RNA-Seq Workshop Manual

Chapter 4. Introduction to Differential Gene Expression Analysis: The DESeq2 and edgeR Tools, and Visualization

The goal of the two R packages, DESeq2 and edgeR, is to identify genes that show statistically significant differences in expression levels between different experimental groups (e.g., treatment vs. control, different disease states). This process is known as differential gene expression (DGE) analysis. They also filter out genes with low counts and genes which show zero to low variance across experimental conditions to increase statistical power.

Both algorithms assume that for each gene in each sample group, the observed counts have a negative binomial distribution, since RNA-seq data consists of read counts, which are discrete, non-negative integers (therefore, RNA-Seq data cannot have a normal distribution).

While a Poisson distribution is the common choice for count data, it has a strict assumption: its mean must equal its variance. In RNA-seq data, the variance of gene counts almost always exceeds the mean. This phenomenon is called overdispersion, which in the case of gene counts, is modeled as a coefficient in the quadratic estimation of the variance.

DESeq2: To determine if there is a significant difference in expression across the conditions, a Wald test or a Likelihood Ratio Test (LRT) is used.

edgeR: edgeR uses an exact test analogous to Fisher's exact test but adapted for overdispersed count data (variance of the counts for a given gene across replicates is significantly larger than the average count for that gene) or a LRT to test for significant difference in expression across the conditions.

Introduction to R:

Savio has R already installed. Let’s check out which version it is:

```
$ R --version
```

Savio replies:

```
R version 4.4.3 (2025-02-28) -- "Trophy Case”
```

Let’s create a Conda environment with latest version of R:

```
$ conda create -n dge_env r-base -y
```

Update R:

```
$ conda update r-base -y
```

Without sudo privileges, you cannot update R to a later version than what is on Savio.

```
$ conda activate dge_env
```

Check that R has been installed properly:

```
$ R --version
R version 4.4.3 (2025-02-28) -- "Trophy Case”
```

Run R:

```
$ R
```

Savio replies:
R version 4.4.3 (2025-02-28) -- "Trophy Case" etc
>   (the R cursor is ">")

To quit R, use either quit() or q():
> quit()

Savio replies:
Save workspace image? [y/n/c]: n

Restart R and get your working directory:
> getwd()

Set working directory:
> setwd("/global/scratch/users/your-username/path/to/directory")

Caution: (smart quotes) " vs (R quotes) "

We want to always use R quotes when using R. If you get an error after pasting a command, most likely it is due to not using R quotes.

The featureCounts output, gene_counts.txt, is not a large file, so you don't need Savio—you can run R on your own computer using your terminal window or RStudio. RStudio does not run on Savio.

To use RStudio on your computer/laptop, you need to install R first and then RStudio:

https://posit.co/download/rstudio-desktop/

Click on "Download and Install R"
Then, click on "Download RStudio Desktop for MACOS 13+" or whatever the system for your computer is. If you have an old version of R on your computer, these steps will update R as well as RStudio.

You can continue using Savio by invoking R in the Conda environment dge_env.

You can install R on your computer and use the same R commands described here.

Or, start up RStudio and open a new file on RStudio and either paste the commands into a new file or download the mouse_dge.R file from GitHub:

To download mouse_dge.R from GitHub, either use your computer's download mechanism or open a terminal window on your computer and paste this command:

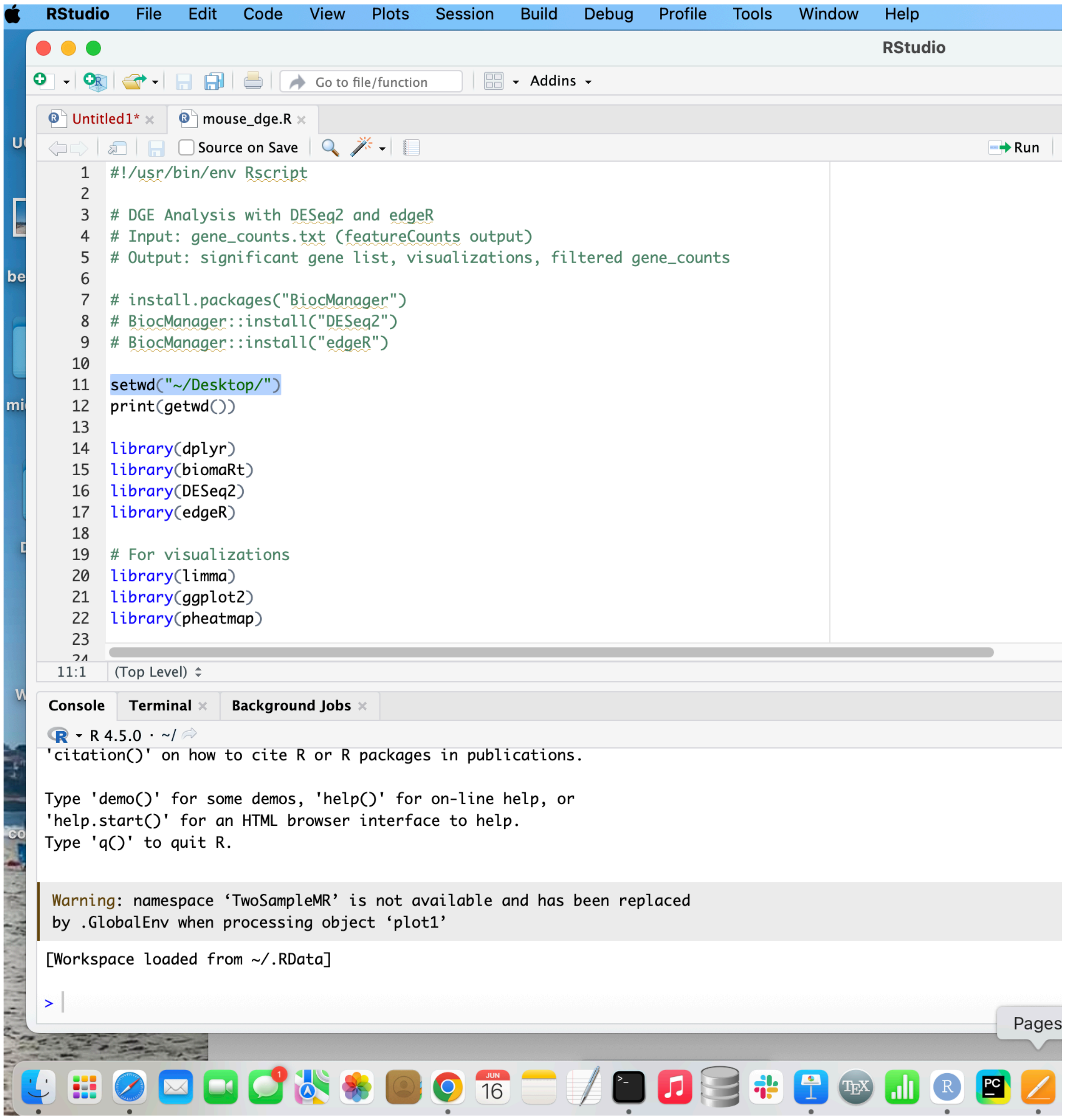
$ cd Desktop (this will cause the mouse_dge.R file to be downloaded onto your desktop)

$ wget -O mouse_dge.R https://raw.githubusercontent.com/elinorv21/RNA-Seq_workshop/main/mouse_dge.R   (all one line)

To run R commands in an R file:

Option A. Using RStudio:

Open the file mouse_dge.R in RStudio and run the commands there. One way to run the commands is to highlight the desired code and click run in the RStudio code window above the Console window. The Console window is the bottom screen and the file with the code is the top screen:



```
#!/usr/bin/env Rscript

# DGE Analysis with DESeq2 and edgeR
# Input: gene_counts.txt (featureCounts output)
# Output: significant gene list, visualizations, filtered gene_counts

# install.packages("BiocManager")
# BiocManager::install("DESeq2")
# BiocManager::install("edgeR")

setwd("~/Desktop/")
print(getwd())

library(dplyr)
library(biomaRt)
library(DESeq2)
library(edgeR)

# For visualizations
library(limma)
library(ggplot2)
library(pheatmap)
```

```
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Warning: namespace ‘TwoSampleMR’ is not available and has been replaced
by .GlobalEnv when processing object ‘plot1’

[Workspace loaded from ~/.RData]

>
```

Figure 1. RStudio with the highlighted text “setwd(“~/Desktop”)”. I set my R working directory to Desktop so that it’s easy to find the downloaded file “mouse_dge.R”.

Option B. Using your computer’s terminal window:

You need to download the featureCounts output: gene_counts.txt. Use your preferred method for downloading a file from Savio. I downloaded the data to my desktop for ease of finding the file.

Download and edit mouse_dge.R as appropriate (to be discussed) and run:

```
$ chmod +x mouse_dge.R
$ ./mouse_dge.R
```

The code will stop at any errors.

-or-

```
$ R
> setwd("~/Desktop")  (or wherever you choose)
```

And, then paste the R script (line by line) from mouse_dge.R into your terminal. The pro of doing this is that you can see immediately if particular parts of the edited code work or not. Of course, you can run in a line-by-line style in RStudio as well.

Option C. Using Savio:

```
$ R
> setwd("/global/scratch/users/your-username")
```

And, paste the R script (line by line) from mouse_dge.R into Savio. The pro of doing this is that you can see immediately if particular parts of the edited code work or not.

You can also run mouse_dge.R in the terminal window on Savio in the same way you use the terminal window of your computer/laptop. Run the R code on the command line in Savio using the same two commands used for your local command line (desktop location using a terminal window).

Differential Gene Expression Analysis

DESeq2, edgeR, and limma (an older software) results can be used to generate several useful diagnostic and visualization plots:

MA plots: Visualize the log2 fold change against the mean of normalized counts, highlighting differentially expressed genes.

Volcano plots: Gives the big picture of significant vs non-significant genes and also acts as a quality check: If most points lie along the x-axis could be a result of a very small sample size or over-conservative estimation of the dispersion.

Heatmaps: Can visualize the expression patterns of differentially expressed genes across samples. The focus is on the significant genes as a heat map of all genes is unwieldy.

PCA plots: Help assess the overall similarity and differences between samples based on the transformed gene expression data.

MA plots:

"M" equals the log ratio between two given conditions, and "A" equals the mean log intensity (abundance). The M = 0 horizontal line has points symmetrically scattered about it (rather than about a diagonal) when the data is normalized (no technical non-biological issues - no bias), meaning that most genes haven't changed between the two conditions. The few genes exhibiting change reflect a biological difference. For a given gene, the distance from the M=0 line indicates its abundance. It is expected that a few genes but not the entire genomics dataset is a good distance from the M = 0 line.

