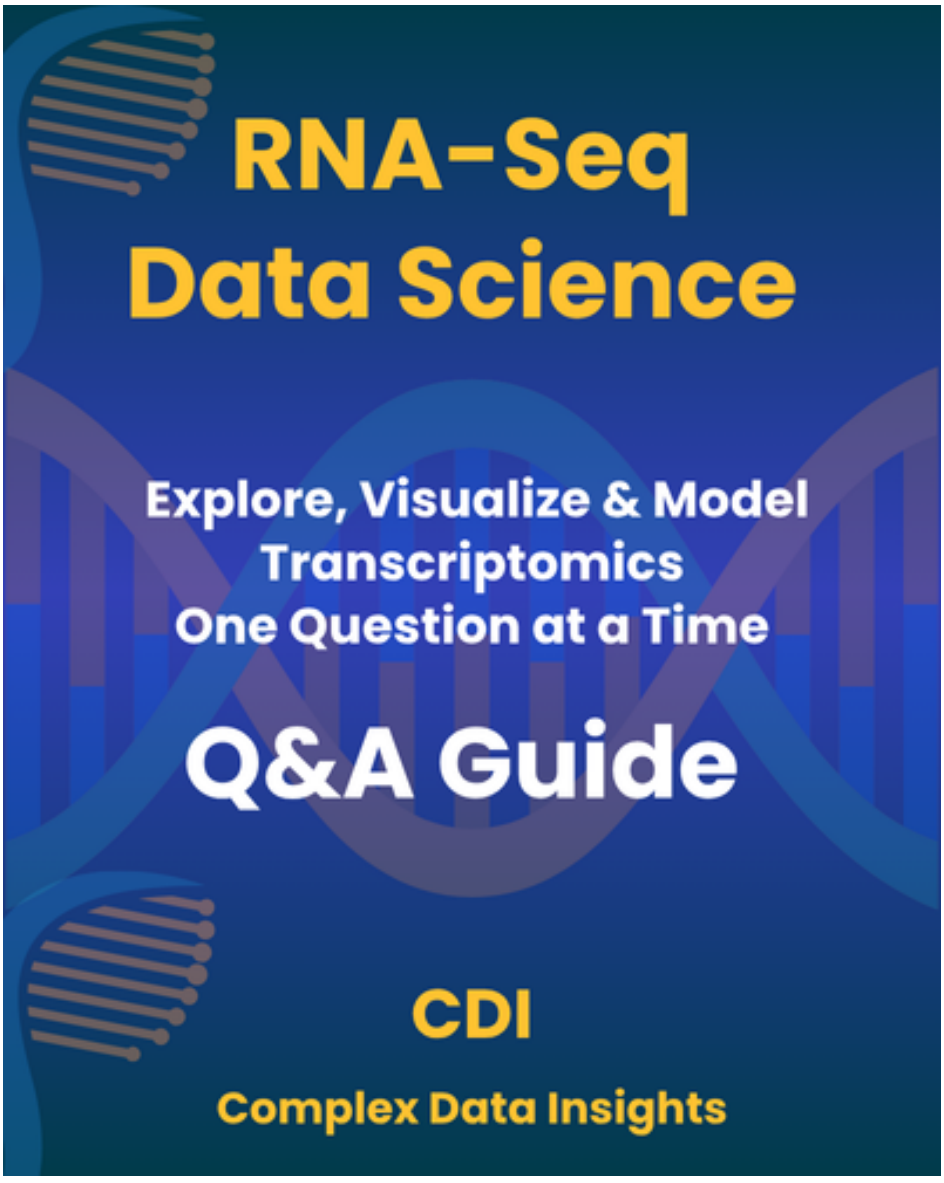
# □ RNA-Seq Data Science




Last updated: June 29, 2025

# Contents

# □ Welcome to the RNA-Seq Q&A Guide

This guide is your hands-on companion for learning and applying RNA sequencing (RNA-Seq) data analysis — one question at a time.

You'll explore each step of the RNA-Seq pipeline using real tools, reproducible workflows, and well-commented code. From quality control and quantification to differential expression and biological interpretation, this guide shows how scripting, statistics, and bioinformatics come together in practice.

Whether you're a student, researcher, or self-taught enthusiast, you'll gain confidence using Python, R, shell scripting, and reproducible workflows — including tools like **DESeq2**, **Salmon**, **edgeR**, **FastQC**, **Snakemake**, and more.

Each Q&A includes a clear explanation, relevant code in both Python and R when applicable, and builds toward real-world problem solving. You're not just learning RNA-Seq — you're learning to think like a modern data-driven bioinformatician.

# Part I

# DATA PREPARATION

# Q&A: 1

# How do you create a project data folder for storing RNA-Seq inputs and outputs?

## 1.1 Explanation

A well-organized project structure improves reproducibility and clarity in RNA-Seq workflows. We'll begin by creating a `data/` directory where all input files (counts, metadata) and output files (DE results, visualizations) will be stored.

This setup mimics a real-world analysis structure and helps you track your data as you progress through the Q&A guide.

## 1.2 R Code

```
#  Create the data folder if it doesn't exist
if (!dir.exists("data")) dir.create("data", showWarnings = FALSE)

message("  data/ folder created.")
```

 **Takeaway:** Always start your RNA-Seq project by creating a consistent directory structure. A `data/` folder keeps inputs and outputs organized throughout your analysis.

