

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)

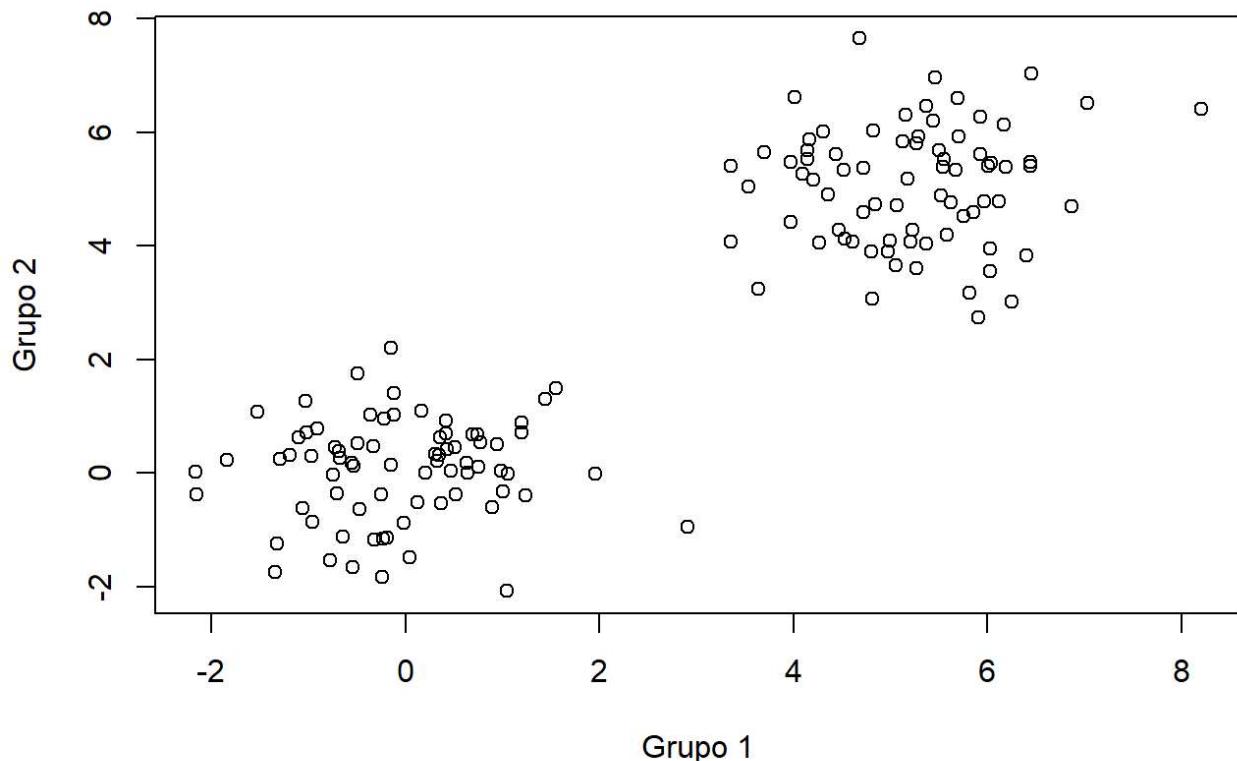
```

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

```

fit2      <- kmeans(x, 2)
y_cluster2 <- fit2$cluster

fit4      <- kmeans(x, 4)
y_cluster4 <- fit4$cluster

fit8      <- kmeans(x, 8)
y_cluster8 <- fit8$cluster

```

Las variables y_cluster2, y_cluster4 e y_cluster8 contienen para cada muestra el identificador del clúster a las que han sido asignadas. Por ejemplo, en el caso de los k=2 las muestras se han asignado al clúster 1 o al 2

```
y_cluster2
```

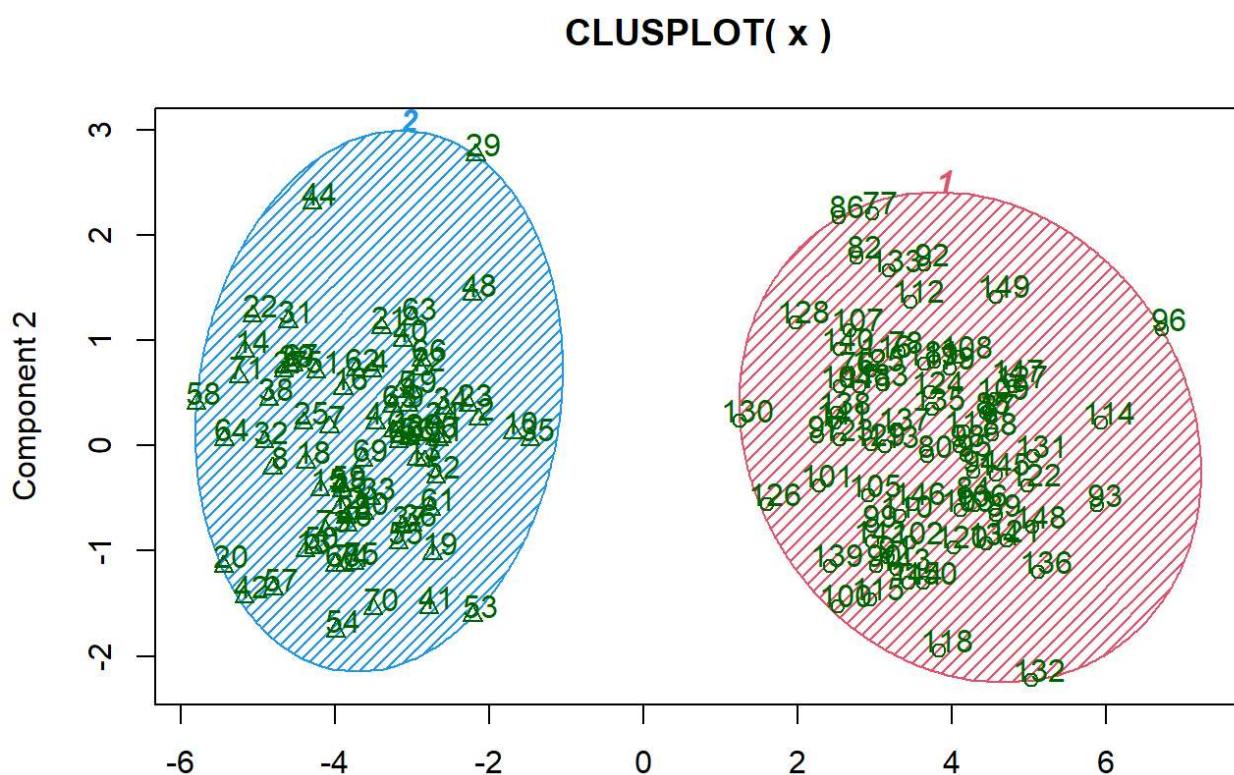
```

##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1
## [95] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

```

Para visualizar los clústers podemos usar la función clusplot. Vemos la agrupación con 2 clústers

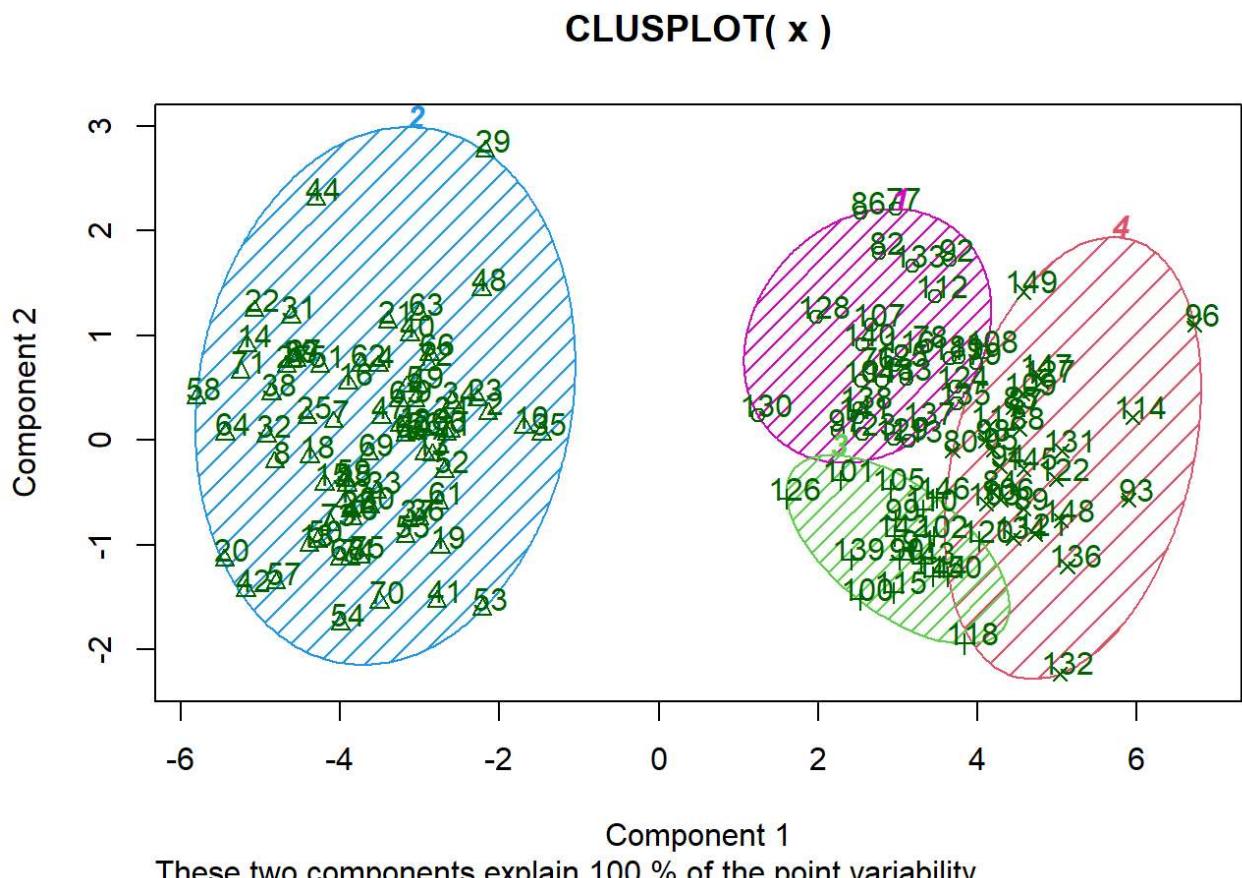
```
clusplot(x, fit2$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



Component 1
These two components explain 100 % of the point variability.

con 4

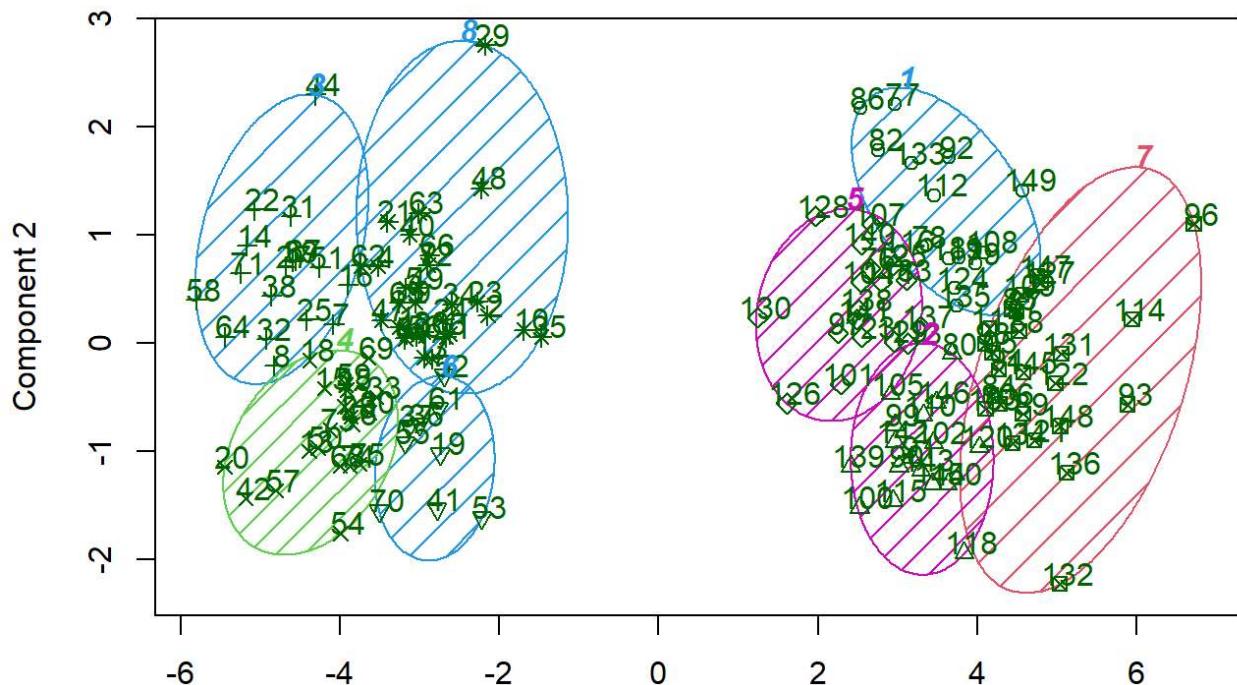
```
clusplot(x, fit4$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



y con 8

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

CLUSPLOT(x)

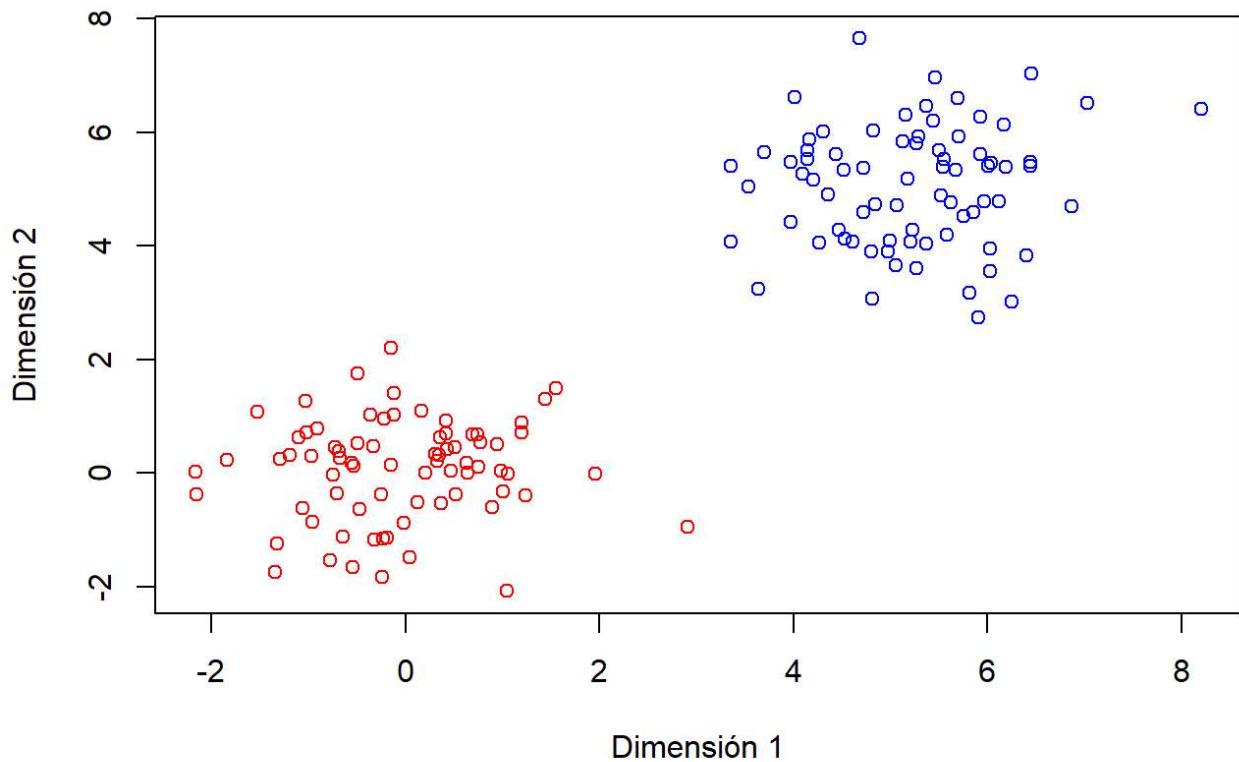


Component 1

These two components explain 100 % of the point variability.

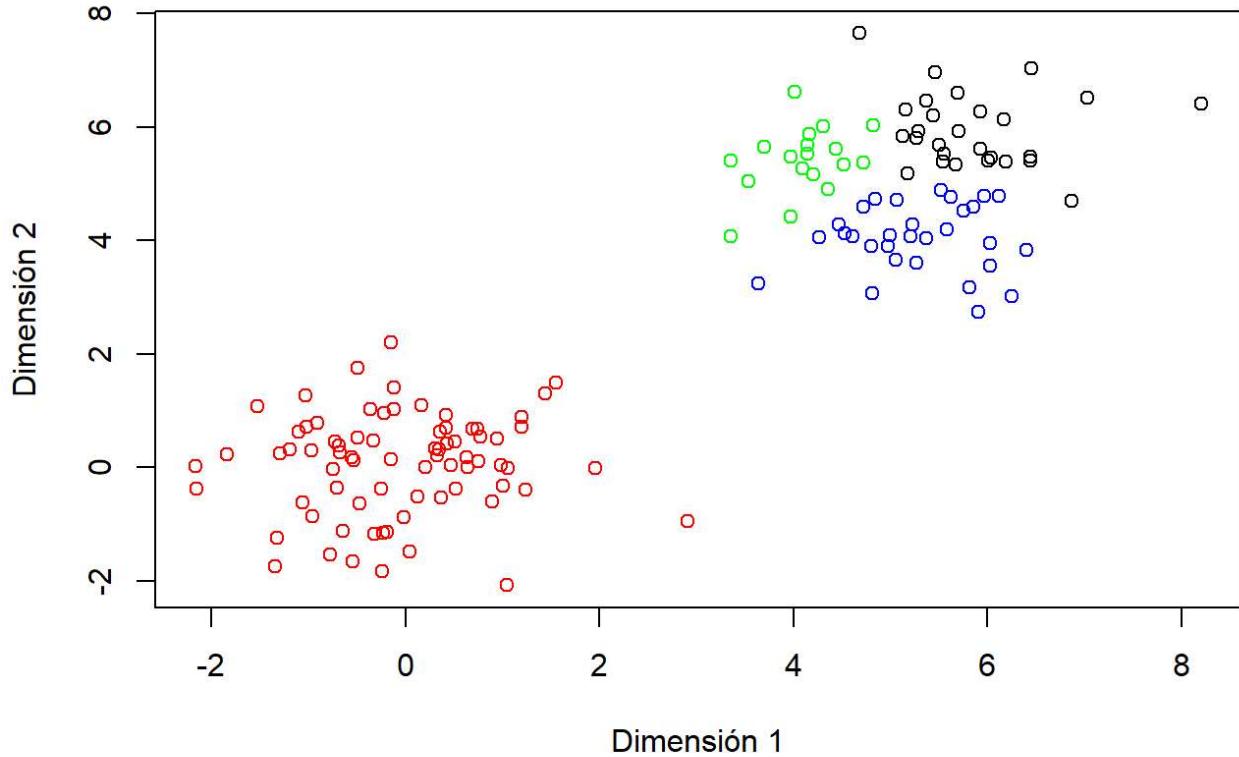
También podemos visualizar el resultado del proceso de agrupamiento con el siguiente código para el caso de 2 clústeres

```
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')
```



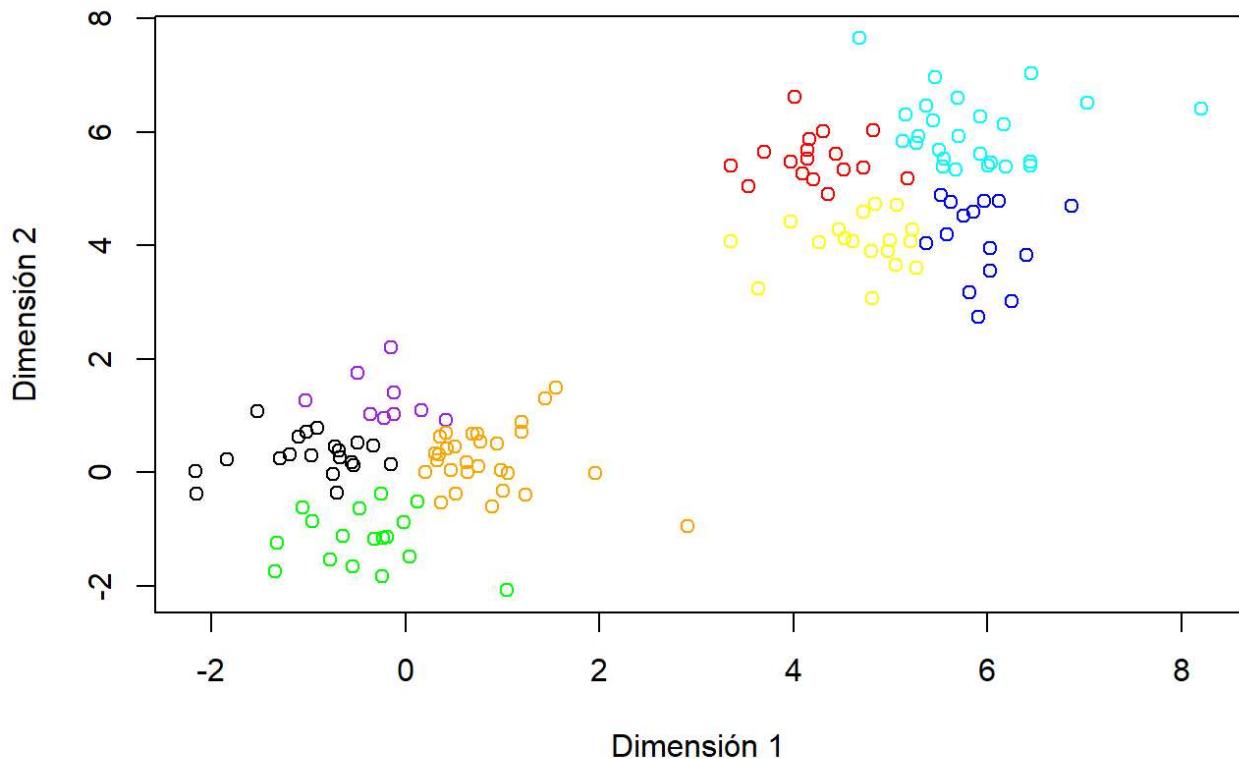
para 4

```
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
```

```

## # A tibble: 344 x 8
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##   <fct>  <fct>           <dbl>          <dbl>            <int>        <int> <fct>
##   <int>
## 1 Adelie  Torgersen     39.1          18.7            181        3750 male 
## 2 Adelie  Torgersen     39.5          17.4            186        3800 femal
## 3 Adelie  Torgersen     40.3          18              195        3250 femal
## 4 Adelie  Torgersen     NA             NA              NA         NA  <NA>
## 5 Adelie  Torgersen     36.7          19.3            193        3450 femal
## 6 Adelie  Torgersen     39.3          20.6            190        3650 male 
## 7 Adelie  Torgersen     38.9          17.8            181        3625 femal
## 8 Adelie  Torgersen     39.2          19.6            195        4675 male 
## 9 Adelie  Torgersen     34.1          18.1            193        3475 <NA>
## 10 Adelie Torgersen     42              20.2            190        4250 <NA>
## # ... with 334 more rows

```

```
summary(penguins)
```

```

##      species      island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##      <fct>       <fct>           <dbl>          <dbl>            <int>        <int> <fct>
##      <int>
## 1 Adelie  Biscoe     Min.   :32.10      Min.   :13.10      Min.   :172.0   Mi
## n. :2700   female:165      Min.   :2007
## 2 Chinstrap:68 Dream     1st Qu.:39.23    1st Qu.:15.60    1st Qu.:190.0   1s
## t Qu.:3550 male :168      1st Qu.:2007
## 3 Gentoo   Torgersen:52 Median :44.45    Median :17.30    Median :197.0   Me
## dian :4050 NA's  :11      Median :2008
## 4 Adelie  Biscoe     Mean   :43.92      Mean   :17.15      Mean   :200.9   Me
## an   :4202
## 5 Chinstrap:68 Dream     3rd Qu.:48.50    3rd Qu.:18.70    3rd Qu.:213.0   3r
## t Qu.:4750 male :168      3rd Qu.:2009
## 6 Gentoo   Torgersen:52 Max.   :59.60      Max.   :21.50      Max.   :231.0   Ma
## dian :4050 NA's  :11      Max.   :2009
## 7 Adelie  Biscoe     NA's   :2          NA's   :2          NA's   :2       N
## A's   :2

```

Como se puede comprobar, esta base de datos está pensada para problemas de clasificación supervisada que pretende clasificar cada tipo de pingüino en una de las tres clases o especies existentes (Adelie, Gentoo o Chinstrap). Como en este ejemplo vamos a usar un método no supervisado, transformaremos el problema

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.

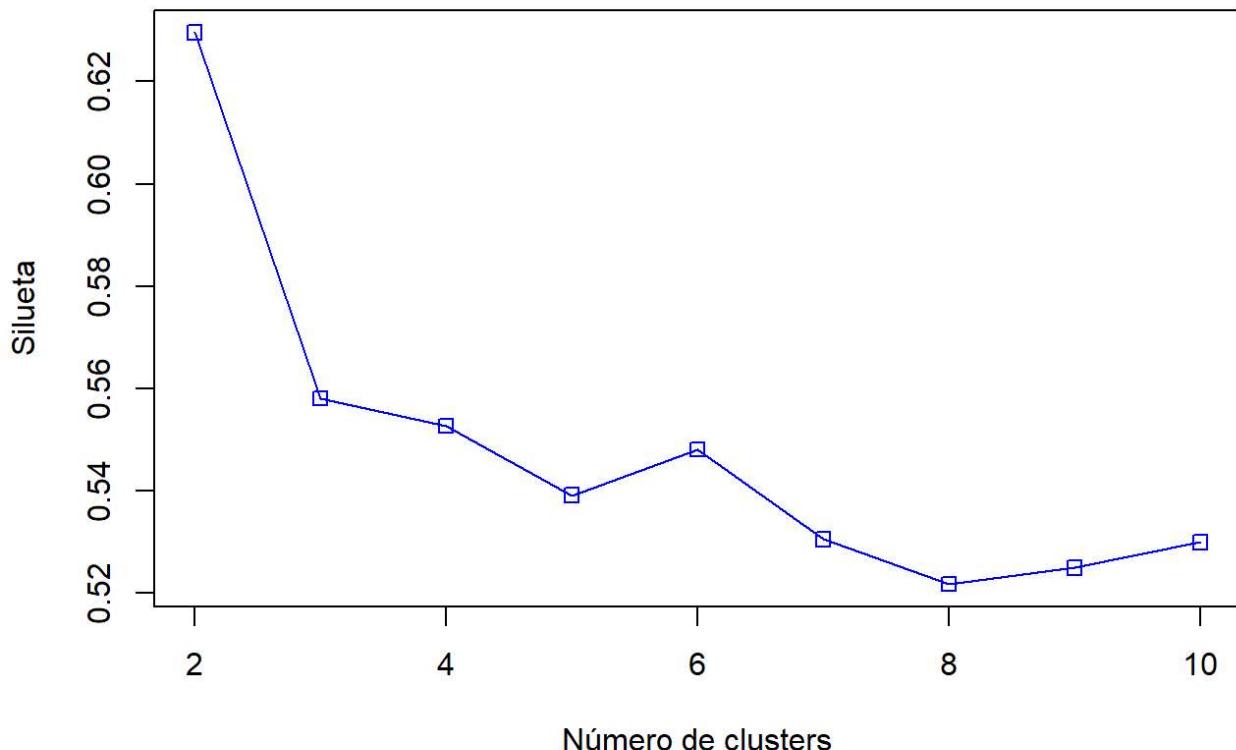
```
x <- na.omit(penguins[,3:6])
```

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 que número de clústers es el mejor.

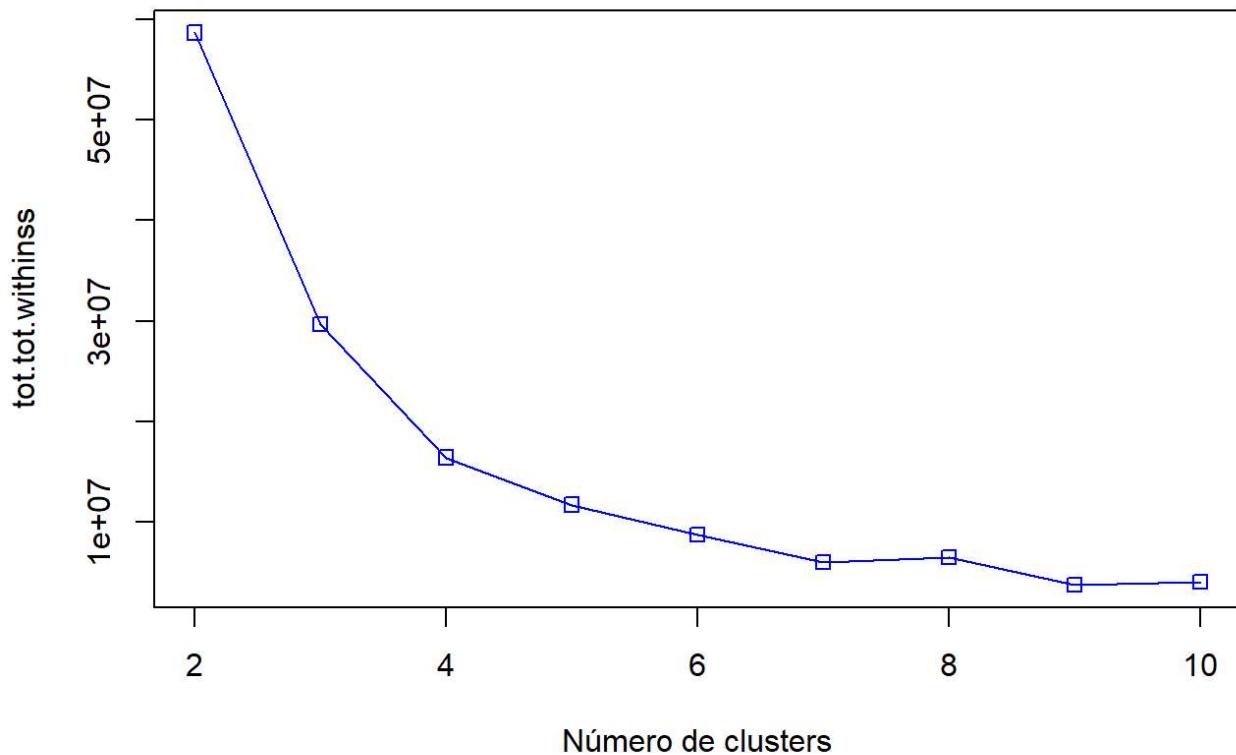
```
plot(2:10,resultados[2:10],type="o",col="blue",pch=0,xlab="Número de clusters",ylab="Silueta")
```



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

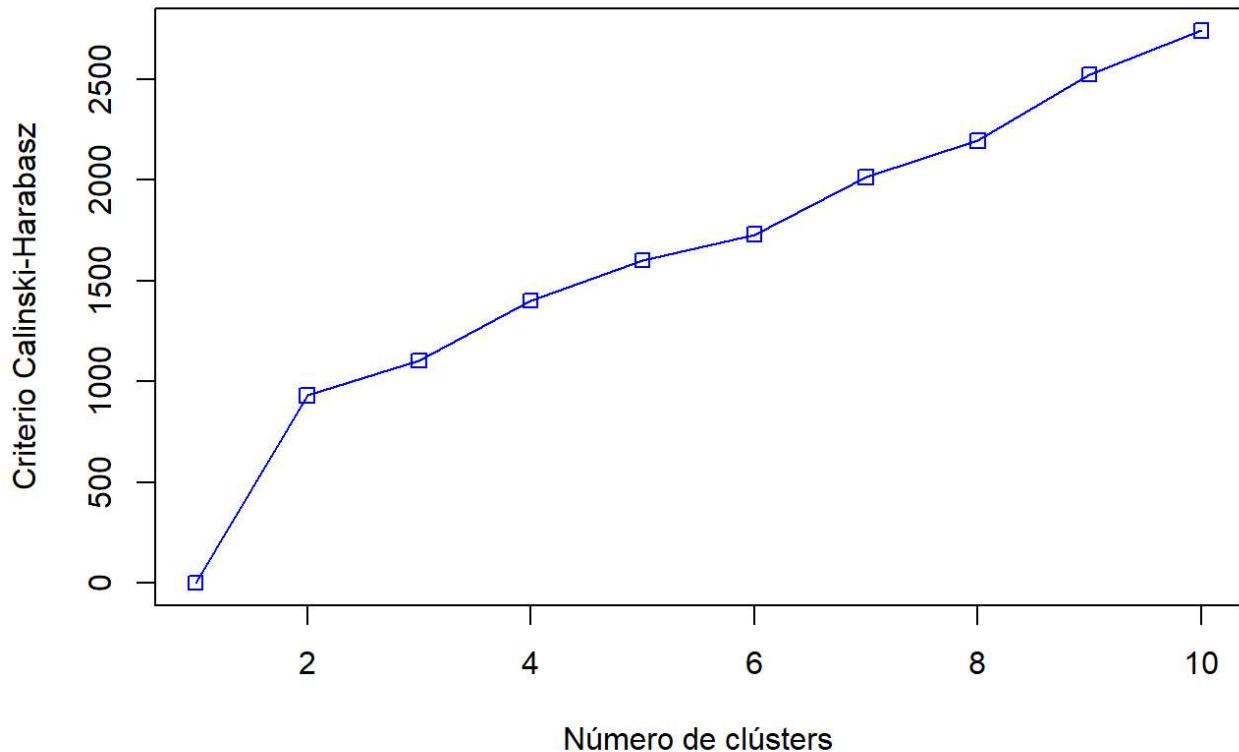
```
fit_ch$bestk
```

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

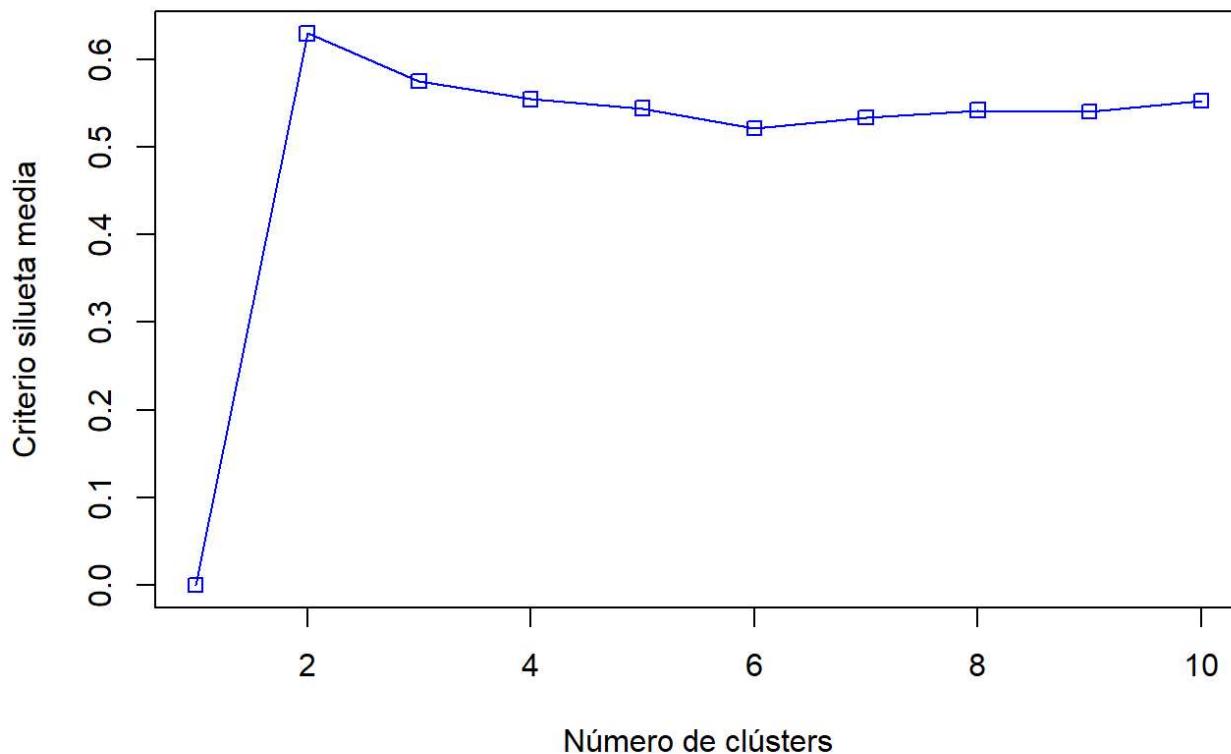
```
fit_asw$bestk
```

```
## [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.

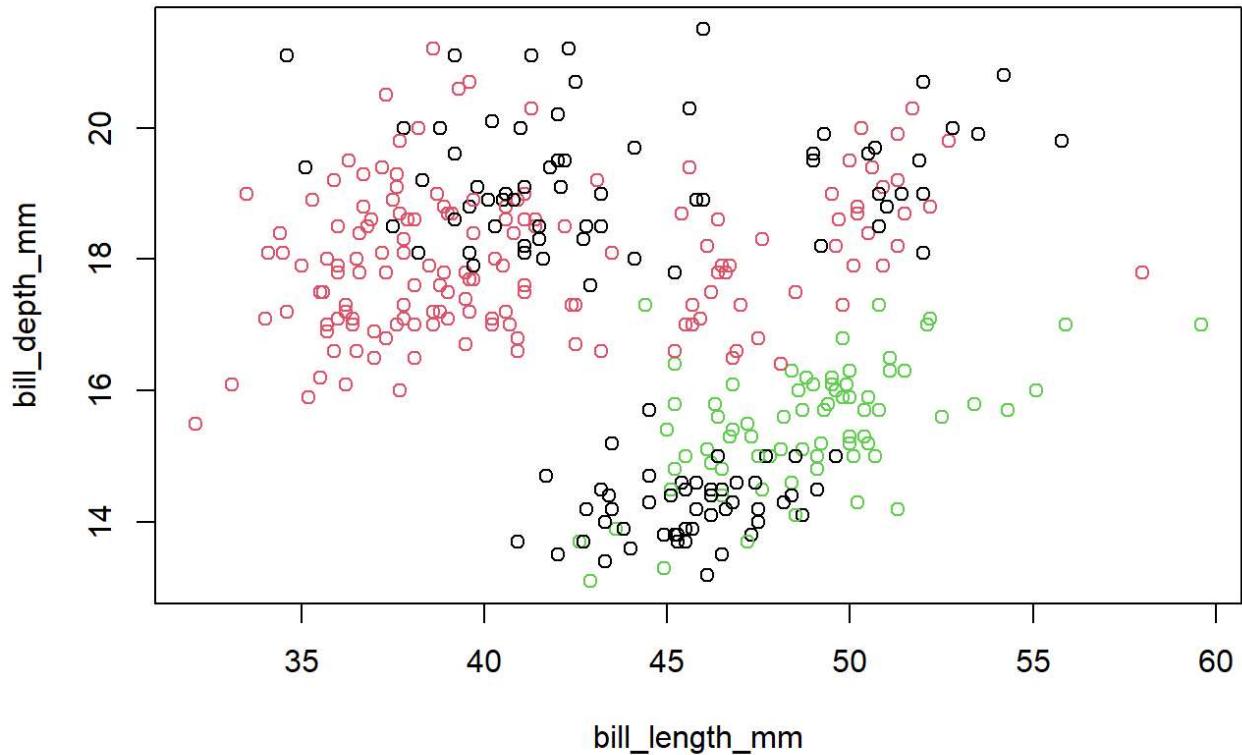
Como se ha comprobado, conocer el número óptimo de clústers no es un problema fácil. Tampoco lo es la evaluación de los modelos de agregación.

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)

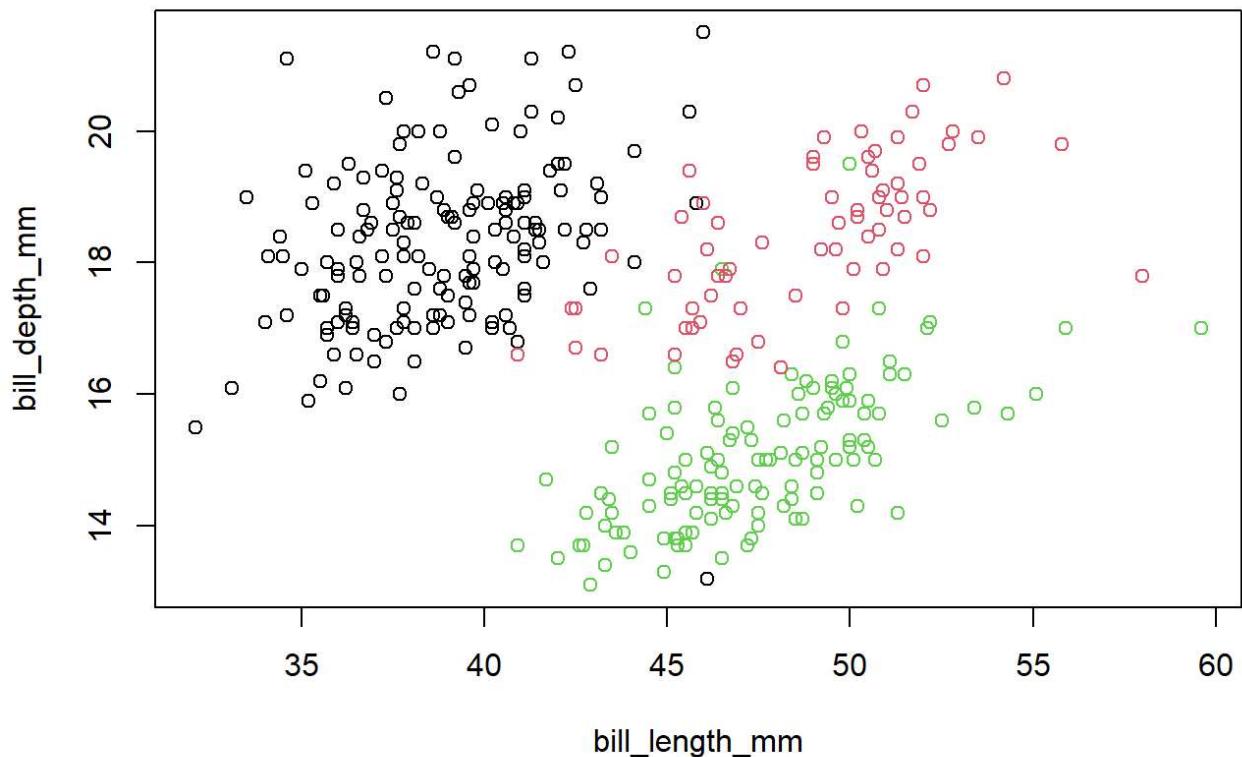
# bill_LLength y bill_depth
plot(x[c(1,2)], col=penguins3clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



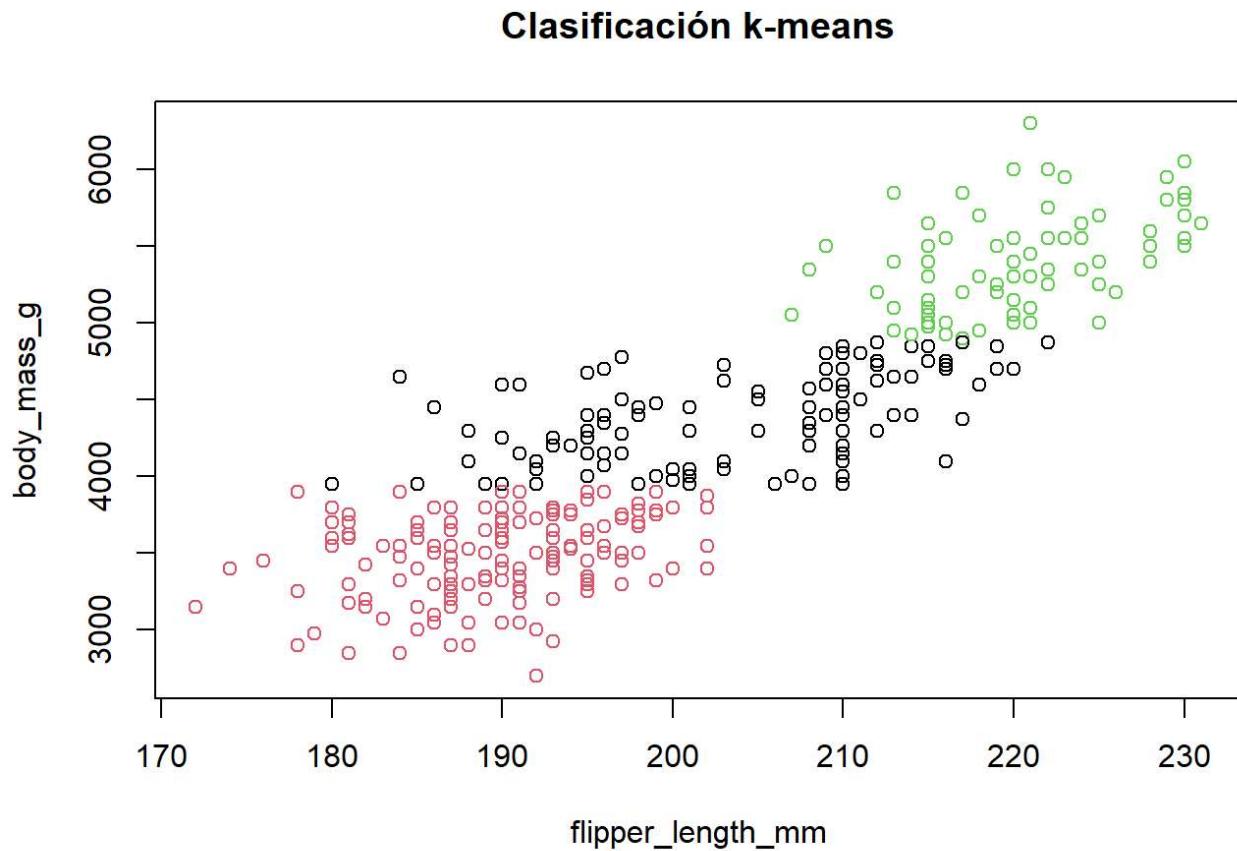
```
plot(x[c(1,2)], col=as.factor(penguins$species), main="Clasificación real")
```

Clasificación real



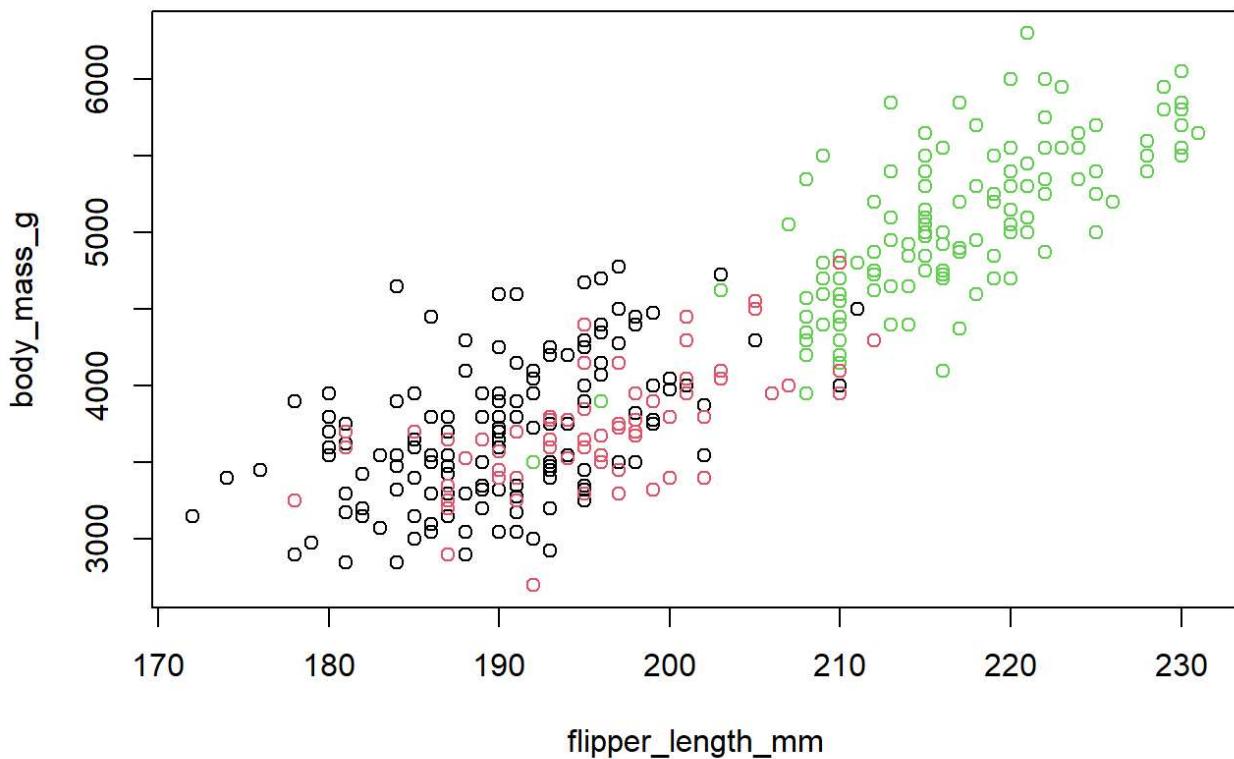
Podemos observar que *flipper_length* y *body_mass* no son buenos indicadores para diferenciar a las tres subespecies, dado que dos de las subespecies están demasiado mezcladas para poder diferenciar nada.

