Model 3 Clustering

Junaira I. Ibrahim

2022-12-15

Import Packages

```
library(dplyr)        # for data manipulation
library(ggplot2)      # for awesome plotting
library(stringr)      # for string functionality
library(gridExtra)    # for manipulating the grid
library(tidyverse)    # for filtering
library(cluster)      # for general clustering algorithms
library(factoextra)   # for visualizing cluster results
library(readr)
library(mclust)       # for fitting clustering algorithms
library(bestNormalize)
```

Importing the dataset\ The dataset used in this model is imported from `radiomics data`. It has 197 observations and 431 variables.

```
datard <- read_csv("radiomics_completedata.csv")
```

```
## Rows: 197 Columns: 431
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (1): Institution
## dbl (430): Failure.binary, Failure, Entropy_cooc.W.ADC, GLNU_align.H.PET, Mi...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dim(datard)
```

```
## [1] 197 431
```

To check for the normality, the `shapiro.test` is used.\ *Checking for Normality*

```
datard1 = datard%>%select_if(is.numeric)
datadl1 = lapply(datard1[,-1], shapiro.test)
r = lapply(datadl1, function(x)x$p.value) #Extracting p-value only
s=unlist(r)    #to convert a list to vector
sum(s[s>0.05])
```

```
## [1] 0.1350113
```

```
r$Entropy_cooc.W.ADC
```

```
## [1] 0.1350113
```

Based on the normality test, there is only one variable that is normally distributed (*Entropy_cooc.W.ADC*), the rest is non normal. Hence, we will try to normalize the data using `orderNorm()` function.

Normalizing the data

```
datard_norm = datard[,c(3,5:length(names(datard)))]
datard_norm = apply(datard_norm,2,orderNorm)
datard_norm = lapply(datard_norm, function(x) x$x.t)
datard_norm = datard_norm%>%as.data.frame()
```

Test again using shapiro-wilk's test.

```
datadl2 = lapply(datard_norm, shapiro.test)
r2 = lapply(datadl2, function(x) x$p.value)
s2 = unlist(r2)
sum(s2>0.05)
```

```
## [1] 428
```

Based on the results, the rest of the variables is now normally distributed.

Substituing the normalized values into the original data, we have

```
r3 = select(datard, c("Failure.binary",  "Entropy_cooc.W.ADC"))
datard_n = cbind(r3,datard_norm)
```

1. K_MEANS CLUSTERING

The **k-means algorithm** is perhaps the most often used clustering method. The k-means algorithm involves assigning each of the n examples to one of the k clusters, where k is a number that has been defined ahead of time. The goal is to minimize the differences within each cluster and maximize the differences between clusters. The basic idea behind k-means clustering is constructing clusters so that the total within-cluster variation is minimized.

In K-means, the commonly used rule of thumb for k is $k = \sqrt{n/2}$, where
n is the number of observations to cluster. However, here we start at $k = 2$ and also a good rule for the number of random starts to apply is 10-20.

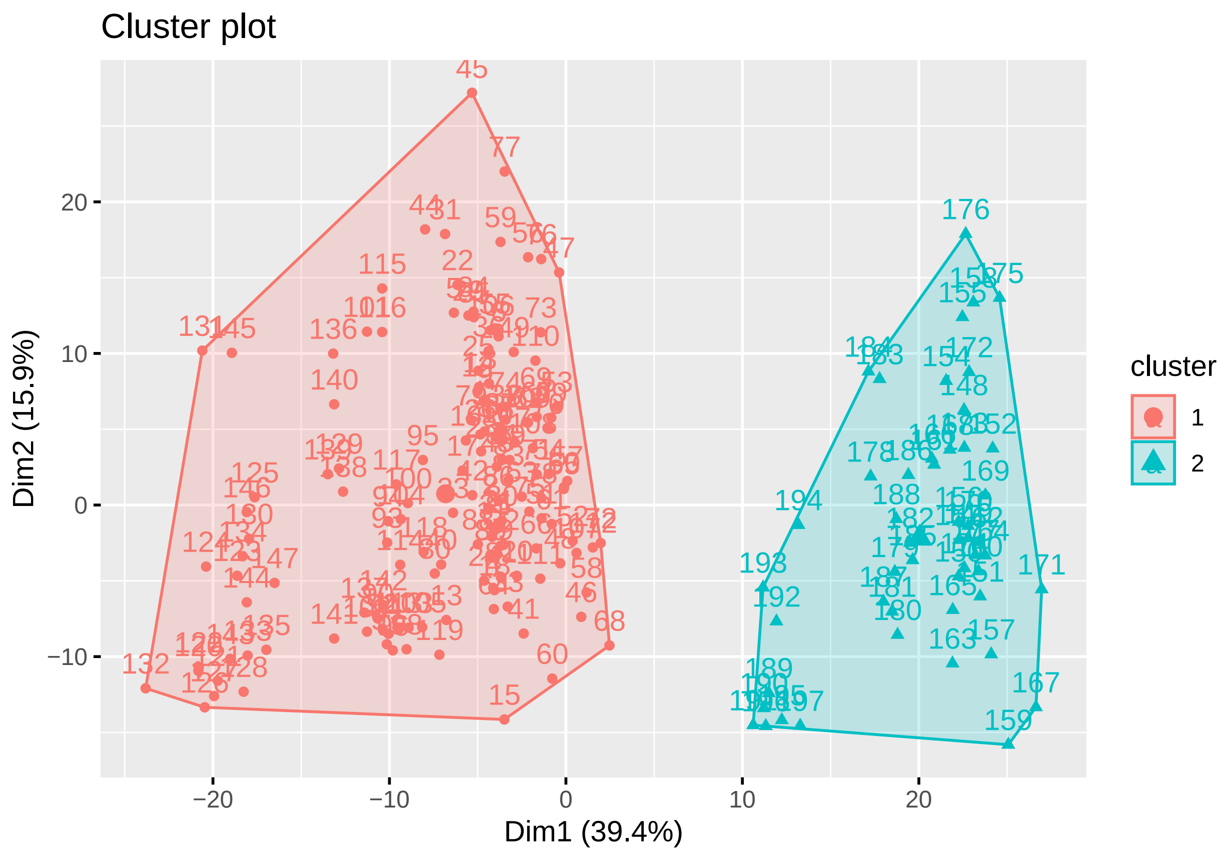
```
#Start at 2 clusters
km2 <- kmeans(datard_n, centers = 2, nstart = 20)
str(km2)
```

```
## List of 9
##  $ cluster     : int [1:197] 1 1 1 1 1 1 1 1 1 1 ...
##  $ centers     : num [1:2, 1:430] 0.3537 0.3 12.2615 12.329 -0.0279 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "1" "2"
##   .. ..$ : chr [1:430] "Failure.binary" "Entropy_cooc.W.ADC" "Failure" "GLNU_align.H.PET" ...
##  $ totss       : num 84022
##  $ withinss    : num [1:2] 42691 13415
##  $ tot.withinss: num 56106
##  $ betweenss   : num 27916
##  $ size        : int [1:2] 147 50
##  $ iter        : int 1
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

From the results, we have 2 K-means clusters of sizes 50, 147.

To see the plot of `km2` we use the function `fviz_cluster()`. `fviz_cluster()` provides ggplot, the observations are represented by points in the plot.

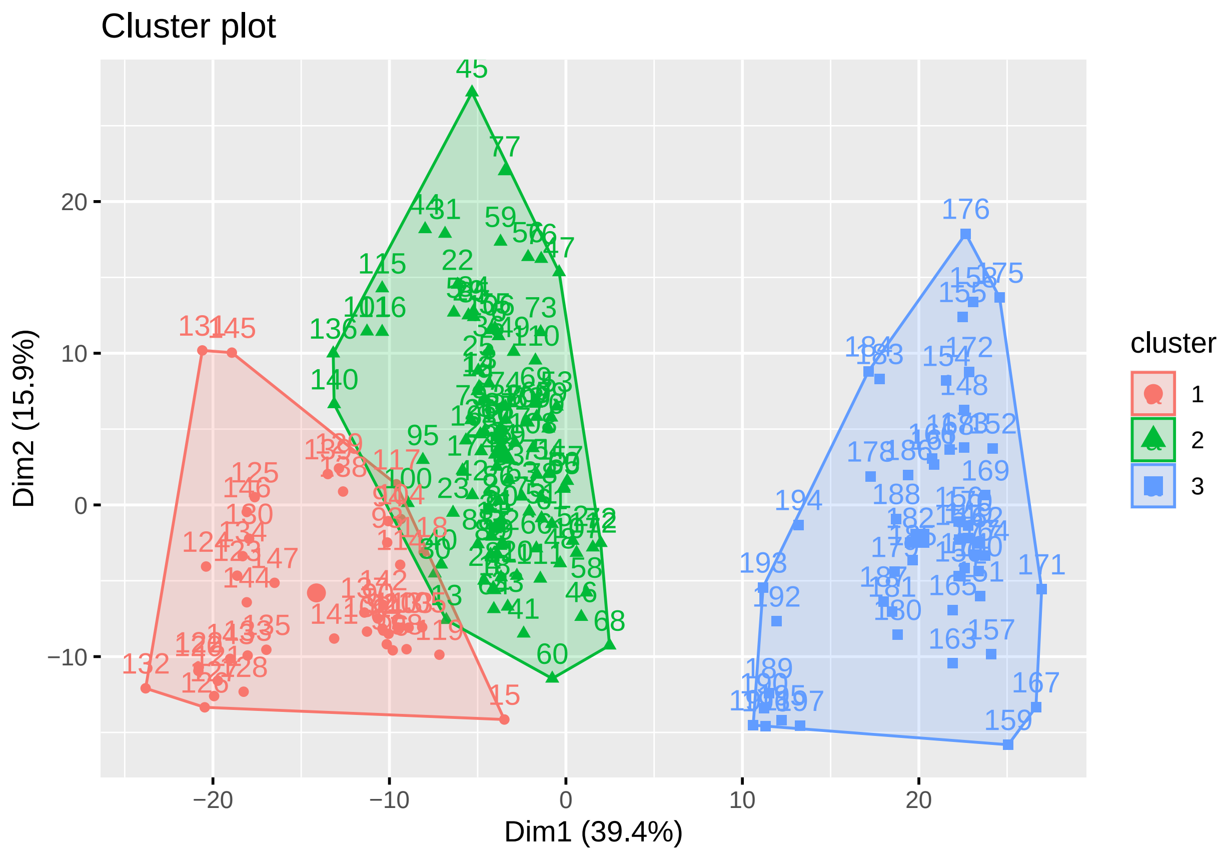
```
#plot the 2 K Means clusters
fviz_cluster(km2, data = datard_n)
```



