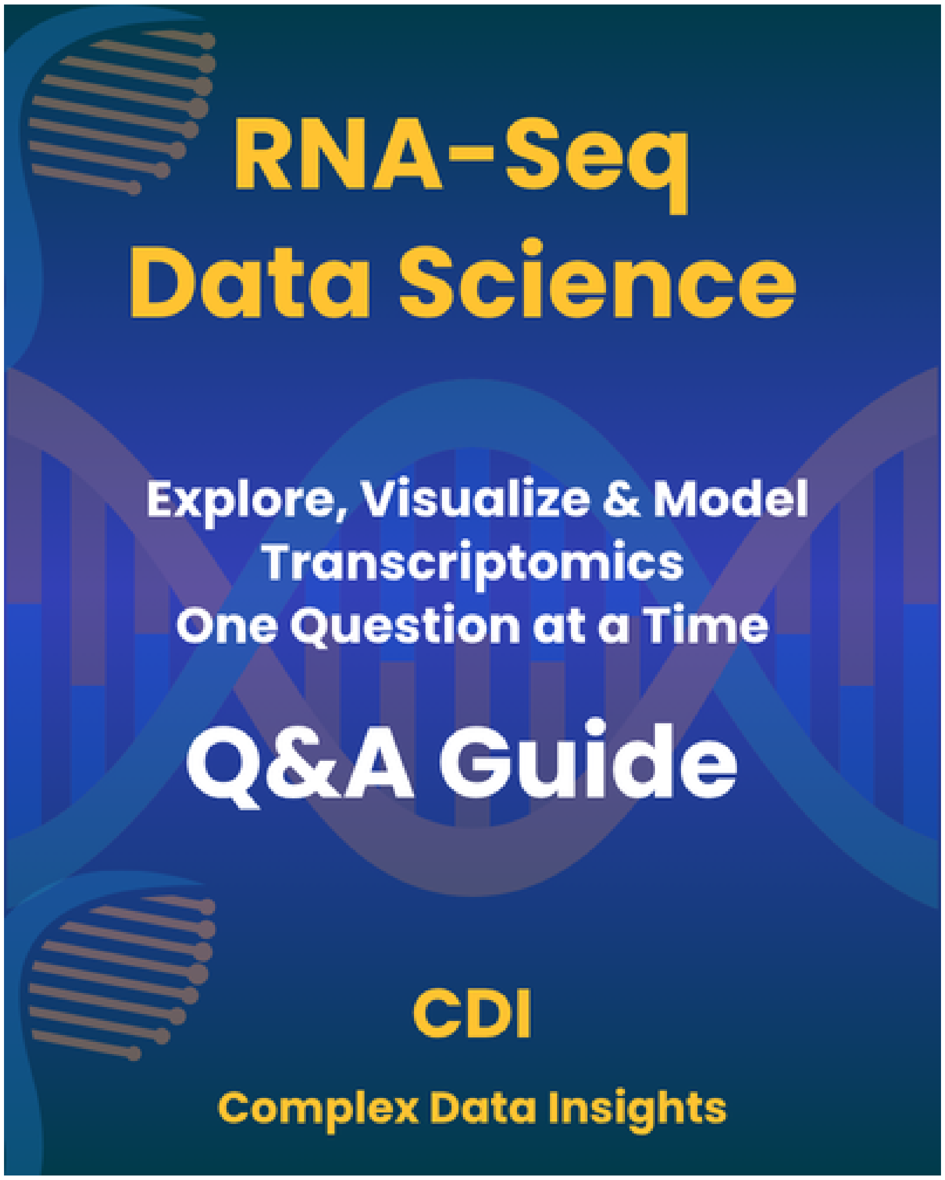
RNA-Seq Data Science



Last updated: June 27, 2025

Contents

4

 Welcome to the RNA-Seq Q&A Guide 5

Getting Started: Preparing RNA-Seq Data for Analysis 6

Prerequisite 6

Step 1: Run DESeq2 in R 6

Output Files 7

Quick Glimpse: How the Demo Data Was Generated 7

I COUNT EXPLORATION 9

1 How do you explore total read counts per sample before normalization? 10

1.1 Explanation 10

1.2 Python Code 10

1.3 R Code 12

2 How do you log-transform RNA-Seq counts for PCA or clustering? 14

2.1 Explanation 14

2.2 Python Code 15

2.3 R Code 16

3 How do you perform PCA to explore sample similarity? 18

3.1 Explanation . . . 18

3.2 Python Code . . . 18

3.3 R Code . . . 20

II DIFFERENTIAL GENE EXPRESSION 23

4 How do you perform differential gene expression analysis using DESeq2? 24

4.1 Explanation . . . 24

4.2 Python Code . . . 25

4.3 R Code . . . 25

5 How do you visualize differentially expressed genes with a volcano plot? 27

5.1 Explanation . . . 27

5.2 Python Code . . . 27

5.3 R Code . . . 29

6 How do you visualize the expression of a single gene with a boxplot? 31

6.1 Explanation . . . 31

6.2 Python Code . . . 31

6.3 R Code . . . 33

7 How do you summarize expression changes with an MA plot? 36

7.1 Explanation . . . 36

7.2 Python Code . . . 36

7.3 R Code . . . 38

□ 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.

Getting Started: Preparing RNA-Seq Data for Analysis

Prerequisite

Before using this CDI Q&A guide, you need input files that are typically generated from an upstream RNA-Seq pipeline (e.g., using **STAR**, **featureCounts**, or **HTSeq-count**):

- `counts.csv` — the raw gene count matrix (genes × samples)

- `metadata.csv` — sample information, including experimental condition or batch

  □ **Note:** Each user may have their own preferred tools and pipelines to generate these files. That part is considered background work and is **not covered** in this guide.
  Once you have these two input files, you're ready to enter the **CDI Q&A learning ecosystem**.

