



FUNDAMENTOS DE LA CIENCIA DE DATOS

---

## Prueba de Laboratorio 1 (PL1)

---

**Jorge Revenga Martín de Vidales**  
**Ángel Salgado Aldao**  
**Adrián García**

Grado en Ingeniería Informática  
Universidad de Alcalá

13 de noviembre de 2023

# Índice

<b>1. Introducción - Consideraciones previas</b>	<b>2</b>
1.1. Funciones básicas . . . . .	2
1.2. Introducción de un archivo .txt en R . . . . .	3
1.3. Uso de Sweave . . . . .	3
1.4. Paquetes por defecto al abrir R . . . . .	3
1.5. Instalar paquetes nuevos . . . . .	4
<b>2. Ejercicios con ayuda del profesor</b>	<b>4</b>
2.1. Análisis de descripción de datos . . . . .	4
2.2. Análisis de asociación . . . . .	9
2.3. Análisis de detección de datos anómalos . . . . .	11
2.3.1. Técnicas con base estadística . . . . .	11
2.3.2. Técnicas basadas en la proximidad y en la densidad . . . . .	12
<b>3. Ejercicios de forma autónoma</b>	<b>12</b>
3.1. Análisis de descripción de datos . . . . .	13
3.2. Análisis de asociación . . . . .	15
3.3. Análisis de detección de datos anómalos . . . . .	15
3.3.1. Técnicas con base estadística . . . . .	15
3.3.2. Técnicas basadas en la proximidad y en la densidad . . . . .	17

# 1. Introducción - Consideraciones previas

## 1.1. Funciones básicas

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

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

En el archivo HTML se distinguen varios elementos:

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

## 1.2. Introducción de un archivo .txt en R

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

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

## 1.3. Uso de Sweave

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

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

## 1.4. Paquetes por defecto al abrir R

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

## 1.5. Instalar paquetes nuevos

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

## 2. Ejercicios con ayuda del profesor

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

### 2.1. Análisis de descripción de datos

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

#### Solución:

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

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

	Nombre	Radio
1	Cordelia	13
2	Ofelia	16
3	Bianca	22

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

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

```
> dim(s)
```

```
[1] 12 2
```

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

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

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

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

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

	Nombre	Radio
6	Julietta	42
8	Belinda	34
4	Crésida	33

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

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

```
> length(s$Radio)
```

```
[1] 12
```

- Para realizar el cálculo del rango:

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

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

```
[1] 29
```

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

```
> range(s$Radio)
```

```
[1] 13 42
```

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

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

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

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

```
[1] 29
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

```
[1] 25.08333
```

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

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

```
[1] 8.857029
```

- `sdr=sqrt((sdr2)*11/12)`: creamos la función, quitándole la raíz cuadrada con el <sup>2</sup> para multiplicar después por  $N-1/N$ , para después añadirle la raíz cuadrada de nuevo. Este código puede ser fácilmente optimizado y definido en una función para su reutilización

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

```
[1] 8.47996
```

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

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

```
[1] 78.44697
```

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

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

```
[1] 71.90972
```

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

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

```
[1] 24.5
```

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

```
> cuart1=quantile(x,0.25)
> cuart1
```

```
25%
19
```

## 2.2. Análisis de asociación

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

### Solución

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

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

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

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

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

most frequent items:

Pan	Leche	Agua	Café	Naranjas	(Other)
5	5	4	2	1	0

element (itemset/transaction) length distribution:

sizes

1 2 3 4

1 1 2 2

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

includes extended item information - examples:

labels

```
1 Pan
2 Agua
3 Café
```

includes extended transaction information - examples:

itemsetID

```
1 suceso1
2 suceso2
3 suceso3
```

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

Apriori

Parameter specification:

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

Algorithmic control:

```
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
```

Absolute minimum support count: 3

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

