Análisis de datos Ómicos (M0-157) SOLUCIÓN de la Primera prueba de evaluación continua.

Table of contents

```
Warning: package 'knitr' was built under R version 4.4.1
```

Fecha de publicación del enunciado: 24/10/2024

Fecha límite para presentar la PEC: 6/11/2024[1]

Presentación y enunciado

Presentación y objetivos

Esta PEC completa la introducción a las ómicas mediante un ejercicio de repaso y ampliación que nos permite trabajar con algunas de las herramientas de este curso, en concreto, Bioconductor y la exploración multivariante de datos.

Para llevar a cabo óptimamente esta primera parte, tenéis que haberos familiarizado con

- las tecnologías ómicas,
- con las herramientas para trabajar con ellas,
  - Bioconductor y
  - github,
- con los contenedores de datos ómicos, como los `expressionSets`,
- y con las herramientas de exploración de datos, introducidas en la tercera actividad.

Descripción de la PEC

El objetivo de esta PEC es que planifiquéis y ejecutéis una versión simplificada del proceso de análisis de datos ómicos, a la vez que practicáis con algunas de las herramientas y métodos que hemos trabajado.

En concreto lo que tendréis que hacer es:

1. Seleccionar un dataset *de metabolómica* que podéis obtener de
- Este repositorio de github: https://github.com/nutrimetabolomics/metaboData/

[1] La fecha de entrega es la que se indica en el enunciado de la PEC. En caso de no coincidir con la indicada en el aula, ésta será la que predomine.

- Si lo preferís podéis usar algún dataset del repositorio metabolomicsWorkbench

2. Una vez descargados los datos cread un contenedor del tipo `SummarizedExperiment` que contenga los datos y los metadatos (información acerca del dataset, las filas y las columnas). La clase `SummarizedExperiment` es una extensión de ExpressionSet y muchas aplicaciones o bases de datos (como metabolomicsWorkbench) lo utilizan en vez de usar `expressionSet`.
3. Llevad a cabo una exploración del dataset que os proporcione una visión general del dataset en la línea de lo que hemos visto en las actividades de este reto.
4. Elaborad un informe que describa el proceso que habéis realizado, incluyendo la descarga de los datos, la creación del contenedor, la exploración de los datos y la reposición de los datos en github. El nombre del repositorio tiene que ser el siguiente: APELLIDO1-Apellido2-Nombre-PEC1. Por ejemplo en mi caso el repositorio se llamaría: "Sanchez-Pla-Alex-PEC1"
5. Cread un *repositorio de github*[2] que contenga:

- el informe,
- el objeto contenedor con los datos y los metadatos en formato binario (.Rda),
- el código R para la exploración de los datos
- los datos en formato texto y
- los metadatos acerca del dataset en un archivo markdown.

La dirección (url) del repositorio deberá estar incluida en la última sección del informe de forma clara.

Observad que, para entregar vuestra PEC tenéis que entregar **únicamente el informe**.

El resto de entregables de vuestra PEC deberán quedar en el repositorio de github.

Recursos

Los recursos para la solución de la PEC son los que se han proporcionado en el aula para las tres primeras unidades, es decir los materiales del curso y casos de estudio.

Tened en cuenta además que, tendréis que hacer un rápido aprendizaje de como usar `SummarizedExperimenten` vez de `ExpressionSet`. Podéis basaros en el tutorial sobre `SummarizedExperiment` que encontraréis en Bioconductor.

[2]Naturalmente si no tenéis cuenta de github deberéis crearos una. Es gratis y es muy sencillo.

Criterios de valoración

Tal como se indica en el plan docente, la nota de la PEC es aproximadamente proporcional a la duración. En la práctica esto significa que la primera PEC vale el 30% de la nota. (La segunda vale un 40% y la tercera, de nuevo un 30%)

Ahora bien, y como cosa importante, recordad que la PEC en si misma es un ejercicio de síntesis y aprendizaje en la que intenta valorar vuestra capacidad para resolver un problema muy parecido a los que se encuentra un/a bioinformática/a en su día a día. Esto quiere decir que para más de uno de los pasos que debéis realizar no hay una solución única. Plantead vuestra propia solución y explicad porqué creéis que es la adecuada. Entre otras cosas valoraremos:

- Capacidad de definir correctamente los objetivos a alcanzar
- Capacidad de organizar el análisis, obtención de los datos, preparación de los archivos etc.
- Dominio adecuado de las herramientas propias del tema (R, Rmarkdown, BioC)
- Capacidad de explicar qué y porqué se hace en cada paso.
- Capacidad de interpretar los resultados obtenidos.
- Capacidad de discutir las posibles limitaciones del estudio.
- Presentación del trabajo en un documento legible y bien organizado.

Código de honor

Cuando presentáis ejercicios individuales os adherís al código de honor de la UOC, con el que os comprometéis a no compartir vuestro trabajo con otros compañeros o a solicitar de su parte que ellos lo hagan. Asimismo, aceptáis que, de proceder así, es decir, en caso de copia probada, la calificación total de la PEC será de cero, independientemente

Solución

Selección del dataset

En principio cualquier dataset debería ser válido, siempre que disponga, de una matriz de datos, y de información acerca de los mismos, que, globalmente, podemos denominar los metadatos.

El catalogo de datasets, el archivo "DataCatalog.xls" contiene 6 datasets por lo que sorteamos con cual vamos a trabajar.

```
set.seed(123)
sample(1:6, 1)
```

```
[1] 3
```

Esto corresponde al dataset "2018-MetabotypingPaper"

Información sobre los datos

- Según la información del catálogo se trata de un dataset de metabolómica formado por 39 muestras y 639 variables o "features".
- A juzgar por la información en el archivo "Description.md" nos basta con los dos archivos:
  - `DataValues_S013.csv`: Clinical and metabolomic values for 39 patients at 5 time points.
  - `DataInfo_S013.csv`: Metadata. Information on each column in the "DataValues_S013.csv" file.

El primero contiene los "datos" y el segundo los "metadatos".

Carga de datos y metadatos:

Podemos leer estos dos archivos a R directamente desde sus URLs o descargarlos y leerlos del disco.

```
metadatosURL <- "https://raw.githubusercontent.com/nutrimetabolomics/metaboData/refs/heads/m
datosURL <- "https://raw.githubusercontent.com/nutrimetabolomics/metaboData/refs/heads/main/
```

```
metaDatos <- read.csv(metadatosURL, row.names = 1)
datos <- read.csv(datosURL, row.names = 1)
class(datos)
```

```
[1] "data.frame"
```

Fijémonos que el número de filas de los "metadatos" coincide con el número de columnas de los datos. Aunque en realidad la información sobre cada columna es escasa, parece que es la estructura que necesitamos.

Ahora bien, va a ser preciso que realicemos un retoque, debido a una peculiaridad de los contenedores, y es que *requieren que los datos se encuentren en matrices numéricas*, es decir no se admiten `data.frames` o `tibbles` con datos de distintos tipos.

Una ojeada al data.frame "datos" muestra como las cuatro primeras columnas (cinco contando al individuo) más que datos ómicos propiamente dichos, contienen información sobre los individuos. Esto ses lo que en los contenedores se suele asignar al "rowData", por lo que separaremos ambos datos en dos `data.frames`.

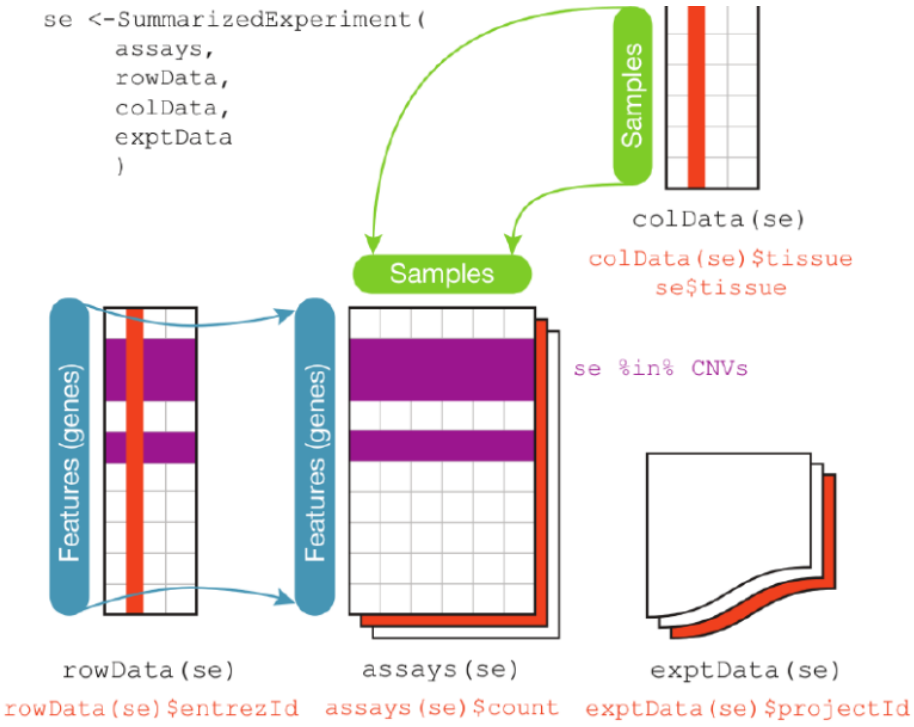
```
infoSamples <- datos[, 1:5]
if (ncol(datos) == 695) datos <- datos[, -c(1:5)]
if (nrow(metaDatos) == 695) metaDatos <- metaDatos[-c(1:5), ]
```

Con estos dos componentes podemos ya crear el contenedor.

