10.7 Exercises

- Exercise 7
- Exercise 8
- Exercise 9
- Exercise 10
- Exercise 11

```
# Import any libraries needed for these exercises
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.4.4
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 3.4.4
```

```
## Loaded glmnet 2.0-16
```

Exercise 7

In the chapter, we mentioned the use of correlation-based distance and Euclidean distance as dissimilarity measures for hierarchical clustering. It turns out that these two measures are almost equivalent: if each observation has been centered to have mean zero and standard deviation one, and if we let r_{ij} denote the correlation between the *i*th and *j*th observations, then the quantity $1 - r_{ij}$ is proportional to the squared Euclidean distance between the *i*th and *j*th observations.

On the **USArrests** data, show that this proportionality holds.

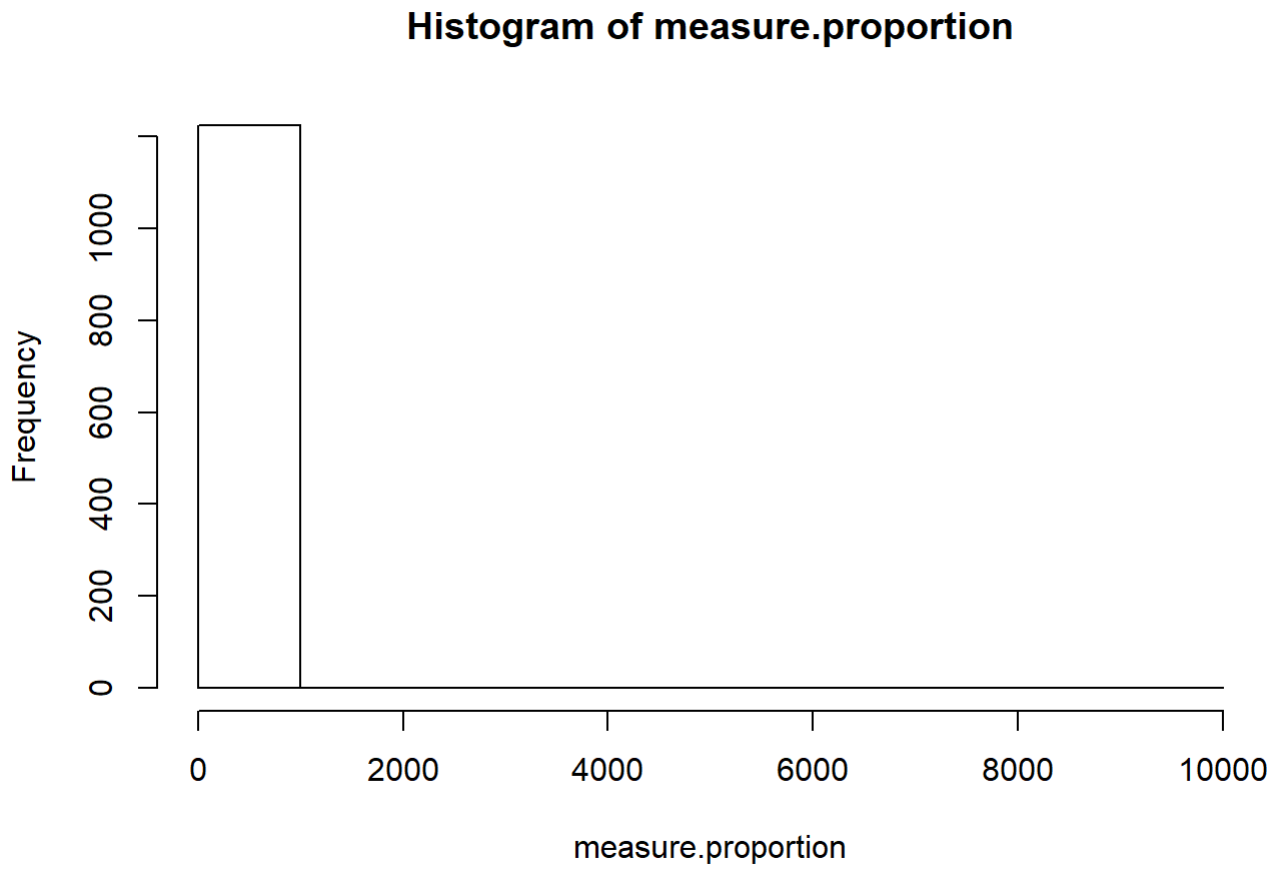
```
dim(USArrests)
```

```
## [1] 50  4
```

```
names(USArrests)
```

