Minería de datos: Aprendizaje no supervisado y detección de anomalías

Práctica final: Detección de anomalías

Antonio Manuel Milán Jiménez

7 de Febrero de 2019

Contents

Introducción

Las anomalías se definen como datos con uno o más valores inusuales que obligan a que sean tratados de manera especial en Ciencia de datos. Así, en este trabajo se estudiarán diferentes métodos para detectar estas anomalías u 'outliers' en un conjunto de datos, concretamente en el dataset 'Glass'. Este dataset trata sobre el estudio de diferentes tipos de cristales aparecido en escenas de crímenes que son utilizados en criminología como evidencias. Así, se tienen 214 observaciones y 9 atributos de acuerdo a diferentes características de estos cristales.

```
mydata <- readMat("./glass.mat")
mydata <- as.data.frame(mydata)
#Eliminamos la última variable pues no es predictora
mydata <- mydata[,1:9]
colnames(mydata) <- c("RI","Na","Mg","Al","Si","K","Ca","Ba","Fe")
```

Outliers univariantes

Estos datos son considerados outliers univariantes por tener algún valor extremo en alguna de sus columnas. Empezamos buscando en alguna columna al azar aunque posteriormente se hará para todas las variables. Lo primero es determinar la columna y escalar los datos:

```
mydata.scaled = scale(mydata)
columna         = mydata[, 1]
nombre.columna  = names(mydata)[1]
columna.scaled  = mydata.scaled[, 1]
```

Utilizamos la primera columna a modo de ejemplo para calcular los outliers según la regla IQR, calculando nosotros los cuantiles:

```
primerCuartil = quantile(columna.scaled, 0.25)
mediana = quantile(columna.scaled, 0.5)
tercerCuartil = quantile(columna.scaled, 0.75)
iqr = IQR(columna.scaled)
```

Y así calcular los umbrales que determinan si el dato es un outlier e incluso un outlier extremo:

```
extremo.superior.outlier.normal  = tercerCuartil + 1.5 * iqr
extremo.inferior.outlier.normal  = primerCuartil - 1.5 * iqr
extremo.superior.outlier.extremo = tercerCuartil + 3 * iqr
extremo.inferior.outlier.extremo = primerCuartil - 3 * iqr

vector.es.outlier.normal = columna.scaled > extremo.superior.outlier.normal | columna.scaled <
  extremo.inferior.outlier.normal
vector.es.outlier.extremo = columna.scaled < extremo.inferior.outlier.extremo | columna.scaled >
  extremo.superior.outlier.extremo
```

Estos son los outliers normales detectados junto con sus valores:

```
claves.outliers.normales = which(vector.es.outlier.normal == TRUE)
claves.outliers.normales
```

```
##  [1]  48  51  57 104 105 106 107 108 111 112 113 132 171 185 186 188 190
```

```
valores.outliers.normales = mydata[claves.outliers.normales,1]
valores.outliers.normales
```

```
##  [1] 1.52667 1.52320 1.51215 1.52725 1.52410 1.52475 1.53125 1.53393
##  [9] 1.52664 1.52739 1.52777 1.52614 1.52369 1.51115 1.51131 1.52315
## [17] 1.52365
```

```
length(claves.outliers.normales)
```

```
## [1] 17
```

Observamos que para la primera variable se han detectado 17 outliers normales debido a sus valores.

Estudiando ahora los outliers extremos:

```
claves.outliers.extremos = which(vector.es.outlier.extremo == TRUE)
claves.outliers.extremos
```

```
## [1] 104 107 108 112 113
```

```
valores.outliers.extremos = mydata[claves.outliers.extremos,1]
valores.outliers.extremos
```

```
## [1] 1.52725 1.53125 1.53393 1.52739 1.52777
```

```
length(claves.outliers.extremos)
```

```
## [1] 5
```

Como era de esperar, se obtienen considerablemente menos outliers. Se obtienen 5 outliers por sus valores muy extremos en esta primera variable.

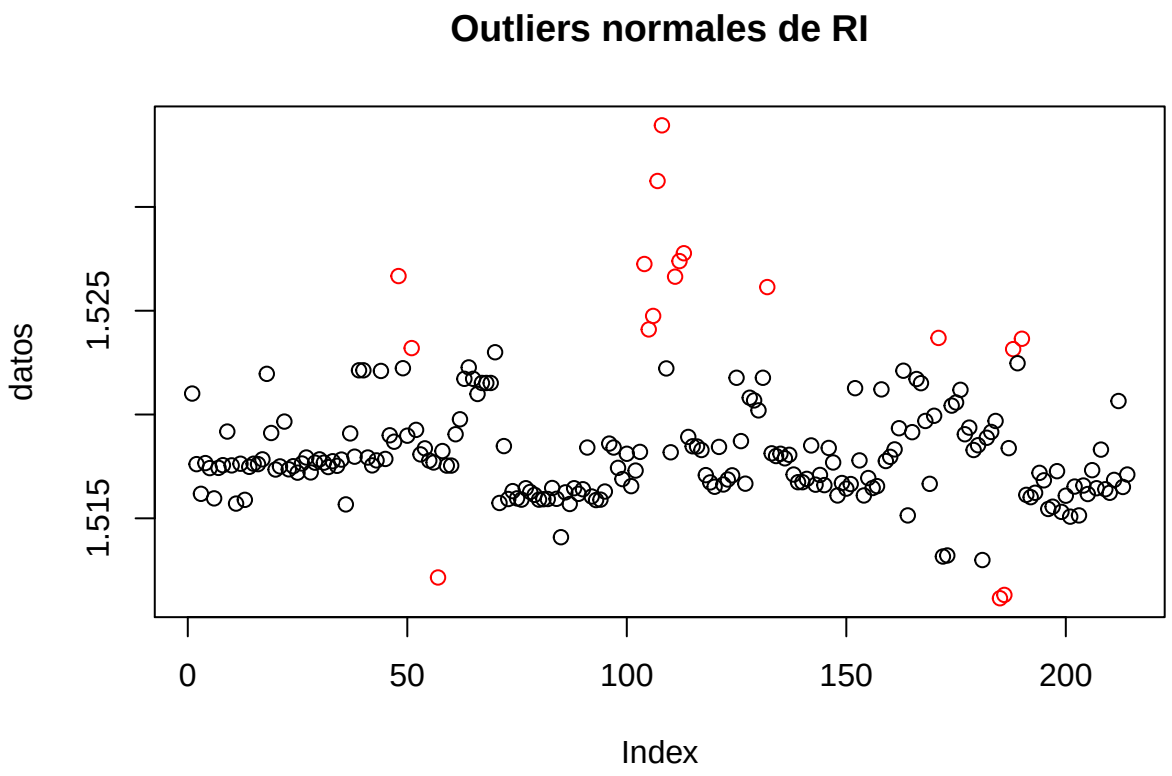
Podemos también saber los valores normalizados de los outliers:

```
valores.normalizados.outliers.normales = columna.scaled[claves.outliers.normales]
valores.normalizados.outliers.normales
```

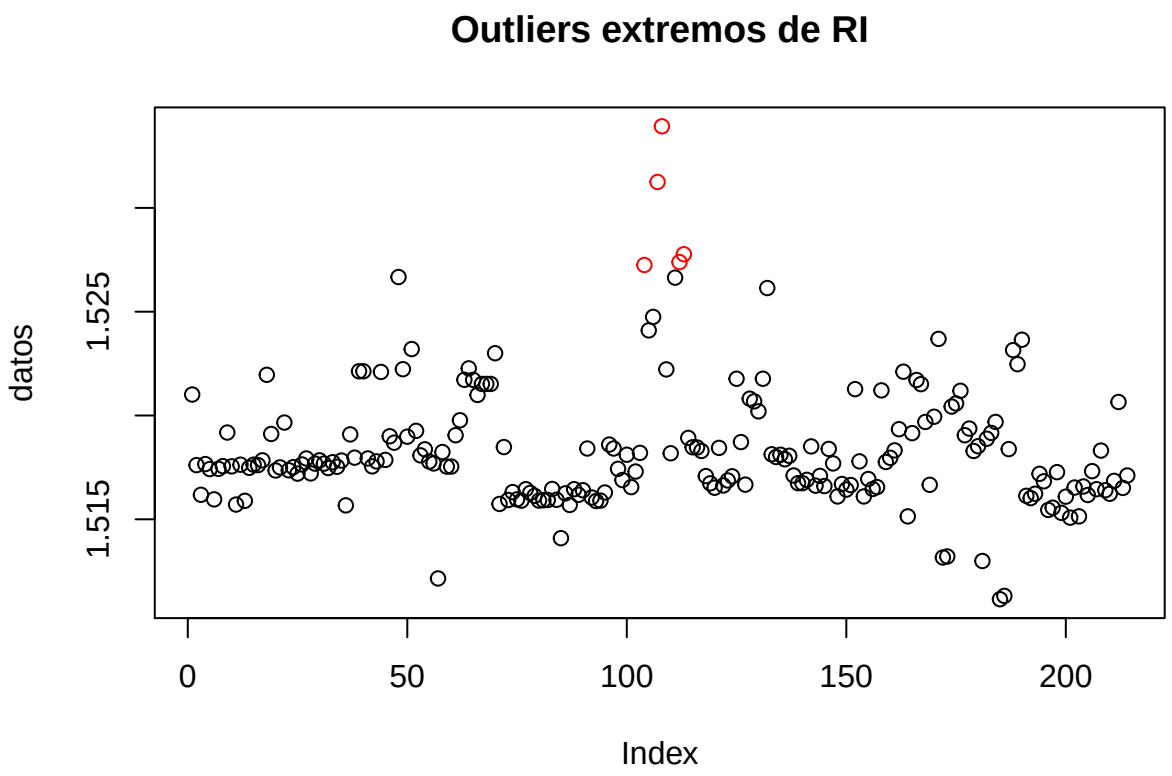
```
##  [1]  2.734591  1.591965 -2.046658  2.925577  1.888323  2.102360  4.242726
##  [8]  5.125215  2.724712  2.971677  3.096807  2.560069  1.753315 -2.375945
## [15] -2.323259  1.575500  1.740144
```

Ahora vamos a representar gráficamente estos outliers:

```
MiPlot_Univariate_Outliers(columna,claves.outliers.normales,"Outliers normales de RI")
```



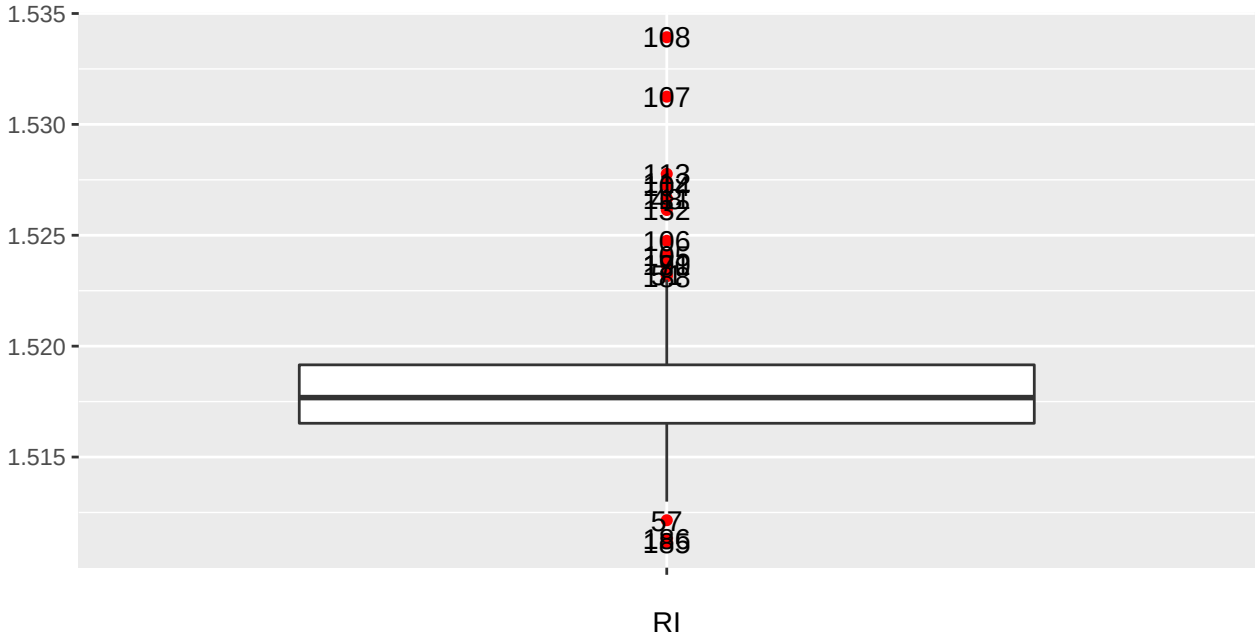

```
MiPlot_Univariate_Outliers(columna,claves.outliers.extremos,"Outliers extremos de RI")
```



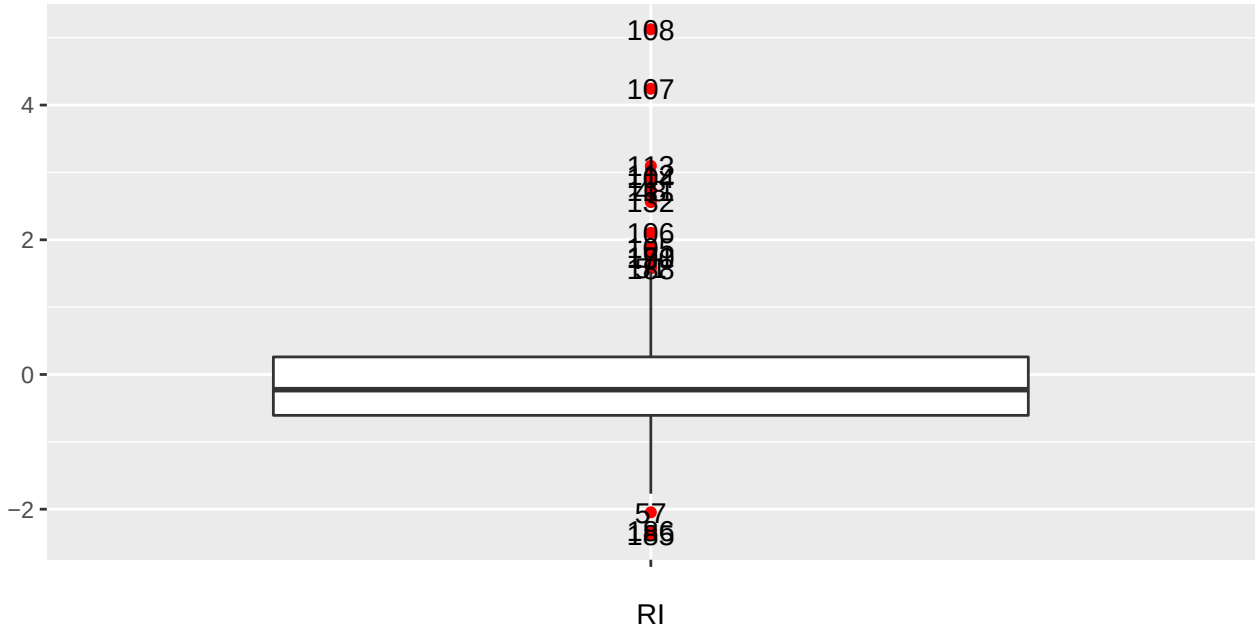
Comprobamos que efectivamente, al estar considerando una única variables, los outliers detectados se corresponden a los datos más extremos en los gráficos.

Ahora dibujamos un "boxplot" tanto con datos normalizados como sin normalizar:

```
MiBoxPlot_IQR_Univariate_Outliers(mydata,1,1.5)
```




```
MiBoxPlot_IQR_Univariate_Outliers(mydata.scaled,1,1.5)
```



Descubrimos que los boxplot son iguales pues, aunque la normalización afecta a los valores de los datos, no afecta a sus distribuciones.