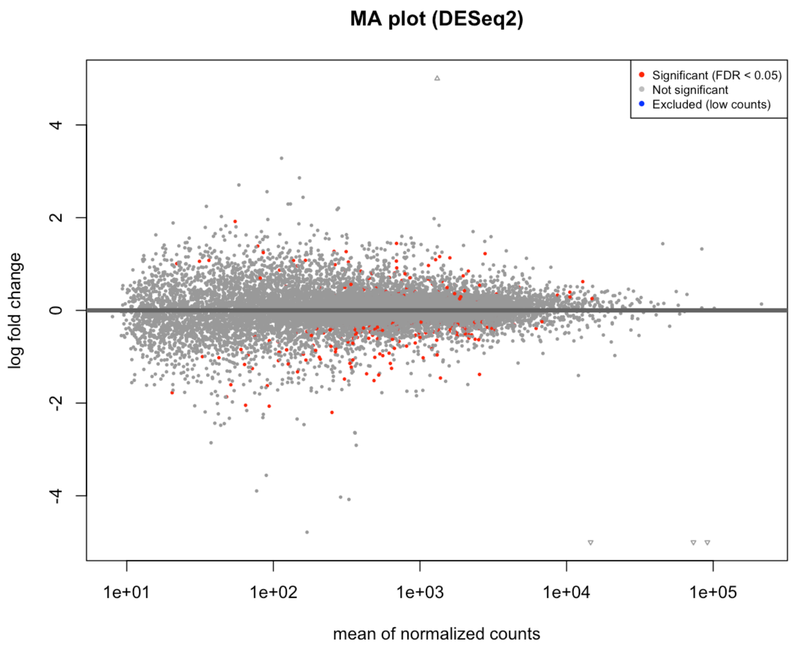


Figure 2. MA plot showing the significant genes (red points), non-significant genes (grey points), and excluded values (blue).

The majority of genes are clustered around the M = 0 line, particularly for genes with higher average expression (right side of the plot). This is expected, as most genes are not typically differentially expressed between conditions.

There's a wider spread of log fold changes for genes with lower average expression (left side of the plot). This is a common phenomenon in RNA-seq data; genes with low counts are inherently more variable, leading to less precise fold change estimates. This is why filtering low-count genes is often beneficial.

Significant Genes:

You may have a good number of significant points, indicating many genes are statistically significant (FDR < 0.05). These significant genes are distributed across a range of average expression values, but they appear to be more concentrated in the mid-to-higher range of average expression.

There are both up-regulated (red points above y=0) and down-regulated (red points below y=0) significant genes, indicating changes in both directions.

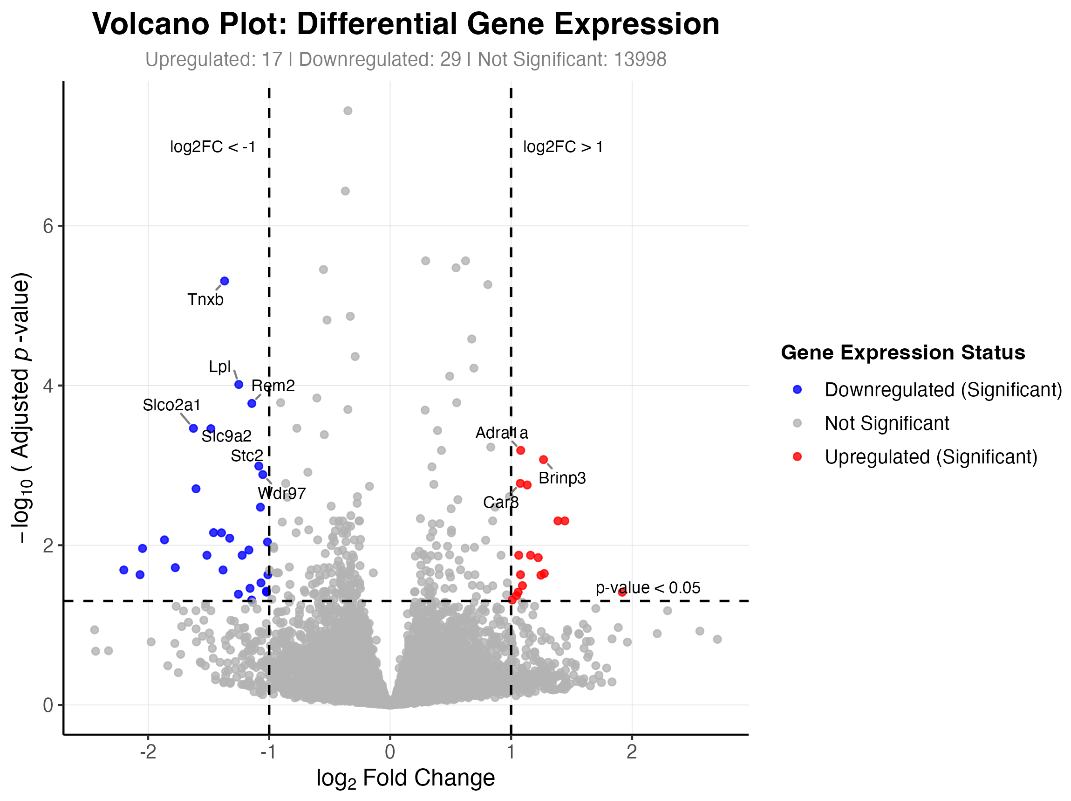
Notice that for low mean counts (left side), even if some points are far from 0, they are less likely to be significant (remain gray). This highlights the importance of filtering or the DESeq2 model handling of low counts.

Volcano plots:

A volcano plot is a scatter plot used in bioinformatics and genomics to visualize the results of differential expression analysis, such as comparing gene expression between two conditions (e.g., baseline vs. stressed).

The x-axis is the $\log_2$ fold change and measures how much a gene's expression changes between two conditions. The y-axis is the p-value and measures the statistical significance of the $\log_2$ fold change.

Points with large fold changes and high statistical significance appear at the top corners of the scatter plot (resembling a volcano's crater). Volcano plots are helpful for displaying statistical significance (-log10(p-value)) and magnitude of change (log2(FC)) for the proteins of interest.



So, for example, the higher position on the y-axis has a point, the more statistically significant it

is; the greater/lesser the value on the x-axis for the point, the greater/lesser the change in expression (upregulated/downregulated) between the two conditions being studied.
Figure 3. Volcano Plot of gene expression, showing upregulated genes (red points) and downregulated genes (blue points) and non-significant genes (grey points). Significant means that the absolute value of log2(fold change) for a gene is greater than or equal to 1 and the adjusted p-value is less than 0.05.

Heatmaps:

Visualizes the expression patterns of differentially expressed genes across samples.

How to interpret:

Each row is a gene (gene symbol or Ensemble ID). The value in each cell has been VST (variance stabilized) transformed (VST (DESeq2) or cpm(log=TRUE) (edgeR)), and plus also standardized (mean = 0, standard deviation = 1). "Red" color means higher-than-average for that gene and "blue" color means lower-than-average. Each column corresponds to a sample (each sample is one of the conditions or levels (e.g., treatment vs. control if two levels) in the experiment and one of the replicates).

For each row (gene) you compute a Z-score: Z_{gene} = (X - mu)/sigma, where X is the transformed (VST and standardized) expression for that gene in a given sample, mu is the mean of that gene across all samples, and sigma = standard deviation of that gene across all samples. Then, each column (sample) is the scaled expression vector for that sample: (Z_{gene1}, Z_{gene2}, …, Z_{geneN}), with N is the number of total genes being considered.

Clustering in the dendrogram is accomplished by the Euclidean distance of the scaled expression vectors meaning the distance between each pair of samples. For example, dist(sample1, sample2) = squareroot[($Z_{gene1\_for\_sample1}$ - $Z_{gene1\_for\_sample2}$)^2 + ($Z_{gene2\_for\_sample1}$ - $Z_{gene2\_for\_sample2}$)^2 + … + ($Z_{geneN\_for\_sample1}$ - $Z_{geneN\_for\_sample2}$)^2]. Samples with highly similar expression patterns have small distances and merge early in the tree (In a hierarchical-clustering dendrogram, the length of the vertical connector line that joins two branches is proportional to the distance between them).

The heat-map shows which genes co-vary and which samples behave similarly. Interpret the color direction per gene, not across genes. Strong, clean clusters validate the DE contrast and guide the biological story (e.g. respiratory-electron-transport genes are suppressed by the drug).

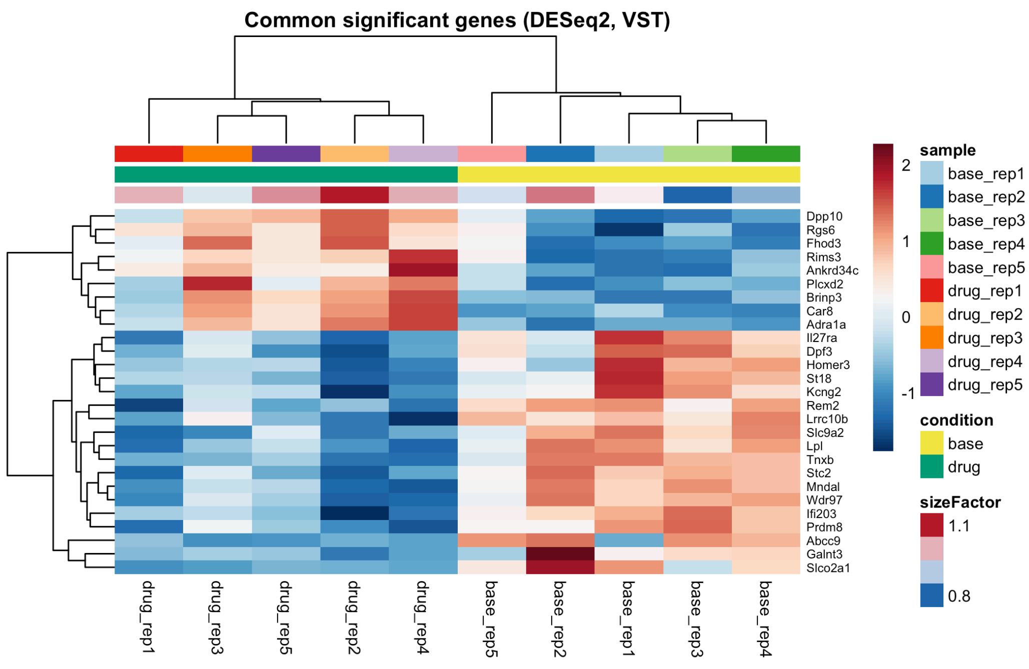


Figure 4. Heatmap from mouse samples for two conditions (non-paired design).

