K-means Clustering

Bradley Warner

May 14, 2019

Contents

Objectives **1**

Unsupervised Learning **1**

K-means Ideas **2**
Distance . . . 2
Scaling . . . 6
Distance for Categorical Variables . . . 7

K-means Algorithm **8**
Setup . . . 8
Metric . . . 9
Pseudo code . . . 9
One Iteration . . . 10

K-means in R **12**
Build First K-means model in R . . . 12
Visualize the Results . . . 13
Change Number of Clusters . . . 14
Starting Seed . . . 16
Issues to Consider . . . 18
Running Multiple Models . . . 18
Elbow Method . . . 19
Other Methods to Determine Numbers of Clusters . . . 21
More Insight on Sihouette . . . 22
Weaknesses . . . 28

Homework **30**

Objectives

1. Understand clustering in the greater scope of unsupervised learning.
2. Understand and explain distance metrics and scaling.
3. Explain K-means clustering algorithm and its parameters.
4. Use K-means clustering in R.

Unsupervised Learning

Question

What is unsupervised learning in comparison to supervised learning?

It is more challenging because it is hard to tune and validate models.

Examples:
1. Marketing segmentation
2. Gene grouping
3. Dimension reduction in predictive modeling
4. Language translation models in deep learning
5. EDA
6. Cyber profiling

Question

For which of the following would clustering likely be appropriate:

1. Predicting if a user will click on an AD
2. Identifying stocks that follow similar trading patterns
3. Modeling 30-day mortality post-surgery
4. Grouping response to customer survey
5. Identifying similar players for a fantasy league draft

K-means Ideas

K-means is a partitioning method that finds subgroups that are similar. We must consider:

1. How many subgroups.
2. What does similar mean.

Distance

In this example, we are creating two points, consider them to be players on a field.

```
two_players<-data.frame(x=c(5,15),y=c(4,10))
```

Load the libraries.

```
library(tidyverse)
```

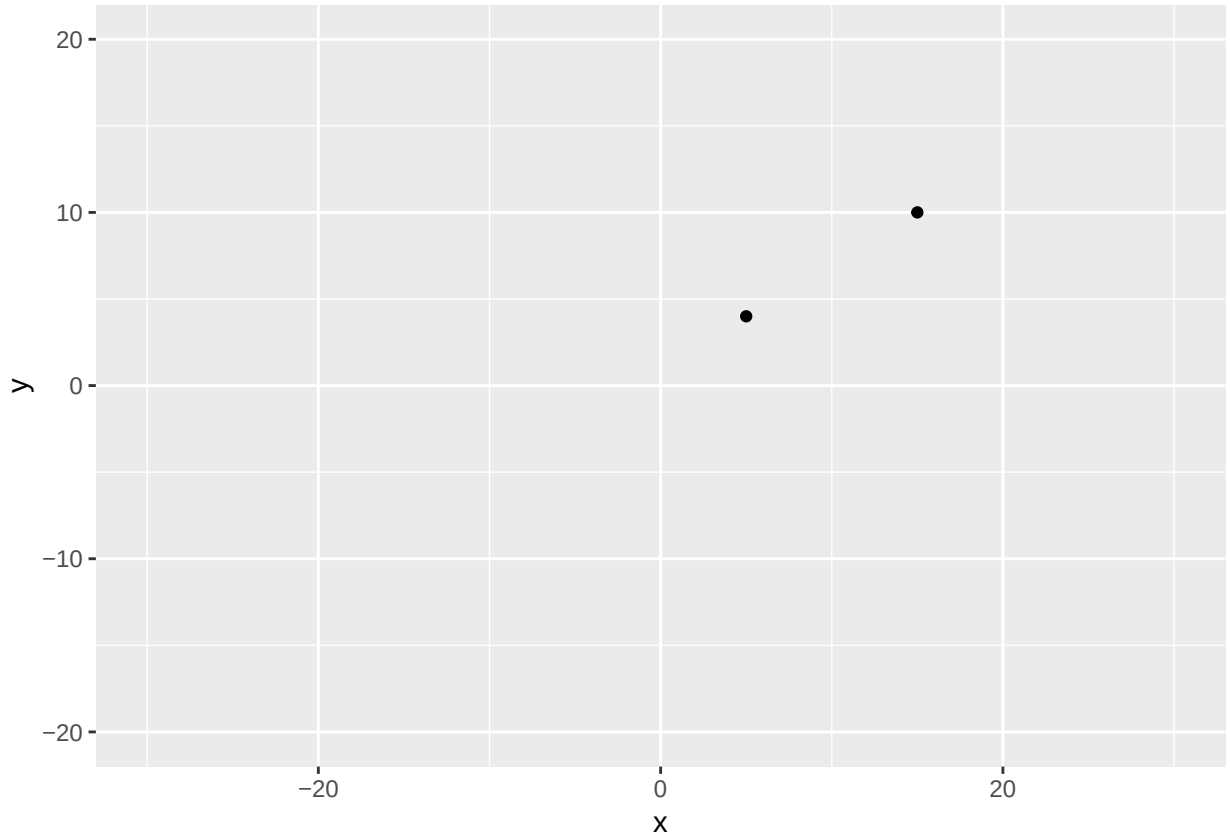
```
## -- Attaching packages --------------------------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.3.0
## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Plot the two players and then calculate their Euclidean distance.

```
# Plot the positions of the players
ggplot(two_players, aes(x = x, y = y)) +
  geom_point() +
  # Assuming a 40x60 field
  lims(x = c(-30,30), y = c(-20, 20))
```



The distance between them is:

```
# Split the players data frame into two observations
player1 <- two_players[1, ]
player2 <- two_players[2, ]

# Calculate and print their distance using the Euclidean Distance formula
player_distance <- sqrt( (player1$x - player2$x)^2 + (player1$y - player2$y)^2 )
player_distance
```

```
## [1] 11.6619
```

There is a built in function for this called dist.

```
# Calculate the Distance Between two_players
dist_two_players <- dist(two_players)
dist_two_players
```

```
##         1
## 2 11.6619
```

Let's add another player and find the distance.

```
three_players <- rbind(c(0,20),two_players)
three_players
```

```
##    x  y
## 1  0 20
## 2  5  4
```

```
## 3 15 10
```

The distance between them pairwise is:

```
# Calculate the Distance Between two_players
dist_three_players <- dist(three_players)
dist_three_players
```

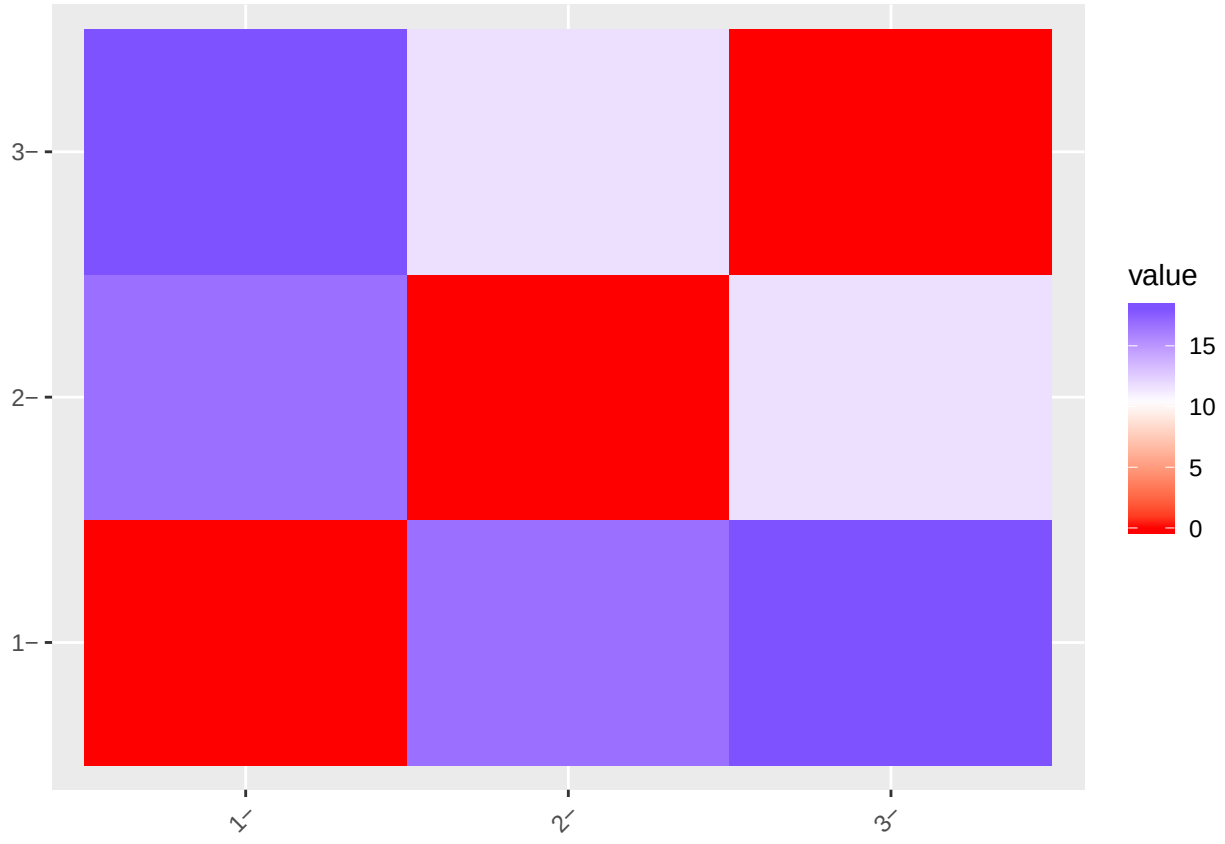
```
##          1        2
## 2 16.76305
## 3 18.02776 11.66190
```

Let's use the factoextra package to look at distance.

