Homework 3

Manikanta Kalyan Gokavarapu

2023-04-24

Loading Packages:

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```
library(datasets)
library("cluster")
```

```
## Warning: package 'cluster' was built under R version 4.2.3
```

```
library("multtest")
```

```
## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min
```

```
## Loading required package: Biobase
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
library("fpc")
```

```
## Warning: package 'fpc' was built under R version 4.2.3
```

```
library("bootcluster")
```

```
## Warning: package 'bootcluster' was built under R version 4.2.3
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
```

```
##   +.gg   ggplot2
```

```
library("fossil")
```

```
## Warning: package 'fossil' was built under R version 4.2.3
```

```
## Loading required package: sp
```

```
## Warning: package 'sp' was built under R version 4.2.3
```

```
## Loading required package: maps
```

```
## Warning: package 'maps' was built under R version 4.2.3
```

```
##
## Attaching package: 'maps'
```

```
## The following object is masked from 'package:cluster':
##
##     votes.repub
```

```
## Loading required package: shapefiles
```

```
## Loading required package: foreign
```

```
##
## Attaching package: 'shapefiles'
```

```
## The following objects are masked from 'package:foreign':
##
##     read.dbf, write.dbf
```

```
library(igraph)
```

```
## Warning: package 'igraph' was built under R version 4.2.3
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:BiocGenerics':
##
##     normalize, path, union
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
library(igraphdata)
```

```
## Warning: package 'igraphdata' was built under R version 4.2.3
```

Question 1:

Suppose that for a particular data set, we perform hierarchical clustering using single linkage and using complete linkage. We obtain two dendrograms.

(a) At a certain point on the single linkage dendrogram, the clusters {1, 2, 3} and {4, 5} fuse. On the complete linkage dendrogram, the clusters {1, 2, 3} and {4, 5} also fuse at a certain point. Which fusion will occur higher on the tree, or will they fuse at the same height, or is there not enough information to tell?

Answer: Given {1,2,3} and {4,5} fuse.

To understand at what height these clusters {1,2,3} and {4,5} merge, we need to consider the dissimilarity matrix to answer the question:

Assume that,

The dissimilarity between {1,4} is 1

The dissimilarity between {1,5} is 2

The dissimilarity between {2,4} is 3

The dissimilarity between {2,5} is 4

The dissimilarity between {3,4} is 5

The dissimilarity between {3,5} is 6

So as we consider minimum height in single linkage, The clusters {1,2,3} and {4,5} merge at a height of 1.

But if the dissimilarity between all the possibilities changes and remains the same value. For example if the dissimilarity between {1,4},{1,5},{2,4},{2,5},{3,4},{3,5} all turn to 3 then the clusters {1,2,3} and {4,5} merge at a height of 3.

So as don't have one specific answer for the given data. We can say that there is not enough information to tell where the clusters fuse

(b) At a certain point on the single linkage dendrogram, the clusters {5} and {6} fuse. On the complete linkage dendrogram, the clusters {5} and {6} also fuse at a certain point. Which fusion will occur higher on the tree, or will they fuse at the same height, or is there not enough information to tell?

Answer: Given {5} and {6} fuse.

To understand at what height these single elements {5} and {6} merge, we need to consider the dissimilarity matrix to answer the question:

Assume that,

The dissimilarity between {5,6} is 4, so cluster {5,6} forms at height of 4 in single linkage.

The dissimilarity between {5,6} is 4, so cluster {5,6} forms at height of 4 in complete linkage.

So, Answer will be they will fuse at same height in both single and complete linkages.

Question 2:

In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

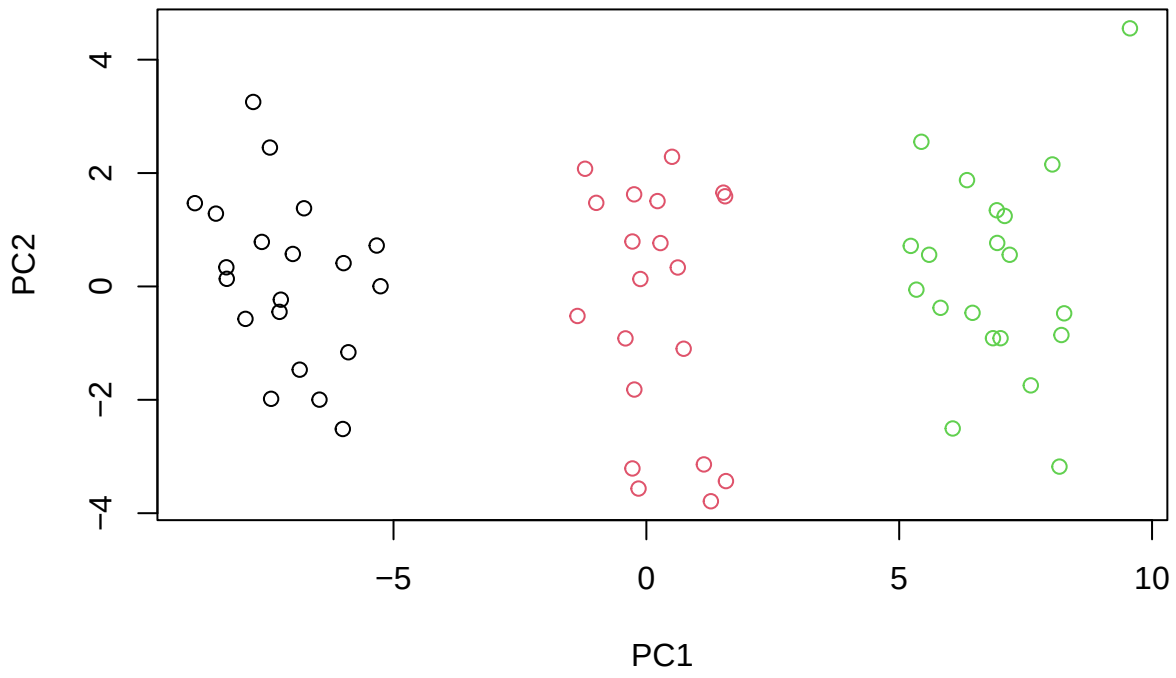
(a) Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables.

Hint: There are a number of functions in R that you can use to generate data. One example is the rnorm() function; runif() is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes. Creating the needed data set with three classes and mean shift in each class.

```
set.seed(123)
data <- matrix(c(rnorm(20 * 50, mean = 1),
                         rnorm(20 * 50, mean = 2),
                         rnorm(20 * 50, mean = 3)), ncol = 50, byrow = TRUE)
true_labels <- unlist(lapply(1:3,function(x){rep(x,20)}))
```

(b) Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component score vectors. Applying the PCA on data and plotting the first two principal components

```
set.seed(123)
fit <- prcomp(data)
PC1 <- fit$x[,1]
PC2 <- fit$x[,2]
plot(PC1, PC2, col=true_labels)
```



From the plot we can say there is good separation between the three classes created.

(c) Perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the trueclass labels?

Hint: You can use the table() function in R to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same Perform k-means cluster with k=3 and plot the table data which true vs cluster labels.

```
set.seed(123)
km <- kmeans(data, 3, nstart = 60)
table(true_labels,km$cluster)
```

