

ANÁLISIS DE DATOS ÓMICOS

PEC1

Ager Zenzano Sarasola - UOC

1 de noviembre de 2024

1 Introducción

En el presente documento recogeremos los pasos que iremos completando en el proceso que seguiremos para analizar datos ómicos. En el mismo recogeremos las técnicas y herramientas que iremos utilizando para la exploración multivariante de los datos.

Inicialmente seleccionaremos un conjunto de datos de metabolómica que obtendremos del siguiente repositorio de github: <https://github.com/nutrimetabolomics/metaboData/>. Utilizaremos el contenedor SummarizedExperiment como soporte de los datos, de manera que podremos acceder tanto a los datos como a los metadatos incluidos en el estudio.

2 Adquisición de los datos

2.1 Breve descripción del estudio ST000291

Los datos de metabolómica que estudiaremos serán los que se pueden encontrar en https://github.com/nutrimetabolomics/metaboData/2024-fobitoools-UseCase_1/. No obstante, obtendremos el estudio, ST000291 leyéndolo manualmente de la base de datos Metabolomics Workbench usando la función `do_query()` del paquete `metabolomicsWorkbenchR` de Bioconductor, tal y como lo hacemos a continuación:

```
SE_ST000291 = do_query(  
    context = 'study',  
    input_item = 'study_id',  
    input_value = 'ST000291',  
    output_item = 'SummarizedExperiment'  
)
```

Este estudio ST000291 tuvo como objetivo investigar los cambios metabólicos globales inducidos por el consumo de jugo de arándano o jugo de manzana utilizando un enfoque metabolómico global basado en LC-MS.

Según la documentación recogida ST000291 para el estudio se reclutaron dieciocho estudiantes universitarias sanas de entre 21 y 29 años con un IMC normal de 18,5 a 25. A cada una de ellas se le proporcionó una lista de alimentos que contenían cantidades significativas de procianidinas, como arándanos rojos, manzanas, uvas, arándanos azules, chocolate y ciruelas. Las participantes fueron asignadas aleatoriamente en dos grupos ($n = 9$) para consumir jugo de arándano o jugo de manzana. El estudio tuvo como objetivo investigar los cambios metabólicos generales provocados por los concentrados de procianidinas de arándanos rojos y manzanas utilizando un enfoque metabolómico global basado en LC-MS. Para mayor detalle de la descripción del estudio se puede consultar en ST000291.

La técnica Liquid Chromatography-Mass Spectrometry o también conocido por sus siglas en inglés como, LC-MS, combina la espectrometría de masas (MS) con el método de cromatografía líquida (Fuente Agarwal.V). De esta manera, se combina la capacidad que tiene la espectrometría de masas (MS) de separar moléculas orgánicas según su masa molecular permitiendo así su detección y cuantificación con una sensibilidad extremadamente alta, con la cromatografía líquida que facilita la separación rápida y cuantitativa de compuestos entre sí.

Vemos que el objeto del estudio ST000291 debido al tipo de técnica utilizada lo componen dos análisis; el positivo AN000464 y el negativo AN000464.

```
SE_ST000291
```

2.2 Descripción del contenedor SummarizedExperiment

SummarizedExperiment es un contenedor (Clase S4) para almacenar tanto los datos de análisis como los metadatos. Los datos de análisis son de tipo matriz donde las filas representan características de interés (por ejemplo, genes, transcripciones, exones, etc.) y las columnas representan muestras. Los datos de análisis y los metadatos se mantienen sincronizados mediante operaciones de subconjunto y reordenación. Existe disponible una serie de funciones para acceder, agregar y editar la información almacenada en los distintos componentes de los objetos SummarizedExperiment.

```
analisis_negativo <- do_query(  
    context = "study",  
    input_item = "analysis_id",  
    input_value = "AN000465",  
    output_item = "SummarizedExperiment")  
  
analisis_positivo <- do_query(  
    context = "study",  
    input_item = "analysis_id",
```

```
input_value = "AN000464",
output_item = "SummarizedExperiment")
```

Ambos análisis los contendremos en dos objetos o instancias `SummarizedExperiment` de clase S4. Además, podemos añadir que la clase `SummarizedExperiment` es una extensión de la clase `ExpressionSet` y muchas aplicaciones o bases de datos (como `metabolomicsWorkbench`) lo utilizan en vez de usar `ExpressionSet`.

Resumido de manera esquemática la estructura de `SummarizedExperiment` se estructuraría de la siguiente manera:

- `SummarizedExperiment`:
 - Samples (Columns): `colData(SE)`
 - Features (Rows): `rowData(SE)`
 - Features (Matriz de datos, Rows x Columns) : `assay(SE)`
 - Metadata: `metada(SE)`

La información sobre las características se almacena en un objeto `DataFrame`, anidado dentro del objeto `SummarizedExperiment` y accesible mediante la función `rowData()`. En nuestro caso vemos los metabolitos.

Table 1: Características: primeros 10 metabolitos del análisis, `rowData()`.

metabolite
10-Deacetyl-2-debenzoylbaccatin III
10-Oxodecanoate
1,1-Diethyl-2-hydroxy-2-nitrosohydrazine_1
1,1-Diethyl-2-hydroxy-2-nitrosohydrazine_2
1,2-Dihydroxynaphthalene-6-sulfonate
12-trans-Hydroxy juvenile hormone III
14-Dihydroxycornestin
1-(5-Phosphoribosyl)imidazole-4-acetate
1alpha,5alpha-Epidithio-17a-oxa-D-homoandrostan-3,17-dione_1
1alpha,5alpha-Epidithio-17a-oxa-D-homoandrostan-3,17-dione_2

La información sobre las muestras se almacena en otro objeto `DataFrame`, también anidado dentro del objeto `SummarizedExperiment`, y al que se puede acceder mediante la función `colData()`. En nuestro caso vemos las muestras y el tratamiento que sigue cada muestra.

Table 2: Muestras: primeras 10 del análisis, `colData()`.

Treatment	
b1	Baseline urine
b10	Baseline urine
b11	Baseline urine
b12	Baseline urine
b13	Baseline urine
b14	Baseline urine
b15	Baseline urine
b16	Baseline urine
b17	Baseline urine
b2	Baseline urine

Para acceder a los datos del experimento de un objeto `SummarizedExperiment`, se puede utilizar el de-

scriptor de acceso `assays()`. Además, hacemos uso de la funcionalidad de `subset` que ofrece el objeto `SummarizedExperiment` para mantener la muestra de datos reducida en caso de requerirlo.

```
analisis_negativo[1:5, 1:10]
```

```
class: SummarizedExperiment
dim: 5 10
metadata(4): data_source analysis_id name description
assays(1): ''
rownames(5): 1 2 3 4 5
rowData names(1): metabolite
colnames(10): b1 b10 ... b17 b2
colData names(1): Treatment
```

Table 3: Matriz de datos: primeras 10 características, primeras 10 muestras del análisis, `assay()`.

b1	b10	b11	b12	b13	b14	b15	b16	b17	b2
941000	757000	612000	858000	185000	671000	1140000	108000	383000	593000
785000	798000	12100000	2760000	697000	874000	1690000	148000	1060000	1130000
8300000	6790000	20800000	320000	1290000	1580000	2340000	1180000	1260000	15000000
1050000	606000	2980000	604000	742000	343000	751000	650000	494000	4540000
1500	890	16200000	1250	968	657	809	767	826	2810
276000	35700	631000	369000	242000	472000	5320000	18000	243000	131000
706000	121000	11600000	164000	424000	749000	267000	3050000	99100	136000
6340	34100	31900	9440	92600	6740	14400	8180	8980	4610
1990000	449000	1490000	1970000	1040000	822000	3950000	128000	1050000	551000
1630	0	0	0	907	1070	2190	0	0	0

Mediante la función `elementMetadata()` podemos acceder a los nombres de los metabolitos,

Table 4: Características: primeros 10 metabolitos del análisis, `elementMetadata()`.

metabolite
10-Deacetyl-2-debenzoylbaccatin III
10-Oxodecanoate
1,1-Diethyl-2-hydroxy-2-nitrosohydrazine_1
1,1-Diethyl-2-hydroxy-2-nitrosohydrazine_2
1,2-Dihydroxynaphthalene-6-sulfonate
12-trans-Hydroxy juvenile hormone III
14-Dihydroxycornestin
1-(5-Phosphoribosyl)imidazole-4-acetate
1alpha,5alpha-Epidithio-17a-oxa-D-homoandrostan-3,17-dione_1
1alpha,5alpha-Epidithio-17a-oxa-D-homoandrostan-3,17-dione_2

3 Pre-procesado de datos

En este apartado llevaremos a cabo una primera exploración de los datos, de tal manera que trataremos de proporcionar una visión general de los mismos. Además, nos basaremos en el flujo de trabajo de preprocessado realizado por Castellano-Escuder y también haremos uso del paquete Poma desarrollado por él mismo.

Inicialmente, obtenemos las características de modo negativo del estudio:

```
caracteristica_n <- SummarizedExperiment::assay(analisis_negativo) %>%
  dplyr::slice(-n())

rownames(caracteristica_n) <- SummarizedExperiment::rowData(analisis_negativo)$
  metabolite[1:(length(SummarizedExperiment::rowData(analisis_negativo)$metabolite)-1)]
```

y posteriormente las características de modo positivo del estudio:

```
caracteristica_p <- SummarizedExperiment::assay(analisis_positivo) %>%
  dplyr::slice(-n())

rownames(caracteristica_p) <- SummarizedExperiment::rowData(analisis_positivo)$
  metabolite[1:(length(SummarizedExperiment::rowData(analisis_positivo)$metabolite)-1)]
```

A continuación, mediante el paquete `rvest` de www.metabolomicsworkbench.org descargamos los nombres estandarizados de los metabolitos que estamos estudiando,

```
metabolitoURL <- paste0('https://www.metabolomicsworkbench.org/data/',
  'show_metabolites_by_study.php?',
  'STUDY_ID=ST000291&SEARCH_TYPE=',
  'KNOWN&STUDY_TYPE=MS&RESULT_TYPE=1')

metabolito <- metabolitoURL %>%
  read_html() %>%
  html_nodes(".datatable")

metabolito_n <- metabolito %>%
  .[[1]] %>%
  html_table() %>%
  dplyr::select(`Metabolite Name`, PubChemCompound_ID, `Kegg Id`)

metabolito_p <- metabolito %>%
  .[[2]] %>%
  html_table() %>%
  dplyr::select(`Metabolite Name`, PubChemCompound_ID, `Kegg Id`)

metabolitos <- bind_rows(metabolito_n, metabolito_p) %>%
  dplyr::rename(names = 1, PubChem = 2, KEGG = 3) %>%
  mutate(KEGG = ifelse(KEGG == "-", "UNKNOWN", KEGG),
        PubChem = ifelse(PubChem == "-", "UNKNOWN", PubChem)) %>%
  distinct(PubChem, .keep_all = TRUE)
```

Una vez hayamos recogido los nombres estandarizados de los metabolitos de la web, procedemos a asignarlos a los metabolitos que estamos estudiando sobre las `características`.

```
características <- bind_rows(caracteristica_n, caracteristica_p) %>%
  tibble::rownames_to_column("names") %>%
  left_join(metabolitos, by = "names") %>%
  select(-names, -KEGG) %>%
  drop_na(PubChem) %>%
```

```
tibble::column_to_rownames("PubChem")
```

Cabe destacar, que al realizar la acción `drop_na(PubChem)` estaremos eliminando aquellos metabolitos que no han sido asociados mediante la `left_join` con la tabla `metabolitos` (nombres estandarizados que hemos obtenido de www.metabolomicsworkbench.org). Es decir, estaremos eliminando aquellos metabolitos que no hemos conseguido identificar.

```
metadatos <- SummarizedExperiment::colData(analisis_negativo) %>%
  as.data.frame() %>%
  tibble::rownames_to_column("ID") %>%
  mutate(Treatment = case_when(
    Treatment == "Baseline urine" ~ "Baseline",
    Treatment == "Urine after drinking apple juice" ~ "Apple",
    Treatment == "Urine after drinking cranberry juice" ~ "Cranberry"))
```

3.1 Comprobación de los datos

A continuación, comprobamos los tipos de los datos,

```
caracteristicas_int <- caracteristicas
num = 0
nrows = nrow(caracteristicas)
for (i in 1:nrows) {
  if (class(caracteristicas[i,])=="numeric") {
    num = num + 1
  }
}
```

y vemos que de los 1359 metabolitos hay 0 de tipo numérico numéricos. Por ello, procedemos a modificar su tipo y los convertimos a numérico.

```
caracteristicas_int <- caracteristicas %>% mutate_if(is.character, as.numeric)
```

Después, analizamos si existen valores nulos e iguales a 0.

```
caracteristicas_int_t <- t(caracteristicas_int)

n_row <- rownames(caracteristicas_int_t)
n_col <- colnames(caracteristicas_int_t)
n_na <- sum(is.na(caracteristicas_int_t))

prc_zero_row <- round(rowSums(caracteristicas_int_t==0) /
  ncol(caracteristicas_int_t) * 100,2)
n_zero_row_all <- sum(prc_zero_row == 100)
prc_zero_col <- round(colSums(caracteristicas_int_t==0) /
  nrow(caracteristicas_int_t) * 100,2)
n_zero_col_all <- sum(prc_zero_col >= 100)
```

El número de filas con todas las columnas a 0 es 0.

