S&DS 563 / F&ES 758b - Multivariate Statistics Homework #3 Cluster Analysis

Lanxin Jiang (lj345), Grace Sun (ys544), Chenglin Lu (cl939)

2018-02-24

1. Think about what metrics are appropriate for your data based on data type. Write a few sentences about this. Also think about whether you should standardize or transform your data (comment as appropriate).

```r
library(RWeka)
CKD <- read.arff("../Chronic_Kidney_Disease/chronic_kidney_disease_full.arff")
# Only Numeric Variables
CKD.numeric <- CKD[,c(1:2,10:18)]
# Remove Missing Observations
CKD.numeric <- CKD.numeric[complete.cases(CKD.numeric),]

# get standard deviation for each patient
round(sqrt(apply(CKD.numeric,2,var)),2) %>% pander
```

age	bp	bgr	bu	sc	sod	pot	hemo	pcv	wbcc	rbcc
15.52	14.32	75.95	45.43	2.91	6.85	3	2.91	9.18	2899	1.04

```r
CKD.Norm <- scale(CKD.numeric)
```

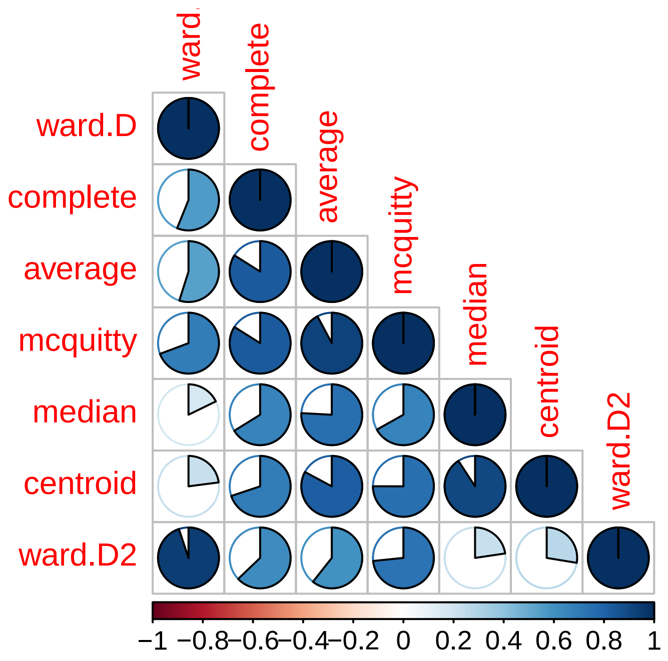
There are 213 observations with 11 dimensions in our data. All of the parameters are continuous variables.Because these measurements could be plotted as scatter plots, we could simply measure the distances between the pairs of points by Euclidean distance.There are 213 observations with 11 dimensions in our data. All of the parameters are continuous variables, and scaled parameters have either negative or positive values.Because these measurements could be plotted as scatter plots, we could simply measure the distances between the pairs of points by Euclidean distance. We also tried another metric:the Maxium Distance.

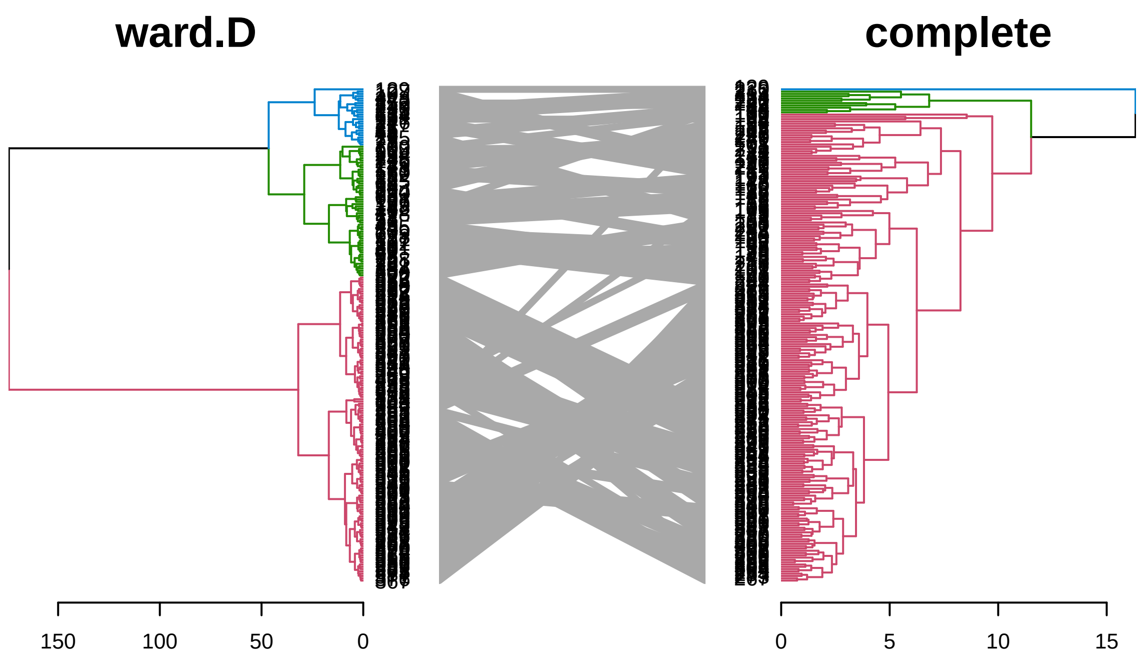
2. Try various forms of hierarchical cluster analysis. Try at least two different metrics and two agglomeration methods. Produce dendrograms and comment on what you observe.

```r
K <- 3
```

Method 1: Euclidean distance

```r
library(cluster)
library(vegan)
library(aplpack)
library(fpc)
library(ape)
library(dplyr)
library(foreach)
library(dendextend)

# Euclidean distance | Get the distance matrix
dist <- dist(CKD.Norm, method="euclidean")

vis.method.hierarchical.cluster <- function(dist, agglomerations=c("ward.D", "ward.D2")){
    methods <- c("ward.D", "complete", "average", "mcquitty", "median", "centroid", "ward.D2")
    CKD.dendlist <- foreach(method=methods, .combine=dendlist) %do% {
        hclust(dist, method=method) %>% as.dendrogram
    }
    names(CKD.dendlist) <- methods

    CKD.dendlist_cor <- cor.dendlist(CKD.dendlist)
    corrplot::corrplot(CKD.dendlist_cor, "pie", "lower")
    plot.new()

    CKD.dendlist %>%
        dendlist(which = names(CKD.dendlist) %in% agglomerations) %>%
        ladderize %>%
        set("branches_k_color", k=K) %>%
        tanglegram(faster = TRUE)
}

vis.method.hierarchical.cluster(dist, agglomerations=c("ward.D", "complete"))
```

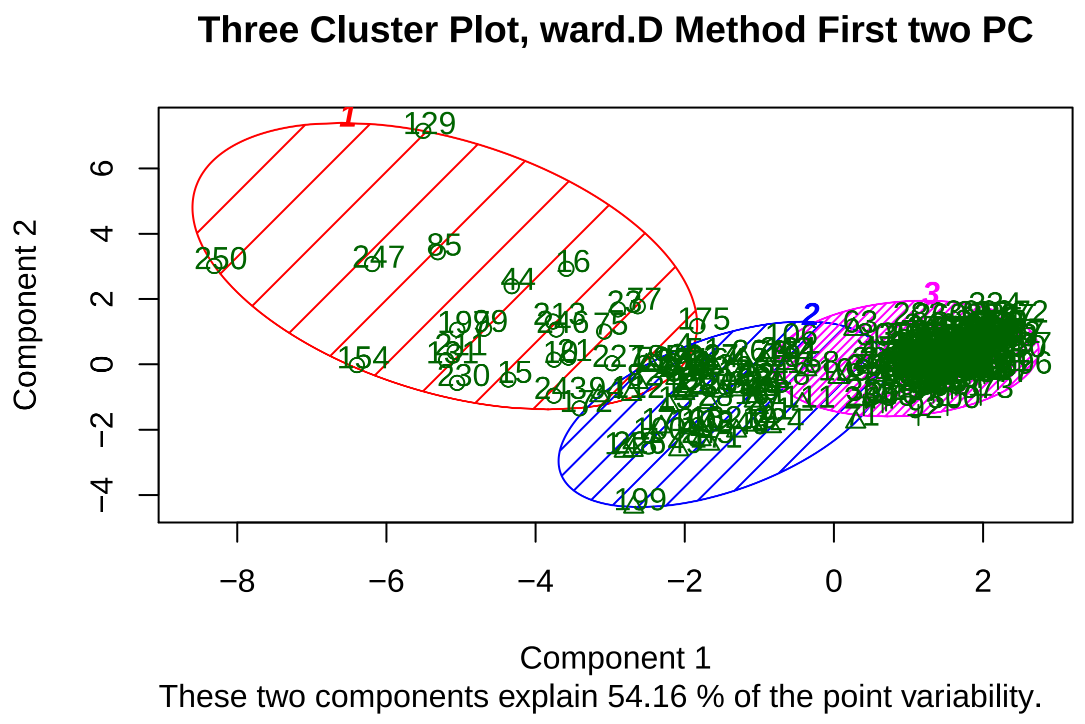


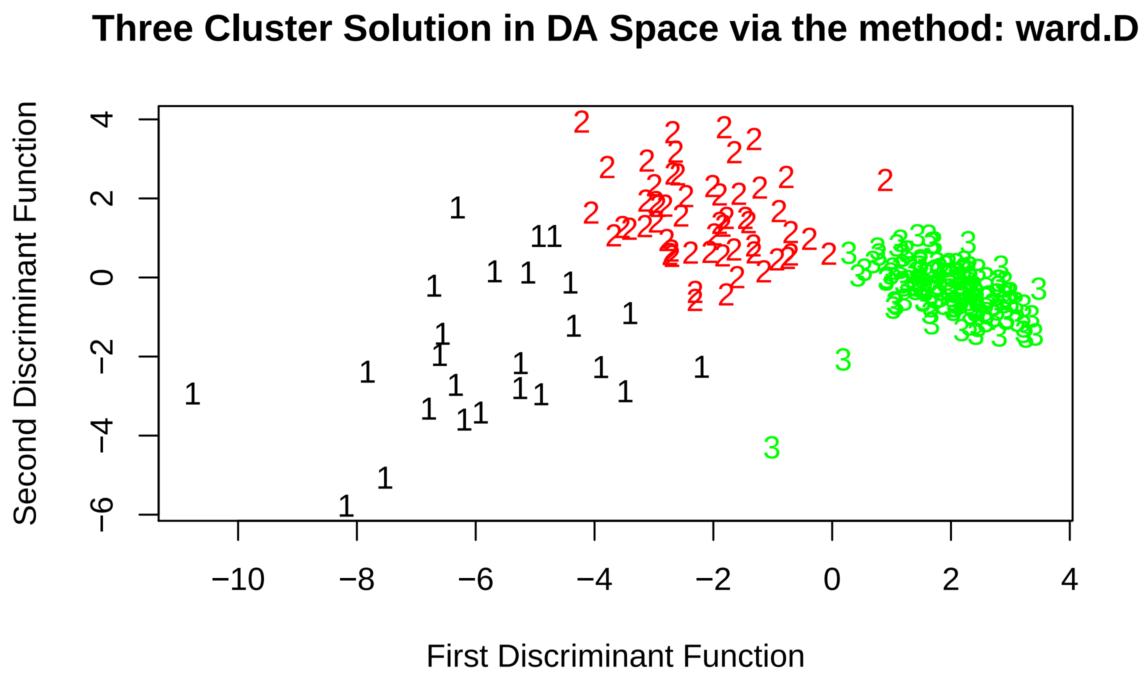


From the above correlation pie figure, we can easily see that different clustering methods yield very distinct correlation.Here we could compare Ward's method that minimizing internal sum of squares and complete linkage.