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



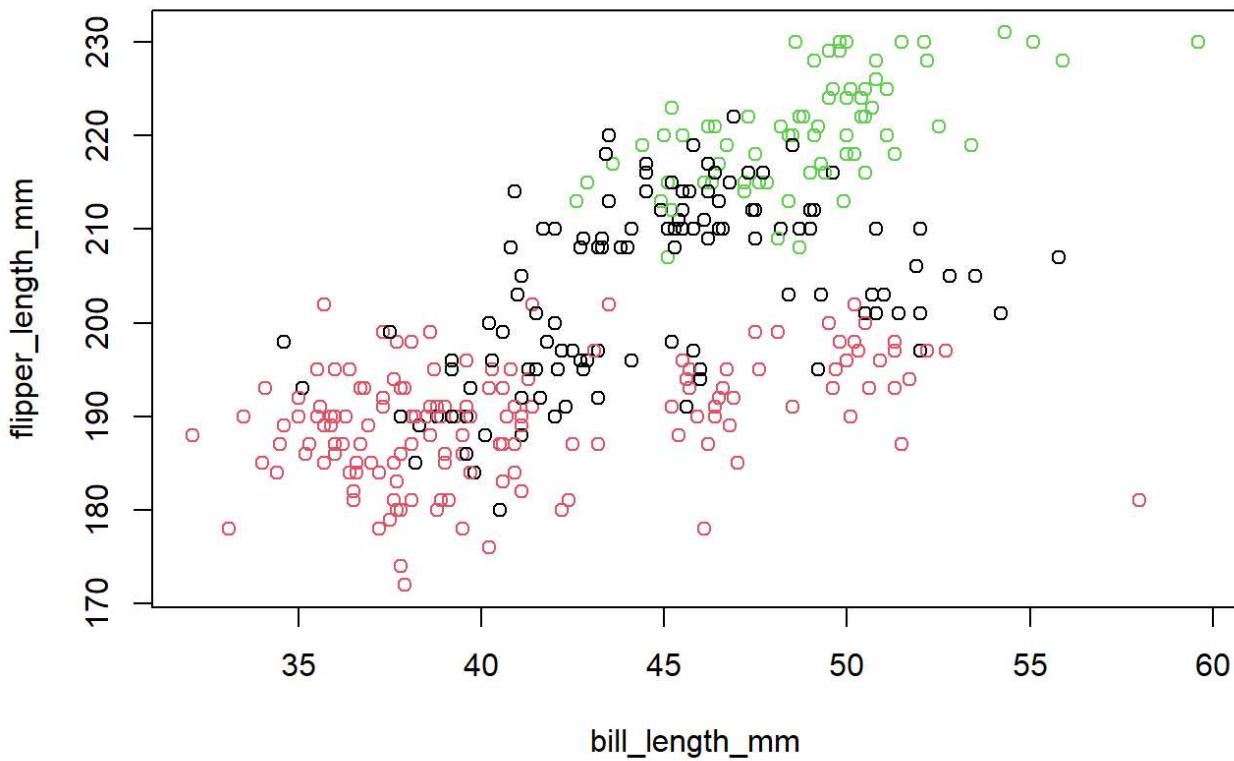
```
plot(x[c(3,4)], col=as.factor(penguins$species), main="Clasificación real")
```

Clasificación real

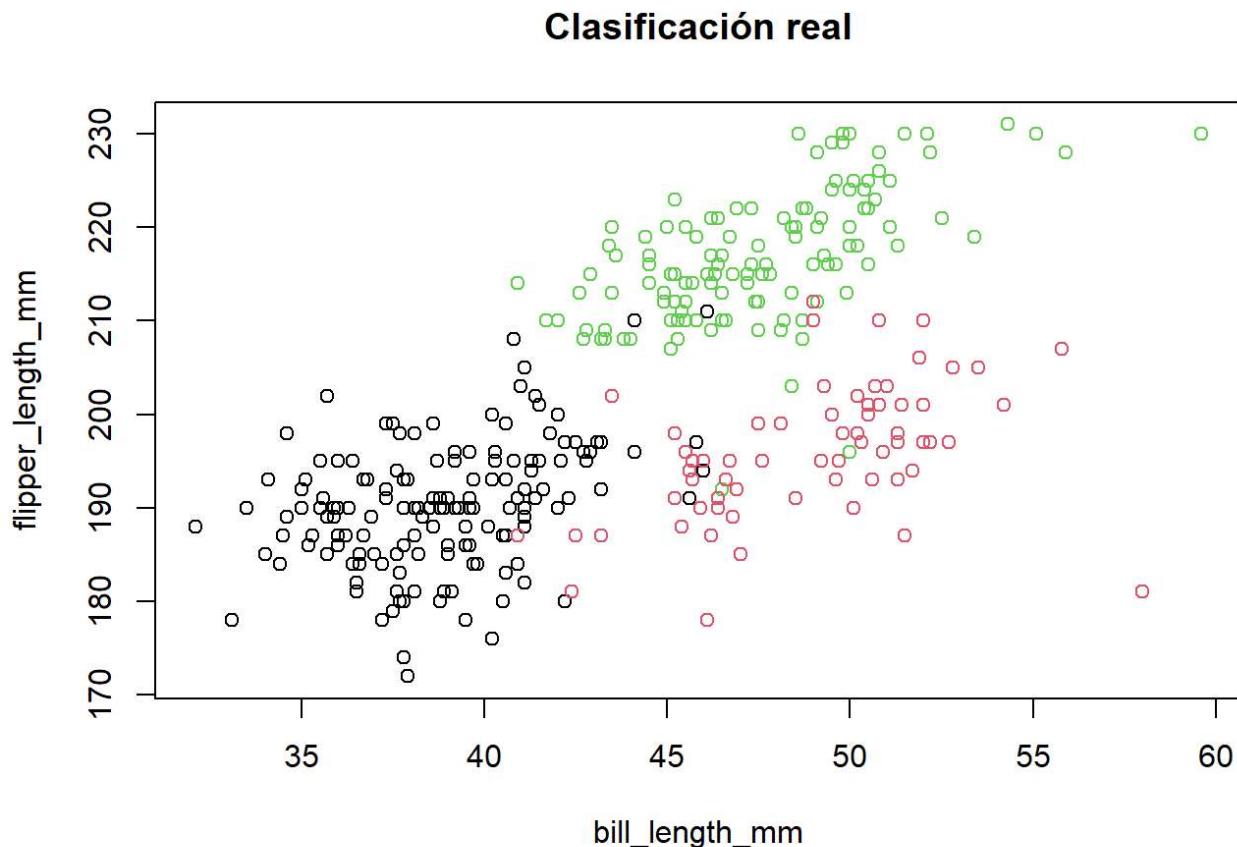


```
# bill_length y flipper_length  
plot(x[c(1,3)], col=penguins3clusters$cluster, main="Clasificación k-means")
```

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.

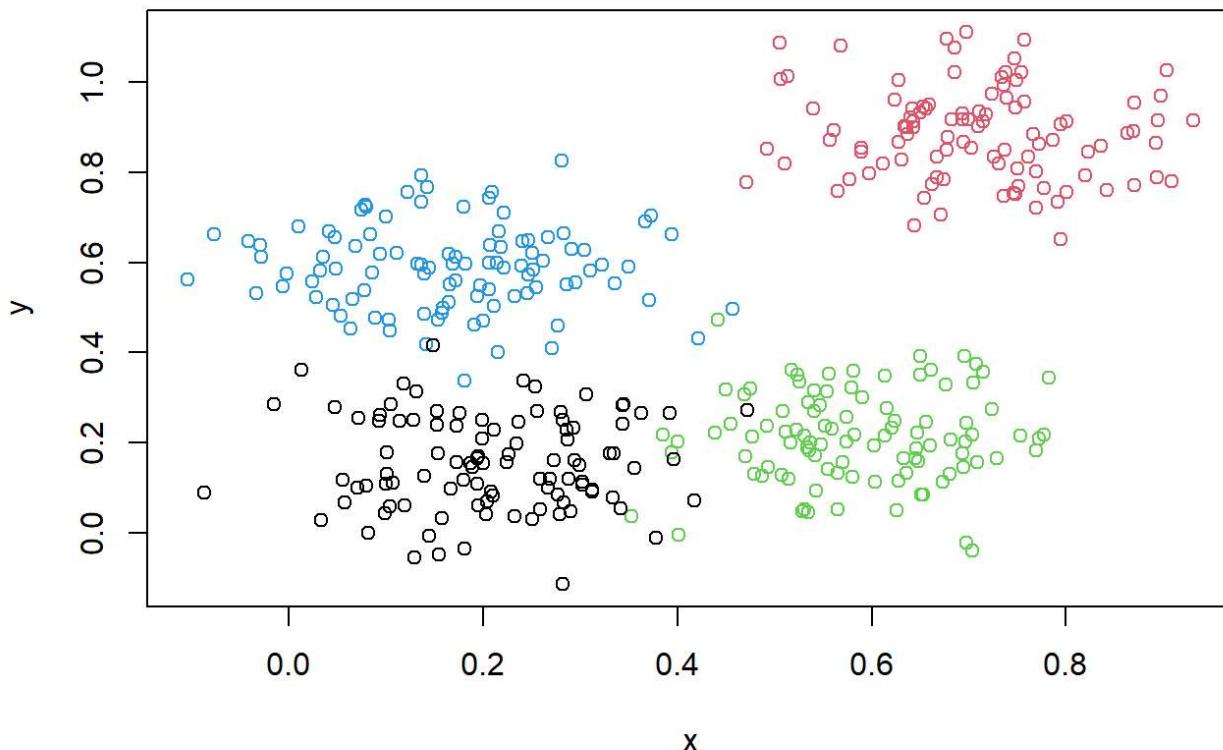
4 Ejemplo 2

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.

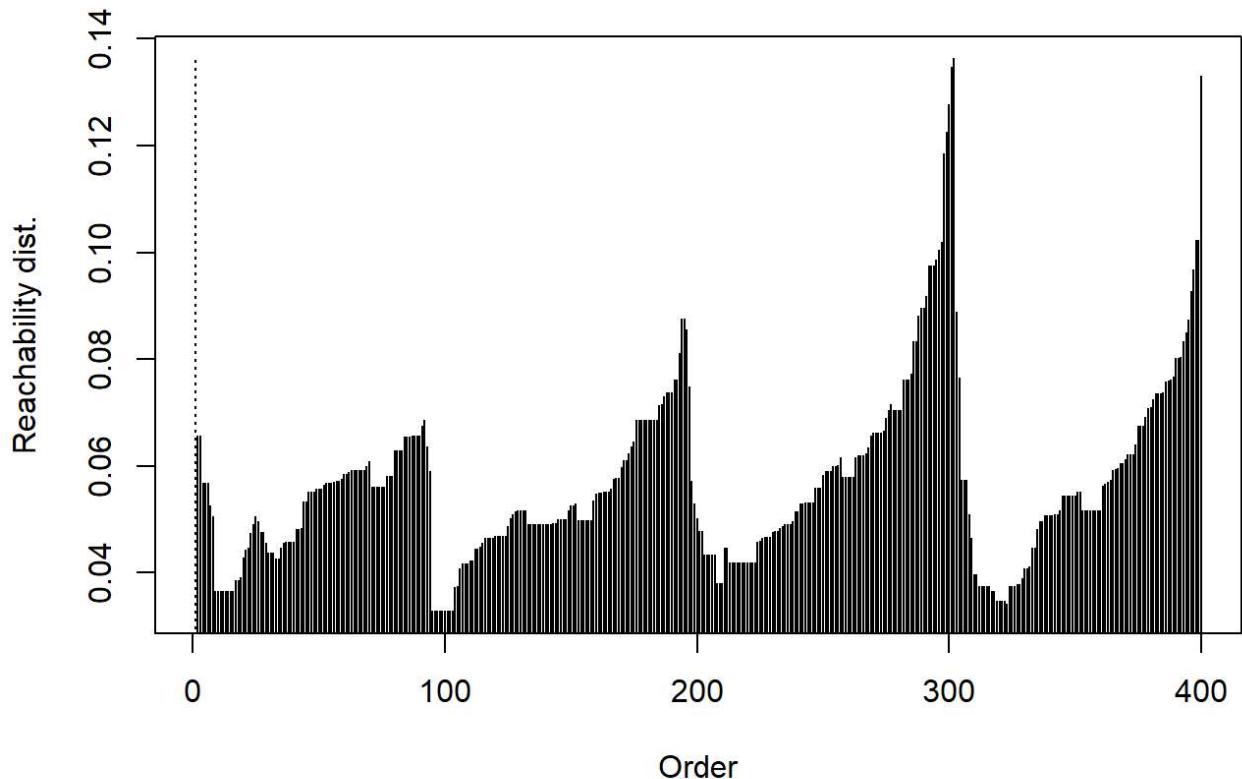
Los valles representan clusters (cuanto más profundo es el valle, más denso es el cluster), mientras que las cimas indican los puntos que están entre las agrupaciones (estos puntos son candidatos a ser considerados *outliers*)

```

### Gráfica de alcanzabilidad
plot(res)

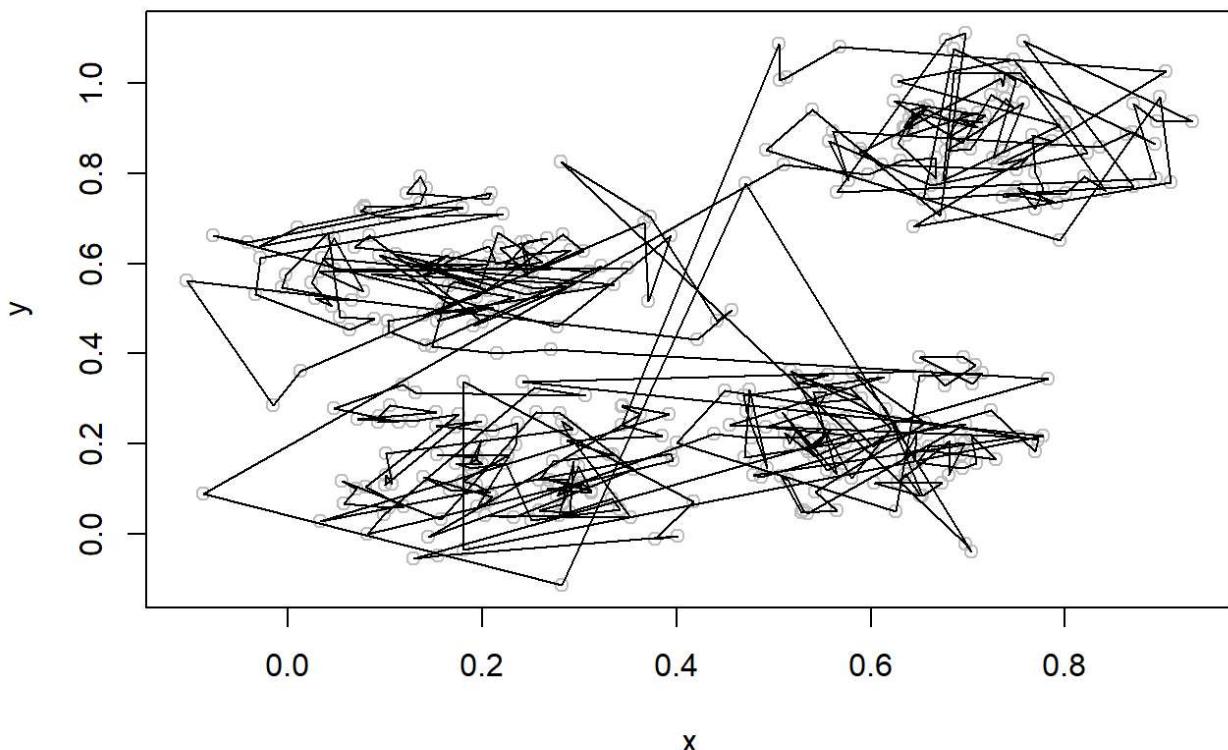
```

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,])
```



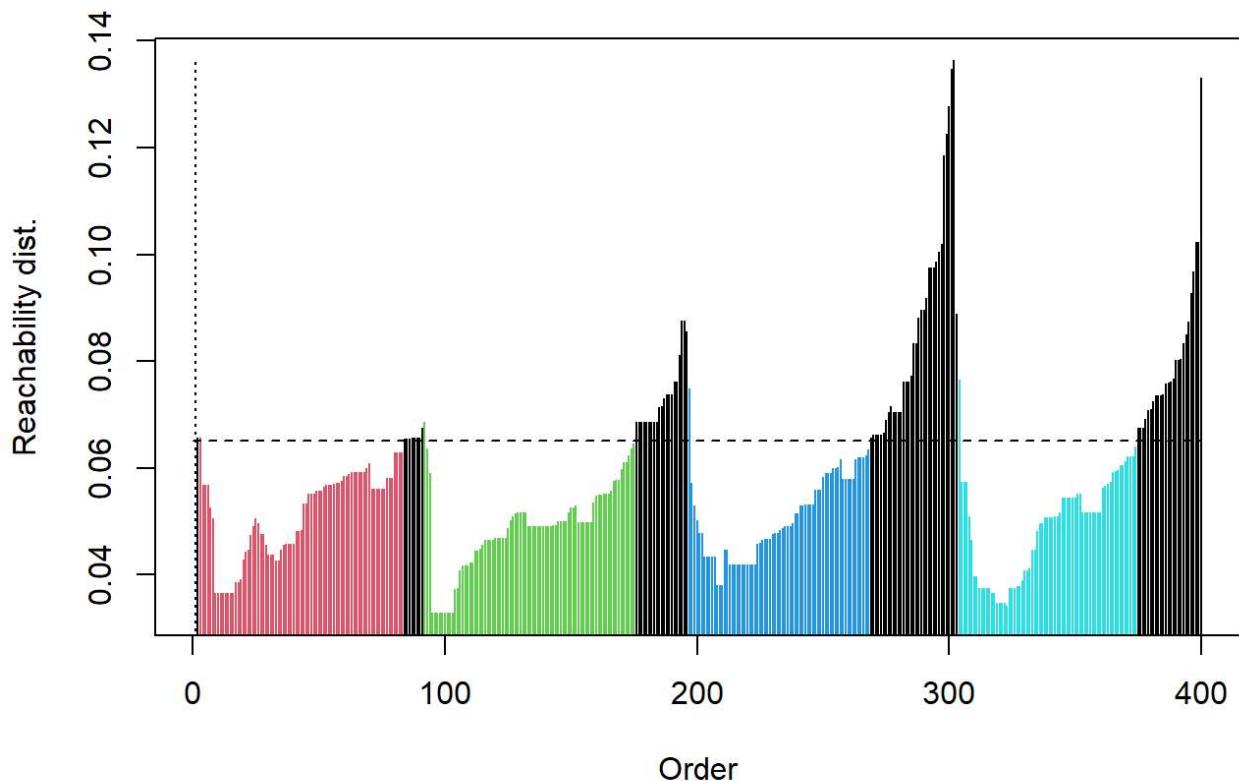
Otro ejercicio interesante a realizar es extraer una agrupación de la ordenación realizada por OPTICS similar a lo que DBSCAN hubiera generado estableciendo el parámetro eps en eps_cl = 0.065

```
### 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

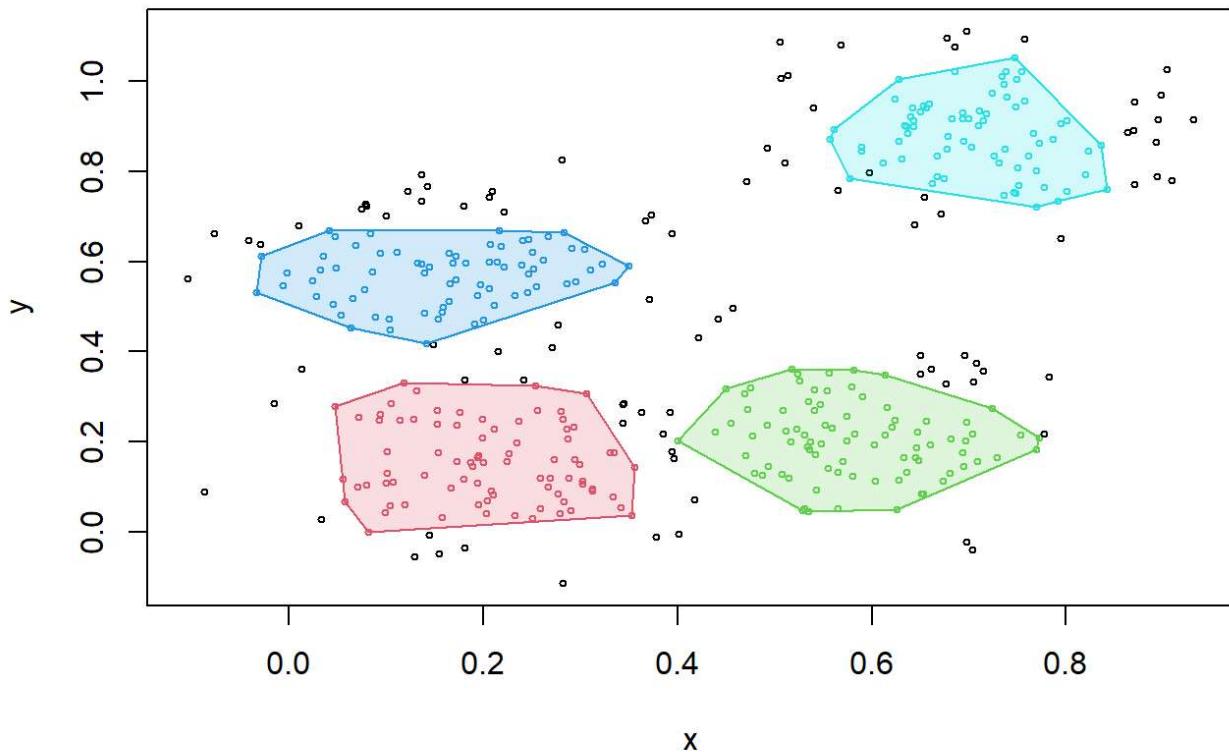


Observamos en el gráfico anterior como se han coloreado los 4 clusters y en negro se mantienen los valores *outliers*.

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

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

Convex Cluster Hulls



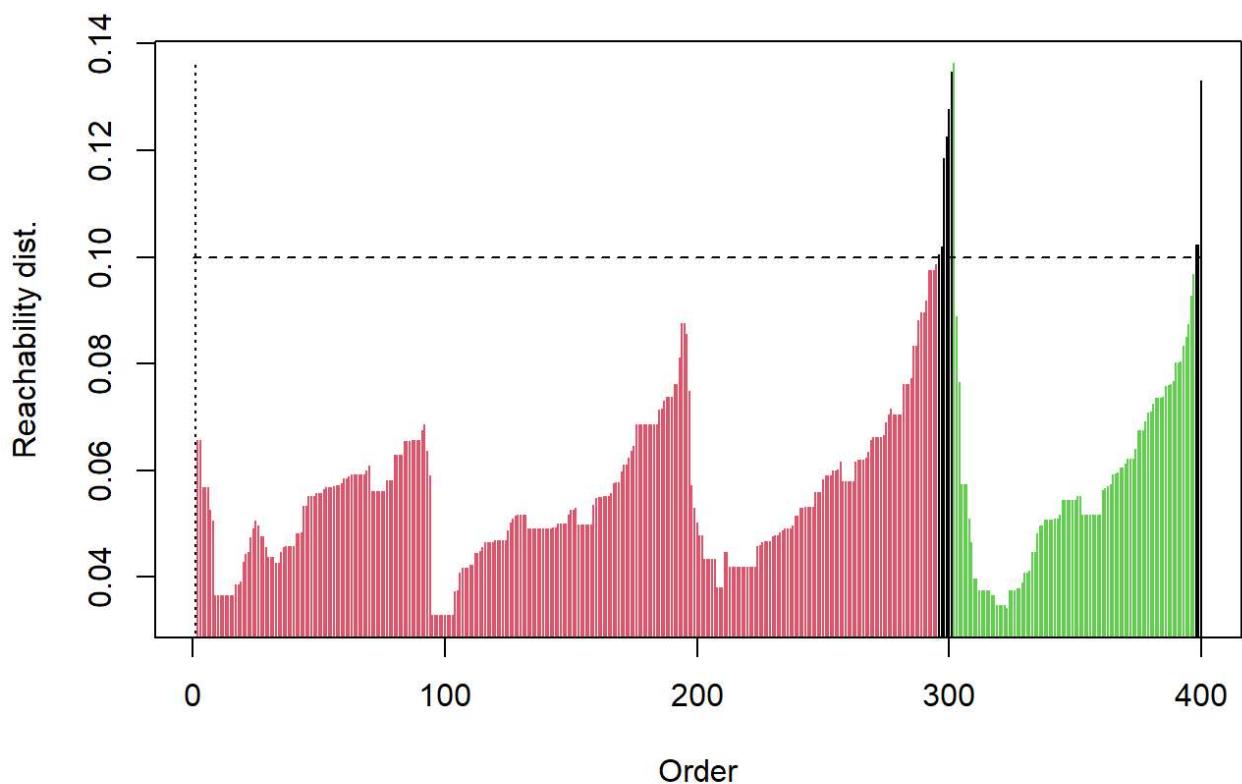
Repetimos el experimento anterior incrementando el parámetro *epc_c*, veamos como el efecto que produce es la concentración de clusters ya que flexibilizamos la condición de densidad.

```
### 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
```

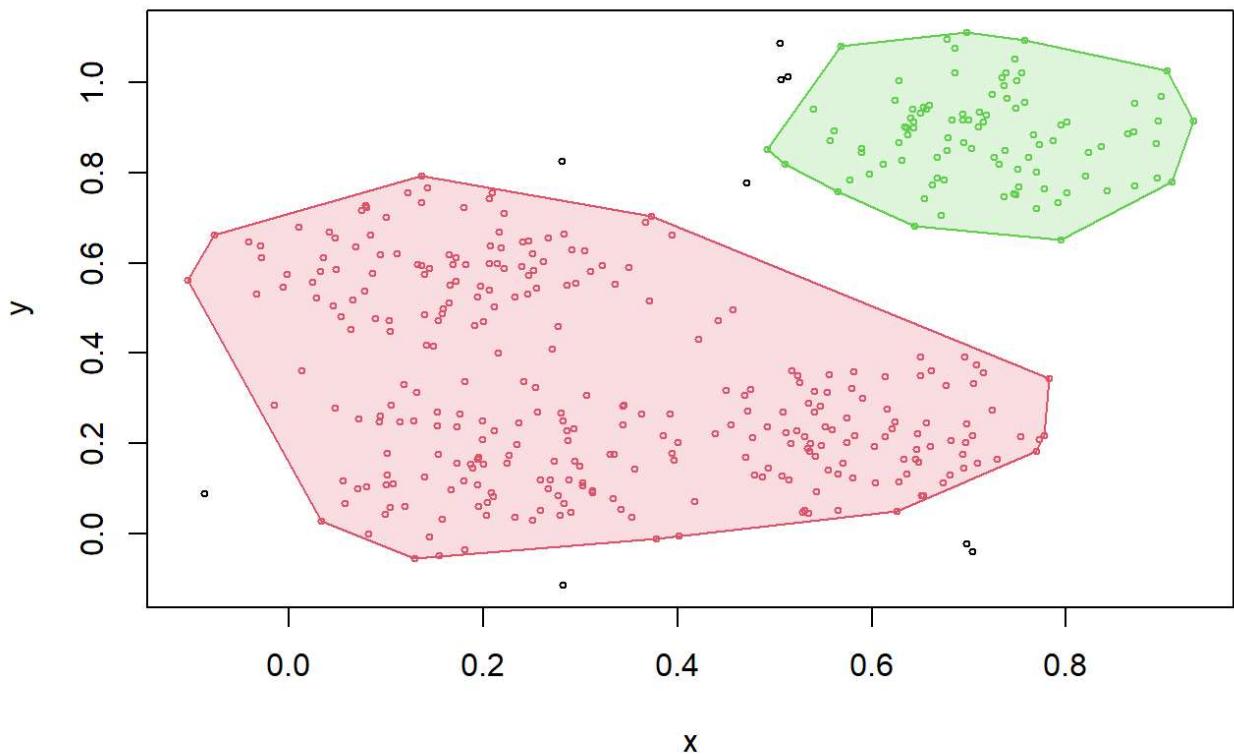
```
plot(res)
```

Reachability Plot



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

Convex Cluster Hulls

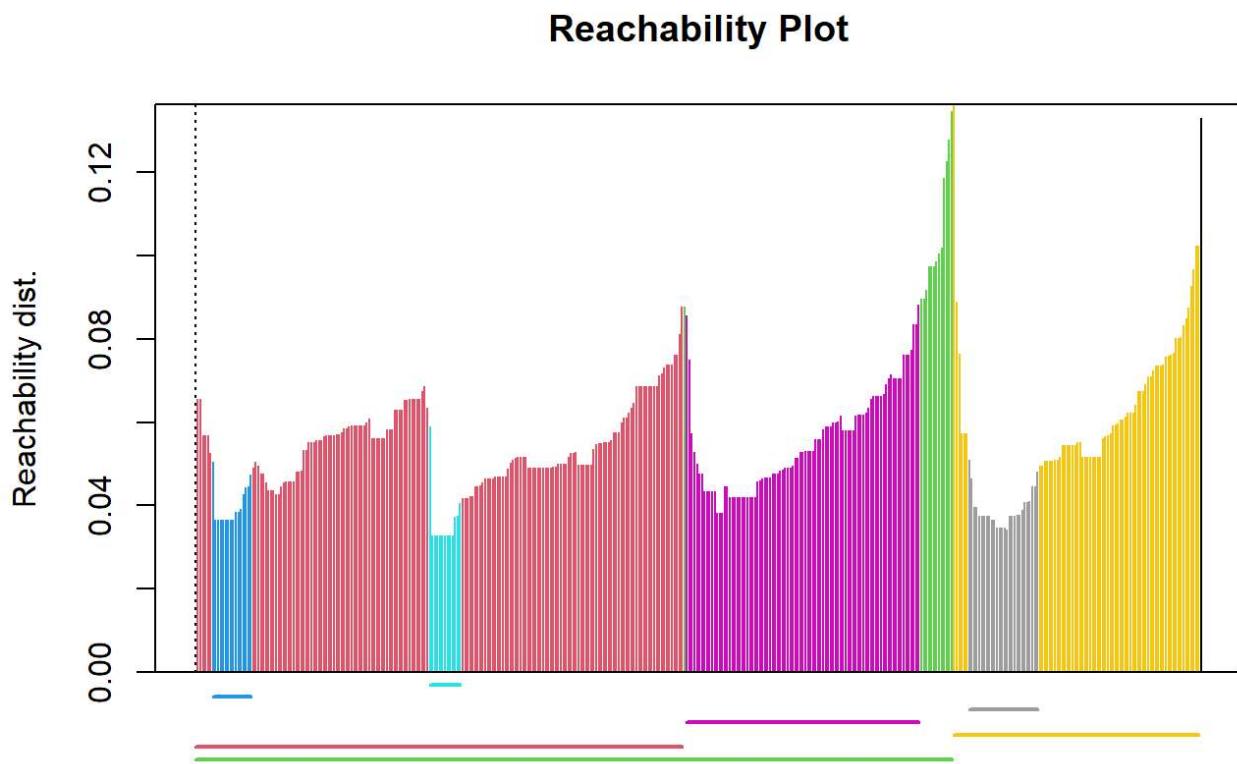


Veamos ahora una variante de la extracción **DBSCAN** anterior. En ella el parámetro ξ nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

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

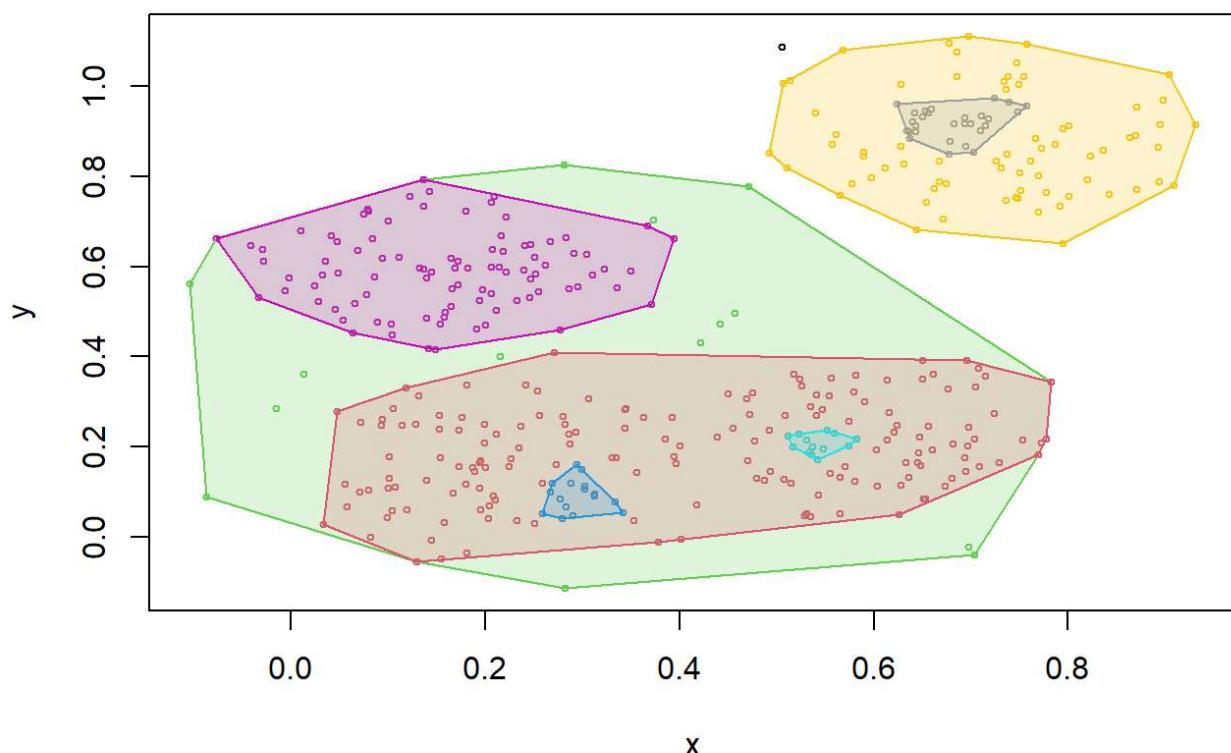
```
## OPTICS ordering/clustering for 400 objects.
## Parameters: minPts = 10, eps = 0.193786846197958, eps_cl = NA, xi = 0.05
## The clustering contains 7 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster, clusters_xi
```

```
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

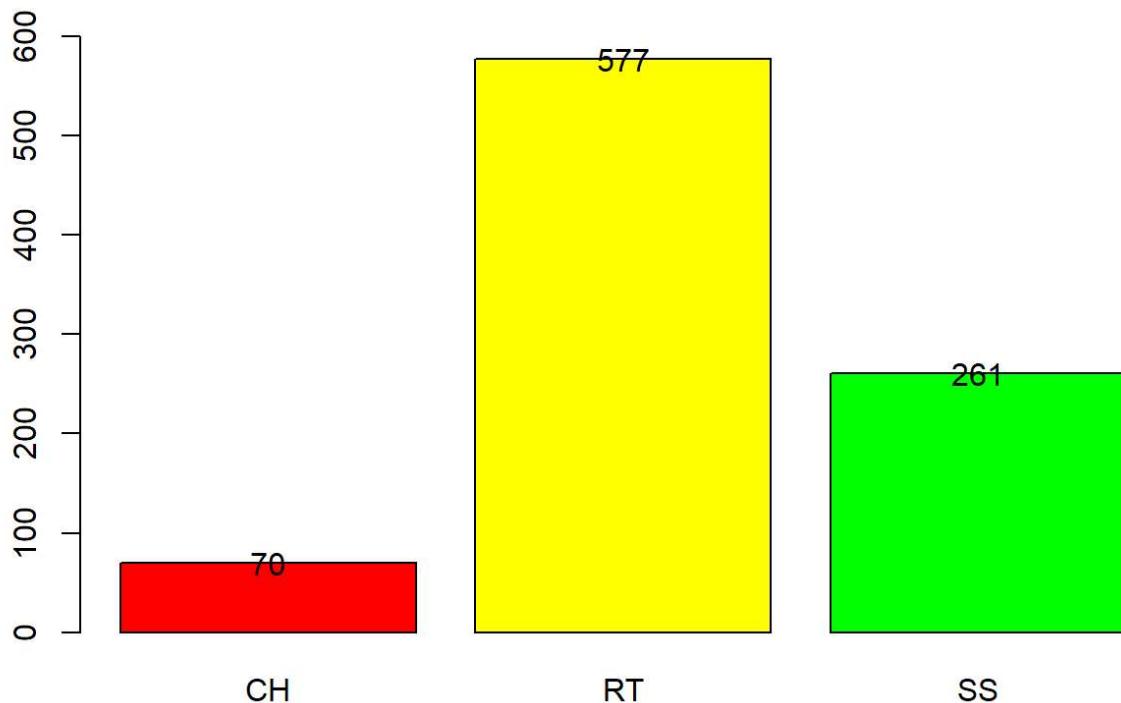
Los campos de la BBDD son:

-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

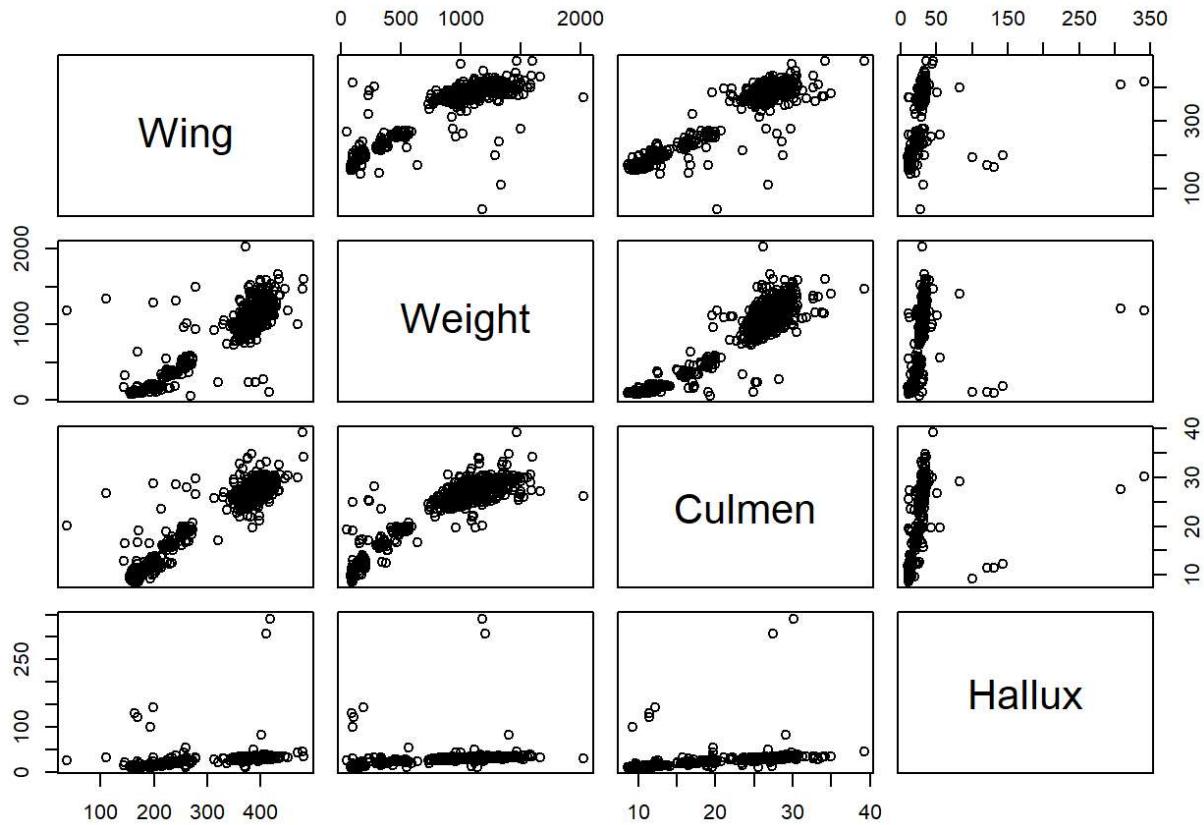
```
Hawks_data <- na.omit(Hawks[c('Wing', 'Weight', 'Culmen', 'Hallux')])
```

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)
```

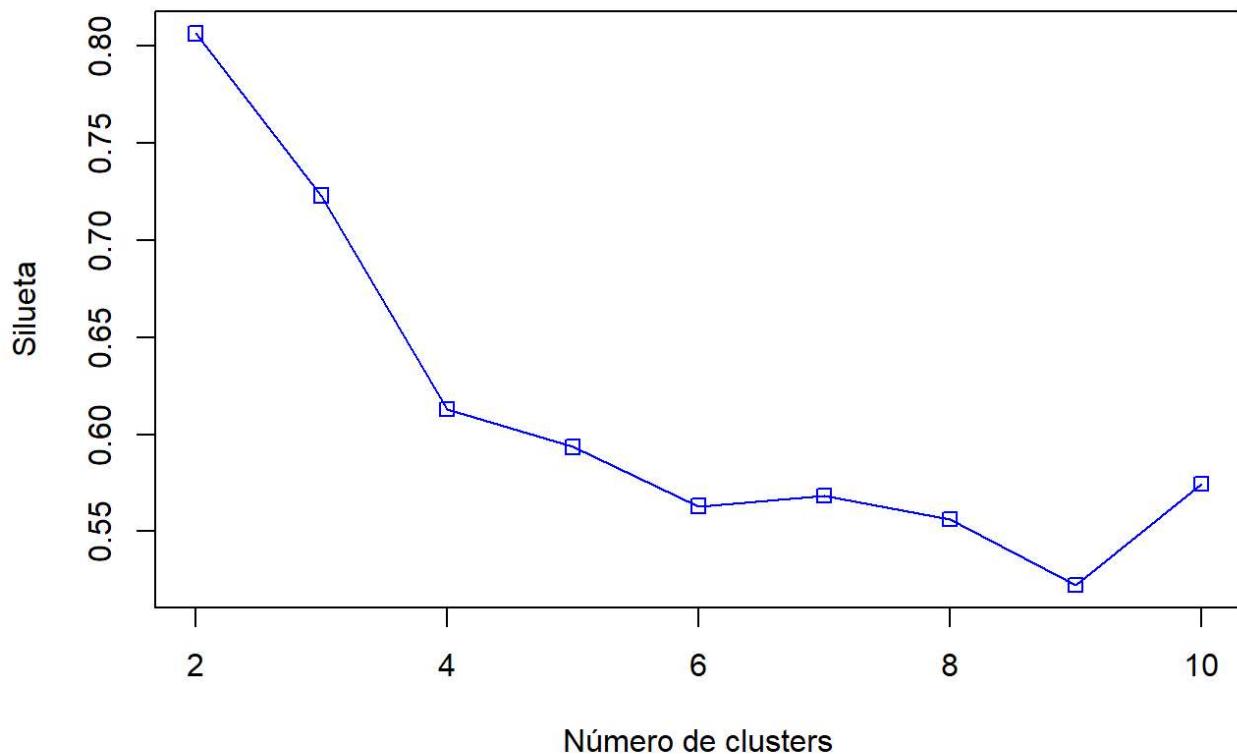


Como inicialmente no conocemos el número óptimo de clústers, probamos con varios valores

```
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")
```



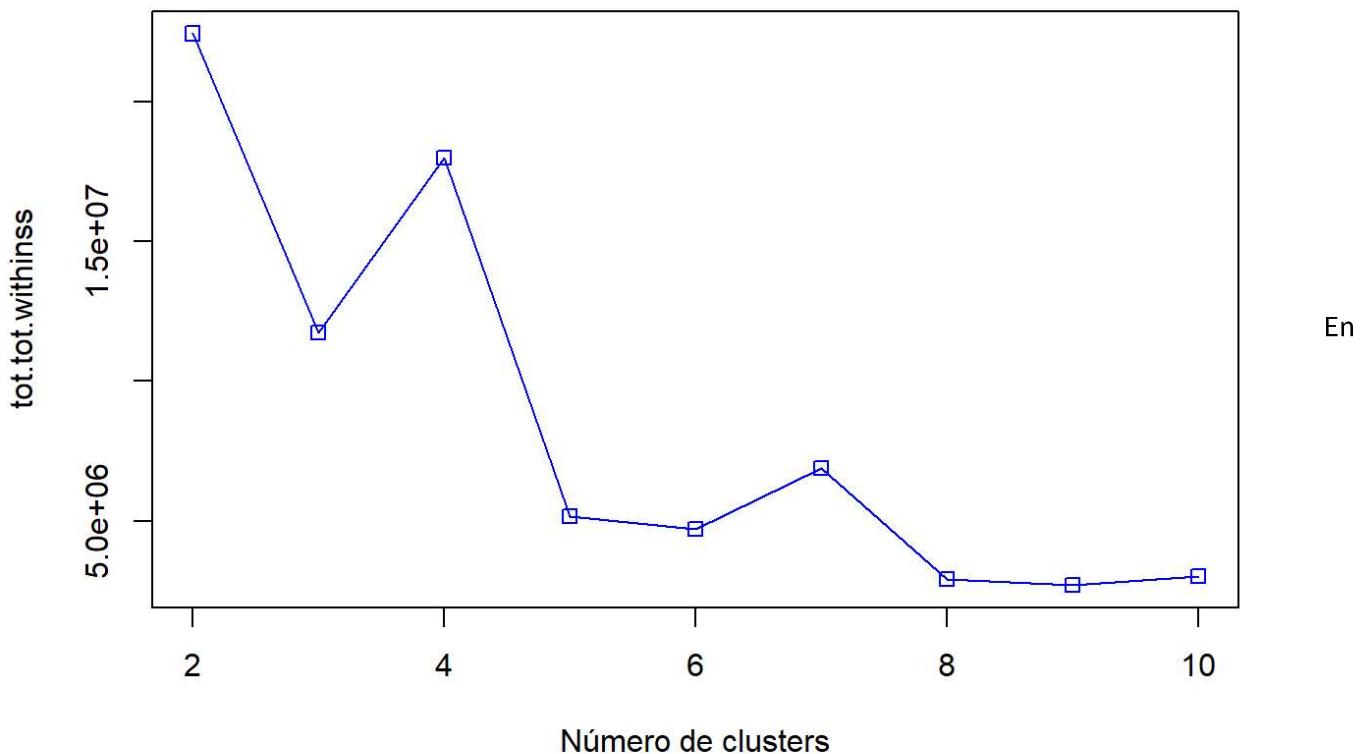
Como se puede comprobar de manera visual, la mejora mas significativa se obtiene para K = 3.

