Portfolio 4: Spectral Cluster Analysis

Kieran Morris

Task 1: Basic Cluster Analysis

For task 1 we will apply agglomarative clustering and K-means clustering to the `diamonds` dataset from `ggplot2`. Unfortunately this dataset contains multiple ordered catagorical variables, which may help to cluster, however since we will lose the ordering by converting to binary we remove them for now. As a reminder lets look at a summary of the dataset.

```r
# Load libraries and data
library(cluster)
library(ggplot2)
library(skimr)
library(ggpubr)
data(diamonds)

# Remove ordered catagorical variables
Diam <- subset(diamonds, select = -c(color, cut, clarity) )
# Summary of the dataset
diamonds_summary <- skim(Diam)
```

As we can see, this dataset is massive, ~50,000 datapoints, and (now) 7 continuous variables.

Agglomerative Clustering

Forming tha dendrogram

Before we run over the entire datset we will take a sample of 1000, as it will be very hard to make sense of a dendrogram so large.

Below we use the standard euclidean distance - since we are only working over continuous variables - by using the `dist()` and scale it appropriately, so our maximum distance is 10.

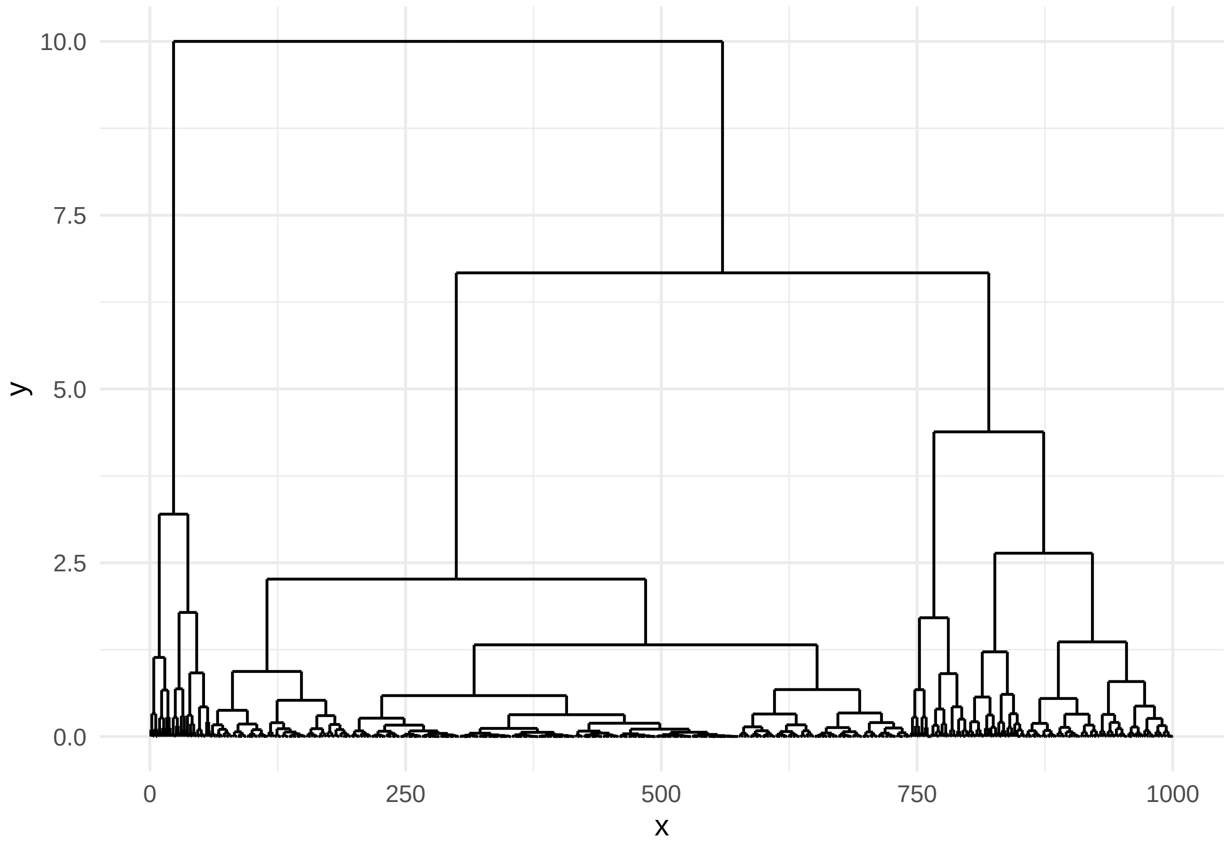
```r
# Set seed for reproducibility
set.seed(420)
# Load ggdendro for dendrogram plotting
library(ggdendro)
# Take a sample of 1000
diam_sample <- Diam[sample(1:nrow(Diam), 1000),]
# Compute and scale the distance
diam_dist <- dist(diam_sample)/(0.1*max(dist(diam_sample)))
```

Below we make use of the `hclust` function from `R` and specify `method = "complete"`, which employs the maximum distance between clusters, or more formally:

$$d_{t,(r,s)} = \max(d_{rt}, d_{st})$$

where d_{rt} and d_{st} are the distances between the rth and sth and the tth cluster, respectively.

```r
#Apply hclust with complete linkage
clusters <- hclust(diam_dist,method = "complete")
#Plot dendrogram of sample
ggdendrogram(clusters,labels = FALSE)+ theme_minimal()
```



Our dendrogram has a few large jumps, if we slice at $d = 5$ then we could have 3 clusters of data, however a higher precision cut at $d = 2.5$ would give us 6 clusters and a more conservative cut at $d = 7$ would give us 2 clusters. We attempt both of these.

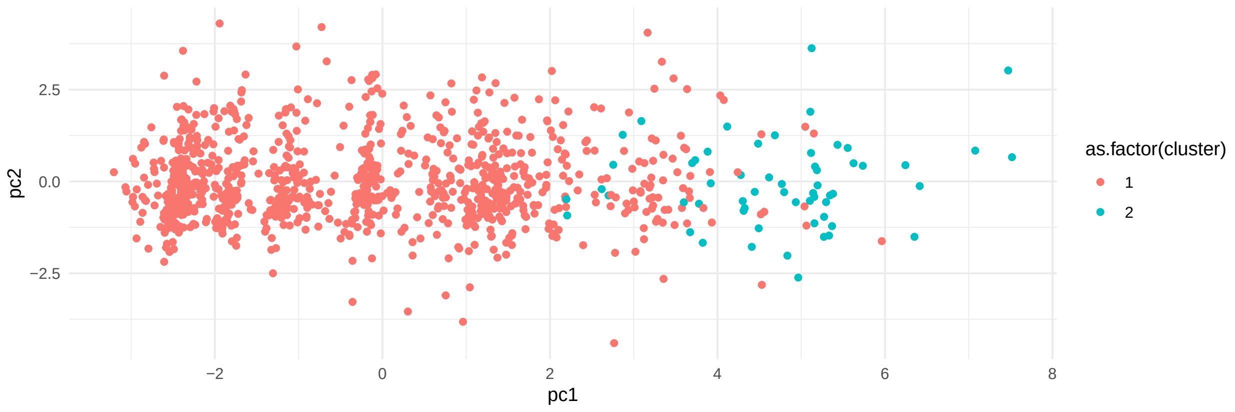
Choice of clusters

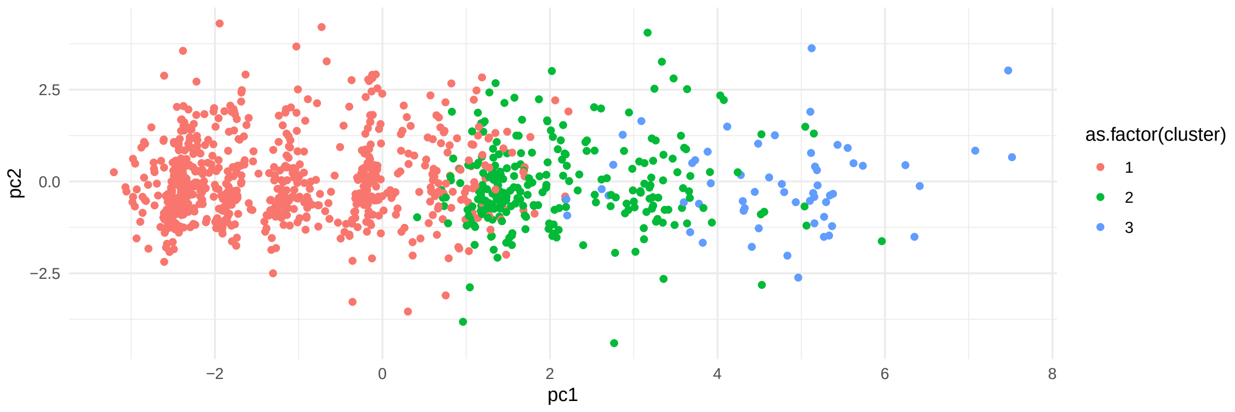
We will now apply the `cutree` function to the dendrogram to cut it into 3 clusters. Then we will use principal component analysis to get a 2 dimensional reduction.