Creando un contenedor

La clase `SummarizedExperiment`proporciona un contenedor para datos ómicos, inspirado en el antigua clase `ExpressionSet`, pero algo más general.

```
knitr::include_graphics("images/clipboard-388301461.png")
```



Tras instalar o cargar el paquete podremos crear un nuevo contenedor.

```
if (!require(BiocManager)) install.packages("BiocManager")
if (!require(SummarizedExperiment)) BiocManager.install("SummarizedExperiment")
```

Para ello utilizaremos los tres bloques de información que hemos recogido

- Los datos (el único imprescindible), - La información sobre los individuos, que irá al campo `rowData`

- La información sobre los variables que irá al campo `colData`.

En primer lugar convertimos los datos y la información de las muestras al tipo de datos adecuado para trabajar con ellos.

Ponemos como factores los nombres del objeto infoSamples y asignamos nombres informativos a las filas.

```
library(dplyr)

# Convertir las columnas 2 (SURGERY), 4 (GENDER) y 5 (Group) a factores
infoSamples <- infoSamples %>%
    mutate(SURGERY = as.factor(SURGERY), GENDER = as.factor(GENDER), Group = as.factor(Group


# Crear el nuevo nombre para las filas
infoSamples <- infoSamples %>%
    mutate(RowName = paste0(toupper(substr(SURGERY, 1, 1)), "_", Group, "_", SUBJECTS))
rownames(infoSamples) <- infoSamples$RowName
infoSamples <- infoSamples %>%
    select(-RowName)
head(infoSamples)
```

```
      SUBJECTS SURGERY AGE GENDER Group
B_1_1        1 by pass  27      F     1
B_2_2        2 by pass  19      F     2
B_1_3        3 by pass  42      F     1
B_2_4        4 by pass  37      F     2
T_1_5        5 tubular  42      F     1
B_2_6        6 by pass  24      F     2
```

Con los datos procedemos de forma similar: los convertimos en una matriz numérica y nombramos las filas de la misma forma que las filas de infoSamples,

```
matDatos <- as.matrix(datos)
if (sum(rownames(matDatos) != infoSamples$SUBJECTS) == 0) rownames(matDatos) <- rownames(inf
```

Ahora podemos crear el objeto de clase `SummarizedExperiment`

```
mySE <- SummarizedExperiment(assays = list(rawValues = matDatos), rowData = infoSamples,
    colData = metaDatos)
show(mySE)
```

```
class: SummarizedExperiment
dim: 39 690
metadata(0):
assays(1): rawValues
rownames(39): B_1_1 B_2_2 ... T_1_38 B_1_39
rowData names(5): SUBJECTS SURGERY AGE GENDER Group
colnames(690): MEDDM_T0 MEDCOL_T0 ... SM.C24.0_T5 SM.C24.1_T5
colData names(3): VarName varTpe Description
```

Una cuestión que puede resultar desconcertante es el hecho de que tanto los "rowData" como los "colData" pueden ser distintos, cuando no inexistentes, entre experimentos.

Esto se explica por dos motivos.

- Por un lado está la razón obvia, de que cada experimento es distinto y puede generar variables diferentes que requieran de distinta información complementaria.
- Por el otro, está el hecho de que, a pesar de los múltiples esfuerzos que se llevan a cabo para estandarizar los *mínimos metadatos* necesarios para repositar datos, este esfuerzo casi nunca acaba en acuerdos definidos y perdurables, por lo que en la mayoría de los casos, y, si hay suerte, los investigadores aportan lo que consideran razonable, que obviamente difiere entre unos y otros casos.

Análisis exploratorio de los datos

Una vez extraídos los datos y la información podemos proceder a realizar una exploración básica de los mismos.

Metadatos y posibles agrupaciones

Una rápida exploración de los metadatos nos proporciona información sobre los posibles grupos en que se organizan las observaciones.

```
data <- rowData(mySE)[, c("SURGERY", "AGE", "GENDER", "Group")]

# Convertir a data.frame para facilidad de manipulación
data <- as.data.frame(data)
```

```
# Calcular tablas de frecuencias para las columnas 1 (SURGERY), 3 (GENDER) y 4
# (Group)
(freq_surgery <- table(data$SURGERY))
```

```
by pass tubular
     26      13
```

```
(freq_gender <- table(data$GENDER))
```

```
 F  M
27 12
```

```
(freq_group <- table(data$Group))
```

```
 1  2
24 15
```

```
(freq_surgery_Group <- table(data$SURGERY, data$Group))
```

```
           1  2
  by pass 13 13
  tubular 11  2
```

```
(freq_surgery_Group_Gender <- table(data$SURGERY, data$Group, data$GENDER))
```

```
, ,  = F

           1  2
  by pass 10 10
  tubular  6  1

, ,  = M

           1  2
  by pass  3  3
  tubular  5  1
```

```
# Calcular el resumen numérico para la columna 2 (AGE)
(summary_age <- summary(data$AGE))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  19.00   35.00   41.00   40.79   46.00   59.00
```

Como puede observarse los datos no están balanceados pero están relativamente repartidos entre grupos.

Valores faltantes

Muchos datasets de ómicas contienen valores faltantes, lo que, aparte de la dificultad en la interpretación conlleva que ciertas pruebas no puedan hacerse.

Es relativamente sencillo identificar los valores faltantes. El decidir que hacer con ellos, que puede ir desde eliminarlos a imputarlos pasando por ignorarlos, es mucho más difícil.

Aquí, como se trata de un ejercicio académico, nos limitaremos a asignarles un valor mínimo asumiendo que los valores faltantes lo están o bien porque están ausentes o porque no se han detectado.

El paquete `naniar` permite una rápida exploración de la distribución de dichos valores.

```
if (!require(naniar)) install.packages("naniar")
library(naniar)
```

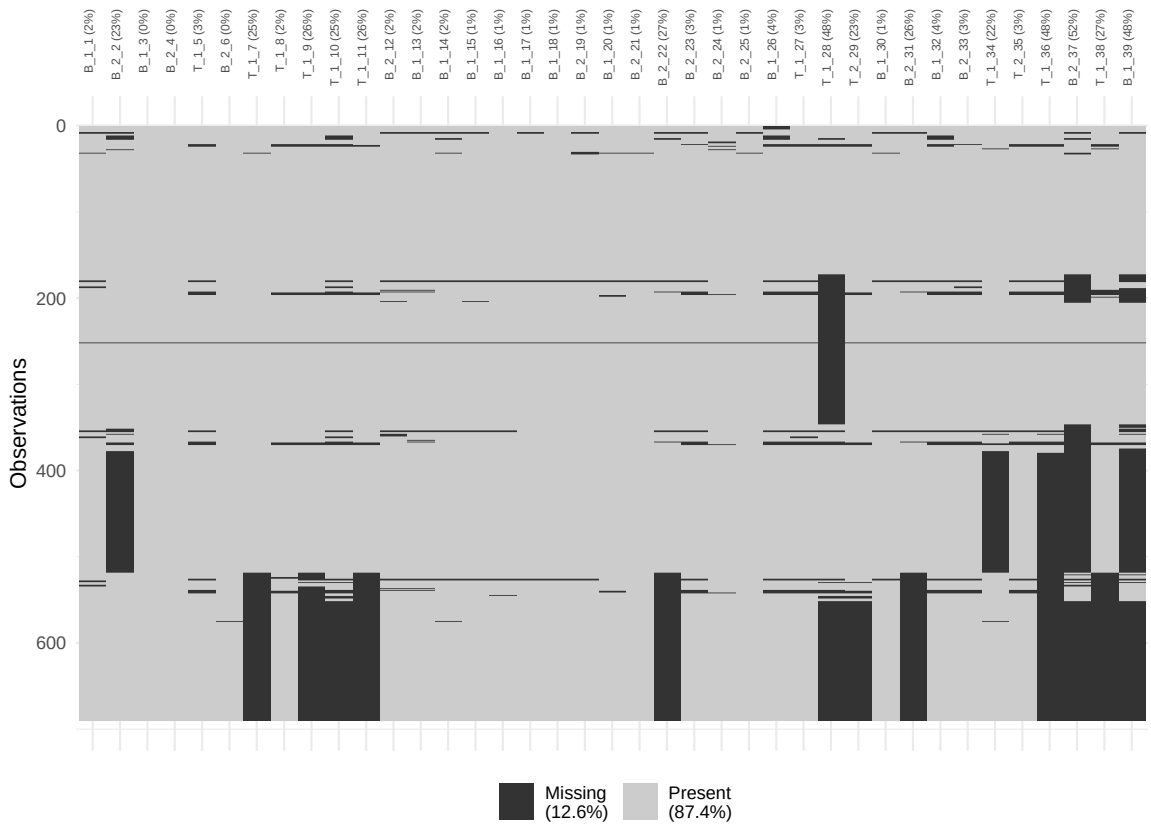
Este paquete requiere que los datos se proporcionen como `data.frame` por lo que antes de invocarlo se transforma en `data.frame` la matriz de datos contenida en `assay(mySE, "rawValues")`.

```
# Visualización de valores faltantes
dataf <- as.data.frame(t(assay(mySE, "rawValues")))
# rownames(dataf) <- rownames(rawVals) colnames(dataf) <- colnames(rawVals)
```

Aunque `naniar`se puede usar directamente, utilizamos `ggplot2` para asegurarnos que las etiquetas se visualizan bien.

```
library(ggplot2)

# Visualización con ajuste del tamaño de etiquetas
vis_miss(dataf) + theme(axis.text.x = element_text(size = 6, angle = 90, hjust = 1))
```



Este gráfico muestra como algunas muestras contienen un altísimo porcentaje de valores faltantes.