El número de columnas con todas las filas a 0 es 0.

Después, evaluamos la varianza 0 de las características, y vemos que no hay ninguna característica con varianza 0.

```
var_zero <- data.frame(caracteristicas_int_t) %>%
  summarise_all(~ var(., na.rm = TRUE)) %>%
  t()
```

```
as.data.frame() %>%
  rownames_to_column("names") %>%
  filter(V1 == 0)
dim(var_zero)

[1] 0 2
var_zero

[1] names V1
<0 rows> (or 0-length row.names)
```

Una vez realizadas las primeras exploraciones, asignados los nombres estandarizados a los metabolitos y quedarnos con aquellos a los que hemos conseguido identificar un nombre estandarizado, utilizamos la función `PomaCreateObject` del paquete `Poma` de `Bioconductor` en R para crear el objeto de tipo `SummarizedExperiment`.

```
se_poma <- PomaCreateObject(metadata = metadatos, features = características_int_t)
```

Comprobamos la dimensión del objeto y vemos esta es: 1359, 45.

3.2 Inputación de datos faltantes

A partir del objeto de tipo `SummarizedExperiment`, el paquete `Poma` de Castellano-Escuder nos permite utilizar funcionalidades para el análisis de datos. En este caso haremos uso de la función `PomaImpute()` para imputar observaciones faltantes sobre el conjunto de datos.

En el caso que vemos a continuación, probamos parametrizar la función con método de imputación `knn`, y al final decidimos no convertir los 0 a nulos y después los eliminamos,

```
se_poma_imputado <- se_poma %>%
  PomaImpute(method = "knn", cutoff = 20) #, zeros_as_na = TRUE, remove_na = TRUE
se_poma_imputado
```

```
class: SummarizedExperiment
dim: 1359 45
metadata(0):
assays(1): ''
rownames(1359): X443489 X107754 ... X442878 X122087
rowData names(0):
colnames(45): b1 b10 ... c8 c9
colData names(1): treatment
```

vemos que no se han identificado datos faltantes en el conjunto de datos. No obstante, era lo esperado ya que no habíamos identificados valores nulos en el apartado anterior.

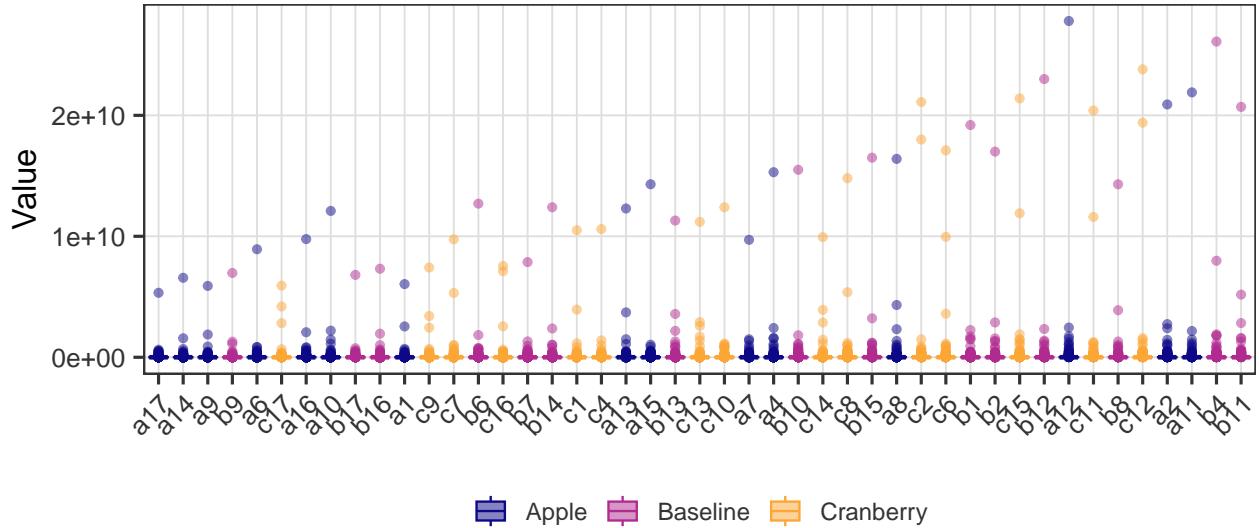


Figure 1: Diagrama de cajas de las muestras, una vez se han inputado observaciones con knn. Se diferencia el color por tipo de tratamiento, PomaImpute().

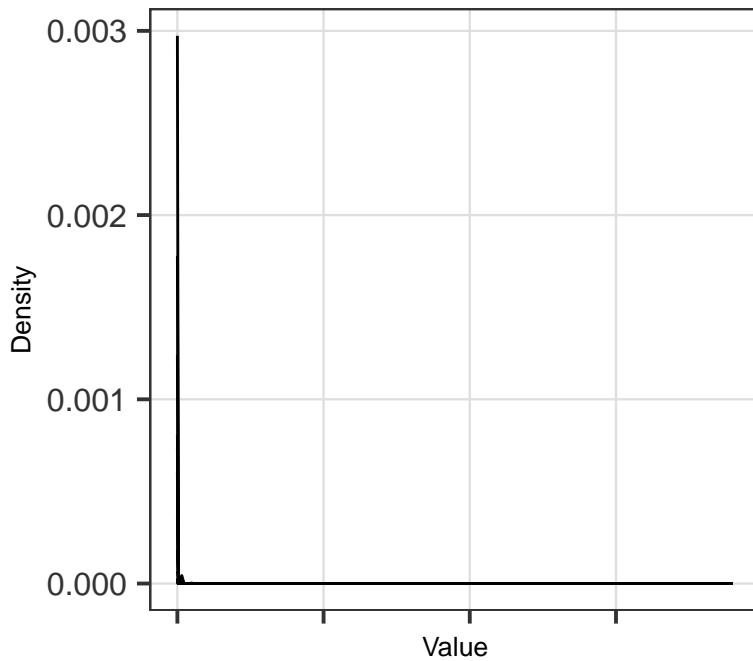


Figure 2: Diagrama de densidad de las muestras, una vez se han inputado observaciones con knn, PomaImpute().

3.3 Normalización de los datos

Probamos diferentes métodos de normalización que se pueden realizar mediante el paquete Poma,

```
se_poma_imputado_sin <- PomaNorm(se_poma_imputado, method = "none")
se_poma_imputado_auto_scaling <- PomaNorm(se_poma_imputado, method = "auto_scaling")
se_poma_imputado_level_scaling <- PomaNorm(se_poma_imputado, method = "level_scaling")
se_poma_imputado_log_scaling <- PomaNorm(se_poma_imputado, method = "log_scaling")
se_poma_imputado_vast_scaling <- PomaNorm(se_poma_imputado, method = "vast_scaling")
se_poma_imputado_log_pareto <- PomaNorm(se_poma_imputado, method = "log_pareto")
```

y comprobamos si después de normalizar el conjunto de datos sufre alguna reducción de la dimensión. Cabe destacar que PomaNorm() solo modifica la dimensión de los datos cuando el conjunto de datos contiene solo características cero o características de varianza cero.

```
dim(SummarizedExperiment::assay(se_poma_imputado_sin))

[1] 1359    45

dim(SummarizedExperiment::assay(se_poma_imputado_auto_scaling))

[1] 1359    45

dim(SummarizedExperiment::assay(se_poma_imputado_level_scaling))

[1] 1359    45

dim(SummarizedExperiment::assay(se_poma_imputado_log_scaling))

[1] 1359    45

dim(SummarizedExperiment::assay(se_poma_imputado_vast_scaling))

[1] 1359    45

dim(SummarizedExperiment::assay(se_poma_imputado_log_pareto))

[1] 1359    45
```

Analizamos la influencia de la normalización sobre las muestras,

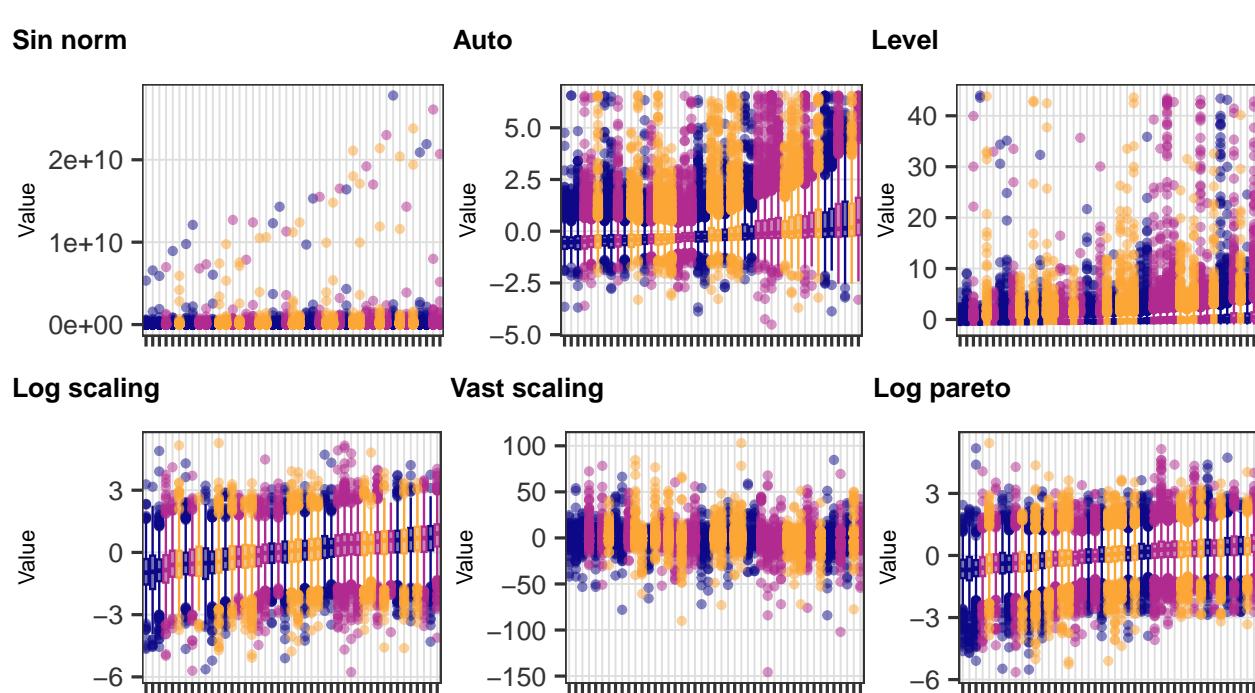


Figure 3: Diagrama de cajas de las muestras, normalizadas con diferentes métodos: *none*, *auto_scaling*, *level_scaling*, *log_scaling*, *log_pareto*, *vast_scaling*. Se diferencia el color por tipo de tratamiento, *PomaNorm()*.

