UNIVERSIDAD DEL BÍO-BÍO

Trabajo

Analisis Exploratorio de Datos

Docente:
Luis Guzman

Estudiantes:
Felipe Ávila
Rudy Miranda

Índice

Índice de figuras

Índice de cuadros

1. Carga de Datos

```
getwd()
setwd("/home/eyeshield/Documents/A.E.D/trabajo")
datos  <-  read.csv("bee_colony.csv")
```

2. Ejercicio 1

2.1. a)

```
library(funModeling)
df_status(datos)
```

	variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
1	year	0	0	0	0.00	0	0	integer	7
2	months	0	0	0	0.00	0	0	character	4
3	state	0	0	0	0.00	0	0	character	47
4	colony_n	0	0	47	3.85	0	0	integer	284
5	colony_max	0	0	72	5.89	0	0	integer	278
6	colony_lost	0	0	47	3.85	0	0	integer	254
7	colony_lost_pct	0	0	54	4.42	0	0	integer	42
8	colony_added	0	0	83	6.79	0	0	integer	256
9	colony_reno	0	0	131	10.72	0	0	integer	252
10	colony_reno_pct	0	0	260	21.28	0	0	integer	54

Cuadro 1: Análisis exploratorio

```
nna_datos  <- na.omit(datos)
df_status(nna_datos)
```

	variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
1	year	0	0	0	0	0	0	integer	7
2	months	0	0	0	0	0	0	character	4
3	state	0	0	0	0	0	0	character	46
4	colony_n	0	0	0	0	0	0	integer	241
5	colony_max	0	0	0	0	0	0	integer	258
6	colony_lost	0	0	0	0	0	0	integer	223
7	colony_lost_pct	0	0	0	0	0	0	integer	39
8	colony_added	0	0	0	0	0	0	integer	228
9	colony_reno	0	0	0	0	0	0	integer	226
10	colony_reno_pct	0	0	0	0	0	0	integer	54

Cuadro 2: data frame limpio

```
library(pastecs)
round(stat.desc(nna_datos), digits = 2)
```

	year	months	state	colony_n	colony_max	colony_lost	colony_lost_pct	colony_added	colony_reno	colony_reno_pct
nbr.val	929	NA	NA	929	929	929	929	929	929	929
nbr.null	0	NA	NA	0	0	0	0	0	0	0
nbr.na	0	NA	NA	0	0	0	0	0	0	0
min	2015	NA	NA	1.6e+06	190	30	1	10	20	1
max	2021	NA	NA	1.44e+06	1.71+e06	255000	48	250000	2.85+e05	77
range	6	NA	NA	1.43+e06	1.7+e06	254970	47	249990	2.84e+05	76
median	2018	NA	NA	1.75+e04	2.2+e04	2200	10	2100	1.3e+03	6
mean	2017.77	NA	NA	1.07+e04	1.34+e04	9646.37	11.46	10329.04	8.95e+03	9.08
var	3.36	NA	NA	2.79+e10	4.37+e10	661651420.77	49.79	736043769.45	6.44e+08	94.64
std.dev	1.91	NA	NA	1.67+e05	2.09+e05	25722.59	7.06	27130.13	2.53e+04	9.73
coef.var	0	NA	NA	2.37	2.36	2.67	0.62	2.63	2.84	1.07

Cuadro 3: Análisis descriptivo

Por otro lado, los datos serán escalados al momento de realizar clusters.

2.2. b)

input

```
attach(nna_datos) #columnas como objetos del entorno
model  <- lm(colony_n ~ colony_lost) #Regresion lineal simple
model_log  <- lm(colony_n ~ log(colony_lost)) #model log lineal
model_loess  <- loess(colony_n ~ colony_lost) #regresion ponderada localmente
```

2.3. c)

Al ejecutar la función *girafe* de la librería *ggiraph* en linux obtenemos un error. La solución de este la encontramos en `https://rdrr.io/r/utils/browseURL.html`, donde solo tenemos que ejecutar en nuestra terminal con *R* corriendo `options(browser = "firefox")`.

input

```
library(ggplot2)
library(ggiraph)
gg_lm  <- ggplot(nna_datos, aes(x = colony_lost, y = colony_n)) +
    ggtitle("Modelo Lineal") +
    geom_point_interactive() +
    stat_smooth(method = "lm", col = "blue")

gg_log  <- ggplot(nna_datos, aes(x = log(colony_lost), y = colony_n)) +
    ggtitle("Modelo Loglineal") +
    geom_point_interactive() +
    stat_smooth(method = "lm", col = "blue")

gg_loess  <- ggplot(nna_datos, aes(x = colony_lost, y = colony_n)) +
    ggtitle("Modelo Loess") +
    geom_point_interactive() +
    stat_smooth(method = "loess", col = "blue")

girafe(ggobj = gg_lm)
girafe(ggobj = gg_log)
girafe(ggobj = gg_loess)
```

