Cluster Analysis Portfolio

Rachel Wood

2023-02-16

Task 1

For this task, I will perform clustering the `prestige` dataset. We first load the data and remove observations with missing values:

```
library(car)
library(dplyr)

data <- na.omit(Prestige)
```

First scale our data and remove the `type` column (we will use this later to see how the job type relate to the clusters):

```
type <- data$type

X <- scale(data[,-6])
```

Agglomerative Clustering

For this part of the task we use the `hclust()` function to perform agglomerative clustering. ### Choosing a distance measure

Since all our variables are continuous, we can use the `dist()` function to compute the Euclidean distances between observations:

```
X_dist <- dist(X)
```

We now use the `hclust` function to perform agglomerative clustering, setting the method to `"complete"`. This means the distance between an old cluster t and a new cluster (r, s) made up of previous clusters r and s is given by:

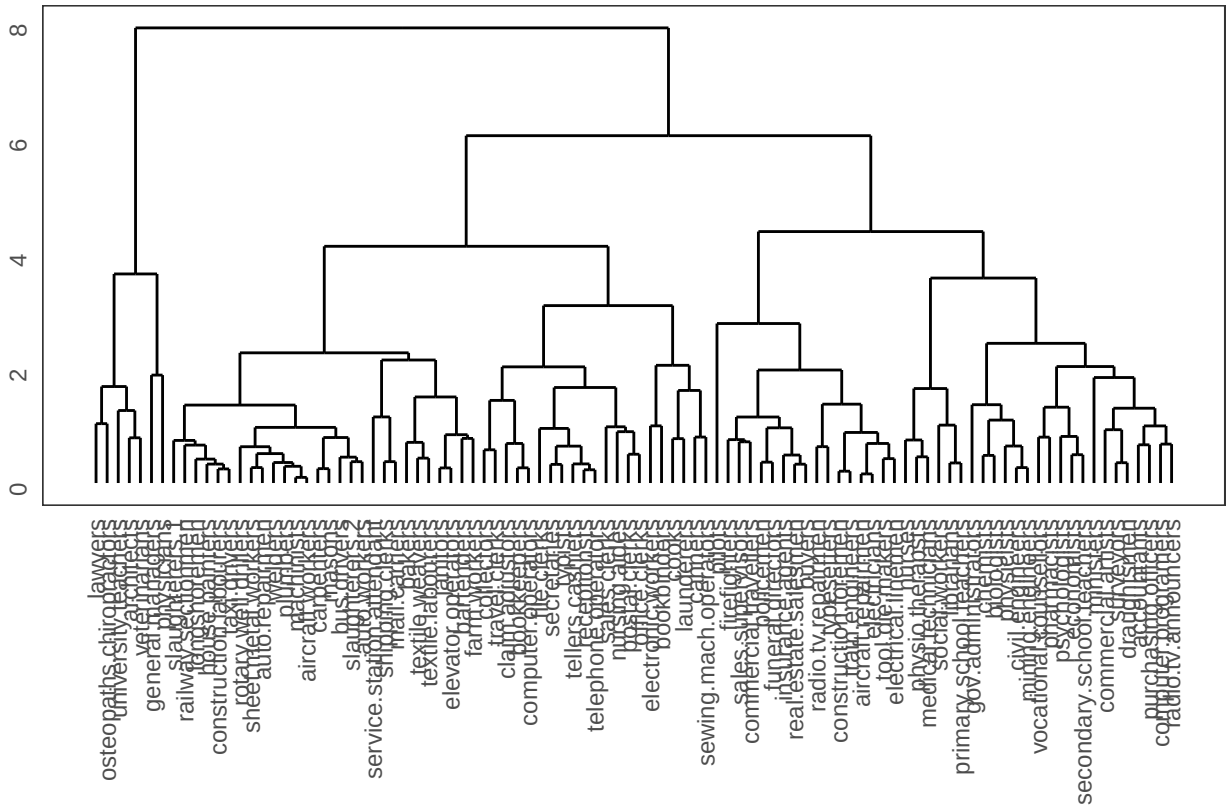
$$d_{t,(r,s)} = \frac{1}{2}d_{rt} + \frac{1}{2}d_{st} + \frac{1}{2}|d_{rt} - d_{st}| = \max\{d_{rt}, d_{st}\}$$

i.e. the euclidean distance between the two furthest members of the cluster. We use this as when determining the number of clusters to use, as when given a maximum acceptable distance between two members of a cluster, we can draw a line at the corresponding point on the y-axis of the dendrogram to obtain a clustering.

Choosing clusters

This gives us:

```
library(ggplot2)
library(ggdendro)

prestige_clusters <- hclust(X_dist, method = "complete", members = type)

ggdendrogram(prestige_clusters)
```



If we look at the areas for then largest jump in distance between the two furthest members of a cluster, it makes sense to either choose two or three clusters. Here we choose 3 clusters as 2 clusters would lead to the majority of the dataset being combined into to same cluster:

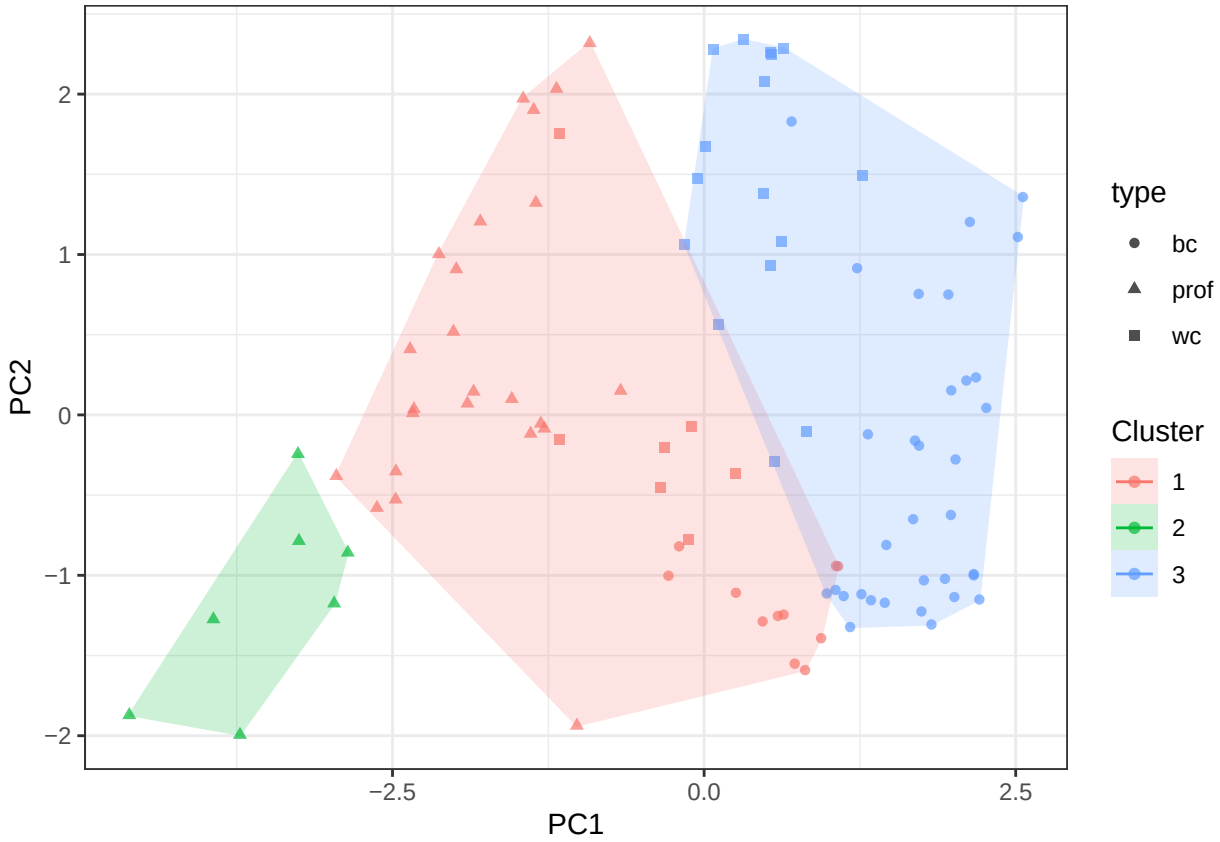
```
agglo_clusters <- cbind(data, "Cluster" = as.factor(cutree(prestige_clusters, k = 3)))
```

