k-means and k-medoids clustering methods for two-dimensional data using R

Francisca A. Soares dos Santos B.

```
# Install pacman ("package manager") if needed
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```
pacman::p_load(
  pacman,          # Load/unload packages
  tidyverse,       # Data manipulation
  factoextra       # ggplot2-based visualization of partitioning methods
  )
```

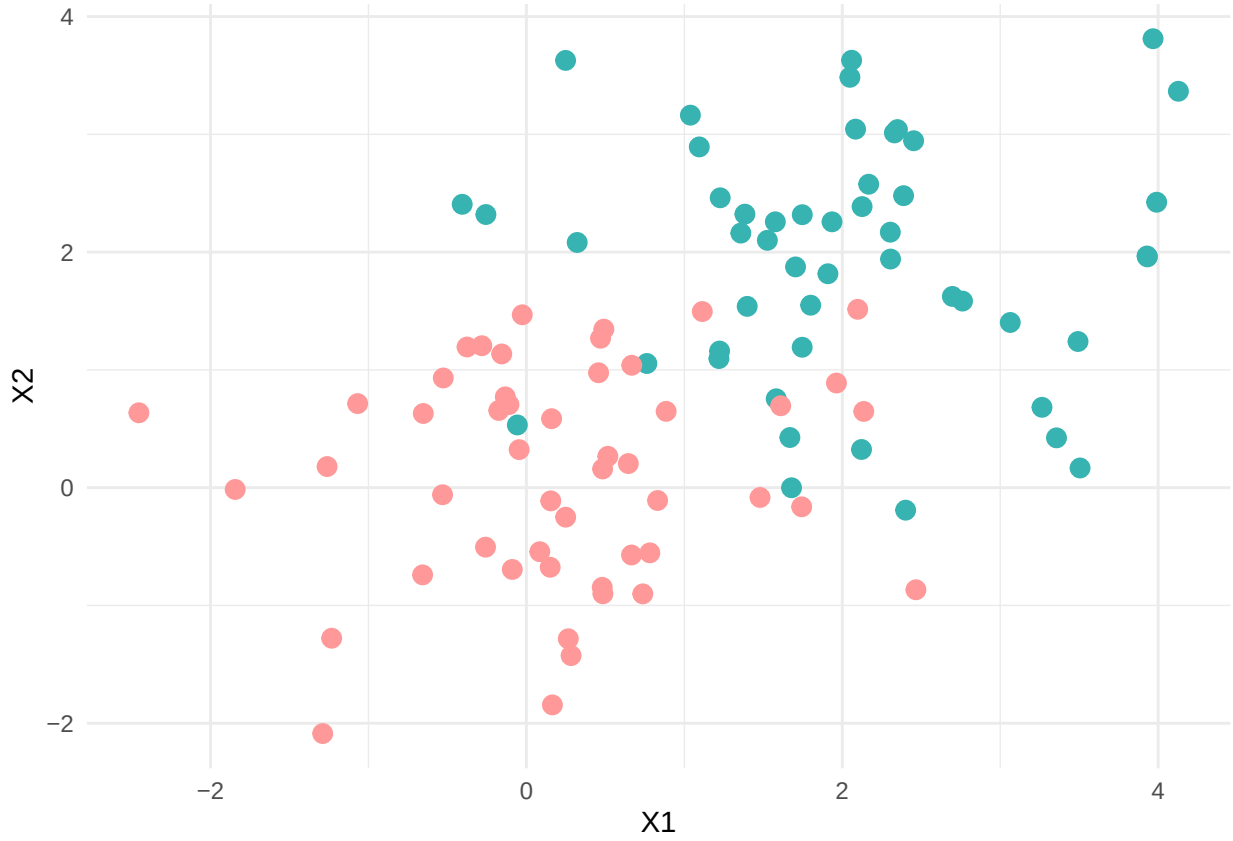
```
library(tidyverse)
library(factoextra)
library(cluster)
```

Dataset

The code bellow generate 2 different sets of 100 normally distributed random values. Each dataset is composed of 50 (`x1, x2`) points and represents one group or category of data. The data is then bound together to form a total of 100 points or observations.

```
set.seed(333)
dist1 = matrix(rnorm(50*2, mean = 2, sd = 1.1), ncol = 2)
dist2 = matrix(rnorm(50*2, mean = 0, sd = 1), ncol = 2)

df_X1 <- data.frame(dist1)
df_X2 <- data.frame(dist2)

# label the points in two different categories (first 50 != last 50)
# data belongs to a same category if they have same mean and standard deviation
y = c(rep(-1,50), rep(1,50))

df_x <- rbind(df_X1, df_X2)
names(df_x) <- c("X1", "X2")

df <- df_x %>%
  mutate(color = case_when(y == -1 ~ "#37b3b2", .default = "#FF9999"))

df %>%
  ggplot(aes(x=X1, y=X2)) +
  geom_point(size=3, colour=df$color) +
  theme_minimal() +
  xlab("X1") +
```

```
  ylab("X2")
```



K-means clustering:

Apply the R function `kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = "Hartigan-Wong", trace=FALSE)` from `{stats}` package to categorize data using k-means clustering method.

In this example the `kmeans()` parameters were set as follow: number of clusters `k` is equal to 2, number of random initial sets of center values `nstart` is equal to 20, and maximum number of iteration `iter.max` is equal to 10 (as default).

Note: The `nstart` argument is used to run the `kmeans()` function with multiple sets of initial cluster center values (as many as the value passed to the argument), where each set of center values is a random configurations of initial cluster assignments. For a `nstart` value greater than one the `kmeans()` function will report only the best results, i.e. the ones with smallest sum of squares from points to the assigned cluster centers.

```
k2m_out=kmeans(df_x, centers = 2, nstart = 20)
```

The output of the fitted method are the resulting components (clusters centers, clustering vector, and within-cluster sum of square)

```
k2m_out
```

```
## K-means clustering with 2 clusters of sizes 50, 50
##
## Cluster means:
##           X1        X2
```

```
## 1 0.08673083 0.1367451
## 2 2.15689187 1.9155827
##
## Clustering vector:
##   [1] 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2
##  [38] 2 2 1 2 2 2 2 2 2 2 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
##  [75] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 84.68474 90.66399
##  (between_SS / total_SS =  51.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

`km_out$tot.withinss` returns the **total within-cluster sum of squares**

```
k2m_out$tot.withinss
```

```
## [1] 175.3487
```

Call `km_out$cluster` to print the resulting **cluster assignments** of the 100 observations. Observe how the 2 clusters were determined by the algorithm:

```
k2m_out$cluster
```

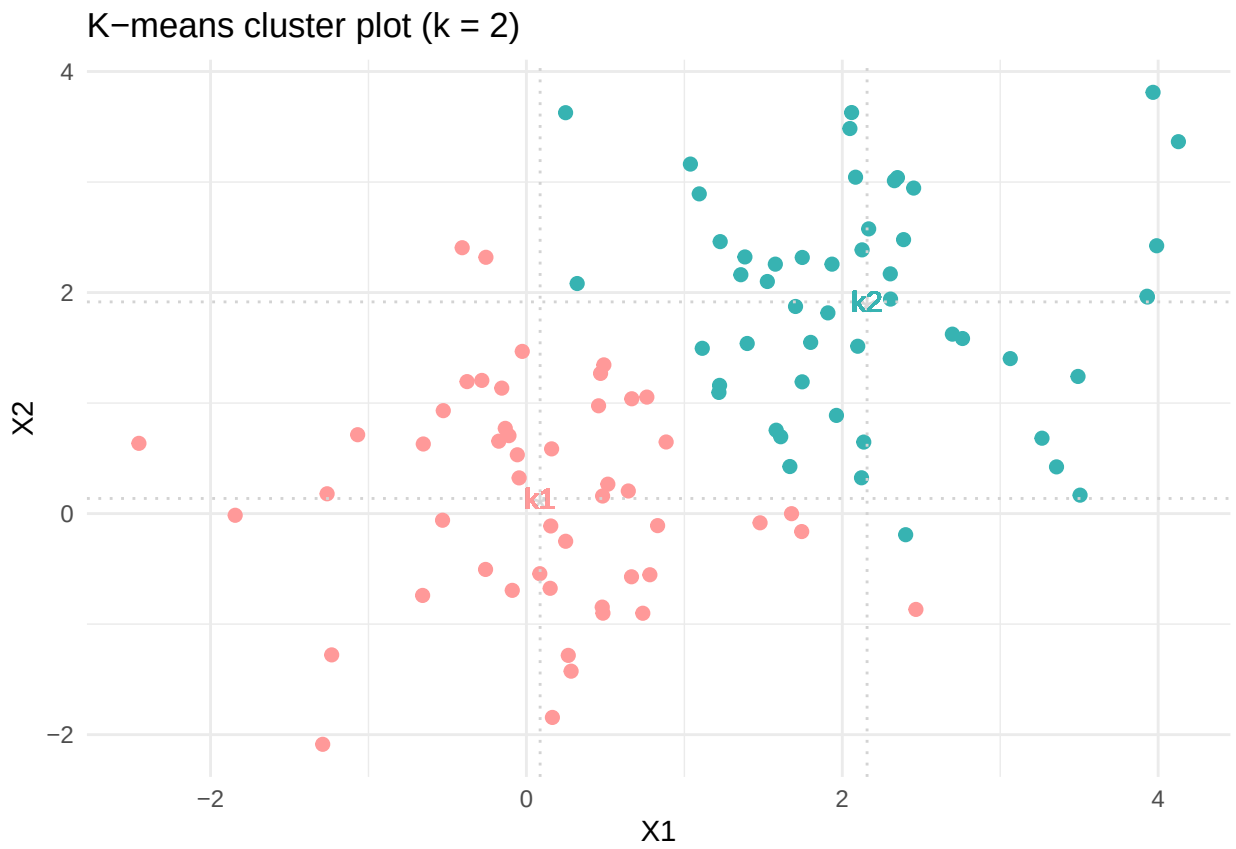
```
##   [1] 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2
##  [38] 2 2 1 2 2 2 2 2 2 2 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
##  [75] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1
```

Plot the result of `kmeans()` clustering:

Note the misclassification of an outlier (left side of chart – shown in the previous chart).

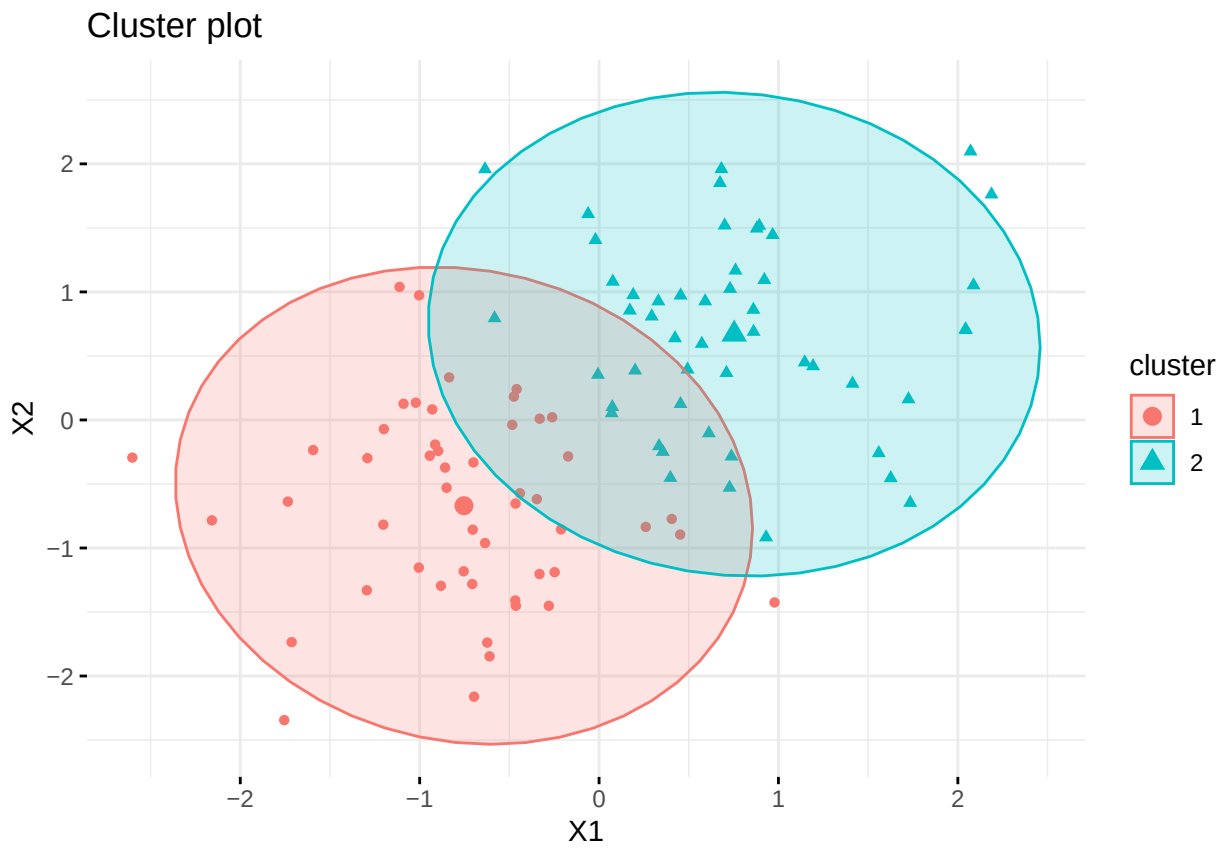
Note also that the centroids k1 and k2 at the chart don't belong to the data points