To interpret this specific heat map, consider the following:

1. Upper module (e.g. Dpp10, Rgs6, Plxcd2 …) is red in base and blue in drug meaning down-regulated by drug
Lower module (e.g. Lpl, Tnxb, Ifi203 …) is blue in base and red in drug meaning up-regulated by drug.

2. Perfect separation: all five drug replicates cluster together; all five base replicates form the other branch meaning treatment effect dominates expression variance.

3. Robust treatment signature: Replicates cluster by condition with zero mixing → high biological or technical reproducibility and strong drug effect on these genes.

4. Bidirectional response: Roughly half the genes are suppressed and half induced by the drug.

5. Candidate pathways: Up-regulated set contains lipid-metabolism (Lpl), stress-response (Stc2, Mndal) and ECM/remodelling (Tnxb). Down-regulated set includes neuronal signalling (Rgs6, Rims3, Dpp10) and ion transport (Kcng2, Slc9a2).

While Euclidean distance is a valid distance metric, for calculating **co-expression** between genes, correlation coefficients (especially Pearson) are overwhelmingly preferred because they focus on the *pattern* of expression rather than just the absolute differences in magnitude.

Heatmap Details:

Raw Data: Gene Counts

- **What it is:** This is the direct output from RNA-seq alignment and quantification. For each gene in each sample, you have an integer count representing the number of sequencing reads that mapped to that gene.

- **"Gene expression value":** At this stage, counts are a *proxy* for gene expression. Higher counts generally mean higher expression, but they are heavily influenced by factors like library size (total reads in a sample) and gene length.

Normalization (e.g., DESeq2's size factors):

- **Purpose:** Raw counts cannot be directly compared across samples because samples have different sequencing depths (library sizes). A gene might have a higher count in one sample just because that sample was sequenced more deeply, not because the gene is actually more expressed. Normalization aims to correct for these technical biases.

- **How it works (DESeq2's estimateSizeFactors):** DESeq2 calculates "size factors" for each sample. These factors represent the relative sequencing depth and RNA composition of each sample. To compare counts between samples, the raw counts are effectively divided by these size factors (or more precisely, used in the statistical model).

The "Normalization (e.g., by size factors)" step is performed by downstream statistical analysis packages specifically designed for differential expression analysis of RNA-seq data. The most common ones are:

- **DESeq2:** This is exactly where the estimateSizeFactors() function comes from. It calculates the size factors to account for library size differences and uses a robust method to normalize the raw counts.
- **edgeR:** This package uses a similar concept of normalization factors (e.g., TMM normalization) to adjust for differences in library size and RNA composition.
- **limma-voom (older R package):** While voom transforms the data for linear modeling, it also incorporates normalization (often using TMM or other methods from edgeR or DESeq2).

VST-Transformation (Variance Stabilizing Transformation):

- **Purpose:** RNA-seq count data exhibits a strong mean-variance relationship: genes with higher mean counts also tend to have higher variance. This non-constant variance makes it challenging for some downstream analyses (like clustering, PCA, and heatmaps that rely on Euclidean distances) which assume homoscedasticity (constant variance).
- **How it works:** VST is a mathematical transformation (specifically, a generalization of the logarithm) applied to the *normalized* counts. It aims to stabilize the variance across the entire range of expression values, making the data more homoscedastic. In DESeq2, vst() or rlog() functions perform this.
- **"Gene expression value":** This is where you get continuous, roughly homoscedastic "gene expression values" suitable for plotting on a heatmap and for distance-based analyses like clustering. These values are typically on a log2-like scale, meaning that differences in VST values correspond to fold changes in expression. They are adjusted for library size and are now on a scale where variance is more stable across different expression levels.

Row Scaling / Z-scoring (for Heatmap Visualization):

- **Purpose:** As discussed previously, even after VST, genes can have vastly different average expression levels. Row scaling allows you to visually compare the *relative change* of each gene *within itself* across samples.
- **How it works:** For each gene (row), its VST-transformed values across all samples are transformed into Z-scores: (VST_value - mean(VST_values_for_that_gene)) / sd(VST_values_for_that_gene).
- **"Gene expression value":** These are now Z-scores. They are no longer directly representing "counts" or "normalized counts." Instead, they represent how many standard deviations away from the gene's own average expression a particular sample's expression is. This is the final value plotted on your heatmap.

PCA plots:

Principal Component Analysis (PCA) is a dimensionality reduction technique that projects high-dimensional data onto a lower-dimensional space while maximizing variance.

Interpretation of Two-Dimensional PCA:

Composition of the Plot:

X-axis (PC1): First principal component, representing the highest variance in the dataset.

Y-axis (PC2): Second principal component, representing the second highest variance orthogonal to PC1.

Example: Biological Interpretation in Alternative Splicing Analysis:

If PC1 primarily separates control and treatment samples, it suggests that the largest source of splicing variability is the treatment effect.

If PC2 separates samples based on individual variability rather than condition, it may indicate the presence of batch effects or outliers.

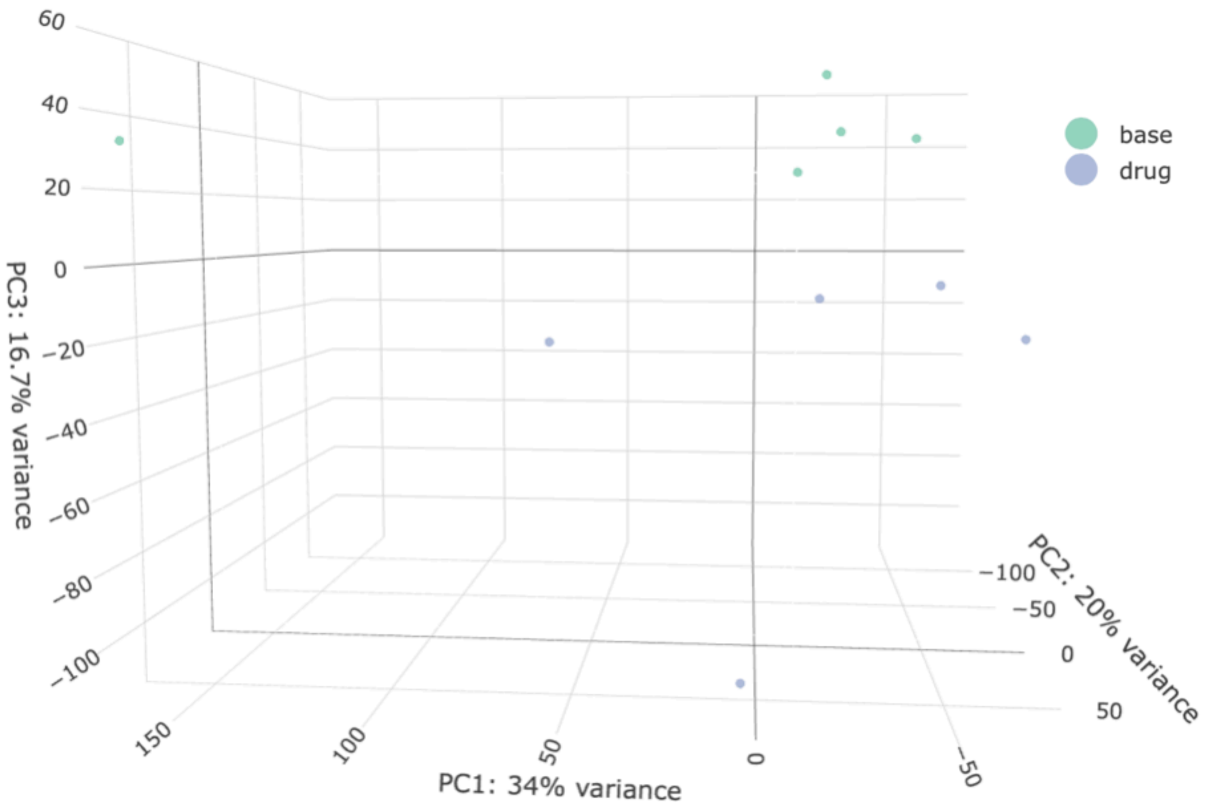


Figure 5. Three-dimensional PCA plot showing approximately two well-separated clusters, namely the baseline samples (green) and the treatment (drug) samples (purple).

Summary of Differential Gene Expression Analysis using R Code and R Packages:

A. Install DESeq2 and edgeR (DESeq2 and edgeR are R packages) on Savio, or your own computer (using a terminal window or RStudio). How to install these R packages will be discussed.

B. Run mouse_dge.R on Savio, or your own computer (either using a terminal window or RStudio):

mouse_dge.R input: featureCounts output (gene_counts.txt)

Summary of code:

1. Clean up row names, column names, and unwanted columns; Rename column names to reflect the experiment.

2. Restrict to protein-coding genes (21,773 genes).

3. Build a paired or unpaired design matrix and metadata.

4. Remove genes with low counts.

5. Run the DESeq2 algorithm on your data and output all genes with adjusted p-values and log2(fold change) information, and all significant differentially expressed genes (with adjusted p-values and log2(fold change) information); significant means that a new list of genes is created such that their adjusted p-values < 0.05 and log2(fold change) >= 1.00.

6. Run the edgeR algorithm on your data and output all genes with adjusted p-values and log2(fold change) information, and all significant differentially expressed genes (with adjusted p-values and log2(fold change) information); significant means that a new list of genes is created such that their adjusted p-values < 0.05 and log2(fold change) >= 1.00.

7. The intersection of DESeq2 significant genes and edgeR significant genes is found.

8. The following visualizations are constructed for the DESeq2 and edgeR data:
   a) MA plots,
   b) PCA plots (both 2-dimensional and 3-dimensional),
   c) Heatmaps (restricted to the common significant genes),
   d) Volcano plots.

mouse_dge.R output:

1. Lists of genes (includes their adjusted p-values and log2(fold change) information for both algorithms),
2. List of significant genes computed according to the DESeq2 algorithm and also the edgeR algorithm,
3. List of significant genes common to both DESeq2 and edgeR,
4. Visualizations (described above).

