

# Minería de datos: PEC2 - Métodos no supervisados

Autor: Pablo Suárez Reyero

Octubre 2023

## Contents

<b>Ejemplo guiado 1.1</b>	<b>1</b>
Método de agregación k-means con datos autogenerados . . . . .	1
<b>Ejemplo guiado 1.2</b>	<b>7</b>
Método de agregación k-means con datos reales . . . . .	7
<b>Ejemplo guiado 2</b>	<b>16</b>
Métodos basados en densidad: DBSCAN y OPTICS . . . . .	16
<b>Ejercicios</b>	<b>23</b>
Ejercicio 1 . . . . .	24
Ejercicio 2 . . . . .	53
Ejercicio 3 . . . . .	107

---

## Ejemplo guiado 1.1

### 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)
```

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)

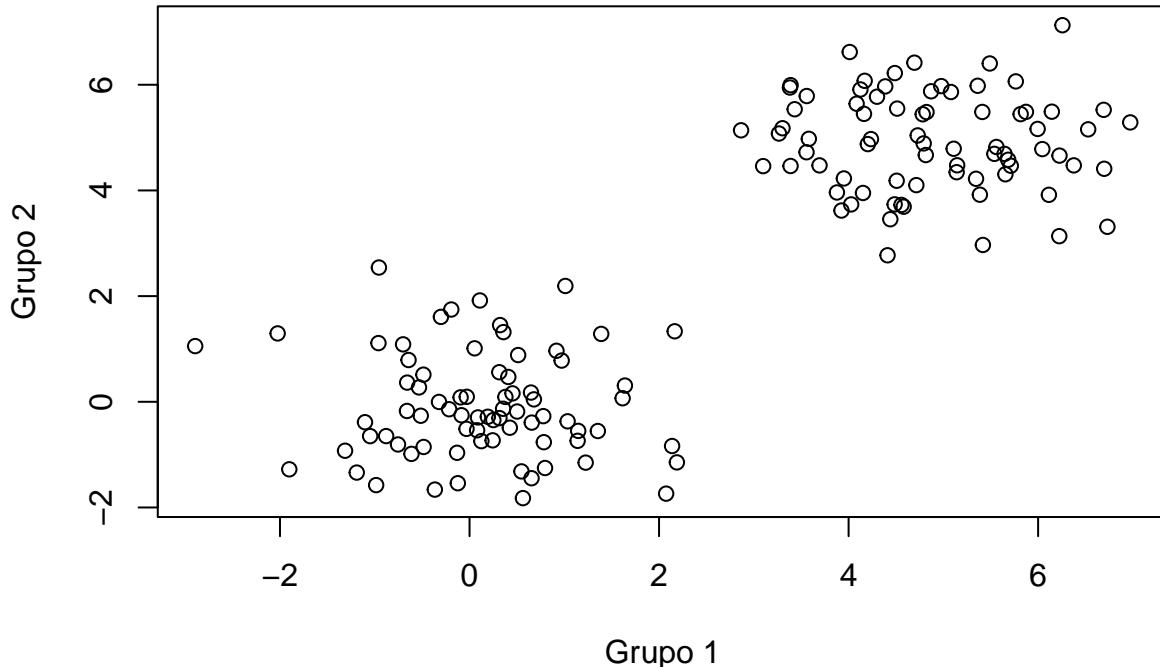
```

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 clústeres

```

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

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

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

```

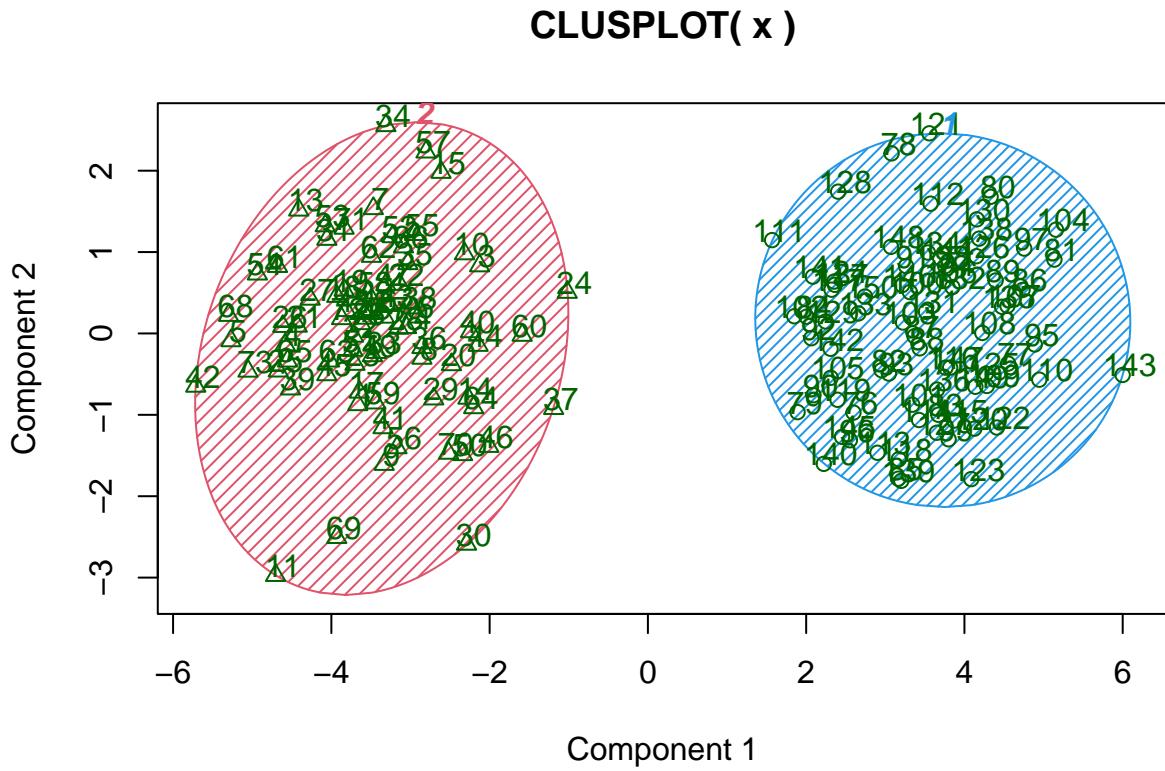
Las variables y\_cluster2, y\_cluster4 e y\_cluster8 contienen para cada muestra el identificador del clúster a

las que han sido asignadas. Por ejemplo, en el caso de los  $k=2$  las muestras se han asignado al clúster 1 ó al 2

y\_cluster2

Para visualizar los clústers podemos usar la función `clusplot`. Vemos la agrupación con 2 clústers y observemos como prácticamente no hay valores extremos y realmente los dos clústers generados son homogéneos.

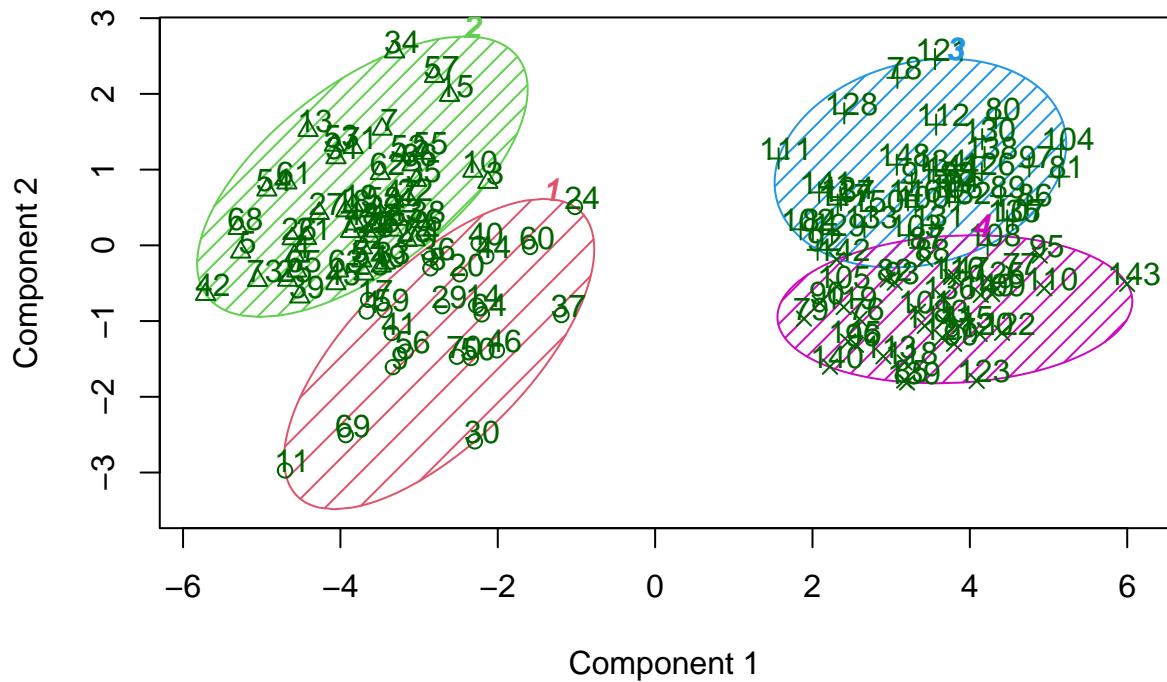
```
clusplot(x, fit2$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



con 4 observamos como el clúster de la izquierda lo ha dividido en 3.

classpiece(n, titlecluster, centerTitle, shadeTitle, labels)

## CLUSPLOT( x )



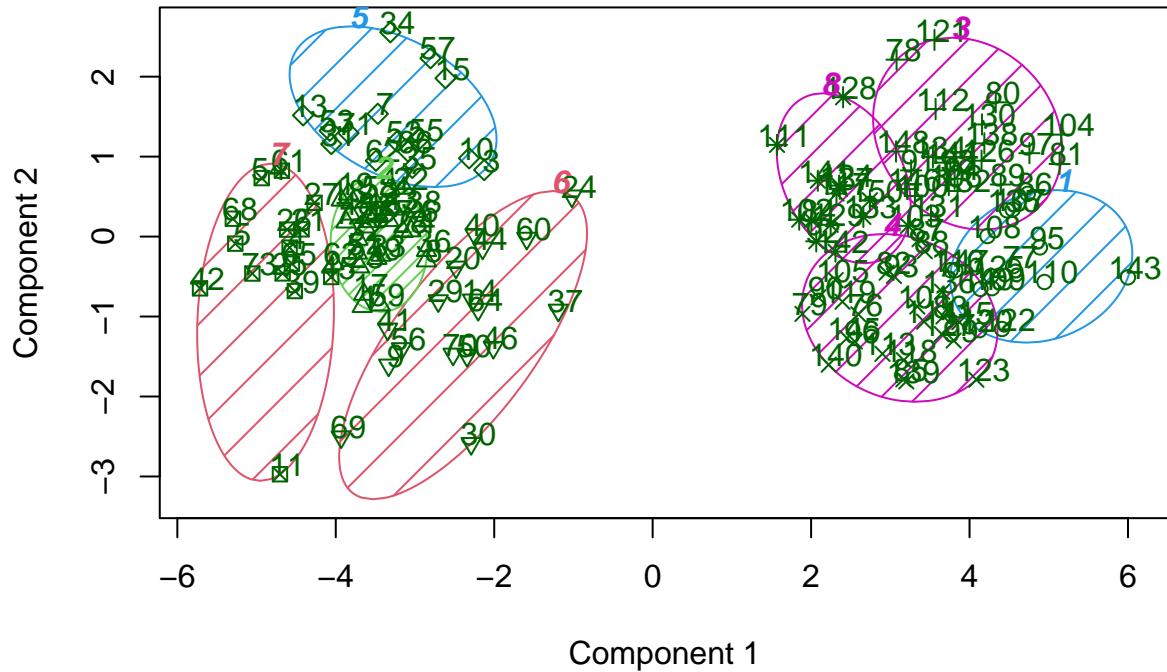
Component 1

These two components explain 100 % of the point variability.

y con 8. El algoritmo obedece y nos genera 8 clústers aunque como se aprecia visualmente no tenga demasiada consistencia.

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

## CLUSPLOT( x )

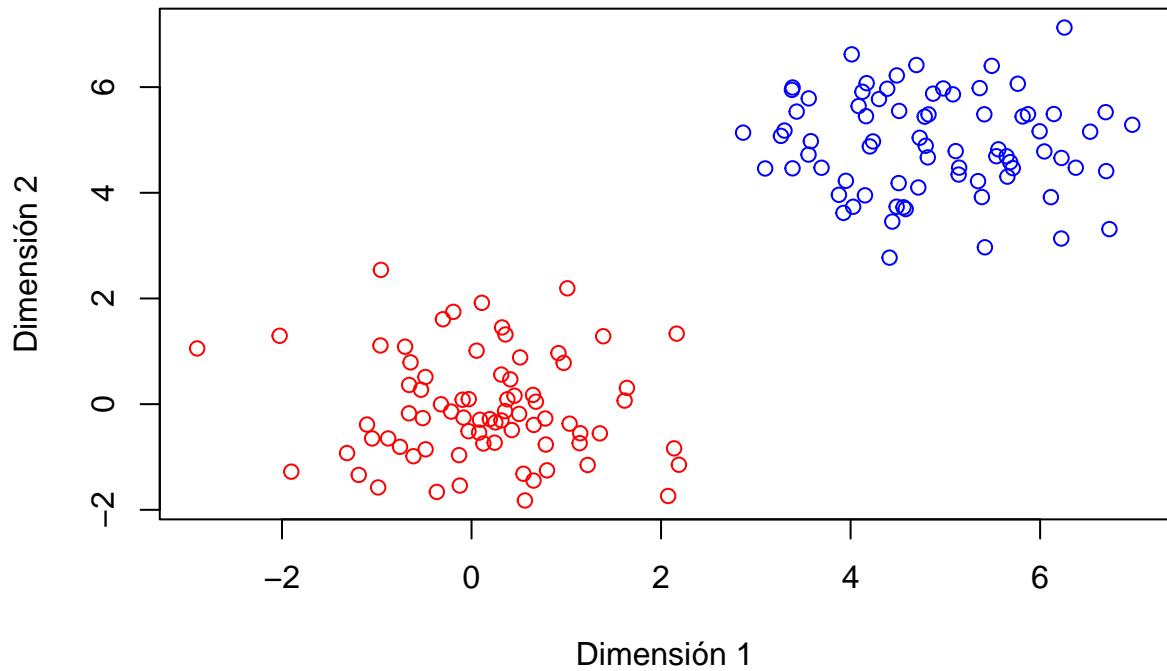


Component 1

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 clústers. El uso de colores facilita la identificación visual de clústers.

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

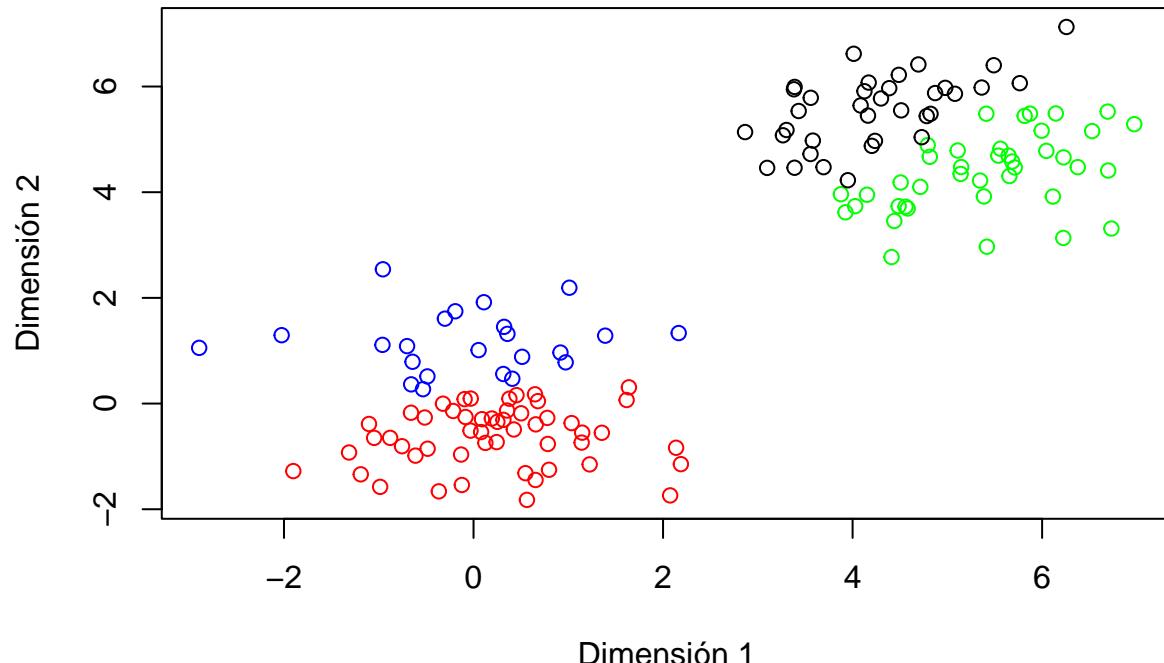


para 4

```

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

```

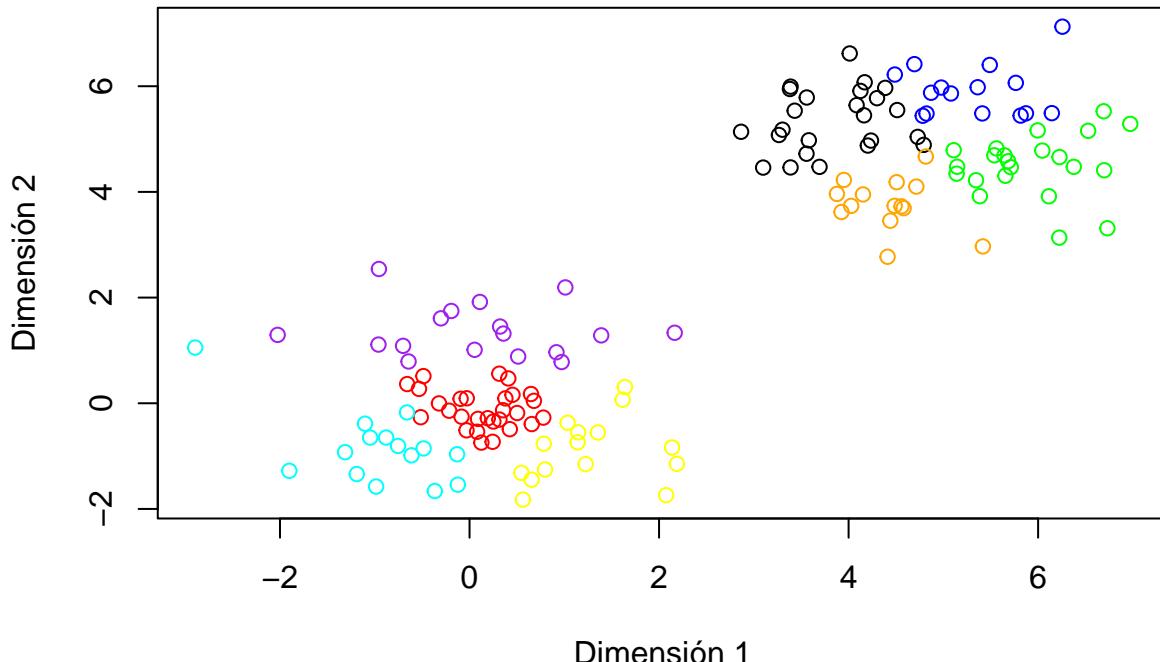


y para 8

```

plot(x[y_cluster8==1], col='blue', xlim=c(min(x[,1]), max(x[,1])), ylim=c(min(x[,2]), max(x[,2])), xlab=
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] 0.7475492
mean(sk4[,3])
## [1] 0.3608202
mean(sk8[,3])
## [1] 0.336868
```

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.

Una buena práctica para entender mejor el juego de datos, consiste en poner nombre a cada uno de los clústers identificados. Lo veremos más claramente en el siguiente ejemplo que parte de datos reales.

## Ejemplo guiado 1.2

### 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> 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
##   <fct>   <fct>           <dbl>        <dbl>          <int>       <int>
## 1 Adelie   Torgersen     39.1         18.7          181        3750
## 2 Adelie   Torgersen     39.5         17.4          186        3800
## 3 Adelie   Torgersen     40.3         18            195        3250
## 4 Adelie   Torgersen     NA            NA             NA          NA
## 5 Adelie   Torgersen     36.7         19.3          193        3450
## 6 Adelie   Torgersen     39.3         20.6          190        3650
## 7 Adelie   Torgersen     38.9         17.8          181        3625
## 8 Adelie   Torgersen     39.2         19.6          195        4675
## 9 Adelie   Torgersen     34.1         18.1          193        3475
## 10 Adelie  Torgersen    42            20.2          190        4250
## # i 334 more rows
## # i 2 more variables: sex <fct>, year <int>
summary(penguins)

##      species      island   bill_length_mm   bill_depth_mm
##      Adelie:152  Biscoe :168   Min.   :32.10   Min.   :13.10
##      Chinstrap: 68  Dream  :124   1st Qu.:39.23   1st Qu.:15.60
##      Gentoo: 124 Torgersen:52   Median :44.45   Median :17.30
##                               Mean   :43.92   Mean   :17.15
##                               3rd Qu.:48.50   3rd Qu.:18.70
##                               Max.  :59.60   Max.  :21.50
##                               NA's   :2       NA's   :2
##      flipper_length_mm   body_mass_g
##      Min.   :172.0   Min.   :2700
##      1st Qu.:190.0  1st Qu.:3550
##      Median :197.0  Median :4050
##      Mean   :200.9  Mean   :4202
##      3rd Qu.:213.0  3rd Qu.:4750
##      Max.  :231.0   Max.  :6300
##      NA's   :2       NA'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.

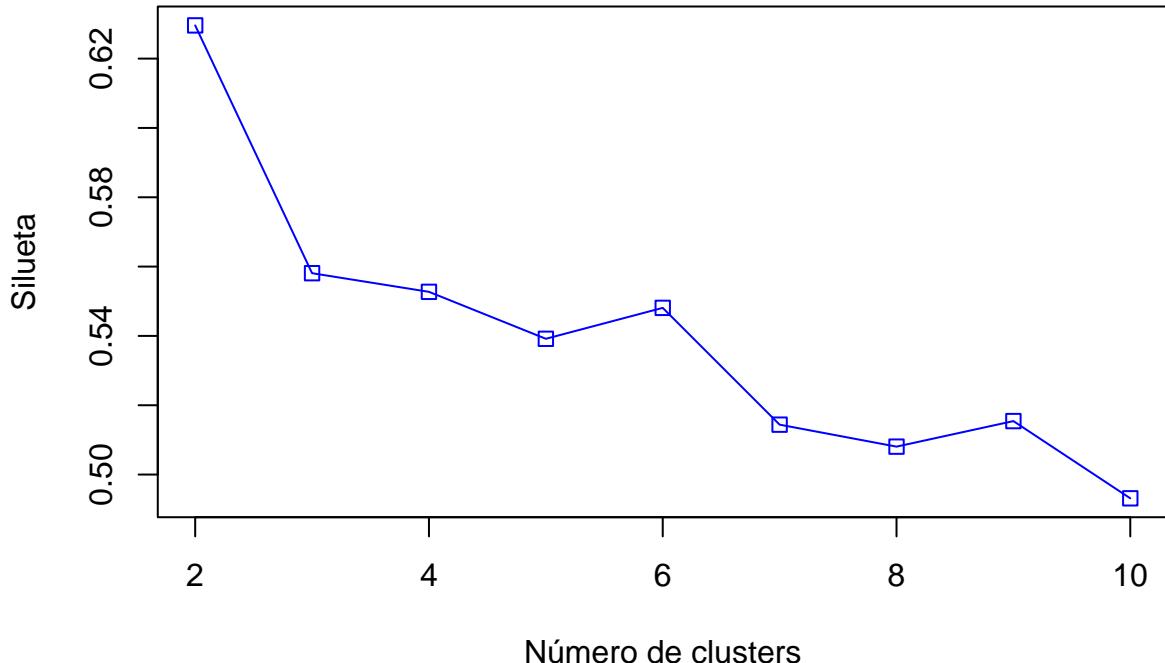
```
x <- na.omit(penguins[,3:6])
```

Planteamos ahora un ejemplo más realista en el que inicialmente no conocemos el número óptimo de clústers. Empecemos 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")
```



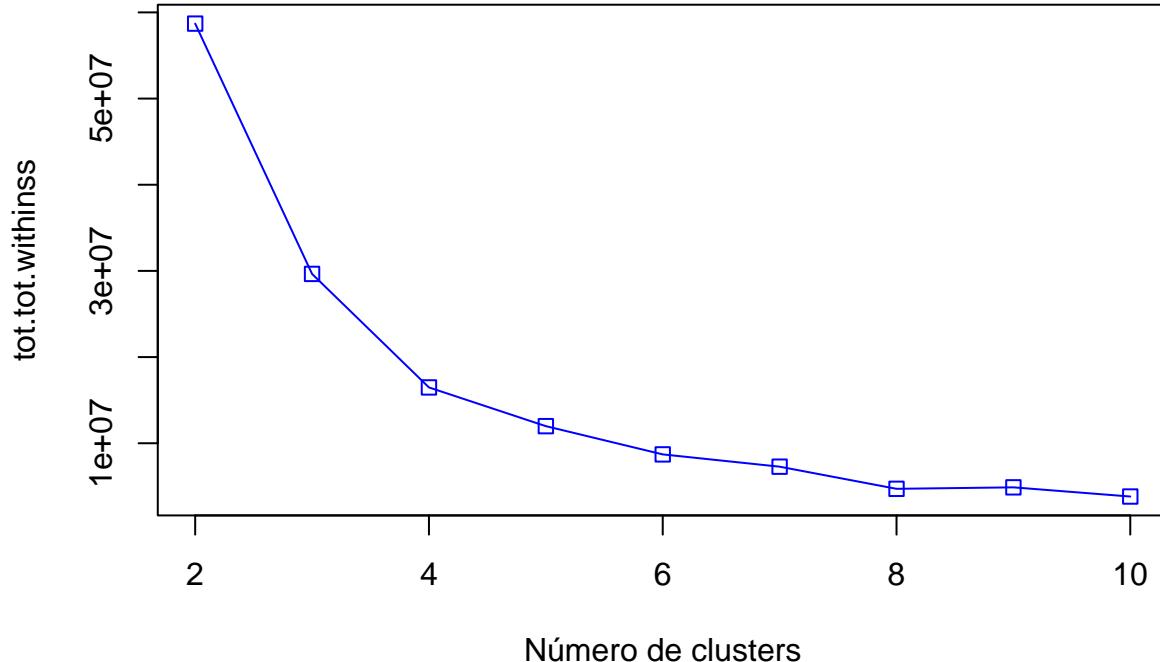
A pesar de que uno esperaría obtener un valor óptimo para  $k = 3$ , parece que del gráfico se desprende que es mejor  $k = 4$  o incluso  $k = 5$ .

Otra 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, krangle = 1:10, criterion = "ch")
fit_asw <- kmeansruns(x, krangle = 1:10, criterion = "asw")

```

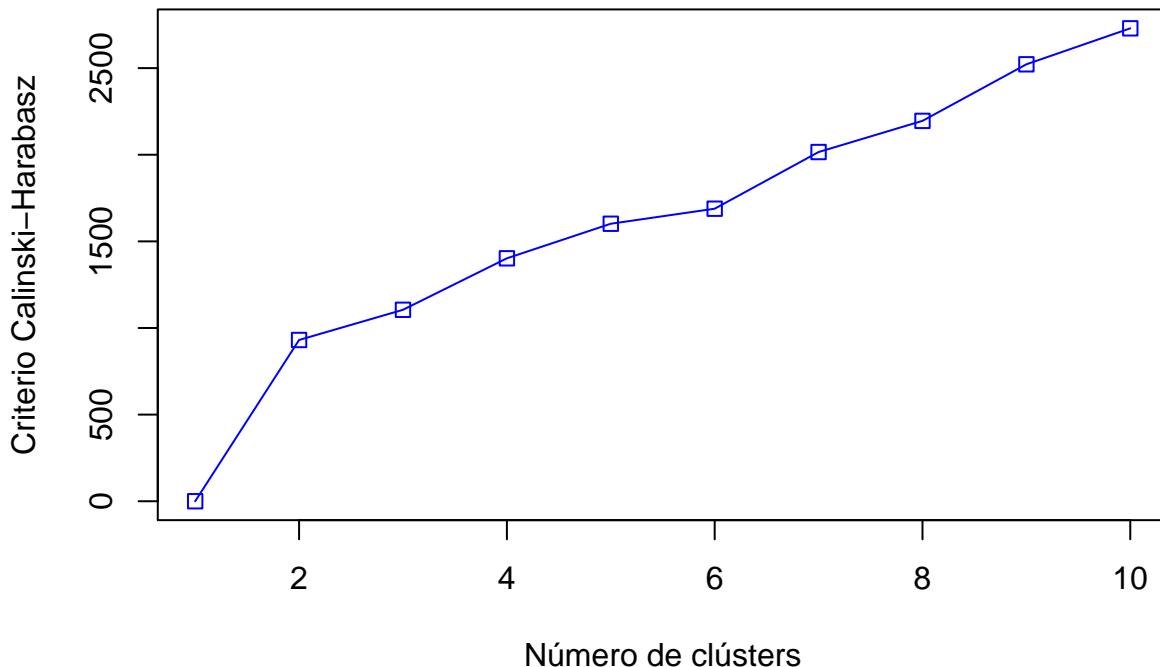
Podemos comprobar el valor con el que se ha obtenido el mejor resultado y también mostrar el resultado obtenido para todos los valores de k usando ambos criterios

```

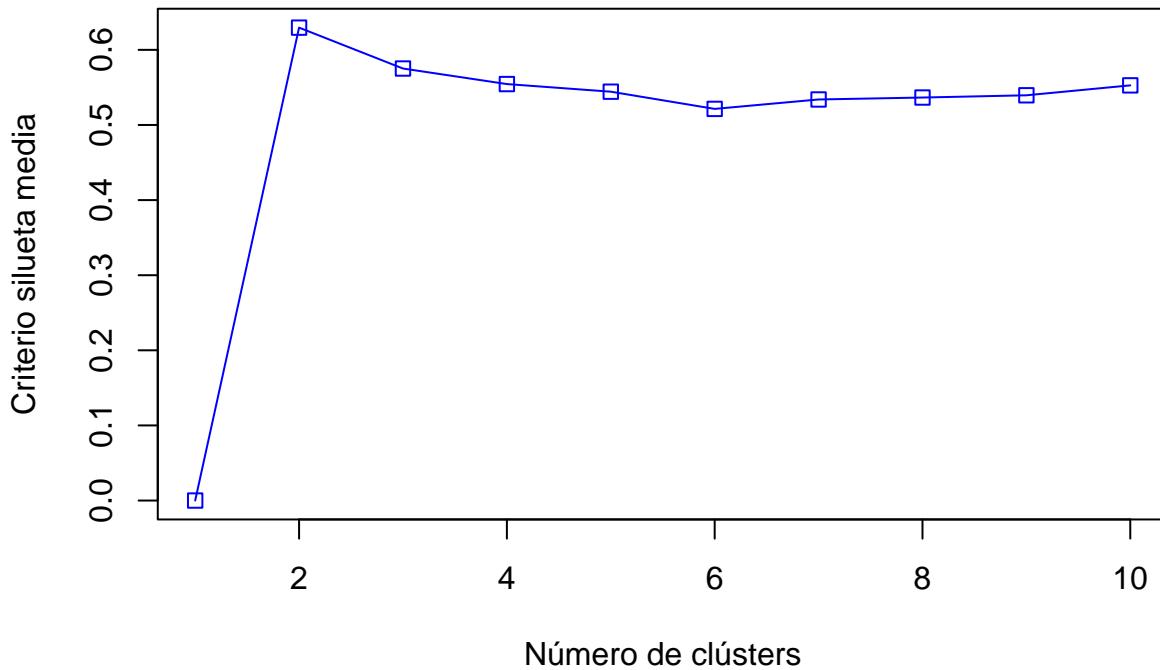
fit_ch$bestk
## [1] 10
fit_asw$bestk
## [1] 2

```

```
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 compararemos visualmente los

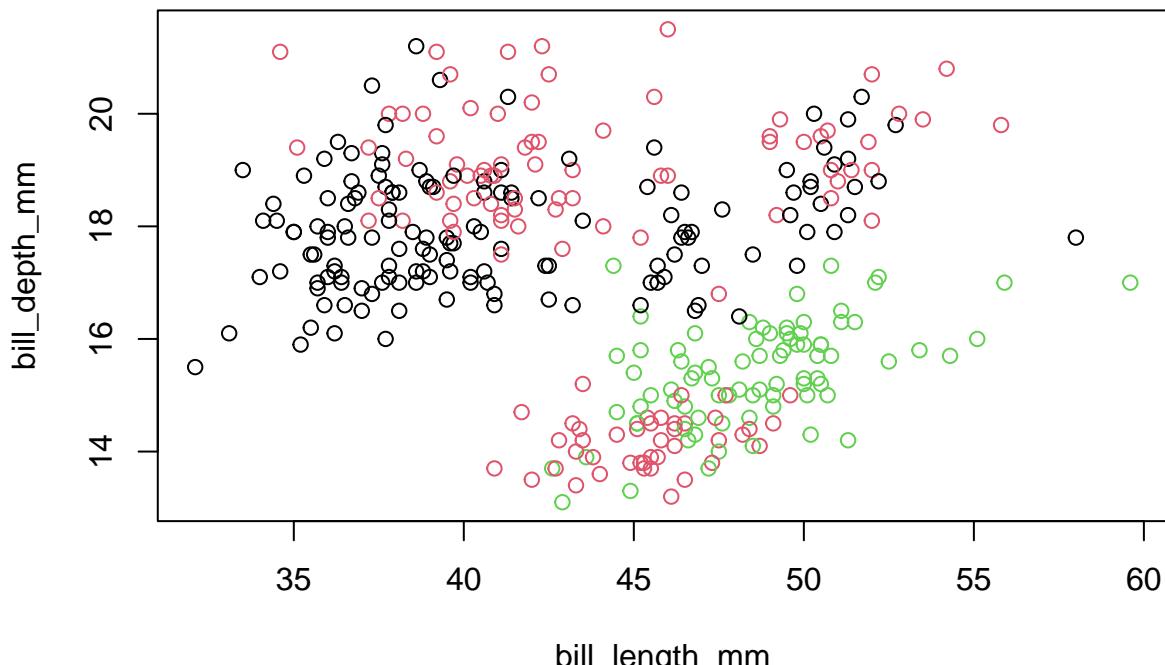
campos dos a dos, con el valor real que sabemos está almacenado en el campo “species” del dataset original. (Aclaramos que obviamente no acostumbra a pasar que conozcamos de forma previa el número de clústers óptimo. Este ejemplo lo planteamos con finalidades didácticas y con voluntad de experimentar)

```
penguins3clusters <- kmeans(x, 3)
print(x[c(1,2)])
```

```
## # A tibble: 342 x 2
##   bill_length_mm bill_depth_mm
##       <dbl>        <dbl>
## 1     39.1         18.7
## 2     39.5         17.4
## 3     40.3          18
## 4     36.7         19.3
## 5     39.3         20.6
## 6     38.9         17.8
## 7     39.2         19.6
## 8     34.1         18.1
## 9      42          20.2
## 10    37.8         17.1
## # i 332 more rows
```

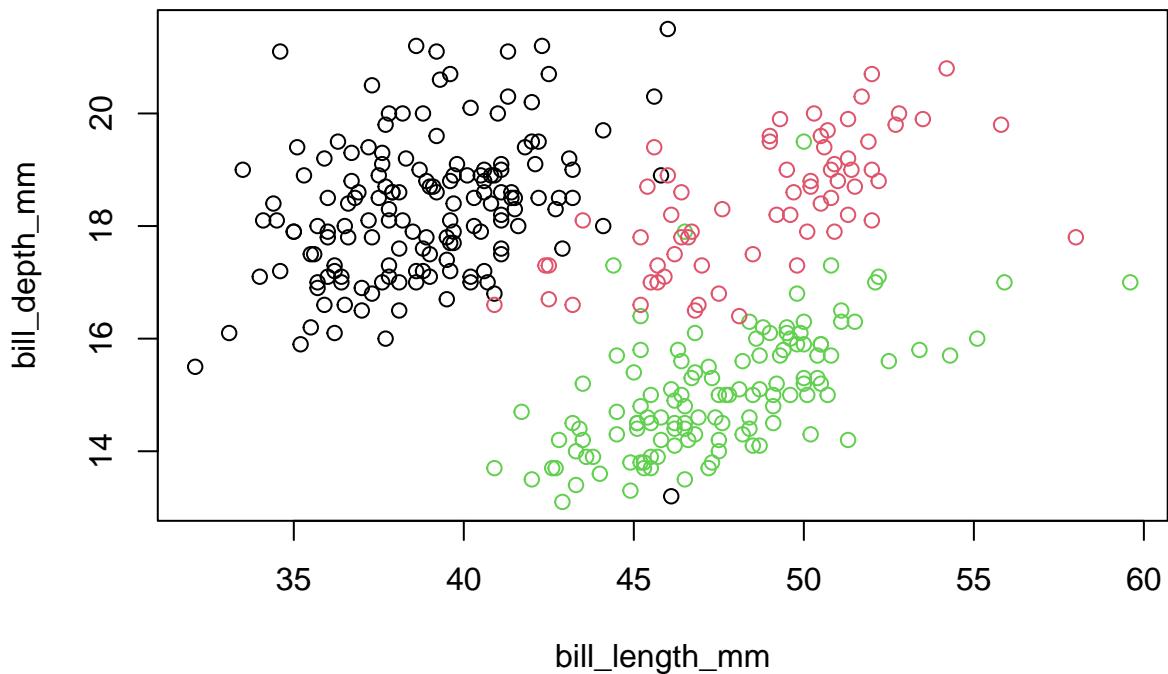
```
# bill_llength y bill_depth
plot(x[c(1,2)], col=penguins3clusters$cluster, main="Clasificación k-means")
```

## Clasificación k-means



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

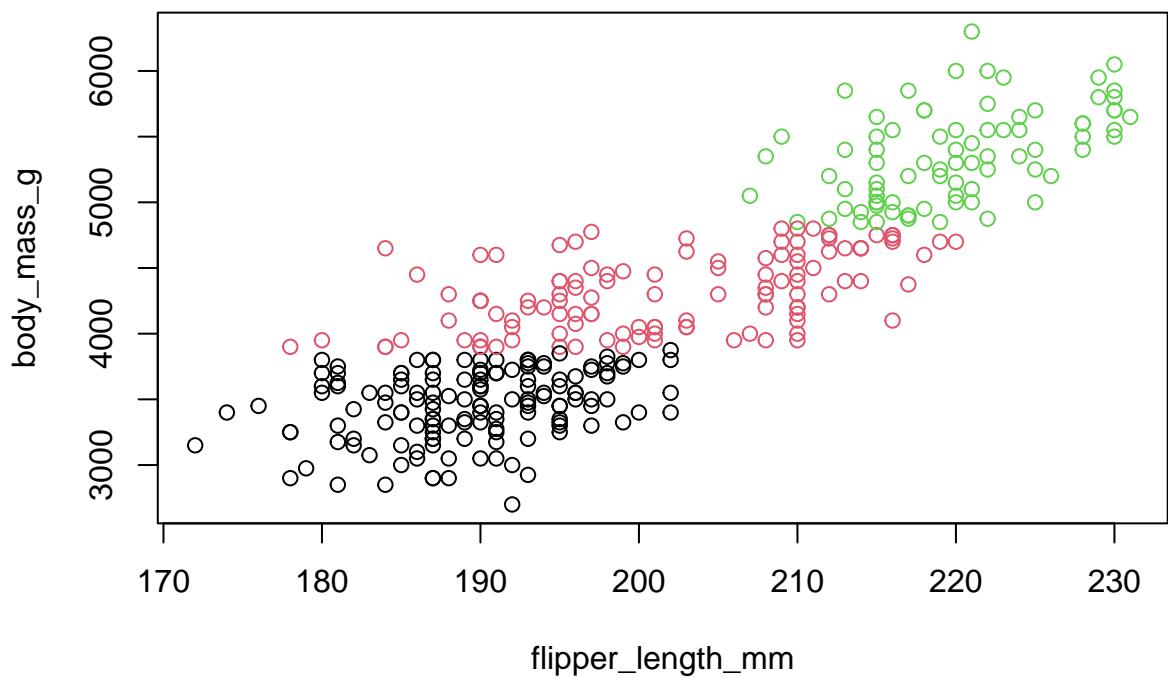
## Clasificación real



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.

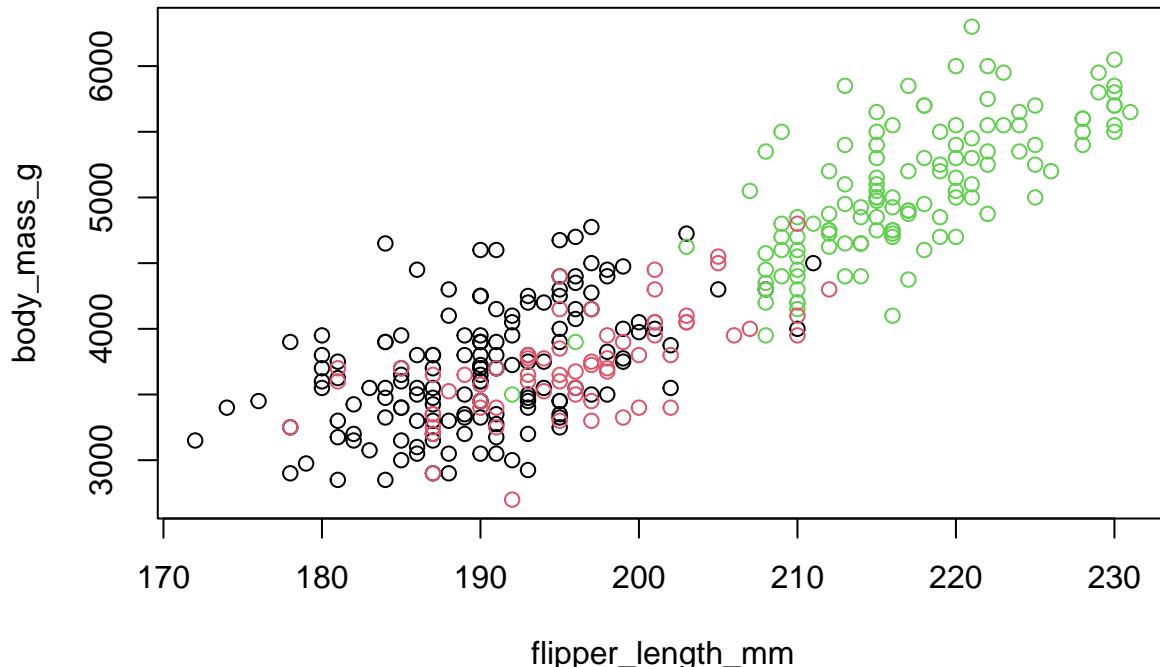
```
# flipper_length y body_mass  
plot(x[c(3,4)], col=penguins3clusters$cluster, main="Clasificación k-means")
```

## Clasificación k-means



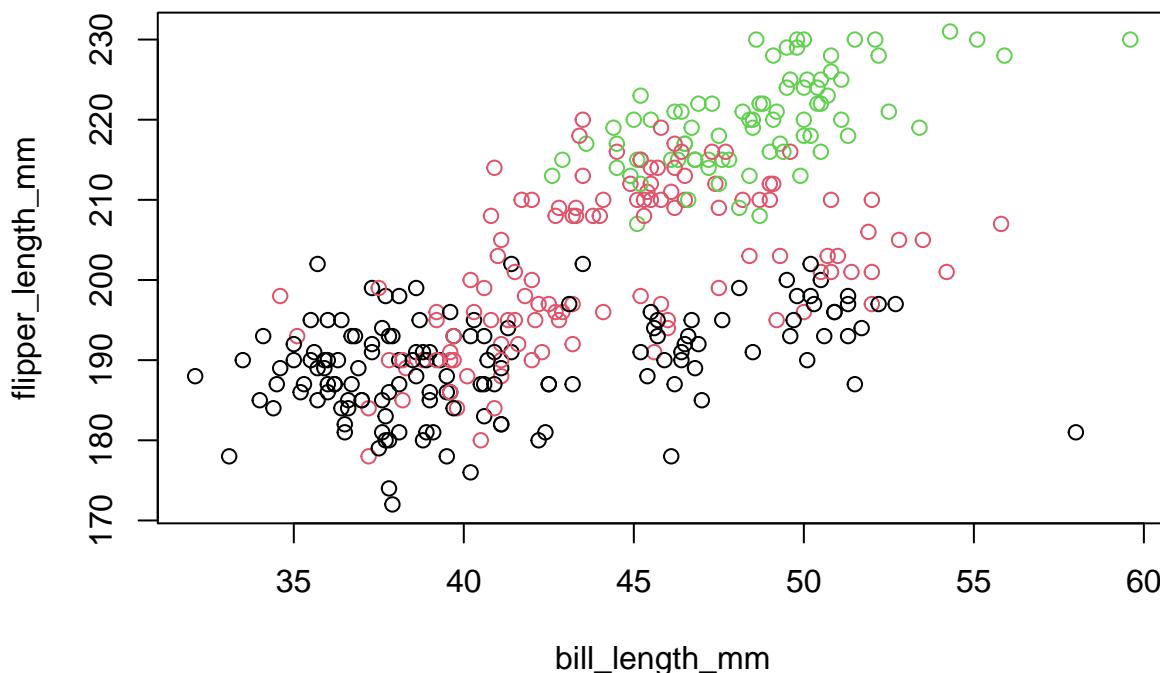
```
plot(x[c(3,4)], col=as.factor(penguins$species), main="Clasificación real")
```

### Clasificación real



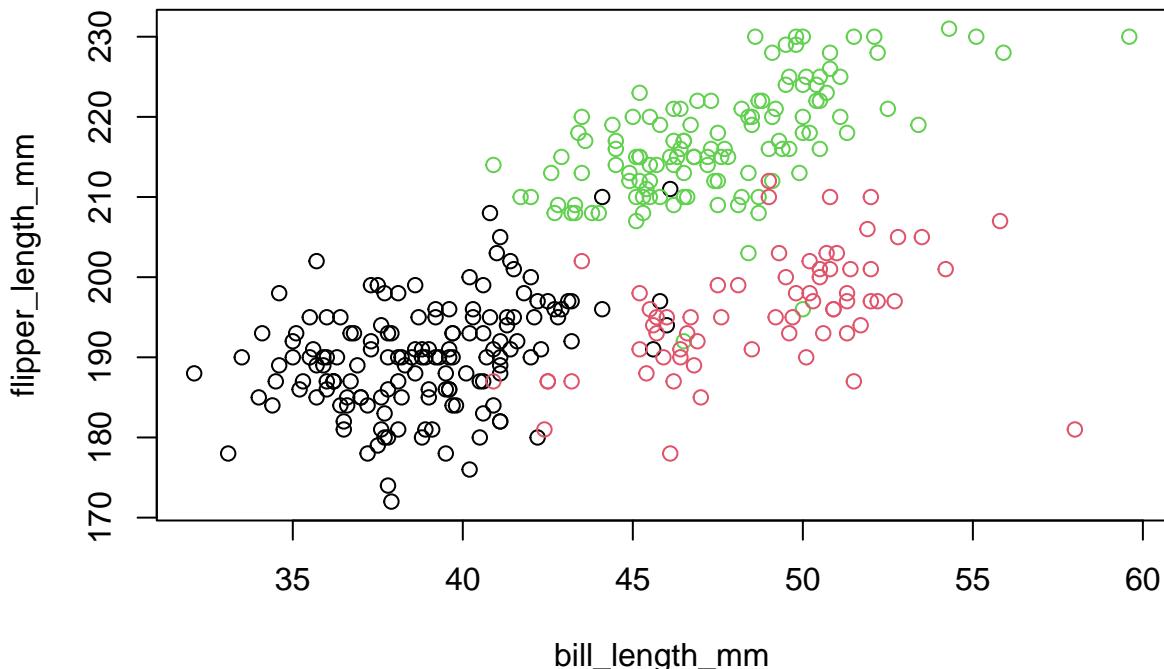
```
# bill_length y flipper_length  
plot(x[c(1,3)], col=penguins3clusters$cluster, main="Clasificación k-means")
```

### Clasificación k-means



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

## 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.

Todo esto nos indica que el número de grupos o clústers en un juego de datos no es un aspecto que podamos asegurar que siempre vamos a encontrar de forma precisa y objetiva, bien al contrario es un ámbito que requiere de análisis en sí mismo.

