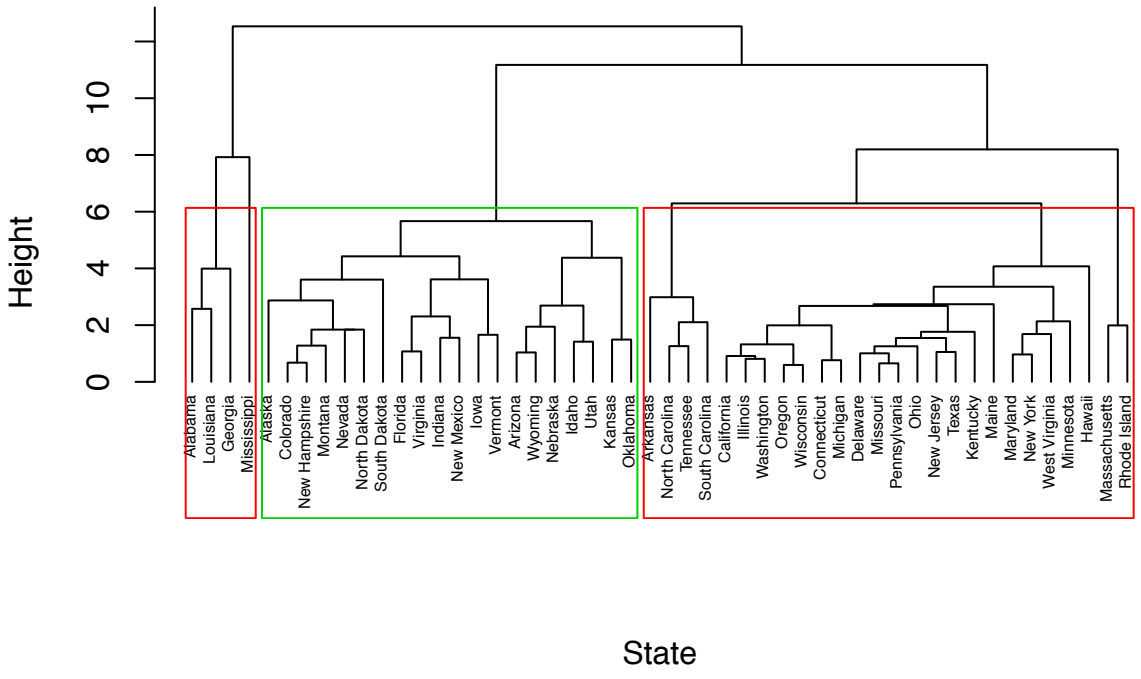
default option, which considers hypercubes of the original data space. By using `original`, we consider all dimensions of the data space and is able to make distinction between smaller differences among clusters, resulting in a larger number of optimal cluster predicted by gap statistics. In the context of the data set here, this makes sense, as the data had very small differenciations among the differently behaved voting clusters, and it therefore necessary for gap statistics to differentiate between small distinctions.

Agenes and Diana clusterings

```
## Agnes clustering
df.agnes = agnes(df, method="ward", stand=T)
pltree(df.agnes, cex=0.5, hang= -1,
  main="AGNES fit",
  xlab="State",sub="")
rect.hclust(df.agnes, k=3, border=2:3)
```




```
## Diana clustering
df.diana = diana(df, stand=T)
pltree(df.diana, cex=0.5, hang= -1,
  main="DIANA fit",
  xlab="State",sub="")
rect.hclust(df.diana, k=3, border=2:3)
```



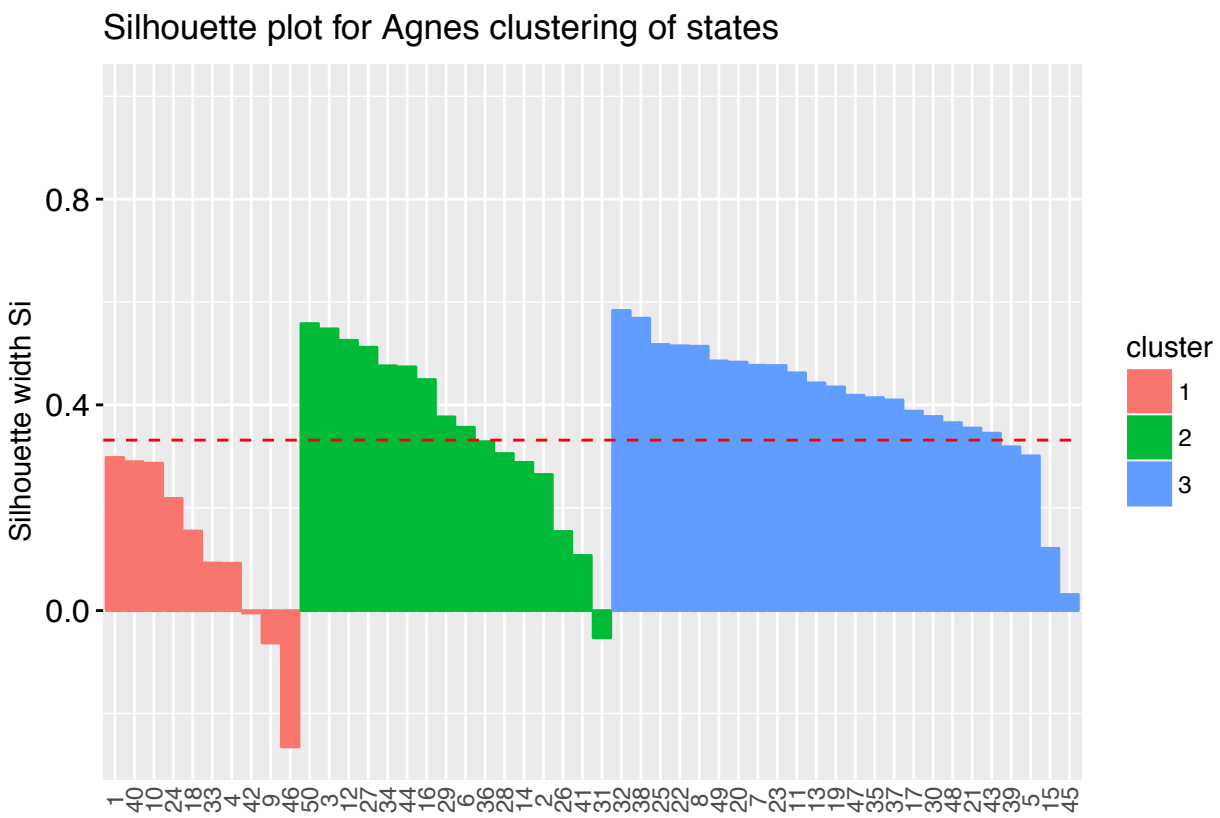
In both the Agglomerative clustering (agnes) and Divisive clustering (diana) methods, Republican states such as Wyoming, Idaho, Alaska, Utah are clustered into the same cluster, but Alabama is not in the same cluster. This can be a result of bad classification for the state Alabama, which we will confirm by looking at the silhouette plot.

