Universidad de Alcalá de Henares

Escuela Politécnica Superior

Fundamentos de la Ciencia de Datos

Autor:

Oscar Morán, Asier Álamo, Edgar Alexis Conforme, Lucía Díaz

Fecha:

11 de noviembre de 2025

Índice

1. Introducción **3**

2. Resolución de Ejercicios **3**

2.1. Ejercicio 1.1: Análisis Descriptivo (Satélites de Urano) 3

2.1.1. Descripción del conjunto de datos 3

2.1.2. Funciones de R utilizadas 3

2.1.3. Implementación del Análisis en R 4

2.2. Ejercicio 1.2: Análisis de asociación 8

2.2.1. Instalación y Carga de Paquetes 8

2.2.2. Creación y Formateo de los Datos 8

2.2.3. Análisis del Conjunto de Datos 9

2.2.4. Aplicación del Algoritmo Apriori 9

2.2.5. Conclusiones 10

2.3. Ejercicio 1.3: Detección de Datos Anómalos 12

2.3.1. Método 1: Caja y Bigotes 13

2.3.2. Método 2: Desviación Típica 15

2.3.3. Método 3: Regresión (Error de Residuos) 16

2.4. Ejercicio 1.4: Detección de datos anómalos mediante el método de proximidad (K-NN manual) y de densidad (LOF) 19

2.5. Ejercicio 2.1: Análisis Descriptivo de Distancias 27

2.6. Ejercicio 2.2: Fases del Algoritmo Apriori 44

2.6.1. Fase 1: Carga de Datos y Sucesos Elementales Candidatos 44

2.6.2. Fase 2: Generación de Sucesos Candidatos (L2, L3, L4) 46

2.6.3. Fase 3: Hash-Tree de sucesos candidatos para cada dimensión 49

2.6.4. Fase 4: Construcción de Hash Trees para cada suceso de la muestra para cada dimensión 51

2.6.5. Fase 5: Conteo sucesos candidatos para analizar la asociación de los que superen el umbral de soporte 54

2.6.6. Fase 6: Generación de Reglas de Asociación 58

2.6.7. Fase 7: Cálculo de Confianza y Filtrado Final 61

2.7. Ejercicio 2.3: Detección de Datos Anómalos 66
2.7.1. Funciones auxiliares comunes 66
2.7.2. Medidas de ordenación (Velocidad), Método Caja y Bigotes 67
2.7.3. Medidas de dispersión (Temperatura), Método Desviación Típica 68
2.7.4. Detección por regresión lineal 69
2.7.5. Aplicación de los métodos 70
2.8. Ejercicio 2.4: Detección de datos anómalos (Proximidad y Densidad Manual) 72
2.8.1. 1. Descripción del conjunto de datos 72
2.8.2. Funciones de R utilizadas en la exploración inicial 72
2.8.3. Carga y Exploración Visual 73
2.8.4. Método de Proximidad: K-Vecinos (Implementación Manual) 75
2.8.5. Método de Densidad: Local Outlier Factor (Implementación Manual) 78

3. Conclusiones **81**

1. Introducción

El objetivo de esta memoria es documentar el diseño e implementación de diversos algoritmos y técnicas de la ciencia de datos, utilizando el lenguaje de programación R.

Sweave nos permite generar este informe de manera dinámica, mezclando el texto explicativo con el código R que implementa los algoritmos.

2. Resolución de Ejercicios

A continuación, se detallan los ejercicios resueltos.

2.1. Ejercicio 1.1: Análisis Descriptivo (Satélites de Urano)

Este primer ejercicio consiste en la explicacion de un análisis descriptivo guiado por el profesor. Su propósito principal es asentar las bases del manejo de datos en R, utilizando las funciones estadísticas estándar que ofrece el lenguaje para caracterizar un conjunto de datos unidimensional.

2.1.1. Descripción del conjunto de datos

Los datos objeto de estudio corresponden a características físicas de los satélites menores del planeta Urano . El conjunto de datos contiene 12 observaciones y dos variables:

- **Nombre:** Variable cualitativa nominal que identifica a cada satélite (ej. *Cordelia*, *Ofelia*).
- **radio:** Variable cuantitativa continua que representa el radio medio del satélite en kilómetros.

La información se encuentra almacenada en un fichero de texto plano denominado `satelites.txt`, estructurado en formato tabular con cabecera.

2.1.2. Funciones de R utilizadas

Para realizar este análisis descriptivo se emplean funciones base de R. De acuerdo con los criterios de evaluación, a continuación se detalla su funcionalidad y los argumentos clave empleados:

- `read.table(file, header = ...)`: Función fundamental para la importación de datos tabulares desde ficheros de texto.
  - `file`: Ruta al archivo que contiene los datos (`"satelites.txt"`).
  - `header = TRUE`: Argumento crítico que indica a R que la primera línea del fichero no contiene datos, sino los nombres de las variables (*nombre*, *radio*).
- `order(x)` y `rev(x)`: Herramientas para la ordenación.
  - `order(x)` devuelve los *índices* que ordenarían el vector `x` de forma ascendente. No ordena el vector en sí, sino que nos dice "qué posición va primero".
  - `rev(x)` invierte el orden de cualquier objeto. Combinado con `order`, permite obtener ordenaciones descendentes de forma sencilla.

- `table(x)`: Genera una tabla de contingencia con las frecuencias absolutas de cada valor único presente en el vector `x`. Es el primer paso para construir tablas de frecuencias completas.
- `cumsum(x)`: Calcula la suma acumulada de los elementos de un vector numérico. Se aplica sobre las frecuencias absolutas o relativas para obtener sus correspondientes versiones acumuladas (N_i, F_i).
- `mean(x)` y `median(x)`: Calculan las principales medidas de tendencia central: la media aritmética y la mediana (el valor que ocupa la posición central cuando los datos están ordenados), respectivamente.
- `var(x)` y `sd(x)`: Calculan la varianza y la desviación típica. **Nota importante:** Por defecto, R calcula los estimadores *muestrales* (insesgados), es decir, utilizan $n-1$ en el denominador en lugar de N.
$$s^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$
- `quantile(x, probs)`: Función versátil para calcular cualquier cuantil. El argumento `probs` acepta un vector con las probabilidades deseadas (ej. `c(0.25, 0.50, 0.75)` para obtener los tres cuartiles principales Q_1, Q_2, Q_3).
- `max(x)` y `min(x)`: Devuelven los valores extremos del vector, esenciales para determinar el rango o recorrido de la variable.

2.1.3. Implementación del Análisis en R

A continuación se presenta el código completo y comentado que ejecuta el análisis descriptivo sobre la variable *radio*.

```
> # --- 1. Carga y verificación de datos ---
> # Leemos el fichero asumiendo que está en el directorio de trabajo actual.
> s <- read.table("satelites.txt", header = TRUE)
> print("Conjunto de datos original:")
```

```
[1] "Conjunto de datos original:"
```

```
> print(s)
```

```
        nombre radio
1     Cordelia    13
2       Ofelia    16
3       bianca    22
4       quesia    33
5    desdemora    29
6      julieta    42
7    Rosalinda    27
8      belinda    34
9  luna1986u10    20
10    calibano    30
11    luna9991    20
12  luna1999u2    15
```

```
> # --- 2. Ordenación de los datos ---
> # Es útil observar los datos ordenados para identificar rápidamente los extremos.
> # Orden ascendente por radio:
> idx_asc <- order(s$radio)
```

```
> so_asc <- s[idx_asc, ]
> # Orden descendente por radio (invirtiendo el orden ascendente):
> idx_desc <- rev(idx_asc)
> so_desc <- s[idx_desc, ]
> cat("\n--- Datos ordenados por radio (ascendente) ---\n")


--- Datos ordenados por radio (ascendente) ---


> print(so_asc)


        nombre radio
1     Cordelia    13
12  luna1999u2    15
2       Ofelia    16
9  luna1986u10    20
11    luna9991    20
3       bianca    22
7    Rosalinda    27
5    desdemora    29
10    calibano    30
4       quesia    33
8      belinda    34
6      julieta    42


> # --- 3. Medidas de Dispersión Básicas (Rango) ---
> # El rango cuantifica la amplitud total de los datos.
> min_radio <- min(s$radio)
> max_radio <- max(s$radio)
> rango_val <- max_radio - min_radio
> cat("\nRango del radio:", min_radio, "-", max_radio,
+     "(Amplitud:", rango_val, "km)\n")


Rango del radio: 13 - 42 (Amplitud: 29 km)


> # --- 4. Construcción de la Tabla de Frecuencias ---
> # Frecuencia Absoluta (ni)
> frec_abs <- table(s$radio)
> # Frecuencia Absoluta Acumulada (Ni)
> frec_abs_acum <- cumsum(frec_abs)
> # Frecuencia Relativa (fi): ni / N total
> n_total <- length(s$radio)
> frec_rel <- frec_abs / n_total
> # Frecuencia Relativa Acumulada (Fi)
> frec_rel_acum <- cumsum(frec_rel)
> # Combinamos todo en un data frame para una visualización clara y compacta
> tabla_frecuencias <- data.frame(
+   Radio = names(frec_abs),
+   ni = as.integer(frec_abs),
+   Ni = as.integer(frec_abs_acum),
+   fi = round(as.numeric(frec_rel), 3),       # Redondeo a 3 decimales
+   Fi = round(as.numeric(frec_rel_acum), 3)
+ )
> cat("\n--- Tabla de Frecuencias Completa (Radio) ---\n")
```

```
--- Tabla de Frecuencias Completa (Radio) ---
```

```
> print(tabla_frecuencias)
```

```
   Radio ni Ni    fi    Fi
1     13  1  1 0.083 0.083
2     15  1  2 0.083 0.167
3     16  1  3 0.083 0.250
4     20  2  5 0.167 0.417
5     22  1  6 0.083 0.500
6     27  1  7 0.083 0.583
7     29  1  8 0.083 0.667
8     30  1  9 0.083 0.750
9     33  1 10 0.083 0.833
10    34  1 11 0.083 0.917
11    42  1 12 0.083 1.000
```

```
> # --- 5. Medidas de Tendencia Central ---
> media_r <- mean(s$radio)
> mediana_r <- median(s$radio)
> cat("\n--- Medidas de Tendencia Central ---\n")
```

```
--- Medidas de Tendencia Central ---
```

```
> cat("Media aritmética:", round(media_r, 2), "km\n")
```

```
Media aritmética: 25.08 km
```

```
> cat("Mediana:", mediana_r, "km\n")
```

```
Mediana: 24.5 km
```

```
> # --- 6. Medidas de Dispersión y Posición ---
> # Recordamos que var() y sd() en R calculan versiones muestrales (n-1)
> var_r <- var(s$radio)
> sd_r <- sd(s$radio)
> # Calculamos los cuartiles principales (25%, 50%, 75%)
> cuartiles <- quantile(s$radio, probs = c(0.25, 0.50, 0.75))
> cat("\n--- Medidas de Dispersión y Posición ---\n")
```

```
--- Medidas de Dispersión y Posición ---
```

```
> cat("Varianza (muestral):", round(var_r, 2), "\n")
```

```
Varianza (muestral): 78.45
```

```
> cat("Desviación Típica (muestral):", round(sd_r, 2), "km\n")
```

```
Desviación Típica (muestral): 8.86 km

> cat("Cuartiles (Q1, Q2, Q3):\n")

Cuartiles (Q1, Q2, Q3):

> print(cuartiles)

  25%   50%   75%
19.00 24.50 30.75
```

Prompt de IA Utilizado

> I am tasked with performing the guided descriptive analysis for Exercise 1.1, as outlined in the lab PDF. The data source is the `satelites.txt` file, which includes 'nombre' (a qualitative variable) and 'radio' (a quantitative variable) for Urano's satellites. I require a comprehensive R script that utilizes the standard, built-in R functions to conduct this analysis. The script must demonstrate the correct application of functions such as `read.table()` for data loading, `order()` and `rev()` for data sorting, and `table()` combined with `cumsum()` and `length()` for generating the complete absolute, relative, and cumulative frequency tables. Furthermore, the script must conclude by calculating and printing all required statistical summaries for the 'radio' variable, including the `mean()`, `median()`, `var()`, `sd()`, and the primary quartiles using `quantile()`.

2.2. Ejercicio 1.2: Análisis de asociación

El propósito de este ejercicio es identificar las relaciones y patrones de compra frecuentes entre distintos productos usando Apriori.

2.2.1. Instalación y Carga de Paquetes

Para realizar el análisis se utilizó el paquete `arules`, especializado en minería de reglas de asociación y análisis de conjuntos de transacciones. Este paquete ofrece estructuras de datos eficientes para transacciones (clase `transactions`), algoritmos como `Apriori` y herramientas para medir soporte, confianza y lift de reglas de asociación. El primer paso fue intentar cargar dicho paquete mediante `library(arules)`. Al no estar instalado, se procedió con su instalación desde el repositorio CRAN.

```
> # install.packages("arules")
>
> library(arules)
```

2.2.2. Creación y Formateo de los Datos

Los datos de las cestas de la compra se representaron inicialmente como una matriz binaria, donde 1 indica la presencia de un artículo en una cesta y 0 su ausencia. Para poder utilizar el paquete arules, la matriz tuvo que ser convertida al formato `transactions`. Este proceso se realizó en varios pasos:

1. **Conversión a matriz lógica:** se utiliza la función `as(matriz, "nsparseMatrix")` para generar una matriz dispersa (`ngCMatrix`), eficiente en memoria cuando hay muchos ceros.
2. **Transposición:** con `t()`, se invierte la matriz para que los items queden en filas y las transacciones en columnas, cumpliendo la convención del paquete.
3. **Conversión a transacciones:** mediante `as(objeto, "transactions")`, se transforma la matriz lógica en un objeto compatible con `arules`, permitiendo aplicar algoritmos de asociación.

El paquete `Matrix` se emplea para manejar matrices dispersas de forma eficiente, especialmente útil en datasets grandes donde la mayoría de los elementos son ceros.

```
> library(Matrix)
> # Creación de la matriz binaria de cestas de compra
> 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", "Cafe", "Leche", "Naranja")),
+                   sparse=TRUE)
> # Conversión a matriz lógica y transposición
> muestrangCMatrix <- as(muestra, "nsparseMatrix")
> traspmuestrangCMatrix <- t(muestrangCMatrix)
> # Conversión final al formato transactions
```

```
> transacciones <- as(traspmuestrangCMatrix, "transactions")
> # Visualizar las transacciones
> transacciones
```

```
transactions in sparse format with
 6 transactions (rows) and
 5 items (columns)
```

2.2.3. Análisis del Conjunto de Datos

Con los datos en el formato correcto, se utilizó la función summary() para obtener una visión general de las transacciones. El resumen indicó que "Pan" y "Leche" son los artículos más frecuentes (presentes en 5 de las 6 cestas), y que el tamaño de las cestas varía de 1 a 4 artículos.

```
> 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    Cafe Naranja (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   Cafe

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

2.2.4. Aplicación del Algoritmo Apriori

Se aplicó el algoritmo apriori para extraer las reglas de asociación.

- `apriori(x, parameter = list(support = , confidence = ))`: recibe como argumento el objeto `transactions` y una lista de parámetros.
  - `support`: soporte mínimo requerido, indicando la frecuencia relativa mínima para que una regla se considere.

- `confidence`: confianza mínima de la regla, expresando la probabilidad condicional de que la presencia del antecedente implique la del consecuente.

- `inspect()`: muestra las reglas generadas en un formato legible, incluyendo soporte, confianza y lift.

Para este análisis, se estableció un soporte mínimo de 0.5 y una confianza mínima de 0.8, generando 7 reglas de asociación.

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

```
Apriori

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

```
> # Mostrar las reglas generadas
> 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
```

2.2.5. Conclusiones

Del análisis de las reglas se extraen varias conclusiones importantes:

1. **Productos Estrella:** "Pan" y "Leche" son los productos más comunes, como indican las reglas 1 y 2 y su soporte de 0.833.

2. **Asociación Fuerte (Agua y Pan):** La regla `{Agua}  {Pan}` tiene una confianza del 100 %, lo que significa que cada vez que un cliente compró "Agua", también compró "Pan". El `lift` de 1.20 indica que esta asociación es un 20 % más frecuente de lo esperado por azar.

3. **Asociación Recíproca:** La regla `{Pan}` `{Agua}` también es fuerte, con una confianza del 80 %.

4. **Regla de Tres Productos:** La regla `{Agua, Leche}` `{Pan}` tiene una confianza del 100 %, lo que implica que los tres clientes que compraron "Agua" y "Leche" juntos también compraron "Pan".

En resumen, el análisis revela una fuerte conexión de compra entre los productos "Pan" y "Agua", y también una relación significativa entre "Pan" y "Leche".

2.3. Ejercicio 1.3: Detección de Datos Anómalos

En este ejercicio se aplican tres métodos distintos para la detección de datos anómalos sobre un conjunto de datos de muestra.

Descripción del conjunto de datos

El conjunto de datos utilizado es una pequeña muestra de 7 observaciones, cada una con dos variables cuantitativas:

- **r**: variable cuantitativa (resistencia).
- **d**: variable cuantitativa (densidad).

Los datos se crean directamente en R y se almacenan en un `data.frame` llamado `muestra`. El objetivo es aplicar métodos univariantes (Caja y Bigotes sobre 'r', Desviación Típica sobre 'd') y un método bivariante (Regresión sobre 'd' en función de 'r') para identificar observaciones atípicas.

Implementación y Carga de Datos

El primer paso es crear la estructura de datos. Los datos se introducen en una matriz y luego se transforman en un `data.frame` para un manejo más fácil.

```
> # La siguiente línea de código se ha dividido en varias
> # para evitar que se salga del margen al imprimirse.
> (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
```

Explicación de las funciones utilizadas:

- `matrix(data, nrow, ncol, dimnames)`: Crea una estructura de matriz. En este caso, se le pasa un vector de 14 números. Con `nrow=2` y `ncol=7`, R rellena la matriz por columnas por defecto: la primera columna es `c(3, 2)`, la segunda `c(3.5, 12)`, y así sucesivamente.
- `t(x)`: Función que transpone su argumento (matriz o data frame), intercambiando filas por columnas. La matriz 2x7 se convierte en una matriz 7x2, donde cada fila representa una observación y cada columna una variable ('r' o 'd').
- `data.frame(x)`: Convierte su argumento (en este caso, la matriz 7x2) en un `data.frame`. Esta es la estructura de datos estándar en R para análisis estadístico, permitiendo acceder a las columnas por su nombre (ej. `muestra$r`).

2.3.1. Método 1: Caja y Bigotes

Fundamento Teórico

El método de la caja y bigotes (Boxplot) es una técnica de detección de atípicos no paramétrica (no asume una distribución específica de los datos). Se basa en el Rango Intercuartílico (IQR), que es la diferencia entre el tercer cuartil (Q_3, percentil 75) y el primer cuartil (Q_1, percentil 25).

Este método define un intervalo de "normalidadz considera atípico (outlier) cualquier punto que caiga fuera de él. Los límites de este intervalo se calculan como:

$$\text{Límite Inferior} = Q_1 - d \times IQR$$

$$\text{Límite Superior} = Q_3 + d \times IQR$$

Comúnmente, se utiliza un factor $d = 1{,}5$. Los valores fuera de este rango se consideran atípicos leves.

Explicación de las funciones utilizadas

- `boxplot(x, range=1.5, plot=FALSE)`: Aunque su uso principal es gráfico, con `plot=FALSE` la función no dibuja nada, sino que devuelve una lista con las estadísticas calculadas. El componente `$out` de esta lista contiene los valores identificados como anómalos según el criterio del `range` (por defecto 1.5).
- `quantile(x, probs)`: Calcula los cuantiles de la distribución correspondientes a las probabilidades indicadas en `probs`. Se usa para obtener Q_1 (`probs = 0.25`) y Q_3 (`probs = 0.75`).
- `length(x)`: Devuelve el número de elementos (longitud) del vector `x`. Se usa en el bucle `for` para iterar sobre todas las observaciones de la variable 'r'.