Os compartimos en el siguiente enlace un material didáctico complementario que os puede ayudar a profundizar en el tema de la selección del número de clústers más adecuado para un juego de datos:  
[datascience.recursos.uoc.edu](http://datascience.recursos.uoc.edu)

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.

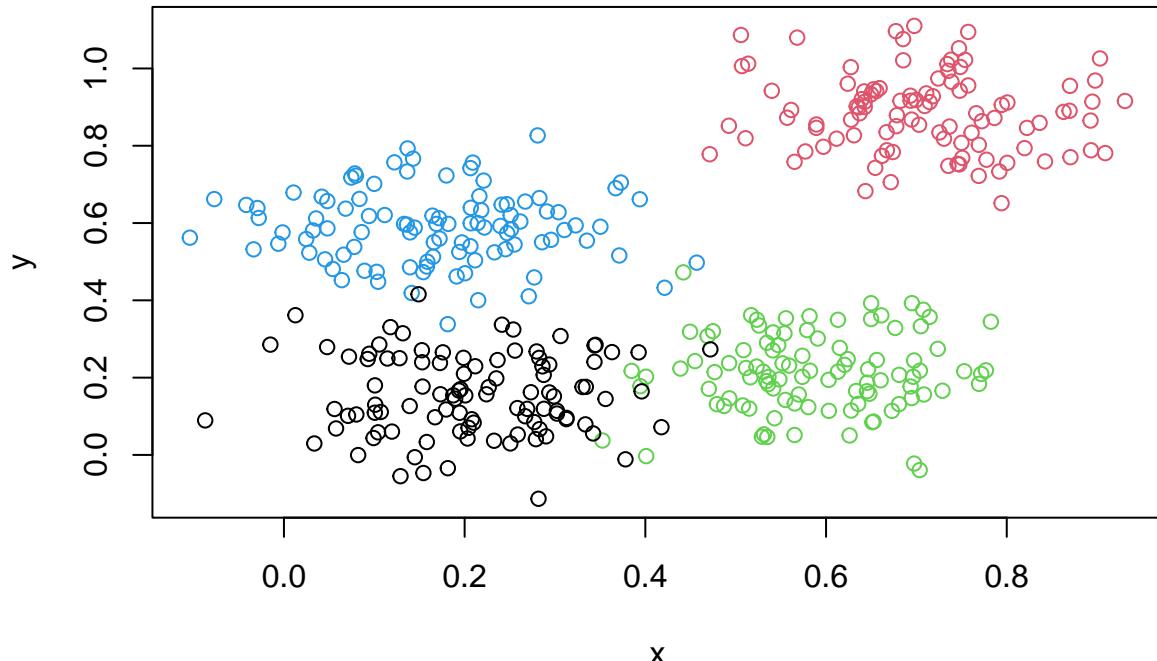
## Ejemplo guiado 2

### 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
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
## [19] 77 169 105 293 229 145 181 385 393 377 317 381 185 117 101 9 73 237
## [37] 397 369 365 273 305 245 249 309 157 345 213 205 97 49 33 41 193 149
## [55] 17 83 389 25 121 329 5 161 341 217 189 141 85 53 225 313 289 261
## [73] 221 173 69 61 297 125 81 133 129 197 109 137 59 93 165 89 21 13
## [91] 277 191 203 379 399 375 351 311 235 231 227 71 11 299 271 291 147 55
## [109] 23 323 219 275 47 263 3 367 331 175 87 339 319 251 247 171 111 223
## [127] 51 63 343 303 207 151 391 359 287 283 215 143 131 115 99 31 183 43
## [145] 243 199 79 27 295 67 347 255 239 195 187 139 107 39 119 179 395 371
## [163] 201 123 159 91 211 355 103 327 95 7 167 35 267 155 387 383 335 315
## [181] 259 135 15 113 279 373 4 353 265 127 45 37 19 276 224 361 260 288
## [199] 336 368 348 292 268 252 120 108 96 88 32 16 340 156 388 372 356 332
## [217] 304 220 188 168 136 124 56 236 28 244 392 184 76 380 232 100 116 112
## [235] 256 72 8 280 64 52 208 172 152 148 360 352 192 160 144 284 216 48
## [253] 84 92 36 20 212 272 264 200 128 80 180 364 196 12 132 40 324 308
## [271] 176 164 68 316 312 384 300 344 328 248 204 140 296 24 320 228 60 44
## [289] 233 65 400 376 240 163 104 396 307 75 14 325 269 262 234 382 294 206
## [307] 198 374 310 362 318 386 358 330 278 210 298 282 122 98 34 26 174 142
## [325] 46 6 62 118 190 202 114 322 286 38 242 394 342 266 162 130 30 182
## [343] 2 74 314 290 246 194 170 126 158 378 350 254 226 214 70 18 10 366
## [361] 354 186 150 86 306 102 338 346 134 250 138 94 78 390 274 58 42 258
## [379] 66 90 146 370 222 218 326 82 110 270 334 178 166 398 22 50 238 106
## [397] 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.

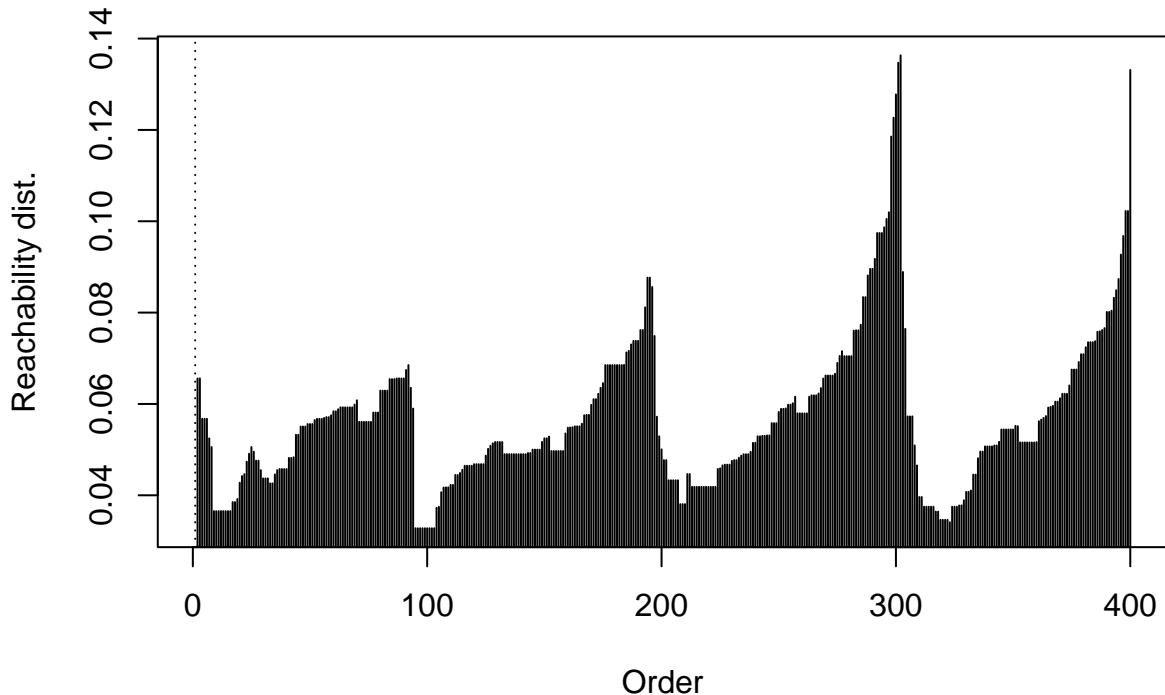
Los valles representan clusters (cuanto más profundo es el valle, más denso es el cluster), mientras que las cimas indican los puntos que están entre las agrupaciones (estos puntos son candidatos a ser considerados *outliers*)

```

### Gráfica de alcanzabilidad
plot(res)

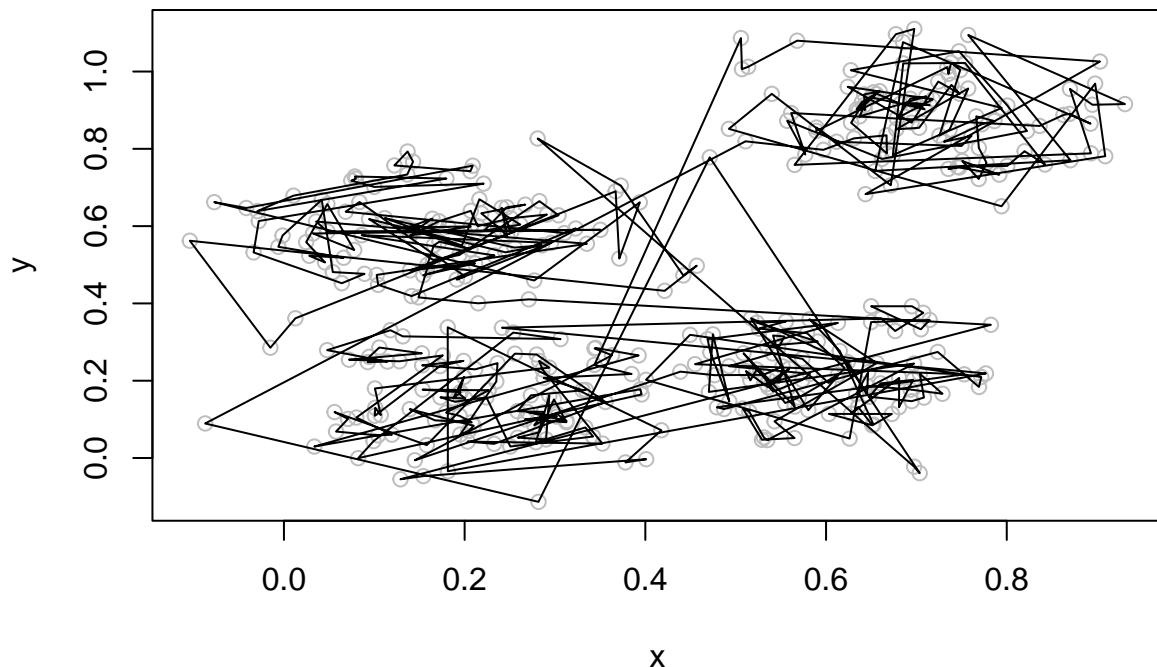
```

## 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,])
```



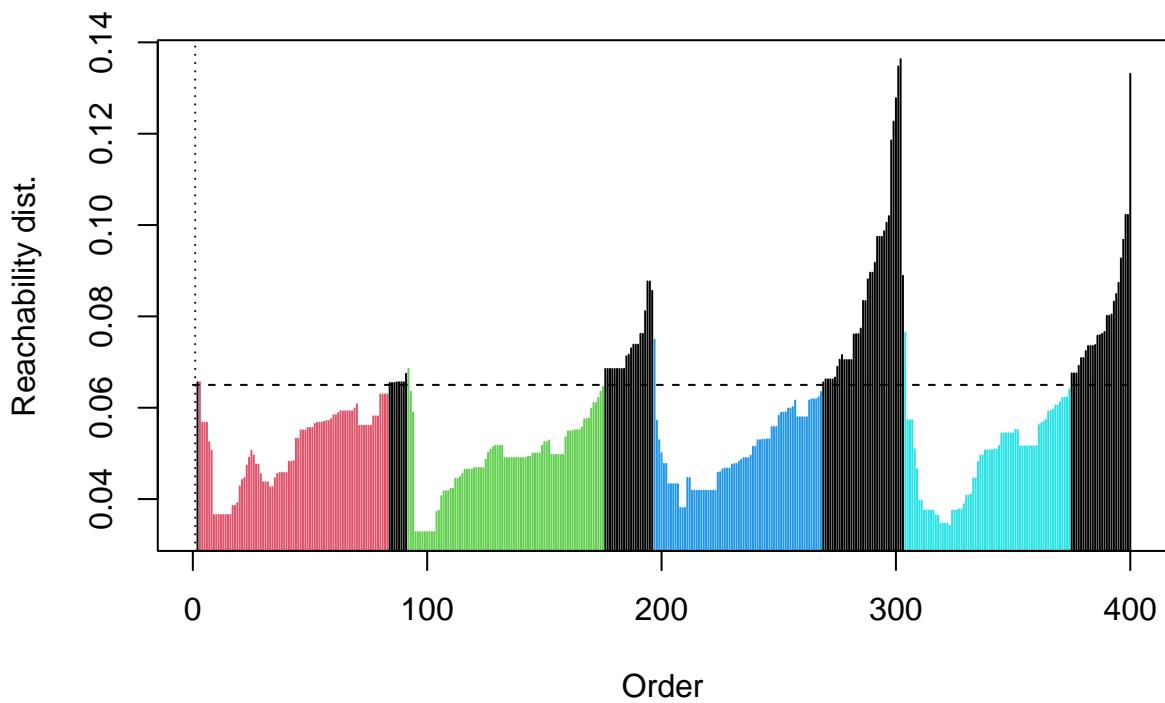
Otro ejercicio interesante a realizar es extraer una agrupación de la ordenación realizada por OPTICS similar a lo que DBSCAN hubiera generado estableciendo el parámetro `eps` en `eps_cl = 0.065`. En este

sentido animamos al estudiante a experimentar con diferentes valores de este parámetro.

```
### Extracción de un clustering DBSCAN cortando la alcanzabilidad 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**

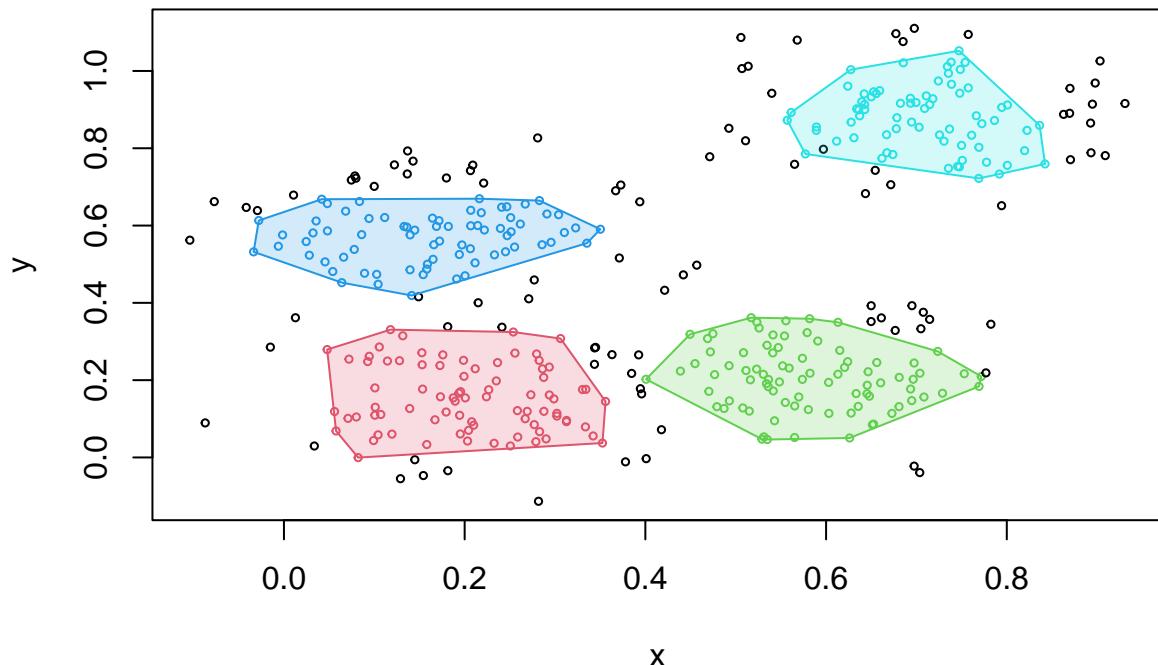


Observamos en el gráfico anterior como se han coloreado los 4 clusters y en negro se mantienen los valores *outliers* o extremos.

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

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

## Convex Cluster Hulls

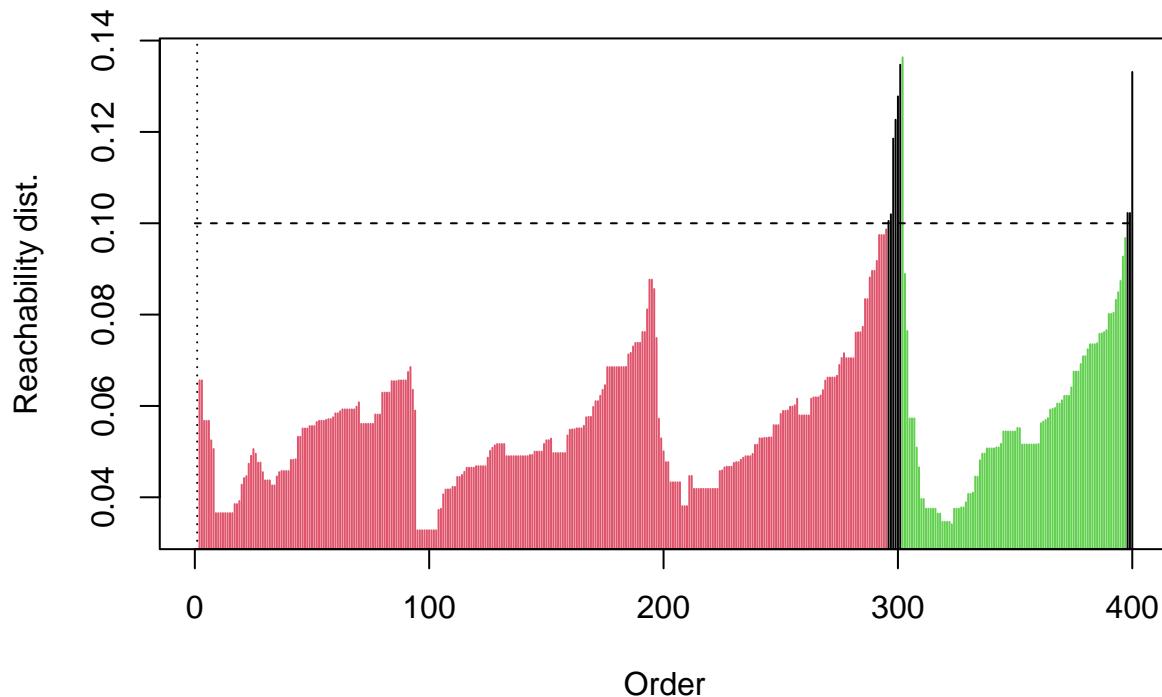


Repetimos el experimento anterior incrementando el parámetro `epc_c`, veamos como el efecto que produce es la concentración de clusters ya que flexibilizamos la condición de densidad.

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

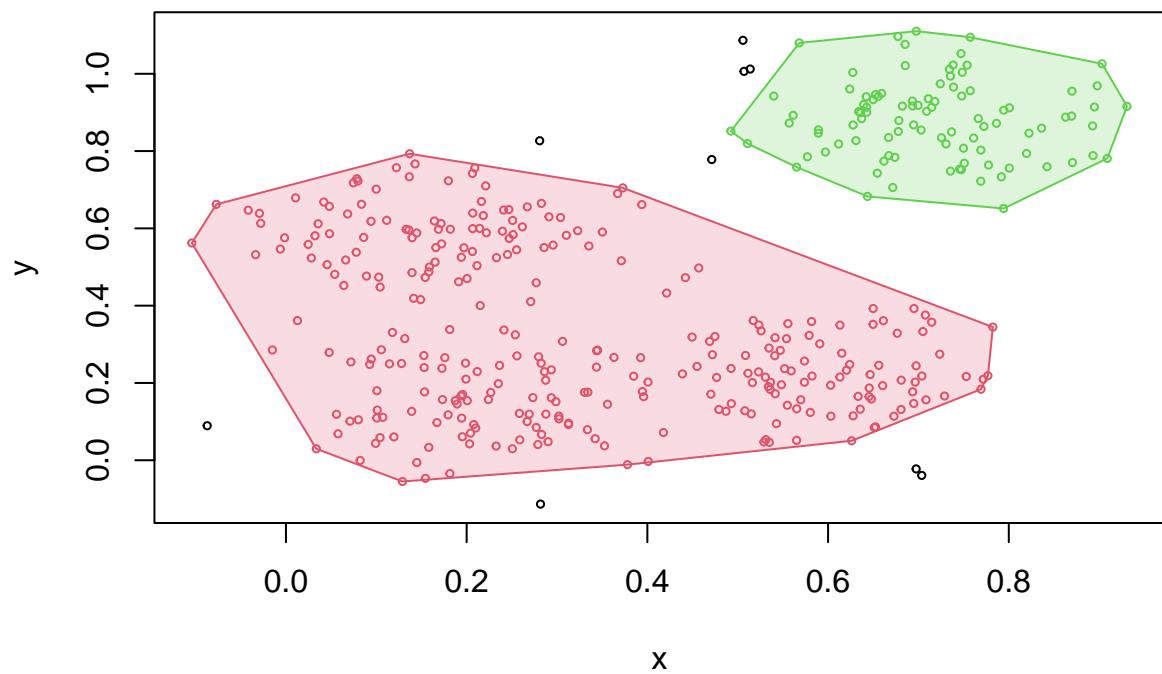
## OPTICS ordering/clustering for 400 objects.
## Parameters: minPts = 10, eps = 0.193786846197958, eps_cl = 0.1, xi = NA
## The clustering contains 2 cluster(s) and 9 noise points.
##
##      0   1   2
##      9 295  96
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   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  $xi$  nos va a servir para clasificar los clusters en función del cambio en la densidad relativa de los mismos.

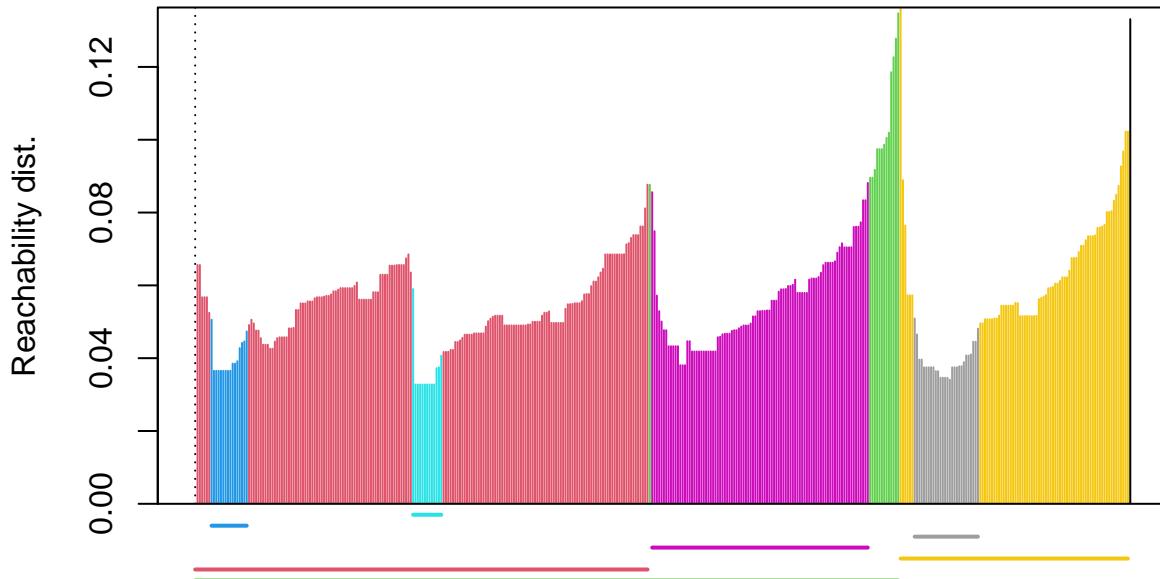
```

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

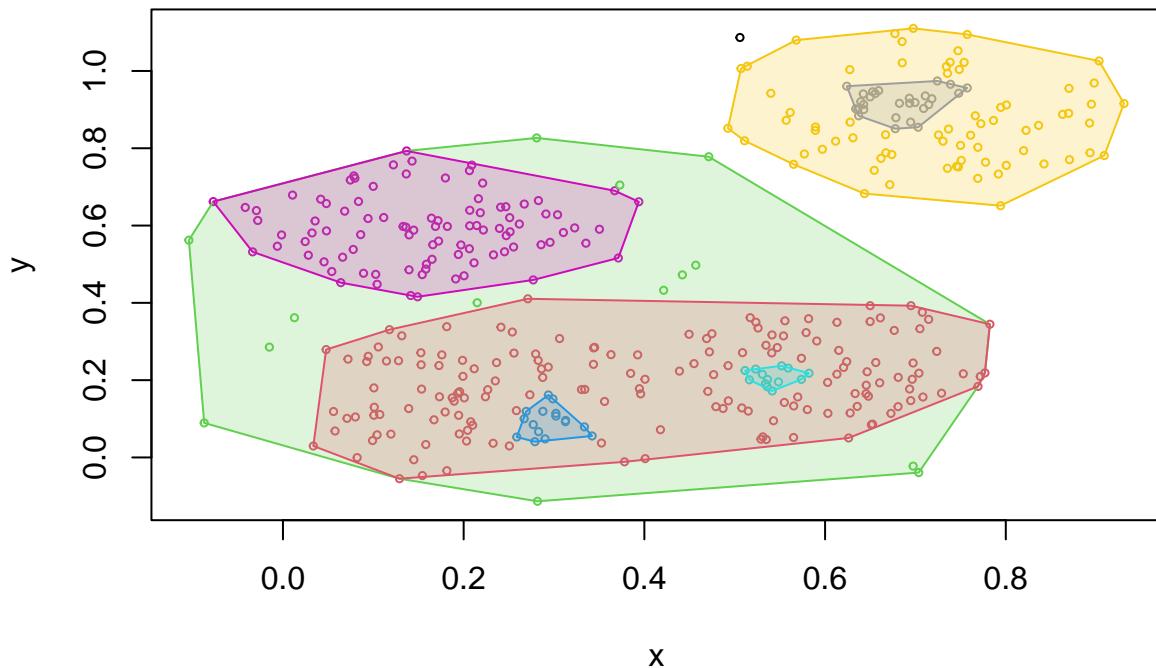
```

## Reachability Plot



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

## Convex Cluster Hulls



## 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	
##	Min. : 8.000	Min. : 1.00	Min. :1992	11:35 : 14	:842	
##	1st Qu.: 9.000	1st Qu.: 9.00	1st Qu.:1995	13:30 : 14	11:00 : 2	
##	Median :10.000	Median :16.00	Median :1999	11:45 : 13	11:35 : 2	
##	Mean : 9.843	Mean :15.74	Mean :1998	12:10 : 13	12:05 : 2	
##	3rd Qu.:10.000	3rd Qu.:23.00	3rd Qu.:2001	14:00 : 13	12:50 : 2	
##	Max. :11.000	Max. :31.00	Max. :2003	13:05 : 12	13:32 : 2	
##				(Other):829	(Other): 56	
##	BandNumber	Species	Age	Sex	Wing	Weight
##	: 2	CH: 70	A:224	:576	Min. : 37.2	Min. : 56.0
##	1142-09240:	1	RT:577	I:684	F:174	1st Qu.:202.0
##						1st Qu.: 185.0

```

## 1142-09241: 1 SS:261          M:158 Median :370.0 Median : 970.0
## 1142-09242: 1                  Mean   :315.6 Mean   : 772.1
## 1142-18229: 1                  3rd Qu.:390.0 3rd Qu.:1120.0
## 1142-19209: 1                  Max.   :480.0 Max.   :2030.0
## (Other)  :901                  NA's    :1    NA's   :10
##      Culmen       Hallux        Tail     StandardTail
## Min.   : 8.6   Min.   : 9.50   Min.   :119.0  Min.   :115.0
## 1st Qu.:12.8  1st Qu.:15.10  1st Qu.:160.0  1st Qu.:162.0
## Median :25.5  Median :29.40  Median :214.0  Median :215.0
## Mean   :21.8  Mean   :26.41  Mean   :198.8  Mean   :199.2
## 3rd Qu.:27.3  3rd Qu.:31.40  3rd Qu.:225.0  3rd Qu.:226.0
## Max.   :39.2  Max.   :341.40 Max.   :288.0  Max.   :335.0
## NA's    : 7    NA's   : 6    NA's   :337
##      Tarsus      WingPitFat    KeelFat      Crop
## Min.   :24.70  Min.   :0.0000  Min.   :0.000  Min.   :0.0000
## 1st Qu.:55.60 1st Qu.:0.0000  1st Qu.:2.000  1st Qu.:0.0000
## Median :79.30  Median :1.0000  Median :2.000  Median :0.0000
## Mean   :71.95  Mean   :0.7922  Mean   :2.184  Mean   :0.2345
## 3rd Qu.:87.00  3rd Qu.:1.0000  3rd Qu.:3.000  3rd Qu.:0.2500
## Max.   :94.00  Max.   :3.0000  Max.   :4.000  Max.   :5.0000
## NA's    :833   NA's   :831   NA's   :341   NA's   :343

```

## Ejercicio 1

Presenta el juego de datos, nombre y significado de cada columna, así como las distribuciones de sus valores.

Realiza un estudio aplicando el método K-means, similar al de los ejemplos 1.1 y 1.2.

### Respuesta 1

En este ejercicio se van a realizar las siguientes tareas:

- Se explican los campos de la base de datos
- Aplicación correcta del algoritmo K-MEANS
- Variación del valor k
- Estudio del impacto de los valores extremos
- Se mide lo bien/bueno que es el grupo
- Descripción e interpretación de los clusters obtenidos

Primero se comienza con la **explicación de la base de datos**. Para ello se ha generado el siguiente chunk de código:

```
#Ya se ha exportado la librería donde se encuentra la base de datos con la que se va a trabajar, por lo
names(Hawks)
```

```

## [1] "Month"      "Day"        "Year"        "CaptureTime" "ReleaseTime"
## [6] "BandNumber"  "Species"     "Age"         "Sex"         "Wing"
## [11] "Weight"      "Culmen"      "Hallux"      "Tail"        "StandardTail"
## [16] "Tarsus"      "WingPitFat"  "KeelFat"     "Crop"

```

```
#Ahora se van a estudiar las primeras posiciones de las columnas con los que se nos especifica arriba y
head(Hawks[c("Wing", "Weight", "Culmen", "Hallux")])
```

```

## Wing Weight Culmen Hallux
## 1 385    920   25.7   30.1

```

```

## 2 376 930 NA NA
## 3 381 990 26.7 31.3
## 4 265 470 18.7 23.5
## 5 205 170 12.5 14.3
## 6 412 1090 28.5 32.2
tail(Hawks[c("Wing", "Weight", "Culmen", "Hallux")])

##      Wing Weight Culmen Hallux
## 903   366    805   23.5  25.7
## 904   380   1525   26.0  27.6
## 905   190    175   12.7  15.4
## 906   360    790   21.9  27.6
## 907   369    860   25.2  28.0
## 908   199   1290   28.7  32.1

summary(Hawks)

##      Month          Day          Year        CaptureTime     ReleaseTime
## Min.   : 8.000   Min.   : 1.00   Min.   :1992  11:35   : 14       :842
## 1st Qu.: 9.000   1st Qu.: 9.00   1st Qu.:1995 13:30   : 14   11:00   :  2
## Median :10.000   Median :16.00   Median :1999 11:45   : 13   11:35   :  2
## Mean   : 9.843   Mean   :15.74   Mean   :1998 12:10   : 13   12:05   :  2
## 3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:2001 14:00   : 13   12:50   :  2
## Max.   :11.000   Max.   :31.00   Max.   :2003 13:05   : 12   13:32   :  2
## (Other):829      (Other):56
##      BandNumber Species Age Sex      Wing      Weight
## : 2      CH: 70  A:224 :576  Min.   : 37.2  Min.   : 56.0
## 1142-09240: 1  RT:577 I:684 F:174  1st Qu.:202.0 1st Qu.: 185.0
## 1142-09241: 1 SS:261 M:158  Median :370.0  Median : 970.0
## 1142-09242: 1           Mean   :315.6  Mean   : 772.1
## 1142-18229: 1           3rd Qu.:390.0 3rd Qu.:1120.0
## 1142-19209: 1           Max.   :480.0  Max.   :2030.0
## (Other)   :901           NA's   : 1    NA's   :10
##      Culmen        Hallux        Tail      StandardTail
## Min.   : 8.6   Min.   : 9.50   Min.   :119.0  Min.   :115.0
## 1st Qu.:12.8  1st Qu.: 15.10  1st Qu.:160.0 1st Qu.:162.0
## Median :25.5  Median : 29.40  Median :214.0  Median :215.0
## Mean   :21.8  Mean   : 26.41  Mean   :198.8  Mean   :199.2
## 3rd Qu.:27.3  3rd Qu.: 31.40  3rd Qu.:225.0 3rd Qu.:226.0
## Max.   :39.2  Max.   :341.40  Max.   :288.0  Max.   :335.0
## NA's   : 7    NA's   : 6      NA's   :337
##      Tarsus        WingPitFat      KeelFat      Crop
## Min.   :24.70   Min.   :0.0000   Min.   :0.000  Min.   :0.0000
## 1st Qu.:55.60  1st Qu.:0.0000  1st Qu.:2.000 1st Qu.:0.0000
## Median :79.30  Median :1.0000  Median :2.000  Median :0.0000
## Mean   :71.95  Mean   :0.7922  Mean   :2.184  Mean   :0.2345
## 3rd Qu.:87.00  3rd Qu.:1.0000  3rd Qu.:3.000 3rd Qu.:0.2500
## Max.   :94.00  Max.   :3.0000  Max.   :4.000  Max.   :5.0000
## NA's   :833    NA's   :831    NA's   :341    NA's   :343

```

Como se puede comprobar por el output del chunk de arriba, en las columnas **Culmen** y **Hallux** puede verse como en la segunda posición hay celdas vacías. Por ello, como ya se aprendió en la primera PEC, se va a comprobar si hay valores NA (no disponibles) también llamados NULOS en las columnas de datos con las que se va a trabajar e implementar luego los algoritmos estudiados en teoría. También se va a comprobar si hay celdas vacías.

```

cat("Comprobación de celdas con valores nulos:")

## Comprobación de celdas con valores nulos:
cat("\nLa columna Wing tiene:",{colSums(is.na(Hawks))["Wing"]},"valores nulos")

##
## La columna Wing tiene: 1 valores nulos
cat("\nLa columna Weight tiene:",{colSums(is.na(Hawks))["Weight"]},"valores nulos")

##
## La columna Weight tiene: 10 valores nulos
cat("\nLa columna Culmen tiene:",{colSums(is.na(Hawks))["Culmen"]},"valores nulos")

##
## La columna Culmen tiene: 7 valores nulos
cat("\nLa columna Hallux tiene:",{colSums(is.na(Hawks))["Hallux"]},"valores nulos")

##
## La columna Hallux tiene: 6 valores nulos
cat("\n\nComprobación de celdas con valores vacíos:")

##
##
## Comprobación de celdas con valores vacíos:
cat("\nLa columna Wing tiene:",colSums(Hawks["Wing"]=="")),"valores nulos")

##
## La columna Wing tiene: NA valores nulos
cat("\nLa columna Weight tiene:",colSums(Hawks["Weight"]=="")),"valores nulos")

##
## La columna Weight tiene: NA valores nulos
cat("\nLa columna Culmen tiene:",colSums(Hawks["Culmen"]=="")),"valores nulos")

##
## La columna Culmen tiene: NA valores nulos
cat("\nLa columna Hallux tiene:",colSums(Hawks["Hallux"]=="")),"valores nulos")

##
## La columna Hallux tiene: NA valores nulos
cat("\n\n Valores vacíos para todas las columnas: \n")

##
##
## Valores vacíos para todas las columnas:
colSums(Hawks == "")

```

	Month	Day	Year	CaptureTime	ReleaseTime	BandNumber
##	0	0	0	0	1	0
##	Species	Age	Sex	Wing	Weight	Culmen
##	0	0	576	NA	NA	NA

```

##      Hallux      Tail StandardTail      Tarsus WingPitFat      KeelFat
##      NA          0        NA          NA        NA        NA
##      Crop
##      NA

#for(i in {colSums(is.na(Hawks[c("Wing", "Weight", "Culmen", "Hallux")]))}){
#cat("La columna",i,"tiene:",{colSums(is.na(Hawks))[i]},"valores nulos")

```

Puede observarse como todos los campos tienen valores nulos, y para poder tratar con estos datos, los vamos a tener que cumplimentar correctamente, con técnicas ya vistas en el bloque pasado referente a la PEC1, como por ejemplo, calculando la media del resto de valores del campo y añadirlo, o con la mediana. Como se puede ver en la segunda parte del código, no hay ninguna celda vacía en las columnas.

Por lo tanto, lo que se va a hacer va a ser cumplimentar esos valores NULOS, por la media del resto.

```

Hawks2 <- Hawks[,10:13] #Seleccionamos las 4 columnas con las que se va a trabajar.
head(Hawks2)

##   Wing Weight Culmen Hallux
## 1  385     920   25.7  30.1
## 2  376     930    NA    NA
## 3  381     990   26.7  31.3
## 4  265     470   18.7  23.5
## 5  205     170   12.5  14.3
## 6  412    1090   28.5  32.2

medias <- colMeans(Hawks2, na.rm = TRUE) #Calcular la media de cada columna
print(medias)

##      Wing     Weight     Culmen     Hallux
## 315.63749 772.08018 21.80150 26.41086

# Rellenar los valores NA con la media de cada columna
for (i in 1:ncol(Hawks2)) {
  Hawks2[is.na(Hawks2[, i]), i] <- medias[i]
}

cat("\n\nComprobación de nuevo de las celdas con valores nulos:")

##
##
## Comprobación de nuevo de las celdas con valores nulos:
cat("\nLa columna Wing tiene:",{colSums(is.na(Hawks2))["Wing"]}, "valores nulos")

##
## La columna Weight tiene: 0 valores nulos
cat("\nLa columna Weight tiene:",{colSums(is.na(Hawks2))["Weight"]}, "valores nulos")

##
## La columna Culmen tiene: 0 valores nulos
cat("\nLa columna Culmen tiene:",{colSums(is.na(Hawks2))["Culmen"]}, "valores nulos")

##
## La columna Hallux tiene: 0 valores nulos

```

```
cat("\nLa columna Hallux tiene:", colSums(is.na(Hawks2))["Hallux"], "valores nulos")
```

```
##
```

```
## La columna Hallux tiene: 0 valores nulos
```

```
head(Hawks2)
```

```
##   Wing Weight Culmen Hallux
## 1  385     920 25.7000 30.10000
## 2  376     930 21.8015 26.41086
## 3  381     990 26.7000 31.30000
## 4  265     470 18.7000 23.50000
## 5  205     170 12.5000 14.30000
## 6  412    1090 28.5000 32.20000
```

```
tail(Hawks2)
```

```
##   Wing Weight Culmen Hallux
## 903 366     805 23.5   25.7
## 904 380    1525 26.0   27.6
## 905 190     175 12.7   15.4
## 906 360     790 21.9   27.6
## 907 369     860 25.2   28.0
## 908 199    1290 28.7   32.1
```

Como se puede comprobar por el output del chunk de arriba, se han conseguido llenar los valores nulos con la media de los datos dentro de la columna a la que pertenece cada celda nula. Me acabo de dar cuenta de que la propia función **summary( · )** ya daba información acerca el número de celdas con valores no válidos (i.e., **NA**) en la columna. En el caso de tener algún problema en el futuro con la implementación de los algoritmos, y los datos que se hayan cumplimentado no fuesen los mejores, se podría calcular la mediana de los valores y reemplazarla por la media.

Ahora que se han cumplimentado los datos erróneos, se explican los campos de la base de datos, más en específico los campos con los que se nos dice que tenemos que trabajar. Para ello también he implementado esta linea de código, para saber si hay más información:

```
structure = str(Hawks2)
```

```
## 'data.frame': 908 obs. of 4 variables:
## $ Wing : num 385 376 381 265 205 412 370 375 412 405 ...
## $ Weight: num 920 930 990 470 170 1090 960 855 1210 1120 ...
## $ Culmen: num 25.7 21.8 26.7 18.7 12.5 ...
## $ Hallux: num 30.1 26.4 31.3 23.5 14.3 ...
```

Como se puede ver, todas las columnas albergan datos de tipo num, i.e., es decir, datos decimales. Pero no hay información de alto nivel, que refleje más detalles acerca de cada una de las columnas de la base de datos que se va a estudiar. Para ello se ha consultado la siguiente página web, dónde se explica un poco más en detalle los campos de la base de datos: <https://rdrr.io/rforge/Stat2Data/man/Hawks.html>

- **Month:** Hace referencia al mes.
- **Day:** Hace referencia al día del mes.
- **Year:** Hace referencia al año.
- **CaptureTime:** Hace referencia a la hora a la que fue capturada el halcón.
- **ReleaseTime:** Hace referencia a la hora a la que se liberó el halcón.

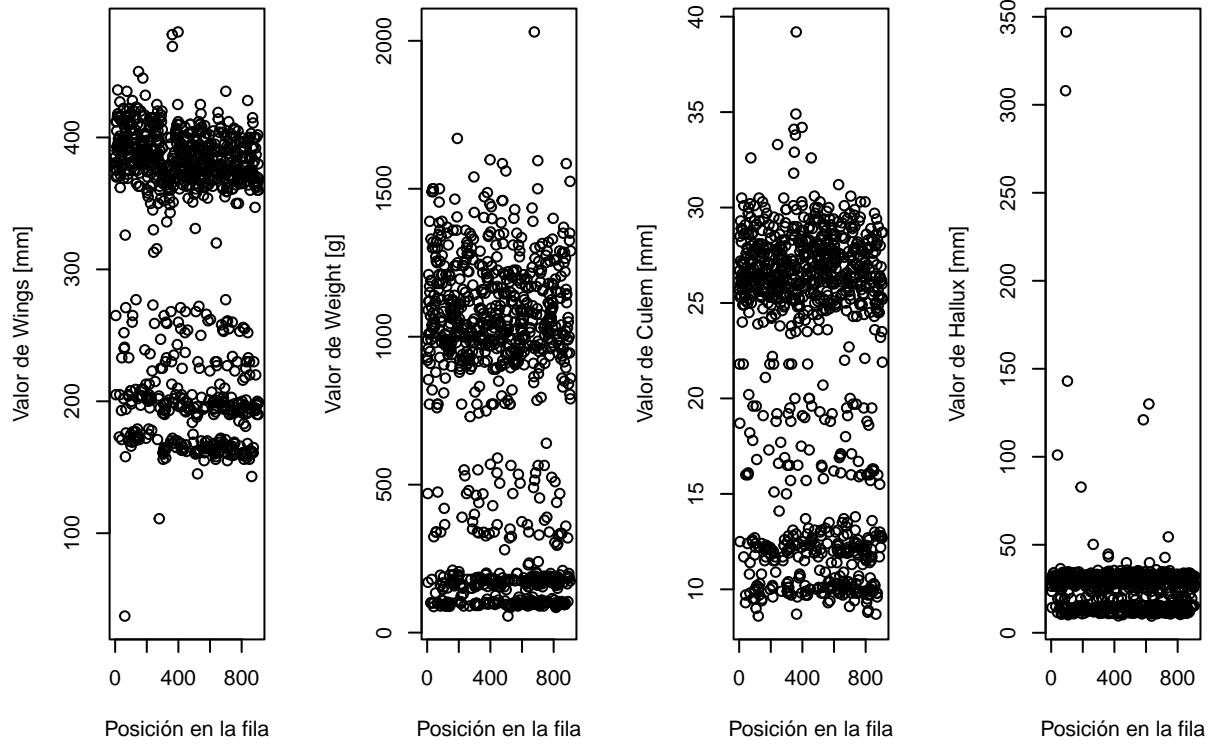
- **Species:** La especie del halcón, en este caso; “CH: Cooper’s”, “RT: Red-Tailed” and “SS: Sharp-Sinned”
- **Age:** Hace referencia a la edad, este campo puede tomar dos valores: “I: Imature” o “A: Adult”
- **Sex:** El sexo del halcón, que puede ser: “F: Female” o “M: Male”
- **Wing:** Este apartado, como ya se intuye por el nombre que recibe, tiene que ver con el ala, más concretamente, con la longitud en **mm** de la pluma primaria del ala desde la punta hasta la muñeca a la que se une.
- **Weight:** Hace referencia al peso del cuerpo en gramos (g)
- **Culmen:** Hace referencia a la longitud de la parte superior del pico desde la punta hasta la parte carnosa del halcón y se mide en mm.
- **Hallux:** Hace referencia a la longitud de la garra “asesina” del halcón.
- **Tail:** Medida en milímetros, relativa a la longitud de la cola (inventada en el MacBride Raptor Center)
- **StandardTail:** Hace referencia a la medición estandarizada de la longitud de la cola, en mm.
- **Tarsus:** Longitud del hueso básico del pie, en mm.
- **WingPitFat:** Se refiere a la cantidad de grasa en el hueso del ala.
- **KeelFat:** Refleja la cantidad de grasa en el esternón, medida al tacto.
- **Crop:** Cantidad de material en el cultivo, este atributo esta codificado como 1= “lleno” o 0= “vacío”

Ahora que ya tenemos contexto acerca de la base de datos con la que vamos a trabajar, vamos a aplicar el algoritmo de los k-means, para ello se vuelve a cargar de nuevo la librería (solo por si acaso ya que ya se cargó en el primer chunk de código del proyecto)

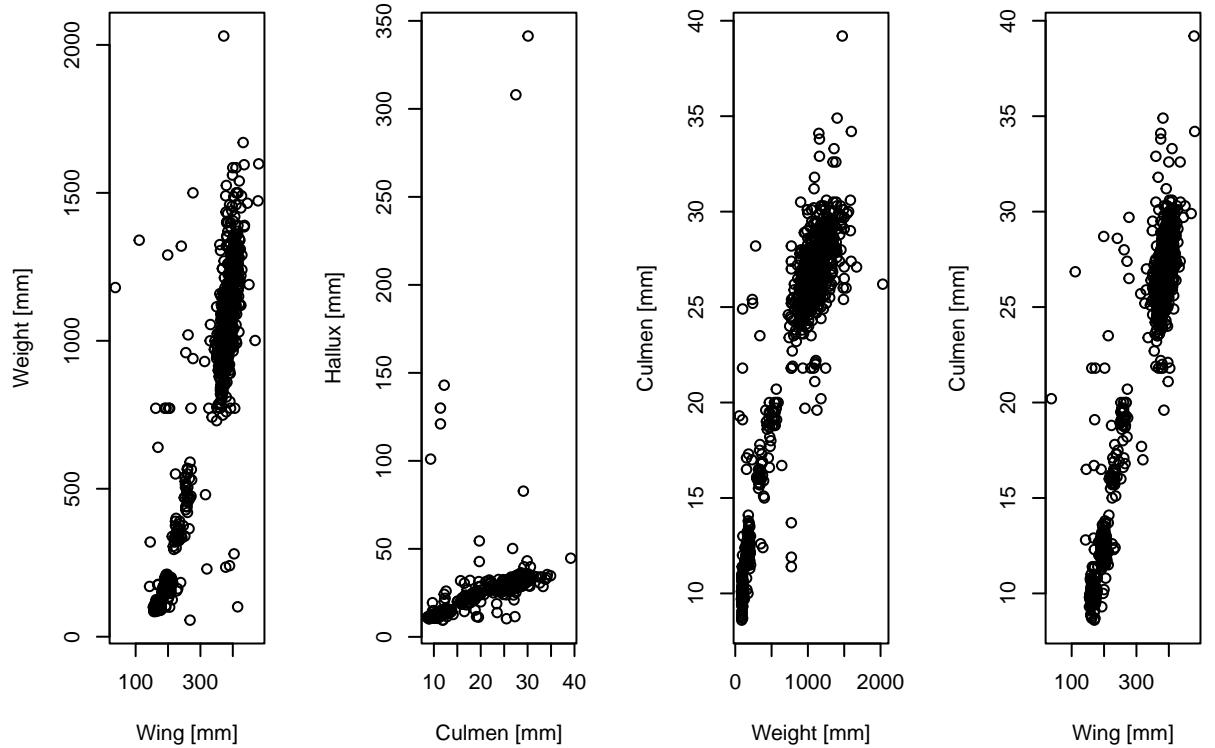
```
if (!require('cluster')) install.packages('cluster')
library(cluster)
```

Ahora se van a representar los datos de las 4 columnas

```
x <- rbind(Hawks2$Wing,Hawks2$Weight,Hawks2$Culmen,Hawks2$Hallux)
par(mfrow = c(1, 4))
plot(Hawks2$Wing,xlab="Posición en la fila", ylab = "Valor de Wings [mm]")
plot(Hawks2$Weight,xlab="Posición en la fila", ylab = "Valor de Weight [g]")
plot(Hawks2$Culmen,xlab="Posición en la fila", ylab = "Valor de Culem [mm]")
plot(Hawks2$Hallux,xlab="Posición en la fila", ylab = "Valor de Hallux [mm]")
```



```
par(mfrow = c(1, 4))
#Ahora se va a probar a representar un atributo frente a otro:
plot(Hawks2$Wing,Hawks2$Weight,xlab="Wing [mm]", ylab = "Weight [mm]")
plot(Hawks2$Culmen,Hawks2$Hallux,xlab="Culmen [mm]", ylab = "Hallux [mm]")
plot(Hawks2$Weight,Hawks2$Culmen,xlab="Weight [mm]", ylab = "Culmen [mm]")
plot(Hawks2$Wing,Hawks2$Culmen,xlab="Wing [mm]", ylab = "Culmen [mm]")
```



Con esta representación, se pueden observar las zonas más densamente pobladas así como los valores sobre los cuales orbitan cada uno de los 4 atributos. Para el caso de la longitud de las alas el valor más poblado es el rango de los 400 +/- mm, y se observan 3 grupos de valores principales (coincide con el número de especies), se pueden observar además los llamados outliers. Luego en el caso del peso, i.e., “weight” hay tres zonas al igual que en el anterior atributo, no obstante en este caso parece que el primer grupo de halcones correspondiente al grupo de valores de menor magnitud, está entre los 100-200 gramos, luego hay un grupo intermedio pero considerablemente menos poblado como es el del rango de los 500 +/- gramos y por último se encuentra el grupo más poblado (al menos a simple vista) que es el relativo al rango de los halcones con un peso comprendido entre los (900-1400) +/- gramos. En el caso del atributo referente a la longitud del pico (Culem) se ve también una clara división entre tres grupos de valores, por cierto muy similar al del atributo anterior. Por último en el caso del atributo de la longitud de la garra asesina (Hallux) se ve como solo existen dos grupos de valores notables y es el atributo del halcón que menos incertidumbre tiene, en este atributo también se observan claramente algunos outliers como el de dos valores de aproximadamente 300 y 350 mm correspondientemente, estas medidas podrían tratarse de otros halcones.