How to install R packages:

You only need to install a package once & there are two ways to install:

```
> if (!requireNamespace("BiocManager", quietly = TRUE)) install.packages("BiocManager")
```

Then, install the desired R packages in one of two ways (for both options, you need to install the packages in R not in bash - note the R cursor is used not the bash cursor $ - see below):

Option A.

```
> BiocManager::install(“edgeR")  # installs the edgeR package       -or-
```

Option B.

```
> install.packages(“edgeR”) # installs the edgeR package
```

Note: You need to load all the R library packages for every R session: (“#” means “comment”)

Contents of mouse_dge.R:

Bolded code indicates where you may want to modify the code.

```
> library(dplyr)
> library(biomaRt) # for Ensembl IDs
> library(DESeq2) # differential gene expression analysis algorithm
> library(edgeR)  # differential gene expression analysis algorithm

# For visualizations
> library(limma)
> library(ggplot2)
> library(pheatmap)
```

Load data: **Edit the location so that R can find your data:**

```
> gene_counts <- read.table("~/Desktop/mouse_data/fcounts_results/gene_counts.txt", header
= TRUE, sep = "\t", row.names = 1)
```

Step 1. Clean up column names

Display column names:

```
> print(colnames(gene_counts))  # to view what your column names currently are as well as
seeing that you have some unwanted columns
```

Drop the unnecessary columns:

```
> df <- gene_counts[, !(names(gene_counts) %in% c("Chr", "Start", "End", "Strand",
"Length"))]

> current_columns <- colnames(df) # current column names

> cat("Current columns of the data, gene_counts: \n")
> print(current_columns)
```

Rename your column names:

Remove the “mouse_data.star_results.renamed.” from the column names
Note: “.” Is a special character in R, so we need to write “\\.” Instead of only “.”:

```
> new_columns <- gsub("mouse_data\\.star_results\\.renamed\\.", "", current_columns)
```

Remove “.bam” from column names

```
> newer_columns <- gsub("\\.bam", "", new_columns)
```

Note: You may have to insert this additional code, depending on whether you previously cleaned your filenames or not when creating your BAM files:

Remove ".Aligned.sortedByCoord.out.bam" from the column names:
```
> newer_columns <- gsub("\\.Aligned\\.sortedByCoord\\.out\\.bam", "", newer_columns)
```

Assign these new column names to your data:

```
> colnames(df) <- newer_columns
```

Step 2. Filter gene_counts data

Filter for mouse protein-coding genes only:

```
> ensembl <- useMart(biomart = "ensembl") # connect to the Ensembl database using biomaRt
> ensembl <- useDataset("mmusculus_gene_ensembl", mart = ensembl)

> ensembl_gene_ids <- rownames(df) # Retrieve gene biotypes for your Ensembl IDs
> gene_info <- getBM(filters = "ensembl_gene_id_version",
            values = ensembl_gene_ids,
            attributes = c("ensembl_gene_id_version", "gene_biotype"),
            mart = ensembl)
```

Filter for counts of protein-coding genes

```
> protein_coding_genes <- gene_info %>% filter(gene_biotype == "protein_coding")
> protein_coding_counts <- df[rownames(df) %in% protein_coding_genes$ensembl_gene_id, ]
```

Check your table dimensions: You should have about 21773 rows and the correct number of columns representing your samples:

```
> dim(protein_coding_counts) # [1] 21773 (genes)   10 (samples) # I had 10 samples so 10 cols
```

Quality Control: You may decide to not do this or to change the parameters (min_count, sample_threshold)

Remove genes with low counts (decreases noise):

Code results: Only genes with at least 5 counts present in at least 25% of samples are included in differential expression analyses.

```
> df <- protein_coding_counts
> min_count <- 5
> sample_threshold <- ceiling(0.25 * ncol(df))  # 25% of samples
> keep_genes <- rowSums(df >= min_count) > sample_threshold
> filtered_counts_df <- df[keep_genes, ]

> cat("Filtered gene counts\n")
> head(filtered_counts_df)  # to inspect a portion of your data as it is now

> getwd()  # check what is your working directory
```

You may want to save your results so far, in case you want to take a break from the analysis and come back later:

```
> saveRDS(filtered_counts_df, “~/Desktop/mouse_data/filtered_gene_counts.rds”)

##################
# DGE with DESeq2 #
##################
```

Note: If you quit R and now you’re restarting R, you need to declare all the libraries again:

```
> library(dplyr)
Etc

> print(getwd())
> setwd("~/Desktop")

> filtered_counts <- readRDS(“~/Desktop/mouse_data/filtered_gene_counts.rds”)
```

Step 1: Build metadata file (for an unpaired design)
Note: Unpaired design means mice samples are independent of each other.

```
> num_samples <- 5 # Assuming each condition has 5 samples (for example, 5 baseline replicates and 5 treatment replicates)

> sample <- colnames(filtered_counts)
```

Note: The following “condition” is a vector whose (ordered) entries are: base, base, base, base, base, drug, drug, drug, drug, drug. Should be the same order as your gene_counts.txt columns. You may need to modify “condition” to match your sample types and their order in your gene_counts.txt file.

```
> condition <- factor(c(rep("base", num_samples), rep("drug", num_samples))) # Same order and sample type as the gene_counts.txt columns. Note: You can have more than two conditions (see below for code addition to compare any two conditions)

> metadata <- data.frame(sample, condition) # unpaired design: “mouse” is omitted since it’s an unpaired design

> metadata$condition <- relevel(metadata$condition, ref = "base")  # Set the reference condition
```

Print the metadata:

```
> print(metadata)
```

Note 1: You can have uneven samples: For example, here are 5 replicates for baseline and 10 replicates for drug (treatment):

```
> condition <- factor(c(rep("base", 5), rep("drug", 10)))
```

Note 2: Constructing the metadata for data having a paired design: Treatment was performed on the same mice as the control.

```
> num_samples <- 4 (Each condition has 4 samples)
> sample <- colnames(df)
> ctrl <- rep("ctrl", times = num_samples)
> trt <- rep("trt", times = num_samples)

> condition <- factor(c(trt, ctrl))
> mouse <- factor(c(paste0("rep", 1:num_samples), paste0("rep", 1:num_samples)))
> metadata <- data.frame(sample, condition, mouse)
```

Set reference condition:

```
> metadata$condition <- relevel(metadata$condition, ref = “ctrl")
```

Step 2. Run DESeq2 algorithm

```
> unpaired_design_filtered <- DESeqDataSetFromMatrix(countData = filtered_counts,
                                        colData = metadata,
                                        design = ~ condition) # unpaired design (if paired, include
mouse: design = ~ mouse + condition)

> deseq2_results_filtered <- DESeq(unpaired_design_filtered, test = "Wald")
> deseq2_results_table_filtered <- results(deseq2_results_filtered)
```

If you have more than two conditions, adjust the condition vector appropriately. Then, to extract the contrasts between any two conditions, use the following code (example):

DrugA vs Control (already a coefficient):

```
> res_drugA_vs_ctrl <- results(deseq2_results_filtered, name = "condition_drugA_vs_control")
```

DrugB vs Control (already a coefficient):

```
> res_drugB_vs_ctrl <- results(deseq2_results_filtered, name = "condition_drugB_vs_control")
```

DrugB vs DrugA (needs an explicit contrast):

```
> res_drugB_vs_drugA <- results(deseq2_results_filtered, contrast = c("condition", "drugB",
“drugA”))   (all one line)
```

Always inspect resultsNames(deseq2_results_filtered) so you know the exact coefficient strings/names:

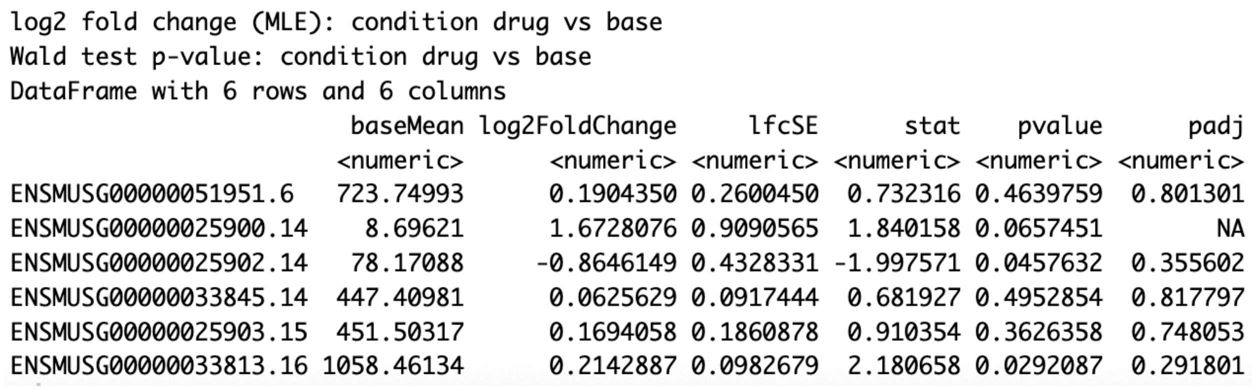
```
> resultsNames(dds)
```

Step 3. Create a list of the Significant Genes (via DESeq2)

DESeq2 uses the mean of the normalized counts (the baseMean column in the results table) as a filter statistic. The idea is that genes with very low overall counts have less power to be detected as differentially expressed so adjusted p-values will show NA. Also genes with zero or very low variance will show NA for their adjusted p-value.

Print headers and first six rows of DESEq2 results:

```
> cat("DESeq2 results\n")
> head(deseq2_results_table_filtered)
```



```
log2 fold change (MLE): condition drug vs base
Wald test p-value: condition drug vs base
DataFrame with 6 rows and 6 columns
                       baseMean log2FoldChange     lfcSE      stat    pvalue      padj
                      <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ENSMUSG00000051951.6   723.74993      0.1904350 0.2600450  0.732316 0.4639759  0.801301
ENSMUSG00000025900.14    8.69621      1.6728076 0.9090565  1.840158 0.0657451        NA
ENSMUSG00000025902.14   78.17088     -0.8646149 0.4328331 -1.997571 0.0457632  0.355602
ENSMUSG00000033845.14  447.40981      0.0625629 0.0917444  0.681927 0.4952854  0.817797
ENSMUSG00000025903.15  451.50317      0.1694058 0.1860878  0.910354 0.3626358  0.748053
ENSMUSG00000033813.16 1058.46134      0.2142887 0.0982679  2.180658 0.0292087  0.291801
```

Figure 2. Results from running the DESeq2 algorithm on the gene_counts.txt data.

