

DESeq2_biotech

Sara González Bodí y Laura Rodriguez Casillas (modificado por Pablo Pérez Rodríguez & Yixi Zhang)

21 junio, 2025

Contents

Important web tutorials/manuals	1
Installation packages	2
Get the paths for all the directories and read the files needed	2
ONLY RUN FOR HTSeq-counts OUTPUTS	2
Upload data for HTSeq-count users and create DESeq object	2
Prefiltering options before running DESeq2	4
Run the Differential Expression Analysis	5
Comparison: run DESeq2 for the comparison -> altered vs control	10
Create DESeq2 dataset object	10
Useful plots for Differential Expression Analysis	11
Gene Ontology enrichment annotation with ClusterProfiler (Up regulated): BiologicalProcess	13
Gene Ontology enrichment annotation with ClusterProfiler (Down_regulated): Biological- Process	14
Gene Ontology enrichment annotation with ClusterProfiler (Down_regulated): Celular- Component	16
Gene set enrichment analysis with HallMark gene set annotation	17

Important web tutorials/manuals

- #1. DESeq2: <https://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html>
- #2. Gene Ontology with ClusterProfiler: <https://yulab-smu.top/biomedical-knowledge-mining-book/clusterprofiler-go.html>

Installation packages

```
## All packages loaded successfully.
```

Get the paths for all the directories and read the files needed

```
# Directory where the counts files are stored
# In case you work with htseq counts output
counts_directory <- ""
# In case you work with feature counts output
counts_directory <- "/home/yixiz/counting_res/"
# Output directory
output_dirertry <- "/home/yixiz/Deseq2_res/"
# Table with metadata (you have to create it by your own)
fullsamples <- read.csv("/home/yixiz/counts_metadata.csv")

# Finally, if the output directory does not exist, create it
if (! dir.exists(output_dirertry)){ dir.create(output_dirertry, recursive = TRUE)}
```

ONLY RUN FOR HTSeq-counts OUTPUTS

Upload data for HTSeq-count users and create DESeq object

```
# Get the samples files sorted and their names
sampleFiles <- sort(list.files(counts_directory, pattern = "*txt"))

# Get the samples names
get.sample.name <- function(fname) strsplit(basename(fname), ".txt")[[1]][1]
sample.names <- unname(sapply(sampleFiles, get.sample.name))

# Create a table with the samples and files names
sampleTable <- data.frame(sampleName = sample.names,
                           fileName = sampleFiles)

# Merge the samples table with the samples information
merge <- merge(sampleTable, fullsamples,
                by.x="sampleName", by.y = "SampleID")

# Order the merged table by sampleName
merge <- merge[order(merge$sampleName, decreasing = TRUE),]
# Initialize an empty variable to store counts for each sample
countData <- NULL

# Loop through each sample file and read the featureCounts files
for (i in 1:length(sampleFiles)) {
  # Get the complete path to the featureCounts file we are iterating over
  countFile <- file.path(counts_directory, sampleFiles[i])
```

```

# Read the featureCounts output file
countTable <- read.delim(countFile, header = TRUE, comment.char="#", row.names = 1)
# Extract the correct count column (usually the last column, but this can vary)
countColumn <- countTable[, ncol(countTable)]
# Add the count column to the countData matrix, preserving row names (gene names)
if (is.null(countData)) {
  countData <- countColumn
} else {
  countData <- cbind(countData, countColumn)
}
}

# Set the row names of countData to the gene names (from the first file, assumed to be the same across all samples)
rownames(countData) <- rownames(countTable)

# Set the column names for the countData matrix to the sample names
colnames(countData) <- sample.names
# Make sure the merge DataFrame matches the order of columns in countData
merge <- merge[match(colnames(countData), merge$sampleName), ]
# Check if the dimensions match
if (ncol(countData) == nrow(merge)) {
  print("Sample count matches!")
} else {
  stop("Number of samples does not match between countData and merge")
}

## [1] "Sample count matches!"

# Ensure that 'Condition' is a factor (it should already be, but let's ensure)
merge$Condition <- factor(merge$Condition)

# Create the DESeqDataSet
ddsHTSeq <- DESeqDataSetFromMatrix(countData = countData, colData = merge,
                                     design = ~Condition)

# Check the DESeqDataSet object
ddsHTSeq

## class: DESeqDataSet
## dim: 78894 6
## metadata(1): version
## assays(1): counts
## rownames(78894): ENSG00000142611 ENSG00000284616 ... ENSG00000309831
##   ENSG00000309258
## rowData names(0):
## colnames(6): counts_sample_1 counts_sample_2 ... counts_sample_5
##   counts_sample_6
## colData names(3): sampleName fileName Condition

