

▼ DGE

Differential Gene Expression" (**DGE**) es un análisis que busca identificar genes cuya expresión varía en diferentes condiciones o estados biológicos. Por lo general, implica comparar la cantidad de ARNm producido por un gen en diferentes muestras, como tejidos, células o condiciones experimentales. Este análisis cuenta de 4 pasos:

1. **Preprocesamiento de datos:** limpieza y normalización de datos brutos de ARNm
2. **Identificación de genes diferencialmente expresados.**
3. **Análisis estadístico:** para determinar la significancia de dichas diferencias.
4. **Validación** por otras técnicas

▼ 1. Preprocesamiento

```
#DGE
#Run a Differentially Expressed Genes Analysis using as input the count files
#produced in previous step

#Load libraries
suppressPackageStartupMessages({
  library(gplots, quietly = T)
  library(ggplot2, quietly = TRUE)
  library(pheatmap, quietly = TRUE)
  library(DESeq2, quietly = TRUE)
  library('org.Rn.eg.db', quietly = TRUE, character.only = TRUE)
  library(EnhancedVolcano, quietly = TRUE)
})
require("ggrepel", quietly = TRUE)
```

Primero se cargan las librerías que vamos a necesitar para correr el programa.

```
#Paths
workingD <- rstudioapi::getActiveDocumentContext()$path
setwd(dirname(workingD))
```

Este script obtiene la ruta del archivo del documento activo en RStudio y cambia el directorio de trabajo en R al directorio que contiene ese archivo. Dicho de manera más simple: es necesario que el script de R se encuentre en la misma carpeta que el archivo con el que quiero trabajar.

rstudioapi::getActiveDocumentContext()\$path utiliza la función `getActiveDocumentContext()` del paquete `rstudioapi` para obtener información sobre el documento activo en RStudio. `$path` es una forma de acceder al camino o la ruta del archivo del documento activo. Por lo tanto, `rstudioapi::getActiveDocumentContext()$path` guarda la ruta del archivo actual en la variable `workingD`.

setwd() cambia el directorio de trabajo en R al directorio que contiene el archivo del documento activo. **dirname()** es una función que toma una ruta de archivo y devuelve el directorio padre de esa ruta. Por lo tanto, `dirname(workingD)` devuelve la ruta del directorio que contiene el archivo del documento activo, y `setwd()` establece ese directorio como el directorio de trabajo actual en R.

```
#Input
#In order to select the right folder we need to specify which group we are going to work with
divide_sex <- F
males <- F

if (divide_sex) {
  if (males) {
    suffix <- '_males'
  } else {
    suffix <- '_females'
  }
} else {
  suffix <- ''
}

configFile <- paste0('Archivo_configuracion_mPFC', suffix, '.txt')
```

Este código se utiliza para configurar nombres de archivos y rutas de directorios basados en ciertas condiciones lógicas definidas por las variables.

divide_sex y **males** son variables booleanas (TRUE o FALSE) que controlan qué grupo se va a seleccionar para trabajar. Si divide_sex es verdadero (TRUE), se verifica si males es verdadero (TRUE) o falso (FALSE) para determinar el sufijo que se agregará a los nombres de los archivos. Dependiendo de los valores de estas variables booleanas, se define el sufijo que se utilizará para los nombres de los archivos.

Dicho con otras palabras, tengo tres opciones:

1. divide_sex F --> los resultados no se separan por sexos.
2. divide_sex T y males F --> se muestran los resultados de hembras.
3. divide_sex T y males T --> se muestran los resultados de machos.

configFile construye el nombre del archivo de configuración basado en suffix, que se determinó previamente según las condiciones.

```
#Outputs
resD <- paste0('DEG_results', 'suffix', '_sinFamilia/')
dir.create(resD)

rawCountsF <- paste0(resD, "counts_raw.tsv")
normCountsF <- paste0(resD, "counts_normalized.tsv")
PCAF <- paste0(resD, "PCA.jpeg")
distancesF <- paste0(resD, "distances.jpeg")
dispersionF <- paste0(resD, "dispersion.tiff")
MAplotF <- paste0(resD, "maplot.jpeg")
genesTSV <- paste0(resD, "all_genes.csv")
sigTSV <- paste0(resD, "sig_pval.csv")
sigPCAF <- paste0(resD, "PCA_sig.jpeg")
volcanoF <- paste0(resD, "volcanoPlot.jpeg")
heatmapF <- paste0(resD, "heatmap.jpeg")
DESEqResultsF <- paste0(resD, 'deseq_objects.RData')
```

Esta parte del script está organizando un conjunto de rutas de archivos para guardar y acceder fácilmente a los resultados generados por el análisis, manteniendo todo ordenado dentro de un directorio principal (resD).

resD es el directorio principal (DEG_results seguido de suffix y _sinFamilia/) donde se almacenarán todos los archivos resultantes.

dir.create(resD) genera la carpeta si no existe.

A continuación hay una serie de rutas de archivos (**rawCountsF**, **normCountsF**, **PCAF**, etc.) que apuntan a archivos específicos dentro de ese directorio. Por ejemplo, rawCountsF y normCountsF son rutas que llevan a archivos de recuentos brutos y normalizados, mientras que PCAF, distancesF, dispersionF, MAplotF, genesTSV, sigTSV, sigPCAF, volcanoF, heatmapF, y DESEqResultsF son rutas que conducen a diferentes tipos de gráficos, archivos de datos o resultados de tu análisis, todos almacenados en el directorio resD. Cada ruta de archivo se crea concatenando el directorio resD con el nombre específico del archivo.

```
#Parameters
cutoff <- 0.05 #significance p value adjusted
FCthres <- NULL #fold change threshold to consider in graphs
```

Este fragmento de código está estableciendo parámetros que se usarán en el análisis

cutoff establece un umbral de significancia estadística para los valores de p-valor ajustados. En este caso, se fija en 0.05. Esto significa que durante el análisis, se considerarán como significativos los resultados cuyos valores de p-valor ajustados sean menores o iguales a 0.05. Este valor es comúnmente utilizado como estándar para determinar la significancia estadística en muchos estudios científicos.