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
```

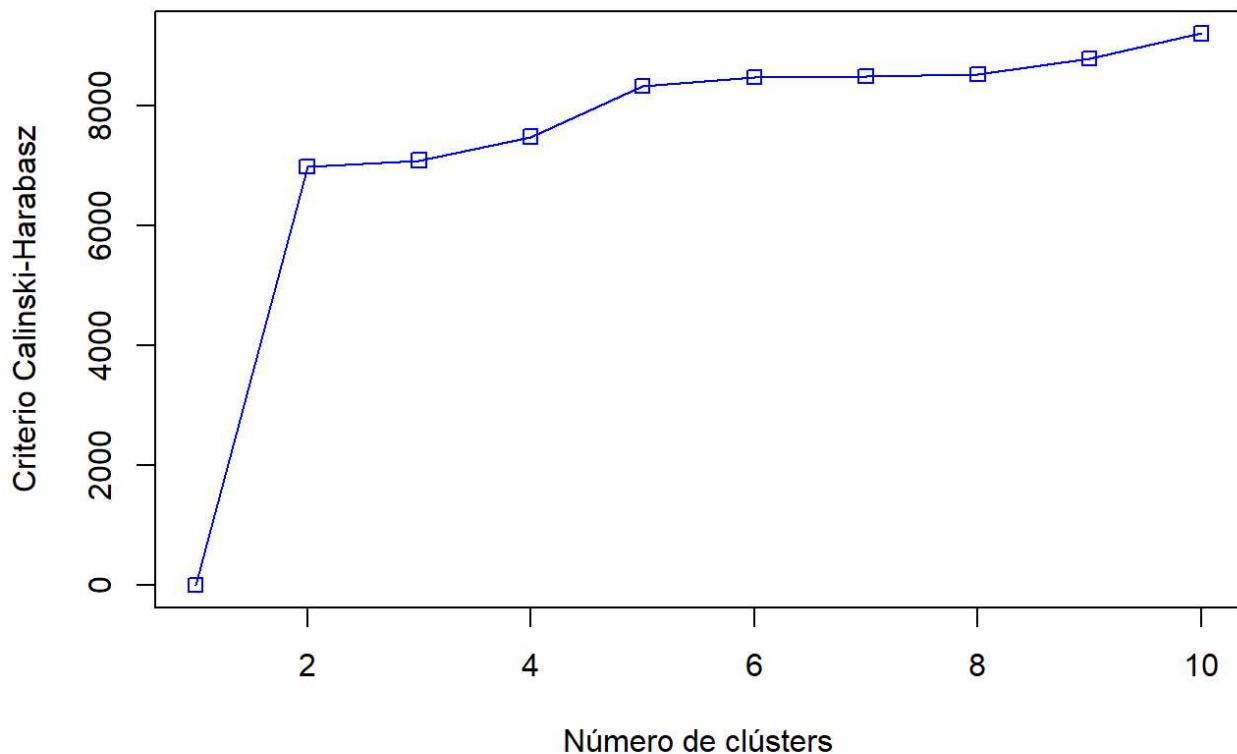
```
## [1] 10
```

```
fit_asw$bestk
```

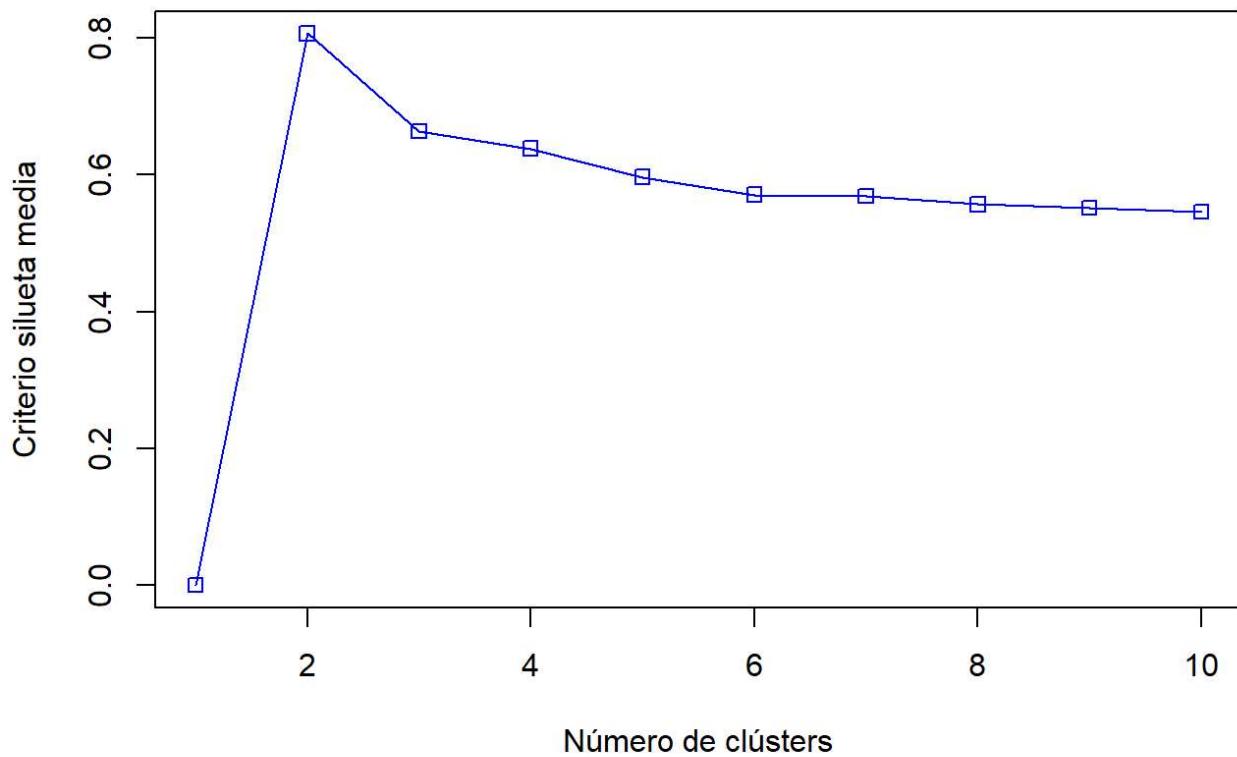
```
## [1] 2
```

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")
```



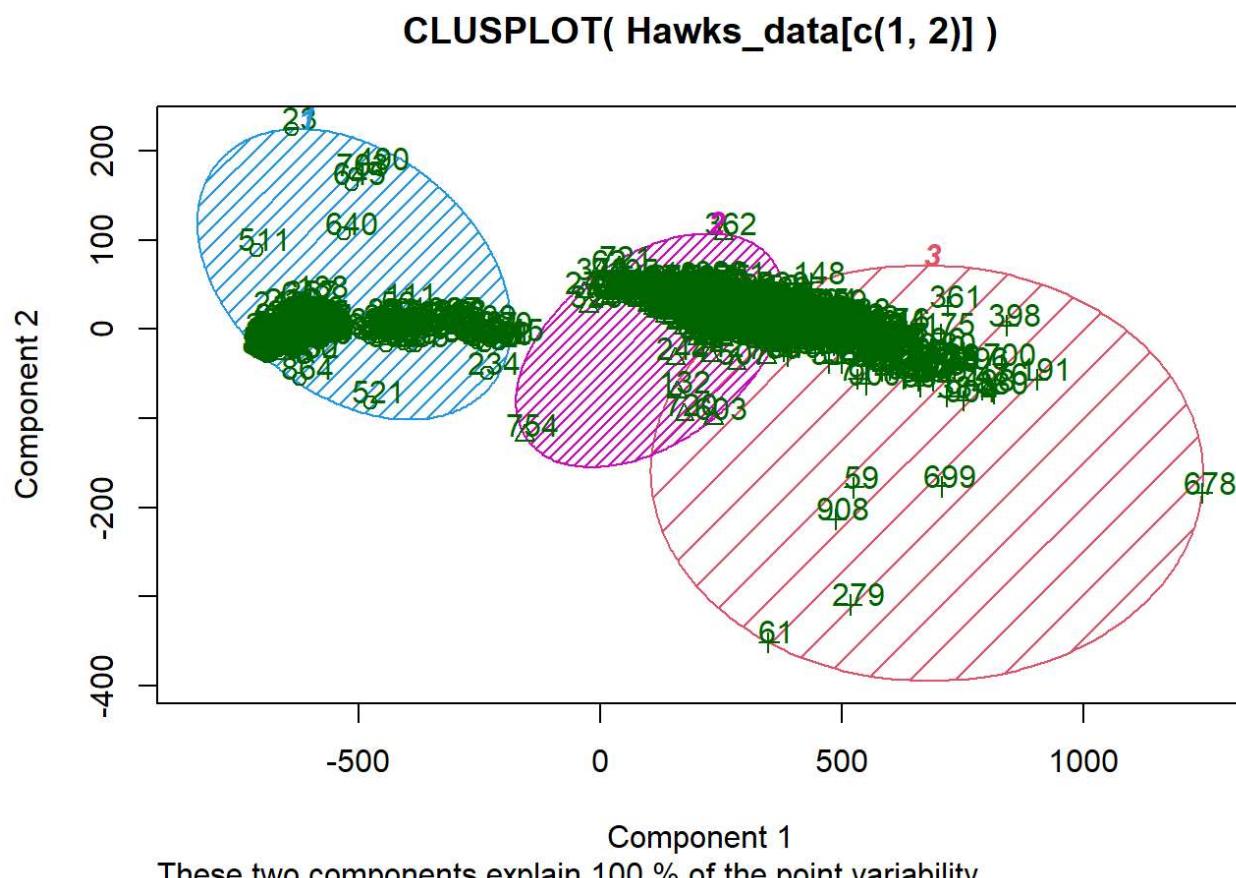
Los resultados son muy parecidos a los que hemos obtenido anteriormente. Con el criterio de la silueta media se obtienen 3 clústers y con el Calinski-Harabasz se obtienen 5.

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)
```

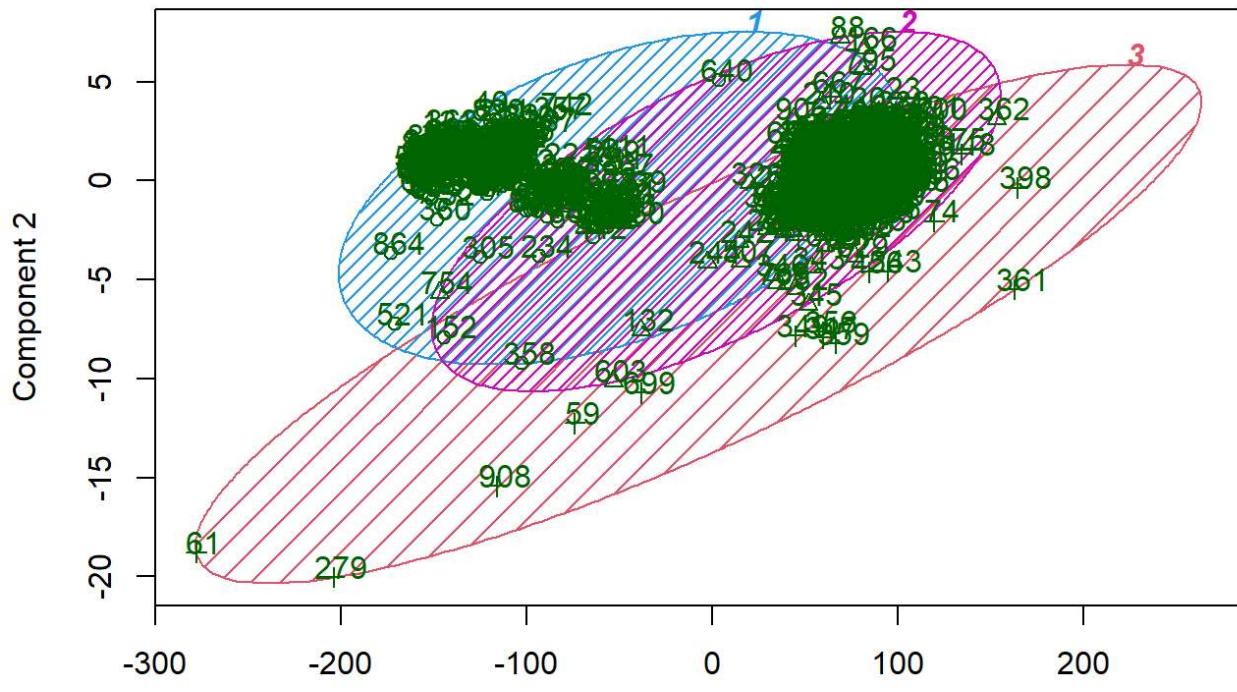
Visualizamos la agrupación con 3 clústers y combinando los diferentes campos

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



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

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

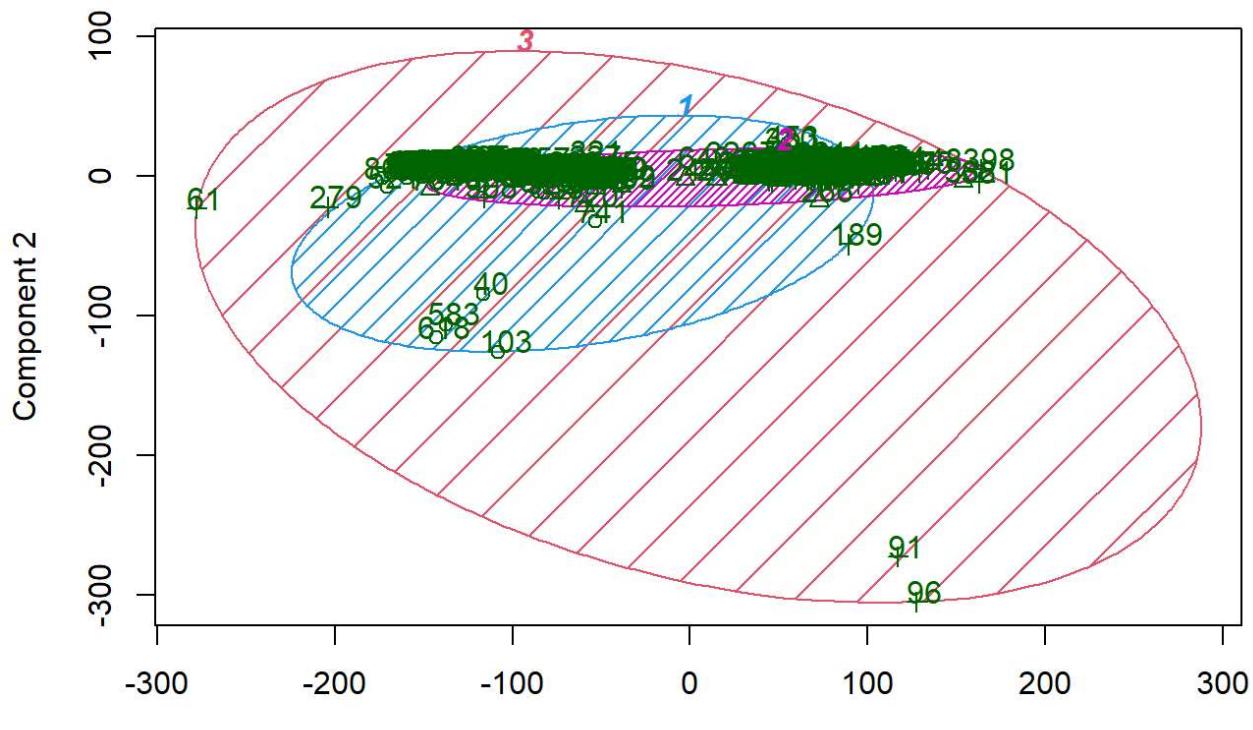


Component 1

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)
```

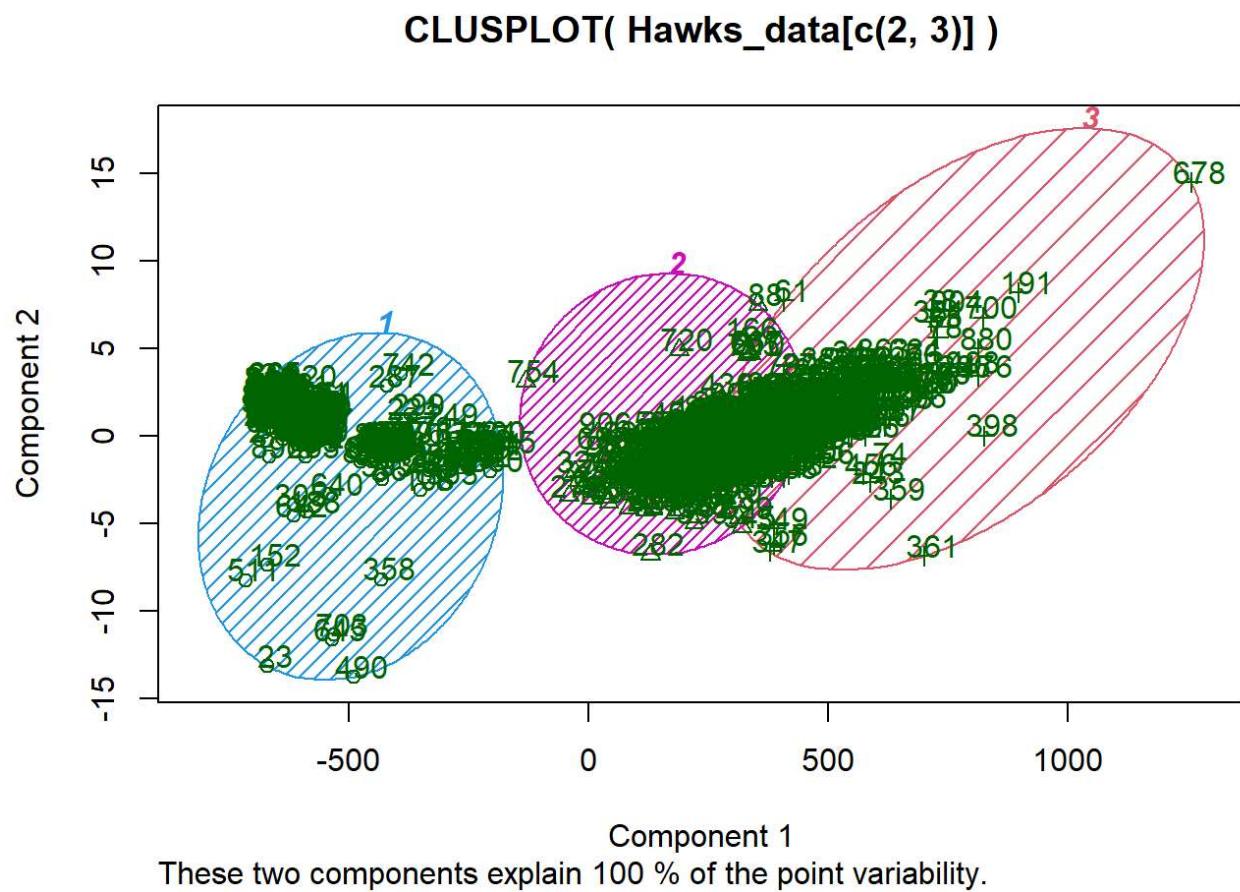
CLUSPLOT(Hawks_data[c(1, 4)])



Component 1

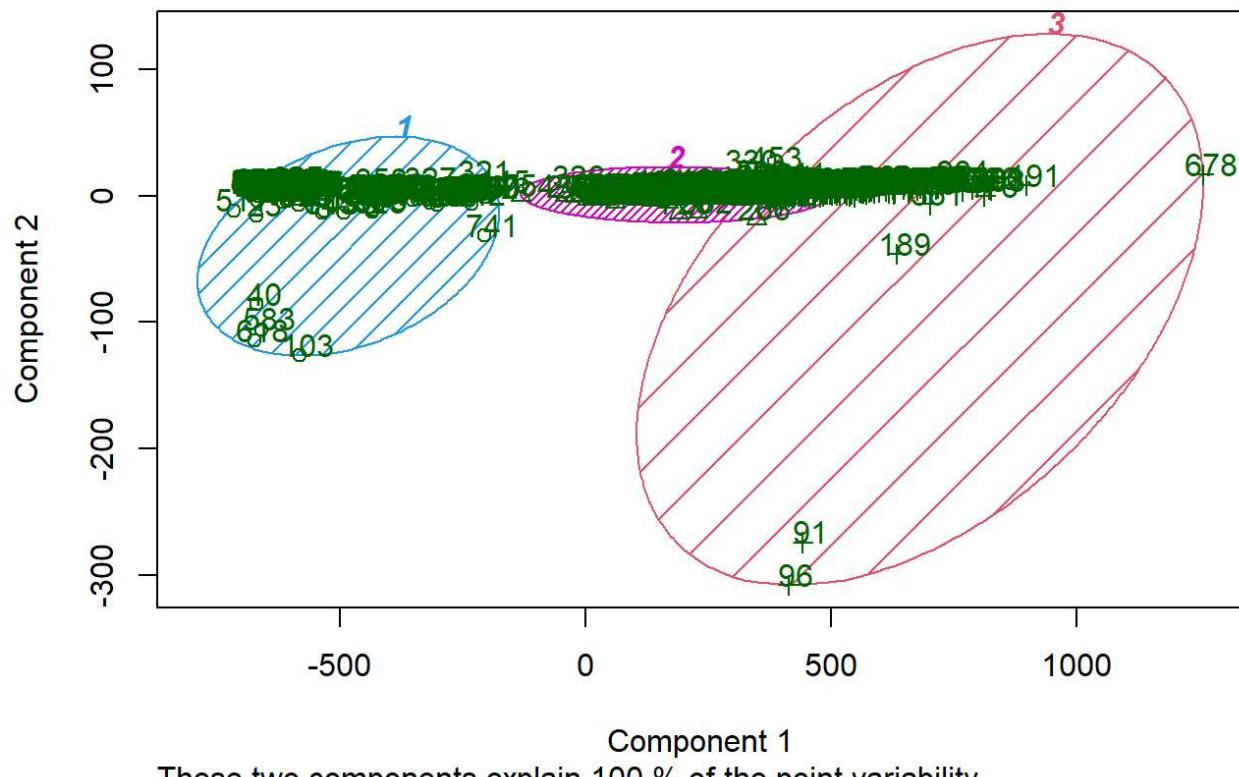
These two components explain 100 % of the point variability.

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

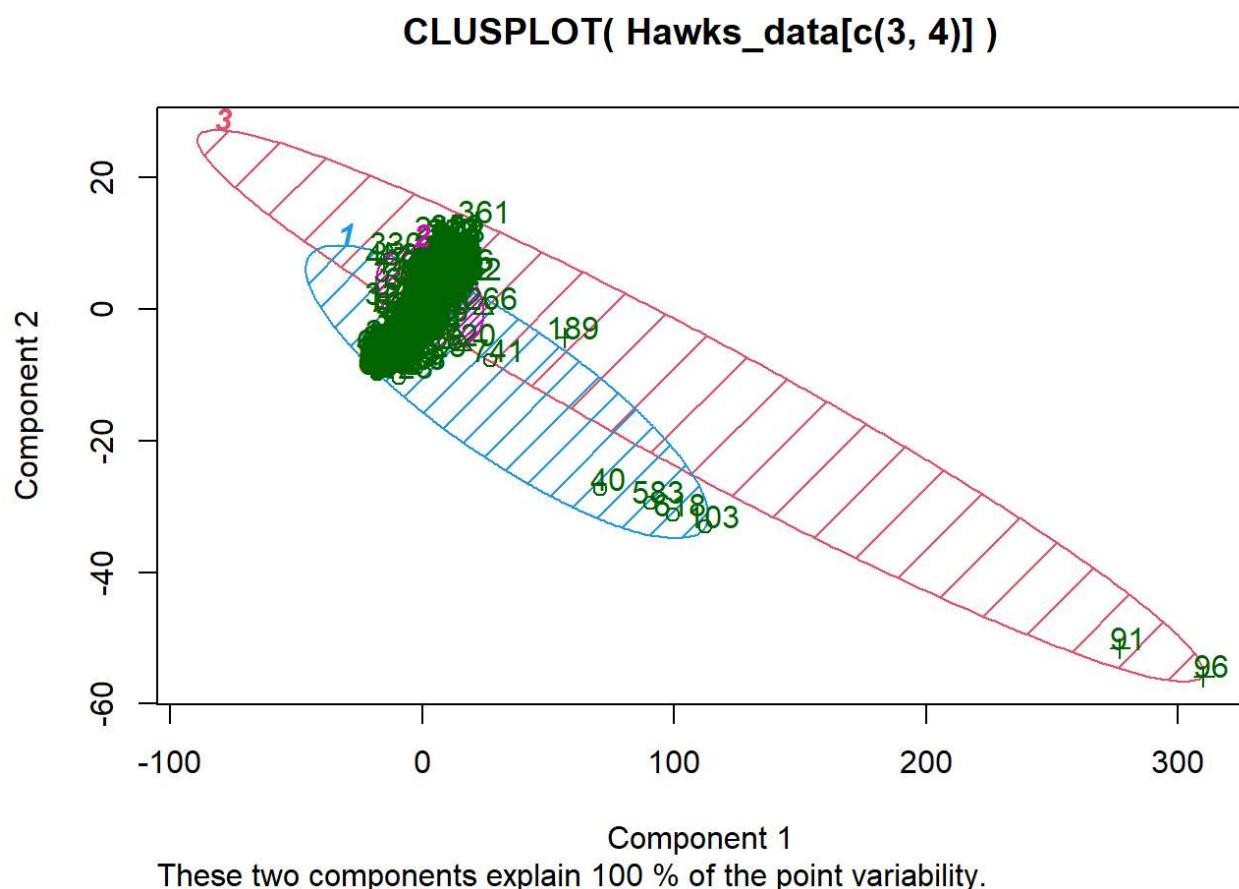


```
clusplot(Hawks_data[c(2,4)], Hawks3clusters$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```

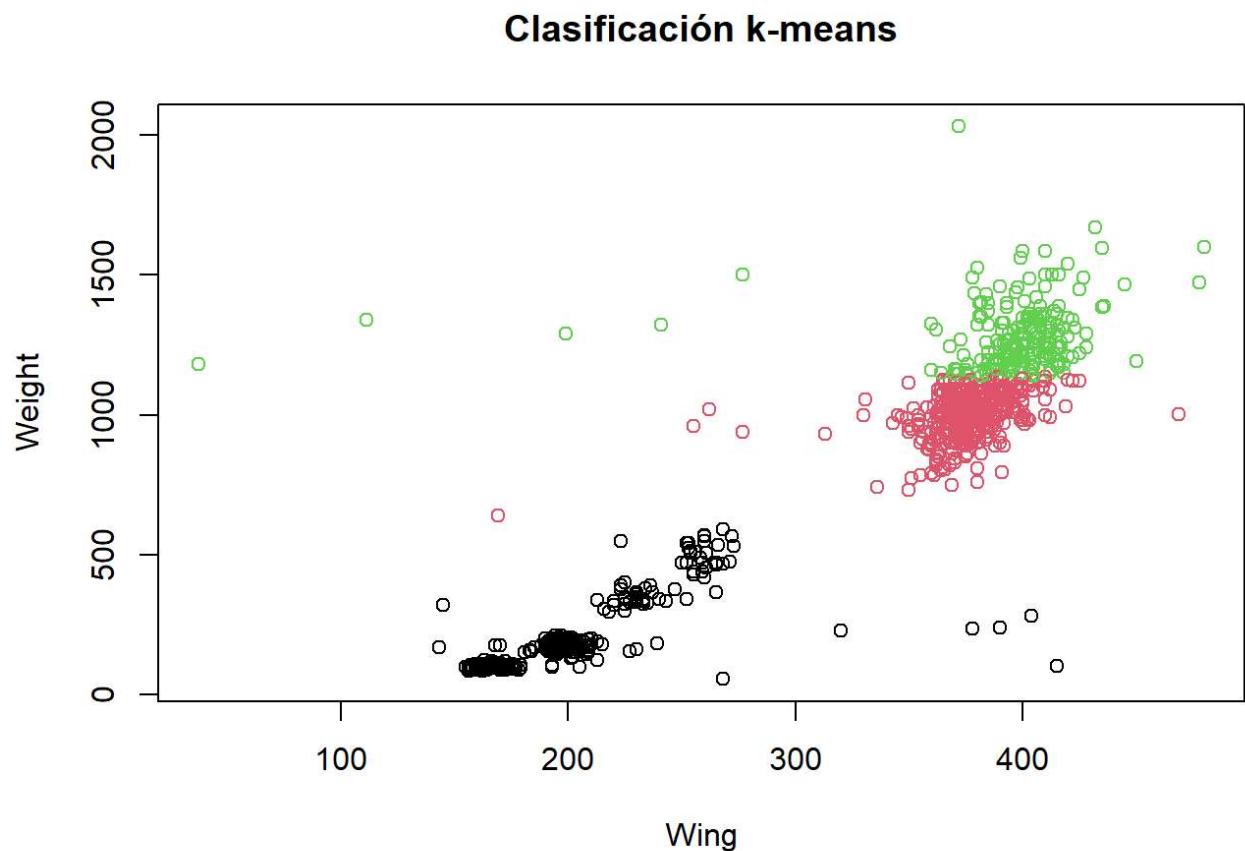
CLUSPLOT(Hawks_data[c(2, 4)])



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

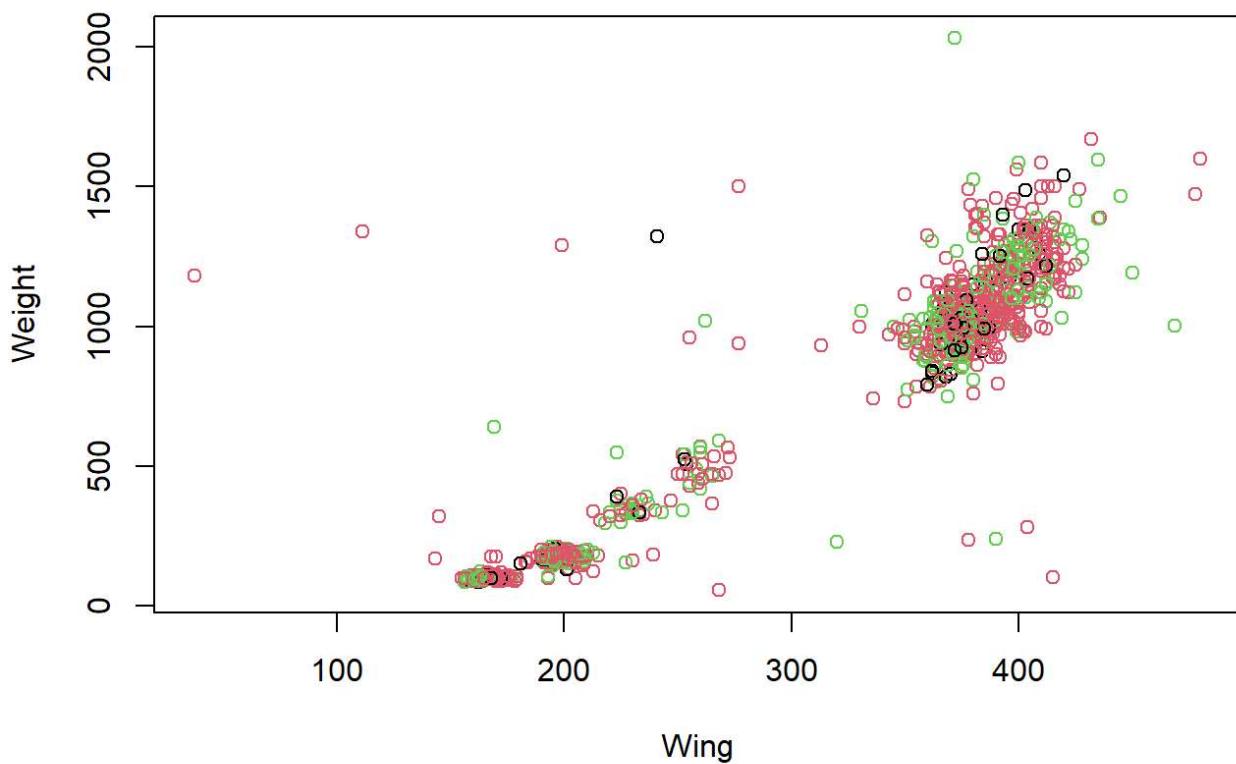


```
plot(Hawks_data[c(1,2)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```



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

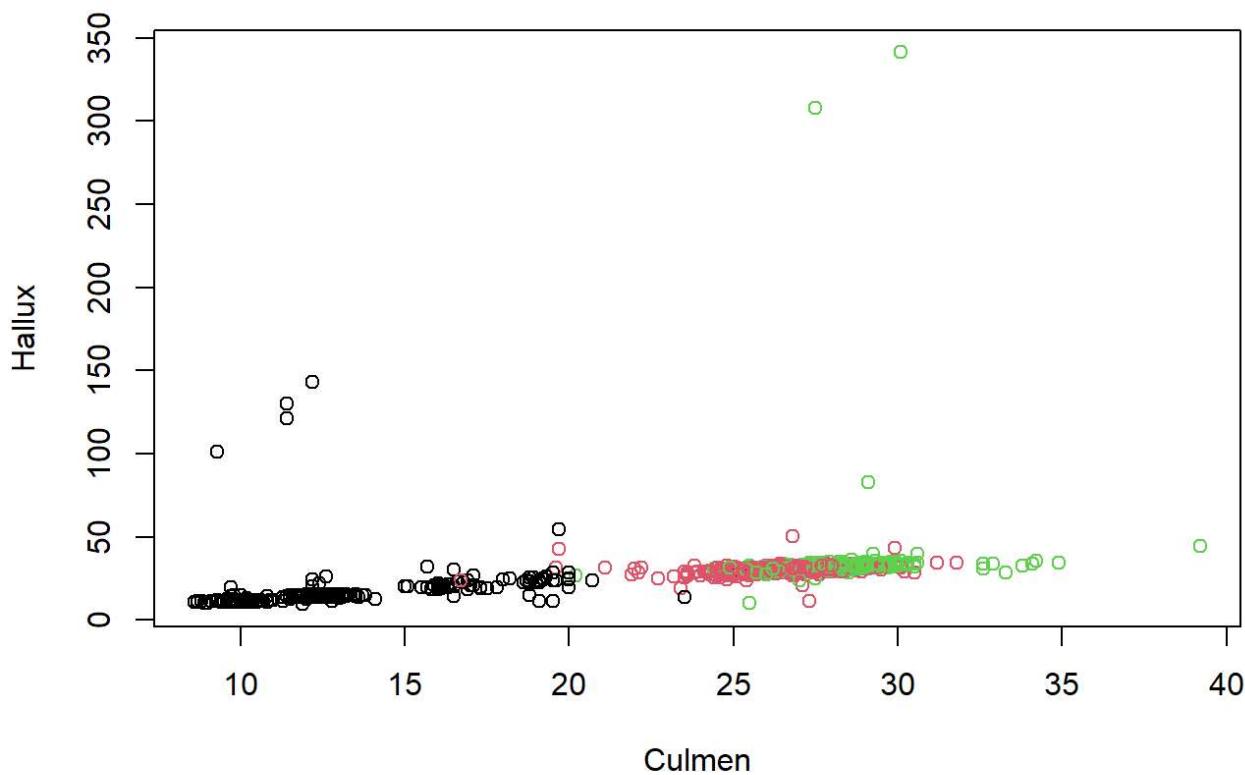
Clasificación real



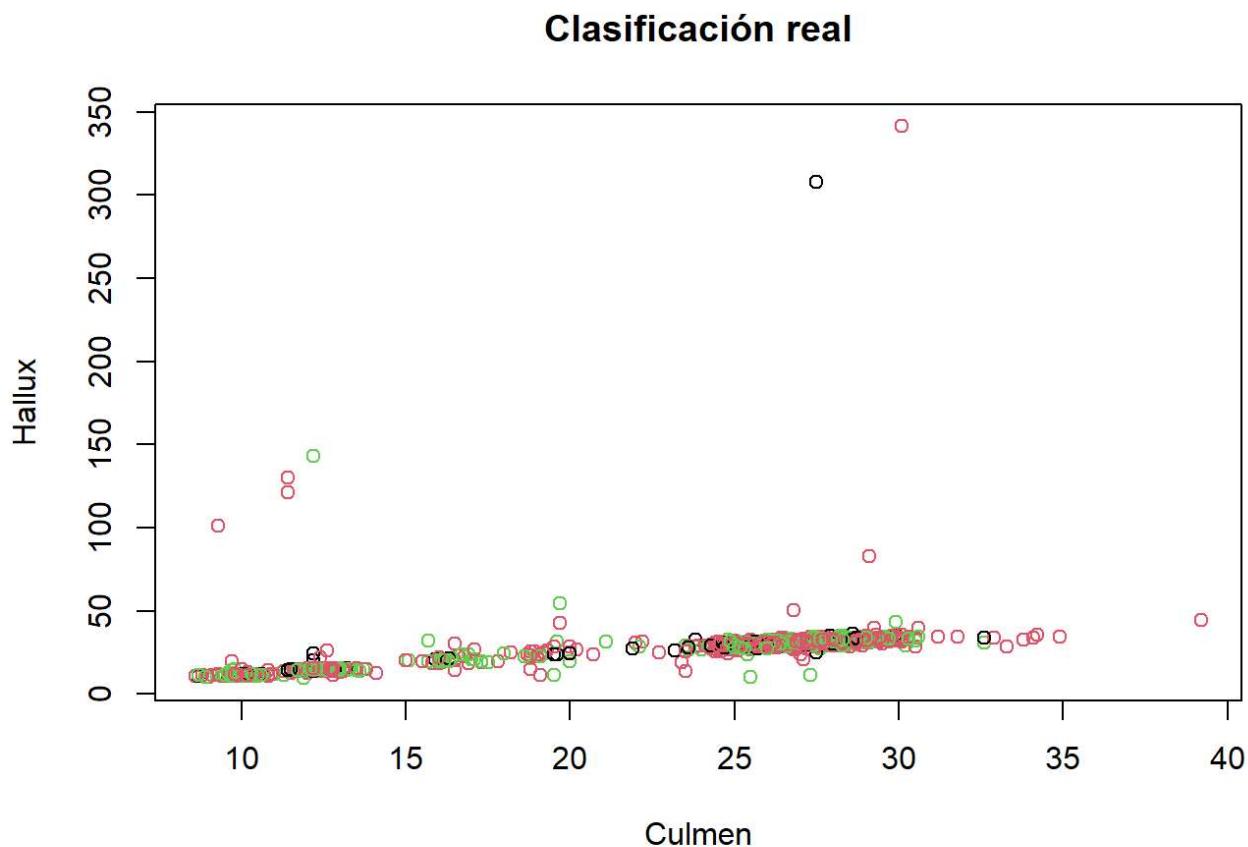
Podemos observar que Weight y Wing no son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

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

Clasificación k-means



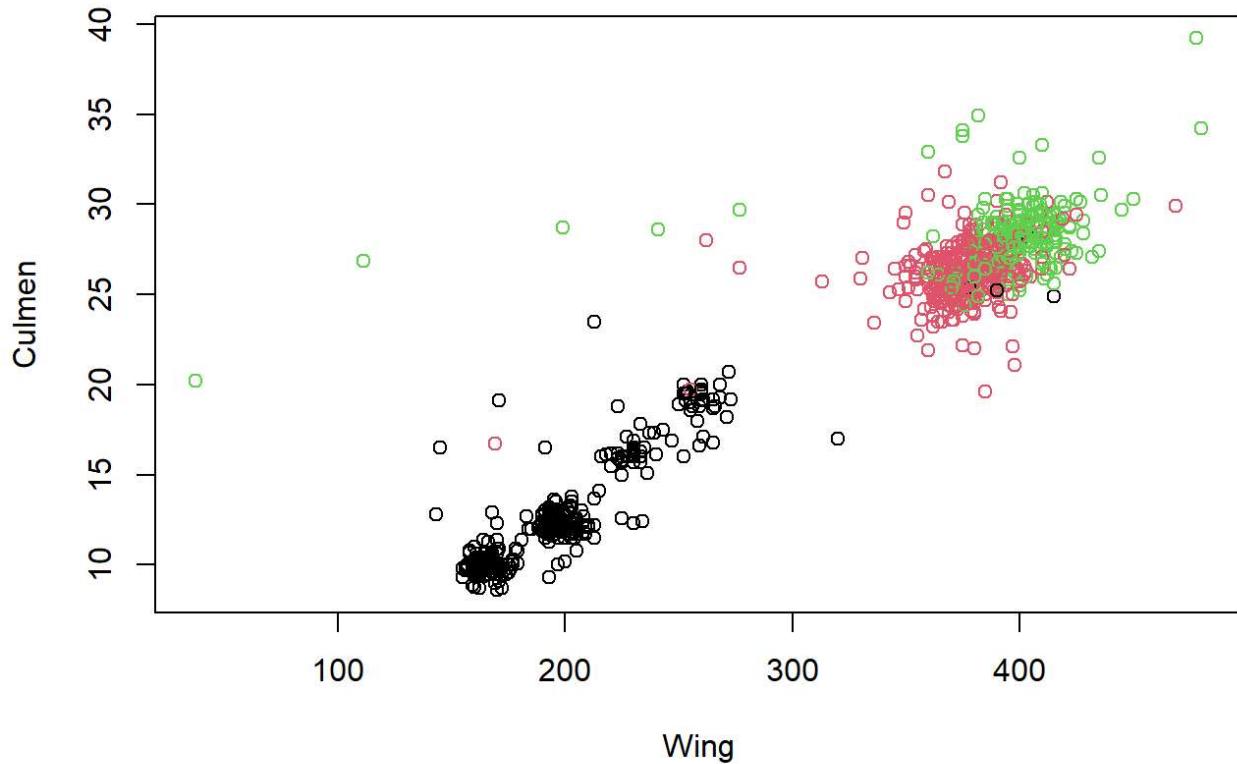
```
plot(Hawks_data[c(3,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```



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.

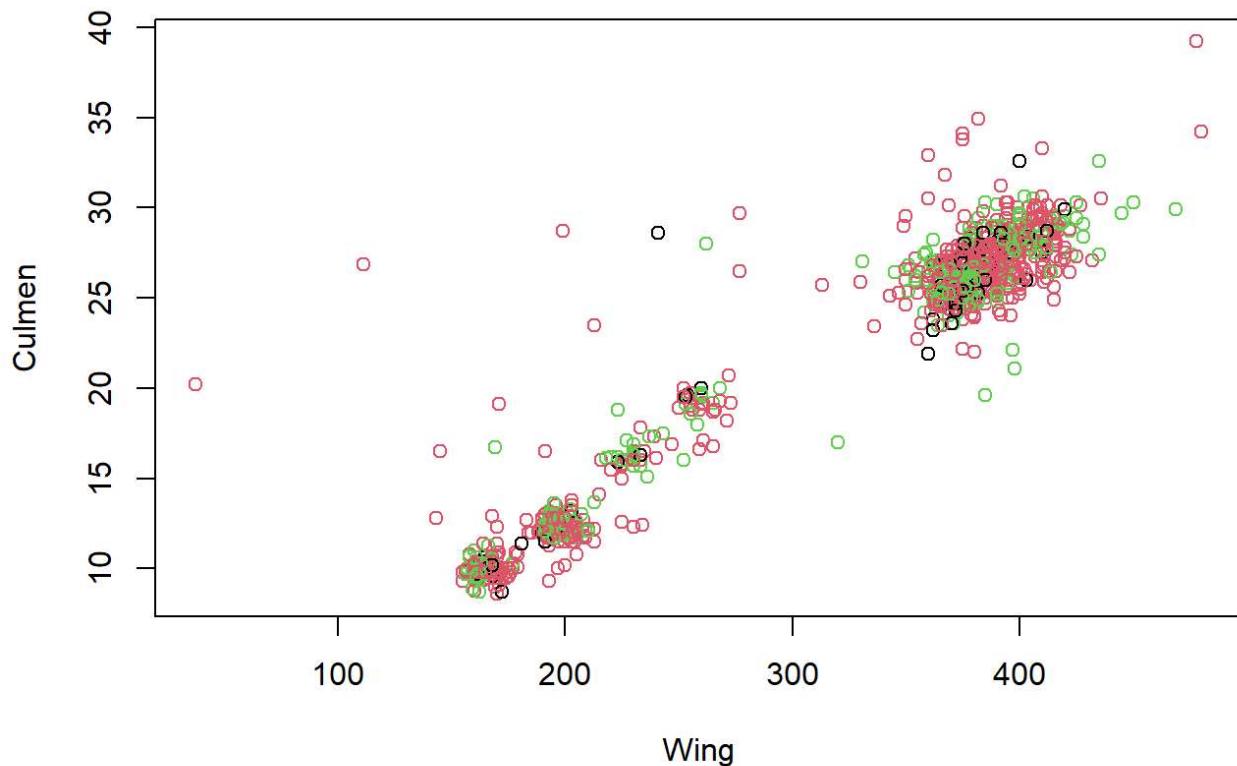
```
plot(Hawks_data[c(1,3)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means



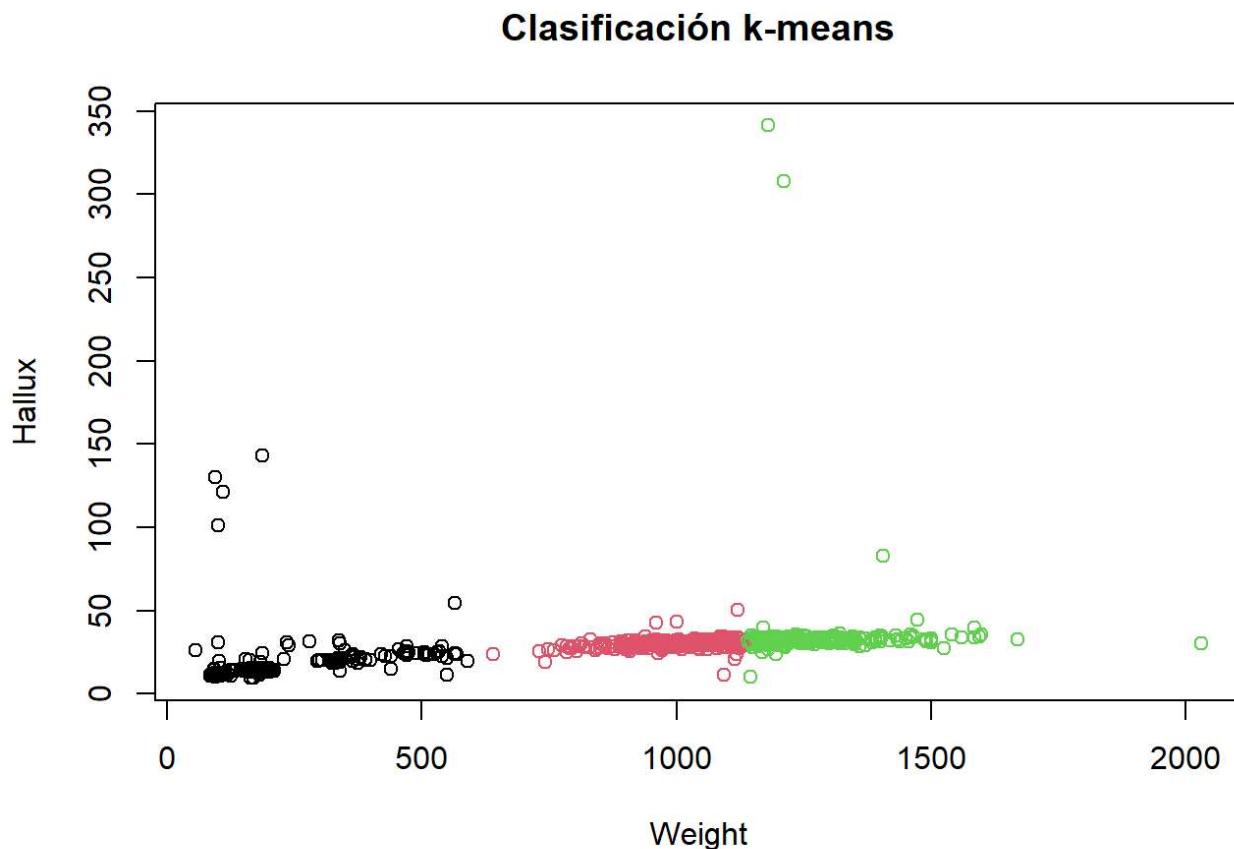
```
plot(Hawks_data[c(1,3)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real



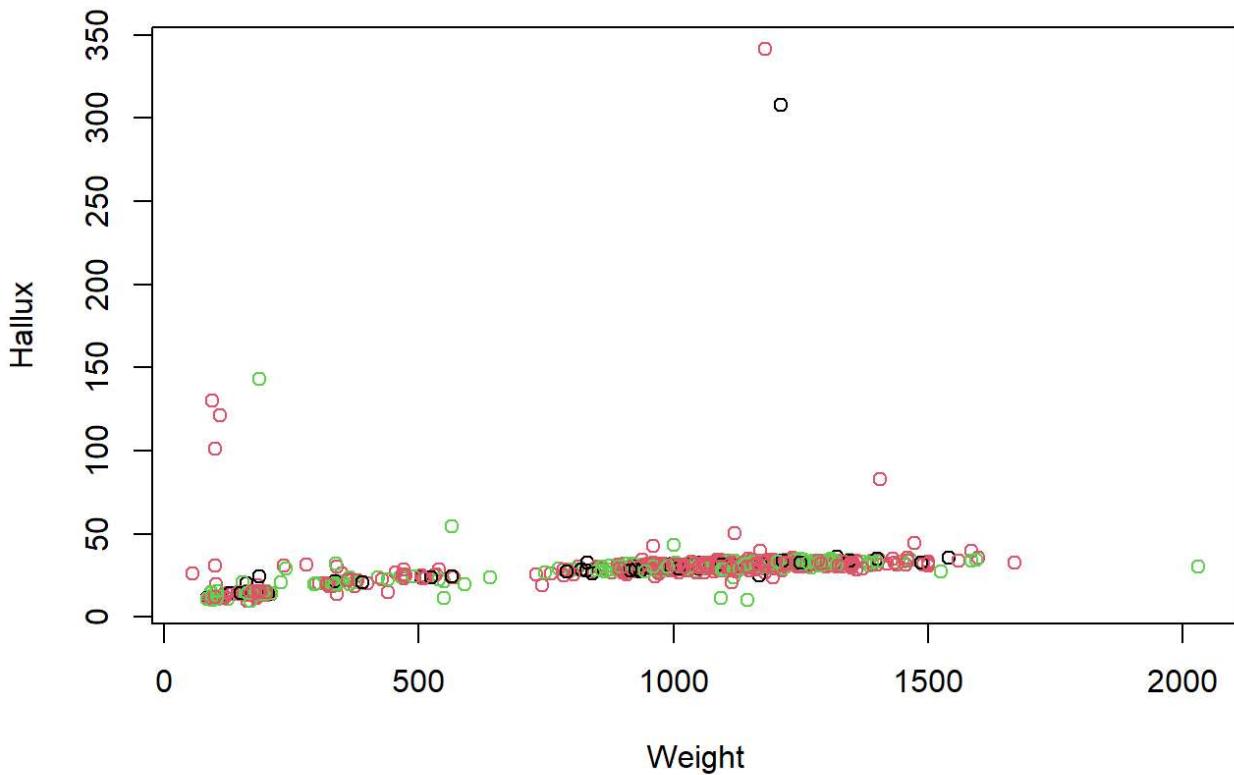
Podemos observar que Wing y Culmen tampoco son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