```

Prefiltering options before running DESeq2

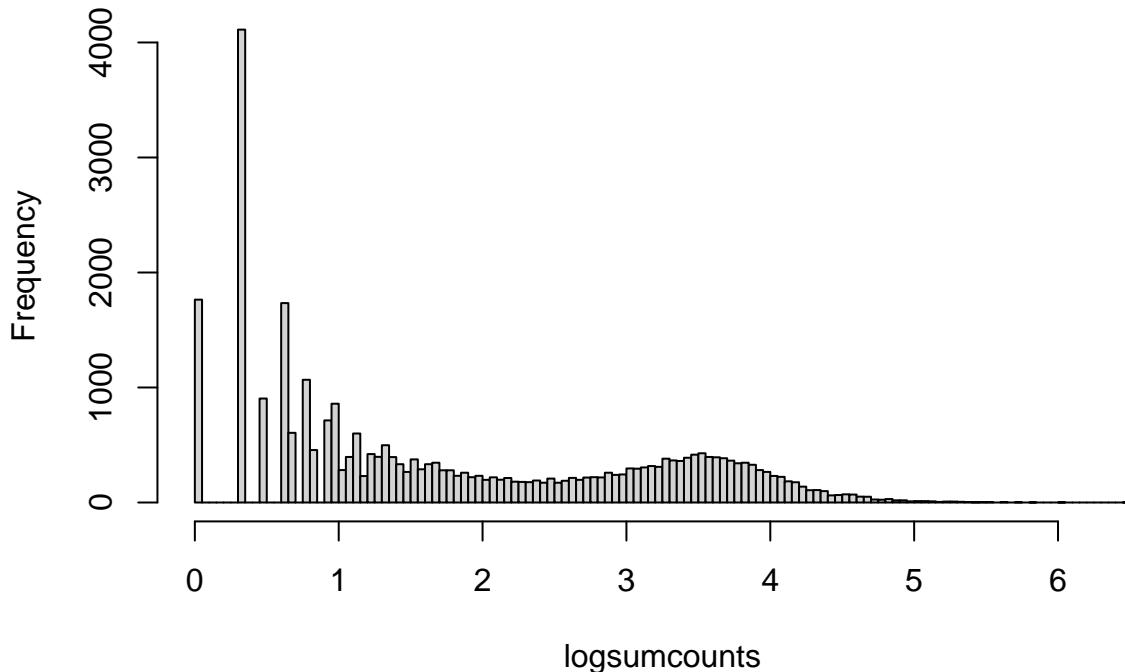
```
# Perform a minimal pre-filtering to keep only rows that have at least 10 (decided below) reads total

# Sum count for each gene across samples
sumcounts <- rowSums(counts(ddsHTSeq))

# take the log of the sum for each gene
logsumcounts <- log(sumcounts,base=10)

# 1st plot a histogram of the log scaled counts
# This will be useful in order to decide in which value keep the genes
plot_without_filter <- hist(logsumcounts,breaks=100, main="Histogram of the log scaled counts")
```

Histogram of the log scaled counts



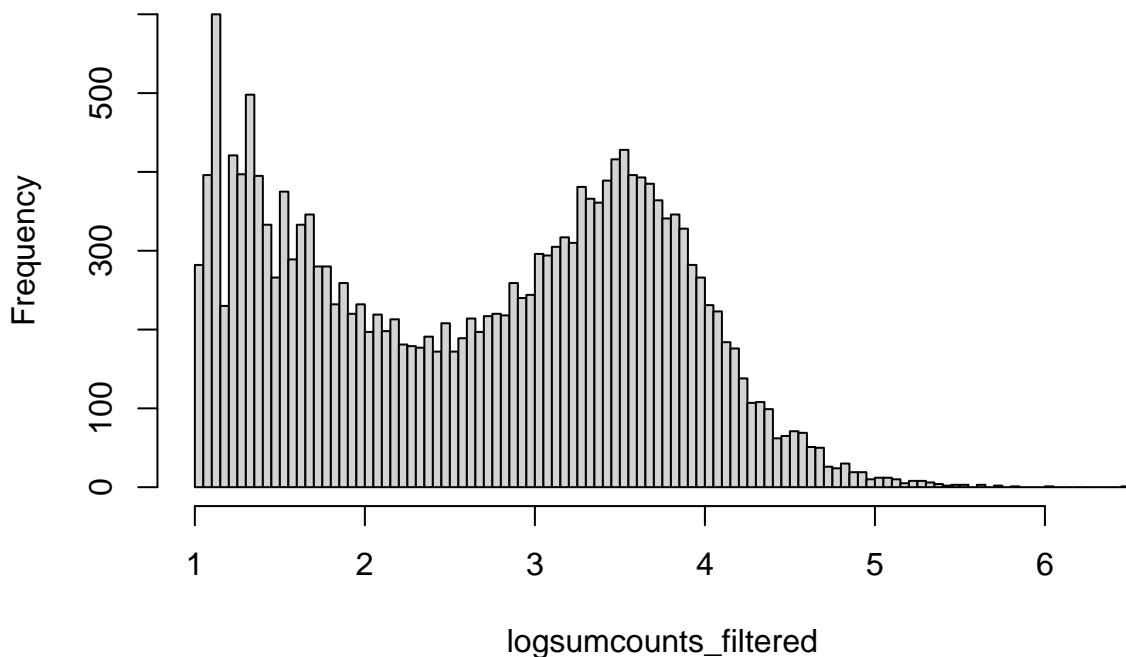
```
# Get the genes with summed counts greater than 10! (This value could vary anytime you run another RNA-seq)

# This remove the lowly expressed genes
keep <- sumcounts > 10
# Keep the genes which sumcounts > 10 from the ddsHTSeq object
ddsHTSeq_filter <- ddsHTSeq[keep,]

# Finally, do another histogram but this time with the filtered ddsHTSeq object
sumcounts_filtered <- rowSums(counts(ddsHTSeq_filter))
logsumcounts_filtered <- log(sumcounts_filtered,base=10)
```

```
plot_with_filter <- hist(logsumcounts_filtered, breaks=100, main="Histogram of the log scaled counts \n k")
```

**Histogram of the log scaled counts
keep only the genes with summed counts greater than 50**



Run the Differential Expression Analysis

```
# Run the Differential Expression Analysis
dds <- DESeq(ddsHTSeq_filter)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

```

# Print the names of the results in the Differential Expression Analysis
print (resultsNames(dds))

## [1] "Intercept"                                "Condition_LowGlucose_vs_HighGlucose"

# IMPORTANT: the following data transformation is only done for REPRESENTATION,
# it is not meant to be used for analysis
## Normalize the counts
normalized_counts <- counts(dds, normalized=TRUE)

## Log transform the normalized counts - Get the Median Absolute Deviation per gene (1 means by row)
normalized_counts_mad <- apply(normalized_counts, 1, mad)
## Order the normalized counts matrix using the Median Absolute Deviation
normalized_counts <- normalized_counts[order(normalized_counts_mad, decreasing=T), ]

