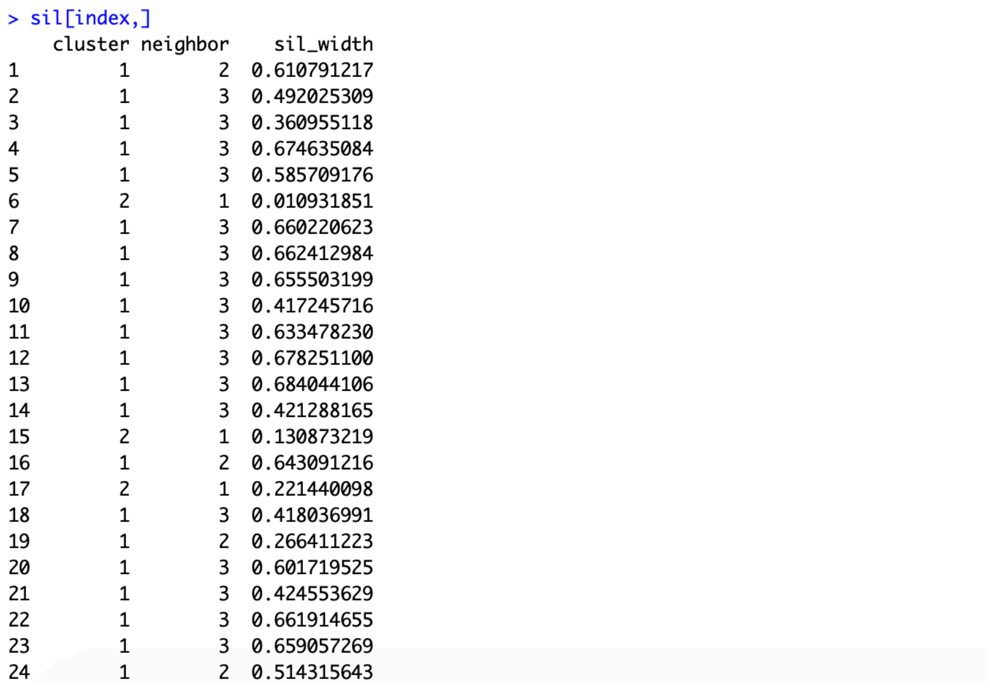
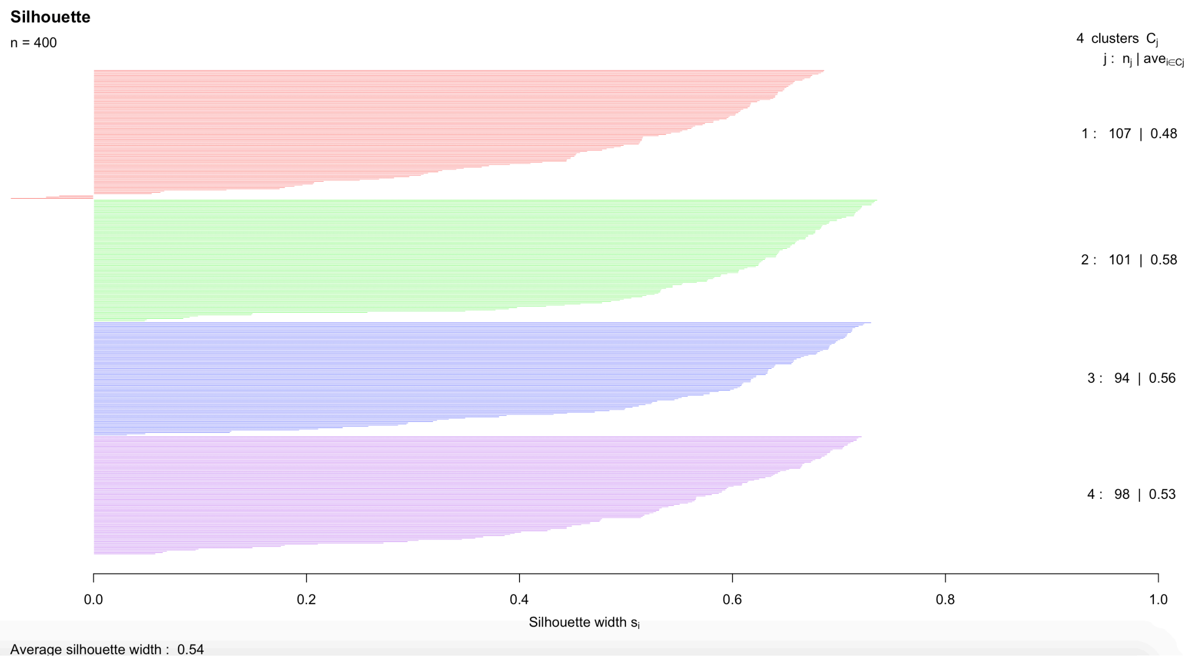
5.1

(a)

The silhouette index for each datapoint is shown below.