FCthres inicializado en NULL, significa que en este punto no se ha establecido un umbral específico para el cambio en la expresión génica (fold change threshold). En algunos análisis de expresión génica diferencial, se utiliza un umbral de fold change para considerar significativas las diferencias en la expresión entre grupos.

```
#Functions: italics for genes in heatmap
make_italics <- function(x){
  as.expression(lapply(x, function(y) bquote(italic.(y))))
}
```

Esta función se encarga de convertir una lista de texto en expresiones que mostrarán ese texto en cursiva en gráficos o visualizaciones.

make_italics es el nombre de la función que se está creando. Utiliza la función **function(x){...}** para definir lo que hace esta función. Toma un argumento x, que se espera sea una lista de texto o caracteres. **lapply** se utiliza para aplicar una función (en este caso, una función anónima definida con **function(y) ...**) a cada elemento de la lista x. La función anónima (**function(y) bquote(italic.(y))**) toma cada elemento y de la lista y lo convierte en una expresión en cursiva usando **italic()** y **bquote()** para formatear el texto como cursiva.

```
#We create a function so that when the name of the gene is NA it gives back de ensemble ID
rellenar_nombres <- function(data_frame) {
  # Obtener los rownames del data frame
  row_names <- rownames(data_frame)

  # Obtener la columna de nombres de genes
  genes_col <- data_frame$symbol

  # Reemplazar los valores NA con los rownames correspondientes
  genes_col[is.na(genes_col)] <- row_names[is.na(genes_col)]

  # Actualizar la columna en el data frame
  data_frame$symbol <- genes_col

  return(data_frame)
}
```

Este fragmento de código crea una función llamada `rellenar_nombres` en R que se utiliza para reemplazar los valores "NA" en una columna específica de un marco de datos con los nombres de las filas correspondientes.

`rellenar_nombres` es el nombre de la función. Toma como argumento un `data_frame`, que se espera sea un marco de datos de R.

`row_names <- rownames(data_frame)` obtiene los nombres de las filas del marco de datos y los almacena en `row_names`.

`genes_col <- data_frame$symbol` extrae la columna llamada 'symbol' del marco de datos y la almacena en `genes_col`.

`genes_col[is.na(genes_col)] <- row_names[is.na(genes_col)]` reemplaza los valores NA (valores faltantes) en la columna 'symbol' con los nombres correspondientes de las filas en caso de que haya valores NA en esa columna.

`data_frame$symbol <- genes_col` actualiza la columna 'symbol' en el marco de datos con los cambios realizados.

`return(data_frame)` devuelve el marco de datos actualizado con los valores "NA" en la columna 'symbol' reemplazados por los nombres de las filas correspondientes cuando sea necesario.

```
#Load variables of each sample
sampleTable <- read.table(configFile, header=TRUE
                           ,colClasses= c('factor','character','factor',
                                         'factor','factor')
)
```

Este código carga datos desde un archivo externo a una variable llamada `sampleTable`. El comando `read.table()` se utiliza para leer datos tabulares, como un archivo CSV o un archivo de texto con formato similar, y cargarlos en R. En este caso, un archivo cuyo nombre de archivo está almacenado en la variable `configFile`.

La función `read.table()` toma varios argumentos, como el **nombre del archivo a leer** (`configFile` en este caso), `header` que indica si la **primera fila del archivo contiene nombres de columna**, y `colClasses` que **especifica las clases de columna** para cada columna en el conjunto de datos. Aquí, las primeras cinco columnas deben ser tratadas como factor, carácter, factor, factor y factor, respectivamente.

Configfile contiene: número de rata, counts, grupo, sexo y familia.

```
#Convert the counts into a DeSeq DataSet object
if (divide_sex) {
  data <- DESeqDataSetFromHTSeqCount(sampleTable, directory=".",
                                      design = ~ Grupo)
} else {
  data <- DESeqDataSetFromHTSeqCount(sampleTable, directory=".", design = ~ Sexo + Grupo)
}
```

La función `DESeqDataSetFromHTSeqCount` del paquete `DESeq2`, es una herramienta que se utiliza para convertir los recuentos de expresión génica (generalmente provenientes de datos de secuenciación de ARN, en este caso nuestra `sampleTable`) en un tipo específico de objeto llamado `DataSet` (que almacena la información de recuentos y metadatos asociados con las muestras experimentales) compatible con `DESeq2` para realizar análisis de expresión génica diferencial.

El script condicionalmente asigna valores al objeto `data` basado en la variable `divide_sex`. De forma que si `divide_sex` es `TRUE` utiliza el modelo de diseño `~ Grupo` para crear el objeto `DataSet`. Esto implica que el análisis se llevará a cabo considerando solo el factor de agrupación llamado "**Grupo**". Si `divide_sex` es `FALSE` utiliza el modelo de diseño `~ Sexo + Grupo` para crear el objeto `DataSet`. Esto implica que el análisis considerará dos factores de agrupación: "**Sexo**" y "**Grupo**". En ambos casos, `sampleTable` es el argumento que proporciona información sobre las muestras y sus características.

`directory="."` especifica que los archivos de recuento están en el directorio actual.

```
#Pre-filtering: clean some of the noise in the counts
keep <- rowSums(counts(data)) >= 10
data <- data[keep,]
#With this filter, the object goes from 30562 elements to 19538 elements
```

Está realizando un prefiltrado en los datos de expresión génica contenidos en el objeto data (creado previamente mediante la función DESeqDataSetFromHTSeqCount).

La línea **keep <- rowSums(counts(data)) >= 10** está creando un vector booleano (keep) que indica que se retendrán solo las filas (genes) cuyos recuentos totales a través de todas las muestras sean al menos 10.

La función **rowSums(counts(data))** suma los recuentos de expresión génica por fila (es decir, por gen) en la matriz de recuentos contenida en data. Luego, la comparación ≥ 10 devuelve un vector de verdadero/falso indicando si la suma de recuentos de cada gen es mayor o igual a 10.

Posteriormente, **data <- data[keep,]** filtra el objeto data manteniendo solo las filas (genes) para las cuales keep es verdadero (es decir, donde la suma de recuentos es mayor o igual a 10). Por lo tanto, se eliminan los genes que tienen recuentos totales menores a 10 en todas las muestras.

```
# DESeq: original DESEQ() function doesn't allow to adjust number of iterations
dds <- estimateSizeFactors(data)
dds <- estimateDispersions(dds)
dds <- nbinomWaldTest(dds, maxit = 10000)
```

dds <- estimateSizeFactors(data) calcula los factores de tamaño (size factors) para normalizar los datos de expresión génica en data. Los factores de tamaño son utilizados para corregir las diferencias sistemáticas en la profundidad de secuenciación entre las muestras. Este cálculo se utiliza para encontrar los "factores de tamaño" que ayudarán a normalizar los datos de expresión génica. La normalización es necesaria porque las muestras pueden tener diferencias en la profundidad de secuenciación, lo que puede afectar a los resultados. Los "factores de tamaño" ajustan o corrigen estas diferencias sistemáticas, para que puedas comparar de manera más precisa la expresión génica entre las muestras.