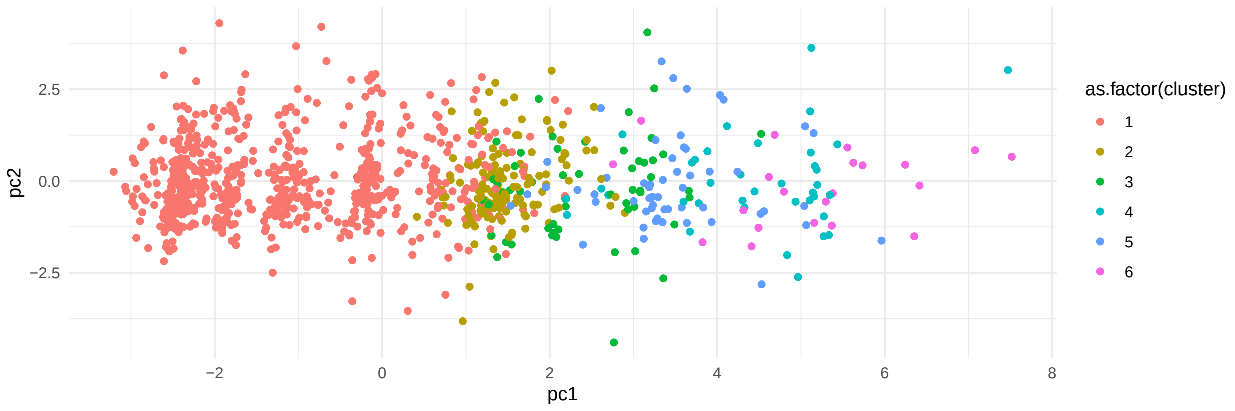
```r
plot_k_clusters <- function(k){
  # Cut the dendrogram into k clusters
  diam_clusters <- cutree(clusters, k = k)
  # Add the cluster to the sample
  diam_sample$cluster <- diam_clusters
  # Apply PCA
  diam_pca <- prcomp(diam_sample[,1:7], scale = TRUE)
  # Add the PCA to the sample
  diam_sample$pc1 <- diam_pca$x[,1]
  diam_sample$pc2 <- diam_pca$x[,2]
  # Plot the clusters
  return(ggplot(diam_sample, aes(x = pc1, y = pc2, color = as.factor(cluster))) + geom_point()+ theme_m

}

plt2 <- plot_k_clusters(2)
plt3 <- plot_k_clusters(3)
plt6 <- plot_k_clusters(6)
ggarrange(plt2, plt3, plt6, nrow = 3)
```







Overall agglormerative clustering seems to be doing a good job, in all cases of clustering we can see that the clusters are linearly seperable (when reduced to 2 principal components).

K-means Clustering

Test Case

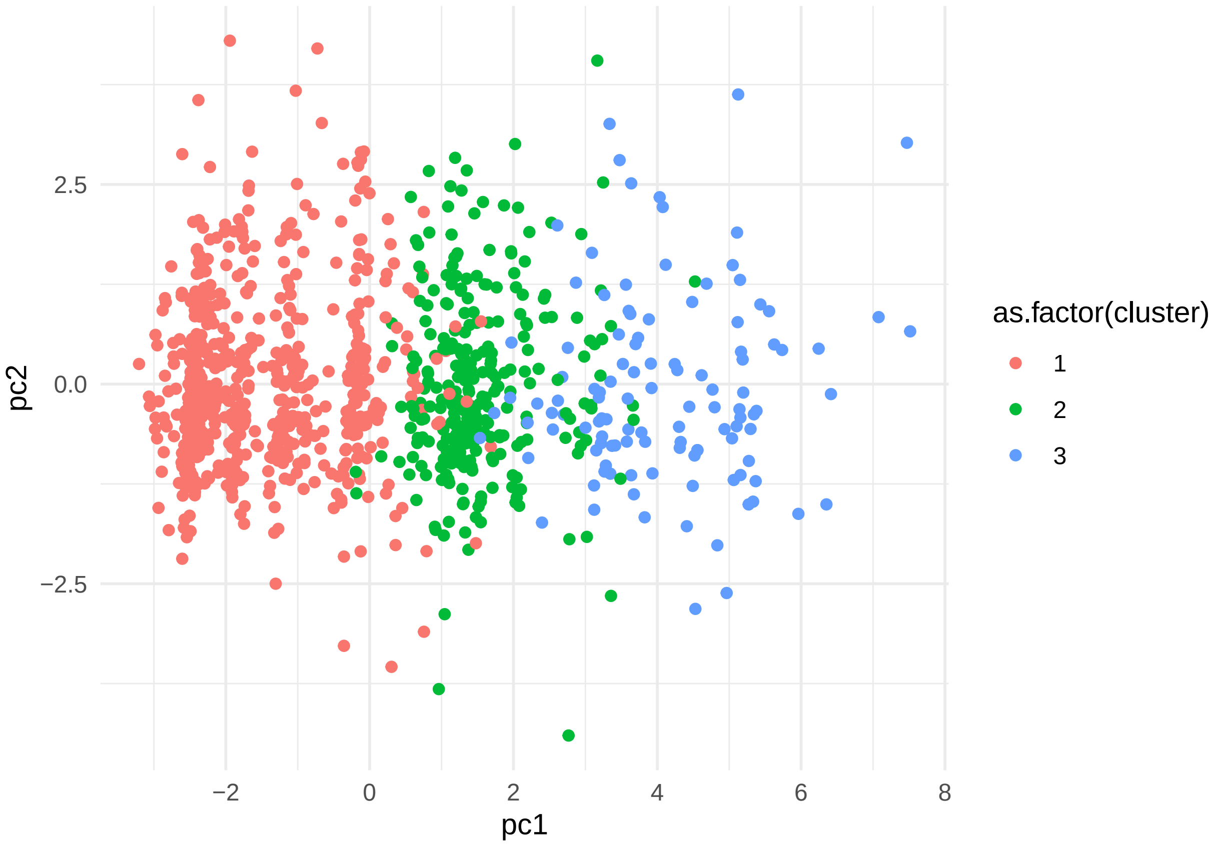
Again for K-means clustering we need to restrict to continuos variables as K-means clustering doesn't allow for categorical variables - as we are constantly computing the mean of data clusters. We simplicity and the ability to compare against agglomarative clustering we stick to euclidean distance.

