

Estudios de Informática, Multimedia y Telecomunicaciones

Minería de datos: PEC2 - Métodos no supervisados

Autor: Eduardo Mora González

Octubre 2021

- 1 Introducción
 - 1.1 Presentación
 - 1.2 Objetivos
 - 1.3 Descripción de la PEC a realizar
 - 1.4 Recursos Básicos
 - 1.5 Criterios de valoración
 - 1.6 Formato y fecha de entrega
 - 1.7 Nota: Propiedad intelectual
- 2 Ejemplo 1.1
 - 2.1 Método de agregación k-means con datos autogenerados
- 3 Ejemplo 1.2
 - 3.1 Método de agregación k-means con datos reales
- 4 Ejemplo 2
 - 4.1 Métodos basados en densidad: DBSCAN y OPTICS
- 5 Ejercicios
 - 5.1 Ejercicio 1
 - 5.2 Ejercicio 2
 - 5.3 Ejercicio 3
- 6 Criterios de evaluación
 - 6.1 Ejercicio 1
 - 6.2 Ejercicio 2
 - 6.3 Ejercicio 3

1 Introducción

1.1 Presentación

Esta Prueba de Evaluación Continuada cubre principalmente el módulo de generación de modelos no supervisados del programa de la asignatura.

1.2 Objetivos

En esta PEC trabajaremos la generación, interpretación y evaluación de un modelo de agregación *k-means* y otro *DBSCAN*. No perderemos de vista las fases de preparación de los datos, calidad del modelo y extracción inicial del conocimiento.

1.3 Descripción de la PEC a realizar

1.4 Recursos Básicos

Material docente proporcionado por la UOC.

Módulo Métodos no supervisados del material didáctico.

1.5 Criterios de valoración

Ejercicios teóricos

Todos los ejercicios deben ser presentados de forma razonada y clara, especificando todos y cada uno de los pasos que se hayan llevado a cabo para su resolución. No se aceptará ninguna respuesta que no esté claramente justificada.

Ejercicios prácticos

Para todas las PEC es necesario documentar en cada apartado del ejercicio práctico que se ha hecho y cómo se ha hecho.

1.6 Formato y fecha de entrega

El formato de entrega es: **usernameestudiant-PECn.html (pdf o word) y rmd**

Se debe entregar la PEC en el buzón de entregas del aula

1.7 Nota: Propiedad intelectual

A menudo es inevitable, al producir una obra multimedia, hacer uso de recursos creados por terceras personas. Es por lo tanto comprensible hacerlo en el marco de una práctica de los estudios de Informática, Multimedia y Telecomunicación de la UOC, siempre y cuando esto se documente claramente y no suponga plagio en la práctica.

Por lo tanto, al presentar una práctica que haga uso de recursos ajenos, se debe presentar junto con ella un documento en qué se detallen todos ellos, especificando el nombre de cada recurso, su autor, el lugar dónde se obtuvo y su estatus legal: si la obra está protegida por el copyright o se acoge a alguna otra licencia de uso (Creative Commons, licencia GNU, GPL ...). El estudiante deberá asegurarse de que la licencia no impide específicamente su uso en el marco de la práctica. En caso de no encontrar la información correspondiente tendrá que asumir que la obra está protegida por copyright.

Deberéis, además, adjuntar los ficheros originales cuando las obras utilizadas sean digitales, y su código fuente si corresponde.

2 Ejemplo 1.1

2.1 Método de agregación k-means con datos autogenerados

En este ejemplo vamos a generar un conjunto de muestras aleatorias para posteriormente usar el algoritmo *k-means* para agruparlas. Se crearán las muestras alrededor de dos puntos concretos. Por lo tanto, lo lógico será agrupar en dos clústers. Puesto que inicialmente, en un problema real, no se conoce cual es el número más idóneo de clústers *k*, vamos a probar primero con dos (el valor óptimo) y posteriormente con 4 y 8 clústers. Para evaluar la calidad de cada proceso de agrupación vamos a usar la silueta media. La silueta de cada muestra evalúa como de bien o mal está clasificada la muestra en el clúster al que ha sido asignada. Para ello se usa una fórmula que tiene en cuenta la distancia a las muestras de su clúster y la distancia a las muestras del clúster vecino más cercano.

A la hora de probar el código que se muestra, es importante tener en cuenta que las muestras se generan de forma aleatoria y también que el algoritmo *k-means* tiene una inicialización aleatoria. Por lo tanto, en cada ejecución se obtendrá unos resultados ligeramente diferentes.

Lo primero que hacemos es cargar la librería `cluster` que contiene las funciones que se necesitan

```
if (!require('cluster')) install.packages('cluster')
library(cluster)
# https://cran.r-project.org/web/packages/ggplot2/index.html
if (!require('ggplot2')) install.packages('ggplot2'); library('ggplot2')
# https://cran.r-project.org/web/packages/dplyr/index.html
if (!require('dplyr')) install.packages('dplyr'); library('dplyr')
```

Generamos las muestras de forma aleatoria tomando como centro los puntos [0,0] y [5,5].

```

n <- 150 # número de muestras
p <- 2   # dimensión

sigma <- 1 # varianza de la distribución
mean1 <- 0 # centro del primer grupo
mean2 <- 5 # centro del segundo grupo

n1 <- round(n/2) # número de muestras del primer grupo
n2 <- round(n/2) # número de muestras del segundo grupo

x1 <- matrix(rnorm(n1*p,mean=mean1,sd=sigma),n1,p)
x2 <- matrix(rnorm(n2*p,mean=mean2,sd=sigma),n2,p)

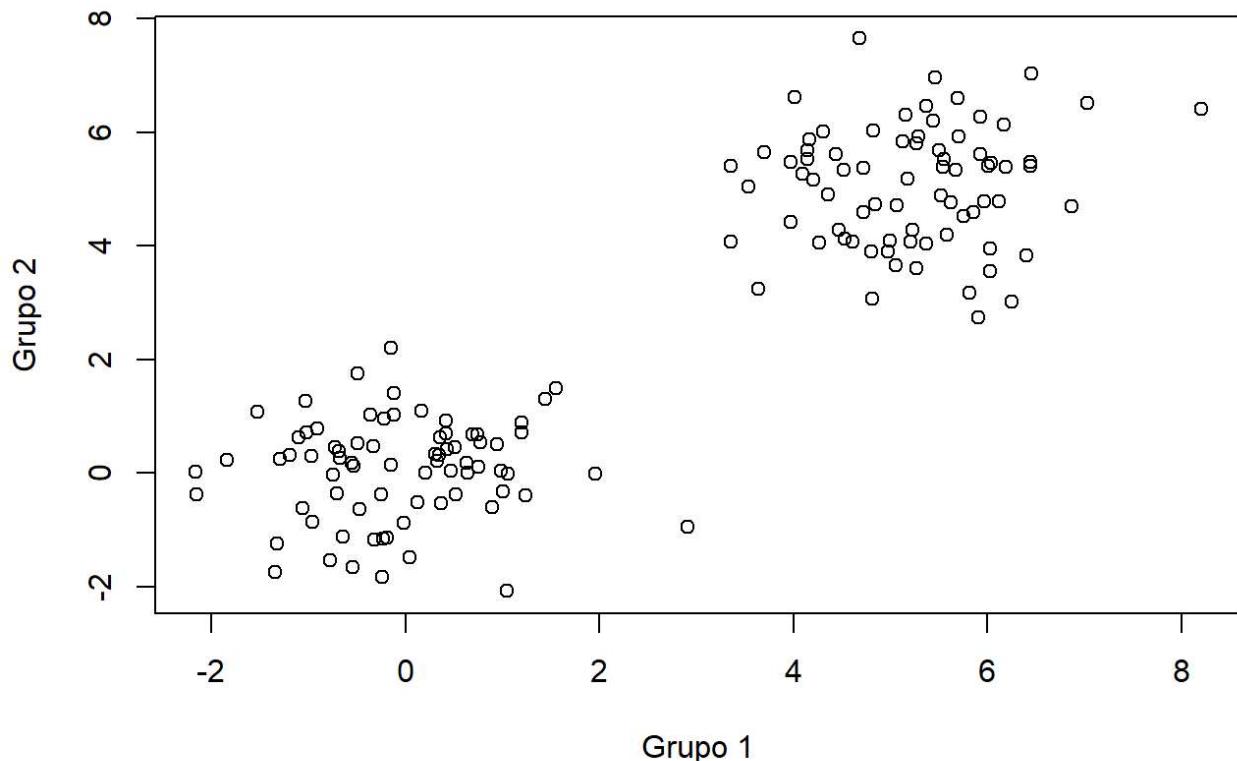
```

Juntamos todas las muestras generadas y las mostramos en una gráfica

```

x <- rbind(x1,x2)
plot (x, xlab="Grupo 1", ylab="Grupo 2")

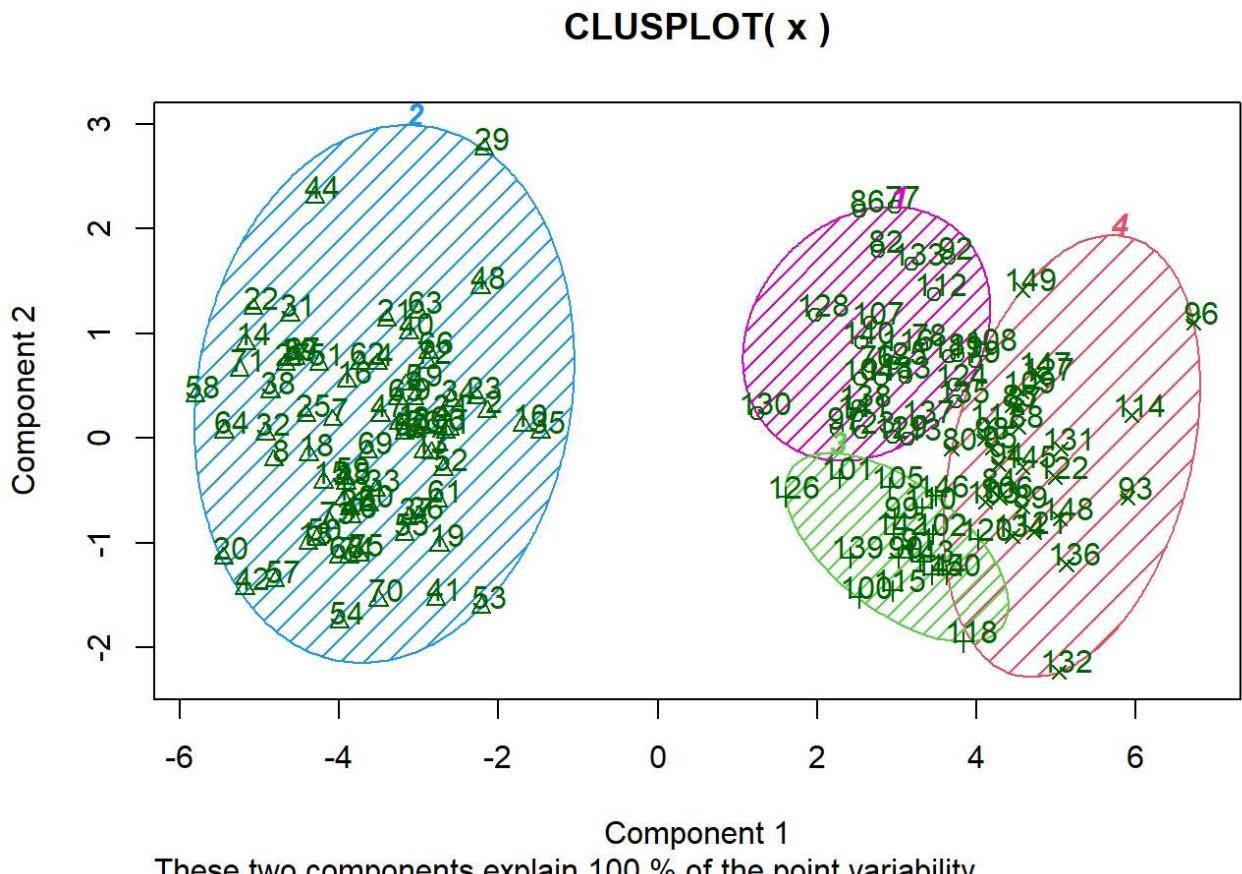
```



Como se puede comprobar las muestras están claramente separadas en dos grupos. Si se quiere complicar el problema se puede modificar los puntos centrales (mean1 y mean2) haciendo que estén más próximos y/o ampliar la varianza (sigma) para que las muestras estén más dispersas.

A continuación vamos a aplicar el algoritmo *k-means* con 2, 4 y 8 clústers

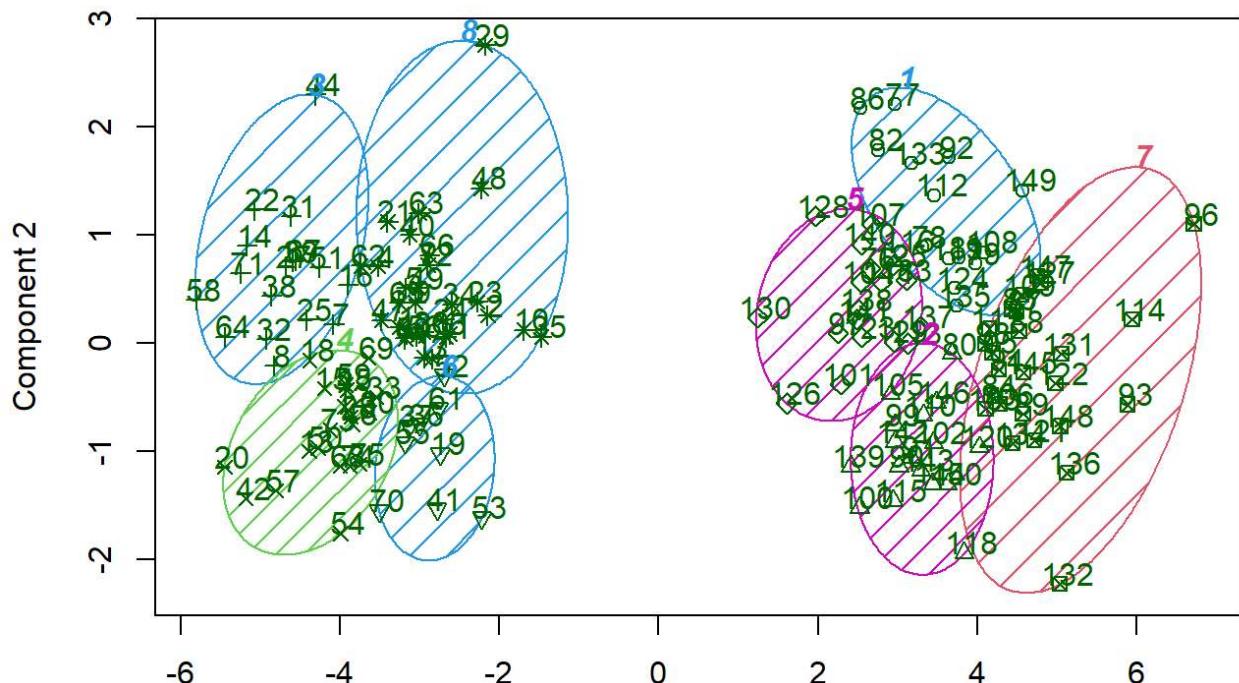

```
clusplot(x, fit4$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



y con 8

```
clusplot(x, fit8$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```

CLUSPLOT(x)

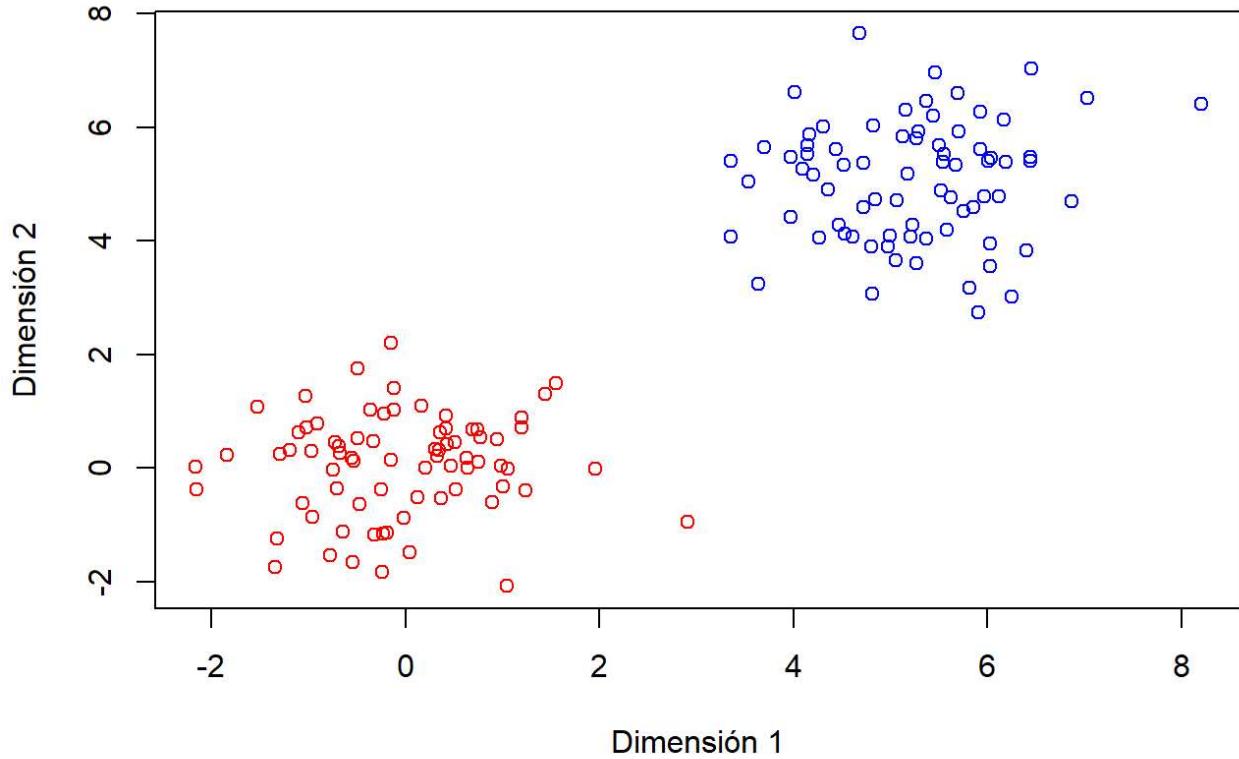


Component 1

These two components explain 100 % of the point variability.

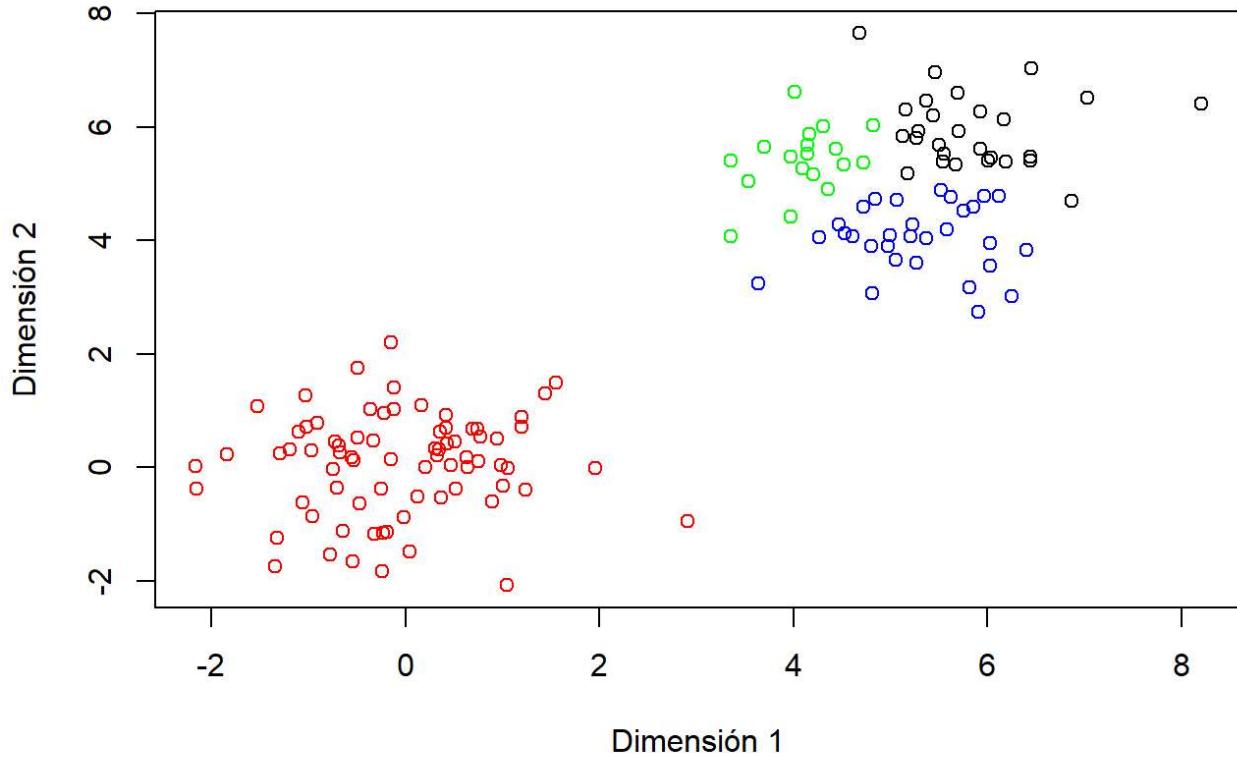
También podemos visualizar el resultado del proceso de agrupamiento con el siguiente código para el caso de 2 clústeres

```
plot(x[y_cluster2==1], col='blue', xlim=c(min(x[,1]), max(x[,1])), ylim=c(min(x[,2]), max(x[,2])), xlab = "Dimensión 1", ylab = "Dimensión 2")
points(x[y_cluster2==2], col='red')
```



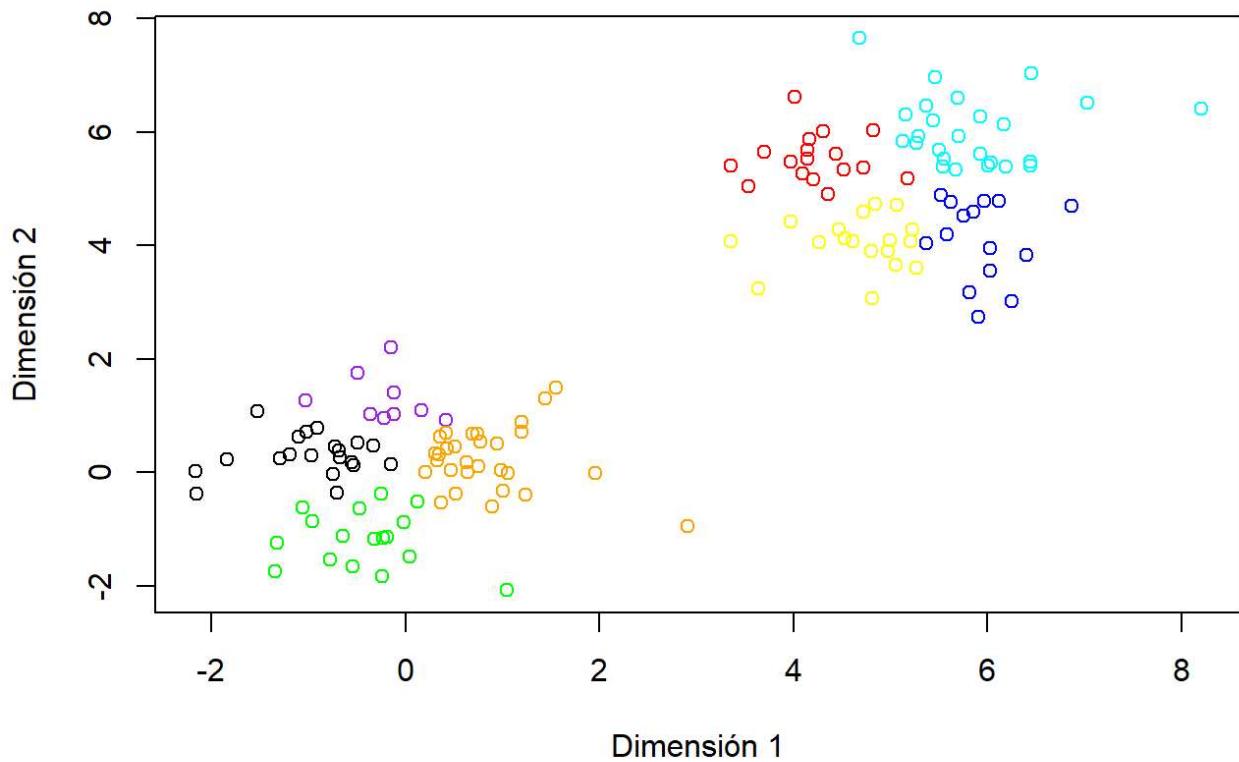
para 4

```
plot(x[y_cluster4==1],col='blue', xlim=c(min(x[,1]), max(x[,1])), ylim=c(min(x[,2]), ma  
x(x[,2])), xlab = "Dimensión 1", ylab = "Dimensión 2")  
points(x[y_cluster4==2],col='red')  
points(x[y_cluster4==3],col='green')  
points(x[y_cluster4==4],col='black')
```



y para 8

```
plot(x[y_cluster8==1], col='blue', xlim=c(min(x[,1]), max(x[,1])), ylim=c(min(x[,2]), ma
x(x[,2])), xlab = "Dimensión 1", ylab = "Dimensión 2")
points(x[y_cluster8==2], col='red')
points(x[y_cluster8==3], col='green')
points(x[y_cluster8==4], col='black')
points(x[y_cluster8==5], col='yellow')
points(x[y_cluster8==6], col='purple')
points(x[y_cluster8==7], col='cyan')
points(x[y_cluster8==8], col='orange')
```



Ahora vamos a evaluar la calidad del proceso de agregación. Para ello usaremos la función silhouette que calcula la silueta de cada muestra

```
d <- daisy(x)
sk2 <- silhouette(y_cluster2, d)
sk4 <- silhouette(y_cluster4, d)
sk8 <- silhouette(y_cluster8, d)
```

La función silhouette devuelve para cada muestra, el clúster dónde ha sido asignado, el clúster vecino y el valor de la silueta. Por lo tanto, calculando la media de la tercera columna podemos obtener una estimación de la calidad del agrupamiento

```
mean(sk2[,3])
```

```
## [1] 0.7665411
```

```
mean(sk4[,3])
```

```
## [1] 0.5731342
```

```
mean(sk8[,3])
```

```
## [1] 0.3695492
```

Como se puede comprobar, agrupar con dos clúster es mejor que en 4 o en 8, lo cual es lógico teniendo en cuenta como se han generado los datos.

3 Ejemplo 1.2

3.1 Método de agregación k-means con datos reales

A continuación vamos a ver otro ejemplo de cómo se usan los modelos de agregación. Para ello usaremos el data set **penguins** contenido en el paquete R **palmerpenguins**. Esta base de datos se encuentra descrita en <https://cran.r-project.org/web/packages/palmerpenguins/index.html> (<https://cran.r-project.org/web/packages/palmerpenguins/index.html>) y contiene mediciones de tamaño, observaciones de puestas y proporciones de isótopos sanguíneos de tres especies de pingüinos observadas en tres islas del archipiélago Palmer, en la Antártida, durante un período de estudio de tres años.

Este dataset está previamente trabajado para que los datos estén limpios y sin errores. De no ser así antes de nada deberíamos buscar errores, valores nulos u *outliers*. Deberíamos tratar de discretizar o eliminar columnas. Incluso realizar este último paso varias veces para comprobar los diferentes resultados y elegir el que mejor rendimiento nos dé. De todos modos contiene algún valor nulo que procederemos a ignorar.

Vamos a visualizar la estructura y resumen de los datos

```
if (!require('palmerpenguins')) install.packages('palmerpenguins')
library(palmerpenguins)
palmerpenguins::penguins
```


supervisado original en uno **no supervisado**. Para conseguirlo no usaremos la columna *species*, que es la variable que se quiere predecir. Por lo tanto, intentaremos encontrar agrupaciones usando únicamente los cuatro atributos numéricos que caracterizan a cada especie de pingüino.

Cargamos los datos y nos quedamos únicamente con las cuatro columnas que definen a cada especie.

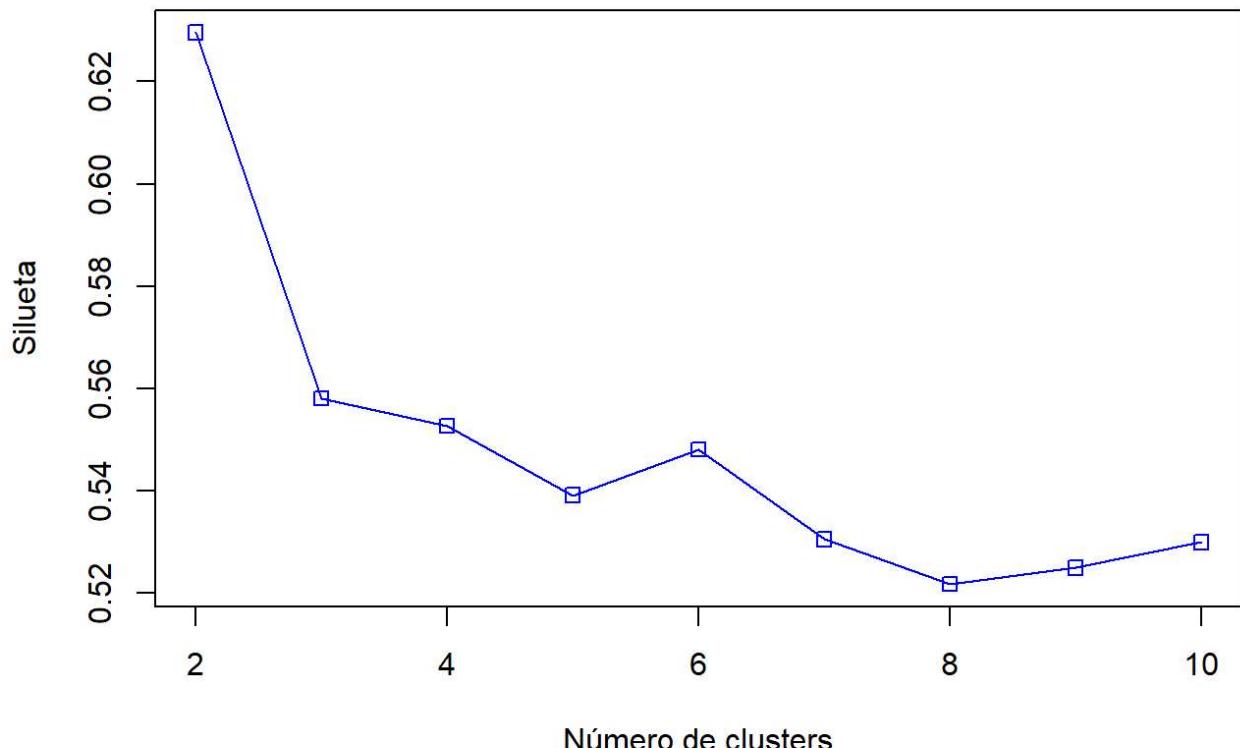
```
x <- na.omit(penguins[,3:6])
```

Como inicialmente no conocemos el número óptimo de clústers, probamos con varios valores

```
d <- daisy(x)
resultados <- rep(0, 10)
for (i in c(2,3,4,5,6,7,8,9,10))
{
  fit       <- kmeans(x, i)
  y_cluster <- fit$cluster
  sk        <- silhouette(y_cluster, d)
  resultados[i] <- mean(sk[,3])
}
```

Mostramos en un gráfica los valores de las siluetas media de cada prueba para comprobar qué número de clústers es el mejor.

```
plot(2:10, resultados[2:10], type="o", col="blue", pch=0, xlab="Número de clusters", ylab="Silueta")
```



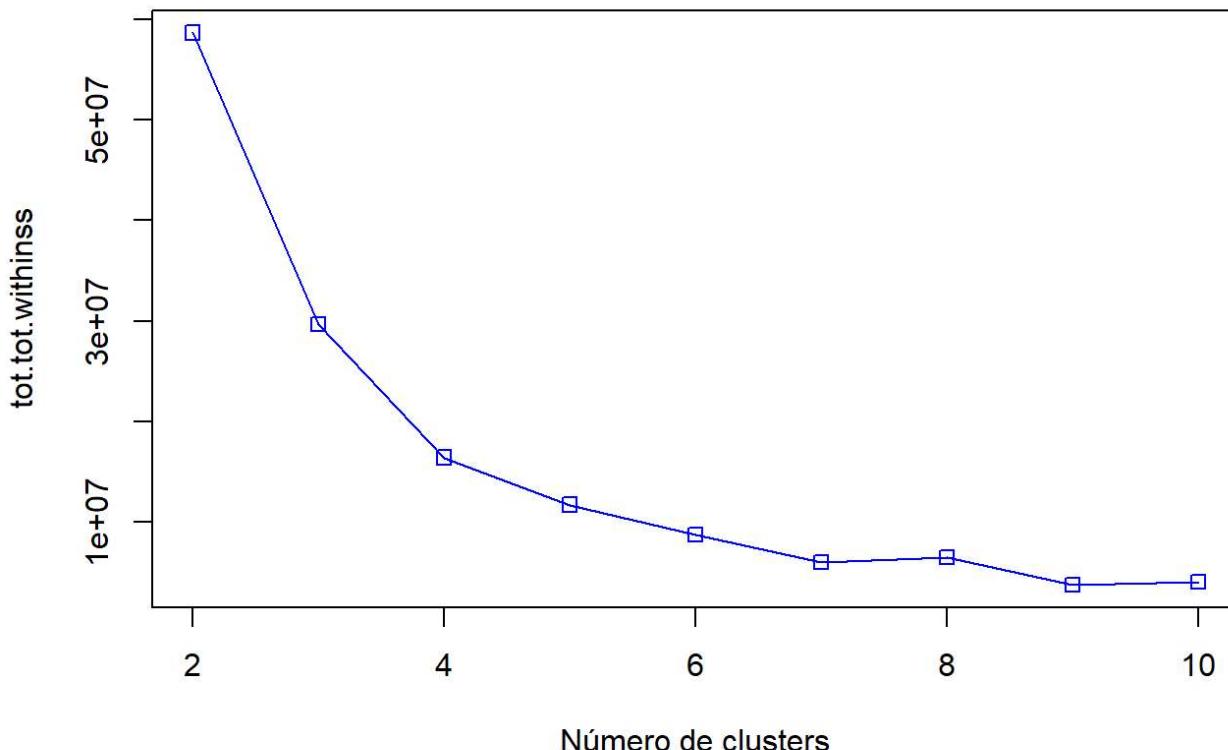
Tal y como era de esperar la mejora más significativa se obtiene para $k=3$.

Otro forma de evaluar cual es el mejor número de clústers es considerar el mejor modelo, aquel que ofrece la menor suma de los cuadrados de las distancias de los puntos de cada grupo con respecto a su centro (*withinss*), con la mayor separación entre centros de grupos (*betweenss*). Como se puede comprobar es una idea conceptualmente similar a la silueta. Una manera común de hacer la selección del número de clústers consiste en aplicar el método *elbow* (codo), que no es más que la selección del número de clústers en base a la inspección de la gráfica que se obtiene al iterar con el mismo conjunto de datos para distintos valores del número de clústers. Se seleccionará el valor que se encuentra en el “codo” de la curva.

```

resultados <- rep(0, 10)
for (i in c(2,3,4,5,6,7,8,9,10))
{
  fit          <- kmeans(x, i)
  resultados[i] <- fit$tot.withinss
}
plot(2:10,resultados[2:10],type="o",col="blue",pch=0,xlab="Número de clusters",ylab="tot.tot.withinss")

```



En este caso el número óptimo de clústers son 4 que es cuando la curva comienza a estabilizarse.

También se puede usar la función *kmeansruns* del paquete **fpc** que ejecuta el algoritmo *kmeans* con un conjunto de valores, para después seleccionar el valor del número de clústers que mejor funcione de acuerdo a dos criterios: la silueta media (“asw”) y *Calinski-Harabasz* (“ch”).

```

if (!require('fpc')) install.packages('fpc')
library(fpc)
fit_ch <- kmeansruns(x, krange = 1:10, criterion = "ch")
fit_asw <- kmeansruns(x, krange = 1:10, criterion = "asw")

```

Podemos comprobar el valor con el que se ha obtenido el mejor resultado y también mostrar el resultado obtenido para todos los valores de k usando ambos criterios

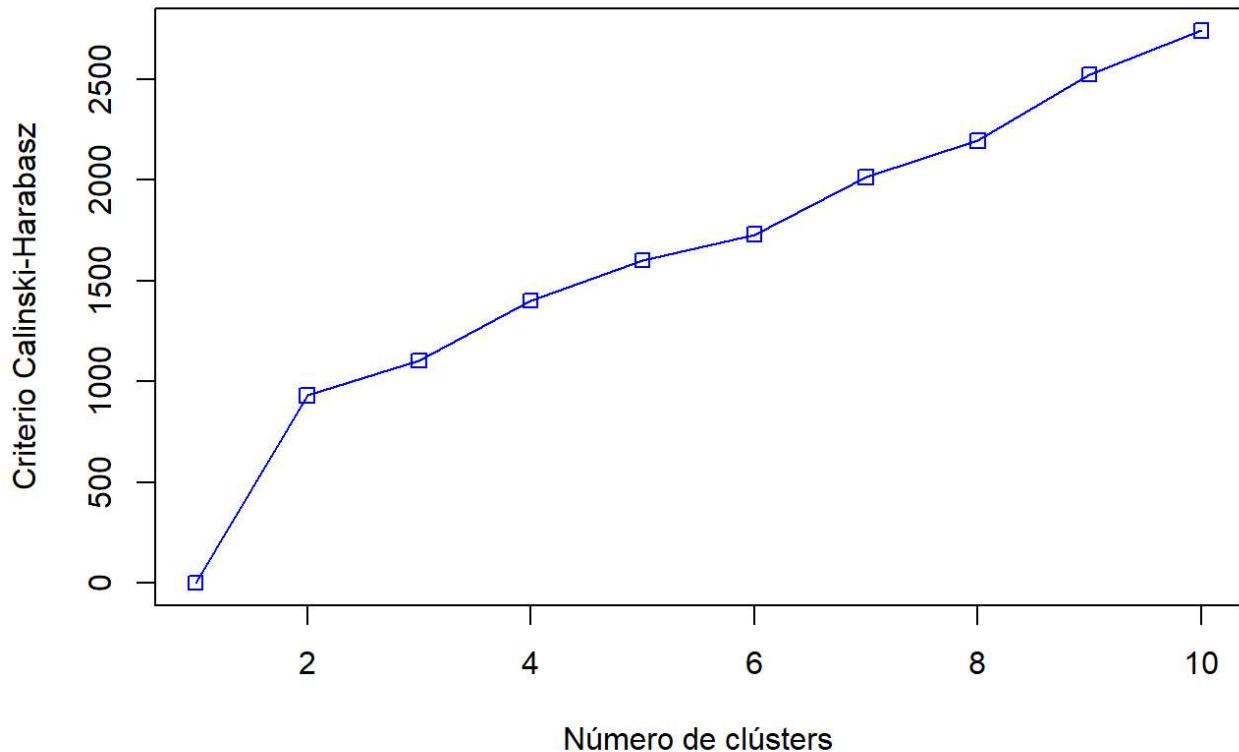
```
fit_ch$bestk
```

```
## [1] 10
```

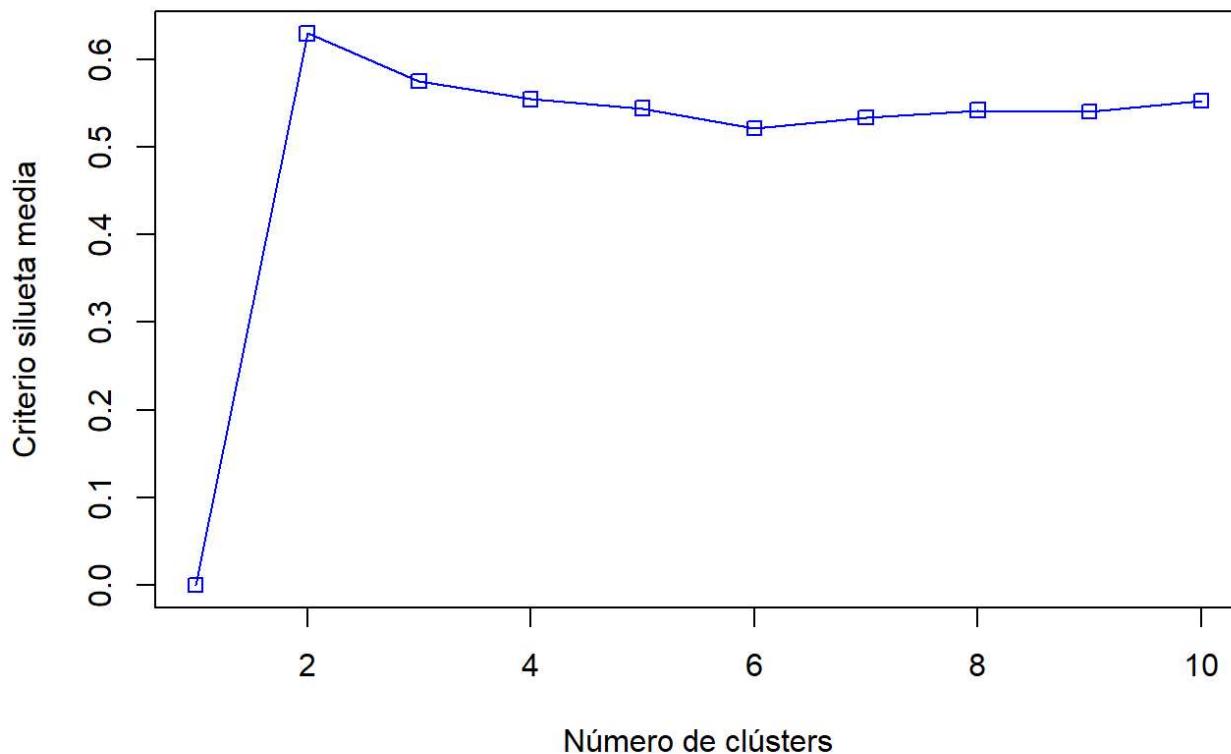
```
fit_asw$bestk
```

```
## [1] 2
```

```
plot(1:10,fit_ch$crit,type="o",col="blue",pch=0,xlab="Número de clústers",ylab="Criterio Calinski-Harabasz")
```



```
plot(1:10,fit_asw$crit,type="o",col="blue",pch=0,xlab="Número de clústers",ylab="Criterio silueta media")
```



Los resultados son muy parecidos a los que hemos obtenido anteriormente. Con el criterio de la silueta media se obtienen dos clústers y con el *Calinski-Harabasz* se obtienen 3.

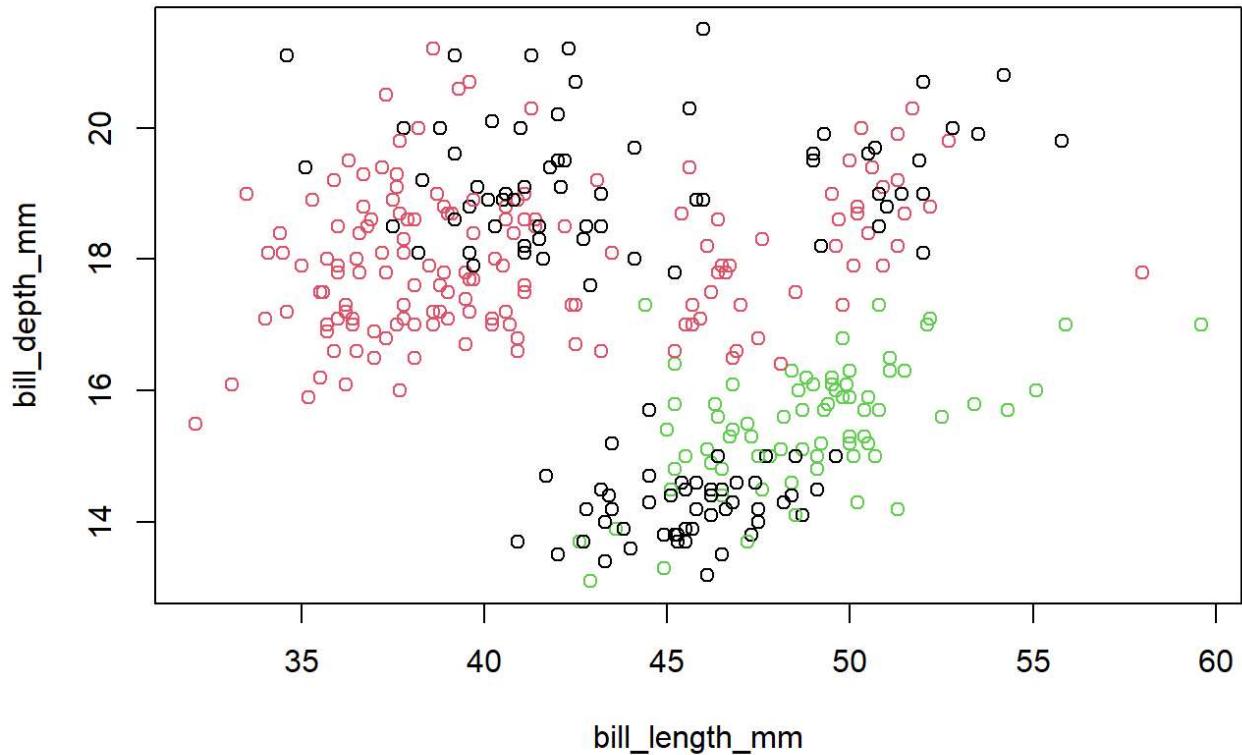
Como se ha comprobado, conocer el número óptimo de clústers no es un problema fácil. Tampoco lo es la evaluación de los modelos de agregación.

Como en el caso que estudiamos sabemos que los datos pueden ser agrupados en 3 clases o especies, vamos a ver cómo se ha comportado *kmeans* en el caso de pedirle 3 clústers. Para eso comparamos visualmente los campos dos a dos, con el valor real que sabemos está almacenado en el campo “species” del dataset original.

```
penguins3clusters <- kmeans(x, 3)

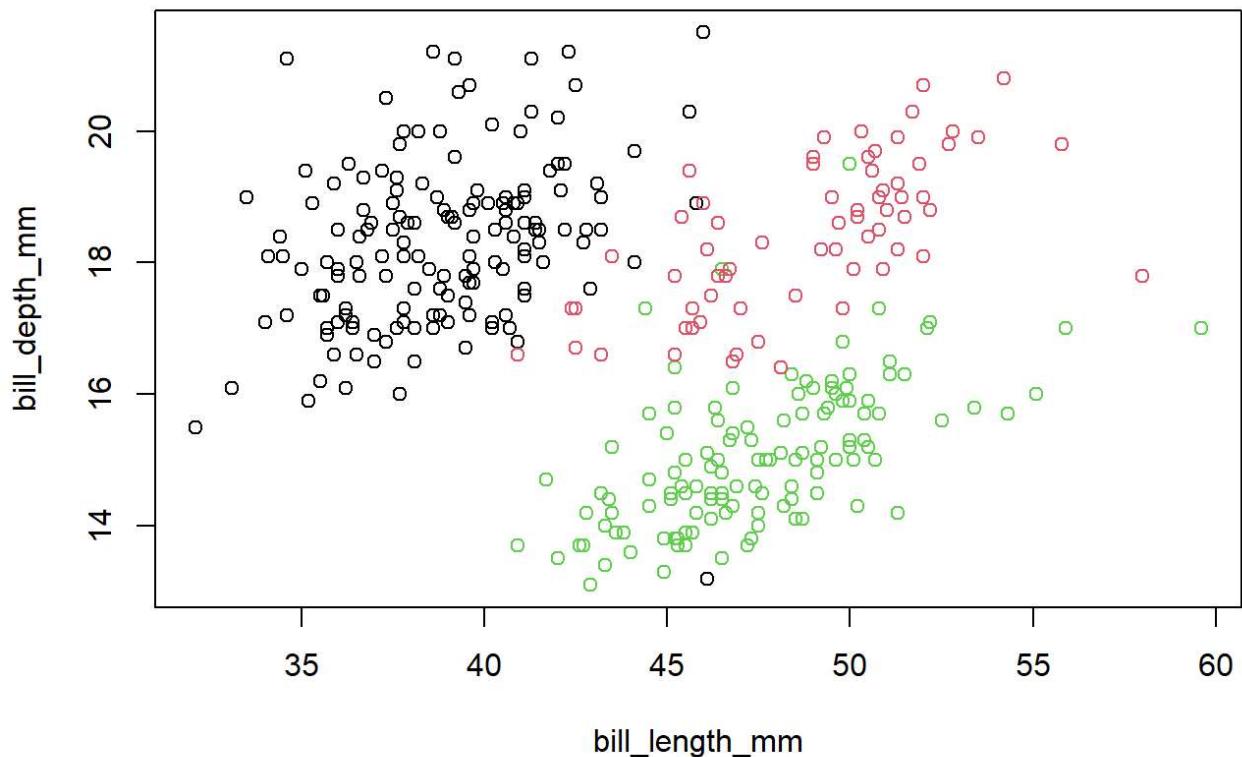
# bill_LLength y bill_depth
plot(x[c(1,2)], col=penguins3clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



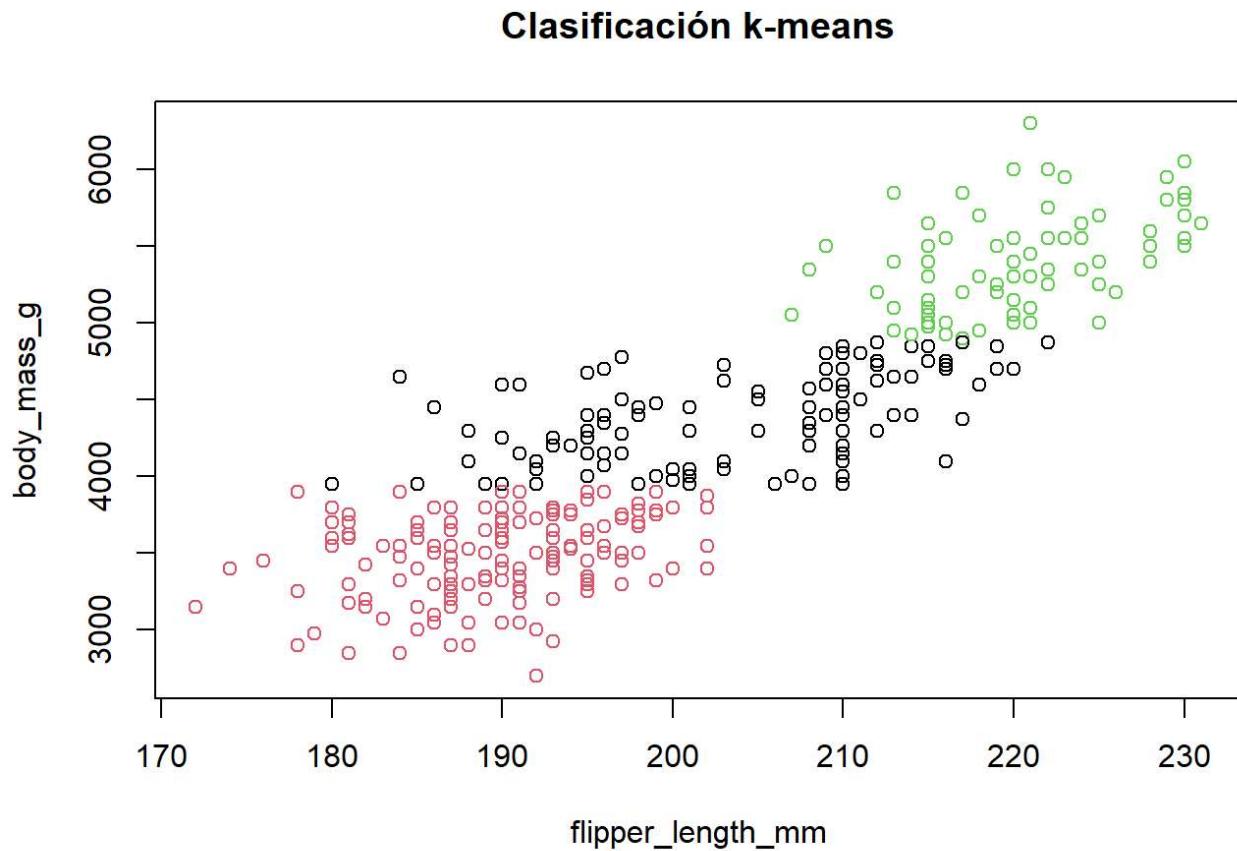
```
plot(x[c(1,2)], col=as.factor(penguins$species), main="Clasificación real")
```

Clasificación real



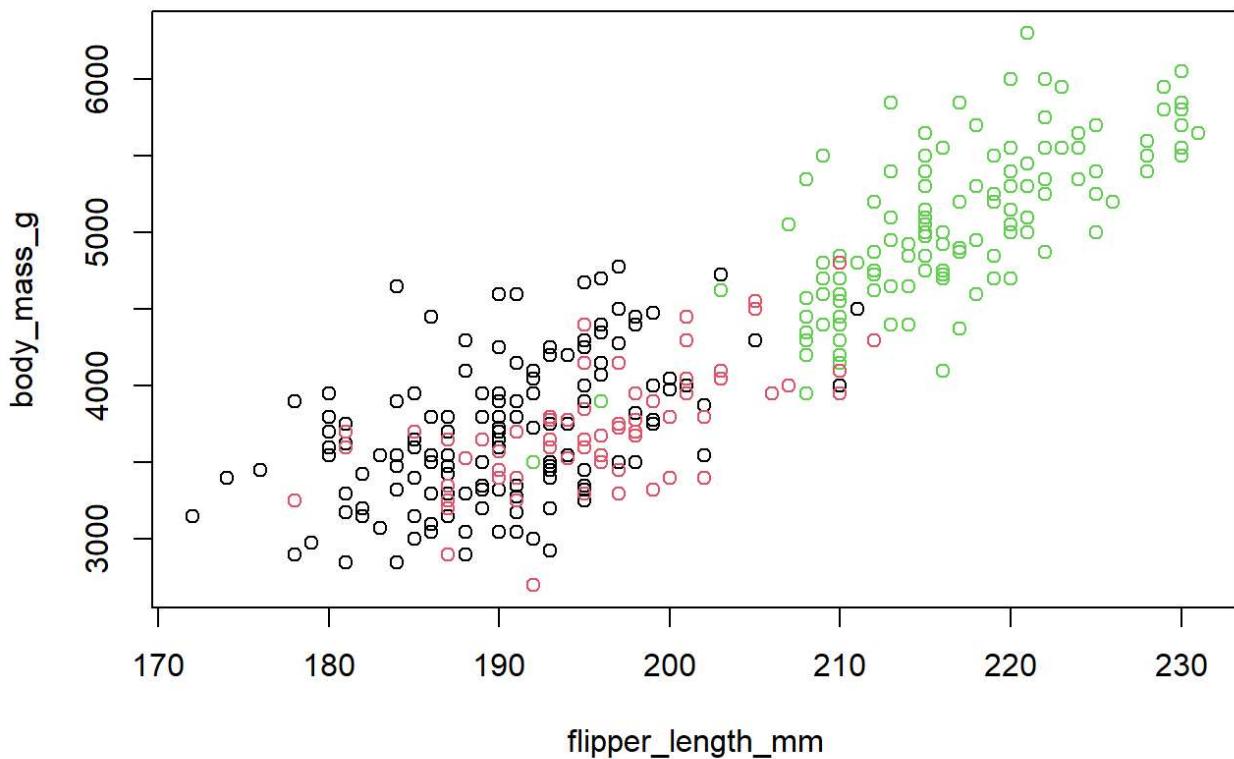
Podemos observar que *flipper_length* y *body_mass* no son buenos indicadores para diferenciar a las tres subespecies, dado que dos de las subespecies están demasiado mezcladas para poder diferenciar nada.