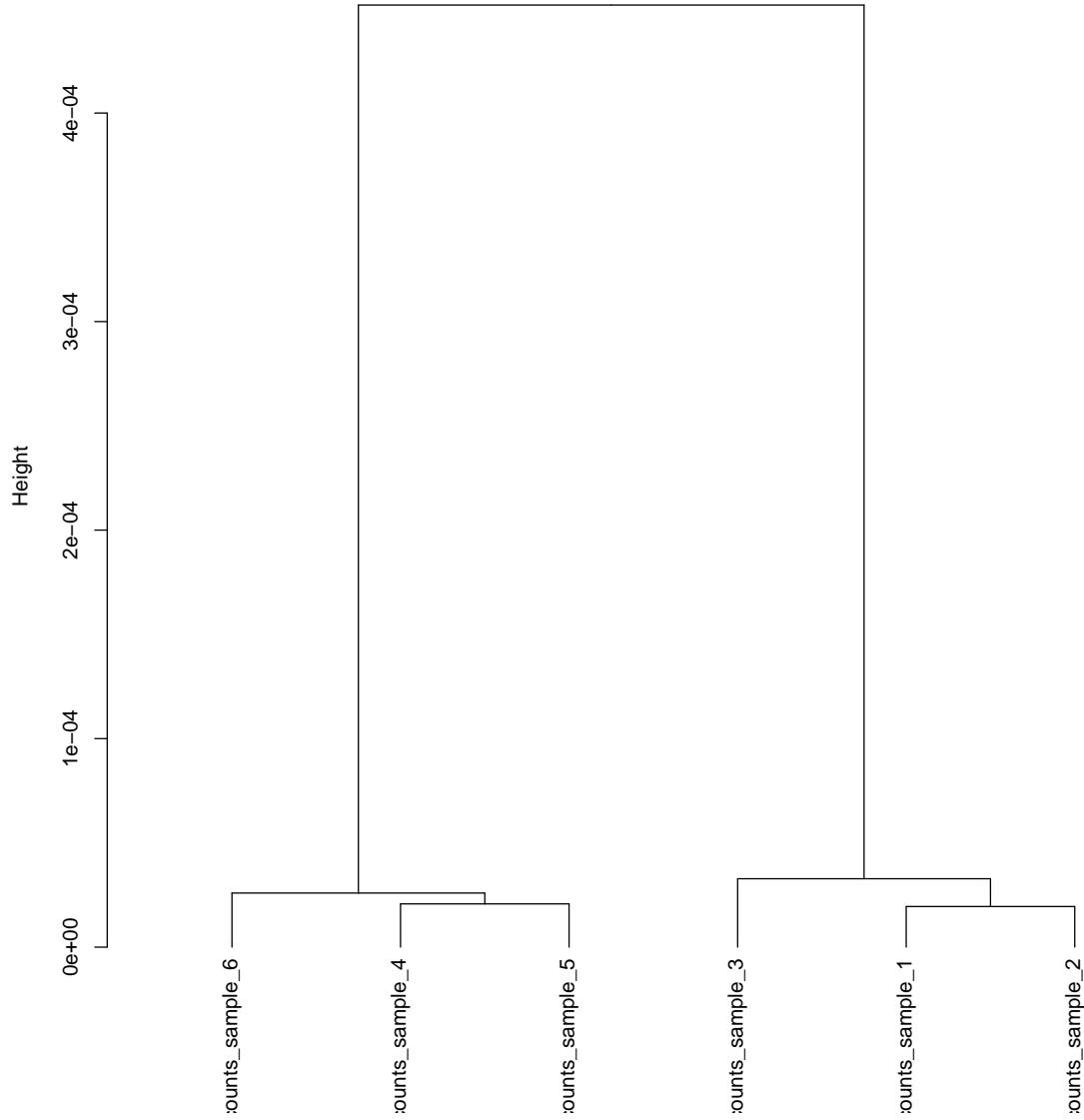
## Another method to create a variance.stabilized transformed counts for visualization
vsd <- vst(dds, blind=FALSE)

rlogMat <- assay(vsd)
rlogMat <- rlogMat[order(normalized_counts_mad, decreasing=T), ]