```
#with K means cluster = 3
km3 <- kmeans(datard_n, centers = 3, nstart = 20)
```

Based on the results, the 3 K-means clusters is of sizes 40, 50, 107.

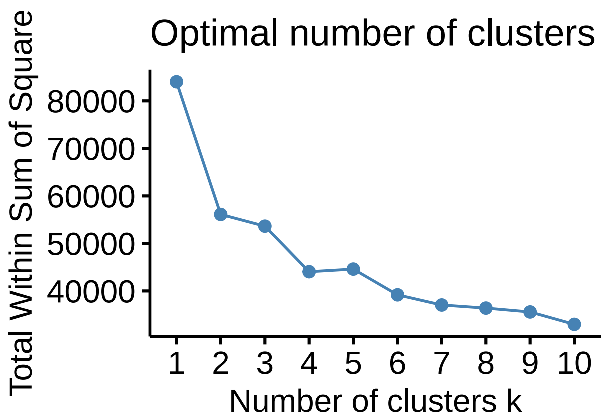
```
#plot the 3 K Means clusters
fviz_cluster(km3, data = datard_n)
```

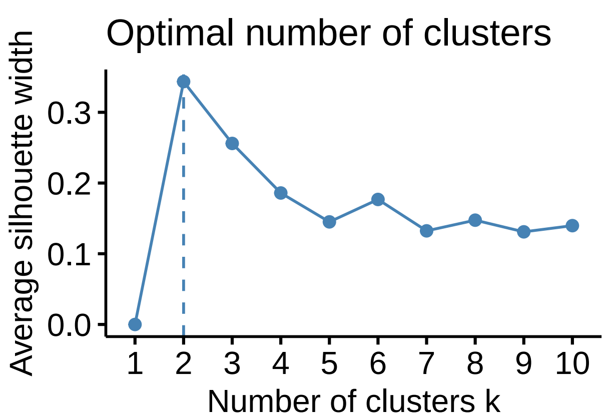


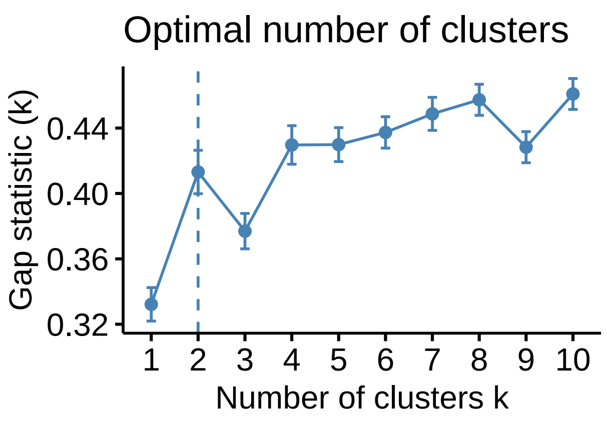
To determine and visualize the optimal number of clusters using different methods: *wws*, *silhoutte*, and *gap statistics*, we use the function `fviz_nbclust`.

```
plot1 <- fviz_nbclust(datard_n, kmeans, method = "wss")
plot2 <- fviz_nbclust(datard_n, kmeans, method = "silhouette")
plot3 <- fviz_nbclust(datard_n, kmeans, method = "gap_stat")

grid.arrange(plot1, plot2, plot3, nrow=2)
```







The location of a knee in the plot is usually considered as an indicator of the appropriate number of clusters because it means that adding another cluster does not improve much better the partition. Based on the plot, the three methods seems to suggest 2 clusters.\

```
#The quality of the 2K means partition
km2$betweenss / km2$totss
```

```
## [1] 0.3322453
```

The quality of the 2 K means partition is 0.3322453 or 33.22%.\

```
#The quality of the 3 K means partition
km3$betweenss / km3$totss
```

```
## [1] 0.4189776
```

The quality of the 3 K means partition is 0.4189776 or 41.9%.

2. HEIRARCHICAL CLUSTERING

Hierarchical clustering is an alternative approach to k-means clustering for identifying groups in a data set. The difference with the partition by k-means is that for hierarchical clustering, the number of classes is not specified in advance. Furthermore, hierarchical clustering has an added advantage over k-means clustering in that its results can be easily visualized using an attractive tree-based representation called a *dendrogram*. It will also help to determine the optimal number of clusters.

