Minería de datos: PEC2

Autor: Jesús González Leal

Abril 2021

Table of Contents

Introducción 3
Presentación 3
Competencias 3
Objetivos 3
Descripción de la PEC a realizar 3
Recursos 3
Formato y fecha de entrega 4
Nota: Propiedad intelectual 4
Ejercicio 1 4
Enunciado 4
Respuesta ejercicio 1 4
Ejercicio 2 5
Enunciado 5
Respuesta ejercicio 2 6
Ejercicio 3 6
Enunciado 6
Ejemplo 1: Métodos de agregación con datos autogenerados 6
Ejemplo 2: Métodos de agregación con datos reales 14
Respuesta ejercicio 3 20
Exploración de datos 20
Escalado 24
Clusterización 25
Ejercicio 4 29
Enunciado 29
Ejemplo 3: Métodos de generación de reglas de asociación 30
Respuesta ejercicio 4 33

Carga de datos 33
Exploración de datos 33
Aplicando Reglas de Asociación 35
Ejercicio 5 39
Enunciado 39
Respuesta ejercicio 5 39
Agrupamiento jerárquico 39
Clusterización mediante K-menoids (PAM) 42
Clusterización mediante CLARA 46
Clusterización mediante DBSCAN 48
Criterios de evaluación 52
Ejercicio 1 (20%) 52
Ejercicio 2 (20%) 52
Ejercicio 3 (25%) 52
Ejercicio 4 (25%) 52
Ejercicio 5 (10%) 53

Introducción

Presentación

Esta prueba de evaluación continuada cubre los módulos 5 y 6 del programa de la asignatura.

Competencias

Las competencias que se trabajan en esta prueba son:

- Uso y aplicación de las TIC en el ámbito académico y profesional
- Capacidad para innovar y generar nuevas ideas.
- Capacidad para evaluar soluciones tecnológicas y elaborar propuestas de proyectos teniendo en cuenta los recursos, las alternativas disponibles y las condiciones de mercado.
- Conocer las tecnologías de comunicaciones actuales y emergentes, así como saberlas aplicar convenientemente para diseñar y desarrollar soluciones basadas en sistemas y tecnologías de la información.
- Aplicación de las técnicas específicas de ingeniería del software en las diferentes etapas del ciclo de vida de un proyecto.
- Capacidad para aplicar las técnicas específicas de tratamiento, almacenamiento y administración de datos.
- Capacidad para proponer y evaluar diferentes alternativas tecnológicas para resolver un problema concreto.
- Capacidad de utilizar un lenguaje de programación.
- Capacidad para desarrollar en una herramienta IDE.
- Capacidad de plantear un proyecto de minería de datos.

Objetivos

En esta PEC trabajaremos la generación, interpretación y evaluación de un modelo de agregación y de un modelo donde generaremos reglas de asociación con el software de prácticas. No perderemos de vista las fases de preparación de los datos, calidad del modelo y extracción inicial del conocimiento.

Descripción de la PEC a realizar

La prueba está estructurada en 2 ejercicios teóricos y 3 ejercicios prácticos.

Recursos

Para realizar esta práctica recomendamos como punto de partida la lectura de los siguientes documentos:

- Módulos 5 y 6 del material didáctico.
- El aula laboratorio de R para resolver dudas o problemas.
- RStudio Cheat Sheet: Disponible en el aula Laboratorio de Minería de datos.
- R Base Cheat Sheet: Disponible en el aula Laboratorio de Minería de datos.

Formato y fecha de entrega

El formato de entrega es: usernameestudiant-PAC1.html (pdf o word) y rmd. Fecha de Entrega: 21/04/2021. Se tiene que depositar la PEC en el buzón de entregas del aula.

Nota: Propiedad intelectual

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

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

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

Ejercicio 1

Enunciado

1. Explica de forma resumida cual es el objetivo de los métodos de agregación. Relaciona la respuesta con un ejemplo.
2. Razona si tiene sentido aplicar un método de agregación al ejemplo que pusiste en la PEC1. Si lo tiene, comenta como se podría aplicar y que tipo de resultados se podrían obtener. Si no lo tiene, reformula el problema para que si lo tenga.

Respuesta ejercicio 1

Respuesta 1.1. Los métodos de agregación buscan la agrupación de los datos bajo el criterio de proximidad o distancia. Se busca agruparlos de manera que todos los

componentes del grupo (o clúster), tengan características comunes, y al mismo tiempo que ese grupo sea lo más diferente posible al resto de grupos.

Como ejemplo, tomemos un dataset formado por pacientes víctimas del cáncer de mama. Las observaciones incluirán datos referentes al estilo de vida de los pacientes, si son o no fumadores, el peso, la altura (de aquí podemos estimar su índice de masa corporal y detectar el sobrepeso), hipertensión, si practican deporte con asiduidad y durante cuánto tiempo. El empleo de algún método de agregación nos va a permitir conocer cómo se agrupan las observaciones del dataset, el grupo más mayoritario, y si los grupos están separados entre ellos, o más bien, si se presentan interacciones entre ellos debido a la mezcla de observaciones de diferentes grupos.

No estamos ante un algoritmo que nos ayude a predecir a priori, pero si podemos conocer cuáles son las características principales de los grupos con más observaciones.

Respuesta 1.2. En el ejemplo planteado en la anterior PEC, se buscaba conocer el movimiento del tráfico con el fin de poder predecir la saturación de la red de carreteras y autovías del estado. Para se utilizaba diferentes fuentes de datos:

- Datos de las operadoras principales de telefonía móvil.
- Datos provenientes de la DGT
- Datos de afectación meteorológica.

El utilizar métodos de agregación con los datos nos puede servir para detectar desde el inicio las vías de tráfico que tienden a colapsarse y poder agruparlas por tipo de vía, comunidad, periodo(si se congestionan en fin de semana, entre semana o tienden siempre al colapso). Esto ya de entrada, nos va a permitir el realizar un tipo de análisis descriptivo de los datos con el objetivo de extraer conclusiones para poder proponer planificaciones de mejora en las vías implicadas.

Ejercicio 2

Enunciado

1. Explica de forma resumida cual es el objetivo de los métodos de generación de reglas de asociación. Relaciona la respuesta con un ejemplo.
2. Razona si tiene sentido aplicar un método de generación de reglas de asociación al ejemplo que pusiste en la PEC1. Si lo tiene, comenta como se podría aplicar y que tipo de resultados se podrían obtener. Si no lo tiene, reformula el problema para que si lo tenga.

Respuesta ejercicio 2

Respuesta 2.1. Las reglas de asociación buscan encontrar las **dependencias** entre los atributos de un conjunto de valores de un dataset, con el fin de poder establecer reglas entre ellos. En este dominio, una transacción se refiere a cada grupo de eventos asociados. Así, por ejemplo, analizando los metadatos asociados a las compras de un portal de ventas por internet, podemos definir la transacción T como la venta de un usuario, con los siguientes itemsets (País origen de la IP, artículo visitado, compra(S/N), tiempo visita). En este escenario, una regla de asociación que se podría descubrir sería del tipo "si Pais = España entonces visitas {lista de artículos o familia de artículos visitados}.

Respuesta 2.2. Usar las reglas de asociación en el ejemplo de la anterior PAC podría servir para encontrar aquellas relaciones ocultas entre las diferentes rutas de la red de carreteras del estado. Así, nos puede ayudar a determinar si la congestión de una determinada vía puede influir en que se produzca una saturación en otras.

Ejercicio 3

Enunciado

En este ejercicio seguiréis los pasos del ciclo de vida de un proyecto de minería de datos para el caso de un algoritmo de agregación. Lo haréis con el fichero *clientes.csv* que encontraréis adjunto. No hay que olvidarse de la fase de preparación y análisis de datos. Es muy importante explicar muy bien los resultados obtenidos.

Los ejemplos 1 y 2 se pueden tomar como un punto de partida para la realización de este ejercicio, pero se espera que la respuesta proporcionada tenga un análisis más amplio al mostrado en estos ejemplos, prestando especial atención a explicar el conocimiento que se ha adquirido tras el proceso de minería de datos.

Ejemplo 1: Métodos de agregación con datos autogenerados

En este ejemplo vamos a generar un conjunto de muestras aleatorias para posteriormente usar el algoritmo kmeans para agruparlas. Se crearán las muestras alrededor de dos puntos concretos. Por lo tanto, lo lógico será agrupar en dos clústeres. Puesto que inicialmente, en un problema real, no se conoce cuál es el número más idóneo de clúster k, vamos a probar primero con dos (el valor óptimo) y posteriormente con 4 y 8 clústeres. Para evaluar la calidad de cada proceso de agregación 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 kmeans 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

