Survival Analysis and Censored Data Exercises

Simone Collier

Principal Components Analysis

We will be making use of the built in `iris` data set. If you are unfamiliar with it then take a look using `?iris`. We will remove the `species` column since we don't want to deal with a categorical variable at this point. Let's examine the data.

```
iris <- iris[, -5]
apply(iris, 2, mean)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

```
apply(iris, 2, var)
```

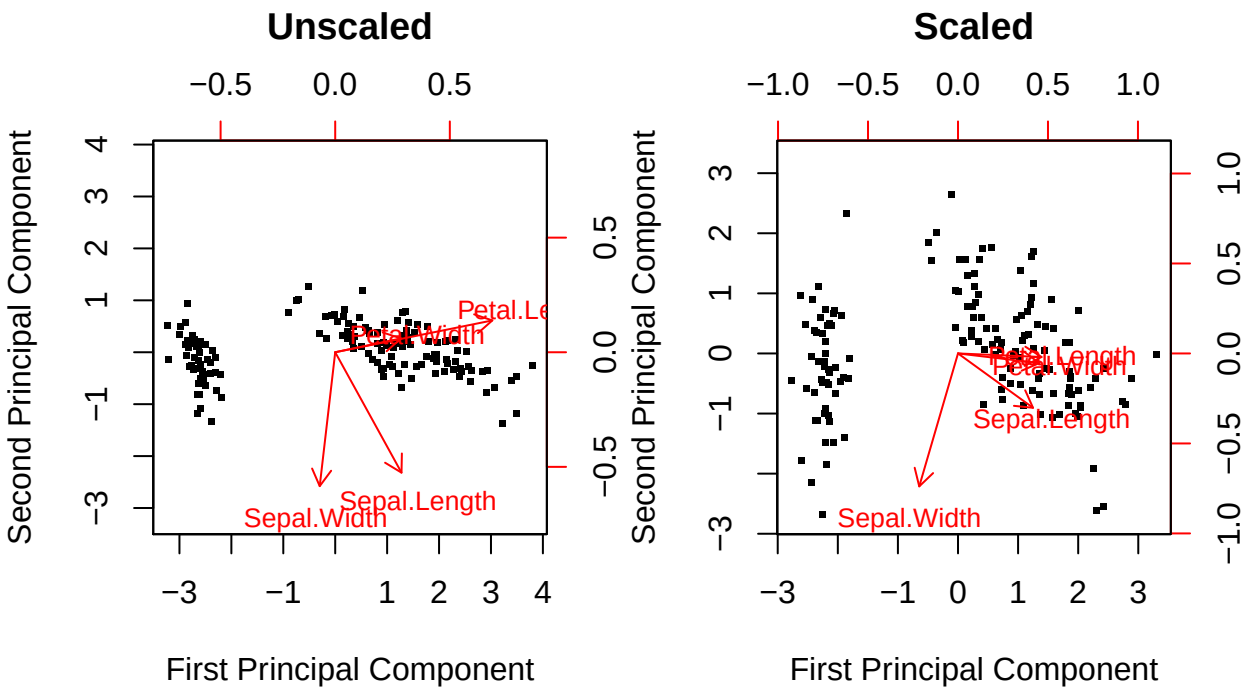
```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.6856935    0.1899794    3.1162779    0.5810063
```

The mean and variance for each feature in the data set are not too different so scaling in this case won't make a huge difference. Let's see what would happen if we performed principal component analysis with and without scaling the features. We will use the function `prcomp()` which is part of the base R package and the argument `scale`.

```
par(mfrow=c(1,2), mar = c(4,4,5,2))

iris.pr <- prcomp(iris)
biplot(iris.pr, scale = 0, xlabs=rep(".", nrow(iris)), cex = c(2,0.8), main = "Unscaled",
       xlab = "First Principal Component", ylab = "Second Principal Component")

iris.pr <- prcomp(iris, scale = TRUE)
biplot(iris.pr, scale = 0, xlabs=rep(".", nrow(iris)), cex = c(2,0.8), main = "Scaled",
       xlab = "First Principal Component", ylab = "Second Principal Component")
```



As we can see the data is a bit more spread out and a little easier to see pattern when the variables are scaled. So, we will stick with the scaled variables.

Let's look more closely at the output of `prcomp()`.