```
##
## true_labels  1  2  3
##           1  0 20  0
##           2 20  0  0
##           3  0  0 20
```

Calculate the adjusted Rand Index.

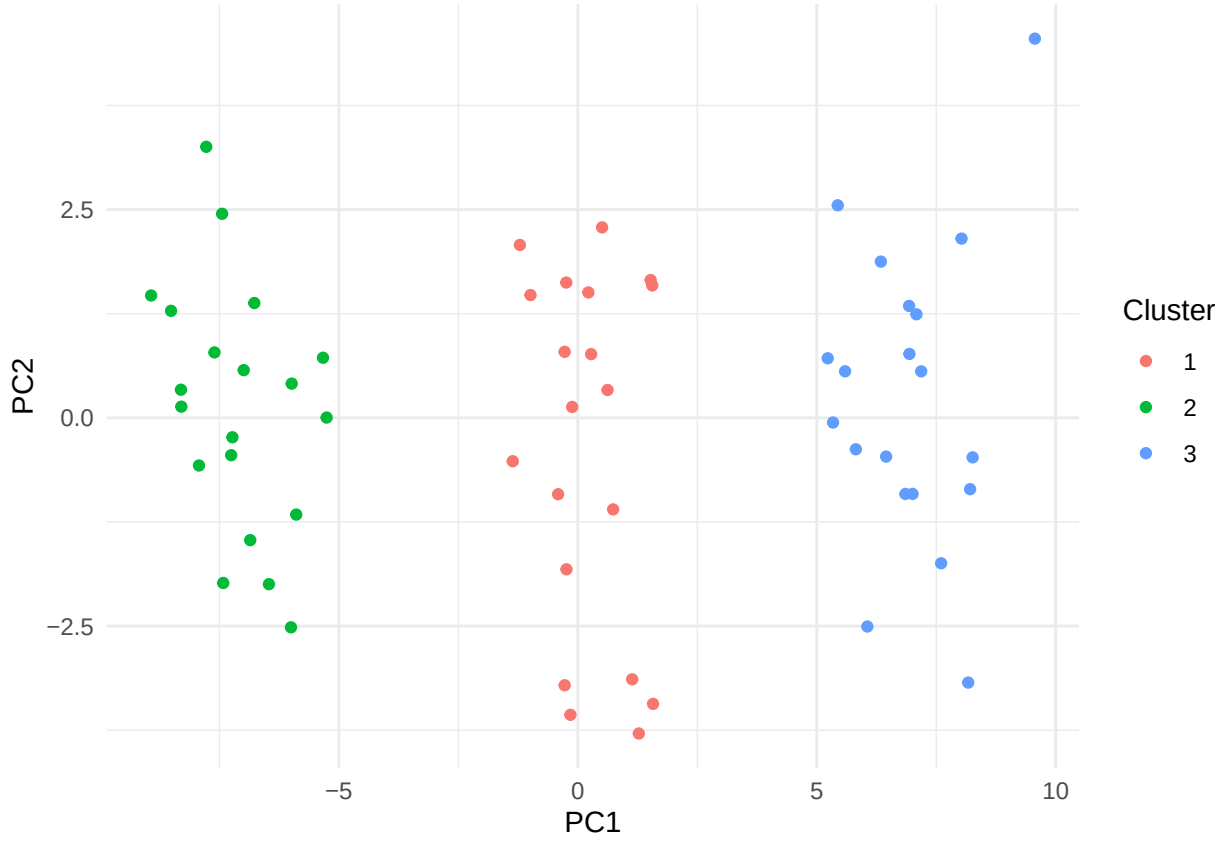
```
adj.rand.index(true_labels,km$cluster)
```

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

Label the data with cluster labels obtained from k-means.

```
# create a data frame with the PC1 and PC2 scores and cluster labels
df <- data.frame(PC1 = PC1, PC2 = PC2, Cluster = as.factor(km$cluster))

# plot the data points colored by the cluster labels
ggplot(df, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point() +
  labels(color = "Cluster") +
  theme_minimal()
```



As adjusted rand index =1, we can say there is a perfect agreement between the true class labels and clustering labels we got from k-means.

From table data we can see that the data is perfectly clustered and each cluster has 20 observations.

you can see k-means plot when k=3 the cluster labels are perfectly assigned to data points.

(d) Perform K-means clustering with K = 2. Describe your results. Applying k-means for k=2 and plotting table data between true labels vs cluster labels

```
set.seed(123)
km2 <- kmeans(data, 2, nstart = 60)
table(true_labels,km2$cluster)
```

```
##
## true_labels  1  2
##           1  0 20
##           2 20  0
##           3 20  0
```

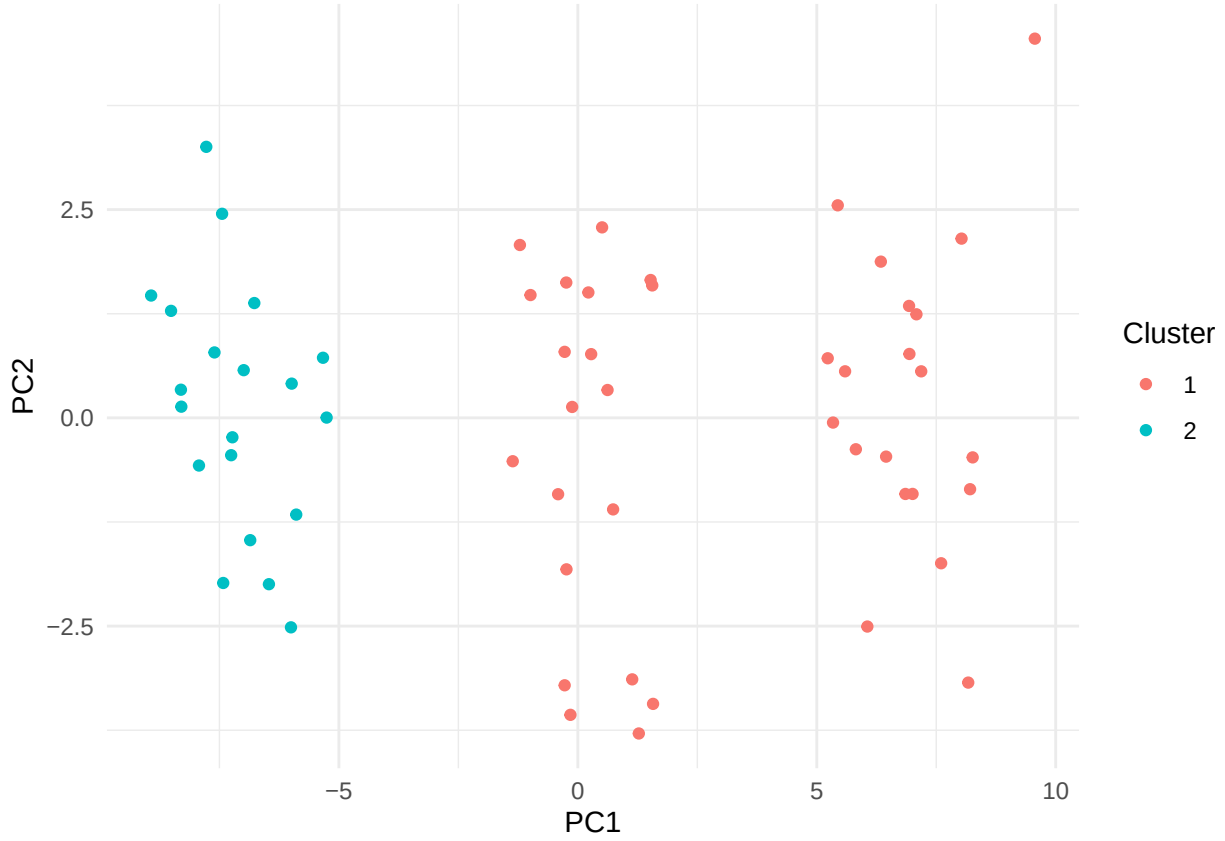
Calculate Adjusted Rand Index.

```
adj.rand.index(true_labels,km2$cluster)
```

```
## [1] 0.562963
```

Label the data with cluster labels obtained from k-means.