Filter the genes which have an adjusted p-value (Benjamini-Hochberg) < 0.05 and log2(fold change) >= 1.00:

```
> sig_genes_deseq2 <- deseq2_results_table_filtered[ !
is.na(deseq2_results_table_filtered$padj) &
(deseq2_results_table_filtered$padj < 0.05) &
(abs(deseq2_results_table_filtered$log2FoldChange) >= 1.00), ]
```

Print number of significant genes (adjusted p-value < 0.05 and log2(Fold Change) >= 1.00):

```
> print("Number of significant genes (adjusted p-value < 0.05 and log2(Fold Change) >= 1.00)")
> nrow(sig_genes_deseq2)
```

Save significant genes (results from DESeq2) to a file **(you may want to modify the file location):**

```
> saveRDS(sig_genes_deseq2, "~/Desktop/mouse_data/sig_genes_deseq2.rds")
```

Save the DESeq2 results (all genes, not just the significant genes (omit genes with missing adjusted p-value; **you may want to modify the file location**):

```
> deseq2_results_table <-
deseq2_results_table_filtered[is.na(deseq2_results_table_filtered$padj), ]  # all one line

> saveRDS(deseq2_results_table, "~/Desktop/mouse_data/res_deseq2.rds") # Save DESeq2
output

##################
# DGE with edgeR #
##################
```

Step 1. Build design matrix for edgeR

Create an DGE list for edgeR from the gene_counts data:

```
> dge <- DGEList(counts = filtered_counts)
```

Ensure your metadata has the sample names as row names:

```
> rownames(metadata) <- metadata$sample # use metadata from deseq2
```

Check if the sample names in dge$samples are in metadata:

```
> print(all(rownames(dge$samples) %in% rownames(metadata))) # Expect TRUE
```

If all sample names are present, you can add the 'condition' information from metadata to dge$samples, ensuring the order matches:

```
> dge$samples$condition <- metadata[rownames(dge$samples), "condition"]
```

For unpaired design:

```
> design_edger <- model.matrix(~ condition, data = dge$samples)
```

Design matrix column names:

```
> print(colnames(design_edger)) # "coef" = entry on the right (e.g., "conditiondrug")
```

Step 2. Filter and normalize data

Filter data using CPM (counts per million) (similar to filtering in DESeq2):

```
> min_samples <- 3 # gene must have CPM >= 1 in at least 30% of the samples to be kept
> cpm_dge <- cpm(dge) # Compute CPM (Counts Per Million)
> keep_cpm <- rowSums(cpm_dge >= 1) >= min_samples
```

Additionally, remove genes too weakly expressed to be testable given your experimental design and library sizes:

```
> keep_fbe <- filterByExpr(dge, design = design_edger)
```

```
# Option 1: Uses “|” for OR (less strict filtering) (Option 2 uses “&” for AND)
# keep_combined_or <- keep_cpm | keep_fbe
# dge <- dge[keep_combined_or, keep.lib.sizes = FALSE]
# dge$samples$lib.size <- colSums(dge$counts) # recalculate library sizes
# dge <- calcNormFactors(dge) # Normalization on dge
```

If you want to use Option 1 instead, uncomment the “#” lines and comment out the Option 3 lines - in other words, remove the “#” for Option 1 and add “#” to Option 3 lines.

Option 3: Recommended: Combines both by dropping low information rows:

```
> keep_fbe <- filterByExpr(dge,
                    design   = design_edger,
                    min.count = 10,      # CPM of 1 in a 10 M-read library
                    min.prop  = 0.3)     # 30 % of libraries
```

```
> dge <- dge[keep_fbe, keep.lib.sizes = FALSE]
```

```
> dge$samples$lib.size <- colSums(dge$counts) # recalculate library sizes
```

> dge <- calcNormFactors(dge) # Normalization of dge using TMM (Trimmed Means of M Values, Reference: Robinson & Oshlack, 2010 (https://genomebiology.biomedcentral.com/articles/10.1186/gb-2010-11-3-r25)

Step 3: Run the edgeR algorithm

Estimating dispersion

Dispersion is the coefficient of the quadratic term employed when estimating the variance used in the negative-binomial model; this model is used to fit each gene's counts. If dispersion is under-estimated, we get more false positives in DGE analysis. If over-estimated, we get more false negatives.

We need a statistical model to decide whether our gene counts support a real expression change between conditions or could have arisen by chance. We need to perform a hypothesis test to see if the observed raw counts are biologically true or if the counts occurred by chance. So, we need a theoretical probability model of the expected counts to compare with our observed counts.

Estimate dispersion (related to the variance of the negative binomial distribution):

```
> dge <- estimateDisp(dge, design = design_edger) # unpaired design
```

(For a paired design: dge <- estimateDisp(dge, design = design_edger, block = metadata$mouse))

Perform the hypothesis test:

Use either Model 1 or 2 for modeling the data. Use Model 1 unless Model 1 gives 0 or few results (you'll need to uncomment the Model 1 code and comment the Model 2 code in order to run Model 1). Model 2 gives a higher false positive rate:

```
# Model 1:
#
# fit <- glmQLFit(dge, design_edger) # unpaired design; paired design: fit <- glmQLFit(dge,
design_edger, block = metadata$mouse)
# qlf <- glmQLFTest(fit, coef = "conditiondrug")
# top_tags_all <- topTags(qlf, n = Inf)$table

# Number of genes with FDR < 0.05 and log2(fold change) >= 1
# cat("Number of genes with FDR < 0.05 and log2(fold change) >= 1: \n")
# sum(top_tags_all$FDR < 0.05 & abs(top_tags_all$logFC) >= 1)
# = 0
```

Model 2:

```
> fit_lrt <- glmFit(dge, design_edger) # unpaired design
> lrt <- glmLRT(fit_lrt, coef = "conditiondrug")

> top_tags_lrt <- topTags(lrt, n = Inf)$table
```

Number of genes with FDR < 0.05 and log2(fold change) >= 1:

```
> cat("Number of genes with FDR < 0.05 and log2(fold change) >= 1: \n")
> sum(top_tags_lrt$FDR < 0.05 & abs(top_tags_lrt$logFC) >= 1)
# > 0
```

Step 4: Create a list of the Significant Genes (via edgeR) using Model 2

```
> top_tags_2 <- as.data.frame(top_tags_lrt) # top_tags_lrt is Model 2
> sig_genes_edger <- subset(top_tags_2, FDR < 0.05 & abs(logFC) >= 1)
```

Save the results of edgeR and the significant genes **(you may want to modify the file location):**

```
> saveRDS(sig_genes_edger, “~/Desktop/mouse_data/sig_genes_edger.rds”) # significant
genes via edgeR (list of genes)

> top_tags_lrt <- top_tags_lrt[!is.na(top_tags_lrt$FDR), ]

> saveRDS(top_tags_lrt, “~/Desktop/mouse_data/res_edger.rds”) # results of edgeR

# Step 5. Create a robust significant gene list using DESeq2 and edgeR gene lists

> library(tibble)   # rownames_to_column()
> deseq2_df <- as.data.frame(sig_genes_deseq2) %>% rownames_to_column("gene")
> edger_df  <- as.data.frame(sig_genes_edger)  %>% rownames_to_column("gene")

> common_sig_genes <- intersect(deseq2_df$gene, edger_df$gene)

> print("number of significant genes common to both DESeq2 and edgeR: ")
> length(common_sig_genes)

> saveRDS(common_sig_genes, “~/Desktop/mouse_data/common_sig_genes.rds”)

##############
# Visualizations #
##############

# MA plot
# M = log₂ fold change (on the y-axis)
# A = mean expression (on the x-axis, in log scale)

#######################################
# MA plot for DESeq2 # deseq2_results_filtered #
#######################################

> res <- results(deseq2_results_filtered)
> sum(!is.na(res$padj) & res$padj < 0.05)  # gives number of significant

> table(is.na(res$padj)) # TRUE means low counts exist
```

Plot with custom point size and highlight order **(you may want to modify the file location)**:

```
> png(“~/Desktop/mouse_data/DESeq2_MAplot.png”, width = 1200, height = 1000, res = 150)

> DESeq2::plotMA(res, alpha = 0.05, ylim = c(-5, 5),
            main = "MA plot (DESeq2)", colSig = "red")
        legend("topright",
        legend = c("Significant (FDR < 0.05)",
                "Not significant",
                "Excluded (low counts)"),
        col = c("red", "grey", "blue"),
        pch = 16, cex = 0.7)
> dev.off()

> sum(res$log2FoldChange > 5, na.rm = TRUE)   # too high - represented by an upside-down
triangle
> sum(res$log2FoldChange < -5, na.rm = TRUE) # too low - represented by an upside-down
triangle

####################################
# MA plot for edgeR # qlf/lrt or top_tags_lrt #
####################################

> library(limma)
> library(ggplot2)
```

Optional: Ranked genes for edgeR (not needed for MA plot)

```
# > res_edger <- topTags(qlf, n = Inf)$table # Model 1 # topTags implicitly sorts the results
> top_tags_lrt <- topTags(lrt, n = Inf)$table # Model 2
```

Calculate logCPM from the original dge object:

```
> logCPM_dge <- cpm(dge, log = TRUE)

# > res_edger$AvgLogCPM <- rowMeans(logCPM_dge[rownames(res_edger), ]) # Model 1
> top_tags_lrt$AvgLogCPM <- rowMeans(logCPM_dge[rownames(top_tags_lrt), ]) # Model 2

# Create the MA plot for edgeR - Model 1
# > ma_plot_edger <- ggplot(res_edger, aes(x = AvgLogCPM, y = logFC)) +
#       geom_point(aes(color = ifelse(abs(logFC) >= 1 & FDR < 0.05, "significant", "not
significant")), size = 2) +
#       scale_color_manual(values = c("significant" = "red", "not significant" = "grey")) +
#       geom_hline(yintercept = c(-1, 0, 1), linetype = "dashed", color = "black") +
#       labs(title = "MA Plot", x = "Average LogCPM", y = "Log2 Fold Change", color =
"Significance") +
#       theme_bw() +
#       geom_vline(xintercept = mean(res_edger$AvgLogCPM), linetype = "dotted", color =
"blue")
```

# **(you may want to modify the file location)**

#> ggsave("~/Desktop/mouse_data/ma_plot_edger.png", plot = ma_plot_edger, width = 8, height = 6, dpi = 300)

Create the MA plot for edgeR - Model 2:

```
> ma_plot_edger_2 <- ggplot(top_tags_lrt, aes(x = AvgLogCPM, y = logFC)) +
        geom_point(aes(color = ifelse(abs(logFC) >= 1 & FDR < 0.05, "significant", "not
significant")), size = 0.5) +
        scale_color_manual(values = c("significant" = "red", "not significant" = "grey")) +
        geom_hline(yintercept = c(-1, 0, 1), linetype = "dashed", color = "black") +
        labs(title = "MA Plot (edgeR)", x = "Average LogCPM", y = "Log2 Fold Change", color =
"Significance") +
        theme_bw() +
        geom_vline(xintercept = mean(top_tags_lrt$AvgLogCPM), linetype = "dotted", color =
"blue")
```

