clustering

Nota: COMPROBAR QUE SIDE_EFFECTS_INVERSE SE HA AÑADIDO CORRECTAMENTE

```
library(tm)
```

```
## Loading required package: NLP
```

```
library(factoextra)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:NLP':
##
##     annotate
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```
library(proxy)
```

```
##
## Attaching package: 'proxy'
```

```
## The following objects are masked from 'package:stats':
##
##     as.dist, dist
```

```
## The following object is masked from 'package:base':
##
##     as.matrix
```

```
library(ggpubr)
```

```
## Loading required package: magrittr
```

```
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Versión 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Escriba 'rattle()' para agitar, sacudir y  rotar sus datos.
```

```
library("randomForest")
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:rattle':
##
##     importance
```

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

```
##     margin
```

```
library(ppclust)
library(cluster)
library(fclust)
require("igraph")
```

```
## Loading required package: igraph
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

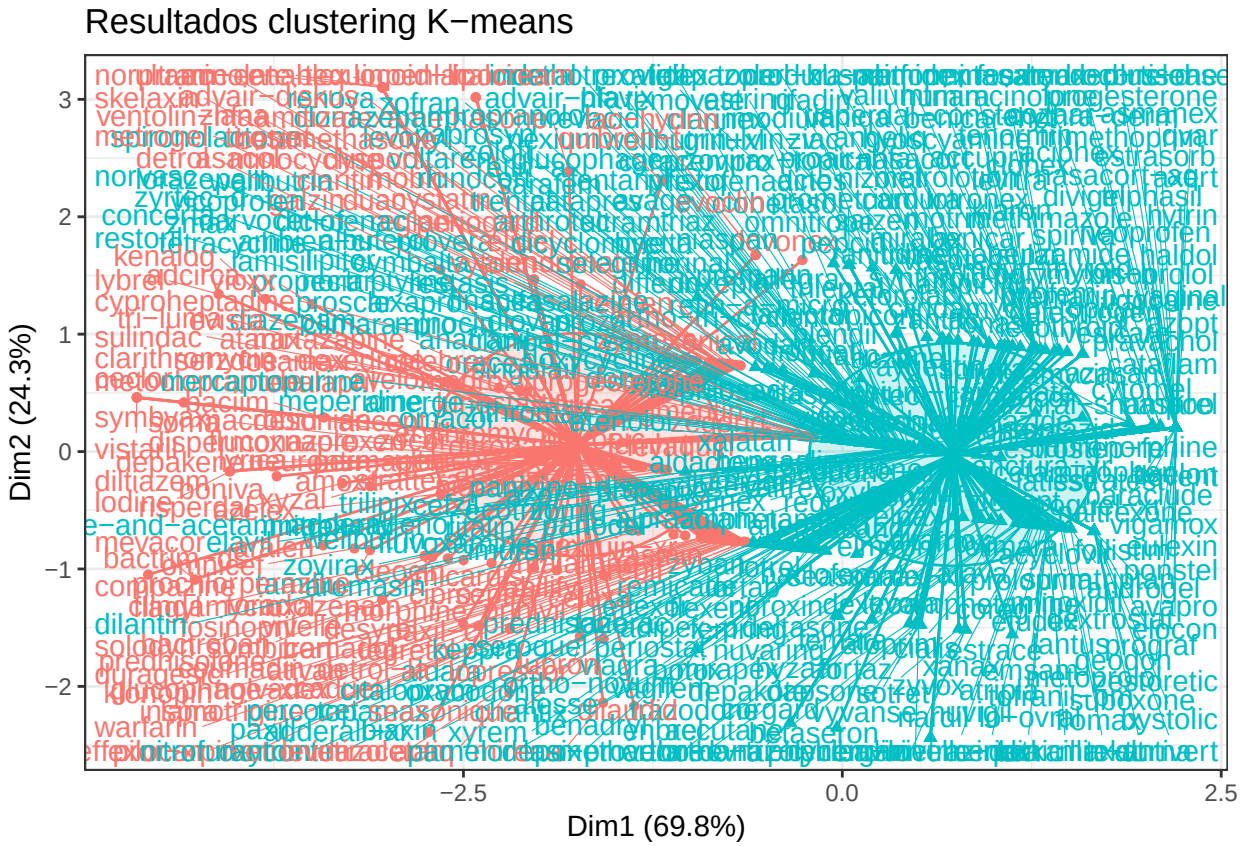
```
## The following object is masked from 'package:base':
##
##     union
```

```
# Establecemos una semilla para obtener siempre los mismos resultados
set.seed(3)

# Cargamos los tados
datos_train <- read.table("datos_train_preprocesado.csv", sep=",", comment.char="",quote = "\"", header=
datos_test <- read.table("datos_test_preprocesado.csv", sep=",", comment.char="",quote = "\"", header=TR

# Nos quedamos con las columnas que nos interesan
# Las columnas que tenemos son: RATING, SIDE_EFFECTS_INVERSE, EFFECTIVENESS_NUMBER
# Nos quedamos con las columnas que nos interesan
datos_train2 = datos_train[c(2,12,9)]
datos_test2 = datos_test[c(2,12,9)]

## Arreglamos el conjunto de datos para poder trabajar con lo que nos interesa
# https://stackoverflow.com/questions/40003028/extracting-unique-values-from-data-frame-using-r
# LO HACEMOS TANTO PARA TRAIN COMO PARA TEST

nombres_farmacos_train <- unique(datos_train[,1])
nombres_farmacos_test <- unique(datos_test[,1])
#print(nombres_farmacos_test[1:10])
#print(nombres_farmacos_train[1:10])

# Creamos una matriz con tantas filas como fármacos, y columnas como datos queramos utilizar.
#En este caso son 3 columnas porque necesitamos guardar la info de "rating", "sideEffectsInverse" y "ef
datos_procesados_train <- matrix(ncol=3, nrow=length(nombres_farmacos_train))
datos_procesados_test <- matrix(ncol=3, nrow=length(nombres_farmacos_test))

# Primero procesamos TRAIN
df_farmacos_train = as.data.frame(nombres_farmacos_train)
for(i in 0:length(nombres_farmacos_train)){
  #https://stackoverflow.com/questions/24831580/return-row-of-data-frame-based-on-value-in-a-column-r
  filas_farmaco_train <- datos_train2[which(datos_train$urlDrugName == nombres_farmacos_train[i]),]

  mean_rating_train <- mean(filas_farmaco_train$rating)
  mean_side_effect_train <- mean(filas_farmaco_train$sideEffectsInverse)
  mean_effectiveness_train <- mean(filas_farmaco_train$effectivenessNumber)
```

```
  datos_procesados_train[i,] <- c(mean_rating_train, round(mean_side_effect_train), round(mean_effectiv
}

# Procesamos TEST
# Necesario para poder hacer predict posteriormente
df_farmacos_test = as.data.frame(nombres_farmacos_test)
for(i in 0:length(nombres_farmacos_test)){
  #https://stackoverflow.com/questions/24831580/return-row-of-data-frame-based-on-value-in-a-column-r
  filas_farmaco_test <- datos_test2[which(datos_test$urlDrugName == nombres_farmacos_test[i]),]

  mean_rating_test <- mean(filas_farmaco_test$rating)
  mean_side_effect_test <- mean(filas_farmaco_test$sideEffectsInverse)
  mean_effectiveness_test <- mean(filas_farmaco_test$effectivenessNumber)

  datos_procesados_test[i,] <- c(mean_rating_test, round(mean_side_effect_test), round(mean_effectivenes
}