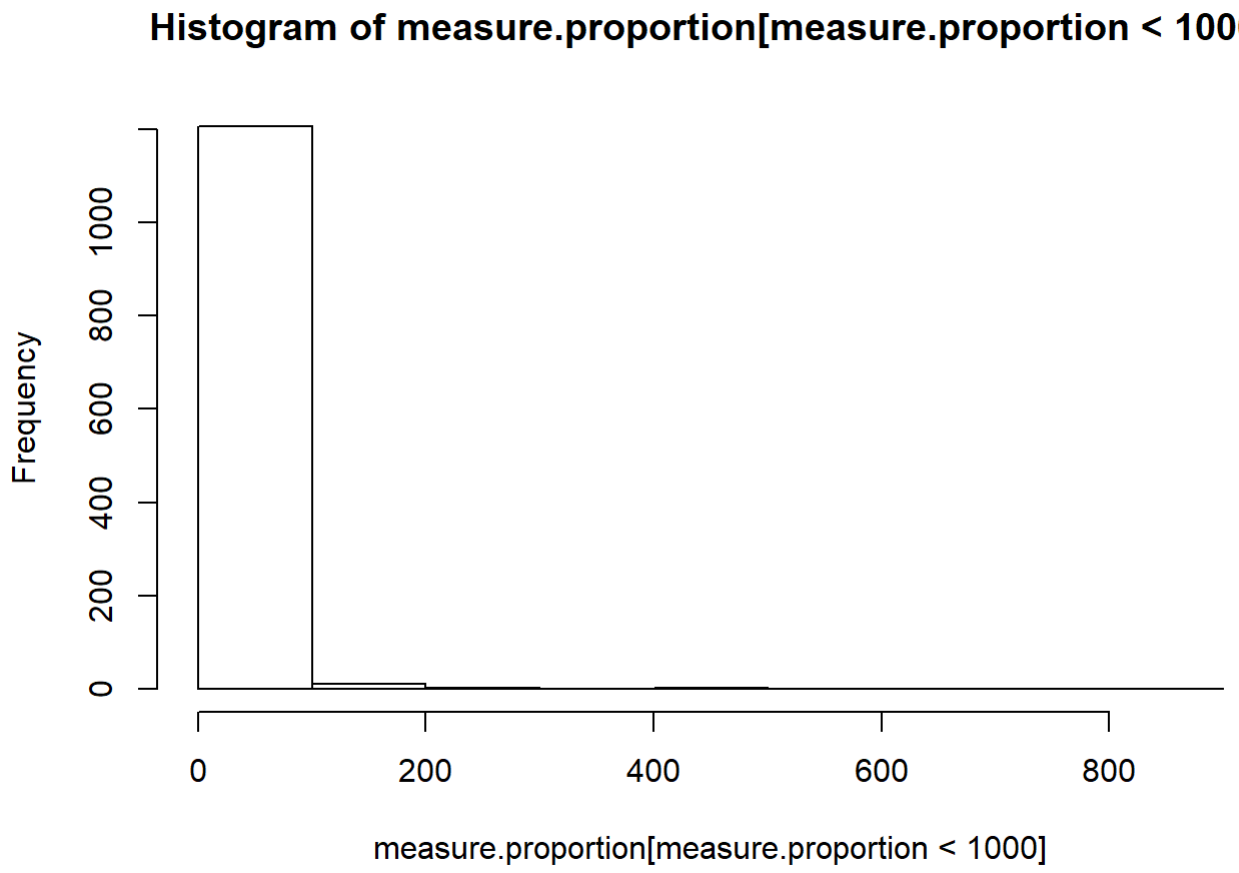
```
## [1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

```
# scale with both center and scale the columns in our data set
scaled.arrests = scale(USArrests)
dist.meas = dist(scaled.arrests)
cor.meas = as.dist(1 - cor(t(scaled.arrests)))
measure.proportion = dist.meas / cor.meas

hist(measure.proportion)
```



Well do to some outliers, it's hard to say much about the vast majority of the proportions.

```
hist(measure.proportion[measure.proportion < 1000])
```



Again, I guess it's tough to say. Maybe plots aren't the way to go…

```
# We can look at how well a linear relationship describes the dissimilarities as measure
d by covariance and Euclidean distance
summary(lm(dist.meas ~ cor.meas))
```

```
##
## Call:
## lm(formula = dist.meas ~ cor.meas)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.3538 -0.7874 -0.1029  0.6867  3.8537
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.98992    0.05656   35.18   <2e-16 ***
## cor.meas     0.59590    0.04714   12.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.063 on 1223 degrees of freedom
## Multiple R-squared:  0.1156, Adjusted R-squared:  0.1148
## F-statistic: 159.8 on 1 and 1223 DF,  p-value: < 2.2e-16
```

Well we produce a strong confidence in this linear fit between them, so that's good enough for me.

Exercise 8

In Section 10.2.3, a formula for calculating PVE was given in Equation 10.8. We also saw that the PVE can be obtained using the **sdev** output of the **prcomp()** function.

On the **USArrests** data, calculate PVE in two ways:

a. Using the **sdev** output of the **prcomp()** function, as was done in Section 10.2.3.

```
pr.out = prcomp(scaled.arrests, scale = F, center = F)
pr.var = pr.out$sdev^2
pve1 = pr.var / sum(pr.var)
print(pve1)
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

b. By applying Equation 10.8 directly. That is, use the **prcomp()** function to compute the principal component loadings. Then, use those loadings in Equation 10.8 to obtain the PVE.

For reference, here is Equation 10.8:

$$\frac{\sum_{i=1}^{n}(\sum_{j=1}^{p}\phi_{jm}x_{ij})}{\sum_{j=1}^{p}\sum_{i=1}^{n}x_{ij}^2}$$

Recall that for the m*th principal component, ϕ_{jm} is the scalar by which each x_{ij} is multiplied. It is the* j*th element of the* m*th principal loading vector*.

```
# We can access the principal loading vectors via the rotation output of the prcomp() fu
nction
rot.mat = pr.out$rotation
rot.mat
```

```
##                 PC1        PC2        PC3         PC4
## Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
## Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
## UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
## Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

```
denom = sum(scaled.arrests^2)
num = rep(NA, ncol(rot.mat))
for (m in 1:length(num)) {
  # This sweep function applys the multiplication of the loading vector to each row in s
caled.arrests
  # Due to the rules of matrix multiplication, a simple scaled.arrests * rot.mat[,m] doe
sn't work
  num[m] = sum(rowSums(sweep(scaled.arrests, MARGIN = 2, rot.mat[,m], "*"))^2)
}
pve2 = num / denom
print(pve2)
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

```
# Use some epsilon for rounding errors
print(pve1 - pve2 < 1e-9)
```

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

And we see that these approaches yield the same result.