# Q&A: 2

# How do you generate synthetic RNA-Seq counts and metadata using R?

## 2.1 Explanation

In this step, we generate **synthetic RNA-Seq data** with known differences between conditions. This allows you to simulate differential expression, save the data into a `data/` folder, and later analyze it using DESeq2.

We simulate:

- 1000 genes
- 14 samples (11 Positive, 3 Negative)
- Upregulation in the top 30 genes for Negative samples

- Downregulation in the next 30 genes for Negative samples

 This setup mimics a real-world study design and ensures that the resulting **volcano and MA plots** clearly show the V-shape pattern of differential expression.

## 2.2 R Code

```r
library(tidyverse)

set.seed(42)

#  Create output directory
if (!dir.exists("data")) dir.create("data", recursive = TRUE)

#  Simulation settings
n_genes <- 1000
n_pos <- 11
n_neg <- 3
n_de_up <- 30
n_de_down <- 30
gene_ids <- paste0("Gene", seq_len(n_genes))

# Simulate Positive group (baseline expression)
counts_pos <- matrix(rnbinom(n_genes * n_pos, mu = 100, size = 1), nrow = n_genes)

# Simulate Negative group
counts_neg <- matrix(rnbinom(n_genes * n_neg, mu = 100, size = 1), nrow = n_genes)

#  Upregulate top 30 genes in Negative samples
counts_neg[1:n_de_up, ] <- counts_neg[1:n_de_up, ] + rnbinom(n_de_up * n_neg, mu = 400,

#  Downregulate next 30 genes in Negative samples
counts_neg[(n_de_up + 1):(n_de_up + n_de_down), ] <- rnbinom(n_de_down * n_neg, mu = 10,

# Combine counts
count_matrix <- cbind(counts_pos, counts_neg)
colnames(count_matrix) <- paste0("Sample", seq_len(n_pos + n_neg))
rownames(count_matrix) <- gene_ids

#  Metadata
metadata <- tibble(
```

```
  Sample = colnames(count_matrix),
  condition = c(rep("Positive", n_pos), rep("Negative", n_neg))
)

#  Save to data/
write_csv(as.data.frame(count_matrix) |> rownames_to_column("gene"), "data/demo_counts.
write_csv(metadata, "data/demo_metadata.csv")

#  Preview first 5 genes × 5 samples
as.data.frame(count_matrix)[1:5, 1:5]
```

| | Sample1 | Sample2 | Sample3 | Sample4 | Sample5 |
|---|---|---|---|---|---|
| Gene1 | 186 | 175 | 175 | 54 | 54 |
| Gene2 | 45 | 101 | 54 | 48 | 287 |
| Gene3 | 38 | 12 | 214 | 92 | 86 |
| Gene4 | 293 | 0 | 172 | 35 | 85 |
| Gene5 | 215 | 377 | 72 | 18 | 3 |

```
#  Preview metadata
head(metadata, 5)
```

| Sample | condition |
|---|---|
| Sample1 | Positive |
| Sample2 | Positive |
| Sample3 | Positive |
| Sample4 | Positive |
| Sample5 | Positive |

 Takeaway: This simulation creates a realistic expression pattern where some genes are clearly upregulated or downregulated in one condition. This structure is ideal for learning DE analysis, producing excellent MA and volcano plots, and testing downstream workflows.

# Q&A: 3

# How do you validate RNA-Seq input data before analysis using R?

## 3.1 Explanation

Before proceeding with differential expression analysis, it's essential to confirm that your input files are loaded correctly and match in structure. This includes:

- □ Ensuring all samples in the metadata are present in the count matrix
- □ Verifying that the matrix is numeric and genes are in rows
- □ Checking for NA or non-finite values

This validation step prevents downstream errors and ensures a smooth analysis.

## 3.2 R Code

```
library(tidyverse)

#  Load the count matrix and metadata
count_df <- read_csv("data/demo_counts.csv")
metadata <- read_csv("data/demo_metadata.csv")
```

```r
#  Inspect the data
glimpse(count_df)
```

```
Rows: 1,000
Columns: 15
$ gene     <chr> "Gene1", "Gene2", "Gene3", "Gene4", "Gene5", "Gene6", "Gene7"~
$ Sample1  <dbl> 186, 45, 38, 293, 215, 246, 99, 8, 38, 197, 28, 98, 93, 142, ~
$ Sample2  <dbl> 175, 101, 12, 0, 377, 376, 115, 13, 66, 82, 89, 14, 38, 114, ~
$ Sample3  <dbl> 175, 54, 214, 172, 72, 34, 74, 13, 88, 49, 223, 237, 315, 145~
$ Sample4  <dbl> 54, 48, 92, 35, 18, 53, 18, 118, 200, 146, 169, 38, 93, 64, 1~
$ Sample5  <dbl> 54, 287, 86, 85, 3, 81, 369, 96, 1, 12, 101, 41, 135, 102, 82~
$ Sample6  <dbl> 67, 74, 110, 173, 27, 180, 82, 22, 146, 8, 148, 70, 0, 432, 1~
$ Sample7  <dbl> 50, 42, 149, 50, 12, 46, 22, 192, 39, 55, 15, 47, 78, 53, 3, ~
$ Sample8  <dbl> 3, 54, 6, 142, 50, 312, 289, 264, 82, 244, 48, 208, 77, 167, ~
$ Sample9  <dbl> 37, 118, 124, 32, 68, 480, 72, 94, 96, 156, 54, 46, 196, 30, ~
$ Sample10 <dbl> 24, 28, 11, 198, 284, 86, 41, 21, 93, 51, 96, 175, 56, 229, 2~
$ Sample11 <dbl> 38, 42, 121, 138, 0, 46, 43, 66, 266, 14, 18, 96, 56, 9, 166,~
$ Sample12 <dbl> 361, 437, 388, 173, 590, 221, 334, 108, 583, 193, 428, 826, 3~
$ Sample13 <dbl> 238, 314, 324, 984, 1208, 287, 350, 438, 170, 346, 857, 986, ~
$ Sample14 <dbl> 266, 586, 589, 889, 650, 189, 659, 190, 650, 172, 106, 434, 1~
```

```r
glimpse(metadata)
```

```
Rows: 14
Columns: 2
$ Sample    <chr> "Sample1", "Sample2", "Sample3", "Sample4", "Sample5", "Samp~
$ condition <chr> "Positive", "Positive", "Positive", "Positive", "Positive", ~
```

```r
# Check that all sample names in metadata are in counts
all(metadata$Sample %in% colnames(count_df))  # Should return TRUE
```

```
[1] TRUE
```

```r
# Set gene names as rownames and confirm dimensions
counts <- count_df |>
  column_to_rownames("gene") |>
  as.matrix()

stopifnot(all(metadata$Sample %in% colnames(counts)))
stopifnot(ncol(counts) == nrow(metadata))
```

