Fundamentos de la Ciencia de Datos

Prueba de Laboratorio 1 (PL1)

Jorge Revenga Martín de Vidales
Ángel Salgado Aldao
Adrián García

Grado en Ingeniería Informática
Universidad de Alcalá

14 de noviembre de 2023

Índice

1. Introducción - Consideraciones previas

1.1. Funciones básicas

Para utilizar una función en R se escribe el nombre de la función, seguido de los parámetros de entrada entre paréntesis e.g.: `función(parámetros)`

- Función `contributors()`: Muestra los creadores del programa (R)
- Función `help()`: Abre un HTML con información sobre la función `help()` o de la función entre paréntesis de haberla. Para todas las funciones que programemos (para todas las que existan) en R debe poder usarse la función `help()`.

  En el archivo HTML se distinguen varios elementos:

  - función {paquete}: la función de la que se obtiene información seguida del paquete al que pertenece.
  - Description: descripción de la función.
  - Usage: aparece la función y todos los argumentos que se le pueden introducir.
  - Arguments: Explicación de los argumentos o parámetros.
  - Details: Detalles adicionales de la función.
  - Offline help: Ayuda sin conexión.
  - Note: Nota del autor.
  - References: Referencias.
  - Examples: Ejemplos de uso de la función.
- Función `getwd()` se utiliza para obtener el directorio de trabajo actual (working directory).
- Función `setwd("C:/...")` permite cambiar el nuevo directorio de trabajo en el que queramos trabajar.
- `help.start()`: Manda a un compendio de todas las ayudas disponibles para trabajar con R.
- Función `list.files()`: Muestra todos los archivos en el directorio. `dir()` hace lo mismo.

1.2. Introducción de un archivo .txt en R

Para introducir una tabla desde un archivo `.txt` en R con seguridad de que vaya a cargar correctamente se deben seguir las siguientes reglas (pueden no ser todas necesarias en todos los casos).

- Debe haber una tabulación entre dato y dato, más tabulaciones no lo rompen del todo.
- Debe haber una primera columna que numere las filas. Excepto en la primera fila, que sólo habrá una tabulación, seguida de los nombres de las variables separados por tabulaciones.
- Hay que introducir un enter al final del documento.
- Los números decimales deben ser introducidos con puntos, no con comas.
- Los datos tienen que ir juntos, los espacios deben ser separados con guiones u otros símbolos.

1.3. Uso de Sweave

Vamos a generar un archivo pdf con código R embedido en el mismo de forma automática, para esto haremos uso de Sweave.

- `rnwfile <- system.file("Sweave", "example-1.Rnw", package ="utils")`: obtiene la ubicación del archivo "example-1.Rnw" que está incluido en el paquete "utils" y la almacena en la variable "rnwfile".
- `Sweave(rnwfile)`: Genera un conjunto de documentos nuevos, entre ellos un .txt, un .pdf y demás, a partir de la variable definida.
- `tools::texi2pdf("example-1.tex")`: Genera el .tex de sweave en un pdf. No funciona sin un compilador de latex (como miktex) en la máquina. Al descargar latex se puede seleccionar una opción para que se realice la carga automáticamente.

1.4. Paquetes por defecto al abrir R

- `getOption("defaultPackages")`: muestra los (6) paquetes por defecto que cargan al abrir R.
- `library(help="base")`: Muestra las funciones del paquete "base"junto con una descripción básica, `library(help="utils")` muestra funciones útiles para trabajar con R. Estos son parte de los 23 paquetes instalados por defecto.
- `library()`: Muestra los paquetes en la biblioteca (los instalados manualmente + los de la biblioteca estándar).
- `library(paquete)`: Carga el paquete, también podemos hacerlo si nos vamos a paquetes/cargar paquete.

1.5. Instalar paquetes nuevos

- Nos vamos a paquetes/seleccionar el espejo CRAN.
- Seleccionamos el repositorio que queremos utilizar y elegimos el paquete (También se puede instalar con `install.packages("paquete")`).
- También se puede instalar desde el dispositivo local, para ello hay que irse a la página oficial de CRAN.
  - Para aprender se usan las viñetas y luego el manual.
  - Se descarga la versión r-release.
  - Se crea un directorio temporal (temp) y se mete ahí el zip descargado.
  - `install.packages("c:direccion_al_zip/tmp/nombre.zip",repos=NULL)`: lo instala.
- Para instalar el paquete Arules vamos a descargarlo, también hay que descargar el manual y las viñetas, luego usar `install.packages()` para las dependencias.

2. Ejercicios con ayuda del profesor

Realización de cuatro ejercicios con ayuda del profesor en los que se van a realizar, utilizando el entorno R, un análisis de descripción de datos, un análisis de asociación y dos análisis de detección de datos anómalos, aplicando todos los conceptos teóricos vistos en cada lección.

2.1. Análisis de descripción de datos

El primer conjunto de datos, que se empleará para realizar el análisis de descripción de datos, estará formado por datos de una característica cualitativa, nombre, y otra cuantitativa, Radio, de los satélites menores de Urano, es decir, aquellos que tienen un Radio menor de 50 Km, dichos datos, los primeros cualitativos nominales, y los segundos cuantitativos continuos, son: (Nombre, Radio en Km): Cordelia, 13; Ofelia, 16; Bianca, 22; Crésida, 33; Desdémona, 29; Julieta, 42; Rosalinda, 27; Belinda, 34; Luna-1986U10, 20; Calíbano, 30; Luna-999U1, 20; Luna 1999U2, 15.

Solución:

- `s<-read.table("satelites.txt")`: Se asigna los valores de la tabla satelites (almacenada en el directorio de trabajo) a la variable `s`. Al teclear introducir el nombre de la variable como comando se deberían mostrar los datos.

```
> s<-read.table("satelites.txt")
> s

         Nombre Radio
1      Cordelia    13
2        Ofelia    16
3        Bianca    22
```

```
4       Crésida    33
5     Desdémona    29
6       Julieta    42
7     Rosalinda    27
8       Belinda    34
9  Luna-1986U10    20
10     Calíbano    30
11   Luna-999U1    20
12  Luna-1999U2    15
```

- `dim(s)`: Devuelve las dimensiones de una matriz. En este caso `[1] 12 2` quiere decir 12 filas, 2 columnas.

```
> dim(s)

[1] 12  2
```

- `so=s[order(s$Radio),]`: ordena las filas de un DataFrame `s` según los valores de la variable "Radio". Para hacerlo, se usa la función `order()` para obtener el orden de las filas en función de los valores de "Radio", y luego se aplica ese orden al DataFrame 's' utilizando los corchetes `[ ]`. Como resultado, `so` contendrá las filas de s ordenadas según los valores de "Radio".

```
> so<-s[order(s$Radio),]
> so

          Nombre Radio
1       Cordelia    13
12  Luna-1999U2    15
2         Ofelia    16
9  Luna-1986U10    20
11   Luna-999U1    20
3         Bianca    22
7     Rosalinda    27
5     Desdémona    29
10     Calíbano    30
4       Crésida    33
8       Belinda    34
6       Julieta    42
```

- `so=s[rev(order(s$Radio)),]`: Añadiendo `rev()` obtenemos las filas en orden decreciente

```
> so<-s[rev(order(s$Radio)),]
> so

          Nombre Radio
6       Julieta    42
8       Belinda    34
4       Crésida    33
```

```
10     Calíbano    30
5     Desdémona    29
7     Rosalinda    27
3        Bianca    22
11   Luna-999U1    20
9  Luna-1986U10    20
2        Ofelia    16
12  Luna-1999U2    15
1      Cordelia    13
```

- `length(s$Radio)`: Devuelve la longitud de la columna

  ```
  > length(s$Radio)

  [1] 12
  ```

- Para realizar el cálculo del rango:

  - `rangor=max(s$Radio)-min(s$Radio)` : siguiendo lo visto en teoría, el resultado es 29

    ```
    > rangor=max(s$Radio)-min(s$Radio)
    > rangor

    [1] 29
    ```

  - `range(s$Radio)`: devuelve el número menor y el mayor, no la diferencia. Queremos una función que calcule el rango, por lo que primero la creamos utilizando el comando `function()`.

    ```
    > range(s$Radio)

    [1] 13 42
    ```

  - Podemos definir la variable Radio=s$Radio para ahorrar tiempo

    ```
    > Radio = s$Radio
    ```

  - `rango=function(Radio){max(Radio)-min(Radio)}`: En los paréntesis, especificamos los argumentos que la función recibirá, y entre llaves, definimos las instrucciones que llevará a cabo.

    ```
    > rango=function(Radio){max(Radio)-min(Radio)}
    > rango(Radio)

    [1] 29
    ```

  - `dump("rango", file="rango.R")`: empleamos la función dump para guardar nuestra función en el archivo "rango.R". Para utilizarla posteriormente, la cargaremos utilizando el comando source (`source("rango.R")`).