For Agglomerative clustering (agnes), Democrats states such as Maryland, New York, Vermont, California, Massachusetts are all classified into the same cluster. For Divisive clustering (diana), Vermont is not classified into the same cluster along with the rest of the Democratic states, we should take a look at the silhouette plot to see if this is a bad classification.

Silhouette diagnostic plots

```
## Agnes
df.agnes.sil <- agnes.reformat(scale(df), k = 3)
fviz_silhouette(silhouette(df.agnes.sil$cluster, dist(scale(df))),
  main="Silhouette plot for Agnes clustering of states")+
  theme(axis.text.x = element_text(angle = 90))
```

```
##   cluster size ave.sil.width
## 1       1   10          0.11
## 2       2   16          0.35
## 3       3   24          0.41
```



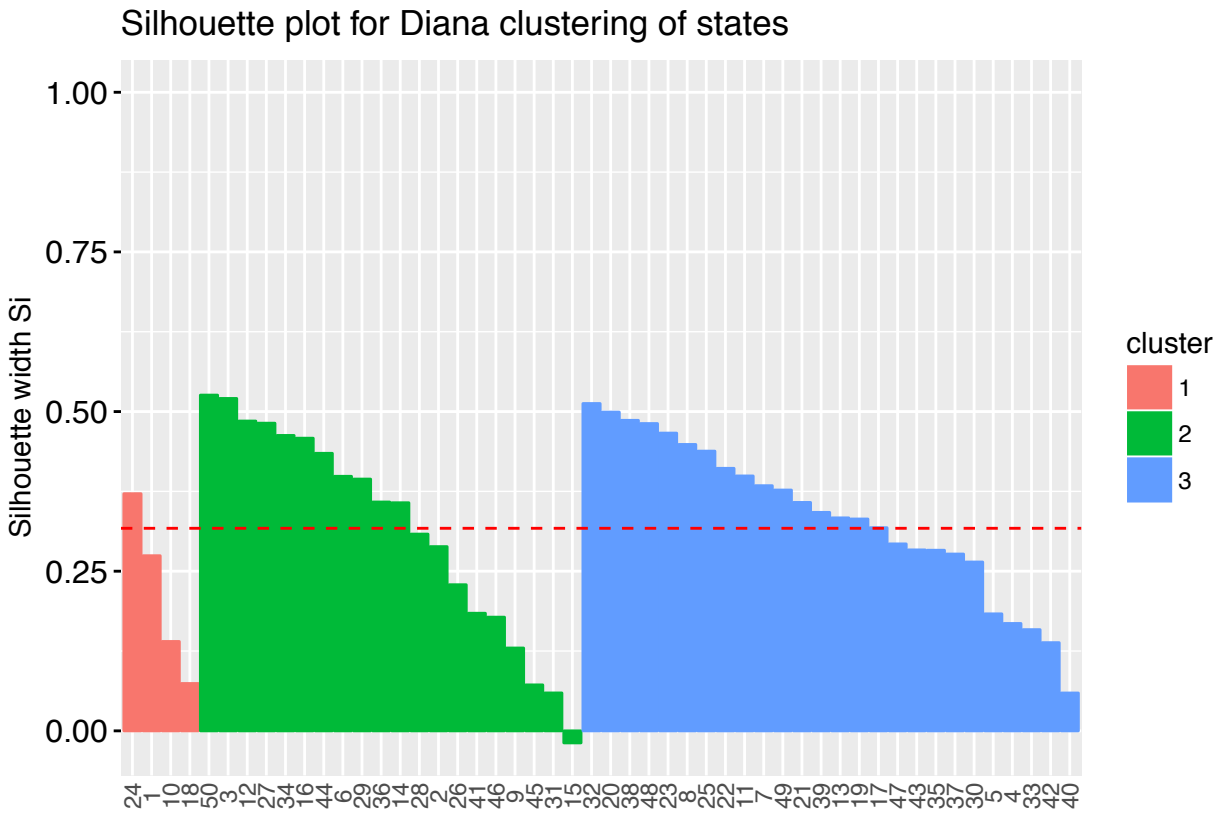

```
# Compute silhouette
sil.km = silhouette(df.agnes.sil$cluster, dist(scale(df)))[, 1:3]
sil.km <- data.frame(states = rownames(df), sil.km)

# Objects with negative silhouette
neg_sil_index.km = which(sil.km[, 'sil_width'] < 0)
print(sil.km[neg_sil_index.km, , drop = FALSE])
```

```
##        states cluster neighbor    sil_width
## 9     Florida       1        2 -0.062876396
## 31 New Mexico       2        3 -0.052873455
## 42  Tennessee       1        3 -0.004571546
## 46   Virginia       1        2 -0.264927616
```

```
## Diana
df.diana.sil <- diana.reformat(scale(df), k = 3)
fviz_silhouette(silhouette(df.diana.sil$cluster, dist(scale(df))),
  main="Silhouette plot for Diana clustering of states")+
  theme(axis.text.x = element_text(angle = 90))
```

```
##   cluster size ave.sil.width
## 1       1    4          0.21
## 2       2   20          0.32
## 3       3   26          0.33
```




```
# Compute silhouette
sil.km = silhouette(df.diana.sil$cluster, dist(scale(df)))[, 1:3]
sil.km <- data.frame(states = rownames(df), sil.km)

# Objects with negative silhouette
neg_sil_index.km = which(sil.km[, 'sil_width'] < 0)
print(sil.km[neg_sil_index.km, , drop = FALSE])
```

```
##    states cluster neighbor   sil_width
## 15   Iowa       2        3 -0.01883196
```

Indeed, in both the solhouette plots, Alabama, which is state number 1, has a lower than average sil_width, this means that it is not well clustered - it has a large dissimilarity with the rest of the states in its cluster. This explains why is it not classified into the same cluster as the other Republican states Wyoming, Idaho, Alaska, and Utah.