```
datahc <- datard %>%
  select_if(is.numeric) %>%       # select numeric columns
  select(-Failure.binary) %>%     # remove Failure.binary
  mutate_all(as.double) %>%       # coerce to double type
  scale()                         # center & scale the resulting columns
```

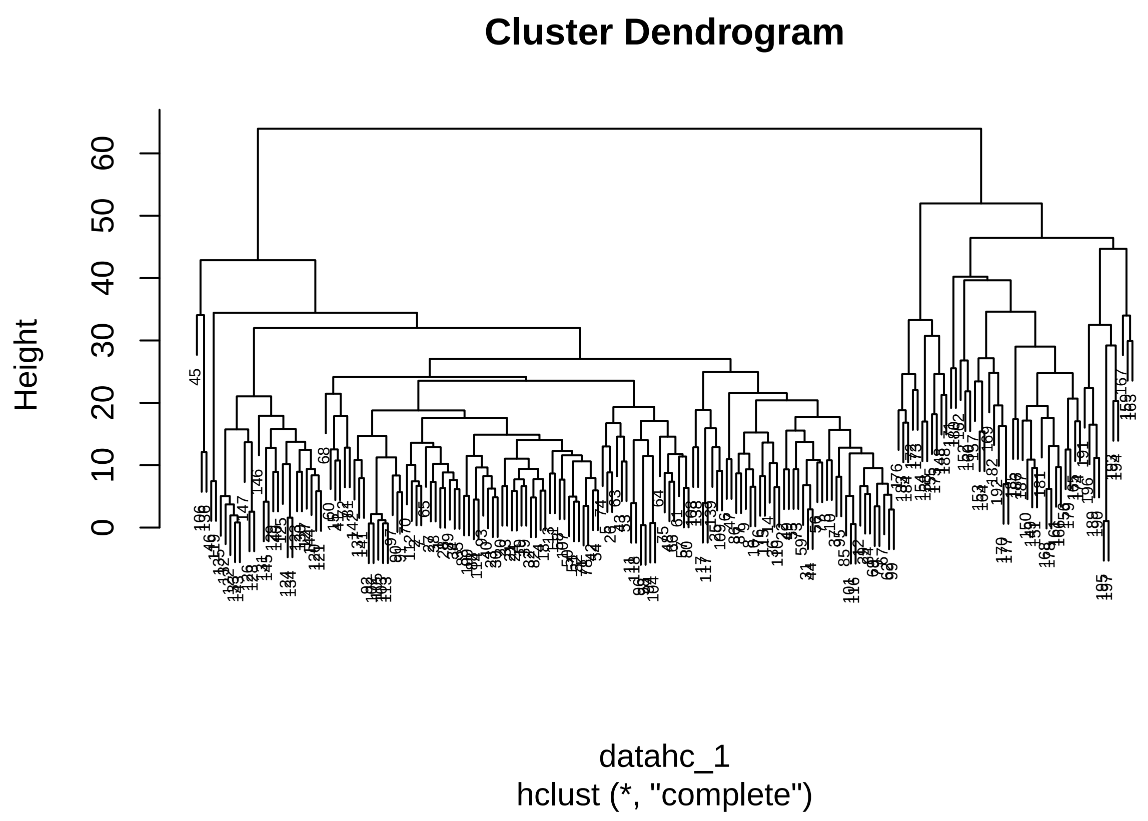
To perform **Agglomerative HC**, we first compute the dissimilarity values with `dist()` and then feed these values into `hclust()` and specify the agglomeration method to be used ie.`"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"`. Note that the `hclust()` function requires a distance matrix. If your data is not already a distance matrix, you can transform it into a distance matrix with the `dist()` function like we did.

```
datahc_1 <- dist(datahc, method = "euclidean")

# Hierarchical clustering using Complete Linkage
hc1 <- hclust(datahc_1, method = "complete")
```

To plot the dendogram, we can use the following syntax

```
plot(hc1, cex = 0.5)
```



A different option is to utilize the `agnes()` function. Similarly to `hclust()` function, it also provides the Agglomerative coefficient (AC), a measure of the amount of clustering structure found.

```
#AGNES
set.seed(123) #for reproducibility
ag <- agnes(datahc, method = "complete")
ag$ac
```

```
## [1] 0.8489113
```

The AC value is 0.8489113 which is closer to 1, hence it suggests a more balanced clustering structure.

To get the Agglomerative coefficient for each linkage method

```
# methods to assess
meth <- c( "average", "single", "complete", "ward")
names(meth) <- c( "average", "single", "complete", "ward")

# function to compute coefficient
AC <- function(x) {
  agnes(datahc, method = x)$ac
}

# get Agglomerative coefficient for each linkage method
purrr::map_dbl(meth, AC)
```

```
##   average    single  complete      ward
## 0.7616680 0.7098672 0.8489113 0.9654737
```

The function `diana()` allows us to perform **Divisive HC**. `diana()` works similar to `agnes()`; however, there is no agglomeration method to provide. A divisive coefficient (DC) closer to one suggests stronger group distinctions.

```
#DIANA
dn <- diana(datahc)
dn$dc
```

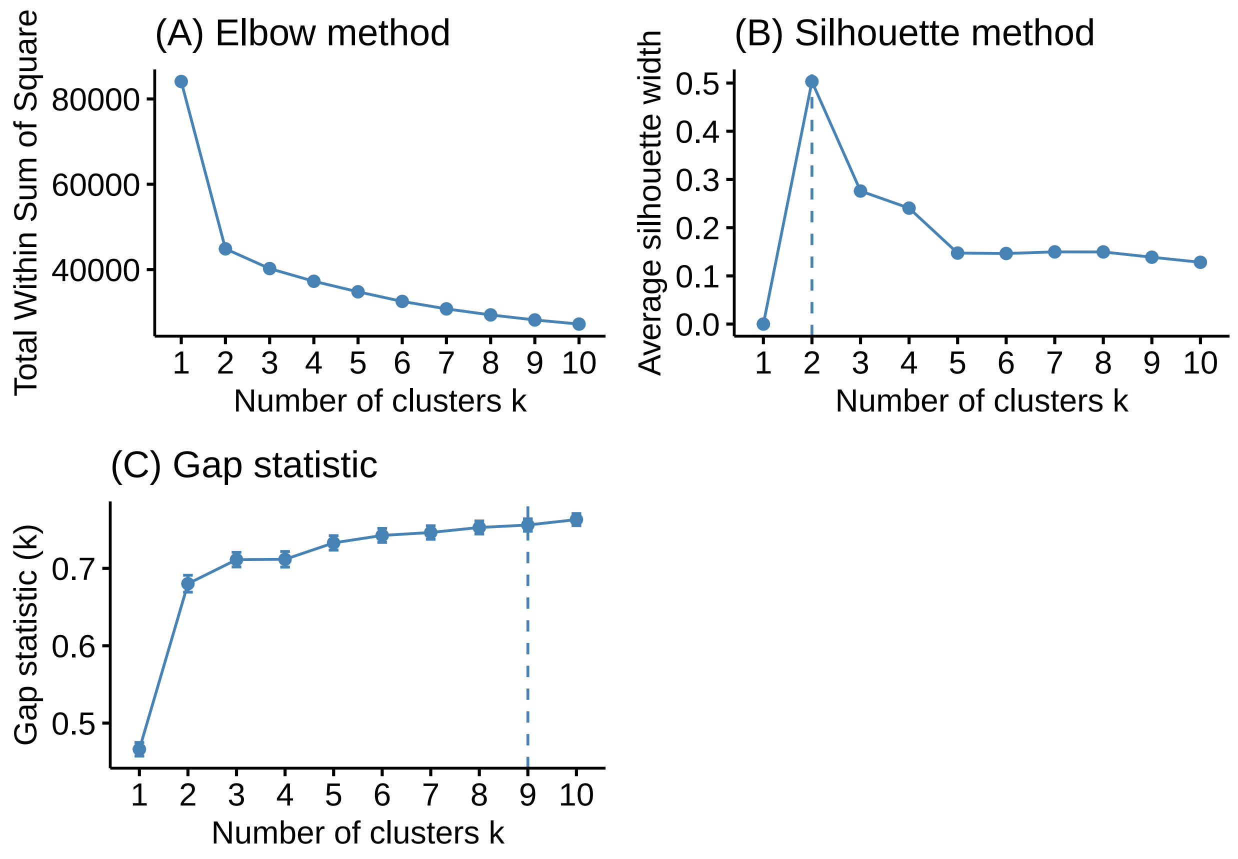
```
## [1] 0.8428381
```

The results gives us 0.8428381 which suggest that there is a stronger group distinctions.

To identify the optimal number of clusters, the following compare the results from the elbow, silhouette, and gap statistic methods.

