Master en Ciencia de Datos e Ingenieria de Computadores

Detección de Anomalías

Autor:
Nicolás Delgado Guerrero

Índice

1. Resumen. **2**

2. Análisis Exploratorio de Datos. **3**

3. Outliers Univariantes IQR. **6**
3.1 Trabajando con una columna. 6
3.2 Trabajando con varias columnas simultaneamente. 9

4. Outliers Univariantes Test Estadísticos. **12**
4.1 Test de Grubbs. 12
4.2 Test de Rosner. 15

5. Outliers Multivariantes Mahalanobis. **18**
5.1 Test de tipo a). 19
5.2 Test de tipo b). 20

6. Outliers Multivarientes LOF. **21**
6.1 Ouliers Basados en su Distancia. 21
6.2 Análisis de los Outliers. 24
6.2.1 Ouliers con valor anormal en alguna variable. 24
6.2.2 Outliers “Puros”. 26

7. Ouliers Multivariantes Clustering. **27**
7.1 Alternativa A. 27
7.1.1 Distancias respecto Centroides. 27
7.1.2 Distancias respecto Mediodes. 35
7.2 Alternativa B. 36
7.3 Alternativa C. 39

1. Resumen.

En el presente trabajo haremos uso de un conjunto de datos que reune información acerca de todos los Pokemon de la famosa saga de videojuegos. Buscaremos anomalías en los stats de los mismos para encontrar Pokemon interesantes para jugar a nivel competitivo. Los stats son los puntos de poder que tiene el Pokemon en distintas características como pueden ser el ataque o la defensa.

Primero analizaremos los outliers univariantes. Nos centraremos en técnicas estadísticas, método IQR y tests estadísticos. Para ello deberemos suponer que la distribución de cada variable es normal aunque también podemos aplicar los métodos para variables unimodales.

Veremos que hay outliers que presentan valores extraños en la combinación de sus variables, para encontrarlos haremos un análisis de outliers multivariante. Usaremos técnicas paramétricas basadas en la distancia de Mahalanobis y no paramétricas, LOF y Clustering.

2. Análisis Exploratorio de Datos.

Vamos a cargar el Dataset y ver que tipo de datos nos encontramos.

```
pokemon =read.csv("pokemon-pokemon-with-stats-QueryResult.csv")
str(pokemon)
```

```
## 'data.frame':    800 obs. of  13 variables:
##  $ column_a: int  1 2 3 3 4 5 6 6 6 7 ...
##  $ column_b: Factor w/ 800 levels "Abomasnow","AbomasnowMega Abomasnow",..: 81 330 746 747 103 104 100 101 102 666 ...
##  $ column_c: Factor w/ 18 levels "Bug","Dark","Dragon",..: 10 10 10 10 7 7 7 7 7 18 ...
##  $ column_d: Factor w/ 19 levels "","Bug","Dark",..: 15 15 15 15 1 1 9 4 9 1 ...
##  $ column_e: int  318 405 525 625 309 405 534 634 634 314 ...
##  $ column_f: int  45 60 80 80 39 58 78 78 78 44 ...
##  $ column_g: int  49 62 82 100 52 64 84 130 104 48 ...
##  $ column_h: int  49 63 83 123 43 58 78 111 78 65 ...
##  $ column_i: int  65 80 100 122 60 80 109 130 159 50 ...
##  $ column_j: int  65 80 100 120 50 65 85 85 115 64 ...
##  $ column_k: int  45 60 80 80 65 80 100 100 100 43 ...
##  $ column_l: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ column_m: Factor w/ 2 levels "false","true": 1 1 1 1 1 1 1 1 1 1 ...
```

Vemos que tenemos 800 observaciones y un total de 13 variables. La primera variable o columna indica el número al que pertenece el Pokemon en la Pokedex nacional, se observa que no coincide con el número de la observación ya que hay algunas transformaciones en ciertos Pokemon que indican sus stats pero no se consideran una especie nueva, como el caso de MegaVenasaur. La segunda columna contiene un string con el nombre del pokemon. La columna c y d nos informa el tipo que tiene cada pokemon, en columna d habra un NA si el pokemon solo tiene un tipo. Desde la columna e hasta la k tenemos: suma de todos los stats, Hp o puntos de vida, ataque, defensa, ataque especial, defensa especial y velocidad respectivamente. La columna l nos dice a que generación pertenece el pokemon y la m si es legendario o no.

Lo primero que haremos será cambiar el nombre de las variables para acceder a ellas de forma más intuitiva. Para ello haremos uso de la biblioteca dplyr.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
pokemon = pokemon %>% rename(Num_Pokedex = column_a, Nom_Pokemon = column_b,
                     Tipo1 = column_c, Tipo2 = column_d, Sum_Stats = column_e,
                     Hp = column_f,  Ataque = column_g, Defensa = column_h,
                     Ataque_Especial = column_i, Defensa_Especial = column_j,
                     Velocidad = column_k, Generacion = column_l,
                     Legendario = column_m)
head(pokemon)
```

```
##   Num_Pokedex           Nom_Pokemon Tipo1  Tipo2 Sum_Stats Hp Ataque Defensa
## 1           1             Bulbasaur Grass Poison       318 45     49      49
## 2           2               Ivysaur Grass Poison       405 60     62      63
## 3           3              Venusaur Grass Poison       525 80     82      83
## 4           3 VenusaurMega Venusaur Grass Poison       625 80    100     123
## 5           4            Charmander  Fire              309 39     52      43
## 6           5            Charmeleon  Fire              405 58     64      58
##   Ataque_Especial Defensa_Especial Velocidad Generacion Legendario
## 1              65               65        45          1      false
## 2              80               80        60          1      false
## 3             100              100        80          1      false
## 4             122              120        80          1      false
## 5              60               50        65          1      false
## 6              80               65        80          1      false
```

Para nuestro estudio solo vamos a hacer uso de las variables numéricas, es decir los stats. Vamos a crear dos

conjunto de datos, ambos solo con el nombre del pokemon y sus stats, una de ellas normalizada y la otra sin normalizar los datos.

```
pokemon = pokemon[c(2,6:11)]
pokemon_normalizado = data.frame(pokemon$Nom_Pokemon,scale(pokemon[,2:7]))
```

Una hipótesis muy importante a la hora de trabajar con los datos es la normalidad. Si los datos se distribuyen como una variable aleatoria normal podremos hacer uso de muchas tecnicas estadísticas y podremos conocer más afondo el comportamiento de la variable.

Para ello usaremos el test de shapiro donde la hipótesis nula y alternativa serían las siguientes:

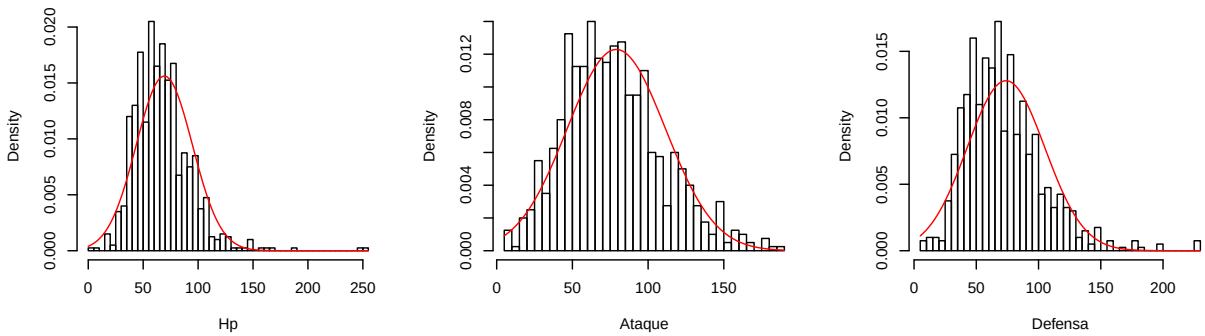
$$\begin{cases} H_0 : X \to \mathcal{N}(\mu, \sigma^2) \\ H_1 : X \nrightarrow \mathcal{N}(\mu, \sigma^2) \end{cases}$$

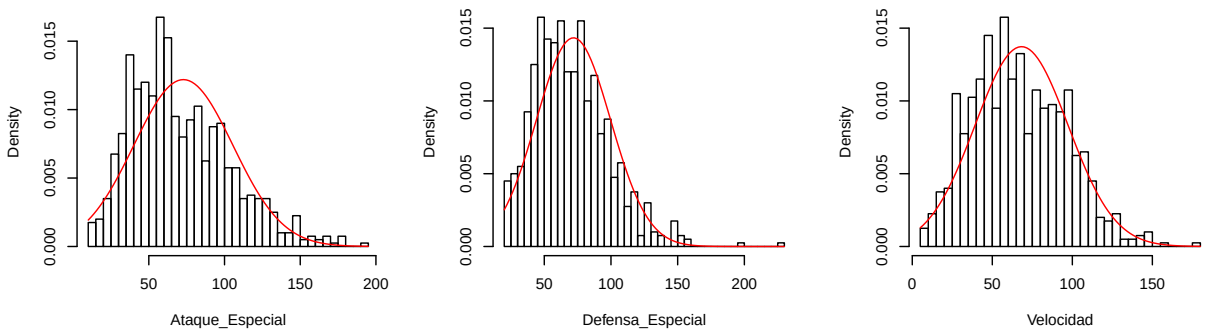
```
Tests=lapply(pokemon[,2:7], shapiro.test)
for (i in 2:7){
  cat(names(pokemon)[i],"p-valor test shapiro: ",Tests[[i-1]]$p.value,"\n")
}
```

```
## Hp p-valor test shapiro:  1.152364e-20
## Ataque p-valor test shapiro:  2.472154e-09
## Defensa p-valor test shapiro:  9.923173e-18
## Ataque_Especial p-valor test shapiro:  4.665141e-14
## Defensa_Especial p-valor test shapiro:  8.251792e-14
## Velocidad p-valor test shapiro:  1.309542e-07
```

En todos los casos rechazamos la hipótesis nula. Veamos como se comporta la función de densidad mediante unos histrogramas.

```
par(mfrow=c(1,3))
for (i in 2:7) {
  hist(pokemon[[i]],xlab = names(pokemon)[i],main = "",breaks = 50,freq=FALSE)
  curve(dnorm(x, mean=mean(pokemon[[i]]), sd=sd(pokemon[[i]])), add=TRUE, col="red")
}
```





Observamos que la distribución no es normal pero por lo menos no presentan varias modas. Al ser unimodal podremos usar distintas tecnicas a lo largo del trabajo, aunque eso sí, no serán tan significantes como lo serían bajo hipótesis de normalidad.

3. Outliers Univariantes IQR.

En este apartado usaremos tecnicas estadísticas para el computo de outliers con respecto una única característica, para ello haremos uso del método IQR el cual se basa en el cálculo del rango intercuartílico. Una hipótesis a resaltar para que estas técnicas tengan eficacia, es la normalidad de los datos. Podemos reducir restricciones a comprobar que los datos no presentan varias modas.

En los histogramas realizados en **2. Análisis Exploratorio de Datos** podemos observar que la mayor parte de área que recoge función de densidad, se localiza en una zona concreta y no presentan varios apuntamientos, por lo que concretamos que las variables son unimodales.

3.1 Trabajando con una columna.

En este parte trabajaremos solo con una variable, podemos seleccionar por ejemplo la variable que recoge los puntos de vida, Hp.

```
indice_columna = 2
```

Primero calcularemos los cuartiles uno, dos y tres y por último la distancia intercuartílica.

```
cuartil_primero = quantile(pokemon[,indice_columna], probs = 0.25)
cuartil_primero
```

```
## 25%
##  50
```

```
cuartil_segundo = quantile(pokemon[,indice_columna], probs = 0.5)
cuartil_segundo
```

```
## 50%
##  65
```

```
cuartil_tercero = quantile(pokemon[,indice_columna], probs = 0.75)
cuartil_tercero
```

```
## 75%
##  80
```

```
iqr = IQR(pokemon[,indice_columna])
iqr
```

```
## [1] 30
```

Mediante los cuartíles y el rango intercuartílico podemos calcular los valores que particionan el rango de los datos para detectar los outliers.

```
extremo_superior_outlier_normal  = cuartil_tercero[[1]] + 1.5*iqr
extremo_superior_outlier_normal
```

```
## [1] 125
```

```
extremo_inferior_outlier_normal  = cuartil_primero[[1]] - 1.5*iqr
extremo_inferior_outlier_normal
```

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

```
extremo_superior_outlier_extremo = cuartil_tercero[[1]] + 3*iqr
extremo_superior_outlier_extremo
```

```
## [1] 170
```

```
extremo_inferior_outlier_extremo = cuartil_primero[[1]] - 3*iqr
extremo_inferior_outlier_extremo
```

```
## [1] -40
```

A priori ya podemos decir que no hay outliers extremos inferiores porque la variable Hp es estrictamente positiva.

Ahora calcularemos los vectores lógicos que nos indican con un True el pokemon que presenta un outlier en sus puntos de vida.

```
vector_es_outlier_normal = pokemon[,indice_columna] > extremo_superior_outlier_normal |
  pokemon[,indice_columna] < extremo_inferior_outlier_normal

vector_es_outlier_extremo = pokemon[,indice_columna] > extremo_superior_outlier_extremo |
  pokemon[,indice_columna] < extremo_inferior_outlier_extremo
```

Con estos dos vectores podemos acceder a las posiciones en la que se encuentran los outliers y ver de que pokemon se tratan. Primero calcularemos un dataframe que contenga todos los outliers normales. Guardaremos en un vector el nombre y los valores de cada pokemon que presenta un outlier.

```
claves_outliers_normales = which(vector_es_outlier_normal == TRUE)
claves_outliers_normales
```

```
##  [1]  46 122 143 146 156 218 262 314 317 322 351 352 474 496 545 546 656 793 794
```

```
data_frame_outliers_normales = pokemon[claves_outliers_normales,]
head(data_frame_outliers_normales)
```

```
##     Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial Velocidad
## 46   Wigglytuff 140     70      45              85               50        45
## 122     Chansey 250      5       5              35              105        50
## 143      Lapras 130     85      80              85               95        60
## 146    Vaporeon 130     65      60             110               95        65
## 156     Snorlax 160    110      65              65              110        30
## 218   Wobbuffet 190     33      58              33               58        33
```

```
nombres_outliers_normales =as.character(data_frame_outliers_normales$Nom_Pokemon)

valores_outliers_normales = pokemon[claves_outliers_normales,indice_columna]
```

Hacemos lo mismo para los outliers extremos.