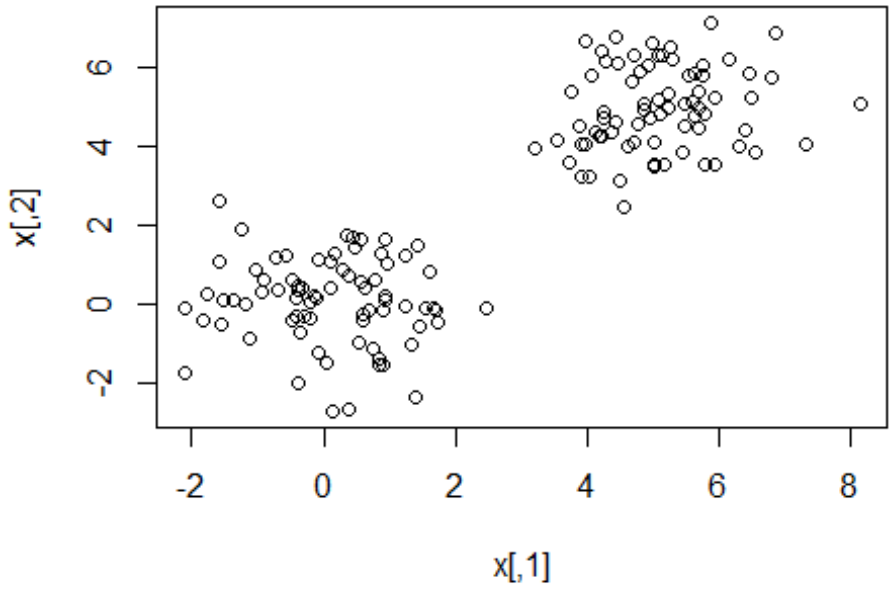
```
library(cluster)
```

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)
```



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 kmeans con 2, 4 y 8 clústers

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

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

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

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

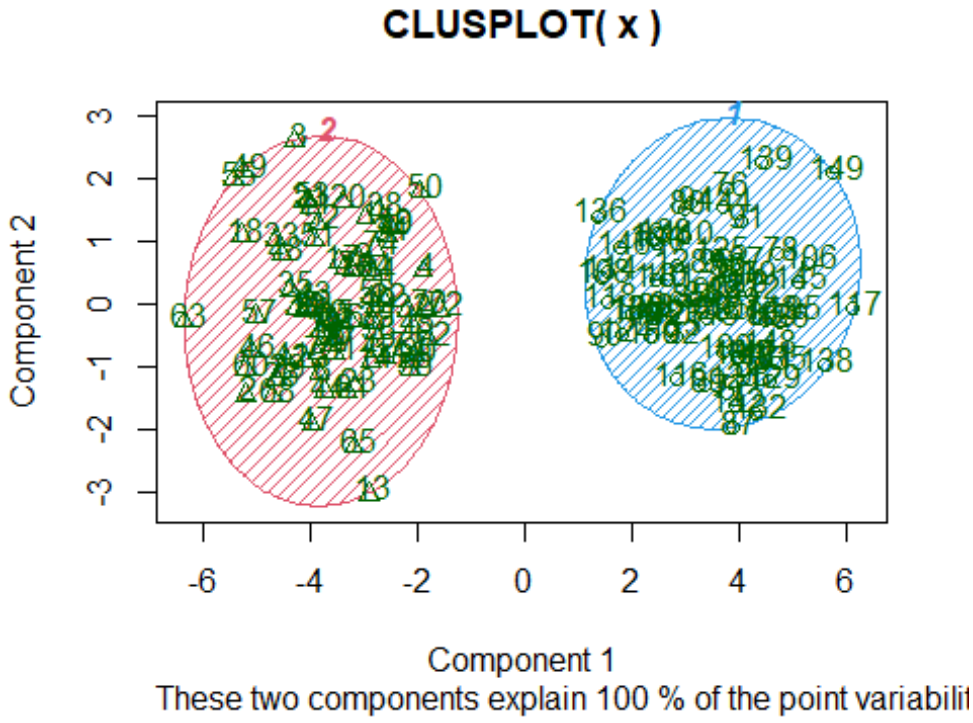
```
y_cluster2
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1
## [149] 1 1
```

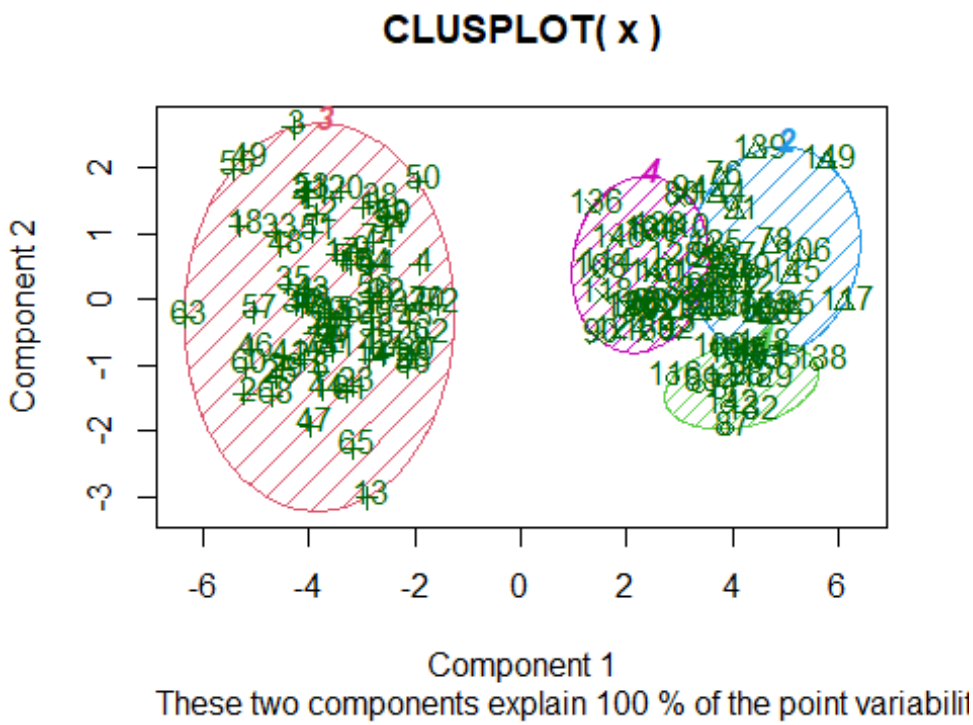
Para visualizar los clústeres podemos usar la función clusplot. Vemos la agrupación con 2 clústeres

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



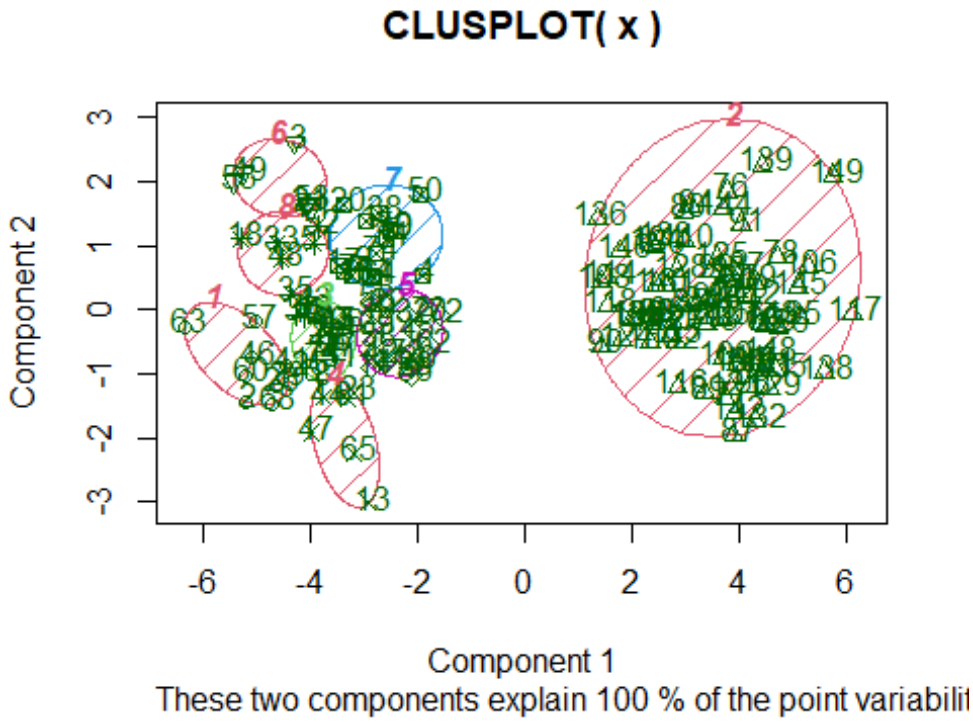
con 4

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



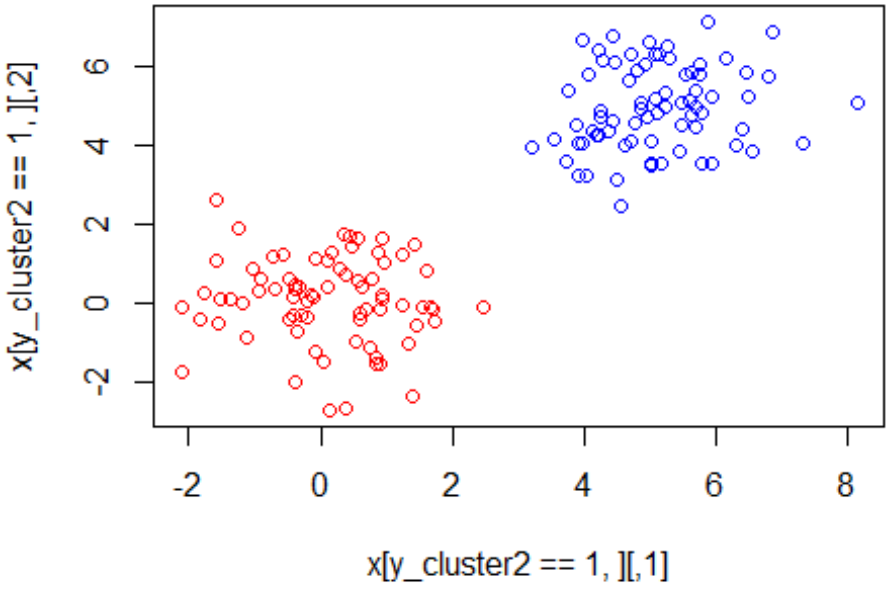
y con 8

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



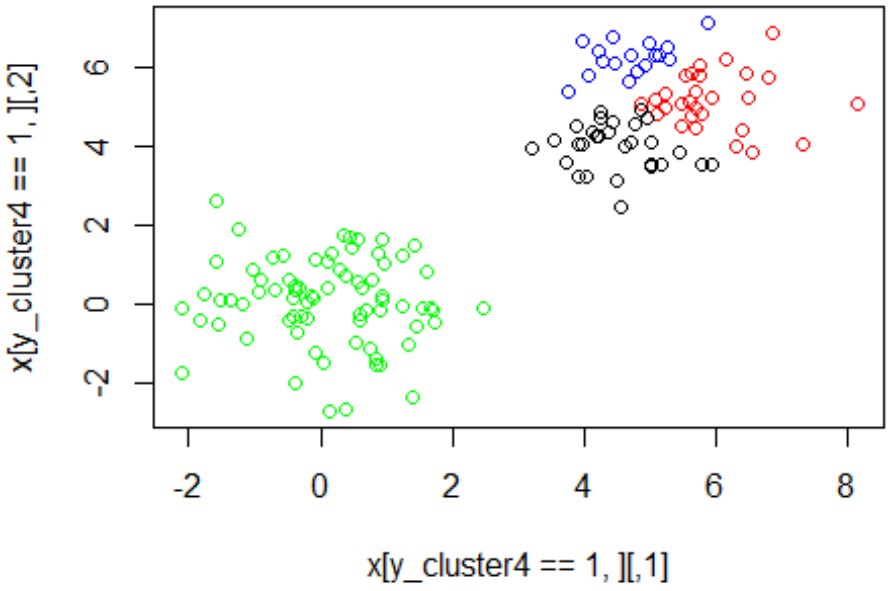
También podemos visualizar el resultado del proceso de agregación con el siguiente código para el caso de 2 clústeres

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



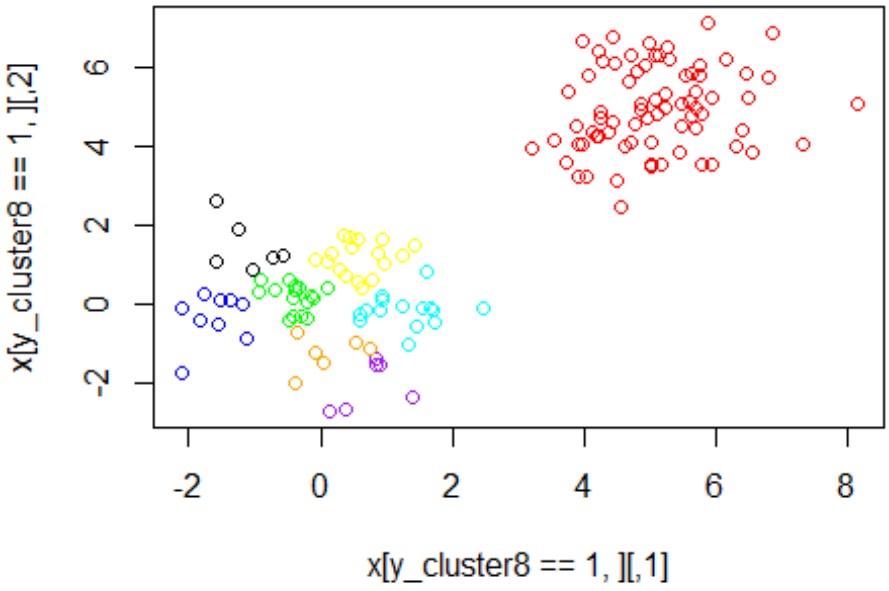
para 4

```
plot(x[y_cluster4==1,],col='blue', xlim=c(min(x[,1]), max(x[,1])), ylim=c
(min(x[,2]), max(x[,2])))
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])))
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] 0.7374764

mean(sk4[,3])

## [1] 0.5328179

mean(sk8[,3])

## [1] 0.5405748
```

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

En este ejercicio no podemos sacar ninguna conclusión a partir de los clústeres obtenidos puesto que los datos de partida no se corresponden con ningún problema real. En un ejemplo real tendríamos que analizar las agrupaciones obtenidas para obtener conocimiento sobre el problema a resolver.

Ejemplo 2: Métodos de agregación con datos reales

A continuación, vamos a ver otro ejemplo de cómo se usan los modelos de agregación. Para ello usaremos el fichero *flores.csv* que encontraréis adjunto.

Cargamos el fichero y visualizamos la estructura de los datos

```
library(cluster)
flores_data<-read.csv("flores.csv", header=T, sep=",")
colnames(flores_data) <- c("sepalLength", "sepalWidth", "petalLength", "p
etalWidth")
summary(flores_data)
```

```
##   sepalLength      sepalWidth     petalLength      petalWidth
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.400   Median :1.300
##  Mean   :5.848   Mean   :3.051   Mean   :3.774   Mean   :1.205
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

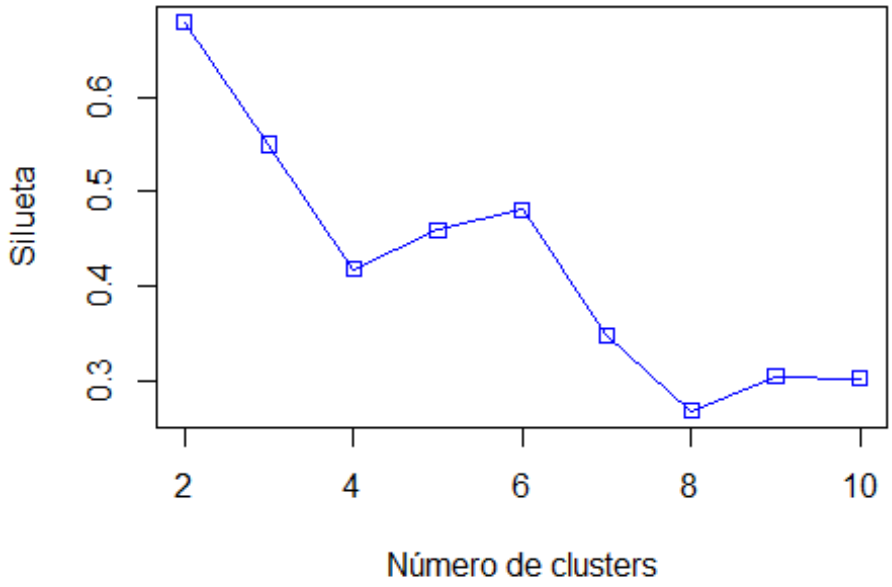
Como podemos comprobar tenemos cuatro características, la longitud y anchura del sépalo y la longitud y anchura del pétalo.

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

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

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

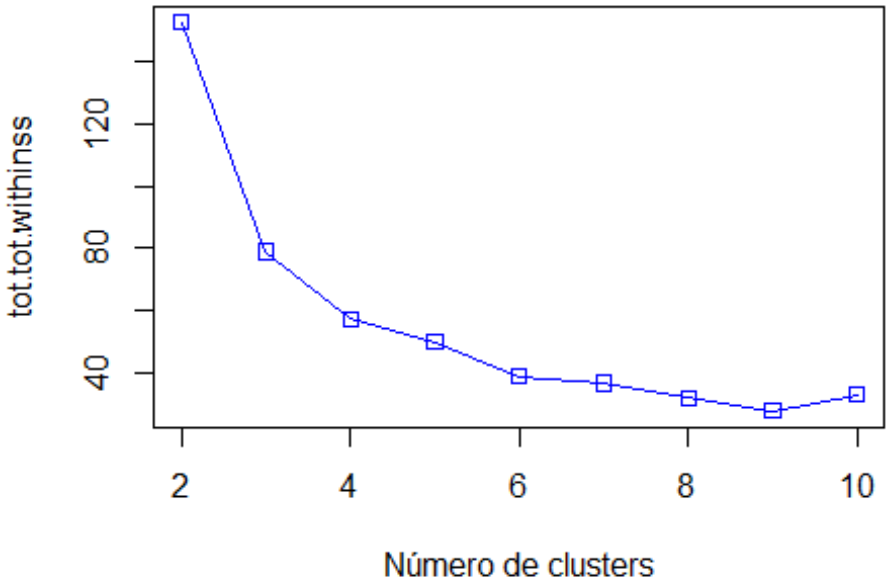
```
plot(2:10,resultados[2:10],type="o",col="blue",pch=0,xlab="Número de clus
ters",ylab="Silueta")
```



El mejor valor que se obtiene es k=2.

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

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



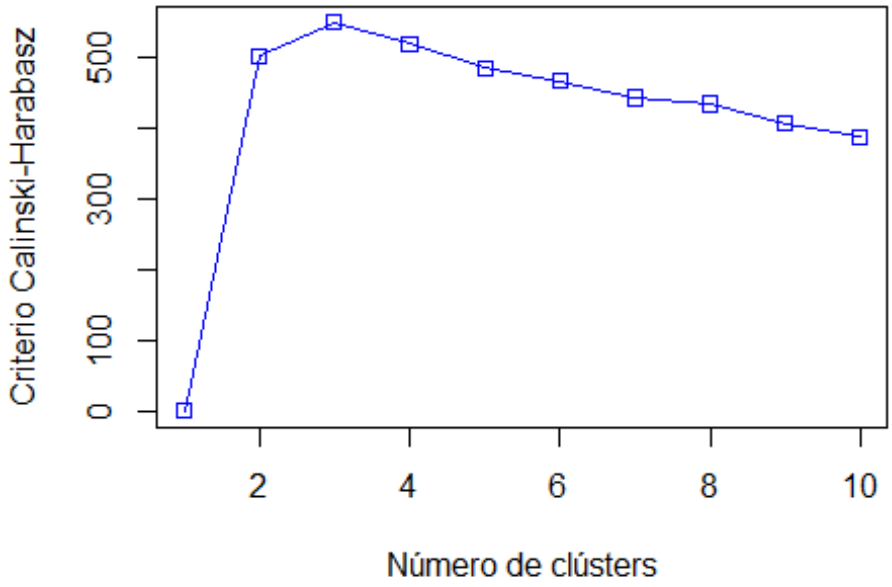
En este caso el número óptimo de clústers es 3 o 4 que es cuando la curva comienza a estabilizarse.

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").