Como resulta interesante ver un comportamiento similar en el segundo y tercer atributo de arriba se ha decidido representar estos dos atributos uno frente al otro, el resultado se puede ver en la segunda página de los “plots de arriba”. Viendo el resultado de la relación entre los dos atributos: “Weight” y “Culmen” puede observarse efectivamente una relación casi lineal. Algo parecido parece que ocurre para los atributos Weight y Wing, aunque se acercaría también a un comportamiento exponencial. En el caso del plot de los atributos relacionados en el plot de en medio: “Culmen” y “Hallux” se ve un crecimiento muy bajo del tamaño del Hallux respecto al atributo Culmen, lo que da a entender que el atributo Hallux suele ser un poco más estándar y no depende del resto de atributos físicos del halcón.

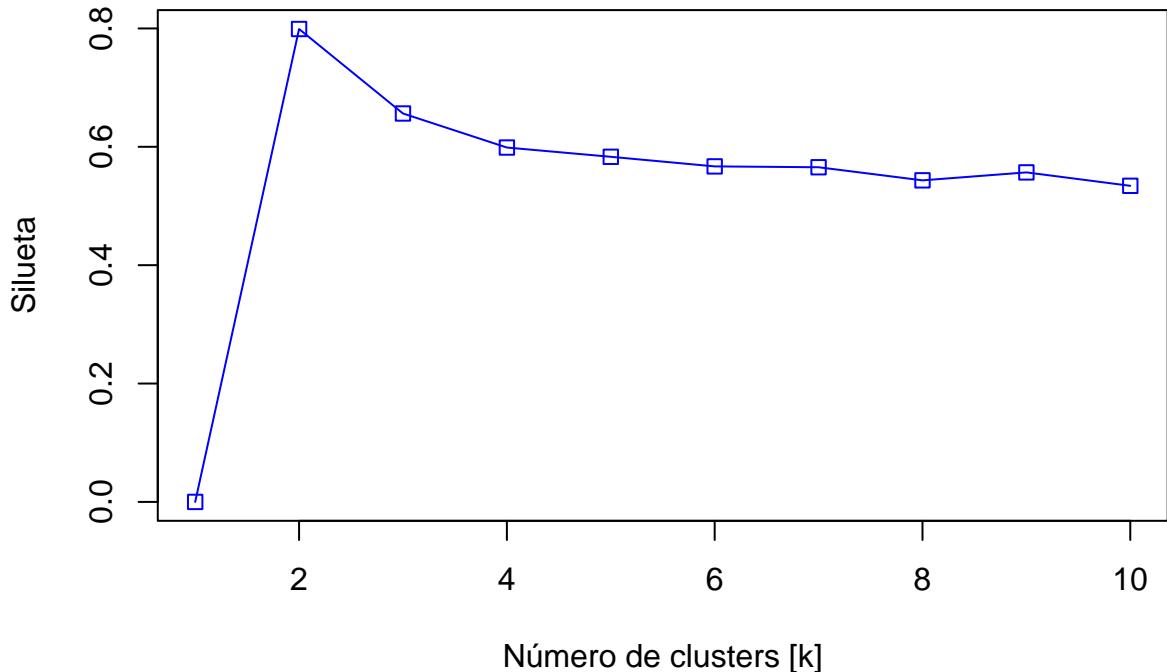
Habiendo realizado el primer análisis, ahora se procede con la implementación del algoritmo k-means.

```
#Ahora se prueba con diferentes valores de k
d <- daisy(Hawks2)
#Se podría también llevar a cabo con la siguiente función:
#d <- dist(Hawks2)
resultados <- rep(0, 10) #Se inicializa un vector lleno de 0 para luego poblarlo con los resultados de
for (i in c(2,3,4,5,6,7,8,9,10))
{
  fit           <- kmeans(Hawks2, i)
  y_cluster     <- fit$cluster
  sk            <- silhouette(y_cluster, d)
  resultados[i] <- mean(sk[,3])
}
cat(resultados)

## 0 0.7990271 0.656291 0.5987655 0.5831328 0.5668971 0.5654666 0.5433327 0.5567552 0.5341058

#Ahora se representan los valores que se han obtenido arriba:
plot(1:10,resultados,type="o",col="blue",pch=0,xlab="Número de clusters [k]",ylab="Silueta", main = "Pr
```

## Primer gráfico



```
p <- recordPlot()
```

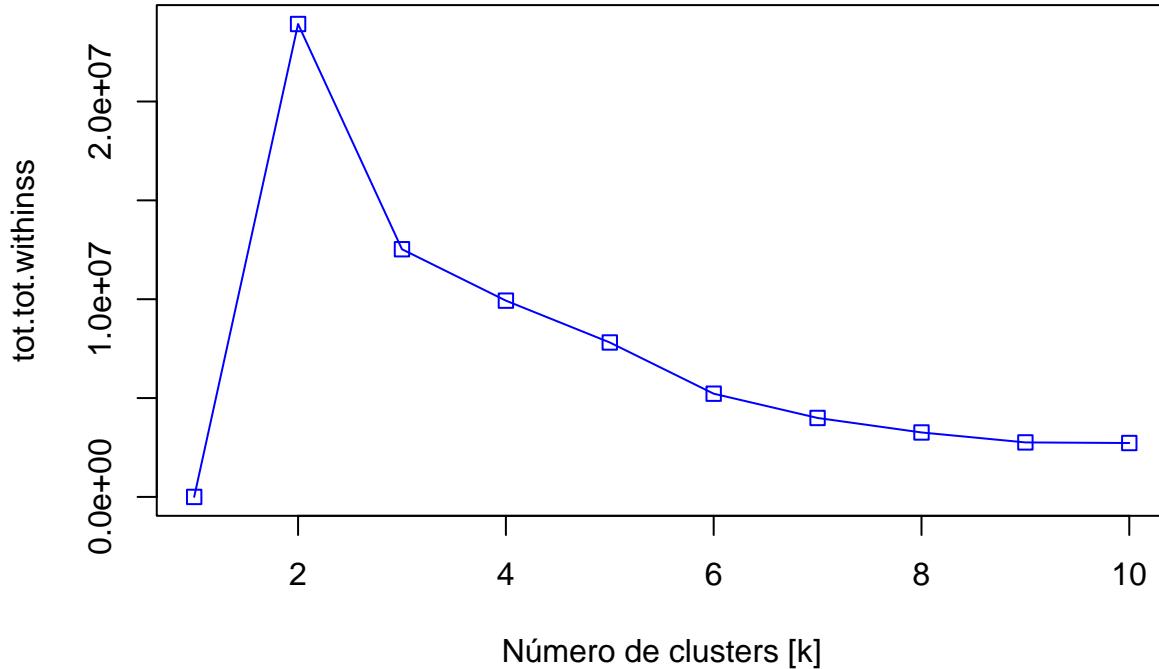
Como se puede ver por el chunk de arriba, la distancia entre muestras se ha calculado con la función “daisy( · )” pues a pesar de que realiza la misma función que “dist( · )”, “daisy( · )” permite/es compatible con otros tipos de variables (nominales, cardinales, etc) No obstante, se ha calculado la distancia también con la función dist( · ) y en este caso se han obtenido los mismos resultados.

Ahora hay que tener en cuenta que para saber como de bien se han clasificado las muestras en los clusters, se ha hecho uso del coeficiente de Silhouette. Como se sabe por teoría, este coeficiente, permite saber lo bien que está integrado un punto en el grupo en el que se encuentra, en definitiva, permite saber si ese punto pertenece o NO al grupo en el que actualmente se encuentra. Viendo el resultado de arriba, uno podría decir que el resultado óptimo en cuánto número de “clusters” sería 3, pues es en k=3 cuando la curva comienza a estabilizarse y dónde el valor de la silueta es mayor (0.8), ya que . No obstante, puede ser que en adelante se pueda observar como 3 clusters no sean suficientes para la tarea de clasificación. Es por ello que se va a probar también con la forma que sigue, para confirmar nuestras sospechas acerca del número óptimo de número de clusters.

Como se ha visto en el ejemplo guiado 1.2, se implementa el siguiente chunk:

```
resultados <- rep(0, 10) #Se inicializa un vector lleno de 0 para luego poblarlo con los resultados de los cluster
for (i in c(2,3,4,5,6,7,8,9,10))
{
  fit           <- kmeans(Hawks2, i)
  resultados[i] <- fit$tot.withinss
}
#Ahora se lleva a cabo la representación de la grafica y se compara con el gráfico obtenido en el chunk anterior
plot(1:10,resultados,type="o",col="blue",pch=0,xlab="Número de clusters [k]",ylab="tot.tot.withinss",main="Silueta")
```

## Segundo gráfico



Al principio se dijo que 3 clusters podrían ser la respuesta, no obstante, viendo el “Segundo gráfico” de arriba, puede ser que 4 clusters también resulten una buena opción, pues a partir de  $k=4$  parece que la curva empieza a estabilizarse y la diferencia de pendiente es muy pequeña respecto al resto de puntos en adelante. No obstante, se es consciente, que el llamado “codo” se encuentra en  $k=3$ , y además, como se ve en el ejemplo guiado, existen otras formas de calcular el número óptimo de clústers, como por ejemplo con el paquete **fpc**. Véase la implementación de este método en el chunk de abajo.

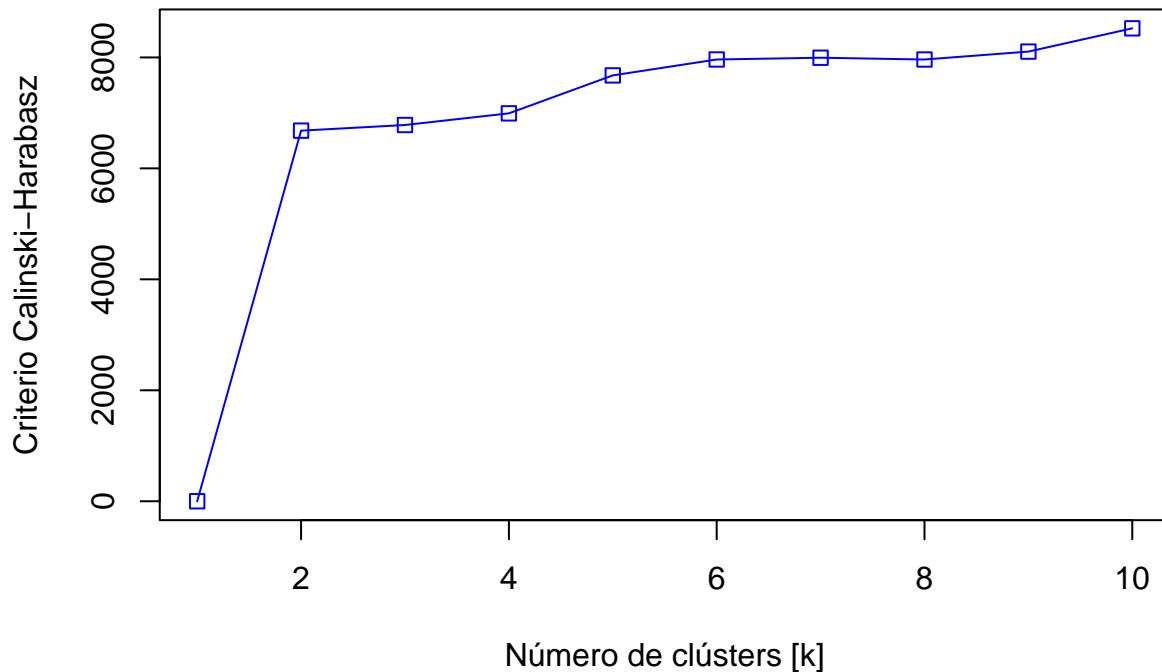
```
if (!require('fpc')) install.packages('fpc') #Se descarga la librería, en caso de que noe estuviese des
library(fpc)
fit_ch <- kmeansruns(Hawks2, krange = 1:10, criterion = "ch")
fit_asw <- kmeansruns(Hawks2, krange = 1:10, criterion = "asw")

#Ahora se comprueba el número óptimo k para los dos métodos:
cat("Este es el número óptimo de clústers para el método Calinski-Harabasz: ", fit_ch$bestk)

## Este es el número óptimo de clústers para el método Calinski-Harabasz: 10
cat("\nEste es el número óptimo de clústers para el método de la silueta media: ", fit_asw$bestk)

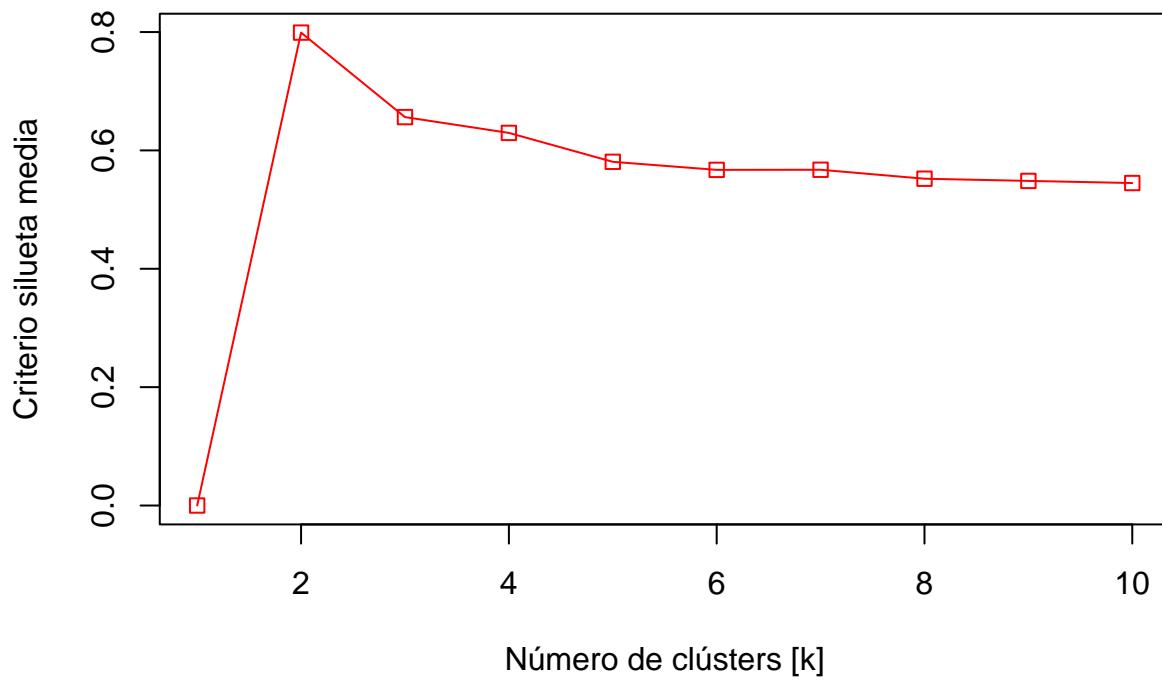
##
## Este es el número óptimo de clústers para el método de la silueta media: 2
#par(mfrow = c(1, 2))
#A continuación se representan los dos métodos:
plot(1:10, fit_ch$crit, type="o", col="blue", pch=0, xlab="Número de clústers [k]", ylab="Criterio Calinski-H
```

## Calinski–Harabasz



```
# Se agrega ahora la segunda curva del segundo método al gráfico existente
plot(1:10,fit_asw$crit,type="o",col="red",pch=0,xlab="Número de clústers [k]",ylab="Criterio silueta media")
```

## Silueta media



Con el método de **Calinski–Harabasz** se podría decir que el número óptimo (mínimo) de clústers podría ser **k=3**, puede verse además como esto también aplica para el método de la **silueta**

**media.** Ahora bien, para el caso del número óptimo (máximo) de clústers, en el caso del método **Calinski-Harabasz** podría ser **k=5** esto también aplica para el método de la **silueta media**, pues en ambos casos la pendiente en el punto **k=5** es mínima en comparación con los puntos en adelante.

Ahora ya se sabe el rango óptimo de valores de **k** :  $k \in [3,5]$  pero hay que recordar la columna/atributo que determinaba el tipo de halcón. Para ello se vuelve a mostrar la base de datos original.

```
head(Hawks2)
```

```
##   Wing Weight Culmen Hallux
## 1  385     920 25.7000 30.10000
## 2  376     930 21.8015 26.41086
## 3  381     990 26.7000 31.30000
## 4  265     470 18.7000 23.50000
## 5  205     170 12.5000 14.30000
## 6  412    1090 28.5000 32.20000
```

Como se puede ver, y tal y como se explicó al principio del ejercicio, es la columna de especies la que determina el tipo de halcón, y por lo que se puede ver a simple vista, el atributo **SPECIES** puede tomar tres valores, i.e.,  $species \in \{RT, SS, CH\}$ . De todos modos, en el siguiente chunk se verifican cada uno de los valores únicos de este atributo:

```
unique(Hawks$Species)
```

```
## [1] RT CH SS
## Levels: CH RT SS

#Ahora se crea otro subconjunto:
Hawks3 <- Hawks[,c("Species","Wing","Weight","Culmen","Hallux")]

#Ahora se eliminan las filas con valores NA:
medias2 <- colMeans(Hawks3[c("Wing","Weight","Culmen","Hallux")], na.rm = TRUE) #Calcular la media de c
print(medias2)
```

```
##      Wing     Weight     Culmen     Hallux
## 315.63749 772.08018 21.80150 26.41086

# Rellenar los valores NA con la media de cada columna
for (i in 1:ncol(Hawks2)) {
  Hawks2[is.na(Hawks2[, i]), i] <- medias2[i]
}

for (i in 2:ncol(Hawks3)) {
  Hawks3[is.na(Hawks3[, i]), i] <- medias2[i-1]
  cat(i)
}
```

```
## 2345
```

Como se puede ver en el resultado se verifica lo anterior, esto es:  $species \in \{RT, SS, CH\}$ . Sabiendo esto, y estudiando el ejemplo guiado 1.2, se van a representar las muestras para **k=3**, ya que hay 3 clases de halcones. Véase el siguiente chunk de código:

```
k=3
set.seed(1)
Hawks2ConKIgualA3 <- kmeans(Hawks2, k)
#colSums(is.na(Hawks3))["Species"]
```

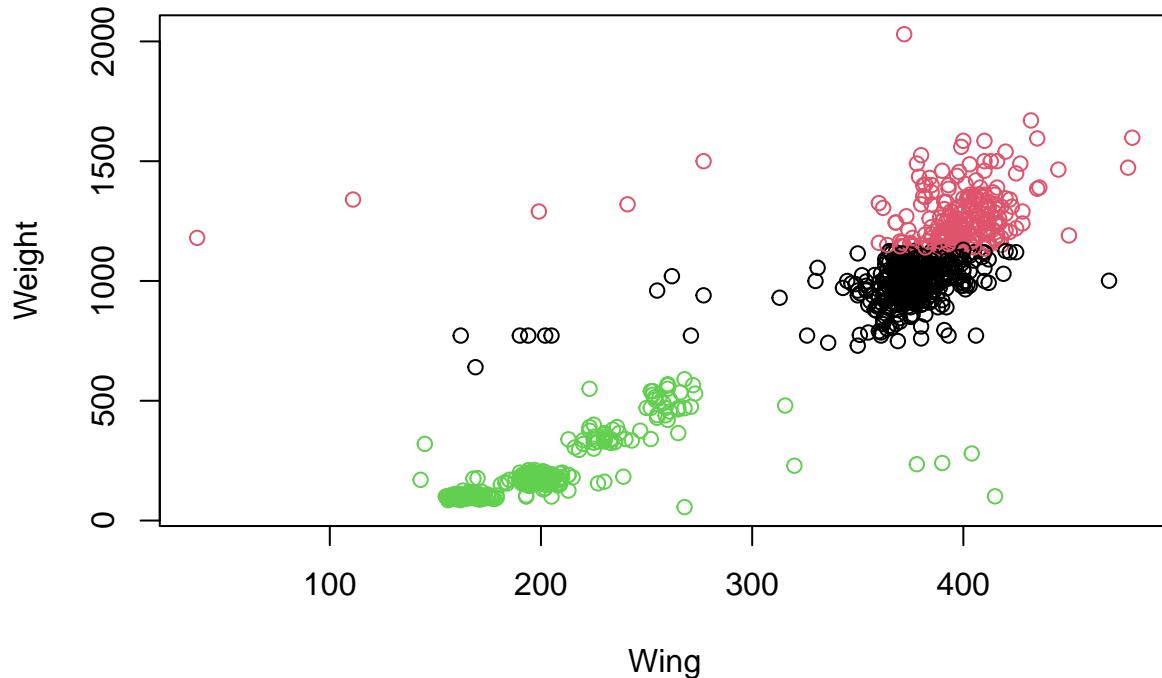
```

Hawks3ConKIgualA3 <- kmeans(Hawks3[,c("Wing", "Weight", "Culmen", "Hallux")], k)

# Wing y Weight con Hawks2
#Hawks2[c(1,2)]
plot(Hawks2[,c(1,2)], col=Hawks2ConKIgualA3$cluster, main="Clasificación k-means con k=3")

```

### Clasificación k-means con k=3

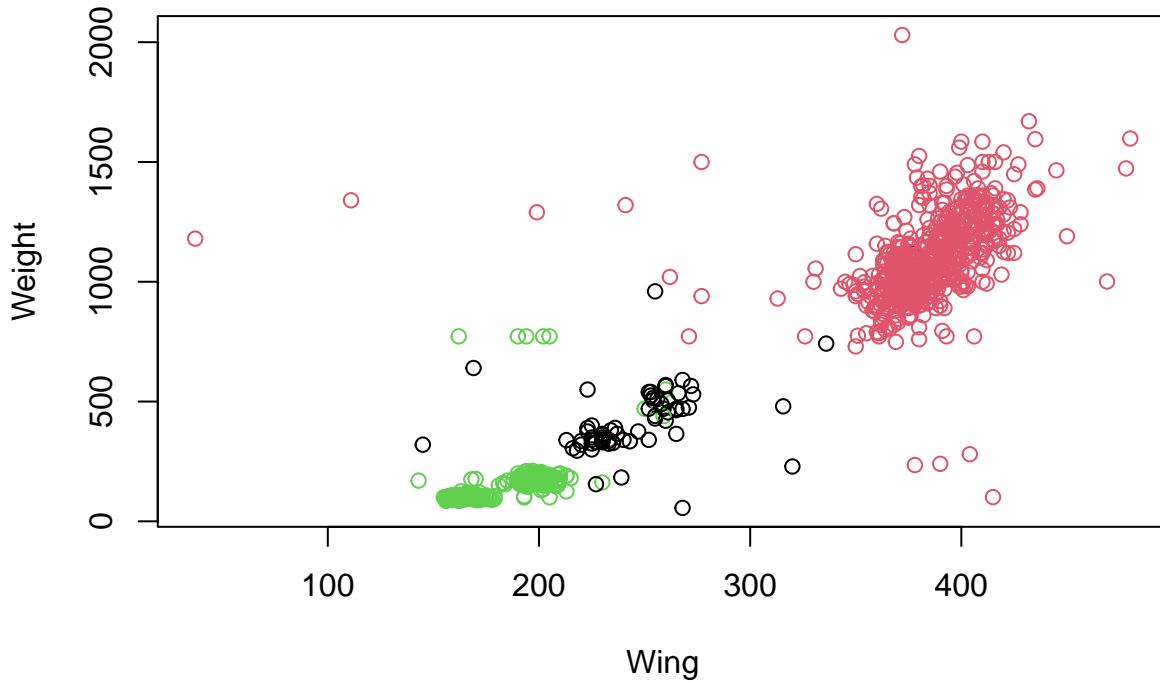


```

plot(Hawks2[,c(1,2)], col=as.factor(Hawks$Species), main="Clasificación real")

```

## Clasificación real



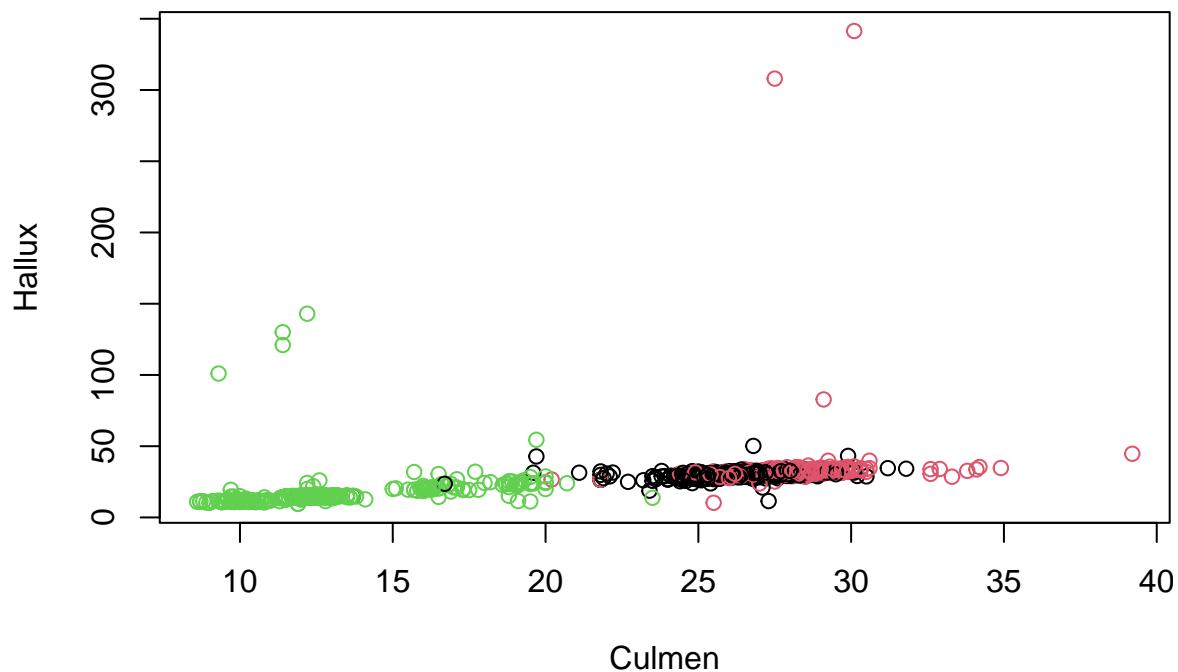
```
#NO VOLVER A EJECUTAR ESTA CELDA PORQUE, K-MEANS HA CONSEGUIDO UN BUEN RESULTADO YA QUE RESPETA LA GAMA

# Wing y Weight con Hawks3
#Hawks3[c(2,3)]
#plot(Hawks3[c(2,3)], col=Hawks3$cluster, main="Clasificación k-means con k=3")
#plot(Hawks3[c(2,3)], col=as.factor(Hawks3$Species), main="Clasificación real")
```

Se puede ver como los atributos **Wing** y **Weight** relacionados (como ya se vió anteriormente) guardan así una relación lineal, pues según aumenta un atributo, el otro también lo hace. Además estudiando la clasificación de estos, se ve como parecen diferenciar las tres especies de halcón: **RT: Red Tailed, SS: Sharped-Shinned** y **CH: Cooper's**, pero no correctamente del todo, ya que el algoritmo **k-means** clasifica bastantes muestras negras dentro del clúster de muestras verdes, y clasifica muestras del clúster rojo como parte del clúster de muestras negras. Cabe mencionar que aunque no son muy buenos resultados, parece que el algoritmo ha conseguido diferenciar a grandes rasgos las tres especies en comparación con la clasificación real. En la clasificación real, se observan muestras negras y verdes invadiendo sus respectivos espacios, así como algunas muestras negras y rojas, esto tiene un efecto negativo en el algoritmo de los **k-means**. Como era de esperar, también se pueden apreciar los famosos “outliers”, pero esto es algo que ya se observó al inicio del ejercicio. Finalmente, esto que se acaba de hacer para los dos primeros atributos, se repite para los otros dos atributos restantes, véase esto en el siguiente chunk de código:

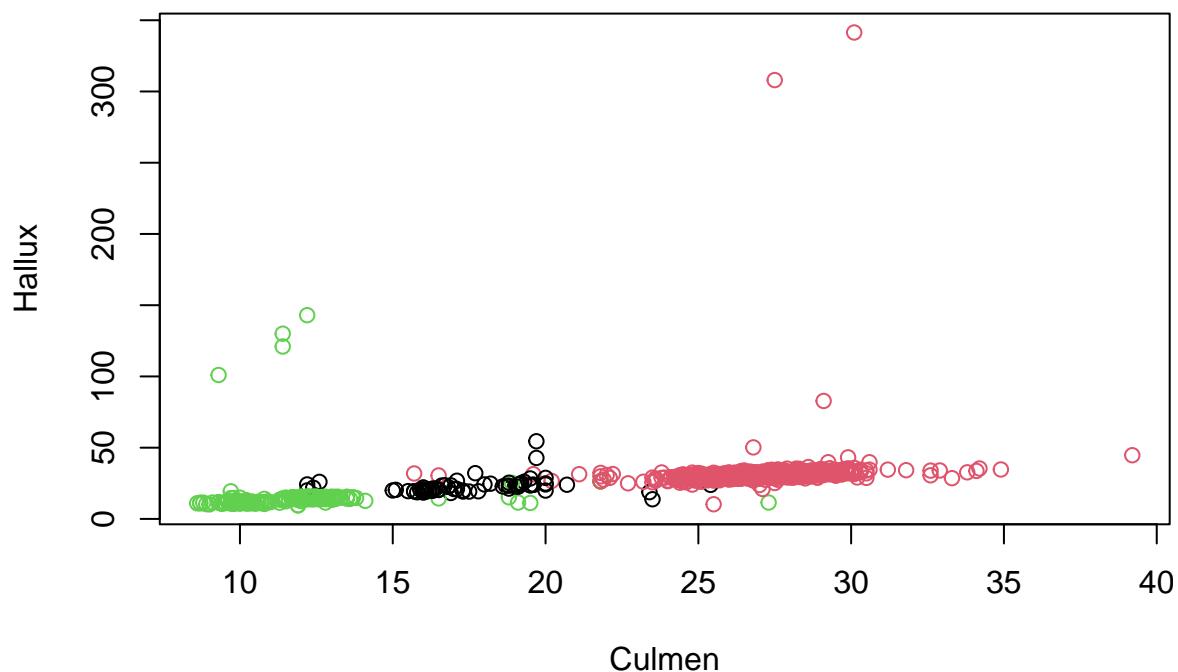
```
# Culmen y Hallux
plot(Hawks2[c(3,4)], col=Hawks2$cluster, main="Clasificación k-means con k=3")
```

## Clasificación k-means con k=3



```
plot(Hawks2[c(3,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```

## Clasificación real



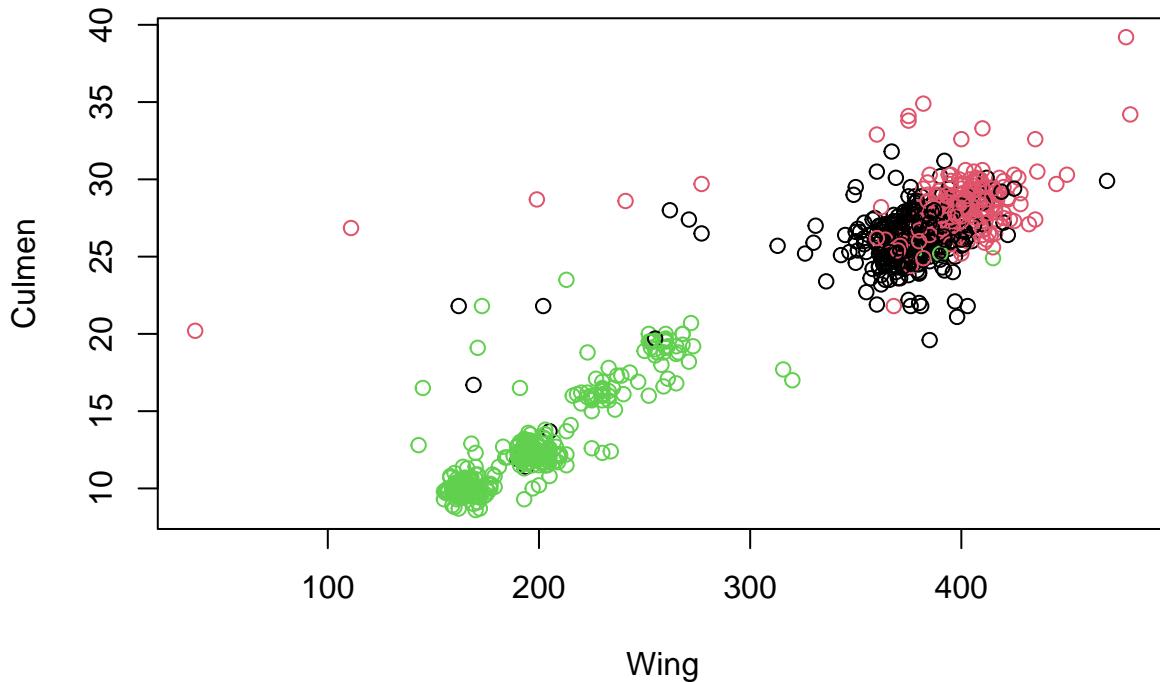
Para esta última pareja de atributos se puede ver como en ambos casos, tanto para la clasificación real, como para la del algoritmo de k-means, como hay una mayor mezcla entre muestras, resultando en un conjunto de clusters no muy bien definidos, y peor respecto a lo obtenido en el apartado anterior (**Wing** y **Weight**). Además puede verse como en la clasificación real las

muestras en rojo y negro se invaden entre sí, siendo las muestras rojas predominantes, en cambio para la clasificación mediante el algoritmo **k-means**, son las muestras negras las predominantes a la hora de superponerse con las rojas. Es pertinente mencionar, que el comportamiento observado para el par de atributos anterior, se repite en este otro par de atributos, es decir, el clúster de muestras verdes es mucho mayor, en comparación a la clasificación real, y el clúster de muestras rojas es mucho menor en comparación a la clasificación real. Esto significa que el algoritmo, está clasificando erróneamente un puñado considerable de muestras. Por lo tanto se puede culminar diciendo que la relación de estos dos atributos, no arroja tan buenos resultados en términos de clústers.

Por último se van a relacionar los 4 atributos mediante las dos últimas combinaciones posibles:

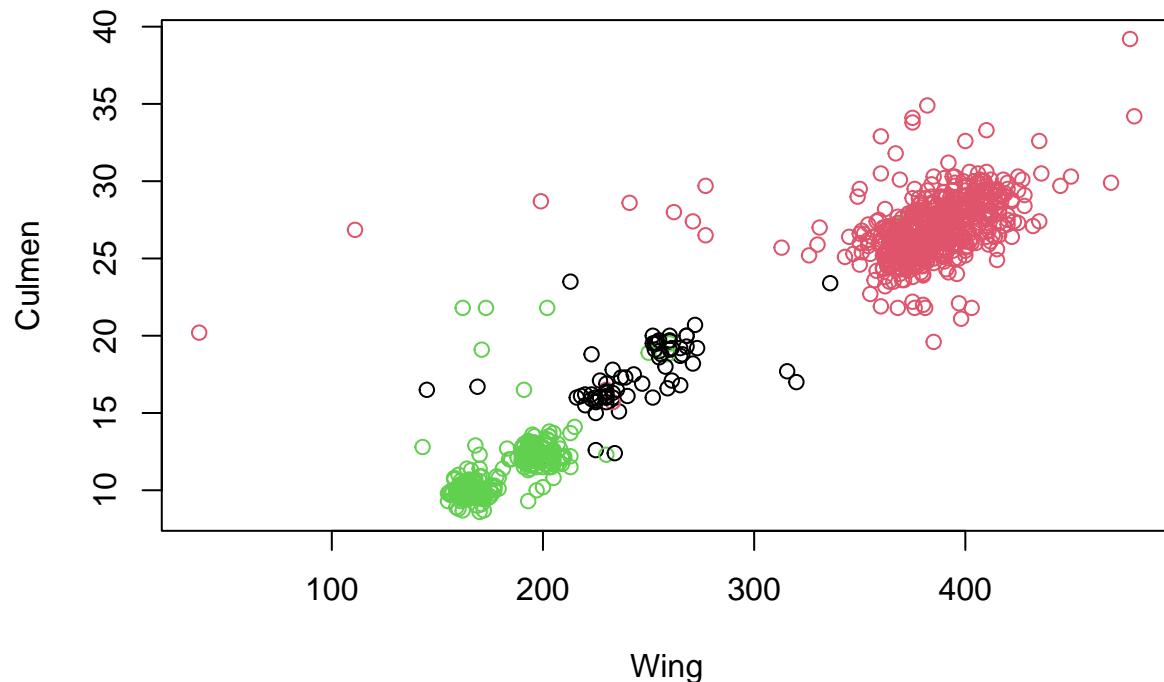
```
#Penúltima combinación:  
# Wing y Culmen  
#Hawks2[c(1,3)]  
plot(Hawks2[c(1,3)], col=Hawks2ConKIgualA3$cluster, main="Clasificación k-means con k=3")
```

### Clasificación k-means con k=3



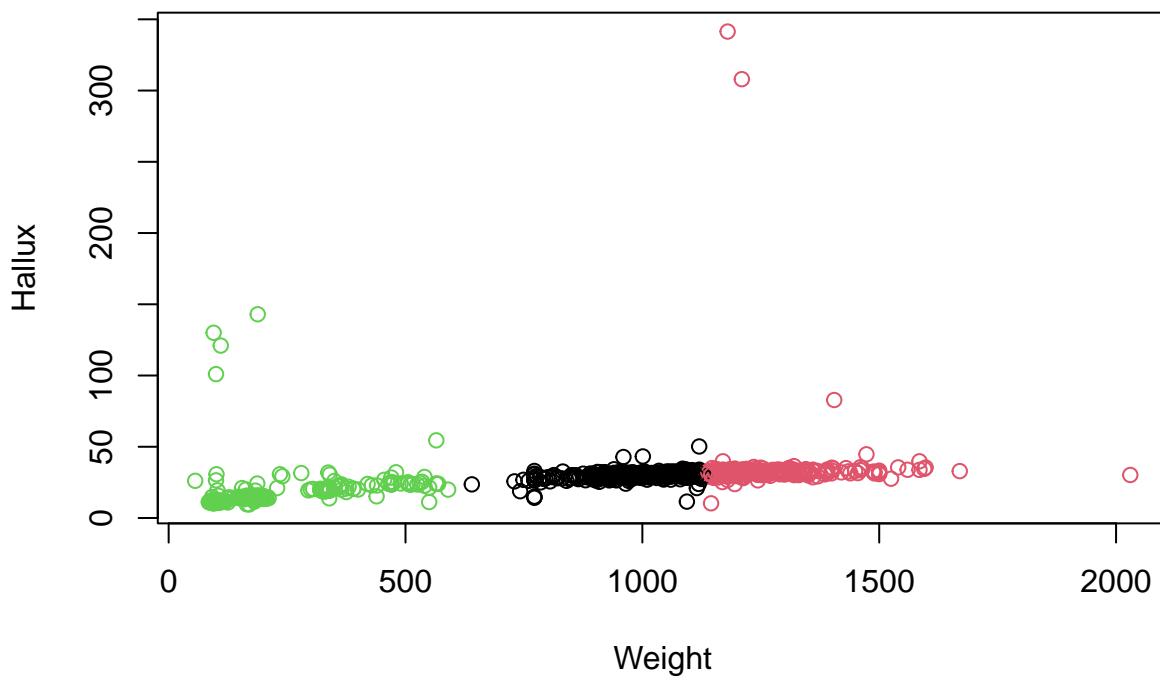
```
plot(Hawks2[c(1,3)], col=as.factor(Hawks$Species), main="Clasificación real")
```

## Clasificación real



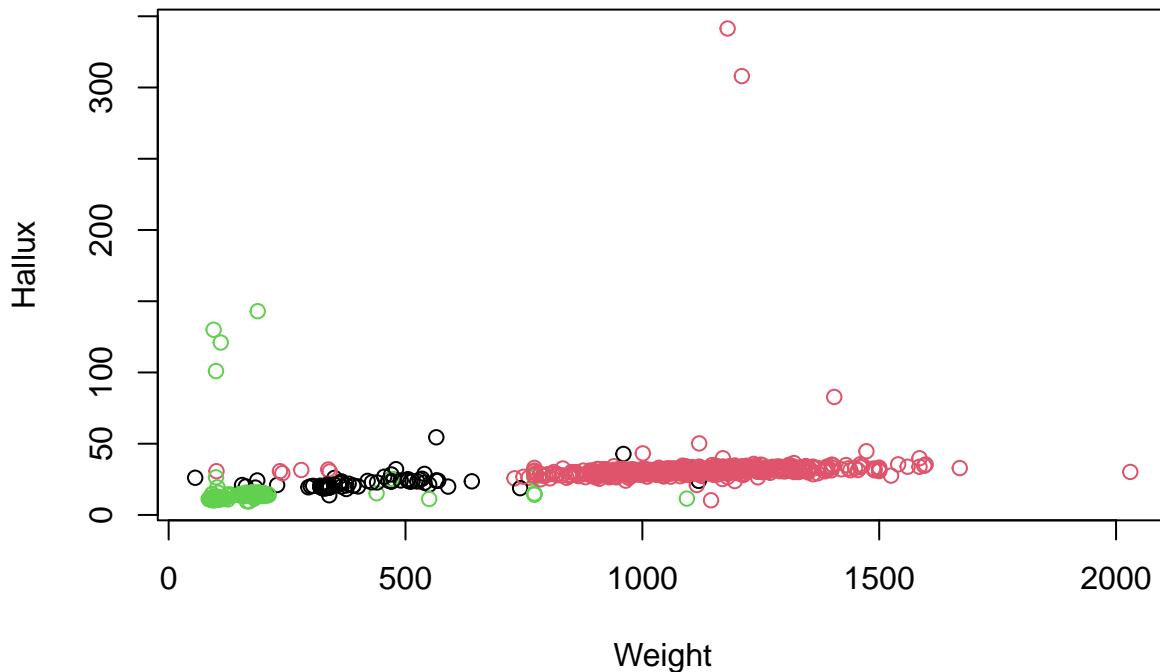
```
# Weight y Hallux  
#Hawks2[c(2,4)]  
plot(Hawks2[c(2,4)], col=Hawks2ConKIgualA3$cluster, main="Clasificación k-means con k=3")
```

## Clasificación k-means con k=3



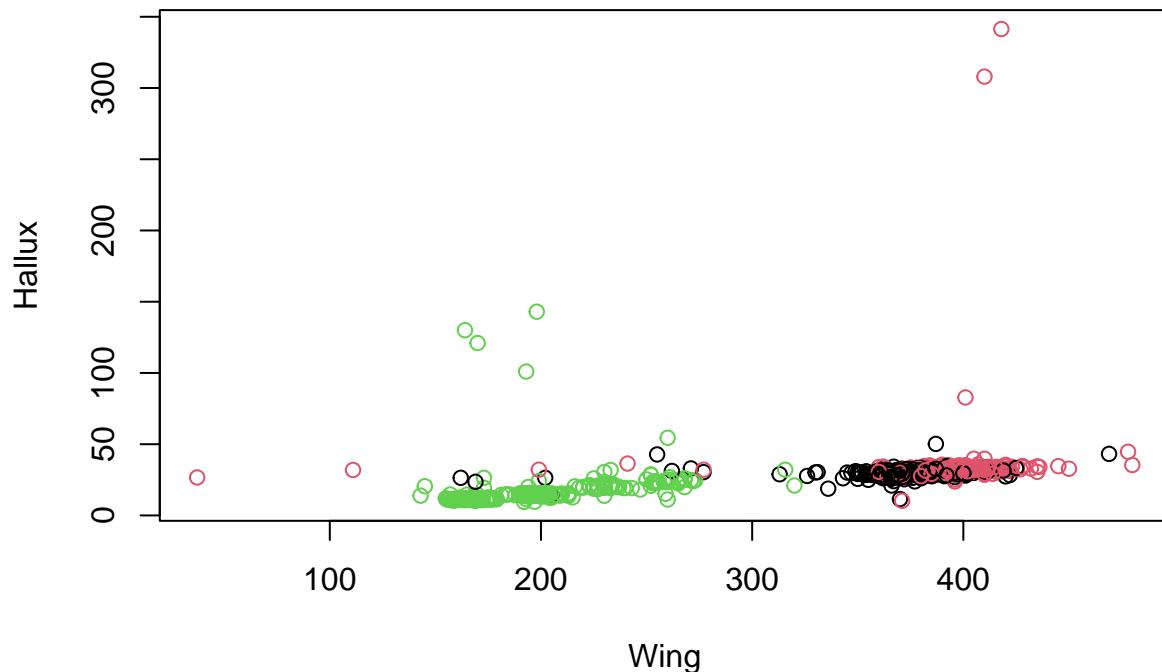
```
plot(Hawks2[c(2,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```

## Clasificación real



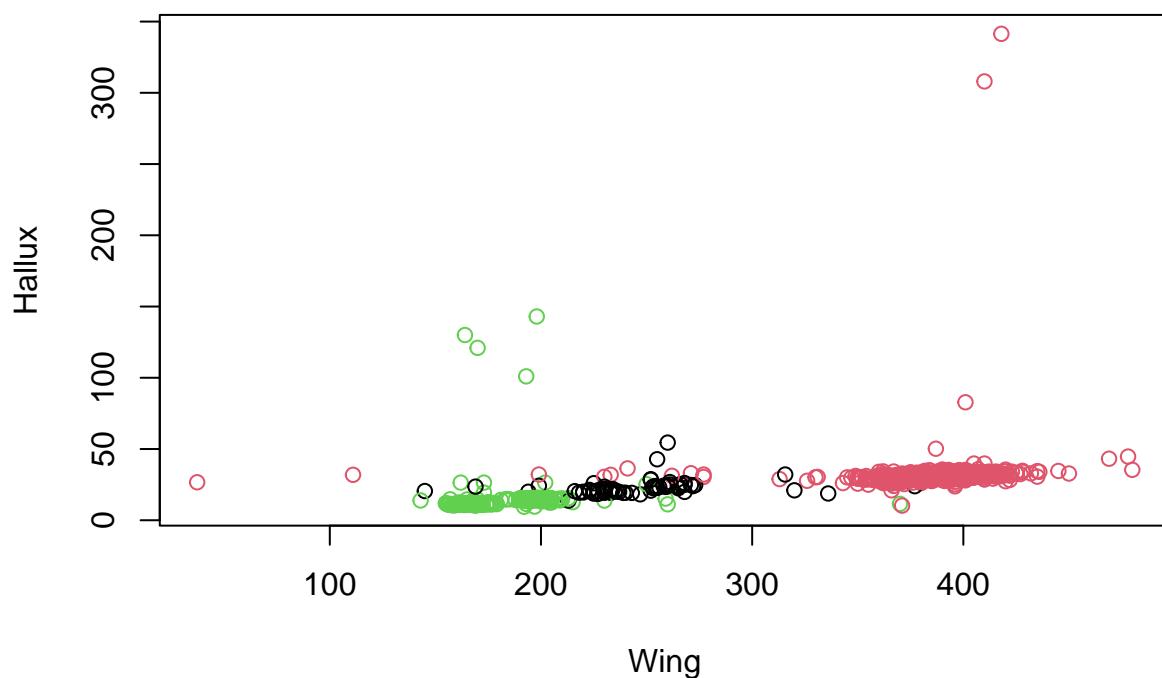
```
#Última combinación:  
# Wing y Hallux  
#Hawks2[c(1,4)]  
plot(Hawks2[c(1,4)], col=Hawks2ConKIgualA3$cluster, main="Clasificación k-means con k=3")
```