    ```
    > dump("rango", file="rango.R")
    > source("rango.R")
    ```

- `frecabsradio<-table(s$Radio)`: calcula la frecuencia absoluta para todos los valores

```
> frecabsradio<-table(s$Radio)
> frecabsradio

13 15 16 20 22 27 29 30 33 34 42
 1  1  1  2  1  1  1  1  1  1  1
```

- `frecabscumradio<-cumsum(s$Radio)`: calcula la frecuencia absoluta acumulada para todos los valores

```
> frecabsacumradio<-cumsum(frecabsradio)
> frecabsacumradio

13 15 16 20 22 27 29 30 33 34 42
 1  2  3  5  6  7  8  9 10 11 12
```

- `frecrel<-function(x){table(x)/length(x)}`: no tenemos función para la frecuencia relativa, por lo que la definimos. La función se puede escribir de forma diferente, ya que en este caso 'Radio' no es una referencia a una variable sino un parámetro de entrada de la función, por lo que `frecrel<-function(x){table(x)/length(x)}` serviría de igual manera. `x=s$Radio` y `frecrelradio=frecrel(x)` calcula usando la función definida.

```
> frecrel<-function(x){table(x)/length(x)}
> dump("frecrel", file="frecrel.R")
> source("frecrel.R")
> x=s$Radio
> frecrelradio=frecrel(x)
> frecrelradio

x
        13         15         16         20         22         27         29
0.08333333 0.08333333 0.08333333 0.16666667 0.08333333 0.08333333 0.08333333
        30         33         34         42
0.08333333 0.08333333 0.08333333 0.08333333
```

- `frecrelacumradio<-cumsum(frecrelradio)`: no tenemos función para la frecuencia relativa acumulada tampoco, se puede calcular usando `cumsum()` de la frecuencia relativa

```
> frecelacumradio <- cumsum(frecrelradio)
> frecelacumradio

        13         15         16         20         22         27         29
0.08333333 0.16666667 0.25000000 0.41666667 0.50000000 0.58333333 0.66666667
        30         33         34         42
0.75000000 0.83333333 0.91666667 1.00000000
```

- `mr<-mean(Radio)`: Calcula la media

```
> mr<-mean(s$Radio)
> mr

[1] 25.08333
```

- `sdr<-sd(Radio)`: Para la desviación estándar se usa la función sd, el problema yace en que esta función realiza la desviación típica muestral (la cual divide entre N-1 y no entre N) para realizar inferencias probabilísticas. Hay casos como es el de este ejercicio que tenemos todos los datos (la población entera) y nos interesa usar la desviación estándar poblacional

```
> sdr<-sd(s$Radio)
> sdr

[1] 8.857029
```

- `sdr=sqrt((sdr2̂)*11/12)`: creamos la función, quitándole la raíz cuadrada con el 2̂ para multiplicar después por N-1/N, para después añadirle la raiz cuadrada de nuevo. Este código puede ser fácilmente optimizado y definido en una función para su reutilización

```
> sdr=sqrt((sdr^2)*11/12)
> sdr

[1] 8.47996
```

- `varr=var(Radio)`: Calcula la varianza (en este caso se trata también de la varianza muestral)

```
> varr=var(x)
> varr

[1] 78.44697
```

- `varr=varr*11/12`: Calculamos la varianza poblacional de la misma forma que la desviación típica, sin tener que preocuparnos por la raíz cuadrada

```
> varr=varr*11/12
> varr

[1] 71.90972
```

- `Medianr<-median(s$Radio)`: Para la mediana usamos la función `median()`, pero esta función usa distribuciones de probabilidad en vez de usar ecuaciones y muchas veces no dará el resultado correcto

```
> medianr<-median(x)
> medianr

[1] 24.5
```

- `cuart1=quantile(Radio, 0.25)`: Calcula el primer cuartil, puede usarse para calcular cualquier centil (0.%). Le ocurre el mismo problema que a la mediana

```
> cuartl=quantile(x,0.25)
> cuartl

25%
 19
```

2.2. Análisis de asociación

El segundo conjunto de datos, que se empleará para realizar el análisis de asociación, estará formado por las siguientes 6 cestas de la compra: {Pan, Agua, Leche, Naranjas}, {Pan, Agua, Café, Leche}, {Pan, Agua, Leche}, {Pan, Café, Leche}, {Pan, Agua}, {Leche}.

Solución

- `library(arules)`: Carga el paquete arules.
- `search()`: Muestra los paquetes cargados.
- `library(Matrix)`: Carga el paquete Matrix.
- `muestra<-Matrix(c(1,1,0,1,1, 1,1,1,1,0, 1,1,0,1,0, 1,0,1,1,0, 1,1,0, 0,0, 0,0,0,1,0) ,6,5, byrow=TRUE, dimnames=list(c(“suceso1” , “suceso2” , “suceso3” , “suceso4” , “suceso5” , “suceso6” ), c(“Pan” , “Agua” , “Café” , “Leche” , “Naranjas”)), sparse=TRUE)`: Carga los datos del problema
- `muestrangCMatrix<-as(muestra,"nsparseMatrix")`: utilizamos la función as para convertir el objeto muestra a una representación de matriz dispersa (sparse matrix)
- `traspmuestrangCMatrix<-t(muestrangCMatrix)`: Trasponemos y tenemos la matriz como arules nos la pide.
- `transacciones<-as(traspmuestrangCMatrix,"transactions")`.
- `summary(transacciones)`: Muestra más información.
- `asociaciones<-apriori (transacciones, parameter=list (support = 0.5, confidence = 0.8))`: Aplica el algoritmo apriori con umbral de soporte de 0.5 y de confianza de 0.8
- `inspect(asociaciones)`: Muestra las asociaciones que pasan el algoritmo.

```
> library(arules)
> search()
```

```
 [1] ".GlobalEnv"         "package:Rlof"       "package:doParallel"
 [4] "package:parallel"   "package:iterators"  "package:foreach"
 [7] "package:arules"     "package:Matrix"     "package:stats"
[10] "package:graphics"   "package:grDevices"  "package:utils"
[13] "package:datasets"   "package:methods"    "Autoloads"
[16] "package:base"
```

```
> library(Matrix)
> muestra<-Matrix(c(1,1,0,1,1, 1,1,1,1,0, 1,1,0,1,0, 1,0,1,1,0, 1,1,0,
+ 0,0, 0,0,0,1,0),6,5,byrow=TRUE,dimnames=list(c("suceso1", "suceso2",
+ "suceso3", "suceso4", "suceso5", "suceso6"), c("Pan", "Agua", "Café",
+ "Leche", "Naranjas")), sparse=TRUE)
> muestrangCMatrix<-as(muestra, "nsparseMatrix")
> traspmuestrangCMatrix<-t(muestrangCMatrix)
> transacciones<-as(traspmuestrangCMatrix, "transactions")
> summary(transacciones)
```

```
transactions as itemMatrix in sparse format with
 6 rows (elements/itemsets/transactions) and
 5 columns (items) and a density of 0.5666667

most frequent items:
     Pan    Leche     Agua     Café Naranjas  (Other)
       5        5        4        2        1        0

element (itemset/transaction) length distribution:
sizes
1 2 3 4
1 1 2 2

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.250   3.000   2.833   3.750   4.000

includes extended item information - examples:
  labels
1    Pan
2   Agua
3   Café

includes extended transaction information - examples:
  itemsetID
1   suceso1
2   suceso2
3   suceso3
```

```
> asociaciones<-apriori (transacciones, parameter=list (support = 0.5,
+ confidence = 0.8))
```

```
Apriori
```

```
Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.8    0.1    1 none FALSE            TRUE       5     0.5      1
 maxlen target  ext
     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 3

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[5 item(s), 6 transaction(s)] done [0.00s].
sorting and recoding items ... [3 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 done [0.00s].
writing ... [7 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

> inspect(asociaciones)