(you may want to modify the file location)
> ggsave("~/Desktop/mouse_data/edgeR_MAplot.png", plot = ma_plot_edger_2, width = 8, height = 6, dpi = 300)

###############
### PCA Plots ###
###############

##
# PCA Plot for DESeq2 # deseq2_results_filtered #
##

1. Variance stabilizing transformation (VST) or regularized log transformation (rlog)
   This helps to account for the mean-variance relationship in RNA-Seq data

# deseq2_results_table <-- deseq2_results_filtered with no NAs

```
> vsd <- vst(deseq2_results_filtered, blind = FALSE) # VST: function from DESeq2 library
```

# Alternatively, you could use:
# rld <- rlog(dds, blind = FALSE)

2. Perform PCA (prcomp is from the R stats package and automatically loaded when you start R)

```
> pcaData_deseq2 <- prcomp(t(assay(vsd)), center = TRUE, scale. = TRUE)
```

3. Create a data frame for plotting

```
> percentVar_deseq2 <- pcaData_deseq2$sdev^2 / sum(pcaData_deseq2$sdev^2)

> pcaDf_deseq2 <- data.frame(PC1 = pcaData_deseq2$x[, 1],
                     PC2 = pcaData_deseq2$x[, 2],
                     group = colData(deseq2_results_filtered)$condition) # Assuming your
condition column is named 'condition'
```

4. Generate the PCA plot using ggplot2

```
> pcaPlot_deseq2 <- ggplot(pcaDf_deseq2, aes(x = PC1, y = PC2, color = group)) +
	geom_point(size = 4) +
	xlab(paste0("PC1: ", round(percentVar_deseq2[1] * 100), "% variance")) +
	ylab(paste0("PC2: ", round(percentVar_deseq2[2] * 100), "% variance")) +
	ggtitle("PCA Plot (DESeq2 - VST)") +
	theme_bw() +
	theme(plot.title = element_text(hjust = 0.5))
```

(you may want to modify the file location)

```
> ggsave("~/Desktop/mouse_data/DESeq2_PCA_2D_plot.png", plot = pcaPlot_deseq2, width
= 8, height = 6, dpi = 300)
```

```
##########################################
# 3-dim PCA plot for DESeq2 ## pcaData_deseq2 #
##########################################
```

```
> library(plotly)
> library(htmlwidgets)
```

1. Create the 3D scatter plot using plotly

Calculate the percentage of variance explained by PC3:

```
> percentVar_pc3_deseq2 <- pcaData_deseq2$sdev[3]^2 / sum(pcaData_deseq2$sdev^2) *
100 # all one line
```

Create the 3D scatter plot using plotly:

```
> pcaPlot_3d_deseq2 <- plot_ly(
	data = pcaDf_deseq2,
	x = ~PC1,
	y = ~PC2,
	z = ~pcaData_deseq2$x[, 3],
	color = ~group,
	size = 4,
	marker = list(symbol = "circle"),
	text = ~paste(
		"Sample:",
		rownames(pcaDf_deseq2),
		"<br>Group:",
		group,
		"<br>PC1:",
		round(PC1, 2),
		"<br>PC2:",
		round(PC2, 2),
		"<br>PC3:",
		round(pcaData_deseq2$x[, 3], 2)
	),
	hoverinfo = "text",
	type = "scatter3d",  # Explicitly set the plot type
	mode = "markers"     # Explicitly set the mode
) %>%
        layout(
                scene = list(
                        xaxis = list(title = paste0("PC1: ", round(percentVar_deseq2[1] * 100), "%
variance")),
                        yaxis = list(title = paste0("PC2: ", round(percentVar_deseq2[2] * 100), "%
variance")),
                        zaxis = list(title = paste0("PC3: ", round(percentVar_pc3_deseq2, 2), "%
variance"))
                ),
        title = "3D PCA Plot (DESeq2 - VST)"
  )
```

(you may want to modify the file location)

```
> saveWidget(pcaPlot_3d_deseq2, "mouse_data/deseq2_PCA_3D.html")
```

##
# PCA plot for edgeR # dge - 2-dimensional plot #
##

1. Use dge after normalization and dispersion calculation

```
> v <- voom(dge, design = design_edger, plot = TRUE)
```

2. Perform PCA on the normalized and transformed data

```
> pcaData_edger <- prcomp(t(v$E), center = TRUE, scale. = TRUE)
```

3. Create a data frame for plotting

```
> percentVar_edger <- pcaData_edger$sdev^2 / sum(pcaData_edger$sdev^2)

> pcaDf_edger <- data.frame(PC1 = pcaData_edger$x[, 1],
                      PC2 = pcaData_edger$x[, 2],
                      group = dge$samples$condition) # Assuming your condition info is in
dgeList$samples$condition
```

4. Generate the PCA plot using ggplot2

```
> pcaPlot_edger <- ggplot(pcaDf_edger, aes(x = PC1, y = PC2, color = group)) +
        geom_point(size = 4) +
        xlab(paste0("PC1: ", round(percentVar_edger[1] * 100), "% variance")) +
        ylab(paste0("PC2: ", round(percentVar_edger[2] * 100), "% variance")) +
        ggtitle("PCA Plot (edgeR - voom)") +
        theme_bw() +
        theme(plot.title = element_text(hjust = 0.5))
```

(you may want to modify the file location)

```
> ggsave("~/Desktop/mouse_data/edgeR_PCA_2D_plot.png", plot = pcaPlot_edger, width = 8,
height = 6, dpi = 300)
```

##
# 3-dimensional plot for PCA with edgeR # pcaData_edger #
##

Assuming the 2-dim plot was completed, compute percentage variance explained by PC3:

```
> percentVar_pc3_edger <- pcaData_edger$sdev[3]^2 / sum(pcaData_edger$sdev^2) * 100
```

Create a dataframe for 3D PCA plot:

```
> pcaDf_3d_edger <- data.frame(
	PC1 = pcaData_edger$x[, 1],
	PC2 = pcaData_edger$x[, 2],
	PC3 = pcaData_edger$x[, 3],
	group = dge$samples$condition,
	sample = rownames(dge$samples) # Add sample names for hover text
	)
```

Create 3D plot with plotly:

```
> pcaPlot_3d_edger <- plot_ly(
	data = pcaDf_3d_edger,
	x = ~PC1,
	y = ~PC2,
	z = ~PC3,
	color = ~group,
	size = 4,
	marker = list(symbol = "circle"),
	text = ~paste("Sample:", sample, "<br>Group:", group,
	      "<br>PC1:", round(PC1, 2), "<br>PC2:", round(PC2, 2), "<br>PC3:", round(PC3, 2)),
	hoverinfo = "text",
	type = "scatter3d",
	mode = "markers"
	) %>%
	layout(
		scene = list(
			xaxis = list(title = paste0("PC1: ", round(percentVar_edger[1] * 100), "% 
variance")),
			yaxis = list(title = paste0("PC2: ", round(percentVar_edger[2] * 100), "% 
variance")),
			zaxis = list(title = paste0("PC3: ", round(percentVar_pc3_edger, 2), "% 
variance"))
		),
		title = "3D PCA Plot (edgeR - voom)"
	)
```

Save the 3D plot **(you may want to modify the file location)**:

```
> saveWidget(pcaPlot_3d_edger, “~/Desktop/mouse_data/edgeR_PCA_3D.html”)
```

```
############################################################
# Heatmap for the common genes via DESeq2 # unpaired_design_filtered #
############################################################
```

Use DESeq2's variance-stabilized transformed data (vsd); Don’t use raw counts.

```
> vsd <- vst(unpaired_design_filtered, blind = FALSE) # Assuming 'unpaired_design_filtered' is
the DESeqDataSet object

> mat_deseq2 <- assay(vsd)[rownames(assay(vsd)) %in% common_sig_genes, ]
> mat_scaled_deseq2 <- t(scale(t(mat_deseq2))) # center and scale the data

> pheatmap(mat_scaled_deseq2,
       cluster_rows = TRUE,
       cluster_cols = TRUE,
       show_rownames = TRUE, # Or FALSE if you have too many genes
       show_colnames = TRUE,
       annotation_col = as.data.frame(colData(vsd)), # Sample annotations
       main = "Heatmap of Common Significant Genes (DESeq2 - VST)”,
       # you may want to modify the file location
       filename = “~/Desktop/mouse_data/
common_sig_genes_deseq2_heatmap_ensembl.png”) # all one line
```

The above produces a heatmap with ensembl names for the genes

The following code produces a heatmap with geneSymbols instead:

```
# BiocManager::install("org.Mm.eg.db")
# BiocManager::install("AnnotationDbi")

> library(org.Mm.eg.db)
> library(AnnotationDbi)

> current_heatmap_genes_deseq2 <- rownames(mat_scaled_deseq2)
```

Remove version numbers from Ensembl IDs:

```
> current_heatmap_genes_unversioned_deseq2 <- sub("\\.\\d+$", "",
current_heatmap_genes_deseq2)

> gene_symbols_mapped_deseq2 <- mapIds(org.Mm.eg.db,
                            keys = current_heatmap_genes_unversioned_deseq2,
                            keytype = "ENSEMBL",
                            column = "SYMBOL",
                            multiVals = "first") # Keep this for handling multiple mappings

> final_gene_names_deseq2 <- ifelse(is.na(gene_symbols_mapped_deseq2),
                          # Fallback to original versioned Ensembl ID
                          current_heatmap_genes_deseq2,
                          gene_symbols_mapped_deseq2)
```

Ensure the names of this vector match the original versioned Ensembl IDs, which allows correct lookup when assigning to rownames:

```
> names(final_gene_names_deseq2) <- current_heatmap_genes_deseq2
```

Assign these as row names to the heatmap matrix:

```
> rownames(mat_scaled_deseq2) <- final_gene_names_deseq2
> pheatmap(mat_scaled_deseq2,
        cluster_rows = TRUE,
        cluster_cols = TRUE,
        show_rownames = TRUE,
        show_colnames = TRUE,
        annotation_col = as.data.frame(colData(vsd)),
        main = "Heatmap of Common Significant Genes - DESeq2",
        fontsize_main = 2,
        margins = c(top = 10, right = 5, bottom = 5, left = 5),
        # you may want to modify the file location
        filename = "~/Desktop/mouse_data/
common_sig_genes_deseq2_heatmap_symbols.png")
```

```
#####################################################
# Publication-quality heatmap via DESeq2 ## mat_scaled_deseq2 #
#####################################################
------------------------------------------------------------
0.  Preprocessing
------------------------------------------------------------
```

```
> library(AnnotationDbi)
> library(org.Mm.eg.db)      # ↔ your organism: Mouse

> mat_scaled <- mat_scaled_deseq2              # rows = genes, cols = samples
```