### Clasificación k-means con k=3



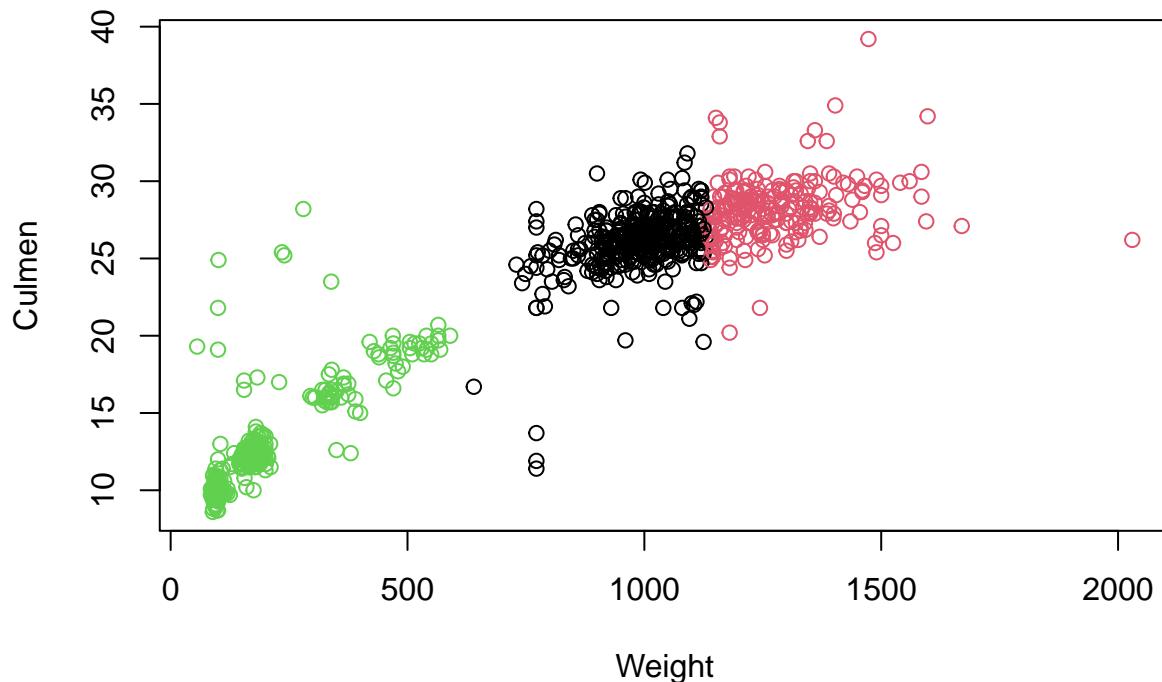
```
plot(Hawks2[c(1,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```

### Clasificación real



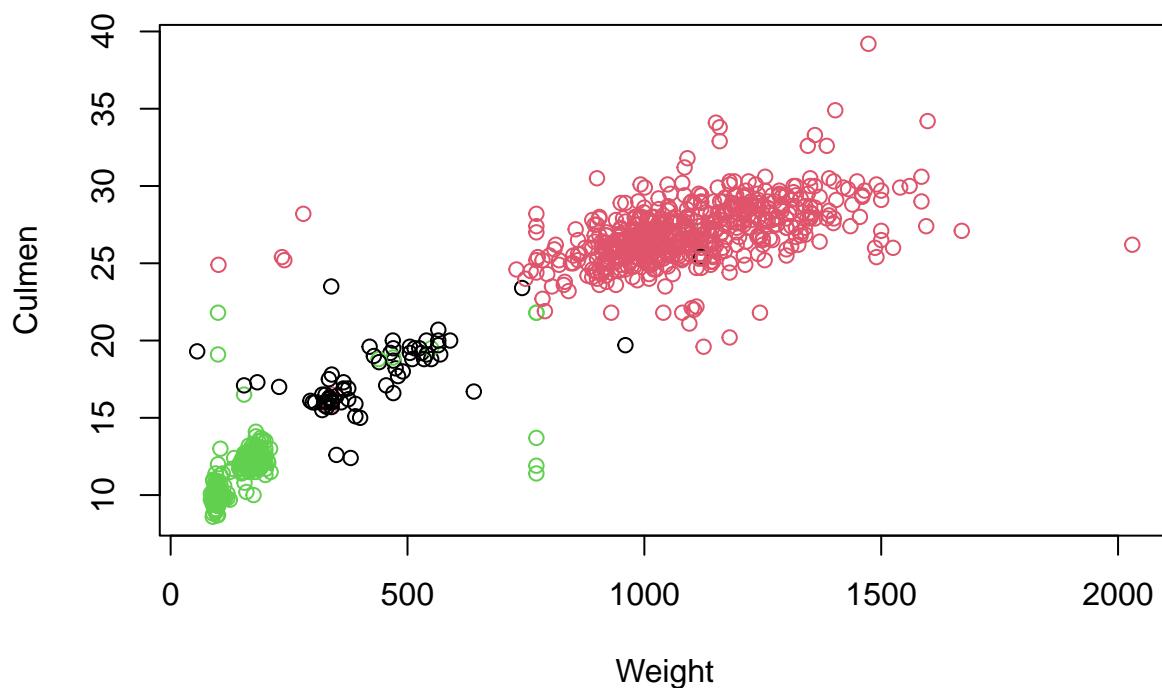
```
# Weight y Culmen  
#Hawks2[c(2,3)]  
plot(Hawks2[c(2,3)], col=Hawks2ConKIgualA3$cluster, main="Clasificación k-means con k=3")
```

### Clasificación k-means con k=3



```
plot(Hawks2[c(2,3)], col=as.factor(Hawks$Species), main="Clasificación real")
```

### Clasificación real



Si uno observa todos los gráficos correspondientes a las últimas combinaciones de atributos posibles, uno puede darse cuenta como el comportamiento descrito en los casos anteriores, se vuelve a repetir en estas últimas combinaciones. Esto es, que a grandes rasgos, el algoritmo de k-means

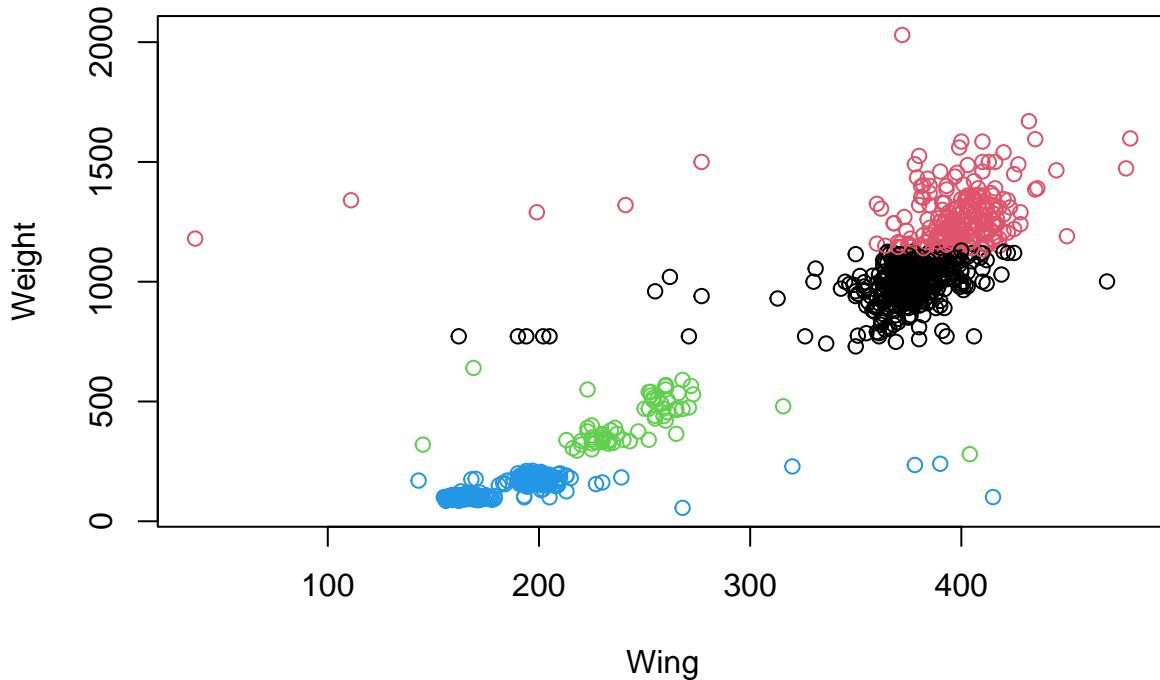
divide correctamente una parte significante de las muestras de cada uno de los clusters, en comparación con la clasificación real. No obstante, el algoritmo de k-means clasifica un mayor número de muestras del clúster negro en el clúster verde, haciendo que casi la mitad del clúster rojo original se divida entre muestras del clúster negro y clúster rojo. Tras esta comprobación me doy cuenta de que parece ser que el algoritmo no es capaz de clasificar correctamente, la especie de halcón correspondiente al clúster de muestras negras.

Por lo tanto, lo que se podría hacer sería, utilizar 4 clústers, para así identificar el clúster de muestras erróneas y que el resto de muestras se clasifiquen correctamente, no obstante esto supondría pasar de 3 clases de halcones a 4, y no tiene mucho sentido, teniendo en cuenta que ya sabemos que solo hay 3 clases de halcones y no 4. No obstante, se intenta a continuación:

```
k=4
set.seed(1)
Hawks2ConKIgualA3 <- kmeans(Hawks2, k)

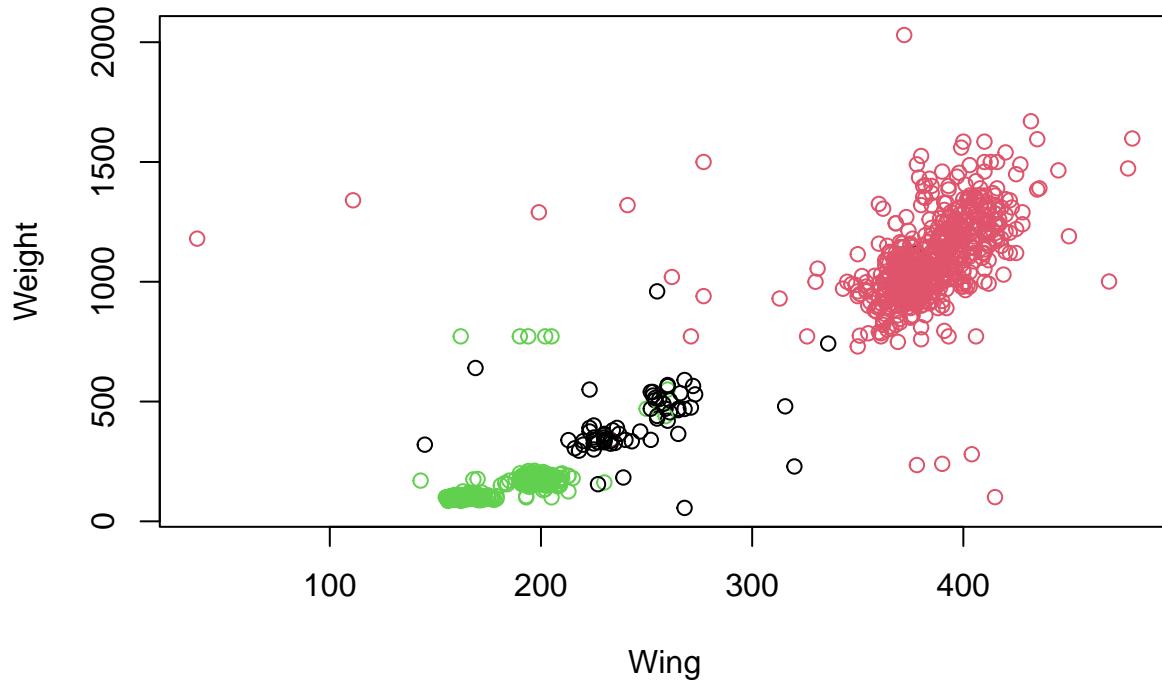
# Wing y Weight con Hawks2
plot(Hawks2[,c(1,2)], col=Hawks2ConKIgualA3$cluster, main="Clasificación k-means con k=3")
```

### Clasificación k-means con k=3



```
plot(Hawks2[,c(1,2)], col=as.factor(Hawks$Species), main="Clasificación real")
```

## Clasificación real



Parece ser que para **k=4** se obtienen mejores resultados en referencia a los clústers verde y azul, ya que a pesar de algunas clasificaciones de muestras erróneas, el clúster azul es muy cercano al clúster verde del gráfico de la clasificación real. Pero, aunque esto a simple vista parezca correcto, no lo es, ya que los clústeres deberían de coincidir, y para **k=4** se vuelve a comprobar el comportamiento de la partición del clúster rojo en dos (en el gráfico del algoritmo de los k-means), significando así.

Ahora se va a repetir el proceso de cálculo, y se va a aplicar el algoritmo de k-means con **k=3**, pero a los datos ya normalizados de las columnas **Wing** y **Culmen**, ya que ha sido la pareja de atributos que mejores resultados ha arrojado en cuanto al algoritmo de los **k-means**, véase el siguiente chunk de código:

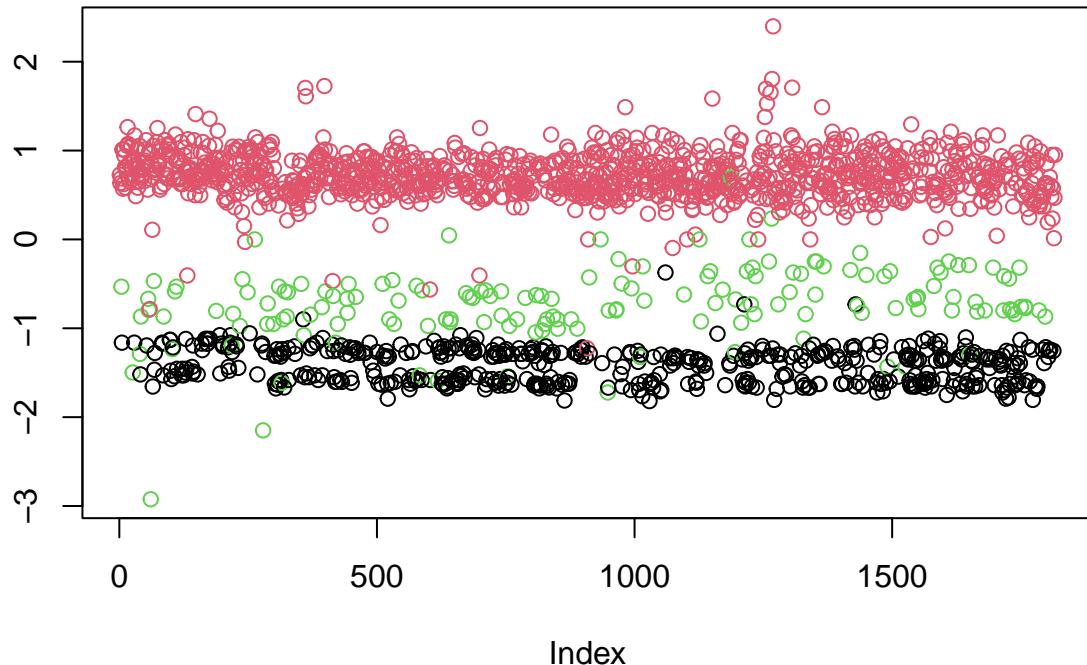
```
#Ahora lo intentamos de otra forma:
k=3
set.seed(1)

#Ahora se normalizan el resto de atributos.
Hawks2Normalizados <- scale(Hawks2)
dfHawks2Normalizado <- as.data.frame(Hawks2Normalizados)
HHawks2NormalizadosK3 <- kmeans(dfHawks2Normalizado,k)

# Wing y Culmen con Hawks2
plot(c(dfHawks2Normalizado$Wing,dfHawks2Normalizado$Culmen), col=HHawks2NormalizadosK3$cluster, main="C")
```

z(dfHawks2Normalizado\$Wing, dfHawks2Normalizado\$Culmen)

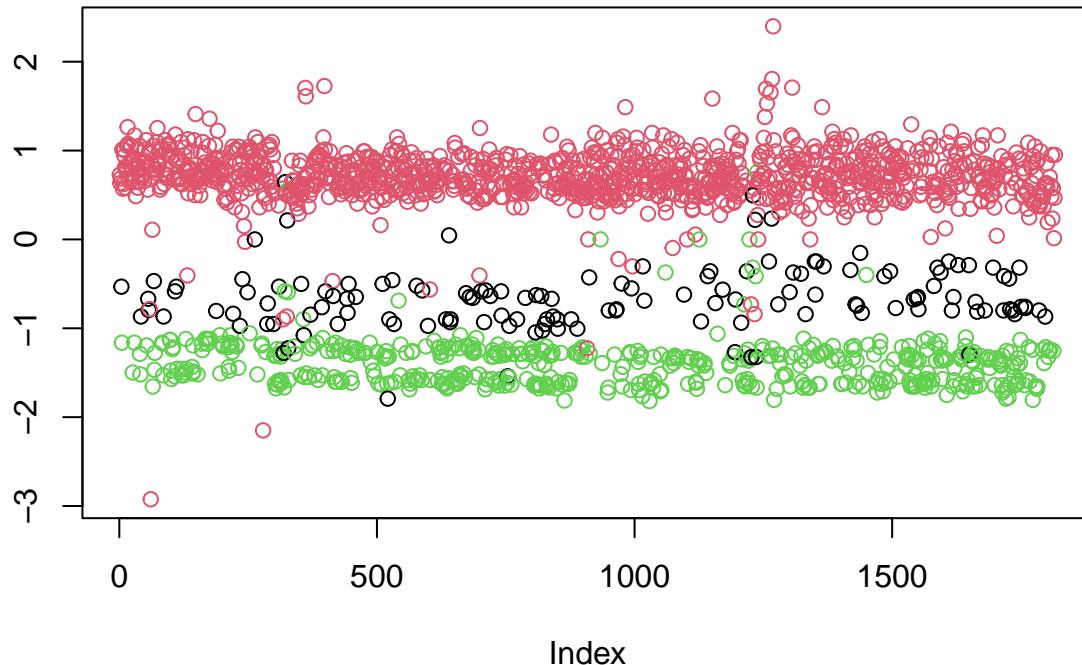
### Clasificación k-means con k=3



```
plot(c(dfHawks2Normalizado$Wing, dfHawks2Normalizado$Culmen), col=as.factor(Hawks$Species), main="Clasifi")
```

`z(dfHawks2Normalizado$Wing, dfHawks2Normalizado$Culm)`

## Clasificación real



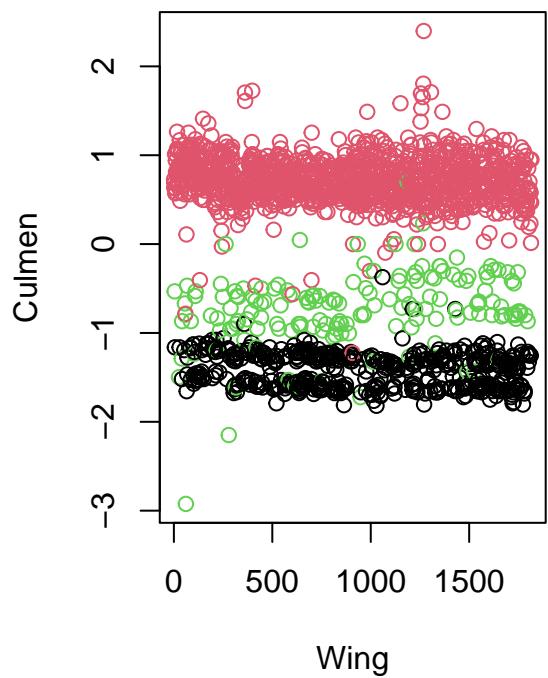
Como se ha podido comprobar al inicio de este ejercicio, existe una dependencia lineal entre muchos de los atributos, dentro de este grupo de atributos con dependencias lineales, se encuentra el par de atributos **Wing** y **Culmen**, por esta razón se ha decidido normalizar los valores de estos dos atributos, porque a parte de guardar una relación lineal entre ellos, al momento de graficar un atributo frente al otro, pudo comprobarse la gran disparidad que existe en ambas escalas, ya que *Culmen*  $\in (0, \dots, 40)[mm]$ , mientras que *Wing*  $\in (0, \dots, 450)[mm]$ . Parece ser como al normalizar los datos se obtienen mejores resultados. Como puede verse en la figura de arriba, el clúster formado por muestras rojas coincide bastante bien con el de la clasificación real, a excepción de algunas muestras, pero resulta sorprendente en comparación con los resultados obtenidos anteriormente. Luego, para los clústers negros y verdes en la gráfica de la clasificación real, se ve claramente como estos coinciden en su mayoría con los clústeres negro y verde. Por lo tanto se puede ver una clara mejora.

Como se ha visto una gran mejora respecto a los resultados anteriores, se va a implementar el algoritmo para el resto de atributos, para saber si la normalización mejora los resultados considerablemente. Véase el siguiente chunk de código.

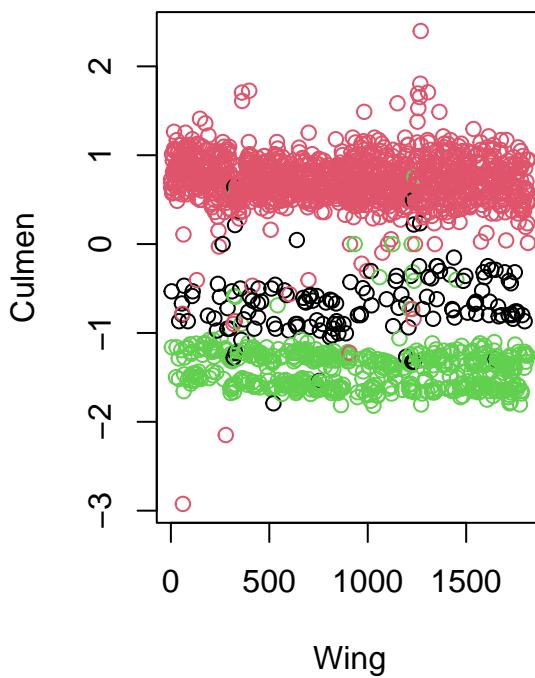
```
k=3
set.seed(1)

par(mfrow = c(1,2))
# Wing y Culmen con el dataframe de Hawks2 normalizado.
plot(c(dfHawks2Normalizado$Wing,dfHawks2Normalizado$Culmen), col=HHawks2NormalizadosK3$cluster, main="Clasificación real")
plot(c(dfHawks2Normalizado$Wing,dfHawks2Normalizado$Culmen), col=as.factor(Hawks$Species), main="Clasificación real")
```

**Clasificación k-means con k=3**

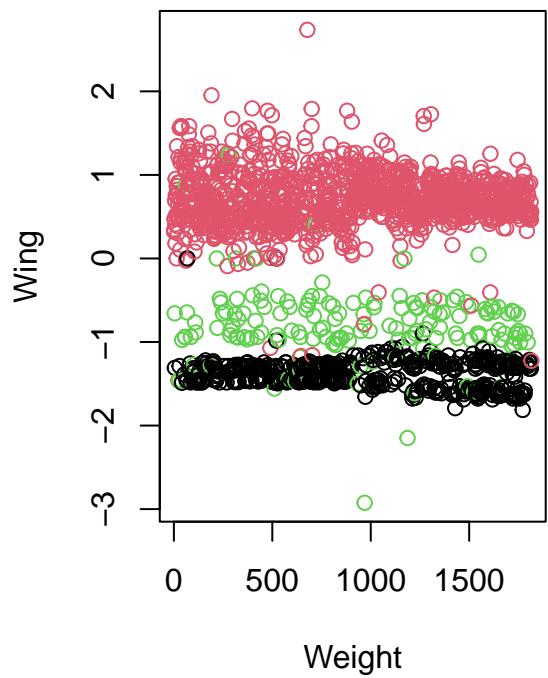


**Clasificación real**

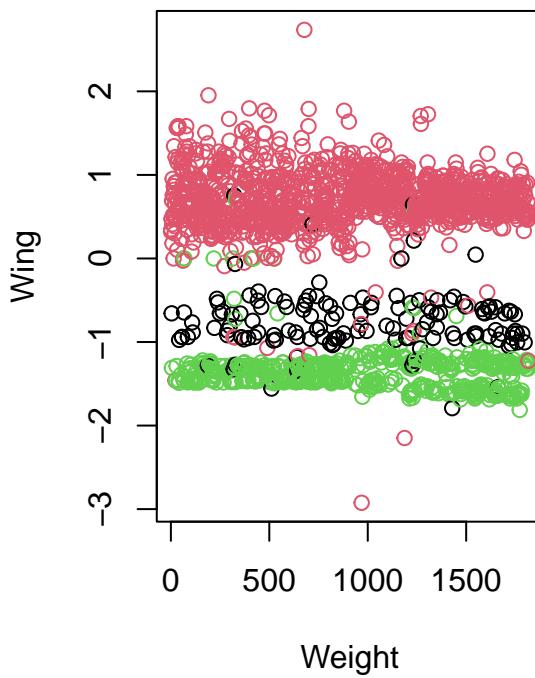


```
# Weight y Wing con el dataframe de Hawks2 normalizado.  
plot(c(dfHawks2Normalizado$Weight,dfHawks2Normalizado$Wing), col=HHawks2NormalizadosK3$cluster, main="Clasificación k-means con k=3")  
plot(c(dfHawks2Normalizado$Weight,dfHawks2Normalizado$Wing), col=as.factor(Hawks$Species), main="Clasificación real")
```

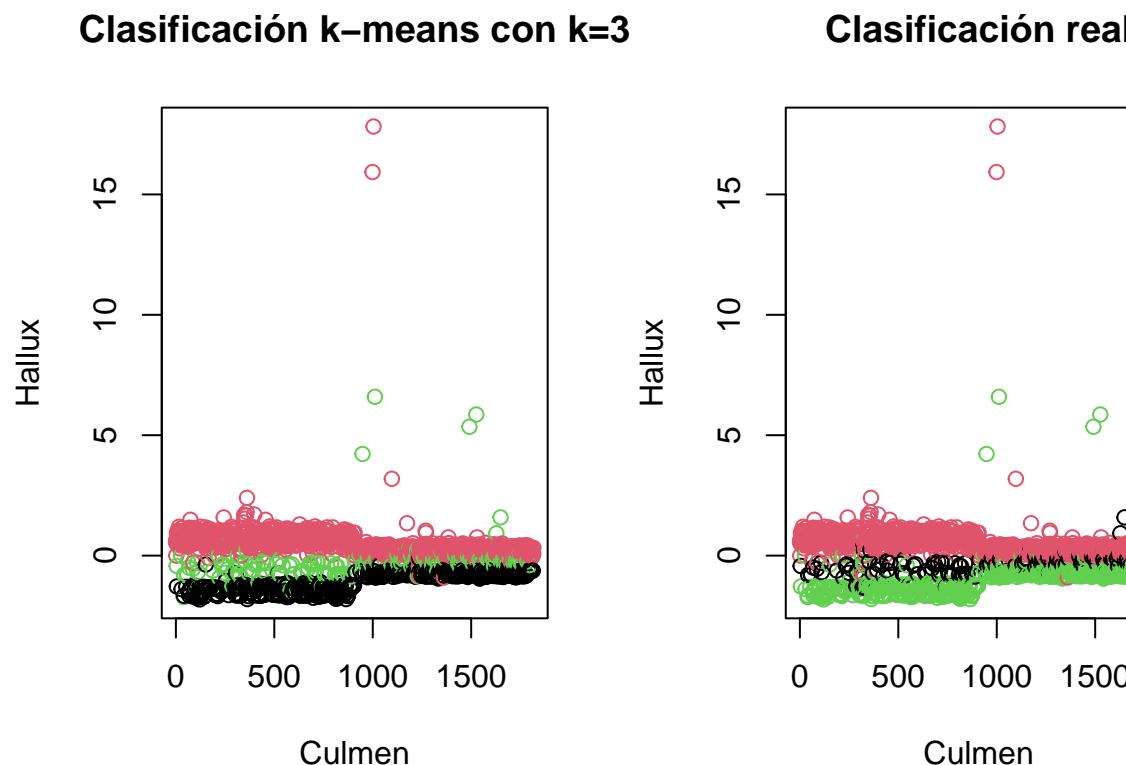
**Clasificación k-means con k=3**



**Clasificación real**

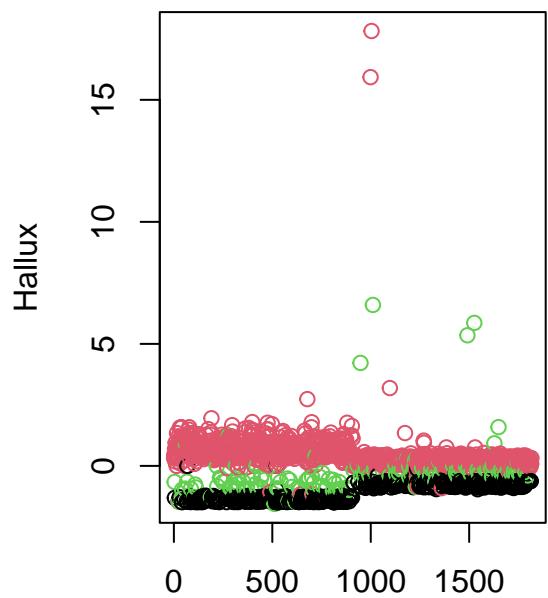


```
# Culmen y Hallux con el dataframe de Hawks2 normalizado.
plot(c(dfHawks2Normalizado$Culmen,dfHawks2Normalizado$Hallux), col=HHawks2NormalizadosK3$cluster, main="Clasificación k-means con k=3")
plot(c(dfHawks2Normalizado$Culmen,dfHawks2Normalizado$Hallux), col=as.factor(Hawks$Species), main="Clasificación real")
```

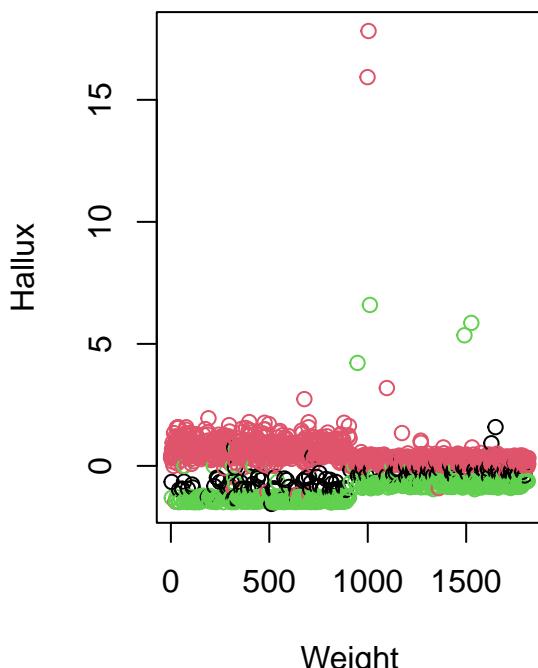


```
# Weight y Hallux con el dataframe de Hawks2 normalizado.
plot(c(dfHawks2Normalizado$Weight,dfHawks2Normalizado$Hallux), col=HHawks2NormalizadosK3$cluster, main="Clasificación k-means con k=3")
plot(c(dfHawks2Normalizado$Weight,dfHawks2Normalizado$Hallux), col=as.factor(Hawks$Species), main="Clasificación real")
```

**Clasificación k-means con k=3**

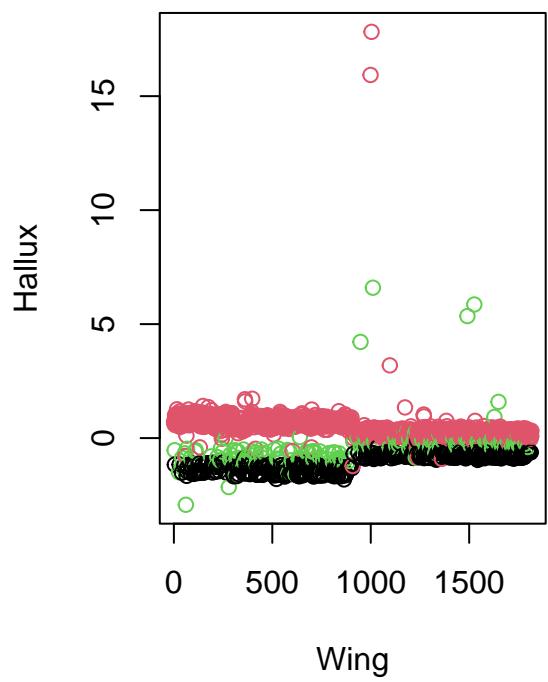


**Clasificación real**

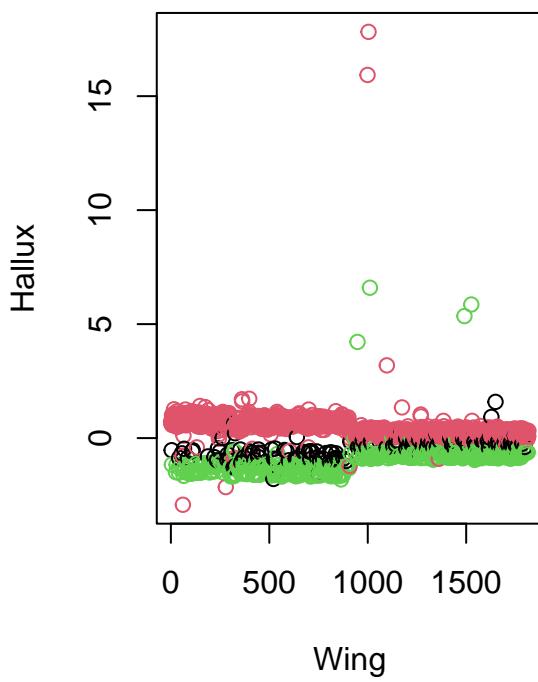


```
# Wing y Hallux con el dataframe de Hawks2 normalizado.  
plot(c(dfHawks2Normalizado$Wing,dfHawks2Normalizado$Hallux), col=HHawks2NormalizadosK3$cluster, main="Clasificación k-means con k=3")  
plot(c(dfHawks2Normalizado$Wing,dfHawks2Normalizado$Hallux), col=as.factor(Hawks$Species), main="Clasificación real")
```

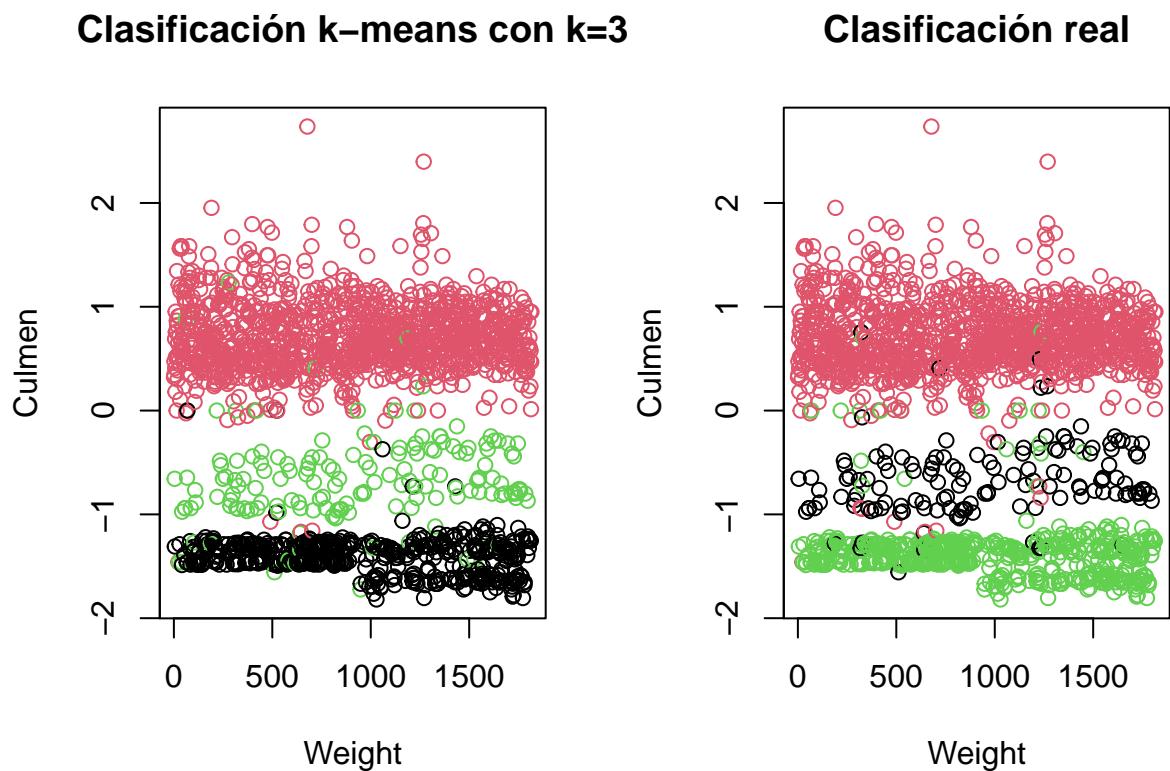
**Clasificación k-means con k=3**



**Clasificación real**



```
#Weight y Culmen con el dataframe de Hawks2 normalizado.
plot(c(dfHawks2Normalizado$Weight,dfHawks2Normalizado$Culmen), col=HHawks2NormalizadosK3$cluster, main="Clasificación k-means con k=3")
plot(c(dfHawks2Normalizado$Weight,dfHawks2Normalizado$Culmen), col=as.factor(Hawks$Species), main="Clasificación real")
```



Como se puede observar por el resultado de arriba, la normalización de los datos ha permitido una clasificación bastante buena, para todos los conjuntos de atributos. En este tipo de ejemplos puede verse como la normalización, es de gran ayuda cuando una relación entre atributos se ve “contaminada” por una diferencia de escalas. Al normalizar, todos los valores de cada uno de los atributos se ven comprendidos dentro del mismo conjunto de valores.

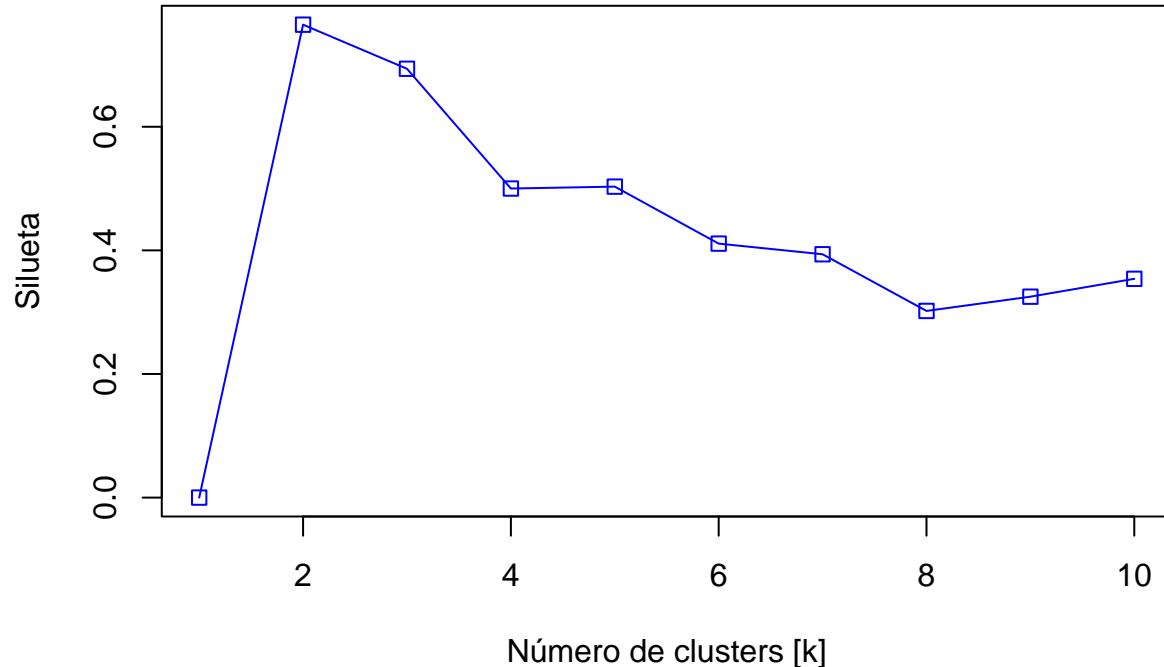
Como se ha acabado concluyendo, que la normalización ha ayudado considerablemente al algoritmo **k-means** a clasificar las muestras dentro de cada uno de los clusters, se va a comprobar que el valor óptimo de **k** sigue siendo 3, representando el valor de la silueta para un conjunto de valores  $k : k \in (2, 3, 4, 5, 6, 7, 8, 9)$  y para los 4 atributos ya NORMALIZADOS.

```
#Ahora se prueba con diferentes valores de k
d <- daisy(dfHawks2Normalizado)
#Se podría también llevar a cabo con la siguiente función:
#d <- dist(Hawks2)
resultados <- rep(0, 10) #Se inicializa un vector lleno de 0 para luego poblarlo con los resultados de k
for (i in c(2,3,4,5,6,7,8,9,10))
{
  fit           <- kmeans(dfHawks2Normalizado, i)
  y_cluster     <- fit$cluster
  sk            <- silhouette(y_cluster, d)
  resultados[i] <- mean(sk[,3])
}
cat(resultados)
```

```
## 0 0.7651524 0.6938324 0.5000752 0.5031269 0.4109098 0.3937015 0.302004 0.3250632 0.3539613
```

```
#Ahora se representan los valores que se han obtenido arriba:
plot(1:10,resultados,type="o",col="blue",pch=0,xlab="Número de clusters [k]",ylab="Silueta", main = "Primer gráfico")
```

## Primer gráfico



```
p <- recordPlot()
```

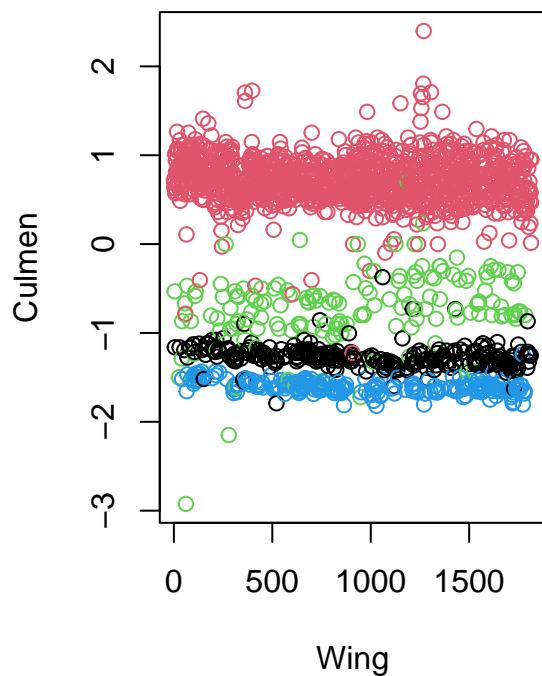
Curiosamente, con los datos normalizados, el valor óptimo de  $k$  es  $k=4$ , pues es en  $k=4$  dónde se observa el famoso “codo” visto en teoría. Además parece observarse muchas más colinas y cambios drásticos de pendiente para valores de  $k$  mayores. A continuación se va a ver que pasa si  $k=4$  y se aplica el algoritmo k-means. Véase el siguiente chunk de código:

```
#Ahora lo intentamos de otra forma:
set.seed(1)

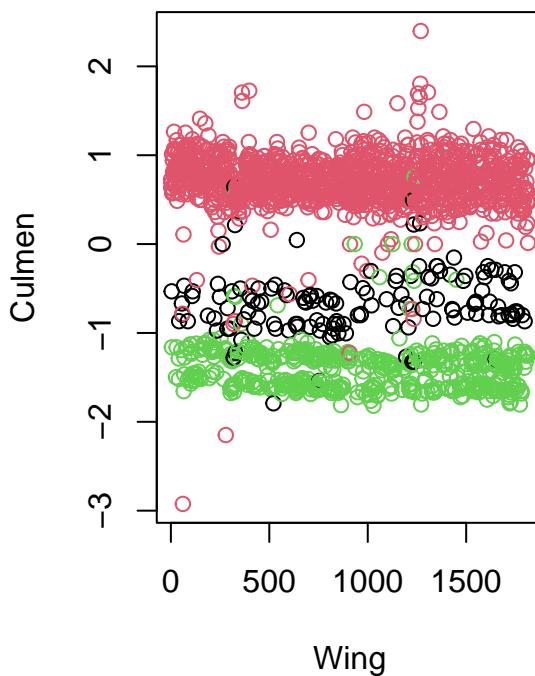
#Ahora se normalizan el resto de atributos.
HHawks2NormalizadosK4 <- kmeans(dfHawks2Normalizado,4)

#Wing y Culmen con el dataframe de Hawks2 normalizado y con k=4.
par(mfrow = c(1,2))
plot(c(dfHawks2Normalizado$Wing,dfHawks2Normalizado$Culmen), col=HHawks2NormalizadosK4$cluster, main="Clasificación")
plot(c(dfHawks2Normalizado$Wing,dfHawks2Normalizado$Culmen), col=as.factor(Hawks$Species), main="Clasificación")
```

## Clasificación k-means con k=4



## Clasificación real



Comparando los resultados, no se observa una mejoría notable en comparación con los resultados para  $k=4$  y con los datos normalizados. Además de esto, sabiendo que hay 3 clases de halcones, no tiene mucho sentido tener 4 clústeres, y mucho menos sabiendo que esto no mejora los resultados de los clústeres existentes, pues si las muestras que se habían clasificado erróneamente para  $k=3$ , fuesen clasificadas dentro del 4 clúster, sería algo positivo. No obstante, lo que se ve, es que la mitad del clúster verde en la clasificación real, queda particionado en dos (azul y negro) por el **k-means**. Por lo tanto, se concluye que para  $k=4$  no se consiguen mejores resultados en cuanto a clasificación de muestras, en comparación con el caso anterior, donde los datos ya estaban normalizados y  $k=3$ .

## Ejercicio 2

Con el juego de datos proporcionado realiza un estudio aplicando DBSCAN y OPTICS, similar al del ejemplo 2

### Respuesta 2

Antes de explicar nada, quería comentar que, este primer análisis de los algoritmos **DBSCAN** y **OPTICS** se realizó antes de la última parte del ejercicio anterior. Esto significa que primero se ha analizado el comportamiento de los dos algoritmos sin normalizar los datos, y aunque no fuese algo muy correcto, se ah creído oportuno mantener las explicaciones de los resultados obtenidos. No obstante, a mitad del ejercicio se repite el análisis de los dos agoritmos, pero con los datos ya normalizados.

Como en este ejercicio se va a trabajar con los algoritmos **DBSCAN** y **OPTICS**, se descargan a continuación los paquetes necesarios:

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

A continuación se van a explicar muy brevemente estos dos algoritmos, se empieza por el algoritmo

## DBSCAN.

El algoritmo **DBSCAN** precisa de dos parámetros;  $\varepsilon$  que determina el radio máximo de cercanía entre dos puntos, y el valor **minPTS** que se refiere al mínimo número de puntos que rodean a un punto en concreto en un radio  $\varepsilon$ . Así pues, este algoritmo irá contruyendo esferas con radio  $\varepsilon$  con **minPTS** puntos. La dinámica de este algoritmo implica dos variables más;  $q_{alcanzable}$  y  $q_{nucleo}$ . El  $q_{nucleo}$  es un punto  $p$  cualquiera, que tiene **minPTS** a una distancia  $\varepsilon$ . Por último,  $q_{alcanzable}$  hace referencia a un punto  $p$  cualquiera, al cual se puede acceder por medio de una senda de  $q_{nucleo}$ . Cualquier punto no alcanzable se denomina outlier.

Una de las ventajas de este algoritmo reside en el potencial que tiene en la búsqueda de valores extremos y es capaz de lidiar con clústeres de distintas formas geométricas. Además este algoritmo no necesita conocer previamente el número de clústeres. Pero una de las semejanzas que guarda con el algoritmo de **k-means**, es que hay que acertar a la hora de darles valor a las variables  $\varepsilon$  y **minPTS** y esto requiere de experiencia en la materia.

El algoritmo **OPTICS** resuelve el problema de las variables iniciales, visto en el anterior algoritmo con  $\varepsilon$  y **minPTS** y en el algoritmo **k-means** con el número de clústeres **k**. Esto no significa que el programador/científico de datos, no tenga que especificar ninguna variable, pues en este algoritmo hay que especificar un radio  $\varepsilon_{OPTICS}$ , pero a diferencia del anterior algoritmo, este parámetro no influirá en cuanto a dinámica fundamental del algoritmo, como hacían los parámetros **k**,  $\varepsilon_{DBSCAN}$  y **minPTS**, sino que aumentará o disminuirá la complejidad de esos cálculos. Es por esto, que el algoritmo **OPTICS** no genera clústeres, sino que lleva a cabo una ordenación de puntos según la **distancia de alcanzabilidad** ( $d_{reach}$ ). Esta distancia se calcula de la siguiente manera;  $d_{reach} := \min(d_{nucleo}, d(p, q))$  dónde por teoría se sabe que  $d : p, q \rightarrow \mathbb{R} \quad \forall p, q \in \mathbb{R}^2$ . La dinámica de este algoritmo, consiste principalmente en asignar a cada punto del juego de datos, una  $d_{reach}$ .

A diferencia del algoritmo **DBSCAN**, **OPTICS** permite que el valor de densidad sea variable dentro del juego de datos, esto es lógico, pues calibrando el parámetro  $\varepsilon_{OPTICS}$  se fija el límite en el punto que se deseé.

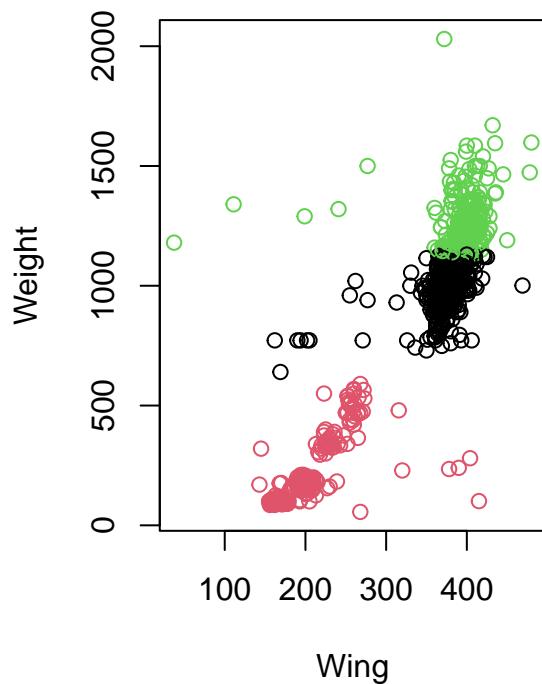
Habiendo mencionado tanto la dinámica, como los parámetros esenciales de cada uno de estos algoritmos, se procede a su aplicación en nuestro juego de datos.

Como se ha visto en teoría y en el ejemplo guiado 3.1, una de las primeras tareas a realizar tiene que ver con el **ordenamiento de las observaciones** a fin de convertir a los vecinos más cercanos en vecinos de ordenamiento.

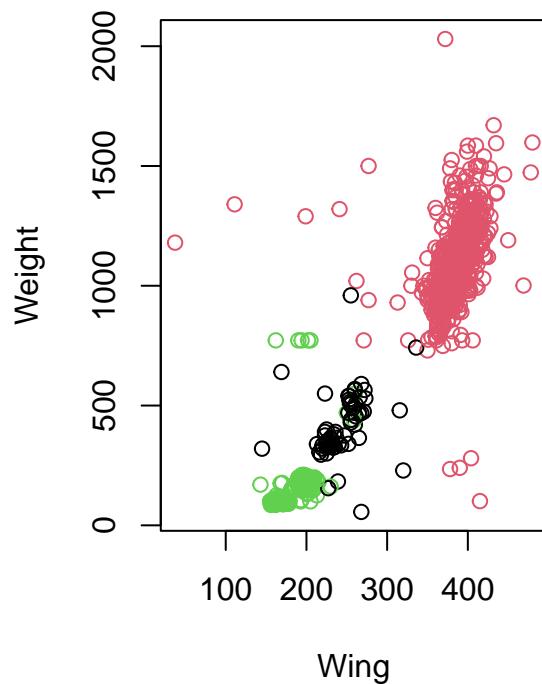
Primero se grafican los 4 atributos en parejas de dos (AÚN NO ESTÁN NORMALIZADOS), estos son los mismos atributos que los del ejercicio anterior y son con los que se va a trabajar, i.e., **Wing**, **Weight**, **Culmen**, **Hallux**.