```
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")
```

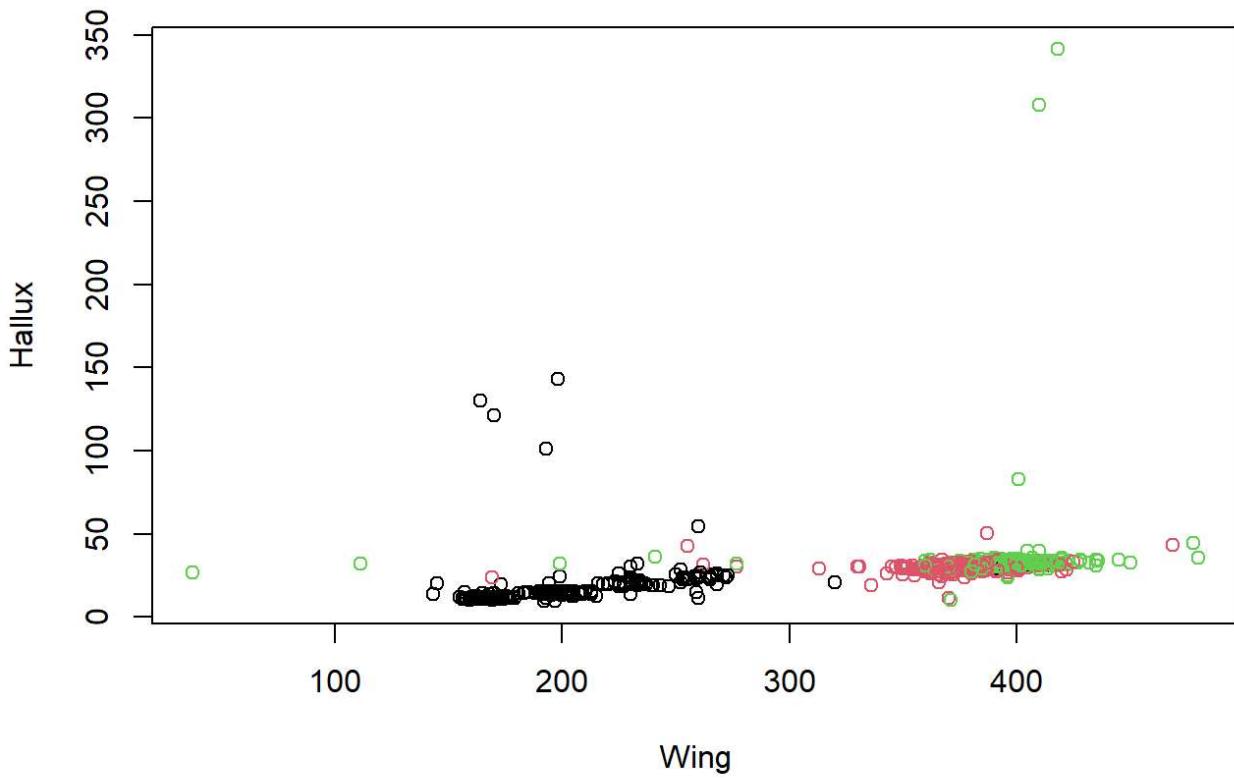
Clasificación real



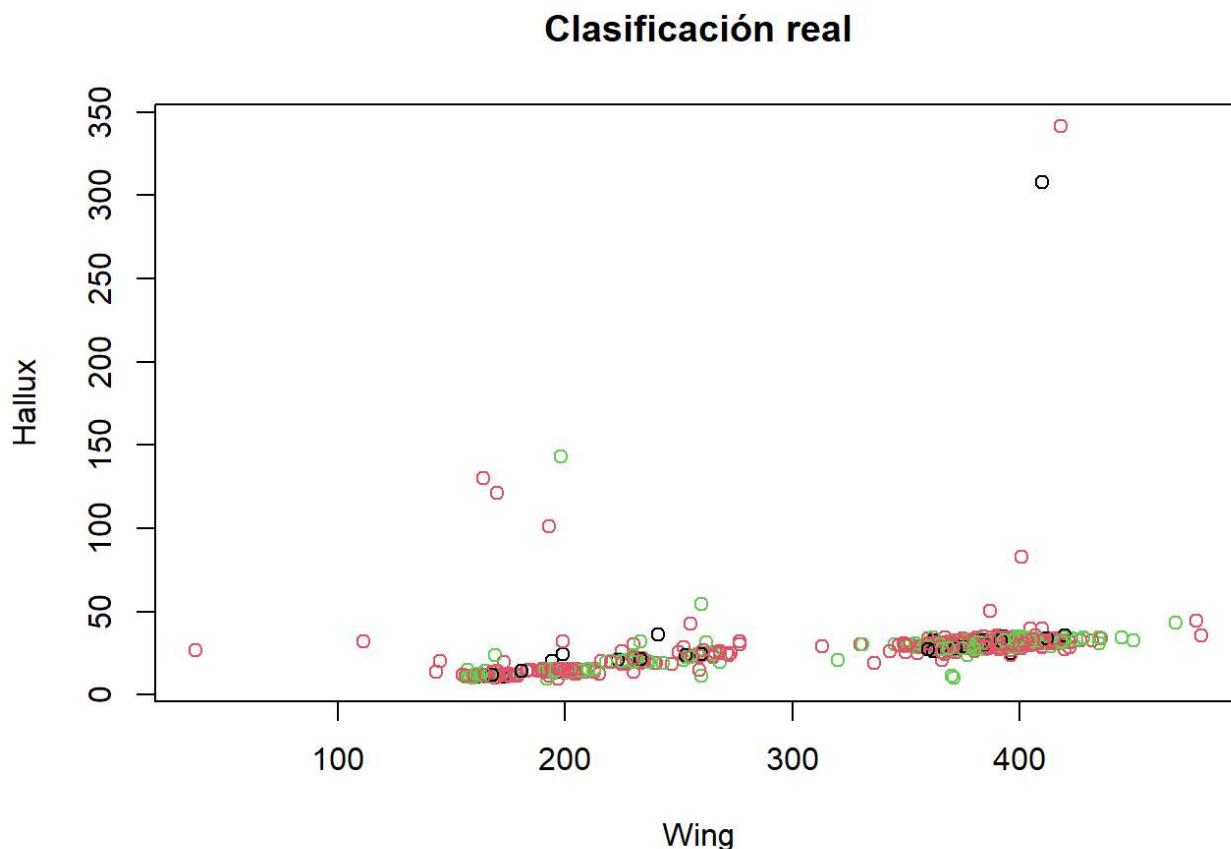
Podemos observar que Hallux y Weight tampoco son buenos indicadores para diferenciar a las tres subespecies, dado que todas las subespecies están demasiado mezcladas para poder diferenciar nada.

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

Clasificación k-means



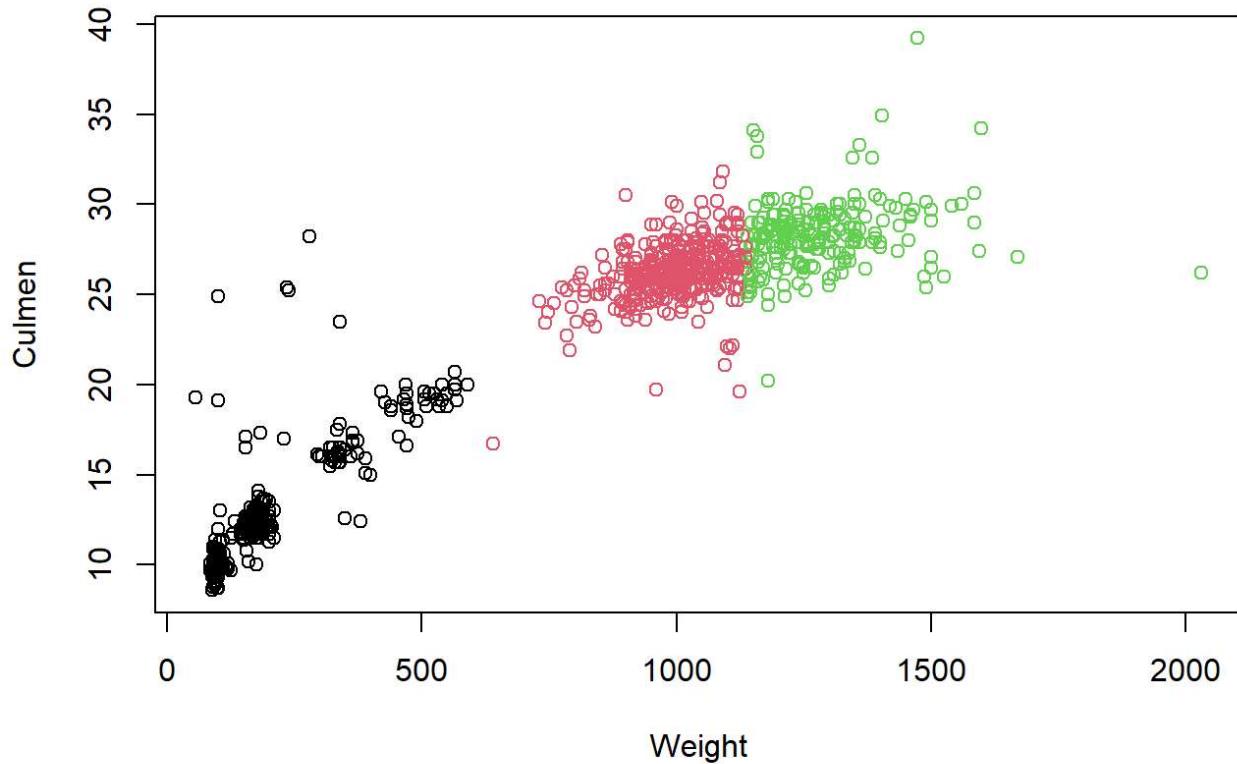
```
plot(Hawks_data[c(1,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```



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.

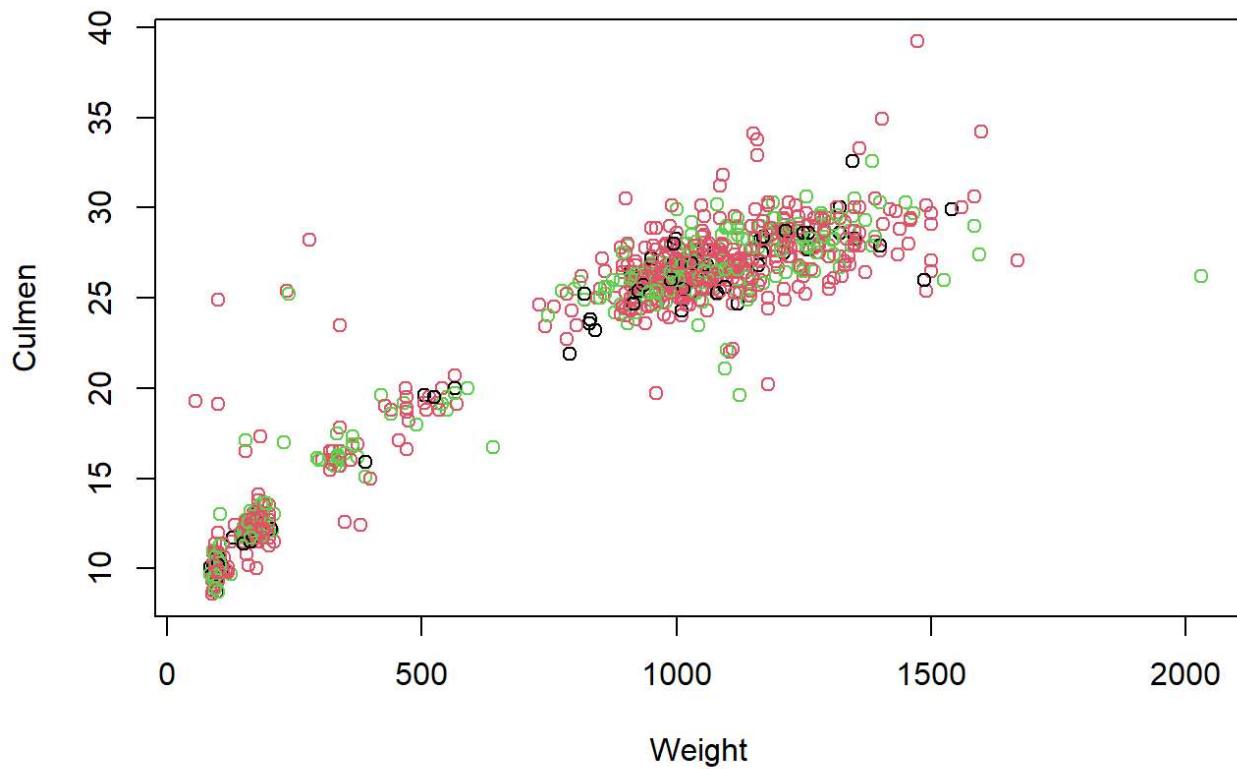
```
plot(Hawks_data[c(2,3)], col=Hawks3clusters$cluster, main="Clasificación k-means")
```

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])
```

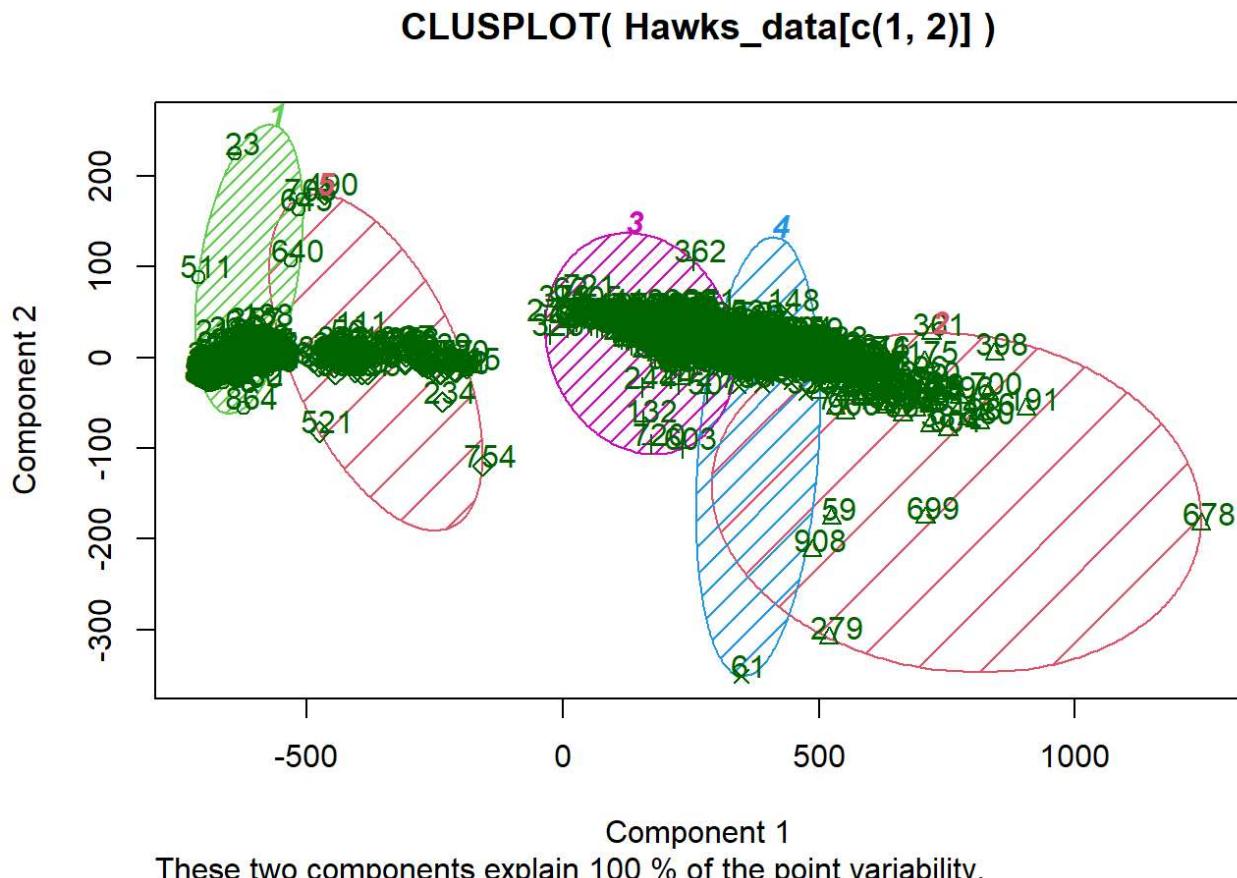
```
## [1] 0.6634045
```

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)
```

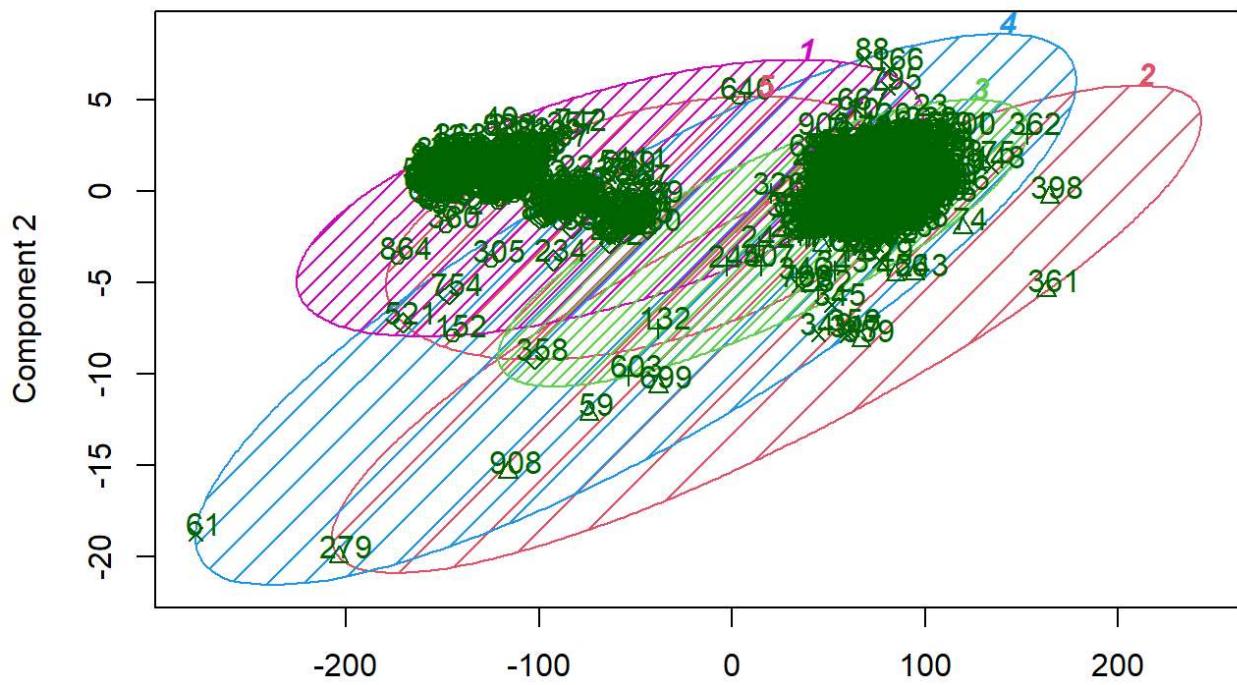
Visualizamos la agrupación con 5 clústeres y combinando los diferentes campos

```
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)])

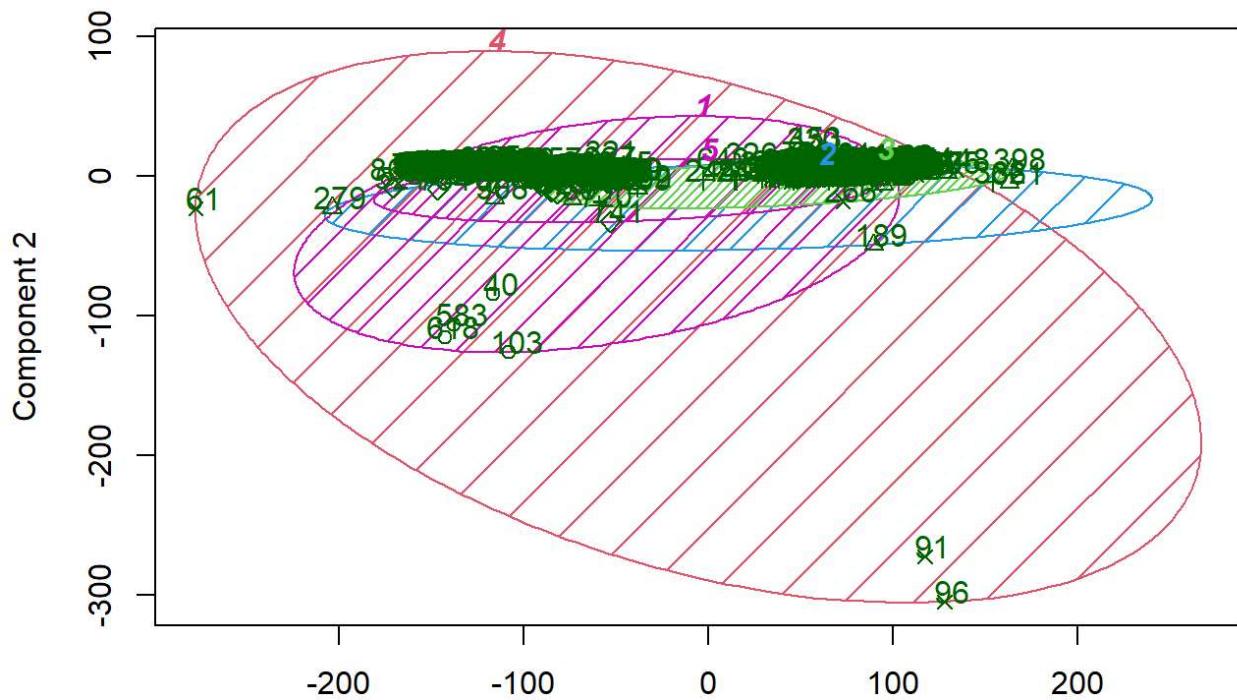


Component 1

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)
```

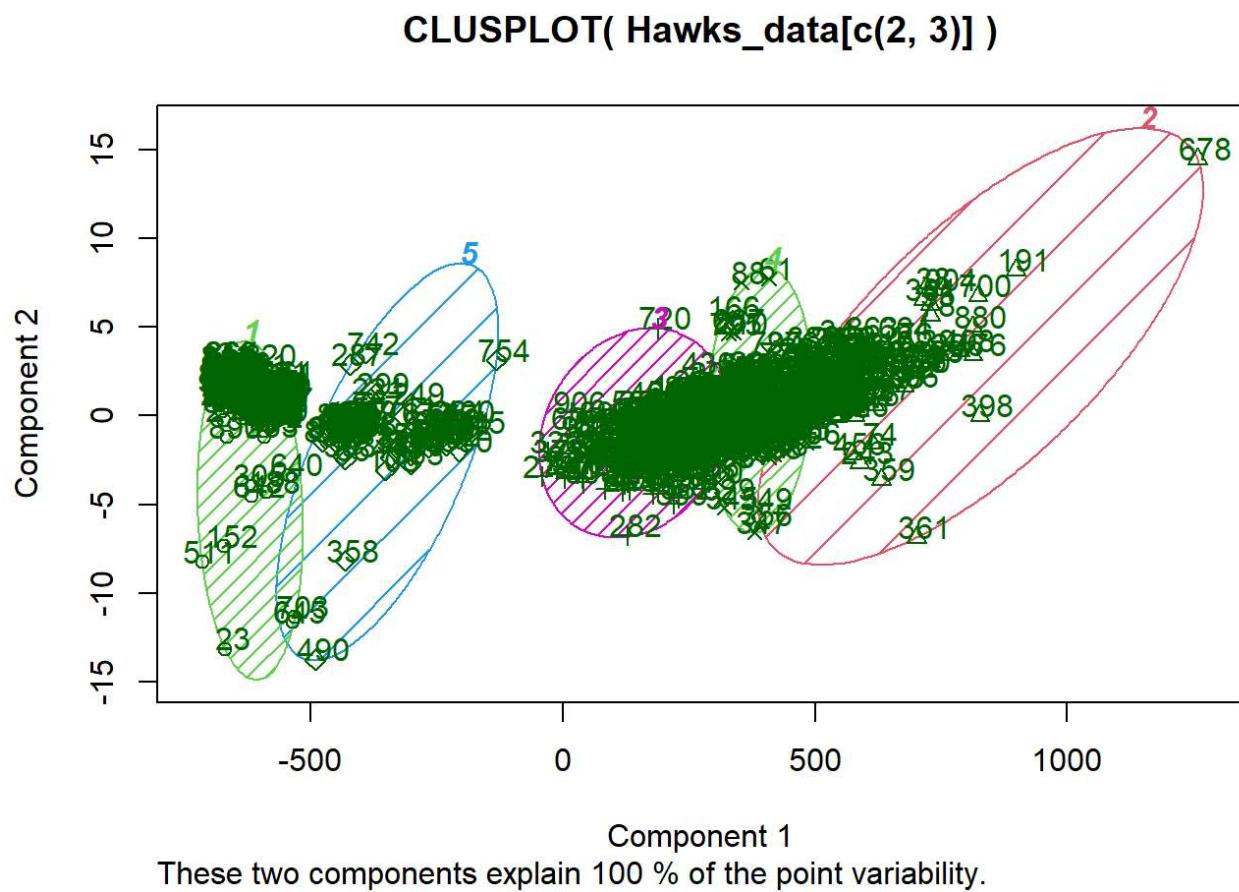
CLUSPLOT(Hawks_data[c(1, 4)])



Component 1

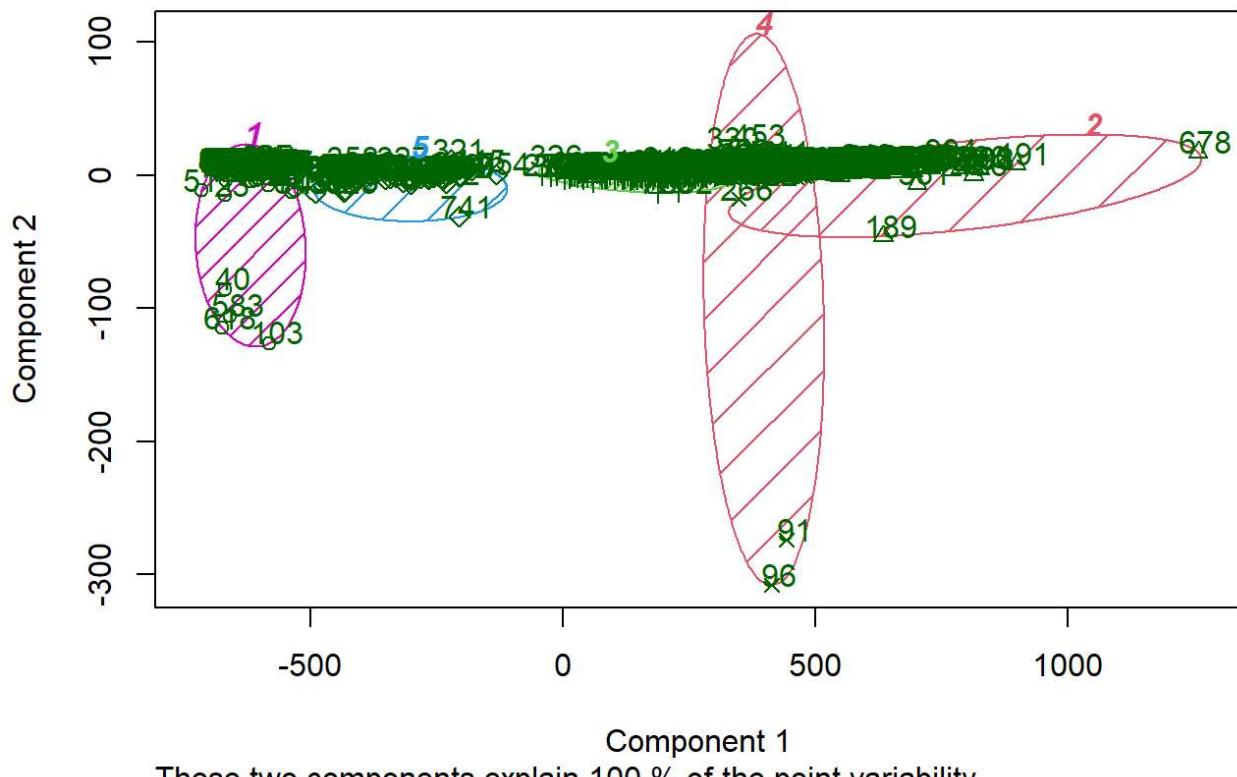
These two components explain 100 % of the point variability.

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

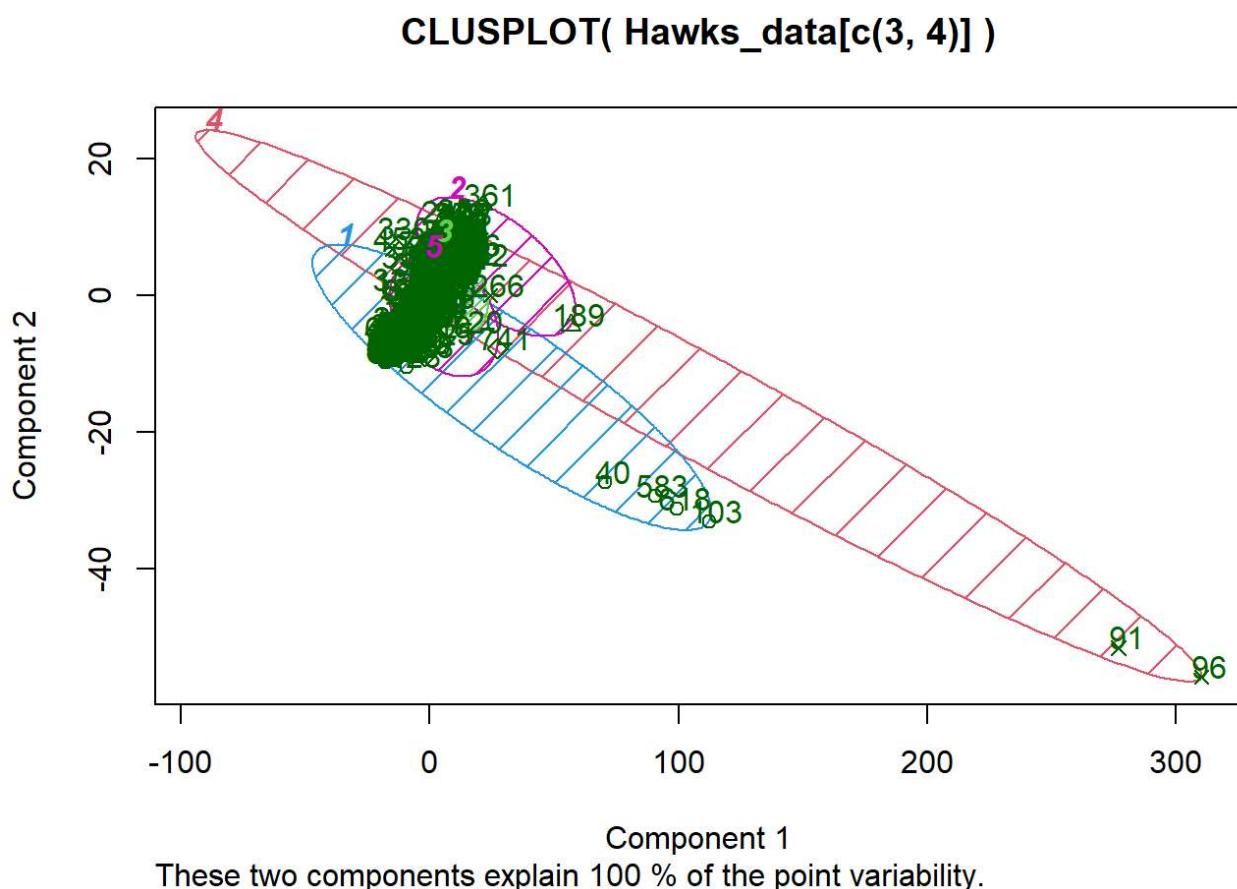


```
clusplot(Hawks_data[c(2,4)], Hawks5clusters$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```

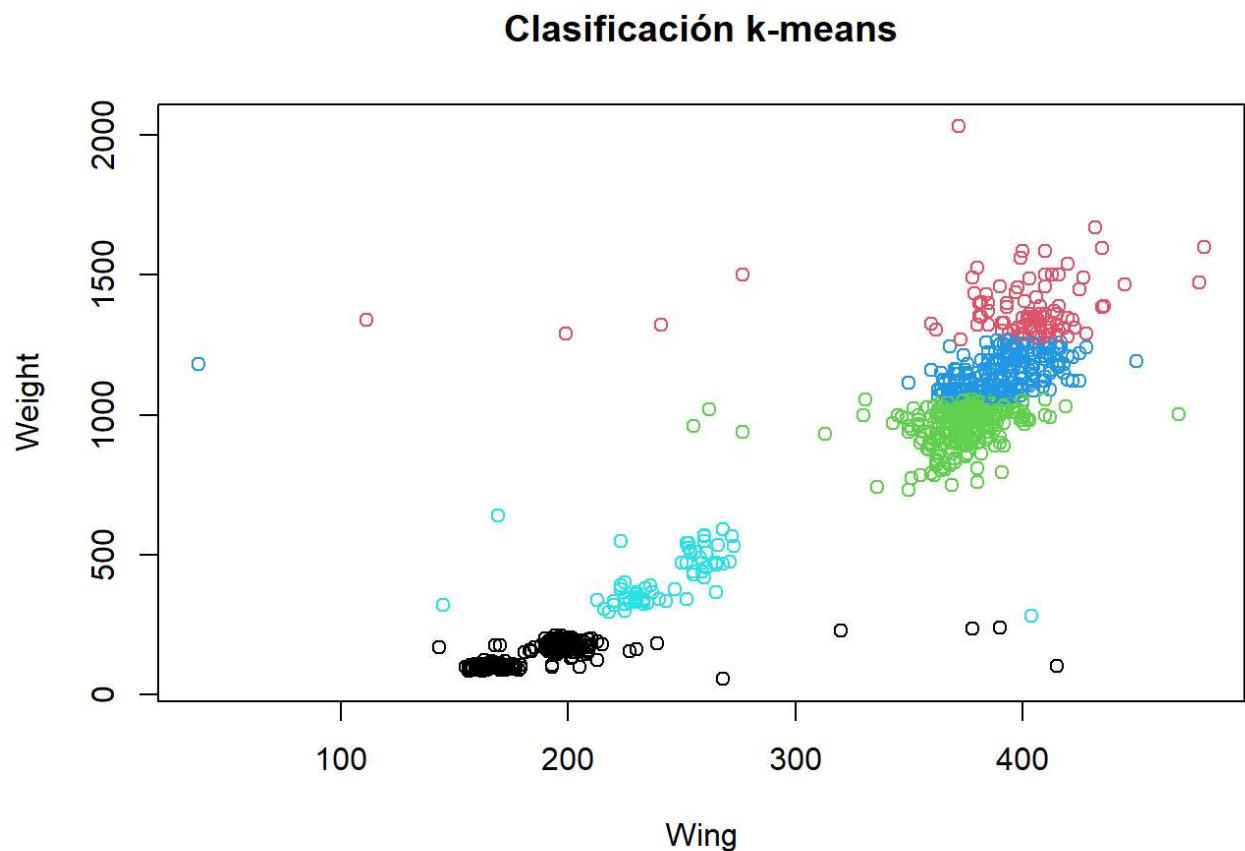
CLUSPLOT(Hawks_data[c(2, 4)])



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

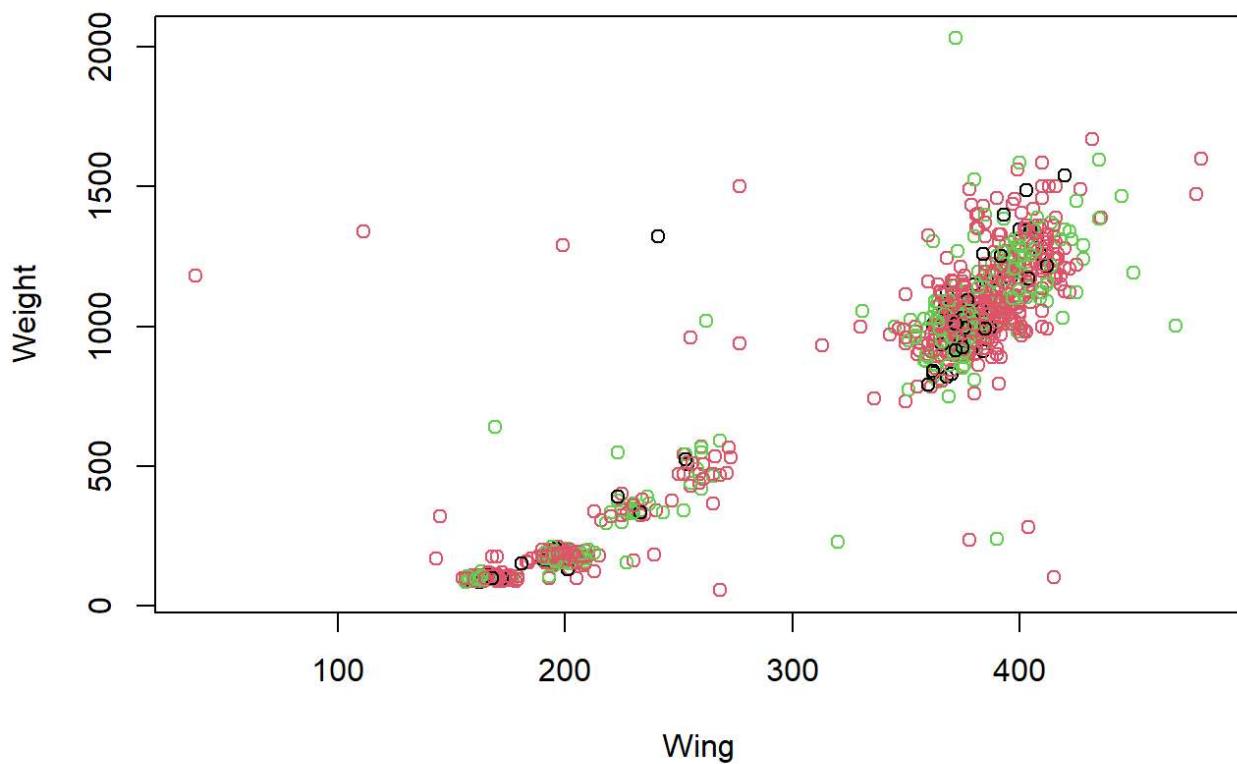


```
plot(Hawks_data[c(1,2)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```



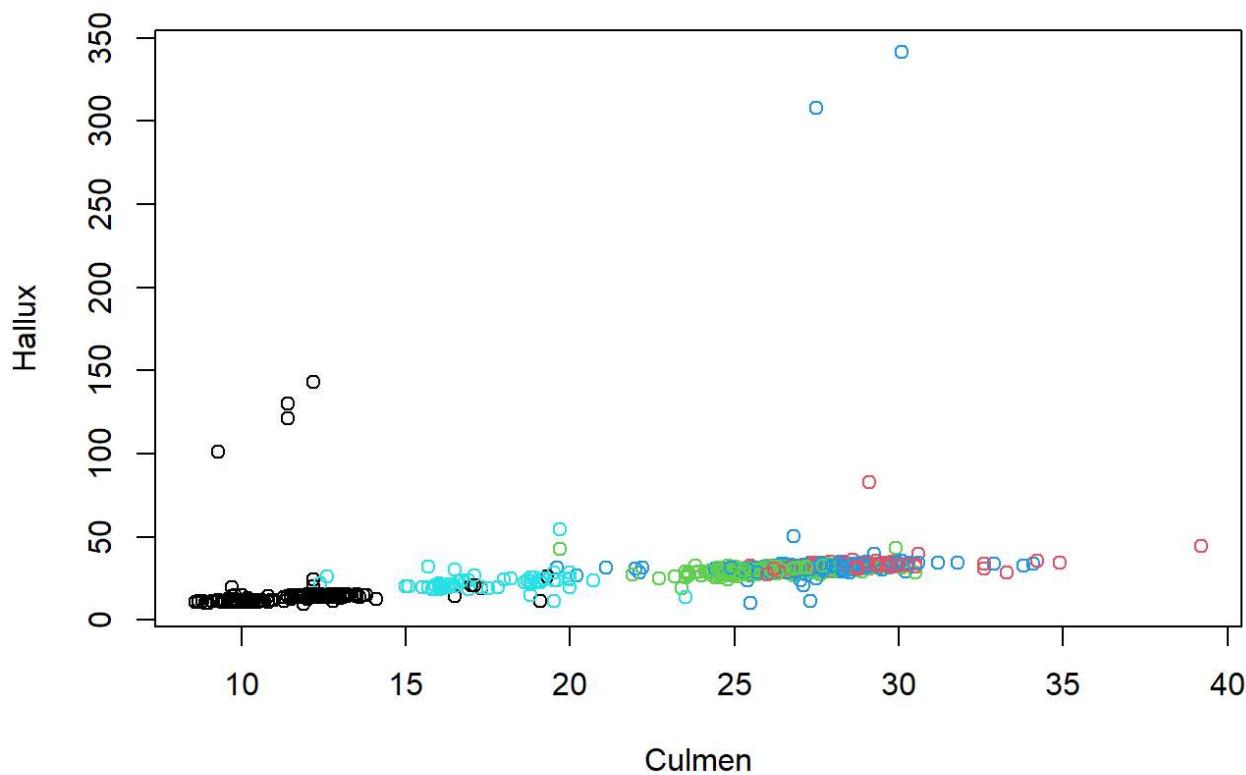
```
plot(Hawks_data[c(1,2)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real

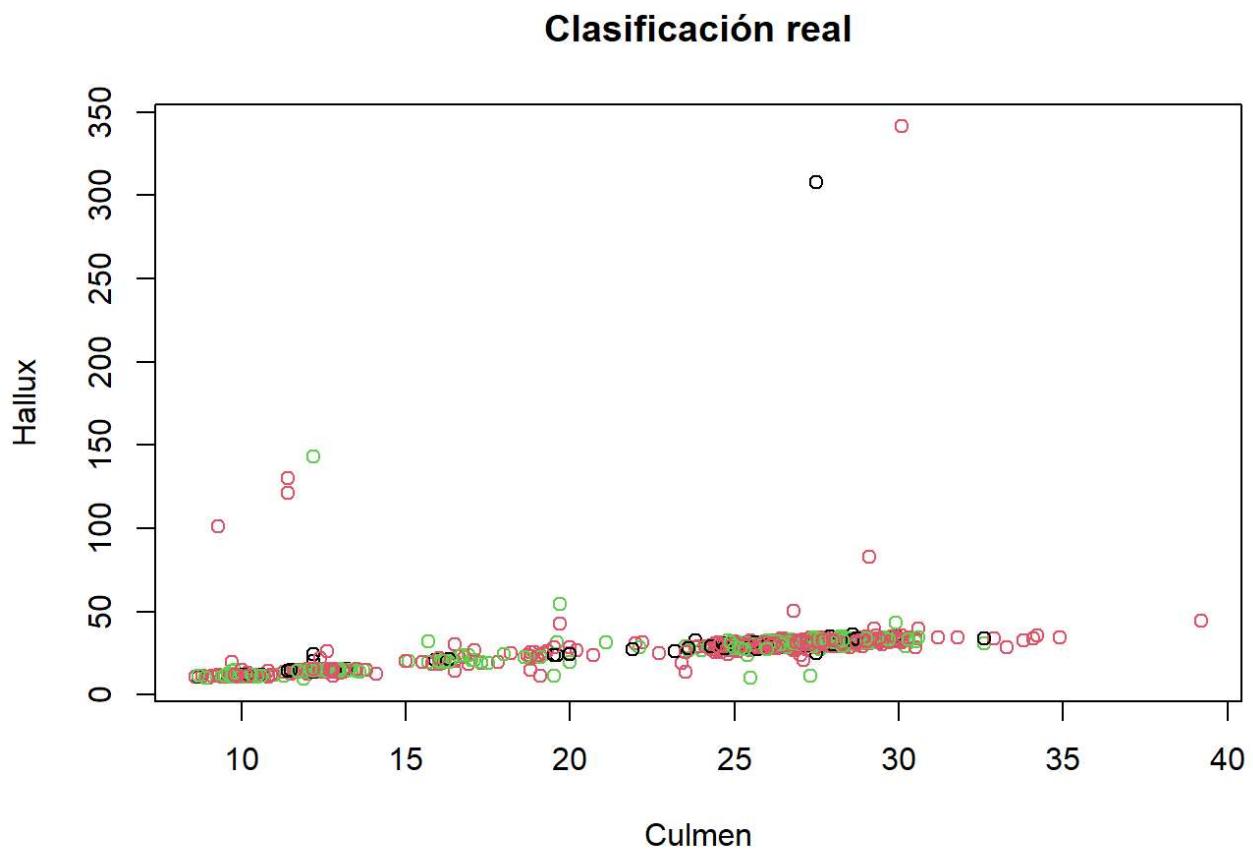


```
plot(Hawks_data[c(3,4)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means