Esto puede confirmarse mediante un resumen numérico. Aunque este resumen puede hacerse variable a variable o muestra a muestra, nos limitamos a un resumen global.

```
n_miss(dataf)
```

```
[1] 3390
```

```
n_complete(dataf)
```

```
[1] 23520
```

```
prop_miss(dataf)
```

```
[1] 0.1259755
```

```
prop_complete(dataf)
```

```
[1] 0.8740245
```

```
pct_miss(dataf)
```

```
[1] 12.59755
```

```
pct_complete(dataf)
```

```
[1] 87.40245
```

Al tratarse de valores de metabolómica es habitual no detectar valores lo que, a la hora del análisis conlleva decisiones complicadas.

Una opción simple es asignar un cero (o un 1) a los valores faltantes. Tomar 1 determina que, si trabajamos con los logaritmos quedaran como ceros.

```
rawValues <- assay(mySE, "rawValues")
rawNoMissings <- rawValues
rawNoMissings[is.na(rawNoMissings)] <- 1
sum(is.na(rawNoMissings))
```

```
[1] 0
```

```
assays(mySE)$rawNoMissings <- rawNoMissings
show(mySE)
```

```
class: SummarizedExperiment
dim: 39 690
metadata(0):
assays(2): rawValues rawNoMissings
rownames(39): B_1_1 B_2_2 ... T_1_38 B_1_39
rowData names(5): SUBJECTS SURGERY AGE GENDER Group
colnames(690): MEDDM_T0 MEDCOL_T0 ... SM.C24.0_T5 SM.C24.1_T5
colData names(3): VarName varTpe Description
```

Descripción de los indivíduos y las muestras

Acerca de los objetos de la exploración

Antes de empezar con la exploración recordemos una característica de los análisis de datos de alto rendimiento: Cuando tenemos pocos individuos y miles de variables, se produce, en cierta forma un intercambio de papeles en los resúmenes de datos. Por ejemplo, en este estudio nos encontramos con 39 muestras y `ncol (mySE)` variables.

- En un análisis tradicional pensaríamos que un diagrama de cajas múltiple tendría `ncol (mySE)` cajas, cada una hecha de `nrow (mySE)` puntos.
- Sin embargo, en estudios de datos ómicos el boxplot que se presenta *suele ser por muestras y ni por variables* es decir de `nrow (mySE)` cajas (una por individuo) y no `ncol (mySE)`, una por variable.

Es decir, al trabajar con datos ómicos, es habitual que, parte de la exploración de los datos, se realice **en el espacio de los individuos y no en el de las variables**

En la práctica esto no tiene grandes repercusiones porque básicamente estamos haciendo un cambio de perspectiva y siempre podemos pasar de una a otra, o hacerlas ambas.

Distribucion por muestras / indivíduos

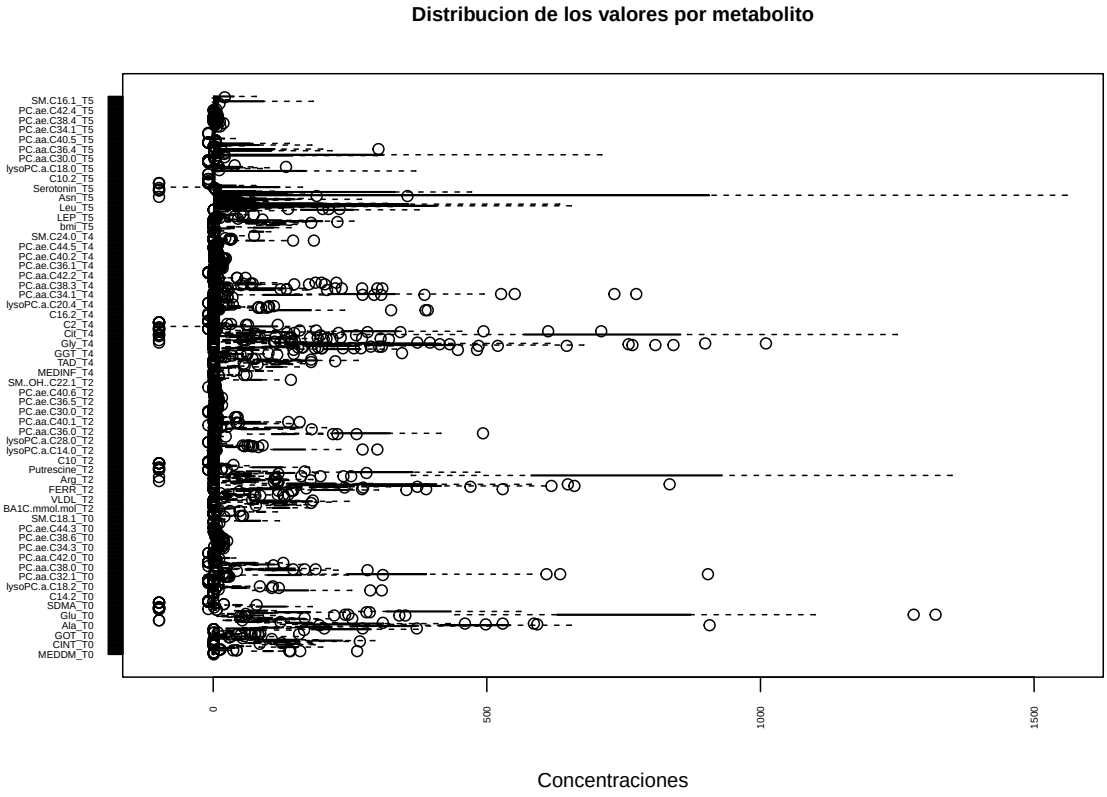
Empezamos con un boxplot múltiple que nos muestra como se distribuyen los en cada individuo.

Para enfatizar que trabajamos a partir del contenedor no accederemos directamente a los datos sino que lo haremos mediante las funciones de acceso a los mismos:

- `assay`
- `nrow`
- `ncol`

Empezamos por observar la distribución de valores metabolito a metabolito en cada muestra.

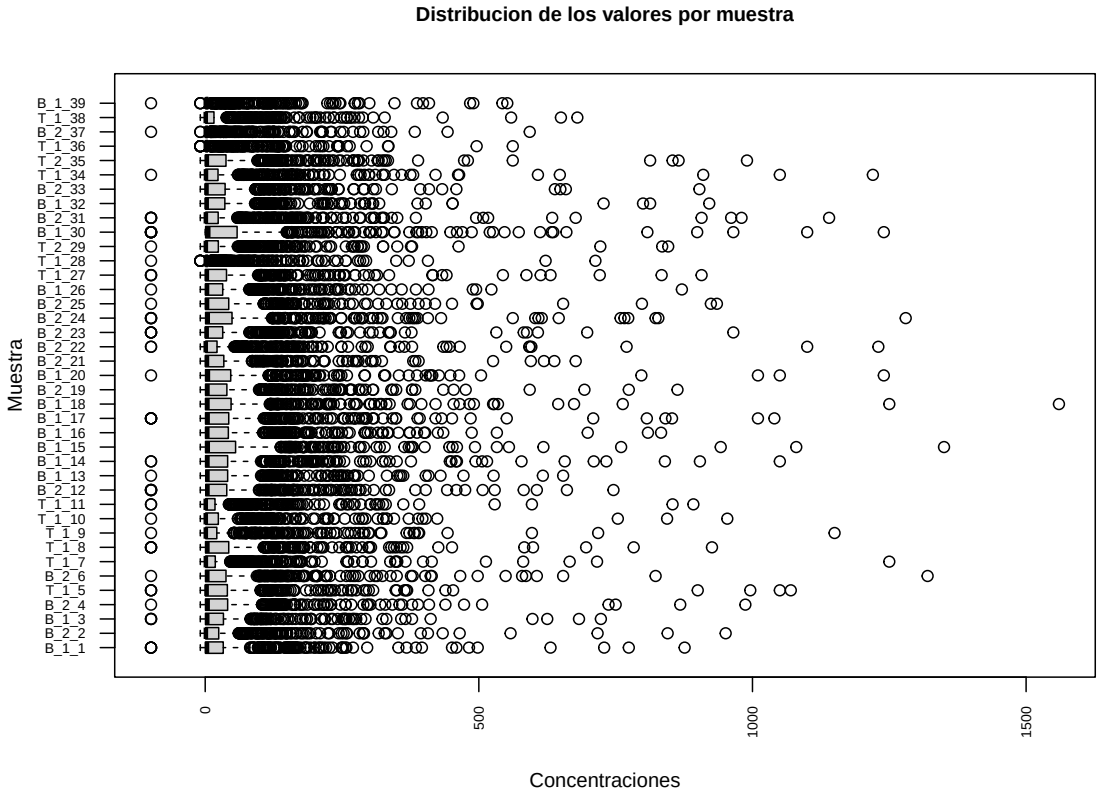
```
boxplot(assay(mySE, "rawNoMissings"), xlab = "Concentraciones", cex.lab = 0.8, horizontal =
    cex.axis = 0.4, las = 2, main = "Distribucion de los valores por metabolito",
    cex.main = 0.8)
```



Como puede verse, el rango de variación varía bastante entre metabolitos.