```
names(iris.pr)
```

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

- `sdev`: the standard deviation of each principal component.
- `rotation`: the principal component loading vector.
- `center`: the means of the variables prior to implementing PCA.
- `scale`: the standard deviations of the variables prior to implementing PCA.
- `x`: A matrix where the columns are the principal component scores $Z_1, Z_2, \ldots$.

```
iris.pr$rotation
```

```
##                    PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

The variance explained by each principal component is the square of the standard deviation. Thus, the proportion of variance explained by each principal component is the variance for each component divided by the total variance.

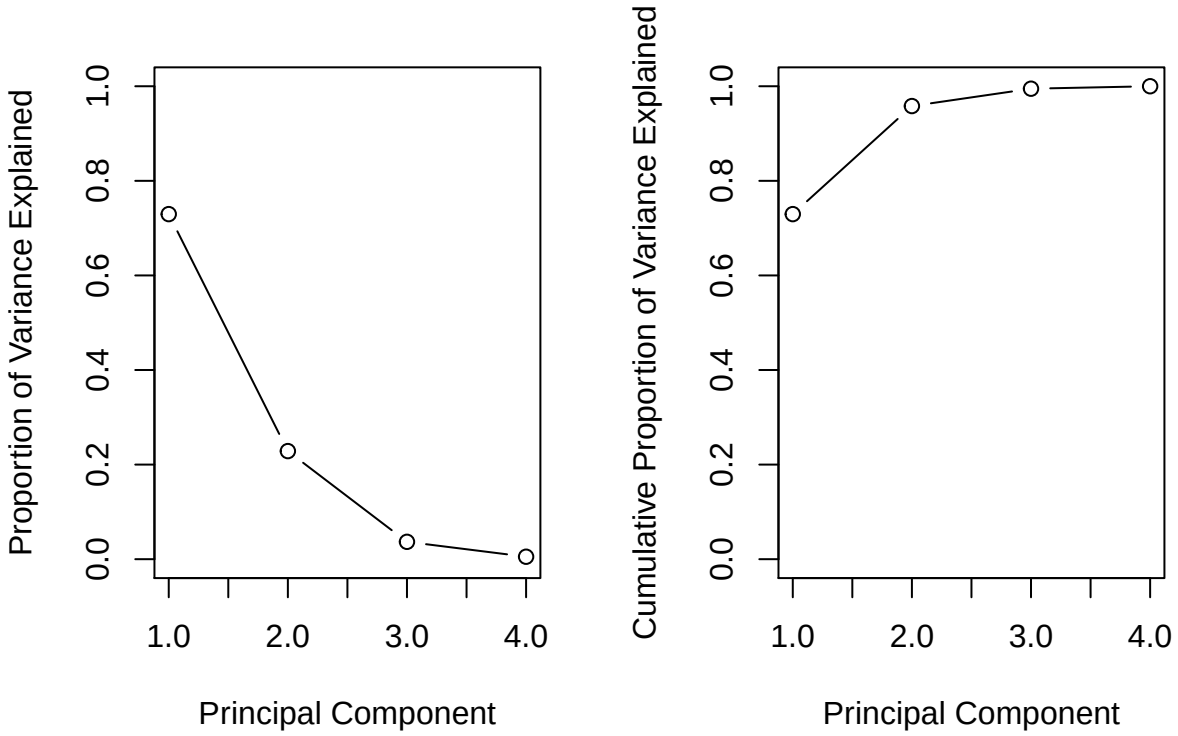
```
pr.var <- iris.pr$sdev^2
pve <- pr.var / sum(pr.var)
pve
```

```
## [1] 0.729624454 0.228507618 0.036689219 0.005178709
```

So the first principal component explains 73% of the variance in the data and the second principal component explains 23% of the variance. Together the first two principal components explain 96% of the variance!