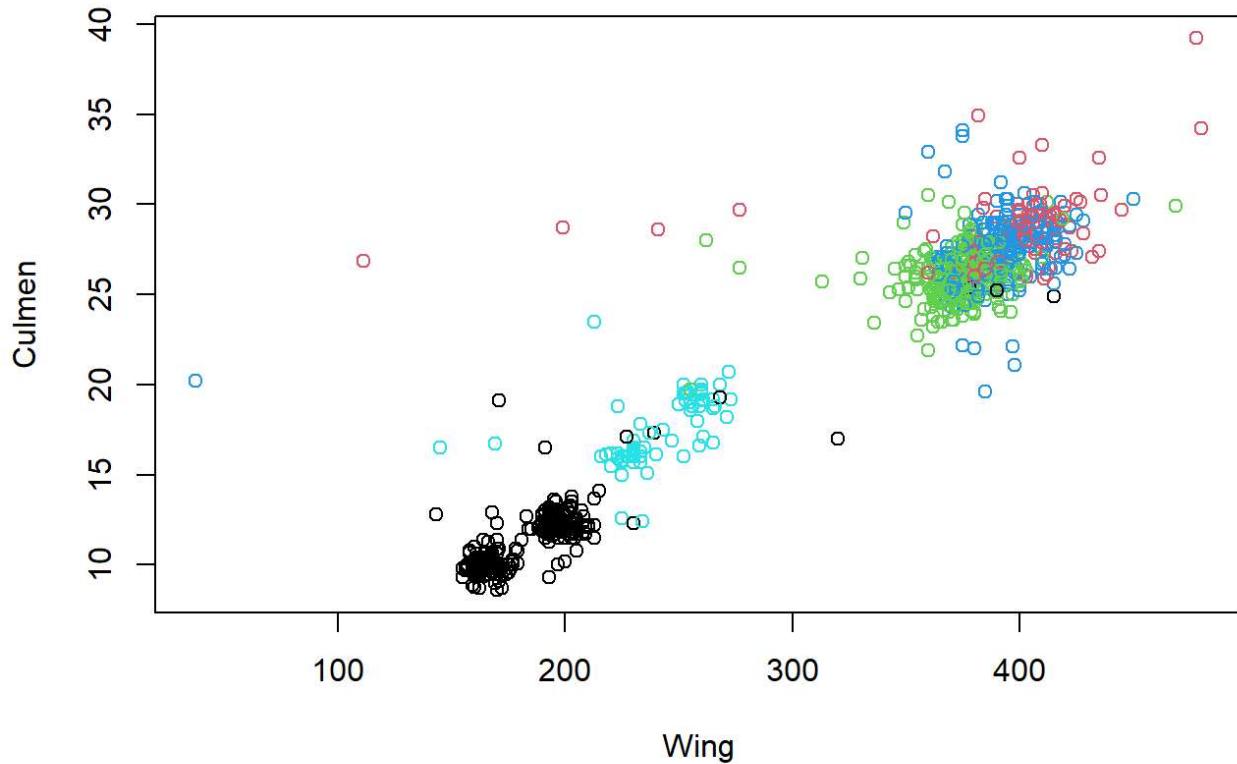


```
plot(Hawks_data[c(3,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```



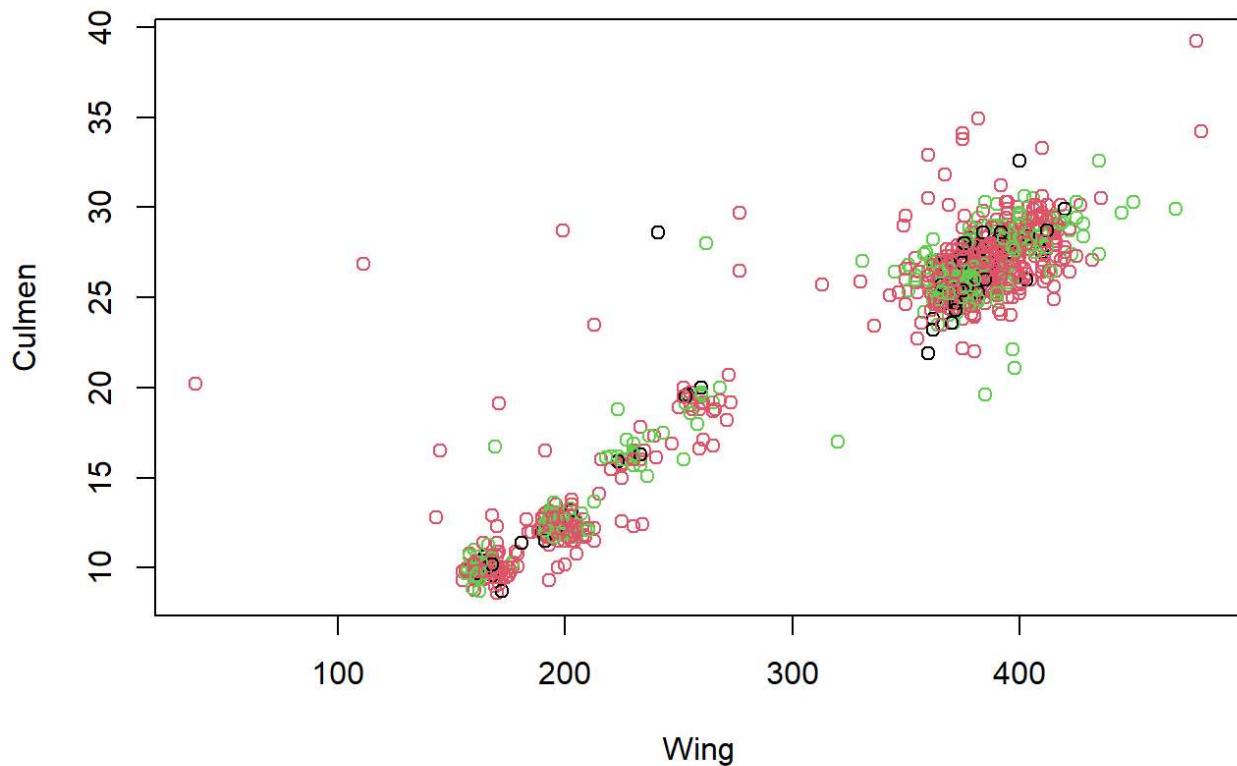
```
plot(Hawks_data[c(1,3)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means

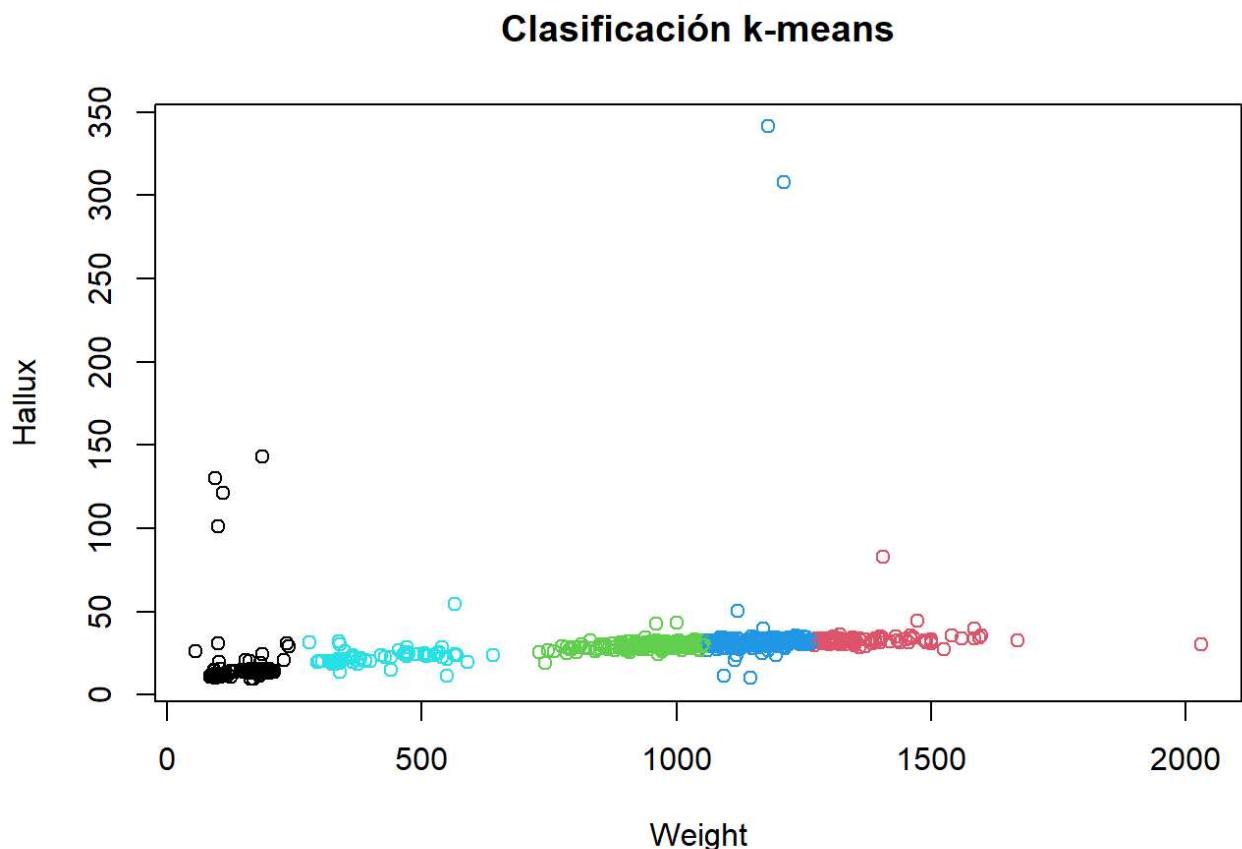


```
plot(Hawks_data[c(1,3)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real

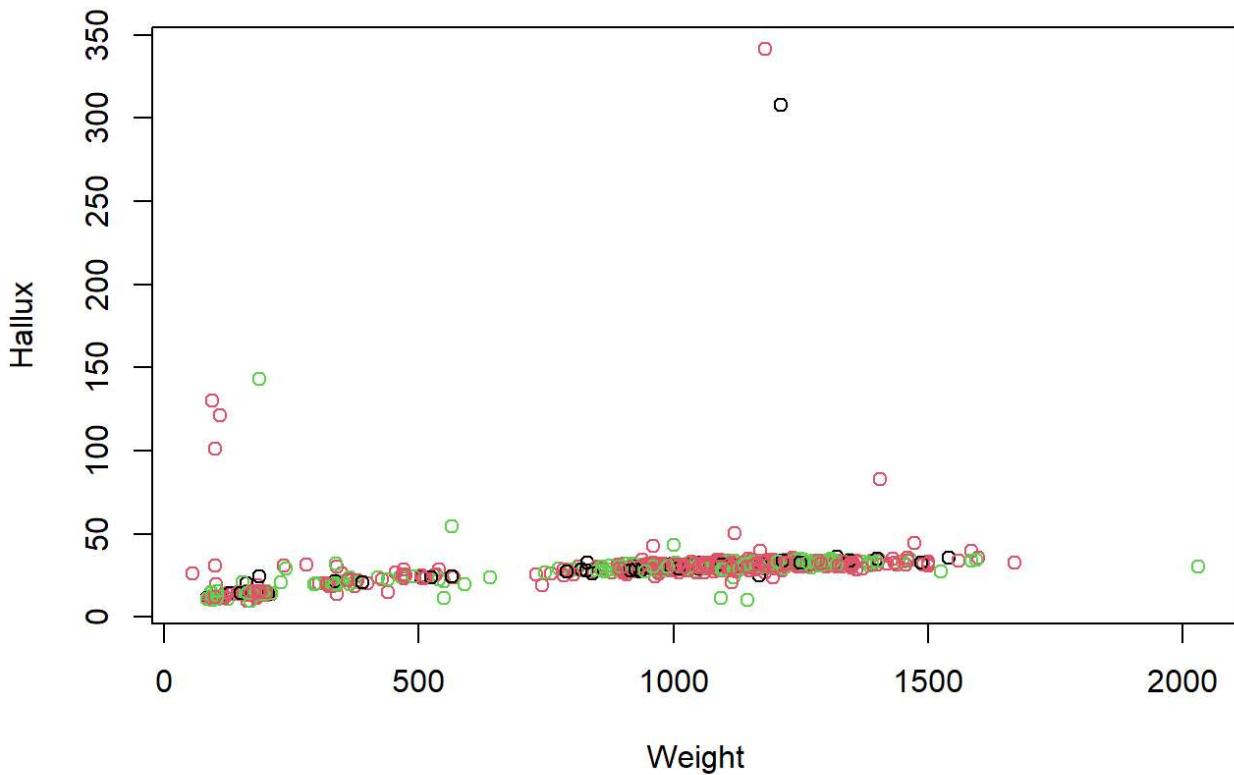


```
plot(Hawks_data[c(2,4)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```



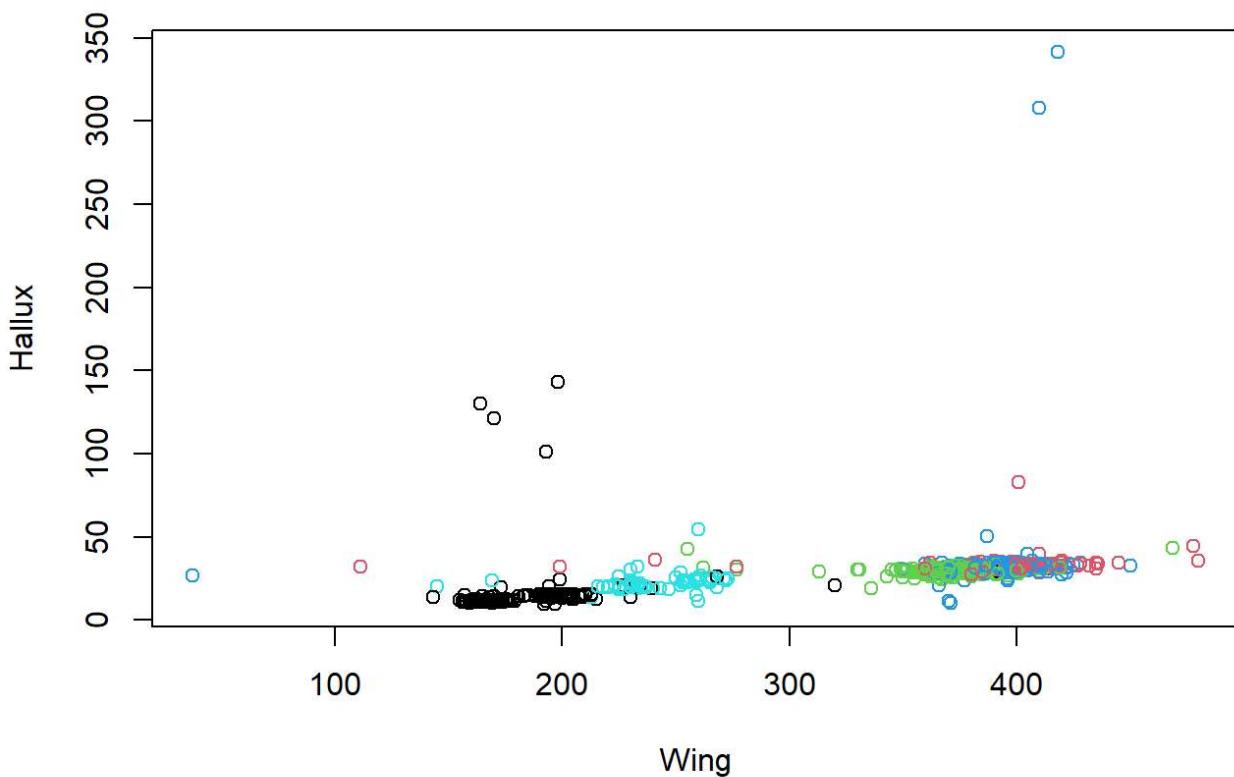
```
plot(Hawks_data[c(2,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```

Clasificación real

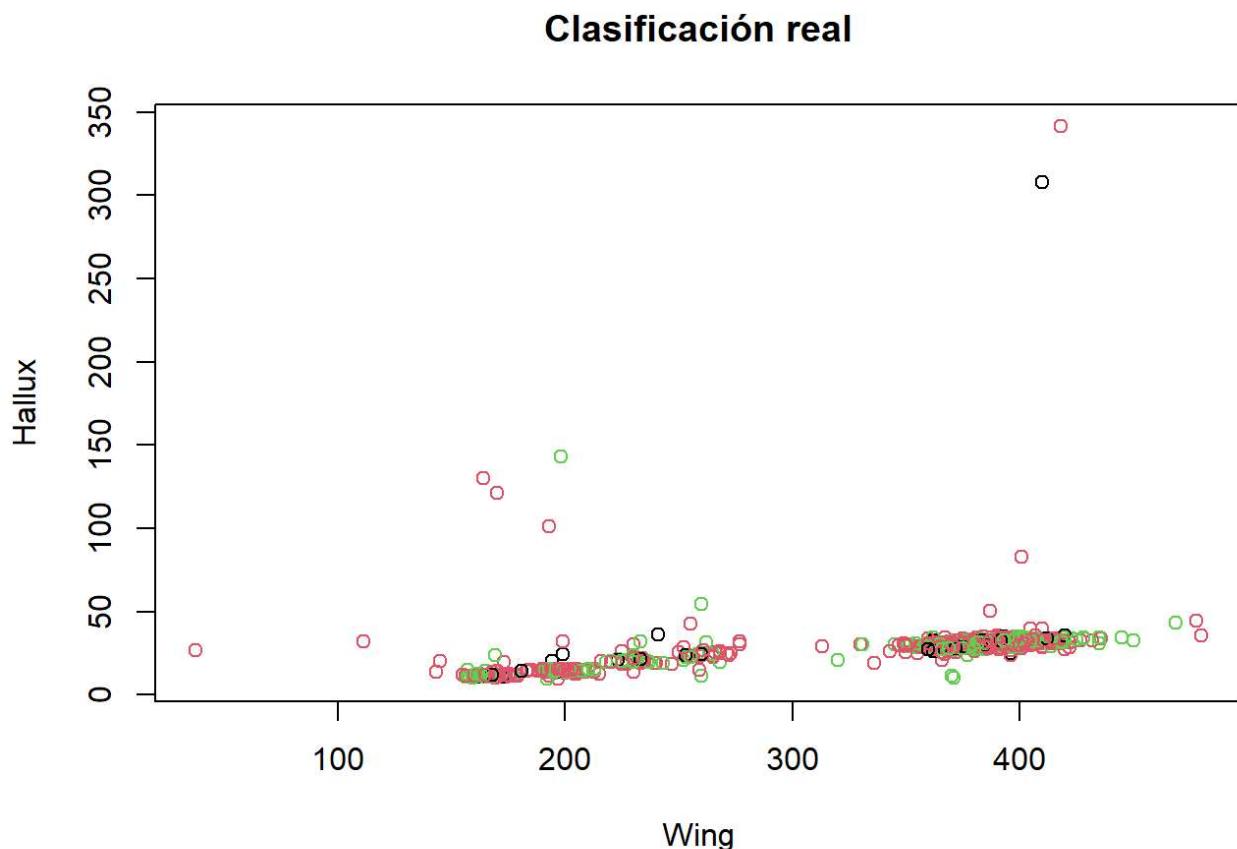


```
plot(Hawks_data[c(1,4)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```

Clasificación k-means

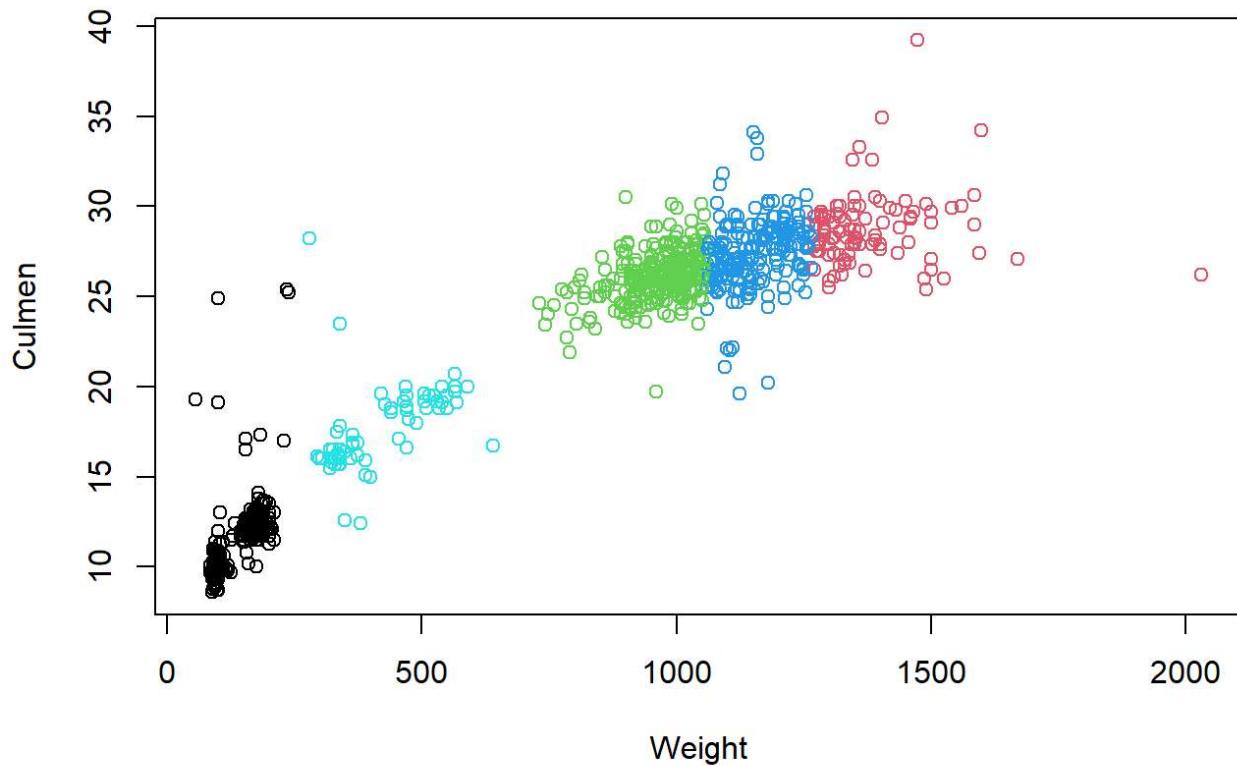


```
plot(Hawks_data[c(1,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```



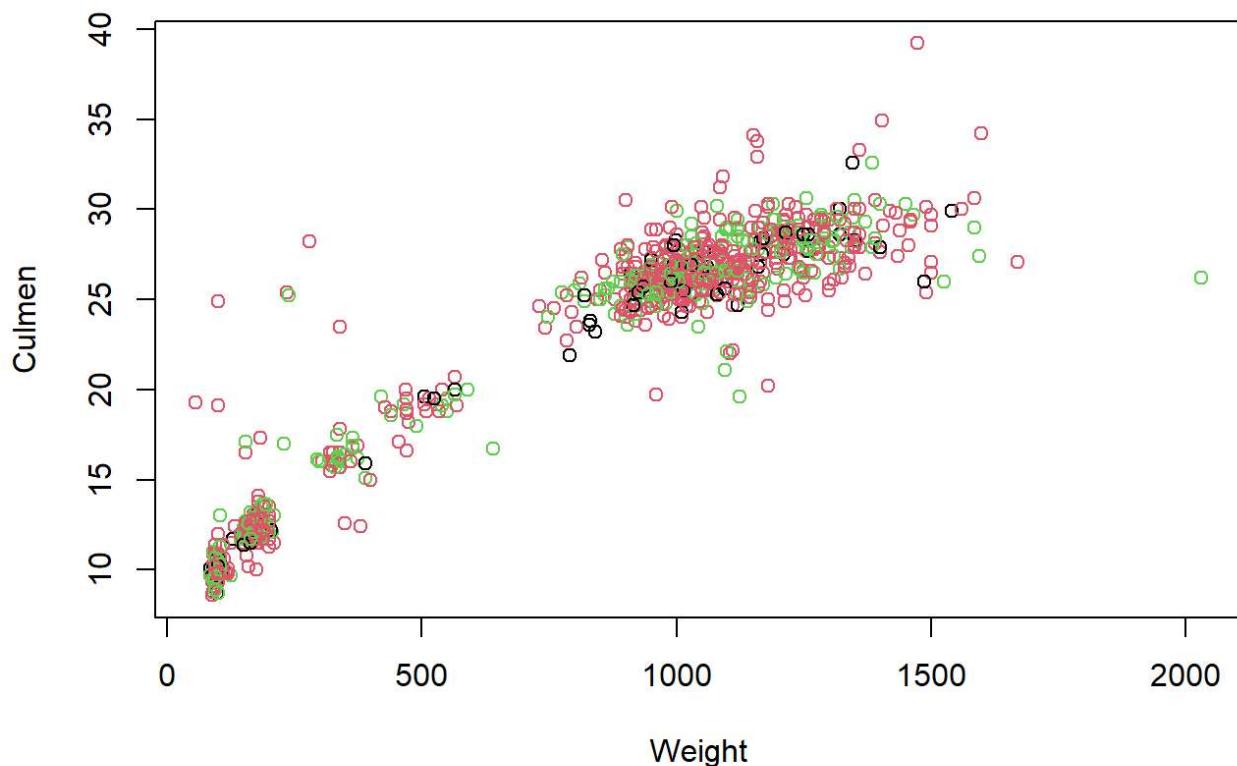
```
plot(Hawks_data[c(2,3)], col=Hawks5clusters$cluster, main="Clasificación k-means")
```

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
```



```
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
```



```
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
```

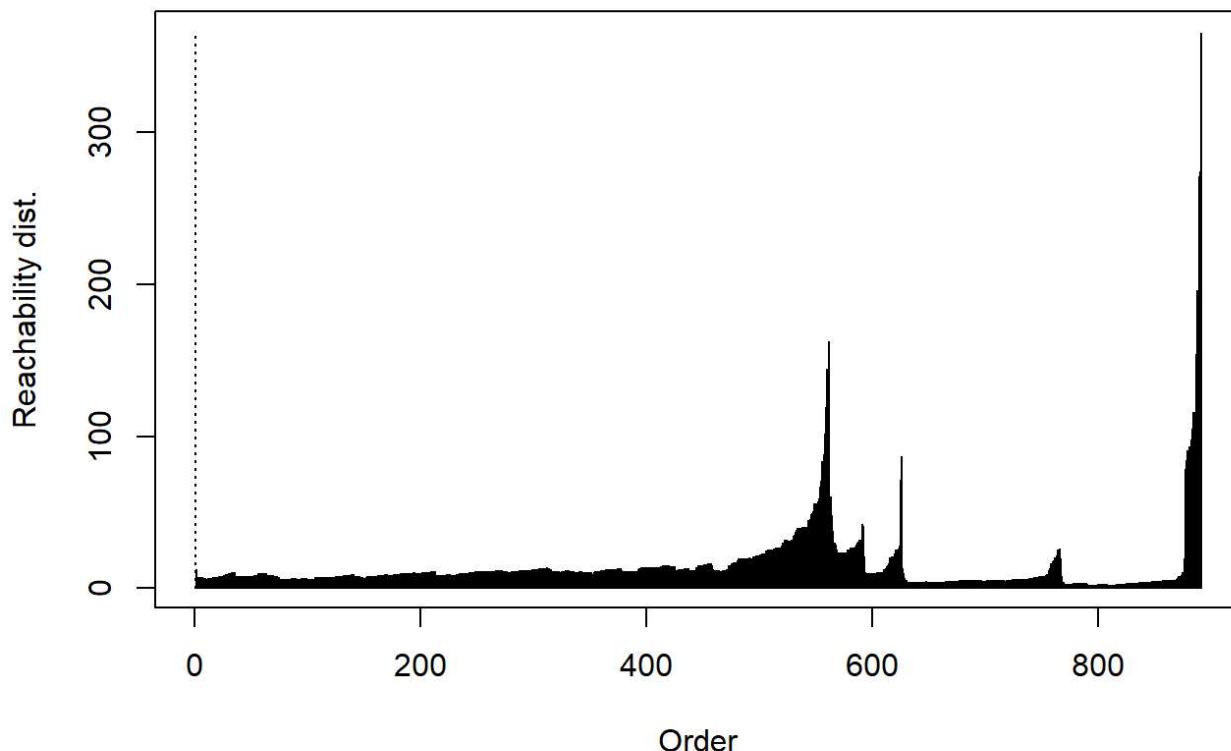
```
res50$order
```



```
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
```

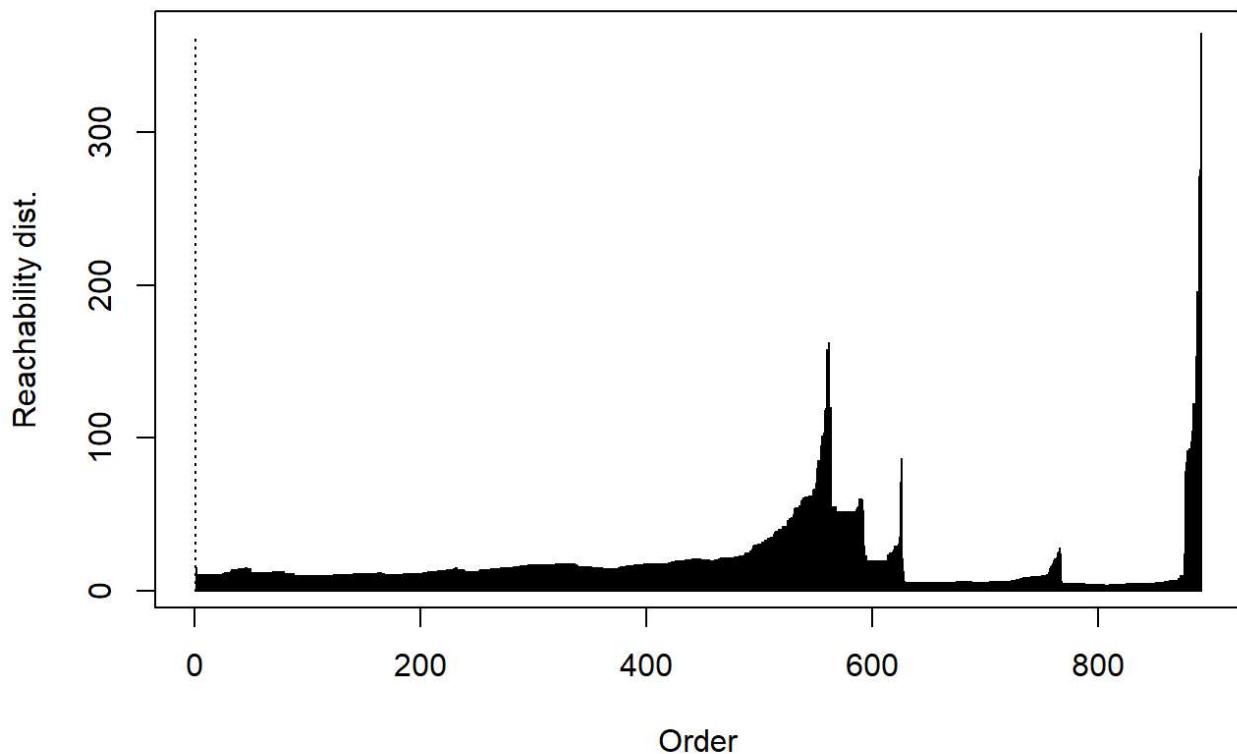
```
### Gráficas de alcanzabilidad  
plot(res10)
```

Reachability Plot



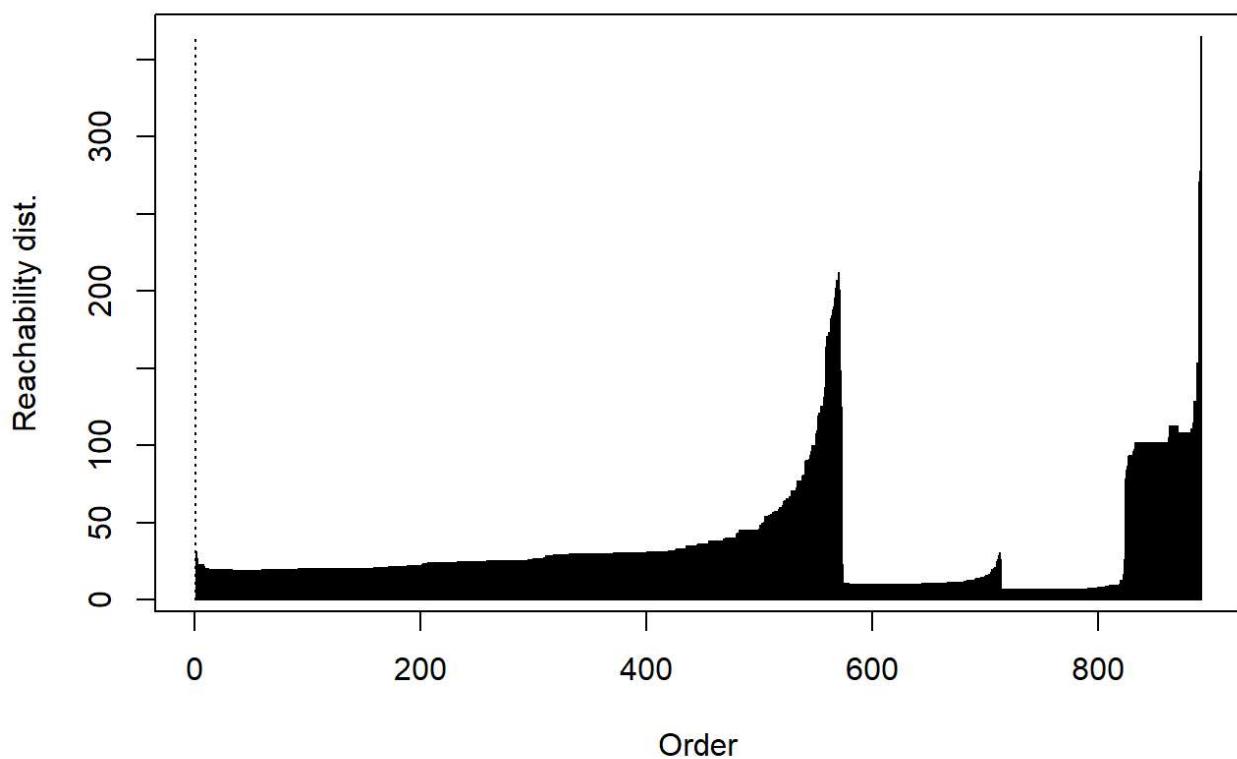
```
plot(res20)
```

Reachability Plot



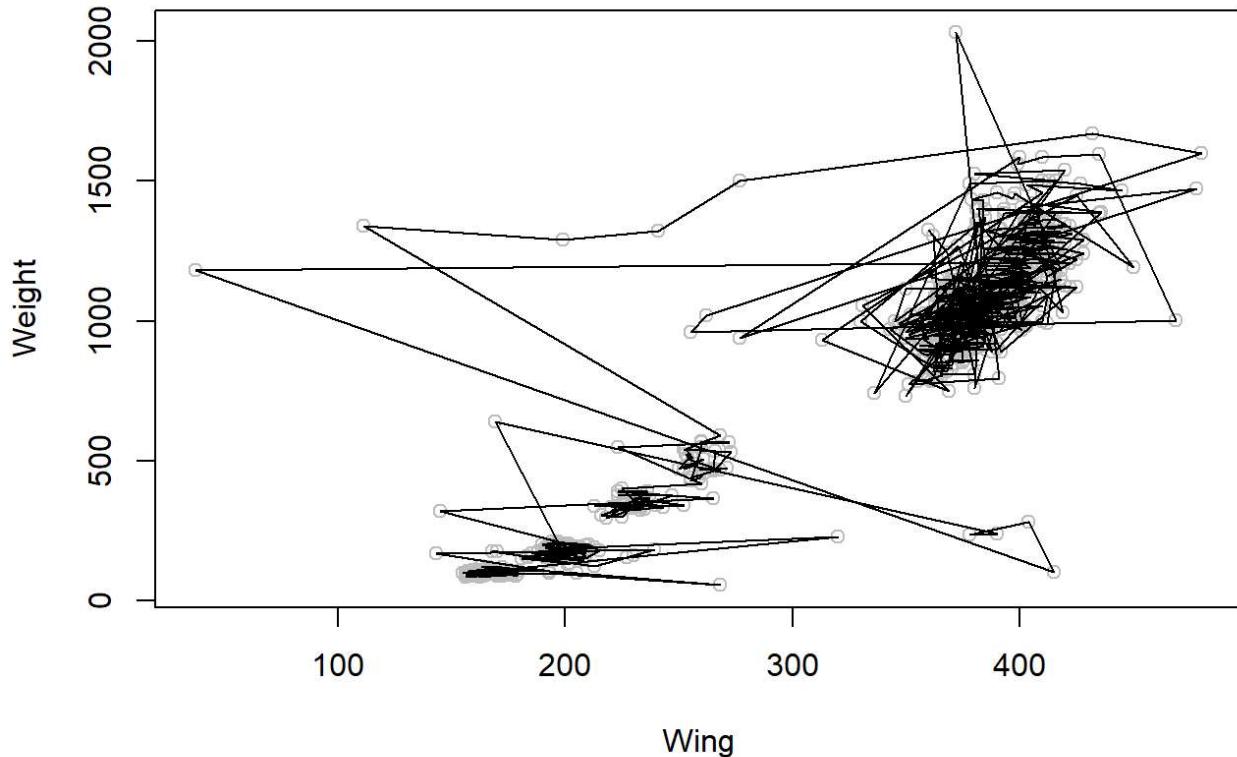
```
plot(res50)
```

Reachability Plot

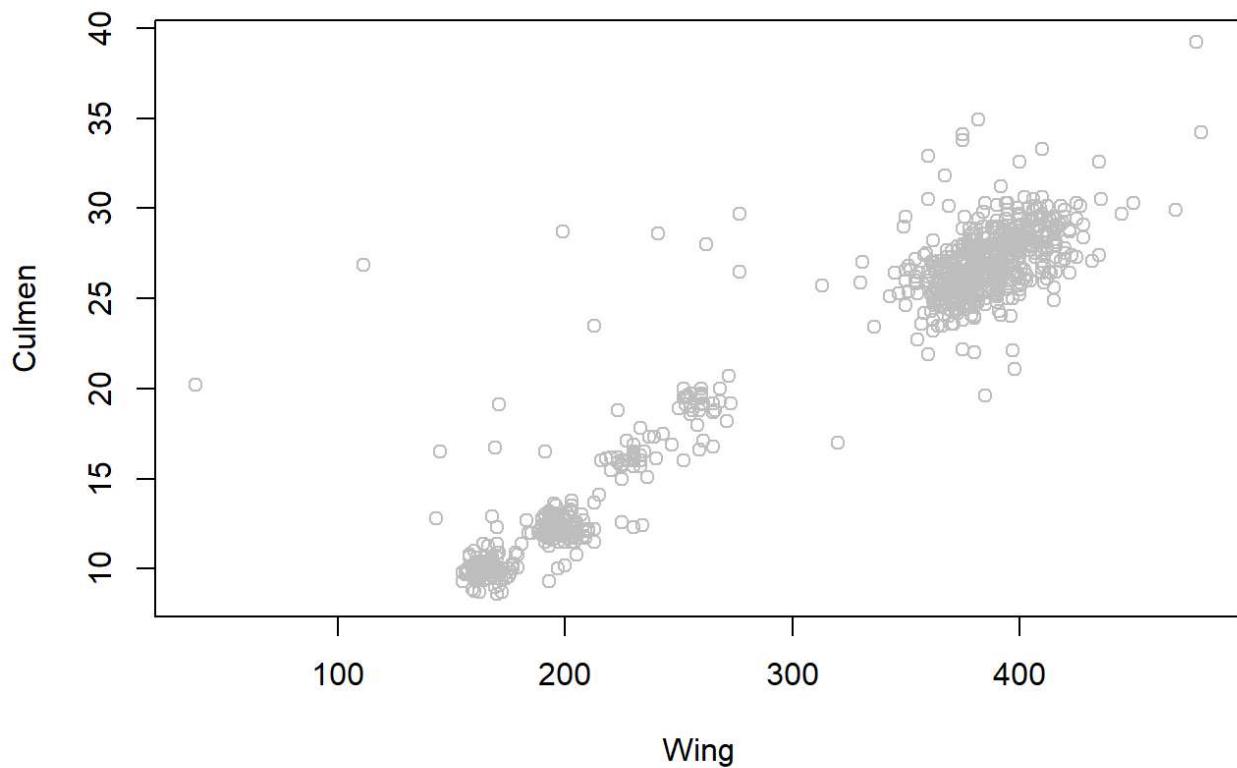


Ahora haremos el proceso con el criterio de vecindad en 10, lo primero seria obtener la relación entre las variables.

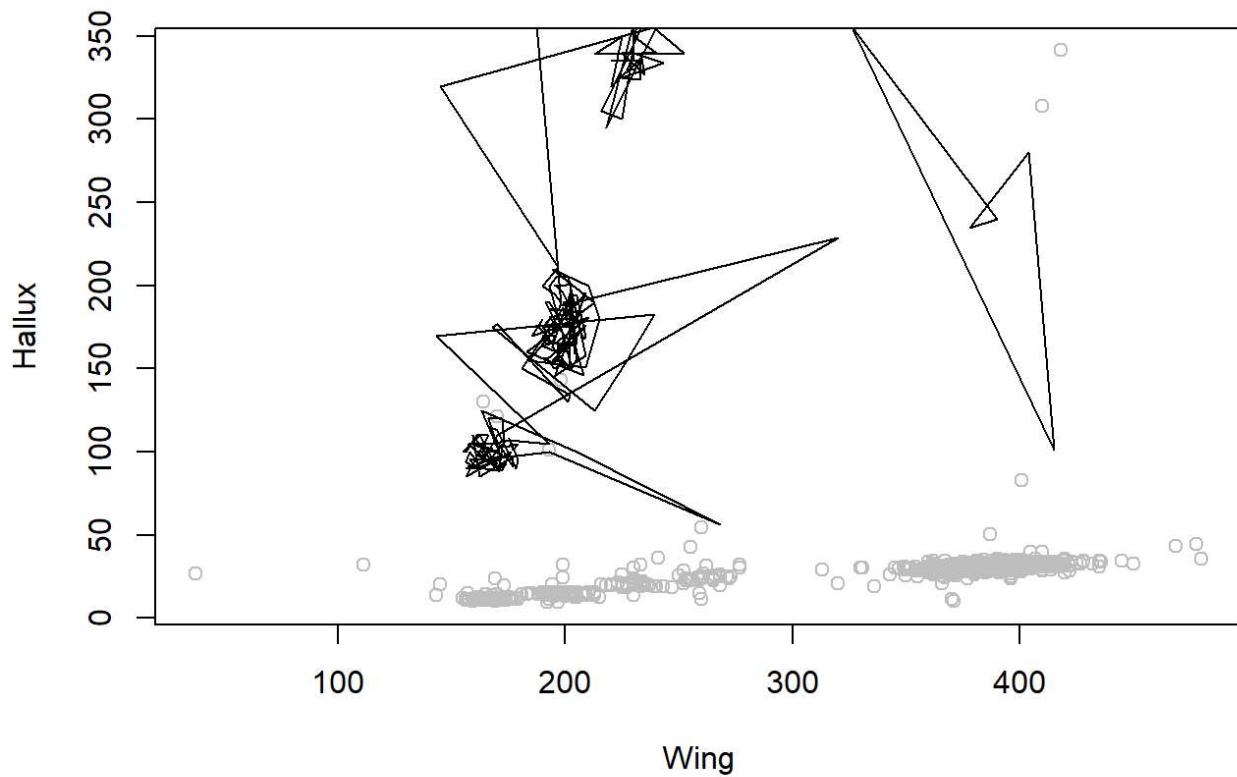
```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data[c(1,2)], col = "grey")
polygon(Hawks_data[res10$order,])
```



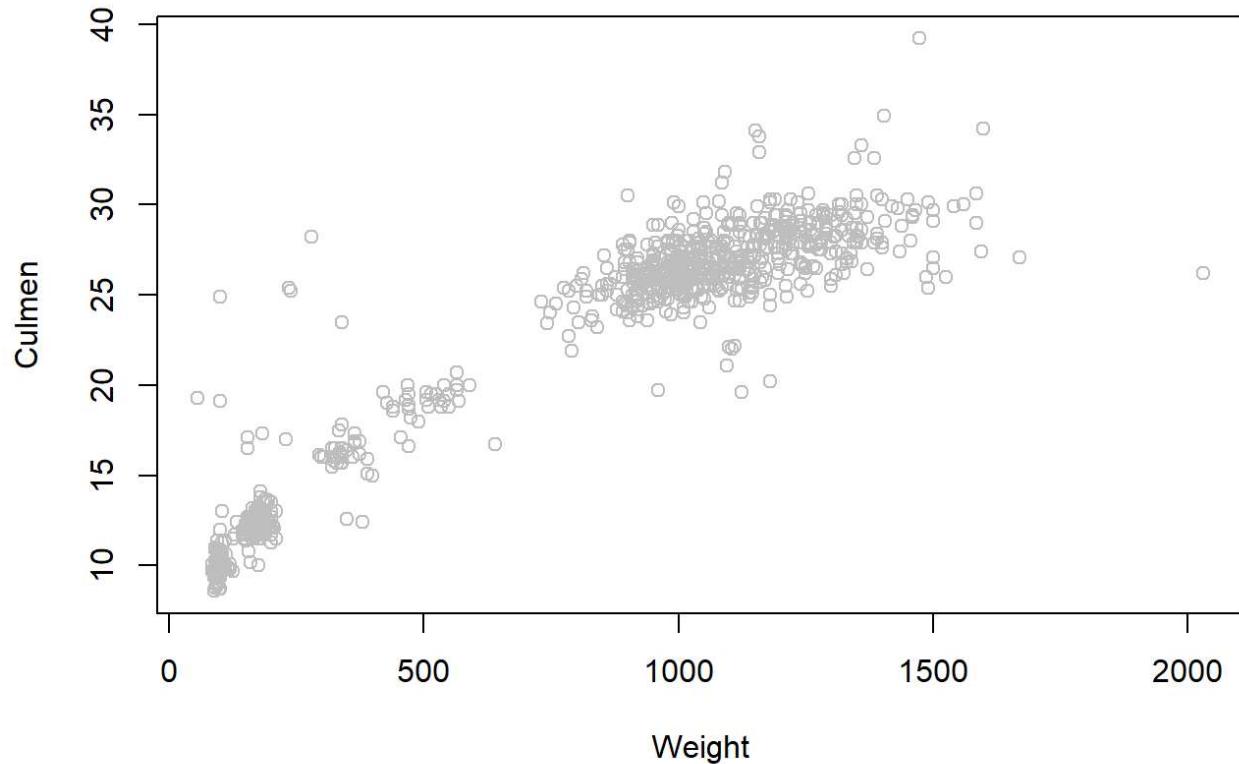
```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data[c(1,3)], col = "grey")
polygon(Hawks_data[res10$order,])
```