```
library(factoextra)
```

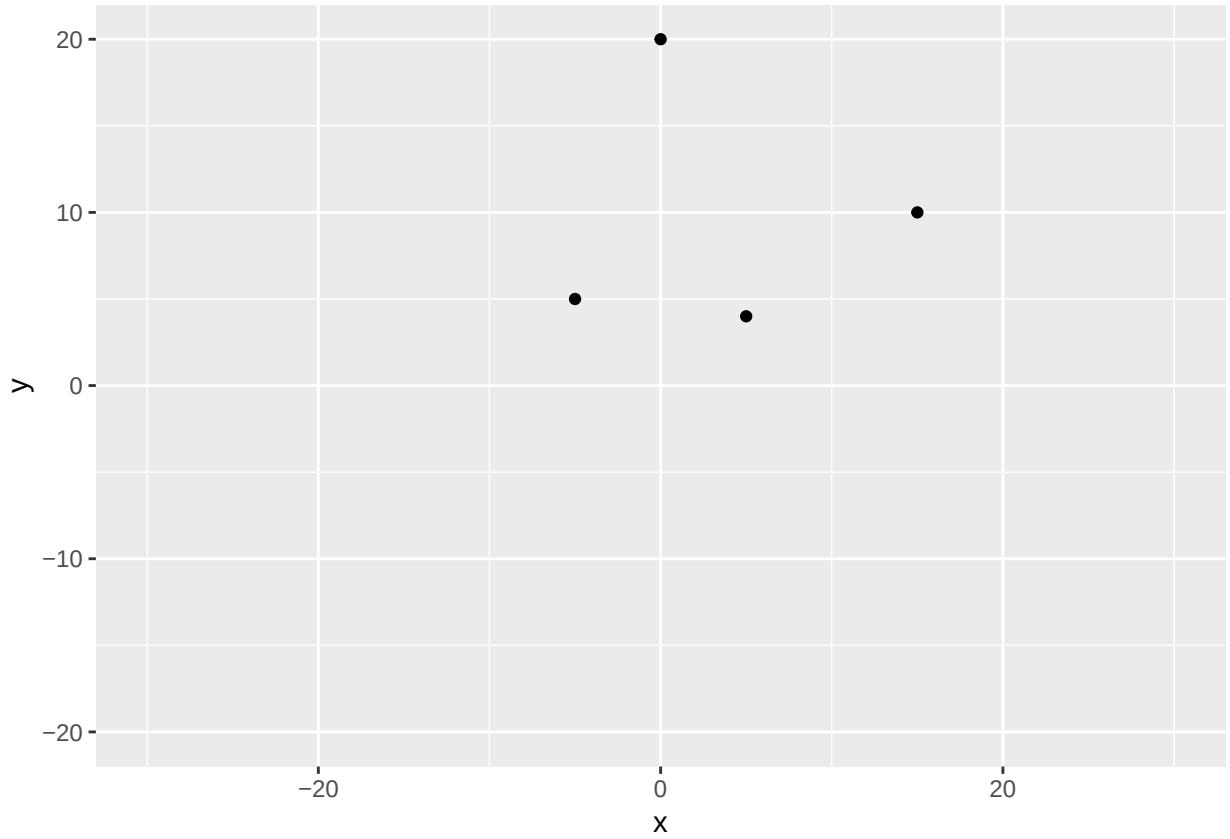
```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```
fviz_dist(get_dist(three_players))
```



Let's add another player.

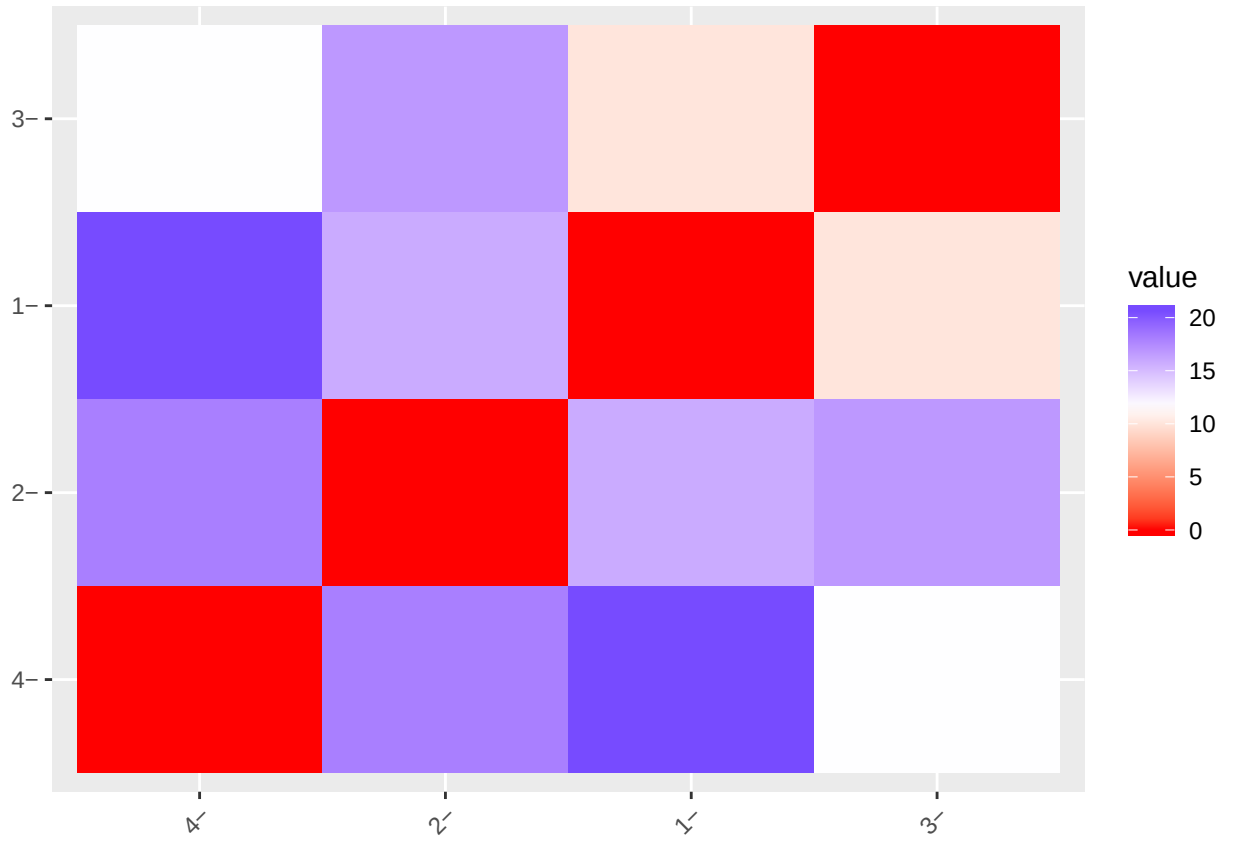
```
four_players<-rbind(c(-5,5),three_players)
# Plot the positions of the players
ggplot(four_players, aes(x = x, y = y)) +
  geom_point() +
  # Assuming a 40x60 field
  lims(x = c(-30,30), y = c(-20, 20))
```




```
get_dist(four_players)
```

```
##          1        2        3
## 2 15.81139
## 3 10.04988 16.76305
## 4 20.61553 18.02776 11.66190
```

```
fviz_dist(get_dist(four_players))
```



Scaling

What happens if are variables have different scales. Let's use the trees data. To get more information on the data type:

```
?trees
```

```
three_trees <- trees[c(1,2,4),1:2]
three_trees
```

```
##   Girth Height
## 1   8.3     70
## 2   8.6     65
## 4  10.5     72
```

Notice that height is a much larger scale than girth and will dominate the distance metric.

```
get_dist(three_trees)
```

```
##          1        2
## 2 5.008992
## 4 2.973214 7.253275
```

Points 1 and 4 are the closest in terms of height but that is because it is a much larger scale than girth.

Scaling will put all variables on same scale.

```
get_dist(three_trees,stand=TRUE)
```

```
##          1        2
## 2 1.409365
## 4 1.925659 2.511082
```

Notice that points 1 and 2 are now the closest.

Question

Should we scale if we are working with height, weight, and body fat?

Distance for Categorical Variables

What about categorical variables?

Let's makeup some data.

```
job_survey <- data.frame(job_satisfaction=c("Low","Low","Hi","Low","Mid"),
                         is_happy = c("No","No","Yes","No","No"))
job_survey
```

```
##   job_satisfaction is_happy
## 1              Low       No
## 2              Low       No
## 3               Hi      Yes
## 4              Low       No
## 5              Mid       No
```

Load a new library.

```
library(dummies)
```

```
## dummies-1.5.6 provided by Decision Patterns
```

```
# Dummify the Survey Data
dummy_survey <- dummy.data.frame(job_survey)

# Calculate the Distance
dist_survey <- get_dist(dummy_survey,method="binary")

# Print the Original Data
job_survey
```

```
##   job_satisfaction is_happy
## 1              Low       No
## 2              Low       No
## 3               Hi      Yes
## 4              Low       No
## 5              Mid       No
```

```
dummy_survey
```

```
##   job_satisfactionHi job_satisfactionLow job_satisfactionMid is_happyNo
## 1                  0                   1                   0          1
## 2                  0                   1                   0          1
## 3                  1                   0                   0          0
## 4                  0                   1                   0          1
## 5                  0                   0                   1          1
```

```
##   is_happyYes
## 1           0
## 2           0
## 3           1
## 4           0
## 5           0
```

One hot encoding in deep learning.

```
# Print the Distance Matrix
dist_survey
```

```
##           1         2         3         4
## 2 0.0000000
## 3 1.0000000 1.0000000
## 4 0.0000000 0.0000000 1.0000000
## 5 0.6666667 0.6666667 1.0000000 0.6666667
```

The binary metric is Jaccard distance which is 1 minus the intersection over the union.

If we have mixed, then we could use the Gower metric with the daisy function in the cluster package.

We are ready to learn about the algorithm for K-means.

K-means Algorithm

We want to develop an insight into the algorithm. We will use the simple players data to motivate the algorithm.

Setup

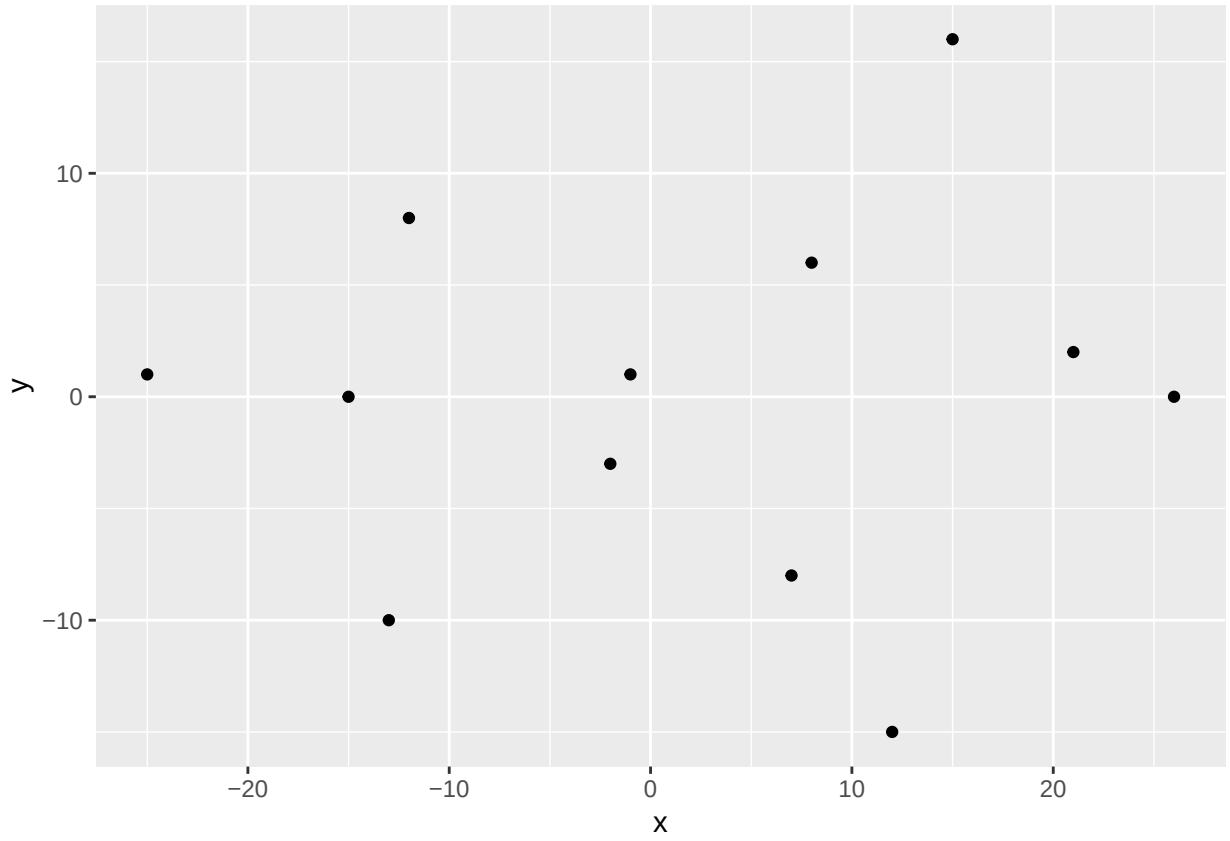
Get data for the two teams.

```
lineup <- readRDS("lineup.rds")
lineup
```

```
## # A tibble: 12 x 2
##        x     y
##    <dbl> <dbl>
##  1    -1     1
##  2    -2    -3
##  3     8     6
##  4     7    -8
##  5   -12     8
##  6   -15     0
##  7   -13   -10
##  8    15    16
##  9    21     2
## 10    12   -15
## 11   -25     1
## 12    26     0
```

Plot the data.