Analizamos la influencia de la normalización sobre las características,

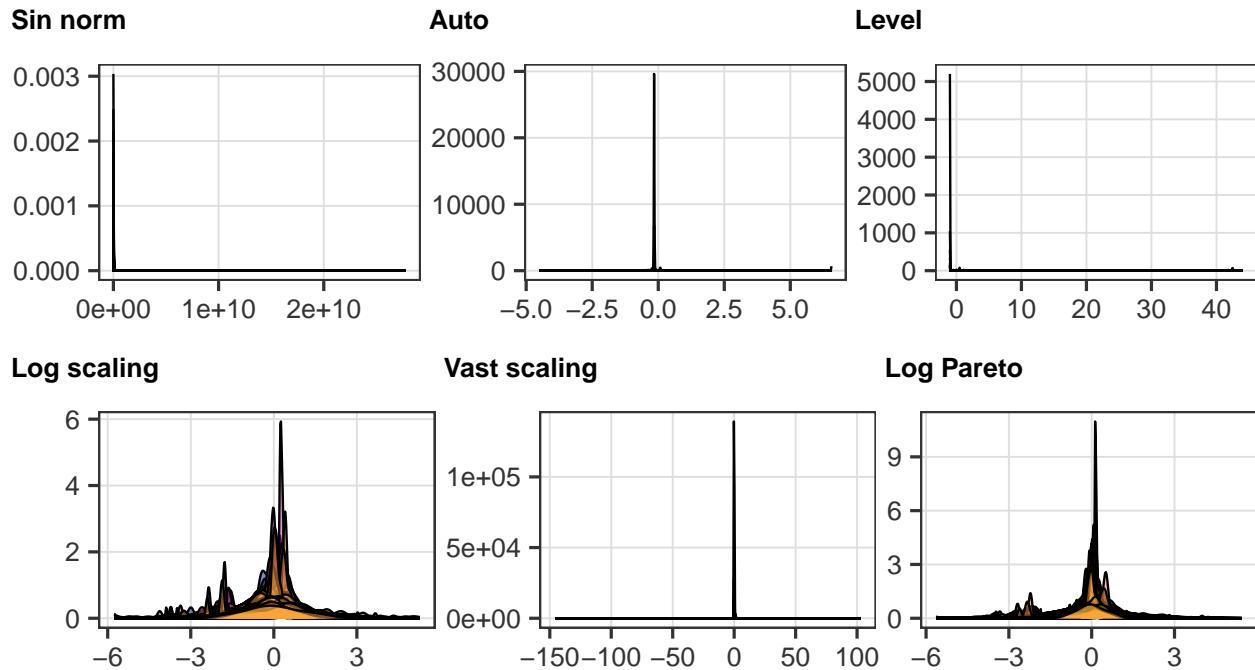


Figure 4: Diagrama de densidad de las muestras, normalizadas con diferentes métodos: none, auto_scaling, level_scaling, log_scaling, log_pareto, vast_scaling. Se diferencia el color por tipo de tratamiento, PomaNorm().

A continuación, vemos con el gráfico de densidad PomaDensity() las dos opciones más destacables: Log scaling y Log pareto.

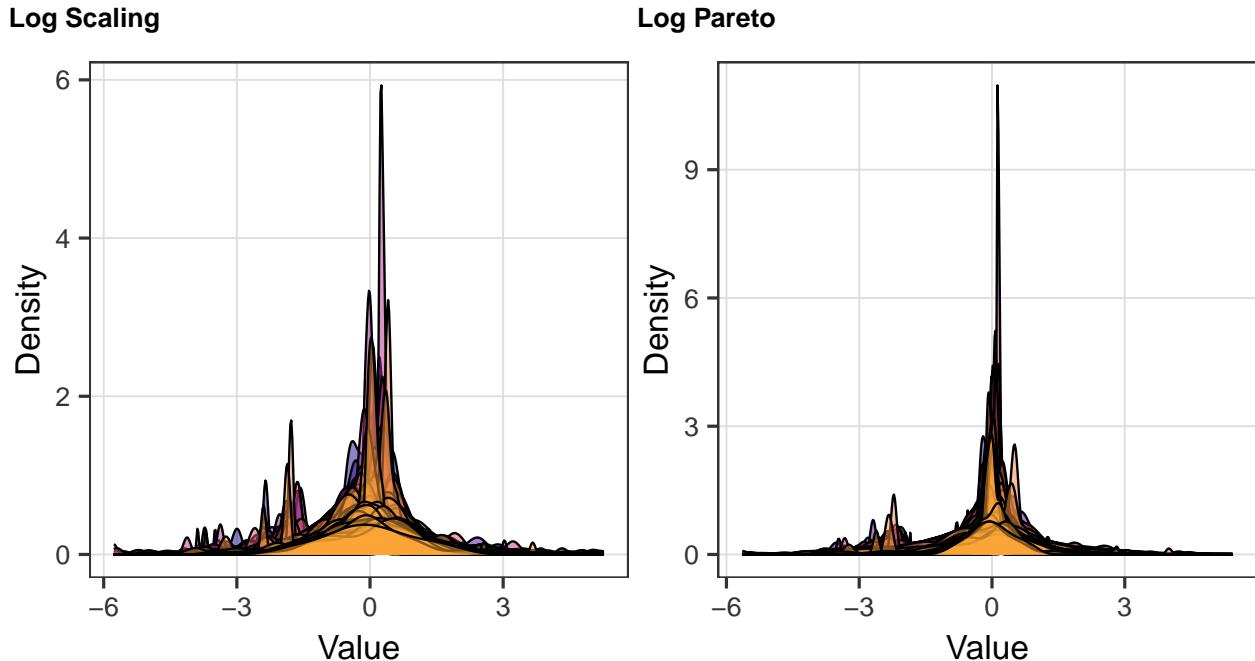


Figure 5: Diagrama de densidad de las muestras, una vez se han inputado observaciones con knn para Log Scaling y Log Pareto. Se diferencia el color por tipo de tratamiento, PomaImpute().

3.4 Pre-procesado sin datos atípicos

Después de normalizar el conjunto de datos utilizamos la función `PomaOutliers()` del paquete `Poma` para pre-procesar los datos eliminando los valores atípicos en caso de que los haya. Utilizamos los parámetros:

- `method`: indica el método de medición de distancia para realizar MDS (Multidimensional Scaling).
- `type`: indica el tipo de análisis de valores atípicos a realizar.
- `coef`: indica el coeficiente de valores atípicos. Cuanto más bajo es mayor será la sensibilidad ante los valores atípicos.

Primero, realizamos una comparativa con los datos sin normalizar,

```
se_poma_preprocesado <- se_poma_imputado_sin %>%
  PomaOutliers(method = "euclidean",
                type = "median",
                coef = 2)
```

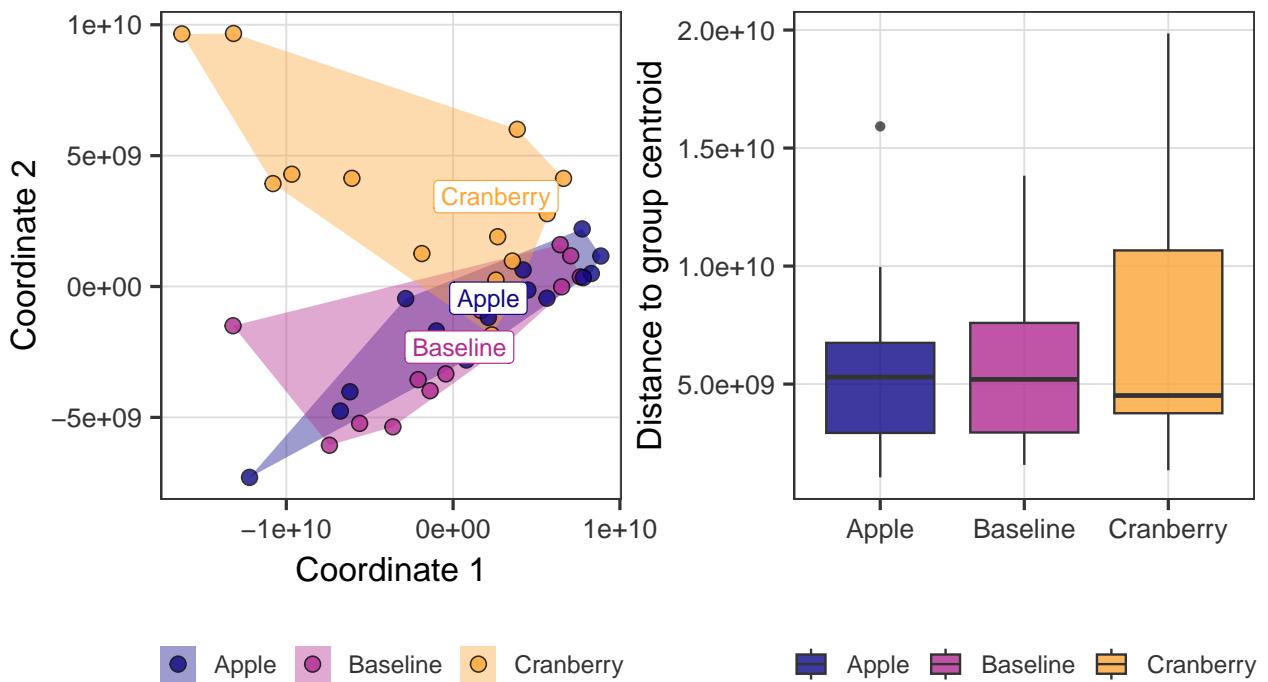


Figure 6: Diagrama poligonal de las de las coordenadas 1 y dos para cada tratamiento: Cranberry, Apple, Baseline, y diagrama de cajas con las distancias al centroide del grupo para cada tratamiento: Cranberry, Apple, Baseline, `PomaOutliers()`.

Table 5: Datos atípicos identificados mediante la función `PomaOutliers()` en el atributo `outliers`.

sample	groups	distance_to_centroid	limit_distance
a12	Apple	15917686960	14376816373

y después con los datos normalizados,

```
se_poma_preprocesado <- se_poma_imputado_log_scaling %>%
  PomaOutliers(method = "euclidean",
                type = "median",
                coef = 2)
```

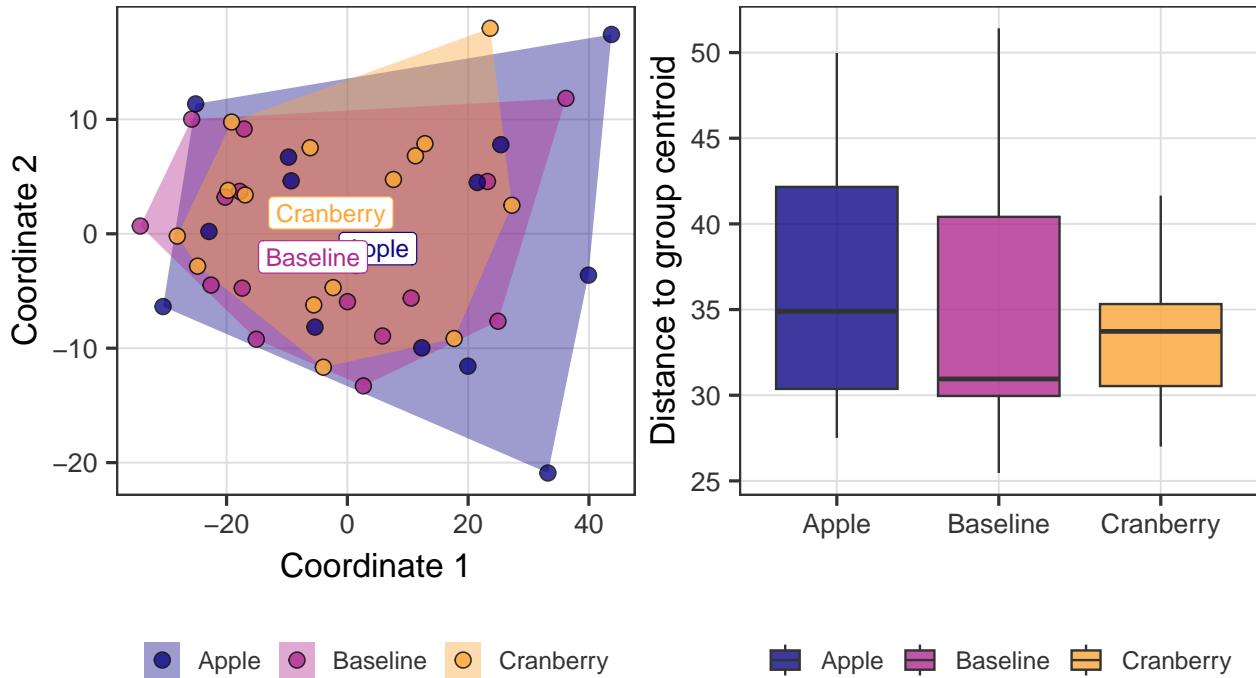


Figure 7: Diagrama poligonal de las de las coordenadas 1 y dos para cada tratamiento: Cranberry, Apple, Baseline, y diagrama de cajas con las distancias al centroide del grupo para cada tratamiento: Cranberry, Apple, Baseline, una vez escalados los datos, `PomaOutliers()`.