data_train_procesado <- data.frame(datos_procesados_train)
data_test_procesado <- data.frame(datos_procesados_test)
rownames(data_train_procesado) <- nombres_farmacos_train
rownames(data_test_procesado) <- nombres_farmacos_test
colnames(data_train_procesado) <- c("rating", "sideEffectsInverse", "effectivenessNumber")
colnames(data_test_procesado) <- c("rating", "sideEffectsInverse", "effectivenessNumber")
#head(data_train_procesado)
#head(data_train_procesado)
```

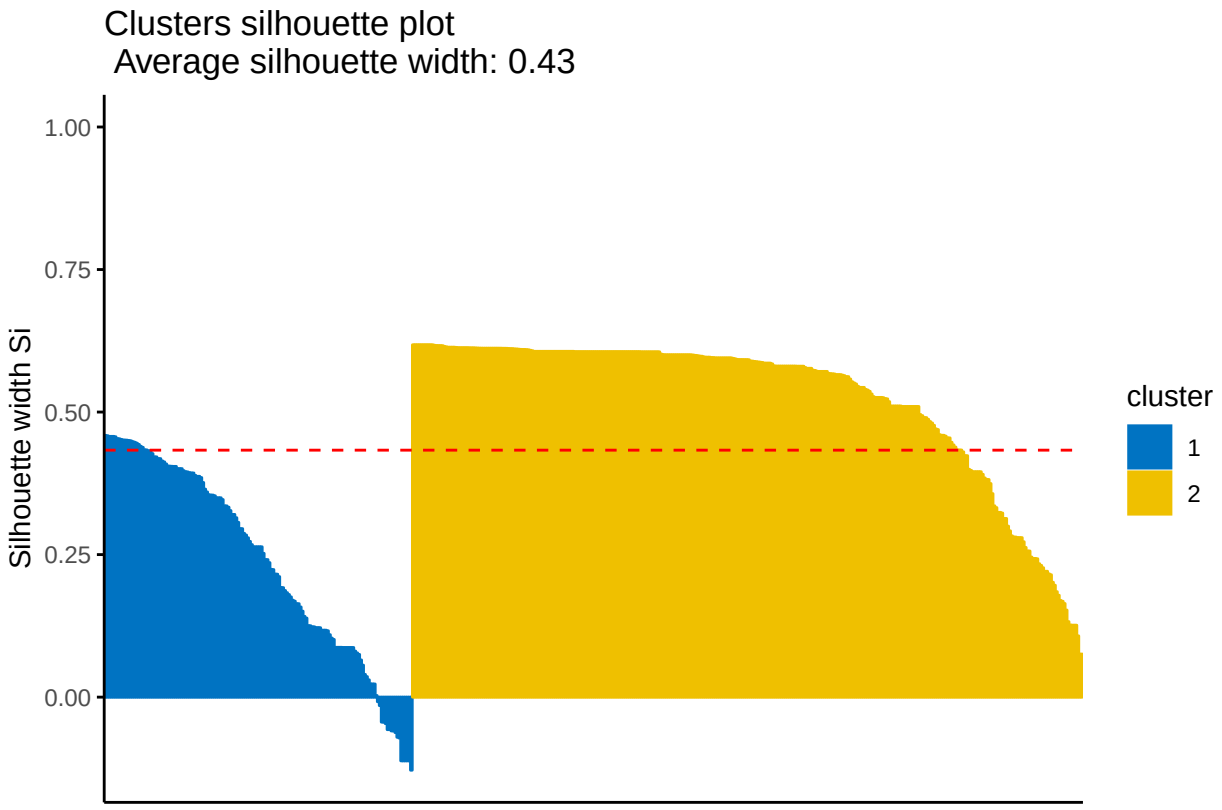
K-medias

```
res_clustering <- kmeans(data_train_procesado, 2)

# Añadir labelsize = 0 si queremos quitar los nombres
fviz_cluster(object = res_clustering, data = data_train_procesado, show.clust.cent = TRUE,
             ellipse.type = "euclid", star.plot = TRUE, repel = TRUE) +
  labs(title = "Resultados clustering K-means") +
  theme_bw() +
  theme(legend.position = "none")
```




```
datos <- scale(data_train_procesado)
km_clusters <- eclust(x = datos, FUNcluster = "kmeans", k = 2, seed = 123,
                      hc_metric = "euclidean", nstart = 50, graph = FALSE)
fviz_silhouette(sil.obj = km_clusters, print.summary = TRUE, palette = "jco",
                ggtheme = theme_classic())
```

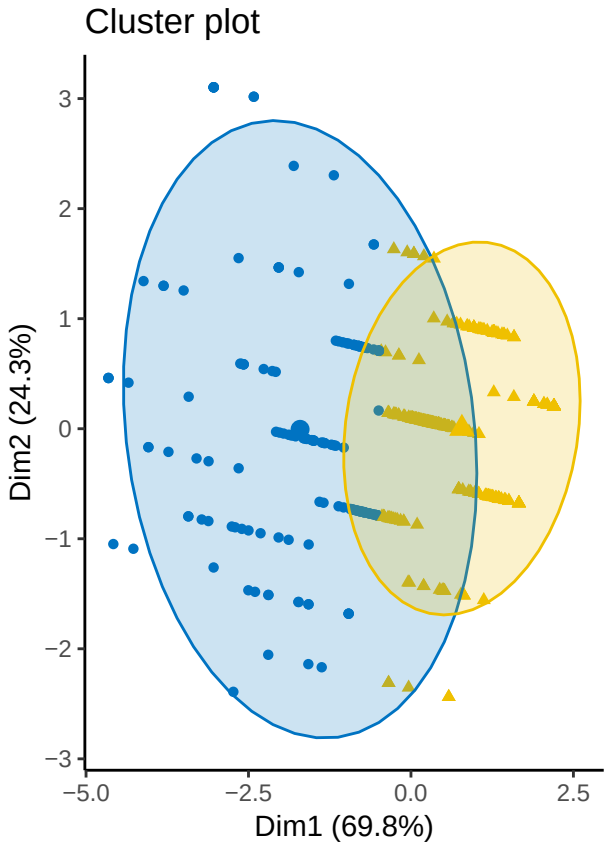
```
##   cluster size ave.sil.width
## 1       1  158          0.24
## 2       2  344          0.52
```

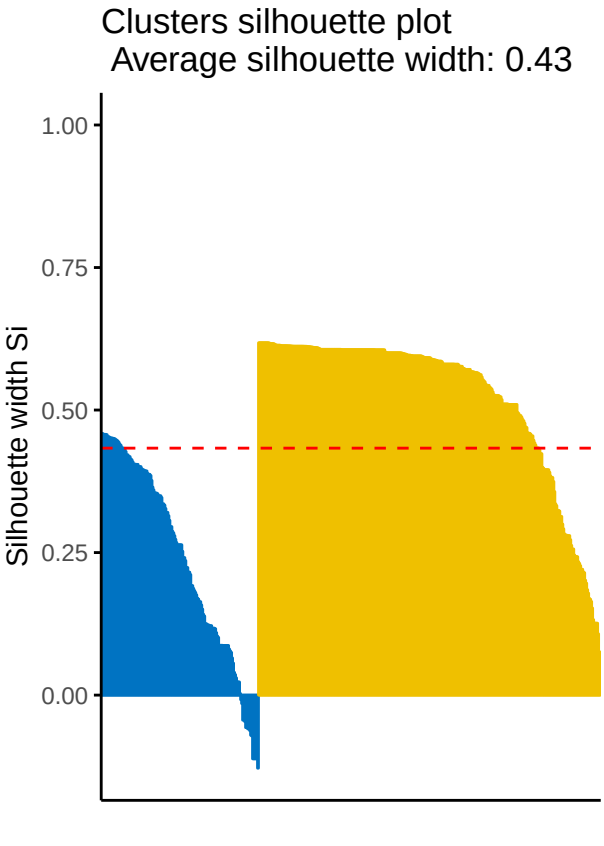



```
# Media silhouette por cluster
km_clusters$silinfo$clus.avg.widths
```

```
## [1] 0.2357250 0.5238887
```

```
km_clusters <- eclust(x = datos, FUNcluster = "kmeans", k = 2, seed = 123,
                      hc_metric = "euclidean", nstart = 50, graph = FALSE)
p1 <- fviz_cluster(object = km_clusters, geom = "point", ellipse.type  = "norm",
                   palette = "jco") +
      theme_classic() + theme(legend.position = "none")

p2 <- fviz_silhouette(sil.obj = km_clusters, print.summary = FALSE,
                      palette = "jco", ggtheme = theme_classic()) +
      theme(legend.position = "none")

ggarrange(p1, p2)
```



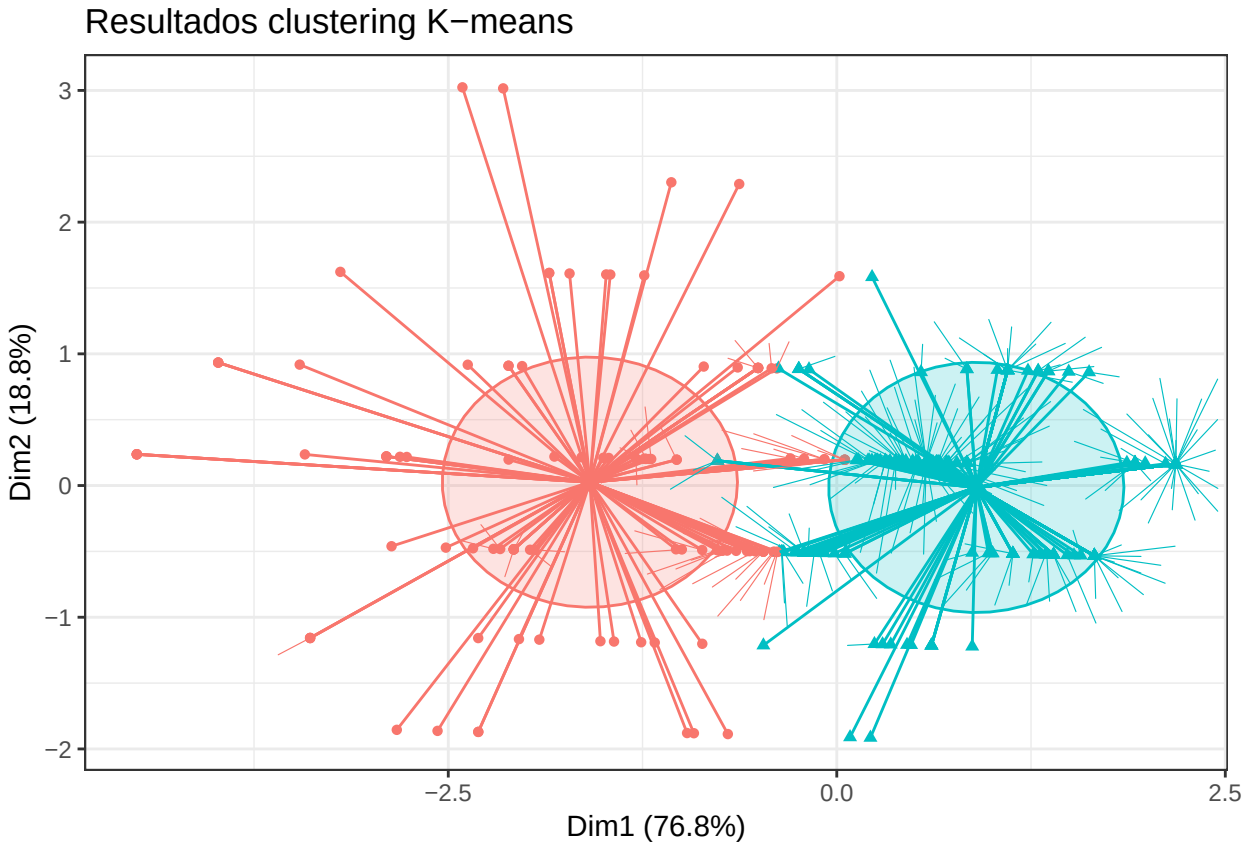


Ahora vamos a hacer predict con el clustering que nos ha generado train