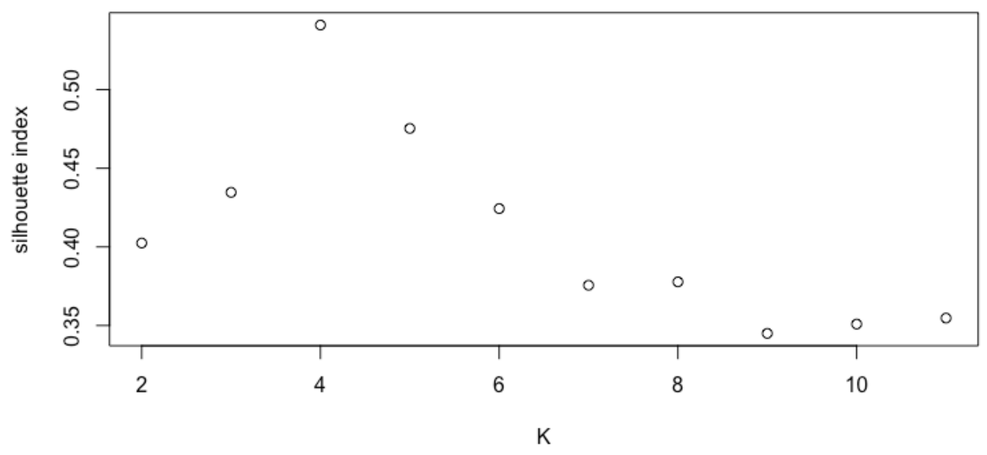


```
> sil[index,]
   cluster neighbor   sil_width
1        1        2 0.610791217
2        1        3 0.492025309
3        1        3 0.360955118
4        1        3 0.674635084
5        1        3 0.585709176
6        2        1 0.010931851
7        1        3 0.660220623
8        1        3 0.662412984
9        1        3 0.655503199
10       1        3 0.417245716
11       1        3 0.633478230
12       1        3 0.678251100
13       1        3 0.684044106
14       1        3 0.421288165
15       2        1 0.130873219
16       1        2 0.643091216
17       2        1 0.221440098
18       1        3 0.418036991
19       1        2 0.266411223
20       1        3 0.601719525
21       1        3 0.424553629
22       1        3 0.661914655
23       1        3 0.659057269
24       1        2 0.514315643
```



(b)

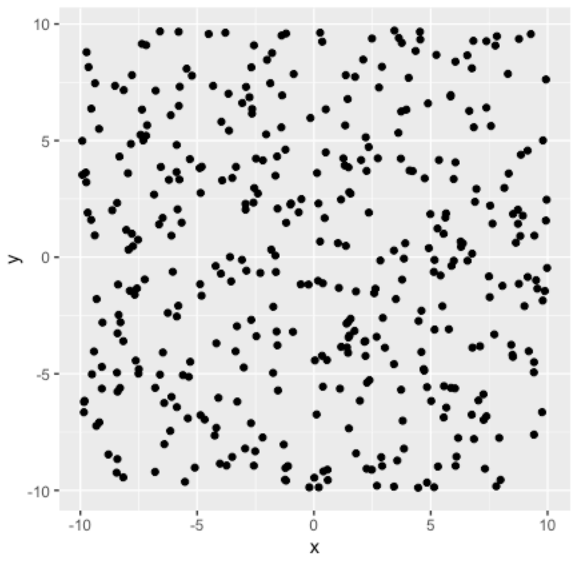
The average silhouette index for different clustering choices is plotted below:



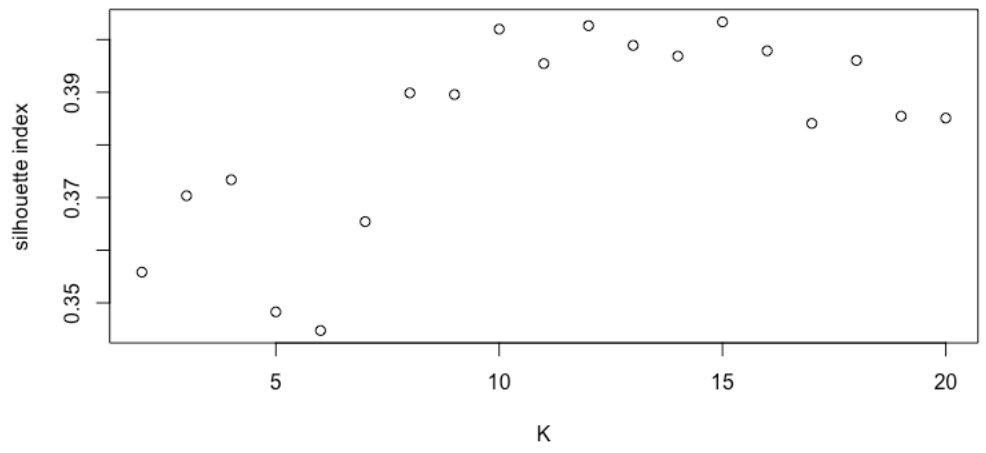
Here we see that the silhouette index peaks at $K = 4$. We know this is the optimal choice because "*simdatxy*" was generated from four Gaussian distributions centered at $(0,0)$, $(0,8)$, $(8,0)$ and $(8,8)$ with the same standard deviation of 2. At $K = 4$, we may approximate $A(i) = 2$ and $B(i) = 4$. Hence $S(i) = \frac{B(i)-A(i)}{max_i(A(i),B(i))} \approx 0.5$.

(c)

400 random datapoints, whose x and y components follow uniform distributions in the range $[-10,10]$, are generated using the "*runif()*" function.



By varying the number of clusters, we see that the silhouette index fluctuates around 0.375 and its profile lacks an obvious peak. As a result, we cannot infer what number of clusters is optimal.



R Code:

```
#5.1
#a
library("dplyr")
simdat = lapply(c(0, 8), function(mx) {
  lapply(c(0,8), function(my) {
    tibble(x = rnorm(100, mean = mx, sd = 2),
           y = rnorm(100, mean = my, sd = 2),
           class = paste(mx, my, sep = ":"))
  }) %>% bind_rows
}) %>% bind_rows
simdatxy = simdat[, c("x", "y")]

library("cluster")
pam4 <- pam(simdatxy, 4)
sil <- silhouette(pam4, 4)
index <- order(as.array(as.numeric(row.names(sil))))
sil[index,]
```

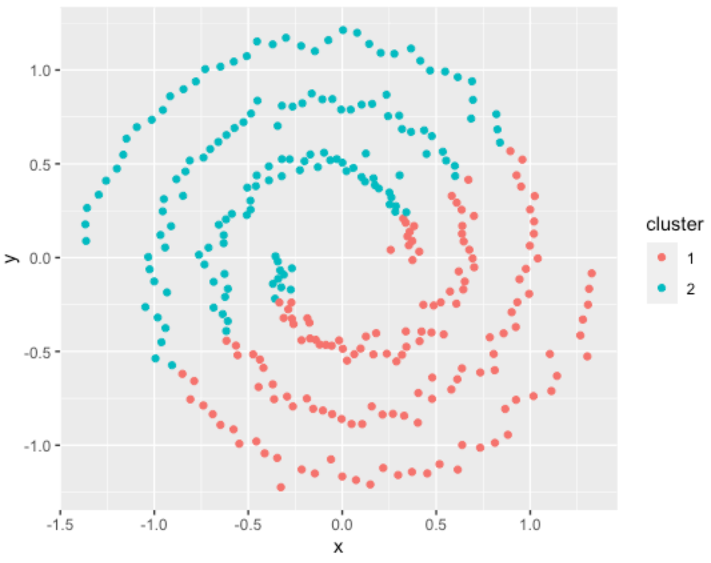
```
plot(sil, col=c("red","green","blue","purple"), main="Silhouette")

#b
sil_wid <- function(k){
  p <- pam(simdatxy,k)
  s <- silhouette(p,k)
  return(mean(s[,"sil_width"]))
}

k_vector <- c(2:11)
width_vector <- c()

for (val in k_vector){
  val
  w <- sil_wid(val)
  width_vector <- c(width_vector,w)
}
plot(k_vector,width_vector,xlab = "K", ylab = "silhouette index")

#c
library(ggplot2)
simdatxy2 <- tibble(x = runif(400,min=-10,max=10), y = runif(400,min=-10,max=10))
ggplot(simdatxy2, aes(x = x, y = y)) + geom_point() +
  coord_fixed()

sil_wid2 <- function(k){
  p <- pam(simdatxy2,k)
  s <- silhouette(p,k)
  return(mean(s[,"sil_width"]))
}

k_vector <- c(2:20)
width_vector <- c()
```

```
for (val in k_vector){
  val
  w <- sil_wid2(val)
  width_vector <- c(width_vector,w)
}
plot(k_vector,width_vector,xlab = "K", ylab = "silhouette index")
```