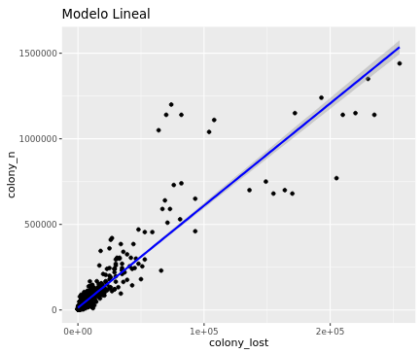

Figura 1: M. Lineal

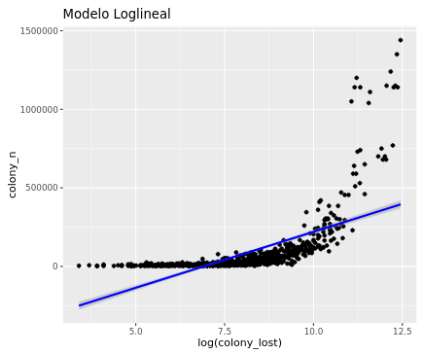

Figura 2: M. LogLineal

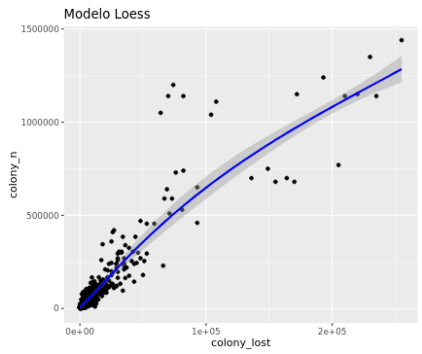

Figura 3: M. Loess

2.4. d)

input

```
summary(model)$sigma
summary(model_log)$sigma
summary(model_loess)$s
```

output

```
> summary(model)$sigma
[1] 66068.51
> summary(model_log)$sigma
[1] 127610.2
> summary(model_loess)$s
[1] 60349.14
```

Fijándonos en la suma de los residuos, entonces el modelo que mejor se ajusta a los datos es el de regresión ponderada localmente (`model_loess`).

2.5. e)

Para generar los números aleatorios nos apoyaremos de la distribución uniforme, donde los parámetros serán el valor mínimo y máximo de las variables *colony_lost*.

input

```
#Numeros aleatorio con runif
new_data <- sort(runif(10, min = min(colony_lost), max = max(colony_lost)))
predict(model_loess, new_data) #Predicciones
```

output

```
> cbind(new_data, predict(model_loess, new_data))
       new_data
 [1,]  69678.62  480021.7
 [2,]  87559.58  582172.7
 [3,] 106706.83  682813.9
 [4,] 124818.13  770726.8
 [5,] 182070.19 1013448.4
 [6,] 183339.79 1018384.0
 [7,] 201319.32 1086938.7
 [8,] 205476.97 1102494.7
 [9,] 234381.95 1208925.2
[10,] 242686.62 1239304.7
```

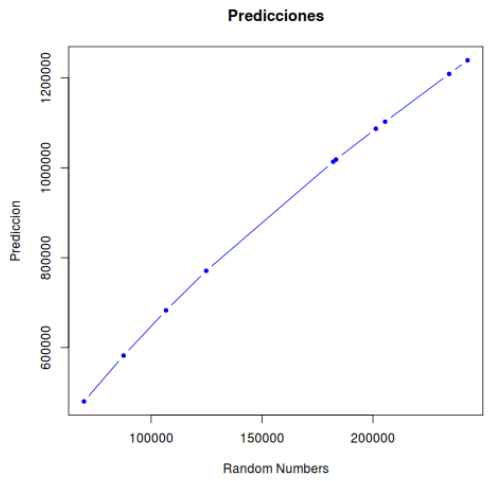

Figura 4: Predicciones

2.6. f)

input

```
severalv_model <- lm(colony_n ~ year + colony_max +
colony_lost + colony_lost_pct + colony_added +
colony_reno + colony_reno_pct)
summary(severalv_model)$sigma
```

output

```
> summary(severalv_model)$sigma
[1] 34798.53
```

Notar que solo se hizo uso de las variables cuantitativas, pero de todos modos se obtuvo un modelo mucho mejor que los anteriores, con un error casi de la mitad que el obtenido por el método *loess*.