 **Takeaway:** A quick check of structure, dimension, and sample consistency ensures your data is clean and ready for analysis. This step can help catch common mistakes early, such as misaligned sample names or non-numeric entries.

# Q&A: 4

# How do you perform differential gene expression analysis using DESeq2 in R?

## 4.1 Explanation

With a validated count matrix and metadata, we can now run **DESeq2** to identify differentially expressed genes between two conditions.

DESeq2 performs:

1. □ Size factor estimation (normalization)
2. □ Dispersion estimation
3. □ Negative binomial model fitting
4. □ Wald test for significance
5. □ Adjusted p-values via FDR

This produces a results table with log2 fold changes, p-values, and adjusted p-values.

## 4.2 R Code

```
library(tidyverse)
library(DESeq2)
```

```r
set.seed(42)  # For reproducibility

#  Load data
count_df <- read_csv("data/demo_counts.csv")
metadata <- read_csv("data/demo_metadata.csv")

#  Prepare DESeq2 inputs
counts <- count_df |>
  column_to_rownames("gene") |>
  as.matrix()

dds <- DESeqDataSetFromMatrix(
  countData = counts,
  colData = metadata,
  design = ~ condition
)

#  Run differential expression analysis
dds <- DESeq(dds)

#  Extract results and clean
res_df <- as.data.frame(results(dds)) |>
  rownames_to_column("gene") |>
  drop_na(log2FoldChange, padj) |>
  arrange(padj)

#  Save for downstream steps
if (!dir.exists("data")) dir.create("data", recursive = TRUE)
write_csv(res_df, "data/deseq2_results.csv")
```

 **Takeaway:** DESeq2 provides a robust statistical framework to identify significantly regulated genes. Once saved, these results can be used for downstream visualizations like volcano plots and MA plots.

# Q&A: 5

# How do you create a volcano plot from DESeq2 results using R?

## 5.1 Explanation

A **volcano plot** combines statistical significance with effect size to highlight genes of interest:

- **X-axis**: log2 fold change (magnitude of differential expression)
- **Y-axis**: -log10 adjusted p-value (statistical significance)

It's ideal for identifying genes that are both **strongly regulated** and **highly significant**. DESeq2 results already contain `log2FoldChange` and `padj` columns, making this visualization straightforward.

## 5.2 R Code

```
library(tidyverse)

#  Load DESeq2 results
res_df <- read_csv("data/deseq2_results.csv") |>
  drop_na(log2FoldChange, padj) |>
```

```r
  mutate(significant = padj < 0.05 & abs(log2FoldChange) > 1)

#   Volcano plot
ggplot(res_df, aes(x = log2FoldChange, y = -log10(padj), color = significant)) +
  geom_point(alpha = 0.6, size = 2) +
  scale_color_manual(values = c("FALSE" = "gray", "TRUE" = "red")) +
  geom_vline(xintercept = c(-1, 1), linetype = "dashed") +
  geom_hline(yintercept = -log10(0.05), linetype = "dashed") +
  labs(title = "Volcano Plot of Differential Expression",
       x = "Log2 Fold Change", y = "-Log10 Adjusted P-Value") +
  theme_minimal()
```


 **Takeaway:** Volcano plots help you visually prioritize genes for downstream validation by showing both statistical significance and magnitude of change.