dds <- estimateDispersions(dds) estima las dispersiones de los datos, que capturan la variabilidad biológica y técnica en los datos de expresión génica. Las dispersiones son fundamentales para el modelo estadístico que se utiliza para identificar genes diferencialmente expresados.

dds <- nbinomWaldTest(dds, maxit = 10000) realiza el test de Wald utilizando una distribución negativa binomial. El **test de Wald** es un tipo de prueba estadística utilizada para evaluar si los parámetros de un modelo son significativamente diferentes de ciertos valores. En el contexto de modelos estadísticos, como modelos de regresión, el test de Wald se utiliza para evaluar la significancia de los coeficientes asociados a las variables. La **distribución binomial negativa** es un tipo de distribución de probabilidad que describe el número de ensayos necesarios para obtener un número fijo de éxitos en una secuencia de ensayos independientes, donde cada ensayo tiene la misma probabilidad de éxito. La función **nbinomWaldTest** lleva a cabo las pruebas de hipótesis para determinar la significancia estadística de las diferencias en la expresión génica entre las condiciones experimentales. El parámetro **maxit = 10000** establece el número máximo de iteraciones permitidas para el ajuste del modelo.

```
#Save raw counts of all the samples in a single data frame to save it, as well as normalized counts
dds_raw <- counts(dds, normalized=FALSE)
dds_normalized <- counts(dds, normalized=TRUE)

#save file with counts and normalized counts
write.table(dds_raw, file=rawCountsF, quote=FALSE,
            sep = "\t", col.names=NA)
write.table(dds_normalized, file=normCountsF, quote=FALSE,
            sep = "\t", col.names=NA)
#Save for the GSEA
save(dds, file = DESeqResultsF)
```

Este script se centra en la extracción y almacenamiento de los recuentos de expresión génica, tanto en su forma cruda como normalizada, así como la preservación de los resultados obtenidos con DESeq2 para su posterior uso en GSEA (Gene Set Enrichment Analysis).

dds_raw <- counts(dds, normalized=FALSE) extrae los recuentos de expresión génica en su forma cruda (sin normalizar) del objeto dds, que contiene los resultados del análisis realizado con DESeq2.

dds_normalized <- counts(dds, normalized=TRUE) extrae los recuentos de expresión génica normalizados del objeto dds.

write.table(dds_raw, file=rawCountsF, quote=FALSE, sep="\t", col.names=NA) guarda los recuentos de expresión génica crudos en un archivo. **file=rawCountsF** especifica el nombre del archivo donde se guardarán los recuentos crudos.

write.table(dds_normalized, file=normCountsF, quote=FALSE, sep="\t", col.names=NA) guarda los recuentos de expresión génica normalizados en otro archivo, **file=normCountsF**.

Estos pasos permiten guardar los datos en formato de tabla/tabulado, sin comillas en los valores, utilizando tabulaciones como separadores y sin nombres de columnas.

`save(dds, file = DESeqResultsF)` guarda el objeto dds (conteniendo los resultados del análisis DESeq2) en un archivo específico identificado por DESeqResultsF. Esto es útil para utilizar estos resultados en análisis posteriores, como GSEA (Gene Set Enrichment Analysis).

2. Identificación de genes diferencialmente expresados

✓ Plots

✓ PCA

```
#PCA: blind must be FALSE to take into account batch effect
#Normalization
vst <- varianceStabilizingTransformation(dds, blind = FALSE)
mat <- assay(vst)
if (!divide_sex) {
  mat <- limma::removeBatchEffect(mat, batch=vst$Sexo,
                                    group=vst$Grupo) }
#Si no es dividido por sexo aparece la covariable del sexo
assay(vst) <- mat

# Para el análisis de hembras o machos por separado
jpeg(filename = PCAF, width=900, height=900, quality=300)
pca <- plotPCA(vst, intgroup = "Grupo")
title <- "Principal Components Plot"
pca + ggtitle(title) +
  geom_point(size = 6) +
  theme(plot.title = element_text(size=40, hjust = 0.5, face = "bold"), axis.title=element_text(size=20),
        legend.text=element_text(size=15),legend.title=element_text(size=15)) +
  geom_text_repel(aes(label=colnames(vst)), size=5, point.padding = 0.6)
dev.off()

# Para el análisis de hembras y machos juntos
jpeg(filename = PCAF, width=900, height=900, quality=300)
pca <- plotPCA(vst, intgroup = c("Grupo", "Sexo"), returnData = TRUE)
title <- "Principal Components Plot"
ggplot(pca, aes(x=PC1, y=PC2, color = Grupo, shape= Sexo)) +
  ggtitle(title) +
  geom_point(size = 6, aes(fill=Grupo)) + scale_shape_manual(values=c(21,22)) +
  theme(plot.title = element_text(size=40, hjust = 0.5, face = "bold"), axis.title=element_text(size=20),
        legend.text=element_text(size=15),legend.title=element_text(size=15)) +
  geom_text_repel(aes(label=colnames(vst)), size=5, point.padding = 0.6)
dev.off()
```

Este fragmento de código en R se centra en la realización de un análisis de componentes principales (PCA) utilizando DESeq2 y limma para explorar la variación en los datos de expresión génica y considerar posibles efectos de lote (batch effects).

El Análisis de Componentes Principales (PCA) es una técnica en estadísticas y análisis de datos que se utiliza para simplificar y resumir la información contenida en un conjunto grande de variables. Se utiliza como herramienta de reducción de dimensionalidad, lo que facilita el análisis y la interpretación de conjuntos de datos complejos.

"Linear Models for Microarray and RNA-Seq Data" **limma** es una librería de software utilizada en el análisis de datos de expresión génica. Su funciones son: análisis diferencial de expresión, modelos lineales empíricos bayesianos, corrección de múltiples comparaciones, normalización y procesamiento de datos y visualización de resultados.

`vst <- varianceStabilizingTransformation(dds, blind = FALSE)` realiza una transformación de estabilización de varianza en los datos utilizando la función varianceStabilizingTransformation de DESeq2. Dicho con otras palabras: Lo que hace es transformar los datos para que la varianza entre las muestras sea más constante, lo que es útil en análisis estadísticos para asegurar que los datos cumplen con ciertos supuestos. Al establecer `blind = FALSE`, se considera explícitamente la información de agrupación (por ejemplo, Sexo y Grupo) para modelar y estabilizar la varianza, lo que puede ayudar a mitigar los efectos de lote.

