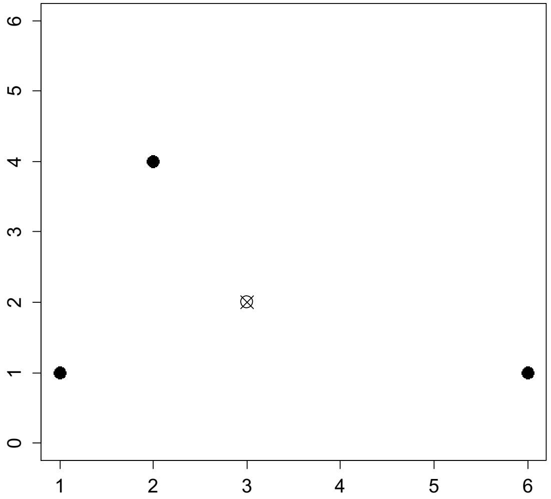
K-Means Clustering

Centroids

For a set of points in G^p , the ***centroid*** of the collection is the point in G^p whose components are the average of the components of the points in the collection. For example, for the three points (1, 1), (2, 4), and (6, 1) in R^2, the centroid is the point

$$\left(\frac{1+2+6}{3}, \frac{1+4+1}{3}\right) = \left(\frac{9}{3}, \frac{6}{3}\right) = (3, 2).$$



Fact: This point minimizes the sum of squared Euclidean distances between itself and each point in the set.

Find the centroid of the four points (in R^4) (2, 0.5, −3, 4), (1, 0, 3, −2), (0, 3, 1, 1), and (1, 2.5, 3, 3)

K-Means Clustering

Another popular method for forming clusters is *K*-means clustering

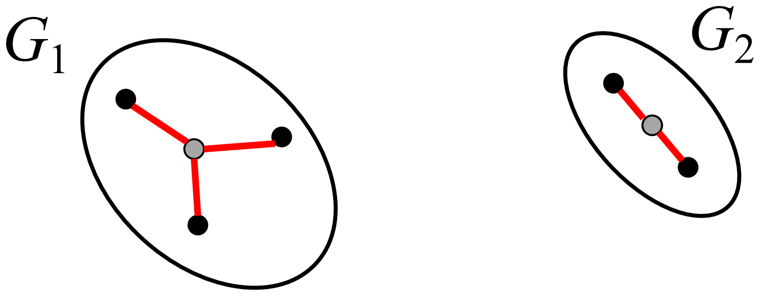
- The researcher specifies a number of clusters, *K*, and the units are split into *K* groups based on the optimization of an objective function.
- A hierarchy is not produced by the process and hence there is no associated dendrogram for visualization.
- There are several closely related algorithms which could be classified as *K*-means algorithms
- Many methods incorporate stochastic (random) elements into the process and hence may arrive at different end groupings when repeated on the same data. Care needs to be taken to understand the stability and reproducibility of the solutions.

K-Means Clustering

Let $G_1, G_2, \ldots, G_K$ be a partition of the units into K groups. Let $\bar{x}_1, \ldots, \bar{x}_K$ be the centroids of these K groups. Then the **within group sum of squares** (WGSS) is defined to be

$$\text{WGSS} = \sum_{k=1}^{K} \sum_{i \in G_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$$

This is the sum of the squared Euclidean distances of the units to the centroid of the cluster they are assigned to.



In the figure above, WGSS would be the sum of the squared lengths of (all) the red lines in the figure, having a total of five terms.

K-Means Clustering

Let $\mathscr{P}_K$ denote the set of all possible partitions of the units into *K* groups. The goal is to minimize the WGSS over $\mathscr{P}_K$. That is, it is the solution to the problem

$$W_K = \min_{\{G_1,\ldots,G_K\} \in P_K} (\text{WGSS})$$

Unfortunately, this problem is NP-hard and can only be solved for small values of *n* and *K*. We have seen that the number of possible partitions of *n* objects into *K* groups grows **very** fast with *n* and *K,* in fact, if $n = 50$ and $K = 4$ then there are approximately 5.3×10^{28} possible partitions!

Computer implementations use different heuristics to find a good solution that, while not guaranteeing the absolute minimum, work well in practice and are computationally feasible.