```
# flipper_length y body_mass  
plot(x[c(3,4)], col=penguins3clusters$cluster, main="Clasificación k-means")
```



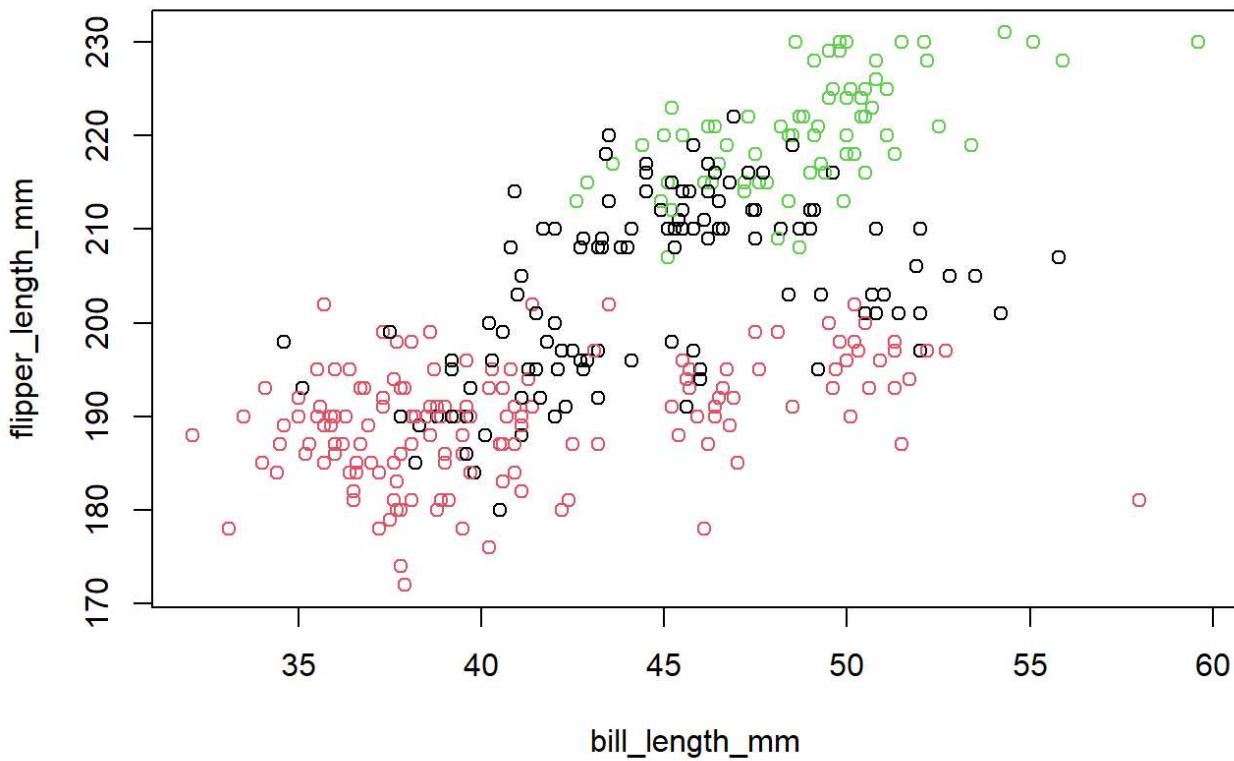
```
plot(x[c(3,4)], col=as.factor(penguins$species), main="Clasificación real")
```

Clasificación real

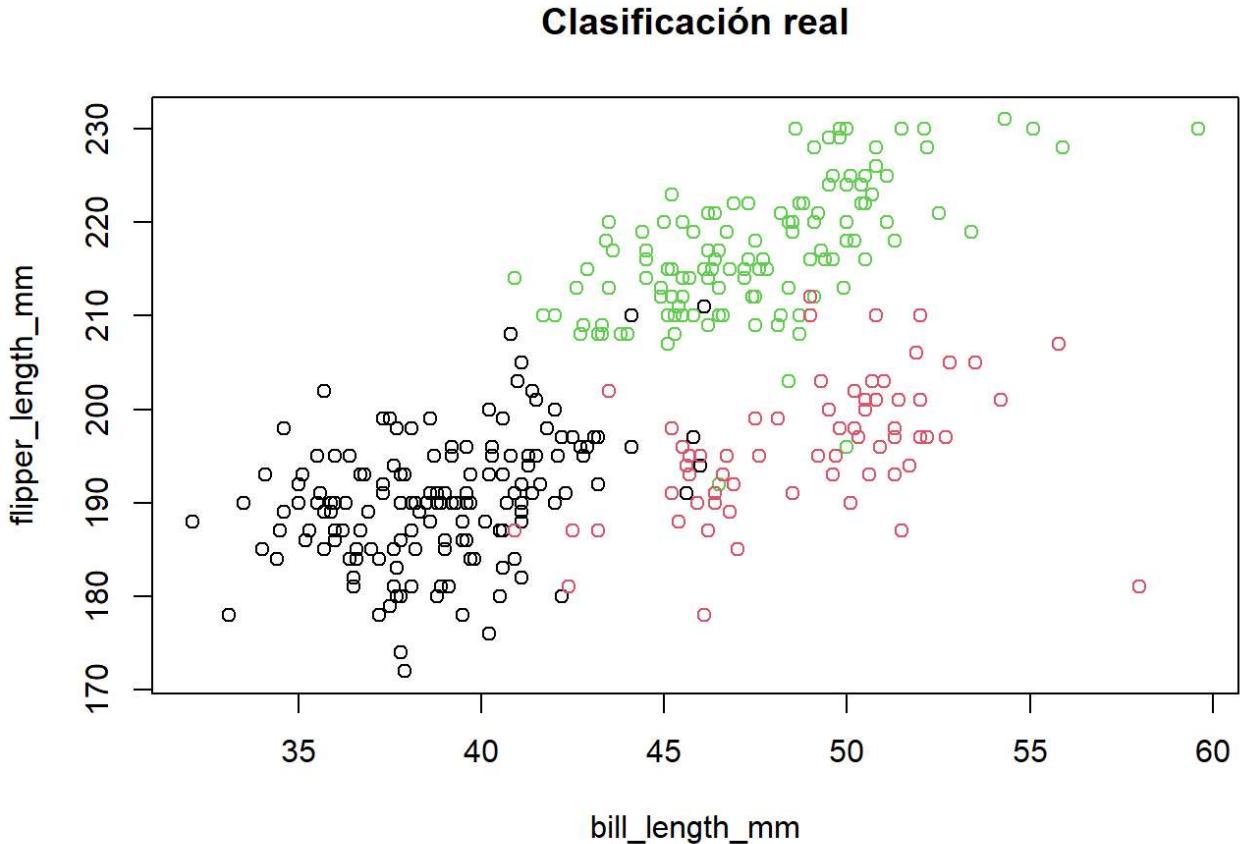


```
# bill_length y flipper_length  
plot(x[c(1,3)], col=penguins3clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



```
plot(x[c(1,3)], col=as.factor(penguins$species), main="Clasificación real")
```



Las dos medidas de *bill* parecen lograr mejores resultados al dividir las tres especies de pingüinos. El grupo formado por los puntos negros que ha encontrado el algoritmo coincide con los de la especie *Adelie*. Los otros dos grupos sin embargo se entremezclan algo más, y hay ciertos puntos que se clasifican como *Gentoo* (verde) cuando en realidad son *Chinstrap* (rojo).

Una buena técnica que ayuda a entender los grupos que se han formado, es mirar de darles un nombre. Cómo por ejemplo:

- Grupo 1: Sólo *Adelie* (color negro)
- Grupo 2: Principalmente *Chinstrap* (color rojo)
- Grupo 3: Mezcla de *Gentoo* (color verde) y *Adelie* (color negro)

Esto nos ayuda a entender cómo están formados los grupos y a referirnos a ellos en análisis posteriores.

Como continuación del estudio podríamos seguir experimentando combinando en gráficos similares a los anteriores. En definitiva se trataría en este punto de profundizar más en el conocimiento de las propiedades de las diferentes características o columnas del juego de datos.

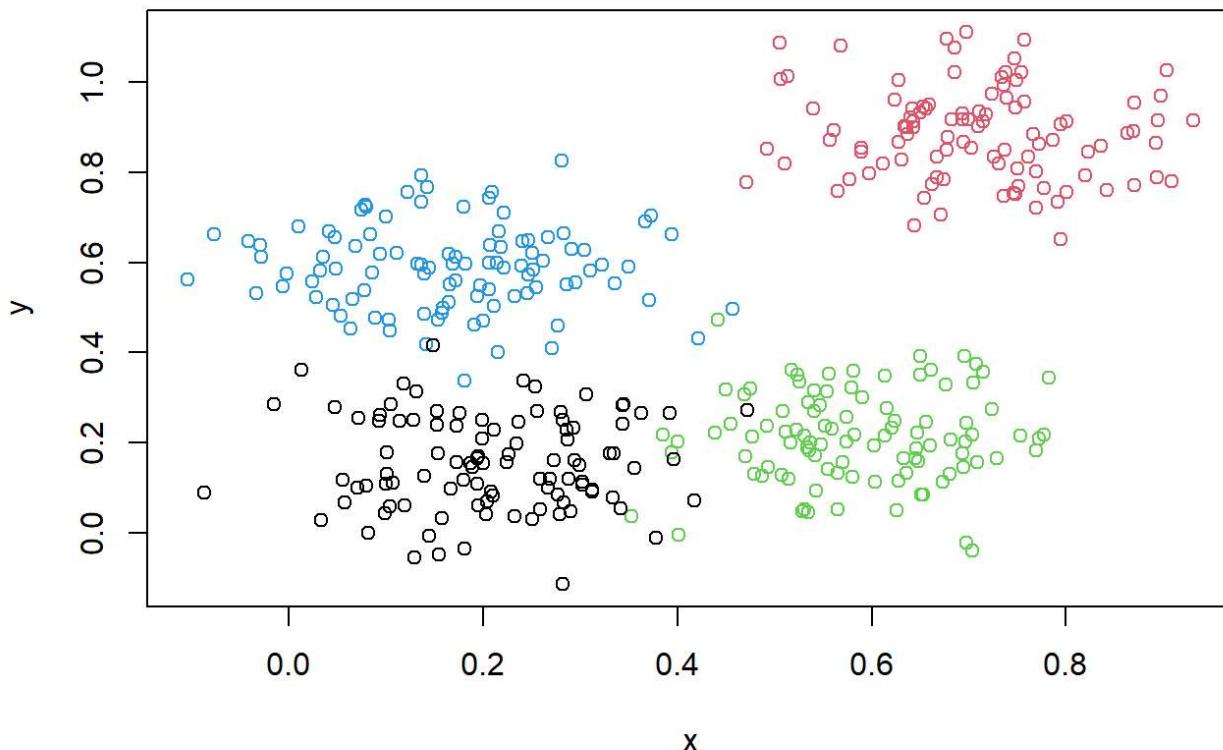
4 Ejemplo 2

4.1 Métodos basados en densidad: DBSCAN y OPTICS

En este ejemplo vamos a trabajar los algoritmos **DBSCAN** y **OPTICS** como métodos de clustering que permiten la generación de grupos no radiales a diferencia de k-means. Veremos que su parámetro de entrada más relevante es *minPts* que define la mínima densidad aceptada alrededor de un centroide.

Incrementar este parámetro nos permitirá reducir el ruido (observaciones no asignadas a ningún cluster), en cualquier caso empezaremos por construir nuestro propio juego de datos en el que dibujaremos 4 zonas de puntos diferenciadas.

```
if (!require('dbSCAN')) install.packages('dbSCAN')
library(dbSCAN)
set.seed(2)
n <- 400
x <- cbind(
  x = runif(4, 0, 1) + rnorm(n, sd=0.1),
  y = runif(4, 0, 1) + rnorm(n, sd=0.1)
)
plot(x, col=rep(1:4, time = 100))
```



Una de las primeras actividades que realiza el algoritmo es **ordenar las observaciones** de forma que los puntos más cercanos se conviertan en vecinos en el ordenamiento. Se podría pensar como una representación numérica del dendograma de una agrupación jerárquica.

```
### Lanzamos el algoritmo OPTICS dejando el parámetro eps con su valor por defecto y fijando el criterio de vecindad en 10
res <- optics(x, minPts = 10)
res
```

```
## OPTICS ordering/clustering for 400 objects.
## Parameters: minPts = 10, eps = 0.193786846197958, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi
```

Obtenemos La ordenación de Las observaciones o puntos
res\$order

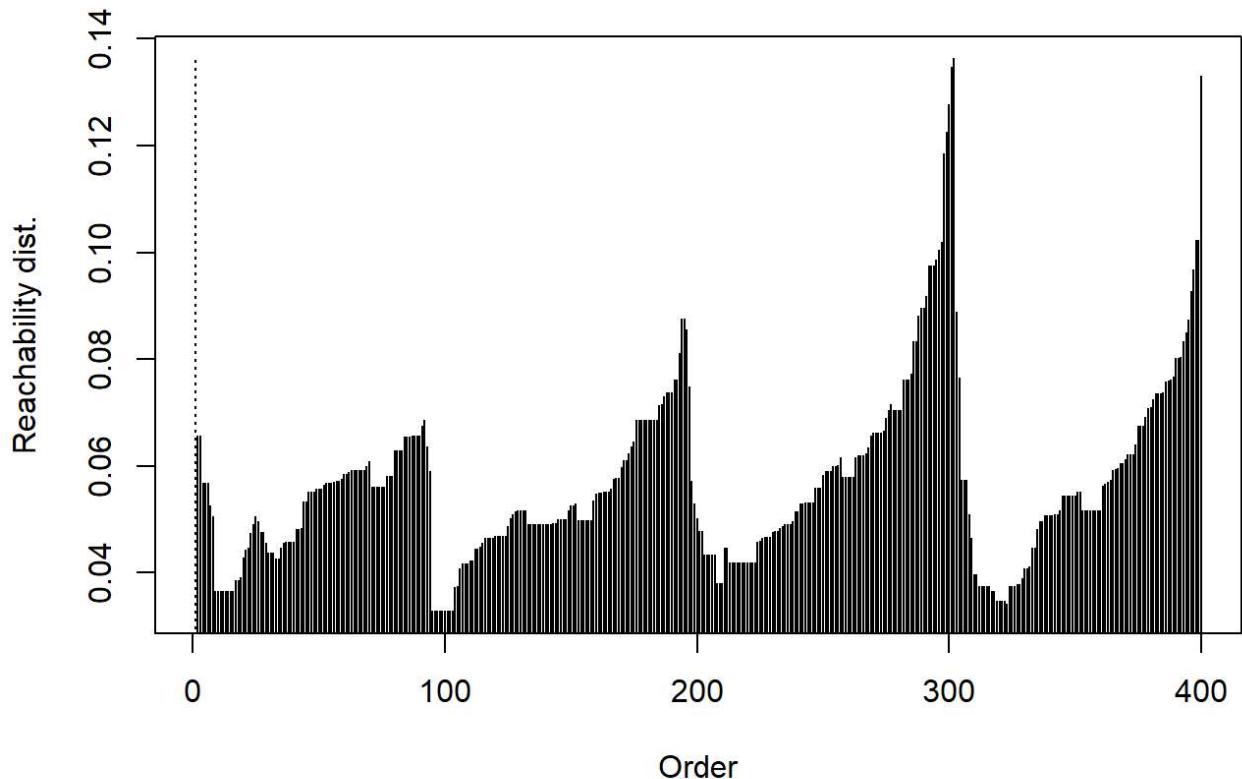
```
## [1] 1 363 209 349 337 301 357 333 321 285 281 253 241 177 153 57 257 29 77 169  
105 293 229 145 181 385 393 377 317 381 185 117 101 9 73 237 397 369 365 273 305 245  
249 309 157 345 213  
## [48] 205 97 49 33 41 193 149 17 83 389 25 121 329 5 161 341 217 189 141 85  
53 225 313 289 261 221 173 69 61 297 125 81 133 129 197 109 137 59 93 165 89 21  
13 277 191 203 379  
## [95] 399 375 351 311 235 231 227 71 11 299 271 291 147 55 23 323 219 275 47 263  
3 367 331 175 87 339 319 251 247 171 111 223 51 63 343 303 207 151 391 359 287 283 21  
5 143 131 115 99  
## [142] 31 183 43 243 199 79 27 295 67 347 255 239 195 187 139 107 39 119 179 395  
371 201 123 159 91 211 355 103 327 95 7 167 35 267 155 387 383 335 315 259 135 15  
113 279 373 4 353  
## [189] 265 127 45 37 19 276 224 361 260 288 336 368 348 292 268 252 120 108 96 88  
32 16 340 156 388 372 356 332 304 220 188 168 136 124 56 236 28 244 392 184 76 380 2  
32 100 116 112 256  
## [236] 72 8 280 64 52 208 172 152 148 360 352 192 160 144 284 216 48 84 92 36  
20 212 272 264 200 128 80 180 364 196 12 132 40 324 308 176 164 68 316 312 384 300 3  
44 328 248 204 140  
## [283] 296 24 320 228 60 44 233 65 400 376 240 163 104 396 307 75 14 325 269 262  
234 382 294 206 198 374 310 362 318 386 358 330 278 210 298 282 122 98 34 26 174 142  
46 6 62 118 190  
## [330] 202 114 322 286 38 242 394 342 266 162 130 30 182 2 74 314 290 246 194 170  
126 158 378 350 254 226 214 70 18 10 366 354 186 150 86 306 102 338 346 134 250 138  
94 78 390 274 58  
## [377] 42 258 66 90 146 370 222 218 326 82 110 270 334 178 166 398 22 50 238 106  
154 302 230 54
```

Otro paso muy interesante del algoritmo es la generación de un **diagrama de alcanzabilidad** o *reachability plot*, en el que se aprecia de una forma visual la distancia de alcanzabilidad de cada punto.

Los valles representan clusters (cuanto más profundo es el valle, más denso es el cluster), mientras que las cimas indican los puntos que están entre las agrupaciones (estos puntos son candidatos a ser considerados *outliers*)

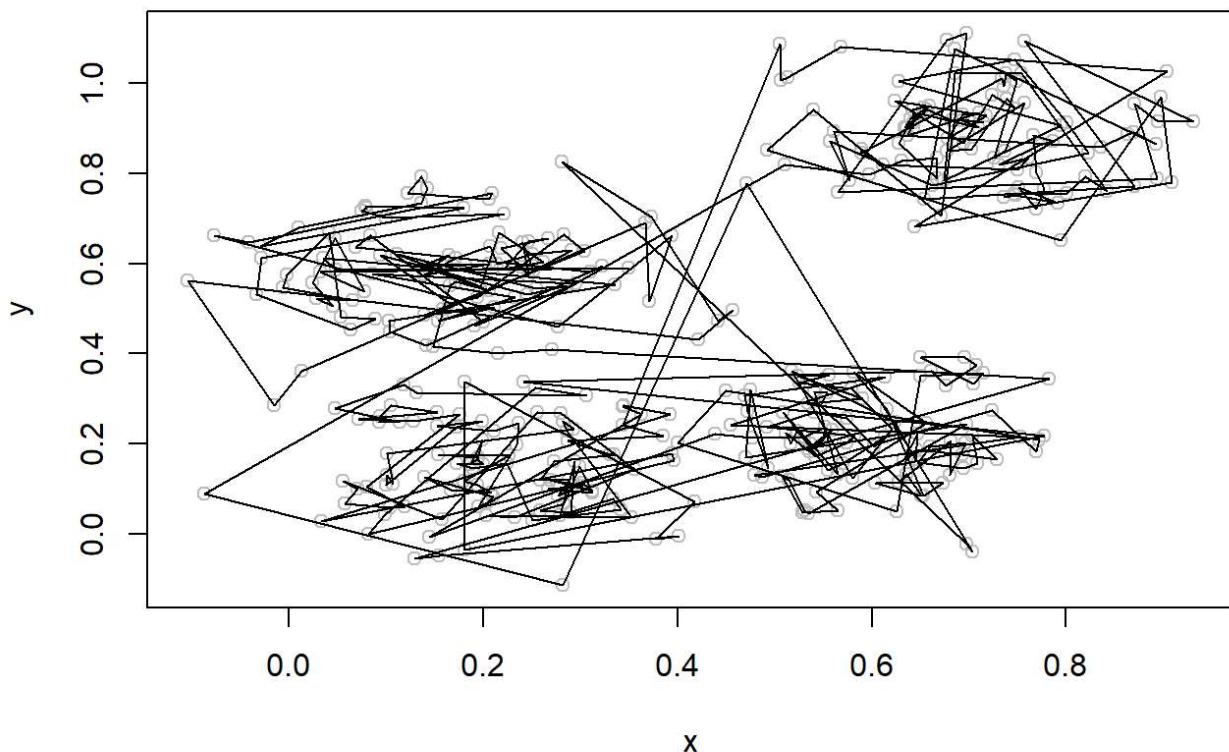
```
### Gráfica de alcanzabilidad
plot(res)
```

Reachability Plot



Veamos otra representación del diagrama de alcanzabilidad, donde podemos observar las trazas de las distancias entre puntos cercanos del mismo cluster y entre clusters distintos.

```
### Dibujo de Las trazas que relacionan puntos
plot(x, col = "grey")
polygon(x[res$order,])
```



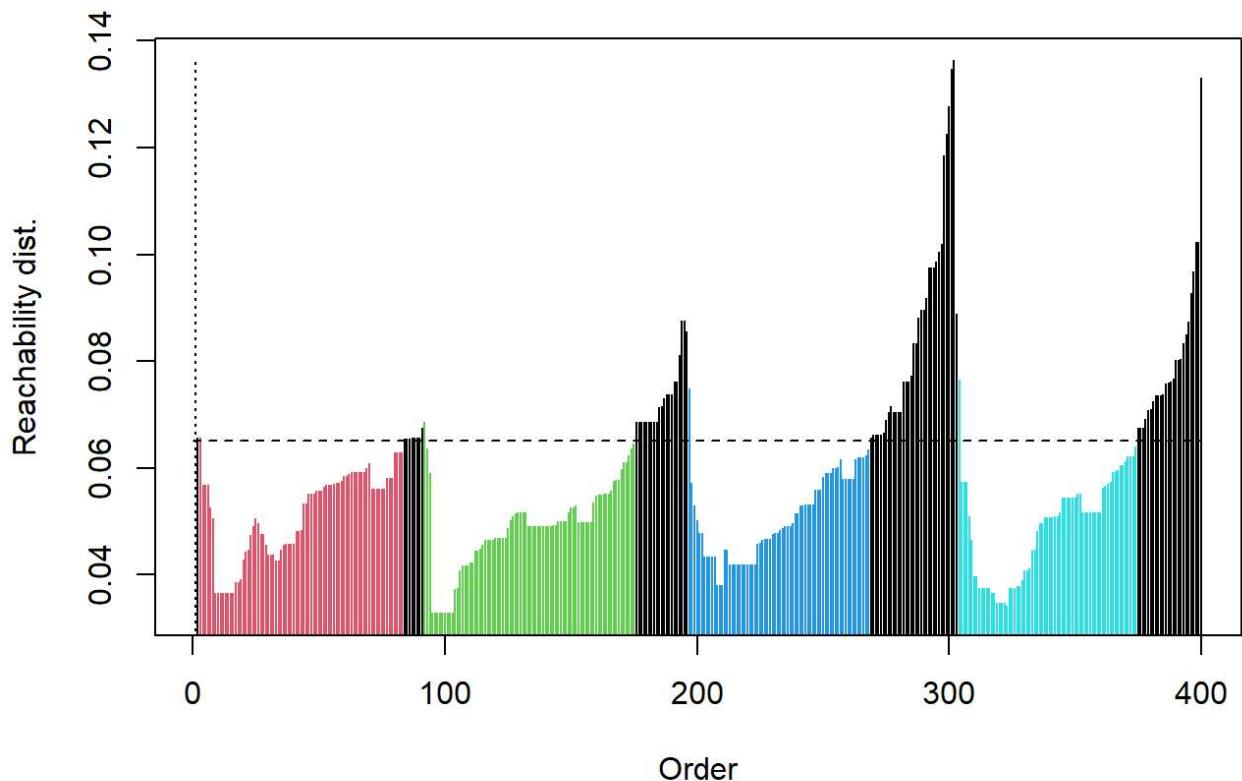
Otro ejercicio interesante a realizar es extraer una agrupación de la ordenación realizada por OPTICS similar a lo que DBSCAN hubiera generado estableciendo el parámetro eps en `eps_cl = 0.065`

```
### Extracción de un clustering DBSCAN cortando La alcancabilidad en el valor eps_cl
res <- extractDBSCAN(res, eps_cl = .065)
res
```

```
## OPTICS ordering/clustering for 400 objects.
## Parameters: minPts = 10, eps = 0.193786846197958, eps_cl = 0.065, xi = NA
## The clustering contains 4 cluster(s) and 92 noise points.
##
##  0  1  2  3  4
## 92 81 84 72 71
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster
```

```
plot(res) ## negro indica ruido
```

Reachability Plot

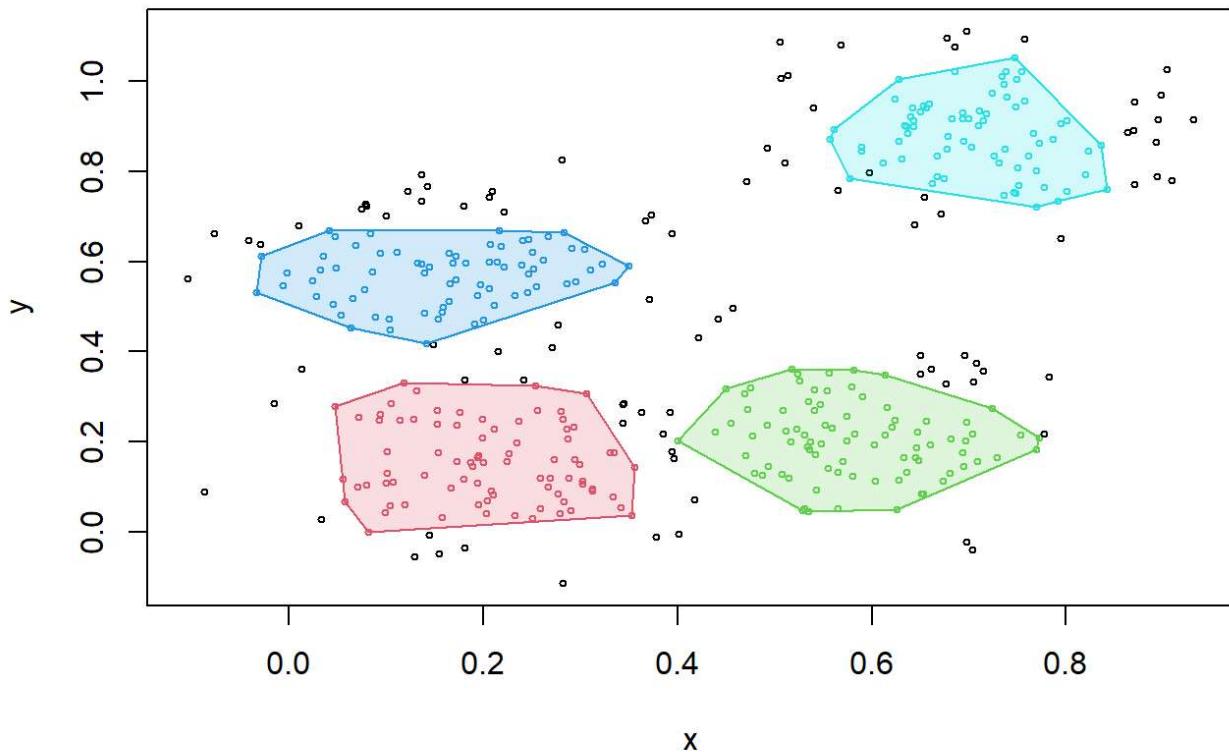


Observamos en el gráfico anterior como se han coloreado los 4 clusters y en negro se mantienen los valores *outliers*.

Seguimos adelante con una representación gráfica que nos muestra los clusters mediante formas convexas.

```
hullplot(x, res)
```

Convex Cluster Hulls



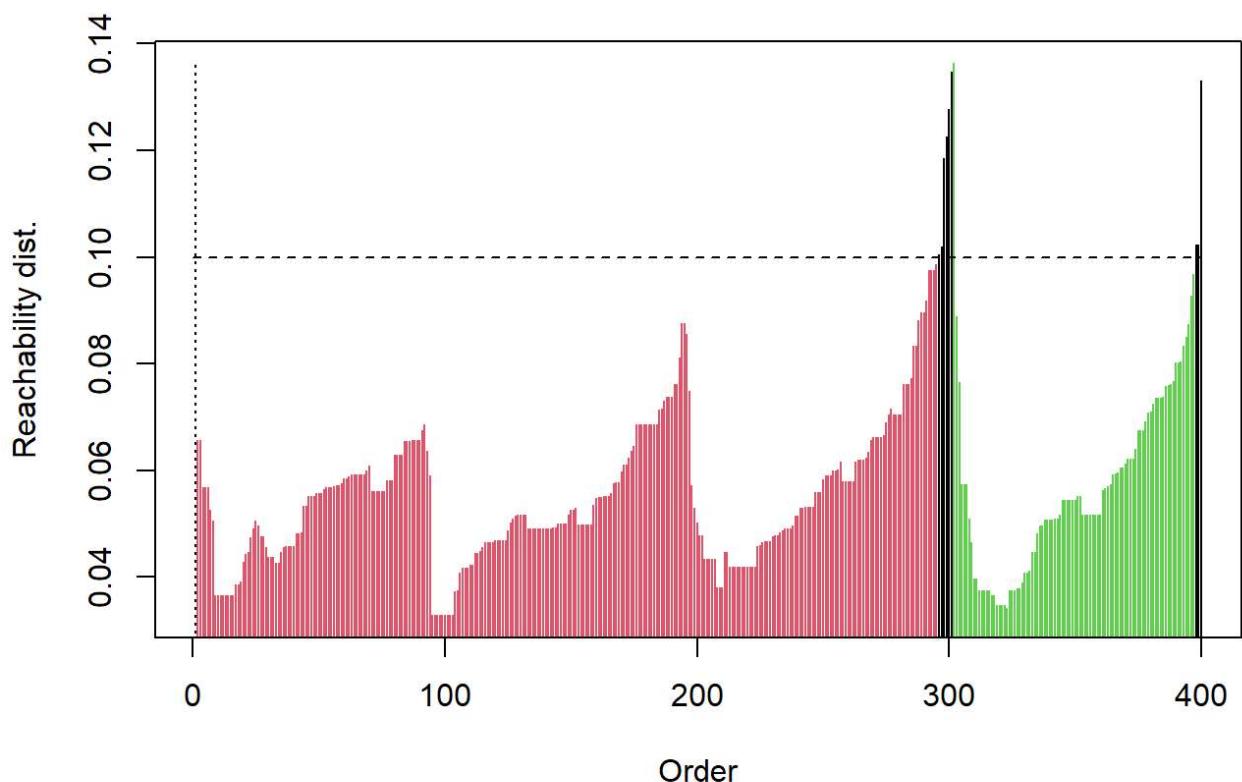
Repetimos el experimento anterior incrementando el parámetro *epc_c*, veamos como el efecto que produce es la concentración de clusters ya que flexibilizamos la condición de densidad.

```
### Incrementamos el parámetro eps
res <- extractDBSCAN(res, eps_cl = .1)
res
```

```
## OPTICS ordering/clustering for 400 objects.
## Parameters: minPts = 10, eps = 0.193786846197958, eps_cl = 0.1, xi = NA
## The clustering contains 2 cluster(s) and 9 noise points.
##
##    0    1    2
##  9 295  96
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster
```

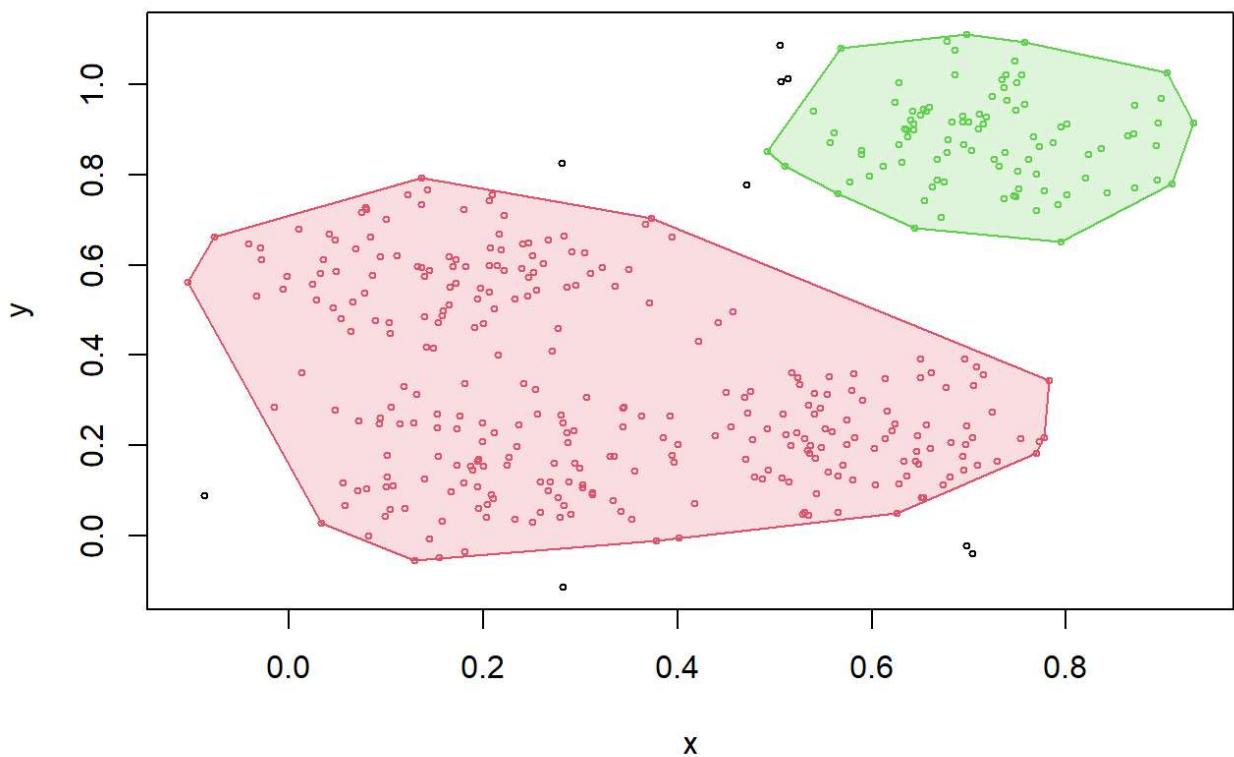
```
plot(res)
```

Reachability Plot



```
hullplot(x, res)
```

Convex Cluster Hulls

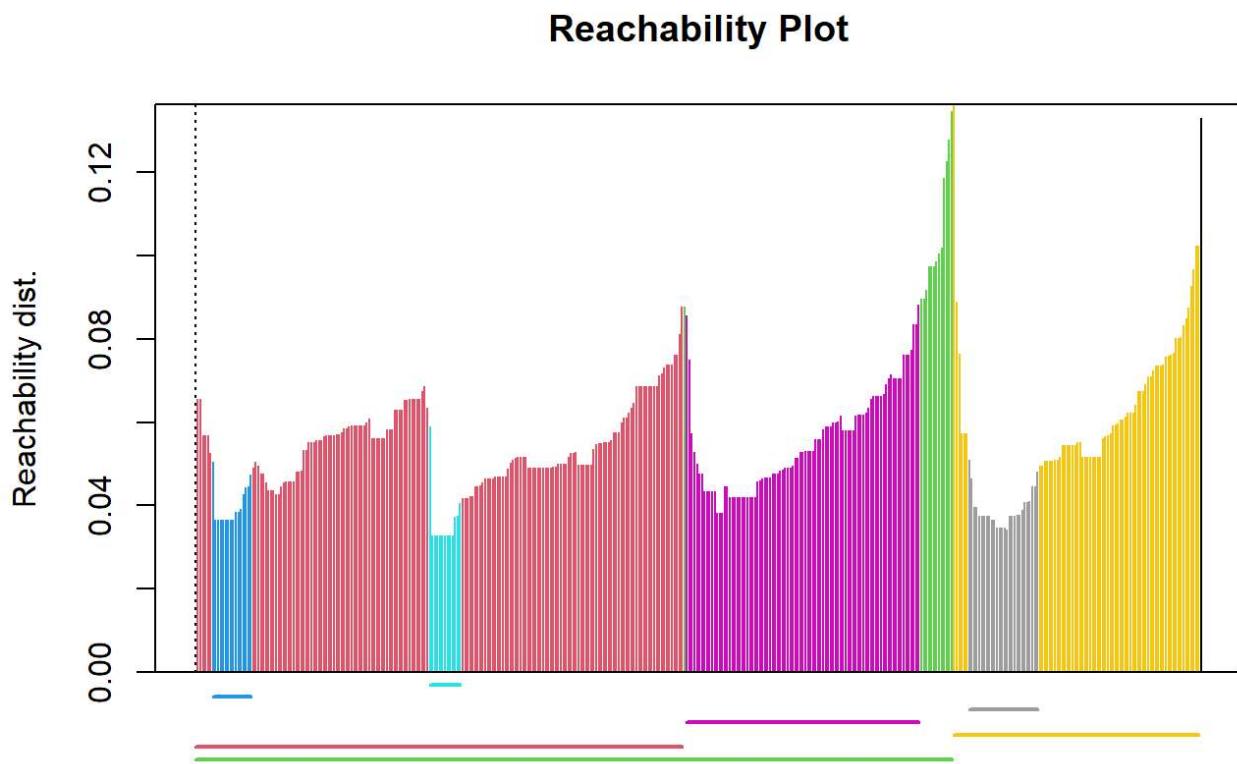


Veamos ahora una variante de la extracción **DBSCN** anterior. En ella el parámetro ξ nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

```
### Extracción del clustering jerárquico en función de la variación de la densidad por el método xi
res <- extractXi(res, xi = 0.05)
res
```

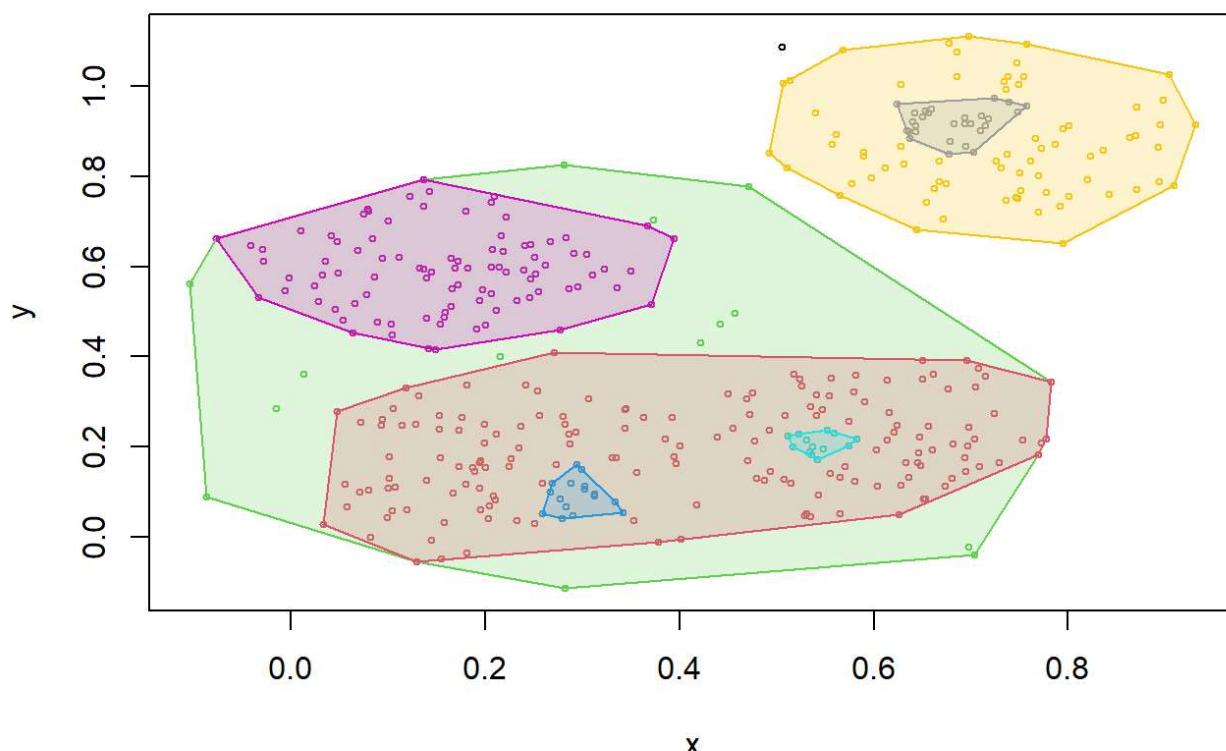
```
## OPTICS ordering/clustering for 400 objects.
## Parameters: minPts = 10, eps = 0.193786846197958, eps_cl = NA, xi = 0.05
## The clustering contains 7 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster, clusters_xi
```

```
plot(res)
```



```
hullplot(x, res)
```

Convex Cluster Hulls



5 Ejercicios

Los ejercicios se realizarán en base al juego de datos *Hawks* presente en el paquete R *Stat2Data*.

Los estudiantes y el profesorado del Cornell College en Mount Vernon, Iowa, recogieron datos durante muchos años en el mirador de halcones del lago MacBride, cerca de Iowa City, en el estado de Iowa. El conjunto de datos que analizamos aquí es un subconjunto del conjunto de datos original, utilizando sólo aquellas especies para las que había más de 10 observaciones. Los datos se recogieron en muestras aleatorias de tres especies diferentes de halcones: Colirrojo, Gavilán y Halcón de Cooper.

Hemos seleccionado este juego de datos por su parecido con el juego de datos *penguins* y por su potencial a la hora de aplicarle algoritmos de minería de datos no supervisados. Las variables numéricas en las que os basaréis son: *Wing*, *Weight*, *Culmen*, *Hallux*

```
if (!require('Stat2Data')) install.packages('Stat2Data')
library(Stat2Data)
data("Hawks")
summary(Hawks)
```

```

##      Month        Day          Year       CaptureTime   ReleaseTime     Ban
dNumber Species  Age    Sex        Wing      Weight      Culmen      Ha
llux
##  Min.   : 8.000   Min.   :1.00   Min.   :1992   11:35   : 14   :842
: 2   CH: 70   A:224   :576   Min.   : 37.2   Min.   : 56.0   Min.   : 8.6   Min.   :
9.50
##  1st Qu.: 9.000   1st Qu.: 9.00   1st Qu.:1995   13:30   : 14   11:00   : 2   1142-092
40: 1   RT:577   I:684   F:174   1st Qu.:202.0   1st Qu.: 185.0   1st Qu.:12.8   1st Q
u.: 15.10
##  Median :10.000   Median :16.00   Median :1999   11:45   : 13   11:35   : 2   1142-092
41: 1   SS:261           M:158   Median :370.0   Median : 970.0   Median :25.5   Median
: 29.40
##  Mean   : 9.843   Mean   :15.74   Mean   :1998   12:10   : 13   12:05   : 2   1142-092
42: 1           Mean   :315.6   Mean   : 772.1   Mean   :21.8   Mean
: 26.41
##  3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:2001   14:00   : 13   12:50   : 2   1142-182
29: 1           3rd Qu.:390.0   3rd Qu.:1120.0   3rd Qu.:27.3   3rd Q
u.: 31.40
##  Max.   :11.000   Max.   :31.00   Max.   :2003   13:05   : 12   13:32   : 2   1142-192
09: 1           Max.   :480.0   Max.   :2030.0   Max.   :39.2   Max.
:341.40
##                                     (Other):829   (Other): 56   (Other)
:901           NA's   :1           NA's   :10   NA's   :7   NA's   :
6
##      Tail      StandardTail      Tarsus      WingPitFat      KeelFat
Crop
##  Min.   :119.0   Min.   :115.0   Min.   :24.70   Min.   :0.0000   Min.   :0.000   Mi
n.   :0.0000
##  1st Qu.:160.0   1st Qu.:162.0   1st Qu.:55.60   1st Qu.:0.0000   1st Qu.:2.000   1st
Qu.:0.0000
##  Median :214.0   Median :215.0   Median :79.30   Median :1.0000   Median :2.000   Med
ian :0.0000
##  Mean   :198.8   Mean   :199.2   Mean   :71.95   Mean   :0.7922   Mean   :2.184   Mea
n   :0.2345
##  3rd Qu.:225.0   3rd Qu.:226.0   3rd Qu.:87.00   3rd Qu.:1.0000   3rd Qu.:3.000   3rd
Qu.:0.2500
##  Max.   :288.0   Max.   :335.0   Max.   :94.00   Max.   :3.0000   Max.   :4.000   Ma
x.   :5.0000
##           NA's   :337   NA's   :833   NA's   :831   NA's   :341   N
A's   :343

```

5.1 Ejercicio 1

Presenta el juego de datos, nombre y significado de cada columna, así como las distribuciones de sus valores. Adicionalmente realiza un estudio similar al de los ejemplos 1.1 y 1.2

5.1.1 Respuesta 1

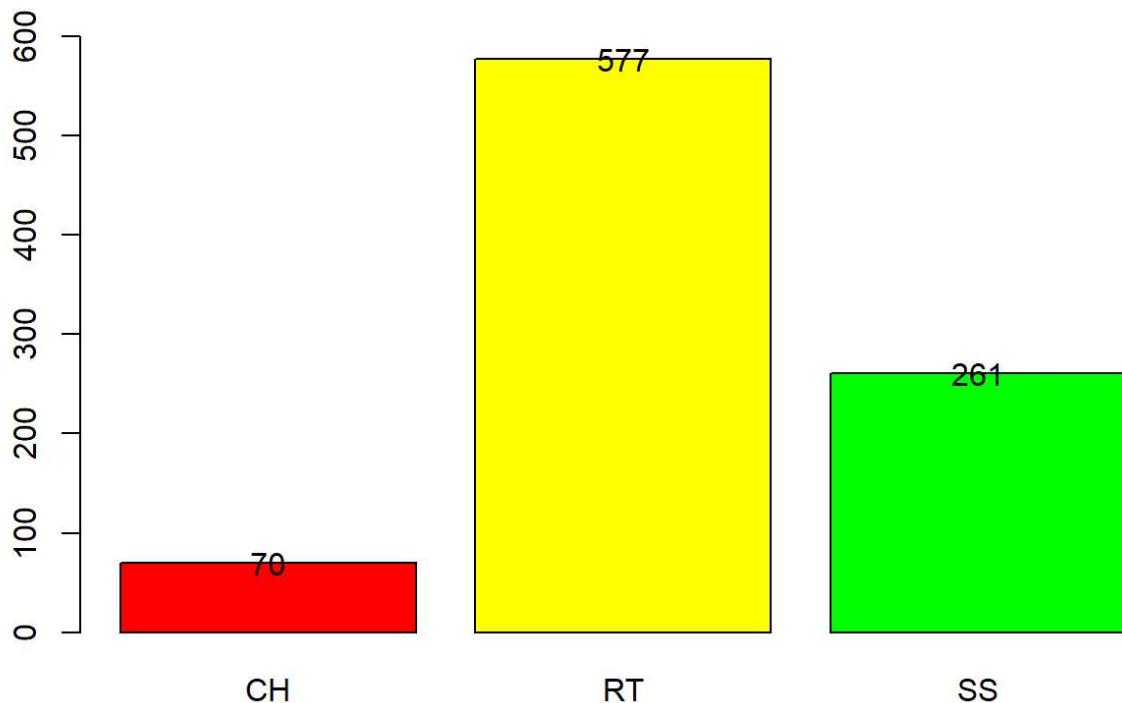
Los campos de la BBDD son:

-Month -> desde el numero 8 = Septiembre a 12 = diciembre.

- Day -> Fecha en el mes.
- Year -> Año que va desde el 1992 hasta el 2003.
- CaptureTime -> Hora de captura (HH: MM).
- ReleaseTime -> Hora de lanzamiento (HH: MM).
- BandNumber -> Código de banda de identificación.
- Species -> Tipo de especie: CH= Cooper's, RT= Red-tailed, SS= Sharp-Shinned.
- Age -> A= Adulto o I= Imaduro.
- Sex -> F= Mujer o M= Hombre.
- Wing -> Longitud (en mm) de la pluma del ala primaria desde la punta hasta la muñeca a la que se adhiere.
- Weight -> Peso corporal (en g).
- Culmen -> Longitud (en mm) del pico superior desde la punta hasta donde choca con la parte carnosa del ave.
- Hallux -> Longitud (en mm) de la garra asesina.
- Tail -> Medida (en mm) relacionada con la longitud de la cola (inventada en el MacBride Raptor Center).
- StandardTail -> Medida estándar de la longitud de la cola (en mm).
- Tarsus -> Longitud del hueso básico del pie (en mm).
- WingPitFat -> Cantidad de grasa en el hoyo de las alas.
- KeelFat -> Cantidad de grasa en el esternón (medida por tacto).
- Crop -> Cantidad de material en el cultivo, codificado de 1= lleno a 0= vacío.

Lo primero que debemos hacer es analizar los datos. Lo primero es saber cuantos datos de cada especie tenemos.

```
species <- table(Hawks['Species'])
barp <- barplot(species, col = rainbow(6), ylim = c(0, 600))
text(barp, species, labels = species)
```



Nos quedamos únicamente con los campos columnas que nos interesan

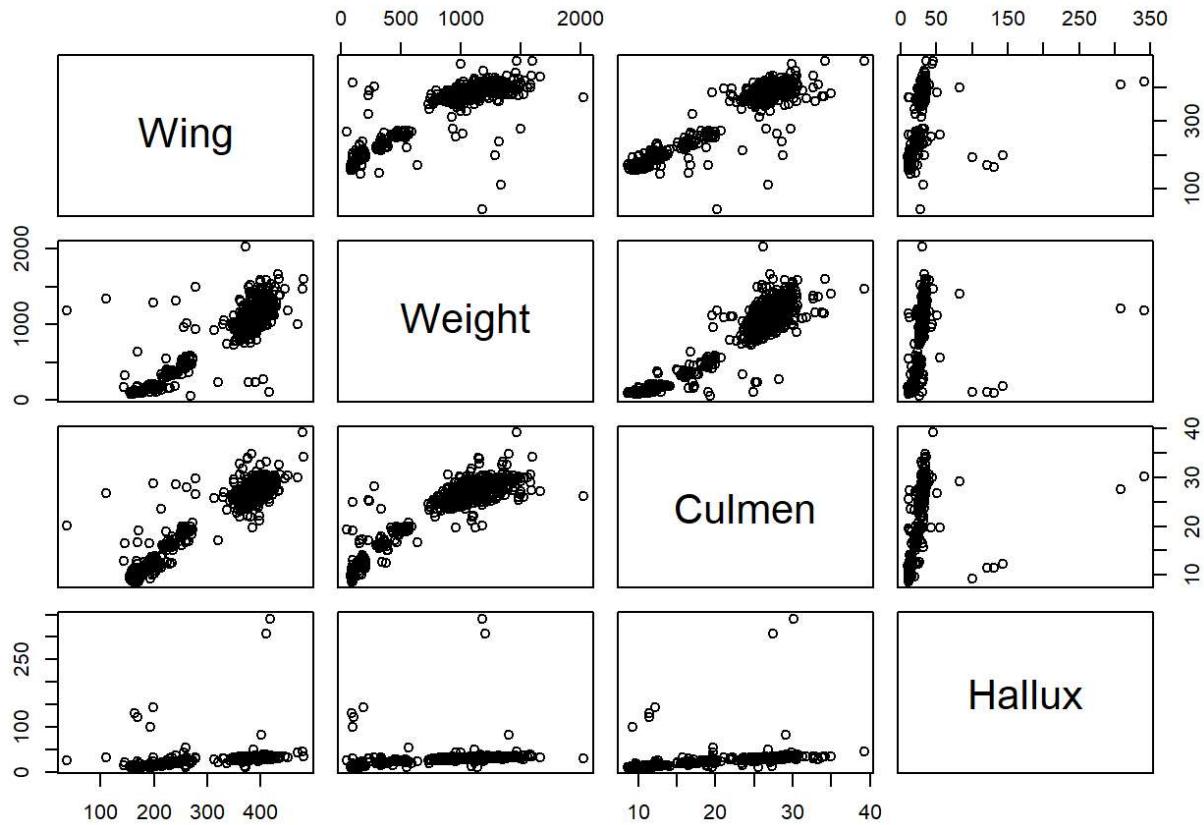
```
Hawks_data <- na.omit(Hawks[c('Wing', 'Weight', 'Culmen', 'Hallux')])
```

Los campos que nos interesan son:

-Wing -> Longitud (en mm) de la pluma del ala primaria desde la punta hasta la muñeca a la que se adhiere. -
Weight -> Peso corporal (en g). **-Culmen** -> Longitud (en mm) del pico superior desde la punta hasta donde choca con la parte carnosa del ave. **-Hallux** -> Longitud (en mm) de la garra asesina.