```
# Hacemos predict del clustering
test_preds <- predict(res_clustering, data_test_procesado)

table(test_preds)
```

```
## test_preds
##   1   2
## 113 200
```

```
datos_test_carlos = cbind(test_preds)
rownames(datos_test_carlos) = rownames(data_test_procesado)

test_carlos = res_clustering
test_carlos$cluster = datos_test_carlos

fviz_cluster(object = test_carlos, data = data_test_procesado, show.clust.cent = TRUE,
             ellipse.type = "euclid", star.plot = TRUE, repel = TRUE, labelsize = 0) +
  labs(title = "Resultados clustering K-means") +
  theme_bw() +
  theme(legend.position = "none")
```



Función de predicción

```
predicciones = function(cluster_train, cluster_test) {
  count = 0
  count_na = 0
  x = intersect(datos_test$urlDrugName, datos_train$urlDrugName)
  for(farmaco in x){
    indice_train = match(farmaco,df_farmacos_train$nombres_farmacos_train)
    indice_test = match(farmaco,df_farmacos_test$nombres_farmacos_test)

    # imprimimos cluster que tiene en train
    c_train = res_clustering$cluster[indice_train]

    # imprimimos cluster que tiene en test
    c_test = test_carlos$cluster[indice_test]

    if ( !is.na(c_train) && !is.na(c_test) && c_train == c_test){
      count = count +1
    }
  }
  count/nrow(df_farmacos_test)
}
```

```
## Vamos a comprobar si las predicciones se realizan correctamente.

# Primero buscamos los fármacos que estén en tanto en train como en test, pq son con los que podemos ve
x = intersect(datos_test$urlDrugName, datos_train$urlDrugName)
#head(x)
```

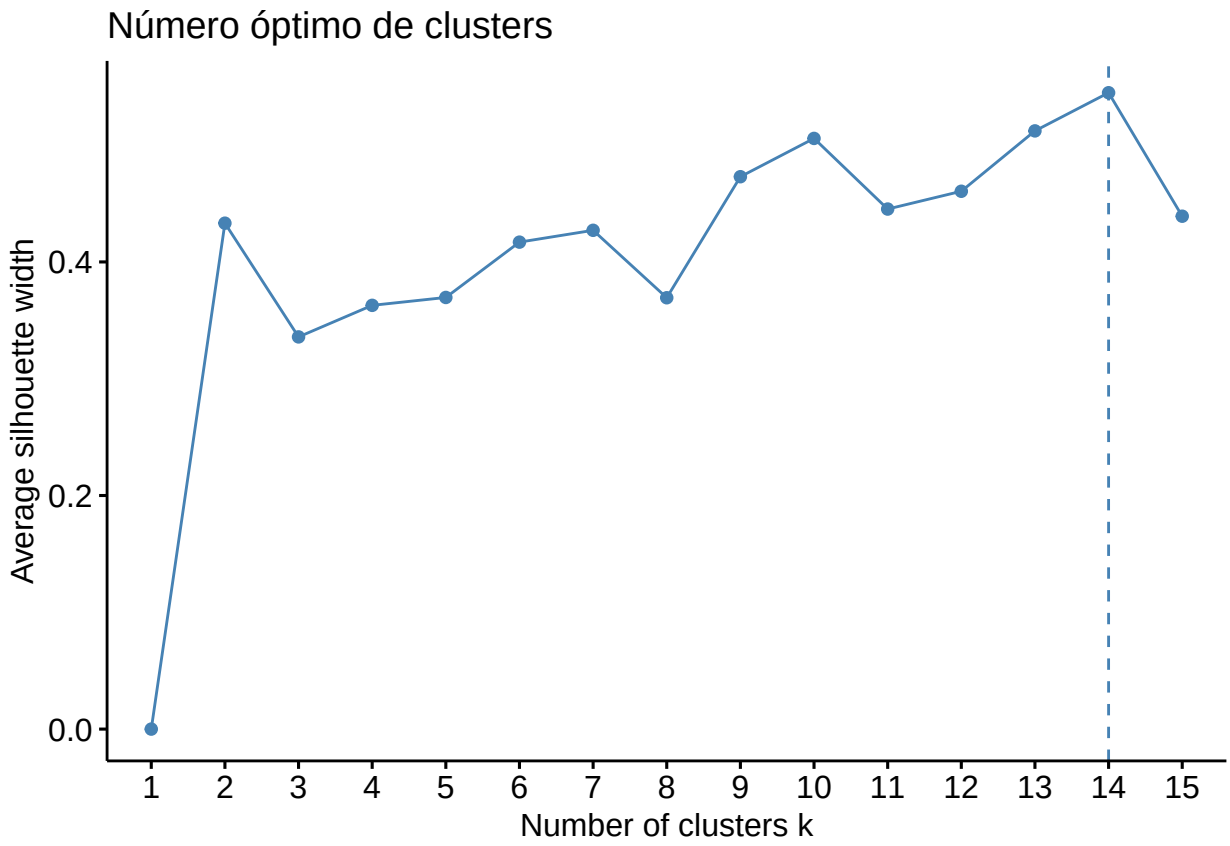
```r
# Probamos para un caso concreto
# Buscamos el indice donde esta el valor
#indice_train = match('sarafem',df_farmacos_train$nombres_farmacos_train)
#indice_test = match('sarafem',df_farmacos_test$nombres_farmacos_test)
# imprimimos cluster que tiene en train
#res_clustering$cluster[indice_train]
#test_carlos$cluster[indice_test]

# Lanzamos la función anterior
res = predicciones(res_clustering, test_carlos)
print(res)
```

```
## [1] 0.5527157
```

Obtenemos un acierto del 0.5527157.

K - MEDIOIDES CLUSTERING

```r
# datos ya lo tenemos previamente escalado (K-medias)