```
> inspect(asociaciones)
```

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

## 2.3. Análisis de detección de datos anómalos

Esto es un poco más de texto en el párrafo.

### 2.3.1. Técnicas con base estadística

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

#### Caja y Bigotes

- (muestra=t(matrix(c(3,2,3.5,12,4.7,4.1,5.2,4.9,7.1,6.1,6.2,5.2,14,5.3),2,7,dimnames=list(c("resistencia","densidad")),data.frame(muestra))):
- (boxplot(muestra\$r,range=1.5,plot=FALSE)): Boxplot de forma predeterminada muestra gráficamente la resolución con caja y bigotes, por lo que se añade plot=FALSE para que no lo haga
- (cuar1r<-quantile(muestra\$r, 0.25)): Cálculo del primer cuartil
- (cuar3r<-quantile(muestra\$r, 0.75)) Cálculo del tercer cuartil

- `(int=c(cuar1r-1.5*(cuar3r-cuar1r),cuar3r+1.5*(cuar3r-cuar1r))):` calcula el rango
- `for(i in 1:length(muestra$r)) if(muestra$r[i]<int[1] || muestra$r[i]>int[2])print(suceso"); print(i); print("es un suceso anómalo o outlier")`

### Desviación típica

- `sdd=sqrt(var(muestra$d)*length(muestra$d)-1)/length(muestra$d)`
- `(intdes=c(mean(muestra$d)-2*sdd,mean(muestra$d)+2*sdd)):`
- `for(i in 1:length(muestra$d)) if (muestra$d[i]<intdes[1] || muestra$d[i]>intdes[2]) print(suceso");print(i);print(muestra$d[i]);print("es un suceso anómalo o outlier")`

### 2.3.2. Técnicas basadas en la proximidad y en la densidad

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

### Vecinos próximos

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

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

## 3. Ejercicios de forma autónoma

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

### 3.1. Análisis de descripción de datos

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

#### Solución:

- `s<-read.table("distancias.txt")`: Se asigna los valores de la tabla distancias (almacenada en el directorio de trabajo) a la variable "s". Al teclear introducir el nombre de la variable como comando se deberían mostrar los datos
- Cálculo del rango con la función: `ranger(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de números.
  - Retorno: Devuelve el rango del vector introducido, es decir, la diferencia entre el número mayor y menor de dicho vector.
  - Explicación: Esta función se encarga de calcular el rango del vector introducido por parámetro. Va recorriendo el vector y durante cada iteración va comparando el valor actual con el máximo y el mínimo actuales, si dicho valor es menor o mayor que el mínimo o máximo respectivamente, actualiza el valor correspondiente. Una vez finalizado el recorrido calcula la diferencia entre el máximo y el mínimo final.
- Cálculo de la frecuencia absoluta ( $f_i$ ) con la función: `frecAbsr(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de datos.
  - Retorno: Devuelve un dataframe con el número de apariciones totales de cada dato del vector.
  - Explicación: Esta función se encarga de contar el número de apariciones de cada elemento del vector. Recorre el vector incrementando en 1 la frecuencia de un elemento cada vez que se repita dicho elemento.
- Cálculo de la frecuencia absoluta acumulada ( $fa_i$ ) con la función: `frecAbsAcumr(distancias$Km)`
  - Parámetros:
    - `vector`: Vector de datos.
  - Retorno: Devuelve un dataframe con el número de apariciones totales acumuladas de cada dato del vector.
  - Explicación: Esta función se encarga de calcular las frecuencias acumuladas de cada dato del vector. Calcula primero las frecuencias absolutas con `frecAbsr(vector)` y recorre el dataframe resultado, sumando a la frecuencia absoluta del dato actual, la frecuencia absoluta acumulada del dato anterior.

- Cálculo de la frecuencia relativa ( $fr_i$ ) con la función: `frecRelr(distancias$Km)`
  - Parámetros:
    - **vector**: Vector de datos.
  - Retorno: Devuelve un dataframe con el número de apariciones totales de cada dato en relación al número de elementos del vector.
  - Explicación: Esta función se encarga de calcular las frecuencias relativas de cada dato del vector. Calcula previamente las frecuencias absolutas y después recorre el dataframe resultado, dividiendo la frecuencia absoluta de cada dato entre el número total de elementos, de la forma  $f_i/\text{length}(\text{vector})$
- Cálculo de la frecuencia relativa acumulada ( $fra_i$ ) con la función: `frecRelAcumr(distancias$Km)`
  - Parámetros:
    - **vector**: Vector de datos.
  - Retorno: Devuelve un dataframe con las frecuencias relativas acumuladas de cada dato del vector.
  - Explicación: Esta función se encarga de calcular las frecuencias relativas acumuladas de cada dato del vector. Calcula primero las frecuencias relativas con `frecRelr(vector)` y recorre el dataframe resultado, sumando a la frecuencia relativa del dato actual, la frecuencia relativa acumulada del dato anterior.
- Cálculo de la media aritmética ( $\bar{x}_a$ ) con la función: `meanr(distancias$Km)`
  - Parámetros:
    - **vector**: Vector de números.
  - Retorno: Devuelve un número que representa la media aritmética
  - Explicación: Esta función se encarga de calcular la media aritmética, recorriendo el vector sumando todos los valores y después dividiéndolos entre el número total de elementos del vector, de la forma:
 
$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$
- Cálculo de la desviación típica ( $s$ ) con la función: `sdr(distancias$Km)`
  - Parámetros:
    - **vector**: Vector de números.
  - Retorno: Devuelve un número que representa la desviación típica.
  - Explicación: Esta función se encarga de calcular la desviación típica. Primero calcula la media con la función `meanr(vector)`, después va recorriendo el vector calculando la suma de cuadrados entre la diferencia de cada valor y la media, lo divide todo entre el número total de elementos del vector y por último realiza la raíz cuadrada de dicho resultado, de la forma:
 
