ISLR - Ch 10 - PCA

10.4 Lab 1: Principal Components Analysis

In this lab, we perform PCA on the USArrests data set, which is part of the base R package. The rows of the data set contain the 50 states, in alphabetical order.

```
states=row.names(USArrests )
head(states)
```

```
## [1] "Alabama"    "Alaska"     "Arizona"    "Arkansas"   "California"
## [6] "Colorado"
```

The columns of the data set contain the four variables.

```
names(USArrests)
```

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

We first briefly examine the data. We notice that the variables have vastly different means.

```
apply(USArrests, 2, mean)
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

Note that the apply() function allows us to apply a function—in this case, the mean() function—to each row or column of the data set. The second input here denotes whether we wish to compute the mean of the rows, 1, or the columns, 2. We see that there are on average three times as many rapes as murders, and more than eight times as many assaults as rapes.

We can also examine the variances of the four variables using the apply() function.

```
apply(USArrests, 2, var)
```

```
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916
```

Not surprisingly, the variables also have vastly different variances: the UrbanPop variable measures the percentage of the population in each state living in an urban area, which is not a comparable number to the number of rapes in each state per 100,000 individuals. If we failed to scale the variables before performing PCA, then most of the principal components that we observed would be driven by the Assault variable, since it has by far the largest mean and variance. Thus, it is important to standardize the variables to have mean zero and standard deviation one before performing PCA.

We now perform principal components analysis using the prcomp() function, which is one of several functions in R that perform PCA.

```r
pr.out=prcomp(USArrests, scale=TRUE)
```

By default, the prcomp() function centers the variables to have mean zero. By using the option scale=TRUE, we scale the variables to have standard deviation one. The output from prcomp() contains a number of useful quantities.

```r
names(pr.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

The center and scale components correspond to the means and standard deviations of the variables that were used for scaling prior to implementing PCA.

```r
pr.out$center
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```r
pr.out$scale
```

```
##    Murder   Assault  UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
```

The rotation matrix provides the principal component loadings; each column of “pr.out*rotation”containsthecorrespondingprinc* *multiplytheXmatrixby”pr.out*rotation”, it gives us the coordinates of the data in the rotated coordinate system. These coordinates are the principal component scores.

```r
pr.out$rotation
```

```
##                 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
```

We see that there are four distinct principal components. This is to be expected because there are in general min(n - 1, p) informative principal components in a data set with n observations and p variables.

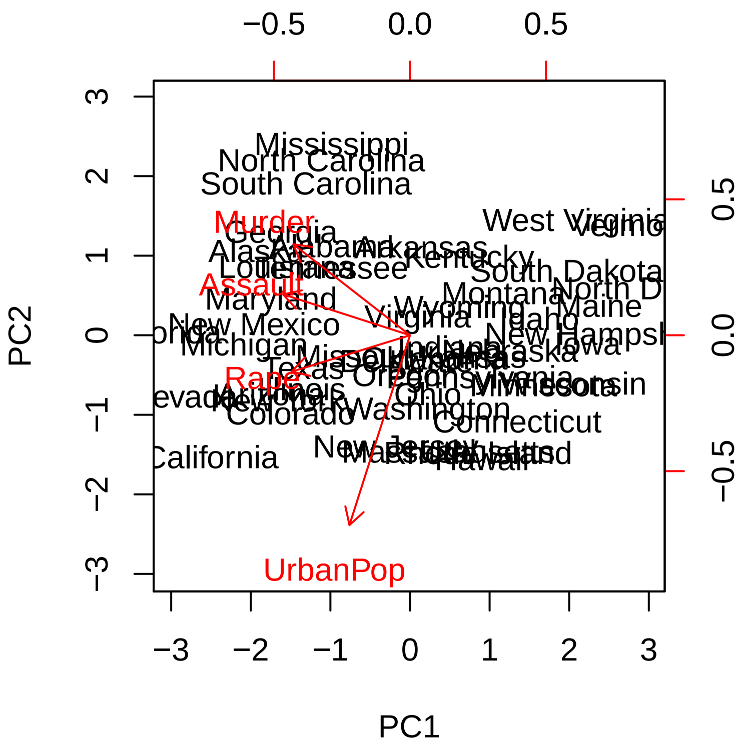
Using the prcomp() function, we do not need to explicitly multiply the data by the principal component loading vectors in order to obtain the principal component score vectors. Rather the 50×4 matrix x has as its columns the principal component score vectors. That is, the kth column is the kth principal component score vector.

```r
dim(pr.out$x)
```

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

We can plot the first two principal components as follows:

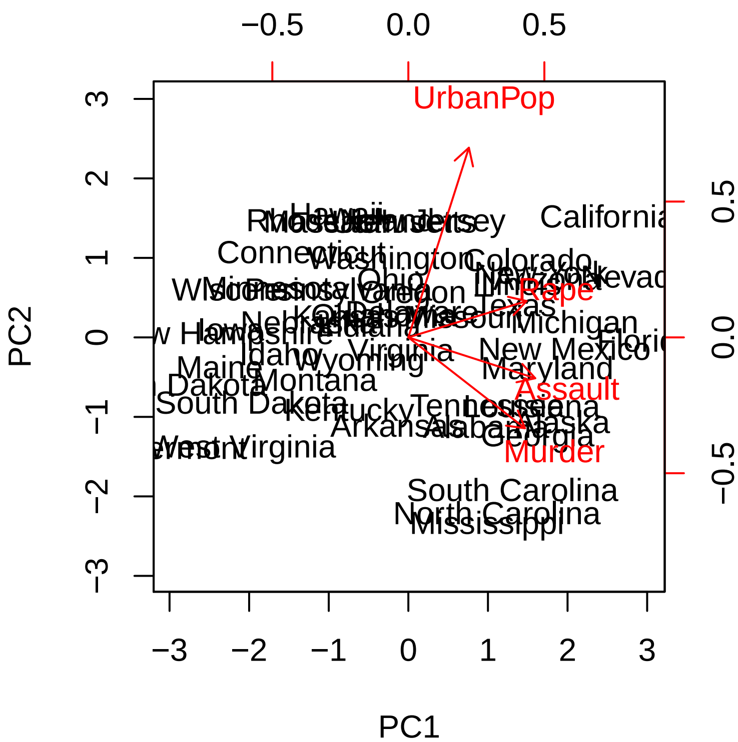
```
biplot(pr.out, scale =0)
```



The scale=0 argument to biplot() ensures that the arrows are scaled to represent the loadings; other values for scale give slightly different biplots with different interpretations.