fviz_nbclust(x = datos, FUNcluster = kmeans, method = "silhouette", k.max = 15) +
  labs(title = "Número óptimo de clusters")
```

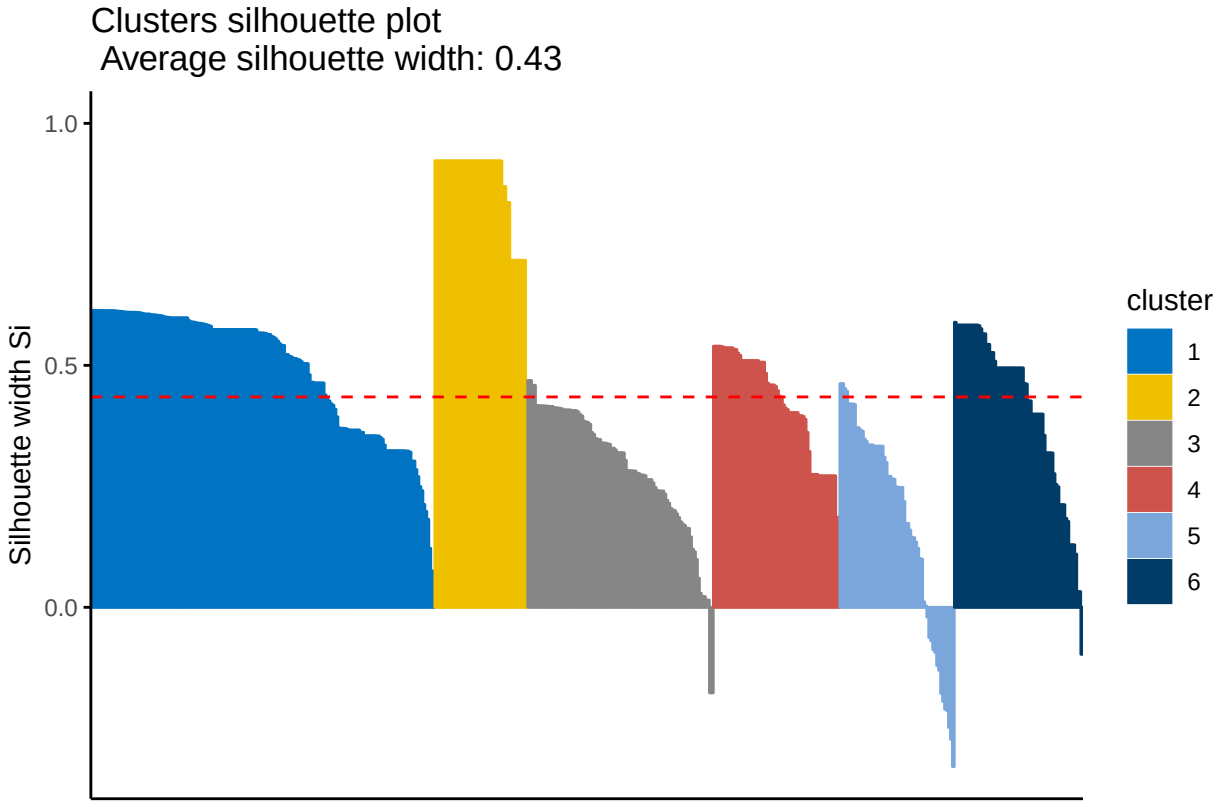


Nuevo método en el que agrupamos las observaciones en K clusters, siendo este valor K preestablecido. En este caso, cada cluster está representado por una observación presente en el cluster. Mediante este método hacemos uso de medoids, consiguiendo que sea menos afectado por ruido.

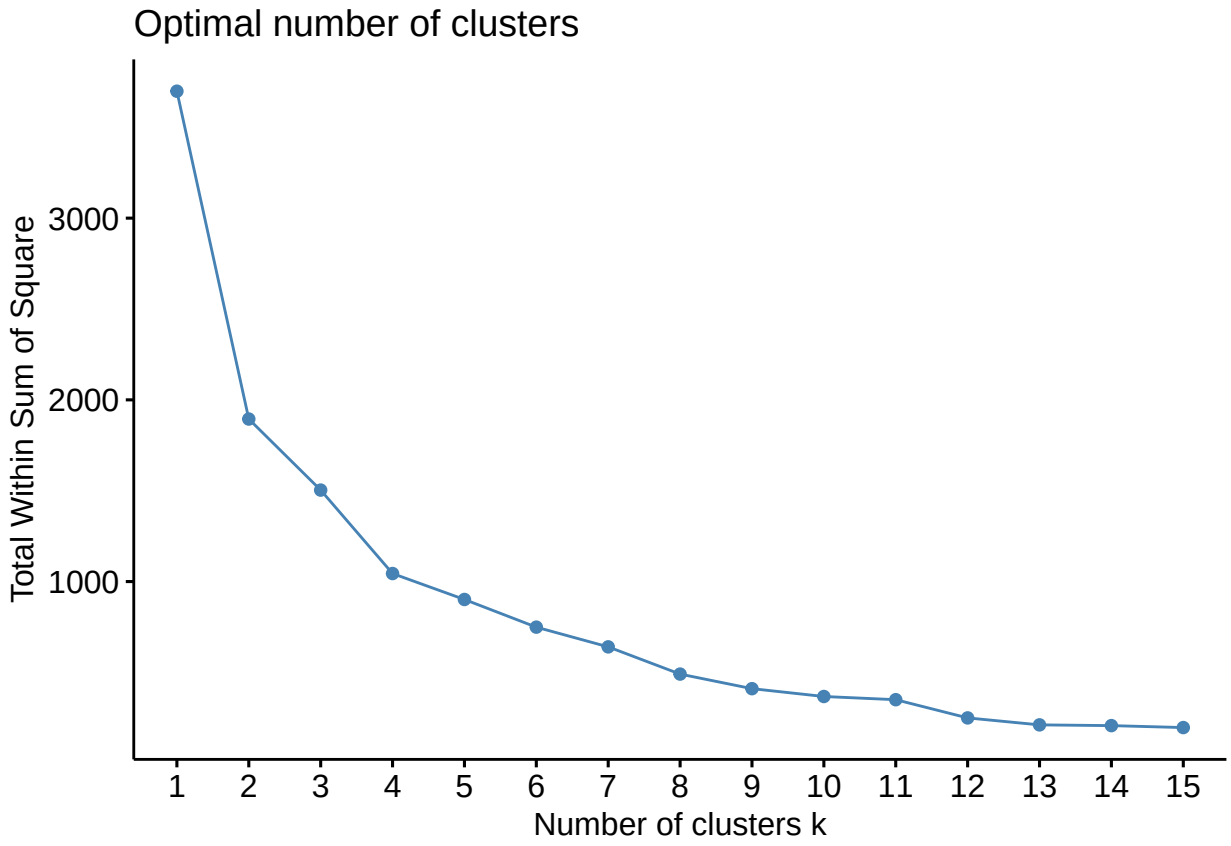
Para su resolución usaremos el alogitmo PAM (Partitioning Around Medoids). Este minimiza la suma de las diferencias de cada observación respecto a su medoid.

```
# Vemos cosas
km_clusters <- eclust(x = datos, FUNcluster = "pam", k = 6, seed = 123,
                      hc_metric = "wss",  graph = FALSE)
fviz_silhouette(sil.obj = km_clusters, print.summary = TRUE, palette = "jco",
                ggtheme = theme_classic())
```

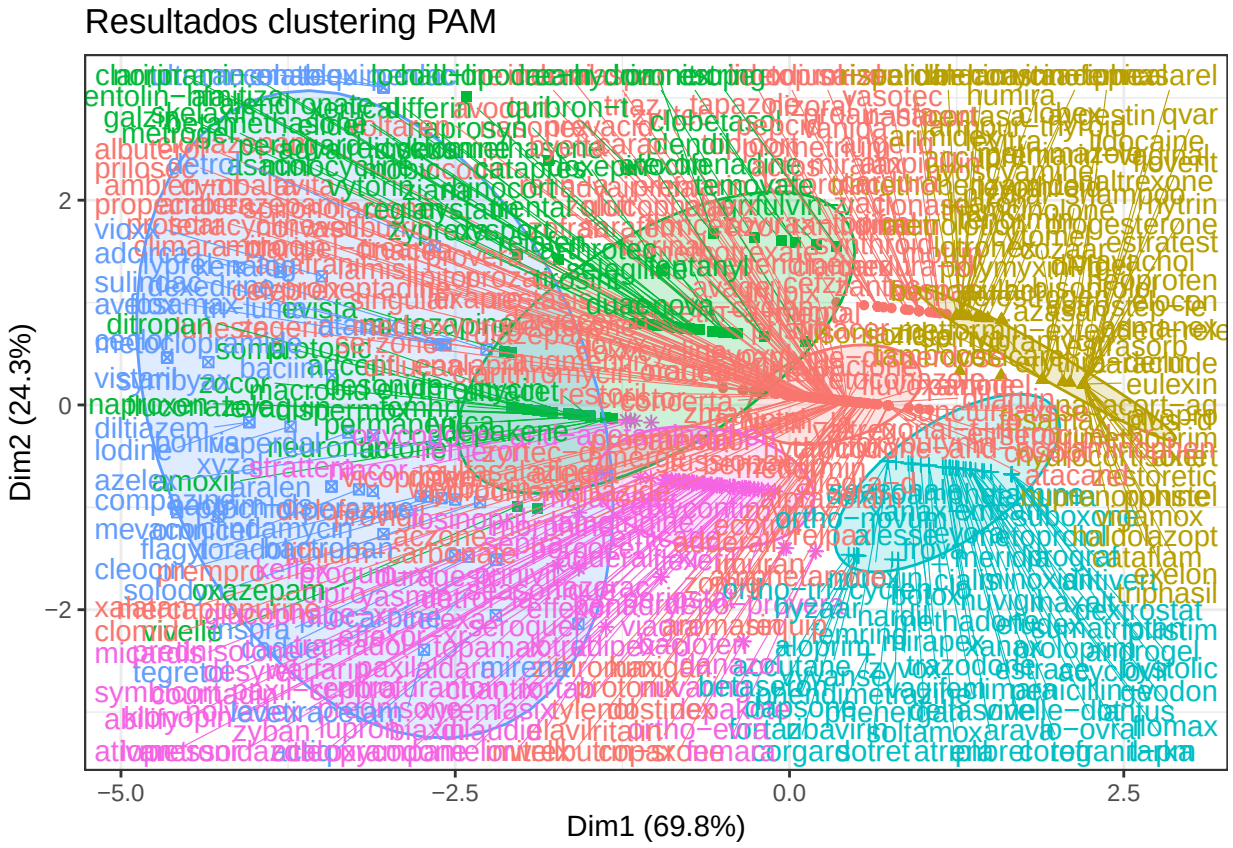
```
##   cluster size ave.sil.width
## 1       1  174          0.49
## 2       2   47          0.87
## 3       3   94          0.29
## 4       4   64          0.42
## 5       5   58          0.18
## 6       6   65          0.41
```




