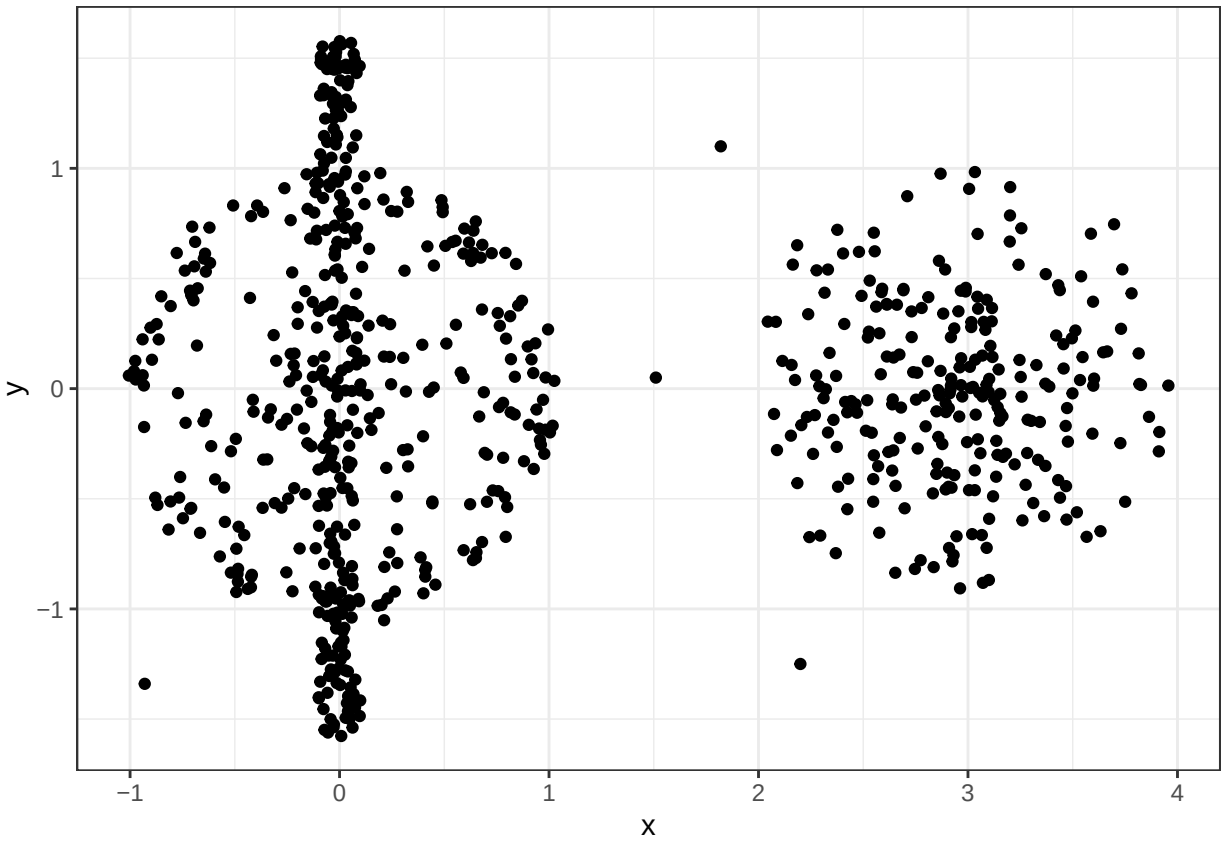
Assignment 12

Jeff Gould

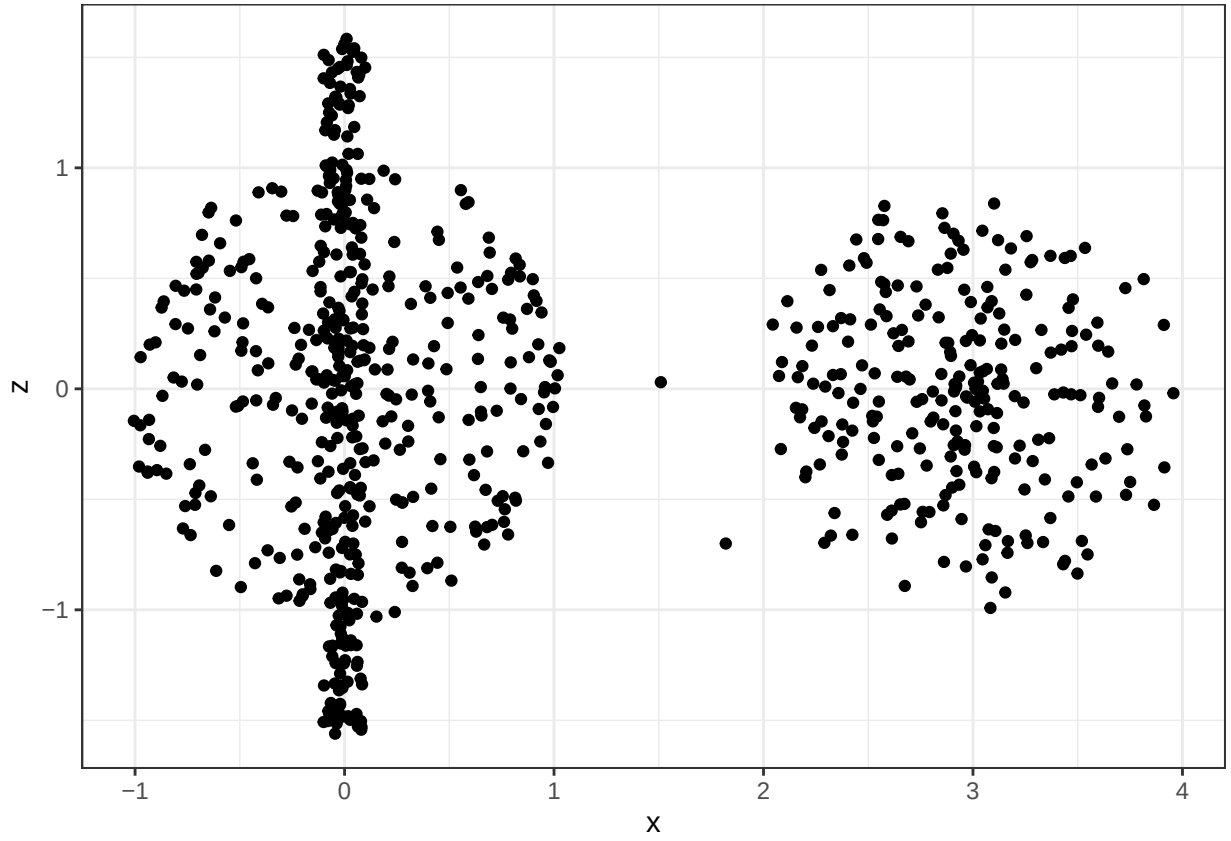
11/11/2020

First let's plot the data to get an idea of how many clusters we want to group the data into. While not as easy to see with this plot, using the animated chart function plotly, which doesn't work with pdf, we can visualize more easily. There is clearly one spherical cluster off to the side. Then there is an outer ring around one mass, a shell of points inside that, and then a mass of points inside the shell. So we will look to cluster the data into 4 groups.