`mat <- assay(vst)` extrae la matriz de recuentos transformada y estabilizada de la variable `vst`.

`if (!divide_sex) { ... }` aplica una corrección de efectos de lote solo si `divide_sex` es falso. Utiliza la función `limma::removeBatchEffect` para eliminar los efectos de lote, identificados por la variable `Sexo` en el conjunto de datos.

`assay(vst) <- mat` actualiza la matriz de recuentos en `vst` con la matriz `mat` después de aplicar las correcciones.

`jpeg(filename = PCAF, width=900, height=900, quality=300)` configura la salida de un gráfico de PCA a un archivo JPEG con las dimensiones y la calidad especificadas.

`pca <- plotPCA(vst, intgroup = "Grupo")` realiza el gráfico de componentes principales utilizando la función `plotPCA` de DESeq2, donde el parámetro `intgroup = "Grupo"` colorea los puntos por grupo en el PCA.

`pca <- plotPCA(vst, intgroup = c("Grupo", "Sexo")` realiza el gráfico de componentes principales utilizando la función `plotPCA` de `DESeq2`, donde el parámetro `intgroup = "Grupo"` colorea los puntos por grupo en el PCA y distingue machos de hembras por distintas formas, siendo las hembras círculos y los machos cuadrados.

Se agregan detalles visuales al gráfico, como título, tamaño de punto, etiquetas y ajustes de estilo utilizando la librería `ggplot2`.

`dev.off()` finaliza la grabación del gráfico en el archivo JPEG.

▼ Distancia entre muestras

```
#Plot distances between samples

distRL <- dist(t(mat))
distMat <- as.matrix(distRL)
hc <- hclust(distRL)
hmcol <- colorRampPalette(c("white", "blue"))(299)
title <- "Distances matrix"
jpeg(filename = distancesF, width=900, height=900, quality=300)
heatmap.2(distMat, Rowv=as.dendrogram(hc), symm=TRUE, trace="none", col=rev(hmcol),
           margin=c(10, 6), main=title, key.title=NA)
invisible(dev.off())
```

Este script R está generando un gráfico de calor que visualiza las distancias entre las muestras en un conjunto de datos de expresión génica.

`distRL <- dist(t(mat))` está calculando las distancias euclidianas entre las muestras en un conjunto de datos 'mat' (en términos más sencillos, es una forma de medir la longitud del "camino más corto" entre dos puntos en un espacio bidimensional o tridimensional). El resultado `distRL` es un objeto de distancia.

`distMat <- as.matrix(distRL)` convierte el objeto de distancia en una matriz, lo que facilita su manipulación y visualización.

`hc <- hclust(distRL)` realiza un clustering jerárquico sobre las distancias calculadas. Este paso agrupa las muestras que son más similares entre sí.

`hmcol <- colorRampPalette(c("white", "blue"))(299)` establece una paleta de colores que va desde blanco hasta azul con 299 pasos. Este conjunto de colores se utilizará en el gráfico de calor.

`title <- "Distances matrix"` establece el título del gráfico de calor como "Matriz de Distancias".

`jpeg(filename = distancesF, width=900, height=900, quality=300)` inicia un dispositivo gráfico para guardar el gráfico en formato JPEG con un ancho y alto específicos y una calidad de 300 ppp (píxeles por pulgada).

`heatmap.2(...)` crea el gráfico de calor utilizando la función `heatmap.2` con parámetros específicos. El gráfico visualiza la matriz de distancias con las muestras agrupadas por el clustering jerárquico. La paleta de colores `hmcol` se utiliza para resaltar las diferencias en las distancias. Se elimina el rastreo (`trace="none"`), y se establecen márgenes y otros parámetros estéticos.

`invisible(dev.off())` guarda el gráfico en el archivo especificado por `distancesF` y cierra el dispositivo gráfico JPEG.

▼ Dispersión entre muestras

```
#Plot gene dispersion
tiff(filename = dispersionF, units="in", width=5, height=5, res=300)
title <- "Per-gene dispersion estimates"
plotDispEsts(dds, main=title)
invisible(dev.off())
```

Este script produce un gráfico que muestra las estimaciones de dispersión para cada gen en el conjunto de datos de expresión génica. Estas estimaciones de dispersión son importantes en el análisis de expresión génica para comprender cuánto varía la expresión de un gen entre las muestras.

`tiff(filename = dispersionF, units="in", width=5, height=5, res=300)` abre un dispositivo gráfico TIFF para guardar el gráfico en un archivo.

`dispersionF` contiene la ruta y el nombre del archivo donde se guardará el gráfico. Se configuran algunas propiedades del archivo TIFF, como la unidad de medida (pulgadas), el ancho y la altura del gráfico (5 pulgadas por 5 pulgadas) y la resolución (300 píxeles por pulgada).

`title <- "Per-gene dispersion estimates"` establece el título del gráfico como "Estimaciones de Dispensión por Gen".

`plotDispEsts(dds, main=title)` llama a la función `plotDispEsts` con el conjunto de datos `dds` y el título especificado. Esta función generará el gráfico de las estimaciones de dispersión de los genes.

`invisible(dev.off())` cierra el dispositivo gráfico actual.

▼ Parámetros necesarios para MAplot

```
#Get factor levels
levels <- unique(sampleTable$Grupo)
l1 <- toString(levels[2]) #reference level has to be control
l2 <- toString(levels[1])
suffix <- paste(l1, l2, sep="_vs_")

#Get results
res <- results(dds, contrast=c("Grupo", l2, l1))
res$log2FoldChange <- 2^res$log2FoldChange #have actual fold change
res <- res[, colnames(res)[c(1,7,2:6)]] # order columns
```

La primera parte del script está extrayendo los niveles únicos de un factor en un conjunto de datos, creando cadenas de texto para representar dos niveles específicos, y luego combinando esos niveles para formar un sufijo que puede usarse en análisis posteriores para etiquetar o identificar comparaciones específicas entre esos niveles del factor.

levels <- unique(sampleTable\$Grupo) extrae los niveles únicos del factor "Grupo" en el DataFrame llamado sampleTable. Cada nivel representa una categoría o grupo en el factor.

l1 <- toString(levels[2]) asigna a la variable l1 el segundo nivel del factor "Grupo" convertido a formato de cadena. Este nivel se asume como la categoría de referencia o "control".

l2 <- toString(levels[1]) asigna a la variable l2 el primer nivel del factor "Grupo" convertido a formato de cadena.

suffix <- paste(l1, l2, sep="vs") crea un sufijo combinando los dos niveles con el separador "vs". El objetivo es crear un sufijo que refleje la comparación entre los dos niveles del factor. Como los niveles son "Control" y "Modelo de rata esquizofrénica", el sufijo será "Control_vs_Modelo".