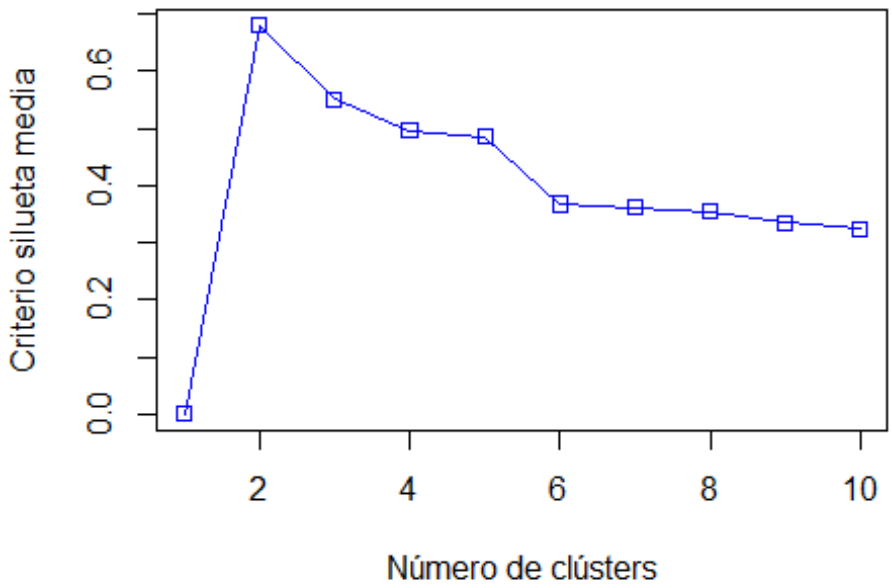
```
library(fpc)
fit_ch  <- kmeansruns(flores_data, krange = 1:10, criterion = "ch")
fit_asw <- kmeansruns(flores_data, krange = 1:10, criterion = "asw")
```

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

```
fit_ch$bestk

## [1] 3

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")
```



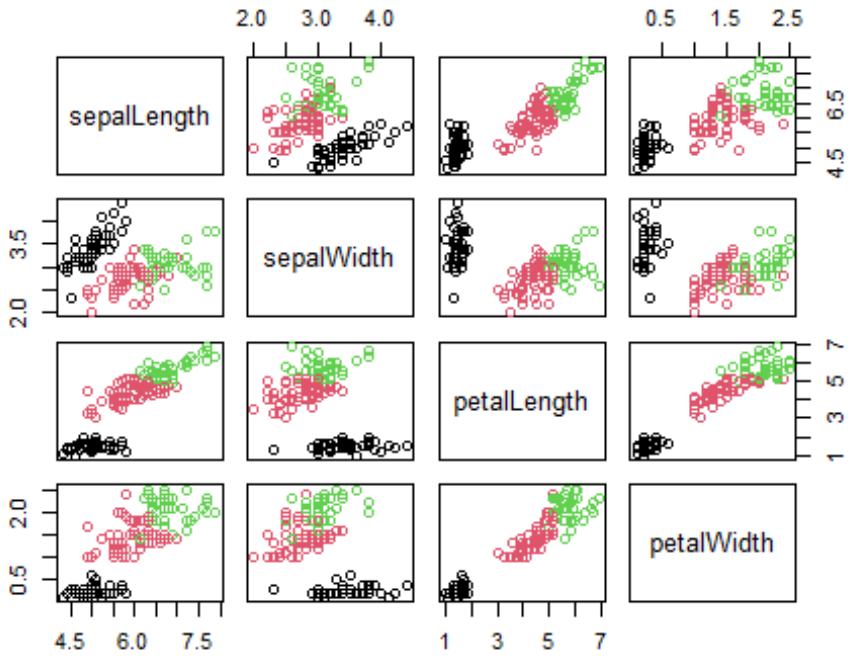
Los resultados son muy parecidos a los que hemos obtenido anteriormente. Con el criterio de la silueta media se obtienen 2 clústers y con el Calinski-Harabasz se obtienen 3.

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.

Tras las pruebas que hemos realizado para obtener el número óptimo de clusters, hemos encontrado que el número de clusters varía según el método entre 2, 3 o 4. Vamos a estudiar los resultados encontrados con 2 y 3 clusters.

A continuación, mostramos visualmente los clusters encontrados suponiendo que hay 3 clusters. Hay que tener en cuenta que nos es posible mostrar los clusters en un espacio de 4 dimensiones. Por lo tanto, mostramos los clusters entre pares de características:

```
cl3 <- kmeans(flores_data, 3)
with(flores_data, pairs(flores_data, col=c(1:4)[cl3$cluster]))
```



La gráfica (que puede cambiar de una ejecución a otra por el factor aleatorio del kmeans) muestra claramente que hay un cluster que está más diferenciado de los otros dos.

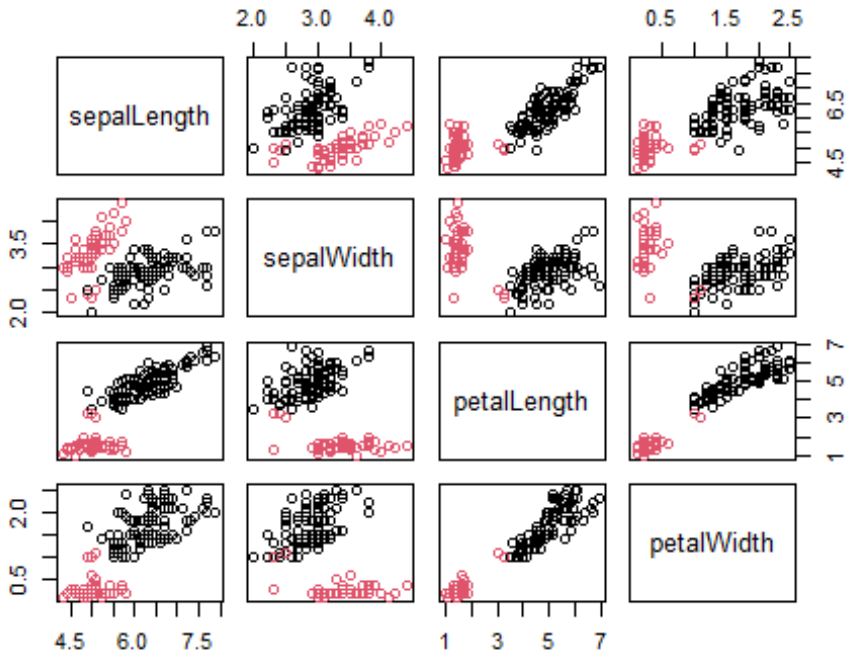
Una buena técnica que ayuda a entender los grupos que se han formado, es analizar las características de cada grupo:

- Grupo 1: Valores del pétalo alto. Longitud del sépalo alto.

- Grupo 2: Valores del pétalo bajos. Longitud del sépalo bajo.
- Grupo 3: Valores intermedios salvo en el ancho del sépalo que es intermedio.

Vamos a ahora a estudiar los conjuntos que se han obtenido con 2 clusters:

```
cl2 <- kmeans(flores_data, 2)
with(flores_data, pairs(flores_data, col=c(1:4)[cl2$cluster]))
```

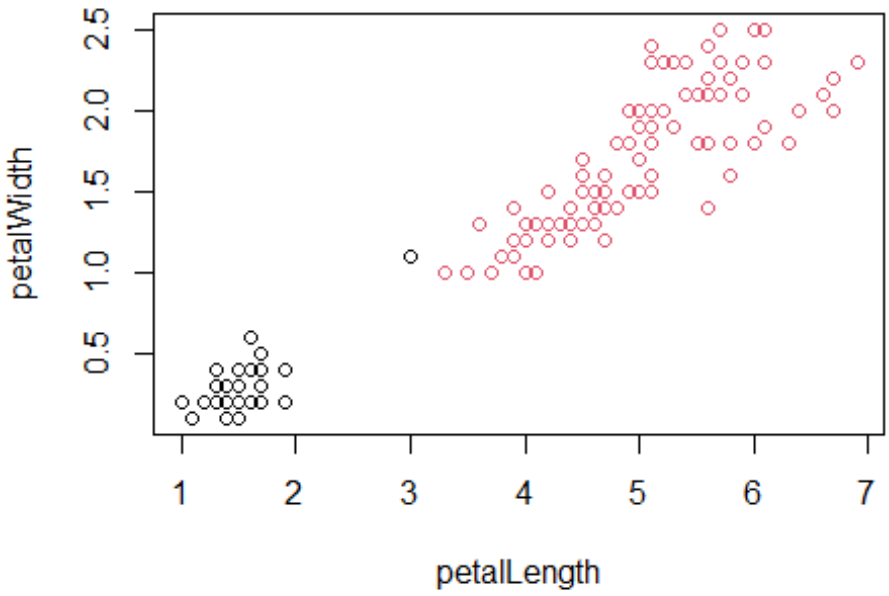


En este caso se comprueba que existen dos grupos con un frontera bastante clara y que con los valores del pétalo tenemos suficiente para diferenciar los dos grupos. Podemos describir los dos grupos como:

- Grupo 1: Valores del pétalo bajos.
- Grupo 2: Valores del pétalo alto.

Esto último lo podemos comprobar usando únicamente las dos características relacionadas con el pétalo:

```
cl2b <- kmeans(flores_data[c(3,4)], 2)
plot(flores_data[c(3,4)], col=cl2b$cluster)
```



Respuesta ejercicio 3

```
# Lectura del fichero
df <- read.csv('clientes.csv', header = T, sep = ',', stringsAsFactors =
T)
print(paste('# de filas cargadas: ', nrow(df)))

## [1] "# de filas cargadas:  200"

str(df)

## 'data.frame':    200 obs. of  5 variables:
##  $ CustomerID        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Genre             : Factor w/ 2 levels "Female","Male": 2 2 1 1 1 1
1 1 2 1 ...
##  $ Age               : int  19 21 20 23 31 22 35 23 64 30 ...
##  $ Annual_Income_.k..: int  15 15 16 16 17 17 18 18 19 19 ...
##  $ Spending_Score    : int  39 81 6 77 40 76 6 94 3 72 ...
```

Exploración de datos

Tenemos un dataset compuesto por 5 variables, una de ellas categórica binaria ('Genre'), y el resto numéricas.

```
print(paste('Número de valores nulos encontrados: ', (which(is.na.data.fr
ame(df) ==T))))

## [1] "Número de valores nulos encontrados:  "
```

Hemos comprobado que el dataset está libre de valores nulos.

Vamos a analizar las variables para ver sus estadísticas básicas

```
summary(df)
```

```
##    CustomerID        Genre          Age        Annual_Income_.k..
##  Min.   :  1.00   Female:112   Min.   :18.00   Min.   : 15.00
##  1st Qu.: 50.75   Male  : 88   1st Qu.:28.75   1st Qu.: 41.50
##  Median :100.50                Median :36.00   Median : 61.50
##  Mean   :100.50                Mean   :38.85   Mean   : 60.56
##  3rd Qu.:150.25                3rd Qu.:49.00   3rd Qu.: 78.00
##  Max.   :200.00                Max.   :70.00   Max.   :137.00
##  Spending_Score
##  Min.   : 1.00
##  1st Qu.:34.75
##  Median :50.00
##  Mean   :50.20
##  3rd Qu.:73.00
##  Max.   :99.00
```

- *Genre*: Vemos que el dataset presenta más muestras de la clase 'Female' que de 'Male'. Examinaremos esto visualmente.

```
ggplot(df, aes(x=Genre, fill= Genre)) + geom_bar()
```


```
print(table(df$Genre))
```

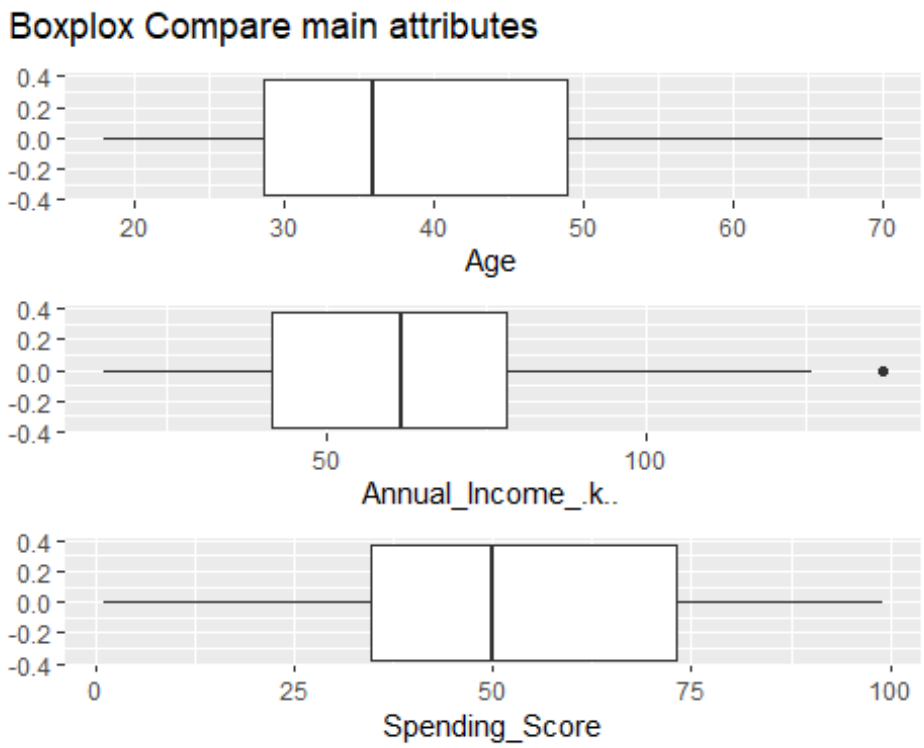
```
##
## Female   Male
##    112     88
```

```
print(paste('% de mujeres respecto al total: ', (length(df[df$Genre == 'F
emale', ]$Genre)/ length(df$Genre))*100.0))
```

```
## [1] "% de mujeres respecto al total:  56"
```

Se comprueba un cierto balanceo del dataset hacia las mujeres, pero poco significativo. Vamos a ver visualmente la distribución de las tres variables.

```
p1 <- ggplot(data=df,aes(x=Age))+geom_boxplot()
p2 <- ggplot(data=df,aes(x=Annual_Income_.k..))+geom_boxplot()
p3 <- ggplot(data=df,aes(x=Spending_Score))+geom_boxplot()

(p1 / p2 / p3) + plot_annotation(title = 'Boxplox Compare main attributes
')
```



En el caso de la edad, podemos ver como la mediana se encuentra escorada hacia la derecha, encontrándonos una población que tiende a los 40 años.