K-Means Clustering

The `kmeans` function is part of base R and will perform *K*-means clustering. The syntax is

`kmeans(x, centers, … )` where

x numeric matrix of data, or an object that can be coerced to such a matrix (such as a data frame with all numeric columns).

`centers` either the number of clusters, say K, or a set of initial (distinct) cluster centers. If a number, a random set of (distinct) rows in x is chosen as the initial centers.

The default method proceeds as follows: After the initialization of the cluster centroids, each unit is assigned to the cluster containing the centroid it is closest to. The centroids are then recalculated. Next, a *greedy* strategy is followed where units in clusters whose centroids changed may be reassigned to other clusters if this improves the objective. Centroids are then recalculated, and this is repeated until no further changes result (or a max number of iterations is achieved).

`kmeans` Function in R

`kmeans` returns an object of class "kmeans" which has a print and a fitted method. It is a list with (among others) the following components:

cluster	A vector of integers (from 1:k) indicating the cluster to which each point is allocated.
centers	A matrix of cluster centers (centroids).
totss	The total sum of squares.
withinss	Vector of within-cluster sum of squares, one component per cluster.
tot.withinss	Total within-cluster sum of squares, i.e. sum(withinss).
betweenss	The between-cluster sum of squares, i.e. totss - tot.withinss.
size	The number of points in each cluster.

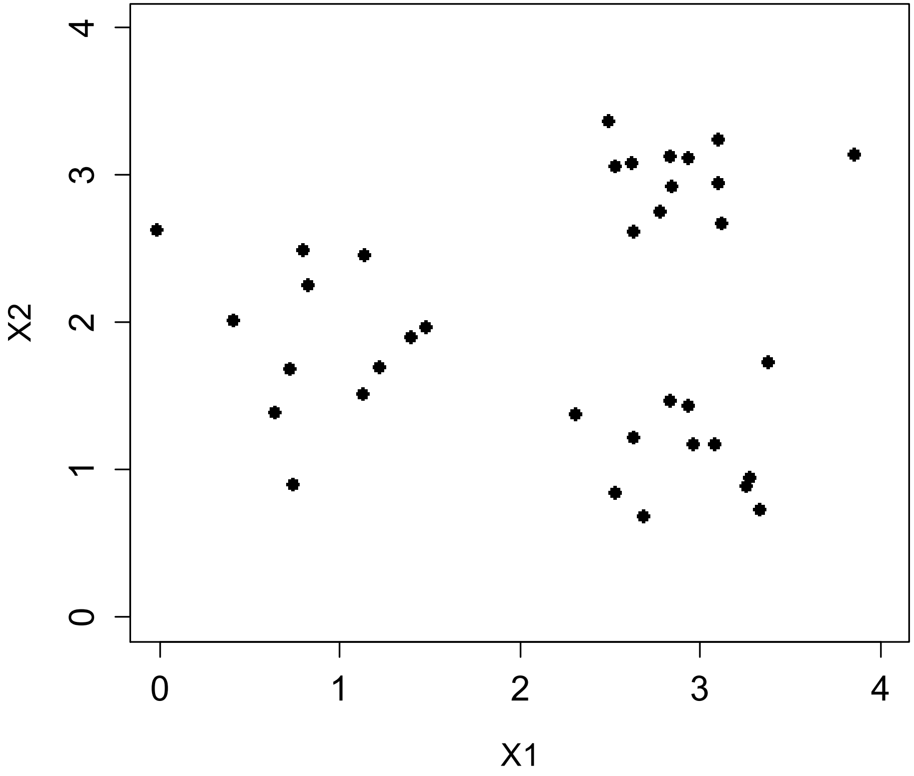
The algorithm of Hartigan and Wong (1979) is used by default.

kmeans Function in R

To demonstrate the use of the kmeans function, we again examine the toy dataset Cluster_Ex, which has 36 units in it, each with two measurements: X1 and X2.

```
> head(Cluster_Ex)
              X1       X2
Unit 1 0.4102665 2.010689
Unit 2 3.0825080 1.175286
Unit 3 1.4751248 1.969292
Unit 4 2.6245358 3.080116
Unit 5 3.3782067 1.736558
Unit 6 2.4948872 3.361988
```

```
> Kmeans_3<-kmeans(Cluster_Ex,centers=3)
> Kmeans_3$centers
        X1       X2
1 0.872247 1.909779
2 2.933556 1.142249
3 2.902353 3.004285
> Kmeans_3$withinss
[1] 4.832921 2.492320 2.082534
```