We can plot the PVE obtained by each component to get the scree plot. We can also plot the cumulative PVE which is computed using the `cumsum()` function.

```
par(mfrow = c(1, 2))
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(0, 1), type = "b")
```



Matrix Completion

We will turn our `iris` data into a matrix, and center and scale it to have mean zero and variance one. Now we will omit 40 random entries from the 150×4 matrix. We will ensure that at most 1 observation is missing from each row.

```
X <- data.matrix(scale(iris))
nomit <- 40
set.seed(15)
```

```r
# sample 20 row indices
ind.r <- sample(1:nrow(iris), nomit)
# sample 20 column indices
ind.c <- sample(1:4, nomit, replace = TRUE)
# create a new matrix Xna which is X but with the entries [ind.r, ind.c] omitted
Xna <- X
index.na <- cbind(ind.r, ind.c)
Xna[index.na] <- NA
```

Now let's try to fill in the missing values using the mean of the column where the value is missing.

```r
Xmean <- Xna
for (i in 1:nomit) {
  Xmean[ind.r[i], ind.c[i]] <- mean(Xmean[, ind.c[i]], na.rm = TRUE)
}
```

We can see how well our method of filling in missing values with the corresponding variable mean works by computing the correlation between our imputed values and the true values. Recall that correlation is a number between -1 and 1.

- -1 means the variables are perfectly negatively correlated (they move in opposite directions)
- 0 mean no correlation.
- 1 mean perfect positive correlation.

So, we want to get as close to 1 as possible.

```r
cor(Xmean[index.na], X[index.na])
```

```
## [1] -0.3386279
```

Our correlation is below zero so this method for imputing missing values is clearly not cutting it.

Let's see if we can do any better using matrix completion. We will use the `softImpute` library. The `softImpute()` function performs matrix completion using the principal component method if we set the argument `type = svd`. Then we use the information we got from the `softImpute()` function to fill in the missing values using the `complete()` function.

```r
library(softImpute)
```

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

```
## Loaded softImpute 1.4-1
```

```r
Ximp <- softImpute(Xna, type = 'svd')
Xcomp <- complete(Xna, Ximp)
```

We can compute the correlation between our imputed matrix and the original data once again.

```
cor(Xcomp[index.na], X[index.na])
```

```
## [1] 0.8877741
```

A correlation of 0.89 is certainly a lot higher than what we saw from the column mean method.

In the real world we would not be able to check whether our imputed values are close to the true values but this shows just how useful principal components can be!

K-Means Clustering

2-D data

We will simulate a data set of 50 observations on two features derived from two different classes. The first 25 observations have a different mean for both of the features compared to the last 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 can perform K-means clustering on this data set to see if it picked up on the pattern we introduced into the data. We choose `centers = 2` to specify we want 2 clusters and we choose `nstart = 20` to specify we want to run the K-means algorithm 20 times wtih different starting values and output the best clustering.

```
set.seed(4)
km2 <- kmeans(x, centers = 2, nstart = 20)
```