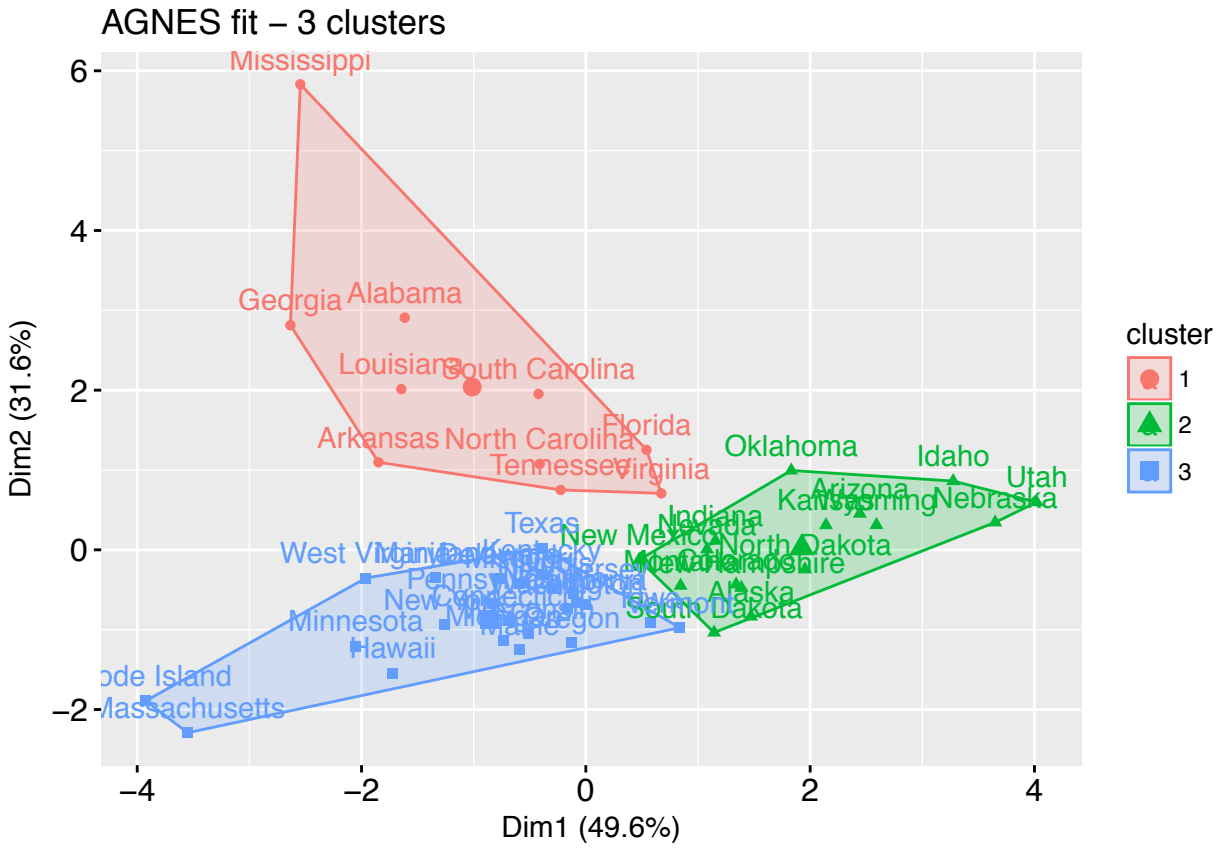
In the Agnes silhouette plot, we see that four states: Florida, New Mexico, Tennessee, and Virginia have negative sil_widths, expecially Virginia, with an extremely negative -0.26 sil_width. This indicates that they are wrongly clustered.

In the Diana silhouette plot, we see that state 45, which is Vermont, indeed has a very small sil_width that is close to 0, indicating that it is not a good classification. This explains why Vermont was not classified along with the rest of the Democratic states into one cluster. In particular, Iowa has a negative sil_width, indicating that it is probably wrongly clustered by the same reasonings as the previous parts. Also, Cluster 1 in general is not well clustered, since most of the data points have below average sil_widths.

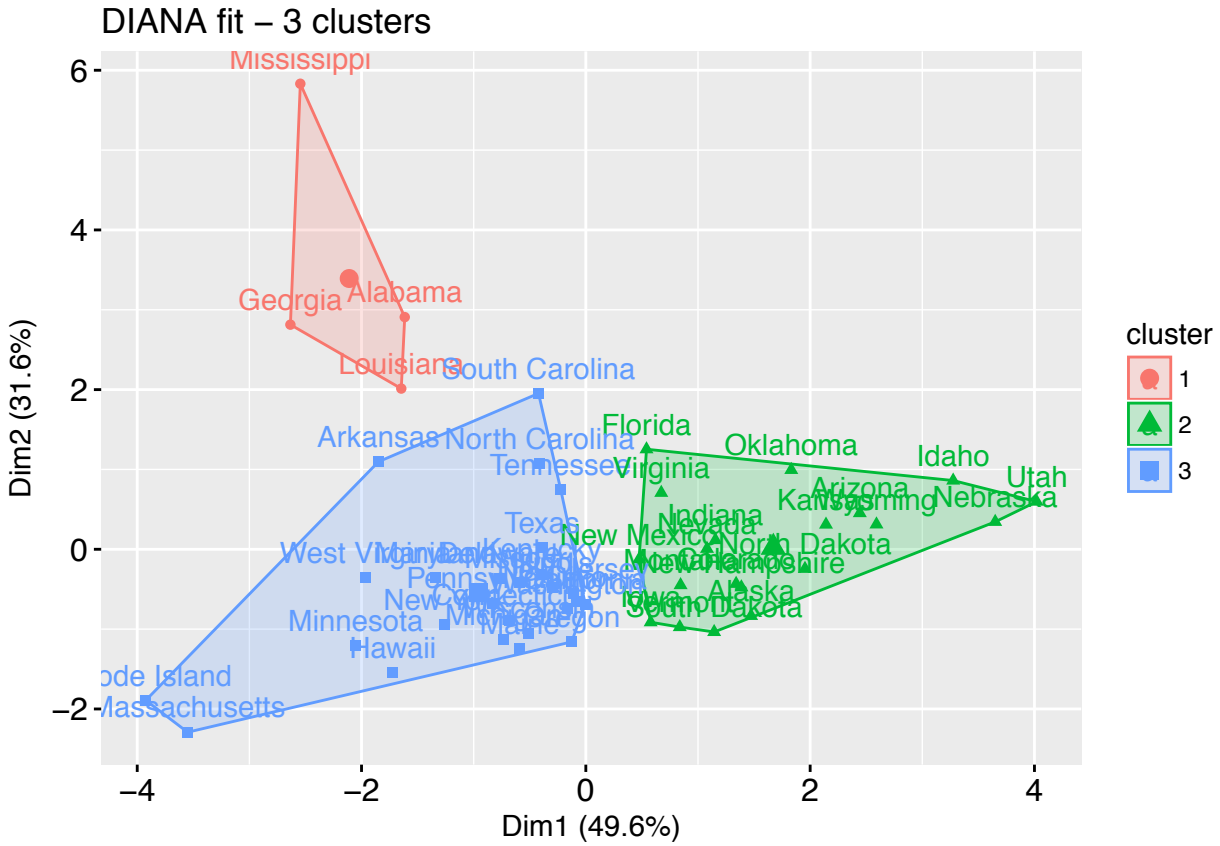
Principal component visualization

```
## Based on your choice of the optimal number of clusters in each case, visualize the clusters using a p
set.seed(123)

# Agnes
grp.agnes = cutree(df.agnes, k=3)
fviz_cluster(list(data = scale(df), cluster = grp.agnes),
  main="AGNES fit - 3 clusters")
```




```
# Diana
grp.diana = cutree(df.diana, k=3)
fviz_cluster(list(data = scale(df), cluster = grp.diana),
  main="DIANA fit - 3 clusters")
```



Compared to part 2b (kmeans and partitioning around medoids), the clusters formed by the Agglomerative method (Agnes) is similar to that from the Partitioning around medoids (PAM) method, the silhouette plots from these two methods are also similar, with two clusters having most datapoints above the average sil_width, indicating that they are well classified (clusters 2 and 3), while the first cluster (cluster 1) are not as well classified.

