# Clustering

*Brandon Kozak*

*30/10/2019*

```r
library(tidyverse)
```

```
## -- Attaching packages -----------------------------------------------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts --------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(BiocManager)
library(here)
```

```
## here() starts at C:/Users/brand/Desktop/STAT_4600
```

## Goals for this chapter

- Study different types of data that work well with clustering
- Learn the types of measures that determine clusters
- Uncover latent clustering via partitioning the data into tighter sets
- Look at non parametric algorithms such as k-means, k-medoids on real cell data
- Hierarchical clustering
- Use bootstrap to validate clusters

## 5.2 Why cluster?

### 5.2.1

It can lead to discoveries, for example in cancer biology.

Clustering is more general than the EM Approach and can be applied to more complex data. This is because many of the clustering techniques do not assume anything about the underlying generating mechanism of the data.

Bioconductor has 212 packages on clustering as of now!

## 5.3 How do we measure similarity?

We are looking at bird data.

Our first question is what variables are we going to use. Weight and size will give different clustering compared to diet or habitat.

Next we list a couple of ways to define distance.

Assume our data exists in a p-dimensional space, such that point $A = (a_1, ..., a_p)$ and point $B = (b_1, ..., b_p)$.

- **Euclidean Distance**: defined to be the square root of the sum of the squared differences between each component of each point. That is, $d(A, B) = \sqrt{(a_1 - b_1)^2 + ... + (a_p - b_p)^2}$
- **Manhattan Distance**: defined to be the sum of the absolute differences in each component of each point. That is, $d(A, B) = |a_1 - b_1| + ... + |a_p - b_p|$
- **Maximum Distance**: defined to be the max of the absolute differences between A and B, that is $d_{\infty(A,B)=\max|a_i - b_i|}$
- **Weighted Euclidean Distance**: similar to the euclidean distance, but now we apply certain weight $(w_1, ..., w_p)$ to each difference.
- **Mahalanobis Distance**: is a special type of weighted euclidean distance that uses the sample covariance matrix of the data, that is $d(A, B) = \sqrt{(A - B)^T S_n^{-1}(A - B)}$, where $S_n^{-1}$ is the inverse of the sample covariance matrix.
- **Malinowski Distance**: is a general form of the Euclidean distance, where we can take the exponent to be $m$ rather than 2. That is, $d(A, B) = ((a_1 - b_1)^m + ... + (a_p - b_p)^m)^{\frac{1}{p}}$
- **Edit and Hamming Distance**: Simplify the number of differences for each index of a character string. Typical used in DNA sequences.

### 5.3.1 Computations related to distances in R

We can use the dist() function. By default we get the euclidean distance, but can set the “method” parameter

```r
# Our points
mx  = c(0, 0, 0, 1, 1, 1)
my  = c(1, 0, 1, 1, 0, 1)
mz  = c(1, 1, 1, 0, 1, 1)
mat = rbind(mx, my, mz)

dist(mat)
```

```
##          mx       my
## my 1.732051
## mz 2.000000 1.732051
```

```r
dist(mat, method = "binary")
```

```
##           mx        my
## my 0.6000000
## mz 0.6666667 0.5000000
```

What about the Jaccard distance?

```
mut =  read_csv(here("Data","HIVmutations.csv"))
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )

## See spec(...) for full column specifications.
```

```
mut[1:3, 10:16]
```

```
## # A tibble: 3 x 7
##    p32I  p33F  p34Q  p35G  p43T  p46I  p46L
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     0     1     0     0     0     0     0
## 2     0     1     0     0     0     1     0
## 3     0     1     0     0     0     0     0
```

```
library("vegan")
```

```
## Loading required package: permute

## Loading required package: lattice

## This is vegan 2.5-6
```

```
mutJ = vegdist(mut, "jaccard")
mutC = sqrt(2 * (1 - cor(t(mut))))
mutJ
```

```
##           1         2         3         4
## 2 0.8000000
## 3 0.7500000 0.8888889
## 4 0.9000000 0.7777778 0.8461538
## 5 1.0000000 0.8000000 0.8888889 0.9000000
```

## 5.4 Non parametric mixture detection

There are two k-methods to clustering. k-means and k-medoids.

k-means uses the mean in it's algorithms, while k-medoids uses the true center.

One example of a k-medoid algorithm is the partitioning around medoids (PAM) algorithm. The steps are as follows:

- Start with p features and n observations
- Randomly pick k distinct cluster centers out of the n observations.
- Assign each of the remaining observations to the group whose center is the closest
- For each group, choose a new center from the observations in that group, such that the sum of the distances of the groups members to the new center is minimal.
- Repeat steps 3 and 4 until the groups stabilize.

### 5.4.2 Tight clusters with re sampling

We can repeat clustering algorithms on the same data but with different starting points. Observations that are almost always grouped together are called tight clusters.

Here is an example using the clusterExperiment package.

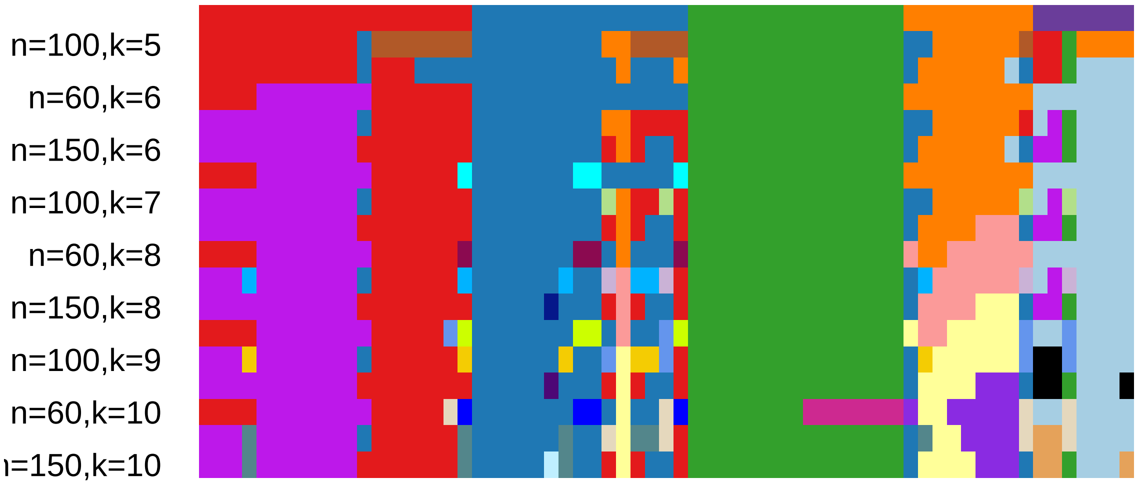
```r
library("clusterExperiment")
```

```
## Loading required package: SingleCellExperiment

## Loading required package: SummarizedExperiment

## Loading required package: GenomicRanges

## Loading required package: stats4

## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter,
##     Find, get, grep, grepl, intersect, is.unsorted, lapply, Map,
##     mapply, match, mget, order, paste, pmax, pmax.int, pmin,
##     pmin.int, Position, rank, rbind, Reduce, rownames, sapply,
##     setdiff, sort, table, tapply, union, unique, unsplit, which,
##     which.max, which.min

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'
## The following objects are masked from 'package:dplyr':
##
##     first, rename

## The following object is masked from 'package:tidyr':
##
##     expand

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

##
## Attaching package: 'IRanges'

## The following objects are masked from 'package:dplyr':
##
##     collapse, desc, slice

## The following object is masked from 'package:purrr':
##
##     reduce

## The following object is masked from 'package:grDevices':
##
##     windows

## Loading required package: GenomeInfoDb

## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: DelayedArray

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians
## The following object is masked from 'package:dplyr':
##
##     count

## Loading required package: BiocParallel

##
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following object is masked from 'package:purrr':
##
##     simplify

## The following objects are masked from 'package:base':
##
##     aperm, apply, rowsum
```

```
data("fluidigm", package = "scRNAseq")
se = fluidigm[, fluidigm$Coverage_Type == "High"]
assays(se) = list(normalized_counts =
   round(limma::normalizeQuantiles(assay(se))))
ce = clusterMany(se, clusterFunction = "pam", ks = 5:10, run = TRUE,
  isCount = TRUE, reduceMethod = "var", nFilterDims = c(60, 100, 150))
```

```
## Note: Not all of the methods requested in 'reduceMethod' have been calculated. Will calculate all the
```

```
clusterLabels(ce) = sub("FilterDims", "", clusterLabels(ce))
plotClusters(ce, whichClusters = "workflow", axisLine = -1)
```



We can see that the right most red, left most blue, and green clusters appear to be tight clusters.

## 5.5 examples

### 5.5.1 Flow cytometry and mass cytometry

```r
library("flowCore")
```

```
##
## Attaching package: 'flowCore'

## The following object is masked from 'package:BiocGenerics':
##
##     normalize

## The following object is masked from 'package:tibble':
##
##     view
```

```r
library("flowViz")
fcsB = read.FCS("../data/Bendall_2011.fcs")
slotNames(fcsB)
```

```
## [1] "exprs"       "parameters"  "description"
```

Look at the structure of the fcsB object (hint: the colnames function). How many variables were measured? 41

```r
length(colnames(fcsB))
```

```
## [1] 41
```

Subset the data to look at the first few rows (hint: use Biobase::exprs(fcsB)). How many cells were measured?

```r
nrow(Biobase::exprs(fcsB))
```

```
## [1] 91392
```

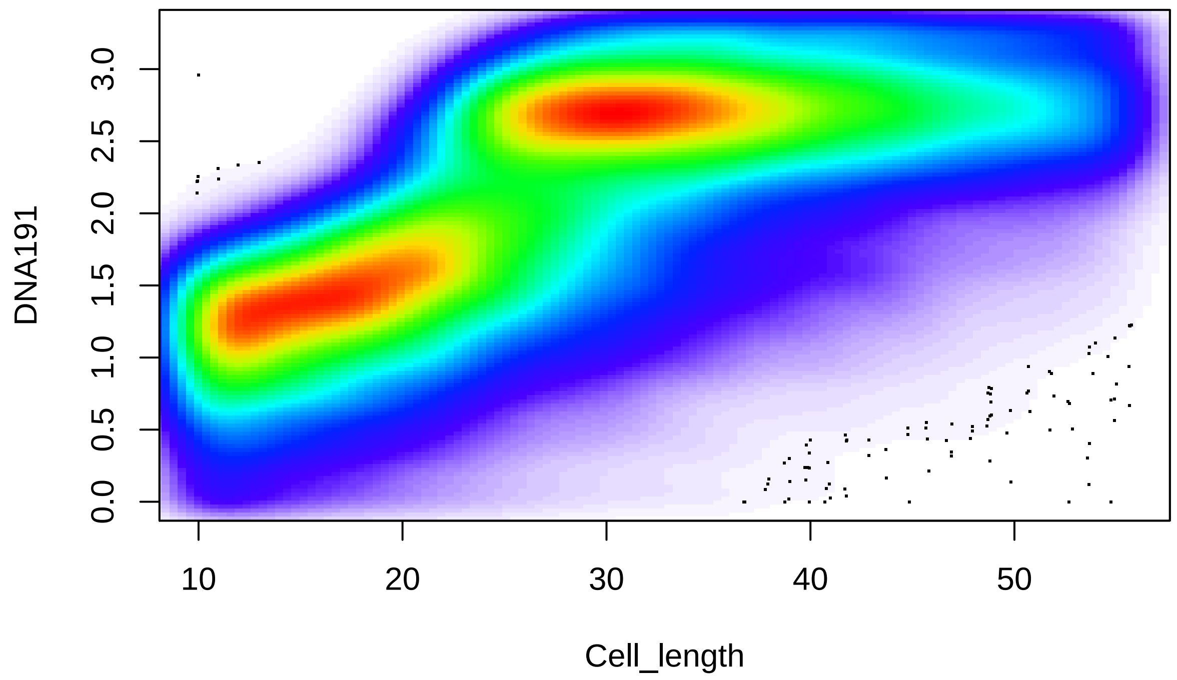
### 5.5.2 Data preprocessing

```r
markersB = readr::read_csv("../data/Bendall_2011_markers.csv")
```

```
## Parsed with column specification:
## cols(
##   isotope = col_character(),
##   marker = col_character()
## )
```

```r
mt = match(markersB$isotope, colnames(fcsB))
stopifnot(!any(is.na(mt)))
colnames(fcsB)[mt] = markersB$marker
```

```r
flowPlot(fcsB, plotParameters = colnames(fcsB)[2:3], logy = TRUE)
```



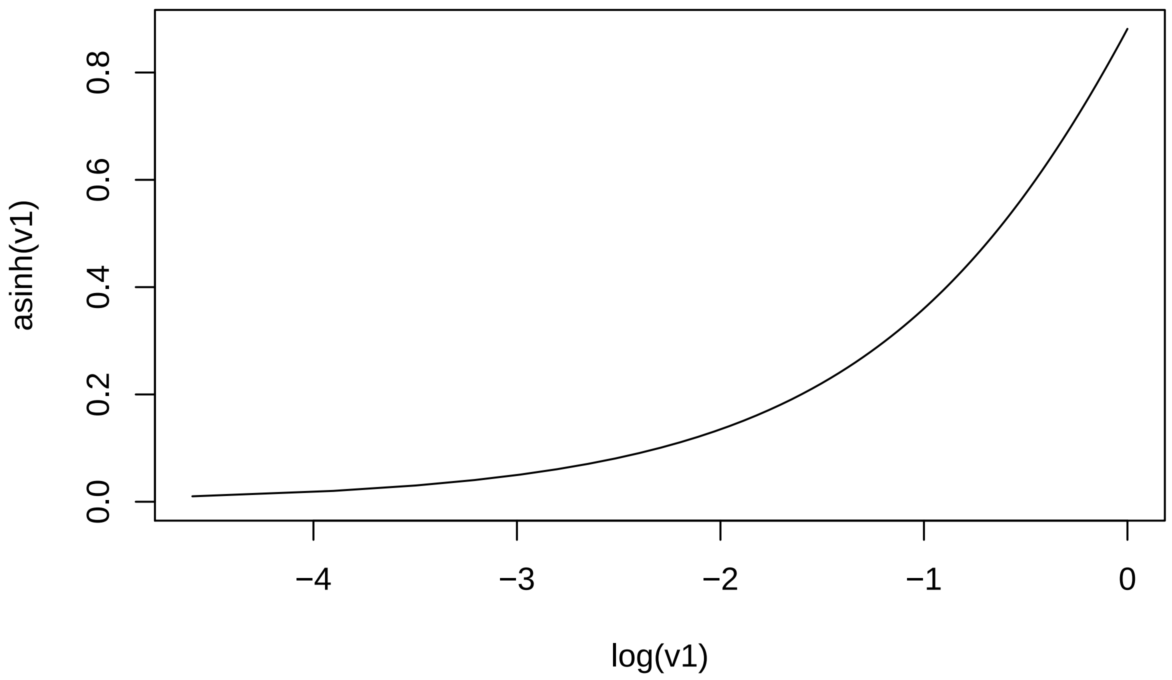
Here we can see clear clustering between cell length and DNA191.

