Capstone Project

2022-12-10

```
library(rmarkdown)
library(dplyr)       # basic data manipulation and plotting

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(ggplot2)     # data visualization
library(h2o) # performing dimension reduction

##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------

##
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':
##
##     cor, sd, var

## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
library(RIA)

##
## ################################
## ##         WELCOME TO         ##
## ##          __________        ##
## ##        / _  / / _ |        ##
## ##       / , _/ / __ |        ##
## ##      /_/|_/_/_/ |_|        ##
## ## RADIOMICS IMAGE ANALYSIS ##
## ##    Márton Kolossváry     ##
## ################################
##
## Please cite:
## Márton KOLOSSVÁRY et al.
## Radiomic Features Are Superior to Conventional Quantitative Computed Tomog
raphic
## Metrics to Identify Coronary Plaques With Napkin-Ring Sign
## Circulation: Cardiovascular Imaging (2017).
## DOI: 10.1161/circimaging.117.006843
##
## Márton KOLOSSVÁRY et al.
## Cardiac Computed Tomography Radiomics: A Comprehensive Review on Radiomic
Techniques.
## Journal of Thoracic Imaging (2018).
## DOI: 10.1097/RTI.0000000000000268
##
## Please check for updates regularly for bug fixes and new functionalities!
##

h2o.init()

##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         58 minutes 36 seconds
##     H2O cluster timezone:       America/Toronto
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.38.0.1
##     H2O cluster version age:    2 months and 24 days
##     H2O cluster name:           H2O_started_from_R_jesia_osn291
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.89 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
```

```
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.2.2 (2022-10-31 ucrt)

# Helper packages
  # for awesome graphics
library(rsample)  # for data splitting

# Modeling packages
library(caret)    # for classification and regression training

## Loading required package: lattice

library(kernlab)  # for fitting SVMs

##
## Attaching package: 'kernlab'

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

library(modeldata) #for attrition data

# Model interpretability packages
library(pdp)      # for partial dependence plots, etc.
library(vip)      # for variable importance plots

##
## Attaching package: 'vip'

## The following object is masked from 'package:utils':
##
##     vi

library(readr)
library(dplyr)
library(plyr)

## ------------------------------------------------------------------------
----

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, th
en dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------
----

##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

#Load the dataset
df = read.csv("radiomics_completedata.csv")
```

Model 1 (GLM, RF, GBM)

```
#Check for null and missing values
df <- na.omit(df)

#Check for normality, if not, normalized the data
newdf1 = select(df, -c("Institution","Failure.binary", "Failure"))
newdf1  <- scale(newdf1[c(1:10)])
head(newdf1, 5)

##   Entropy_cooc.W.ADC GLNU_align.H.PET Min_hist.PET Max_hist.PET Mean_hist.
PET
## 1         0.55290547      -0.57063689   -0.4541408   -0.4361311    -0.4204
856
## 2        -0.06486729      -0.78903636    0.4998369    0.1486951     0.3153
953
## 3         0.45990825      -0.06024275   -1.1504338   -1.1768823    -1.1362
283
## 4         1.14318298       2.67468822   -0.4446190   -0.1516658    -0.3486
295
## 5         0.34499368      -0.06740573   -0.9887407   -1.1061760    -1.1155
134
##   Variance_hist.PET Standard_Deviation_hist.PET Skewness_hist.PET
## 1        -0.2625994                  -0.2362506        -0.3229376
## 2         0.3949731                   0.2970175        -0.1769772
## 3        -0.8957972                  -1.1289710        -0.9586986
## 4        -0.2802885                  -0.2534091        -0.1155757
## 5        -0.9335606                  -1.2398300         0.9580073
##   Kurtosis_hist.PET Energy_hist.PET
## 1        -0.2730969      0.05021980
## 2        -0.2664840      0.09191129
## 3        -0.4718456      0.04744499
## 4         0.1199784     -0.01242149
## 5         0.9071980      0.15326924

#Get the correlation of the whole data expect the categorical variables
library(caret)

cor.newdf1 = cor(newdf1)
```

```r
corr = round(cor.newdf1,2)

corMatrix =  cor(newdf1, y = NULL, use = "ev")
highly_correlated_columns = findCorrelation(
  corMatrix,
  cutoff = 0.85, # correlation coefficient
  verbose = FALSE,
  names = FALSE,
  exact = TRUE
)
DT <- newdf1[, -highly_correlated_columns]

finaldata <- cbind(df['Failure.binary'], DT)

# Helper packages
library(rsample)   # for creating our train-test splits
library(recipes)   # for minor feature engineering tasks

##
## Attaching package: 'recipes'

## The following object is masked from 'package:RIA':
##
##     discretize

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

# Modeling packages
library(h2o)       # for fitting stacked models

#Split the data into training (80%) and testing (20%)

set.seed(123)  # for reproducibility
split <- initial_split(finaldata,  prop = 0.8, strata = "Failure.binary")
radio_train <- training(split)
radio_test <- testing(split)

# Make sure we have consistent categorical levels
blueprint <- recipe(Failure.binary ~ ., data = radio_train) %>%
  step_other(all_nominal(), threshold = 0.005)
blueprint

## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          7
```

```
##
## Operations:
##
## Collapsing factor levels for all_nominal()
```

```
# Create training & test sets for h2o
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         58 minutes 42 seconds
##     H2O cluster timezone:       America/Toronto
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.38.0.1
##     H2O cluster version age:    2 months and 24 days
##     H2O cluster name:           H2O_started_from_R_jesia_osn291
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.89 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.2.2 (2022-10-31 ucrt)
```

```
train_h2o <- prep(blueprint, training = radio_train, retain = TRUE) %>%
  juice() %>%
  as.h2o()
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

```
train_h2o
```

```
##   Entropy_cooc.W.ADC GLNU_align.H.PET Min_hist.PET Variance_hist.PET
## 1         0.34499368      -0.06740573  -0.98874071        -0.9335606
## 2        -0.07231092      -0.68049871   0.02814910         0.9075403
## 3        -0.47508410      -0.04700109   0.25645562         0.4115808
## 4         0.30131288      -0.60733786  -1.00812837        -0.8163440
## 5        -1.86279117      -0.95781836  -0.49187889        -0.7596741
## 6        -0.94025294      -0.29104352   0.04032431        -0.1026903
##   Skewness_hist.PET Kurtosis_hist.PET Energy_hist.PET Failure.binary
## 1         0.9580073        0.90719796     0.153269240              0
## 2        -0.1049733       -0.10117242     0.035617374              0
## 3        -1.0693544       -0.43267939    -0.008883612              0
## 4        -0.1978712       -0.02509205     0.119416581              0
```

```
## 5        -0.2477281       -0.37819404     0.580624383              0
## 6        -1.1042312       -0.51084220     0.011545913              0
##
## [157 rows x 8 columns]
```

```
test_h2o <- prep(blueprint, training = radio_train) %>%
  bake(new_data = radio_test) %>%
  as.h2o()
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

```
# Get response and feature names


Y <- "Failure.binary"
X <- setdiff(names(radio_train), Y)
```

```
# Train & cross-validate a GLM model
best_glm <- h2o.glm(
  x = X, y = Y, training_frame = as.factor(train_h2o), alpha = 0.1,
  remove_collinear_columns = TRUE, nfolds = 10, fold_assignment = "Modulo",
  keep_cross_validation_predictions = TRUE, seed = 123
)
```

```
##   |
|                                                                      |   0%
|
|=========================                                             |  38%
|
|======================================================================| 100%
```

```
# Train & cross-validate a GBM model
best_gbm <- h2o.gbm(
  x = X, y = Y, training_frame = as.factor(train_h2o), ntrees = 5000, learn_r
ate = 0.01,
  max_depth = 7, min_rows = 5, sample_rate = 0.8, nfolds = 10,
  fold_assignment = "Modulo", keep_cross_validation_predictions = TRUE,
  seed = 123, stopping_rounds = 50, stopping_metric = "AUC",
  stopping_tolerance = 0
)
```

```
## Warning in .h2o.processResponseWarnings(res): early stopping is enabled bu
t neither score_tree_interval or score_each_iteration are defined. Early stop
ping will not be reproducible!.
```

```
##   |
|                                                                      |   0%
|
```

```
|=========================                                             |  37%
|
|=================================================                     |  73%
|
|======================================================================| 100%
```

```
best_rf <- h2o.randomForest(
  x = X, y = Y, training_frame = as.factor(train_h2o), ntrees = 1000, mtries
= 1,
  max_depth = 30, min_rows = 1, sample_rate = 0.8, nfolds = 10,
  fold_assignment = "Modulo", keep_cross_validation_predictions = TRUE,
  seed = 123, stopping_rounds = 50, stopping_metric = "AUC",
  stopping_tolerance = 0
)
```

```
## Warning in .h2o.processResponseWarnings(res): early stopping is enabled bu
t neither score_tree_interval or score_each_iteration are defined. Early stop
ping will not be reproducible!.
```

```
##   |
|                                                                      |   0%
|
|==                                                                    |   3%
|
|=================================================                     |  74%
|
|======================================================================| 100%
```

```
# Get results from base learners
get_rmse <- function(model) {
  results <- h2o.performance(model, newdata = test_h2o)
  results@metrics$RMSE
}
list(best_glm, best_rf, best_gbm) %>%
  purrr::map_dbl(get_rmse)
```

```
## [1] 0.4793936 0.5916080 0.5277967
```

```
# Train a stacked tree ensemble
ensemble_tree <- h2o.stackedEnsemble(
  x = X, y = Y, training_frame = as.factor(train_h2o), model_id = "my_tree_en
semble",
  base_models = list(best_glm, best_rf, best_gbm),
  metalearner_algorithm = "drf"
)
```

```
##   |
|                                                                      |   0%
|
|======================================================================| 100%
```

```r
# Stacked results
h2o.performance(ensemble_tree, newdata = test_h2o)@metrics$RMSE
```

```
## [1] 0.4773375
```