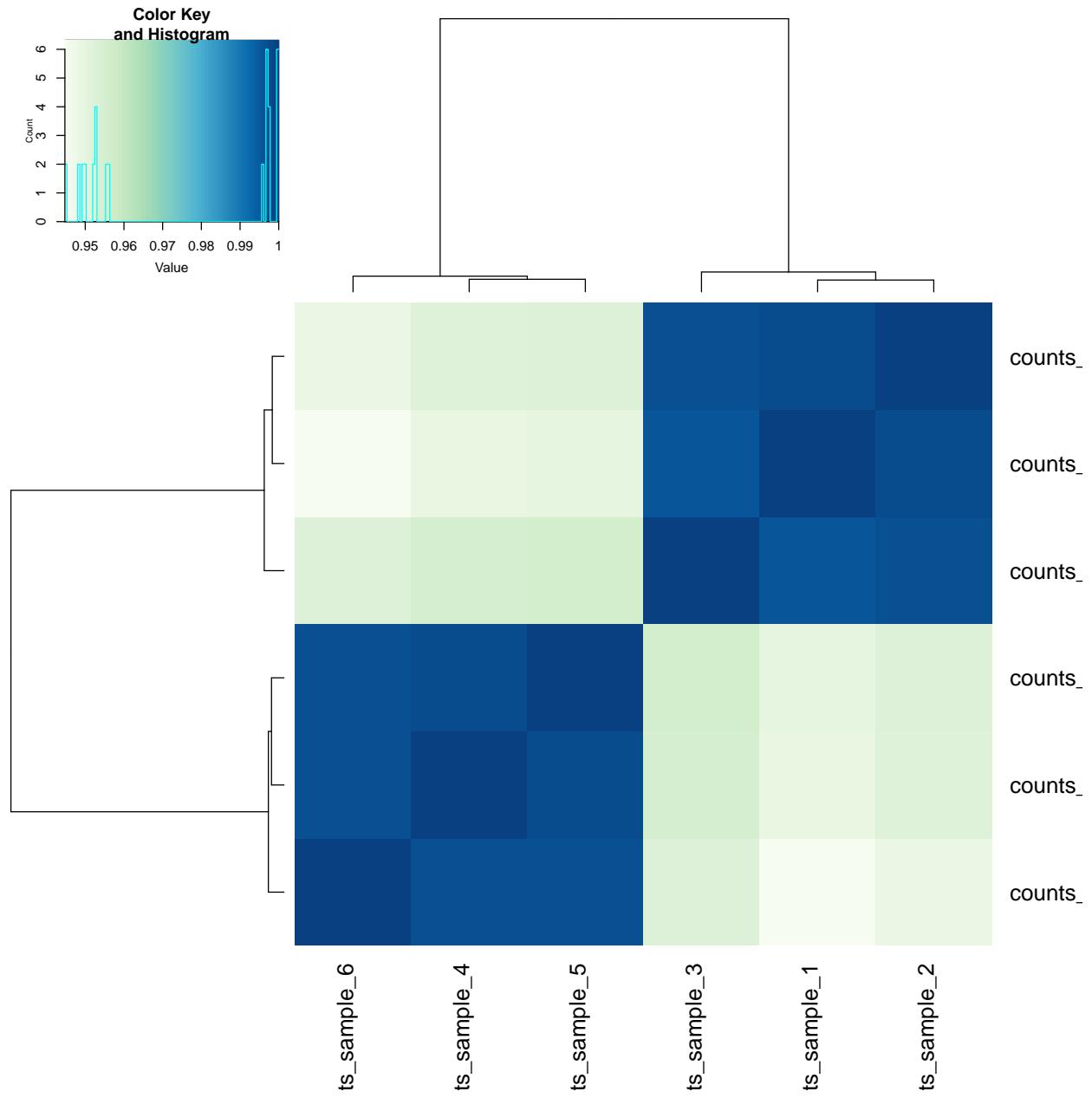
## Hierarchical clustering
hc <- hcluster(t(rlogMat), method="pearson")
# Get a dendrogram to check for outliers
TreeC = as.dendrogram(hc, method="average")
plot(TreeC, main = "Sample Clustering-checking outliers ", ylab = "Height")

```

Sample Clustering–checking outliers



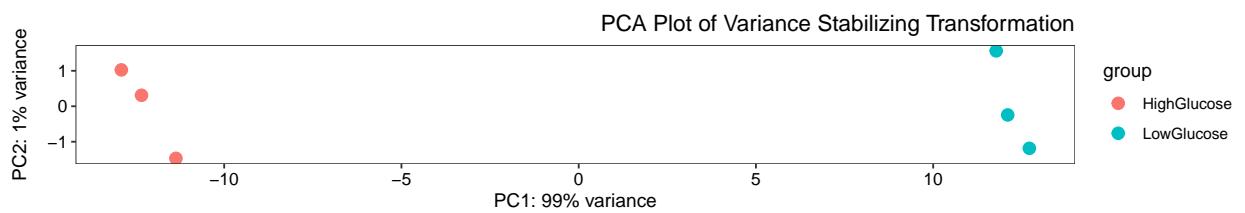
```
# Estimate the pearson correlation to see it as a dendrogram!
pearson_cor <- as.matrix(cor(rlogMat, method="pearson"))
hmcol <- colorRampPalette(brewer.pal(9, "GnBu"))(100) # This is to establish the color palette
heatmap.2(pearson_cor, Rowv=dendrogram(hc), symm=T, trace="none",
          col=hmcol, margins=c(8,5))
```



```
# plot PCA
plotPCA(vsd, intgroup="Condition") + ggtitle("PCA Plot of Variance Stabilizing Transformation") +
  geom_point(size = 1) +
  theme(axis.line = element_line(colour = "black"),
        plot.title = element_text(angle = 0, size = 14, face = 'bold', vjust = 0.5, hjust = 0.5),
        plot.subtitle = element_blank(), plot.caption = element_blank(), # subtitle elements
        axis.text.x = element_text(angle = 0, size = 12, vjust = 1), # x axis title
        axis.text.y = element_text(angle = 0, size = 12, vjust = 0.5), # y axis title
        axis.title = element_text(size = 13), # both axis titles

        legend.position = 'right', legend.key.size = unit(0.5, 'cm'), # legend position
        legend.text = element_text(size = 12), legend.title = element_blank()) + # legend text
  theme_bw() +
```

```
theme(  
  panel.grid.minor = element_blank(),  
  panel.grid.major = element_blank(),  
  panel.border = element_rect(size = 0.20),  
  axis.title.x = element_text(size=11),  
  axis.title.y = element_text(size = 11, margin = margin(t = 0, r = 10, b = 0, l = 0)),  
  axis.text.x = element_text(color="black",size=10),  
  axis.text.y = element_text(color="black",size=10),  
  legend.position = "right") +  
theme(plot.title = element_text(hjust = 1.0))  
  
## using ntop=500 top features by variance  
  
## Warning: The `size` argument of `element_rect()` is deprecated as of ggplot2 3.4.0.  
## i Please use the `linewidth` argument instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was  
## generated.
```



Comparison: run DESeq2 for the comparison → altered vs control

Create DESeq2 dataset object

```
# What sample is the reference in our analysis?
ref <- "LowGlucose"

# Relabel the filtered DESeq2 object by the reference in the specific analysis
ddsHTSeq_filter$Condition <- relevel(ddsHTSeq_filter$Condition, ref = ref)
```

```

## Normalize and calculate dispersion with the reference of the specific analysis
dds <- DESeq(ddsHTSeq_filter) # <- normalize and calculate dispersion

## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

print (resultsNames(dds))

## [1] "Intercept"                               "Condition_LowGlucose_vs_HighGlucose"

# Get the comparison results including the condition
paired_comparison_results <- results(dds, name="Condition_LowGlucose_vs_HighGlucose", pAdjustMethod="BH")
# Custom the output comparison
paired_comparison_results_tb <- paired_comparison_results %>% data.frame() %>%
rownames_to_column(var="gene") %>% as_tibble()

# Store the output comparison data frame to a csv file
# Create the name for the output folder
comparison_name_folder <- "Condition_LowGlucose_vs_HighGlucose"
# Get the complete path to store the output and create the directory if it does not already exists
output_path <- file.path(output_directory, comparison_name_folder)
if (!dir.exists(output_path)) {dir.create(file.path(output_path), recursive = TRUE)}

# Applying normalization to data (ignoring design) - for clustering, etc.
dataf <- vst(dds, blind = TRUE) %>% assay()
dataf <- dataf %>% as_tibble(rownames = "gene")