Vemos la distribución de los datos elegidos:

```
plot(Hawks_data)
```

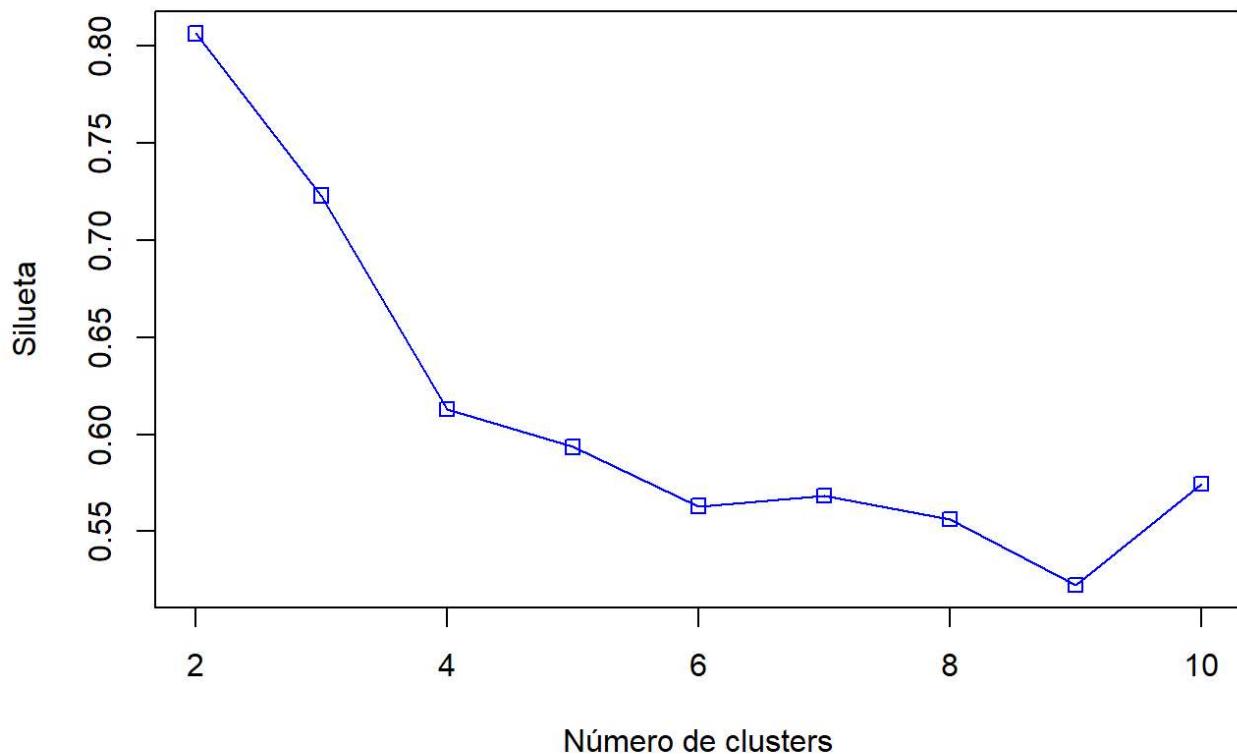


Como inicialmente no conocemos el número óptimo de clústers, probamos con varios valores

```
d <- daisy(Hawks_data)
resultados <- rep(0, 10)
for (i in c(2,3,4,5,6,7,8,9,10))
{
  fit           <- kmeans(Hawks_data, i)
  y_cluster     <- fit$cluster
  sk            <- silhouette(y_cluster, d)
  resultados[i] <- mean(sk[,3])
}
```

Mostramos en un gráfica los valores de las siluetas media de cada prueba para comprobar que número de clústers es el mejor.

```
plot(2:10,resultados[2:10],type="o",col="blue",pch=0,xlab="Número de clusters",ylab="Silueta")
```



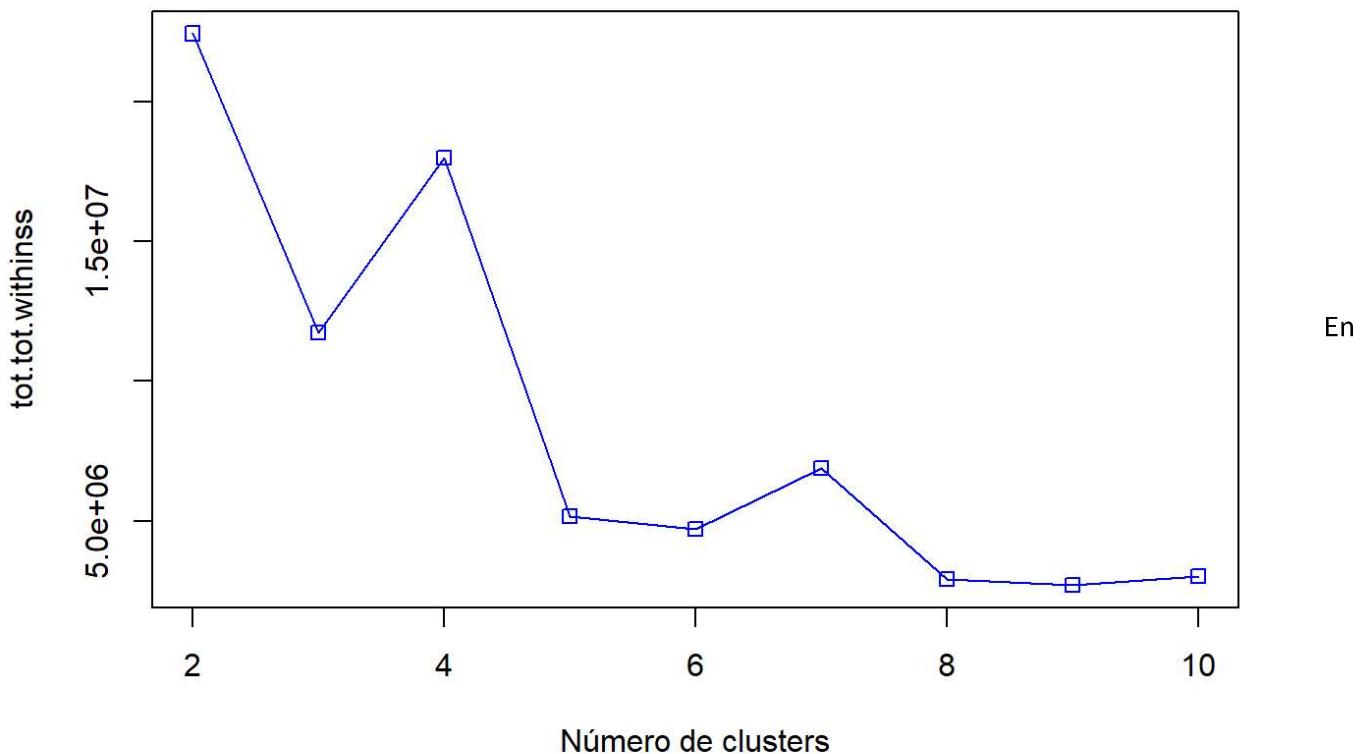
Como se puede comprobar de manera visual, la mejora mas significativa se obtiene para K = 3.

A continuación, se aplicará el método elbow (codo), para evaluar cual es el mejor número de clústers.

```

resultados <- rep(0, 10)
for (i in c(2,3,4,5,6,7,8,9,10))
{
  fit           <- kmeans(Hawks_data, i)
  resultados[i] <- fit$tot.withinss
}
plot(2:10,resultados[2:10],type="o",col="blue",pch=0,xlab="Número de clusters",ylab="tot.tot.withinss")

```



en este caso el número óptimo de clústers son 5 que es cuando la curva comienza a estabilizarse.

Ahora se aplica la función kmeansruns para seleccionar el valor del número de clústers que mejor funcione de acuerdo a dos criterios: la silueta media (“asw”) y Calinski-Harabasz (“ch”).

```
if (!require('fpc')) install.packages('fpc')
library(fpc)
fit_ch <- kmeansruns(Hawks_data, krange = 1:10, criterion = "ch")
fit_asw <- kmeansruns(Hawks_data, krange = 1:10, criterion = "asw")
```

Podemos comprobar el valor con el que se ha obtenido el mejor resultado y también mostrar el resultado obtenido para todos los valores de k usando ambos criterios

```
fit_ch$bestk
```

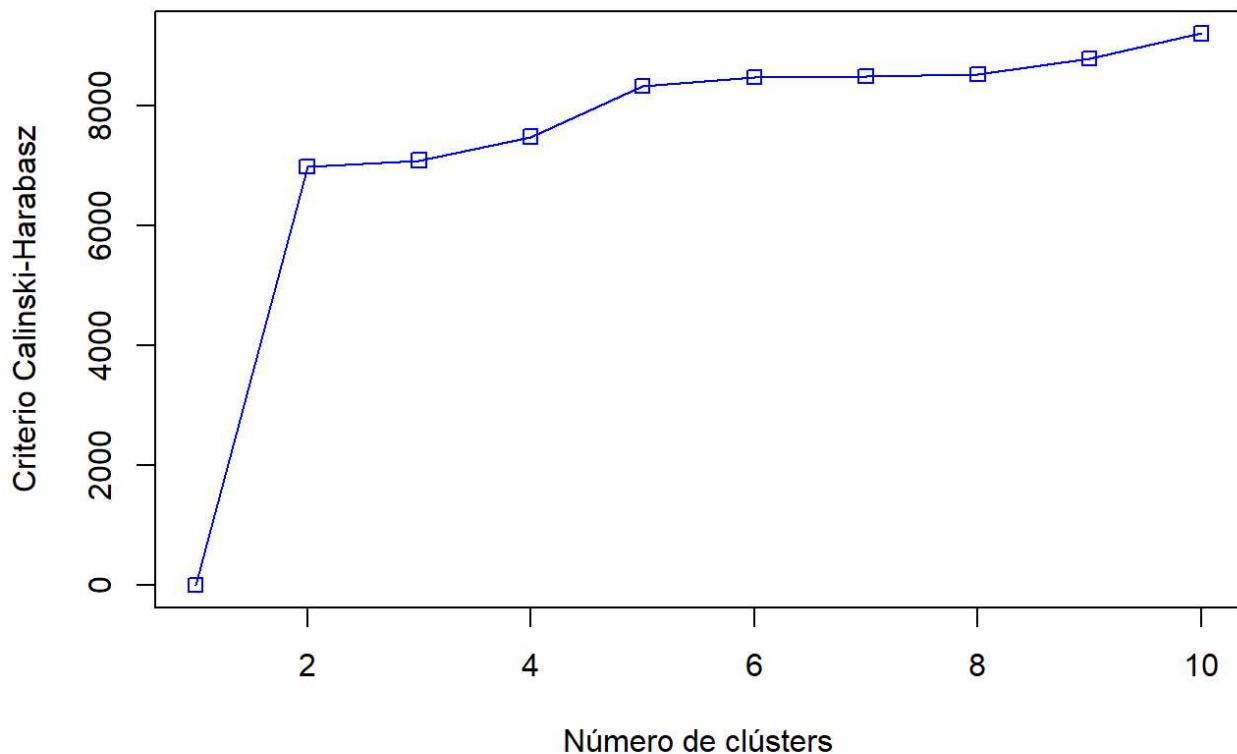
```
## [1] 10
```

```
fit_asw$bestk
```

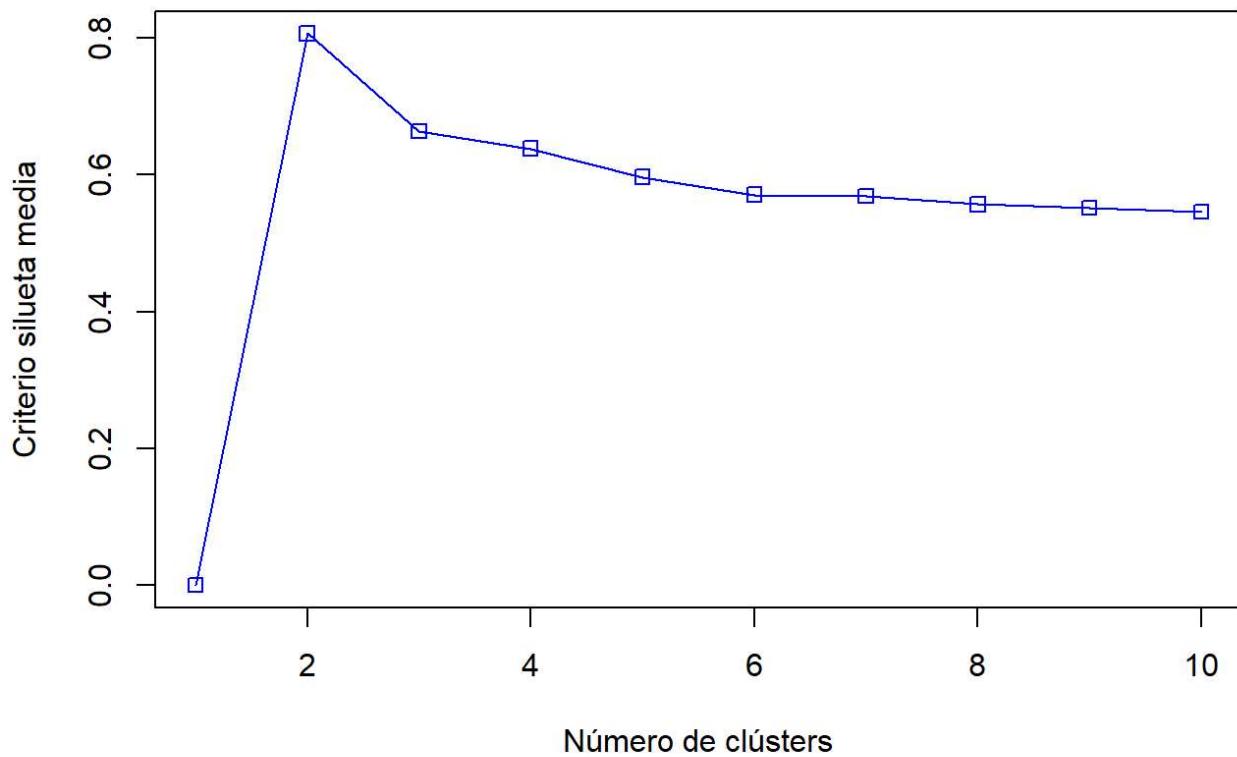
```
## [1] 2
```

Tras obtener los resultados, lo mostramos de una manera visual.

```
plot(1:10, fit_ch$crit, type="o", col="blue", pch=0, xlab="Número de clústers", ylab="Criterio Calinski-Harabasz")
```



```
plot(1:10,fit_asw$crit,type="o",col="blue",pch=0,xlab="Número de clústers",ylab="Criterio silueta media")
```



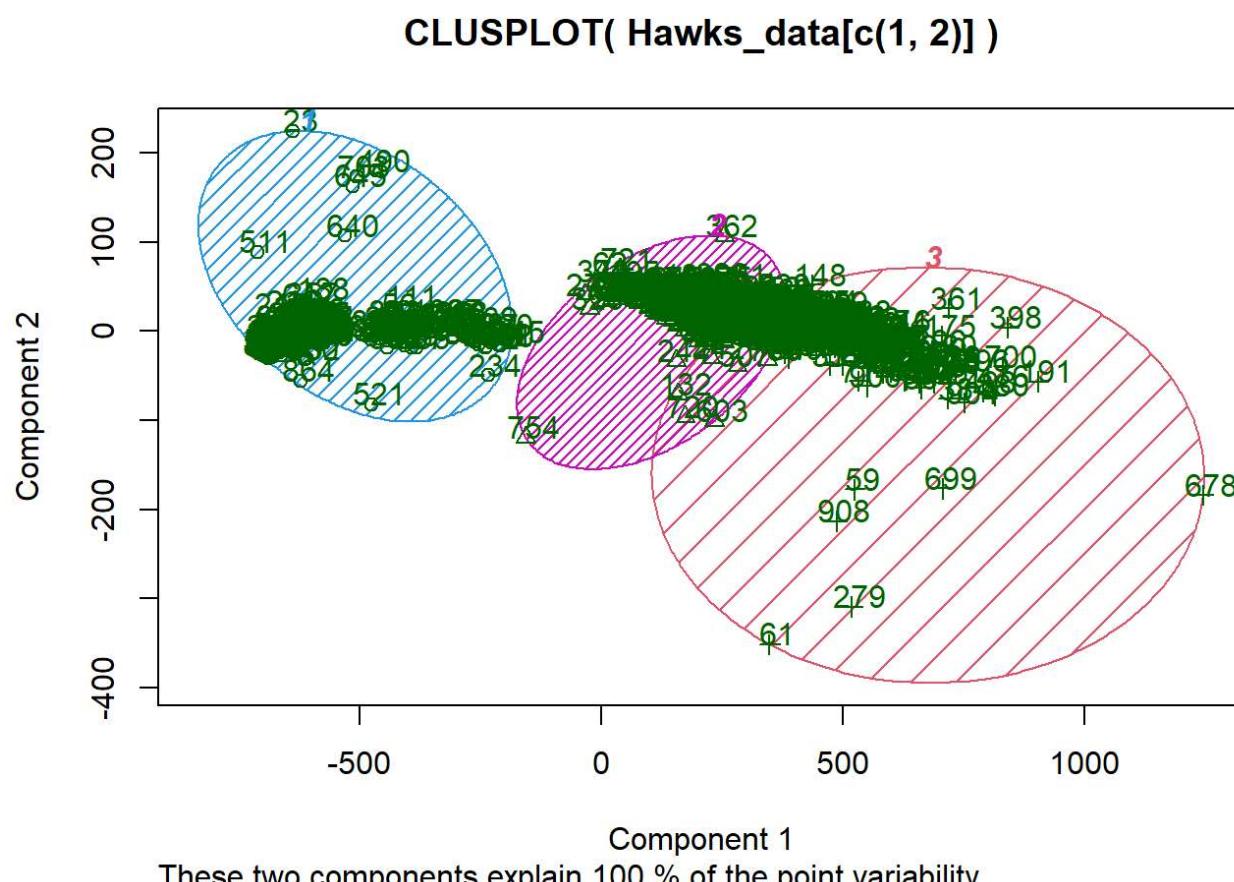
Los resultados son muy parecidos a los que hemos obtenido anteriormente. Con el criterio de la silueta media se obtienen 3 clústers y con el Calinski-Harabasz se obtienen 5.

Como en el caso que estudiamos sabemos que los datos pueden ser agrupados en 3 clases o especies, vamos a ver cómo se ha comportado kmeans en el caso de pedirle 3 clústers. Para eso comparamos visualmente los campos dos a dos, con el valor real que sabemos está almacenado en el campo "species" del dataset original.

```
Hawks3clusters <- kmeans(Hawks_data, 3)
```

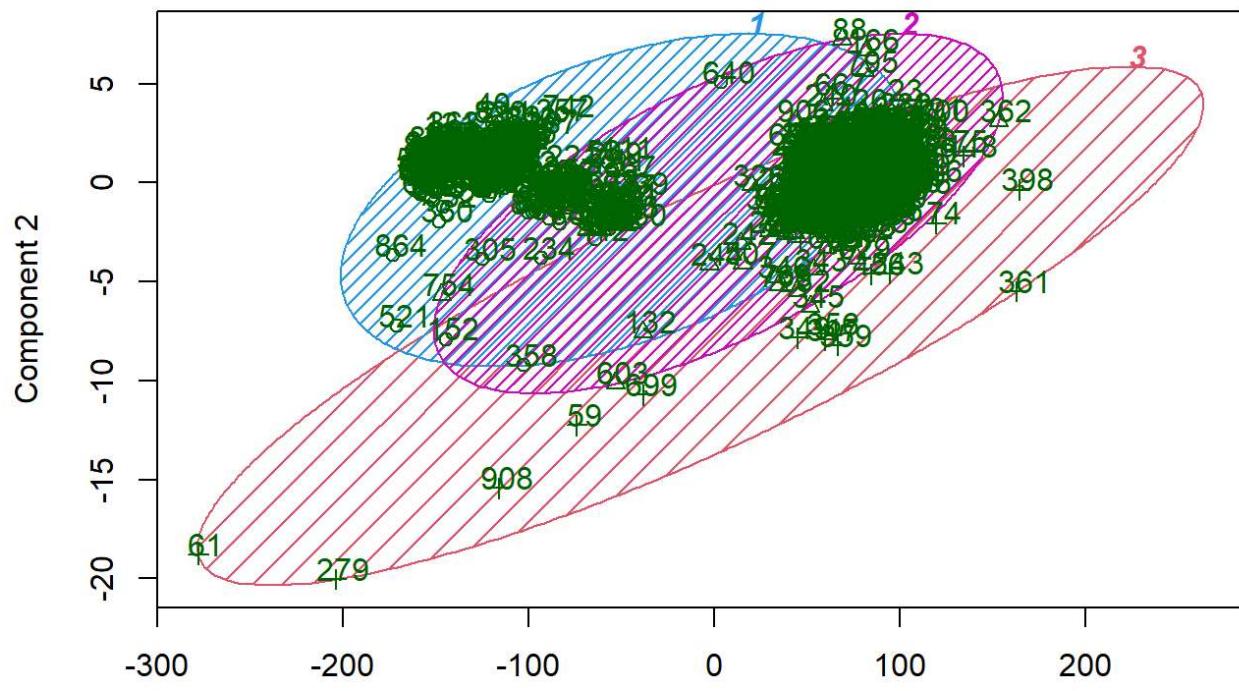
Visualizamos la agrupación con 3 clústers y combinando los diferentes campos

```
clusplot(Hawks_data[c(1,2)], Hawks3clusters$cluster, color=TRUE, shade=TRUE, labels=2, l  
ines=0)
```



```
clusplot(Hawks_data[c(1,3)], Hawks3clusters$cluster, color=TRUE, shade=TRUE, labels=2, l  
ines=0)
```

CLUSPLOT(Hawks_data[c(1, 3)])

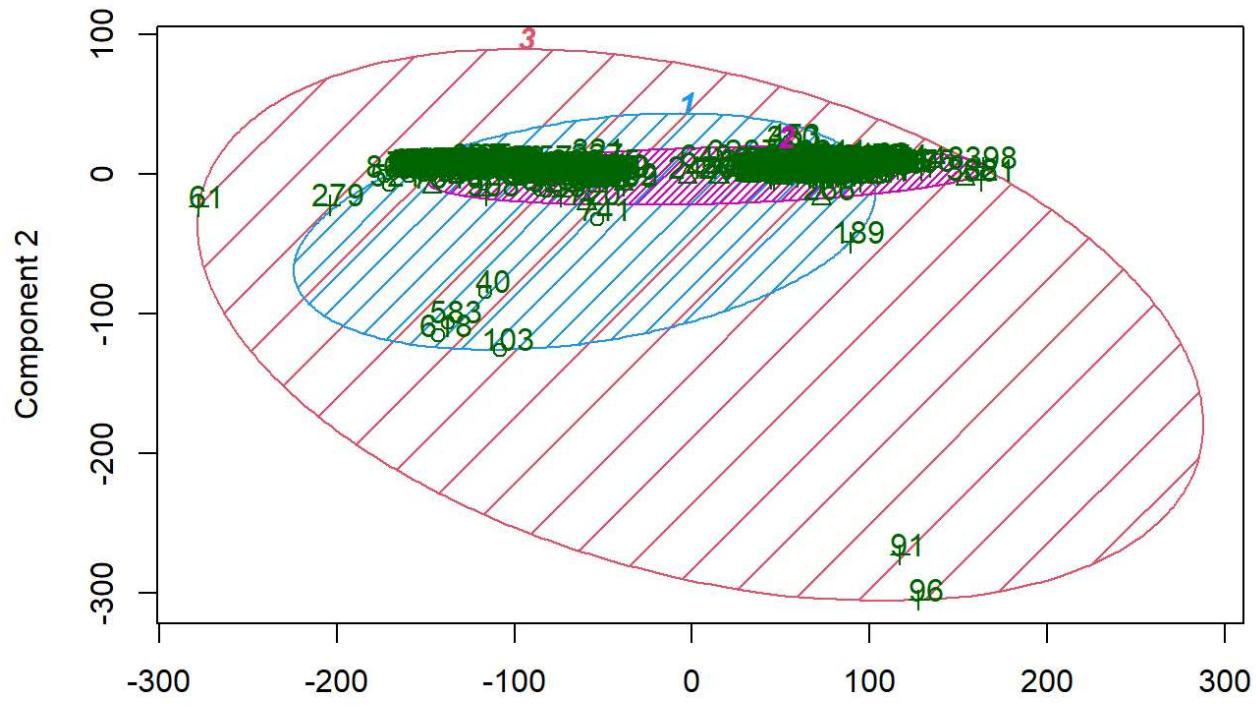


Component 1

These two components explain 100 % of the point variability.

```
clusplot(Hawks_data[c(1,4)], Hawks3clusters$cluster, color=TRUE, shade=TRUE, labels=2, 1  
ines=0)
```

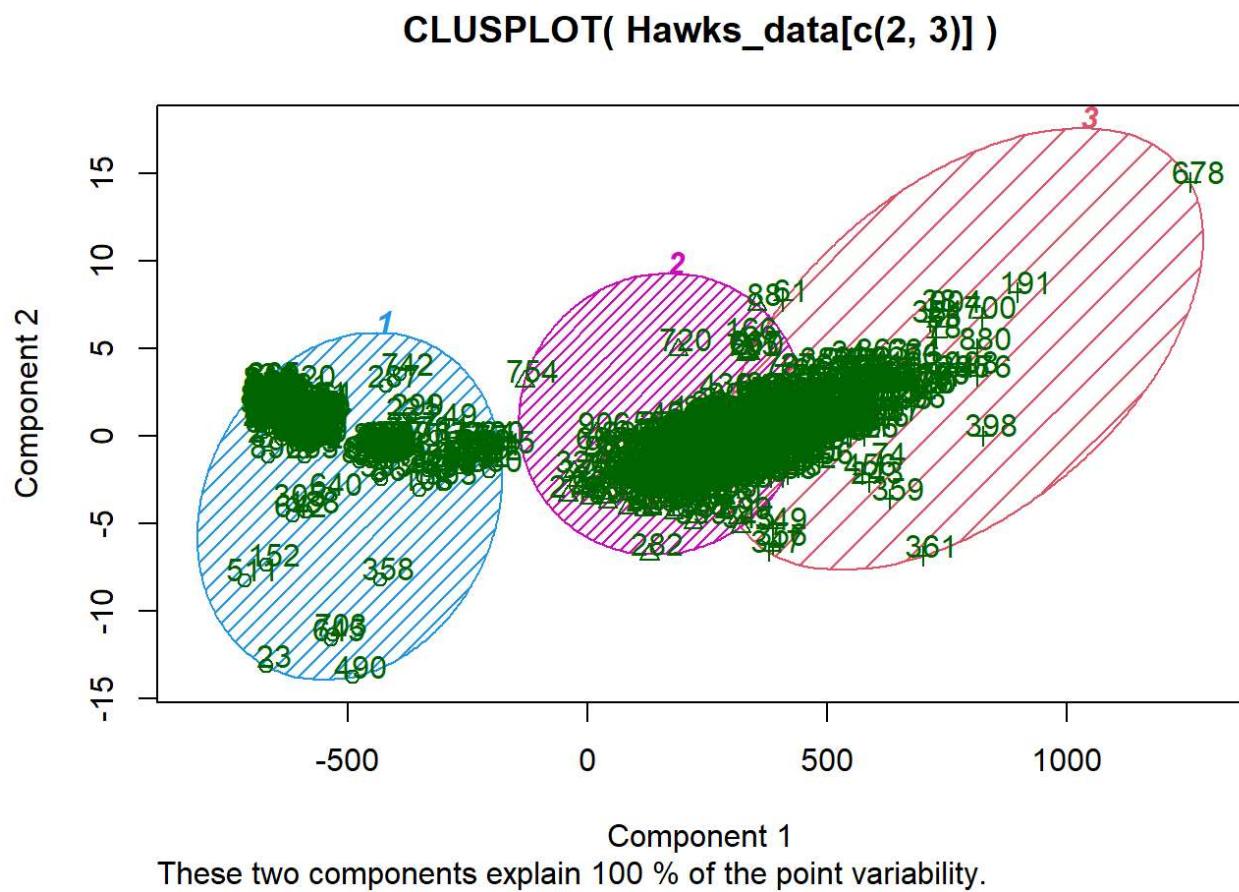
CLUSPLOT(Hawks_data[c(1, 4)])



Component 1

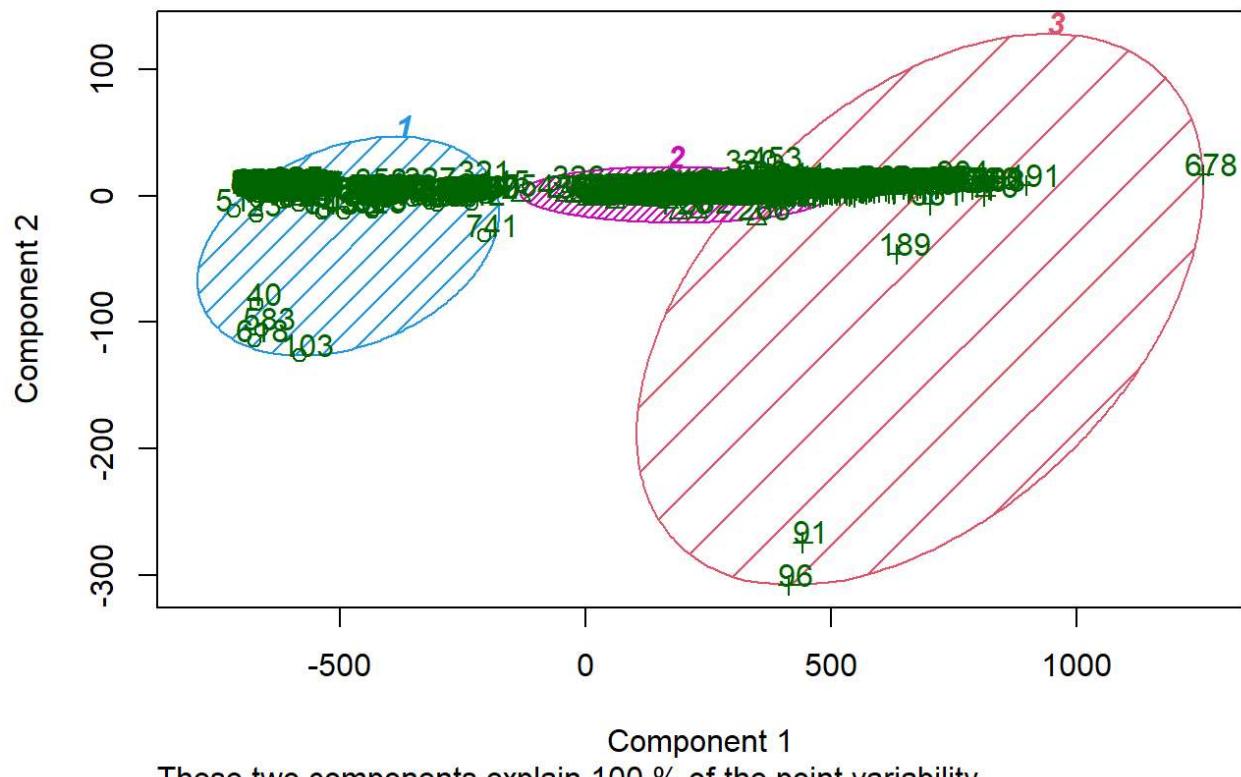
These two components explain 100 % of the point variability.

```
clusplot(Hawks_data[c(2,3)], Hawks3clusters$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```

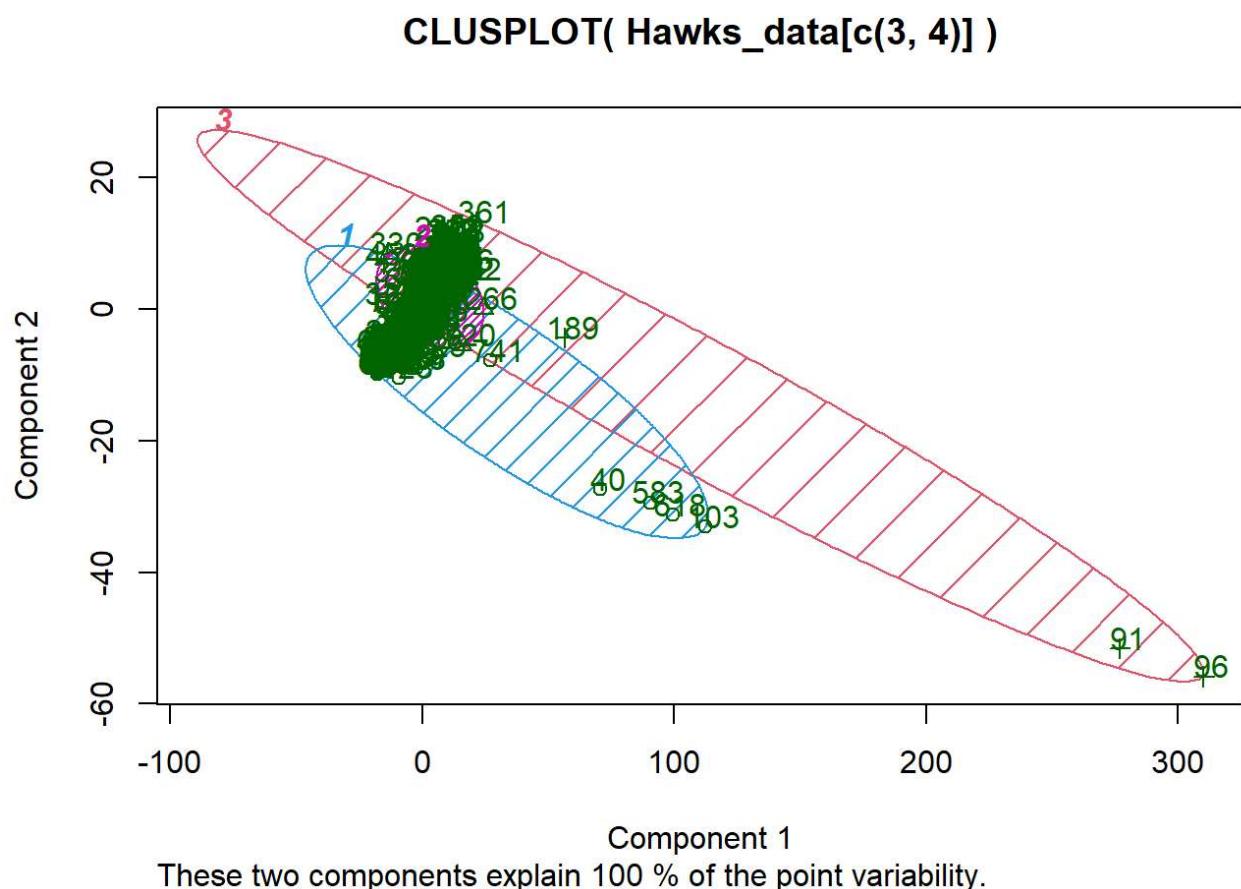


```
clusplot(Hawks_data[c(2,4)], Hawks3clusters$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```

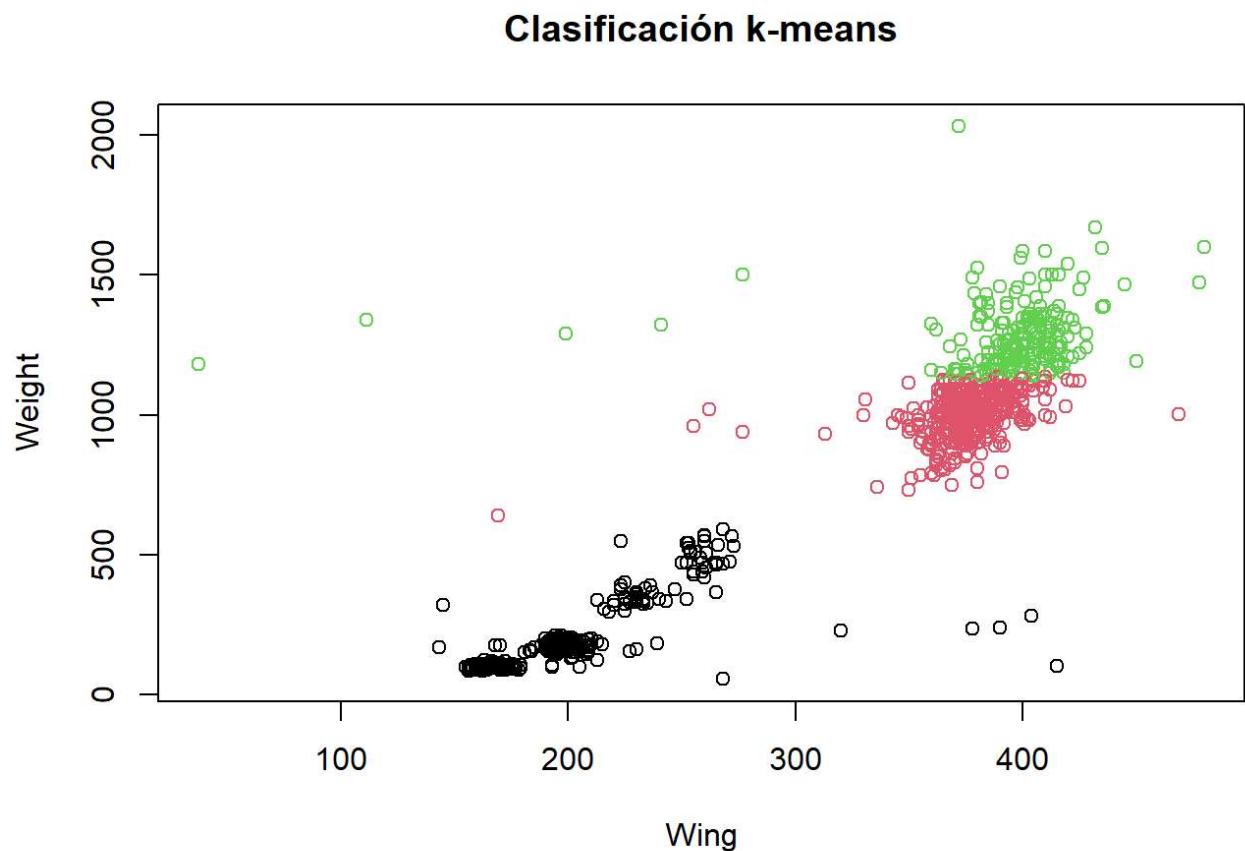
CLUSPLOT(Hawks_data[c(2, 4)])



```
clusplot(Hawks_data[c(3,4)], Hawks3clusters$cluster, color=TRUE, shade=TRUE, labels=2, 1  
ines=0)
```

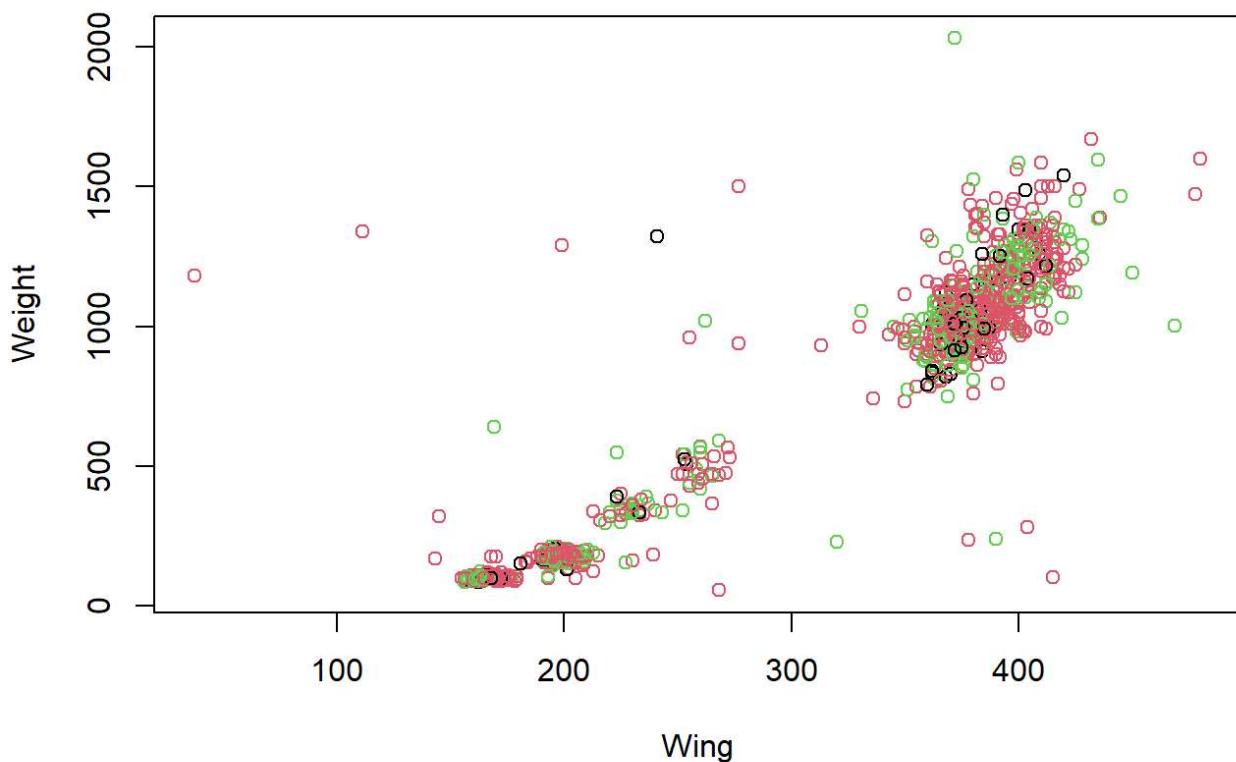


```
plot(Hawks_data[c(1,2)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```



```
plot(Hawks_data[c(1,2)], col=as.factor(Hawks$Species), main="Clasificación real")
```

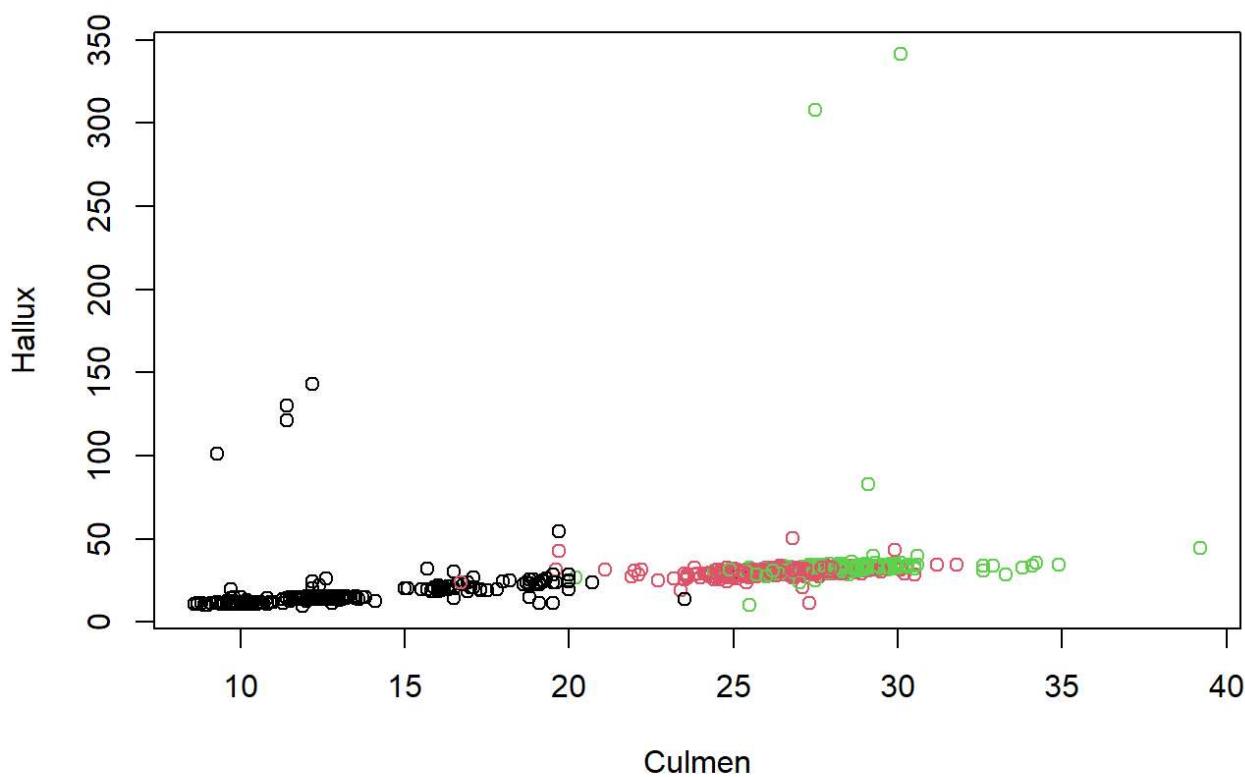
Clasificación real



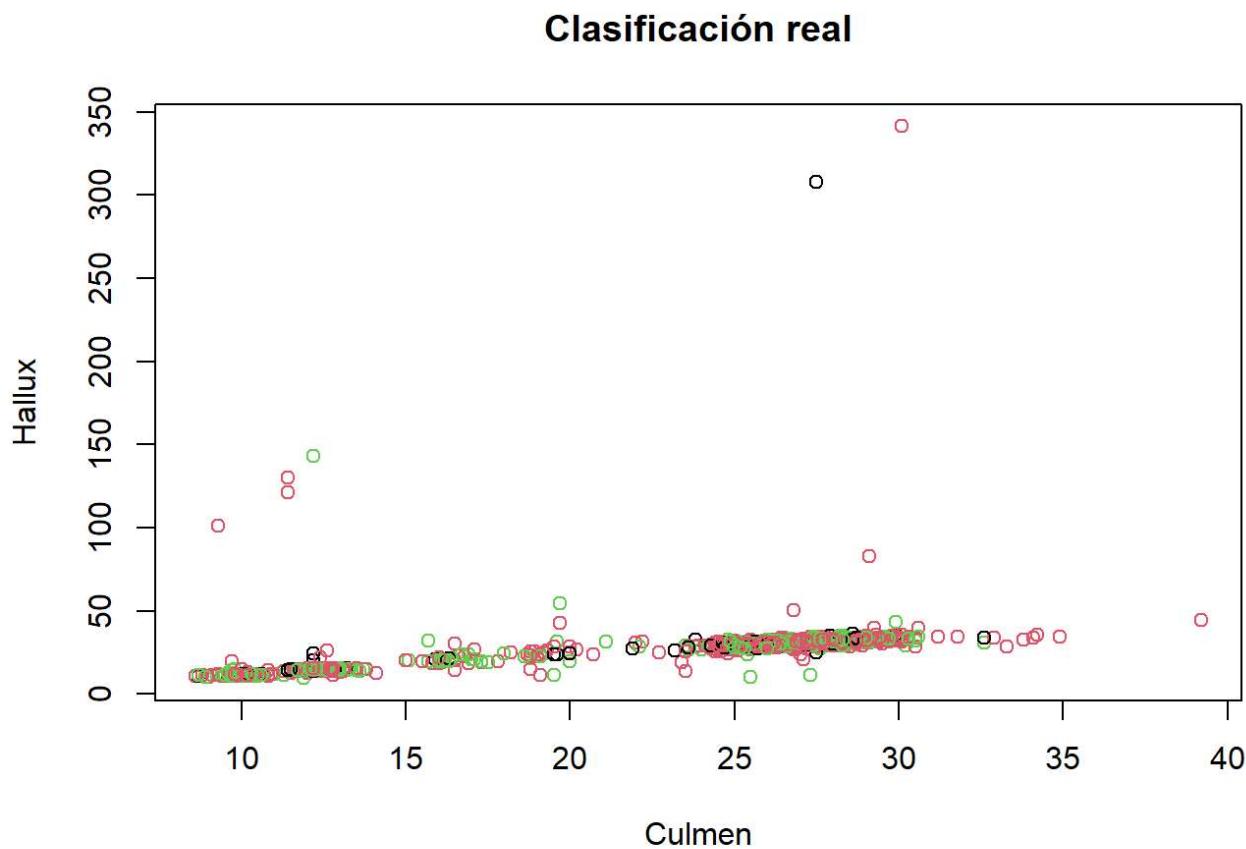
Podemos observar que Weight y Wing no son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

```
plot(Hawks_data[c(3,4)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



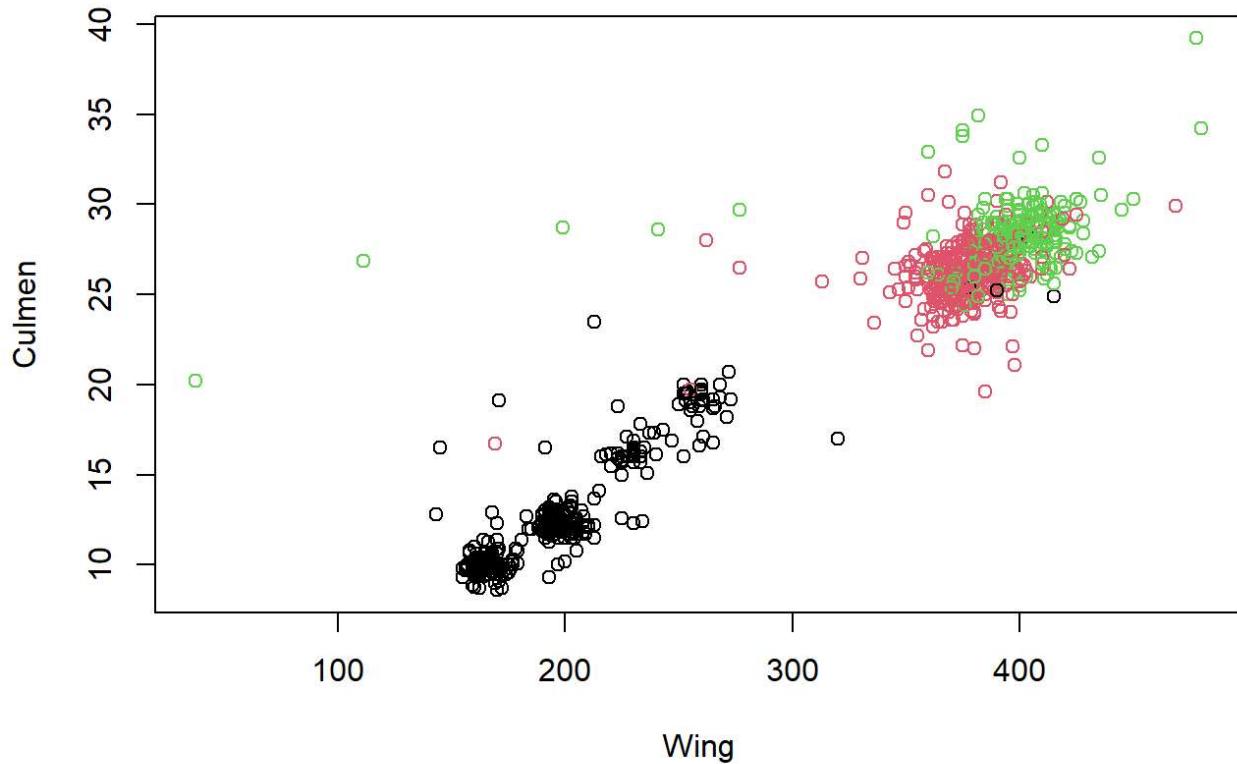
```
plot(Hawks_data[c(3,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```



Podemos observar que Hallux y Culmen tampoco son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

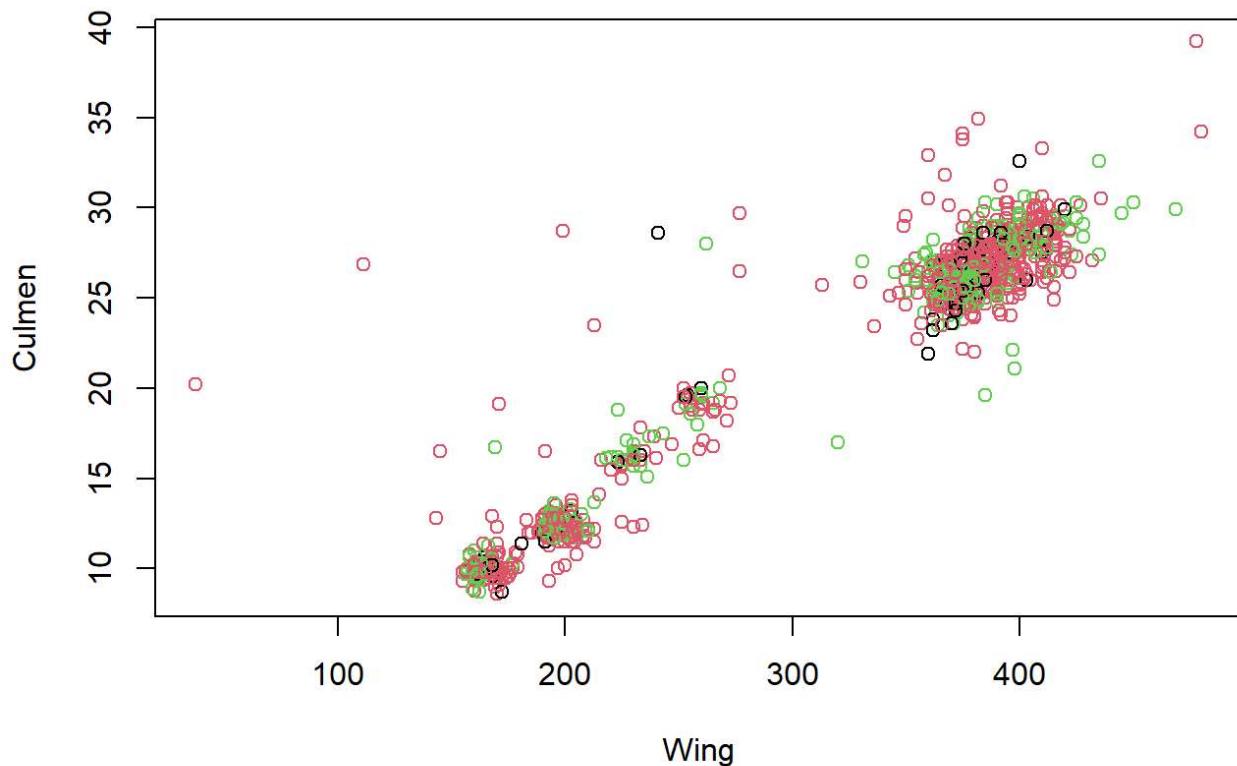
```
plot(Hawks_data[c(1,3)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