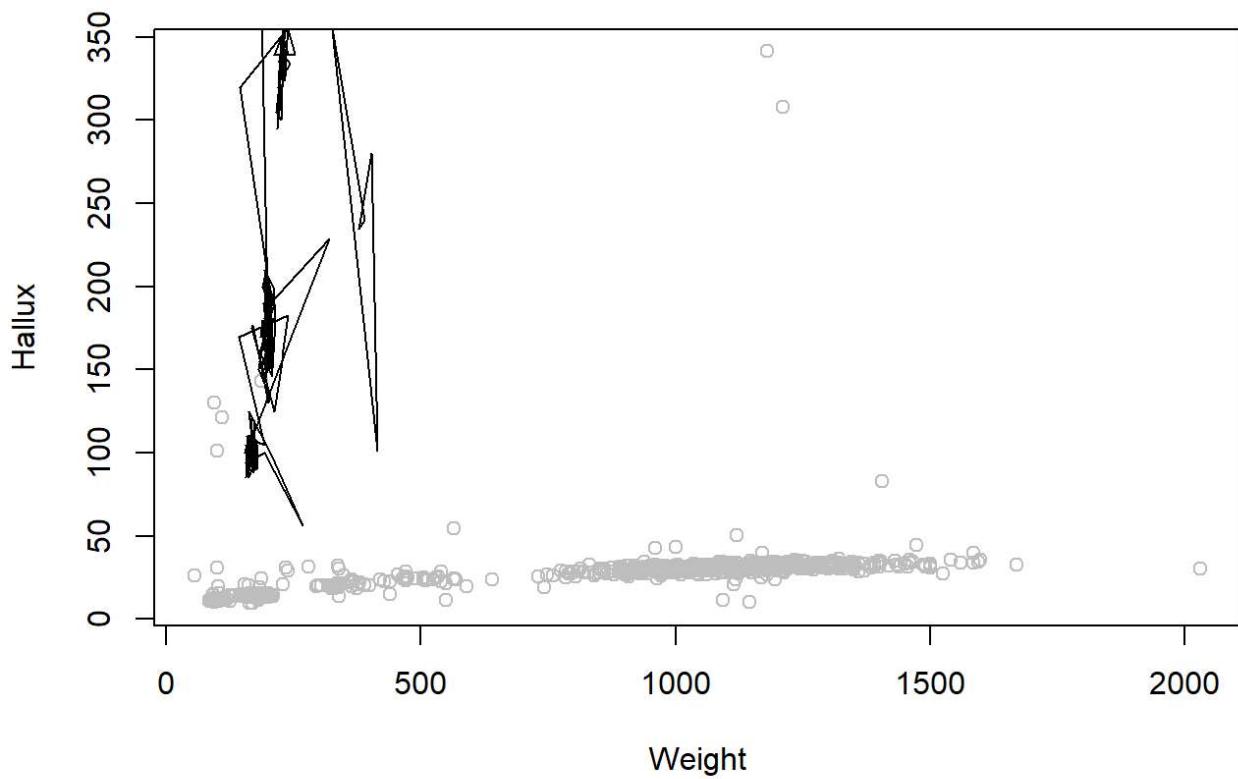
```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data[c(1,4)], col = "grey")
polygon(Hawks_data[res10$order,])
```



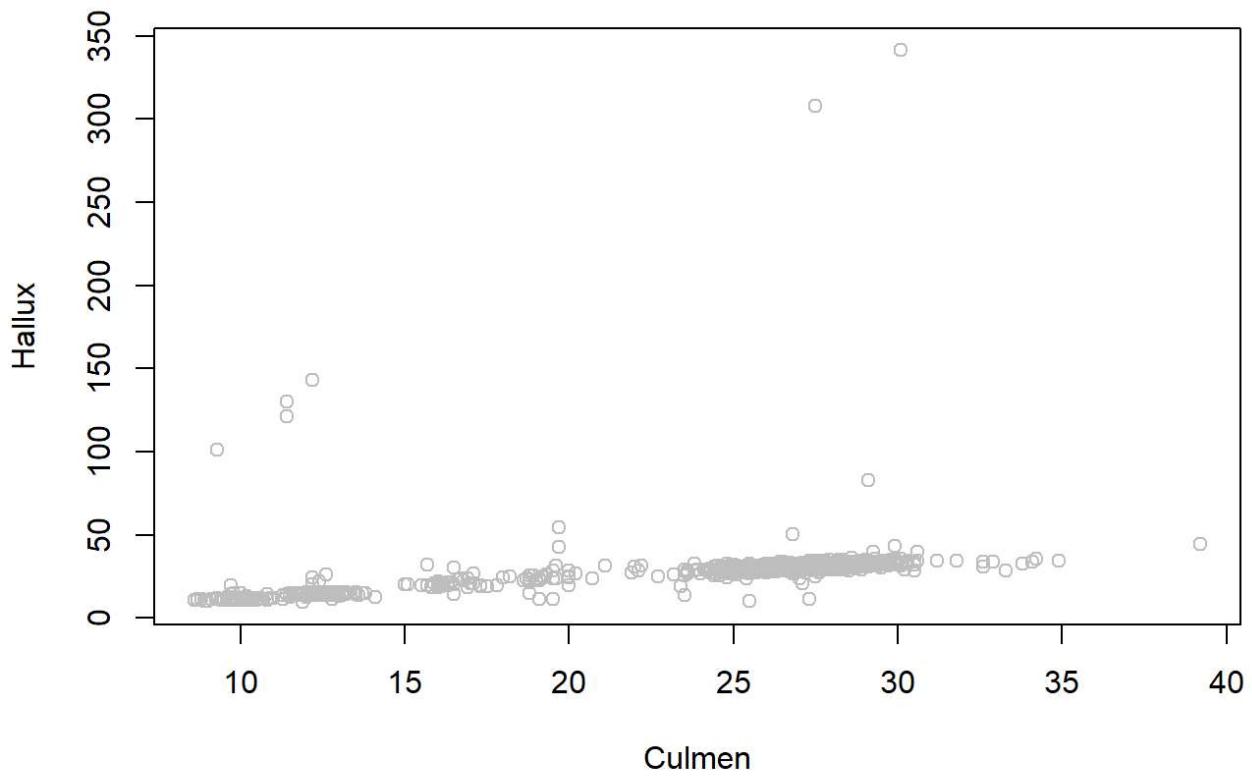
```
### Dibujo de las trazas que relacionan puntos (res10)
plot(Hawks_data[c(2,3)], col = "grey")
polygon(Hawks_data[res10$order,])
```



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

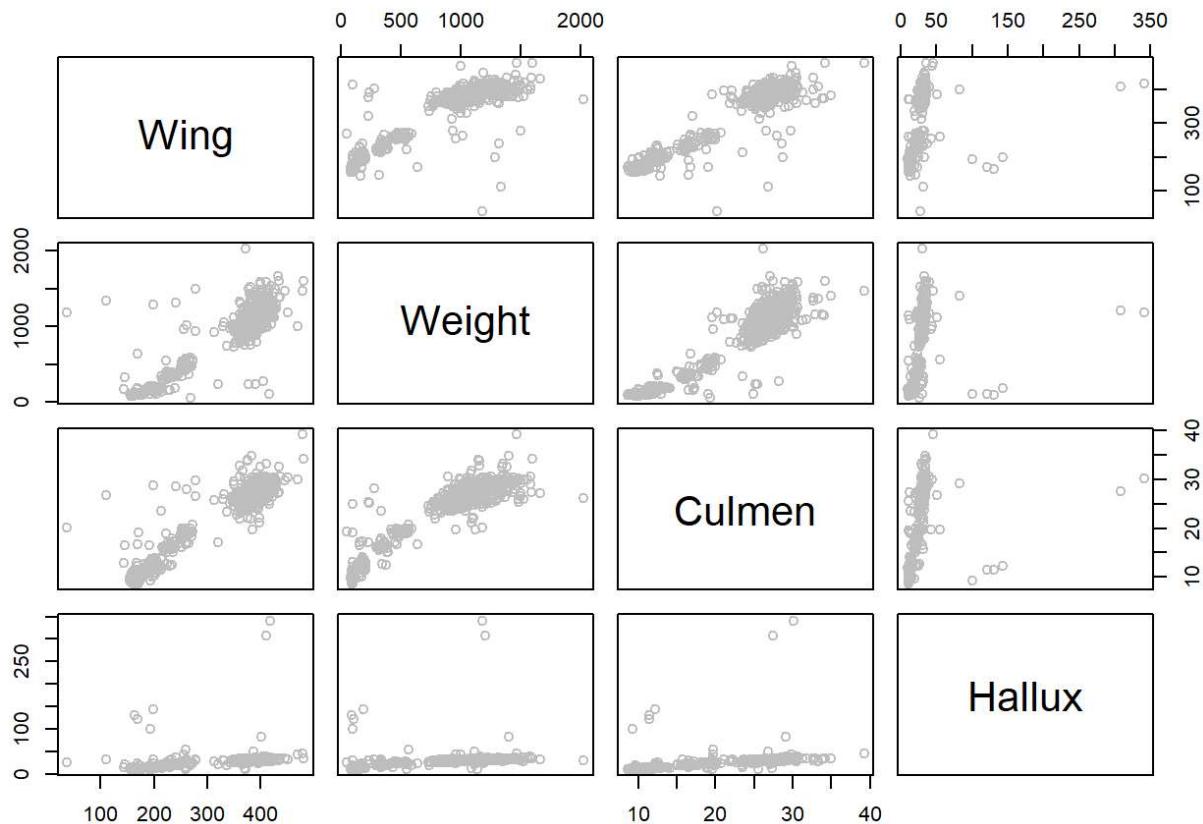


```
### Dibujo de Las trazas que relacionan puntos (res10)
plot(Hawks_data[c(3,4)], col = "grey")
polygon(Hawks_data[res10$order,])
```



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,])
```



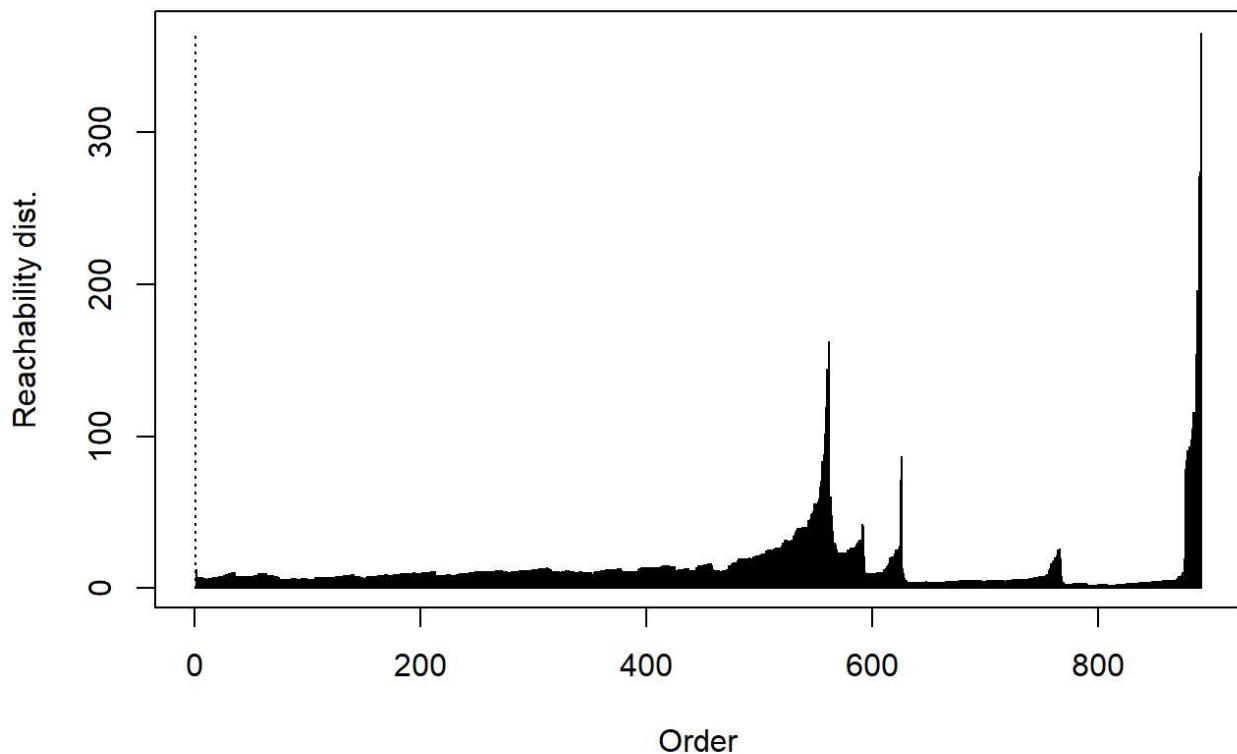
Ahora vamos a probar la alcanzabilidad probando distintos valores de epsilon (0,065).

```
### 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
```

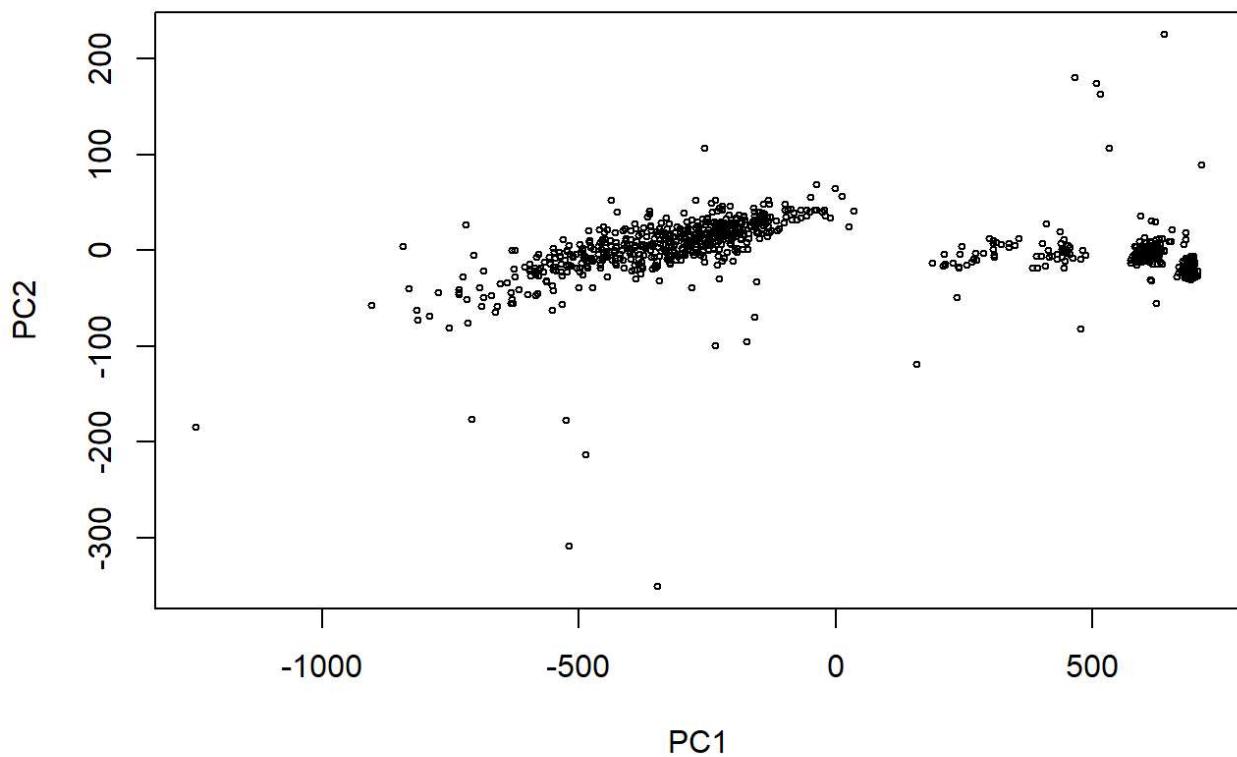
```
plot(res)
```

Reachability Plot



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

Convex Cluster Hulls

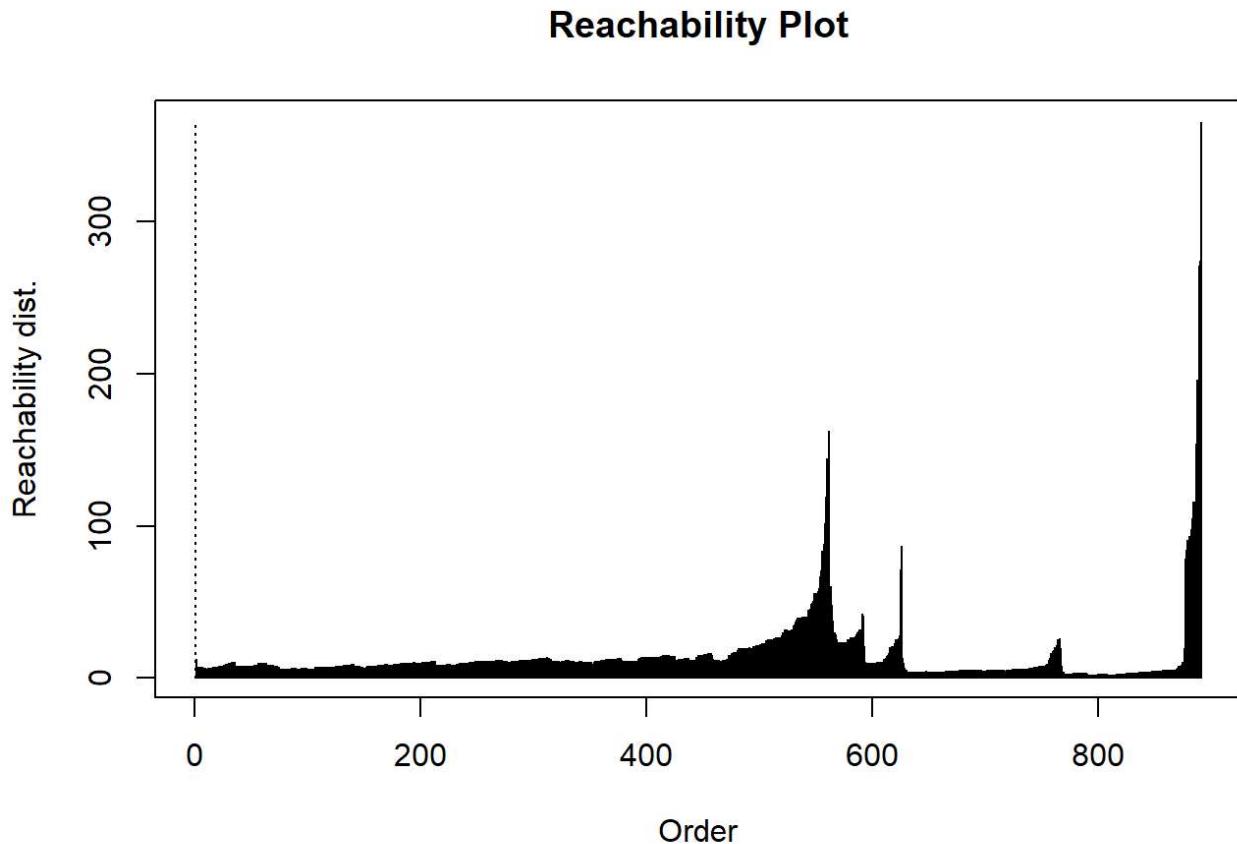


Repetimos el experimento anterior incrementando el parámetro *eps_cl*

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

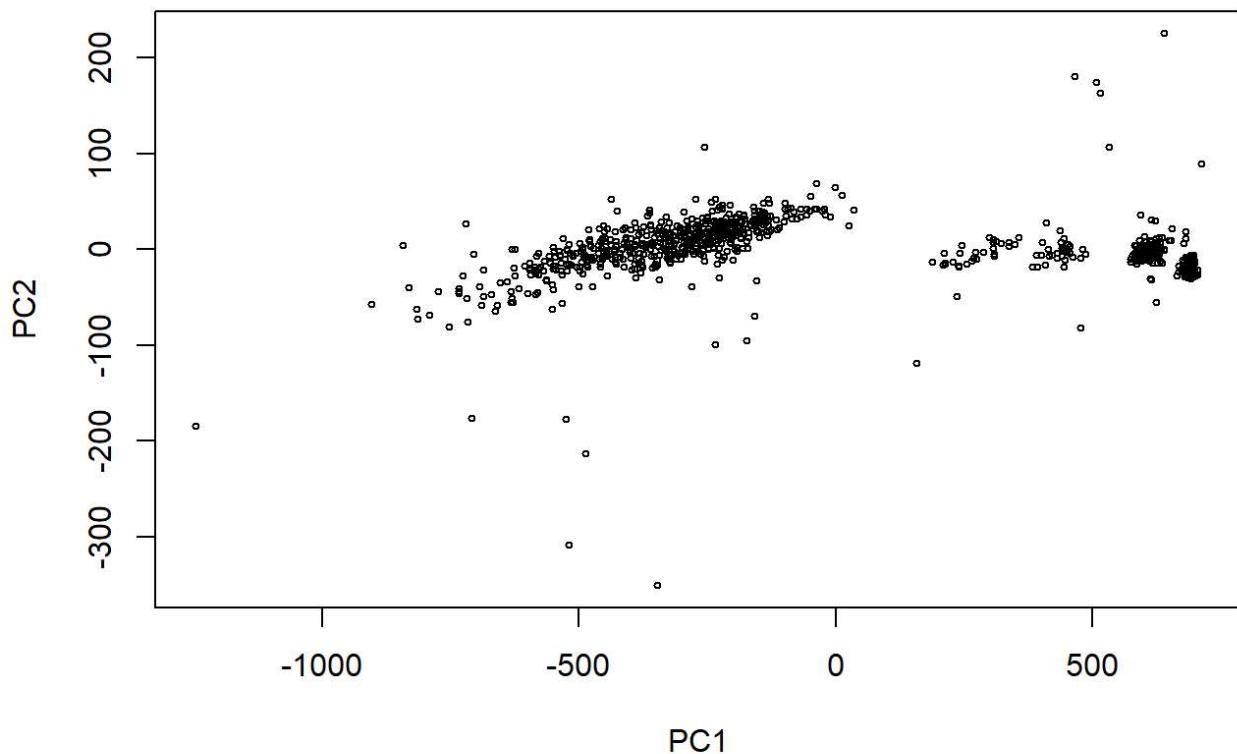
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 10, eps = 531.584706326283, eps_cl = 0.1, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster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

```
plot(res)
```



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

Convex Cluster Hulls



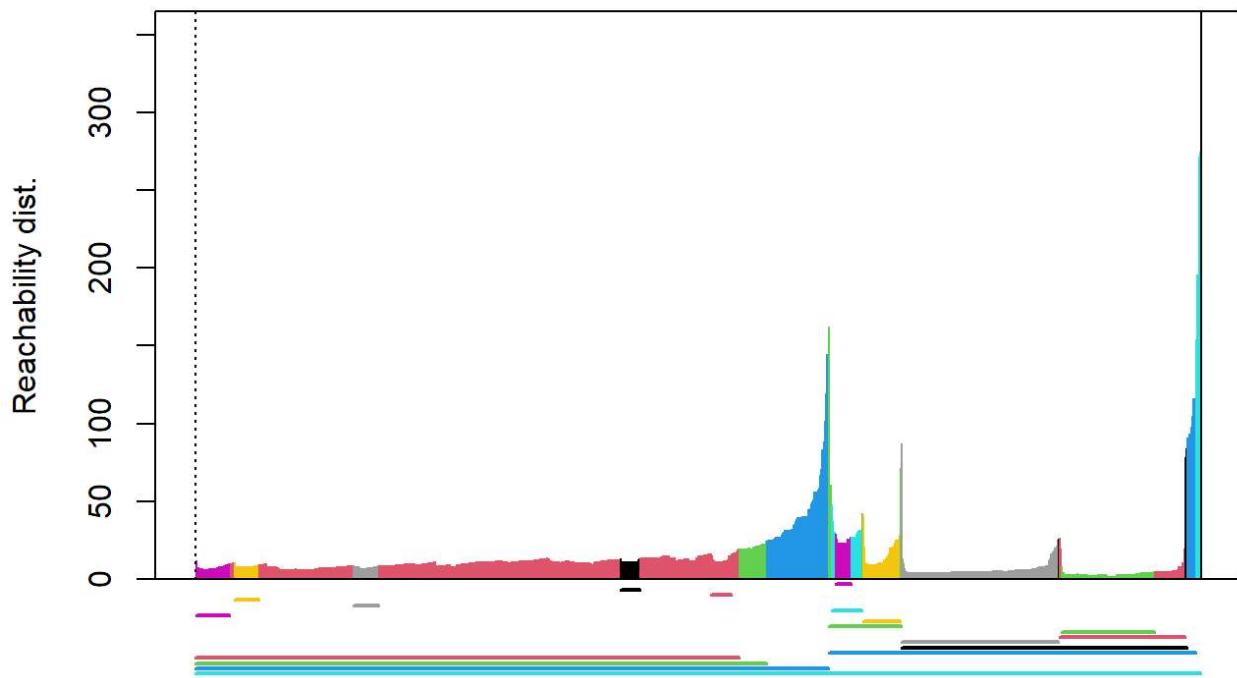
Veamos ahora una variante de la extracción **DBSCN** anterior. En ella el parámetro ξ nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

```
### 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
```

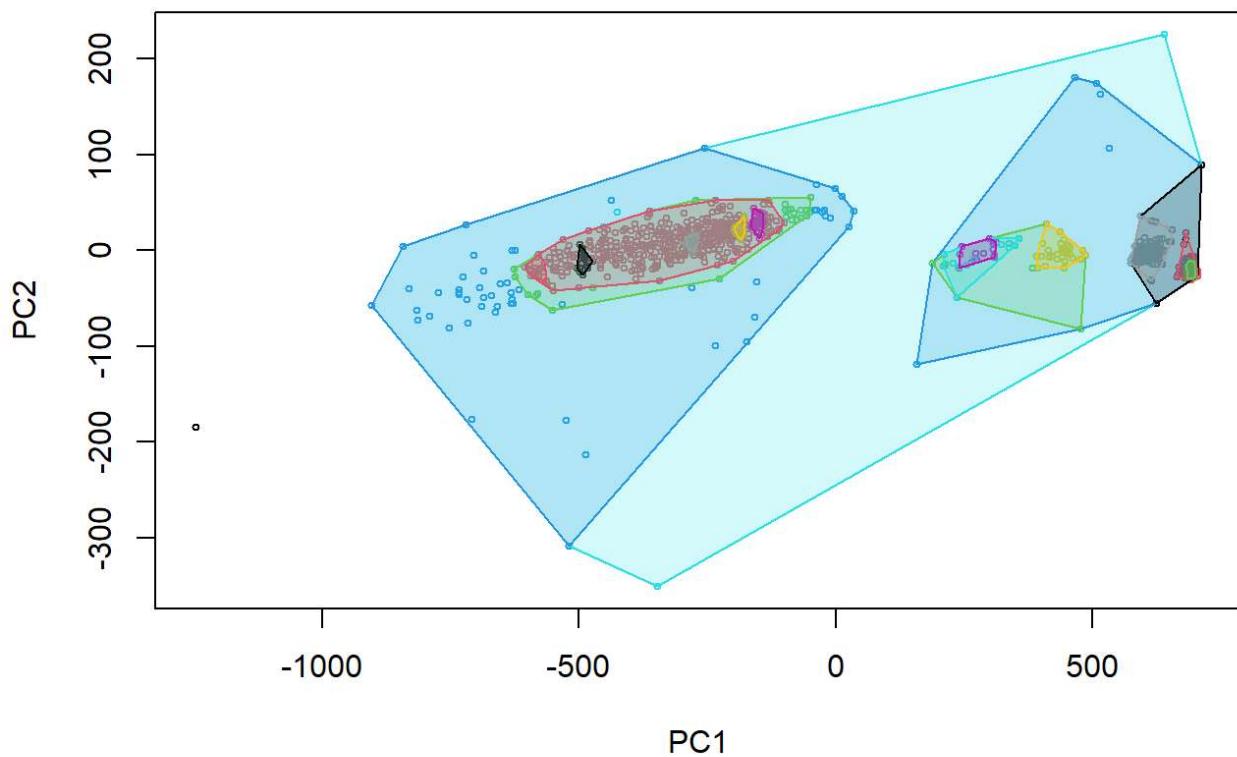
```
plot(res)
```

Reachability Plot



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

Convex Cluster Hulls

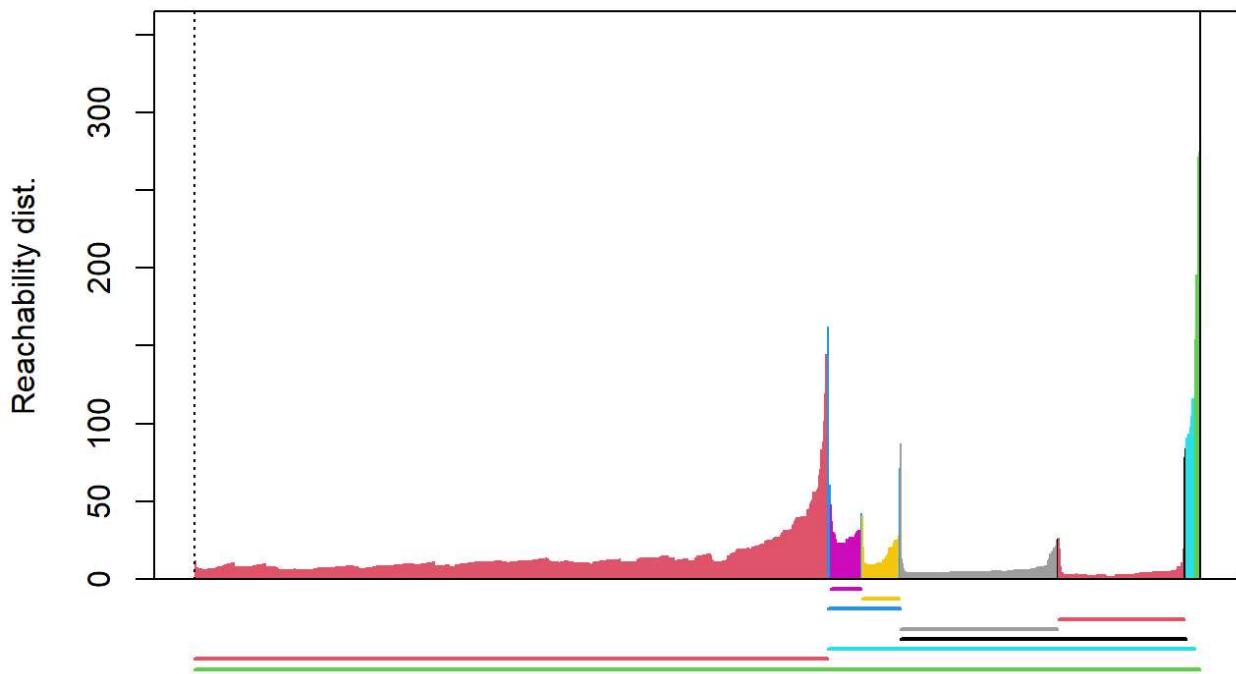


```
### Extracción del clustering jerárquico en función de La variación de La densidad por el método xi
res <- extractXi(res10, xi = 0.1)
res
```

```
## 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
```

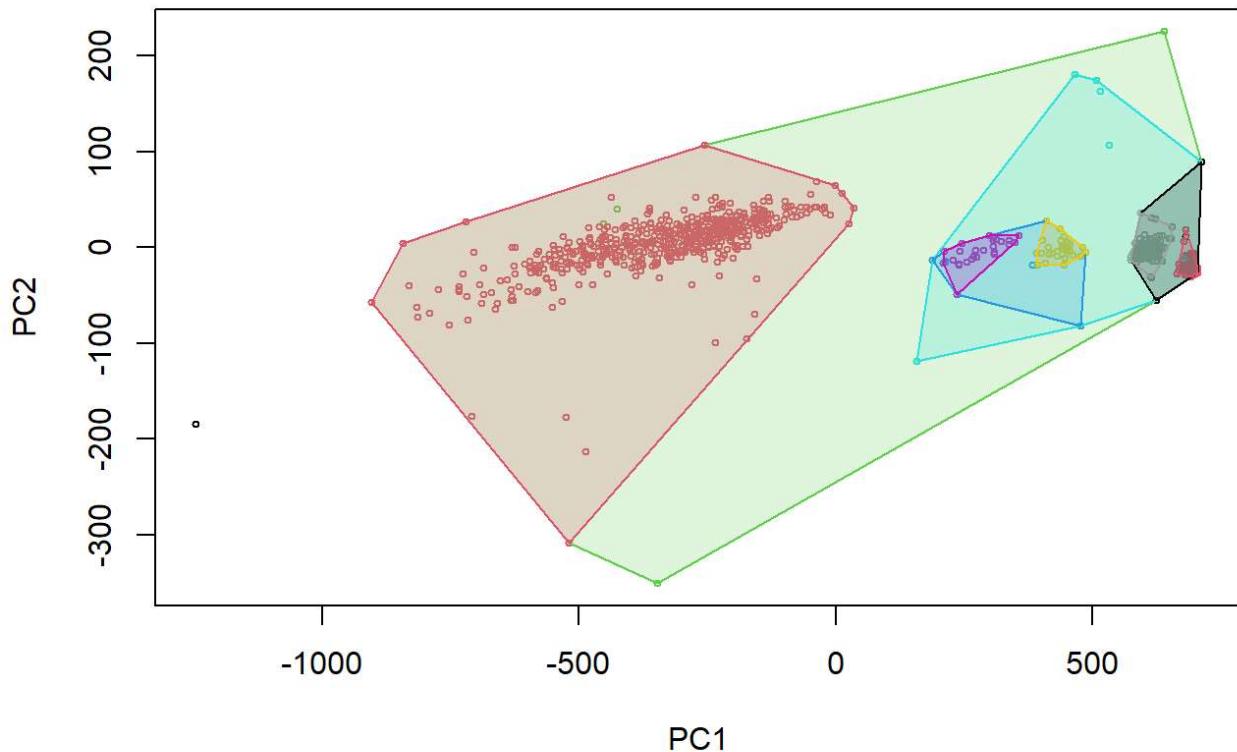
```
plot(res)
```

Reachability Plot



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

Convex Cluster Hulls

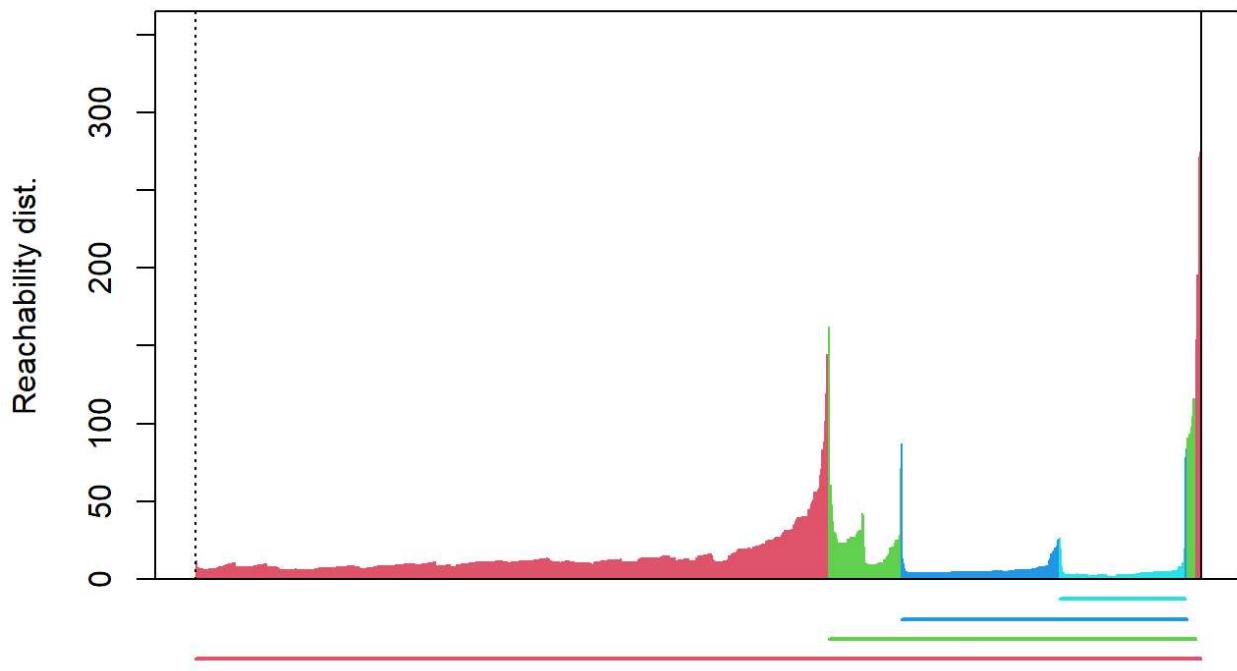


```
### Extracción del clustering jerárquico en función de La variación de La densidad por e  
l método xi  
res <- extractXi(res10, xi = 0.2)  
res
```

```
## 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
```

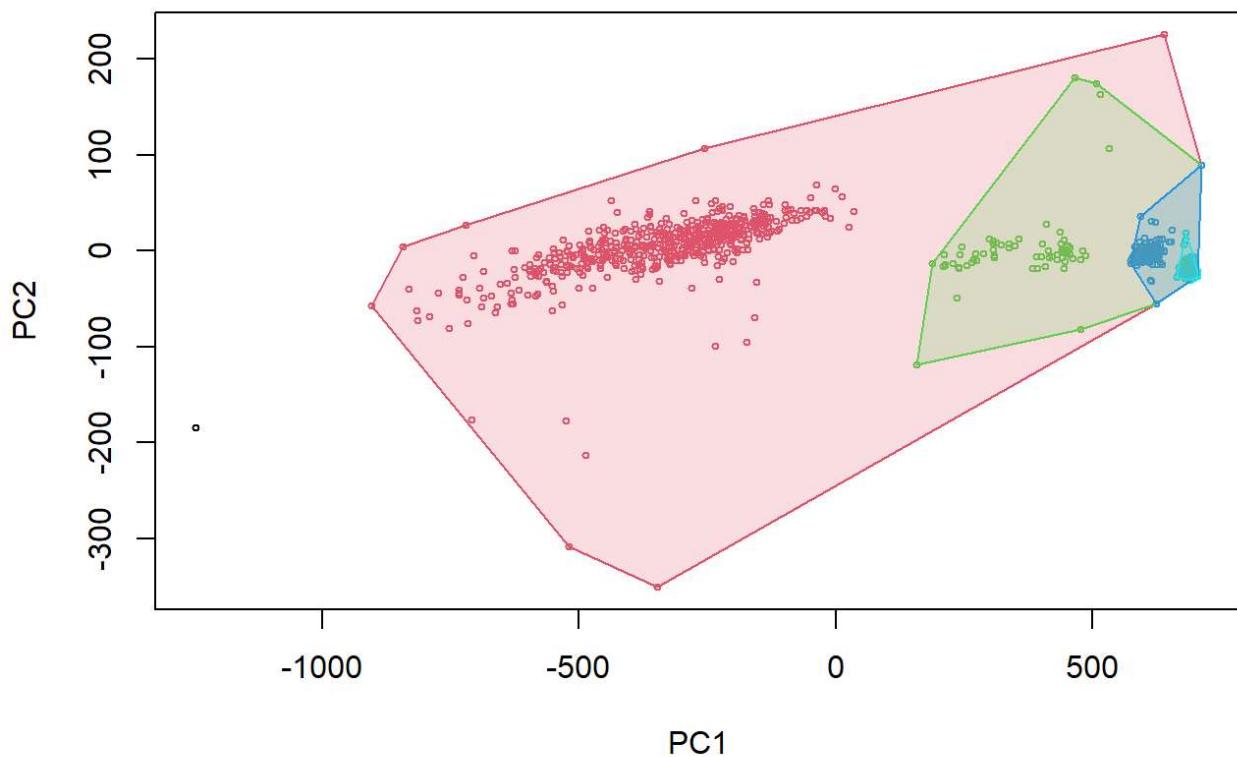
```
plot(res)
```

Reachability Plot



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

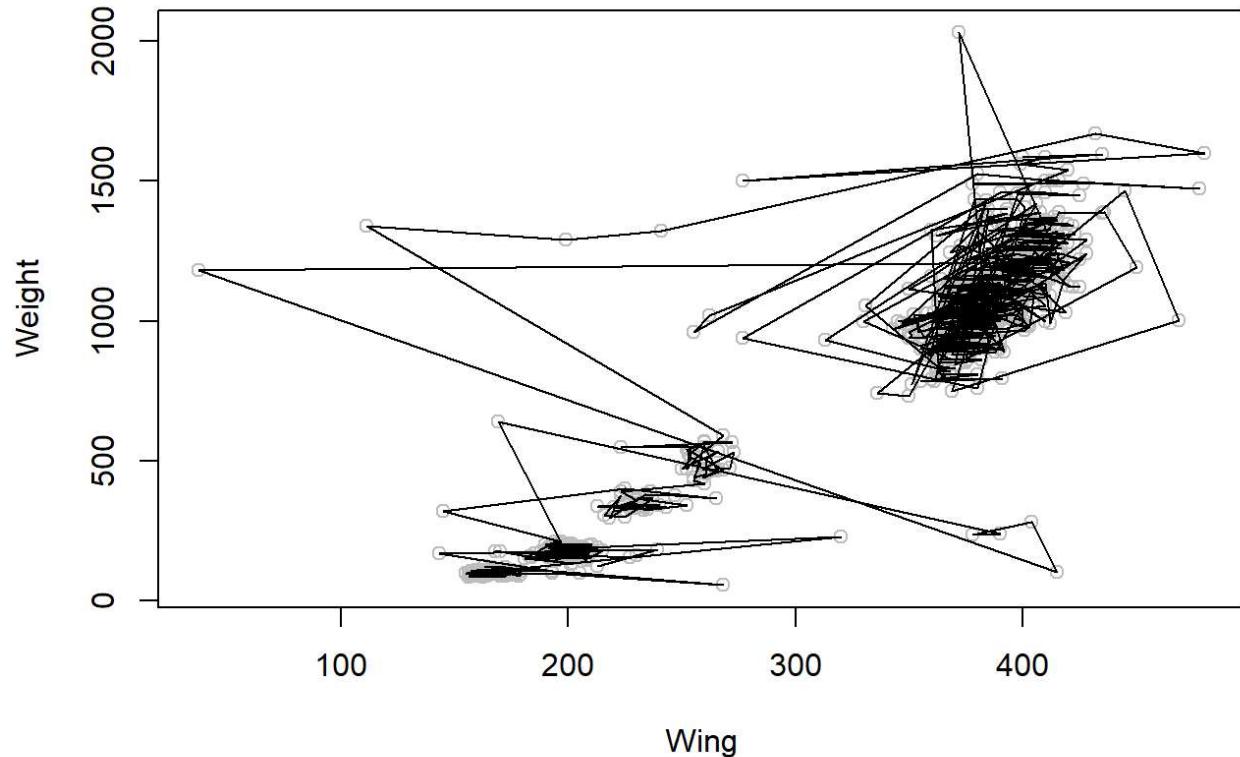
Convex Cluster Hulls



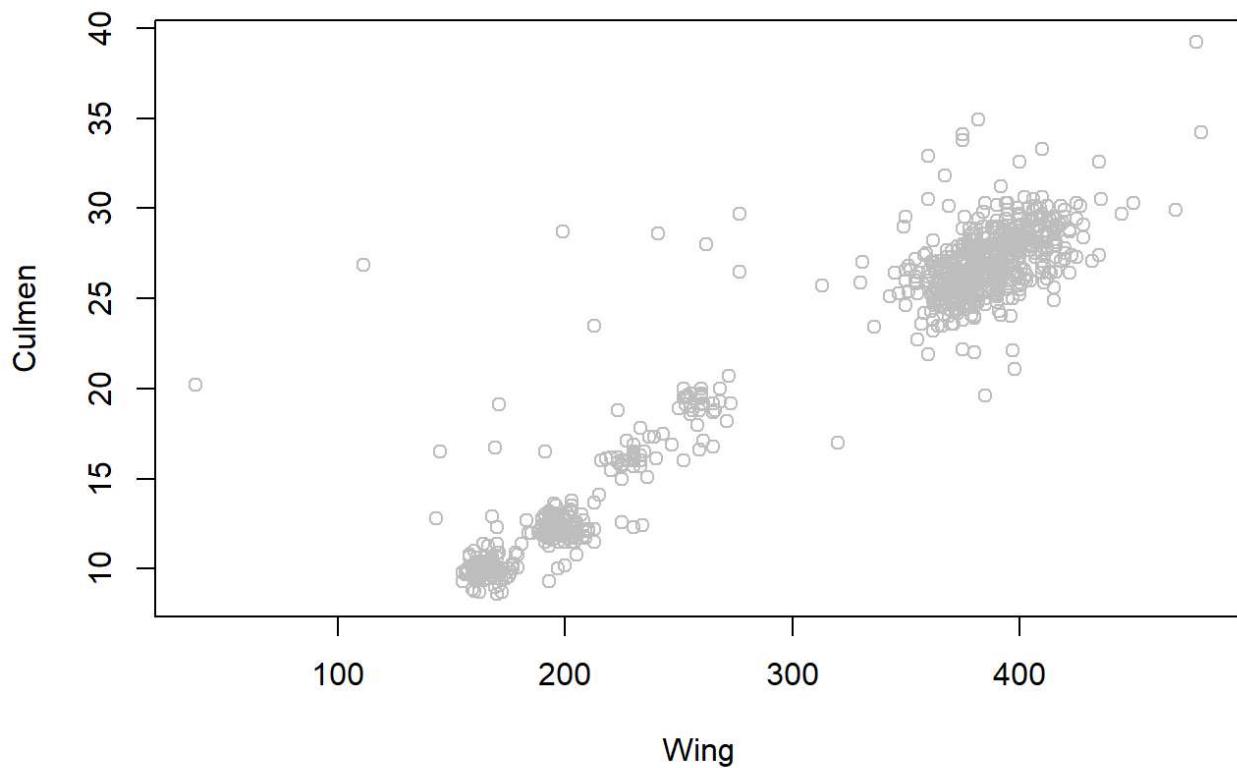
Como se puede observar, con los distintos valores de x_i el numero de clúster que obtenemos es distintitos, mientras más grande es la distancia menos clústeres necesitan para clasificar los valores.

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

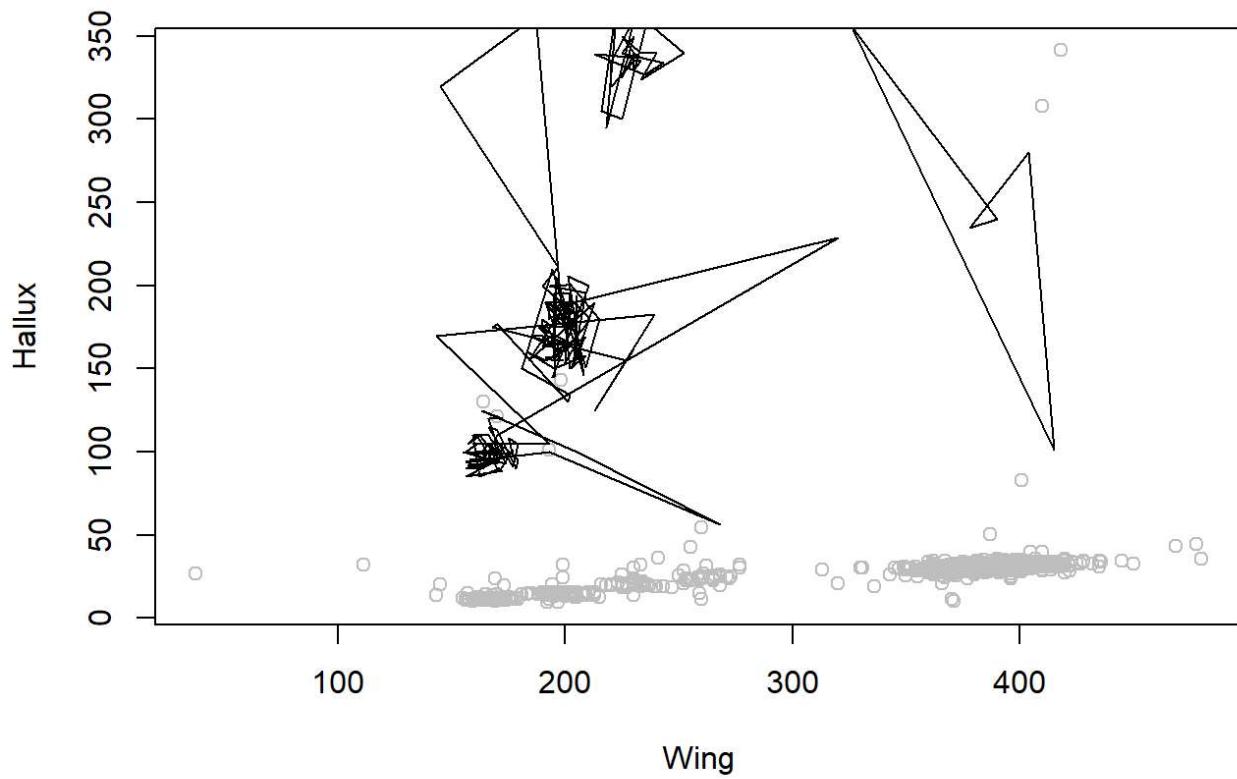
```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data[c(1,2)], col = "grey")
polygon(Hawks_data[res20$order,])
```



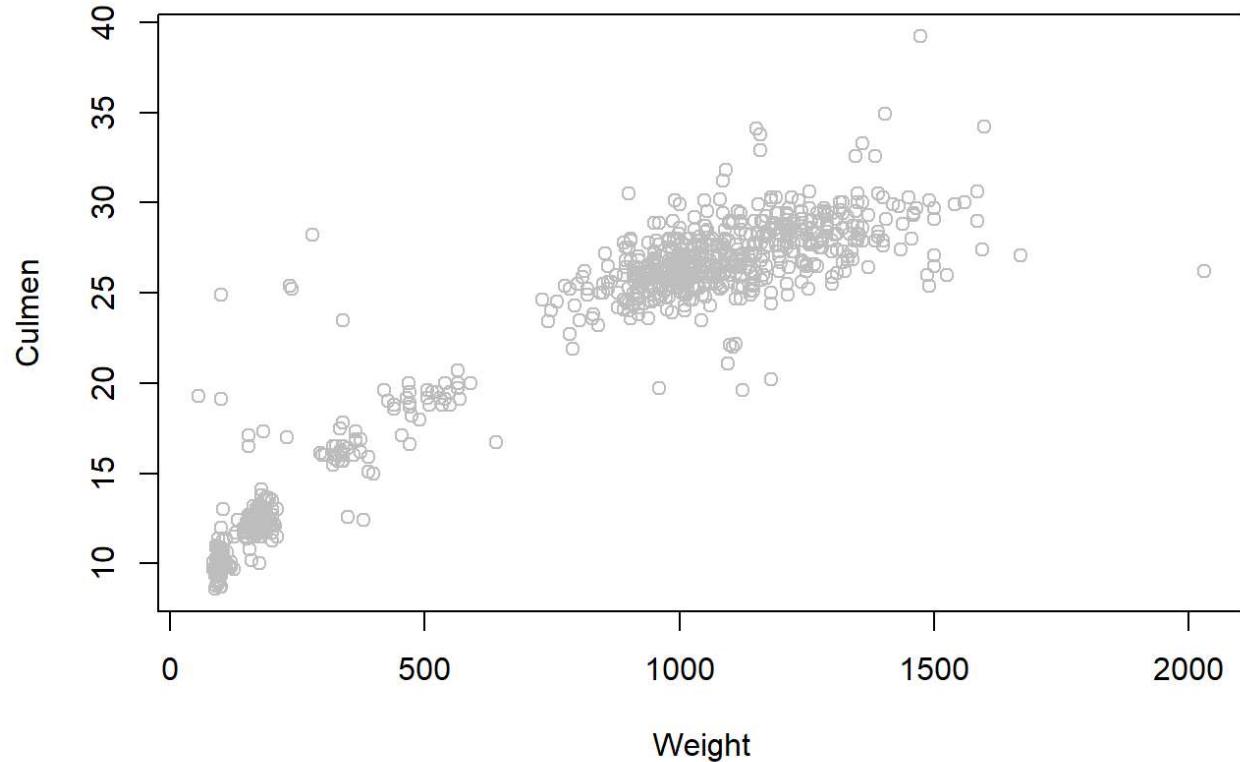
```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data[c(1,3)], col = "grey")
polygon(Hawks_data[res20$order,])
```



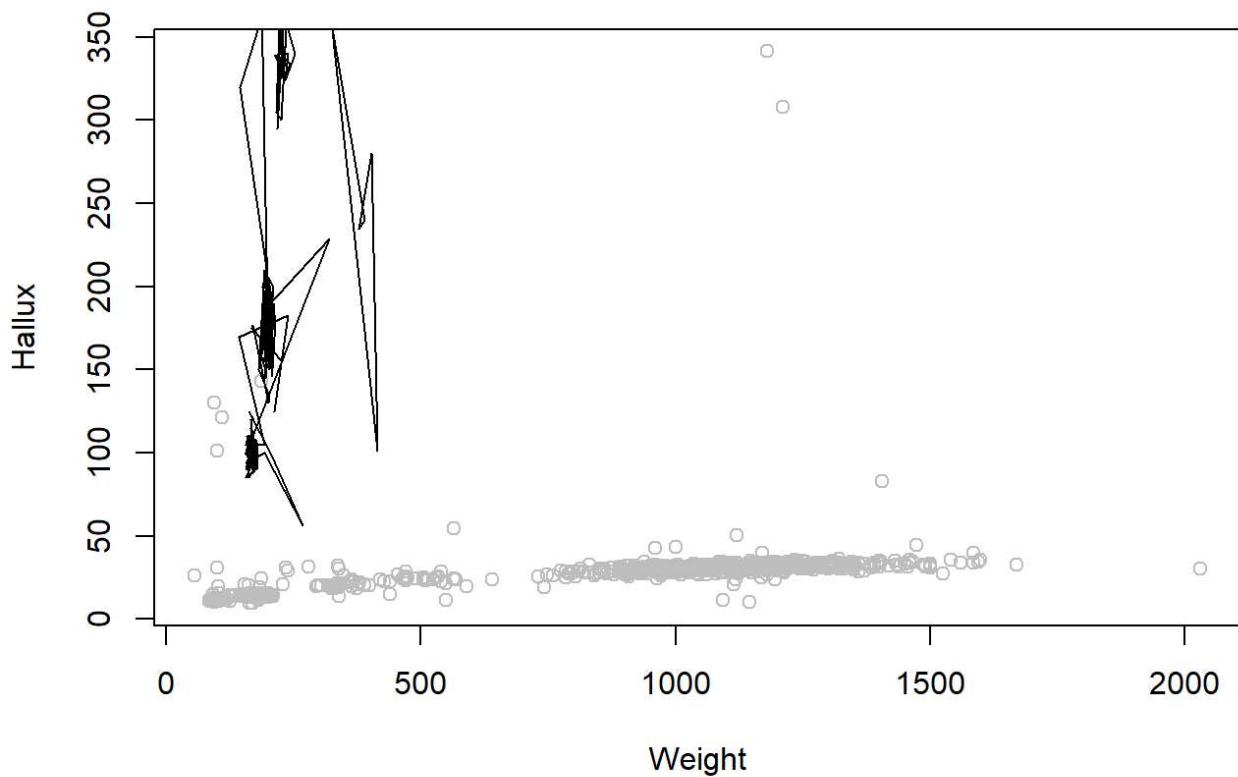
```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data[c(1,4)], col = "grey")
polygon(Hawks_data[res20$order,])
```



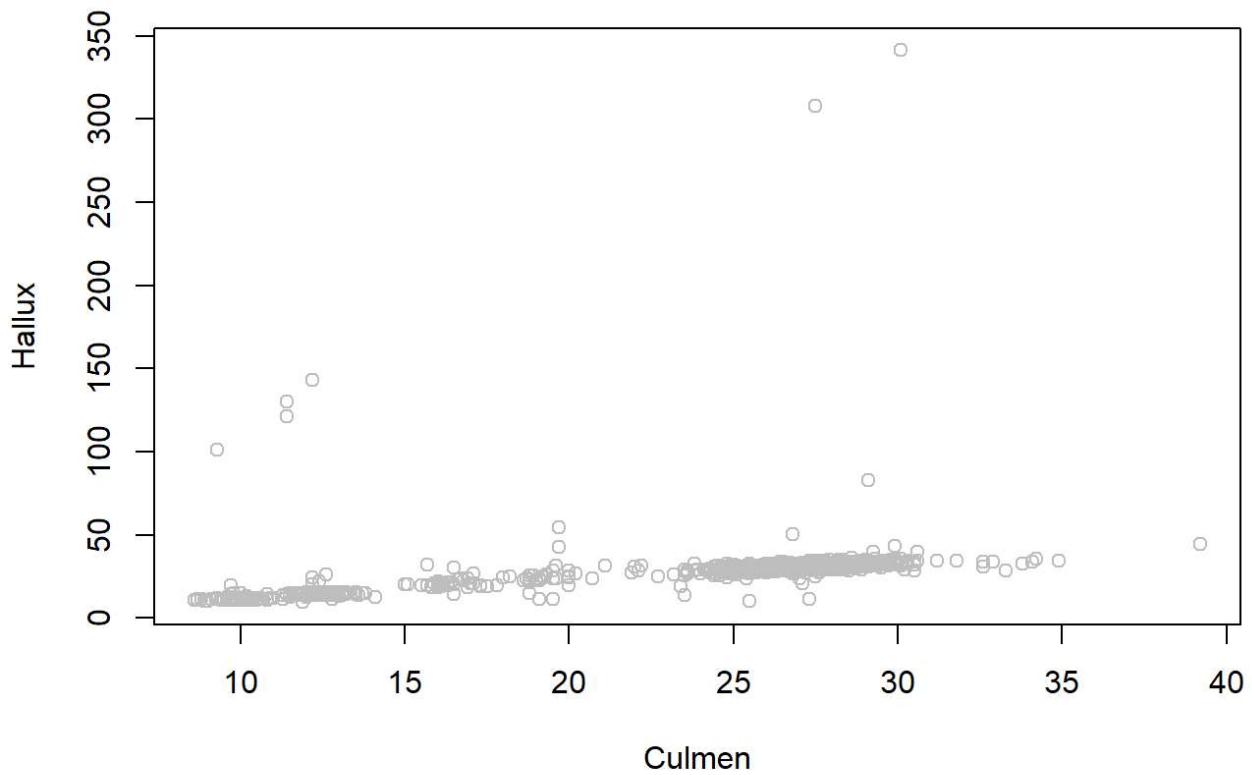
```
### Dibujo de las trazas que relacionan puntos (res20)
plot(Hawks_data[c(2,3)], col = "grey")
polygon(Hawks_data[res20$order,])
```



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

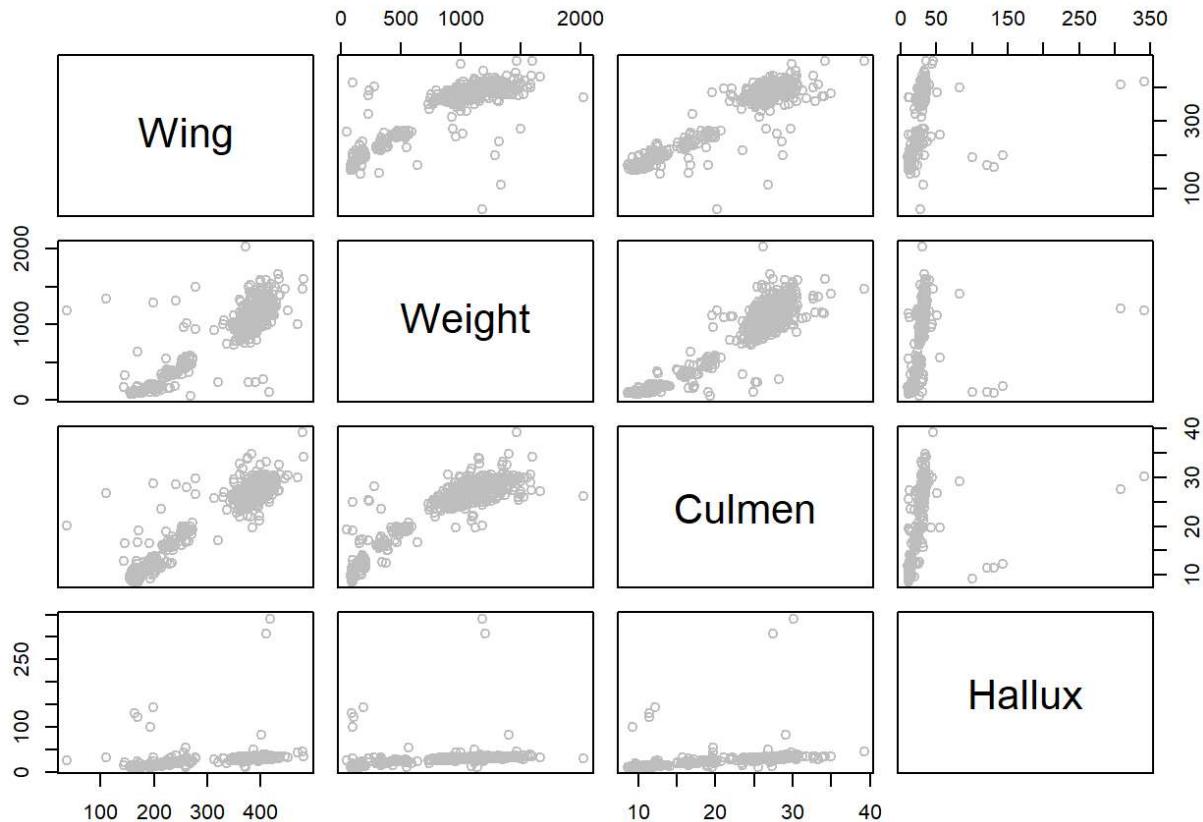