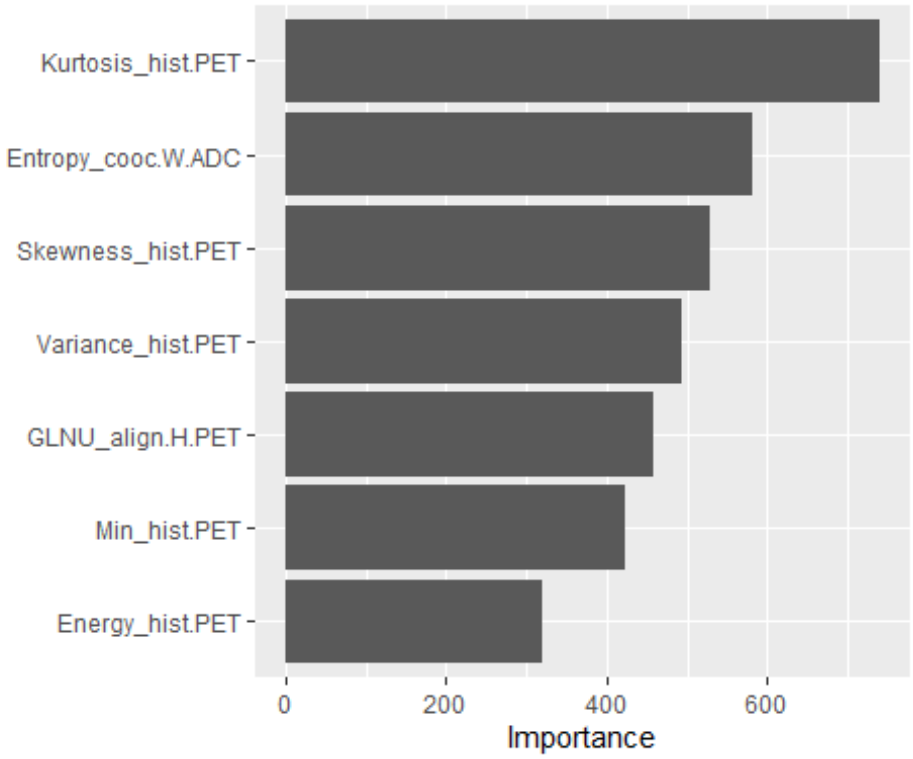
```r
data.frame(
  GLM_pred = as.vector(as.numeric(h2o.getFrame(best_glm@model$cross_validatio
n_holdout_predictions_frame_id$name))),
  RF_pred = as.vector(as.numeric(h2o.getFrame(best_rf@model$cross_validation_
holdout_predictions_frame_id$name))),
  GBM_pred = as.vector(as.numeric(h2o.getFrame(best_gbm@model$cross_validatio
n_holdout_predictions_frame_id$name)))
) %>% cor()
```

```
##           GLM_pred   RF_pred  GBM_pred
## GLM_pred 1.0000000 0.7550327 0.8296038
## RF_pred  0.7550327 1.0000000 0.9874022
## GBM_pred 0.8296038 0.9874022 1.0000000
```

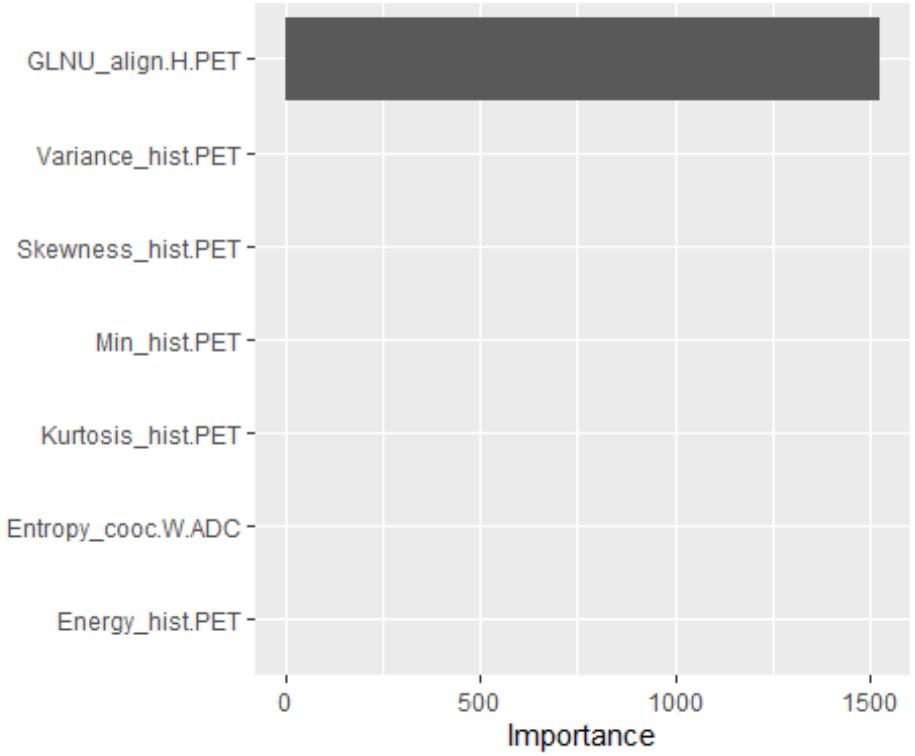
```r
#Print the AUC values during Training
perf1 <- h2o.performance(ensemble_tree, newdata = train_h2o)
h2o.auc(perf1)
```

```
## [1] 0
```

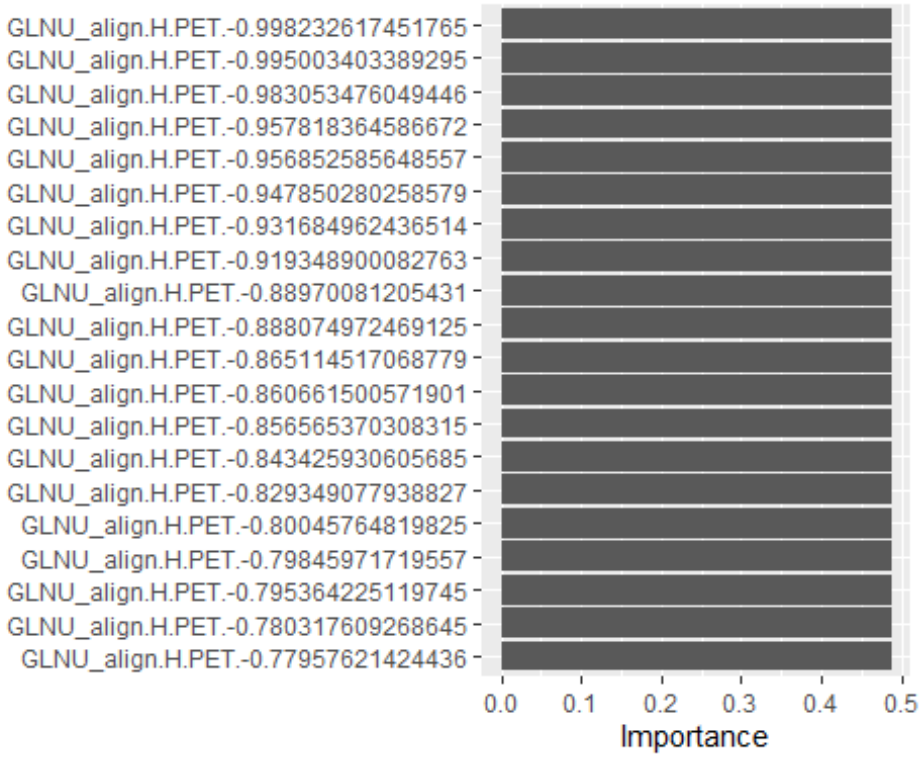
```r
#Print the Top 20 important features during Training
vip::vip(best_rf, 20)
```



```
vip::vip(best_gbm, 20)
```




```
vip::vip(best_glm, 20)
```



```
#calculate AUC values during testing
perf <- h2o.performance(ensemble_tree, newdata = test_h2o)
h2o.auc(perf)
```

```
## [1] 0.5
```

Model 3 (K-Means, Hierarchical & Model Based)

K-Means

```
# Helper packages
library(dplyr)       # for data manipulation
library(ggplot2)     # for data visualization
library(stringr)     # for string functionality
```

```
##
## Attaching package: 'stringr'
```

```
## The following object is masked from 'package:recipes':
##
##     fixed
```

```
library(gridExtra)   # for manipulaiting the grid
```

```
##
## Attaching package: 'gridExtra'
```