Step 1: Run DESeq2 in R

To generate the analysis-ready files for use in this Q&A guide, run the provided R script using the command line:

```
Rscript scripts/res-df.R
```

□ This script will:

- Load `counts.csv` and `metadata.csv`

- Run DESeq2 for differential expression analysis
- Save cleaned outputs to the `data/` folder

Output Files

After running the script, your `data/` folder will contain:

- □ `demo_counts.csv` — the (optionally cleaned) count matrix
- □ `demo_metadata.csv` — the sample metadata
- □ `deseq2_results.csv` — DESeq2 differential expression results

Quick Glimpse: How the Demo Data Was Generated

The RNA-Seq dataset used in this guide was generated using a standard, reproducible pipeline. This overview is provided for transparency—you do **not** need to reproduce these steps to follow the guide.

- □ **Data type**: Paired-end RNA-Seq
- □ **Preprocessing**: Adapter trimming performed with **Cutadapt**
- □ **Alignment**: Reads were mapped to the *Saccharomyces cerevisiae* genome using **STAR**
- □ **Pipeline orchestration**: Implemented using **Snakemake** from the Workflow Catalog
- □ **Quantification**: Gene counts were generated by STAR and compiled into a TSV table
- □ **Differential Expression**: Conducted with **DESeq2** using the official Bioconductor guidelines

- □ **Parallelization**: The DESeq2 step was assigned **3 CPU cores**

□ **You are now ready to begin the CDI Q&A journey**, exploring and visualizing RNA-Seq results using **Python, R, and Bash**.

The next sections will walk you through each question—step by step—with explanations, code, and visuals.

Part I

COUNT EXPLORATION

Q&A: 1

How do you explore total read counts per sample before normalization?

1.1 Explanation

Before performing any normalization or downstream analysis, it's important to check the total read counts (library sizes) across all samples. This helps identify:

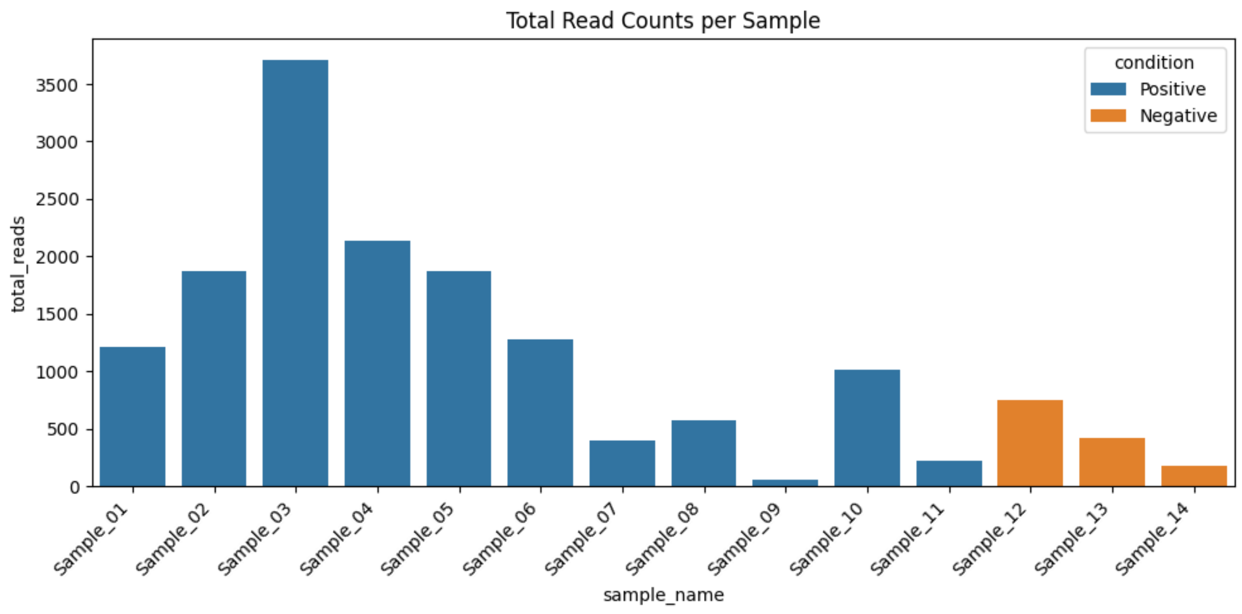
- □ Potential outliers or failed libraries
- □ Low-coverage samples that might skew differential expression
- □ Overall distribution differences across conditions

By visualizing total counts per sample, we can assess whether further filtering or batch correction might be necessary.

1.2 Python Code

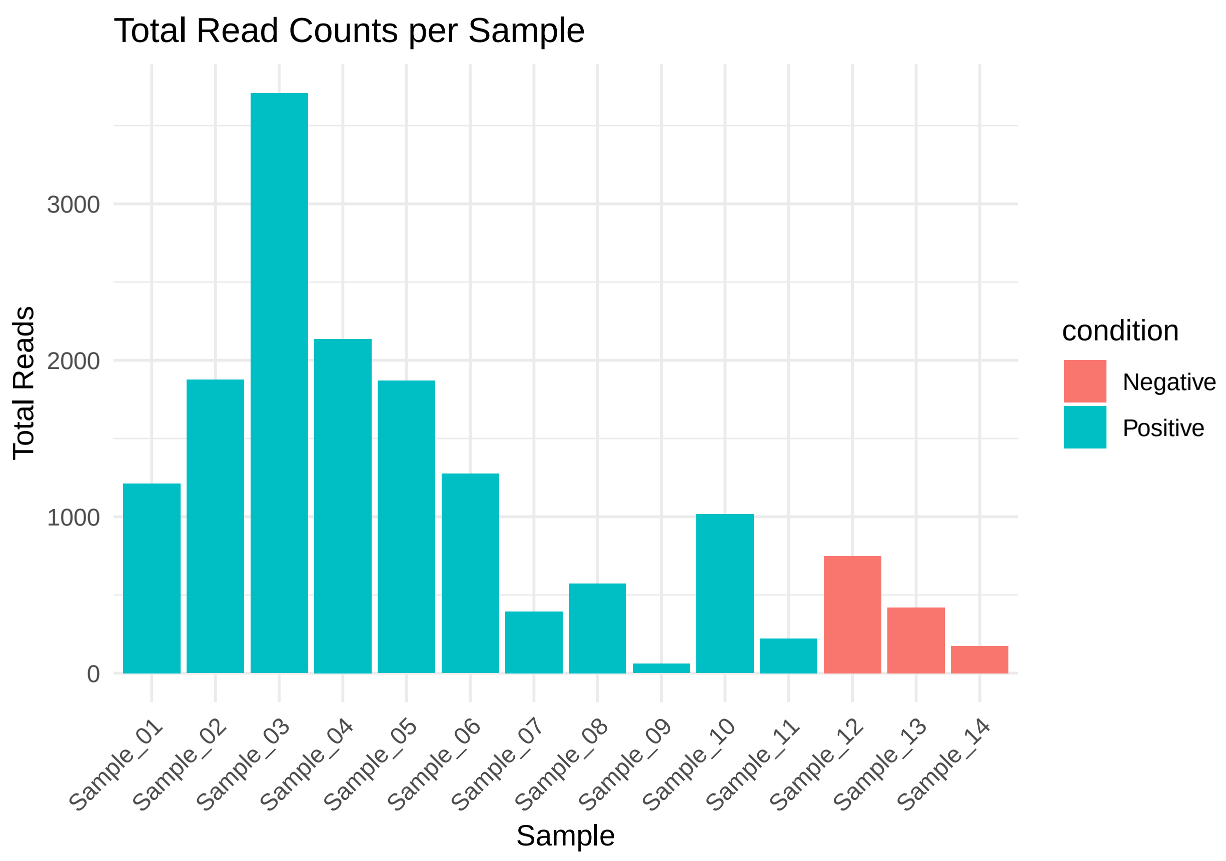
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#  Load data
```

