Introducción a la programación estadística en R enfocado al análisis de metabolitos por cromatografía de gases

Manual del curso

Elaboró: M.C. Arturo Ramírez-Ordorica

Mayo 2023

¡Bienvenidos al apasionante mundo de la programación en R!

En este curso, te embarcarás en un viaje fascinante en el que combinarás el poder del lenguaje de programación R con su potencial para explorar y analizar los datos generados por la cromatografía de gases.

La cromatografía de gases es una técnica esencial en el campo del análisis químico. Nos permite separar y cuantificar compuestos volátiles presentes en muestras complejas. Sin embargo, dominar esta técnica no es suficiente en la era de los datos. La programación se ha convertido en una habilidad crucial para explorar, visualizar y sacar el máximo provecho de los resultados cromatográficos, y también en casi todas las áreas de las ciencias.

No importa si eres un químico o biólogo curioso que busca ampliar sus habilidades analíticas, o un programador entusiasta que desea adentrarse en el apasionante campo de la química analítica. Este curso está diseñado para adquirir la habilidades básicas necesarias para comenzar a utilizar éste maravilloso recurso de programación.

¡Bienvenidos a bordo! Estoy seguro de que este curso te abrirá las puertas a un nuevo mundo de posibilidades en el emocionante campo de la programación estadística.

Una breve historia de R

El lenguaje R, nacido en los laboratorios de estadística de la Universidad de Auckland, Nueva Zelanda, en 1992, se ha convertido en uno de los lenguajes de programación más populares y poderosos para el análisis y minería de datos. Fue desarrollado por Ross Ihaka y Robert Gentleman con el objetivo de proporcionar una herramienta flexible y de código abierto para la estadística y la visualización de datos, con el fin de ser utilizado para la enseñanza y la investigación. Al ser inspirado en el lenguaje S, R heredó su sintaxis elegante y su capacidad para manipular y analizar datos de manera eficiente. Hoy en día, R es ampliamente utilizado en la industria, la academia y la investigación, desempeñando un papel fundamental en el avance de la ciencia de datos y el análisis estadístico. Alrededor de R se ha construido una comunidad muy activa que da ayuda y orientación a quienes se interesan en aprenderlo, además de generar paquetería orientada a tareas analíticas muy específicas en diferentes áreas del conocimiento.

¿Por qué usar R?

Versatilidad y potencia: R es un lenguaje de programación versátil y altamente potente diseñado específicamente para el análisis estadístico y la visualización de datos. Con una amplia gama de paquetes y librerías especializadas, R te permite abordar una variedad de problemas y aplicaciones, desde el análisis de datos químicos hasta la modelización estadística compleja.

Comunidad activa y colaborativa: R cuenta con una comunidad de usuarios activa y colaborativa en constante crecimiento. Puedes acceder a una gran cantidad de recursos, tutoriales y ejemplos de código, así como participar en foros y grupos de discusión donde podrás obtener ayuda y compartir conocimientos con otros programadores y analistas de datos.

Herramientas de visualización avanzadas: La visualización de datos es una parte esencial del análisis de datos y la toma de decisiones informadas. R ofrece una amplia gama de herramientas de visualización, como ggplot2 y lattice, que te permiten crear gráficos impresionantes y personalizados para explorar y comunicar tus resultados de manera efectiva.

Integración con otras tecnologías: R se integra fácilmente con otras tecnologías y lenguajes de programación, lo que te brinda flexibilidad y capacidad de expansión. Puedes combinar R con herramientas de bases de datos, como SQL, o utilizarlo en conjunción con lenguajes como Python para aprovechar lo mejor de ambos mundos en tus proyectos de análisis de datos.

Oportunidades laborales y académicas: El dominio de R abre un amplio abanico de oportunidades laborales en el campo del análisis de datos, la ciencia de datos y la estadística. Las habilidades en R son altamente demandadas en la industria y la investigación, lo que te brinda la posibilidad de trabajar en sectores tan diversos como la salud, el medio ambiente, las finanzas, la tecnología y más.

Objetivo del curso

Introducir al participante en los fundamentos de programación estadística y matemática por el uso del lenguaje R, enfocado al análisis de datos en cromatografía de gases para la exploración de perfiles químicos.

En este manual, todo lo que pueda ser introducido en la consola de R se escribirá asi:

```
#Este es un comentario y no es interpretado por R como un objeto
Esto es un objeto y puede ser interpretado como una función o invocar algo
```

De esta forma, deberías de poder pegar el código en tu computadora y ejecutarlo.

Objetos básicos

En R, un objeto es una entidad fundamental que almacena datos y contiene información sobre cómo trabajar con esos datos. Puedes pensar en un objeto como un contenedor que guarda tanto los valores de los datos como las funciones y atributos asociados a ellos. Cada objeto en R tiene un nombre único que te permite acceder y manipular sus contenidos. Los objetos pueden ser de diferentes tipos, como vectores, matrices, dartaframes o listas, entre otros. Además, R es un lenguaje orientado a objetos, lo que significa que los objetos pueden tener métodos y clases que definen su comportamiento y características específicas. Los objetos en R no solo te permiten almacenar datos, sino también realizar operaciones, análisis y manipulaciones complejas en ellos. Al comprender y trabajar con objetos en R, puedes aprovechar al máximo la capacidad de este lenguaje para el análisis de datos y la programación estadística. En éste curso nos enfocaremos en los siguientes objetos básicos:

1. Vector

En R, un vector es una colección ordenada de elementos del mismo tipo, como números, caracteres o valores lógicos. Los vectores en R pueden tener una longitud arbitraria y se pueden acceder a sus elementos utilizando [índices]. Estos elementos pueden ser modificados, reemplazados o agregados, lo que brinda flexibilidad en el manejo de los datos. Los vectores son ampliamente utilizados en R para almacenar y manipular conjuntos de datos, realizar cálculos y realizar operaciones vectorizadas, lo que permite un procesamiento eficiente y simplificado de los datos.

```
##Vectores
#Un vector es una colección de objetos que tienen la misma categoría de dato
#(numerico, caracter, lógico, etc)
c()#Función concatenar
un_vector<-c(1,2,3,6,11,43,1e10)
print(un_vector)

un_vector[5]
un_vector[c(2,6)]
un_vector[-5]

#Podemos crear un vector con vector()
un_vector2<-vector("numeric",3)
?vector()
#Crear un vector lógico de tamaño 10

#¿Cuál es su longitud?

#Funciones útiles para crear vectores:
seq(from=2,to=20,by=2)#Esta es una "función" de nombre seq() con argumentos
#from=, to=, by=. Casi todas las funciones requieren de argumentos específicos
rep(2,4)#Para repetir un objeto n cantidad de veces

#Ejercicio
#Crear un vector con los números del 1 al 100 de 5 en 5
#Crear un vector con 10 nombres
```

```
#unirlos
#¿Qué longitud tiene?

#OPERACIONES MATEMÁTICAS
#Podemos hacer operaciones con vectores...los operadores son fáciles de
#identificar por su sintaxis

#Ejercicio
#Crear un vector con la raíz cuadrada de los primeros 100 enteros
#Crear un vector con el cuadrado de los primeros 100 enteros
#arcotangente del vector anterior

##Nombres "names()"
vector_3<-vector("numeric",6)
print(vector_3)
names(vector_3)
##Crea un vector de caracteres con los nombres
#"Jośe","Inés","Raúl","Arturo","Berenice","Maricela"
#¿Cómo llamamos a los casos de cada posición en el vector?
#Asigne calificaciones a cada caso

mean()#media de calificaciones
sd()#Desviación estandar
```