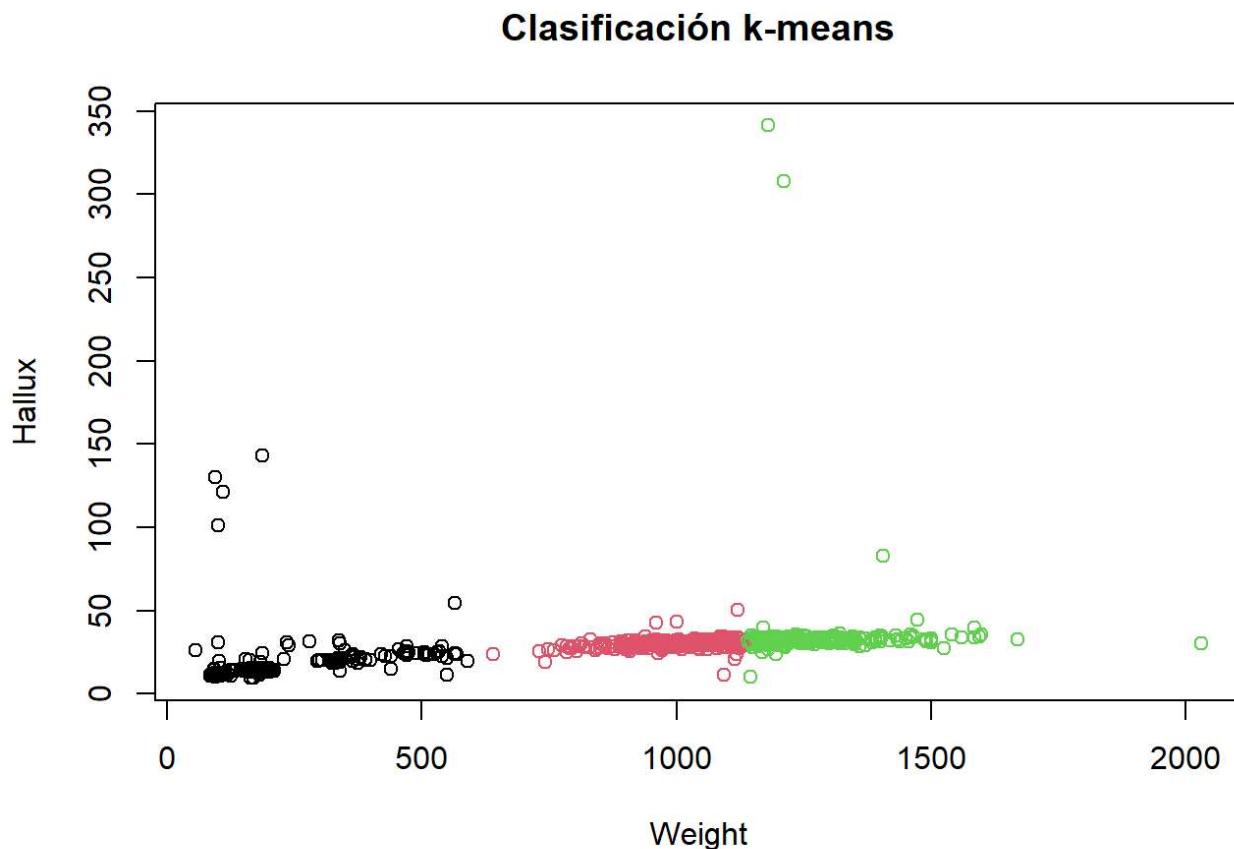
```
plot(Hawks_data[c(1,3)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real



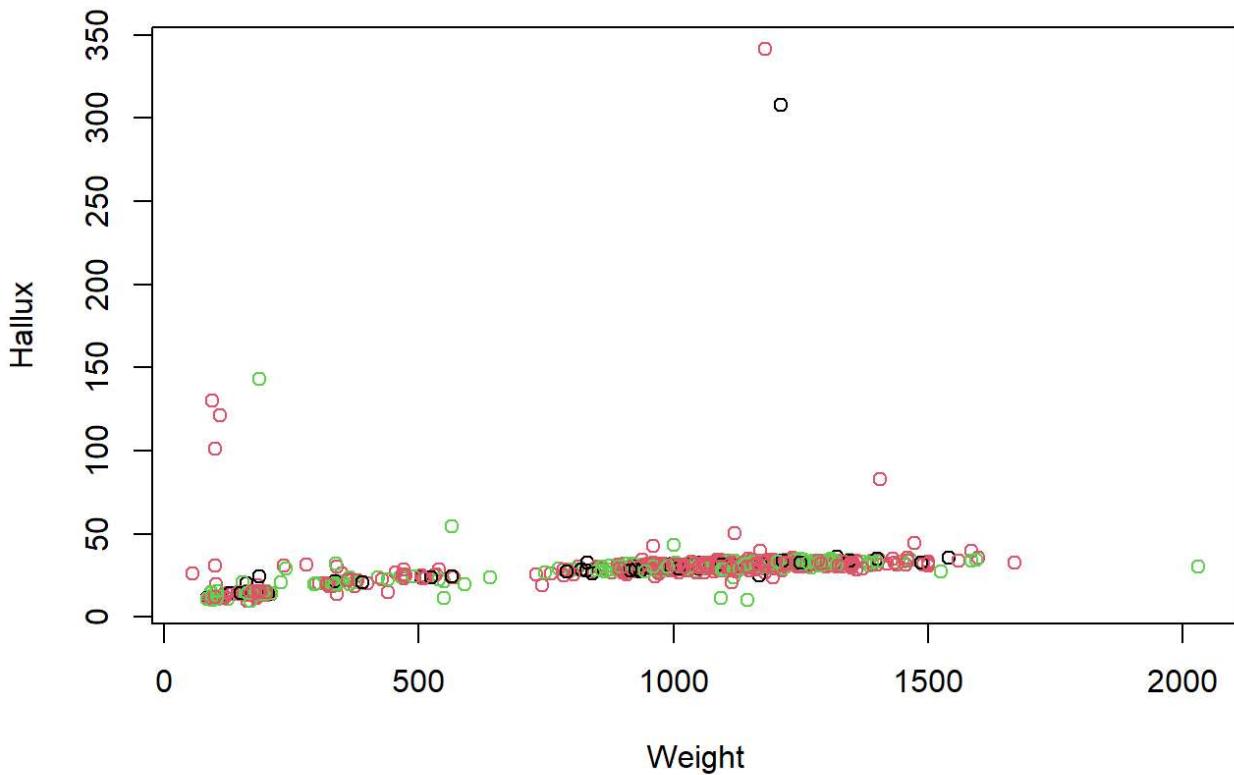
Podemos observar que Wing y Culmen tampoco son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

```
plot(Hawks_data[c(2,4)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```



```
plot(Hawks_data[c(2,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```

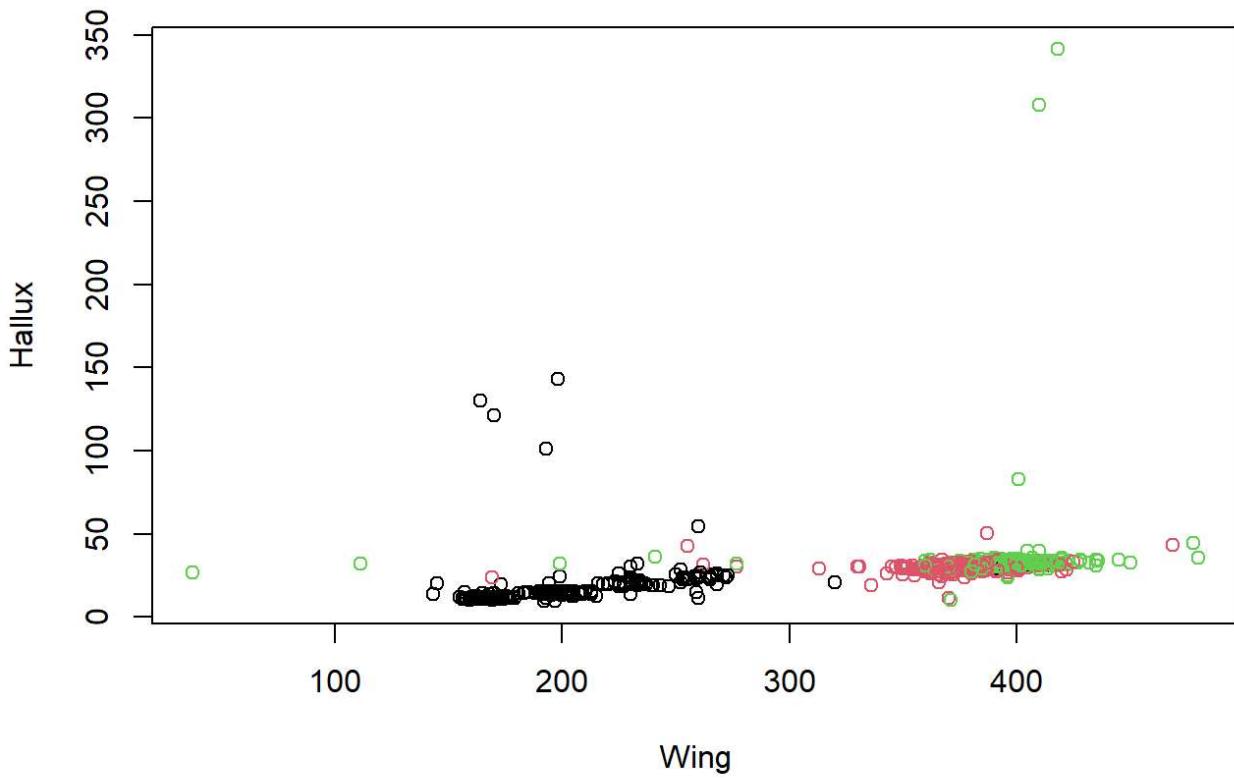
Clasificación real



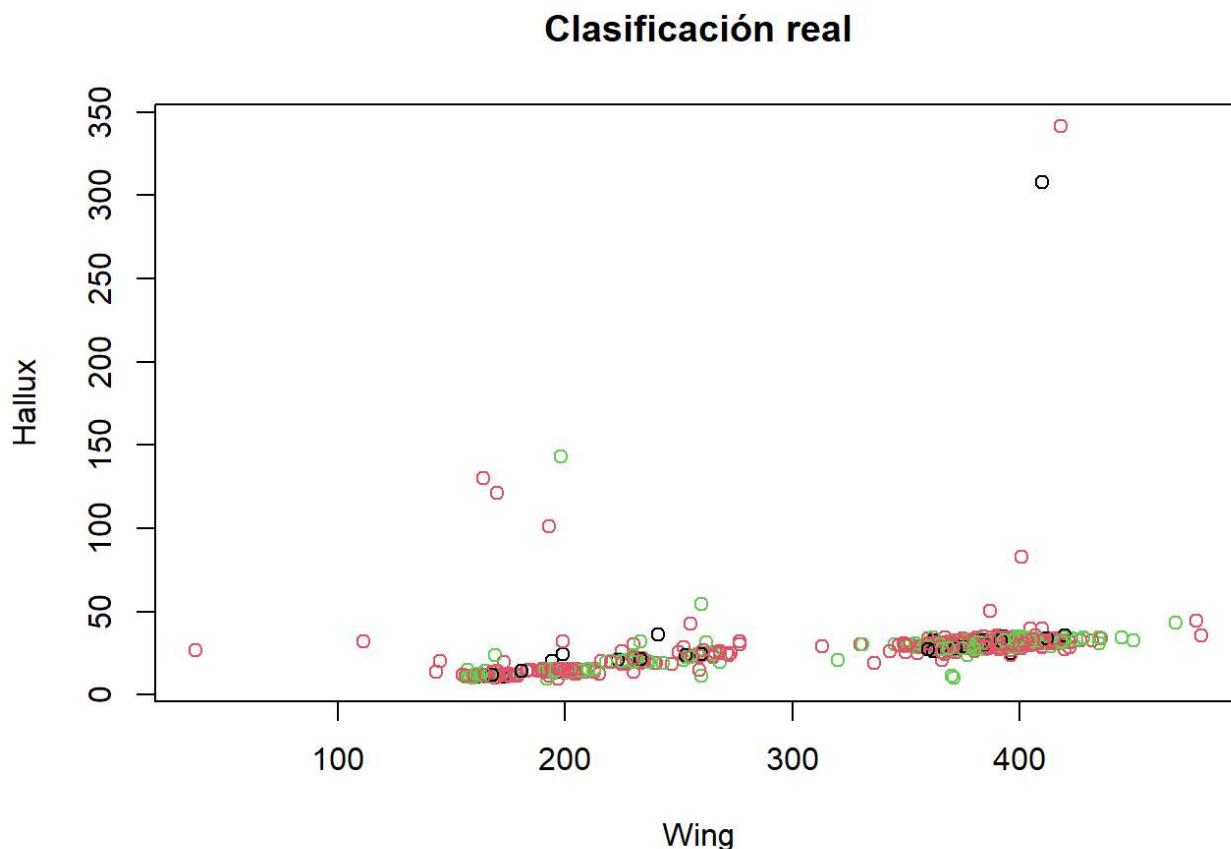
Podemos observar que Hallux y Weight tampoco son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

```
plot(Hawks_data[c(1,4)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



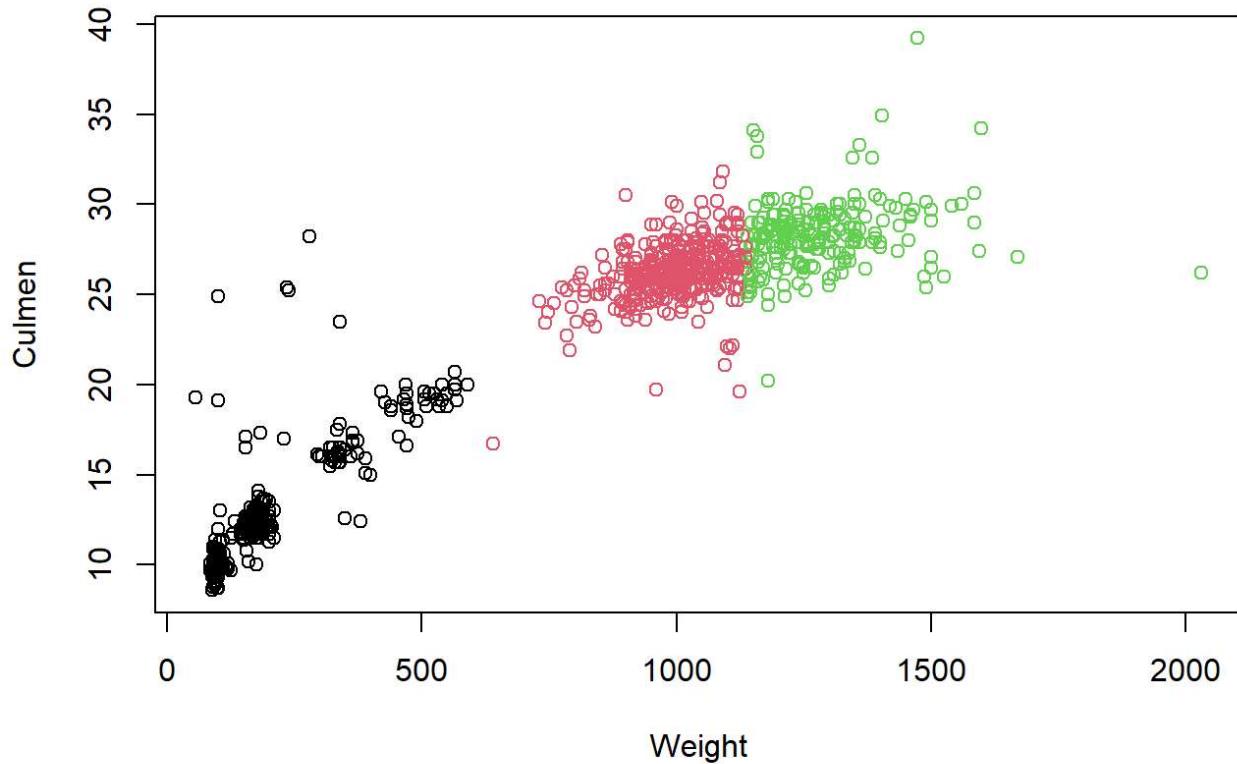
```
plot(Hawks_data[c(1,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```



Podemos observar que Hallux y Wing tampoco son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

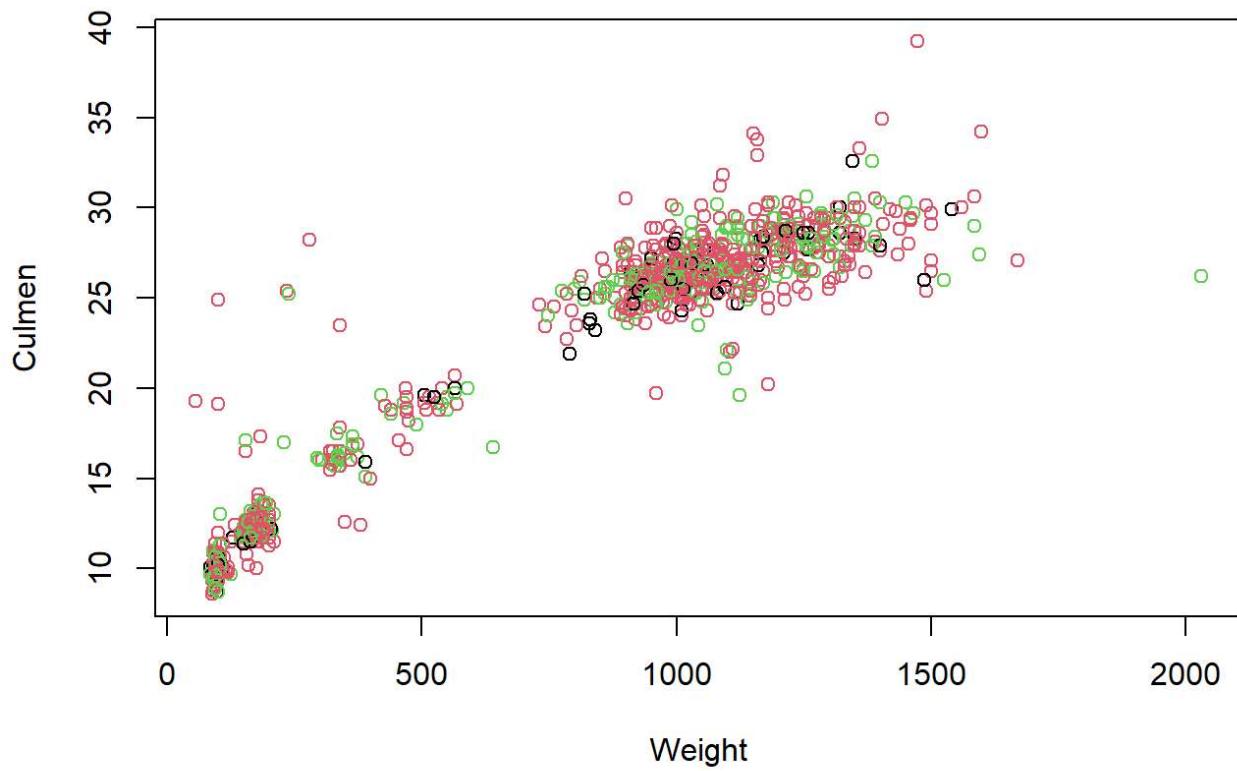
```
plot(Hawks_data[c(2,3)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



```
plot(Hawks_data[c(2,3)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real



Finalmente podemos observar que Culmen y Weight tampoco son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

Ahora vamos a evaluar la calidad del proceso de agregación.

```
d <- daisy(Hawks_data)
sk3 <- silhouette(Hawks3clusters$cluster, d)
```

Calculando la media de la tercera columna podemos obtener una estimación de la calidad del agrupamiento

```
mean(sk3[,3])
```

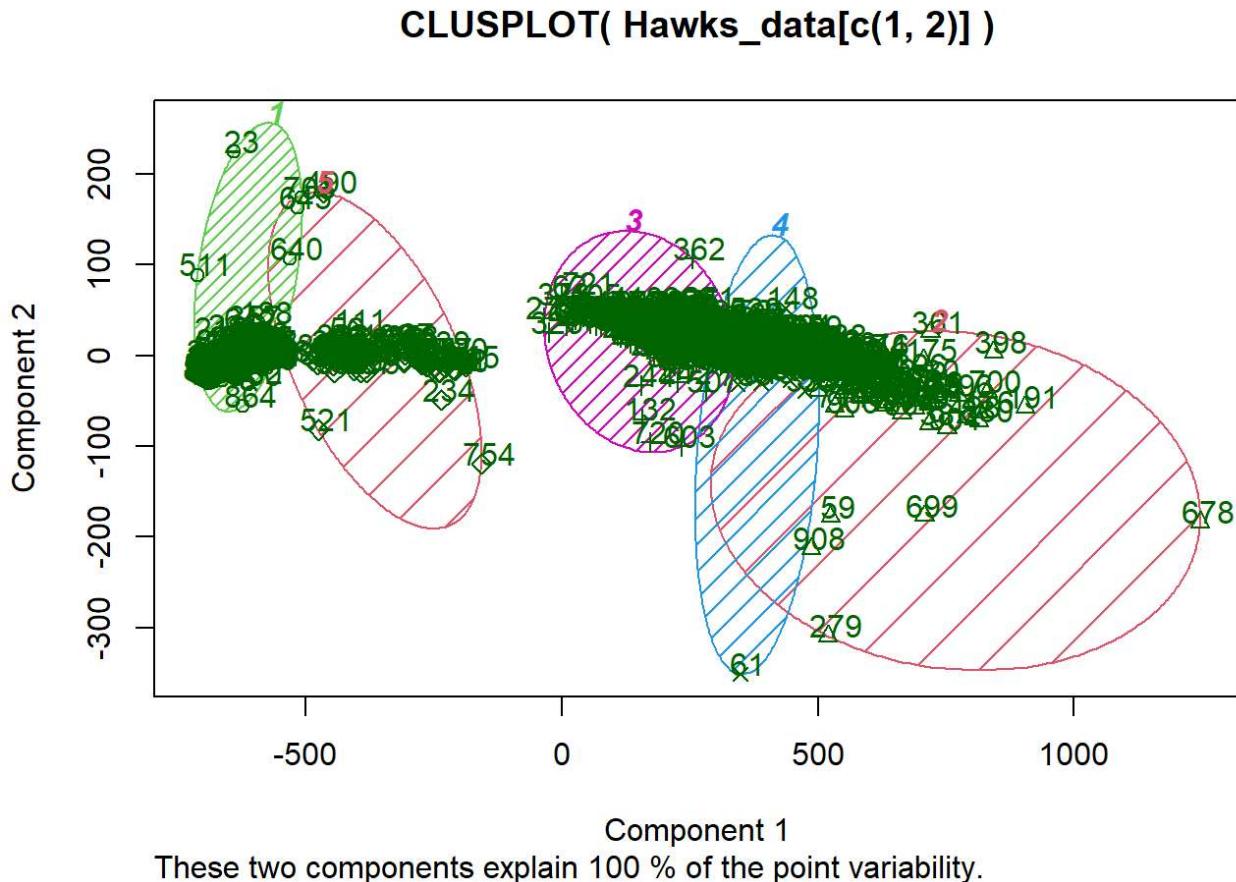
```
## [1] 0.6634045
```

Como en el algoritmo Calinski-Harabasz se obtienen 5 clúster, se va a probar cómo funciona el algoritmo con este número de clústeres.

```
Hawks5clusters <- kmeans(Hawks_data, 5)
```

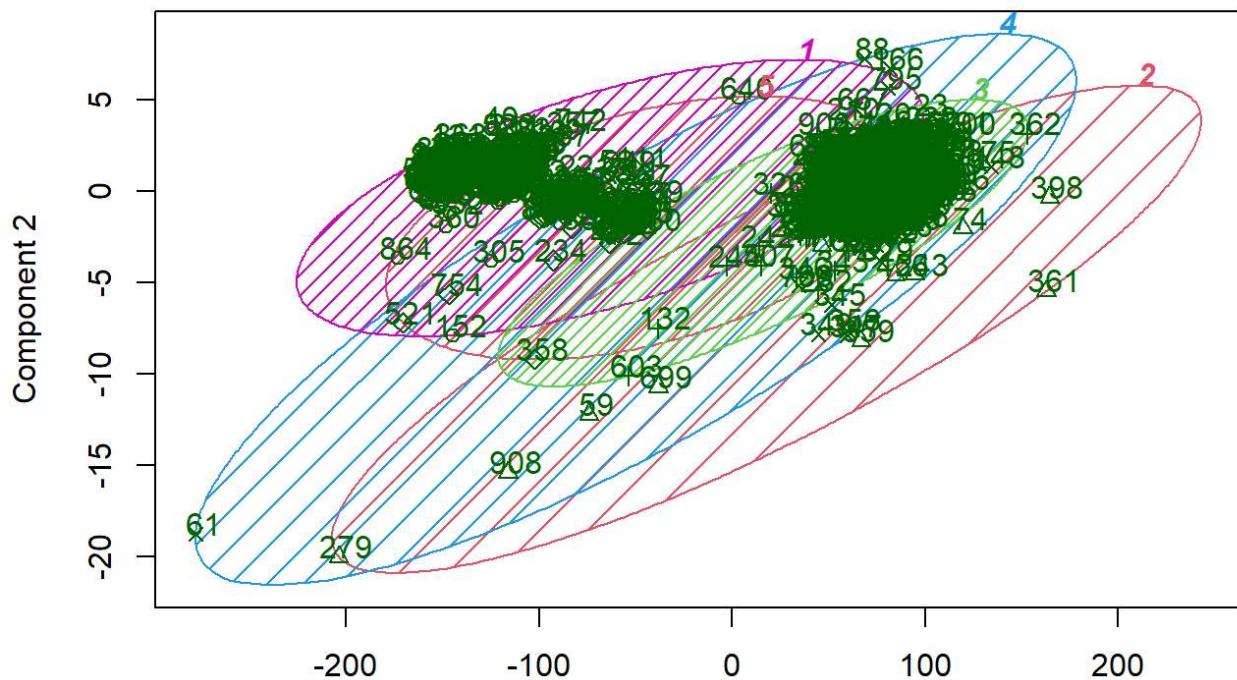
Visualizamos la agrupación con 5 clústeres y combinando los diferentes campos

```
clusplot(Hawks_data[c(1,2)], Hawks5clusters$cluster, color=TRUE, shade=TRUE, labels=2, l  
ines=0)
```



```
clusplot(Hawks_data[c(1,3)], Hawks5clusters$cluster, color=TRUE, shade=TRUE, labels=2, l  
ines=0)
```

CLUSPLOT(Hawks_data[c(1, 3)])

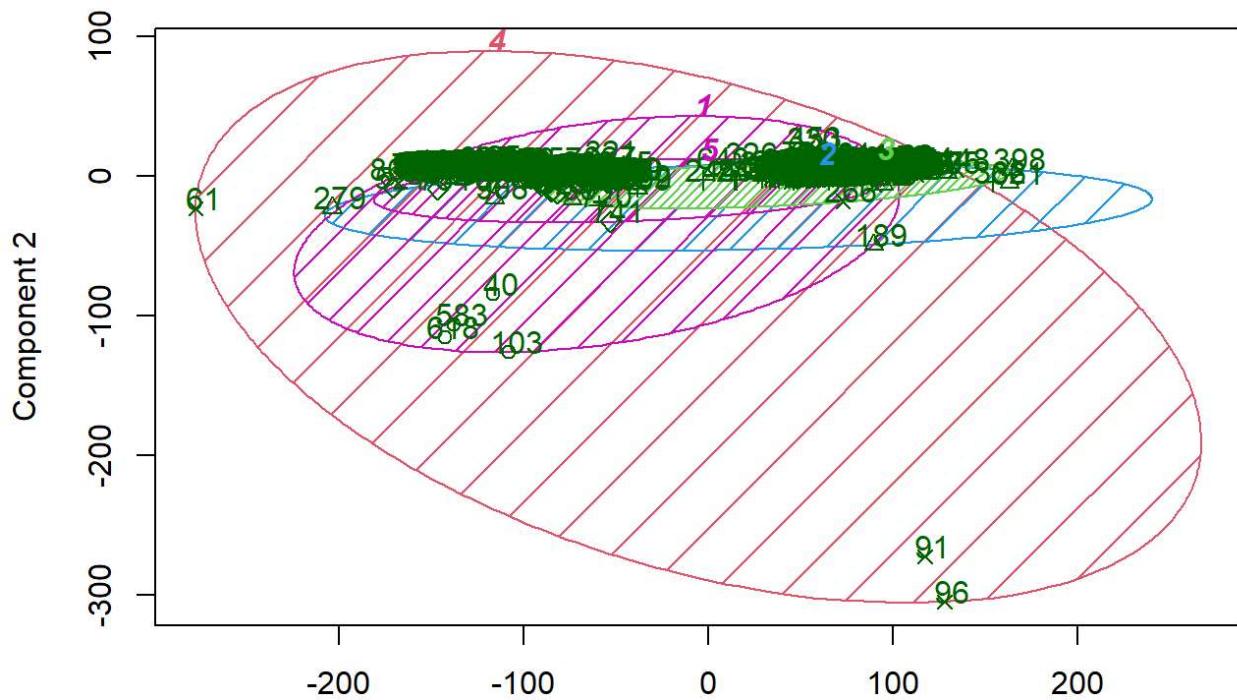


Component 1

These two components explain 100 % of the point variability.

```
clusplot(Hawks_data[c(1,4)], Hawks5clusters$cluster, color=TRUE, shade=TRUE, labels=2, 1  
ines=0)
```

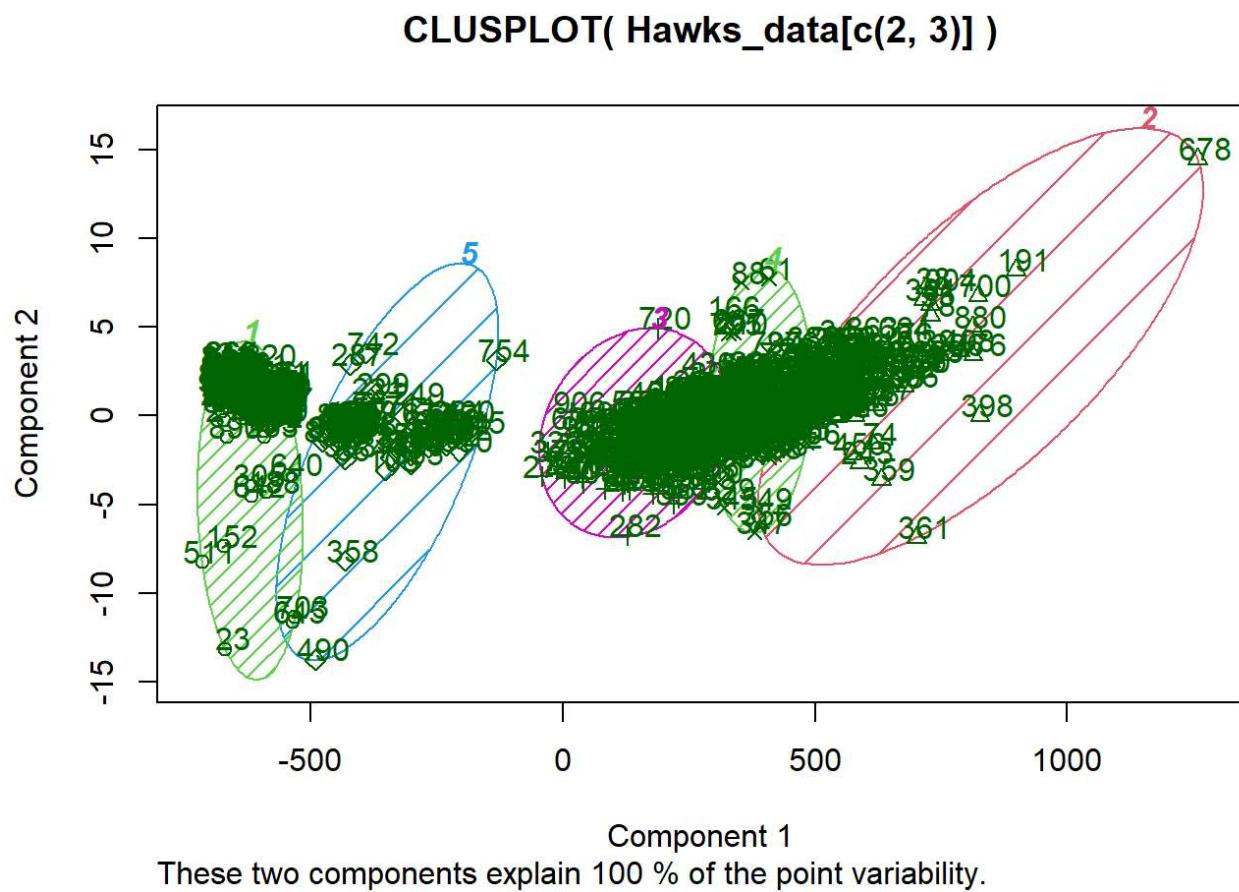
CLUSPLOT(Hawks_data[c(1, 4)])



Component 1

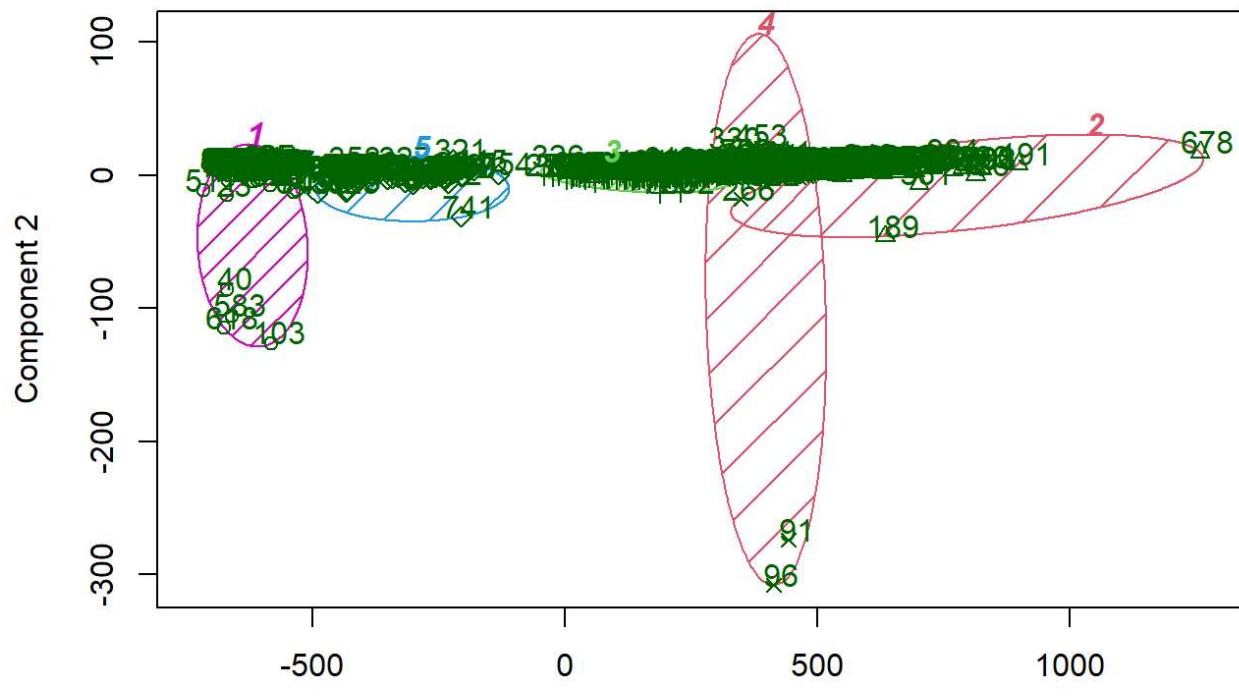
These two components explain 100 % of the point variability.

```
clusplot(Hawks_data[c(2,3)], Hawks5clusters$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



```
clusplot(Hawks_data[c(2,4)], Hawks5clusters$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```

CLUSPLOT(Hawks_data[c(2, 4)])

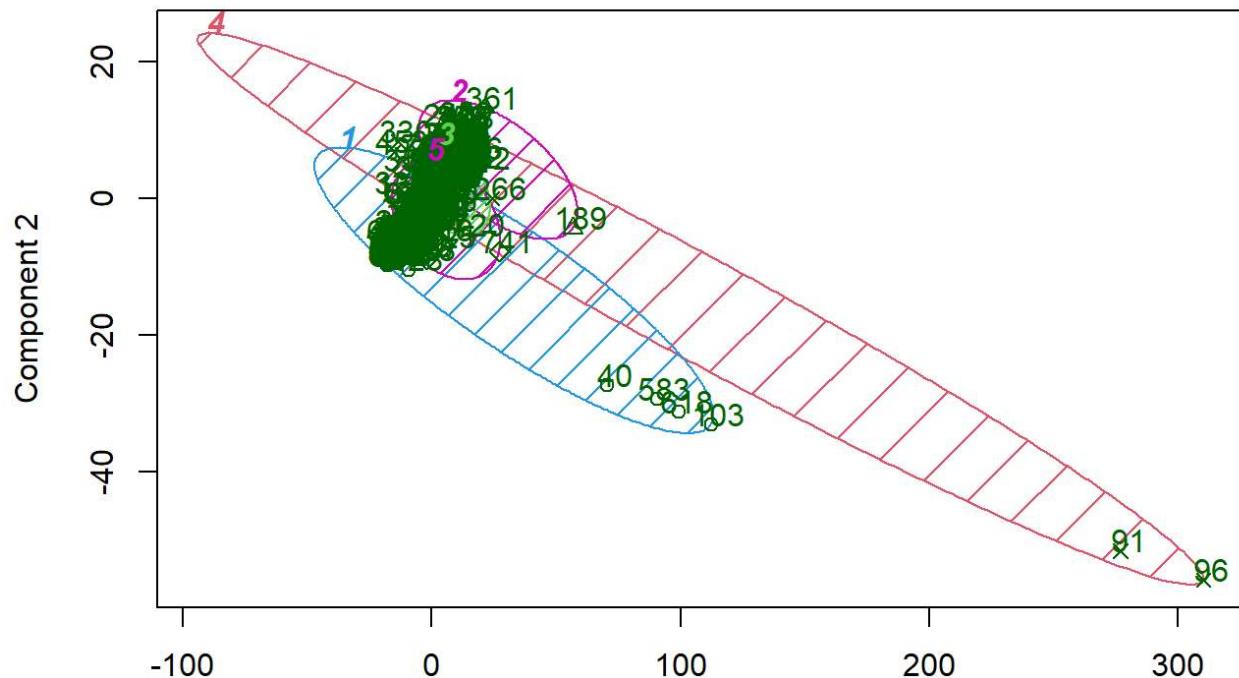


Component 1

These two components explain 100 % of the point variability.

```
clusplot(Hawks_data[c(3,4)], Hawks5clusters$cluster, color=TRUE, shade=TRUE, labels=2, 1  
ines=0)
```

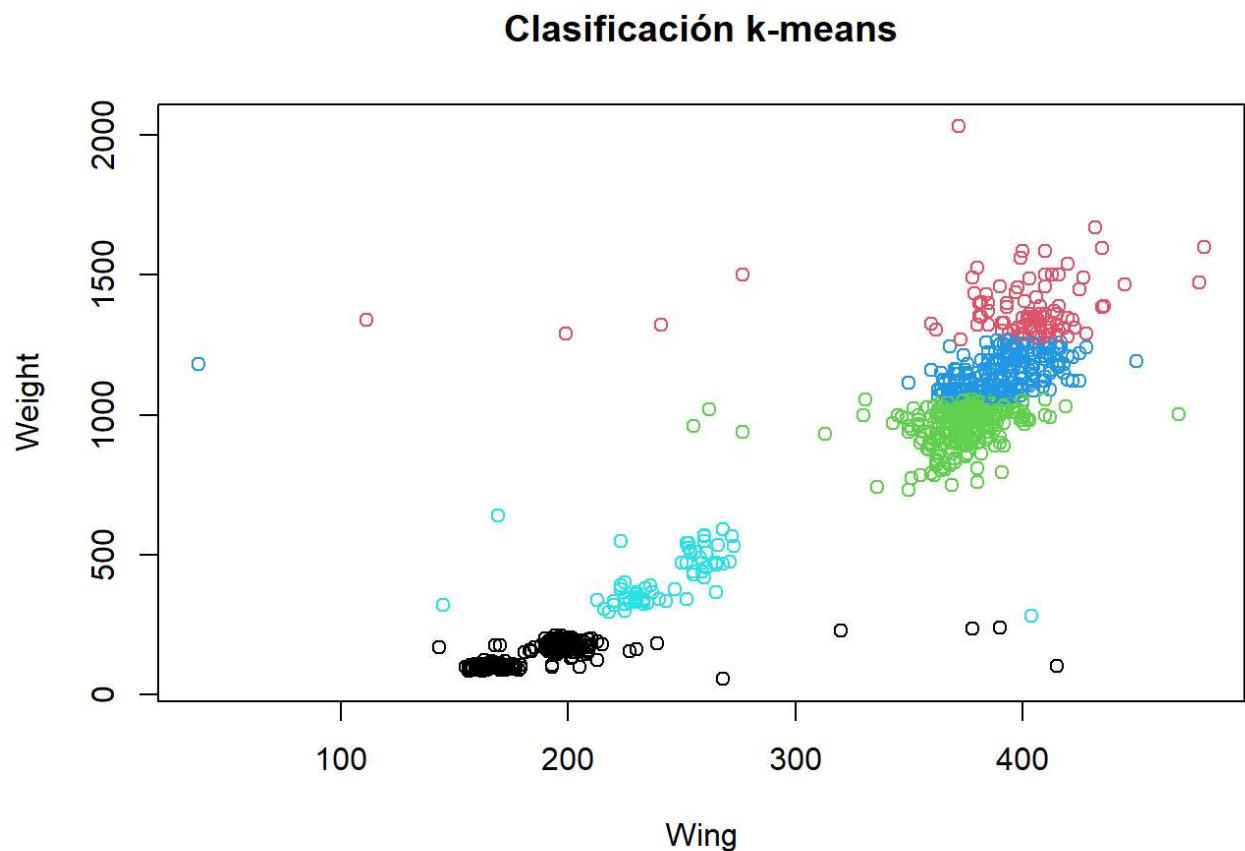
CLUSPLOT(Hawks_data[c(3, 4)])



Component 1

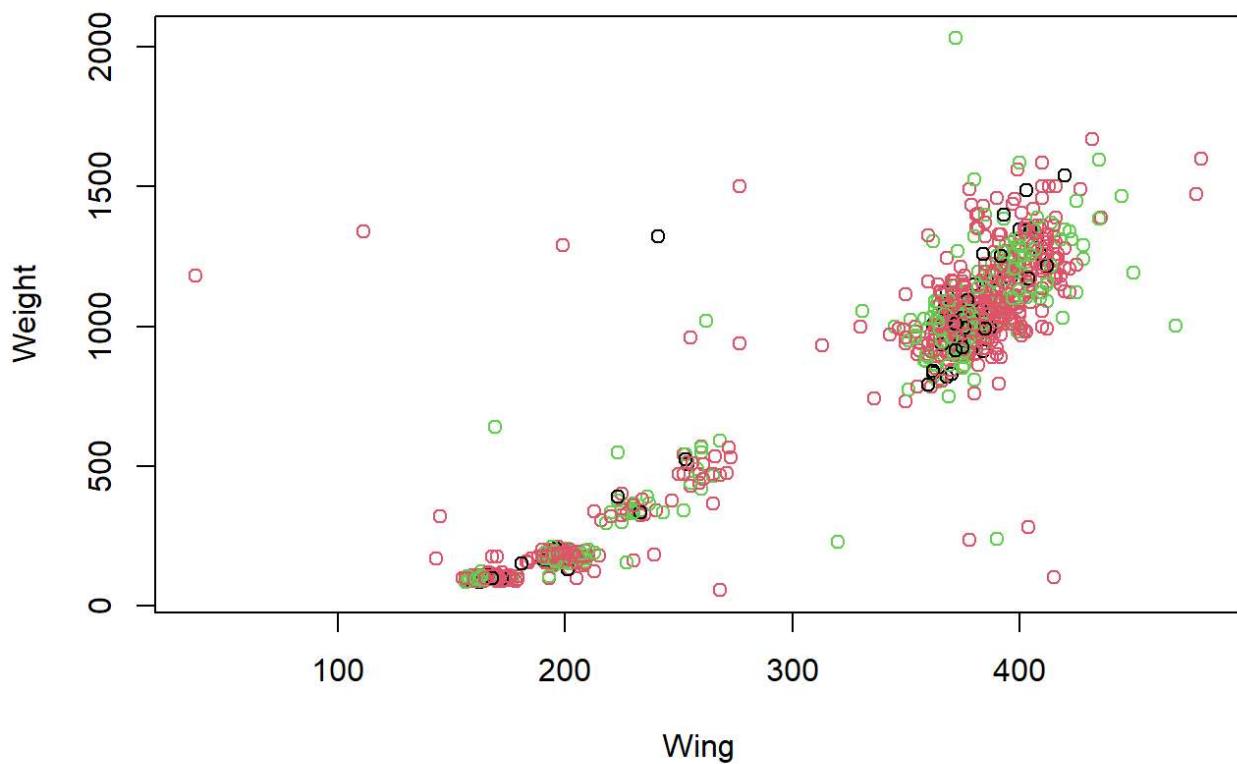
These two components explain 100 % of the point variability.