# Set the thresholds where you want to filter your data
padj.cutoff <- 0.01
lfc.cutoff <- 0.58 # remember log!

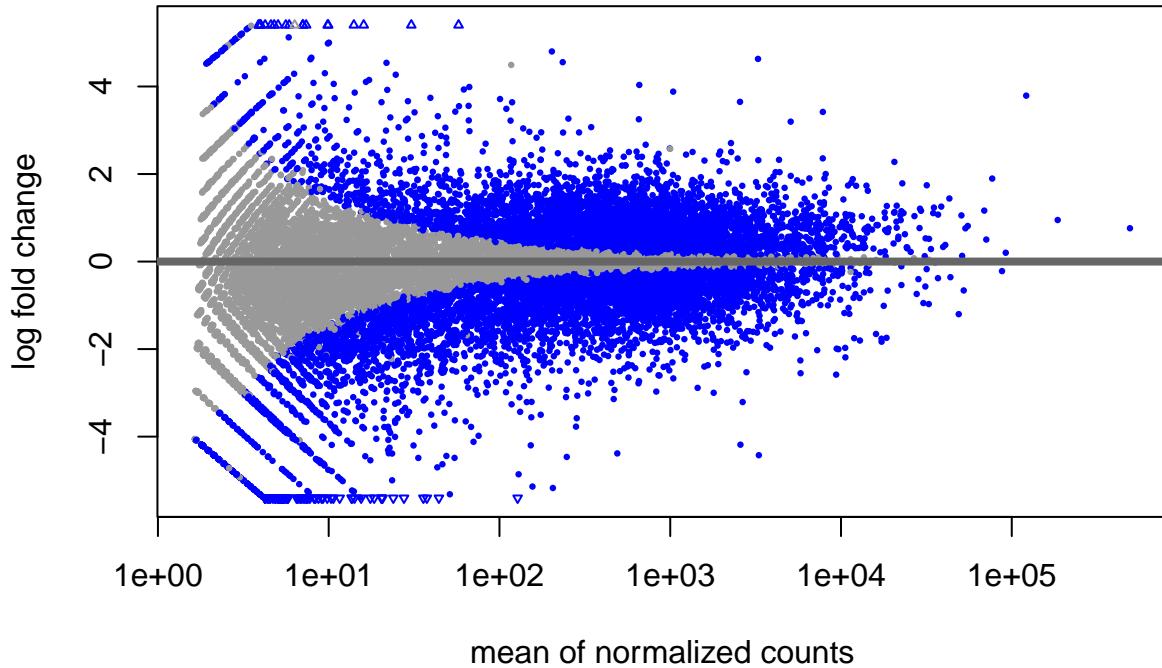
# Merge the output table from the comparison with the counts (dataf)
# Add also another column that includes TRUE/FALSE in case the gene pass the thresholds set
resdata <- merge(as.data.frame(paired_comparison_results_tb), as.data.frame(dataf), by = 'gene', sort = TRUE)

# Write the complete table to the output folder
write.table(resdata, file = paste(output_path, (paste0(comparison_name_folder, "_final_results.csv"))),

```

Useful plots for Differential Expression Analysis

```
# Create MA plot
plotMA(dds)
```



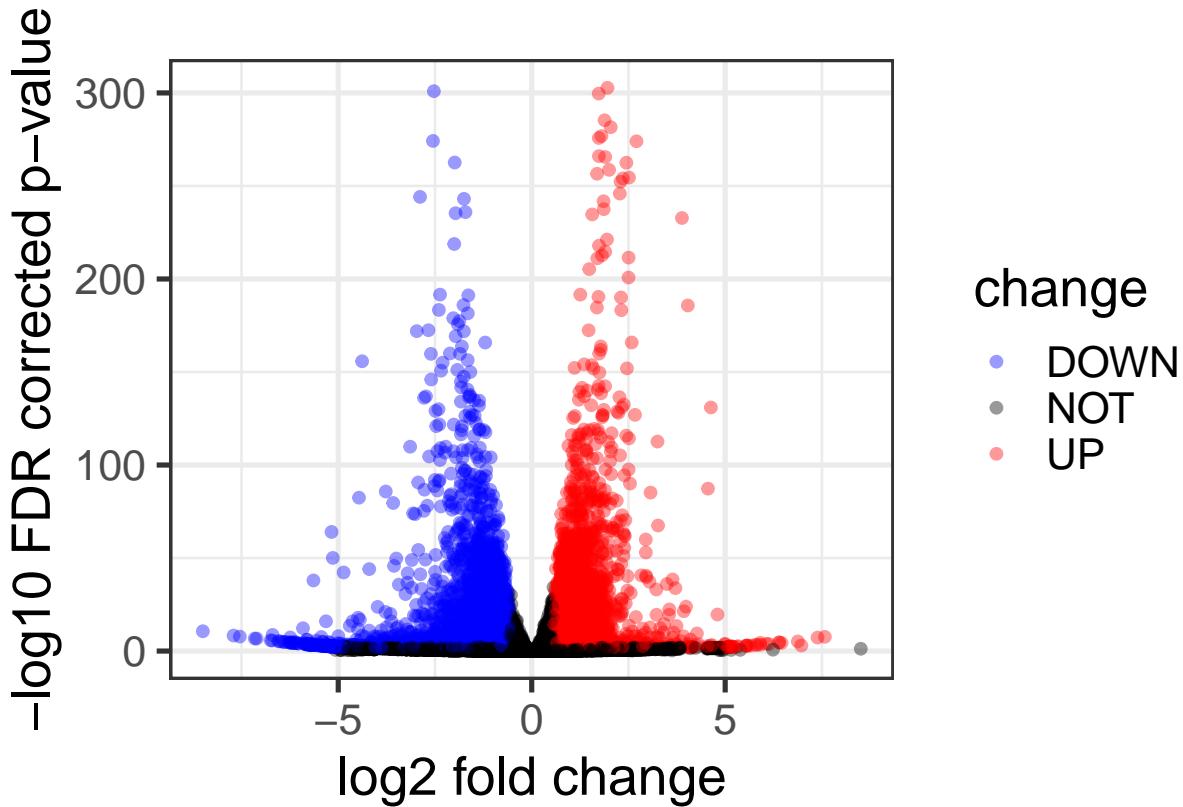
```
# Create a volcano plot
#   Add a variable to resdata to include UP/DOWN or NOT taking into account the thresholds
resdata$change <- as.factor(ifelse(
  is.na(resdata$padj) | resdata$padj > padj.cutoff,
  'NOT',
  ifelse(resdata$padj <= padj.cutoff & abs(resdata$log2FoldChange) > lfc.cutoff,
    ifelse(resdata$log2FoldChange > lfc.cutoff, 'UP', 'DOWN'),
    'NOT')))