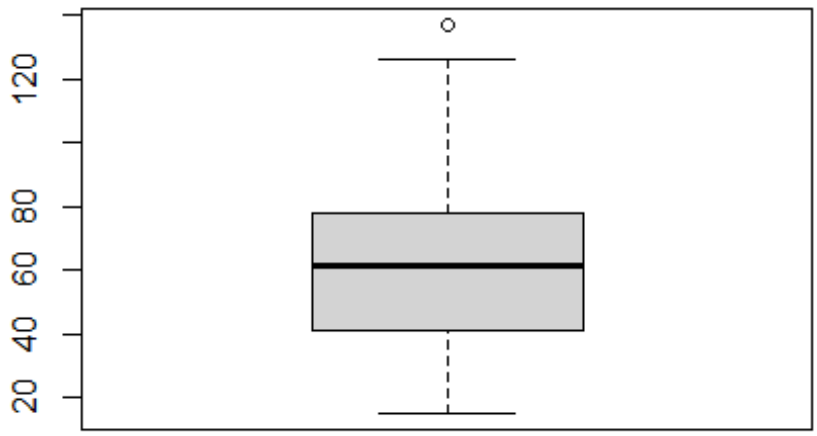
```
summary(df$Age)[3:4]
```

```
## Median   Mean
##  36.00  38.85
```

En efecto, vemos una diferencia entre media y mediana que, no obstante, siendo de casi 3 años **no representa una gran desviación de la muestra**.

El resto de las variables se encuentran bastante centradas, aunque en los ingresos anuales vemos un dato extremo. Vamos a examinarlo en detalle.

```
out <- boxplot(df$Annual_Income_.k..)$out
```




```
df[df$Annual_Income_.k..== 137, ]
```

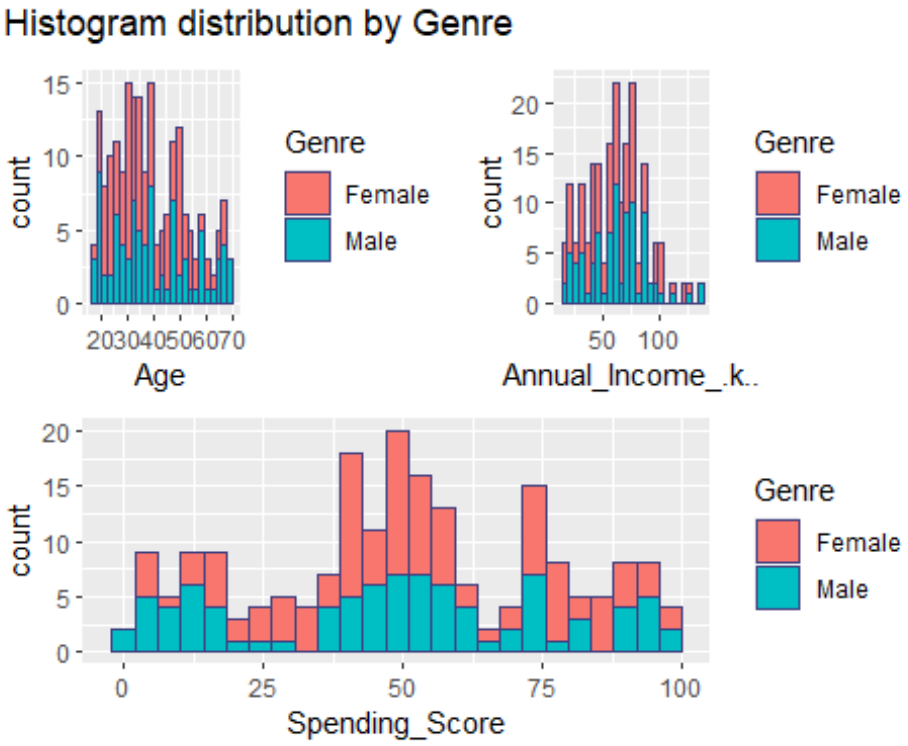
```
##     CustomerID Genre Age Annual_Income_.k.. Spending_Score
## 199        199  Male  32                137             18
## 200        200  Male  30                137             83
```

Corresponden a dos valores de 137k de ingresos. Siendo un valor elevado, **no podemos considerarlo un error**, por lo que no tomaremos medidas adicionales.

Examinemos ahora la distribución de los datos

```
color <- "#404080"
p1 <- ggplot(data=df,aes(x=Age, fill=Genre))+ geom_histogram(bins = 25, c
olor=color)
p2 <- ggplot(data=df,aes(x=Annual_Income_.k.., fill=Genre))+ geom_histogr
am(bins = 25, color=color)
p3 <- ggplot(data=df,aes(x=Spending_Score, fill=Genre))+ geom_histogram(b
ins = 25, color=color)
```

```
(p1 | p2) / p3 + plot_annotation(title = 'Histogram distribution by Genre
')
```



En la visualización de las tres variables, **no se aprecia normalidad** en las muestras. Es interesante ver como el valor del spending_score muestra valores máximos alrededor de la puntuación media, cosa que no ocurre en los hombres, que muestran una distribución más monótona. Los valores de edad se encuentran bastante bien representados, decayendo pasado los 60 años.

Escalado

Como paso siguiente vamos a aplicar un escalado a todas las variables numéricas. La variable categórica de edad, al ser factor, ya se encuentra normalizada. La razón de aplicar el escalado, es que las variables tienen diferentes rangos, tal y como se puede ver a continuación.

```
print('Valores extremos de Ingresos anuales: '); print(summary(df$Annual_
Income_.k..)[c(1,6)])

## [1] "Valores extremos de Ingresos anuales: "

## Min. Max.
##   15  137

print('Valores extremos de Puntuación del gasto: '); print(summary(df$ Sp
ending_Score)[c(1,6)])

## [1] "Valores extremos de Puntuación del gasto: "
```

```
## Min. Max.
##    1   99
```

Con el escalado, mantenemos **los valores de media y la varianza intactos**, cosa que no pasa con la normalización. Los algoritmos de clasificación se basan en distancias, por lo que es importante que los rangos de valores de las diferentes variables se encuentren alineados, pero sin tocar su media para no alterar el resultado.

```
df.scale <- as.data.frame(scale(df[,c('Age', 'Annual_Income_.k..', 'Spending_Score')], center = T, scale = T))
#df.scale <- cbind(df[, 1:2], df.scale)
kable(head(df.scale), format = "markdown")
```

Age	Annual_Income_.k..	Spending_Score
-1.4210029	-1.734646	-0.4337131
-1.2778288	-1.734646	1.1927111
-1.3494159	-1.696572	-1.7116178
-1.1346547	-1.696572	1.0378135
-0.5619583	-1.658499	-0.3949887
-1.2062418	-1.658499	0.9990891

```
head(df.scale)
```

```
##          Age Annual_Income_.k.. Spending_Score
## 1 -1.4210029          -1.734646     -0.4337131
## 2 -1.2778288          -1.734646      1.1927111
## 3 -1.3494159          -1.696572     -1.7116178
## 4 -1.1346547          -1.696572      1.0378135
## 5 -0.5619583          -1.658498     -0.3949887
## 6 -1.2062418          -1.658498      0.9990891
```

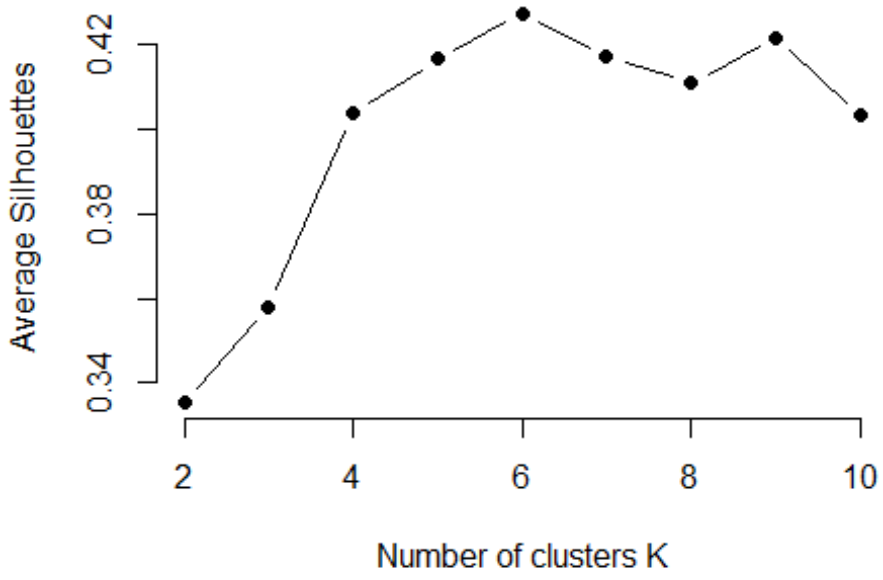
Hemos eliminado las variables que contienen el ID y el sexo, ya que no son necesarias y **los algoritmos de clasificación no trabajan con variables categóricas**.

Clusterización

Vamos a aplicar el algoritmo de **K-means** para encontrar que grupos podemos tener en el dataset. Para la evaluación de la calidad del proceso de agregación utilizaremos la función **silhouette**.

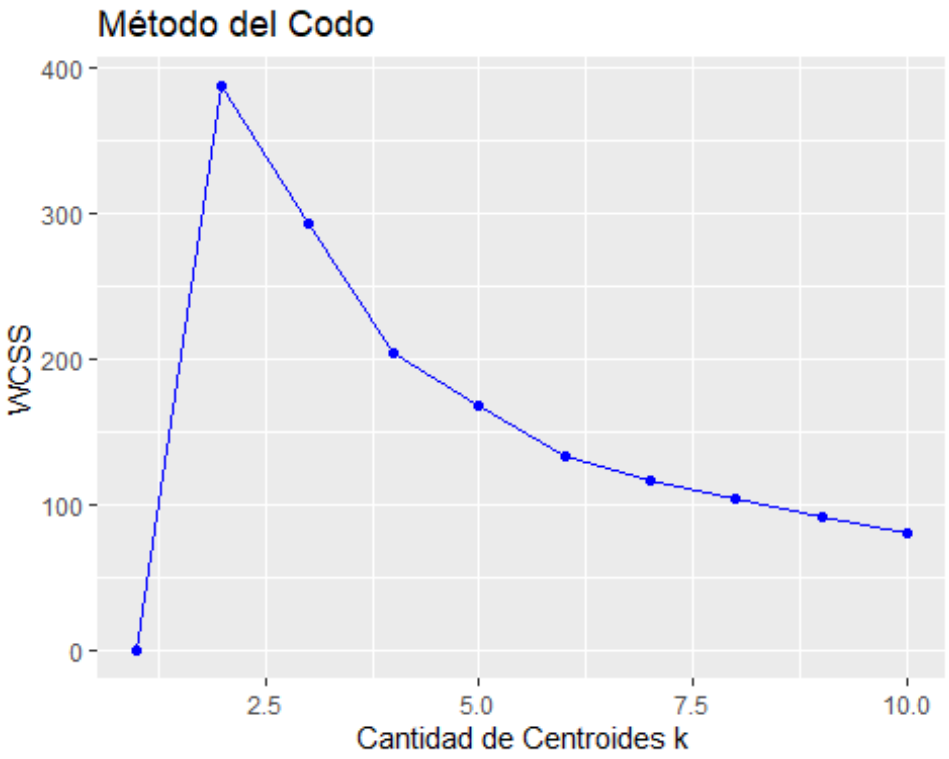
```
set.seed(123)
d <- daisy(df.scale)
resultados.s <- rep(0,10)
resultados.w <- rep(0,10)
for (i in 2:10){
  fit <- kmeans(df.scale, i, nstart = 25)
  y_cluster <- fit$cluster
  sk <- silhouette(y_cluster, d)
  resultados.s[i] <- mean(sk[,3])
```

```
  resultados.w[i] <- sum(fit$withinss)
}
plot(2:10, resultados.s[2:10],
     #type = "o", col="blue", pch=0,
      type = "b", pch = 19, frame = FALSE,
     xlab= "Number of clusters K", ylab="Average Silhouettes")
```



Encontramos que el mejor valor lo obtenemos para 6 agrupaciones. Contrastaremos el número de clústers encontrado con la visualización de la medida por el método del Codo.

```
ggplot() + geom_point(aes(x = 1:10, y = resultados.w), color = 'blue') +
  geom_line(aes(x = 1:10, y = resultados.w), color = 'blue') +
  ggtitle("Método del Codo") +
  xlab('Cantidad de Centroides k') +
  ylab('WCSS')
```



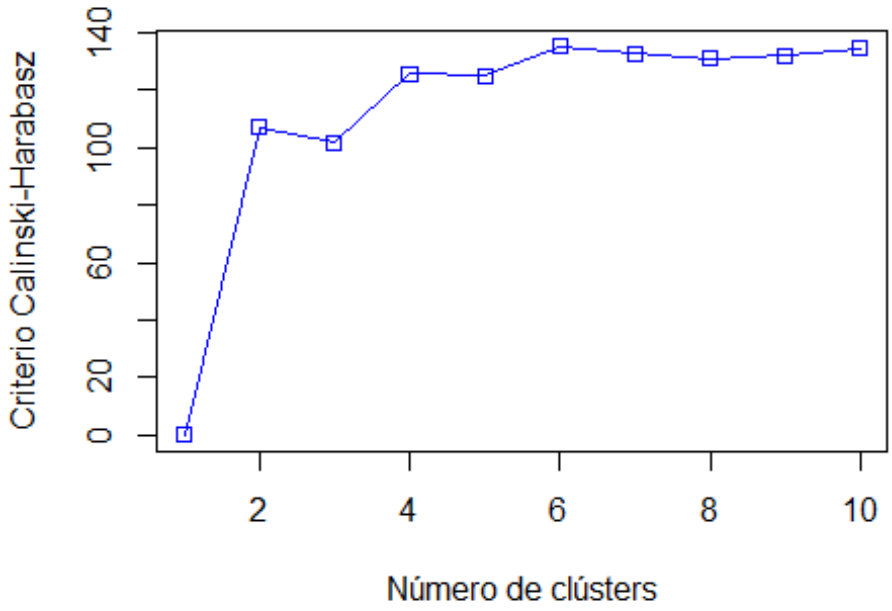
En este caso, es complicado determinar donde se suaviza la pendiente de la curva, aunque podríamos acordar que parece que k=7 sea el punto en donde esto sucede.

Repetiremos las pruebas con **Calinski-Harabasz** y otra vez la silueta media, pero utilizando kmeansruns.