# Q&A: 6

# How do you create an MA plot from DESeq2 results using R?

## 6.1 Explanation

An **MA plot** shows the relationship between:

- **M (log ratio)** = log2 fold change (Y-axis)
- **A (mean average)** = average expression (X-axis), often `baseMean`

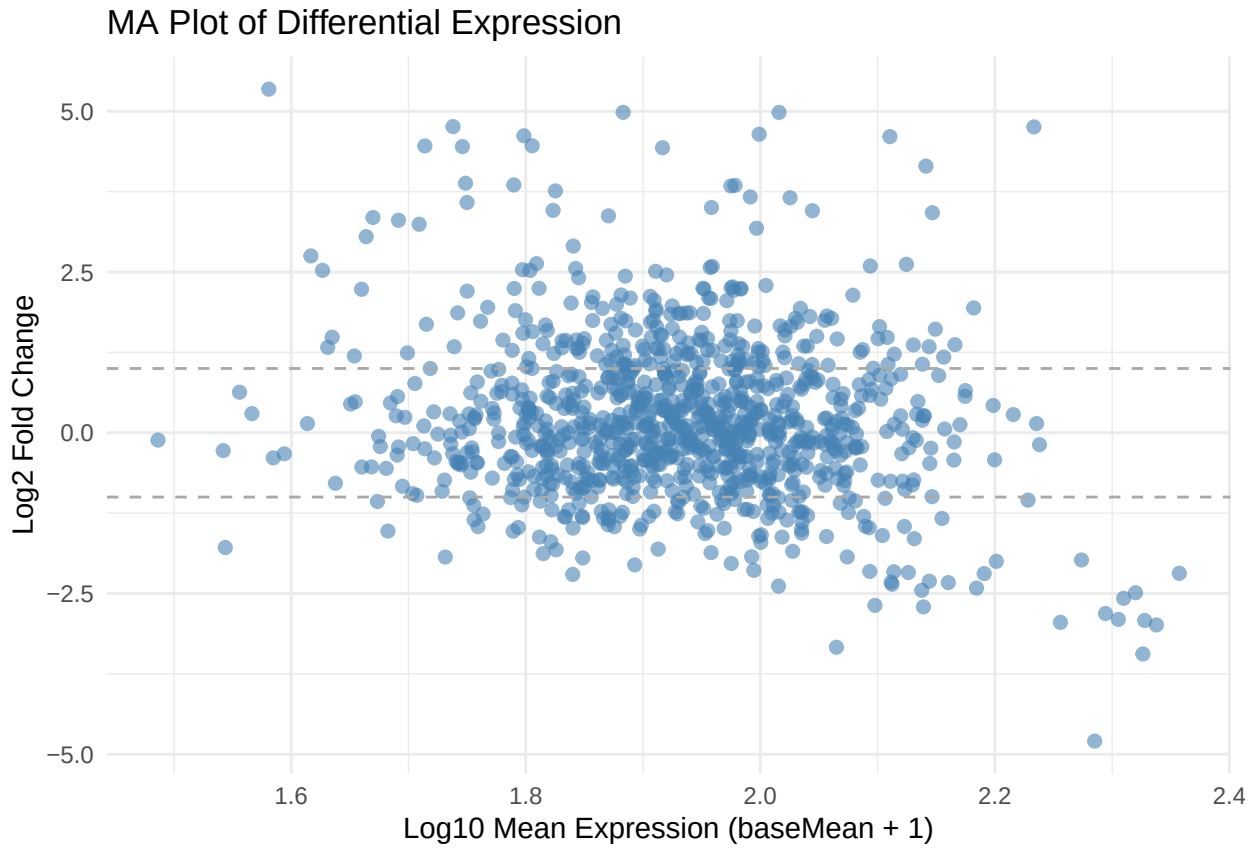
It helps you visualize:

- Genes with **large fold changes**
- Genes with **low expression and unstable variance**
- Potential **systematic bias** in your differential expression results

MA plots are especially useful after running DESeq2, as the result object includes `baseMean` and `log2FoldChange`.

## 6.2 R Code

```r
library(tidyverse)

#  Load DESeq2 results
res_df <- read_csv("data/deseq2_results.csv") |>
  drop_na(log2FoldChange, padj, baseMean)

#  MA plot
ggplot(res_df, aes(x = log10(baseMean + 1), y = log2FoldChange)) +
  geom_point(alpha = 0.6, color = "steelblue", size = 2) +
  geom_hline(yintercept = c(-1, 1), linetype = "dashed", color = "darkgray") +
  labs(title = "MA Plot of Differential Expression",
       x = "Log10 Mean Expression (baseMean + 1)",
       y = "Log2 Fold Change") +
  theme_minimal()
```




 **Takeaway:** MA plots offer a quick summary of how expression changes relate to

average gene abundance, helping detect outliers and trends in your DE analysis.

# Q&A: 7

# How do you log-transform RNA-Seq counts for PCA or clustering using R?

## 7.1 Explanation

Raw RNA-Seq counts are:

- Not normally distributed
- Heteroscedastic (variance increases with mean)
- Influenced by a few highly expressed genes

These properties make them unsuitable for **PCA**, **clustering**, or **heatmaps** without transformation.

To correct this, we apply a **log transformation** to stabilize variance:

- `log2(count + 1)` — simple and fast
- `rlog()` — regularized log transformation (DESeq2), ideal for small sample sizes
- `vst()` — variance-stabilizing transformation, faster for large datasets

We save the rlog matrix so it can be reused by downstream visualizations.