A continuación vamos a calcular los mismos outliers utilizando las funciones proporcionadas "vector_es_outlier_IQR" y "vector_claves_outliers_IQR". En función de si queremos encontrar outliers normales o extremos, cambiaremos el coeficiente en la función a 1.5 o 3 respectivamente:

```
head(vector_es_outlier_IQR(mydata,1,1.5),20)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
vector_claves_outliers_IQR(mydata,1,1.5)
```

```
##  [1]  48  51  57 104 105 106 107 108 111 112 113 132 171 185 186 188 190
```

```
head(vector_es_outlier_IQR(mydata,1,3),20)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

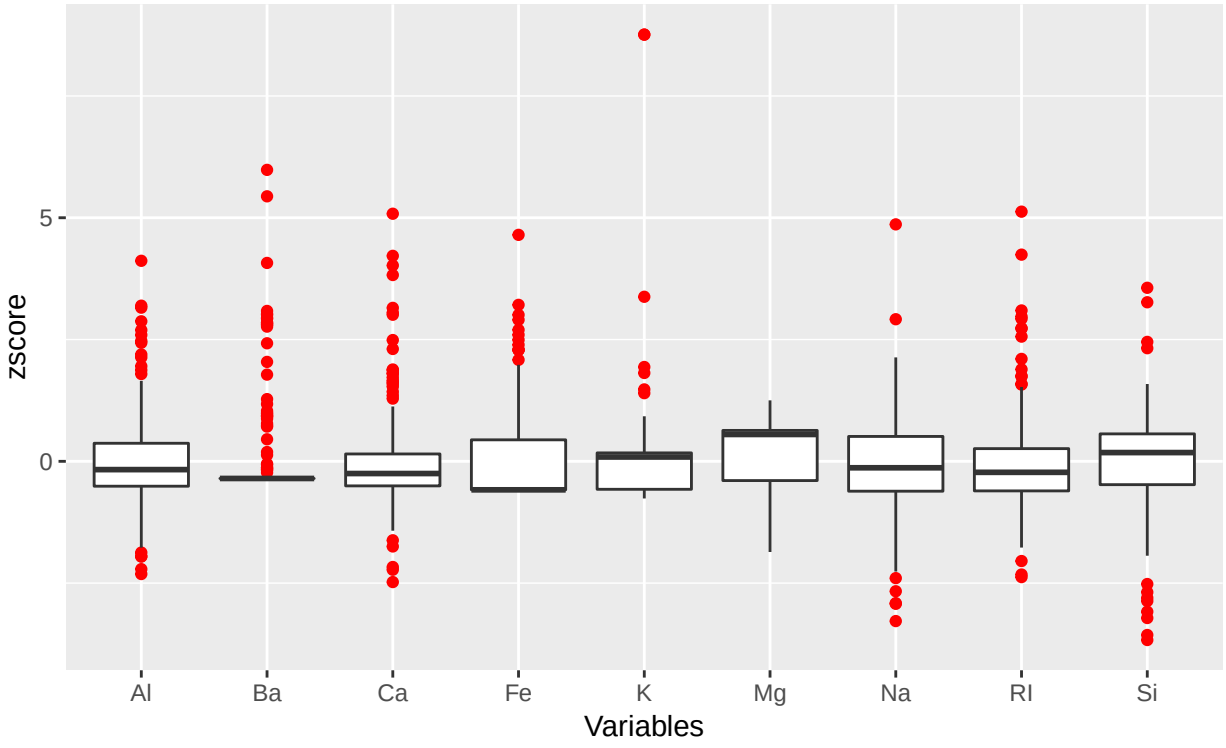
```
vector_claves_outliers_IQR(mydata,1,3)
```

```
## [1] 104 107 108 112 113
```

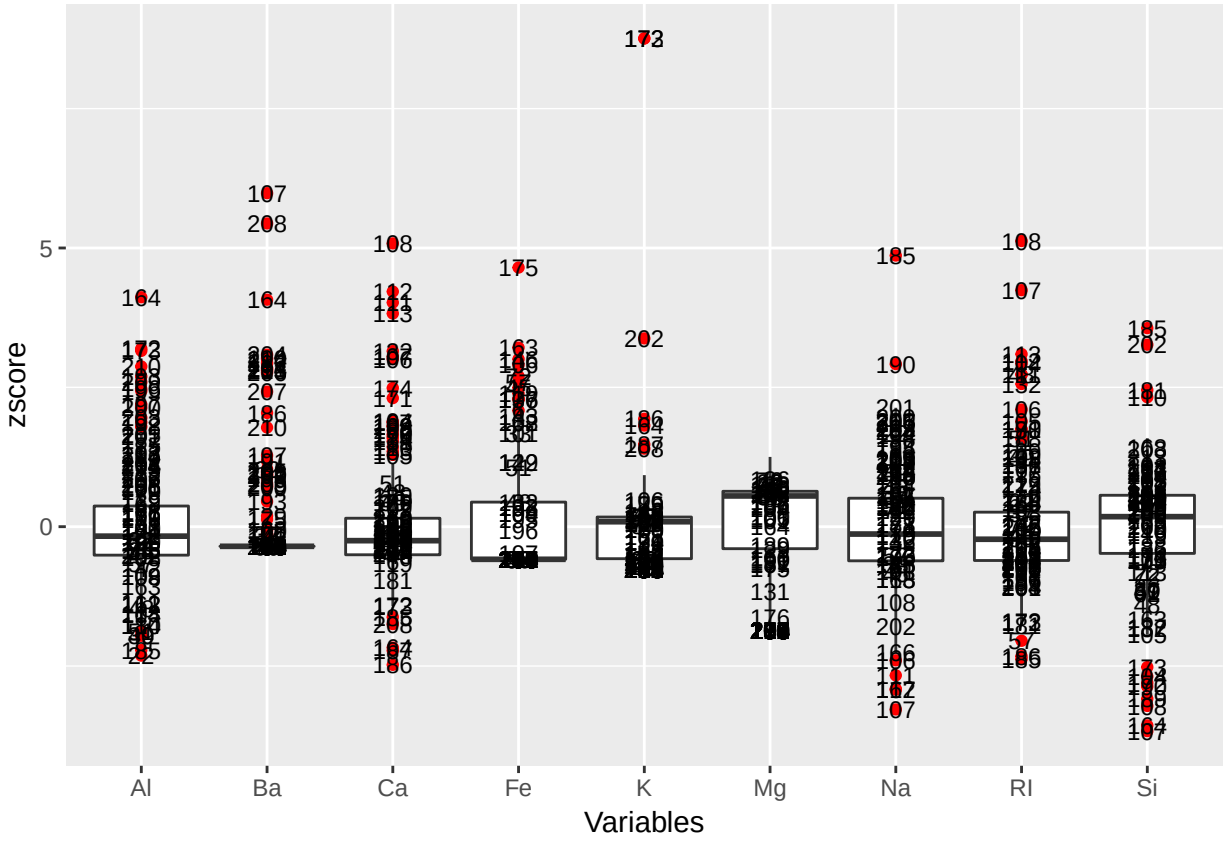
Efectivamente son los mismos outliers que ya habíamos calculado.

Ahora vamos a utilizar las funciones proporcionadas "MiBoxPlot_juntos" y "MiBoxPlot_juntos_con_etiquetas" que dibujarán el "boxplot" para todas las variables con lo que podremos observar sus diferentes outliers. Dado que se comparan las variables entre ellas, es necesario que se normalicen los datos para que se encuentren en el mismo rango, algo que ya hacen estas funciones:

```
MiBoxPlot_juntos(mydata)
```



```
MiBoxPlot_juntos_con_etiquetas(mydata)
```



Descubrimos los outliers que se han detectado para cada variables lo cuál nos da bastante información sobre ellas. Por ejemplo, sabemos que las variables con mayor número de anomalías son "Ba" y "Ca". De igual forma, en "Mg" no se ha detectado ningún outlier. También descubrimos que la mayoria de los outliers en el conjunto de datos son outliers superiores, es decir, que tienen valores por encima de los normal. Son muchos menos los que son outliers por el hecho de presentar un valor demasiado bajo.

En este estudio realizado se ha trabajado sobre una única variable, concretamente la primera. Ahora vamos a realizar este estudio de outliers univariantes sobre todas las variables. Empezamos determinando aquellas observaciones que puedan ser consideradas outliers por alguna de sus columnas gracias a la función "vector_claves_outliers_IQR":

```
indices.de.outliers.en.alguna.columna <- unlist(sapply(1:ncol(mydata),
    vector_claves_outliers_IQR,datos=mydata))
indices.de.outliers.en.alguna.columna
```

```
##   [1]  48  51  57 104 105 106 107 108 111 112 113 132 171 185 186 188 190
##  [18] 106 107 111 112 167 185 190  22  39  40  51 164 172 173 185 192 193
##  [35] 196 197 198 199 200 203 209 210 104 107 108 110 164 172 173 181 185
##  [52] 189 190 202 164 172 173 186 187 202 208 104 105 106 107 108 109 110
##  [69] 111 112 113 131 132 164 166 167 168 170 171 174 176 183 184 185 186
##  [86] 187 208  33  37  62 100 101 107 129 142 143 162 164 175 186 187 190
## [103] 191 192 193 194 195 196 197 198 199 200 201 203 204 205 206 207 208
## [120] 209 210 211 212 213 214   6  45  57  72 106 107 119 136 146 163 175
## [137] 176
```

A continuación con estos índices seleccionamos las observaciones normalizadas correspondientes:

```
mydata.scaled = scale(mydata)
head(mydata.scaled[indices.de.outliers.en.alguna.columna,])
```

```
##             RI         Na         Mg         Al         Si          K
## [1,]  2.734591  0.7128913  0.7040084 -1.4719632 -1.3955722 -0.7314659
## [2,]  1.591965  0.3822535  0.7178741 -1.8725483 -1.1631779 -0.6241355
## [3,] -2.046658 -0.5116932  0.5445528 -0.6507637  0.4248495  0.1885088
## [4,]  2.925577  0.4802202  0.3227016 -1.5721095 -2.6866514 -0.6394684
## [5,]  1.888323  0.5169577  0.1493803 -0.5506174 -1.9378255 -0.6394684
## [6,]  2.102360 -2.3975532 -1.8611468  0.8714599 -0.5951031  0.4798342
##              Ca         Ba         Fe
## [1,]  0.6064261 -0.3520514  0.4412072
## [2,]  0.7750657 -0.3520514  1.0569789
## [3,] -0.4264913 -0.3520514  2.5964083
## [4,]  1.8852762 -0.3520514 -0.5850791
## [5,]  1.2880110 -0.3520514 -0.5850791
## [6,]  3.0095400 -0.3520514  2.9042942
```

Vamos a construir una función que realice este cálculo anterior:

```
vector_claves_outliers_IQR_en_alguna_columna = function(misdatos, micoef = 1.5){

  milista <- unlist(sapply(1:ncol(misdatos),vector_claves_outliers_IQR,datos=misdatos,
                           coef = micoef))
  milista
}

indices.de.outliers.en.alguna.columna <-
  vector_claves_outliers_IQR_en_alguna_columna(mydata,1.5)
indices.de.outliers.en.alguna.columna
```

```
##   [1]  48  51  57 104 105 106 107 108 111 112 113 132 171 185 186 188 190
##  [18] 106 107 111 112 167 185 190  22  39  40  51 164 172 173 185 192 193
##  [35] 196 197 198 199 200 203 209 210 104 107 108 110 164 172 173 181 185
##  [52] 189 190 202 164 172 173 186 187 202 208 104 105 106 107 108 109 110
##  [69] 111 112 113 131 132 164 166 167 168 170 171 174 176 183 184 185 186
##  [86] 187 208  33  37  62 100 101 107 129 142 143 162 164 175 186 187 190
## [103] 191 192 193 194 195 196 197 198 199 200 201 203 204 205 206 207 208
## [120] 209 210 211 212 213 214   6  45  57  72 106 107 119 136 146 163 175
## [137] 176
```

Por último construimos una función que nos indique para cada dato si es outlier para alguna de las variables:

```
vector_es_outlier_IQR_en_alguna_columna = function(datos, coef = 1.5){
  indices.de.outliers.en.alguna.columna =
    vector_claves_outliers_IQR_en_alguna_columna(datos, coef)
  todos = c(1:nrow(datos))
  bools = todos %in% indices.de.outliers.en.alguna.columna
  return (bools)
}
son.outliers = vector_es_outlier_IQR_en_alguna_columna(mydata,1.5)
head(son.outliers,20)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
length(indices.de.outliers.en.alguna.columna)/nrow(mydata)
```

```
## [1] 0.6401869
```

Observamos que este test ha detectado demasiados outliers, concretamente determina que el 64% de los datos lo son para alguna variable.

Podemos estudiar el número de outliers extremos cambiando el coeficiente en la función:

```
length(vector_claves_outliers_IQR_en_alguna_columna(mydata,3))/nrow(mydata)
```

```
## [1] 0.2897196
```

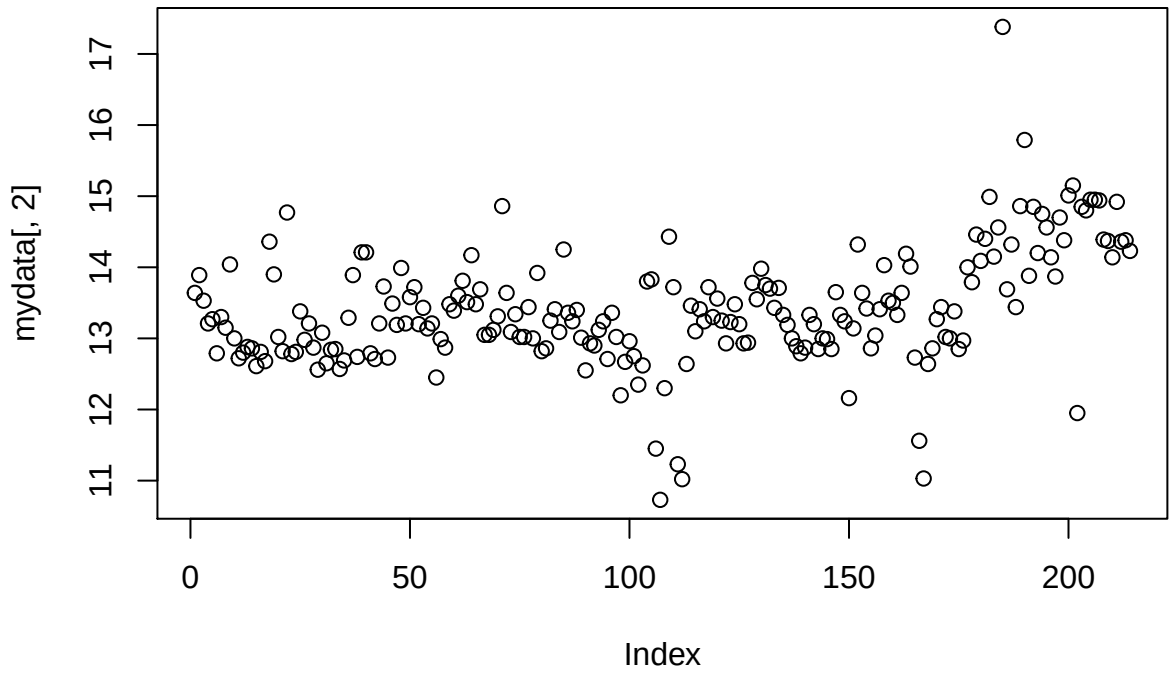
Ahora se han determinado que en torno al 29% de los datos son outliers extremos para algún atributo. Sigue siendo un valor muy elevado que se seguirá estudiando en los siguientes apartados.

Test de Grubb

El test de Grubb se ideó para detectar un outlier en unos datos univariables. A modo de ejemplo vamos ahora a aplicar este test sobre una de las variables del 'dataset' y comprobar así su funcionamiento.

En concreto vamos a utilizar la segunda variable, "Na":

```
plot(mydata[,2])
```



En la distribución de los datos para esta variable diferenciamos rápidamente varios datos más extremos por lo que debería detectar alguno de ellos.

```
grubbs.test(mydata[,2],two.sided=TRUE)
```

```
##
## Results of Hypothesis Test
## --------------------------
##
## Alternative Hypothesis:          highest value 17.38 is an outlier
##
## Test Name:                       Grubbs test for one outlier
##
## Data:                            mydata[, 2]
##
## Test Statistics:                 G = 4.8642325
##                                  U = 0.8883951
##
## P-value:                         0.0001208375
```

Con un p-value tan bajo de 0.0001 podemos confirmar que el valor 17.38 es un outlier. Ahora calculamos manualmente cuál es el indice de este outlier en función de la desviación de la media de la variable:

```
media = mean(mydata[,2])
desviaciones = abs(mydata[,2] - media)
indice.de.outlier.Grubbs = order(desviaciones)[length(desviaciones)]
indice.de.outlier.Grubbs
```

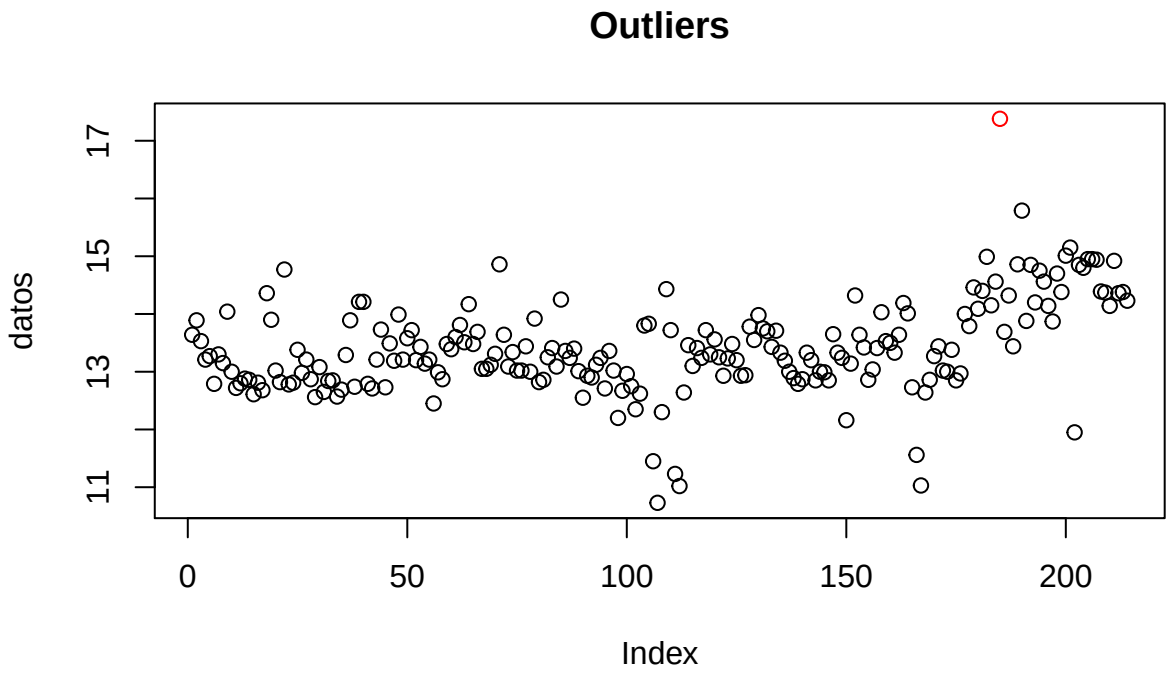
```
## [1] 185
```

```
valor.de.outlier.Grubbs = mydata[indice.de.outlier.Grubbs,2]
valor.de.outlier.Grubbs
```

```
## [1] 17.38
```

Descubrimos que el outlier detectado es el dato 185:

```
MiPlot_Univariate_Outliers(mydata[,2],indice.de.outlier.Grubbs,"Outliers")
```



Efectivamente se trata del dato que visualmente aparece más extremo en el gráfico.

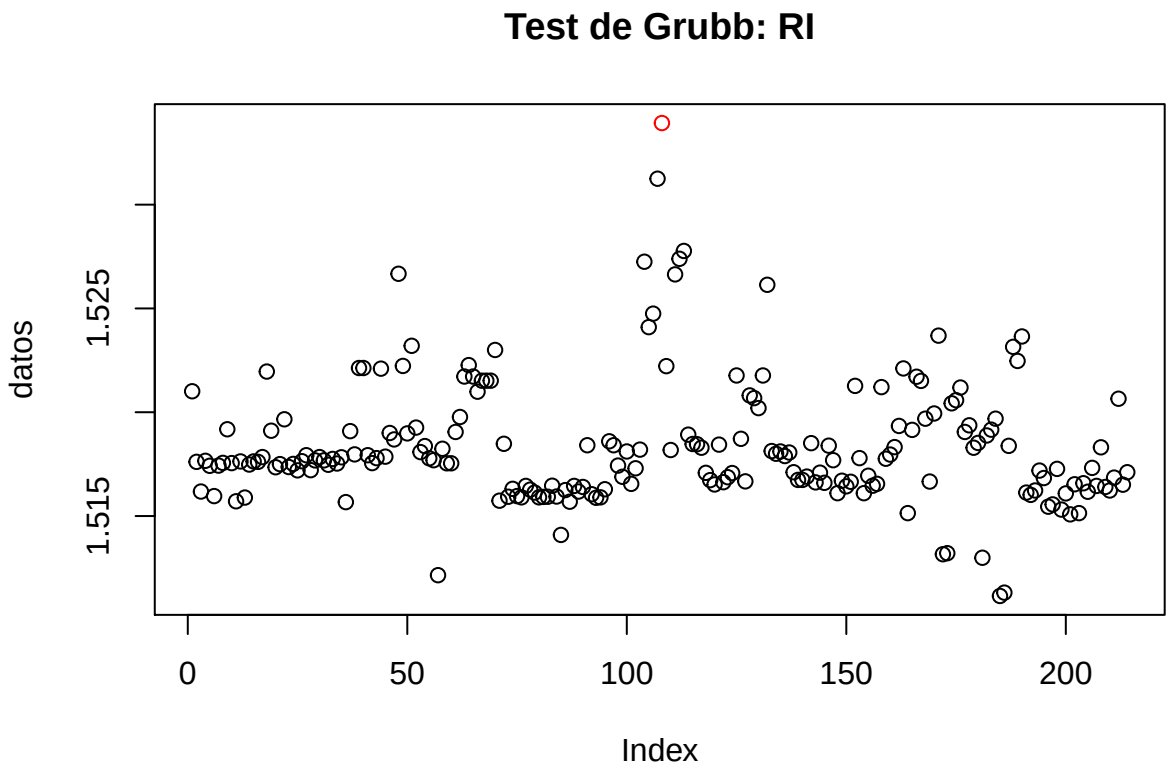
A continuación vamos a aplicar el test de Grubb para cada una de las variables que tenemos en nuestros datos. Aquí se muestran los índices de los outliers detectados para cada variable apoyándonos en la función "MiPlot_resultados_TestGrubbs":

```
result = apply(mydata,2,MiPlot_resultados_TestGrubbs)
result
```

```
##  RI  Na  Mg  Al  Si   K  Ca  Ba  Fe
## 108 185 106 164 107 172 108 107 175
```

Y aquí se visualiza un ejemplo de este test para la primera variable:

```
MiPlot_Univariate_Outliers(mydata[,1],result[1],"Test de Grubb: RI")
```



Hay que tener en cuenta en este test que al detectar únicamente un outlier; si hay dos outliers en los datos de forma que uno "enmascare" a otro, es decir, que no le haga parecer outlier, este test no detectaría ninguna anomalía.

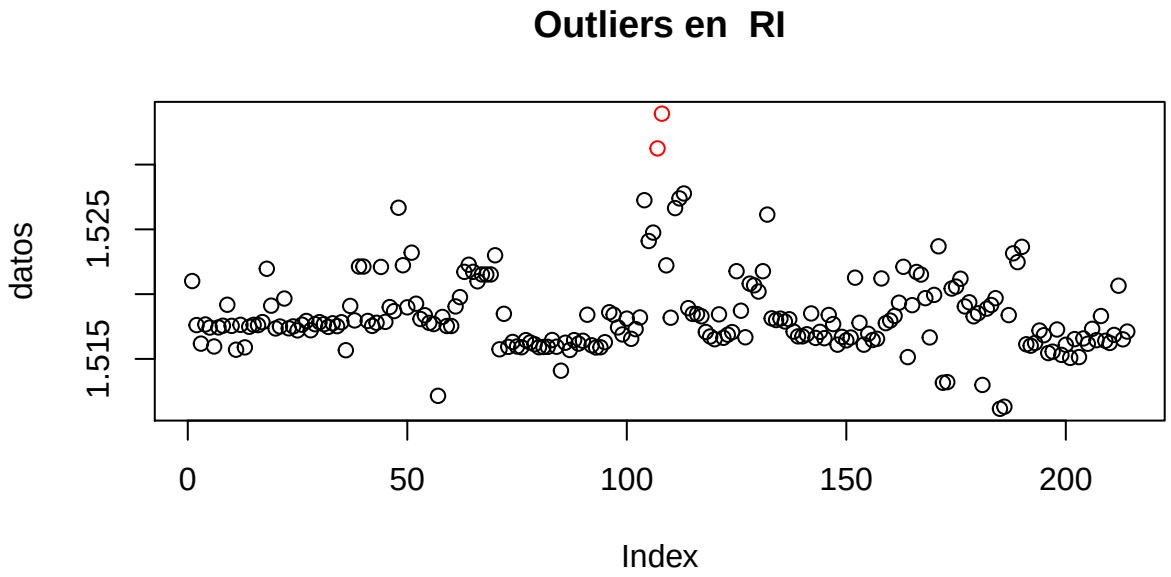
Test de Rosner

El hecho de que el test de Grubb sólo vaya a detectar un único outlier hace que sea necesario complementarlo con otros tests. El test de Rosner detectará si hay al menos "k" outliers para cada una de las variables del dataset (se ha escogido k=7, un 3% de los datos). Estos son los outliers que se detectan para cada variable junto con su representación gráfica:

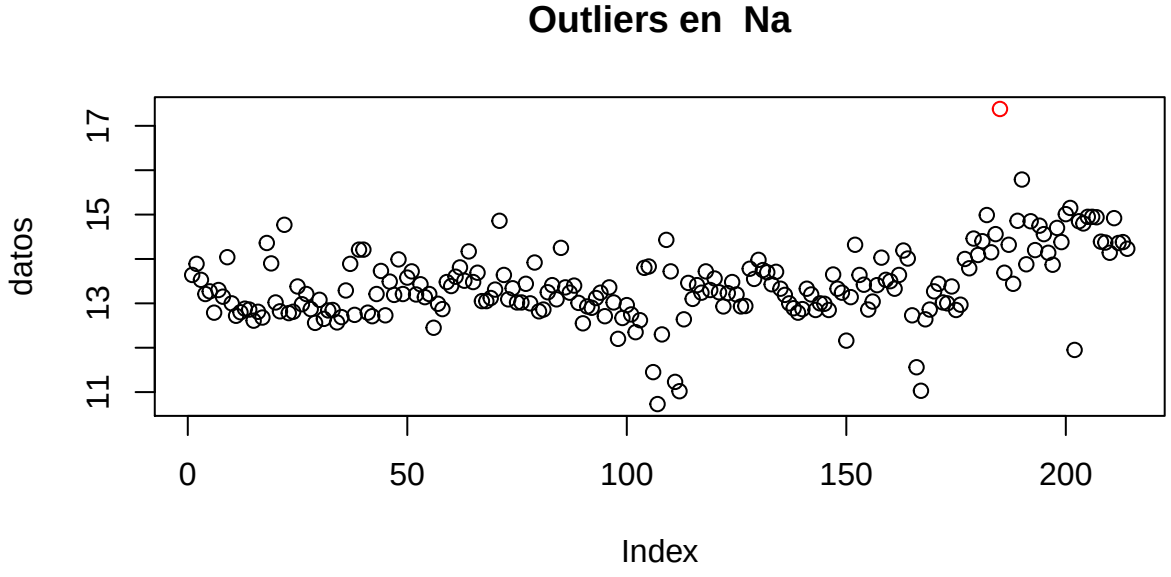
```
pasarTestRosner <- function(ind, datos=mydata,miK=7){
  rostest <- rosnerTest(mydata[,ind],miK,warn=FALSE)
  indices <- rostest$all.stats$Obs.Num[rostest$all.stats$Outlier]
  cat(paste("En la variable ",toString(colnames(mydata)[ind]), " son outliers los datos ",
            toString(indices), " con valores:\n ",toString(mydata[indices,ind]),"\n"))
```

```
  MiPlot_Univariate_Outliers(mydata[,ind],indices,paste("Outliers en ",
                                                        toString(colnames(mydata)[ind])))
}
par(mfrow=c(1,1))
result <- lapply(1:ncol(mydata),pasarTestRosner)
```

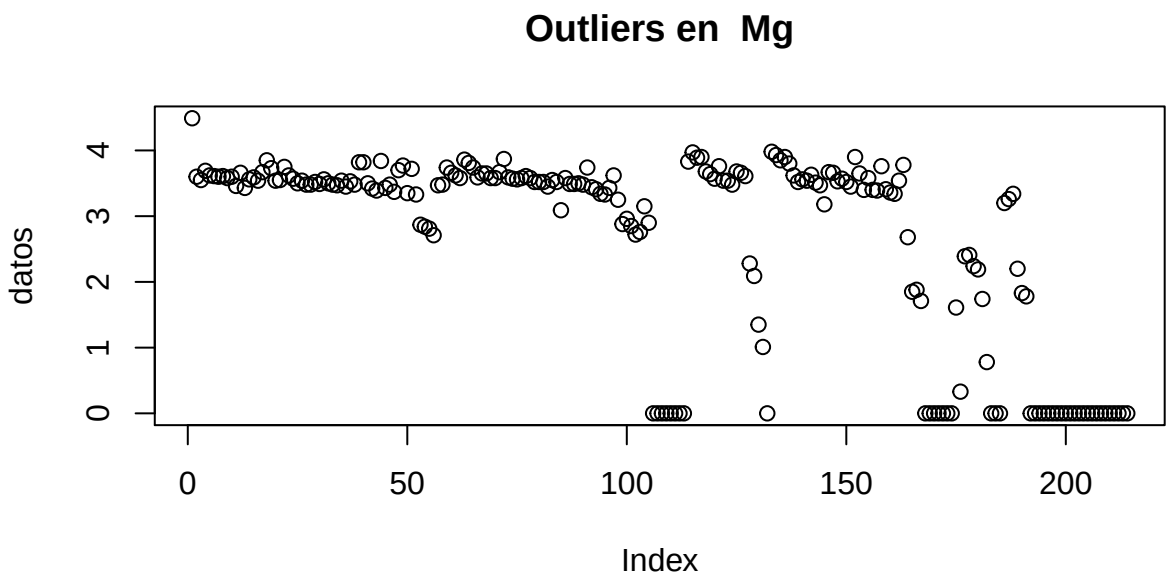
```
## En la variable  RI  son outliers los datos  108, 107  con valores:
##   1.53393, 1.53125
```



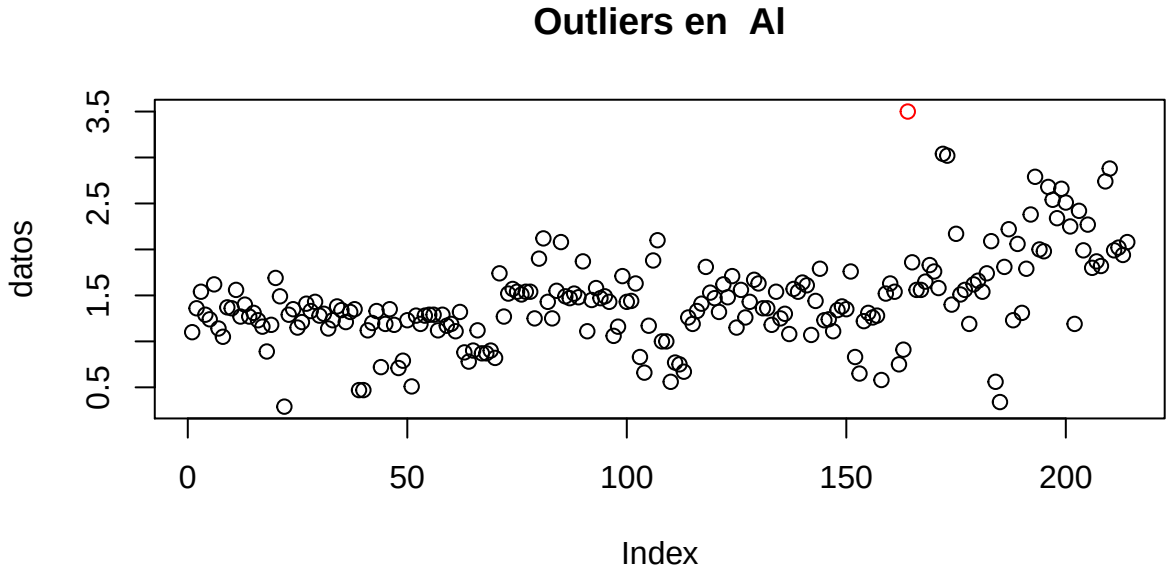

```
## En la variable  Na  son outliers los datos  185  con valores:
##   17.38
```



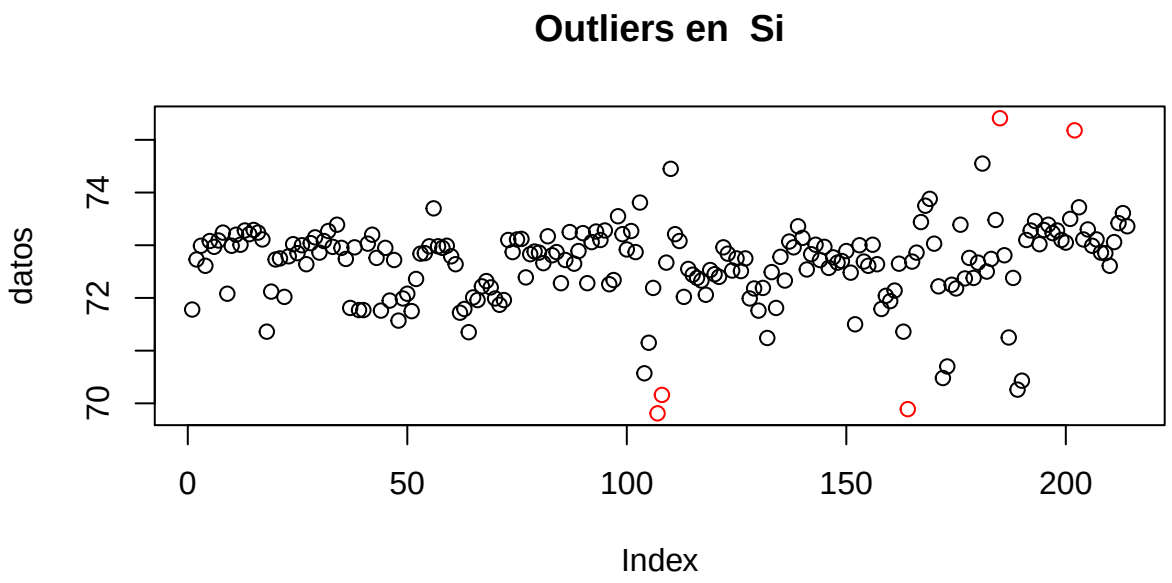

```
## En la variable  Mg  son outliers los datos    con valores:
##
```



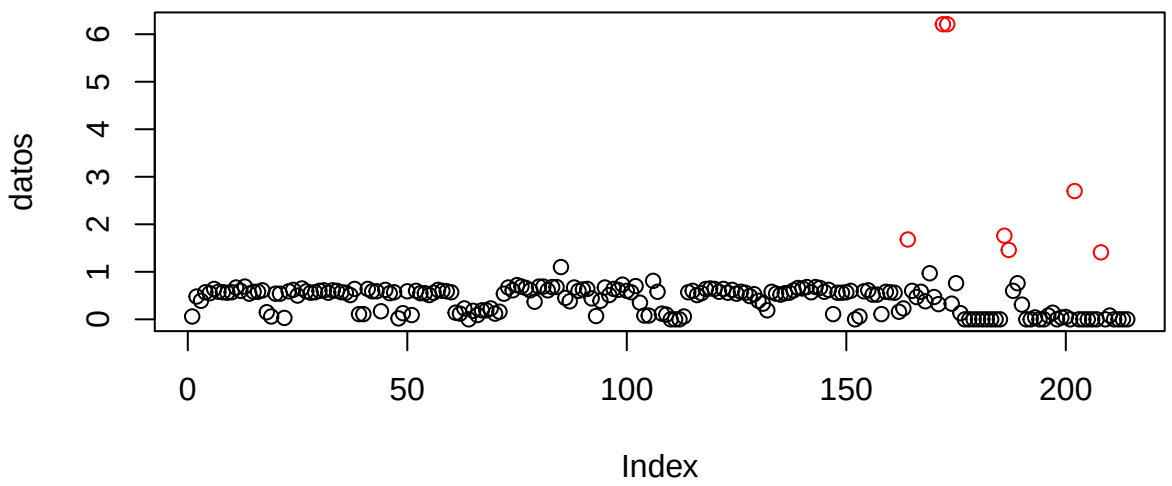