```
counts = pd.read_csv("data/demo_counts.csv", index_col=0)
metadata = pd.read_csv("data/demo_metadata.csv")

#  Compute total counts per sample
total_counts = counts.sum(axis=0).reset_index()
total_counts.columns = ["sample_name", "total_reads"]

#  Merge with metadata
plot_data = pd.merge(total_counts, metadata, on="sample_name")

#  Barplot
plt.figure(figsize=(10, 5))
sns.barplot(data=plot_data, x="sample_name", y="total_reads", hue="condition")
plt.xticks(rotation=45, ha="right")
plt.title("Total Read Counts per Sample")
plt.tight_layout()
plt.show()
```



1.3 R Code

```r
library(tidyverse)

#  Load data
counts <- read_csv("data/demo_counts.csv") %>%
  column_to_rownames("gene")
metadata <- read_csv("data/demo_metadata.csv")

#  Compute total counts per sample
total_reads <- colSums(counts) %>%
  enframe(name = "sample_name", value = "total_reads")

#  Merge with metadata
plot_data <- left_join(total_reads, metadata, by = "sample_name")

#  Barplot
ggplot(plot_data, aes(x = sample_name, y = total_reads, fill = condition)) +
  geom_col() +
  labs(title = "Total Read Counts per Sample", y = "Total Reads", x = "Sample") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



□ **Takeaway:** Exploring total read counts helps detect inconsistencies early. Investigate samples with unusually low or high counts before trusting downstream results.

Q&A: 2

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

2.1 Explanation

Raw RNA-Seq counts are highly skewed and **heteroscedastic**—meaning the variance increases with expression. This makes them unsuitable for clustering, PCA, or heatmap generation because:

- Counts are **not normally distributed**
- **High-expression genes dominate** signal
- **Variance is not stabilized**, which distorts distance-based methods

To address this, we apply a **log2 transformation**, such as:

- `log2(count + 1)` – quick and interpretable
- `rlog()` or `vst()` – DESeq2 methods that stabilize variance more robustly

These transformations are essential to ensure that visualizations like **PCA plots**, **boxplots**, and **heatmaps** reflect meaningful biological structure rather than artifacts of scale or count depth.

> □ **Reminder:** Transformation is a core step in RNA-Seq EDA—it unlocks the ability to "see" patterns clearly.

2.2 Python Code

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#  Load and log-transform
counts = pd.read_csv("data/demo_counts.csv", index_col=0)
log_counts = np.log2(counts + 1)

#  Preview a few genes

log_counts.iloc[:5, :5]
```

Sample_01

Sample_02

Sample_03

Sample_04

Sample_05

gene

SEC24B-AS1

1.584963

3.169925

2.321928

2.807355

2.321928

A1BG

0.000000

```
1.000000

1.000000

2.584963

0.000000

A1CF

3.807355

5.727920

5.930737

0.000000

5.727920

GGACT

2.807355

0.000000

1.000000

2.584963

2.584963

A2M

10.211888

10.824959

11.830515

11.050529

10.820977
```

2.3 R Code

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

#  Load and log-transform
counts <- read_csv("data/demo_counts.csv") %>%
  column_to_rownames("gene")
metadata <- read_csv("data/demo_metadata.csv")

#  Prepare DESeq2 object
dds <- DESeqDataSetFromMatrix(countData = counts, colData = metadata, design = ~ condit

#  Apply regularized log transformation
rlog_counts <- rlog(dds)

#  Preview
assay(rlog_counts)[1:5, 1:5]
```

```
            Sample_01  Sample_02  Sample_03  Sample_04  Sample_05
SEC24B-AS1  0.7183764  1.3717481  0.4156507 1.02934610  0.8621451
A1BG       -1.2882010 -0.8695495 -1.2162496 0.07920678 -1.4217020
A1CF        4.3879484  4.7552744  4.5169602 4.02417352  4.7566223
GGACT       2.4002760  2.0134330  2.0322561 2.21584055  2.2472710
A2M         9.2848548  9.2848548  9.2848548 9.28485478  9.2848548
```

□ **Takeaway:** Always log-transform count data before applying distance-based methods like PCA or clustering. Use `rlog()` or `vst()` for more reliable results on small datasets.

Q&A: 3

How do you perform PCA to explore sample similarity?