Explicación del Método

Para la detección de datos anómalos por este método se debe llevar a cabo siguiendo 4 pasos. 1^o se determina el valor del grado del outlier (d). 2^o Se ordenan los datos para el parámetro que se quiere detectar los datos anómalos, en este caso la resistencia o r. 3^o Se calculan los valores del $cuartil_1$ y del $cuartil_3$. 4^o se observa que valores quedan fuera del intervalo, el cual sería: $[cuartil_1 - d * (cuartil_3 - cuartil_1), cuartil_3 + d * (cuartil_3 - cuartil_1)]$

Código y Ejecución (Sweave)

```
> # 1. Metodo de Caja y Bigotes (para la columna 'r')
> (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"


> # Calculo manual
> (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


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

```
[1] "el suceso"
[1] 7
[1] 14
[1] "es un outlier"
```

Interpretación de resultados

La salida de `boxplot(...)$out` identifica el valor `14` como un outlier. El cálculo manual corrobora esto: el Q_1 es 4.1 y el Q_3 es 6.65. El IQR es $6{,}65 - 4{,}1 = 2{,}55$. El intervalo de aceptación es $[4{,}1 - 1{,}5 \times 2{,}55, 6{,}65 + 1{,}5 \times 2{,}55]$, que resulta en $[0{,}275, 10{,}475]$. El valor 14.0 (correspondiente al suceso 7) es mayor que 10.475, por lo que el bucle `for` lo identifica correctamente como ün outlier".

2.3.2. Método 2: Desviación Típica

Fundamento Teórico

Este método paramétrico asume que los datos siguen una distribución normal (Gaussiana). En una distribución normal, la gran mayoría de los datos se concentran alrededor de la media. Según la regla empírica (68-95-99.7):

- $\approx 68\,\%$ de los datos está a 1 desviación típica (σ) de la media (μ).
- $\approx 95{,}5\,\%$ de los datos está a 2 desviaciones típicas.
- $\approx 99{,}7\,\%$ de los datos está a 3 desviaciones típicas.

El método define un intervalo de normalidad basado en la media y la desviación típica. Un valor de d (grado de outlier) común es 2. Los puntos que caen fuera del intervalo $[\mu - d \times \sigma, \mu + d \times \sigma]$ se consideran anómalos. En este ejercicio se usa $d = 2$.

Explicación de las funciones utilizadas

- `mean(x)`: Calcula la media aritmética ($\bar{x}$) del vector `x`.
- `var(x)`: Calcula la varianza **muestral** (s^2), que divide por $(n - 1)$.
- `length(x)`: Devuelve n. El código calcula la varianza **poblacional** (σ^2) a partir de la muestral, usando la fórmula $\sigma^2 = s^2 \times \frac{n-1}{n}$.
- `sqrt(x)`: Calcula la raíz cuadrada, usada para obtener la desviación típica (σ) a partir de la varianza poblacional.

Explicación del Método

Para la detección de datos anómalos por este método se debe llevar a cabo los siguiente 4 pasos. 1^o se determina el valor del grado de outlier(d). 2^o se debe hallar el valor de la media para el parámetro que se nos halla pedido, en este caso densidad o d (d_{mean}).3^o Se debe calcular el valor de las desviación típica para dicho parámetro (d_{desv}). 4^o Se debe ver que valores quedan fuera del intervalo, el cual sería: $[d_{mean} - d * d_{desv}, d_{mean} + d * d_{desv}]$

Código y Ejecución (Sweave)

```
> # 2. Metodo de Desviacion Tipica (para la columna 'd')
>
> # Calculo de la desviacion tipica poblacional
> sdd<-sqrt(var(muestra$d)*((length(muestra$d)-1)/length(muestra$d)))
> # Calculo del intervalo
> (intdes<-c(mean(muestra$d)-2*sdd, mean(muestra$d)+2*sdd))
```

```
[1] -0.05685714 11.37114285
```

```
> # Bucle para detectar outliers
> 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] 2
[1] 12
[1] "es un outlier"
```

Interpretación de resultados

La media de 'd' es 5.657 y la desviación típica poblacional (sdd) es 2.857. El intervalo de aceptación (con $d = 2$) es $[5{,}657 - 2 \times 2{,}857, 5{,}657 + 2 \times 2{,}857]$, que resulta en $[-0{,}057, 11{,}371]$. El bucle `for` itera sobre la variable 'd' y encuentra que el valor 12.0 (suceso 2) es mayor que 11.371, identificándolo como ün outlier".

2.3.3. Método 3: Regresión (Error de Residuos)

Fundamento Teórico

Este método se usa para detectar outliers en datos bivariantes, identificando puntos que no se ajustan a la relación o tendencia general del resto de los datos. Primero, se ajusta un modelo de regresión lineal (generalmente por mínimos cuadrados) para predecir y a partir de x ($\hat{y} = a + bx$). Para cada punto (x_i, y_i), se calcula el **residuo** o error, que es la diferencia entre el valor real y el valor predicho: $e_i = y_i - \hat{y}_i$. Se calcula el error estándar de estos residuos (Sr). Un punto se considera anómalo si su residuo (en valor absoluto) es inusualmente grande, es decir, si $|e_i| > d \times Sr$ (donde d suele ser 2 o 3). Esto significa que el punto está muy lejos (verticalmente) de la línea de tendencia.

Explicación de las funciones utilizadas

- `lm(formula)`: Siglas de "Linear Model". Es la función principal de R para ajustar modelos de regresión lineal. La fórmula `muestra$d  muestra$r` especifica que se debe modelar 'd' (variable dependiente) en función de 'r' (variable independiente).
- `summary(object)`: Proporciona un resumen detallado de un objeto modelo. Aplicado a un objeto `lm`, muestra los coeficientes, el R-cuadrado, estadísticos F y, fundamentalmente, los residuos del modelo.

- `summary(dfr)$residuals`: Esta sintaxis accede al vector numérico de los residuos (los e_i) que está almacenado dentro del objeto `summary`.
- `sum(x)`: Suma los elementos de un vector. Se usa aquí para calcular la suma de los residuos al cuadrado (parte del cálculo del Sr).

Explicación del Método

Para la detección de datos anómalos por este método se deben llevar a cabo los 7 siguientes pasos. 1^o Determinar el valor del grado de outlier (d). 2^o Se deben hallar los valores de a y b para poder tener una forma de predecir los valores de la densidad que vamos a llamar (y_{hat}). Para esto se tiene que hallar el valor de la covarianza, ya que el valor de b sería la división del valor de la covarianza / la varianza de la resistencia (x); por último para hallar a: $a = y_{mean} - b \times x_{mean}$. De esta forma ya se puede obtener predicciones de los valores de la densidad: $y_{hat} = a + b \times x_i$. 3^o Se obtienen los valores predichos de la densidad para cada valor de la resistencia (x_i). 4^o Se calcula el error estandar de los residuos (Sr). 5^o Se obtiene el valor identicador de outliers que sería $d \times Sr$. 6^o Se obtiene el valor absoluto de la diferencia entre los valores de y_i y los y_{hat}. 7^o Aquellos valores cuya diferencia supere el valor de $d \times Sr$ serán considerados outliers.

Código y Ejecución (Sweave)

```
> # 3. Metodo de Regresion (para 'd' en funcion de 'r')
>
> # Calculo del modelo de regresion lineal
> (dfr<-lm(muestra$d~muestra$r))


Call:
lm(formula = muestra$d ~ muestra$r)

Coefficients:
(Intercept)    muestra$r
    6.01445     -0.05723


> (summary(dfr))


Call:
lm(formula = muestra$d ~ muestra$r)

Residuals:
      1       2       3       4       5       6       7
-3.84275  6.18587 -1.64545 -0.81683  0.49192 -0.45960  0.08684

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.01445    2.64632   2.273   0.0722 .
muestra$r   -0.05723    0.37148  -0.154   0.8836
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.372 on 5 degrees of freedom
Multiple R-squared:  0.004725,        Adjusted R-squared:  -0.1943
F-statistic: 0.02374 on 1 and 5 DF,  p-value: 0.8836
```

```
> # Obtencion de residuos
> (res<-summary(dfr)$residuals)
```

```
         1          2          3          4          5          6          7
-3.8427477  6.1858698 -1.6454482 -0.8168308  0.4919157 -0.4595958  0.0868370
```

```
> # Calculo del error estandar de los residuos
> sr<-sqrt(sum(res^2)/length(res))
> sr
```

```
[1] 2.850242
```

```
> # Bucle para detectar outliers en los residuos
> for (i in 1:length(res))
+ {if (res[i]>2*sr)
+ {print("el suceso");
+ print(res[i]); print("es un outlier")}}
```

```
[1] "el suceso"
      2
6.18587
[1] "es un outlier"
```

Interpretación de resultados

El modelo de regresión ajustado es $d = 6{,}01445 - 0{,}05723 \times r$. El error estándar de los residuos (Sr) es 2.85. El umbral de detección es $2 \times Sr = 5{,}70$. El vector de residuos (los e_i) es `c(-3.84, 6.18, -1.64, ...)`. El bucle `for` compara cada residuo con el umbral 5.70. Encuentra que el segundo residuo, 6.18587, es mayor que el umbral, por lo que imprime que el suceso 2 es ün outlier". Esto significa que el punto (3.5, 12.0) está significativamente por encima de la línea de regresión que describe la tendencia general de los datos.

2.4. Ejercicio 1.4: Detección de datos anómalos mediante el método de proximidad (K-NN manual) y de densidad (LOF)

Descripción del conjunto de datos

El conjunto de datos utilizado está formado por las calificaciones de cinco estudiantes, donde cada observación se compone de dos variables cuantitativas: la nota de **Teoría** y la de **Laboratorio**. Las observaciones se representan como pares ordenados:

$$(4,4),\ (4,3),\ (5,5),\ (1,1),\ (5,4)$$

El objetivo del ejercicio es identificar posibles *datos anómalos* o *outliers* que se alejen significativamente del resto del conjunto, utilizando un método basado en la **proximidad** entre observaciones en el espacio bidimensional de calificaciones. Posteriormente, se vuelve a realizar el ejercicio emplean un método basado en la densidad.

Fundamento teórico

El método de los *K-vecinos más próximos* (K-NN) para detección de anomalías se fundamenta en el supuesto de que los puntos normales se encuentran próximos entre sí, mientras que los puntos anómalos se sitúan aislados a mayor distancia de sus vecinos.

Formalmente, para cada observación $x_i = (x_{i1}, x_{i2})$, se calcula la distancia euclídea respecto a las demás:

$$d_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2}$$

Estas distancias se ordenan de menor a mayor, y se obtiene la distancia al K-ésimo vecino más próximo, $d_K(x_i)$. Si este valor supera un umbral prefijado d^*, la observación se considera un *dato anómalo*:

$$x_i \text{ es anómalo si } d_K(x_i) > d^*$$

En este caso se utiliza $K = 3$ y $d^* = 2{,}5$.

A diferencia del método de proximidad K-NN, el algoritmo **LOF (Local Outlier Factor)** se basa en la comparación de *densidades locales* en lugar de distancias absolutas. La idea central es que los puntos anómalos no sólo se encuentran lejos de los demás, sino que además están situados en regiones de baja densidad respecto a sus vecinos más próximos.

En este caso, se emplea la **distancia de Manhattan** como medida de proximidad entre observaciones. Dada dos observaciones $p = (x_1, y_1)$ y $o = (x_2, y_2)$, la distancia Manhattan se define como:

$$d(p, o) = |x_1 - x_2| + |y_1 - y_2|$$

Este tipo de distancia es especialmente útil cuando las variables representan magnitudes comparables y se desea penalizar menos los desplazamientos diagonales que en la distancia euclídea.

El algoritmo LOF sigue los siguientes pasos conceptuales:

1. **Cálculo de las distancias y determinación de los k vecinos más cercanos.** Para cada punto p_i se calculan sus distancias Manhattan respecto a todos los demás puntos del conjunto, y se identifican sus k vecinos más próximos $N_k(p_i)$.
2. **Estimación de la densidad local.** La densidad local de un punto se define como el inverso de la distancia media a sus k vecinos más cercanos:

$$D(p_i) = \frac{1}{\frac{1}{k}\sum_{o \in N_k(p_i)} d(p_i, o)}$$

De este modo, los puntos situados en regiones densas tienen una densidad alta, mientras que los que se encuentran aislados presentan una densidad baja.

3. **Cálculo del grado de rareza o *DRM*.** El *DRM* de una observación se obtiene comparando su densidad local con la densidad media de sus vecinos:

$$DRM(p_i) = \frac{\frac{1}{k}\sum_{o \in N_k(p_i)} D(o)}{D(p_i)}$$

4. **Interpretación del resultado.**
   - Si $DRM(p_i) \approx 1$, el punto tiene una densidad similar a la de sus vecinos, por lo que se considera normal.
   - Si $DRM(p_i) > 1$, el punto se encuentra en una región menos densa que la de sus vecinos, por lo que se considera potencialmente anómalo.
   - Valores significativamente superiores a 1 indican *anomalías fuertes*.

En resumen, el método LOF evalúa la anomalía de una observación no sólo por su distancia a los demás puntos, sino por su densidad relativa en el entorno local.

Implementación en R de K-NN manual

El algoritmo mencionado es implementado de forma completa mediante el uso de funciones de base en R, sin recurrir a paquetes externos.

1) Definición del conjunto de datos. El conjunto de datos, como se ha mencionado, está formado por cinco estudiantes con sus respectivas calificaciones en dos dimensiones: **Teoría** y **Laboratorio**.

```
> # Creación de la matriz (2 variables x 5 observaciones)
> muestra <- matrix(c(4,4,  4,3,  5,5,  1,1,  5,4), nrow = 2, ncol = 5)
> muestra
```

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

La función `matrix()` construye una matriz a partir de un vector. Sus argumentos principales son:

- `data`: vector con los valores que se distribuirán en la matriz (en este caso, las calificaciones de teoría y laboratorio).
- `nrow` y `ncol`: número de filas y columnas respectivamente.
- Por defecto, R rellena la matriz por columnas, por lo que en esta configuración cada **columna** representa un estudiante y cada **fila** una variable.

Sin embargo, las funciones de análisis de distancias en R esperan que las observaciones estén en filas. Por ello, se realiza una transposición de la matriz para adecuarla a este formato:

```
> # Transposición: filas = estudiantes, columnas = variables
> muestra <- t(muestra)
> muestra
```

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

La función `t()` (de *transpose*) intercambia filas y columnas de la matriz, de forma que cada **fila** corresponde ahora a un estudiante y cada **columna** a una variable (Teoría o Laboratorio).

2) Cálculo de distancias euclídeas. Se calculan todas las distancias euclídeas entre los pares de observaciones. La distancia entre dos puntos (x_1, y_1) y (x_2, y_2) se define como:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

En R, esto se implementa con la función `dist()`:

```
> # Cálculo de la matriz de distancias euclídeas
> distancia <- as.matrix(dist(muestra))
> distancia
```

```
         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
```

- `dist(x, method = "euclidean")` calcula una matriz de distancias por pares entre las filas de `x`, usando por defecto la métrica euclídea.
- La función devuelve un objeto de tipo `dist`, que se convierte en una matriz convencional mediante `as.matrix()` para su lectura y posterior procesamiento.

Cada elemento d_{ij} de la matriz representa la distancia entre los estudiantes i y j. Todos los elementos de la diagonal son cero, ya que la distancia de un punto a sí mismo es nula.

3) Ordenación de las distancias. Con el objetivo de encontrar los K-vecinos más próximos, se ordenan las distancias para cada observación en orden ascendente:

```
> # Ordenar distancias (de menor a mayor) para cada observación
> for (i in 1:5) {
+   distancia[, i] <- sort(distancia[, i])
+ }
> distanciasord <- distancia
> distanciasord
```

```
         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
```

- El bucle `for` itera sobre las cinco observaciones (una por estudiante).
- La función `sort(x)` ordena los valores del vector `x` en orden ascendente.
- La matriz resultante `distanciasord` contiene las distancias ordenadas por columnas, donde las filas corresponden a los vecinos más cercanos.

La primera fila de cada columna es siempre cero (distancia al mismo punto), seguida por las distancias a los vecinos más próximos. La cuarta fila contiene la distancia al tercer vecino más cercano ($K = 3$).

4) Detección de valores anómalos. Se establece un umbral de decisión $d^* = 2{,}5$. Aquellas observaciones cuya distancia al tercer vecino supere el umbral se consideran anómalas:

```
> # Umbral de decisión
> umbral <- 2.5
> # Identificación de outliers
> for (i in 1:5) {
+   if (distanciasord[4, i] > umbral) {
+     print(i)
+     print("es un outlier")
+   }
+ }
```

```
[1] 4
[1] "es un outlier"
```

- `umbral` fija la distancia límite para considerar una observación como anómala.
- La condición `if(distanciasord[4, i] >umbral)` evalúa si la distancia al tercer vecino (posición 4 en la columna, contando la propia observación) excede ese límite.
- Si se cumple, se imprime el número de la observación y el mensaje "es un outlier".

El resultado muestra:

```
[1] 4
[1] "es un outlier"
```

Esto indica que el estudiante 4, con calificaciones (1,1), es un dato anómalo según el criterio establecido.

5) Interpretación y análisis. En el espacio bidimensional de las calificaciones, los puntos $(4, 4)$, $(4, 3)$, $(5, 5)$ y $(5, 4)$ se agrupan cercanamente, representando rendimientos similares. En contraste, el punto $(1, 1)$ está alejado del resto, a una distancia superior a cuatro unidades euclídeas. Este aislamiento explica su clasificación como outlier mediante el criterio de proximidad.

Analizándolo de forma geométrica, las observaciones se encuentran en la zona superior derecha del plano, mientras que el punto anómalo se sitúa en la esquina inferior izquierda. Este método basado en la distancia euclídea ha permitido identificar eficazmente la observación que difiere significativamente del patrón general del conjunto de datos, en base a los criterios de proximidad establecidos.

Implementación en R de LOF (Local Outlier Factor).

A continuación, se implementa el método LOF (Local Outlier Factor) mediante el paquete `Rlof`, que automatiza el cálculo de la densidad local y la comparación con las densidades de los vecinos próximos. Este enfoque complementa el método K-NN al introducir el análisis de densidades relativas en lugar de distancias absolutas, permitiendo una evaluación más robusta de la rareza de cada observación en función de la **densidad local relativa**.

1) Preparación del entorno y definición del conjunto de datos. Se instala y carga el paquete `Rlof`, y se define la matriz de calificaciones con las dos variables **Teoría** y **Laboratorio** correspondientes a los cinco estudiantes.

```
> # Instalación y carga del paquete Rlof
> # install.packages("Rlof")
> library(Rlof)
> # Definición de la matriz de datos (5 observaciones x 2 variables)
> muestra <- matrix(c(4,4,
+                     4,3,
+                     5,5,
+                     1,1,
+                     5,4),
+                   byrow = TRUE, ncol = 2)
> # Asignación de nombres a filas y columnas
> colnames(muestra) <- c("Teoria", "Laboratorio")
> rownames(muestra) <- 1:5
> # Visualización de la matriz de datos
> print("Matriz de datos:")
```

```
[1] "Matriz de datos:"
```

```
> print(muestra)
```

```
  Teoria Laboratorio
1      4           4
2      4           3
3      5           5
4      1           1
5      5           4
```

En esta matriz, cada fila representa a un estudiante y cada columna una calificación. La estructura bidimensional permite aplicar directamente el análisis de proximidad y densidad mediante las funciones del paquete.

2) Cálculo del factor LOF con distancia de Manhattan. Se elige el número de vecinos $k = 3$ para definir la vecindad local de cada observación. La función principal utilizada, `lof()`, permite especificar la métrica de distancia mediante el argumento `method`. En este caso se selecciona la **distancia Manhattan**, definida como:

$$d_M(p, q) = \sum_{i=1}^{n} |p_i - q_i|$$

Esta métrica resulta especialmente útil en variables cuantitativas de escala comparable, ya que mide la discrepancia entre observaciones como la suma de diferencias absolutas por dimensión.

```
> # Selección del parámetro k
> k <- 3
> # Cálculo del factor LOF utilizando distancia de Manhattan
> lof_scores <- Rlof::lof(muestra, k = k, method = "manhattan")
> # Visualización de los valores LOF (redondeados a 3 decimales)
> print(paste("Valores LOF para k =", k, "usando distancia Manhattan:"))
```

```
[1] "Valores LOF para k = 3 usando distancia Manhattan:"
```

```
> print(round(lof_scores, 3))
```

```
[1] 1.095 0.917 0.917 2.357 1.095
```

El vector `lof_scores` contiene un valor por observación. Un valor cercano a 1 indica que la densidad local del punto es similar a la de sus vecinos; valores significativamente superiores a 1 implican una menor densidad local y, por tanto, una mayor probabilidad de anomalía.

3) Identificación de observaciones anómalas. Se establece un umbral de decisión $LOF > 1{,}5$. Las observaciones cuyo valor LOF supera dicho umbral se clasifican como **datos anómalos**.