Exercise 9

Consider the **USArrests** data. We will now perform hierarchical clustering on the states.

a. Using hierarchical clustering with complete linkage and Euclidean distance, cluster the states.

```
hc.unscaled = hclust(dist(USArrests), method = "complete")
```

b. Cut the dendrogram at a height that results in three distinct clusters. Which states belong to which clusters?

```
hc.unscaled.cut = cutree(hc.unscaled, 2)
hc.unscaled.cut
```

```
##        Alabama         Alaska        Arizona       Arkansas     California 
##              1              1              1              2              1 
##       Colorado    Connecticut       Delaware        Florida        Georgia 
##              2              2              1              1              2 
##         Hawaii          Idaho       Illinois        Indiana           Iowa 
##              2              2              1              2              2 
##         Kansas       Kentucky      Louisiana          Maine       Maryland 
##              2              2              1              2              1 
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri 
##              2              1              2              1              2 
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey 
##              2              2              1              2              2 
##     New Mexico       New York North Carolina   North Dakota           Ohio 
##              1              1              1              2              2 
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina 
##              2              2              2              2              1 
##   South Dakota      Tennessee          Texas           Utah        Vermont 
##              2              2              2              2              2 
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming 
##              2              2              2              2              2
```

c. Hierarchically cluster the states using complete linkage and Euclidean distance, after scaling the variables to have standard deviation one.

```
hc.scaled = hclust(dist(scale(USArrests, scale = T, center = F)), method = "complete")
```

d. What effect does scaling the variables have on the hierarchical clustering obtained? In your opinion, should the variables be scaled before the inter-observation dissimilarities are computed? Provide a justification for your answer.

```
hc.scaled.cut = cutree(hc.unscaled, 2)
hc.scaled.cut
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              1              1              2              1
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              2              2              1              1              2
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              2              2              1              2              2
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              2              2              1              2              1
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              1              2              1              2
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              2              2              1              2              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              1              1              1              2              2
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              2              2              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              2              2              2              2              2
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              2              2              2              2
```

```
hc.scaled.cut - hc.unscaled.cut
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              0              0              0              0              0
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              0              0              0              0              0
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              0              0              0              0              0
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              0              0              0              0              0
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              0              0              0              0              0
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              0              0              0              0              0
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              0              0              0              0              0
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              0              0              0              0              0
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              0              0              0              0              0
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              0              0              0              0              0
```

We can see that in this data, scaling the variables doesn't make any difference in the result.

```
var(USArrests)
```

```
##              Murder   Assault   UrbanPop      Rape
## Murder    18.970465  291.0624   4.386204  22.99141
## Assault  291.062367 6945.1657 312.275102 519.26906
## UrbanPop   4.386204  312.2751 209.518776  55.76808
## Rape      22.991412  519.2691  55.768082  87.72916
```

Looking at the features comprising our data set, I would say that we probably don't need to scale the variables, or at least not the arrest rate variables. **Murder, Assault**, *and* **Rape** *are all measured in arrests per 100,000. So they are captured under the same measurement. Furthermore, these variables are much more comparable in numbers than the example of sock purchases versus computer purchases. However, you could argue that the* **UrbanPop** *feature should be scaled somehow - perhaps to the mean standard deviation of the three arrest rate variables.*

Exercise 10

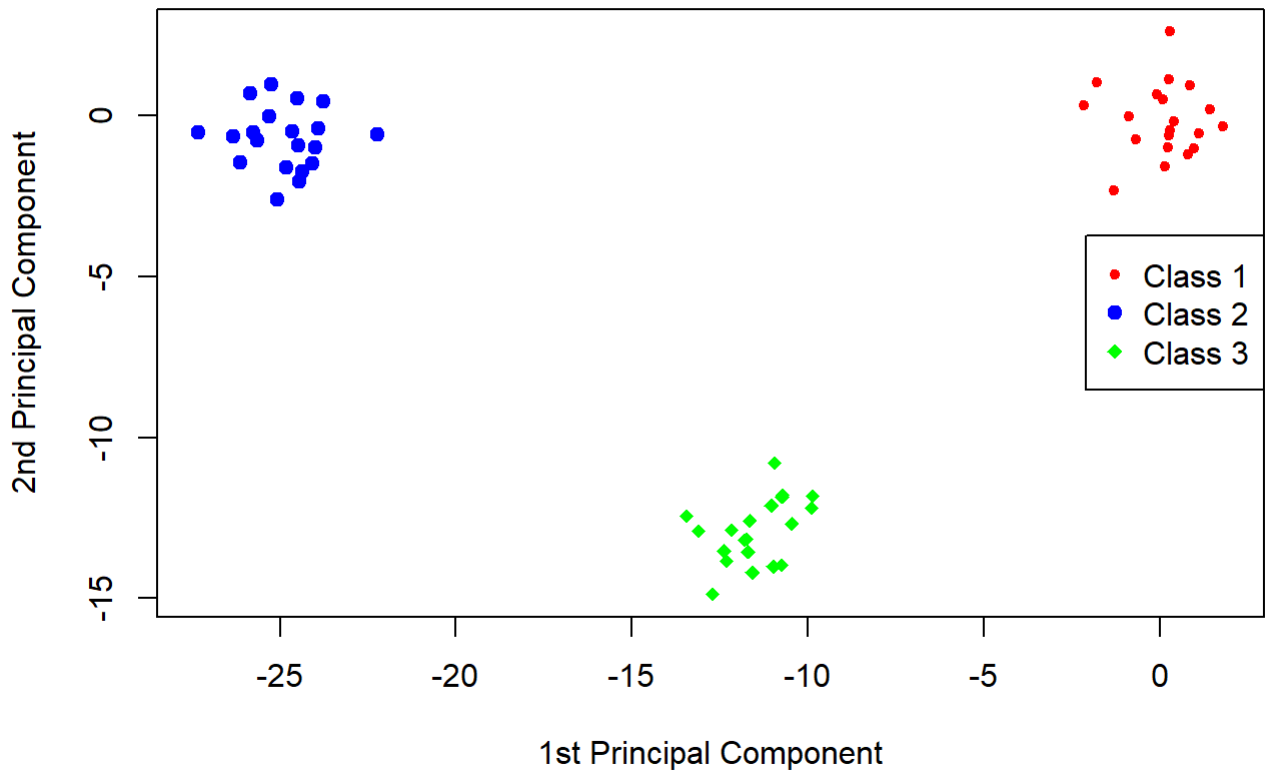
In this problem, you will generate simulated data and perform PCA and K-means clustering on the data.