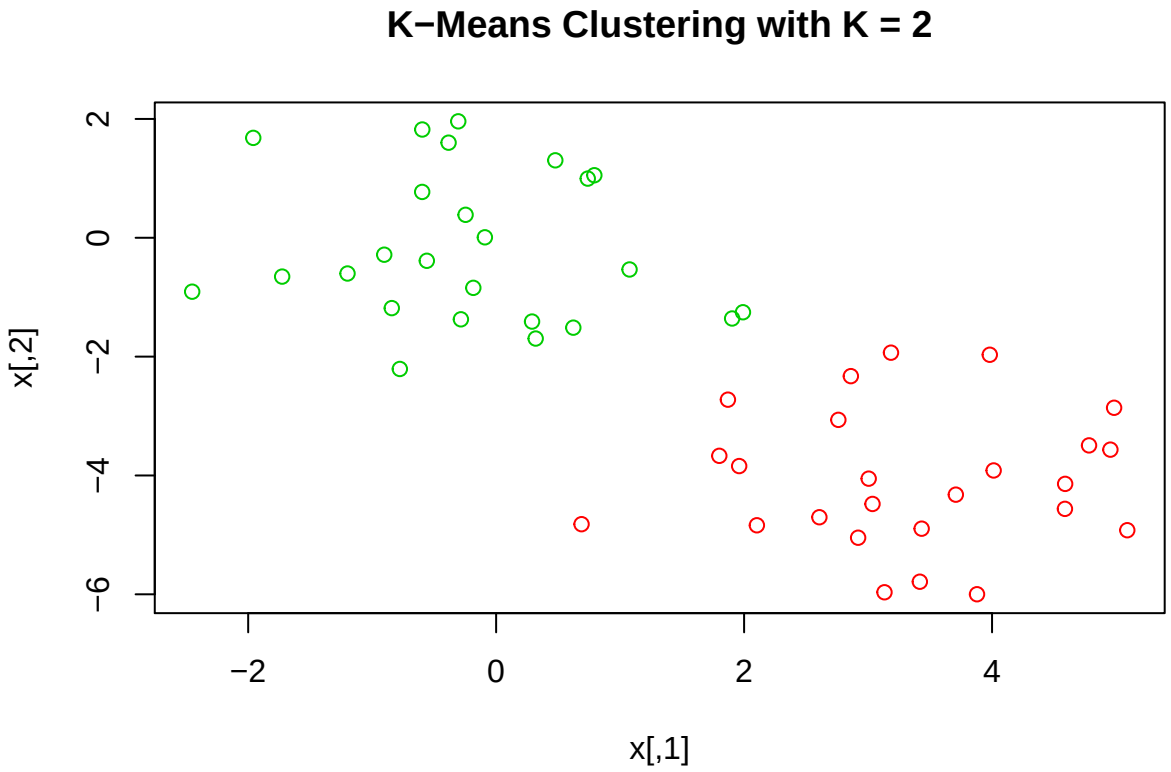
We can find the cluster assignments using:

```
km2$cluster
```

```
##  [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 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2
```

So the clustering method perfectly clustered the observation according to the two classes we made. We can plot the data and color each observation according to the cluster assignment.

```
plot(x, col = (km2$cluster + 1),
     main = "K-Means Clustering with K = 2")
```