La segunda parte del script ajusta el conjunto de resultados para incluir la columna "FoldChange" con los cambios reales en la expresión génica, y reorganiza el orden de las columnas en el conjunto de resultados.

res <- results(dds, contrast=c("Grupo", l2, l1)) utiliza la función results de DESeq2 para obtener los resultados del análisis de expresión génica diferencial. El contraste se define utilizando los niveles de un factor llamado "Grupo", donde l2 y l1 son los dos niveles específicos que se comparan. Esto significa que se están comparando las muestras asociadas a los niveles l2 y l1 del factor "Grupo".

res\$log2FoldChange <- 2^res\$log2FoldChange calcula el cambio en el doble de la escala logarítmica (fold change) a partir del resultado del logaritmo en base 2 del cambio. Esta línea añade una nueva columna llamada "FoldChange" al conjunto de resultados, que representa el cambio real (no logarítmico) en la expresión génica entre los dos niveles.

res <- res[, colnames(res)[c(1,7,2:6)]] reorganiza el orden de las columnas en el conjunto de resultados. La expresión colnames(res)[c(1,7,2:6)] selecciona las columnas en un orden específico (1, 7 y luego las demás de 2 a 6), y luego se reorganiza el conjunto de resultados según ese orden.

Maplot

```
# Maplot
jpeg(filename = MAplotF, width=900, height=900, quality=300)
title <- paste("MA-plot", suffix, sep=" ")
plotMA(res, alpha=cutoff, main=title, colSig = 'red', cex = 1.2)
invisible(dev.off())
```

Un **gráfico MA** (diferencia frente a media) es una herramienta visual comúnmente utilizada para examinar la variación entre dos conjuntos de datos, como dos condiciones experimentales. En el **eje x** se representa la media de la expresión génica en escala logarítmica. En el **eje y** la diferencia en la expresión génica entre dos condiciones en escala logarítmica.

Cada punto en el gráfico MA representa un gen específico. La posición del punto en el eje x indica cuán expresado está el gen en promedio, mientras que la posición en el eje y indica la diferencia en la expresión entre las dos condiciones.

Los genes que están diferencialmente expresados tienden a estar ubicados más arriba o más abajo en el gráfico, dependiendo de si su expresión es mayor en una condición en comparación con la otra. Los puntos en la parte superior o inferior del gráfico, que se alejan del eje x, son aquellos que muestran diferencias significativas en expresión.

jpeg(filename = MAplotF, width=900, height=900, quality=300) inicia un dispositivo gráfico JPEG para guardar el gráfico. MAplotF contiene la ruta y el nombre del archivo donde se guardará el gráfico MA. Se configuran propiedades como el ancho, el alto y la calidad de la imagen.

title <- paste("MA-plot", suffix, sep=" ") crea un título para el gráfico MA combinando el texto "MA-plot" con el sufijo generado anteriormente (suffix). Se utiliza un espacio como separador.

plotMA(res, alpha=cutoff, main=title, colSig = 'red', cex = 1.2) crea el gráfico MA utilizando la función plotMA de DESeq2. Los resultados del análisis de expresión génica (res) se utilizan para generar el gráfico. Se especifica el nivel de significancia (alpha) y se establece el título del gráfico (main). Los genes diferencialmente expresados que son significativos se resaltan en rojo (colSig = 'red'), y cex = 1.2 ajusta el tamaño de los puntos en el gráfico.

invisible(dev.off()) cierra el dispositivo gráfico JPEG actual.

✓ Parámetros necesarios para Volcano plot

```
#Add annotation to results
symbol <- mapIds(get('org.Rn.eg.db'), keys=row.names(res), column="SYMBOL",
  keytype="ENSEMBL", multiVals="first") #to obtain gene symbols

description <- mapIds(get('org.Rn.eg.db'), keys=row.names(res),
  column="GENENAME", keytype="ENSEMBL",
  multiVals="first") #to obtain description

#Bind dataframe symbol and res and create a new one for description
res <- cbind(symbol, res)
res$description <- description

write.table(res, file=genesTSV, quote=FALSE, sep=",", col.names=NA)
sig_pval <- subset(res, res$pvalue < cutoff) #Select only the roles with a p-value smaller than cutoff
write.table (sig_pval, file=sigTSV, quote=FALSE, sep=",", col.names=NA)
```

La primera parte de este script está agregando anotaciones (símbolos y descripciones) a los resultados de un análisis de expresión génica diferencial.

symbol <- mapIds(get('org.Rn.eg.db'), keys=row.names(res), column="SYMBOL", keytype="ENSEMBL", multiVals="first") utiliza la función mapIds del paquete org.Rn.eg.db para obtener los símbolos de genes correspondientes a las identificaciones ENSEMBL en los resultados (res). La columna "SYMBOL" se utiliza para mapear las identificaciones ENSEMBL a los símbolos de genes, y multiVals="first" indica que si hay múltiples símbolos para una misma identificación ENSEMBL, se debe seleccionar el primero.

description <- mapIds(get('org.Rn.eg.db'), keys=row.names(res), column="GENENAME", keytype="ENSEMBL", multiVals="first") similar a la línea anterior, esta vez obtiene descripciones de genes. La columna "GENENAME" se utiliza para mapear las identificaciones ENSEMBL a las descripciones de genes.

La segunda parte del script agrega información de símbolos de genes y descripciones al conjunto de resultados, guarda el conjunto de resultados actualizado en un archivo de texto, y crea un subconjunto de genes significativamente diferencialmente expresados para guardarlo en otro archivo de texto

res <- cbind(symbol, res) combina (cbind) las columnas de símbolos de genes (symbol) con los resultados del análisis de expresión génica diferencial (res). De esta manera, se agrega la información de los símbolos de genes al conjunto de resultados.

res\$description <- description crea una nueva columna en el conjunto de resultados (res) llamada "description" y le asigna los valores de la variable description, que contiene las descripciones de los genes.

write.table(res, file=genesTSV, quote=FALSE, sep=",", col.names=NA) escribe el conjunto de resultados actualizado en un archivo de texto (TSV). genesTSV es probablemente la ruta y el nombre del archivo donde se guardará el archivo. Se utiliza una coma como separador, y quote=FALSE indica que no se deben incluir comillas alrededor de los datos. col.names=NA significa que no se deben incluir nombres de columna en el archivo.

sig_pval <- subset(res, res\$pvalue < cutoff) crea un nuevo conjunto de datos llamado sig_pval que contiene solo las filas donde el valor de p es menor que un umbral específico (cutoff). Esto selecciona solo los genes significativamente diferencialmente expresados.

write.table(sig_pval, file=sigTSV, quote=FALSE, sep=",", col.names=NA) escribe el conjunto de datos sig_pval (genes significativamente diferencialmente expresados) en otro archivo de texto (TSV). sigTSV es la ruta y el nombre del archivo donde se guardará este archivo.