```
Hawks2ConKIgualA3 <- kmeans(Hawks2, 3)
#Wing y Weight
par(mfrow = c(1,2))
plot(Hawks2[c(1,2)], col=Hawks2ConKIgualA3$cluster, main="Clasificación k-means con k=3")
plot(Hawks2[c(1,2)], col=as.factor(Hawks$Species), main="Clasificación real")
```

**Clasificación k-means con k=3**

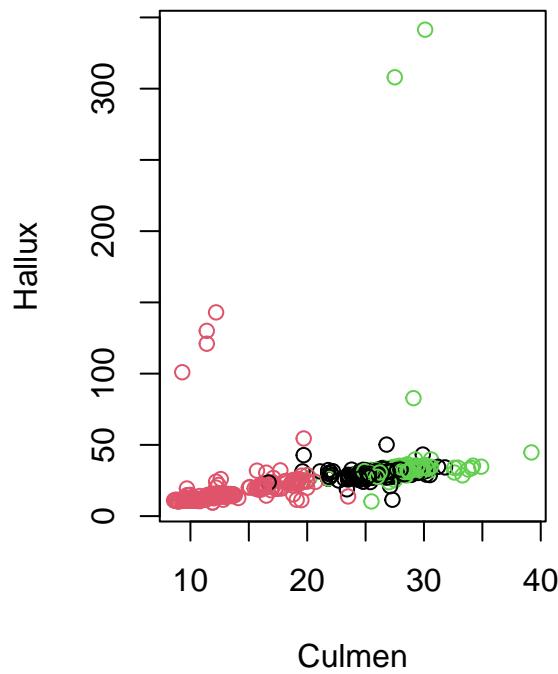


**Clasificación real**

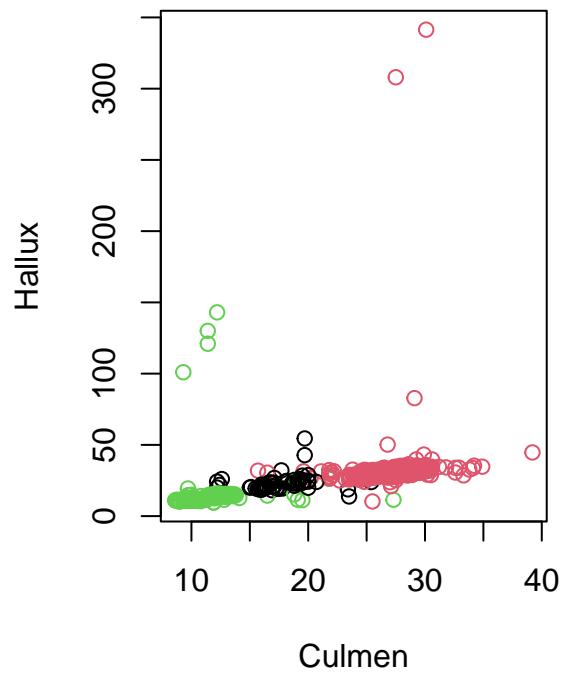


```
#Culmen y Hallux  
plot(Hawks2[c(3,4)], col=Hawks2ConIGualA3$cluster, main="Clasificación k-means con k=3")  
plot(Hawks2[c(3,4)], col=as.factor(Hawks$Species), main="Clasificación real")
```

**Clasificación k-means con k=3**



**Clasificación real**



Ahora se lleva a cabo el ordenamiento de las observaciones:

```

print("Observación para el par Wing Culmen")

## [1] "Observación para el par Wing Culmen"
observaciones <- optics(Hawks2[c("Wing", "Culmen")], minPts=5) #El mejor resultado de epsilon que se obtiene

#Lo que hago a partir de aqui es simplemente crear un codigo que tenga en cuenta todas las combinaciones
resultados_observaciones <- list()
combinaciones <- combn(c("Wing", "Weight", "Culmen", "Hallux"), 2)
print(combinaciones[, ])

##      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,] "Wing"   "Wing"   "Wing"   "Weight" "Weight" "Culmen"
## [2,] "Weight" "Culmen" "Hallux" "Culmen" "Hallux" "Hallux"

for (i in 1:ncol(combinaciones)) {
  etiqueta1 <- combinaciones[1, i]
  etiqueta2 <- combinaciones[2, i]
  cat("\n\nObservación para el par de atributos: ", etiqueta1, etiqueta2)
  # Utilizar [[]] para extraer las columnas del dataframe
  observaciones <- optics(Hawks2[, c(etiqueta1, etiqueta2)], minPts = 5)
  resultados_observaciones[[i]] <- observaciones
  cat("\n")
  print(resultados_observaciones[[i]])
}

##
## Observación para el par de atributos: Wing Weight
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 446.619524875481, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## Observación para el par de atributos: Wing Culmen
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 118.303212128834, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## Observación para el par de atributos: Wing Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 301.706082139555, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## Observación para el par de atributos: Weight Culmen
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 445.021752277347, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##

```

```

## 
## Observación para el par de atributos: Weight Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 445.103538516602, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## Observación para el par de atributos: Culmen Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 241.29815581558, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi

```

Cabe destacar que este análisis se ha llevado a cabo de dos en dos, es decir, para pares de atributos. Se observa como para un menor número de puntos, epsilon se reduce, porque no queremos agrupar puntos que podrían considerarse ruido o formar clústeres más amplios que incluyan subestructuras. No obstante, un epsilon más pequeño requiere que los puntos estén más cerca para que se considere que están conectados, lo que podría resultar en clústeres más pequeños y más densos. No obstante las magnitudes que se han obtenido para epsilon son desproporcionadas, pues son demasiado grandes. Este problema puede deberse al hecho de que los datos no están normalizados, por ello, en el siguiente chunk de código, se va a volver a aplicar el algoritmo pero estavez para los 4 atributos normalizados.

Para el caso que se ha estudiado, es decir, para **minPTS = 5** se observa como el algoritmo **OPTICS**, para el par de atributos **Wing** y **Culmen** arroja el menor valor de  $\varepsilon_{OPTICS}$  con un valor de  $\varepsilon_{OPTICS} = 118.3$ . Este experimento puede repetirse para valores distintos a **minPTS = 5**, véase el resultado para  $minPTS \in [10, 20]$ . Pero, como se ha dicho antes, primero se vuelve a aplicar el algoritmo OPTICS, pero para el conjunto de datos normalizados:

```

for (i in 1:ncol(combinaciones)) {
  etiqueta1 <- combinaciones[1, i]
  etiqueta2 <- combinaciones[2, i]
  cat("\n\nObservación para el par de atributos: ", etiqueta1, etiqueta2)
  # Utilizar [[]] para extraer las columnas del dataframe
  observaciones <- optics(dfHawks2Normalizado[, c(etiqueta1, etiqueta2)], minPts = 5)
  resultados_observaciones[[i]] <- observaciones
  cat("\n")
  print(resultados_observaciones[[i]])
}

## 
## Observación para el par de atributos: Wing Weight
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 1.83358167368754, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## Observación para el par de atributos: Wing Culmen
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 1.44307386131567, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##

```

```

## 
## Observación para el par de atributos: Wing Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 13.8041256425086, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## Observación para el par de atributos: Weight Culmen
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 1.14217003461741, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## Observación para el par de atributos: Weight Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 13.8017783783724, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## Observación para el par de atributos: Culmen Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 13.8992186446814, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi

```

Como se puede observar, ahora los resultados tienen más sentido, porque epsilon no tiene 3 cifras, sino 1. El resultado de epsilon más bajo, se da para el mismo par de atributos que en el anterior caso, es decir, para: Wing-Culmen, donde  $\varepsilon = 1.1421$ . Esto tiene sentido porque lo único que se ha hecho ha sido normalizar los datos. Ahora nos centramos en el resultado de  $\varepsilon$ , se sabe por teoría que  $\varepsilon$  está relacionado con la distancia de alcance entre puntos. La pregunta ahora es; ¿que significa que  $\varepsilon$  sea grande o pequeño? En el caso de que  $\varepsilon$  sea grande, significaría, que los puntos que se considerarían como que están “lejos” quedarían conectados, ahora bien, si  $\varepsilon$  es pequeño, entonces significa que los puntos quedarían conectados solo en el caso de estar muy cerca unos de otros.

Ahora que se ha visto como la normalización ha dado los resultados que debía de dar, a la hora de aplicar el algoritmo OPTICS. Se procede a repetir el mismo proceso con los datos ya normalizados, pero esta vez para diferentes valores de **minPTS**, véase el siguiente chunk de código:

```

resultados_observaciones2 <- list()

for (i in 1:ncol(combinaciones)) {
  etiqueta1 <- combinaciones[1, i]
  etiqueta2 <- combinaciones[2, i]
  cat("\n\nObservación para el par de atributos: ", etiqueta1, etiqueta2)
  # Utilizar [[]] para extraer las columnas del dataframe
  for (i in c(5:20)){
    observaciones <- optics(dfHawks2Normalizado[, c(etiqueta1, etiqueta2)], minPts = i)
    resultados_observaciones2 <- observaciones
    cat("\n")
    print(resultados_observaciones2)
  }
}

```

```

}

## 
## Observación para el par de atributos: Wing Weight
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 1.83358167368754, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 6, eps = 1.87045255428053, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 7, eps = 1.94481918562941, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.97290629570105, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 9, eps = 2.16174752945366, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 10, eps = 2.1864427420409, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 11, eps = 2.33673287899132, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 12, eps = 2.38427197539077, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 13, eps = 2.38623675622616, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 14, eps = 2.46179634869535, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi

```

```

## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 15, eps = 2.57178105792203, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 16, eps = 2.58168253589853, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 17, eps = 2.58925184033053, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 18, eps = 2.59891614918726, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 19, eps = 2.5999262085549, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 20, eps = 2.60148848228469, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## Observación para el par de atributos: Wing Culmen
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 1.44307386131567, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 6, eps = 1.46573317255239, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 7, eps = 1.50815073163072, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.69370507620783, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.

```

```

## Parameters: minPts = 9, eps = 1.70261729378344, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 10, eps = 1.74465435350789, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 11, eps = 1.76925569237451, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 12, eps = 1.80110103191792, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 13, eps = 1.80848391380764, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 14, eps = 1.81315780160123, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 15, eps = 1.82303012818048, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 16, eps = 1.82591308307108, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 17, eps = 1.84734372729518, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 18, eps = 1.84817135888205, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 19, eps = 1.85489621416177, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##

```

```

## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 20, eps = 1.87465585656625, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
##
## Observación para el par de atributos: Wing Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 13.8041256425086, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 6, eps = 14.6311395175254, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 7, eps = 16.3156830704118, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 16.4775806605301, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 9, eps = 16.797343544843, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 10, eps = 16.8788819737327, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 11, eps = 16.9795142879257, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 12, eps = 17.0629414658345, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 13, eps = 17.0632807773838, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 14, eps = 17.2894167618564, eps_cl = NA, xi = NA

```

```

## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 15, eps = 17.3028442745718, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 16, eps = 17.3060158203821, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 17, eps = 17.3182380673736, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 18, eps = 17.319307835665, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 19, eps = 17.3238998413213, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 20, eps = 17.3242484697602, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## Observación para el par de atributos: Weight Culmen
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 1.14217003461741, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 6, eps = 1.14861839273948, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 7, eps = 1.15352627947916, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.15943872924627, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi

```

```

## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 9, eps = 1.21000028610239, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 10, eps = 1.21252054036257, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 11, eps = 1.22767942780772, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 12, eps = 1.2284904493138, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 13, eps = 1.24963327464822, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 14, eps = 1.2750282029928, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 15, eps = 1.29167516221915, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 16, eps = 1.30468849134184, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 17, eps = 1.31133548946268, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 18, eps = 1.31589005778337, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 19, eps = 1.32018769970661, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi

```

```

##           eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 20, eps = 1.33447135682691, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## 
## 
## Observación para el par de atributos: Weight Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 13.8017783783724, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 6, eps = 14.6382333317241, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 7, eps = 16.2861238063336, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 16.4748813785044, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 9, eps = 16.797615473133, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 10, eps = 16.874874845712, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 11, eps = 16.8997883691818, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 12, eps = 17.0627485073415, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 13, eps = 17.0854587746709, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##

```

```

## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 14, eps = 17.257773132839, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 15, eps = 17.2894447300207, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 16, eps = 17.3110618938063, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 17, eps = 17.3179871882752, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 18, eps = 17.3237082860862, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 19, eps = 17.3239278158699, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 20, eps = 17.3252828372485, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## Observación para el par de atributos: Culmen Hallux
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 5, eps = 13.8992186446814, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 6, eps = 14.6306994385124, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 7, eps = 16.2942570075143, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 16.4806408826088, eps_cl = NA, xi = NA

```

```

## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 9, eps = 16.832307912179, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 10, eps = 16.8704053778441, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 11, eps = 16.9537076505533, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 12, eps = 17.0628737877846, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 13, eps = 17.0631367663321, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 14, eps = 17.2563247213559, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 15, eps = 17.2890308594949, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 16, eps = 17.3006969837502, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 17, eps = 17.3060250257765, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 18, eps = 17.3192974788585, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.

```

```

## Parameters: minPts = 19, eps = 17.3199719143527, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                     eps_cl, xi
##
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 20, eps = 17.3208795571116, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                     eps_cl, xi

```

Como se puede comprobar, el mejor valor de epsilon que se ha encontrado, es para el par; Weight y Culmen, así como para el par; Wing y Culmen, y además, puede notarse como según aumenta el número de **minPTS** el valor de  $\varepsilon_{OPTICS}$  también lo hace. Esto es normal, ya que por teoría, se sabe que **minPTS** se refiere al número mínimo de puntos que tiene que haber dentro de un radio  $\varepsilon_{OPTICS}$  para que un punto sea alcanzable por otro. Es por esto, que al aumentar el valor **minPTS** se está aumentando el límite mínimo de densidad que se requiere para considerar a un punto como parte del clúster, y al aumentar este límite, puede que se requiera aumentar el radio  $\varepsilon_{OPTICS}$  afín de englobar áreas más extensas de densidad en las muestras.

Ahora, con el fin de estudiar el resultado a partir de algo más intuitivo, se ha representado el **diagrama de alcanzabilidad** para las “observaciones” del mejor par de atributos, en base al resultado que han arrojado de  $\varepsilon_{OPTICS}$  y para un valor de **minPTS** más o menos razonable y realista, en este caso **minPTS = 8**. Véase a continuación el **diagrama de alcanzabilidad** de las observaciones realizadas por el algoritmo **OPTICS** para los dos pares de atributos **Wing-Culmen** y **Weight-Culmen**:

```

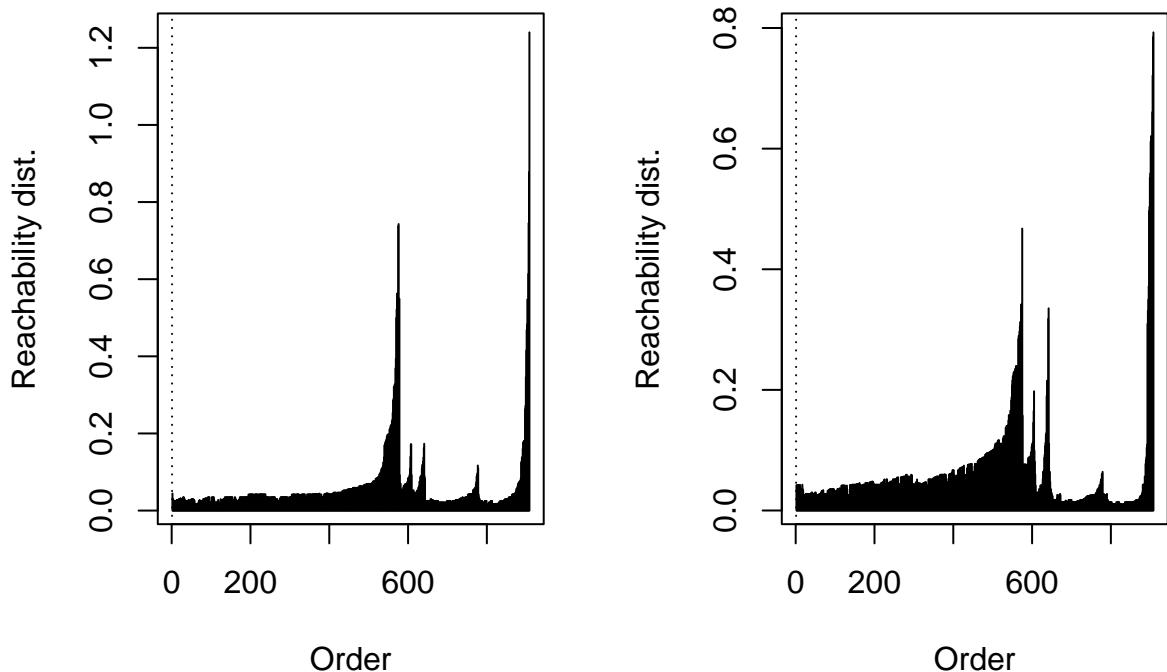
par(mfrow = c(1,2))

#Par Wing-Culmen
observacionesWingCulmen <- optics(dfHawks2Normalizado[c("Wing","Culmen")],minPts = 8)
plot(observacionesWingCulmen, main = 'Diagrama de alcance: Wing-Culmen')

#Par Weight-Culmen
observaciones <- optics(dfHawks2Normalizado[c("Weight","Culmen")],minPts = 8)
plot(observaciones, main = 'Diagrama de alcance: Weight-Culmen')

```

## Diagrama de alcance: Wing–Culm Diagrama de alcance: Weight–Culr



Como se puede ver, hay dos picos considerables, pero para los dos pares de atributos se observan 5 picos. Para el par Weight-Culmen, a parte de los dos picos más grandes, se observa como hay otros dos picos a la derecha del segundo pico más grande, que son más altos que para el par Wing-Culmen. Cabe destacar además, que los picos hacen referencia a las muestras que se encuentran entre los clústeres, esto quiere decir que hay muestras “inter-clúster” a una distancia de alcanzabilidad de aproximadamente 0.8 y 1.3, estas muestras son consideradas como outliers, y por tanto en un proyecto de minería de datos, en etapas más avanzadas del proyecto serán obviados para evitar que penalicen al modelo en un futuro.

Luego, siempre que hay una cima, hay dos valles, uno a cada lado, según la profundidad de los valles, se podrá inferir la densidad del clúster. Esto, en términos prácticos, significa que cuanto más profundo sea el valle, más denso será el clúster, pues la distancia entre muestras será menor, esto puede verse en el eje Y de la gráfica de arriba. Por lo tanto, se observan dos clústeres altamente poblados, contabilizando el valle de la izquierda del segundo pico más alto y el valle de su derecha, que coincide con el valle de la izquierda del pico más alto. No obstante, al haber otros 3 picos más a parte de los dos más grandes, podrían quedar representados más clústeres, a parte de los dos ya mencionados.

Ahora se representa otra variante del diagrama de alcanzabilidad, en el cual se pueden observar las distancias entre puntos cercanos, trazadas dentro del mismo clúster e incluso entre clústeres diferentes.

```
#Se aplica el algoritmo k-means
Hawks2ConKIgualA3 <- kmeans(dfHawks2Normalizado, 3)

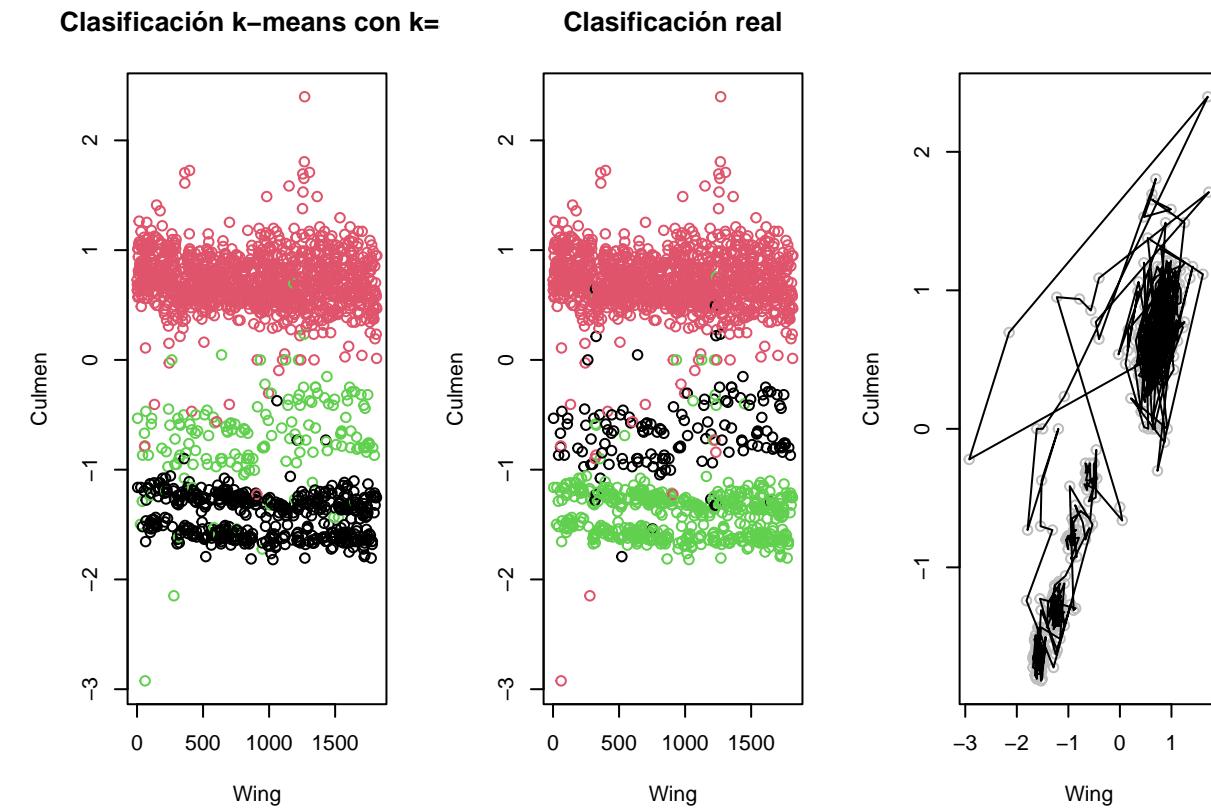
k=3
set.seed(1)

par(mfrow = c(1,3))
# Wing y Culmen con el dataframe de Hawks2 normalizado.
plot(c(dfHawks2Normalizado$Wing,dfHawks2Normalizado$Culmen), col=Hawks2NormalizadosK3$cluster, main="C")
```

```

plot(c(dfHawks2Normalizado$Wing, dfHawks2Normalizado$Culmen), col=as.factor(Hawks$Species), main="Clasificación k-means con k=3")
plot(dfHawks2Normalizado[c("Wing", "Culmen")], col = "grey")
polygon(dfHawks2Normalizado[c('Wing', 'Culmen')][observacionesWingCulmen$order,])

```



Como se puede ver en el resultado de arriba, se muestran tres gráficos, el correspondiente a la clasificación mediante el algoritmo de **k-means**, la clasificación real y el relacionado con las distancias intraclústers e interclústers. Puede verse claramente como hay un cluster altamente poblado, pues se ven todas las distancias aglutinadas en un área determinada. Luego, para el resto de distancias del diagrama de trazabilidad, se observan los clústers negros y verdes. Gracias al algoritmo **OPTICS** y al plot **polygon(·)** se pueden ver las distancias entre muestras dentro de un mismo clúster, así como fuera del mismo, y aunque el **k-means** y el **OPTICS** son dos algoritmos distintos, los dos computan distancias, y si solo se observa el resultado gráfico de las distancias, se ve como los dos clústers de abajo están muy próximos y sus muestras podrían perfectamente clasificarse bajo un mismo clúster, pero aun así el algoritmo k-means logra clasificar la mayoría de las muestras correctamente, esto puede deberse a su dependencia en la media de las características, en la escala de los datos de cada atributo o en los valores iniciales de los centroides, que posteriormente los nuevos, serán calculados a partir de las medias de los clústeres actuales.

Ahora se va a proceder a la extracción de una agrupación de la ordenación llevada a cabo por el algoritmo **OPTICS**, de manera similar a la que el algoritmo **DBSCAN** habría generado, estableciendo inicialmente  $\varepsilon_{cl} = 1$ . No obstante, en el resto de análisis se calcularán los resultados para diferentes valores de  $\varepsilon$ . Véase el primer experimento en el chunk de abajo:

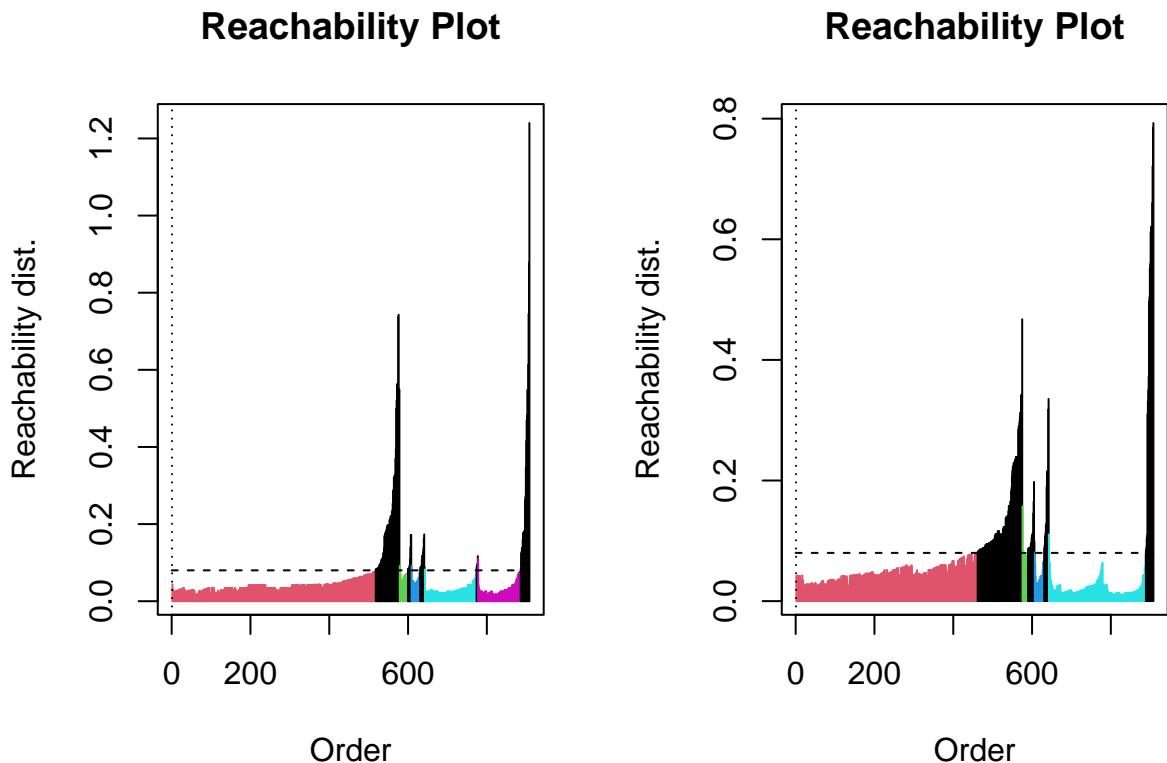
```

#Se vuelve a implementar el algoritmo de OPTICS, para asegurarse que se están usando los datos deseados
resultadosWingCulmen <- extractDBSCAN(observacionesWingCulmen, eps_cl = 0.08) #El valor observacionesWi

observacionesWeightCulmen <- optics(dfHawks2Normalizado[c("Weight", "Culmen")], minPts = 8)
resultadosWeightCulmen <- extractDBSCAN(observacionesWeightCulmen, eps_cl = 0.08)

```

```
#Ahora se representa la extracción llevada a cabo arriba para los dos pares de atributos:  
par(mfrow = c(1,2))  
  
plot(resultadosWingCulmen)  
plot(resultadosWeightCulmen)
```

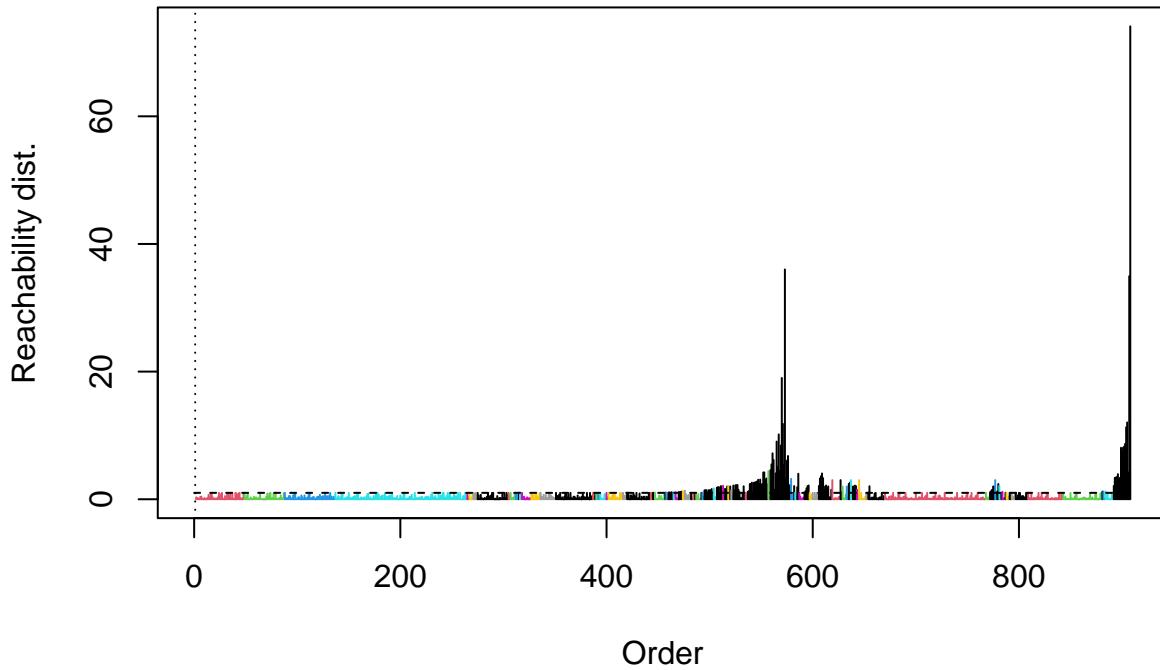


Como puede observarse, hay ruido en las muestras (el color negro de las cimas). Además, el algoritmo detecta 4 clústeres para el par Weight-Culmen, mientras que para el par Wing-Culmen, detecta 5. Esto era de esperar, sabiendo que en los dos pares hay 5 picos, no obstante, en el segundo par solo detecta 4 clústeres porque esa colina no es lo suficientemente alta como para que esas muestras asociadas sean consideradas outliers debido a la distancia de alcanzabilidad a la que se encuentran, por lo que produce que los dos valles colindantes queden bajo el paraguas del mismo clúster.

Ahora a continuación se prueba con más valores de  $\varepsilon_{cls}$  (EN ESTE CHUNK, los datos NO están NORMALIZADOS)

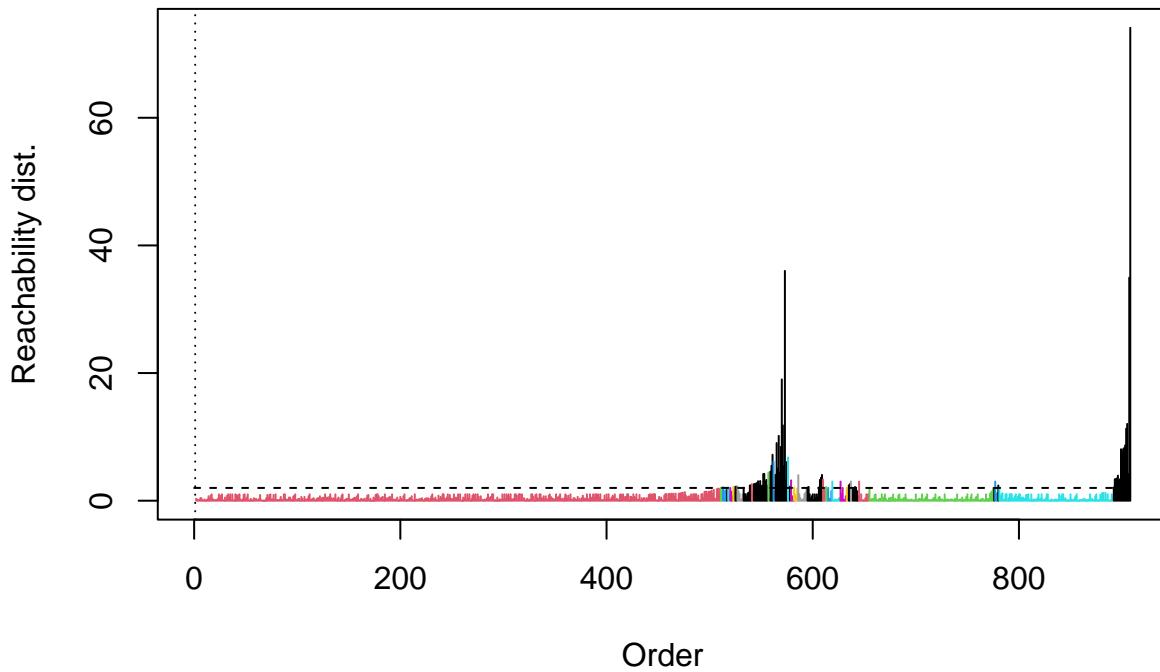
```
observaciones <- optics(Hawks2[c("Wing", "Culmen")], minPts = 2)  
  
#Ahora se representa la extracción llevada a cabo arriba:  
for (i in 1:30){  
  cat("\nEste es el valor de i: ", i)  
  ress <- extractDBSCAN(observaciones, eps_cl = i)  
  plot(ress, title(main = "Resultado para $\varepsilon_{\text{cl}}"))  
}  
  