```
plot(Hawks_data[c(1,2)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```



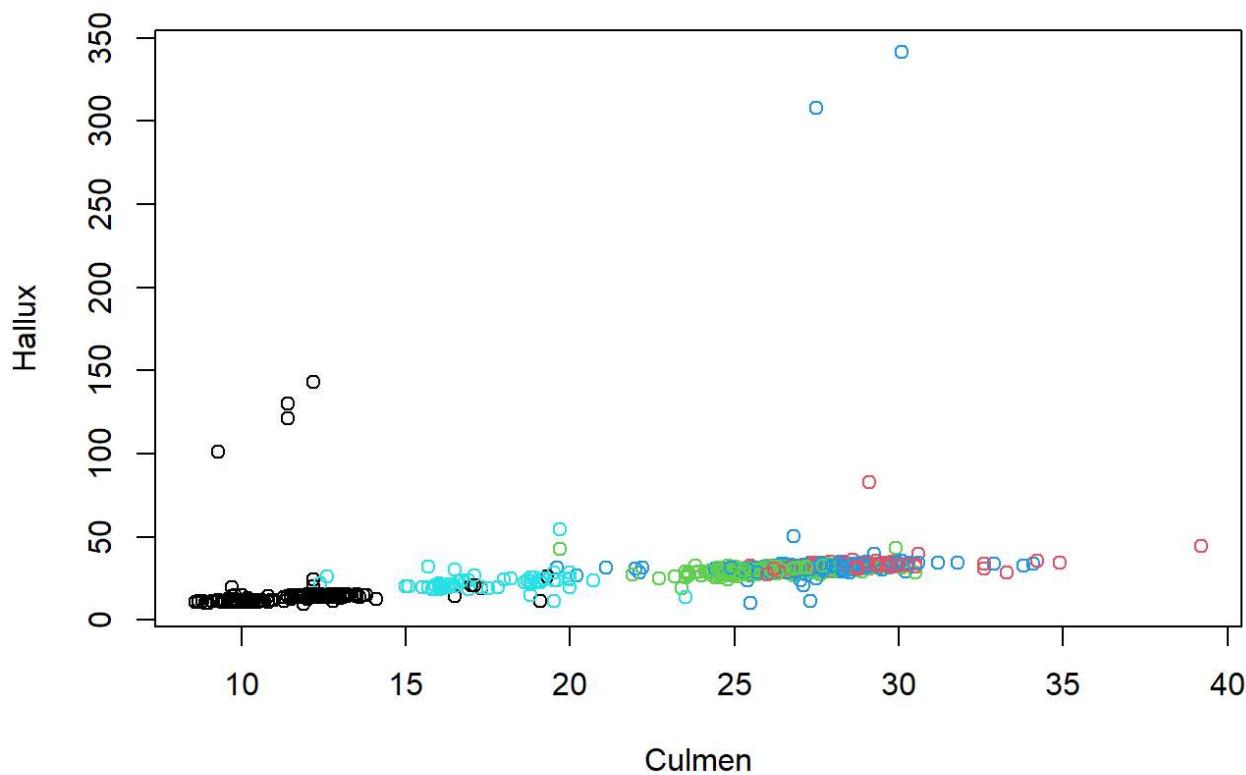
```
plot(Hawks_data[c(1,2)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real

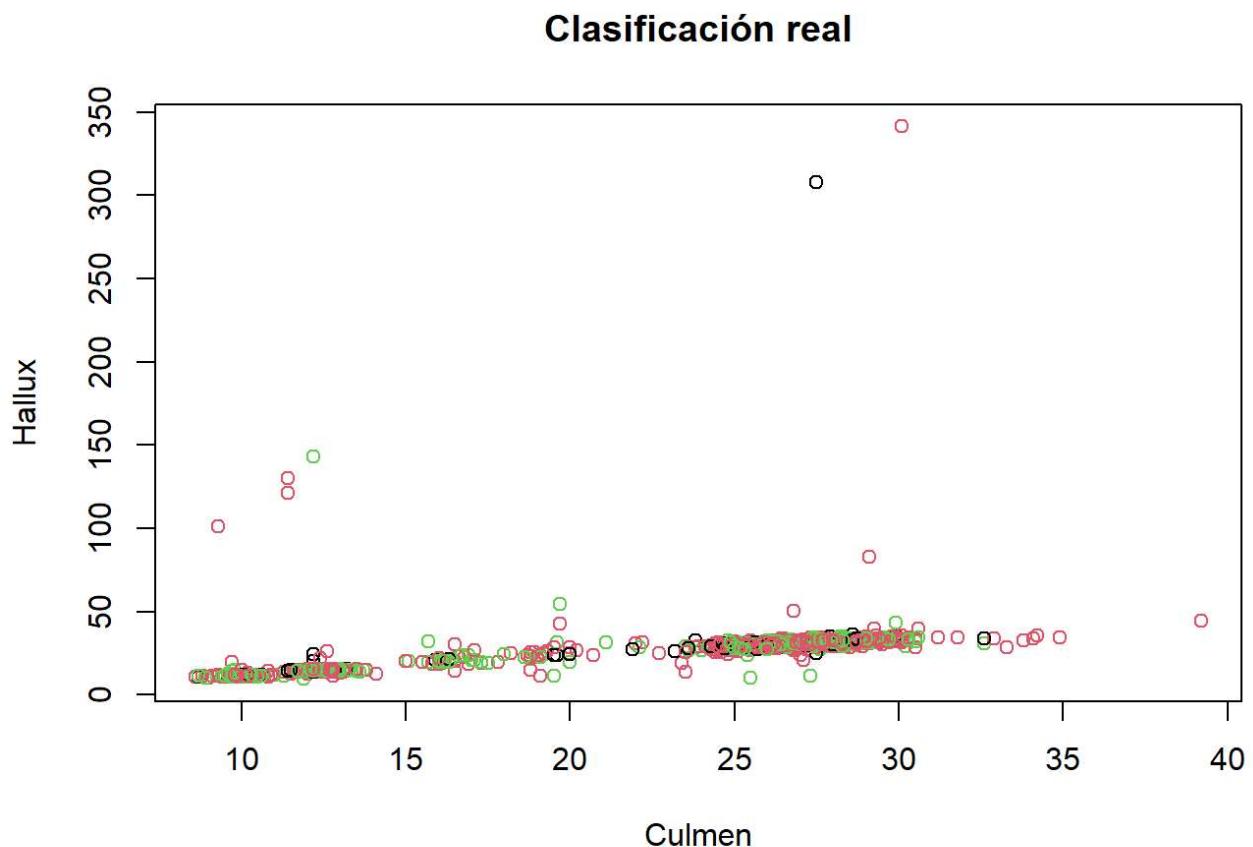


```
plot(Hawks_data[c(3,4)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means

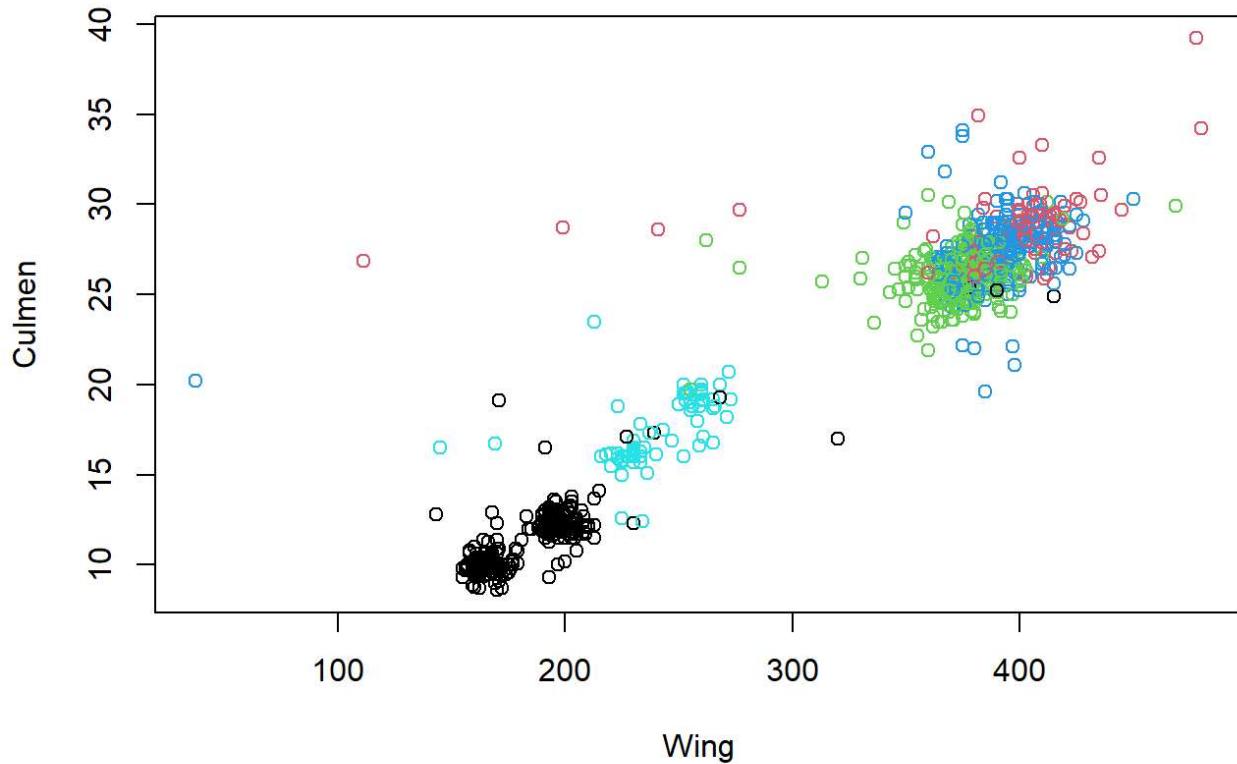


```
plot(Hawks_data[c(3,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```



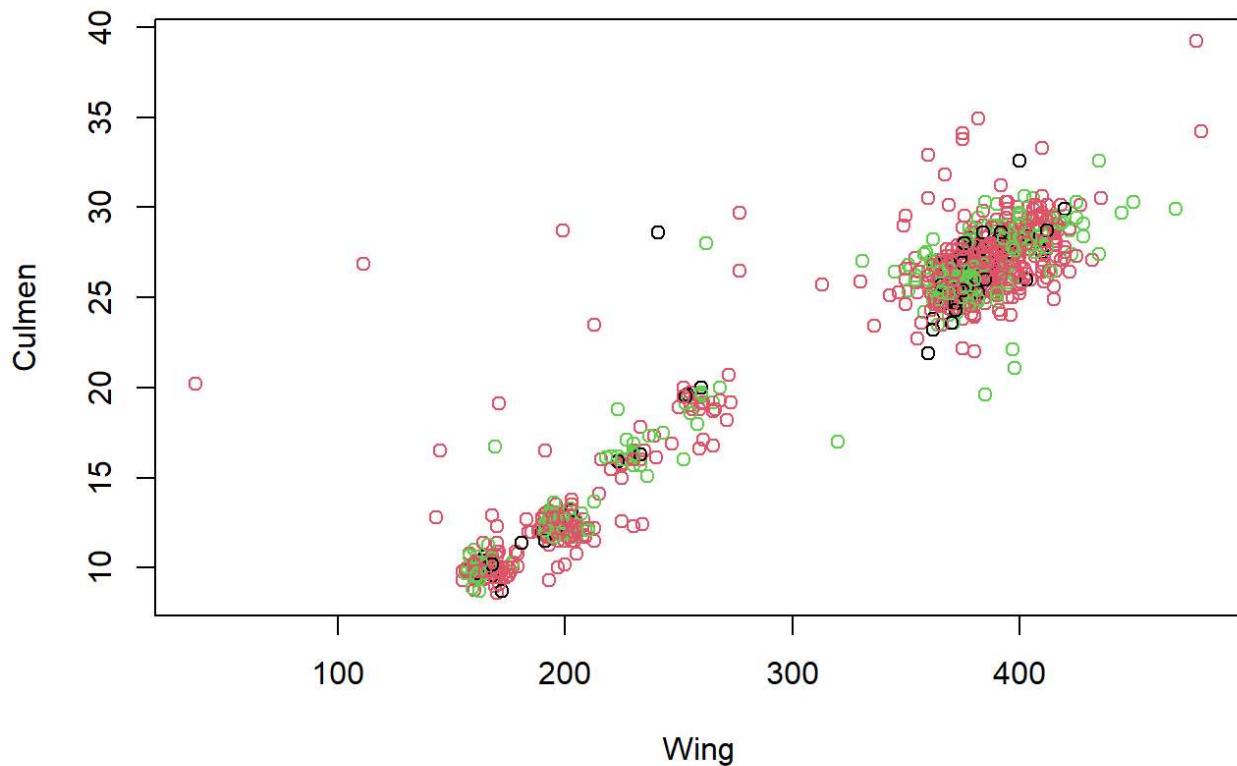
```
plot(Hawks_data[c(1,3)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means

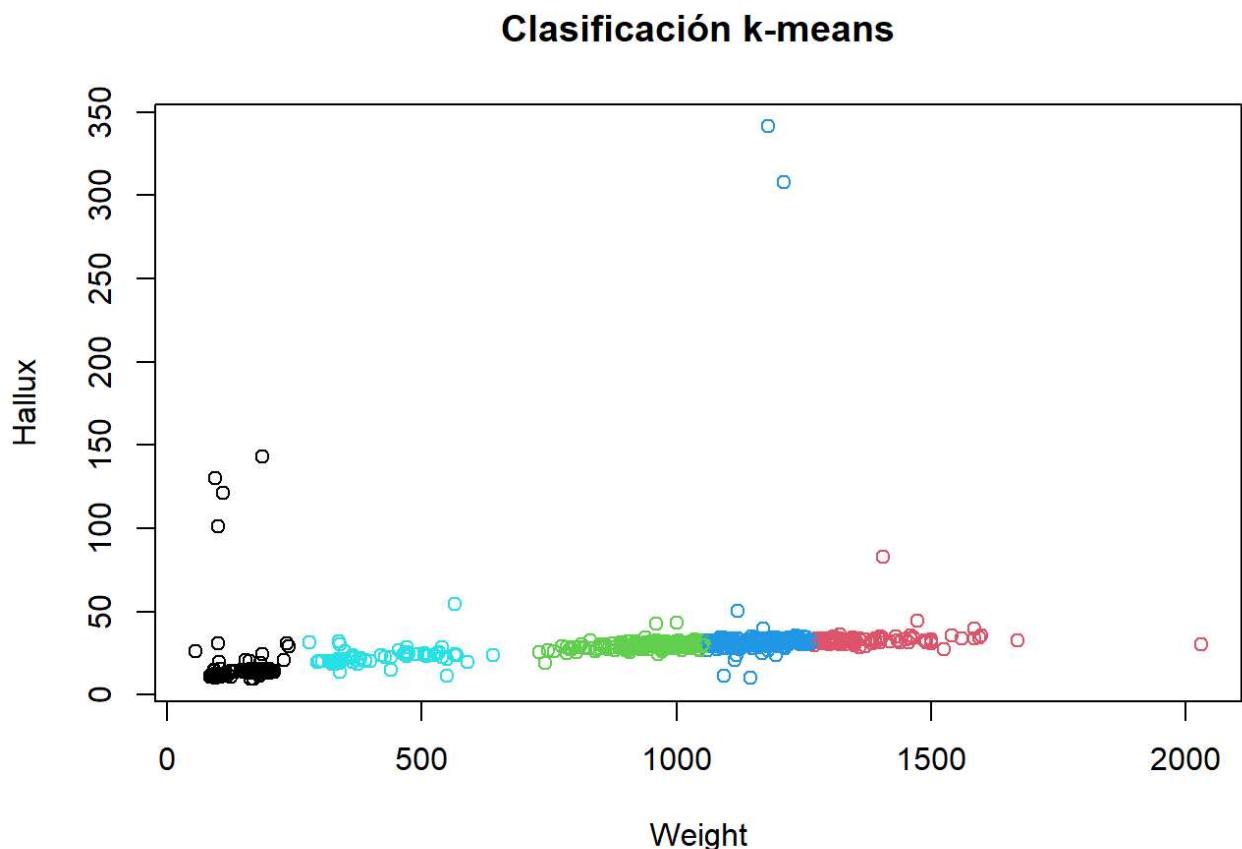


```
plot(Hawks_data[c(1,3)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real

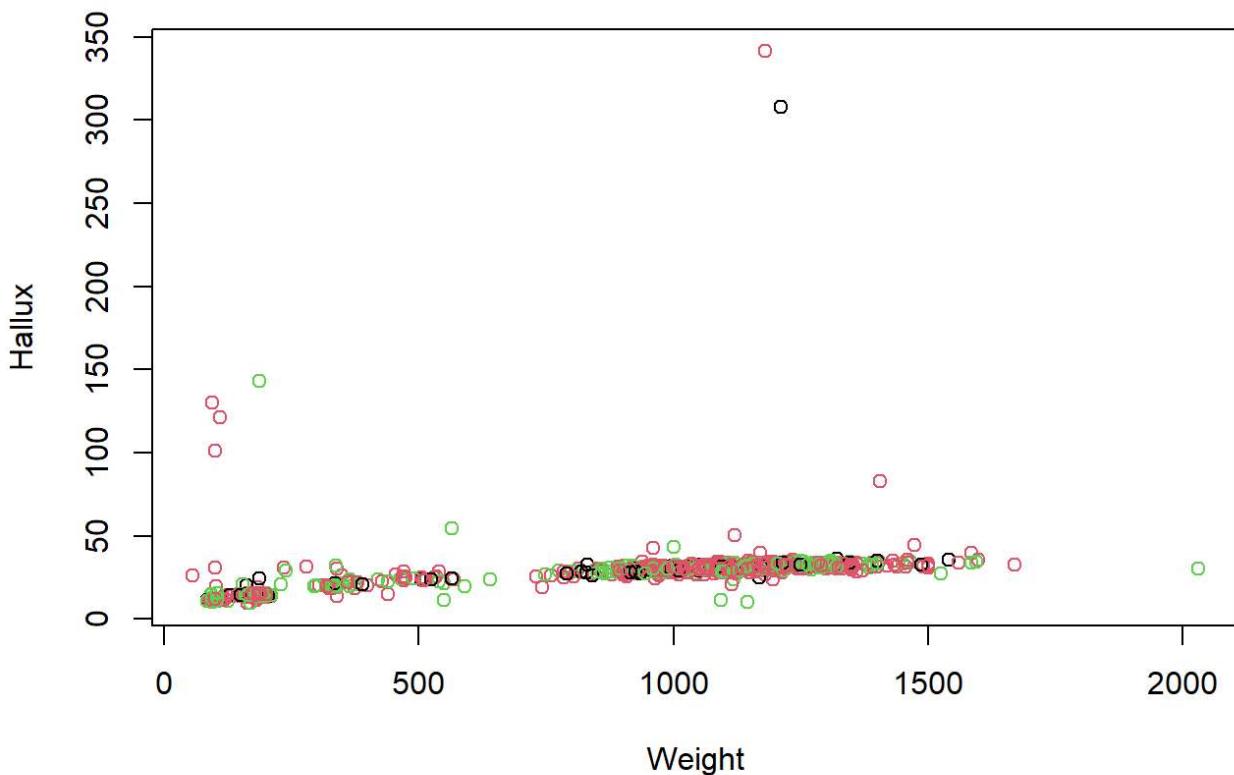


```
plot(Hawks_data[c(2,4)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```



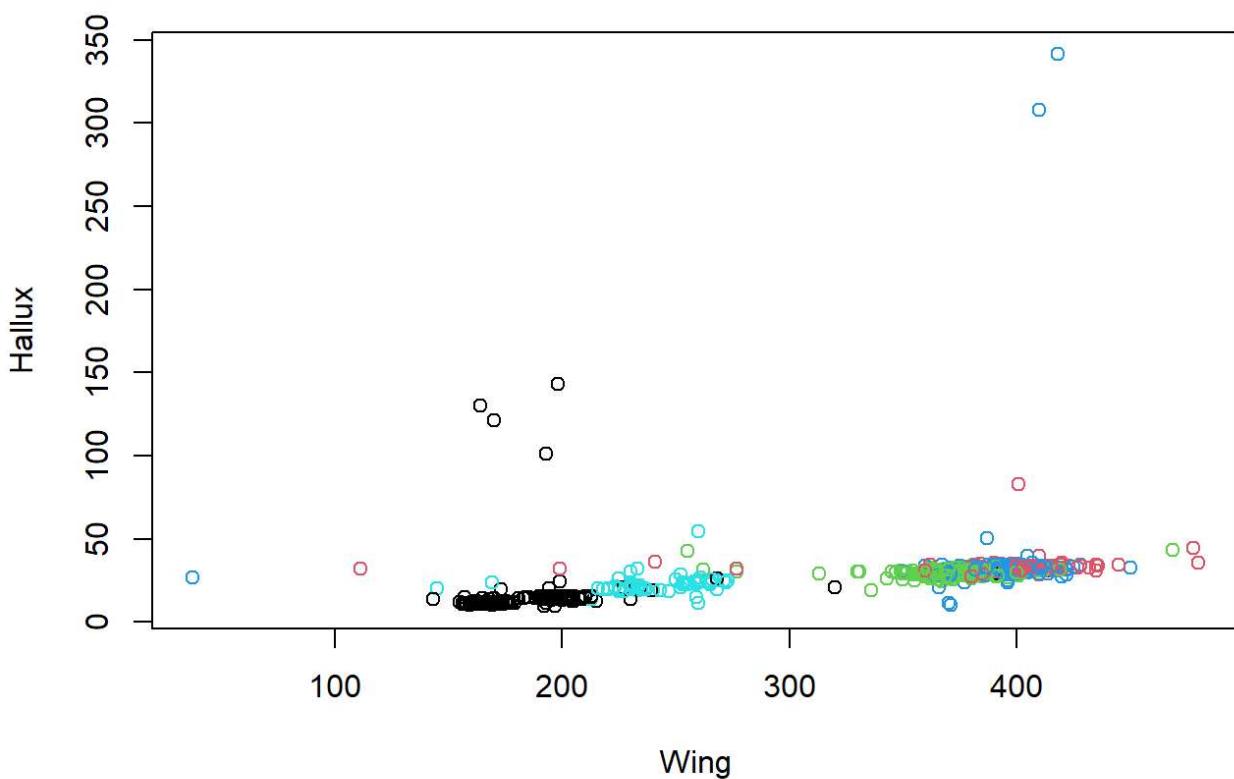
```
plot(Hawks_data[c(2,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real

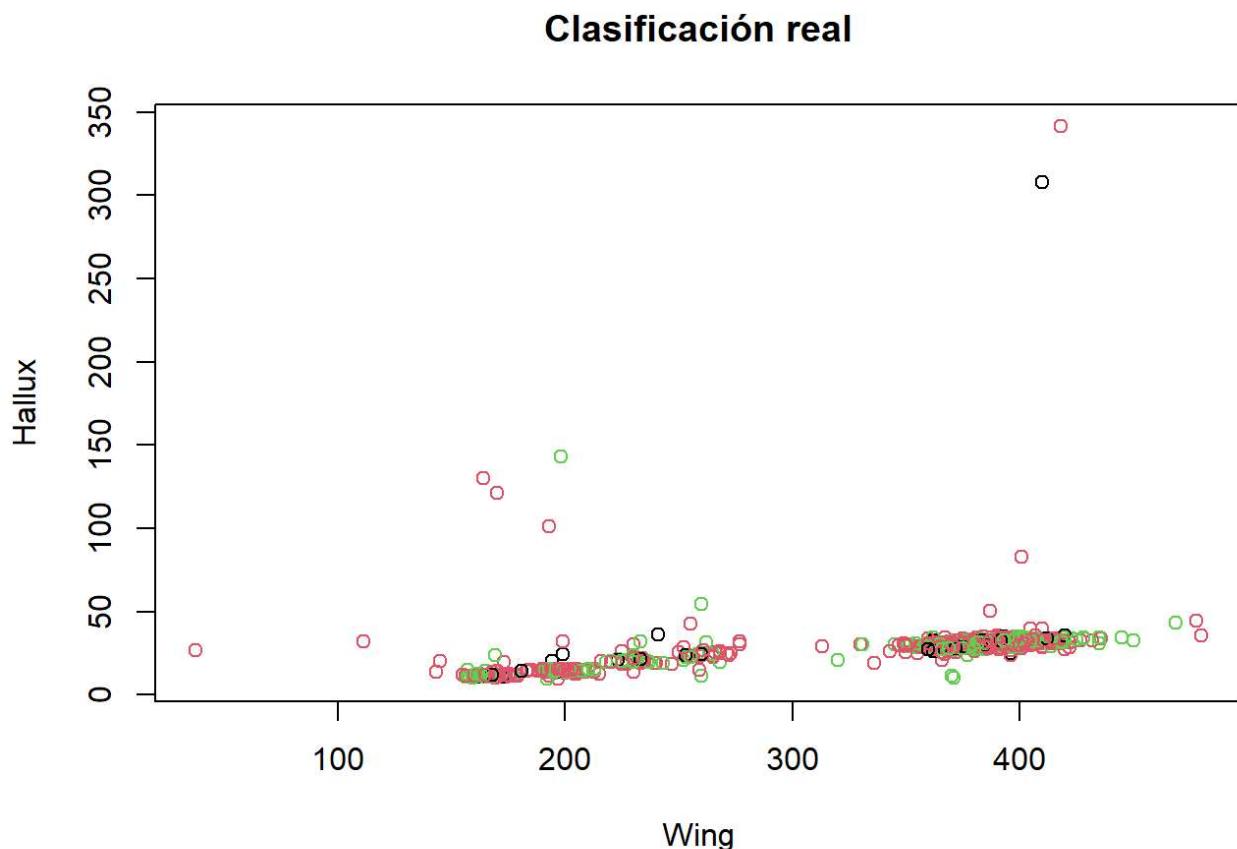


```
plot(Hawks_data[c(1,4)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means

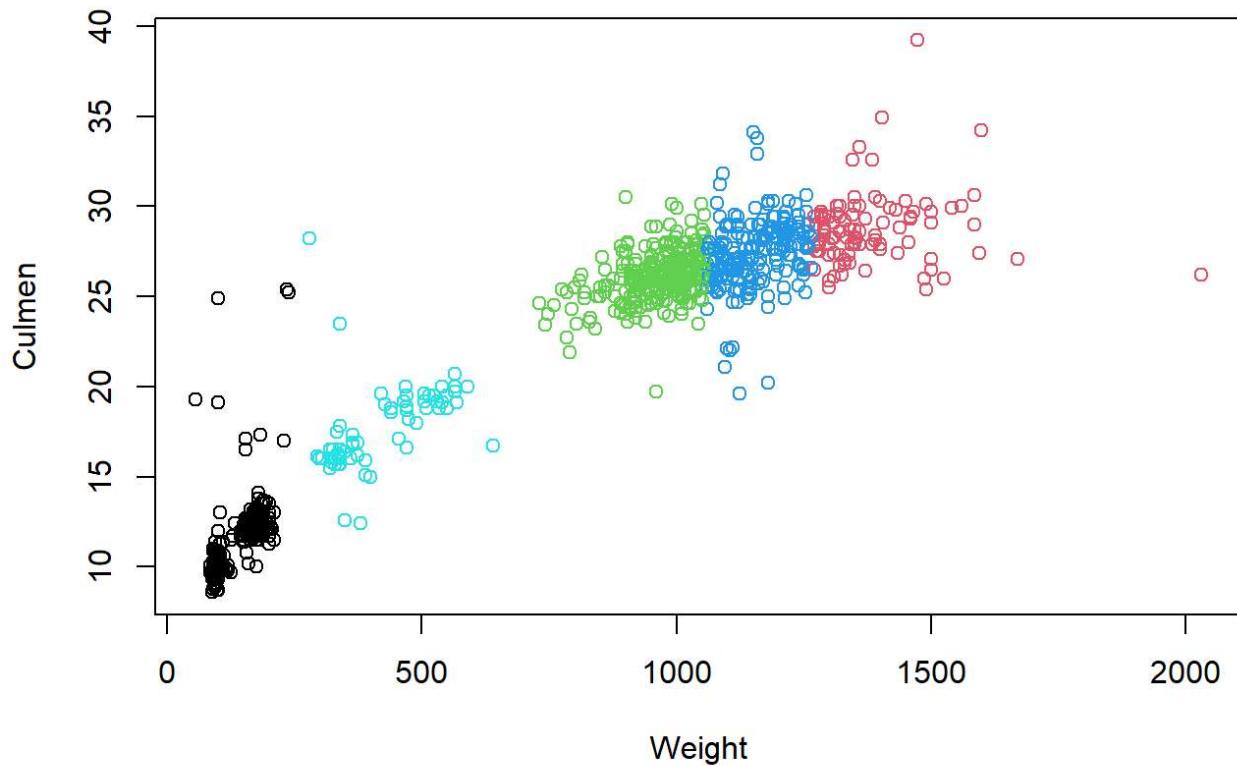


```
plot(Hawks_data[c(1,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```



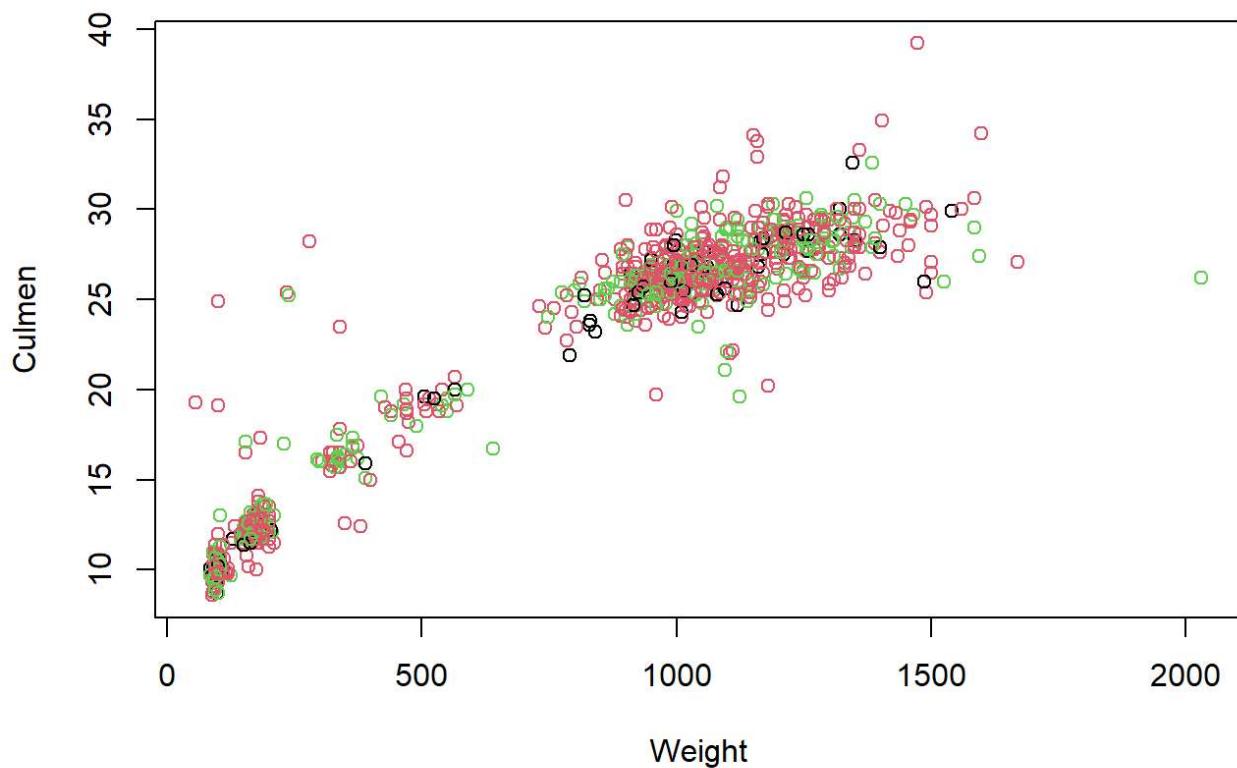
```
plot(Hawks_data[c(2,3)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



```
plot(Hawks_data[c(2,3)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real



Finalmente podemos observar que con 5 clústeres tampoco se obtiene una diferenciación de especies clara.

Ahora vamos a evaluar la calidad del proceso de agregación.

```
d <- daisy(Hawks_data)
sk5 <- silhouette(Hawks5clusters$cluster, d)
```

Calculando la media de la tercera columna podemos obtener una estimación de la calidad del agrupamiento

```
mean(sk5[,3])
```

```
## [1] 0.5934635
```

Tras realizar todas las combinaciones posibles con todos los campos y clústeres, concluyo que ninguna combinación nos permite obtener una diferenciación de las especies de una manera clara, no obstante podemos comprobar que la estimación de la calidad del agrupamiento es más alta con 3 clústeres.

5.2 Ejercicio 2

Con el juego de datos proporcionado realiza un estudio similar al del ejemplo 1.3

5.2.1 Respuesta 2

Escribe aquí la respuesta a la pregunta

Instalamos las librerías necesarias

```
if (!require('dbSCAN')) install.packages('dbSCAN')
library(dbSCAN)
```

Lanzamos el algoritmo OPTICS dejando el parámetro eps con su valor por defecto y fijando el criterio de vecindad en 10
res10 <- optics(Hawks_data, minPts = 10)
res10

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 10, eps = 531.584706326283, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi
```

Lanzamos el algoritmo OPTICS dejando el parámetro eps con su valor por defecto y fijando el criterio de vecindad en 20
res20 <- optics(Hawks_data, minPts = 20)
res20

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 20, eps = 575.591808489315, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi
```

```
### Lanzamos el algoritmo OPTICS dejando el parámetro eps con su valor por defecto y fijando el criterio de vecindad en 50
res50 <- optics(Hawks_data, minPts = 50)
res50
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 50, eps = 686.699861657187, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi
```

```
### Obtenemos la ordenación de las observaciones o puntos
res10$order
```

```

## [1] 1 830 556 528 851 710 464 370 354 287 251 259 540 656 544 17 314 215 480 223
75 542 456 249 101 254 560 200 728 543 597 452 148 757 580 811 599 563 555 278 253 151 1
34 16 531 494 315
## [48] 244 884 691 510 545 846 828 426 6 663 380 145 144 130 717 515 632 816 784 699
665 596 565 773 685 566 754 604 589 501 326 118 114 88 23 2 516 413 179 286 150 113
855 748 729 469 108
## [95] 473 807 845 839 634 860 762 467 453 397 197 11 457 749 874 585 881 578 534 308
74 808 262 375 238 491 180 132 126 87 414 324 459 160 275 273 229 43 410 111 359 270 2
16 742 724 633 492
## [142] 727 446 832 591 573 771 451 368 243 45 284 746 650 577 193 185 141 188 333 755
344 226 539 443 277 85 44 436 199 175 758 362 355 82 40 129 875 592 154 327 796 601
424 289 28 876 500
## [189] 871 701 42 417 328 744 490 266 178 176 173 18 10 769 174 149 381 392 233 30
485 862 274 512 631 590 586 550 536 441 393 191 65 613 768 680 332 776 766 574 335
329 582 395 655 345
## [236] 33 142 779 421 177 172 161 24 651 158 146 703 164 676 487 449 211 194 46 257
240 423 693 420 203 202 112 615 357 152 97 282 759 518 475 204 131 829 593 537 83 66
168 133 12 865 753
## [283] 482 356 292 234 41 535 483 471 9 19 334 838 217 382 301 48 448 569 867 346
337 67 153 800 252 105 5 267 336 716 715 642 782 675 752 625 430 415 401 400 119 72
700 575 557 508 606
## [330] 795 616 600 576 488 367 201 831 378 374 25 594 503 478 525 707 689 548 498 468
408 341 250 70 432 856 702 532 135 636 522 115 92 8 281 13 649 817 398 225 222 218
181 169 256 219 140
## [377] 850 801 859 761 734 690 666 372 228 187 60 340 55 51 239 77 788 283 427 513
520 863 455 678 313 20 389 387 109 869 598 866 780 524 230 770 265 774 764 15 338 198
52 736 853 840 261
## [424] 255 247 743 213 90 667 570 325 241 47 562 422 272 248 527 466 245 128 62 57
31 712 822 688 157 49 827 339 568 406 523 750 231 581 885 844 551 442 379 377 263 182 1
59 121 236 76 122
## [471] 68 285 84 777 22 34 104 258 321 366 660 751 470 890 852 837 438 110 7 439
658 877 842 322 235 847 521 29 391 635 653 89 295 100 323 883 698 886 458 889 679 496
454 69 781 533 399
## [518] 349 705 477 14 290 59 538 648 465 396 71 143 386 493 363 237 460 371 170 27
73 32 35 317 36 887 288 264 184 351 127 864 484 462 684 352 704 587 388 186 683 56 8
91 271 431 725 686
## [565] 670 793 772 664 657 823 572 526 445 343 242 63 3 561 428 232 302 803 695 318
403 312 390 514 227 103 291 861 756 834 835 826 814 692 627 411 361 81 54 509 309 383
517 39 316 872 279
## [612] 348 53 618 429 792 810 726 583 280 805 214 741 106 507 156 879 825 878 880 882
868 783 778 747 735 671 463 394 304 330 298 812 760 669 637 192 661 719 628 530 504 418
365 269 165 26 402
## [659] 659 644 603 579 472 433 763 733 718 722 791 804 786 731 824 608 541 499 730 888
505 785 820 711 672 668 619 609 461 434 502 706 767 385 870 819 787 709 681 614 605 440
425 305 404 806 595
## [706] 331 208 166 412 205 155 714 623 450 447 307 212 99 276 293 297 163 4 79 196
268 310 138 78 171 93 360 553 552 444 364 373 220 765 673 195 167 607 486 162 474 124
246 677 841 646 189
## [753] 798 94 319 809 50 221 342 350 210 626 347 183 848 873 479 740 723 836 697 654
647 643 630 546 405 737 610 549 745 358 821 739 554 529 435 384 858 721 849 790 713 617
588 409 407 303 300
## [800] 306 376 294 571 559 813 857 854 843 815 802 640 638 612 489 621 299 639 416 674

```

```
611 481 120 102  96 682 645 797 794 720 696 506 495 296  61 519 320 190 260 139  80 818  
732 353 620 558 311  
## [847] 116 137 207  64 799 708 641 833 694 547 437  38 789 511 117 107 775 622 369 224  
136  95 125 206 123 652 564 147 419 209 497  37 602 567 624  98 738 687 629 476  21  58  
86   91 662
```

```
res20$order
```

```

## [1] 1 851 830 710 656 597 560 556 544 542 540 528 480 464 370 354 314 287 251 215
17 259 223 75 456 254 249 728 452 148 101 200 862 693 512 423 274 203 543 757 240 420 2
02 580 838 494 884
## [48] 811 691 599 846 828 663 563 531 510 426 315 278 244 144 134 145 16 6 717 545
151 596 632 380 130 111 555 515 253 816 784 773 699 685 566 565 516 491 457 414 413 808
754 604 881 589 585
## [95] 501 469 459 326 179 160 118 114 88 23 262 126 113 2 375 286 238 150 855 807
729 275 229 108 43 748 180 87 473 845 839 634 197 11 860 762 749 601 467 453 397 273
490 270 328 874 744
## [142] 132 359 324 742 727 633 578 534 492 410 368 344 308 243 226 216 188 129 74 45
327 10 871 832 796 591 573 771 755 746 650 451 446 333 284 193 185 141 577 539 277 175
40 443 436 424 362
## [189] 355 289 199 82 381 42 28 875 758 592 154 85 44 417 701 176 724 500 769 257
178 173 18 876 30 665 392 105 112 703 676 174 152 149 233 48 387 266 301 448 282 485
655 768 776 766 680
## [236] 631 613 590 586 584 550 536 441 393 335 332 329 191 177 158 33 65 574 582 395
164 779 487 161 24 345 142 759 651 172 449 421 146 46 204 19 569 334 615 475 194 12
764 357 217 678 109
## [283] 97 866 518 313 131 83 336 520 211 153 9 265 52 598 321 406 865 780 770 593
537 524 230 66 41 5 231 382 829 292 168 133 535 483 482 471 356 267 234 867 863 827
800 753 715 625 401
## [330] 346 337 67 716 675 642 513 430 782 752 700 616 606 576 575 557 508 415 400 374
201 831 795 594 522 503 478 468 408 378 367 341 856 702 689 636 548 532 525 498 250 135
92 70 8 432 25
## [377] 600 119 72 649 140 115 817 707 340 218 488 281 13 398 225 222 169 219 20 774
181 256 869 455 850 761 734 859 801 690 372 239 228 187 77 60 55 666 427 283 51 788
258 338 252 389 15
## [424] 104 198 853 840 736 261 255 247 213 90 743 366 570 325 157 523 49 568 688 822
667 712 562 527 466 777 581 422 285 272 248 245 241 128 62 47 31 57 34 22 339 750
635 751 439 844 551
## [471] 442 379 377 263 121 84 68 885 236 182 159 76 323 470 391 122 235 890 837 660
110 7 847 454 698 653 883 438 852 89 69 14 143 658 781 533 399 349 493 877 842 322
290 237 521 29 886
## [518] 458 295 100 889 705 679 496 538 477 648 396 184 59 127 465 460 386 71 363 352
170 317 264 371 35 27 73 351 36 32 887 704 587 288 484 462 864 684 683 388 186 56
891 271 431 793 823
## [565] 772 686 670 664 657 803 695 572 561 526 445 428 343 318 302 242 232 63 3 312
227 514 390 725 403 103 741 861 835 872 834 826 814 756 692 627 517 509 411 383 361 348
316 309 279 81 54
## [612] 39 53 618 429 810 792 583 805 726 280 106 214 291 507 156 879 878 882 880 868
791 783 812 778 760 747 735 671 661 719 644 628 603 579 530 504 433 418 402 365 330 298
269 165 26 637 463
## [659] 394 304 659 804 731 472 763 730 718 331 595 192 733 722 786 541 706 888 824 785
668 609 608 505 499 820 819 767 711 672 870 806 787 709 681 605 461 440 434 425 412 404
310 305 614 619 502
## [706] 208 205 669 385 163 4 714 623 450 447 307 293 212 825 360 196 138 99 166 155
79 78 373 93 171 297 798 276 167 268 319 474 646 189 220 162 765 553 552 444 364 246 1
24 673 486 607 677
## [753] 195 94 841 809 50 221 342 350 626 347 210 183 848 873 647 836 745 740 737 732
723 721 858 849 821 790 739 713 617 588 554 529 435 409 407 376 857 854 813 674 640 639
638 843 815 802 621
## [800] 612 571 559 489 416 300 306 303 294 61 96 299 611 797 794 720 696 620 519 506

```

```
495 481 296 558 311 102 320 358 546 384 353 833 610 549 405 643 630 654 682 120 697 190  
645 437 818 641 80  
## [847] 260 511 789 708 139 694 547 207 799 137 116 38 64 125 117 479 95 206 107 775  
622 369 224 136 123 147 564 652 419 209 497 37 602 567 624 98 738 687 629 476 21 58  
86 91 662
```

```
res50$order
```

```

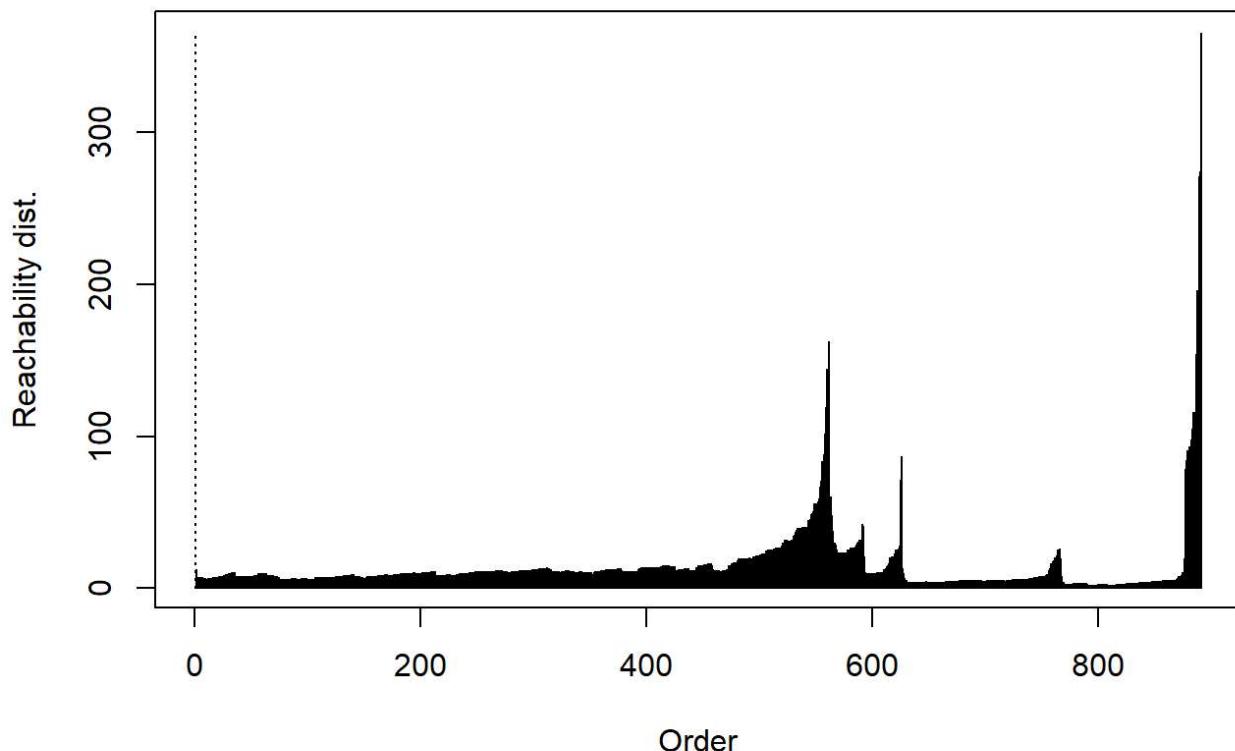
## [1] 1 884 846 828 816 811 784 773 881 874 860 855 845 839 808 807 762 754 749 748
744 729 634 604 601 589 585 566 516 501 490 473 469 467 459 453 397 328 326 275 273 270
229 197 179 178 176
## [48] 173 160 118 114 108 88 43 23 18 11 10 2 685 457 413 262 126 113 327 286
238 150 257 308 266 216 188 74 699 375 180 87 387 132 112 105 48 491 226 565 414 111
578 534 871 796 742
## [95] 701 633 492 446 410 368 243 42 30 334 359 324 632 282 832 727 591 573 381 344
129 45 380 392 130 771 755 724 500 451 417 333 284 141 109 28 665 663 596 426 144 746
650 577 539 443 355
## [142] 277 193 185 175 85 44 40 592 436 424 362 289 199 154 876 82 233 336 6 717
678 563 531 515 278 145 134 16 758 267 691 599 580 555 545 510 494 315 253 244 151 875
769 764 569 448 869
## [189] 301 231 703 676 655 536 393 174 164 152 149 33 19 65 382 338 389 345 485 357
768 631 590 776 766 680 613 586 584 582 574 550 487 441 395 335 332 329 191 177 161 158
142 97 838 757 217
## [236] 75 779 759 651 449 421 211 204 172 146 46 24 5 194 866 780 615 524 518 475
321 313 12 598 223 851 830 728 710 656 560 556 544 543 542 540 528 480 464 370 354 314
287 259 254 251 249
## [283] 215 200 17 131 597 230 456 83 452 148 101 770 258 635 153 66 9 862 512 423
274 240 203 520 829 593 537 693 235 523 252 157 49 420 750 202 867 865 753 535 483 482
471 356 292 234 168
## [330] 133 41 265 827 800 716 715 625 513 439 401 346 337 67 415 863 782 774 752 700
675 642 616 606 576 575 557 508 488 455 430 400 374 339 281 219 119 72 25 13 600 15
406 201 856 831 795
## [377] 702 689 636 594 548 525 522 503 498 478 468 408 378 367 366 341 250 140 135 115
92 70 20 8 52 850 817 734 707 649 532 432 398 340 256 228 225 222 218 187 181 169
60 761 690 372 104
## [424] 55 77 323 859 801 788 666 427 283 239 51 198 493 890 837 688 660 110 7 736
90 143 853 840 743 570 568 325 261 255 247 213 237 822 667 562 527 422 272 248 241 128
62 47 31 852 466
## [471] 438 777 751 712 698 245 57 34 22 285 877 658 581 885 883 844 551 470 442 391
379 377 263 182 159 121 84 68 236 76 322 842 122 847 653 454 89 69 29 521 295 781
533 14 399 100 352
## [518] 349 886 458 290 705 127 889 496 679 184 704 538 648 396 587 477 465 460 386 71
170 363 59 371 35 317 264 351 27 73 36 32 887 288 56 683 484 864 462 684 388 891
271 186 431 514 390
## [565] 686 725 738 312 58 561 861 872 879 882 880 878 868 812 804 888 870 824 820 787
786 785 783 735 731 730 719 711 709 706 672 668 661 659 644 671 628 619 614 609 608 605
603 579 541 530 505
## [612] 504 499 472 440 434 433 418 402 365 331 330 298 269 165 163 155 26 4 502 463
461 394 778 747 718 637 304 425 763 760 208 93 791 733 722 669 595 319 196 192 138 825
819 385 78 171 360
## [659] 767 166 806 681 412 404 310 305 205 714 623 450 447 373 307 293 212 99 79 167
297 646 189 798 246 162 765 553 552 444 364 677 474 276 268 220 124 673 195 486 607 94
841 156 809 342 350
## [706] 626 50 221 347 183 210 848 419 858 857 854 849 843 836 821 813 790 745 740 739
737 723 721 833 815 802 732 713 697 654 643 640 639 638 630 617 612 611 610 588 571 559
554 549 546 529 489
## [753] 435 416 409 407 384 376 358 306 303 621 405 300 294 96 61 818 694 674 547 794
696 682 645 558 519 506 495 481 320 299 120 102 80 647 797 353 260 116 720 38 311 296
190 620 137 139 207
## [800] 708 64 479 125 117 147 123 511 107 95 799 206 224 136 641 437 369 789 775 622

```