Si procedemos con el gráfico más habitual, el "boxplot por muestras" vemos que la variación en términos relativos es inferior

```
boxplot(t(assay(mySE, "rawNoMissings")), ylab = "Muestra", xlab = "Concentraciones",
    cex.lab = 0.8, horizontal = TRUE, cex.axis = 0.6, las = 2, main = "Distribucion de los v
    cex.main = 0.8)
```



Transformación de los datos

En vista de los resultados anteriores podemos considerar tomar logaritmos como una opción para simetrizar los datos.

La ventaja de hacerlo sobre un contenedor es que nos permitirá tener dos capas, una con los valores originales y otra con los logaritmos manteniendo los mismos metadatos:

```
logNoMissings <- log(assay(mySE, "rawNoMissings"))
assays(mySE)$logNoMissings <- logNoMissings
show(mySE)
```

```
class: SummarizedExperiment
dim: 39 690
metadata(0):
assays(3): rawValues rawNoMissings logNoMissings
rownames(39): B_1_1 B_2_2 ... T_1_38 B_1_39
rowData names(5): SUBJECTS SURGERY AGE GENDER Group
colnames(690): MEDDM_T0 MEDCOL_T0 ... SM.C24.0_T5 SM.C24.1_T5
```

```
colData names(3): VarName varTpe Description
```

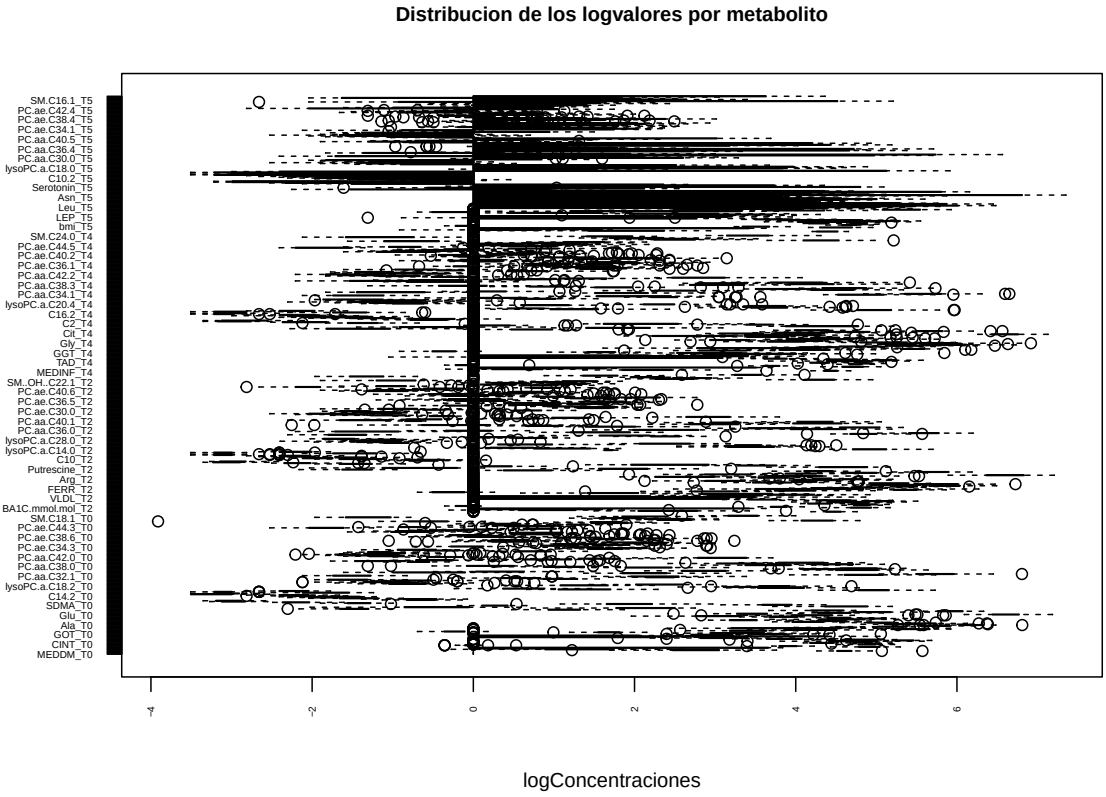
Podemos grabar el objeto creado para posteriores reutilizaciones.

```
save(mySE, file = "metabodatSE.Rda")
```

Visualización con datos transformados

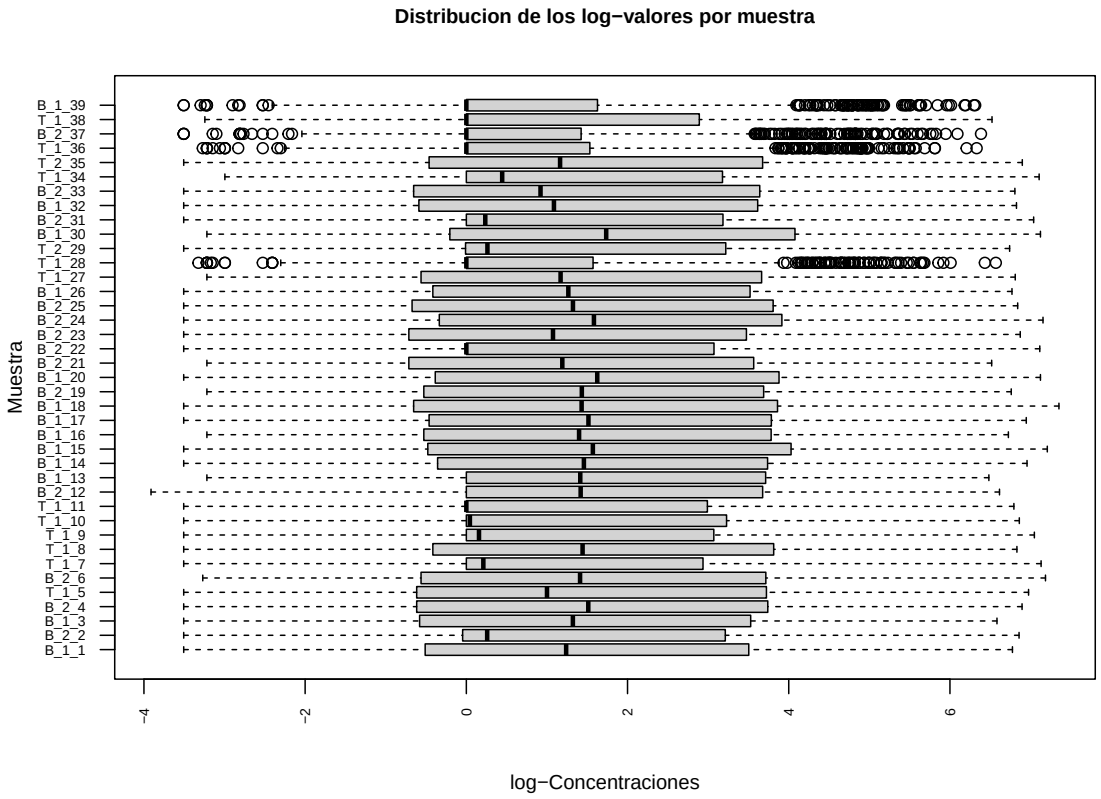
Ahora podemos repetir los diagramas de caja para ver el efecto de la transformación.

```
boxplot(assay(mySE, "logNoMissings"), xlab = "logConcentraciones", cex.lab = 0.8,
    horizontal = TRUE, cex.axis = 0.4, las = 2, main = "Distribucion de los logvalores por m
    cex.main = 0.8)
```



Y con las muestras:

```
boxplot(t(assay(mySE, "logNoMissings")), ylab = "Muestra", xlab = "log-Concentraciones",
    cex.lab = 0.8, horizontal = TRUE, cex.axis = 0.6, las = 2, main = "Distribucion de los l
    cex.main = 0.8)
```



Como puede verse la transformación logarítmica ha simetrizado considerablemente algunas muestras, pero aquellas que tenían muchos missings aparecen peor ya que concentran un elevado número de ceros.

Pros y contras de la transformación logarítmica

En general esto suele hacerse así porque conlleva una serie de ventajas:

1. Mejora la simetría de los datos lo cual facilita la aplicación de métodos estadísticos que asumen normalidad o simetría, como pruebas t o ANOVA.
2. Estabiliza la varianza evitando que la variabilidad aumentar con la media (es decir, los valores más altos presentan mayor variabilidad).
3. Reduce el impacto de valores extremos

Ahora bien, es importante recalcar que esta transformación *es una opción* no un requerimiento_ puesto que tampoco está libre de problemas:

1. Puede reducir la interpretabilidad, al cambiar la escala de los datos.

2. Puede conllevar una pérdida de información, por ejemplo al reducir las diferencias absolutas entre muestras.
3. Puede no ser necesaria si los métodos estadísticos que se emplearán son robustos frente a la falta de simetría o a la heterocedasticidad.