2. Matrices

Una matriz en R es una estructura bidimensional que organiza datos en filas y columnas. Al igual que un vector, una matriz es un objeto fundamental en R que puede contener elementos del mismo tipo, como números, caracteres o valores lógicos. A diferencia de un vector, una matriz tiene una dimensión adicional, lo que la convierte en una estructura tabular de dos dimensiones. Puedes pensar en una matriz como una tabla en la que los datos se alinean en filas y columnas, lo que permite un acceso eficiente y una manipulación sistemática de los datos. Las matrices en R son especialmente útiles cuando trabajas con datos estructurados y necesitas realizar operaciones matemáticas o estadísticas en ellos. Puedes acceder a los elementos de una matriz utilizando índices que especifican tanto la posición de la fila como la columna. En resumen, las matrices en R son una herramienta poderosa para organizar y manipular datos en una estructura tabular de dos dimensiones, facilitando así el análisis y la transformación de datos en un contexto matricial.

```
#MATRICES
#Pueden ser concebidos como conjuntos de vectores de la misma longitud
#ordenados en filas y/o columnas.
#Están compuestos por la misma clase de elementos (a semejanza de los vectores)
matrix(seq(1:20),nrow=5,ncol=4)
#Llenar por filas??
matriz_1<-matrix(seq(1:20),nrow=5,ncol=4,byrow=TRUE)
##Posiciones [fila, columna]
matriz_1[3,2]
##Podemos hace operaciones sobre matrices.
#Suma,multiplicación por un escalar
matriz_2<-matrix(seq(1:20),nrow=5,ncol=4,byrow=TRUE)
matriz_3<-matrix(seq(1:20),nrow=5,ncol=4)
```

```
matriz_2+matriz_3
matriz_2*matriz_3#multiplicación elemento a elemento
#dim() para ver las dimensiones en la matríz
dim(matriz_2)
#Multiplicacion matricial
matriz_2 %*% t(matriz_3)#t() transpone la matriz y le das las dimensiones
#adecuadas para multiplicarlas (5x4)%*%(4x5)=5x5

#Hagamos una pequeña gŕafica
heatmap(matriz_2,Rowv = NA,Colv = NA,scale="none")
heatmap(matriz_3,Rowv = NA,Colv = NA,scale = "none")
heatmap(matriz_2 %*% t(matriz_3),Rowv = NA,Colv = NA,scale = "none")

class(matriz_2)
typeof(matriz_2)
matriz_2[2,]

#Para muchos casos, es conveniente saber los nombres de las columnas y
#renglones de las matrices
colnames(matriz_2)
row.names(matriz_2)
#Asignemos nombres a las columnas y las filas de la matriz

#Podemos crear matrices uniendo filas y columnas
cbind()#unir columnas
rbind()#unir filas
```

3. Listas

En R, una lista es una estructura de datos versátil y flexible que permite almacenar diferentes tipos de elementos en un solo objeto. A diferencia de las matrices y los vectores, una lista puede contener elementos de diferentes longitudes y tipos, como vectores, matrices, marcos de datos e incluso otras listas. Es como un contenedor que puede guardar distintos objetos en su interior. Cada elemento de la lista puede tener su propio nombre o etiqueta, lo que facilita el acceso y la manipulación de datos específicos dentro de la lista. Las listas son especialmente útiles cuando necesitas almacenar conjuntos heterogéneos de información o cuando deseas agrupar objetos relacionados en una única entidad. Además, las listas en R permiten una estructura de datos jerárquica y anidada, lo que brinda una gran flexibilidad para organizar y manipular datos complejos.

```
##LISTAS
#Es un objeto que contiene elementos de diferentes clases:
lista_1<-list("a",TRUE,FALSE,vector_3,matriz_2)
length(lista_1)
lista_1[[3]]
lista_1[[5]]
lista_1[[5]][3,2]

length(lista_1)
#Demosle nombres
names(lista_1)<-c("Caracter","Verdadero","Falso","Un vector","Una matriz")
#Llamemos elementos de la lista
lista_1$Verdadero
```

4. Dataframe

Un dataframe es una estructura de datos tabular que organiza información en filas y columnas, similar a una tabla o a una hoja de cálculo. Es una de las estructuras de datos más utilizadas en R y se compone de vectores de igual longitud que representan las columnas, donde cada columna puede contener diferentes tipos de datos, como números, caracteres o factores. A diferencia de una matriz, los dataframes en R permiten tener columnas con diferentes tipos de datos y también pueden tener nombres de columna asociados. Los dataframes son especialmente útiles cuando trabajas con conjuntos de datos heterogéneos que pueden contener variables de diferentes tipos y necesitas realizar operaciones, filtrados y análisis sobre ellos. Los dataframes en R facilitan el acceso y la manipulación de datos mediante la identificación de columnas y filas mediante nombres o índices, permitiendo así un manejo intuitivo y estructurado de los datos tabulares en el análisis de datos y la estadística.

```
##DATAFRAME
#Son tablas de datos, y son un tipo especial de lista en forma de tabla.
#Los dataframe son el tipo más común de objeto que se usa en aplicaciones de
#análisis de datos

nombres=c("Jośe","Inés","Raúl","Arturo","Berenice","Maricela")
calificaciones=c(9,8,10,9,8,10)
salon=c("A","C","E","C","D","C")
mi_tabla=cbind(nombres,calificaciones,salon)
print(mi_tabla)
data(iris)
iris

#Nombres
colnames()
rownames()

#Para invocar columnas específicas de un data.frame, se utiliza los operadores
#[] y el $. No olvidar que un dataframe es un tipo de lista.
names(iris)
iris$Petal.Length
#Ejercicio:Poner un nombre a cada fila del dataframe iris. Con formato
#"especie_número consecutivo. Usar paste()
```

Operadores lógicos

Los operadores lógicos son herramientas fundamentales para evaluar y comparar condiciones en expresiones booleanas. Estos operadores permiten tomar decisiones basadas en la veracidad o falsedad de ciertas afirmaciones. Los operadores lógicos más comunes en R son:

1. Operador de igualdad (==): Este operador compara si dos valores son iguales y devuelve TRUE si la afirmación es verdadera y FALSE si es falsa. Por ejemplo, 3 == 3 evaluaría como TRUE.

2. Operador de desigualdad (!=): Este operador verifica si dos valores son diferentes y devuelve TRUE si la afirmación es verdadera y FALSE si es falsa. Por ejemplo, 5 != 2 evaluaría como TRUE.

3. Operadores de comparación (<, >, <=, >=): Estos operadores comparan dos valores y devuelven TRUE o FALSE según la relación establecida. Por ejemplo, 4 < 7 evaluaría como TRUE, mientras que 3 >= 5 evaluaría como FALSE.

4. Operador lógico AND (&&): Este operador combina dos condiciones y devuelve TRUE si ambas condiciones son verdaderas, y FALSE si alguna de ellas es falsa. Por ejemplo, 2 < 5 && 3 < 7 evaluaría como TRUE.

5. Operador lógico OR (||): Este operador también combina dos condiciones, pero devuelve TRUE si al menos una de las condiciones es verdadera, y FALSE si ambas son falsas. Por ejemplo, 4 > 6 || 3 < 5 evaluaría como TRUE.

Además de estos operadores, R también ofrece el operador de negación (!), que invierte el valor de una afirmación. Por ejemplo, !(2 < 5) evaluaría como FALSE.

Estos operadores lógicos son fundamentales para controlar el flujo del programa, realizar filtrados de datos y construir expresiones condicionales en R. Con su ayuda, puedes tomar decisiones lógicas y escribir código más eficiente y preciso.

```
##OPERADORES LÓGICOS
# !=, ==, <, >, <=, >=
#Usar operadores lógicos para encontrar el número de individuos de cada especie en
#el vector
#de nombres, y asignar un nombre a las filas con formato.
# "especie_número de individuo"


#Ejercicio: Crear un dataframe que simule una matriz de datos con
#50 observaciones y 100 señales m/z que tenga una distribución normal
#Introducir 1500 ceros aleatoriamente en el dataframe
datos<-rnorm(50*100,mean=0,sd=2)
muestra<-sample(datos,1500)#sample() sirve para hacer un muestreo de datos
posiciones<-match(muestra,datos)
datos[posiciones]<-0
print(datos)#Observe los ceros
matriz_mz<-matrix(datos,ncol=100,nrow=50,byrow=T)
dim(matriz_mz)
mz<-seq(from=50,to=250,by=200/99)
mz<-round(mz,3)##función para redondear con 3 decimales
sufijo<-rep("m.z",length(mz))
etiqueta<-paste(mz,sufijo,sep="")
colnames(matriz_mz)<-etiqueta
obs<-1:50
obs_nomb<-paste(rep("obs",50),obs,sep = "")
rownames(matriz_mz)<-obs_nomb
mz_tabla<-data.frame(obs_nomb,matriz_mz)
dim(mz_tabla)
```

Estructuras de control de datos

Las estructuras de control de datos son herramientas poderosas que te permiten controlar el flujo y la ejecución de tu código en función de condiciones específicas. Estas estructuras te brindan la capacidad de tomar decisiones, repetir tareas y aplicar lógica condicional en tus programas.