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

```
set.seed(1)
n.classes = 3
n = 20 * n.classes
p = 50

# Generate all data as a normal distribution
x = matrix(data = rnorm(n * p, mean = 0, sd = 1), nrow = n, ncol = p)
class.index = matrix(data = sample(n, size = n, replace = F), nrow = n / n.classes, ncol
= n.classes)

# Create differences in the classes
## Class 1 will just be a normal distribution
## Class 2 will have a +5 offset for class 1..25
x[class.index[,2], 1:p/2] = x[class.index[,2], 1:p/2] + 5
## Class 3 will have a +5 offset for class 26..50
x[class.index[,3], p/2:p] = x[class.index[,3], p/2:p] + 5
```

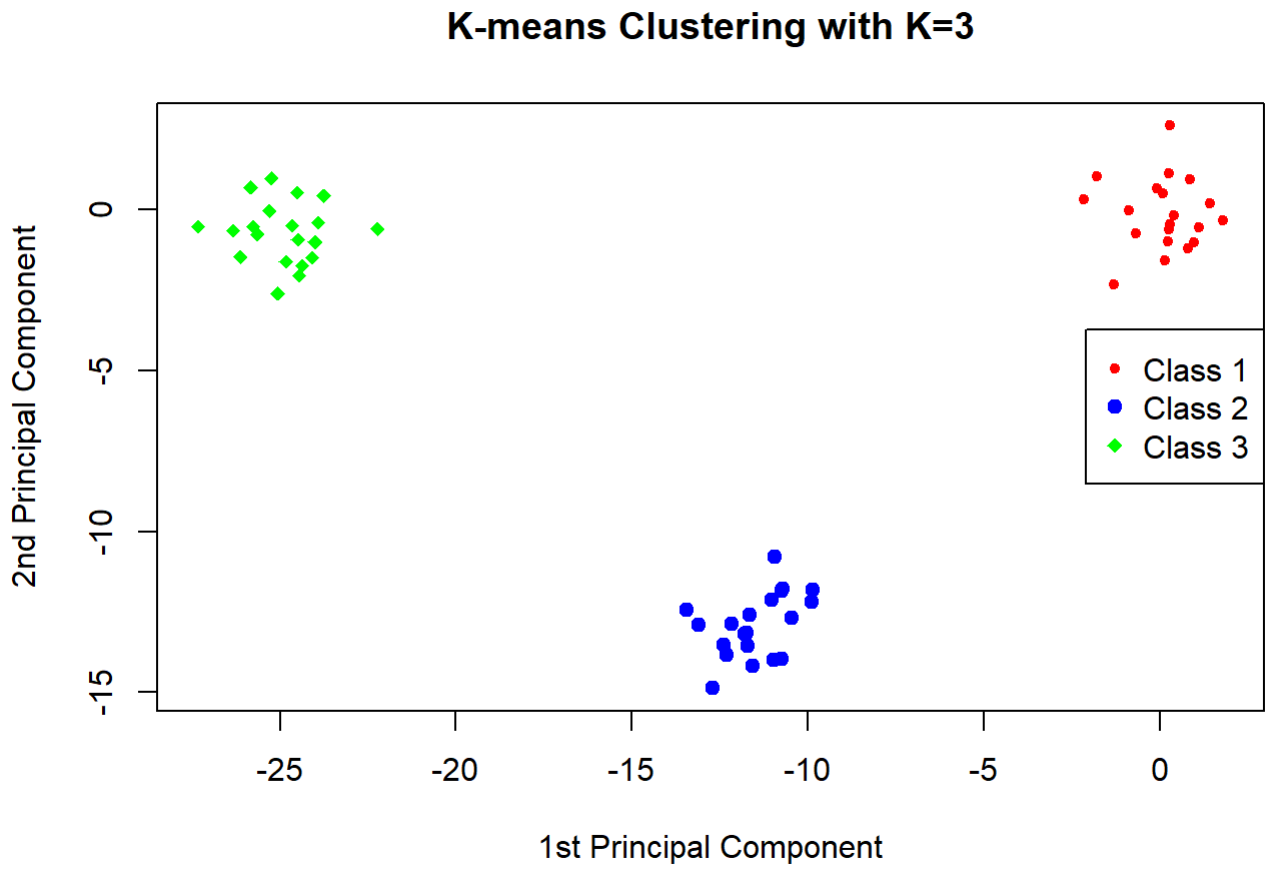
b. Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify 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.

```
pr.out = prcomp(x, scale = F)
score.vectors = matrix(data = rep(NA, n * 2), nrow = n, ncol = 2)
for (i in 1:n) {
  for (j in 1:2) {
    score.vectors[i, j] = sum(x[i,] * pr.out$rotation[,j])
  }
}

xlab = "1st Principal Component"
ylab = "2nd Principal Component"
xlim = c(min(score.vectors[,1]), max(score.vectors[,1]))
ylim = c(min(score.vectors[,2]), max(score.vectors[,2]))
plot(score.vectors[class.index[,1],], xlab = xlab, ylab = ylab, col = "red", pch = 20, x
lim = xlim, ylim = ylim)
points(score.vectors[class.index[,2],], col = "blue", pch = 19)
points(score.vectors[class.index[,3],], col = "green", pch = 18)
legend("right", col = c("red", "blue", "green"), pch = c(20, 19, 18), legend = c("Class
 1", "Class 2", "Class 3"))
```



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 true class labels?