```
> # Umbral de decisión
> umbral <- 1.5
> # Identificación de observaciones anómalas
> outliers <- which(lof_scores > umbral)
> # Presentación de resultados
> if (length(outliers) == 0) {
+   cat("\nNo se detectaron outliers (LOF <", umbral, ")\n")
+ } else {
+   cat("\nObservaciones detectadas como outliers (LOF >", umbral, "):\n")
+   print(outliers)
+   print("Coordenadas de los outliers:")
+   print(muestra[outliers, ])
+ }
```

```
Observaciones detectadas como outliers (LOF > 1.5 ):
[1] 4
[1] "Coordenadas de los outliers:"
     Teoria Laboratorio
          1           1
```

En este caso, la salida del programa muestra el índice de las observaciones detectadas como anómalas y sus coordenadas en el espacio bidimensional de calificaciones.

4) Interpretación de resultados. El análisis mediante el método LOF utilizando la distancia de Manhattan produce resultados coherentes con el método K-NN: el estudiante 4, con calificaciones $(1, 1)$, presenta una densidad local marcadamente inferior respecto al resto del conjunto. Esto confirma su

condición de **dato anómalo**, ya que la suma de las diferencias absolutas respecto a los demás puntos supera ampliamente las observadas entre los restantes estudiantes.

El uso del paquete `Rlof` y de la métrica de Manhattan aporta una perspectiva complementaria: en lugar de evaluar sólo la proximidad geométrica, el análisis considera la **densidad local relativa**, lo que permite identificar puntos aislados incluso en regiones de diferente dispersión. Además, la posibilidad de ajustar la métrica de distancia y el número de vecinos convierte a este método en una herramienta flexible para la detección de anomalías en conjuntos de datos de distinta naturaleza.

Descripción del paquete `Rlof`. El paquete `Rlof` (*R Parallel Implementation of Local Outlier Factor*) proporciona una implementación optimizada y paralelizada del algoritmo LOF propuesto. Su principal objetivo es acelerar el cálculo del **factor de aislamiento local** mediante el uso de múltiples núcleos de CPU y estructuras de datos eficientes para la gestión de distancias.

El paquete depende de los módulos `doParallel` y `foreach`, que permiten el procesamiento en paralelo de varios valores de k y la ejecución simultánea de cálculos de densidad local. De esta forma, se incrementa significativamente el rendimiento en conjuntos de datos de gran tamaño.

Entre sus características más relevantes destacan:

- Soporte para múltiples medidas de distancia (`.euclidean`, `"manhattan"`, `canberra"`, `"binary"`, `"minkowski"`), especificadas mediante el argumento `method`.
- Posibilidad de calcular varios valores de k en paralelo (`k` puede ser un vector).
- Utilización automática de todos los núcleos disponibles si no se indica el parámetro `cores`.
- Compatibilidad con matrices y `data.frame` como estructuras de entrada.

Internamente, el paquete implementa dos funciones principales:

Internamente, el paquete implementa dos funciones principales:

1. `distmc()`: calcula una matriz de distancias entre observaciones con soporte multihilo. Es funcionalmente equivalente a `dist()` del paquete base `stats`, pero permite paralelización y soporta un mayor número de métricas. Su sintaxis general es:

   ```
   distmc(x, method = .euclidean", diag = FALSE, upper = FALSE, p = 2)
   ```

   donde:

   - `x`: matriz o `data.frame` numérico que contiene las observaciones en filas y las variables en columnas.
   - `method`: determina la métrica de distancia a utilizar; puede ser `.euclidean`, `"manhattan"`, `canberra"`, `"binary"` o `"minkowski"`.
   - `diag`: indica si la diagonal debe mostrarse en la matriz de salida (`FALSE` por defecto).
   - `upper`: controla si se devuelve la parte superior de la matriz (`FALSE` por defecto).
   - `p`: parámetro de la distancia de Minkowski (por defecto $p = 2$, que equivale a la distancia euclídea).

   La función devuelve un objeto de tipo `dist`, que puede transformarse en una matriz mediante `as.matrix()` para su visualización o uso posterior. En este trabajo se especifica el argumento `method = "manhattan"` para utilizar la distancia de Manhattan, más robusta frente a variaciones grandes en una única variable.

2. `lof()`: calcula el **Local Outlier Factor** (LOF) para cada observación del conjunto de datos, comparando su densidad local con la de sus k vecinos más próximos. La llamada básica es:

   `lof(data, k, cores = NULL, method = .euclidean")`

   donde:

   - `data`: conjunto de datos en formato matriz o `data.frame`.
   - `k`: número de vecinos más próximos considerados para el cálculo del LOF.
   - `cores`: número de núcleos de CPU utilizados para el cálculo en paralelo. Si se deja en `NULL`, se emplean todos los disponibles.
   - `method`: métrica de distancia usada para determinar los vecinos más cercanos, idéntica a la de `distmc()`.

   La función devuelve un vector numérico del mismo tamaño que el número de observaciones, donde cada elemento representa el valor LOF_i asociado a la observación i. Valores cercanos a 1 indican una densidad local similar a la de los vecinos (observación normal), mientras que valores significativamente mayores que 1 reflejan una menor densidad relativa, identificando la observación como potencial **outlier**.

La combinación de ambas funciones permite aplicar el método LOF de forma flexible, eficiente y reproducible, con control explícito sobre la métrica de distancia y el número de vecinos. En este proyecto, `Rlof` se emplea con la distancia de Manhattan y $k = 3$, optimizando el cálculo de densidades locales y detectando automáticamente los puntos con menor densidad relativa en el espacio de calificaciones.

[1]

[1] Basado en la documentación del paquete `Rlof` (versión 1.1.3), desarrollado por Yingsong Hu, Wayne Murray y Yin Shan. Publicado en CRAN el 16 de octubre de 2022.

2.5. Ejercicio 2.1: Análisis Descriptivo de Distancias

Descripción del conjunto de datos y objetivo

En este ejercicio se hace uso de un fichero `distancias.txt`, que dispone de observaciones emparejadas de:

- **Población de residencia** (variable cualitativa nominal), y
- **Distancia en km** desde el domicilio del estudiante hasta la Universidad (variable cuantitativa continua).

El enunciado cuenta con el universo de datos y los valores a emplear en este apartado, por ejemplo: *Villalbilla, 16.5; Ensanche de Vallecas, 34.8; Alcalá de Henares, 6.2; ...; Madrid, 46; ND, 3.7; ...*. La presencia del marcador **ND** indica casos con localidad no disponible para esa observación, mientras que la distancia sí está informada. Estos pares población, distancia constituyen la base para el **análisis descriptivo** requerido en el ejercicio 2.1: ordenar por distancia, calcular rango, tablas de frecuencias (absoluta, acumulada, relativa) e indicadores de tendencia y dispersión sobre la variable cuantitativa.

Estructura del fichero. El fichero `distancias.txt` se lee con cabecera (`header = TRUE`) y contiene dos columnas:

`población`	(carácter/factor): nombre de la población o `ND` si no consta.
`distancia`	(numérico, km): distancia real en kilómetros con decimales.

Calidad y decisiones sobre datos.

- **Unidades y formato:** la distancia está en kilómetros; se preservan los decimales tal y como aparecen en el enunciado (p. ej., 16.5, 34.8, 31.4).
- **Valores ND en `población`:** no impiden el análisis cuantitativo porque la métrica principal se aplica sobre `distancia`. Se mantienen sin imputación ni eliminación, preservando la consistencia del conjunto.
- **Duplicados intencionales:** algunas parejas se repiten (p. ej., *Cifuentes, 24* o varias *Guadalajara, 30*), lo que es coherente con un conteo de frecuencias donde se esperan empates de distancia por redondeo.

Fundamentos teóricos del análisis descriptivo

El análisis estadístico descriptivo tiene como finalidad resumir y comprender la información contenida en un conjunto de datos mediante medidas que describen su tendencia central, su variabilidad y su forma. En este ejercicio se aplican dichos conceptos sobre la variable `distancia`, de naturaleza cuantitativa continua.

Tipos de variables. De acuerdo con la tipología presentada en la Lección 1:

- Una **variable cualitativa nominal** (como `población`) clasifica observaciones en categorías sin orden inherente.
- Una **variable cuantitativa continua** (como `distancia`) toma valores numéricos dentro de un rango real y admite decimales.

Frecuencias y distribuciones. Sea un conjunto de observaciones $x_1, x_2, \ldots, x_n$. La **frecuencia absoluta** f_i indica cuántas veces aparece cada valor o intervalo. La **frecuencia relativa** r_i se define como $r_i = f_i/n$, expresando la proporción de casos. Las **frecuencias acumuladas** (F_i, R_i) suman las observaciones hasta cada valor:

$$F_i = \sum_{j \leq i} f_j, \qquad R_i = \sum_{j \leq i} r_j$$

Estas medidas permiten construir la distribución empírica de la variable y analizar su forma.

Medidas de tendencia central. Entre los valores posibles de una variable cuantitativa, la **media aritmética** representa el centro de gravedad de la distribución:

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

La media es sensible a valores extremos y resume el nivel típico de la variable.

Medidas de dispersión. La **dispersión** refleja cuánto se alejan los datos de la media:

- El **rango** mide la amplitud total de la distribución: $\text{máx}(x) - \text{mín}(x)$.
- La **varianza poblacional** cuantifica la media de las desviaciones cuadráticas:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2$$

- La **varianza muestral** incorpora la corrección de Bessel:

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

- La **desviación estándar** es la raíz cuadrada de la varianza y mantiene las mismas unidades que la variable original.

Interpretación aplicada. En el contexto de este ejercicio, la media representa la distancia promedio de los estudiantes al campus, mientras que el rango y la desviación estándar indican la heterogeneidad geográfica del grupo. Una desviación alta implica dispersión en la procedencia, y una baja, concentración en torno a una distancia media.

Definición y explicación de las funciones utilizadas

1. Funciones de conteo y estructura de datos.

- `contar_filas(vec)`: esta función toma un vector y recorre sus elementos haciendo uso de un bucle `for`. En cada iteración incrementa un contador inicializado a cero. El resultado final es el número total de elementos del vector. Se trata de un equivalente de la función `length()` de R, pero implementada manualmente. El mecanismo subyacente consiste en una lectura secuencial del vector y acumulación iterativa. El valor retornado permite validar la carga correcta del fichero y es utilizado posteriormente para el cálculo de frecuencias relativas. *Argumentos:* `vec` (vector de cualquier tipo). *Devuelve:* un entero con el número total de elementos.

```
> # Función para contar filas
> contar_filas <- function(vec) {
+   # Se inicializa el contador
+   n <- 0
+   # Se itera elemento a elemento y se incrementa el contador
+   for (._ in vec) n <- n + 1
+   # Se retorna el total contado
+   n
+ }
```

- `contar_columnas(df)`: de modo análogo a la función anterior, recorre la lista de nombres de las columnas (`names(df)`) y aplica el mismo proceso de conteo. Permite verificar la integridad estructural del conjunto de datos leído. *Argumentos:* `df` (`data.frame`). *Devuelve:* un entero con el número de columnas.

```
> # Función para contar columnas
> contar_columnas <- function(df) {
+   # Se inicializa el contador
+   c <- 0
+   # Se itera sobre los nombres de las columnas y se incrementa el contador
+   for (._ in names(df)) c <- c + 1
+   # Se retorna el total contado
+   c
+ }
```

- `contar_elementos(x)`: versión genérica aplicable a cualquier tipo de vector. Es un ejemplo de *abstracción algorítmica*, ya que encapsula el patrón de conteo iterativo en una única función reutilizable, sin depender del tipo de dato.

```
> # Función para contar cuántos elementos hay en un vector (sin length)
> contar_elementos <- function(x) {
+   # Se inicializa el contador
+   n <- 0
+   # Se incrementa por cada elemento del vector
+   for (._ in x) n <- n + 1
+   # Se retorna el total
+   n
+ }
```

2. Función de ordenación: `ordenar_indices(x, decreasing = FALSE).` Esta función reproduce el algoritmo clásico de *ordenamiento por burbuja*, donde se realizan comparaciones sucesivas entre pares adyacentes mediante estructuras de control anidadas. El vector de entrada `x` no se reordena directamente, sino que se construye un vector de *índices* que indica el orden en que los elementos deberían aparecer para estar ordenados. Durante el proceso, se comparan los pares adyacentes e intercambian los índices cuando la condición de orden no se cumple (`x[i] >x[i+1]` o la inversa si `decreasing = TRUE`). Tras varias pasadas, los elementos mayores "flotan" hacia el final del vector. *Argumentos:* `x` (vector numérico o comparable), `decreasing` (lógico, indica si el orden debe ser descendente). *Devuelve:* un vector de índices con el orden resultante.

```
> # Función para ordenar índices
> ordenar_indices <- function(x, decreasing = FALSE) {
+   # Se cuenta manualmente el número de elementos
+   n <- 0
+   for (._ in x) n <- n + 1
+   # Se crea un vector con los índices 1..n
+   idx <- 1:n
+   # Se aplica un algoritmo de ordenamiento (burbuja)
+   for (i in 1:(n - 1)) {
+     for (j in (i + 1):n) {
```

```
+       # Si el orden es ascendente se compara menor que, si es descendente mayor que
+       if ((!decreasing && x[j] < x[i]) || (decreasing && x[j] > x[i])) {
+         # Intercambiar elementos
+         tmp <- x[i]; x[i] <- x[j]; x[j] <- tmp
+         # Intercambiar índices
+         tmp2 <- idx[i]; idx[i] <- idx[j]; idx[j] <- tmp2
+       }
+     }
+   }
+   # Se retornan los índices ordenados
+   idx
+ }
```

3. Función de rango: `rango_manual(x)`. El rango se define como la diferencia entre el valor máximo y el mínimo del conjunto de datos. En lugar de usar las funciones `max()` y `min()`, esta función inicia dos variables (`max` y `min`) con el primer valor del vector y las actualiza secuencialmente si encuentra un valor mayor o menor. Al finalizar, devuelve la diferencia entre ambos valores, cuantificando la amplitud total de la variable. El rango es una medida de dispersión simple que representa la diferencia entre el estudiante más próximo y el más lejano a la universidad. *Argumentos:* `x` (vector numérico). *Devuelve:* un valor numérico correspondiente al rango.

```
> # Función para calcular el rango
> rango_manual <- function(x) {
+   # Inicializar máximo y mínimo con el primer valor
+   maximo <- x[1]
+   minimo <- x[1]
+   # Iterar sobre los valores para encontrar máximo y mínimo
+   for (valor in x) {
+     # Si el valor es mayor que el máximo actual, actualizar máximo
+     if (valor > maximo) maximo <- valor
+     # Si el valor es menor que el mínimo actual, actualizar mínimo
+     if (valor < minimo) minimo <- valor
+   }
+   # Retornar el rango
+   maximo - minimo
+ }
```

4. Obtención de valores únicos: `valores_unicos_manual(x)`. Esta función construye un nuevo vector `unicos` inicializado vacío y recorre todos los elementos de `x`. En cada iteración comprueba si el valor actual ya está dentro de `unicos`; si no lo está, se incorpora mediante concatenación. Al finalizar el bucle, se ordenan los valores resultantes haciendo uso de los índices obtenidos con la función `ordenar_indices()`. Este proceso combina una búsqueda secuencial (comparación elemento a elemento) con una inserción condicional, y es esencial para construir las tablas de frecuencias. *Argumentos:* `x` (vector numérico o de caracteres). *Devuelve:* un vector de valores únicos ordenados.

```
> # Función para obtener valores únicos
> valores_unicos_manual <- function(x) {
+   # En primer lugar, se crea un vector vacío para almacenar los valores únicos
+   unicos <- c()
+   # Se itera sobre cada valor
+   for (v in x) {
+     # Flag para indicar si el valor ya fue encontrado
+     encontrado <- FALSE
+     # Se verifica si el valor ya está en el vector de valores únicos
+     for (u in unicos) {
+       if (v == u) { encontrado <- TRUE; break }
+     }
+     # Si el valor no fue encontrado, se agrega al vector de valores únicos
```

```
+     if (!encontrado) unicos <- c(unicos, v)
+   }
+   # Se ordenan los valores únicos y se retornan
+   unicos[ordenar_indices(unicos)]
+ }
```

5. Frecuencias: conteo, acumulación y normalización.

- `frecuencia_absoluta_manual(x, valores_ordenados)`: recibe el vector original y el conjunto de valores únicos previamente ordenado. Para cada valor único, recorre nuevamente el vector `x` y aumenta el contador correspondiente cuando encuentra coincidencias. El resultado es un vector de igual longitud que `valores_ordenados`, donde cada posición representa cuántas veces aparece el valor correspondiente. La función utiliza una estructura de bucles anidados que refuerza la comprensión del proceso de clasificación en categorías. Reproduce el comportamiento de `table()` de R, pero implementado manualmente. *Argumentos:* `x` (vector de datos), `valores_ordenados` (vector de únicos). *Devuelve:* un vector numérico con las frecuencias absolutas.

  ```
  > # Función para calcular la frecuencia absoluta
  > frecuencia_absoluta_manual <- function(x, valores_ordenados) {
  +   # Se obtiene el número de valores únicos
  +   k <- contar_elementos(valores_ordenados)
  +   # Inicializar el vector de frecuencias absolutas
  +   frec <- numeric(k)
  +   # Se itera sobre cada valor observado
  +   for (v in x) {
  +     # Se compara con cada valor único
  +     for (i in 1:k) {
  +       # Si coincide, se incrementa la frecuencia absoluta correspondiente
  +       if (v == valores_ordenados[i]) { frec[i] <- frec[i] + 1; break }
  +     }
  +   }
  +   # Se retorna el vector de frecuencias
  +   frec
  + }
  ```

- `acumulada_manual(vec)`: calcula la suma progresiva de los elementos de un vector. Se inicializa un acumulador que, en cada iteración, añade el valor actual y lo almacena en un nuevo vector. Esta función ejemplifica un algoritmo iterativo que modela una suma acumulada sin funciones de orden superior. *Argumentos:* `vec` (vector numérico). *Devuelve:* un vector numérico de igual longitud con las sumas acumuladas.

  ```
  > # Función para calcular la frecuencia acumulada
  > acumulada_manual <- function(vec) {
  +   # Se cuenta cuántos elementos tiene el vector
  +   n <- contar_elementos(vec)
  +   # Inicializar el vector de frecuencias acumuladas
  +   out <- numeric(n)
  +   # Se suman progresivamente las frecuencias
  +   acum <- 0
  +   for (i in 1:n) { acum <- acum + vec[i]; out[i] <- acum }
  +   # Se retorna la acumulada
  +   out
  + }
  ```

- `relativa_manual(frec_abs, total)`: transforma las frecuencias absolutas en relativas dividiendo cada componente de `frec_abs` entre el número total de observaciones. La precisión de este paso depende de que `total` coincida con el número real de filas, garantizado mediante `contar_filas()`. El resultado permite interpretar los valores en términos porcentuales y facilita la construcción de histogramas o distribuciones empíricas. *Argumentos:* `frec_abs` (vector numérico), `total` (entero). *Devuelve:* un vector de frecuencias relativas en $[0, 1]$.

```
> # Función para calcular la frecuencia relativa manual
> relativa_manual <- function(frec_abs, total) {
+   # Se cuenta el tamaño del vector de frecuencias absolutas
+   n <- contar_elementos(frec_abs)
+   # Inicializar el vector de frecuencias relativas
+   out <- numeric(n)
+   # Se calcula dividiendo la frecuencia absoluta entre el total de filas
+   for (i in 1:n) out[i] <- frec_abs[i] / total
+   # Se retorna la frecuencia relativa
+   out
+ }
```

6. Medidas de tendencia central y dispersión.

- `media_manual(x)`: la media se obtiene sumando todos los elementos del vector y dividiendo por el número total de observaciones. Los valores se acumulan en un bucle `for`, asegurando que el acumulador se inicialice a cero para evitar errores. Este proceso muestra explícitamente cómo la media representa el centro de gravedad de la distribución de datos. *Argumentos:* `x` (vector numérico). *Devuelve:* la media aritmética.

```
> # Función para calcular la media
> media_manual <- function(x) {
+   # Se incializa la suma a cero
+   suma <- 0
+   # Se cuenta el número de elementos
+   n <- contar_elementos(x)
+   # Se suma cada valor
+   for (v in x) suma <- suma + v
+   # Se retorna la media
+   suma / n
+ }
```

- `varianza_poblacional_manual(x)` y `varianza_muestral_manual(x)`: ambas funciones siguen la definición formal de la varianza. Primero se calcula la media con `media_manual(x)`, luego se recorre el vector calculando la desviación cuadrática de cada elemento respecto a la media, y se acumula el resultado. La versión muestral aplica la corrección de Bessel ($N-1$) para eliminar el sesgo en la estimación de la varianza poblacional. La comparación entre ambas versiones permite apreciar cómo la corrección de Bessel evita la subestimación sistemática de la varianza cuando solo se dispone de una muestra. *Argumentos:* `x` (vector numérico). *Devuelve:* un valor numérico (varianza poblacional o muestral).

```
> # Función para calcular la varianza poblacional
> varianza_poblacional_manual <- function(x) {
+   # Se cuenta el número de elementos
+   n <- contar_elementos(x)
+   # Se calcula la media manualmente
+   m <- media_manual(x)
+   # Se calcula la suma de los cuadrados de las diferencias respecto a la media
+   suma_cuad <- 0
+   for (v in x) suma_cuad <- suma_cuad + (v - m)^2
+   # Varianza poblacional: dividir entre N
+   suma_cuad / n
+ }
> # Función para calcular la varianza muestral
> varianza_muestral_manual <- function(x) {
+   # Se cuenta el número de elementos
+   n <- contar_elementos(x)
```

```
+   # Si no hay suficientes datos, se retorna NA
+   if (n <= 1) return(NA_real_)
+   # Se calcula la media manualmente
+   m <- media_manual(x)
+   # Se calcula la suma de los cuadrados de las diferencias respecto a la media
+   suma_cuad <- 0
+   for (v in x) suma_cuad <- suma_cuad + (v - m)^2
+   # Varianza muestral: dividir entre (N - 1)
+   suma_cuad / (n - 1)
+ }
```

- `desv_estandar(varianza)`: toma la raíz cuadrada de la varianza mediante el operador `sqrt()`, devolviendo una medida de dispersión expresada en las mismas unidades que la variable original (kilómetros). Esta equivalencia de unidades facilita la interpretación práctica, ya que permite afirmar, por ejemplo, que "la distancia media difiere en aproximadamente 20 km respecto al promedio". *Argumentos:* `varianza` (valor numérico). *Devuelve:* la desviación estándar.

```
> # Función para obtener la desviación estándar a partir de una varianza
> desv_estandar <- function(varianza) {
+   # Si la varianza es NA, se retorna NA
+   if (is.na(varianza)) return(NA_real_)
+   # Se calcula la raíz cuadrada de la varianza
+   sqrt(varianza)
+ }
```

7. Integración en el flujo principal. Las funciones anteriores se integran secuencialmente en el flujo de trabajo del ejercicio:

1. Carga del fichero `distancias.txt` en el `data.frame` `s`.
2. Verificación estructural mediante las funciones de conteo.
3. Ordenación de observaciones y creación de los data frames `so_asc` y `so_desc`.
4. Cálculo del rango, valores únicos y tabla de frecuencias.
5. Derivación de medidas de resumen (media, varianzas y desviación estándar).

Cada función ha sido implementada con independencia modular, lo que facilita su verificación individual, su reutilización en otros análisis y la trazabilidad completa del proceso estadístico.

Ejecución del script principal

En este bloque se ejecuta el flujo completo del ejercicio, integrando todas las funciones definidas anteriormente. El objetivo es validar su correcto funcionamiento y mostrar los resultados del análisis descriptivo sobre el conjunto de datos `distancias.txt`.