3.1 Explanation

Principal Component Analysis (PCA) is a standard technique to explore global sample differences based on gene expression patterns. In RNA-Seq, PCA is often applied **after log-transformation** (e.g., `log2(count + 1)`, `rlog()`, or `vst()`).

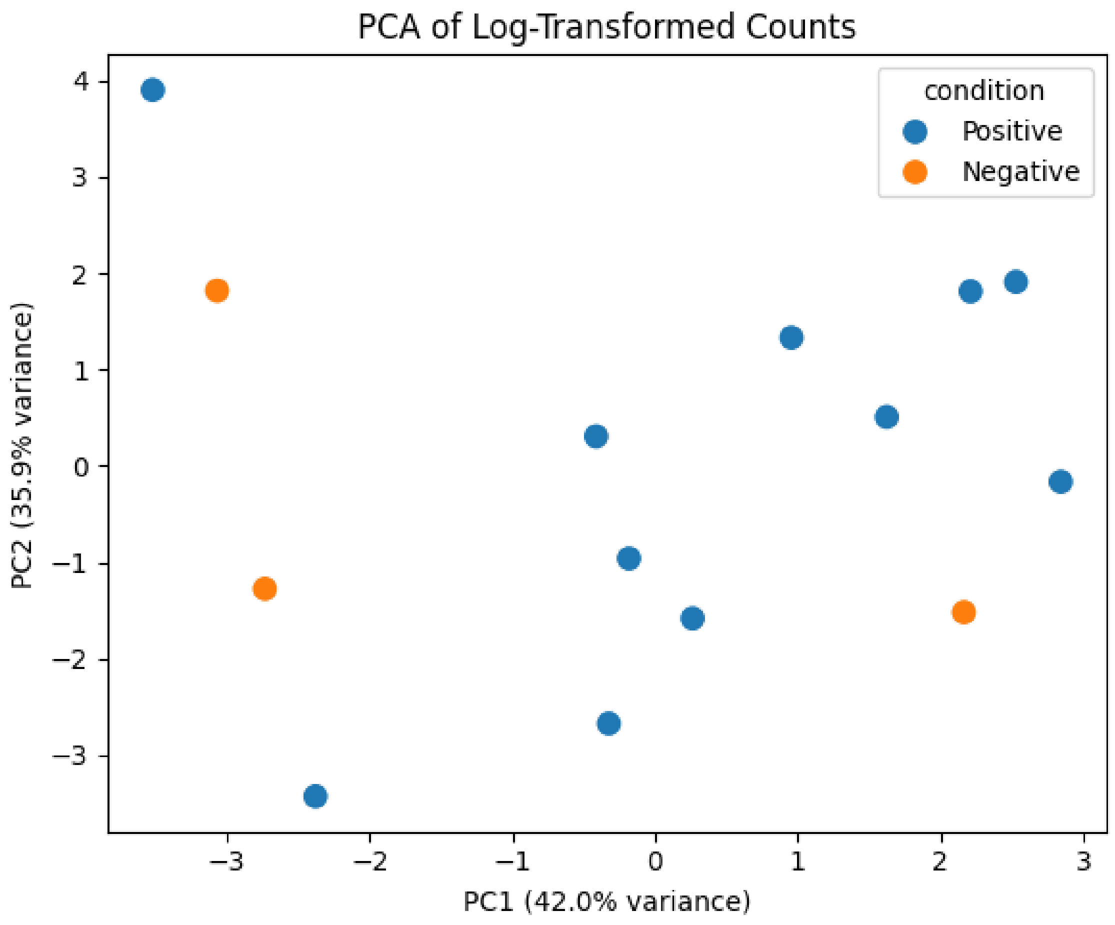
It helps identify:

- □ Clusters of biologically similar samples
- □ Potential outliers or mislabeled conditions
- □ The amount of variance explained by key components

3.2 Python Code

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

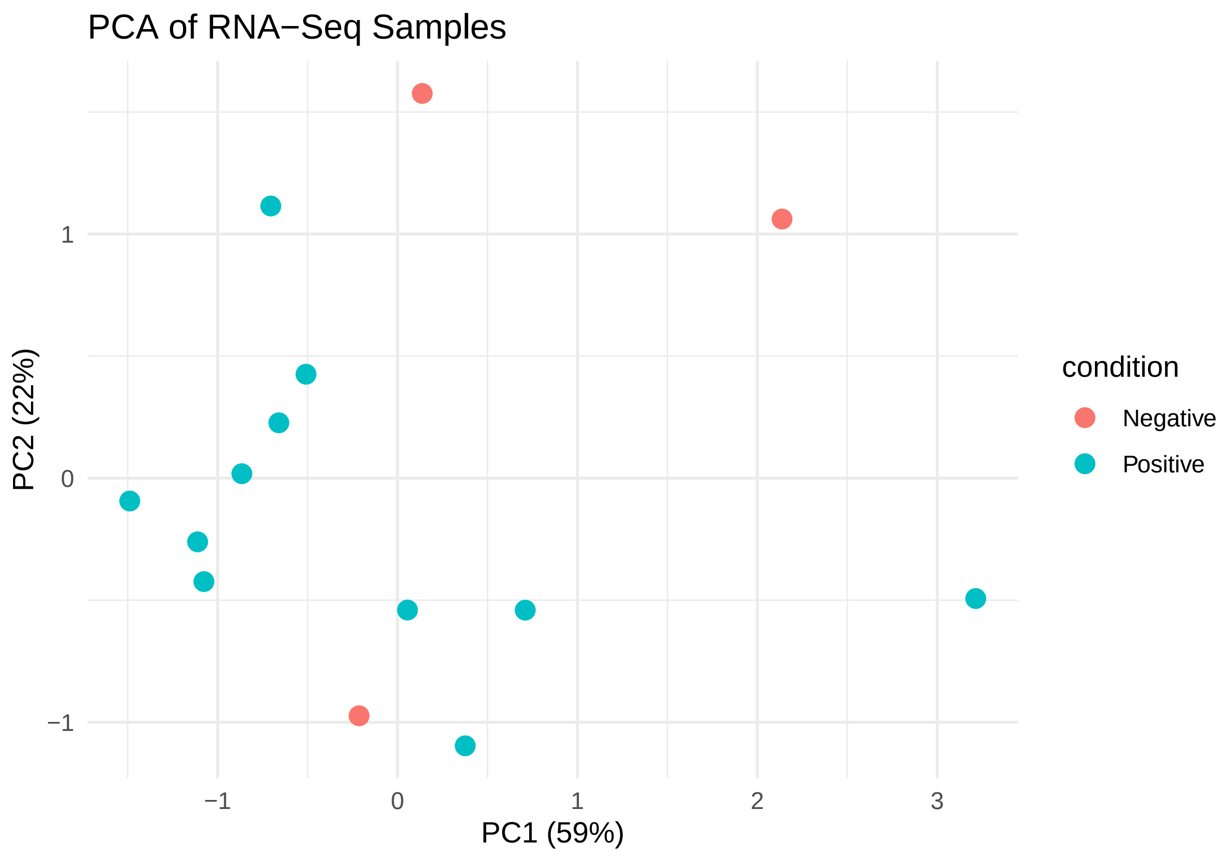
```
#  Set seed for reproducibility
np.random.seed(42)

#  Load data
counts = pd.read_csv("data/demo_counts.csv", index_col=0)
metadata = pd.read_csv("data/demo_metadata.csv")
log_counts = np.log2(counts + 1)

#  PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(log_counts.T)
pca_df = pd.DataFrame(pca_result, columns=["PC1", "PC2"])
pca_df["sample_name"] = log_counts.columns
pca_df = pca_df.merge(metadata, on="sample_name")

#  Plot
plt.figure(figsize=(6, 5))
sns.scatterplot(data=pca_df, x="PC1", y="PC2", hue="condition", s=100)
plt.title("PCA of Log-Transformed Counts")
plt.xlabel(f"PC1 ({pca.explained_variance_ratio_[0]:.1%} variance)")
plt.ylabel(f"PC2 ({pca.explained_variance_ratio_[1]:.1%} variance)")
plt.tight_layout()
plt.show()
```