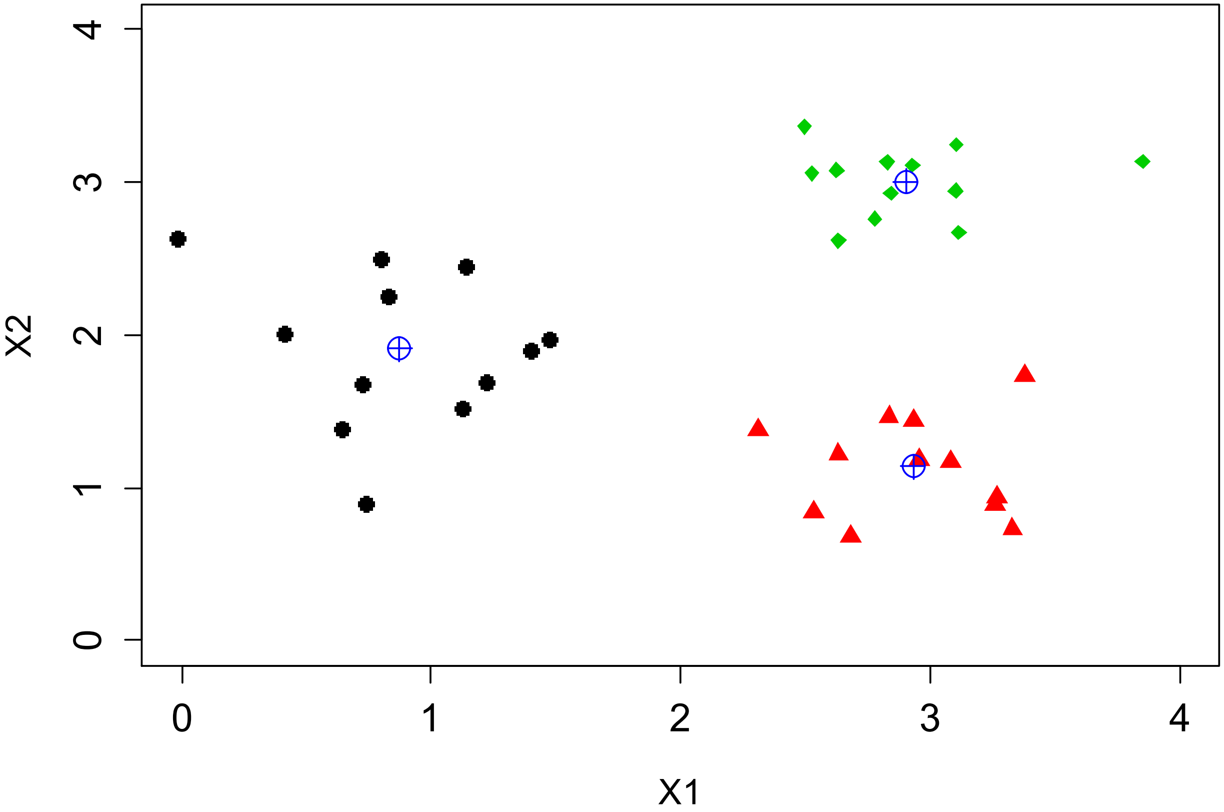
```
> Kmeans_3$cluster
 Unit 1  Unit 2  Unit 3  Unit 4  Unit 5  Unit 6  Unit 7  Unit 8
      1       2       1       3       2       3       3       2
```

`kmeans` Function in R

Three group *K*-means solution for clustering plotted with group centroids (blue crossed circles).

```
> plot(X2~X1,data=Cluster_Ex,xlim=c(0,4),ylim=c(0,4),cex.axis=1.3,
+ cex.lab=1.2,cex=1.2,pch=15+Kmeans_3$cluster,col=Kmeans_3$cluster)
> points(X2~X1,data=Kmeans_3$centers,pch=10,cex=1.8,col="blue")
```