Below we use the `kmeans()` function from `R` to perform our clustering, and we specify Lloyd's algorithm must be used via `algorithm = "Lloyd"` in the `kmeans()` function. We usually would immediately go for an elbow plot to find the optimal k, so the below is purely to verify we're on the right track.

```
# Set seed for reproducibility
set.seed(420)
# Get sample
diam_sample <- Diam[sample(1:nrow(Diam), 1000),]

plot_kmeans_clusters <- function(k){
  # Perform K-means clustering
  kmeans_clusters <- kmeans(diam_sample, centers = k, algorithm = "Lloyd")
  # Add the cluster to the sample
  diam_sample$cluster <- kmeans_clusters$cluster
  # Apply PCA
  diam_pca <- prcomp(diam_sample[,1:7], scale = TRUE)
  # Add the PCA to the sample
  diam_sample$pc1 <- diam_pca$x[,1]
  diam_sample$pc2 <- diam_pca$x[,2]
  # Plot the clusters
  return(ggplot(diam_sample, aes(x = pc1, y = pc2, color = as.factor(cluster))) + geom_point()+ theme_mi
}

plt2 <- plot_kmeans_clusters(3)
plt2
```

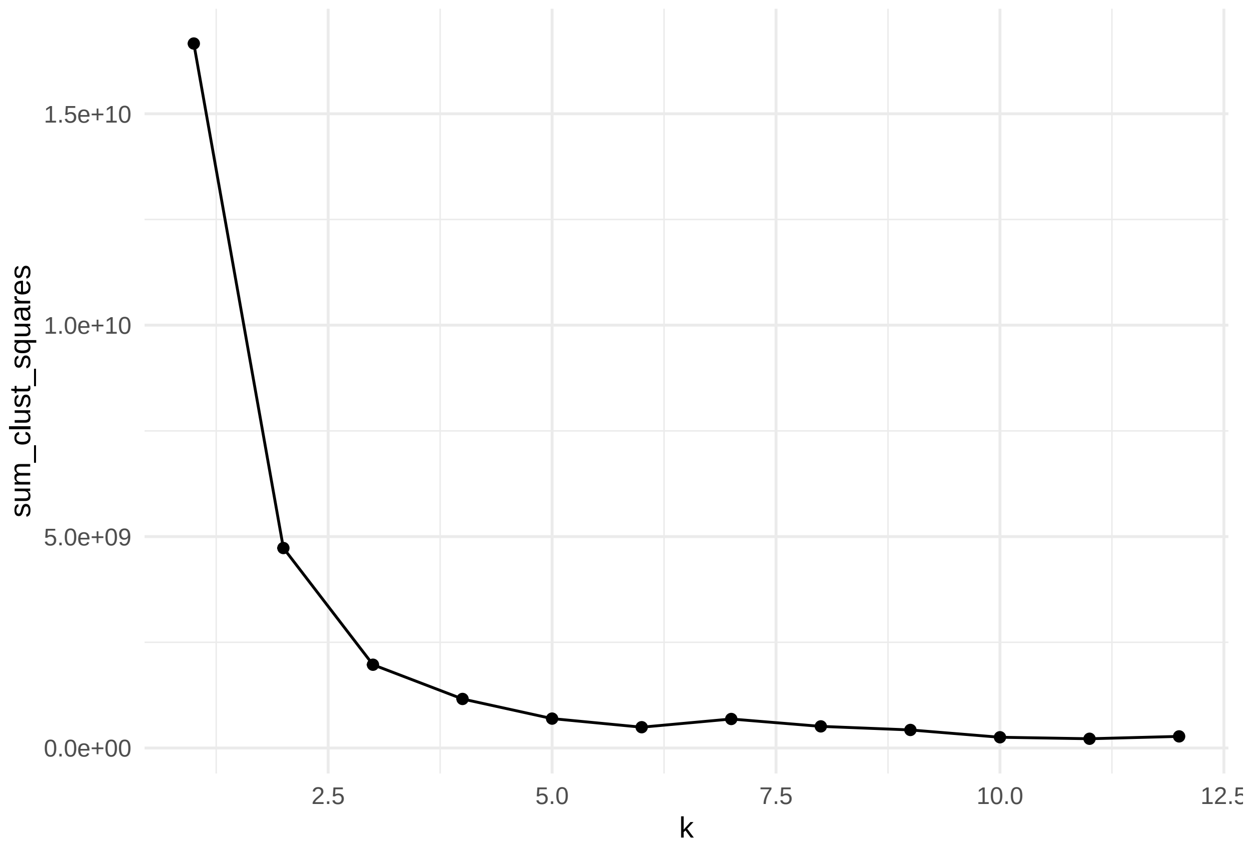


Unsurprisingly, K-means clustering is very similar to agglomarative clustering in the $k = 3$ case.

Elbow Plot

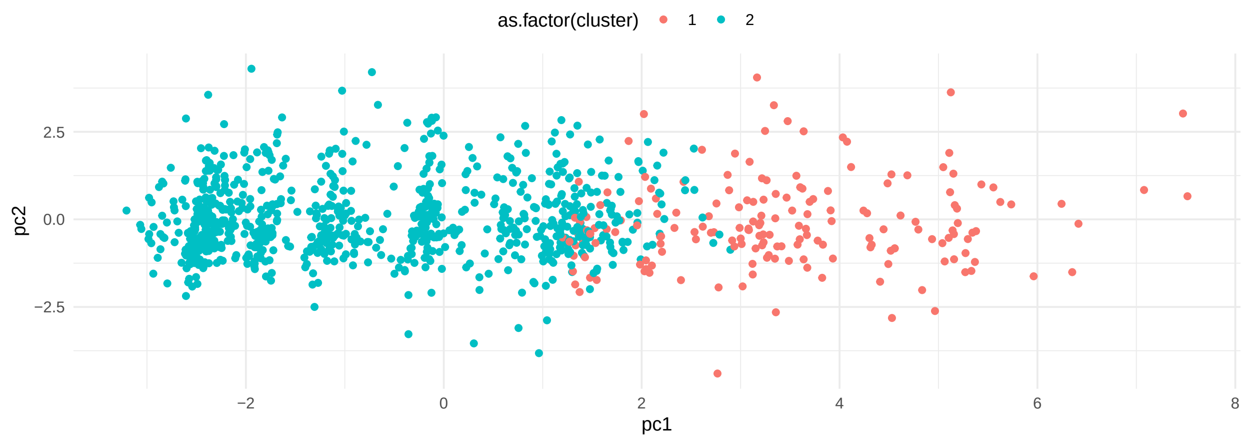
To be formal let's run this over multiple k values and plot the sum of squares difference.

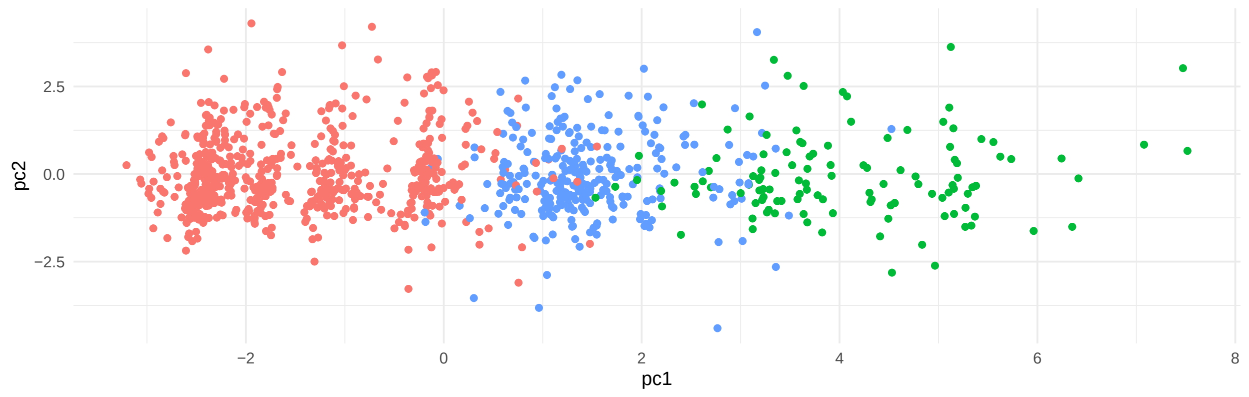
```r
Upper_K <- 12
sum_clust_squares <- rep(0,Upper_K)
for (k in 1:Upper_K){
  kmeans_clusters <- kmeans(diam_sample, centers = k, algorithm = "Lloyd")
  sum_clust_squares[k] <- kmeans_clusters$tot.withinss
}
elbow_plot <- ggplot(data.frame(k = 1:Upper_K, sum_clust_squares = sum_clust_squares),
aes(x = k, y = sum_clust_squares)) +
geom_point() + geom_line() +
theme_minimal()
elbow_plot
```