```
# Identificamos el número optimo de clusters
fviz_nbclust(x = datos, FUNcluster = pam, method = "wss", k.max = 15, diss = dist(datos, method = "manha
```

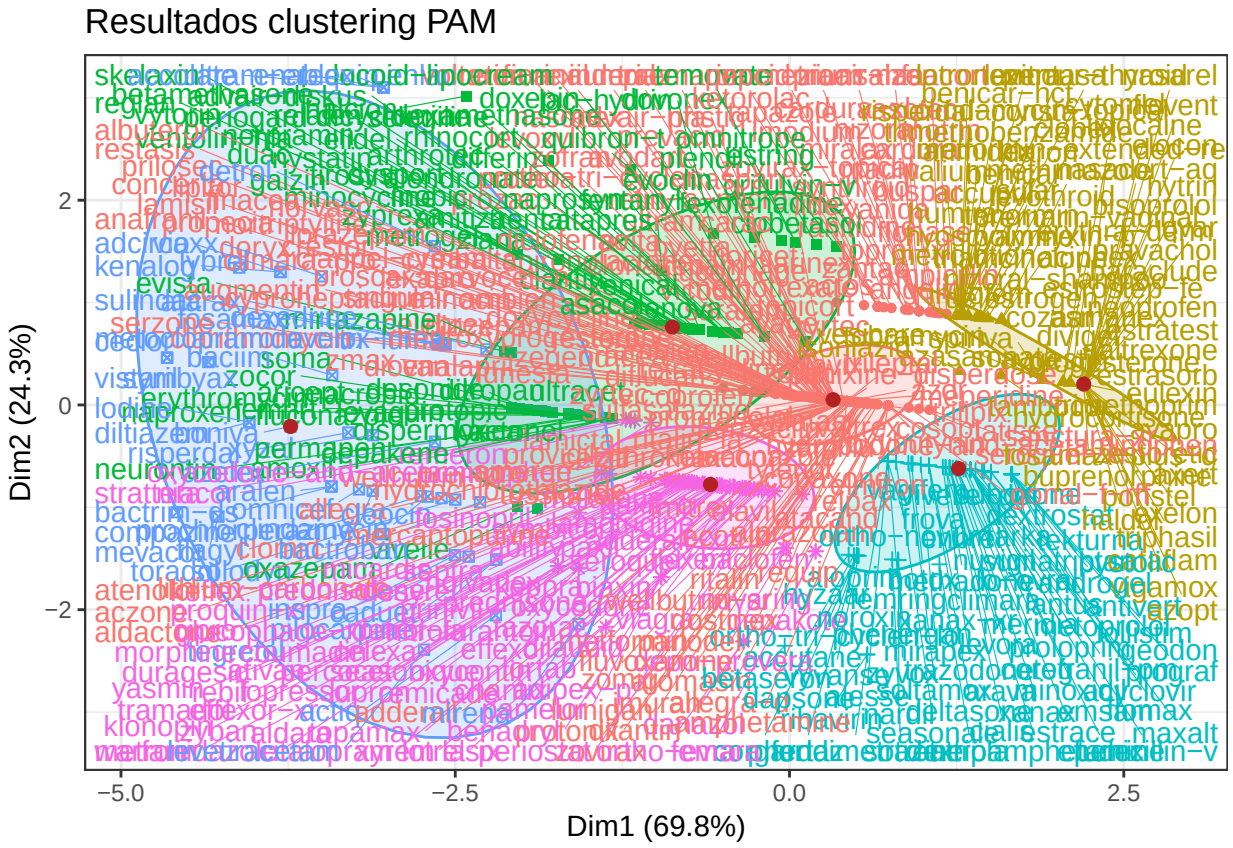



```
# Vemos que a partir de 6 clusters parece estabilizarse, por tanto, k=6
pam_clusters <- pam(x = datos, k = 6, metric = "manhattan")
#pam_clusters
# Observaciones que finalmente se han seleccionado
#pam_clusters$medoids
# Cluster al que se ha asignado cada observación
#pam_clusters$clustering

fviz_cluster(object = pam_clusters, data = datos, ellipse.type = "t", repel = TRUE) +
  theme_bw() +
  labs(title = "Resultados clustering PAM") +
  theme(legend.position = "none")
```



Como podemos observar, no hay centroides. Ahora vamos a resaltar las observaciones que actúan como medoids.

```
# Calculamos el PCA y extraemos las proyecciones almacenadas en el elemento x.
medoids <- prcomp(datos)$x

# Se seleccionan únicamente las proyecciones de las observaciones que son medoids
medoids <- medoids[rownames(pam_clusters$medoids), c("PC1", "PC2")]
medoids <- as.data.frame(medoids)

# Se emplean los mismos nombres que en el objeto ggplot
colnames(medoids) <- c("x", "y")

# Creación del gráfico
fviz_cluster(object = pam_clusters, data = datos, ellipse.type = "t",
             repel = TRUE) +
  theme_bw() +
  # Se resaltan las observaciones que actúan como medoids
  geom_point(data = medoids, color = "firebrick", size = 2) +
  labs(title = "Resultados clustering PAM") +
  theme(legend.position = "none")
```

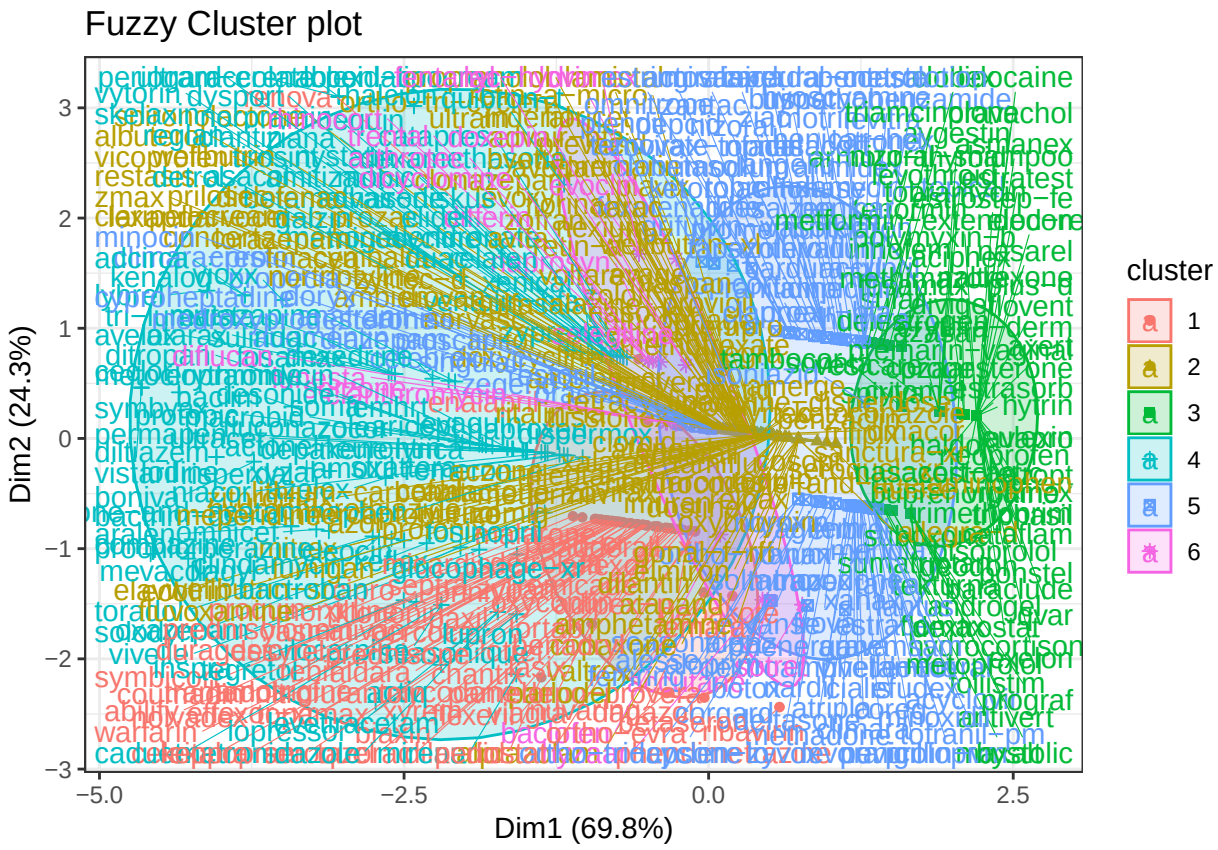


Clustering Difuso (FUZZY CLUSTERING)

```
fuzzy_cluster <- fanny(x = datos, k = 6, metric = "euclidean", stand = FALSE, maxit = 5000)
```

```
## Warning in fanny(x = datos, k = 6, metric = "euclidean", stand = FALSE, :
## FANNY algorithm has not converged in 'maxit' = 5000 iterations
```