Visualisation of Results

We now use PCA to visualise a 2-dimensional representation of our clusters:

```
library(ggalt)
prestige_pca <- prcomp(~education+income+women+prestige+census, data = data, scale. = TRUE)

agglo_clusters <- agglo_clusters %>%
  as_tibble() %>%
  mutate("PC1" = prestige_pca$x[,1], "PC2" = prestige_pca$x[,2])
```

```
ggplot(agglo_clusters, aes(x = PC1, y= PC2)) +
  geom_point(aes(color = Cluster, shape = type), alpha = 0.7) +
  geom_encircle(aes(fill = Cluster, color = Cluster), alpha = 0.2, s_shape = 1, expand = 0)
```



We can see the resulting clusters are not entirely separate as there is some overlap, so we try K-means clustering in the hopes that it will perform better. ## K-Means Clustering

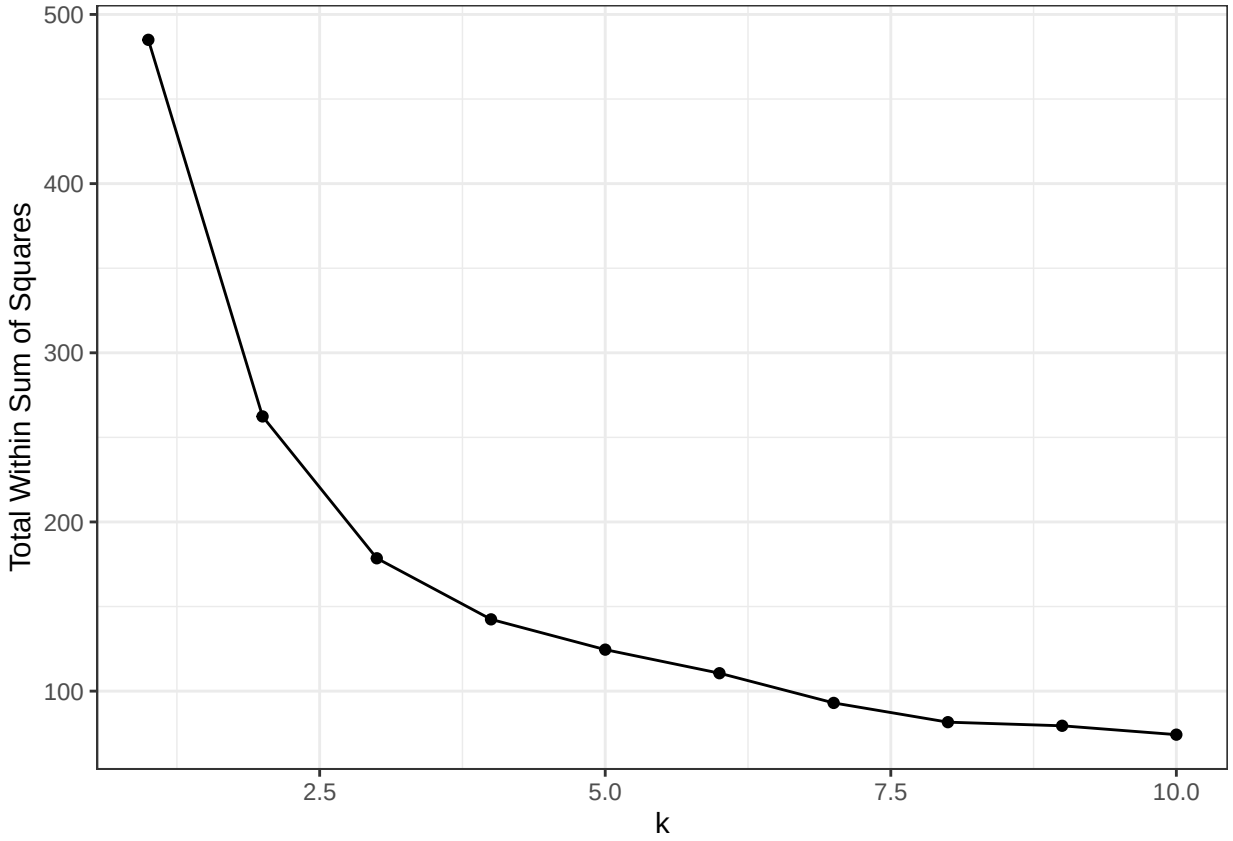
For this section of the task we use the `kmeans()` function. We again use the euclidean distance as is the default in K-means clustering.

Choosing Clusters

We implement the clustering using Lloyd's algorithm by setting `algorithm = "Lloyd"` in the `kmeans()` function. We try this for $k = 1, ..., 10$ and plot the total sum of squares. Since this values always decreases with k, we look for the point where the slope of the function changes the most.

```
K <- 10
sum_squares <- numeric(K)
fits <- vector(mode = "list", length = K)

for (k in 1:K){
  fit <- kmeans(X,
                centers = k,
                iter.max = 15,
                algorithm = "Lloyd")
  sum_squares[k] <- fit$tot.withinss
```

```
  fits[[k]] <- fit
}

sum_squares <- tibble(k = 1:K, totwss = sum_squares)

ggplot(data = sum_squares, aes(x = k, y = totwss)) +
  geom_point() +
  geom_line() +
  labs(y = "Total Within Sum of Squares")
```



Looking at this, we can see as with the agglomerative clustering, choosing $k = 2$ or $k = 3$ seems most appropriate. We again choose $k = 3$ clusters.

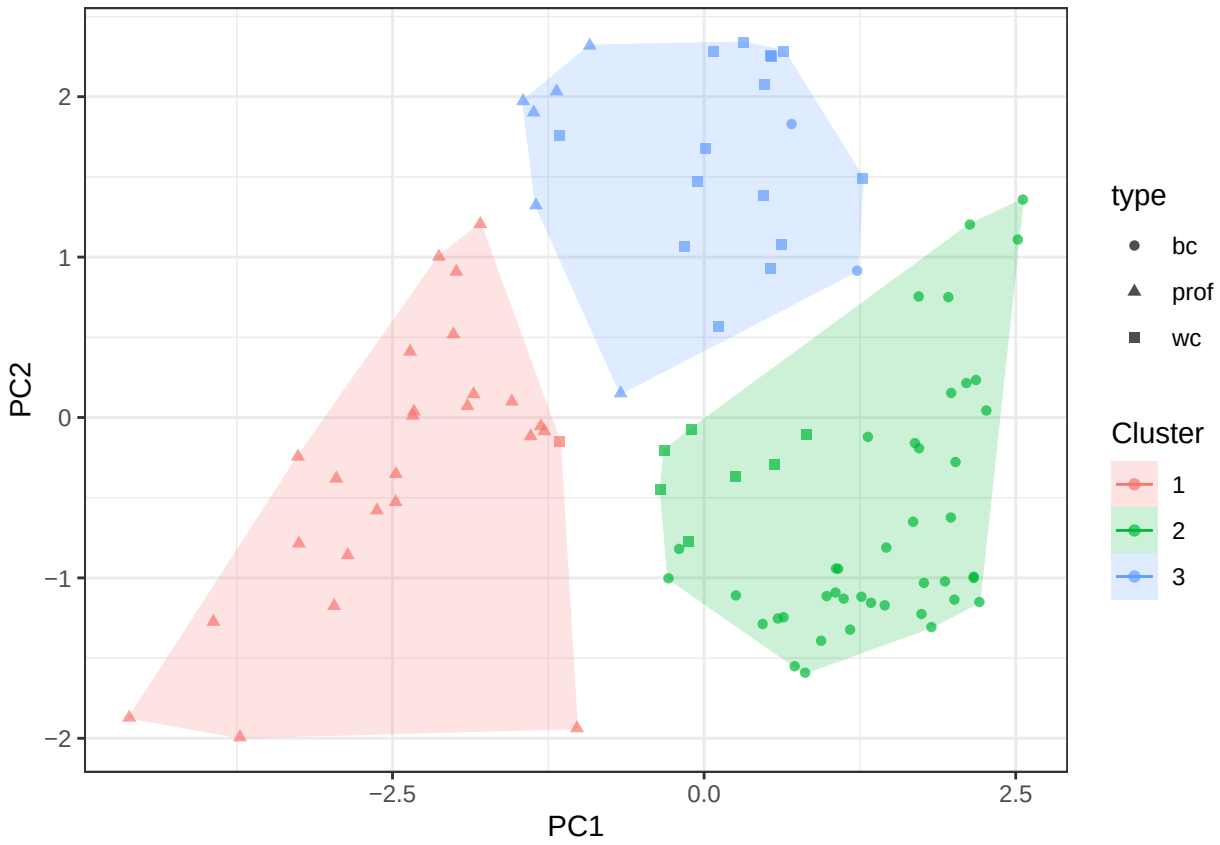
```
k_clusters <- cbind(data, "Cluster" = fits[[3]]$cluster)
```