```
    lhs              rhs     support   confidence coverage  lift count
[1] {}            => {Leche} 0.8333333 0.8333333  1.0000000 1.00 5
[2] {}            => {Pan}   0.8333333 0.8333333  1.0000000 1.00 5
[3] {Agua}        => {Pan}   0.6666667 1.0000000  0.6666667 1.20 4
[4] {Pan}         => {Agua}  0.6666667 0.8000000  0.8333333 1.20 4
[5] {Leche}       => {Pan}   0.6666667 0.8000000  0.8333333 0.96 4
[6] {Pan}         => {Leche} 0.6666667 0.8000000  0.8333333 0.96 4
[7] {Agua, Leche} => {Pan}   0.5000000 1.0000000  0.5000000 1.20 3
```

2.3. Análisis de detección de datos anómalos - Técnicas con base estadística

El tercer conjunto de datos, que se empleará para realizar el análisis de detección de datos anómalos utilizando técnicas con base estadística, estará formado por los siguientes 7 valores de resistencia y densidad para diferentes tipos de hormigón Resistencia, Densidad: 3, 2; 3.5, 12; 4.7, 4.1; 5.2, 4.9; 7.1, 6.1; 6.2, 5.2; 14, 5.3. Aplicar las medidas de ordenación a la resistencia y las de dispersión a la densidad.

Caja y Bigotes

- `(muestra=t(matrix(c(3,2,3.5,12,4.7,4.1,5.2,4.9,7.1,6.1,6.2,5.2,14,5.3),2,7, dimnames=list(c(r","d")))))`: cargamos los datos
- `(muestra=data.frame(muestra))`: Convertimos la matriz a un data.frame
- `(boxplot(muestra$r,range=1.5,plot=FALSE))`: Boxplot de forma predeterminada muestra gráficamente la resolución con caja y bigotes, por lo que se añade `plot=FALSE` para que no lo haga

- `(cuar1r<-quantile(muestra$r, 0.25))`: Cálculo del primer cuartil
- `(cuar3r<-quantile(muestra$r, 0.75))` Cálculo del tercer cuartil
- `(int=c(cuar1r-1.5*(cuar3r-cuar1r),cuar3r+1.5*(cuar3r-cuar1r)))`: calcula el rango de los datos que no son outliers
- `for(i in 1:length(muestra$r)) if(muestra$r[i]<int[1] || muestra$r[i]> int[2]) print("el suceso"); print(i); print("es un outlier")`

Ejecución

```
> (muestra=t(matrix(c(3,2,3.5,12,4.7,4.1,5.2,4.9,7.1,6.1,6.2,5.2,14,5.3)
+ ,2,7,dimnames=list(c("r","d")))))

        r    d
[1,]  3.0  2.0
[2,]  3.5 12.0
[3,]  4.7  4.1
[4,]  5.2  4.9
[5,]  7.1  6.1
[6,]  6.2  5.2
[7,] 14.0  5.3

> (muestra=data.frame(muestra))

     r    d
1  3.0  2.0
2  3.5 12.0
3  4.7  4.1
4  5.2  4.9
5  7.1  6.1
6  6.2  5.2
7 14.0  5.3

> (boxplot(muestra$r,range=1.5,plot=FALSE))

$stats
     [,1]
[1,] 3.00
[2,] 4.10
[3,] 5.20
[4,] 6.65
[5,] 7.10

$n
[1] 7

$conf
         [,1]
```

```
[1,] 3.677181
[2,] 6.722819

$out
[1] 14

$group
[1] 1

$names
[1] "1"
```

```
> (cuar1r<-quantile(muestra$r, 0.25))

25%
4.1

> (cuar3r<-quantile(muestra$r, 0.75))

 75%
6.65

> (int=c(cuar1r-1.5*(cuar3r-cuar1r),cuar3r+1.5*(cuar3r-cuar1r)))

   25%    75%
 0.275 10.475

> for(i in 1:length(muestra$r)) {if(muestra$r[i]<int[1] || muestra$r[i]>int[2])
+ {print("el suceso"); print(i); print("es un outlier")}}

[1] "el suceso"
[1] 7
[1] "es un outlier"
```

Desviación típica

- `sdd=sqrt(var(muestra$d)*length(muestra$d)-1)/length(muestra$d)`: calculamos la desviación estándar poblacional a partir del dataframe del apartado anterior y la función var() que calcula la varianza muestral
- `(intdes=c(mean(muestra$d)-2*sdd,mean(muestra$d)+2*sdd))`: calcula el rango de los datos que no son outliers
- `for(i in 1:length(muestra$d)) if (muestra$d[i]<intdes[1] || muestra$d[i]> intdes[2]) print(´´el suces”);print(i);print(muestra$d[i]); print(´´es un outlier”)`

Ejecución

```
> sdd=sqrt(var(muestra$d)*length(muestra$d)-1)/length(muestra$d)
> (intdes=c(mean(muestra$d)-2*sdd,mean(muestra$d)+2*sdd))

[1] 3.341975 7.972310

> for(i in 1:length(muestra$d))
+ {if (muestra$d[i]<intdes[1] || muestra$d[i]>intdes[2]){print("el suceso")
+ ;print(i);print(muestra$d[i]);print("es un outlier")}}

[1] "el suceso"
[1] 1
[1] 2
[1] "es un outlier"
[1] "el suceso"
[1] 2
[1] 12
[1] "es un outlier"
```

2.4. Análisis de detección de datos anómalos - Técnicas basadas en la proximidad y en la densidad

El cuarto conjunto de datos, que se empleará para realizar el análisis de detección de datos anómalos utilizando técnicas basadas en la proximidad y en la densidad, estará formado por las siguientes 5 calificaciones de estudiantes: 1. 4, 4; 2. 4, 3; 3. 5, 5; 4. 1, 1; 5. 5, 4 donde las características de las calificaciones son: (Teoría, Laboratorio).

Vecinos próximos

- `(muestra=matrix(c(4,4,4,3,5,5,1,1,5,4),2,5))`: obtenemos la matriz
- `(muestra=t(muestra))`: Trasponemos la matriz
- Calculamos las distancias euclídeas: `(distancias=as.matrix(dist(muestra)))`
- Hay que ordenar los valores `(distancias=matrix(distancias,5,5)) for (i in 1:5)distancias[,i]=sort(distancias[,i]); (distanciasordenadas = distancias)`
- Como el primer vecino es él mismo, la distancia es cero, por lo que vamos a usar k=4 `for(i in 1:5)if(distanciasordenadas[4,i]>2.5)print(i);print(“es un outlier”)`
- `(distanciasM=as.matrix(dist(muestra,method="manhattan")))`: Cálculo de distancias de Manhattan

Ejecución

```
> (muestra=matrix(c(4,4,4,3,5,5,1,1,5,4),2,5))

     [,1] [,2] [,3] [,4] [,5]
[1,]    4    4    5    1    5
[2,]    4    3    5    1    4
```

```
> (muestra=t(muestra))

     [,1] [,2]
[1,]    4    4
[2,]    4    3
[3,]    5    5
[4,]    1    1
[5,]    5    4

> (distancias=as.matrix(dist(muestra)))

         1        2        3        4        5
1 0.000000 1.000000 1.414214 4.242641 1.000000
2 1.000000 0.000000 2.236068 3.605551 1.414214
3 1.414214 2.236068 0.000000 5.656854 1.000000
4 4.242641 3.605551 5.656854 0.000000 5.000000
5 1.000000 1.414214 1.000000 5.000000 0.000000

> (distancias=matrix(distancias,5,5))

         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.000000 1.000000 1.414214 4.242641 1.000000
[2,] 1.000000 0.000000 2.236068 3.605551 1.414214
[3,] 1.414214 2.236068 0.000000 5.656854 1.000000
[4,] 4.242641 3.605551 5.656854 0.000000 5.000000
[5,] 1.000000 1.414214 1.000000 5.000000 0.000000

> for (i in 1:5){distancias[,i]=sort(distancias[,i])};
>  (distanciasordenadas=distancias)

         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.000000 0.000000 0.000000 0.000000 0.000000
[2,] 1.000000 1.000000 1.000000 3.605551 1.000000
[3,] 1.000000 1.414214 1.414214 4.242641 1.000000
[4,] 1.414214 2.236068 2.236068 5.000000 1.414214
[5,] 4.242641 3.605551 5.656854 5.656854 5.000000

> for(i in 1:5){if(distanciasordenadas[4,i]>2.5)
+ {print(i);print("es un suceso anómalo o outlier")}}

[1] 4
[1] "es un suceso anómalo o outlier"

> (distanciasM=as.matrix(dist(muestra,method="manhattan")))

  1 2 3 4 5
1 0 1 2 6 1
2 1 0 3 5 2
3 2 3 0 8 1
4 6 5 8 0 7
5 1 2 1 7 0
```

Local Outlier Factor Existen varios paquetes para hacer este cálculo que utilizan métodos distintos al visto en teoría: RLof, DDoutlier, DMwR son algunos de ellos.

- Parámetros:
  - `datos`: Matriz de valores númericos.
  - `k`: Número de orden k.
  - `dist`: Método de cálculo de las distancias.
- Retorno: Imprime por pantalla los valores lof de cada punto.
- Explicación: Utilizamos el paquete Rlof, el cual contiene una función llamada `lof(datos,k,dist)`, el cual recibe los puntos que va a evaluar, el número de vecinos cercanos (k) y el método que se emplea para calcular las distancias (al ser LOF usamos *manhattan*).
  Esta función hace uso de varias funciones externas para el calculo del LOF, primero llama a `f.dist.knn()` la cual ordena las distancias entre vecinos de menor a mayor, antes de esto llama a la función `distmc()` para que calcule esas distancias previamente (empleando manhattan). Después `lof()` llama a `f.reachability()` que calcula las densidades locales de cada punto. Por último calcula las densidades relativas medias de cada punto. Una vez obtenidos los resultados los imprime por pantalla, aquellos que sean >1 serán posibles outliers, en nuestro caso a pesar de que los puntos 1 y 5 sean >1, el punto 4 es mucho mayor, siendo así el outlier.

Ejecución