```
plot4 <- fviz_nbclust(datahc, FUN = hcut, method = "wss",
                   k.max = 10) +
  ggtitle("(A) Elbow method")
plot5 <- fviz_nbclust(datahc, FUN = hcut, method = "silhouette",
                   k.max = 10) +
  ggtitle("(B) Silhouette method")
plot6 <- fviz_nbclust(datahc, FUN = hcut, method = "gap_stat",
                   k.max = 10) +
  ggtitle("(C) Gap statistic")

gridExtra::grid.arrange(plot4, plot5, plot6, nrow = 2)
```



Based on the plot, the Elbow and Silhouette methods seems to suggest 2 clusters, while the Gap Statistics suggest 9 clusters.

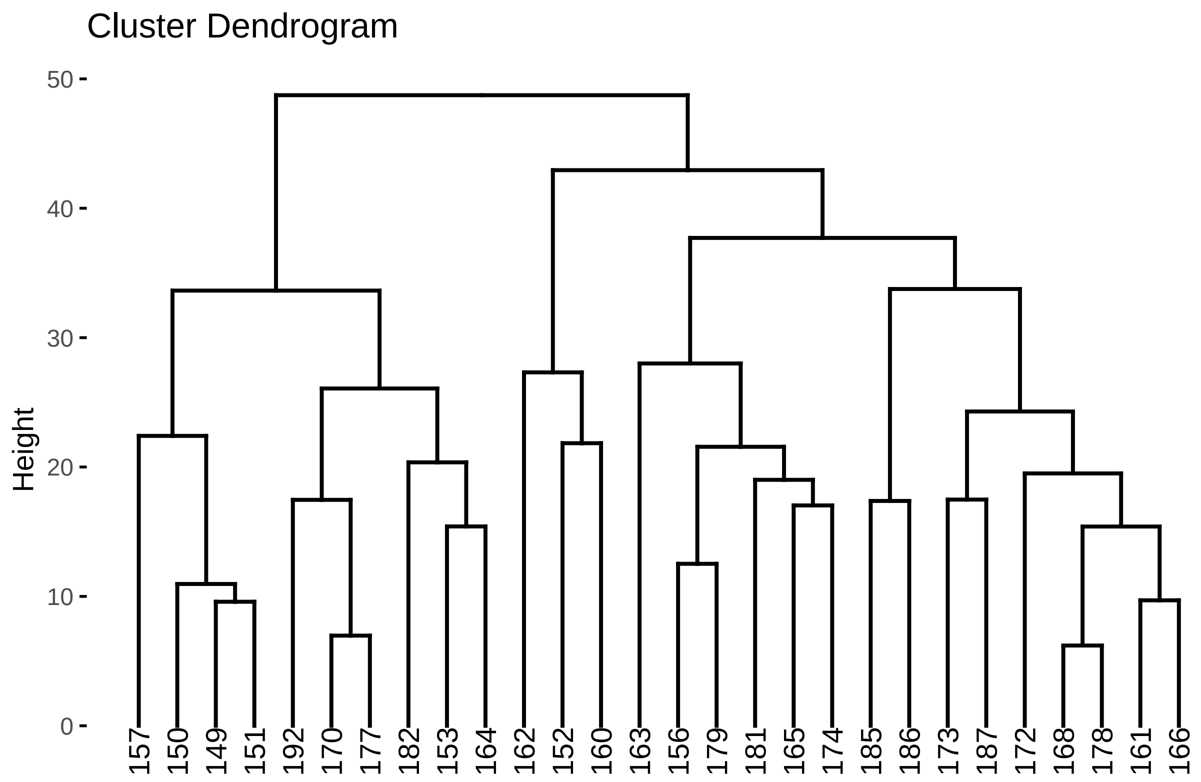
The wonderful thing about hierarchical clustering is that it gives us a complete dendrogram that shows the connections between the clusters in our data. The following syntax provides us a dendorgram.

```r
# Construct dendorgram for the radiomics data
datahc_2 <- hclust(datahc_1, method = "ward.D2" )
dend_plot <- fviz_dend(datahc_2)                        # create full dendogram
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```

```r
dend_data <- attr(dend_plot, "dendrogram")              # extract plot info
dend_cuts <- cut(dend_data, h = 50)                     # cut the dendogram at height=50.
fviz_dend(dend_cuts$lower[[4]])
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



Using the ward's method

```
datahc_2 <- hclust(datahc_1, method = "ward.D2" )
```

We can use the `cutree()` function to trim the dendrogram and identify clusters. Cut tree into 8 groups/clusters.

```
sub_grp <- cutree(datahc_2, k = 8)
```

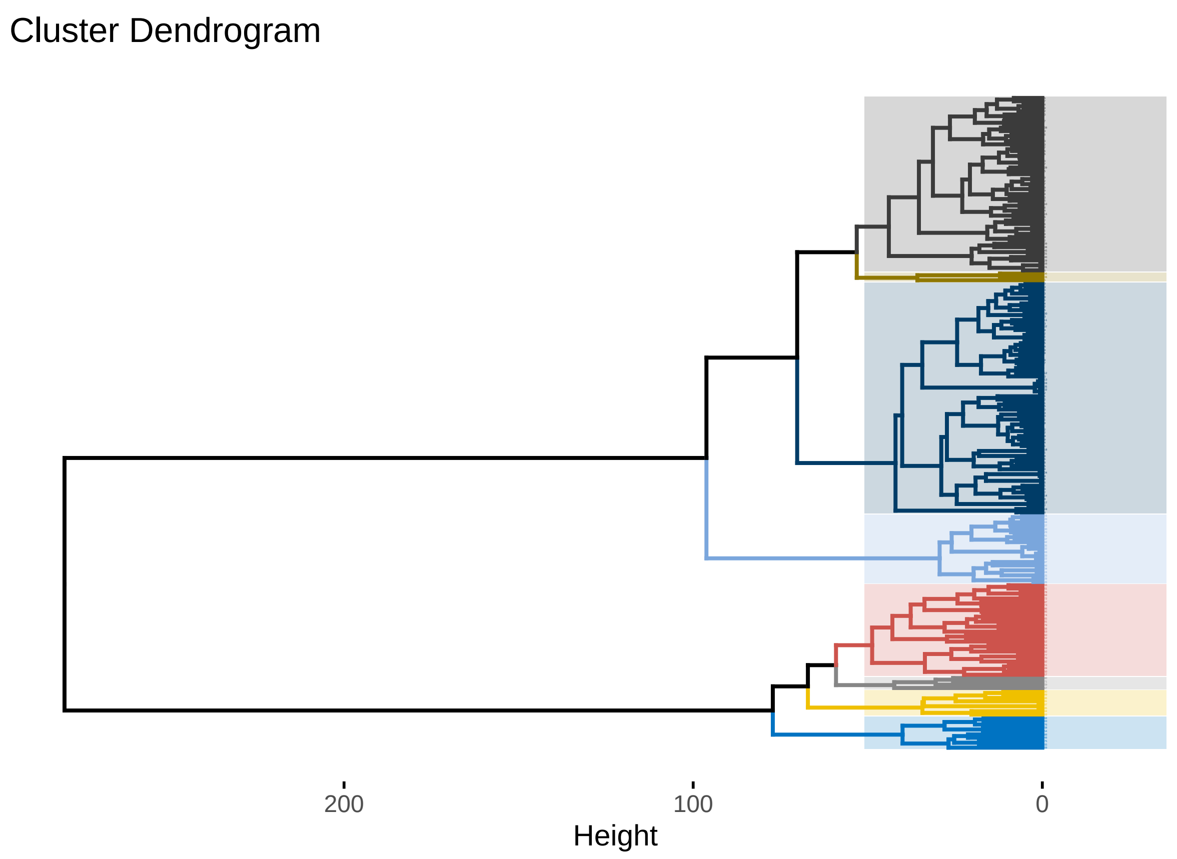
The number of members in each cluster is