```
ggplot(lineup, aes(x = x, y = y)) +
  geom_point()
```



Metric

In this problem we know that there are two teams so we have two clusters. In most problems we will not know the clusters. To find the clusters we need a metric. We minimize the within-cluster variation.

$$\frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

by expanding around the cluster mean, this is equivalent to

$$2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$$

Pseudo code

```
Pick number of cluster

1. Randomly assign each point to a cluster or randomly pick centroid
2. Iterate until cluster assignments stop changing
  a) Find centroids, mean of the features
  b) Assign each observation to cluster with closet centroid.
```

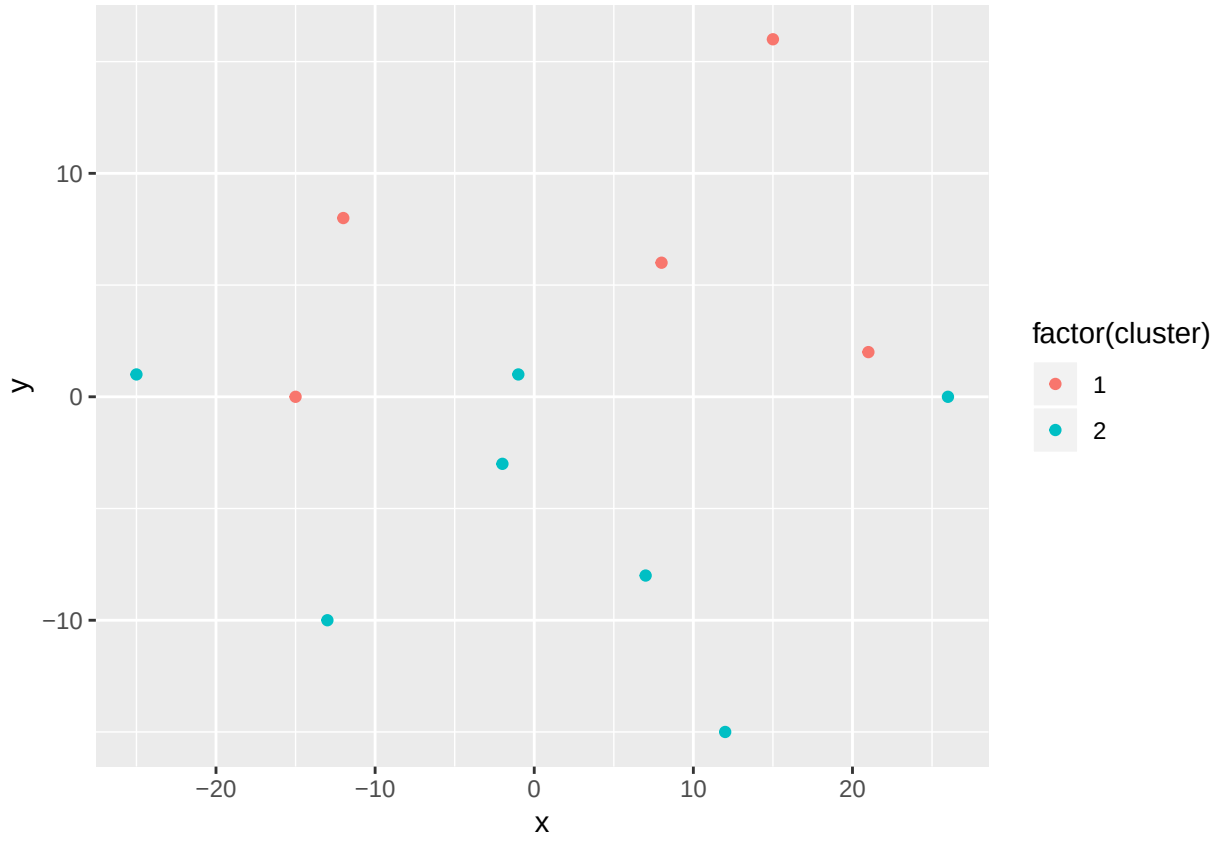
One Iteration

Let take a couple of steps manually through the algorithm

```
set.seed(2019)
cluster <- sample(c(1,2),12,replace=TRUE)
```

Plot the positions of the players and color them using their cluster

```
ggplot(lineup, aes(x = x, y = y, color = factor(cluster))) +
  geom_point()
```



Find the centroids.

```
map_dbl(lineup[cluster==1,],mean)
```

```
##   x   y
## 3.4 6.4
```

```
centroids <- cbind(lineup,cluster) %>% group_by(cluster) %>%
  summarise(x=mean(x),y=mean(y))
centroids
```

```
## # A tibble: 2 x 3
##   cluster     x     y
##     <dbl> <dbl> <dbl>
## 1       1 3.4    6.4
## 2       2 0.571 -4.86
```

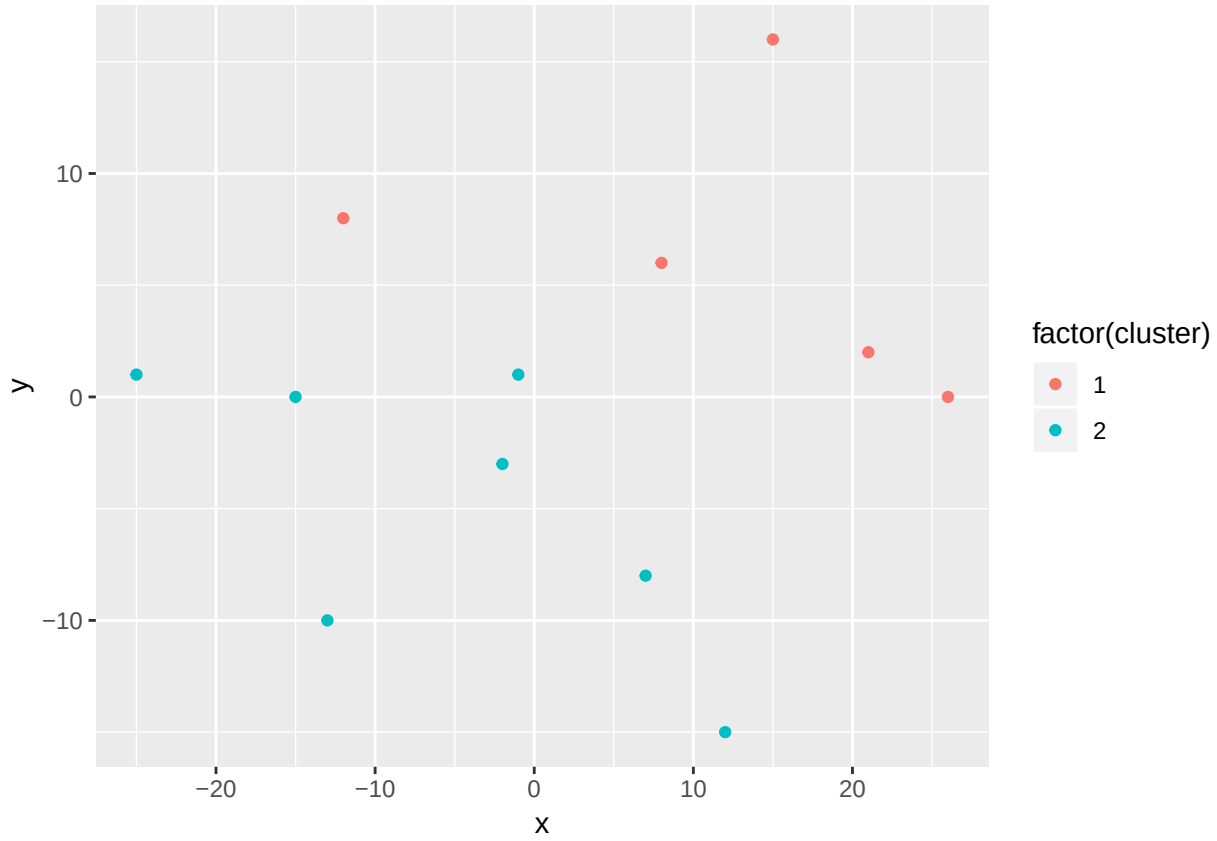
```
dist_new <- as.matrix(get_dist(rbind(centroids[,-1],lineup)))[-(1:2),1:2]
dist_new
```

```
##            1         2
## 3   6.965630  6.064282
## 4  10.840664  3.171943
## 5   4.617359 13.155274
## 6  14.843180  7.155703
## 7  15.482894 17.981850
## 8  19.481273 16.311383
## 9  23.193102 14.513189
## 10 15.057224 25.361468
## 11 18.141665 21.548711
## 12 23.063391 15.280373
## 13 28.908822 26.233644
## 14 23.488721 25.888300
```

```
cluster <- (dist_new[,1]>dist_new[,2]) +1
cluster
```

```
##  3  4  5  6  7  8  9 10 11 12 13 14
##  2  2  1  2  1  2  2  1  1  2  2  1
```

```
ggplot(lineup, aes(x = x, y = y, color = factor(cluster))) +
  geom_point()
```



K-means in R

We will now build are first K-means model using the function in R. We will use default parameters at first.

Build First K-means model in R

```
model_km2 <- kmeans(lineup, centers = 2)
```

```
names(model_km2)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

Extract the cluster assignment vector from the K-means model

```
clust_km2 <- model_km2$cluster
clust_km2
```

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

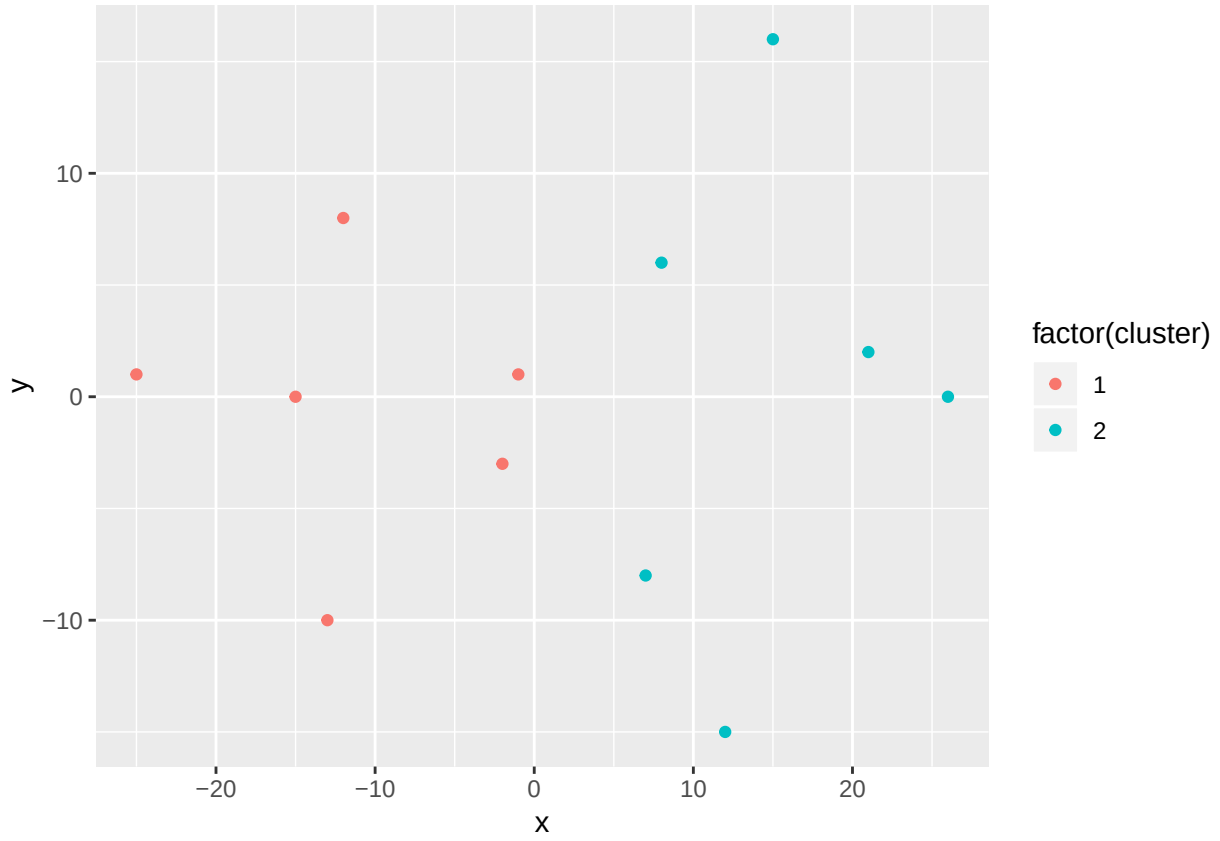
Create a new data frame appending the cluster assignment.

```
lineup_km2 <- mutate(lineup, cluster = clust_km2)
lineup_km2
```

```
## # A tibble: 12 x 3
##        x     y cluster
##    <dbl> <dbl>   <int>
##  1    -1     1       1
##  2    -2    -3       1
##  3     8     6       2
##  4     7    -8       2
##  5   -12     8       1
##  6   -15     0       1
##  7   -13   -10       1
##  8    15    16       2
##  9    21     2       2
## 10    12   -15       2
## 11   -25     1       1
## 12    26     0       2
```