```
df <- df_x %>%
  add_column(color = k2m_out$cluster) %>%
  mutate(type = case_when(color == 2 ~ "#37b3b2", .default = "#FF9999"))

df %>%
  ggplot(aes(X1, X2)) +
  geom_point(colour = df$type, size = 2) +
  geom_point(data = as.data.frame(k2m_out$centers), shape = "*", size = 6, colour = "gray", alpha = 0.6
  geom_vline(xintercept = 2.15689187, colour="lightgray", linetype="dotted") +
  geom_hline(yintercept = 1.9155827, colour="lightgray", linetype="dotted") +
  geom_vline(xintercept = 0.08673083, colour="lightgray", linetype="dotted") +
  geom_hline(yintercept = 0.1367451, colour="lightgray", linetype="dotted") +
  geom_text(label="k1", x=0.08673083, y=0.1367451, color="#FF9999") +
  geom_text(label="k2", x=2.15689187, y=1.9155827, color="#37b3b2") +
  theme_minimal() +
  labs(title = "K-means cluster plot (k = 2)", x="X1", y="X2")
```



Using the function `fviz_cluster` from `{factoextra}` package to plot the result:

```
fviz_cluster(k2m_out, df_x, geom = "point", ellipse.type = "norm", ggtheme = theme_minimal())
```



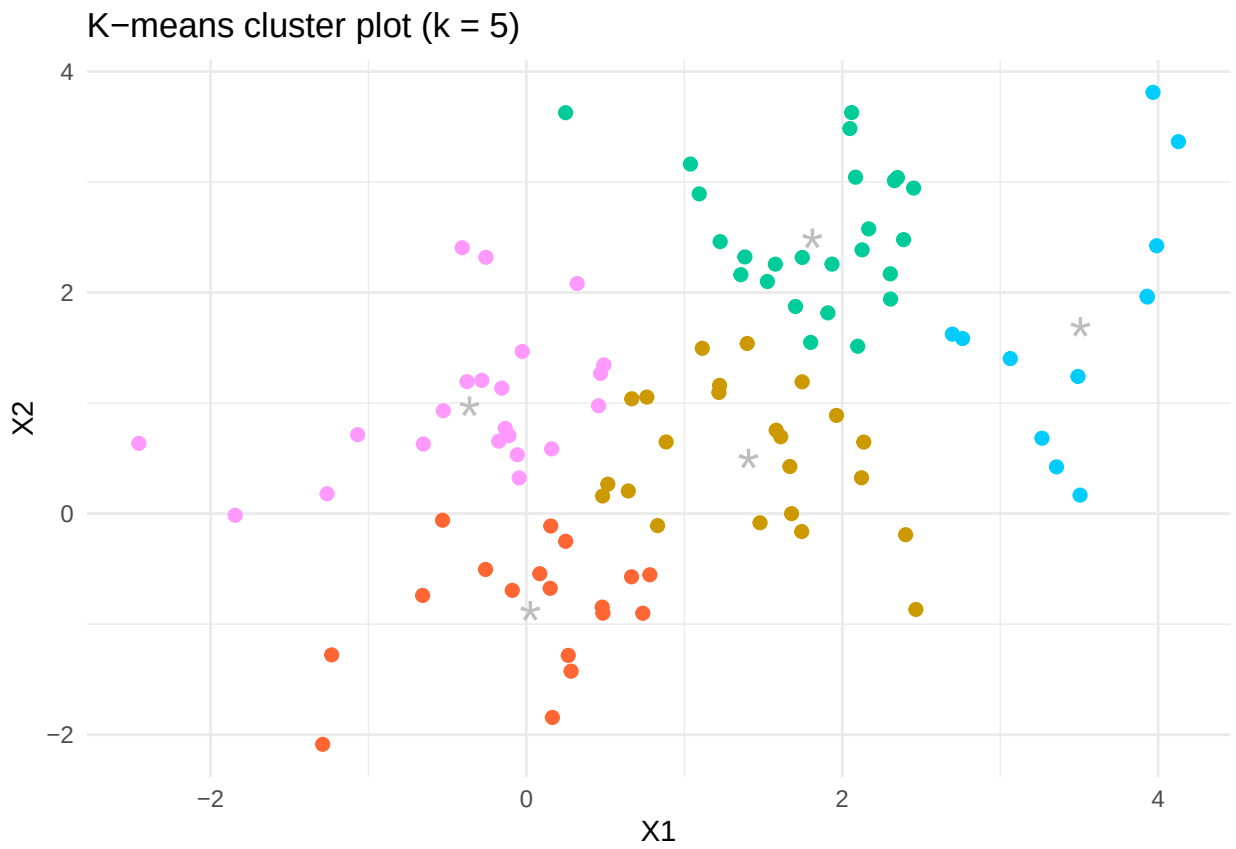
Observations:

1) The number of variables: Two variables yields a two-dimensional plot: In the chart above one feature is plotted on the x-axis and the other feature is plotted on the y-axis. If there were more than two variables the solution would be to perform PCA first and then plot the first two principal components score vectors.

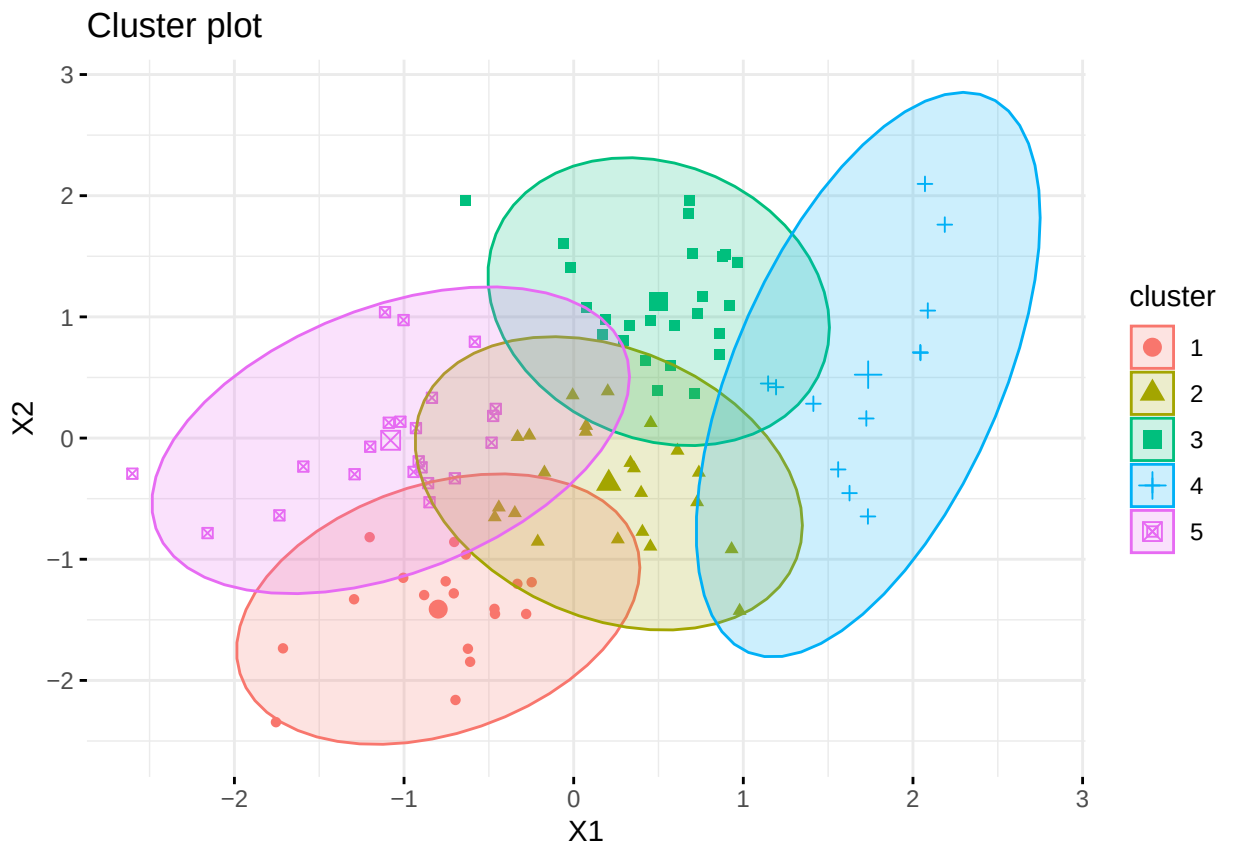
2) The value of `k`, or number of clusters: In this example we knew the value of `k` because we generated the data, however, in real data, usually the number of clusters is unknown. The number of clusters centers may affect the algorithm performance. That is easily noted by observing changes in the value of within-cluster sum of squares (or **within-cluster variation**).

To illustrate this last observation, let's, run `kmeans()` to `k` equal to 5 and the same dataset:

```
k5m_out=kmeans (df_x, centers = 5, nstart = 20)

df5 <- df_x %>%
  add_column(color = k5m_out$cluster) %>%
  mutate(type = case_when(color == 1 ~ "#FF6633", color == 2 ~ "#CC9900", color == 3 ~ "#00CC99", color

df5 %>%
  ggplot(aes(X1, X2)) +
  geom_point(colour = df5$type, size = 2) +
  geom_point(data = as.data.frame(k5m_out$centers), shape = "*", size = 7, colour = "gray") +
  theme_minimal() +
  labs(title = "K-means cluster plot (k = 5)", x="X1", y="X2")
```




```
#plotting with `fviz_cluster` from {factoextra}
fviz_cluster(k5m_out, df_x, geom = "point", ellipse.type = "norm", ggtheme = theme_minimal())
```



The resulting 5 **clusters assignments** were determined by the algorithm as follow:

```
k5m_out$cluster
```

```
##   [1] 3 4 5 3 5 3 4 4 3 3 2 3 3 3 3 2 2 3 4 2 2 4 2 3 2 4 4 4 3 5 3 3 4 2 2 3 3
##  [38] 3 3 5 3 3 3 3 3 4 3 2 4 4 5 2 5 2 1 1 5 5 1 5 5 1 5 5 3 1 1 1 2 1 1 5 1 5
##  [75] 2 2 5 1 2 1 1 5 5 1 5 1 1 2 1 2 1 2 5 2 2 2 2 5 5 5
```

Clusters centers:

```
k5m_out$centers
```

```
##           X1         X2
## 1  0.02439829 -0.8486363
## 2  1.40591538  0.5293952
## 3  1.81016997  2.5207814
## 4  3.50734163  1.7208526
## 5 -0.36074713  1.0020904
```

Note how the total within-cluster sum of squares changed (recall that it's equal to 175.3487 for `k = 2`)

```
k5m_out$tot.withinss
```

```
## [1] 78.97523
```

Minimizing the within-cluster variation

The total within-cluster sum of squares (within-cluster variation) is the value that we seek to minimize by performing k-means clustering. The best solution is that for which the within-cluster variation is smallest and provides better separation between the clusters.

1) By increasing the number of initial cluster assignments:

To illustrate how the number of initial cluster assignments affects the algorithm performance we run k-means to different values of `nstart` (namely, `nstart=1` and `nstart=4`) and compare the resulting total within-cluster sum of squares.

```
set.seed(73)
km_out_nstart1 = kmeans(df_x, 5, nstart = 1)
km_out_nstart1$tot.withinss
```

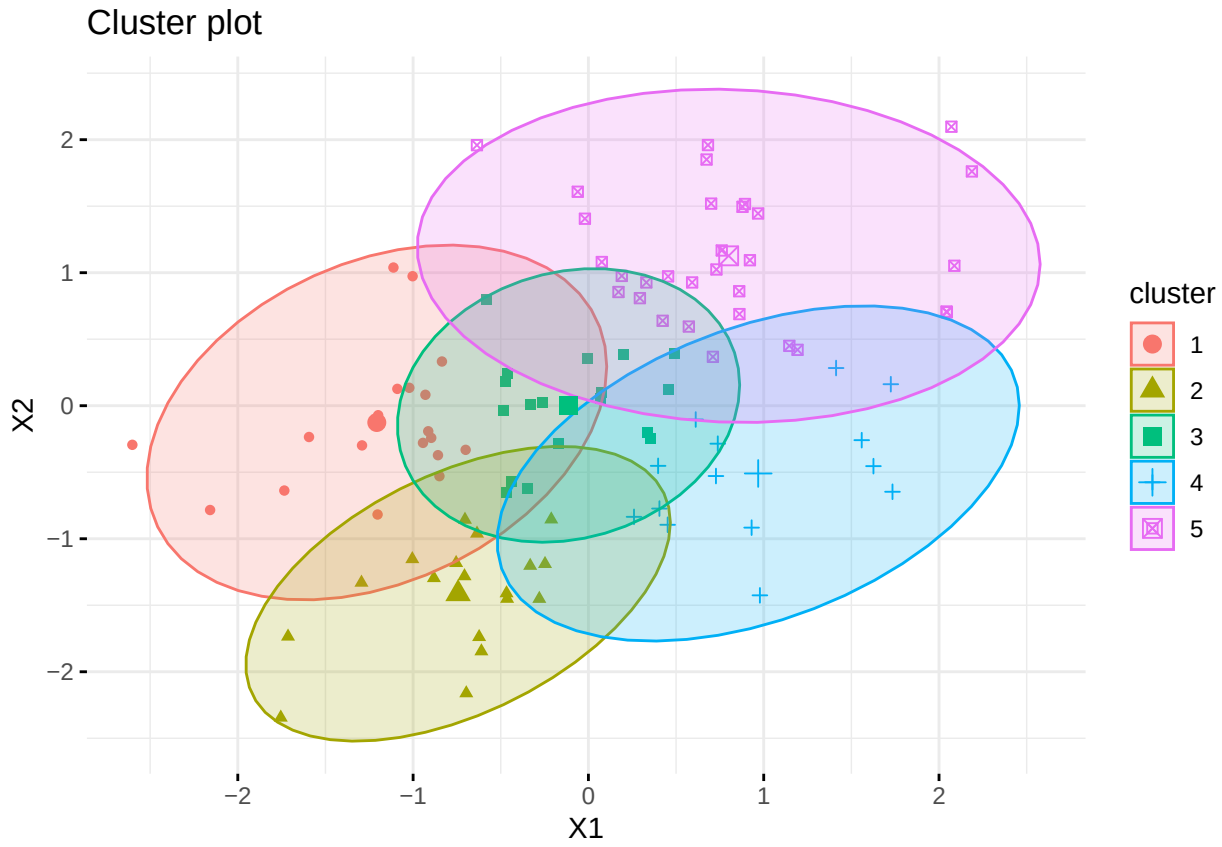
```
## [1] 87.97444
```