```
# create a data frame with the PC1 and PC2 scores and cluster labels
df <- data.frame(PC1 = PC1, PC2 = PC2, Cluster = as.factor(km2$cluster))
# plot the data points colored by the cluster labels
ggplot(df, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point() +
  labs(color = "Cluster") +
  theme_minimal()
```



The adjusted rand index came down to 0.56 when we only take 2 clusters because 20 observation are assigned to one cluster and remaining 40 observations are assigned to other cluster in K-means.

So, there is an increase in no.of observations in a single cluster because of having only two clusters.

You can see result of clustering labels in k-means plot above.

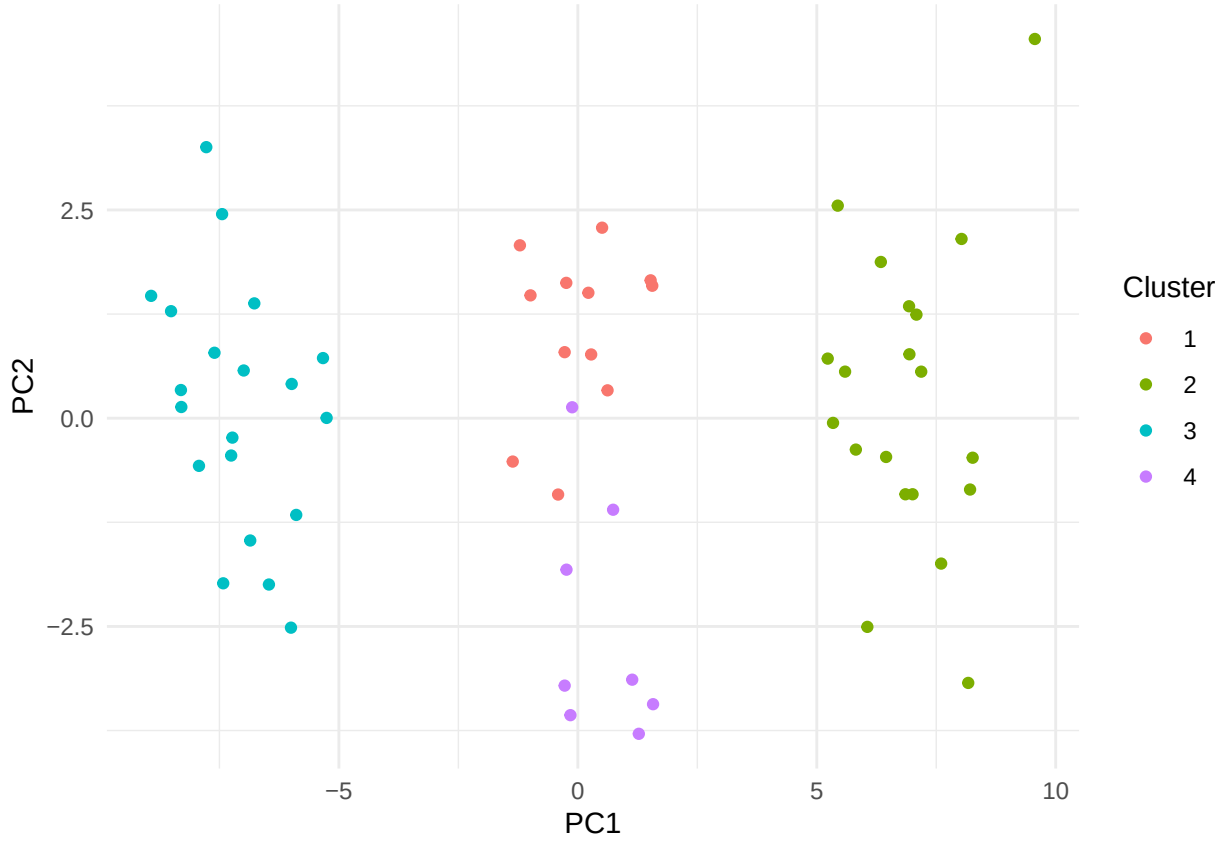
(e) Now perform K-means clustering with K = 4, and describe your results. Applying k-means for k=4 and plotting table data between true labels vs cluster labels

```
set.seed(123)
km3 <- kmeans(data, 4, nstart = 60)
table(true_labels,km3$cluster)
```

```
##
## true_labels  1  2  3  4
##           1  0  0 20  0
##           2 12  0  0  8
##           3  0 20  0  0
```

Label the data with cluster labels obtained from k-means when k=4.

```
# create a data frame with the PC1 and PC2 scores and cluster labels
df <- data.frame(PC1 = PC1, PC2 = PC2, Cluster = as.factor(km3$cluster))
# plot the data points colored by the cluster labels
ggplot(df, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point() +
  labels(color = "Cluster") +
  theme_minimal()
```



From the table output we can observe that class2 is further divided into two clusters 1,4 in K-means.

This is unnecessary clustering of one class which we can observe in the k-means plot above also.

(f) Now perform K-means clustering with K = 3 on the first two principal component score vectors, rather than on the raw data. That is, perform K-means clustering on the 60 × 2 matrix of which the first column is the first principal component score vector, and the second column is the second principal component score vector. Comment on the results. Perform k means with k=3 on the first two principal components.

```
PC_dats <- cbind(PC1, PC2)
set.seed(123)
```

```
km4 <- kmeans(PC_dats, 3, nstart = 60)
table(true_labels,km4$cluster)
```

```
##
## true_labels  1  2  3
##           1  0 20  0
##           2 20  0  0
##           3  0  0 20
```

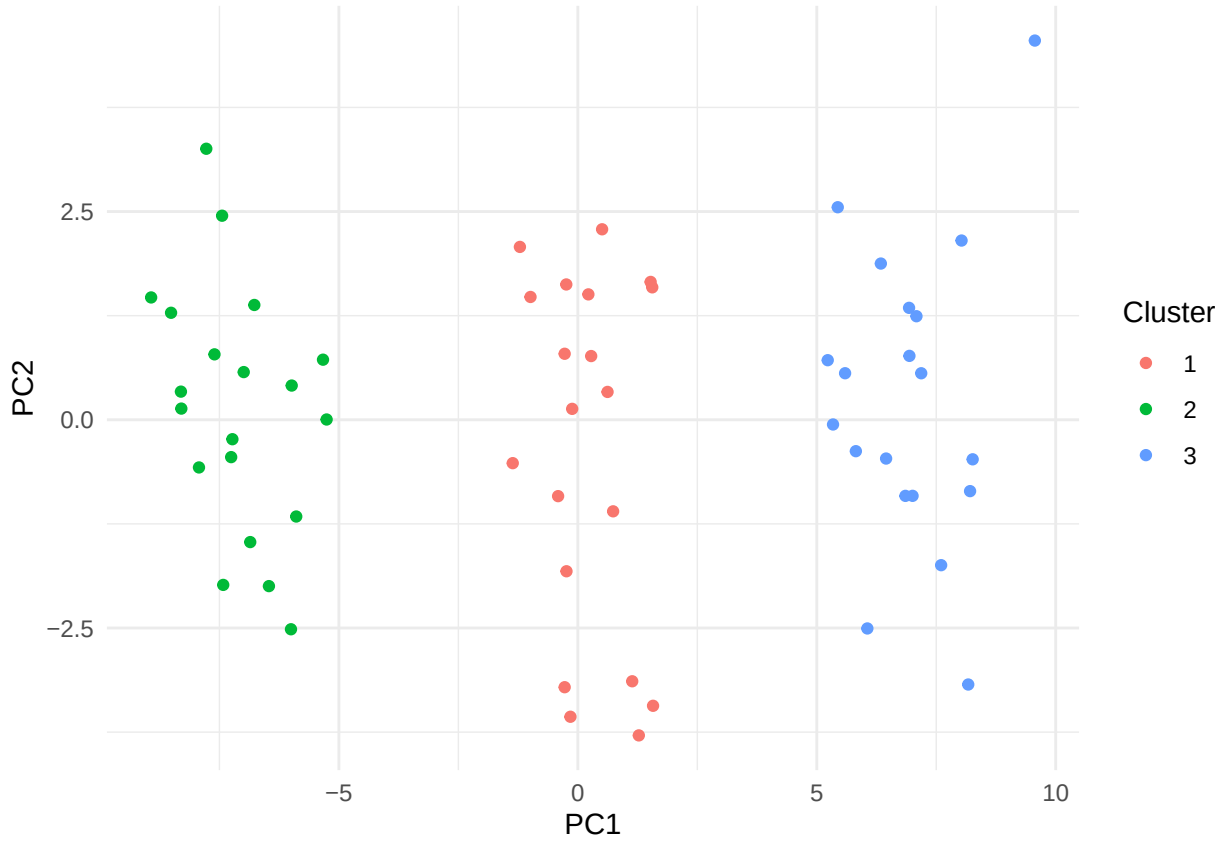
Calculate adjusted Rand Index.

```
adj.rand.index(true_labels,km4$cluster)
```

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

Label the data with cluster labels obtained from k-means when k=3 on PC data.

```
# create a data frame with the PC1 and PC2 scores and cluster labels
df <- data.frame(PC1 = PC1, PC2 = PC2, Cluster = as.factor(km4$cluster))
# plot the data points colored by the cluster labels
ggplot(df, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point() +
  labels(color = "Cluster") +
  theme_minimal()
```



We can see a perfect clustering as same as in c which is expected and the rand index =1

And from table data we can see each cluster has 20 observations.

(g) Using the scale() function, perform K-means clustering with K = 3 on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (b)? Explain. Perform K-means with k=3 on scaled data which will have SD of 1.

