Fundamentos de la Ciencia de Datos

Prueba de Laboratorio 2 (PL2)

Jorge Revenga Martín de Vidales
Ángel Salgado Aldao
Adrián García

Grado en Ingeniería Informática
Universidad de Alcalá

21 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.

2. Ejercicios con ayuda del profesor

Realización de cuatro ejercicios con ayuda del profesor en los que se van a realizar, utilizando el entorno RStudio, un análisis de clasificación no supervisada con k-Means, un análisis de clasificación no supervisada con Clusterización Jerárquica Aglomerativa, un análisis de clasificación supervisada utilizando árboles de decisión y un análisis de clasificación supervisada utilizando regresión, 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 clasificación no supervisada con k-Means, estará formado por las siguientes 8 calificaciones de estudiantes: 1.{4, 4}; 2.{3, 5}; 3.{1, 2}; 4.{5, 5}; 5.{0, 1}; 6.{2, 2}; 7.{4, 5}; 8.{2, 1}, donde las características de las calificaciones son: {Teoría, Laboratorio}.

Solución:

- `m<-matrix(c(4,4, 3,5, 1,2, 5,5, 0,1, 2,2, 4,5, 2,1),2,8)`: Explicacion

```
> m<-matrix(c(4,4, 3,5, 1,2, 5,5, 0,1, 2,2, 4,5, 2,1),2,8)
> (m<-t(m))

     [,1] [,2]
[1,]    4    4
[2,]    3    5
[3,]    1    2
[4,]    5    5
[5,]    0    1
[6,]    2    2
[7,]    4    5
[8,]    2    1
```

- `c<-matrix(c(0,1,2,2),2,2)`: Explicacion

```
> c<-matrix(c(0,1,2,2),2,2)
> (c<-t(c))

     [,1] [,2]
[1,]    0    1
[2,]    2    2
```

- `(clasificacionns=(kmeans(m,c,4)))`: Explicacion

```
> (clasificacionns=(kmeans(m,c,4)))

K-means clustering with 2 clusters of sizes 4, 4

Cluster means:
  [,1] [,2]
```

```
1 1.25 1.50
2 4.00 4.75

Clustering vector:
[1] 2 2 1 2 1 1 2 1

Within cluster sum of squares by cluster:
[1] 3.75 2.75
 (between_SS / total_SS =  84.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

- : Explicacion

```
> (m=cbind(clasificacionns$cluster,m))
```

```
     [,1] [,2] [,3]
[1,]    2    4    4
[2,]    2    3    5
[3,]    1    1    2
[4,]    2    5    5
[5,]    1    0    1
[6,]    1    2    2
[7,]    2    4    5
[8,]    1    2    1
```

- : Explicacion

```
> mc1=subset(m,m[,1]==1)
> mc2=subset(m,m[,1]==2)
> mc1
```

```
     [,1] [,2] [,3]
[1,]    1    1    2
[2,]    1    0    1
[3,]    1    2    2
[4,]    1    2    1
```

```
> mc2
```

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

- : Explicacion

```
> (mc1=mc1[,-1])

     [,1] [,2]
[1,]    1    2
[2,]    0    1
[3,]    2    2
[4,]    2    1

> (mc2=mc2[,-1])

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

- : Explicacion

```
> install.packages("LearnClust")

package 'LearnClust' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Jorge\AppData\Local\Temp\Rtmpk1Msop\downloaded_packages

> library(LearnClust)
> search()

 [1] ".GlobalEnv"         "package:LearnClust" "package:magick"
 [4] "package:arules"     "package:Matrix"     "package:stats"
 [7] "package:graphics"   "package:grDevices"  "package:utils"
[10] "package:datasets"   "package:methods"    "Autoloads"
[13] "package:base"
```

- : Explicacion

```
> m<-matrix(c(0.89,2.94, 4.36,5.21, 3.75,1.12, 6.25,3.14, 4.1,1.8, 3.9,4.27),2,6)
> (m<-t(m))

     [,1] [,2]
[1,] 0.89 2.94
[2,] 4.36 5.21
[3,] 3.75 1.12
[4,] 6.25 3.14
[5,] 4.10 1.80
[6,] 3.90 4.27
```

- : Explicacion

```
> agglomerativeHC(m,'EUC','MIN')
```

```
$dendrogram
Number of objects: 6


$clusters
$clusters[[1]]
    X1   X2
1 0.89 2.94

$clusters[[2]]
    X1   X2
1 4.36 5.21

$clusters[[3]]
    X1   X2
1 3.75 1.12

$clusters[[4]]
    X1   X2
1 6.25 3.14

$clusters[[5]]
   X1  X2
1 4.1 1.8

$clusters[[6]]
   X1   X2
1 3.9 4.27

$clusters[[7]]
    X1   X2
1 3.75 1.12
2 4.10 1.80

$clusters[[8]]
    X1   X2
1 4.36 5.21
2 3.90 4.27

$clusters[[9]]
    X1   X2
1 3.75 1.12
2 4.10 1.80
3 4.36 5.21
4 3.90 4.27

$clusters[[10]]
    X1   X2
```

```
1 6.25 3.14
2 3.75 1.12
3 4.10 1.80
4 4.36 5.21
5 3.90 4.27

$clusters[[11]]
    X1   X2
1 0.89 2.94
2 6.25 3.14
3 3.75 1.12
4 4.10 1.80
5 4.36 5.21
6 3.90 4.27


$groupedClusters
  cluster1 cluster2
1        3        5
2        2        6
3        7        8
4        4        9
5        1       10
```