```
claves_outliers_extremos = which(vector_es_outlier_extremo == TRUE)
claves_outliers_extremos
```

```
## [1] 122 218 262
```

```
data_frame_outliers_extremos = pokemon[claves_outliers_extremos,]
data_frame_outliers_extremos
```

```
##     Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial Velocidad
## 122     Chansey 250      5       5              35              105        50
## 218   Wobbuffet 190     33      58              33               58        33
## 262     Blissey 255     10      10              75              135        55
```

```
nombres_outliers_extremos =as.character(data_frame_outliers_extremos$Nom_Pokemon)

valores_outliers_extremos = pokemon[claves_outliers_extremos,indice_columna]
```

Veamos la desviación de los outliers con respecto a la media de la columna. Para ello usaremos las claves de los outliers en el dataset con los datos normalizados previamente.

```
valores_normalizados_outliers_normales = pokemon_normalizado[claves_outliers_normales,]$Hp
valores_normalizados_outliers_normales
```

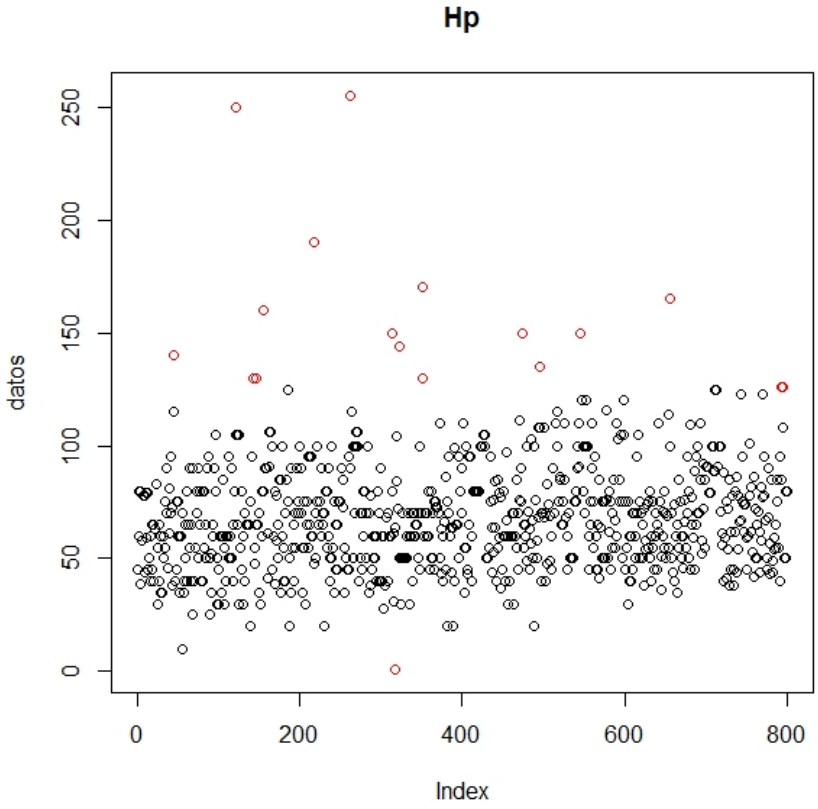
```
##  [1]  2.770400  7.078269  2.378776  2.378776  3.553649  4.728522  7.274081
##  [8]  3.162025 -2.673179  2.927050  2.378776  3.945273  3.162025  2.574588
## [15]  3.162025  3.162025  3.749461  2.222126  2.222126
```

```
valores_normalizados_outliers_extremos = pokemon_normalizado[claves_outliers_extremos,]$Hp
valores_normalizados_outliers_extremos
```

```
## [1] 7.078269 4.728522 7.274081
```

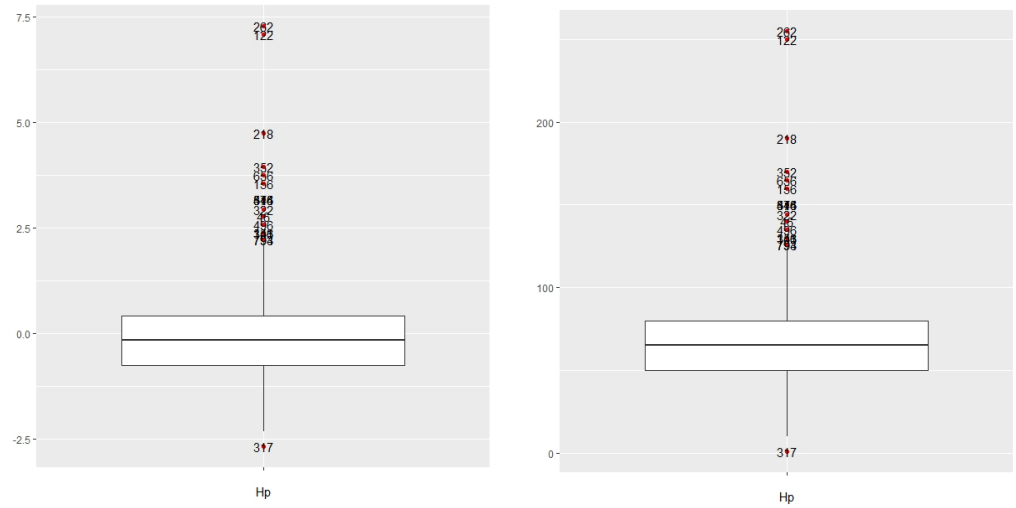
Veamos gráficamente los resultados obtenidos. Haremos uso de las funciones dadas en la clase de prácticas. Vamos a hacer un plot que en el eje x recoja la observaciones y el eje y el valor de sus puntos de vida, con ello podemos observar los outliers pintados en color rojo.

```
windows()
MiPlot_Univariate_Outliers(pokemon[,indice_columna],claves_outliers_normales,"Hp")
```



Podemos usar los diagramas de cajas para observar los outliers. Lo hacemos para los valores normalizados y sin normalizar para ver que la distribución no cambia.

```
windows()
MiBoxPlot_IQR_Univariate_Outliers(pokemon,indice_columna,coef = 1.5)
windows()
MiBoxPlot_IQR_Univariate_Outliers(pokemon_normalizado,indice_columna,coef = 1.5)
```



3.2 Trabajando con varias columnas simultaneamente.

En esta sección haremos un estudio análogo al anterior pero con todas las variables del dataset. El estudio sigue siendo univariante porque no vamos a centrarnos en si hay relación entre las distintas características.

Primero obtendremos los pokemon que presentan un outlier en alguno de sus stats.

```
indices_de_outliers_en_alguna_columna =
  unlist(lapply(2:ncol(pokemon),vector_claves_outliers_IQR,datos = pokemon))
```

Veamos los pokemon que presentan valores anómalos en alguna de sus variables.

```
tail(pokemon[indices_de_outliers_en_alguna_columna,])
```

```
##              Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 416               Regice  80     50     100             100              200
## 423.1 KyogrePrimal Kyogre 100    150      90             180              160
## 431.1 DeoxysDefense Forme  50     70     160              70              160
## 740               Florges  78     65      68             112              154
## 316               Ninjask  61     90      45              50               50
## 432     DeoxysSpeed Forme  50     95      90              95               90
##       Velocidad
## 416          50
## 423.1        90
## 431.1        90
## 740          75
## 316         160
## 432         180
```

Vemos que cada pokemon destaca en al menos uno de sus stats, por ejemplo Ninjask presenta una velocidad muy alta y los demás stats muy bajos. Regice destaca en su defensa especial y así con cada uno de los pokemon.

Podemos obtener lo mismo mediante una función dada en el archivo A3.

```
vector_claves_outliers_IQR_en_alguna_columna(pokemon[,2:7],coef = 1.5)
```

```
##  [1]  46 122 143 146 156 218 262 314 317 322 351 352 474 496 545 546 656 793 794
## [20] 164 233 425 427 430 495 712  88  99 104 224 225 231 333 334 415 425 431 457
## [39] 790  72 103 165 197 307 423 427 430 713 799 231 270 271 416 423 431 740 316
## [58] 432
```

Vamos a construir una matriz que nos informe para cada pokemon en que stat se presenta un valor anómalo, cada fila de la matriz es un pokemon y las columnas los valores de sus stats, habrá un TRUE si el valor de esa característica es un outlier. Contruiremos la matriz con ayuda de la función vector_es_outlier_IQR() dada en la clase de practicas.

```
frame_es_outlier = sapply(2:ncol(pokemon),vector_es_outlier_IQR,datos = pokemon)
as.character(pokemon[423,1])
```

```
## [1] "KyogrePrimal Kyogre"
```

```
frame_es_outlier[423,]
```

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

Observamos que KyogrePrimal tiene dos valores anómalos, en su ataque especial y defensa especial. Ahora crearemos un vector donde en la posición i-ésima se almacena el número de stats que presentan valores anómalos el pokemon de la posición i.

```
numero_total_outliers_por_columna = apply(frame_es_outlier, 1, sum)
```

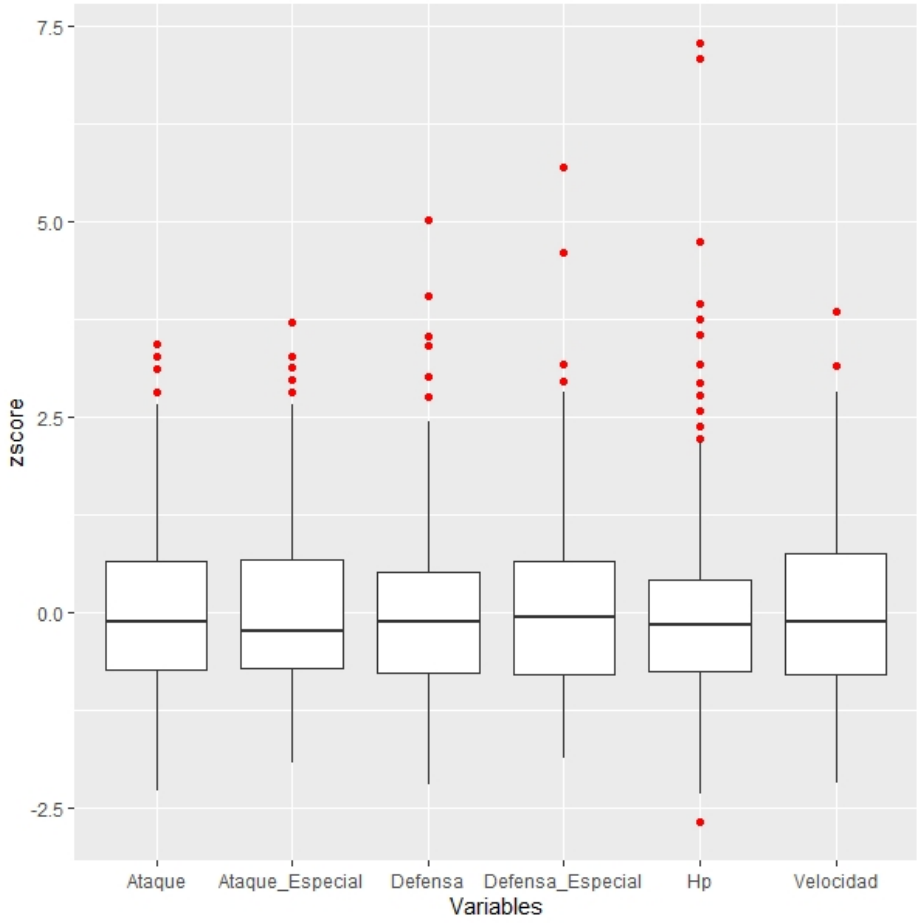
Comprobamos que en la posición 423 hay almacenado un 2.

```
numero_total_outliers_por_columna[423]
```

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

Por último vamos a hacer un plot de todas las variables a la vez para poder compararlas, usaremos la función MiBoxPlot_juntos_con_etiquetas() del guión de prácticas A3. Esta función ya normaliza los datos por lo que se podran comparar las distintas variables sin procupación de la unidad en la que se encuentren las características.

```
windows()
MiBoxPlot_juntos(pokemon[,2:7])
```



Es interesante comprobar que no hay apenas outliers inferiores, de hecho solo encontramos uno en la característica Hp. Eso ocurre porque los pokemon que presentan varias evoluciones en su primera etapa tienen un valor de los stats muy bajo, lo que hace que en el computo de los outliers se vean afectado y para los outlier inferiores se obtengan cotas muy bajas, incluso valores negativos. El único pokemon que presenta un valor outlier inferior es Shedinja y es un pokemon muy especial ya que solo tiene 1 punto de vida.