1. Estructura if-else: Es una estructura de control condicional que evalúa una condición y ejecuta un bloque de código si la condición es verdadera. Si la condición es falsa, se puede proporcionar un bloque de código alternativo para ejecutar. Esta estructura es ideal para tomar decisiones basadas en condiciones booleanas.
2. Estructura for: Es una estructura de control iterativa que permite repetir un bloque de código un número específico de veces. Se utiliza para recorrer secuencias de valores o elementos, y realizar una acción en cada iteración. Es útil cuando se necesita realizar una tarea repetitiva con una cantidad conocida de repeticiones.
3. Estructura while: Es otra estructura de control iterativa que ejecuta un bloque de código mientras una condición se mantenga verdadera. La condición se evalúa antes de cada iteración, y si es verdadera, el bloque de código se ejecuta. Esta estructura es útil cuando no se conoce de antemano el número exacto de repeticiones.
4. Estructura repeat: Es una estructura de control iterativa similar a la estructura while, pero se repite de forma indefinida hasta que se cumpla una condición de salida explícita. Para detener el bucle repeat, se utiliza la instrucción break.

Estas estructuras de control de datos en R te brindan flexibilidad y control sobre la ejecución de tu código, permitiéndote automatizar tareas, aplicar condiciones lógicas y realizar operaciones repetitivas de manera eficiente. Dominar estas estructuras te permitirá escribir código más sofisticado y funcional en R, abriendo un mundo de posibilidades en el análisis de datos y la programación estadística.

```
########BASES DE ESTRUCTURAS DE CONTROL
##Ejercicio: imputar las celdas con ceros por la media de la columna
#correspondiente
mz_tabla[,-1]
media_intensidad<-sapply(mz_tabla[,-1],mean,simplify = TRUE)#sapply entrega el
#resultado de una
#operación aplicada sobre cada columna de un dataframe. El promedio de las
#intensidades de cada m/z

##for...
#Permite repetir recursivamente una operación por un número dado de ciclos
mz_imputar<-mz_tabla[,-1]
#Vamos a repetir una operación de 1 al 100 (el número de columnas)
for (i in 1:length(media_intensidad)){
  mz_imputar[,i][mz_imputar[,i]==0]<-media_intensidad[i]
}

mz_imputar[,1][mz_imputar[,1]==media_intensidad[1]]
```

```
mz_imputado<-data.frame(obs_nomb,mz_imputar)##Dataframe imputado con la media
```

Gráficas en R

El potencial gráfico de R es extraordinario y ofrece una amplia gama de herramientas para crear visualizaciones impactantes y significativas. Con la ayuda de paquetes como ggplot2, lattice y plotly, entre otros, R te permite generar gráficos de alta calidad y personalizados para explorar, analizar y comunicar datos de manera efectiva. Puedes crear gráficos estáticos y dinámicos, desde simples gráficos de barras y diagramas de dispersión hasta visualizaciones más complejas como gráficos de líneas, diagramas de cajas, mapas y gráficos interactivos. Además, R proporciona una gran flexibilidad para ajustar aspectos visuales como colores, etiquetas, leyendas y títulos, lo que te permite personalizar tus gráficos para adaptarlos a tus necesidades y presentar tus resultados de manera impactante. Ya sea que estés explorando datos, comunicando hallazgos o construyendo visualizaciones interactivas para aplicaciones web, el potencial gráfico de R es una herramienta poderosa que te ayudará a dar vida a tus datos y transmitir información de manera clara y atractiva.

```
###HISTOGRAMAS EN R####
hist(as.matrix(mz_tabla[,-1]),breaks = 25)
hist(as.matrix(mz_imputado[,-1]),breaks = 25)
#as.matrix se conoce como función de coerción. Las funciones de coerción
#obligan al programa a interpretar un objeto como uno de una clase determinada
#funciones de coerción comunes: as.matrix(), as.character(), as.list(),
as.data.frame()

#Ejercicio: crear un vector que simule datos periódicos con un error aleatorio
#y con regiones mínimas planas.
x<-seq(from=0, to=40,by=0.1)
sin_datos<-sin(x)
plot(sin_datos,type = "l")
error<-rnorm(length(sin_datos),mean=0,sd=0.1)
simulado=sin_datos+error
plot(simulado,type="l")
min(simulado)
simulado[simulado<=0]<-0
plot(x,simulado,type="l")
```

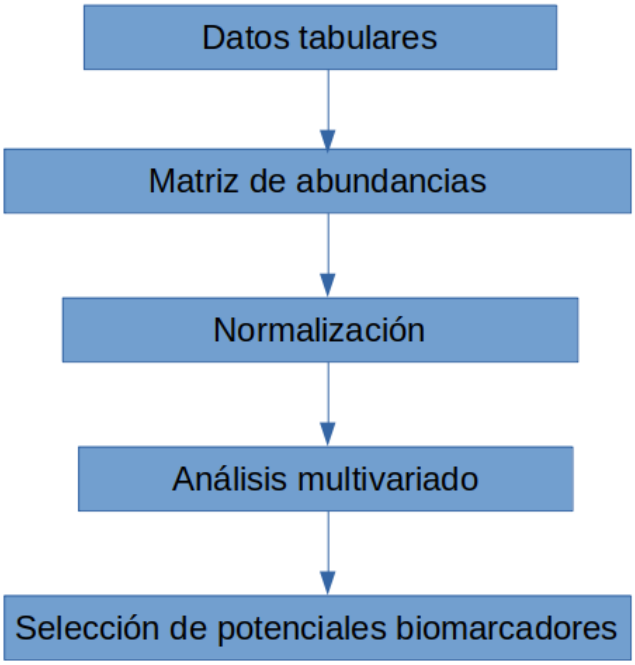
Datos cromatográficos

En cromatografía acoplada a espectrometría de masas, los datos generados se clasifican en tres categorías principales: tiempos de retención (RT), intensidad y/o abundancia de cada componente, y masa-carga (m/z) de las especies cargadas resultantes del proceso de ionización. En consecuencia muchas veces trabajamos con datos tridimensionales que incluyen RT, intensidad y m/z. Sin embargo, en el caso específico de la cromatografía de gases nuestro interés casi siempre se centra en los tiempos de retención de cada compuesto presente en la muestra y en las áreas bajo la curva o abundancias correspondientes. Por lo tanto, el análisis se simplifica al trabajar con datos de dos dimensiones.