```
km_out_nstart4 = kmeans(df_x, 5, nstart = 4)
km_out_nstart4$tot.withinss
```

```
## [1] 79.49242
```

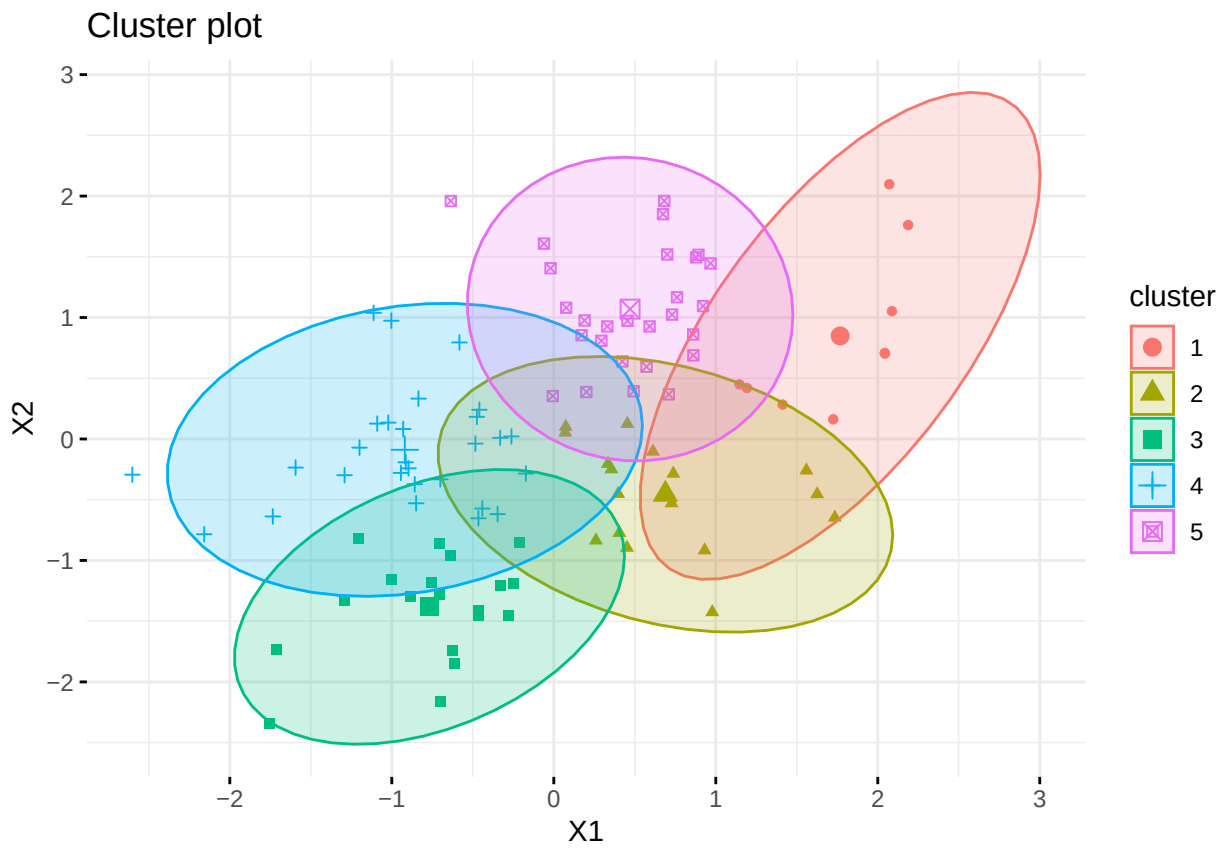
Resulting cluster configuration for `nstart=1`:

```
fviz_cluster(km_out_nstart1, df_x, geom = "point", ellipse.type = "norm", ggtheme = theme_minimal())
```



Resulting cluster configuration for `nstart=4`:

```
fviz_cluster(km_out_nstart4, df_x, geom = "point", ellipse.type = "norm", ggtheme = theme_minimal())
```



2) By changing the number of clusters:

Finding the best value of k using the elbow method

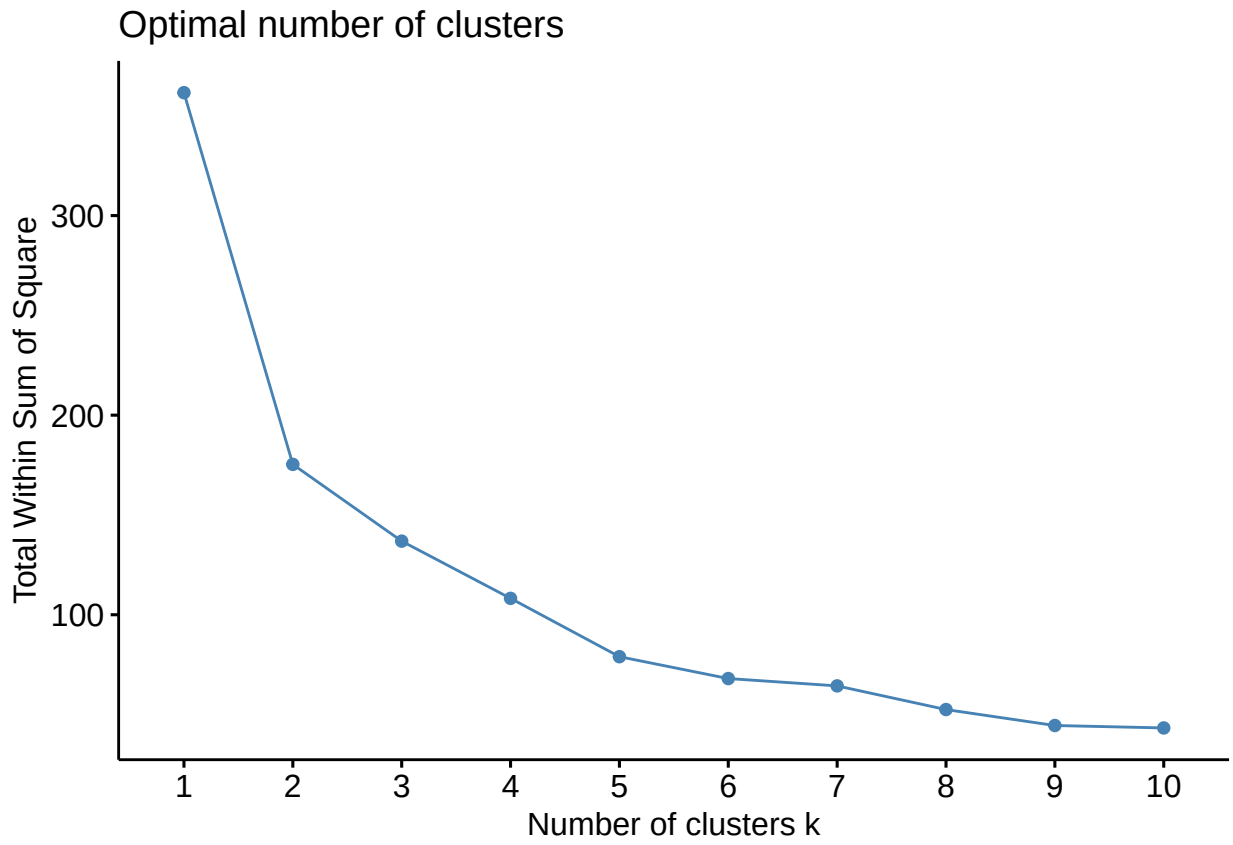
Compute k-means for a given range of `k` values and plot the `Total within-cluster sum of squares` vs `k`. The optimal value of `k` can be picked by finding the elbow in the plot, or point where the total sum of squares begins to level off (or the line flattens).

The algorithm runs k-means clustering for different values of `k` within a given range and memorize their respective within cluster sum of squares.

In the plot `Total within-cluster sum of squares` vs `k`, the optimal value of `k` is in the point where the total sum of squares begins to level off (in `k = 2` in this example).

The R function `fviz_nbclust()` automatically computes k-mean and does the plot of the `Total within-cluster sum of squares` vs `k`. The values of `k` are determined by the `k.max` parameter (which represents the maximum number of clusters to consider, and has default value equal to 10)