#   Create the plot
volcano_data <- resdata %>%
  filter(!is.na(log2FoldChange),
    !is.na(padj),
    padj > 0,
    is.finite(log2FoldChange),
    is.finite(-log10(padj)))

volcanoPlot = ggplot(data=volcano_data, aes(x=log2FoldChange, y=-log10(padj), color=change)) +
  geom_point(alpha=0.4, size=1.75) +
  theme_set(theme_set(theme_bw(base_size=20))) +
  xlab("log2 fold change") + ylab("-log10 FDR corrected p-value") +
  ggtitle( "") + theme(plot.title = element_text(size=15,hjust = 0.5)) +
  scale_colour_manual(values = c('blue','black','red')) ## corresponding to the levels(res)

# This is to show the plot
```

```
show(volcanoPlot)
```



Gene Ontology enrichment annotation with ClusterProfiler (Up regulated): BiologicalProcess

```
# We need to get the Universe of genes or background genes
background_genes <- as.vector(resdata$gene)

# We need to get the significant genes in the DEGs (Differentially Expressed Genes)
interesting_set <- resdata %>% filter (log2FoldChange >= lfc.cutoff & padj <= padj.cutoff)
interesting_set <- as.vector(interesting_set$gene)

# Run the Gene Ontology Enrichment analysis
enrichment <- enrichGO(gene = interesting_set, # DEGs list
                        OrgDb = org.Hs.eg.db, # organism
                        keyType = "ENSEMBL", # The name included in your background/interesting set
                        ont = "BP", # The analysis you want to run: CC, BP, MF
                        pvalueCutoff = 0.01, # Adjusted pvalue Cutoff on enrichment test to report
                        pAdjustMethod = "BH", # Method for the p.adjusted to be calculated
                        universe = background_genes, # the background genes
                        qvalueCutoff = 0.05)
```

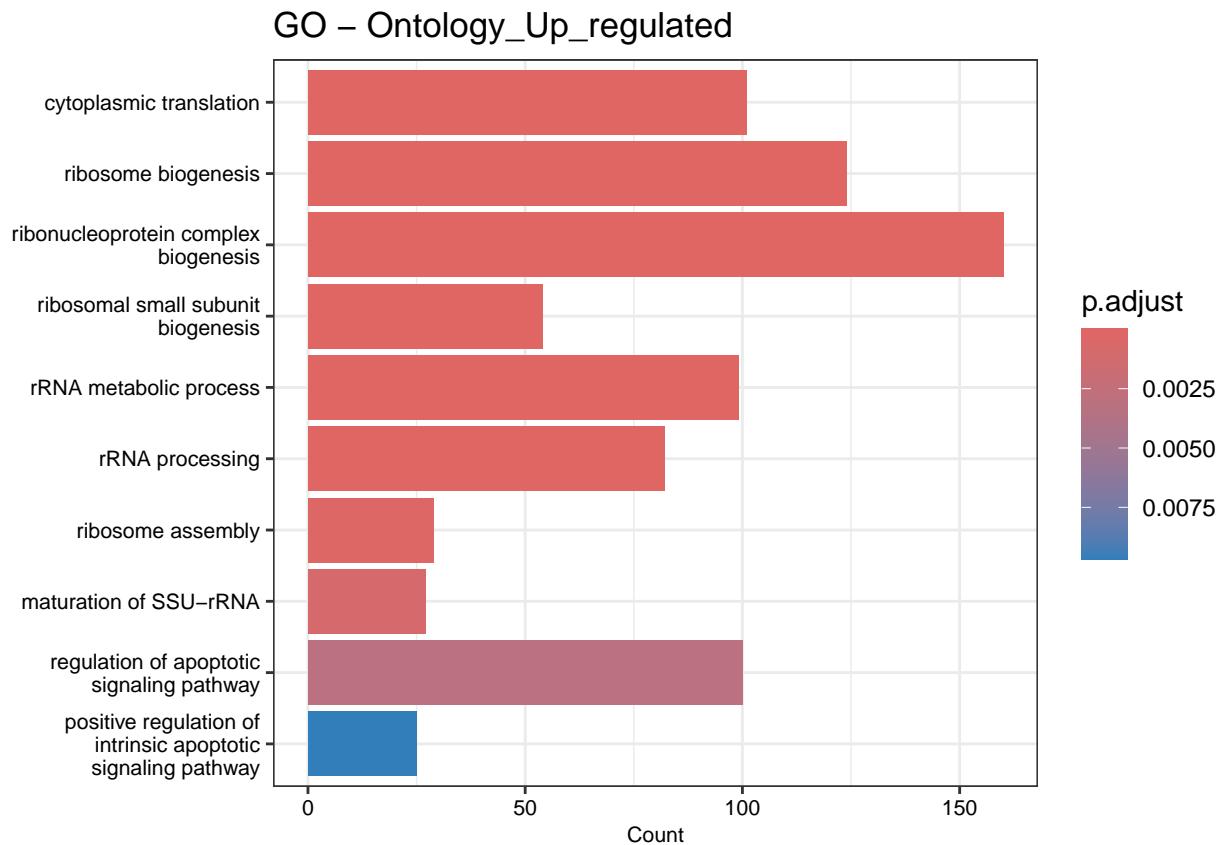
```