The Divisive clustering (Diana) method does not do a good job at classifying cluster 1, we saw that from the silhouette plot previously, and also here, we see that it only has four states, and includes the outlier Mississippi which is far away from every other state. This is a clear indication that this is not a good cluster.

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.

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

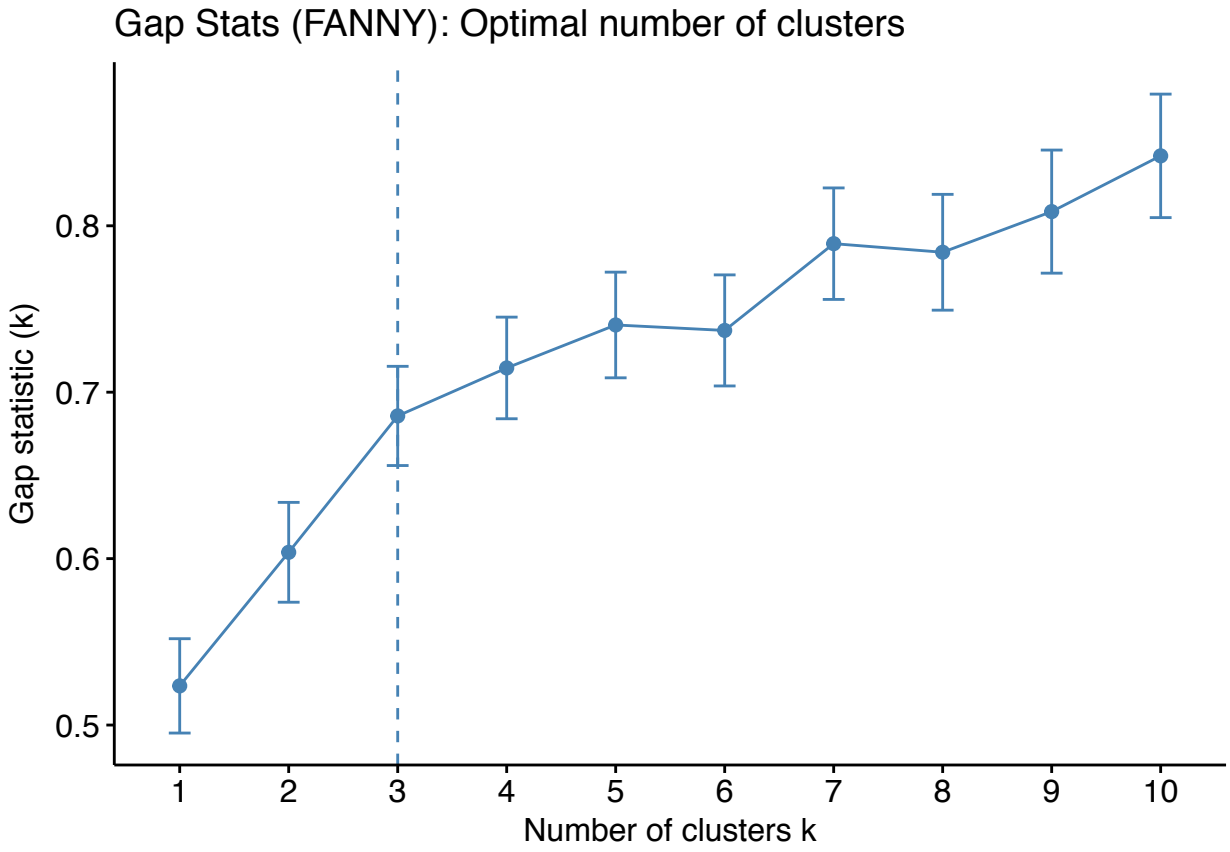
Answer

Fuzzy clustering: FANNY

```
# Optimal number of clusters
set.seed(123)
gapstat <- clusGap(scale(df), FUN=fanny, K.max=10,
                   maxit = 5000, memb.exp = 1.2, spaceH0 = "original")
print(gapstat, method="Tibs2001SEmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = scale(df), FUNcluster = fanny, K.max = 10, spaceH0 = "original",     maxit = 5000, memb.
## B=100 simulated reference sets, k = 1..10; spaceH0="original"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 3
##           logW   E.logW       gap     SE.sim
##  [1,] 3.620266 4.143804 0.5235376 0.02834446
##  [2,] 3.403831 4.007632 0.6038006 0.02999326
##  [3,] 3.225126 3.910880 0.6857536 0.02980370
##  [4,] 3.114446 3.829025 0.7145788 0.03053101
##  [5,] 3.017488 3.757887 0.7403984 0.03174797
##  [6,] 2.953739 3.690862 0.7371227 0.03336858
##  [7,] 2.842146 3.631399 0.7892530 0.03349999
##  [8,] 2.789846 3.573939 0.7840936 0.03479511
##  [9,] 2.710797 3.519309 0.8085120 0.03698070
## [10,] 2.625176 3.467195 0.8420193 0.03710104
```

```
fviz_gap_stat(gapstat,
  maxSE=list(method="Tibs2001SEmax",SE.factor=1)) +
  ggtitle("Gap Stats (FANNY): Optimal number of clusters")
```