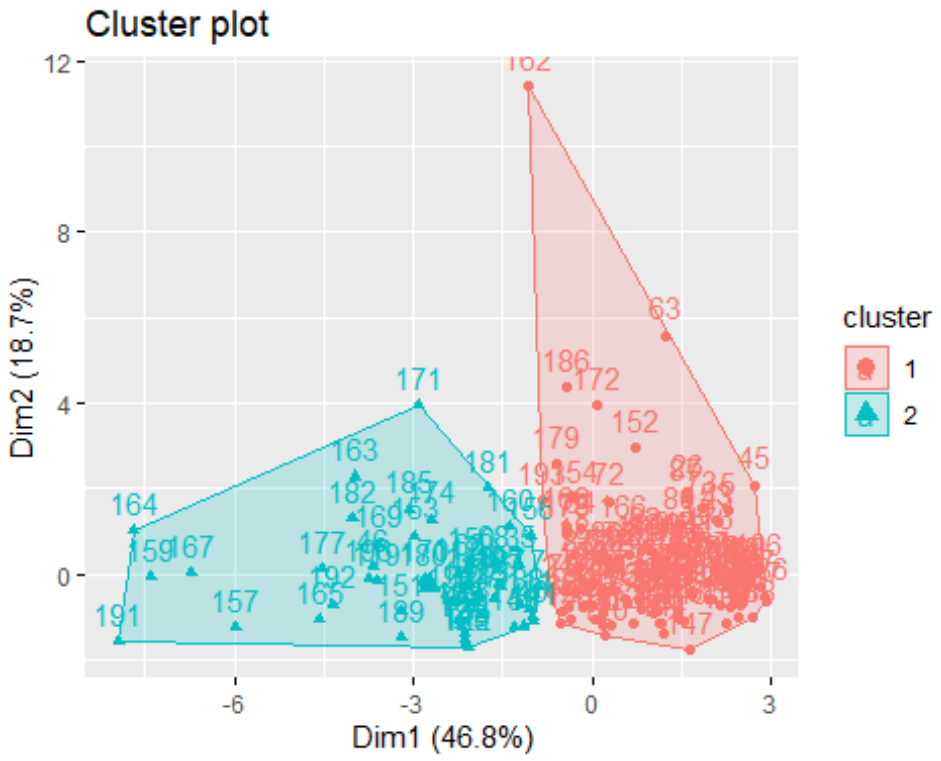
```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
# Modeling packages
library(tidyverse)  # data manipulation
```

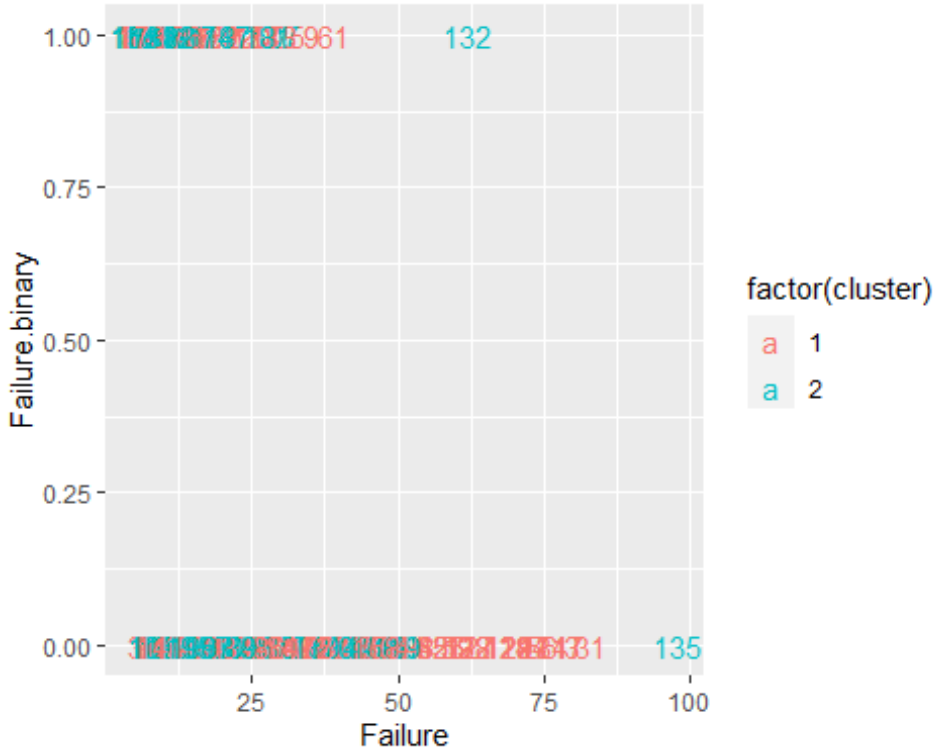
```
## ── Attaching packages
## ─────────────────────────────────────
## tidyverse 1.3.2 ──
```

```
## ✓ tibble  3.1.8     ✓ purrr   0.3.5
## ✓ tidyr   1.2.1     ✓ forcats 0.5.2
## ── Conflicts ──────────────────────────────────────── tidyverse_conflict
s() ──
## ✖ kernlab::alpha()     masks ggplot2::alpha()
## ✖ plyr::arrange()      masks dplyr::arrange()
## ✖ gridExtra::combine() masks dplyr::combine()
## ✖ purrr::compact()     masks plyr::compact()
## ✖ plyr::count()        masks dplyr::count()
## ✖ purrr::cross()       masks kernlab::cross()
## ✖ plyr::failwith()     masks dplyr::failwith()
## ✖ dplyr::filter()      masks stats::filter()
```

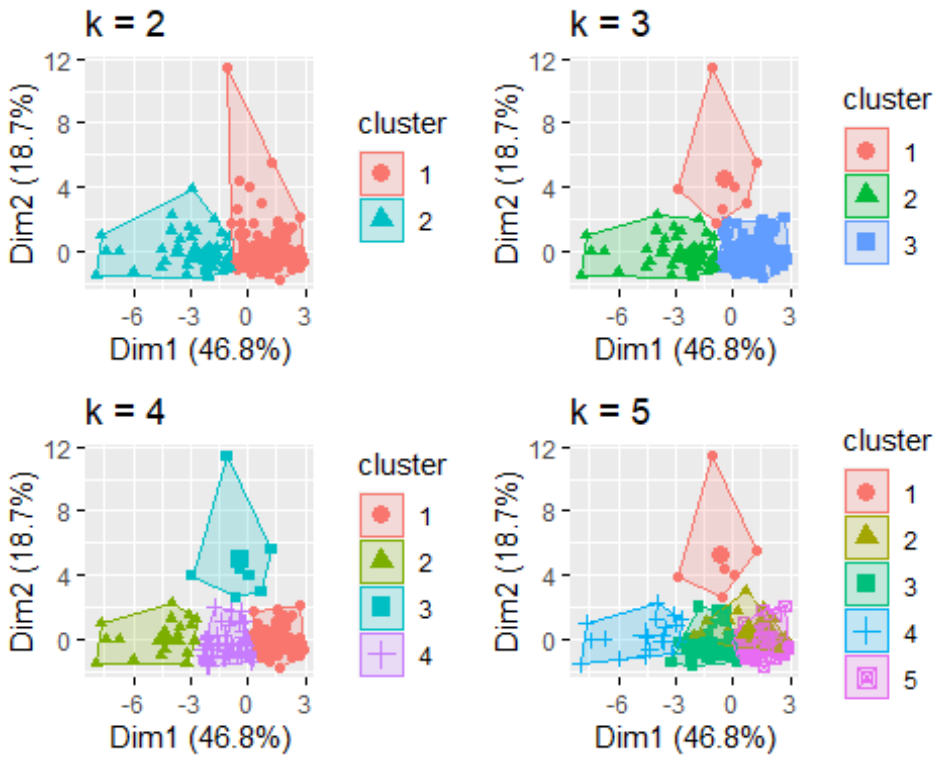
```
## ✖ stringr::fixed()     masks recipes::fixed()
## ✖ plyr::id()           masks dplyr::id()
## ✖ dplyr::lag()         masks stats::lag()
## ✖ purrr::lift()        masks caret::lift()
## ✖ plyr::mutate()       masks dplyr::mutate()
## ✖ purrr::partial()     masks pdp::partial()
## ✖ plyr::rename()       masks dplyr::rename()
## ✖ plyr::summarise()    masks dplyr::summarise()
## ✖ plyr::summarize()    masks dplyr::summarize()

library(cluster)     # for general clustering algorithms
library(factoextra)  # for visualizing cluster results

## Welcome! Want to learn more? See two factoextra-related books at https://g
oo.gl/ve3WBa

#K-means

#start at 2 clusters
k2 <- kmeans(newdf1, centers = 2, nstart = 25)
str(k2)

## List of 9
##  $ cluster     : Named int [1:197] 1 1 1 1 1 1 2 1 1 1 ...
##   ..- attr(*, "names")= chr [1:197] "1" "2" "3" "4" ...
##  $ centers     : num [1:2, 1:10] -0.0408 0.0978 -0.0208 0.0498 -0.5028 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "1" "2"
##   .. ..$ : chr [1:10] "Entropy_cooc.W.ADC" "GLNU_align.H.PET" "Min_hist.PE
T" "Max_hist.PET" ...
##  $ totss       : num 1960
##  $ withinss    : num [1:2] 864 475
##  $ tot.withinss: num 1339
##  $ betweenss   : num 621
##  $ size        : int [1:2] 139 58
##  $ iter        : int 1
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"

#plot the 2 clusters
fviz_cluster(k2, data = newdf1)
```