5.3

(a)

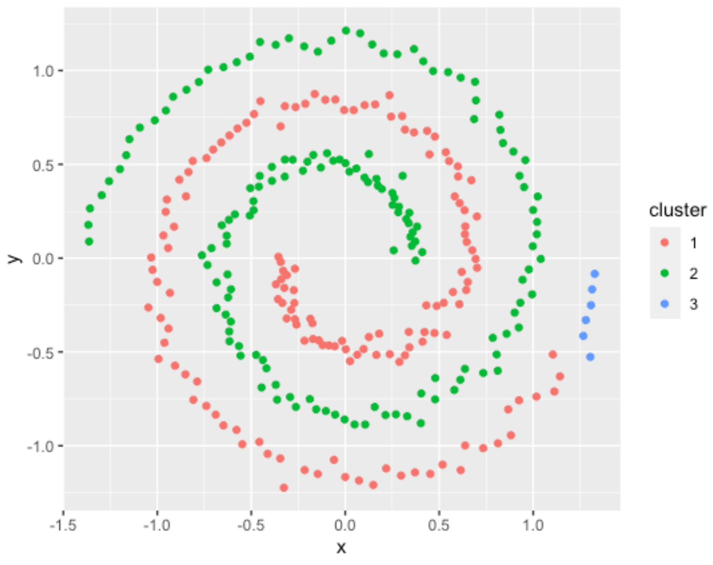
"kmeans()" was used to separate the *spirals* data into 2 clusters. It is evident that the result is far from satisfactory.



(b)

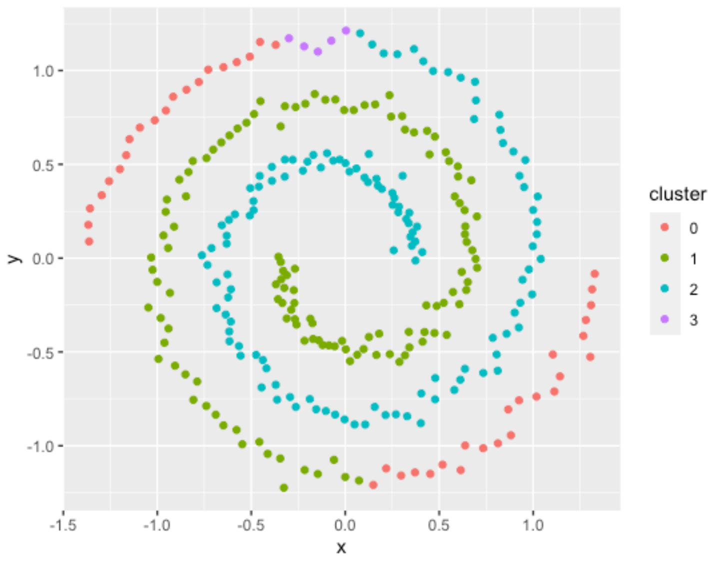
Using *"dbscan::dbscan()"* has proven to provide us with better results, except a small amount of misclassifications at the tail. The *eps* and *minPts* parameters were 0.15 and 3 respectively.

The density-based clustering method starts from a randomly selected seed. Branching out from that seed, all data points within *eps* are labeled as the next frontier where breadth-first search is executed. When all nodes of the outermost layer no longer have enough close-by neighbours to hop to, a new class is created along with a new seed and the algorithm repeats [1].

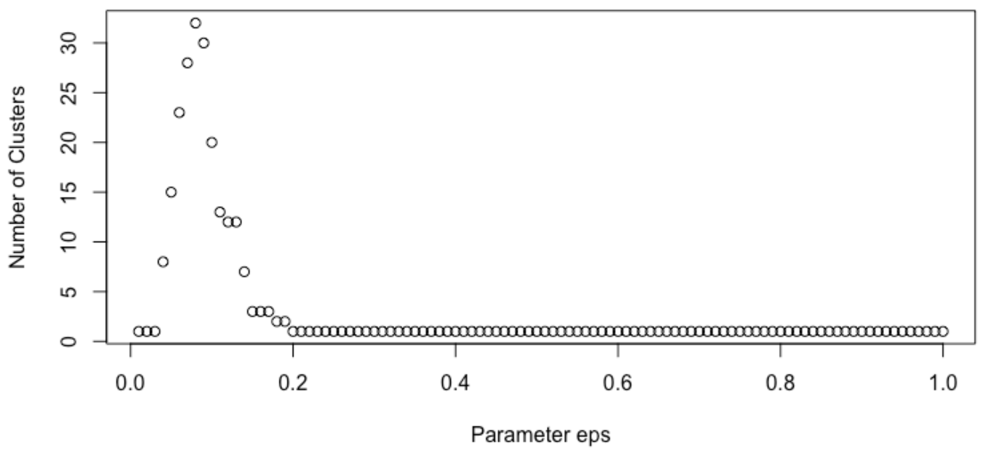


(c)