From the above dendrograms We see that the two algorithms perform quite different for all of the three clusters. The following analysis compares which performs better.

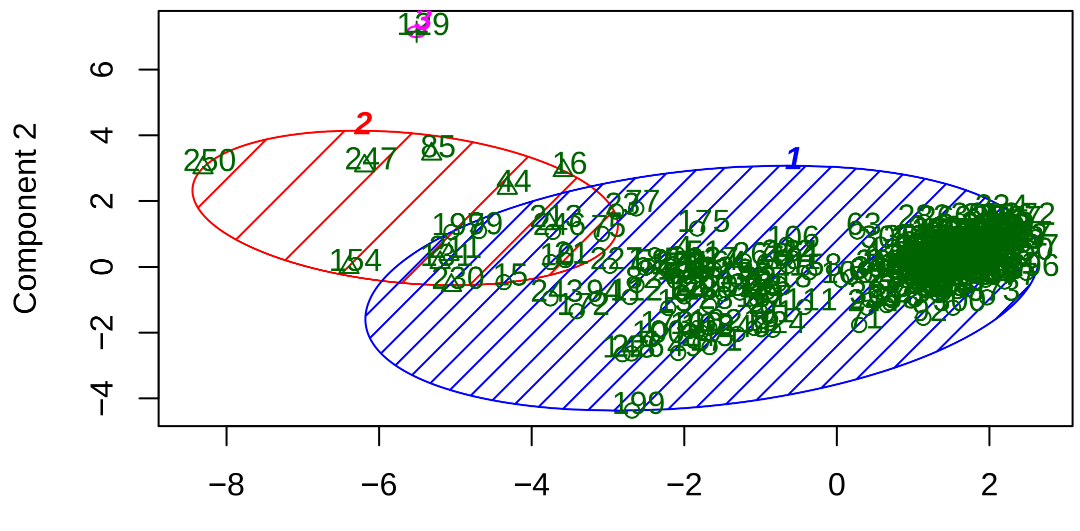
```
vis.hierarchical.cluster <- function(dist, agglomeration="average"){
    # Clustering;
    clust <- hclust(dist, method=agglomeration)

    # draw the dendrogram
    #plot(clust, labels=row.names(CKD.numeric), cex=0.5, xlab="", ylab="Distance", main="Clustering for Pa
    #rect.hclust(clust, k=K)
    #plot.new()
    #plot.new()
    # Get membership vector
    cuts <- stats::cutree(clust, k=K)

    # Make plot of three cluster solution in space desginated by first two principal components
    clusplot(CKD.Norm, cuts, color=TRUE, shade=TRUE, labels=2, lines=0,main=paste("Three Cluster Plot,", a
    plot.new()

    # Make plot of three cluster solution in space desginated by first two discriminant functions
    plotcluster(CKD.Norm, cuts, main=paste("Three Cluster Solution in DA Space via the method:", agglomera
}

vis.hierarchical.cluster(dist, agglomeration="ward.D")
```





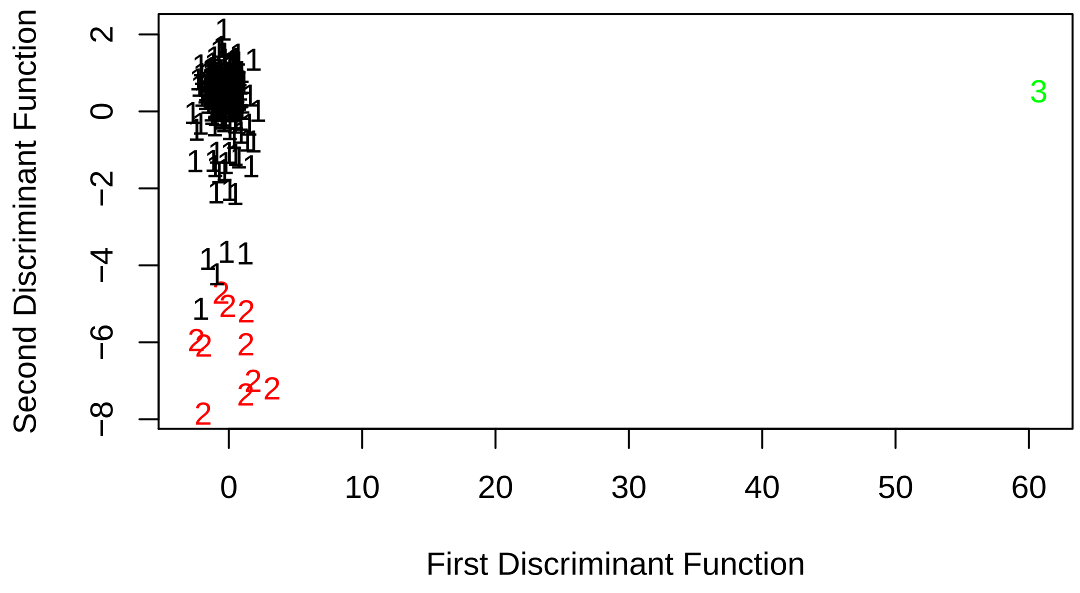
```
vis.hierarchical.cluster(dist, agglomeration="complete")
```

Three Cluster Plot, complete Method First two PC



These two components explain 54.16 % of the point variability.

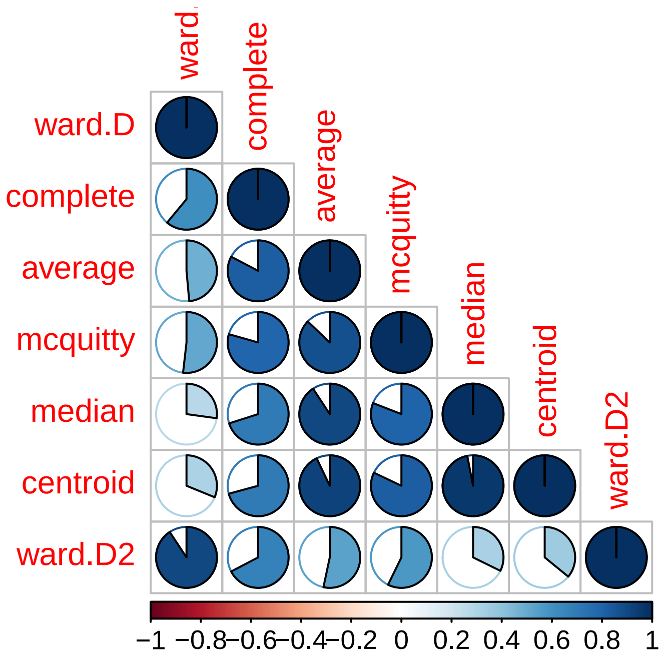
Three Cluster Solution in DA Space via the method: complete

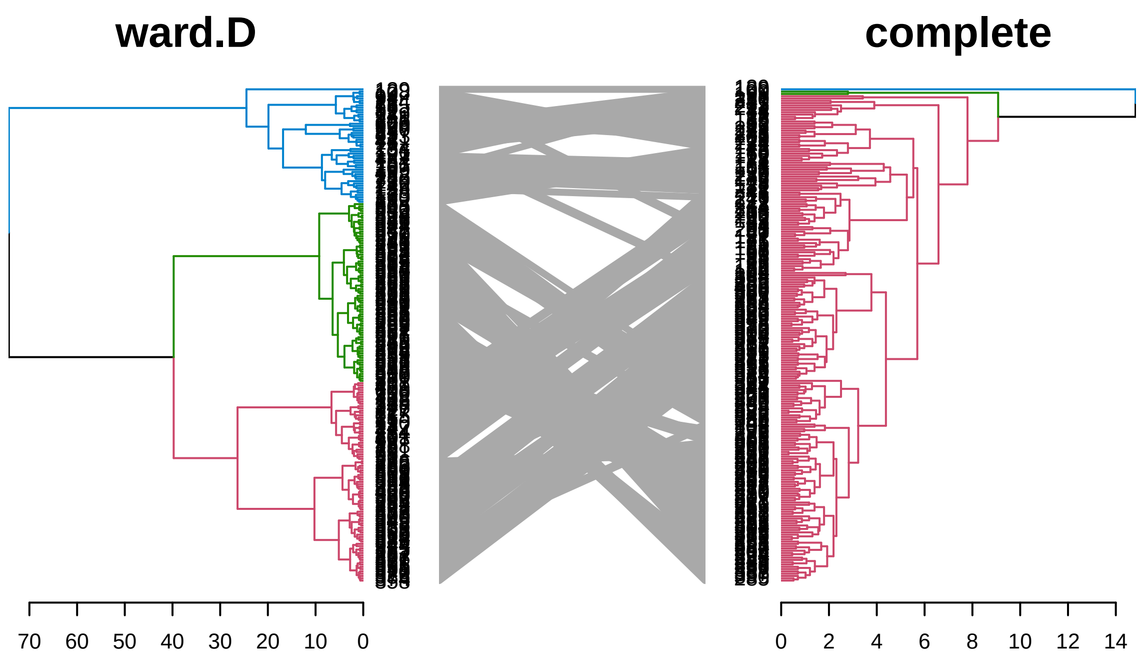


From the above component and discriminant plot, we could see that the Ward's method did a much better clustering than the complete linkage.

Method 2: Jaccard Distance

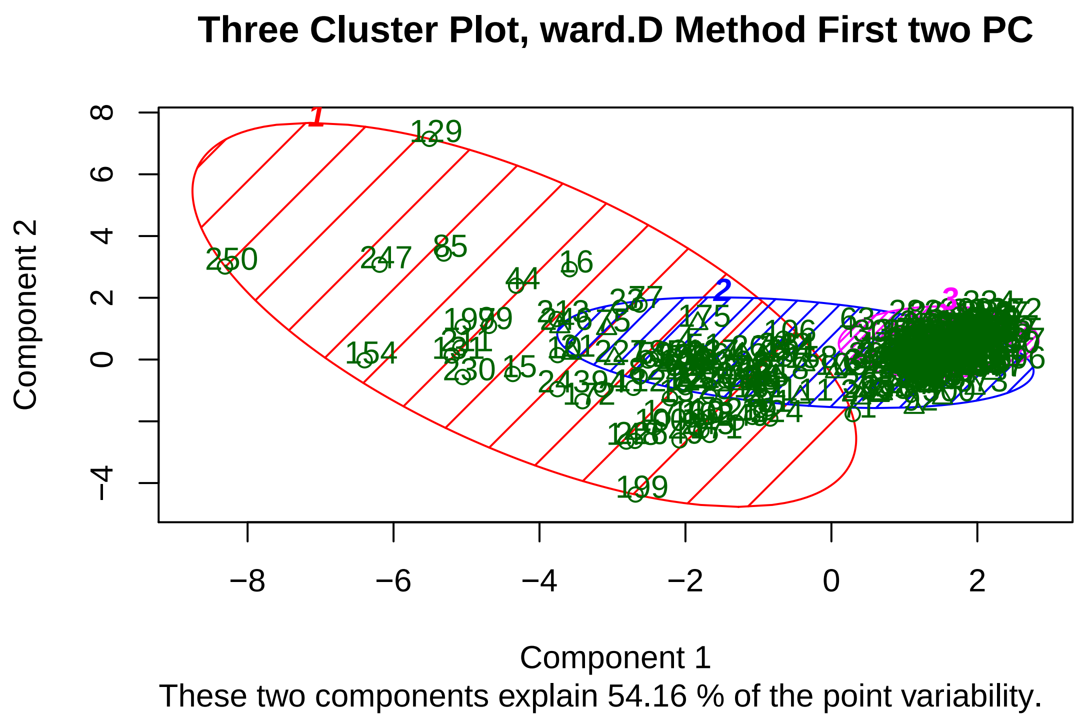
```r
# Maximum distance / Get the distance matrix
dist2 <- dist(CKD.Norm, method="maximum")

vis.method.hierarchical.cluster(dist2, agglomerations=c("ward.D", "complete"))
```