```
pokemon[pokemon[,1] == "Shedinja",]
```

```
##     Nom_Pokemon Hp Ataque Defensa Ataque_Especial Defensa_Especial Velocidad
## 317    Shedinja  1     90      45              30               30        40
```

4. Outliers Univariantes Test Estadísticos.

El problema que queremos solucionar es análago al de **3. Outliers Univariantes IQR**. Vamos a plantear contrastes de hipóteis y los resolveremos mediante distintos tests, con ello podremos saber si un valor está significativamente alejado de la media de esa variable. El nivel de confianza o significación lo estableceremos previamente.

4.1 Test de Grubbs.

El test de Grubbs es un test paramétrico, bajo hipótesis de normalidad de la variable a la que se aplica. Se quieren contrastar las siguientes hipóteis:

$$\begin{cases} H_0 : \text{No hay outliers en los datos} \\ H_1 : \text{Hay exactamente un outlier} \end{cases}$$

Para resolver el test se calcula el siguiente estadístico:

$$G = \frac{\text{máx}\,|X_i - \bar{X}|}{S},$$

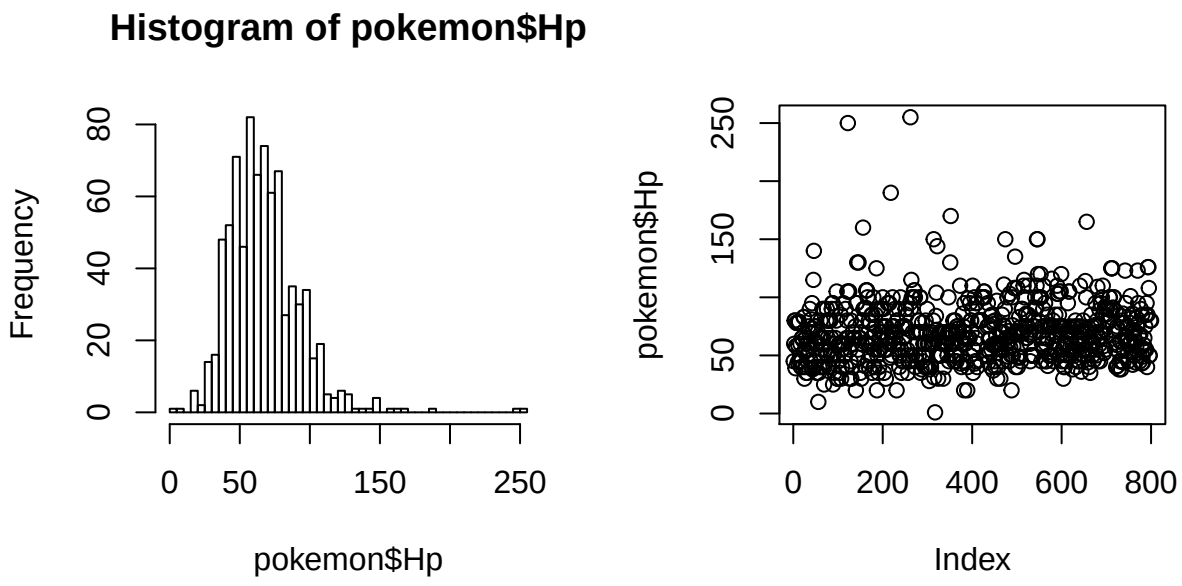
y se rechazará la hipótesis nula cuando:

$$G > \frac{(N-1)}{\sqrt{N}} \sqrt{\frac{t^2_{(\alpha/2N,N-2)}}{N-2+t^2_{(\alpha/2N,N-2)}}},$$

El valor de α es el nivel de confianza que fijamos a priori.

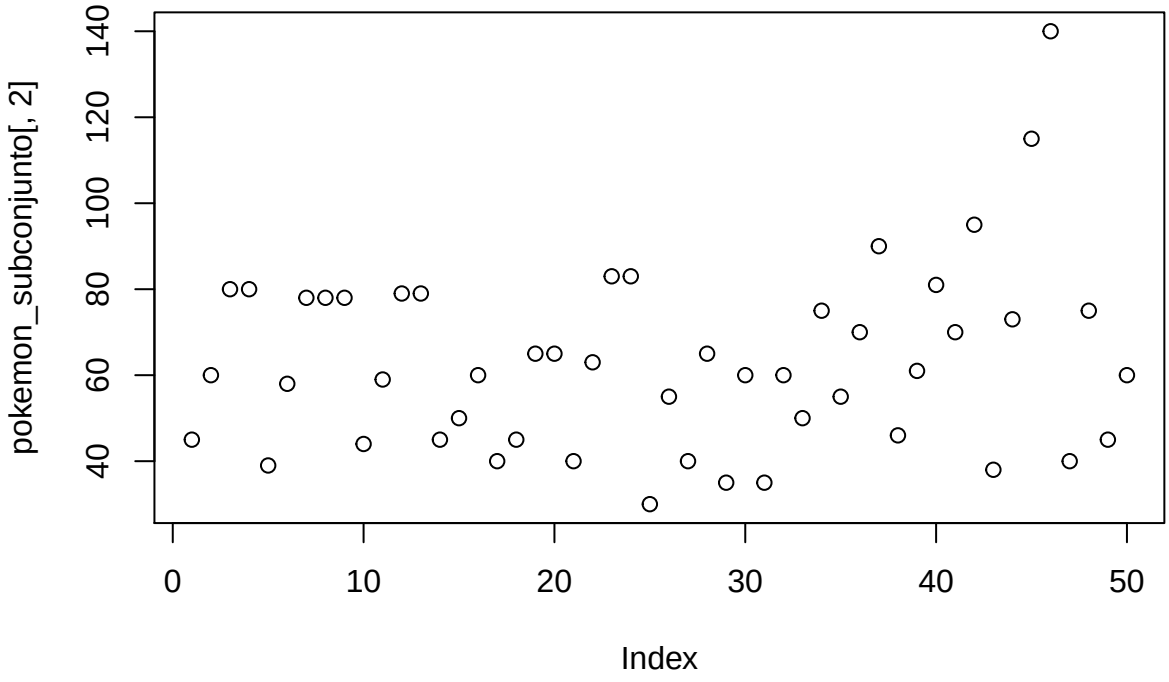
Vamos a aplicar el test para la variable Hp para comparar resultados con el apartado anterior. Primero observamos como se comporta la variable mediante un histograma y un scatterplot.

```
par(mfrow=c(1,2))
hist(pokemon$Hp, breaks = 50)
plot(pokemon$Hp)
```



Como ya sabiamos, se observan algunos valores que destacan superiormente respecto a la mayoría. Vamos a considerar primero un subconjunto de pokemon donde solo se presente un outlier para ver como funciona el test. Cuando hay varios outliers se produce lo que se conoce como masking, veremos como resolverlo más adelante.

```
pokemon_subconjunto = pokemon[1:50,]
plot(pokemon_subconjunto[,2])
```



Aplicamos el test a este subconjunto de datos.

```
library(outliers)
grubbs.test(pokemon_subconjunto$Hp, two.sided = TRUE)
```

```
##
##  Grubbs test for one outlier
##
## data:  pokemon_subconjunto$Hp
## G = 3.60889, U = 0.72878, p-value = 0.005281
## alternative hypothesis: highest value 140 is an outlier
```

El p-valor es pequeño por lo que hay evidencias estadísticas de rechazar la hipóteis nula y concluimos que hay un único valor anómalo. Sabemos que 140 es el valor que más se aleja de la media pero no sabemos a que índice pertenece, calculemoslo.

```
indice_de_outlier_Grubbs =
  order(abs(pokemon_subconjunto$Hp - mean(pokemon_subconjunto$Hp)),decreasing = TRUE)[1]
indice_de_outlier_Grubbs
```

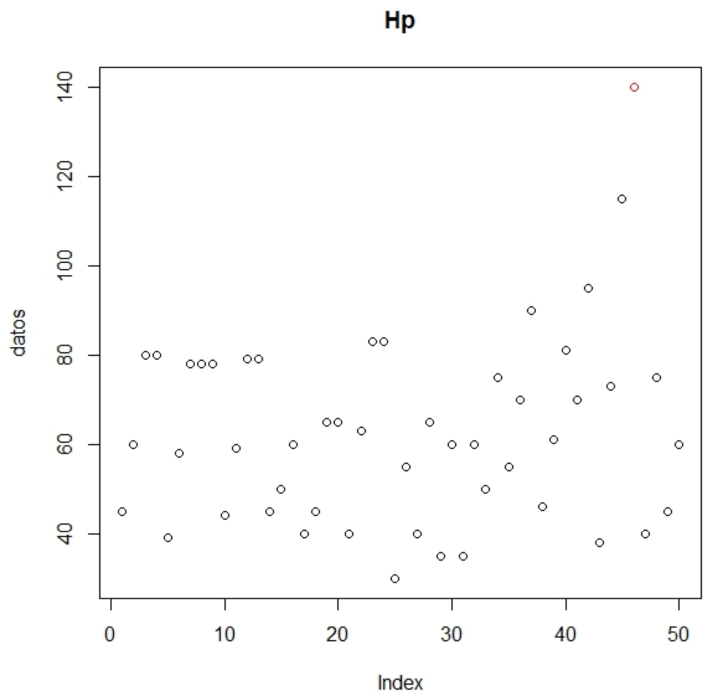
```
## [1] 46
```

```
valor_de_outlier_Grubbs = pokemon_subconjunto[indice_de_outlier_Grubbs,2]
valor_de_outlier_Grubbs
```

```
## [1] 140
```

Usamos la función MiPlot_Univariate_Outliers() para dibujar en el plot mediante color rojo el outlier que nos da el test estadístico.

```
windows()
MiPlot_Univariate_Outliers(pokemon_subconjunto[,2],indice_de_outlier_Grubbs,"Hp")
```



Podemos hacer todo lo anterior directamente con la función MiPlot_resultados_TestGrubbs().

```
windows()
MiPlot_resultados_TestGrubbs(pokemon_subconjunto[,2])
```

Obtenemos:

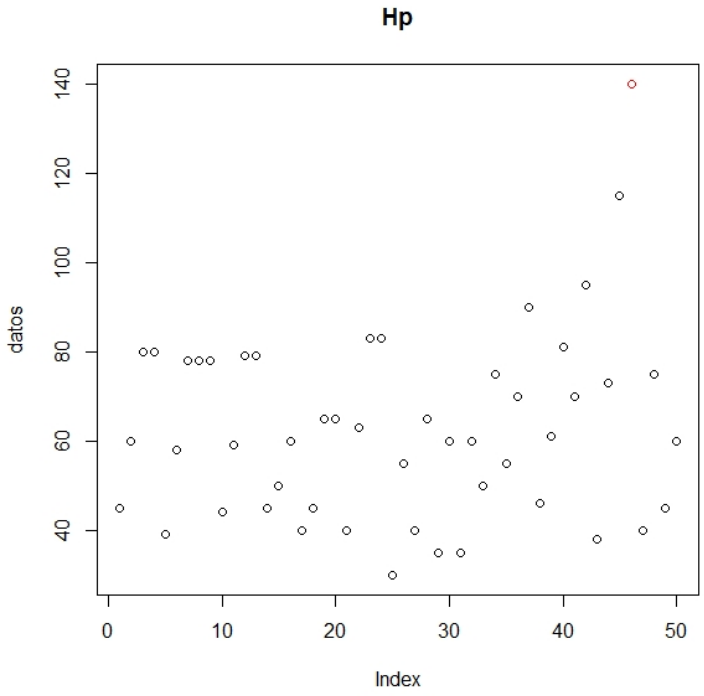
p.value: 0.005281324

Índice de outlier: 46

Valor del outlier: 140

Número de datos: 50

¿Quién es outlier?: FALSE TRUE FALSE FALSE FALSE FALSE



El test de Grubs se usa para detectar un único outlier de una distribución normal. La pregunta es, ¿qué ocurre cuando hay más de un outlier? El test puede fallar por un problema de masking, varios outliers hacen que el valor de la media cambie, ya que es una medida poco robusta. Con ello la distancia entre el outlier y la media no es significativamente grande para poder rechazar la hipótesis nula y no detectariamos el outlier. Veamos un ejemplo.

```
grubbs.test(pokemon[30:46,]$Hp, two.sided = TRUE)
```

```
##
##  Grubbs test for one outlier
##
## data:  pokemon[30:46, ]$Hp
## G = 2.52540, U = 0.57649, p-value = 0.07935
## alternative hypothesis: highest value 140 is an outlier
```

No tenemos evidencias estadísticas para rechazar la hipótesis aún sabiendo que entre esos pokemon hay uno que presenta una anomalía en sus puntos de vida.

4.2 Test de Rosner.

Cuando haya más de un outlier vamos a usar el test de Rosner, con él podemos detectar si hay menos de k valores anómalos, el valor de k se debe fijar previamente a la construcción del test. Al igual que el test de Grubbs trabajamos en inferencia paramétrica. Este test usa multiples comparaciones para detectar los varios outlier y aplica una corrección del FWER.

El FWER es un fenómeno que aparece cuando hacemos varios contrastes de hipótesis de forma pareada en un mismo conjunto de datos, esto conlleva un error que se va acumulando y para lidiar con él se penalida el α o nivel de significación del test.

El contraste de hipótesis que resuelve el test de Rosner es el siguiente:

$$\begin{cases} H_0 : \text{No hay outliers en los datos} \\ H_1 : \text{Hay menos de k outliers} \end{cases}$$

Vamos a aplicar el test de Rosner con k=10. El test previamente ordena los datos de mayor a menos distancia de la media.

```
library(EnvStats)
```

```
##
## Attaching package: 'EnvStats'

## The following objects are masked from 'package:stats':
##
##     predict, predict.lm

## The following object is masked from 'package:base':
##
##     print.default
```

```
test = rosnerTest(pokemon[,2],k=10)
```

Si imprimimos por pantalla el valor de test$all.stats$Outlier obtenemos un vector booleano con 10 valores más alejados de la media, nos indica con un TRUE si ese valor en una anomalía.

```
test$all.stats$Outlier
```

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

Vemos que el test nos indica que hay cinco outliers. La variable test$all.stats$Obs.Num nos da los indices de los 10 valores que más se alejan de la media.

```
test$all.stats$Obs.Num
```

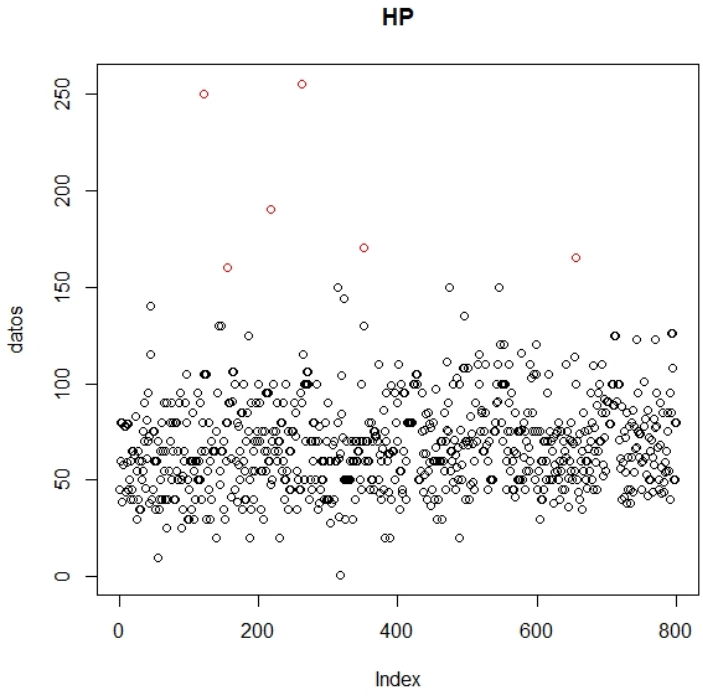
```
##  [1] 262 122 218 352 656 156 314 474 545 546
```

Construimos el vector con los índices de los outliers y se lo pasamos como parámetro a la función MiPlot_Univariate_Outliers().