```
> library(Rlof)
> datos <- matrix(c(9, 9, 9, 7, 11, 11, 2, 1, 11, 9), ncol=2, byrow=TRUE)
> outliersLof <- lof(datos, k=3, method="manhattan")
> outliersLof

[1] 1.0952381 0.9166667 0.9166667 2.9464286 1.0952381
```

3. Ejercicios de forma autónoma

Realización de cuatro ejercicios de forma autónoma por cada grupo de estudiantes en los que se van a realizar, utilizando el entorno R, un análisis de descripción de datos, un análisis de asociación y dos análisis de detección de datos anómalos, aplicando todos los conceptos teóricos vistos en cada lección:

3.1. Análisis de descripción de datos

El primer conjunto de datos, que se empleará para realizar el análisis de descripción de datos, estará formado por datos de una característica cuantitativa, distancia, desde el domicilio de cada estudiantes hasta la Universidad, dichos datos, cuantitativos continuos, son: 16.5, 34.8, 20.7, 6.2, 4.4, 3.4, 24, 24, 32, 30, 33, 27, 15, 9.4, 2.1, 34, 24, 12, 4.4, 28, 31.4, 21.6, 3.1, 4.5, 5.1, 4, 3.2, 25, 4.5, 20, 34, 12, 12, 12, 12, 5, 19, 30, 5.5, 38, 25, 3.7, 9, 30, 13, 30, 30, 26, 30, 30, 1, 26, 22, 10, 9.7, 11, 24.1, 33, 17.2, 27, 24, 27, 21, 28, 30, 4, 46, 29, 3.7, 2.7, 8.1, 19, 16.

Solución:

- `s<-read.table("distancias.txt")`: Se asigna los valores de la tabla distancias (almacenada en el directorio de trabajo) a la variable "s". Al teclear introducir el nombre de la variable como comando se deberían mostrar los datos
- Cálculo del rango con la función: `ranger(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de números.
  - Retorno: Devuelve el rango del vector introducido, es decir, la diferencia entre el número mayor y menor de dicho vector.
  - Explicación: Esta función se encarga de calcular el rango del vector introducido por parámetro. Va recorriendo el vector y durante cada iteración va comparando el valor actual con el máximo y el mínimo actuales, si dicho valor es menor o mayor que el mínimo o máximo respectivamente, actualiza el valor correspondiente. Una vez finalizado el recorrido calcula la diferencia entre el máximo y el mínimo final.

```
> ranger <- function(vector) {
+   max_value <- vector[1]
+   min_value <- vector[1]
+   # Encontrar el máximo y mínimo recorriendo el vector
+   for (value in 2:length(vector)) {
+     if (vector[value] > max_value) {
+       max_value <- vector[value]
+     }
+     if (vector[value] < min_value) {
+       min_value <- vector[value]
+     }
+   }
+   # Calcular el rango
+   range <- max_value - min_value
+   return(range)
+ }
> datos <- read.table("distancias.txt")
> rango <- ranger(datos$Km)
> rango
[1] 45
```

- Cálculo de la frecuencia absoluta (f_i) con la función: `frecAbsr(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de datos.
  - Retorno: Devuelve un dataframe con el número de apariciones totales de cada dato del vector.
  - Explicación: Esta función se encarga de contar el número de apariciones de cada elemento del vector. Recorre el vector incrementando en 1 la frecuencia de un elemento cada vez que se repita dicho elemento.

```
> frecAbsr <- function(vector) {
+   vector <- sort(vector)
+   v_frecs <- list()
+   # Conteo de las frecuencias de cada valor
+   for (value in vector) {
+     if (value %in% names(v_frecs)) {
+       v_frecs[[as.character(value)]] <-
+           v_frecs[[as.character(value)]] + 1
+     } else {
+       v_frecs[[as.character(value)]] <- 1
+     }
+   }
+   # Marco de datos con las frecuencias absolutas
+   res <- data.frame(Valor = as.numeric(names(v_frecs)),
+   FrecAbs = unname(unlist(v_frecs)))
+   return(res)
+ }
> datos <- read.table("distancias.txt")
> frecAbs <- frecAbsr(datos$Km)
> frecAbs
```

	Valor	FrecAbs
1	1.0	1
2	2.1	1
3	2.7	1
4	3.1	1
5	3.2	1
6	3.4	1
7	3.7	2
8	4.0	2
9	4.4	2
10	4.5	2
11	5.0	1
12	5.1	1
13	5.5	1
14	6.2	1
15	8.1	1
16	9.0	1
17	9.4	1
18	9.7	1
19	10.0	1
20	11.0	1
21	12.0	5
22	13.0	1
23	15.0	1
24	16.0	1
25	16.5	1
26	17.2	1
27	19.0	2

```
28  20.0       1
29  20.7       1
30  21.0       1
31  21.6       1
32  22.0       1
33  24.0       4
34  24.1       1
35  25.0       2
36  26.0       2
37  27.0       3
38  28.0       2
39  29.0       1
40  30.0       8
41  31.4       1
42  32.0       1
43  33.0       2
44  34.0       2
45  34.8       1
46  38.0       1
47  46.0       1
```

- Cálculo de la frecuencia absoluta acumulada ($\texttt{fa}_i$) con la función: `frecAbsAcumr(distancias$Km)`

  - Parámetros:
    - `vector`: Vector de datos.
  - Retorno: Devuelve un dataframe con el número de apariciones totales acumuladas de cada dato del vector.
  - Explicación: Esta función se encarga de calcular las frecuencias acumuladas de cada dato del vector. Calcula primero las frecuencias absolutas con `frecAbsr(vector)` y recorre el dataframe resultado, sumando a la frecuencia absoluta del dato actual, la frecuencia absoluta acumulada del dato anterior.

```
> frecAbsr <- function(vector) {
+   vector <- sort(vector)
+   v_frecs <- list()
+   # Conteo de las frecuencias de cada valor
+   for (value in vector) {
+     if (value %in% names(v_frecs)) {
+       v_frecs[[as.character(value)]] <-
+                   v_frecs[[as.character(value)]] + 1
+     } else {
+       v_frecs[[as.character(value)]] <- 1
+     }
+   }
+   # Marco de datos con las frecuencias absolutas
+   res <- data.frame(Valor = as.numeric(names(v_frecs)),
+   FrecAbs = unname(unlist(v_frecs)))
+   return(res)
```

```
+ }
> frecAbsAcumr <- function(vector) {
+   # Frecuencias absolutas del vector
+   frec_Abs <- frecAbsr(vector)
+   # Inicializar la frecuencia acumulada
+   frec_Abs$FrecAbsAcum <- rep(0, nrow(frec_Abs))
+   # Calculo de las frecuencias absolutas acumuladas
+   for (i in seq_along(frec_Abs$FrecAbsAcum)) {
+     frec_Abs$FrecAbsAcum[i] <- sum(frec_Abs$FrecAbs[1:i])
+   }
+   # Marco de datos con las frecuencias absolutas acumuladas
+   res <- data.frame(Valor = frec_Abs$Valor, FrecAbsAcum =
+    frec_Abs$FrecAbsAcum)
+   return(res)
+ }
> datos <- read.table("distancias.txt")
> frecAbsAcum <- frecAbsAcumr(datos$Km)
> frecAbsAcum
```

	Valor	FrecAbsAcum
1	1.0	1
2	2.1	2
3	2.7	3
4	3.1	4
5	3.2	5
6	3.4	6
7	3.7	8
8	4.0	10
9	4.4	12
10	4.5	14
11	5.0	15
12	5.1	16
13	5.5	17
14	6.2	18
15	8.1	19
16	9.0	20
17	9.4	21
18	9.7	22
19	10.0	23
20	11.0	24
21	12.0	29
22	13.0	30
23	15.0	31
24	16.0	32
25	16.5	33
26	17.2	34
27	19.0	36
28	20.0	37
29	20.7	38