Table 6: Datos atípicos identificados mediante la función `PomaOutliers()` en el atributo `outliers`.

sample	groups	distance_to_centroid	limit_distance
--------	--------	----------------------	----------------

Vemos que en el conjunto de datos sin normalizar existen valores atípicos, mientras que en el normalizado no.

4 Análisis estadísticos

4.1 Análisis PCA

El análisis de los componentes principales se utiliza para resumir y obtener la información contenida en un conjuntos de datos multivariantes. Este método de proyección o de reducción de la dimensión es de uso habitual en la exploración de datos ómicos (fuente Bakker.J).

El análisis de componentes principales (PCA) lleva a cabo una transformación mediante la combinación de las variables originales correlacionadas entre sí, a componentes principales no correlacionados. Cada uno de los componentes explica una parte de la variabilidad, y se construyen de manera ordenada, siendo la primera la que mayor variabilidad explica, después la segunda y así sucesivamente.

El análisis de componentes principales (PCA) es un enfoque basado en el cálculo de los valores propios. Este método identifica un conjunto de ecuaciones lineales que resumen una matriz cuadrada simétrica. A partir de este método, se produce un conjunto de valores propios λ , y cada uno de estos tiene un vector propio asociado x . La conexión citada se puede describir mediante la siguiente ecuación:

$$Ax = \lambda x$$

siendo los valores propios λ los que representan la varianza extraída de cada eje, y x los vectores propios asociados con la misma posición relativa

Inicialmente utilizaremos la función `PomaPCA()` del paquete `Poma` para calcular los componentes principales, `se_poma_imputado_log_scaling$pca <- PomaPCA(se_poma_imputado_log_scaling)`

mediante el argumento `factors` con el operador `$` obtendremos los vectores propios contenidos el objeto calculado,

Table 7: Primeros 10 muestras sobre los 4 primeros componentes, `factors` de `PomaPCA()`.

sample_id	group	PC1	PC2	PC3	PC4
b1	Baseline	-17.818	3.706	-12.613	1.435
b10	Baseline	-15.080	-9.216	3.179	-6.052
b11	Baseline	-34.288	0.680	-5.520	-11.126
b12	Baseline	-22.569	-4.481	3.833	5.588
b13	Baseline	0.020	-5.933	-1.064	-5.041
b14	Baseline	2.653	-13.294	0.469	0.905
b15	Baseline	-17.408	-4.751	-4.244	-5.039
b16	Baseline	23.203	4.571	9.097	-11.140
b17	Baseline	24.945	-7.642	-12.737	5.935
b2	Baseline	-20.242	3.201	-5.095	-16.456

mediante el argumento `loadings` con el operador `$` obtendremos los vectores propios contenidos el objeto calculado,

Table 8: Vectores propios, primeras 10 de las características sobre los 4 primeros componentes, ‘loadings’ de PomaPCA().

feature	PC1	PC2	PC3	PC4
X443489	-0.033	-0.032	-0.013	0.042
X107754	-0.017	-0.003	-0.020	-0.051
X9543071	-0.003	0.008	-0.019	-0.024
X11011465	-0.029	-0.044	-0.016	0.004
X5281160	-0.024	-0.005	-0.023	0.006
X440341	-0.005	-0.033	-0.012	-0.039
X5281760	-0.027	0.010	-0.033	0.037
X124021	-0.018	0.008	0.018	0.015
X5280406	-0.017	0.023	-0.013	-0.022
X14132344	-0.018	0.046	-0.012	-0.008

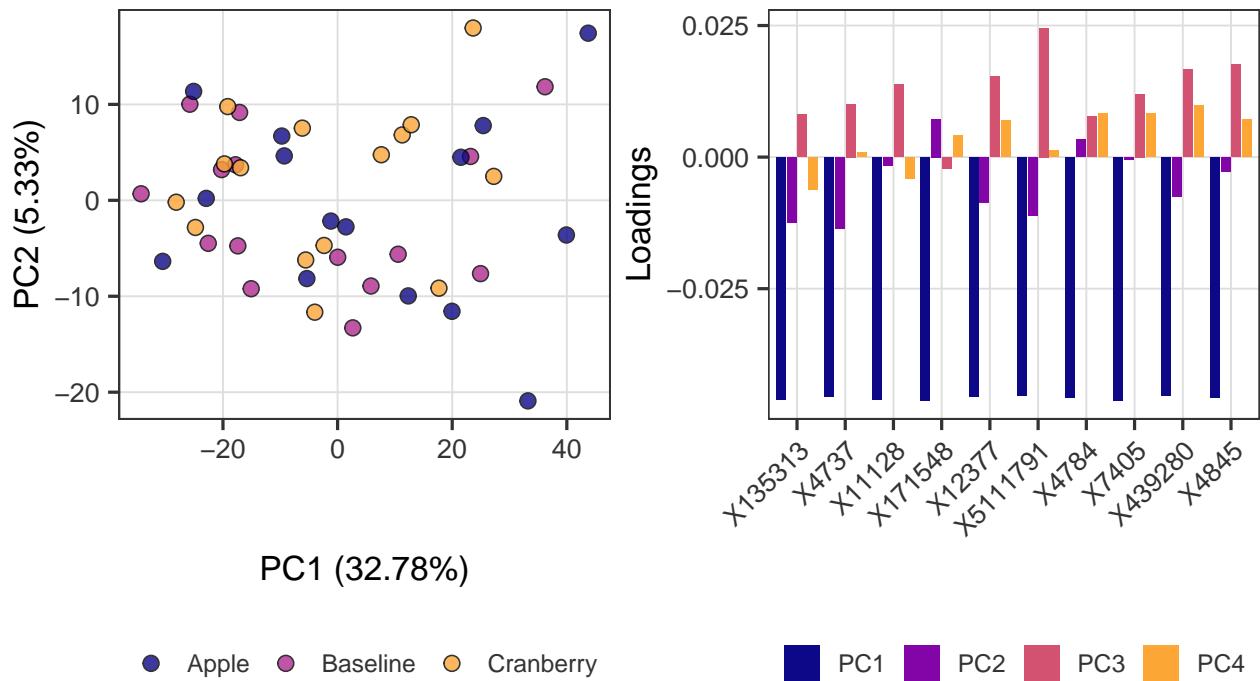


Figure 8: Diagrama de puntos de los dos primeros PCs, y Loadings para las características por PC.

4.2 Análisis PCA manual

A continuación, realizaremos el cálculo del PCA de manera manual siguiendo los pasos de Sanchez.A:

- normalizaremos los datos,
- calcularemos la matriz de varianza/covarianza,
- obtendremos los valores y vectores propios y los interpretaremos.

Normalizar datos

Si las variables no están en la misma escala, no contribuyen de la misma manera a los resultados (fuente Bakker.J), por ello procedemos a usar la función `scale()` y normalizar los datos; restando la media y dividiendo por la desviación estándar.

```
características_int_t_scaled <- scale(características_int_t)
```

Calcular matriz de Varianza/Covarianza

Diagonalizando la matriz de varianzas-covarianzas de los datos centrados obtendremos: un vector de valores propios proporcionales a la varianza de cada variable en las nuevas coordenadas, y una matriz de vectores propios que son las puntuaciones de las observaciones en estas coordenadas (fuente Sanchez.A). Al diagonalizar la matriz de varianzas conseguiremos que las covarianzas de las nuevas variables valgan 0, es decir, que sean independientes (fuente Sanchez.A).

```
VCM <- cov(características_int_t_scaled)
```

Calcular valores propios

```
VCM_eigen <- eigen(VCM)
```

Interpretar valores propios

Habrá tantos valores propios como variables haya, no obstante valores > 0 , habrá 44

```
[1] 337.240 96.604 77.755 69.978 59.124 53.205 45.890 42.384 40.070  
[10] 36.472 30.493 28.469 27.748 26.849 25.686 23.585 22.621 21.878  
[19] 21.028 20.573 18.485 18.159 16.920 16.289 15.634 14.594 13.425  
[28] 12.315 11.603 11.289 10.698 10.285 9.486 9.355 8.307 8.062  
[37] 7.720 6.917 6.827 6.083 5.306 4.981 4.580 4.026
```

Comprobamos que la suma de los valores propios es 1359, la misma que el número características, 1359.

Los valores propios representan la varianza de las componentes, por lo que cada valor dividido por la suma de éstos es el porcentaje de variabilidad explicado (fuente Sanchez.A).

```
VCM_eigen_prop <- round(VCM_eigen$values / sum(VCM_eigen$values),3)  
VCM_eigen_prop <- VCM_eigen_prop[VCM_eigen_prop > 0]  
VCM_eigen_prop
```

```
[1] 0.248 0.071 0.057 0.051 0.044 0.039 0.034 0.031 0.029 0.027 0.022 0.021  
[13] 0.020 0.020 0.019 0.017 0.017 0.016 0.015 0.015 0.014 0.013 0.012 0.012  
[25] 0.012 0.011 0.010 0.009 0.009 0.008 0.008 0.008 0.007 0.007 0.006 0.006  
[37] 0.006 0.005 0.005 0.004 0.004 0.004 0.003 0.003
```

En total suman el 1

Interpretar vectores propios

Los elementos de un vector propio se conocen como cargas (loadings) y éstas son los coeficientes de la ecuación lineal que combinan las variables para ese componente principal (fuente Bakker.J). La transformación de los datos asociada a las componentes principales se obtiene multiplicando la matriz original por la matriz de vectores propios (fuente Sanchez.A, Bakker.J).

```
pcas <- round(características_int_t_scaled %*% VCM_eigen$vectors,3)
```

Table 9: Primeras 10 de las primeras muestras sobre los 4 primeros componentes, factors.

	PC1	PC2	PC3	PC4
b1	-14.104	2.390	10.100	-2.138
b10	-9.720	-0.401	-4.734	1.692
b11	-42.513	28.379	-6.343	-26.186
b12	-21.228	-8.978	-9.645	-1.004
b13	7.058	2.064	-2.683	0.523
b14	7.502	-1.830	-4.821	0.732
b15	-11.466	5.416	-6.639	-1.839
b16	18.785	0.484	-1.161	2.473
b17	20.043	-0.858	-0.823	-3.596
b2	-15.009	20.239	11.216	19.502

A continuación mostramos los 4 primeros vectores propios del atributo `vectors`, que son las coordenadas de las componentes principales. Por motivos de no sobre extender demasiado su visualización mostramos solo los 10 primeros valores.

Table 10: Vectores propios, primeras 10 de las características primeras sobre los 4 primeros componentes, vectors.

	PC1	PC2	PC3	PC4
443489	-0.026	-0.054	0.039	0.000
107754	-0.025	0.046	0.002	-0.014
9543071	-0.020	0.044	-0.013	-0.058
11011465	-0.022	-0.004	-0.031	-0.037
5281160	-0.016	0.019	0.002	-0.072
440341	0.001	0.006	-0.006	-0.007
5281760	-0.043	0.014	0.007	-0.011
124021	-0.007	0.001	-0.007	-0.005
5280406	-0.003	0.014	0.002	0.002
14132344	0.000	0.009	0.015	0.000

Vemos que los resultados son ligeramente diferentes respecto a los obtenidos con el paquete `Poma` porque el tratamiento que se ha realizado sobre el conjunto de datos ha sido ligeramente diferente.