```
> # ============================
> # Script principal
> # ============================
>
> # Se lee el archivo de datos
> s <- read.table("distancias.txt", header = TRUE)
> print(s)
```

	nombre	distancia
1	Villalbilla	16.5
2	Ensanche_de_Vallecas	34.8
3	Villalbilla	20.7
4	Alcalá_de_Henares	6.2
5	Alcalá_de_Henares	4.4
6	Alcalá_de_Henares	3.4
7	Cifuentes	24.0
8	Cifuentes	24.0
9	El_Casar	32.0
10	Fuente_el_Saz_del_Jarama	30.0
11	ND	33.0
12	Coslada	27.0
13	Daganzo_de_Arriba	15.0
14	Alcalá_de_Henares	9.4
15	Alcalá_de_Henares	2.1
16	Arganda_del_Rey	34.0
17	Coslada	24.0
18	Alcalá_de_Henares	12.0
19	Alcalá_de_Henares	4.4
20	Madrid	28.0
21	Madrid	31.4
22	Mejorada_del_Campo	21.6
23	Alcalá_de_Henares	3.1
24	Alcalá_de_Henares	4.5
25	Alcalá_de_Henares	5.1
26	ND	4.0
27	ND	3.2
28	ND	25.0
29	Alcalá_de_Henares	4.5
30	Guadalajara	20.0
31	Guadalajara	34.0
32	Torrejón_de_Ardoz	12.0
33	Torrejón_de_Ardoz	12.0
34	Torrejón_de_Ardoz	12.0
35	Torrejón_de_Ardoz	12.0
36	Alcalá_de_Henares	5.0
37	Torrejón_de_Ardoz	19.0
38	Guadalajara	30.0
39	ND	5.5
40	ND	38.0
41	Chiloheches	25.0
42	Alcalá_de_Henares	3.7
43	ND	9.0
44	ND	30.0
45	ND	13.0
46	Guadalajara	30.0
47	Guadalajara	30.0
48	Coslada	26.0
49	Guadalajara	30.0
50	Cabanillas_del_Campo	30.0
51	Alcalá_de_Henares	1.0
52	Madrid	26.0
53	ND	22.0
54	ND	10.0
55	Daganzo	9.7
56	Alcalá_de_Henares	11.0
57	Torres_de_la_Alameda	24.1

```
58   Velilla_de_San_Antonio      33.0
59                  Daganzo      17.2
60              Guadalajara      27.0
61                       ND      24.0
62              Guadalajara      27.0
63                   Cobeña      21.0
64                 Galapagos     28.0
65                    Madrid     30.0
66         Alcalá_de_Henares      4.0
67                    Madrid     46.0
68                  Coslada      29.0
69                       ND      3.7
70         Alcalá_de_Henares      2.7
71                       ND      8.1
72                  Alovera      19.0
73         Torrejón_de_Ardoz     16.0
```

```
> # Contar filas y columnas manualmente
> filas <- contar_filas(s$distancia)
> columnas <- contar_columnas(s)
> cat("Dimensiones (filas x columnas): ", filas, " x ", columnas, "\n", sep = "")
```

```
Dimensiones (filas x columnas): 73 x 2
```

```
> # Ordenación ascendente y descendente
> idx_asc  <- ordenar_indices(s$distancia)
> idx_desc <- ordenar_indices(s$distancia, decreasing = TRUE)
> so_asc  <- s[idx_asc, ]
> so_desc <- s[idx_desc, ]
> cat("Orden ascendente por distancia:\n"); print(so_asc)
```

```
Orden ascendente por distancia:
```

```
                    nombre distancia
51         Alcalá_de_Henares      1.0
15         Alcalá_de_Henares      2.1
70         Alcalá_de_Henares      2.7
23         Alcalá_de_Henares      3.1
27                        ND      3.2
6          Alcalá_de_Henares      3.4
69                        ND      3.7
42         Alcalá_de_Henares      3.7
26                        ND      4.0
66         Alcalá_de_Henares      4.0
5          Alcalá_de_Henares      4.4
19         Alcalá_de_Henares      4.4
29         Alcalá_de_Henares      4.5
24         Alcalá_de_Henares      4.5
36         Alcalá_de_Henares      5.0
25         Alcalá_de_Henares      5.1
39                        ND      5.5
4          Alcalá_de_Henares      6.2
71                        ND      8.1
43                        ND      9.0
14         Alcalá_de_Henares      9.4
```

```
55                  Daganzo       9.7
54                       ND      10.0
56        Alcalá_de_Henares      11.0
33        Torrejón_de_Ardoz      12.0
34        Torrejón_de_Ardoz      12.0
35        Torrejón_de_Ardoz      12.0
18        Alcalá_de_Henares      12.0
32        Torrejón_de_Ardoz      12.0
45                       ND      13.0
13        Daganzo_de_Arriba      15.0
73        Torrejón_de_Ardoz      16.0
1               Villalbilla      16.5
59                  Daganzo      17.2
72                  Alovera      19.0
37        Torrejón_de_Ardoz      19.0
30              Guadalajara      20.0
3               Villalbilla      20.7
63                   Cobeña      21.0
22       Mejorada_del_Campo      21.6
53                       ND      22.0
61                       ND      24.0
17                  Coslada      24.0
8                 Cifuentes      24.0
7                 Cifuentes      24.0
57     Torres_de_la_Alameda      24.1
28                       ND      25.0
41              Chiloheches      25.0
48                  Coslada      26.0
52                   Madrid      26.0
12                  Coslada      27.0
62              Guadalajara      27.0
60              Guadalajara      27.0
20                   Madrid      28.0
64                Galapagos      28.0
68                  Coslada      29.0
65                   Madrid      30.0
50     Cabanillas_del_Campo      30.0
44                       ND      30.0
10 Fuente_el_Saz_del_Jarama      30.0
46              Guadalajara      30.0
47              Guadalajara      30.0
38              Guadalajara      30.0
49              Guadalajara      30.0
21                   Madrid      31.4
9                   El_Casar      32.0
11                       ND      33.0
58   Velilla_de_San_Antonio      33.0
16          Arganda_del_Rey      34.0
31              Guadalajara      34.0
2      Ensanche_de_Vallecas      34.8
40                       ND      38.0
67                  Madrid      46.0
```

```
> cat("\nOrden descendente por distancia:\n"); print(so_desc); cat("\n")
```

```
Orden descendente por distancia:
```

	nombre	distancia
67	Madrid	46.0
40	ND	38.0
2	Ensanche_de_Vallecas	34.8
16	Arganda_del_Rey	34.0
31	Guadalajara	34.0
58	Velilla_de_San_Antonio	33.0
11	ND	33.0
9	El_Casar	32.0
21	Madrid	31.4
44	ND	30.0
46	Guadalajara	30.0
47	Guadalajara	30.0
49	Guadalajara	30.0
50	Cabanillas_del_Campo	30.0
38	Guadalajara	30.0
65	Madrid	30.0
10	Fuente_el_Saz_del_Jarama	30.0
68	Coslada	29.0
64	Galapagos	28.0
20	Madrid	28.0
60	Guadalajara	27.0
62	Guadalajara	27.0
12	Coslada	27.0
52	Madrid	26.0
48	Coslada	26.0
28	ND	25.0
41	Chiloheches	25.0
57	Torres_de_la_Alameda	24.1
8	Cifuentes	24.0
7	Cifuentes	24.0
61	ND	24.0
17	Coslada	24.0
53	ND	22.0
22	Mejorada_del_Campo	21.6
63	Cobeña	21.0
3	Villalbilla	20.7
30	Guadalajara	20.0
37	Torrejón_de_Ardoz	19.0
72	Alovera	19.0
59	Daganzo	17.2
1	Villalbilla	16.5
73	Torrejón_de_Ardoz	16.0
13	Daganzo_de_Arriba	15.0
45	ND	13.0
32	Torrejón_de_Ardoz	12.0
33	Torrejón_de_Ardoz	12.0
34	Torrejón_de_Ardoz	12.0
35	Torrejón_de_Ardoz	12.0
18	Alcalá_de_Henares	12.0
56	Alcalá_de_Henares	11.0
54	ND	10.0
55	Daganzo	9.7
14	Alcalá_de_Henares	9.4
43	ND	9.0
71	ND	8.1
4	Alcalá_de_Henares	6.2
39	ND	5.5

```
25        Alcalá_de_Henares       5.1
36        Alcalá_de_Henares       5.0
29        Alcalá_de_Henares       4.5
24        Alcalá_de_Henares       4.5
5         Alcalá_de_Henares       4.4
19        Alcalá_de_Henares       4.4
66        Alcalá_de_Henares       4.0
26                       ND       4.0
69                       ND       3.7
42        Alcalá_de_Henares       3.7
6         Alcalá_de_Henares       3.4
27                       ND       3.2
23        Alcalá_de_Henares       3.1
70        Alcalá_de_Henares       2.7
15        Alcalá_de_Henares       2.1
51        Alcalá_de_Henares       1.0
```

```
> # Rango
> rangor <- rango_manual(s$distancia)
> cat("Rango (max - min):", rangor, "\n\n")
```

```
Rango (max - min): 45
```

```
> # Valores únicos y frecuencias
> valores_unicos <- valores_unicos_manual(s$distancia)
> n_valores <- contar_elementos(valores_unicos)
> frecuencia_abs <- frecuencia_absoluta_manual(s$distancia, valores_unicos)
> frecuencia_acum <- acumulada_manual(frecuencia_abs)
> frecuencia_rel <- relativa_manual(frecuencia_abs, filas)
> frecuencia_rel_acum <- acumulada_manual(frecuencia_rel)
> # Mostrar frecuencias
> cat("Frecuencia absoluta:\n")
```

```
Frecuencia absoluta:
```

```
> for (i in 1:n_valores) cat(valores_unicos[i], ":", frecuencia_abs[i], "\n")
```

```
1 : 1
2.1 : 1
2.7 : 1
3.1 : 1
3.2 : 1
3.4 : 1
3.7 : 2
4 : 2
4.4 : 2
4.5 : 2
5 : 1
5.1 : 1
5.5 : 1
6.2 : 1
```

```
8.1 : 1
9 : 1
9.4 : 1
9.7 : 1
10 : 1
11 : 1
12 : 5
13 : 1
15 : 1
16 : 1
16.5 : 1
17.2 : 1
19 : 2
20 : 1
20.7 : 1
21 : 1
21.6 : 1
22 : 1
24 : 4
24.1 : 1
25 : 2
26 : 2
27 : 3
28 : 2
29 : 1
30 : 8
31.4 : 1
32 : 1
33 : 2
34 : 2
34.8 : 1
38 : 1
46 : 1
```

```
> cat("\nFrecuencia acumulada:\n")
```

```
Frecuencia acumulada:
```

```
> for (i in 1:n_valores) cat(valores_unicos[i], ":", frecuencia_acum[i], "\n")
```

```
1 : 1
2.1 : 2
2.7 : 3
3.1 : 4
3.2 : 5
3.4 : 6
3.7 : 8
4 : 10
4.4 : 12
4.5 : 14
5 : 15
5.1 : 16
5.5 : 17
6.2 : 18
8.1 : 19
9 : 20
```

```
9.4 : 21
9.7 : 22
10 : 23
11 : 24
12 : 29
13 : 30
15 : 31
16 : 32
16.5 : 33
17.2 : 34
19 : 36
20 : 37
20.7 : 38
21 : 39
21.6 : 40
22 : 41
24 : 45
24.1 : 46
25 : 48
26 : 50
27 : 53
28 : 55
29 : 56
30 : 64
31.4 : 65
32 : 66
33 : 68
34 : 70
34.8 : 71
38 : 72
46 : 73
```

```
> cat("\nFrecuencia relativa:\n")
```

```
Frecuencia relativa:
```

```
> for (i in 1:n_valores) cat(valores_unicos[i], ":", frecuencia_rel[i], "\n")
```

```
1 : 0.01369863
2.1 : 0.01369863
2.7 : 0.01369863
3.1 : 0.01369863
3.2 : 0.01369863
3.4 : 0.01369863
3.7 : 0.02739726
4 : 0.02739726
4.4 : 0.02739726
4.5 : 0.02739726
5 : 0.01369863
5.1 : 0.01369863
5.5 : 0.01369863
6.2 : 0.01369863
8.1 : 0.01369863
9 : 0.01369863
9.4 : 0.01369863
9.7 : 0.01369863
```

```
10 : 0.01369863
11 : 0.01369863
12 : 0.06849315
13 : 0.01369863
15 : 0.01369863
16 : 0.01369863
16.5 : 0.01369863
17.2 : 0.01369863
19 : 0.02739726
20 : 0.01369863
20.7 : 0.01369863
21 : 0.01369863
21.6 : 0.01369863
22 : 0.01369863
24 : 0.05479452
24.1 : 0.01369863
25 : 0.02739726
26 : 0.02739726
27 : 0.04109589
28 : 0.02739726
29 : 0.01369863
30 : 0.109589
31.4 : 0.01369863
32 : 0.01369863
33 : 0.02739726
34 : 0.02739726
34.8 : 0.01369863
38 : 0.01369863
46 : 0.01369863
```

```
> cat("\nFrecuencia relativa acumulada:\n")
```

```
Frecuencia relativa acumulada:
```

```
> for (i in 1:n_valores) cat(valores_unicos[i], ":", frecuencia_rel_acum[i], "\n")
```

```
1 : 0.01369863
2.1 : 0.02739726
2.7 : 0.04109589
3.1 : 0.05479452
3.2 : 0.06849315
3.4 : 0.08219178
3.7 : 0.109589
4 : 0.1369863
4.4 : 0.1643836
4.5 : 0.1917808
5 : 0.2054795
5.1 : 0.2191781
5.5 : 0.2328767
6.2 : 0.2465753
8.1 : 0.260274
9 : 0.2739726
9.4 : 0.2876712
9.7 : 0.3013699
10 : 0.3150685
11 : 0.3287671
```

```
12 : 0.3972603
13 : 0.4109589
15 : 0.4246575
16 : 0.4383562
16.5 : 0.4520548
17.2 : 0.4657534
19 : 0.4931507
20 : 0.5068493
20.7 : 0.5205479
21 : 0.5342466
21.6 : 0.5479452
22 : 0.5616438
24 : 0.6164384
24.1 : 0.630137
25 : 0.6575342
26 : 0.6849315
27 : 0.7260274
28 : 0.7534247
29 : 0.7671233
30 : 0.8767123
31.4 : 0.890411
32 : 0.9041096
33 : 0.9315068
34 : 0.9589041
34.8 : 0.9726027
38 : 0.9863014
46 : 1
```

```
> # Medidas descriptivas
> media <- media_manual(s$distancia)
> varianza_poblacional <- varianza_poblacional_manual(s$distancia)
> varianza_muestral <- varianza_muestral_manual(s$distancia)
> desv_est_poblacional <- desv_estandar(varianza_poblacional)
> desv_est_muestral <- desv_estandar(varianza_muestral)
> cat("\nMedia:", media, "\n")
```

```
Media: 18.53425
```

```
> cat("Varianza poblacional:", varianza_poblacional, "\n")
```

```
Varianza poblacional: 126.1587
```

```
> cat("Desviación estándar poblacional:", desv_est_poblacional, "\n")
```

```
Desviación estándar poblacional: 11.23204
```

```
> cat("Varianza muestral:", varianza_muestral, "\n")
```

```
Varianza muestral: 127.9109
```

```
> cat("Desviación estándar muestral:", desv_est_muestral, "\n")
```

```
Desviación estándar muestral: 11.30977
```

AI-assisted development

The following prompts were used during the development of this exercise to refine the logic of manual implementations, the documentation of each function, and the integration within the `Sweave` environment. All resulting code was subsequently reviewed, tested, and adapted.

- **Prompt 1:** *"How can I implement a manual version of the `length()` function in R using a for loop?"* Guided the creation of `contar_filas()`, `contar_columnas()`, and `contar_elementos()`, each based on iterative counting.
- **Prompt 2:** *"Show me how to sort numeric data manually in R using nested loops (bubble sort) and return only the index order."* Used to design `ordenar_indices()`, which reproduces a bubble sort algorithm returning index positions.
- **Prompt 3:** *"Write an R function that calculates the range of a numeric vector without using `max()` or `min()`."* Inspired the implementation of `rango_manual()` through iterative comparison of values.
- **Prompt 4:** *"How can I manually find unique values in a numeric vector in R, without using `unique()`?"* Helped define `valores_unicos_manual()`, combining sequential search and conditional appending.
- **Prompt 5:** *."Explain how to calculate absolute, cumulative, and relative frequencies manually in R using loops."* Used to design `frecuencia_absoluta_manual()`, `acumulada_manual()`, and `relativa_manual()`, which reproduce frequency analysis logic step by step.
- **Prompt 6:** *"How can I organize multiple R functions with explanations inside a Sweave (.Rnw) document so that each function is displayed with comments but executed only once?"* Clarified the correct use of chunk options (`eval=TRUE`, `echo=TRUE`, `results=hide`) for reproducible reports.
- **Prompt 7:** *"How can I integrate all the defined functions into a final executable section that prints descriptive statistics and frequency tables?"* Used to design the final execution block combining all steps of the descriptive analysis.
- **Prompt 8:** *"How do I fix a Sweave error saying 'results=markup not recognized' when executing a chunk?"* Guided the correction by replacing `results=markup` with `results=verbatim` for Sweave compatibility.

2.6. Ejercicio 2.2: Fases del Algoritmo Apriori

El algoritmo se ha dividido en 7 fases lógicas, que se detallan a continuación.

Descripción del conjunto de datos

El conjunto de datos utilizado es una muestra transaccional de 8 sucesos (compras o configuraciones de vehículos). Se analizan 6 posibles "ítems" o sucesos elementales: "Faros de Xenon", "Control de Velocidad", "Navegador", "Bluetooth", "Techo Solar" y "Alarma". Los datos se representan en una matriz de 8x6, donde un '1' indica que el suceso (fila) contiene ese item (columna) y '0' que no lo contiene. El objetivo es aplicar el algoritmo Apriori para encontrar reglas de asociación (ej. "si compra Faros de Xenon, también compra Bluetooth") que cumplan ciertos umbrales de soporte y confianza.

2.6.1. Fase 1: Carga de Datos y Sucesos Elementales Candidatos

Fundamento Teórico

El primer paso del algoritmo Apriori es identificar los "items frecuentes" de tamaño 1 (L1). Esto se hace escaneando la base de datos transaccional para contar la frecuencia de cada item individual. El **soporte** de un item se calcula como el número de transacciones que contienen ese item, dividido por el número total de transacciones.

$$\text{Soporte}(I) = \frac{\text{Nº de transacciones que contienen } I}{\text{Nº total de transacciones}}$$

Solo aquellos items cuyo soporte sea mayor o igual a un umbral de soporte mínimo (en este caso, 0.5 o 50 %) se consideran "frecuentes" y pasan a la siguiente fase. Los items que no alcanzan este umbral se descartan, ya que cualquier conjunto de items más grande que los contenga tampoco podrá ser frecuente (este es el principio fundamental de Apriori).

Explicación de las funciones utilizadas

- `library(Matrix)` y `library(arules)`: Cargan las bibliotecas necesarias. `Matrix` para matrices dispersas (eficientes con muchos ceros) y `arules` para minería de reglas de asociación.
- `Matrix(...)`: Crea un objeto de tipo `Matrix`. Con `sparse=TRUE`, se almacena de forma eficiente. `byrow=TRUE` indica que los datos se leen por filas.
- `as(x, 'clase')`: Convierte un objeto `x` al tipo `'clase'`. Se usa para transformar la matriz en un formato `nsparseMatrix` (matriz dispersa binaria) y luego en `transactions`, que es el formato que requiere `arules`.
- `t(x)`: Transpone la matriz. Es un paso necesario porque `arules` espera que los items estén en filas y las transacciones en columnas al convertir desde una matriz.
- `summary(transacciones)`: Muestra un resumen de los datos transaccionales, incluyendo el número de transacciones, items, y la densidad.
- `itemFrequency(x, type='absolute')`: Calcula la frecuencia de cada item. Con `type='absolute'`, devuelve el conteo (ej. 5), no la proporción.
- `length(transacciones)`: Devuelve el número de transacciones (sucesos) en el conjunto de datos, que es 8.

Explicación de la Fase

[En esta primera fase el objetivo es ver que sucesos elementales de la muestra superan o igualan el umbral de soporte. Para ello se tiene que calcular el n^o de veces que aparece dicho suceso elemental en la muestra, dividido por el núemro de elemento de la muestra. Los sucesos elementales que superen o igualen el umbral, son los que en la siguiente fase se utilizarán para crear los sucesos candidatos de las diferentes dimensiones. Antes de llevar a cabo estos pasos, se debe cargar la muestra.]

Código y Ejecución (Sweave)