Visualising Clusters

We again use the principle components obtained previously to visualise a two dimensional representation of the clusters:

```
k_clusters <- k_clusters %>%
  as_tibble() %>%
  mutate("PC1" = prestige_pca$x[,1], "PC2" = prestige_pca$x[,2]) %>%
  mutate(Cluster = as.factor(Cluster))
```

```
ggplot(k_clusters, aes(x = PC1, y= PC2)) +
  geom_point(aes(color = Cluster, shape = type), alpha = 0.7) +
  geom_encircle(aes(fill = Cluster, color = Cluster), alpha = 0.2, s_shape = 1, expand = 0)
```



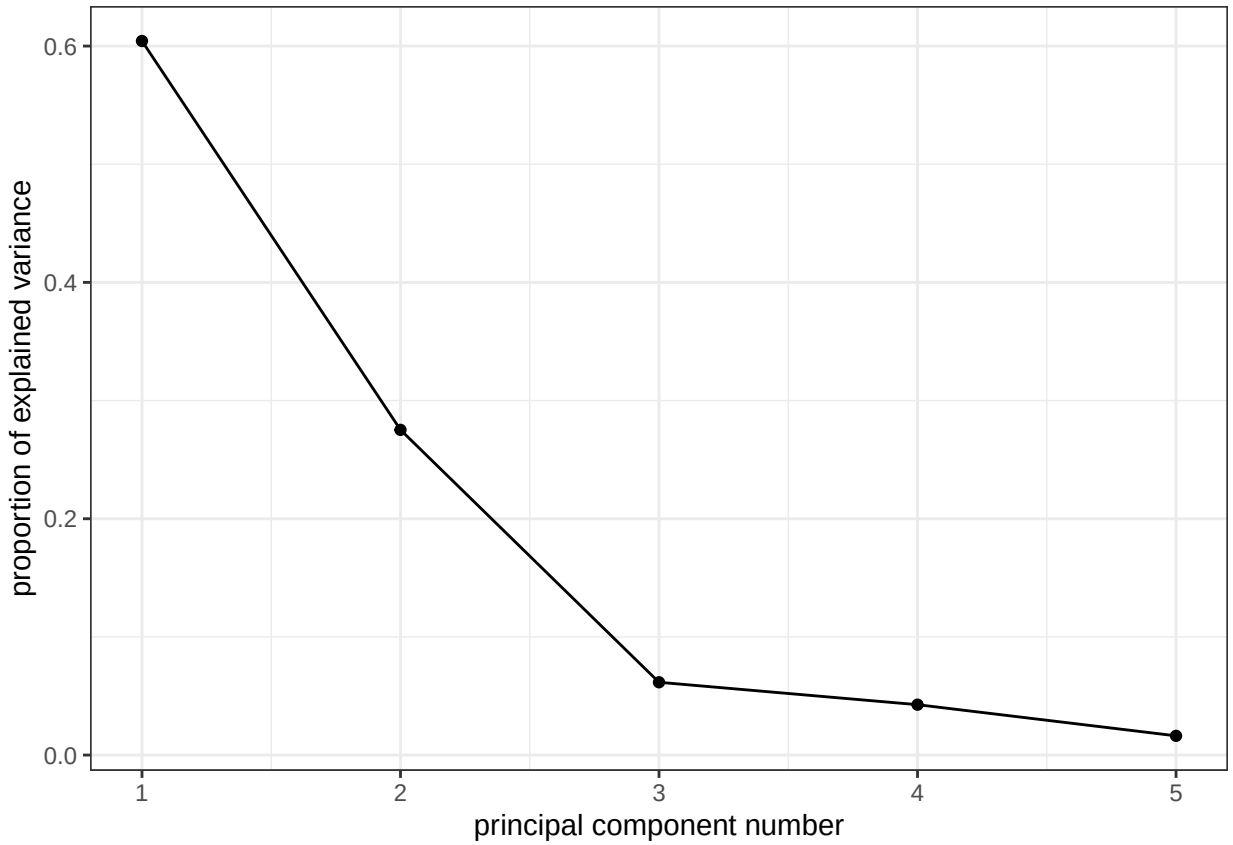
We can see the k-means clustering has performed a clearer separation of the data.

Performing clustering on PCA components

We now decide on a reduction of the dataset and perform the k-means algorithm again.

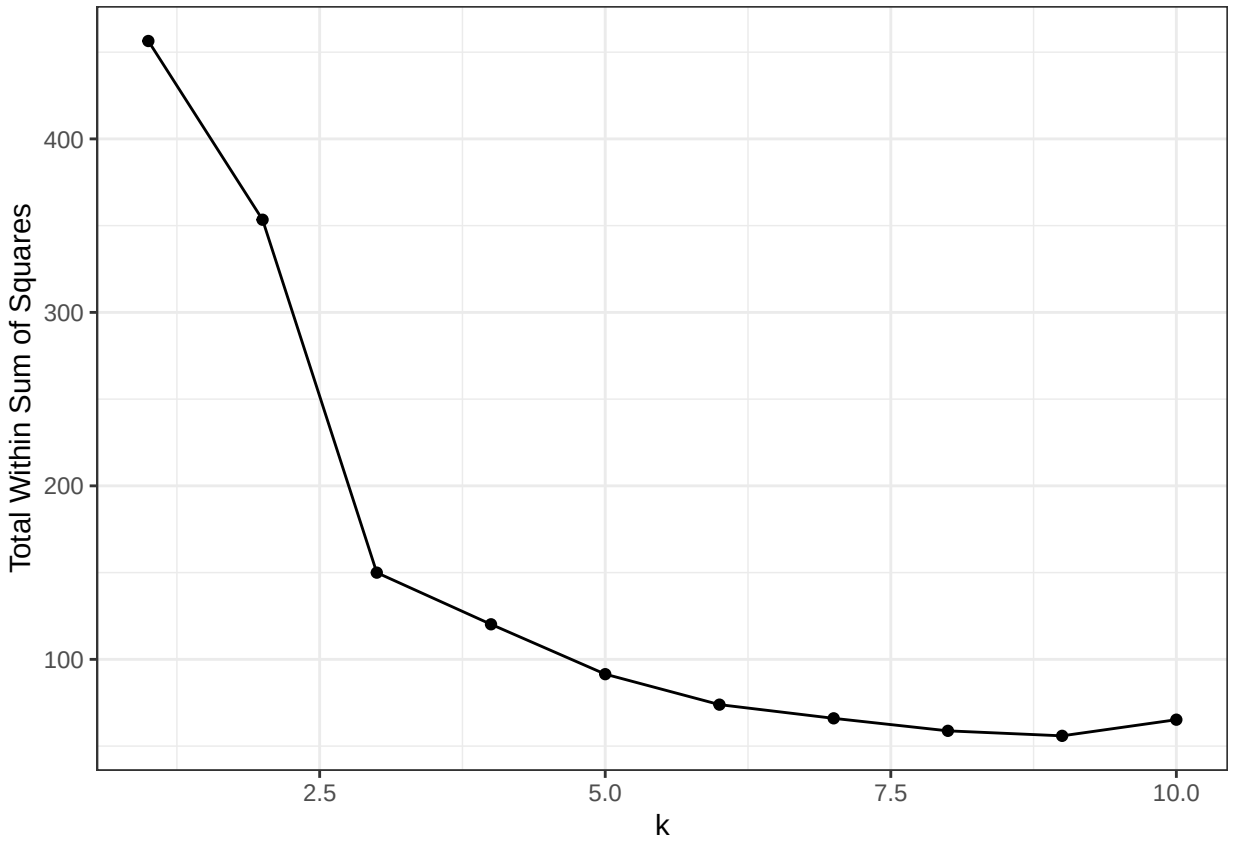
First, we use a scree plot to choose the $q < p$ principle components we will use:

```
library(ggbiplot)
ggscreeplot(prestige_pca)
```