3. Ejercicio 3

Haremos primero el tres, ya que se podría dar el caso de que nos entregue que la cantidad optima de clusters sea de dos, lo cual nos puede dar una idea de como realizar el ejercicio 2, ya que de momento no se tiene una noción clara de como abordarlo, puesto que las dos variables cualitativas que tenemos, ninguna de ellas es dicotómica.

Volvemos a cargar nuestros datos.

```
datos  <-  read.csv("bee_colony.csv") #cargamos .csv
nna_datos <- na.omit(datos) # se quitar na
trimestres <- nna_datos[, 2] # guardamos variables trimestres
nna_datos <- nna_datos[,-2] # se quita de nuestro conjunto
nna_datos$state <- as.factor(nna_datos$state) # estados a factor
datos <- scale(nna_datos[,-2]) # se escalan todas las variables,
# menos estado, que es factor
datos <- cbind(datos,nna_datos$state) # se unen nuevamente
```

Partiremos con el método jerárquico acumulativo.

3.1. Metodos Jerarquicos

3.1.1. Metodo Jerarquico Acumulativo

```
# Matriz de distancias euclídeas
mat_dist <- dist(x = datos, method = "euclidean")
# Dendrogramas con linkage complete y average
hc_complete <- hclust(d = mat_dist, method = "complete")
hc_average <- hclust(d = mat_dist, method = "average")
```

Ahora evaluamos hasta que punto nuestros dendrogramas reflejan las verdaderas distancias entre las observaciones con la función `cor`

input

```
cor(x = mat_dist, cophenetic(hc_complete))
cor(x = mat_dist, cophenetic(hc_average))
```

output

```
> cor(x = mat_dist, cophenetic(hc_complete))
[1] 0.7236253
> cor(x = mat_dist, cophenetic(hc_average))
[1] 0.7184969
```

Los cuales no son valores del todo buenos. De todas formas, procedemos a graficar.

input

```
library(magrittr)
library(factoextra)
#2 clusters
fviz_dend(x = hc_complete, k = 2, cex = 0.6)
#4 clusters
fviz_dend(x = hc_complete, k = 4, cex = 0.6)
```

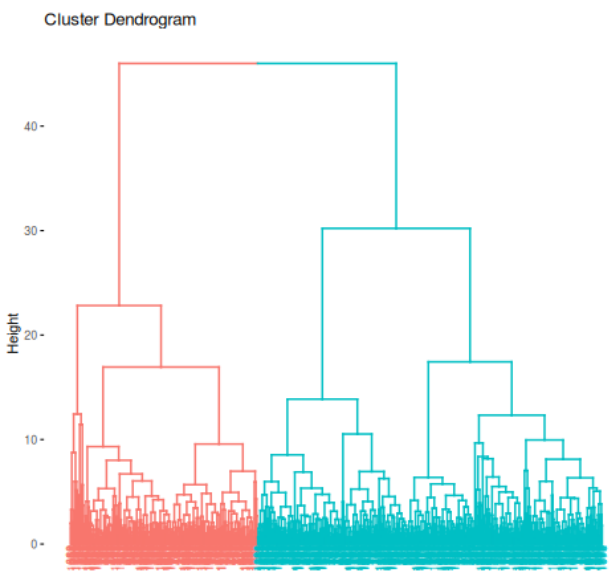

Figura 5: 2 Clusters

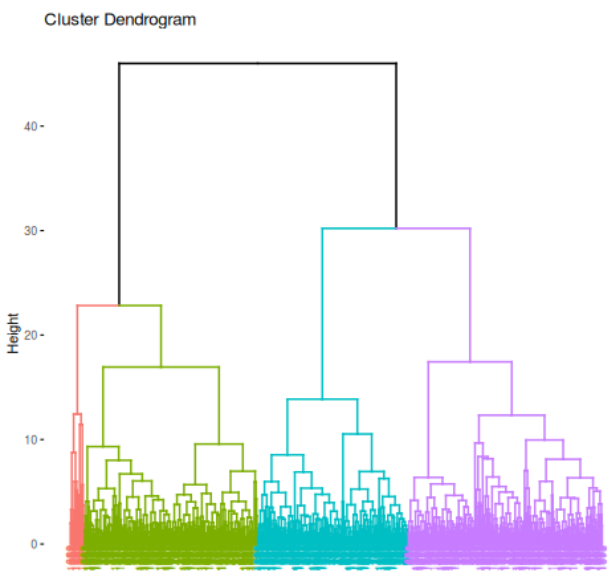

Figura 6: 4 Clusters

Nuestra esperanza era que se distribuyeran en los distintos clusters los trimestres. Ahora veremos si los resultados cumplieron con ello.