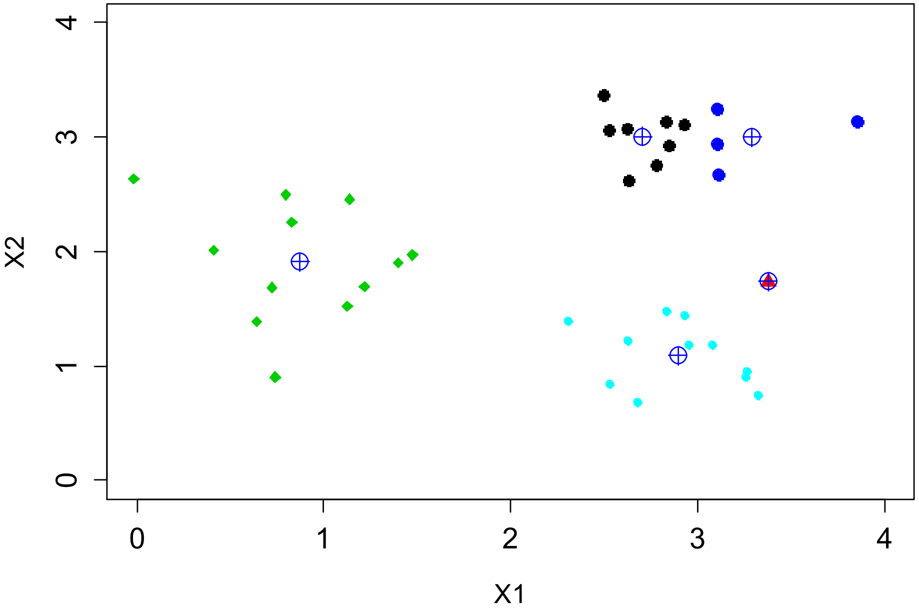
```
> palette()
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"  "gray"
```

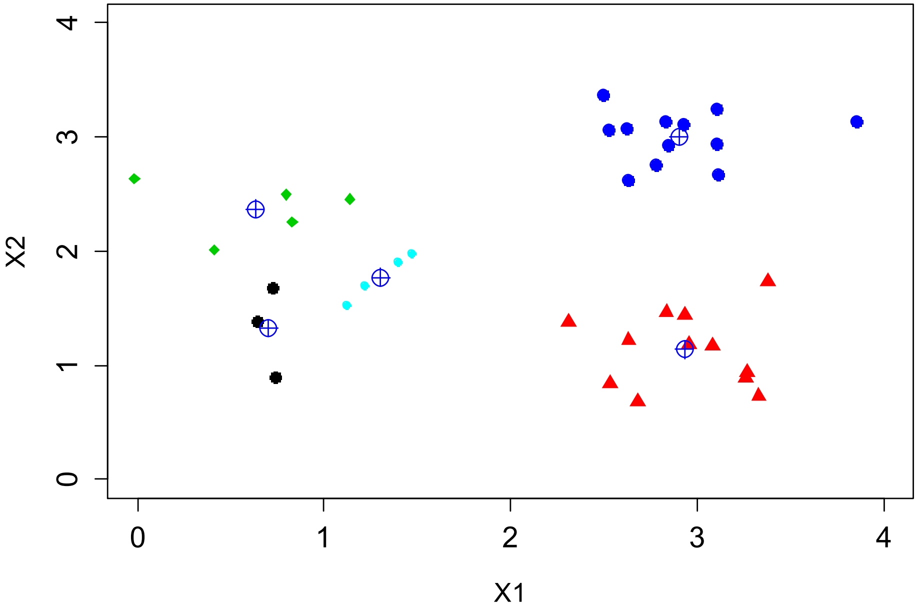


`kmeans` Function in R

We now use the `kmeans` function on the data to produce two $K = 5$ cluster solutions and then plot them.

```
> Kmeans_5_1<-kmeans(Cluster_Ex,centers=5)
> Kmeans_5_2<-kmeans(Cluster_Ex,centers=5)
```





What do we notice? Why did this happen?

By default, the `kmeans` function *randomly* selects 5 rows from the data matrix for the initial cluster centers. The algorithm can then converge to different solutions.

`kmeans` Function in R

We can obtain a reproducible result by specifying the initial centroids or by using the `set.seed` function in R prior to using the `kmeans` function.

```
> Kmeans_5_3<-kmeans(Cluster_Ex,centers=Cluster_Ex[c(2,5,7,21,36),])
> set.seed(42)
> Kmeans_5_4<-kmeans(Cluster_Ex,centers=5)
```

In the first example, we selected rows 2, 5, 7, 21, and 36 from the data matrix to be the initial cluster centroids. Choosing these same rows again would produce the same final result.