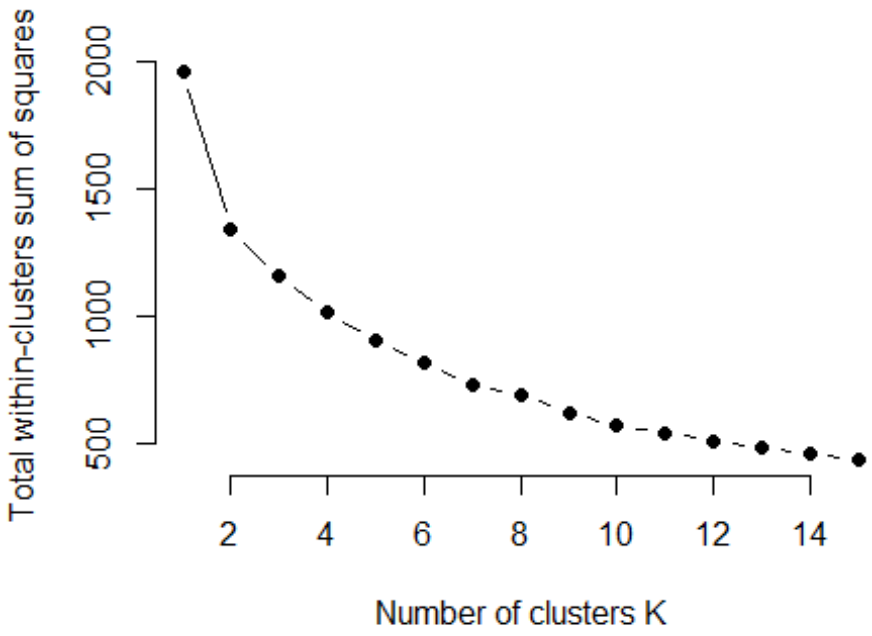

```
 #get the each cluster's data
df %>%
  as_tibble() %>%
  mutate(cluster = k2$cluster,
         d = row.names(df)) %>%
  ggplot(aes(Failure, Failure.binary, color = factor(cluster), label = d)) +
  geom_text()
```



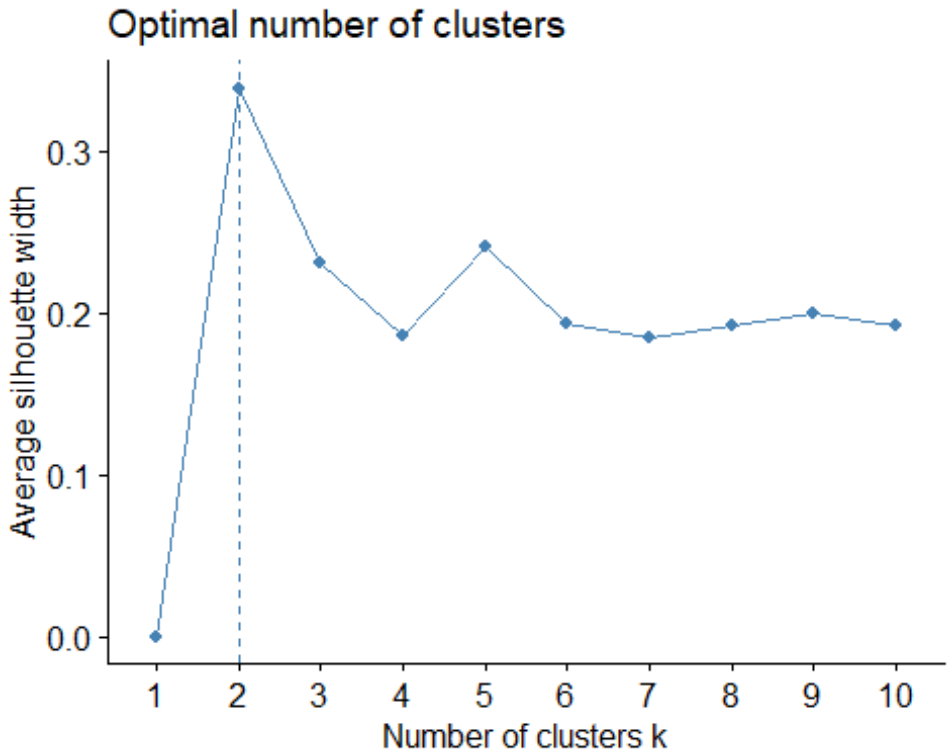

```
k3 <- kmeans(newdf1, centers = 3, nstart = 25)
k4 <- kmeans(newdf1, centers = 4, nstart = 25)
k5 <- kmeans(newdf1, centers = 5, nstart = 25)

# plots to compare
p1 <- fviz_cluster(k2, geom = "point", data = newdf1) + ggtitle("k = 2")
p2 <- fviz_cluster(k3, geom = "point",  data = newdf1) + ggtitle("k = 3")
p3 <- fviz_cluster(k4, geom = "point",  data = newdf1) + ggtitle("k = 4")
p4 <- fviz_cluster(k5, geom = "point",  data = newdf1) + ggtitle("k = 5")

grid.arrange(p1, p2, p3, p4, nrow = 2)
```




```
#Determining Optimal Number of Clusters
set.seed(123)

#function to compute total within-cluster sum of square
wss <- function(k) {
  kmeans(newdf1, k, nstart = 10 )$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)

plot(k.values, wss_values,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")
```



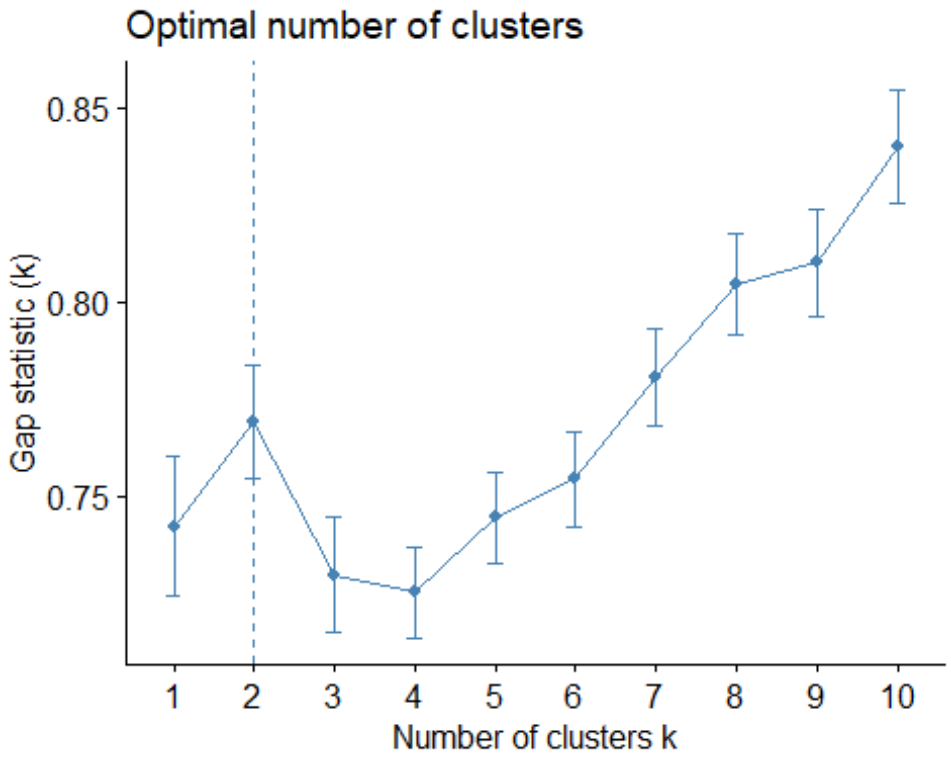

```
#or use this
fviz_nbclust(newdf1, kmeans, method = "silhouette")
```



```r
# compute gap statistic
set.seed(123)
gap_stat <- clusGap(newdf1, FUN = kmeans, nstart = 25,
                    K.max = 10, B = 50)

# Print the result
print(gap_stat, method = "firstmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = newdf1, FUNcluster = kmeans, K.max = 10, B = 50, nstart = 25)
## B=50 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstmax'): 2
##          logW   E.logW       gap     SE.sim
##  [1,] 5.277491 6.019719 0.7422279 0.01801678
##  [2,] 5.083916 5.853084 0.7691685 0.01451192
##  [3,] 5.029109 5.758704 0.7295948 0.01478736
##  [4,] 4.964879 5.690045 0.7251655 0.01169000
##  [5,] 4.904257 5.648689 0.7444321 0.01177528
##  [6,] 4.858554 5.612890 0.7543364 0.01223561
##  [7,] 4.800585 5.581224 0.7806391 0.01230034
##  [8,] 4.747935 5.552359 0.8044238 0.01306701
##  [9,] 4.715484 5.525663 0.8101794 0.01381194
## [10,] 4.661080 5.501048 0.8399675 0.01459808
```