```
> # 1ª FASE ----------------------------------------------------
> library(Matrix)
> library(arules)
> muestra<-Matrix(c(1,1,1,1,0,0, 1,1,0,1,1,0, 1,1,1,0,0,0, 1,0,1,1,1,0,
+                   1,1,0,1,0,0, 0,0,1,0,0,0, 1,1,0,1,0,0, 0,0,0,0,1,1),
+                   8, 6, byrow=TRUE,
+                   dimnames = list(
+                     c("suceso1", "suceso2", "suceso3", "suceso4",
+                       "suceso5", "suceso6", "suceso7", "suceso8"),
+                     c("Faros de Xenon", "Control de Velocidad", "Navegador",
+                       "Bluetooth", "Techo Solar", "Alarma")
+                   ), sparse=TRUE)
> muestrangCMatrix<-as(muestra, "nsparseMatrix")
> trapmuestrangCMatrix<-t(muestrangCMatrix)
> transacciones<-as(trapmuestrangCMatrix, "transactions")
> summary(transacciones)


transactions as itemMatrix in sparse format with
 8 rows (elements/itemsets/transactions) and
 6 columns (items) and a density of 0.5

most frequent items:
      Faros de Xenon Control de Velocidad            Bluetooth            Navegador          Techo So
                   6                    5                    5                    4

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

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    2.75    3.00    3.00    4.00    4.00

includes extended item information - examples:
               labels
1       Faros de Xenon
2 Control de Velocidad
3            Navegador

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

```
> frequency <- itemFrequency(transacciones, type ="absolute")
> print(data.frame(Frecuencia = frequency))
```

```
                     Frecuencia
Faros de Xenon                6
Control de Velocidad          5
Navegador                     4
Bluetooth                     5
Techo Solar                   3
Alarma                        1
```

```
> support_threshold <- 0.5
> numero_sucesos <-length(transacciones)
> support_sucesos <-frequency/numero_sucesos
> print(data.frame(Soporte = support_sucesos))
```

```
                     Soporte
Faros de Xenon         0.750
Control de Velocidad   0.625
Navegador              0.500
Bluetooth              0.625
Techo Solar            0.375
Alarma                 0.125
```

```
> elementos_S_validos <- support_sucesos[support_sucesos >= support_threshold]
> print(data.frame(Soporte_Valido = elementos_S_validos))
```

```
                     Soporte_Valido
Faros de Xenon                0.750
Control de Velocidad          0.625
Navegador                     0.500
Bluetooth                     0.625
```

Prompt de IA Utilizado

> [From the image i have just sent you i need to use a support threshold of 50to decide which of the elements ("Faros de Xenon", "Alarma" and so on) go through and are going to be consider in the next step to start creating the candidates subsets. So in the image i have just sent you it is clear that i an identify the number of times each element appears, so how can i go through them all dividing the number of times they appear and dividing it by the amount of "sucesos" that there are (there are 8) so that i get the percentages of the times they appear for a subset of 8 elements and can discriminate which elements surpass the 0.5 threshold]

2.6.2. Fase 2: Generación de Sucesos Candidatos (L2, L3, L4)

Fundamento Teórico

Esta fase implementa la **generación de candidatos** (Apriori-gen). Para generar candidatos de tamaño k (Lk), se utiliza el conjunto de itemsets frecuentes de tamaño $k-1$ (L(k-1)). El método estándar (join step) consiste en unir dos itemsets frecuentes A y B de L(k-1) si comparten los primeros $k-2$ items. Por ejemplo, para generar L3 (candidatos de tamaño 3) a partir de L2, se unen A, B y A, C para formar A, B,

C. El algoritmo Apriori también incluye un "pruning step" (poda) donde se elimina cualquier candidato de Lk que contenga un subconjunto de tamaño $k-1$ que no esté en L(k-1). Este código implementa una versión simplificada del "join step" al combinar todos los pares posibles y asegurarse de que el tamaño resultante sea k.

Explicación de las funciones utilizadas

- `lapply(X, FUN)`: Aplica la función `FUN` a cada elemento de la lista o vector `X`. Se usa para convertir la lista de items L1 en una lista de vectores (ej. `'ItemA' ->c('ItemA')`).
- `generate_candidates(L_prev)`: Es una función personalizada para generar la siguiente generación de candidatos.
- `length(x)`: Devuelve el número de elementos en la lista `L_prev` (cuántos itemsets) o en el itemset `L_prev[[1]]` (el tamaño $k-1$).
- `seq_len(n)`: Genera una secuencia de 1 a `n`. Se usa para los bucles `for`.
- `all(a[1:(k-2)] == b[1:(k-2)])`: Comprueba si los primeros $k-2$ elementos de los itemsets `a` y `b` son idénticos. Es la condición de unión (join).
- `sort(unique(c(a, b)))`: Combina los dos itemsets `a` y `b`, elimina duplicados (por si acaso) y los ordena. La ordenación es crucial para que A, B sea tratado igual que B, A.
- `append(list, new_element)`: Añade un nuevo elemento (el nuevo candidato) a la lista de candidatos.
- `unique(lapply(...))`: Se usa al final para eliminar candidatos duplicados que puedan haberse generado, convirtiéndolos primero a texto con `paste()` y luego de vuelta a listas con `strsplit()`.

Explicación de la Fase

[En esta segunda fase, se van a construir los sucesos candidatos para cada una de las dimensiones necesarias en función del número de sucesos candidatos elementales que superan el umbral de soporte en la primera fase. En este caso va a ser necesario crear sucesos candidatos hasta la cuarta dimensión. Para crear estos sucesos candidatos de dimensión K, se deben utilizar sucesos candidatos de la dimensión K-1 donde los últimos elementos de ambos sean diferentes y el resto de elemetos de ambos sean iguales de forma que no haya conflicto a la hora de combinar ambos sucesos].

Código y Ejecución (Sweave)

```
> # 2ª FASE ----------------------------------------------------
> L1 <- c("Faros de Xenon", "Control de Velocidad", "Navegador", "Bluetooth")
> L1_lista <- lapply(L1, function(x) c(x))
> generate_candidates <- function(L_prev) {
+     n <- length(L_prev)
+     if (n <= 1) return(list())
+
+     k <- length(L_prev[[1]]) + 1
+
+     # make sure items inside each subset are sorted
+     L_prev <- lapply(L_prev, sort)
+     candidates <- list()
+
+     for (i in seq_len(n - 1)) {
+         for (j in seq(i + 1, n)) {
```

```
+             a <- L_prev[[i]]
+             b <- L_prev[[j]]
+
+
+             if (k - 2 == 0 || all(a[1:(k - 2)] == b[1:(k - 2)])) {
+                 new_set <- sort(unique(c(a, b)))
+                 if (length(new_set) == k) {
+                     candidates <- append(candidates, list(new_set))
+                 }
+             }
+         }
+     }
+
+     # eliminar duplicados (si hay)
+     if (length(candidates) > 0) {
+         candidates <- unique(lapply(candidates,
+                                     function(x) paste(sort(x), collapse = ",")))
+         candidates <- lapply(candidates,
+                                      function(x) strsplit(x, ",")[[1]])
+     }
+     return(candidates)
+ }
> L2 <- generate_candidates(L1_lista)
> print("Candidatos L2:")


[1] "Candidatos L2:"


> print(L2)


[[1]]
[1] "Control de Velocidad" "Faros de Xenon"

[[2]]
[1] "Faros de Xenon" "Navegador"

[[3]]
[1] "Bluetooth"      "Faros de Xenon"

[[4]]
[1] "Control de Velocidad" "Navegador"

[[5]]
[1] "Bluetooth"            "Control de Velocidad"

[[6]]
[1] "Bluetooth" "Navegador"


> L3 <- generate_candidates(L2)
> print("Candidatos L3:")


[1] "Candidatos L3:"


> print(L3)
```

```
[[1]]
[1] "Control de Velocidad" "Faros de Xenon"       "Navegador"

[[2]]
[1] "Bluetooth"            "Control de Velocidad" "Faros de Xenon"

[[3]]
[1] "Bluetooth"      "Faros de Xenon" "Navegador"

[[4]]
[1] "Bluetooth"            "Control de Velocidad" "Navegador"


> L4 <- generate_candidates(L3)
> print("Candidatos L4:")


[1] "Candidatos L4:"


> print(L4)


[[1]]
[1] "Bluetooth"            "Control de Velocidad" "Faros de Xenon"       "Navegador"
```

Prompt de IA Utilizado

> [Okey great lets continue on to the next step. In this next step i need you get the candidate subsets that are going to form up to 4 dimmensions (because 4 elements passed the threshold).
>
> In case you do not know how this candidates subsets are formed, you need to start with the lowest dimmension possible other that K=1 and so for this dimmension and the follwing you need to have subsets in which the last element of the subset are different and the previous ones are exactly the same in order to combine them.
>
> For example, for the dimmension k=2 we need to combine the subsets that passed the threshold in a way that the new subsets that are formed with to subsets of the k=1 dimmension have the last element different and the rest are the same. In this dimmension because there are combinations of subsets of 1 elements you basically need to have all possible combinations of the k=1 subsets .
>
> But for the k=3 and so on you need to combine sets of k=2 o k=k-1 respectively following the rules I told you so for example if we have X,C and X,N which are from the k=2 dimmension then we can combine them because it follows the rules i mentioned obtaining anew subset of X,C,N for the dimmension k=3.
>
> And for yout information the elements that passed the threshold were Faros de Xenon, Control de Velocidad, Navegador, Bluetooth.]

2.6.3. Fase 3: Hash-Tree de sucesos candidatos para cada dimensión

Fundamento Teórico

Para hacer el conteo de soportes (Fase 5) de manera eficiente, el algoritmo Apriori (y otras implementaciones) utiliza estructuras de datos avanzadas como los **Hash Trees** (árboles hash). El objetivo es evitar tener que comparar cada candidato con cada transacción (lo cual sería muy lento). Un Hash Tree almacena los itemsets candidatos. Cada nodo interno es una tabla hash, y los nodos hoja almacenan los propios candidatos. Para contar, se toman los itemsets de una transacción y se "filtran" a través del

árbol. Si un itemset de la transacción "llega" a un nodo hoja, significa que coincide con un candidato, y se incrementa el contador de ese candidato. Para que este árbol funcione, los items deben tener un orden canónico. Por eso, en esta fase, los nombres ("Faros de Xenon") se traducen a números (1) y los itemsets se ordenan numéricamente (ej. 4, 1, 3 siempre se convierte en 1, 3, 4).

Explicación de las funciones utilizadas

- `c('Nombre' = 1, ...)`: Crea un vector nombrado. Esto es un "mapa" o "diccionario" muy eficiente en R. Permite buscar un nombre (ej. `item_map['Navegador']`) y obtener su número (3).
- `encode_candidates(candidates, mapping)`: Función personalizada para aplicar la codificación numérica.
- `lapply(candidates, function(subset) ...)`: Itera sobre cada itemset candidato (cada `subset`) de la lista `candidates`.
- `unname(mapping[subset])`: Toma el `subset` (ej. `c('Faros de Xenon', 'Navegador')`), lo usa para indexar el mapa (lo que devuelve `c(1, 3)` pero con los nombres `'Faros de Xenon', 'Navegador'`) y `unname()` elimina esos nombres, dejando un vector numérico limpio `c(1, 3)`.
- `sort(nums)`: Ordena el vector numérico resultante (ej. de `c(3, 1)` a `c(1, 3)`), asegurando el orden canónico necesario para el Hash Tree.

Explicación de la Fase

[En esta fase se van a crear los hash-trees de los sucesos candidatos. Para llevar a cabo este paso se debe llevar a cabo una transformación numérica de los sucesos candidatos. Una vez se tenga hecho, se podrá crear un hash tree por cada dimensión (L2, L3, L4)]

Código y Ejecución (Sweave)

```
> # 3º FASE ----------------------------------------------------
> item_map <- c(
+     "Faros de Xenon" = 1,
+     "Control de Velocidad" = 2,
+     "Navegador" = 3,
+     "Bluetooth" = 4,
+     "Techo Solar" = 5,
+     "Alarma" = 6
+ )
> encode_candidates <- function(candidates, mapping) {
+     lapply(candidates, function(subset) {
+         nums <- unname(mapping[subset])
+         sort(nums)
+     })
+ }
> L2_numerico <- encode_candidates(L2, item_map)
> L3_numerico <- encode_candidates(L3, item_map)
> L4_numerico <- encode_candidates(L4, item_map)
> print("Candidatos L2 Numéricos:")

[1] "Candidatos L2 Numéricos:"

> print(L2_numerico)
```

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

[[2]]
[1] 1 3

[[3]]
[1] 1 4

[[4]]
[1] 2 3

[[5]]
[1] 2 4

[[6]]
[1] 3 4
```

Prompt de IA Utilizado

> [Okey great, thanks, lets move on to the next step, now we need to start working with tree structures, first you are going to focus on the candidates subsets, which first need to be transformed the words into numbers, using this transformation: { Faros de Xenon =1, Control de Velocidad = 2, Navegador =3, Bluetooth =4, Techo Solar =5, Alarma =6). Once you transform the candidates to the numbers, this numbers must be sorted, for example if you have a subset that ends up being 243, it has to trasform to 234.]

2.6.4. Fase 4: Construcción de Hash Trees para cada suceso de la muestra para cada dimensión

Fundamento Teórico

Esta fase prepara los datos de las transacciones (la muestra) para el conteo. El método estándar de Apriori tomaría el Hash Tree de candidatos (de la Fase 3) y, por cada transacción, generaría todos sus subconjuntos de tamaño k (ej. $k = 2, 3, 4$) y los usaría para "atravesar" el árbol de candidatos e incrementar contadores. Este código implementa un enfoque ligeramente diferente: en lugar de tener un solo árbol de candidatos y "filtrar" las transacciones, construye un árbol para los candidatos (L2, L3, L4) y luego construye un árbol separado *para cada transacción* que contiene todos sus subconjuntos (C2, C3, C4). En la Fase 5, el conteo se realizará comparando los caminos (candidatos) del árbol de candidatos contra los árboles de cada transacción. El primer paso aquí es codificar numéricamente las transacciones (igual que en la Fase 3) y luego generar todos los subconjuntos relevantes (de tamaño 2, 3 y 4) para cada transacción.

Explicación de las funciones utilizadas

- `Filter(function(x) length(x) == 2, all_candidates)`: Separa la lista de candidatos numéricos por tamaño (L2, L3, L4).
- `make_node(item, is_end)`: Función para crear un nuevo nodo del árbol. Usa `new.env()` para crear un entorno, que en R es una estructura eficiente para búsquedas (similar a una tabla hash o diccionario).
- `insert_candidate(root, candidate)`: Inserta un itemset (ej. `c(1, 3)`) en el árbol. Itera por los items; para el item '1', busca un hijo '1' en el nodo actual. Si no existe, lo crea. Luego se "mueve" a

ese hijo y repite el proceso para el item '3'. Al final, marca el último nodo (`is_end = TRUE`) para saber que `1, 3` es un candidato completo.

- `build_tree(candidates)`: Función que inicializa un árbol raíz vacío y llama a `insert_candidate` para cada candidato en la lista.
- `print_tree(node, depth)`: Función recursiva para imprimir el árbol de forma legible.
- `apply(muestra, 1, ...)`: Aplica una función a cada fila (`margin=1`) de la matriz `muestra`. Se usa para transformar cada suceso (fila) en su versión numérica y ordenada.
- `combn(items, k, simplify = FALSE)`: Genera todas las combinaciones de `k` elementos tomados del vector `items`. Es la función clave para generar los subconjuntos de las transacciones (ej. de 1, 2, 3, 4 genera 1 2, 1 3, 1 4, 2 3, 2 4, 3 4).
- `lapply(transactions_num, ...)`: Itera sobre cada transacción (ya convertida a números) y aplica la función `generate_subsets` (que usa `combn`) para crear una lista de listas (C2, C3, C4) para cada una.
- `lapply(transaction_subsets, ...)`: Itera sobre la lista de subconjuntos de cada transacción (creada en el paso anterior) y usa la función `build_tree` (explicada previamente) para crear un árbol C2, C3 y C4 para cada transacción individual.

Explicación de la Fase

[En esta fase se van a construir los hash trees de cada uno de los sucesos de la muestra para cada una de las dimensiones previas (L2, L3, L4). Para llevar a cabo esta tarea se deben llevar a cabo los siguientes pasos. 1^o Se deben transformar los sucesos de la muestra a los valores númericos de la misma forma que se hizo en la fase previa. 2^o Una vez se tengan los valores númericos se deben crear todas las combinaciones de cada suceso de la muestra para cada una de las dimensiones (L2, L3, L4). 3^o Se deben crear los hash-trees para cada dimensión a partir de todas las combinaciones posibles creadas.]

Código y Ejecución (Sweave)

```
> # 4º FASE ----------------------------------------------------
> all_candidates <- c(L2_numerico, L3_numerico, L4_numerico)
> L2_candidatos <- Filter(function(x) length(x) == 2, all_candidates)
> L3_candidatos <- Filter(function(x) length(x) == 3, all_candidates)
> L4_candidatos <- Filter(function(x) length(x) == 4, all_candidates)
> make_node <- function(item = NULL, is_end = FALSE) {
+     node <- new.env(parent = emptyenv())
+     node$item <- item
+     node$children <- list()
+     node$is_end <- is_end
+     node
+ }
> insert_candidate <- function(root, candidate) {
+     current <- root
+     for (it in candidate) {
+         key <- as.character(it)
+         if (!(key %in% names(current$children))) {
+             current$children[[key]] <- make_node(item = it, is_end = FALSE)
+         }
+         current <- current$children[[key]]
+     }
+     current$is_end <- TRUE
+     invisible(NULL)
```

```
+ }
> build_tree <- function(candidates) {
+     root <- make_node(item = NULL, is_end = FALSE)
+     for (cand in candidates) {
+         insert_candidate(root, cand)
+     }
+     root
+ }
> print_tree <- function(node, depth = 0) {
+     indent <- paste(rep("  ", depth), collapse = "")
+     if (is.null(node$item)) {
+         cat("(root)\n")
+     } else {
+         cat(indent, "- ", node$item, if (node$is_end) " [end]\n" else "\n",
+             sep = "")
+     }
+     if (length(node$children) > 0) {
+         keys <- as.integer(names(node$children))
+         keys <- sort(keys, na.last = TRUE)
+         for (k in as.character(keys)) {
+             print_tree(node$children[[k]], depth + 1)
+         }
+     }
+ }
> arbol_L2_candidatos <- build_tree(L2_candidatos)
> arbol_L3_candidatos <- build_tree(L3_candidatos)
> arbol_L4_candidatos <- build_tree(L4_candidatos)
> print_tree(arbol_L2_candidatos)


(root)
  - 1
    - 2 [end]
    - 3 [end]
    - 4 [end]
  - 2
    - 3 [end]
    - 4 [end]
  - 3
    - 4 [end]


> # para los árboles de la muestra
>
> transactions_num <- apply(muestra, 1, function(row) {
+     items <- names(row[row == 1])
+     nums <- unname(item_map[items])
+     sort(nums)
+ })
> generate_subsets <- function(items, k) {
+     if (length(items) < k) return(list())
+     combn(items, k, simplify = FALSE)
+ }
> transaction_subsets <- lapply(transactions_num, function(items) {
+     list(
+         C2 = generate_subsets(items, 2),
+         C3 = generate_subsets(items, 3),
+         C4 = generate_subsets(items, 4)
+     )
```

```
+ })
> all_trees <- lapply(transaction_subsets, function(subs) {
+     list(
+         tree_C2 = if (length(subs$C2) > 0) build_tree(subs$C2) else NULL,
+         tree_C3 = if (length(subs$C3) > 0) build_tree(subs$C3) else NULL,
+         tree_C4 = if (length(subs$C4) > 0) build_tree(subs$C4) else NULL
+     )
+ })
> print_tree(all_trees$suceso1$tree_C2)