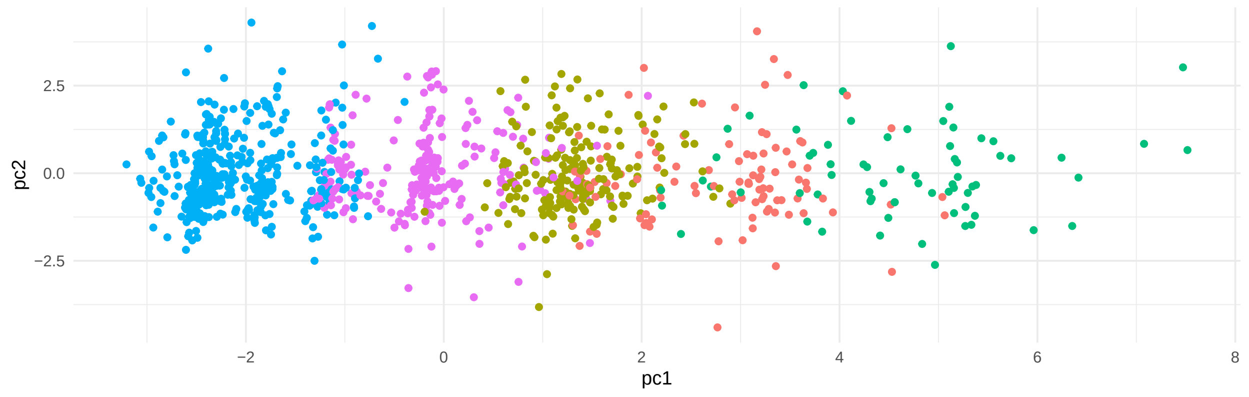


The largest changes in gradient occur at $k = 2, 3$ and 5, so we will plot all of these, our expectation is very similar to the agglomarative clustering.

```r
plt2 <- plot_kmeans_clusters(2)
plt3 <- plot_kmeans_clusters(3)
plt5 <- plot_kmeans_clusters(5)
ggarrange(plt2, plt3, plt5, nrow = 3,common.legend = TRUE)
```







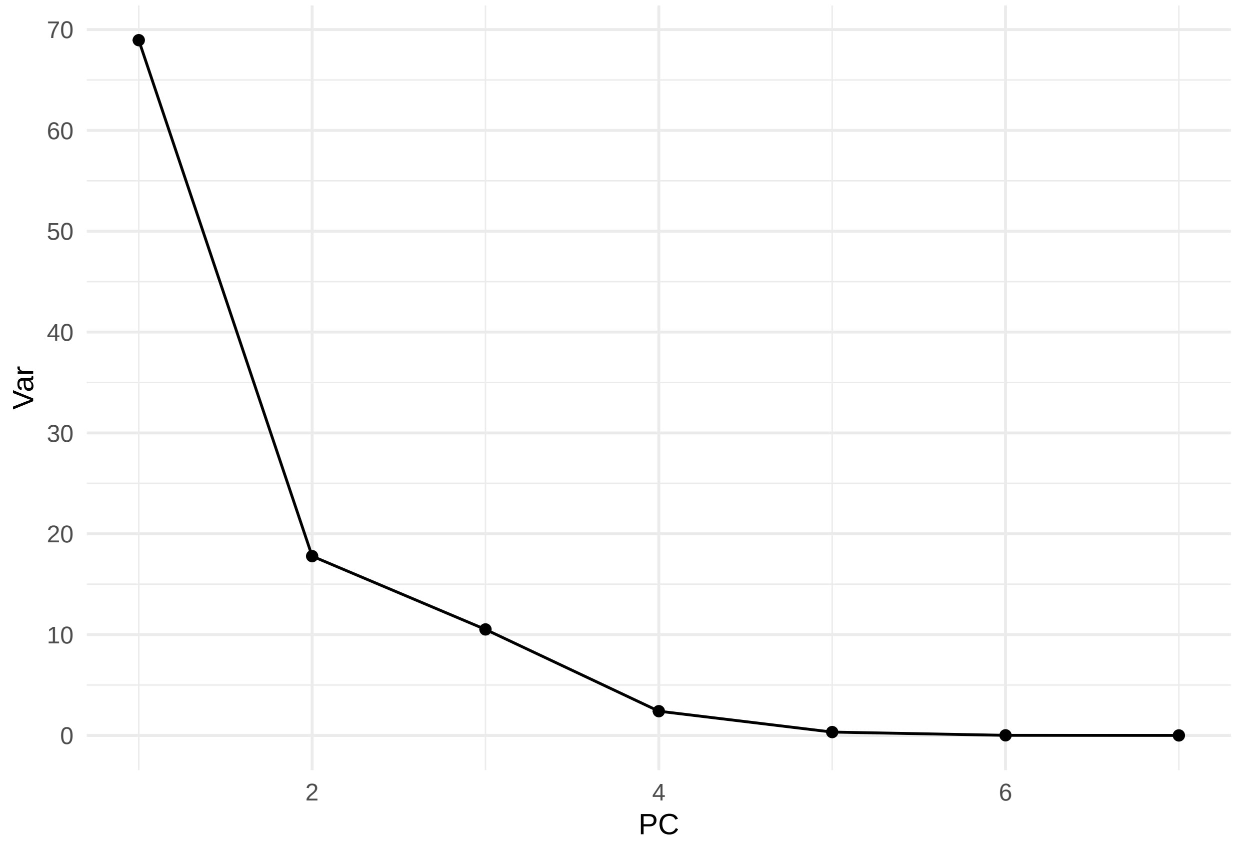
Again, we see that the clusters are linearly seperable, in particular they are almost indistinguishable from the agglomarative clustering.

PCA First

If we reverse the order in which we perform clustering and PCA, i.e instead of projecting the data onto PC1&2 after clustering we will use the data from some number of principal components as our input. The decision on what choice of PCs will be decided via a scree plot.

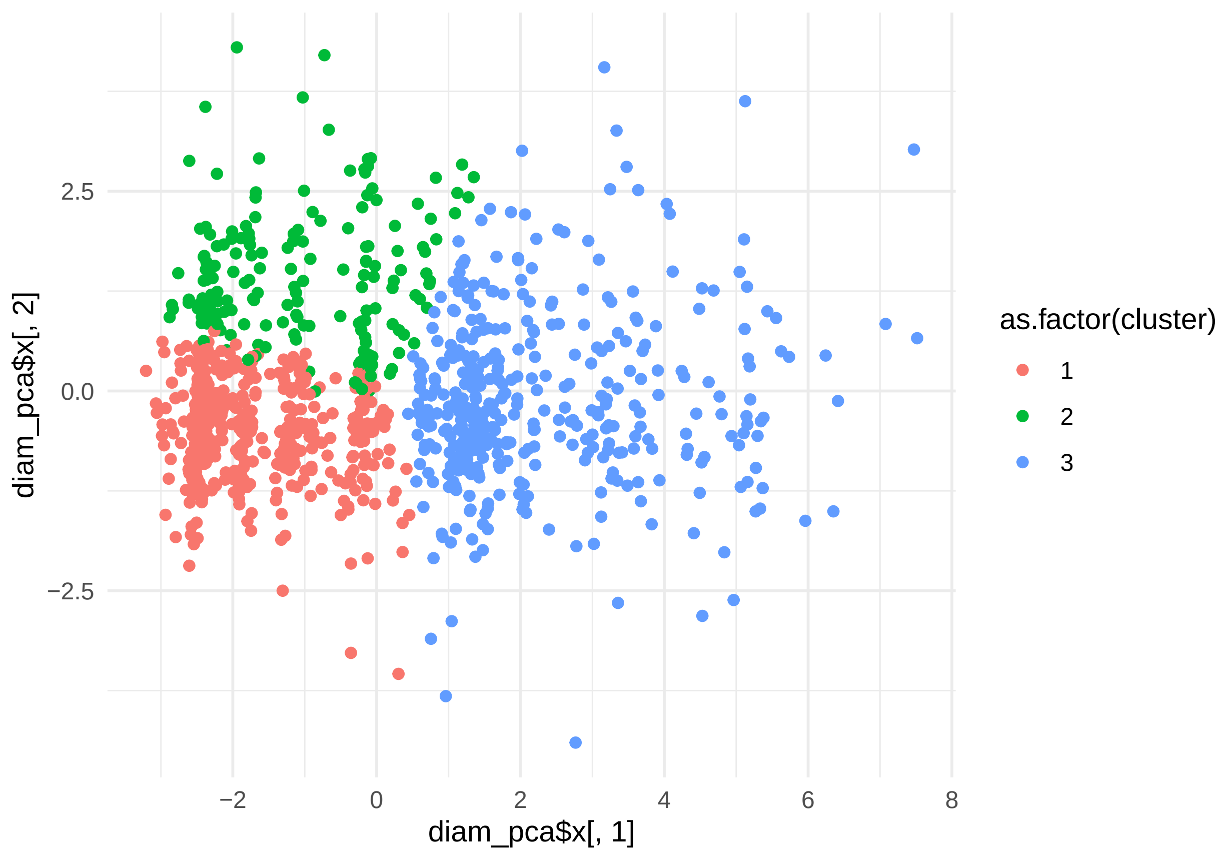
```r
set.seed(420)
# Get sample
diam_sample <- Diam[sample(1:nrow(Diam), 1000),]
# Perform PCA
```

```r
diam_pca <- prcomp(diam_sample[,1:7], scale = TRUE)
# Scree plot
explained_var <- diam_pca$sdev^2 / sum(diam_pca$sdev^2)
# Convert to percentage
explained_var_percent <- explained_var * 100
#Make scree plot
scree_plot <- ggplot(data.frame(PC = 1:7, Var = explained_var_percent), aes(x = PC, y = Var)) +
geom_point() + geom_line() + theme_minimal()+ scale_y_continuous(breaks = seq(0,100,10))
scree_plot
```