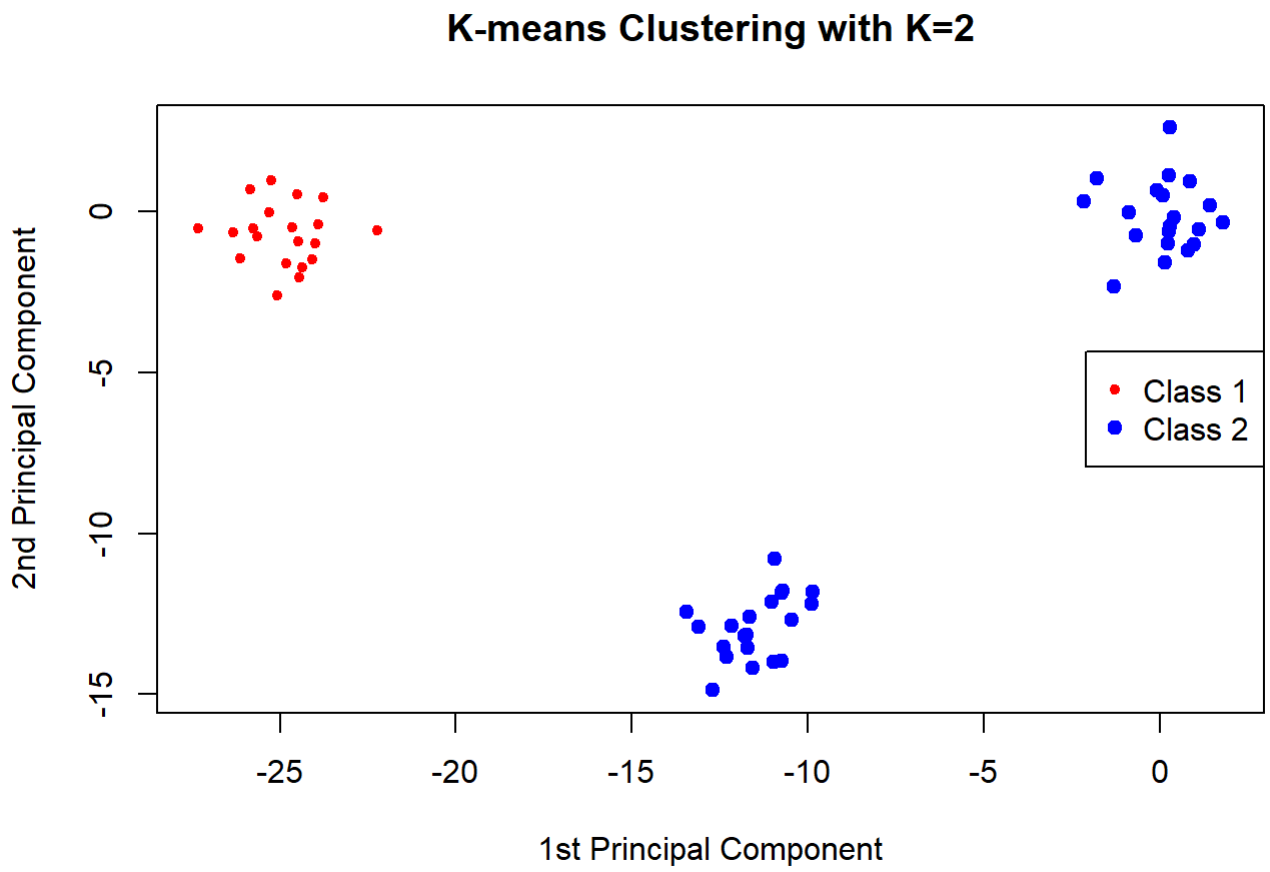
```
set.seed(1)
km.out = kmeans(x, 3, nstart = 20)

class.labels = c("Class 1", "Class 2", "Class 3")
class.colors = c("red", "blue", "green")
class.pch = c(20, 19, 18)
plot(c(), xlab = xlab, ylab = ylab, main = "K-means Clustering with K=3", xlim = xlim, y
lim = ylim)
for (k in 1:3) {
  points(score.vectors[km.out$cluster == k,], col = class.colors[k], pch = class.pch[k])
}
legend("right", col = class.colors, pch = class.pch, legend = class.labels)
```



We have perfect clustering with $K = 3$.

d. Perform K-means clustering with $K = 2$. Describe your results.

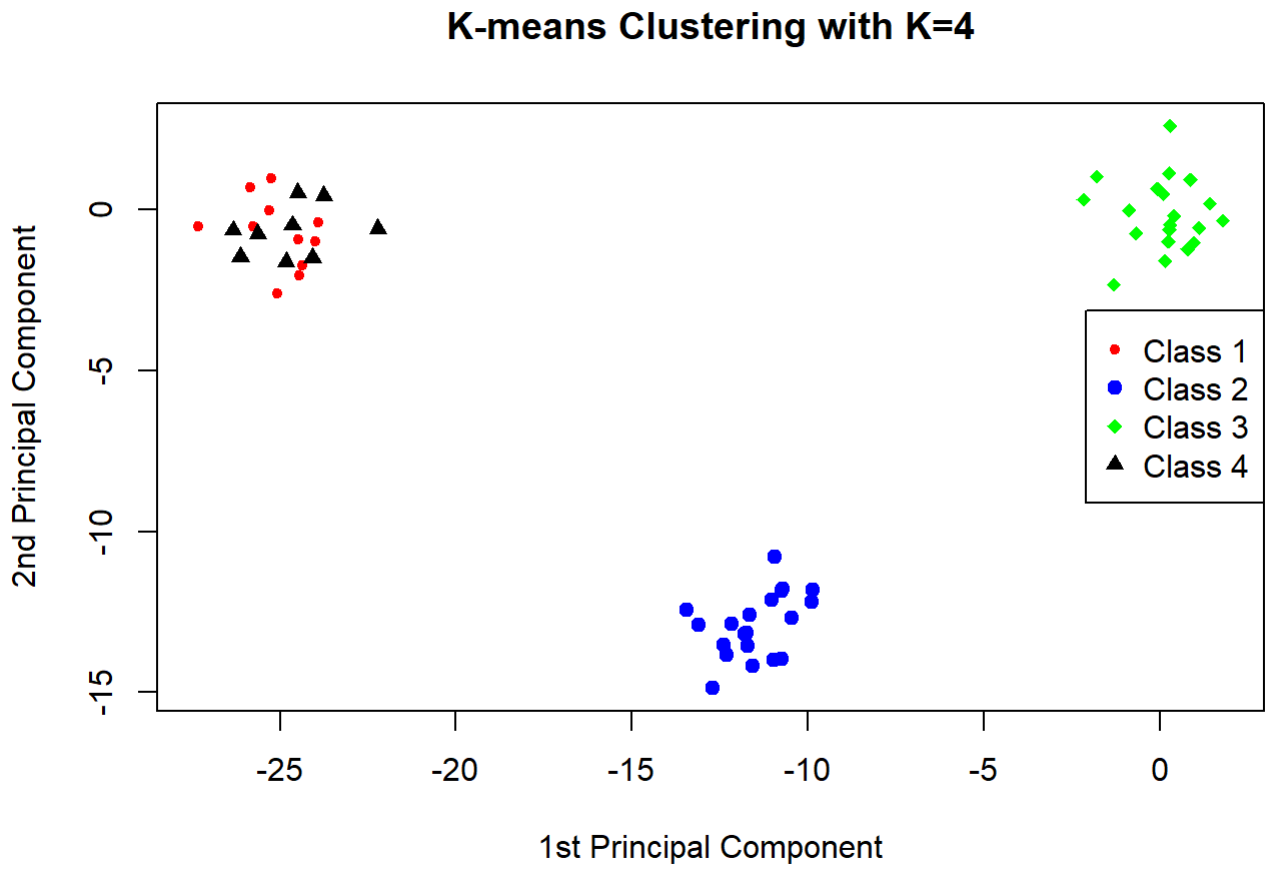
```
set.seed(1)
km.out = kmeans(x, 2, nstart = 20)

plot(c(), xlab = xlab, ylab = ylab, main = "K-means Clustering with K=2", xlim = xlim, y
lim = ylim)
for (k in 1:2) {
  points(score.vectors[km.out$cluster == k,], col = class.colors[k], pch = class.pch[k])
}
legend("right", col = class.colors[1:2], pch = class.pch[1:2], legend = class.labels[1:2
])
```