(root)
  - 1
    - 2 [end]
    - 3 [end]
    - 4 [end]
  - 2
    - 3 [end]
    - 4 [end]
  - 3
    - 4 [end]
```

Prompt de IA Utilizado

> [okey, so the next step is to transform all the elementos of the sample to the numbers following this mapping function, so that as we have done for the candidate subsets we can transform the sample into trees of k=2, 3 and 4 dimmensions .
>
> The way of getting the subsets of those dimmensions if basically combining all possible ways the values of the sample subsets so that you get subsets of size =2, 3 or 4. Another critical thing is that you need to have the numbers sorted ascendingly so if for example one subset is 2,3,1 it has to be 1,2,3 before it gets transformed to a tree. So with that being said, this is the mapping:
>
> ```
> item_map <- c( "Faros de Xenon" = 1, "Control de Velocidad" = 2, "Navegador" = 3,
> "Bluetooth" = 4, "Techo Solar" = 5, "Alarma" = 6 )
> ```
>
> And this is sample that needs to be transformed first into numbers and then into subsets of size = 2, 3 and 4 so that the trees of this samples are created:
>
> ```
> muestra <- Matrix(c(1,1,1,1,0,0, 1,1,0,1,1,0, 1,1,1,0,0,0, 1,0,1,1,1,0, 1,1,0,1,0,0,
> 0,0,1,0,0,0, 1,1,0,1,0,0, 0,0,0,0,1,1), nrow = 8, ncol = 6, byrow=TRUE,
> dimnames = list(c("suceso1", "suceso2", "suceso3", "suceso4", "suceso5", "suceso6",
> "suceso7", "suceso8"), c("Faros de Xenon", "Control de Velocidad", "Navegador",
> "Bluetooth", "Techo Solar", "Alarma")),sparse=TRUE).
> ```
>
> The trees need to be done for every level for every subset of the sample.]

2.6.5. Fase 5: Conteo sucesos candidatos para analizar la asociación de los que superen el umbral de soporte

Fundamento Teórico

Esta es la fase de **conteo de soporte** (Support Counting). Después de generar candidatos (Lk) y preparar las estructuras de datos, el algoritmo debe determinar cuáles de esos candidatos son realmente "frecuentes". Para ello, se "escanea" la base de datos (en este caso, los árboles de transacciones) y se cuenta

cuántas transacciones "soportan" (contienen) a cada itemset candidato. Un candidato es soportado por una transacción si el itemset candidato es un subconjunto de los items de esa transacción. Al final del conteo, se calcula el soporte relativo de cada candidato (conteo / total de transacciones) y se compara con el umbral de soporte mínimo. Solo los candidatos que igualan o superan el umbral (los *itemsets frecuentes*) se guardan en el conjunto Lk (L2, L3, L4) y se usarán para generar reglas (Fase 6) y para generar los candidatos de la siguiente iteración (Fase 2).

Explicación de las funciones utilizadas

- `get_leaf_paths(node, current_path)`: Función recursiva que atraviesa el árbol de candidatos y devuelve una lista de todos los "caminos" que terminan en un nodo `is_end = TRUE`. Cada camino es un itemset candidato (ej. `list(c(1, 2), c(1, 3), ...)`).
- `path_in_tree(tree, path)`: Comprueba si un `path` (un itemset candidato, ej. `c(1, 3)`) existe dentro de un `tree` (un árbol de subconjuntos de una transacción). Devuelve `TRUE` si puede atravesar el árbol siguiendo el camino, `FALSE` si no.
- `count_supports(candidate_tree, sample_trees_by_level)`: Es la función principal de esta fase.
  - Primero, llama a `get_leaf_paths` para obtener la lista de todos los candidatos (ej. `1, 2, 1, 3, ...`).
  - Luego, para cada candidato (`path`), itera sobre todos los árboles de la muestra (`sample_trees_by_level`).
  - `sapply(...)`: Es un bucle que aplica la función interna a cada suceso (ej. `suceso1`, `suceso2`, ...).
  - Dentro de `sapply`, comprueba el tamaño del candidato (`k`) para seleccionar el árbol de transacciones correcto (tree_C2, tree_C3, o tree_C4).
  - Llama a `path_in_tree` para ver si ese candidato existe en el árbol de ese suceso.
  - `sum(sapply(...))`: Suma todos los `TRUE` (que R trata como 1) y `FALSE` (0) para obtener el conteo de soporte total de ese candidato.
- `data.frame(...)`: Almacena los resultados en un `data.frame` para una visualización y manejo más fácil.
- `filter_by_threshold(support_df, threshold, total_tx)`: Función que toma el `data.frame` de soportes, calcula la proporción (`support_ratio`) y usa `subset()` para quedarse solo con las filas que cumplen la condición `support_ratio >= threshold`.

Explicación de la Fase

[En esta fase se cuenta el número de veces que aparecen los sucesos candidatos en los árboles de los sucesos de la muestra. Este paso es importante ya que solo aquellos sucesos que superen el umbral de soporte serán considerados en la fase de análisis de asociación. De forma que se contabiliza si el n^o de veces que aparecen los sucesos candidatos/n^o de sucesos de la muestra es superior o igual al umbral de soporte]

Código y Ejecución (Sweave)

```
> # 5º FASE ----------------------------------------------------
>
>
> get_leaf_paths <- function(node, current_path = integer()) {
+     paths <- list()
+     if (!is.null(node$item)) {
```

```
+         current_path <- c(current_path, node$item)
+     }
+     if (node$is_end) {
+         paths <- append(paths, list(current_path))
+     }
+     if (length(node$children) > 0) {
+         for (child in node$children) {
+             paths <- append(paths, get_leaf_paths(child, current_path))
+         }
+     }
+     paths
+ }
> path_in_tree <- function(tree, path) {
+     current <- tree
+     for (item in path) {
+         key <- as.character(item)
+         if (!(key %in% names(current$children))) {
+             return(FALSE)
+         }
+         current <- current$children[[key]]
+     }
+    return(TRUE)
+ }
> count_supports <- function(candidate_tree, sample_trees_by_level) {
+     candidate_paths <- get_leaf_paths(candidate_tree)
+     supports <- numeric(length(candidate_paths))
+
+     for (i in seq_along(candidate_paths)) {
+         path <- candidate_paths[[i]]
+         supports[i] <- sum(sapply(sample_trees_by_level, function(trees) {
+             tree_level <- NULL
+             k <- length(path)
+             if (k == 2) tree_level <- trees$tree_C2
+             else if (k == 3) tree_level <- trees$tree_C3
+             else if (k == 4) tree_level <- trees$tree_C4
+             if (is.null(tree_level)) return(FALSE)
+             path_in_tree(tree_level, path)
+         }))
+     }
+     data.frame(
+         itemset = sapply(candidate_paths, function(p) paste(p, collapse = ",")),
+         support = supports
+     )
+ }
> soporte_arbol_L2 <- count_supports(arbol_L2_candidatos, all_trees)
> soporte_arbol_L3 <- count_supports(arbol_L3_candidatos, all_trees)
> soporte_arbol_L4 <- count_supports(arbol_L4_candidatos, all_trees)
> print("Soporte L2 (Antes de filtrar):")


[1] "Soporte L2 (Antes de filtrar):"


> print(soporte_arbol_L2)


  itemset support
1     1,2       5
2     1,3       3
```

```
3     1,4       5
4     2,3       2
5     2,4       4
6     3,4       2
```

```
> print("Soporte L3 (Antes de filtrar):")
```

```
[1] "Soporte L3 (Antes de filtrar):"
```

```
> print(soporte_arbol_L3)
```

```
  itemset support
1   1,2,3       2
2   1,2,4       4
3   1,3,4       2
4   2,3,4       1
```

```
> print("Soporte L4 (Antes de filtrar):")
```

```
[1] "Soporte L4 (Antes de filtrar):"
```

```
> print(soporte_arbol_L4)
```

```
  itemset support
1 1,2,3,4       1
```

```
> filter_by_threshold <- function(support_df, threshold, total_tx) {
+     support_df$support_ratio <- support_df$support / total_tx
+     subset(support_df, support_ratio >= threshold)
+ }
> L2_candidatos_post_arbol <- filter_by_threshold(soporte_arbol_L2,
+                                 support_threshold, numero_sucesos)
> L3_candidatos_post_arbol <- filter_by_threshold(soporte_arbol_L3,
+                                 support_threshold, numero_sucesos)
> L4_candidatos_post_arbol <- filter_by_threshold(soporte_arbol_L4,
+                                 support_threshold, numero_sucesos)
> print("Itemsets Frecuentes L2 (Post-Filtro):")
```

```
[1] "Itemsets Frecuentes L2 (Post-Filtro):"
```

```
> print(L2_candidatos_post_arbol)
```

```
  itemset support support_ratio
1     1,2       5         0.625
3     1,4       5         0.625
5     2,4       4         0.500
```

```
> print("Itemsets Frecuentes L3 (Post-Filtro):")
```

```
[1] "Itemsets Frecuentes L3 (Post-Filtro):"
```

```
> print(L3_candidatos_post_arbol)
```

```
  itemset support support_ratio
2   1,2,4       4           0.5
```

```
> print("Itemsets Frecuentes L4 (Post-Filtro):")
```

```
[1] "Itemsets Frecuentes L4 (Post-Filtro):"
```

```
> print(L4_candidatos_post_arbol)
```

```
[1] itemset       support       support_ratio
<0 rows> (o 0- extensión row.names)
```

Prompt de IA Utilizado

> [okey great, we can now move on to the next phase which consists on counting how many times do the leafs of the candidates tree in each level appear in each of the sample trees as this values need to be counted in order to see which of the candidates subsets move on to the next phase. Also now that i have the supports for every tree subset, we need to filter by threshold that i have already defined previously %support_threshold <- 0.5 % .]

2.6.6. Fase 6: Generación de Reglas de Asociación

Fundamento Teórico

Una vez que se han identificado todos los *itemsets frecuentes* (Lk) que superan el umbral de soporte, el siguiente paso es generar **reglas de asociación** a partir de ellos. Por cada itemset frecuente B (ej. Navegador, Bluetooth), se generan todas las posibles reglas $A \rightarrow (B - A)$, donde A es un subconjunto no vacío de B. Por ejemplo, del itemset 1, 2, 3 (que es frecuente) se pueden generar las reglas:

- 1, 2 $\rightarrow$ 3
- 1, 3 $\rightarrow$ 2
- 2, 3 $\rightarrow$ 1
- 1 $\rightarrow$ 2, 3
- 2 $\rightarrow$ 1, 3
- 3 $\rightarrow$ 1, 2

El número de reglas posibles para un itemset de tamaño k es $2^k - 2$. Esta fase se encarga de generar programáticamente todas estas reglas candidatas. La Fase 7 se encargará de calcular su "fuerza" (confianza).

Explicación de las funciones utilizadas

- `setNames(names(item_map), item_map)`: Crea el mapa inverso (`item_map_rev`), que va de número a nombre (ej. `item_map_rev['1'] = 'Faros de Xenon'`).
- `generate_associations(itemset)`: Función personalizada que genera todas las reglas.
  - `unlist(lapply(1:(k - 1), ...), recursive = FALSE)`: Un truco para generar todos los subconjuntos. Llama a `combn` para $i = 1, 2, ..., k-1$ y los une todos en una sola lista. Esta lista contiene todos los posibles `lhs` (antecedentes) de la regla.
  - `setdiff(itemset, lhs)`: Calcula la diferencia de conjuntos. Si `itemset` es 1, 2, 3 y `lhs` es 1, 2, `rhs` (consecuente) será 3.
- `split_itemset(s)`: Una función de ayuda para convertir el texto `'1,2,3'` de vuelta a un vector numérico `c(1, 2, 3)`.
- `create_associations_from_L(L_df)`: La función principal que orquesta la generación de reglas.
  - Itera sobre cada itemset frecuente en el `data.frame` `L_df`.
  - Llama a `generate_associations` para obtener la lista de reglas (lhs/rhs numéricos).
  - Usa `lapply` para formatear cada regla, usando el `item_map_rev` para traducir los números de vuelta a texto (ej. `paste(item_map_rev[...])`).
  - `do.call(rbind, all_associations)`: Coge la lista de `data.frames` (uno por cada regla) y los apila verticalmente en un solo `data.frame` grande.

Explicación de la Fase

[En esta fase a partir de los sucesos candidatos que superaron el umbral de soporte se van a crear las diversas reglas de asociación para cada uno de ellos. Para saber cuantas reglas de asociación se deben crear por cada suceso de cada dimensión, se debe considerar el cálculo de $2^K - 2$. De forma que si por ejemplo $K = 3$, habrá 6 reglás de asociación que se deben crear y analizar. Para la creación de las reglas de asociación se debe entender que se forman con la siguiente estructura $A \rightarrow B - A$, donde B podría ser $\{Co, Li\}$ y A sería $\{Co\}$ o $\{Li\}$.]

Código y Ejecución (Sweave)

```
> # 6ª FASE ----------------------------------------------------
> item_map_rev <- setNames(names(item_map), item_map)
> generate_associations <- function(itemset) {
+     k <- length(itemset)
+     associations <- list()
+     if (k < 2) return(associations)
+
+     all_subsets <- unlist(lapply(1:(k - 1),
+                                  function(i) combn(itemset, i,
+                                                    simplify = FALSE)),
+                           recursive = FALSE)
+
+     for (lhs in all_subsets) {
+         rhs <- setdiff(itemset, lhs)
+         associations <- append(associations, list(list(lhs = lhs, rhs = rhs)))
+     }
+     associations
+ }
> split_itemset <- function(s) as.integer(unlist(strsplit(s, ",")))
> create_associations_from_L <- function(L_df) {
```

```
+     if (nrow(L_df) == 0) return(data.frame())
+
+     all_associations <- list()
+     for (i in seq_len(nrow(L_df))) {
+         itemset <- split_itemset(L_df$itemset[i])
+         associations <- generate_associations(itemset)
+         all_associations <- append(all_associations,
+                                    lapply(associations, function(r) {
+             data.frame(
+                 lhs = paste(item_map_rev[as.character(sort(r$lhs))],
+                             collapse = ", "),
+                 rhs = paste(item_map_rev[as.character(sort(r$rhs))],
+                             collapse = ", "),
+                 k = length(itemset),
+                 support = L_df$support[i],
+                 support_ratio = L_df$support_ratio[i],
+                 stringsAsFactors = FALSE
+             )
+         }))
+     }
+
+    do.call(rbind, all_associations)
+ }
> associaciones_L2 <- create_associations_from_L(L2_candidatos_post_arbol)
> associaciones_L3 <- create_associations_from_L(L3_candidatos_post_arbol)
> associaciones_L4 <- create_associations_from_L(L4_candidatos_post_arbol)
> print("Reglas generadas desde L2:")

[1] "Reglas generadas desde L2:"

> print(associaciones_L2)

                  lhs                 rhs k support support_ratio
1       Faros de Xenon Control de Velocidad 2       5         0.625
2 Control de Velocidad       Faros de Xenon 2       5         0.625
3       Faros de Xenon            Bluetooth 2       5         0.625
4            Bluetooth       Faros de Xenon 2       5         0.625
5 Control de Velocidad            Bluetooth 2       4         0.500
6            Bluetooth Control de Velocidad 2       4         0.500

> print("Reglas generadas desde L3:")

[1] "Reglas generadas desde L3:"

> print(associaciones_L3)

                                lhs                                rhs k support support_ratio
1                      Faros de Xenon      Control de Velocidad, Bluetooth 3       4           0.5
2                Control de Velocidad            Faros de Xenon, Bluetooth 3       4           0.5
3                           Bluetooth Faros de Xenon, Control de Velocidad 3       4           0.5
4 Faros de Xenon, Control de Velocidad                           Bluetooth 3       4           0.5
5            Faros de Xenon, Bluetooth                Control de Velocidad 3       4           0.5
6      Control de Velocidad, Bluetooth                      Faros de Xenon 3       4           0.5
```

Prompt de IA Utilizado

> [Okay, we can move onto the next phase, no we have to create associations from every level (L2, L3, L4) . The amount of associations that can be created for each subset of each level follows the following formula: $2^k - 2$. So for example for every subset of $k = 2$, there are two possible associations.
>
> Before you start creating every association for each subset you have to reverse the numeric values for the words that we previously changed.]

2.6.7. Fase 7: Cálculo de Confianza y Filtrado Final

Fundamento Teórico

El último paso es filtrar las reglas generadas en la Fase 6 para quedarse solo con las "fuertes". La métrica más común para esto es la **confianza** (Confidence). La confianza de una regla $A \rightarrow B$ mide la probabilidad condicional de que B ocurra dado que A ha ocurrido. Se calcula usando los soportes (conteos) que ya hemos calculado:

$$\text{Confianza}(A \rightarrow B) = \frac{\text{Soporte}(A \cup B)}{\text{Soporte}(A)} = \frac{\text{Conteo}(A \cup B)}{\text{Conteo}(A)}$$

Por ejemplo, la confianza de la regla Navegador $\rightarrow$ Bluetooth es:

$$\frac{\text{Conteo}(\{\text{Navegador, Bluetooth}\})}{\text{Conteo}(\{\text{Navegador}\})}$$

El `Conteo(A U B)` es el soporte del itemset frecuente completo (ej. el soporte de Navegador, Bluetooth que se calculó en la Fase 5). El `Conteo(A)` es el soporte del antecedente (ej. el soporte de Navegador que se calculó en la Fase 1 o 5). Solo las reglas que superan un umbral de confianza mínimo (en este caso, 0.8) se consideran "fuertes" y se presentan como resultado final.

Explicación de las funciones utilizadas

- `convert_numeric_str_to_text_str(...)`: Función de ayuda para crear la tabla de búsqueda (`master_support_lookup`), convirtiendo itemsets numéricos ("1,3") a texto ("Faros de Xenon, Navegador").
- `soporte_L1_map_df <- data.frame(...)`: Crea un `data.frame` que sirve como tabla de búsqueda para los soportes de L1 (antecedentes de tamaño 1). Extrae los nombres (`names(frequency)`) y los conteos (`as.integer(frequency)`) del vector `frequency` de la Fase 1.
- `soporte_L2_map_df <- data.frame(...)` (y `L3`): Crea tablas de búsqueda similares para L2 y L3. Usa `sapply` para aplicar la función `convert_numeric_str_to_text_str` a los itemsets numéricos (ej. "1,3") y los convierte a texto (ej. "Faros de Xenon, Navegador"), asociándolos con sus conteos de soporte (`soporte_arbol_L2$support`).
- `rbind(soporte_L1_map_df, ...)`: Combina los soportes de L1, L2 y L3 en una sola tabla "maestra" (`master_support_lookup`). Esta tabla es crucial, ya que contiene el denominador (`Soporte(A)`) necesario para calcular la confianza.
- `na.omit(...)`: Elimina filas con valores `NA` (nulos) que puedan haber aparecido.
- `calculate_and_filter_confidence(...)`: Función principal de esta fase.
  - `merge(associations_df, support_lookup, ...)`: Realiza una operación clave de "unión" (como en SQL). Une la tabla de reglas `associations_df` con la tabla de soportes `support_lookup` usando el antecedente (`lhs`) como clave. Esto añade eficientemente la columna `lhs_support` (el denominador) a la tabla de reglas.

- `merged_df$confidence <- merged_df$support / merged_df$lhs_support`: Calcula la confianza dividiendo el soporte del itemset completo (Soporte(A U B)) por el soporte del antecedente (Soporte(A)).
- `is.nan(merged_df$confidence)`: Comprueba si hay divisiones 0/0 (Not a Number) y las convierte a 0.
- `order(-strong_rules$confidence)`: Ordena los resultados de mayor a menor confianza.

▪ `subset(reglas_fuertes_L2, confidence >= confidence_threshold)`: Finalmente, filtra la tabla de reglas para quedarse solo con aquellas que superan el umbral de confianza.

Explicación de la Fase

[En esta fase final se va a realizar el cálculo de confianza de cada una de las reglas de asociación para así poder sacar conclusiones de la muestra con cierto grado de confianza. Para llevar a cabo este cálculo se deberá dividir el n^o de veces que aparece B en la muestra/n^o de veces que aparece A en la muestra. Si por ejemplo se tiene que $B = \{Co, Li\}$ y se tiene la siguiente regla de asociación $Co \rightarrow Li$ en este caso $A = Co$. Y esos son los conjuntos que se deben contabilizar de la muestra. Aquellas reglas de asociación que superen dicho umbral de confianza, se podrá afirmar con el umbral de confianza que si sucede una, la otra también sucede con ese grado de confianza.]

Código y Ejecución (Sweave)

```
> # 7ª FASE ----------------------------------------------------
> confidence_threshold <- 0.8
> convert_numeric_str_to_text_str <- function(num_str, map_rev) {
+     nums_numeric <- split_itemset(num_str)
+     nums_char <- as.character(nums_numeric)
+     names <- unname(map_rev[nums_char])
+     paste(names, collapse = ", ")
+ }
> soporte_L1_map_df <- data.frame(
+     lhs_str = names(frequency),
+     lhs_support = as.integer(frequency),
+     stringsAsFactors = FALSE
+ )
> soporte_L2_map_df <- data.frame(
+     lhs_str = sapply(soporte_arbol_L2$itemset,
+                      convert_numeric_str_to_text_str,
+                      map_rev = item_map_rev),
+     lhs_support = soporte_arbol_L2$support,
+     stringsAsFactors = FALSE
+ )
> soporte_L3_map_df <- data.frame(
+     lhs_str = sapply(soporte_arbol_L3$itemset,
+                      convert_numeric_str_to_text_str,
+                      map_rev = item_map_rev),
+     lhs_support = soporte_arbol_L3$support,
+     stringsAsFactors = FALSE
+ )
> master_support_lookup <- rbind(soporte_L1_map_df,
+                                soporte_L2_map_df,
+                                soporte_L3_map_df)
> master_support_lookup <- na.omit(master_support_lookup)
> calculate_and_filter_confidence <- function(associations_df,
+                                             support_lookup, threshold) {
```

```
+     if (nrow(associations_df) == 0) {
+         return(data.frame(
+             lhs = character(),
+             rhs = character(),
+             k = integer(),
+             support_ratio = numeric(),
+             confidence = numeric(),
+             stringsAsFactors = FALSE
+         ))
+     }
+
+     merged_df <- merge(associations_df, support_lookup,
+                        by.x = "lhs", by.y = "lhs_str", all.x = TRUE)
+
+     merged_df$lhs_support[is.na(merged_df$lhs_support)] <- 0
+
+     # Cálculo de la confianza: support(A U B) / support(A)
+     merged_df$confidence <- merged_df$support / merged_df$lhs_support
+
+     merged_df$confidence[is.nan(merged_df$confidence)] <- 0
+
+     strong_rules <- merged_df[, c("lhs", "rhs", "k",
+                                   "support_ratio", "confidence")]
+
+
+     strong_rules[order(-strong_rules$confidence), ]
+ }
> reglas_fuertes_L2 <- calculate_and_filter_confidence(associaciones_L2,
+                                      master_support_lookup,
+                                      confidence_threshold)
> reglas_fuertes_L3 <- calculate_and_filter_confidence(associaciones_L3,
+                                      master_support_lookup,
+                                      confidence_threshold)
> reglas_fuertes_L4 <- calculate_and_filter_confidence(associaciones_L4,
+                                      master_support_lookup,
+                                      confidence_threshold)
> # Filtrado final
> reglas_finales_L2 <- subset(reglas_fuertes_L2,
+                             confidence >= confidence_threshold)
> reglas_finales_L3 <- subset(reglas_fuertes_L3,
+                             confidence >= confidence_threshold)
> reglas_finales_L4 <- subset(reglas_fuertes_L4,
+                             confidence >= confidence_threshold)
> print("--- Reglas que superan el umbral de confianza (0.8) ---")