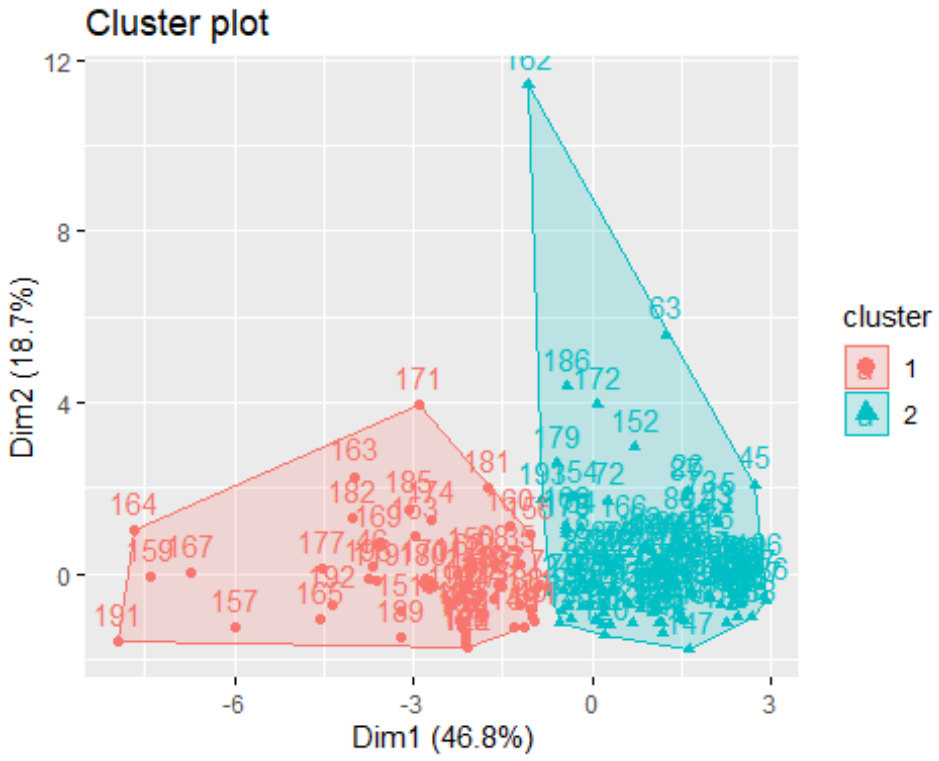
```r
fviz_gap_stat(gap_stat)
```



```r
# Compute k-means clustering with k = 2
set.seed(123)
final <- kmeans(newdf1, 2, nstart = 25)
print(final)
```

```
## K-means clustering with 2 clusters of sizes 58, 139
##
## Cluster means:
##   Entropy_cooc.W.ADC GLNU_align.H.PET Min_hist.PET Max_hist.PET Mean_hist.
PET
## 1         0.09777282       0.04981199    1.2049724    1.2260973     1.2588
047
## 2        -0.04079729      -0.02078486   -0.5027942   -0.5116089    -0.5252
566
##   Variance_hist.PET Standard_Deviation_hist.PET Skewness_hist.PET
## 1         1.1920516                   1.2500207        0.06915627
## 2        -0.4974028                  -0.5215914       -0.02885657
##   Kurtosis_hist.PET Energy_hist.PET
## 1       -0.13185460      0.05189485
## 2        0.05501846     -0.02165397
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  1
9  20
##   2   2   2   2   2   2   1   2   2   2   2   2   2   2   1   2   2   2
2   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  3
9  40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   1   2   2   2
2   1
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  5
9  60
##   1   2   2   2   2   1   2   2   2   2   2   2   2   2   2   2   2   1
2   1
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  7
9  80
##   2   2   2   2   1   2   2   1   2   2   2   2   2   2   2   2   2   2
2   2
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  9
9 100
##   2   2   2   2   2   2   2   2   2   2   2   1   2   2   2   2   2   2
2   2
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120
##   2   1   1   2   1   2   1   2   2   2   1   1   1   2   2   2   2   2
1   2
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 13
9 140
##   2   1   2   2   2   2   2   2   2   2   2   1   1   2   1   2   1   2
2   2
```

```
## 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 15
9 160
##   1   1   1   2   2   2   2   1   1   1   1   2   1   2   2   1   1   2
1   1
## 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 17
9 180
##   2   2   1   1   1   2   1   2   1   1   1   2   2   1   2   2   1   2
2   1
## 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
##   1   1   2   2   1   2   1   1   1   1   1   1   2   2   1   1   1
##
## Within cluster sum of squares by cluster:
## [1] 474.5285 864.0037
##  (between_SS / total_SS =  31.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withi
nss"
## [6] "betweenss"    "size"         "iter"         "ifault"

#final data
fviz_cluster(final, data = newdf1)
```



Hierarchical

```
library(cluster)     # for general clustering algorithms
library(factoextra)  # for visualizing cluster results
```

```r
# Dissimilarity matrix
d <- dist(newdf1, method = "euclidean")

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

# For reproducibility
set.seed(123)

# Compute maximum or complete linkage clustering with agnes
hc2 <- agnes(newdf1, method = "complete")

# Agglomerative coefficient
hc2$ac
```

```
## [1] 0.9185821
```

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

# function to compute coefficient
ac <- function(x) {
  agnes(newdf1, method = x)$ac
}

# get agglomerative coefficient for each linkage method
purrr::map_dbl(m, ac)
```

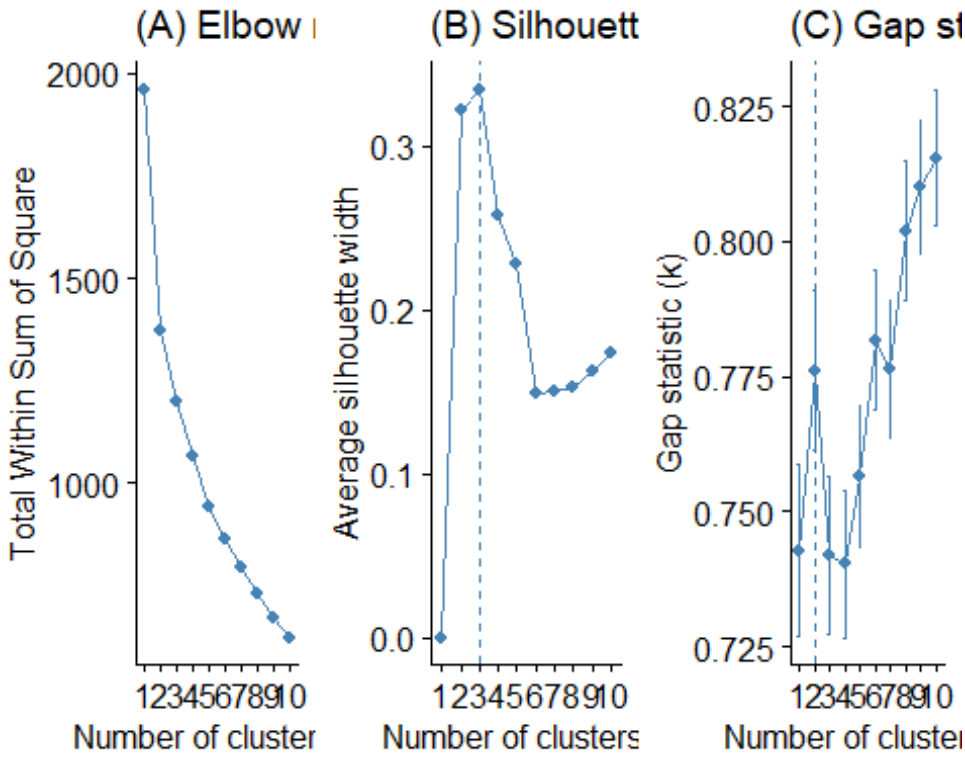
```
##   average    single  complete      ward
## 0.9037837 0.8548549 0.9185821 0.9637063
```

```r
# compute divisive hierarchical clustering
hc4 <- diana(df)