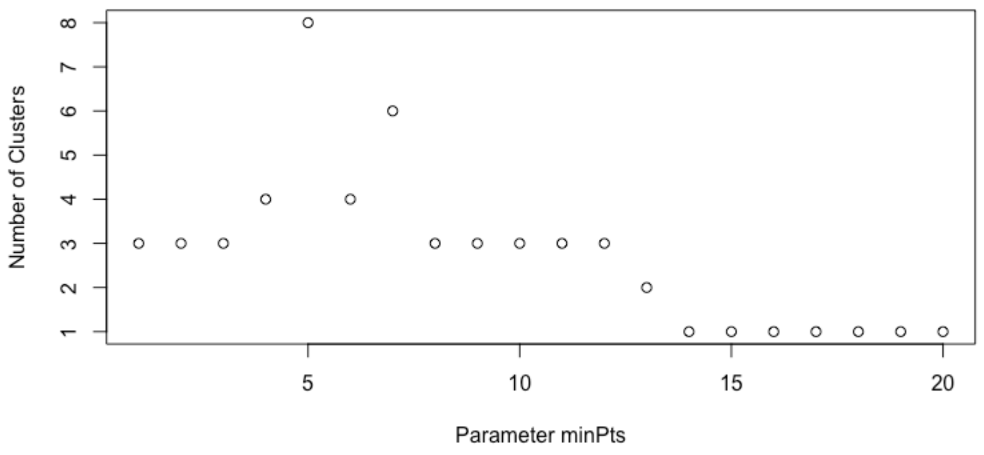
As soon as the *eps* and *minPts* parameters leaves their ranges of tolerance 0.15 ± 0.02 and 3 ± 2, the clustering algorithm fails and produces too many groups.



Setting *minPts* constant at 3, sweeping *eps* from 0.01 to 1 with increments of 0.01 changes the number of clustering produced as shown in the Figure below:



Setting *eps* constant at 0.15, sweeping *minPts* from 1 to 20 with increments of 1 changes the number of clustering produced as shown in the Figure below:



It is thus evident that the clustering algorithm used by *dbscan* is not robust for this non-convex set. The more efficient way would be to use kernel clustering to transform the datapoints into a linearly separable space.

R Code:

```
#5.3
#a
library(kernlab)
data(spirals)
s_data <- tibble(x = spirals[,1], y = spirals[,2],cluster = as.factor(kmeans(spirals, centers=2)$cluster))
ggplot(s_data, aes(x = x, y = y, col = cluster)) + geom_point() + coord_fixed()

#b
library(dbscan)
s_data2 <- tibble(x = spirals[,1], y = spirals[,2],cluster = as.factor(dbscan::dbscan(spirals, eps = 0.15,
minPts = 3)$cluster))
ggplot(s_data2, aes(x = x, y = y, col = cluster)) + geom_point() + coord_fixed()

#c
s_data3 <- tibble(x = spirals[,1], y = spirals[,2],cluster = as.factor(dbscan::dbscan(spirals, eps = 0.15,
minPts = 3)$cluster))
ggplot(s_data3, aes(x = x, y = y, col = cluster)) + geom_point() + coord_fixed()

e_vector <- seq(0.01,1,0.01)
ncluster_vector <- vapply(e_vector, function(x){length(unique(as.factor(dbscan::dbscan(spirals, eps = x,
minPts = 3)$cluster)))},numeric(1))
plot(e_vector,ncluster_vector, xlab="Parameter eps", ylab="Number of Clusters")

P_vector <- seq(1,20,1)
ncluster_vector2 <- vapply(P_vector, function(x){length(unique(as.factor(dbscan::dbscan(spirals, eps =
0.15, minPts = x)$cluster)))},numeric(1))
plot(P_vector,ncluster_vector2, xlab="Parameter minPts", ylab="Number of Clusters")
```

Reference

[1] “DBSCAN Clustering Tutorial - Nearist.ai - Medium.” [Online]. Available: https://medium.com/nearist-ai/dbscan-clustering-tutorial-dd6a9b637a4b. [Accessed: 27-Jun-2020].