- : Explicacion

```
> agglomerativeHC.details(m,'EUC','MIN')

[[1]]
     [,1] [,2] [,3]
[1,] 0.89 2.94    1

[[2]]
     [,1] [,2] [,3]
[1,] 4.36 5.21    1

[[3]]
     [,1] [,2] [,3]
[1,] 3.75 1.12    1

[[4]]
     [,1] [,2] [,3]
[1,] 6.25 3.14    1

[[5]]
     [,1] [,2] [,3]
[1,]  4.1  1.8    1

[[6]]
```

```
     [,1] [,2] [,3]
[1,]  3.9 4.27    1

         [,1]     [,2]      [,3]     [,4]      [,5]     [,6]
[1,] 0.000000 4.146541 3.3899853 5.363730 3.4064204 3.290745
[2,] 4.146541 0.000000 4.1352388 2.803034 3.4198977 1.046518
[3,] 3.389985 4.135239 0.0000000 3.214094 0.7647876 3.153569
[4,] 5.363730 2.803034 3.2140940 0.000000 2.5333969 2.607566
[5,] 3.406420 3.419898 0.7647876 2.533397 0.0000000 2.478084
[6,] 3.290745 1.046518 3.1535694 2.607566 2.4780839 0.000000
    X1   X2
1 3.75 1.12
2 4.10 1.80
         [,1]     [,2] [,3]     [,4] [,5]     [,6]     [,7]
[1,] 0.000000 4.146541    0 5.363730    0 3.290745 3.389985
[2,] 4.146541 0.000000    0 2.803034    0 1.046518 3.419898
[3,] 0.000000 0.000000    0 0.000000    0 0.000000 0.000000
[4,] 5.363730 2.803034    0 0.000000    0 2.607566 2.533397
[5,] 0.000000 0.000000    0 0.000000    0 0.000000 0.000000
[6,] 3.290745 1.046518    0 2.607566    0 0.000000 2.478084
[7,] 3.389985 3.419898    0 2.533397    0 2.478084 0.000000
    X1   X2
1 4.36 5.21
2 3.90 4.27
         [,1] [,2] [,3]     [,4] [,5] [,6]     [,7]     [,8]
[1,] 0.000000    0    0 5.363730    0    0 3.389985 3.290745
[2,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[3,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[4,] 5.363730    0    0 0.000000    0    0 2.533397 2.607566
[5,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[6,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[7,] 3.389985    0    0 2.533397    0    0 0.000000 2.478084
[8,] 3.290745    0    0 2.607566    0    0 2.478084 0.000000
    X1   X2
1 3.75 1.12
2 4.10 1.80
3 4.36 5.21
4 3.90 4.27
          [,1] [,2] [,3]     [,4] [,5] [,6] [,7] [,8]     [,9]
 [1,] 0.000000    0    0 5.363730    0    0    0    0 3.290745
 [2,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [3,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [4,] 5.363730    0    0 0.000000    0    0    0    0 2.533397
 [5,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [6,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [7,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [8,] 0.000000    0    0 0.000000    0    0    0    0 0.000000
 [9,] 3.290745    0    0 2.533397    0    0    0    0 0.000000
```

```
    X1   X2
1 6.25 3.14
2 3.75 1.12
3 4.10 1.80
4 4.36 5.21
5 3.90 4.27
          [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]     [,10]
 [1,] 0.000000    0    0    0    0    0    0    0    0 3.290745
 [2,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [3,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [4,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [5,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [6,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [7,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [8,] 0.000000    0    0    0    0    0    0    0    0 0.000000
 [9,] 0.000000    0    0    0    0    0    0    0    0 0.000000
[10,] 3.290745    0    0    0    0    0    0    0    0 0.000000
    X1   X2
1 0.89 2.94
2 6.25 3.14
3 3.75 1.12
4 4.10 1.80
5 4.36 5.21
6 3.90 4.27
```