```
table(sub_grp)
```

```
## sub_grp
##  1  2  3  4  5  6  7  8
## 70 53  3 21 10 28  4  8
```

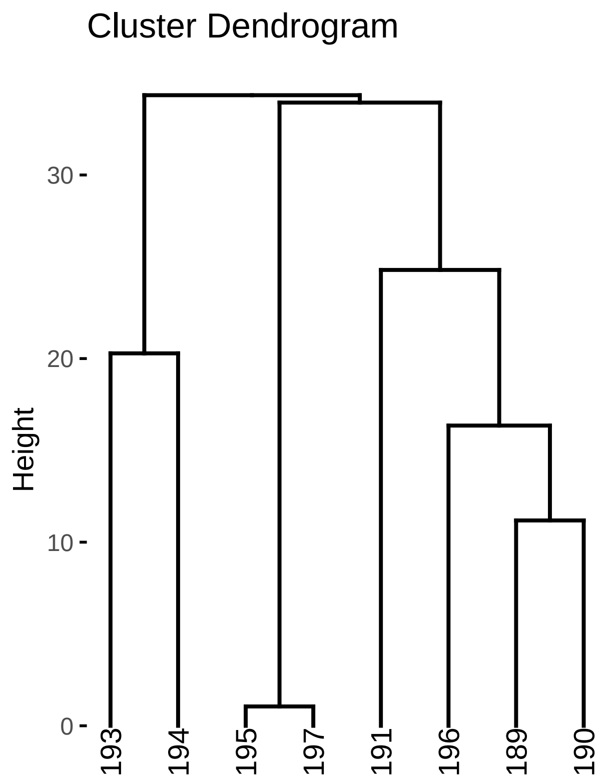
The following syntax plot the full dendogram of `datahc_2`. We use the function `fviz_dend()` to plot the entire dendogram.

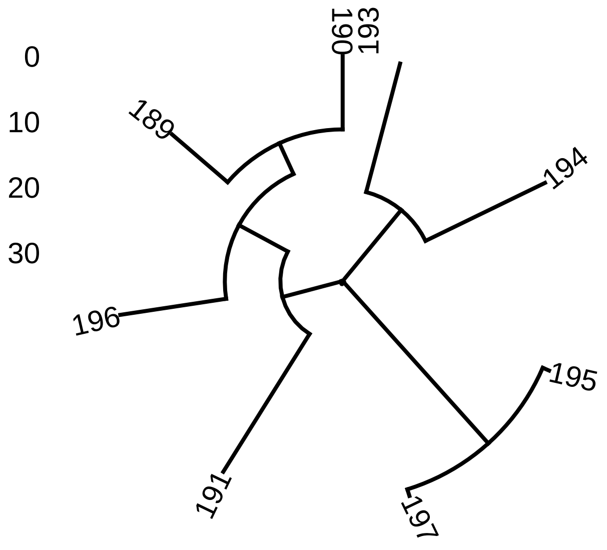
```
# Plot full dendogram
fviz_dend(
  datahc_2,
  k = 8,
  horiz = TRUE,
  rect = TRUE,
  rect_fill = TRUE,
  rect_border = "jco",
  k_colors = "jco",
  cex = 0.1
)
```



```
# Create sub dendrogram plots
plot7 <- fviz_dend(dend_cuts$lower[[2]])
plot8 <- fviz_dend(dend_cuts$lower[[2]], type = 'circular')

# Side by side plots
gridExtra::grid.arrange(plot7, plot8, nrow = 1)
```





3. MODEL-BASED CLUSTERING

Traditional clustering algorithms such as k-means and hierarchical clustering are heuristic-based algorithms that derive clusters directly based on the data rather than incorporating a measure of probability or uncertainty to the cluster assignments. **Model-based clustering** attempts to address this concern and provide soft assignment where observations have a probability of belonging to each cluster. Moreover, model-based clustering provides the added benefit of automatically identifying the optimal number of clusters.

The key idea behind model-based clustering is that the data are considered as coming from a mixture of underlying probability distributions. The most popular approach is the *Gaussian mixture model (GMM)* where each observation is assumed to be distributed as one of
k multivariate-normal distributions.

To do so we apply `Mclust()` for column 1 to column 20 and specify 3 components.

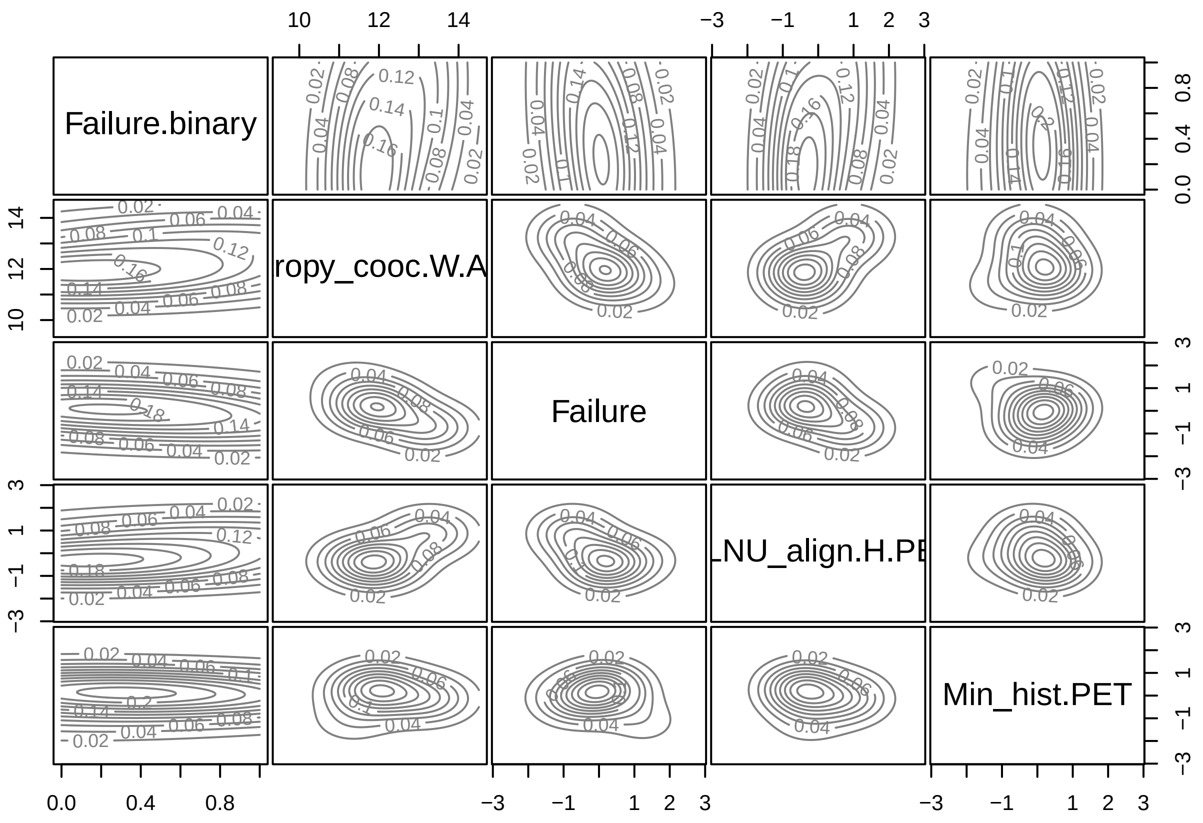
```
datamb <- Mclust(datard_n[,1:5], G=3)
summary(datamb)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust EII (spherical, equal volume) model with 3 components:
##
##  log-likelihood   n df       BIC       ICL
##       -1231.824 197 18 -2558.746 -2607.523
##
```

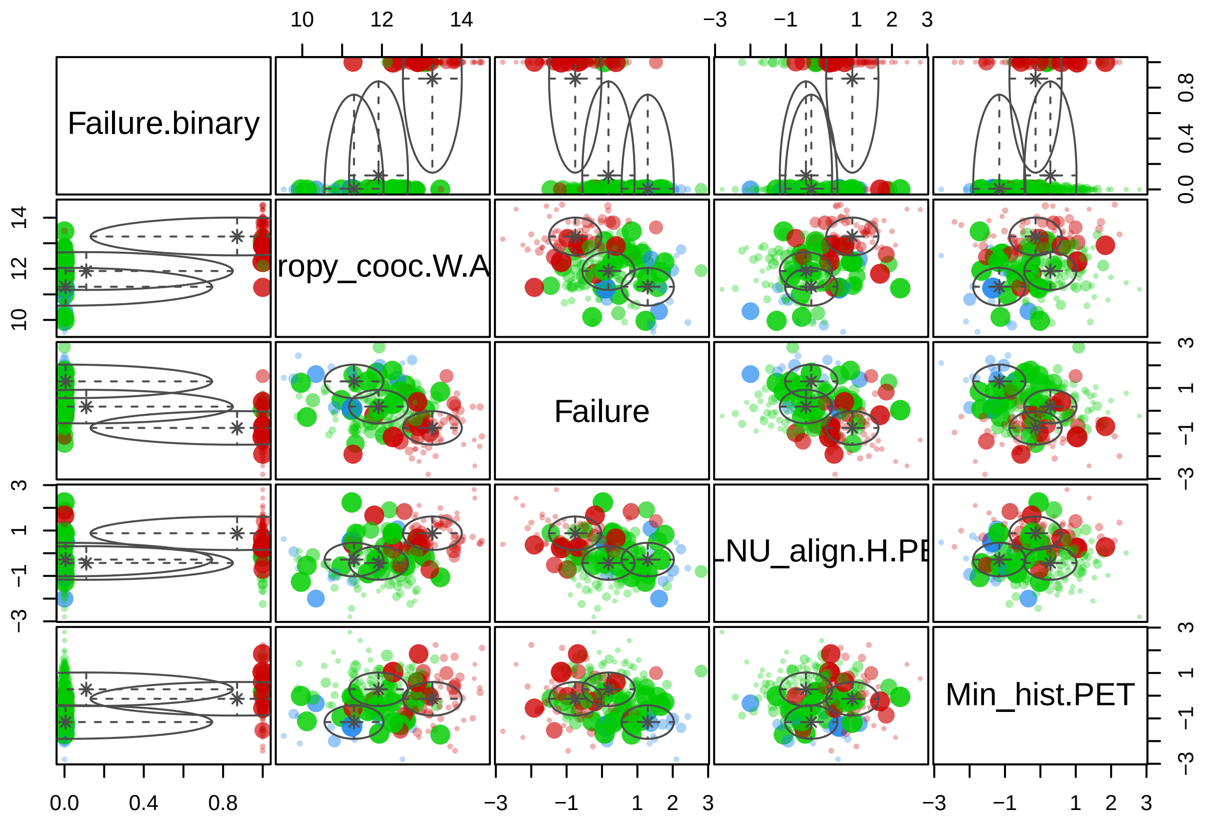
```
## Clustering table:
##   1   2   3
##  14  62 121
```

To plot the results, we have