✓ Volcano plot

```
#Volcano plot
discardNA <- !is.na(res$padj) #Remove padj which have NA due to the independent filtering
res2 <- res[discardNA,]
remove_outliers<- (res2$log2FoldChange >= -FCthres) & (res2$log2FoldChange <= FCthres)
res2 <- res2[remove_outliers,]

# Aquí añades la función rellenar_nombres() - para hembras
rellenar_nombres <- function(res) {
  for (i in seq_along(res$symbol)) {
    if (grep("ENS", res$symbol[i])) {
      res$symbol[i] <- ""
    } else {
      res$symbol[i] <- res$symbol[i]
    }
  }
  return(res)
}

# Aplicas la función al dataframe res2
res2 <- rellenar_nombres(res2)

jpeg(filename = volcanoF, units="in", width=8, height=10, res=300)
EnhancedVolcano(res2, lab = res2$symbol, x = 'log2FoldChange', y = 'pvalue',
                 pCutoff = 0.000976, FCcutoff= 0.3,
                 #pCutOff is p value for last significant acc to adjp value
                 ylim = c(0, 11),
                 #xlim = c(-FCthres, FCthres), Esto lo hemos dejado comentado porque no vamos a hacer corte de foldchange, pero por si q
                 labSize = 3,
                 legendLabSize = 9, legendIconSize = 5, drawConnectors = TRUE,
                 widthConnectors = 0.5, max.overlaps = 50, title = '', arrowheads = FALSE,
                 subtitle= '', gridlines.major = FALSE, gridlines.minor = FALSE)
invisible(dev.off())

```

Este script realiza filtrado y preparación de los resultados de un análisis de expresión génica diferencial y crea un gráfico de volcán utilizando la librería EnhancedVolcano.

El **volcano plot** es una herramienta visual utilizada en análisis de expresión génica diferencial para representar de manera conjunta la significancia estadística y el tamaño del cambio en la expresión de genes entre dos condiciones o grupos experimentales.

Eje X (Horizontal - log2FoldChange) representa el cambio en la expresión génica entre las dos condiciones, generalmente en una escala logarítmica. Cada punto en el eje x representa un gen, y su posición horizontal indica la magnitud del cambio en la expresión génica. **Los genes que están más hacia la derecha o izquierda del gráfico muestran cambios más grandes.**

Eje Y (Vertical - log10(p-value)) representa la significancia estadística de la diferencia en la expresión génica, generalmente en una escala negativa logarítmica. Cada punto en el eje y representa un gen, y su posición vertical indica la significancia estadística del cambio en la expresión génica. **Los genes que están más arriba en el gráfico tienen p-values más pequeños, lo que indica mayor significancia.**

A menudo se aplican **líneas de corte** para destacar genes que son diferencialmente expresados de manera significativa. Por ejemplo, se pueden aplicar umbrales para el p-value y el cambio en la expresión para resaltar los genes que cumplen con ciertos criterios.

Interpretación

Los genes ubicados en la parte superior tienen p-values pequeños, lo que indica que la **diferencia en su expresión entre las dos condiciones es estadísticamente significativa**.

Los genes ubicados hacia los lados (izquierda o derecha) muestran un **cambio grande en la expresión entre las dos condiciones**.

Explicación por pasos

discardNA <- !is.na(res\$padj) crea un vector lógico (discardNA) que indica qué filas tienen valores no nulos en la columna "padj" (ajustado por el procedimiento de corrección de p-values). Esto se hace para eliminar los valores con NA debido al filtrado independiente.

res2 <- res[discardNA,] filtra el conjunto de resultados (res) para quedarse solo con las filas que no tienen valores NA en la columna "padj".

remove_outliers<- (res2\$log2FoldChange >= -FCthres) & (res2\$log2FoldChange <= FCthres) crea un vector lógico (remove_outliers) que indica qué filas tienen un cambio en el log2FoldChange dentro de ciertos límites (definidos por FCthres). Esto se hace para eliminar genes con cambios pequeños que pueden considerarse como outliers o no interesantes para el análisis.

res2 <- res2[remove_outliers,] filtra el conjunto de resultados para quedarse solo con las filas que cumplen con la condición de remove_outliers.

rellenar_nombres <- function(res) define la función rellenar_nombres, que toma un argumento res. En el caso de hembras, al presentar tantos DEGs decidimos que sólo aparecerían los genes que tengan nombre y obviar aquellos que únicamente tienen el código ENSEMBL para facilitar la lectura del gráfico.

for (i in seq_along(res\$symbol)) inicia un bucle for que recorre todos los elementos de la columna symbol del data frame res. La función seq_along(res\$symbol) genera una secuencia de números desde 1 hasta la longitud de res\$symbol.

if (grepl("ENS", res\$symbol[i])) evalúa dentro del bucle una condición if para cada elemento ressymbol[i]. La función grepl("ENS", ressymbol[i]) verifica si el valor del elemento comienza con la cadena "ENS". El carácter ^ en la expresión regular ^ENS indica el inicio de la cadena, por lo que esta condición es verdadera si ressymbol[i] comienza con "ENS".

res\$symbol[i] <- "" Si la condición if es verdadera (es decir, si ressymbol[i] comienza con "ENS"), entonces el valor de ressymbol[i] se reemplaza por una cadena vacía "".

} else {ressymbol[i] <- res\$symbol[i]} Si la condición if es falsa (es decir, si ressymbol[i] no comienza con "ENS"), el valor de ressymbol[i] se mantiene sin cambios.

}return(res)} Despues de procesar todos los elementos de res\$symbol, el bucle termina y la función devuelve el data frame res modificado.

res2 <- llenar_nombres(res2) llama a la función llenar_nombres.

jpeg(filename = volcanoF, units="in", width=8, height=10, res=300) inicia un dispositivo gráfico JPEG para guardar el gráfico de volcán en un archivo. Se configuran propiedades como el ancho, el alto, la resolución y la ruta/nombre del archivo.