A common data transformation is the hyperbolic arcsin.

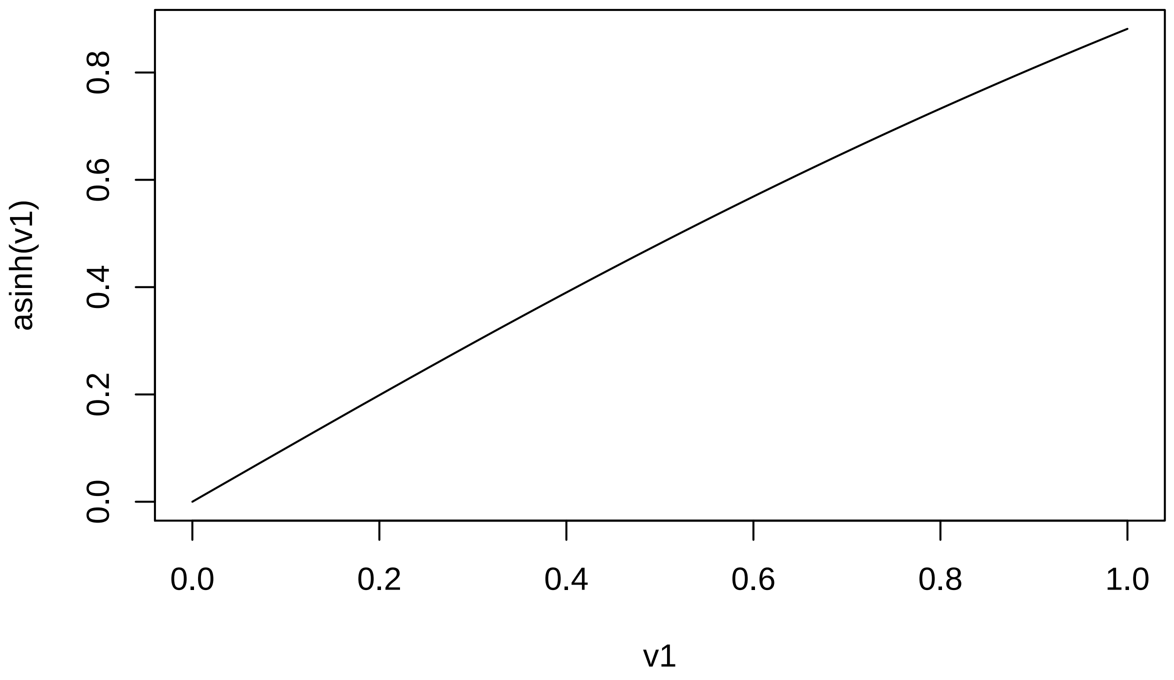
$asinh(x) = \log(x + \sqrt{x^2 + 1})$

We can view this transofrmation for different values of x.

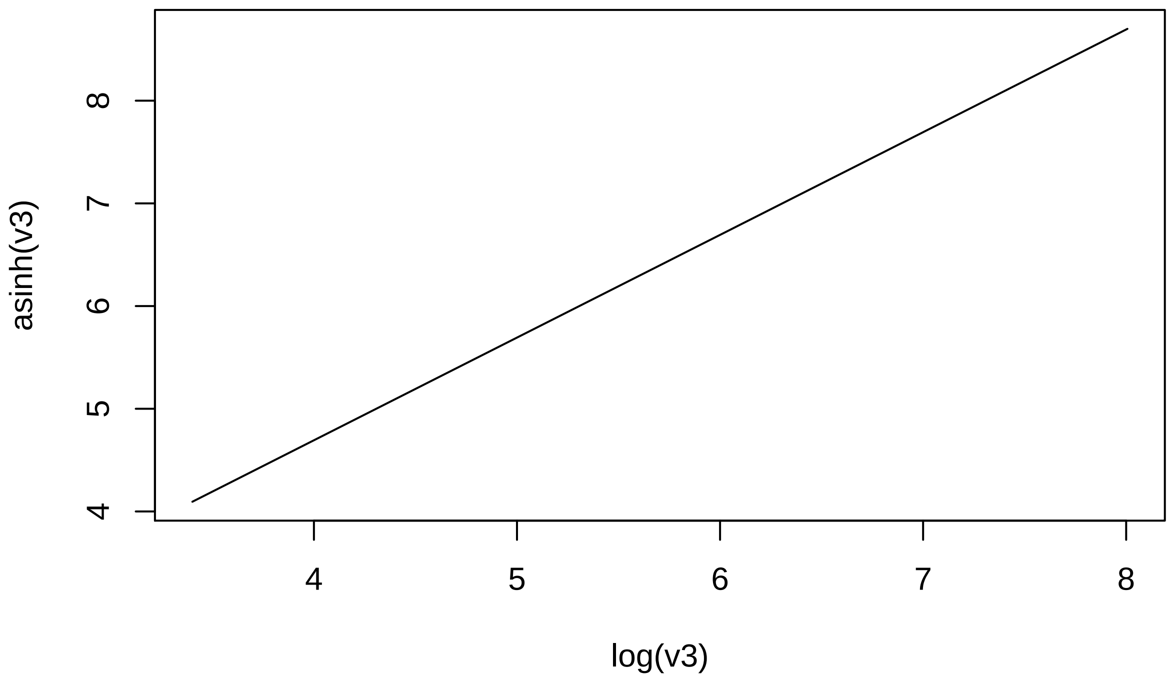
```r
v1 = seq(0, 1, length.out = 100)
plot(log(v1), asinh(v1), type = 'l')
```



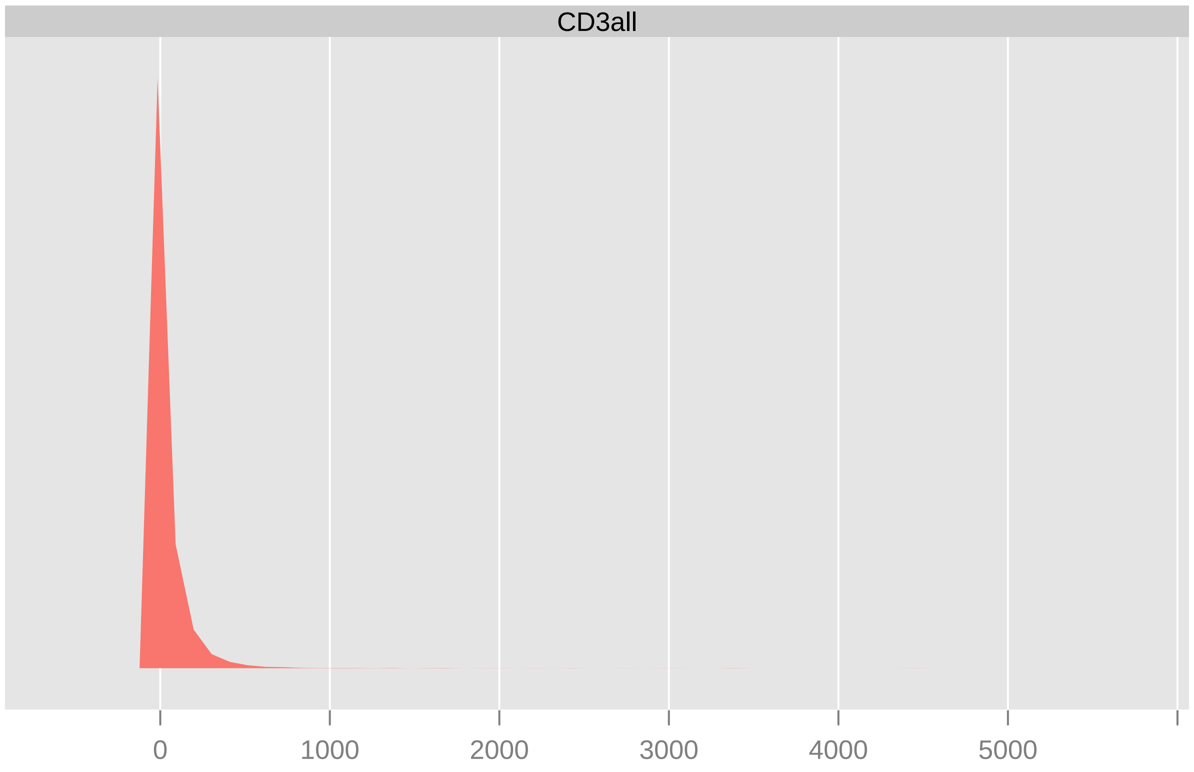
```r
plot(v1, asinh(v1), type = 'l')
```



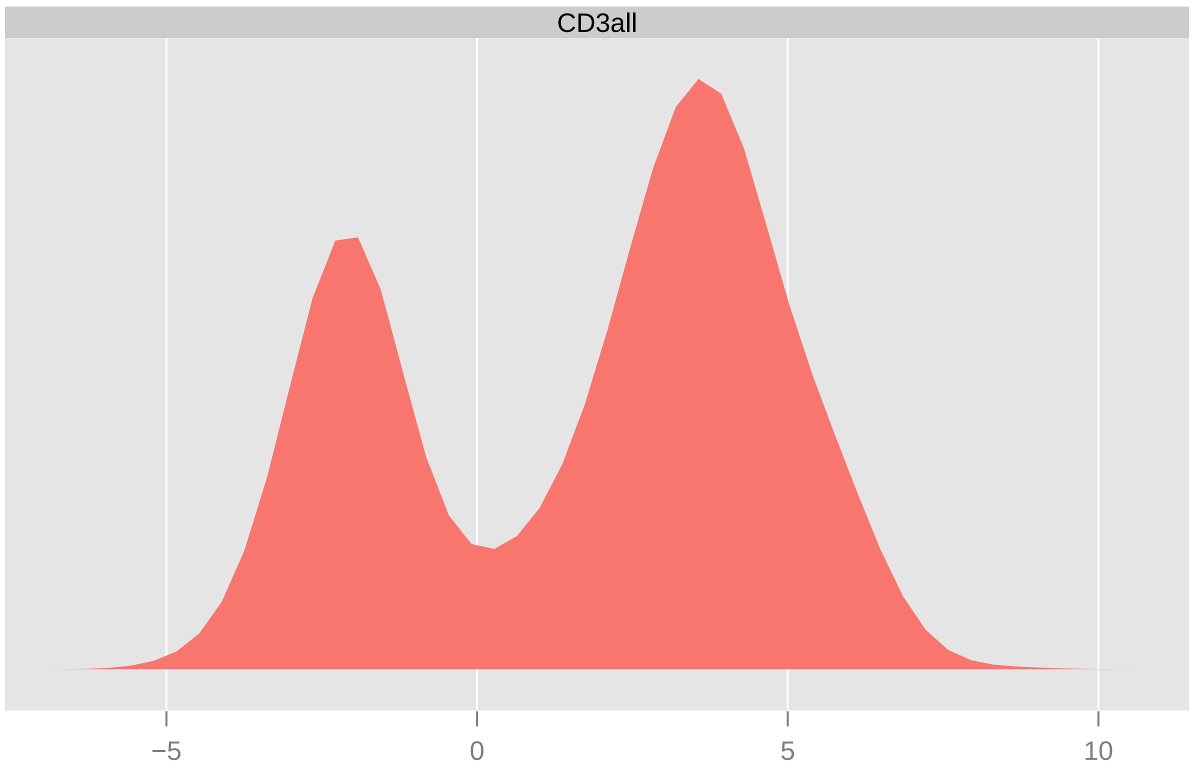
```
v3 = seq(30, 3000, length = 100)
plot(log(v3), asinh(v3), type= 'l')
```



```r
asinhtrsf = arcsinhTransform(a = 0.1, b = 1)
fcsBT = transform(fcsB,
  transformList(colnames(fcsB)[-c(1, 2, 41)], asinhtrsf))
densityplot( ~`CD3all`, fcsB)
```



```
densityplot( ~`CD3all`, fcsBT)
```



How many dimensions does the following code use to split the data into 2 groups using k-means ?

```
kf = kmeansFilter("CD3all" = c("Pop1","Pop2"), filterId="myKmFilter")
fres = flowCore::filter(fcsBT, kf)
summary(fres)
```

```
## Pop1: 33429 of 91392 events (36.58%)
## Pop2: 57963 of 91392 events (63.42%)
```

```
fcsBT1 = flowCore::split(fcsBT, fres, population = "Pop1")
fcsBT2 = flowCore::split(fcsBT, fres, population = "Pop2")
```