```
## En la variable  Al  son outliers los datos  164  con valores:
##   3.5
```



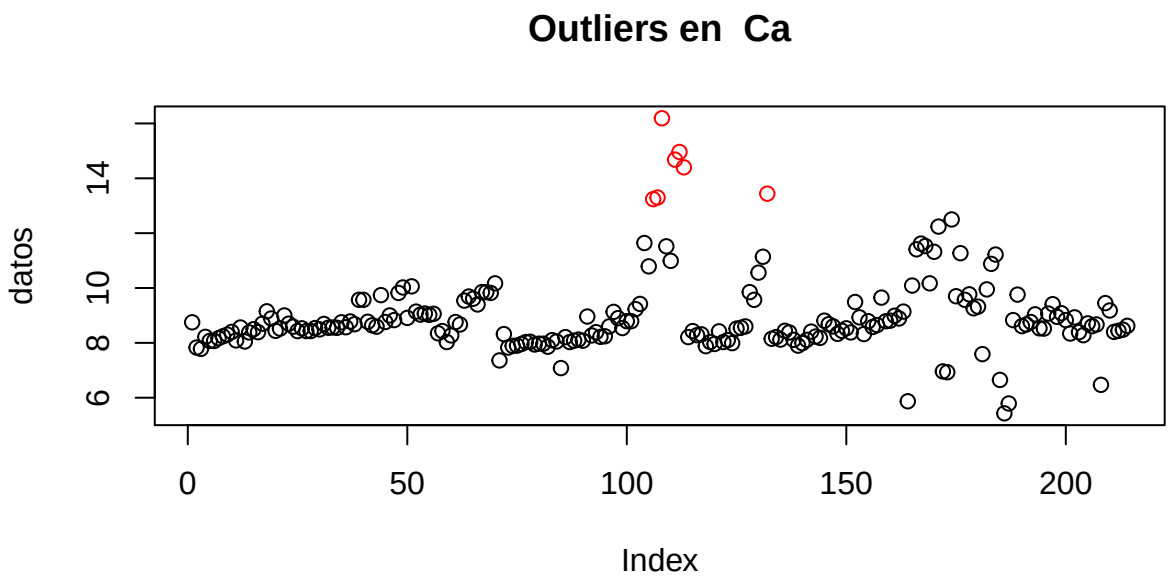

```
## En la variable  Si  son outliers los datos  107, 164, 185, 202, 108  con valores:
##   69.81, 69.89, 75.41, 75.18, 70.16
```



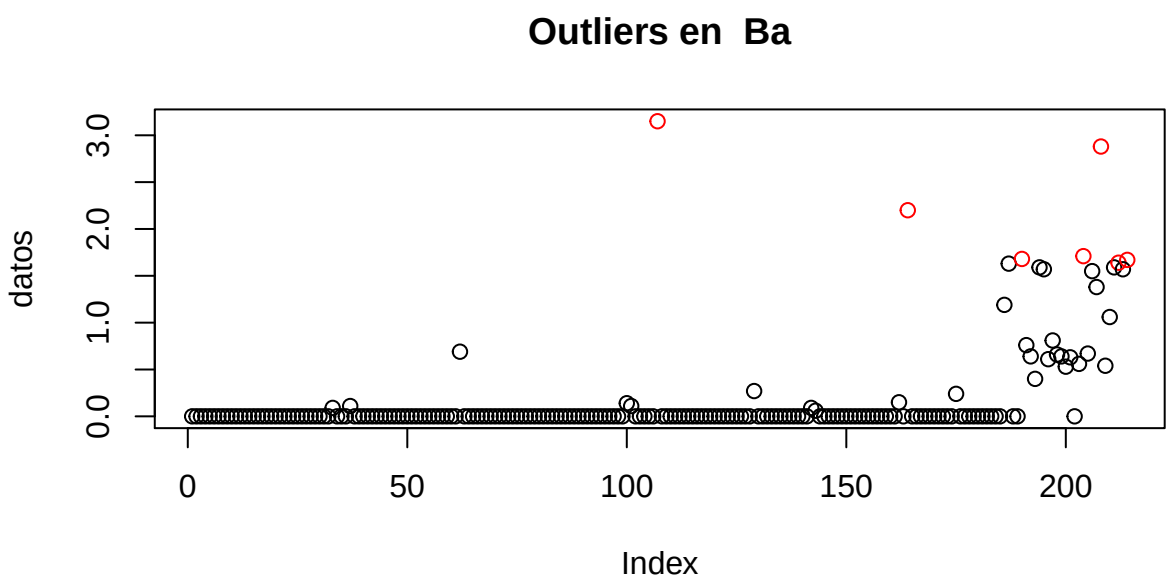

```
## En la variable  K  son outliers los datos  172, 173, 202, 186, 164, 187, 208  con valores:
##   6.21, 6.21, 2.7, 1.76, 1.68, 1.46, 1.41
```



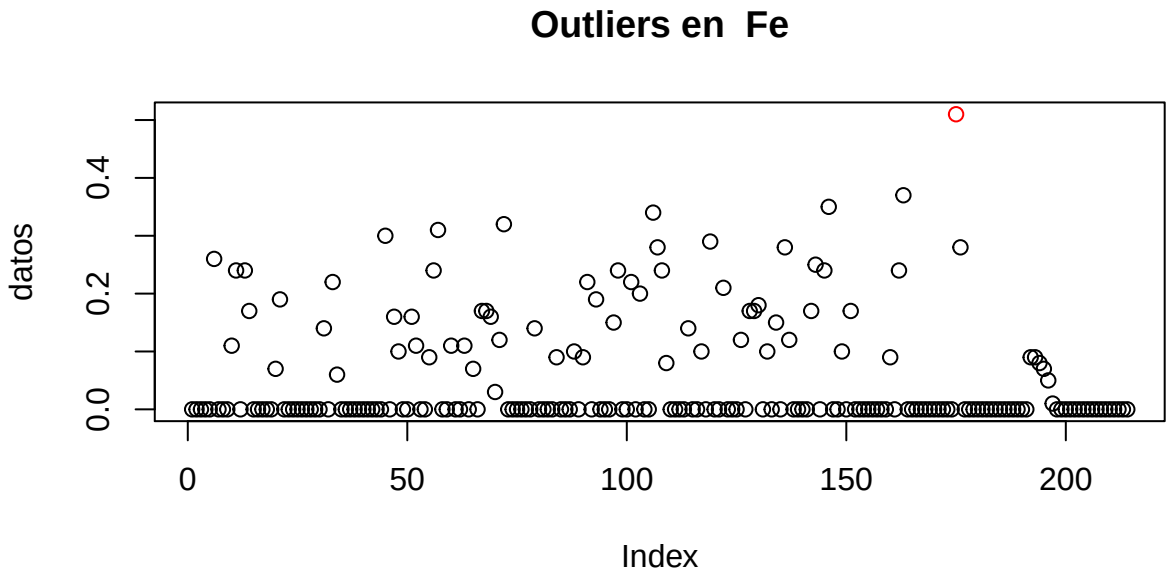

```
## En la variable  Ca  son outliers los datos  108, 112, 111, 113, 132, 107, 106  con valores:
##   16.19, 14.96, 14.68, 14.4, 13.44, 13.3, 13.24
```




```
## En la variable  Ba  son outliers los datos  107, 208, 164, 204, 190, 214, 212  con valores:
##   3.15, 2.88, 2.2, 1.71, 1.68, 1.67, 1.64
```



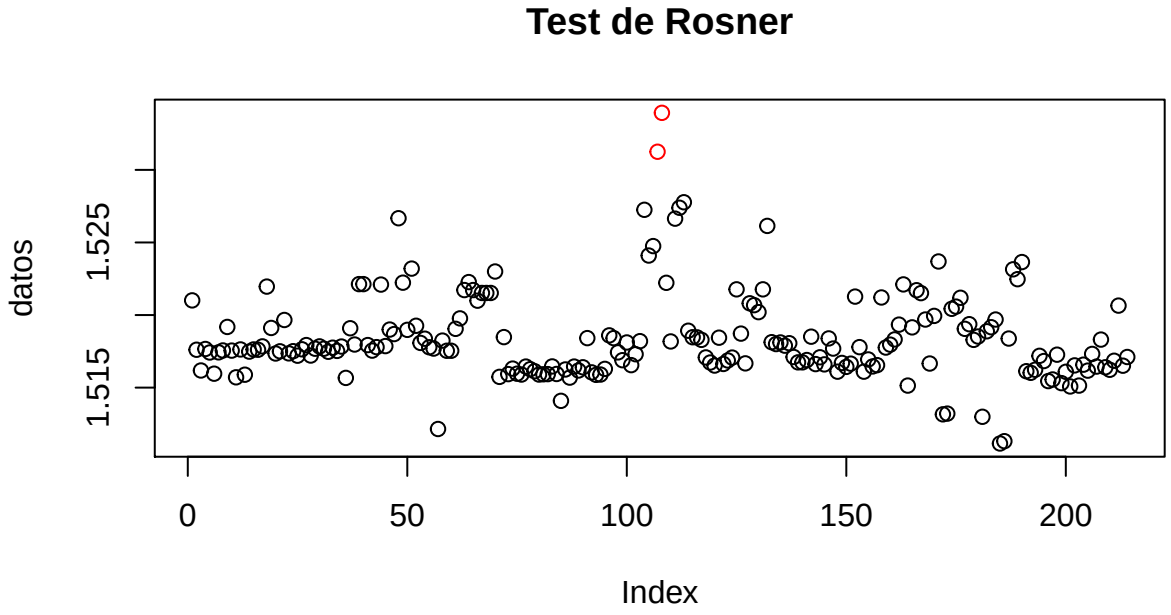

```
## En la variable  Fe  son outliers los datos  175  con valores:
##   0.51
```



Podemos ahora comparar estos resultados con los de la función proporcionada "MiPlot_resultados_TestRosner", por ejemplo para la primera variable, y saber así si coinciden:

```
MiPlot_resultados_TestRosner(mydata[,1])
```

```
##
## Test de Rosner
## ?ndices de las k-mayores desviaciones de la media: 108 107 113 112
## De las k mayores desviaciones, ?Qui?n es outlier? TRUE TRUE FALSE FALSE
## Los ?ndices de los outliers son: 108 107
## Los valores de los outliers son: 1.53393 1.53125
```



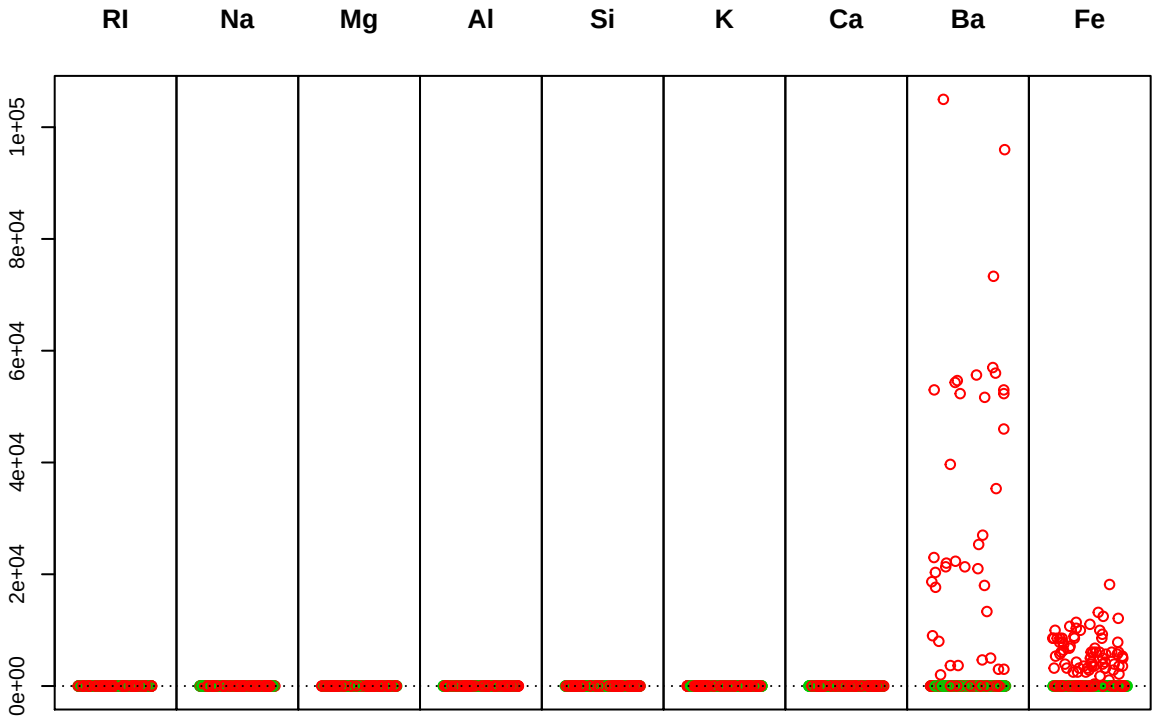
Efectivamente coinciden los resultados obtenidos.

Outliers multivariantes

En este apartado se descubrirá si este dataset contiene outliers multivariantes, es decir, que tengan una combinación anómala de valores en varias variables, no necesariamente que tenga que ser un valor extremo por sí solo.

En el momento de realizar este proceso, la función "uni.plot" nos informa que las variables 8 y 9 propician a que el determinante de la matriz sea 0, por lo tanto que no se pueda invertir y que la función de la matriz no pueda trabajar. Para solventarlo se puede añadir una mínima modificación a los valores de estas variables, que sea despreciable, pero sea suficiente para que el determinante no sea exactamente 0 y el método funcione.

```
random8 <- runif(214,0,0.0001)
random9 <- runif(214,0,0.0001)
mydataMod <- data.frame(mydata)
mydataMod[,8] <- mydataMod[,8] + random8
mydataMod[,9] <- mydataMod[,9] + random9
set.seed(12)
mvoutlier = uni.plot(mydataMod,symb=FALSE,alpha=0.05)
```




```
head(mvoutlier$outliers,20)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE
## [12] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
```

Gracias a la función "uni.plot" descubrimos los outliers multivariantes, aunque observando este gráfico parece que la función ha detectado demasiados. Para saber cuáles son los outliers encontrados:

```
is.MCD.outlier = mvoutlier$outliers
indices.de.outliers.multivariantes.MCD = which(is.MCD.outlier == TRUE)
indices.de.outliers.multivariantes.MCD
```

```
##   [1]   6  10  11  13  14  20  21  31  33  34  37  45  47  48  51  52  55
```

```
##  [18]  56  57  60  62  63  65  67  68  69  70  71  72  79  84  88  90  91
##  [35]  93  97  98 100 101 103 106 107 108 109 110 114 117 119 122 126 128
##  [52] 129 130 132 134 136 137 142 143 145 146 149 150 151 160 162 163 164
##  [69] 172 173 175 176 181 184 185 186 187 188 189 190 191 192 193 194 195
##  [86] 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212
## [103] 213 214
```

```
numero.de.outliers.MCD = length(indices.de.outliers.multivariantes.MCD)
numero.de.outliers.MCD
```

```
## [1] 104
```

Efectivamente se han detectado 104 outliers lo que se traduce en que cerca de la mitad de los datos los considera como tales, ya sea por un valor extremo o por una combinación anómala como ya se comentó.

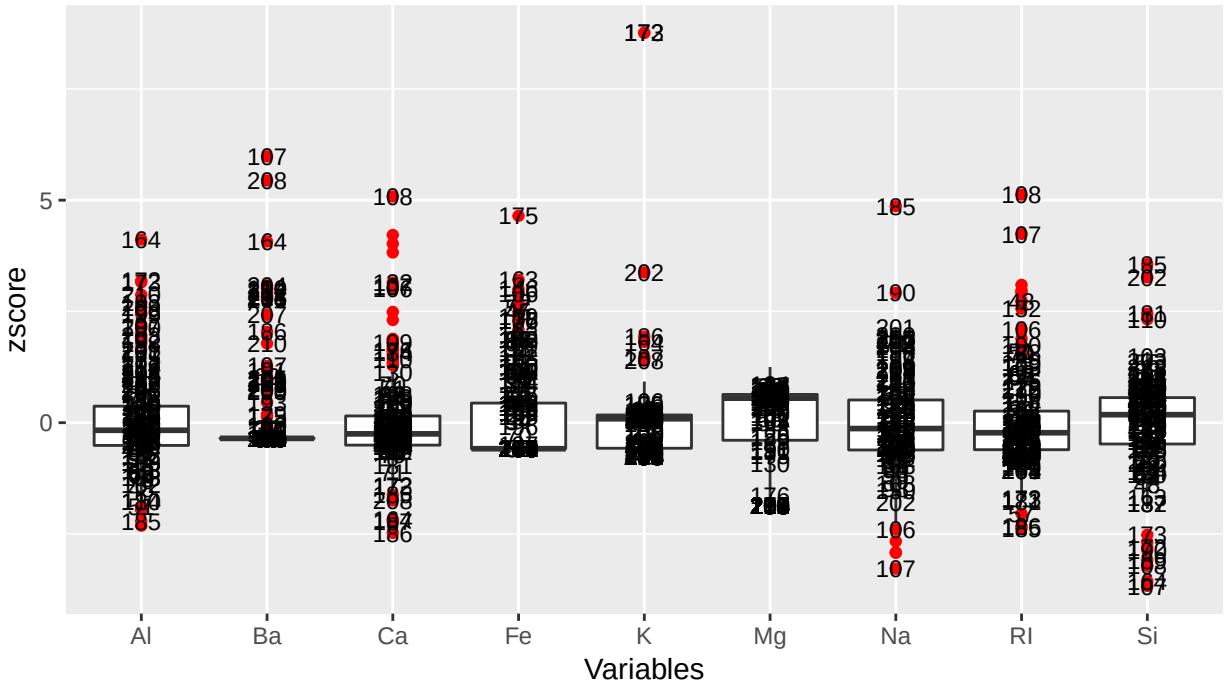
Podemos ahora construir un 'dataframe' que recoja estos outliers con los datos normalizados:

```
mydataMod.scaled = scale(mydataMod)
data.frame.solo.outliers = mydataMod.scaled[is.MCD.outlier,]
head(data.frame.solo.outliers)
```