```
indices_outliers = test$all.stats$Obs.Num[test$all.stats$Outlier]
indices_outliers
```

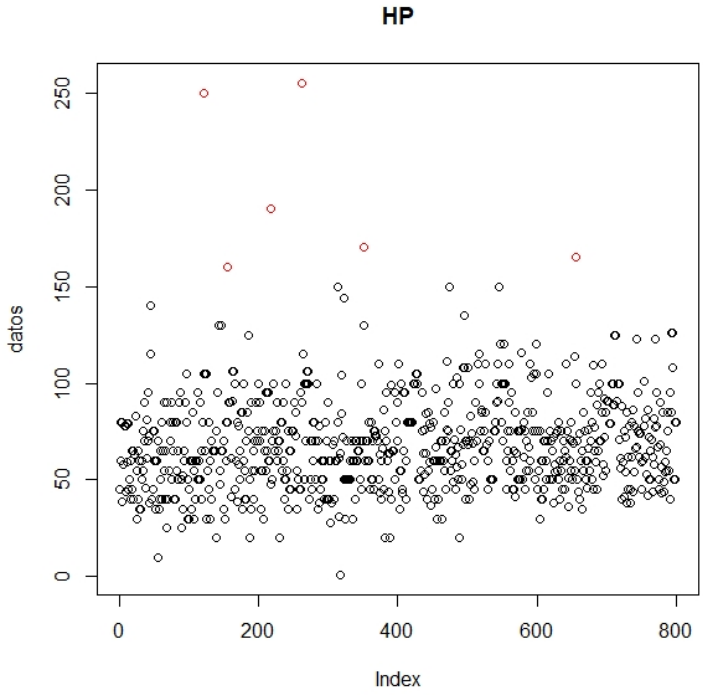
```
## [1] 262 122 218 352 656 156
```

```
windows()
MiPlot_Univariate_Outliers(pokemon[,2],indices_outliers,"HP")
```



la función MiPlot_resultados_TestRosner() hace directamente lo anterior y lanza el plot.

```
windows()
MiPlot_resultados_TestRosner(pokemon[,2],num_outliers = 10)
```



5. Outliers Multivariantes Mahalanobis.

Hasta ahora solo nos hemos centrado en buscar outliers univariantes, no hemos prestado atención en la relación que puede haber entre las variables. Es posible que un pokemon no presente valores en sus stats muy altos o bajos, sin embargo la combinación en la que se presentan sus valores en las variables sí que lo sea. Por ejemplo es muy raro que un pokemon que presente una defensa alta tenga pocos puntos de vida. En este estudio veremos anomalias de este tipo. Podemos clasificar los valores anómalos multivariantes en dos tipos:

- Valores anormalmente altos en alguna de sus variables.
- Combinación anómala de valores en dos o más características.

Estas últimas son las que acabamos de comentar y son las más interesantes de detectar.

Durante todo el estudio de **5. Outliers Multivariantes Mahalanobis** vamos a usar técnicas de estadística paramétrica, más concretamente supondremos que el conjunto de datos se distribuye como una normal multivariante. Se pueden aplicar los test si cada variable es unimodal pero los test tendrán menor potencia. La función de densidad de $\mathbf{X} = (X_1, \dots, X_p) \to \mathcal{N}(\mu, \Sigma)$ viene dada por:

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^p |\Sigma|}} \exp\left(-\frac{(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)}{2}\right), \quad \forall \mathbf{x} \in \mathbb{R}^p,$$

donde μ es el vector de medias y Σ la matriz de covarianzas.

Si $\mathbf{X} = (X_1, \dots, X_p) \to \mathcal{N}(\mu, \Sigma)$ implica que las distribución de cada componente $X_i \to \mathcal{N}(\mu_i, \sigma_i^2)$, el recíproco no es cierto aunque se puede probar la siguiente afirmación.

Proposición 1.
Sean $X_1, \dots, X_p$ variables aleatorias identicamente distribuidas e independientes, $X_i \to \mathcal{N}(\mu_i, \sigma_i^2)$. Consideremos:

$$\mu = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_p \end{pmatrix}, \Sigma = \begin{pmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_p^2 \end{pmatrix}.$$

Entonces $\boldsymbol{X} = (X_1, \dots, X_p) \to \mathcal{N}(\mu, \Sigma)$.

Demostración.
Como X_i y X_j son independientes para todo $i, j \in \{1, \dots, p\}$ con $i \neq j$, la función de densidad de $\mathbf{X} = (X_1, \dots, X_p)$ es el producto de las marginales:

$$f(\mathbf{x}) = f_1(x_1) \cdots f_p(x_p) =$$

$$= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(x_1-\mu_1)^2}{2\sigma_1^2}\right) \cdots \frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{(x_p-\mu_p)^2}{2\sigma_p^2}\right) =$$

$$= \frac{1}{\sqrt{2\pi\sigma_1^2 \cdots \sigma_p^2}} \exp\left(-\frac{(x_1-\mu_1)^2}{2\sigma_1^2} - \cdots - \frac{(x_p-\mu_p)^2}{2\sigma_p^2}\right),$$

teniendo en cuenta que:

$$|\Sigma| = \begin{vmatrix} \sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_p^2 \end{vmatrix} = \sigma_1^2 \cdots \sigma_p^2,$$

$$\Sigma^{-1} = \begin{pmatrix} 1/\sigma_1^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1/\sigma_p^2 \end{pmatrix},$$

$$(x_1-\mu_1,\dots,x_p-\mu_p)\begin{pmatrix}1/\sigma_1^2 & \dots & 0\\ \vdots & \ddots & \vdots\\ 0 & \dots & 1/\sigma_p^2\end{pmatrix}\begin{pmatrix}x_1-\mu_1\\ \vdots\\ x_p-\mu_p\end{pmatrix}=$$

$$=\left(\frac{x_1-\mu_1}{\sigma_1^2},\dots,\frac{x_p-\mu_p}{\sigma_p^2}\right)\begin{pmatrix}x_1-\mu_1\\ \vdots\\ x_p-\mu_p\end{pmatrix}=\frac{(x_1-\mu_1)^2}{\sigma_1^2}+\dots+\frac{(x_p-\mu_p)^2}{\sigma_p^2}.$$

Se tiene por tanto que:

$$f(\mathbf{x})=\frac{1}{\sqrt{(2\pi)^p|\Sigma|}}\exp\left(-\frac{(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)}{2}\right),\quad \forall\mathbf{x}\in\mathbb{R}^p,$$

que es la función de densidad de una normal multivariante.

□

Queremos medir como de alejado está un valor del vector de medias , teniendo en cuenta las variazas y covarianzas de las variables, para ello debemos definir una distancia conveniente. La distancia de Mahalanobis va a tener en cuenta todas estas premisas, viene definida como a continuación:

$$d_M(\mathbf{x},\mu)=\sqrt{(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)},\quad \forall\mathbf{x}\in\mathbb{R}^p.$$

A la hora de trabajar con la distancia de Mahalanobis no hace falta normalizar los datos, esta medida ya está diseñada para no tener problemas de escala.

5.1 Test de tipo a).

Podemos plantear dos tipo de test. Para los test que llamaremos de tipo a), queremos contrastar las siguientes hipótesis:

$$\begin{cases}H_0: \text{El dato con máxima distancia de Mahalanobis no es un outlier}\\ H_1: \text{El dato con máxima distancia de Mahalanobis es un outlier}\end{cases}$$

en la construcción del test usaremos un nivel de significación común $\alpha = 0{,}05$. Usaremos el paquete CerioliOutlierDetection. Determinaremos la covarianza mediante el método iterativo MCD por lo que es importante establecer una semilla antes de lanzar la función correspondiente a este computo.

```
set.seed(12)
```

Aplicamos el test de tipo a) mediante la función cerioli2010.fsrmcd.test() y calculamos los índices de los pokemon que presentan outliers multivaraintes.

```
library(CerioliOutlierDetection)
cericioli = cerioli2010.fsrmcd.test(pokemon[,2:7],signif.alpha = 0.05)
is_outlier_cericioli = cericioli$outliers
claves_outliers = which(is_outlier_cericioli == TRUE)
length(claves_outliers)
```

```
## [1] 249
```

Nos salen varios outliers porque el procedimiento realiza un test por separado a cada valor del dataset. Esto puede confundir ya que sólo podemos fijarnos en el valor más extremo, la hipótesis nula es *El dato con máxima distancia de Mahalanobis no es un outlier*. Tenemos que ver los valores de la distancia de Mahalanobis y calcular el máximo de ellos.

```
dist_mah_ponderadas = cericioli$mahdist.rw
claves_outliers_ordenado = order(dist_mah_ponderadas,decreasing = TRUE)
pokemon[claves_outliers_ordenado[1],]
```

```
##     Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial Velocidad
## 262     Blissey 255     10      10              75              135        55
```

Rechazamos la hipótesis de que Blissey no es un pokemon que presenta anomalías en sus stats. Si nos fijamos en los valores de sus columnas vemos que es un pokemon que presenta una combinación de anomalias, tiene un Hp muy alto, de hecho es el pokemon que tiene mayor Hp. También tiene un ataque muy bajo y más de lo mismo con la defensa. Es algo raro que un pokemon con tan poca defensa tenga tantísimos puntos de vida. Vemos que la defensa especial es relativamente alta, por lo que competitivamente Blissey es un pokemon de gran utilidad para utilizar como muro defensivo especial.

5.2 Test de tipo b).

Ahora usaremos un test de tipo b). Con este tipo de test podemos averiguar si hay más de un outlier, las hipótesis a contrastar son las siguientes:

$$\begin{cases} H_0 : \text{No hay outliers} \\ H_1 : \text{Hay al menos un outlier} \end{cases}$$

El problema de este tipo de test es el error acumulado FWER del que ya hemos hablado en **4.2 Test de Rosner**, debemos penalizar el nivel de significación esto conlleva que rechazar la hipótesis se haga con mayor confianza pero también más dificultad, si penalizamos mucho el α va a ser tan complicado rechazar H_0 que es posible no rechazarla aún habiendo presencia de valores anómalos.

Fijamos el nivel de significación penalizado para el posterior uso del test tipo b).

```
nivel_de_significacion_penalizado = 1-(1-0.01)^(1/6)  #Hay 6 variables númericas en el dataset
```

Aplicamos el test de tipo b).

```
test2 = cerioli2010.irmcd.test(pokemon[,2:7],signif.gamma = nivel_de_significacion_penalizado)
head(pokemon[test2$outliers,])
```

```
##             Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 20 BeedrillMega Beedrill  65    150      40              15               80
## 45            Jigglypuff 115     45      20              45               25
## 46            Wigglytuff 140     70      45              85               50
## 72 AlakazamMega Alakazam  55     50      65             175               95
## 79              Tentacool  40     40      35              50              100
## 81                Geodude  40     80     100              30               30
##    Velocidad
## 20       145
## 45        20
## 46        45
## 72       150
## 79        70
## 81        20
```

Con ello hemos visto que hay 128 pokemon que presentan outliers. Podemos observar por ejemplo en Tentacool que no presume de tener un valor muy alto en ningún stat pero sin embargo lo anormal es que prestan 100 en defensa especial lo cual es mucho en comparación con las demás características.

6. Outliers Multivarientes LOF.

En los próximos apartados haremos uso de técnicas no paramétricas para el calculo de outliers, por lo cual no vamos a necesitar la suposición de normalidad multivariante ni nungún otro tipo de distribución sobre el conjunto de datos. Primero veremos el método LOF (Local Outlier Factor).

La filosofía de LOF es la misma que en la de knn, vamos a calcular si un punto es o no es un outlier dependiendo de la distancia a la que se encuentra con sus vecinos más cercanos. Como vamos a trabajar con distancias es importante normalizar los datos para que las escalas de las variables no afecten a la distancia. También es importante seleccionar una buena medida de distancia, como estamos trabajando solo con valores númericos la distancia Euclidea es una buena elección.

6.1 Ouliers Basados en su Distancia.

Previamente establecemos el número de vecinos más cercanos con los que queremos comparar cada punto para determinar si es un outlier.

```
numero_de_vecinos_lof = 8
```

Llamamos a la función lofactor(), esta nos devuelve la distancia media a los vecinos más cercados de cada pokemon. Ordenamos los valores para dictaminar los ourliers.