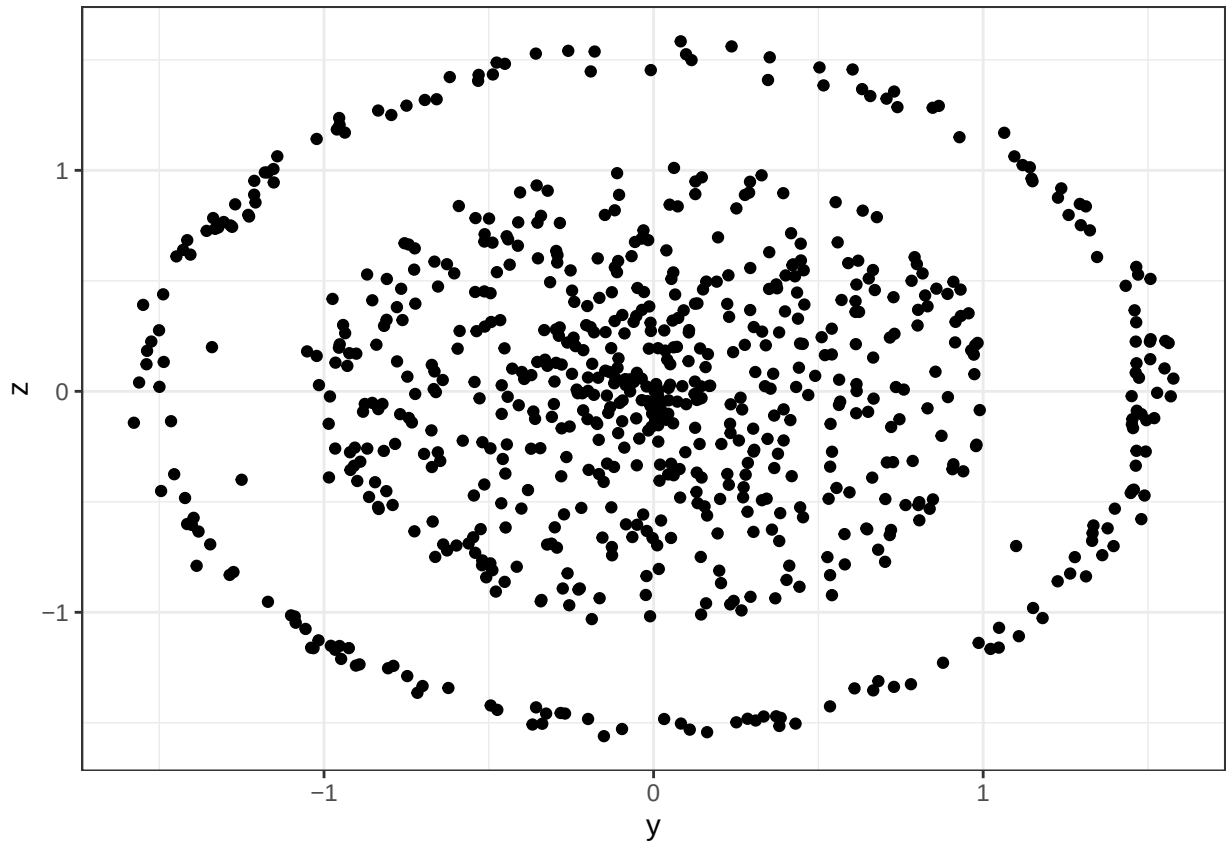
```
theme_set(theme_bw())
Cluster3D <- read.arff("Cluster3D.arff")

ggplot(data = Cluster3D, aes(x = x, y = y)) +
  geom_point()
```



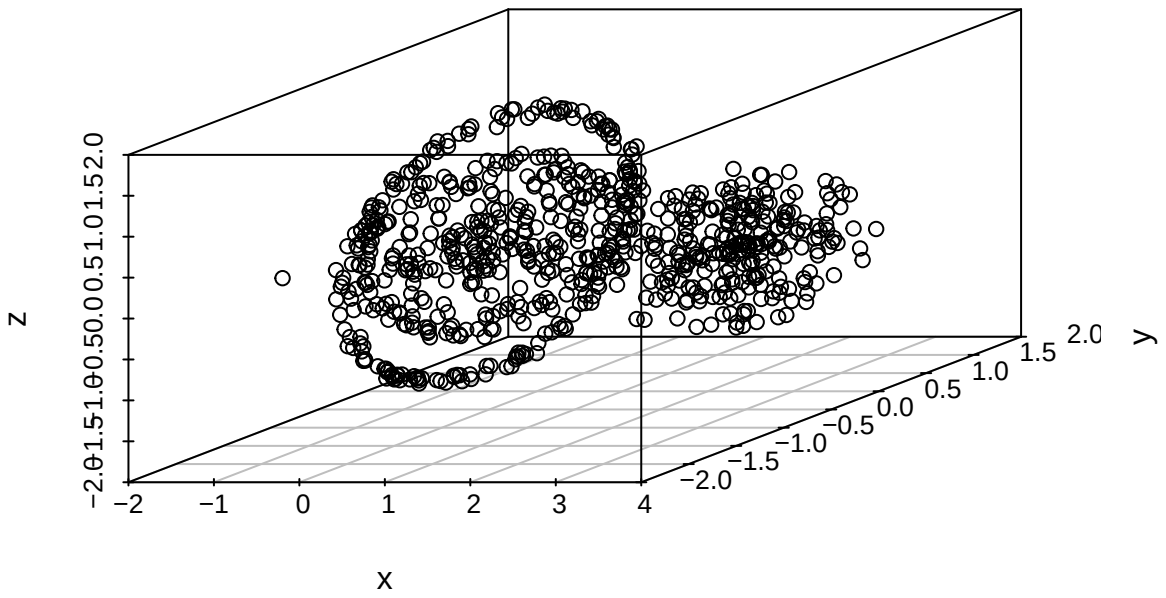
```
ggplot(data = Cluster3D, aes(x = x, y = z)) +
  geom_point()
```




```
ggplot(data = Cluster3D, aes(x = y, y = z)) +
  geom_point()
```




```
attach(Cluster3D)
scatterplot3d::scatterplot3d(x = x, y = y, z = z)
```




```
plotly::plot_ly(x = x, y = y, z = z, type="scatter3d", data = Cluster3D)
```