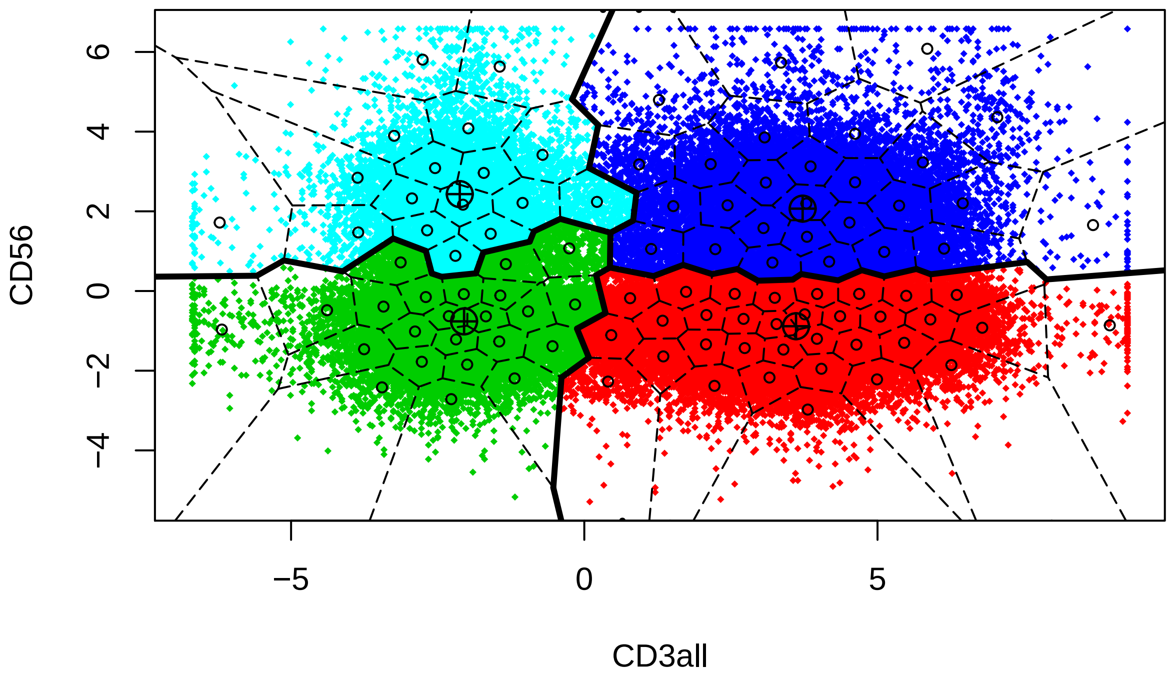
naive projection of the data into the two dimensions spanned by the CD3 and CD56 markers, clusrering was performed using kmeans.

```
library("flowPeaks")
fp = flowPeaks(Biobase::exprs(fcsBT)[, c("CD3all", "CD56")])
```

```
##
## Starting the flow Peaks analysis...
##
##     Task A: compute kmeans...
```

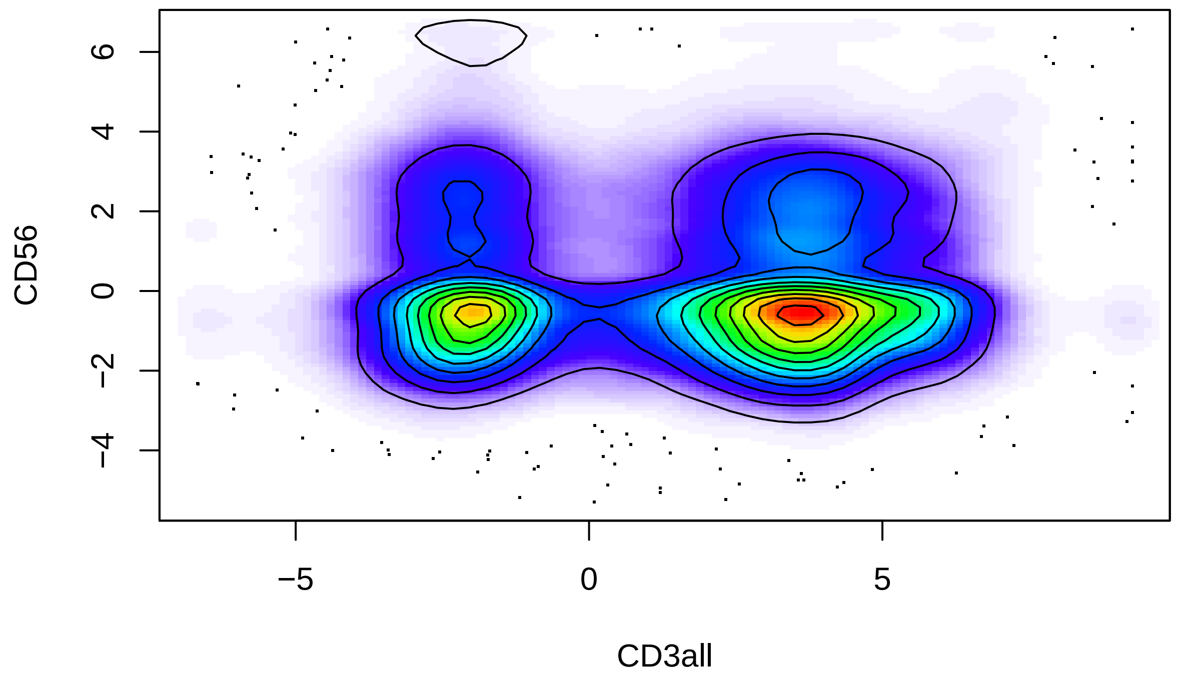
```
##         step 0, set the intial seeds, tot.wss=17597.8
##          step 1, do the rough EM, tot.wss=11900.7 at 0.341 sec
##          step 2, do the fine transfer of Hartigan-Wong Algorithm
##                   tot.wss=11846.3 at 0.694 sec

##          ...finished summarization at 0.71 sec
##
##      Task B: find peaks...

## finished at 0.86 sec
```

```
plot(fp)
```



To avoid overplotting, we can use a 2d kenernal density plot.

```
flowPlot(fcsBT, plotParameters = c("CD3all", "CD56"), logy = FALSE)
contour(fcsBT[, c(40, 19)], add = TRUE)
```



A more recent Bioconductor package, ggcyto, has been designed to enable the plotting of each patient in a diffent facet using ggplot.

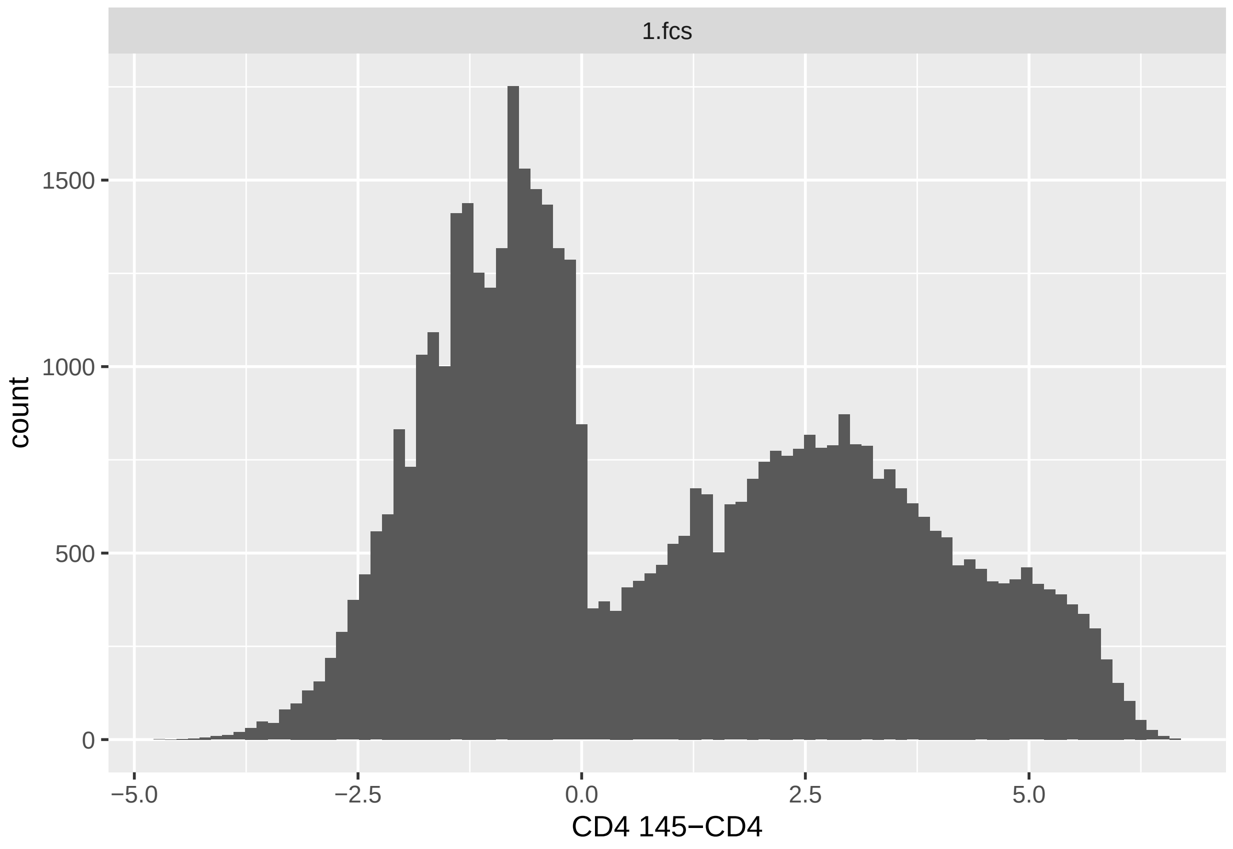
Try comparing the output using this approach to what we did above using the following:

```
library("ggcyto")
```

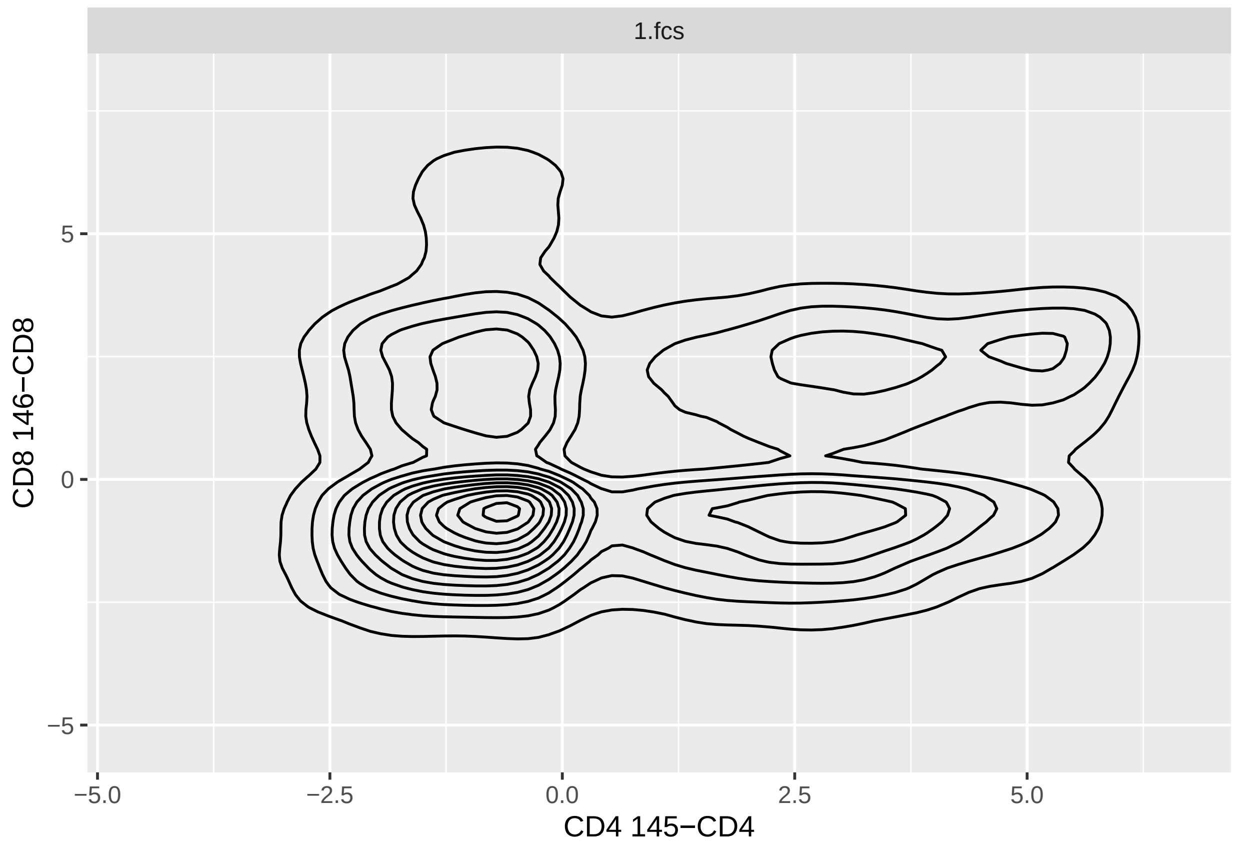
```
## Loading required package: ncdfFlow

## Loading required package: RcppArmadillo

## Loading required package: BH

## Loading required package: flowWorkspace
```

```
library("labeling")
ggcd4cd8=ggcyto(fcsB,aes(x=CD4,y=CD8))
ggcd4=ggcyto(fcsB,aes(x=CD4))
ggcd8=ggcyto(fcsB,aes(x=CD8))
p1=ggcd4+geom_histogram(bins=60)
p1b=ggcd8+geom_histogram(bins=60)
asinhT = arcsinhTransform(a=0,b=1)
transl = transformList(colnames(fcsB)[-c(1,2,41)], asinhT)
fcsBT = transform(fcsB, transl)
p1t=ggcyto(fcsBT,aes(x=CD4))+geom_histogram(bins=90)
```