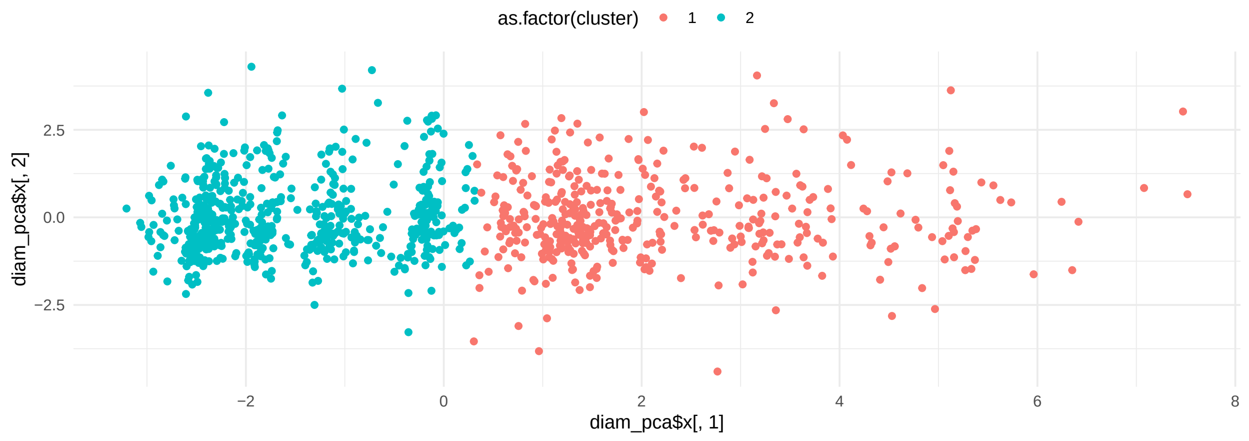
Here we see that the first 4 principal components comprise 95+% of the variance in the data, so we choose to restrict to PC1-4. We will now perform K-means clustering on this data. For now we arbitrarily choose the same $k = 2, 3, 5$ as before.

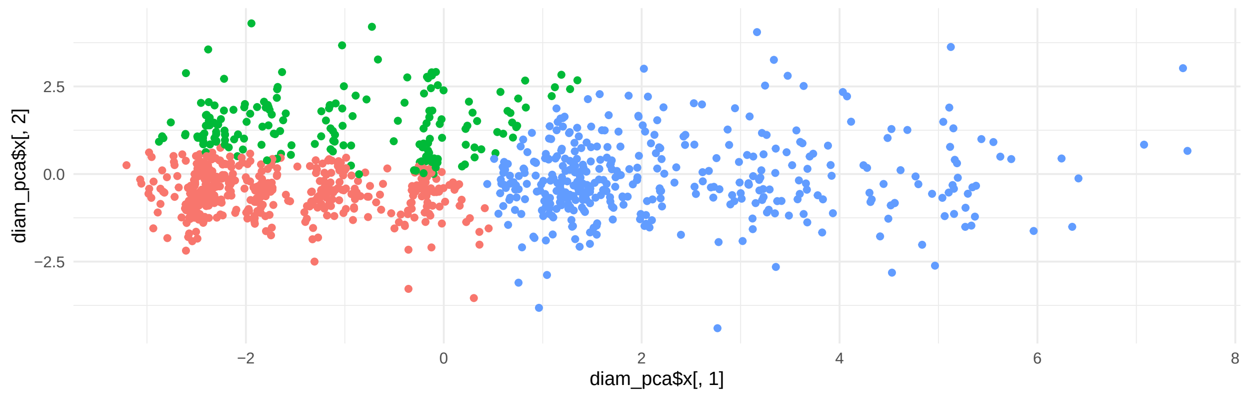
```r
set.seed(1)
# Perform K-means clustering
kmeans_clusters <- kmeans(diam_pca$x[,1:4], centers = 3, algorithm = "Lloyd")
# Add the cluster to the sample
diam_sample$cluster <- kmeans_clusters$cluster
# Plot the clusters
plt <- ggplot(diam_sample, aes(x = diam_pca$x[,1], y = diam_pca$x[,2], color = as.factor(cluster))) + g
plt
```

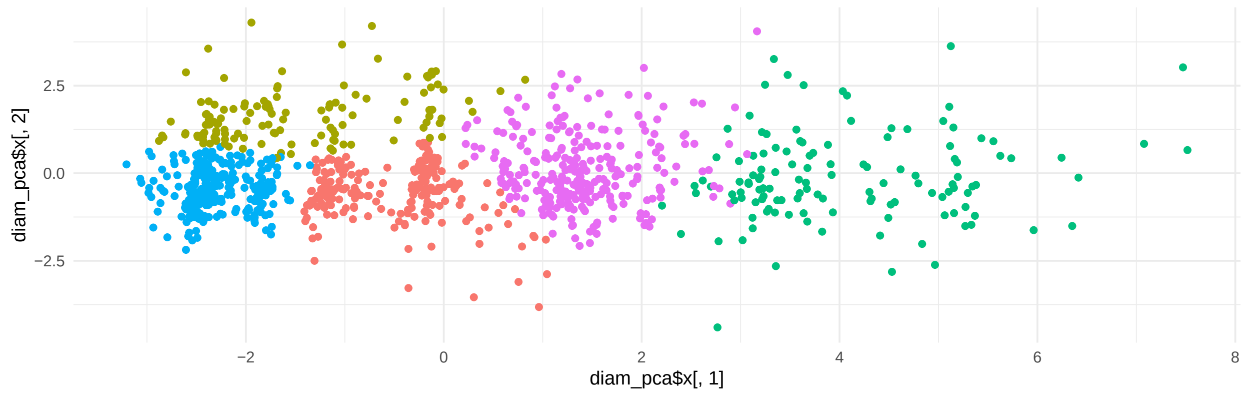


```r
plot_kmeans_clusters_pca <- function(k){
  set.seed(1)
  # Perform K-means clustering
  kmeans_clusters <- kmeans(diam_pca$x[,1:4], centers = k, algorithm = "Lloyd")
  # Add the cluster to the sample
  diam_sample$cluster <- kmeans_clusters$cluster
  # Plot the clusters
  return(ggplot(diam_sample, aes(x = diam_pca$x[,1], y = diam_pca$x[,2], color = as.factor(cluster))) +
}

plt_pca_clust2 <- plot_kmeans_clusters_pca(2)
plt_pca_clust3 <- plot_kmeans_clusters_pca(3)
plt_pca_clust5 <- plot_kmeans_clusters_pca(5)
ggarrange(plt_pca_clust2, plt_pca_clust3, plt_pca_clust5, nrow = 3,common.legend = TRUE)
```