Table 11: Vectores propios, primeras 10 características primeras sobre los 4 primeros componentes, loadings de PomaPCA().

	PC1	PC2	PC3	PC4
X443489	-0.026	0.054	0.039	0.000
X107754	-0.025	-0.046	0.002	-0.014
X9543071	-0.020	-0.044	-0.013	-0.058
X11011465	-0.022	0.004	-0.031	-0.037
X5281160	-0.016	-0.019	0.002	-0.072
X440341	0.001	-0.006	-0.006	-0.007
X5281760	-0.043	-0.014	0.007	-0.011
X124021	-0.007	-0.001	-0.007	-0.005
X5280406	-0.003	-0.014	0.002	0.002
X14132344	0.000	-0.009	0.015	0.000

Calculo PCA mediante prcomp

También podemos calcular los componentes principales mediante la función `prcomp`. La función `princomp` calcula las componentes principales mediante la descomposición en valores singulares de la matriz de datos (fuente Sanchez.A).

```
caracteristicas_t_PCA_prcomp <- prcomp(caracteristicas_int_t,
                                         scale. = TRUE, center = TRUE) # rank. = 4
```

El argumento `sdev` contiene las desviaciones estándar, es decir las raíces cuadradas de los valores propios (fuente Sanchez.A).

```
round(caracteristicas_t_PCA_prcomp$sdev,3)
```

```
[1] 18.364 9.829 8.818 8.365 7.689 7.294 6.774 6.510 6.330 6.039
[11] 5.522 5.336 5.268 5.182 5.068 4.856 4.756 4.677 4.586 4.536
[21] 4.299 4.261 4.113 4.036 3.954 3.820 3.664 3.509 3.406 3.360
[31] 3.271 3.207 3.080 3.059 2.882 2.839 2.778 2.630 2.613 2.466
[41] 2.304 2.232 2.140 2.007 0.000
```

y podemos compararlos con los obtenidos manualmente:

```
round(sqrt(VCM_eigen_values),3)
```

```
[1] 18.364 9.829 8.818 8.365 7.689 7.294 6.774 6.510 6.330 6.039
[11] 5.522 5.336 5.268 5.182 5.068 4.856 4.756 4.677 4.586 4.536
[21] 4.299 4.261 4.113 4.036 3.954 3.820 3.664 3.509 3.406 3.360
[31] 3.271 3.207 3.080 3.059 2.882 2.839 2.778 2.630 2.613 2.466
[41] 2.304 2.232 2.140 2.007
```

En el objeto que obtenemos a partir de la función `prcomp` los vectores los encontramos en el argumento `rotation`,

Table 12: Vectores propios, primeras 10 de las características primarios sobre los 4 primeros componentes, ‘rotation’ de `prcomp()`.

	PC1	PC2	PC3	PC4
443489	-0.026	0.054	0.039	0.000
107754	-0.025	-0.046	0.002	-0.014
9543071	-0.020	-0.044	-0.013	-0.058
11011465	-0.022	0.004	-0.031	-0.037
5281160	-0.016	-0.019	0.002	-0.072
440341	0.001	-0.006	-0.006	-0.007
5281760	-0.043	-0.014	0.007	-0.011
124021	-0.007	-0.001	-0.007	-0.005
5280406	-0.003	-0.014	0.002	0.002
14132344	0.000	-0.009	0.015	0.000

Finalmente el argumento `x` contiene las componentes principales calculadas por `prcomp`.

Table 13: Primeras 10 muestras sobre los 4 primeros componentes, `x`.

	PC1	PC2	PC3	PC4
b1	-14.104	-2.390	10.100	-2.138
b10	-9.720	0.401	-4.734	1.692
b11	-42.513	-28.379	-6.343	-26.186
b12	-21.228	8.978	-9.645	-1.004
b13	7.058	-2.064	-2.683	0.523
b14	7.502	1.830	-4.821	0.732
b15	-11.466	-5.416	-6.639	-1.839
b16	18.785	-0.484	-1.161	2.473
b17	20.043	0.858	-0.823	-3.596
b2	-15.009	-20.239	11.216	19.502

Vemos el porcentaje de varianza explicada por cada uno de los primeros 4 componente calculados con la función `prcomp`.

Table 14: Importancia para cada uno de los primeros 4 componente, `prcomp()`.

	PC1	PC2	PC3	PC4
Standard deviation	18.364	9.829	8.818	8.365
Proportion of Variance	0.248	0.071	0.057	0.051
Cumulative Proportion	0.248	0.319	0.376	0.428

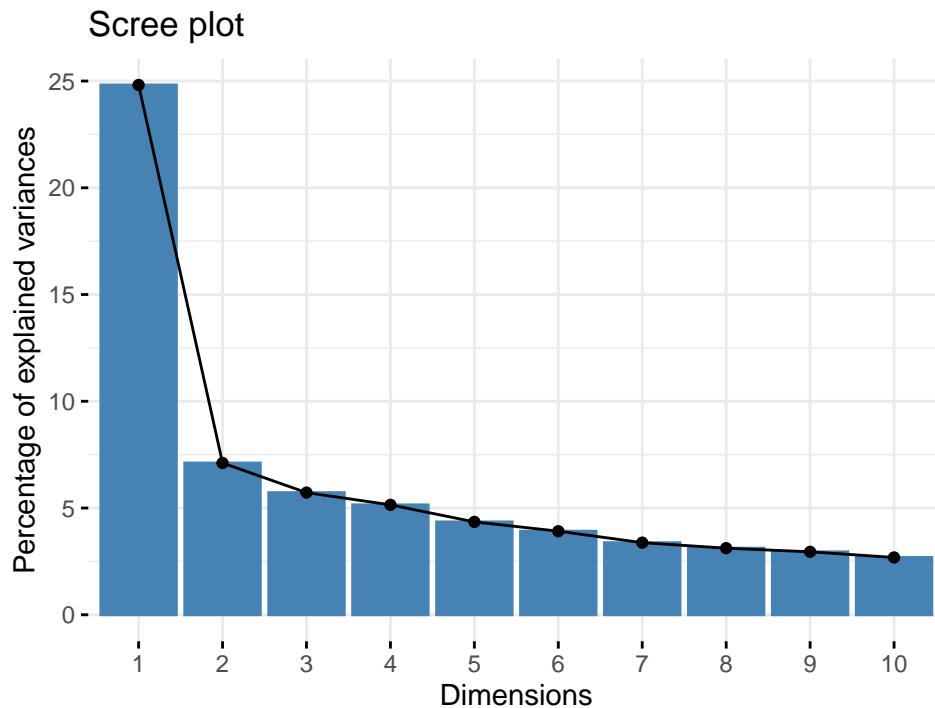


Figure 9: Digráma de barras del porcentaje de varianza explicada por cada dimensión PC, `prcomp()`.

En el gráfico anterior, vemos que con el primer componente principal se explica aproximadamente un 25% de la varianza total por lo que no parece un valor muy elevado. Además, cabe destacar que de este primer componente al segundo hay un salto considerable.

A continuación visualizamos las muestras de forma sencilla el PCA de los datos en los dos primeros componentes,

Gráfica de puntos de 2 PCs

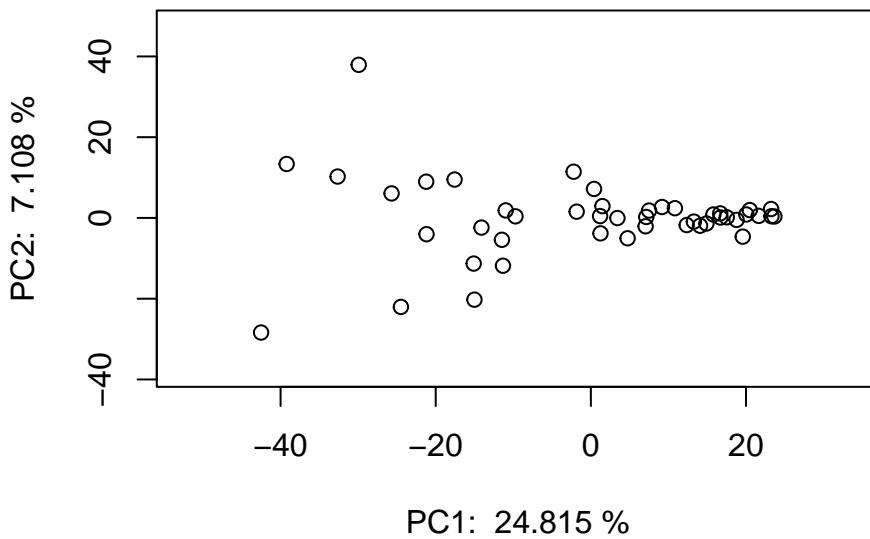


Figure 10: Digráma de puntos de los dos primeros PCs.

Ha simple vista y con los resultados obtenidos resulta complicado debido sobre todo a la gran dimensión del conjunto de datos y sin ningún conocimiento de los metabolitos poder buscar un explicación razonable a cada componente como agrupación abstracta de las diferentes características.

4.3 Test de Limma

Utilizamos el modelo Limma (fuente Ritchie) mediante la función `PomaLimma()` del paquete Poma para identificar los metabolitos más significativos entre los grupos `Base` y `Cranberry`.

```
limma_res_bc <- se_poma_imputado_log_scaling %>%
  PomaLimma(contrast = "Baseline-Cranberry", adjust = "fdr") %>%
  dplyr::rename(PubChemCID = feature) %>%
  dplyr::mutate(PubChemCID = gsub("X", "", PubChemCID))
```

y mostramos las primeras 10 características.

Table 15: Primeros 10 de los metabolitos más significativos según el test de Limma entre los grupos: `Baseline-Cranberry`, `PomaLimma()`.

PubChemCID	logFC	AveExpr	t	pvalue	adj_pvalue	B
94214	-1.570141	0	-4.322026	1.55e-05	0.0092734	2.770742
71485	-1.547094	0	-4.258585	2.06e-05	0.0092734	2.516305
54678503	-1.541948	0	-4.244421	2.20e-05	0.0092734	2.460010
5378303	-1.524096	0	-4.195279	2.73e-05	0.0092734	2.266158
5486800	1.488068	0	4.096107	4.21e-05	0.0095639	1.881821
10178	-1.487766	0	-4.095276	4.22e-05	0.0095639	1.878641
443071	-1.459491	0	-4.017445	5.89e-05	0.0107191	1.583510
3037	-1.449762	0	-3.990666	6.60e-05	0.0107191	1.483275
3035199	-1.443433	0	-3.973244	7.10e-05	0.0107191	1.418422
17531	-1.418133	0	-3.903602	9.49e-05	0.0118971	1.162024

y a continuación, identificaremos lo metabolitos más significativos entre los grupos `Base` y `Apple`,

```
limma_res_ba <- se_poma_imputado_log_scaling %>%
  PomaLimma(contrast = "Baseline-Apple", adjust = "fdr") %>%
  dplyr::rename(PubChemCID = feature) %>%
  dplyr::mutate(PubChemCID = gsub("X", "", PubChemCID))
```

y mostramos las primeras 10 características

Table 16: Primeros 10 de los metabolitos más significativos según el test de Lima entre los grupos: Baseline-Apple, PomaLimma().

PubChemCID	logFC	AveExpr	t	pvalue	adj_pvalue	B
192798	-1.7817001	0	-4.904370	0.0000009	0.0012773	4.8955154
79034	1.4835162	0	4.083579	0.0000444	0.0301750	1.6646462
20975673	1.2954337	0	3.565856	0.0003630	0.1552838	-0.0697671
10413	-1.2733049	0	-3.504944	0.0004571	0.1552838	-0.2583923
439230	-1.2259620	0	-3.374626	0.0007397	0.1983154	-0.6510291
8117	1.2089893	0	3.327907	0.0008756	0.1983154	-0.7881693
5275508	1.1524449	0	3.172261	0.0015134	0.2938107	-1.2312562
7501	1.0124870	0	2.787008	0.0053215	0.6830501	-2.2367165
441445	0.9921072	0	2.730909	0.0063179	0.6830501	-2.3722820
7768	-0.9717887	0	-2.674980	0.0074755	0.6830501	-2.5046965

4.4 Análisis de Correspondencias

El análisis de correspondencias es una técnica de ordenación; una manera de representar las variables en un espacio de dimensión menor, que resulta especialmente adecuada para analizar y visualizar datos de tablas de contingencia con datos de frecuencias numéricas (fuente Sanchez.A). El análisis de la tabla de frecuencias para el análisis de correspondencias puede plantearse de dos formas: como un escalado multidimensional de filas y de columnas con la distancia ji-cuadrado, o como un análisis de componentes principales sobre la matriz Z estandarizada (fuente Sanchez.A). El Análisis de Correspondencias es una manera de representar las variables en un espacio de dimensión menor.