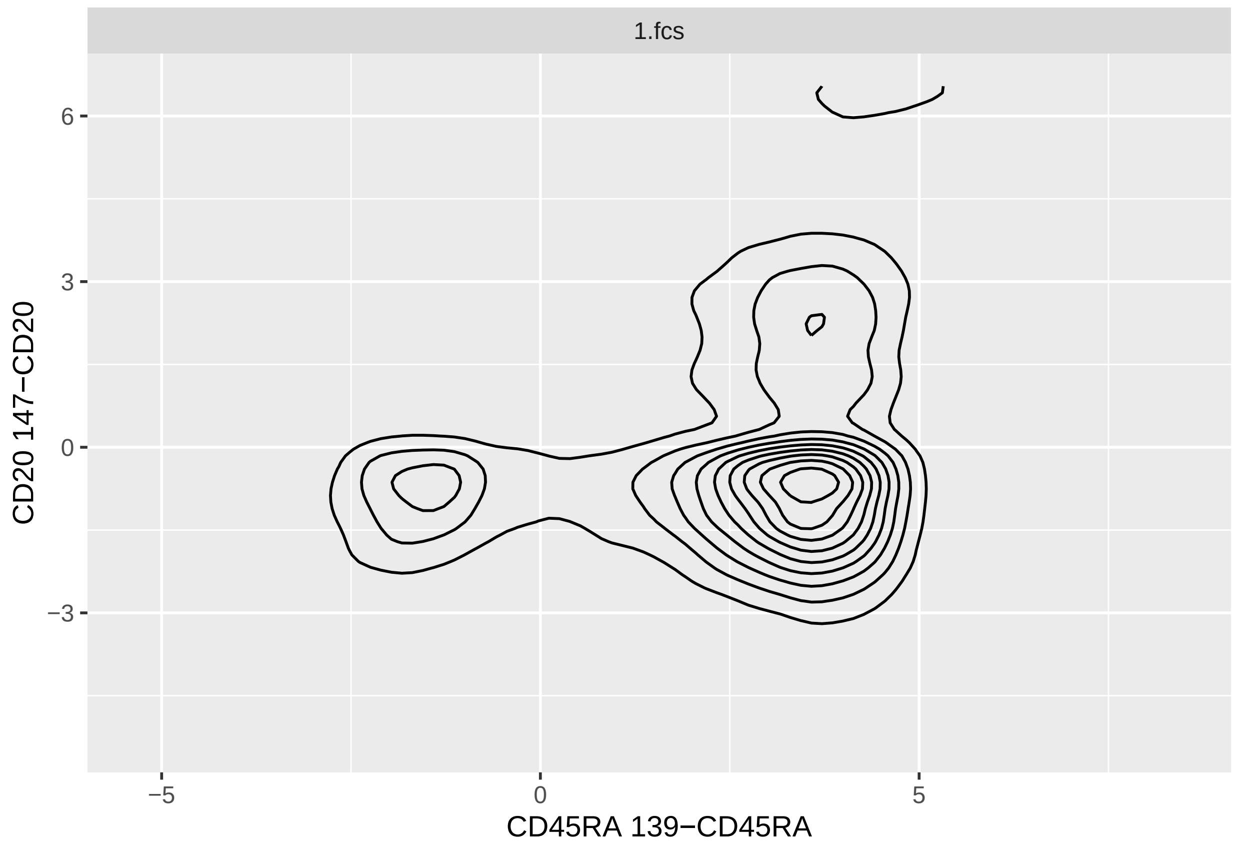
```r
p2t=ggcyto(fcsBT,aes(x=CD4,y=CD8))+geom_density2d(colour="black")
p3t=ggcyto(fcsBT,aes(x=CD45RA,y=CD20))+geom_density2d(colour="black")

p1t
```



```r
p2t
```



```
p3t
```



Output looks similar, but this method appears to be less senistive to clusters.

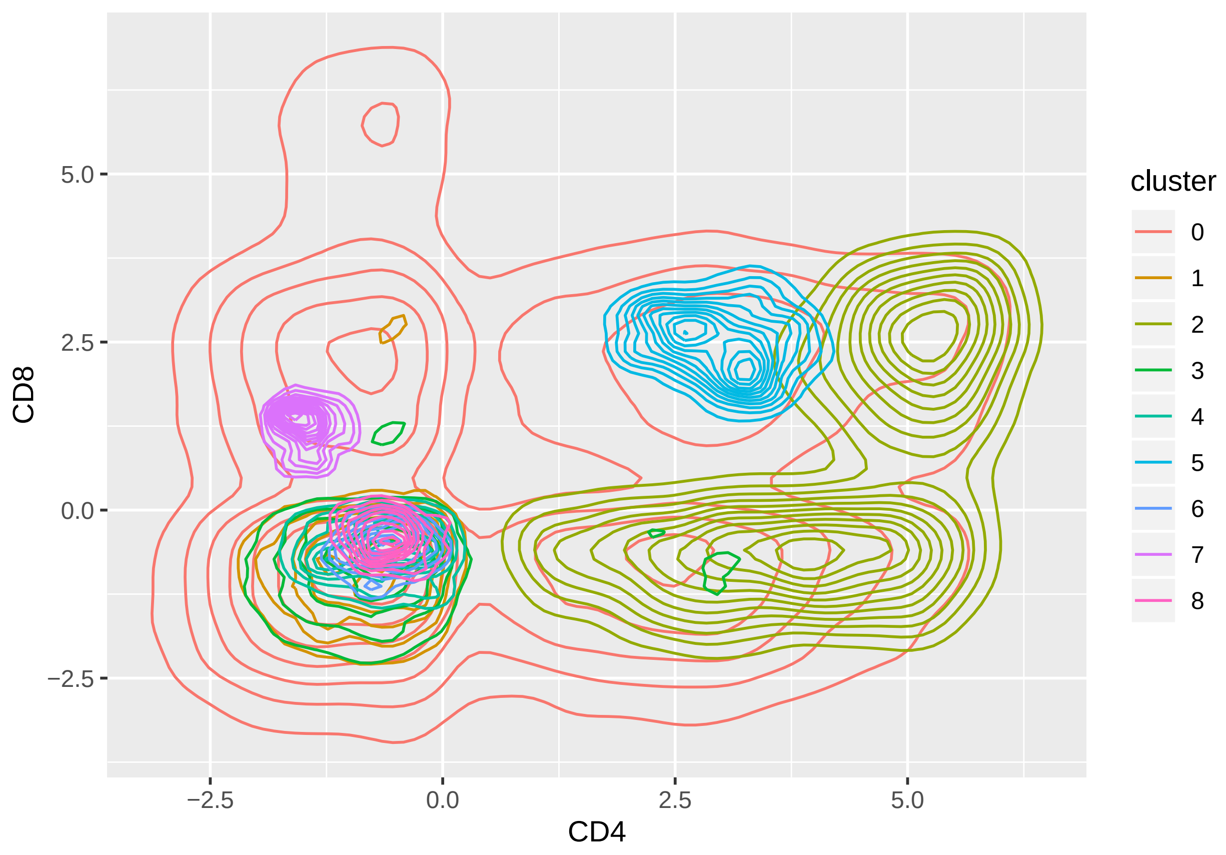
### 5.5.3 Density-based clustering

Used when we have only a few markers but a large amount of data (cells).

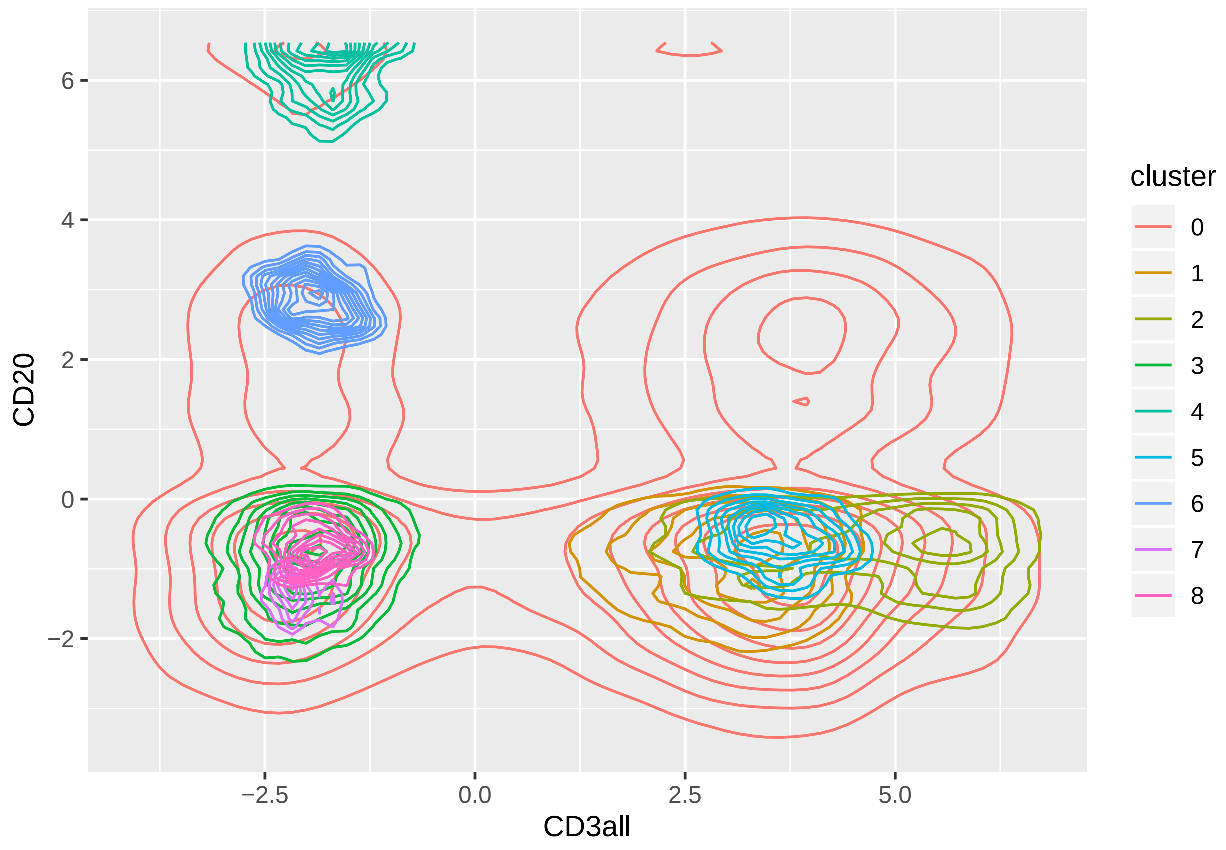
```
library("dbscan")
mc5 = Biobase::exprs(fcsBT)[, c(15,16,19,40,33)]
res5 = dbscan::dbscan(mc5, eps = 0.65, minPts = 30)
mc5df = data.frame(mc5, cluster = as.factor(res5$cluster))
table(mc5df$cluster)
```

```
##
##     0     1     2     3     4     5     6     7     8
## 75954  4031  5450  5310   259   257    63    25    43
```

```
ggplot(mc5df, aes(x=CD4,    y=CD8,  col=cluster))+geom_density2d()
```



```r
ggplot(mc5df, aes(x=CD3all, y=CD20, col=cluster))+geom_density2d()
```



Try increasing the dimension to 6 by adding one CD marker-variables from the input data.

Then vary eps, and try to find four clusters such that at least two of them have more than 100 points.

Repeat this will 7 CD marker-variables, what do you notice?

```
mc6 = Biobase::exprs(fcsBT)[, c(15, 16, 19, 33, 25, 40)]
res = dbscan::dbscan(mc6, eps = 0.65, minPts = 20)
mc6df = data.frame(mc6, cluster = as.factor(res$cluster))
table(mc6df$cluster)
```

```
##
##     0     1     2     3     4     5     6
## 91068    34    61    20    67   121    21
```

```
mc7 = Biobase::exprs(fcsBT)[, c(11, 15, 16, 19, 25, 33, 40)]
res = dbscan::dbscan(mc7, eps = 0.95, minPts = 20)
mc7df = data.frame(mc7, cluster = as.factor(res$cluster))
table(mc7df$cluster)
```

```
##
##     0     1     2     3     4     5     6     7     8     9    10
## 90249    21   102   445   158   119    19   224    17    20    18
```

In general, the higher the dimension, the larger we need to set eps in order to find large enough clusters.

How does density-based clustering work?

Uses something called the density-connectedness criterion. That is, it looks at small neighborhood spheres of radius $\epsilon$ to see if points are connected.

The rest of this sounds very abstract and mathematical…

## 5.6 Hierarchical clustering

Bottom-up approach, where similar observations and subclasses are assembled iteratively.

Note that the order of the labels does not matter within sibling pairs and the horizontal distances are usually meaningless, while the vertical distances do encode some information.

There is also a top-down apporach that takes all the objects and splits them sequentially according to a chosesn criterion.