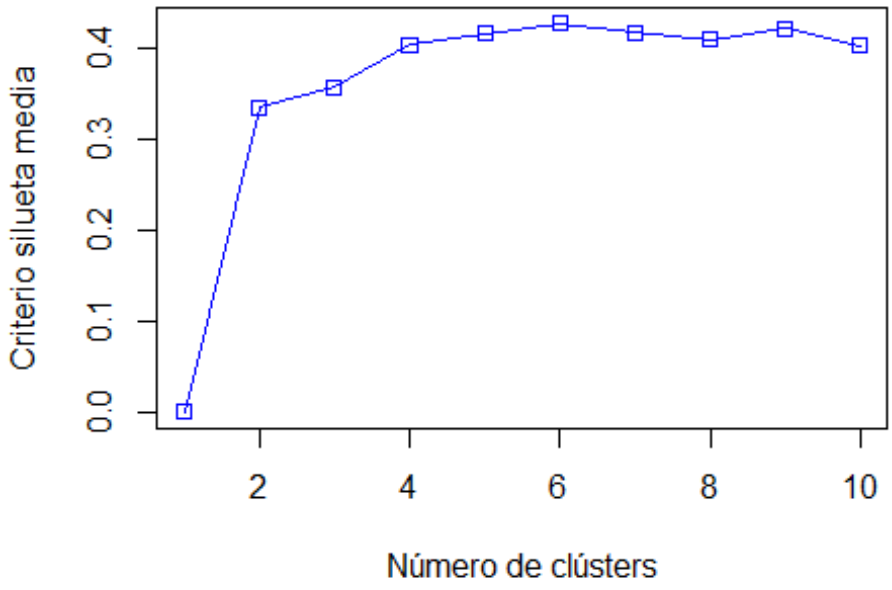
```
library(fpc)
fit_ch  <- kmeansruns(df.scale, krange = 2:10, criterion = "ch")
print(paste('Mejor valor según Calinski-Harabasz: ', fit_ch$bestk))

## [1] "Mejor valor según Calinski-Harabasz:  6"

fit_asw <- kmeansruns(df.scale, krange = 2:10, criterion = "asw")
print(paste('Mejor valor según la silueta media: ', fit_asw$bestk))

## [1] "Mejor valor según la silueta media:  6"
```

Tanto Calisnski-Harabasz, como el valor medio de siluette, nos muestra que el mejor valor lo tenemos con 6 clústeres.

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

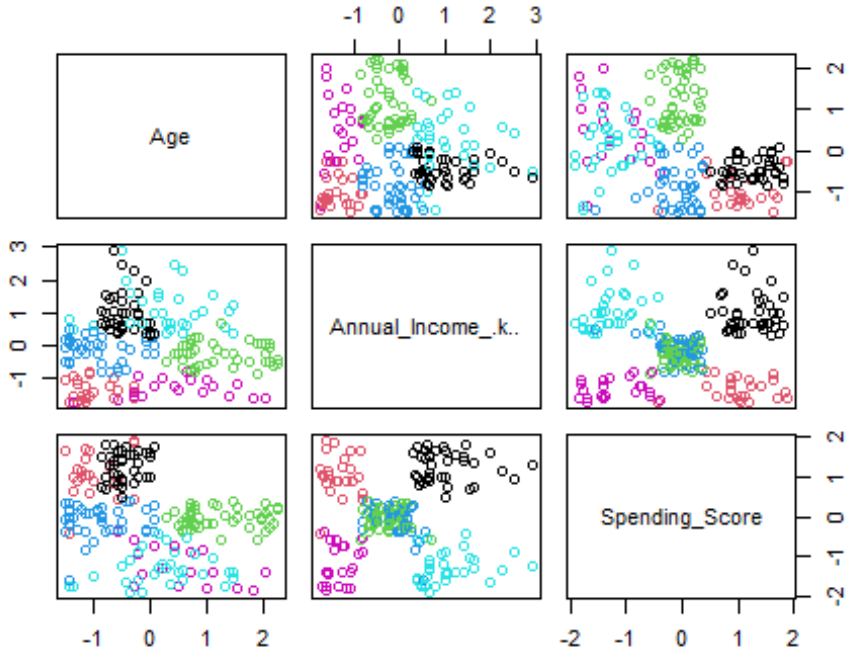



```
plot(1:10,fit_asw$crit,type="o",col="blue",pch=0,xlab="Número de clústers
",ylab="Criterio silueta media")
```



Una vez consensuado que el valor de k=6 parece ser el mejor para agrupar los datos del dataset, visualizaremos los resultados.

```
k <- 6
cl <- kmeans(df.scale, k)
pairs(df.scale, col= cl$cluster)
```



Vemos una gran **confusión** en cuanto a separación de grupos entre las variables Annual_Income_k.. y Spending_Score, sobre todo en la parte central de los datos cuesta distinguir a que grupo pertenecen las observaciones.

Enlaces Externos: <https://uc-r.github.io/kmeans_clustering>

Ejercicio 4

Enunciado

En este ejercicio seguiréis los pasos del ciclo d[e vida de un proyecto de minería de datos para el caso de un algoritmo de generación de reglas de asociación. Lo haréis con el fichero *Lastfm.csv* que encontraréis adjunto. Este fichero contiene un conjunto de registros del histórico de las canciones que ha escuchado un usuario en un portal

Web de música. “artist” es el nombre del grupo que ha escuchado, “sex” y “country” corresponden a variables que describen al usuario.

El ejemplo 3 se pueden tomar como un punto de partida para la realización de este ejercicio, pero se espera que la respuesta proporcionada tenga un análisis más amplio al mostrado en este ejemplo, prestando especial atención a explicar el conocimiento que se ha adquirido tras el proceso de minería de datos.

Ejemplo 3: Métodos de generación de reglas de asociación

En este ejemplo vamos a trabajar el algoritmo “apriori” para obtener reglas de asociación a partir de un data set. Dichas reglas nos ayudarán a comprender cómo la información del data set se relaciona entre sí.

Para dicho objetivo vamos a trabajar el dataset de Groceries, que ya viene incluido con las librerías de arules.

```
# install.packages("arules")
library(arules)
data("Groceries")
```

Para saber más sobre este dataset ejecutar el comando “?Groceries”.

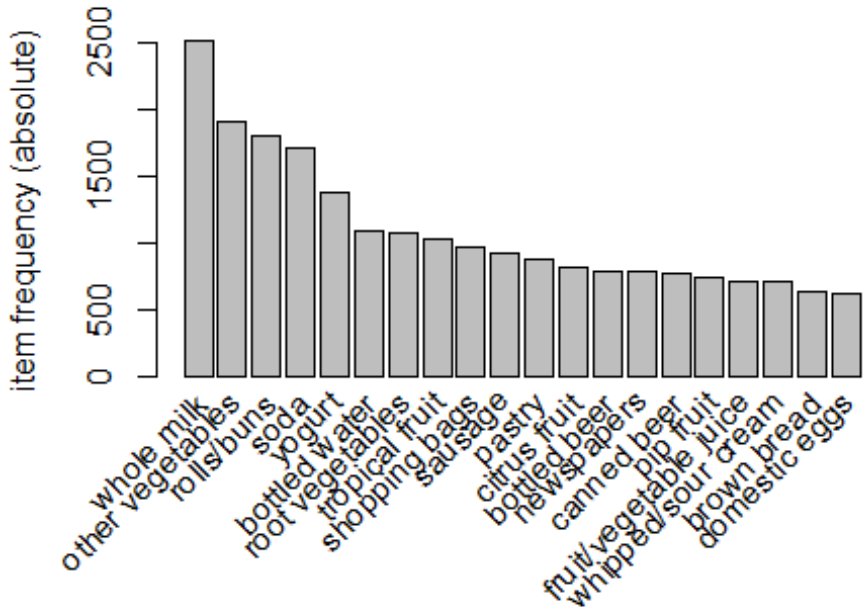
Inspeccionamos el dataset y vemos que tiene un listado de elementos que fueron comprados juntos. Vamos a analizarlo un poco visualmente.

```
inspect(head(Groceries, 5))
```

```
##     items
## [1] {citrus fruit,
##      semi-finished bread,
##      margarine,
##      ready soups}
## [2] {tropical fruit,
##      yogurt,
##      coffee}
## [3] {whole milk}
## [4] {pip fruit,
##      yogurt,
##      cream cheese ,
##      meat spreads}
## [5] {other vegetables,
##      whole milk,
##      condensed milk,
##      long life bakery product}
```

En el siguiente plot podemos ver que los tres elementos más vendidos son la leche entera, otras verduras y bollería. Dada la simplicidad del Dataset no se pueden hacer mucho más análisis. Pero para datasets más complejos miraríamos la frecuencia y distribución de todos los campos, en busca de posibles errores.

```
itemFrequencyPlot(Groceries,topN=20,type="absolute")
```



Si lanzamos el algoritmo “apriori”, generaremos directamente un set de reglas con diferente soporte, confianza y lift. El soporte indica cuantas veces se han encontrado las reglas {lsh => rhs} en el dataset, cuanto más alto mejor. La confianza habla de la probabilidad de que {rhs} se de en función de {lhs}. Y el lift es un parámetro que nos indica cuánto de aleatoriedad hay en las reglas. Un lift de 1 o menos es que las reglas son completamente fruto del azar.

```
grocery_rules <- apriori(Groceries, parameter = list(support = 0.01, conf
idence = 0.5))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support min
len
##         0.5    0.1    1 none FALSE            TRUE       5    0.01
1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
```

```
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(head(sort(grocery_rules, by = "confidence"), 3))
```

```
##     lhs                                rhs                support
## [1] {citrus fruit,root vegetables}   => {other vegetables} 0.01037112
## [2] {tropical fruit,root vegetables} => {other vegetables} 0.01230300
## [3] {curd,yogurt}                    => {whole milk}       0.01006609
##     confidence coverage   lift     count
## [1] 0.5862069  0.01769192 3.029608 102
## [2] 0.5845411  0.02104728 3.020999 121
## [3] 0.5823529  0.01728521 2.279125  99
```

Podemos probar a ordenar las reglas por los diferentes parámetros, para ver que información podemos obtener.

```
inspect(head(sort(grocery_rules, by = "support"), 3))
```

```
##     lhs                                          rhs          support    c
onfidence
## [1] {other vegetables,yogurt}                 => {whole milk} 0.02226741 0
.5128806
## [2] {tropical fruit,yogurt}                   => {whole milk} 0.01514997 0
.5173611
## [3] {other vegetables,whipped/sour cream} => {whole milk} 0.01464159 0
.5070423
##     coverage   lift     count
## [1] 0.04341637 2.007235 219
## [2] 0.02928317 2.024770 149
## [3] 0.02887646 1.984385 144
```

Ordenando por support vemos que, con un lift de 2 y una confianza del 51%, podemos decir que la gente que en la misma compra hacía verduras y yogurt, compraban también leche entera. Hay que tener en cuenta que la leche entera es por otro lado el elemento más vendido de la tienda.

```
inspect(head(sort(grocery_rules, by = "lift"), 3))
```

```
##     lhs                                rhs                support
## [1] {citrus fruit,root vegetables}   => {other vegetables} 0.01037112
## [2] {tropical fruit,root vegetables} => {other vegetables} 0.01230300
## [3] {root vegetables,rolls/buns}     => {other vegetables} 0.01220132
##     confidence coverage   lift     count
## [1] 0.5862069  0.01769192 3.029608 102
```

```
## [2] 0.5845411  0.02104728 3.020999 121
## [3] 0.5020921  0.02430097 2.594890 120
```

Por otro lado, si ordenamos por lift, vemos que con un soporte del 1% y una confianza del 58%, la gente que compra cítricos y tubérculos compra también verduras.

Esta información nos puede ayudar a dar consejos a la dirección de la disposición de los elementos en la tienda o de que productos poner en oferta según lo que se ha comprado. Y si tuviéramos más información podríamos hacer análisis más profundos y ver que clientes compran exactamente qué.

Respuesta ejercicio 4

Carga de datos

El dataset lo componen 289955 registros, repartidos en cuatro variables: user, artist, sex y country.

```
df <- read.csv(file = 'lastfm.csv', header = T, sep = ',', stringsAsFacto
rs = T)
```

Hemos cargado el fichero y comprobado el número de filas totales. Vamos a buscar relaciones entre los usuarios y los artistas o grupos que escuchan.

```
dats <- split(x = df$artist, f = df$user)
tr.artist <- as(dats, "transactions")

## Warning in asMethod(object): removing duplicated items in transactions

(tr.artist)

## transactions in sparse format with
##  15000 transactions (rows) and
##  1004 items (columns)
```

Exploración de datos

Visualicemos las estadísticas básicas del dataset.

```
summary(tr.artist)

## transactions as itemMatrix in sparse format with
##  15000 rows (elements/itemsets/transactions) and
##  1004 columns (items) and a density of 0.01925319
##
## most frequent items:
##             radiohead           the beatles              coldplay
##                  2704                  2668                  2378
## red hot chili peppers                  muse               (Other)
##                  1786                  1711                278706
##
## element (itemset/transaction) length distribution:
```

```
## sizes
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  1
8  19  20
## 185 222 280 302 359 385 472 461 491 501 504 482 472 471 479 477 456 45
5 444 455
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  3
8  39  40
## 436 478 426 438 408 446 417 375 348 340 316 293 274 286 238 208 193 18
1 128 102
##  41  42  43  44  45  46  47  48  49  50  51  52  54  55  63  76
##  93  61  55  36  23  15   6  11   2   1   5   3   1   2   1   1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   11.00   19.00   19.33   27.00   76.00
##
## includes extended item information - examples:
##                                        labels
## 1 ...and you will know us by the trail of dead
## 2                                     [unknown]
## 3                                          2pac
##
## includes extended transaction information - examples:
##   transactionID
## 1             1
## 2             3
## 3             4
```

La mayoría de las transacciones la forman 19 artistas o grupos, aunque llegamos a un máximo de 76.

Podemos ahora visualizar la salida de las dos primeras transacciones.

```
inspect(tr.artist [1:2])
```

```
##     items                      transactionID
## [1] {dropkick murphys,
##      edguy,
##      eluveitie,
##      goldfrapp,
##      guano apes,
##      jack johnson,
##      john mayer,
##      judas priest,
##      le tigre,
##      red hot chili peppers,
##      rob zombie,
##      schandmaul,
##      the black dahlia murder,
##      the killers,
##      the rolling stones,
##      the who}                              1
```

```
## [2] {aesop rock,
##      air,
##      amon tobin,
##      animal collective,
##      aphex twin,
##      arcade fire,
##      atmosphere,
##      autechre,
##      beastie boys,
##      boards of canada,
##      broken social scene,
##      cocorosie,
##      devendra banhart,
##      four tet,
##      goldfrapp,
##      joanna newsom,
##      m83,
##      massive attack,
##      max richter,
##      mf doom,
##      neutral milk hotel,
##      pavement,
##      plaid,
##      portishead,
##      prefuse 73,
##      radiohead,
##      sage francis,
##      the books,
##      the flashbulb}                          3
```

Estamos en disposición de encontrar los items más frecuentes dentro del conjunto de las transacciones.

```
fr_items <- itemFrequency(x = tr.artist, type = "relative")
fr_items %>% sort(decreasing = TRUE) %>% head(5)
```

```
##             radiohead           the beatles              coldplay
##             0.1802667             0.1778667            0.1585333
## red hot chili peppers                  muse
##             0.1190667             0.1140667
```

Radiohead y The beatles son los que cuentan con mayor soporte, con un 18% ambos.