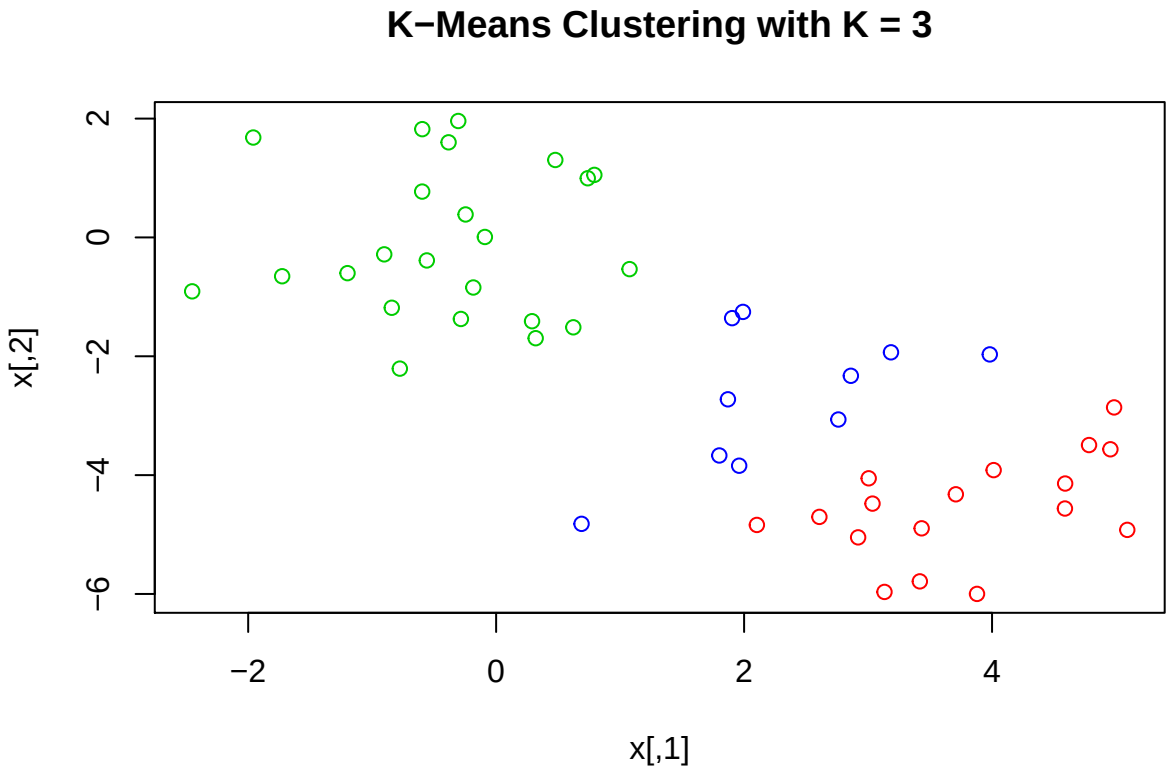


We knew there were 2 groups in this dataset but in real life we would not. So, let's see how K-means does with K = 3 clusters.

```
set.seed(4)
km3 <- kmeans(x, 3, nstart = 20)
km3
```

```
## K-means clustering with 3 clusters of sizes 17, 23, 10
##
## Cluster means:
##         [,1]        [,2]
## 1  3.7789567 -4.56200798
## 2 -0.3820397 -0.08740753
## 3  2.3001545 -2.69622023
##
## Clustering vector:
##  [1] 1 3 1 3 1 1 1 3 1 3 1 3 1 3 1 3 1 1 1 1 1 3 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 3 2 3 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 25.74089 52.67700 19.56137
##  (between_SS / total_SS =  79.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
plot(x, col = (km3$cluster + 1),
     main = "K-Means Clustering with K = 3")
```



We can find the total within-cluster sum of squares from `km3$tot.withinss`. This is the value that `kmeans()` seeks to minimize and it outputs the model from the `nstart` runs with the lowest `tot.withinss`.

```
km3$tot.withinss
```

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

iris data

We will now try clustering on the `iris` data set.