In the second example, we used the `set.seed` function in R (any integer as input will do) to set the state of the random number generator. If we set it to the same value (immediately) before running `kmeans` again, we would obtain the same clustering.

More on the `kmeans` Function in R

As we have seen, the `kmeans` function (by default) initializes the algorithm by randomly selecting K rows from the data matrix **X**.

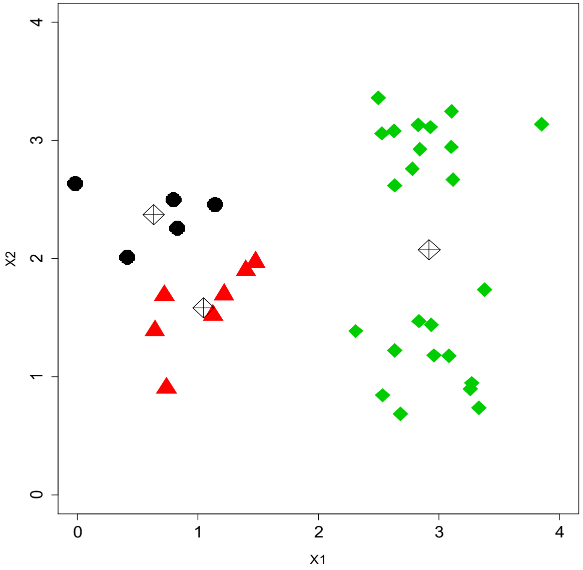
We can obtain a reproducible result by specifying the initial centroids (e.g. rows in the data matrix) or by using the `set.seed` function in R (immediately) prior to using the `kmeans` function.

Another common way to approach the uncertainty in a k-means clustering is to *repeat* the process with several random starts and retain the best solution (the one that minimizes the total WGSS).

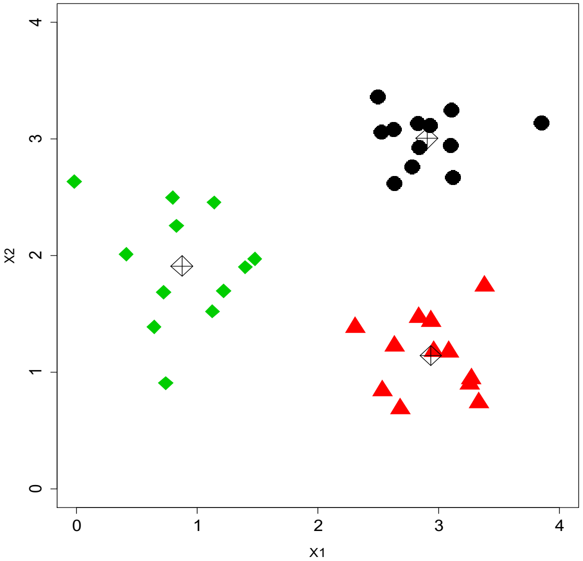
This can be accomplished using the `nstart` optional argument in the `kmeans` function. This value tells the function how many random starts you would like it to complete before reporting the best result. The default is set to `nstart=1`.

More on the `kmeans` Function in R

```
> Cluster3_S1<-kmeans(Cluster_Ex,centers=3)
> Cluster3_S50<-kmeans(Cluster_Ex,centers=3,nstart=50)

> Cluster3_S1
K-means clustering with 3 clusters of sizes 5, 7, 24

Cluster means:
         X1       X2
1 0.6305623 2.370460
2 1.0448789 1.580722
3 2.9179548 2.073267

> Cluster3_S1$withinss
[1]  1.030018  1.483150 25.383755
> Cluster3_S1$tot.withinss
[1] 27.89692
```



```
> Cluster3_S50
K-means clustering with 3 clusters of sizes 12, 12, 12

Cluster means:
        X1       X2
1 2.902353 3.004285
2 2.933556 1.142249
3 0.872247 1.909779
> Cluster3_S50$withinss
[1] 2.082534 2.492320 4.832921
> Cluster3_S50$tot.withinss
[1] 9.407774
```



How Many Clusters?