EnhancedVolcano(...) utiliza la función EnhancedVolcano para crear el gráfico de volcán. Esta función es parte de la librería EnhancedVolcano y toma como argumentos varios parámetros, como los datos (res2), las columnas para las coordenadas x e y, los umbrales de corte para valores p (pCutoff) y cambio en el logaritmo de plegamiento (FCcutoff), entre otros.

invisible(dev.off()) cierra el dispositivo gráfico JPEG.

▼ PCA significant genes

```
#Get most significant genes according to cut off
significant <- subset(res, res$pvalue < cutoff)
significant <- significant[order(significant$pvalue),]
```

Filtrar los resultados del análisis (almacenados en el objeto res) para incluir solo los genes cuyo valor de p es menor que un umbral específico (cutoff). Los resultados se ordenan por p-value de manera ascendente.

```
#Discard those genes with unbelievable Fold Change (outliers)
#significant <- significant[(significant$log2FoldChange >= -FCthres) &
#                           #(significant$log2FoldChange <= FCthres),]
```

En algunos casos, se puede optar por eliminar genes cuyos valores de cambio en el logaritmo base 2 (log2FoldChange) se consideren atípicos o inverosímiles. Sin embargo, este bloque de código está comentado (#) y, por lo tanto, no se ejecuta en la versión actual del script.

```
#PCA only significant genes
interest_genes <- rownames(significant)
dds_sig <- dds[interest_genes,]
vst_sig <- varianceStabilizingTransformation(dds_sig, blind = FALSE)
mat_sig <- assay(vst_sig)
```

Se seleccionan los genes de interés basados en los resultados más significativos. Luego, se crea un nuevo objeto DESeqDataSet llamado dds_sig que contiene solo estos genes. Se aplica la transformación de estabilización de varianza (varianceStabilizingTransformation) al nuevo conjunto de datos.

```
if (!divide_sex) {
  mat_sig <- limma::removeBatchEffect(mat_sig, batch=vst$Sexo, group=vst$Grupo)

}
assay(vst_sig) <- mat_sig
```

Opcionalmente, se elimina el efecto de lote de las muestras utilizando la función removeBatchEffect del paquete limma. Esto puede ser útil si hay variabilidad no biológica entre las muestras debido a factores técnicos, como el sexo. Este bloque de código solo se ejecuta si la condición !divide_sex se cumple.

```
jpeg(filename = sigPCAF, width=900, height=900, quality=300)
pca_sig <- plotPCA(vst_sig, intgroup='Grupo')
title <- "PCA - only significant genes"
pca_sig + ggtitle(title) +
  geom_point(size = 6) +
  theme(plot.title = element_text(size=40, hjust = 0.5, face = "bold"),
        axis.title=element_text(size=20),
        legend.text=element_text(size=15),legend.title=element_text(size=15)) +
  geom_text_repel(aes(label=colnames(vst_sig)), size=5, point.padding = 0.6)
invisible(dev.off())
```

Se crea un gráfico de Análisis de Componentes Principales (PCA) utilizando solo los genes significativos. Se utiliza la función plotPCA del paquete DESeq2. El gráfico se personaliza con un título y se guarda como un archivo JPEG.

Heatmap

```
# Ordenar los genes significativos por sus valores de padj de menor a mayor
significant <- significant[order(significant$padj), ]
# Seleccionar los top 10 genes significativos - En el caso de hembras, para hembras y machos juntos sólo se cogieron los 3 primeros genes
top_10_genes <- head(significant, 10)
```

Se ordenan los genes significativos por sus valores ajustados de p (padj) de menor a mayor. Esto facilita la selección de los genes más significativos. Se seleccionan los primeros 10 genes significativos después de la ordenación.

```
# Subset de la matriz original utilizando solo los genes top 10
subcounts <- subset(mat, rownames(mat) %in% rownames(top_10_genes))
```

Se crea un subconjunto de la matriz original de expresión génica (mat) que incluye solo las filas (genes) correspondientes a los top 10 genes seleccionados.

```
# Transformar la matriz de expresión a escala logarítmica
lsubcounts <- log2(subcounts + 1)
```

La matriz de expresión se transforma a una escala logarítmica base 2 para normalizar los datos y facilitar la visualización en el heatmap.

```
# Obtener los nombres de los genes seleccionados
top_10_genes <- rellenar_nombres(top_10_genes)
sig_symbol <- as.character(top_10_genes$symbol)
```

Se llama a la función rellenar_nombres para procesar los nombres de los genes y luego se extraen los símbolos de los genes como caracteres.

```
# Definir las condiciones de agrupación (l1 y l2)
conditions <- c(l1, l2)
conds <- subset(sampleTable, sampleTable$Grupo %in% conditions)
samples <- conds$Rata
```

Se definen las condiciones de agrupación (l1 y l2), se filtra la tabla de muestras (sampleTable) para incluir solo las filas que pertenecen a estas condiciones, y se extraen las muestras correspondientes.

```
# Crear un dataframe con la información de las condiciones
df <- data.frame(condition = conds$Grupo)
rownames(df) <- samples
```

Se crea un dataframe que contiene la información de las condiciones para cada muestra. Los nombres de las filas del dataframe son los identificadores de las muestras.

```
# Definir colores para las condiciones
my_colour <- list(df = c(l1 = "skyblue", l2 = "orange"))
```

Se definen los colores que se usarán para representar las diferentes condiciones en el heatmap: l1 será "skyblue" y l2 será "orange".

```
# Título del heatmap
title <- "Heatmap of top 10 significant genes"

# Generar el heatmap
jpeg(filename = heatmapF, units="in", width=8, height=5, res=300)
pheatmap(lsubcounts, scale= 'row', cluster_rows = TRUE,
         cluster_cols = TRUE, legend= TRUE, drop_levels = TRUE,
         labels_row = make_italics(sig_symbol),
         main = title,
         annotation_col = df, annotation_colors = my_colour,
         treeheight_row = 30, treeheight_col = 20)
invisible(dev.off())
```

Utiliza la función pheatmap del paquete pheatmap para crear el mapa de calor. Se personaliza el título, las etiquetas de fila, las anotaciones de columna y otros parámetros.

✓ Validación MAGMA

Para poder realizar este análisis necesitamos crear un documento previo de los ortólogos de nuestros genes en rata con respecto a humano. Para ello utilizaremos la herramienta de biomart: <https://www.ensembl.org/info/data/biomart/index.html>.

MAGMA (Multi-marker Analysis of Genomic Annotation) es una herramienta utilizada en genómica para realizar análisis de asociación entre variantes genéticas y fenotipos complejos. Su principal enfoque es realizar análisis de asociación de genes en lugar de variantes individuales, lo que puede proporcionar una perspectiva más integrada de la contribución genética a un fenotipo.