input

```
clusters = cutree(tree = hc_complete, k = 2)
table(clusters, trimestres, dnn = list("clusters", "trimestres"))
clusters = cutree(tree = hc_complete, k = 4)
table(clusters, trimestres, dnn = list("clusters", "trimestres"))
```

output

```
> clusters = cutree(tree = hc_complete, k = 2)
> table(clusters, trimestres, dnn = list("clusters", "trimestres"))
        trimestres
clusters April-June January-March July-September October-December
       1         88            82             92               65
       2        185           134            173              110
> clusters = cutree(tree = hc_complete, k = 4)
> table(clusters, trimestres, dnn = list("clusters", "trimestres"))
        trimestres
clusters April-June January-March July-September October-December
       1         82            75             86               59
       2          6             7              6                6
       3         84            57             75               47
       4        101            77             98               63
```

Tanto para dos, como para cuatro clusters, podemos notar que los trimestres no se asignan a un cluster en particular, por lo que el algoritmo debe de estar apuntando a otra cosa.

3.2. Metodos no Jerarquicos

Comenzaremos usando la función `clValid` de la librería con el mismo nombre, para así tener una de idea de la cantidad de clusters y en que método aplicarlos.

```
library(clValid)
comparacion <- clValid(obj = datos, nClust = c(2, 3, 4),
    clMethods = c("hierarchical", "kmeans", "pam"), validation = c("stability", "internal"))
summary(comparacion)
```

	Score	Method	Clusters
APN	0.0249	kmeans	2
AD	6.7405	pam	4
ADM	1.2041	pam	3
FOM	2.0976	kmeans	4
Connectivity	10.7472	pam	2
Dunn	0.0633	kmeans	4
Silhouette	0.5978	pam	2

Cuadro 4: Optimal Scores

3.2.1. K-means

De la tabla anterior, tenemos que dentro de nuestro rango de cluster, la mejor combinación que podríamos hacer es *Kmeans* con 2 clusters. Esto lo corroboraremos con una representacion visual.