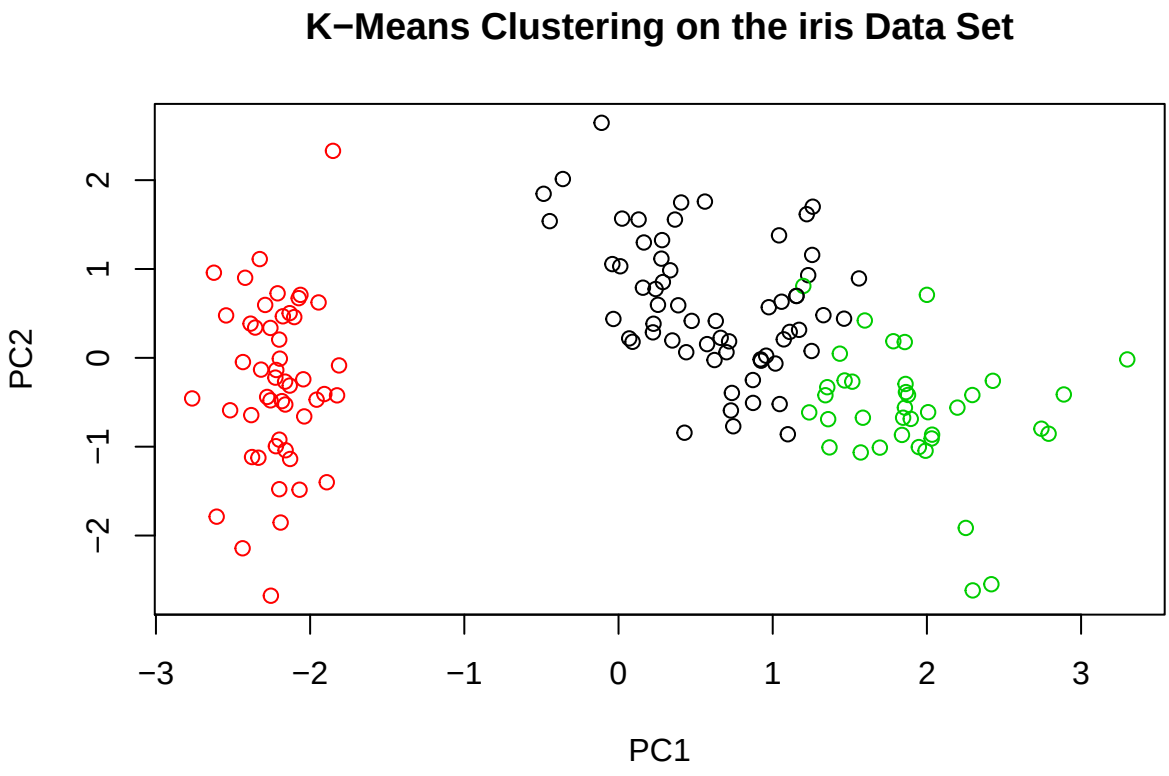
```
set.seed(2)
iris.km <- kmeans(iris, 3, nstart = 40)
iris.km
```

```
## K-means clustering with 3 clusters of sizes 62, 50, 38
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.901613    2.748387     4.393548    1.433871
## 2     5.006000    3.428000     1.462000    0.246000
## 3     6.850000    3.073684     5.742105    2.071053
##
## Clustering vector:
```

```
##   [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 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3 3 3 3 1 3 3 3 3
## [112] 3 3 1 1 3 3 3 3 1 3 1 3 1 3 3 1 1 3 3 3 3 3 1 3 3 3 3 1 3 3 3 1 3 3 3 1 3
## [149] 3 1
##
## Within cluster sum of squares by cluster:
## [1] 39.82097 15.15100 23.87947
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Since there are 4 variables used to cluster the data visualizing the clusters isn't as simple as the previous example. Luckily we have PCA! We can plot the first two principal components we have already computed and then colour code them according to the clusters.