Visualización simultánea de indivíduos y variables

Una forma compacta de visualizar todos los datos es mediante un `heatmap` o mapa de calor.

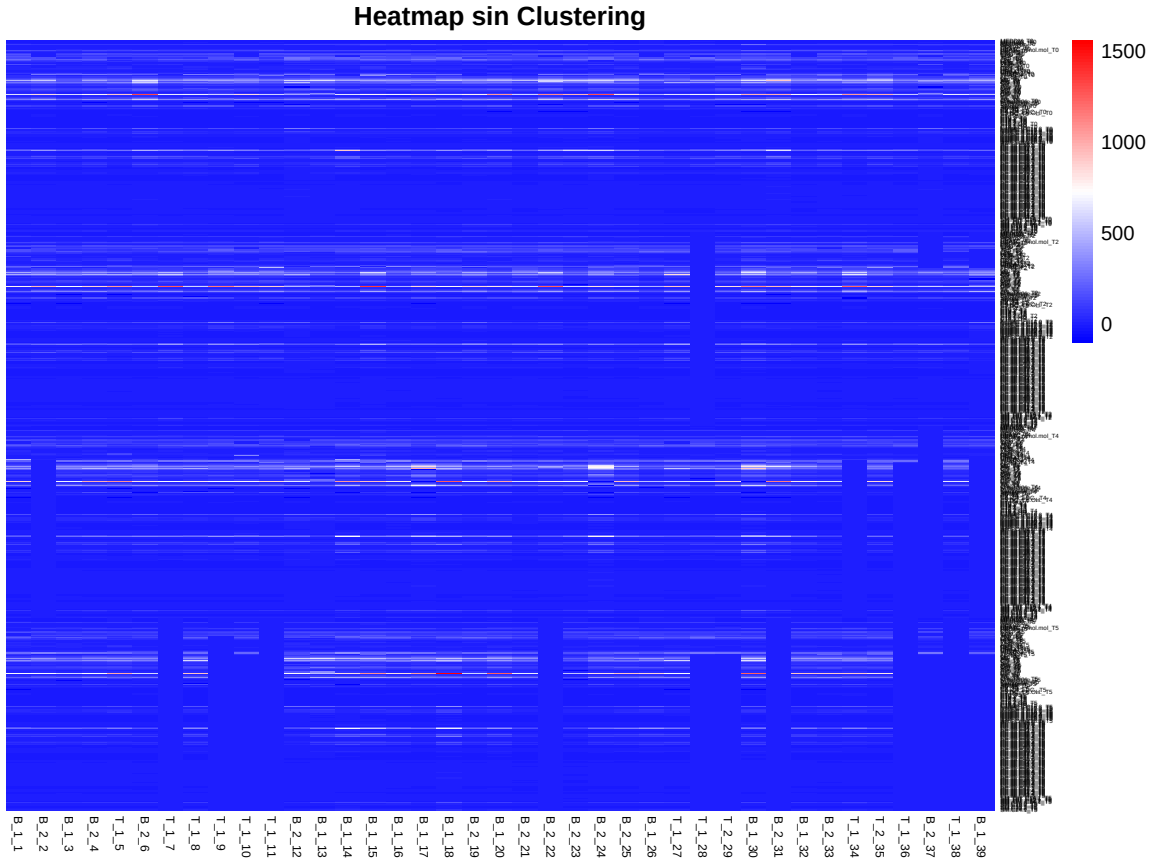
Aunque este gráfico puede utilizarse para visualizar toda la matriz de expresión sin más, es habitual realizarlo acompañado de un clustering jerárquico en las filas y/o las columnas para detectar posibles grupos de muestras o de variables que varíen de forma coordinada.

Podemos usar el paquete `pheatmap` aunque hay multitud de paquetes en Bioconductor para realizar este tipo de gráficos.

```
if (!require(pheatmap)) install.packages("pheatmap")
library(pheatmap)
```

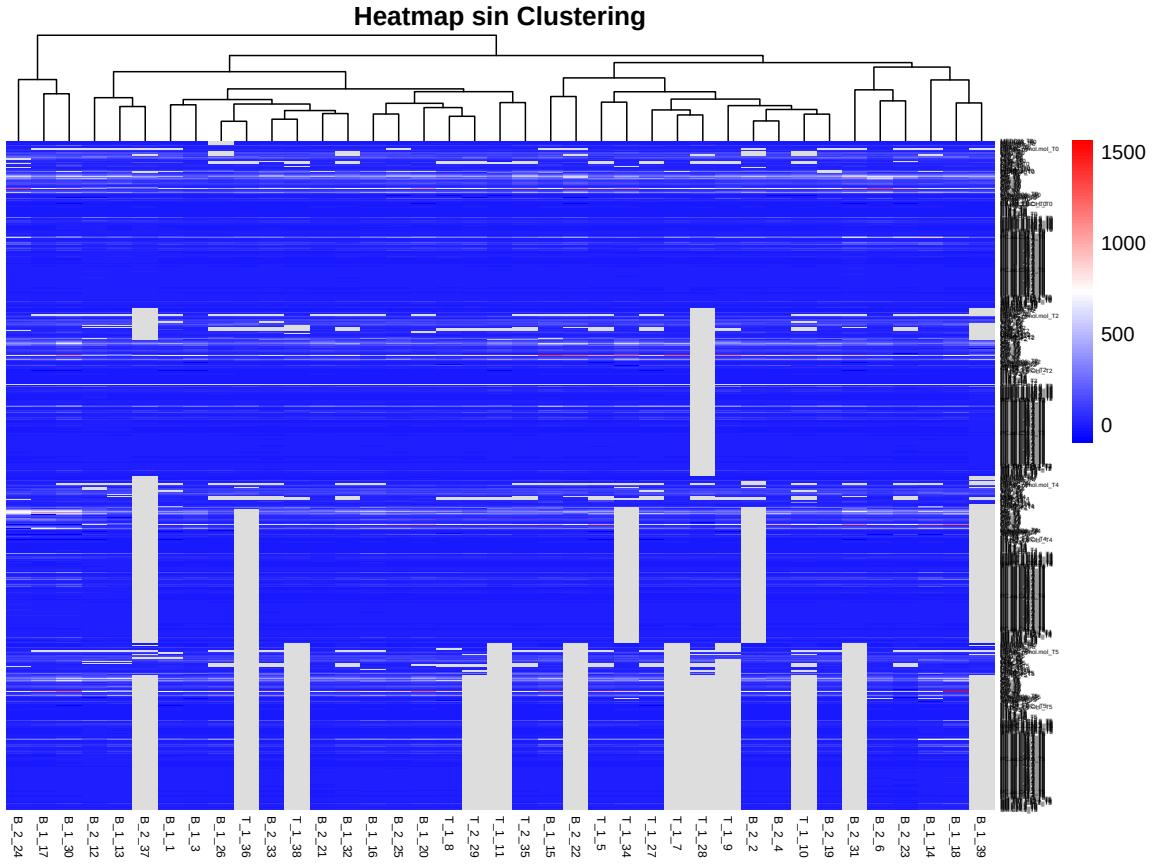
Si no hacemos clustering de filas ni de columnas:

```
# Heatmap sin clustering
pheatmap(t(assay(mySE, "rawNoMissings")), cluster_rows = FALSE, cluster_cols = FALSE,
    main = "Heatmap sin Clustering", fontsize_row = 3, fontsize_col = 6, color = colorRampPa
        "white", "red"))(100))
```



Si pedimos que se agrupen filas y columnas se produce un error debido a la presencia de valores faltantes. Podemos visualizarlo agrupando solo por observaciones.

```
# Heatmap sin clustering
pheatmap(t(assay(mySE, "rawValues")), cluster_rows = FALSE, cluster_cols = TRUE,
    main = "Heatmap sin Clustering", fontsize_row = 3, fontsize_col = 6, color = colorRampPa
        "white", "red"))(100))
```



Análisis de componentes principales

Un análisis en componentes principales puede facilitar la visualización de los datos en dimensión reducida y, sobretodo, detectar posibles patrones que no se detecten a simple vista.

Existen muchos paquetes de R que permiten hacer un análisis de componentes principales y algunos proporcionan gráficos muy vistosos de manera muy sencilla.

Puesto que trabajamos con datos genómicos podemos explorar Bioconductor donde encontramos el paquete PCAtools con una viñeta relativamente sencilla de adaptar a nuestro problema.