# Divise coefficient; amount of clustering structure found
hc4$dc
```

```
## [1] 0.9936897
```

```r
# Plot cluster results
p1 <- fviz_nbclust(newdf1, FUN = hcut, method = "wss",
                   k.max = 10) +
  ggtitle("(A) Elbow method")
p2 <- fviz_nbclust(newdf1, FUN = hcut, method = "silhouette",
                   k.max = 10) +
  ggtitle("(B) Silhouette method")
p3 <- fviz_nbclust(newdf1, FUN = hcut, method = "gap_stat",
                   k.max = 10) +
  ggtitle("(C) Gap statistic")
```

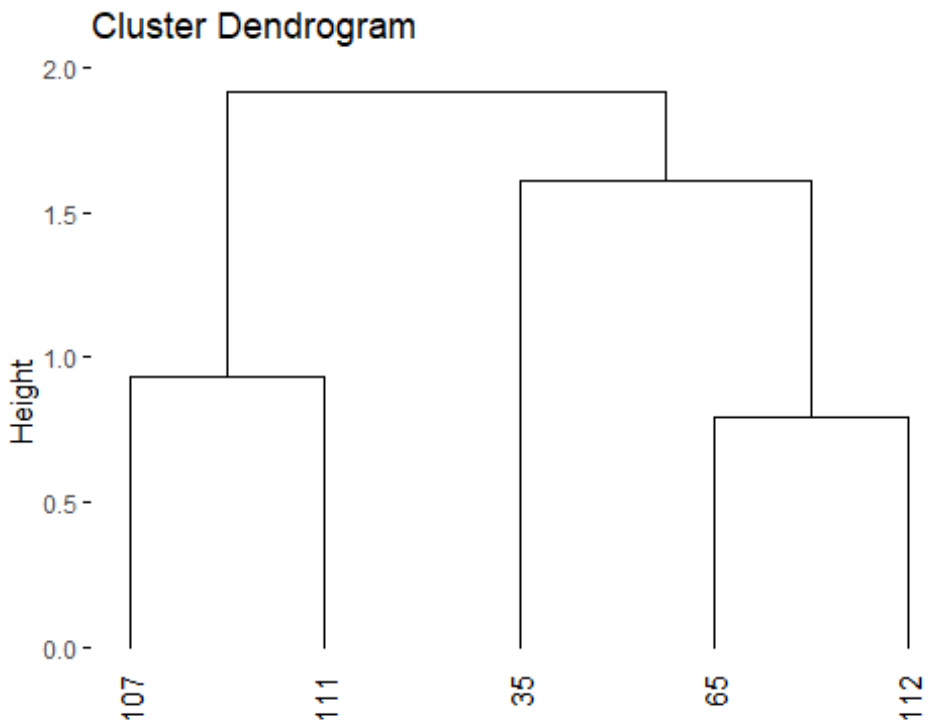
```r
# Display plots side by side
gridExtra::grid.arrange(p1, p2, p3, nrow = 1)
```




```r
# Construct dendorgram
hc5 <- hclust(d, method = "ward.D2" )
dend_plot <- fviz_dend(hc5)
```

```
## Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none
" instead as
## of ggplot2 3.3.4.
## ℹ The deprecated feature was likely used in the factoextra package.
##   Please report the issue at <]8;;https://github.com/kassambara/factoextra
/issueshttps://github.com/kassambara/factoextra/issues]8;;>.
```

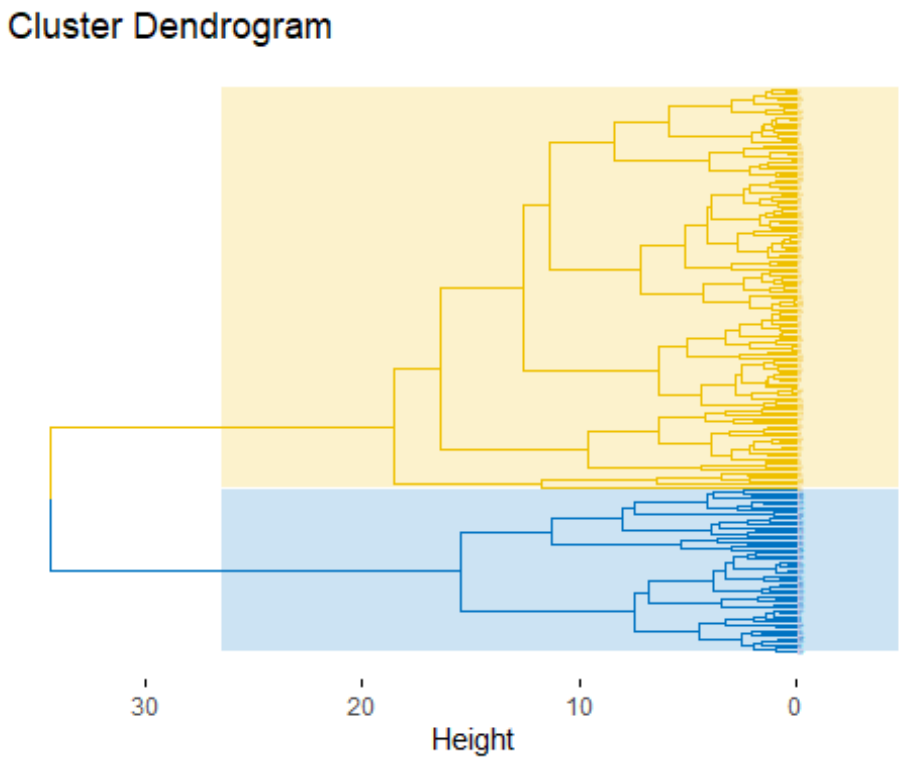
```r
dend_data <- attr(dend_plot, "dendrogram")
dend_cuts <- cut(dend_data, h = 2)
fviz_dend(dend_cuts$lower[[1]])
```




```
# Ward's method
hc5 <- hclust(d, method = "ward.D2" )

# Cut tree into 4 groups
sub_grp <- cutree(hc5, k = 2)

# Number of members in each cluster
table(sub_grp)
```

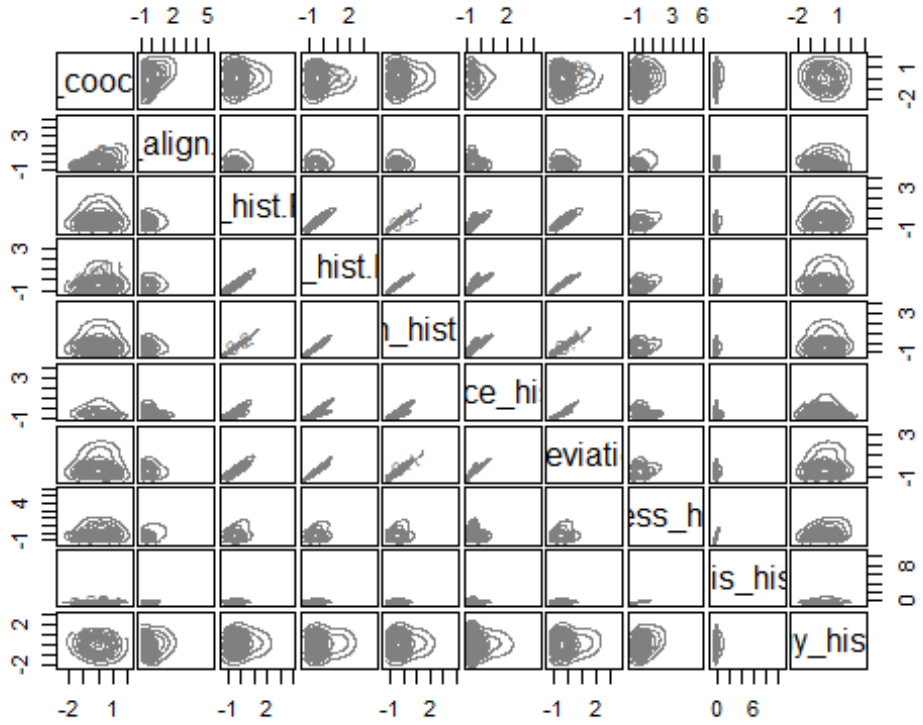
```
## sub_grp
##   1   2
## 140  57
```

```
# Plot full dendogram
fviz_dend(
  hc5,
  k = 2,
  horiz = TRUE,
  rect = TRUE,
  rect_fill = TRUE,
  rect_border = "jco",
  k_colors = "jco",
  cex = 0.1
)
```

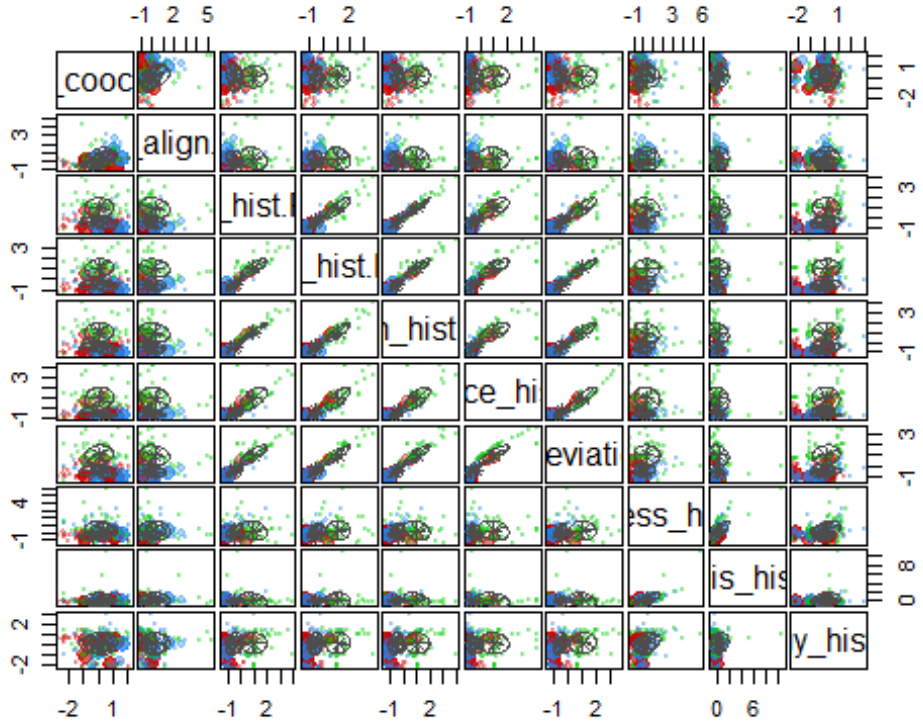


Model Based