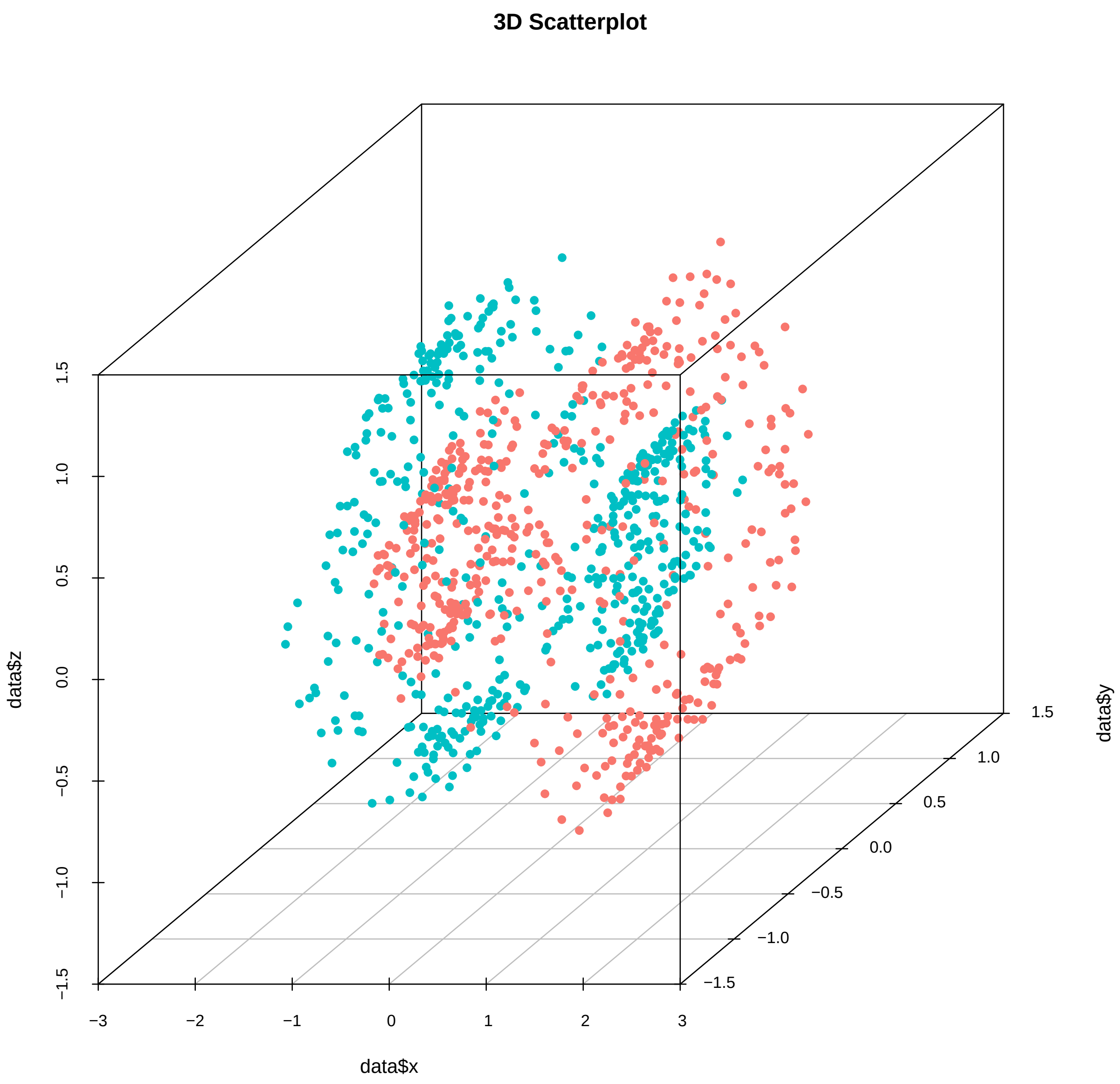




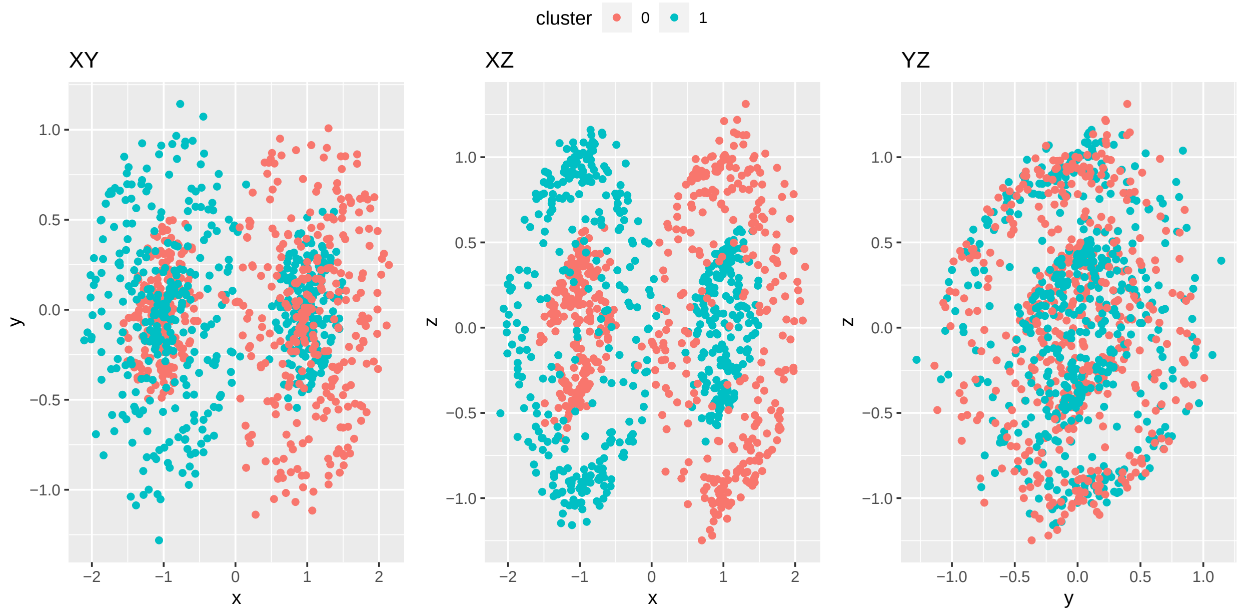
Woah! What's going on here? We got some very distinct chunks, now technically this data is less linearly seperable than before, but the clusters certainly seem more $*clustery*. Cool! I think I prefer PCA first as it seems closer to what a cluster is, then again we are working over principal compinents anyway.

Task 2: Spectral Clustering

For this task we use our old friend the double concentric sphere dataset from our kernelPCA Portfolio which we custom made. Recall the data was:



Which projects onto the 2D plane as:



Which was a pain to to seperate with kernelPCA, we remove the cluster column so we can't see the classes.