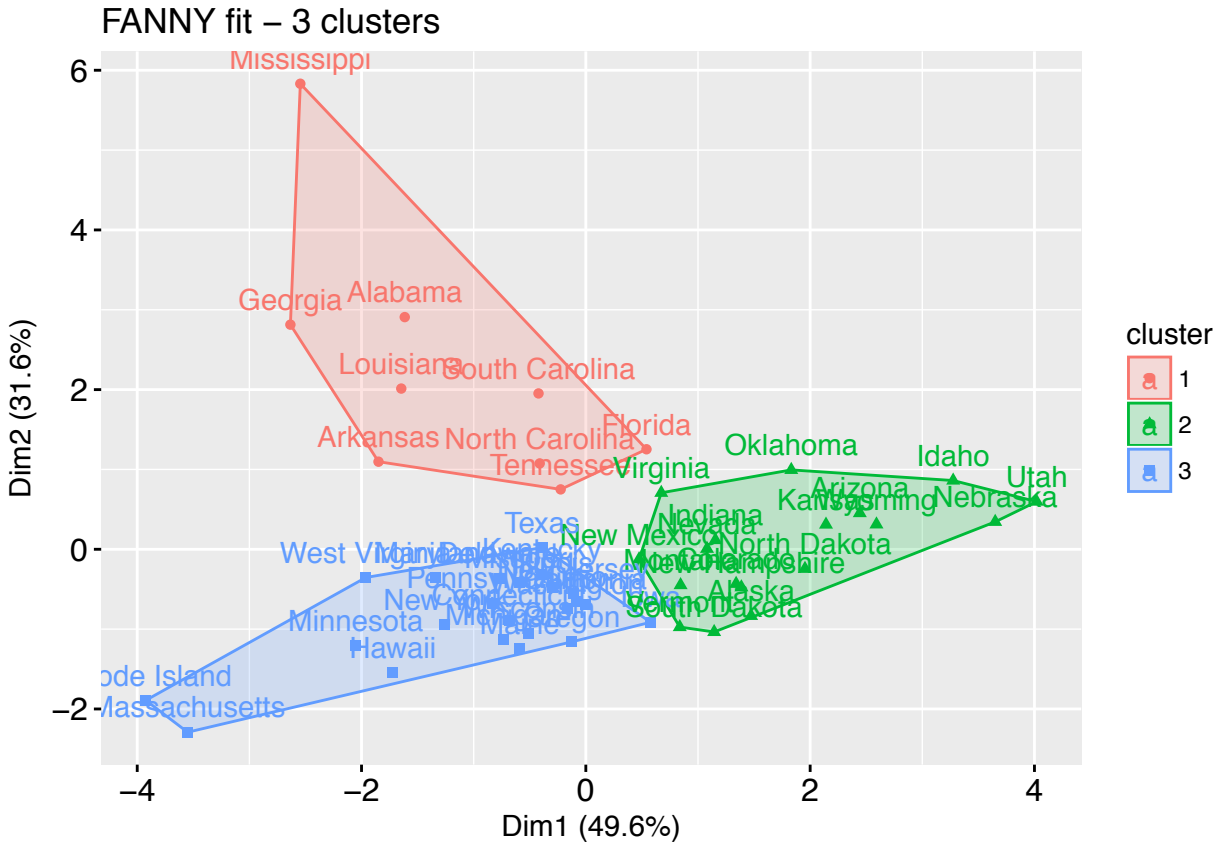
The optimal number of clusters picked by gap statistics in 3. In the following, we use k = 3 for fuzzy clustering.

```
# FANNY clustering
df.fanny = fanny(scale(df), k=3, memb.exp = 1.2)
print((round(df.fanny$membership,3)))
```

```
##                [,1]  [,2]  [,3]
## Alabama       0.993 0.002 0.005
## Alaska        0.004 0.945 0.051
## Arizona       0.000 0.999 0.001
## Arkansas      0.970 0.003 0.027
## California    0.002 0.024 0.973
## Colorado      0.000 0.994 0.005
## Connecticut   0.001 0.003 0.996
## Delaware      0.002 0.001 0.997
## Florida       0.568 0.364 0.067
## Georgia       0.990 0.003 0.008
## Hawaii        0.021 0.012 0.967
## Idaho         0.003 0.994 0.003
## Illinois      0.001 0.002 0.997
## Indiana       0.002 0.986 0.012
## Iowa          0.008 0.249 0.743
## Kansas        0.002 0.995 0.003
## Kentucky      0.023 0.015 0.961
## Louisiana     0.979 0.004 0.017
## Maine         0.005 0.011 0.984
## Maryland      0.007 0.001 0.992
```

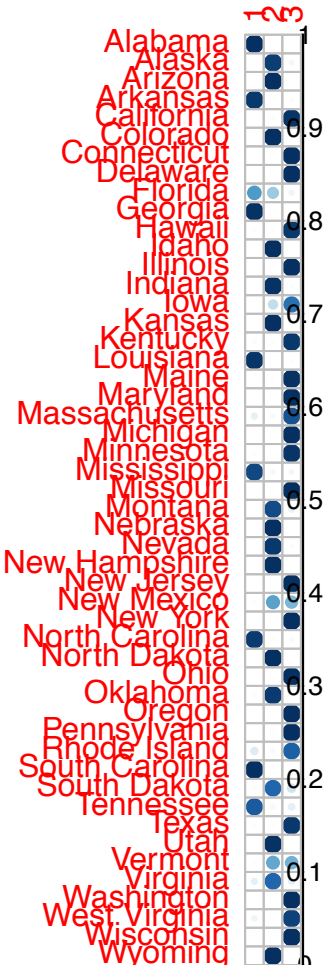
```
## Massachusetts  0.102 0.035 0.863
## Michigan       0.001 0.001 0.998
## Minnesota      0.017 0.004 0.979
## Mississippi    0.906 0.043 0.051
## Missouri       0.001 0.000 0.999
## Montana        0.003 0.898 0.099
## Nebraska       0.003 0.994 0.004
## Nevada         0.005 0.969 0.027
## New Hampshire  0.000 0.994 0.006
## New Jersey     0.001 0.007 0.992
## New Mexico     0.014 0.531 0.455
## New York       0.001 0.000 0.999
## North Carolina 0.950 0.013 0.037
## North Dakota   0.001 0.997 0.002
## Ohio           0.001 0.004 0.996
## Oklahoma       0.028 0.955 0.017
## Oregon         0.001 0.007 0.992
## Pennsylvania   0.000 0.000 1.000
## Rhode Island   0.145 0.036 0.819
## South Carolina 0.993 0.003 0.004
## South Dakota   0.012 0.790 0.198
## Tennessee      0.835 0.044 0.120
## Texas          0.021 0.019 0.960
## Utah           0.007 0.983 0.010
## Vermont        0.012 0.510 0.478
## Virginia       0.101 0.796 0.102
## Washington     0.000 0.003 0.996
## West Virginia  0.074 0.005 0.921
## Wisconsin      0.001 0.002 0.997
## Wyoming        0.000 0.999 0.001
```

```
fviz_cluster(df.fanny,
  main="FANNY fit - 3 clusters")
```



The Fuzzy clustering method has classified most democratic states together, New York, Pennsylvania, Maryland, etc. are in the same cluster, while Republican states such as Idaho, Kansas, Utah are classified into the same cluster. The clusters looks similar to the clusters selected by the Agnes method.