```
set.seed(123)
scaled_data <- scale(data)
km5 <- kmeans(scaled_data, 3, nstart = 60)
table(true_labels,km5$cluster)
```

```
##
## true_labels  1  2  3
##           1  0 20  0
##           2 20  0  0
##           3  0  0 20
```

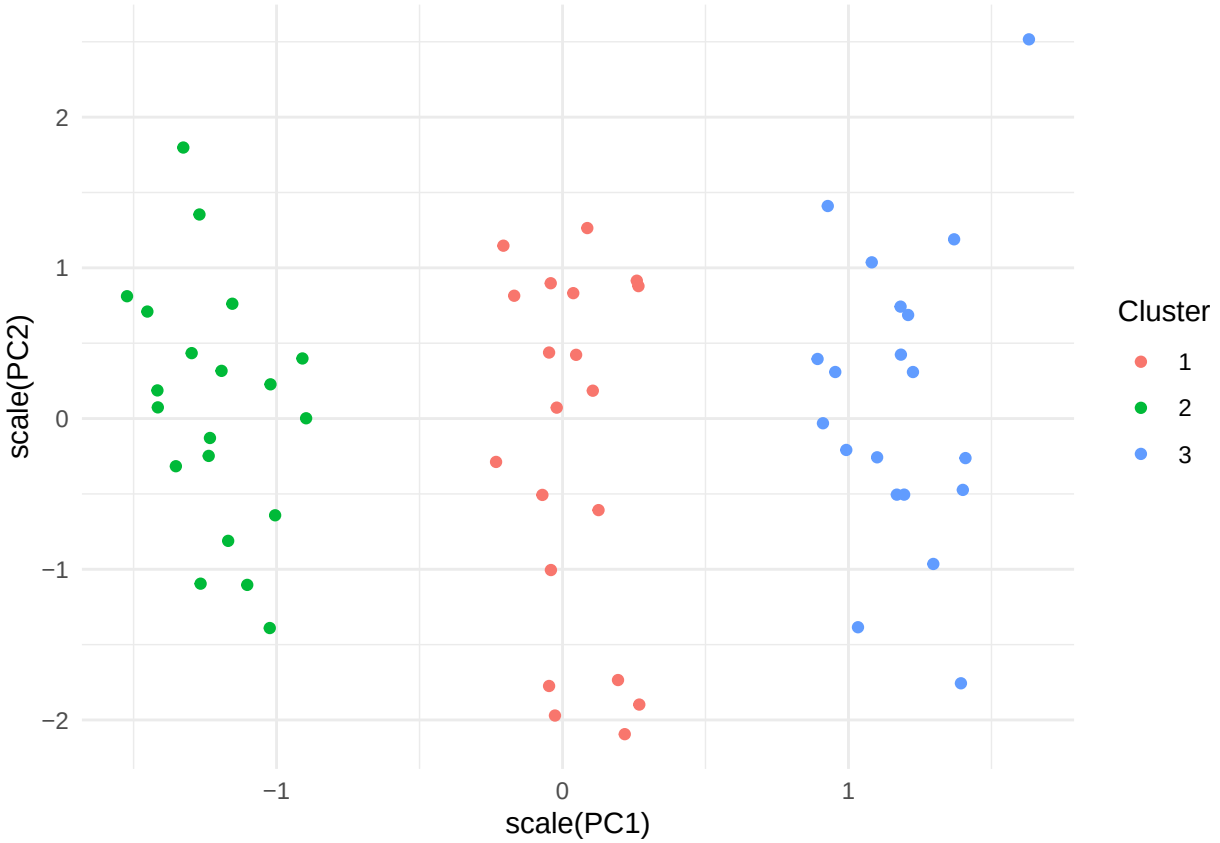
Calculate the Adjusted Rand Index.

```
adj.rand.index(true_labels,km5$cluster)
```

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

Label the data with cluster labels obtained from k-means when k=3 on Scaled data.

```
# create a data frame with the PC1 and PC2 scores and cluster labels
df <- data.frame(PC1 = scale(PC1), PC2 = scale(PC2), Cluster = as.factor(km5$cluster))
# plot the data points colored by the cluster labels when k=3 on scaled data.
ggplot(df, aes(x = scale(PC1), y = scale(PC2), color = Cluster)) +
  geom_point() +
  labs(color = "Cluster") +
  theme_minimal()
```



As the data is well separated into three classes we can still see that the data is perfectly clustered on the scaled data as well.

And the adjusted rand index is 1 and also we have 20 observations per clusters for 3 clusters which implying perfect clustering.

In b and g in both places we could see data is well separated into three classes and have perfect clustering. but as we scaled the data the axis scale changes that all.

Question 3:

On the book website, www.statlearning.com, there is a gene expression data set(Ch12Ex13.csv) that consists of 40 tissue samples with measurements on 1,000 genes. The first 20 samples are from healthy patients, while the second 20 are from a diseased group.

(a) Load in the data using read.csv(). You will need to select header = F. Loading the genedata with header false.

```
genedata <- read.csv("Ch12Ex13.csv",header = FALSE)
head(genedata)
```

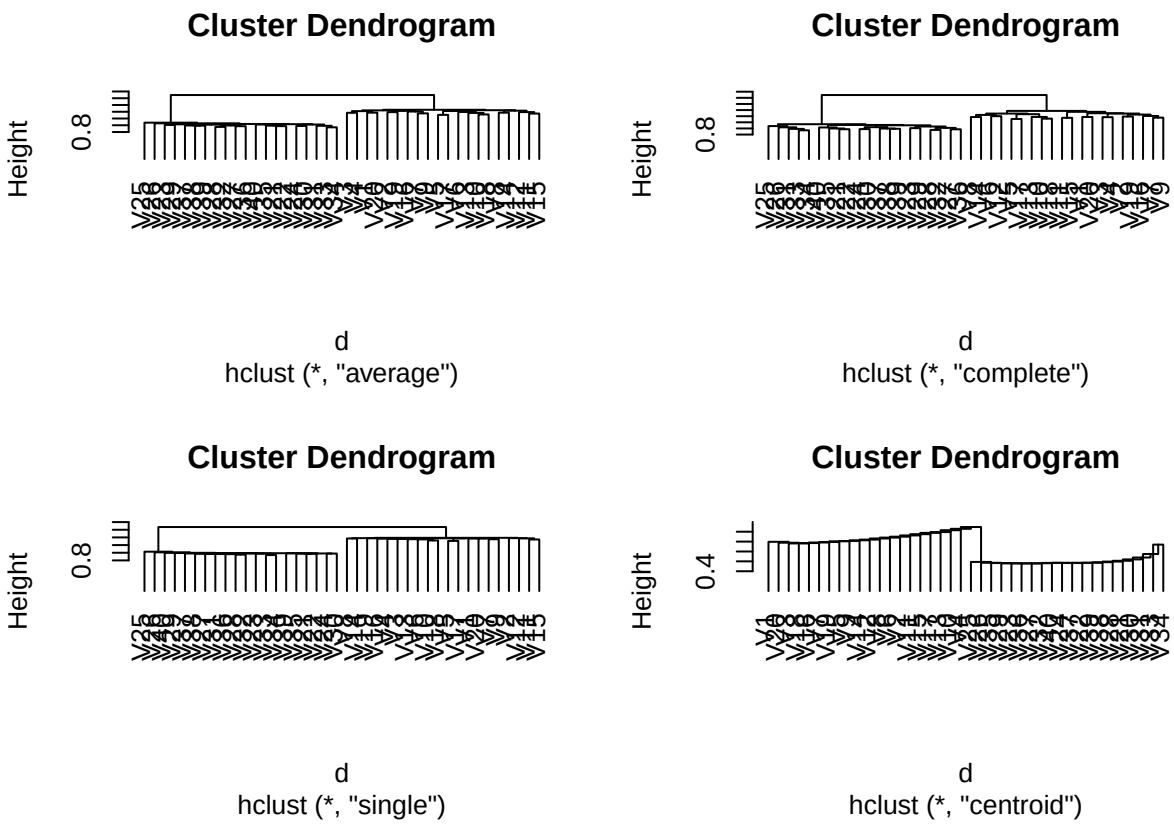
```
##            V1         V2         V3         V4         V5         V6
## 1 -0.96193340  0.4418028 -0.9750051  1.4175040  0.8188148  0.3162937
## 2 -0.29252570 -1.1392670  0.1958370 -1.2811210 -0.2514393  2.5119970
## 3  0.25878820 -0.9728448  0.5884858 -0.8002581 -1.8203980 -2.0589240
## 4 -1.15213200 -2.2131680 -0.8615249  0.6309253  0.9517719 -1.1657240
## 5  0.19578280  0.5933059  0.2829921  0.2471472  1.9786680 -0.8710180
## 6  0.03012394 -0.6910143 -0.4034258 -0.7298590 -0.3640986  1.1253490
##            V7          V8          V9        V10        V11        V12
## 1 -0.02496682 -0.06396600  0.03149702 -0.3503106 -0.7227299 -0.2819547
## 2 -0.92220620  0.05954277 -1.40964500 -0.6567122 -0.1157652  0.8259783
## 3 -0.06476437  1.59212400 -0.17311700 -0.1210874 -0.1875790 -1.5001630
## 4 -0.39155860  1.06361900 -0.35000900 -1.4890580 -0.2432189 -0.4330340
## 5 -0.98971500 -1.03225300 -1.10965400 -0.3851423  1.6509570 -1.7449090
## 6 -1.40404100 -0.80613040 -1.23792400  0.5776018 -0.2720642  2.1765620
##           V13         V14        V15        V16        V17        V18
## 1  1.33751500  0.70197980  1.0076160 -0.4653828  0.6385951  0.2867807
## 2  0.34644960 -0.56954860 -0.1315365  0.6902290 -0.9090382  1.3026420
## 3 -1.22873700  0.85598900  1.2498550 -0.8980815  0.8702058 -0.2252529
## 4 -0.03879128 -0.05789677 -1.3977620 -0.1561871 -2.7359820  0.7756169
## 5 -0.37888530 -0.67982610 -2.1315840 -0.2301718  0.4661243 -1.8004490
## 6  1.43640700 -1.02578100  0.2981582 -0.5559659  0.2046529 -1.1916480
##          V19         V20        V21        V22        V23        V24
## 1 -0.2270782 -0.22004520 -1.2425730 -0.1085056 -1.8642620 -0.5005122
## 2 -1.6726950 -0.52550400  0.7979700 -0.6897930  0.8995305  0.4285812
## 3  0.4502892  0.55144040  0.1462943  0.1297400  1.3042290 -1.6619080
## 4  0.6141562  2.01919400  1.0811390 -1.0766180 -0.2434181  0.5134822
## 5  0.6262904 -0.09772305 -0.2997108 -0.5295591 -2.0235670 -0.5108402
## 6  0.2350916  0.67096470  0.1307988  1.0689940  1.2309870  1.1344690
##           V25         V26        V27        V28         V29         V30
## 1 -1.32500800  1.06341100 -0.2963712 -0.1216457  0.08516605  0.62417640
## 2 -0.67611410 -0.53409490 -1.7325070 -1.6034470 -1.08362000  0.03342185
## 3 -1.63037600 -0.07742528  1.3061820  0.7926002  1.55946500 -0.68851160
## 4 -0.51285780  2.55167600 -2.3143010 -1.2764700 -1.22927100  1.43439600
## 5  0.04600274  1.26803000 -0.7439868  0.2231319  0.85846280  0.27472610
## 6  0.55636800 -0.35876640  1.0798650 -0.2064905 -0.00616453  0.16425470
##          V31          V32         V33        V34        V35        V36
## 1 -0.5095915 -0.216725500 -0.05550597 -0.4844491 -0.5215811  1.9491350
## 2  1.7007080  0.007289556  0.09906234  0.5638533 -0.2572752 -0.5817805
## 3 -0.6154720  0.009999363  0.94581000 -0.3185212 -0.1178895  0.6213662
## 4 -0.2842774  0.198945600 -0.09183320  0.3496279 -0.2989097  1.5136960
## 5 -0.6929984 -0.845707200 -0.17749680 -0.1664908  1.4831550 -1.6879460
## 6  1.1567370  0.241774500  0.08863952  0.1829540  0.9426771 -0.2096004
##           V37        V38         V39        V40
```