```
# Plot results
plot(datamb, what = "density")
```



```
plot(datamb, what = "uncertainty")
```



The observation with high uncertainty are as follows:

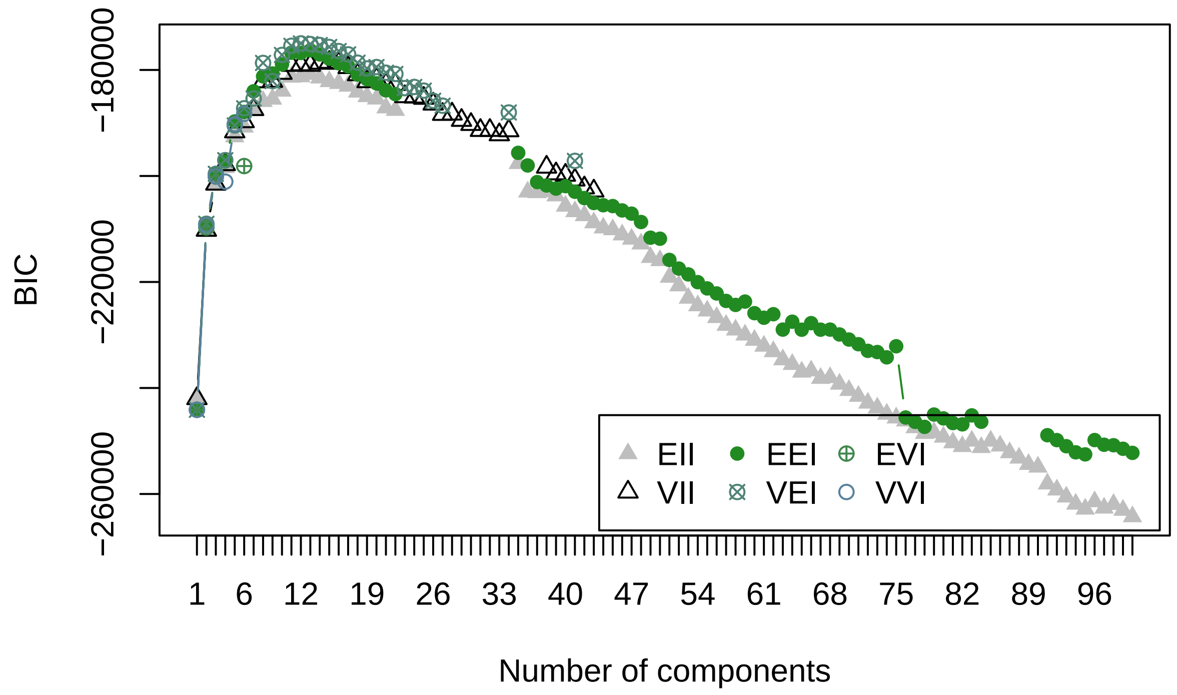
```r
sort(datamb$uncertainty, decreasing = TRUE) %>% head()
```

```
## [1] 0.4901136 0.4865014 0.4760988 0.4745343 0.4639933 0.4625717
```

```r
legend_args <- list(x = "bottomright", ncol = 5)
```

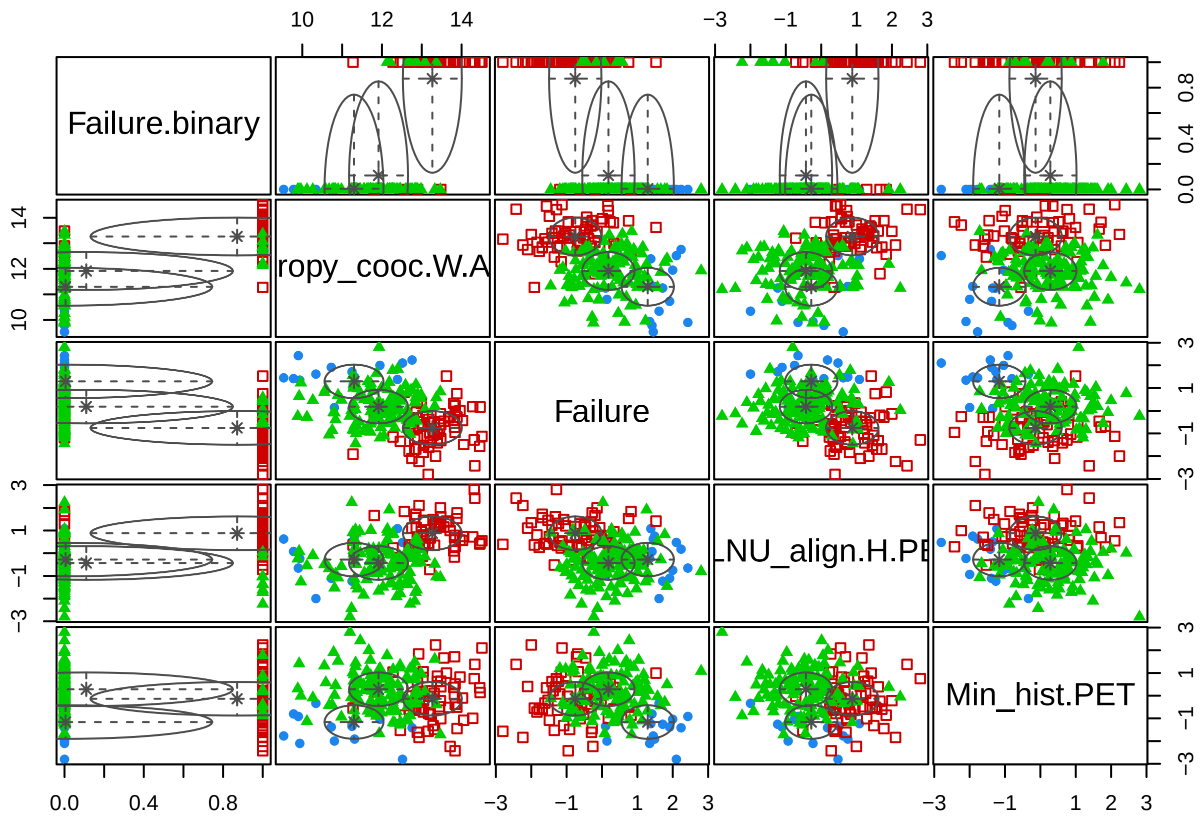
We can use `what = "BIC"` to identify the optimal covariance parameters and to identify the optimal number of clusters. Here, we define a new function `datamb1` to have a visualization of the `BIC` plot.