Trying Normal Clustering

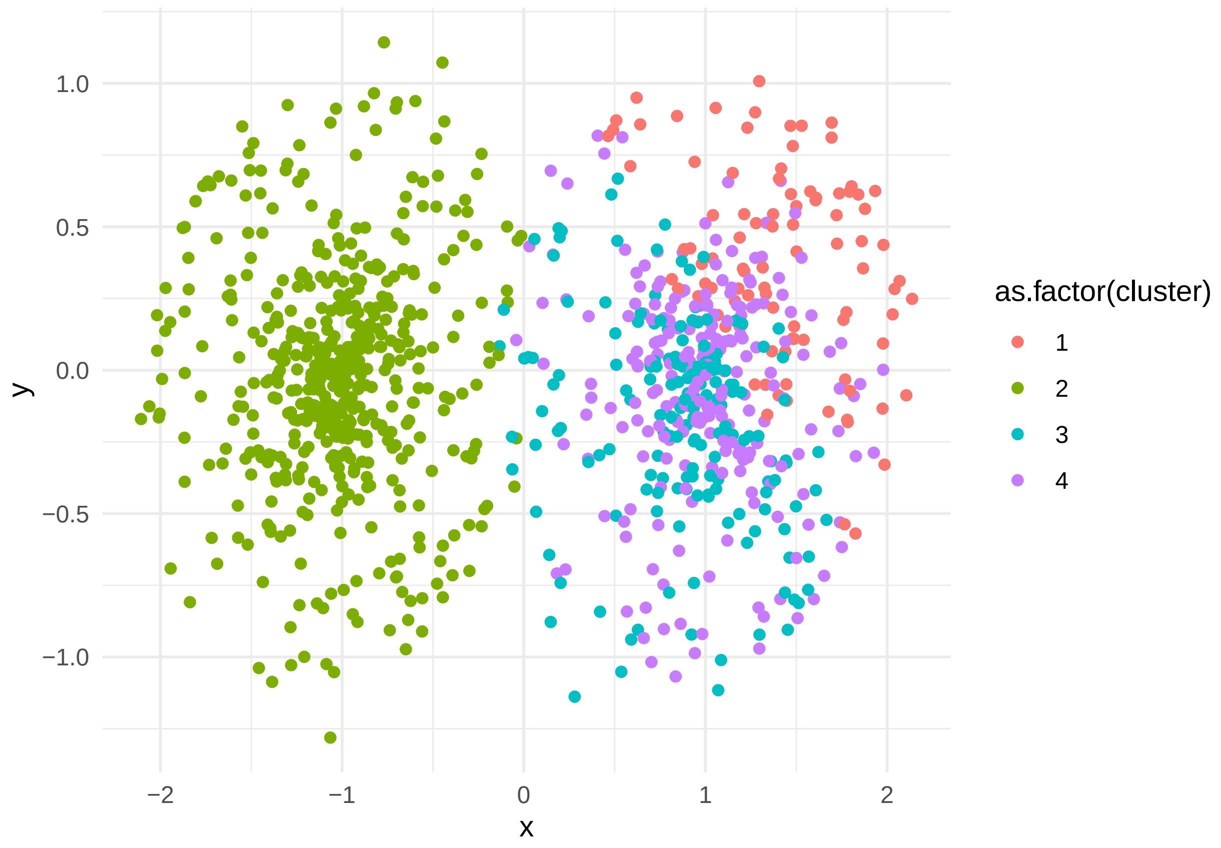
We run the exact same code as before for normal K-means clustering, considering we know that the data has four connected components, we will continue with k=4 for our entire process.

```
# Remove the cluster column
data <- data[,1:3]

# Perform K-means clustering
kmeans_clusters <- kmeans(data, centers = 4, algorithm = "Lloyd")
```

```
## Warning: did not converge in 10 iterations
```

```
# Add the cluster to the sample
data$cluster <- kmeans_clusters$cluster
# Plot the clusters
plt <- ggplot(data, aes(x = x, y = y, color = as.factor(cluster))) + geom_point()+ theme_minimal()
plt
```



I have no idea how its managed that, to give it credit it did notice that the larger spheres are linearly seperable, but it has absolutely no sense of how seperated the smaller and outer balls are.

Spectral Clustering

We now use spectra clustering to hopefully save us from non linear seperability. Usually we would have to use graph method, recursively removing edges, to determine our clusters, with a defined similarity function, however we both know that our data has 4 components and is seperably in euclidean space.

Polynomial kernel

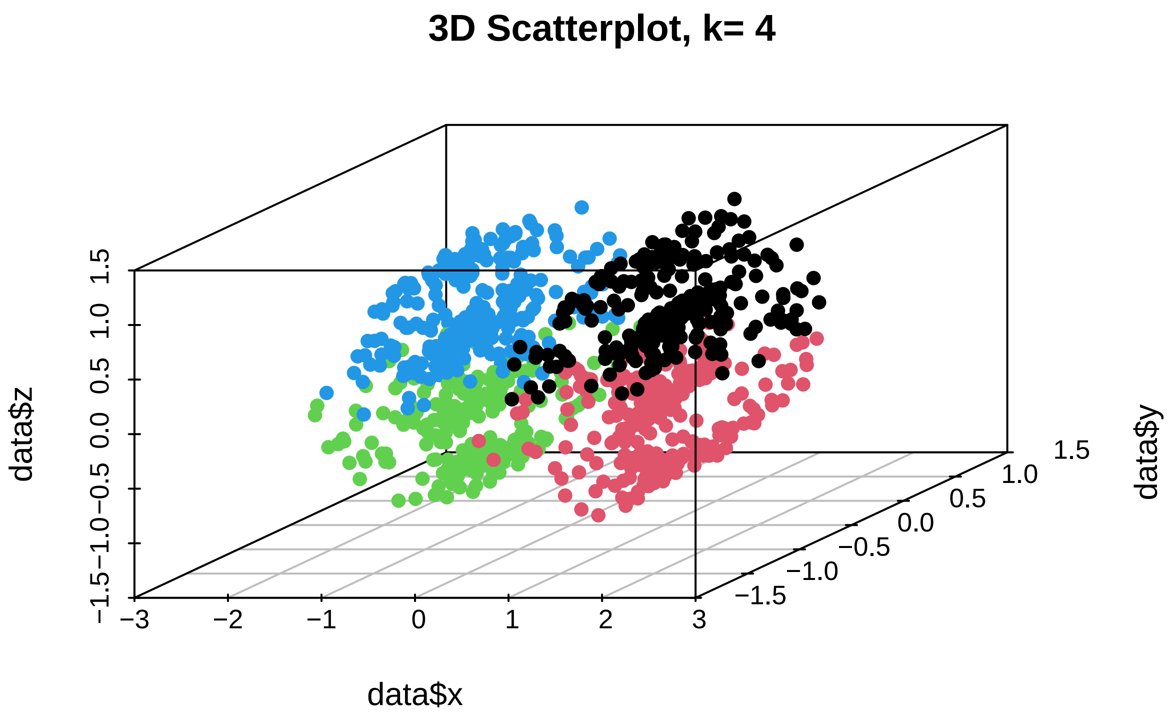
Like with our PCA, a polynomial kernel should be a good way to differentiate our data. We will use `specc` function from the `kernlab` package, apparently it doesn't just do PCA! We attempt this with a 4'th degree polynomial kernel.

```r
library(kernlab)
```

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

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

```r
plot_polynomial_kernel_deg <- function(k){
  #clear cluster from before
  data$cluster <- NULL
  #Run spectral cluster
  spec_cluster <- specc(~x+y+z,
                        data = data,
                        centers = 4,
                        kernel = "polydot",
                        kpar = list(degree = k))
  #Add as factor to dataset
  data$cluster <- as.factor(spec_cluster@.Data)
  # Plot the clusters
  #plot 3dscatterplot
  return(scatterplot3d(data$x, data$y, data$z, color = as.factor(data$cluster), pch = 16, main = paste(
  #No pc required as they are almost identical to the xy, xz and yz projections
}
#Plot 4th degree polynomial kernel
plot_polynomial_kernel_deg(4)
```



Unfortunately we still can't seperate the data, we can clearly see it's working in some fashion, as we are seperating the hemispheres. This isn't surprising as the polynomial kernel didn't perform very well last time either.