```
library(DMwR)
```

```
## Loading required package: lattice
```

```
## Loading required package: grid
```

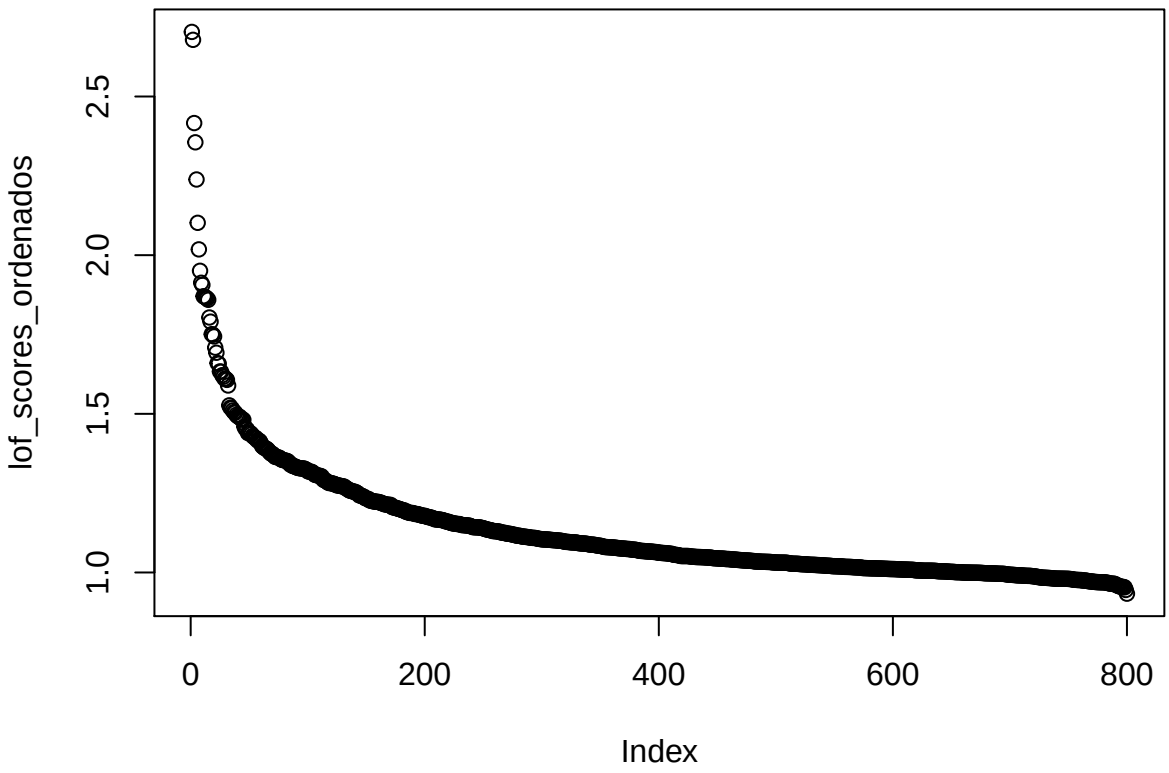
```
## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo
```

```
## Registered S3 method overwritten by 'quantmod':
##   method           from
##   as.zoo.data.frame zoo
```

```
lof_scores = lofactor(pokemon_normalizado[,2:7],numero_de_vecinos_lof)
lof_scores_ordenados = lof_scores[order(lof_scores,decreasing = TRUE)]
```

Hacemos un plot de las distancias para ver cuales destacan sobre las demás y poder establecer un número de outliers.

```
plot(lof_scores_ordenados)
```



Vemos que hay 7 valores que destacan sobre los demás. Calculemos los pokemon que presentan los outliers.

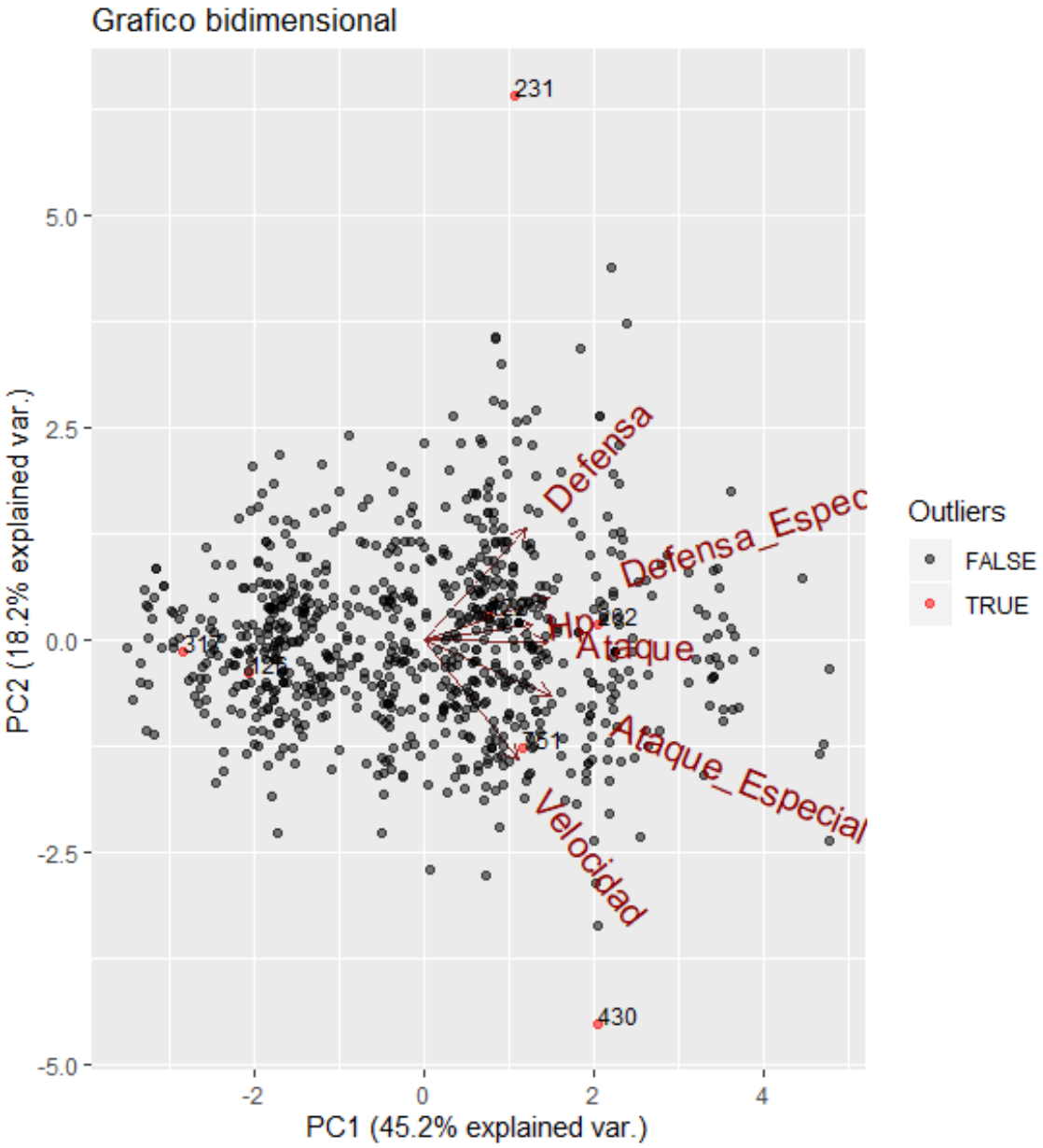
```
numero_de_ouliers = 7
indices_de_lof_top_ordenados = order(lof_scores,decreasing = TRUE)[1:7]
pokemon_normalizado[indices_de_lof_top_ordenados,]
```

```
##      pokemon.Nom_Pokemon         Hp     Ataque    Defensa Ataque_Especial
## 430   DeoxysAttack Forme -0.7542197  3.1117359 -1.7266342      3.27544271
## 231              Shuckle -1.9290929 -2.1259042  5.0076963     -1.91979204
## 262              Blissey  7.2740810 -2.1259042 -2.0473167      0.06662125
## 317             Shedinja -2.6731794  0.3388676 -0.9249282     -1.30858795
## 126               Horsea -1.5374685 -1.2016148 -0.1232222     -0.08617978
## 122              Chansey  7.0782688 -2.2799524 -2.2076579     -1.15578693
## 751 AegislashBlade Forme -0.3625953  2.1874465 -0.7645870      2.35863658
##     Defensa_Especial  Velocidad
## 430       -1.8650565  2.8121531
## 231        5.6810514 -2.1774421
## 262        2.2673359 -0.4568921
## 317       -1.5057180 -0.9730571
## 126       -1.6853873 -0.2848371
## 122        1.1893205 -0.6289471
## 751       -0.7870411 -0.2848371
```

Veamos en un biplot los valores anómalos, con ello podemos discutir si se tratan de pokemon que son raros porque presentan alguna variable muy destacada o porque tienen una combinación de sus stats anormal. Podemos presentar estos datos mediante un biplot, este nos muestra los valores extremos en alguna de las variables y además, también nos muestra las correlaciones entre variables el precio a pagar es que es una representación aproximada, los puntos mostrados son resultados de proyecciones de n dimensiones a 2, por lo que sólo es una representación aproximada (mejor cuanto mayor sea la suma de los porcentajes que aparecen como componentes principales PC1 y PC2).

```
is_lof_outlier = row(pokemon_normalizado)[,1] %in% indices_de_lof_top_ordenados
```

```
windows()
MiBiPlot_Multivariate_Outliers(pokemon_normalizado[,2:7],is_lof_outlier,"Grafico bidimensional")
```



Los pokemon como Shuckle (231) o DeoxysAttack Forme (430) estan muy alejados de la nube de puntos por lo que presntan valores muy altos en alguna de sus variables. Los que se encuentran entre la nube de puntos como Horsea (126) se tratan de valores anómalos porque tienen una combianacion extraña en los valores de sus stats.

6.2 Análisis de los Outliers.

En este apartado queremos ver cuáles son los outliers multivariantes "puros" es decir, aquellos outliers que lo son porque tienen una combinación anormal de valores en varias variables y no porque tengan un valor anormal en alguna variable.

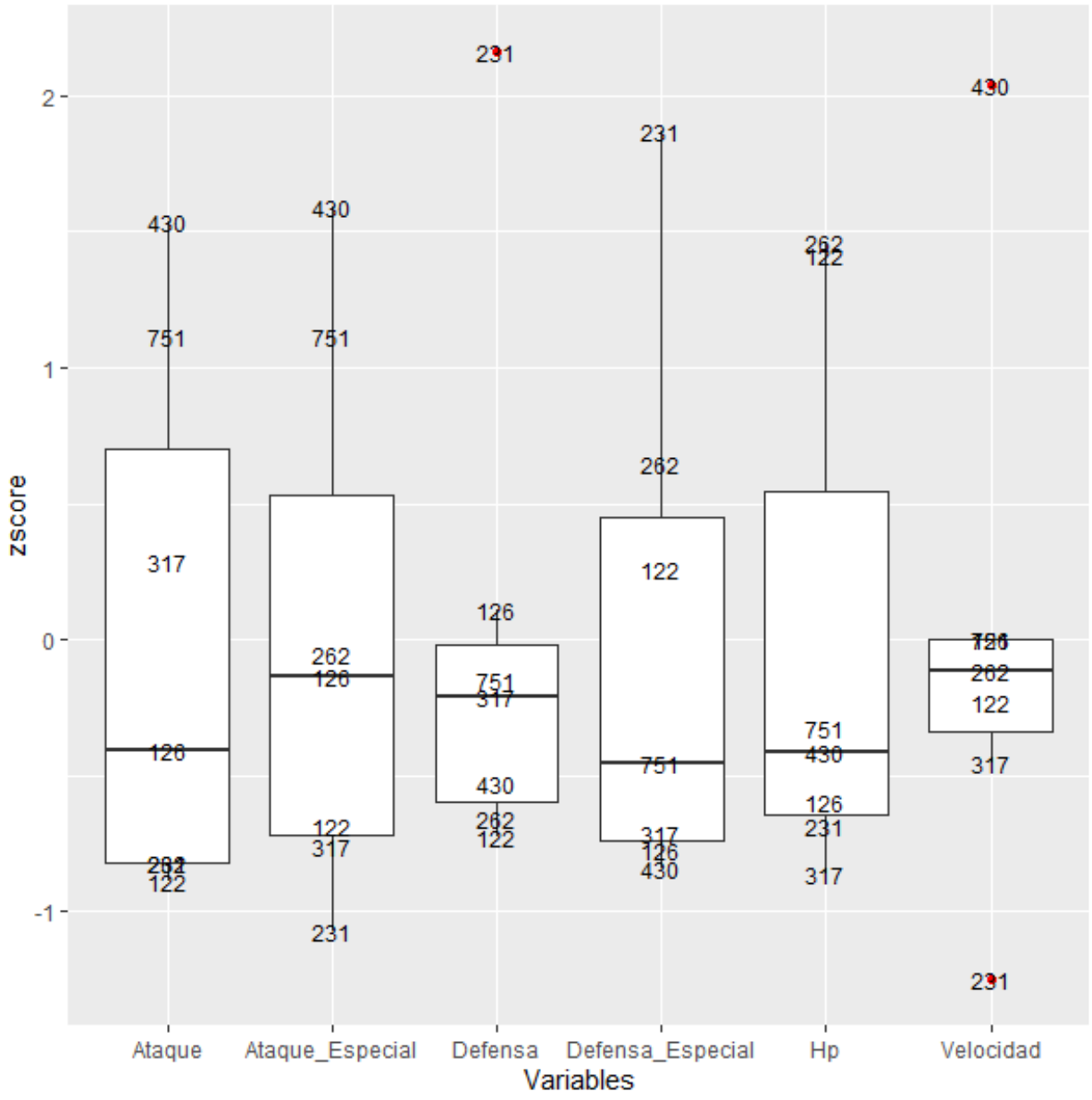
6.2.1 Ouliers con valor anormal en alguna variable.

En primera instancia vamos a verlos gráficamente. Mostraremos en diagrama de cajas cada una de los stats de los pokemon que hemos considerado como anómalos.

```
pokemon[is_lof_outlier,]
```

```
##              Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 122              Chansey 250      5       5              35              105
## 126               Horsea  30     40      70              70               25
## 231              Shuckle  20     10     230              10              230
## 262              Blissey 255     10      10              75              135
## 317             Shedinja   1     90      45              30               30
## 430   DeoxysAttack Forme  50    180      20             180               20
## 751 AegislashBlade Forme  60    150      50             150               50
##     Velocidad
## 122        50
## 126        60
## 231         5
## 262        55
## 317        40
## 430       150
## 751        60
```

```
MiBoxPlot_juntos_con_etiquetas(pokemon_normalizado[is_lof_outlier,2:7],is_lof_outlier)
```



Con ello podemos observar que tanto Shuckle, DeoxysAttack Forme son valores anómalos de forma univarainte, es lo mismo que vimos en el biplot comentado anteriormente.

También podemos usar el método IQR para detectar los otliers univariantes entre los que obteniamos por el método de LOF.