```r
datamb1 <- Mclust(datard_n,1:100)
plot(datamb1, what = 'BIC', legendArgs = legend_args)
```

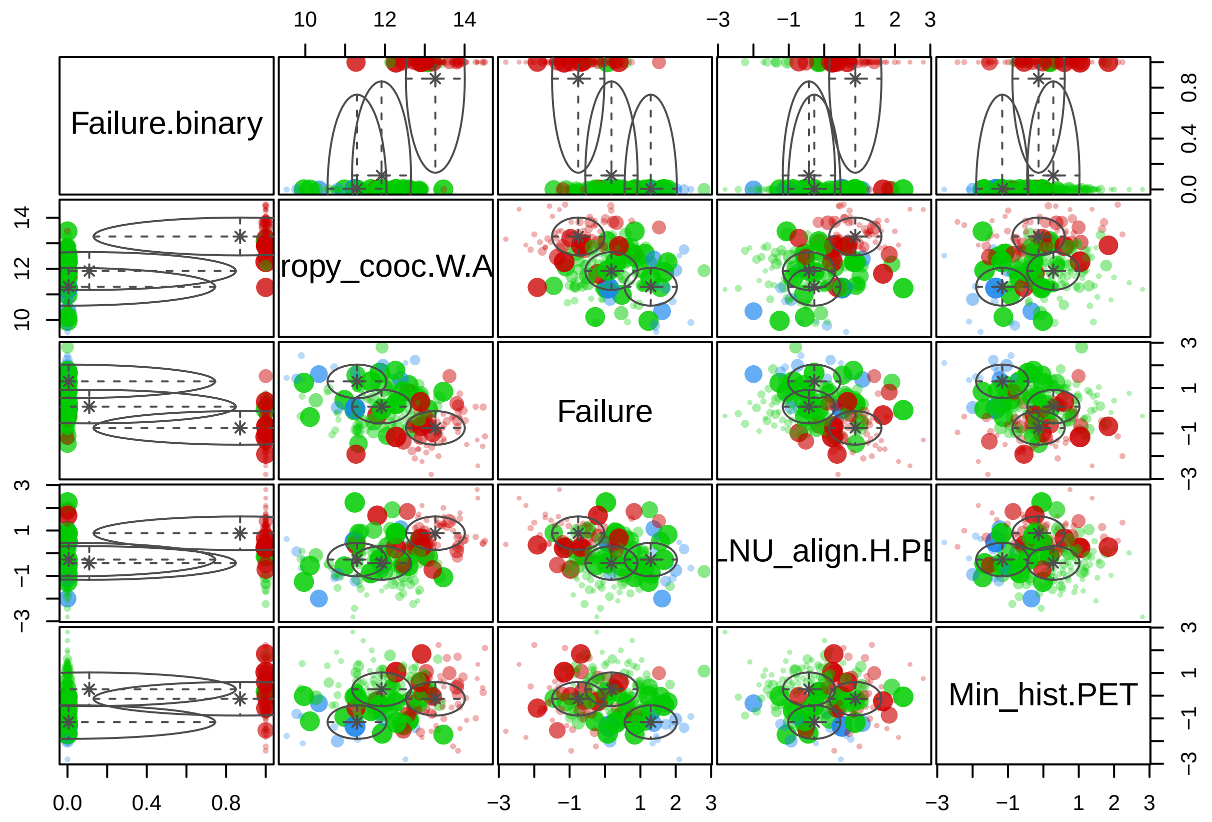


Based on the plot, it also shows that the EII and VII models perform particularly poor while the rest of the models perform much better, VVI is the Mclust model object.

```
plot9 <- plot(datamb, what = 'classification')
```



```r
plot10 <- plot(datamb, what = 'uncertainty')
```



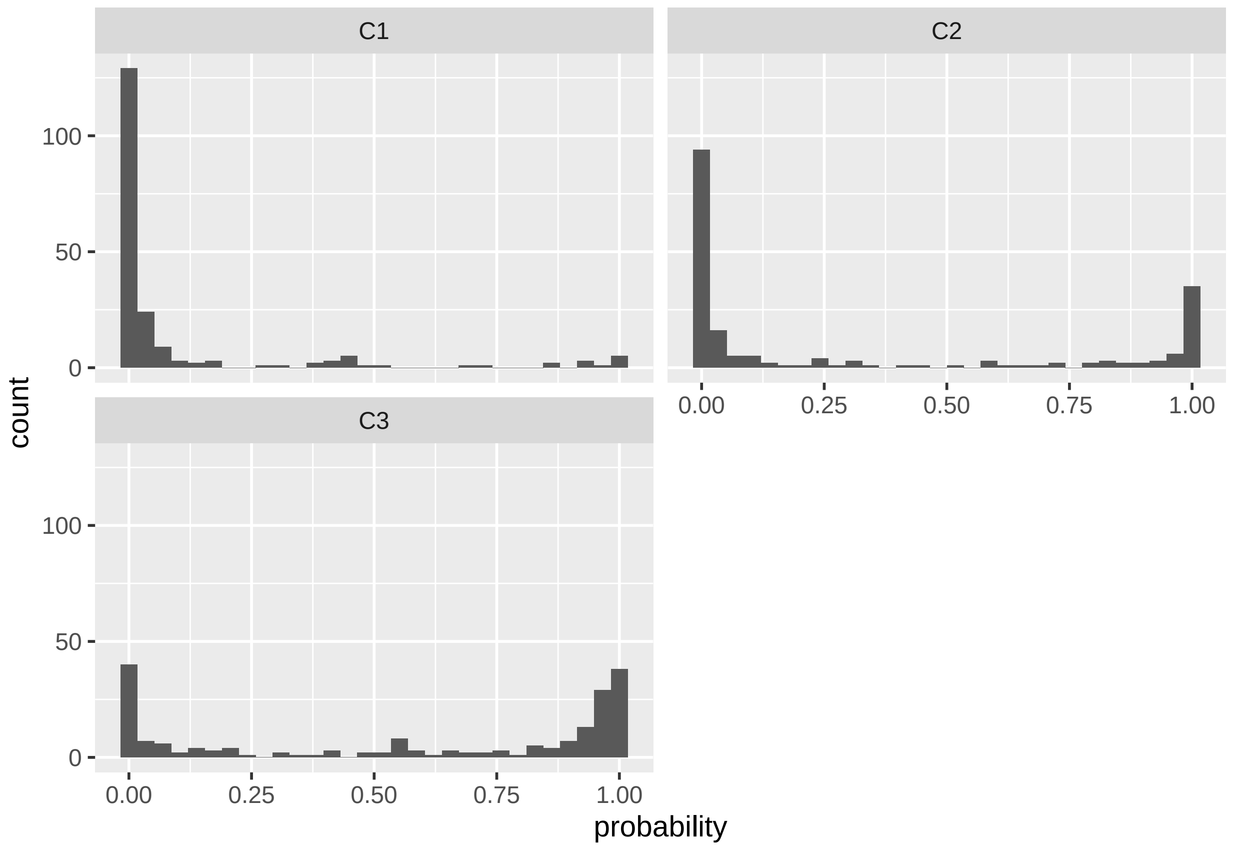
The classification and uncertainty plots illustrate which observations are assigned to each cluster and their level of assignment uncertainty.