Para realizar las pruebas con el análisis de correspondencias, partiremos de los resultados obtenidos en el test de Lima del apartado anterior. De esta manera limitaremos el conjunto de metabolitos a analizar (seleccionamos los 8 que hemos identificado como los más significativos) .

Iniciaremos el análisis preparando la estructura de datos necesaria:

```
lima_bc_scope <- limma_res_bc[1:8,1]
lima_bc_scope$scope <- 1
lima_bc_scope_meta <- metadatos %>%
  filter(Treatment != "Apple")

sample_caracteristicas_int_t <- caracteristicas_int %>%
  tibble::rownames_to_column("PubChemID") %>%
  left_join(lima_bc_scope, by = "PubChemID") %>%
  filter(scope == 1) %>%
  select(-scope) %>%
  tibble::column_to_rownames("PubChemID") %>%
  t %>%
  data.frame() %>%
  tibble::rownames_to_column("ID") %>%
  right_join(lima_bc_scope_meta, by = "ID") %>%
  select(-Treatment) %>%
  tibble::column_to_rownames("ID")

sample_caracteristicas_int <- data.frame(t(sample_caracteristicas_int_t))
```

Antes de realizar el análisis de correspondencias con la función CA(), observamos un gráfico sencillo balloonplot() donde cada celda contiene un punto cuyo tamaño refleja la magnitud relativa del componente correspondiente (fuente Sanchez.A).

Características: metabolitos

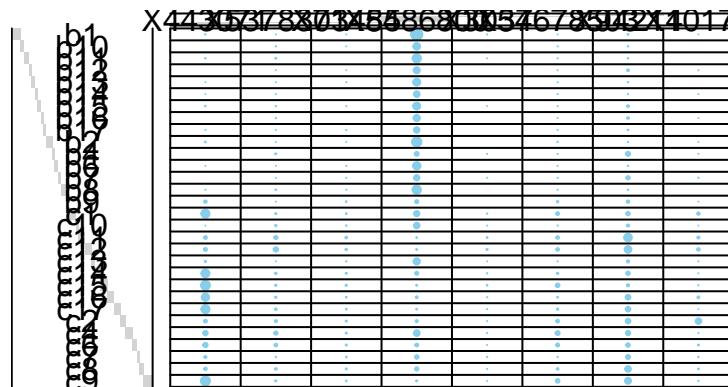


Figure 11: Diagrama de balones para los metabolitos seleccionados.

Utilizamos la función CA() del paquete FactoMineR para realizar el análisis de correspondencias,

```
datos.ca <- CA(sample_caracteristicas_int_t, graph = FALSE)
```

y después visualizamos con la función fviz_ca_biplot las dos primeras dimensiones:

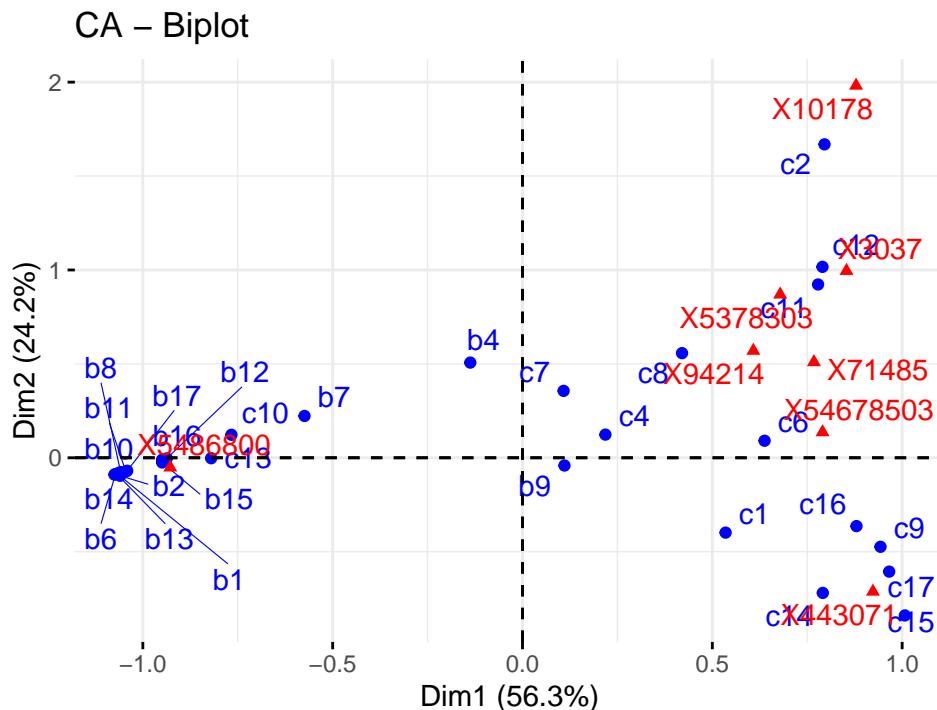


Figure 12: Diagrama de mapa de puntos en las dos primeras dimensiones del análisis de correspondencias, CA().

El gráfico anterior se denomina gráfico simétrico y muestra un patrón global dentro de los datos. Las filas (muestras) se representan con puntos azules y las columnas (metabolitos) con triángulos rojos. La distancia entre los puntos de las filas o columnas proporciona una medida de su similitud o disimilitud. Los puntos

de las filas con un perfil similar están cerca en el mapa de factores. Lo mismo se aplica a los puntos de las columnas.

A continuación, se muestra el diagrama que muestra el porcentaje de varianza explicada por cada dimensión, donde la linea discontinua roja especifica la media de los valores propios.

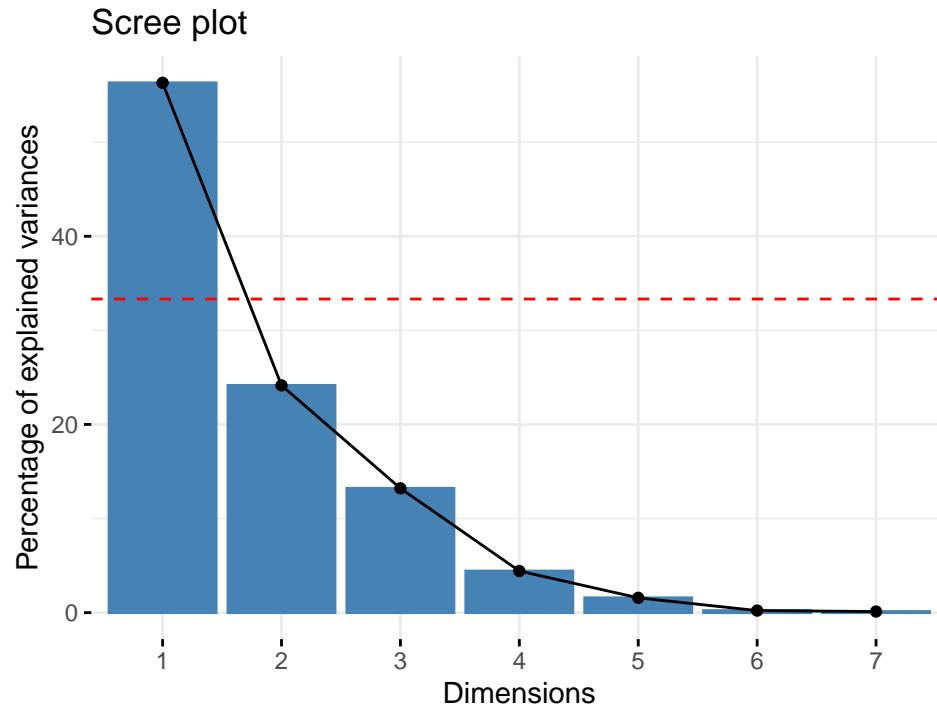


Figure 13: Diagrama de barras del porcentaje de varianza explicada por cada dimensión AC, CA().

Según el gráfico anterior, la dimensión 1 explica el 56,3 % de la inercia total y la 2 explica aproximadamente el 24,2 % de la inercia, por lo que entre las dos llegarían a un 80,5 % de la inercia total. La dimensión 3 explica solo el 12 % de la inercia total, que es inferior al valor propio promedio (33 %) y demasiado poco como para conservarlo para un análisis posterior.

Podemos utilizar la función `get_ca_row()` del paquete `factoextra` para extraer los resultados de las variables de fila (muestras). Esta función devuelve una lista que contiene las coordenadas, el `cos2`, que la contribución y la inercia de las variables de fila:

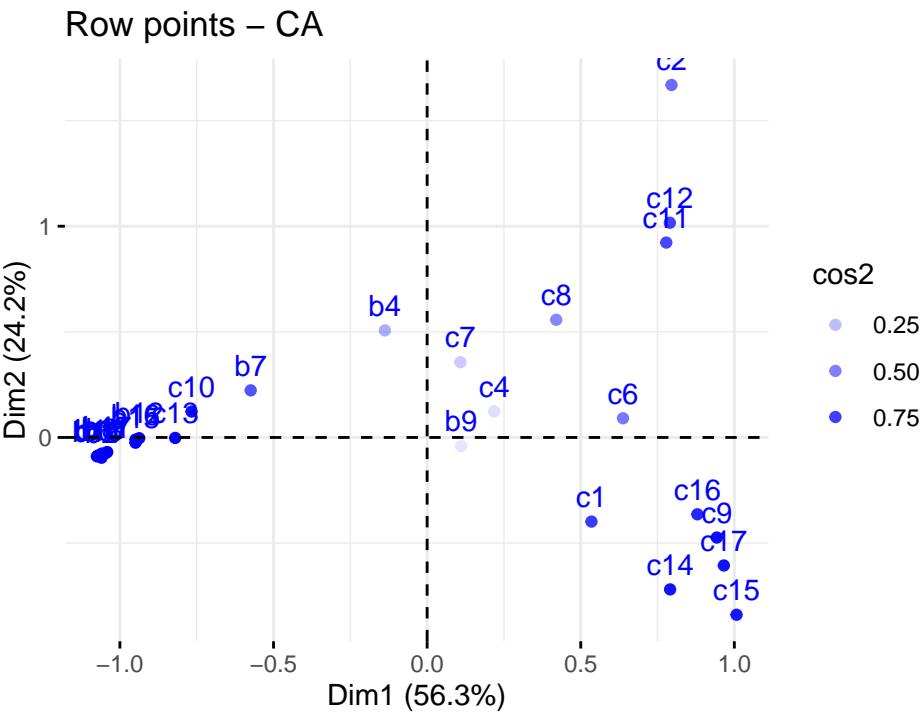


Figure 14: Diagrama de mapa de puntos en las dos primeras dimensiones del análisis de correspondencias para las muestras, CA().