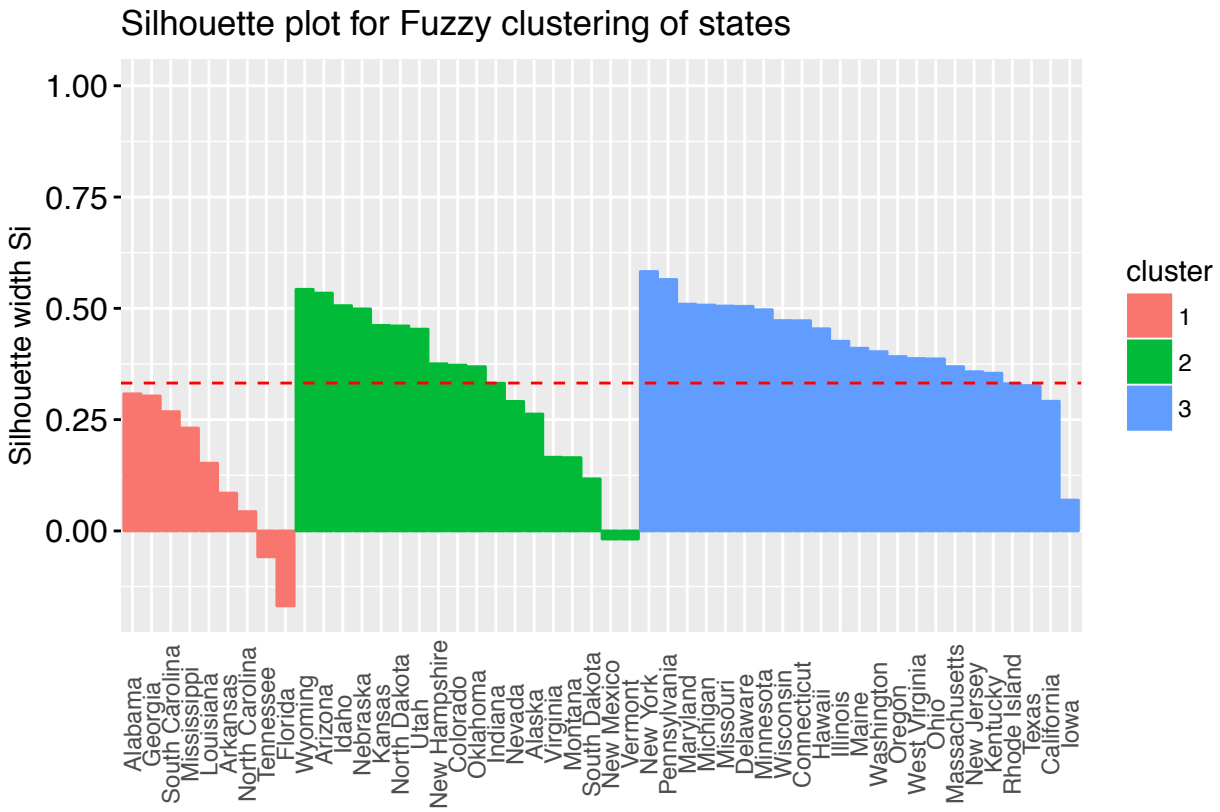
```
library(corrplot)
corrplot(df.fanny$membership, is.corr=F)
```



The correlation plot maps the probabilities of each state belong to a certain cluster. The darker the circle, the more likle that it belongs to a certain cluster, the lighter the circle, the less likely it belongs to a particular cluster. From the correlation plot, we see that most states have a dominating dark circle indicating that there is a strongly probability that the state belongs in this cluster. We also see that there are several states that have similar cluster membership probabilities. There are the states with lighter blue dots in more than one clusters. For example, it looks like New Mexico can be classified into either cluster 2 or 3 with very similar probabilities (0.531 and 0.455, respectively). Florida, on the other hand, belongs to cluster 1 with probability of 0.568, but also belongs to clusters 2 and 3 with probabilities 0.364 and 0.067 repsectively. Florida will get classified into cluster 1, but might have a negative silhouette width, since it has low dissimilarities with datapoints in other clusters.

```
## silhouette diagnostic plot
## Fanny
fviz_silhouette(silhouette(df.fanny),
  main="Silhouette plot for Fuzzy clustering of states")+
  theme(axis.text.x = element_text(angle = 90))
```

```
##   cluster size ave.sil.width
## 1       1    9          0.13
## 2       2   18          0.33
## 3       3   23          0.42
```




```
# Compute silhouette
sil.km = silhouette(df.fanny)[, 1:3]
# Objects with negative silhouette
neg_sil_index.km = which(sil.km[, 'sil_width'] < 0)
print(sil.km[neg_sil_index.km, , drop = FALSE])
```

```
##            cluster neighbor   sil_width
## Tennessee        1        3 -0.05804651
## Florida          1        2 -0.16835297
## New Mexico       2        3 -0.01811239
## Vermont          2        3 -0.01828401
```

The silhouette plot shows that the second and third clusters are generally well clusered, as most data points are above the average sil_width. However, most data points in the first cluster are below the average sil_width, indicating that this is not a very good classification. Tennessee and Florida have negative sil_widths, indicating that they are wrongly classified, expecially Florida which has a large negative sil_width. Vermont and New Mexico also have negative sil_width values, this means that they are likely classified wrongly too.

Gaussian mixture model (Mclust)

```
# Gaussian mixture model (Mclust)
library(mclust)
```

```
## Package 'mclust' version 5.2.3
```

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

```
df.mc = Mclust(scale(df))
print(summary(df.mc))
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust EVE (ellipsoidal, equal volume and orientation) model with 2 components:
##
##  log.likelihood  n df       BIC       ICL
##       -242.5017 50 39 -637.5722 -637.5763
##
## Clustering table:
##  1  2
##  6 44
```

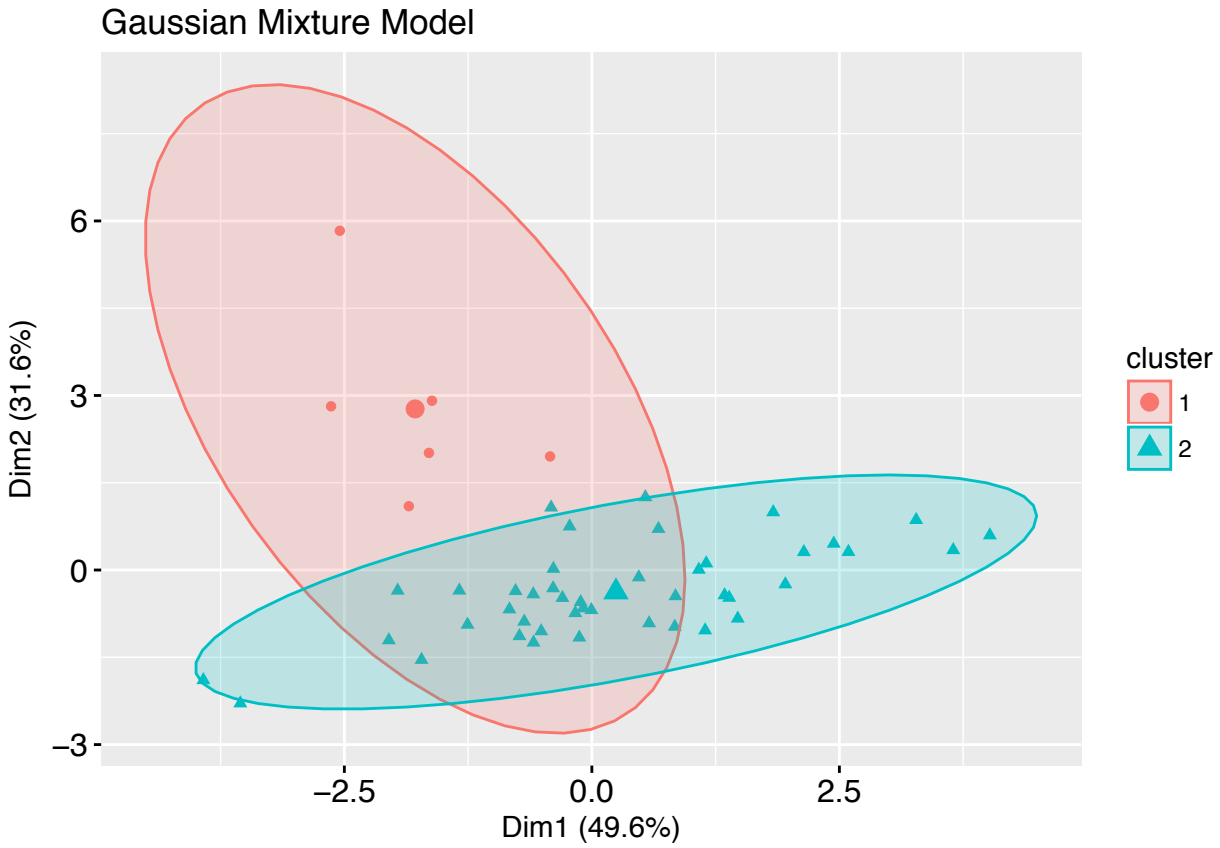
```
# optimal number of clusters
print(df.mc$G)
```