```
### Dibujo de Las trazas que relacionan puntos (res20)
plot(Hawks_data[c(3,4)], col = "grey")
polygon(Hawks_data[res20$order,])
```



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,])
```



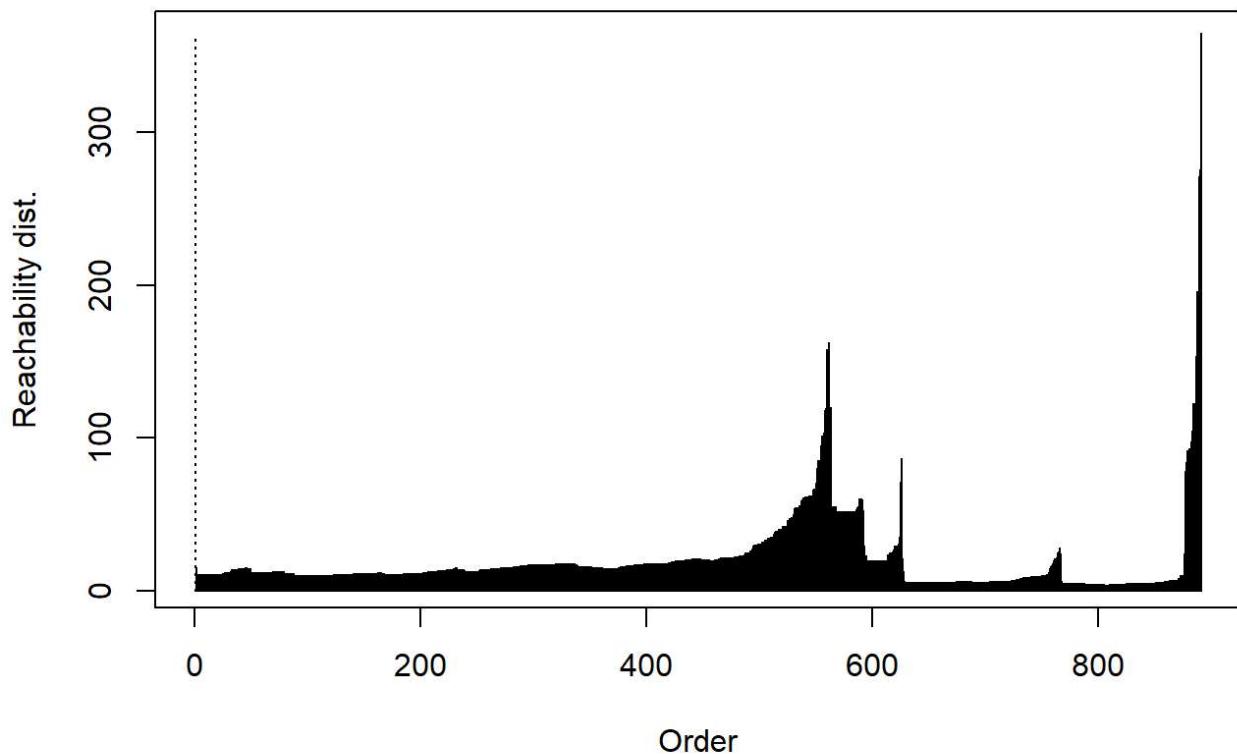
Ahora vamos a probar la alcanzabilidad probando distintos valores de epsilon.

```
### 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
```

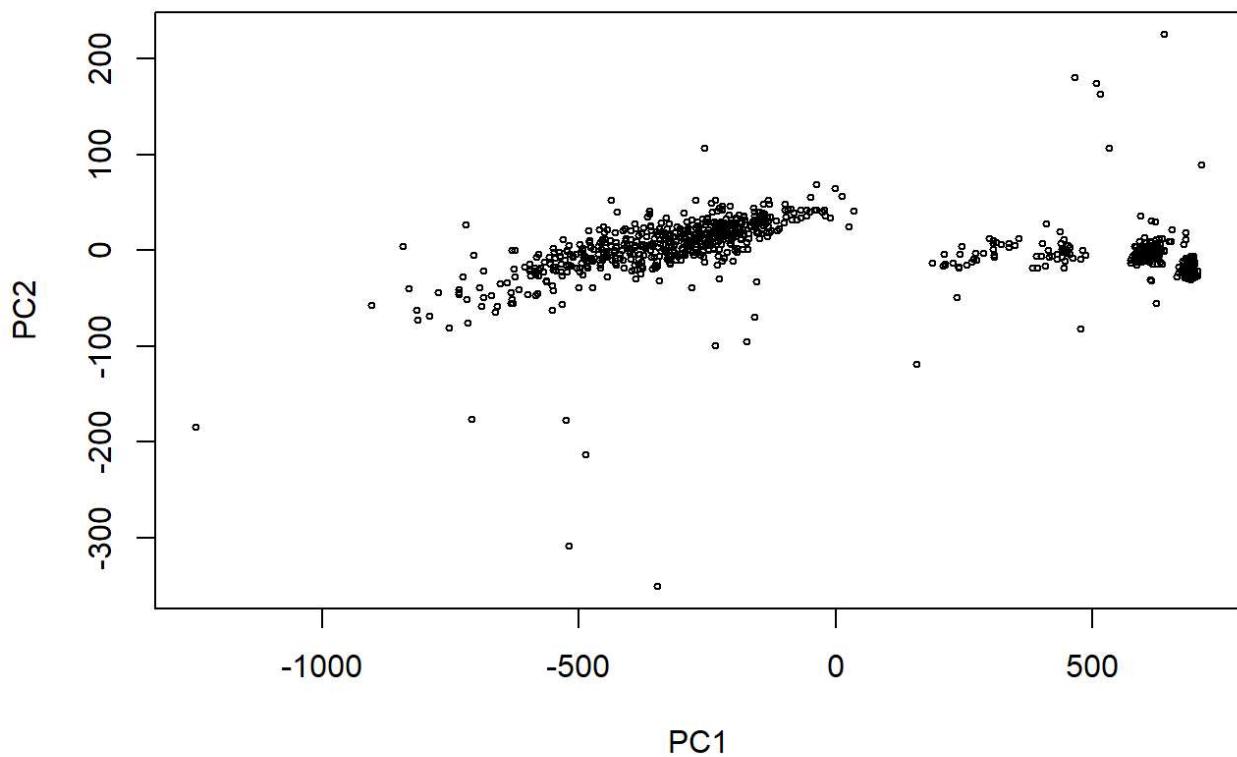
```
plot(res)
```

Reachability Plot



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

Convex Cluster Hulls

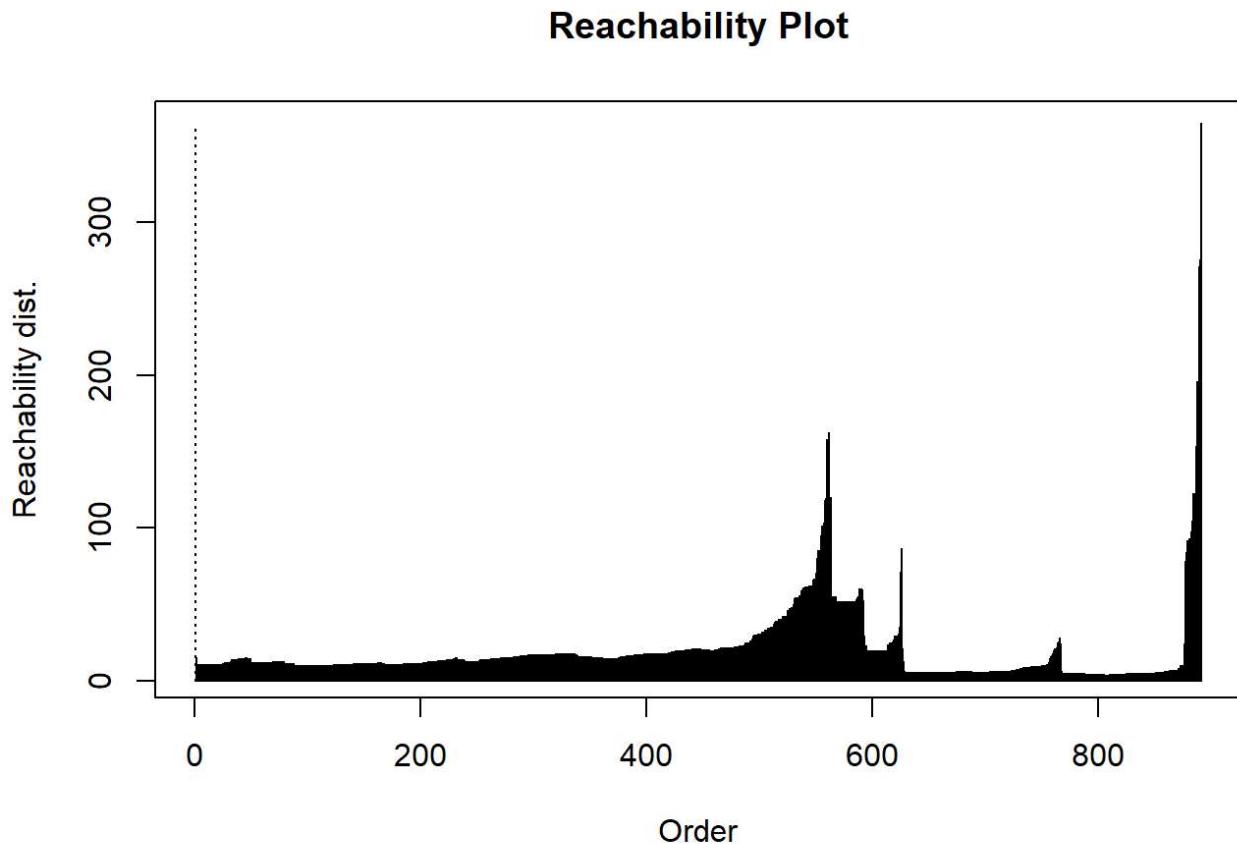


Repetimos el experimento anterior incrementando el parámetro *eps_cl*

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

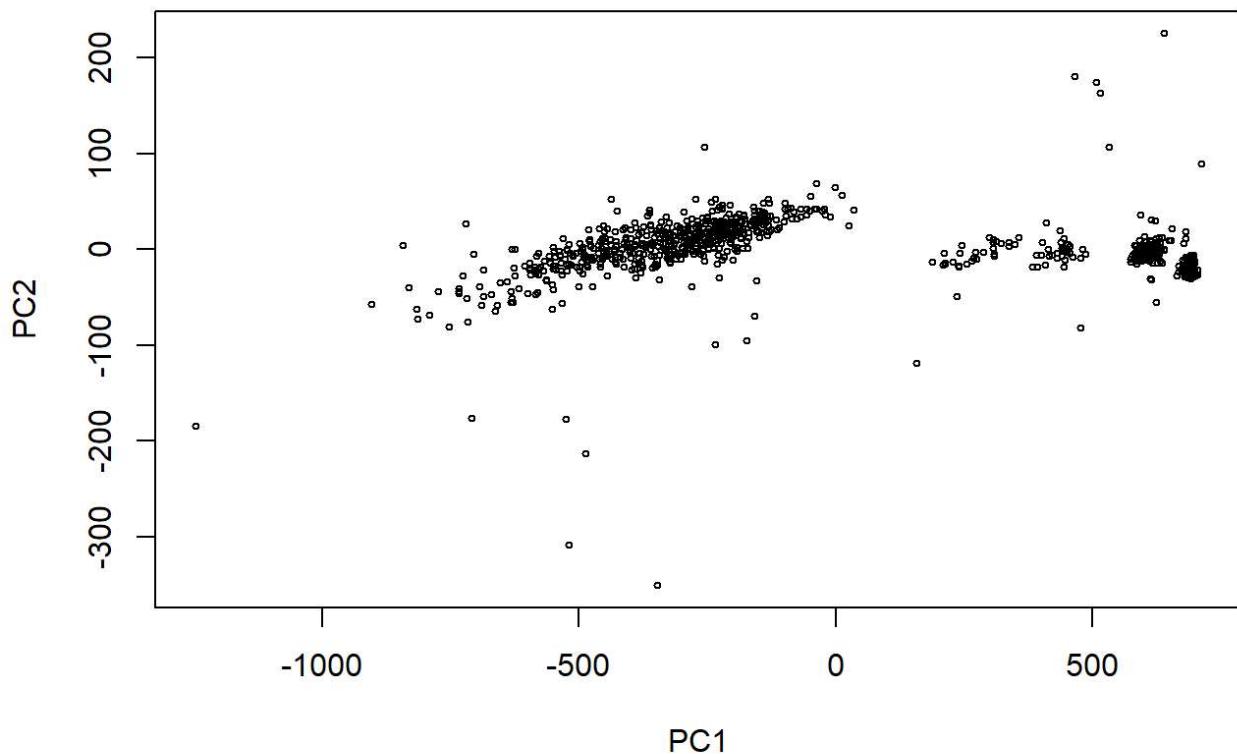
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 20, eps = 575.591808489315, eps_cl = 0.1, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster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

```
plot(res)
```



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

Convex Cluster Hulls



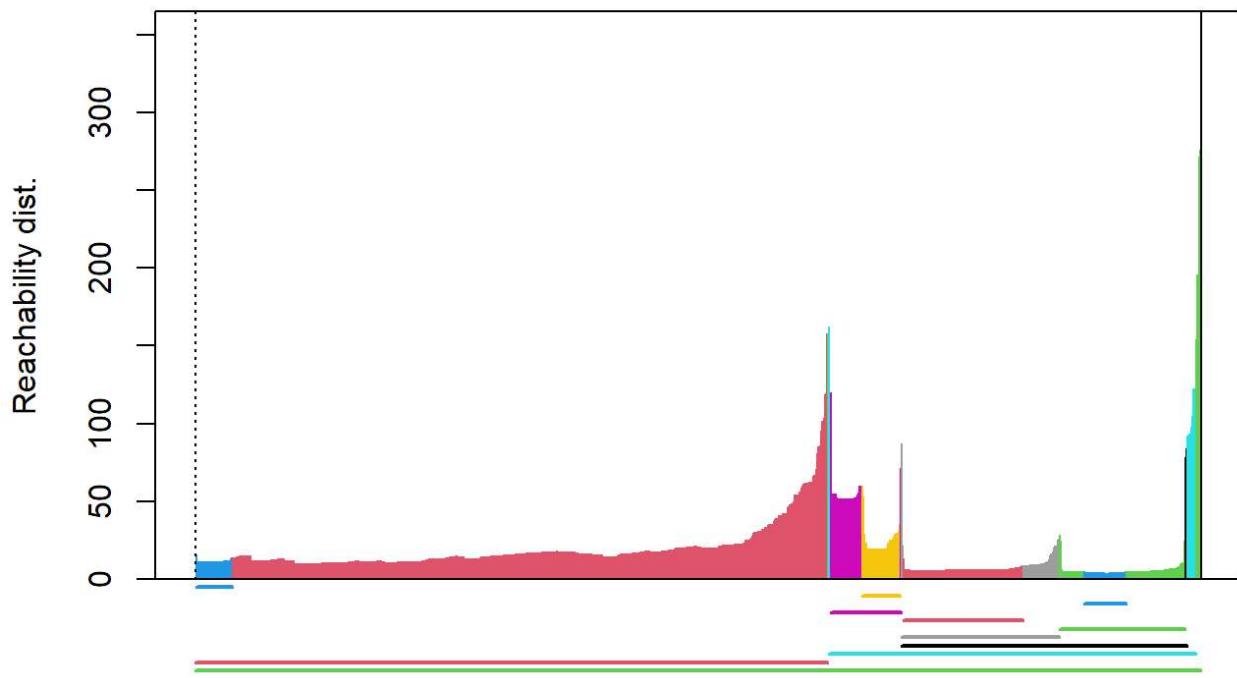
Veamos ahora una variante de la extracción **DBSCN** anterior. En ella el parámetro ξ nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

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

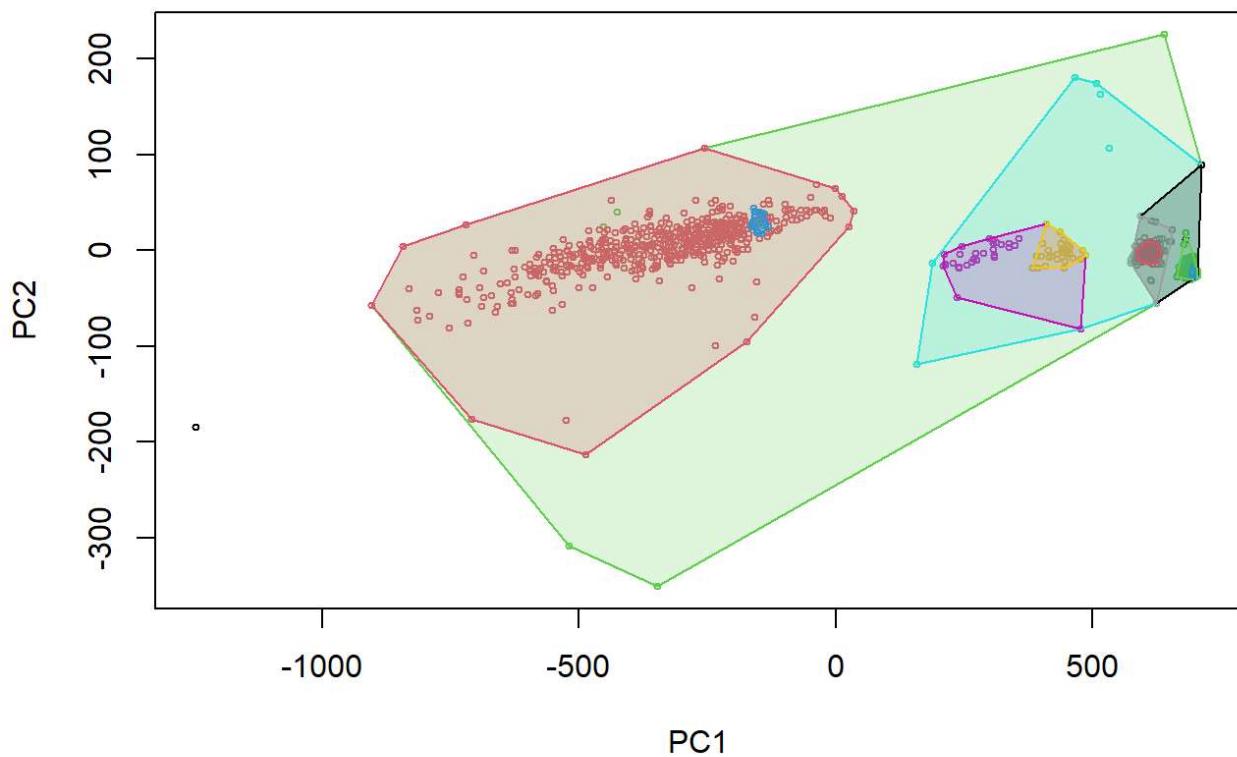
```
plot(res)
```

Reachability Plot



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

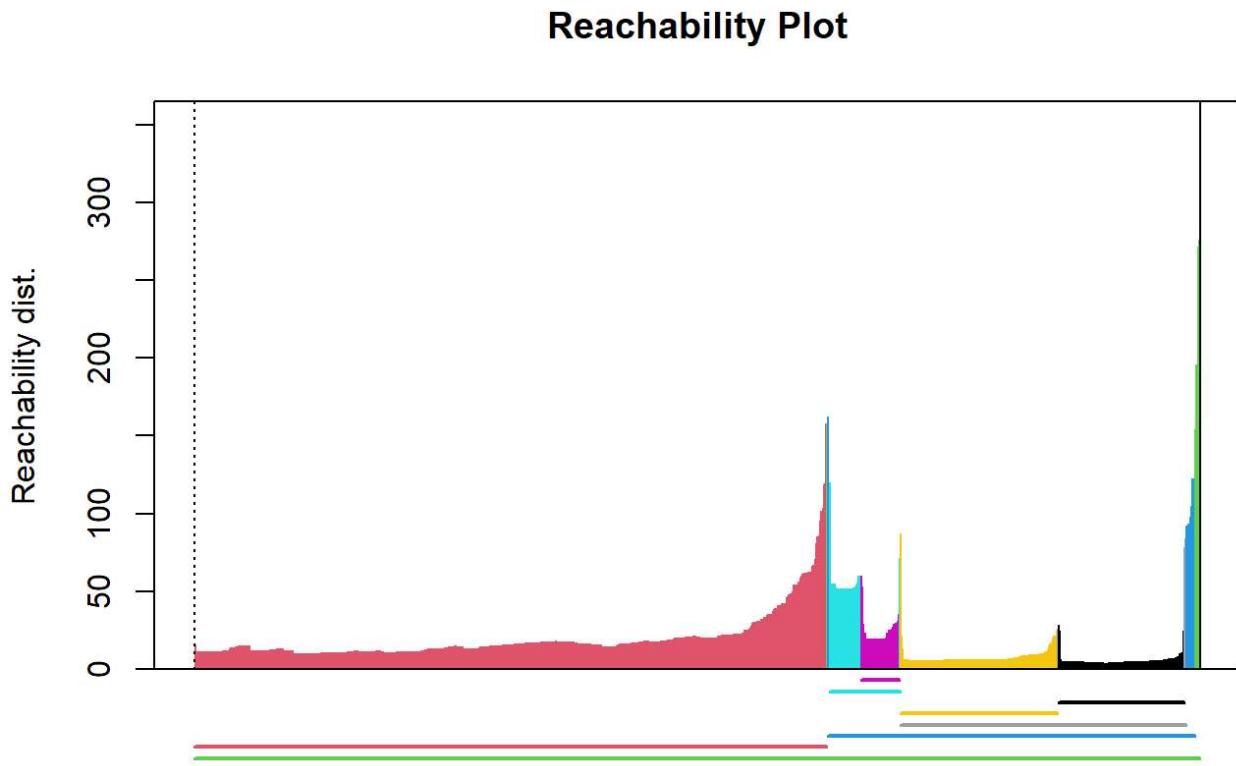
Convex Cluster Hulls



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

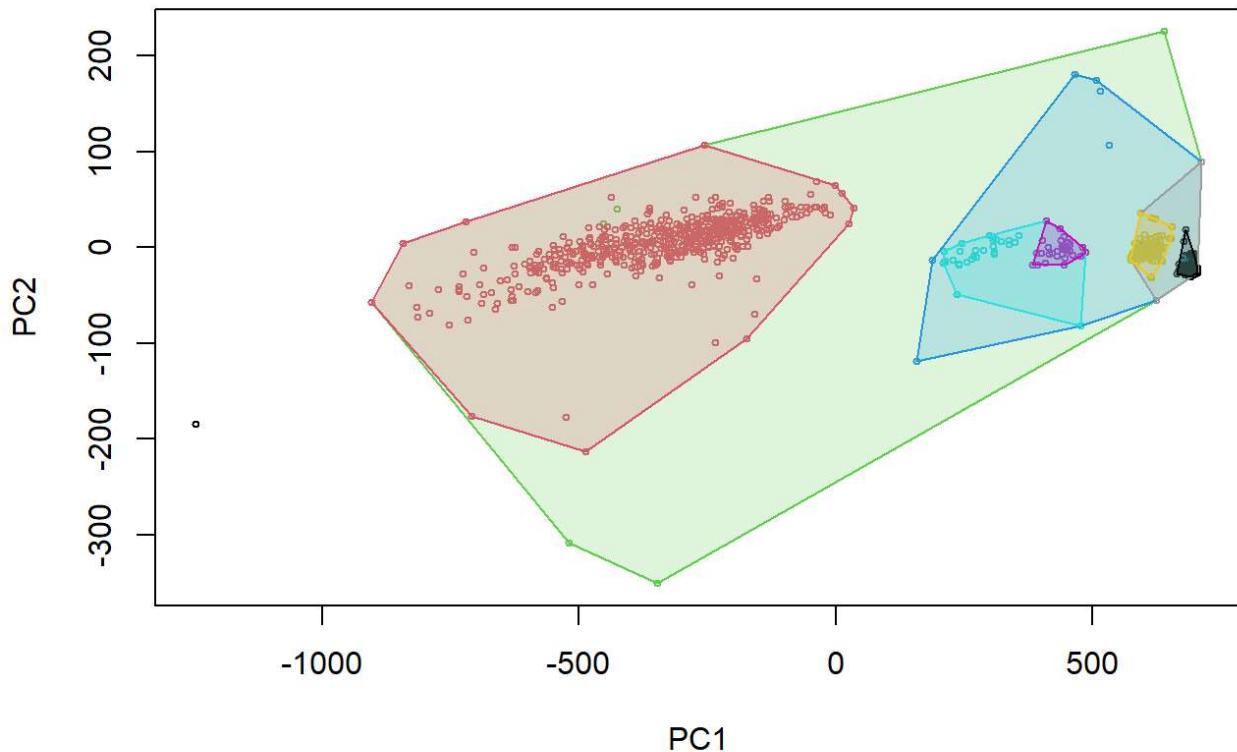
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 20, eps = 575.591808489315, eps_cl = NA, xi = 0.1
## The clustering contains 8 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

```
plot(res)
```



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

Convex Cluster Hulls



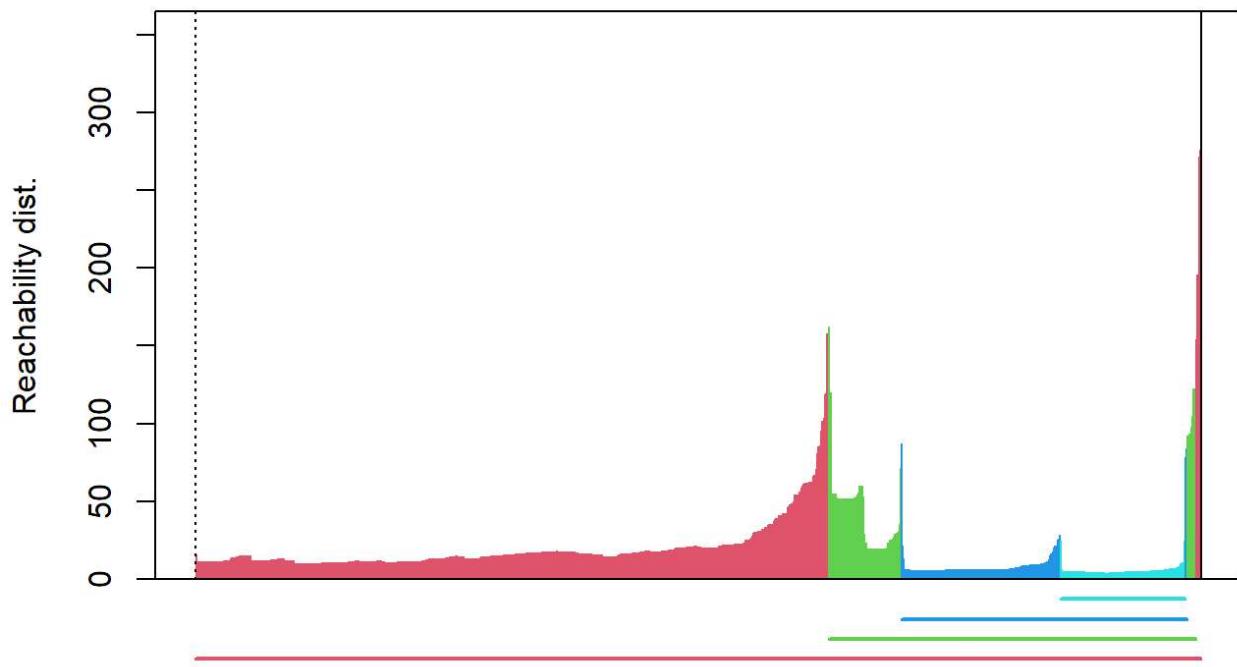
```
### Extracción del clustering jerárquico en función de La variación de La densidad por e  
l método xi
```

```
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
```

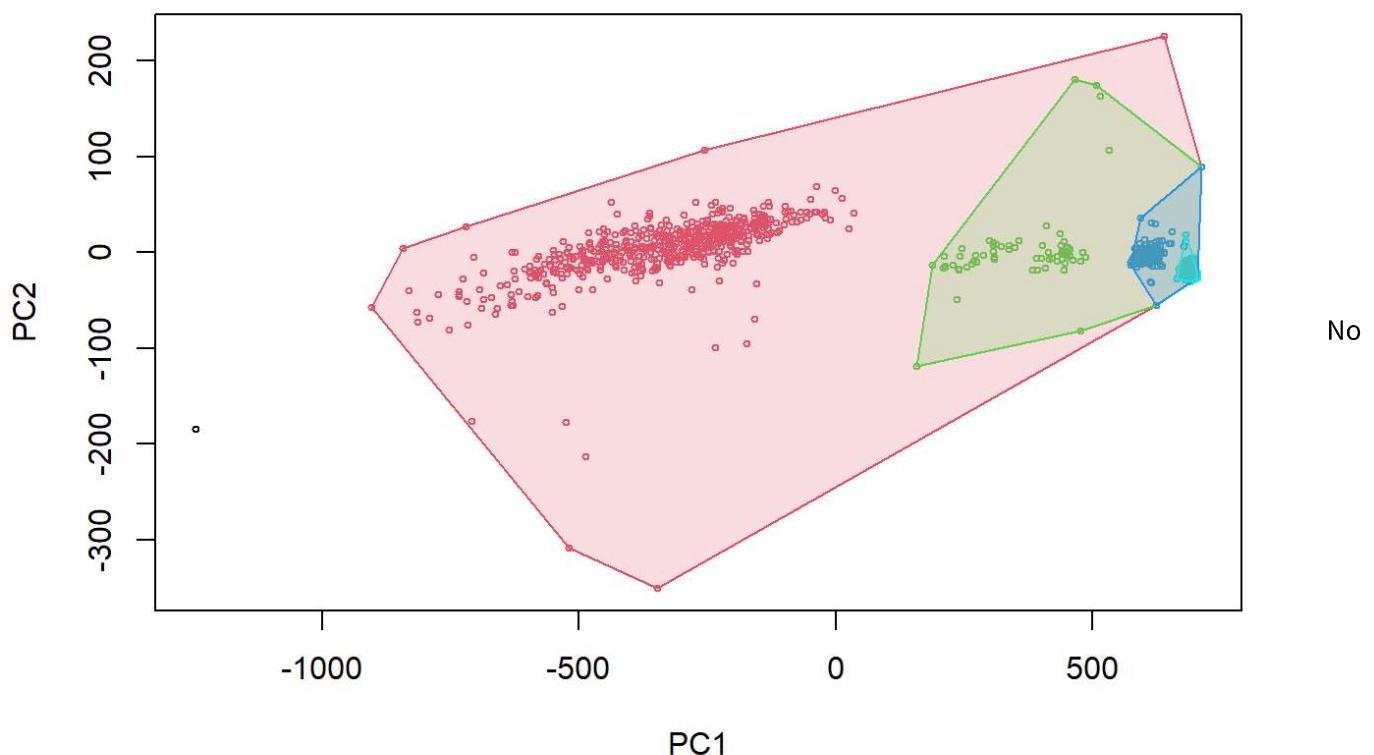
```
plot(res)
```

Reachability Plot



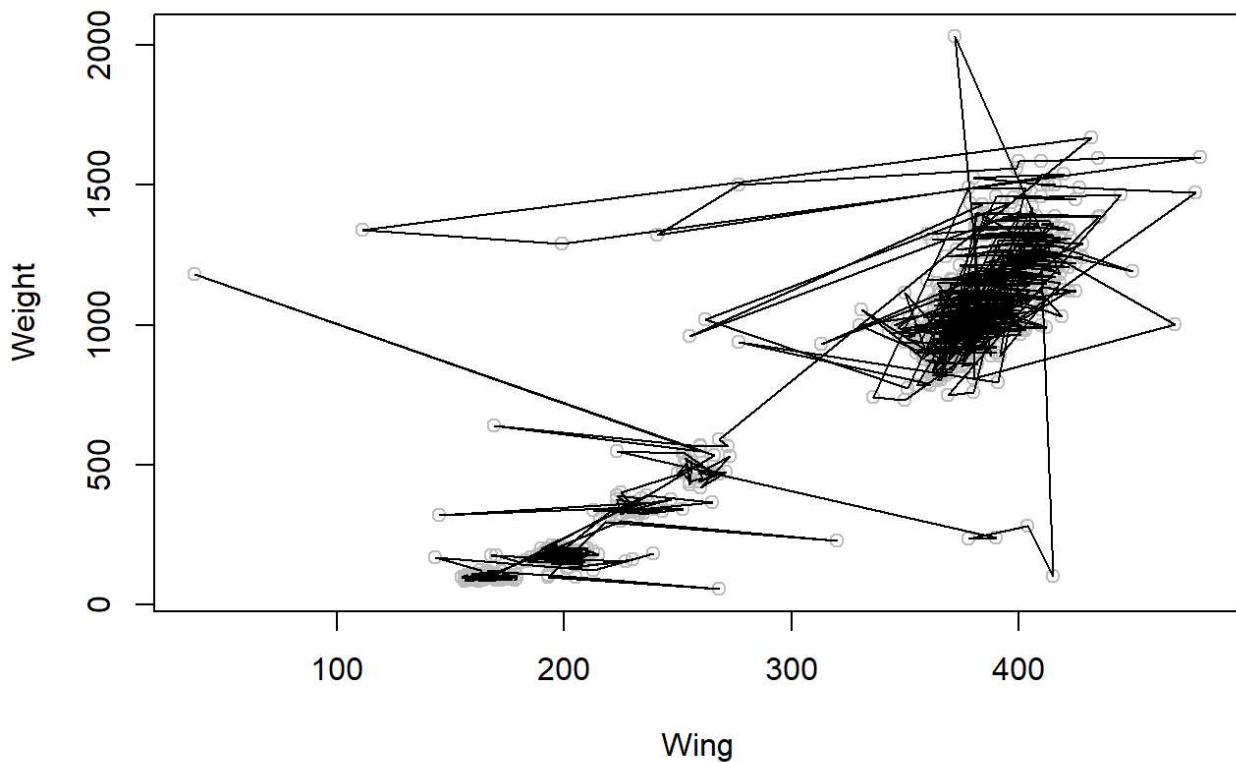
```
hullplot(Hawks_data, res)
```

Convex Cluster Hulls

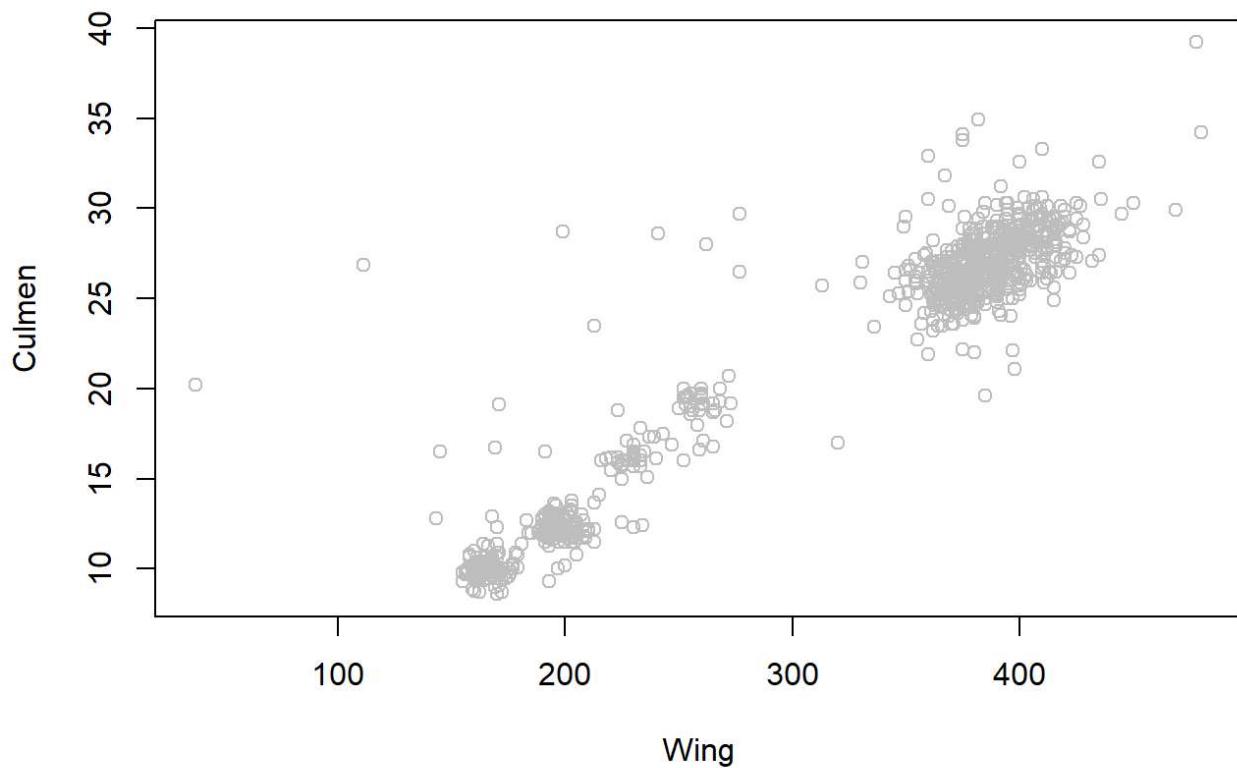


se encuentra mucha diferencia entre una vecindad de 10 y de 20. Ahora se repetirá el proceso con una vecindad de 50.