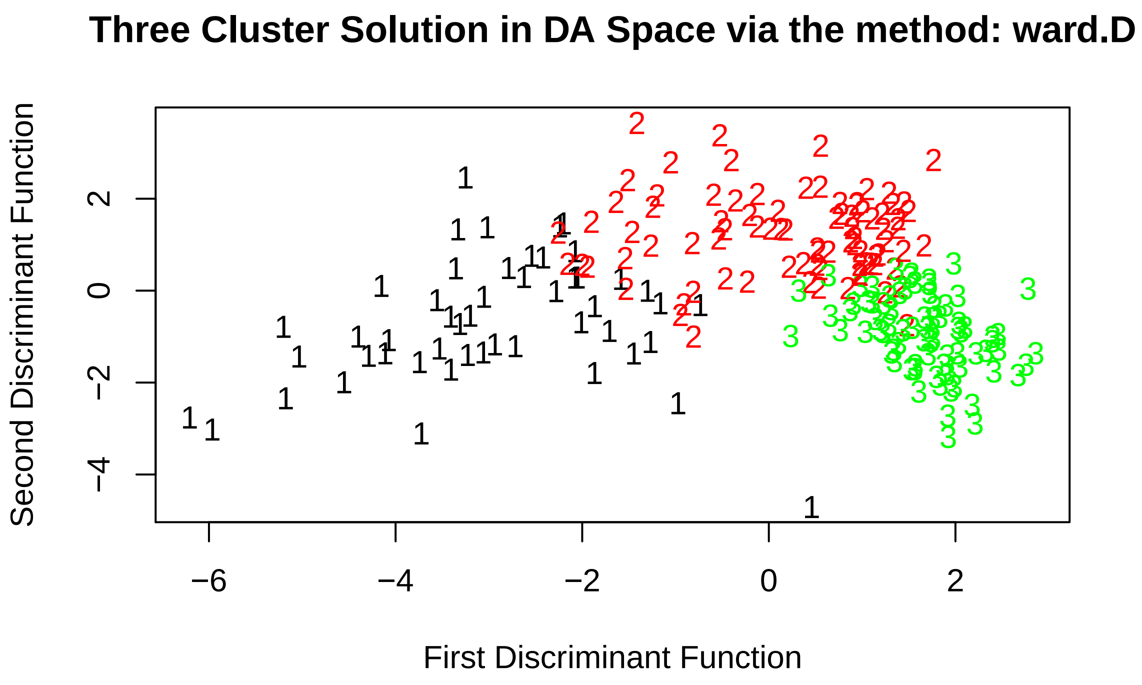

ward.D
complete
average
mcquitty
median
centroid
ward.D2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1



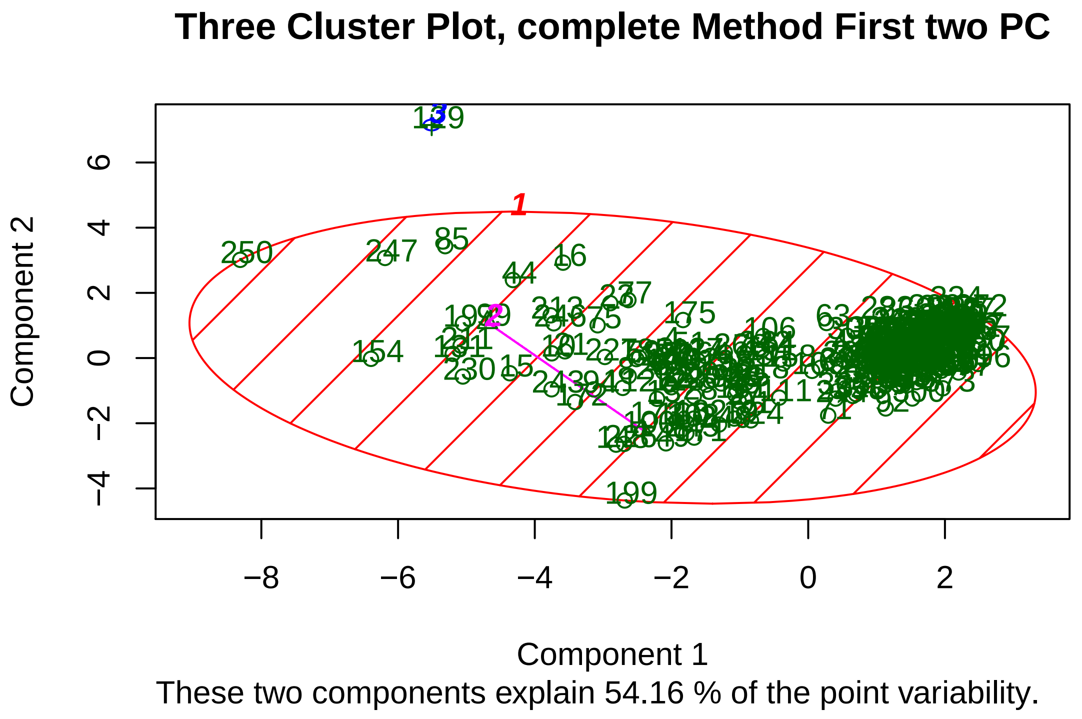
The Maximum produce very similar results for different agglomerations. Still we compare the Ward's method and the complete linkage here. These two methods vary a lot in the clustering.

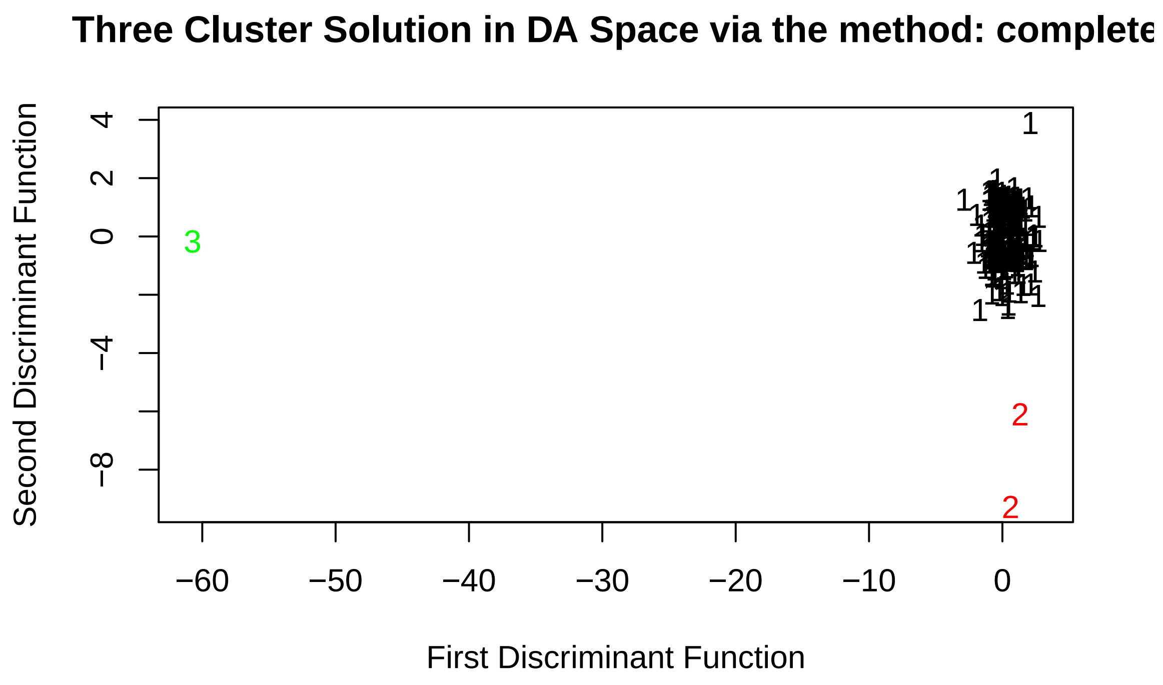
```
vis.hierarchical.cluster(dist2, agglomeration="ward.D")
```





```r
vis.hierarchical.cluster(dist2, agglomeration="complete")
```