## 7.2 R Code

```r
library(tidyverse)
library(DESeq2)

#  Load count data and metadata
counts <- read_csv("data/demo_counts.csv") |>
  column_to_rownames("gene") |>
  as.matrix()

metadata <- read_csv("data/demo_metadata.csv")

#  Create DESeq2 object
dds <- DESeqDataSetFromMatrix(countData = counts,
                              colData = metadata,
                              design = ~ condition)

#  Transform
rlog_dds <- rlog(dds)

#  Extract transformed matrix
rlog_mat <- assay(rlog_dds) |>
  as.data.frame() |>
  rownames_to_column("gene")

#  Save for reuse
write_csv(rlog_mat, "data/rlog_matrix.csv")
```

 **Takeaway:** Log transformations—especially `rlog()`—stabilize variance and prepare RNA-Seq data for PCA, clustering, and heatmaps. Saving the transformed matrix improves reproducibility.

# Q&A: 8

# How do you create a heatmap of top differentially expressed genes using R?

## 8.1 Explanation

A **heatmap** allows you to visualize the expression patterns of the most differentially expressed genes across all samples. It is especially helpful for:

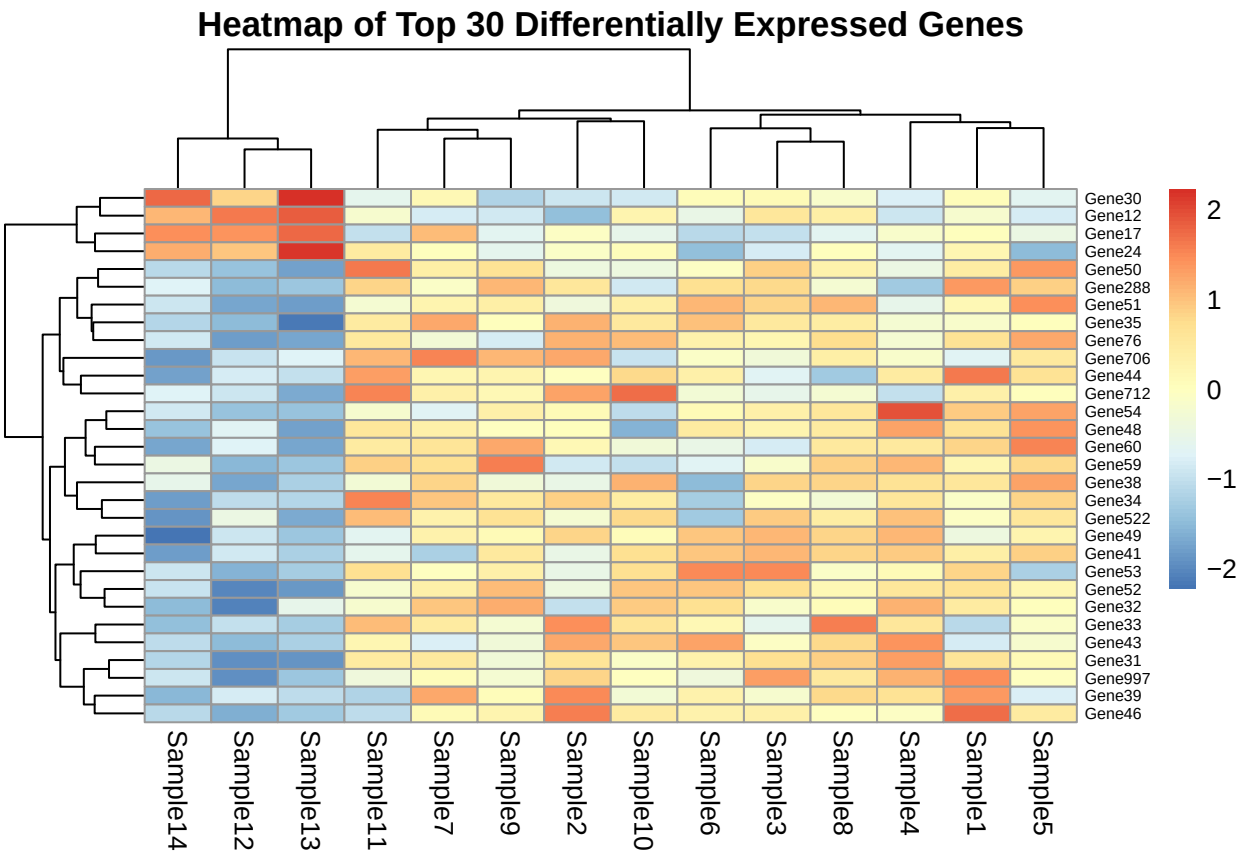
- Revealing sample clustering and gene expression trends
- Highlighting contrasts between conditions
- Identifying outlier samples or expression signatures

We typically use **rlog-transformed data** to ensure that variance is stabilized, making expression patterns more interpretable.

## 8.2 R Code

```
library(tidyverse)
library(pheatmap)

#  Load transformed expression matrix
```

```r
rlog_mat <- read_csv("data/rlog_matrix.csv") |>
  column_to_rownames("gene") |>
  as.matrix()

#  Load DESeq2 results and select top 30 DE genes
res_df <- read_csv("data/deseq2_results.csv") |>
  drop_na(padj) |>
  arrange(padj)

top_genes <- res_df$gene[1:30]
top_mat <- rlog_mat[top_genes, ]

#  Plot heatmap
pheatmap(top_mat,
         cluster_rows = TRUE,
         cluster_cols = TRUE,
         show_rownames = TRUE,
         fontsize_row = 6,
         scale = "row",
         main = "Heatmap of Top 30 Differentially Expressed Genes")
```




 **Takeaway:** Heatmaps are powerful tools to explore gene expression dynamics across conditions. Always use a variance-stabilized matrix and select top DE genes for clarity.