```r
probabilities <- datamb$z
colnames(probabilities) <- paste0('C', 1:3)
```

```r
probabilities <- probabilities %>%
  as.data.frame() %>%
  mutate(id = row_number()) %>%
  tidyr::gather(cluster, probability, -id)
```

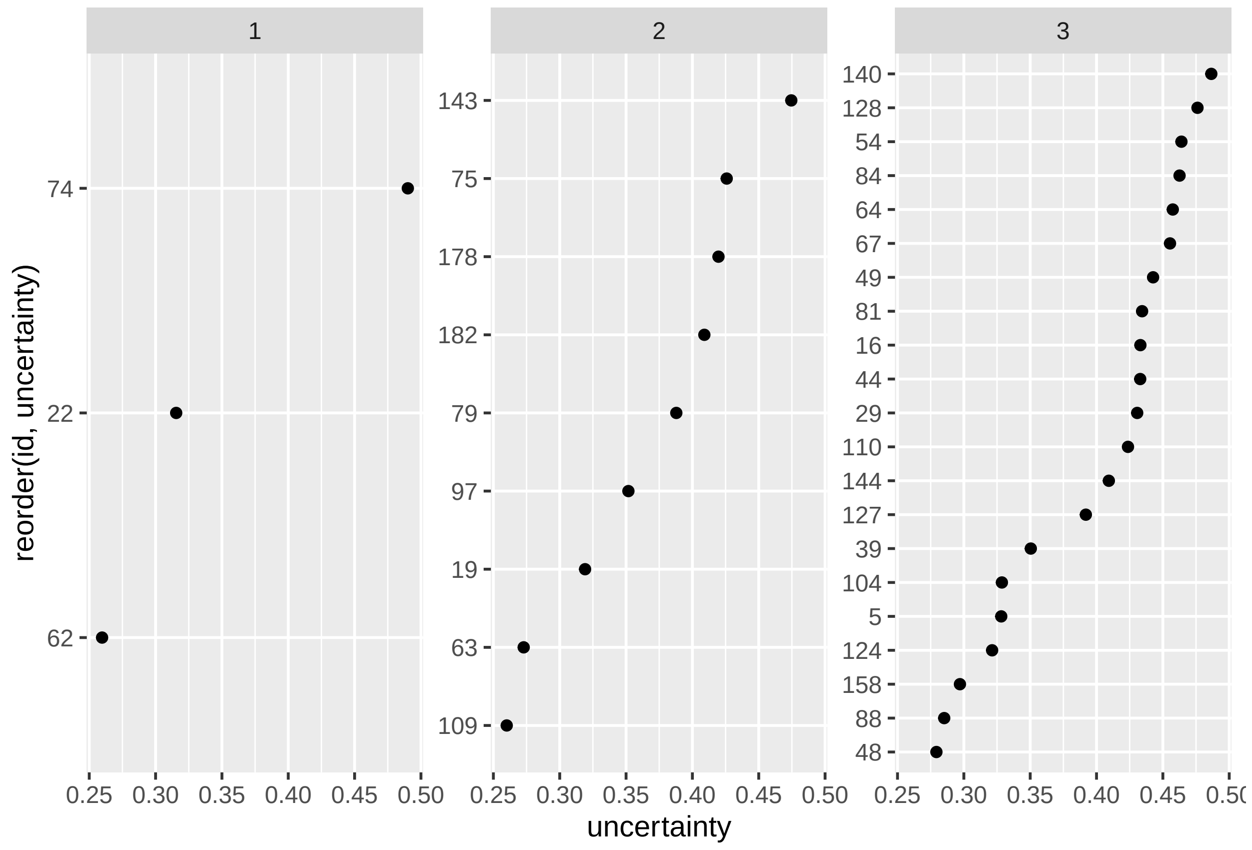
```r
ggplot(probabilities, aes(probability)) +
  geom_histogram() +
  facet_wrap(~ cluster, nrow = 2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



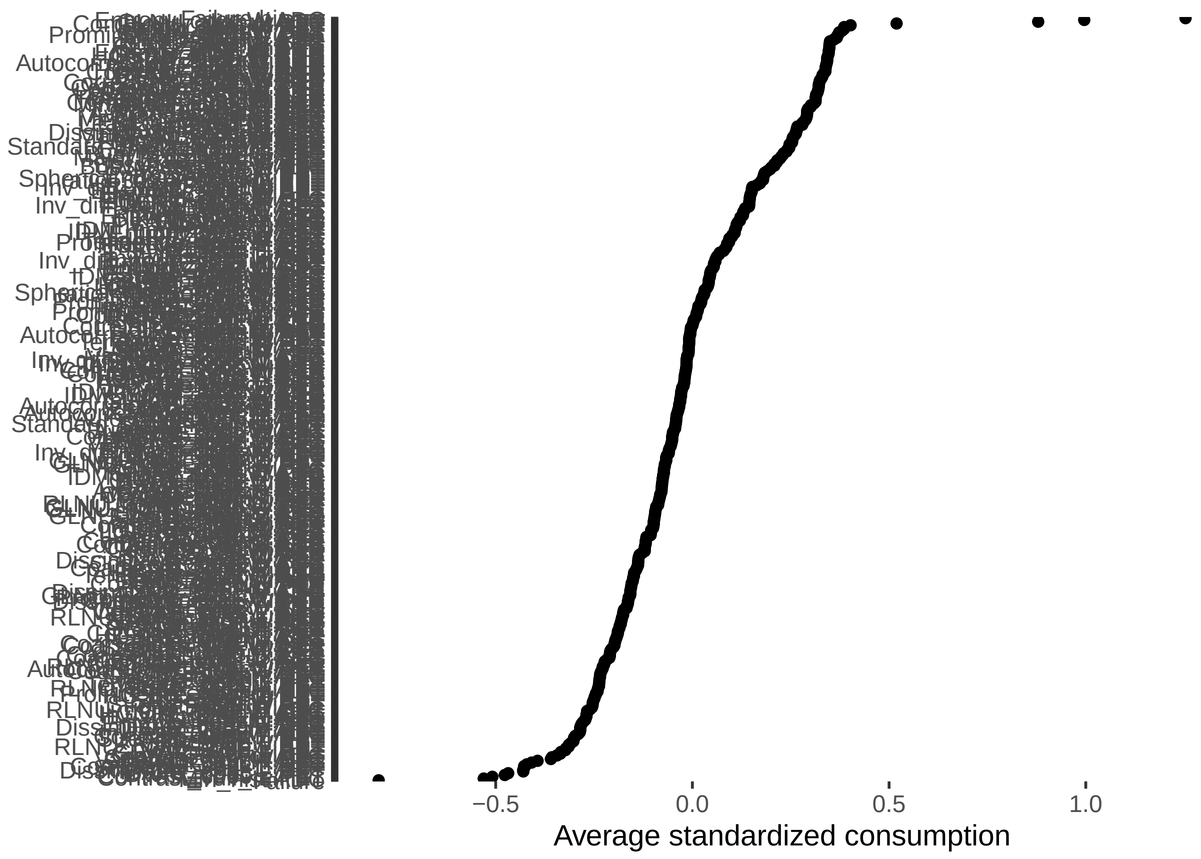
```r
uncertainty <- data.frame(
  id = 1:nrow(datard_n),
  cluster = datamb$classification,
  uncertainty = datamb$uncertainty
)
```

```r
uncertainty %>%
  group_by(cluster) %>%
  filter(uncertainty > 0.25) %>%
  ggplot(aes(uncertainty, reorder(id, uncertainty))) +
  geom_point() +
  facet_wrap(~ cluster, scales = 'free_y', nrow = 1)
```



```
cluster2 <- datard_n %>%
  scale() %>%
  as.data.frame() %>%
  mutate(cluster = datamb$classification) %>%
  filter(cluster == 2) %>%
  select(-cluster)
```

```
cluster2 %>%
  tidyr::gather(product, std_count) %>%
  group_by(product) %>%
  summarize(avg = mean(std_count)) %>%
  ggplot(aes(avg, reorder(product, avg))) +
  geom_point() +
  labs(x = "Average standardized consumption", y = NULL)
```



−0.5
0.0
0.5
1.0
Average standardized consumption