3.3 R Code

```r
library(tidyverse)
library(DESeq2)
library(ggplot2)

#  Set seed for reproducibility
set.seed(42)

#  Load and transform
counts <- read_csv("data/demo_counts.csv") %>%
  column_to_rownames("gene")
metadata <- read_csv("data/demo_metadata.csv")
```

```
dds <- DESeqDataSetFromMatrix(countData = counts, colData = metadata, design = ~ conditi
rlog_counts <- rlog(dds)

#  PCA plot
pca_data <- plotPCA(rlog_counts, intgroup = "condition", returnData = TRUE)
percentVar <- round(100 * attr(pca_data, "percentVar"))

#  Plot
ggplot(pca_data, aes(PC1, PC2, color = condition)) +
  geom_point(size = 3) +
  labs(title = "PCA of RNA-Seq Samples",
       x = paste0("PC1 (", percentVar[1], "%)"),
       y = paste0("PC2 (", percentVar[2], "%)")) +
  theme_minimal()
```



```r
#  Preview top 5x5 matrix
assay(rlog_counts)[1:5, 1:5]
```

```
            Sample_01  Sample_02  Sample_03  Sample_04  Sample_05
SEC24B-AS1  0.7183764  1.3717481  0.4156507 1.02934610  0.8621451
A1BG       -1.2882010 -0.8695495 -1.2162496 0.07920678 -1.4217020
A1CF        4.3879484  4.7552744  4.5169602 4.02417352  4.7566223
GGACT       2.4002760  2.0134330  2.0322561 2.21584055  2.2472710
A2M         9.2848548  9.2848548  9.2848548 9.28485478  9.2848548
```

□ **Takeaway:** PCA helps you understand how samples cluster before any statistical testing. Always log-transform your data and set a seed for reproducibility.

Part II

DIFFERENTIAL GENE EXPRESSION

Q&A: 4

How do you perform differential gene expression analysis using DESeq2?

4.1 Explanation

Once the count matrix and sample metadata are properly loaded and matched, we use **DESeq2** to identify genes that are significantly differentially expressed between experimental conditions.

DESeq2 performs a robust multi-step process:

1. □ **Estimates size factors** to normalize for library depth
2. □ **Calculates dispersion** for each gene to model biological variability
3. □ **Fits a negative binomial generalized linear model (GLM)** to the counts
4. □ **Tests for significance** using the Wald test (or LRT if specified)
5. □ **Adjusts p-values** using False Discovery Rate (FDR) correction

The result is a table of genes with corresponding **log2 fold changes**, **p-values**, and **adjusted p-values**—ready for interpretation and visualization.

4.2 Python Code

```
# NOTE: DESeq2 is R-based; in Python, use rpy2 or export counts and run DE in R
# Placeholder: Python can handle visualization of DE results after exporting from R
```

4.3 R Code

The code below is store in `scripts/res-df.R`

```
library(tidyverse)
library(DESeq2)

#  Set seed for reproducibility
set.seed(42)

#  Load data
counts <- read_csv("data/demo_counts.csv") %>%
  column_to_rownames("gene")
metadata <- read_csv("data/demo_metadata.csv")

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

#  Run DE analysis
dds <- DESeq(dds)

#  Extract results
res <- results(dds)

#  Clean results
res_df <- as.data.frame(res) %>%
  rownames_to_column("gene") %>%
  arrange(padj)

#  Save for downstream visualization if not already saved
if (!file.exists("data/deseq2_results.csv")) {
  write_csv(res_df, "data/deseq2_results.csv")
}

#  Preview top results
head(res_df, 5)
```

gene	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
A1BG	1.116303	-2.9257945	1.810125	-1.6163498	0.1060187	0.6361123
SEC24B-AS1	5.020260	-0.8370475	1.221274	-0.6853887	0.4930987	0.9861974
A2ML1	2.376188	-2.0927894	2.344610	-0.8925959	0.3720736	0.9861974
A1CF	60.556927	0.1956390	1.364227	0.1434065	0.8859692	0.9999971
GGACT	18.275000	-0.1999325	1.724350	-0.1159465	0.9076949	0.9999971

□ **Takeaway:** DESeq2 provides robust statistical testing for identifying differentially expressed genes. Always inspect and sort results by adjusted p-value (`padj`) to focus on the most significant findings.