```
30  21.0          39
31  21.6          40
32  22.0          41
33  24.0          45
34  24.1          46
35  25.0          48
36  26.0          50
37  27.0          53
38  28.0          55
39  29.0          56
40  30.0          64
41  31.4          65
42  32.0          66
43  33.0          68
44  34.0          70
45  34.8          71
46  38.0          72
47  46.0          73
```

- Cálculo de la frecuencia relativa ($\texttt{fr}_i$) con la función: `frecRelr(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de datos.
  - Retorno: Devuelve un dataframe con el número de apariciones totales de cada dato en relación al numero de elementos del vector.
  - Explicación: Esta función se encarga de calcular las frecuencias relativas de cada dato del vector. Calcula previamente las frecuencias absolutas y después recorre el dataframe resultado, dividiendo la frecuencia absoluta de cada dato entre el número total de elementos, de la forma $\texttt{f}_i$`/length(vector)`

```
> frecAbsr <- function(vector) {
+   vector <- sort(vector)
+   v_frecs <- list()
+   # Conteo de las frecuencias de cada valor
+   for (value in vector) {
+     if (value %in% names(v_frecs)) {
+       v_frecs[[as.character(value)]] <-
+           v_frecs[[as.character(value)]] + 1
+     } else {
+       v_frecs[[as.character(value)]] <- 1
+     }
+   }
+   # Marco de datos con las frecuencias absolutas
+   res <- data.frame(Valor = as.numeric(names(v_frecs)),
+   FrecAbs = unname(unlist(v_frecs)))
+   return(res)
+ }
> frecRelr <- function(vector) {
+   # Frecuencias absolutas del vector
```

```
+   frec_Abs <- frecAbsr(vector)
+   # Calculo de frecuencias relativas
+   frec_Rel <- frec_Abs$FrecAbs / length(vector)
+   frec_Abs$FrecRel <- frec_Rel
+   # Marco de datos con las frecuencias relativas
+   res <- data.frame(Valor = frec_Abs$Valor, FrecRel =
+   frec_Abs$FrecRel)
+   return(res)
+ }
> datos <- read.table("distancias.txt")
> frecRel <- frecRelr(datos$Km)
> frecRel
```

	Valor	FrecRel
1	1.0	0.01369863
2	2.1	0.01369863
3	2.7	0.01369863
4	3.1	0.01369863
5	3.2	0.01369863
6	3.4	0.01369863
7	3.7	0.02739726
8	4.0	0.02739726
9	4.4	0.02739726
10	4.5	0.02739726
11	5.0	0.01369863
12	5.1	0.01369863
13	5.5	0.01369863
14	6.2	0.01369863
15	8.1	0.01369863
16	9.0	0.01369863
17	9.4	0.01369863
18	9.7	0.01369863
19	10.0	0.01369863
20	11.0	0.01369863
21	12.0	0.06849315
22	13.0	0.01369863
23	15.0	0.01369863
24	16.0	0.01369863
25	16.5	0.01369863
26	17.2	0.01369863
27	19.0	0.02739726
28	20.0	0.01369863
29	20.7	0.01369863
30	21.0	0.01369863
31	21.6	0.01369863
32	22.0	0.01369863
33	24.0	0.05479452
34	24.1	0.01369863
35	25.0	0.02739726

```
36  26.0 0.02739726
37  27.0 0.04109589
38  28.0 0.02739726
39  29.0 0.01369863
40  30.0 0.10958904
41  31.4 0.01369863
42  32.0 0.01369863
43  33.0 0.02739726
44  34.0 0.02739726
45  34.8 0.01369863
46  38.0 0.01369863
47  46.0 0.01369863
```

- Cálculo de la frecuencia relativa acumulada ($\texttt{fra}_i$) con la función: `frecRelAcumr(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de datos.
  - Retorno: Devuelve un dataframe con las frecuencias relativas acumuladas de cada dato del vector.
  - Explicación: Esta función se encarga de calcular las frecuencias relativas acumuladas de cada dato del vector. Calcula primero las frecuencias relativas con `frecRelr(vector)` y recorre el dataframe resultado, sumando a la frecuencia relativa del dato actual, la frecuencia relativa acumulada del dato anterior.

```
> frecAbsr <- function(vector) {
+   vector <- sort(vector)
+   v_frecs <- list()
+   # Conteo de las frecuencias de cada valor
+   for (value in vector) {
+     if (value %in% names(v_frecs)) {
+       v_frecs[[as.character(value)]] <-
+           v_frecs[[as.character(value)]] + 1
+     } else {
+       v_frecs[[as.character(value)]] <- 1
+     }
+   }
+   # Marco de datos con las frecuencias absolutas
+   res <- data.frame(Valor = as.numeric(names(v_frecs)),
+   FrecAbs = unname(unlist(v_frecs)))
+   return(res)
+ }
> frecRelr <- function(vector) {
+   # Frecuencias absolutas del vector
+   frec_Abs <- frecAbsr(vector)
+   # Calculo de frecuencias relativas
+   frec_Rel <- frec_Abs$FrecAbs / length(vector)
+   frec_Abs$FrecRel <- frec_Rel
+   # Marco de datos con las frecuencias relativas
```

```
+   res <- data.frame(Valor = frec_Abs$Valor, FrecRel =
+   frec_Abs$FrecRel)
+   return(res)
+ }
> frecRelAcumr <- function(vector) {
+   # Frecuencias relativas del vector
+   frec_Rel <- frecRelr(vector)
+   # Inicializar la frecuencia acumulada
+   frec_Rel$FrecRelAcum <- rep(0, nrow(frec_Rel))
+   # Calculo de las frecuencias relativas acumuladas
+   for (i in seq_along(frec_Rel$FrecRelAcum)) {
+     frec_Rel$FrecRelAcum[i] <- sum(frec_Rel$FrecRel[1:i])
+   }
+   # Marco de datos con las frecuencias relativas acumuladas
+   res <- data.frame(Valor = frec_Rel$Valor, frecRelAcum =
+   frec_Rel$FrecRelAcum)
+   return(res)
+ }
> datos <- read.table("distancias.txt")
> frecRelAcum <- frecRelAcumr(datos$Km)
> frecRelAcum
   Valor frecRelAcum
1    1.0  0.01369863
2    2.1  0.02739726
3    2.7  0.04109589
4    3.1  0.05479452
5    3.2  0.06849315
6    3.4  0.08219178
7    3.7  0.10958904
8    4.0  0.13698630
9    4.4  0.16438356
10   4.5  0.19178082
11   5.0  0.20547945
12   5.1  0.21917808
13   5.5  0.23287671
14   6.2  0.24657534
15   8.1  0.26027397
16   9.0  0.27397260
17   9.4  0.28767123
18   9.7  0.30136986
19  10.0  0.31506849
20  11.0  0.32876712
21  12.0  0.39726027
22  13.0  0.41095890
23  15.0  0.42465753
24  16.0  0.43835616
25  16.5  0.45205479
26  17.2  0.46575342
```

```
27  19.0  0.49315068
28  20.0  0.50684932
29  20.7  0.52054795
30  21.0  0.53424658
31  21.6  0.54794521
32  22.0  0.56164384
33  24.0  0.61643836
34  24.1  0.63013699
35  25.0  0.65753425
36  26.0  0.68493151
37  27.0  0.72602740
38  28.0  0.75342466
39  29.0  0.76712329
40  30.0  0.87671233
41  31.4  0.89041096
42  32.0  0.90410959
43  33.0  0.93150685
44  34.0  0.95890411
45  34.8  0.97260274
46  38.0  0.98630137
47  46.0  1.00000000
```

- Cálculo de la media aritmética ($\bar{x}_a$) con la función: `meanr(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de números.
  - Retorno: Devuelve un numero que representa la media aritmética
  - Explicación: Esta función se encarga de calcular la media aritmética, recorriendo el vector sumando todos los valores y después dividiéndolos entre el número total de elementos del vector, de la forma:
    $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

```
> meanr <- function(vector) {
+   sum <- 0
+   # Suma de los elementos del vector
+   for (value in vector) {
+     sum <- sum + value
+   }
+   # Calculo de la media
+   mean <- sum / length(vector)
+   return(mean)
+ }
> datos <- read.table("distancias.txt")
> mean <- meanr(datos$Km)
> mean
[1] 18.53425
```

- Cálculo de la desviación típica (`s`) con la función: `sdr(distancias$Km)`

- Parámetros:
  - `vector`: Vector de números.
- Retorno: Devuelve un número que representa la desviación típica.
- Explicación: Esta función se encarga de calcular la desviación típica. Primero calcula la media con la función `meanr(vector)`, después va recorriendo el vector calculando la suma de cuadrados entre la diferencia de cada valor y la media, lo divide todo entre el número total de elementos del vector y por último realiza la raíz cuadrada de dicho resultado, de la forma:
  $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$