##  
## Este es el valor de i: 1
```

## Reachability Plot



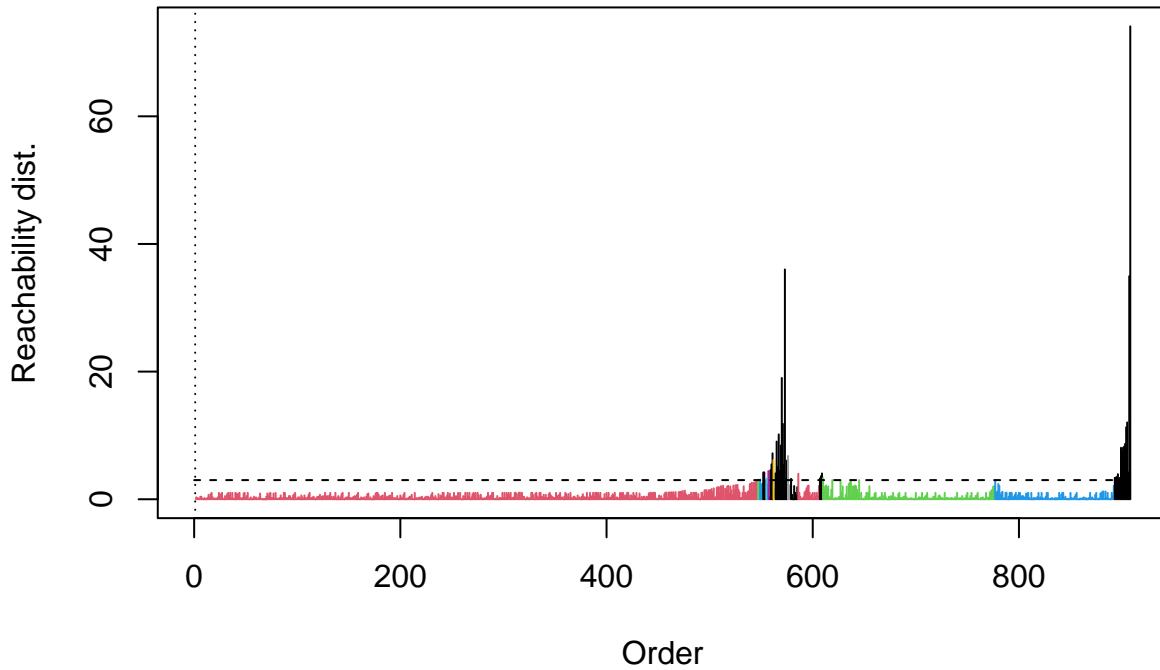
```
##  
## Este es el valor de i: 2
```

## Reachability Plot



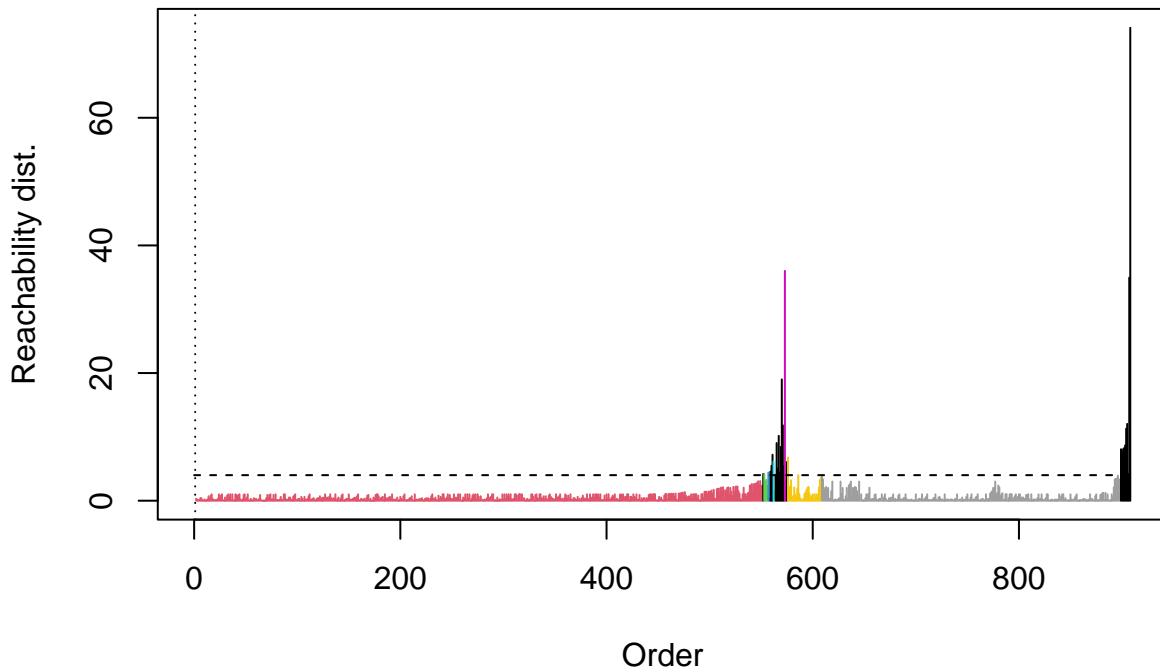
```
##  
## Este es el valor de i: 3
```

## Reachability Plot



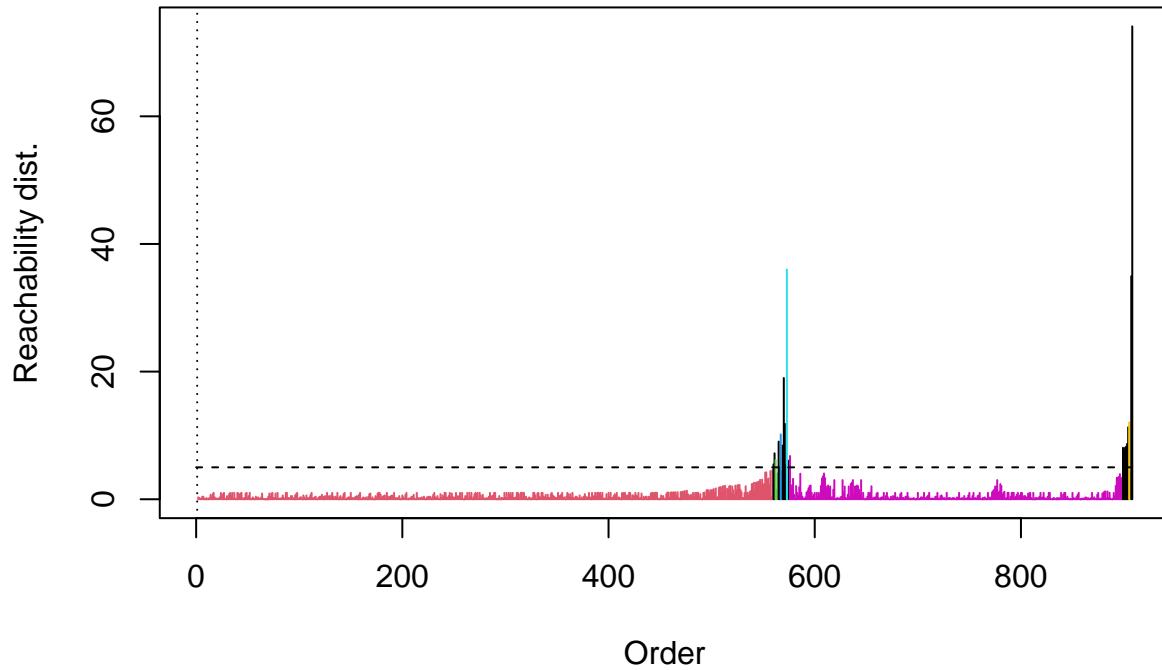
```
##  
## Este es el valor de i: 4
```

## Reachability Plot



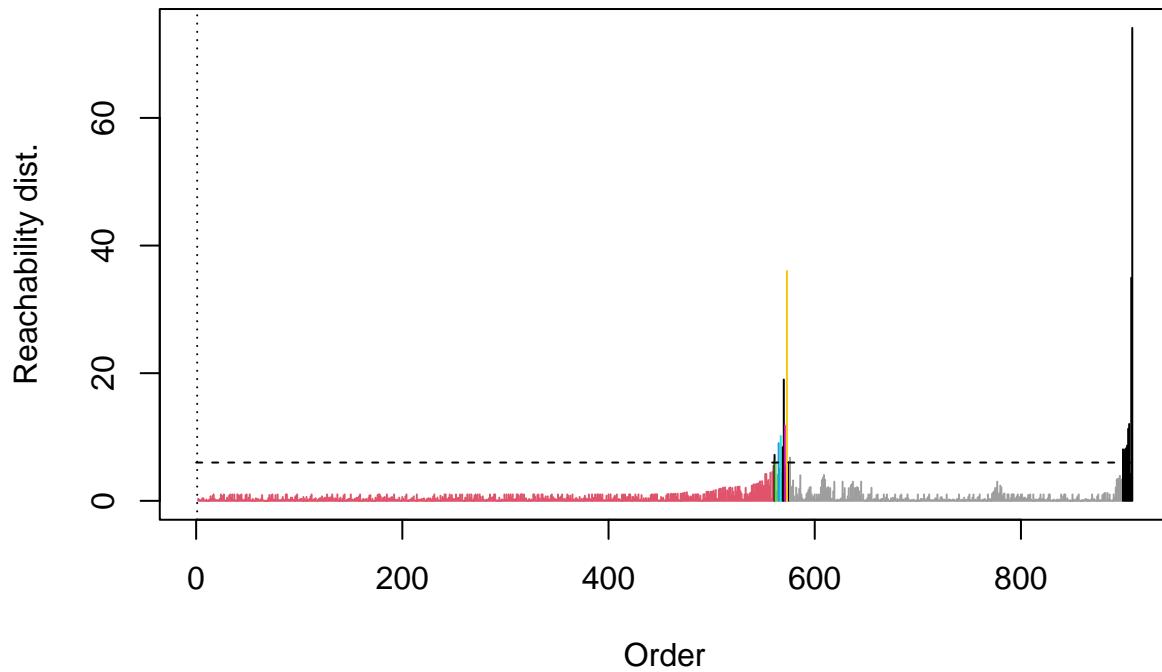
```
##  
## Este es el valor de i: 5
```

## Reachability Plot



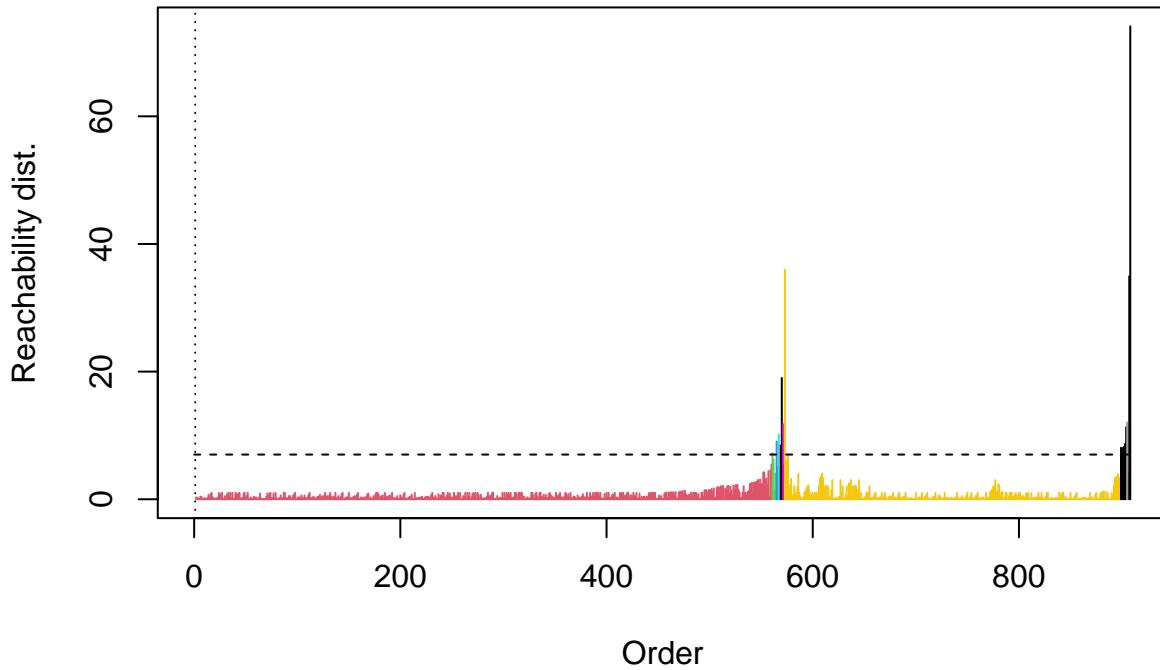
```
##  
## Este es el valor de i: 6
```

## Reachability Plot



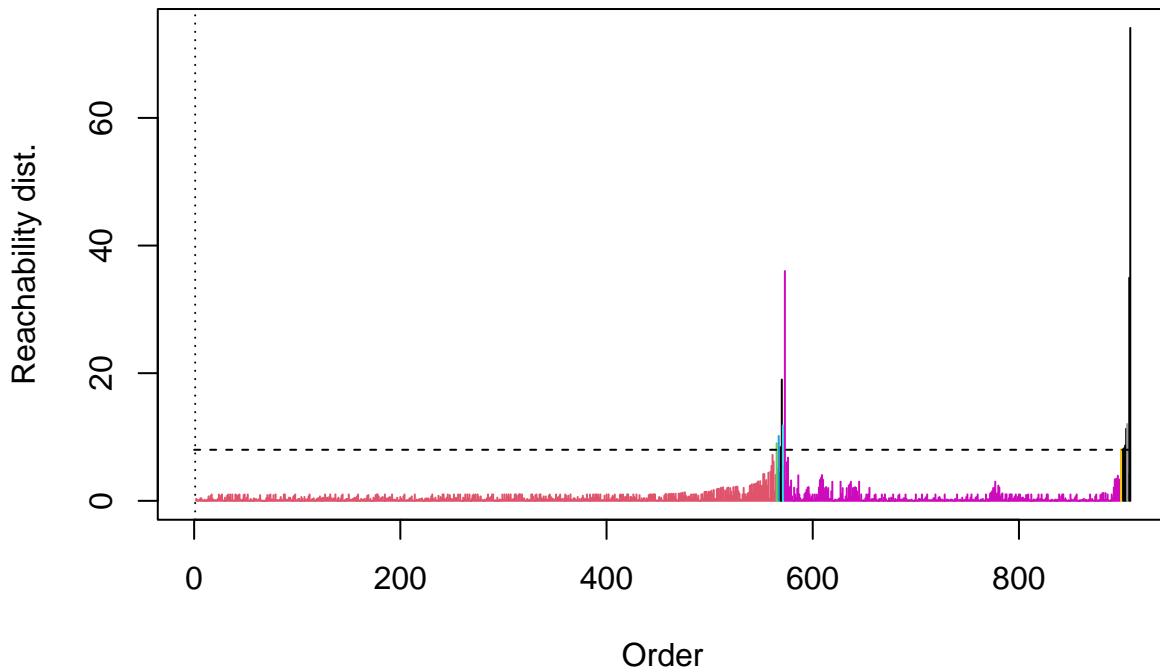
```
##  
## Este es el valor de i: 7
```

## Reachability Plot



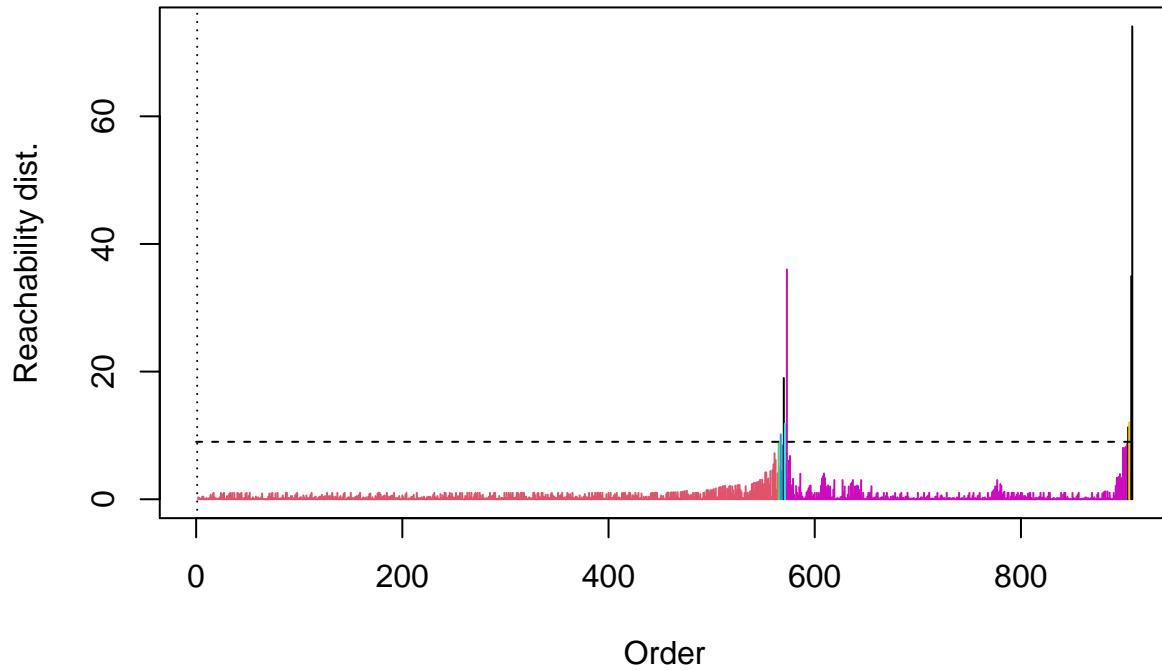
```
##  
## Este es el valor de i: 8
```

## Reachability Plot



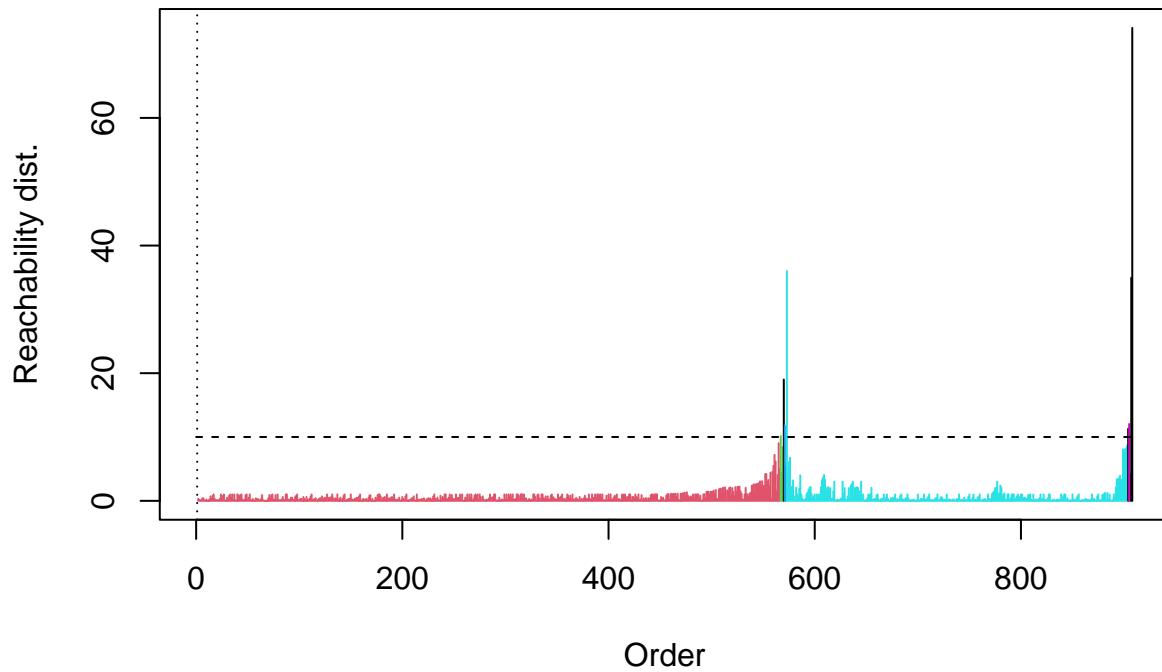
```
##  
## Este es el valor de i: 9
```

## Reachability Plot



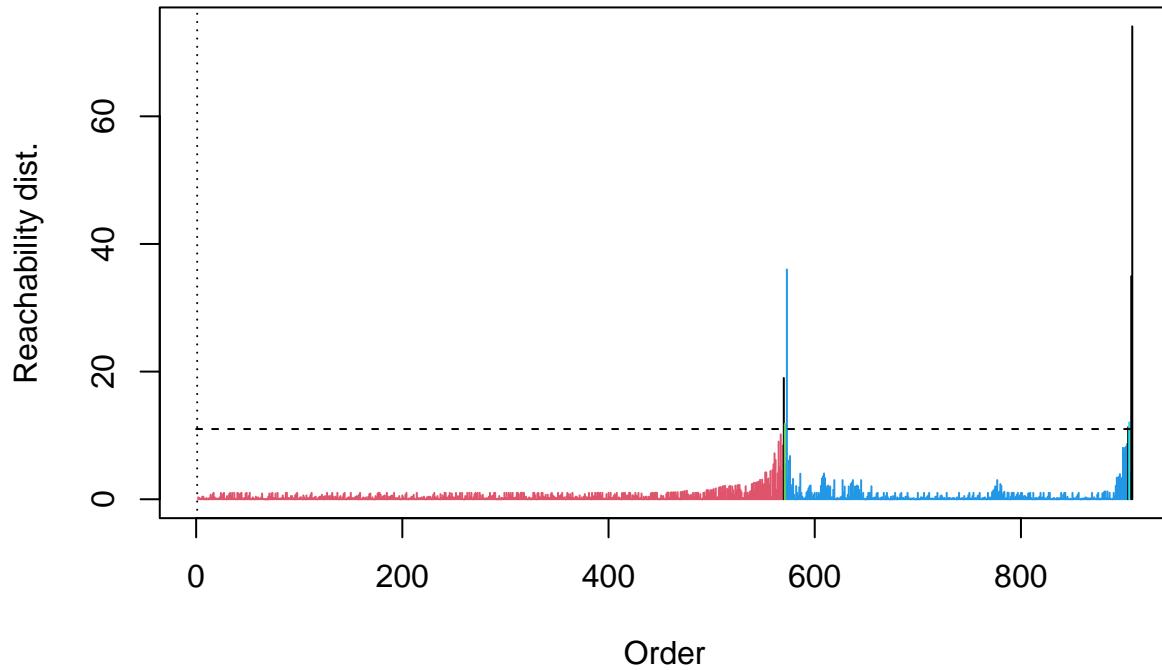
```
##  
## Este es el valor de i: 10
```

## Reachability Plot



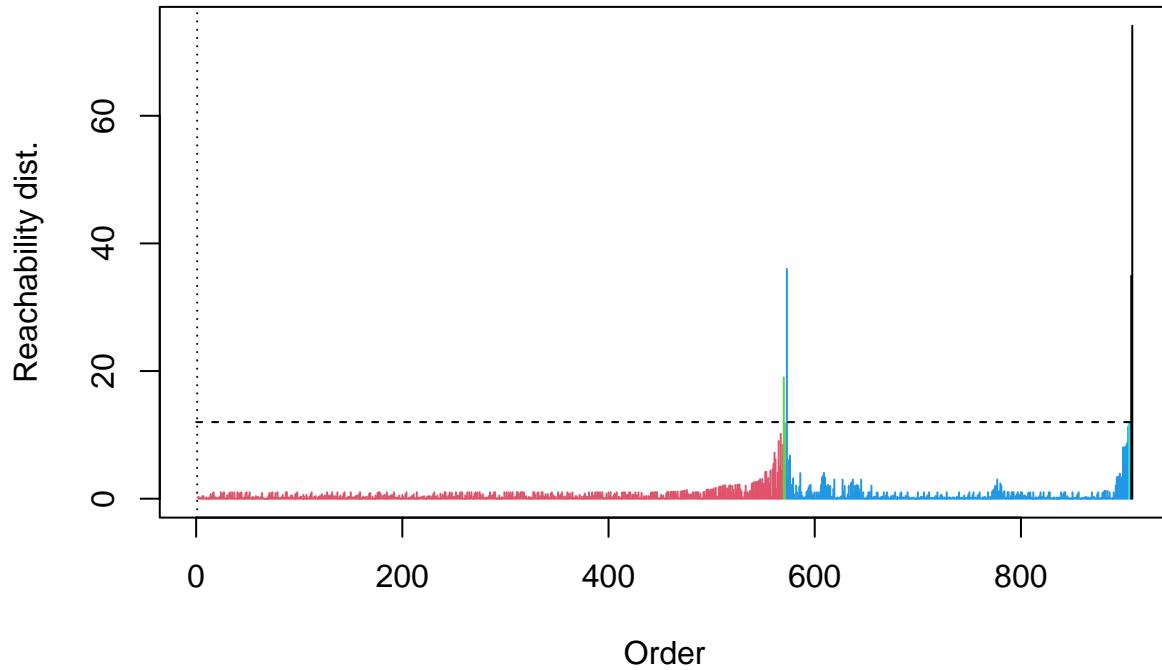
```
##  
## Este es el valor de i: 11
```

## Reachability Plot



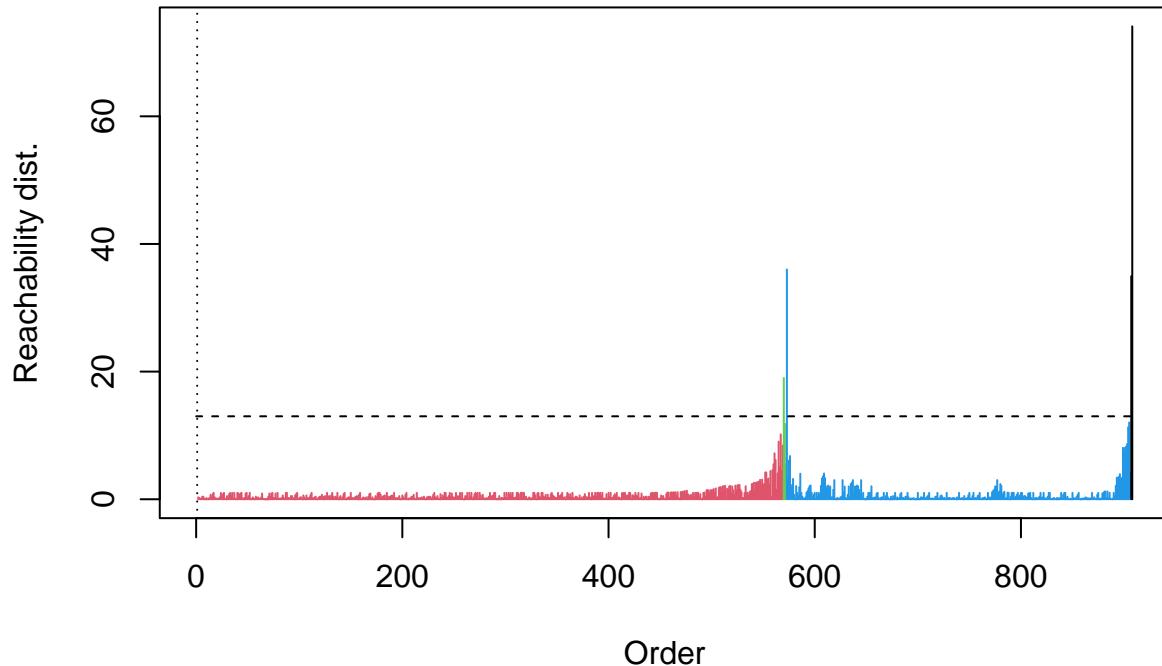
```
##  
## Este es el valor de i: 12
```

## Reachability Plot



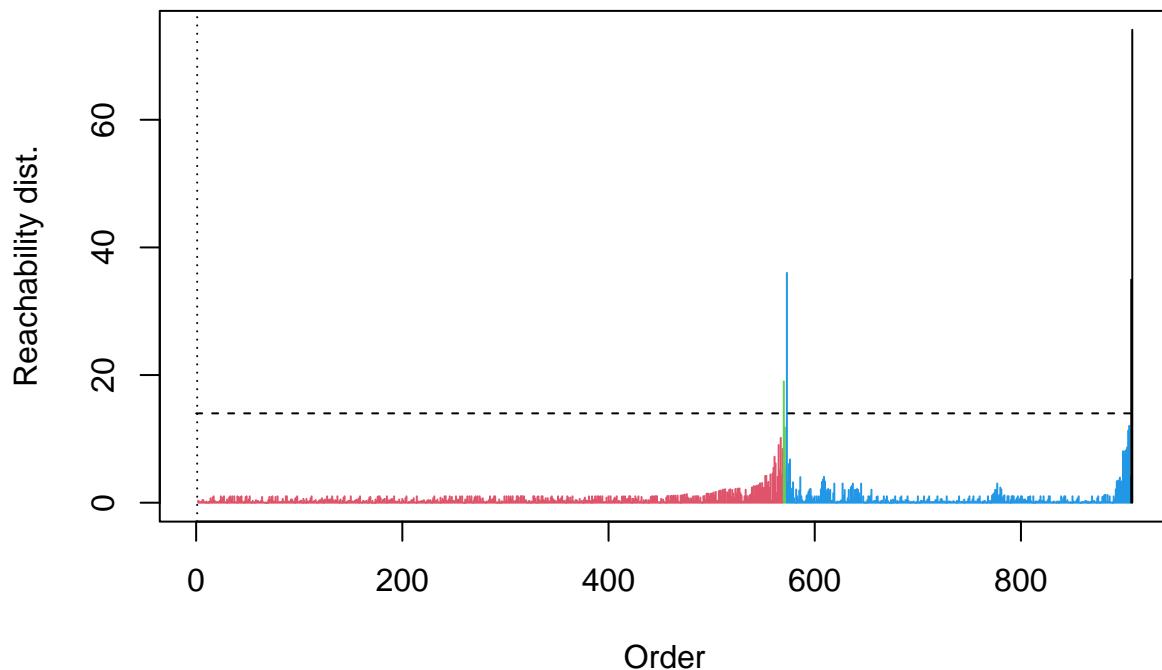
```
##  
## Este es el valor de i: 13
```

## Reachability Plot



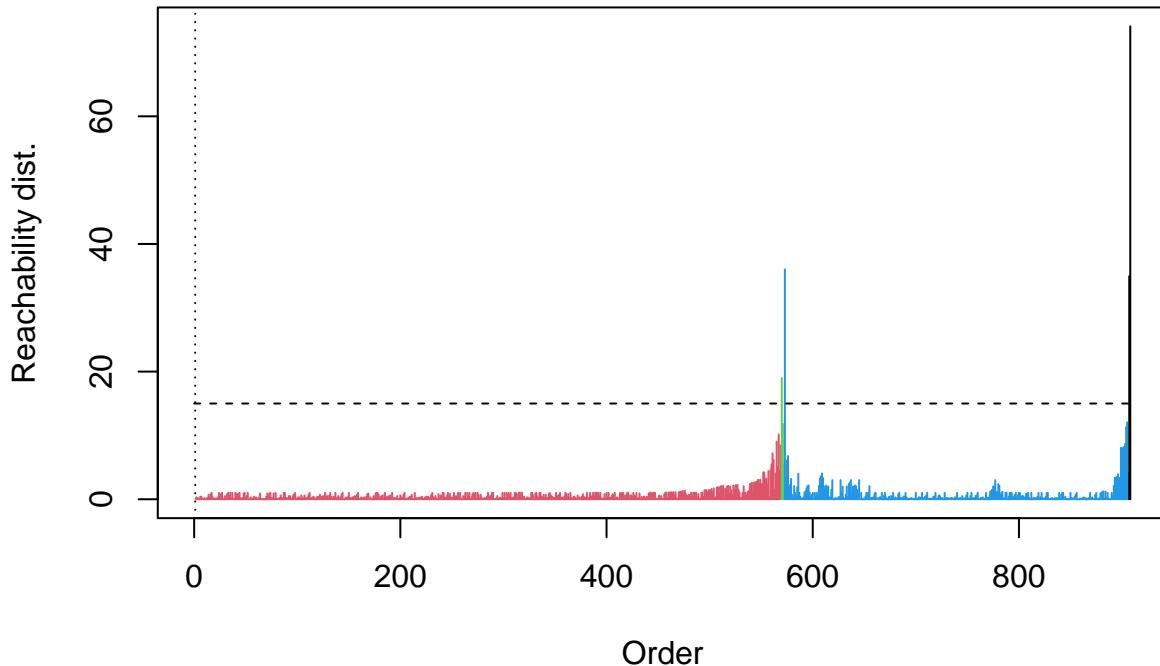
```
##  
## Este es el valor de i: 14
```

## Reachability Plot



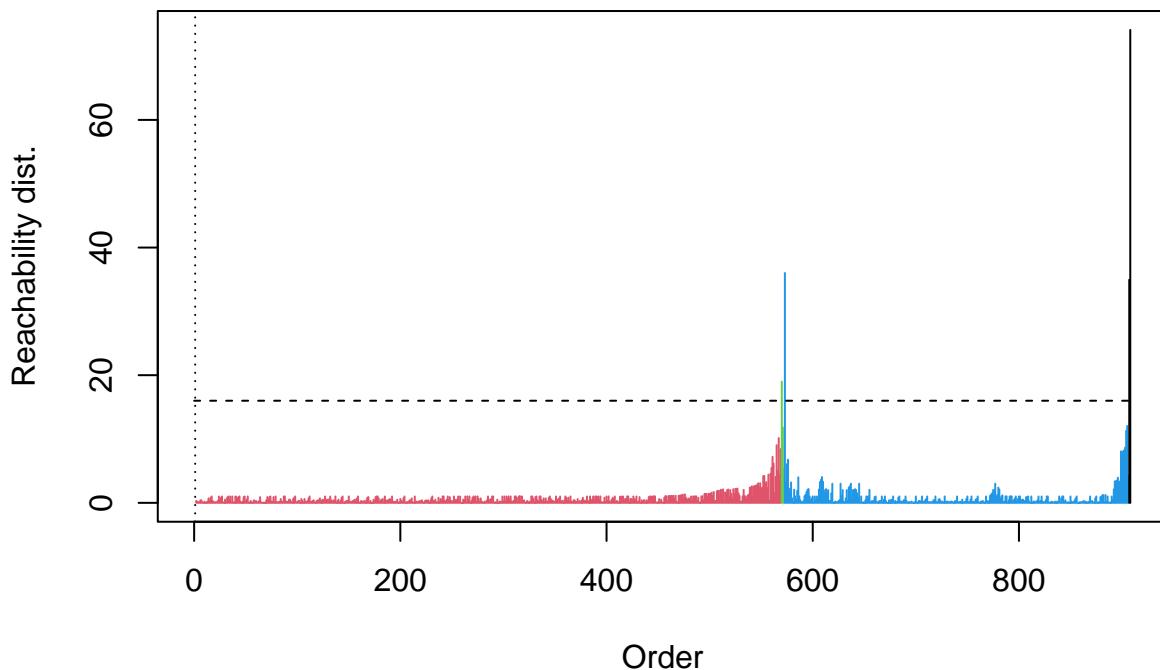
```
##  
## Este es el valor de i: 15
```

## Reachability Plot



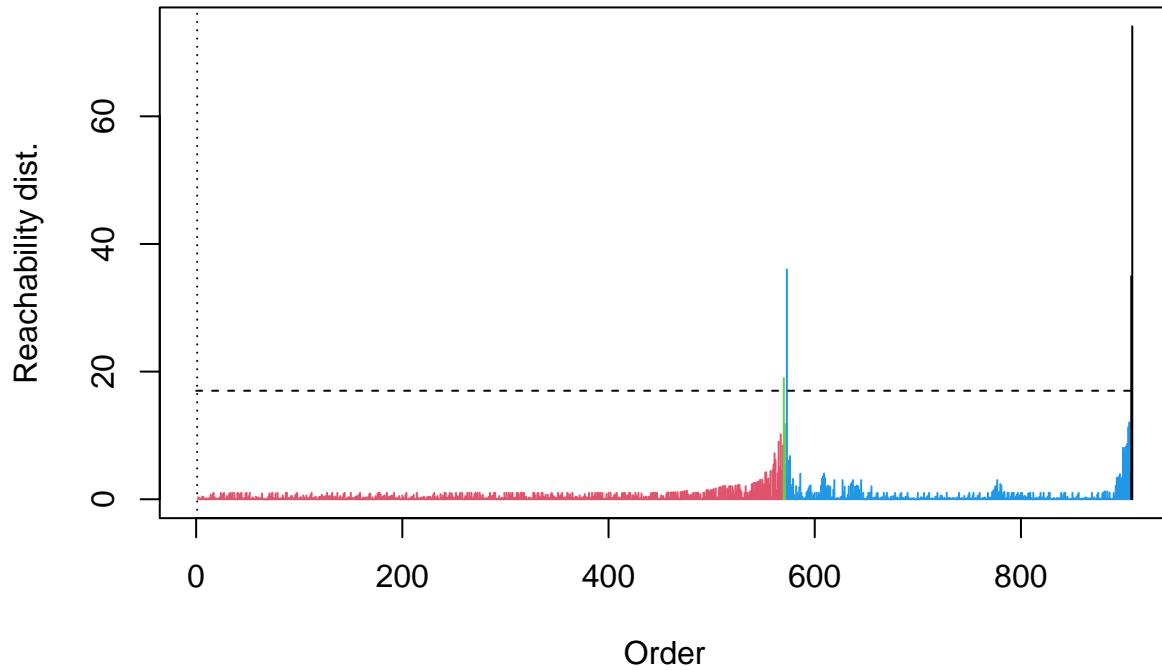
```
##  
## Este es el valor de i: 16
```

## Reachability Plot



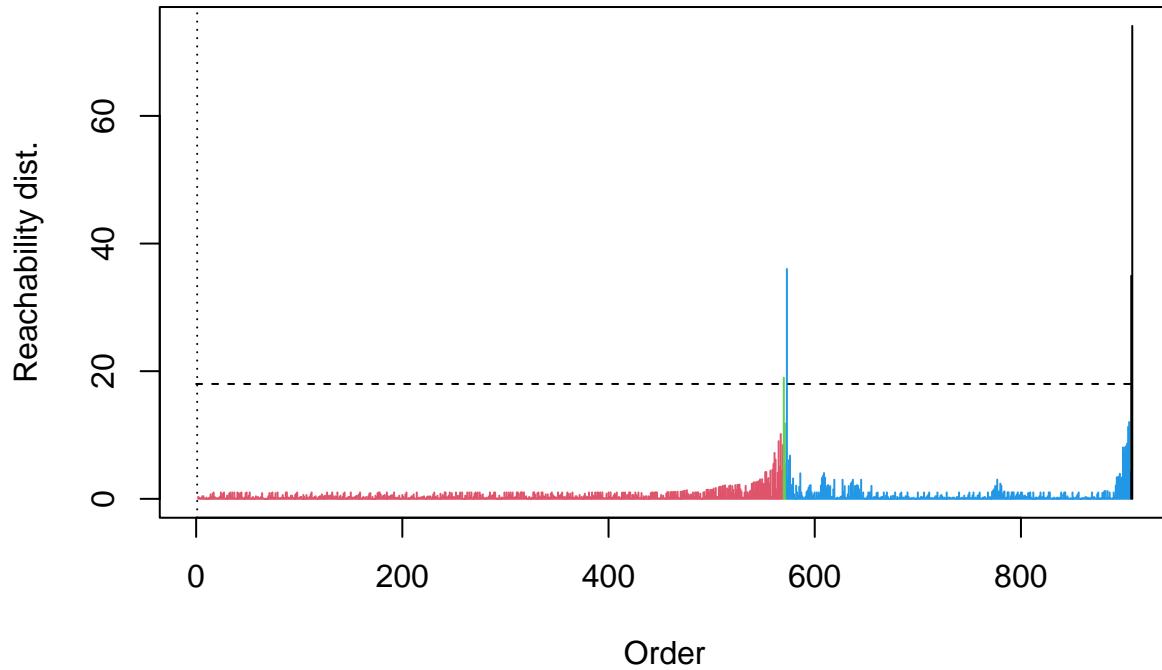
```
##  
## Este es el valor de i: 17
```

## Reachability Plot



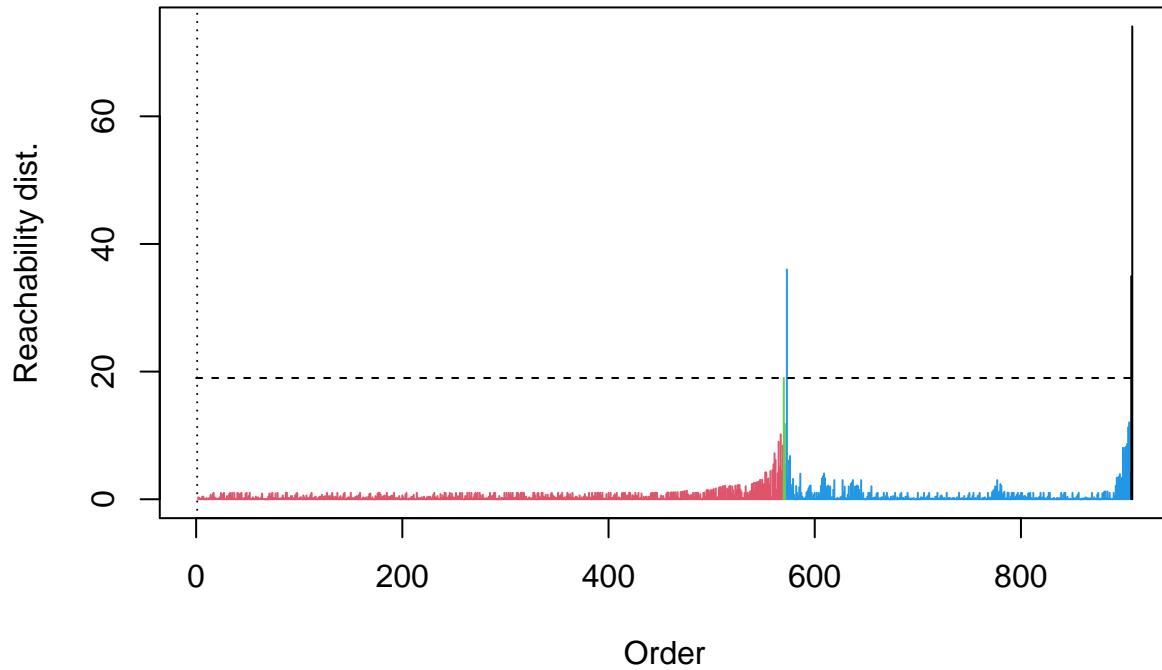
```
##  
## Este es el valor de i: 18
```

## Reachability Plot



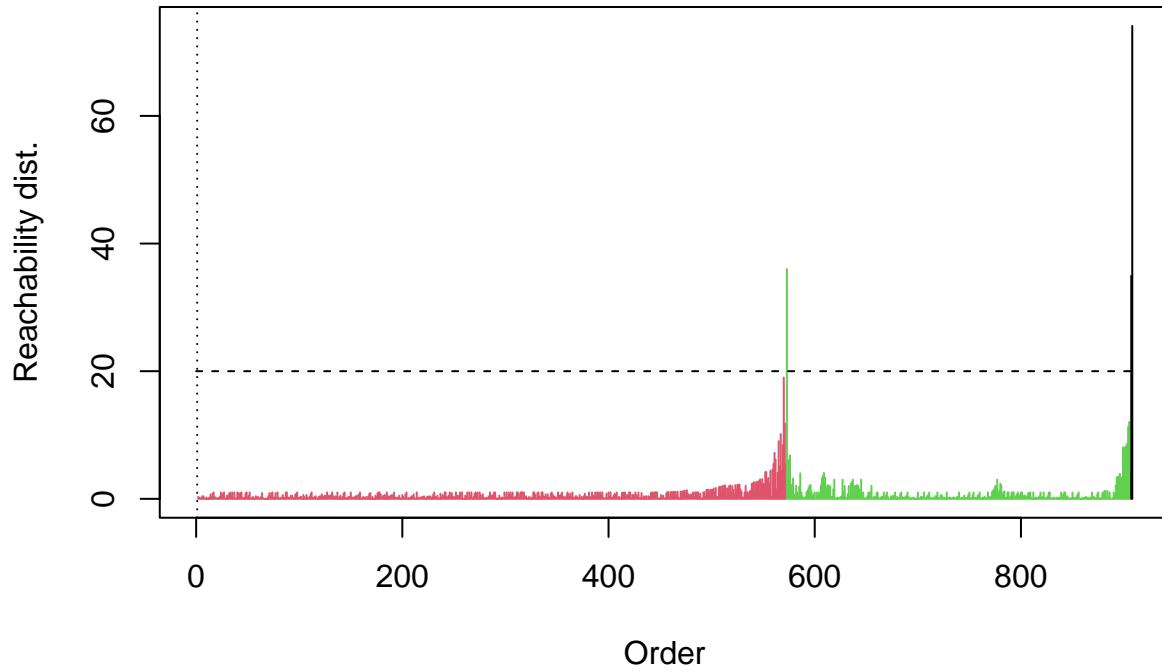
```
##  
## Este es el valor de i: 19
```

## Reachability Plot



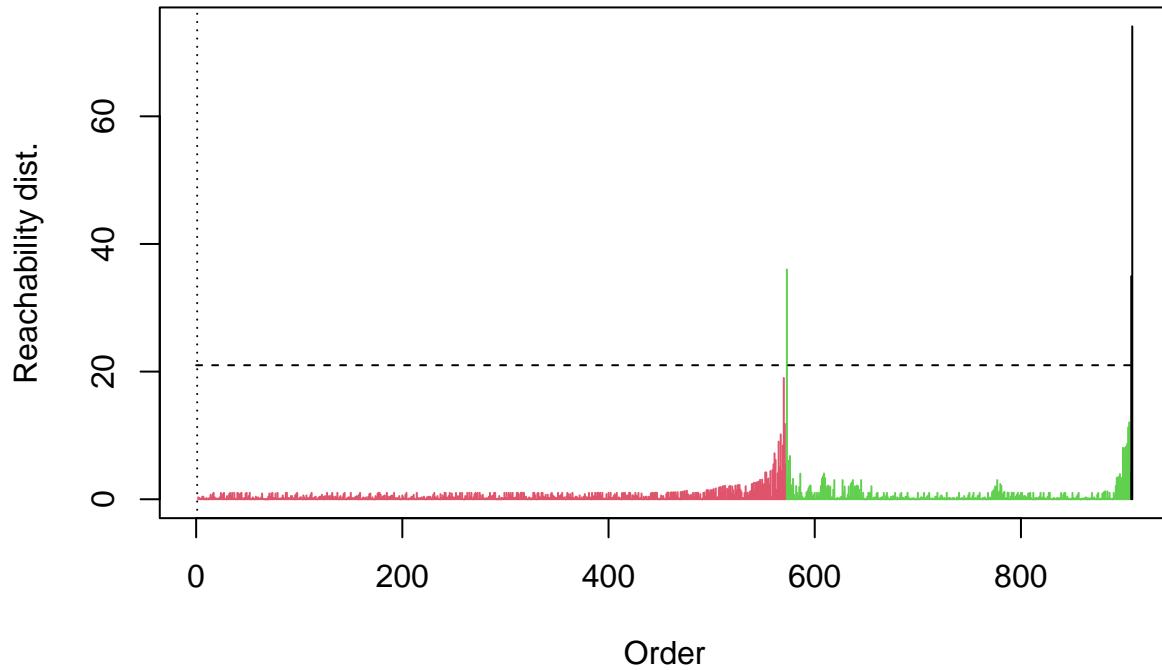
```
##  
## Este es el valor de i: 20
```

## Reachability Plot



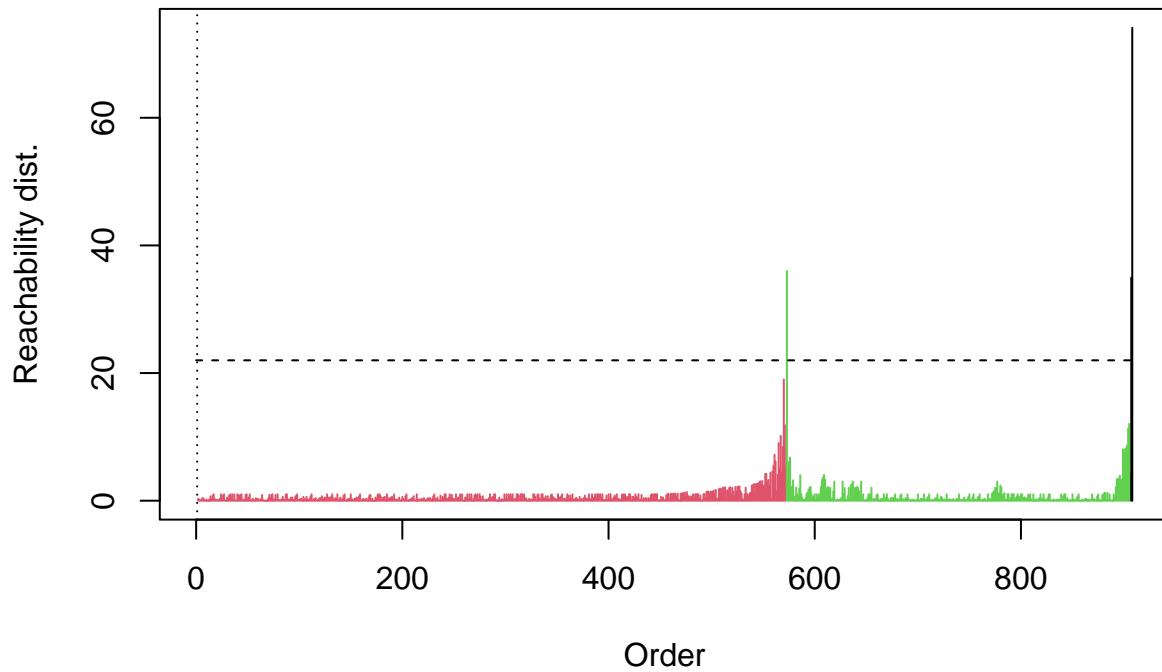
```
##  
## Este es el valor de i: 21
```

## Reachability Plot



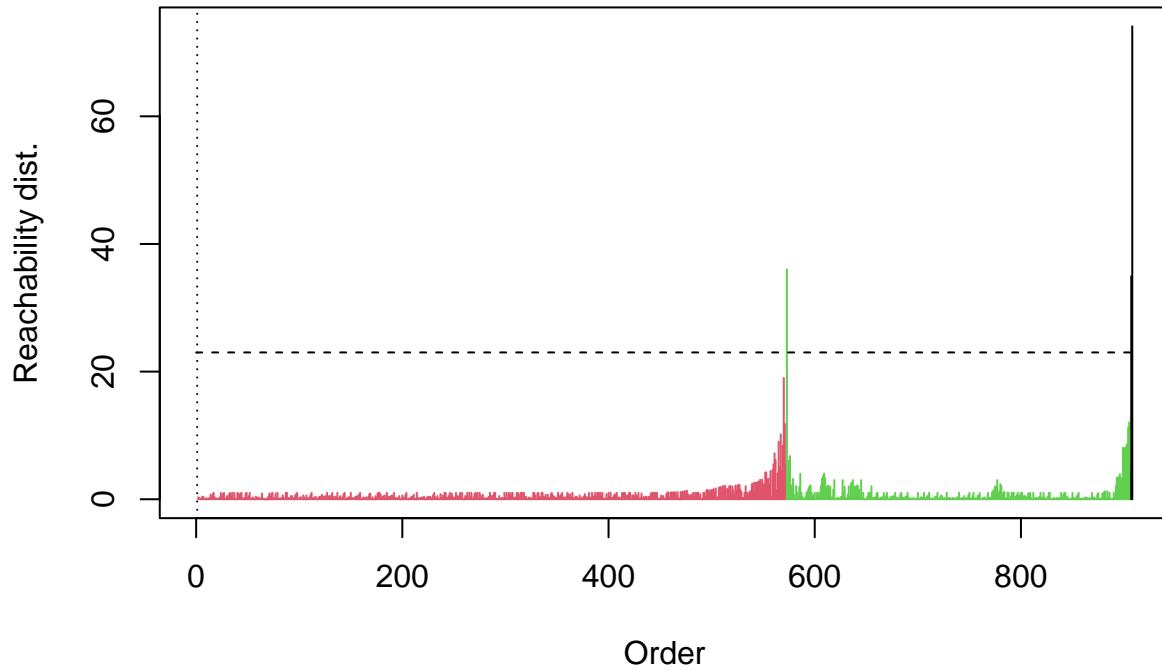
```
##  
## Este es el valor de i: 22
```

## Reachability Plot



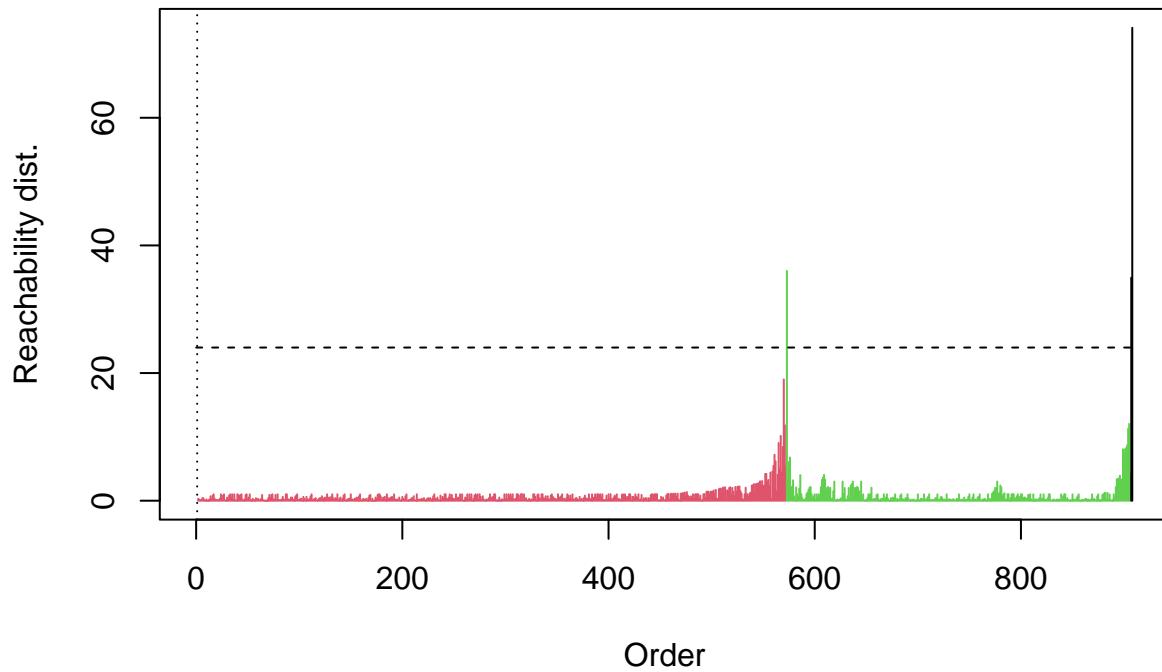
```
##  
## Este es el valor de i: 23
```

## Reachability Plot



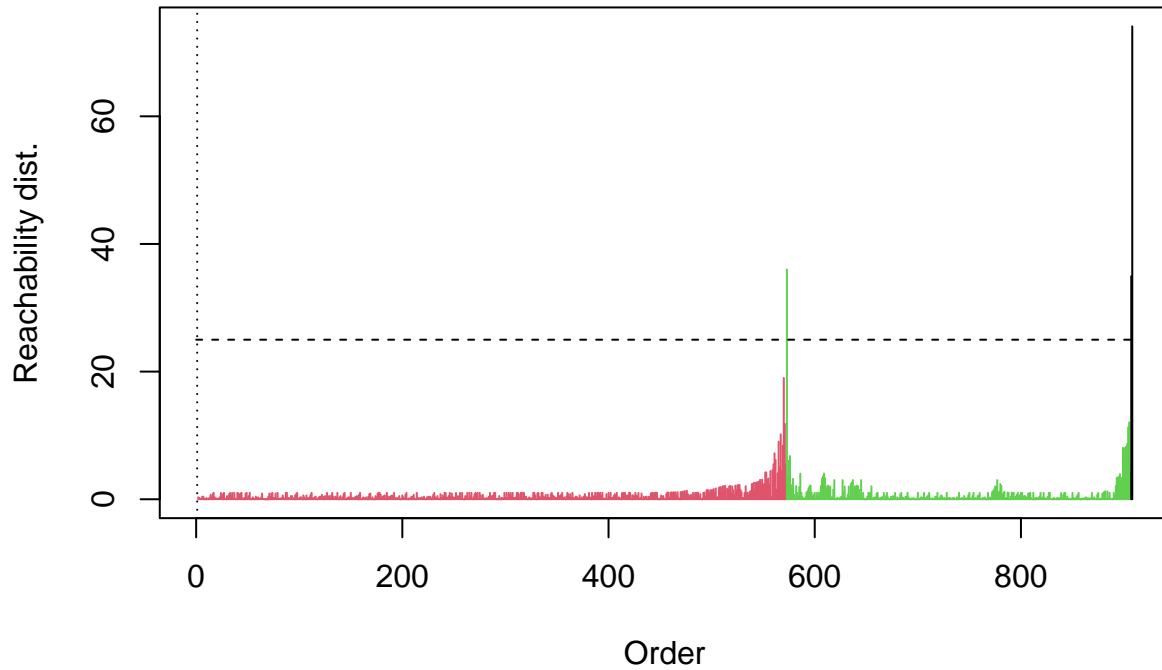
```
##  
## Este es el valor de i: 24
```

## Reachability Plot



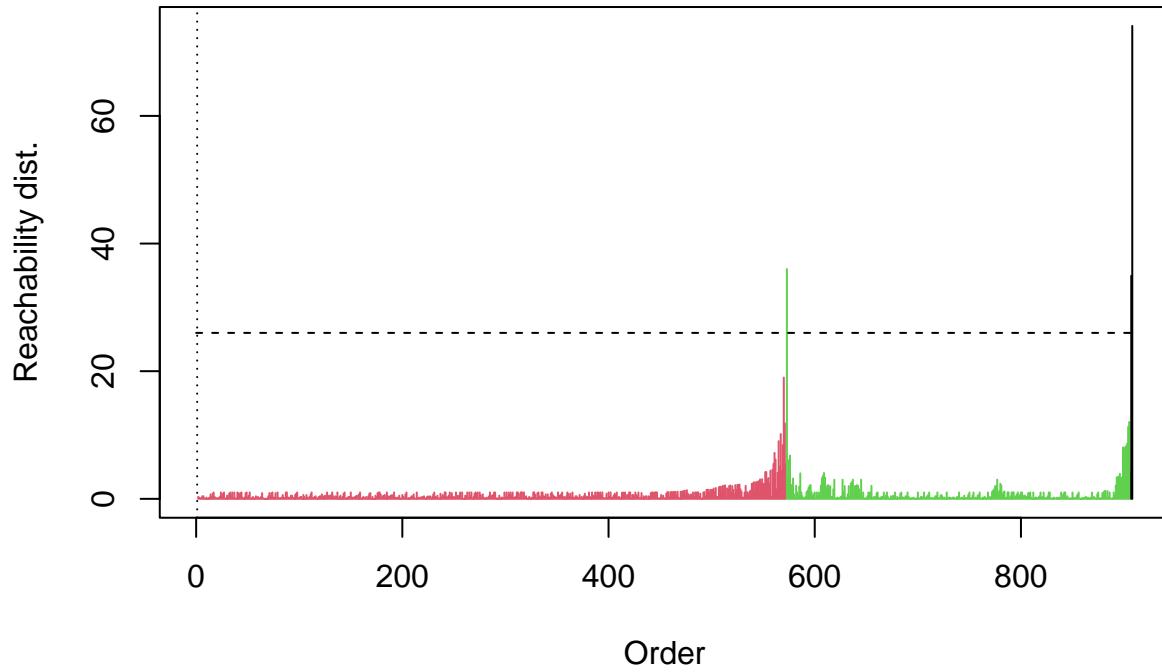
```
##  
## Este es el valor de i: 25
```

## Reachability Plot



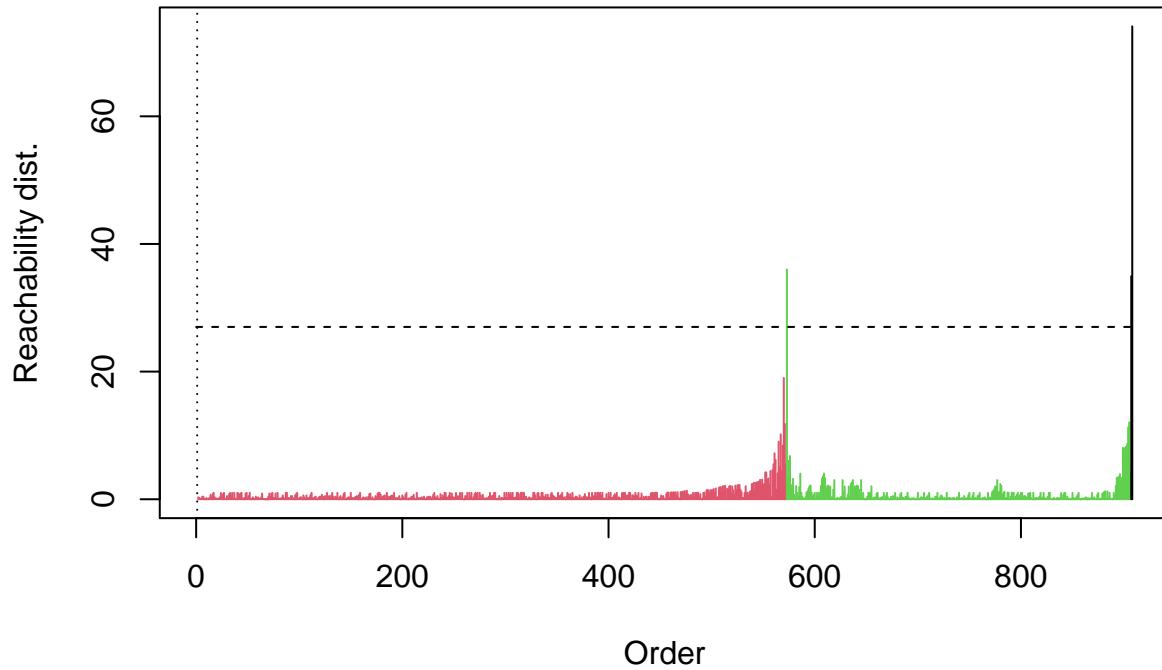
```
##  
## Este es el valor de i: 26
```

## Reachability Plot



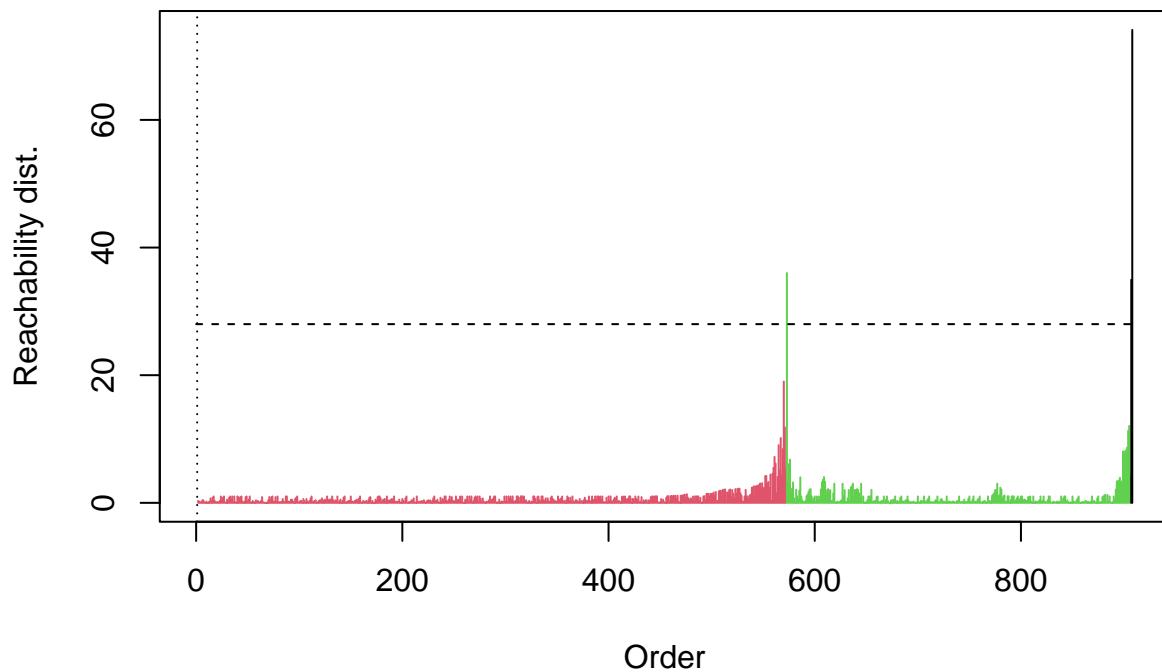
```
##  
## Este es el valor de i: 27
```

## Reachability Plot



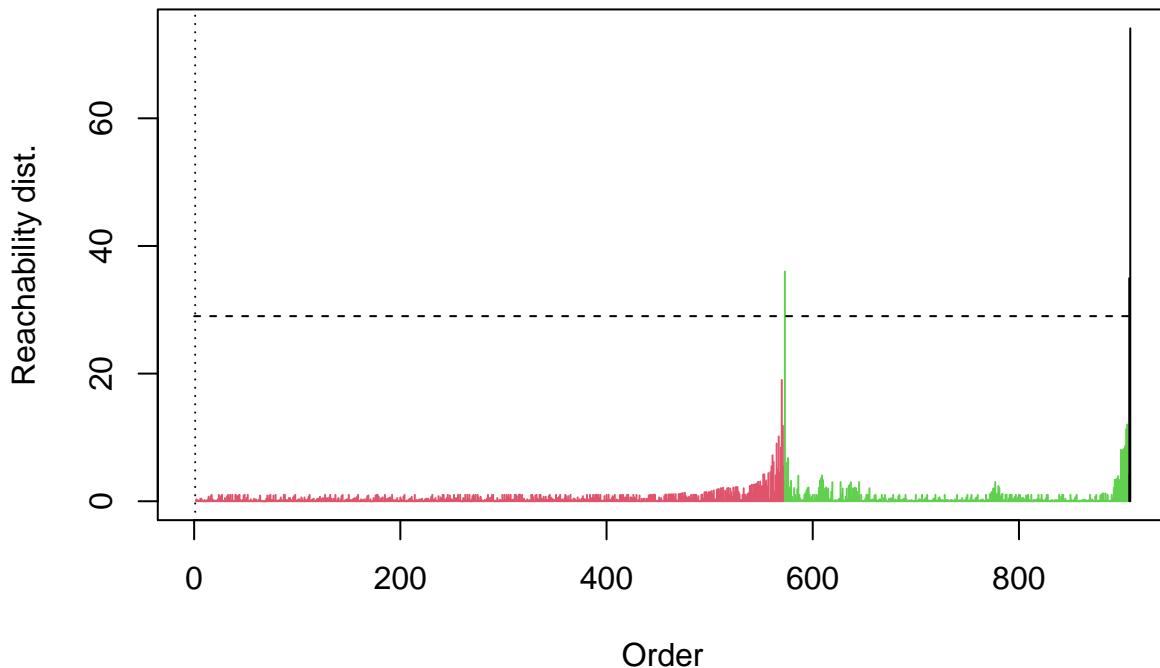
```
##  
## Este es el valor de i: 28
```

## Reachability Plot



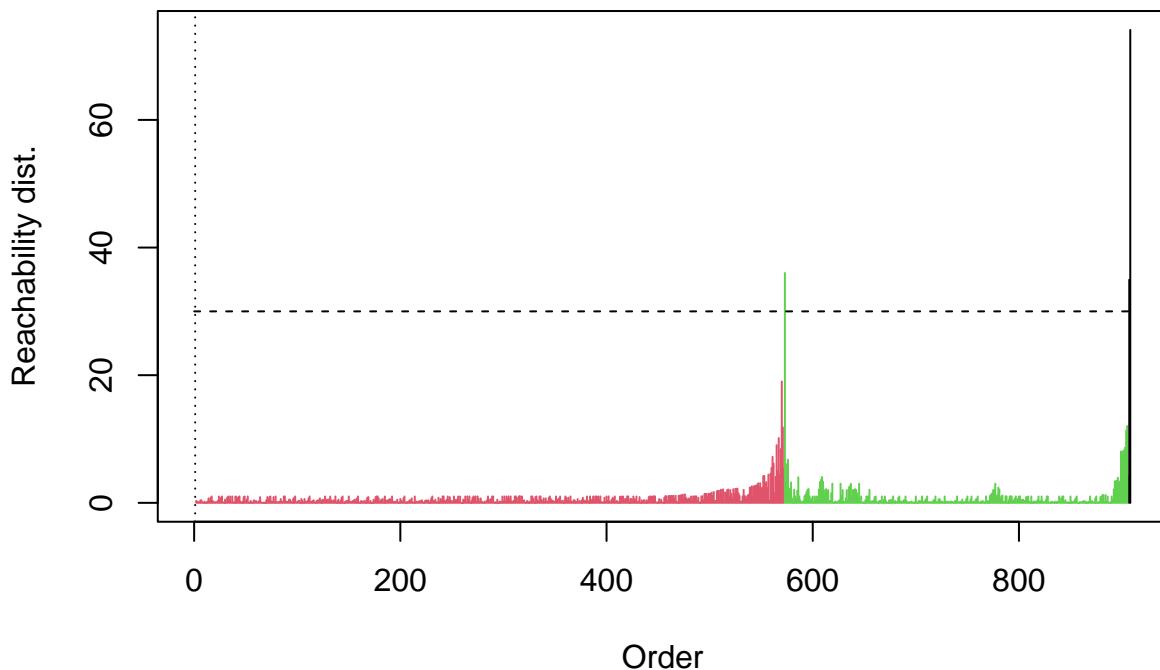
```
##  
## Este es el valor de i: 29
```

## Reachability Plot