```
fviz_nbclust(df_x, FUNcluster = kmeans, method = "wss")
```



---

k-medoids: alternative method to deal with outliers

The algorithm is very similar to k-means algorithm differing only by the fact that in k-medoids the cluster centroids are the medians instead of the means. The consequence of this change is that the centroids are points within the dataset.

In R, k-medoids is implemented by the `pam()` function (Partitioning Around Medoids).

```
pam_out <- pam(df_x, k = 2, metric = "euclidean")
pam_out
```

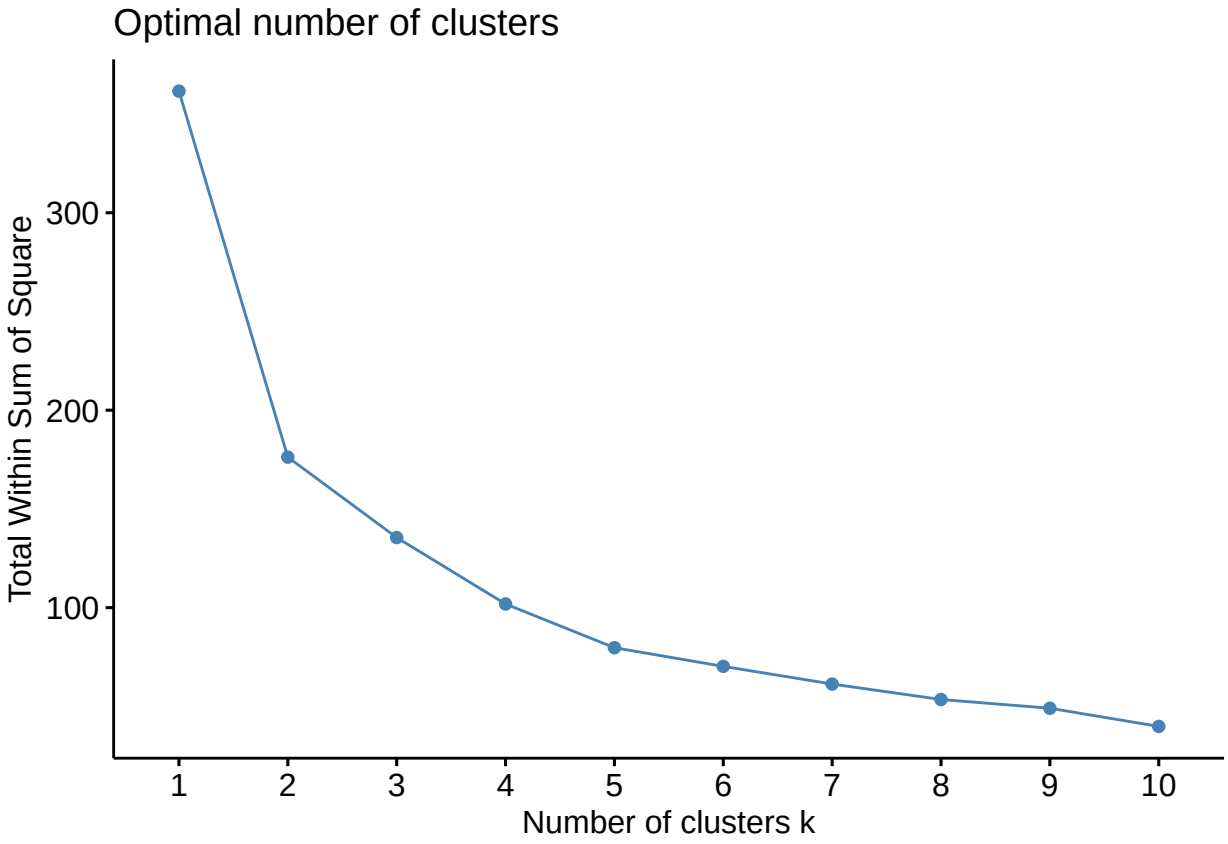
```
## Medoids:
##      ID          X1        X2
## [1,]  1  1.90890720 1.8162242
## [2,] 77 -0.04614425 0.3230553
## Clustering vector:
##   [1] 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
##  [38] 1 1 2 1 1 1 1 1 1 1 2 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2
##  [75] 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 2 2
## Objective function:
##    build     swap
## 1.334614 1.178313
##
## Available components:
##  [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"
```