# Get the result from the enrichment as a data frame
ora_analysis_bp_df <- as.data.frame(enrichment@result)

# Get the representation of the Gene Ontology
barplot <- barplot(enrichment,
                     showCategory = 100,
                     font.size = 8) +
  ggtitle("GO - Ontology_Up_regulated ")

show(barplot)

```



Gene Ontology enrichment annotation with ClusterProfiler (Down_regulated): BiologicalProcess

```

# We need to get the Universe of genes or background genes
background_genes <- as.vector(resdata$gene)

# We need to get the significant genes in the DEGs (Differentially Expressed Genes)
interesting_set <- resdata %>% filter (log2FoldChange <= -lfc.cutoff & padj <= padj.cutoff) #In this case
interesting_set <- as.vector(interesting_set$gene)

```

```

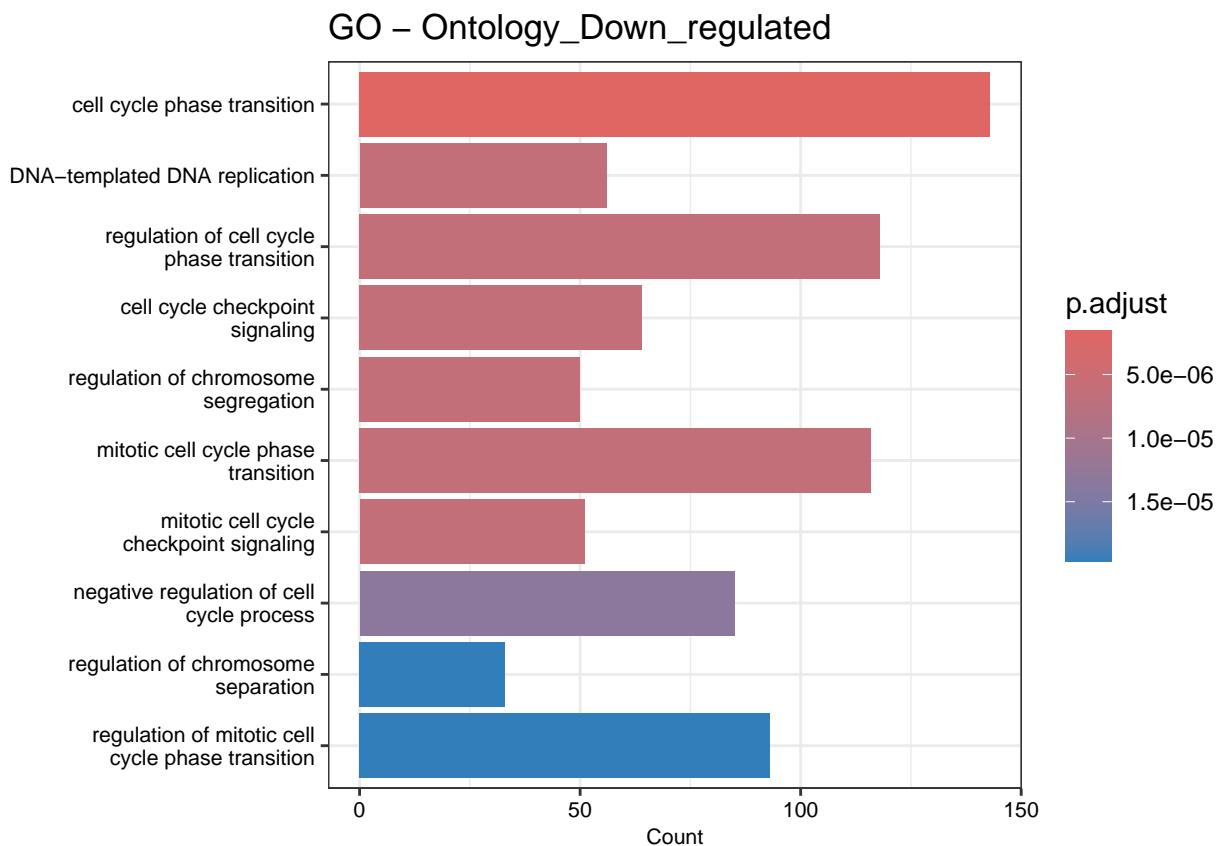
# Run the Gene Ontology Enrichment analysis
enrichment <- enrichGO(gene = interesting_set, # DEGs list
                       OrgDb = org.Hs.eg.db, # organism
                       keyType = "ENSEMBL", # The name included in your background/interesting set
                       ont = "BP", # The analysis you want to run: CC, BP, MF
                       pvalueCutoff = 0.01, # Adjusted pvalue Cutoff on enrichment test to report
                       pAdjustMethod = "BH", # Method for the p.adjusted to be calculated
                       universe = background_genes, # the background genes
                       qvalueCutoff = 0.05)

# Get the result from the enrichment as a data frame
ora_analysis_bp_df <- as.data.frame(enrichment$result)

# Get the representation of the Gene Ontology
barplot <- barplot(enrichment,
                     showCategory = 10,
                     font.size = 8) +
  ggtitle("GO - Ontology_Down_regulated")