One possible option is to use diffusion maps, a strong technique for non-linear dimension reduction

- For each $x^{(i)}$, select the 4 nearest neighbors and connect with an edge
- Assign a weight w_{ij} to each edge with $w_{ij} = \exp^{-||x^{(i)}-x^{(j)}||^2/\delta}$ if $(i,j) \in E$
- Create a Kernel matrix K, where $K_{ij} = w_{ij}$, and since $w_{ij} = w_{ji}$, then $K_{ij} = K_{ji}$, and K is symmetric
- Create a matrix A, where each row i of A is the i^{th} row of K divided by the row sum of K_i, that is the row normalization of K
- Consider A as a transition probabilty matrix from point to point
- Let D by a diagonal matrix where each D_{ii} is the i^{th} row sum of K
- $S = D^{1/2}AD^{1/2}$, a symmetric matrix
- From here, take the eigen data of S, and multiply the $\lambda_i^t \cdot r_i$, where λ is the eigen value and r is the corresponding right eigenvector, and t is the number of times we run the chain
- after running the diffusion map to reduce dimensions, use K-nearest neighbors to cluster the reduced dimension data

We are able to effectivley cluster the data into 4 groups as found above. Again, it is easier to see the distinction between the shell and the mass inside it with the interactive function plotly, but that only works with html output.

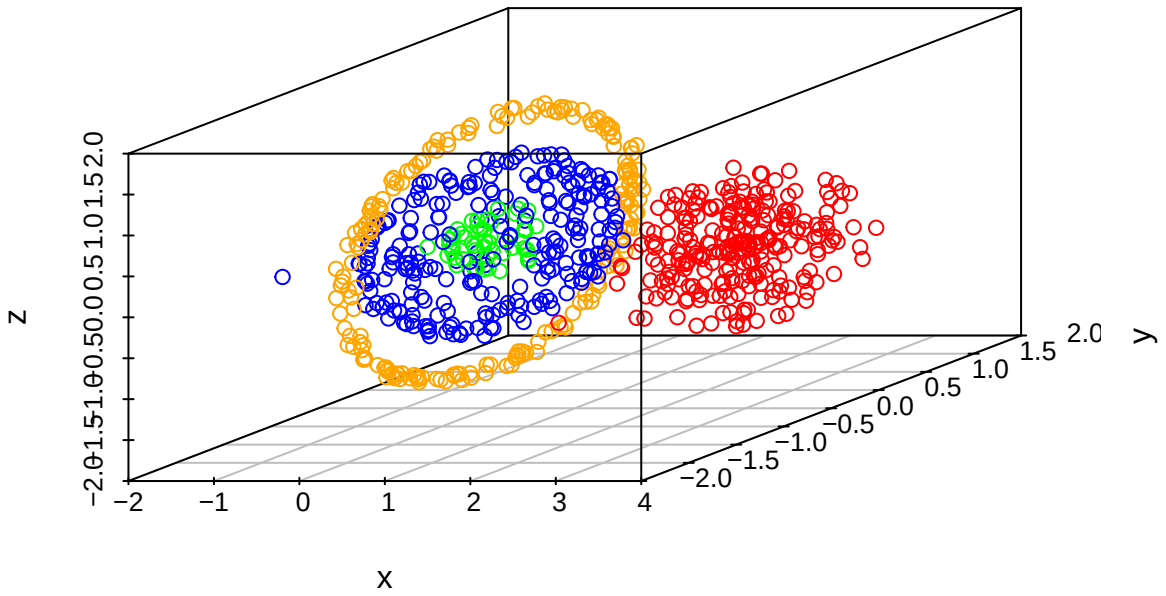
```
#m <- read_csv("TripleHelix.csv")
m <- Cluster3D
N <- nrow(m)
K <- matrix(0, nrow=N, ncol=N)
```

```r
delta <- .1

distMat <- dist(m) %>% as.matrix()

for(i in 1:N){
  distances <- distMat[i, ]
  closest_4 <- order(distances)[2:5]
  K[i,closest_4] <- exp(-distances[closest_4]/delta)
  K[closest_4,i] <- exp(-distances[closest_4]/delta)
}


# create D and A
D <- diag(rowSums(K))/sum(K)
A <- t( apply(K, 1, function(r) r/sum(r)) )

# create symmetric matrix
S <- D^{1/2} %*% A %*% solve(D^{1/2})

# compute eigenvalues and grab first three
eig.out <- eigen(S)
ind <- order(eig.out$values %>% abs, decreasing = T)
q <- eig.out$vectors[,ind][,1:3]
lambdas <- eig.out$values[ind][1:3]

r <- solve(D^{1/2}) %*% q

# let's just check r1 and lambda1
#r[,1]
#lambdas[1]



#2-d projection
# t_vals <- c(1,1000, 10000)
# for (t in t_vals) {
#   y <- cbind(lambdas[2]^t*r[,2], lambdas[3]^t*r[,3])
#   g <- ggplot(data=data.frame(y1=y[,1], y2=y[,2])) +
#     geom_jitter(aes(x=y1, y=y2)) +
#     ggtitle(paste("time", t))
#   print(g)
# }

set.seed(3)
t <- 1000
y <- cbind(lambdas[2]^t*r[,2], lambdas[3]^t*r[,3])

knn <- kmeans(x = y, centers = 4)

rm(y)
Cluster3D$assignment <- as.factor(knn$cluster)
Cluster3D <- Cluster3D %>%
  mutate(color = case_when(
```