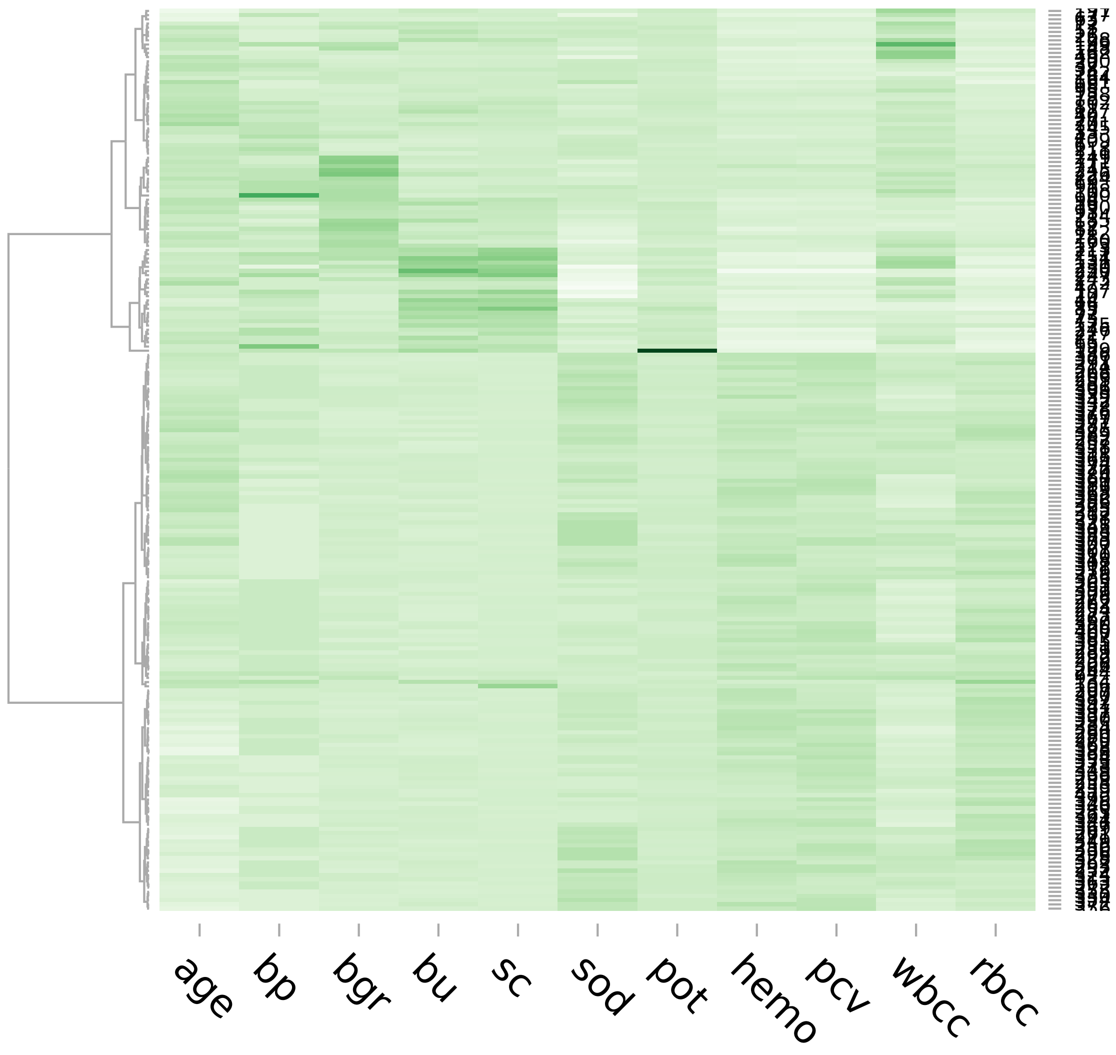


The Ward's method seems to cluster different types of patients better than the complete linkage. In addition, the Euclidean distances seems to cluster better than the Maximum distance based on the discriminant plot.
Thus, we used the below heatmap to see how those clusters display differences in each patient when we use the Euclidean distance.

```
library(d3heatmap)
clust <- hclust(dist, method="ward.D")
dend <- as.dendrogram(clust)
dend <- rotate(dend, 1:213)
```

```
## Warning in weights_for_order[order_x[order]] <- weights: number of items to
## replace is not a multiple of replacement length
```

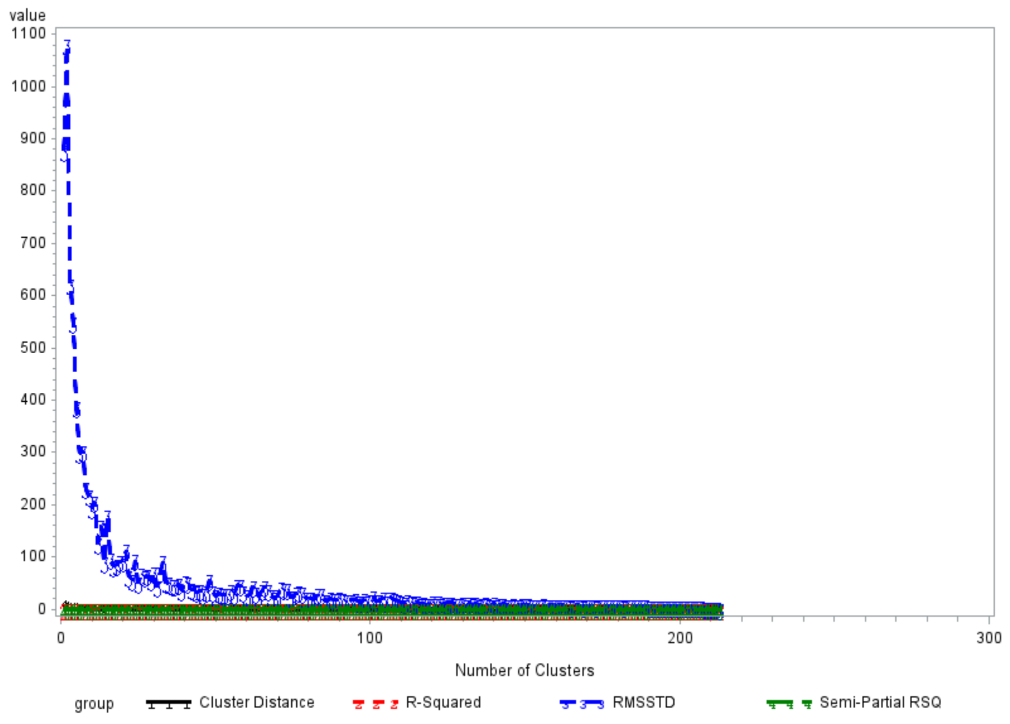
```
d3heatmap::d3heatmap(as.matrix(CKD.Norm),
          dendrogram = "row",
          Rowv = dend,
          colors = "Greens",
          # scale = "row",
          width = 600,
          show_grid = FALSE)
```



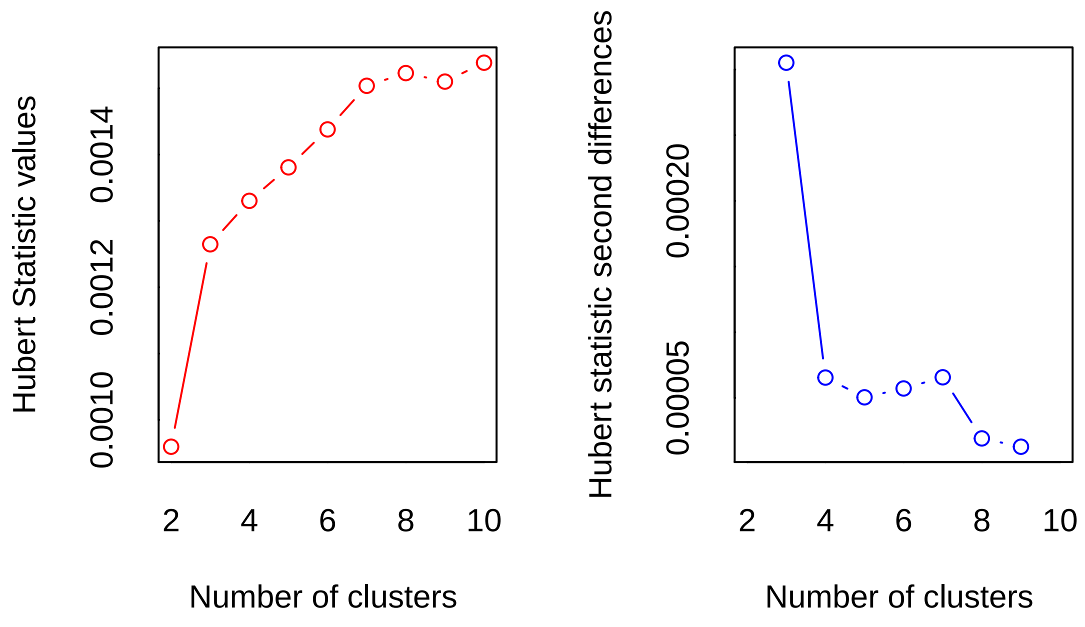
The rows are ordered based on the order of the hierarchical clustering (using the "ward" method). The colored bar indicates which category each patient belongs to. The color from light green to dark green indicates how the distacne of the patients in each measurement. We can see that some patients are quite diffferent from others in bu(blood urea), sc(serum creatinine), and sod(sodium), but overall the distinction between each patient is very hard to observe. That might be the reason why different clutering methods would get quite different results.

3. If possible, run the SAS macro to think about how many groups you want to retain. If you can't run this, discuss how many groups you think are present.

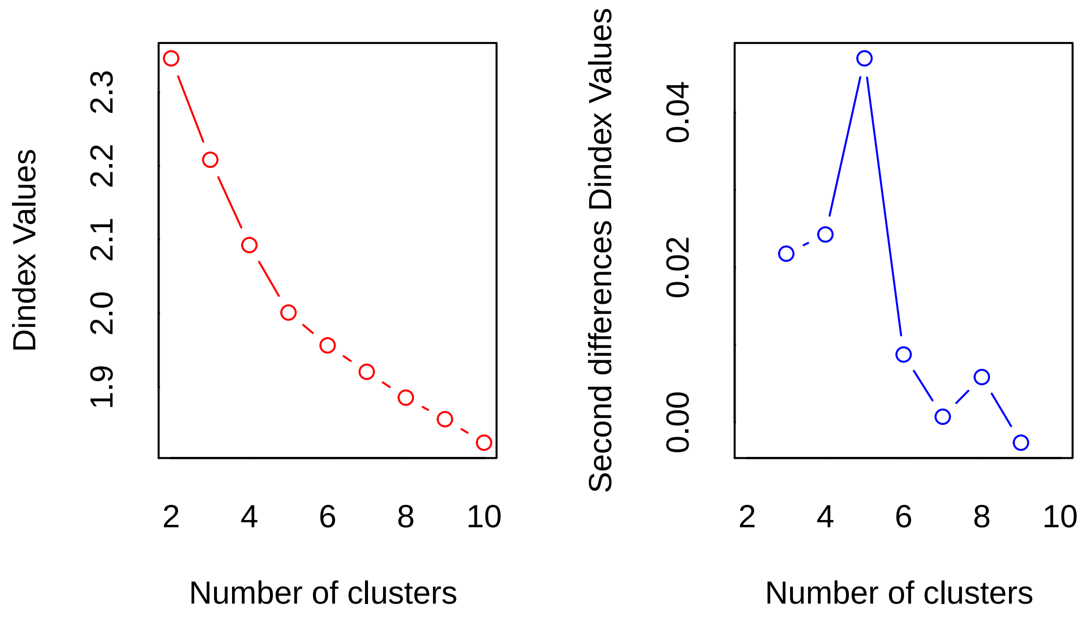
   See SAS code



```r
library("NbClust")
nb <- NbClust(CKD.Norm, distance="euclidean", min.nc=2, max.nc=10, method="kmeans")
```



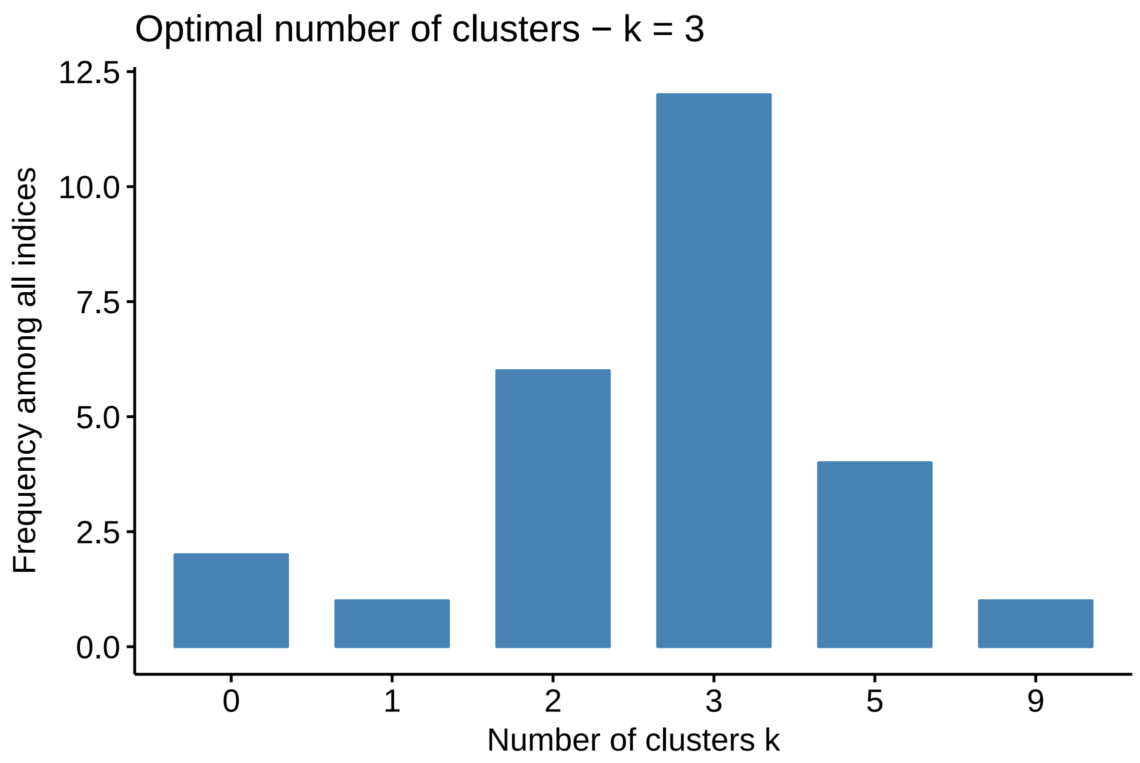
```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 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:
## * 6 proposed 2 as the best number of clusters
## * 12 proposed 3 as the best number of clusters
## * 4 proposed 5 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  3
##
##
## *******************************************************************
```