```
## 1  1.32433500  0.4681471  1.06110000  1.6559700
## 2 -0.16988710 -0.5423036  0.31293890 -1.2843770
## 3 -0.07076396  0.4016818 -0.01622713 -0.5265532
## 4  0.67118470  0.0108553 -1.04368900  1.6252750
## 5 -0.14142960  0.2007785 -0.67594210  2.2206110
## 6  0.53626210 -1.1852260 -0.42274760  0.6243603
```

(b) Apply hierarchical clustering to the samples using correlation based distance, and plot the dendrogram. Do the genes separate the samples into the two groups? Applying hierarchical clustering on correlation based distance matrix.

```
#create correlation based distance matrix
d <- dist(cor(genedata))

#apply hierarchical clustering on the distance matrix with different type of linkages.
#x11()
par(mfrow=c(2,2))
hc <- hclust(d, method = "ave")
plot(hc,hang=-1)
hc1 <- hclust(d, method = "complete")
plot(hc1,hang=-1)
hc2 <- hclust(d, method = "single")
plot(hc2,hang=-1)
hc3 <- hclust(d, method = "centroid")
plot(hc3,hang=-1)
```



Plot the cutree two understand the groupings in clusterings.

```
ct <- cutree(hc, k = 2)
ct
```

```
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
ct1 <- cutree(hc1, k = 2)
ct1
```

```
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
ct2 <- cutree(hc2, k = 2)
ct2
```

```
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
ct3<- cutree(hc3, k = 2)
ct3
```

```
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
##   1   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
## V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38 V39 V40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

From the dendrogram plots we can see that genes separate the samples into two distinct groups .

From the cutree results we can see that the first 20 tissue samples are assigned one group and second 20 tissue samples assigned into another group.

In most of the linkages the 2 groups are evident but in some types of linkages like centroid they are not that clear and also the dendrograms of all the linkages are not that similar. so we can say that the results depend on linkage used.

(c) Your collaborator wants to know which genes differ the most across the two groups. Suggest a way to answer this question, and apply it here. Approach:

To know which genes differ the most across two groups first we need to perform PCA on the gene data.

Then we perform k-means clustering on the first two principal components and check if we have good separation between the data points.

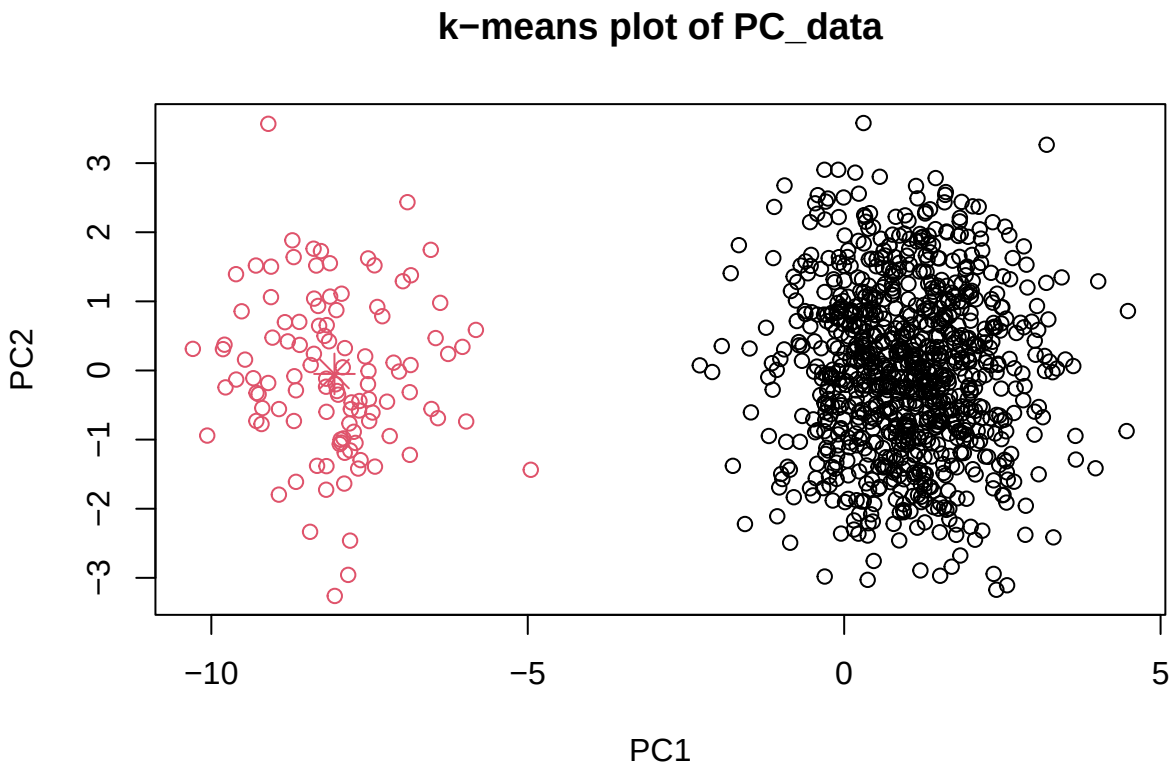
If there is good separation between the groups, we need to check which genes differ the most across the two groups. To do so we need to get each gene's weight, In order to do that we will look at the absolute values of the total loadings for each gene.