```
> meanr <- function(vector) {
+   sum <- 0
+   # Suma de los elementos del vector
+   for (value in vector) {
+     sum <- sum + value
+   }
+   # Calculo de la media
+   mean <- sum / length(vector)
+   return(mean)
+ }
> sdr <- function(vector) {
+     # Calculo de la media de los elementos del vector
+     mean <- meanr(vector)
+     # Suma de cuadrados
+     for (value in vector) {
+         sum <- (value-mean)^2
+     }
+     # Calculo desviacion estandar
+     sd <- sqrt(sum/length(vector))
+     return(sd)
+ }
> datos <- read.table("distancias.txt")
> desv <- sdr(datos$Km)
> desv
[1] 0.2966111
```

- Cálculo de la varianza (s^2) con la función: `varr(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de números.
  - Retorno: Devuelve un número que representa la varianza.
  - Explicación: Esta función se encarga de calcular la varianza. Primero calcula la desviación típica con la función `sdr(vector)` y después eleva al cuadrado dicho resultado, de la forma:
    $s^2 = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$

```
> meanr <- function(vector) {
+   sum <- 0
```

```
+   # Suma de los elementos del vector
+   for (value in vector) {
+     sum <- sum + value
+   }
+   # Calculo de la media
+   mean <- sum / length(vector)
+   return(mean)
+ }
> sdr <- function(vector) {
+     # Calculo de la media de los elementos del vector
+     mean <- meanr(vector)
+     # Suma de cuadrados
+     for (value in vector) {
+         sum <- (value-mean)^2
+     }
+     # Calculo desviacion estandar
+     sd <- sqrt(sum/length(vector))
+     return(sd)
+ }
> varr <- function(values) {
+     # Calculo desviacion estandar
+     sd <- sdr(values)
+     # Calculo varianza
+     var <- sd^2
+     return(var)
+ }
> datos <- read.table("distancias.txt")
> var <- varr(datos$Km)
> var
[1] 0.08797816
```

- Cálculo de la mediana ($\tilde{x}$) con la función: `medianr(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de números.
  - Retorno: Devuelve un número que representa la mediana.
  - Explicación: Esta función se encarga de calcular la mediana. Primero ordena de forma creciente el vector con la función `sort(vector)` y calcula la longitud del mismo, dependiendo de si la longitud es par o impar, calcula la mediana de la forma:

$$\tilde{x} = \begin{cases} x_{\frac{n+1}{2}} & \text{si } n \text{ es impar} \\ \frac{1}{2}(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}) & \text{si} n \text{ es par} \end{cases}$$

```
> medianr <- function(vector) {
+   n <- length(vector)
+   v <- sort(vector)
+ # Comprobar si el numero de elementos del vector es impar o par
+   if(n%%2==1) {
```

```
+         # Si es impar la mediana es el elemento del medio
+     median <- v[(n+1)/2]
+   }else {
+ # Si es par la mediana es la media de los 2 elementos del medio
+     median <- (v[n/2] + v[(n/2)+1]) / 2
+   }
+   return(median)
+ }
> datos <- read.table("distancias.txt")
> median <- medianr(datos$Km)
> median
[1] 20
```

- Cálculo de laos cuartiles ($\texttt{Q}_i$) con la función: `quantilerr(distancias$Km,p)`
  - Parámetros:
    - `vector`: Vector de números.
    - `p`: Número decimal del cuantil.
  - Retorno: Devuelve un número que representa el cuantil de `p`.
  - Explicación: Esta función se encarga de calcular el cuantil. Primero ordena de forma creciente el vector con la función `sort(vector)`, después calcula la posición del cuantil, de la forma:

$$Q_p = \begin{cases} x_{\frac{np}{2}} + x_{\frac{np+1}{2}} & \text{si } np \in \mathbb{N} \\ x_{\lfloor np \rfloor + 1} & \text{si } np \notin \mathbb{N} \end{cases}$$

```
> quantiler <- function(values, p) {
+   n <- length(values)
+   if (p < 0 || p > 1) {
+     stop("El cuantil debe estar en el rango de 0 a 1.")
+   }
+   v <- sort(values)
+   # Calcular posición del cuantil
+   np <- n * p
+   # Calcular cuantil interpolando
+   if (np %% 10 == 0) {
+     quantile <- (v[np]+v[np+1])/2
+   } else {
+     int <- floor(np)
+     quantile <- v[int+1]
+   }
+   return(quantile)
+ }
> datos <- read.table("distancias.txt")
> quantile1 <- quantiler(datos$Km,0.25)
> quantile1
[1] 8.1
> quantile3 <- quantiler(datos$Km,0.75)
> quantile3
```

```
[1] 28
```

3.2. Análisis de asociación

El segundo conjunto de datos, que se empleará para realizar el análisis de asociación, estará formado por las siguientes conjuntos de extras incluidos en 8 ventas de coches: {X, C, N, B}, {X, T, B, C}, {N, C, X}, {N, T, X, B}, {X, C, B}, {N}, {X, B, C}, {T, A}. Donde: X: Faros de Xenon, A: Alarma, T: Techo Solar, N: Navegador, B: Bluetooth, C: Control de Velocidad, son los extras que se pueden incluir en cada coche

Solución: Algoritmo de minería de reglas de asociación (apriori): Su objetivo principal es descubrir patrones de asociación entre diferentes conjuntos de datos.

- Parámetros:
  - `transacciones`: Lista de sucesos que conforman la muestra.
  - `soporte`: Umbral mínimo de soporte que deben superar las asociaciaciones.
  - `confianza`: Umbral mínimo de confianza que deben superar las asociaciaciones.
- Retorno: Imprime por pantalla todas aquellas asociaciaciones que superen los umbrales de soporte y confianza
- Explicación: Este algoritmo comienza identificando los sucesos elementales que cumplen con el umbral de soporte, luego, utiliza estos elementos frecuentes para generar conjuntos de dos elementos que también cumplan con el umbral de soporte, este proceso continúa iterativamente hasta que ya no se pueden generar conjuntos más grandes que cumplan con el umbral. Una vez que se han identificado los conjuntos de elementos frecuentes, se generan reglas de asociación a partir de ellos, cada regla de asociación tiene una parte izquierda (antecedente) y una parte derecha (consecuente). Estas reglas se crean combinando los elementos frecuentes de manera que el apoyo de la regla sea mayor o igual al umbral especificado, las reglas de asociación generadas se evalúan según la confianza las reglas que cumplen con el umbral se seleccionan como resultados finales.

```
> # Define los sucesos
> transactions <- list(
+         c("X", "C", "N", "B"),
+         c("X", "T", "B", "C"),
+         c("N", "C", "X"),
+         c("N", "T", "X", "B"),
+         c("X", "C", "B"),
+         c("N"),
+         c("X", "B", "C"),
+         c("T", "A")
+ )
> # Define soporte y confianza
> soporte <- 0.4
> confianza <- 0.9
> calculate_support <- function(itemset, transactions) {
```

```
+         # Función para calcular el soporte de un conjunto de
+         # elementos en las transacciones
+         count <- sum(sapply(transactions, function(transaction)
+         all(itemset %in% transaction)))
+         return(count / length(transactions))
+ }
> filtar_sucesos_elementales <- function(transactions, soporte) {
+         # Obtener todos los elementos únicos en los sucesos
+         sucesos_elementales <- unique(unlist(transactions))
+
+         # Inicializar lista para almacenar sucesos después de
+         #aplicar el umbral de soporte
+         sucesos_filtrados <- list()
+
+         # Calcular el soporte para cada elemento y filtrar
+         for (suceso in sucesos_elementales) {
+         soporte_suceso <- calculate_support(c(suceso), transactions)
+         if (soporte_suceso >= soporte) {
+                 sucesos_filtrados <- c(sucesos_filtrados, list(c(suceso)))
+         }
+         }
+
+         return(sucesos_filtrados)
+ }
> apriori_gen <- function(conjuntos_anteriores, k) {
+         #Identifica los candidatos para cada dimensión
+         # Inicialización de la lista que contendrá los sucesos candidatos.
+         sucesos_candidatos <- list()
+
+   # Verificación de que haya al menos dos conjuntos anteriores.
+   if (length(conjuntos_anteriores) < 2) {
+     return(sucesos_candidatos)
+   }
+
+   # Iteración sobre cada par único de conjuntos anteriores.
+   for (i in 1:(length(conjuntos_anteriores) - 1)) {
+     for (j in (i + 1):length(conjuntos_anteriores)) {
+           conjunto_a <- conjuntos_anteriores[[i]]
+           conjunto_b <- conjuntos_anteriores[[j]]
+
+           # Generación de candidatos para k = 2.
+           if (k == 2) {
+             if (conjunto_a[1] != conjunto_b[1]) {
+                   nuevo_conjunto <- c(conjunto_a, conjunto_b[1])
+                   sucesos_candidatos <-
+                   append(sucesos_candidatos, list(nuevo_conjunto))
+             }
+           } else if (k > 2) {
```