```
##  
## Este es el valor de i: 30
```

## Reachability Plot



Como se puede observar por el resultado de arriba, se puede ver como según aumenta el valor de  $\varepsilon$  el ruido en los datos disminuye. Se puede ver además cómo para  $\varepsilon = 2.065$  se observan 3 clústeres pronunciados, con los outliers (cimas). Ahora bien, para ese mismo valor de  $\varepsilon$  se observa

como gran parte de la cima primaria (la más alta) y la de la cima secundaria están negro, esto quiere decir que se corresponde con ruido dentro de nuestro juego de datos, ¿esto quiere decir acaso que ese ruido se corresponde con muestras de nuestro juego de datos? no, pues pueden corresponderse perfectamente con los llamados outliers. En el ejercicio anterior, se ha visto como las cimas se correspondían con los outliers, y aunque no se esperaba que la mayor parte de la montaña fuese de otro color que no fuese negro, se esperaba que esta no fuese del todo negra, y que solo la cima fuese negra.

Dicho lo anterior, se ha investigado la relación entre el valor de  $\varepsilon$ , la cantidad de ruido observable en el gráfico, y el número de clusters. Según aumenta el valor de  $\varepsilon$ , el ruido disminuye, y el número de clústers representados también lo hace. A partir de  $\varepsilon = 4.065$  se obtienen dos clústers principales, algo extraño ya que se deberían de ver como mínimo 3 clústers de tamaño considerable. También es verdad, que a partir de  $\varepsilon = 8.065$  se pueden observar los dos clústers principales, y luego, un clúster de color cian a los dos lados del segundo pico más alto, este clúster podría corresponderse al tercer clúster que faltaba, aunque esto no es algo que sea conveniente mucho, sabiendo además que cuanto más alto esté el clúster más crece la distancia entre muestras, algo que no tiene mucho sentido, sabiendo que la distancia entre el clúster verde y rojo en la clasificación real no es tan grande. No obstante, puede que ese cluster color cian no se corresponda con el clúster que faltaba, sino con el propio algoritmo debido a los valores de epsilon. Dicho esto, la falta de un cluster, puede deberse al hecho de que los tres clústeres estén bastante juntos, especialmente los dos de abajo (mirando la gráfica de la clasificación real para el algoritmo de k-means y para los atributos **Wing** y **Culmen**), al estar tan juntos, en el diagrama de arriba no habrá puntos que estén situados a una distancia de alcanzabilidad alta. Esta puede ser la razón por la cual no se indique un tercer clúster de tamaño considerable, entre los dos picos, porque entre los dos picos grandes solos hay un valle con picos muy pequeños, porque no hay casi puntos que se encuentren lejos de los dos clústeres principales, ya que están demasiado juntos. Por lo tanto, a pesar de ruido que se observaba en el gráfico de arriba, para  $\varepsilon = 2.065$  que se observaban 3 clústeres principales, es el resultado que más se ajusta a la clasificación real, a pesar de tener muchos más clústeres diminutos.

Es importante mencionar que en el mejor resultado de entre todos los gráficos de arriba (el gráfico con  $\varepsilon = 2.065$ ) se observa ruido y muchos más clústeres de los necesarios. Es por esto, que se va a repetir la implementación del algoritmo y de todo el ejercicio, pero esta vez, para los 4 atributos normalizados.

Ahora se va a proceder a la extracción de antes, pero para los atributos ya normalizados y para diferentes valores de epsilon:

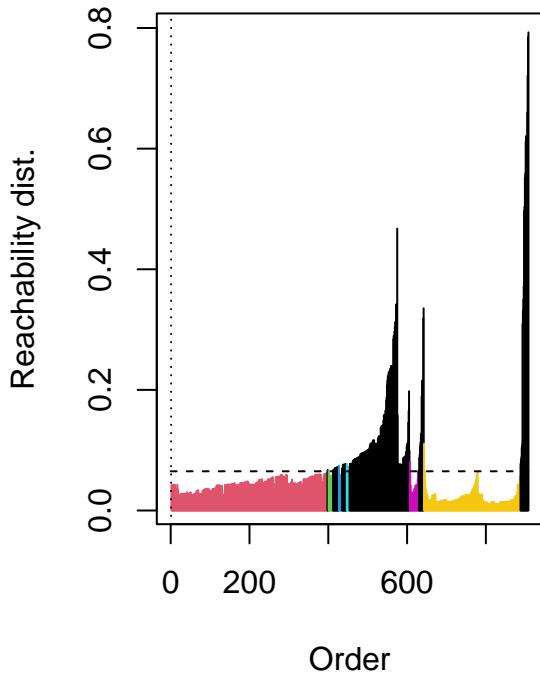
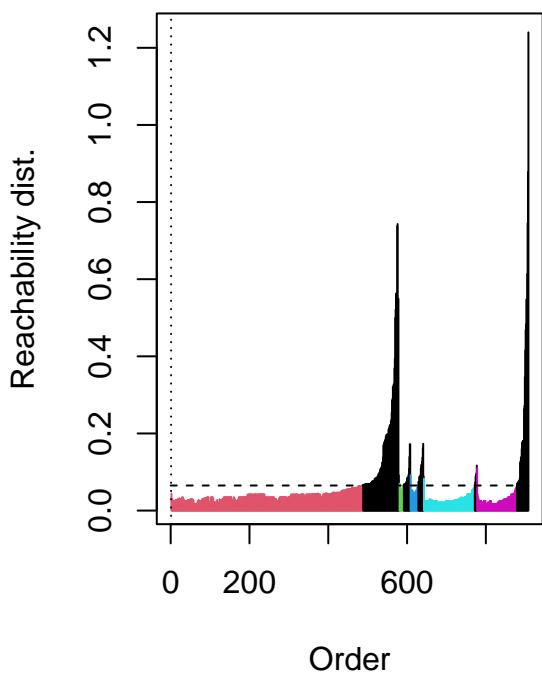
```
par(mfrow = c(1, 2))

observacionesWingCulmen <- optics(dfHawks2Normalizado[c("Wing", "Culmen")], minPts = 8)
observacionesWeightCulmen <- optics(dfHawks2Normalizado[c("Weight", "Culmen")], minPts = 8)

#Ahora se representa la extracción llevada a cabo arriba:
for (i in c(0.065, 0.07, 0.08, 0.09, 0.1, 0.2, 0.5, 0.8, 1)){
  cat("\nEste es el valor de i: ", i)
  resultatWingCulmen <- extractDBSCAN(observacionesWingCulmen, eps_cl = i)
  resultatWeightCulmen <- extractDBSCAN(observacionesWeightCulmen, eps_cl = i)
  plot(resultatWingCulmen, main = paste("Wing-Culmen con Epsilon: ", i))
  plot(resultatWeightCulmen, main = paste("Weight-Culmen con Epsilon: ", i))
}

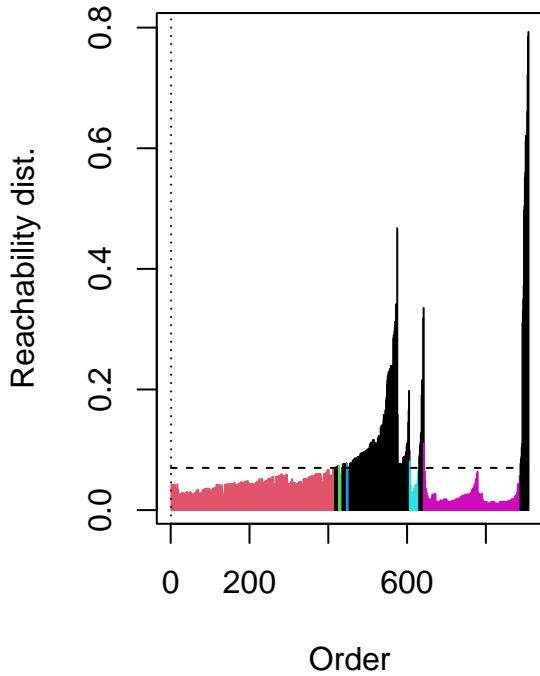
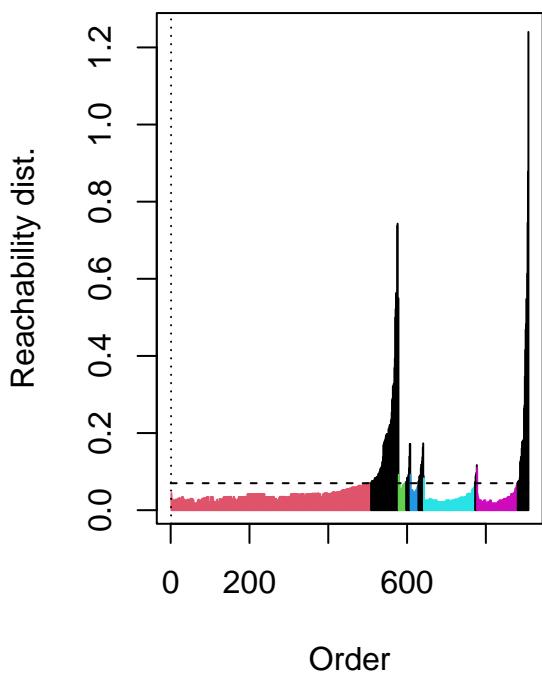
## 
## Este es el valor de i:  0.065
```

**Wing–Culmen con Epsilon: 0.06 Weight–Culmen con Epsilon: 0.0**



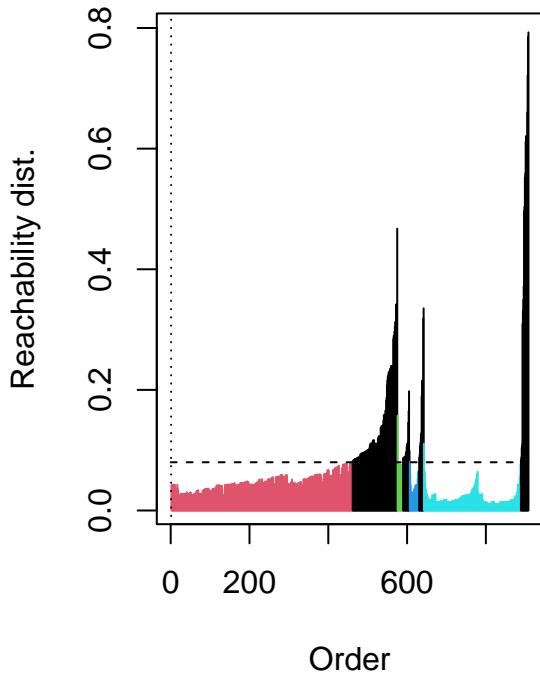
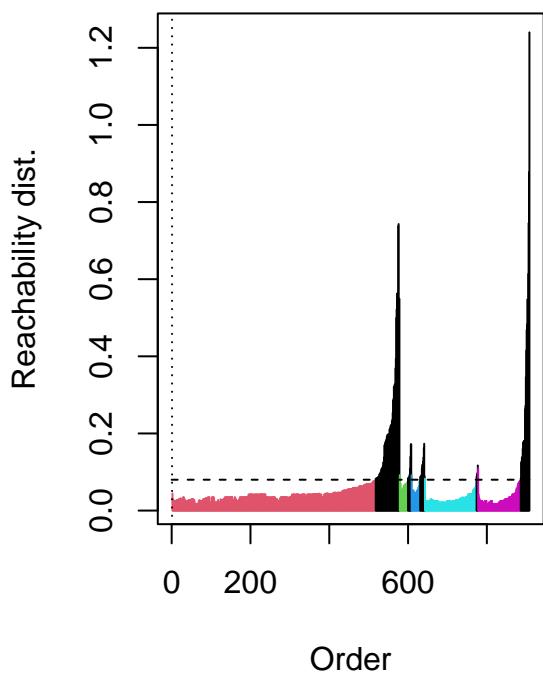
```
##  
## Este es el valor de i: 0.07
```

**Wing–Culmen con Epsilon: 0.07 Weight–Culmen con Epsilon: 0.0**



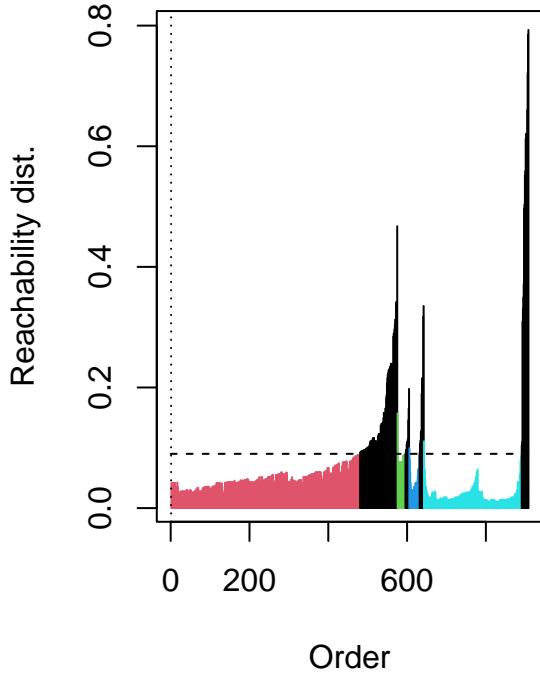
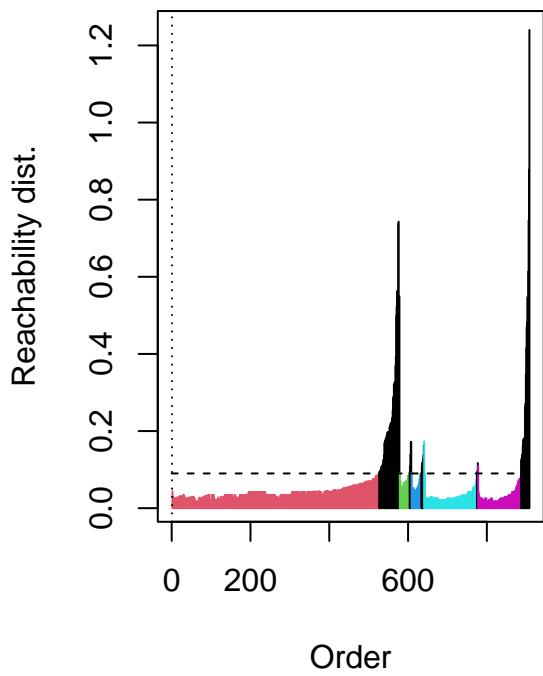
```
##  
## Este es el valor de i: 0.08
```

**Wing–Culmen con Epsilon: 0.09**    **Weight–Culmen con Epsilon: 0.09**



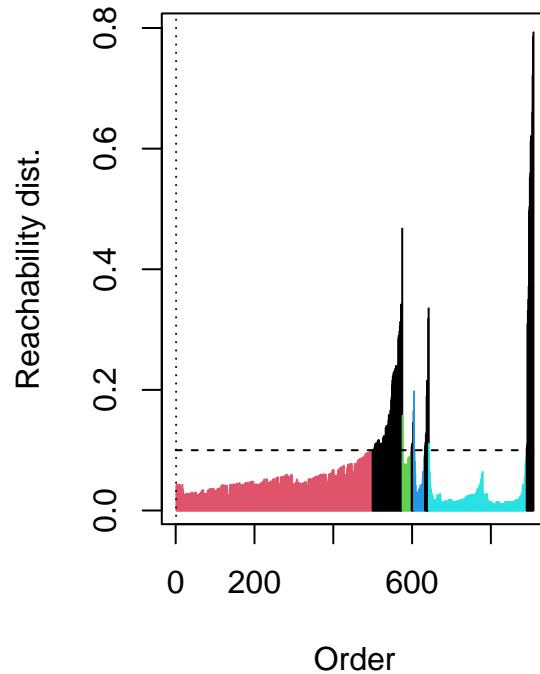
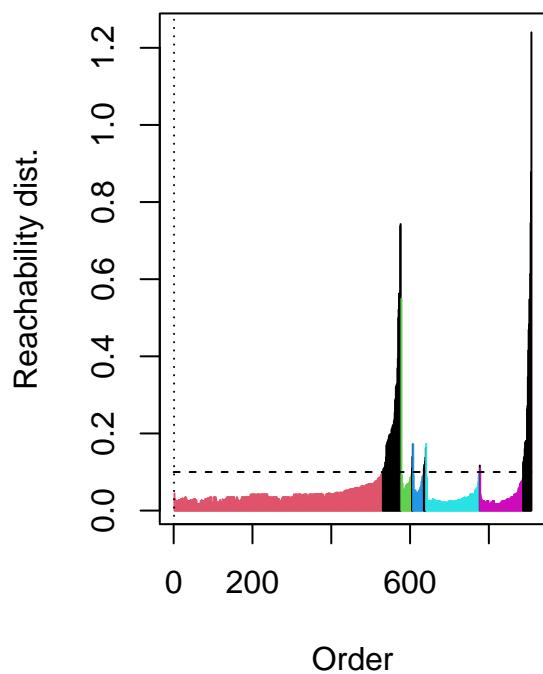
```
##  
## Este es el valor de i: 0.09
```

**Wing–Culmen con Epsilon: 0.09**    **Weight–Culmen con Epsilon: 0.09**



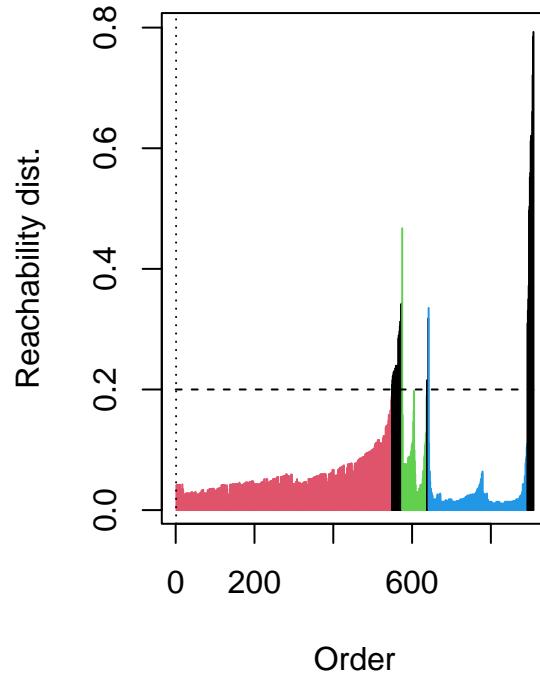
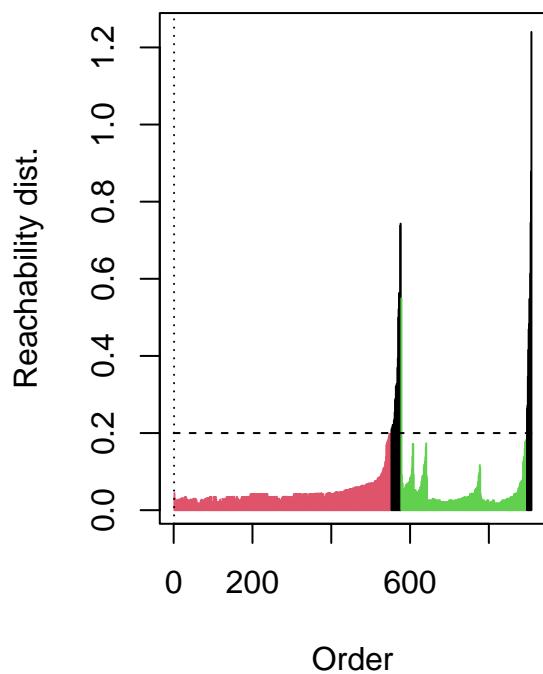
```
##  
## Este es el valor de i: 0.1
```

**Wing–Culmen con Epsilon: 0.1      Weight–Culmen con Epsilon: 0.**



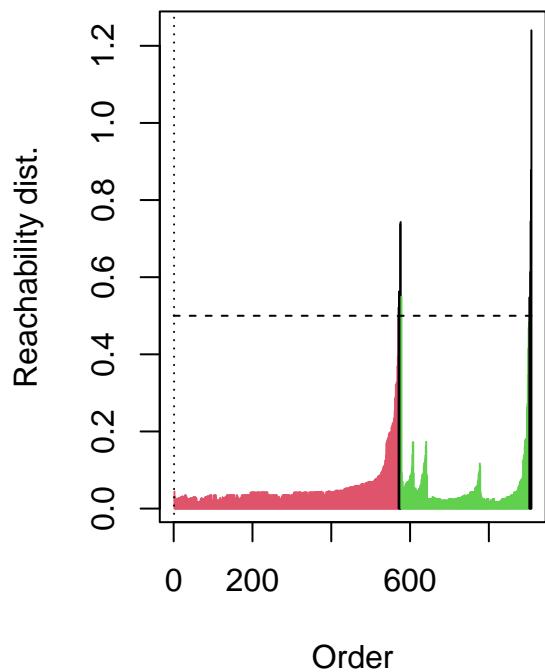
```
##  
## Este es el valor de i: 0.2
```

**Wing–Culmen con Epsilon: 0.2      Weight–Culmen con Epsilon: 0.**

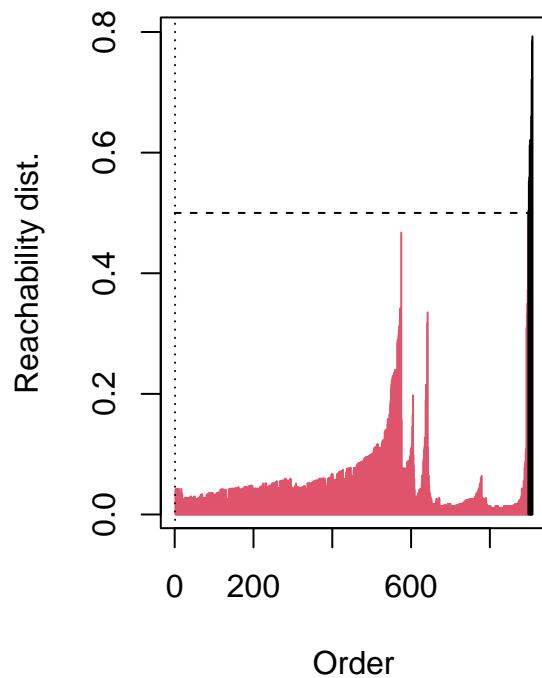


```
##  
## Este es el valor de i: 0.5
```

**Wing–Culmen con Epsilon: 0.5**

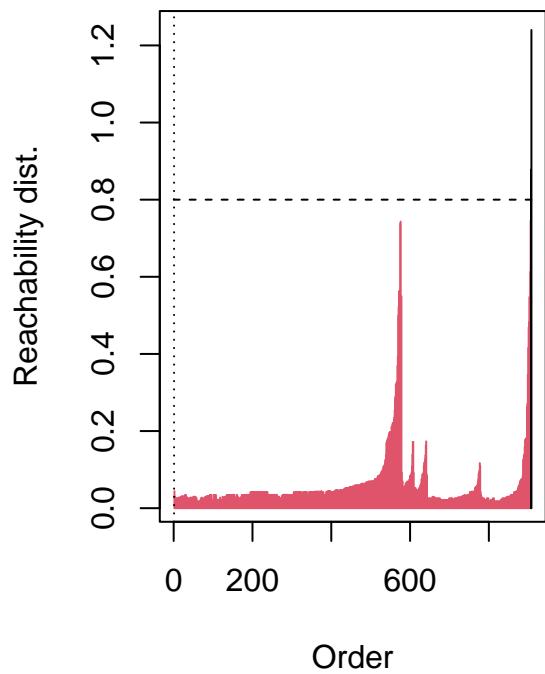


**Weight–Culmen con Epsilon: 0.**

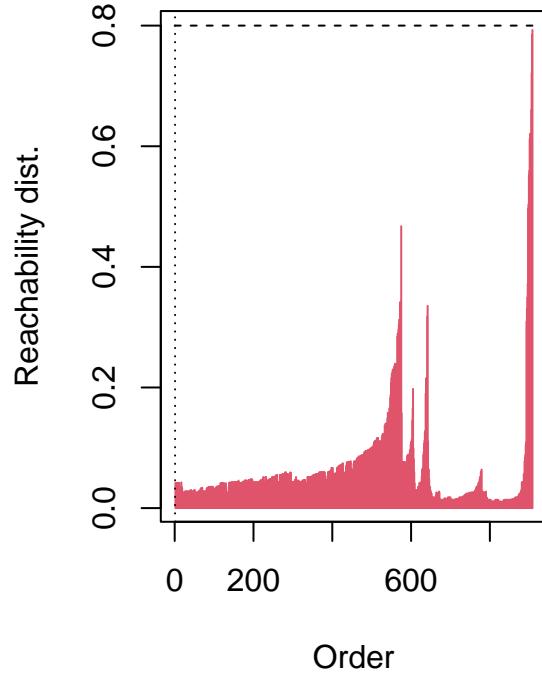


```
##  
## Este es el valor de i:  0.8
```

**Wing–Culmen con Epsilon: 0.8**

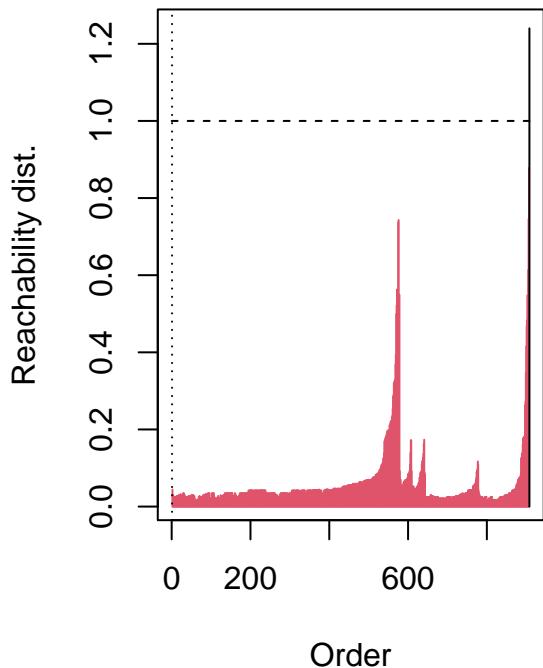


**Weight–Culmen con Epsilon: 0.**

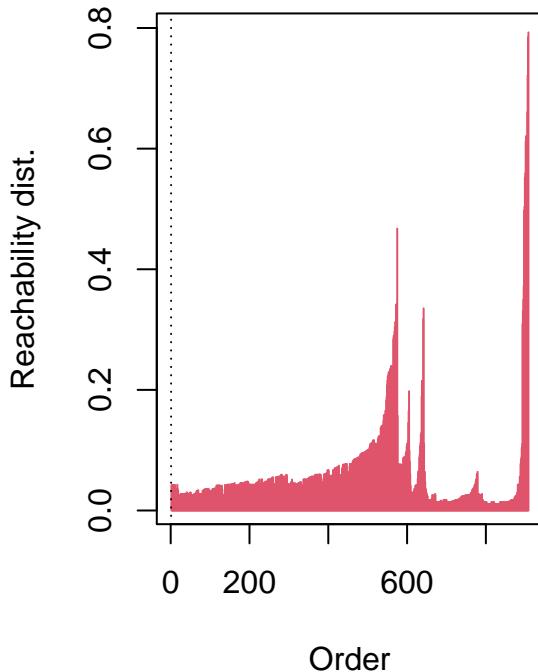


```
##  
## Este es el valor de i:  1
```

**Wing–Culmen con Epsilon: 1**



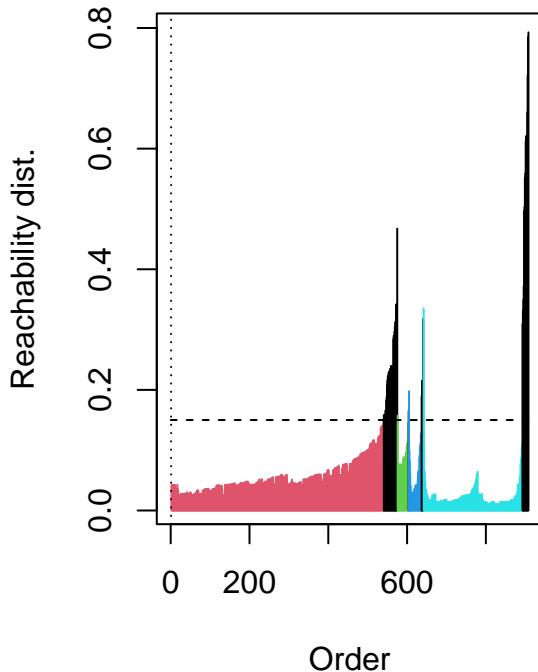
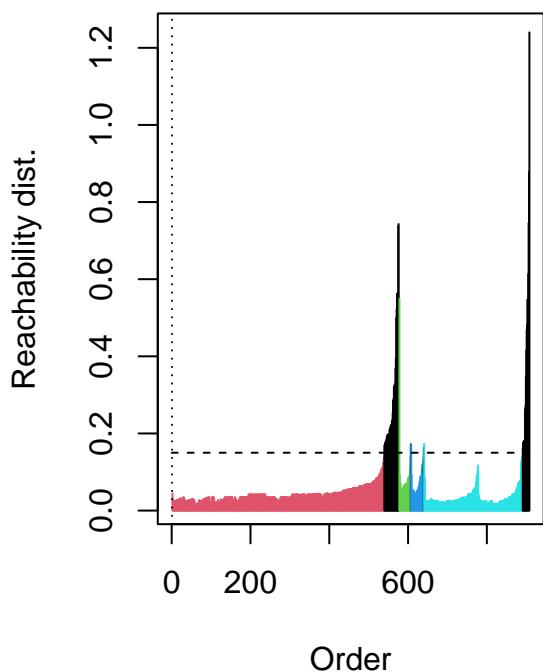
**Weight–Culmen con Epsilon: 1**



Como se ve por los resultados de arriba, parece que para  $\varepsilon = 0.2$  se obtienen exactamente tres clústers para el par Weight-Culmen, mientras que para el mismo valor de  $\varepsilon$  en el par Wing-Culmen solo se observan dos clústers. Se va a probar a continuación un valor medio entre 0.1 y 0.2 para ver si conseguimos que en el par Wing-Culmen aparezcan también tres clústers (queremos tres clústers, porque hay tres tipos de halcones), véase el siguiente chunk de código:

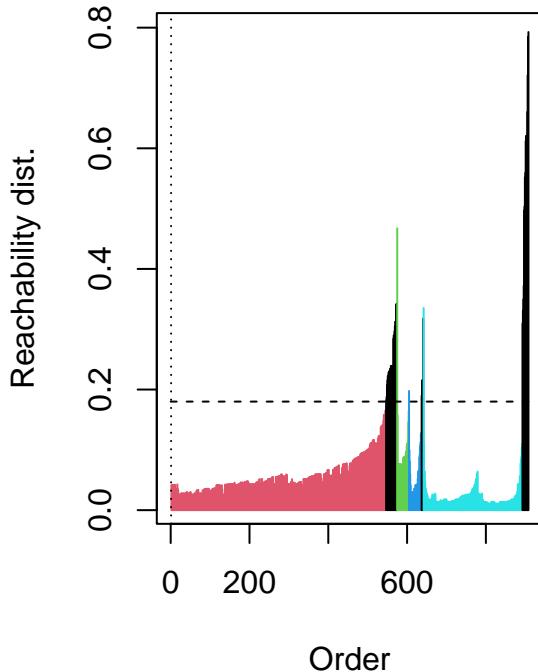
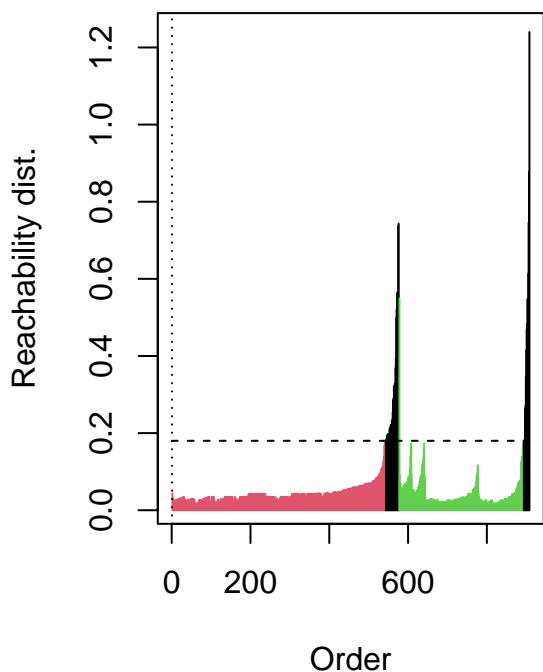
```
set.seed(1)
par(mfrow = c(1, 2))
resultatWingCulmen15 <- extractDBSCAN(observacionesWingCulmen, eps_cl = 0.15)
resultatWeightCulmen15 <- extractDBSCAN(observacionesWeightCulmen, eps_cl = 0.15)
plot(resultatWingCulmen15, main = paste("Wing–Culmen con Epsilon: ",0.15))
plot(resultatWeightCulmen15, main = paste("Weight–Culmen con Epsilon: ",0.15))
```

## Wing–Culmen con Epsilon: 0.1! Weight–Culmen con Epsilon: 0.1



```
resultatWingCulmen18 <- extractDBSCAN(observacionesWingCulmen, eps_cl = 0.18)
resultatWeightCulmen18 <- extractDBSCAN(observacionesWeightCulmen, eps_cl = 0.18)
plot(resultatWingCulmen18, main = paste("Wing–Culmen con Epsilon: ",0.18))
plot(resultatWeightCulmen18, main = paste("Weight–Culmen con Epsilon: ",0.18))
```

## Wing–Culmen con Epsilon: 0.1! Weight–Culmen con Epsilon: 0.1



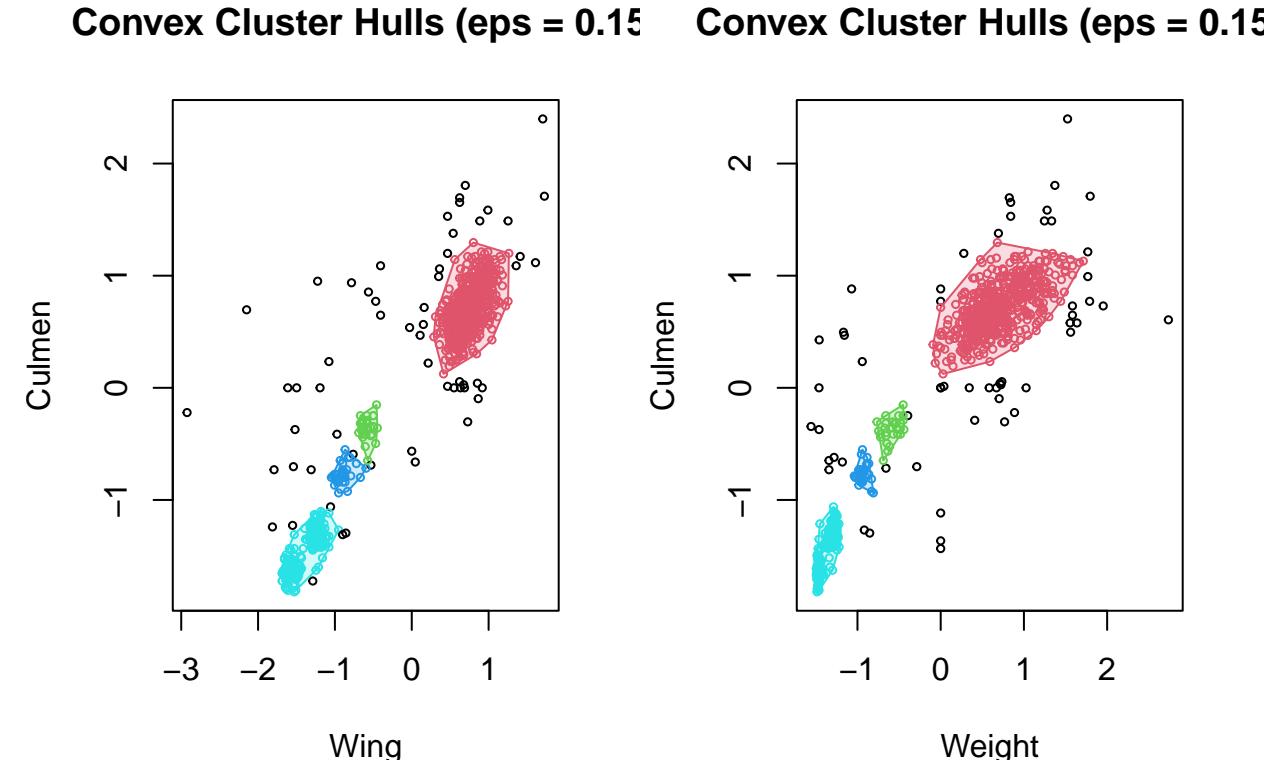
Curiosamente, para  $\varepsilon = 0.15$  se obtienen 4 clústeres para los dos pares de atributos. Pero cuando

$\varepsilon = 0.18$  en el par de atributos Wing-Culmen se observan solo dos clústeres, mientras que el par de atributos Weight-Culmen permanece intacto. Esto tiene una explicación, y es que al aumentar el valor de  $\varepsilon = 0.15$  a  $\varepsilon = 0.18$ , lo que se ha hecho ha sido ampliar el rango de conexión entre las muestras, por lo tanto, como en el par Wing-Culmen, la altura media de alcanzabilidad de entre los 5 picos que se observan es menor que la altura media de alcanzabilidad en el par Weight-Culmen, al aumentar  $\varepsilon$  se acaba forzando a que las muestras de diferentes clústeres acaben bajo clasificadas bajo el mismo clúster. Por eso Weight-Culmen no es tan sensible al aumento de  $\varepsilon$  porque la distancia de alcanzabilidad es mayor, y por ende hay una mayor separación entre muestras de distintos clústeres, e incluso a veces dentro del mismo clúster.

A continuación, se continua con el análisis de los datos, pero esta vez mediante una representación un tanto más intuitiva en comparación con la anterior, ya que permite visualizar las muestras de datos, al mismo tiempo que se visualiza los clústeres mediante formas convexas.

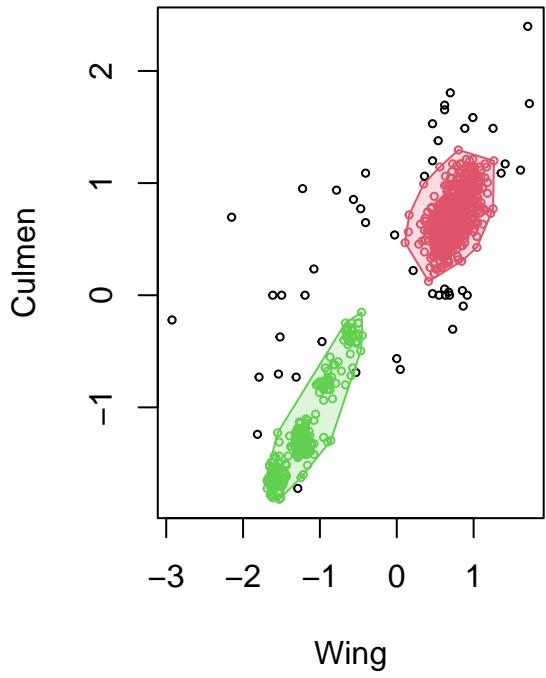
```
set.seed(1)
par(mfrow = c(1, 2))