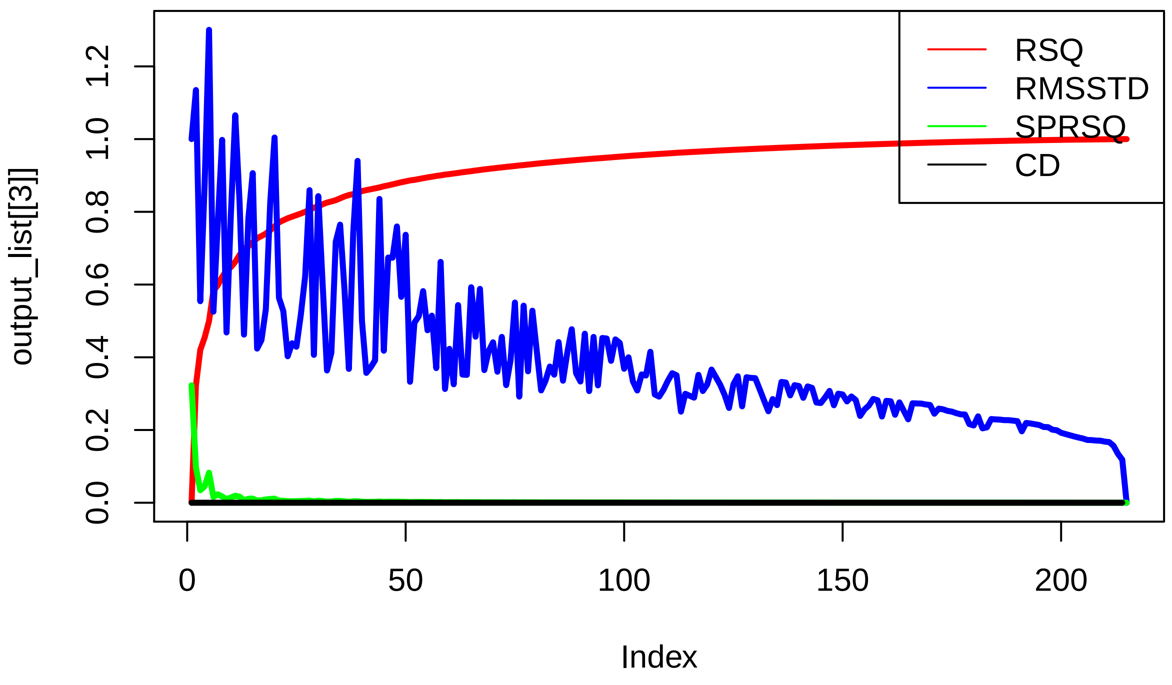
```
library("factoextra")
fviz_nbclust(nb)
```

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



```
#Evaluate Number of Clusters
source("http://reuningscherer.net/stat660/R/HClusEval.R.txt")
hclus_eval(CKD.Norm, dist_m='euclidean', clus_m='ward', plot_op=T)
```

```
## [1] "Creating Distance Matrix using euclidean"
## [1] "Clustering using ward"

## The "ward" method has been renamed to "ward.D"; note new "ward.D2"

## [1] "Clustering Complete. Access the Cluster object in first element of output"
## [1] "Calculating RMSSTD"
## [1] "RMSSTD Done. Access in Element 2"
## [1] "Calculating RSQ"
## [1] "RSQ Done. Access in Element 3"
## [1] "Calculating SPRSQ"
## [1] "SPRSQ Done. Access in Element 4"
## [1] "Calculating Cluster Dist. "
## [1] "CD Done. Access in Element 5"
```