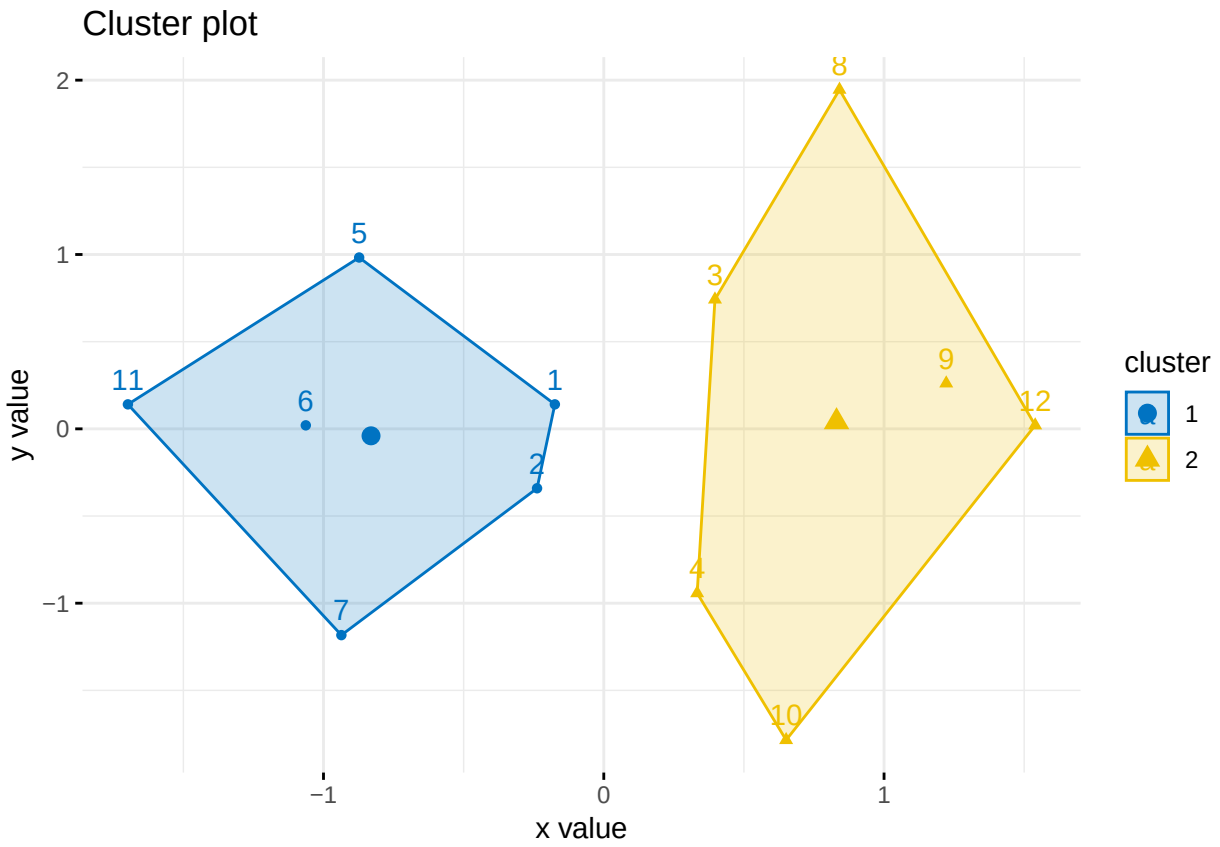
Plot the positions of the players and color them using their cluster

```
ggplot(lineup_km2, aes(x = x, y = y, color = factor(cluster))) +
  geom_point()
```



Visualize the Results

```
fviz_cluster(model_km2, data = lineup, palette = "jco",
             ggtheme = theme_minimal())
```



Change Number of Clusters

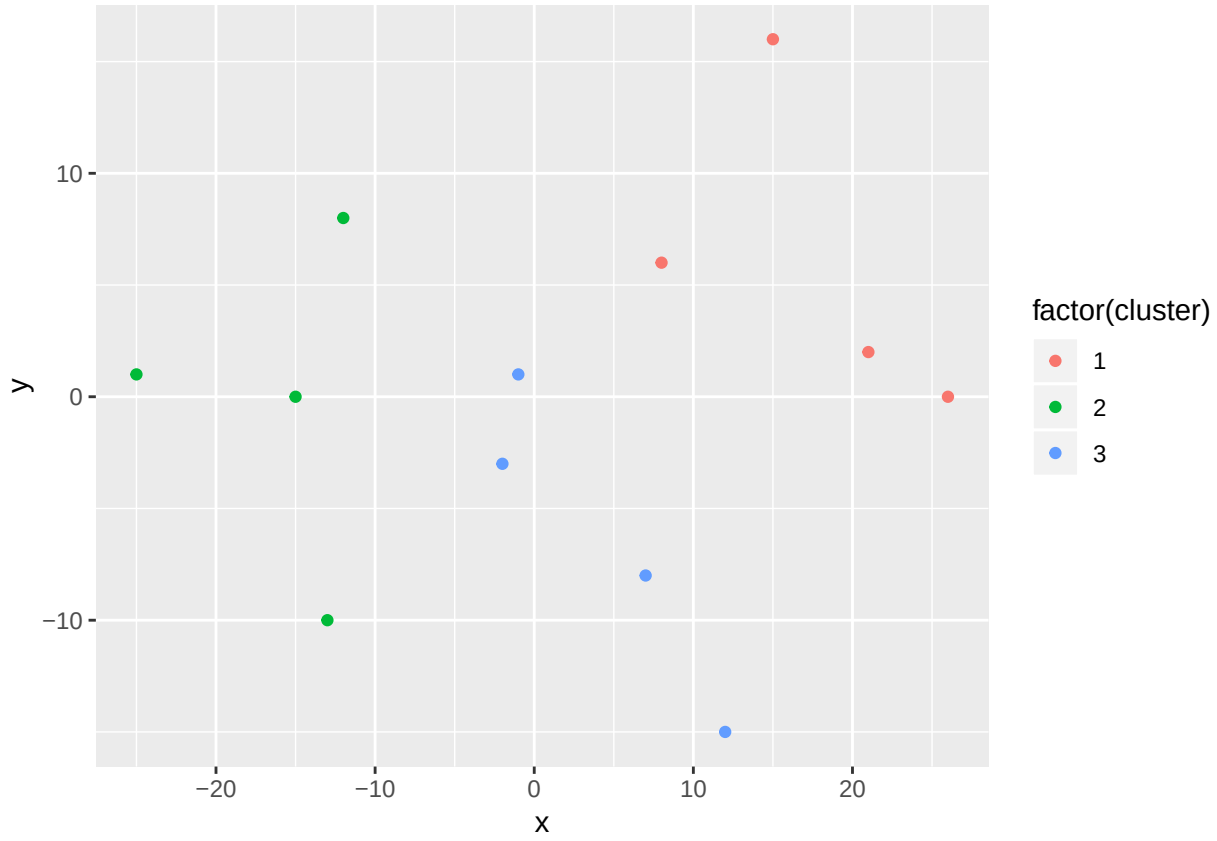
Since we knew beforehand that there were two teams we made this problems easier. Often we do not know how many clusters. Before we attach this problem, let's see what happens if we use three clusters.

Build the K-means model.

```
# Build a kmeans model
model_km3 <- kmeans(lineup,centers=3)

# Extract the cluster assignment vector from the kmeans model
clust_km3 <- model_km3$cluster

# Create a new data frame appending the cluster assignment
lineup_km3 <- mutate(lineup,cluster = clust_km3)
```

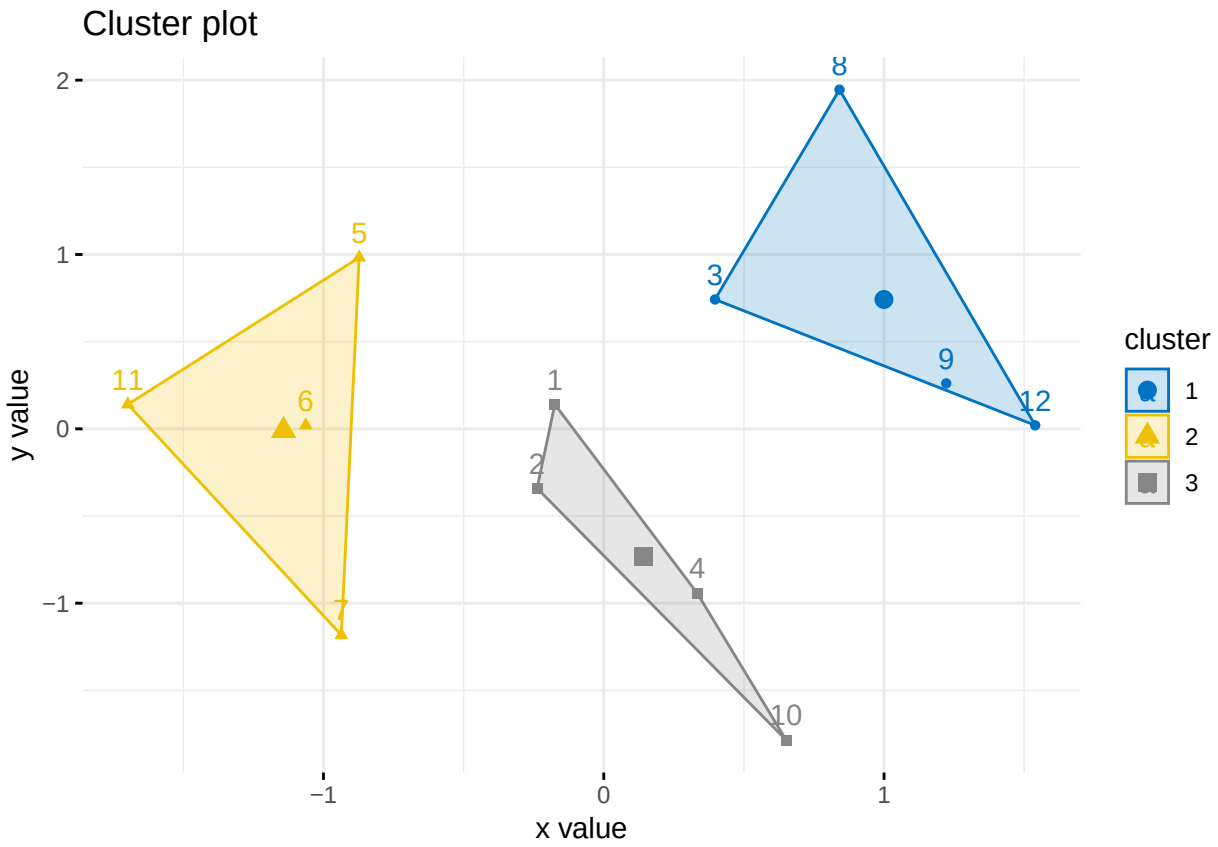
Now let's plot the results.

```
# Plot the positions of the players and color them using their cluster
ggplot(lineup_km3, aes(x = x, y = y, color = factor(cluster))) +
  geom_point()
```



We will look at the cluster plots.

```
fviz_cluster(model_km3, data = lineup, palette = "jco",
             ggtheme = theme_minimal())
```



The algorithm let's us use any number of clusters from 2 up to the number of data points.

Starting Seed

Let's run the same model again but since the seed is different, we may get a different model.

```
set.seed(11)
# Build a kmeans model
model_km3a <- kmeans(lineup,centers=3)

# Extract the cluster assignment vector from the kmeans model
clust_km3a <- model_km3a$cluster

# Create a new data frame appending the cluster assignment
lineup_km3a <- mutate(lineup,cluster = clust_km3a)
```

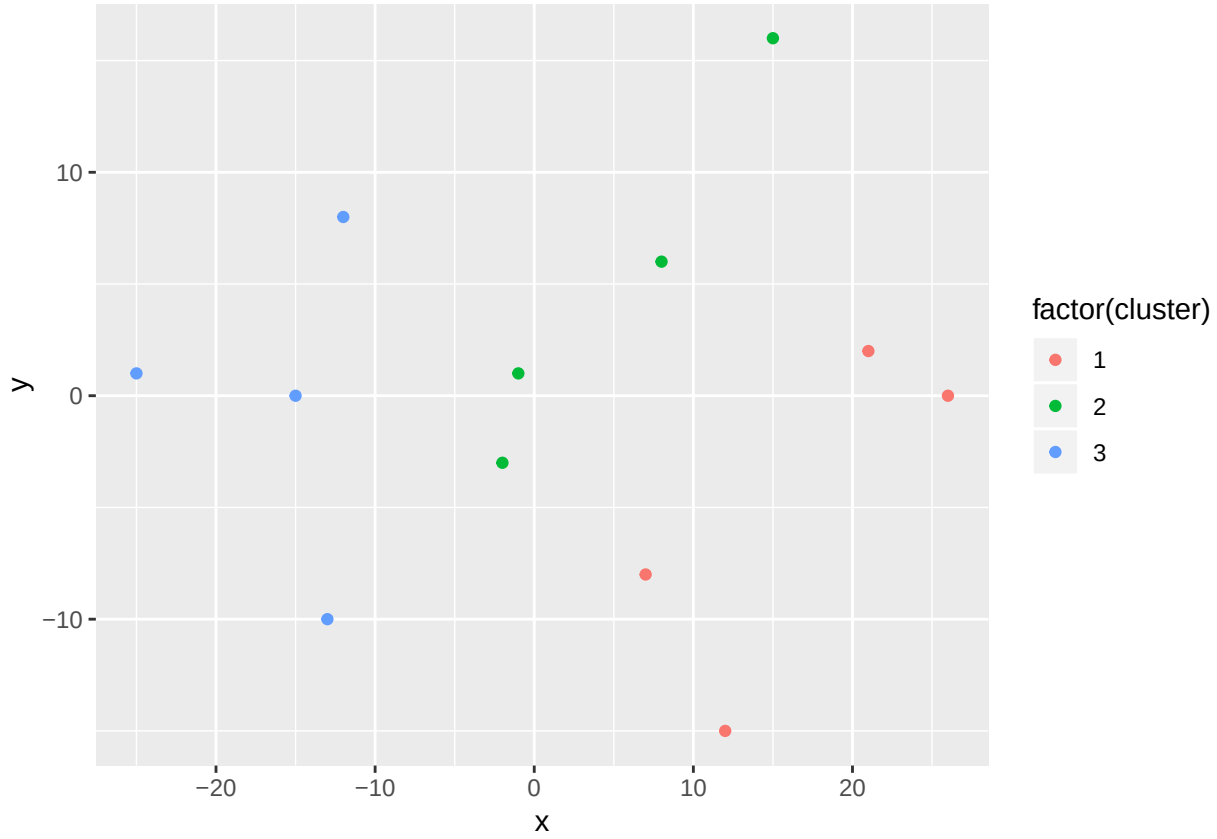
```
model_km3
```

```
## K-means clustering with 3 clusters of sizes 4, 4, 4
##
## Cluster means:
##         x     y
## 1  17.50  6.00
## 2 -16.25 -0.25
## 3   4.00 -6.25
##
```