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

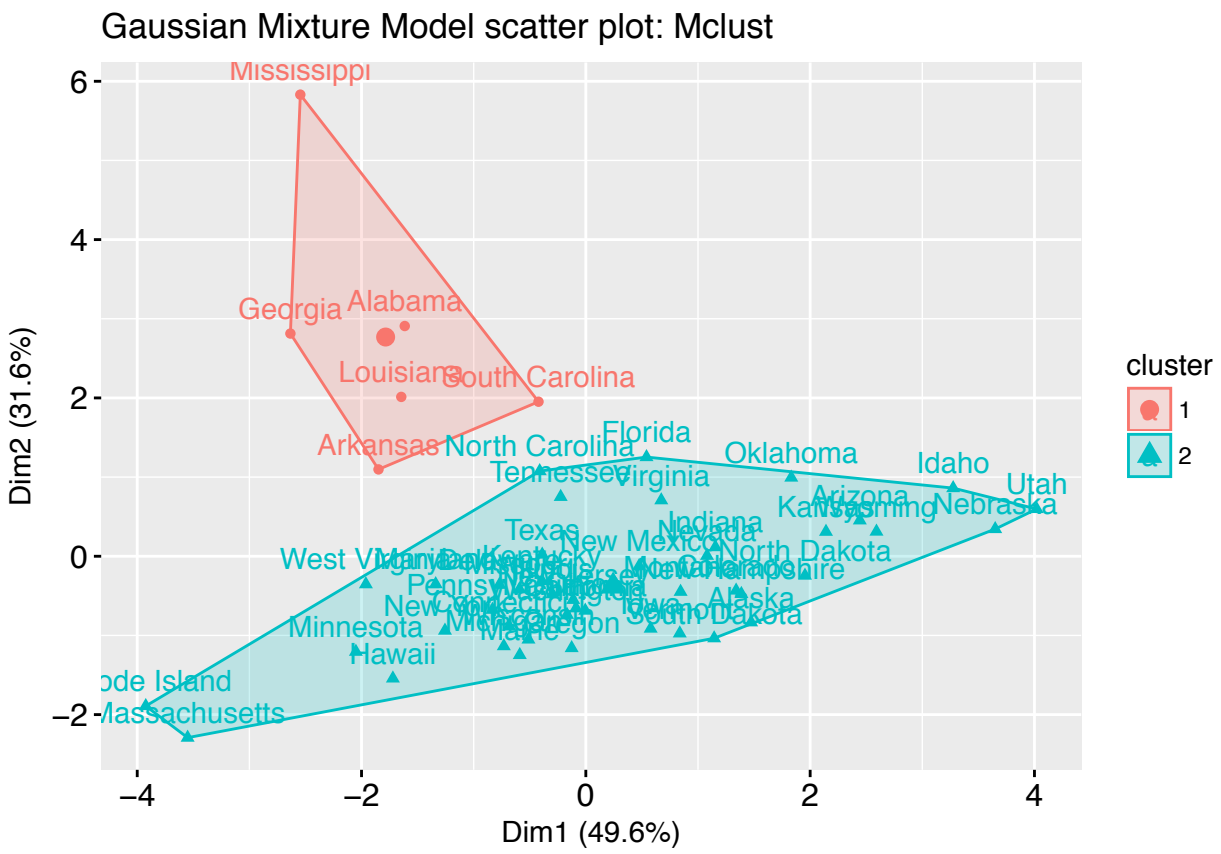
```
# estimated probability for an observation to be in each cluster
fviz_cluster(df.mc, frame.type="norm", geom="point") +
  ggtitle("Gaussian Mixture Model")
```

```
## Warning: argument frame is deprecated; please use ellipse instead.
```

```
## Warning: argument frame.type is deprecated; please use ellipse.type
## instead.
```

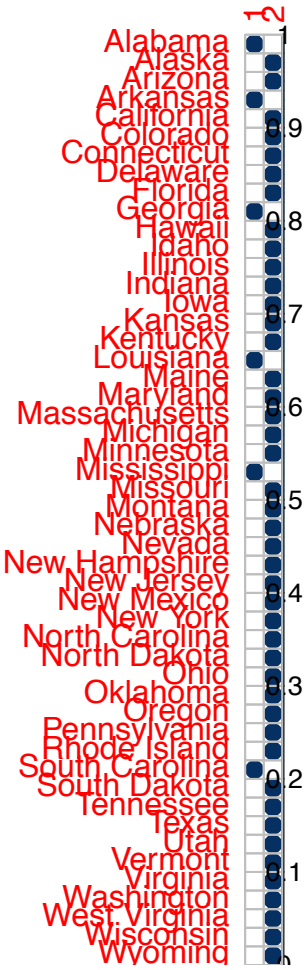



```
# estimated probability for an observation to be in each cluster
fviz_cluster(df.mc) +
  ggtitle("Gaussian Mixture Model scatter plot: Mclust")
```



Compared to the fuzzy clustering (fanny) method, which chooses to classify the data points into three clusters, the optimal number of clusters selected by the `Mclust` function is 2. This assumes the data come from a mixture of 2 multivariate normal distributions with mean vector μ_k and covariance matrix Σ_k. We see that both Republican states and Demoratic states are classified into one common cluster, while states such as Georgia, Alabama, Arkansas - which are predicted to be in the "swing states" cluster by other algorithms above, form their own cluster.

```
corrplot(df.mc$z, is.corr=F)
```



From the correlation heat map, the algorithm is able to classify datapoints into each cluster with fairly high certainty, as we see dark circles in the visual representation above. There is a large separation between the two clusters.