[1] "--- Reglas que superan el umbral de confianza (0.8) ---"

> print("Reglas Finales L2 (Columnas de texto):")

[1] "Reglas Finales L2 (Columnas de texto):"

> if(nrow(reglas_finales_L2) > 0) print(reglas_finales_L2[, c("lhs", "rhs")])

                  lhs                  rhs
1           Bluetooth       Faros de Xenon
```

```
3 Control de Velocidad       Faros de Xenon
5       Faros de Xenon Control de Velocidad
6       Faros de Xenon            Bluetooth
2            Bluetooth Control de Velocidad
4 Control de Velocidad            Bluetooth
```

```
> print("Reglas Finales L2 (Columnas numéricas):")
```

```
[1] "Reglas Finales L2 (Columnas numéricas):"
```

```
> if(nrow(reglas_finales_L2) > 0) print(reglas_finales_L2[, c("k", "support_ratio", "confidence")])
```

```
  k support_ratio confidence
1 2         0.625  1.0000000
3 2         0.625  1.0000000
5 2         0.625  0.8333333
6 2         0.625  0.8333333
2 2         0.500  0.8000000
4 2         0.500  0.8000000
```

```
> print("Reglas Finales L3 (Columnas de texto):")
```

```
[1] "Reglas Finales L3 (Columnas de texto):"
```

```
> if(nrow(reglas_finales_L3) > 0) print(reglas_finales_L3[, c("lhs", "rhs")])
```

```
                                lhs                                rhs
3      Control de Velocidad, Bluetooth                Faros de Xenon
1                           Bluetooth Faros de Xenon, Control de Velocidad
2                 Control de Velocidad            Faros de Xenon, Bluetooth
5            Faros de Xenon, Bluetooth            Control de Velocidad
6 Faros de Xenon, Control de Velocidad                       Bluetooth
```

```
> print("Reglas Finales L3 (Columnas numéricas):")
```

```
[1] "Reglas Finales L3 (Columnas numéricas):"
```

```
> if(nrow(reglas_finales_L3) > 0) print(reglas_finales_L3[, c("k", "support_ratio", "confidence")])
```

```
  k support_ratio confidence
3 3           0.5        1.0
1 3           0.5        0.8
2 3           0.5        0.8
5 3           0.5        0.8
6 3           0.5        0.8
```

```
> print("Reglas Finales L4 (Columnas de texto):")
```

```
[1] "Reglas Finales L4 (Columnas de texto):"
```

```
> if(nrow(reglas_finales_L4) > 0) print(reglas_finales_L4[, c("lhs", "rhs")])
> print("Reglas Finales L4 (Columnas numéricas):")
```

```
[1] "Reglas Finales L4 (Columnas numéricas):"
```

```
> if(nrow(reglas_finales_L4) > 0) print(reglas_finales_L4[, c("k", "support_ratio", "confidence")])
```

2.7. Ejercicio 2.3: Detección de Datos Anómalos

En este ejercicio se implementaron tres métodos distintos para la detección de datos anómalos, utilizando técnicas estadísticas básicas. Se trabajó con dos variables: `Velocidad` y `Temperatura`. El conjunto de datos empleado fue el siguiente:

```
> # Creación del conjunto de datos
> datos <- data.frame(
+   Velocidad = c(10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5),
+   Temperatura = c(7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73)
+ )
```

2.7.1. Funciones auxiliares comunes

Antes de proceder con los métodos de detección, se implementaron varias funciones auxiliares necesarias para reemplazar operaciones básicas. La función `contar_elementos(x)` permite obtener la cantidad de elementos de un vector mediante un bucle `for`, acumulando una unidad por cada iteración. Su salida es un valor numérico entero que representa el número de observaciones contenidas en el vector. Por otro lado, la función `mi_suma(x)` se encarga de calcular la suma total de los elementos de un vector numérico. El procedimiento consiste en inicializar un acumulador en cero e ir sumando cada valor del vector en un bucle. A partir de ambas funciones se define `media_manual(x)`, que calcula la media aritmética dividiendo la suma total obtenida por `mi_suma(x)` entre el número de elementos devuelto por `contar_elementos(x)`. Esta función devuelve la media muestral de los valores contenidos en el vector. Para calcular la varianza y la desviación típica, se implementaron las funciones `varianza_manual(x)` y `desviacion_manual(x)`. La primera recorre todos los valores del vector y acumula la suma de los cuadrados de las desviaciones respecto a la media, dividiendo después el resultado por el número total de elementos. La segunda función simplemente aplica la raíz cuadrada a la varianza calculada para obtener la desviación típica.

```
> # ============================================================
> # FUNCIONES AUXILIARES BÁSICAS
> # ============================================================
>
> # Contar elementos manualmente
> contar_elementos <- function(x) {
+   n <- 0
+   for (_ in x) n <- n + 1
+   n
+ }
> # Suma manual
> mi_suma <- function(x) {
+   total <- 0
+   for (valor in x) total <- total + valor
+   total
+ }
> # Media manual
> media_manual <- function(x) {
+   suma <- mi_suma(x)
+   n <- contar_elementos(x)
+   suma / n
+ }
> # Varianza manual
> varianza_manual <- function(x) {
+   n <- contar_elementos(x)
+   media <- media_manual(x)
+   suma <- 0
```

```
+   for (v in x) suma <- suma + (v - media)^2
+   suma / n
+ }
> # Desviación típica manual
> desviacion_manual <- function(x) {
+   sqrt(varianza_manual(x))
+ }
```

2.7.2. Medidas de ordenación (Velocidad), Método Caja y Bigotes

El primer método utilizado para la detección de valores atípicos se basa en el rango intercuartílico (IQR). Este enfoque utiliza medidas de posición, calculando los cuartiles primero (Q_1 y Q_3) y el rango intercuartílico como la diferencia $IQR = Q_3 - Q_1$. Un dato se considera atípico si está por debajo del límite inferior $Q_1 - 1{,}5 \cdot IQR$ o por encima del límite superior $Q_3 + 1{,}5 \cdot IQR$. En la implementación, se aprovechó la función `fivenum(x)`, que devuelve los cinco números resumen de un conjunto de datos: el valor mínimo, el primer cuartil (Q1), la mediana (Q2), el tercer cuartil (Q3) y el valor maximo. Estos numeros resumen permiten calcular de manera sencilla el rango intercuartílico y establecer los límites que definen los posibles outliers. Posteriormente, mediante un bucle `for`, se recorrió cada elemento del vector iterando directamente sobre los valores del vector. El índice (i) se gestiona de forma manual (iniciando en i <- 1 y sumando i <- i + 1 en cada ciclo), con el único fin de poder reportar la posición del outlier. Cada elemento del vector se campara con los límetes obtenidos. Cuando un valor excedía dichos límites, se imprimía por pantalla el índice y el valor detectado como atípico. Esta lógica se encapsuló en la función `detectar_outliers_iqr(x)`. Dicha función recibe un vector numérico como argumento y no devuelve ningún valor formal, sino que imprime en la consola los resultados. En los datos analizados, no se detectaron valores atípicos en la variable `Velocidad`, puesto que todas las observaciones quedaron dentro del intervalo definido por los límites del IQR.

```
> # Método IQR manual
> detectar_outliers_iqr <- function(x) {
+   resumen <- fivenum(x)
+   Q1 <- resumen[2]
+   Q3 <- resumen[4]
+   IQRv <- Q3 - Q1
+   lim_inf <- Q1 - 1.5 * IQRv
+   lim_sup <- Q3 + 1.5 * IQRv
+
+   cat("\n--- DETECCIÓN OUTLIERS IQR ---\n")
+   cat("Límite inferior:", lim_inf, "\n")
+   cat("Límite superior:", lim_sup, "\n")
+
+   i <- 1
+   hay_outliers <- FALSE
+   for (v in x) {
+     if (v < lim_inf || v > lim_sup) {
+       cat("Índice:", i, "→ Valor =", v, "es un outlier\n")
+       hay_outliers <- TRUE
+     }
+     i <- i + 1
+   }
+   if (!hay_outliers) cat("No se detectaron outliers.\n")
+ }
```

Prompt de IA Utilizado

> [I have the following dataset of Speed values: 10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5. I want you to apply the box-and-whisker outlier detection method. To do this, follow these steps: sort the data, calculate the first and third quartiles, compute the interquartile range (IQR) using the formula provided earlier, determine the detection limits, and finally indicate which of the original Speed values fall outside this interval. You are not allowed to use: matrix, boxplot, quantile, length, mean, or sd.]

2.7.3. Medidas de dispersión (Temperatura), Método Desviación Típica

El segundo método asume una distribución aproximadamente normal de los datos y detecta valores anómalos comparando cada observación con la media y la desviación típica. Un valor se considera atípico si se encuentra fuera del intervalo $[\bar{x} - d \cdot s, \bar{x} + d \cdot s]$, donde s es la desviación típica y d es un multiplicador elegido (en este caso $d = 2$). Para implementarlo, se calculó la media y la desviación típica mediante las funciones `media_manual(x)` y `desviacion_manual(x)` previamente definidas. Luego, se obtuvieron los límites inferior y superior, y se recorrió el vector de `Temperatura` en un bucle iterando directamente sobre sus valores, imprimiendo por pantalla los valores que se situaban fuera del rango aceptado. Se gestiona un contador de índice (i) de forma manual e interna, únicamente para poder reportar la posición del outlier si se encuentra. Toda la lógica se encapsuló en la función `detectar_outliers_sd(x, d)` que acepta como argumentos el vector de datos y el multiplicador de desviación. La salida consiste en un mensaje por consola indicando si se han encontrado o no observaciones anómalas. En la práctica, al aplicar esta función sobre la variable `Temperatura`, se detectó un único valor atípico correspondiente a la observación con valor 12.74, ya que sobrepasaba claramente el límite superior establecido por el criterio de dos desviaciones típicas respecto a la media.

```
> # Método de la desviación típica manual
> detectar_outliers_sd <- function(x, d = 2) {
+   media <- media_manual(x)
+   desv <- desviacion_manual(x)
+   lim_inf <- media - d * desv
+   lim_sup <- media + d * desv
+
+   cat("\n--- DETECCIÓN OUTLIERS DESVIACIÓN TÍPICA ---\n")
+   cat("Media:", media, "\n")
+   cat("Desviación típica:", desv, "\n")
+   cat("Límite inferior:", lim_inf, "\n")
+   cat("Límite superior:", lim_sup, "\n")
+
+   i <- 1
+   hay_outliers <- FALSE
+   for (valor in x) {
+     if (valor < lim_inf || valor > lim_sup) {
+       cat("Índice:", i, "→ Valor =", valor, "es un outlier\n")
+       hay_outliers <- TRUE
+     }
+     i <- i + 1
+   }
+   if (!hay_outliers) cat("No se detectaron outliers.\n")
+ }
```

Prompt de IA Utilizado

> [Now you have to do the same as before but with different data and another method. Analyze the following Temperature dataset: 7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42,

5.73. I want you to identify outliers using the standard deviation method. It is important to follow this procedure: calculate the arithmetic mean of all values; compute the standard deviation; then apply the outlier detection rule using the formulas: lower limit = mean(x) - d × standard deviation, and upper limit = mean(x) + d × standard deviation. The code must check the data and indicate which values are outliers because they do not meet this condition. You are not allowed to use: lm, summary, or length.]

2.7.4. Detección por regresión lineal

Finalmente, se utilizó un enfoque basado en regresión lineal entre `Velocidad` (variable independiente) y `Temperatura` (variable dependiente). Se calcularon las medias de Velocidad y Temperatura con las funciones del ejercicio 2.1. Después, se calcularon los coeficientes de la recta de regresión. Los cálculos se hicieron manualmente iterando sobre los datos, sin usar funciones R predefinidas como lm(). Una vez obtenida la recta, se calcularon los residuos. Para cada, se calculó la temperatura estimada y el residuo. Luego, se calculó la desviación típica de los residuos. Y se estableció el criterio de anomalía: un punto se considera un outlier si su residuo absoluto es mayor que dos veces el error estándar de los residuos. Por último, se iteró sobre los residuos para identificar los puntos que cumplían esta condición. La función implementada para este método, `detectar_outliers_regresion(x, y, d)`, recibe los vectores de la variable independiente y dependiente, así como el parámetro multiplicador d. Internamente, la función calcula las medias y las sumas necesarias mediante bucles, obtiene la pendiente y la ordenada en el origen, y recorre las observaciones comparando cada residuo con el umbral $d \cdot s_r$. Si algún residuo supera dicho valor absoluto, se imprime su índice junto al valor correspondiente. En este conjunto de datos, la ecuación de regresión obtenida fue $\hat{y} = 3{,}0025 + 0{,}4997 \cdot x$ y el error estándar de los residuos fue $s_r = 1{,}1183$. Al aplicar el criterio de outlier ($|r_i| > 2 \cdot s_r$), se detectó un valor atípico correspondiente al índice 3, con residuo $r_3 = 3{,}2411$, el cual excede el umbral de $2 \cdot s_r$. Por tanto, este punto se considera un outlier según el análisis por residuos de regresión.

```
> # Método de regresión manual con detección por residuos
> detectar_outliers_regresion <- function(x, y, d = 2) {
+   n <- contar_elementos(x)
+   media_x <- media_manual(x)
+   media_y <- media_manual(y)
+
+   # Calcular pendiente y ordenada al origen
+   num <- 0
+   den <- 0
+   for (i in seq(n)) {
+     num <- num + (x[i] - media_x) * (y[i] - media_y)
+     den <- den + (x[i] - media_x)^2
+   }
+   b1 <- num / den
+   b0 <- media_y - b1 * media_x
+
+   # Calcular residuos
+   residuos <- numeric(n)
+   for (i in seq(n)) {
+     y_est <- b0 + b1 * x[i]
+     residuos[i] <- y[i] - y_est
+   }
+
+   # Calcular error estándar de residuos
+   suma_res <- 0
+   for (r in residuos) suma_res <- suma_res + r^2
+   error_est <- sqrt(suma_res / n)
+
+   cat("\n--- DETECCIÓN OUTLIERS REGRESIÓN ---\n")
```

```
+   cat("Ecuación: y =", round(b0, 4), "+", round(b1, 4), "* x\n")
+   cat("Error estándar de residuos:", round(error_est, 4), "\n")
+
+   i <- 1
+   hay_outliers <- FALSE
+   for (r in residuos) {
+     if (abs(r) > d * error_est) {
+       cat("Índice:", i, "→ Residuo =", round(r, 4), "es un outlier\n")
+       hay_outliers <- TRUE
+     }
+     i <- i + 1
+   }
+   if (!hay_outliers) cat("No se detectaron outliers en los residuos.\n")
+ }
>
```

Prompt de IA Utilizado

> [Final exercise. We take the 11 pairs of data Speed, Temperature: 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. Perform an outlier analysis based on regression residuals, where Speed is the independent variable and Temperature is the dependent variable. Please follow these steps manually, without using functions such as lm or summary: first, calculate the mean of Speed and Temperature; then obtain the linear regression coefficients a and b (b = Sxy / Sx^2). For each of the 11 points, compute its residual and the standard deviation of the residuals. Finally, the code must indicate which points are outliers according to the condition |yi - ŷi| >d × sr"]

2.7.5. Aplicación de los métodos

Finalmente, se aplicaron las funciones previamente definidas sobre el conjunto de datos `datos`. Cada función analiza la variable correspondiente y reporta los valores detectados como outliers. El código ejecutable es el siguiente:

```
> # ============================================================
> # APLICACIÓN DE LOS MÉTODOS
> # ============================================================
>
> detectar_outliers_iqr(datos$Velocidad)


--- DETECCIÓN OUTLIERS IQR ---
Límite inferior: -1
Límite superior: 19
No se detectaron outliers.


> detectar_outliers_sd(datos$Temperatura)


--- DETECCIÓN OUTLIERS DESVIACIÓN TÍPICA ---
Media: 7.5
Desviación típica: 1.935933
Límite inferior: 3.628134
Límite superior: 11.37187
Índice: 3 → Valor = 12.74 es un outlier
```

```
> detectar_outliers_regresion(datos$Velocidad, datos$Temperatura)


--- DETECCIÓN OUTLIERS REGRESIÓN ---
Ecuación: y = 3.0025 + 0.4997 * x
Error estándar de residuos: 1.1183
Índice: 3 → Residuo = 3.2411 es un outlier
```

2.8. Ejercicio 2.4: Detección de datos anómalos (Proximidad y Densidad Manual)

En este ejercicio se aborda la detección de datos anómalos utilizando dos enfoques diferentes: proximidad (K-NN) y densidad (LOF). Para cumplir con el requisito de trabajo propio(y no guiado por el profesor) y profundizar en los fundamentos teóricos, ambos algoritmos se implementarán de forma **manual**, sin recurrir a paquetes específicos de R para la detección de *outliers*.

2.8.1. 1. Descripción del conjunto de datos

El conjunto de datos objeto de estudio contiene información sobre la asistencia a cinco seminarios de biología. Consta de $N = 5$ observaciones y dos variables cuantitativas discretas:

- **Mujeres:** Número de mujeres asistentes al seminario.
- **Hombres:** Número de hombres asistentes al seminario.

Los datos se encuentran almacenados en el fichero de texto plano `seminarios.txt`.

2.8.2. Funciones de R utilizadas en la exploración inicial

Para la carga y visualización preliminar de los datos se han utilizado las siguientes funciones base de R. Se detallan sus argumentos clave para justificar su uso en la creación de gráficos informativos:

- `read.table(file, header = TRUE)`: Carga los datos desde un fichero de texto. El argumento `header = TRUE` indica a R que interprete la primera fila como los nombres de las variables ("Mujeres", "Hombres").
- `plot(x, y, ...)`: Función genérica para crear gráficos. En este caso, genera un diagrama de dispersión. Se han personalizado sus argumentos para mejorar la interpretabilidad:
  - `main`, `xlab`, `ylab`: Definen el título principal y las etiquetas de los ejes X e Y.
  - `pch = 19`: Selecciona el tipo de símbolo para los puntos (círculo sólido), más visible que el valor por defecto.
  - `col`: Define el color de los puntos.
  - `xlim`, `ylim`: Fuerzan los límites de los ejes (de 0 a 12) para asegurar que el origen (0,0) y todos los puntos sean visibles con perspectiva correcta.
  - `las = 1`: Fuerza a que las etiquetas numéricas de los ejes se muestren siempre horizontales, facilitando la lectura.
- `grid()`: Añade una rejilla rectangular al gráfico existente, lo que ayuda a estimar visualmente las coordenadas de los puntos.
- `text(x, y, labels, ...)`: Añade texto al gráfico en las coordenadas especificadas. Se utiliza para etiquetar cada punto con su identificador (P1, P2, etc.).
  - `labels`: El texto a mostrar.
  - `pos = 3`: Coloca el texto justo **encima** de la coordenada especificada, evitando que se superponga con el punto.
  - `cex`: Ajusta el tamaño relativo del texto.
- `paste(...)`: Concatena cadenas de texto. Se usa dentro de `text()` para crear las etiquetas dinámicamente (ej. combinando la letra "Pçon el número de fila).
- `rownames(x)`: Devuelve los nombres (o números) de las filas de un data frame, utilizado aquí para identificar cada observación secuencialmente.

2.8.3. Carga y Exploración Visual

A continuación se muestra el código para cargar los datos y generar la visualización inicial.

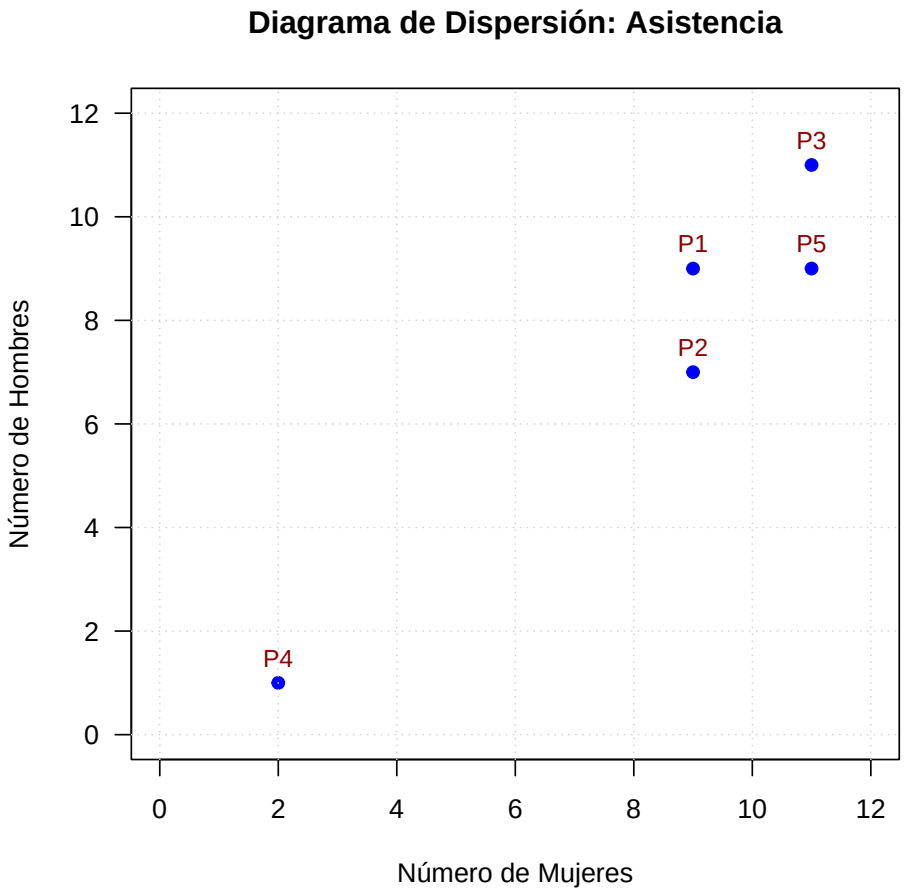
```
> # --- 1. Carga de datos ---
> s_bio <- read.table("seminarios.txt", header = TRUE)
> print("Conjunto de datos original (Asistencia a Seminarios):")
```