Una vez finalizados nuestros experimentos, el enfoque para el análisis cuantitativo de los datos generados por GC-MS generalmente sigue los siguientes pasos:

1. Los tiempos de retención (RT) y las abundancias se extraen en formato tabular, utilizando comúnmente el software proporcionado por el fabricante del equipo.
2. Debido a las variaciones aleatorias entre repeticiones y/o experimentos, así como a las variaciones inherentes a la naturaleza de nuestras muestras, los compuestos eluirán con cierta variación temporal. Por lo tanto, es necesario asignar el orden correcto de cada componente en cada muestra y establecer su correspondencia con las demás. A este proceso se le conoce como "alineamiento". El resultado suele ser una matriz con muestras en filas y compuestos en columnas, donde cada entrada representa la abundancia de dicho componente en cada muestra.
3. En la mayoría de los casos, es necesario normalizar las matrices de abundancia. Esto se puede lograr mediante la centralización de los datos utilizando la media de cada compuesto en nuestro experimento y dividiéndolos por una medida de dispersión, o mediante la normalización por abundancia relativa de cada componente en cada muestra, o por la intensidad máxima en toda la matriz. Esta normalización reduce la variabilidad en las intensidades y mejora la identificación de biomarcadores.
4. Los datos se analizan utilizando los métodos estadísticos apropiados, generalmente mediante técnicas multivariadas que permiten detectar diferencias globales en los perfiles químicos. Existen numerosas técnicas disponibles y se seleccionan en función de los objetivos de investigación.
5. A menudo, nos interesa identificar características que sean propias o que difieran significativamente entre las condiciones experimentales. A estos compuestos los denominamos "biomarcadores", ya que sus abundancias aumentan o disminuyen de manera significativa entre las muestras por razones de relevancia biológica.
6. Se pueden utilizar otros métodos adicionales que brinden información sobre los procesos metabólicos modificados, como el análisis de enriquecimiento metabólico. Existen muchos métodos y aproximaciones a seguir.
7. Por último, el investigador realiza la interpretación de los datos, analizando los resultados obtenidos y extrayendo conclusiones relevantes en función de los objetivos planteados.

La estrategia a seguir puede ser diferente dependiendo de los objetivos del trabajo. Para éste curso introductorio nos apegaremos al modelo anterior como una primera aproximación analítica que es útil para muchos casos.



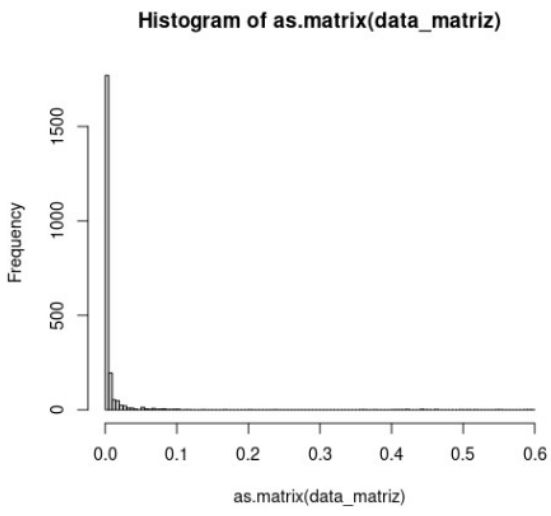
Importar datos en R

Para importar datos desde un archivo CSV, podemos utilizar la función read.csv() en R. Con esta función, podemos especificar la ubicación del archivo CSV como argumento y asignar los datos importados a un objeto, como un dataframe. Además de la ubicación del archivo, podemos ajustar diversos parámetros según sea necesario, como el separador de columnas, si hay una fila de encabezados (que se escribe con el argumento header=) o si queremos convertir ciertas columnas en factores. Una vez que los datos han sido importados, podemos realizar diversas operaciones de exploración, manipulación y análisis utilizando las funciones y técnicas disponibles en R. Importar datos desde un archivo CSV en R es un proceso ágil y nos proporciona una sólida base para llevar a cabo análisis de datos y visualizaciones de manera eficiente y efectiva.

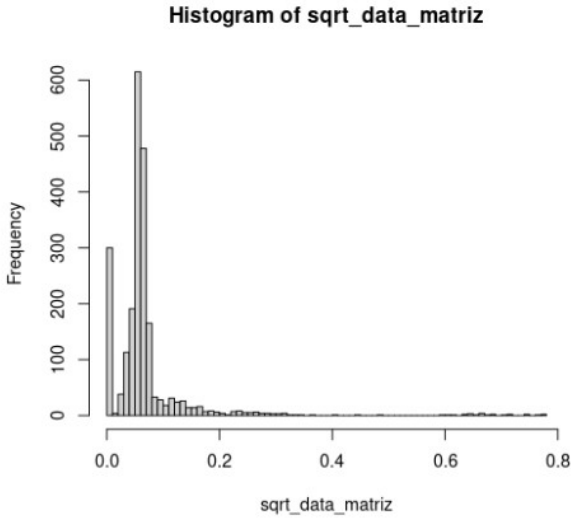
```
library(ggplot2)
library(viridis)
# Cargar los datos
data <- read.csv("tricopca.csv",header = T)
head(names(data),n=10)
#Preparar los datos
data_matriz<-as.matrix(data[,-c(1:2)])
```

Transformación

Puede ser conveniente aplicar una transformación preliminar de la matriz de datos, por ejemplo extrayendo la raíz cuadrada de los datos, para mejorar la dispersión de los mismos y volverlos más normales. Esto permite además darle a las variaciones aleatorios forma normal, lo que facilita la aplicación de diferentes modelos estadísticos a nuestra matriz.



Datos originales



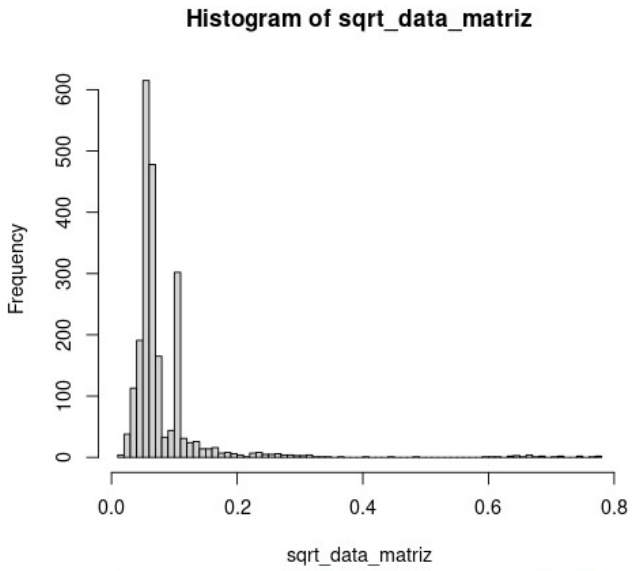
Raíz cuadrada de los datos

```
#Graficar intensidades por cada tratamiento
hist(as.matrix(data_matriz),breaks=100)
sqrt_data_matriz<-sqrt(as.matrix(data_matriz))
hist(sqrt_data_matriz,breaks=100)
```

Imputar datos

Imputar datos significa adicionar un valor definido a aquellas celdas vacías que tienen intensidades muy bajas o cuyo valor inicial es cero. Esto se hace con varios propósitos, pero el principal es que muchas técnicas multivariadas tienen problemas para lidiar con las celdas vacías o los ceros. La interpretación de los datos puede ser confundida con la formación de artefactos analíticos producto de estas celdas. Sin embargo, cuando se tiene certeza de la ausencia de un compuesto se pueden dejar estos valores para considerarlos en el análisis, y usando técnicas analíticas que sean menos sensibles a estos artefactos (NMDS por ejemplo). Puede imputarse la media de las intensidades de cada compuesto o de las repeticiones, la mediana, la moda, una media ponderada, el valor más bajo de cada repetición, un valor que consideremos ruido aleatorio, valores aleatorios muy pequeños, etc. La elección de que valores imputar depende enteramente del criterio del analista.

```
#EJERCICIO: Imputar los ceros con la media de la intensidad de cada tratamiento
media_repeticiones<-apply(data_matriz,1,mean)

for(i in 1:length(media_repeticiones)){
    data_matriz[i,][data_matriz[i,]==0]<-media_repeticiones[i]
}

sqrt_data_matriz<-sqrt(as.matrix(data_matriz))
hist(sqrt_data_matriz,breaks=100)
```