Q&A: 5

How do you visualize differentially expressed genes with a volcano plot?

5.1 Explanation

A **volcano plot** combines both statistical significance and effect size in one visual:

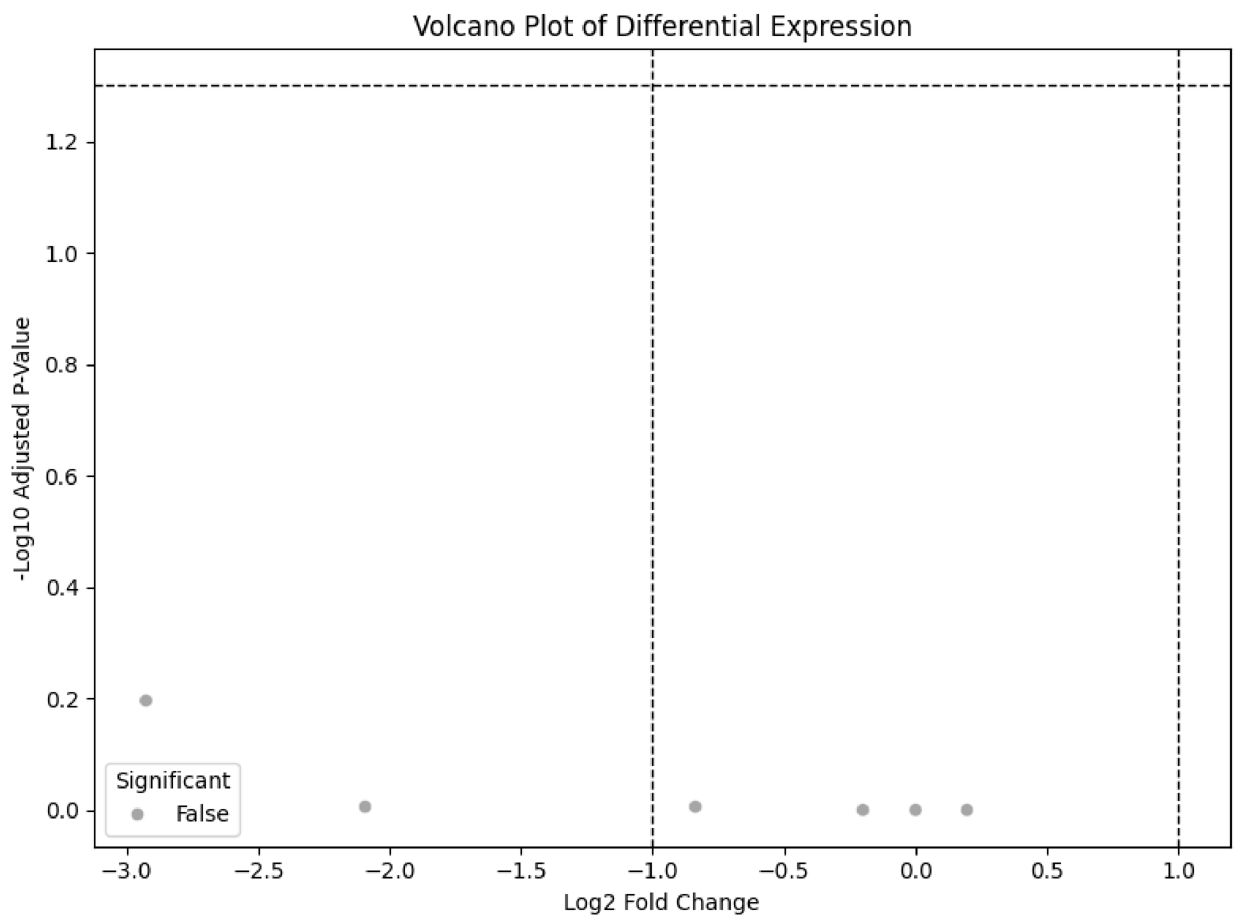
- **X-axis**: log2 fold change (magnitude of expression difference)
- **Y-axis**: -log10 adjusted p-value (significance)
- Helps highlight genes that are both **statistically significant** and **strongly regulated**
- Typically uses color to distinguish significant genes for quick interpretation

5.2 Python Code

> □ Make sure the file `data/deseq2_results.csv` has already been saved by the (above) R code from the DESeq2 step.