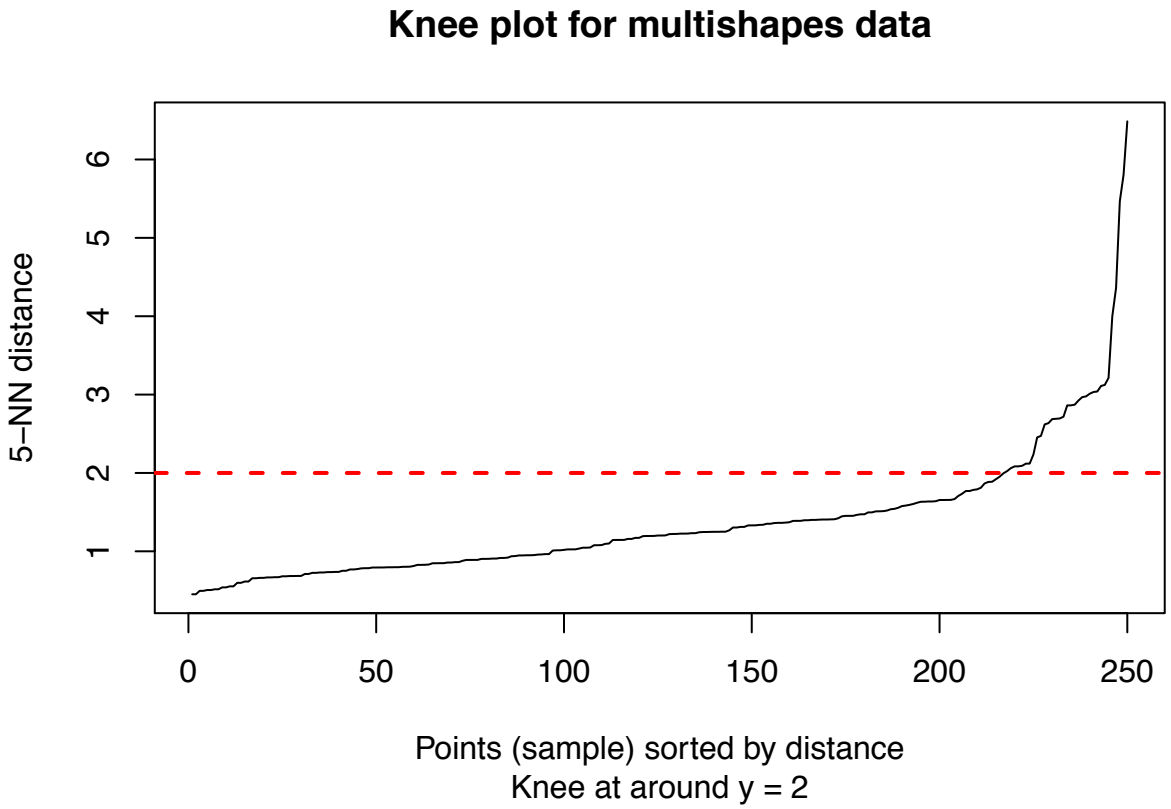
Part 2e: Density-based clustering

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?

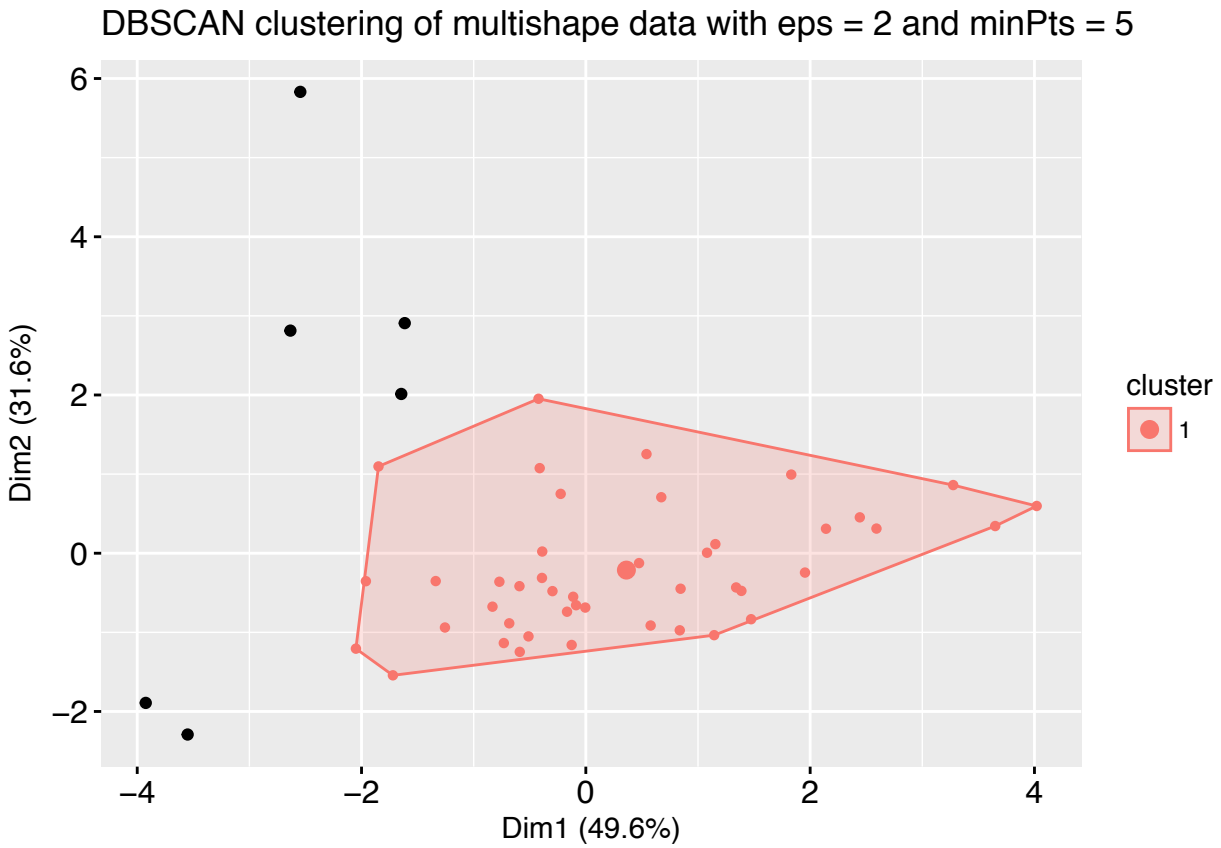
Answer

```
library(dbscan)

set.seed(123)
kNNdistplot(scale(df), k = 5)
abline(2, 0, lty=2, lwd = 2, col = "red") # added after seeing kNN plot
title(sub="Knee at around y = 2",main="Knee plot for multishapes data")
```




```
df.db = dbscan(scale(df), eps = 2, minPts = 5)
fviz_cluster(df.db, scale(df), stand = FALSE, geom = "point") +
  ggtitle("DBSCAN clustering of multishape data with eps = 2 and minPts = 5")
```



The density-based clustering is good at picking out clusters that are irregular in shape. The ϵ defines a radius of a neighborhood around an observation, and `MinPts` is the minimum number of points within an ϵ radius of an observation to be considered a core point. Here, the knee plot identifies the size of the ϵ neighborhood to be 2. Dbscan also identifies observations that do not belong to clusters as outliers, which are the black points From the scatter plot, we see that the cluster.

The density-based clustering (Dbscan) classifies both Democratic and Republican states into one cluster, with a few outliers: Rhoad Island, Massachusetts, Louisiana, Alabama, Georgia, and Mississippi. This is different from other methods, where most methods classify the states into three clusters, and MClust clusters the states into two clusters. Dbscan is good at picking out clusters with irrugular shapes, such as circular/donut shaped clusters, and probably isn't the most appropriate algorithm to use here.