One of the challenges of K-means clustering is the decision on the number of clusters.

Because no hierarchy is created and the algorithm produces the final clustering directly from the value K, it can be challenging to decide on the correct number of clusters.

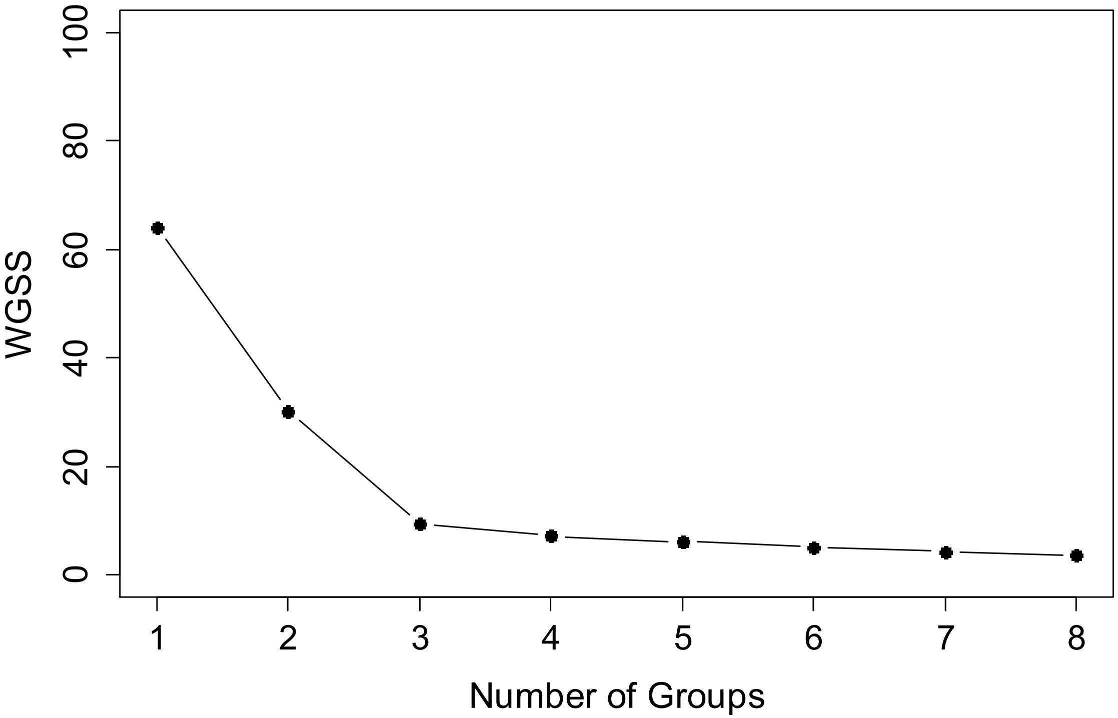
Moreover, as the number of clusters increases, the value of W_K tends to decrease.

The following code (a small `for` loop) allows us to look at the WGSS as a function of the number of clusters for values of K from 1 to 8.

```
> wgss<-rep(0,8)
> for(i in 1:8){
+ wgss[i]<-kmeans(Cluster_Ex,centers=i,nstart=50)$tot.withinss
+ }
> plot(c(1:8),wgss,type="b",pch=16,cex=1.3,ylim=c(0,100),
+ xlab="Number of Groups",ylab="WGSS")
```

How Many Clusters?

We can plot the WGSS against the number of clusters and look for an "elbow" in the curve, where the rate of decrease changes dramatically. This is, of course, somewhat subjective.



From the graph above, it appears that $K = 3$ clusters is the "best" solution, where the downward slope of the curve changes markedly.

Between Group Sum of Squares

Let $\bar{x}$ be the global centroid (calculated from all units). Then we define the Between-Group Sum of Squares (BGSS) as

$$\mathrm{BGSS} = \sum_{k=1}^{K} n_k \sum_{j=1}^{p} (\bar{x}_j - \bar{x}_{kj})^2$$

where n_k is the number of units in cluster k and $\bar{x}_k$ is the centroid of cluster k.