```
# Modeling packages
library(mclust)   # for fitting clustering algorithms

## Package 'mclust' version 6.0.0
## Type 'citation("mclust")' for citing this R package in publications.

##
## Attaching package: 'mclust'

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

#Model Based

# Apply GMM model with 3 components
radio_mc <- Mclust(newdf1, G = 3)

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




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



```r
# Observations with high uncertainty
sort(radio_mc$uncertainty, decreasing = TRUE) %>% head()
```

```
##        29       125       140        37       138        16
## 0.3607057 0.2761841 0.2622943 0.2601872 0.1920009 0.1906699
```

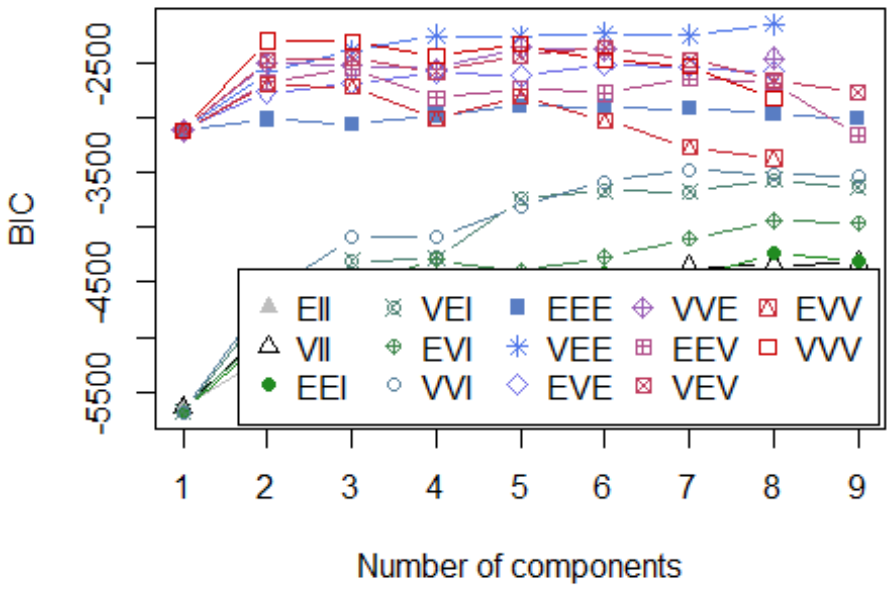
```r
summary(radio_mc)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model wit
h 3
## components:
##
##  log-likelihood   n  df       BIC       ICL
##        -636.189 197 197 -2313.169 -2319.813
##
## Clustering table:
##  1  2  3
## 54 80 63
```

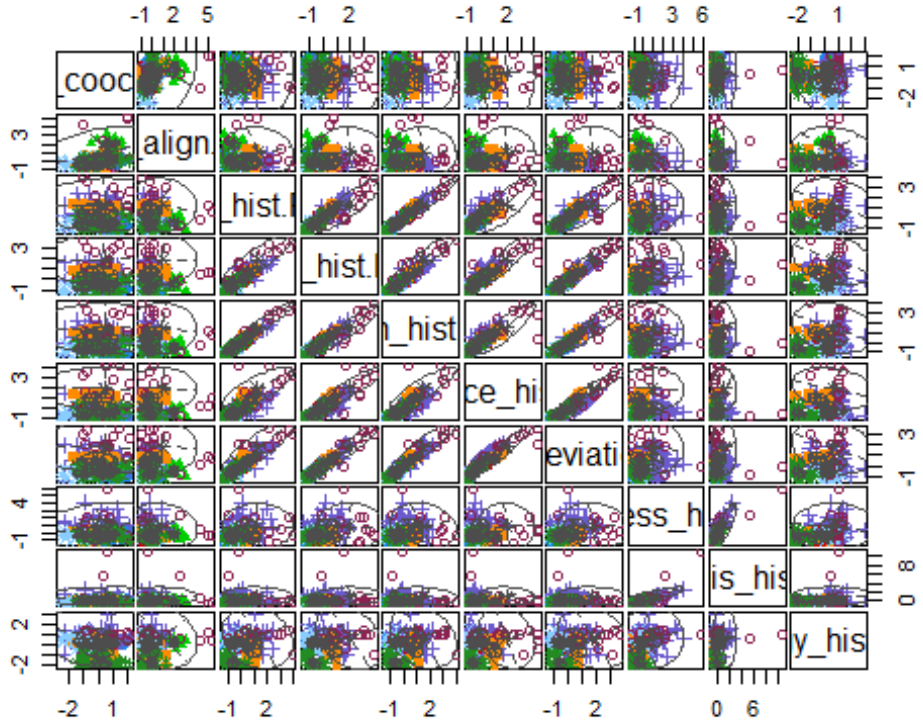
```r
radio_optimal_mc <- Mclust(newdf1)

summary(radio_optimal_mc)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEE (ellipsoidal, equal shape and orientation) model with 8 compone
nts:
##
##  log-likelihood   n  df       BIC       ICL
##       -684.3928 197 149 -2155.983 -2169.882
##
## Clustering table:
##  1  2  3  4  5  6  7  8
## 32 18 11 47 25 29 15 20
```

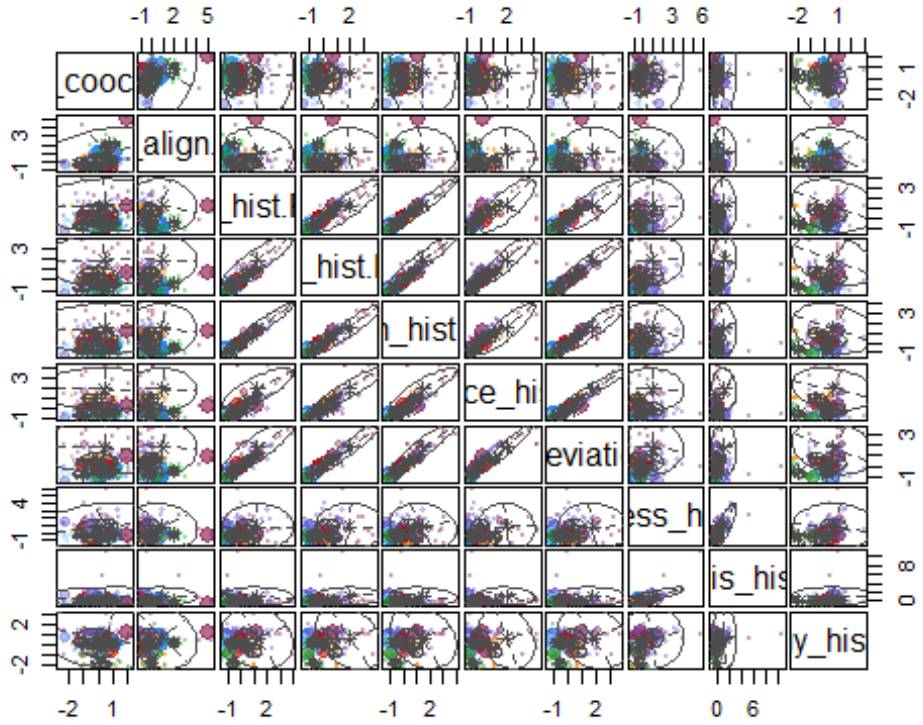
```r
legend_args <- list(x = "bottomright", ncol = 5)
plot(radio_optimal_mc, what = 'BIC', legendArgs = legend_args)
```



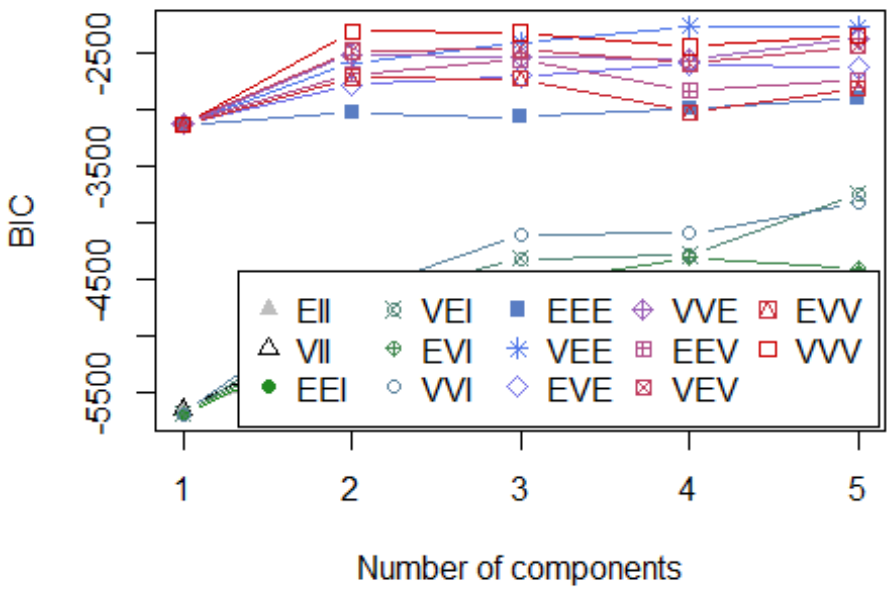

```
plot(radio_optimal_mc, what = 'classification')
```