4.5 Cluster Analysis

Los métodos de búsqueda de patrones o clases: análisis de conglomerados y métodos aglomerativos (KMeans/PAM) (fuente Sanchez.A). Inicialmente realizaremos el análisis cluster con la función PomaClust() del paquete Poma sobre el conjunto de datos completos y posteriormente haremos pruebas sobre el subconjunto obtenidos a partir del test de Lima.

```
se_poma_imputado_log_scaling_clust <- PomaClust(se_poma_imputado_log_scaling)
```

El número óptimo de clusters obtenido con la función PomaClust() es 4

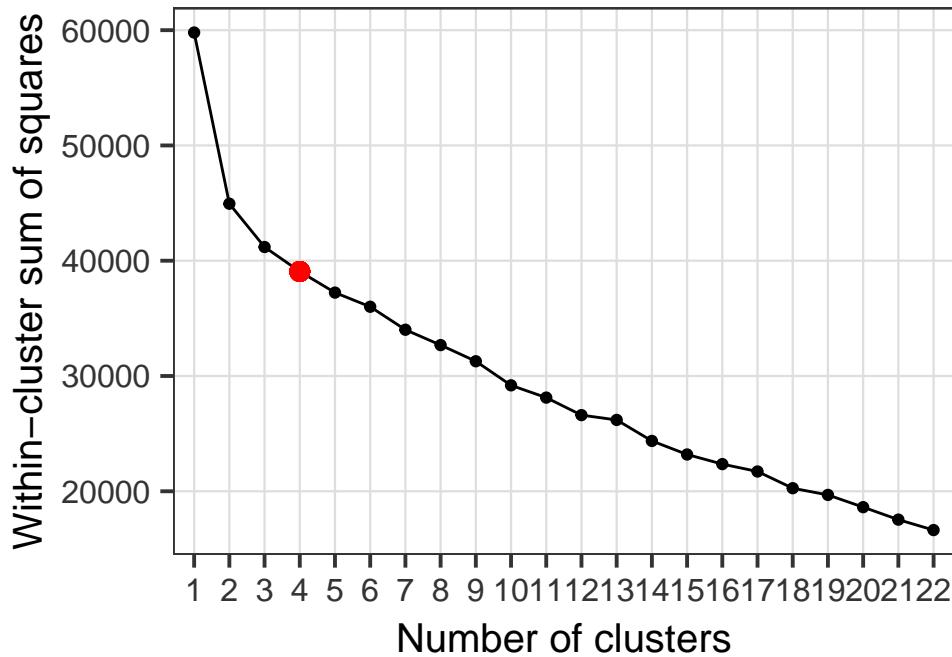


Figure 15: Diagrama de análisis de numero de clusters óptimo, PomaClust()

Table 17: Primeras 10 muestras sobre los 4 primeros componentes, factors.

sample	clust	Dim1	Dim2
b1	4	-17.8181106	3.7058416
b10	2	-15.0795872	-9.2163014
b11	4	-34.2884118	0.6802952
b12	4	-22.5685481	-4.4811062
b13	2	0.0201414	-5.9333406
b14	2	2.6530575	-13.2944065
b15	4	-17.4084078	-4.7505479
b16	3	23.2028502	4.5705725
b17	1	24.9445692	-7.6424656
b2	4	-20.2416421	3.2006506

En el siguiente gráfico vemos el diagrama de puntos para las dos primeras dimensiones diferenciando cada cluster por color.

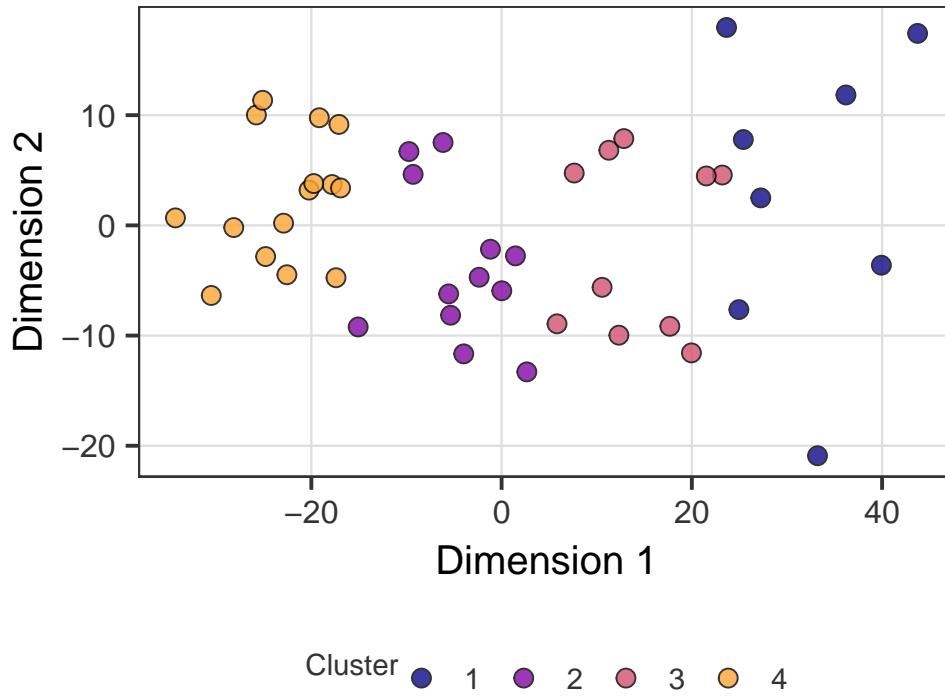


Figure 16: Digrama de puntos para las dos primeras dimensiones.

4.5.1 Agrupación jerárquica

El método jerárquico crea un árbol de clasificación o dendograma. Estrictamente, estos métodos no definen grupos, sino la estructura de asociación en cadena que puede existir entre los elementos (fuentes Peña y Sanchez.A). La similitud entre casos puede medirse mediante la distancia entre ellos. Hay diferentes formas de medir esta distancia, la más sencilla es la distancia euclídea. Al calcular la distancia entre cada par obtenemos una matriz de distancias (fuente Sanchez.A).

Como ya hemos mencionado, una vez realizados las pruebas sobre el conjunto de datos completo lo haremos sobre el subconjunto ya mencionado.

Agrupación de características

Utilizaremos diferentes métodos de conglomeración: `average`, `centroid`, y el método para calcular la distancia será `euclidian`. Representaremos la estructura jerárquica mediante dendrogramas:

```
carac_dist_single <- dist(sample_caracteristicas_int, method="euclidian")

carac_hc_UPGMA <- hclust(carac_dist_single, method="average")
carac_hc_centroid <- hclust(carac_dist_single, method="centroid")
```

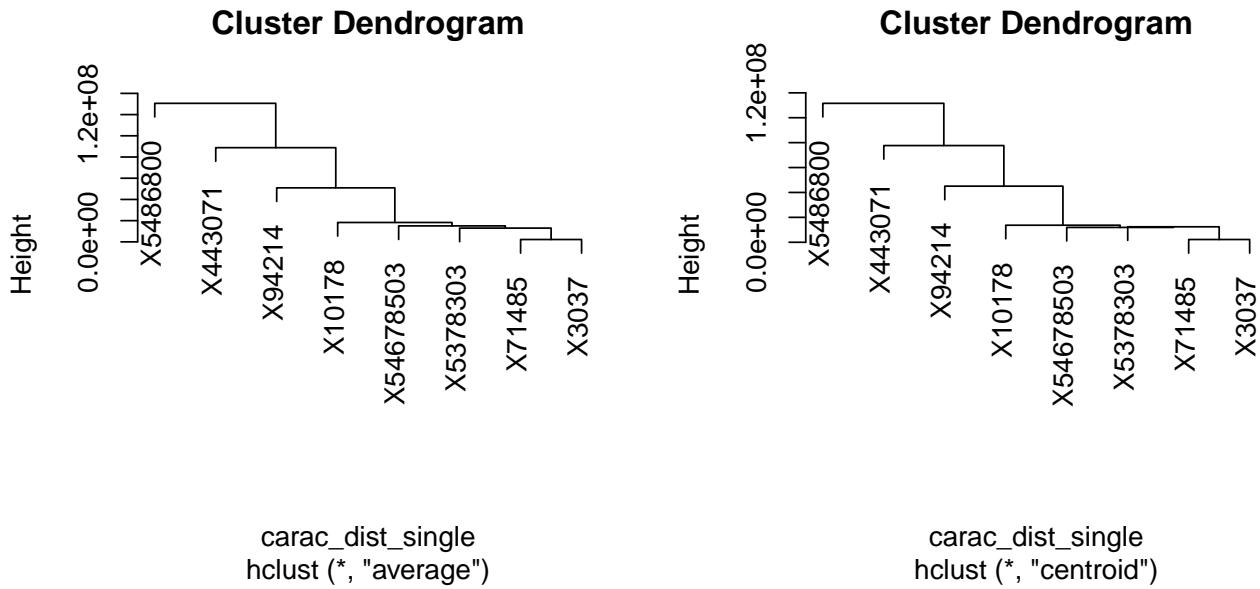


Figure 17: Dendograma clusters para agrupación de características para método average y centroid sobre distancia euclídea.

Agrupación de muestras

Utilizaremos diferentes métodos de conglomeración: `average`, `centroid`, y el método para calcular la distancia será `euclidian`. Representaremos la estructura jerárquica mediante dendrogramas:

```
carac_dist_single <- dist((sample_caracteristicas_int_t), method="euclidian")
carac_hc_UPGMA <- hclust(carac_dist_single, method="average")
carac_hc_centroid <- hclust(carac_dist_single, method="centroid")
```

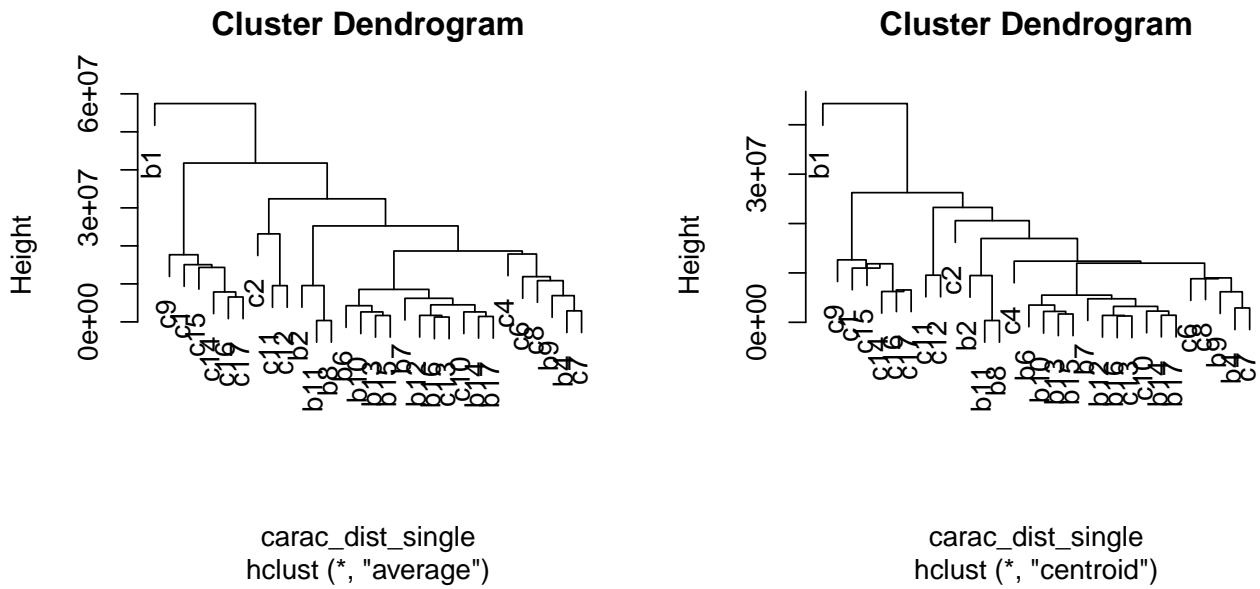


Figure 18: Dendograma clusters para agrupación de muestras para método average y centroid sobre distancia euclídea.

4.5.2 Agrupación por el método de las k-means

La agrupación por el método k-medias es un método no jerárquico que crea una única partición de los datos. Se especifica a priori el número de grupos de casos que se desean formar. El método de las k-means permite estudiar la consistencia de la agrupación a partir del cálculo de las sumas de cuadrados dentro de cada grupo (fuente Sanchez.A).