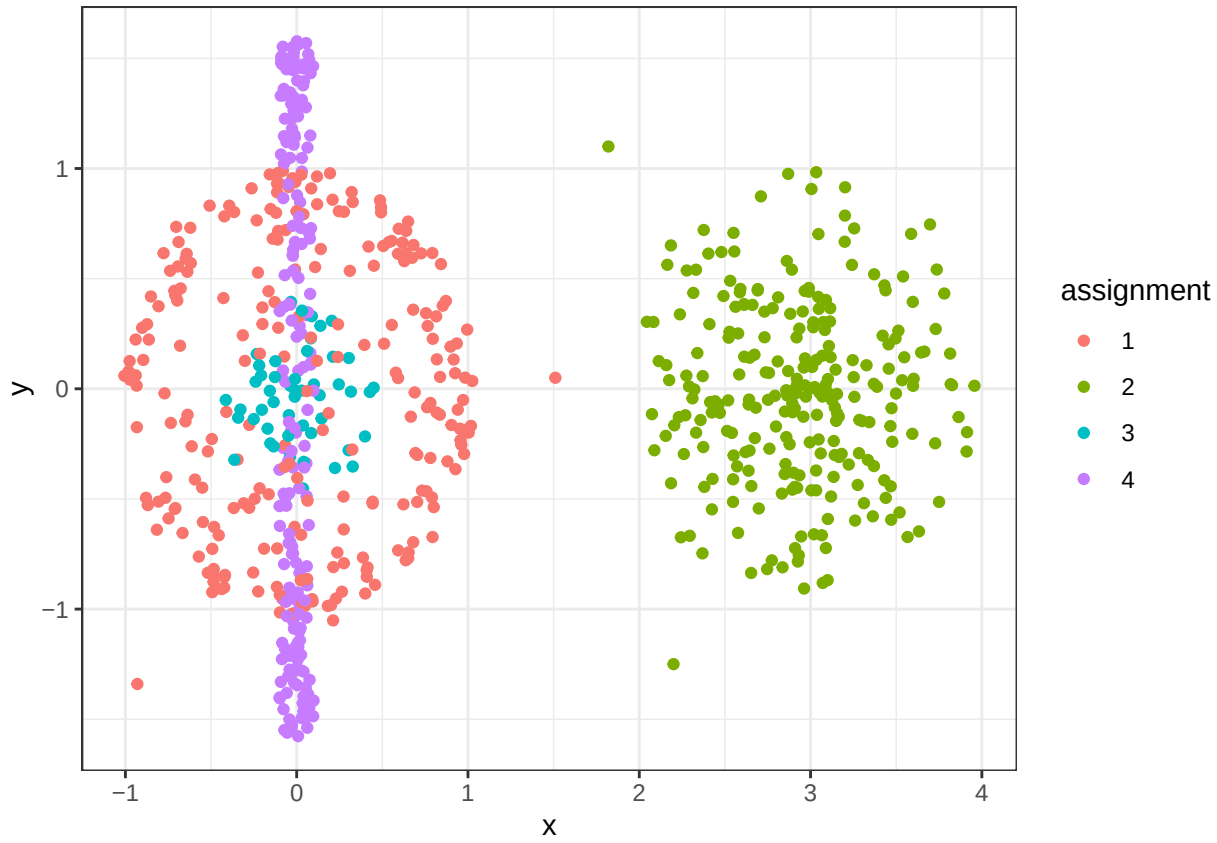
```
    assignment == "1" ~ "blue",
    assignment == "2" ~ "red",
    assignment == "3" ~ "green",
    assignment == "4" ~ "orange",
    TRUE ~ "black"
  ))
attach(Cluster3D)
```

```
## The following objects are masked from Cluster3D (pos = 3):
##
##     x, y, z
```