Aplicando Reglas de Asociación

La salida del algoritmo ***"apriori"*** nos indica el soporte, la confianza y el lift.

- ***Soporte***: Indica el número de transacciones encontradas en la regla de {lhs} sobre el total. Mide la frecuencia.

- ***Confianza***: Nos habla de la *probabilidad* de que la transacción que contiene {rhs} también encuentre los items de {lhs}. Por tanto, la confianza nos muestra la fortaleza de la regla.
- ***Lift***: Nos indicará la aleatoriedad en la regla. Un lift igual a 1 o cercano a 1 indica que la relación es producto del azar. Un lift>1 implica una relación fuerte, mientras que un lift<=1 nos indica una relación trivial.

Lanzamos el algoritmo *"apriori"* para generar las reglas del dataset con un mínimo de soporte del 1 % y una confianza mínima del 50 %.

```
df_rules <- apriori(tr.artist, parameter = list(support = 0.01, confidenc
e = 0.5, target = "rules"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support min
len
##         0.5    0.1    1 none FALSE            TRUE       5    0.01
1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 150
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[1004 item(s), 15000 transaction(s)] done [0.17s].
## sorting and recoding items ... [655 item(s)] done [0.01s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 done [0.05s].
## writing ... [50 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

Se han escrito 50 reglas. Vamos a mirar sus estadísticas.

```
summary(df_rules)
```

```
## set of 50 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3
## 15 35
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0     2.0     3.0     2.7     3.0     3.0
##
```

```
## summary of quality measures:
##     support          confidence        coverage            lift       
##  Min.   :0.01000   Min.   :0.5013   Min.   :0.01587   Min.   : 2.781  
##  1st Qu.:0.01062   1st Qu.:0.5216   1st Qu.:0.01847   1st Qu.: 3.120  
##  Median :0.01147   Median :0.5430   Median :0.02150   Median : 3.283  
##  Mean   :0.01296   Mean   :0.5556   Mean   :0.02359   Mean   : 3.963  
##  3rd Qu.:0.01372   3rd Qu.:0.5858   3rd Qu.:0.02655   3rd Qu.: 3.711  
##  Max.   :0.02927   Max.   :0.6627   Max.   :0.05747   Max.   :13.416  
##      count        
##  Min.   :150.0    
##  1st Qu.:159.2    
##  Median :172.0    
##  Mean   :194.4    
##  3rd Qu.:205.8    
##  Max.   :439.0    
##
## mining info:
##       data ntransactions support confidence
##  tr.artist         15000    0.01        0.5
```

Del set de 50 reglas, en su mayoría cuentan con 3 items (Median), con una confianza de alrededor el 55% y un lift mínimo muy por encima de 1 que nos indica que existe una fuerte relación en las reglas encontradas.

Visualizamos las reglas ordenada por confianza en orden decreciente

```
inspect(head(sort(df_rules, decreasing = T, by='confidence')))
```

```
##     lhs                                      rhs         support    conf
idence
## [1] {oasis,the killers}                   => {coldplay}  0.01113333 0.66
26984
## [2] {sigur rÃ³s,the beatles}               => {radiohead} 0.01046667 0.64
34426
## [3] {keane}                               => {coldplay}  0.02226667 0.63
74046
## [4] {radiohead,snow patrol}               => {coldplay}  0.01006667 0.63
44538
## [5] {coldplay,the smashing pumpkins}      => {radiohead} 0.01093333 0.62
83525
## [6] {the beatles,the smashing pumpkins} => {radiohead} 0.01146667 0.62
09386
##     coverage   lift     count
## [1] 0.01680000 4.180183 167
## [2] 0.01626667 3.569393 157
## [3] 0.03493333 4.020634 334
## [4] 0.01586667 4.002021 151
## [5] 0.01740000 3.485683 164
## [6] 0.01846667 3.444556 172
```

En el cuadro anterior vemos las reglas ordenadas por la confianza. Por ejemplo, en la primera línea vemos que si un oyente escucha a Oasis y a The killer, es muy probable (66 %) que también escuche a Coldplay. Todo ello con un Lift muy bueno, superior a 1 y soportado por 167 transacciones.

Vamos a centrarnos en el grupo Coldplay encontrado y vamos a explorar que afinidad muestran los oyentes con el resto de grupos de música o artista.

```
df_rules_search <- apriori(tr.artist, parameter = list(support = 0.01, co
nfidence = 0.5, target = "rules"), appearance = list(rhs= c('coldplay')))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support min
len
##         0.5    0.1    1 none FALSE            TRUE       5    0.01
1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 150
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[1004 item(s), 15000 transaction(s)] done [0.10s].
## sorting and recoding items ... [655 item(s)] done [0.01s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 done [0.03s].
## writing ... [16 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

Mostramos las reglas ordenada por confianza.

```
inspect(head(sort(df_rules_search, decreasing = T, by='confidence')))
```

```
##     lhs                                   rhs        support    confide
nce
## [1] {oasis,the killers}               => {coldplay} 0.01113333 0.66269
84
## [2] {keane}                           => {coldplay} 0.02226667 0.63740
46
## [3] {radiohead,snow patrol}           => {coldplay} 0.01006667 0.63445
38
## [4] {snow patrol,the killers}         => {coldplay} 0.01040000 0.59541
98
## [5] {death cab for cutie,the killers} => {coldplay} 0.01086667 0.58844
77
```

```
## [6] {oasis,radiohead}                 => {coldplay} 0.01273333 0.58769
23
##     coverage   lift     count
## [1] 0.01680000 4.180183 167
## [2] 0.03493333 4.020634 334
## [3] 0.01586667 4.002021 151
## [4] 0.01746667 3.755802 156
## [5] 0.01846667 3.711823 163
## [6] 0.02166667 3.707058 191
```

Hemos afinado más la búsqueda mostrando que las reglas de asociación entre los principales artistas o grupos y su afinidad con el grupo Coldplay.

Ejercicio 5

Enunciado

Busca información sobre otros métodos de agregación diferentes al *k-means*. Partiendo del Ejercicio 3, probar el funcionamiento de al menos 2 métodos diferentes y comparar los resultados obtenidos.

Respuesta ejercicio 5

Agrupamiento jerárquico

El método del agrupamiento jerárquico se basa en construir una jerarquía de grupos, basado en la **semejanza de los datos**. Utilizando el concepto de distancia entre los puntos, se calcula los puntos que se encuentra más cercanos entre ellos, por pares. El siguiente paso es ir *agregando* nuevos grupos por cercanía. Para visualizar el funcionamiento de la clusterización, se suele utilizar el **dendograma.** En este diagrama, cada clase es visualizada por una línea vertical y la selección del número de clases idóneas es estimada a partir del diagrama.

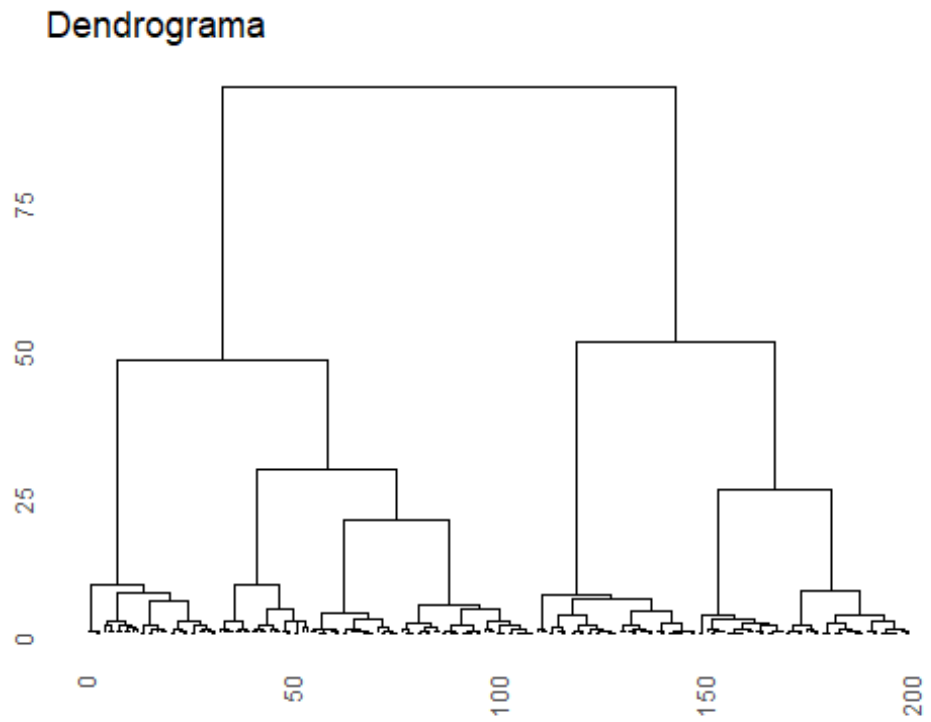
Los métodos llamados `Hierarchical Clustering`, y tienen la ventaja de que no requieren conocer de antemano el número de clústers a utilizar y, además utilizan solamente la matriz de distancia, por lo que son independientes de las observaciones.

Como parámetros críticos de configuración encontramos la función utilizada para el cálculo de la distancia, y el método de agrupamiento.

A continuación podemos ver el dendograma del dataset normalizado del Ejercicio 3, utilizando la **distancia euclidiana** y como método de agregación **el criterio de Ward**, basado en el decrecimiento de la varianza entre los grupos que van siendo mezclados.

```
library(ggdendro)
dendrogram <- hclust(dist(df.scale, method = 'euclidean'), method = 'ward
```

```
.D')
ggdendrogram(dendrogram, rotate = FALSE, labels = FALSE, theme_dendro = T
RUE) +
  labs(title = "Dendrograma")
```



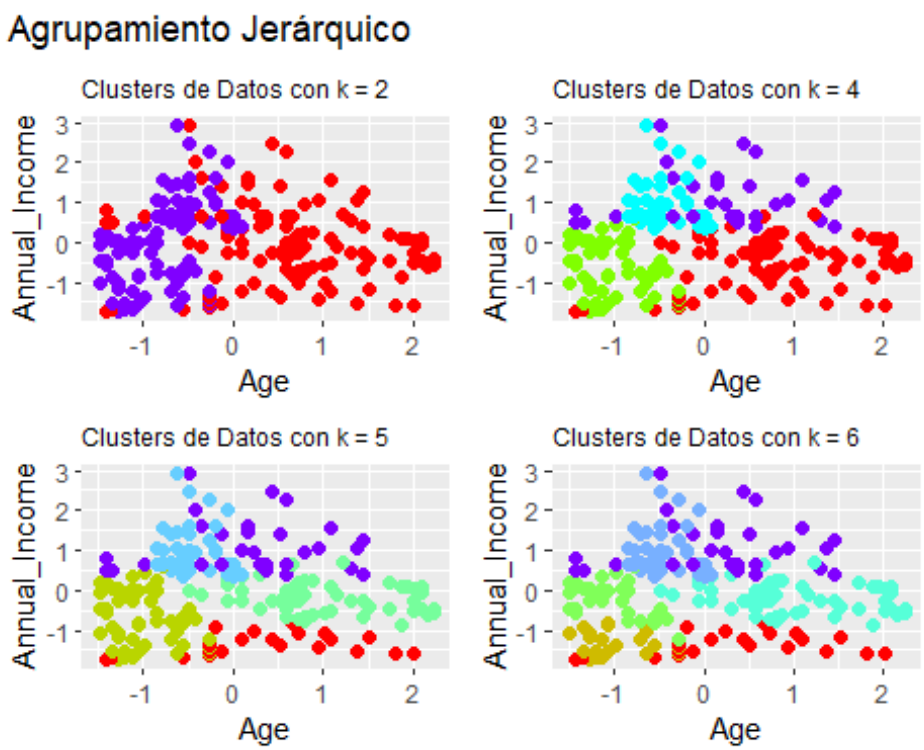
Fijándonos en el dendograma, encontramos las primeras agrupaciones formadas por los número de clases 2, 4, 5 y 6.

```
k <- c(2,4,5,6)
agrupamientoJ <- hclust(dist(df.scale, method = 'euclidean'), method = 'w
ard.D')
for (i in k){
  clases_aj <- cutree(agrupamientoJ, k = i)
  df.scale[, paste0('cluster.', i)] <- clases_aj
}
```

Una vez generado los cluster, vamos a verlos representados

```
library(patchwork)
p2 <- ggplot() + geom_point(aes(x = Age, y = Annual_Income_.k.., color =
cluster.2), data = df.scale, size = 2, show.legend = F) +
  scale_colour_gradientn(colours=rainbow(4)) +
  ggtitle('Clusters de Datos con k = 2') +
  theme(plot.title = element_text(size=9)) +
  xlab('Age') + ylab('Annual_Income')

p4 <- ggplot() + geom_point(aes(x = Age, y = Annual_Income_.k.., color =
```

```
cluster.4), data = df.scale, size = 2, show.legend = F) +
  scale_colour_gradientn(colours=rainbow(4)) +
  ggtitle('Clusters de Datos con k = 4') +
  theme(plot.title = element_text(size=9)) +
  xlab('Age') + ylab('Annual_Income')
p5 <- ggplot() + geom_point(aes(x = Age, y = Annual_Income_.k.., color =
cluster.5), data = df.scale, size = 2, show.legend = F) +
  scale_colour_gradientn(colours=rainbow(4)) +
  theme(plot.title = element_text(size=9)) +
  ggtitle('Clusters de Datos con k = 5') +
  xlab('Age') + ylab('Annual_Income')

p6 <- ggplot() + geom_point(aes(x = Age, y = Annual_Income_.k.., color =
cluster.6), data = df.scale, size = 2, show.legend = F) +
  scale_colour_gradientn(colours=rainbow(4)) +
  theme(plot.title = element_text(size=9)) +
  ggtitle('Clusters de Datos con k = 6') +
  xlab('Age') + ylab('Annual_Income')

(p2 | p4 ) / (p5 | p6) +  plot_annotation(title = 'Agrupamiento Jerárquic
o')
```



Aunque con una separación de dos grupos no se llega a obtenerse una separación perfecta entre los grupos, no vemos mejora al aumentar el numero de clústeres.