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

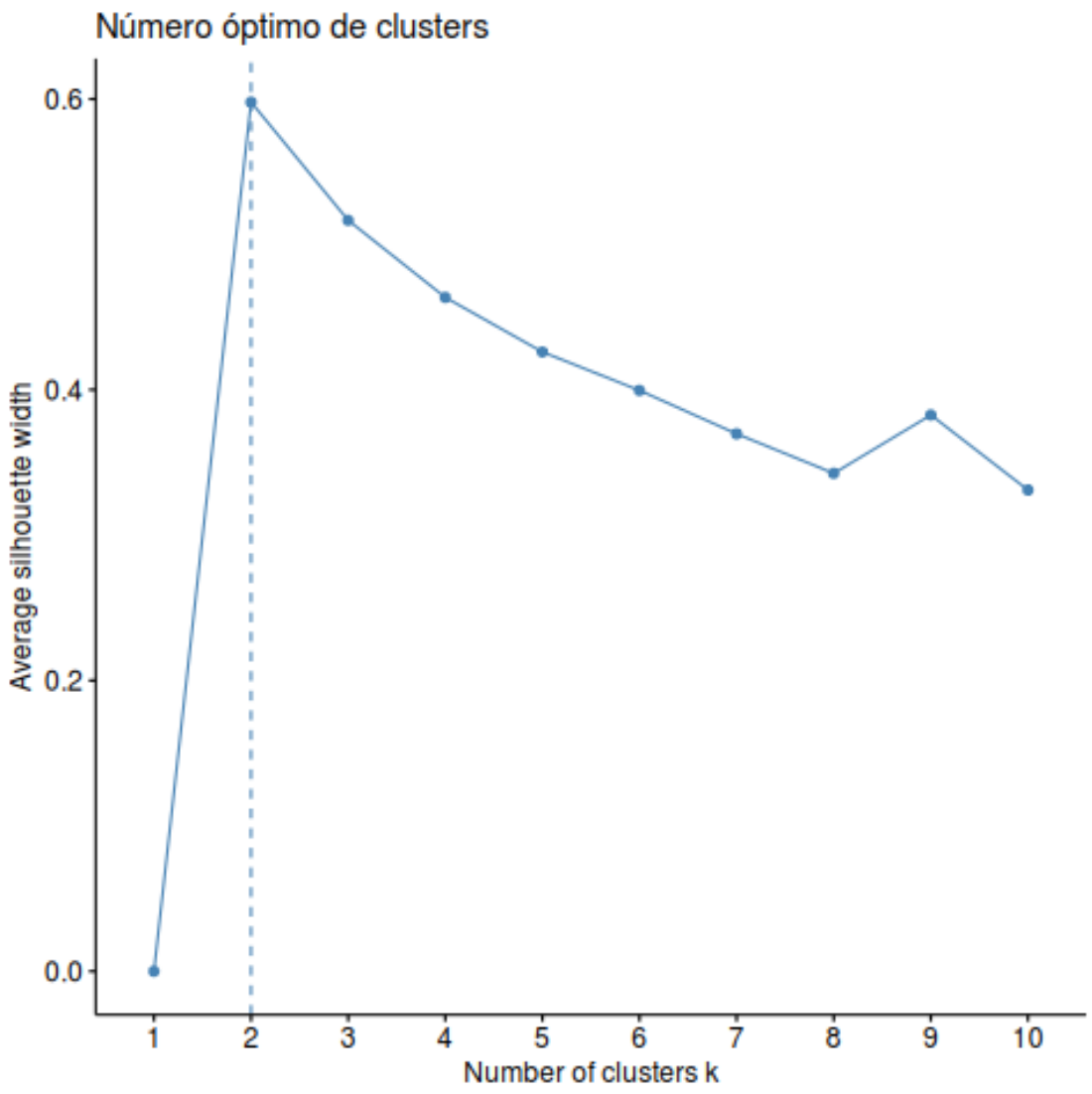

Figura 7: Numeros Optimo de Clusters

```
km_clusters <- kmeans(x = datos, centers = 2, nstart = 25)
fviz_cluster(object = km_clusters, data = datos, show.clust.cent = TRUE,
    ellipse.type = "euclid", star.plot = TRUE, repel = TRUE) + theme_bw()
```