RBF kernel

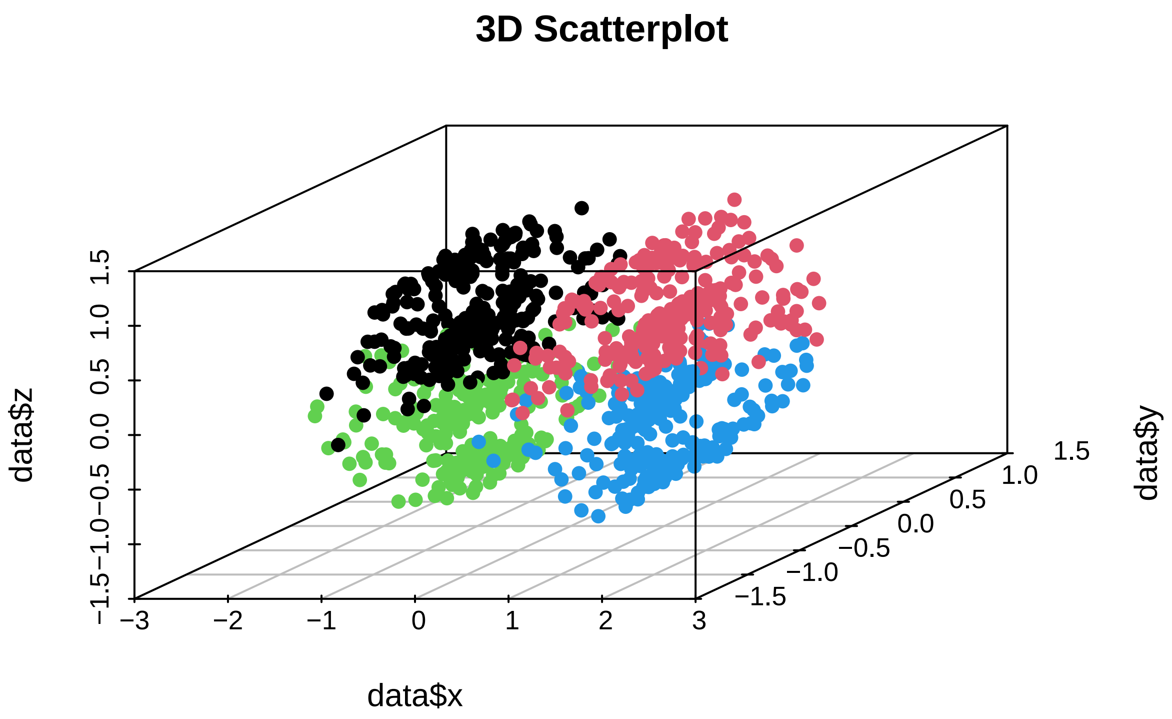
Not much to say here, lets run it again but with RBF! We'll use a median trick to get a good γ value.

```
#Get the median squared distance of the data
data_median <- median(dist(data)^2)

#Run spectral cluster
spec_cluster <- specc(~x+y+z,
                     data = data,
                     centers = 4,
                     kernel = "rbfdot",
                     kpar = list(sigma = data_median))
#Add as factor to dataset

data$cluster <- as.factor(spec_cluster@.Data)

#plot 3dscatterplot
scatterplot3d(data$x, data$y, data$z, color = as.factor(data$cluster), pch = 16, main = '3D Scatterplot
```



Unfortunately we have similar performance to the polynomial kernel, a change in distance may improve performance, possibly something linked to the radius of the spheres themslves. Additionally we could vary the γ too. At least it's better than no kernel at all!

Task 3: Kernel K-Means Clustering

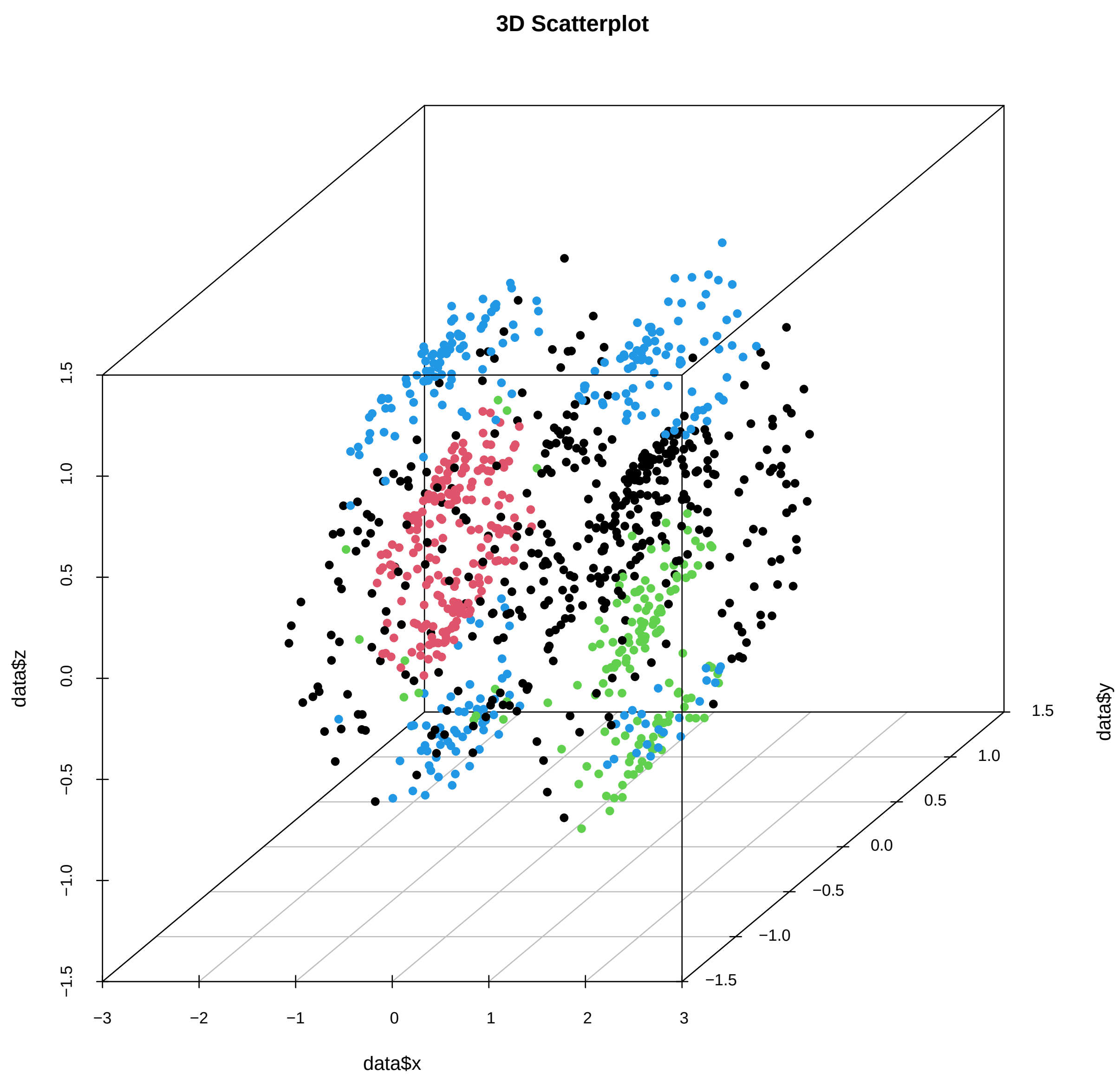
Now on its own we saw that K-means clustering failed spectalularly, but hopefully with our good friend the kenerl we can get some improved performance. Considering Guassian gave the best performance for spectral clustering (and kernel PCA last week) we will go with that for our kernel K-means clustering. We impliment the `kkmeans()` function from `kernlab`.

```r
#empty cluster from before
data$cluster <- NULL
print(data_median)
```

```
## [1] 4.896728
```

```r
#Run kernel kmeans
kernel_kmeans <- kkmeans(~x+y+z,data = data, centers = 4, kernel = "rbfdot", kpar = list(sigma = data_me
#Add as factor to dataset
data$cluster <- as.factor(kernel_kmeans@.Data)

#plot 3dscatterplot
scatterplot3d(data$x, data$y, data$z, color = as.factor(data$cluster), pch = 16, main = '3D Scatterplot
```



It's so close! I think the variation we're experiencing will be because of the choice of γ, we could formalise this by running over a range of γ values, but we'll settle for a few more examples, we choose $\gamma \in \{5, 6, 7\}$ in these examples.

"'{r.echo = FALSE,fig.align='center',fig.width=10,fig.height=10,warning = FALSE}

plot_kernel_kmeans_gamma <- function(gamma){

kernel_kmeans <- kkmeans(~x+y+z,data = data, centers = 4, kernel = "rbfdot", kpar = list(sigma = gamma)) data$cluster < -as.factor(kernel_kmeans@.Data) scatterplot3d(data$x, data$y, data$z, color = as.factor(data$cluster), pch = 16, main = paste('3D Scatterplot, gamma =',gamma)) }

plot_kernel_kmeans_gamma(5.75) plot_kernel_kmeans_gamma(6.25) plot_kernel_kmeans_gamma(6.5) "'

With more time we could train this model to find the optimal γ, but these pretty plots will do for now.