```
652 564 873 209 497 37 805 624 810 602 567 792 98 835 834 826 814 756 741 726 692 627  
618 583 517 509 507  
## [847] 429 411 383 361 348 316 309 280 279 214 106 81 54 53 39 291 823 803 695 657  
572 526 445 403 793 772 664 343 318 302 242 232 103 63 3 670 428 227 687 629 476 21  
86 91 662
```

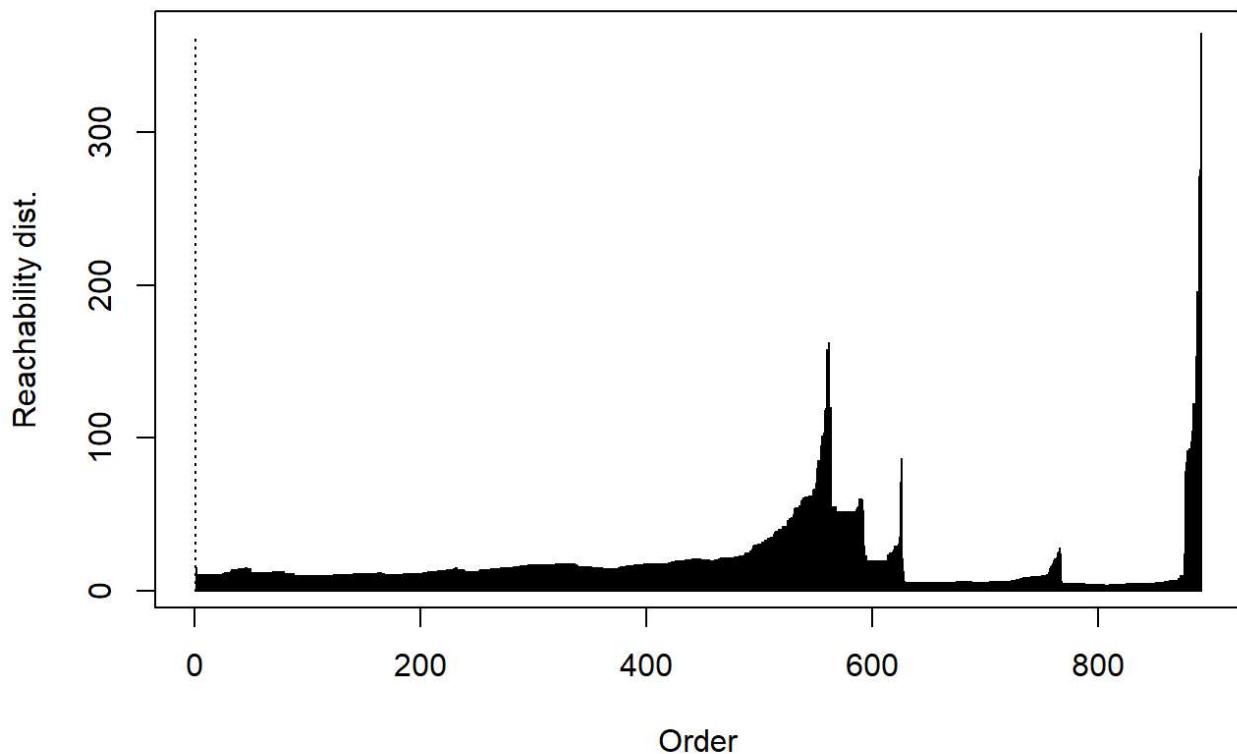
```
### Gráficas de alcanzabilidad  
plot(res10)
```

Reachability Plot



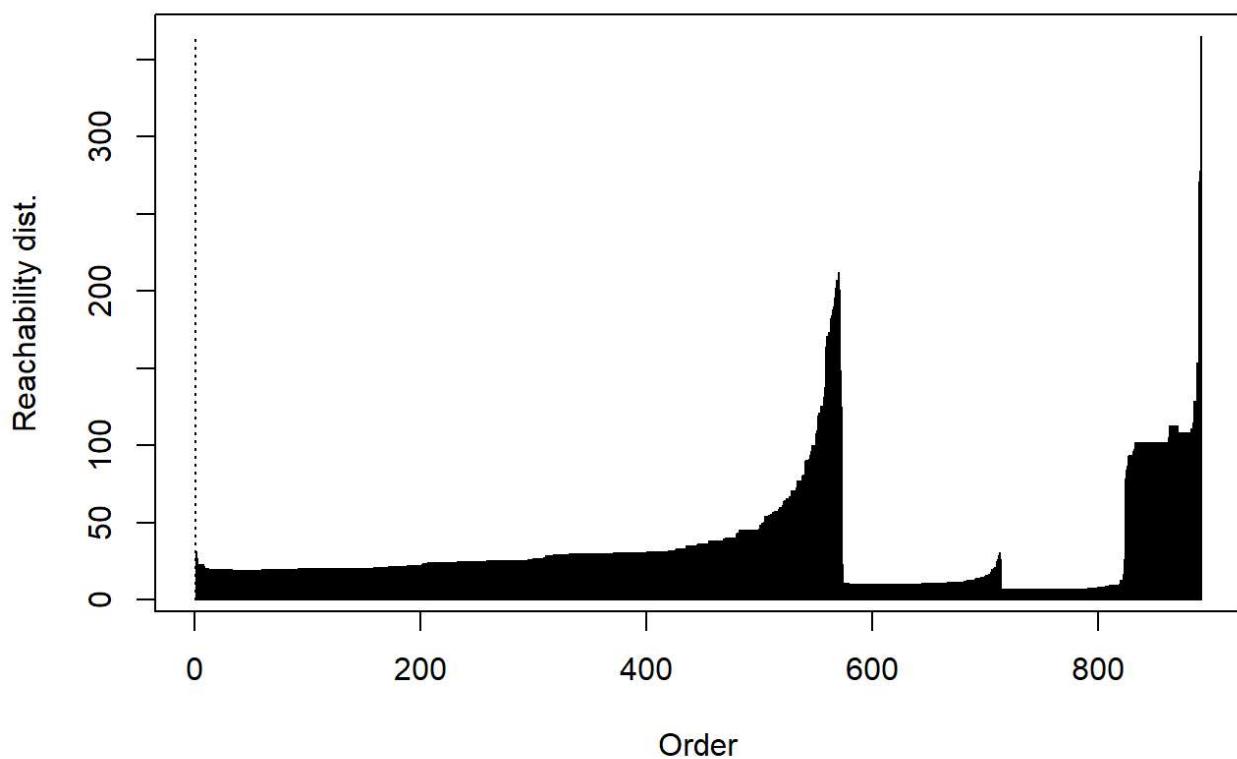
```
plot(res20)
```

Reachability Plot



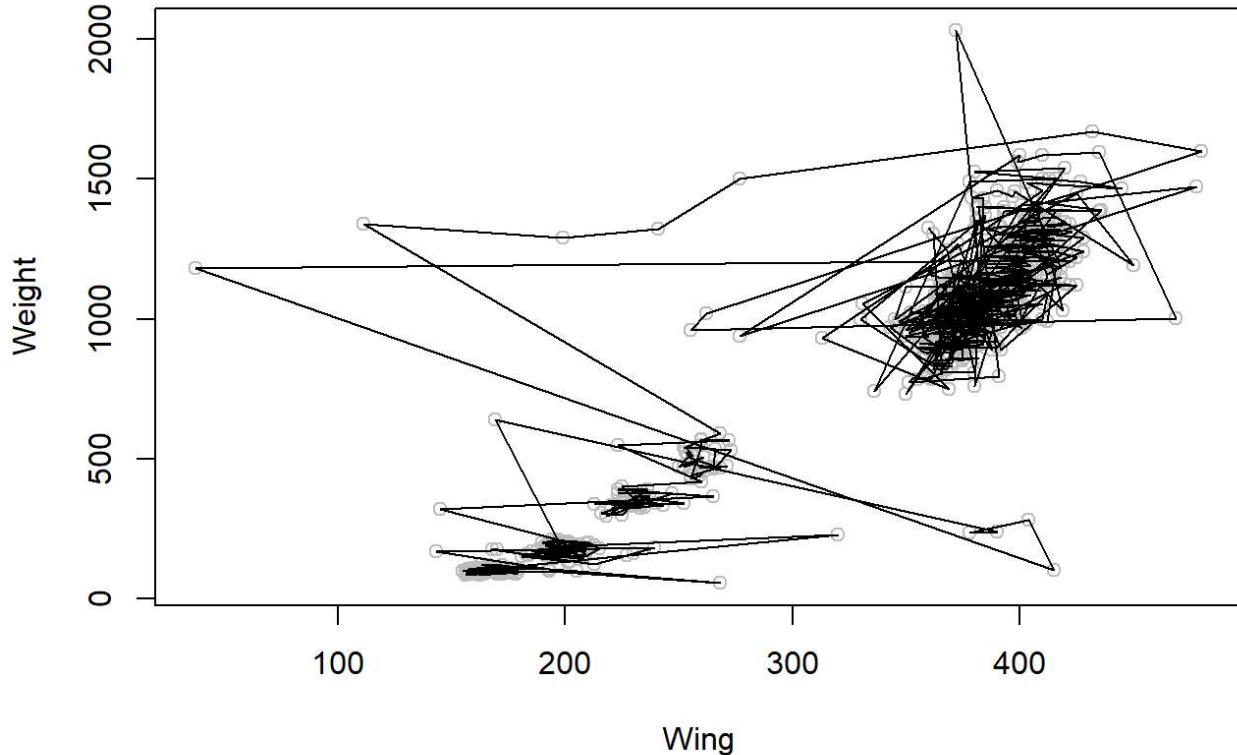
```
plot(res50)
```

Reachability Plot

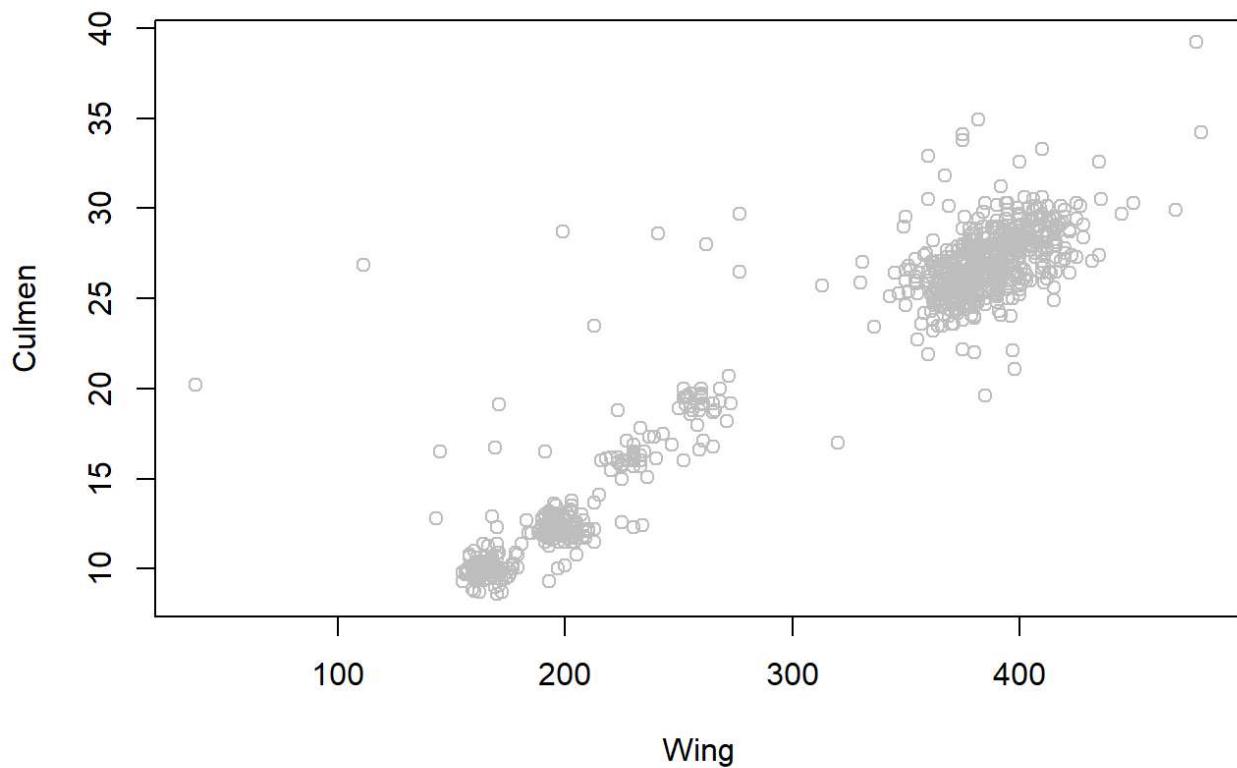


Ahora haremos el proceso con el criterio de vecindad en 10, lo primero seria obtener la relación entre las variables.

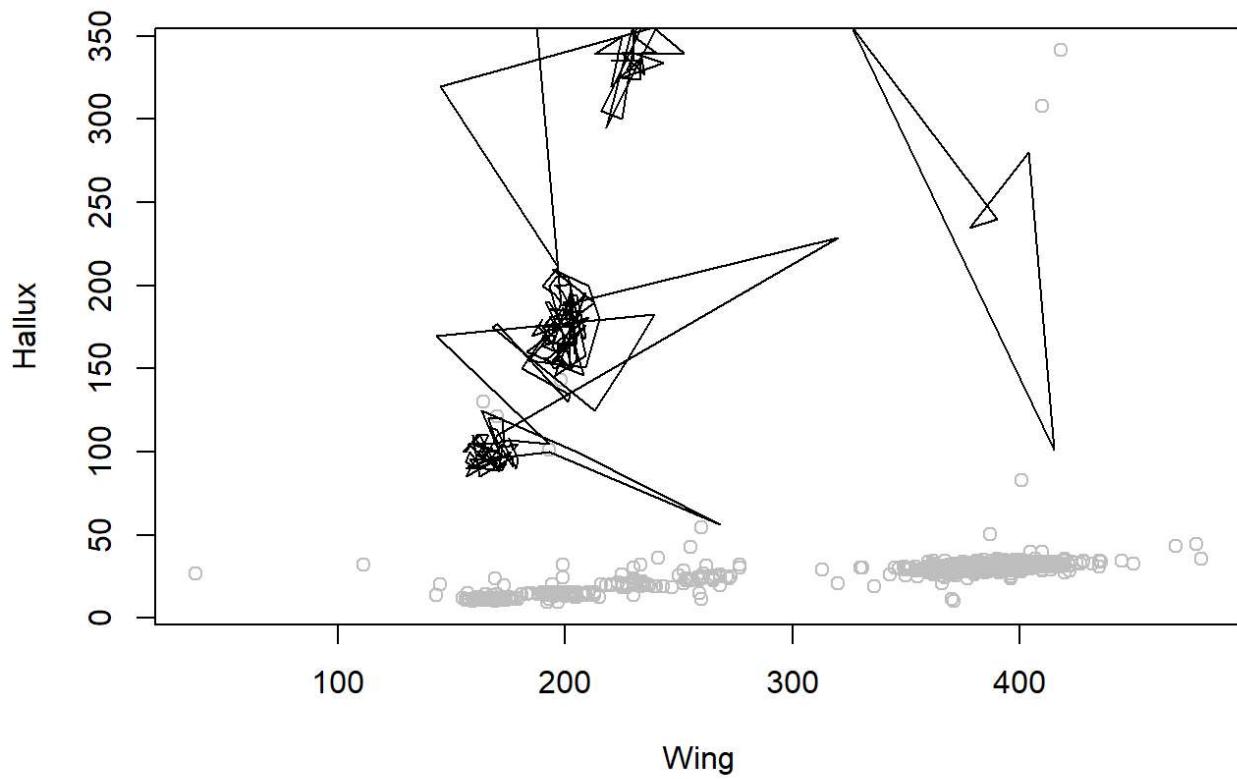
```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data[c(1,2)], col = "grey")
polygon(Hawks_data[res10$order,])
```



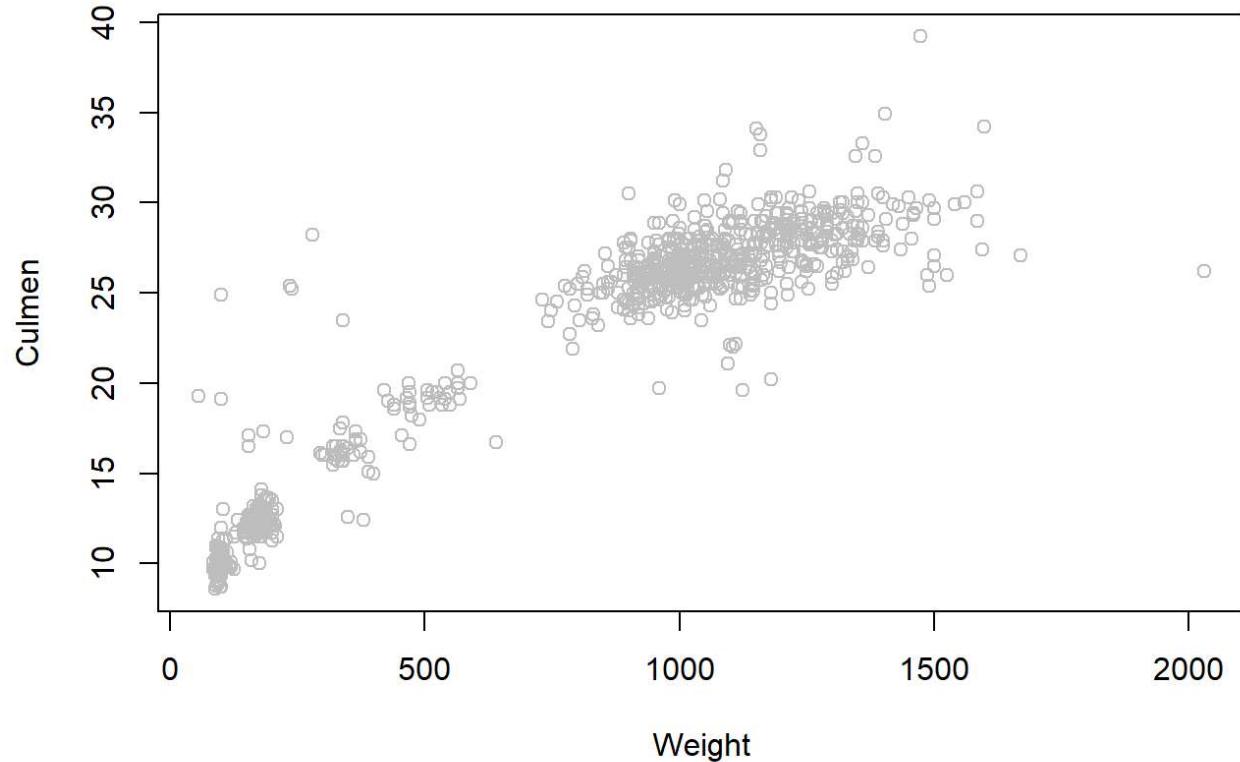
```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data[c(1,3)], col = "grey")
polygon(Hawks_data[res10$order,])
```



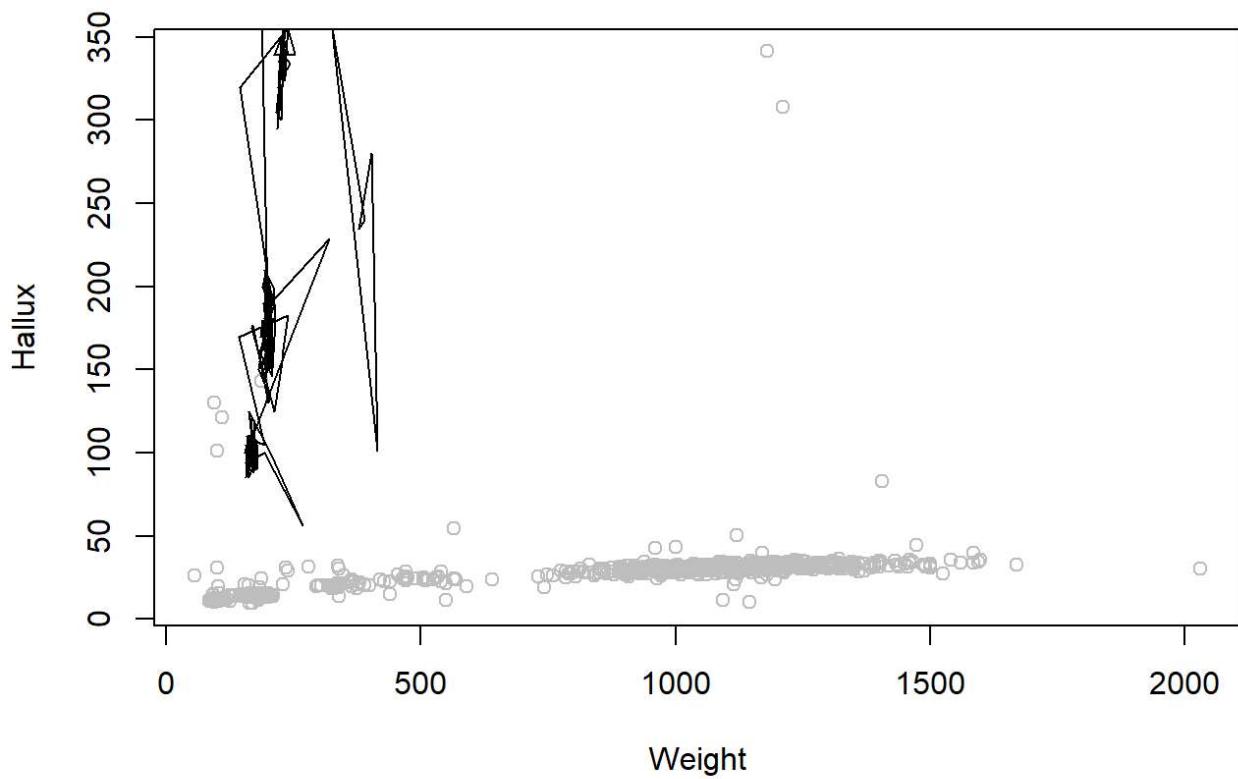
```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data[c(1,4)], col = "grey")
polygon(Hawks_data[res10$order,])
```



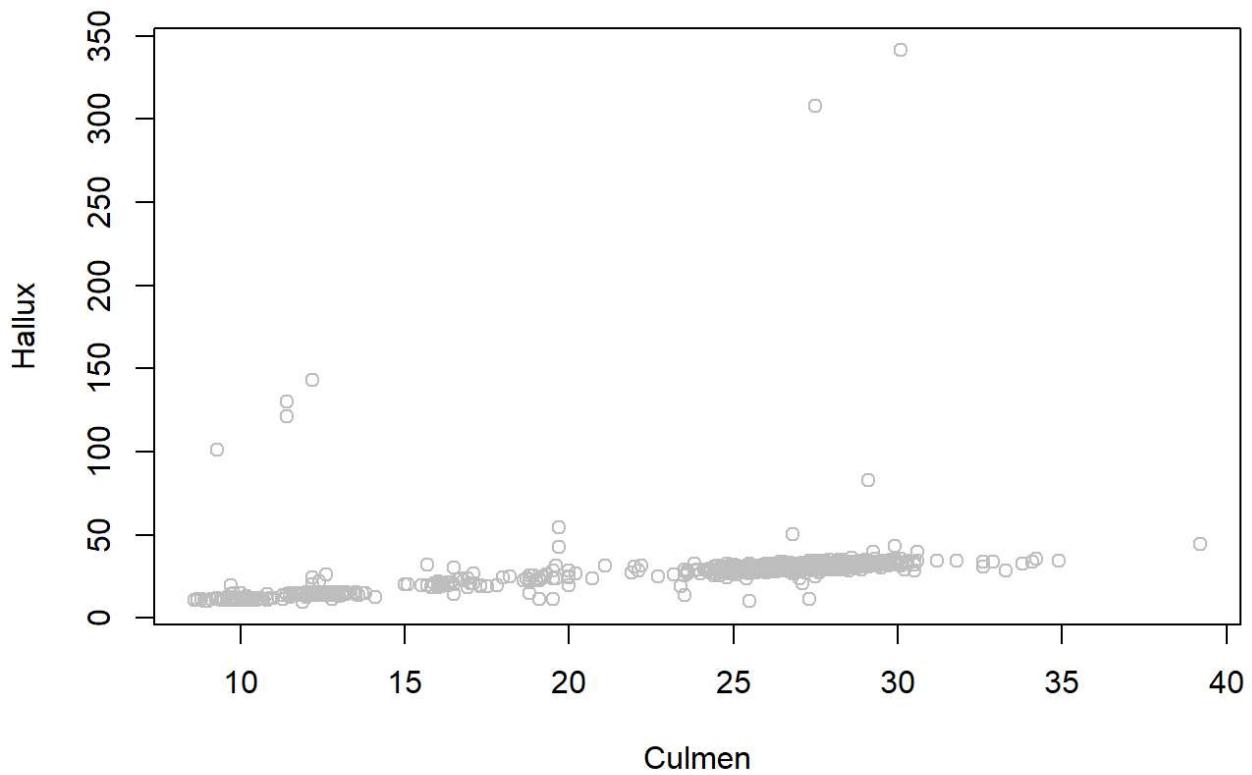
```
### Dibujo de las trazas que relacionan puntos (res10)
plot(Hawks_data[c(2,3)], col = "grey")
polygon(Hawks_data[res10$order,])
```



```
### Dibujo de las trazas que relacionan puntos (res10)
plot(Hawks_data[c(2,4)], col = "grey")
polygon(Hawks_data[res10$order,])
```

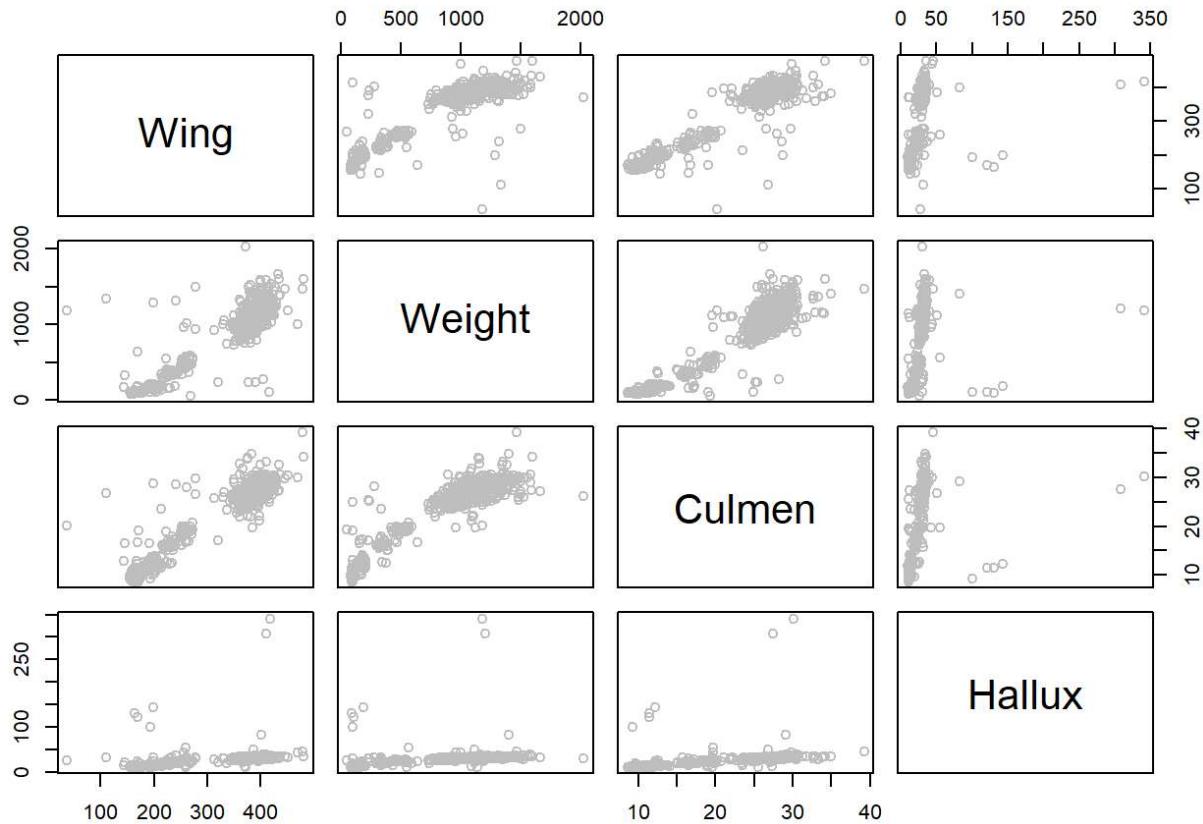


```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data[c(3,4)], col = "grey")
polygon(Hawks_data[res10$order,])
```



La relación de los puntos de manera general es la siguiente:

```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data, col = "grey")
polygon(Hawks_data[res10$order,])
```



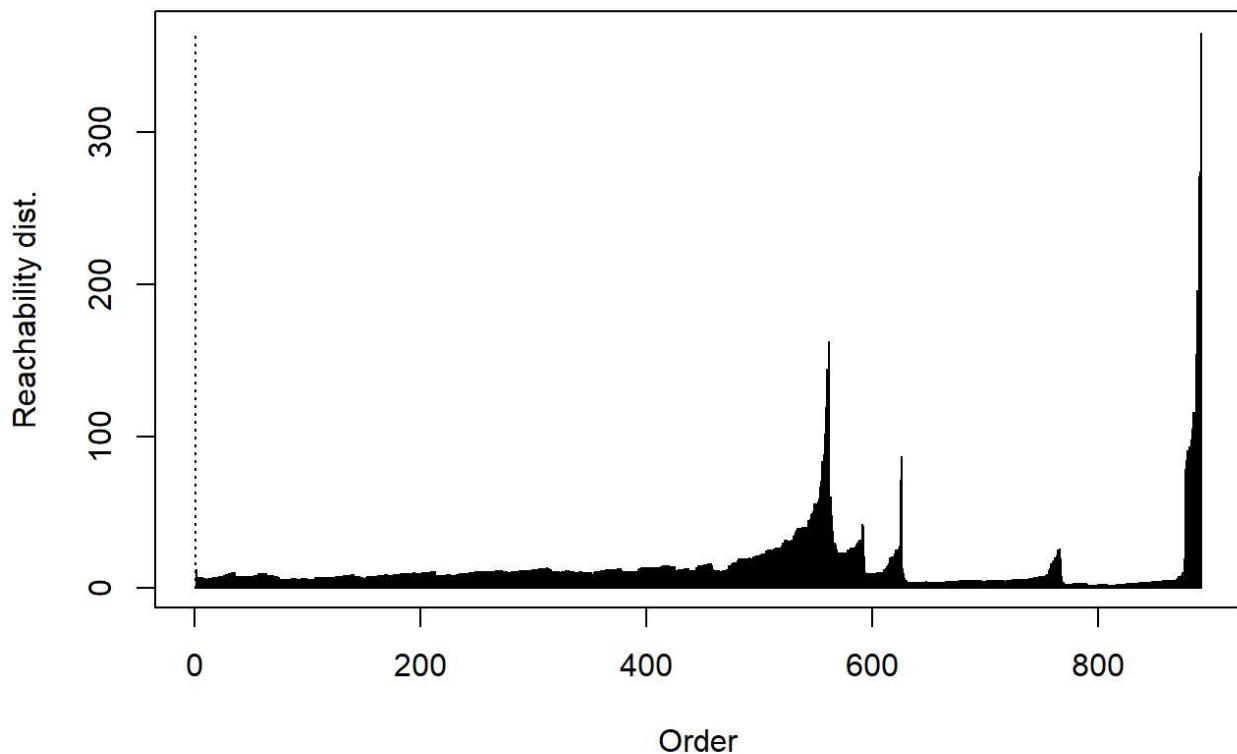
Ahora vamos a probar la alcanzabilidad probando distintos valores de epsilon (0,065).

```
### Extracción de un clustering DBSCAN cortando La alcanzabilidad en el valor eps_cl y
con res10
res <- extractDBSCAN(res10, eps_cl = .065)
res
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 10, eps = 531.584706326283, eps_cl = 0.065, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, c
luster
```

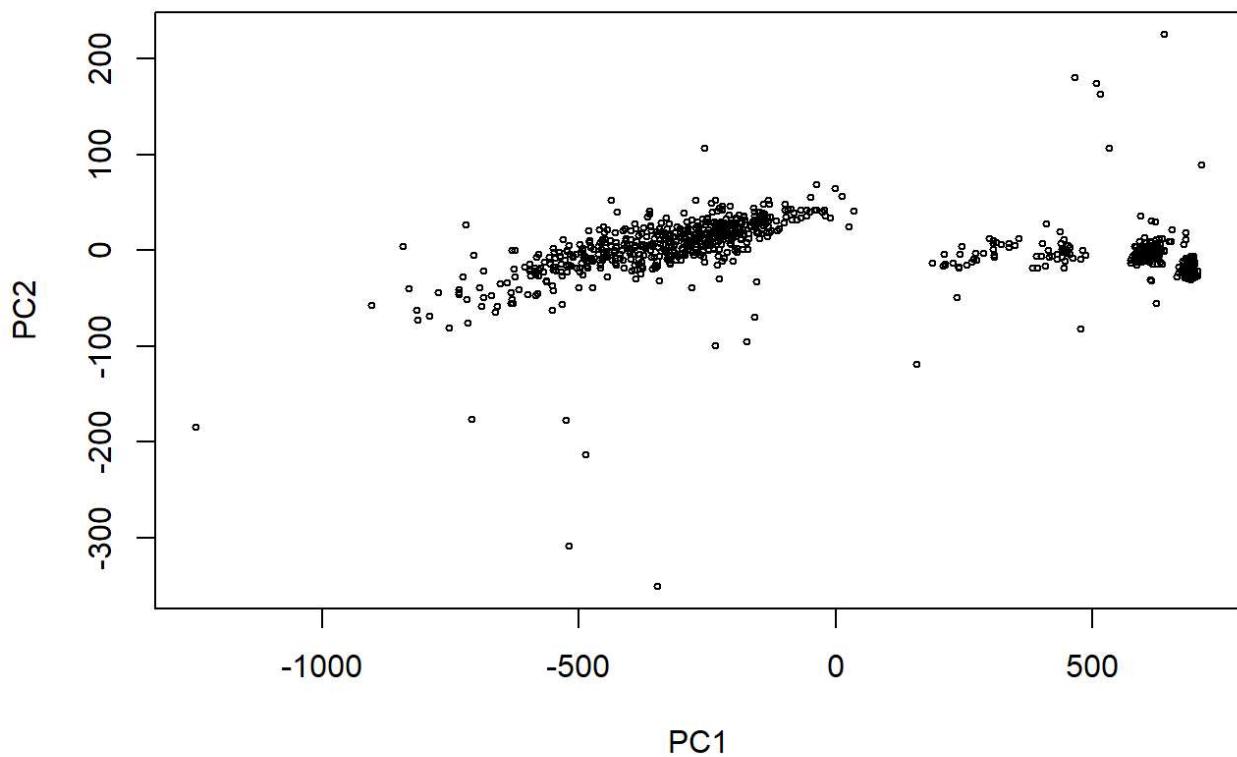
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls

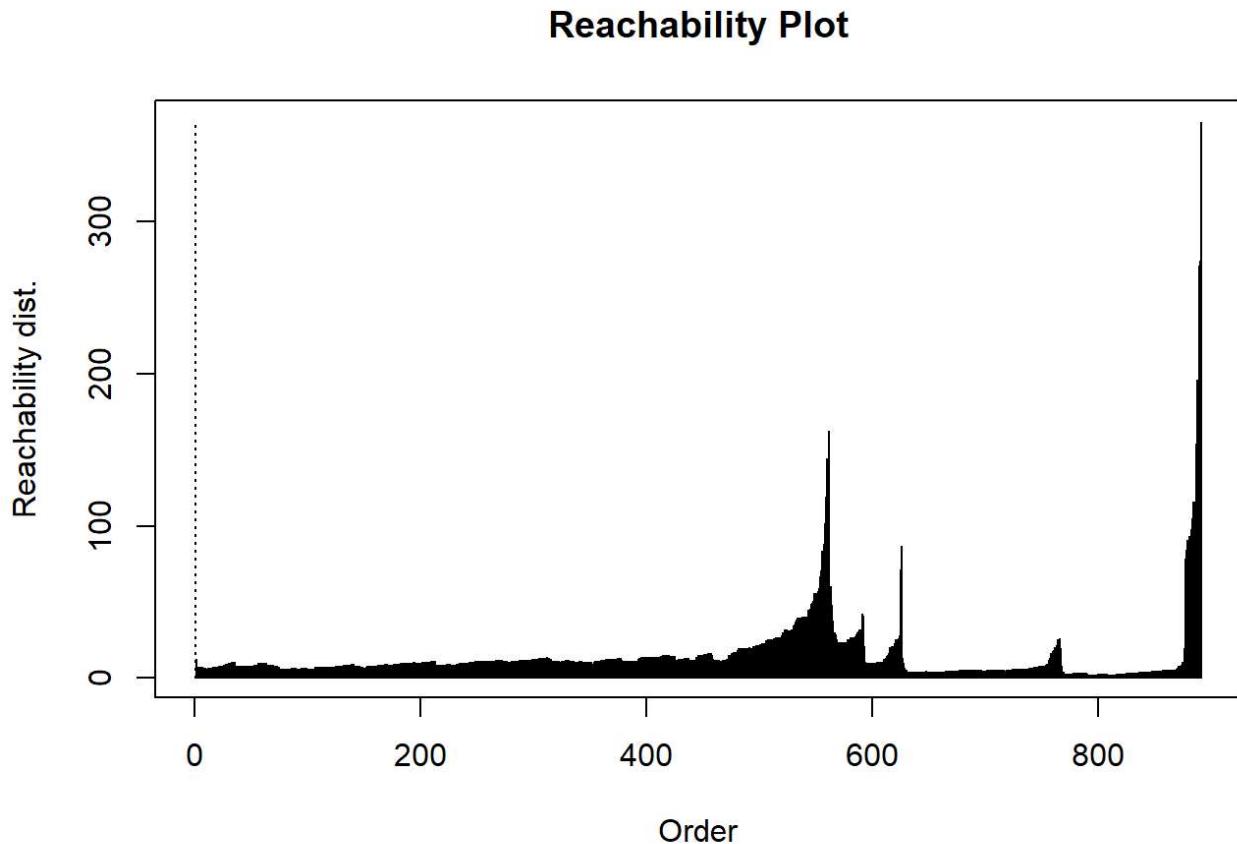


Repetimos el experimento anterior incrementando el parámetro *eps_cl*

```
### Incrementamos el parámetro eps
res <- extractDBSCAN(res10, eps_cl = .1)
res
```

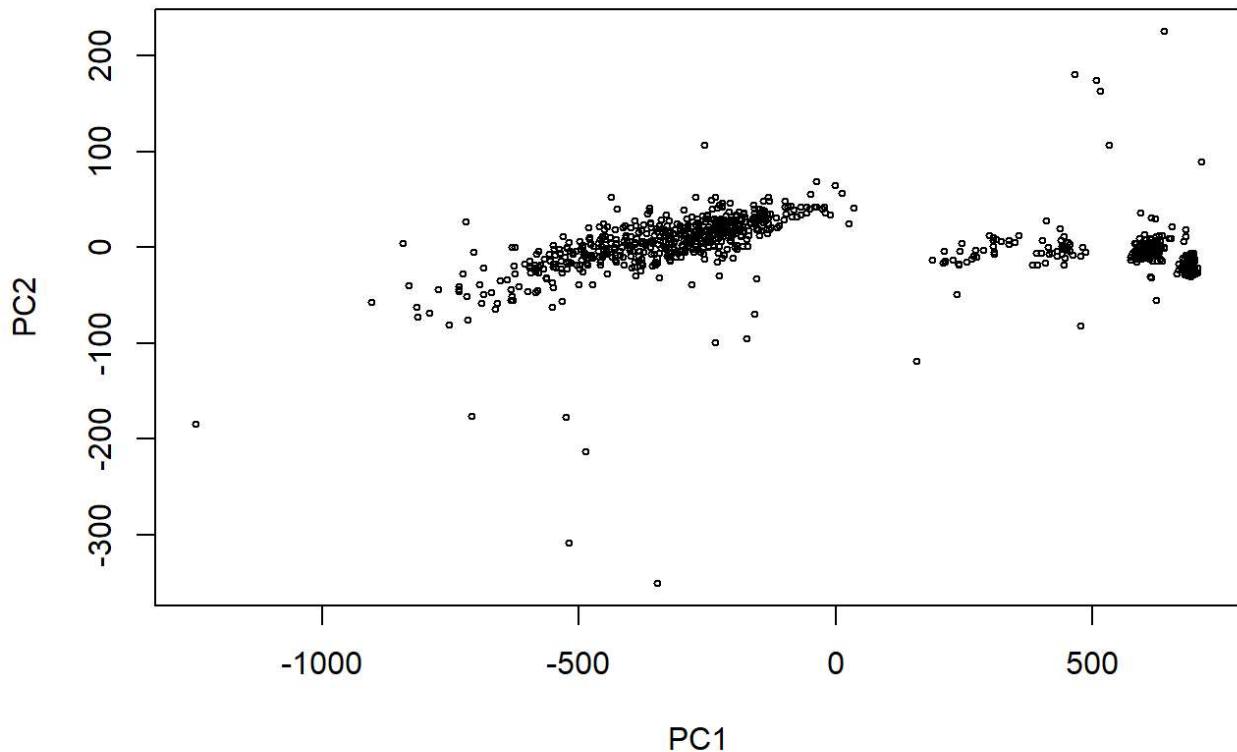
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 10, eps = 531.584706326283, eps_cl = 0.1, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster
```

```
plot(res)
```



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls



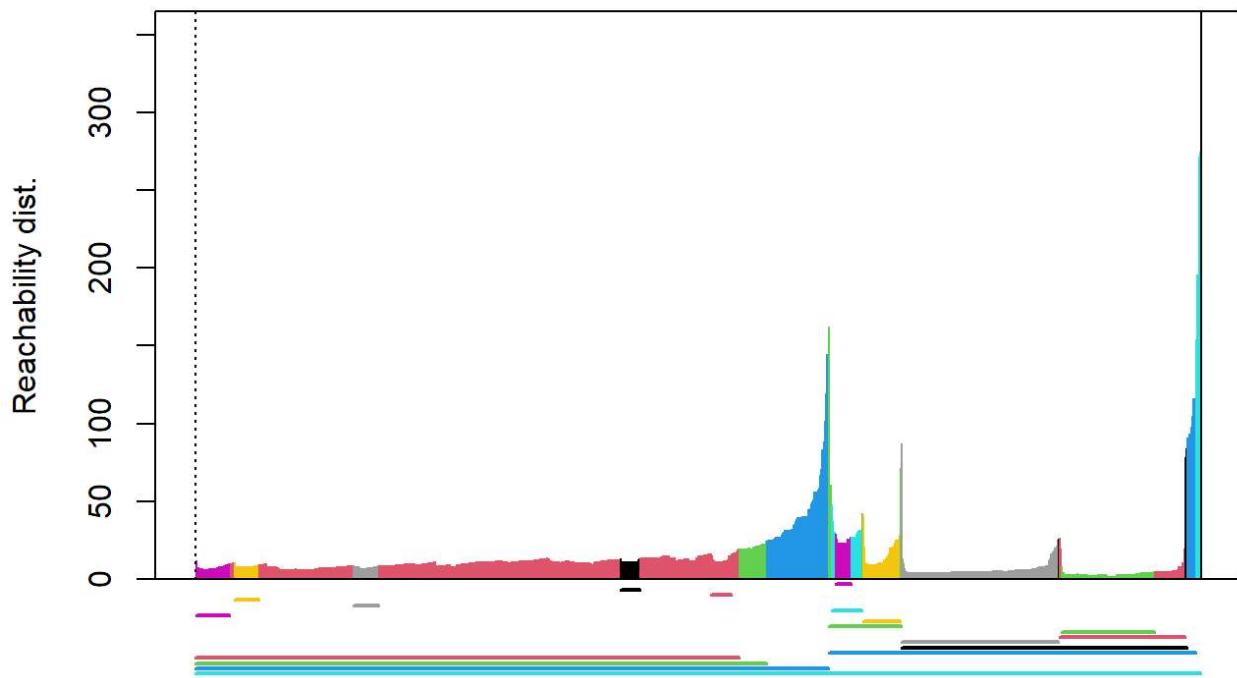
Veamos ahora una variante de la extracción **DBSCN** anterior. En ella el parámetro ξ nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

```
### Extracción del clustering jerárquico en función de La variación de La densidad por el método xi
res <- extractXi(res10, xi = 0.05)
res
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 10, eps = 531.584706326283, eps_cl = NA, xi = 0.05
## The clustering contains 18 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

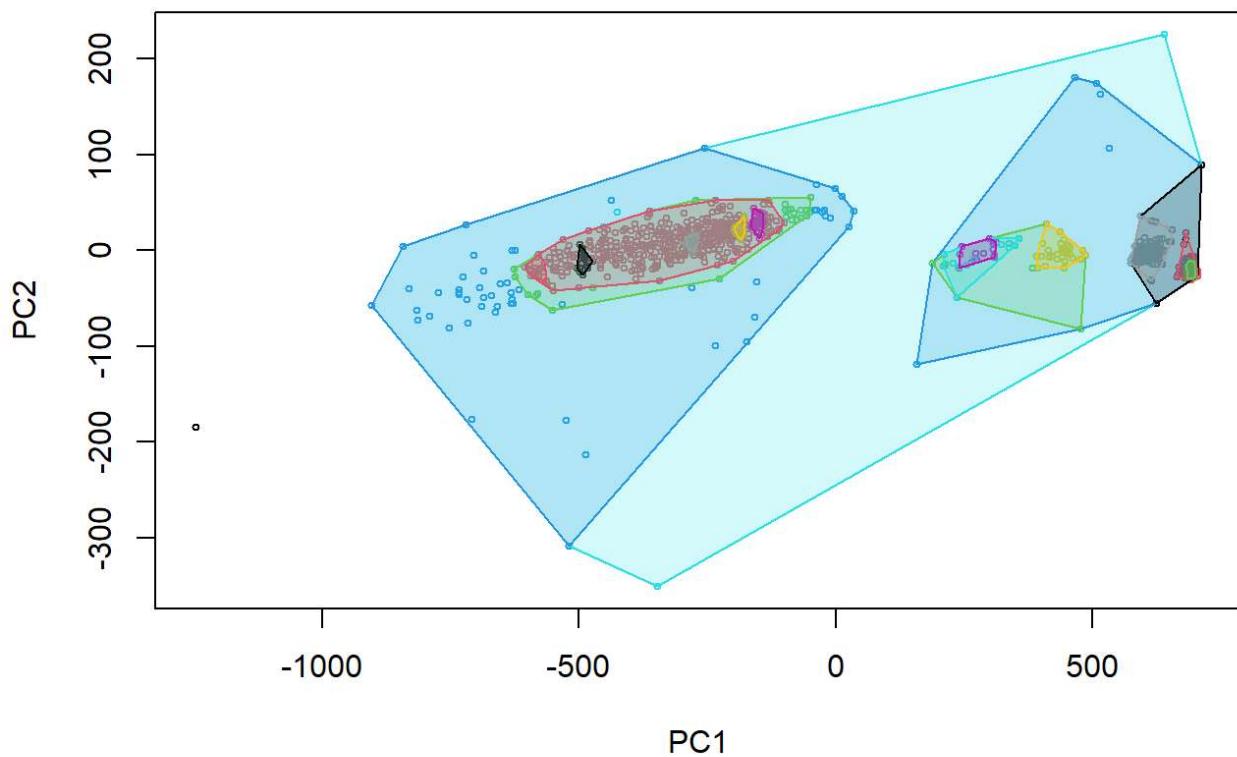
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls

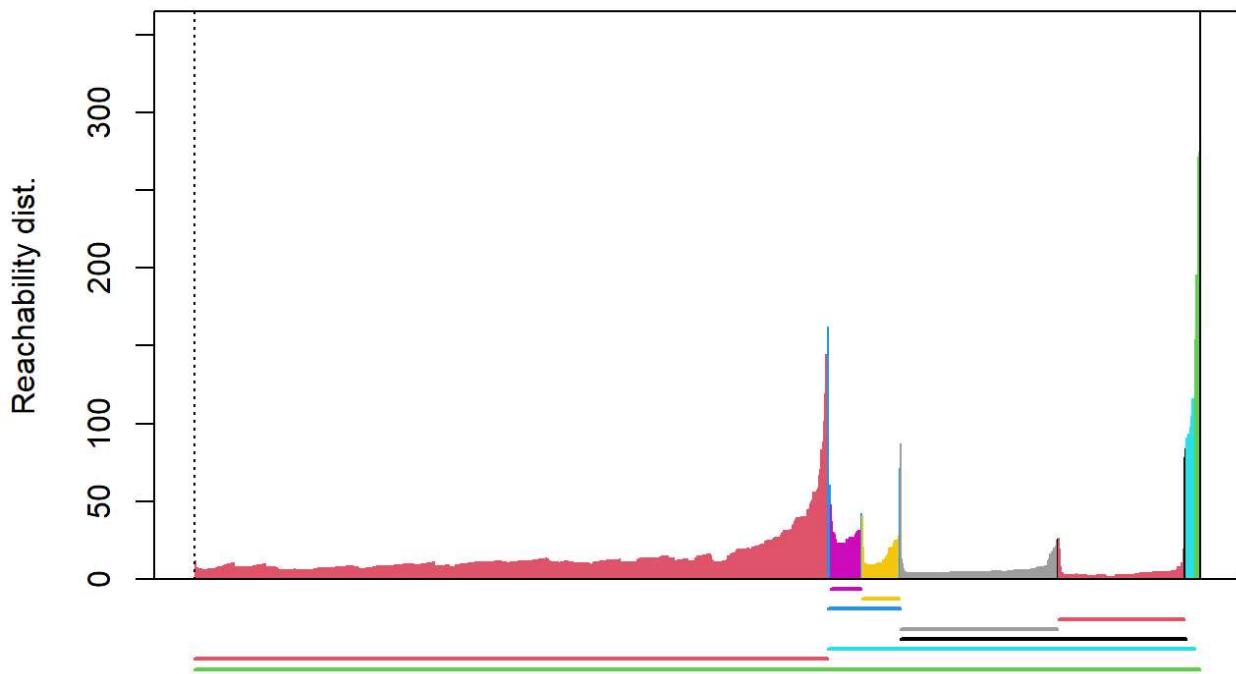


```
### Extracción del clustering jerárquico en función de La variación de La densidad por el método xi
res <- extractXi(res10, xi = 0.1)
res
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 10, eps = 531.584706326283, eps_cl = NA, xi = 0.1
## The clustering contains 9 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

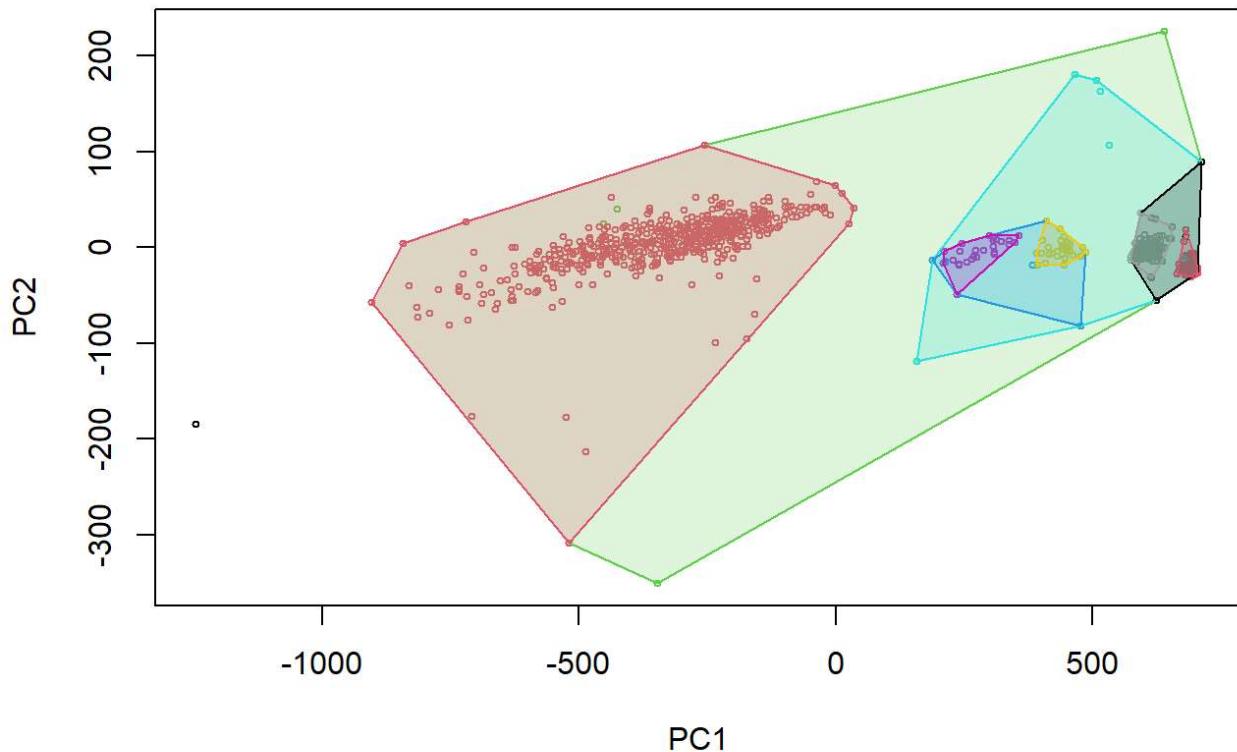
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls



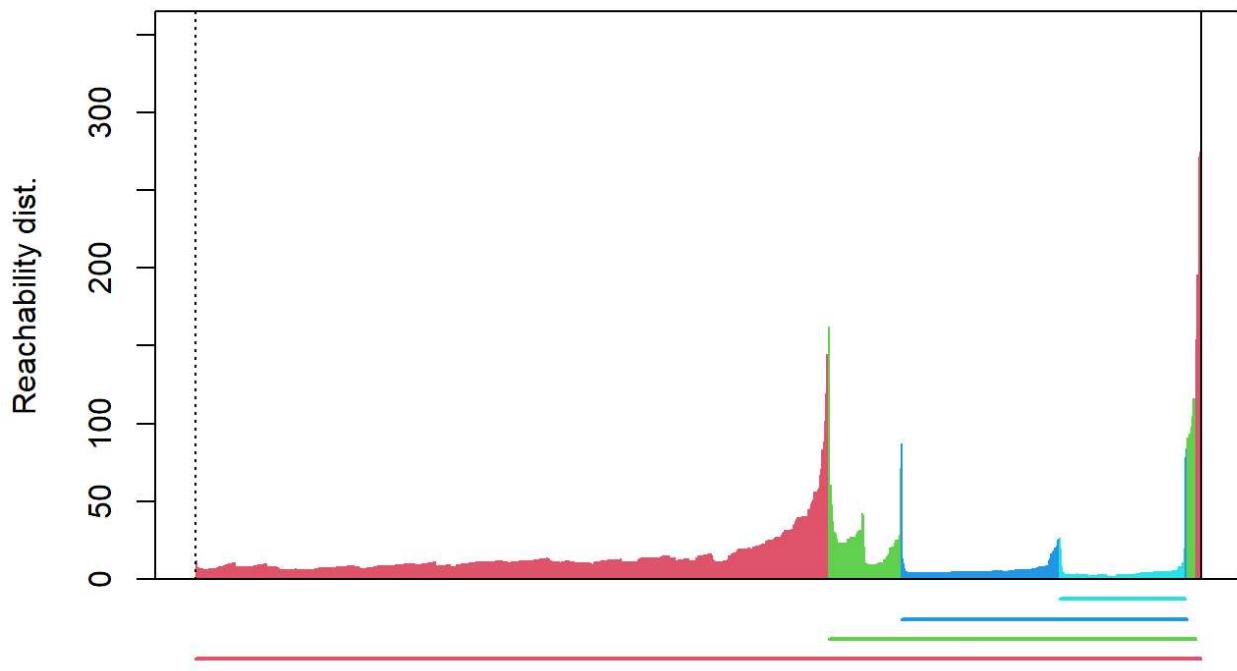
```
### Extracción del clustering jerárquico en función de La variación de La densidad por e  
l método xi
```

```
res <- extractXi(res10, xi = 0.2)  
res
```

```
## OPTICS ordering/clustering for 891 objects.  
## Parameters: minPts = 10, eps = 531.584706326283, eps_cl = NA, xi = 0.2  
## The clustering contains 4 cluster(s) and 1 noise points.  
##  
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, c  
lusters_xi, cluster
```

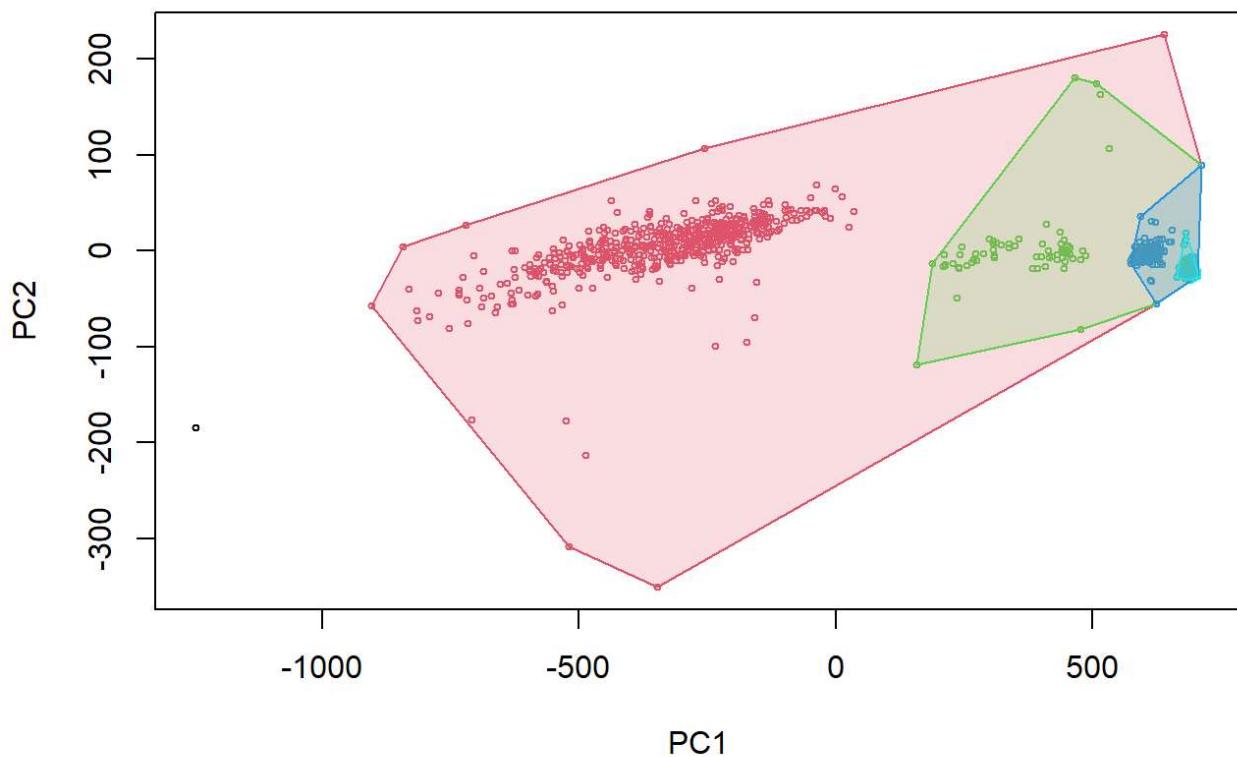
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

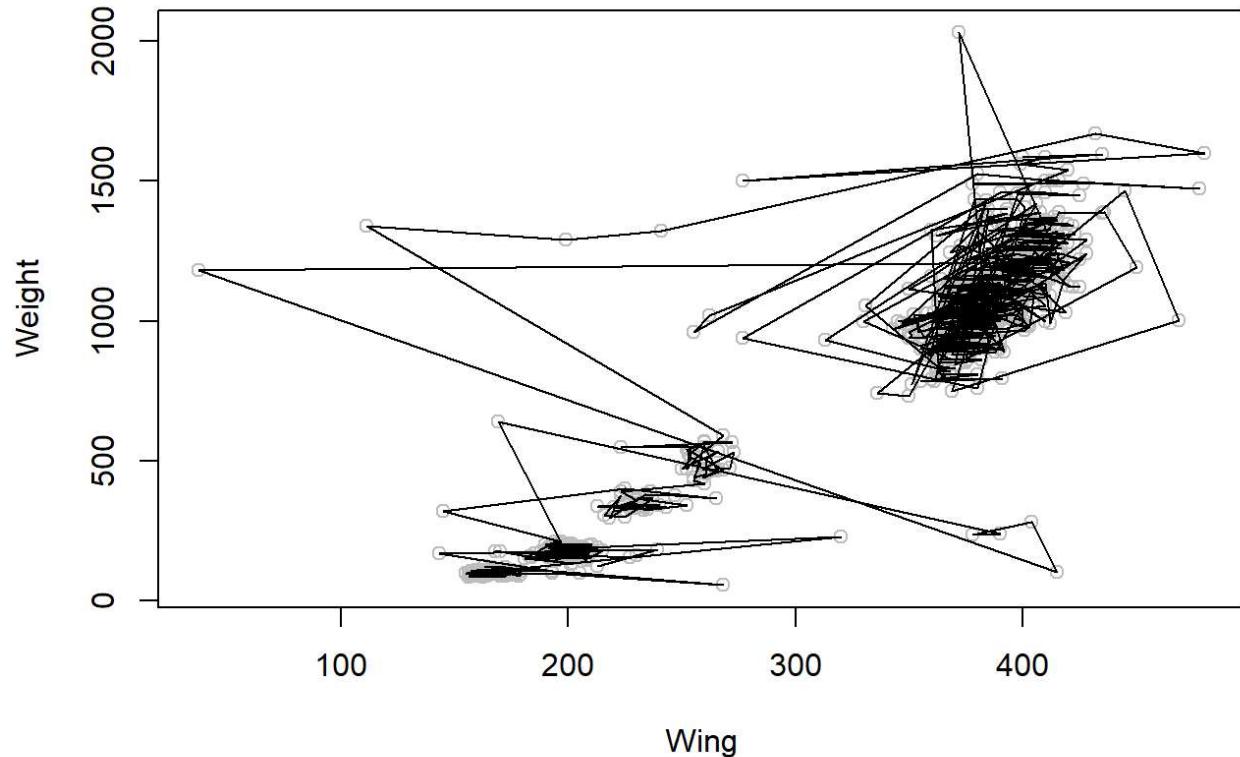
Convex Cluster Hulls



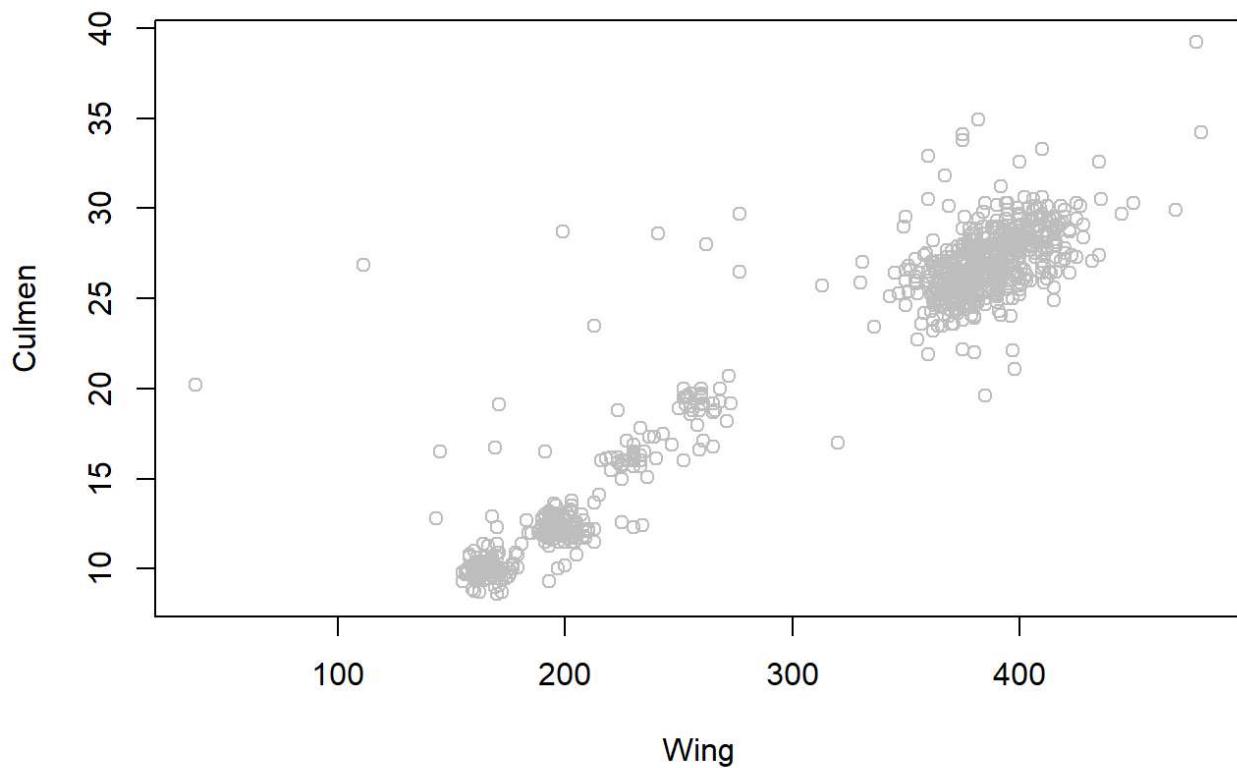
Como se puede observar, con los distintos valores de x_i el numero de clúster que obtenemos es distintitos, mientras más grande es la distancia menos clústeres necesitan para clasificar los valores.