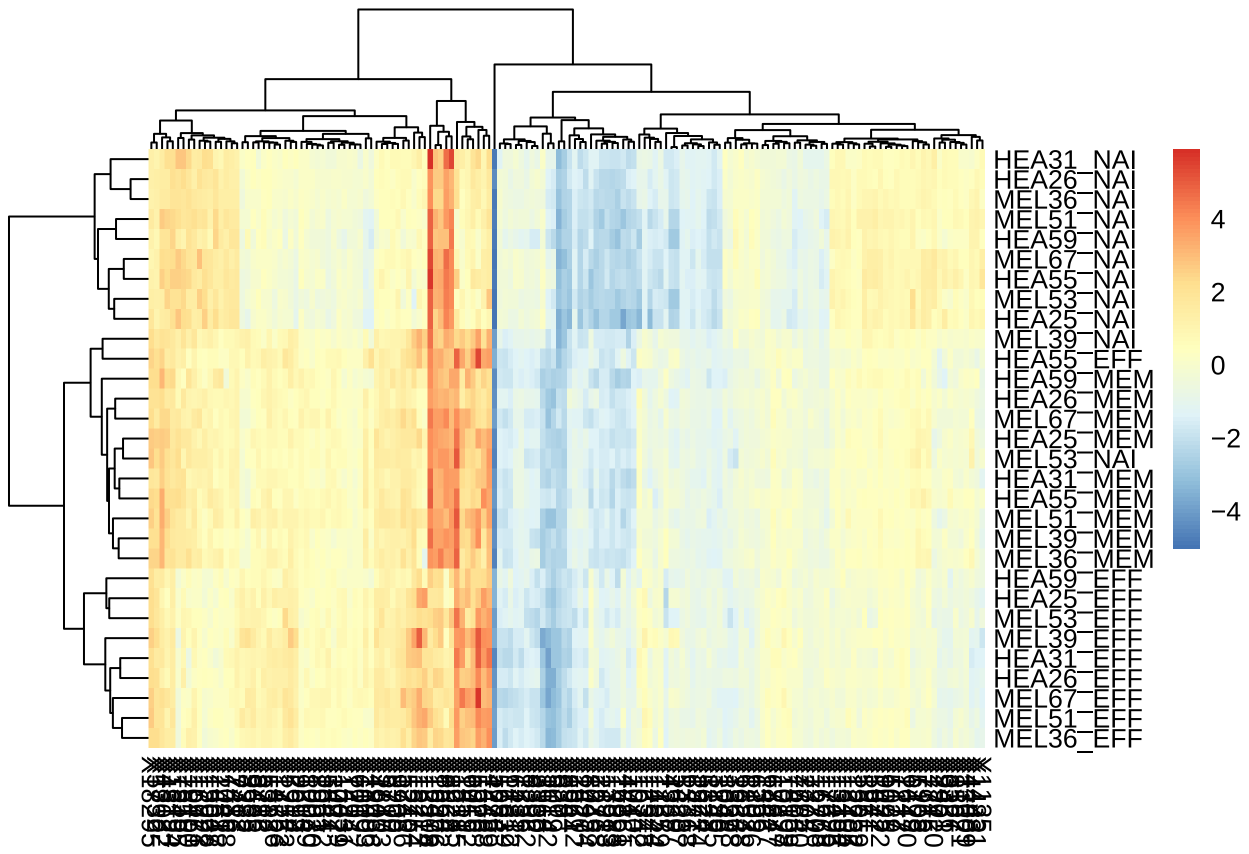
### 5.6.1 How to compute (dis)similarities between aggregated clusters?

We will need more than just the distances between all pairs of individual objects. We also need a way to calculate distances between the aggergates.

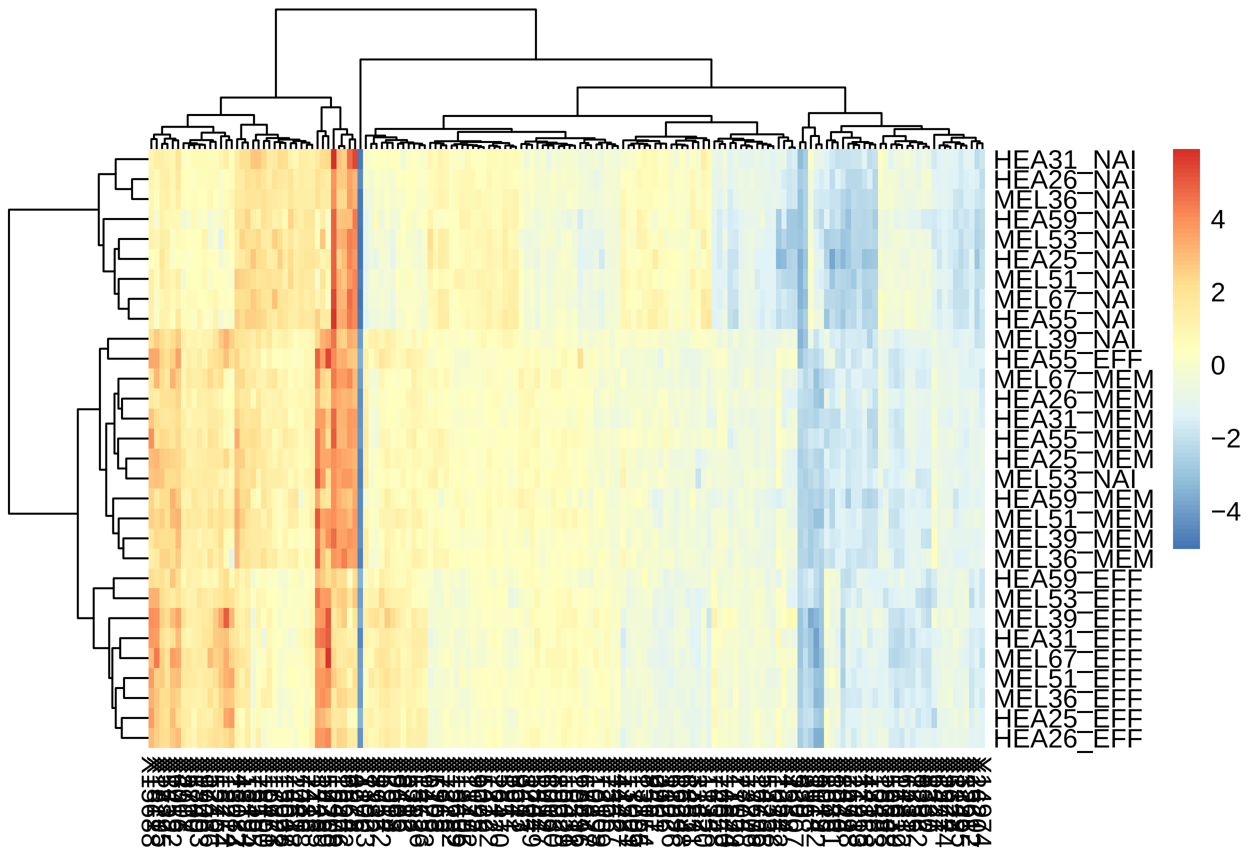
Hierarchical clustering for cell populations The Morder data are gene expression measurements for 156 genes on T cells of 3 types (naïve, effector, memory) from 10 patients (Holmes et al. 2005).

Using the pheatmap package, make two simple heatmaps, without dendogram or reordering, for Euclidean and Manhattan distances of these data.

```r
library(pheatmap)

load(here("Data", "Morder.RData"))

pheatmap(Morder)
```



```
pheatmap(Morder,clustering_distance_rows="manhattan",clustering_distance_cols="manhattan")
```



## 5.7 Validating and choosing the number of clusters

Need a way to validate our clusters with a number.

We define the within-groups sum of squared distances (WSS)

$WSS_k = \sum_{i=1}^{k} \sum_{x_i \in C_i} d^2(x_i, \bar{x}_i)$

where:

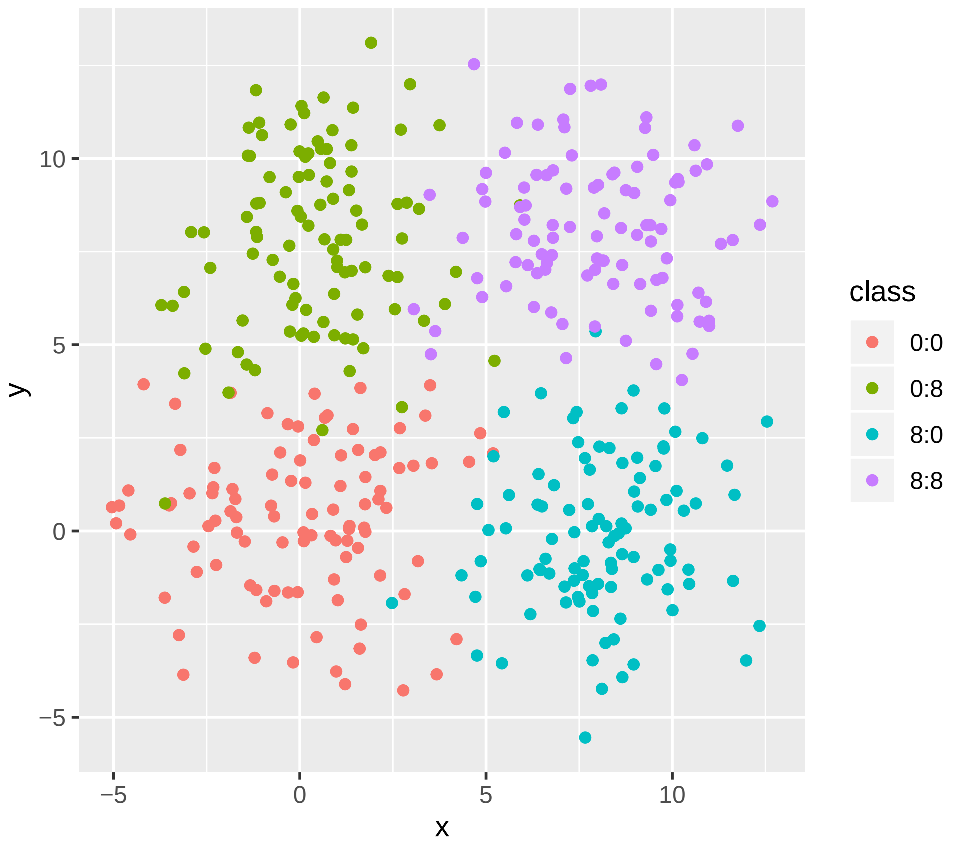
- k is the number of clusters
- $C_i$ is the set of objects in the $i^{th}$ cluster
- $\bar{x}_i$ is the center of mass (average point) of the $i^{th}$ cluster.

This is a good starting point, but we can always minizine this value by taking the number of clusters to be equal to the number of data points.

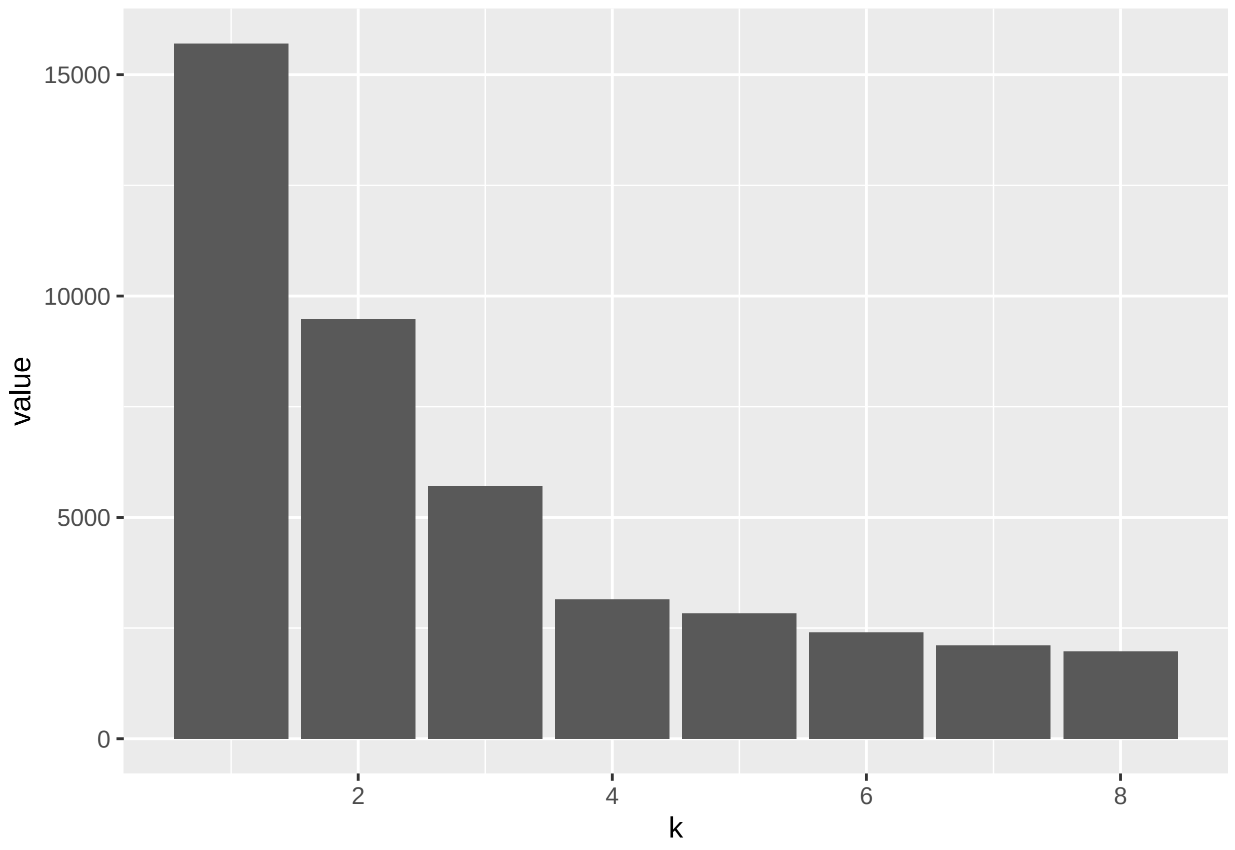
```
library("dplyr")
simdat = lapply(c(0, 8), function(mx) {
  lapply(c(0,8), function(my) {
    tibble(x = rnorm(100, mean = mx, sd = 2),
           y = rnorm(100, mean = my, sd = 2),
           class = paste(mx, my, sep = ":"))
   }) %>% bind_rows
}) %>% bind_rows
simdat
```

```
## # A tibble: 400 x 3
##          x      y class
##      <dbl>  <dbl> <chr>
##   1  2.17   2.11  0:0
##   2  0.674  3.04  0:0
##   3  1.76   1.45  0:0
##   4  0.376  2.44  0:0
##   5 -0.681 -1.61  0:0
##   6  2.67   1.69  0:0
##   7  3.50   3.91  0:0
##   8 -1.81   1.13  0:0
##   9  2.78  -4.28  0:0
##  10 -2.25  -0.912 0:0
## # ... with 390 more rows
```

```r
simdatxy = simdat[, c("x", "y")] # without class label

ggplot(simdat, aes(x = x, y = y, col = class)) + geom_point() +
  coord_fixed()
```



```r
wss = tibble(k = 1:8, value = NA_real_)
wss$value[1] = sum(scale(simdatxy, scale = FALSE)^2)
for (i in 2:nrow(wss)) {
  km  = kmeans(simdatxy, centers = wss$k[i])
  wss$value[i] = sum(km$withinss)
}
ggplot(wss, aes(x = k, y = value)) + geom_col()
```