```
##  [6] "clusinfo"   "silinfo"    "diss"       "call"       "data"
```

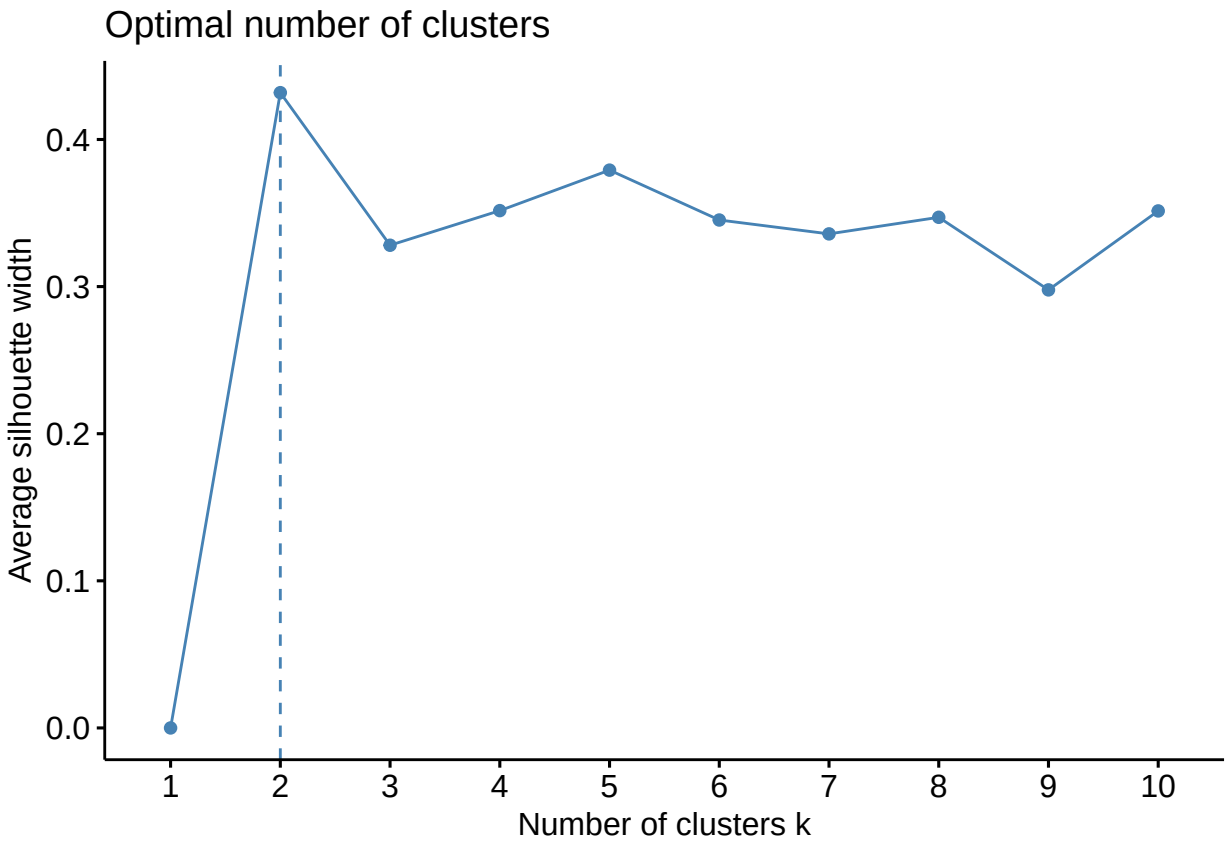
Estimating the optimal number of clusters

There are different methods that can be used for estimating the optimal number of clusters. We can use the R function `fviz_nbclust()` to automatically compute and plotting the metric value vs the value of `k`. Possible metrics for estimating the optimal number of clusters are: "*silhouette*" (for average silhouette width), "*wss*" (for total within sum of square) and "*gap_stat*" (for gap statistics)

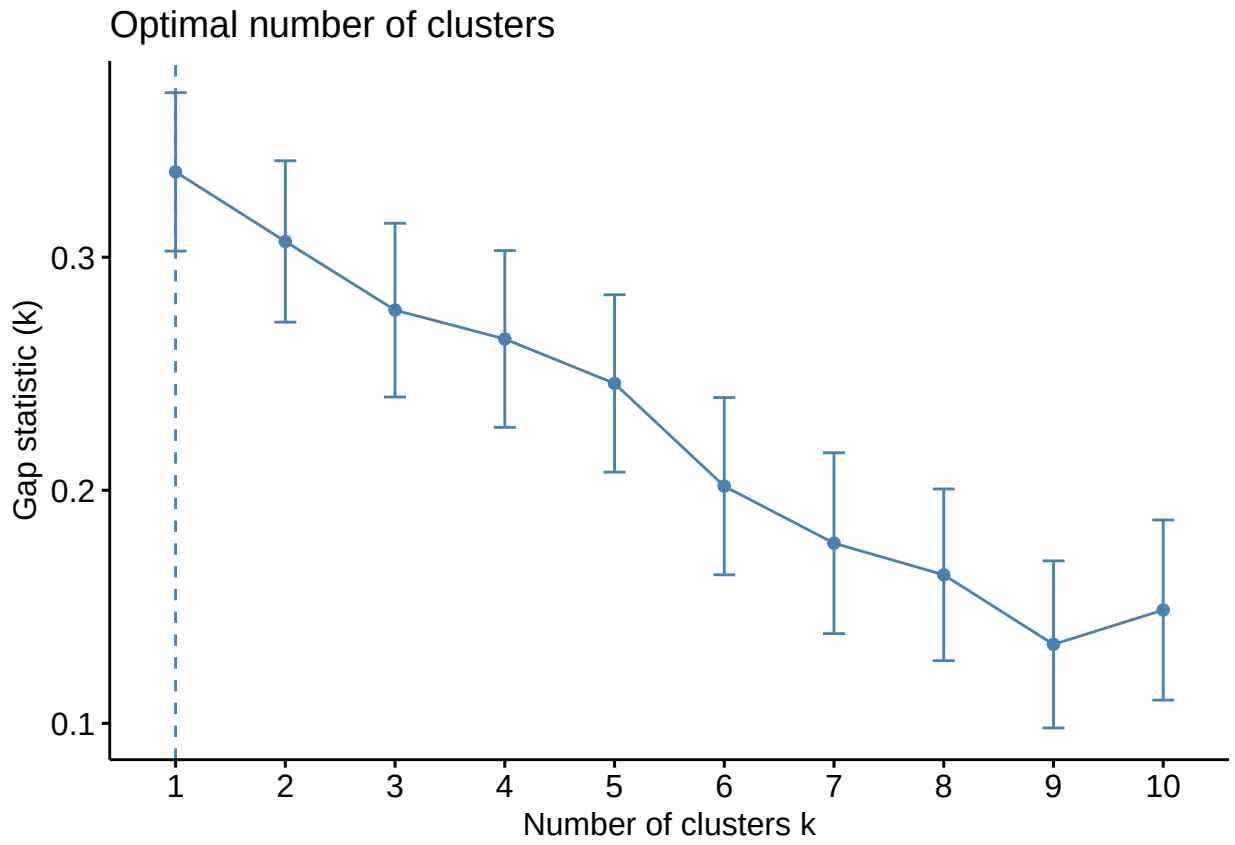
```
# within cluster sums of squares for estimating optimal number of clusters
fviz_nbclust(df_x, FUNcluster = pam, method = "wss")
```



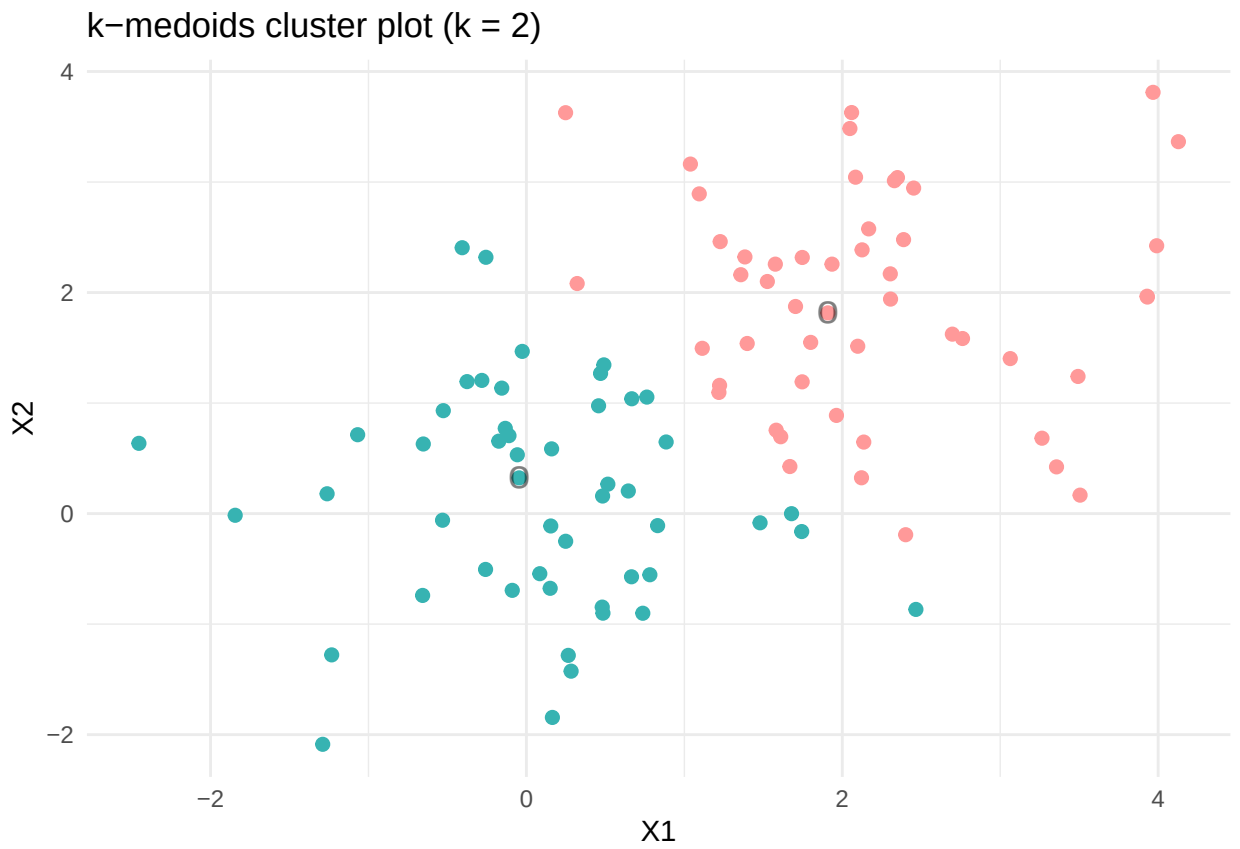