Applying K-means on PCdata of genes.

```
set.seed(123)
fit1 <- prcomp(genedata)
PC1 <- fit1$x[,1]
PC2 <- fit1$x[,2]
```

```r
# run k-means on the PC values
PC_data <- cbind(PC1, PC2)
k_means <- kmeans(PC_data, centers = 2)
```

```r
# plot the groups
#x11()
plot(PC_data, col = k_means$cluster, main = "k-means plot of PC_data")
points(k_means$centers, col = 1:2, pch = 8, cex= 2)
```



From the k-means plot above we can say there is a good separation between gene data.

Below Performing PCA on transpose of gene expression data and plotting summary.

```r
pca_transpose_data <- prcomp(t(genedata))
summary(pca_transpose_data)
```

```
## Importance of components:
##                            PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     11.9409 6.06818 5.93476 5.83115 5.75209 5.70031 5.63448
## Proportion of Variance  0.1267 0.03271 0.03129 0.03021 0.02939 0.02887 0.02821
## Cumulative Proportion   0.1267 0.15939 0.19068 0.22089 0.25029 0.27915 0.30736
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     5.57726 5.54943 5.50625 5.48852 5.46025 5.40230 5.33441
## Proportion of Variance 0.02764 0.02736 0.02694 0.02676 0.02649 0.02593 0.02528
## Cumulative Proportion  0.33499 0.36236 0.38929 0.41605 0.44254 0.46847 0.49375
##                           PC15    PC16    PC17    PC18    PC19    PC20    PC21
## Standard deviation     5.27756 5.21594 5.20000 5.15140 5.11600 5.05591 5.03836
## Proportion of Variance 0.02475 0.02417 0.02402 0.02358 0.02325 0.02271 0.02255
```

```
## Cumulative Proportion  0.51850 0.54267 0.56669 0.59027 0.61352 0.63623 0.65878
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     5.01868 4.95965 4.91393 4.86397 4.81796 4.80811 4.73485
## Proportion of Variance 0.02238 0.02185 0.02145 0.02102 0.02062 0.02054 0.01992
## Cumulative Proportion  0.68116 0.70301 0.72447 0.74548 0.76611 0.78665 0.80656
##                           PC29    PC30    PC31    PC32    PC33    PC34    PC35
## Standard deviation     4.70098 4.65564 4.61621 4.56733 4.53032 4.49528 4.36502
## Proportion of Variance 0.01963 0.01926 0.01893 0.01853 0.01823 0.01795 0.01693
## Cumulative Proportion  0.82620 0.84545 0.86439 0.88292 0.90115 0.91910 0.93603
##                           PC36    PC37    PC38    PC39      PC40
## Standard deviation     4.35858 4.26700 4.20277 4.13922 5.251e-15
## Proportion of Variance 0.01688 0.01618 0.01569 0.01522 0.000e+00
## Cumulative Proportion  0.95291 0.96909 0.98478 1.00000 1.000e+00
```

We will identify the top 15 genes with the highest contribution to the first principal component.

```
total_load <- apply(pca_transpose_data$rotation, 1, sum)
index = order(abs(total_load), decreasing = TRUE)
index[1:15]
```

```
##  [1] 865  68 911 428 624  11 524 803 980 822 529 765 801 771 570
```

Above are the 15 genes that differ the most across the two groups i.e. healthy and diseased patients.

Question 4:

Consider the two networks "karate" and "kite", which are available in the package "igraphdata".

library(igraphdata)

data(karate)

karate

data(kite)

?kite

```
#Load the needed data.
data(karate)
data(kite)
```

(a) Focus on the karate network. Create noisy datasets. Do this by deleting 5% of the edges randomly (track which ones they are). Perform MCMC for a random graph model (as in Clauset et al.) on this data followed by link-prediction. Are you able to predict the edges that you deleted? Deleted 5% of edges randomly and created noisy karate data set.

```
set.seed(123)
deleted_edges <- sample(E(karate), size = 0.05 * ecount(karate))
#deleted_edges
noisy_karate <- delete.edges(karate, deleted_edges)

#sanity check
ecount(karate)
```

```
## [1] 78
```

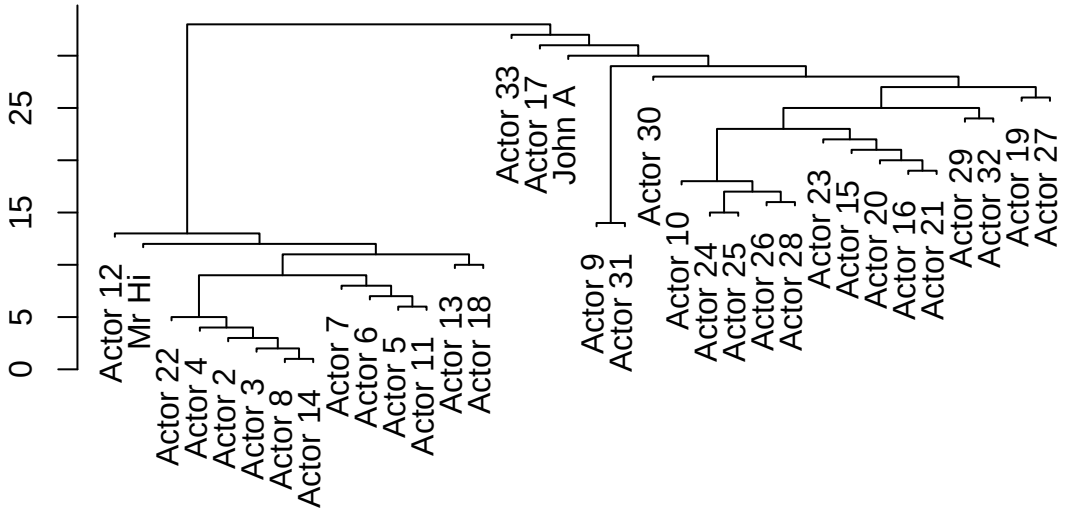
```
ecount(noisy_karate)
```

```
## [1] 75
```

performing MCMC using hierarchical random graph model on noisy_karate data and plotting dendrogram.

```
set.seed(123)
mcmc_noisy_karate <- fit_hrg(noisy_karate)

#Plotting dendrogram
set.seed(123)
plot_dendrogram(mcmc_noisy_karate)
```




```
#Predict missing edges of noisy_karate data
set.seed(123)
pred_edges <- predict_edges(noisy_karate)
#print(pred_edges)

#List predicted edges that are same as deleted edges.
selected_pred_edges <- pred_edges$edges[c(6, 72, 124), ]
selected_pred_edges
```

```
##      [,1] [,2]
## [1,]   19   33
## [2,]    1   20
## [3,]    3   29
```

```
deleted_edges
```

```
## + 3/78 edges from 4b458a1 (vertex names):
## [1] Actor 3 --Actor 29 Actor 19--Actor 33 Mr Hi   --Actor 20
```

observation: In the above output, we see the predicted and deleted edges, from that we can say that all the deleted edges are predicted. But based on probability in top 10 of predicted edges we have only one deleted edge.

```
data(kite)
#Deleted 5% of edges randomly and created noisy kite data set.
set.seed(123)
deleted_edges1 <- sample(E(kite), size = 0.05 * ecount(kite) + 1)
```

```
#deleted_edges
noisy_kite <- delete.edges(kite, deleted_edges1)
#sanity check
ecount(kite)
```

(b) Focus on the yeast network (or kite network if yeast is too big). Create noisy datasets. Do this by deleting 5% of the edges randomly (track which ones they are). Perform MCMC on this data followed by link-prediction. Are you able to predict the edges that you deleted at random well?

```
## [1] 18
```