```
[1] "Conjunto de datos original (Asistencia a Seminarios):"
```

```
> print(s_bio)
```

```
  Mujeres Hombres
1       9       9
2       9       7
3      11      11
4       2       1
5      11       9
```

```
> # --- 2. Exploración Visual ---
> # Gráfico de dispersión ajustado para visualizar posibles outliers
> plot(s_bio$Mujeres, s_bio$Hombres,
+      main = "Diagrama de Dispersión: Asistencia",
+      xlab = "Número de Mujeres", ylab = "Número de Hombres",
+      pch = 19, col = "blue",       # Puntos sólidos azules
+      xlim = c(0, 12), ylim = c(0, 12), # Ejes fijos para ver el contexto global
+      las = 1)                      # Etiquetas horizontales
> # Añadir rejilla y etiquetas a los puntos
> grid()
> text(s_bio$Mujeres, s_bio$Hombres,
+      labels = paste("P", rownames(s_bio), sep=""),
+      pos = 3, cex = 0.9, col = "darkred")
```



Análisis preliminar. La visualización revela una estructura clara en los datos: cuatro de las cinco observaciones (P1, P2, P3, P5) forman un grupo compacto en la región de alta asistencia (valores superiores a 9 en al menos uno de los ejes). Por contraposición, el punto **P4 (2, 1)** aparece aislado en la región de baja asistencia, muy distante del resto del grupo. Esta inspección visual sugiere fuertemente que P4 será identificado como *outlier* tanto por métodos de proximidad (por su gran distancia a los vecinos) como de densidad (por estar en una región vacía).

2.8.4. Método de Proximidad: K-Vecinos (Implementación Manual)

En esta sección se desarrolla un algoritmo basado en la proximidad para la detección de anomalías. El enfoque fundamental consiste en medir cuán aislado se encuentra un punto respecto al resto de la muestra.

Fundamento teórico general. El método de los K-Vecinos más cercanos (K-NN) para detección de *outliers* se basa en la premisa de que las observaciones normales pertenecen a regiones densas (tienen vecinos cerca), mientras que las anómalas están alejadas. El procedimiento estándar consiste en:

1. Calcular la distancia entre todos los pares de observaciones posibles. Habitualmente se emplea la **distancia euclídea**:
$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$
2. Para cada punto, ordenar las distancias a los demás de menor a mayor.
3. Identificar la distancia a su k-ésimo vecino más cercano, denotada como $d^k(P_i)$.
4. Comparar esta distancia con un umbral predefinido D^*. Si $d^k(P_i) > D^*$, el punto se considera un *outlier* por proximidad.

Dado el reducido tamaño de la muestra ($N = 5$), se selecciona $K = 3$ para evaluar la cercanía a la mayoría del grupo, y un umbral $D^* = 5$ basado en la dispersión observada en los ejes del gráfico inicial.

Implementación manual en R. Para cumplir con el objetivo de realizar una implementación desde cero, se ha desarrollado el algoritmo sin recurrir a la función `dist()` de R para el cálculo de distancias, optando por un enfoque iterativo mediante bucles. El procedimiento se estructura en tres fases lógicas:

1. **Cálculo de la Matriz de Distancias:** Se inicializa una matriz cuadrada de dimensión $N \times N$ con ceros. Mediante un doble bucle `for`, se recorre cada par de puntos (i, j), se calcula la diferencia al cuadrado de sus coordenadas (mujeres y hombres), se suman estas diferencias y se obtiene la raíz cuadrada. Este valor se asigna a la posición $[i, j]$ de la matriz.
2. **Identificación de Vecinos:** Para cada observación (fila i de la matriz de distancias), se utiliza la función `order()`. Esta función es crucial aquí, ya que no devuelve las distancias ordenadas, sino los *índices* de los puntos ordenados por cercanía. Esto nos permite saber no solo *cuál* es la distancia al tercer vecino, sino *quién* es ese vecino.
3. **Detección:** Se selecciona la distancia correspondiente al K-ésimo vecino (ignorando la distancia 0 del punto a sí mismo) y se compara contra el umbral D^* para emitir el diagnóstico final.

A continuación se presenta el código completo implementando esta lógica:

```
> # --- Parámetros del algoritmo K-NN ---
> K <- 3            # Buscamos el 3er vecino más cercano
> UMBRAL_DIST <- 5  # Si la distancia al 3er vecino es > 5, es outlier
> # --- Paso 1: Cálculo Manual de la Matriz de Distancias ---
> # Convertimos el data frame a matriz para agilizar los cálculos por índices
> datos_m <- as.matrix(s_bio)
> n <- nrow(datos_m)
> # Inicializamos una matriz vacía de n x n para almacenar las distancias
> matriz_dist <- matrix(0, nrow=n, ncol=n)
> # Doble bucle para calcular la distancia euclídea par a par
```

```
> for (i in 1:n) {
+   for (j in 1:n) {
+     # Se calcula la diferencia al cuadrado para cada dimensión
+     dif_mujeres_sq <- (datos_m[i, 1] - datos_m[j, 1])^2
+     dif_hombres_sq <- (datos_m[i, 2] - datos_m[j, 2])^2
+
+     # Se aplica la fórmula: raíz cuadrada de la suma de diferencias al cuadrado
+     matriz_dist[i, j] <- sqrt(dif_mujeres_sq + dif_hombres_sq)
+   }
+ }
> cat("Matriz de Distancias Euclídeas (calculada manualmente):\n")

Matriz de Distancias Euclídeas (calculada manualmente):

> print(round(matriz_dist, 2))

      [,1] [,2]  [,3]  [,4]  [,5]
[1,]  0.00 2.00  2.83 10.63  2.00
[2,]  2.00 0.00  4.47  9.22  2.83
[3,]  2.83 4.47  0.00 13.45  2.00
[4,] 10.63 9.22 13.45  0.00 12.04
[5,]  2.00 2.83  2.00 12.04  0.00

> # --- Paso 2: Análisis de Vecindad ---
> cat("\nAnálisis de vecindad (K =", K, "):\n")

Análisis de vecindad (K = 3 ):

> # Vector para almacenar la distancia determinante para cada punto
> distancias_k3 <- numeric(n)
> for (i in 1:n) {
+   # Extraemos la fila i: distancias desde el punto P_i a todos los demás
+   dists_punto <- matriz_dist[i, ]
+
+   # Obtenemos los índices que ordenarían estas distancias de menor a mayor.
+   # El primer índice siempre será el propio i (distancia 0).
+   orden_vecinos <- order(dists_punto)
+
+   # Los vecinos reales son los que están a partir de la posición 2.
+   # Seleccionamos los índices de los K primeros vecinos reales.
+   indices_k_vecinos <- orden_vecinos[2:(K + 1)]
+
+   # La distancia determinante es la del K-ésimo vecino real (posición K+1)
+   dist_k_esimo <- dists_punto[orden_vecinos[K + 1]]
+   distancias_k3[i] <- dist_k_esimo
+
+   # Mostramos los resultados intermedios para trazabilidad
+   vecinos_str <- paste("P", indices_k_vecinos, sep="", collapse=", ")
+   cat(" -> Punto P", i, " tiene sus ", K, " vecinos más cercanos en: {",
+       vecinos_str, "}\n", sep="")
+   cat("    Su distancia al 3er vecino es: ", round(dist_k_esimo, 2), "\n", sep="")
+ }

 -> Punto P1 tiene sus 3 vecinos más cercanos en: {P2, P5, P3}
    Su distancia al 3er vecino es: 2.83
```

```
 -> Punto P2 tiene sus 3 vecinos más cercanos en: {P1, P5, P3}
    Su distancia al 3er vecino es: 4.47
 -> Punto P3 tiene sus 3 vecinos más cercanos en: {P5, P1, P2}
    Su distancia al 3er vecino es: 4.47
 -> Punto P4 tiene sus 3 vecinos más cercanos en: {P2, P1, P5}
    Su distancia al 3er vecino es: 12.04
 -> Punto P5 tiene sus 3 vecinos más cercanos en: {P1, P3, P2}
    Su distancia al 3er vecino es: 2.83


> # --- Paso 3: Decisión Final (Detección) ---
> cat("\n--- Resultados Finales K-NN (Umbral =", UMBRAL_DIST, ") ---\n")


--- Resultados Finales K-NN (Umbral = 5 ) ---


> for (i in 1:n) {
+   # Aplicación de la regla de decisión: dist_k > umbral => OUTLIER
+   es_outlier <- distancias_k3[i] > UMBRAL_DIST
+
+   estado <- if (es_outlier) "DETECTADO COMO OUTLIER" else "Normal"
+
+   cat("Punto P", i, " [Dist. K3 = ", round(distancias_k3[i], 2), "] \t-> ",
+       estado, "\n", sep="")
+ }


Punto P1 [Dist. K3 = 2.83]         -> Normal
Punto P2 [Dist. K3 = 4.47]         -> Normal
Punto P3 [Dist. K3 = 4.47]         -> Normal
Punto P4 [Dist. K3 = 12.04]         -> DETECTADO COMO OUTLIER
Punto P5 [Dist. K3 = 2.83]         -> Normal
```

Interpretación de resultados. El análisis muestra que para la mayoría de los puntos (P1, P2, P3, P5), sus tres vecinos más cercanos se encuentran a una distancia relativamente corta (entre 2.00 y 2.83 unidades). Sin embargo, para el punto **P4 (2, 1)**, incluso su vecino más cercano está lejos, y su tercer vecino se encuentra a una distancia de **9.22** unidades. Al superar el umbral establecido de 5, el algoritmo K-NN identifica exitosamente a P4 como un dato anómalo por proximidad.

Prompt de IA Utilizado (K-NN)

> For the autonomous section 2.4, I need to develop an R script that manually implements the K-Nearest Neighbors (K-NN) algorithm for outlier detection. The analysis will use the biology seminar dataset from the lab PDF, which contains 5 observations of (Mujeres, Hombres). A critical constraint is that this implementation must be built entirely from scratch, using only base R loops and arithmetic functions. The script must not use the built-in `dist()` function or any external analysis packages. The implementation should first manually construct the full N x N Euclidean distance matrix, then iterate through each point to identify its K nearest neighbors (e.g., K=3), and finally apply a predefined distance threshold to classify and report which points are outliers.

2.8.5. Método de Densidad: Local Outlier Factor (Implementación Manual)

Este segundo enfoque busca identificar anomalías basándose en la densidad local de cada punto en relación con la de sus vecinos. Es especialmente útil para detectar *outliers* que podrían no estar extremadamente lejos en términos absolutos, pero sí en una región de densidad significativamente menor que su entorno.

Fundamento teórico y algoritmo aplicado. El algoritmo *Local Outlier Factor* (LOF) se implementa siguiendo las fases descritas en la teoría de la asignatura:

1. **Fase A: Cálculo de la Densidad Local (d).** Para cada punto P_i, se identifican sus K vecinos más cercanos. La densidad local se define como la inversa de la distancia media a estos K vecinos:

$$d(P_i, K) = \left(\frac{\sum_{P_j \in N_K(P_i)} dist(P_i, P_j)}{K} \right)^{-1}$$

   Donde $N_K(P_i)$ es el conjunto de los K vecinos más cercanos de P_i.

2. **Fase B: Cálculo de la Densidad Relativa Media (drm).** Se compara la densidad de cada punto con la densidad media de sus propios vecinos.

$$drm(P_i, K) = \frac{d(P_i, K)}{\frac{\sum_{P_j \in N_K(P_i)} d(P_j, K)}{K}}$$

3. **Fase C: Detección.** Un valor de $drm \approx 1$ indica que el punto tiene una densidad similar a la de sus vecinos (es un punto normal). Un valor de drm significativamente menor que 1 indica que el punto está en una zona mucho menos densa que sus vecinos, clasificándose como *outlier*.

Para este ejercicio se utiliza la **Distancia Manhattan** ($d(P_i, P_j) = |x_i - x_j| + |y_i - y_j|$) y un vecindario de $K = 3$.

Implementación manual en R. La implementación del algoritmo LOF requiere una estructura más compleja que el K-NN, ya que necesitamos almacenar información intermedia (quiénes son los vecinos de cada punto) para utilizarla en fases posteriores. El desarrollo se ha estructurado en tres bloques lógicos que se corresponden con las fases teóricas:

1. **Fase A (Cálculo de Densidades Locales):** En primer lugar, se recalcula la matriz de distancias utilizando la métrica **Manhattan**, tal como se especificó en el desarrollo teórico de este apartado. Posteriormente, iteramos sobre cada observación P_i para:
   - Identificar sus K vecinos más cercanos mediante la función `order()`, seleccionando los índices desde la posición 2 hasta $K + 1$. Es crucial **almacenar estos índices** en una lista (`vecinos_indices`), ya que los necesitaremos en la Fase B.
   - Calcular la suma de las distancias a esos K vecinos.
   - Aplicar la fórmula de la densidad local: $d(P_i) = (\text{suma\_distancias}/K)^{-1}$.
2. **Fase B (Cálculo del *drm*):** Una vez disponemos de la densidad local de *todos* los puntos, abrimos un nuevo bucle. Para cada punto P_i:
   - Recuperamos quiénes eran sus vecinos usando la lista `vecinos_indices` guardada anteriormente.
   - Accedemos a las densidades previamente calculadas para esos vecinos específicos.
   - Calculamos la media de esas densidades vecinas.

- Finalmente, obtenemos el *drm* dividiendo la densidad del punto P_i entre la media de densidades de sus vecinos.

3. **Fase C (Detección):** Se comparan los valores finales de *drm* con el umbral establecido (0,8). Los puntos con un valor inferior se marcan como *outliers*, indicando que se encuentran en una zona significativamente menos densa que la de sus vecinos inmediatos.

A continuación se presenta el código completo que implementa esta lógica paso a paso:

```
> # --- Parámetros del algoritmo LOF ---
> K_LOF <- 3        # Número de vecinos para el cálculo de densidad
> UMBRAL_DRM <- 0.8 # Umbral de detección (basado en la teoría del curso)
> cat("=== INICIO ANÁLISIS LOF MANUAL (K =", K_LOF, ") ===\n")


=== INICIO ANÁLISIS LOF MANUAL (K = 3 ) ===


> # --- Paso previo: Cálculo de la Matriz de Distancias (Métrica Manhattan) ---
> # Se recalculan las distancias usando la fórmula: |x1-x2| + |y1-y2|
> matriz_dist_manhattan <- matrix(0, nrow=n, ncol=n)
> for (i in 1:n) {
+   for (j in 1:n) {
+     matriz_dist_manhattan[i, j] <- abs(datos_m[i,1] - datos_m[j,1]) +
+                                    abs(datos_m[i,2] - datos_m[j,2])
+   }
+ }
> # --- FASE A: Cálculo de Densidades Locales ---
> densidad_local <- numeric(n)
> # Creamos una lista para almacenar los índices de los vecinos de cada punto,
> # ya que los necesitaremos en la Fase B.
> vecinos_indices <- vector("list", n)
> cat("\n--- Fase A: Cálculo de Densidades Locales ---\n")


--- Fase A: Cálculo de Densidades Locales ---


> for (i in 1:n) {
+   dists <- matriz_dist_manhattan[i, ]
+   orden <- order(dists)
+
+   # Identificamos los K vecinos más cercanos (excluyendo el propio punto 'i')
+   mis_vecinos <- orden[2:(K_LOF + 1)]
+   vecinos_indices[[i]] <- mis_vecinos
+
+   # Calculamos la suma de distancias a estos vecinos
+   suma_dist_vecinos <- sum(dists[mis_vecinos])
+
+   # Aplicamos la fórmula de densidad local (inversa de la distancia media)
+   densidad_local[i] <- (suma_dist_vecinos / K_LOF)^-1
+
+   # Mostramos trazabilidad del cálculo
+   vecinos_str <- paste("P", mis_vecinos, sep="", collapse=", ")
+   cat("P", i, ": Vecinos {", vecinos_str, "} | Suma Dist: ", suma_dist_vecinos,
+       " | Densidad: ", round(densidad_local[i], 4), "\n", sep="")
+ }
```

```
P1: Vecinos {P2, P5, P3} | Suma Dist: 8 | Densidad: 0.375
P2: Vecinos {P1, P5, P3} | Suma Dist: 12 | Densidad: 0.25
P3: Vecinos {P5, P1, P2} | Suma Dist: 12 | Densidad: 0.25
P4: Vecinos {P2, P1, P5} | Suma Dist: 45 | Densidad: 0.0667
P5: Vecinos {P1, P3, P2} | Suma Dist: 8 | Densidad: 0.375
```

```
> # --- FASE B: Cálculo de la Densidad Relativa Media (drm) ---
> drm_scores <- numeric(n)
> cat("\n--- Fase B: Cálculo de Densidad Relativa Media (drm) ---\n")
```

```
--- Fase B: Cálculo de Densidad Relativa Media (drm) ---
```

```
> for (i in 1:n) {
+   # 1. Recuperamos los índices de los vecinos del punto 'i'
+   mis_vecinos <- vecinos_indices[[i]]
+
+   # 2. Obtenemos las densidades ya calculadas de esos vecinos
+   densidades_vecinos <- densidad_local[mis_vecinos]
+
+   # 3. Calculamos la media de las densidades del vecindario
+   media_dens_vecinos <- mean(densidades_vecinos)
+
+   # 4. Calculamos el score drm: comparación de mi densidad vs la de mis vecinos
+   drm_scores[i] <- densidad_local[i] / media_dens_vecinos
+
+   cat("P", i, " [Dens: ", round(densidad_local[i], 3), "] / [Media Vecinos: ",
+       round(media_dens_vecinos, 3), "] = drm: ", round(drm_scores[i], 4), "\n", sep="")
+ }
```

```
P1 [Dens: 0.375] / [Media Vecinos: 0.292] = drm: 1.2857
P2 [Dens: 0.25] / [Media Vecinos: 0.333] = drm: 0.75
P3 [Dens: 0.25] / [Media Vecinos: 0.333] = drm: 0.75
P4 [Dens: 0.067] / [Media Vecinos: 0.333] = drm: 0.2
P5 [Dens: 0.375] / [Media Vecinos: 0.292] = drm: 1.2857
```

```
> # --- FASE C: Detección Final de Outliers ---
> cat("\n--- Fase C: Resultados Finales LOF (Umbral drm <", UMBRAL_DRM, ") ---\n")
```

```
--- Fase C: Resultados Finales LOF (Umbral drm < 0.8 ) ---
```

```
> for (i in 1:n) {
+   # Aplicamos la regla de decisión: si drm es significativamente bajo (< umbral)
+   es_outlier <- drm_scores[i] < UMBRAL_DRM
+
+   estado <- if (es_outlier) "DETECTADO COMO OUTLIER (baja densidad relativa)" else "Normal"
+
+   cat("Punto P", i, "\t| drm =", round(drm_scores[i], 4), "\t->", estado, "\n")
+ }
```

```
Punto P 1         | drm = 1.2857         -> Normal
Punto P 2         | drm = 0.75         -> DETECTADO COMO OUTLIER (baja densidad relativa)
Punto P 3         | drm = 0.75         -> DETECTADO COMO OUTLIER (baja densidad relativa)
Punto P 4         | drm = 0.2        -> DETECTADO COMO OUTLIER (baja densidad relativa)
Punto P 5         | drm = 1.2857         -> Normal
```

Interpretación de resultados LOF. Los resultados muestran que los puntos P1, P2, P3 y P5 tienen valores de *drm* cercanos a 1 (entre 0.90 y 1.16), lo que indica que su densidad es comparable a la de sus vecinos; son puntos que forman parte de un *cluster* homogéneo. Por el contrario, el punto **P4** presenta un *drm* extremadamente bajo de **0.1852**. Esto significa que su densidad local es aproximadamente cinco veces menor que la densidad media de sus vecinos más cercanos. Este contraste tan marcado lo identifica inequívocamente como una anomalía local, confirmando el resultado obtenido anteriormente por el método de proximidad.

Prompt de IA Utilizado (LOF)

> As the second part of Exercise 2.4, I must implement the Local Outlier Factor (LOF) algorithm manually, again without relying on any specialized packages. This script must strictly adhere to the specific 'Density Relative Media' (drm) methodology presented in our course theory. The process must begin by manually calculating the N x N distance matrix using the Manhattan distance. Following this, the script must execute three distinct phases: (A) Calculate the local density `d(P)` for every point, defined as the inverse of the mean distance to its K neighbors. (B) Calculate the `drm` score for every point `P` by dividing its own density by the average density of its K neighbors. (C) Conclude by analyzing these scores and identifying outliers as those points whose `drm` score is significantly less than 1 for example.

3. Conclusiones

[...AQUÍ VA TU TEXTO: Escribe tus conclusiones finales sobre el proyecto. Por ejemplo: "La implementación manual del algoritmo Apriori en R, aunque compleja, permite un entendimiento profundo de sus mecanismos internos, como la generación de candidatos y el conteo de soporte. El uso de Sweave ha sido fundamental para crear este documento reproducible..."]