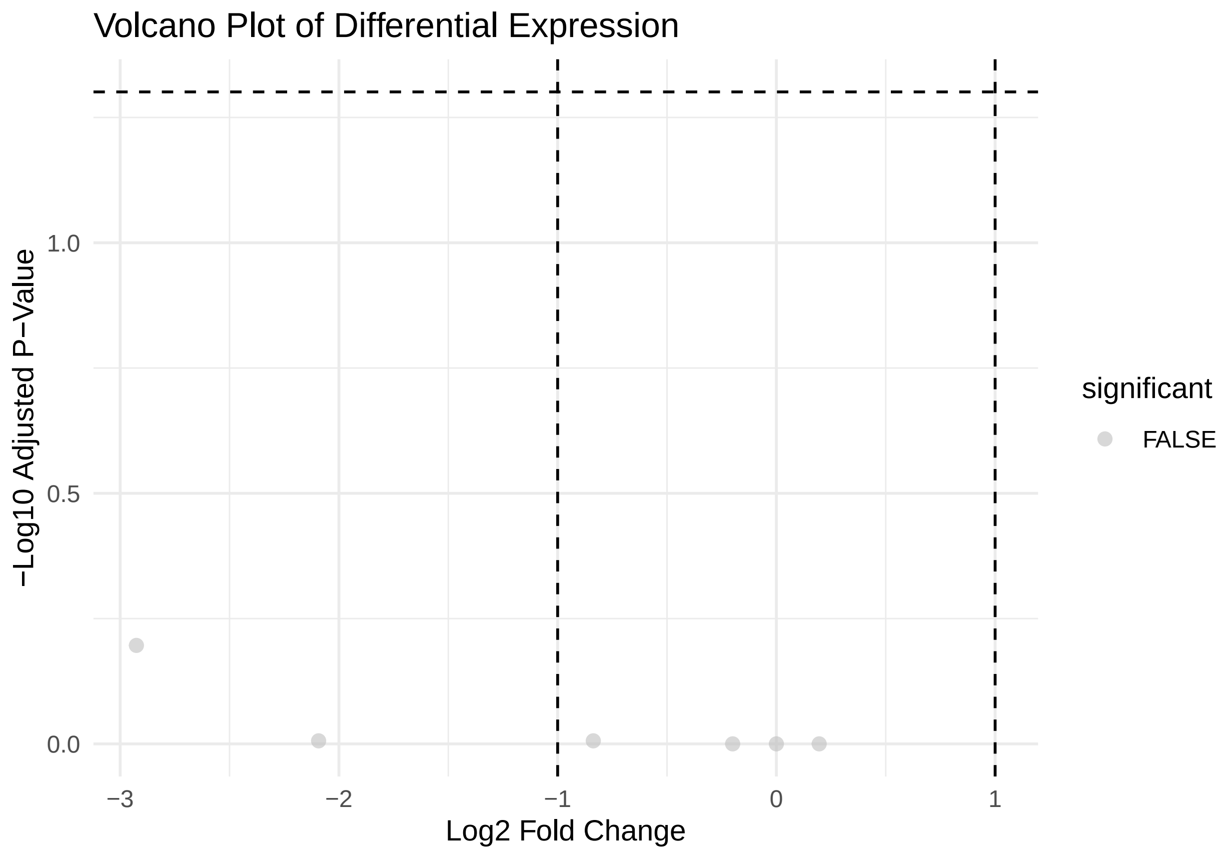
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
#   Load DE results (exported from R)
res_df = pd.read_csv("data/deseq2_results.csv")

#   Add significance status
res_df["significant"] = (res_df["padj"] < 0.05) & (abs(res_df["log2FoldChange"]) > 1)

#   Volcano plot
plt.figure(figsize=(8, 6))
sns.scatterplot(data=res_df,
                x="log2FoldChange",
                y=-np.log10(res_df["padj"]),
                hue="significant",
                palette={True: "red", False: "gray"},
                alpha=0.7)
plt.axhline(-np.log10(0.05), linestyle="--", color="black", linewidth=1)
plt.axvline(x=-1, linestyle="--", color="black", linewidth=1)
plt.axvline(x=1, linestyle="--", color="black", linewidth=1)
plt.title("Volcano Plot of Differential Expression")
plt.xlabel("Log2 Fold Change")
plt.ylabel("-Log10 Adjusted P-Value")
plt.legend(title="Significant")
plt.tight_layout()
plt.show()
```



5.3 R Code

```r
library(tidyverse)

#  Load saved DESeq2 results from CSV
res_df <- read_csv("data/deseq2_results.csv") %>%
  drop_na(log2FoldChange, padj) %>%
  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 visualize genes with both strong effect size and statistical significance, making it easier to prioritize candidates for downstream analysis or validation. Use clear thresholds (e.g., log2FC > 1, padj < 0.05) to highlight key hits

Q&A: 6

How do you visualize the expression of a single gene with a boxplot?

6.1 Explanation

Once differential expression is complete, it's common to visualize individual genes of interest. A **boxplot** shows how expression levels vary across conditions.

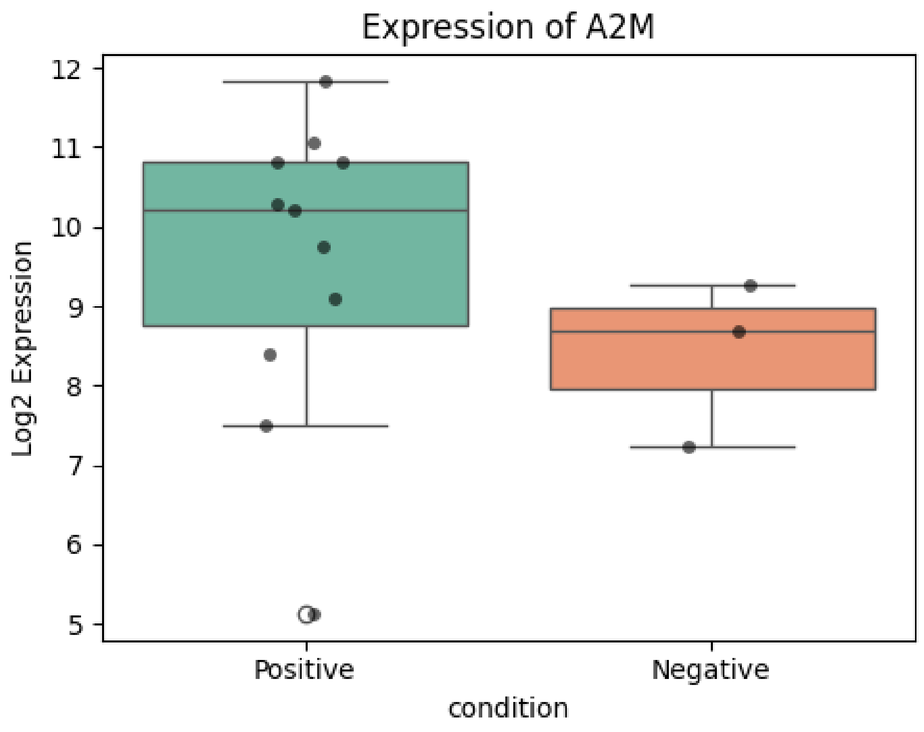
To make this plot: - Use **log-transformed counts** (e.g., from `rlog()` or `log2(count + 1)`) - Select one gene of interest from the DE results - Combine expression values with sample metadata

6.2 Python Code

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#  Load and prepare data
log_counts = np.log2(pd.read_csv("data/demo_counts.csv", index_col=0) + 1)
metadata = pd.read_csv("data/demo_metadata.csv")
gene_to_plot = "A2M"
```

```python
#  Check if gene exists
if gene_to_plot not in log_counts.index:
    raise ValueError(f"{gene_to_plot} not found in count matrix.")