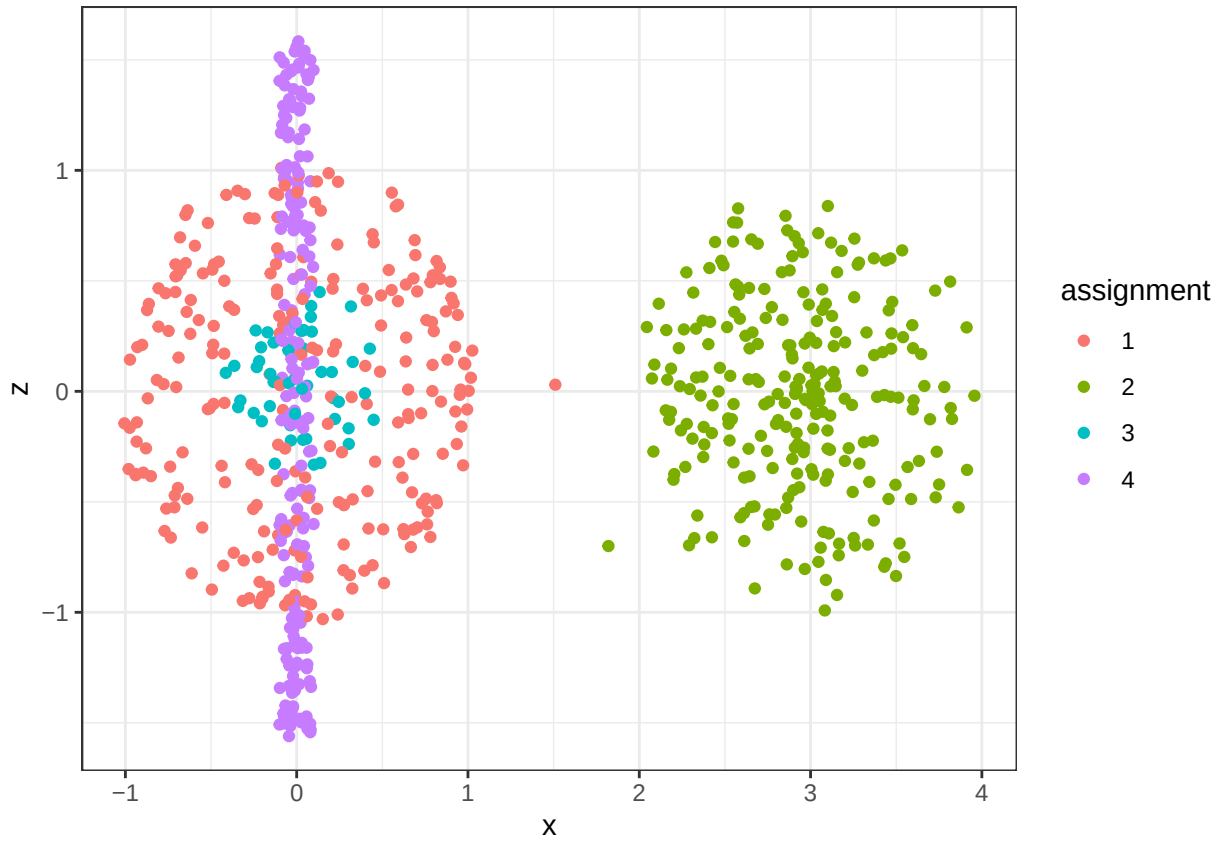
```
scatterplot3d::scatterplot3d(x = x, y = y, z = z, color = Cluster3D$color)
```