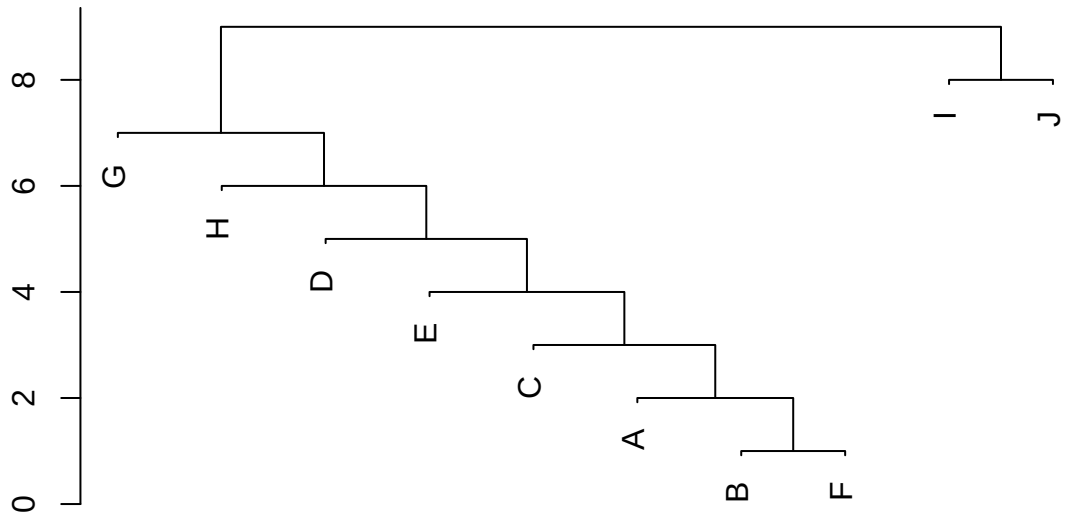
```
ecount(noisy_kite)
```

```
## [1] 17
```

performing MCMC using hierarchical random graph model on noisy_karate data and plotting dendrogram.

```
set.seed(123)
mcmc_noisy_kite <- fit_hrg(noisy_kite)

#Plotting dendrogram
set.seed(123)
plot_dendrogram(mcmc_noisy_kite)
```




```
#Predict missing edges of noisy_karate data
set.seed(123)
pred_edges1 <- predict_edges(noisy_kite)
#print(pred_edges1)
```

```
#List predicted edges that are same as deleted edges.
selected_pred_edges1 <- pred_edges1$edges[c(16), ]
selected_pred_edges1
```

```
## [1] 6 8
```

```
deleted_edges1
```

```
## + 1/18 edge from 6b7ddad (vertex names):
## [1] F--H
```

observation:

In the above output, we see the predicted and deleted edges, from that we can say that all the deleted edges are predicted. But based on probability in top 10 of predicted edges, we have no deleted edges. (5% of kite data will result in number of deleted edges = 0.8 = 0. so I have taken one edge and done the analysis)

```
#Delete 10% of edges randomly and create noisy karate data set.
set.seed(123)
deleted_edges3 <- sample(E(karate), size = 0.1 * ecount(karate))
#deleted_edges
noisy_karate_10 <- delete.edges(karate, deleted_edges3)

#sanity check
ecount(karate)
```

(c) Repeat the exercise in part (a) and (b) after deleting 10%, and 20% of the edges. Comment on your findings.

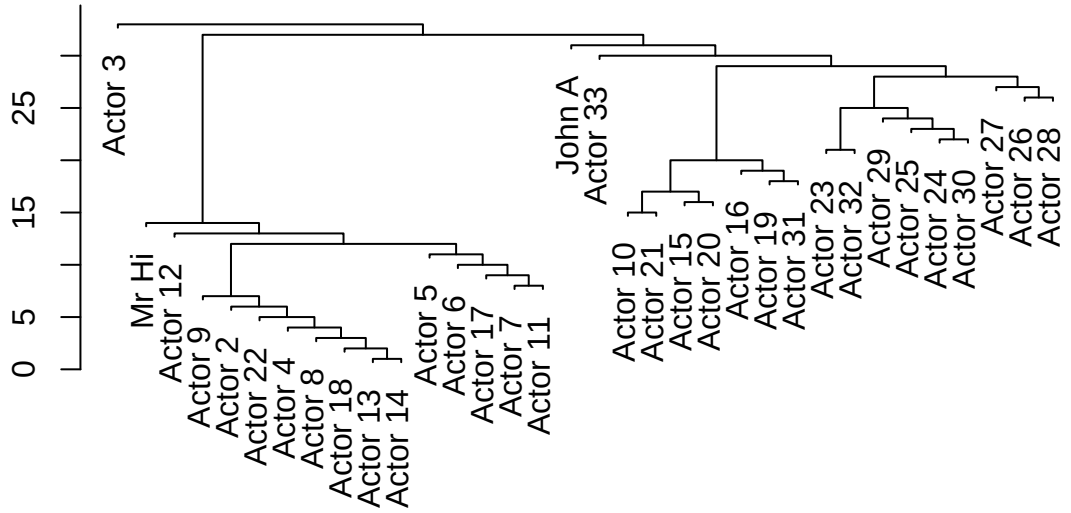
```
## [1] 78
```

```
ecount(noisy_karate_10)
```

```
## [1] 71
```

Performing MCMC using hierarchical random graph model on noisy_karate data and Plotting dendrogram.

```
set.seed(123)
mcmc_noisy_karate_10 <- fit_hrg(noisy_karate_10)
set.seed(123)
plot_dendrogram(mcmc_noisy_karate_10)
```



predicted vs deleted edges.

```
#Predict missing edges of noisy_karate data
set.seed(123)
pred_edges2 <- predict_edges(noisy_karate_10)
#pred_edges2

#List predicted edges that are same as deleted edges.
deleted_edges3
```

```
## + 7/78 edges from 4b458a1 (vertex names):
## [1] Actor 3 --Actor 29 Actor 19--Actor 33 Mr Hi   --Actor 20 Actor 27--Actor 30
## [5] Actor 9 --Actor 31 Actor 16--John A   Actor 9 --Actor 33
```

```
selected_pred_edges2 <- pred_edges2$edges[c(100,10,62,427,393,1,31), ]
selected_pred_edges2
```

```
##      [,1] [,2]
## [1,]    3   29
## [2,]   19   33
## [3,]    1   20
## [4,]   27   30
## [5,]    9   31
## [6,]   16   34
## [7,]    9   33
```

```
#Delete 20% of edges randomly and create noisy karate data set.
set.seed(123)
```

```
deleted_edges4 <- sample(E(karate), size = 0.2 * ecount(karate))
#deleted_edges
noisy_karate_20 <- delete.edges(karate, deleted_edges4)
#sanity check
ecount(karate)
```

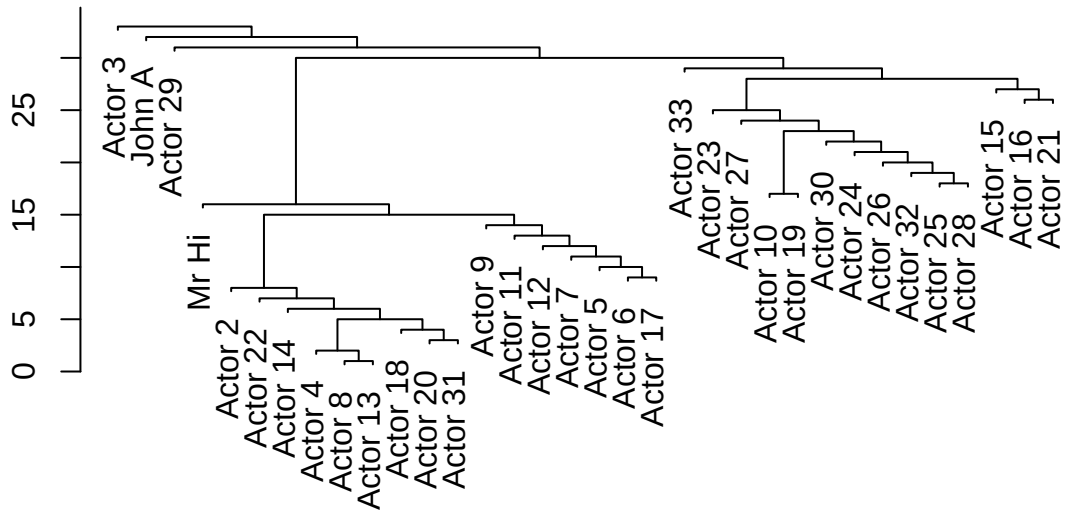
```
## [1] 78
```

```
ecount(noisy_karate_20)
```

```
## [1] 63
```