Referencias Externas:

- https://rpubs.com/rdelgado/399475

- https://es.wikipedia.org/wiki/Agrupamiento_jer%C3%A1rquico

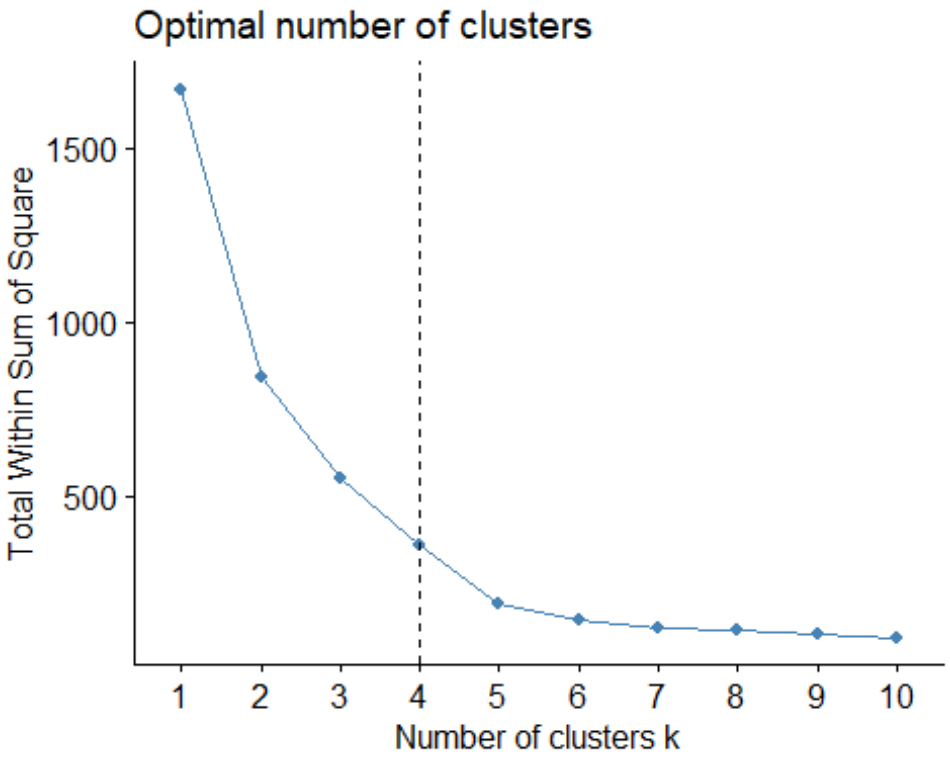
Clusterización mediante K-menoids (PAM)

Es un método de agrupación muy similar al K-means, pero mientras que en éste, el centroide constituye el elemento central del clúster, en K-menoids cada clúster se encuentra representado por una observación, el **menoid**, elemento del clúster cuya distancia promedio entre él y el resto de los elementos que integran la agrupación es mínima, y que constituye el elemento central.

Con respecto a K-means, es un algoritmo más robusto y que se ve menos influenciado por el ruido presente en la serie. Como desventaja, encontramos que necesita de muchos recursos computacionales, por lo que en está desaconsejado en el caso de datasets grandes.

Vamos a evaluar la reducción de la varianza para un valor máximo de k de 10. Al no apreciarse outliers en el dataset, utilizamos la distancia euclídea. Como estimador utilizamos la reducción de la suma total de los cuadrados de las diferencias internas.

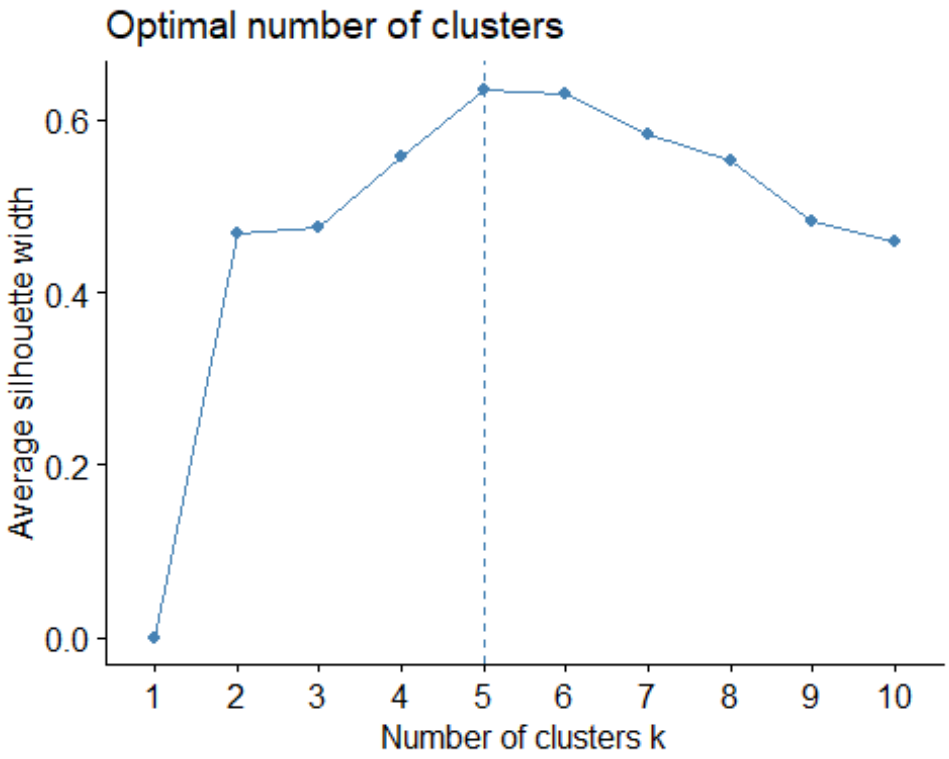
```
library(cluster)
library(factoextra)

## Welcome! Want to learn more? See two factoextra-related books at https
://goo.gl/ve3WBa

fviz_nbclust(x = df.scale, FUNcluster = pam, method = "wss", k.max = 10,
             diss = dist(df.scale, method = "euclidean")) +
geom_vline(xintercept = 4, linetype = 2)
```



Se aprecia que el mejor valor encontrado corresponde para un valor de k de 4, que es en donde la curva parece que comienza a suavizar su pendiente.

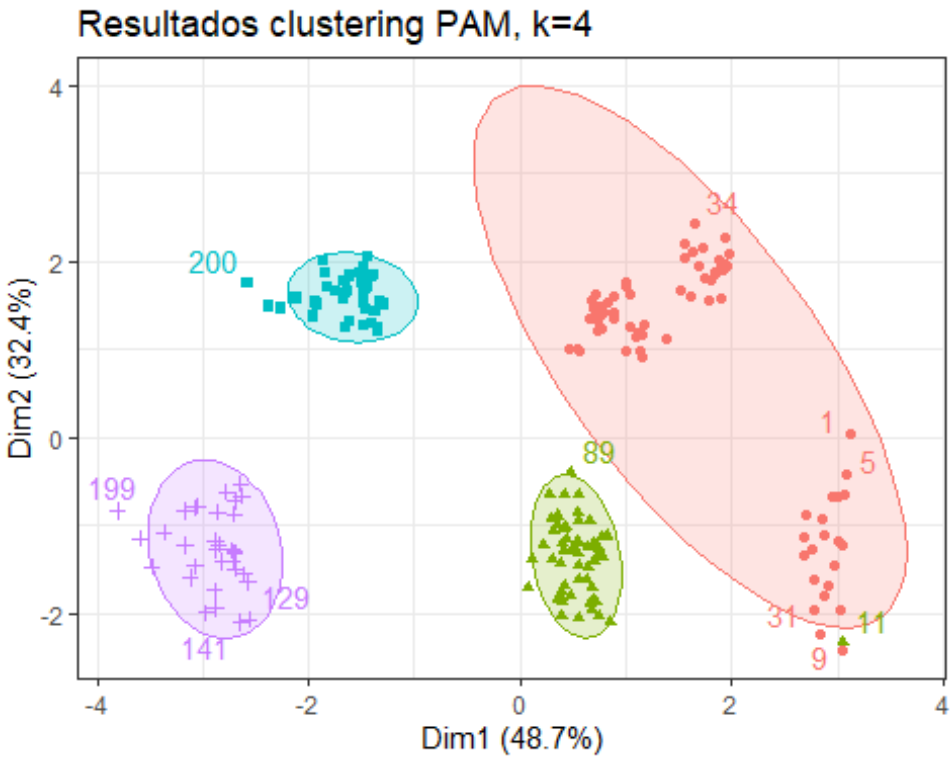
Vamos a probar también con "**silhouette**" como estimador, tal y como hicimos con el método k-means del ejercicio 3.

```
fviz_nbclust(x = df.scale, FUNcluster = pam, method = "silhouette", k.max
= 10,
             diss = dist(df.scale, method = "euclidean"))
```



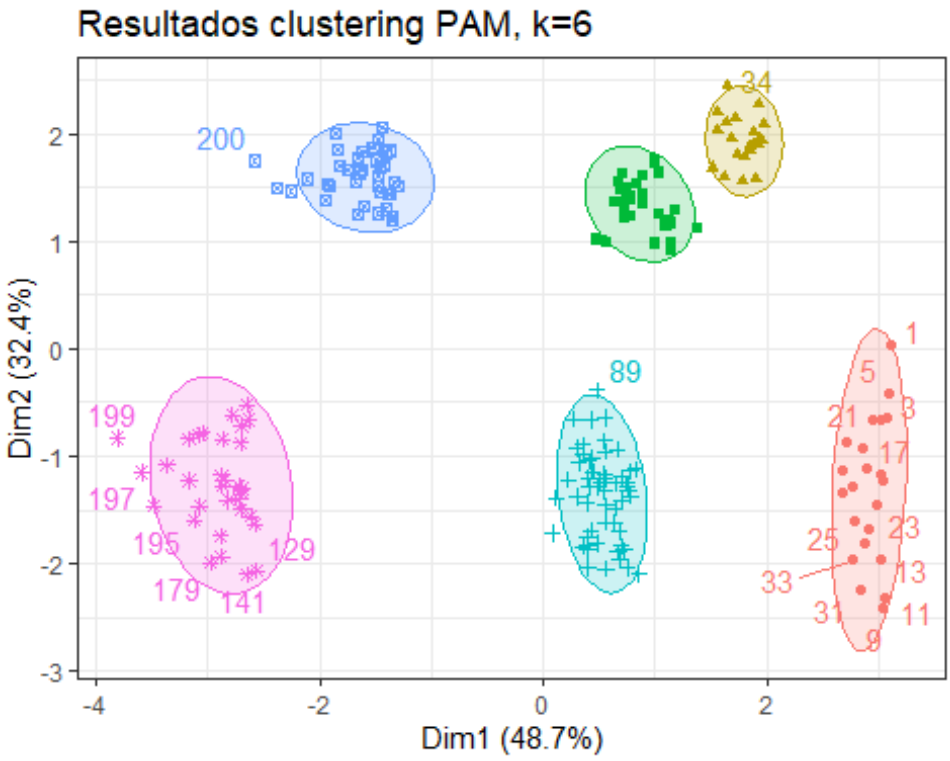
En este caso, vemos que el mejor número de agrupaciones encontrado corresponde a 6.

```
set.seed(1234)

pam_clusters.k4 <- pam(x=df.scale, k= 4, metric = "euclidean")
pam_clusters.k6 <- pam(x=df.scale, k= 6, metric = "euclidean")

fviz_cluster(object = pam_clusters.k4, data = df.scale, ellipse.type = "t
",
             repel = TRUE) +
  theme_bw() +
  labs(title = "Resultados clustering PAM, k=4") +
  theme(legend.position = "none")
```



El dataset queda muy bien catalogado en cuatro grupos. Vamos a visualizar lo que se obtiene con 6 clusteres.

```
fviz_cluster(object = pam_clusters.k6, data = df.scale, ellipse.type = "t
",
             repel = TRUE) +
  theme_bw() +
  labs(title = "Resultados clustering PAM, k=6") +
  theme(legend.position = "none")
```



Parece que con este número de agrupaciones se ha conseguido una mejor diferenciación entre los elementos que componen los grupos, sobretodo en el centro.

Clusterización mediante CLARA

Hemos visto que una de las debilidades de K-menoids se encuentra en el consumo de recursos. El algoritmo de CLARA (***Clustering Large Applications***) combina el concepto de **K-menoids** con el **resampling**, utilizando muestras aleatorias de un tamaño definido y aplicando el algoritmo de PAM (K-menoids). Este funcionamiento permite que sea un algoritmo útil para que pueda usar en largos datasets.

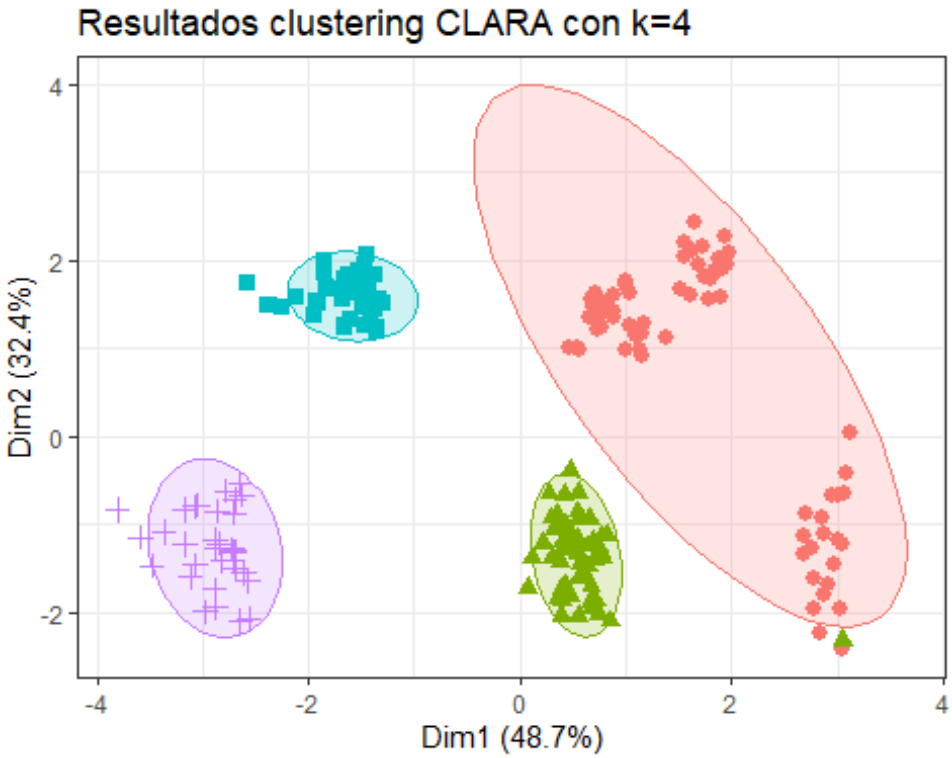
Lanzamos el dataset según los valores de k encontrados anteriormente, 4 y 6.

```
set.seed(1234)

clara_clusters.k4 <- clara(x = df.scale, k = 4, metric = "euclidean", sta
nd = F,
                           samples = 50, pamLike = TRUE)
clara_clusters.k6 <- clara(x = df.scale, k = 6, metric = "euclidean", sta
nd = F,
                           samples = 50, pamLike = TRUE)
```