show(barplot)

```



Gene Ontology enrichment annotation with ClusterProfiler (Down_regulated): CelularComponent

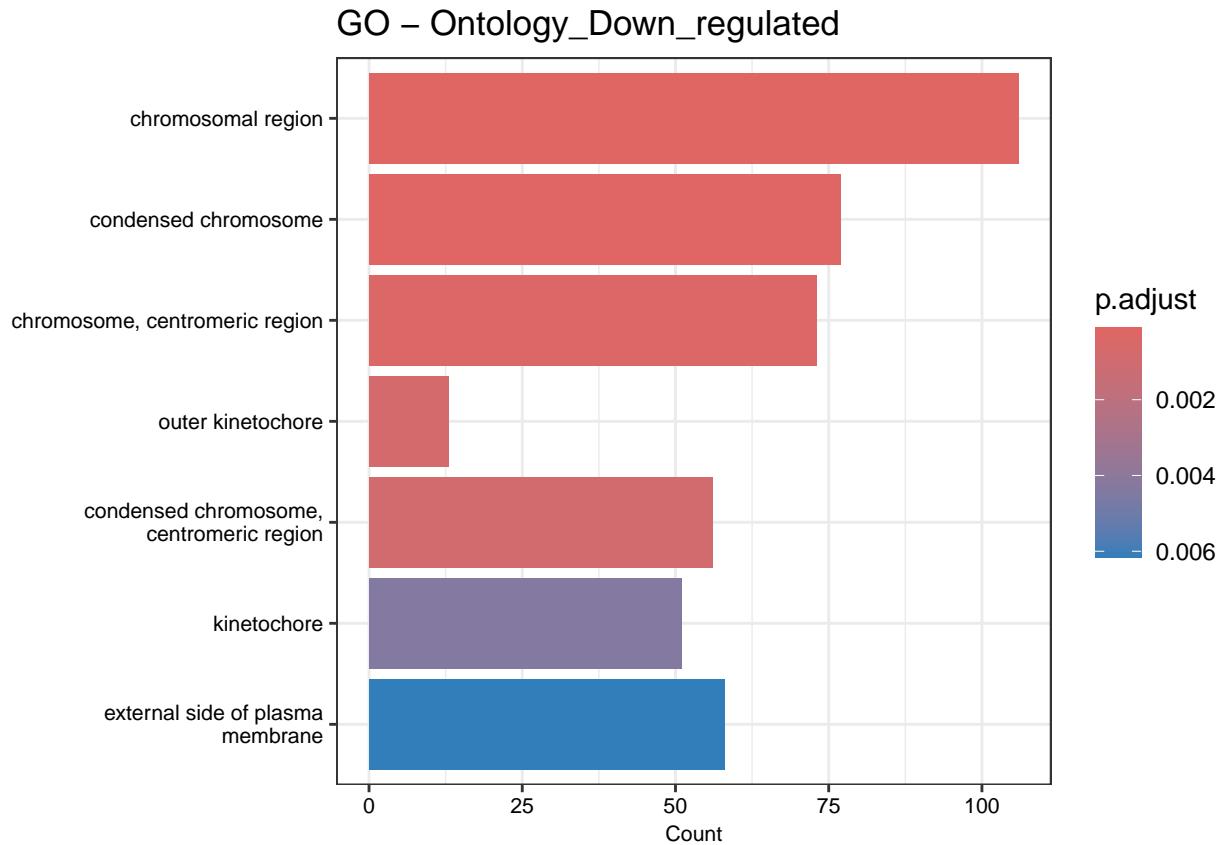
```
# We need to get the Universe of genes or background genes
background_genes <- as.vector(resdata$gene)

# We need to get the significant genes in the DEGs (Differentially Expressed Genes)
interesting_set <- resdata %>% filter (log2FoldChange <= -lfc.cutoff & padj <= padj.cutoff)
interesting_set <- as.vector(interesting_set$gene)
# Run the Gene Ontology Enrichment analysis
enrichment <- enrichGO(gene = interesting_set, # DEGs list
                        OrgDb = org.Hs.eg.db, # organism
                        keyType = "ENSEMBL", # The name included in your background/interesting set
                        ont = "CC", # The analysis you want to run: CC, BP, MF
                        pvalueCutoff = 0.01, # Adjusted pvalue Cutoff on enrichment test to report
                        pAdjustMethod = "BH", # Method for the p.adjusted to be calculated
                        universe = background_genes, # the background genes
                        qvalueCutoff = 0.05)

# Get the result from the enrichment as a data frame
ora_analysis_bp_df <- as.data.frame(enrichment@result)

# Get the representation of the Gene Ontology
barplot <- barplot(enrichment,
                     showCategory = 10,
                     font.size = 8) +
  ggtitle("GO - Ontology_Down_regulated")

show(barplot)
```



Gene set enrichment analysis with HallMark gene set annotation

```

# Converting the geneID to geneSymbols for GSEA using HallMark gene set annotation
gene_map <- bitr(resdata$gene,
                  fromType = "ENSEMBL",
                  toType = "SYMBOL",
                  OrgDb = org.Hs.eg.db)

## 'select()' returned 1:many mapping between keys and columns

## Warning in bitr(resdata$gene, fromType = "ENSEMBL", toType = "SYMBOL", OrgDb =
## org.Hs.eg.db): 18.79% of input gene IDs are fail to map...

# Creating a new data.frame merging both resdata and gene_map
resdata2 <- merge(resdata, gene_map, by.x = "gene", by.y = "ENSEMBL")

# print (resdata2) #Optional: to print resdata2

# Now, we have to create a sorted gene list, in this case, by log2FoldChange in a descendant order

original_gene_list = resdata2$log2FoldChange #This line creates a list with all log2FC valor

```

```

names (original_gene_list) <- resdata2$SYMBOL #Associating the SYMBOL to the correspondent log2FC

gene_list = sort (original_gene_list, decreasing = TRUE) #Sorting the list in a descendant order

gene_list <- gene_list[!duplicated(names(gene_list))] #Eliminating duplicated valors, which gives error

# print (gene_list) #Optional: to see the resulting gene list

#Downloading the HallMark gene set annotation with msigdbr
m_df <- msigdbr(collection = "Homo sapiens", category = "H")

## Warning: The `category` argument of `msigdbr()` is deprecated as of msigdbr 10.0.0.
## i Please use the `collection` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

m_df <- m_df[, c("gs_name","gene_symbol")] #We only need this two columns
# print (m_df) #Optional: to see the m_df

#Running the GSEA (gene set enrichment analysis)
gsea_hallmark <- GSEA (gene_list,
                        TERM2GENE = m_df,
                        pvalueCutoff = 0.05)

## using 'fgsea' for GSEA analysis, please cite Korotkevich et al (2019).

## preparing geneSet collections...

## GSEA analysis...

## Warning in preparePathwaysAndStats(pathways, stats, minSize, maxSize, gseaParam, : There are ties in
## The order of those tied genes will be arbitrary, which may produce unexpected results.

## leading edge analysis...

## done...

#Show the result with a dotplot
dotplot (gsea_hallmark,
          showCategory = 10 ,
          title = "Hallmark Pathways Enriquecidos",
          font.size = 12)

```