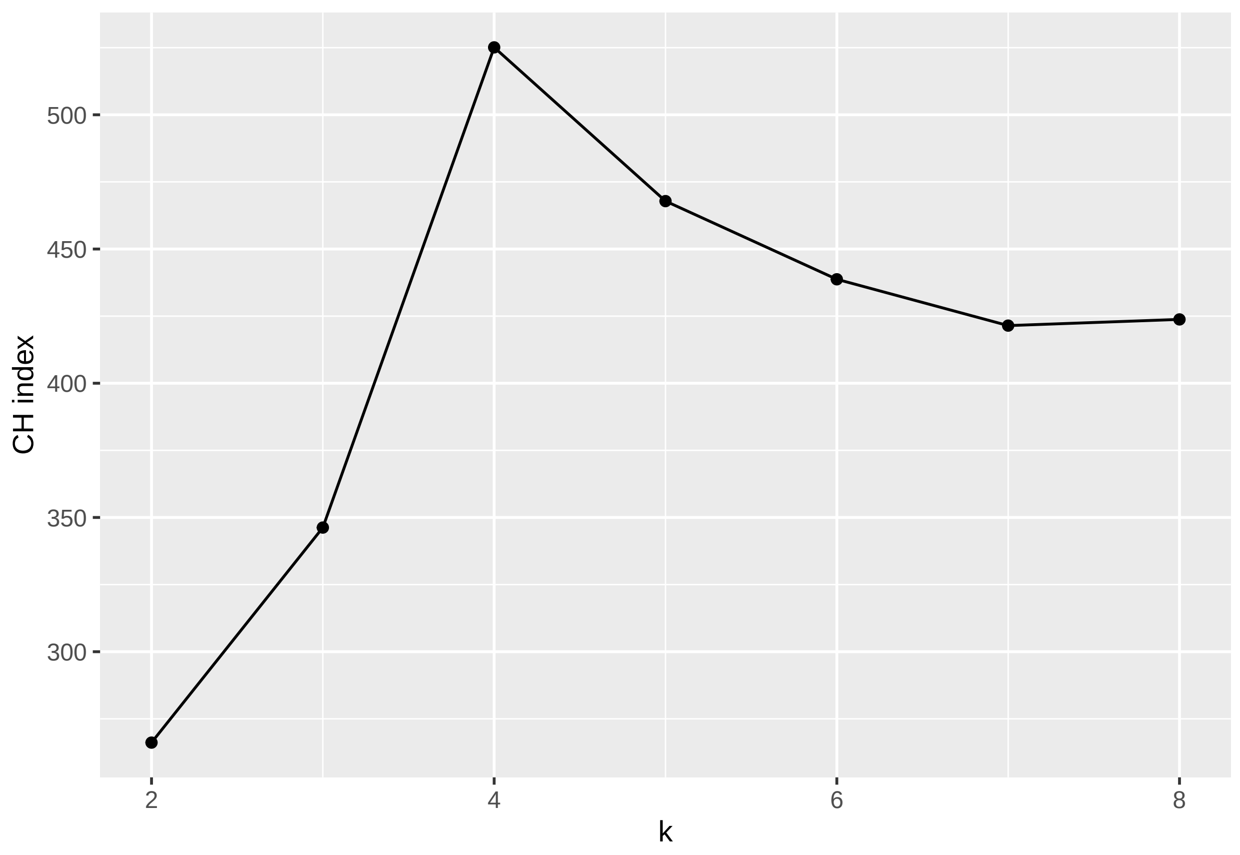


We can also use the calinski-Harabasz index, it is the ratio between Between Sum of Squares and Within Sum of squares errors distances.

```r
library("fpc")
```

```
##
## Attaching package: 'fpc'

## The following object is masked from 'package:dbscan':
##
##     dbscan
```

```r
library("cluster")
CH = tibble(
  k = 2:8,
  value = sapply(k, function(i) {
    p = pam(simdatxy, i)
    calinhara(simdatxy, p$cluster)
  })
)
ggplot(CH, aes(x = k, y = value)) + geom_line() + geom_point() +
  ylab("CH index")
```



### 5.7.1 Using the gap statistic

The basic idea is to take some trasformation of the within-sum-of-squares (typically the log) and compare it to the averages from simulated data with less structure.

The general steps for computing the gap statistic are:

- Cluster the data with k clusters and compute $WSS_k$ for various choices of k
- Generate B plausible reference data sets, using Monte Carlo sampling from a homogeneous distribution and redo Step 1 above for these new simulated data.
- Compute:

$gap(k) = \bar{l}_k - \log WWS_k$ with

$\bar{l}_k = \frac{1}{B}\sum_{b=1}^{B} \log W^*_{k,b}$

- Then we can use the standard deviation, defined as:

$sd_k^2 = \frac{1}{B-1}\sum_{b=1}^{B}(\log W^*_{k,b} - \bar{l}_k)^2$

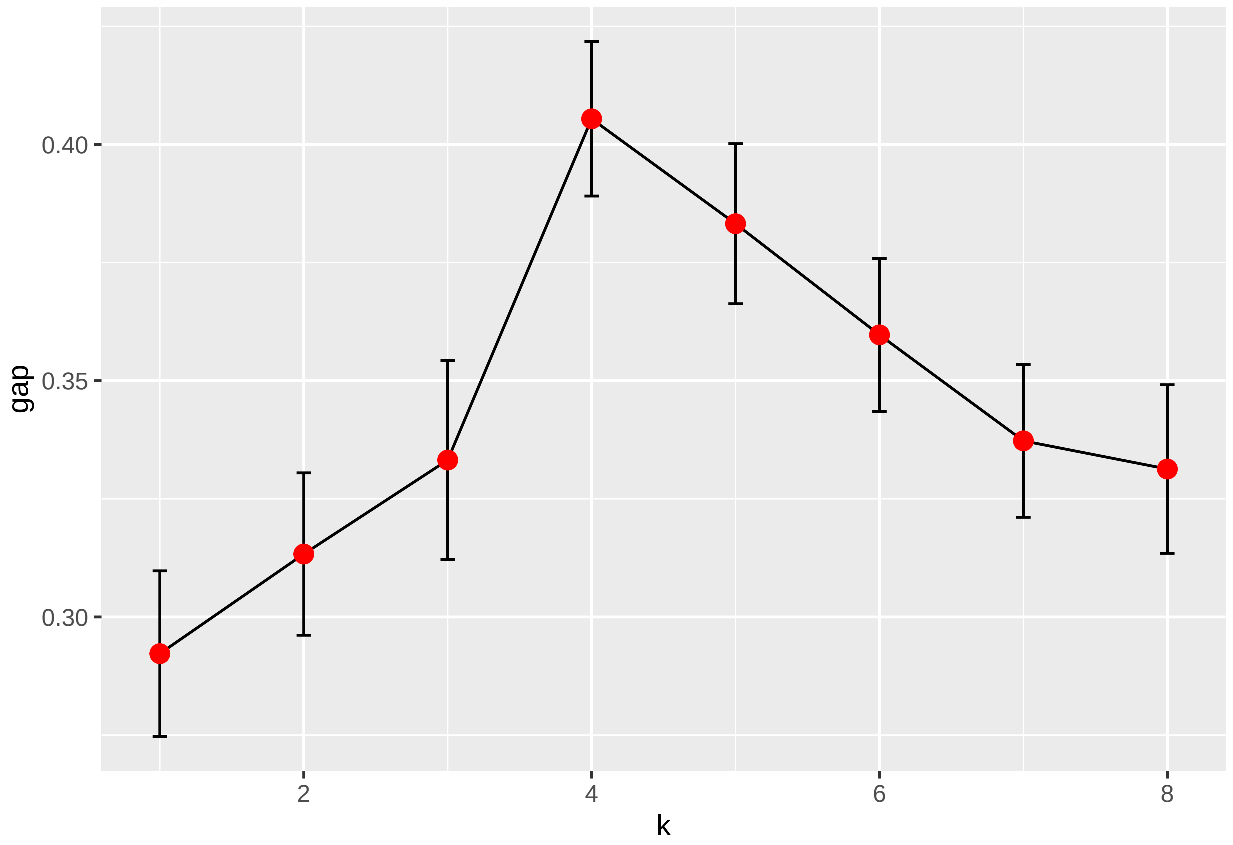
in order to determine the best value for k.

The packages cluster and clusterCrit provide implementations.

Make a function that plots the gap statistic as in Figure 5.27.

Show the output for the simdat example dataset clustered with the pam function.

```r
library("cluster")
library("ggplot2")
pamfun = function(x, k)
  list(cluster = pam(x, k, cluster.only = TRUE))

gss = clusGap(simdatxy, FUN = pamfun, K.max = 8, B = 50,
              verbose = FALSE)
plot_gap = function(x) {
  gstab = data.frame(x$Tab, k = seq_len(nrow(x$Tab)))
  ggplot(gstab, aes(k, gap)) + geom_line() +
    geom_errorbar(aes(ymax = gap + SE.sim,
                      ymin = gap - SE.sim), width=0.1) +
    geom_point(size = 3, col=  "red")
}
plot_gap(gss)
```



```r
library("Hiiragi2013")
```

```
## Loading required package: affy

## Loading required package: boot

##
## Attaching package: 'boot'
## The following object is masked from 'package:lattice':
##
##     melanoma
```

```
## Loading required package: clue
```

```
## Loading required package: genefilter
```

```
##
## Attaching package: 'genefilter'
```

```
## The following objects are masked from 'package:matrixStats':
##
##     rowSds, rowVars
```

```
## The following object is masked from 'package:readr':
##
##     spec
```

```
## Loading required package: geneplotter
```

```
## Loading required package: annotate
```

```
## Loading required package: AnnotationDbi
```

```
##
## Attaching package: 'AnnotationDbi'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## Loading required package: XML
```

```
##
## Attaching package: 'annotate'
```

```
## The following object is masked from 'package:flowCore':
##
##     journal
```

```
## Loading required package: gplots
```

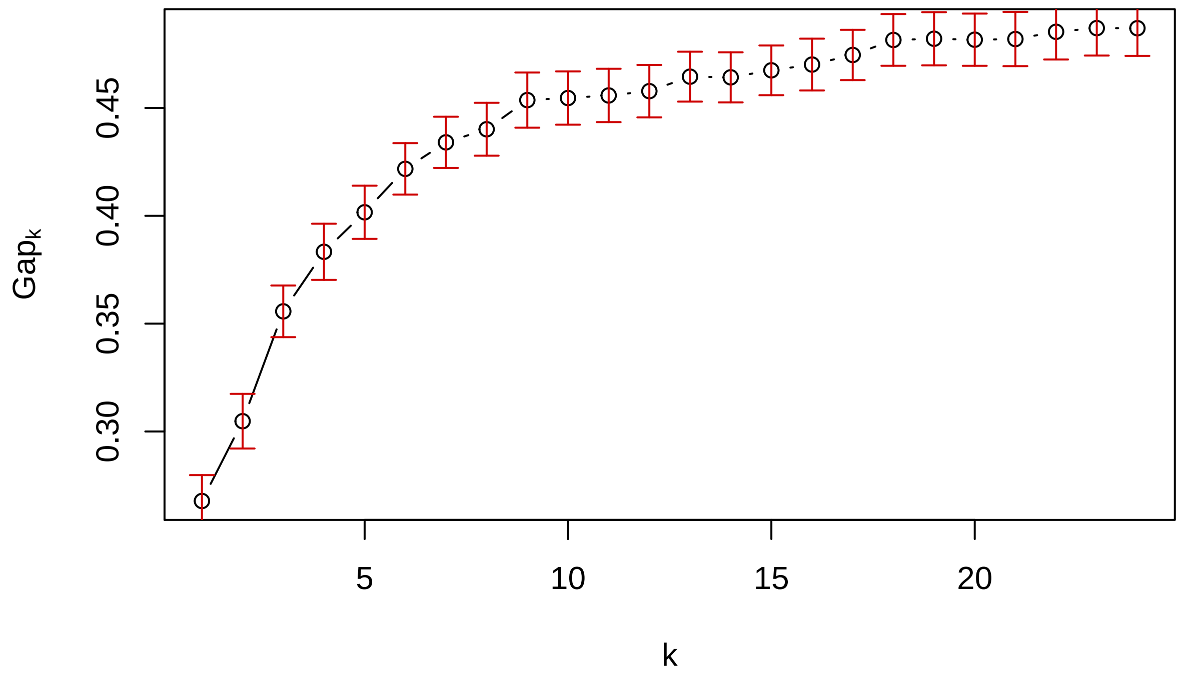
```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:IRanges':
##
##     space
## The following object is masked from 'package:S4Vectors':
##
##     space

## The following object is masked from 'package:stats':
##
##     lowess

## Loading required package: gtools

##
## Attaching package: 'gtools'

## The following objects are masked from 'package:boot':
##
##     inv.logit, logit

## The following object is masked from 'package:permute':
##
##     permute

## Loading required package: KEGGREST

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:AnnotationDbi':
##
##     select

## The following object is masked from 'package:genefilter':
##
##     area

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: mouse4302.db

## Loading required package: org.Mm.eg.db

##

##

## Loading required package: RColorBrewer
## Loading required package: xtable

##
## Attaching package: 'xtable'

## The following object is masked from 'package:fpc':
##
##     xtable
```

```
data("x")

selFeats = order(rowVars(Biobase::exprs(x)), decreasing = TRUE)[1:50]
embmat = t(Biobase::exprs(x)[selFeats, ])
embgap = clusGap(embmat, FUN = pamfun, K.max = 24
                 , verbose = FALSE)
k1 = maxSE(embgap$Tab[, "gap"], embgap$Tab[, "SE.sim"])
k2 = maxSE(embgap$Tab[, "gap"], embgap$Tab[, "SE.sim"],
           method = "Tibs2001SEmax")
c(k1, k2)
```

```
## [1] 9 7
```

```
plot(embgap, main = "")
```