Este paquete no trabaja con datos de tipo `SummarizedExperiment` sino que requiere una matriz o data.frame con datos numéricos y otra con los meta-datos.

```
if (!require(PCAtools)) BiocManager::install("PCAtools", dep = TRUE)
```

```
library(PCAtools)
x <- t(assay(mySE, "rawNoMissings"))
```

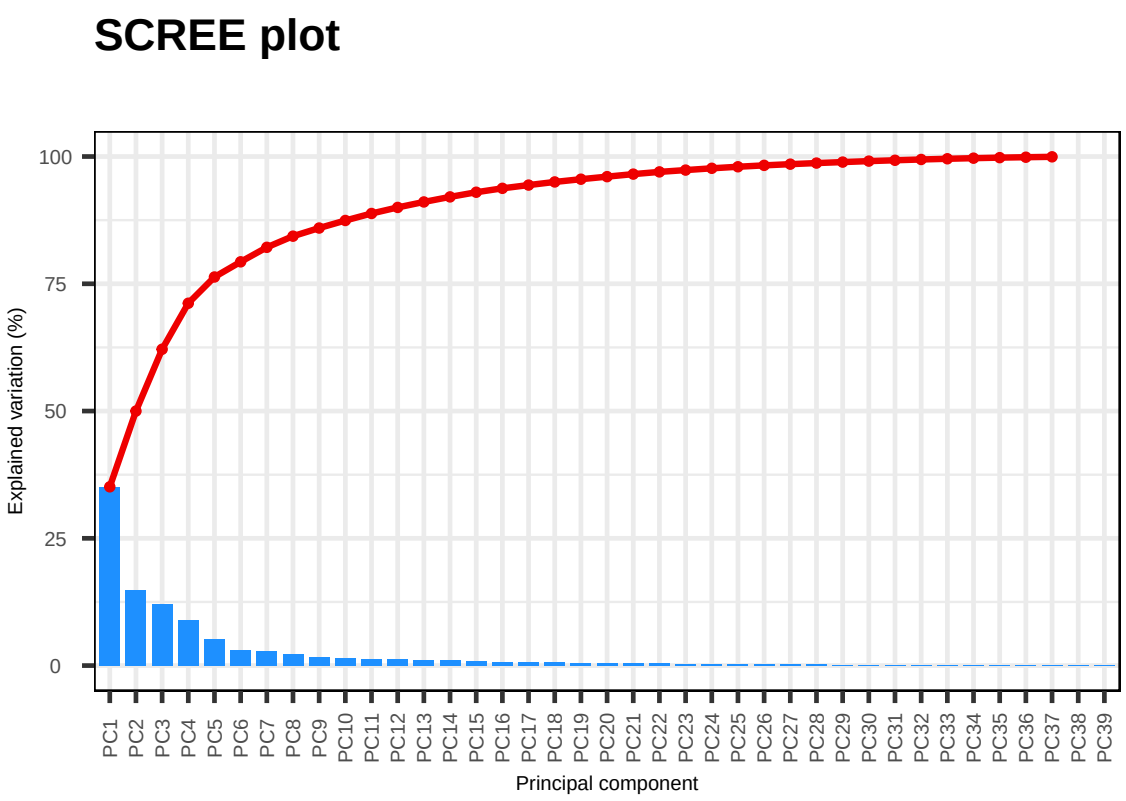
```
p <- pca(x, metadata = rowData(mySE), removeVar = 0.1)
class(p)
```

```
[1] "pca"
```

Ahora, con el objeto "p" y siguiendo la viñeta podemos realizar la exploración de los datos.

En primer lugar vemos el peso relativo de cada componentes con la función `screeplot` que muestra el porcentaje de variabilidad explicado por cada componente.

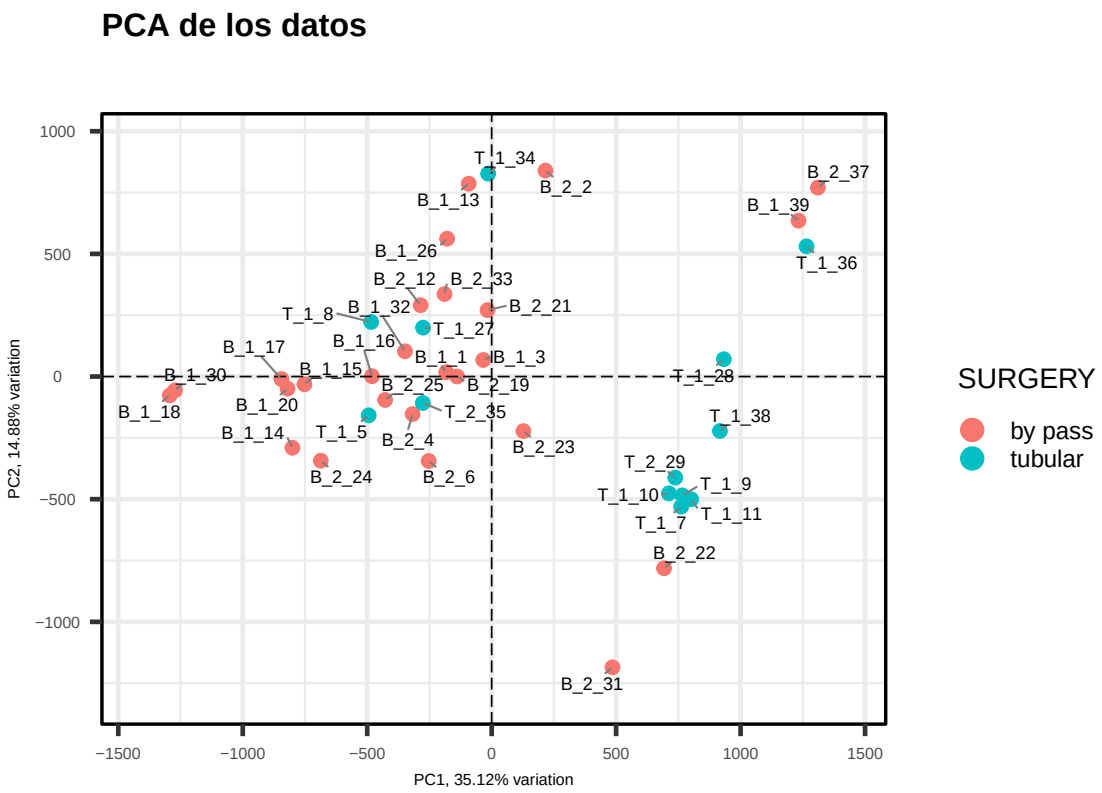
```
screeplot(p, axisLabSize = 10, titleLabSize = 22)
```



Las dos primeras componentes explican un 50% de la variabilidad con lo que con las 2-3 primeras componentes bastará para explorar estos datos.

La función `biplot` permite visualizar únicamente las muestras o, en el mismo gráfico, las muestras y las variables cuyos valores están más correlacionadas con las componentes.

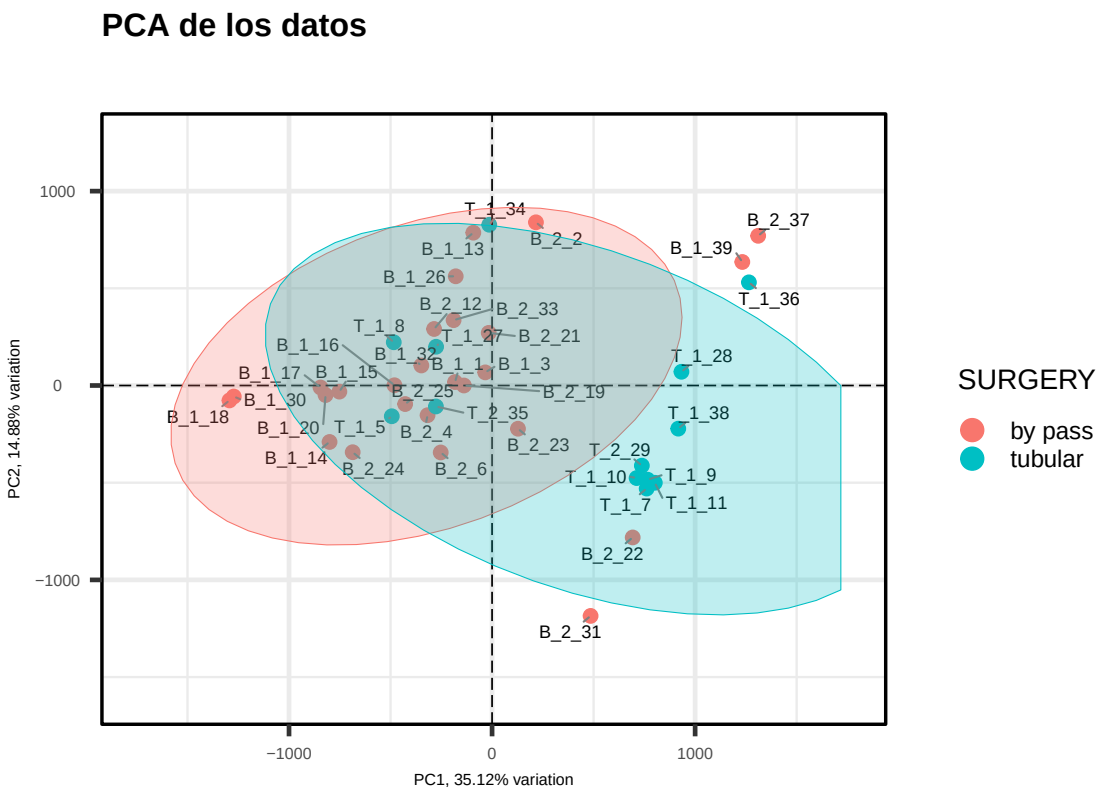
```
biplot(p, showLoadings = FALSE, labSize = 3, pointSize = 3, axisLabSize = 8, title = "PCA de
    sizeLoadingsNames = 3, colby = "SURGERY", hline = 0, vline = 0, legendPosition = "right"
```



En este caso podemos ver, al colorear por "SURGERY" (una de las columnas de los metadatos), como hay una cierta separación por la 1ª PC, aunque no todas las muestras quedan bien separadas.