```
# head(fuzzy_cluster$membership)
# fuzzy_cluster$coeff
# head(fuzzy_cluster$clustering)
fviz_cluster(object = fuzzy_cluster, repel = TRUE, ellipse.type = "norm", pallete = "jco") + theme_bw()
```



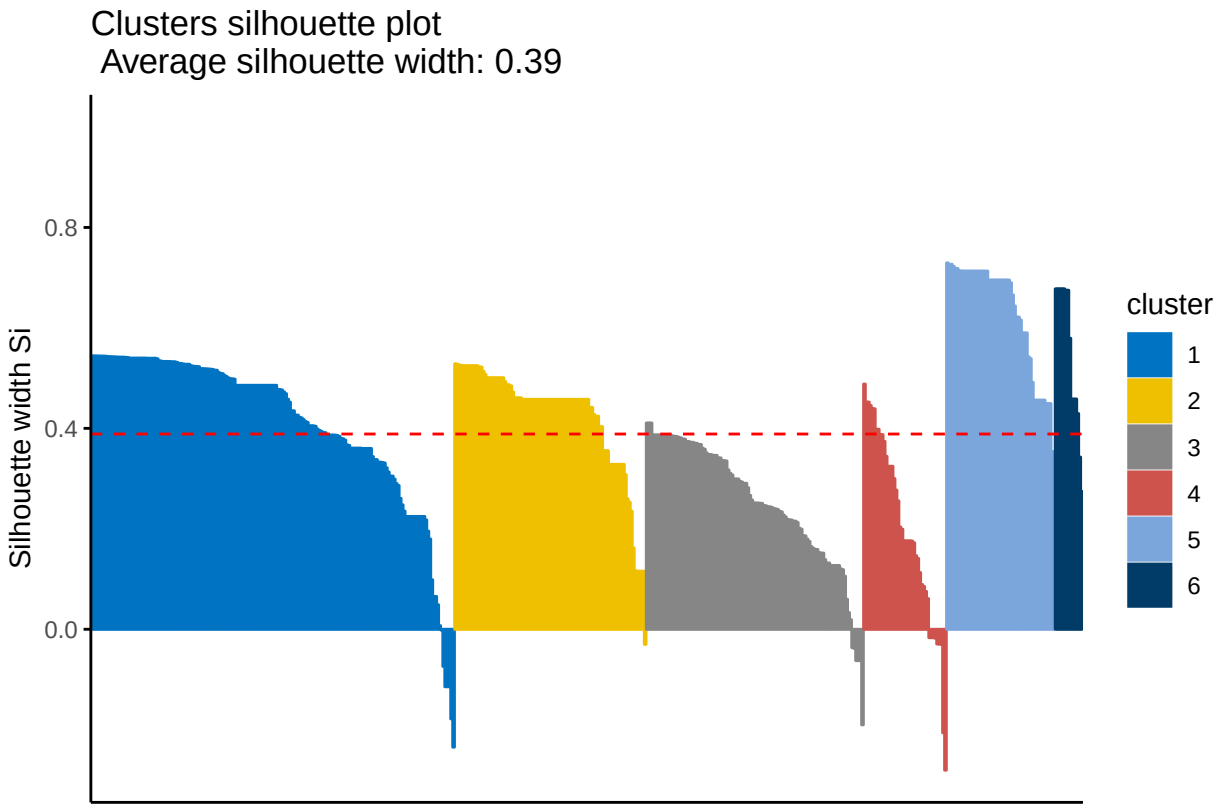
Hierarchical clustering

Hierarchical clustering es una alternativa a los métodos de partitioning clustering que no requiere que se pre-especifique el número de clusters. Se pueden seguir dos estrategias, aglomerativa o divisiva, en este caso vamos a usar la primera.

En el método aglomerativo el agrupamiento se inicia en el base del árbol, siendo cada observación un cluster individual. Entonces los cluster se van combinando conforme la estructura crece hasta unirse en única rama central.

```
# Vemos cosas
km_clusters <- eclust(x = datos, FUNcluster = "hclust", k = 6, seed = 123,
                      hc_metric = "euclidean",  graph = FALSE)
fviz_silhouette(sil.obj = km_clusters, print.summary = TRUE, palette = "jco",
                ggtheme = theme_classic())
```

```
##   cluster size ave.sil.width
## 1       1  184          0.41
## 2       2   97          0.42
## 3       3  110          0.25
## 4       4   42          0.20
## 5       5   55          0.63
## 6       6   14          0.55
```




```
# Verificar el árbol

# Evaluamos hasta que punto la estructura refleja las distancias originales entre observaciones. Para e

# Cuanto más cercano al 1 mejor. Valores superiores al 0.75 se consideran buenos.
# Matriz de distancias euclídeas
mat_dist <- dist(x = datos, method = "euclidean")
# Dendrogramas con linkage complete y average
hc_euclidea_complete <- hclust(d = mat_dist, method = "complete")
hc_euclidea_average  <- hclust(d = mat_dist, method = "average")

cor(x = mat_dist, cophenetic(hc_euclidea_complete)) # 0.7864
```

```
## [1] 0.6432507
```

```
cor(x = mat_dist, cophenetic(hc_euclidea_average)) # 0.9374272
```

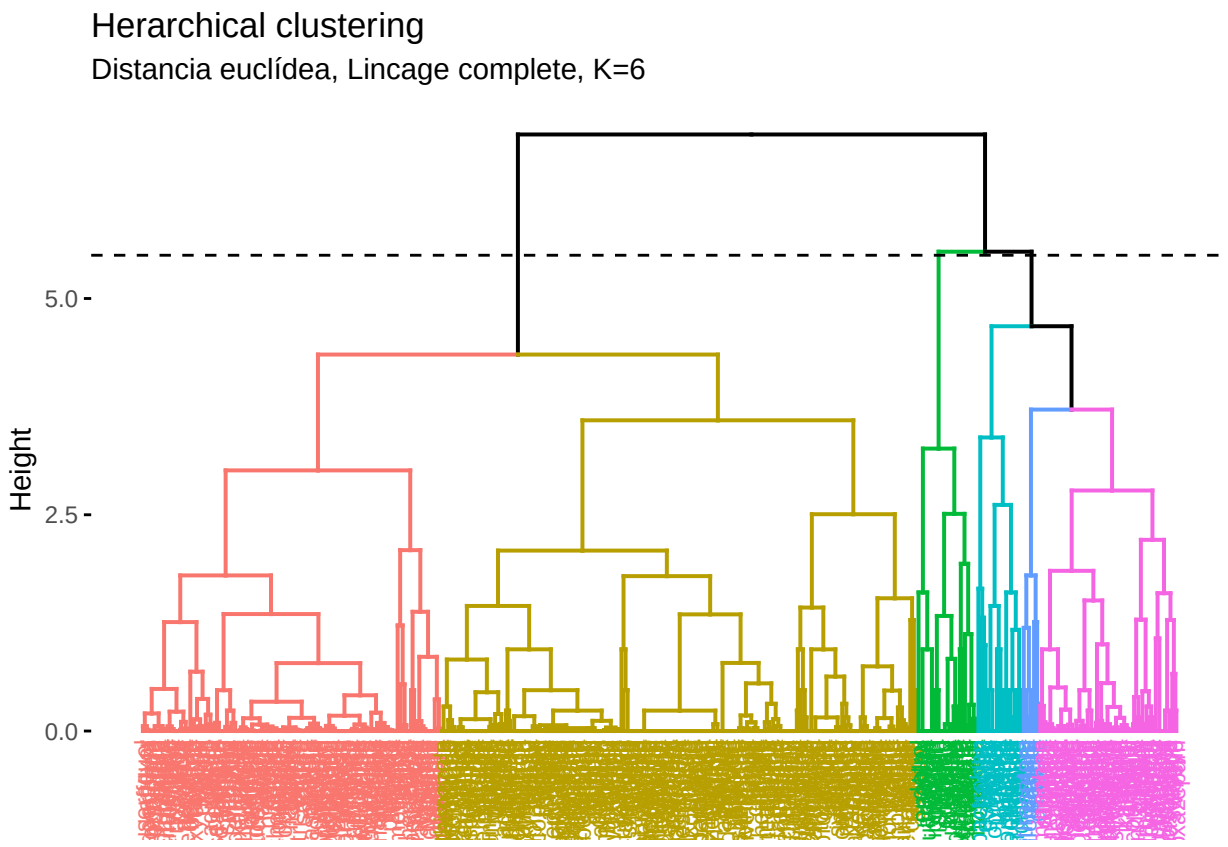
```
## [1] 0.7773464
```

Además, debemos de poder identificar el número de clusters creados y que observaciones forman parte de cada uno. Si realizamos un corte horizontal en el dendrograma, el número de ramas que lo sobrepasan se corresponden con el número de clusters.

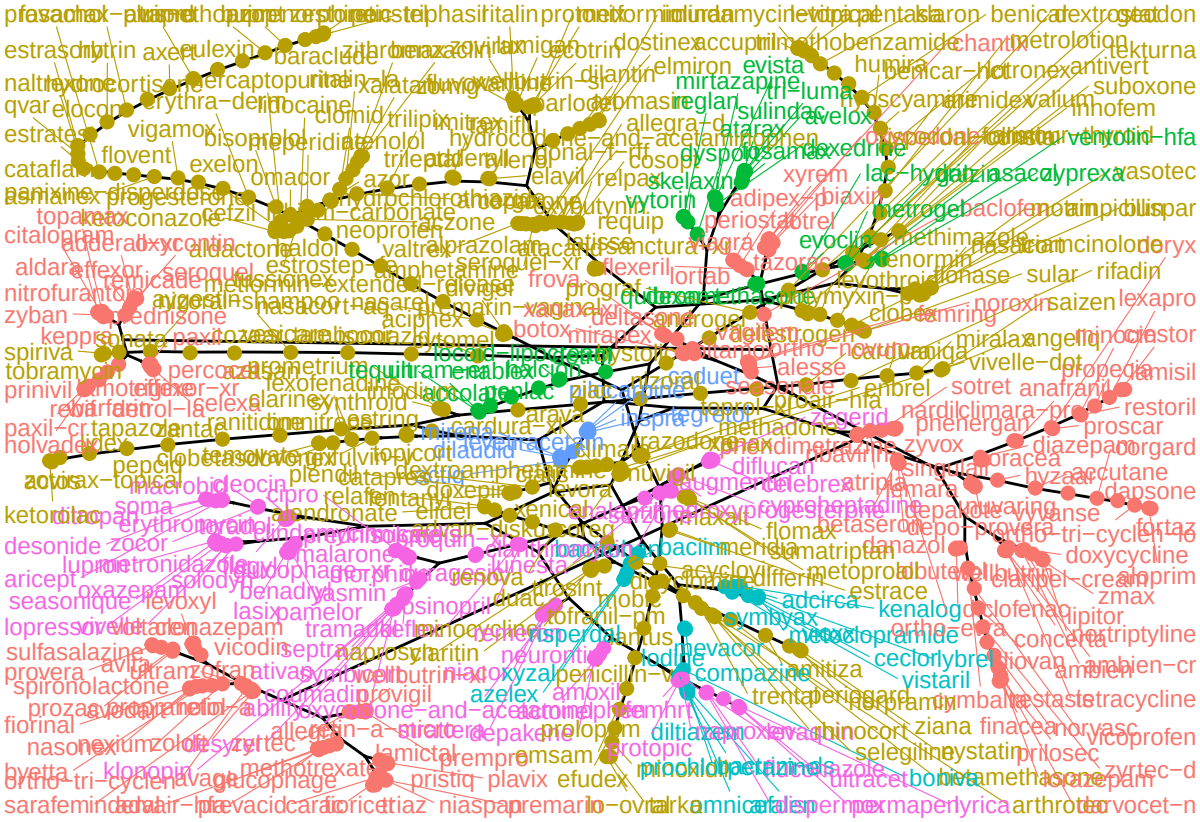
Usamos como altura de corte el valor K de K-means. K=6