```
vector_clave_outlier_IQR_en_alguna_columna =
  vector_claves_outliers_IQR_en_alguna_columna(pokemon[is_lof_outlier,2:7])
pokemon[is_lof_outlier,][vector_clave_outlier_IQR_en_alguna_columna,]
```

```
##             Nom_Pokemon Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 231             Shuckle 20     10     230             10              230
## 231.1           Shuckle 20     10     230             10              230
## 430   DeoxysAttack Forme 50    180      20            180               20
##       Velocidad
## 231           5
## 231.1         5
```

```
## 430         150
```

Obtenemos los mismos resultados que habiamos comentado en los gráficos, además Shuckle presenta valores anómalos univariantes en dos variables, defensa y velocidad.

6.2.2 Outliers "Puros".

Construimos una variable que contenga los valores anómalos de LOF pero despreciendo los univariantes que hemos calculado anteriormente.

```
booleano_outlier_LOF_variente = 1:7 %in% vector_clave_outlier_IQR_en_alguna_columna
indices_de_outliers_multivariantes_LOF_pero_no_1variantes =
  which(is_lof_outlier == TRUE)[!booleano_outlier_LOF_variente]
pokemon[indices_de_outliers_multivariantes_LOF_pero_no_1variantes,]
```

```
##              Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 122              Chansey 250      5       5              35              105
## 126               Horsea  30     40      70              70               25
## 262              Blissey 255     10      10              75              135
## 317             Shedinja   1     90      45              30               30
## 751 AegislashBlade Forme  60    150      50             150               50
##     Velocidad
## 122        50
## 126        60
## 262        55
## 317        40
## 751        60
```

Los 5 pokemon obtenidos todos presentan una combinación anómala en sus stats.

- Chansey presenta un Hp alto sin embargo la defensa es muy baja, la combinación de estos dos valores para esas características es anómala ya que el Hp tiene una correlación positiva frente la defensa. Blissey es la evolución de Chansey y le ocurre lo mismo.
- En Horsey los valores de ataque especial y defensa son muy altos frente al demás stats, cosa extraña y más para una preevolución como es este caso, normalmente los stats para los pokemon débiles suelen ser bastante parecidos.
- Shedinja es un caso muy especial ya que solo tiene 1 punto de vida, valor anómalo y más si lo juntamos con su alto ataque.
- Finalmente tanto el ataque como ataque especial de AegislashBlade Forme son muy altos frente al resto de sus variables.

7. Ouliers Multivariantes Clustering.

En esta parte seguiremos usando técnicas no paramétricas para la estimación de los outliers como hicimos en **6. Outliers Multivariantes LOF**. La idea principal que siguen las posteriores técnicas es la siguiente, en primera instancia vamos a hacer un análisis clustering sobre los datos. Compararemos cada elemento del conjunto con los demás elementos del cluster al que pertenece, con ello podremos dictaminar si se trata de un valor anómalo o no.

Una forma rápida para detectar si un punto es un outlier es viendo si este no pertenece a ningún cluster. Usando técnicas de clustering basadas en algoritmos de densidad, por ejemplo DBSCAM, los puntos ruido no pertenecen a ningún cluster y podrían considerarse outliers.

También podemos medir las distancias de los puntos respecto los centroides que forman los clusters. La pregunta es, ¿cómo realizamos esas medidas? Lo haremos de tres formas distintas.

- La primera forma a la que denominaremos **Alternativa A** consiste en calcular la distancia euclidea de cada punto a su centroide más cercano.
- Otra manera, **Alternativa B**, será midiendo la distancia de Mahalanobis de los puntos a la distribución de cada cluster. Para ello se deberá calcular la matriz de covarianzas de cada cluster lo cual supone una complejidad computacional muy alto.
- En la **Alternativa c** vamos a medir la distancia relativa a su centroide más cercano. Esta distancia relativa es, el cociente entre la distancia de un punto a su centroide y la media de la distancia de todos los puntos en el cluster a su centroide.

7.1 Alternativa A.

Vamos a sleccionar a priori el número de clusters que usaremos para emplear las técnicas para el computo de outliers. Los clusters son agrupamientos que se realizan mediante calculo de distancias de los stats entre los pokemon, por ello nos los van a agrupar dependiendo de la suma de sus stats. En cada clusters tendremos pokemon muy similares en cuanto la suma de sus características, podemos decir que los que mayor suma de stats tienen son más buenos para jugarlos. Por tanto hay que considerar un número de clusters acorde a eso.

```
numero_de_clusters   = 3
```

Se pueden calcular las distancias respecto los centroides para ello haremos un kmeans o respecto de los mediodes. Haremos ambos estudios.

7.1.1 Distancias respecto Centroides.

Construimos el modelo kmeans con los datos normalizados, así la distancias no se vean afectadas por las unidades de las características.

```
library(cluster)
set.seed(11)
modelo_kmeans = kmeans(pokemon_normalizado[,2:7],numero_de_clusters)
```

Podemos conocer el cluster al que pertenece cada elemento del conjunto.

```
indices_clustering = modelo_kmeans$cluster
```

También conocemos el valor de cada variable para los centroides.

```
centroides_normalizados = modelo_kmeans$centers
centroides_normalizados
```

```
##           Hp     Ataque    Defensa Ataque_Especial Defensa_Especial  Velocidad
## 1  0.4101780  0.6709271  0.1498197     0.890992193        0.5058380  1.0914614
## 2  0.6032896  0.3457782  0.8681959     0.005358032        0.5720643 -0.4538369
## 3 -0.6669891 -0.6910549 -0.6478138    -0.634570617       -0.7152782 -0.4900992
```

Ahora definiremos una función que nos ayude a calcular la distancia de cada dato a su centroide.

```
distancias_a_centroides = function (datos_normalizados,
                                    indices_asignacion_clustering,
                                    datos_centroides_normalizados){

  sqrt(rowSums((datos_normalizados-
                  datos_centroides_normalizados[indices_asignacion_clustering,])^2))
}
```

Con ello calculamos los puntos con mayor distancia a su centroide y su indice, esos seran los puntos que consideraremos anómalos.

```
top_clustering_outliers = 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)

  indices = order(dist_centroides, decreasing=T)[1:numero_de_outliers]

  list(distancias = dist_centroides[indices]  , indices = indices)
}
```

```
top_outliers_kmeans = top_clustering_outliers(pokemon_normalizado[,2:7],
                                              indices_clustering,
                                              centroides_normalizados,
                                              7)
```

Las distancias de los outliers a su centroide vendrían dadas por

```
top_outliers_kmeans$distancias
```

```
## [1] 7.901671 7.873156 7.748588 5.023946 5.010187 4.481571 4.420513
```

y los índices de los outliers en el conjunto de datos son

```
top_outliers_kmeans$indices
```

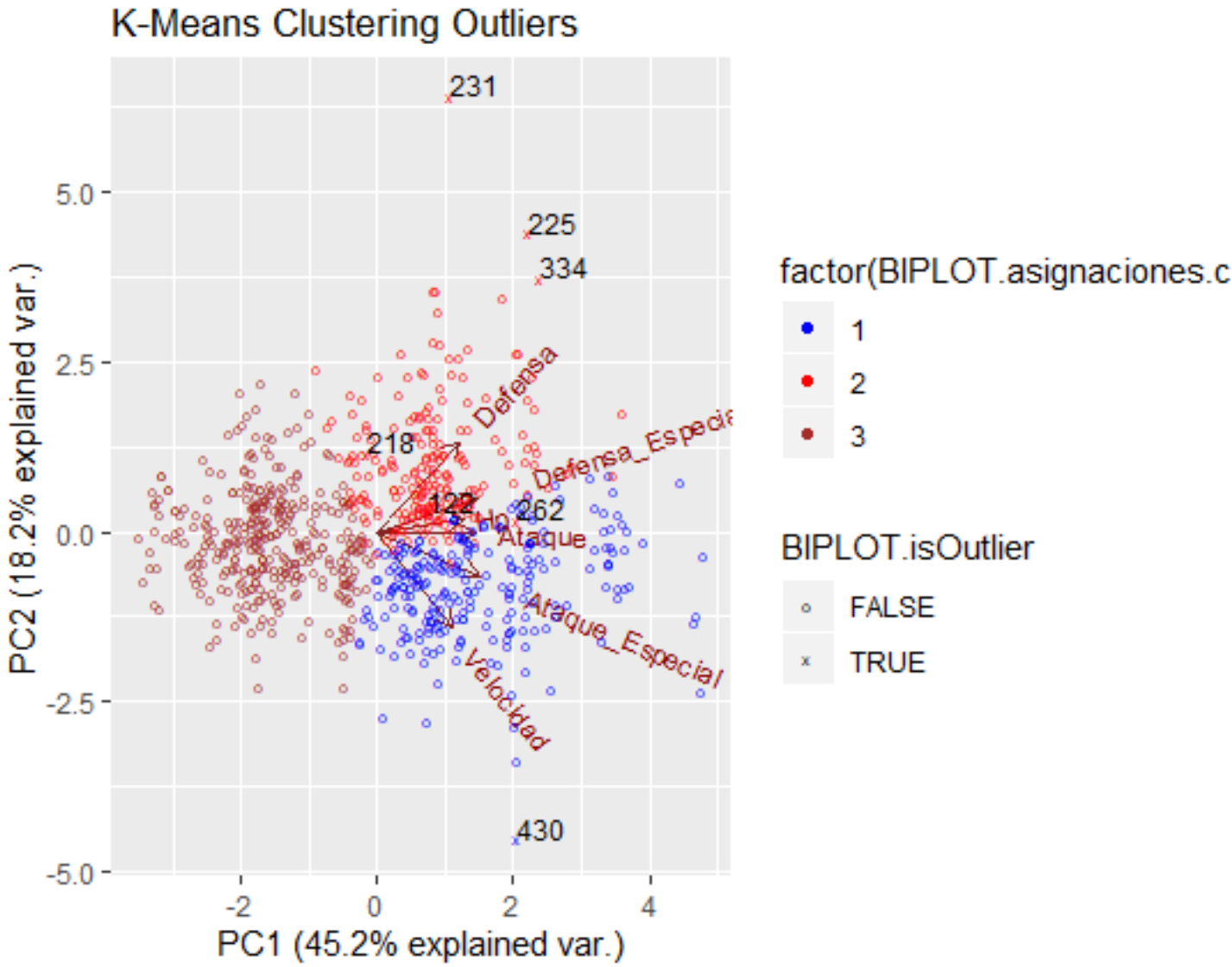
```
## [1] 231 262 122 218 430 334 225
```

Finalmente crearemos un vector booleano que nos informara donde se localizan los valores anómalos, con ello podremos llamar a la función MiBiPlot_Clustering_Outliers() y los visualizaremos en una gráfica.

```
is_kmeans_outliers =
  row(pokemon_normalizado)[,1] %in% top_outliers_kmeans$indices
```

```
BIPLOT.isOutlier             = is_kmeans_outliers
BIPLOT.cluster.colors        = c("blue","red","brown")     # Tantos colores como diga numero.de.cluster
BIPLOT.asignaciones.clusters = indices_clustering
```

```
MiBiPlot_Clustering_Outliers(pokemon[,2:7], "K-Means Clustering Outliers")
```



Hemos contruido el modelo con los datos normalizados, vamos a invertir el cambio de variable que hemos

usado para reescalar los datos:

$$Z = \frac{X - \bar{X}}{\sigma} \iff X = Z\sigma + \bar{X}.$$

Primero vamos a construir un vector con las medias de cada columna.

```
pokemon_medias = colMeans(pokemon[,2:7])
pokemon_medias
```

```
##              Hp          Ataque         Defensa  Ataque_Especial
##         69.25875         79.00125         73.84250         72.82000
## Defensa_Especial       Velocidad
##         71.90250         68.27750
```

Ahora hacemos lo mismo pero para las desviaciones típicas

```
pokemon_sd = apply(pokemon[,2:7],2,sd)
pokemon_sd
```

```
##              Hp          Ataque         Defensa  Ataque_Especial
##         25.53467         32.45737         31.18350         32.72229
## Defensa_Especial       Velocidad
##         27.82892         29.06047
```

Para recuperar los datos de los centroides multiplicamos por el vector de desviaciones típicas y le sumamos el vector de medias.

```
centroides_valores = sweep(centroides_normalizados,2,pokemon_sd,"*")
centroides_valores = sweep(centroides_valores,2,pokemon_medias,"+")
centroides_valores
```

```
##         Hp    Ataque  Defensa Ataque_Especial Defensa_Especial Velocidad
## 1 79.73251 100.77778  78.5144       101.97531         85.97942  99.99588
## 2 84.66355  90.22430 100.9159        72.99533         87.82243  55.08879
## 3 52.22741  56.57143  53.6414        52.05539         51.99708  54.03499
```

```
pokemon[is_kmeans_outliers,]
```

```
##             Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 122             Chansey 250      5       5              35              105
## 218           Wobbuffet 190     33      58              33               58
## 225 SteelixMega Steelix  75    125     230              55               95
## 231             Shuckle  20     10     230              10              230
## 262             Blissey 255     10      10              75              135
## 334   AggronMega Aggron  70    140     230              60               80
## 430  DeoxysAttack Forme  50    180      20             180               20
##     Velocidad
## 122        50
## 218        33
## 225        30
## 231         5
## 262        55
## 334        50
## 430       150
```

En este caso puede ser muy interesante analizar los clusters para buscar cual agrupa a los pokemon más fuertes y buscar anomalias en ese cluster.