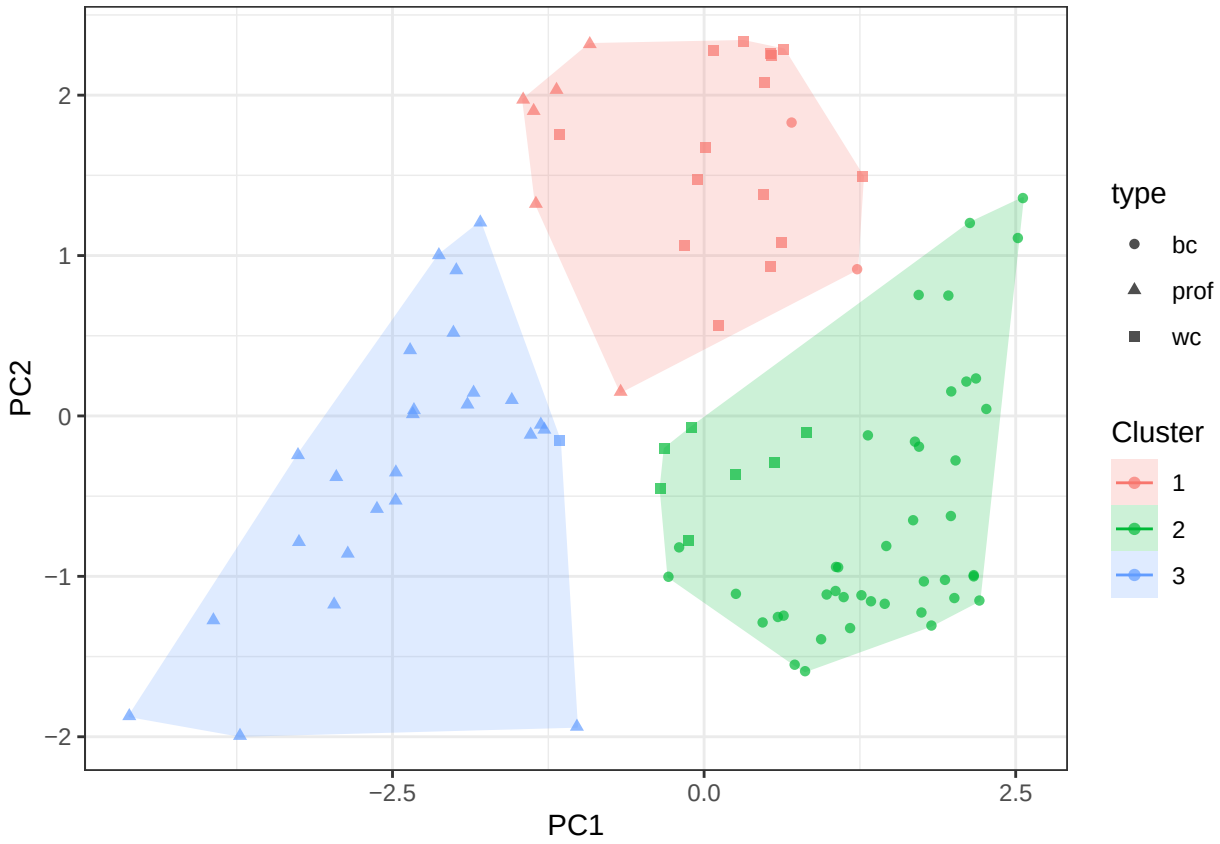
We can see from this the obvious choice for components to keep is 3. We now apply `kmeans()` function to these for $k = 1, \ldots , 10$ and plot the total within sum of squares as before:

```
prestige_pcs <- prestige_pca$x[,1:3]

pc_sum_squares <- numeric(K)
pc_k_fits <- vector("list",K)

for (k in 1:K){
  pc_fit <- kmeans(prestige_pcs,
                   centers = k,
                   iter.max = 15,
                   algorithm = "Lloyd")
  pc_sum_squares[k] <- pc_fit$tot.withinss

  pc_k_fits[[k]] <- pc_fit
}

pc_sum_squares <- tibble(k = 1:K, totwss = pc_sum_squares)

ggplot(data = pc_sum_squares, aes(x = k, y = totwss)) +
  geom_point() +
  geom_line() +
  labs(y = "Total Within Sum of Squares")
```



We cna see there is a sharp change in the slope at $k = 3$. We use 3 clusters and visualise the clusters using the first two principal components:

```
pc_k_clusters <- cbind(data, "Cluster" = pc_k_fits[[3]]$cluster)

pc_k_clusters <- pc_k_clusters %>%
  as_tibble() %>%
  mutate("PC1" = prestige_pca$x[,1], "PC2" = prestige_pca$x[,2]) %>%
  mutate(Cluster = as.factor(Cluster))

ggplot(pc_k_clusters, aes(x = PC1, y= PC2)) +
  geom_point(aes(color = Cluster, shape = type), alpha = 0.7) +
  geom_encircle(aes(fill = Cluster, color = Cluster), alpha = 0.2, s_shape = 1, expand = 0)
```

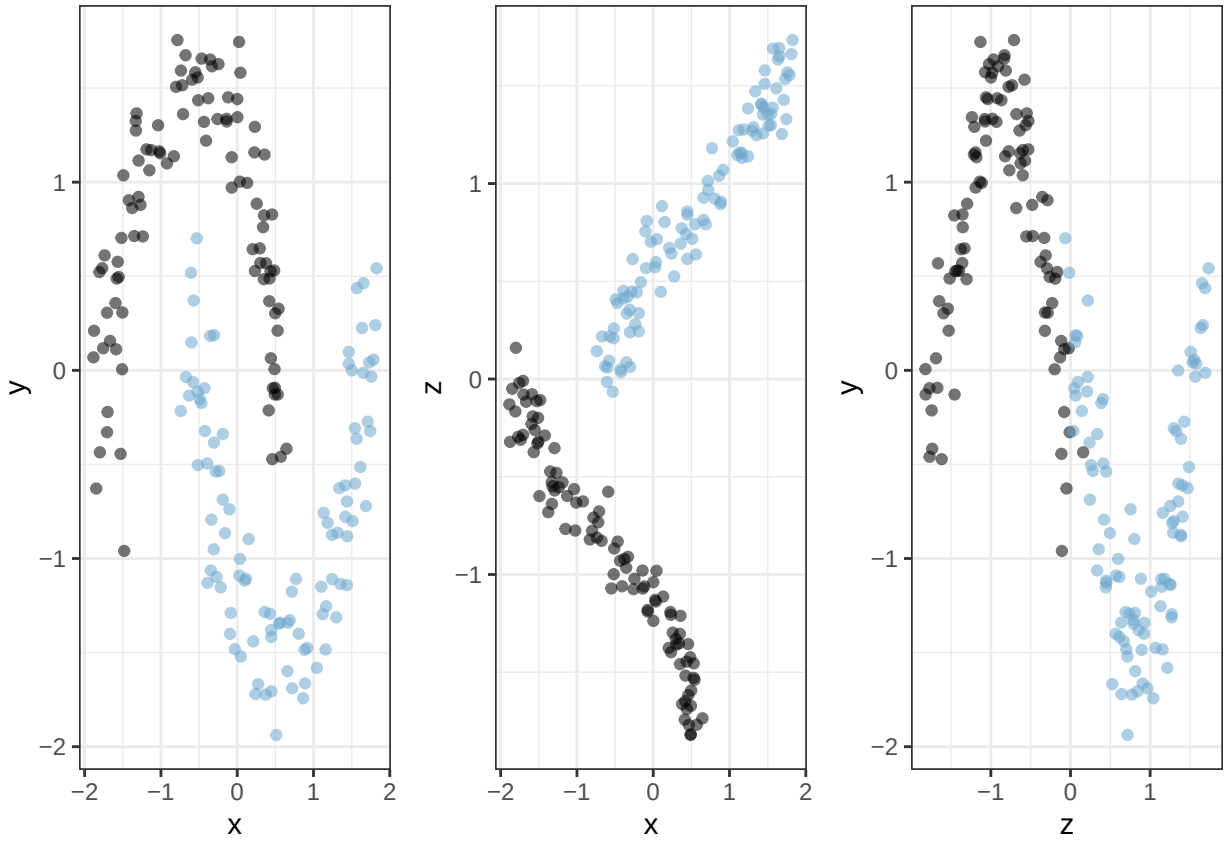


This performs a similar seperation to the K-means clustering on the original data and still obtains a separation of the clusters. # Task 2