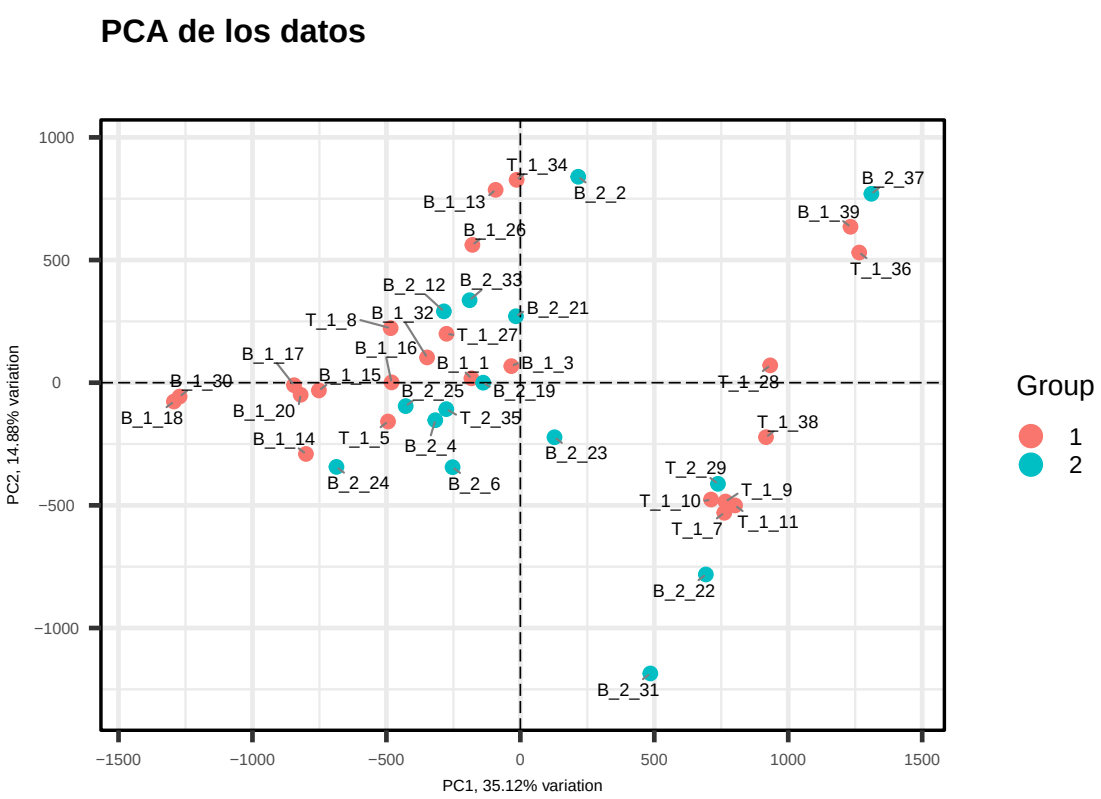
Esto se ve más claro si usamos una elipse de confianza para cada grupo.

```
biplot(p, showLoadings = FALSE, labSize = 3, pointSize = 3, axisLabSize = 8, ellipse = TRUE,
    title = "PCA de los datos", sizeLoadingsNames = 3, colby = "SURGERY", hline = 0,
    vline = 0, legendPosition = "right")
```



Podemos colorear por la otra variable de los metadatos, “Group”.

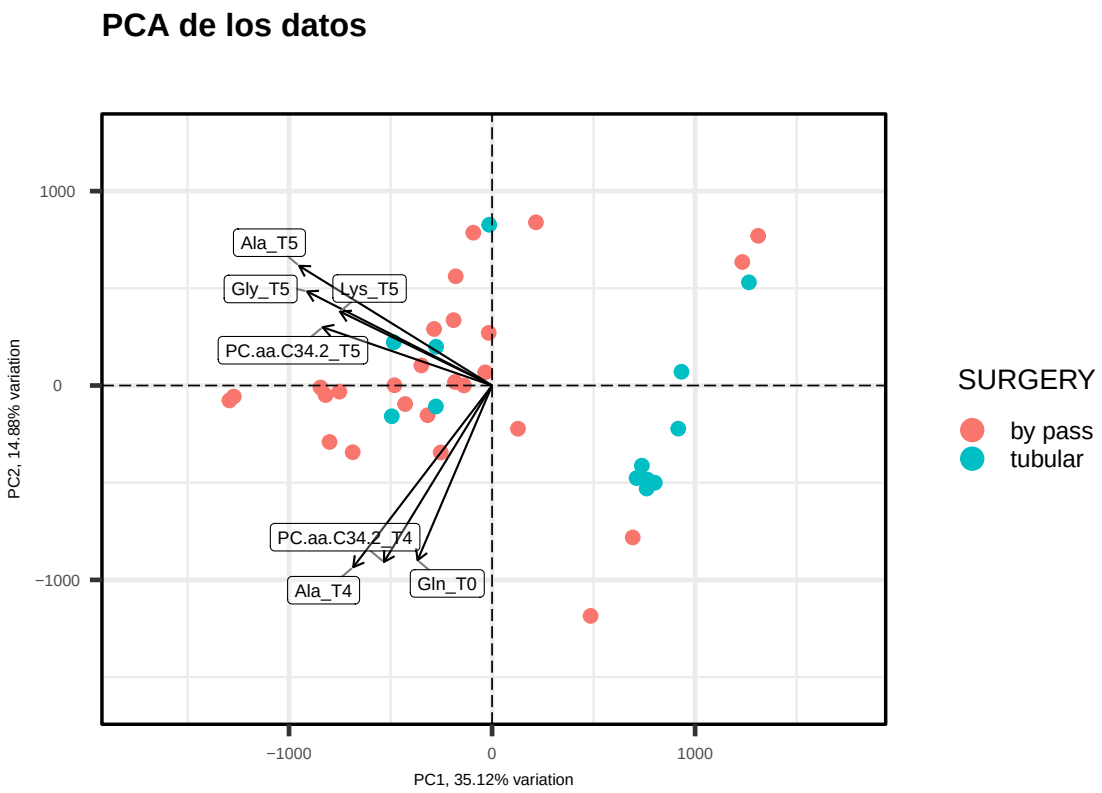
```
biplot(p, showLoadings = FALSE, labSize = 3, pointSize = 3, axisLabSize = 8, title = "PCA de
    sizeLoadingsNames = 3, colby = "Group", hline = 0, vline = 0, legendPosition = "right")
```



En este caso la separación está menos claramente asociada a los grupos por lo que no investigaremos más.

Si queremos ver que metabolitos están asociados con las componentes podemos hacerlo usando la opción. `showLoadings=TRUE`

```
biplot(p, showLoadings = TRUE, labSize = 3, pointSize = 3, axisLabSize = 8, title = "PCA de
    sizeLoadingsNames = 3, colby = "SURGERY", lab = NULL, hline = 0, vline = 0, legendPositi
```



Hay metabolitos claramente asociados con valores altos o bajos de la segunda PC, pero no de la primera.

Esto se puede ver, también mediante un gráfico de los pesos de cada componente donde aparecen las variables más correlacionadas con cada una.

La función `plotloadings` determina, para cada componente, las variables que se encuentran dentro del 5% superior/inferior del rango de cargas y luego crea una lista de consenso. Se puede modificar el límite de inclusión/exclusión de variables con el parámetro `rangeRetain`, donde 0,01 equivale al 1% superior/inferior del rango de cargas por PC.

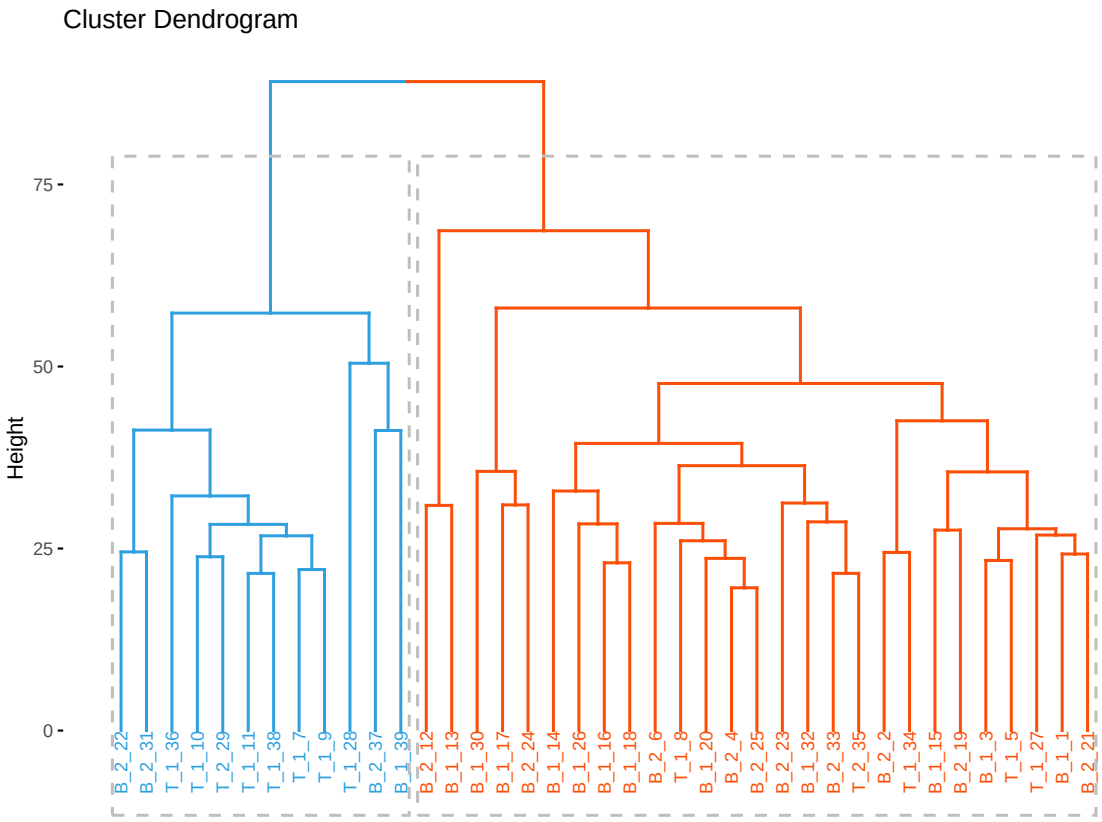
```
{r. plotloadings}   plotloadings(p,                rangeRetain = 0.01,
labSize = 3)
```