```
pokemon_cluster1 = pokemon[which(modelo_kmeans$cluster == 1),]
pokemon_cluster1 =
  pokemon_cluster1 %>% mutate(Sum_Stats = apply(pokemon_cluster1[,2:7],1,sum)) %>%
  arrange(desc(Sum_Stats))
head(pokemon_cluster1)
```

```
##             Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 1   MewtwoMega Mewtwo X 106    190     100             154              100
## 2   MewtwoMega Mewtwo Y 106    150      70             194              120
## 3 RayquazaMega Rayquaza 105    180     100             180              100
## 4   KyogrePrimal Kyogre 100    150      90             180              160
## 5 GroudonPrimal Groudon 100    180     160             150               90
## 6               Arceus 120    120     120             120              120
```

```
##   Velocidad Sum_Stats
## 1       130       780
## 2       140       780
## 3       115       780
## 4        90       770
## 5        90       770
## 6       120       720
```

```
pokemon_cluster2 = pokemon[which(modelo_kmeans$cluster == 2),]
pokemon_cluster2 =
  pokemon_cluster2 %>% mutate(Sum_Stats = apply(pokemon_cluster2[,2:7],1,sum))%>%
  arrange(desc(Sum_Stats))
head(pokemon_cluster2)
```

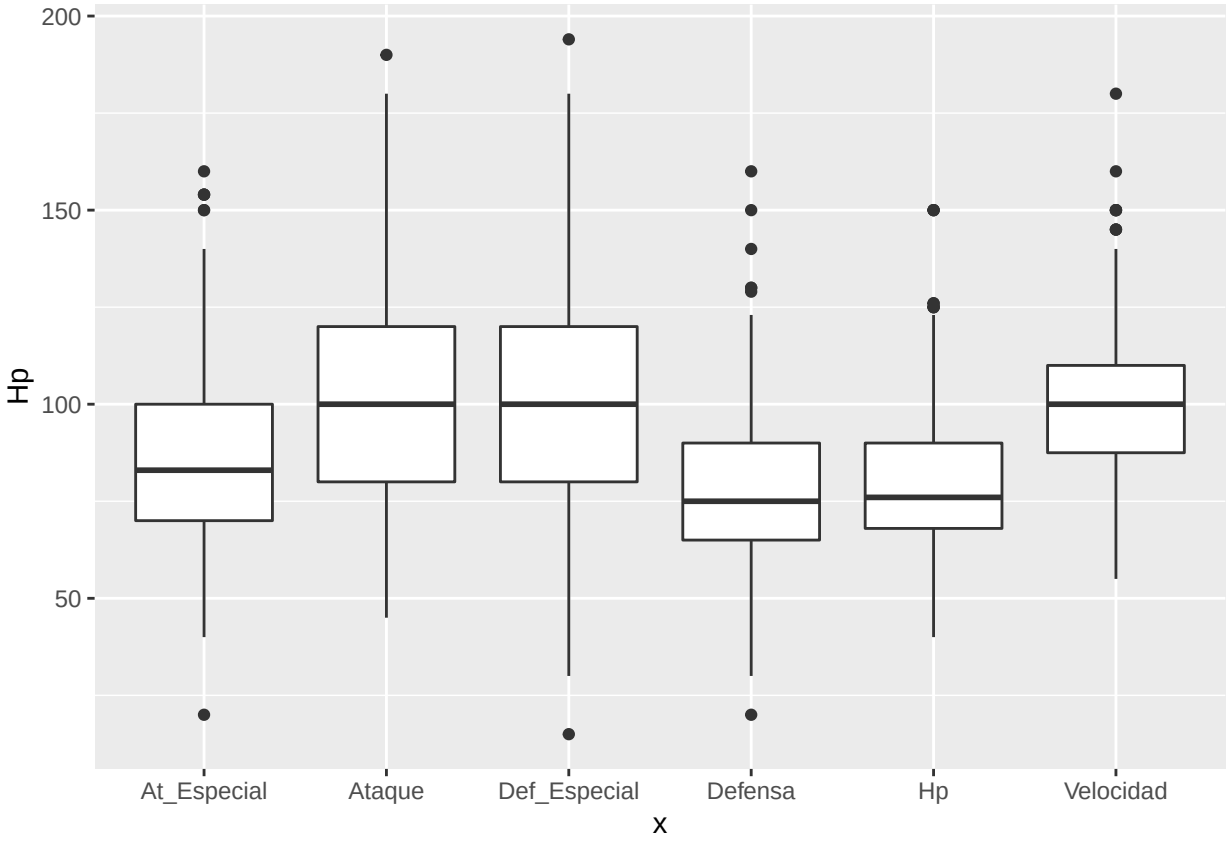
```
##              Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 1 TyranitarMega Tyranitar 100    164     150              95              120
## 2   GiratinaAltered Forme 150    100     120             100              120
## 3   GyaradosMega Gyarados  95    155     109              70              130
## 4   SwampertMega Swampert 100    150     110              95              110
## 5       AggronMega Aggron  70    140     230              60               80
## 6   AmpharosMega Ampharos  90     95     105             165              110
##   Velocidad Sum_Stats
## 1        71       700
## 2        90       680
## 3        81       640
## 4        70       635
## 5        50       630
## 6        45       610
```

```
pokemon_cluster3 = pokemon[which(modelo_kmeans$cluster == 3),]
pokemon_cluster3 =
  pokemon_cluster3 %>% mutate(Sum_Stats = apply(pokemon_cluster3[,2:7],1,sum))%>%
  arrange(desc(Sum_Stats))
head(pokemon_cluster3)
```

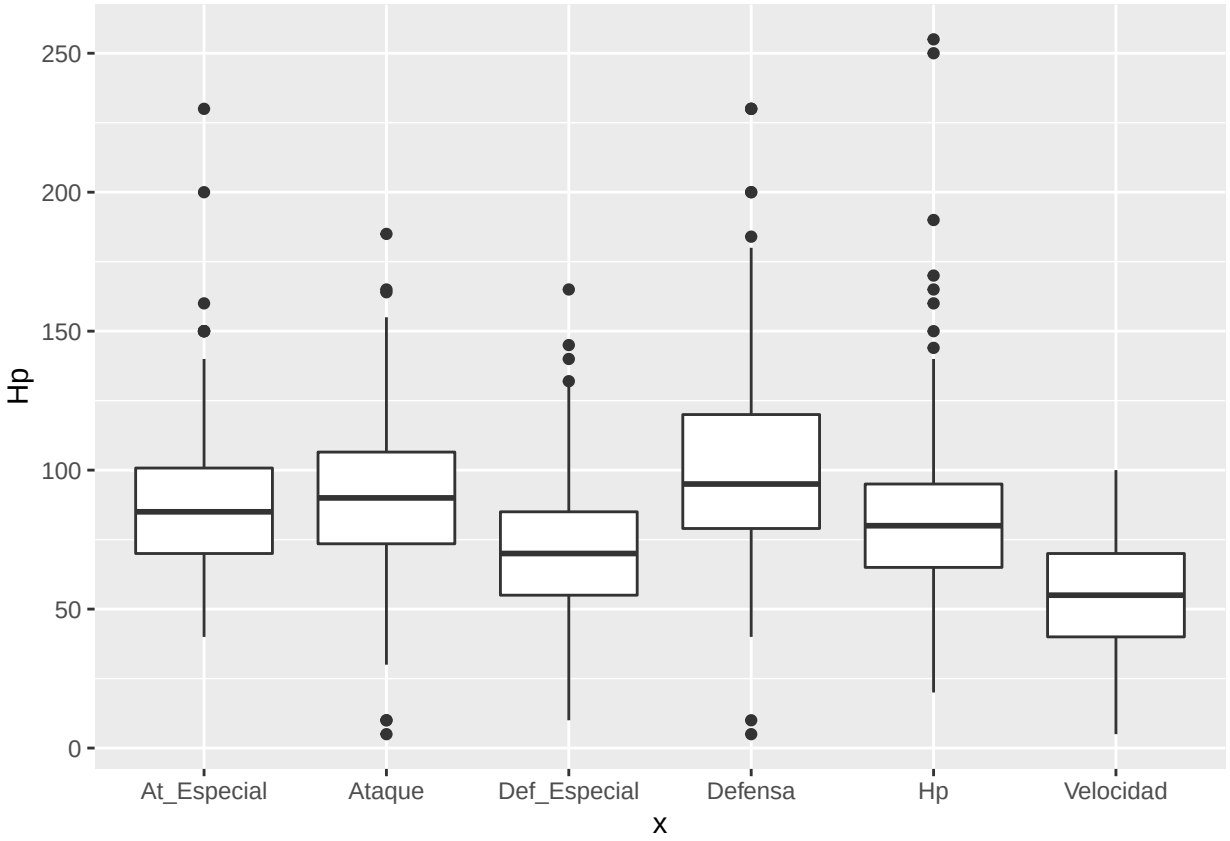
```
##   Nom_Pokemon Hp Ataque Defensa Ataque_Especial Defensa_Especial Velocidad
## 1      Seadra 55     65      95              95               45        85
## 2    Qwilfish 65     95      75              55               55        85
## 3    Chimecho 65     50      70              95               80        65
## 4      Mothim 70     94      50              94               50        66
## 5   Dragonair 61     84      65              70               70        70
## 6   Mightyena 70     90      70              60               60        70
##   Sum_Stats
## 1       440
## 2       430
## 3       425
## 4       424
## 5       420
## 6       420
```

Podemo observar que los clusters 1 y 2 recogen a los pokemon cuya suma de stats es mayor, centremonos en ellos y veamos cmo se comportan sus stats mediante uno diagrama de cajas.

```
library(ggplot2)
ggplot() +
  geom_boxplot(data = pokemon_cluster1,aes(y=Hp,x="Hp")) +
  geom_boxplot(data = pokemon_cluster1,aes(y=Ataque,x="Ataque"))  +
  geom_boxplot(data = pokemon_cluster1,aes(y=Defensa,x="Defensa")) +
  geom_boxplot(data = pokemon_cluster1,aes(y=Ataque_Especial,x="Def_Especial")) +
  geom_boxplot(data = pokemon_cluster1,aes(y=Defensa_Especial,x="At_Especial")) +
  geom_boxplot(data = pokemon_cluster1,aes(y=Velocidad,x="Velocidad"))
```




```
ggplot() +
  geom_boxplot(data = pokemon_cluster2,aes(y=Hp,x="Hp")) +
  geom_boxplot(data = pokemon_cluster2,aes(y=Ataque,x="Ataque"))  +
  geom_boxplot(data = pokemon_cluster2,aes(y=Defensa,x="Defensa")) +
  geom_boxplot(data = pokemon_cluster2,aes(y=Ataque_Especial,x="Def_Especial")) +
  geom_boxplot(data = pokemon_cluster2,aes(y=Defensa_Especial,x="At_Especial")) +
  geom_boxplot(data = pokemon_cluster2,aes(y=Velocidad,x="Velocidad"))
```

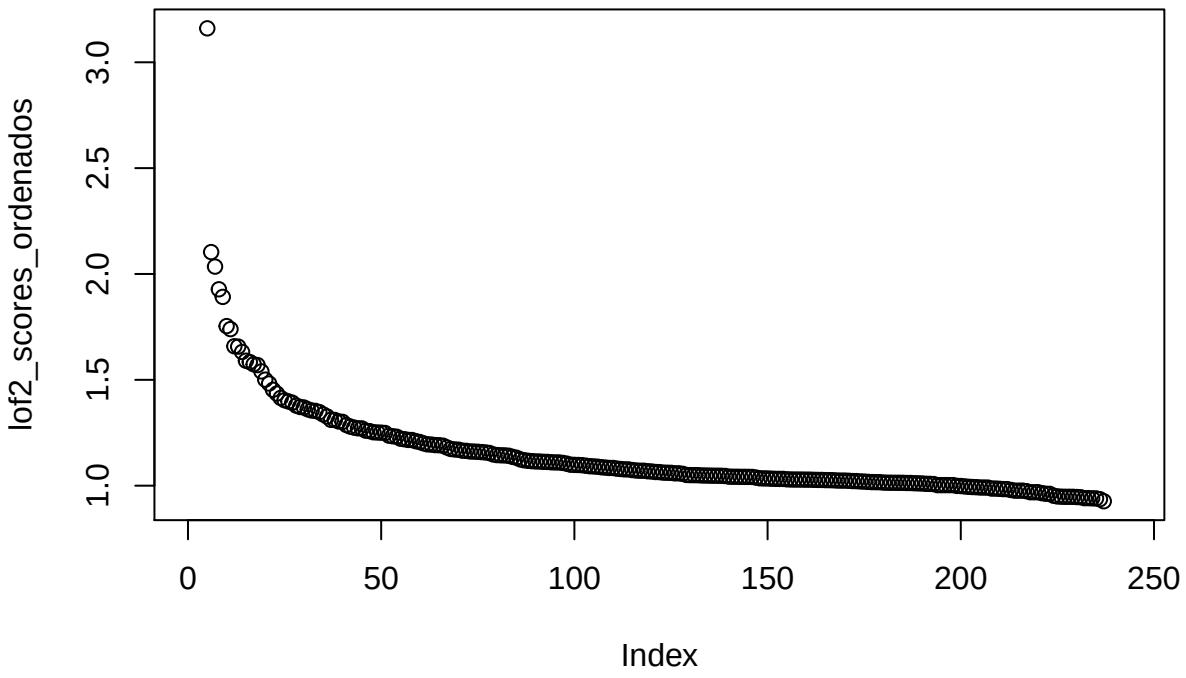


Fijandonos en los boxplot vemos que el cluster 1 la media de los stats de los pokemon son más altas y además el rango intercuartílico más amplio por lo tanto la dispersión de los datos es mayor.

En el biplot anterior se puede observar como los pokemon del cluster1, los puntos azules, tienden a ser más especializados en ataque y velocidad por lo que nos serán muy útiles a la hora de construir un equipo pokemon ofensivo.

Ahora buscamos mediante LOF outliers multivariantes en el agrupamiento.

```
numero_de_vecinos = 5
pokemon_cluster1_normalizados =
  data.frame(pokemon_cluster1$Nom_Pokemon,scale(pokemon_cluster1[,2:7]))
lof2_scores = lofactor(pokemon_cluster1_normalizados[,2:7],
                       numero_de_vecinos)
lof2_scores_ordenados = lof2_scores[order(lof2_scores,decreasing = TRUE)]
plot(lof2_scores_ordenados)
```



Consideraremos como 5 el número de otuliers. Veamos de quienes son los que presentan los valores anómalos.