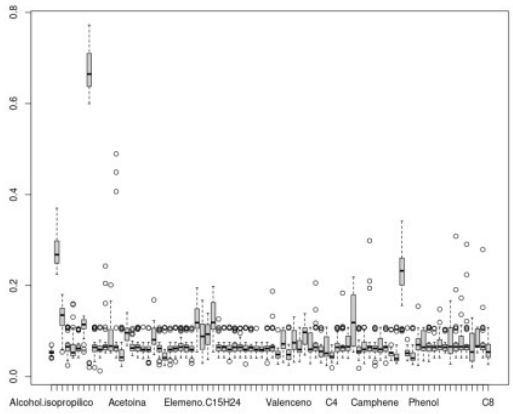


Imputar ceros y extraer raíz cuadrada

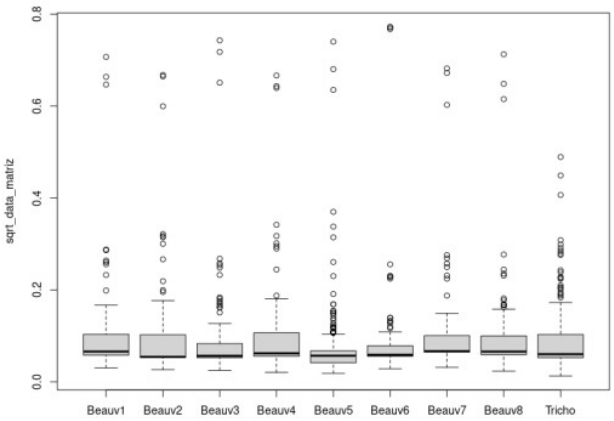
Normalización

Los datos generados en cromatografía a menudo exhiben variaciones significativas entre repeticiones debido a diversos factores, como las condiciones instrumentales, la sensibilidad de los detectores y el aumento del ruido de fondo. Estas variaciones pueden llevar a confusiones al interpretar diferencias en las abundancias como efectos reales. Por lo tanto, es justificable utilizar la normalización de los datos, que consiste en transformarlos a una escala común que preserve las variaciones relacionadas con los efectos de interés y reduzca las que puedan atribuirse a otros factores. Existen diversas opciones para llevar a cabo la normalización. La función scale() nos permite centrar los datos y ajustarlos a una escala de dispersión común, lo cual implica calcular la media de cada columna (es decir, la media de abundancia de cada compuesto) y dividir por la desviación estándar muestral (sd), logrando así una normalización a una variable estandarizada.

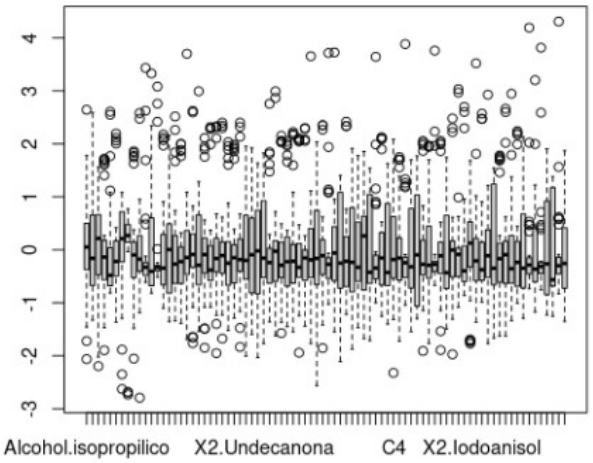
```
#Boxplot intensidades
boxplot(sqrt_data_matriz)
boxplot(sqrt_data_matriz~data$Tratamiento)

# Normalizar las variables
data_norm <- scale(sqrt_data_matriz,center = T,scale =T)
hist(data_norm,breaks = 100)
dim(data_norm)
boxplot(data_norm)
boxplot(data_norm~data$Tratamiento)
```



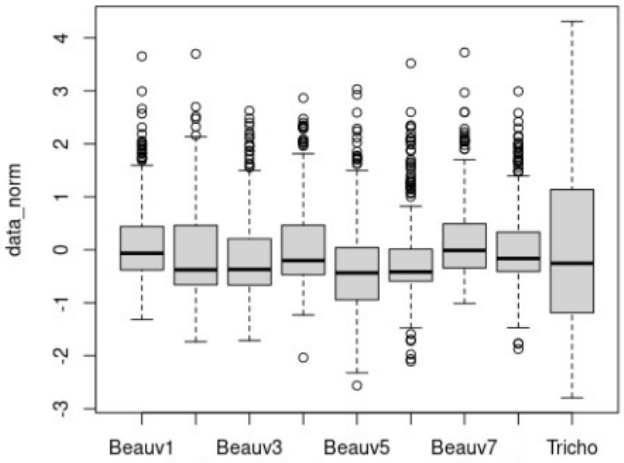
Intensidades entre compuestos



Intensidades entre repeticiones



Intensidades normalizada
entre repeticiones



Intensidades normalizadas
entre repeticiones

Técnicas de ordenación

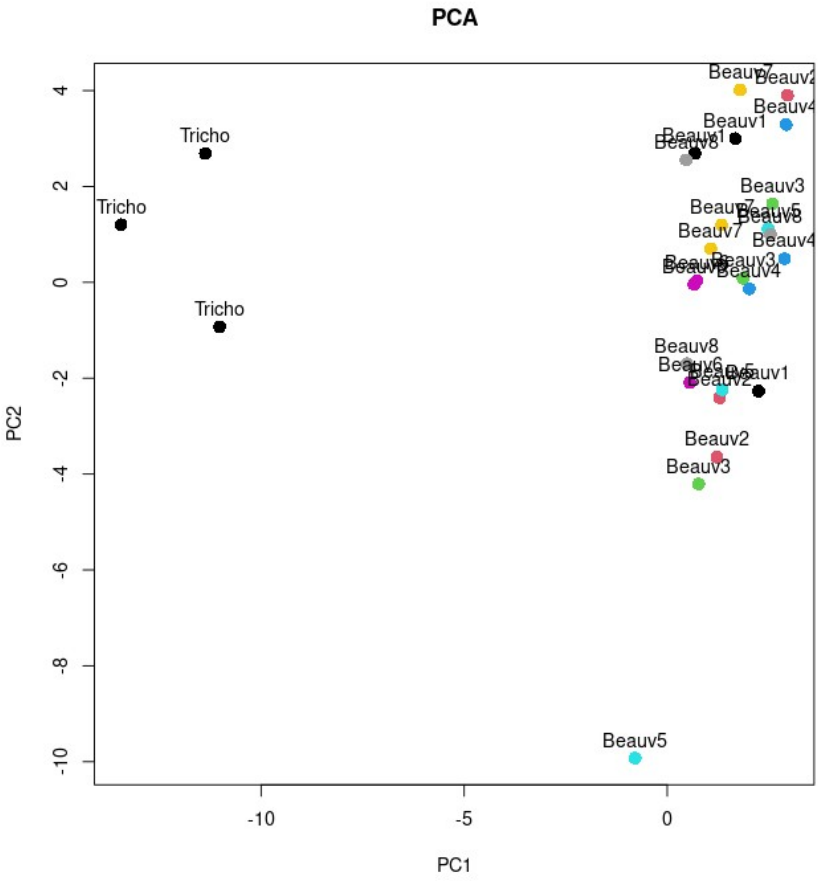
Las técnicas de ordenación son un conjunto de métodos multivariados que persiguen dos objetivos principales: reducir la dimensionalidad de una matriz de datos manteniendo la máxima información posible, y facilitar la visualización e interpretación de patrones subyacentes. En el contexto de las matrices de intensidad, cada compuesto representa una dimensión de los datos. Existen diversas técnicas de ordenación, entre las más populares se encuentran el análisis de componentes principales (PCA) y el análisis de escalamiento multidimensional no métrico (NMDS). Estas técnicas permiten transformar y reorganizar los datos en un espacio de menor dimensión (generalmente 2 o 3 dimensiones), donde las relaciones entre los compuestos y las muestras se mantienen, lo que facilita su análisis y visualización en busca de patrones y tendencias relevantes.