Visualización en un cluster jerárquico

Los resultados obtenidos se confirman si realizamos un agrupamiento jerárquico y lo visualizamos con un dendrograma

```
if (!require(factoextra)) install.packages("factoextra")
res.hc <- assay(mySE, "rawNoMissings") %>%
    scale() %>%
    dist(method = "euclidean") %>%
    hclust(method = "ward.D2")

library(factoextra)
fviz_dend(res.hc, cex = 0.6, k = 2, k_colors = c("#2E9FDF", "#FC4E07"), color_labels_by_k =
    rect = TRUE)
```



Como puede observarse hay una tendencia bastante clara, aunque no en todos los individuos, a quedar separados por el tipo de cirugía, "B" o "T".

Resumen y conclusiones

En este ejercicio hemos explorado un dataset con un conjunto de datos de metabolómica escogido al azar de un repositorio.

Hemos utilizado la información disponible[3] para definir unos metadatos básicos con información sobre los indivíduos (las observaciones) y hemos combinado datos y metadatos en un objeto de clase `SummarizedExperiment`.

Los datos resultantes 39 muestras con 690 variables contienen un elevado percentaje de valores faltantes (12,6%). De forma informal se han imputado a un valor mínimo de 1 asumiendo que se trataba de valores no detectados.

Sobre estos datos hemos realizado un análisis descriptivo básico, que muestra una cierta heterogeneidad en los datos y sugiere la posibilidad de realizar una transformación logarítmica.

Finalmente un análisis de componentes principales y un análisis de cluster jerárquico muestra una tendencia a separar los datos en dos grupos, definidos principalmente por el tipo de cirugía, aunque esta separación no se aplica a todos los indivíduos por lo que parece razonable continuar depurando los datos, mejorando la gestión de valores faltantes, valores extremos y realizando algunas transformaciones previas a la visualización.

Repositorio con resultados

El estudio realizado se ha depositado en un repositorio de github que contiene, tal como se solicitaba:

- el informe del análisis, en fomato PDF
- el objeto contenedor, con los datos y los metadatos en, formato binario (.Rda),
- el código R para la exploración de los datos.
- los datos en formato texto y
- los metadatos acerca del dataset en un archivo markdown.

Aunque esto se puede hacer manualmente, también es posible automatizar el proceso:

Preparación de archivos para el Repositorio

En primer lugar preparamos los archivos, por ejemplo cambiándoles el nombre o creando de nuevos, y los copiamos a un directorio:

- El informe se encuentra en el archivo "UOC-MU-AD0-2024-25-S2-PEC1-Solucion.pdf" se tiene que copia a "InformeAnalisis.pdf"
- El objeto contenedor con los datos se ha salvado a un objeto binario "metaboDatSE.Rda"
- El código R se ha extraido del archivo .qmd, con la instrucción `knitr::purl` a un archivo "UOC-MU-AD0-2024-25-S2-PEC1-Solucion.R" que se copia a "codigoAnalisis.R"

[3] Aunque el repositorio contiene información que nos permitiría conocer más cosas sobre el problema para el que se generaron los datos no lo hemos hecho así para contener el tamaño de la PEC.

- Los metadatos se encuentran en los objetos rowData(mySE) y colData(mySE) y se copian a un archivo “metaDatos.md”

```
# Definir nombres de archivos
informe_origen <- "UOC-MU-AD0-2024-25-S2-PEC1-Solucion.pdf"
informe_destino <- "InformeAnalisis.pdf"

codigo_origen <- "UOC-MU-AD0-2024-25-S2-PEC1-Solucion.R"
codigo_destino <- "codigoAnalisis.R"

objeto_binario <- "metaboDatSE.Rda"
metadatos_destino <- "metaDatos.md"

# Realizar las copias de los archivos
file.copy(informe_origen, informe_destino, overwrite = TRUE)
```

```
[1] FALSE
```

```
file.copy(codigo_origen, codigo_destino, overwrite = TRUE)
```

```
[1] TRUE
```

```
metaDatos_path <- "metaDatos.md"
file.create(metaDatos_path)
```

```
[1] TRUE
```

```
# Obtener rowData y colData de mySE
row_data <- SummarizedExperiment::rowData(mySE)
col_data <- SummarizedExperiment::colData(mySE)

# Función para formatear datos como tabla Markdown
to_markdown_table <- function(data, title) {
    # Crear el encabezado de la tabla
    headers <- paste0("| ", paste(colnames(data), collapse = " | "), " |")
    separator <- paste0("| ", paste(rep("---", ncol(data)), collapse = " | "), " |")

    # Crear las filas
    rows <- apply(as.data.frame(data), 1, function(row) {
        paste0("| ", paste(row, collapse = " | "), " |")
    })
```

```
    # Combinar título, encabezado y filas
    c(paste0("## ", title), headers, separator, rows, "")
}

# Escribir contenido al archivo metaDatos.md
cat("# Metadatos del análisis\n\n", file = metaDatos_path)

# Añadir rowData
if (ncol(row_data) > 0) {
    cat(to_markdown_table(row_data, "Row Data (Información por individuo)"), sep = "\n",
        file = metaDatos_path, append = TRUE)
} else {
    cat("## Row Data: No disponible\n\n", file = metaDatos_path, append = TRUE)
}

# Añadir colData
if (ncol(col_data) > 0) {
    cat(to_markdown_table(col_data, "Column Data (Información por variable)"), sep = "\n",
        file = metaDatos_path, append = TRUE)
} else {
    cat("## Column Data: No disponible\n\n", file = metaDatos_path, append = TRUE)
}

cat("El archivo 'metaDatos.md' ha sido creado con éxito.\n")
```

```
El archivo 'metaDatos.md' ha sido creado con éxito.
```

Los archivos creados se copian a un directorio en el que se basará el repositorio

```
# Crear un nuevo directorio para el repositorio
repo_name <- "ExploreMetaboData"
repo_path <- file.path(getwd(), repo_name)
dir.create(repo_path, showWarnings = FALSE)

# Mover los archivos al nuevo directorio
file.copy(informe_destino, file.path(repo_path, informe_destino), overwrite = TRUE)
```

```
[1] TRUE
```

```
file.copy(codigo_destino, file.path(repo_path, codigo_destino), overwrite = TRUE)
```

```
[1] TRUE
```

```r
file.copy(objeto_binario, file.path(repo_path, objeto_binario), overwrite = TRUE)
```

```
[1] TRUE
```

```r
file.copy(metadatos_destino, file.path(repo_path, metadatos_destino), overwrite = TRUE)
```

```
[1] TRUE
```

Creación del Repositorio

Finalmente, desde R se puede crear y actualizar el repositorio con el paquete `git2r`

El código siguiente creará el repositorio y cargará los archivos. Como es plausible que falle debido a errores de comunicación se deja indicado pero no se ejecuta al crear el archivo de solucion.

```r
# Inicializar el repositorio Git

if (!require(git2r)) install.packages("git2r", dep = TRUE)
if (!require(usethis)) install.packages("usethis", dep = TRUE)

library(git2r)

git_repo <- git2r::init(repo_path)

# Crear un archivo README.md para el repositorio
readme_path <- file.path(repo_path, "README.md")
writeLines(c("# ExploreMetaboData", "Repositorio de análisis metabolómico"), readme_path)

# Agregar todos los archivos al repositorio
git2r::add(git_repo, "*")

# Realizar el primer commit
git2r::commit(git_repo, "Primer commit: Añadir archivos iniciales")
```

```r
# Configurar y crear el repositorio en GitHub con usethis

library(usethis)
usethis::use_git_config(user.name = "ASPteaching", user.email = "asanchez@ub.edu")
setwd(repo_path)

# Configurar GitHub usando usethis
```

```
usethis::use_git_config(user.name = "ASPteaching", user.email = "asanchez@ub.edu")

# Crear el repositorio en GitHub y publicarlo
usethis::use_github(private = FALSE)
```

Si necesitáramos modificar el repositorio y actualizarlo podríamos haverlo con las siguientes instrucciones

```
# Agregar todos los archivos modificados
git2r::add(git_repo, ".")

# Crear un commit con un mensaje descriptivo
git2r::commit(git_repo, "Actualización del repositorio")

# Subir los cambios
git2r::push(git_repo)
```

URL del repositorio

La url del repositorio creado es:

http://ASPteaching.github.com/ExploracionDatosMetabolomica