Column annotation:

```
> anno_col <- as.data.frame(colData(vsd)) |>
  dplyr::select(sizeFactor, condition, sample)   # keep only needed
```

Set colors for annotation bars:

```
> library(RColorBrewer)

> anno_colours <- list(
        sizeFactor = colorRampPalette(c("#2166ac", "white", "#b2182b"))(100),
        condition  = c(base = "#F0E442", drug = "#009E73"),
        sample     = setNames(brewer.pal(n = nrow(anno_col), "Paired"),
                     rownames(anno_col))
        )
```

```
------------------------------------------------------------
1.  HEAT-MAP
------------------------------------------------------------
```

```
> library(pheatmap)
```

Pick a balanced red-blue palette:

```
> pal <- colorRampPalette(rev(brewer.pal(11, "RdBu")))(100)

> p <- pheatmap(
        mat_scaled,
        color                  = pal,
        cluster_rows          = TRUE,
        cluster_cols          = TRUE,
        clustering_distance_rows = "euclidean",
        clustering_distance_cols = "euclidean",
        clustering_method     = "complete",
        show_rownames         = TRUE,
        show_colnames         = TRUE,
        annotation_col        = anno_col,
        annotation_colors     = anno_colours,
        annotation_names_col  = FALSE,     # hide long titles
        fontsize              = 8,         # general text
        fontsize_row          = 6,         # smaller gene labels
        fontsize_col          = 8,
        border_color          = NA,        # no cell borders
        main                  = "Common significant genes (DESeq2, VST)",
        legend                = TRUE
        )
```

----
2. SAVE (vector PDF + high-res PNG)

Vector-quality PDF: **(you may want to modify the file location)**

```
> pdf("~/Desktop/mouse_data/pub_common_sig_genes_deseq2_heatmap_symbols.pdf",
width = 7, > height = 4.5)
> print(p)
> dev.off()
```

300-dpi PNG for slides: **(you may want to modify the file location)**

```
> png("~/Desktop/mouse_data/pub_common_sig_genes_deseq2_heatmap_symbols.png",
   width = 2100, height = 1350, res = 300)
> print(p)
> dev.off()
```

```
########################################
# Publication-quality heatmap via edgeR ## dge #
########################################
```

Load necessary libraries:

```
> library(org.Mm.eg.db)
> library(AnnotationDbi)
```

Use voom-transformed and normalized logCPM values:

```
> v <- voom(dge, design_edger, plot = FALSE) # edgeR objects
> mat_edger <- v$E[rownames(v$E) %in% common_sig_genes, ]
> mat_scaled_edger <- t(scale(t(mat_edger))) #  Center and scale
```

Choose the annotations:

```
> anno_col_edger_modified <- dge$samples %>% dplyr::select(lib.size, condition)
```

This produces a heatmap with ensembl names for the genes:

```
> pheatmap(mat_scaled_edger,
        cluster_rows = TRUE,
        cluster_cols = TRUE,
        show_rownames = TRUE,
        show_colnames = TRUE,
        annotation_col = anno_col_edger_modified, # Sample annotations
        main = "Heatmap of Common Significant Genes (edgeR)",
        # you may want to modify the file location
        filename = "~/Desktop/mouse_data/common_sig_genes_edger_heatmap_ensembl.png")
```

Heatmap for edgeR with gene symbols (or Ensembl IDs if gene_symbol not found):

```
current_heatmap_genes_edger <- rownames(mat_scaled_edger)

# Remove version numbers from Ensembl IDs:
current_heatmap_genes_unversioned_edger <- sub("\\.\\d+$", "",
current_heatmap_genes_edger)

gene_symbols_mapped_edger <- mapIds(org.Mm.eg.db,
                        keys = current_heatmap_genes_unversioned_edger,
                        keytype = "ENSEMBL",
                        column = "SYMBOL")

final_gene_names_edger <- ifelse(is.na(gene_symbols_mapped_edger),
                          # Fallback to original versioned Ensembl ID
                          current_heatmap_genes_edger,
                          gene_symbols_mapped_edger)

# Ensure the names of this vector match the original versioned Ensembl IDs,
# which allows correct lookup when assigning to rownames:

names(final_gene_names_edger) <- current_heatmap_genes_edger

rownames(mat_scaled_edger) <- final_gene_names_edger

anno_col_edger_modified <- dge$samples %>% dplyr::select(lib.size, condition)
```

(you may want to modify the file location)

```
> pheatmap(mat_scaled_edger,
        cluster_rows = TRUE,
        cluster_cols = TRUE,
        show_rownames = TRUE,
        show_colnames = TRUE,
        annotation_col = anno_col_edger_modified, # Sample annotations
        main = "Heatmap of Common Significant Genes (edgeR)",
        filename = "~/Desktop/mouse_data/common_sig_genes_edger_heatmap_symbols.png")
```

##

```
# Volcano Plot for DESeq2 # deseq2_results_table_filtered #
################################################

> deseq2_volcano_df <- as.data.frame(deseq2_results_table_filtered)
> deseq2_volcano_df <- deseq2_volcano_df[complete.cases(deseq2_volcano_df), ]

> deseq2_volcano_df$significant <- ifelse(deseq2_volcano_df$padj < 0.05, "Significant", "Not
Significant") # all one line

> logFC_cutoff <- 1

> deseq2_volcano_df$sig_logFC <- ifelse(deseq2_volcano_df$padj < 0.05 &
abs(deseq2_volcano_df$log2FoldChange) >= logFC_cutoff, "Sig & LogFC",
deseq2_volcano_df$significant)
```

Create the Volcano Plot with ggplot2:

```
> volcano_plot_deseq2 <- ggplot(deseq2_volcano_df, aes(x = log2FoldChange, y =
-log10(padj))) +   # all one line
        geom_point(aes(color = sig_logFC), size = 1) + # Color by combined significance
        scale_color_manual(values = c("Significant" = "blue", "Not Significant" = "grey", "Sig &
LogFC" = "red")) +  # all one line
        geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "black") + #
Significance cutoff line
        geom_vline(xintercept = c(-logFC_cutoff, logFC_cutoff), linetype = "dashed", color =
"black") + # logFC cutoff lines
        labs(
                title = "Volcano Plot (DESeq2)",
                x = "Log2 Fold Change",
                y = "-Log10 (Adjusted p-value)",
                color = "Significance"
         ) +
        theme_bw() +
        theme(plot.title = element_text(hjust = 0.5))
```

Save the plot **(you may want to modify the file location):**

```
> ggsave("~/Desktop/mouse_data/DESeq2_Volcano_Plot.png", plot = volcano_plot_deseq2,
width = 8, height = 6, dpi = 300)

################################
# Volcano Plot for edgeR # top_tags_lrt #
################################

> edgeR_volcano_df <- as.data.frame(top_tags_lrt)

> edgeR_volcano_df <- edgeR_volcano_df[complete.cases(edgeR_volcano_df), ]
> edgeR_volcano_df$significant <- ifelse(edgeR_volcano_df$FDR < 0.05, "Significant", "Not
Significant")

> logFC_cutoff <- 1 # Or your chosen cutoff
> edgeR_volcano_df$sig_logFC <- ifelse(edgeR_volcano_df$FDR < 0.05 &
abs(edgeR_volcano_df$logFC) >= logFC_cutoff, "Sig & LogFC", edgeR_volcano_df$significant)
```

Create the Volcano Plot with ggplot2:

```
> volcano_plot_edger <- ggplot(edgeR_volcano_df, aes(x = logFC, y = -log10(FDR))) + # Use
logFC and FDR
        geom_point(aes(color = sig_logFC), size = 1) +
        scale_color_manual(values = c("Significant" = "blue", "Not Significant" = "grey", "Sig &
LogFC" = "red")) +
        geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "black") +
        geom_vline(xintercept = c(-logFC_cutoff, logFC_cutoff), linetype = "dashed", color =
"black") +
        labs(
                title = "Volcano Plot (edgeR)",
                x = "Log2 Fold Change",
                y = "-Log10 (FDR)",
                color = "Significance"
         ) +
        theme_bw() +
        theme(plot.title = element_text(hjust = 0.5))
```

Save the plot **(you may want to modify the file location):**

```
> ggsave("~/Desktop/mouse_data/edgeR_Volcano_Plot.png", plot = volcano_plot_edger, width
= 8, height = 6, dpi = 300)
```

```
##################################################
# publication quality of volcano plot: ### deseq2_volcano_df #
##################################################
```

Load necessary libraries:

```
> library(ggplot2)
> library(ggrepel) # For non-overlapping text labels
> library(org.Mm.eg.db)
> library(dplyr)
```

Getting gene names:

```
> deseq2_volcano_df$ensembl_id <- rownames(deseq2_volcano_df)

> current_volcano_genes_unversioned_deseq2 <- sub("\\.\\d+$", "",
deseq2_volcano_df$ensembl_id)

> deseq2_volcano_df$gene_symbol <- mapIds(
  x = org.Mm.eg.db,
  keys = current_volcano_genes_unversioned_deseq2,
  column = "SYMBOL",
  keytype = "ENSEMBL",
  multiVals = "first" # Keep this for handling multiple mappings
)

> deseq2_volcano_df <- deseq2_volcano_df %>%
  mutate(
    gene_symbol = ifelse(is.na(gene_symbol), ensembl_id, gene_symbol)
  )
```

Assuming deseq2_volcano_df is already prepared as per your existing code.
Ensure 'deseq2_volcano_df' contains:
- log2FoldChange
- padj (adjusted p-value)
- gene_name (or similar identifier for labeling, if desired)
- sig_logFC (the column you use for coloring, which defines significance categories)

--- Define cutoffs (from your existing code or chosen values) ---

```
> p_cutoff <- 0.05
> logFC_cutoff <- 1 # Example: 2-fold change
```

--- Re-categorize 'sig_logFC' for more descriptive colors ---

```
> deseq2_volcano_df <- deseq2_volcano_df %>%
  mutate(
    sig_logFC_category = case_when(
      padj < p_cutoff & log2FoldChange > logFC_cutoff ~ "Upregulated (Significant)",
      padj < p_cutoff & log2FoldChange < -logFC_cutoff ~ "Downregulated (Significant)",
      TRUE ~ "Not Significant"
    )
  )
```