```
# average silhouette for estimating optimal number of clusters
fviz_nbclust(df_x, FUNcluster = pam, method = "silhouette")
```



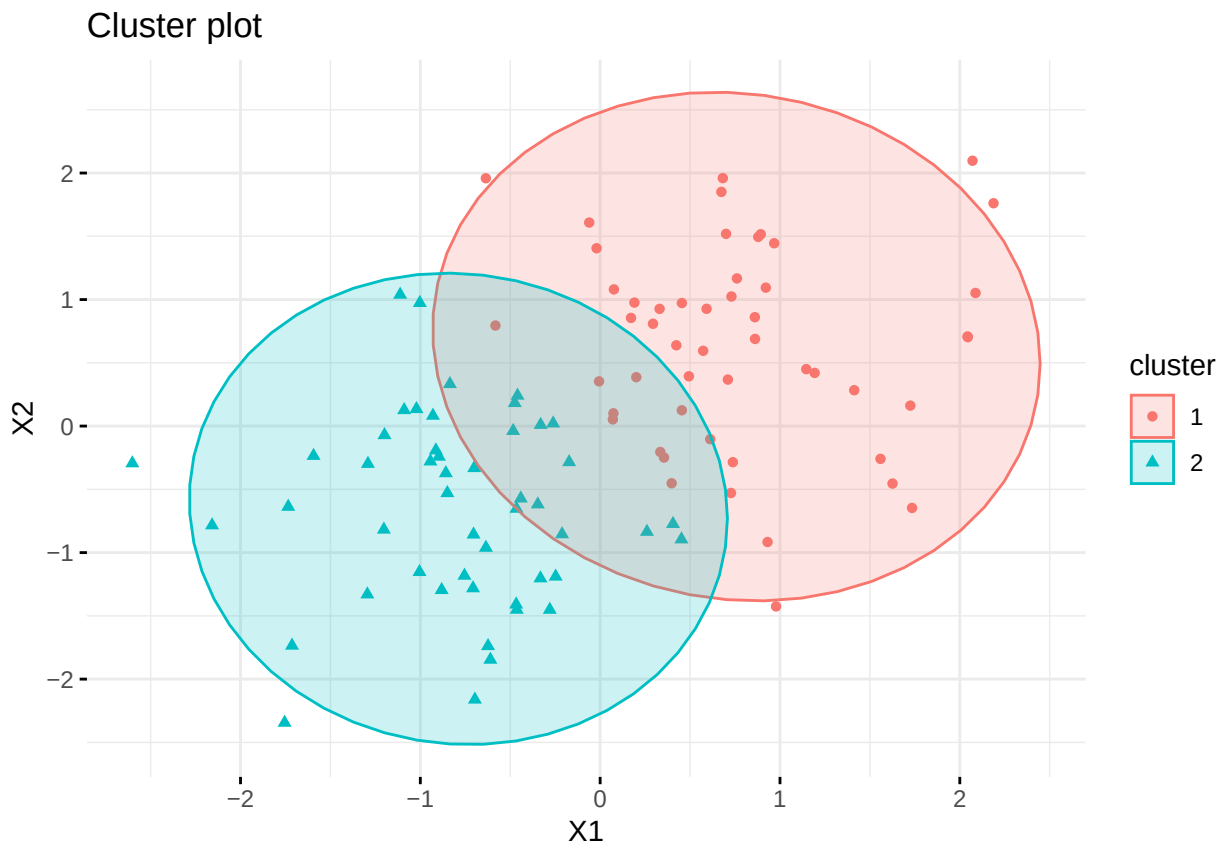

```
# Gap Statistic for estimating optimal number of clusters
fviz_nbclust(df_x, FUNcluster = pam, method = "gap_stat")
```




```
dfm <- df_x %>%
  add_column(color = k2m_out$cluster) %>%
  mutate(type = case_when(color == 1 ~ "#37b3b2", .default = "#FF9999"))

dfm %>%
  ggplot(aes(X1, X2)) +
  geom_point(colour = dfm$type, size = 2) +
  geom_point(data = as.data.frame(pam_out$medoids), shape = "o", size = 5, colour = "black", alpha = 0.5
  theme_minimal() +
  labs(title = "k-medoids cluster plot (k = 2)", x="X1", y="X2")
```




```
#plotting with `fviz_cluster` from {factoextra}
fviz_cluster(pam_out, df_x, geom = "point", ellipse.type = "norm", ggtheme = theme_minimal())
```


Cluster plot
X2
X1
2
1
0
−1
−2
−2
−1
0
1
2
cluster
1
2