Class 3 is now a part of class 2, which I guess makes more sense than splitting the class in two.

e. Now perform K-means clustering with $K = 4$ and describe your results.

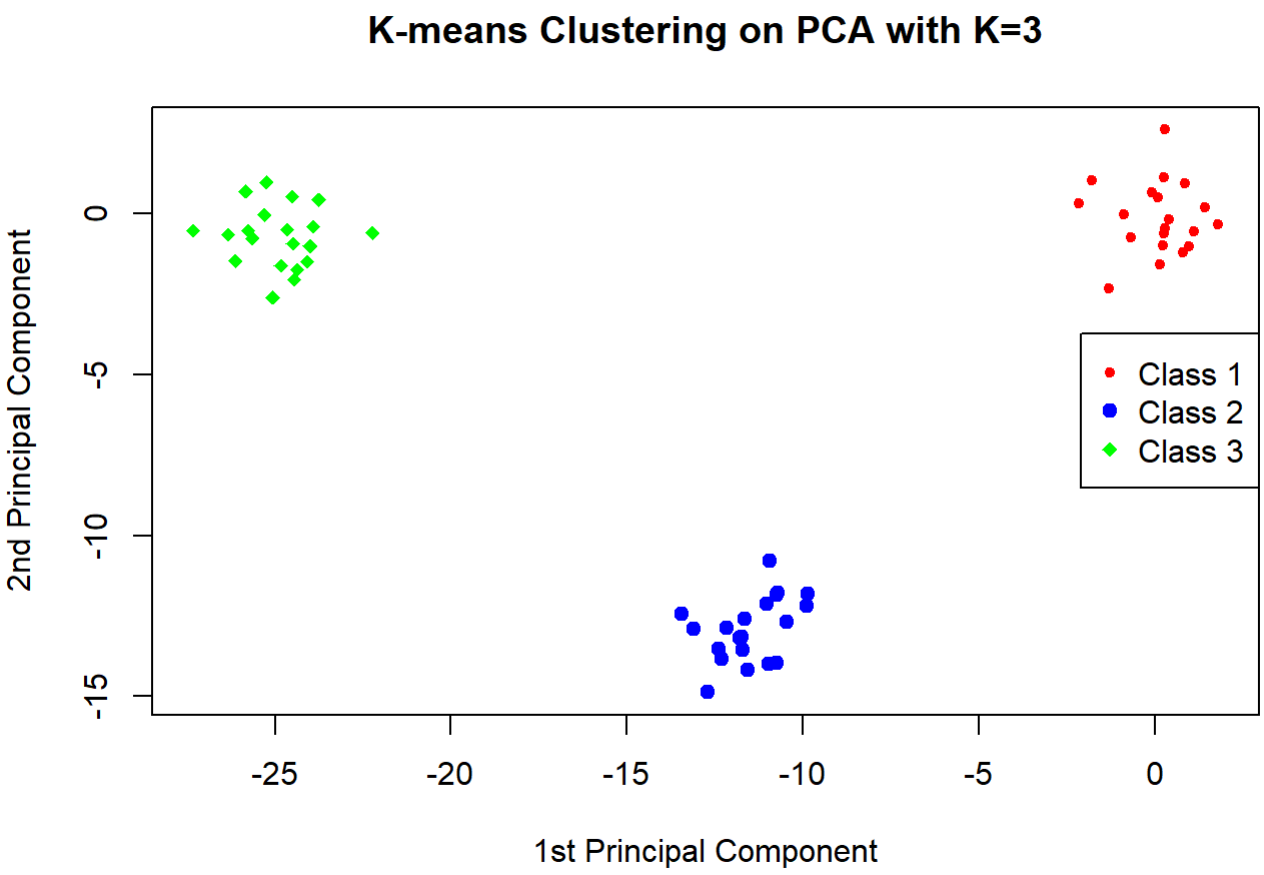
```
set.seed(1)
km.out = kmeans(x, 4, nstart = 20)

these.labels = c(class.labels, "Class 4")
these.colors = c(class.colors, "black")
these.pch = c(class.pch, 17)
plot(c(), xlab = xlab, ylab = ylab, main = "K-means Clustering with K=4", xlim = xlim, y
lim = ylim)
for (k in 1:4) {
  points(score.vectors[km.out$cluster == k,], col = these.colors[k], pch = these.pch[k])
}
legend("right", col = these.colors, pch = these.pch, legend = these.labels)
```



Class 1 has been divided seemingly randomly into two classes.