Agrupación de características

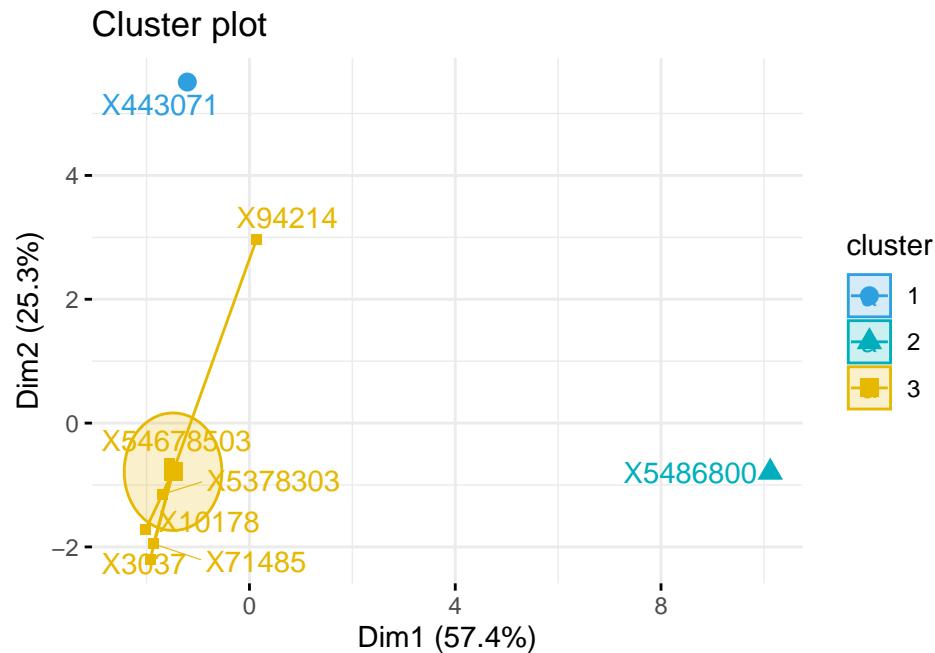


Figure 19: Diagrama de mapa de puntos sobre dos dimensiones a partir de agrupación por características con k-medias.

X443071	X5378303	X71485	X5486800	X3037	X54678503	X94214	X10178
1	3	3	2	3	3	3	3

Agrupación de muestras

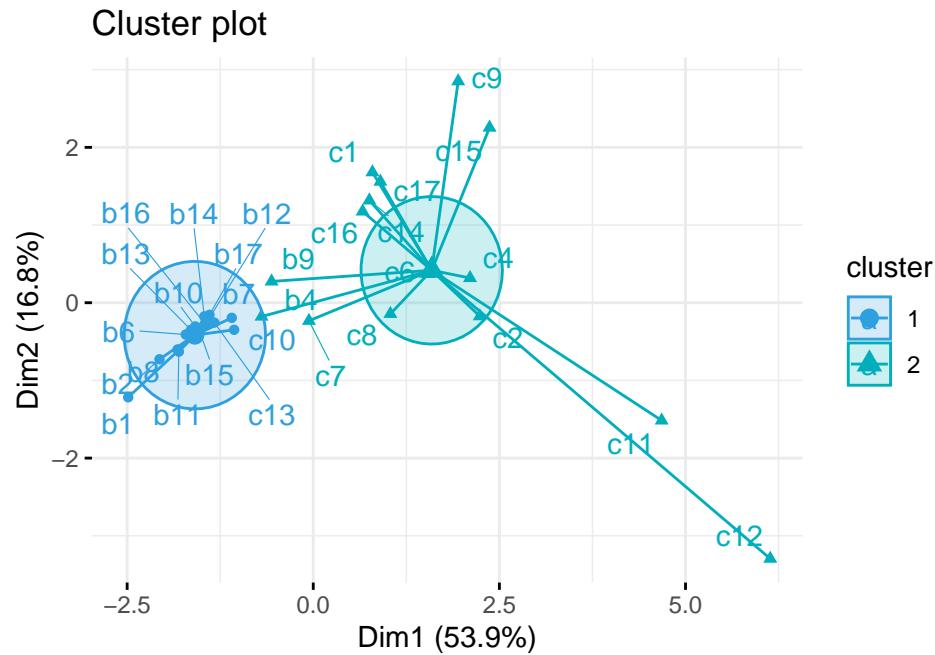


Figure 20: Digráma de mapa de puntos sobre dos dimensiones a partir de agrupación por características con k -medias.

	b1	b10	b11	b12	b13	b14	b15	b16	b17	b2	b4	b6	b7	b8	b9	c1	c10	c11	c12	c13
1	1	1	1	1	1	1	1	1	1	1	2	1	1	1	2	2	1	2	1	
c14	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	1	

En el gráfico anterior parece que se puede apreciar como las muestras de control se agrupan en torno al centroide de un cluster y las del otro tratamiento en torno al centroide del otro cluster.

4.6 Correlaciones

Calculamos las correlaciones entre características mediante la función PomaCorr(),

```
poma_cor <- PomaCorr(se_poma_imputado_log_scaling, label_size = 8)
```

y mostramos las correlaciones entre características > 0.99 .

Table 18: Correlaciones mayores a 0.99 obtenidas mediante PomaCorr().

feature1	feature2	corr	pvalue	FDR
X51	X643757	0.9990976	0	0
X998	X5610	0.9978092	0	0
X441125	X656979	0.9967897	0	0
X7057	X1150	0.9963910	0	0
X439351	X439774	0.9939417	0	0
X802	X24798720	0.9935509	0	0
X244	X5816	0.9932277	0	0
X440020	X48	0.9924279	0	0
X18189	X6406	0.9923702	0	0
X171	X12377	0.9920545	0	0
X29393	X1021	0.9915518	0	0
X11799	X33474	0.9909569	0	0
X5960	X65065	0.9905875	0	0
X27689	X4195	0.9904087	0	0

5 Descarga de datos

Finalmente exportamos 3 ficheros .csv: características.csv, metadatos.csv y metabolitos.csv. Y un fichero .Rda que contiene lo siguientes objetos de tipo `SummarizedExperiment`: se_poma, se_poma_imputado, se_poma_imputado_log_scaling, se_poma_preprocesado

```
write.table(caracteristicas, sep=";", file="caracteristicas.csv")
write.table(metadatos, sep=";", file="metadatos.csv")
write.table(metabolitos, sep=";", file="metabolitos.csv")

save(se_poma, se_poma_imputado,
      se_poma_imputado_log_scaling, se_poma_preprocesado, file = "azs_mydata.rda")
```

Se pueden consultar el presente documento y así como las exportaciones realizadas en la siguiente dirección git:
<https://github.com/azenzano/Zenzano-Sarasola-Ager-PEC1>.

6 Información de servicio

```
sessionInfo()

R version 4.4.1 (2024-06-14)
Platform: aarch64-apple-darwin20
Running under: macOS Sonoma 14.5

Matrix products: default
BLAS:      /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib; LAPACK vers

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: Europe/Madrid
tzcode source: internal

attached base packages:
[1] stats4    stats     graphics  grDevices utils      datasets  methods
[8] base

other attached packages:
[1] factoextra_1.0.7          made4_1.78.0
[3] SummarizedExperiment_1.34.0 GenomicRanges_1.56.2
[5] GenomeInfoDb_1.40.1        IRanges_2.38.1
[7] S4Vectors_0.42.1          MatrixGenerics_1.16.0
[9] matrixStats_1.4.1          Biobase_2.64.0
[11] BiocGenerics_0.50.0       scatterplot3d_0.3-44
[13] RColorBrewer_1.1-3         FactoMineR_2.11
[15] gplots_3.2.0               patchwork_1.3.0
[17] ggtext_0.1.2              POMA_1.14.0
[19] kableExtra_1.4.0           ggrepel_0.9.6
[21] rvest_1.0.4                lubridate_1.9.3
[23]forcats_1.0.0              stringr_1.5.1
[25] purrrr_1.0.2              readr_2.1.5
[27] tidyrr_1.3.1              tibble_3.2.1
[29] ggplot2_3.5.1              tidyverse_2.0.0
[31] dplyr_1.1.4                metabolomicsWorkbenchR_1.14.2

loaded via a namespace (and not attached):
[1] rstudioapi_0.17.1          jsonlite_1.8.9
[3] MultiAssayExperiment_1.30.3 magrittr_2.0.3
[5] estimability_1.5.1          farver_2.1.2
[7] rmarkdown_2.29.1            zlibbioc_1.50.0
[9] vctrs_0.6.5                rstatix_0.7.2
[11] janitor_2.2.0              tinytex_0.54
[13] htmltools_0.5.8.1          S4Arrays_1.4.1
[15] curl_5.2.3                 broom_1.0.7
[17] Formula_1.2-5              SparseArray_1.4.8
[19] KernSmooth_2.23-24         htmlwidgets_1.6.4
[21] emmeans_1.10.5             commonmark_1.9.2
[23] lifecycle_1.0.4             pkgconfig_2.0.3
[25] Matrix_1.7-1               R6_2.5.1
```

```
[27] fastmap_1.2.0           GenomeInfoDbData_1.2.12
[29] snakecase_0.11.1        digest_0.6.37
[31] selectr_0.4-2          colorspace_2.1-1
[33] ggpubr_0.6.0           vegan_2.6-8
[35] labeling_0.4.3         fansi_1.0.6
[37] timechange_0.3.0       httr_1.4.7
[39] abind_1.4-8            mgcv_1.9-1
[41] compiler_4.4.1          withr_3.0.2
[43] backports_1.5.0        carData_3.0-5
[45] highr_0.11              ggsignif_0.6.4
[47] MASS_7.3-61             DelayedArray_0.30.1
[49] gtools_3.9.5            caTools_1.18.3
[51] permute_0.9-7          flashClust_1.01-2
[53] tools_4.4.1             glue_1.8.0
[55] nlme_3.1-166            gridtext_0.1.5
[57] grid_4.4.1               cluster_2.1.6
[59] ade4_1.7-22             generics_0.1.3
[61] gtable_0.3.6             tzdb_0.4.0
[63] data.table_1.16.2       hms_1.1.3
[65] xml2_1.3.6              car_3.1-3
[67] utf8_1.2.4               XVector_0.44.0
[69] pillar_1.9.0             markdown_1.13
[71] limma_3.60.6            splines_4.4.1
[73] lattice_0.22-6          tidyselect_1.2.1
[75] knitr_1.48               bookdown_0.41
[77] svglite_2.1.3            xfun_0.49
[79] statmod_1.5.0            DT_0.33
[81] stringi_1.8.4            UCSC.utils_1.0.0
[83] yaml_2.3.10              evaluate_1.0.1
[85] multcompView_0.1-10      cli_3.6.3
[87] ontologyIndex_2.12       xtable_1.8-4
[89] systemfonts_1.1.0        munsell_0.5.1
[91] struct_1.16.0             Rcpp_1.0.13-1
[93] parallel_4.4.1           leaps_3.2
[95] bitops_1.0-9              viridisLite_0.4.2
[97] mvtnorm_1.3-2            scales_1.3.0
[99] crayon_1.5.3             rlang_1.1.4
```

7 References

- Agarwal.V. "Joint Negative and Positive Mode Development for Combination LC-MS/MS Method." <https://www.nebiolab.com/positive-and-negative-mode-in-mass-spectroscopy/>.
- Bakker.J. "Applied Multivariate Statistics in r." <https://uw.pressbooks.pub/appliedmultivariatestatistics/>.
- Castellano-Escuder. "Poma Workflow." <https://bioconductor.org/packages/3.21/bioc/vignettes/POMA/inst/doc/POMA-workflow.html>.
- Peña, Daniel. "Análisis de Datos Multivariantes Mc GrawHill."
- Poma. "Tools for Omics Data Analysis." <https://bioconductor.org/packages/release/bioc/html/POMA.html>
- Ritchie, Phipson, M. E. "Limma Powers Differential Expression Analyses for RNA-Sequencing and Microarray Studies." <https://research.monash.edu/en/publications/limma-powers-differential-expression-analyses-for-rna-sequencing->.
- Sanchez.A. "Casos y Ejemplos de Análisis Multivariante Con r." <https://aspteaching.github.io/AMVCasos/>.
- _____. "Exploración Multivariante de Datos Ómicos."
- ST000291. "LC-MS Based Approaches to Investigate Metabolomic Differences in the Urine of Young Women After Drinking Cranberry Juice or Apple Juice." <https://www.metabolomicsworkbench.org/data/DRCCMetadata.php?Mode=Study&StudyID=ST000291>.
- SummarizedExperiment. "SummarizedExperiment: A Container (S4 Class) for Matrix-Like Assays." <https://bioconductor.org/packages/release/bioc/html/SummarizedExperiment.html>.
- www.metabolomicsworkbench.org. "Raw Data File Inspection and Reuse." <https://www.metabolomicsworkbench.org>