- : Explicacion

```
> agglomerativeHC.details(m,'EUC','MAX')

[[1]]
     [,1] [,2] [,3]
[1,] 0.89 2.94    1

[[2]]
     [,1] [,2] [,3]
[1,] 4.36 5.21    1

[[3]]
     [,1] [,2] [,3]
[1,] 3.75 1.12    1

[[4]]
     [,1] [,2] [,3]
[1,] 6.25 3.14    1

[[5]]
     [,1] [,2] [,3]
[1,]  4.1  1.8    1
```

```
[[6]]
     [,1] [,2] [,3]
[1,]  3.9 4.27    1

         [,1]     [,2]      [,3]     [,4]      [,5]     [,6]
[1,] 0.000000 4.146541 3.3899853 5.363730 3.4064204 3.290745
[2,] 4.146541 0.000000 4.1352388 2.803034 3.4198977 1.046518
[3,] 3.389985 4.135239 0.0000000 3.214094 0.7647876 3.153569
[4,] 5.363730 2.803034 3.2140940 0.000000 2.5333969 2.607566
[5,] 3.406420 3.419898 0.7647876 2.533397 0.0000000 2.478084
[6,] 3.290745 1.046518 3.1535694 2.607566 2.4780839 0.000000
    X1   X2
1 3.75 1.12
2 4.10 1.80
         [,1]     [,2] [,3]     [,4] [,5]     [,6]     [,7]
[1,] 0.000000 4.146541    0 5.363730    0 3.290745 3.406420
[2,] 4.146541 0.000000    0 2.803034    0 1.046518 4.135239
[3,] 0.000000 0.000000    0 0.000000    0 0.000000 0.000000
[4,] 5.363730 2.803034    0 0.000000    0 2.607566 3.214094
[5,] 0.000000 0.000000    0 0.000000    0 0.000000 0.000000
[6,] 3.290745 1.046518    0 2.607566    0 0.000000 3.153569
[7,] 3.406420 4.135239    0 3.214094    0 3.153569 0.000000
    X1   X2
1 4.36 5.21
2 3.90 4.27
         [,1] [,2] [,3]     [,4] [,5] [,6]     [,7]     [,8]
[1,] 0.000000    0    0 5.363730    0    0 3.406420 4.146541
[2,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[3,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[4,] 5.363730    0    0 0.000000    0    0 3.214094 2.803034
[5,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[6,] 0.000000    0    0 0.000000    0    0 0.000000 0.000000
[7,] 3.406420    0    0 3.214094    0    0 0.000000 4.135239
[8,] 4.146541    0    0 2.803034    0    0 4.135239 0.000000
    X1   X2
1 6.25 3.14
2 4.36 5.21
3 3.90 4.27
         [,1] [,2] [,3] [,4] [,5] [,6]     [,7] [,8]     [,9]
 [1,] 0.00000    0    0    0    0    0 3.406420    0 5.363730
 [2,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [3,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [4,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [5,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [6,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [7,] 3.40642    0    0    0    0    0 0.000000    0 4.135239
 [8,] 0.00000    0    0    0    0    0 0.000000    0 0.000000
 [9,] 5.36373    0    0    0    0    0 4.135239    0 0.000000
```

```
    X1   X2
1 0.89 2.94
2 3.75 1.12
3 4.10 1.80
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]    [,9]   [,10]
 [1,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [2,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [3,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [4,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [5,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [6,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [7,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [8,]    0    0    0    0    0    0    0    0 0.00000 0.00000
 [9,]    0    0    0    0    0    0    0    0 0.00000 5.36373
[10,]    0    0    0    0    0    0    0    0 5.36373 0.00000
    X1   X2
1 6.25 3.14
2 4.36 5.21
3 3.90 4.27
4 0.89 2.94
5 3.75 1.12
6 4.10 1.80
```

- : Explicacion

  >

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:LearnClust" "package:magick"
 [4] "package:arules"     "package:Matrix"     "package:stats"
 [7] "package:graphics"   "package:grDevices"  "package:utils"
[10] "package:datasets"   "package:methods"    "Autoloads"
[13] "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$Km)`
  - Parámetros:
    - `vector`: Vector de números.
  - Retorno: Devuelve el rango del vector introducido, es decir, la diferencia entre el número mayor y menor de dicho vector.
  - Explicación: Esta función se encarga de calcular el rango del vector introducido por parámetro. Va recorriendo el vector y durante cada iteración va comparando el valor actual con el máximo y el mínimo actuales, si dicho valor es menor o mayor que el mínimo o máximo respectivamente, actualiza el valor correspondiente. Una vez finalizado el recorrido calcula la diferencia entre el máximo y el mínimo final.
- Cálculo de la frecuencia absoluta ($\texttt{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.
- 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.

- 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)`

- 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.

- 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:
    $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

- 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:
  $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$

- Cálculo de la varianza ($\mathtt{s}^2$) 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:
    $s^2 = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$

- Cálculo de la mediana ($\tilde{\mathtt{x}}$) 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:
    $$\tilde{\mathtt{x}} = \begin{cases} x_{\frac{n+1}{2}} & \text{si } n \text{ es impar} \\ \frac{1}{2}(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}) & \text{si} n \text{ es par} \end{cases}$$

- Cálculo de laos cuartiles ($\mathtt{Q}_i$) 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:
    $$Q_p = \begin{cases} x_{\frac{np}{2}} + x_{\frac{np+1}{2}} & \text{si } np \in \mathbb{N} \\ x_{\lfloor np \rfloor + 1} & \text{si } np \notin \mathbb{N} \end{cases}$$

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 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 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 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 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
```