# Q&A: 9

# How do you visualize RNA-Seq samples using PCA in R?

## 9.1 Explanation

**Principal Component Analysis (PCA)** reduces the dimensionality of high-throughput data like RNA-Seq by finding the principal directions of variation. PCA is useful for:

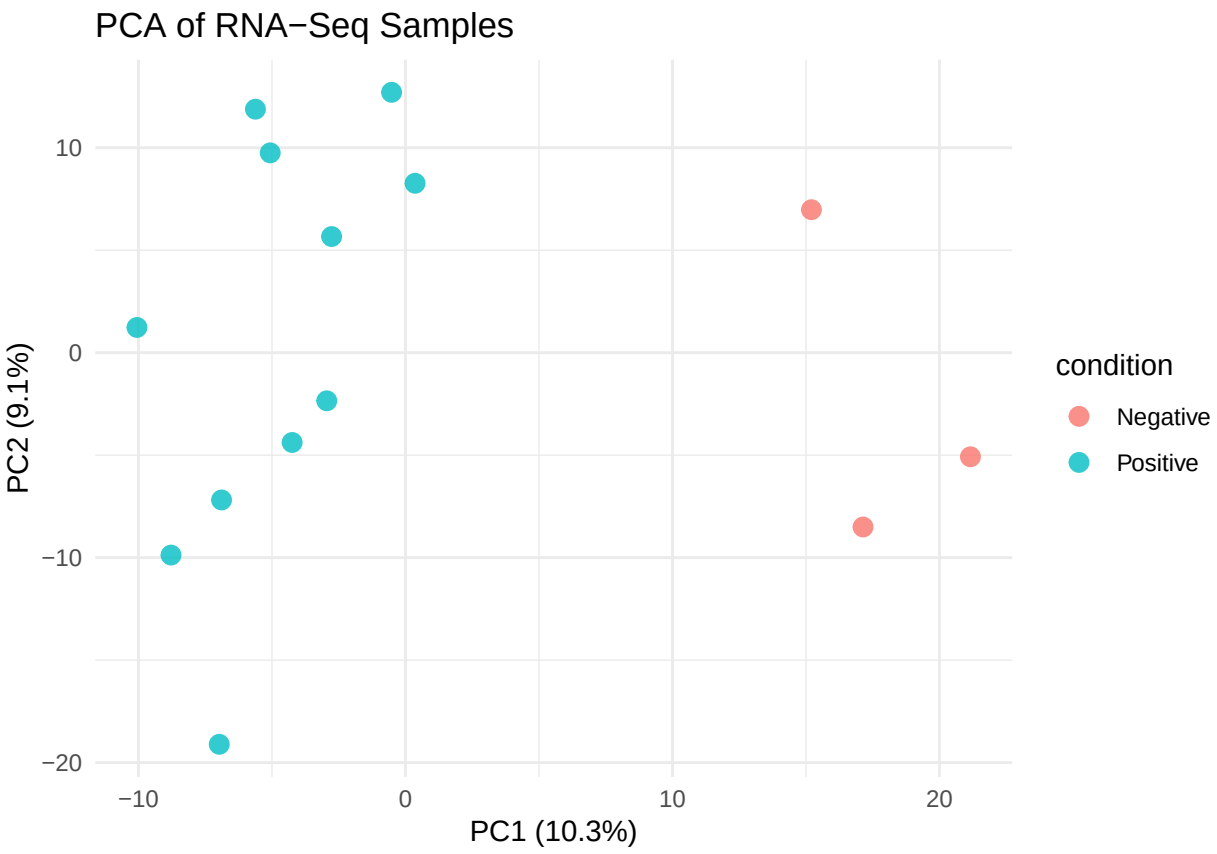
- Detecting **sample outliers**
- Checking for **batch effects**
- Visualizing **group separation**

We apply PCA on the rlog-transformed data (`rlog_matrix.csv`) to ensure homoscedasticity and interpretability.

## 9.2 R Code

```
library(tidyverse)

#  Load rlog-transformed matrix
rlog_mat <- read_csv("data/rlog_matrix.csv")
```

```r
#  Prepare PCA input
pca_input <- rlog_mat |>
  column_to_rownames("gene") |>
  t() |>
  as.data.frame()

#  Run PCA
pca_res <- prcomp(pca_input, center = TRUE, scale. = TRUE)
pca_df <- as_tibble(pca_res$x) |>
  mutate(Sample = rownames(pca_input))

#  Join with metadata
metadata <- read_csv("data/demo_metadata.csv")
plot_df <- left_join(pca_df, metadata, by = "Sample")

#  Plot
ggplot(plot_df, aes(x = PC1, y = PC2, color = condition)) +
  geom_point(size = 3, alpha = 0.8) +
  labs(title = "PCA of RNA-Seq Samples",
       x = paste0("PC1 (", round(summary(pca_res)$importance[2, 1] * 100, 1), "%)"),
       y = paste0("PC2 (", round(summary(pca_res)$importance[2, 2] * 100, 1), "%)")) +
  theme_minimal()
```




 **Takeaway:** PCA on log-transformed RNA-Seq data helps visualize sample similarities, spot outliers, and confirm that experimental conditions drive the major sources of variation.