1. Análisis de componentes principales

El análisis de componentes principales (PCA, por sus siglas en inglés) es una técnica estadística utilizada para reducir la dimensionalidad de un conjunto de datos multivariados. El objetivo principal del PCA es transformar un conjunto de variables correlacionadas en un nuevo conjunto de variables no correlacionadas, llamadas componentes principales. Estos componentes principales se ordenan de manera que el primero captura la mayor varianza en los datos, el segundo captura la segunda mayor varianza y así sucesivamente. El PCA logra esto al encontrar combinaciones lineales de las variables originales que maximizan la varianza explicada. Geométricamente, esto puede entenderse como una reorientación de los datos y su proyección en un conjunto de dimensiones reducidas que es el plano de ordenación. Al reducir la dimensionalidad, el PCA facilita la visualización y comprensión de los patrones subyacentes en los datos, así como la identificación de las variables más relevantes en la explicación de la variabilidad de los datos. Además, el PCA también se utiliza para detectar colinealidad entre variables, identificar valores atípicos y realizar análisis de agrupamiento o clasificación.

Existen diversos supuestos en los datos sometidos a este análisis, siendo la normalidad de los datos y la linealidad del efecto de las variables en la respuesta observada los supuestos más destacados. Estos supuestos suelen ser rigurosos y no necesariamente se cumplen en todos los conjuntos de datos utilizados. Además, la técnica es propensa a generar artefactos debido a la presencia de ceros o valores extremos en la matriz, lo cual limita su aplicabilidad. No obstante, el análisis de componentes principales se caracteriza por ser de fácil interpretación biológica, lo que lo convierte en la opción preferida en una amplia gama de estudios.

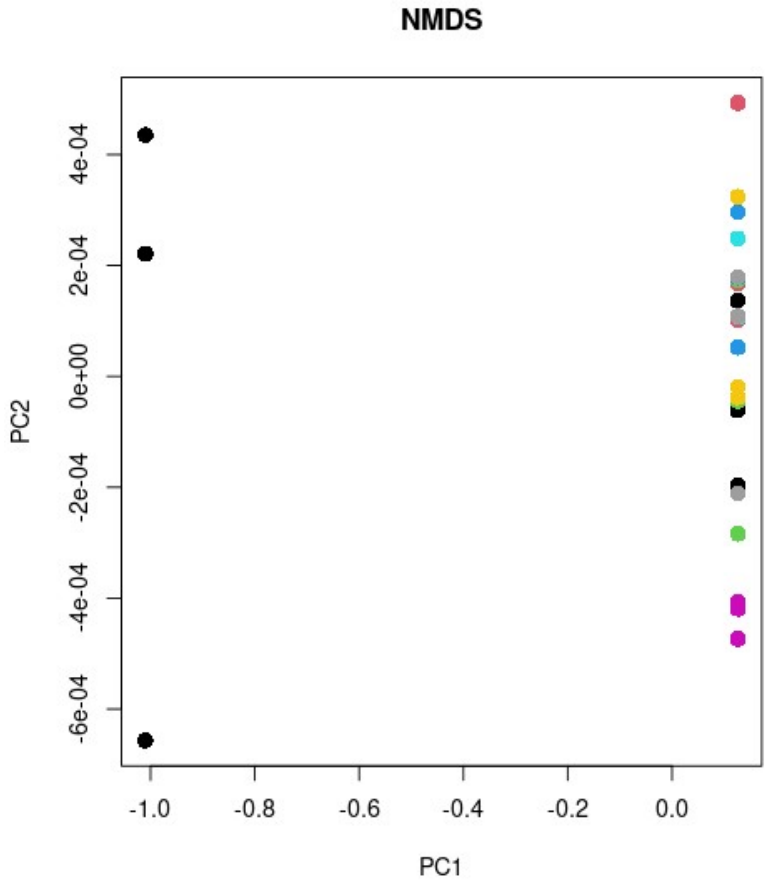
```
##ANÁLISIS DE COMPONENTES PRINCIPALES (PCA)##

# Realizar el análisis de componentes principales
pca <- prcomp(data_norm)
summary(pca)

# Graficar las coordenadas de las observaciones en las dos primeras componentes
#principales
colores<-as.factor(data$Tratamiento)
plot(pca$x[,1], pca$x[,2],
     main="PCA", xlab="PC1", ylab="PC2",col=as.numeric(colores),
     pch=16,cex=1.5)

# Graficar texto correspondiente
text(pca$x[,1], pca$x[,2],data$Tratamiento,pos=3)


#Tres componentes principales
library(rgl)
plot3d(x=pca$x[,1],y=pca$x[,2],z=pca$x[,3],
       col=as.numeric(colores),type="s",size=2)

text3d(x=pca$x[,1],y=pca$x[,2],z=pca$x[,3],
       data$Tratamiento,add=T,pos=3)
```



2. NMDS

El Escalamiento Multidimensional No Métrico (NMDS en inglés) es una técnica de ordenación que opera sobre las matrices de distancia de los datos. Su objetivo es generar una representación visual en dos o tres dimensiones que refleje las distancias o similitudes presentes en los datos originales. A diferencia del PCA, el NMDS no asume la normalidad de los datos ni la linealidad de la respuesta, lo que lo convierte en un método alternativo muy interesante en diversos casos. La matriz de distancias puede construirse utilizando diferentes medidas de similitud, como aquellas que capturan la presencia o ausencia de compuestos, acordes con el experimento en análisis.

```
###### NMDS (Non metric Multidimensional scaling) #######
library(vegan)
nmds_result<-metaMDS(data_matriz,distance = "bray",k = 3)
plot(nmds_result$points[,1], nmds_result$points[,2],
     main="PCA", xlab="PC1", ylab="PC2",col=as.numeric(colores),
     pch=16,cex=1.5)

plot3d(nmds_result$points[,1], nmds_result$points[,2],z=nmds_result$points[,3],
       col=as.numeric(colores),type="s",size=2)
```



Contraste de hipótesis

Los análisis de ordenación son ante todo métodos visuales para detectar patrones en los datos, que en nuestro contexto se interpretan como diferencias en los perfiles químicos entre las muestras. Sin embargo, no pueden ser tomadas como pruebas de contraste de hipótesis para evaluar que efectivamente lo observado son patrones reales o aleatorios. Para eso, existen otro grupo de métodos multivariados que pueden hacer esta tarea. Dos aproximaciones analíticas son el PERMANOVA y la estimación de k-medias.

1. PERMANOVA

El análisis de varianza pemutacional (PERMANOVA) es un método desarrollado en 2001 por Marti J. Anderson. Permite contrastar las diferencias en la dispersión y la ubicación de los centroides de los grupos experimentales. La hipótesis nula (Ho) es que los centroides y dispersión de los grupos son equivalentes. Para evaluar esta hipótesis, el PERMANOVA realiza permutaciones aleatorias entre las filas de los diferentes grupos experimentales. Es una técnica muy robusta que se emplea comúnmente y es ampliamente citada.

```
###### PERMANOVA ####################
factor_tratamiento<-as.factor(data$Tratamiento)
adonis(data_matriz~factor_tratamiento,dist="euclidean")
```

2. K-medias

Si bien el algoritmo de k-medias no es una prueba formal de contraste de hipótesis, puede darnos una buena idea de que tan robusta es la separación de los grupos que nosotros estamos observando en nuestro conjunto de datos. La idea es solicitar al programa que genere n grupos con la mejor separación posible y ver si estos coinciden con los grupos que a priori nosotros hemos definido que deben existir en nuestros datos.

```
###### k-medias ######################
library(ggplot2)
library(factoextra)
coordenadas<-data.frame(pca$x[,c(1,2)])
kmedias<-kmeans(coordenadas,centers = 2,nstart = 25)
fviz_cluster(kmedias,data=coordenadas)
plot(pca$x[,1], pca$x[,2],
     main="PCA", xlab="PC1", ylab="PC2", col=kmedias$cluster,
     pch=16,cex=1.5)
ordihull(pca,groups = kmedias$cluster)
```

Buscando potenciales biomarcadores

Utilizamos el término "biomarcadores", o más específicamente en nuestro contexto, "compuestos biomarcadores", para hacer referencia a aquellas moléculas cuyas abundancias se ven significativamente alteradas entre los diferentes tratamientos, y que además poseen un valor explicativo destacado para comprender las diferencias observadas en los perfiles químicos que estamos analizando. Los biomarcadores son indicativos de los cambios en los procesos metabólicos que se producen de una condición a otra, y por ende, juegan un papel fundamental en el estudio de estos perfiles químicos.