$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$
- Cálculo de la varianza ( $s^2$ ) con la función: `varr(distancias$Km)`
  - Parámetros:

- **vector**: Vector de números.
- Retorno: Devuelve un número que representa la varianza.
- Explicación: Esta función se encarga de calcular la varianza. Primero calcula la desviación típica con la función `sdr(vector)` y después eleva al cuadrado dicho resultado, de la forma:
 
$$s^2 = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$
- Cálculo de la mediana ( $\tilde{x}$ ) con la función: `medianr(distancias$Km)`
  - Parámetros:
    - **vector**: Vector de números.
  - Retorno: Devuelve un número que representa la mediana.
  - Explicación: Esta función se encarga de calcular la mediana. Primero ordena de forma creciente el vector con la función `sort(vector)` y calcula la longitud del mismo, dependiendo de si la longitud es par o impar, calcula la mediana de la forma:
 
$$\tilde{x} = \begin{cases} x_{\frac{n+1}{2}} & \text{si } n \text{ es impar} \\ \frac{1}{2}(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}) & \text{sin es par} \end{cases}$$
- Cálculo de los cuartiles ( $Q_i$ ) con la función: `quantilerr(distancias$Km,p)`
  - Parámetros:
    - **vector**: Vector de números.
    - **p**: Número decimal del cuantil.
  - Retorno: Devuelve un número que representa el cuantil de p.
  - Explicación: Esta función se encarga de calcular el cuantil. Primero ordena de forma creciente el vector con la función `sort(vector)`, después calcula la posición del cuantil, de la forma:
 
$$Q_p = \begin{cases} x_{\frac{np}{2}} + x_{\frac{np+1}{2}} & \text{si } np \in \mathbb{N} \\ x_{[np]+1} & \text{si } np \notin \mathbb{N} \end{cases}$$

### 3.2. Análisis de asociación

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

### 3.3. Análisis de detección de datos anómalos

#### 3.3.1. Técnicas con base estadística

El tercer conjunto de datos, que se empleará para realizar el análisis de detección de datos anómalos utilizando técnicas con base estadística, estará formado por los siguientes 10 valores de velocidades de respuesta y temperaturas normalizadas de un microprocesador



Velocidad, Temperatura: 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. Aplicar las medidas de ordenación a la velocidad y las de dispersión a la temperatura.

### Solución:

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

Todo esto está implementado en la función: `cajaBigotes(matriz,valor,d)`.

- Parámetros:

- **matriz**: Una matriz de datos numéricos.
- **valor**: El nombre de la medida (fila de la matriz) sobre la que queremos realizar el análisis.
- **d**: Grado de outlier.

- Retorno: Imprime por pantalla todos aquellos valores que queden fuera de los límites establecidos, es decir, los outliers.

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

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

Todo esto está implementado en la función: `desvTip(matriz,valor,d)`.

- Parámetros:

- **matriz**: Una matriz de datos numéricos.
- **valor**: El nombre de la medida (fila de la matriz) sobre la que queremos realizar el análisis.
- **d**: Grado de outlier.

- Retorno: Imprime por pantalla todos aquellos valores que queden fuera de los límites establecidos, es decir, los outliers.

- Explicación: Esta función calcula los outliers sobre una serie de valores. Recibe el grado de outlier (**d**) y una matriz (**matriz**) con dichos valores y lo primero que hace es crear un dataframe de la traspuesta de dicha matriz, para facilitar su tratamiento. Después a partir de **valor**, obtiene la fila de la matriz que

contiene los valores que vamos a evaluar. Una vez hecho esto hace uso de las funciones `meanr()` y `sdr()`, las cuales hemos implementado previamente, para calcular la media  $\bar{x}_a$  y la desviación típica  $s$  respectivamente. Por último calcula los límites a partir de,  $(\bar{x}_a - d \cdot s, \bar{x}_a + d \cdot s)$  e imprimirá por pantalla todos aquellos valores que se encuentren fuera de estos límites establecidos, ya que, serán considerados outliers.

### 3.3.2. Técnicas basadas en la proximidad y en la densidad

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

#### Solución:

- Técnicas basadas en proximidad (k-Vecinos): Para este análisis de detección de outliers, se va a hacer uso de la técnica k-vecinos. Esta técnica consiste en buscar sucesos muy separados al resto en función de sus distancias. Para ello se calculan las distancias euclídeas entre todos los puntos, para posteriormente ordenar estas distancias de forma creciente, solo se ordenará el número de distancias establecido por **k**. Por último se comprueba si el vecino **k** supera el grado de outlier establecido por **d**, si lo hace, será considerado outlier.
  - Parámetros:
    - **matriz**: Matriz de valores numéricos.
    - **k**: Número de orden k.
    - **d**: Grado de outlier.
  - Retorno: Imprime por pantalla todos aquellos sucesos cuyo **k**, se encuentre a una distancia mayor que el grado de outlier **d**, es decir, los outliers.
  - Explicación: Esta función calcula los outliers sobre una serie de sucesos. Recibe el número de orden **k**, el grado de outlier (**d**) y una matriz (**matriz**) con dichos sucesos. El primer paso es calcular las distancias euclídeas de la forma,  $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ , a partir de la función `distEuc()`. Después se ordenan dichas distancias de menor a mayor. Por último se verifica dentro de las distancias ordenadas, si el vecino **k** de cada punto, supera el valor establecido inicialmente por **d**, si esto sucede se imprime dicho suceso por pantalla, ya que, será considerado outlier.
- Técnicas basadas en densidad (Local Outlier Factor):