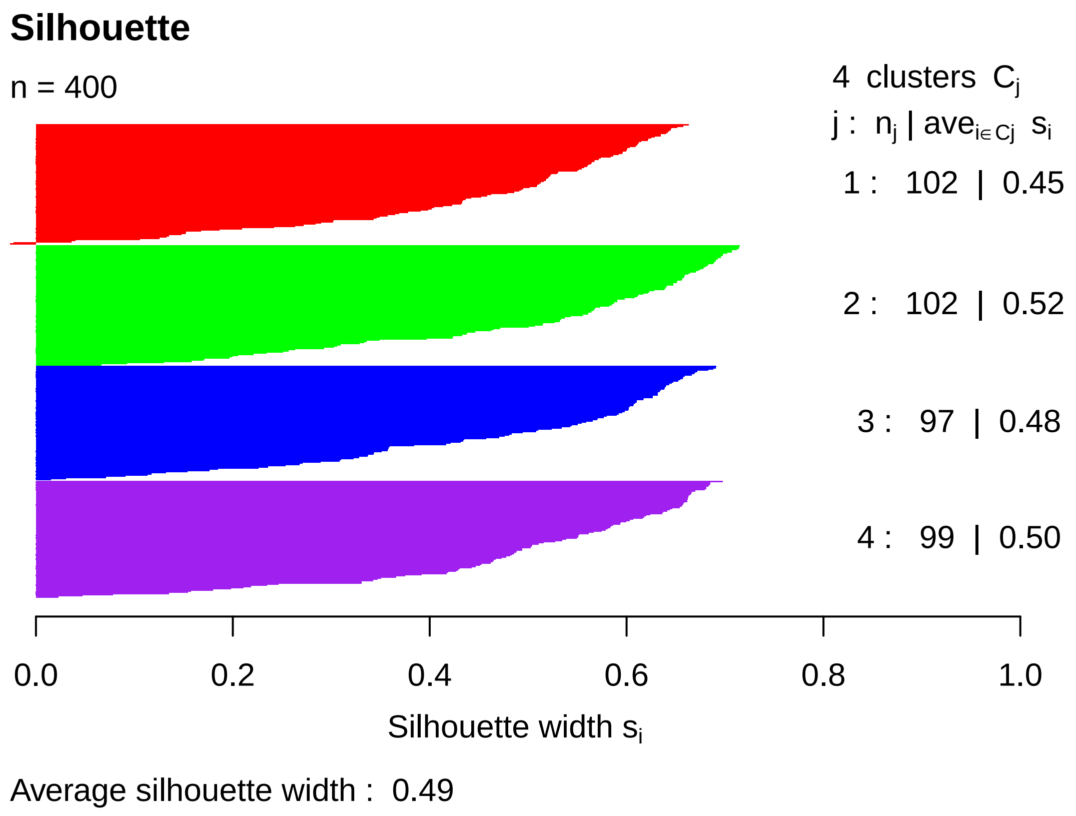
```
cl = pamfun(embmat, k = k1)$cluster
table(pData(x)[names(cl), "sampleGroup"], cl)
```

```
##                    cl
##                      1  2  3  4  5  6  7  8  9
##   E3.25             23 11  1  1  0  0  0  0  0
##   E3.25 (FGF4-KO)    0  0  1 16  0  0  0  0  0
##   E3.5 (EPI)         2  1  0  0  0  8  0  0  0
##   E3.5 (FGF4-KO)     0  0  8  0  0  0  0  0  0
##   E3.5 (PE)          0  0  0  0  9  2  0  0  0
##   E4.5 (EPI)         0  0  0  0  0  0  0  4  0
##   E4.5 (FGF4-KO)     0  0  0  0  0  0  0  0 10
##   E4.5 (PE)          0  0  0  0  0  0  4  0  0
```

## Exercises

### 5.1

```
library("cluster")
pam4 = pam(simdatxy, 4)
sil = silhouette(pam4, 4)
plot(sil, col=c("red","green","blue","purple"), main="Silhouette")
```

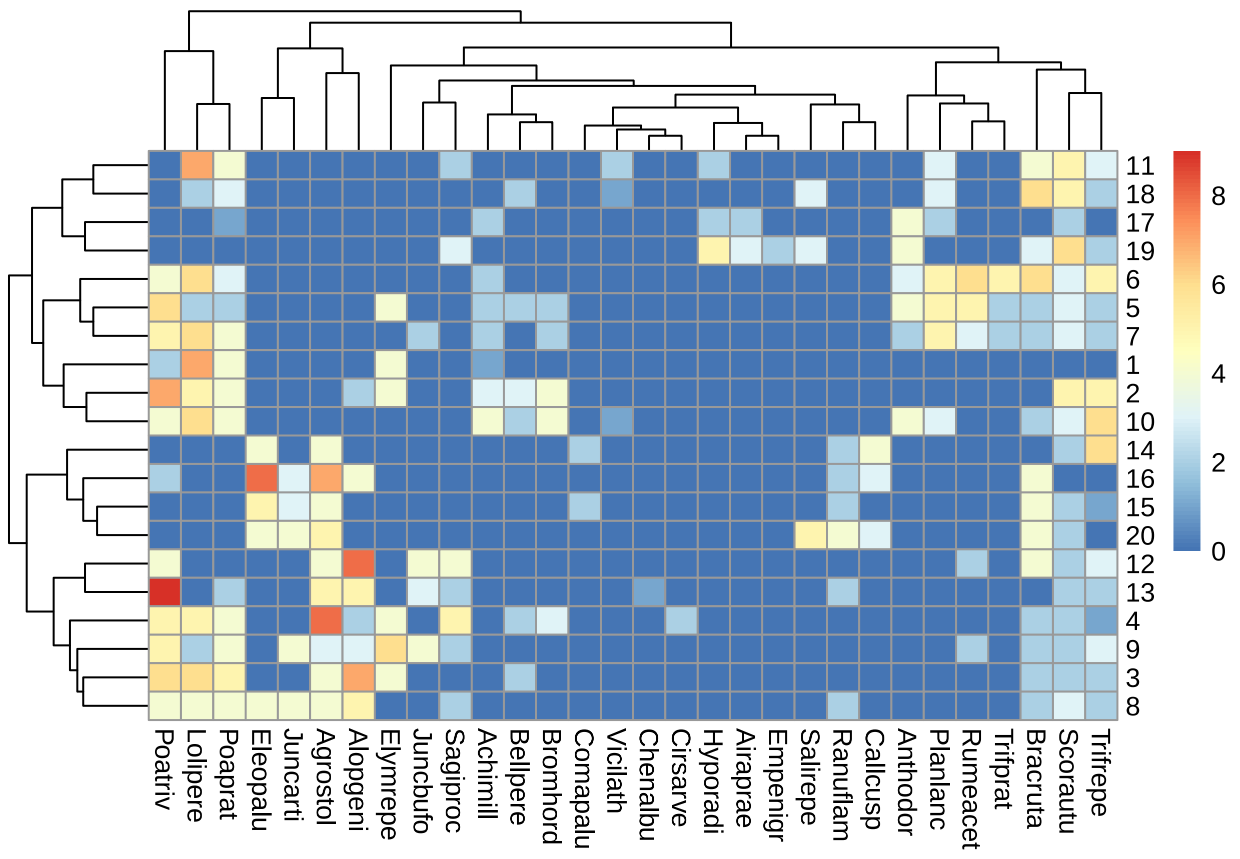


I belive 4 gives the best silhouette index.

c.) Not sure about this

### 5.2

```r
data(dune)

pheatmap(dune)
```

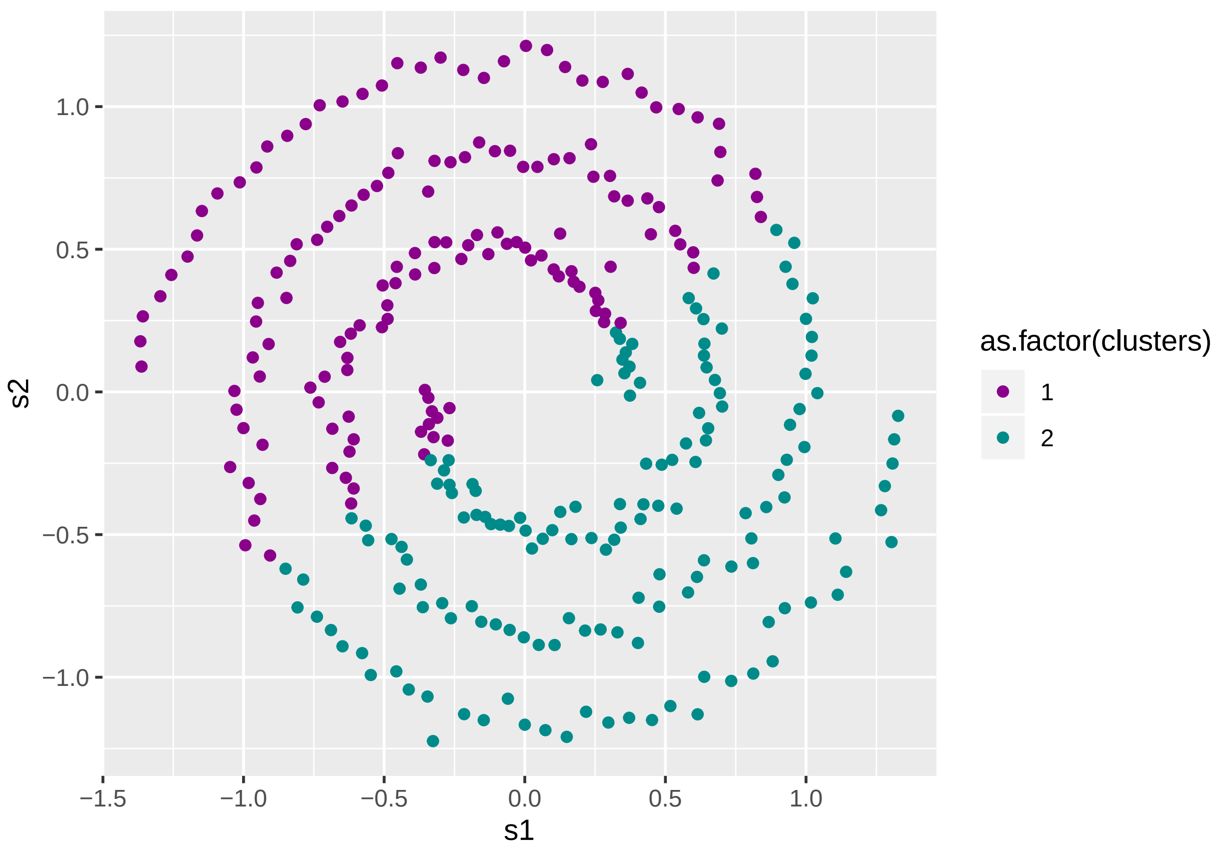


### 5.3

```r
library(kernlab)
```

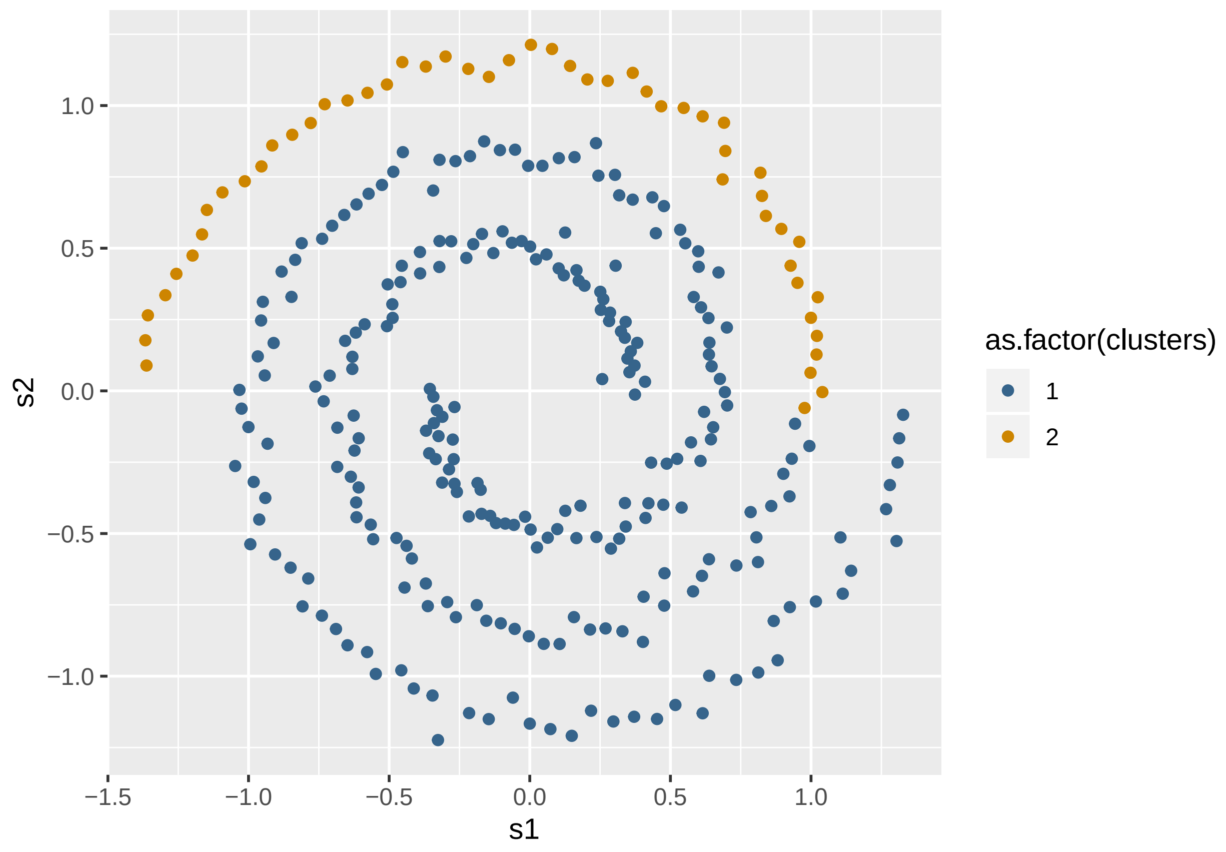
```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:DelayedArray':
##
##     type
## The following object is masked from 'package:permute':
##
##     how

## The following object is masked from 'package:purrr':
##
##     cross

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