```
##              RI         Na        Mg          Al        Si          K
## [1,] -0.7920739 -0.7566101 0.6416128  0.35069919 0.4119387 0.21917466
## [2,] -0.2685075 -0.4994473 0.6346799 -0.17006149 0.4377603 0.11184428
## [3,] -0.8743957 -0.8423310 0.5376200  0.23052364 0.7088870 0.26517339
## [4,] -0.8151240 -0.6463975 0.5168214 -0.08994447 0.8121733 0.29583922
## [5,] -0.2915576 -0.6708892 0.6069485 -0.35032481 0.7217978 0.06584554
## [6,] -0.3343649 -0.4749556 0.5930828  0.49090398 0.1020797 0.06584554
##              Ca         Ba        Fe
## [1,] -0.6232375 -0.3521371 2.0832369
## [2,] -0.3913581 -0.3520697 0.5437401
## [3,] -0.6091842 -0.3520589 1.8782541
## [4,] -0.6372908 -0.3521146 1.8779361
## [5,] -0.4054114 -0.3519976 1.1592109
## [6,] -0.3632515 -0.3521045 0.1330965
```

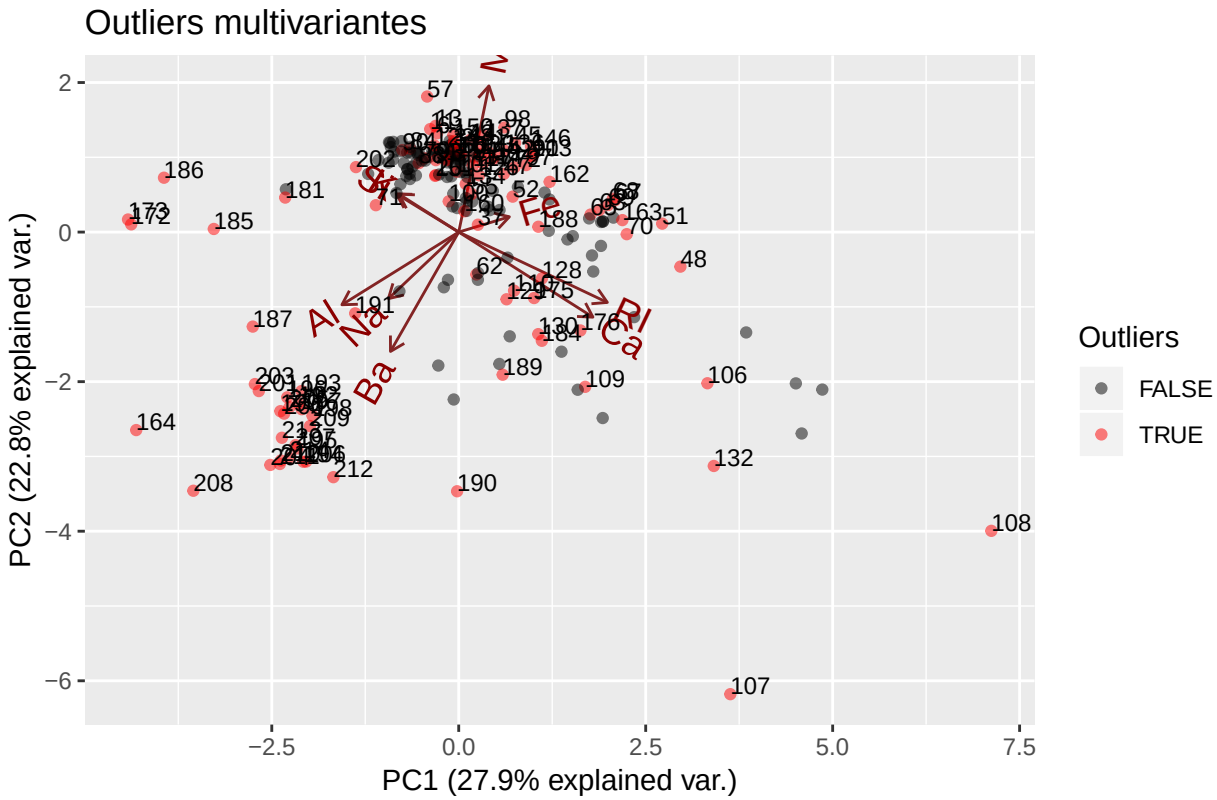
Ahora vamos a dibujar los boxplots correspondientes utilizando las etiquetas de los outliers encontrados. Utilizamos la función "MiBoxPlot_juntos":

```
MiBoxPlot_juntos(mydata,is.MCD.outlier)
```



Con este gráfico observamos que bastantes outliers univariantes han sido detectados de nuevo.

```
MiBiPlot_Multivariate_Outliers(mydata,is.MCD.outlier,"Outliers multivariantes")
```



Observamos que si bien muchos de los outliers indicados lo son también de forma univariante, es decir, por un valor extremo; existen otros datos como el 128 que, sin tener valores extremos, sí se ha detectado ahora como outlier multivariante por una combinación anómala de dos o más variables. Este tipo de outliers son conocidos como outliers multivariantes puros. Descubrimos cuáles son estos outliers en nuestro 'dataset' quitando los que fueron detectados anteriormente como univariantes:

```
indices.de.outliers.multivariantes.MCD.pero.no.1variantes =
setdiff(indices.de.outliers.multivariantes.MCD,indices.de.outliers.en.alguna.columna)

indices.de.outliers.multivariantes.MCD.pero.no.1variantes
```

```
##  [1]  10  11  13  14  20  21  31  34  47  52  55  56  60  63  65  67  68
## [18]  69  70  71  79  84  88  90  91  93  97  98 103 114 117 122 126 128
## [35] 130 134 137 145 149 150 151 160
```

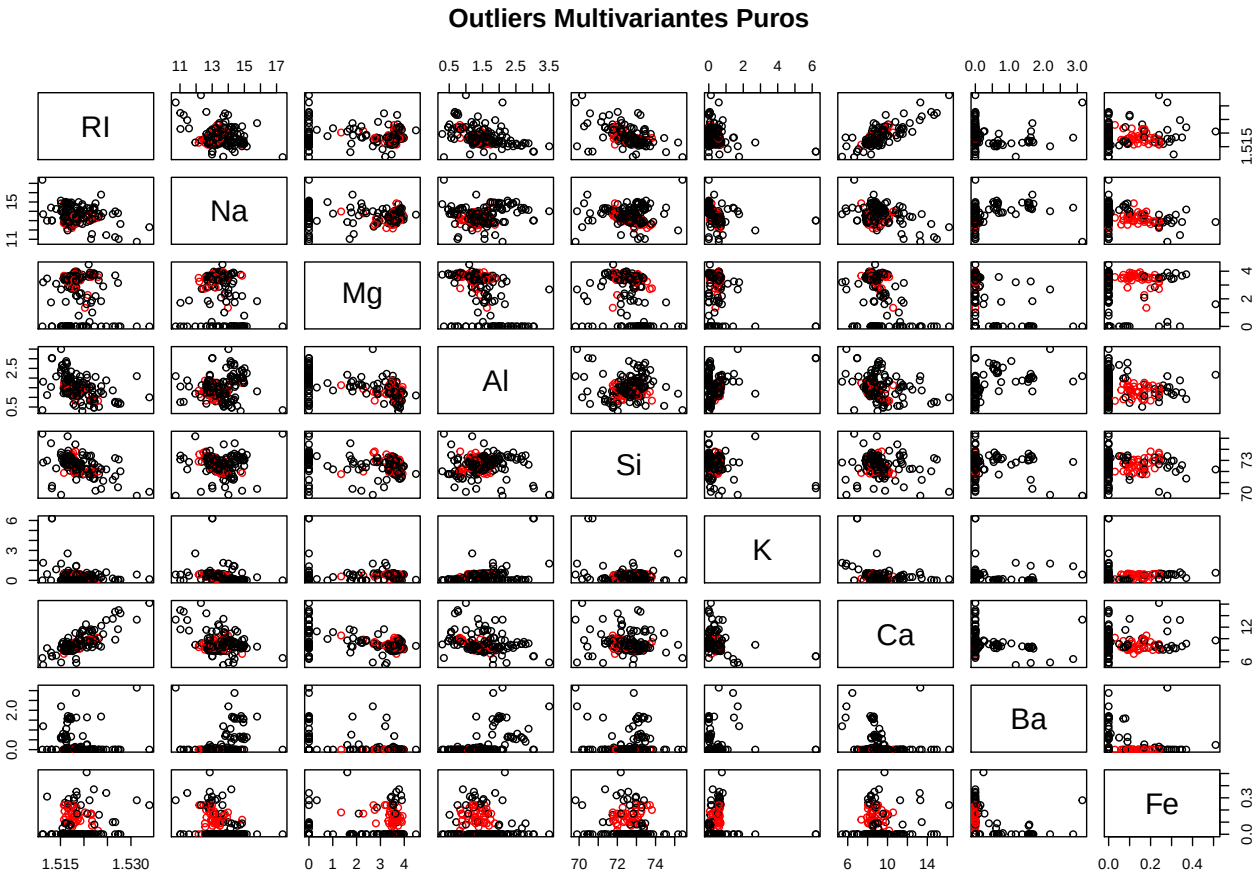
```
length(indices.de.outliers.multivariantes.MCD.pero.no.1variantes)
```

```
## [1] 42
```

Se tiene un total de 42 outliers multivariantes 'puros'.

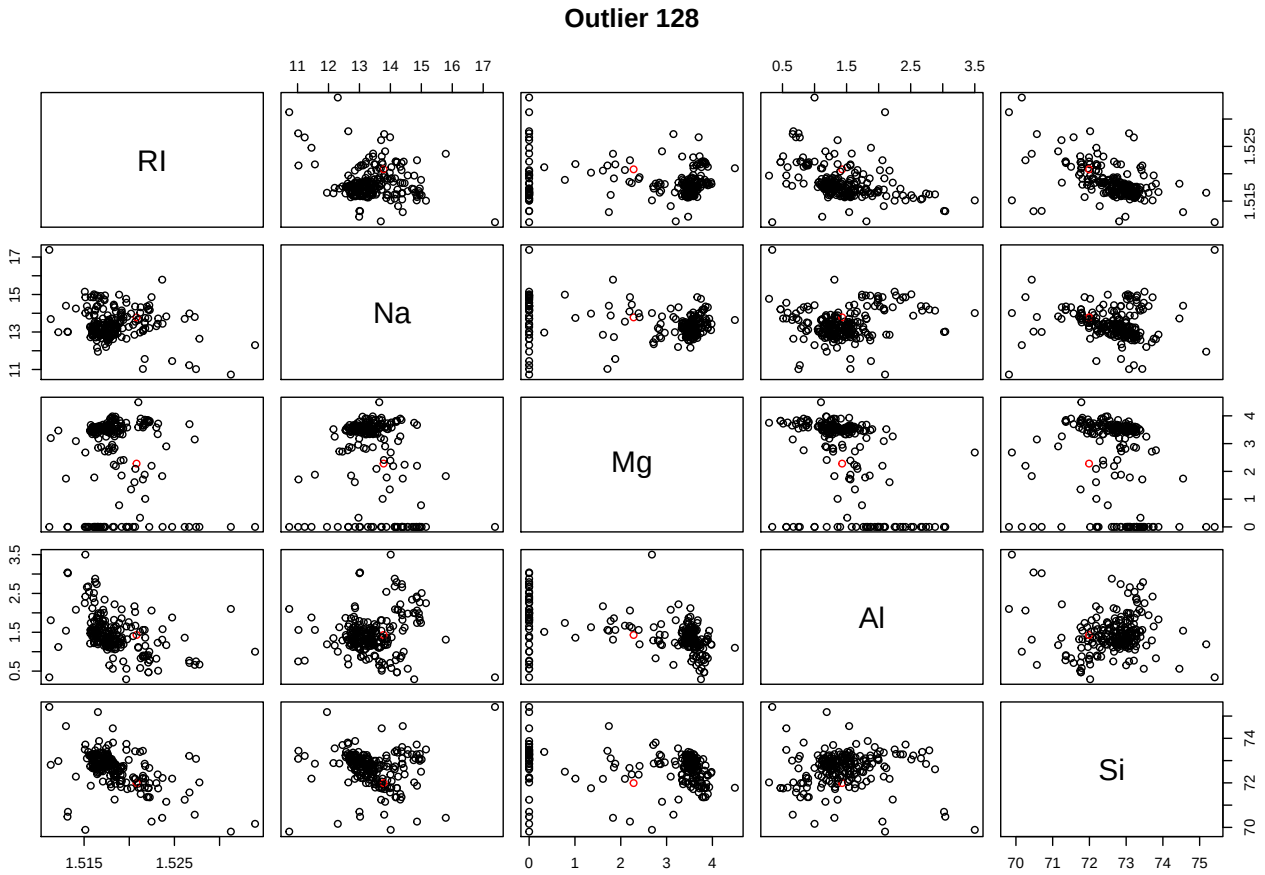
Aquí tenemos los gráficos de distribución para cada par de variables en los que se marcan en rojo estos outliers:

```
MiPlot_Univariate_Outliers(mydata,indices.de.outliers.multivariantes.MCD.pero.no.1variantes,
                           "Outliers Multivariantes Puros")
```

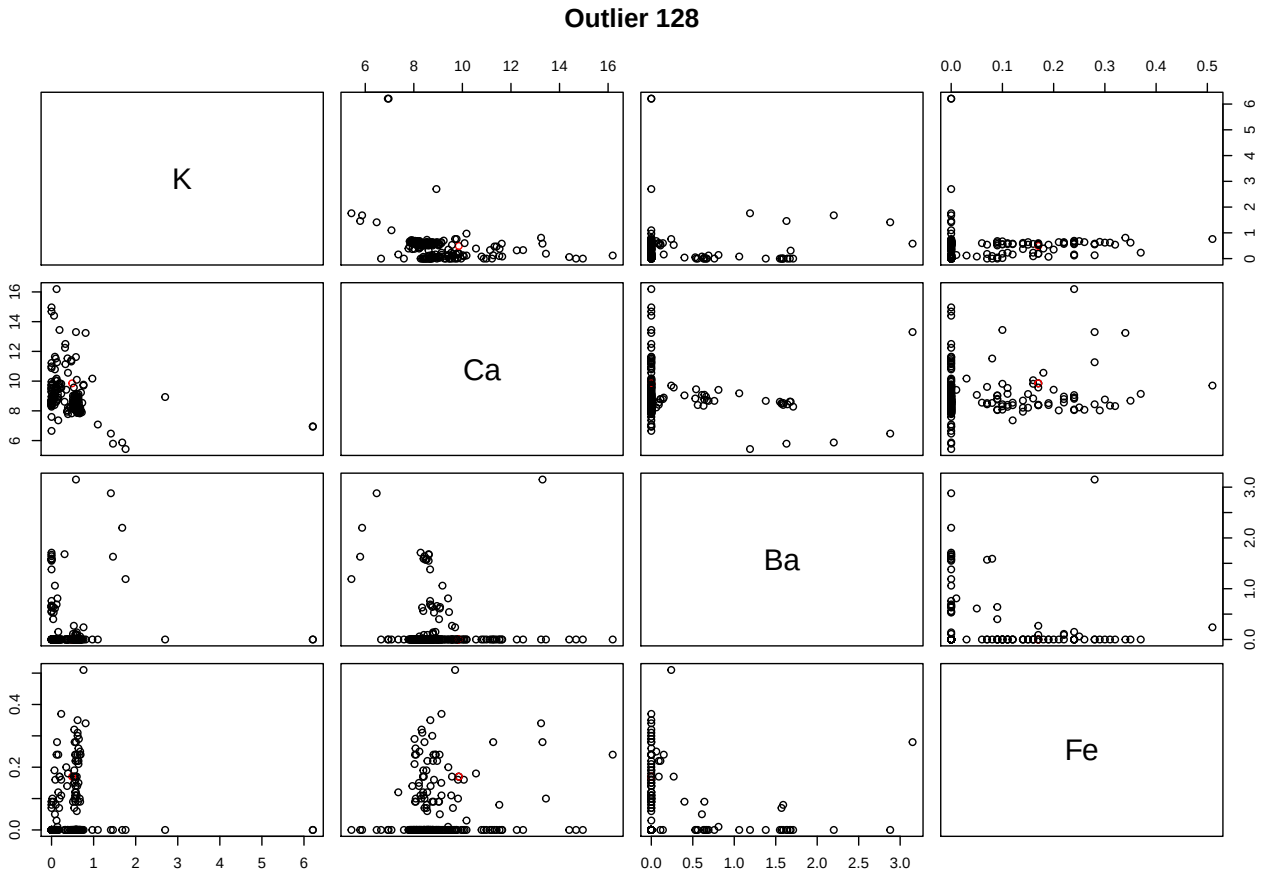


Nos fijamos en alguno de los outliers multivariantes puros encontrados, por ejemplo, el 128 que se comentó anteriormente:

```
MiPlot_Univariate_Outliers(mydata[,1:5],128,"Outlier 128")
```

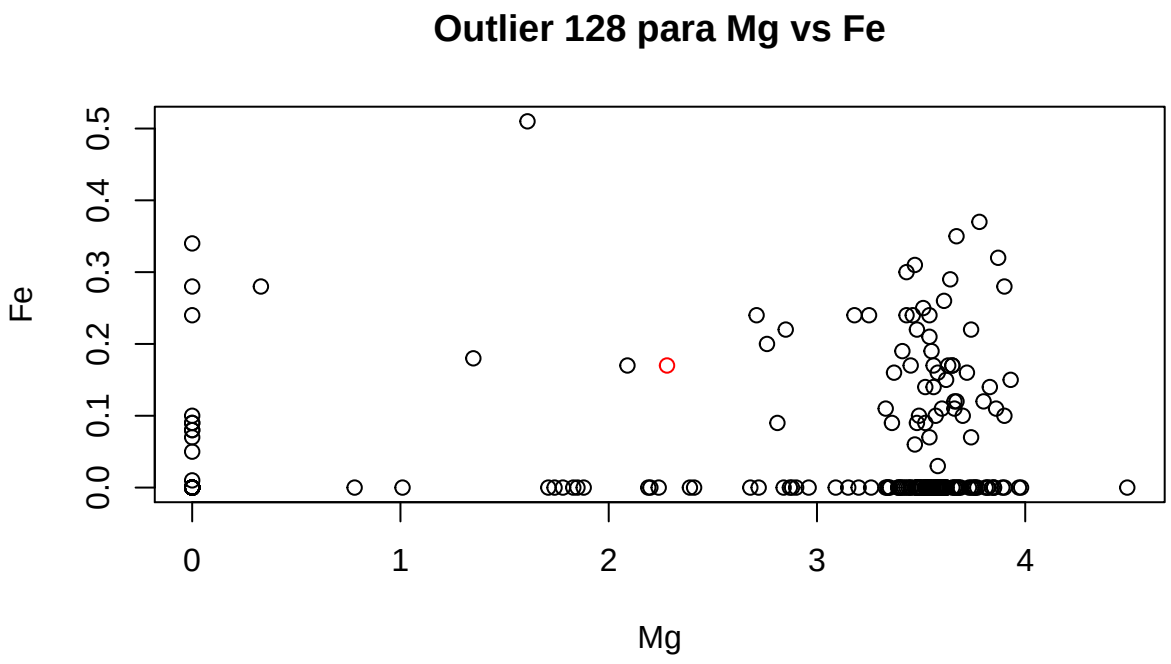



```
MiPlot_Univariate_Outliers(mydata[,6:9],128,"Outlier 128")
```



Se trata de un dato que para ninguna de las variables observamos un valor extremo en las distribuciones que lo diferencie del resto de datos. Sin embargo, por ejemplo para la distribución "Mg vs Fe", no se encuentra dentro de un cúmulo de datos lo cuál nos indica que su combinación de valores en las variables "Mg" y "Fe" es inusual y lo podría convertir por ello en un outlier multivariante "puro".