La búsqueda de posibles biomarcadores es uno de los objetivos principales al realizar un perfilado global, y el uso de técnicas multivariadas facilita enormemente este proceso. Para lograr este objetivo, existen diversas aproximaciones analíticas que podemos emplear. Una de ellas implica evaluar la contribución de cada compuesto en la separación de los grupos observados en los análisis de ordenación. Por ejemplo, el PCA proporciona información sobre la contribución de cada compuesto en la construcción de los componentes principales. Estos valores, conocidos como "loadings" y organizados en una matriz, nos ofrecen un criterio para comprender cómo cada compuesto contribuye a la separación de los grupos experimentales.

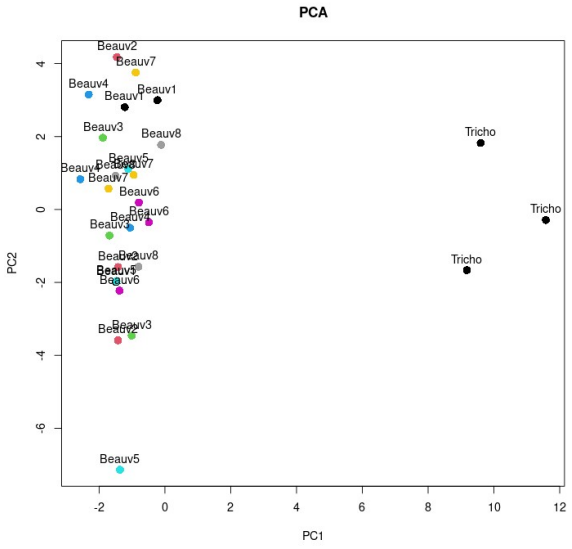
```
##### En búsqueda de biomarcadores ########
loadings_contribucion<-pca$rotation[,c(1,2)]
write.csv(loadings_contribucion,"loadings.csv")
##Podemos hacerlo manualmente...o crearnos una función para automatizar la tarea
```

Funciones en R

En tareas complejas, el uso de funciones en programas nos ayuda a simplificar el trabajo. Este es el caso al construir una matriz que contenga únicamente los 10 compuestos biomarcadores seleccionados debido a tener los valores absolutos más altos en los dos primeros componentes principales. Mientras que encontrarlos manualmente puede resultar tedioso en matrices de gran tamaño, esto puede realizarse rápidamente utilizando una función específicamente diseñada para esta tarea.

```
##loadbiomarc()
# FUNCION PARA SELECCIONAR BIOMARCADORES DE LOS LOADINGS DE UN PCA
# loadings_contribucion1=un dataframe con los loadings de cada compuesto,
#                        los compuestos en las filas y los componentes principales
#                        en columnas
# data_norm1=la matriz numérica usada para el PCA, los nombres de los compuestos
#           deben ser los mismos que en loadings_contribucion
# n_biom=numero de biomarcadores por componente principal, por defecto n_biom = 10

loadbiomarc<-function(loadings_contribucion1,data_norm1,n_biom = 10){ #Parámetros

  nombres = vector() #Creamos un vector vacio llamado nombres
     for(i   in   1:ncol(loadings_contribucion1)){   #Para   cada   i   (columna)   en
#loadings_contribucion
```

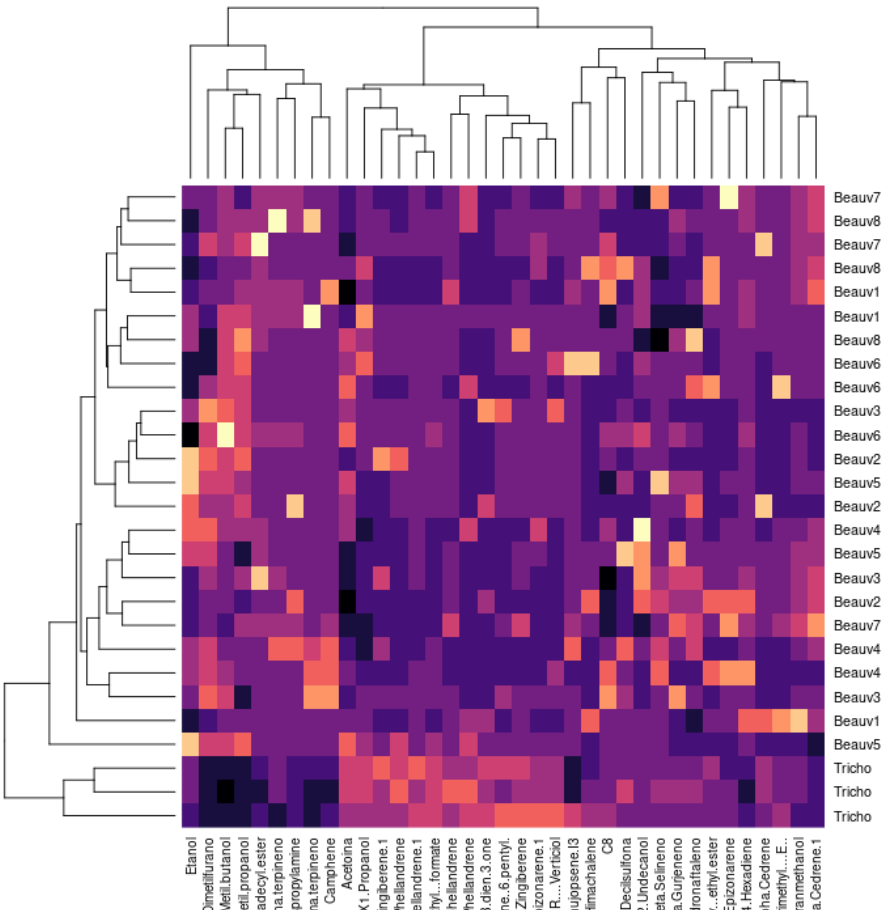
```
      contribucion = loadings_contribucion1[,c(i)] #Extraer la i-esima columna y
#ponerla en el vector contribucion
    orden = order(abs(contribucion),decreasing = T) #ordenar los valores absolutos
#de cada contribucion
     nombres = c(nombres,names(contribucion)[head(orden,n = n_biom)]) #guardar los
#nombres ordenados en en el vector nombre y mostrar los 10 primeros
  }
  #despues...
  if(any(duplicated(nombres)) == TRUE){ #Si alguno de los nombres está duplicado
     nombres = nombres[!duplicated(nombres)] #conservar solo uno de los nombres y
#crear un vector sin nombres duplicados
  } else { #y si no hay duplicados...
    nombres = nombres #usar el mismo vector de nombres
  }
  #despues...
   matriz_simplificada = data_norm1[,nombres] #extraer las columnas cuyos nombres
#coincidan  con los nombres sinn duplicar
    return(matriz_simplificada)  #...y  finalmente  regresar  la  matriz  de  las
#intensidades correspondientes
}

data_biomar<-loadbiomarc(loadings_contribucion,data_norm,n_biom = 20)
dim(data_biomar)
colnames(data_biomar)

#### Ahora reconstruimos el PCA unicamente con los biomarcadores
pca2 <- prcomp(data_biomar)
summary(pca2)
colores<-as.factor(data$Tratamiento)
plot(pca2$x[,1], pca2$x[,2],
     main="PCA", xlab="PC1", ylab="PC2",col=as.numeric(colores),
     pch=16,cex=1.5)
text(pca2$x[,1], pca2$x[,2],data$Tratamiento,pos=3)
plot3d(x=pca2$x[,1],y=pca2$x[,2],z=pca2$x[,3],
       col=as.numeric(colores),type="s",size=2)

text3d(x=pca2$x[,1],y=pca2$x[,2],z=pca2$x[,3],
       data$Tratamiento,add=T,pos=3)
```



Mapas de calor

Un mapa de calor es una representación visual que utiliza un código de colores para mostrar las intensidades de los compuestos entre diferentes tratamientos. Estas representaciones son altamente útiles para facilitar la interpretación de la información. Los mapas de calor suelen ir acompañados de dendrogramas, uno o dos, que indican la agrupación de filas y columnas en función de su similitud. Si las unidades comparten ramas cercanas, se consideran más similares entre sí. R ofrece una amplia gama de paquetes y funciones dedicados a crear y personalizar este tipo de gráficas. En nuestro caso, utilizaremos la función incorporada en la versión base del lenguaje para generar nuestros mapas de calor.