```r
library(tidyverse)
data(spirals)


kmeans = kmeans(spirals,2)

clusters = kmeans$cluster

data = tibble(s1=spirals[,1],s2=spirals[,2],clusters)

data %>% ggplot(aes(x=s1,y=s2,color=as.factor(clusters))) +
  geom_point() +
  scale_color_manual(values=c("darkmagenta","cyan4"))
```

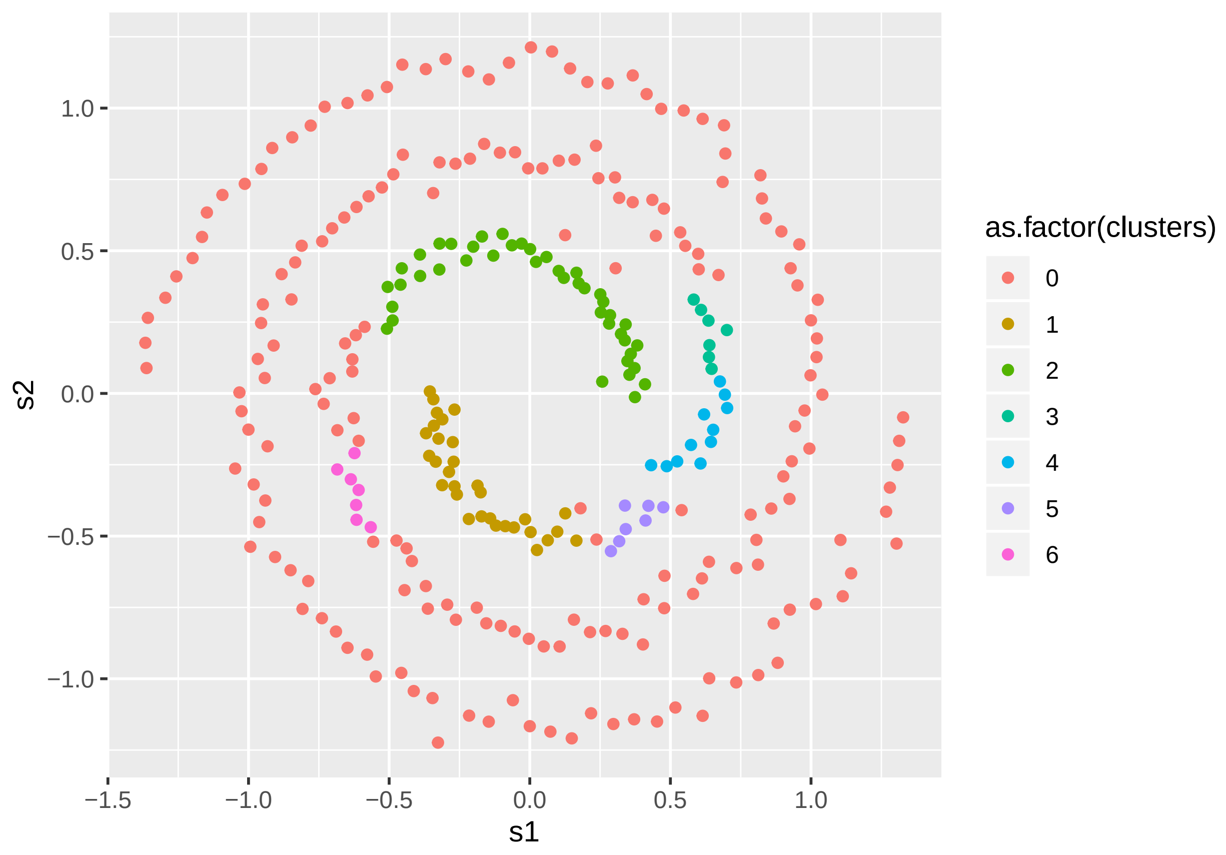


Let's try specc

```r
spec = specc(spirals,2)

clusters = spec@.Data

data = tibble(s1=spirals[,1],s2=spirals[,2],clusters)

data %>% ggplot(aes(x=s1,y=s2,color=as.factor(clusters))) +
  geom_point() +
  scale_color_manual(values=c("steelblue4","orange3"))
```



```r
library(dbscan)
db = dbscan(spirals,.1)

clusters = db$cluster

data = tibble(s1=spirals[,1],s2=spirals[,2],clusters)

data %>% ggplot(aes(x=s1,y=s2,color=as.factor(clusters))) +
  geom_point()
```



### 5.4

Not sure why this data is clustured.

### 5.5

```
library(dada2)
```

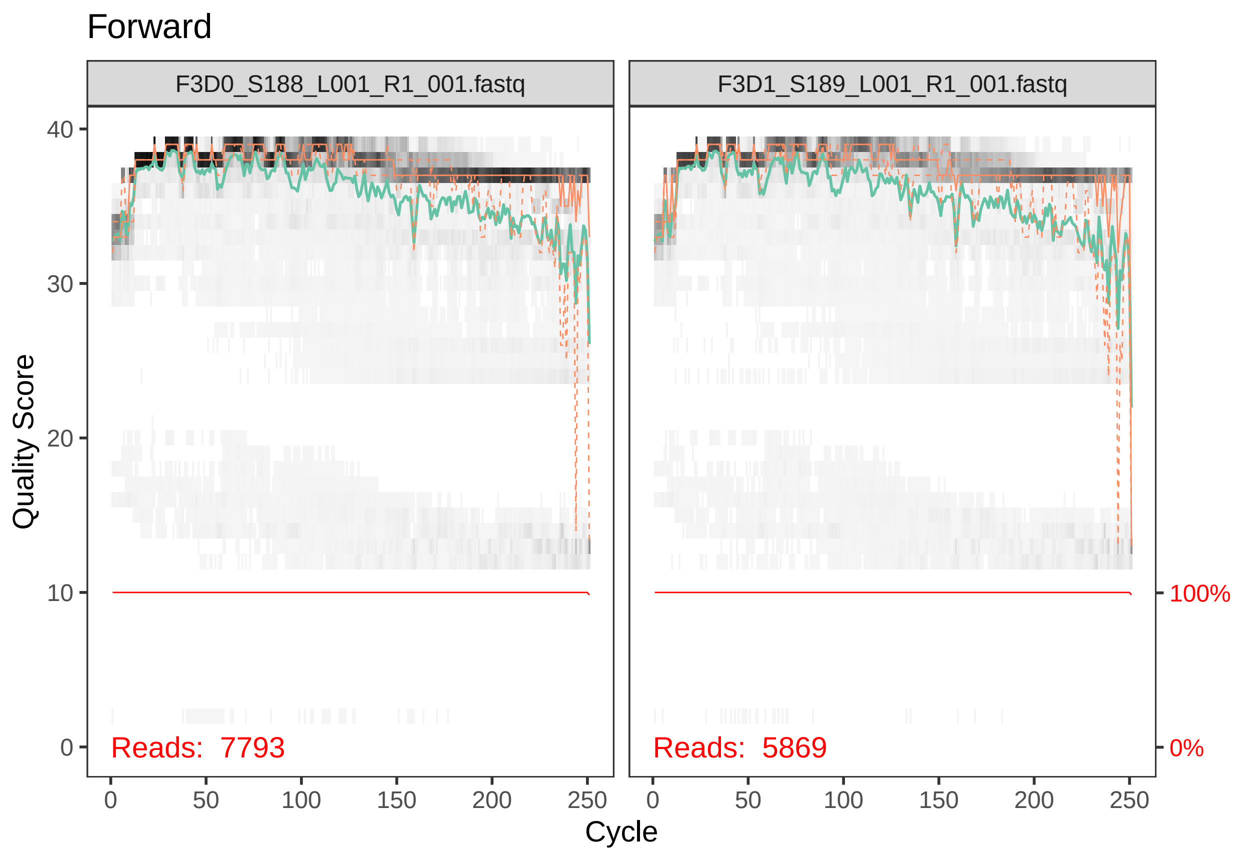
```
## Loading required package: Rcpp
```

```
base_dir = "../data"
miseq_path = file.path(base_dir, "MiSeq_SOP")
filt_path = file.path(miseq_path, "filtered")
fnFs = sort(list.files(miseq_path, pattern="_R1_001.fastq"))
fnRs = sort(list.files(miseq_path, pattern="_R2_001.fastq"))
sampleNames = sapply(strsplit(fnFs, "_"), `[`, 1)
if (!file_test("-d", filt_path)) dir.create(filt_path)
filtFs = file.path(filt_path, paste0(sampleNames, "_F_filt.fastq.gz"))
filtRs = file.path(filt_path, paste0(sampleNames, "_R_filt.fastq.gz"))
fnFs = file.path(miseq_path, fnFs)
fnRs = file.path(miseq_path, fnRs)
print(length(fnFs))
```

```
## [1] 20
```

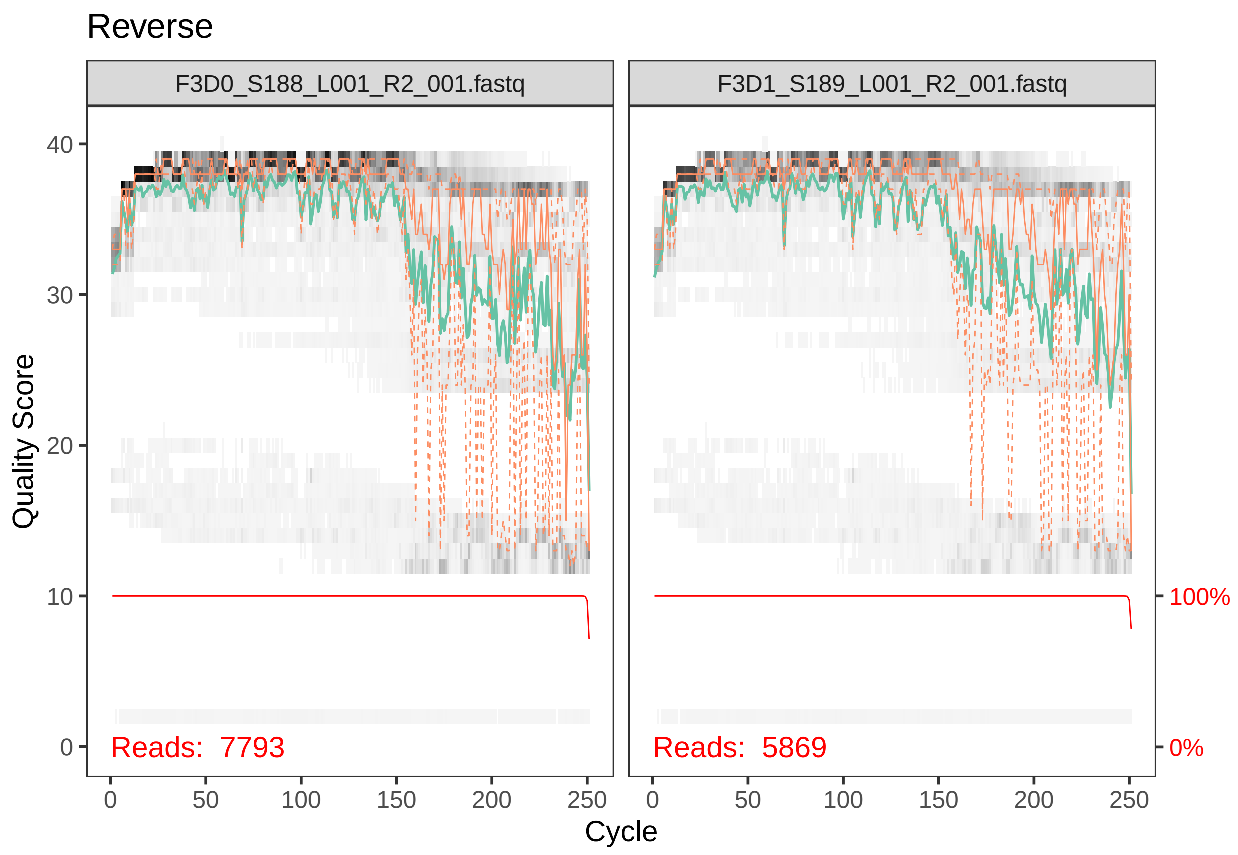
```r
plotQualityProfile(fnFs[1:2]) + ggtitle("Forward")
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```



```r
plotQualityProfile(fnRs[1:2]) + ggtitle("Reverse")
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```


Reverse
F3D0_S188_L001_R2_001.fastq
F3D1_S189_L001_R2_001.fastq
Quality Score
40
30
20
10
0
Reads: 7793
Reads: 5869
100%
0%
0
50
100
150
200
250
Cycle