```
MiPlot_Univariate_Outliers(mydata[,c(3,9)],128,"Outlier 128 para Mg vs Fe")
```



Local outlier factor (LOF)

En este apartado vamos a utilizar el algoritmo LOF, basado en la densidad local de los datos, para detectar las anomalías en el conjunto de datos. Se basa en la idea de que una anomalía aparecerá en una región de baja densidad de datos.

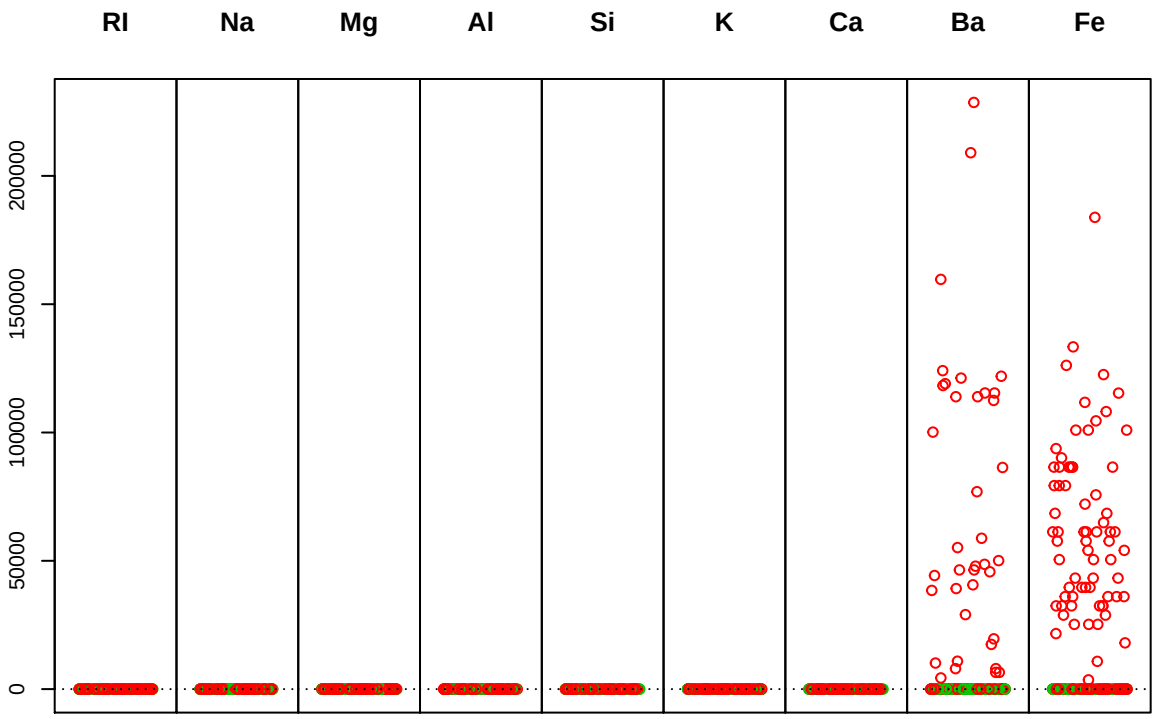
Dado que LOF utiliza distancias entre registros, lo primero será normalizar los datos:

```
mydata.scaled <- scale(mydata)
```

A continuación comprobamos los outliers que se detectan según MCD:

```
random8 <- runif(214,0,0.0001)
random9 <- runif(214,0,0.0001)
mydataMod.scaled <- data.frame(mydata.scaled)
mydataMod.scaled[,8] <- mydataMod.scaled[,8] + random8
mydataMod.scaled[,9] <- mydataMod.scaled[,9] + random9

mvoutlier = uni.plot(mydataMod.scaled,symb=FALSE,alpha=0.05)
```




```
is.MCD.outlier = mvoutlier$outliers
head(is.MCD.outlier,20)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE
## [12] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
```

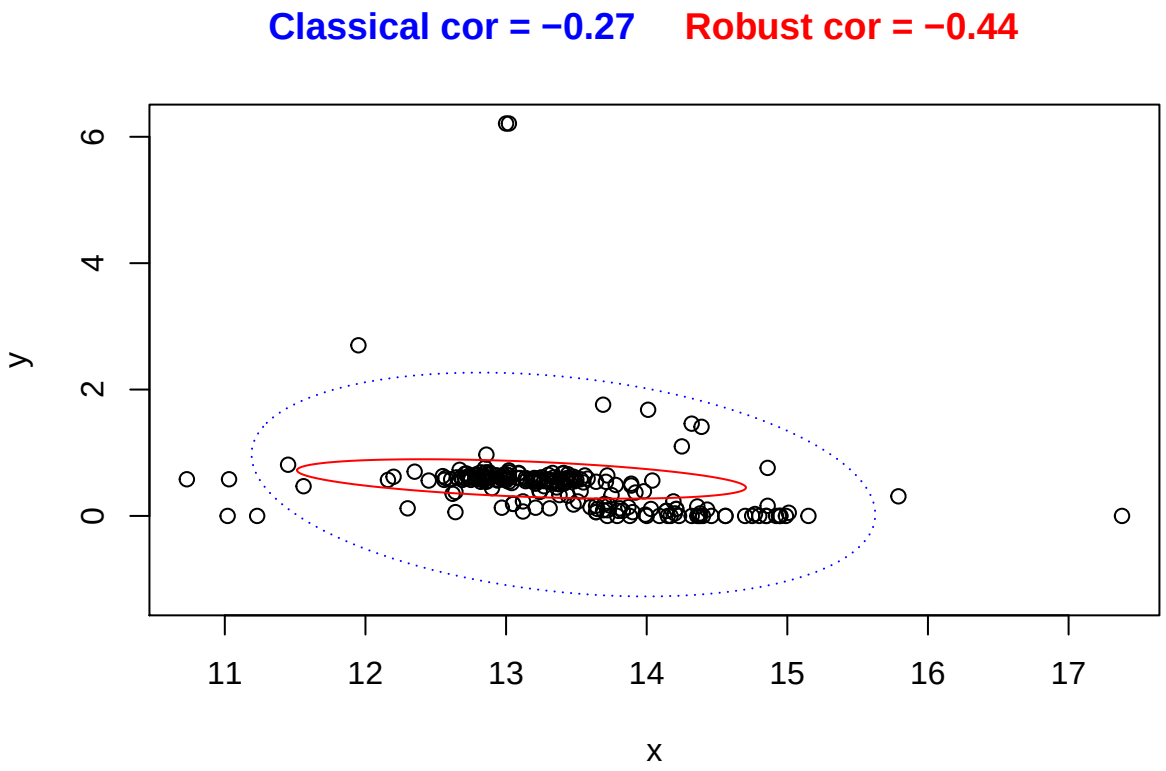
```
numero.de.outliers.MCD = length(which(is.MCD.outlier == TRUE))
numero.de.outliers.MCD
```

```
## [1] 104
```

Descubrimos que se han detectado un total de 104. Un valor muy alto teniendo en cuenta que se tienen 214 observaciones, es decir, ha detectado casi la mitad de los datos como outliers.

Respecto a esto podemos dibujar la distribucion de los datos cruzando variables. La elipsis azul muestra el uso de la distancia de Mahalanobis clásica mientras que la elipsis roja es construida mediante la distancia de Mahalanobis que utiliza la matriz de covarianza. Vemos un ejemplo con las variables 2 y 6:

```
corr.plot(mydata[,2],mydata[,6])
```

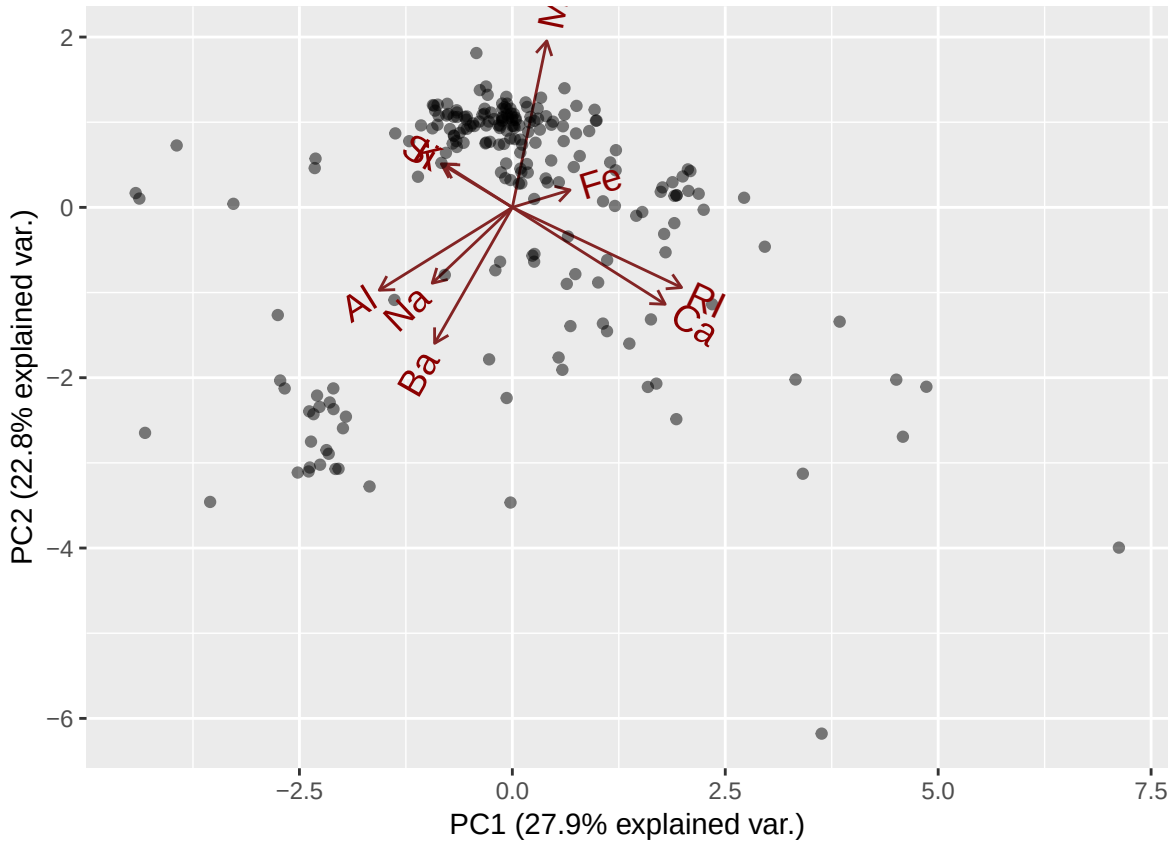



```
## $cor.cla
## [1] -0.2660865
##
## $cor.rob
## [1] -0.4398215
```

La distribución muestra dos grupos que se encuentran a dos niveles en el eje "y". La distancia clásica incluye a ambos grupos junto con otros datos que no son de ningún grupo y que deberían ser outliers. La distancia robusta sólo incluye un único grupo, clasificando al otro grupo como outliers, de ahí que detecte tantos outliers. Por lo tanto esto nos sugiere que no es recomendable utilizar aquí la distancia de Mahalanobis en ninguna de sus dos versiones.

Aunque la aproximación no sea muy buena, podemos utilizar "MiBiplot" que contempla todas las variables:

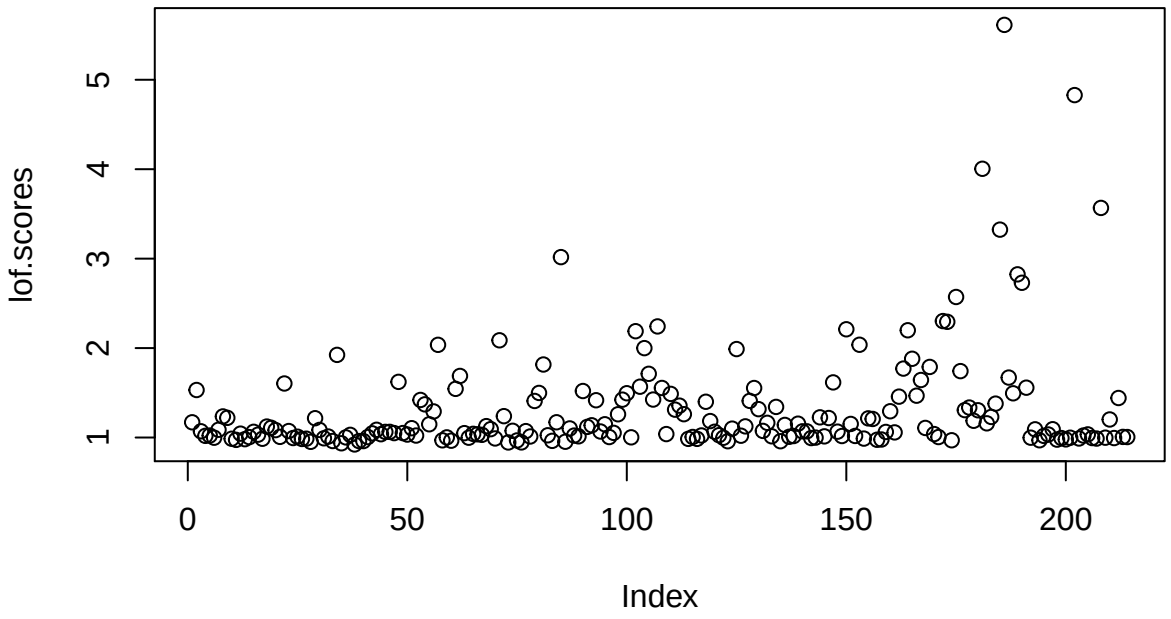
```
MiBiplot(mydata)
```



Y se pueden llegar a diferenciar hasta tres grupos diferentes que hacen que Mahalanobis no nos resulte útil.

Utilizando ya LOF, empezamos calculando la puntuación de anomalía para cada uno de los datos:

```
numero.de.vecinos.lof = 5
lof.scores <- lofactor(mydata.scaled,numero.de.vecinos.lof)
plot(lof.scores)
```



A priori no sabemos cuántos outliers hay en el 'dataset'; sin embargo, fijándonos en este gráfico con las puntuaciones de anomalías calculadas, podemos diferenciar visualmente hasta 9 valores más extremos a partir de los cuales ya hay muchos más datos menos diferenciados. Para probar esto escogemos las 20 instancias con mayor puntuación de anomalías:

```
numero.de.outliers = 20
indices.de.lof.outliers <- order(lof.scores,decreasing = TRUE)[1:numero.de.outliers]
indices.de.lof.outliers
```

```
##  [1] 186 202 181 208 185  85 189 190 175 172 173 107 150 164 102  71 153
## [18]  57 104 125
```

```
is.lof.outliers <- 1:nrow(mydata) %in% indices.de.lof.outliers
lof.scores[indices.de.lof.outliers]
```

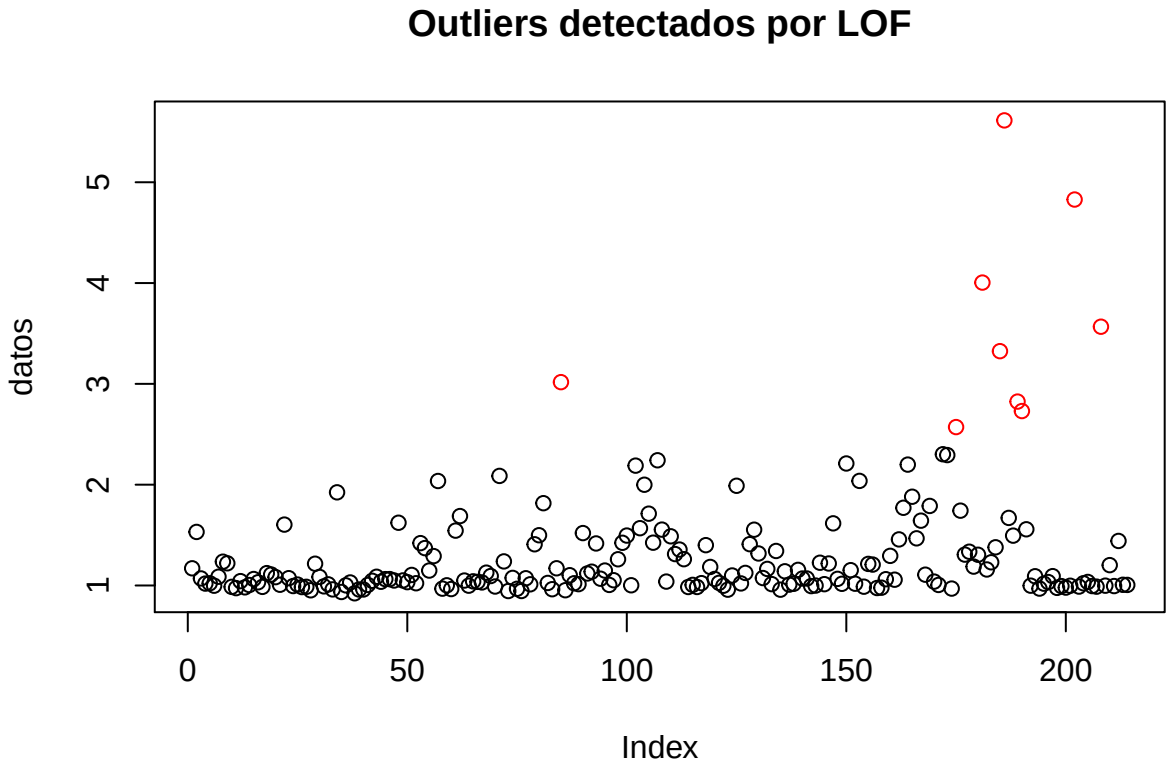
```
##  [1] 5.613473 4.828503 4.004665 3.566978 3.324387 3.017120 2.824255
##  [8] 2.730228 2.571385 2.301961 2.293591 2.242070 2.210085 2.199092
## [15] 2.188828 2.086983 2.037419 2.036413 1.999504 1.989110
```

Observamos diferencias notables entre los valores hasta el valor número 10 (2.30), a partir del cual las diferencias son mucho menores por lo que podemos asumir que los datos del top 9 son realmente outliers, es decir, los datos:

```
numero.de.outliers = 9
indices.de.lof.top.outliers = indices.de.lof.outliers[1:numero.de.outliers]
indices.de.lof.top.outliers
```

```
## [1] 186 202 181 208 185  85 189 190 175
```

```
MiPlot_Univariate_Outliers(lof.scores,indices.de.lof.top.outliers,
                           "Outliers detectados por LOF")
```

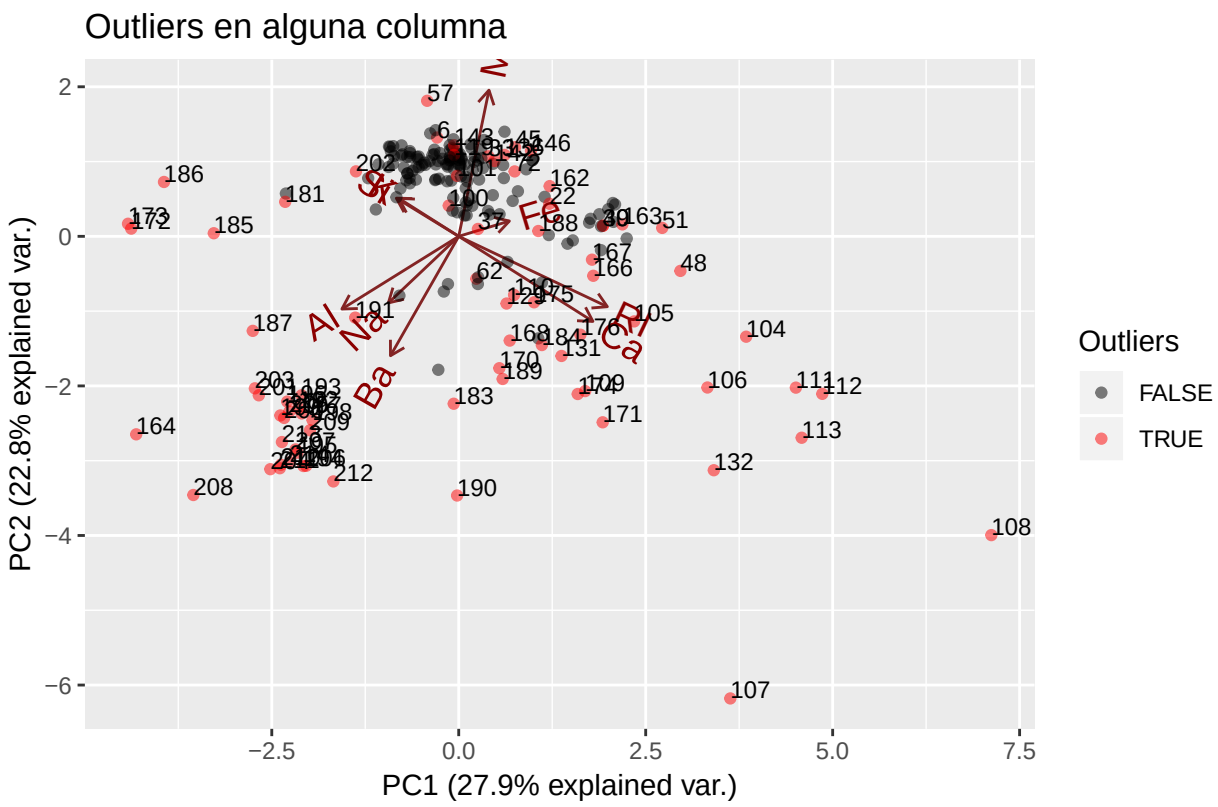


Ahora visualizaremos los outliers univariantes que se detectaron en apartados anteriores mediante la función proporcionada "MiBiPlot_Multivariate_Outliers". Primero obtenemos los outliers univariantes:

```
vector.claves.outliers.IQR.en.alguna.columna = vector_claves_outliers_IQR_en_alguna_columna(mydata)
vector.es.outlier.IQR.en.alguna.columna = vector_es_outlier_IQR_en_alguna_columna(mydata)
```

Dibujando el "biplot" de estos outliers:

```
MiBiPlot_Multivariate_Outliers(mydata,vector.es.outlier.IQR.en.alguna.columna,
                               "Outliers en alguna columna")
```



Descubrimos porqué se detectaron tantos outliers univariantes entonces ya que si no pertenecían al grupo principal, aún cuando sí que pertenecen a otros grupos homogéneos, eran clasificados como anomalías.

Por último se estudia la diferencia entre los outliers univariantes que se detectaron y los outliers encontrados por LOF:

```
indices.de.outliers.multivariantes.LOF.pero.no.1variantes =
  setdiff(indices.de.lof.top.outliers,vector.claves.outliers.IQR.en.alguna.columna)
indices.de.outliers.multivariantes.LOF.pero.no.1variantes
```

```
## [1] 85
```

Descubrimos que solamente el dato 85 ha sido detectado como un outlier multivariante por el método LOF y no lo fue detectado como univariante, por lo tanto es el outlier multivariante más "puro" encontrado por LOF.

Clustering con outliers

En este tipo de detección de outliers, en función de cómo se relacionen los datos con los clusters que se detecten, sabremos si es una anomalía o no.

Lo primero es establecer el número de clusters. Dado que originalmente este dataset 'Glass' trabajaba con 7 tipos diferentes de cristales, entonces el mejor número de clusters será 7:

```
set.seed(99)
numero.de.clusters = 7
```

Empezamos aplicando "kmeans" con nuestros datos normalizados:

```
modelo.kmeans <- kmeans(mydata.scaled,numero.de.clusters)
indices.clustering.glass <- modelo.kmeans$cluster
indices.clustering.glass
```

```
##   [1] 2 7 7 7 7 3 7 7 2 7 3 7 3 3 7 7 7 2 2 7 3 2 7 7 7 7 7 7 7 7 3 7 3 7 7
##  [36] 7 2 7 2 2 7 7 7 2 3 2 1 2 2 2 2 1 7 7 7 3 3 7 7 1 2 2 2 2 2 2 1 1 1 2
##  [71] 1 1 7 7 7 7 7 7 1 7 7 7 7 7 7 7 7 7 7 1 7 3 7 7 7 1 3 7 7 3 7 3 2 2
## [106] 5 5 5 2 6 5 5 5 1 7 7 1 7 3 7 7 3 7 7 2 1 7 1 1 1 2 5 7 1 7 3 1 7 7 7
## [141] 7 1 3 7 3 3 7 7 7 7 3 2 2 7 7 7 7 2 7 1 7 1 1 4 7 5 5 6 6 6 5 7 7 5 3
## [176] 3 2 2 2 2 6 6 6 2 6 7 4 2 2 4 6 6 6 4 4 6 6 6 6 6 6 7 6 4 6 4 4 4 6 6
## [211] 4 4 4 4
```

```
centroides.normalizados.glass <- modelo.kmeans$centers
centroides.normalizados.glass
```

```
##           RI          Na         Mg          Al         Si           K
## 1  0.2350154  0.06226422  0.5144100 -0.39822086 -0.4519617 -0.06615083
## 2  0.9305549  0.60642044  0.3540920 -0.85717628 -0.8694574 -0.47762100
## 3 -0.3922049 -0.68579714  0.3501316 -0.08297777  0.4731247  0.09917796
## 4 -0.1970775  1.51640704 -1.4467755  1.19192798 -0.1054861 -0.18891668
## 5  2.4903196 -1.67059535 -1.6348837 -0.23014927 -0.7840701 -0.27984192
## 6 -0.6576934  1.18581410 -1.7191884  1.27967520  1.0150572 -0.59785049
## 7 -0.4701674 -0.38768641  0.4627930  0.02401510  0.2240645  0.39682495
##            Ca         Ba         Fe
## 1 -0.05041285 -0.3074555  1.1328349
## 2  0.43642021 -0.3073584 -0.4282854
## 3 -0.19583393 -0.3083300  1.9761745
## 4 -0.71080035  3.1535991 -0.4666614
## 5  3.15901601  0.2238789  0.3105889
## 6  0.22966385  0.4629575 -0.4677892
## 7 -0.46412832 -0.3213057 -0.5013247
```

```
unique(indices.clustering.glass)
```

```
## [1] 2 7 3 1 5 6 4
```

Observamos que para los 7 clusters se han asociado datos.

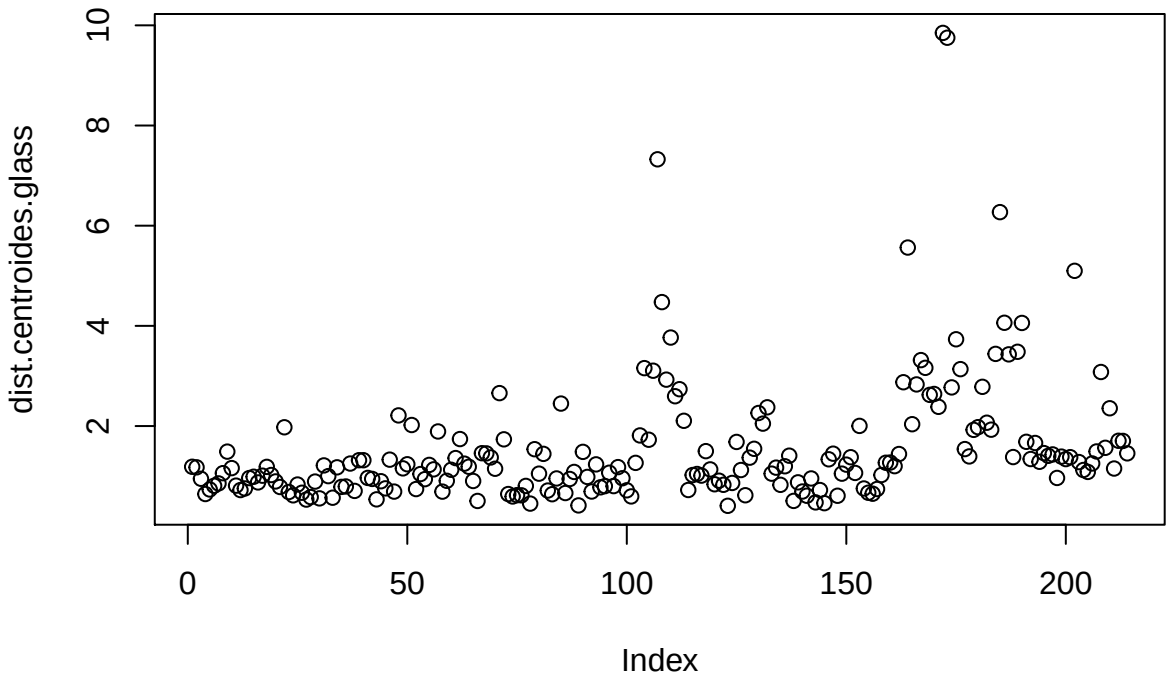
Ahora procedemos a calcular la distancia Euclidea de cada dato a su centroide. Ordenando a continuación las distancias, sabremos aquellos datos que están más alejados de su propio cluster y por lo tanto serán posiblemente los outliers al no pertenecer realmente a ningún cluster:

```
distancias_a_centroides = function (datos.normalizados,
                                    indices.asignacion.clustering,
```

```
                                    datos.centroides.normalizados){

  sqrt(rowSums((datos.normalizados -
                 datos.centroides.normalizados[indices.asignacion.clustering,])^2))
}

dist.centroides.glass <- distancias_a_centroides(mydata.scaled, indices.clustering.glass,
                                                 centroides.normalizados.glass)
plot(dist.centroides.glass)
```



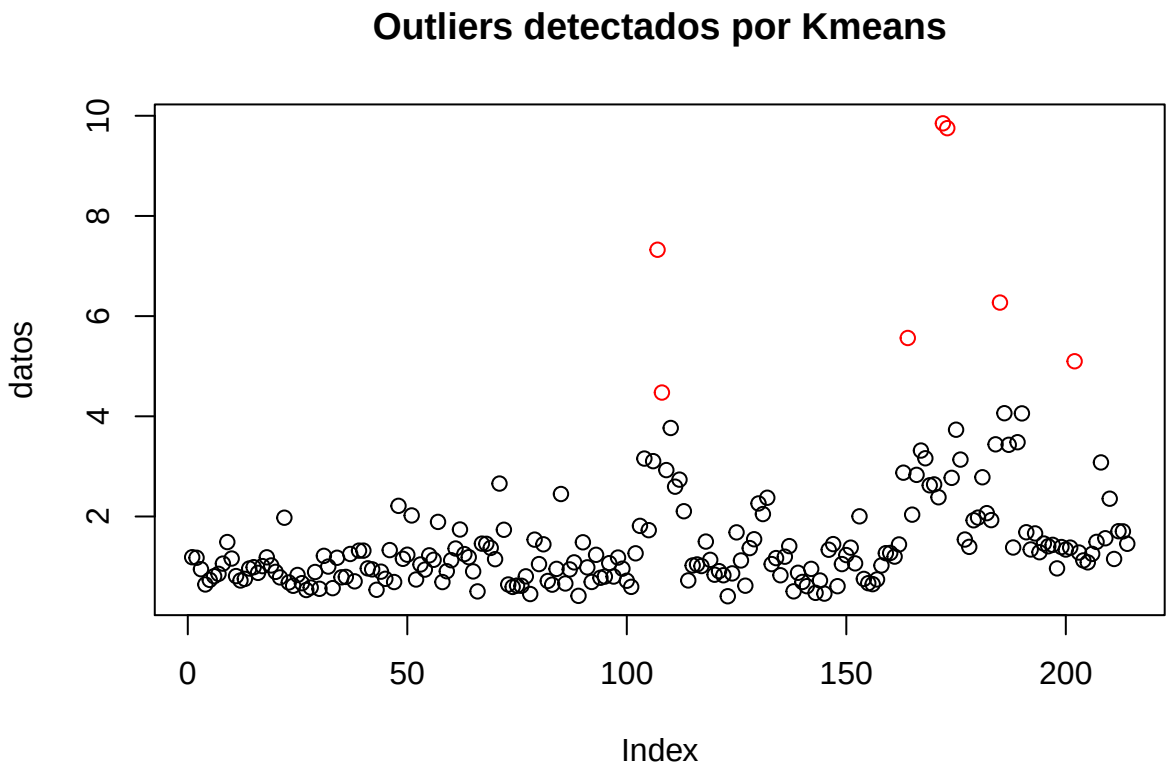
Al igual que se hizo en LOF estudiamos las distancias calculadas. En el gráfico podemos diferenciar hasta 7 datos como outliers ya que a continuación sí que hay más datos que están a distancias más similares.