For this task we use data simulated using the `make_moons()` function:

```
library(dplyr)
library(clusteringdatasets)
library(gridExtra)

moons <- make_moons(n_samples = 200, shuffle = FALSE, noise = 0.1)
z <- as.vector((scale(1:200) + rnorm(200, sd = 0.1)))

data <- cbind(scale((moons$samples)), moons$labels)

data <- data %>%
  as_tibble() %>%
  dplyr::rename("x" = "V1",
                "y" = "V2",
                "label" = "V3") %>%
  mutate(z = z,
         label = as.factor(label))

p1 <- ggplot(data = data, aes(x =x, y=y, color = label, alpha = 0.7)) +
   geom_point() +
   theme(legend.position = "none")

 p2 <- ggplot(data = data, aes(x = x, y=z, color = label, alpha = 0.7))+
   geom_point() +
```

```
  theme(legend.position = "none")

p3 <- ggplot(data = data, aes(x = z, y = y, color = label, alpha = 0.7)) +
  geom_point() +
  theme(legend.position = "none")

grid.arrange(p1, p2, p3 , ncol = 3)
```

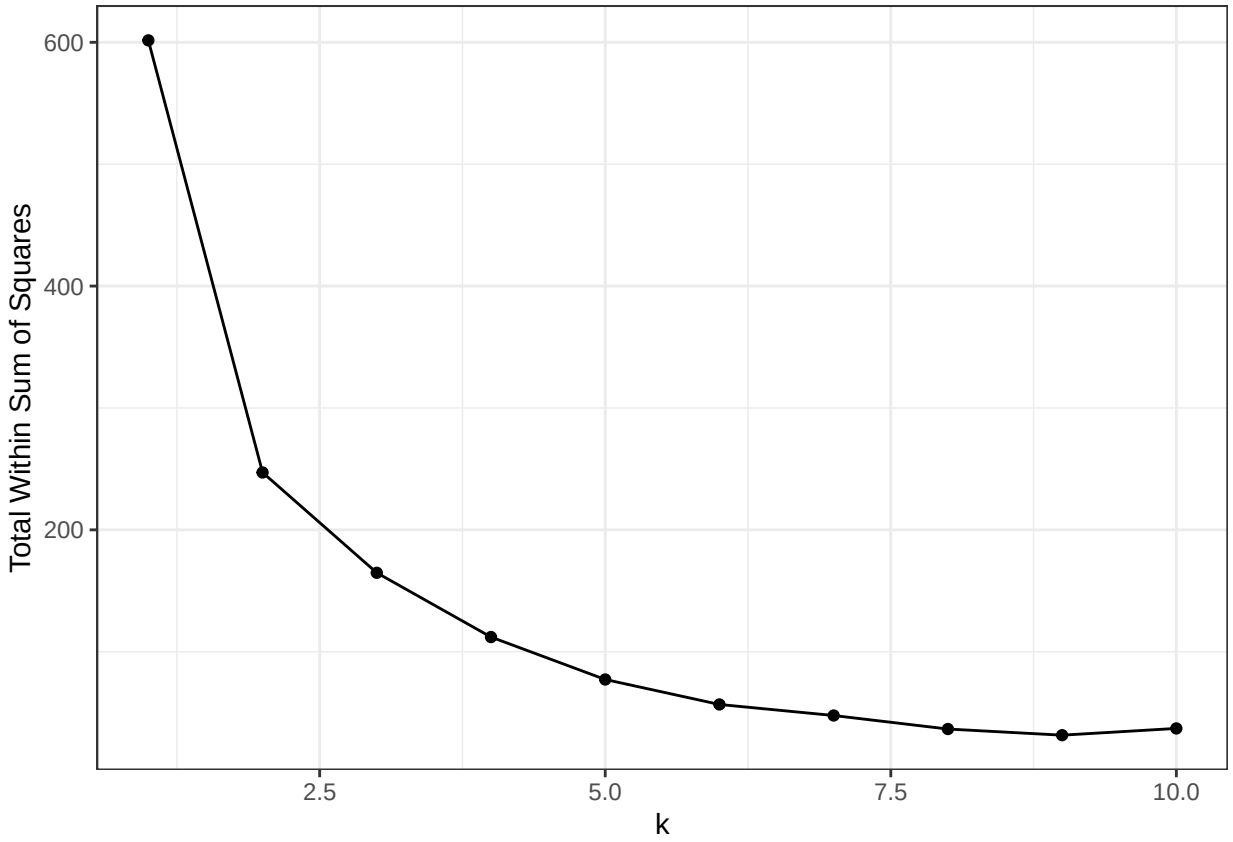


K-means Clustering

We first k-means clustering to see if they are appropriate for our data. We use `kmeans()` and plot the total within sum of squares to choose k:

```
sum_squares <- numeric(K)
fits <- vector("list",K)

for (k in 1:K){
  fit <- kmeans(data[,-3],
                centers = k,
                iter.max = 20,
                algorithm = "Lloyd")
  sum_squares[k] <- fit$tot.withinss

  fits[[k]] <- fit
}
```

```
sum_squares <- tibble(k = 1:K, totwss = sum_squares)

ggplot(data = sum_squares, aes(x = k, y = totwss)) +
  geom_point() +
  geom_line() +
  labs(y = "Total Within Sum of Squares")
```