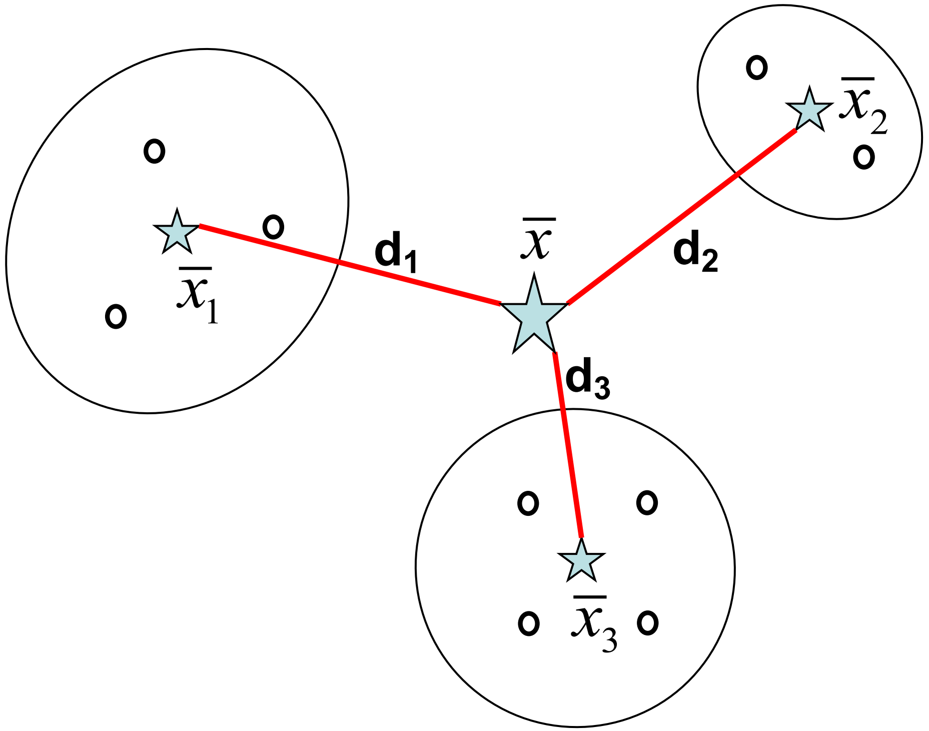
This value can be obtained from a `kmeans` object using

```
> Kmeans_3$betweenss
[1] 54.50209
```

The BGSS is also a useful measure related to a clustering and has been combined the WGSS to form various statistics for examining the number of groups question.

Between Group Sum of Squares

The Between-Group Sum of Squares (BGSS) can be visualized with



$$\text{BGSS} = \sum_{k=1}^{K} n_k \sum_{j=1}^{p} (\bar{x}_j - \bar{x}_{kj})^2$$

$$= 3d_1^2 + 2d_2^2 + 4d_3^2$$

Some Statistics for the Number of Groups

Calinski and Harabasz (1974) proposed the CH index defined by:

$$CH(K) = \frac{BGSS(K)/(K-1)}{WGSS(K)/(n-K)}$$

Total WGSS

The “best” solution is the *K* that maximizes the CH index.

This index was shown to be the most effective (under a fixed set of assumptions) by Milligan and Cooper (1985) at identifying the number of groups present in generated data.

Hartigan (1975) proposed the index:

$$H(K) = \left(\frac{WGSS(K)}{WGSS(K+1)} - 1 \right) \Big/ (n-K-1)$$

The “best” solution is smallest *K* such that $H(K) \leq 10$.