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-menas clustering on the 60 x 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.

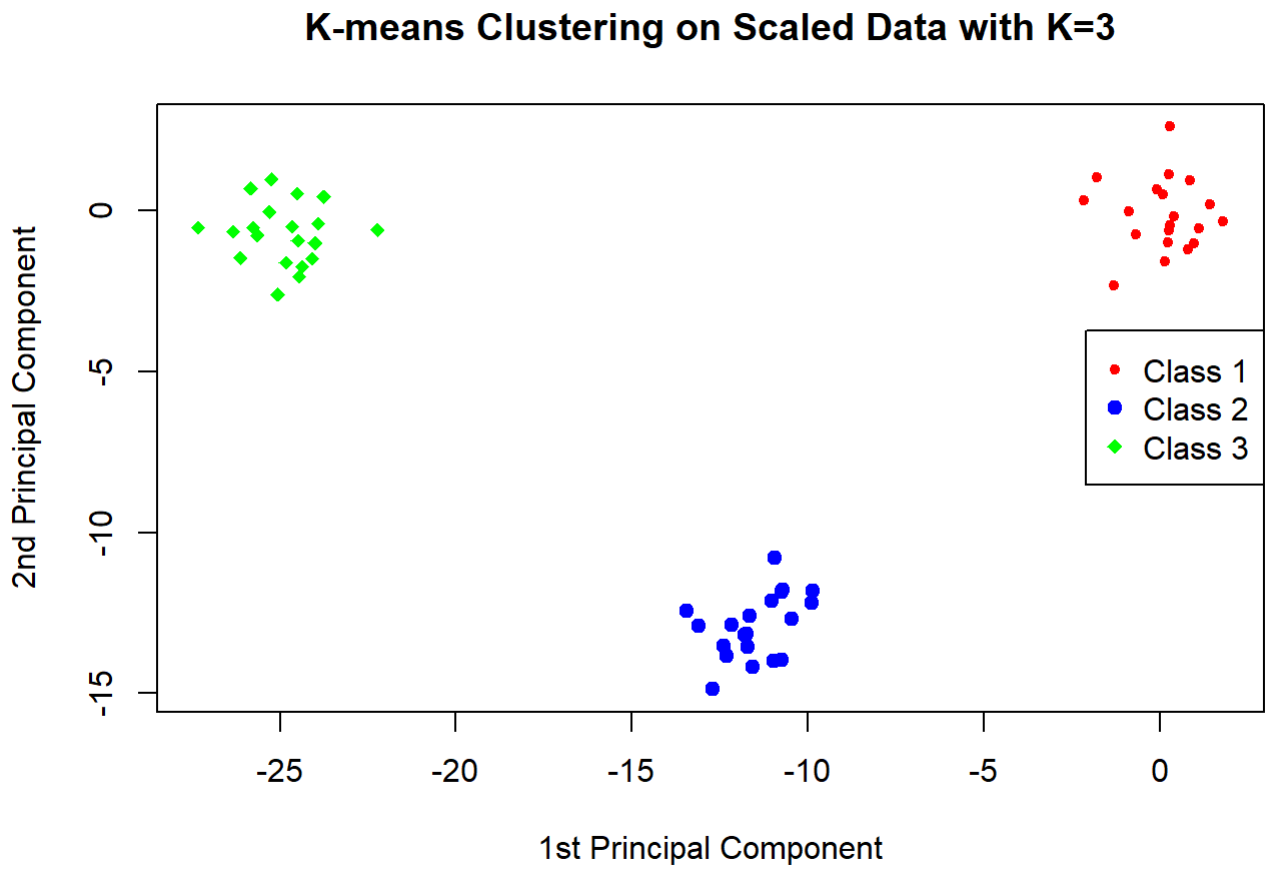
```
set.seed(1)
km.out = kmeans(score.vectors, 3, nstart = 20)

plot(c(), xlab = xlab, ylab = ylab, main = "K-means Clustering on PCA with K=3", xlim =
 xlim, ylim = ylim)
for (k in 1:3) {
  points(score.vectors[km.out$cluster == k,], col = class.colors[k], pch = class.pch[k])
}
legend("right", col = class.colors, pch = class.pch, legend = class.labels)
```



Again, we have perfect clustering.

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.

```
set.seed(1)
km.out = kmeans(scale(x, center = T, scale = T), 3, nstart = 20)

plot(c(), xlab = xlab, ylab = ylab, main = "K-means Clustering on Scaled Data with K=3",
xlim = xlim, ylim = ylim)
for (k in 1:3) {
  points(score.vectors[km.out$cluster == k,], col = class.colors[k], pch = class.pch[k])
}
legend("right", col = class.colors, pch = class.pch, legend = class.labels)
```



And again, we have perfect clustering. I suppose I could have generated the classes with a bit more overlap to make the results more interesting.