Notice that this figure is a mirror image of Figure 10.1. Recall that the principal components are only unique up to a sign change, so we can reproduce the Figure by making a few small changes:

```
pr.out$rotation=-pr.out$rotation
pr.out$x=-pr.out$x
biplot(pr.out, scale =0)
```



The prcomp() function also outputs the standard deviation of each principal component. For instance, on the USArrests data set, we can access these standard deviations as follows:

```
pr.out$sdev
```

```
## [1] 1.5748783 0.9948694 0.5971291 0.4164494
```

The variance explained by each principal component is obtained by squaring these:

```
pr.var=pr.out$sdev^2
pr.var
```

```
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

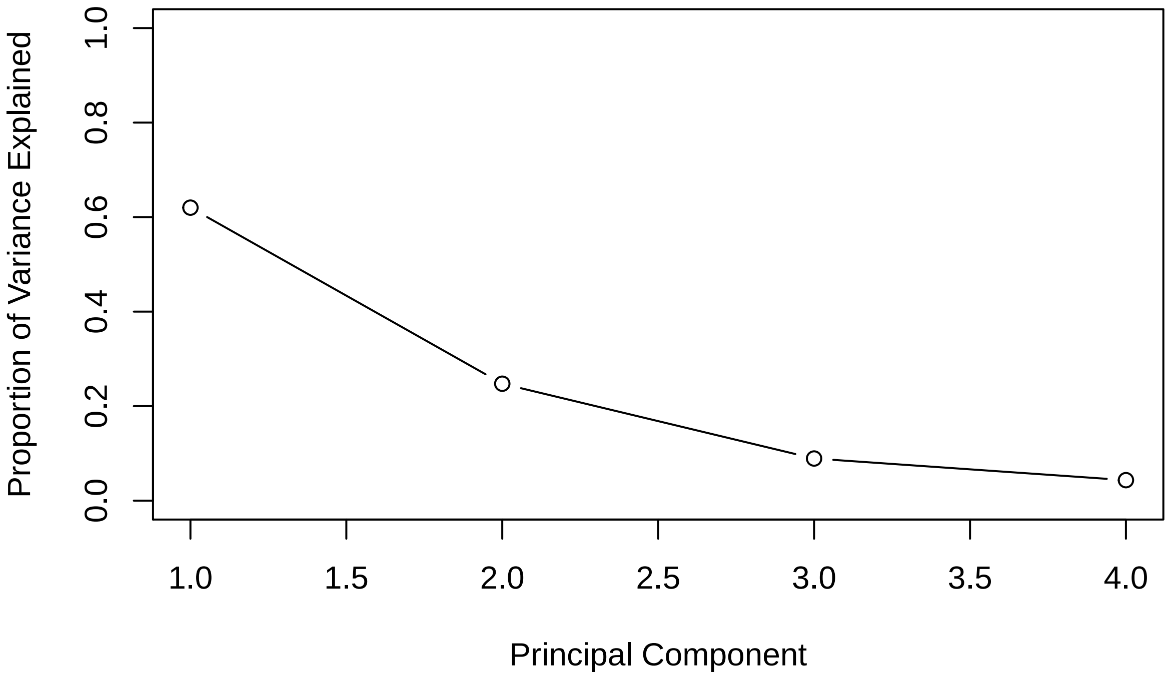
To compute the proportion of variance explained by each principal component, we simply divide the variance explained by each principal component by the total variance explained by all four principal components:

```
pve=pr.var/sum(pr.var)
pve
```

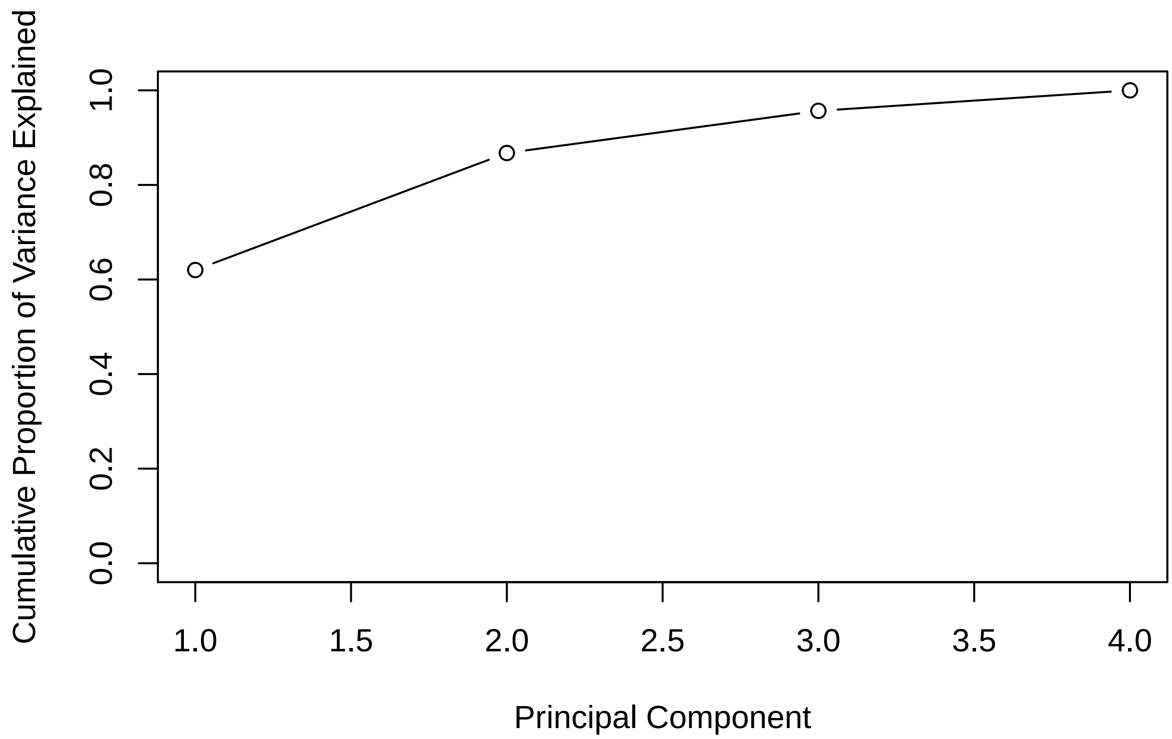
```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

We see that the first principal component explains 62.0 % of the variance in the data, the next principal component explains 24.7 % of the variance, and so forth. We can plot the PVE explained by each component, as well as the cumulative PVE, as follows:

```
plot(pve, xlab="Principal Component", ylab="Proportion of Variance Explained", ylim=c(0,1),type="b")
```




```
plot(cumsum(pve), xlab="Principal Component", ylab="Cumulative Proportion of Variance Explained", ylim=c
```



Note the function cumsum() computes the cumulative sum of the elements of a numeric vector. For instance:

```
a=c(1,2,8,-3)
cumsum(a)
```

```
## [1]  1  3 11  8
```

10.5 Lab 2: Clustering

10.5.1 K-Means Clustering

The function kmeans() performs K-means clustering in R. We begin with kmeans() a simple simulated example in which there truly are two clusters in the data: the first 25 observations have a mean shift relative to the next 25 observations.

```
set.seed(2)
x=matrix(rnorm (50*2), ncol=2)
x[1:25,1]=x[1:25,1]+3
x[1:25,2]=x[1:25,2]-4
```

We now perform K-means clustering with K = 2.