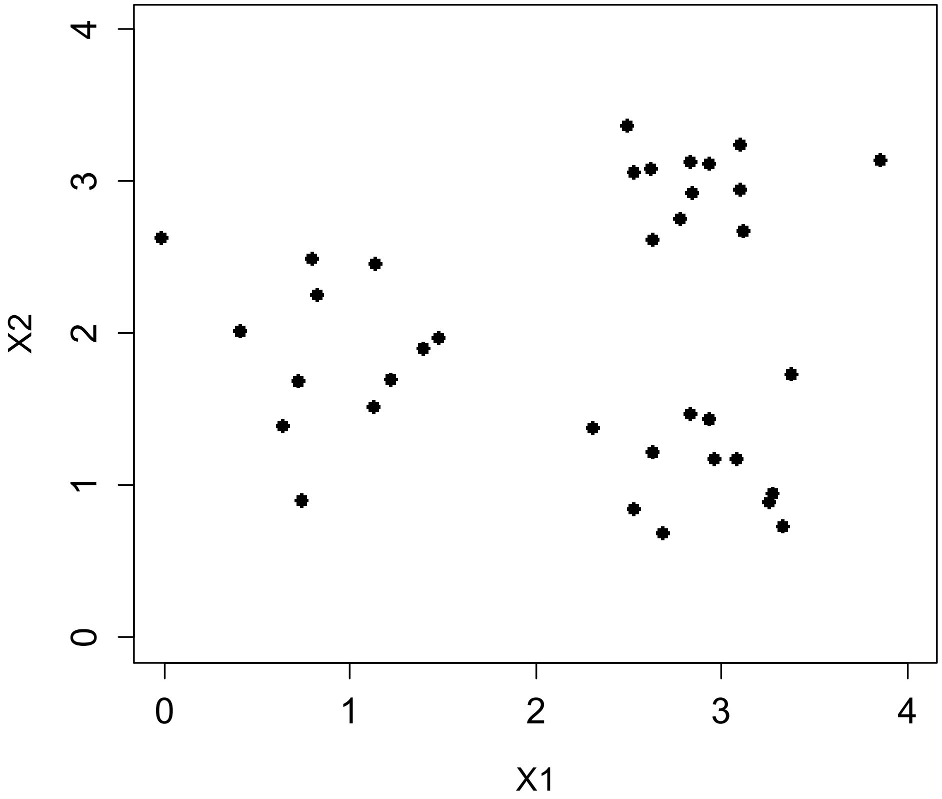
Example with the CH Index

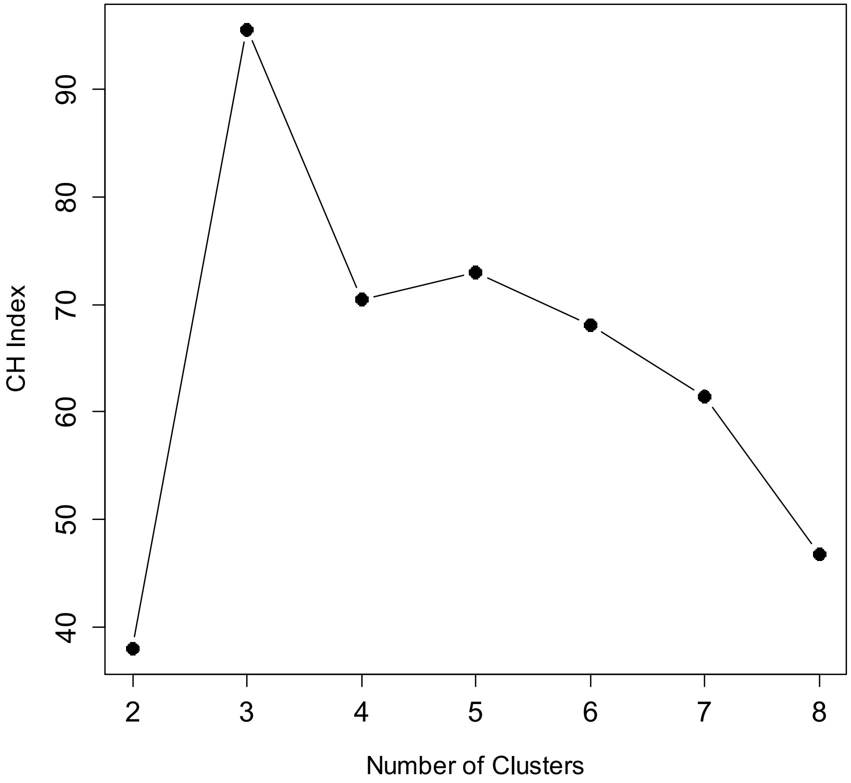
For our toy dataset `Cluster_Ex`, we can code the following,

```
chindex=c() # or rep(0,7)
i=1
for(k in 2:8){
  chindex[i] <- (kmeans(df[2:3],k)$betweenss/(k-1))   /
(kmeans(df[2:3],k)$tot.withinss/((nrow(df)-k)))
  i=i+1
}
plot(2:8,chindex,pch=19,type='b')
```

Example with the CH Index

For our toy dataset `Cluster_Ex`, we have the following output for the CH index.





The CH index clearly favors the 3-cluster solution.