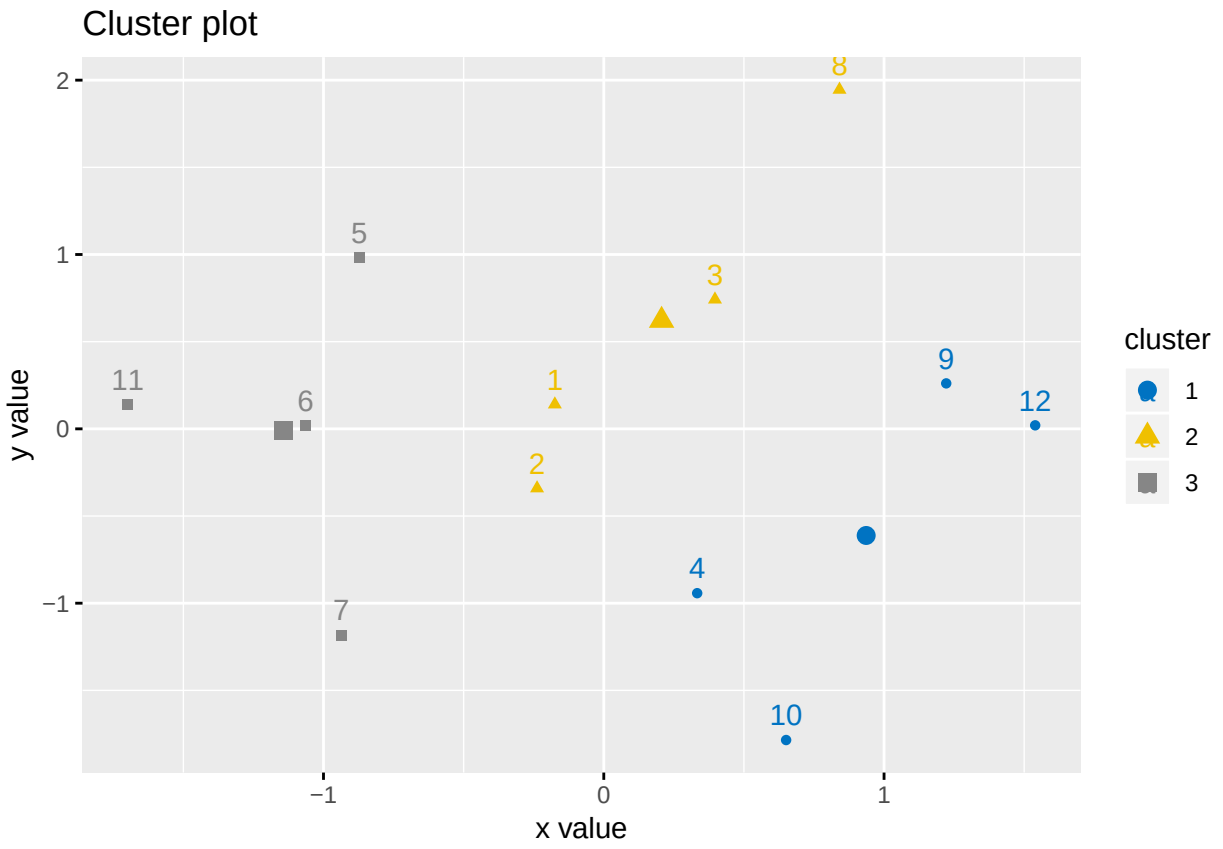
```
## Clustering vector:
##  [1] 3 3 1 3 2 2 2 1 1 3 2 1
##
## Within cluster sum of squares by cluster:
## [1] 333.00 271.50 276.75
##  (between_SS / total_SS =  74.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

Now let's plot the results.

```
# Plot the positions of the players and color them using their cluster
ggplot(lineup_km3a, aes(x = x, y = y, color = factor(cluster))) +
  geom_point()
```




```
fviz_cluster(model_km3a, data = lineup, palette = "jco", ellipse = FALSE,
             ggtheme = theme_gray())
```



Issues to Consider

The model is sensitive to the starting position of clusters and also to the number of clusters.

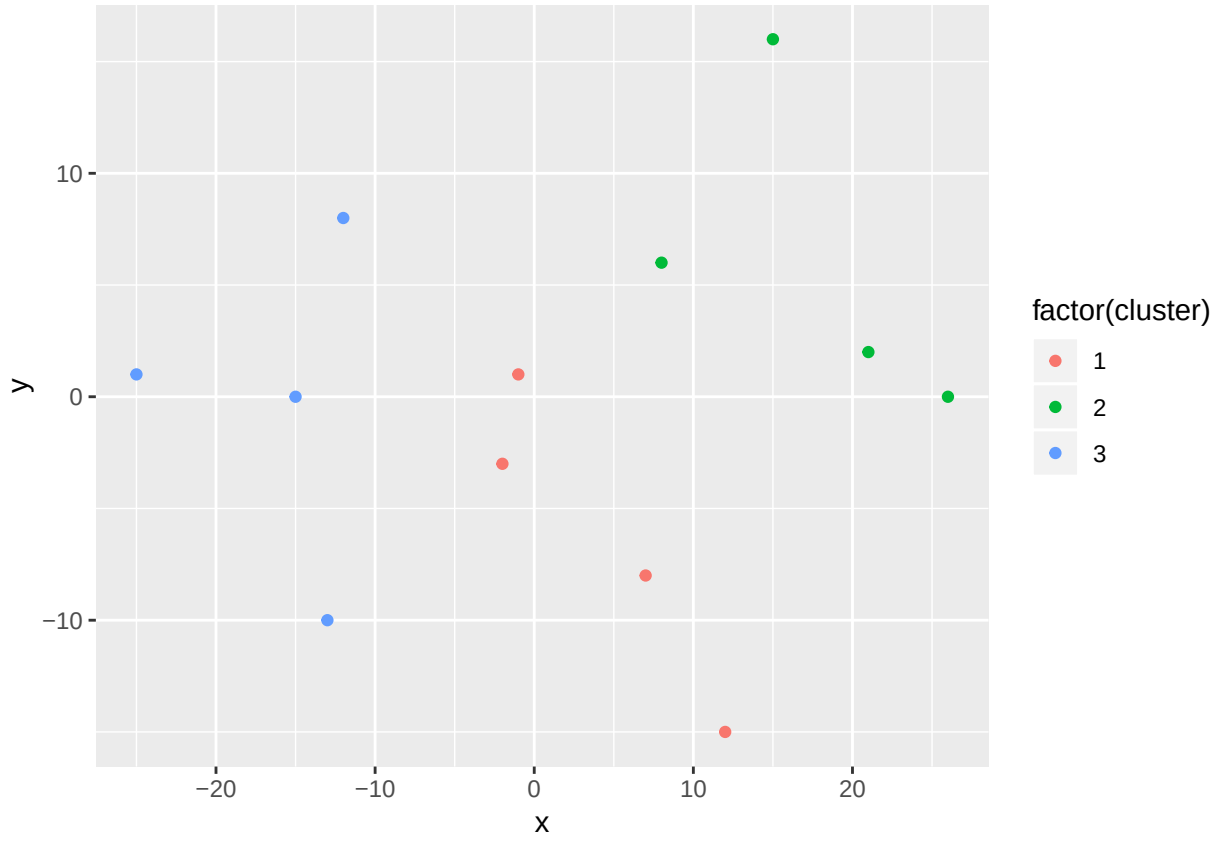
Thus we need: 1 Run K-means for different random seeds with fixed number of clusters 2 Find method to determine the number of clusters

Running Multiple Models

To address the first issue, R let's use run multiple models with the **nstart** option and will return the single best model. What is best?

```
# Build a kmeans model
model_km3_best <- kmeans(lineup,centers=3,nstart=25)

# Extract the cluster assignment vector from the kmeans model
clust_km3_best <- model_km3_best$cluster

# Create a new data frame appending the cluster assignment
lineup_km3_best <- mutate(lineup,cluster = clust_km3_best)
```

```
# Plot the positions of the players and color them using their cluster
ggplot(lineup_km3_best, aes(x = x, y = y, color = factor(cluster))) +
  geom_point()
```



Let's compare the models we

```
c(model_km3a$tot.withinss,model_km3_best$tot.withinss)
```

```
## [1] 1071.25  881.25
```

Elbow Method

To determine the number of clusters we can plot the total within sum of squares versus clusters and look for a diminishing return. This is called a scree plot. We will use a mapping function to perform this.

First create the mapping.

```
# Use map_dbl to run many models with varying value of k (centers)
tot_withinss <- map_dbl(1:10,  function(k){
  model <- kmeans(x = lineup, centers = k,nstart=25)
  model$tot.withinss
})
```

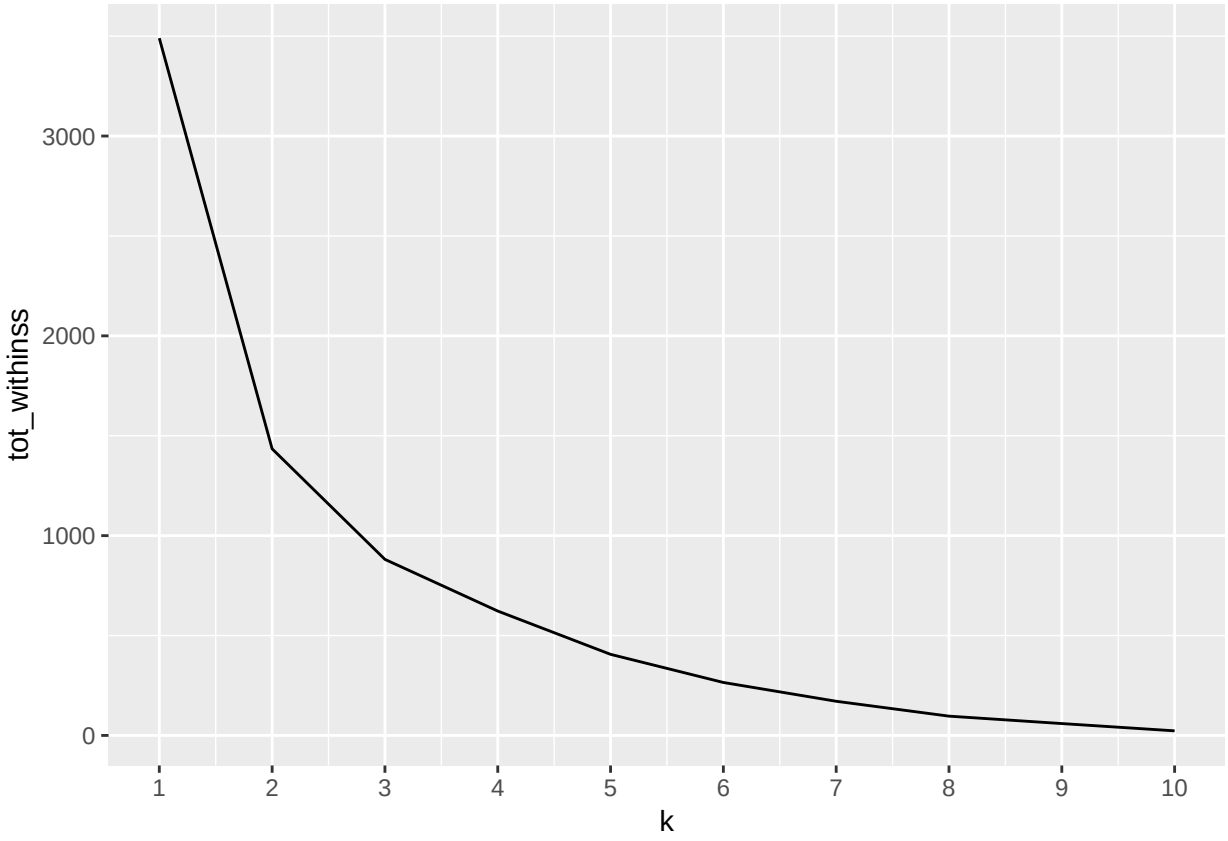
Now merge into a data frame.