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

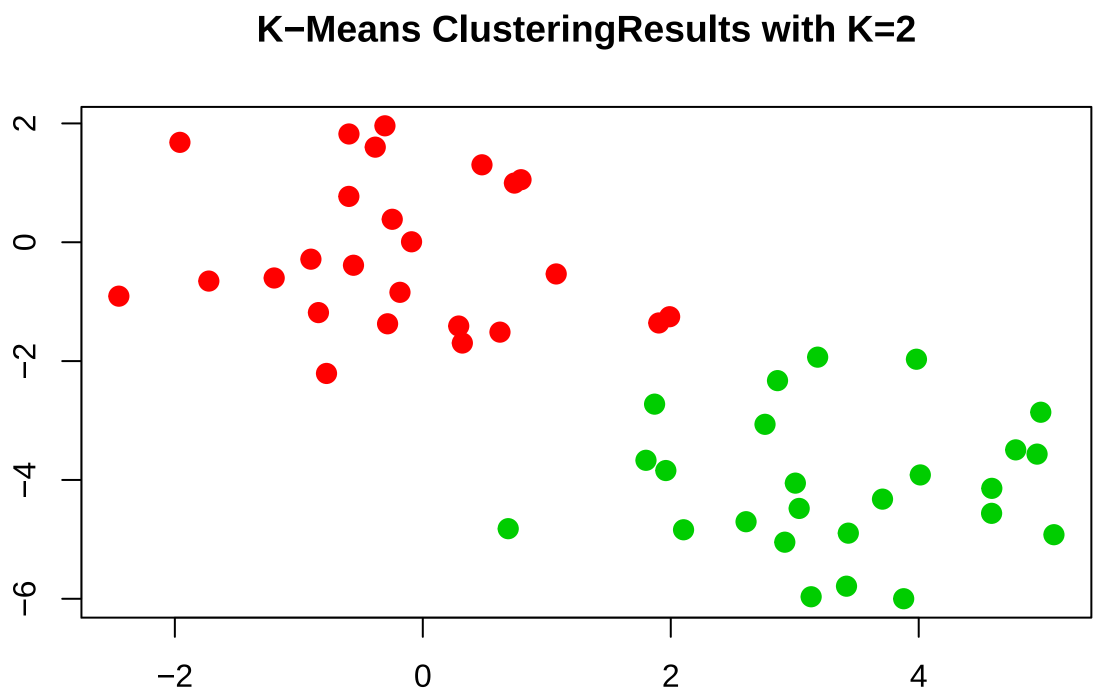
The cluster assignments of the 50 observations are contained in km.out$cluster.

```
km.out$cluster
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The K-means clustering perfectly separated the observations into two clusters even though we did not supply any group information to kmeans(). We can plot the data, with each observation colored according to its cluster assignment.

```
plot(x, col=(km.out$cluster + 1), main="K-Means ClusteringResults with K=2", xlab="", ylab="", pch=20, c
```



Here the observations can be easily plotted because they are two-dimensional. If there were more than two variables then we could instead perform PCA and plot the first two principal components score vectors. In this example, we knew that there really were two clusters because we generated the data. However, for real data, in general we do not know the true number of clusters. We could instead have performed K-means clustering on this example with K = 3.

```
set.seed(4)
km.out=kmeans(x, 3, nstart = 20)
km.out
```

```
## K-means clustering with 3 clusters of sizes 10, 23, 17
##
## Cluster means:
```

```
##         [,1]        [,2]
## 1  2.3001545 -2.69622023
## 2 -0.3820397 -0.08740753
## 3  3.7789567 -4.56200798
##
## Clustering vector:
##  [1] 3 1 3 1 3 3 3 1 3 1 3 1 3 1 3 1 3 3 3 3 3 1 3 3 3 2 2 2 2 2 2 2 2 2 2
## [36] 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 19.56137 52.67700 25.74089
##  (between_SS / total_SS =  79.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

When K = 3, K-means clustering splits up the two clusters. To run the kmeans() function in R with multiple initial cluster assignments, we use the nstart argument. If a value of nstart greater than one is used, then K-means clustering will be performed using multiple random assignments and the kmeans() function will report only the best results. Here we compare using nstart=1 to nstart=20.

```r
set.seed(3)
km.out=kmeans(x, 3, nstart = 1)
km.out$tot.withinss
```

```
## [1] 104.3319
```

```r
km.out=kmeans(x, 3, nstart = 20)
km.out$tot.withinss
```

```
## [1] 97.97927
```

Note that "km.out$tot.withinss""isthetotalwithin-clustersumofsquares, whichweseektominimizebyperformingK-meansclustering.Theindividualwithin-clustersum-of-squaresarecontainedinthevector"km.out$withinss".

We **strongly recommend** always running K-means clustering with a large value of nstart, such as 20 or 50, since otherwise an undesirable local optimum may be obtained.

When performing K-means clustering, in addition to using multiple initial cluster assignments, it is also important to set a random seed using the set.seed() function. This way, the initial cluster assignments can be replicated, and the K-means output will be fully reproducible.

10.5.2 Hierarchical Clustering

The hclust() function implements hierarchical clustering in R. In the following example we plot the hierarchical clustering dendrogram using complete, single, and average linkage clustering, with Euclidean distance as the dissimilarity measure. We begin by clustering observations using complete linkage. The dist() function is used to compute the 50×50 inter-observation Euclidean distance matrix.

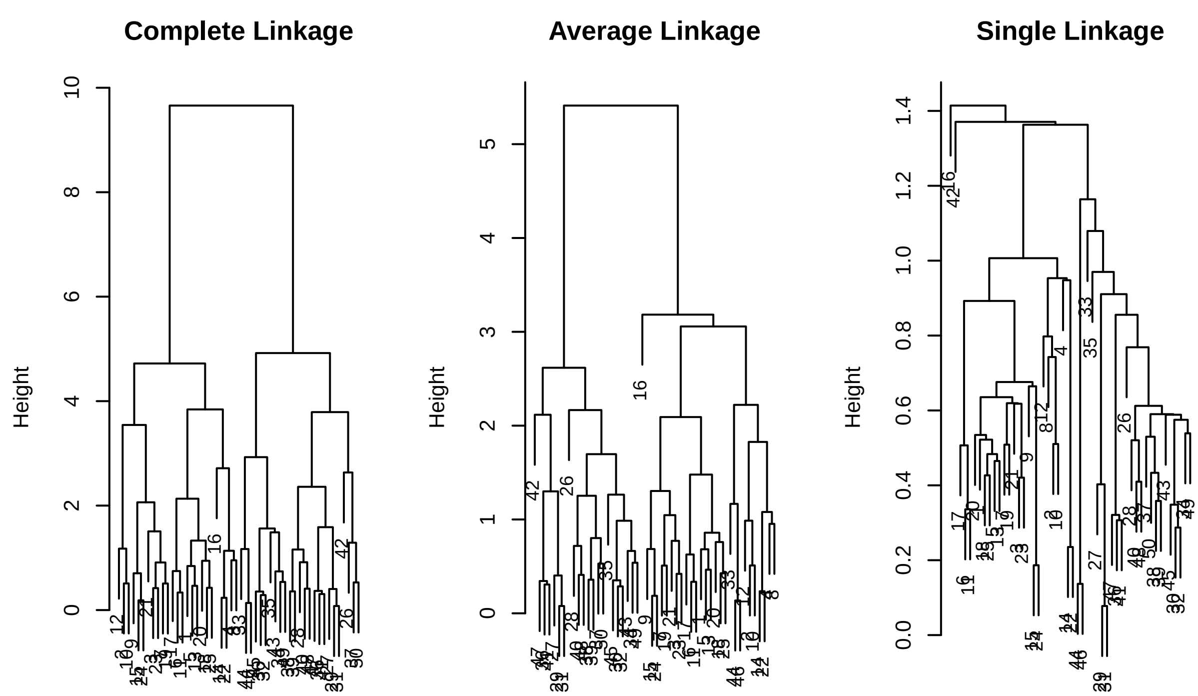
```
hc.complete = hclust(dist(x), method="complete")
```