Performing MCMC using hierarchical random graph model on noisy_karate data Plotting dendrogram.

```
set.seed(123)
mcmc_noisy_karate_20 <- fit_hrg(noisy_karate_20)
set.seed(123)
plot_dendrogram(mcmc_noisy_karate_20)
```



predicted edges vs deleted edges:

```
#Predict missing edges of noisy_karate data
set.seed(123)
pred_edges3 <- predict_edges(noisy_karate_20)
#pred_edges3
#List predicted edges that are same as deleted edges.
deleted_edges4
```

```
## + 15/78 edges from 4b458a1 (vertex names):
```

```
##  [1] Actor 3 --Actor 29 Actor 19--Actor 33 Mr Hi   --Actor 20 Actor 27--Actor 30
##  [5] Actor 9 --Actor 31 Actor 16--John A   Actor 9 --Actor 33 Actor 32--Actor 33
##  [9] Actor 3 --Actor 4  Actor 28--John A   Actor 23--John A   Mr Hi   --Actor 11
## [13] Actor 3 --Actor 8  Mr Hi   --Actor 8  Actor 31--Actor 33
```

```
selected_pred_edges3 <- pred_edges3$edges[c(50,73,14,298,313,1,71,63,40,10,8,12,41,11,93), ]
selected_pred_edges3
```

```
##       [,1] [,2]
##  [1,]    3   29
##  [2,]   19   33
##  [3,]    1   20
##  [4,]   27   30
##  [5,]    9   31
##  [6,]   16   34
##  [7,]    9   33
##  [8,]   32   33
##  [9,]    3    4
## [10,]   28   34
## [11,]   22   34
## [12,]    1   11
## [13,]    3    8
## [14,]    1    8
## [15,]   31   33
```

```
#Delete 10% of edges randomly and create noisy kite data set.
set.seed(123)
deleted_edges5 <- sample(E(kite), size = 0.1 * ecount(kite))
#deleted_edges
noisy_kite_10 <- delete.edges(kite, deleted_edges5)
#sanity check
ecount(kite)
```

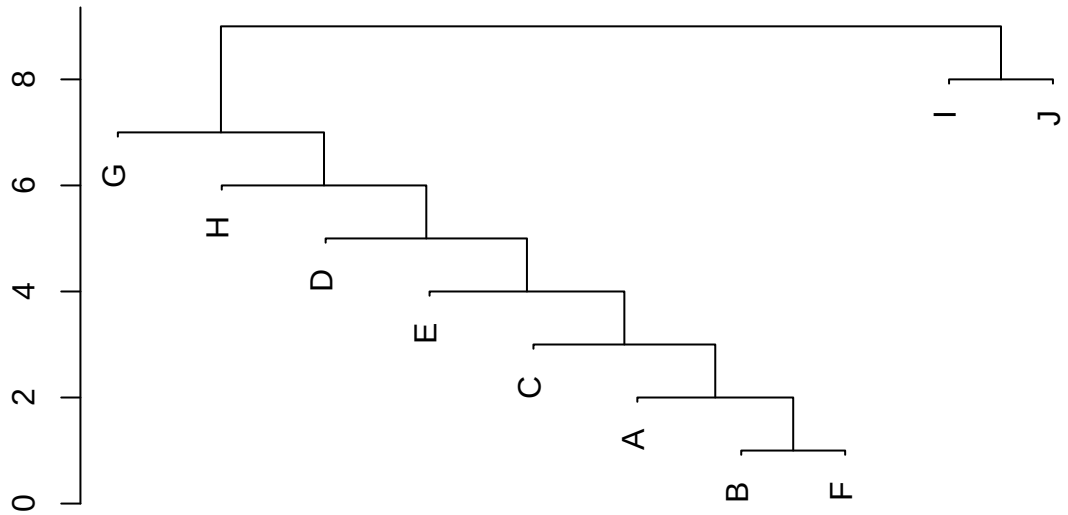
```
## [1] 18
```

```
ecount(noisy_kite_10)
```

```
## [1] 17
```

Performing MCMC using hierarchical random graph model on noisy_kite data and plotting dendrogram.

```
set.seed(123)
mcmc_noisy_kite_10 <- fit_hrg(noisy_kite_10)
set.seed(123)
plot_dendrogram(mcmc_noisy_kite_10)
```



Predicted edges vs deleted edges.

```
#Predict missing edges of noisy_kite data
set.seed(123)
pred_edges4 <- predict_edges(noisy_kite_10)
#pred_edges4
#List predicted edges that are same as deleted edges.
deleted_edges5
```

```
## + 1/18 edge from 6b7ddad (vertex names):
## [1] F--H
```

```
selected_pred_edges4 <- pred_edges4$edges[c(16), ]
selected_pred_edges4
```

```
## [1] 6 8
```

```
#Delete 20% of edges randomly and create noisy kite data set.
set.seed(123)
deleted_edges6 <- sample(E(kite), size = 0.2 * ecount(kite))
#deleted_edges
noisy_kite_20 <- delete.edges(kite, deleted_edges6)
#sanity check
ecount(kite)
```

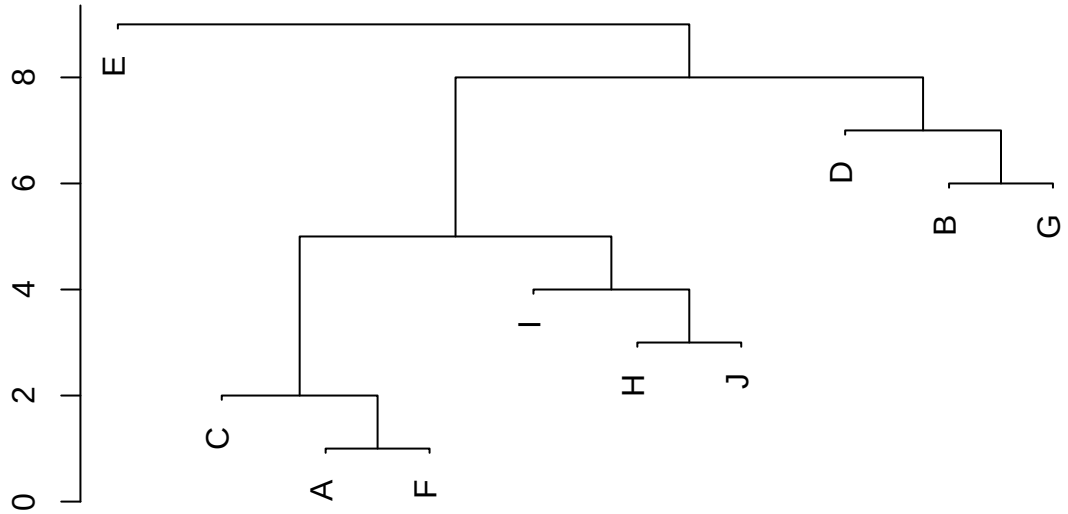
```
## [1] 18
```

```
ecount(noisy_kite_20)
```

```
## [1] 15
```

Performing MCMC using hierarchical random graph model on noisy_kite data and plotting dendrogram.

```
set.seed(123)
mcmc_noisy_kite_20 <- fit_hrg(noisy_kite_20)
set.seed(123)
plot_dendrogram(mcmc_noisy_kite_20)
```



Predicted vs Deleted Edges.

```
#Predict missing edges of noisy_kite data
set.seed(123)
pred_edges5 <- predict_edges(noisy_kite_20)
#pred_edges5
#List predicted edges that are same as deleted edges.
deleted_edges6
```

```
## + 3/18 edges from 6b7ddad (vertex names):
## [1] F--H F--G A--D
```

```
selected_pred_edges5 <- pred_edges5$edges[c(17,7,1), ]
selected_pred_edges5
```

```
##      [,1] [,2]
## [1,]    6    8
## [2,]    6    7
## [3,]    1    4
```

Observations:

When we remove 10% of edges in karate data we can see that all the deleted edges are predicted at indexes 100,10,62,427,393,1,31 in predicted edge matrix. Based on probability in top 10 predicted edges, we have 2 deleted edge predictions.

When we remove 20% of edges in karate data we can see that all the deleted edges are predicted at indexes 50,73,14,298,313,1,71,63,40,10,8,12,41,11,93 in predicted edge matrix . Based on probability in top 20 predicted edges, we have 5 deleted edge predictions.

When we remove 10% of edges in kite data we can see that only one edge is deleted and it is predicted at index 16 in predicted edges matrix. Based on probability in top 10 predicted edges, we have no deleted edge predictions.

When we remove 20% of edges in kite data we can see that only 3 edges are deleted and these 3 edges are predicted at index 17,7,1 in predicted edges matrix. Based on probability in top 10 predicted edges, we have 2 deleted edge predictions.