# Q&A: 10

# How do you visualize the expression of a single gene across conditions using R?

## 10.1 Explanation

To explore how a specific gene behaves across experimental conditions, a **boxplot** of rlog-transformed expression values can provide insight into group differences and variability.

This is useful for:

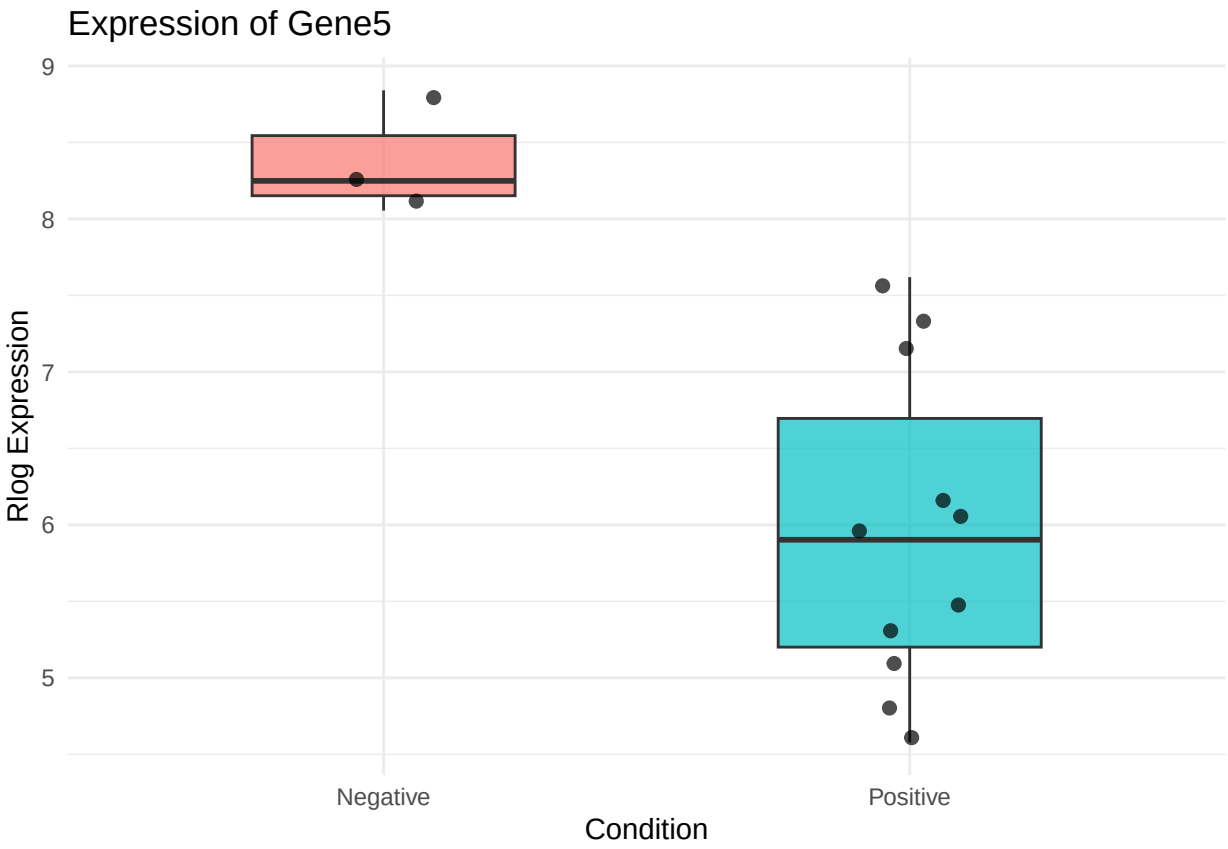
- Validating top hits from DE results
- Highlighting genes of interest
- Creating publication-ready visual summaries

We'll visualize **Gene5** as an example.

## 10.2 R Code

```
library(tidyverse)
```

```r
#  Load rlog-transformed expression matrix
#  Load rlog-transformed expression matrix
rlog_mat <- read_csv("data/rlog_matrix.csv") |>
  column_to_rownames("gene") |>
  as.matrix()

#  Load metadata
metadata <- read_csv("data/demo_metadata.csv")

#  Select one gene
gene_to_plot <- "Gene5"

#  Check gene exists
if (!gene_to_plot %in% rownames(rlog_mat)) stop("Gene not found in rlog matrix.")

#  Create dataframe for plotting
plot_df <- tibble(
  Expression = rlog_mat[gene_to_plot, ],
  Sample = colnames(rlog_mat)
) |>
  left_join(metadata, by = "Sample")

#  Boxplot
ggplot(plot_df, aes(x = condition, y = Expression, fill = condition)) +
  geom_boxplot(width = 0.5, alpha = 0.7, outlier.shape = NA) +
  geom_jitter(width = 0.1, size = 2, alpha = 0.7) +
  labs(title = paste("Expression of", gene_to_plot),
       x = "Condition", y = "Rlog Expression") +
  theme_minimal() +
  theme(legend.position = "none")
```




 **Takeaway:** Boxplots of individual genes help confirm biological patterns and support gene selection for follow-up experiments or reporting.