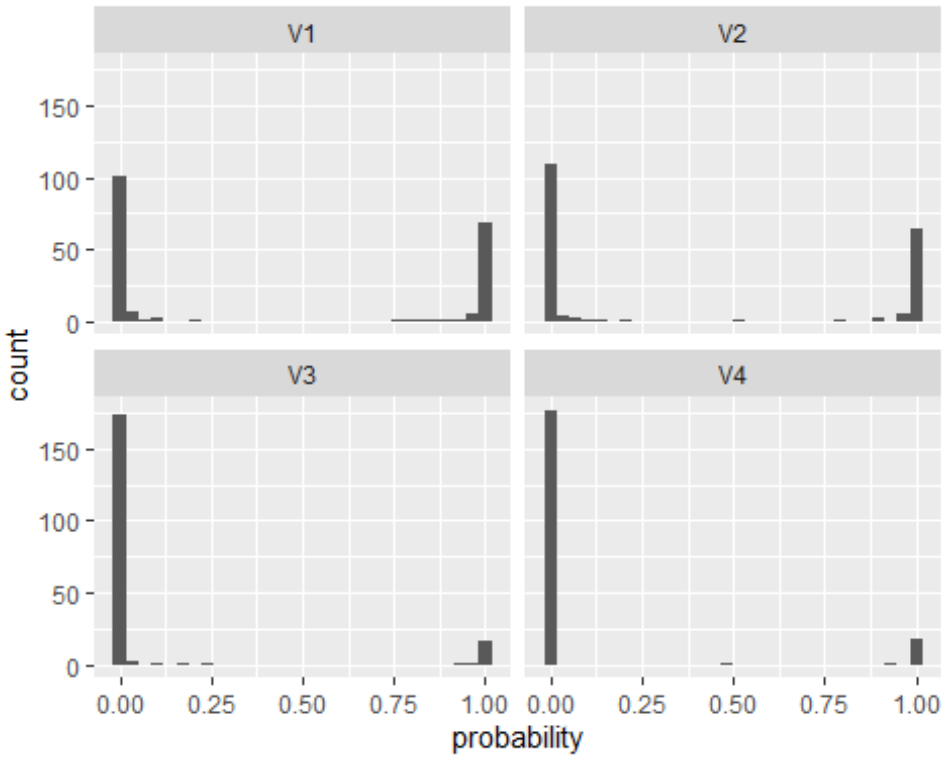

```
plot(radio_optimal_mc, what = 'uncertainty')
```



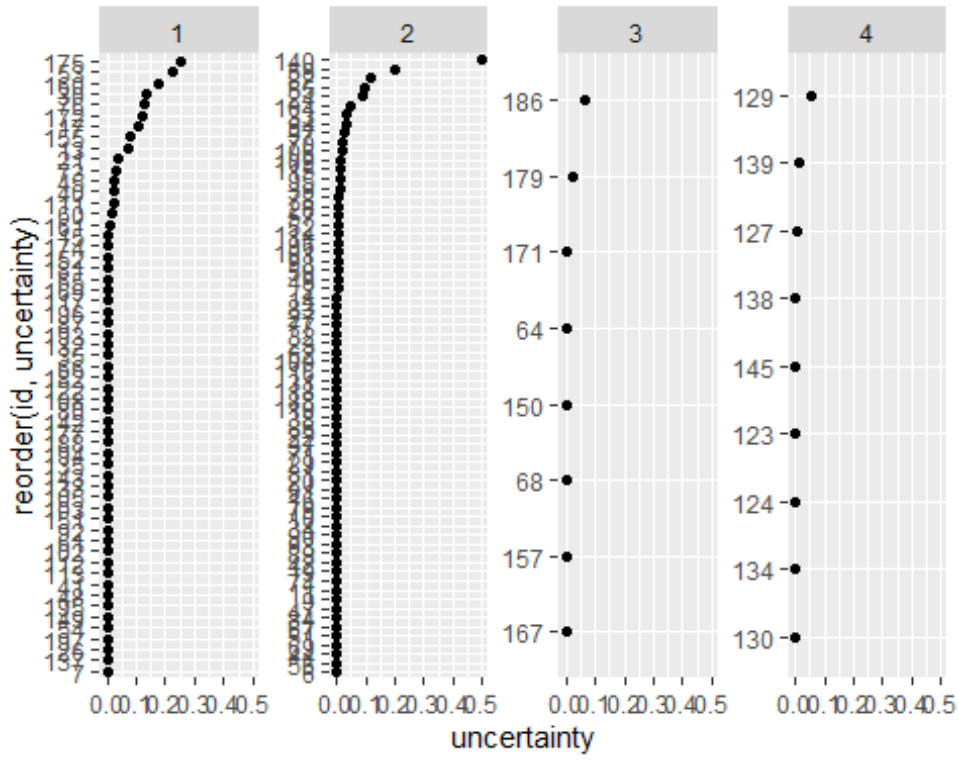

```
df_mc <- Mclust(newdf1, 1:5)

summary(df_mc)

## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEE (ellipsoidal, equal shape and orientation) model with 4 compone
nts:
##
##  log-likelihood   n  df       BIC       ICL
##       -862.8837 197 101 -2259.371 -2266.032
##
## Clustering table:
##  1  2  3  4
## 84 76 18 19

plot(df_mc, what = 'BIC',
     legendArgs = list(x = "bottomright", ncol = 5))
```



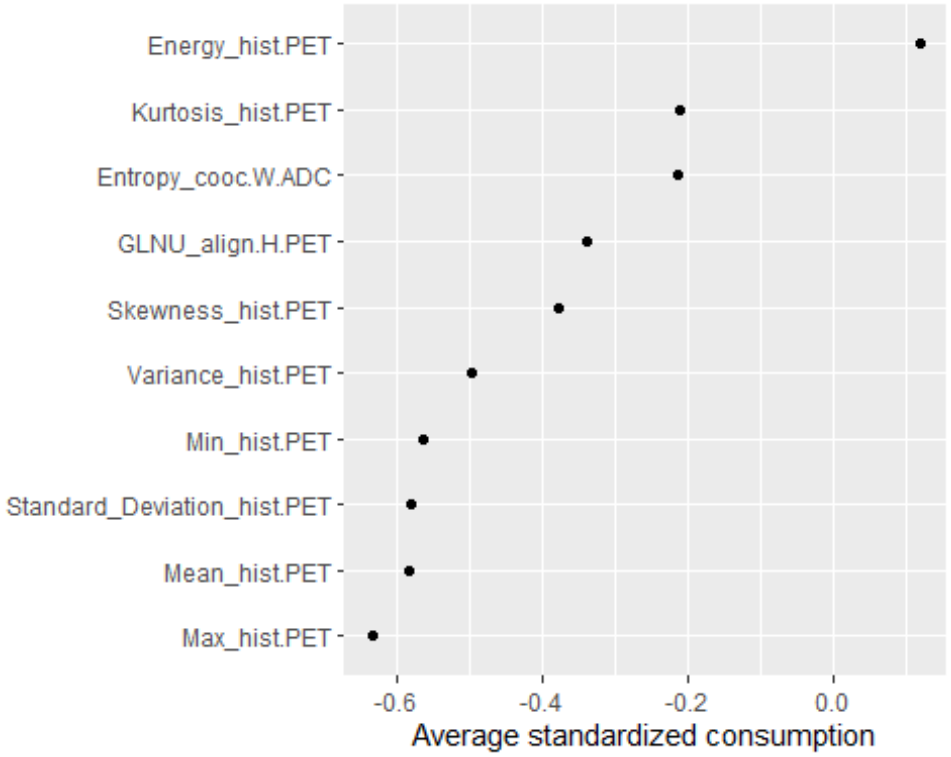

```
probabilities <- df_mc$z

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

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

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




```
uncertainty <- data.frame(
  id = 1:nrow(newdf1),
  cluster = df_mc$classification,
  uncertainty = df_mc$uncertainty
)

uncertainty %>%
  group_by(cluster) %>%
  filter(uncertainty > 0.0001) %>%
  ggplot(aes(uncertainty, reorder(id, uncertainty))) +
  geom_point() +
  facet_wrap(~ cluster, scales = 'free_y', nrow = 1)
```




```
cluster2 <- newdf1 %>%
  scale() %>%
  as.data.frame() %>%
  mutate(cluster = df_mc$classification) %>%
  filter(cluster == 2) %>%
  select(-cluster)

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


Energy_hist.PET
Kurtosis_hist.PET
Entropy_cooc.W.ADC
GLNU_align.H.PET
Skewness_hist.PET
Variance_hist.PET
Min_hist.PET
Standard_Deviation_hist.PET
Mean_hist.PET
Max_hist.PET
-0.6
-0.4
-0.2
0.0
Average standardized consumption