```
top.outliers.glass <- order(dist.centroides.glass, decreasing = TRUE)[1:20]
dist.centroides.glass[top.outliers.glass]
```

```
##  [1] 9.848755 9.752483 7.325553 6.270583 5.563447 5.098866 4.474421
##  [8] 4.061917 4.058152 3.766873 3.731922 3.482376 3.440665 3.430892
## [15] 3.316308 3.162163 3.156313 3.135647 3.104494 3.078696
```

Observamos que efectivamente hasta el dato 8 (4.06) las distancias difieren más entre ellas, sugiriéndonos que son los primero 7 los que son outliers:

```
top.outliers.glass <- top.outliers.glass[1:7]
MiPlot_Univariate_Outliers(dist.centroides.glass,top.outliers.glass,
                           "Outliers detectados por Kmeans")
```



Vamos a encapsular este proceso en una función:

```
top_clustering_outliers = function(datos.normalizados, indices.asignacion.clustering,
                                   datos.centroides.normalizados, numero.de.outliers){

  distancias <- distancias_a_centroides(datos.normalizados, indices.asignacion.clustering,
                                        datos.centroides.normalizados)
  indices <- order(distancias, decreasing = TRUE)[1:numero.de.outliers]
  distancias <- distancias[indices]
  milista <- list(distancias,indices)
  names(milista) <- c("distancias","indices")
  milista

}

top.outliers.kmeans <- top_clustering_outliers(mydata.scaled, indices.clustering.glass,
                                               centroides.normalizados.glass, 7)
top.outliers.kmeans$indices
```

```
## [1] 172 173 107 185 164 202 108
```

```
top.outliers.kmeans$distancias
```

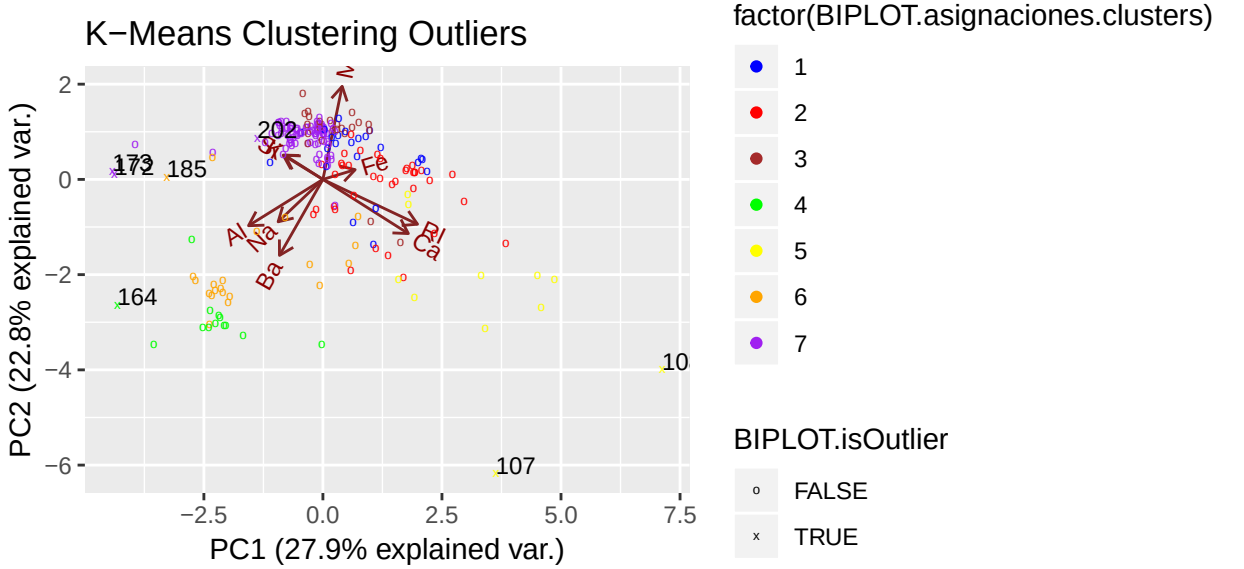
```
## [1] 9.848755 9.752483 7.325553 6.270583 5.563447 5.098866 4.474421
```

Se obtiene el mismo resultado.

Ahora vamos a dibujar estos outliers detectados mediante clustering utilizando "biplot":

```
numero.de.datos    = nrow(mydata)
is.kmeans.outlier = rep(FALSE, numero.de.datos)
is.kmeans.outlier[top.outliers.kmeans$indices] = TRUE


BIPLOT.isOutlier              = is.kmeans.outlier
BIPLOT.cluster.colors         = c("blue","red","brown","green","yellow","orange","purple")
BIPLOT.asignaciones.clusters = indices.clustering.glass
MiBiPlot_Clustering_Outliers(mydata, "K-Means Clustering Outliers")
```



Podemos visualizar los datos cuyos colores indican el cluster al que pertenecen y etiquetados con su índice los outliers detectados, situados éstos muy alejados de sus clusters respectivos.

Ahora vamos a revertir la normalización z-score con la que están construidos los centroides. Para ello empezamos calculando la media para cada columna:

```
mis.datos.media <- colMeans(mydata)
mis.datos.media
```

```
##          RI          Na          Mg          Al          Si           K
##  1.51836542 13.40785047  2.68453271  1.44490654 72.65093458  0.49705607
##          Ca          Ba          Fe
##  8.95696262  0.17504673  0.05700935
```

Igualmente calculamos las desviación típica para cada variable:

```
mis.datos.desviaciones <- apply(mydata,2,sd)
mis.datos.desviaciones
```

```
##          RI          Na          Mg          Al          Si           K
## 0.003036864 0.816603556 1.442407845 0.499269646 0.774545795 0.652191846
##          Ca          Ba          Fe
## 1.423153487 0.497219261 0.097438701
```

Ahora multiplicamos cada centroide por su desviación:

```
result <- sweep(centroides.normalizados.glass,2,mis.datos.desviaciones,"*")
result
```

```
##              RI          Na         Mg          Al          Si           K
## 1  0.0007137099  0.05084518  0.7419890 -0.19881959 -0.35006501 -0.04314303
## 2  0.0028259683  0.49520509  0.5107451 -0.42796210 -0.67343458 -0.31150052
## 3 -0.0011910727 -0.56002438  0.5050325 -0.04142828  0.36645672  0.06468306
## 4 -0.0005984975  1.23830338 -2.0868404  0.59509346 -0.08170381 -0.12320992
## 5  0.0075627613 -1.36421410 -2.3581691 -0.11490654 -0.60729822 -0.18251062
## 6 -0.0019973253  0.96834001 -2.4797708  0.63890298  0.78620828 -0.38991322
## 7 -0.0014278344 -0.31658610  0.6675363  0.01199001  0.17354818  0.25880599
##            Ca         Ba          Fe
## 1 -0.07174523 -0.1528728  0.11038196
## 2  0.62109294 -0.1528245 -0.04173157
## 3 -0.27870175 -0.1533076  0.19255587
## 4 -1.01157800  1.5680302 -0.04547088
## 5  4.49576466  0.1113169  0.03026338
## 6  0.32684691  0.2301914 -0.04558077
## 7 -0.66052584 -0.1597594 -0.04884843
```

Y ya finalmente sumamos las medias que se habían calculado anteriormente para finalmente invertir la normalización y obtener los centroides:

```
sinNormalizar = sweep(result,2,mis.datos.media,"+")
sinNormalizar
```

```
##         RI       Na        Mg       Al       Si         K        Ca
## 1 1.519079 13.45870 3.4265217 1.246087 72.30087 0.4539130  8.885217
## 2 1.521191 13.90306 3.1952778 1.016944 71.97750 0.1855556  9.578056
## 3 1.517174 12.84783 3.1895652 1.403478 73.01739 0.5617391  8.678261
## 4 1.517767 14.64615 0.5976923 2.040000 72.56923 0.3738462  7.945385
## 5 1.525928 12.04364 0.3263636 1.330000 72.04364 0.3145455 13.452727
## 6 1.516368 14.37619 0.2047619 2.083810 73.43714 0.1071429  9.283810
## 7 1.516938 13.09126 3.3520690 1.456897 72.82448 0.7558621  8.296437
##           Ba         Fe
## 1 0.02217391 0.16739130
## 2 0.02222222 0.01527778
## 3 0.02173913 0.24956522
## 4 1.74307692 0.01153846
## 5 0.28636364 0.08727273
## 6 0.40523810 0.01142857
## 7 0.01528736 0.00816092
```

A continuación vamos a calcular la distancia de cada punto a su centroide utilizando la distancia de Mahalanobis y determinar así los "mayores" outliers. Para ello se utilizará el método "top_clustering_outliers_distancia_mahalanobis". Al igual que se hizo anteriormente, los datos son minimamente modifcados para evitar una división por 0 en el proceso de "cov.rob":

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:EnvStats':
##
##     boxcox
```

```r
top_clustering_outliers_distancia_mahalanobis = function(datos,
                                                         indices.asignacion.clustering,
                                                         numero.de.outliers){

  cluster.ids = unique(indices.asignacion.clustering)
  k           = length(cluster.ids)
  seleccion   = sapply(1:k, function(x) indices.asignacion.clustering == x)


  # Usando medias y covarianzas:
  #lista.matriz.de.covarianzas   = lapply(1:k, function(x) cov(datos[seleccion[,x],]))
  #lista.vector.de.medias        = lapply(1:k, function(x) colMeans(datos[seleccion[,x],]))


  # Usando la estimaci?n robusta de la media y covarianza: (cov.rob del paquete MASS:
  lista.matriz.de.covarianzas   = lapply(1:k, function(x) cov.rob(datos[seleccion[,x],])$cov)
  lista.vector.de.medias        = lapply(1:k, function(x) cov.rob(datos[seleccion[,x],])$center)


  mah.distances   = lapply(1:k,
                           function(x) mahalanobis(datos[seleccion[,x],],
                                                   lista.vector.de.medias[[x]],
                                                   lista.matriz.de.covarianzas[[x]]))

  todos.juntos = unlist(mah.distances)
  todos.juntos.ordenados = names(todos.juntos[order(todos.juntos, decreasing=TRUE)])
  indices.top.mah.outliers = as.numeric(todos.juntos.ordenados[1:numero.de.outliers])


  list(distancias = mah.distances[indices.top.mah.outliers]  , indices =
         indices.top.mah.outliers)
}

random3 <- runif(214,0,0.00001)
random8 <- runif(214,0,0.00001)
random9 <- runif(214,0,0.00001)
mydataMod <- data.frame(mydata)
mydataMod[,3] <- mydataMod[,3] + random3
mydataMod[,8] <- mydataMod[,8] + random8
mydataMod[,9] <- mydataMod[,9] + random9
top.clustering.outliers.mah = top_clustering_outliers_distancia_mahalanobis(mydataMod,
                                                         indices.clustering.glass,9)
```

Estos son los top 9 outliers encontrados:

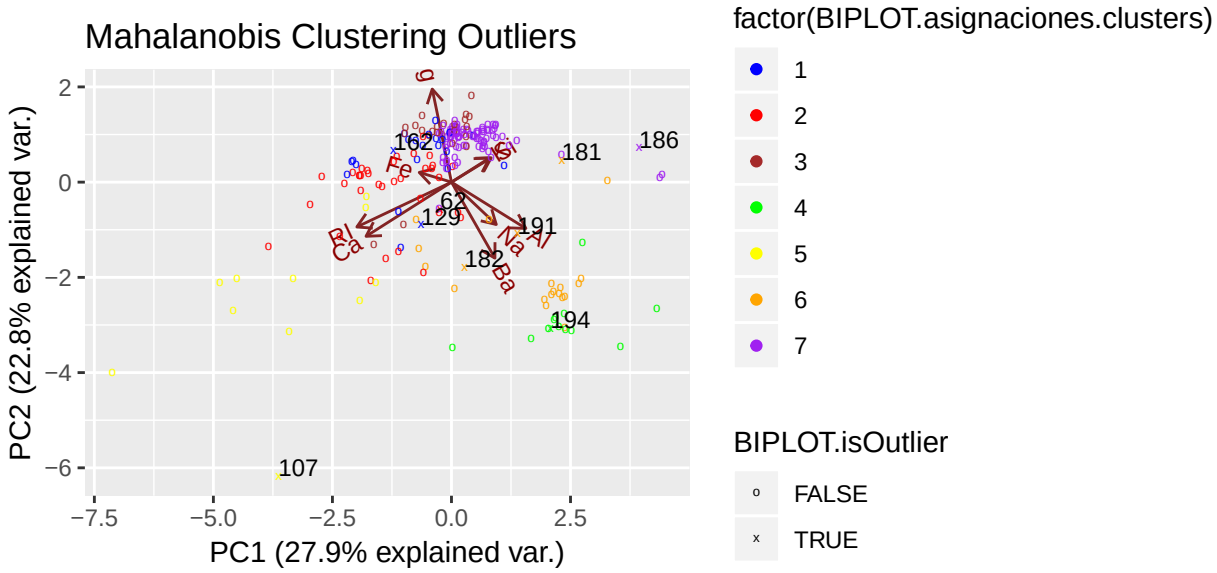
```
top.clustering.outliers.mah$indices
```

```
## [1] 107 191 181 182 186  62 129 194 162
```

Podemos representarlos graficamente mediante "MiBiplot":

```
numero.de.datos = nrow(mydataMod)
is.kmeans.outlier.mah = rep(FALSE, numero.de.datos)
is.kmeans.outlier.mah[top.clustering.outliers.mah$indices] = TRUE

BIPLOT.isOutlier             = is.kmeans.outlier.mah
BIPLOT.cluster.colors        = c("blue","red","brown","green","yellow","orange","purple")
BIPLOT.asignaciones.clusters = indices.clustering.glass
MiBiPlot_Clustering_Outliers(mydataMod, "Mahalanobis Clustering Outliers")
```



Ahora vamos a calcular de nuevo los outliers pero esta vez utilizando la distancia relativa, es decir, en función de la distancia que tengan los datos de un cluster a su centroide. Se utiliza el método proporcionado "top_clustering_outliers_distancia_relativa":

```
top_clustering_outliers_distancia_relativa = function(datos.normalizados,
                                                      indices.asignacion.clustering,
                                                      datos.centroides.normalizados,
                                                      numero.de.outliers){

  dist_centroides = distancias_a_centroides (datos.normalizados,
                                             indices.asignacion.clustering,
                                             datos.centroides.normalizados)

  cluster.ids = unique(indices.asignacion.clustering)
  k           = length(cluster.ids)

  distancias.a.centroides.por.cluster = sapply(1:k ,
        function(x) dist_centroides [indices.asignacion.clustering  == cluster.ids[x]])
```

```
  distancias.medianas.de.cada.cluster   = sapply(1:k ,
        function(x) median(dist_centroides[[x]]))

  todas.las.distancias.medianas.de.cada.cluster  =
    distancias.medianas.de.cada.cluster[indices.asignacion.clustering]
  ratios = dist_centroides   /  todas.las.distancias.medianas.de.cada.cluster

  indices.top.outliers           = order(ratios, decreasing=T)[1:numero.de.outliers]

  list(distancias = ratios[indices.top.outliers]  , indices = indices.top.outliers)
}


top.outliers.kmeans.distancia.relativa = top_clustering_outliers_distancia_relativa(
  mydata.scaled, indices.clustering.glass, centroides.normalizados.glass, 9)
```

Estos son los outliers encontrados así como sus distancias:

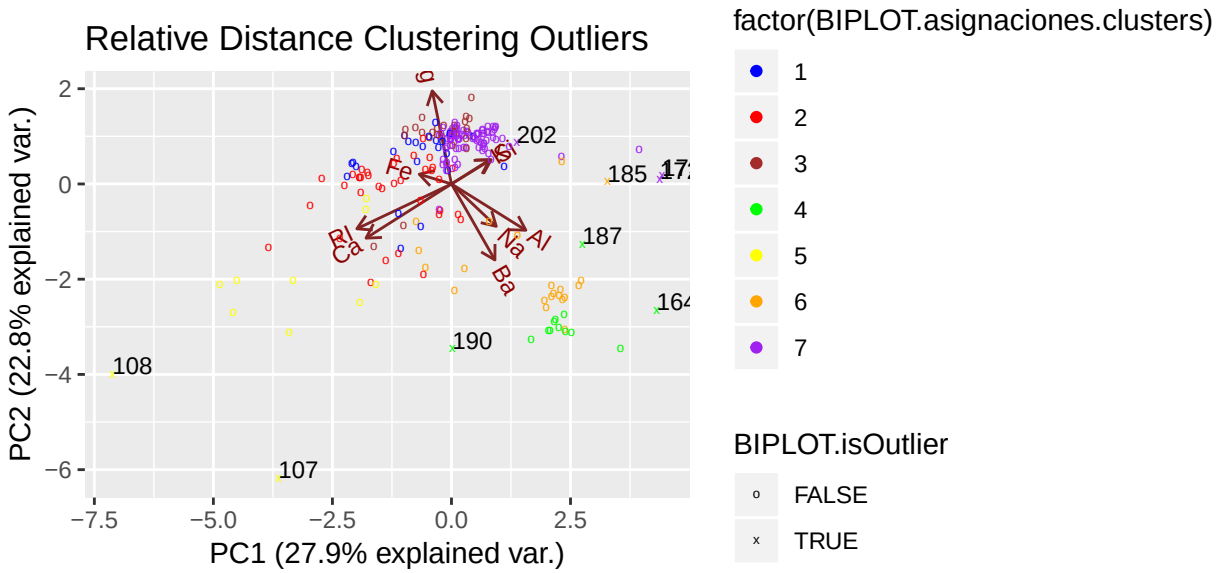
```
top.outliers.kmeans.distancia.relativa
```

```
## $distancias
## [1] 11.526130 11.413462  9.986301  8.655961  7.764909  6.313929  6.099596
## [8]  5.967271  5.337998
##
## $indices
## [1] 172 173 107 164 185 190 108 202 187
```

Representándolo graficamente:

```
numero.de.datos = nrow(mydata)
is.kmeans.outlier.rel = rep(FALSE, numero.de.datos)
is.kmeans.outlier.rel[top.outliers.kmeans.distancia.relativa$indices] = TRUE

BIPLOT.isOutlier             = is.kmeans.outlier.rel
BIPLOT.cluster.colors        = c("blue","red","brown","green","yellow","orange","purple")
BIPLOT.asignaciones.clusters = indices.clustering.glass
MiBiPlot_Clustering_Outliers(mydataMod, "Relative Distance Clustering Outliers")
```



Conclusión

Para concluir vamos a comparar los outliers detectados por los diferentes métodos. Se han evitado los outliers univariantes que se detectaron en un principio pues, como pudimos comprobar, la mitad los datos se detectaban como anomalías:

```
## K-means
## [1] 107 108 164 172 173 185 202
## K-means Distancia relativa
## [1] 107 108 164 172 173 185 187 190 202
## Clustering Mahalanobis
## [1]  62 107 129 162 181 182 186 191 194
## LOF
## [1]  85 175 181 185 186 189 190 202 208
## K-means vs K-means relativa:  0.78 %
## K-means vs Clustering Mahalanobis:  0.11 %
## K-means vs LOF:  0.22 %
## K-means relativa vs Clustering Mahalanobis:  0.11 %
## K-means relativa vs LOF:  0.33 %
## Clustering Mahalanobis vs LOF:  0.22 %
```

Descubrimos que los outliers detectados por k-means en sus dos variantes son comunes en un 78%, es decir, utilizar uno u otro influye en que cambien 2 outliers. Ambas variantes comparadas con el método por densidad LOF tienen en común un 22-33% de los outliers detectados, encontrando entonces varios outliers diferentes. Si comparamos ambas versiones de Kmeans con Clustering Mahalanobis sólo tienen en común un único outlier mientras que LOF detectó 3 outliers que también lo hizo Clustering con la distancia de Mahalanobis.