Ahora se va a realizar el mismo análisis con un análisis de vecindad de 20.

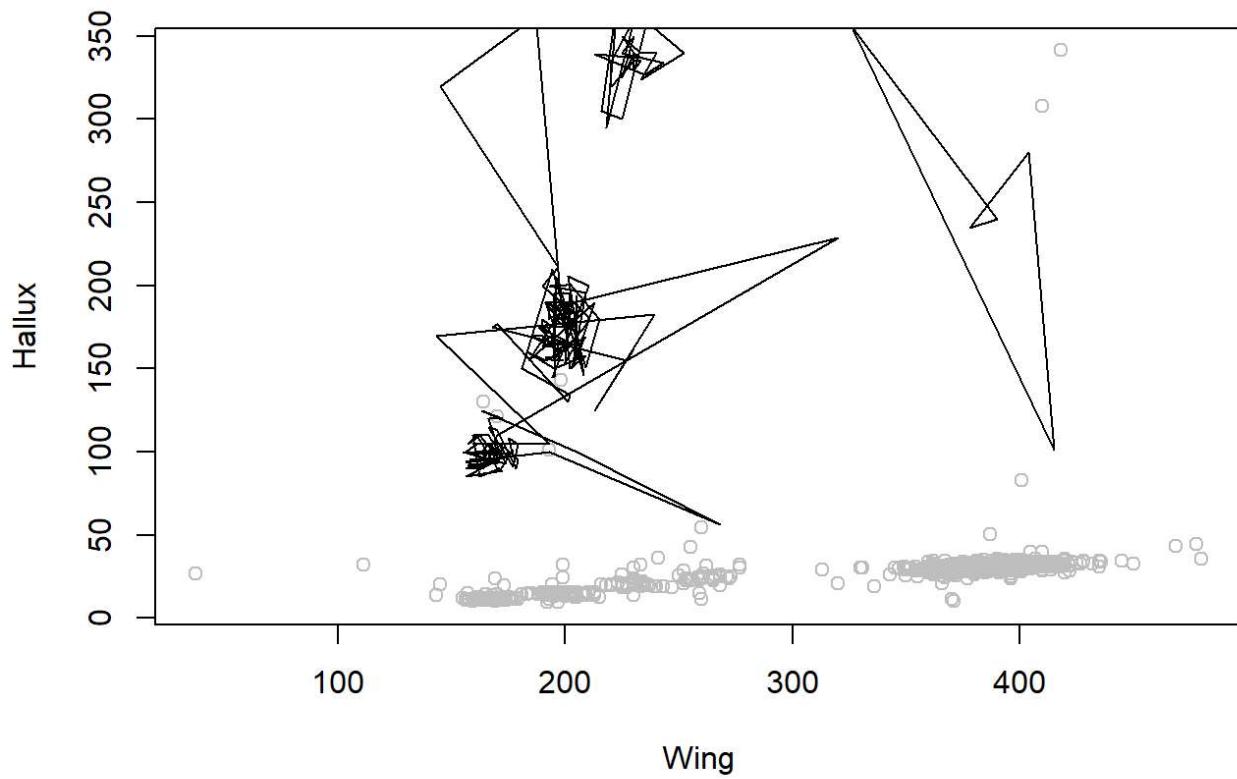
```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data[c(1,2)], col = "grey")
polygon(Hawks_data[res20$order,])
```



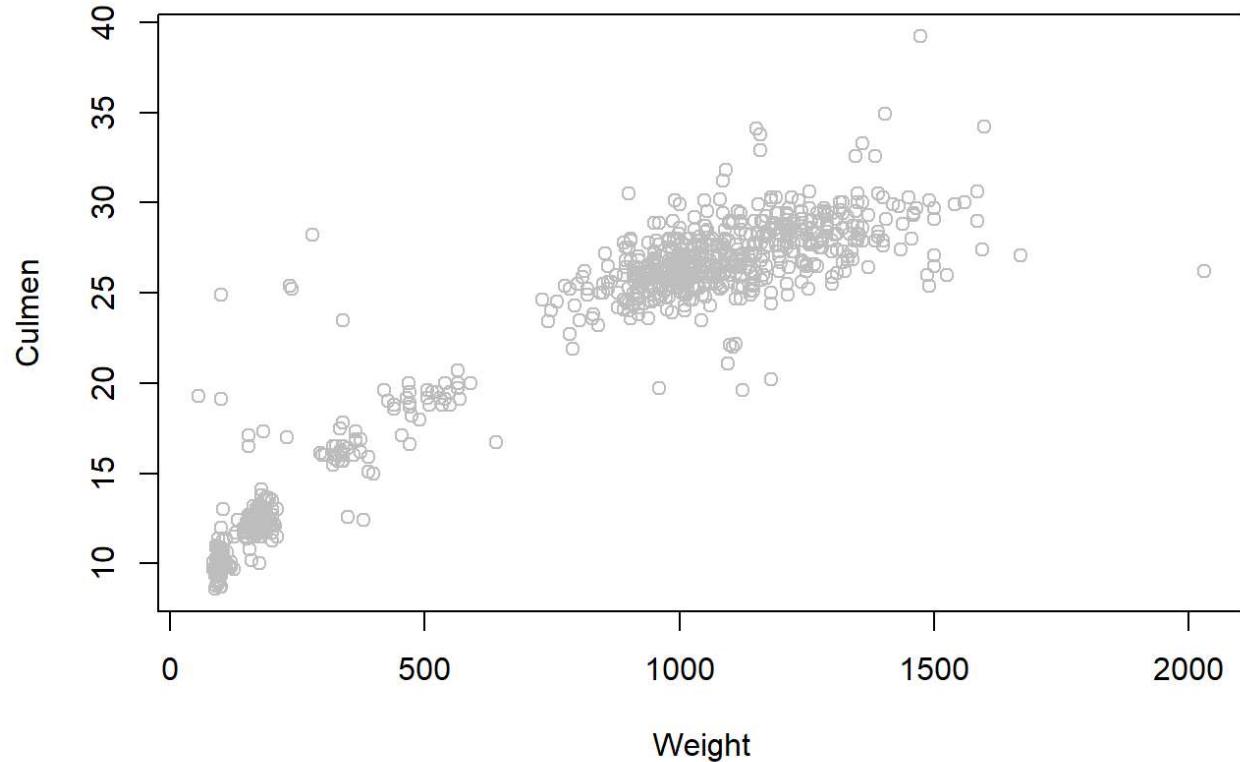
```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data[c(1,3)], col = "grey")
polygon(Hawks_data[res20$order,])
```



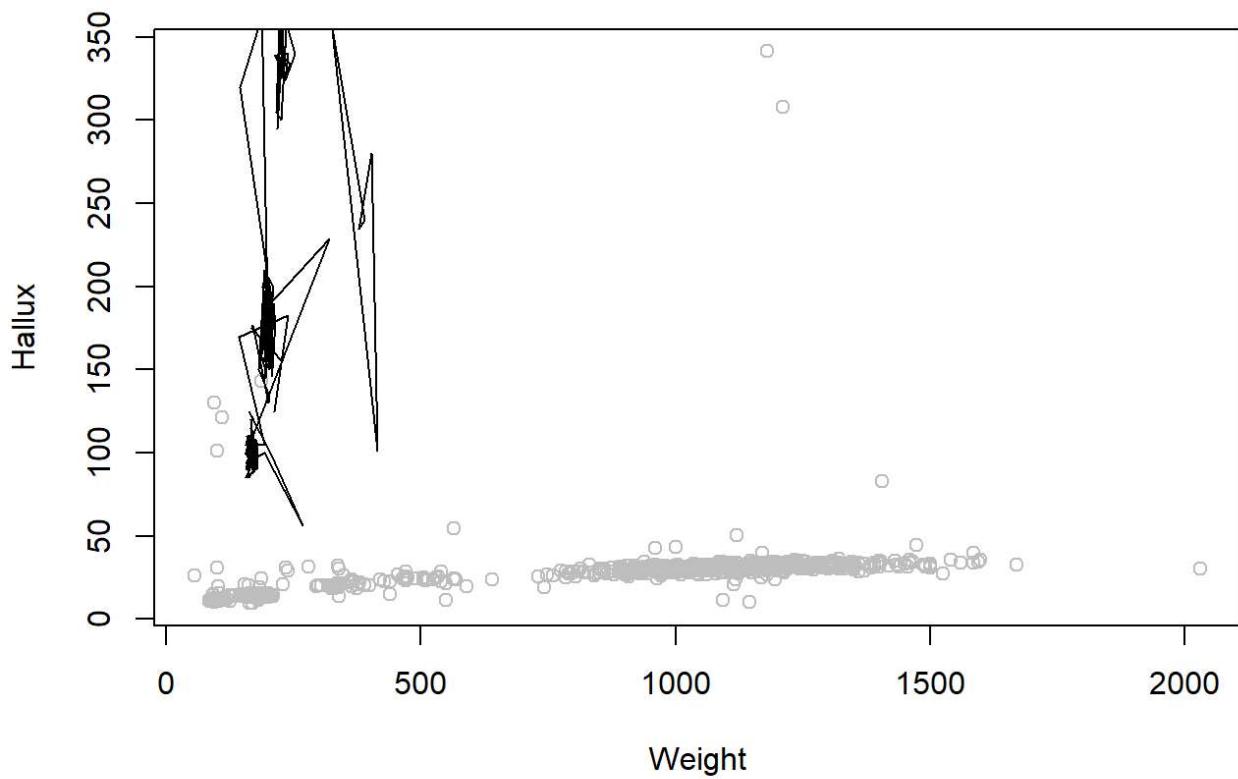
```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data[c(1,4)], col = "grey")
polygon(Hawks_data[res20$order,])
```



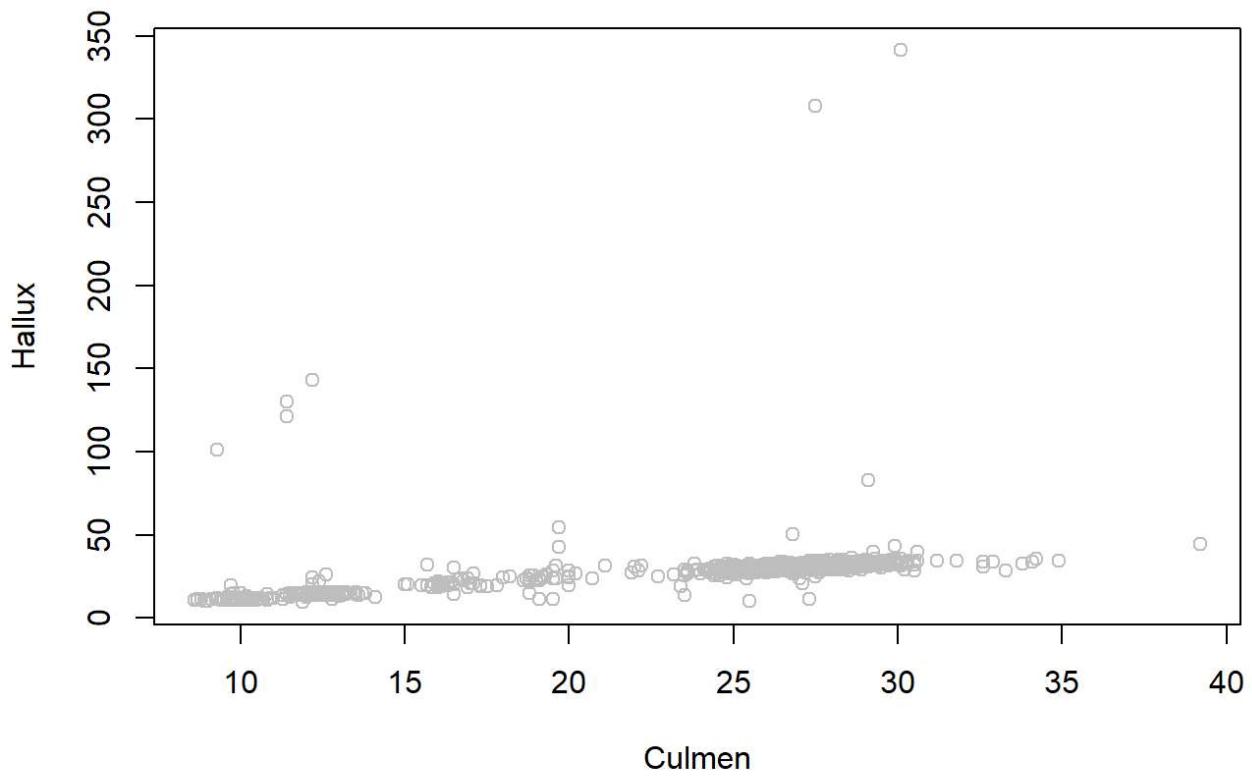
```
### Dibujo de las trazas que relacionan puntos (res20)
plot(Hawks_data[c(2,3)], col = "grey")
polygon(Hawks_data[res20$order,])
```



```
### Dibujo de las trazas que relacionan puntos (res20)
plot(Hawks_data[c(2,4)], col = "grey")
polygon(Hawks_data[res20$order,])
```

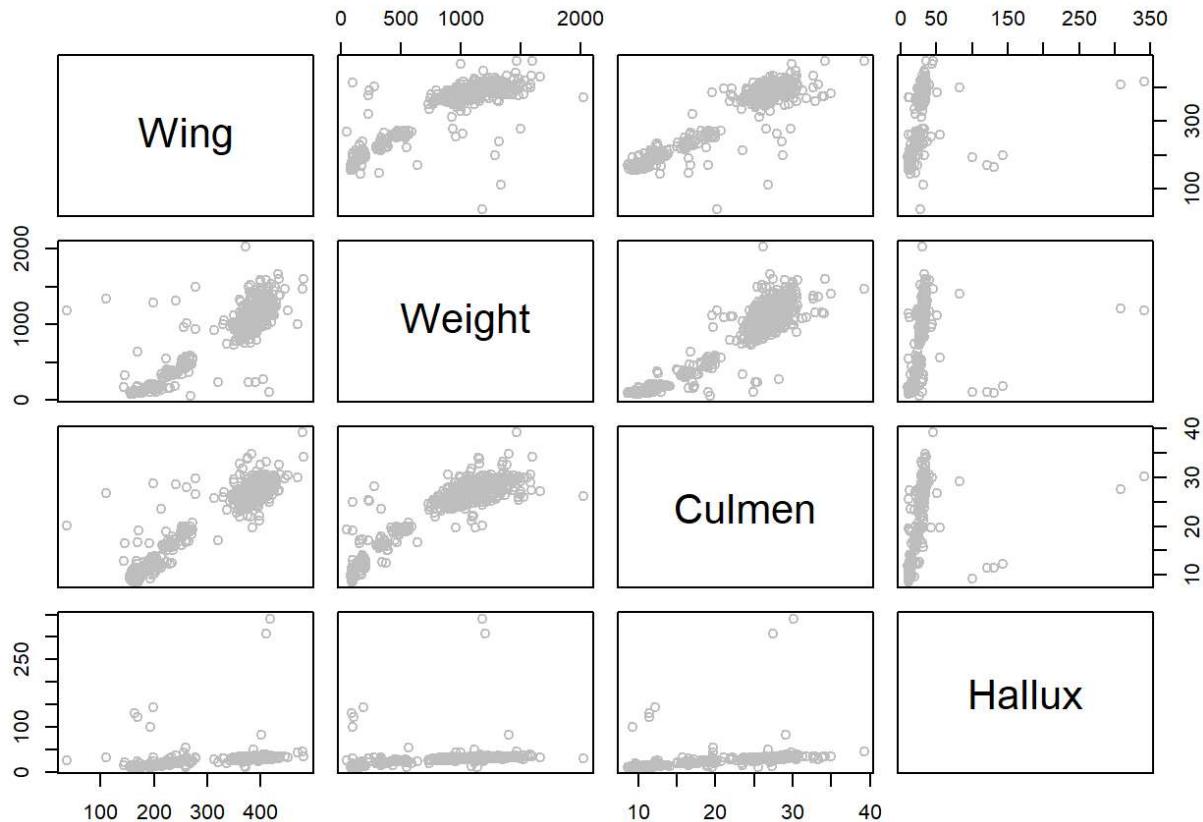


```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data[c(3,4)], col = "grey")
polygon(Hawks_data[res20$order,])
```



La relación de los puntos de manera general es la siguiente:

```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data, col = "grey")
polygon(Hawks_data[res20$order,])
```



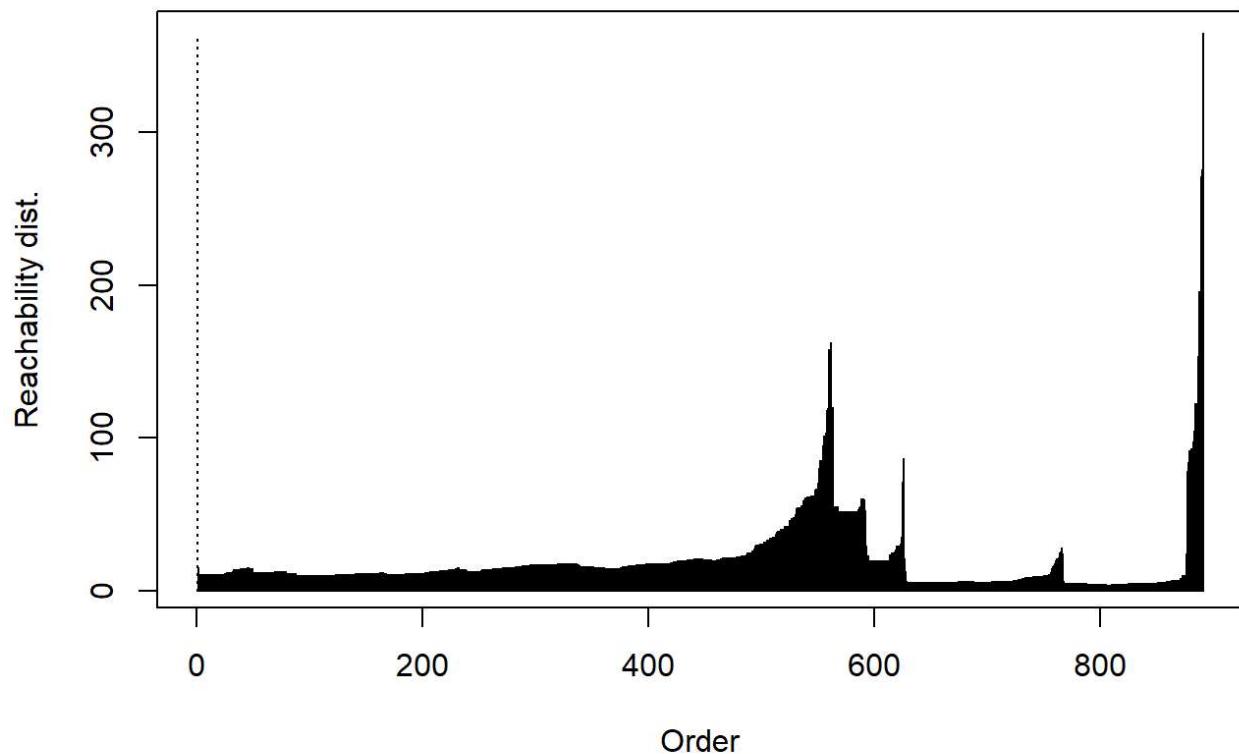
Ahora vamos a probar la alcanzabilidad probando distintos valores de epsilon.

```
### Extracción de un clustering DBSCAN cortando la alcanzabilidad en el valor eps_cl y
con res20
res <- extractDBSCAN(res20, eps_cl = .065)
res
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 20, eps = 575.591808489315, eps_cl = 0.065, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, c
luster
```

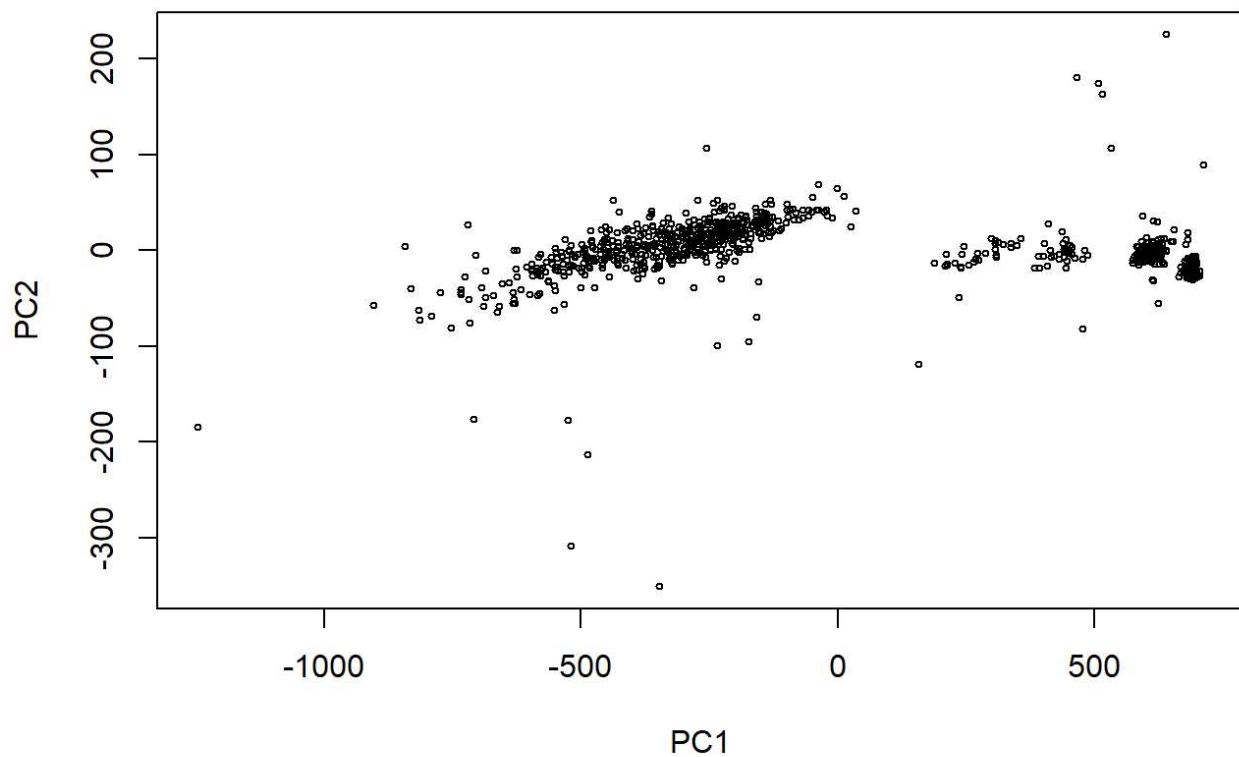
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls

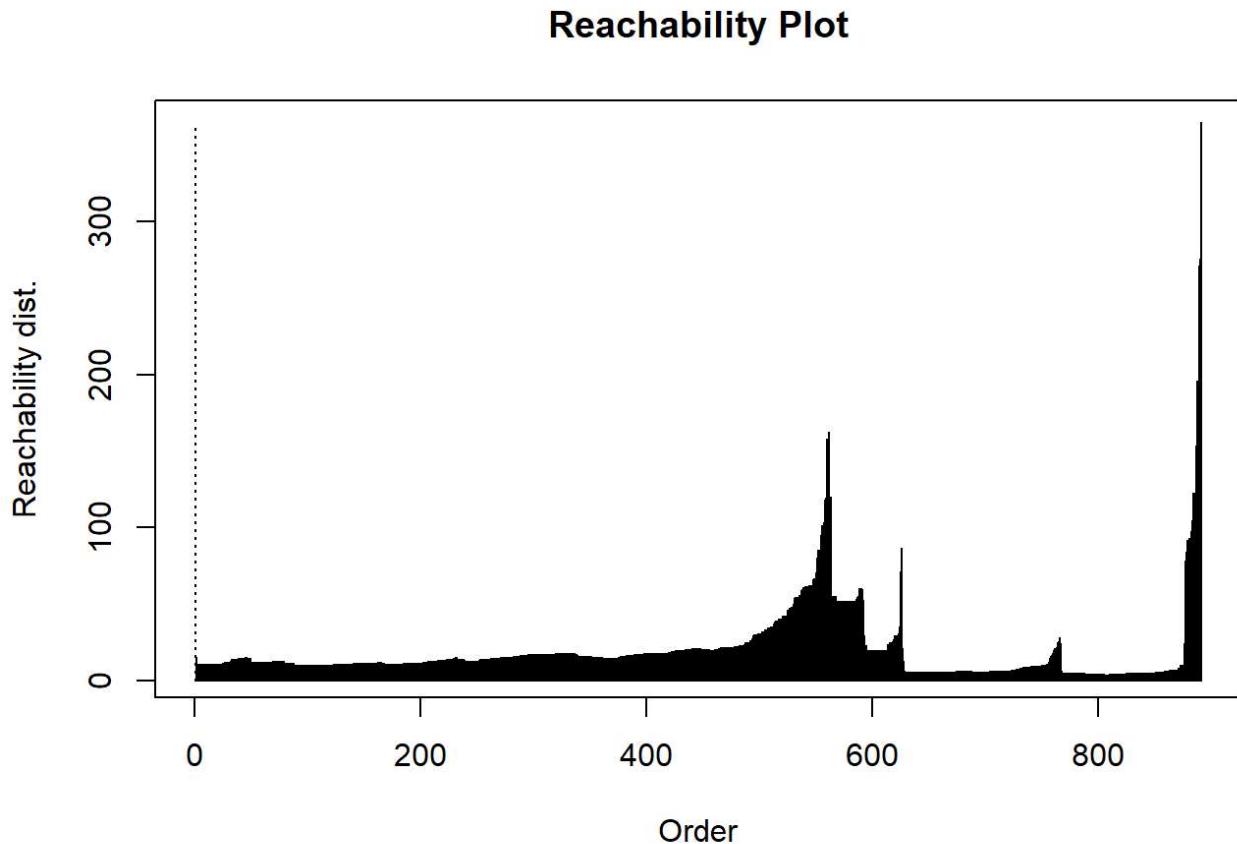


Repetimos el experimento anterior incrementando el parámetro *eps_cl*

```
### Incrementamos el parámetro eps
res <- extractDBSCAN(res20, eps_cl = .1)
res
```

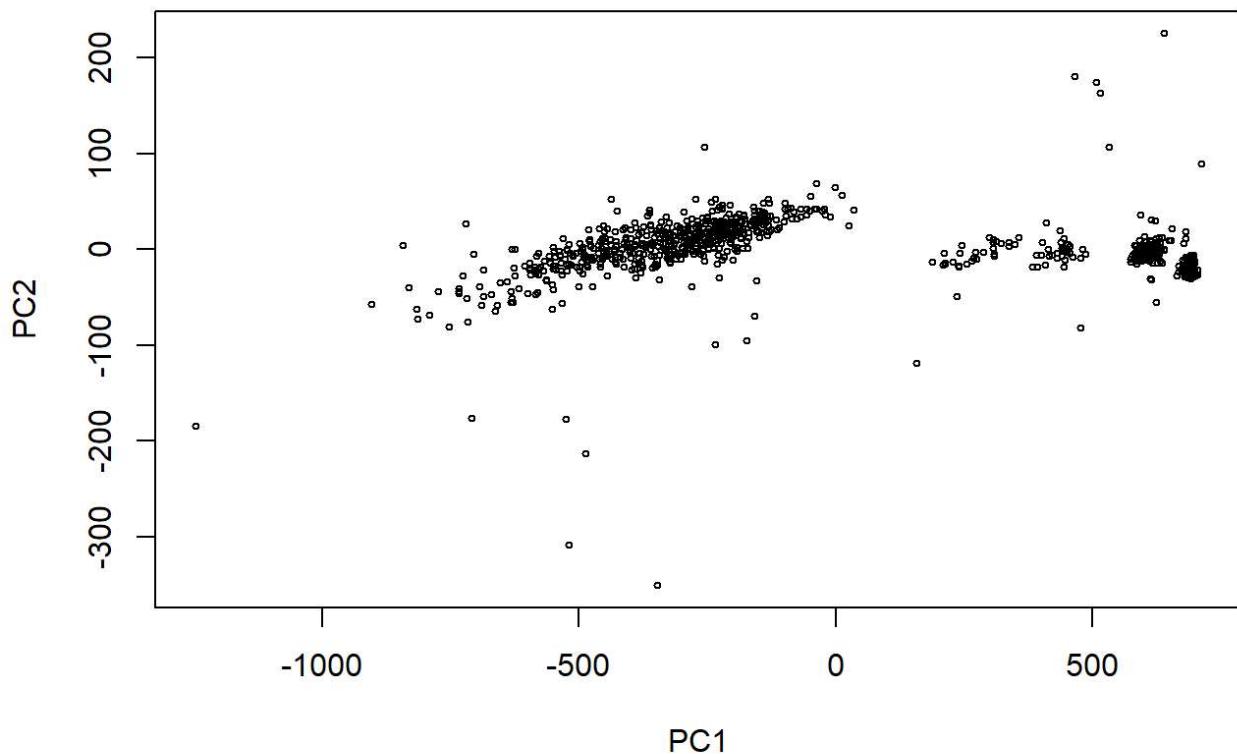
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 20, eps = 575.591808489315, eps_cl = 0.1, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster
```

```
plot(res)
```



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls



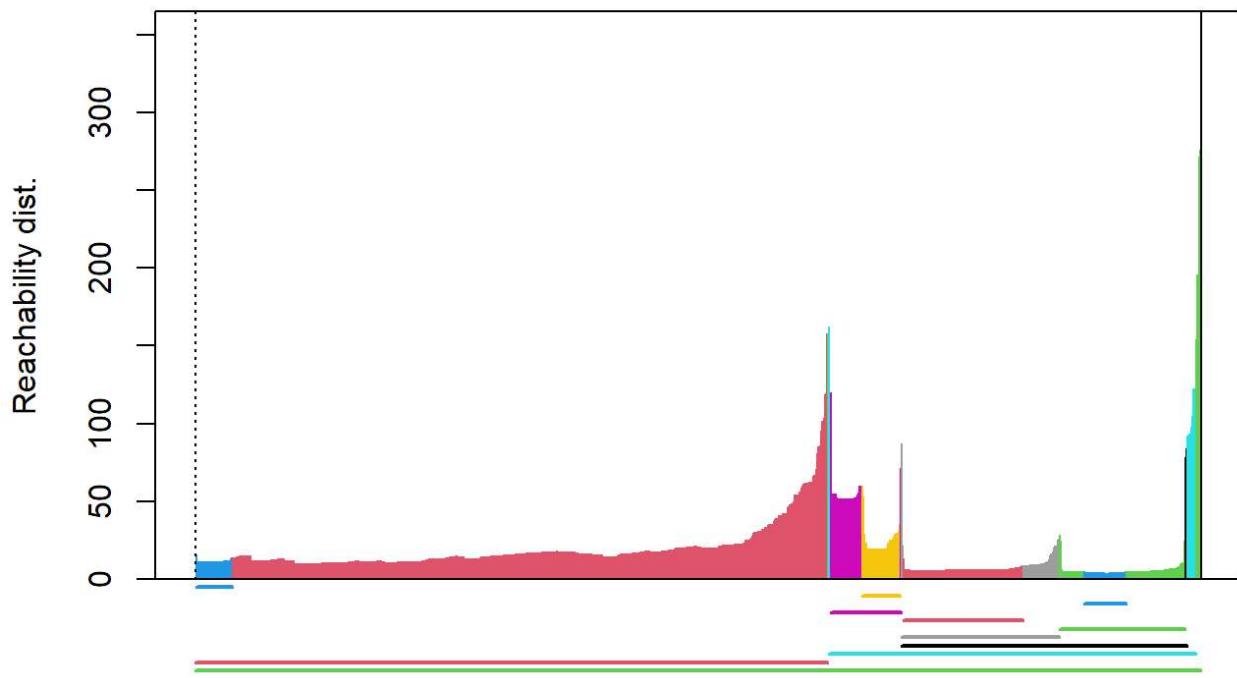
Veamos ahora una variante de la extracción **DBSCN** anterior. En ella el parámetro xi nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

```
### Extracción del clustering jerárquico en función de La variación de La densidad por el método xi
res <- extractXi(res20, xi = 0.05)
res
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 20, eps = 575.591808489315, eps_cl = NA, xi = 0.05
## The clustering contains 11 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

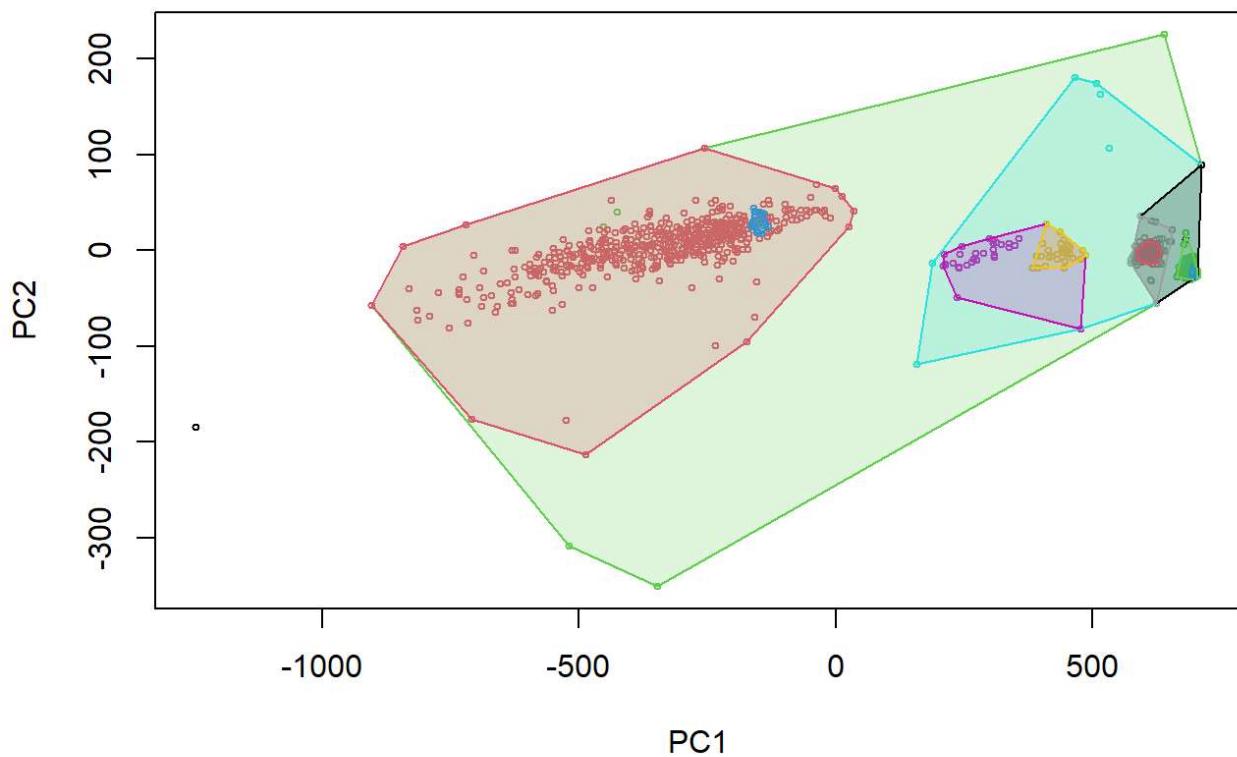
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

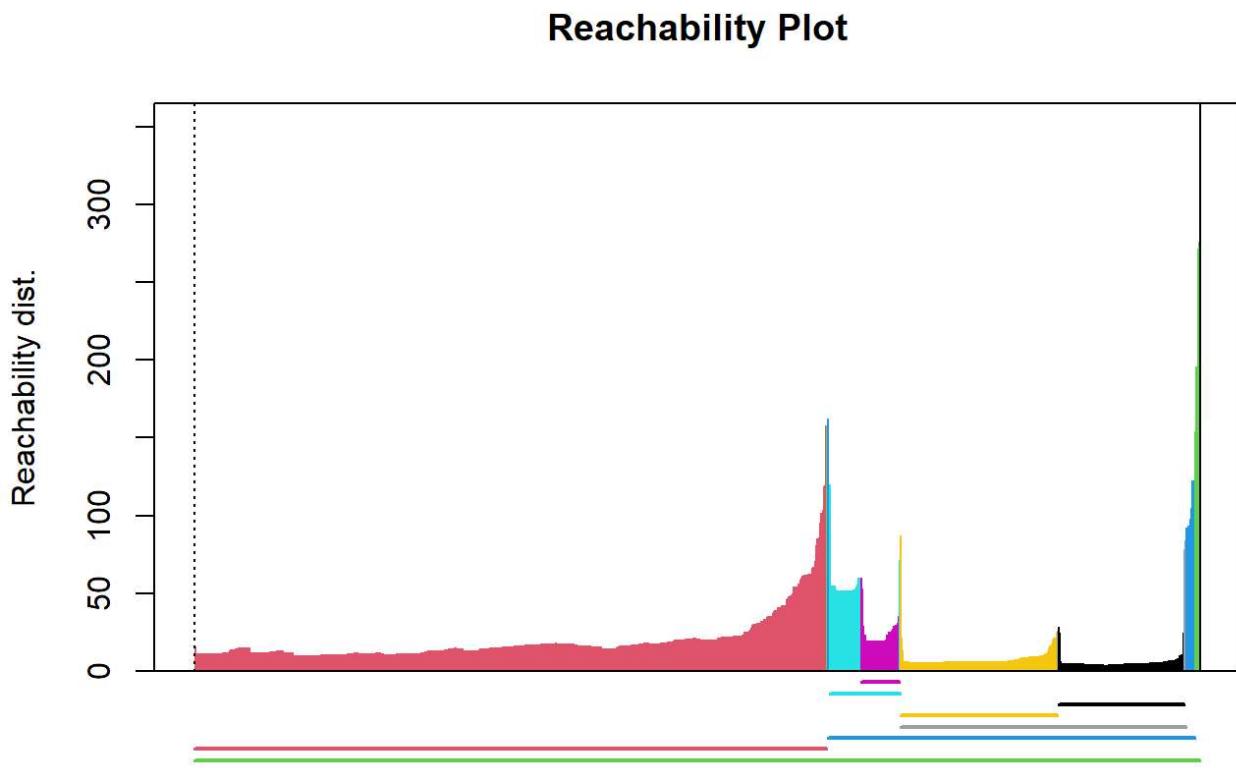
Convex Cluster Hulls



```
### Extracción del clustering jerárquico en función de La variación de La densidad por el método xi
res <- extractXi(res20, xi = 0.1)
res
```

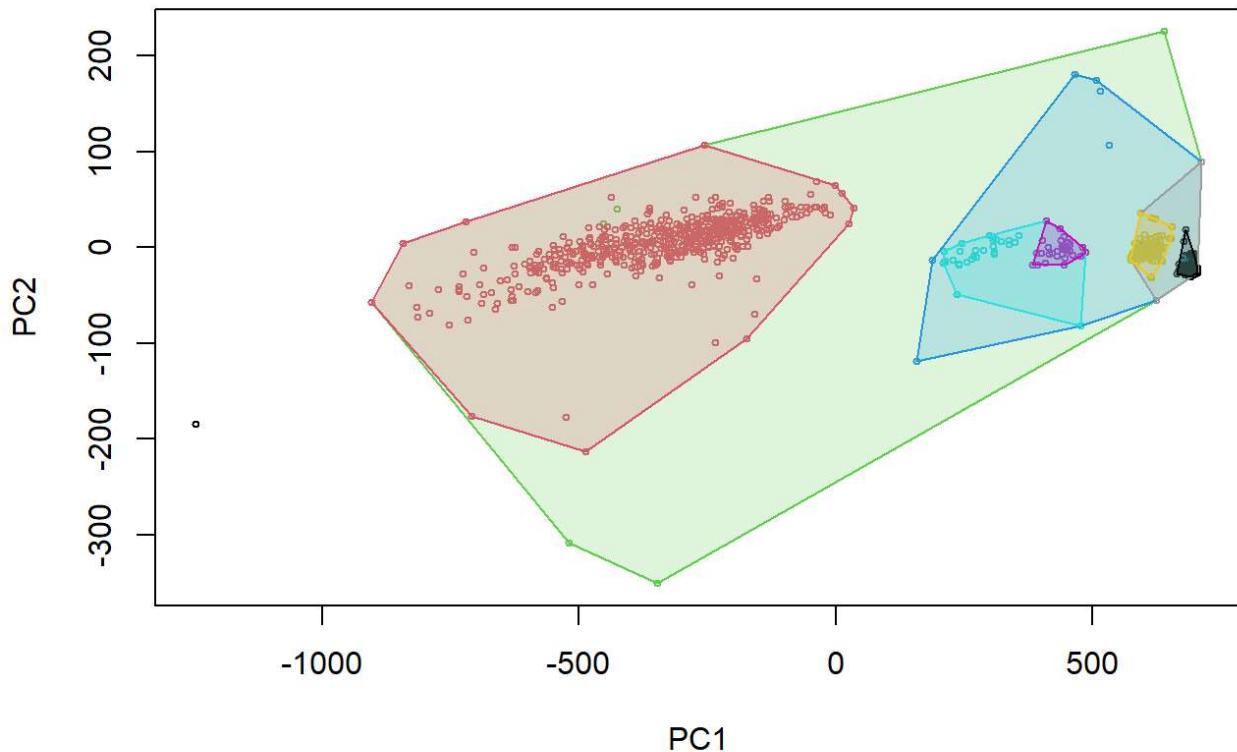
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 20, eps = 575.591808489315, eps_cl = NA, xi = 0.1
## The clustering contains 8 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

```
plot(res)
```



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls

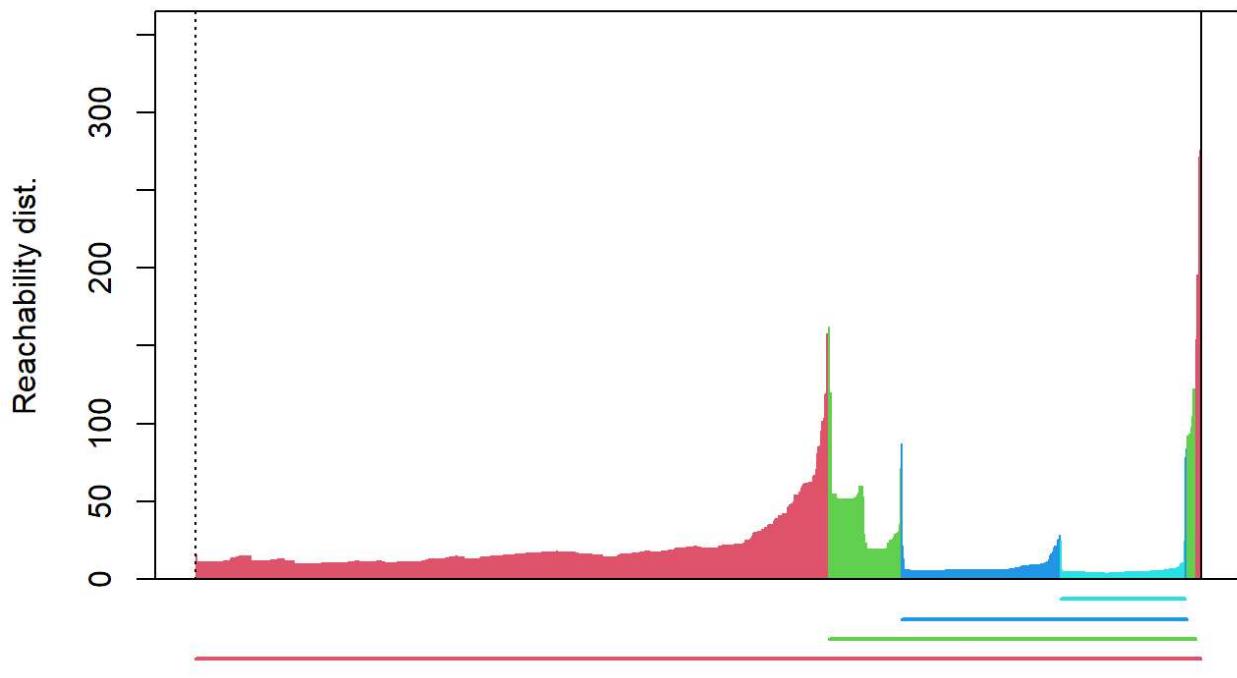


```
### Extracción del clustering jerárquico en función de La variación de La densidad por e  
l método xi  
res <- extractXi(res20, xi = 0.2)  
res
```

```
## OPTICS ordering/clustering for 891 objects.  
## Parameters: minPts = 20, eps = 575.591808489315, eps_cl = NA, xi = 0.2  
## The clustering contains 4 cluster(s) and 1 noise points.  
##  
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, c  
lusters_xi, cluster
```