We could just as easily perform hierarchical clustering with average or single linkage instead:

```
hc.average =hclust(dist(x), method ="average")
hc.single=hclust(dist(x), method ="single")
```

We can now plot the dendrograms obtained using the usual plot() function. The numbers at the bottom of the plot identify each observation.

```
par(mfrow=c(1,3))
plot(hc.complete, main="Complete Linkage", xlab="", sub="", cex=.9)
plot(hc.average, main="Average Linkage", xlab="", sub="", cex=.9)
plot(hc.single, main="Single Linkage", xlab="", sub="", cex=.9)
```



To determine the cluster labels for each observation associated with a given cut of the dendrogram, we can use the cutree() function:

```
cutree(hc.complete, 2)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
## [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```r
cutree(hc.average, 2)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 2 2
## [36] 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2
```

```r
cutree(hc.single, 2)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

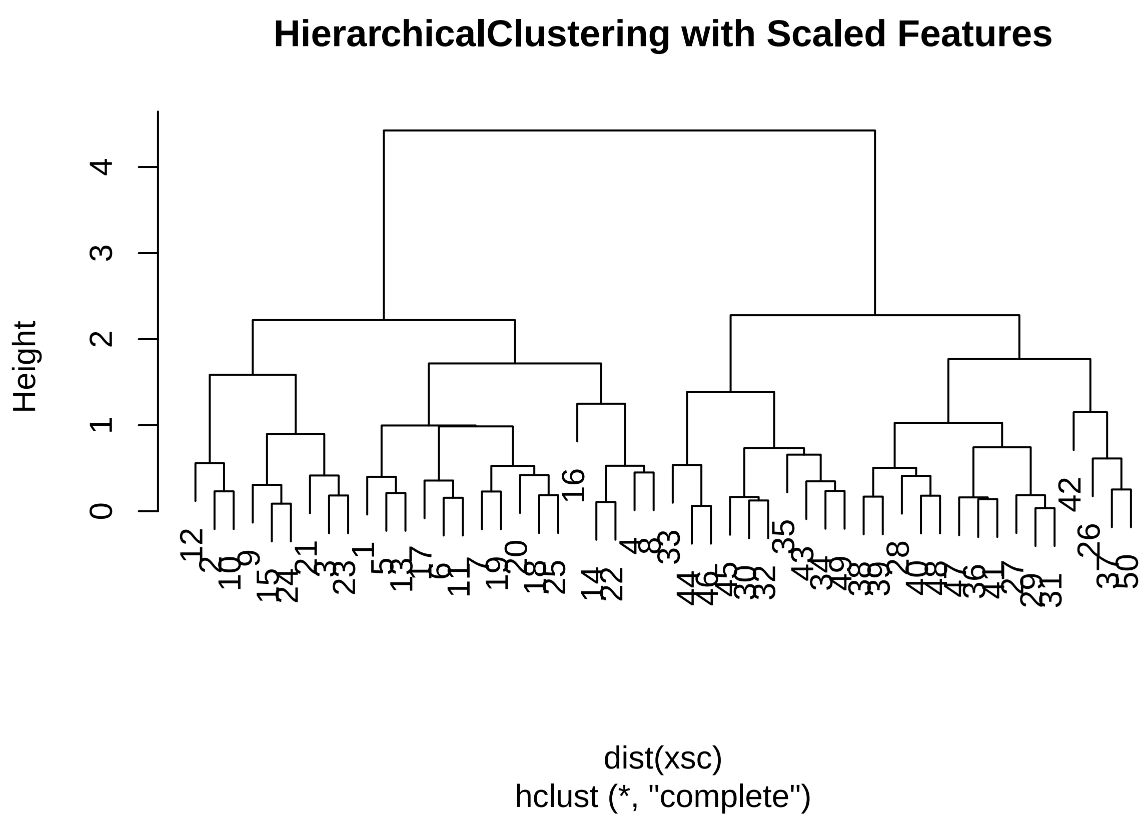
For this data, complete and average linkage generally separate the observations into their correct groups. However, single linkage identifies one point as belonging to its own cluster. A more sensible answer is obtained when four clusters are selected, although there are still two singletons.

```r
cutree(hc.single, 4)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3
## [36] 3 3 3 3 3 3 4 3 3 3 3 3 3 3 3
```

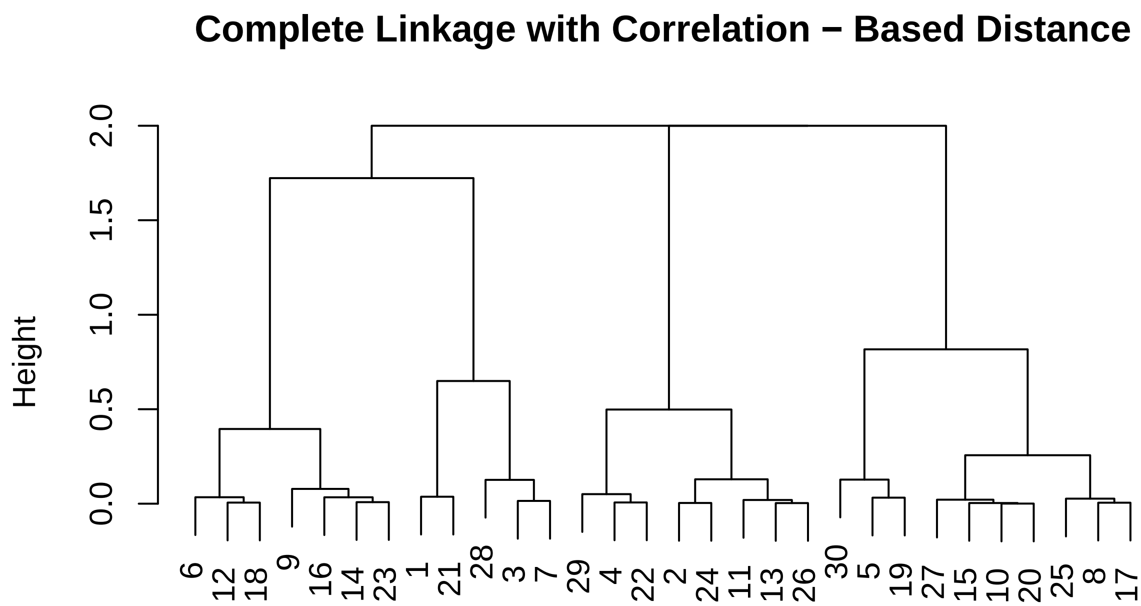
To scale the variables before performing hierarchical clustering of the observations, we use the scale() function:

```r
xsc=scale(x)
plot(hclust(dist(xsc), method ="complete"), main ="HierarchicalClustering with Scaled Features")
```



Correlation-based distance can be computed using the as.dist() function, which converts an arbitrary square symmetric matrix into a form that the hclust() function recognizes as a distance matrix. However, this only makes sense for data with at least three features since the absolute correlation between any two observations with measurements on two features is always 1. Hence, we will cluster a three-dimensional data set.

```
x=matrix(rnorm(30*3), ncol=3)
dd=as.dist(1-cor(t(x)))
plot(hclust(dd, method ="complete"), main="Complete Linkage with Correlation - Based Distance", xlab=""
```



###10.6 Lab 3: NCI60 Data Example

Unsupervised techniques are often used in the analysis of genomic data. In particular, PCA and hierarchical clustering are popular tools. We illustrate these techniques on the NCI60 cancer cell line microarray data, which consists of 6,830 gene expression measurements on 64 cancer cell lines.