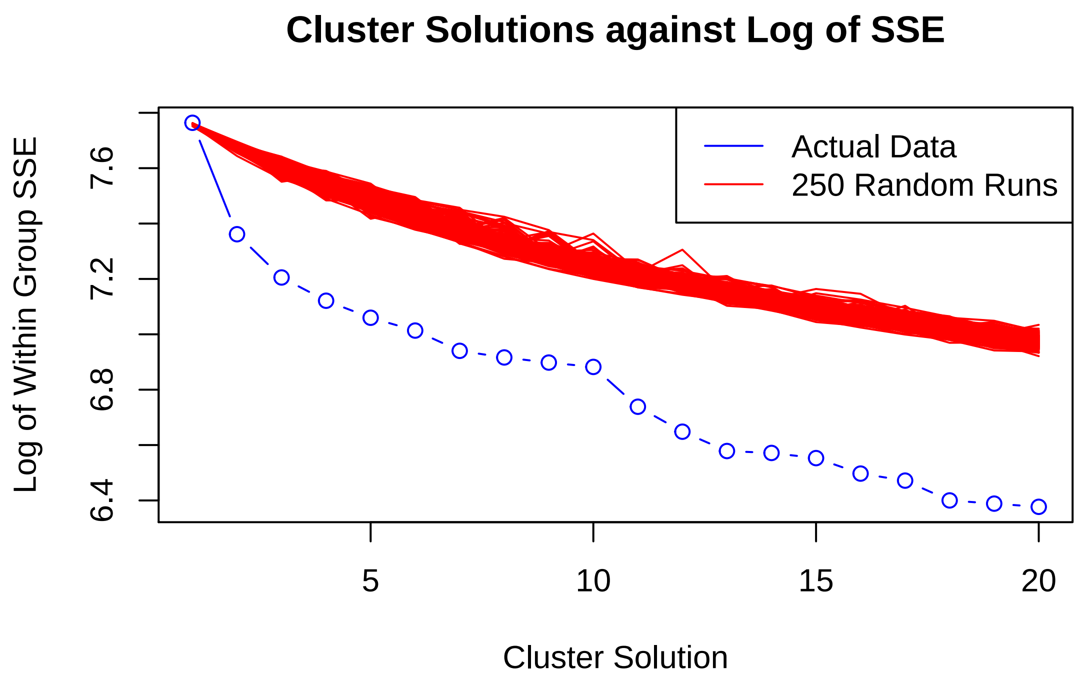
```
## [[1]]
##
## Call:
## hclust(d = dist1, method = clus_m)
##
## Cluster method   : ward.D
## Distance         : euclidean
## Number of objects: 215
##
##
## [[2]]
##   [1] 1.0000000 1.1349533 0.5542121 0.8758064 1.3004291 0.5253641 0.7690581
##   [8] 0.9978749 0.4682670 0.7924972 1.0655423 0.8244467 0.4623501 0.7814451
##  [15] 0.9064196 0.4238169 0.4472258 0.5320964 0.8242411 1.0043775 0.5635828
##  [22] 0.5268448 0.4028117 0.4386522 0.4288970 0.5160546 0.6227814 0.8596607
##  [29] 0.4065364 0.8431808 0.6005670 0.3636969 0.4128741 0.7178405 0.7648641
##  [36] 0.5859705 0.3681265 0.7428197 0.9401734 0.4999975 0.3570517 0.3725527
##  [43] 0.3917498 0.8354649 0.4179955 0.6741943 0.6737724 0.7597768 0.5662119
##  [50] 0.7369596 0.3325926 0.4951898 0.5129502 0.5820353 0.4742964 0.5145058
##  [57] 0.3702352 0.6623628 0.3127925 0.4232423 0.3257436 0.5435966 0.3524948
##  [64] 0.3515765 0.5927496 0.4565406 0.5881676 0.3647331 0.4172860 0.4412440
##  [71] 0.3602321 0.4560697 0.3234321 0.3917168 0.5506444 0.2918322 0.5420677
##  [78] 0.3613346 0.5281244 0.4174025 0.3088377 0.3355027 0.3745948 0.3521819
##  [85] 0.4417890 0.3355807 0.4136836 0.4771364 0.3561727 0.3333301 0.4650094
##  [92] 0.3070095 0.4560455 0.3226941 0.4531000 0.4516216 0.3901419 0.4492042
##  [99] 0.4395172 0.3683211 0.3998974 0.3338722 0.3087108 0.3528598 0.3498007
## [106] 0.4152026 0.2979395 0.2917816 0.3104466 0.3354102 0.3562117 0.3504858
## [113] 0.2504555 0.2996114 0.2941233 0.2889675 0.3516677 0.3072694 0.3250031
## [120] 0.3663028 0.3454324 0.3240376 0.2963764 0.2606865 0.3255821 0.3478419
## [127] 0.2650337 0.3452318 0.3432850 0.3425516 0.3125913 0.2817255 0.2515320
## [134] 0.2850574 0.2684602 0.3322466 0.3305574 0.2951674 0.3234503 0.3209003
## [141] 0.2884586 0.3199322 0.3159152 0.2755227 0.2741485 0.2896675 0.3075587
## [148] 0.2680154 0.2999543 0.2977591 0.2785578 0.2921928 0.2824501 0.2386376
## [155] 0.2564737 0.2672627 0.2854459 0.2817660 0.2370870 0.2802727 0.2790467
## [162] 0.2418900 0.2759994 0.2528722 0.2294571 0.2735953 0.2730876 0.2726226
## [169] 0.2701920 0.2687800 0.2447347 0.2588850 0.2567728 0.2526122 0.2505644
## [176] 0.2463484 0.2431594 0.2426022 0.2162569 0.2125118 0.2376363 0.2046179
## [183] 0.2072639 0.2301395 0.2291904 0.2285406 0.2271824 0.2271152 0.2257742
## [190] 0.2242842 0.1963716 0.2194497 0.2179911 0.2158697 0.2136392 0.2083003
## [197] 0.2079645 0.2008708 0.1992950 0.1923776 0.1889787 0.1856008 0.1822600
## [204] 0.1792959 0.1765855 0.1726868 0.1722487 0.1709791 0.1705516 0.1680555
## [211] 0.1668381 0.1566286 0.1346071 0.1184261 0.0000000
##
## [[3]]
##   [1] 0.0000000 0.3229833 0.4201646 0.4544501 0.4995700 0.5822076 0.5980802
##   [8] 0.6212826 0.6385533 0.6478920 0.6623976 0.6815400 0.6982522 0.7050288
##  [15] 0.7159115 0.7273510 0.7334643 0.7401191 0.7492936 0.7597761 0.7709844
##  [22] 0.7767516 0.7823175 0.7867229 0.7908532 0.7952266 0.8001009 0.8053670
##  [29] 0.8114451 0.8150750 0.8207989 0.8253710 0.8285393 0.8319886 0.8371259
##  [36] 0.8421962 0.8460826 0.8487772 0.8528205 0.8569510 0.8597547 0.8620703
##  [43] 0.8644926 0.8670386 0.8703003 0.8727166 0.8755916 0.8784174 0.8812745
##  [50] 0.8837865 0.8863244 0.8879918 0.8901568 0.8924085 0.8946316 0.8966025
##  [57] 0.8986346 0.9002769 0.9023270 0.9037187 0.9054452 0.9069980 0.9088569
##  [64] 0.9103314 0.9118898 0.9135316 0.9151274 0.9167439 0.9180636 0.9194769
##  [71] 0.9209157 0.9221788 0.9236310 0.9247591 0.9260968 0.9275137 0.9285965
##  [78] 0.9299696 0.9311541 0.9324574 0.9337518 0.9347778 0.9358832 0.9370188
##  [85] 0.9380691 0.9392059 0.9402572 0.9413059 0.9423697 0.9433421 0.9443149
##  [92] 0.9453254 0.9461895 0.9471614 0.9479126 0.9488719 0.9498250 0.9507693
##  [99] 0.9517122 0.9526149 0.9535260 0.9543935 0.9551880 0.9560049 0.9568046
## [106] 0.9576068 0.9584123 0.9591600 0.9598886 0.9606401 0.9614194 0.9622078
## [113] 0.9629569 0.9635987 0.9642647 0.9649180 0.9655374 0.9661774 0.9667816
## [120] 0.9674128 0.9680398 0.9686661 0.9692803 0.9698882 0.9704272 0.9709974
## [127] 0.9715628 0.9720849 0.9726418 0.9731925 0.9737408 0.9742831 0.9747998
## [134] 0.9752869 0.9758044 0.9763227 0.9768385 0.9773491 0.9778453 0.9783342
## [141] 0.9788154 0.9792996 0.9797779 0.9802443 0.9807063 0.9811623 0.9816089
## [148] 0.9820509 0.9824811 0.9829015 0.9833158 0.9837278 0.9841267 0.9845250
## [155] 0.9849219 0.9853189 0.9857028 0.9860835 0.9864545 0.9868077 0.9871748
## [162] 0.9875387 0.9878999 0.9882558 0.9886124 0.9889588 0.9893086 0.9896571
## [169] 0.9900044 0.9903455 0.9906831 0.9910078 0.9913210 0.9916291 0.9919273
## [176] 0.9922206 0.9925042 0.9927805 0.9930555 0.9933257 0.9936091 0.9938730
## [183] 0.9941324 0.9943836 0.9946311 0.9948766 0.9951206 0.9953618 0.9956028
## [190] 0.9958410 0.9960761 0.9963135 0.9965385 0.9967606 0.9969784 0.9971916
## [197] 0.9973944 0.9975965 0.9977850 0.9979706 0.9981436 0.9983105 0.9984714
## [204] 0.9986267 0.9987769 0.9989226 0.9990619 0.9992006 0.9993372 0.9994731
## [211] 0.9996051 0.9997352 0.9998498 0.9999345 1.0000000
##
## [[4]]
##   [1] 3.229833e-01 9.718128e-02 3.428552e-02 4.511994e-02 8.263752e-02
##   [6] 1.587261e-02 2.320238e-02 1.727075e-02 9.338718e-03 1.450556e-02
##  [11] 1.914239e-02 1.671218e-02 6.776642e-03 1.088272e-02 1.143950e-02
##  [16] 6.113300e-03 6.654800e-03 9.174428e-03 1.048253e-02 1.120830e-02
##  [21] 5.767264e-03 5.565893e-03 4.405371e-03 4.130286e-03 4.373410e-03
##  [26] 4.874309e-03 5.266052e-03 6.078144e-03 3.629893e-03 5.723939e-03
##  [31] 4.572078e-03 3.168291e-03 3.449251e-03 5.137347e-03 5.070334e-03
##  [36] 3.886374e-03 2.694618e-03 4.043303e-03 4.130495e-03 2.803626e-03
##  [41] 2.315603e-03 2.422332e-03 2.546029e-03 3.261690e-03 2.416307e-03
##  [46] 2.874946e-03 2.825837e-03 2.857066e-03 2.512039e-03 2.537894e-03
##  [51] 1.667446e-03 2.164903e-03 2.251738e-03 2.223111e-03 1.970913e-03
##  [56] 2.032085e-03 1.642269e-03 2.050114e-03 1.391672e-03 1.726544e-03
##  [61] 1.552777e-03 1.858968e-03 1.474492e-03 1.558344e-03 1.641832e-03
##  [66] 1.595740e-03 1.616547e-03 1.319722e-03 1.413245e-03 1.438847e-03
##  [71] 1.263127e-03 1.452189e-03 1.128065e-03 1.337729e-03 1.416866e-03
##  [76] 1.082831e-03 1.373072e-03 1.184488e-03 1.303343e-03 1.294364e-03
##  [81] 1.026059e-03 1.105361e-03 1.135570e-03 1.050345e-03 1.136750e-03
##  [86] 1.051333e-03 1.048709e-03 1.063828e-03 9.723186e-04 9.728789e-04
##  [91] 1.010438e-03 8.641215e-04 9.718575e-04 7.512329e-04 9.593439e-04
##  [96] 9.530938e-04 9.442278e-04 9.429179e-04 9.026885e-04 9.110932e-04
## [101] 8.675614e-04 7.945283e-04 8.168111e-04 7.997801e-04 8.021260e-04
## [106] 8.055756e-04 7.476495e-04 7.285672e-04 7.515774e-04 7.792365e-04
## [111] 7.884412e-04 7.490840e-04 6.418414e-04 6.660042e-04 6.532632e-04
## [116] 6.194244e-04 6.399654e-04 6.041708e-04 6.312075e-04 6.269988e-04
## [121] 6.262941e-04 6.142425e-04 6.078700e-04 5.390183e-04 5.702564e-04
## [126] 5.653924e-04 5.220518e-04 5.569393e-04 5.506756e-04 5.483254e-04
## [131] 5.422179e-04 5.167304e-04 4.871317e-04 5.174898e-04 5.182786e-04
## [136] 5.158309e-04 5.105990e-04 4.961801e-04 4.888791e-04 4.812008e-04
## [141] 4.842719e-04 4.783019e-04 4.663666e-04 4.619690e-04 4.559978e-04
## [146] 4.465973e-04 4.420203e-04 4.301527e-04 4.204326e-04 4.143012e-04
## [151] 4.119974e-04 3.989561e-04 3.982850e-04 3.968553e-04 3.969989e-04
## [156] 3.839155e-04 3.807447e-04 3.709909e-04 3.531961e-04 3.670691e-04
## [161] 3.638649e-04 3.612290e-04 3.559610e-04 3.565767e-04 3.463490e-04
## [166] 3.497868e-04 3.484899e-04 3.473042e-04 3.411389e-04 3.375827e-04
## [171] 3.247041e-04 3.131843e-04 3.080947e-04 2.981913e-04 2.933763e-04
## [176] 2.835865e-04 2.762920e-04 2.750273e-04 2.701924e-04 2.834247e-04
## [181] 2.638832e-04 2.593191e-04 2.512599e-04 2.474961e-04 2.454590e-04
## [186] 2.440692e-04 2.411769e-04 2.410341e-04 2.381961e-04 2.350627e-04
## [191] 2.374010e-04 2.250382e-04 2.220567e-04 2.177558e-04 2.132791e-04
## [196] 2.027524e-04 2.020992e-04 1.885471e-04 1.856005e-04 1.729399e-04
## [201] 1.668829e-04 1.609704e-04 1.552276e-04 1.502198e-04 1.457123e-04
## [206] 1.393492e-04 1.386431e-04 1.366068e-04 1.359245e-04 1.319751e-04
## [211] 1.300699e-04 1.146379e-04 8.466856e-05 6.553622e-05 0.000000e+00
##
## [[5]]
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [71] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [106] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [141] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [176] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [211] 0 0 0 0
```

4. Run k-means clustering on your data. Compare results to what you got in 3. Include a sum of squares vs. k plot and comment on how many groups exist.

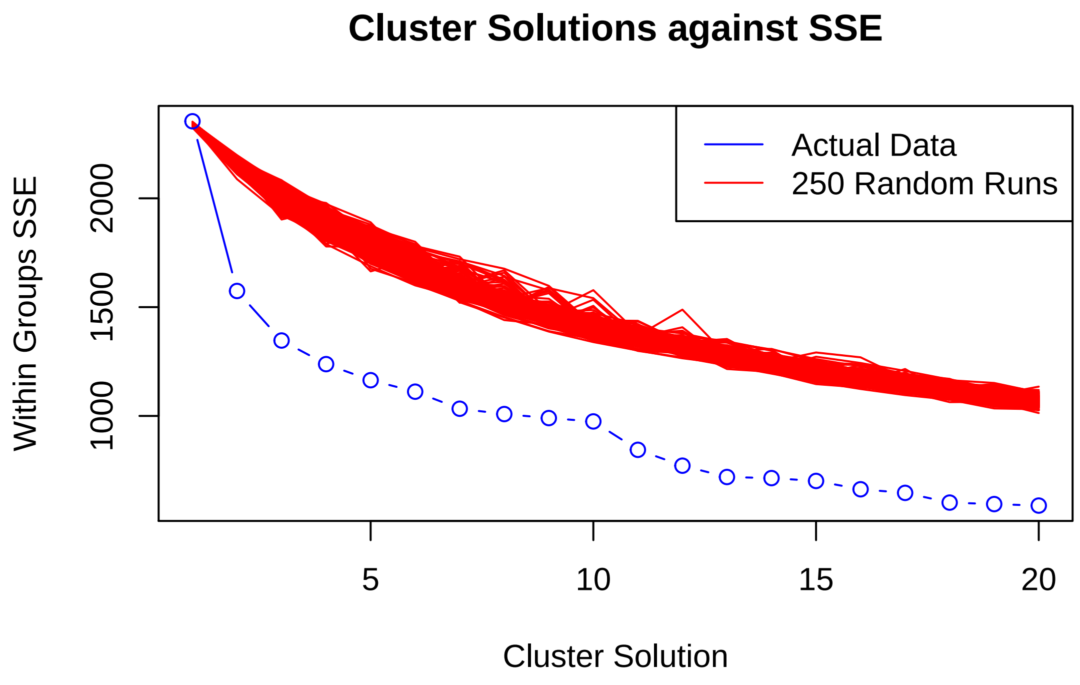
```
# Modified Script by Matt Peeples http://www.mattpeeples.net/kmeans.html
#   Produces screeplot like diagram with randomized comparison based
#   on randomization within columns (i.e. as if points had been randomly assigned
#   data values, one from each column.  Keeps total internal SS the same.

#kdata is just normalized input dataset
kdata=CKD.Norm
n.lev=20  #set max value for k

# Calculate the within groups sum of squared error (SSE) for the number of cluster solutions selected by 
wss <- rnorm(10)
while (prod(wss==sort(wss,decreasing=T))==0) {
  wss <- (nrow(kdata)-1)*sum(apply(kdata,2,var))
  for (i in 2:n.lev) wss[i] <- sum(kmeans(kdata, centers=i)$withinss)}
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
```

```
# Calculate the within groups SSE for 250 randomized data sets (based on the original input data)
k.rand <- function(x){
  km.rand <- matrix(sample(x),dim(x)[1],dim(x)[2])
  rand.wss <- as.matrix(dim(x)[1]-1)*sum(apply(km.rand,2,var))
  for (i in 2:n.lev) rand.wss[i] <- sum(kmeans(km.rand, centers=i)$withinss)
  rand.wss <- as.matrix(rand.wss)
  return(rand.wss)
}

rand.mat <- matrix(0,n.lev,250)

k.1 <- function(x) {
  for (i in 1:250) {
    r.mat <- as.matrix(suppressWarnings(k.rand(kdata)))
    rand.mat[,i] <- r.mat}
  return(rand.mat)
}

# Same function as above for data with < 3 column variables
k.2.rand <- function(x){
  rand.mat <- matrix(0,n.lev,250)
  km.rand <- matrix(sample(x),dim(x)[1],dim(x)[2])
  rand.wss <- as.matrix(dim(x)[1]-1)*sum(apply(km.rand,2,var))
  for (i in 2:n.lev) rand.wss[i] <- sum(kmeans(km.rand, centers=i)$withinss)
  rand.wss <- as.matrix(rand.wss)
  return(rand.wss)
}

k.2 <- function(x){
  for (i in 1:250) {
    r.1 <- k.2.rand(kdata)
    rand.mat[,i] <- r.1}
  return(rand.mat)
}

# Determine if the data data table has > or < 3 variables and call appropriate function above
if (dim(kdata)[2] == 2) { rand.mat <- k.2(kdata) } else { rand.mat <- k.1(kdata) }

# Plot within groups SSE against all tested cluster solutions for actual and randomized data - 1st: Log s

xrange <- range(1:n.lev)
yrange <- range(log(rand.mat),log(wss))
plot(xrange,yrange, type='n', xlab='Cluster Solution', ylab='Log of Within Group SSE', main='Cluster Solut
for (i in 1:250) lines(log(rand.mat[,i]),type='l',col='red')
lines(log(wss), type="b", col='blue')
legend('topright',c('Actual Data', '250 Random Runs'), col=c('blue', 'red'), lty=1)
```