```
+             # Generación de candidatos para k > 2.
+             if (identical(conjunto_a[1:(k-2)], conjunto_b[1:(k-2)]) &&
+                 conjunto_a[(k-1)] != conjunto_b[(k-1)]) {
+                   nuevo_conjunto <- c(conjunto_a, conjunto_b[(k-1)])
+                   sucesos_candidatos <-
+                   append(sucesos_candidatos, list(nuevo_conjunto))
+             }
+           }
+     }
+   }
+
+   # Llamada recursiva para generar candidatos de dimensión k + 1.
+   sucesos_cand_recur <- apriori_gen(sucesos_candidatos, k + 1)
+   sucesos_candidatos <- append(sucesos_candidatos, sucesos_cand_recur)
+
+   # Devolver la lista de sucesos candidatos generados.
+   return(sucesos_candidatos)
+ }
> filtrar_sucesos <- function(sucesos_candidatos, transactions) {
+   #Se filtran los sucesos candidatos en función del soporte
+   # Inicialización de la lista que contendrá los sucesos filtrados.
+   sucesos_filtrados <- list()
+
+   # Iteración sobre cada suceso candidato.
+   for (suceso in sucesos_candidatos) {
+     # Se hace uso de la función calculate_support.
+     soporte_suceso <- calculate_support(suceso, transactions)
+
+     # Verificación de que el soporte del suceso cumple con el umbral.
+     if (soporte_suceso >= soporte) {
+       # Agregar el suceso a la lista de sucesos filtrados.
+       sucesos_filtrados <- c(sucesos_filtrados, list(suceso))
+     }
+   }
+
+   # Devolver la lista de sucesos filtrados.
+   return(sucesos_filtrados)
+ }
> calcular_confianza <- function(regla, transactions) {
+   #Calcula la confianza de una regla de asociación
+   # Obtener el antecedente y el consecuente de la regla
+   ant <- regla$antecedente
+   cons <- regla$consecuente
+
+   # Paso 2: Calcular el soporte de la regla y el soporte del antecedente
+   soporte_regla <- calculate_support(c(ant, cons), transactions)
+   soporte_antecedente <- calculate_support(ant, transactions)
+
```

```
+   # Paso 3: Calcular la confianza de la regla
+   confianza_regla <- soporte_regla / soporte_antecedente
+
+   # Paso 4: Devolver la confianza de la regla
+   return(confianza_regla)
+ }
> ap_genrules <- function(sucesos_filtrados, transactions, confianza) {
+   # Genera reglas de asociación a partir de sucesos filtrados basándose
+   # en un umbral de confianza.
+
+   # Inicialización de la lista que contendrá las reglas generadas.
+   reglas <- list()
+
+   # Iteración sobre cada suceso en la lista de sucesos filtrados.
+   for (i in 1:length(sucesos_filtrados)) {
+     suceso <- sucesos_filtrados[[i]]
+     k <- length(suceso)
+
+     # Verificación de que el suceso tiene al menos dos elementos.
+     if (k >= 2) {
+       # Generación de todos los conjuntos anteriores de tamaño k-1.
+       conjuntos_anteriores <- combn(suceso, k - 1, simplify = FALSE)
+
+       # Iteración sobre cada conjunto anterior.
+       for (conjunto_anterior in conjuntos_anteriores) {
+         antecedente <- conjunto_anterior
+         consecuente <- setdiff(suceso, antecedente)
+
+         # Cálculo de la confianza.
+         confianza_regla <- calcular_confianza(list(antecedente =
+                 antecedente, consecuente = consecuente), transactions)
+
+         # Verificación si la confianza cumple con el umbral.
+         if (confianza_regla >= (confianza - 0.001)) {
+           # Agregar la regla a la lista de reglas generadas.
+           reglas <-
+                   append(reglas, list(list(antecedente = antecedente,
+                                            consecuente = consecuente,
+                                            soporte =
+                                       calculate_support(suceso, transactions),
+                                            confianza = confianza_regla)))
+         }
+       }
+     }
+   }
+
+   # Devolver la lista de reglas generadas.
+   return(reglas)
```

```
+ }
> apriori_ej <- function(transactions, soporte, confianza) {
+         # Paso A:
+         # Paso A.1: Filtrar sucesos elementales que llegan al soporte
+         sucesos_elem_candidatos <-
+         filtar_sucesos_elementales(transactions, soporte)
+         # Paso A.2:
+         # Paso A.2.1: Identificar sucesos candidatos en cada dimensión
+         sucesos_candidatos <- apriori_gen(sucesos_elem_candidatos, 2)
+         # Paso A.2.2: Calcular soporte de los sucesos candidatos y filtrar
+         sucesos_filtrados <- filtrar_sucesos(sucesos_candidatos,
+         transactions)
+
+         # Paso B: Aplicar la función ap-genrules para establecer
+         #                   asociaciones con el umbral de confianza
+         rules <- ap_genrules(sucesos_filtrados, transactions, confianza)
+
+         return(rules)
+ }
> # Llamar a la función Apriori
> resultados <- apriori_ej(transactions, soporte, confianza)
> # Imprimir las reglas de asociación
> for (i in 1:length(resultados)) {
+         antecedente <- unlist(resultados[[i]]$antecedente)
+         consecuente <- unlist(resultados[[i]]$consecuente)
+         soporte <- resultados[[i]]$soporte
+         confianza <- resultados[[i]]$confianza
+
+         cat(
+         sprintf("[%d] {%s} => {%s}  support: %.7f  confidence: %.7f\n",
+                         i, paste(antecedente, collapse = ", "),
+                         paste(consecuente, collapse = ", "),
+                         soporte, confianza)
+         )
+ }

[1] {C} => {X}  support: 0.6250000  confidence: 1.0000000
[2] {B} => {X}  support: 0.6250000  confidence: 1.0000000
[3] {C, B} => {X}  support: 0.5000000  confidence: 1.0000000
```

3.3. Análisis de detección de datos anómalos - Técnicas con base estadística

Solución:

- Medidas de ordenación (Velocidad): Para este análisis de detección de outliers, se va a hacer uso de Caja y Bigotes. Esta técnica consiste en ordenar los elementos, calcular el primer y tercer cuartil y establecer unos límites, si algún valor no se

encuentra dentro de estos límites, será considerado outlier.
Todo esto está implementado en la función: `cajaBigotes(matriz,valor,d)`.

- Parámetros:
  - `matriz`: Una matriz de datos númericos.
  - `valor`: El nombre de la medidad (fila de la matriz) sobre la que queremos realizar el análisis.
  - `d`: Grado de outlier.
- Retorno: Imprime por pantalla todos aquellos valores que queden fuera de los límites establecidos, es decir, los outliers.
- Explicación: Esta función calcula los outliers sobre una serie de valores. Recibe el grado de outlier (`d`) y una matriz (`matriz`) con dichos valores y lo primero que hace es crear un dataframe de la traspuesta de dicha matriz, para facilitar su tratamiento. Después a partir de `valor`, obtiene la fila de la matriz que contiene los valores que vamos a evaluar. Una vez hecho esto hace uso de la función `quantiler()`, la cual hemos implementado previamente, para calcular el primer y tercer cuartil, Q_1 y Q_3 respectivamente . Por último calcula los límites a partir de, $(Q_1 - d \cdot (Q_3 - Q_1), Q_3 + d \cdot (Q_3 - Q_1))$ e imprimirá por pantalla todos aquellos valores que se encuentren fuera de estos límites establecidos, ya que, serán considerados outliers.

```
> quantiler <- function(values, p) {
+   n <- length(values)
+   v <- sort(values)
+   # Calcular posición del cuantil
+   np <- n * p
+   # Calcular cuantil interpolando
+   if (np %% 10 == 0) {
+     quantile <- (v[np]+v[np+1])/2
+   } else {
+     int <- floor(np)
+     quantile <- v[int+1]
+   }
+   return(quantile)
+ }
> cajaBigotes <- function(matriz,col,d){
+     # Dataframe
+     matriz <- data.frame(matriz)
+     # Obtiene indice de la col a evaluar
+     col <- which(colnames(matriz) == col)
+     # Calculo de 1er y 3er cuartil de los valores de dicha col
+     cuart1 <- quantiler(matriz[,col],0.25)
+     cuart3 <- quantiler(matriz[,col],0.75)
+     # Calculo de los limites
+     limites <- c(cuart1-d*(cuart3-cuart1),cuart3+d*(cuart3-cuart1))
+     # Calculo de los outliers
+     for(i in 1:length(matriz[, col])) {
+         # Si el punto se encuentra fuera de los limites es outlier
```