```
# Cortar el árbol para generar los clusters

hc_euclidea_completo <- hclust(d = dist(x = datos, method = "euclidean"), method = "complete")

fviz_dend(x = hc_euclidea_completo, k = 6, cex = 0.6) +
  geom_hline(yintercept = 5.5, linetype = "dashed") +
```

```
  labs(title = "Herarchical clustering",
       subtitle = "Distancia euclídea, Lincage complete, K=6")
```

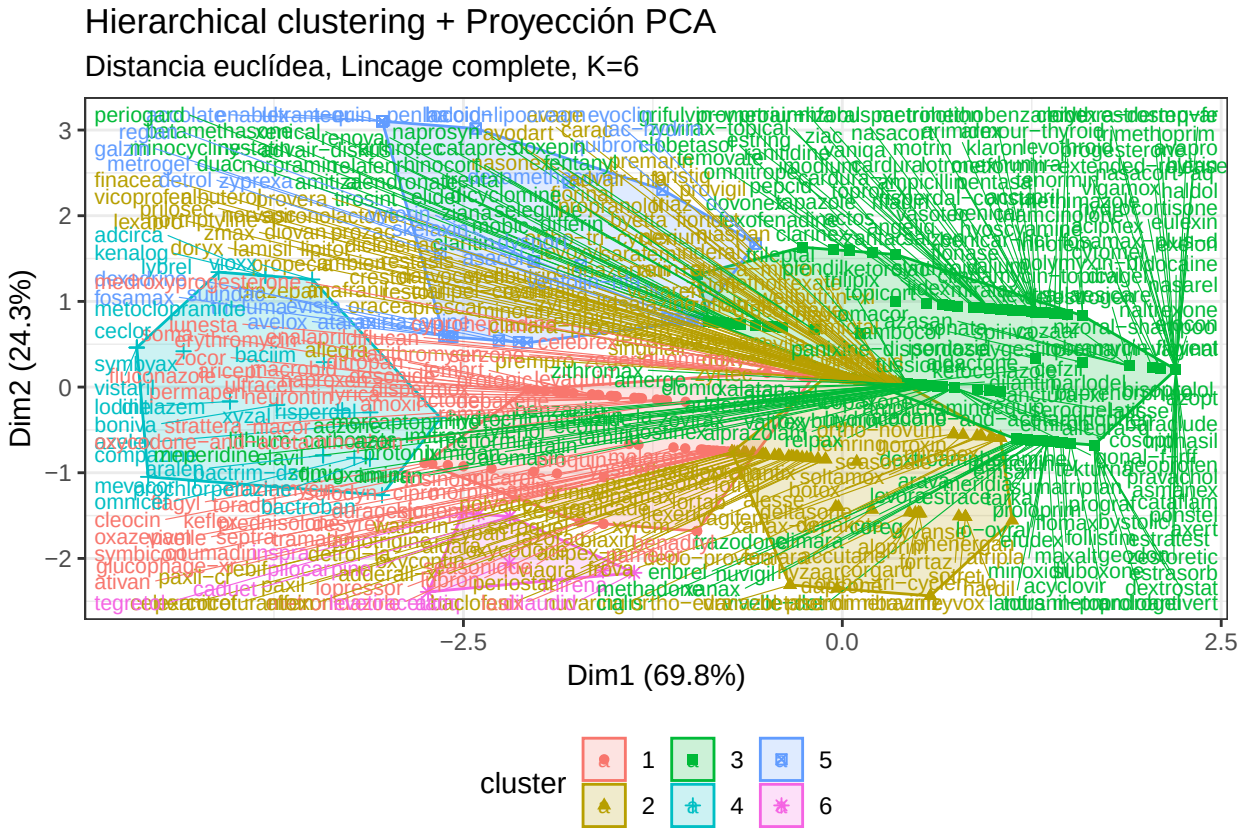



```
fviz_dend(x = hc_euclidea_completo,
          k = 6,
          color_labels_by_k = TRUE,
          cex = 0.8,
          type = "phylogenic",
          repel = TRUE, labelsize=0)
```



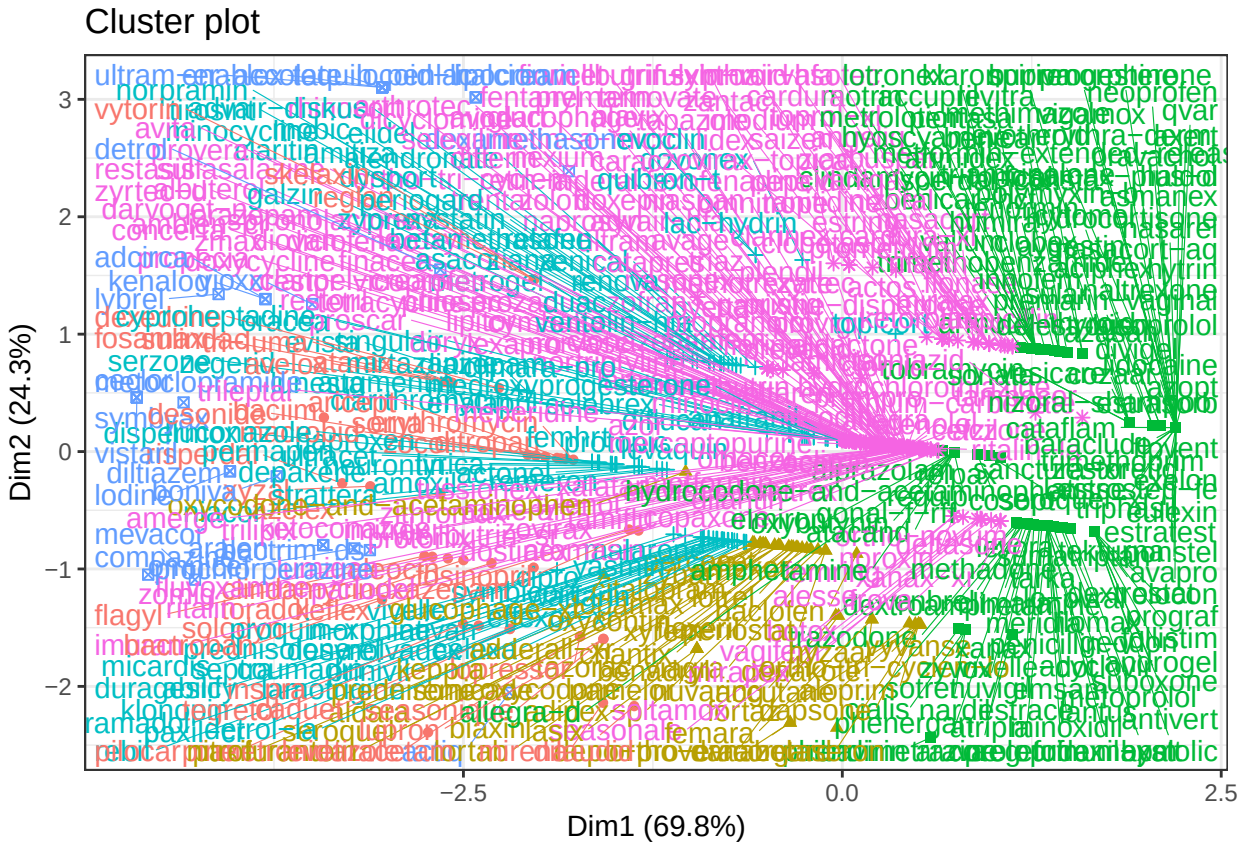
Otra forma de representar sería combinando con una reducción de dimensionalidad por PCA. Primero, se calculan los componentes principales y después se representan empleando las dos primeras componentes. Finalmente, se colorean los clusters mediante elipses.