```
library(ISLR)
nci.labs=NCI60$labs
nci.data=NCI60$data
```

Each cell line is labeled with a cancer type. We do not make use of the cancer types in performing PCA and clustering, as these are unsupervised techniques. But after performing PCA and clustering, we will check to see the extent to which these cancer types agree with the results of these unsupervised techniques.

The data has 64 rows and 6,830 columns.

```r
dim(nci.data)
```

```
## [1]   64 6830
```

We begin by examining the cancer types for the cell lines.

```r
nci.labs[1:4]
```

```
## [1] "CNS"    "CNS"    "CNS"    "RENAL"
```

```r
table(nci.labs)
```

```
## nci.labs
##      BREAST         CNS       COLON K562A-repro K562B-repro    LEUKEMIA
##           7           5           7           1           1           6
## MCF7A-repro MCF7D-repro    MELANOMA       NSCLC     OVARIAN    PROSTATE
##           1           1           8           9           6           2
##       RENAL     UNKNOWN
##           9           1
```

10.6.1 PCA on the NCI60 Data We first perform PCA on the data after scaling the variables (genes) to have standard deviation one, although one could reasonably argue that it is better not to scale the genes.

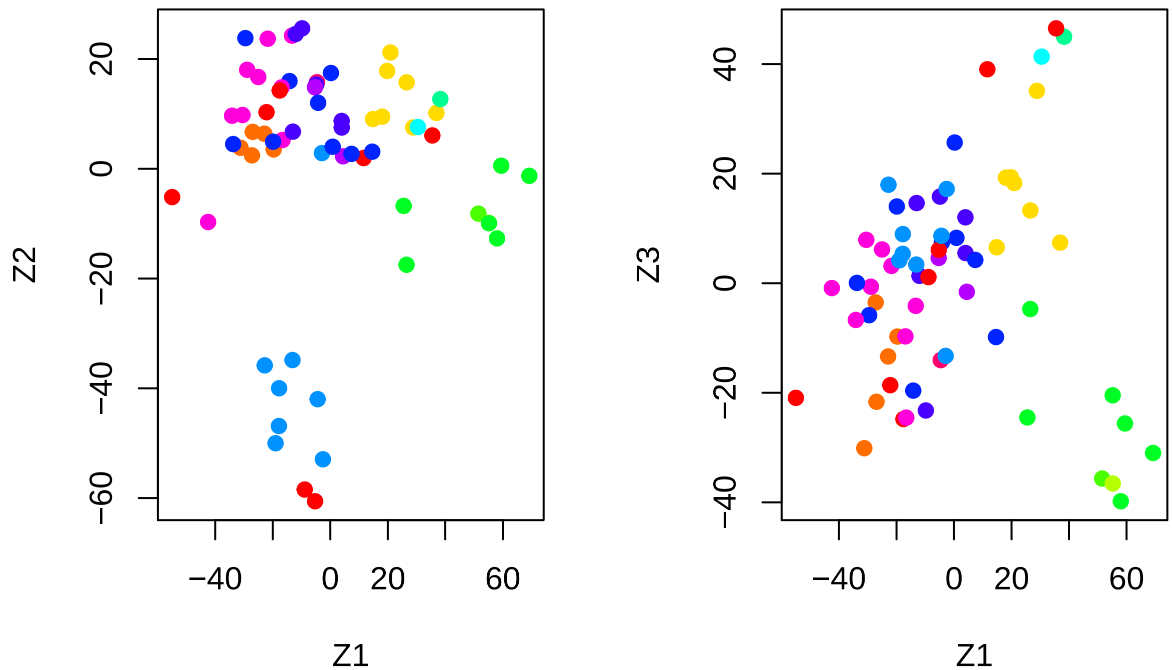
```r
pr.out=prcomp(nci.data, scale=TRUE)
```

We now plot the first few principal component score vectors, in order to visualize the data. The observations (cell lines) corresponding to a given cancer type will be plotted in the same color, so that we can see to what extent the observations within a cancer type are similar to each other. We first create a simple function that assigns a distinct color to each element of a numeric vector. The function will be used to assign a color to each of the 64 cell lines, based on the cancer type to which it corresponds.

```r
Cols=function (vec){
   cols=rainbow (length(unique(vec)))
   return(cols[as.numeric (as.factor(vec))])
}
```

Note that the rainbow() function takes as its argument a positive integer and returns a vector containing that number of distinct colors. We now can plot the principal component score vectors.

```r
par(mfrow=c(1,2))
plot(pr.out$x[,1:2], col=Cols(nci.labs), pch=19, xlab="Z1",ylab="Z2")
plot(pr.out$x[,c(1,3)], col=Cols(nci.labs), pch=19, xlab="Z1",ylab="Z3")
```



On the whole, cell lines corresponding to a single cancer type do tend to have similar values on the first few principal component score vectors. This indicates that cell lines from the same cancer type tend to have pretty similar gene expression levels.

We can obtain a summary of the proportion of variance explained (PVE) of the first few principal components using the summary() method for a prcomp object:

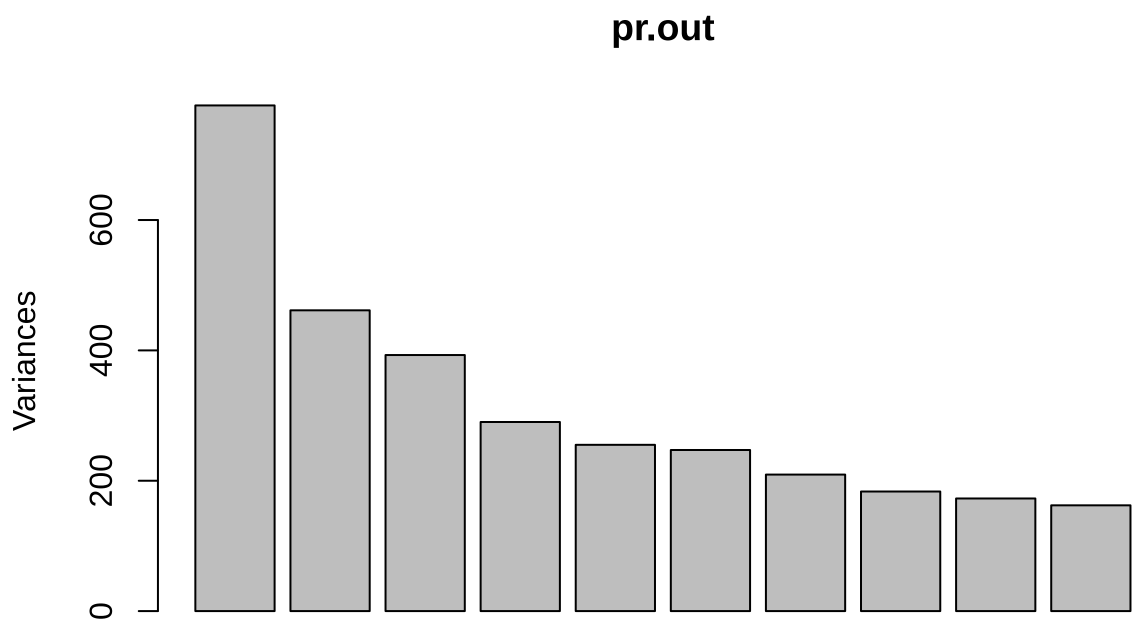
```
summary(pr.out)
```