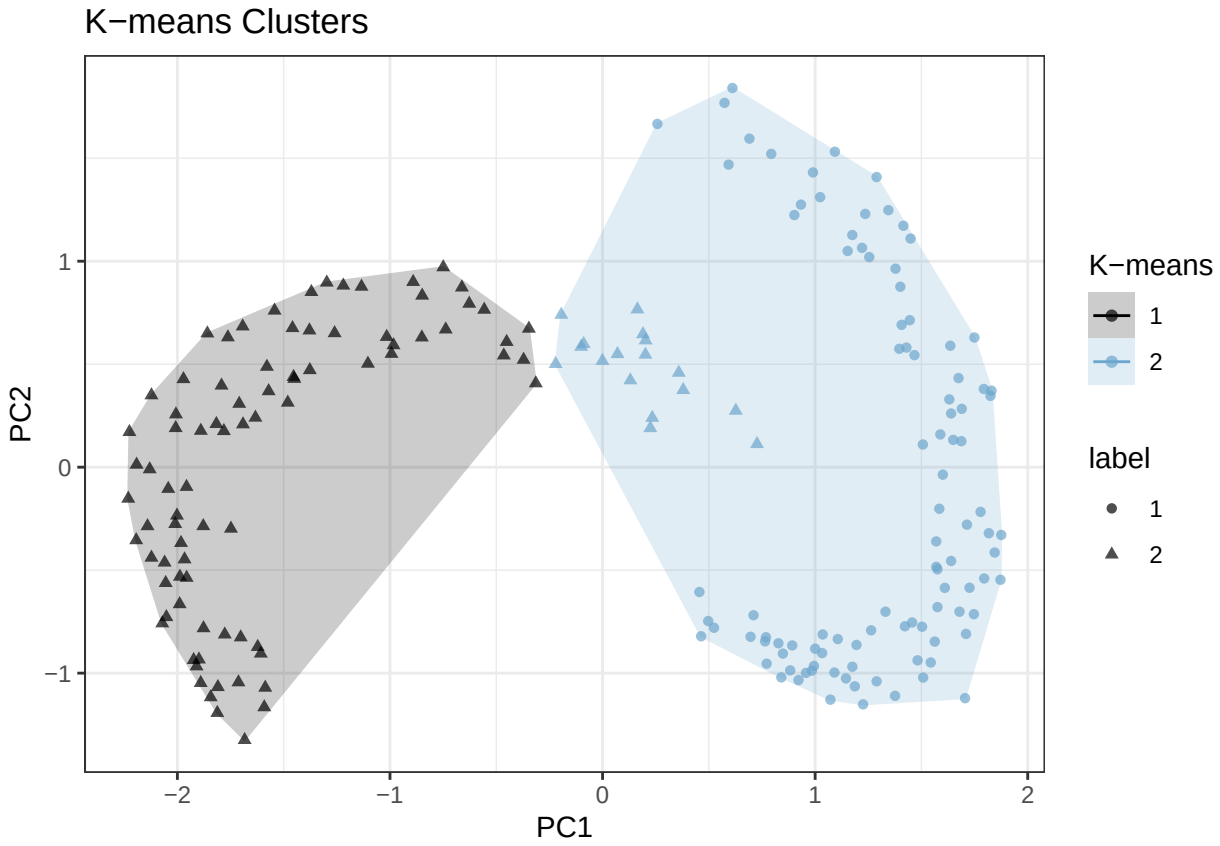


As with the agglomerative clustering a clear choice is $k = 2$, we can see there is a change of slope here.

```
moons_clusters_k <- fits[[2]]$cluster
```

We now plot the two-dimensional representations of these clusterings:

```
moons_pc <- prcomp(~ x + y + z, data = data, scale. = TRUE)

plot_data <- data %>%
  mutate("PC1" = moons_pc$x[,1],
         "PC2" = moons_pc$x[,2],
         "K-means" = as.factor(moons_clusters_k))

ggplot(plot_data, aes(x = PC1, y= PC2)) +
  geom_point(aes(color = `K-means`, shape = label),
             alpha = 0.7) +
  geom_encircle(aes(fill = `K-means`, color = `K-means`),
                alpha = 0.2, s_shape = 1, expand = 0) +
  labs(title = "K-means Clusters")
```



We can see the data is not separated into our two samples here due to the fact that K-means clustering does not allow non-convex clusters.

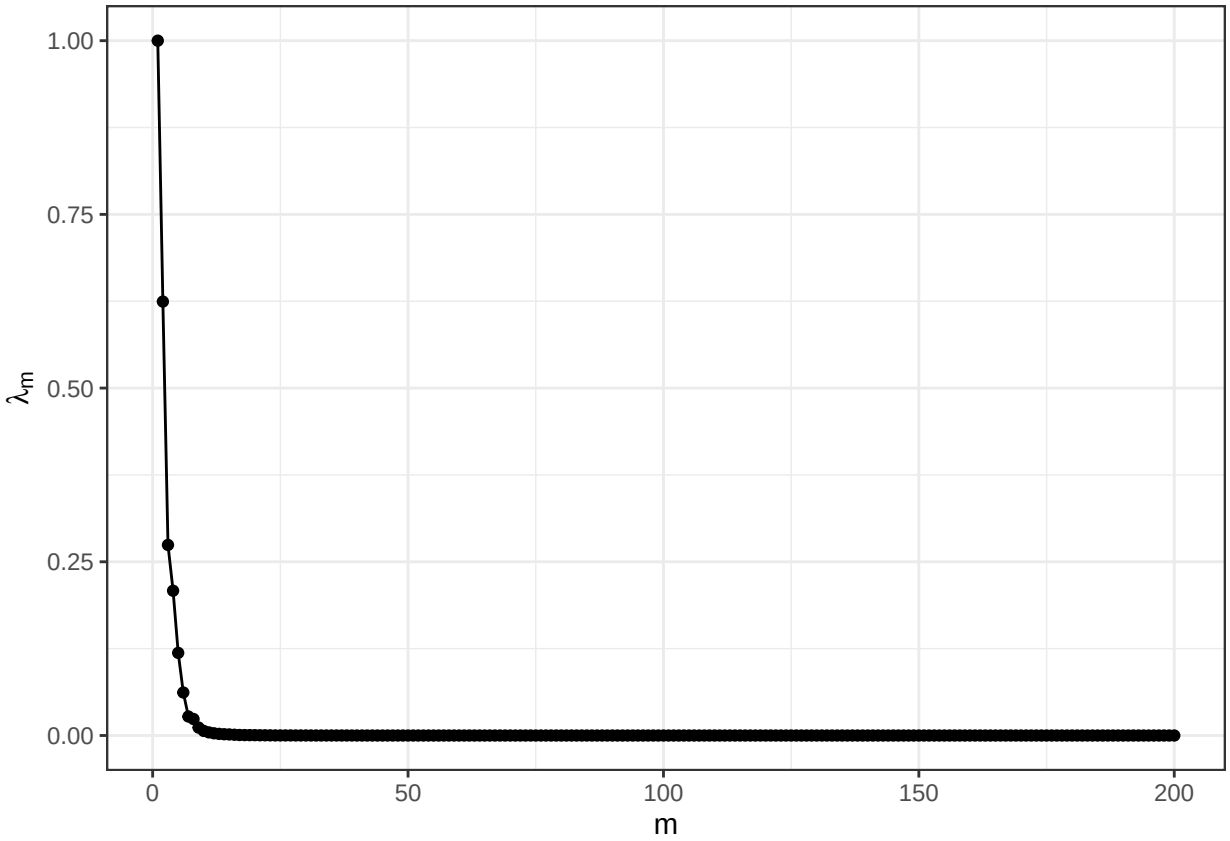
Spectral Clustering

We will perform spectral clustering with different similarity measures, which we implement by using different kernels in the `specc()` function from the `kernlab` package.

To determine the number of clusters to use, we use the complete graph and obtain the graph Laplacian and it's eigen-values, using the median heuristic as a starting value of `c`:

```
library(rdist)
library(latex2exp)

c <- median(as.vector(pdist(data[,-3])))

X_dist <- as.matrix(dist(data[,-3], upper = TRUE))

W <- exp(-X_dist^2/c^2)
G <- diag(colSums(W)^{-1/2})
L_tilde <- G %*% W %*% G

eigen_l <- eigen(L_tilde)
eigen_l <- tibble(m = 1:length(eigen_l$values), lambda = eigen_l$values)

ggplot(eigen_l, aes(x = m, y = lambda)) +
  geom_point() +
```

```
  geom_line() +
  labs(y = TeX("$\\lambda_m$"), x = "m")
```



We can see that the largest gaps occur for $m = 1$ or $m = 2$. 1 cluster would not provide any additional insight, so we choose $m = 2$.