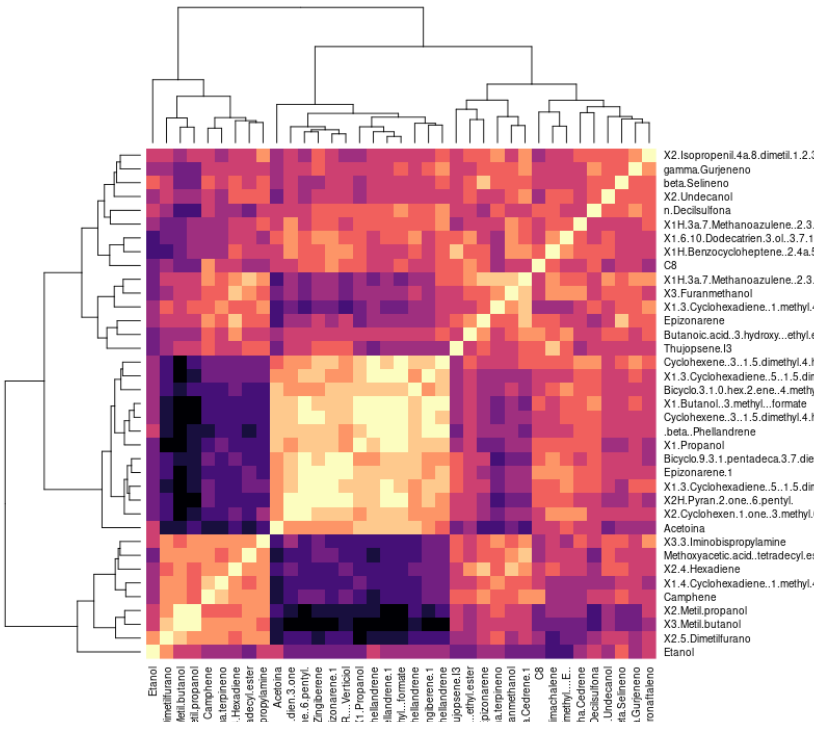
```
#### Mapa de calor de compuestos biomarcadores #####
class(data_biomar)
row.names(data_biomar) <- colores
dist_renglones<-vegdist(data_biomar,method = "euclidean")
dist_columnas<-vegdist(t(data_biomar),method = "euclidean")
heatmap(x = data_biomar,
        Rowv = dist_renglones,
        Colv = dist_columnas,
        col=magma(10))
#ver:          https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-
#viridis.html
```



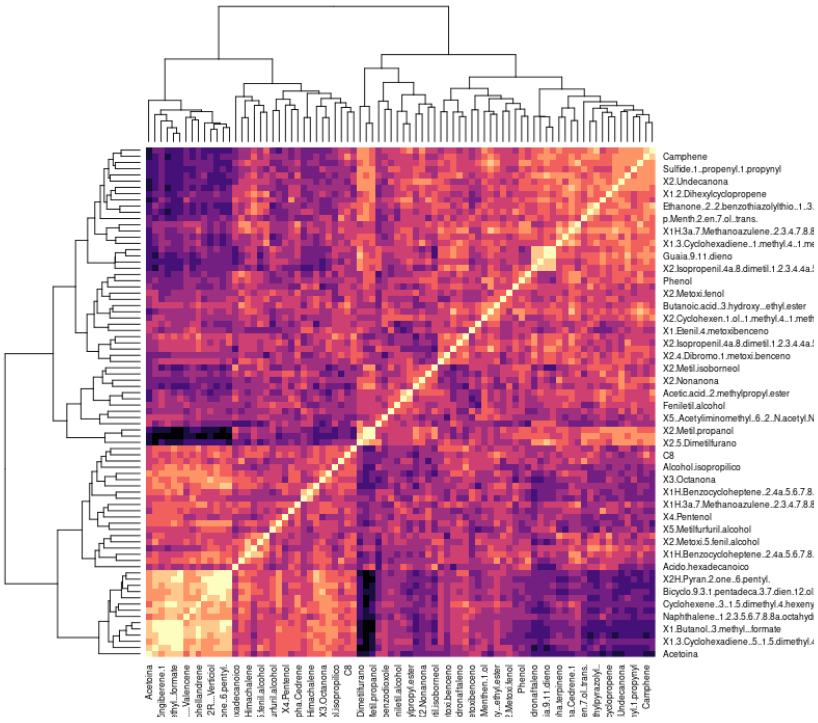
Correlogramas

Por su parte, un correlograma es una representación por un código de color de la correlación (generalmente correlación de Pearson) entre las variables. En casi todos los correlogramas se representa la diagonal principal, que corresponde al valor uno (por ser la correlación perfecta de una variable consigo misma). Estos diagramas nos permiten identificar grupos de compuestos altamente correlacionados, que pueden dar indicios muy relevantes de los procesos metabólicos subyacentes activos en las condiciones experimentales.

```
#### Correlogramas ####
corr_matriz<-cor(data_biomar)
dim(corr_matriz)
dist<-vegdist(corr_matriz,method = "euclidean")
heatmap(x = corr_matriz,
        Rowv = dist,
        Colv = dist,
        col=magma(10),
        symm = T)
```




```
corr_matriz<-cor(data_norm)
dim(corr_matriz)
dist<-vegdist(corr_matriz,method = "euclidean")
heatmap(x = corr_matriz,
        Rowv = dist,
        Colv = dist,
        col=magma(10),
        symm = T)
```



Machine learning en R. Random forest

Una de las principales motivaciones para utilizar la programación estadística en R es la amplia gama de métodos disponibles para analizar la información. En particular, en el campo de la metabolómica (y las "ómicas" en general), se ha incrementado el uso de técnicas de aprendizaje automático (machine learning) para estudiar perfiles complejos y obtener huellas o marcadores que brinden información sobre los procesos celulares subyacentes en diversos fenómenos. Uno de los algoritmos más utilizados es el de Random Forest, el cual construye un conjunto de árboles de decisión. Estos árboles se crean identificando las características que mejor permiten separar los grupos experimentales, buscando la máxima heterogeneidad entre los grupos y la máxima homogeneidad dentro de cada grupo.

```
### Bosques aleatorios (random forest) ###
library(randomForest)
modelo_rf <- randomForest(as.factor(data$Fuente) ~ .,
                                                    data = data_norm, ntree =
1000,keep.forest=F,importance=T)
contribucion <- importance(modelo_rf)
write.csv(contribucion,"random_forest.csv")

########## BOXPLOT ALGUNOS METABOLITOS ######################
tabla_norm<-as.data.frame(data_norm)
boxplot(tabla_norm$Acetoina~as.factor(data$Fuente))

##### FIN INTRODUCCION #####
```

Consideraciones finales

El lenguaje R es ampliamente utilizado en la exploración y análisis de datos en diversas áreas del conocimiento, incluyendo las ciencias químico-biológicas. En el ámbito de la cromatografía, se pueden encontrar numerosas implementaciones que permiten un análisis más profundo y robusto de los experimentos generados, además de aprovechar su potencial gráfico. En esta introducción, se han revisado algunos conceptos básicos sobre la estructura y sintaxis del lenguaje, así como una línea general de análisis para el GC-MS. No obstante, esta línea de análisis puede ser mejorada y ampliada en función de las necesidades específicas de cada estudio.