```
fviz_cluster(object = list(data=datos, cluster=cutree(hc_euclidea_completo, k=6)),
             ellipse.type = "convex", repel = TRUE, show.clust.cent = FALSE,
             labelsize = 8)  +
  labs(title = "Hierarchical clustering + Proyección PCA",
       subtitle = "Distancia euclídea, Lincage complete, K=6") +
  theme_bw() +
  theme(legend.position = "bottom")
```



Density based clustering (DBSCAN)

```
datos_ <- data_train_procesado
km_clusters <- kmeans(x = datos_, centers = 6, nstart = 50)

# Ellipse FALSE??
# geom = "point"
fviz_cluster(object = km_clusters, data = datos_, repel = TRUE, ellipse = FALSE,
             show.clust.cent = FALSE, pallete = "jco") +
  theme_bw() +
  theme(legend.position = "none")
```



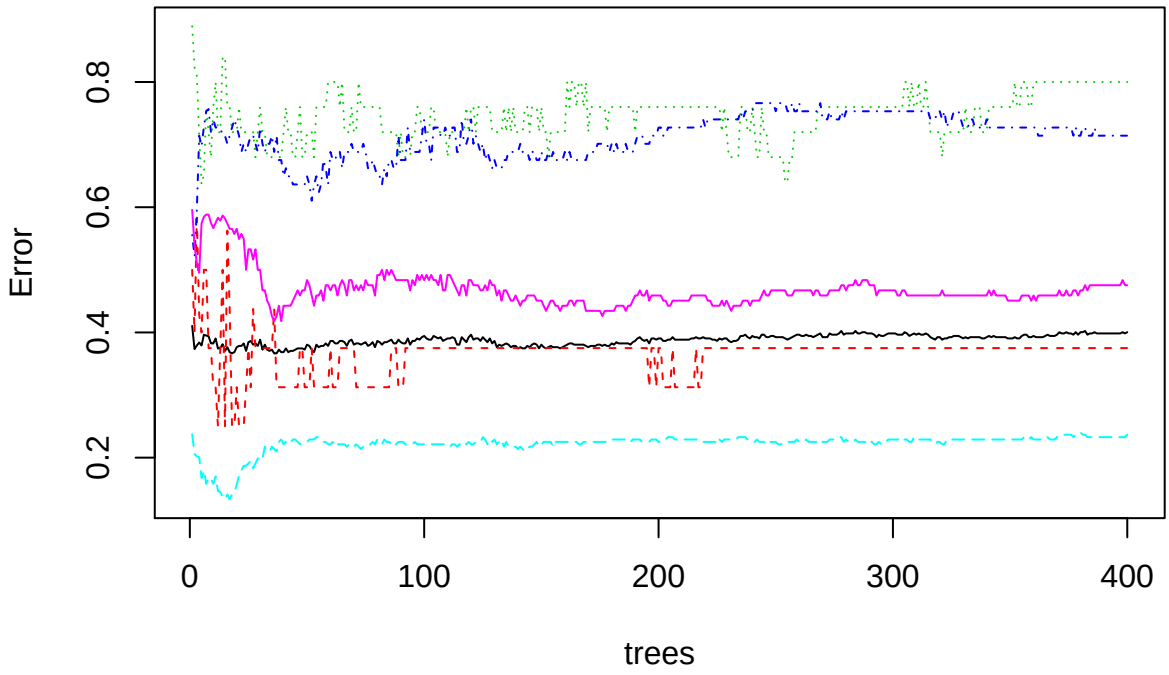
RANDOM FOREST

A ver, aquí hay que repasar un poquito esto, porque, con el modelo de RandomForest sale que el errorn

```
output.forest <- randomForest(as.factor(effectivenessNumber) ~ . , data = data_train_procesado, replace

output.forest
```

```
##
## Call:
##  randomForest(formula = as.factor(effectivenessNumber) ~ ., data = data_train_procesado,      replac
##                Type of random forest: classification
##                      Number of trees: 400
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 40.04%
## Confusion matrix:
##    1 2  3   4  5 class.error
## 1 10 0  4   2  0   0.3750000
## 2  4 5  9   7  0   0.8000000
## 3  2 1 22  52  0   0.7142857
## 4  0 2 28 200 32   0.2366412
## 5  1 0  1  56 64   0.4754098
```

```
plot(output.forest)
```




```
# Predicciones
mod_rf = predict(output.forest, newdata = data_test_procesado[-3], type="response")

# Fallo
y.falladas = sum(mod_rf!=data_test_procesado$effectivenessNumber)/length(data_test_procesado$effectiven

print("Error en test:")
```

```
## [1] "Error en test:"
```

```
print(y.falladas)
```

```
## [1] 0.3578275
```

Segun esto, nos podemos hcer una funcion que te diga en qué rango encontramos el mejor resultado, y con 150 árboles tenemos más que suficiente para alcanzar un 35% de error. Este error en nuestro caso es muy bueno porque tenemos muchisima subjetividad. Para unas personas, poca efectividad tendrá asociado rating 1 y para otras puede ser rating 3. Esto nos afecta mucho los resultados, y por tanto, aumenta el error.

```
valores_arboles_150 = seq(from=1, to=150, by=1)
valores_arboles_35 = seq(from=1, to=35, by=1)
valores_arboles_15 = seq(from=1, to=15, by=1)
valores_arboles_20 = seq(from=1, to=20, by=1)
valores_arboles_400 = seq(from=1, to=400, by=1)
valores_arboles_500 = seq(from=1, to=500, by=1)

obtener_arboles_optimo = function(x){
  #error_min = 100.0
  arboles_optimo = 1;

  for(i in x){
    error_min = 100.0
```

```
    set.seed(3)

    output.forest <- randomForest(as.factor(effectivenessNumber) ~ . , data = data_train_procesado, repl


    # Predicciones
    mod_rf = predict(output.forest, newdata = data_test_procesado[-3], type="response")

    # Fallo
    y.falladas = sum(mod_rf!=data_test_procesado$effectivenessNumber)/length(data_test_procesado$effecti


    if(y.falladas < error_min){
      error_min = y.falladas
      arboles_optimo = i
    }
  }

  cat("Numero de árboles óptimo",arboles_optimo, "\n")

  cat("Error minimo conseguido",error_min, "\n")

}


cat("El error minimo con 150 árboles:\n")
```

```
## El error minimo con 150 árboles:
```

```
## El error minimo con 150 árboles:
obtener_arboles_optimo(x = valores_arboles_150)
```

```
## Numero de árboles óptimo 150
## Error minimo conseguido 0.3546326
```

```
cat("El error minimo con 20 árboles:\n")
```

```
## El error minimo con 20 árboles:
```

```
## El error minimo con 20 árboles:
obtener_arboles_optimo(x = valores_arboles_20)
```

```
## Numero de árboles óptimo 20
## Error minimo conseguido 0.3738019
```

```
cat("El error minimo con 15 árboles:\n")
```

```
## El error minimo con 15 árboles:
```

```
## El error minimo con 15 árboles:
obtener_arboles_optimo(x = valores_arboles_15)
```

```
## Numero de árboles óptimo 15
## Error minimo conseguido 0.370607
```

```
cat("El error minimo con 400 árboles:\n")
```

```
## El error minimo con 400 árboles:
```

```
## El error minimo con 400 árboles:
obtener_arboles_optimo(x = valores_arboles_400)
```

```
## Numero de árboles óptimo 400
## Error minimo conseguido 0.3578275
```

```
cat("El error minimo con 500 árboles:\n")
```

```
## El error minimo con 500 árboles:
```

```
## El error minimo con 500 árboles:
obtener_arboles_optimo(x = valores_arboles_500)
```

```
## Numero de árboles óptimo 500
## Error minimo conseguido 0.3642173
```