```
iris2PC <- iris.pr$x[, c(1, 2)]
plot(iris2PC, col = (iris.km$cluster),
     main = "K-Means Clustering on the iris Data Set")
```



Hierarchical Clustering

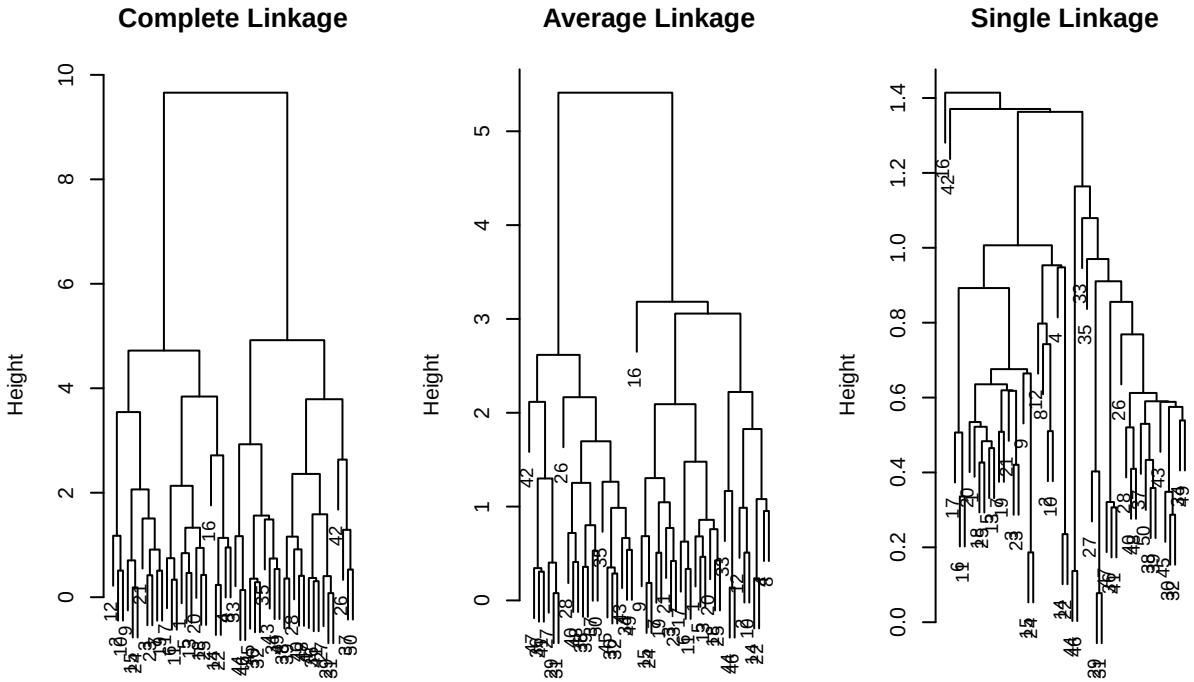
We will perform hierarchical clustering using the `hclaust()` function on the same data we generated in the K-means clustering section. We use the `dist()` function to compute the inter-observation Euclidean distance matrix.

We will fit with complete, average, and single linkage.

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

Now we plot the dendrograms we obtained.

```
par(mfrow = c(1, 3))
plot(hc.complete, main = "Complete Linkage", xlab = "", sub = "", cex = 0.9)
plot(hc.average, main = "Average Linkage", xlab = "", sub = "", cex = 0.9)
plot(hc.single, main = "Single Linkage", xlab = "", sub = "", cex = 0.9)
```



Now that we have the dendrograms we can cluster the observations using `cutree()`. The argument of `cutree()` is the number of clusters we want.

```
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 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2
```