```
+         if(matriz[i,col]<limites[1] || matriz[i,col]>limites[2]) {
+             print(paste("El suceso", i, ":", matriz[i, col],
+                         "es un dato anómalo"))
+         }
+     }
+ }
> datos <- t(matrix(c(10, 7.46, 8, 6.77, 13, 12.74, 9, 7.11, 11,
+ 7.81, 14, 8.84, 6, 6.08, 4, 5.39, 12, 8.15, 7, 6.42, 5, 5.73),
+ 2,11, dimnames=list(c("r","d"))))
> outliers <- cajaBigotes(datos,"r",0.25)
[1] "El suceso 6 : 14 es un dato anómalo"
[1] "El suceso 8 : 4 es un dato anómalo"
```

- Medidas de dispersión (Temperatura): Para este análisis de detección de outliers, se va a hacer uso de la Desviación Típica. Esta técnica consiste en calcular la media y la desviación típica de los valores a evaluar y establecer unos límites, si algún valor no se encuentra dentro de estos límites, será considerado outlier.
  Todo esto está implementado en la función: `desvTip(matriz,valor,d)`.

  - Parámetros:
    - `matriz`: Una matriz de datos números.
    - `valor`: El nombre de la medidad (fila de la matriz) sobre la que queremos realizar el análisis.
    - `d`: Grado de outlier.
  - Retorno: Imprime por pantalla todos aquellos valores que queden fuera de los límites establecidos, es decir, los outliers.
  - Explicación: Esta función calcula los outliers sobre una serie de valores. Recibe el grado de outlier (`d`) y una matriz (`matriz`) con dichos valores y lo primero que hace es crear un dataframe de la traspuesta de dicha matriz, para facilitar su tratamiento. Después a partir de `valor`, obtiene la fila de la matriz que contiene los valores que vamos a evaluar. Una vez hecho esto hace uso de las funciónes `meanr()` y `sdr()`, las cuales hemos implementado previamente, para calcular la media $\bar{x}_a$ y la desviación típica `s` respectivamente. Por último calcula los límites a partir de, $(\bar{x}_a - d \cdot s, \bar{x}_a + d \cdot s)$ e imprimirá por pantalla todos aquellos valores que se encuentren fuera de estos límites establecidos, ya que, serán considerados outliers.

```
> sdr <- function(vector) {
+     # Calculo de la media de los elementos del vector
+     mean <- meanr(vector)
+     # Suma de cuadrados
+     for (value in vector) {
+         sum <- (value-mean)^2
+     }
+     # Calculo desviacion estandar
+     sd <- sqrt(sum/length(vector))
+     return(sd)
+ }
```

```
> meanr <- function(vector) {
+   sum <- 0
+   # Suma de los elementos del vector
+   for (value in vector) {
+     sum <- sum + value
+   }
+   # Calculo de la media
+   mean <- sum / length(vector)
+   return(mean)
+ }
> desvTip <- function(matriz,col,d){
+     # Dataframe
+     matriz <- data.frame(matriz)
+     # Obtiene indice de la col a evaluar
+     col <- which(colnames(matriz) == col)
+     # Calculo de la media de los valores de dicha col
+     media <- meanr(matriz[,col])
+     # Calculo de la desviacion de los valores de dicha col
+     sd <- sdr(matriz[,col])
+     # Calculo de los limites
+     limites <- c(media-d*sd,media+d*sd)
+     # Calculo de los outliers
+     for(i in 1:length(matriz[,col])) {
+     # Si el punto se encuentra fuera de los limites es outlier
+         if(matriz[i,col]<limites[1] ||
+                 matriz[i,col]>limites[2]) {
+             print(paste("El suceso", i, ":", matriz[, col][i],
+                         "es un dato anómalo"))
+         }
+     }
+ }
> datos <- t(matrix(c(10, 7.46, 8, 6.77, 13, 12.74, 9, 7.11, 11,
+ 7.81, 14, 8.84, 6, 6.08, 4, 5.39, 12, 8.15, 7, 6.42, 5, 5.73),
+ 2,11, dimnames=list(c("r","d"))))
> outliers <- desvTip(datos,"d",9)

[1] "El suceso 3 : 12.74 es un dato anómalo"
```

3.4. Análisis de detección de datos anómalos - Técnicas basadas en la proximidad y en la densidad

El cuarto conjunto de datos, que se empleará para realizar el análisis de detección de datos anómalos utilizando técnicas basadas en la proximidad y en la densidad, estará formado por el número de Mujeres y Hombres inscritos en una serie de cinco seminarios que se han impartido sobre biología. Los datos son: Mujeres, Hombres: 1. 9, 9; 2. 9, 7; 3. 11, 11; 4. 2, 1; 5. 11, 9.

Solución:

- Técnicas basadas en proximidad (k-Vecinos): Para este análisis de detección de outliers, se va a hacer uso de la técnica k-vecinos. Esta técnica consiste en buscar sucesos muy separados al resto en función de sus distancias. Para ello se calculan las distancias euclídeas entre todos los puntos, para posteriormente ordenar estas distancias de forma creciente, solo se ordenará el número de distancias establecido por `k`. Por último se comprueba si el vecino `k` supera el grado de outlier establecido por `d`, si lo hace, será considerado outlier.

  - Parámetros:
    - `matriz`: Matriz de valores númericos.
    - `k`: Número de orden k.
    - `d`: Grado de outlier.
  - Retorno: Imprime por pantalla todos aquellos sucesos cuyo `k`, se encuentre a una distancia mayor que el grado de outlier `d`, es decir, los outliers.
  - Explicación: Esta función calcula los outliers sobre una serie de sucesos. Recibe el número de orden `k`, el grado de outlier (`d`) y una matriz (`matriz`) con dichos sucesos. El primer paso es calcular las distancias euclídeas de la forma, $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, a partir de la función `distEuc()`. Después se ordenan dichas distancias de menor a mayor. Por último se verifica dentro de las distancias ordenadas, si el vecino `k` de cada punto, supera el valor establecido inicialmente por `d`, si esto sucede se imprime dicho suceso por pantalla, ya que. será considerado outlier.

```
> kvecinos <- function(matriz,k,d){
+     # Traspuesta
+     matrizt <- t(matriz)
+     # Numero de filas
+     n <- nrow(matrizt)
+     # Matriz distancias
+     distancias <- matrix(0, n, n)
+
+     # Calculo distancias euclideas
+     for (i in 1:n) {
+         for (j in 1:n) {
+             if (i != j) {
+                 distancias[i, j] <- round(distEuc(matrizt[i, ],
+                                     matrizt[j, ]),2)
+             }
+         }
+     }
+
+     # Ordenacion de las distancias
+     for(i in 1:n){
+         distancias[,i]=sort(distancias[,i])
+     }
+
+     distanciasordenadas <- distancias
+
```

```
+     # Calculo de los outliers
+     for (i in 1:n) {
+         if (distanciasordenadas[k+1,i]>d) {
+             print(paste("Para k =",k," el suceso ",i," es anómalo"))
+         }
+     }
+ }
> distEuc <- function(x1, x2) {
+     # Calcular la distancia euclidiana
+     distancia <- sqrt(sum((x1 - x2)^2))
+
+     return(distancia)
+ }
> datos <- matrix(c(9, 9, 9, 7, 11, 11, 2, 1, 11, 9), ncol=5,
+ byrow=TRUE)
> outliers <- kvecinos(datos,3,9.5)

[1] "Para k = 3  el suceso  3  es anómalo"
```

- Técnicas basadas en densidad (Local Outlier Factor):

  - Parámetros:
    - `datos`: Matriz de valores númericos.
    - `k`: Número de orden k.
    - `dist`: Método de cálculo de las distancias.
  - Retorno: Imprime por pantalla los valores lof de cada punto.
  - Explicación: Utilizamos el paquete Rlof, el cual contiene una función llamada *lof(datos,k,dist)*, el cual recibe los puntos que va a evaluar, el número de vecinos cercanos (k) y el método que se emplea para calcular las distancias (al ser LOF usamos *manhattan*).
    Esta función hace uso de varias funciones externas para el calculo del LOF, primero llama a `f.dist.knn()` la cual ordena las distancias entre vecinos de menor a mayor, antes de esto llama a la función `distmc()` para que calcule esas distancias previamente (empleando manhattan). Después `lof()` llama a `f.reachability()` que calcula las densidades locales de cada punto. Por último calcula las densidades relativas medias de cada punto. Una vez obtenidos los resultados los imprime por pantalla, aquellos que sean >1 serán posibles outliers, en nuestro caso a pesar de que los puntos 1 y 5 sean >1, el punto 4 es mucho mayor, siendo así el outlier.

```
> library(Rlof)
> datos <- matrix(c(9, 9, 9, 7, 11, 11, 2, 1, 11, 9), ncol=2,
+ byrow=TRUE)
> outliersLof <- lof(datos, k=3, method="manhattan")
> outliersLof

[1] 1.0952381 0.9166667 0.9166667 2.9464286 1.0952381
```