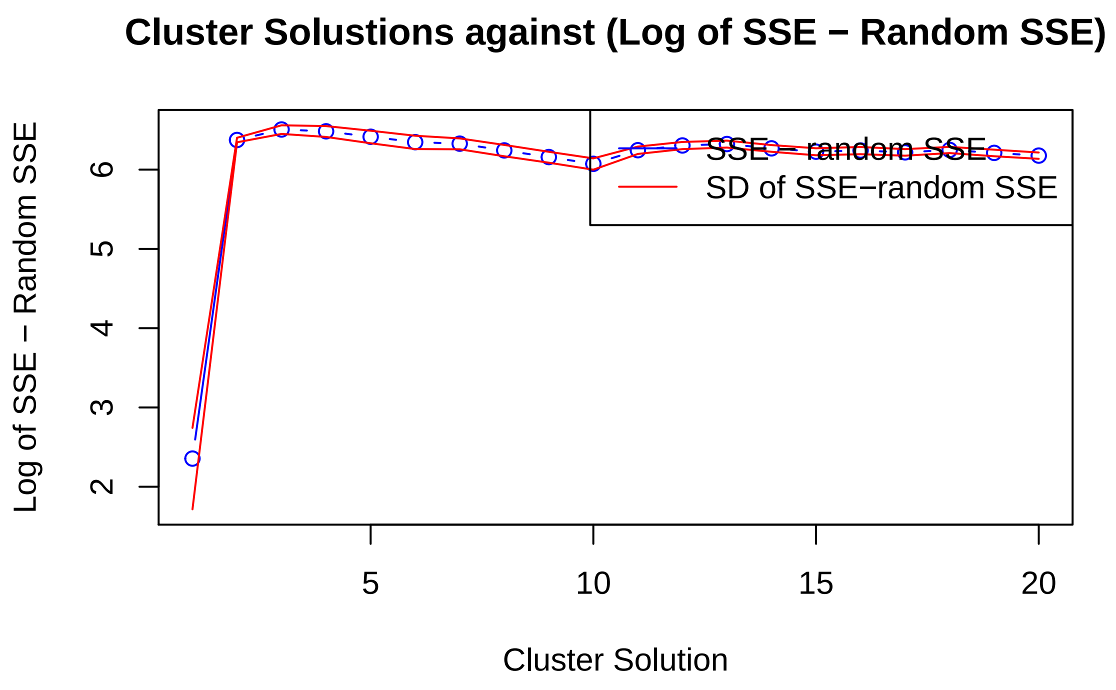
```
yrange <- range(rand.mat,wss)
plot(xrange,yrange, type='n', xlab="Cluster Solution", ylab="Within Groups SSE", main="Cluster Solutions a
for (i in 1:250) lines(rand.mat[,i],type='l',col='red')
lines(1:n.lev, wss, type="b", col='blue')
legend('topright',c('Actual Data', '250 Random Runs'), col=c('blue', 'red'), lty=1)
```



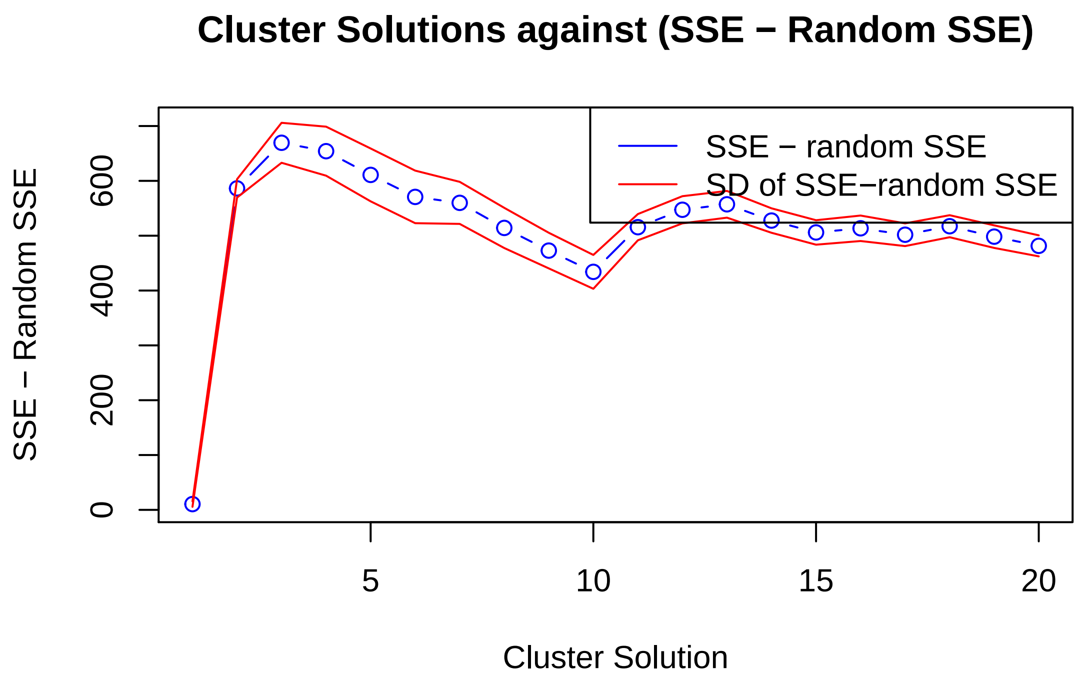
The above plots:cluster size vs log of within group SSE and within group SSE, show that within group SSE drops most when there is two clusters.

```
# Calculate the mean and standard deviation of difference between SSE of actual data and SSE of 250 rando
r.sse <- matrix(0,dim(rand.mat)[1],dim(rand.mat)[2])
```

```r
wss.1 <- as.matrix(wss)
for (i in 1:dim(r.sse)[2]) {
  r.temp <- abs(rand.mat[,i]-wss.1[,1])
  r.sse[,i] <- r.temp}
r.sse.m <- apply(r.sse,1,mean)
r.sse.sd <- apply(r.sse,1,sd)
r.sse.plus <- r.sse.m + r.sse.sd
r.sse.min <- r.sse.m - r.sse.sd

# Plot differeince between actual SSE mean SSE from 250 randomized datasets - 1st: Log scale, 2nd: Normal

xrange <- range(1:n.lev)
yrange <- range(log(r.sse.plus),log(r.sse.min))
plot(xrange,yrange, type='n',xlab='Cluster Solution', ylab='Log of SSE - Random SSE', main='Cluster Solust
lines(log(r.sse.m), type="b", col='blue')
lines(log(r.sse.plus), type='l', col='red')
lines(log(r.sse.min), type='l', col='red')
legend('topright',c('SSE - random SSE', 'SD of SSE-random SSE'), col=c('blue', 'red'), lty=1)
```



```r
xrange <- range(1:n.lev)
yrange <- range(r.sse.plus,r.sse.min)
plot(xrange,yrange, type='n',xlab='Cluster Solution', ylab='SSE - Random SSE', main='Cluster Solutions aga
lines(r.sse.m, type="b", col='blue')
lines(r.sse.plus, type='l', col='red')
lines(r.sse.min, type='l', col='red')
legend('topright',c('SSE - random SSE', 'SD of SSE-random SSE'), col=c('blue', 'red'), lty=1)
```

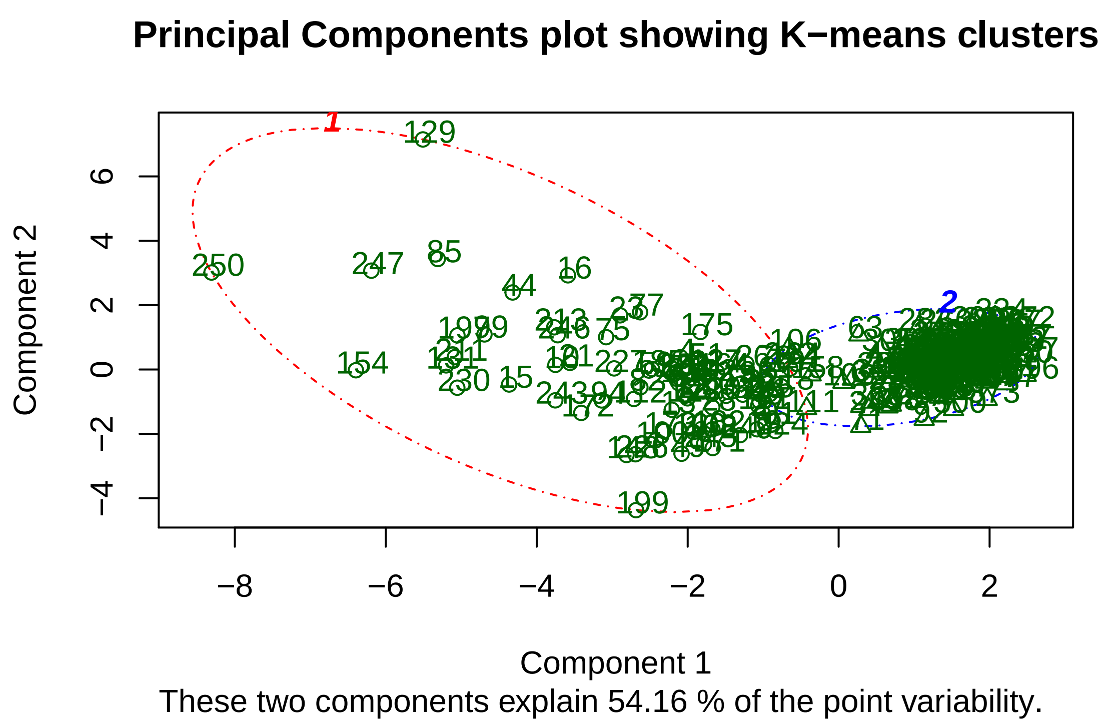


The above two plots also show that SSE - random SSE reaches plateau after two clusters. Thus the cluster level show be two.