```
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 2 2 2
## [39] 2 2 2 2 2 1 2 1 2 2 2 2
```

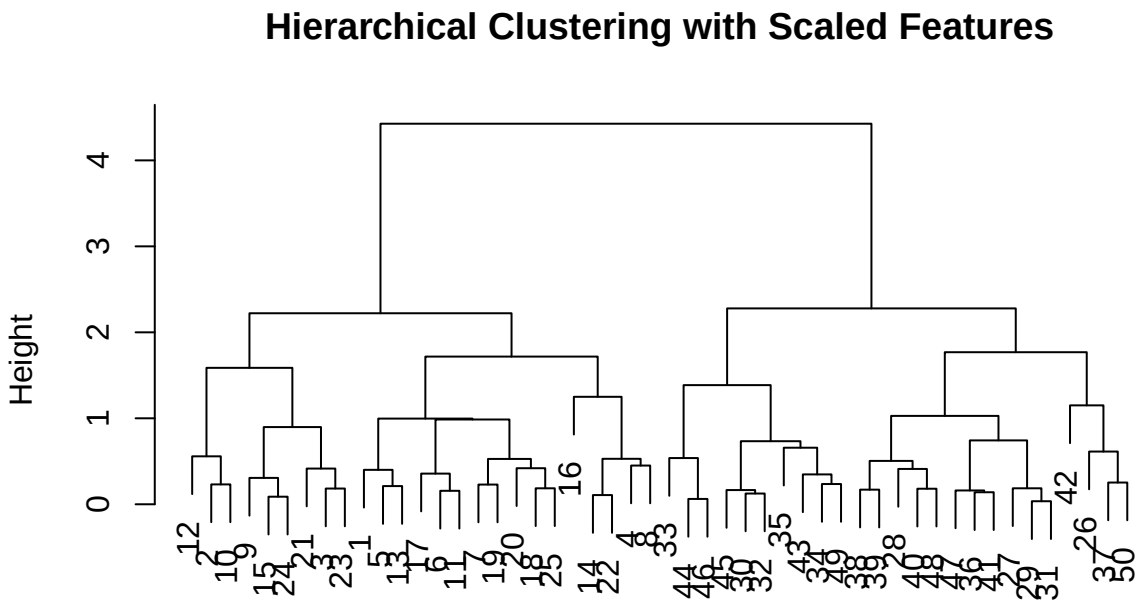
```
cutree(hc.single, 2)
```

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

Complete and average linkage correctly clusters the observations but single linkage identifies one point as being its own cluster.

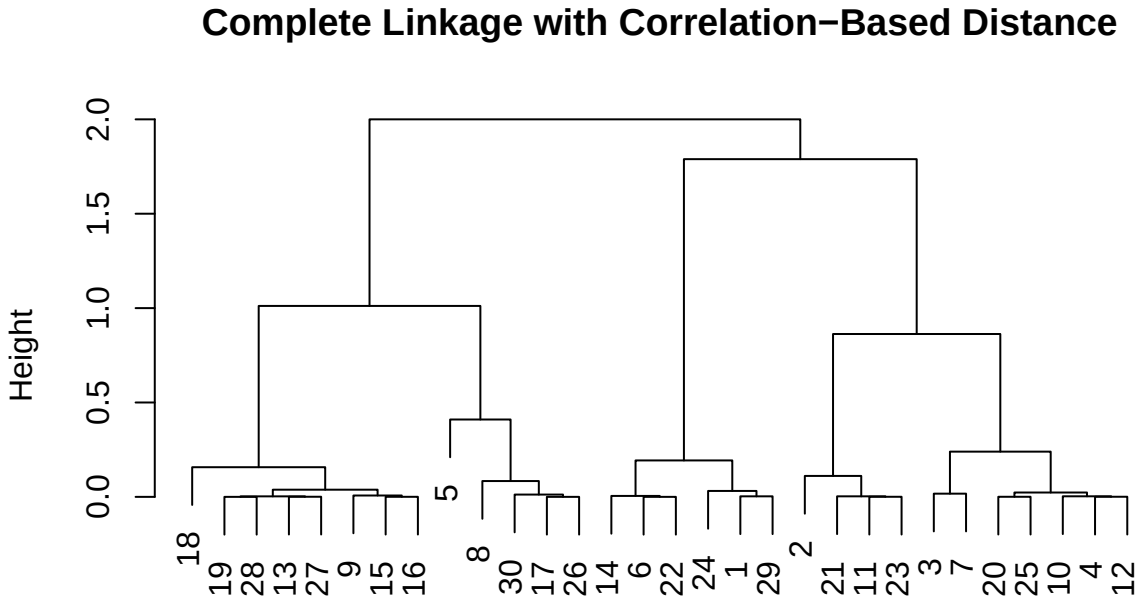
To scale the variables before performing hierarchical clustering we use the `scale()` function.

```
xsc <- scale(x)
plot(hclust(dist(xsc), method = "complete"),
     main = "Hierarchical Clustering with Scaled Features", xlab = "", sub = "")
```



Correlation-based distance can be used instead of Euclidean distance. This only makes sense on data with at least 3 features since the correlation between any two observations with two features is always 1. So, let's generate a new data set but this time we won't include any true clusters.

```
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 = "", sub = "")
```



These exercises were adapted from : James, Gareth, et al. An Introduction to Statistical Learning: with Applications in R, 2nd ed., Springer, 2021.