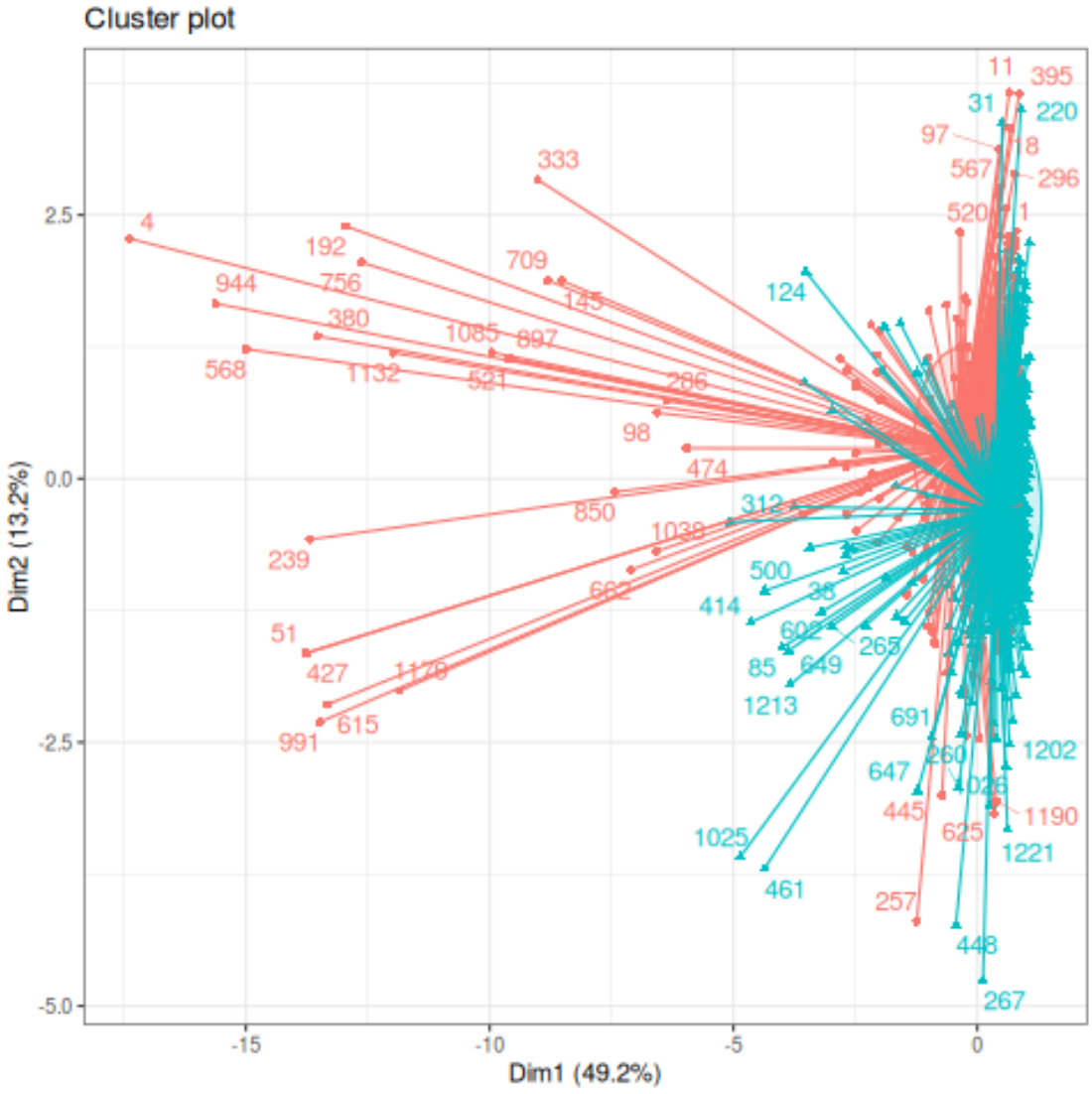

Figura 8: Kmeans k = 2

3.2.2. PAM

```
library(cluster)
pam_clusters = pam(x = datos, k = 4, metric = "euclidean")
fviz_cluster(object = pam_clusters, data = datos, ellipse.type = "t",
        repel = TRUE) + theme_bw() + theme(legend.position = "none")
```

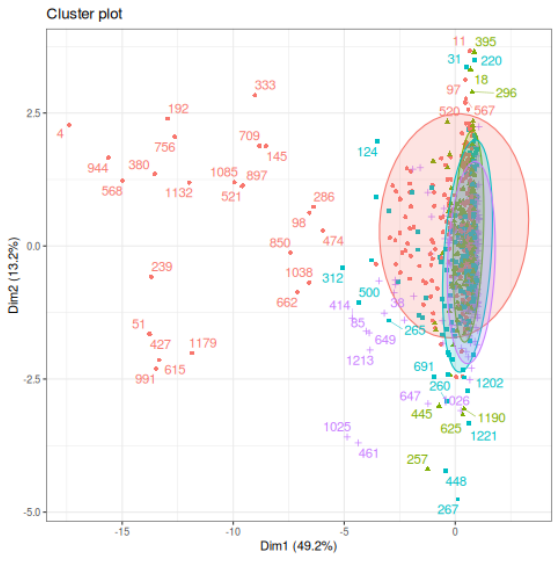

Figura 9: Pam k = 4

3.2.3. Clara

```
clara_clusters = clara(x = datos, k = 3, metric = "manhattan", stand = TRUE,
    samples = 50, pamLike = TRUE)
clara_clusters
#Grafica
fviz_cluster(object = clara_clusters, ellipse.type = "t", geom = "point") +
        theme_bw() + theme(legend.position = "none")
```