Exercise 11

On the book website, www.StatLearning.com, there is a gene expression data set (**Ch10Ex11.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**.

```
data = read.csv("D:\\GoogleDrive\\Introduction to Statistical Learning with Applications
in R\\data-sets\\Ch10Ex11.csv", header = F)
dim(data)
```

```
## [1] 1000   40
```

```
# clearly the genes are represented by the rows, but we want to cluster the samples, so
 we need to transpose the data
data = t(data)
dim(data)
```

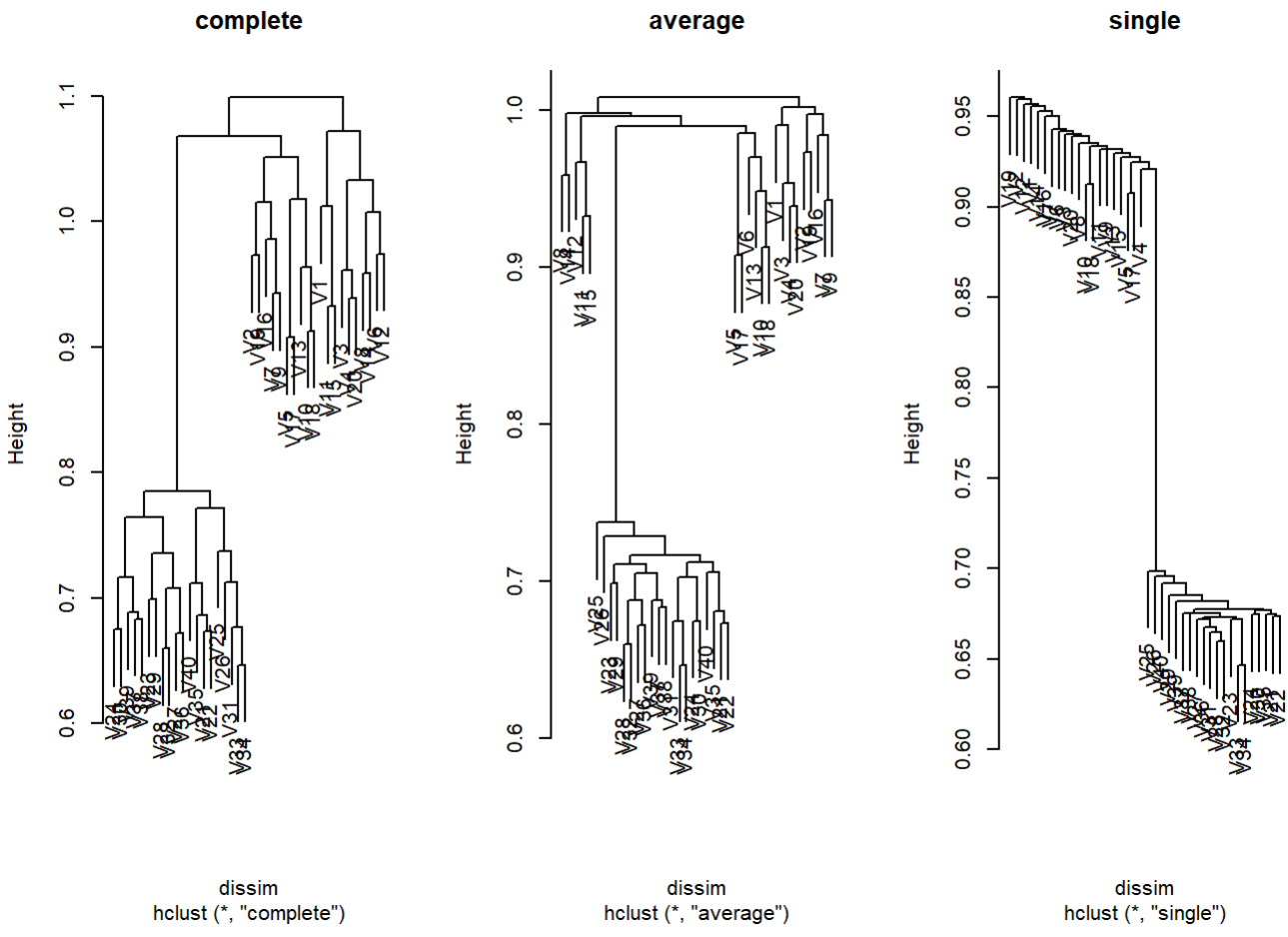
```
## [1]   40 1000
```

b. Apply hierarchical clustering to the samples using correlation-based distance and plot the dendrogram. Do the genese separate the samples into the two groups? Do your results depend on the type of linkage used?

```
dissim = as.dist(1 - cor(t(data)))
n = dim(data)[1]
true.category = rep(2, n)
true.category[1:20] = 1
par(mfrow = c(1,3))
for (linkage in c("complete", "average", "single")) {
  hc.out = hclust(dissim, method = linkage)
  plot(hc.out, main = linkage)
  hc.cut = cutree(hc.out, 2)
  num.incorrect = sum(true.category != hc.cut)
  if (num.incorrect > n/2) {
    num.incorrect = n - num.incorrect
  }
  print(paste(linkage, num.incorrect))
}
```

```
## [1] "complete 10"
```

```
## [1] "average 11"
```



```
## [1] "single 19"
```

None of these linkage methods perfectly separate the two groups using correlation-based distance. The complete linkage comes the closest though.

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.

```
# Since we already know which genes belong to which groups, we can use a supervised technique which selects a subset
# of the predictors (in this case a subset of the genes) to predict the response (in this case either 'healthy' or 'diseased')
# Here, we will apply the lasso technique to determine which genes differ most across the two groups
set.seed(1)
HEALTHY = 0
DISEASED = 1
y = rep(HEALTHY, n)
y[20:n] = DISEASED
cv.out = cv.glmnet(data, y, alpha = 1)
bestlambda = cv.out$lambda.min
lasso.out = glmnet(data, y, alpha = 1, lambda = bestlambda)
lasso.coef = predict(lasso.out, type = 'coefficients', s = bestlambda)
which(lasso.coef != 0)
```

```
##  [1]   1   5  17  21  55 103 136 157 233 311 383 425 503 507 512 520 521
## [18] 525 534 535 542 559 565 567 570 574 577 580 585 589 599 601 621 642
## [35] 795 829 857 893 983
```

```
sum(lasso.coef != 0)
```

```
## [1] 39
```

So using the lasso, we can narrow the data down quite a bit to only 39 of the original 1000 genes as being predictive of a healthy or diseased subject.