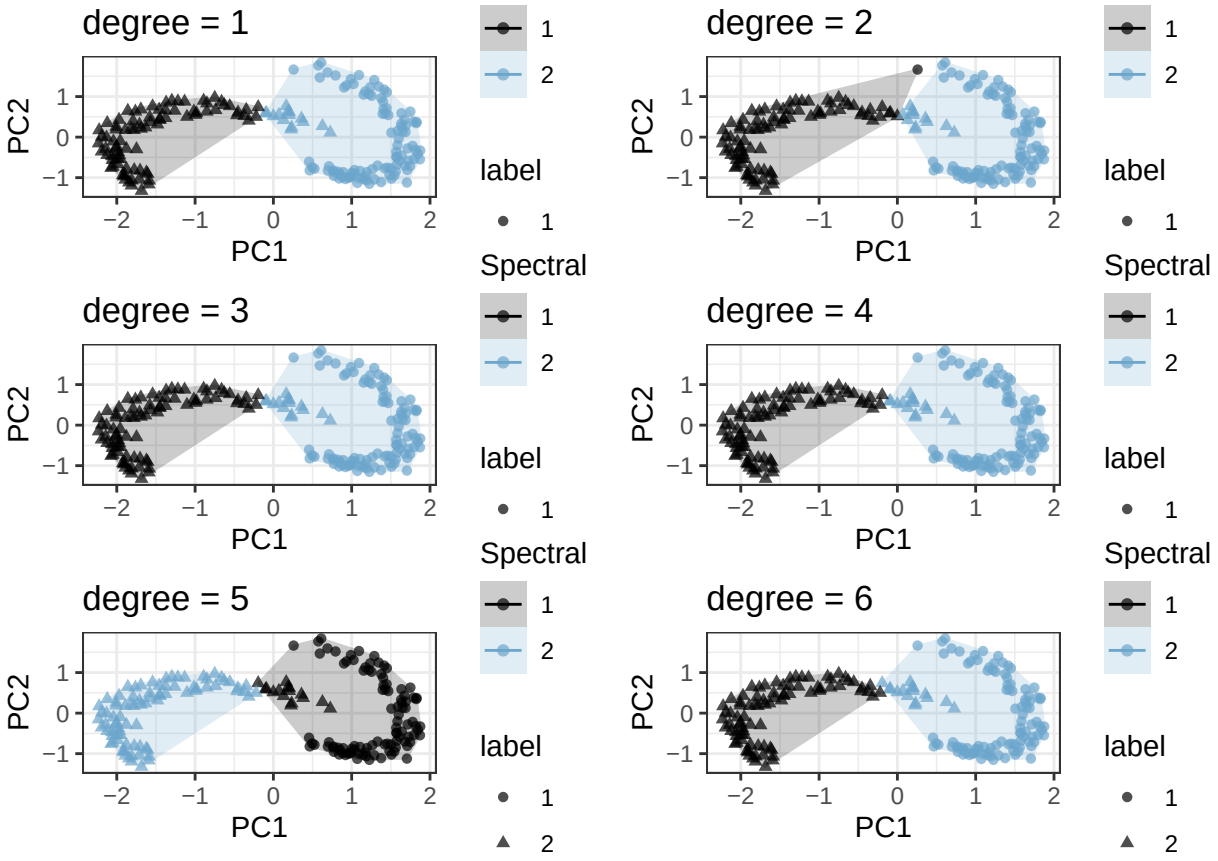
Polynomial Kernel

For this, we again assume 2 clusters and plot our clusters for different values of the degree.

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:scales':
##
##     alpha

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
degree <- as.vector(1:6)

fit_list <- vector("list", length(degree))
plot_list <- vector("list", length(degree))

for (i in 1:length(degree)){
  spectral_fit <- specc(~x +y + z,
                        data = data,
                        centers = 2,
                        kernel = "polydot",
                        kpar = list(degree = degree[i]))
  fit_list[[i]] <- spectral_fit
  spec_rbf_clusters <- as.vector(spectral_fit)

  plot_data <- plot_data %>%
  mutate("Spectral" = as.factor(spec_rbf_clusters))

  p <- ggplot(plot_data, aes(x = PC1, y= PC2)) +
    geom_point(aes(color = Spectral, shape = label),
               alpha = 0.7) +
    geom_encircle(aes(fill = Spectral, color = Spectral),
                  alpha = 0.2, s_shape = 1, expand = 0) +
    labs(title = paste("degree =", degree[i]))

  plot_list[[i]] <- p
}


do.call("grid.arrange", c(plot_list, ncol = 2))
```



We can see this performs similarly to the K-means clustering and so we try a Gaussian kernel/

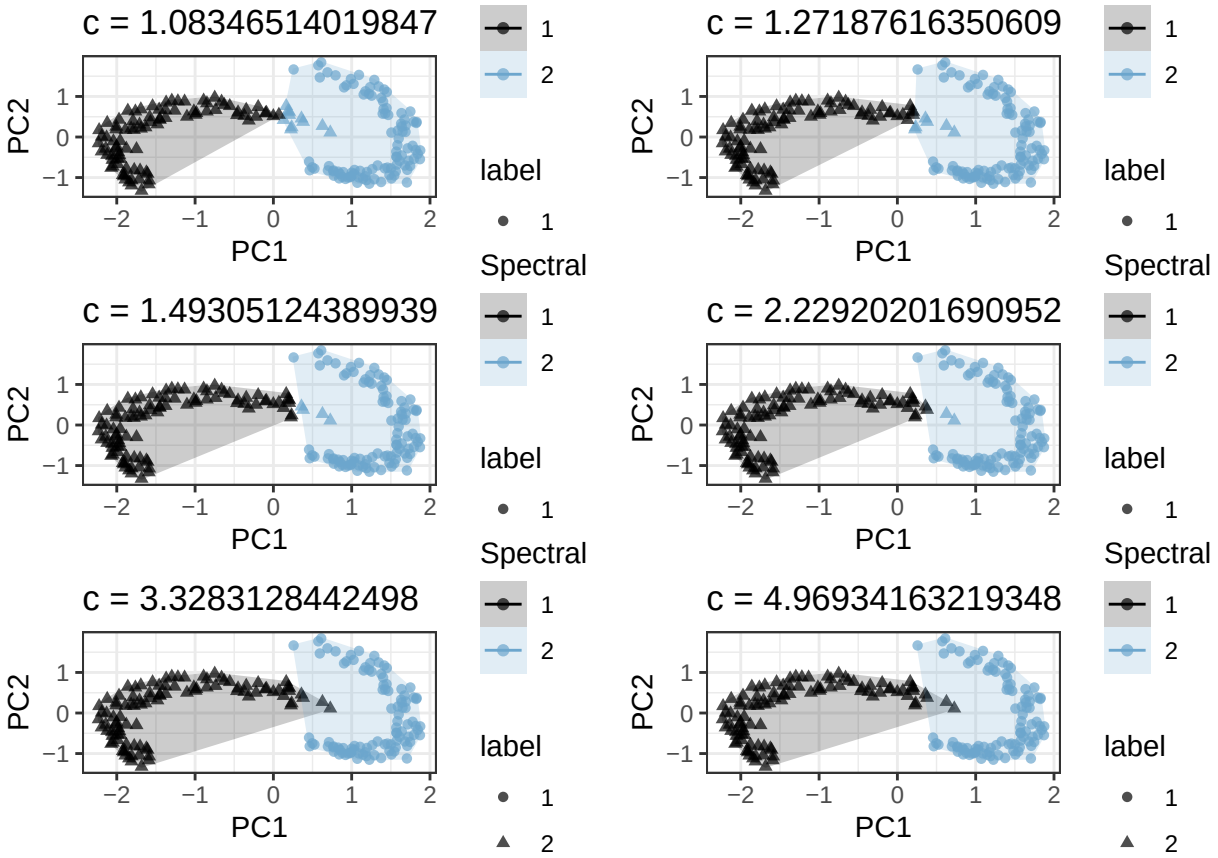
Gaussian Kernel

The Gaussian kernel is equivalent to taking $s_{il} = \exp\left(-\frac{d_{il}^2}{c^2}\right)$, where `c>0` corresponds to the bandwidth parameter.

We now implement a spectral clustering for different values of `c`:

```
c <- median(as.vector(pdist(data[c(1:2,4)])))

c <- c^c(0.1,0.3,0.5,1,1.5,2)

fit_list <- vector("list", length(c))
plot_list <- vector("list", length(c))

for (i in 1:length(c)){
  spectral_fit <- specc(~x +y + z,
                        data = data,
                        centers = 2,
                        kernel = "rbfdot",
                        kpar = list(sigma = c[i]))
  fit_list[[i]] <- spectral_fit
  spec_rbf_clusters <- as.vector(spectral_fit)

  plot_data <- plot_data %>%
  mutate("Spectral" = as.factor(spec_rbf_clusters))
```