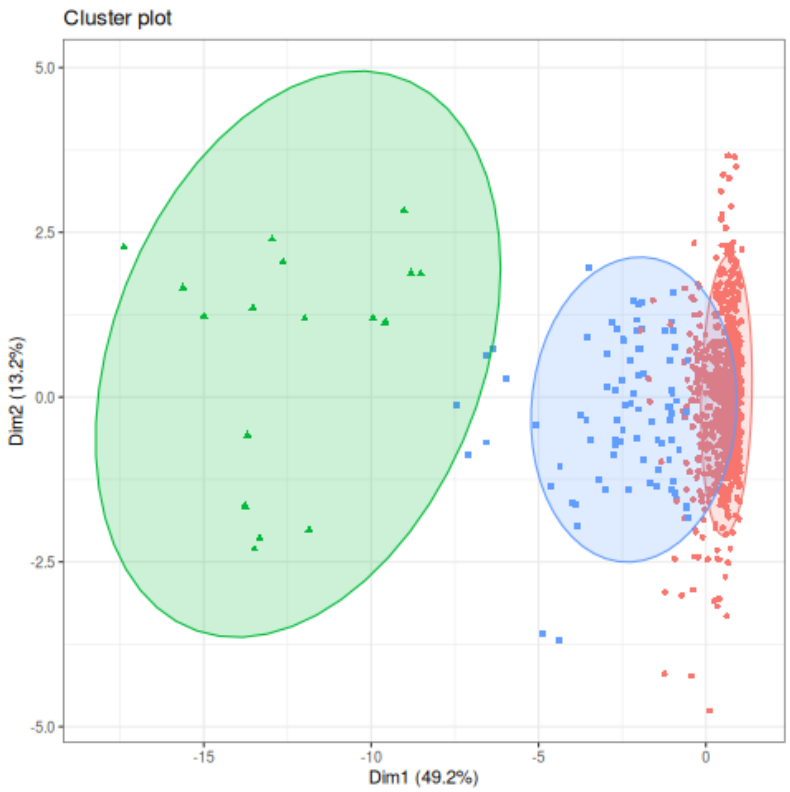

Figura 10: Clara k = 3

3.2.4. K-means Jerarquico

```
#Consideramos k = 2
hkmeans_cluster = hkmeans(x = datos, hc.metric = "euclidean",
    hc.method = "complete", k = 2)
fviz_cluster(object = hkmeans_cluster, pallete = "jco", repel = TRUE) +
        theme_bw() + labs(title = "Hierarchical k-means Cluster plot")
#Consideramos k = 3
hkmeans_cluster = hkmeans(x = datos, hc.metric = "euclidean",
    hc.method = "complete", k = 3)
fviz_cluster(object = hkmeans_cluster, pallete = "jco", repel = TRUE) +
        theme_bw() + labs(title = "Hierarchical k-means Cluster plot")
```

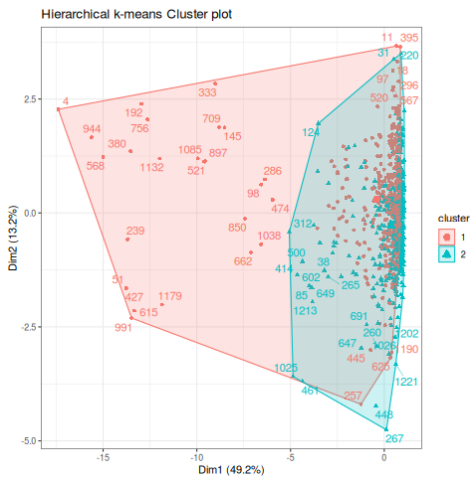

Figura 11: 2 Clusters

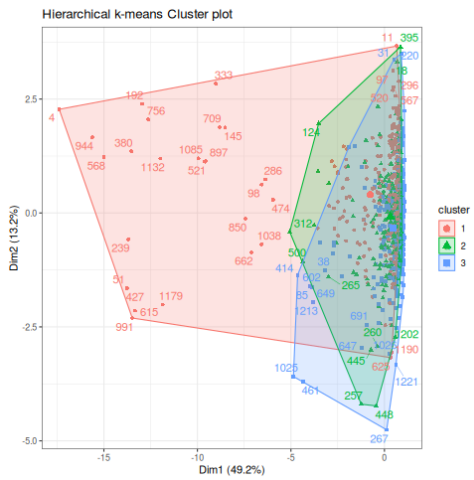

Figura 12: 4 Clusters

3.2.5. Fuzzy Clustering

input

```
fuzzy_cluster = fanny(x = datos, diss = FALSE, k = 3,
    metric = "euclidean", stand = FALSE)
#coef de Dunn
fuzzy_cluster$coeff
```

output

```
> fuzzy_cluster$coeff # alto nivel de fuzzy
dunn_coeff normalized
 0.4565086  0.2753448
```