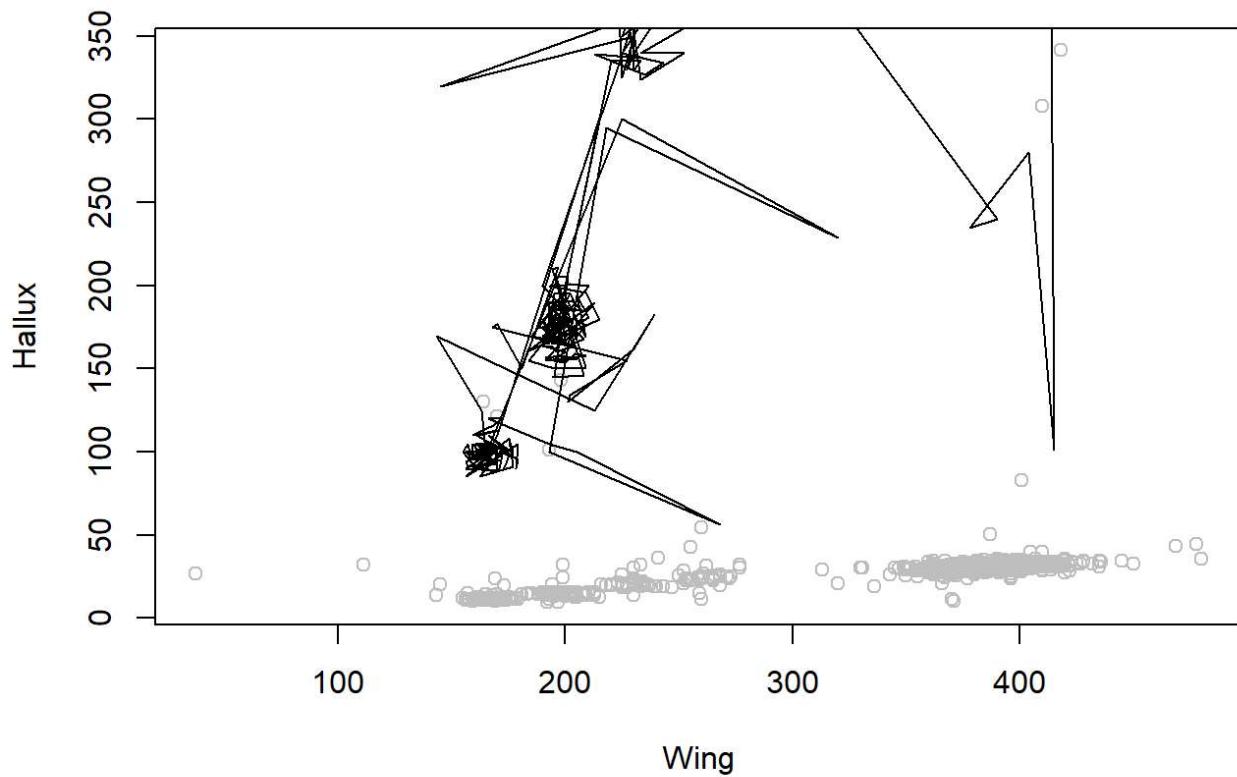
```
### Dibujo de las trazas que relacionan puntos (res50)
plot(Hawks_data[c(1,2)], col = "grey")
polygon(Hawks_data[res50$order,])
```



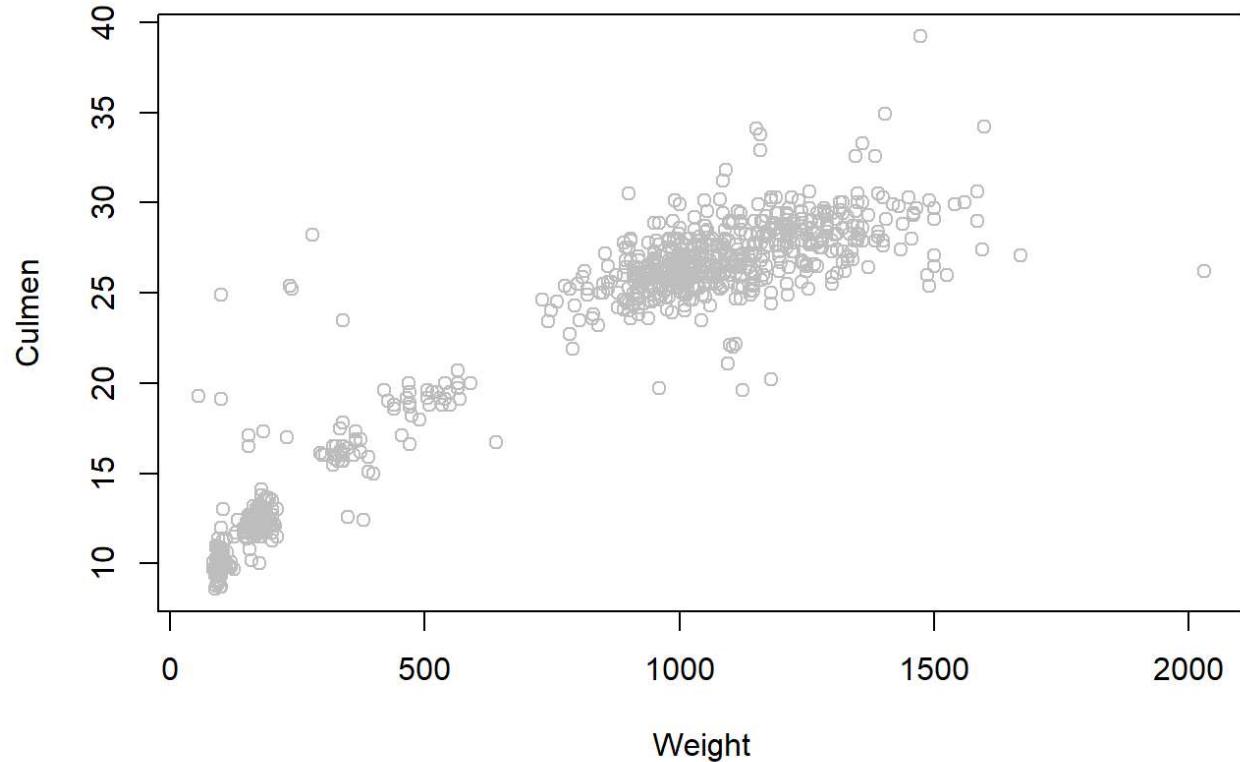
```
### Dibujo de las trazas que relacionan puntos (res50)
plot(Hawks_data[c(1,3)], col = "grey")
polygon(Hawks_data[res50$order,])
```



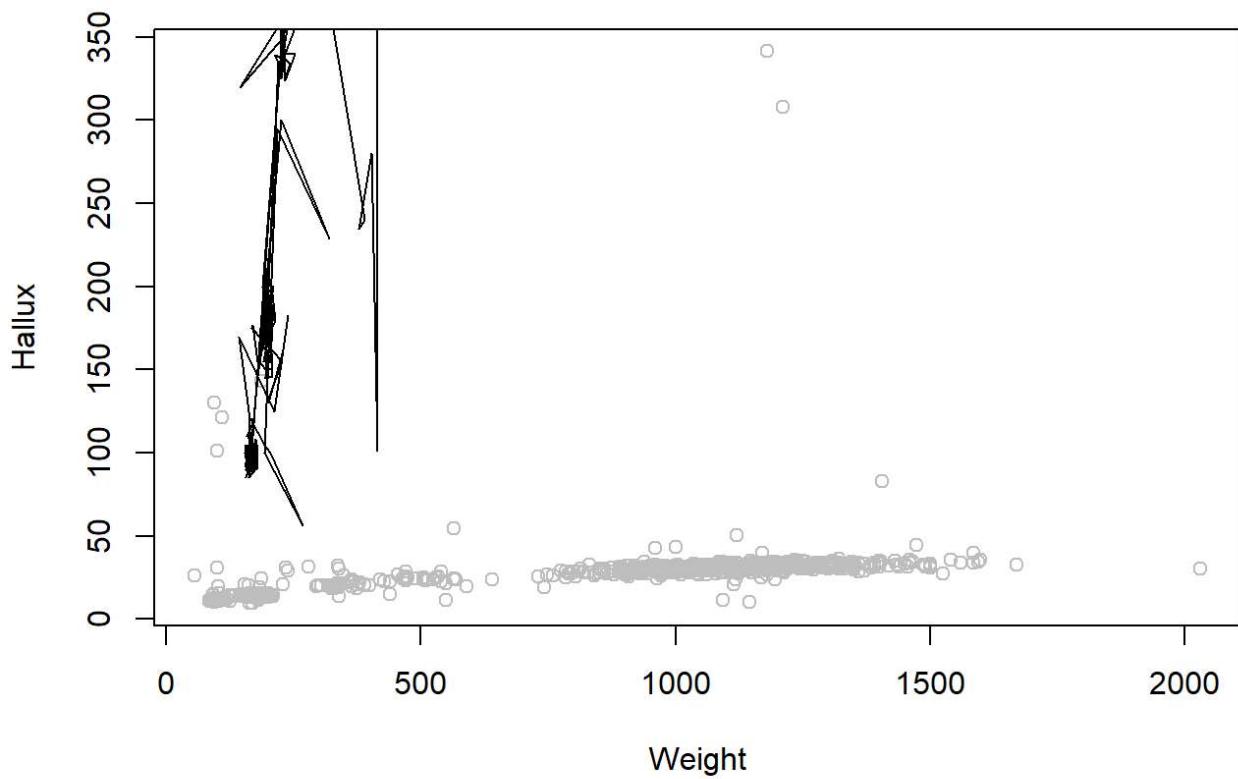
```
### Dibujo de Las trazas que relacionan puntos (res50)
plot(Hawks_data[c(1,4)], col = "grey")
polygon(Hawks_data$res50$order, )
```



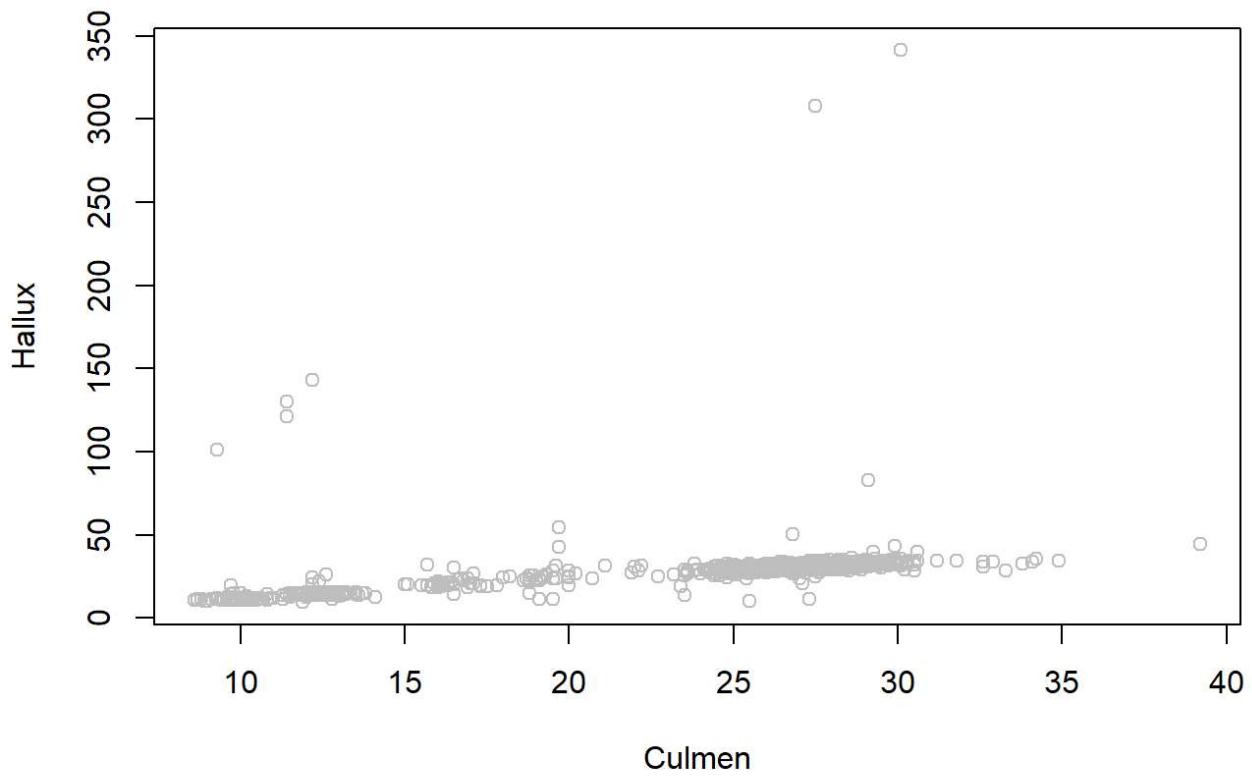
```
### Dibujo de las trazas que relacionan puntos (res50)
plot(Hawks_data[c(2,3)], col = "grey")
polygon(Hawks_data[res50$order,])
```



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

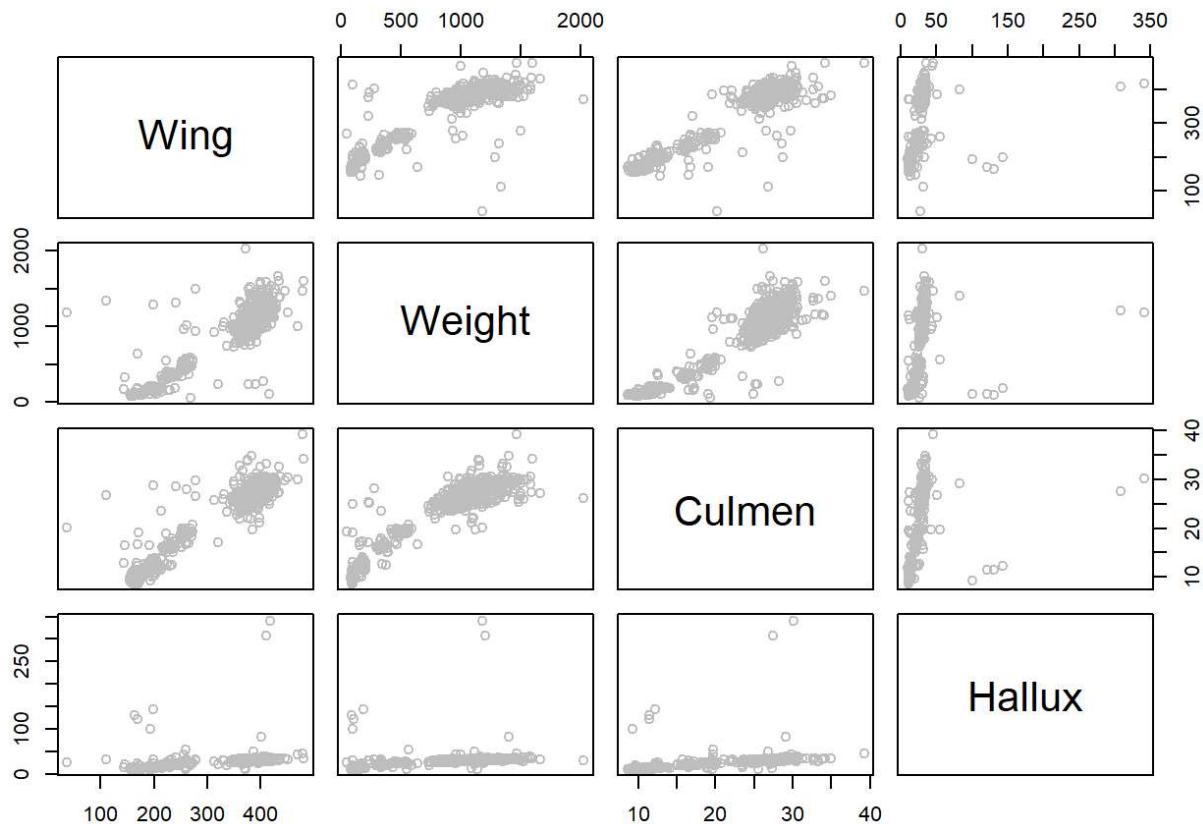


```
### Dibujo de Las trazas que relacionan puntos (res50)
plot(Hawks_data[c(3,4)], col = "grey")
polygon(Hawks_data$res50$order, )
```



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,])
```



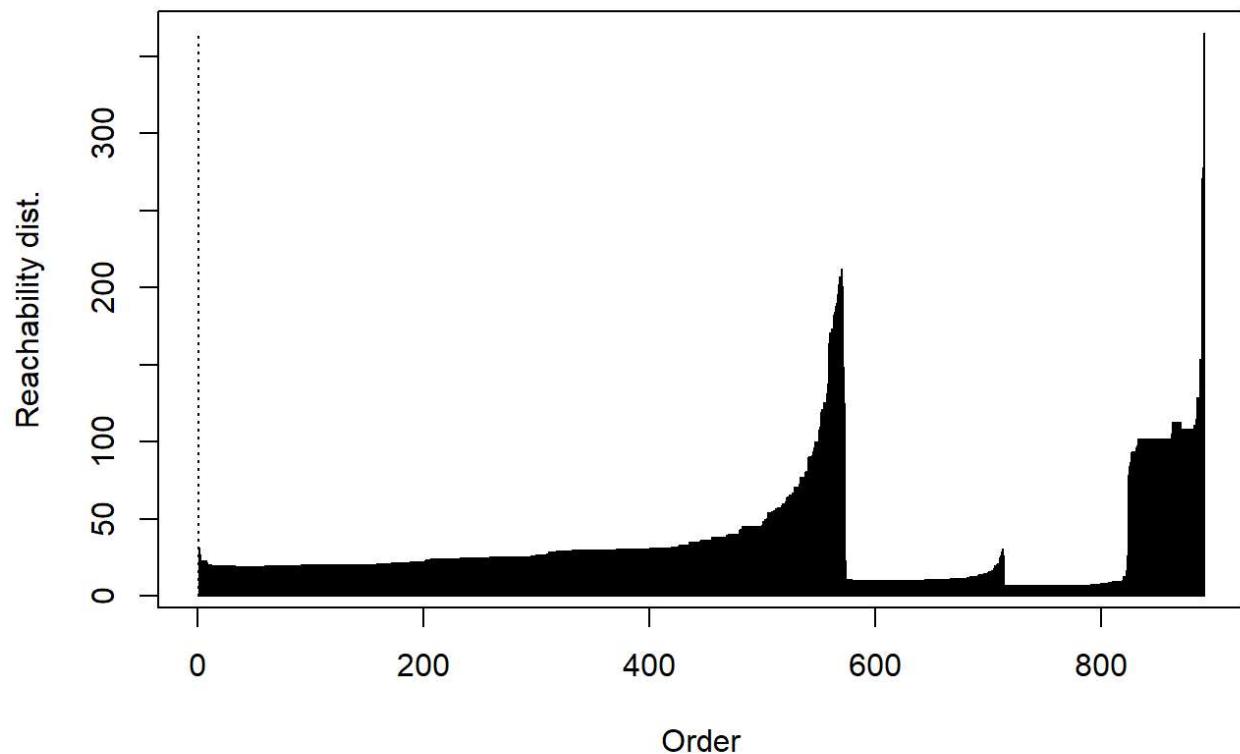
Ahora vamos a probar la alcanzabilidad probando distintos valores de epsilon.

```
### 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
```

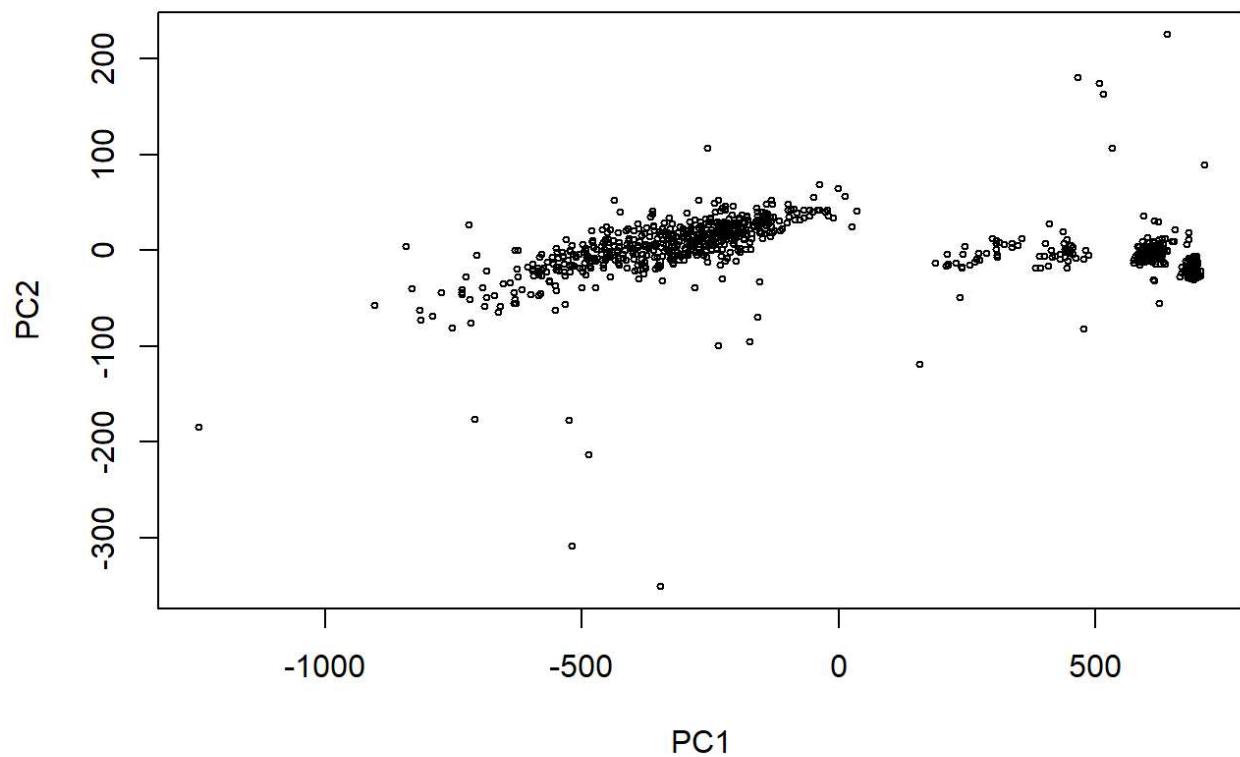
```
plot(res)
```

Reachability Plot



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

Convex Cluster Hulls

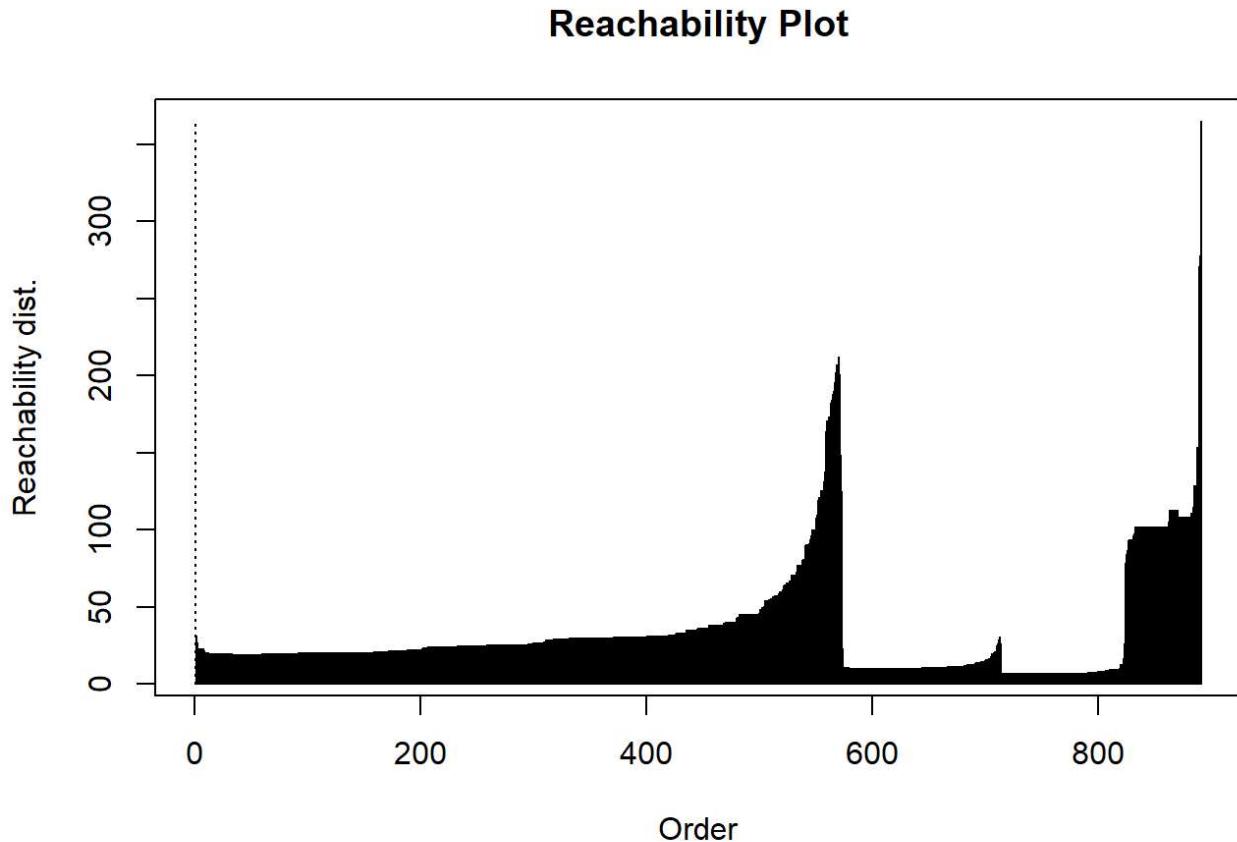


Repetimos el experimento anterior incrementando el parámetro *eps_cl*

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

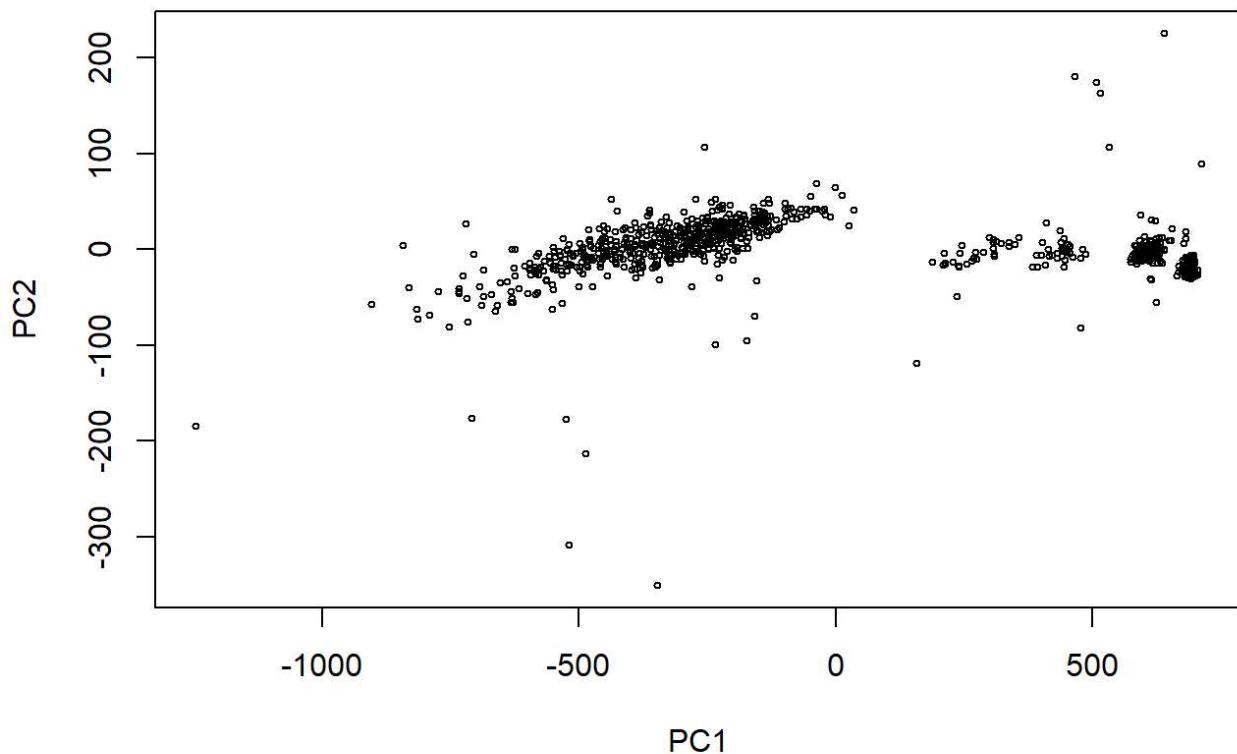
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 50, eps = 686.699861657187, eps_cl = 0.1, xi = NA
## The clustering contains 0 cluster(s) and 891 noise points.
##
##    0
## 891
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, cluster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

```
plot(res)
```



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

Convex Cluster Hulls



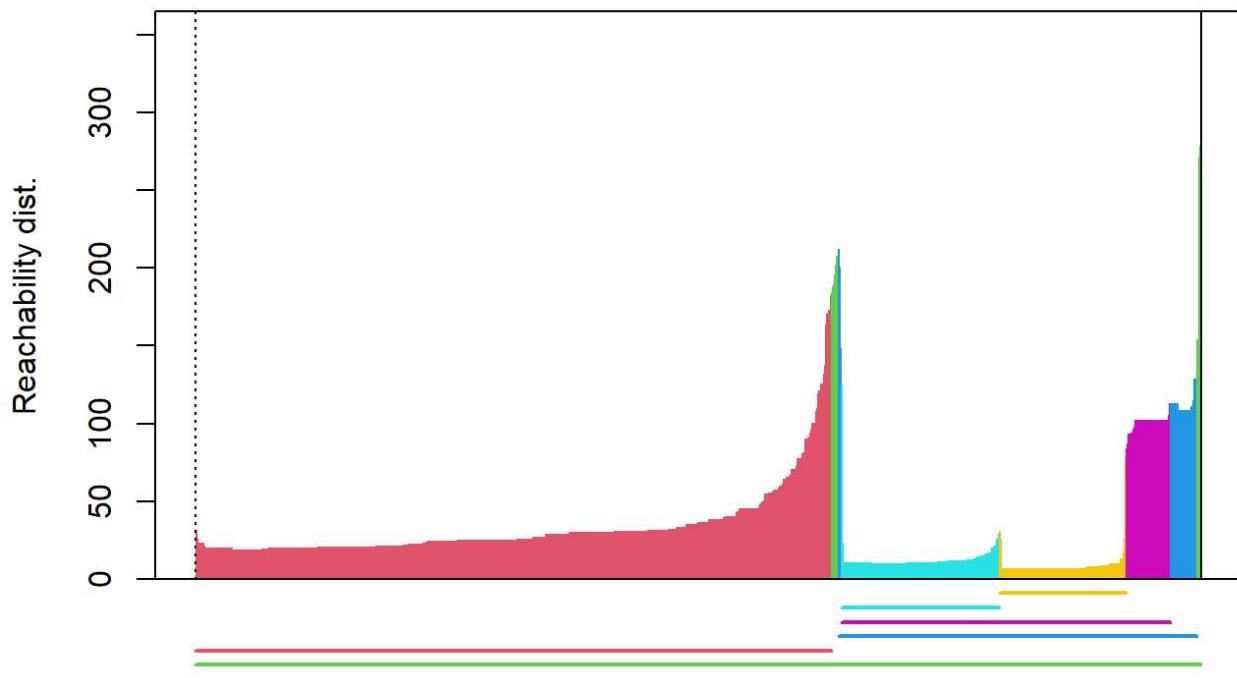
Veamos ahora una variante de la extracción **DBSCN** anterior. En ella el parámetro ξ nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

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

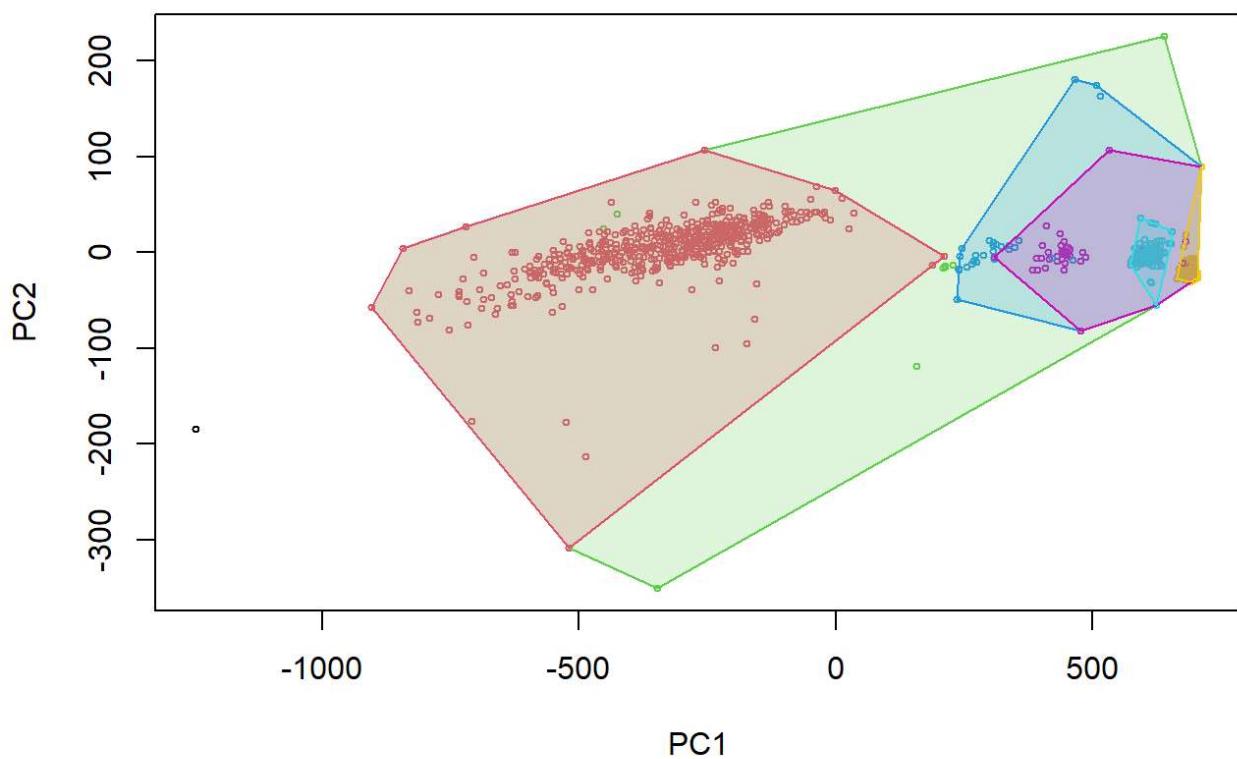
```
plot(res)
```

Reachability Plot



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

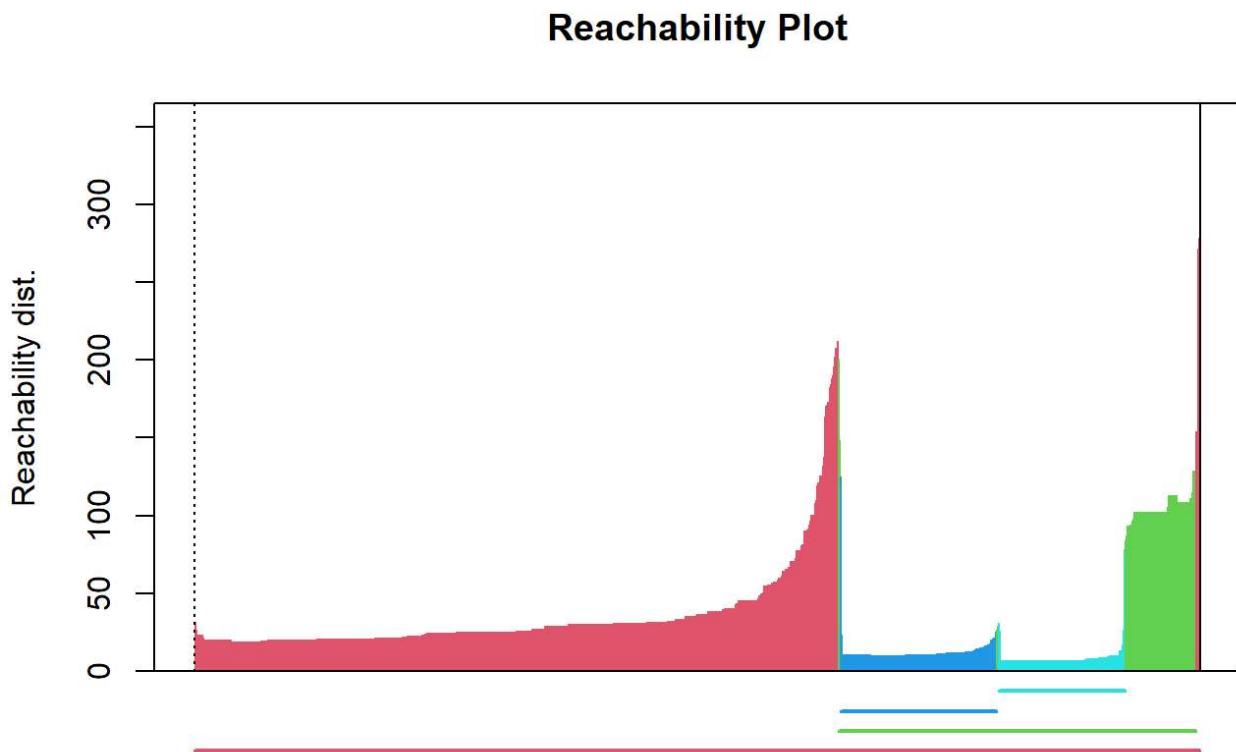
Convex Cluster Hulls



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

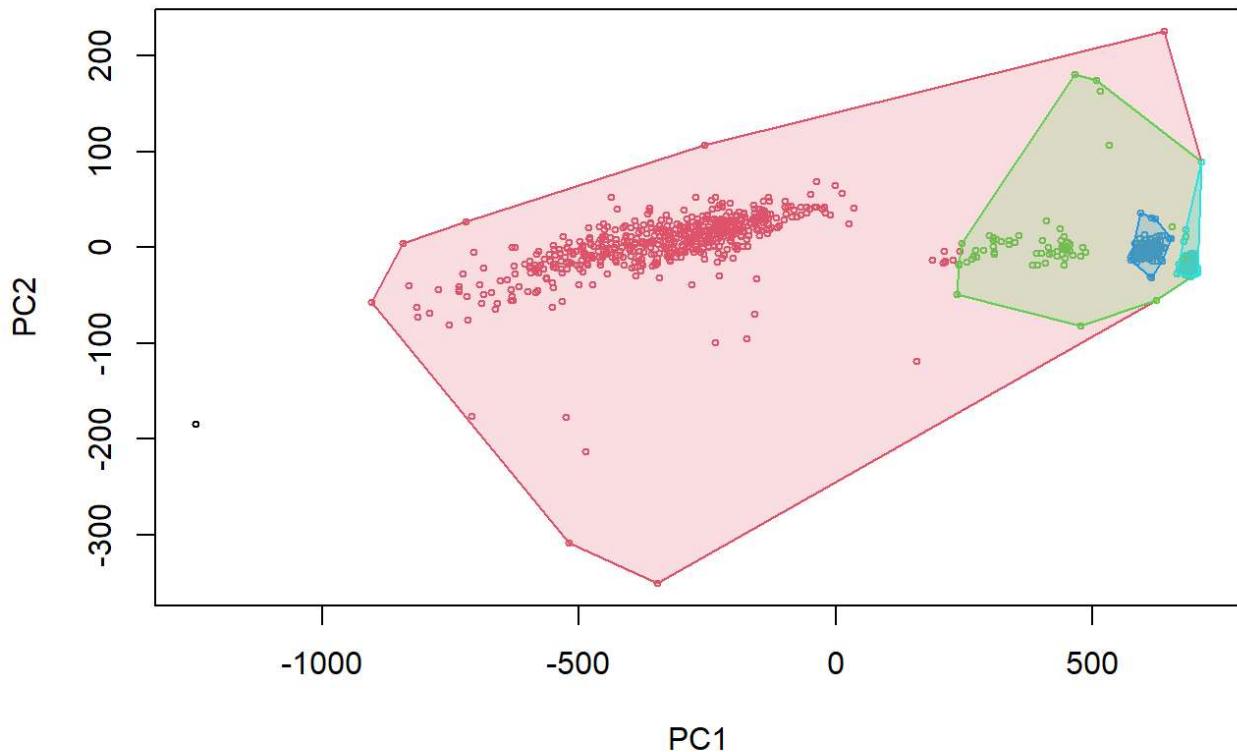
```
## OPTICS ordering/clustering for 891 objects.
## Parameters: minPts = 50, eps = 686.699861657187, eps_cl = NA, xi = 0.1
## The clustering contains 4 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps, eps_cl, xi, clusters_xi, cluster
```

```
plot(res)
```



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

Convex Cluster Hulls



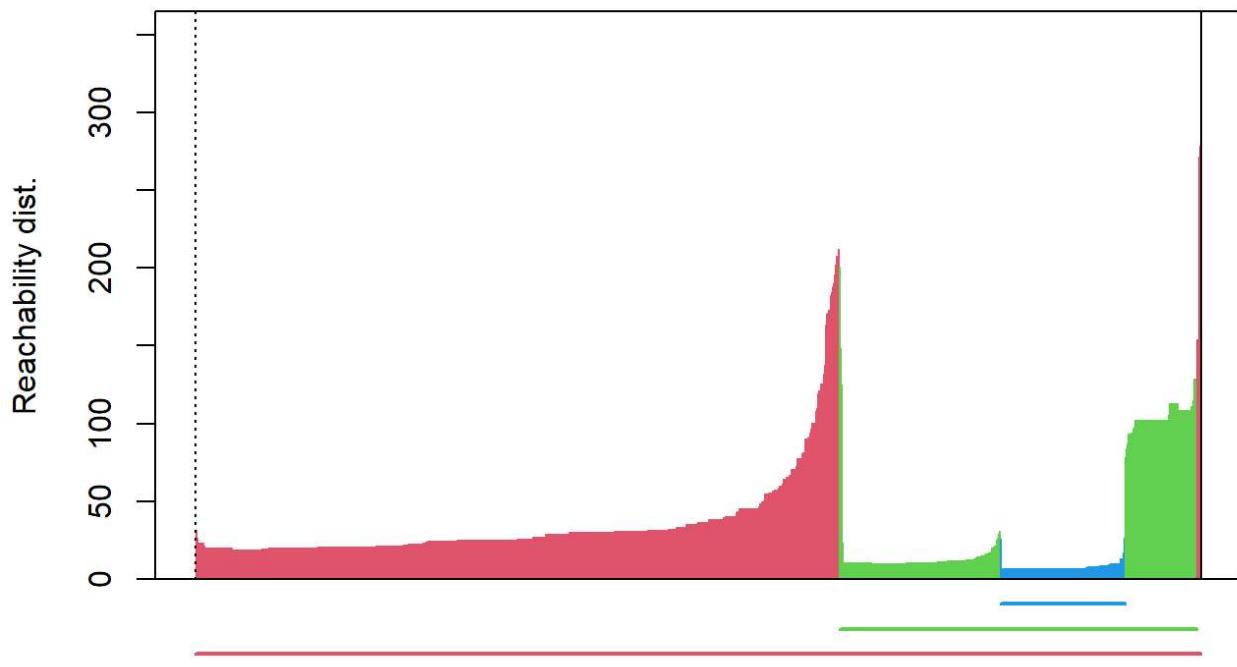
```
### Extracción del clustering jerárquico en función de La variación de La densidad por e  
l método xi
```

```
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, c  
lusters_xi, cluster
```

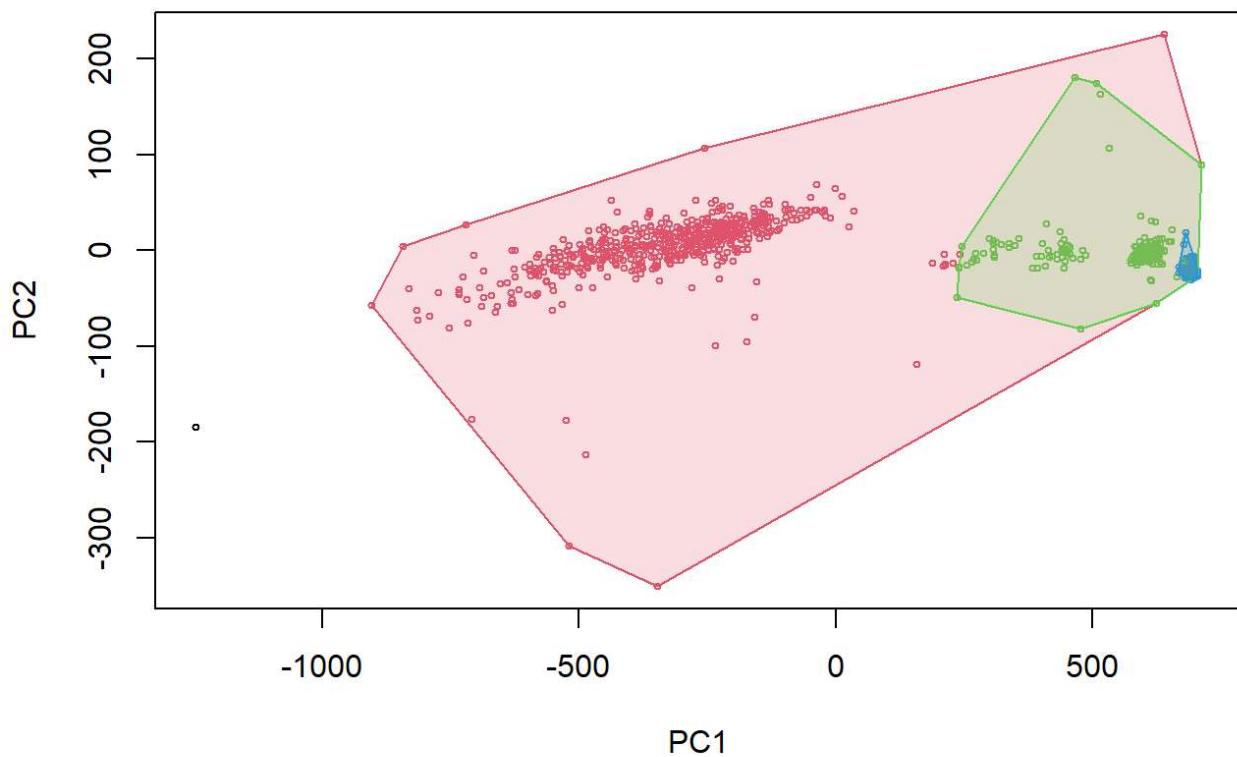
```
plot(res)
```

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 ϵ 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