--- Count significant genes for annotation ---

```
> num_up <- sum(deseq2_volcano_df$sig_logFC_category == "Upregulated (Significant)",
na.rm = TRUE)
> num_down <- sum(deseq2_volcano_df$sig_logFC_category == "Downregulated
(Significant)", na.rm = TRUE)
> num_not_sig <- sum(deseq2_volcano_df$sig_logFC_category == "Not Significant", na.rm =
TRUE)
```

--- Create the production-quality volcano plot ---

```
> volcano_plot_production <- ggplot(deseq2_volcano_df, aes(x = log2FoldChange, y =
-log10(padj))) +
        # Add points, colored by the new category
        geom_point(aes(color = sig_logFC_category), size = 1.5, alpha = 0.8) +

        # Define manual colors for the categories
        scale_color_manual(
                values = c(
                        "Upregulated (Significant)" = "red",
                        "Downregulated (Significant)" = "blue",
                        "Not Significant" = "grey70" # Use a lighter grey for less emphasis
                ),
                name = "Gene Expression Status" # More descriptive legend title
         ) +

        # Significance cutoff line (p-value = 0.05)
```

```
    geom_hline(yintercept = -log10(p_cutoff), linetype = "dashed", color = "black", linewidth =
0.6) +
    # Label the p-value cutoff line
    annotate("text", x = max(deseq2_volcano_df$log2FoldChange, na.rm = TRUE) * 0.95,
        y = -log10(p_cutoff) + 0.1,
        label = paste0("p-value < ", p_cutoff),
        hjust = 1, vjust = 0, size = 3) +

    # logFC cutoff lines (e.g., Fold Change > 2 or < 0.5)
    geom_vline(xintercept = c(-logFC_cutoff, logFC_cutoff), linetype = "dashed", color =
"black", linewidth = 0.6) +
    # Label the logFC cutoff line for up-regulated
    annotate("text", x = logFC_cutoff + 0.1,
        y = max(-log10(deseq2_volcano_df$padj), na.rm = TRUE) * 0.95,
        label = paste0("log2FC > ", logFC_cutoff),
        hjust = 0, vjust = 1, size = 3) +
    # Label the logFC cutoff line for down-regulated
    annotate("text", x = -logFC_cutoff - 0.1,
        y = max(-log10(deseq2_volcano_df$padj), na.rm = TRUE) * 0.95,
        label = paste0("log2FC < ", -logFC_cutoff),
        hjust = 1, vjust = 1, size = 3) +

    # Labels for title, axes, and legend
    labs(
        title = "Volcano Plot: Differential Gene Expression",
        subtitle = paste0(
        "Upregulated: ", num_up,
        " | Downregulated: ", num_down,
        " | Not Significant: ", num_not_sig
    ),
    x = expression(log[2]~"Fold Change"), # Use expression for subscript
    # Use expression for subscript and italic p
    y = expression(-log[10]~"("~Adjusted~italic("p")~"-value)")  ) +

    # Use a classic theme and customize elements for a clean look
    theme_classic() + # A clean, common theme
    theme(
        plot.title = element_text(hjust = 0.5, face = "bold", size = 16), # Center and bold title
        plot.subtitle = element_text(hjust = 0.5, size = 10, color = "grey50"), # Centered
subtitle for counts
        axis.title = element_text(size = 12, face = "bold"),
        axis.text = element_text(size = 10),
        legend.title = element_text(size = 11, face = "bold"),
        legend.text = element_text(size = 10),
        panel.grid.major = element_line(color = "grey90", linewidth = 0.2), # Lighter grid
lines
        panel.grid.minor = element_blank(), # Remove minor grid lines
        axis.line = element_line(color = "black", linewidth = 0.5), # Ensure axis lines are
visible
        plot.margin = unit(c(0.5, 0.5, 0.5, 0.5), "cm") # Adjust plot margins
    ) +

    # Add labels for top N most significant genes
    # You would need a 'gene_name' column in your deseq2_volcano_df
```

```
        # Adjust 'top_n' as needed
        geom_text_repel(
                data = deseq2_volcano_df %>%
                filter(sig_logFC_category != "Not Significant") %>%
                arrange(padj) %>%
                head(10), # Label top 10 significant genes
                aes(label = gene_symbol), # Assuming 'gene_symbol' is your gene identifier column
                size = 3,
                box.padding = unit(0.35, "lines"),
                point.padding = unit(0.3, "lines"),
                min.segment.length = 0.2, # Draw segments if needed
                segment.color = 'grey50'
         )
```

(you may want to modify the file location)
ggsave("mouse_data/DESeq2_Volcano_Plot_pub.png", plot = volcano_plot_production, width = 8, height = 6, dpi = 300)

##
######## PUBLICATION-QUALITY MA PLOT ########### deseq2_volcano_df
##

Load necessary libraries

```
> library(ggplot2)
> library(ggrepel) # For non-overlapping text labels
> library(dplyr)   # For %>% pipe operator and mutate/case_when
```

--- Define cutoffs (same as for volcano plot for consistency) ---

```
> p_cutoff <- 0.05
> logFC_cutoff <- 1 # Example: 2-fold change, adjust as needed
```

--- Categorize genes for coloring based on significance and fold change ---

This ensures that your 'deseq2_volcano_df' has the 'sig_status_ma' column
It requires 'log2FoldChange', 'padj', and 'baseMean' columns to be present.

```
> deseq2_volcano_df <- deseq2_volcano_df %>%
        mutate(
                sig_status_ma = case_when(
                        padj < p_cutoff & log2FoldChange > logFC_cutoff ~ "Upregulated
(Significant)",
                        padj < p_cutoff & log2FoldChange < -logFC_cutoff ~ "Downregulated
(Significant)",
                        TRUE ~ "Not Significant"
                )
        )
```

--- Count significant genes for plot subtitle annotation ---

```
> num_up_ma <- sum(deseq2_volcano_df$sig_status_ma == "Upregulated (Significant)", na.rm
= TRUE)
```

```
> num_down_ma <- sum(deseq2_volcano_df$sig_status_ma == "Downregulated (Significant)",
na.rm = TRUE)

> num_not_sig_ma <- sum(deseq2_volcano_df$sig_status_ma == "Not Significant", na.rm =
TRUE)

--- Create the production-quality MA plot ---

> ma_plot_production <- ggplot(deseq2_volcano_df, aes(x = log10(baseMean), y =
log2FoldChange)) +
        # Add points, colored by their significance status
        geom_point(aes(color = sig_status_ma), size = 1.5, alpha = 0.8) +

        # Define manual colors for each category
        scale_color_manual(
                values = c(
                        "Upregulated (Significant)" = "red",
                        "Downregulated (Significant)" = "blue",
                        "Not Significant" = "grey70" # Use a lighter grey for less emphasis
                ),
        name = "Gene Expression Status" # More descriptive legend title
        ) +

  # Horizontal line at 0 fold change (no differential expression)
  geom_hline(yintercept = 0, linetype = "solid", color = "black", linewidth = 0.6) +

  # Fold Change cutoff lines (e.g., at log2FC = 1 and -1)
  geom_hline(yintercept = logFC_cutoff, linetype = "dashed", color = "darkgreen", linewidth =
0.6) +
  geom_hline(yintercept = -logFC_cutoff, linetype = "dashed", color = "darkgreen", linewidth =
0.6) +

  # Annotate the fold change cutoff lines directly on the plot
  annotate("text", x = min(log10(deseq2_volcano_df$baseMean), na.rm = TRUE),
         y = logFC_cutoff + 0.1,
         label = paste0("log2FC = ", logFC_cutoff),
         hjust = 0, vjust = 0, size = 3, color = "darkgreen") +
  annotate("text", x = min(log10(deseq2_volcano_df$baseMean), na.rm = TRUE),
         y = -logFC_cutoff - 0.1,
         label = paste0("log2FC = ", -logFC_cutoff),
         hjust = 0, vjust = 1, size = 3, color = "darkgreen") +

  # Labels for title, axes, and legend
  labs(
        title = "       MA Plot: Differential Gene Expression vs. Mean Expression",
        subtitle = paste0(
                "Upregulated: ", num_up_ma,
                " | Downregulated: ", num_down_ma,
                " | Not Significant: ", num_not_sig_ma
         ),
        x = expression(log[10]~"Mean Normalized Counts"), # Formats x-axis label with subscript
        y = expression(log[2]~"Fold Change")            # Formats y-axis label with subscript
        ) +
  # Apply a classic theme and customize elements for a clean, publication-ready look
  theme_classic() + # Provides a clean background and axis lines
  theme(
        plot.title = element_text(hjust = 0.5, face = "bold", size = 16), # Centered, bold title
        plot.subtitle = element_text(hjust = 0.5, size = 10, color = "grey50"), # Centered subtitle
for counts
        axis.title = element_text(size = 12, face = "bold"), # Bold axis titles
        axis.text = element_text(size = 10), # Adjust axis text size
        legend.title = element_text(size = 11, face = "bold"), # Bold legend title
        legend.text = element_text(size = 10), # Adjust legend text size
        panel.grid.major = element_line(color = "grey90", linewidth = 0.2), # Lighter major grid
lines
        panel.grid.minor = element_blank(), # Remove minor grid lines for cleaner look
        axis.line = element_line(color = "black", linewidth = 0.5), # Ensure axis lines are visible
        plot.margin = unit(c(0.5, 0.5, 0.5, 0.5), "cm") # Adjust plot margins
  ) +

  # Label the top 10 most significant genes (by adjusted p-value)
  # Uses ggrepel to prevent labels from overlapping
  geom_text_repel(
        data = deseq2_volcano_df %>%
        filter(sig_status_ma != "Not Significant") %>% # Only label significant genes
        arrange(padj) %>% # Sort by adjusted p-value to get the most significant first
        head(10), # Select the top 10 genes
        aes(label = gene_symbol), # Use the 'gene_symbol' column for labels
        size = 3.5, # Adjust label text size
        box.padding = unit(0.4, "lines"), # Padding around text
        point.padding = unit(0.3, "lines"), # Minimum distance from text to point
        min.segment.length = 0.2, # Draw segments to point if needed
        segment.color = 'grey50', # Color of the segment lines
        max.overlaps = Inf # Allows all labels to be shown, even if they would overlap slightly.  )
```

(you may want to modify the file location)

```
ggsave(“mouse_data/DESeq2_MA_Plot_pub.png”, plot = ma_plot_production, width = 8,
height = 6, dpi = 300)
```