```
## Importance of components:
##                            PC1      PC2      PC3      PC4      PC5
## Standard deviation     27.8535 21.48136 19.82046 17.03256 15.97181
## Proportion of Variance  0.1136  0.06756  0.05752  0.04248  0.03735
## Cumulative Proportion   0.1136  0.18115  0.23867  0.28115  0.31850
##                             PC6      PC7      PC8      PC9     PC10
## Standard deviation     15.72108 14.47145 13.54427 13.14400 12.73860
## Proportion of Variance  0.03619  0.03066  0.02686  0.02529  0.02376
## Cumulative Proportion   0.35468  0.38534  0.41220  0.43750  0.46126
##                            PC11     PC12     PC13     PC14     PC15
## Standard deviation     12.68672 12.15769 11.83019 11.62554 11.43779
## Proportion of Variance  0.02357  0.02164  0.02049  0.01979  0.01915
## Cumulative Proportion   0.48482  0.50646  0.52695  0.54674  0.56590
##                            PC16     PC17     PC18     PC19    PC20
## Standard deviation     11.00051 10.65666 10.48880 10.43518 10.3219
## Proportion of Variance  0.01772  0.01663  0.01611  0.01594  0.0156
## Cumulative Proportion   0.58361  0.60024  0.61635  0.63229  0.6479
```

```
##                             PC21    PC22    PC23    PC24    PC25    PC26
## Standard deviation     10.14608 10.0544 9.90265 9.64766 9.50764 9.33253
## Proportion of Variance  0.01507  0.0148 0.01436 0.01363 0.01324 0.01275
## Cumulative Proportion   0.66296  0.6778 0.69212 0.70575 0.71899 0.73174
##                            PC27   PC28    PC29    PC30    PC31    PC32
## Standard deviation     9.27320 9.0900 8.98117 8.75003 8.59962 8.44738
## Proportion of Variance 0.01259 0.0121 0.01181 0.01121 0.01083 0.01045
## Cumulative Proportion  0.74433 0.7564 0.76824 0.77945 0.79027 0.80072
##                            PC33    PC34    PC35    PC36    PC37    PC38
## Standard deviation     8.37305 8.21579 8.15731 7.97465 7.90446 7.82127
## Proportion of Variance 0.01026 0.00988 0.00974 0.00931 0.00915 0.00896
## Cumulative Proportion  0.81099 0.82087 0.83061 0.83992 0.84907 0.85803
##                            PC39    PC40    PC41   PC42    PC43   PC44
## Standard deviation     7.72156 7.58603 7.45619 7.3444 7.10449 7.0131
## Proportion of Variance 0.00873 0.00843 0.00814 0.0079 0.00739 0.0072
## Cumulative Proportion  0.86676 0.87518 0.88332 0.8912 0.89861 0.9058
##                            PC45   PC46    PC47    PC48    PC49    PC50
## Standard deviation     6.95839 6.8663 6.80744 6.64763 6.61607 6.40793
## Proportion of Variance 0.00709 0.0069 0.00678 0.00647 0.00641 0.00601
## Cumulative Proportion  0.91290 0.9198 0.92659 0.93306 0.93947 0.94548
##                            PC51    PC52    PC53    PC54    PC55    PC56
## Standard deviation     6.21984 6.20326 6.06706 5.91805 5.91233 5.73539
## Proportion of Variance 0.00566 0.00563 0.00539 0.00513 0.00512 0.00482
## Cumulative Proportion  0.95114 0.95678 0.96216 0.96729 0.97241 0.97723
##                            PC57   PC58    PC59    PC60    PC61    PC62
## Standard deviation     5.47261 5.2921 5.02117 4.68398 4.17567 4.08212
## Proportion of Variance 0.00438 0.0041 0.00369 0.00321 0.00255 0.00244
## Cumulative Proportion  0.98161 0.9857 0.98940 0.99262 0.99517 0.99761
##                            PC63      PC64
## Standard deviation     4.04124 2.148e-14
## Proportion of Variance 0.00239 0.000e+00
## Cumulative Proportion  1.00000 1.000e+00
```

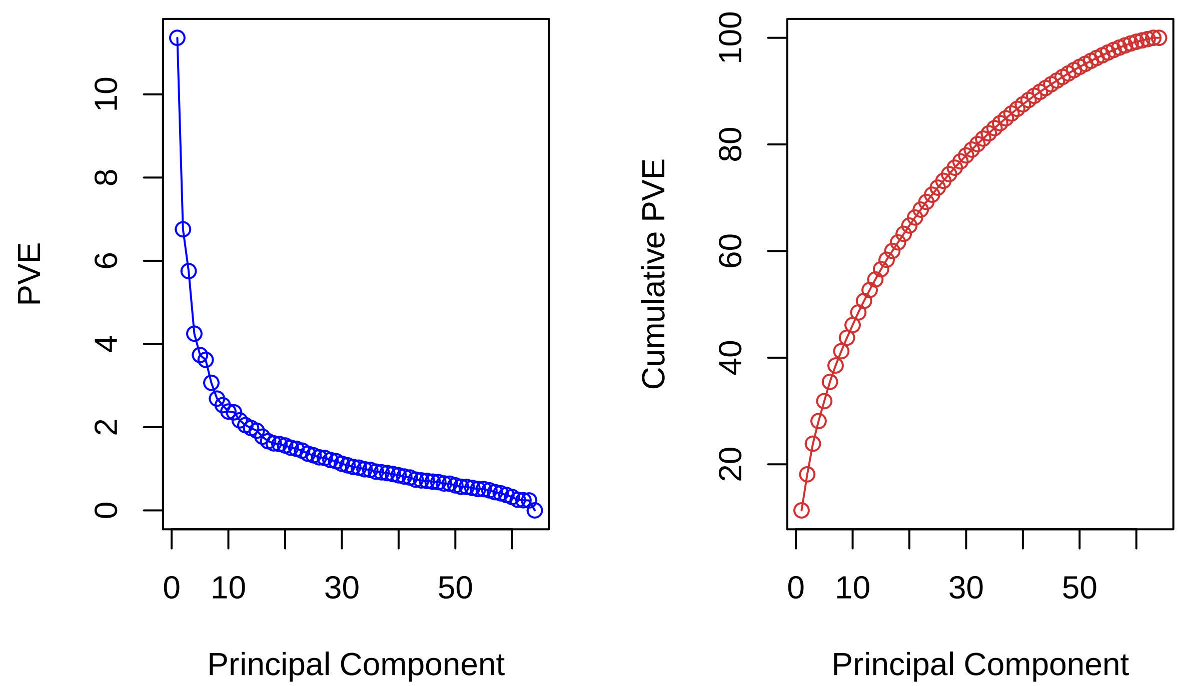
Using the plot() function, we can also plot the variance explained by the first few principal components.

```r
plot(pr.out)
```



Note that the height of each bar in the bar plot is given by squaring the corresponding element of pr.out$sdev. However, it is more informative to plot the PVE of each principal component (i.e. a scree plot) and the cumulative PVE of each principal component. This can be done with just a little work.