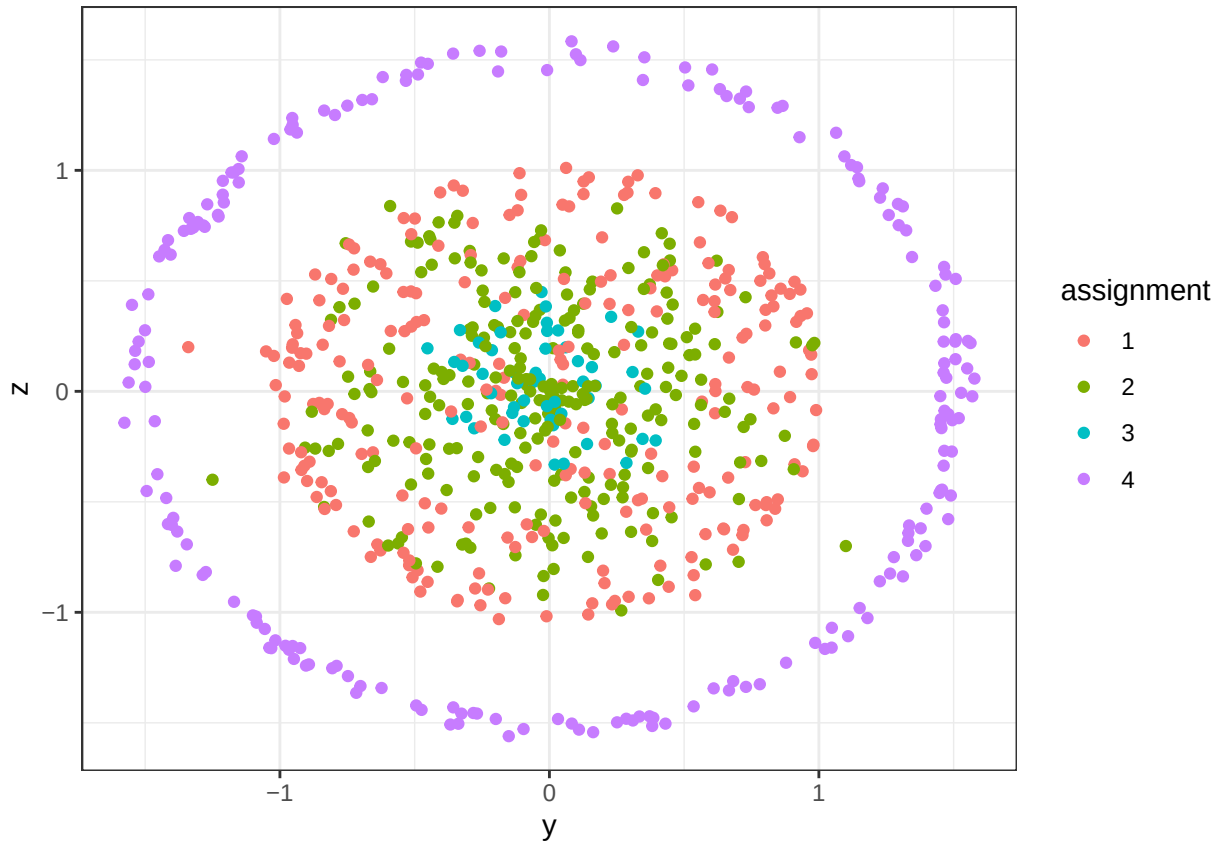

```
ggplot(Cluster3D, aes(x = x, y = y, color = assignment)) +
  geom_point()
```




```
ggplot(Cluster3D, aes(x = x, y = z, color = assignment)) +
  geom_point()
```



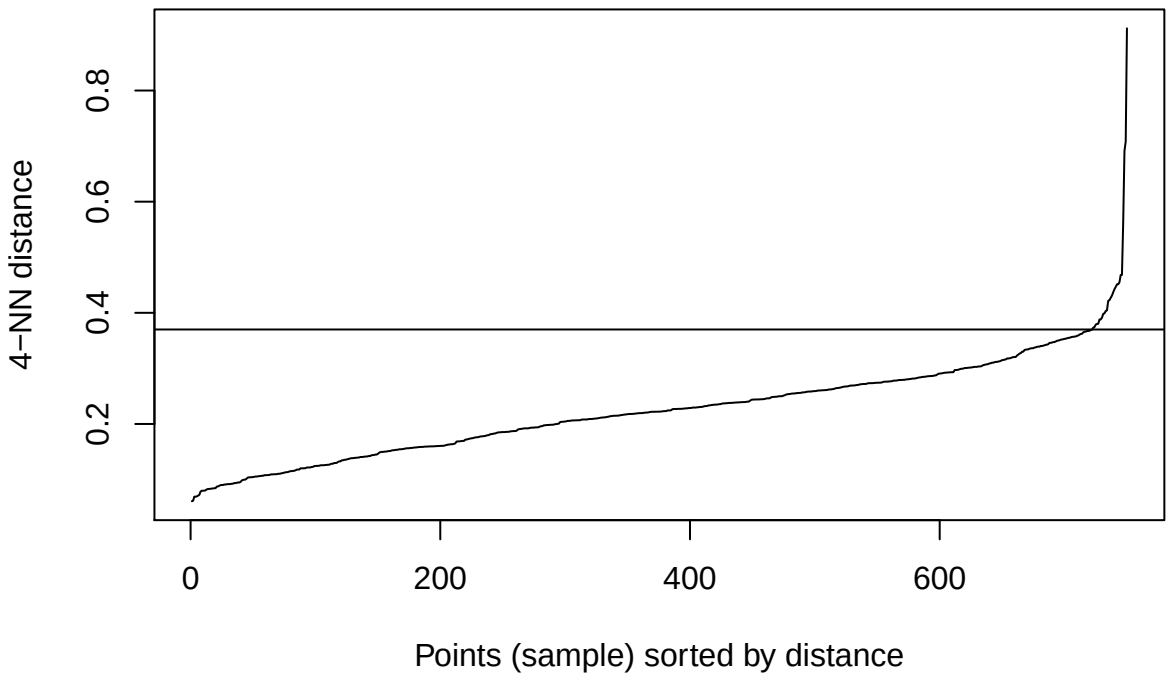

```
ggplot(Cluster3D, aes(x = y, y = z, color = assignment)) +
  geom_point()
```