1. HPO (Human Phenotype Ontology):

MAGMA podría utilizarse para evaluar la asociación entre variantes genéticas y fenotipos específicos definidos por términos de HPO. Los análisis de enriquecimiento podrían revelar conjuntos de genes asociados con fenotipos particulares y proporcionar información sobre las vías biológicas subyacentes.

2. GOCC (Gene Ontology Cellular Component), GOMF (Gene Ontology Molecular Function) y GOBP (Gene Ontology Biological Process):

Similar a HPO, MAGMA podría utilizarse para realizar análisis de enriquecimiento en relación con los términos de las ontologías GOCC, GOMF y GOBP. Estos análisis podrían revelar la asociación entre genes y funciones celulares, componentes celulares y procesos biológicos específicos.

3. KEGG (Kyoto Encyclopedia of Genes and Genomes):

MAGMA podría emplearse para realizar análisis de asociación entre genes y vías específicas de KEGG. Esto ayudaría a identificar genes asociados con vías metabólicas y otros procesos biológicos importantes.

```
library('dplyr')
library('msigdbr')
workingD <- rstudioapi::getActiveDocumentContext()$path
setwd(dirname(workingD))
```

Primero se establece el directorio y se cargan las librerías necesarias para la validación.

```
filterF <- 'sets_all.txt'
resF <- 'validate_all_mPFC'

#Enriched categories to validate
filter <- as.data.frame(read.csv(filterF, sep = '\t', header = F)[,1])
colnames(filter) <- 'gs_name'
```

Este script carga nombres de conjuntos genéticos desde un archivo de texto, los almacena en un marco de datos llamado filter, y cambia el nombre de la única columna a 'gs_name'.

filterF <- 'sets_all.txt' define el nombre del archivo de texto ('sets_all.txt') que se utilizará para filtrar conjuntos genéticos.

resF <- 'validate_all_mPFC' define el nombre del archivo resultante ('validate_all_mPFC') después del proceso de validación.

read.csv(filterF, sep = '\t', header = F) lee el archivo de texto usando ';' como separador y sin encabezado.

[, 1] selecciona la primera columna del marco de datos resultante (presumiblemente contiene nombres de conjuntos genéticos).

as.data.frame(...) convierte el resultado en un nuevo marco de datos.

filter asigna este marco de datos a la variable filter.

colnames(filter) <- 'gs_name' cambia el nombre de la única columna del marco de datos filter a 'gs_name'. Esto es útil para una identificación más clara del contenido.

```
#KEGG and REACTOME
C2_set <- as.data.frame(msigdbr(species = 'Homo sapiens', category = 'C2',
                                   subcategory = 'KEGG'))%>%
  dplyr::select(gs_name, ensembl_gene)

#GO and HPO
GO_set <- as.data.frame(msigdbr(species = 'Homo sapiens', category = 'C5',
                                   subcategory = NULL))%>%
  dplyr::select(gs_name, ensembl_gene)
```

La primera parte del código está extrayendo información sobre genes asociados a vías metabólicas de la base de datos KEGG para la especie 'Homo sapiens', y luego selecciona y almacena dos columnas ('gs_name' y 'ensembl_gene') en un data frame llamado C2_set.

msigdbr(species = 'Homo sapiens', category = 'C2', subcategory = 'KEGG') utiliza la función msigdbr para obtener información sobre genes relacionados con la categoría 'C2' y la subcategoría 'KEGG' para la especie 'Homo sapiens' (ser humano). Esto sugiere que se está obteniendo información sobre genes asociados a vías metabólicas según la base de datos KEGG.

%>% se utiliza para encadenar funciones en R. Lo que sigue después de este operador se aplica a la salida de la función anterior.

dplyr::select(gs_name, ensembl_gene) utiliza la función select de la biblioteca dplyr para seleccionar las columnas 'gs_name' y 'ensembl_gene' del resultado obtenido en el paso anterior. Esto implica que se están extrayendo específicamente dos columnas de la información obtenida de KEGG.

as.data.frame(...) bold text se utiliza para convertir el resultado en un objeto de tipo data frame en R.

La segunda parte de este código está creando un conjunto de datos llamado GO_set que contiene información sobre genes relacionados con ontologías genéticas (GO) y fenotípicas (HPO) en la especie 'Homo sapiens'. Este conjunto de datos se estructura como un data frame y contiene las columnas 'gs_name' y 'ensembl_gene'.

```
#Hallmark
sets <- bind_rows(C2_set, GO_set)
target_sets <- merge(sets, filter, by='gs_name')
```

Este código combina dos conjuntos de datos previamente definidos (C2_set y GO_set) en un solo data frame (sets). Luego, realiza una fusión con otro data frame (filter) basándose en la columna 'gs_name'. El resultado final se almacena en el data frame target_sets. La variable target_sets contendrá las filas que tienen valores coincidentes en la columna 'gs_name' en ambos conjuntos de datos originales.

bind_rows(C2_set, GO_set) combina los data frames C2_set y GO_set en un solo data frame utilizando la función bind_rows. Ambos data frames deben tener la misma estructura (mismas columnas) para que esta operación tenga éxito.

sets <- bind_rows(C2_set, GO_set) almacena el resultado de la combinación en un nuevo data frame llamado sets.

merge(sets, filter, by='gs_name') realiza una fusión (merge) entre el data frame sets y el data frame filter utilizando la columna 'gs_name' como clave de unión. Esto implica que se están seleccionando las filas de sets que tienen valores coincidentes en la columna 'gs_name' con las filas de filter.

target_sets <- merge(sets, filter, by='gs_name') almacena el resultado de la fusión en un nuevo data frame llamado target_sets.

```
#Check merge
uni <- unique(target_sets[c('gs_name')])
dim(filter) == dim(uni)

write.table(target_sets, file = resF, sep="\t", col.names=F, quote = F, row.names = F)
```

Este bloque de código verifica si los valores en la columna 'gs_name' de target_sets son únicos y luego compara las dimensiones de dos data frames. Posteriormente, escribe el contenido del data frame target_sets en un archivo de texto.

uni <- unique(target_sets[c('gs_name')]) extrae la columna 'gs_name' del data frame target_sets y luego utiliza la función unique para obtener los valores únicos en esa columna. Estos valores únicos se almacenan en un nuevo data frame llamado uni.

dim(filter) == dim(uni) compara las dimensiones (número de filas y columnas) del data frame filter con las dimensiones del data frame uni. Si estas dimensiones son iguales, la expresión devuelve TRUE; de lo contrario, devuelve FALSE. Esto se utiliza para verificar si filter y uni tienen la