# Q&A: 11

# How do you visualize the expression of two or more genes across conditions using R?

## 11.1 Explanation

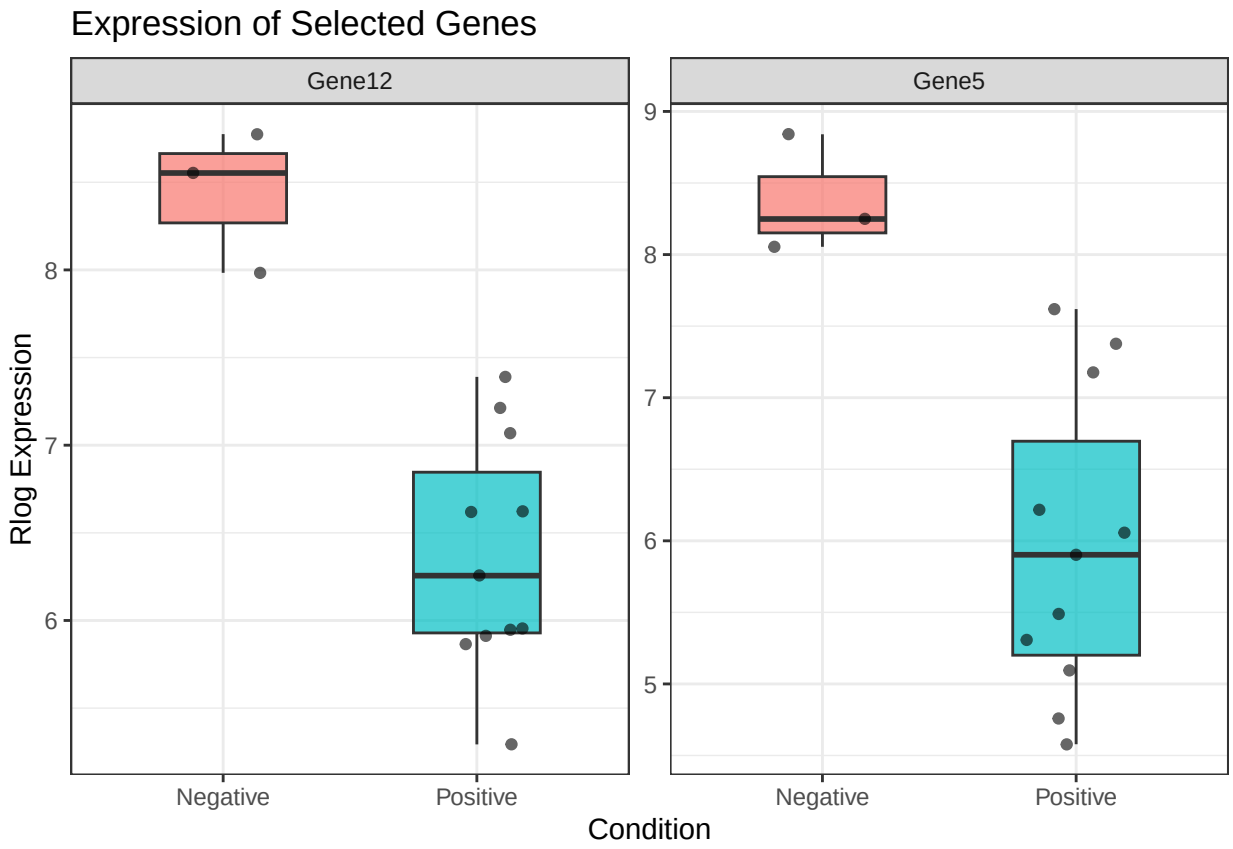
Sometimes, you want to inspect the expression patterns of specific genes of interest — such as those that appear highly upregulated or downregulated in your DESeq2 results.

By selecting two or more genes and reshaping the rlog-transformed matrix into a **tidy long format**, you can easily create grouped boxplots, violin plots, or faceted charts. These plots help validate patterns visually and are useful for presentations or downstream interpretation.

## 11.2 R Code

```
library(tidyverse)

#  Load transformed matrix and metadata
rlog_mat <- read_csv("data/rlog_matrix.csv") |>
  column_to_rownames("gene") |>
  as.matrix()

metadata <- read_csv("data/demo_metadata.csv")
```

```r
#  Select two genes
genes_to_plot <- c("Gene5", "Gene12")

#  Check genes exist
missing_genes <- setdiff(genes_to_plot, rownames(rlog_mat))
if (length(missing_genes) > 0) stop(paste("Missing genes:", paste(missing_genes, collaps

#  Create tidy dataframe for ggplot
plot_df <- rlog_mat[genes_to_plot, ] |>
  as.data.frame() |>
  rownames_to_column("gene") |>
  pivot_longer(-gene, names_to = "Sample", values_to = "Expression") |>
  left_join(metadata, by = "Sample")

#  Boxplot with multiple genes
ggplot(plot_df, aes(x = condition, y = Expression, fill = condition)) +
  geom_boxplot(width = 0.5, alpha = 0.7, outlier.shape = NA) +
  geom_jitter(width = 0.2, size = 1.5, alpha = 0.6) +
  facet_wrap(~gene, scales = "free_y") +
  labs(title = "Expression of Selected Genes",
       x = "Condition", y = "Rlog Expression") +
  theme_bw() +
  theme(legend.position = "none")
```




 **Takeaway:** Visualizing individual gene expression helps verify and communicate biological differences.

 This approach uses `facet_wrap()` to display each gene's distribution across conditions in its own panel, making it easy to compare across genes and conditions.