#Primero para epsilon = 0.15
hullplot(dfHawks2Normalizado[c("Wing", "Culmen")], resultatWingCulmen15, main = "Convex Cluster Hulls (eps = 0.15)
hullplot(dfHawks2Normalizado[c("Weight", "Culmen")], resultatWeightCulmen15, main = "Convex Cluster Hulls (eps = 0.15)
```

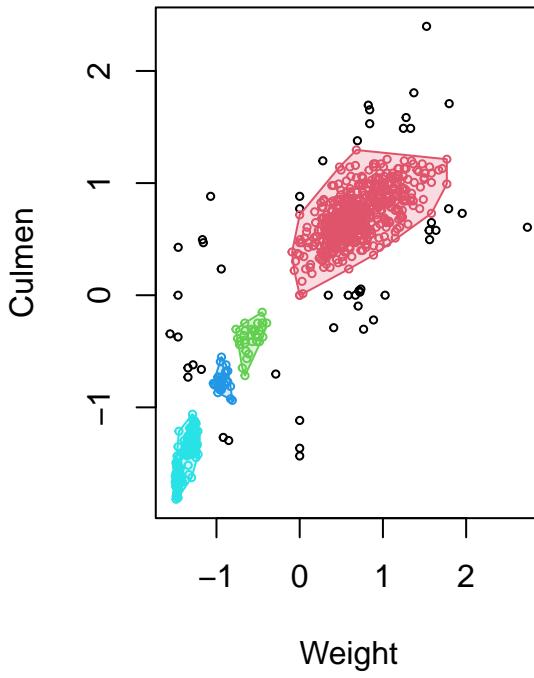


```
#Seguidamente para epsilon = 0.18
hullplot(dfHawks2Normalizado[c("Wing", "Culmen")], resultatWingCulmen18, main = "Convex Cluster Hulls (eps = 0.18)
hullplot(dfHawks2Normalizado[c("Weight", "Culmen")], resultatWeightCulmen18, main = "Convex Cluster Hulls (eps = 0.18)
```

## Convex Cluster Hulls ( $\text{eps} = 0.18$ )



## Convex Cluster Hulls ( $\text{eps} = 0.18$ )

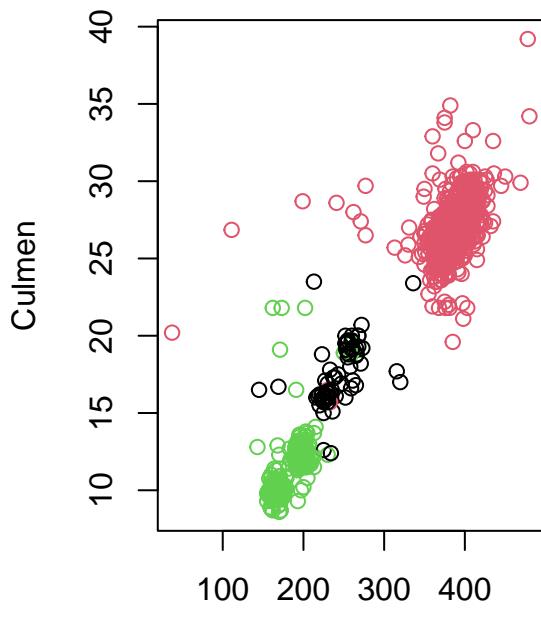


Los resultados que se ven arriba se corresponden con  $\varepsilon = 0.18$ , por ello se identifican solo 2 clústeres en el par Wing-Culmen y 4 en el par Weight-Culmen. Se va a dejar de lado la clasificación llevada a cabo para  $\varepsilon = 0.18$ , ya que esta solo contempla dos clústeres en el par Wing-Culmen. A continuación se va a comparar el proceso de clasificación de Wing-Culmen con la clasificación real (mostrada en el ejercicio anterior) con el fin de ver si al menos se detectan 3 de 4 clústeres correctamente.

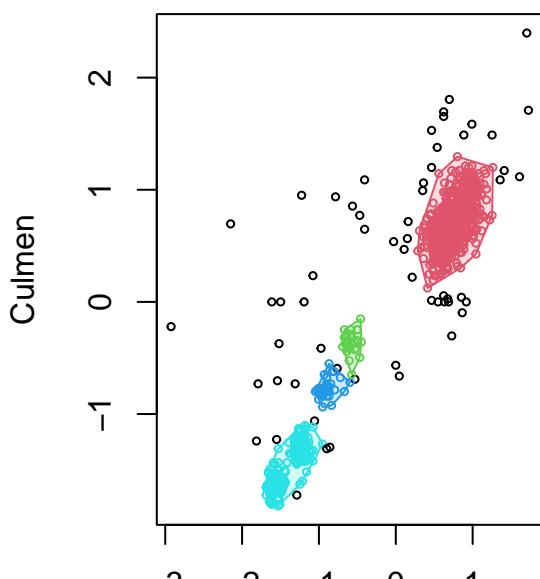
Véase a continuación la clasificación real y la clasificación anterior:

```
par(mfrow = c(1, 2))
#Clasificación real
plot(Hawks2[c("Wing", "Culmen")], col=as.factor(Hawks$Species), main="Clasificación real")
hullplot(dfHawks2Normalizado[c("Wing", "Culmen")], resultatWingCulmen15, main = "Convex Cluster Hulls (eps = 0.18)")
```

**Clasificación real**



**Convex Cluster Hulls (eps = 0.15)**

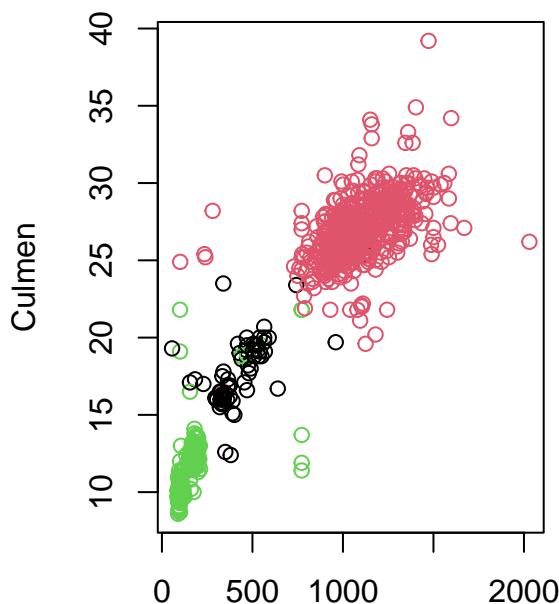


Wing

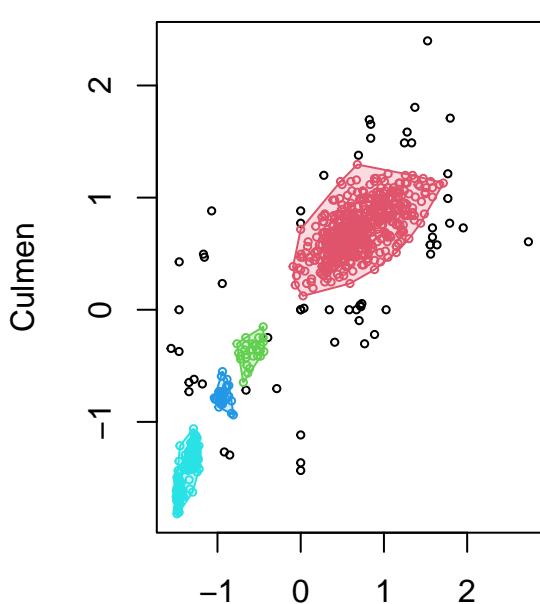
Wing

```
#Primero para epsilon = 0.15
plot(Hawks2[c("Weight","Culmen")], col=as.factor(Hawks$Species), main="Clasificación real")
hullplot(dfHawks2Normalizado[c("Weight","Culmen")],resultatWeightCulmen15, main = "Convex Cluster Hulls")
```

**Clasificación real**



**Convex Cluster Hulls (eps = 0.15)**



Weight

Weight

Para el par Wing-Culmen, se puede comprobar y se observa, como el clúster rojo auna gran parte

de las muestras, para ambos pares de atributos. No cabe ninguna duda que el DBSCAN no tiene problema con ese subconjunto de muestras, luego, para el clúster negro en la clasificación real se ve como el algoritmo lo divide en dos (azul y verde), esto aplica a los dos pares de atributos. Por último, el último clúster, el verde, en la clasificación real, se corresponde casi en su totalidad al de color cyan en el algoritmo DBSCAN, esto aplica para los dos pares de atributos. En definitiva, parece que DBSCAN hace un buen trabajo, pero para clústeres que están muy juntos y donde la distancia de alcanzabilidad entre muestras de distinto clúster es pequeña, el algoritmo no logra particionar bien los dos conjuntos de muestras. Esto último puede remediararse, como se ha visto en este ejemplo y en el anterior, disminuyendo  $\varepsilon$  para evitar que el rango de conexión entre puntos que realmente pertenecen a clústeres/clases diferentes, aumente. Esto último provocará la creación de clústeres nuevos, pero desde mi punto de vista, y retomando un ejemplo de la teoría del tema anterior, “es mejor que una enfermedad que es mortal, cuando se clasifica erróneamente, que esta se clasifique con otra etiqueta nueva que como no mortal”

Siguiendo el ejemplo guiado, ahora se debería de repetir el experimento, pero modificando el valor de  $\varepsilon_{cls}$ . No obstante, esto ya se ha hecho en los ejemplos anteriores, incluso en el propio ejemplo anterior se explica la influencia de aumentar/disminuir este parámetro.

Ahora se va a estudiar el funcionamiento de una variante del algoritmo **DBSCAN**, en la cual se va a poder especificar el parámetro  $\xi$  (intuyo que de la letra griega  $\xi$ ). Este parámetro va a permitir clasificar los clústeres en función de la variación en la densidad relativa de estos. Véase el siguiente chunk de código:

```
set.seed(1)
par(mfrow = c(1, 2))
resultatWingCulmen05Xi <- extractXi(observacionesWingCulmen, xi = 0.05)
resultatWingCulmen05Xi

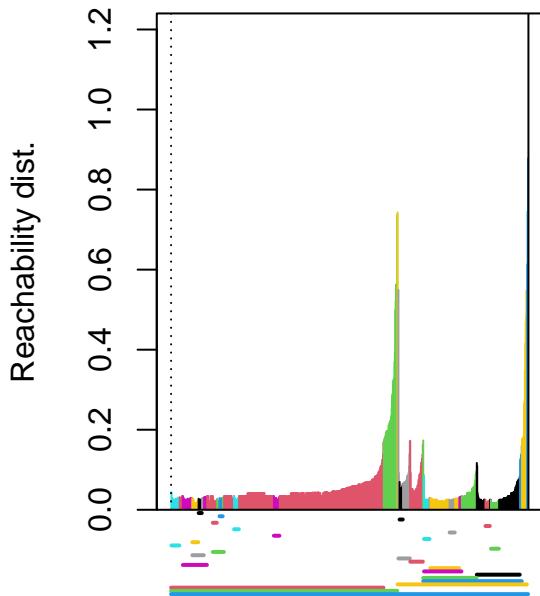
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.69370507620783, eps_cl = NA, xi = 0.05
## The clustering contains 26 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi, clusters_xi, cluster

resultatWeightCulmen05Xi <- extractXi(observacionesWeightCulmen, xi = 0.05)
resultatWeightCulmen05Xi

## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.15943872924627, eps_cl = NA, xi = 0.05
## The clustering contains 22 cluster(s) and 0 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi, clusters_xi, cluster

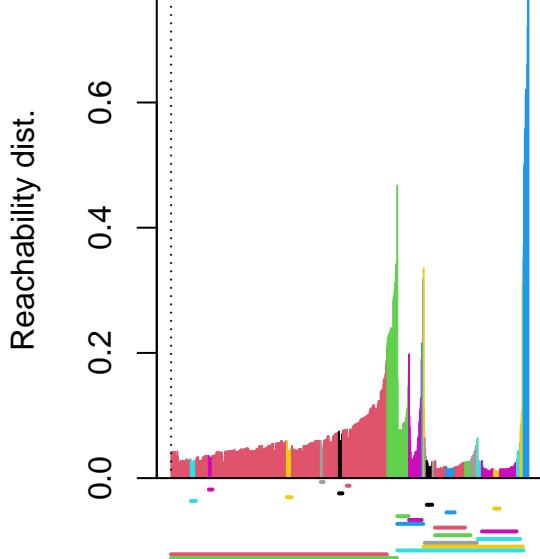
plot(resultatWingCulmen05Xi, main = paste("Wing-Culmen con xi: ", 0.05))
```

## Wing–Culmen con xi: 0.05



```
plot(resultatWeightCulmen05Xi, main = paste("Weight–Culmen con xi: ",0.05))
```

## Weight–Culmen con xi: 0.05



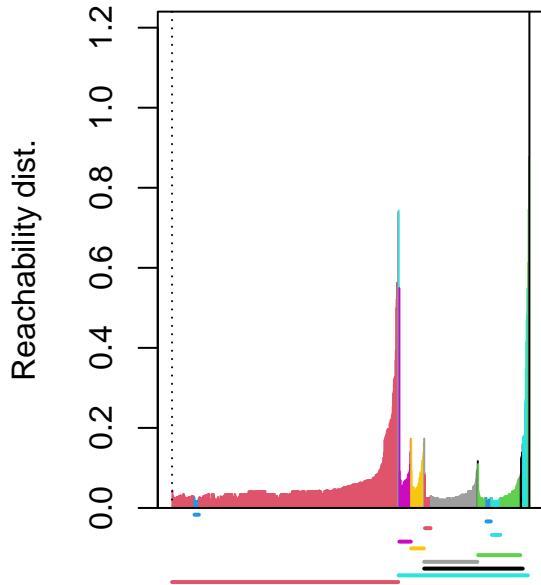
```
resultatWingCulmen1Xi <- extractXi(observacionesWingCulmen, xi = 0.1)
resultatWingCulmen1Xi
```

```
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.69370507620783, eps_cl = NA, xi = 0.1
## The clustering contains 12 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi, clusters_xi, cluster
```

```
resultatWeightCulmen1Xi <- extractXi(observacionesWeightCulmen, xi = 0.1)
resultatWeightCulmen1Xi
```

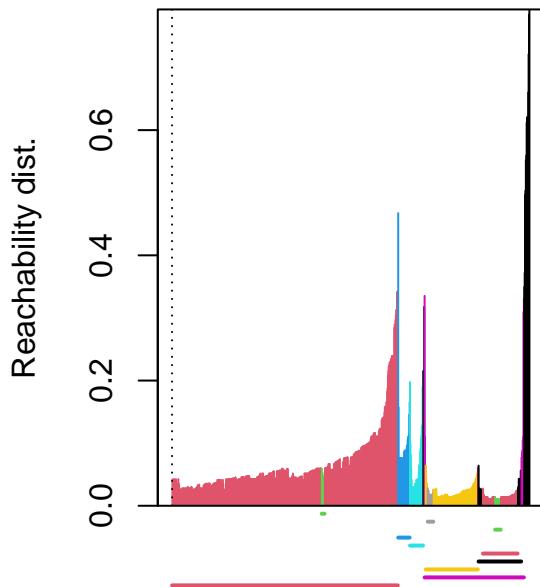
```
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.15943872924627, eps_cl = NA, xi = 0.1
## The clustering contains 10 cluster(s) and 17 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi, clusters_xi, cluster
plot(resultatWingCulmen1Xi, main = paste("Wing-Culmen con xi: ",0.1))
```

## Wing–Culmen con xi: 0.1



```
plot(resultatWeightCulmen1Xi, main = paste("Weight-Culmen con xi: ",0.1))
```

## Weight-Culmen con xi: 0.1



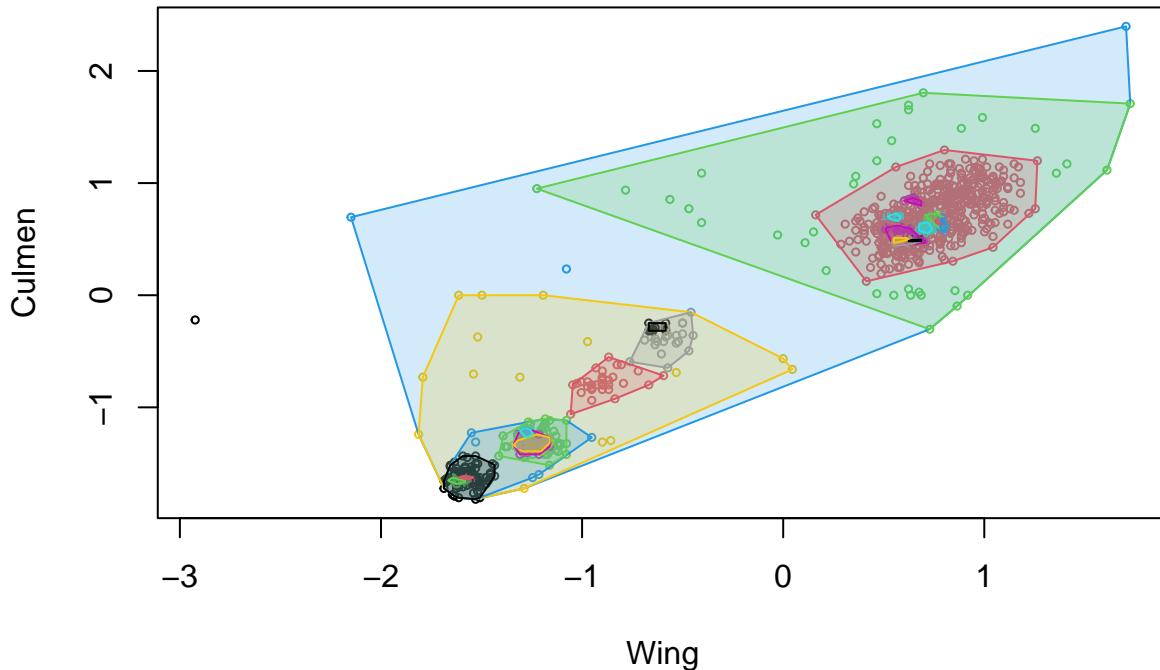
Como se puede comprobar, el algoritmo “detecta” una gran cantidad de clústeres, algo que no se corresponde con la clasificación real de los datos, por lo tanto, más tarde se va a probar a modificar el valor de  $\xi$ , pero antes explotamos todas las posibilidades en cuanto a representación de datos. Ahora bien, la creación de nuevos clústeres nos podría servir para otros proyectos de minería de datos, como por ejemplo encontrar patrones de comportamiento dentro de cada uno de los clústeres, etc.

A continuación se van a representar los clústeres con forma convexa:

```
#Primero para xi = 0.05
hullplot(dfHawks2Normalizado[c("Wing", "Culmen")], resultatWingCulmen05Xi, main = "Convex Cluster Hulls (xi=0.05)")

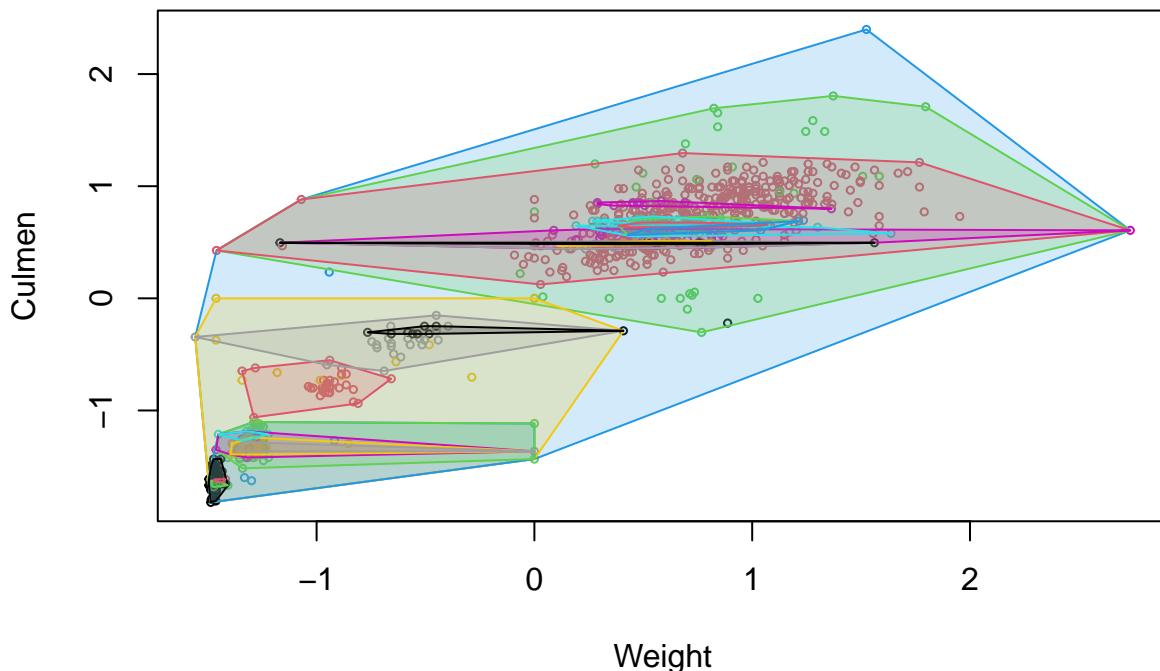
## Warning in hullplot(dfHawks2Normalizado[c("Wing", "Culmen")]),
## resultatWingCulmen05Xi, : Not enough colors. Some colors will be reused.
```

### Convex Cluster Hulls ( $\xi = 0.05$ )



```
hullplot(dfHawks2Normalizado[c("Weight", "Culmen")], resultatWingCulmen05Xi, main = "Convex Cluster Hulls  
## Warning in hullplot(dfHawks2Normalizado[c("Weight", "Culmen")],  
## resultatWingCulmen05Xi, : Not enough colors. Some colors will be reused.
```

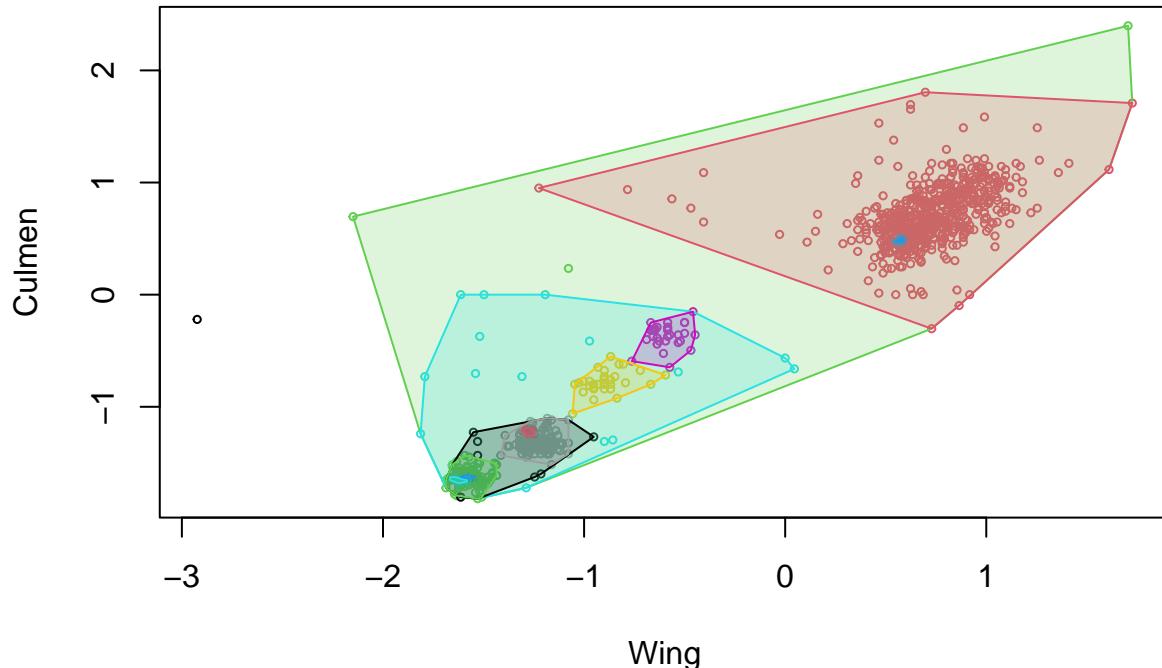
### Convex Cluster Hulls ( $\xi = 0.05$ )



```
#Seguidamente para xi = 0.1
hullplot(dfHawks2Normalizado[c("Wing", "Culmen")], resultatWingCulmen1Xi, main = "Convex Cluster Hulls (xi = 0.1)")

## Warning in hullplot(dfHawks2Normalizado[c("Wing", "Culmen")]),
## resultatWingCulmen1Xi, : Not enough colors. Some colors will be reused.
```

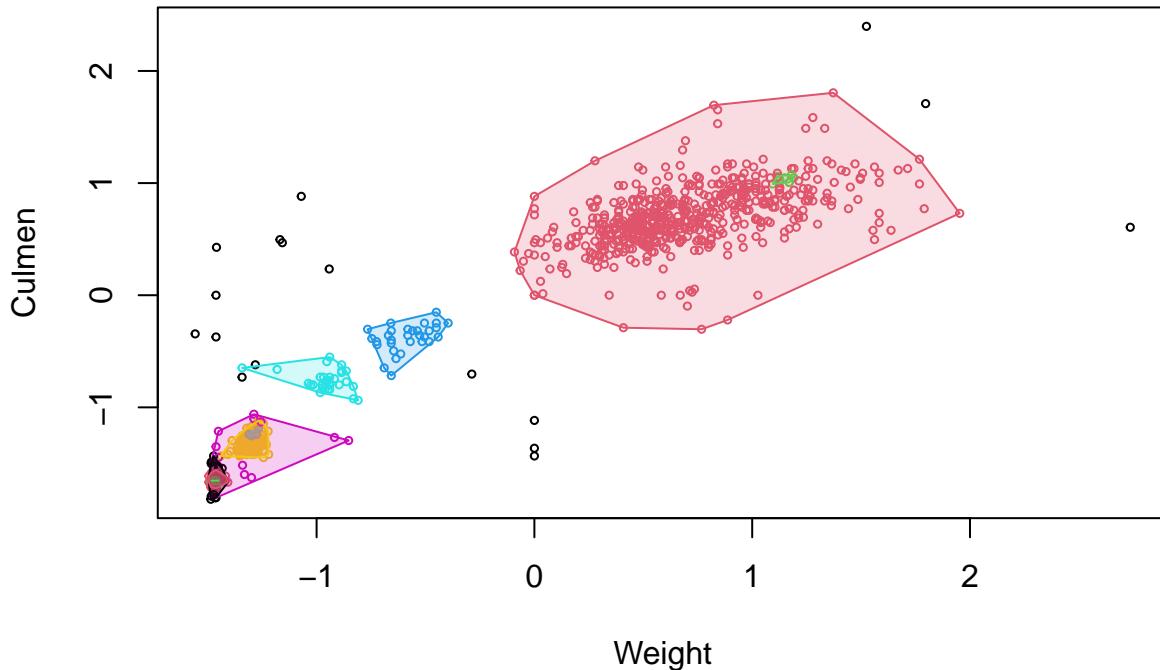
## Convex Cluster Hulls (xi = 0.1)



```
hullplot(dfHawks2Normalizado[c("Weight", "Culmen")], resultatWeightCulmen1Xi, main = "Convex Cluster Hulls (xi = 0.1)")

## Warning in hullplot(dfHawks2Normalizado[c("Weight", "Culmen")]),
## resultatWeightCulmen1Xi, : Not enough colors. Some colors will be reused.
```

## Convex Cluster Hulls ( $\xi = 0.1$ )



Como se puede observar por los propios gráficos y por el aviso que lanza al terminar de representar los clústeres, hay demasiados clústeres. Esto se debe a que esta variable del DBSCAN, clasifica las muestras en función de la densidad relativa de los clústeres. No obstante, para este caso, esto no funciona, ya que dos de los clústeres están muy juntos. No obstante se va a probar a aumentar el valor de  $\xi$  para que las muestras más distantes entre sí, queden bajo el mismo clúster, y así evitar que se formen tantos clústeres. Véase el siguiente chunk de código:

```
set.seed(1)
par(mfrow = c(1, 2))
resultatWingCulmen02Xi <- extractXi(observacionesWingCulmen, xi = 0.2)
resultatWingCulmen02Xi

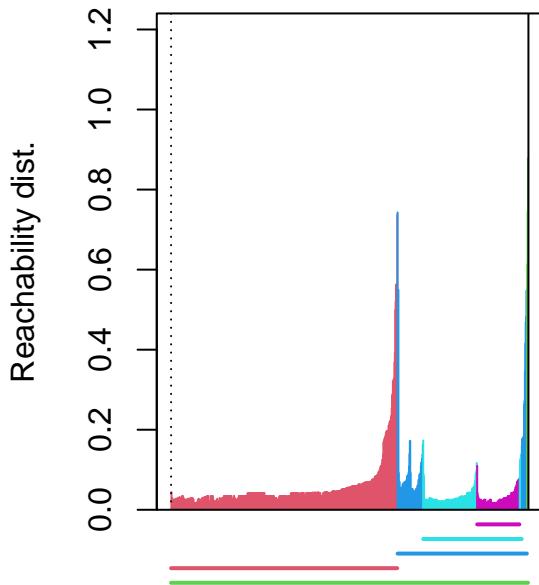
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.69370507620783, eps_cl = NA, xi = 0.2
## The clustering contains 5 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi, clusters_xi, cluster

resultatWeightCulmen02Xi <- extractXi(observacionesWeightCulmen, xi = 0.2)
resultatWeightCulmen02Xi

## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.15943872924627, eps_cl = NA, xi = 0.2
## The clustering contains 4 cluster(s) and 621 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi, clusters_xi, cluster

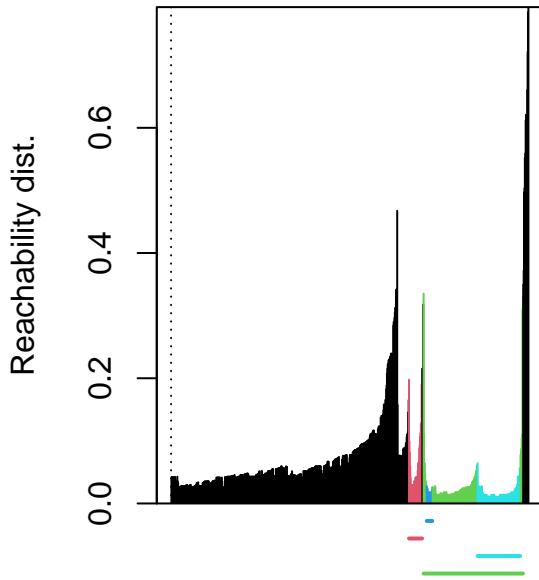
plot(resultatWingCulmen02Xi, main = paste("Wing-Culmen con xi: ", 0.2))
```

## Wing–Culmen con xi: 0.2



```
plot(resultatWeightCulmen02Xi, main = paste("Weight-Culmen con xi: ",0.2))
```

## Weight–Culmen con xi: 0.2



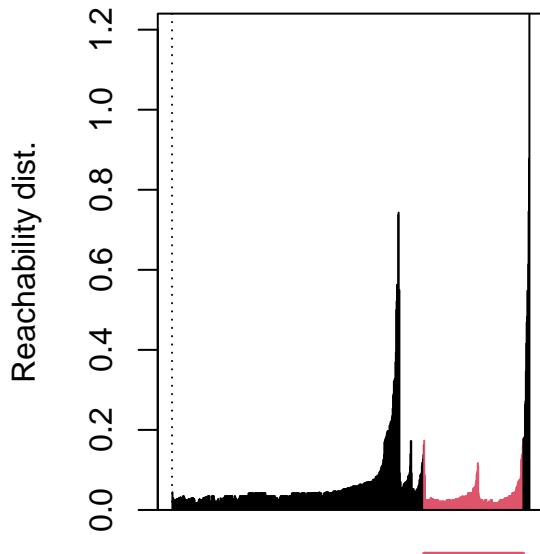
```
resultatWingCulmen03Xi <- extractXi(observacionesWingCulmen, xi = 0.3)  
resultatWingCulmen03Xi
```

```
## OPTICS ordering/clustering for 908 objects.  
## Parameters: minPts = 8, eps = 1.69370507620783, eps_cl = NA, xi = 0.3  
## The clustering contains 1 cluster(s) and 656 noise points.  
##  
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,  
##                   eps_cl, xi, clusters_xi, cluster
```

```
resultatWeightCulmen03Xi <- extractXi(observacionesWeightCulmen, xi = 0.3)
resultatWeightCulmen03Xi
```

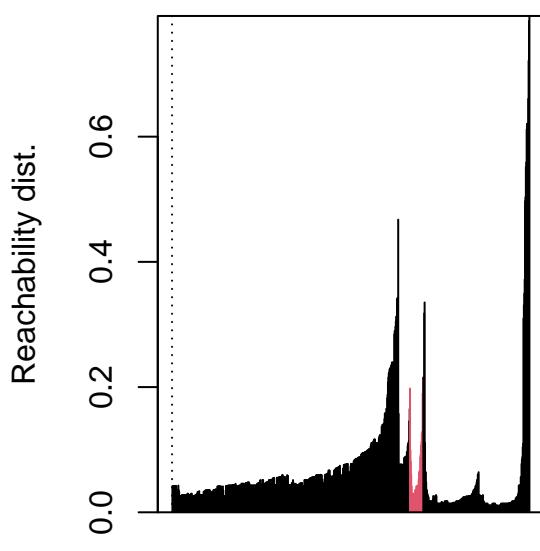
```
## OPTICS ordering/clustering for 908 objects.
## Parameters: minPts = 8, eps = 1.15943872924627, eps_cl = NA, xi = 0.3
## The clustering contains 1 cluster(s) and 874 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                    eps_cl, xi, clusters_xi, cluster
plot(resultatWingCulmen03Xi, main = paste("Wing-Culmen con xi: ",0.3))
```

### **Wing–Culmen con xi: 0.3**



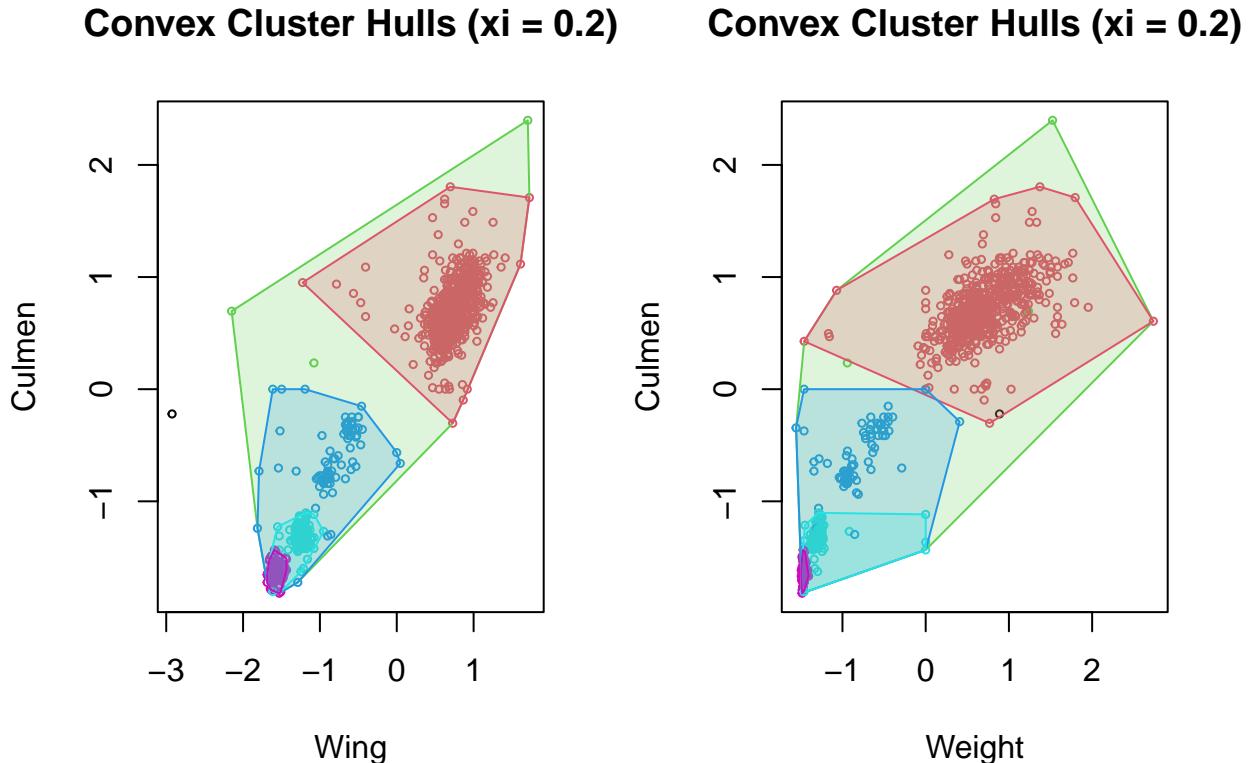
```
plot(resultatWeightCulmen03Xi, main = paste("Weight-Culmen con xi: ",0.3))
```

### **Weight–Culmen con xi: 0.3**



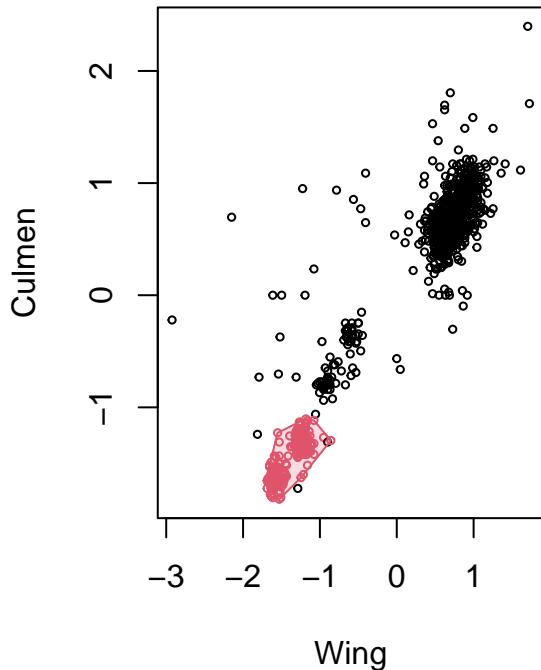
Como se puede observar, por los resultados de arriba, para  $\xi = 0.2$  se obtiene un mejor resultado, ya que el número de clústers que se obtiene es menor. A continuación se va a representar las muestras reales con cada uno de los clústers con forma convexa.

```
par(mfrow = c(1, 2))
#Primero para xi = 0.2
hullplot(dfHawks2Normalizado[c("Wing", "Culmen")], resultatWingCulmen02Xi, main = "Convex Cluster Hulls (xi = 0.2)
hullplot(dfHawks2Normalizado[c("Weight", "Culmen")], resultatWeightCulmen02Xi, main = "Convex Cluster Hulls (xi = 0.2)
```

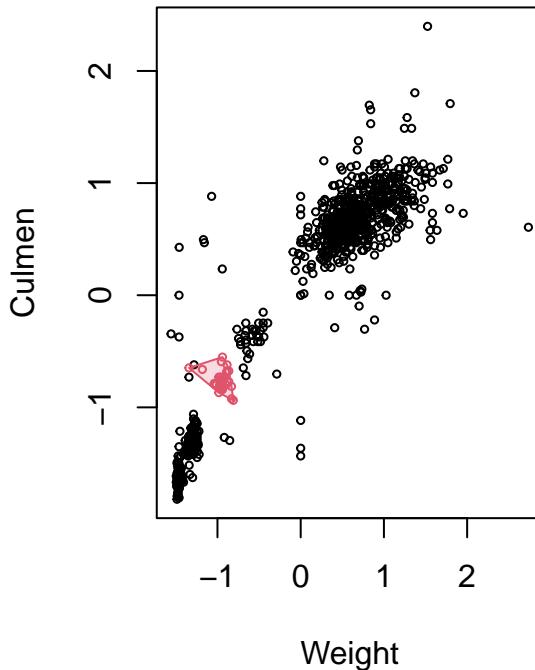


```
#Seguidamente para xi = 0.3
hullplot(dfHawks2Normalizado[c("Wing", "Culmen")], resultatWingCulmen03Xi, main = "Convex Cluster Hulls (xi = 0.3)
hullplot(dfHawks2Normalizado[c("Weight", "Culmen")], resultatWeightCulmen03Xi, main = "Convex Cluster Hulls (xi = 0.3)
```

**Convex Cluster Hulls ( $\xi = 0.3$ )**



**Convex Cluster Hulls ( $\xi = 0.3$ )**



Como se puede apreciar, se obtienen mejores resultados para  $\xi = 0.2$  que para  $\xi = 0.3$ , pues a partir de este último valor, solo se agrupan muestras en un clúster, y el resto son consideradas como ruido, pues están en negro.

### Ejercicio 3

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

#### Respuesta 3

Antes de discutir los resultados de los diferentes experimentos que se han llevado a cabo, primero se va a hacer una pequeña revisión teórica de estos dos algoritmos, para destacar los puntos que los caracterizan, no obstante, entre las explicaciones, se hará referencia a algunos de los resultados obtenidos, en caso de que consigan demostrar las propiedades teóricas de estos algoritmos.

Por teoría, se sabe que el algoritmo de **k-means** divide las muestras en  $k$  clústeres (dónde  $k$  ha de ser predefinida, lo cuál es un punto en contra de este algoritmo). Como este algoritmo es más antiguo que el de **DBSCAN**, el primero asume que todos los clústeres son esféricos, para todos los juegos de datos que se le introduzcan, también asume que todos los clústeres son del mismo tamaño (aproximadamente). Estos clústeres vienen caracterizados por sus centroides correspondientes, fruto de la media de todas las posiciones de las muestras dentro del clúster. Un punto en contra, aparte de las suposiciones esféricas que lleva a cabo, es que, **k-means** no se comporta muy bien cuando las muestras de diferentes clusteres se encuentran próximas entre sí, esto es exactamente lo que se ha visto en el primer ejercicio de este proyecto, donde muestras de clústeres determinados, son clasificadas en el clúster equivocado de manera sistemática, a pesar de cambiar el valor de  $k$ . Por último, este algoritmo es susceptible a los centroides iniciales, a partir de los cuales calculará el resto, por lo tanto los resultados dependerán en gran medida de las condiciones iniciales, y como no, del número de  $k$  especificado.

A diferencia del anterior algoritmo, el **DBSCAN** centra su dinámica en la densidad de los clústeres, y no requiere que se le predetermine un número de clústeres, lo único que necesita, es que haya áreas de muestras, de baja densidad, para así poder determinar bien las fronteras

entre los diferentes clústeres. La gran diferencia de este algoritmo frente al anterior reside en las suposiciones relativas a las formas geométricas de los clústeres, ya que este algoritmo no supone que todos los clústeres sean esféricos, de hecho una ventaja que tiene **DBSCAN** es que puede identificar clústeres de diferentes formas, por eso en las últimas figuras veíamos clústeres de naturaleza convexa, e incluso clústeres más pequeños dentro de clústeres más grandes, una dinámica que resultaría imposible en el algoritmo de los **k-means**. El **DBSCAN** introduce los conceptos de puntos núcleos, y las distancias de alcanzabilidad, como propiedades imprescindibles para su funcionamiento, y que por ello lo caracterizan.

Si bien el **DBSCAN** no requiere de información acerca del número de clústeres, para sus cálculos basados en las densidades relativas de los clústeres, si que necesita el número mínimo de puntos que tiene que haber a una mínima distancia de alcanzabilidad ( $\epsilon$ ). A partir de estos valores y de su posterior modificación para la obtención de mejores resultados, estos conseguirán arrojar mejores resultados o incluso conocimiento escondido entre los datos y que hasta entonces permanecía desconocido. Por último, el algoritmo de **DBSCAN**, debido a que no supone una forma geométrica determinada de los clústeres, es capaz de lidiar con valores atípicos, y en cierto modo, es más inteligente que el anterior, ya que a estos los considera como ruido, evitando así que estos queden clasificados en un clúster erróneo.

Los aspectos compartidos entre estos dos algoritmos son los relativos a los constantes cálculos, que involucran distancias, a fin de poder clasificar las muestras dentro de clústeres.

Ahora, poniendo el foco en los resultados, se ha podido constatar, que aunque el algoritmo de **k-means** conseguía identificar los clústeres más voluminosos y separados, este solía particionar o fusionar los clústeres que estaban más juntos. Este problema se debe a las cortas distancias que a veces existía entre clústeres, haciendo la tarea de clasificación más difícil. A esto hay que sumarle el handicap de que todos los clústeres detectados, solían tener el mismo área y forma geométrica, algo que ha penalizado los resultados a pesar de haber elegido el valor de **k** más adecuado, gracias a la métrica **Silhouette**. No obstante, el problema de las distancias también ha perjudicado a las tareas de clasificación llevadas a cabo por el algoritmo **DBSCAN**, con la diferencia de que este ha obtenido mejores resultados de clasificación a costa de aumentar de sus parámetros iniciales como  $\epsilon$  o  $\xi$  permitiendo que haya menos clústeres, pues las muestras de los ya existentes, extendían su rango de alcanzabilidad, mediante este aumento de  $\epsilon$  o  $\xi$ .

En definitiva, ha sido un proyecto muy interesante, puesto que entre otras cosas como las mencionadas antes, ha puesto de manifiesto la importancia de la normalización de los datos, cuando estos se encuentran en escalas radicalmente diferentes y cuando existe un comportamiento lineal entre ellos. Se ha visto claramente, como la normalización ha conseguido salvar el proyecto de obtener resultados erróneos, y aunque me he dado cuenta a mitad del trayecto, de que tenía que normalizar los datos, antes de aplicar los distintos algoritmos, la comparación de los resultados me ha hecho darme cuenta de la importancia de esta técnica de procesado de datos. Además, se ha conseguido detectar un problema común, que los dos algoritmos han sabido afrontar en mejor o peor medida, y es la corta distancia entre clústeres, se podría decir que este ha sido el principal obstáculo de la práctica. Ha sido una práctica interesante y enriquecedora que me ha requerido recordar teoría del tema pasado aplicando algoritmos de este tema.