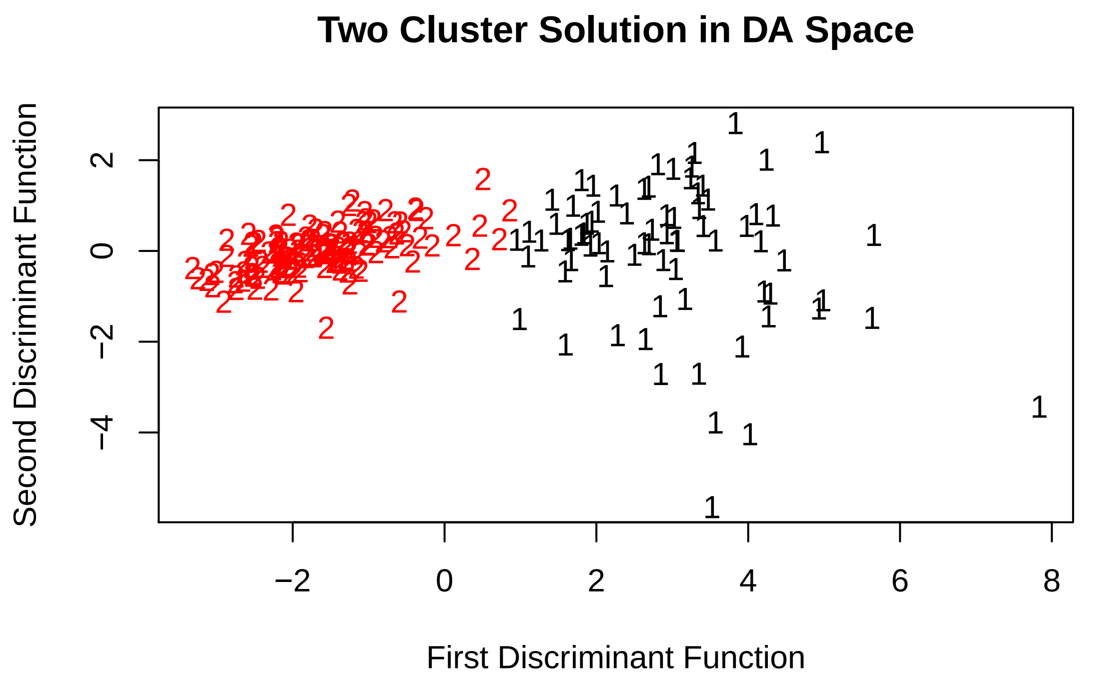
```
clust.level <- 2
# Apply K-means cluster solutions - append clusters to CSV file
fit <- kmeans(kdata, clust.level)
aggregate(kdata, by=list(fit$cluster), FUN=mean)
```

```
##   Group.1        age         bp        bgr         bu         sc
## 1       1  0.4697567  0.4706694  0.6837001  0.8256868  0.8008518
## 2       2 -0.2568454 -0.2573444 -0.3738216 -0.4514546 -0.4378758
##          sod         pot       hemo        pcv       wbcc       rbcc
## 1 -0.7483863  0.16170105 -1.1234635 -1.1210116  0.4490429 -1.0462174
## 2  0.4091896 -0.08841209  0.6142678  0.6129272 -0.2455199  0.5720325
```

```
# Display Principal Components plot of data with clusters identified
clusplot(kdata, fit$cluster, shade=F, labels=2, lines=0, color=T, lty=4, main='Principal Components plot s
```



```
#Make plot of Two cluster solution in space desgignated by first two
#  two discriminant functions
plotcluster(kdata, fit$cluster, main="Two Cluster Solution in DA Space", xlab="First Discriminant Functio
```

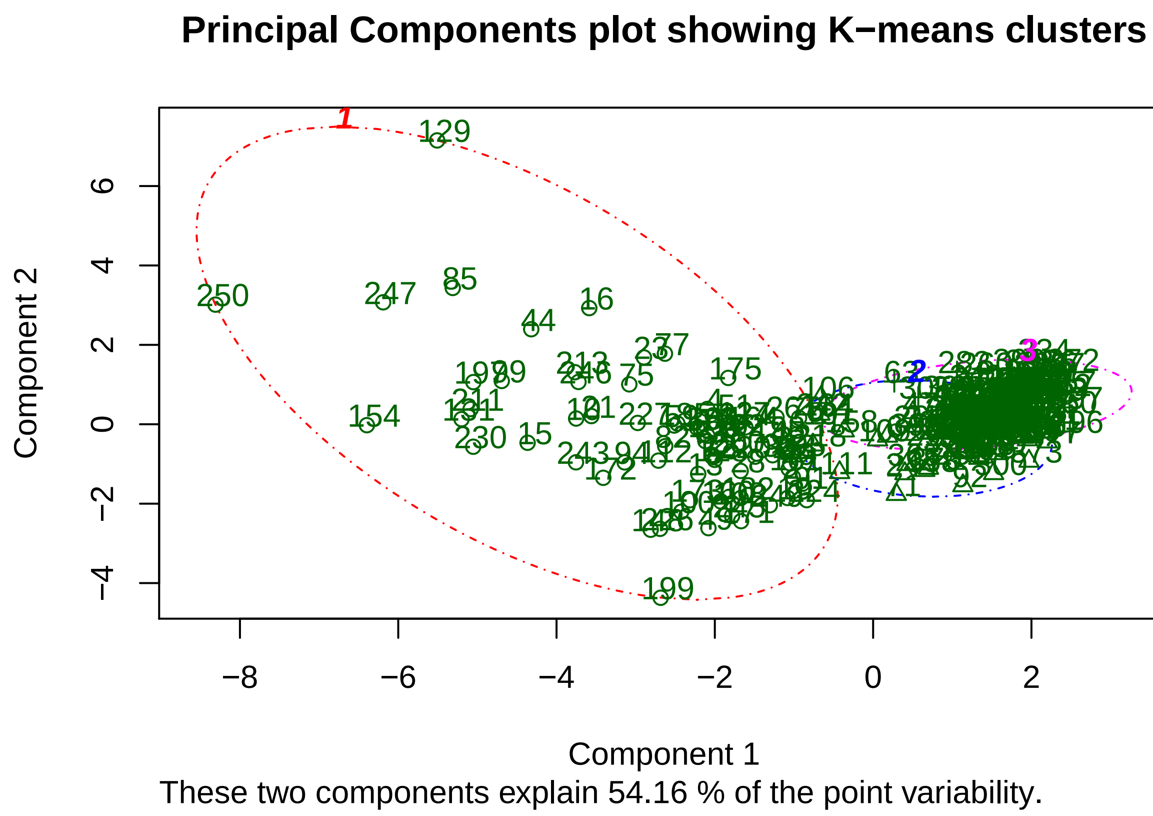


The K-means method shows that there are two clusters for the patients.

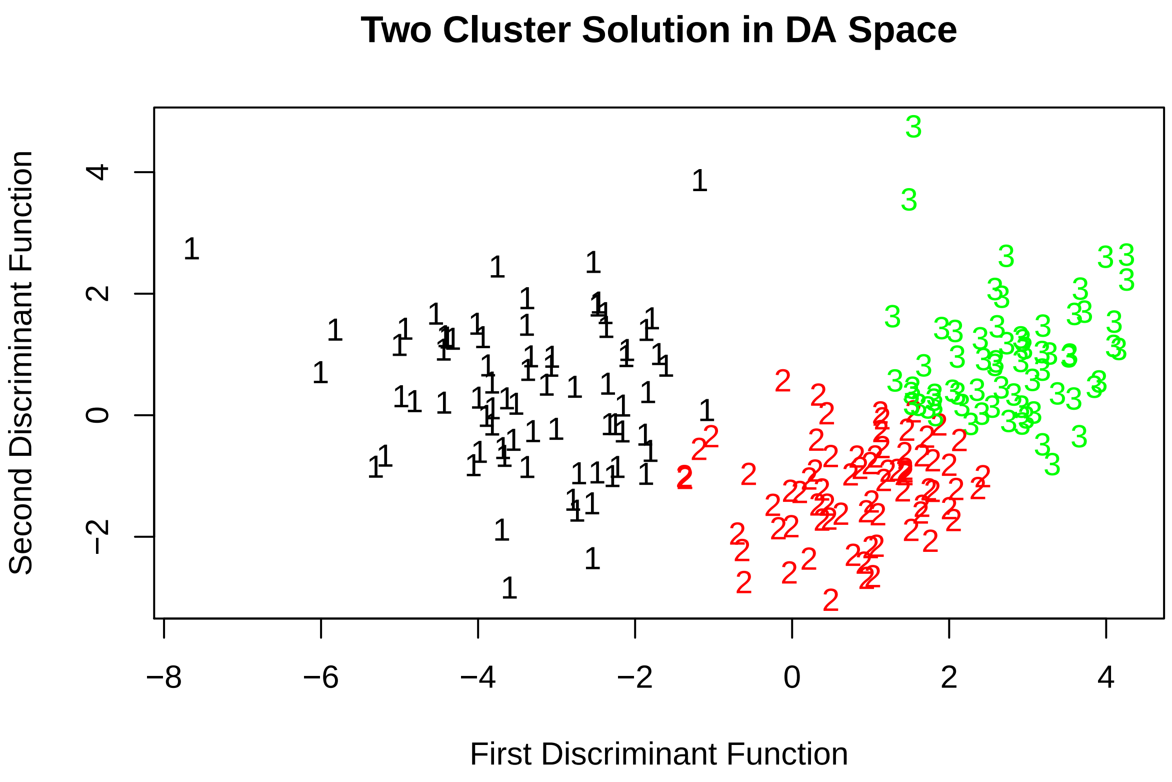
```
 # try cluster size=3
fit <- kmeans(kdata,3)
aggregate(kdata, by=list(fit$cluster), FUN=mean)
```

```
##   Group.1       age        bp       bgr        bu        sc
## 1       1 0.5129327 0.4751391 0.7209267 0.8419373 0.8307283
## 2       2  0.5129327 -0.1382055 -0.2958637 -0.4356381 -0.3852571
## 3       3 -1.1330454 -0.3721356 -0.4694726 -0.4487484 -0.4920129
##          sod         pot       hemo         pcv       wbcc       rbcc
## 1 -0.7654735  0.16530283 -1.1337223 -1.1296491  0.4139846 -1.0693669
## 2  0.4719758 -0.11563597  0.5636216  0.5453174 -0.1309277  0.4791902
## 3  0.3241617 -0.05485594  0.6296635  0.6453813 -0.3126301  0.6518370
```

```
# Display Principal Components plot of data with clusters identified
clusplot(kdata, fit$cluster, shade=F, labels=2, lines=0, color=T, lty=4, main='Principal Components plot s
```



```
#Make plot of Two cluster solution in space desginated by first two
#  two discriminant functions
plotcluster(kdata, fit$cluster, main="Two Cluster Solution in DA Space", xlab="First Discriminant Functio
```



When there are three clusters, the clusterig produced by K means and hierachical clustering are very similar.

5. Comment on the number of groups that seem to be present based on what you find above.