```
# Generate a data frame containing both k and tot_withinss
elbow_df <- data.frame(
  k = 1:10 ,
  tot_withinss = tot_withinss
)
head(elbow_df)
```

```
##   k tot_withinss
## 1 1    3489.9167
## 2 2    1434.5000
## 3 3     881.2500
## 4 4     622.5000
## 5 5     406.0000
## 6 6     265.1667
```

Let's plot the within sum of squares as a function of the number of clusters.

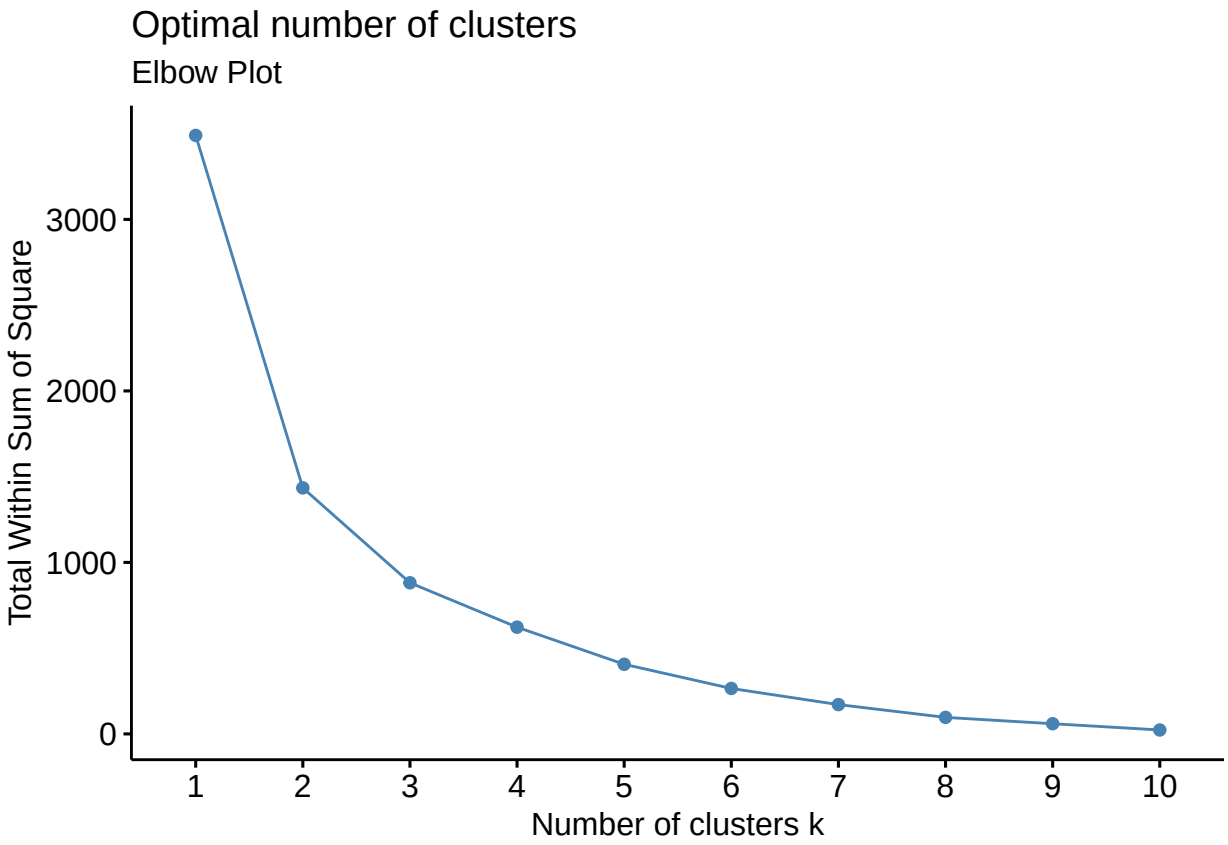
```
# Plot the elbow plot
ggplot(elbow_df, aes(x = k, y = tot_withinss)) +
  geom_line() +
  scale_x_continuous(breaks = 1:10)
```



Note: Determining an elbow, or knee, in the plot has always been difficult for me. This methods is lacking in a clear guidance. I would think 2 or 3 are where the knee occurs but it is not obvious.

Instead of writing all this code, we could use the following code.

```
fviz_nbclust(lineup, kmeans,nstart=25, method = "wss") +
  labs(subtitle = "Elbow Plot")
```

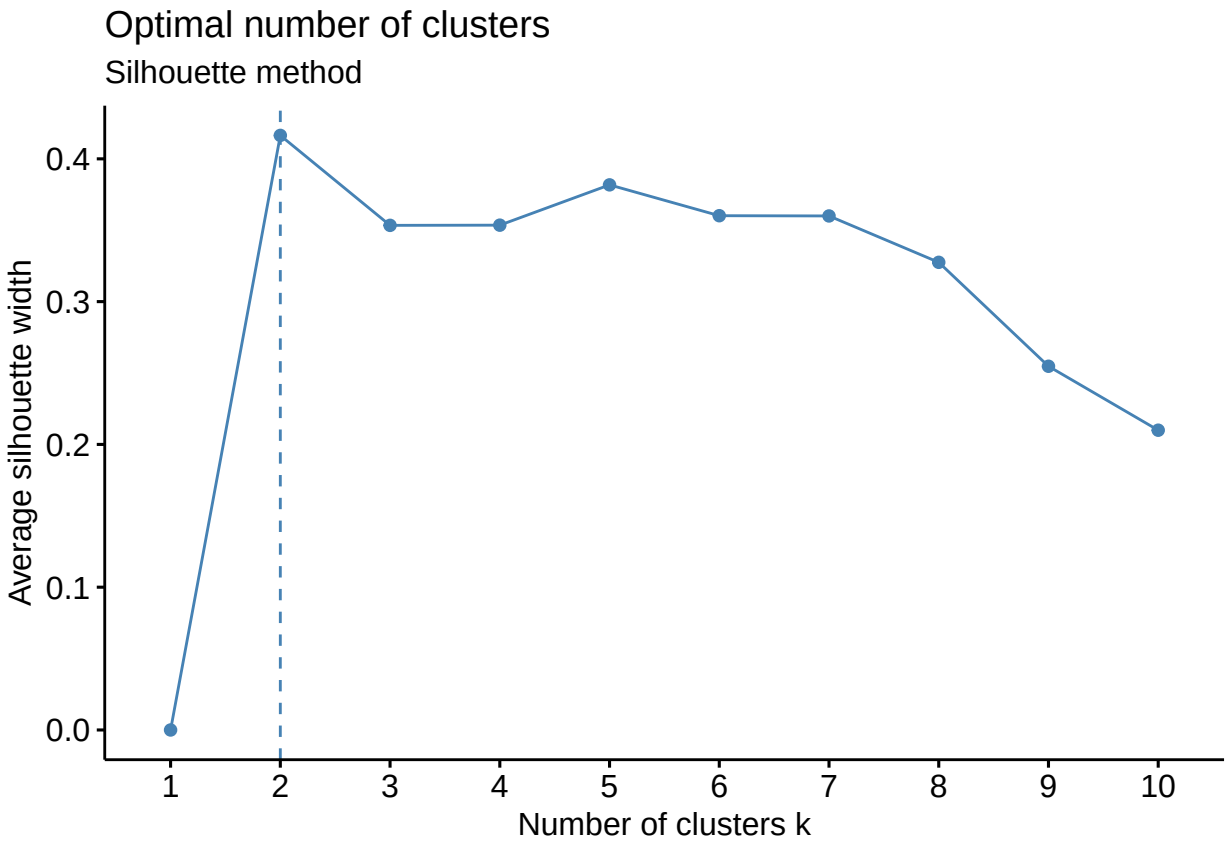


Other Methods to Determine Numbers of Clusters

There are other methods to determine the number of clusters, datanovia has a great tutorial.

The silhouette method compares the average distance of each point in a cluster to the average distance in the closet neighbor. For a particular number of cluster we can find the average silhouette distance. The closer the value to 1 the better.

```
fviz_nbclust(lineup, kmeans, nstart=25, method = "silhouette")+
  labs(subtitle = "Silhouette method")
```

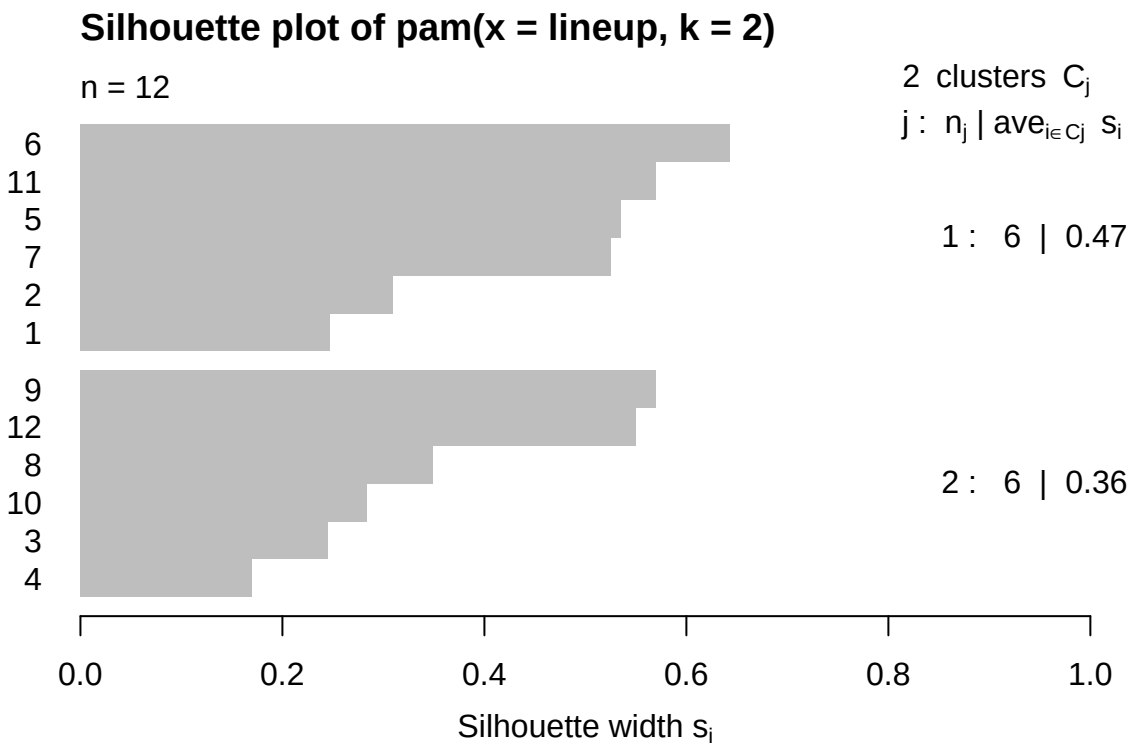


This much more satisfying.

More Insight on Sihouette

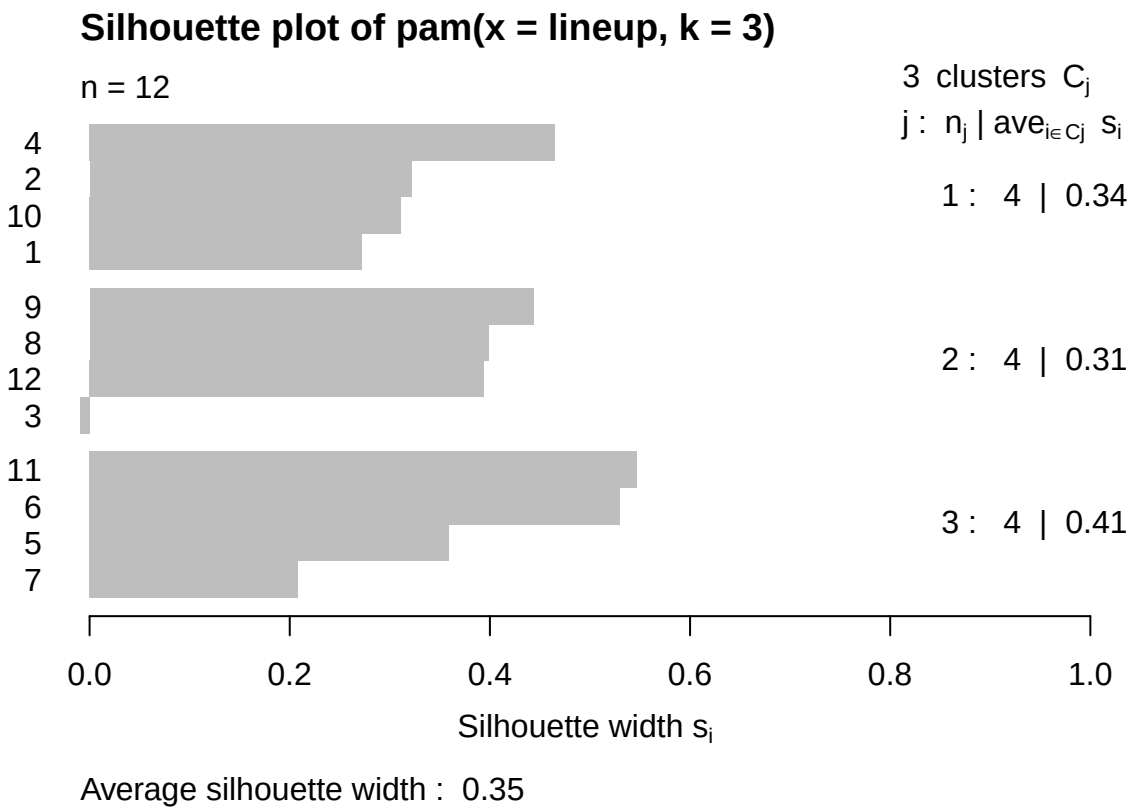
```
library(cluster)
```