```
pve =100*pr.out$sdev^2/sum(pr.out$sdev^2)
par(mfrow=c(1,2))
plot(pve , type="o", ylab="PVE", xlab="Principal Component", col="blue")
plot(cumsum(pve), type="o", ylab="Cumulative PVE", xlab="Principal Component", col="brown3")
```



(Note that the elements of pve can also be computed directly from the summary, “summary(pr.out)$importance[2,]$””, $andtheelem$
The resulting plots are shown in Figure 10.16. We see that together, the first seven principal components explain around 40% of the variance in the data. This is not a huge amount of the variance. However, looking at the scree plot, we see that while each of the first seven principal components explain a substantial amount of variance, there is a marked decrease in the variance explained by further principal components. That is, there is an elbow in the plot after approximately the seventh principal component. This suggests that there may be little benefit to examining more than seven or so principal components (though even examining seven principal components may be difficult).

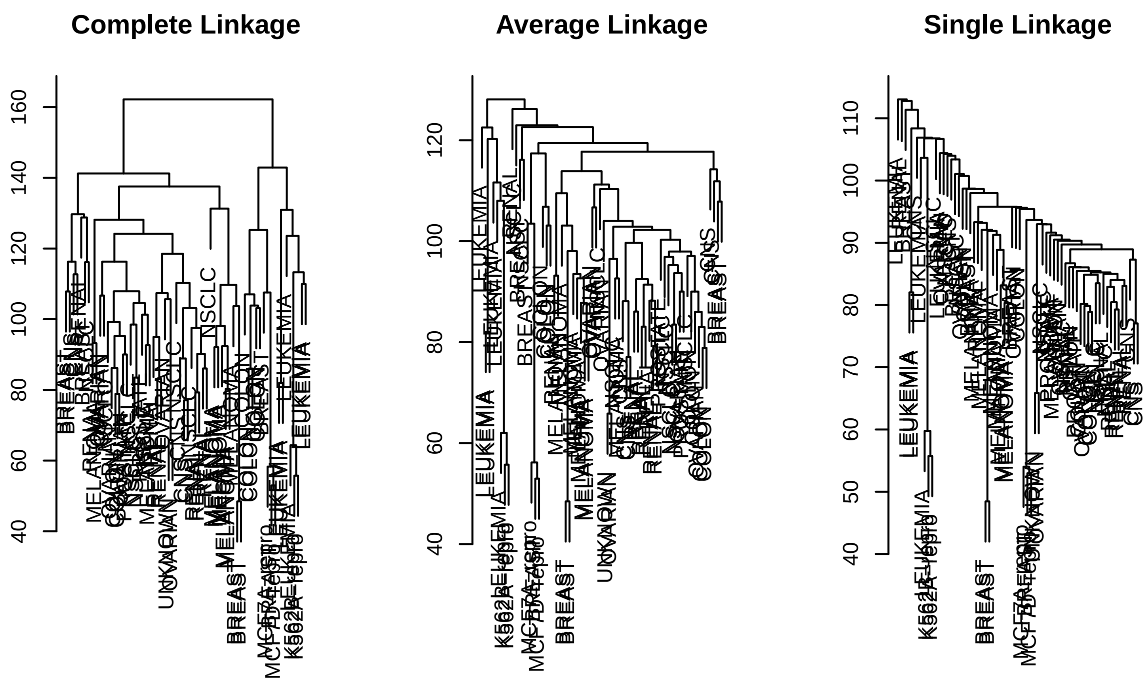
10.6.2 Clustering the Observations of the NCI60 Data

We now proceed to hierarchically cluster the cell lines in the NCI60 data, with the goal of finding out whether or not the observations cluster into distinct types of cancer. To begin, we standardize the variables to have mean zero and standard deviation one. As mentioned earlier, this step is optional and should be performed only if we want each gene to be on the same scale.

```
sd.data = scale(nci.data)
```

We now perform hierarchical clustering of the observations using complete, single, and average linkage. Euclidean distance is used as the dissimilarity measure.

```
par(mfrow=c(1,3))
data.dist=dist(sd.data)
plot(hclust(data.dist), labels = nci.labs, main = "Complete Linkage", xlab = "", sub = "",ylab = "")
plot(hclust(data.dist, method = "average"), labels = nci.labs, main = "Average Linkage", xlab = "", sub
plot(hclust(data.dist, method  = "single"), labels=nci.labs, main = "Single Linkage", xlab = "", sub =
```



We see that the choice of linkage certainly does affect the results obtained. Typically, single linkage will tend to yield trailing clusters: very large clusters onto which individual observations attach one-by-one. On the other hand, complete and average linkage tend to yield more balanced, attractive clusters. For this reason, complete and average linkage are generally preferred to single linkage. Clearly cell lines within a single cancer type do tend to cluster together, although the clustering is not perfect. We will use complete linkage hierarchical clustering for the analysis that follows.

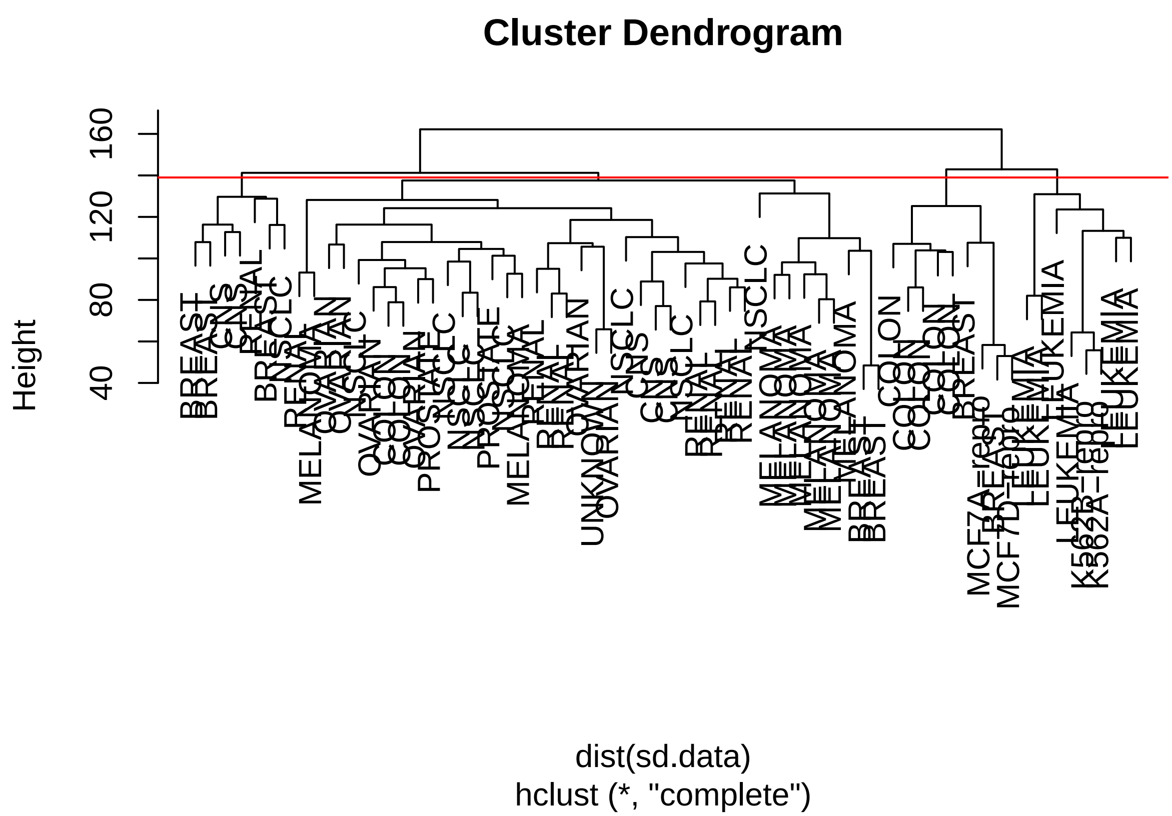
We can cut the dendrogram at the height that will yield a particular number of clusters, say four:

```
hc.out = hclust(dist(sd.data))
hc.clusters = cutree(hc.out, 4)
table(hc.clusters, nci.labs)
```

```
##            nci.labs
## hc.clusters BREAST CNS COLON K562A-repro K562B-repro LEUKEMIA MCF7A-repro
##           1      2   3     2           0           0        0           0
##           2      3   2     0           0           0        0           0
##           3      0   0     0           1           1        6           0
##           4      2   0     5           0           0        0           1
##            nci.labs
## hc.clusters MCF7D-repro MELANOMA NSCLC OVARIAN PROSTATE RENAL UNKNOWN
##           1           0        8     8       6        2     8       1
##           2           0        0     1       0        0     1       0
##           3           0        0     0       0        0     0       0
##           4           1        0     0       0        0     0       0
```

There are some clear patterns. All the leukemia cell lines fall in cluster 3, while the breast cancer cell lines are spread out over three different clusters. We can plot the cut on the dendrogram that produces these four clusters:

```r
par(mfrow=c(1,1))
plot(hc.out, labels = nci.labs)
abline(h=139, col = "red")
```



The abline() function draws a straight line on top of any existing plot in R. The argument h=139 plots a horizontal line at height 139 on the dendrogram; this is the height that results in four distinct clusters. It is easy to verify that the resulting clusters are the same as the ones we obtained using cutree(hc.out,4).

Printing the output of hclust gives a useful brief summary of the object:

```r
hc.out
```

```
##
## Call:
## hclust(d = dist(sd.data))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 64
```

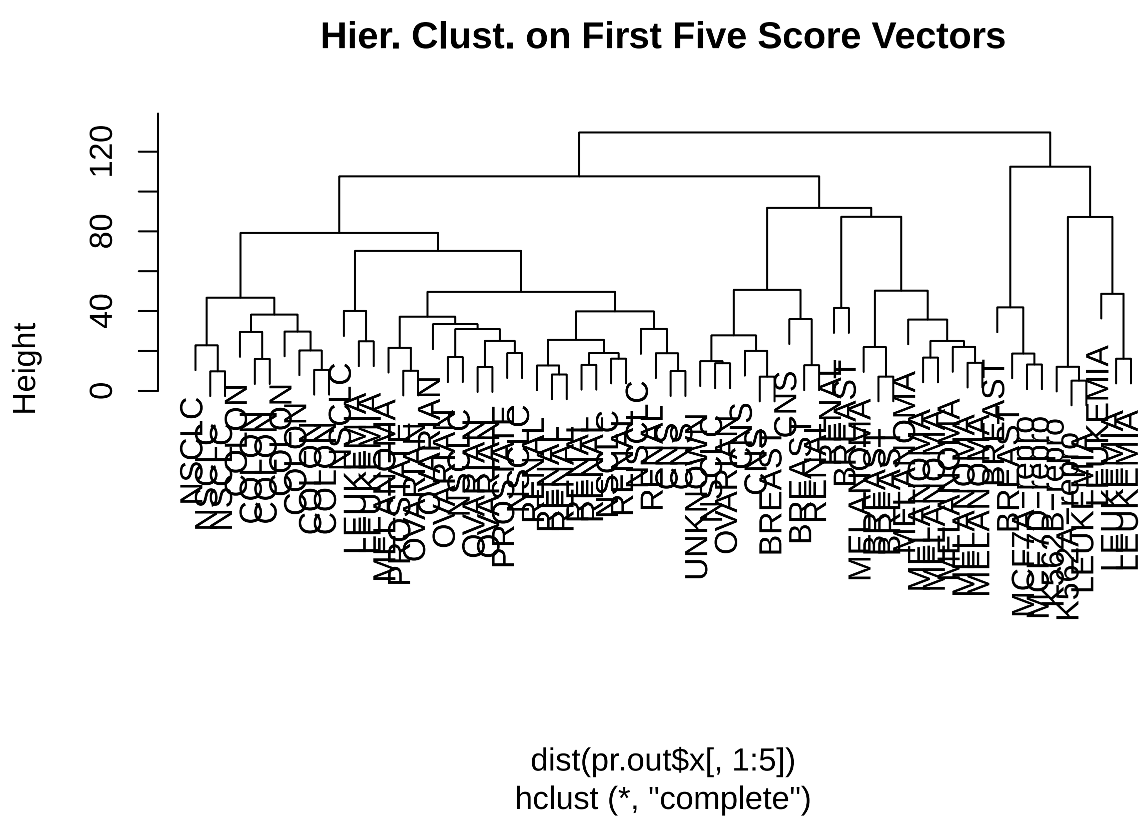
We claimed earlier that K-means clustering and hierarchical clustering with the dendrogram cut to obtain the same number of clusters can yield very different results. How do these NCI60 hierarchical clustering results compare to what we get if we perform K-means clustering with K = 4?

```r
set.seed(2)
km.out = kmeans(sd.data, 4, nstart = 20)
km.clusters = km.out$cluster
table(km.clusters, hc.clusters)
```

```
##            hc.clusters
## km.clusters  1  2  3  4
##           1 11  0  0  9
##           2  0  0  8  0
##           3  9  0  0  0
##           4 20  7  0  0
```

We see that the four clusters obtained using hierarchical clustering and Kmeans clustering are somewhat different. Cluster 2 in K-means clustering is identical to cluster 3 in hierarchical clustering. However, the other clusters differ: for instance, cluster 4 in K-means clustering contains a portion of the observations assigned to cluster 1 by hierarchical clustering, as well as all of the observations assigned to cluster 2 by hierarchical clustering. Rather than performing hierarchical clustering on the entire data matrix, we can simply perform hierarchical clustering on the first few principal component score vectors, as follows:

```r
hc.out=hclust(dist(pr.out$x[,1:5]) )
plot(hc.out, labels = nci.labs, main="Hier. Clust. on First Five Score Vectors")
```




```r
table(cutree(hc.out, 4), nci.labs)
```

```
##    nci.labs
##     BREAST CNS COLON K562A-repro K562B-repro LEUKEMIA MCF7A-repro
##   1      0   2     7           0           0        2           0
##   2      5   3     0           0           0        0           0
##   3      0   0     0           1           1        4           0
##   4      2   0     0           0           0        0           1
##    nci.labs
##     MCF7D-repro MELANOMA NSCLC OVARIAN PROSTATE RENAL UNKNOWN
##   1           0        1     8       5        2     7       0
##   2           0        7     1       1        0     2       1
##   3           0        0     0       0        0     0       0
##   4           1        0     0       0        0     0       0
```

Not surprisingly, these results are different from the ones that we obtained when we performed hierarchical clustering on the full data set. Sometimes performing can give better results than performing clustering on the full data. In this situation, we might view the principal component step as one of denoising the data. We could also perform K-means clustering on the first few principal component score vectors rather than the full data set.