```
  p <- ggplot(plot_data, aes(x = PC1, y= PC2)) +
    geom_point(aes(color = Spectral, shape = label),
               alpha = 0.7) +
    geom_encircle(aes(fill = Spectral, color = Spectral),
                  alpha = 0.2, s_shape = 1, expand = 0) +
    labs(title = paste("c =", c[i]))

  plot_list[[i]] <- p
}


do.call("grid.arrange", c(plot_list, ncol = 2))
```



We see that a value of $c = 3.328$ provides a complete separation of our data and thus an improvement on k-means clustering.

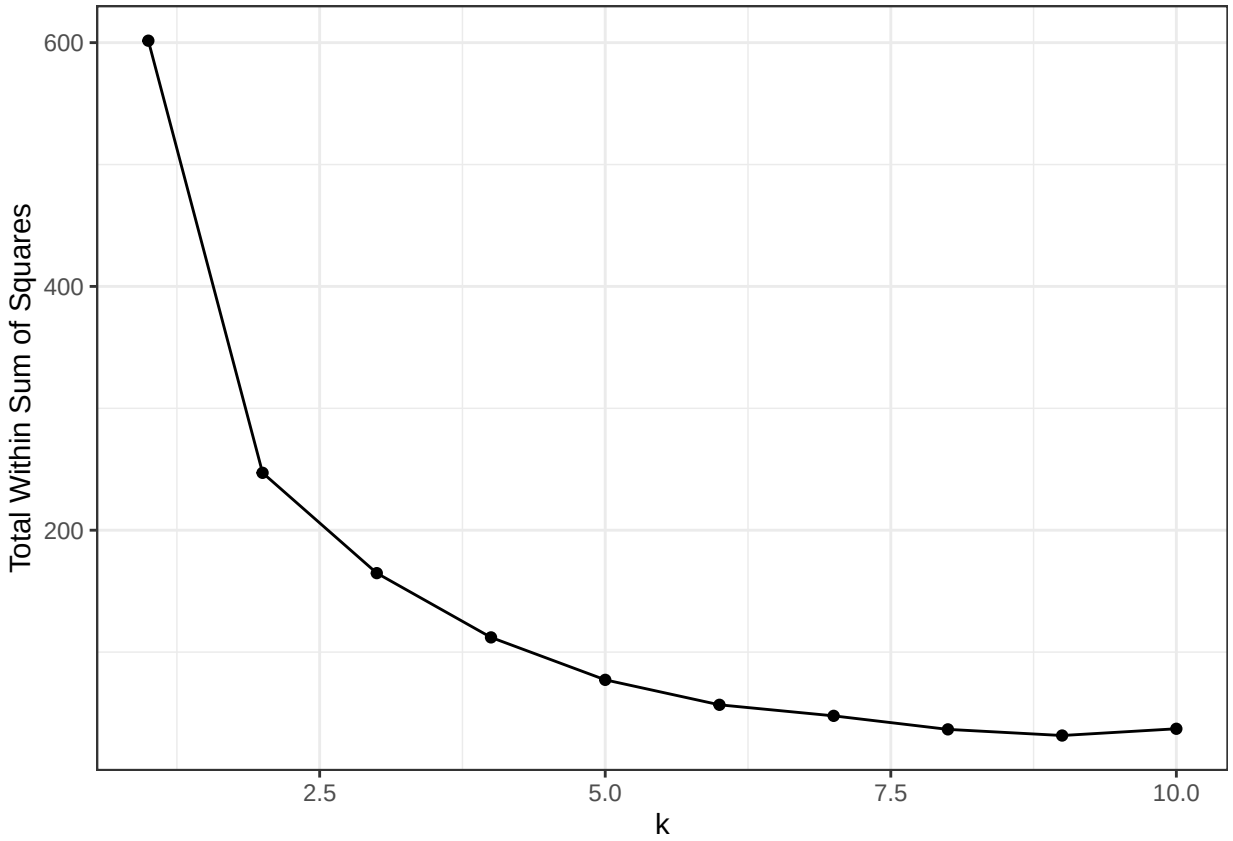
Task 3

If we consider Lloyd's K-means clustering algorithm, the norm is only computed for pairs of observations from the data set, and thus if we replace the norm with our kernel function, it only needs to be defined pointwise on our dataset.

We use the `kkmeans()` function, selecting a Gaussian kernel and choosing the bandwidth according to the median heuristic. We choose k by plotting the total within sum of squares from different values of k.

```
K <- 10

withinss <- numeric(K)
fit_list <- vector("list", K)

for (k in 1:K){
  kkmeans_moons <- kkmeans(~ x + y + z, data = data, kernel = "rbfdot", centers = k)

  withinss[k] <- sum(withinss(kkmeans_moons))
  fit_list[[k]] <- kkmeans_moons
}

wihtinss <- tibble(k = 1:K, totwss = withinss)
```

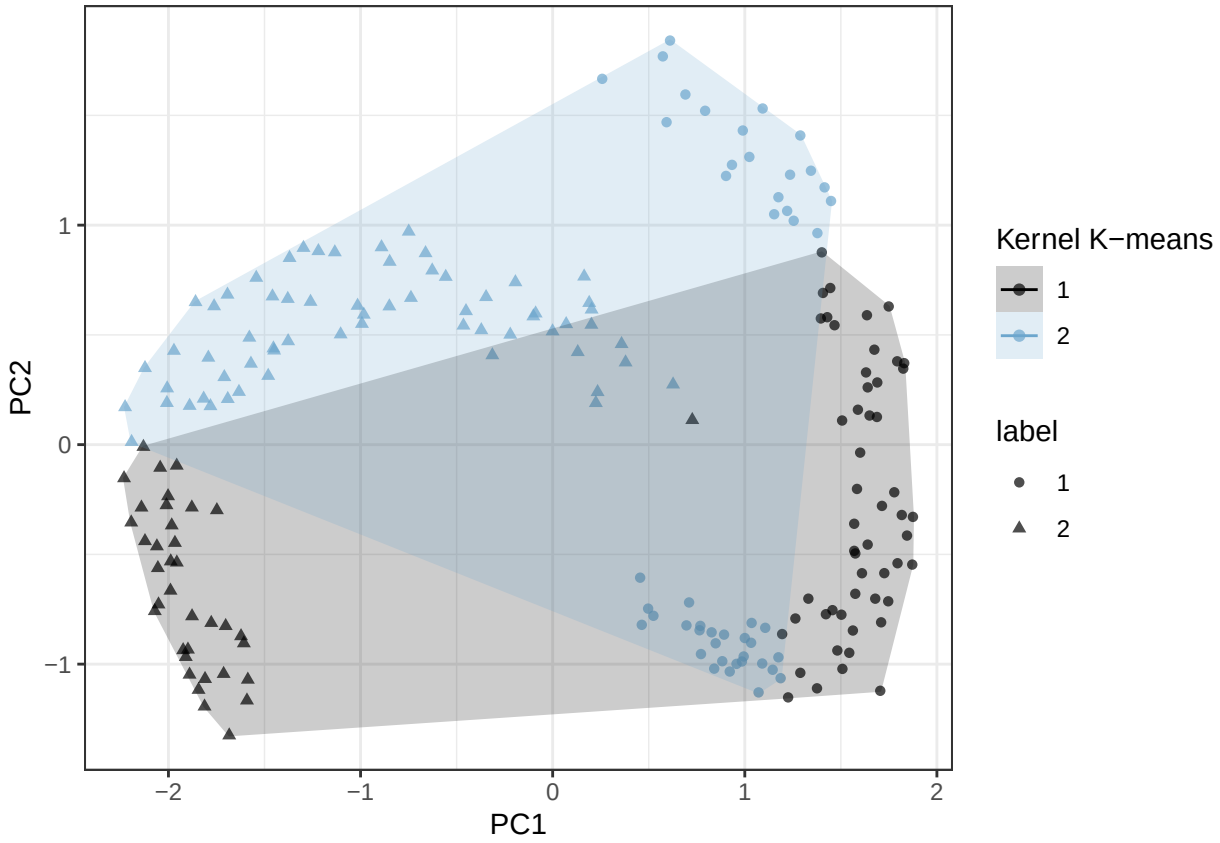
```
ggplot(data = sum_squares, aes(x = k, y = totwss)) +
  geom_point() +
  geom_line() +
  labs(y = "Total Within Sum of Squares")
```



We can see the clear choice of k is 2 as this is where the slope changes the most. We visualise the 2 dimensional reduction of

```
plot_data <- plot_data %>%
  mutate("Kernel K-means" = as.factor(as.vector(fit_list[[2]])))

ggplot(plot_data, aes(x = PC1, y= PC2)) +
  geom_point(aes(color = `Kernel K-means`, shape = label),
```

```
            alpha = 0.7) +
  geom_encircle(aes(fill = `Kernel K-means`, color = `Kernel K-means`),
                alpha = 0.2, s_shape = 1, expand = 0)
```



We can see this doesn't provide a good clustering, the spectral clustering with a Gaussian kernel is better by far.