```
# Generate a k-means model using the pam() function with a k = 2
pam_k2 <- pam(lineup, k = 2)

# Plot the silhouette visual for the pam_k2 model
plot(silhouette(pam_k2))
```



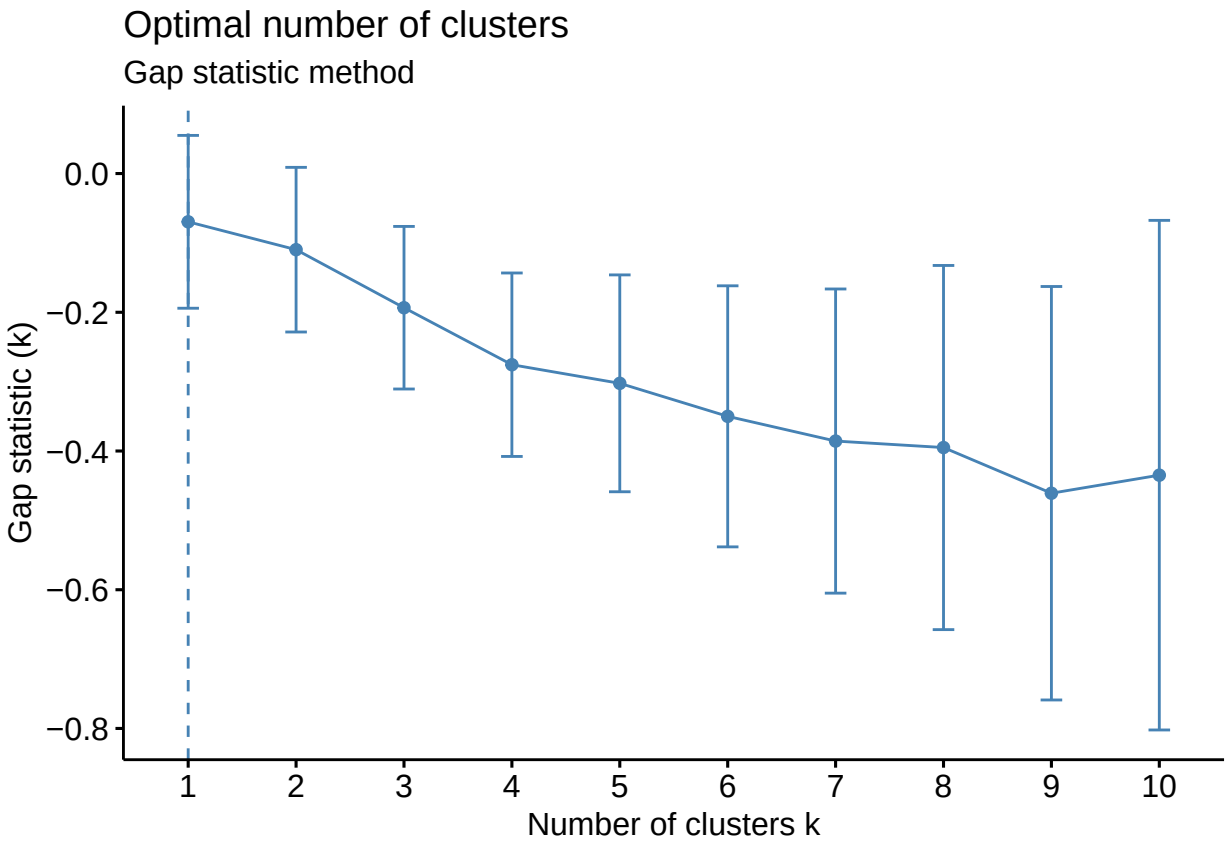
Average silhouette width : 0.42

```
# Generate a k-means model using the pam() function with a k = 3
pam_k3 <- pam(lineup,k=3)

# Plot the silhouette visual for the pam_k3 model
plot(silhouette(pam_k3))
```



Another method is using the gap statistic. Note there are over 30 indices that could be used and the package NbClust uses them all.

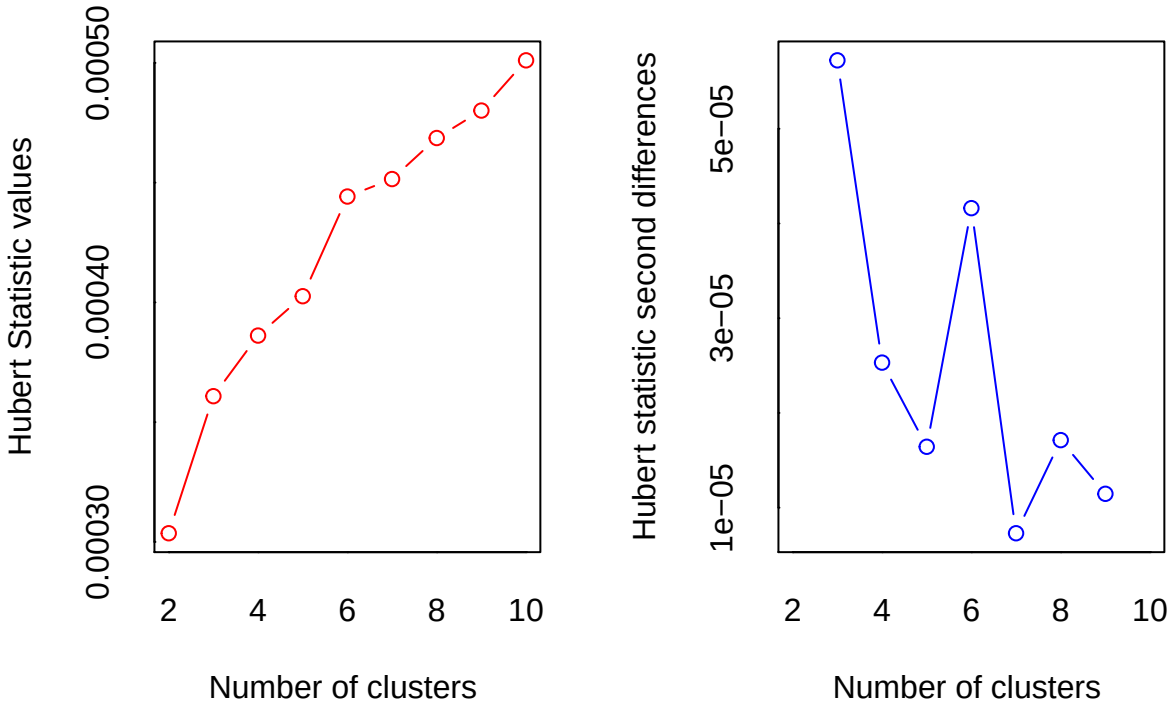
```
set.seed(511)
fviz_nbclust(lineup, kmeans, nstart = 25,  method = "gap_stat", nboot = 50)+
  labs(subtitle = "Gap statistic method")
```



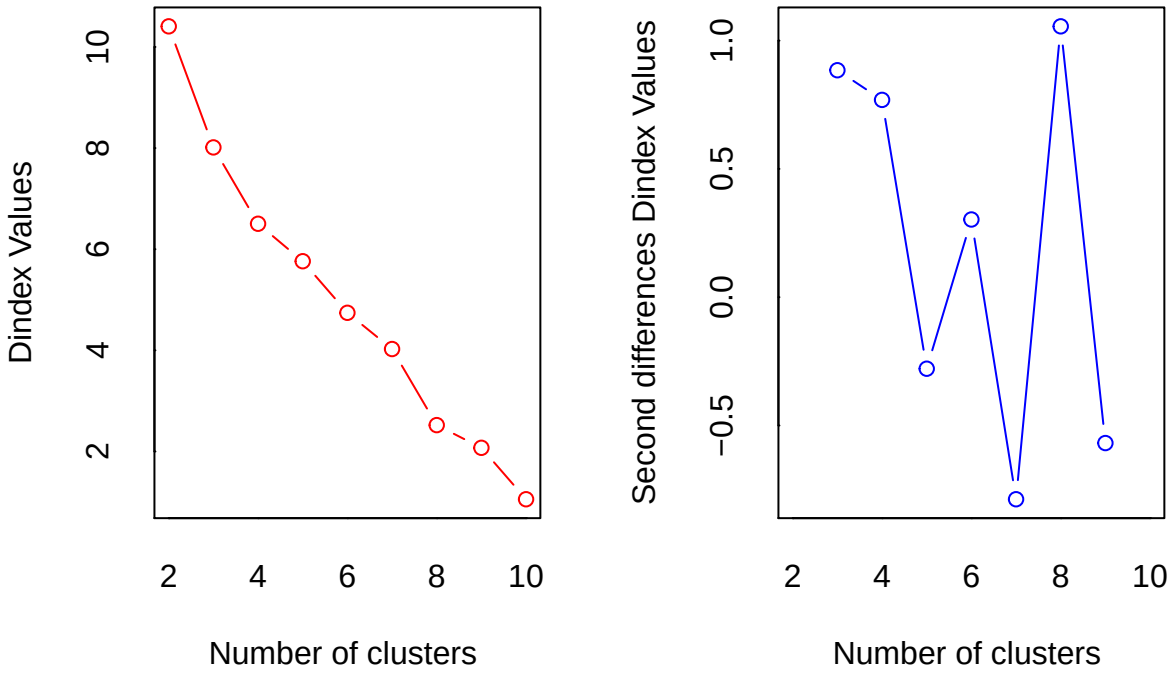