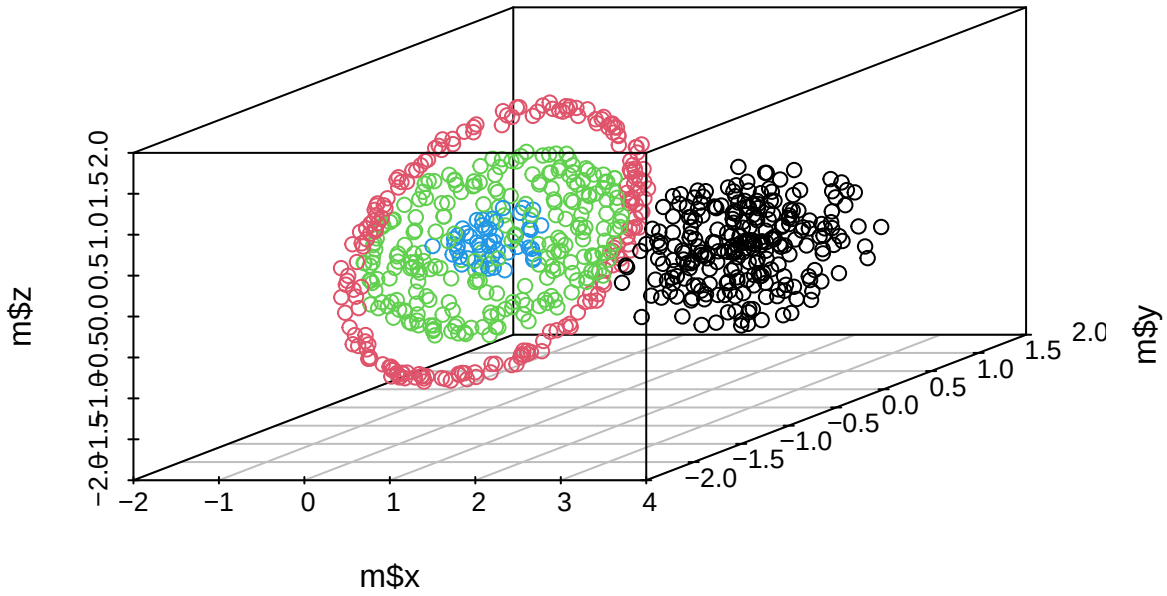

```
attach(Cluster3D)
scatterplot3d::scatterplot3d(x = x, y = y, z = z, color = Cluster3D$color)
plotly::plot_ly(data = Cluster3D, x = x, y = y, z = z, type="scatter3d", color = assignment)
```

Another option would be to use density based mapping. We see the results are very similar to what we got with diffusion mapping, maybe even a little cleaner. Both options create very similar results on the test data. Density based mapping was a little simpler, and given the already low dimensions of the data, density based mapping would probably be better. But on data such as the triple-helix, diffusion mapping worked better. So both are effective tools on non-linear data when used properly.

```
library(cluster)
library(dbscan)
kNNdistplot(m)
abline(h = 0.37)
```




```
results <- dbscan(m, eps = 0.37)
labels <- results$cluster
scatterplot3d::scatterplot3d(x = m$x, y = m$y, z = m$z, color = labels)
```


m$z
m$x
m$y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
−2
−1
0
1
2
3
4