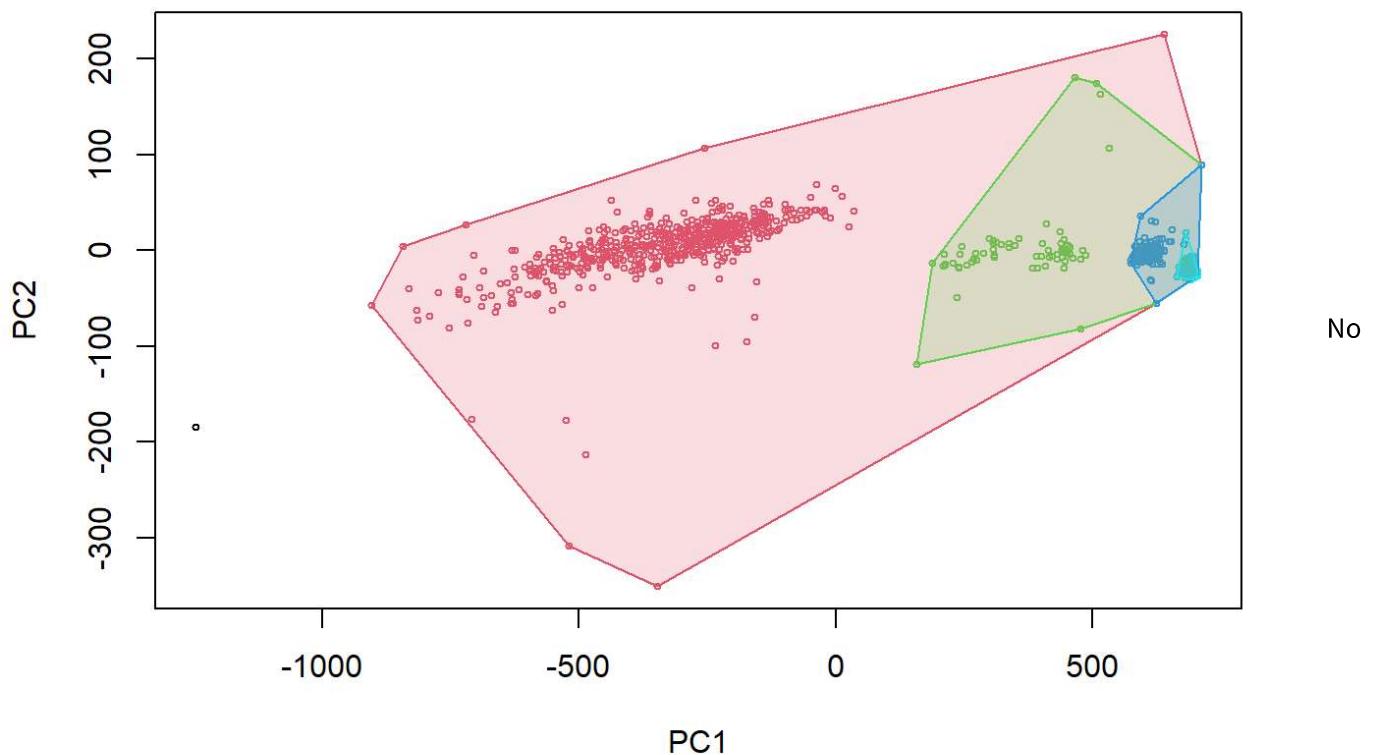
```
plot(res)
```

Reachability Plot



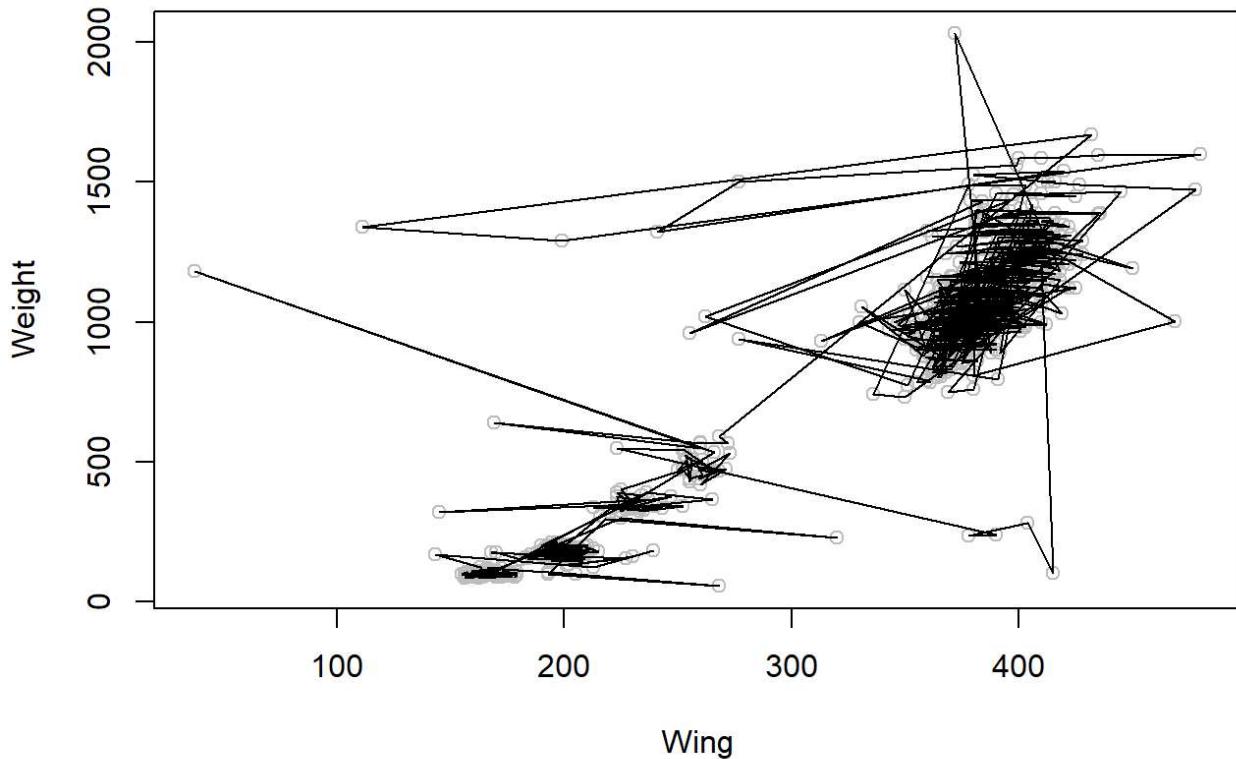
```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls

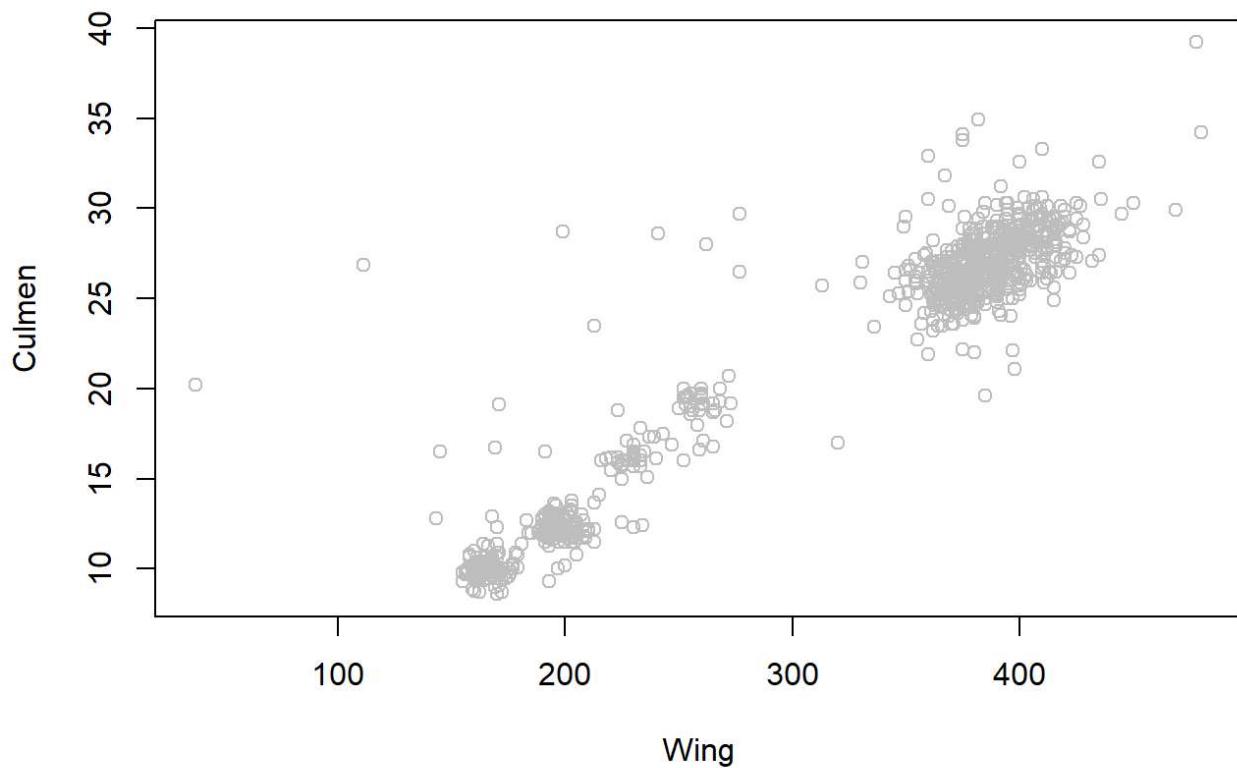


se encuentra mucha diferencia entre una vecindad de 10 y de 20. Ahora se repetirá el proceso con una vecindad de 50.

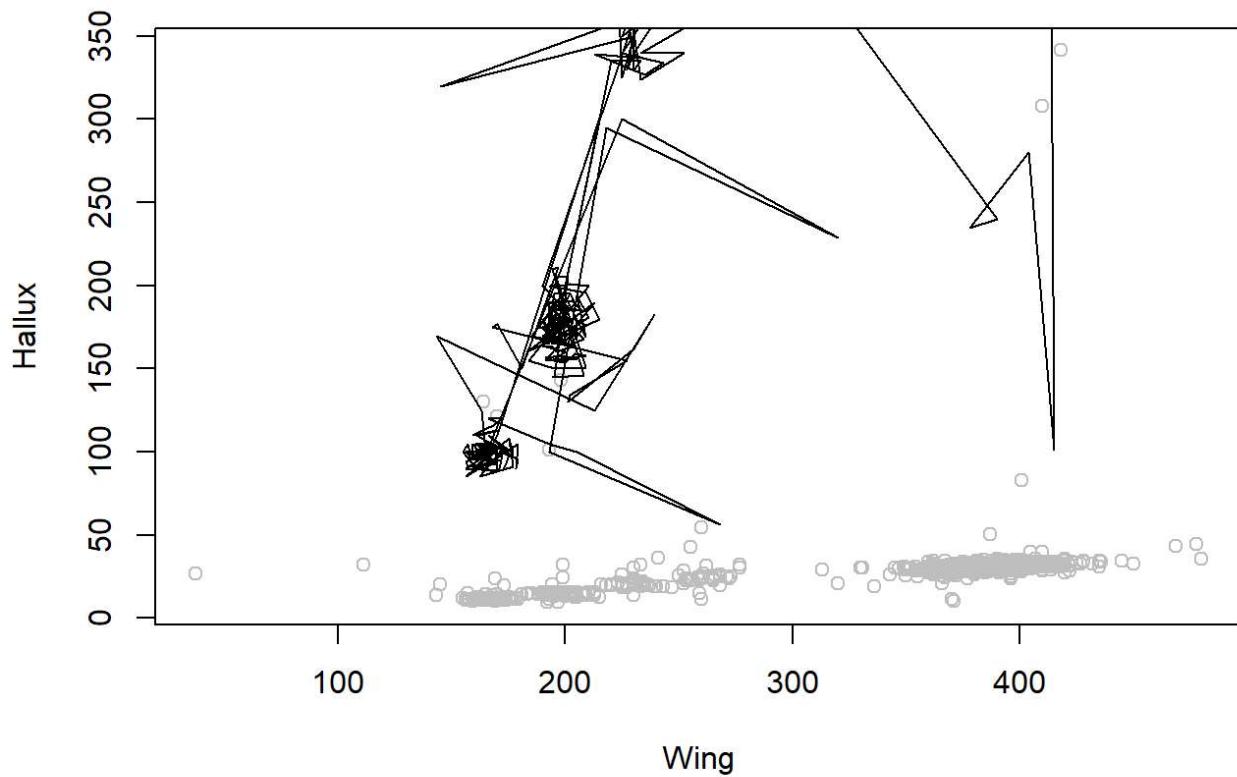
```
### Dibujo de las trazas que relacionan puntos (res50)
plot(Hawks_data[c(1,2)], col = "grey")
polygon(Hawks_data[res50$order,])
```



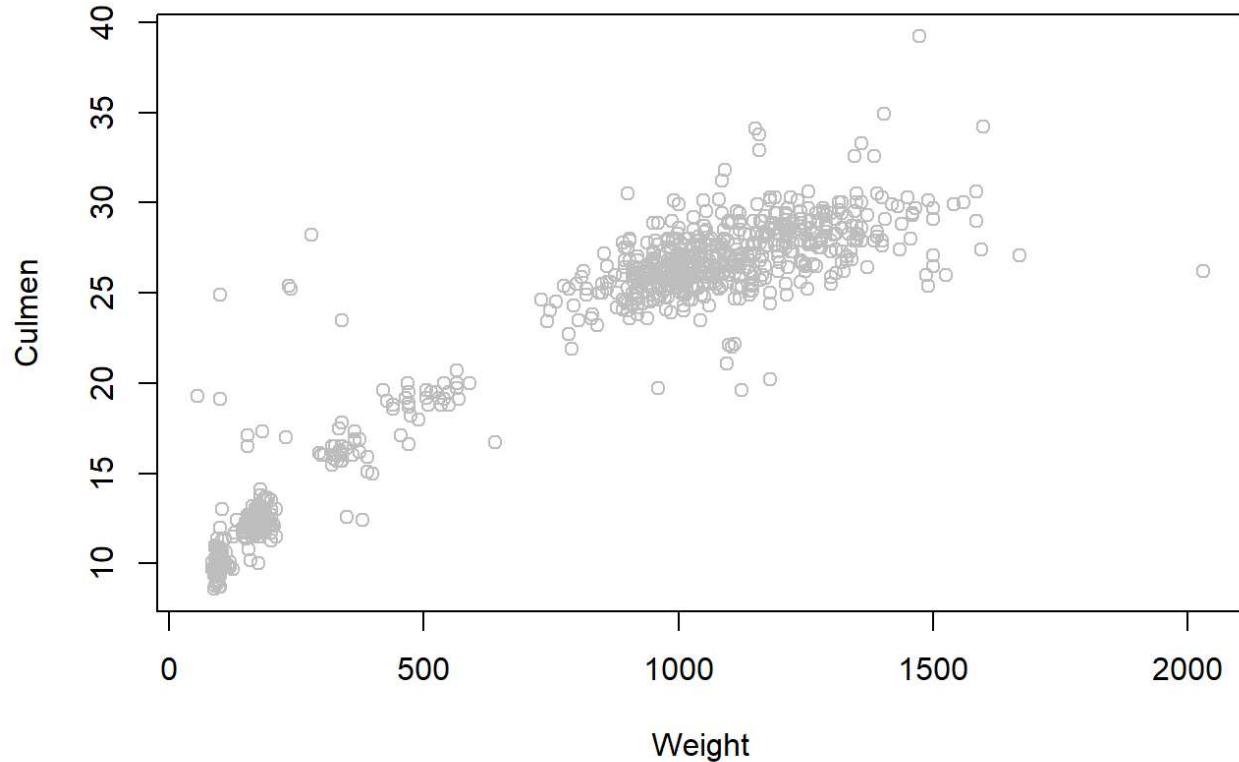
```
### Dibujo de las trazas que relacionan puntos (res50)
plot(Hawks_data[c(1,3)], col = "grey")
polygon(Hawks_data[res50$order,])
```



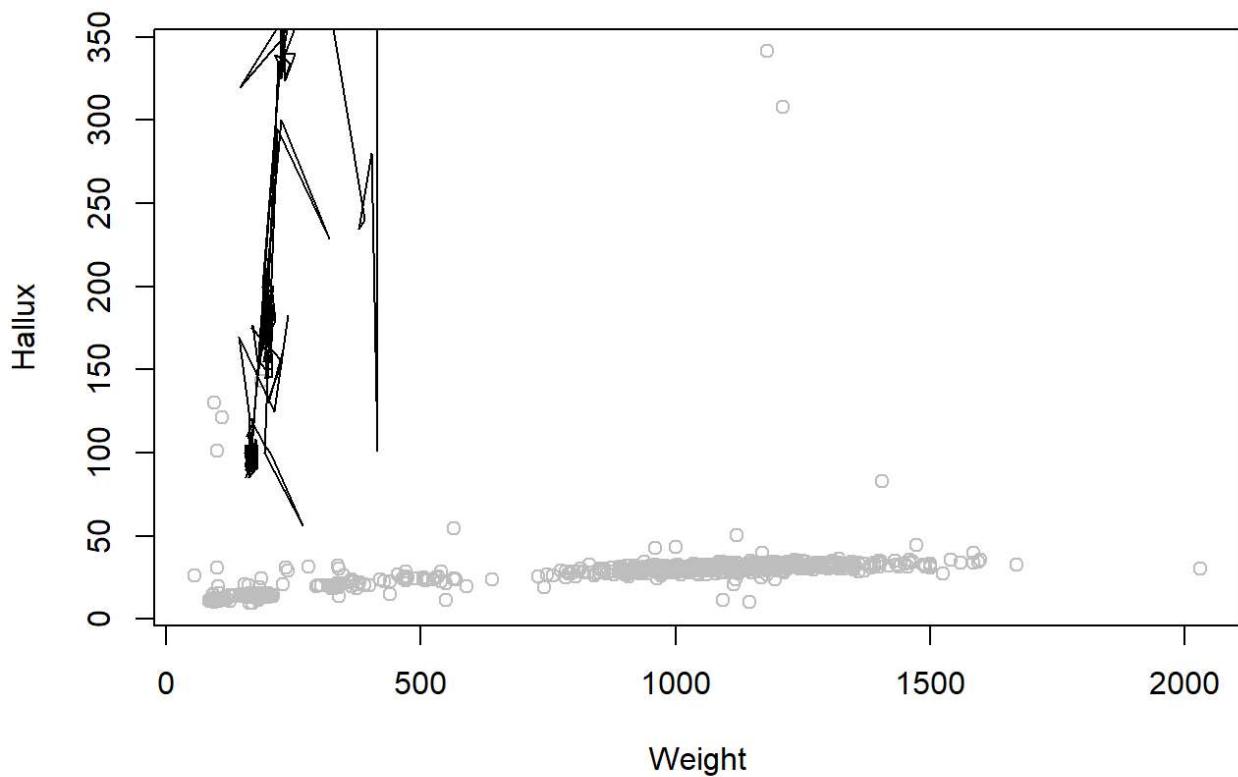
```
### Dibujo de Las trazas que relacionan puntos (res50)
plot(Hawks_data[c(1,4)], col = "grey")
polygon(Hawks_data$res50$order, )
```



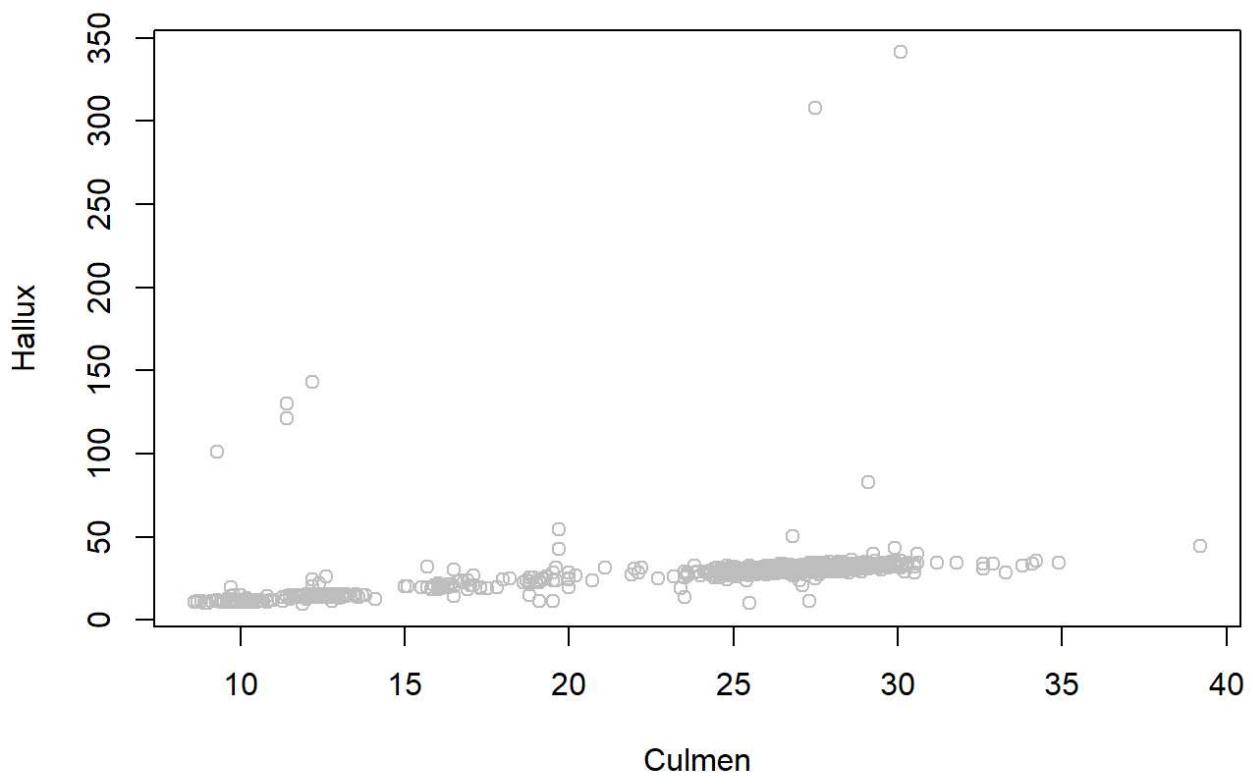
```
### Dibujo de las trazas que relacionan puntos (res50)
plot(Hawks_data[c(2,3)], col = "grey")
polygon(Hawks_data[res50$order,])
```



```
### Dibujo de las trazas que relacionan puntos (res50)
plot(Hawks_data[c(2,4)], col = "grey")
polygon(Hawks_data[res50$order,])
```

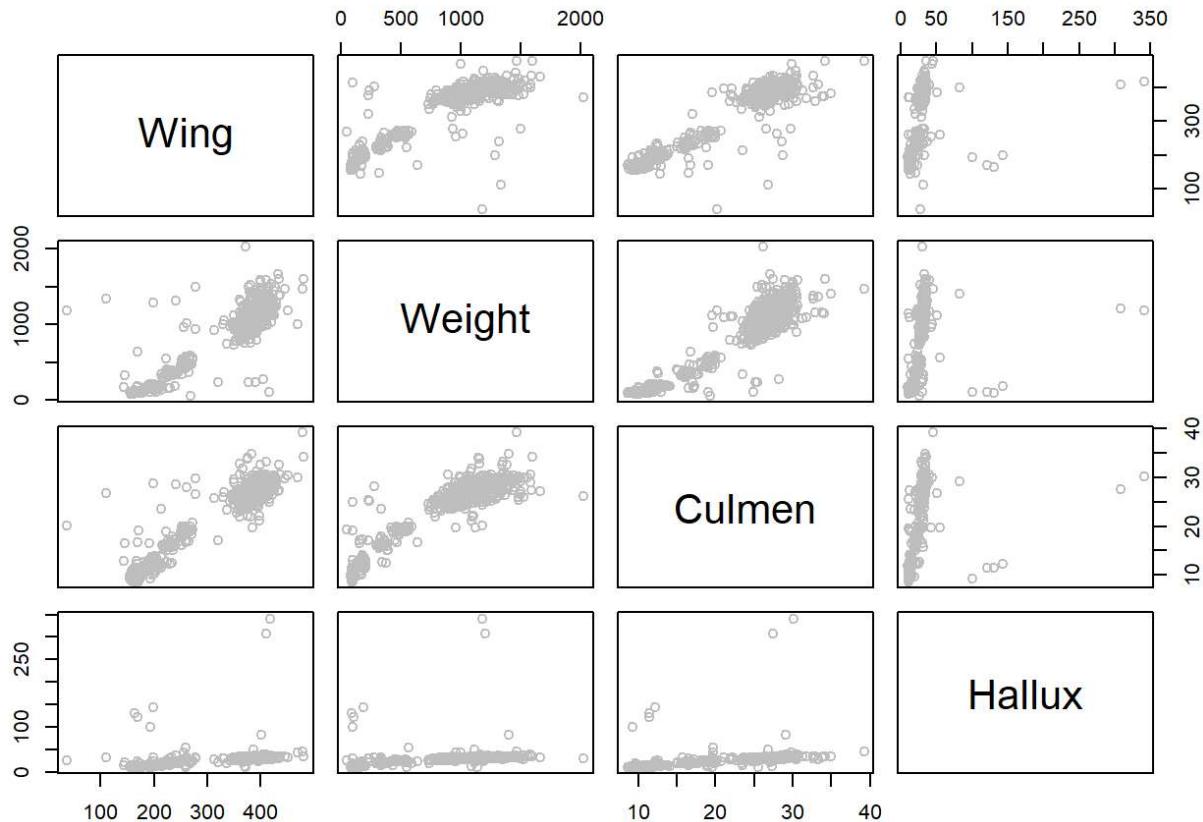


```
### Dibujo de Las trazas que relacionan puntos (res50)
plot(Hawks_data[c(3,4)], col = "grey")
polygon(Hawks_data$res50$order, )
```



La relación de los puntos de manera general es la siguiente:

```
### Dibujo de Las trazas que relacionan puntos (res50)
plot(Hawks_data, col = "grey")
polygon(Hawks_data[res50$order,])
```



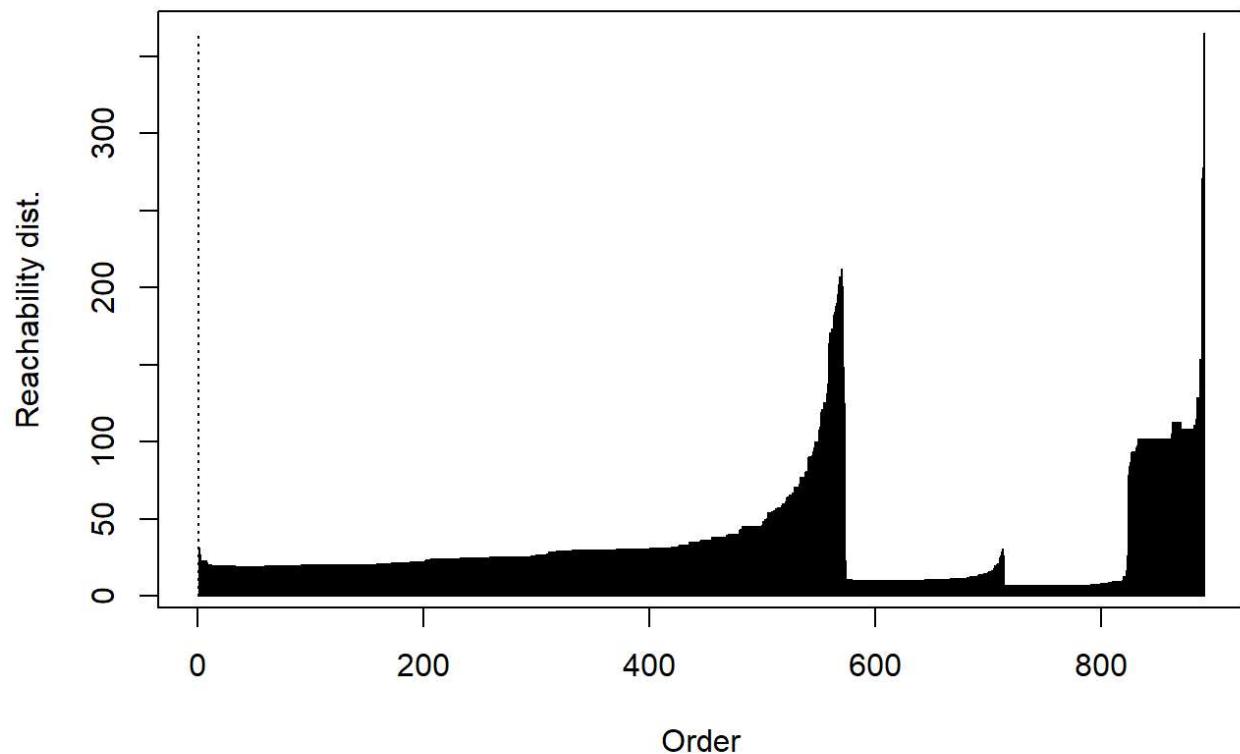
Ahora vamos a probar la alcanzabilidad probando distintos valores de epsilon.

```
### Extracción de un clustering DBSCAN cortando la alcanzabilidad en el valor eps_cl y
con res50
res <- extractDBSCAN(res50, eps_cl = .065)
res
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 50, eps = 686.699861657187, eps_cl = 0.065, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, c
luster
```

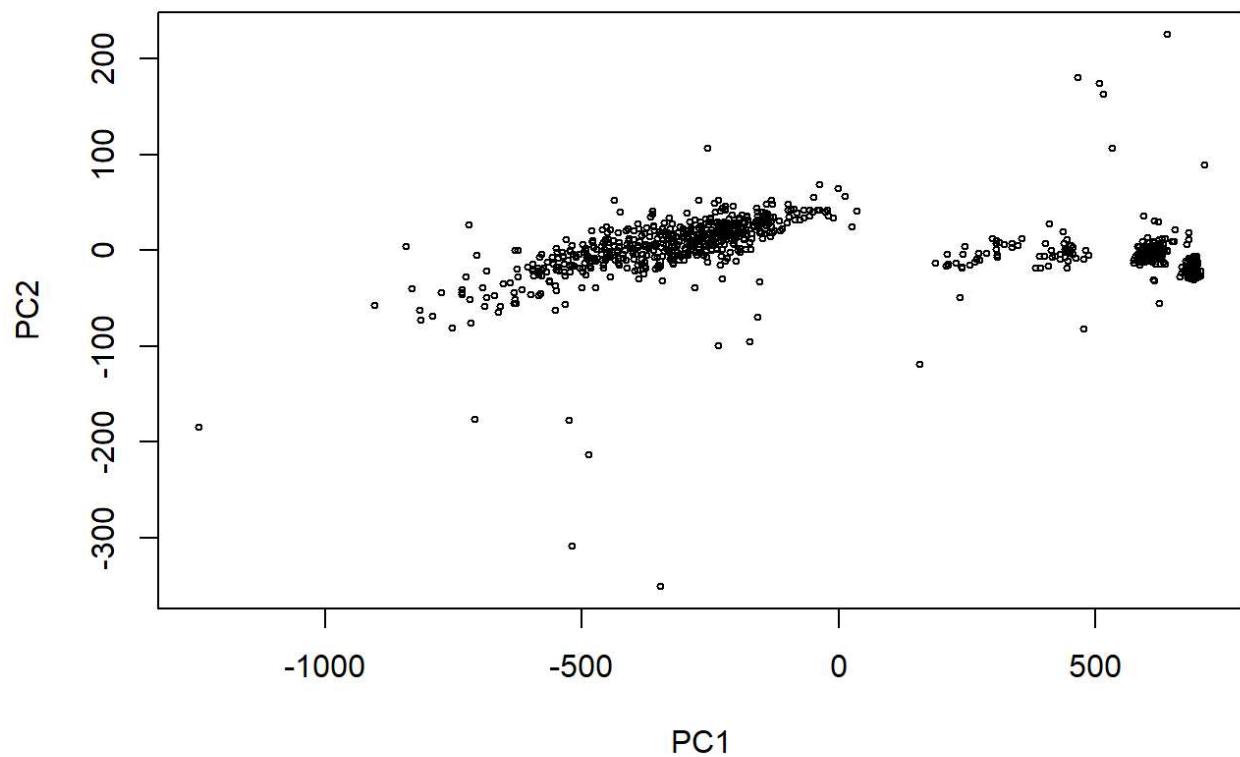
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls

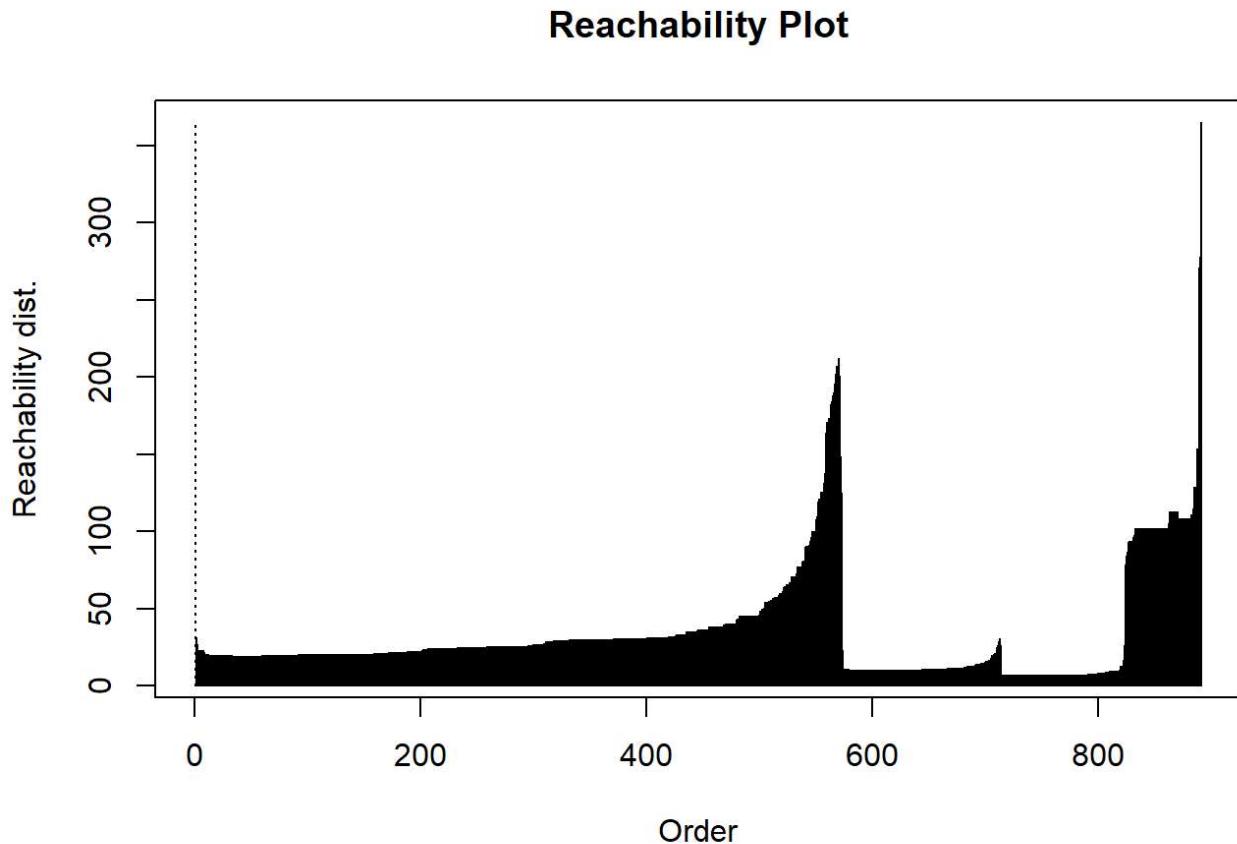


Repetimos el experimento anterior incrementando el parámetro *eps_cl*

```
### Incrementamos el parámetro eps
res <- extractDBSCAN(res50, eps_cl = .1)
res
```

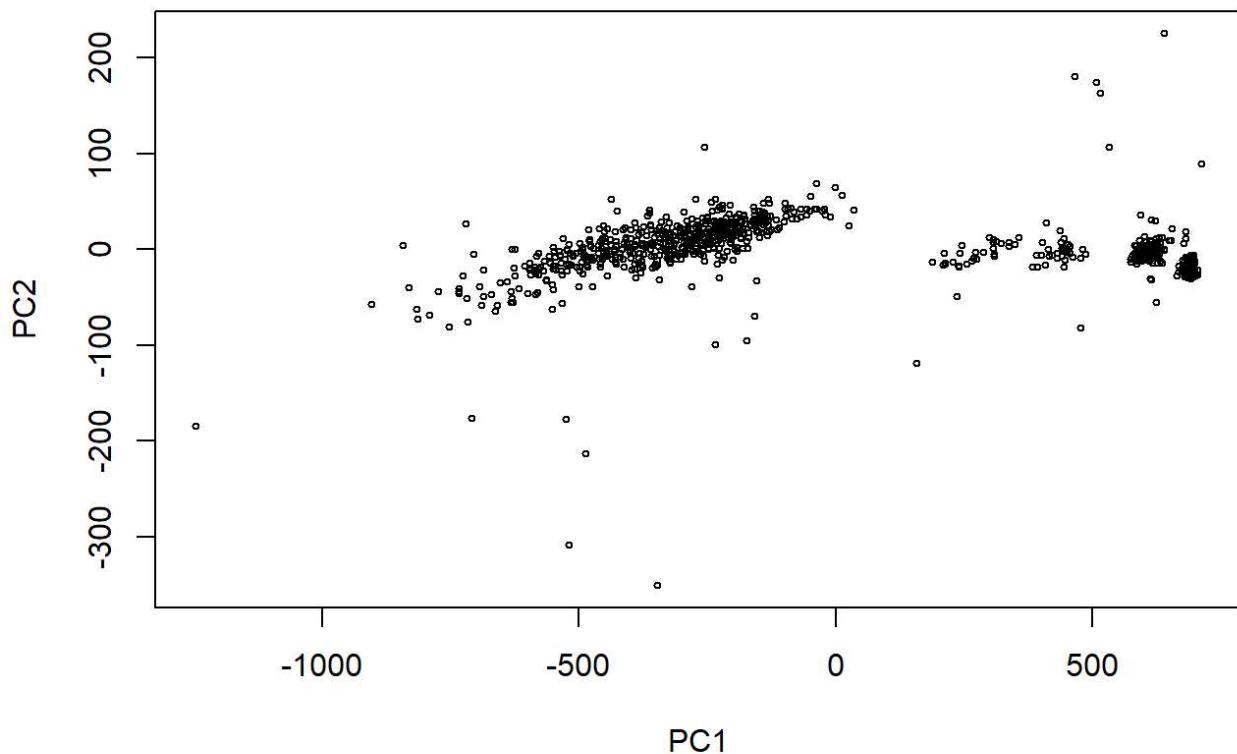
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 50, eps = 686.699861657187, eps_cl = 0.1, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster
```

```
plot(res)
```



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls



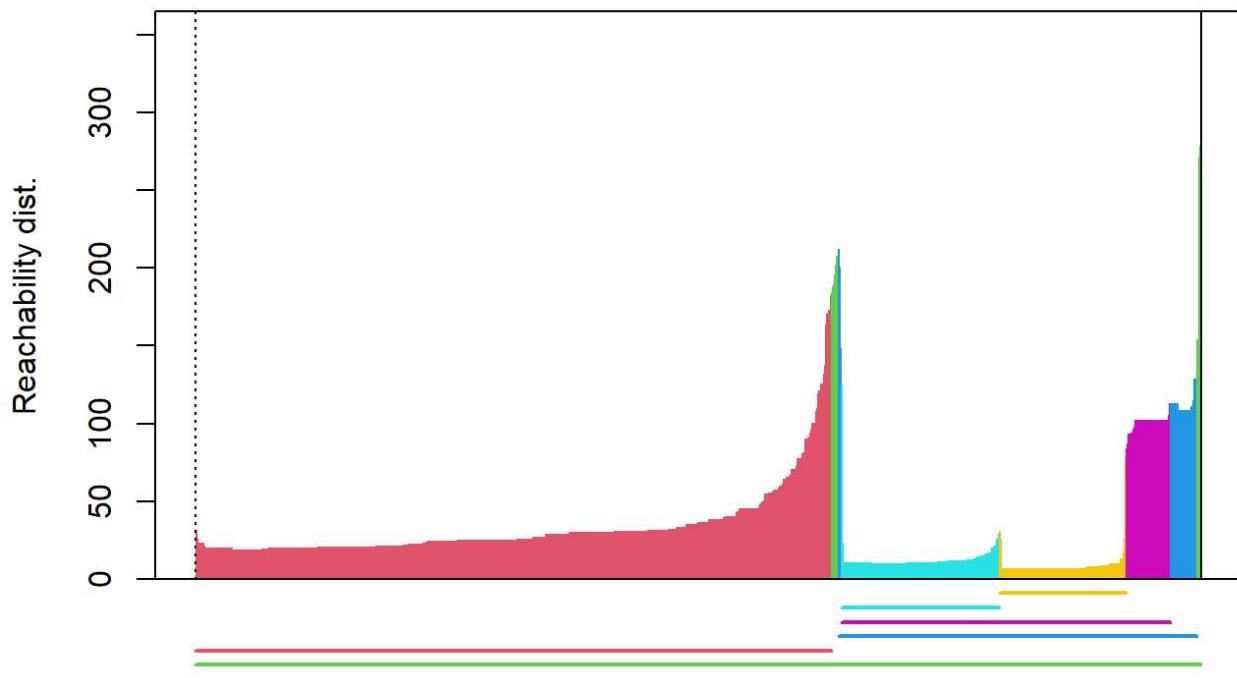
Veamos ahora una variante de la extracción **DBSCN** anterior. En ella el parámetro xi nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

```
### Extracción del clustering jerárquico en función de La variación de La densidad por el método xi
res <- extractXi(res50, xi = 0.05)
res
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 50, eps = 686.699861657187, eps_cl = NA, xi = 0.05
## The clustering contains 6 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

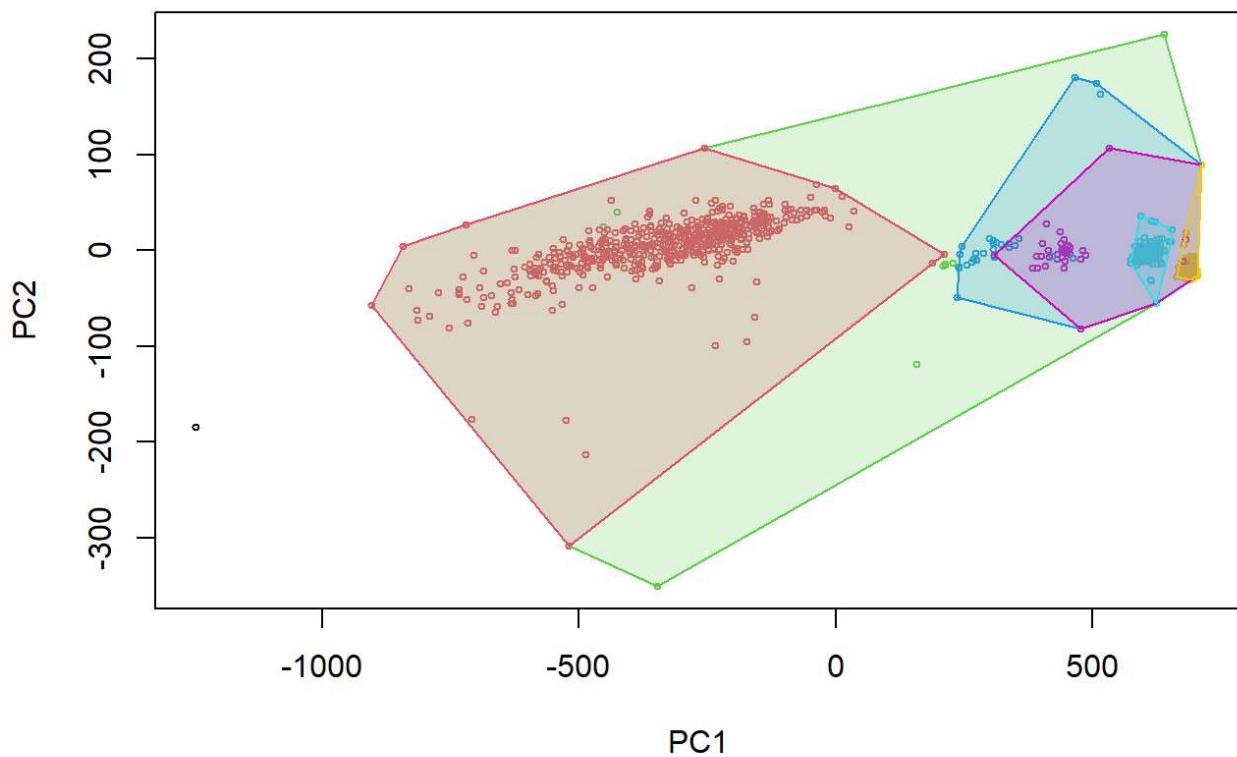
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

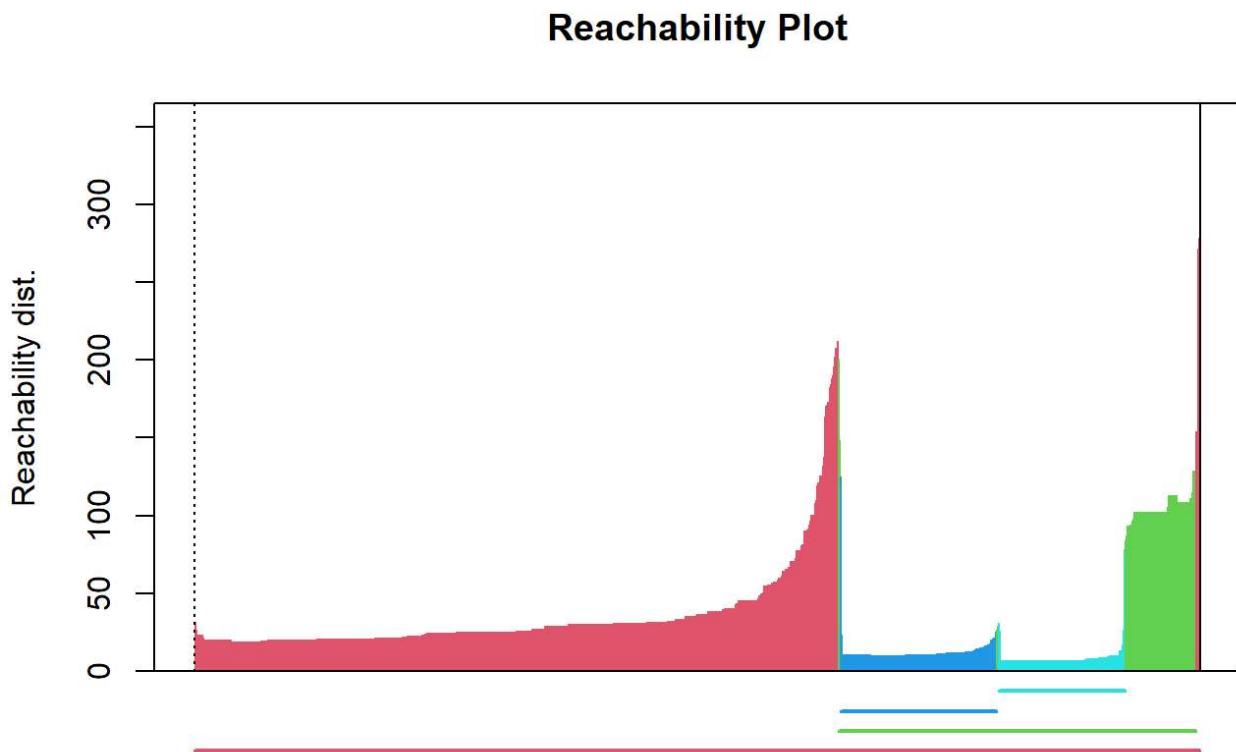
Convex Cluster Hulls



```
### Extracción del clustering jerárquico en función de La variación de La densidad por el método xi
res <- extractXi(res50, xi = 0.1)
res
```

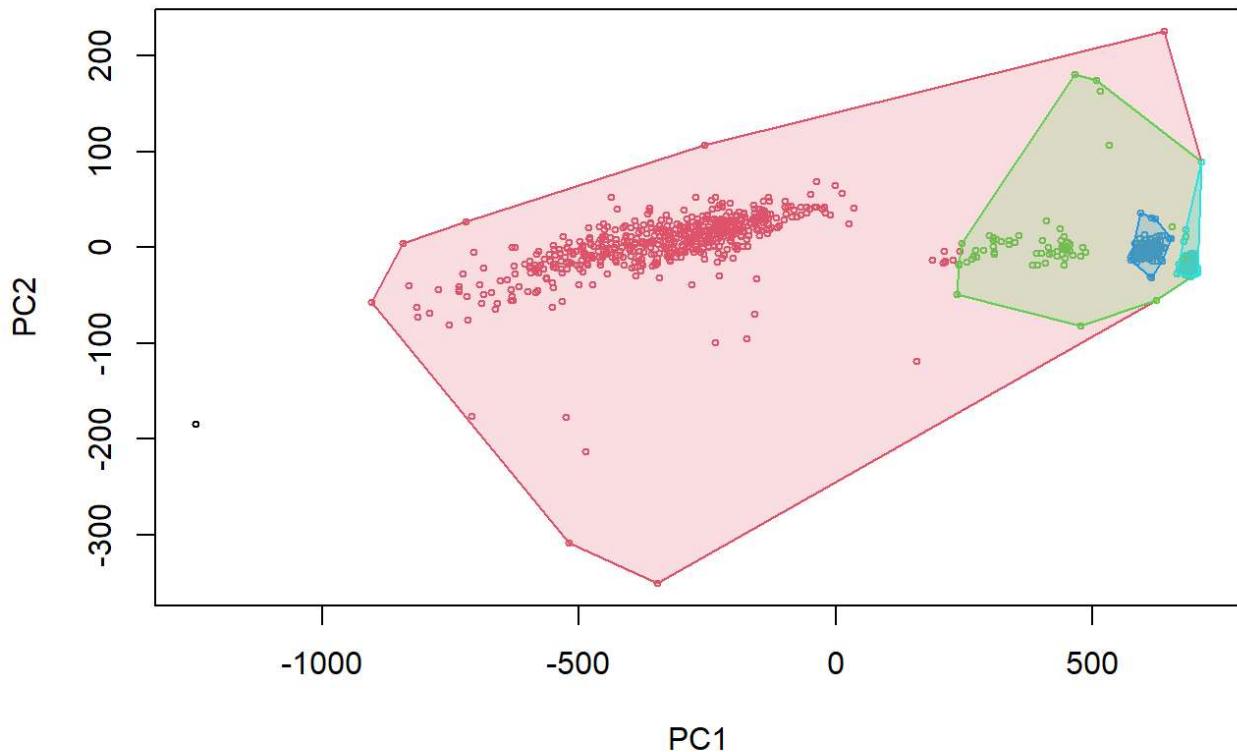
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 50, eps = 686.699861657187, eps_cl = NA, xi = 0.1
## The clustering contains 4 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

```
plot(res)
```



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls



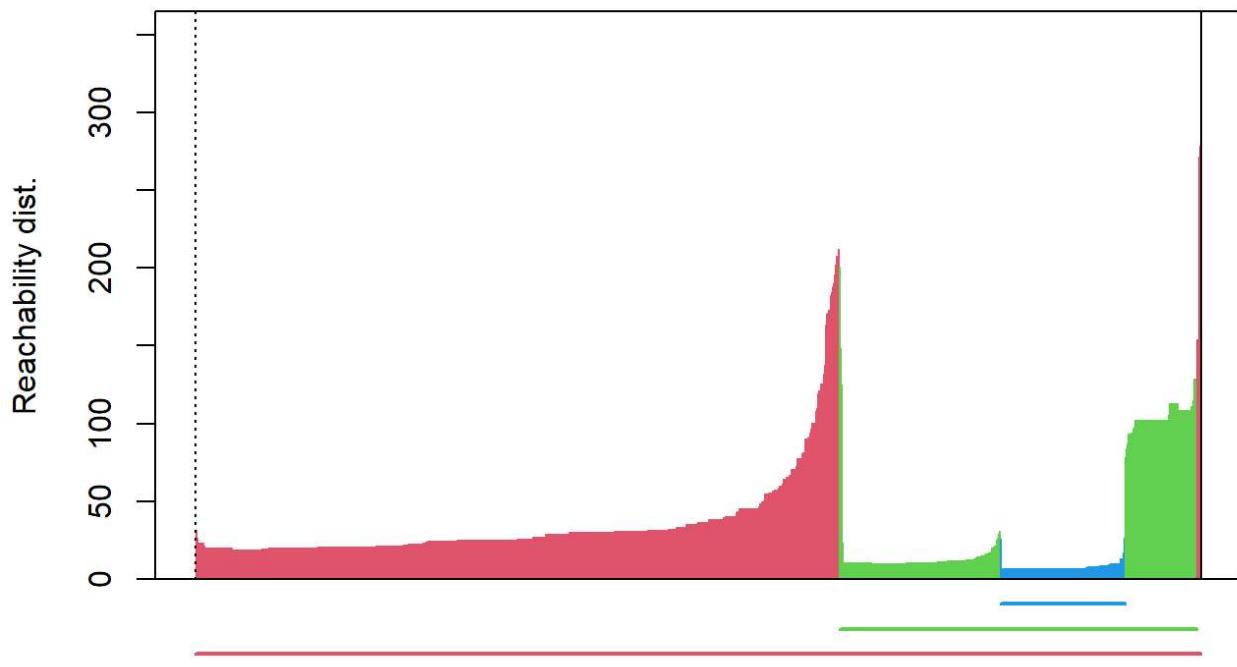
```
### Extracción del clustering jerárquico en función de La variación de La densidad por el método xi
```

```
res <- extractXi(res50, xi = 0.2)
res
```

```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 50, eps = 686.699861657187, eps_cl = NA, xi = 0.2
## The clustering contains 3 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

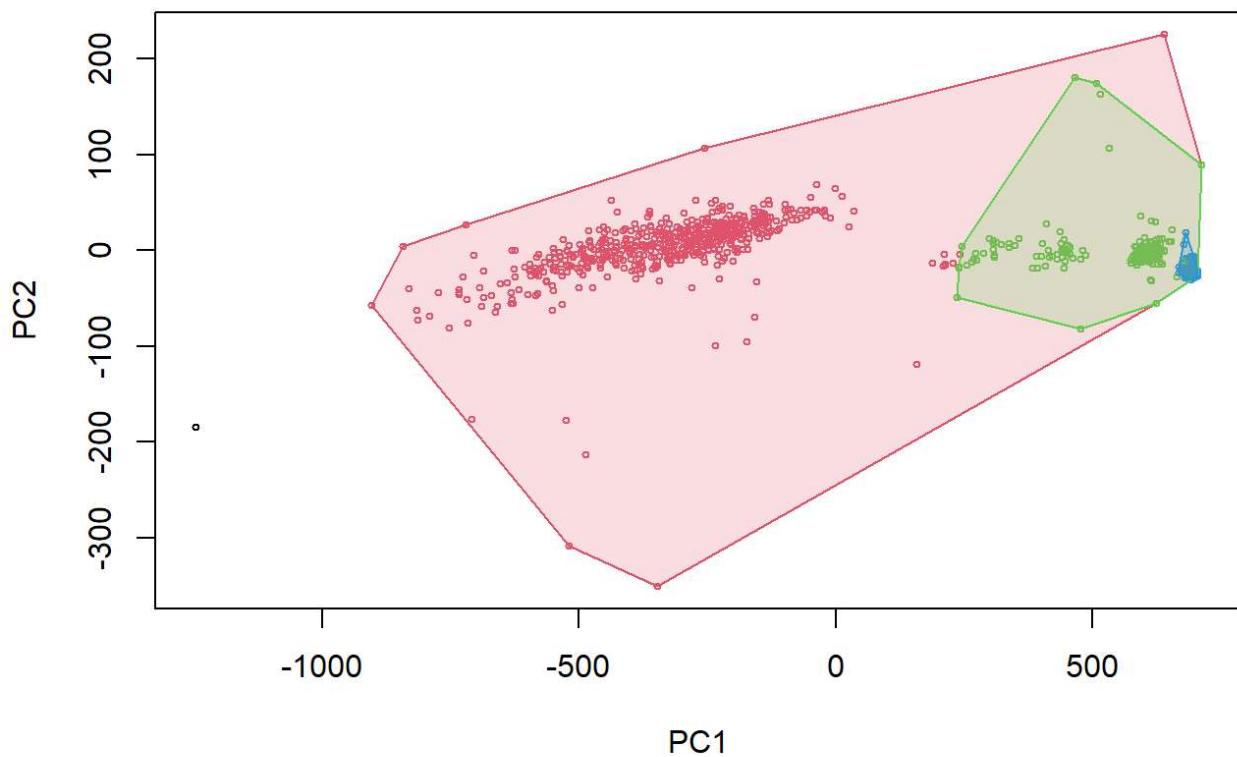
```
plot(res)
```

Reachability Plot



```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls



Ahora con un índice de vecindad se encuentran distintas diferencias notables tanto en el numero de clústeres y en la clasificación que estos hacen.

Como conclusión se puede ver que este algoritmo agrupa los distintos tipos de una manera mas visual y con distintas geometrías.

5.3 Ejercicio 3

Realiza una comparativa de los métodos *k-means* y *DBSCAN*

5.3.1 Respuesta 3

Escribe aquí la respuesta a la pregunta

Tras aplicar los algoritmos *k-means* y *DBSCAN* vemos que con los dos se obtienen resultados razonables, es decir, con el algoritmo *k-means* con 3 clústeres tenemos una calidad de agrupamiento del 66,3% y con 5 un poco más bajo un 59%. Con el algoritmo *DBSCAN* se obtiene distinto numero de clústeres con los análisis de vecindad. Por ejemplo, con 50 y un xi de 0.2 obtenemos 3 clústeres.

Entre las diferencias más notables encontramos:

1. Los grupos formados en *k-means* son de forma más o menos esférica o convexa y deben tener el mismo tamaño de característica, mientras que los grupos formados en *DBSCAN* tienen una forma arbitraria y pueden no tener el mismo tamaño de característica.
2. La agrupación de *k-means* es sensible a la cantidad de agrupaciones especificadas mientras que en *DBSCAN* no es necesario especificar el número de conglomerados.
3. La agrupación en clústeres de *k-means* es más eficiente para grandes conjuntos de datos mientras que en *DBSCAN* no puede manejar de manera eficiente conjuntos de datos de gran dimensión.
4. La agrupación en clústeres de *k-means* no funciona bien con valores atípicos y conjuntos de datos ruidosos, mientras que la agrupación en clústeres de *DBSCAN* maneja de manera eficiente los valores atípicos y los conjuntos de datos ruidosos.
5. En el ámbito de la detección de anomalías, *k-means* causa problemas, ya que los puntos anómalos se asignarán al mismo grupo que los puntos de datos “normales” mientras que el algoritmo *DBSCAN*, por otro lado, localiza regiones de alta densidad que están separadas entre sí por regiones de baja densidad.
6. *k-means* requiere el número de clústeres (K) mientras que *DBSCAN* requiere dos parámetros: *MinPts* y *Eps*.
7. Las densidades variables de los puntos de datos no afectan el algoritmo de agrupación de *k-means*, mientras que en *DBSCAN* no funciona muy bien para conjuntos de datos dispersos o para puntos de datos con densidad variable.

La realización de este PEC me ha ayudado a entender el funcionamiento de los algoritmos *DBSCAN* y *k-means*. Lo que se puede concluir que depende de lo que se quiera buscar es mas eficiente usar un algoritmo que otro.

La realización de este PEC me ha ayudado a entender el funcionamiento de los algoritmos *DBSCAN* y *k-means*, para este conjunto de datos ha sido difícil (o al menos para mi) encontrar un resultado bastante claro en la forma de agrupar las distintas especies. Lo que si puedo sacar en claro que depende el conjunto de datos se debe usar un algoritmo u otro, ya que como se ha mencionado anteriormente, depende de lo que se quiera encontrar o clasificar.

6 Criterios de evaluación

6.1 Ejercicio 1

- 10%. Se explican los campos de la base de datos
- 25%. Se aplica el algoritmo de *k-means* de forma correcta.
- 25%. Se prueban con diferentes valores de k.
- 10%. Se obtiene una medida de lo bueno que es el agrupamiento.
- 20%. Se describen e interpretan los diferentes clusters obtenidos.
- 10%. Se presenta el código y es fácilmente reproducible.

6.2 Ejercicio 2

- 20%. Se aplican los algoritmos *DBSCAN* y *OPTICS* de forma correcta.
- 25%. Se prueban con diferentes valores de eps.
- 25%. Se obtiene una medida de lo bueno que es el agrupamiento.
- 20%. Se describen e interpretan los diferentes clusters obtenidos.
- 10%. Se presenta el código y es fácilmente reproducible.

6.3 Ejercicio 3

- 35%. Se comparan los resultados obtenidos en *k-means* y *DBSCAN*.
- 35%. Se mencionan pros y contras de ambos algoritmos
- 30%. Se exponen las conclusiones del trabajo