```
library(NbClust)
```

```
summary(NbClust(lineup,method='kmeans',max.nc = 10))
```

```
## Warning in pf(beale, pp, df2): NaNs produced

## Warning in pf(beale, pp, df2): NaNs produced

## Warning in pf(beale, pp, df2): NaNs produced

## Warning in pf(beale, pp, df2): NaNs produced

## Warning in pf(beale, pp, df2): NaNs produced
```




```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```




```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 6 proposed 2 as the best number of clusters
## * 3 proposed 3 as the best number of clusters
## * 2 proposed 4 as the best number of clusters
## * 1 proposed 6 as the best number of clusters
## * 5 proposed 8 as the best number of clusters
## * 6 proposed 10 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

	Length	Class	Mode
All.index	234	-none-	numeric
All.CriticalValues	27	-none-	numeric
Best.nc	52	-none-	numeric
Best.partition	12	-none-	numeric

Too few data. It is clear that selecting the number of clusters is not trivial.

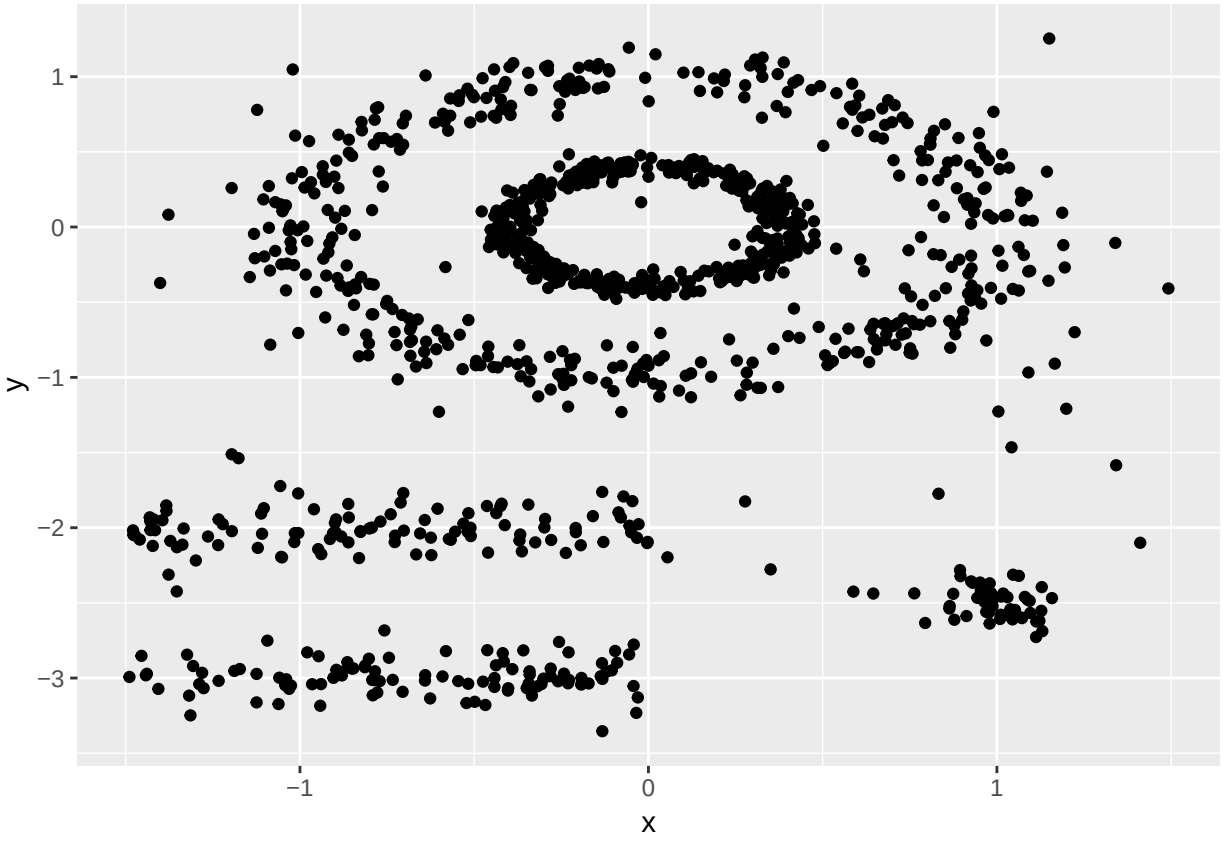
Weaknesses

Another potential problem with K-means clustering is if there are not clear partition based on a centroid. Consider the following data.

```
data("multishapes")
Small_df <- multishapes[, 1:2]
head(Small_df)
```

```
##            x          y
## 1 -0.8037393 -0.8530526
## 2  0.8528507  0.3676184
## 3  0.9271795 -0.2749024
## 4 -0.7526261 -0.5115652
## 5  0.7068462  0.8106792
## 6  1.0346985  0.3946550
```

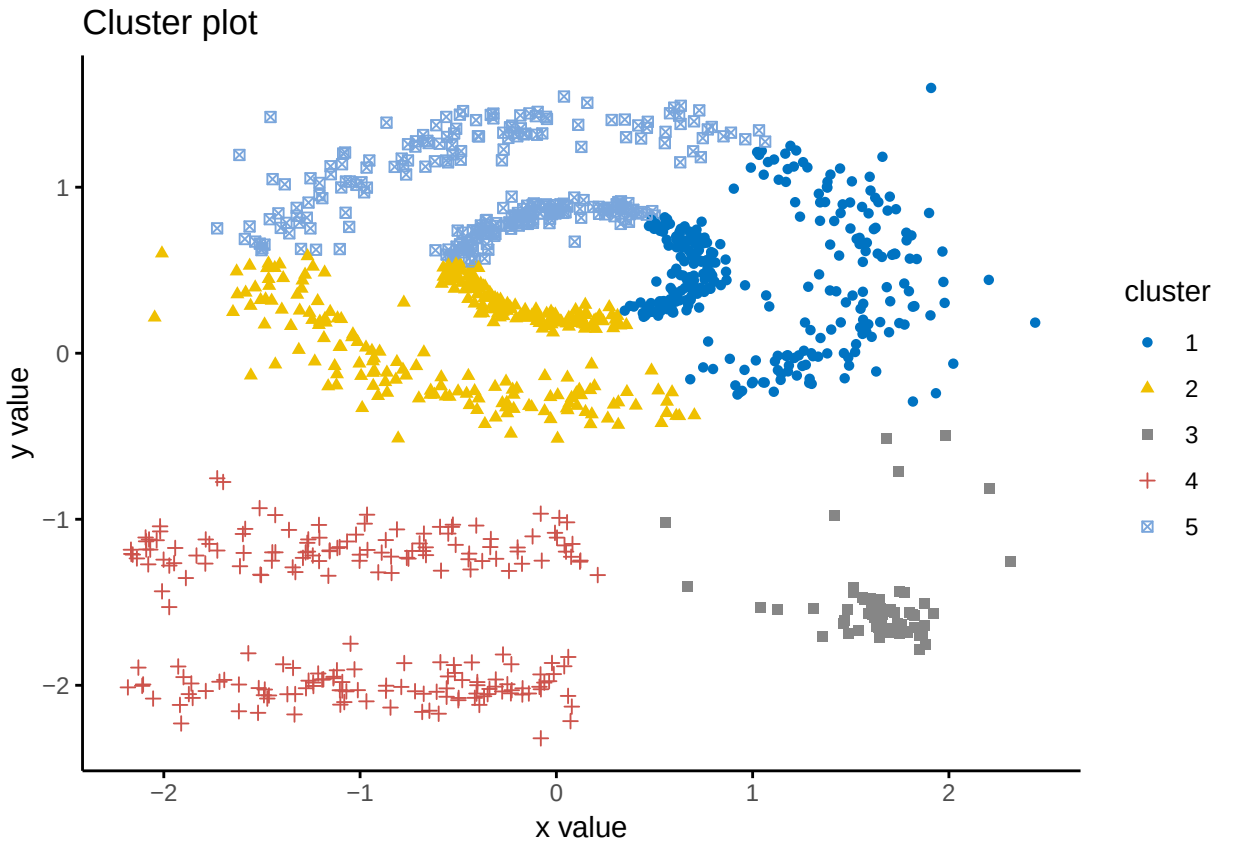
```
# Plot the positions of the players and color them using their cluster
ggplot(Small_df, aes(x = x, y = y)) +
  geom_point()
```



Let's see what K-means does with this data.

```
set.seed(123)
km.res <- kmeans(Small_df, 5, nstart = 25)
```

```
fviz_cluster(km.res, Small_df,  geom = "point",
             ellipse= FALSE, show.clust.cent = FALSE,
             palette = "jco", ggtheme = theme_classic())
```



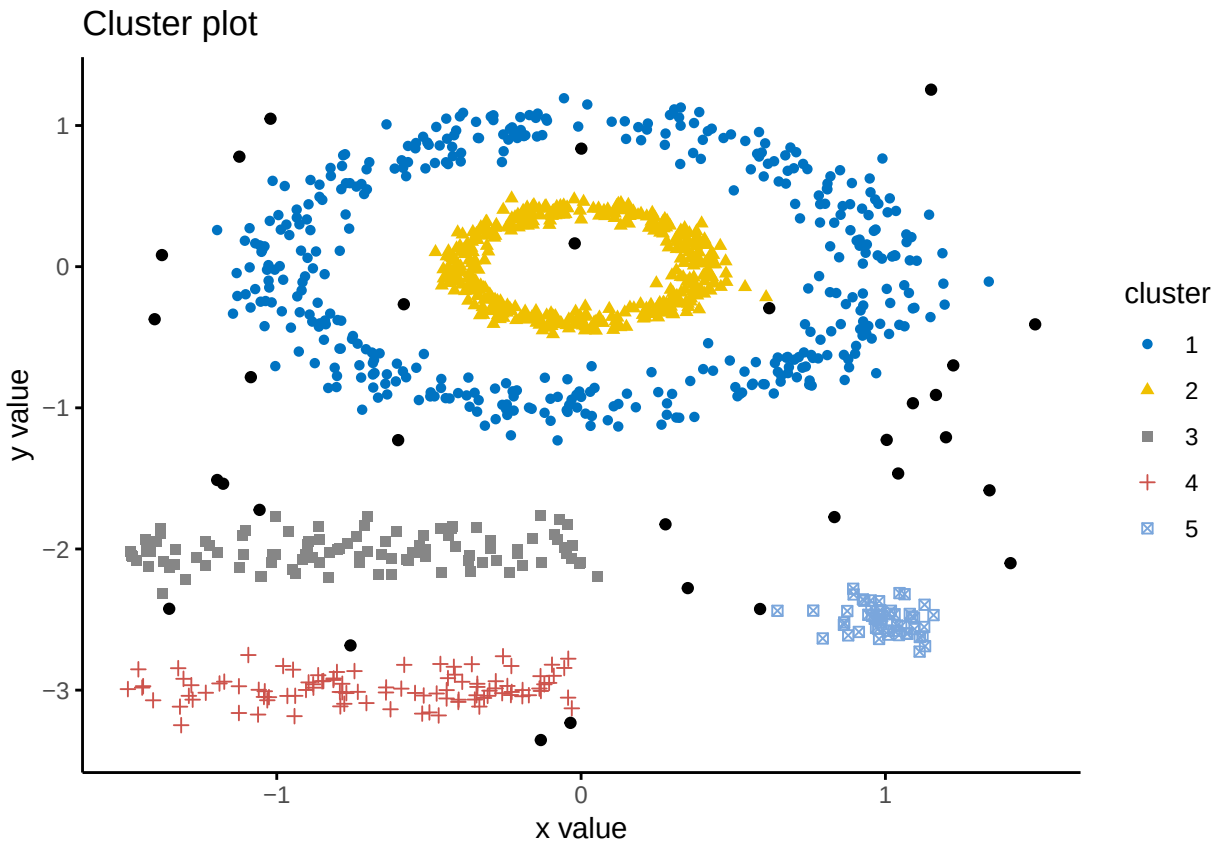
Not the five clusters we can see visually. Also since each point must be in a cluster, some of the points that appear to be noise are included. Other methods such DBSCAN work better on this type of data.

```
library(dbscan)
```

```
## Warning: package 'dbscan' was built under R version 3.5.3
```

```
db <- dbscan(Small_df, eps = 0.15, minPts = 5)
```

```
# Plot DBSCAN results
fviz_cluster(db, data = Small_df, stand = FALSE,
             ellipse = FALSE, show.clust.cent = FALSE,
             geom = "point",palette = "jco", ggtheme = theme_classic())
```



In high dimension K-means is going to have problems because of the curse of dimensionality.

Homework

Use K-means to group the customers in the market segmenting data into the appropriate marketing groups.

```
market_seg <- readRDS("ws_customers.rds")
market_seg
```

```
##      Milk Grocery Frozen
## 1   11103   12469    902
## 2    2013    6550    909
## 3    1897    5234    417
## 4    1304    3643   3045
## 5    3199    6986   1455
## 6    4560    9965    934
## 7     879    2060    264
## 8    6243    6360    824
## 9   13316   20399   1809
## 10   5302    9785    364
## 11   3688   13829    492
## 12   7460   24773    617
## 13  12939    8852    799
## 14  12867   21570   1840
## 15   2374    2842   1149
```

```
## 16  1020    3007    416
## 17 20655   13567   1465
## 18  1492    2405  12569
## 19  2335    8280   3046
## 20  3737   19172   1274
## 21   925    2405   4447
## 22  1795    7647   1483
## 23 14982   11924    662
## 24  1375    2201   2679
## 25  3088    6114    978
## 26  2713    3558   2121
## 27 25071   17645   1128
## 28  3696    2280    514
## 29  1897    5167   2714
## 30   713    3315   3703
## 31   944   11593    915
## 32  3587    2464   2369
## 33  1610    1431   3498
## 34   899    1664    414
## 35  2209    3389   7849
## 36  1486    4583   5127
## 37  1786    5109   3570
## 38 14881   26839   1234
## 39  3216    1447   2208
## 40   928    2743  11559
## 41  6817   10790   1365
## 42  1511    1330    650
## 43  1347    2611   8170
## 44   333    7021  15601
## 45  1188    5332   9584
```