```
numero_ouliers = 5
indices_de_lof2_top_ordenados = order(lof2_scores,decreasing =
                                        TRUE)[1:numero_ouliers]
pokemon_cluster1[indices_de_lof2_top_ordenados,]
```

```
##                 Nom_Pokemon  Hp Ataque Defensa Ataque_Especial
## 6                    Arceus 120    120     120             120
## 55                 Garchomp 108    130      95              80
## 72 KangaskhanMega Kangaskhan 105    125     100              60
## 83                   Cobalion  91     90     129              90
## 53        DeoxysAttack Forme  50    180      20             180
##    Defensa_Especial Velocidad Sum_Stats
## 6               120       120       720
## 55               85       102       600
## 72              100       100       590
## 83               72       108       580
## 53               20       150       600
```

Anteriormente hablamos sobre que tipos de pokemon había en el cluster 1, ¿son realmente anómalos los datos encontrados?

- Arceus, es un pokemon que tiene una puntuación muy alta en todos los stats, 120, por ello es un individuo extraño en el cluster. Presenta altas defensas cosa que lo hace anormal, pertenece al cluster porque también tiene características ofensivas elevadas.
- Garchomp sería un outlier puro, la combinación de sus variables se distancian mucho de las de sus vecinos más cercanos pero sin embargo no presenta outliers univariantes. Algo similar ocurre con KangaskhanMega Kangaskhan.
- Cobalion es un pokemon que presenta mucha defensa para pertenecer a este cluster.

- DeoxysAttack Forme es el ataque ofensivo por excelencia, gran velocidad y un ataque y ataque especial muy elevados. Es un outlier porque esas características son valores anómalos univariantes.

7.1.2 Distancias respecto Mediodes.

Ahora aplicaremos un análisis de agrupamientos mediante mediodes. Primero calculamos la matriz de distancia de los datos. Mediante pam() realizamos el clustering, el parámetro k es el número de agrupamientos.

```
library(cluster)
matriz_de_distancias = dist(pokemon_normalizado[,2:7])
set.seed(2)
modelo_pam = pam(matriz_de_distancias, k = numero_de_clusters)
```

Procedemos de forma análoga a como hicimos para kmedias.

```
indices_pam = modelo_pam$clustering
medoides_indices = modelo_pam$medoids
```

```
medoides_valores_normalizados = pokemon_normalizado[medoides_indices, ]
medoides_valores_normalizados
```

```
##     pokemon.Nom_Pokemon          Hp     Ataque    Defensa Ataque_Especial
## 396            Snorunt -0.75421968 -0.8935183 -0.7645870      -0.6973839
## 286          Mightyena  0.02902916  0.3388676 -0.1232222      -0.3917818
## 250            Kingdra  0.22484137  0.4929158  0.6784838       0.6778253
##     Defensa_Especial   Velocidad
## 396       -0.7870411 -0.62894708
## 286       -0.4277026  0.05927295
## 250        0.8299820  0.57543797
```

```
medoides_valores = pokemon[medoides_indices, ]
medoides_valores
```

```
##     Nom_Pokemon Hp Ataque Defensa Ataque_Especial Defensa_Especial Velocidad
## 396     Snorunt 50     50      50              50               50        50
## 286   Mightyena 70     90      70              60               60        70
## 250     Kingdra 75     95      95              95               95        85
```

```
top_outliers_pam = top_clustering_outliers(pokemon_normalizado[,2:7],
                                           indices_pam,
                                           medoides_valores_normalizados[,2:7],
                                           7)
top_outliers_pam
```

```
## $distancias
## [1] 8.252186 8.214940 8.035358 5.710964 5.255895 4.967155 4.856723
##
## $indices
## [1] 231 262 122 430 218 225 334
```

Y los pokemon que prestan valores anómalos serían los siguientes.

```
pokemon[top_outliers_pam$indices,]
```

```
##             Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 231             Shuckle  20     10     230              10              230
## 262             Blissey 255     10      10              75              135
## 122             Chansey 250      5       5              35              105
## 430  DeoxysAttack Forme  50    180      20             180               20
## 218           Wobbuffet 190     33      58              33               58
## 225 SteelixMega Steelix  75    125     230              55               95
## 334   AggronMega Aggron  70    140     230              60               80
##     Velocidad
## 231         5
## 262        55
## 122        50
## 430       150
## 218        33
## 225        30
## 334        50
```

Hemos obtenido unos valores muy parecidos a los que conseguimos aplicando kmedias.

7.2 Alternativa B.

Vamos a calcular la distancia de cada punto a su centroide pero en lugar de usar la distancia Euclidea, tomaremos la distancia de Mahalanobis. Para ello calcularemos los puntos que distan más de su centroide, tomando como medida la distancia de Mahalanobis.

Primero debemos crear tantos dataframes como número de clusters, cada uno de ellos contendran los valores del conjunto de datos que pertenezcan a dicho agrupamiento. Calcularemos la matriz de covarianzas de cada dataframe, necesaria para el computo de la distancia.

```
col1 = modelo_kmeans$cluster==1
col2 = modelo_kmeans$cluster==2
col3 = modelo_kmeans$cluster==3
seleccion = data.frame(Cluster1 = col1,Cluster2 = col2, Cluster3 = col3)
```

Con ello los pokemon que pertenecen al cluster 1 vienen dados por

```
head(pokemon[seleccion[,1],])
```

```
##                 Nom_Pokemon Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 3                 Venusaur 80     82      83             100             100
## 4     VenusaurMega Venusaur 80    100     123             122             120
## 7                 Charizard 78     84      78             109              85
## 8  CharizardMega Charizard X 78    130     111             130              85
## 9  CharizardMega Charizard Y 78    104      78             159             115
## 13   BlastoiseMega Blastoise 79    103     120             135             115
##    Velocidad
## 3         80
## 4         80
## 7        100
## 8        100
## 9        100
## 13        78
```

Ahora calculamos las matrices de covarianzas y los vectores de medias para los pokemon de cada agrupamiento.

```
x = list(pokemon[seleccion[,1],2:7],
            pokemon[seleccion[,2],2:7],
            pokemon[seleccion[,3],2:7])
lista_matriz_covarianzas = lapply(x,cov)
lista_vector_medias = list(apply(pokemon[seleccion[,1],2:7], 2, mean),
                           apply(pokemon[seleccion[,2],2:7], 2, mean),
                           apply(pokemon[seleccion[,3],2:7], 2, mean))
```

Otra forma de calcular la matriz de covarianzas es mediante la función cov.rob del paquete MASS. Esta función realiza una estimación robusta de la matriz de covarianzas y de la media. El método se basa en que $d^2_{S,\bar{x}}(x_i) = (x_i - \bar{x})^T S (x_i - \bar{x})$ es un estimador puntual de la distribución teórica de la distancia de Mahalanobis.

```
library(MASS)
```

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

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

## The following object is masked from 'package:dplyr':
##
##     select
```

```
lista_matriz_de_covarianzas = lapply(1:3, function(x) cov.rob(pokemon[seleccion[,x],2:7])$cov)
lista_vector_de_medias = lapply(1:3, function(x) cov.rob(pokemon[seleccion[,x],2:7])$center)
```

Con ello ya podemos calcular las distancias de mahalanobis a los centroides de cada elemento.

```
distancias_mah = lapply(1:3,
                        function(x) mahalanobis(pokemon[seleccion[,x],2:7],
                                                lista_vector_de_medias[[x]],
                                                lista_matriz_de_covarianzas[[x]]))
todos_juntos = unlist(distancias_mah)
todos_juntos_ordenados = names(todos_juntos[order(todos_juntos,decreasing = TRUE)])
indices_top_mah_outliers = as.numeric(todos_juntos_ordenados[1:numero_de_ouliers])
```

Tenemos los indices de los puntos que consideramos outliers y las distancias de mahalanobis respectivas.

```
pokemon[indices_top_mah_outliers,]
```

```
##     Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial Velocidad
## 231     Shuckle  20     10     230              10              230         5
## 122     Chansey 250      5       5              35              105        50
## 262     Blissey 255     10      10              75              135        55
## 104        Onix  35     45     160              30               45        70
## 489     Happiny 100      5       5              15               65        30
## 351     Wailmer 130     70      35              70               35        60
## 45   Jigglypuff 115     45      20              45               25        20
```

```
distancias_mah_outliers = unlist(distancias_mah)[indices_top_mah_outliers]
names(distancias_mah_outliers) = NULL
distancias_mah_outliers
```

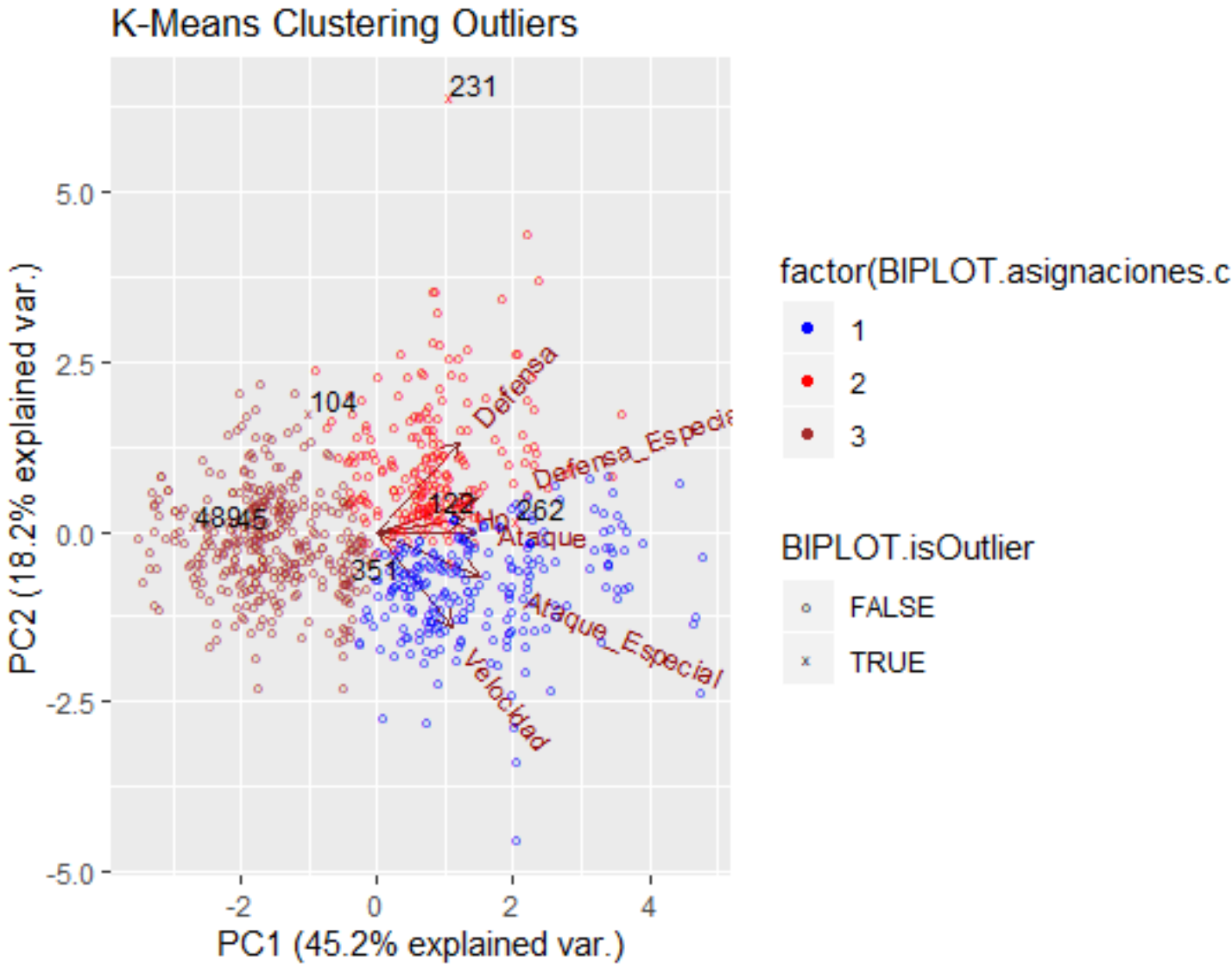
```
## [1]  4.328431  2.766425  7.114559 16.400755  2.494337  2.185523  8.533008
```

Veamos los resultados mediante un biplot.

```
numero.de.datos = nrow(pokemon)
is.kmeans.outlier.mah = rep(FALSE, numero.de.datos)
is.kmeans.outlier.mah[indices_top_mah_outliers] = TRUE

BIPLOT.isOutlier             = is.kmeans.outlier.mah
BIPLOT.cluster.colors        = c("blue","red","brown")
BIPLOT.asignaciones.clusters = indices_clustering
```

```
MiBiPlot_Clustering_Outliers(pokemon[,2:7], "K-Means Clustering Outliers")
```


K-Means Clustering Outliers
231
104
48945
122
262
351
Defensa
Defensa_Especia
Ataque
Ataque_Especial
Velocidad
5.0
2.5
0.0
-2.5
-5.0
-2
0
2
4
PC2 (18.2% explained var.)
PC1 (45.2% explained var.)
factor(BIPLOT.asignaciones.c
1
2
3
BIPLOT.isOutlier
FALSE
TRUE

7.3 Alternativa C.

Definiremos una función que nos calcule la distancia relativa de cada punto a su cluster. Con ello calcularemos los outliers, tendremos en cuenta no lo la distancia al centroide del punto, sino la distancias de los demás puntos del cluster.

```
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(pokemon_normalizado[,2:7],
                                             indices_clustering,
                                             centroides_normalizados,
                                             7)
top_outliers_kmeans_distancia_relativa$distancias
```

```
## [1] 6.321516 5.594774 5.574585 5.486384 5.038119 5.000824 4.842663
```

```
pokemon[top_outliers_kmeans_distancia_relativa$indices,]
```

```
##              Nom_Pokemon  Hp Ataque Defensa Ataque_Especial Defensa_Especial
## 430    DeoxysAttack Forme  50    180      20            180              20
## 231               Shuckle  20     10     230             10             230
## 262               Blissey 255     10      10             75             135
## 122               Chansey 250      5       5             35             105
## 423   KyogrePrimal Kyogre 100    150      90            180             160
## 425 GroudonPrimal Groudon 100    180     160            150              90
## 165   MewtwoMega Mewtwo Y 106    150      70            194             120
##     Velocidad
## 430       150
## 231         5
## 262        55
## 122        50
## 423        90
## 425        90
## 165       140
```