Como realizamos anteriormente, vamos a visualizar los resultados para cuatro agrupaciones

```
fviz_cluster(object = clara_clusters.k4, ellipse.type = "t", geom = "poin
t",
```

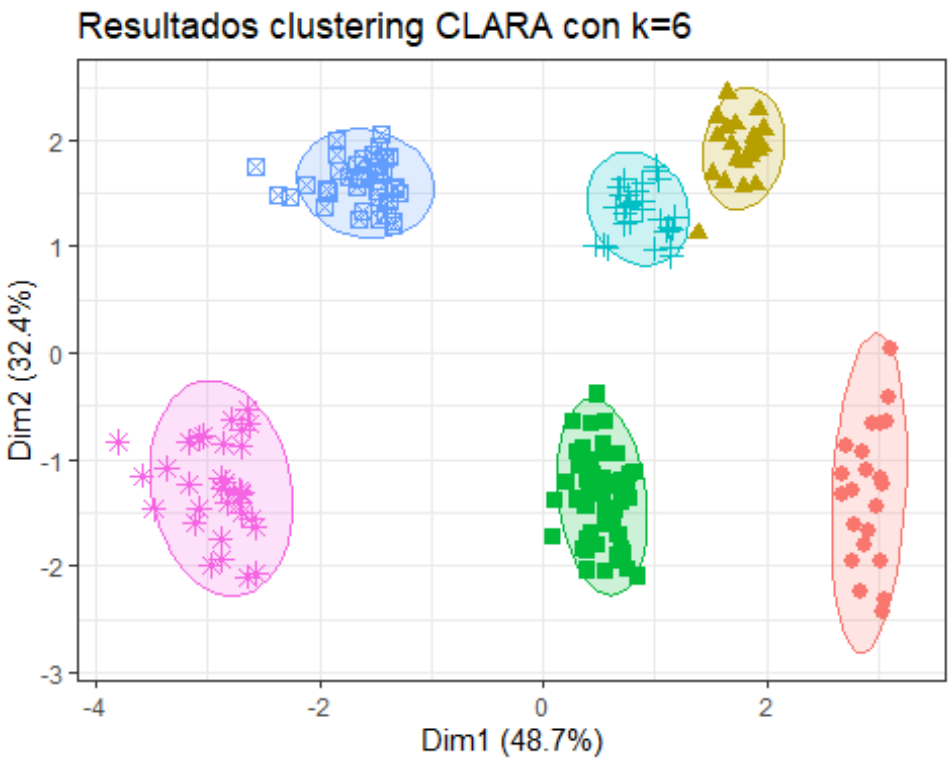
```
            pointsize = 2.5) +
  theme_bw() +
  labs(title = "Resultados clustering CLARA con k=4") +
  theme(legend.position = "none")
```



Vemos un resultado muy parecido al obtenido con k-menoids, lógico si sabemos que se basa en el mismo concepto.

Probemos con el siguiente valor de agrupaciones.

```
fviz_cluster(object = clara_clusters.k6, ellipse.type = "t", geom = "poin
t",
             pointsize = 2.5) +
  theme_bw() +
  labs(title = "Resultados clustering CLARA con k=6") +
  theme(legend.position = "none")
```



Si comparamos los resultados obtenidos no apreciamos una diferencia importante, incluso algo peores que con PAM si vemos los puntos del clúster inferior izquierda.

Clusterización mediante DBSCAN

La idea detrás del algoritmo de cluesterización "**`Density based clustering`**" (DBSCAN), está en realizar las agrupaciones por regiones, según la densidad de las observaciones.

A diferencia del resto de algoritmos de particionado que fallan al identificar formas arbitrarias, **DBSCAN** agrupa las observaciones en el cluster partiendo del principio de que tiene que haber un mínimo de observaciones vecinas dentro de un radio próximo y de que los clústeres han de estar separados por regiones vacías o con pocas observaciones.

Como **ventajas** del algoritmo DBSCAN, tenemos que al contrario que los algoritmos vistos hasta ahora, no se requiere conocer el número de clusters. Además, es independiente de la forma, no estando obligado a que sigan una forma redonda alrededor de un punto como K-means o K-menoids. Esto permite identificar a los outliers, por lo que es una algoritmo recomendado en datasets con ruido.

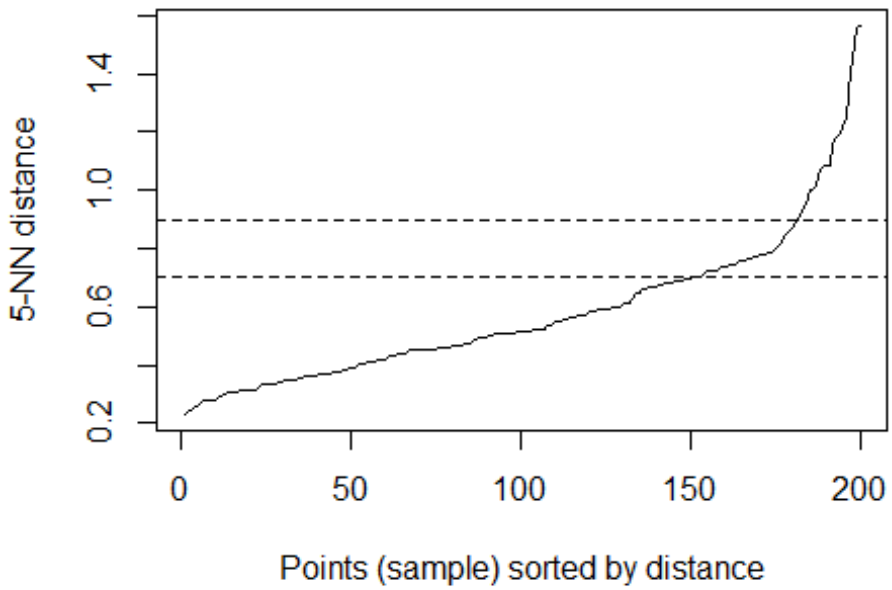
Por el contrario, tenemos que es un algoritmo totalmente determinístico, teniendo mucha influencia en la asignación del cluster el orden de los datos. No funciona bien cuando la densidad de los puntos de observación no tiende a ser homogénea, ya que en ese caso se vuelve complejo encontrar los valores paramétricos utilizados por el algoritmo.

El algoritmo necesita los siguientes parámetros de entrada:

- **Épsilon** (ϵ): radio que define la región vecina a una observación, también llamada ϵ-neighborhood.
- **Minimum points** (minPts): El mínimo número de observaciones dentro de la región épsilon. Cuanto mayor sea el tamaño del dataset, más grande ha de ser este valor, no bajando nunca de 3.

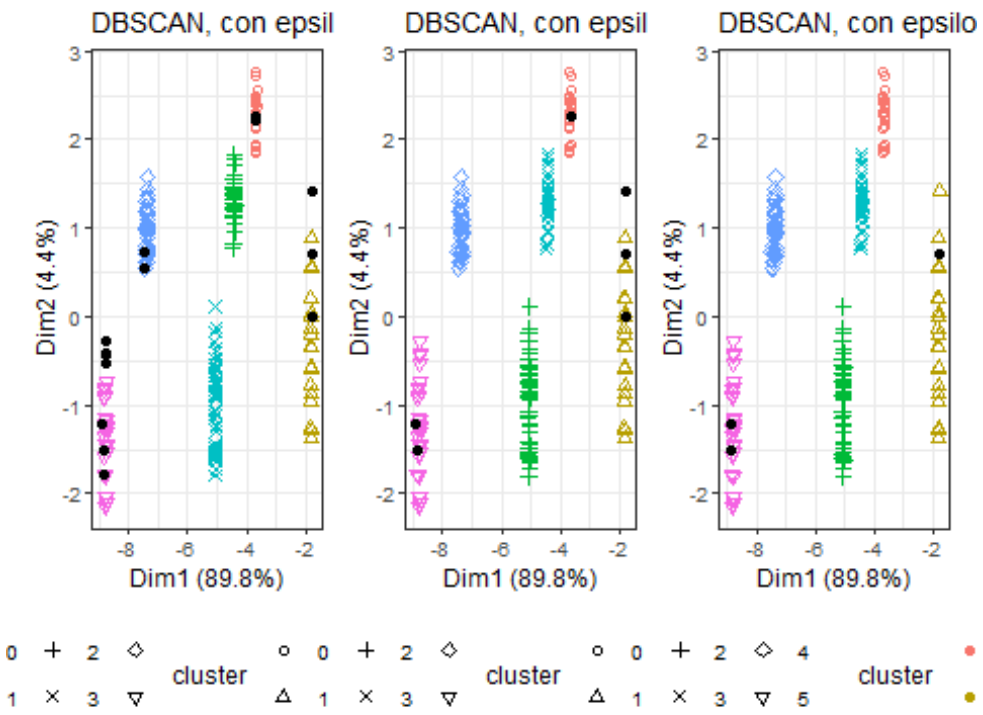
Vamos a encontrar el valor óptimo para épsilon, utilizando 5 para minPts (se ha probado con diferentes valores entre 2 y 10, no encontrando diferencia aparente).

```
set.seed(1234)
library(fpc)
library(dbscan)

##
## Attaching package: 'dbscan'

## The following object is masked from 'package:fpc':
##
##     dbscan

minpts <- 5
kNNdistplot(df.scale, k = minpts)
abline(h = 0.9, lty = 2)
abline(h = 0.7, lty = 2)
```



No queda muy claro que valor de épsilon escoger. Estaría entre 0.7 y 0.9, por lo que vamos a probar con diferentes valores y ver el resultado.

```
e <- c(0.7, 0.8, 0.9)
dbscan_cluster.e1 <- fpc::dbscan(data = df.scale, eps = e[1], MinPts = mi
npts)
dbscan_cluster.e2 <- fpc::dbscan(data = df.scale, eps = e[2], MinPts = mi
npts)
dbscan_cluster.e3 <- fpc::dbscan(data = df.scale, eps = e[3], MinPts = mi
npts)

p1 <- fviz_cluster(object = dbscan_cluster.e1, data = df.scale, stand = F
ALSE,
             geom = "point", ellipse = FALSE, show.clust.cent = FALSE,
             pallete = "jco") +
  labs(title = paste0("DBSCAN, con epsilon= ",e[1])) +
  theme_bw(base_size = 9) +
  theme(legend.position = "bottom")

p2 <- fviz_cluster(object = dbscan_cluster.e2, data = df.scale, stand = F
ALSE,
             geom = "point", ellipse = FALSE, show.clust.cent = FALSE,
             pallete = "jco") +
  labs(title = paste0("DBSCAN, con epsilon= ",e[2])) +
  theme_bw(base_size = 9) +
  theme(legend.position = "bottom")

p3 <- fviz_cluster(object = dbscan_cluster.e3, data = df.scale, stand = F
ALSE,
             geom = "point", ellipse = FALSE, show.clust.cent = FALSE,
             pallete = "jco") +
  labs(title = paste0("DBSCAN, con epsilon= ",e[3])) +
  theme_bw(base_size = 9) +
  theme(legend.position = "bottom")


p1 + p2 + p3
```



Ya vemos la diferencia visual con respecto al resto de algoritmos vistos hasta el momento. Pero para saber cuál de los tres ha dado mejor resultado, miraremos sus números.

```
print.dbscan(dbscan_cluster.e1)

## dbscan Pts=200 MinPts=5 eps=0.7
##          0  1  2  3  4  5  6
## border 14  0  8  2  3  2  6
## seed    0 19 11 29 52 35 19
## total  14 19 19 31 55 37 25

print.dbscan(dbscan_cluster.e2)

## dbscan Pts=200 MinPts=5 eps=0.8
##         0  1  2  3  4  5  6
## border 6  1  4  0  1  2  7
## seed   0 19 15 55 30 37 23
## total  6 20 19 55 31 39 30

print.dbscan(dbscan_cluster.e3)

## dbscan Pts=200 MinPts=5 eps=0.9
##         0  1  2  3  4  5  6
## border 3  2  4  0  1  2  5
## seed   0 19 17 55 30 37 25
## total  3 21 21 55 31 39 30
```

El número de columna corresponde al número del clúster. DBScan reserva el valor 0 para los outliers (en negro).

Conforme se disminuye el valor de épsilon, al algoritmo le cuesta más diferenciar los clústeres, siendo el valor de 0,9 el que obtiene un diferenciación más alta entre los grupos de puntos, dejando fuera a sólo 3 observaciones sin grupo. Jugando con el valor de épsilon, podríamos afinar más el algoritmo hasta conseguir que todas las observaciones queden agrupadas.

Referencia externa:
https://www.cienciadedatos.net/documentos/37_clustering_y_heatmaps#Introducci%C3%B3n

Criterios de evaluación

Ejercicio 1 (20%)

- 60%. Explicar de forma resumida cual es el objetivo de los métodos de agregación. Relacionar la respuesta con un ejemplo.
- 40%. Razonar si tiene sentido aplicar un método agregación al ejemplo que pusiste en la PEC1. Si lo tiene, comentar como se podría aplicar y que tipo de resultados se podrían obtener. Si no lo tiene, reformular el problema para que si lo tenga.

Ejercicio 2 (20%)

- 60%. Explicar de forma resumida cual es el objetivo de los métodos de generación de reglas de asociación. Relacionar la respuesta con un ejemplo.
- 40%. Razonar si tiene sentido aplicar un método de generación de reglas de asociación al ejemplo que pusiste en la PEC1. Si lo tiene, comentar como se podría aplicar y que tipo de resultados se podrían obtener. Si no lo tiene, reformular el problema para que si lo tenga.

Ejercicio 3 (25%)

- 20%. Se explican los campos de la base de datos, preparación y análisis de datos
- 20%. Se aplica el algoritmo de agrupamiento de forma correcta y se prueban con diferentes valores de k.
- 10%. Se obtiene una medida de lo bueno que es el agrupamiento.
- 40%. Se ponen nombres a las asociaciones y se describen e interpretan los diferentes clústers obtenidos.
- 10%. Se presenta el código y es fácilmente reproducible.

Ejercicio 4 (25%)

- 10%. Se realiza un resumen de los datos incluidos en la base de datos.
- 15%. Se preparan los datos de forma correcta.

- 10%. Se aplica el algoritmo de reglas de asociación.
- 20%. Se realizan diferentes pruebas variando algunos parámetros.
- 35%. Se explican las conclusiones que se obtienen.
- 10%. Se presenta el código y es fácilmente reproducible.

Ejercicio 5 (10%)

- 25%. Se prueba un algoritmo diferente al kmeans.
- 25%. Se prueba otro algoritmo diferente al kmeans.
- 40%. Se comparan los resultados del kmeans y los otros dos métodos probados en este ejercicio.
- 10%. Se presenta el código y es fácilmente reproducible.