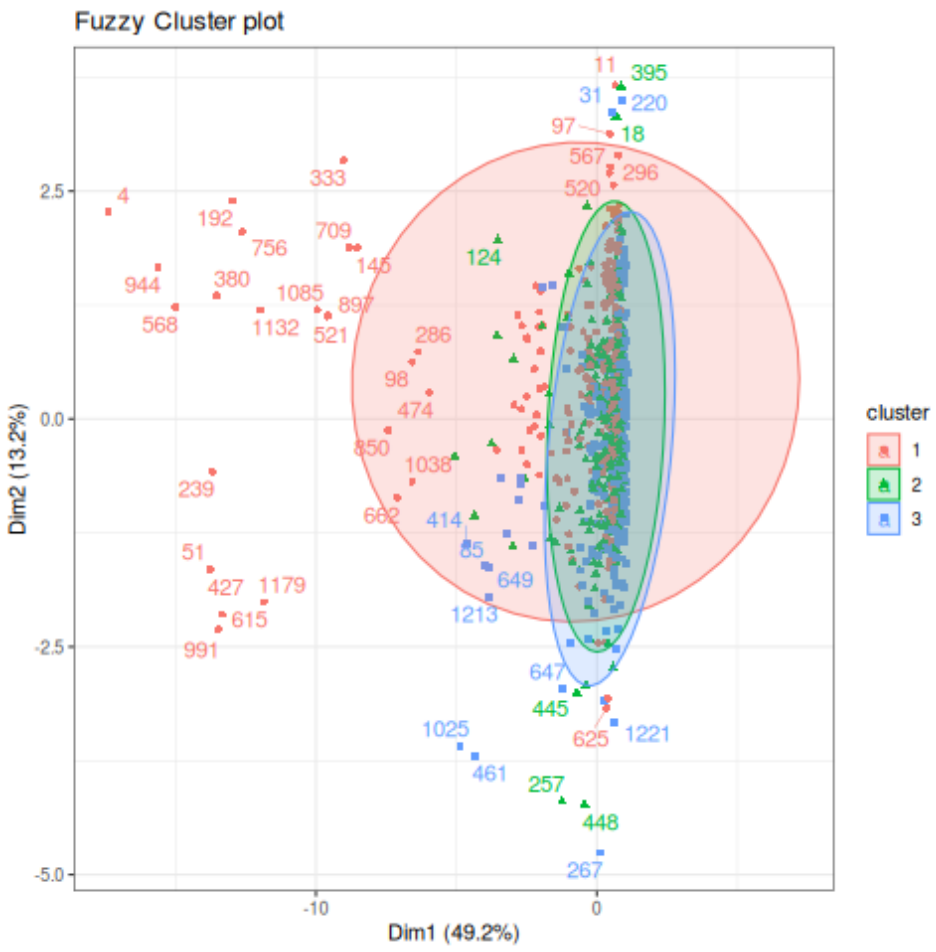

Figura 13: Fuzzy Cluster k = 4

3.2.6. Validación de Clusters

Solo aplicaremos criterios de validación a *kmeans* con k = 2, ya que según la función `clValid` es la mejor opción.

input

```
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())
km_clusters$silinfo$clus.avg.widths
```

output

```
> km_clusters$silinfo$clus.avg.widths
[1] 0.6807337 0.5322582
```

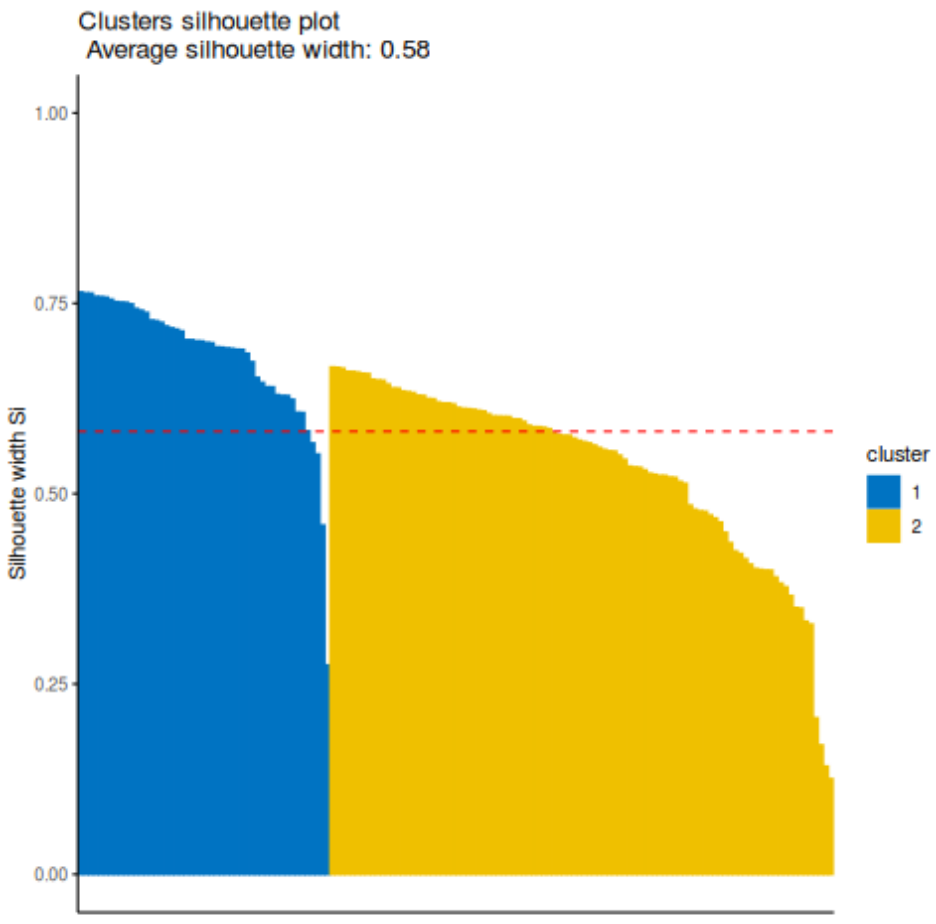

Figura 14: Validación Kmeans k = 2

Notamos que no hay observaciones que tomen valores negativos, y pocos de estos son cercanos a cero, por lo que muy probablemente las observaciones estén bien clasificadas.