#  Prepare long-form dataframe
df_plot = pd.DataFrame({
    "expression": log_counts.loc[gene_to_plot],
    "sample_name": log_counts.columns
}).merge(metadata, on="sample_name")

#  Boxplot
plt.figure(figsize=(5, 4))
sns.boxplot(data=df_plot, x="condition", y="expression", palette="Set2")
sns.stripplot(data=df_plot, x="condition", y="expression", color="black", alpha=0.6)
plt.title(f"Expression of {gene_to_plot}")
plt.ylabel("Log2 Expression")
plt.tight_layout()
plt.show()
```

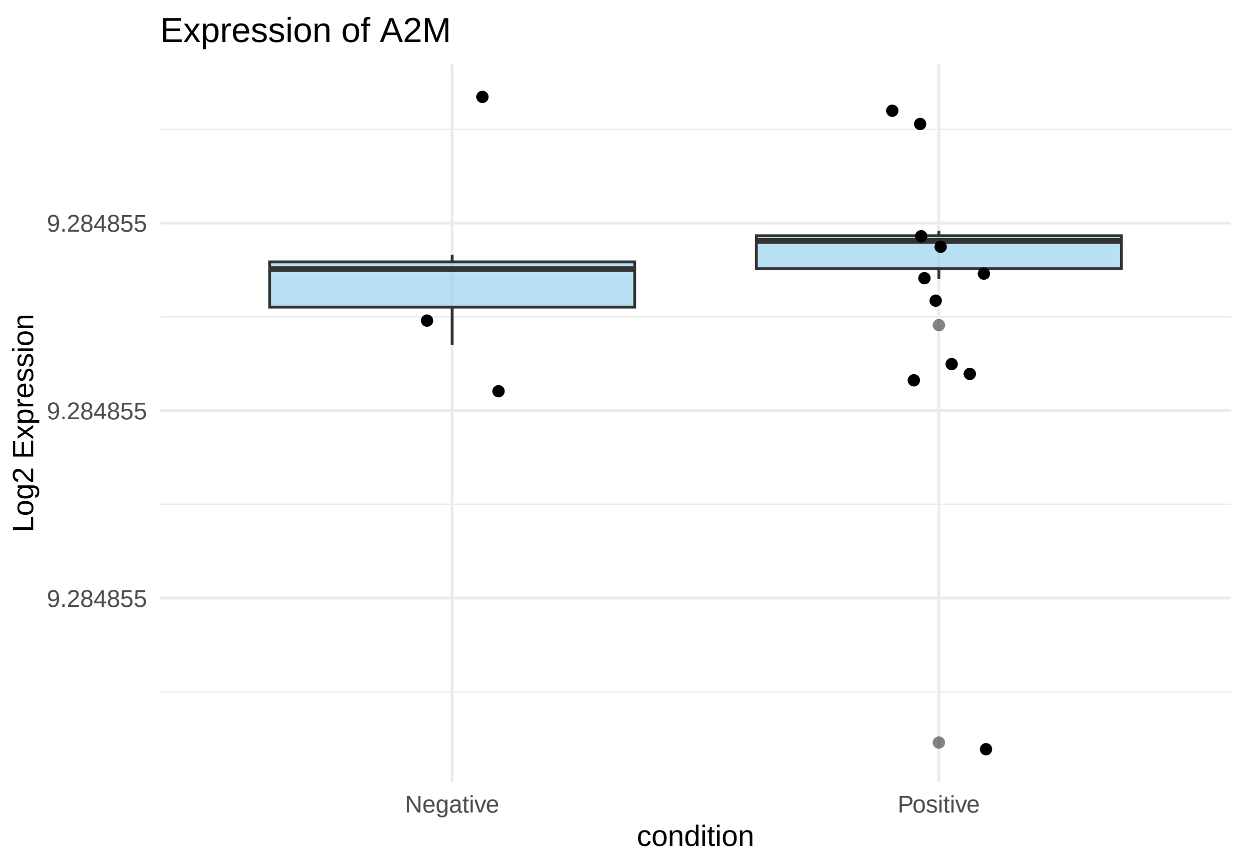
```
/var/folders/m1/0dxpqygn2ds41kxkjgwtftr00000gn/T/ipykernel_41833/1503619118.py:23: Futur

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.boxplot(data=df_plot, x="condition", y="expression", palette="Set2")
```



6.3 R Code

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

#  Set seed for reproducibility
set.seed(42)

#  Load and transform
counts <- read_csv("data/demo_counts.csv") %>%
  column_to_rownames("gene")
metadata <- read_csv("data/demo_metadata.csv")

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

```r
#   Gene to plot
gene_to_plot <- "A2M"
if (!gene_to_plot %in% rownames(rlog_counts)) stop("Gene not found.")

#   Prepare dataframe
plot_data <- data.frame(
  expression = assay(rlog_counts)[gene_to_plot, ],
  sample_name = colnames(rlog_counts)
) %>%
  left_join(metadata, by = "sample_name")

#   Boxplot
ggplot(plot_data, aes(x = condition, y = expression)) +
  geom_boxplot(fill = "skyblue", alpha = 0.6) +
  geom_jitter(width = 0.1) +
  labs(title = paste("Expression of", gene_to_plot),
       y = "Log2 Expression") +
  theme_minimal()
```



□ **Takeaway:** Use boxplots to clearly visualize how expression of a specific gene differs between conditions. Combine with DE results to validate biological interpretation.

Q&A: 7

How do you summarize expression changes with an MA plot?

7.1 Explanation

An **MA plot** (short for **Minus vs. Average plot**) displays the relationship between:

- **M (log ratio)** = log2 fold change (Y-axis), showing the difference in expression between conditions

- **A (mean average)** = average expression across all samples (X-axis), typically on a log scale

This visualization helps identify:

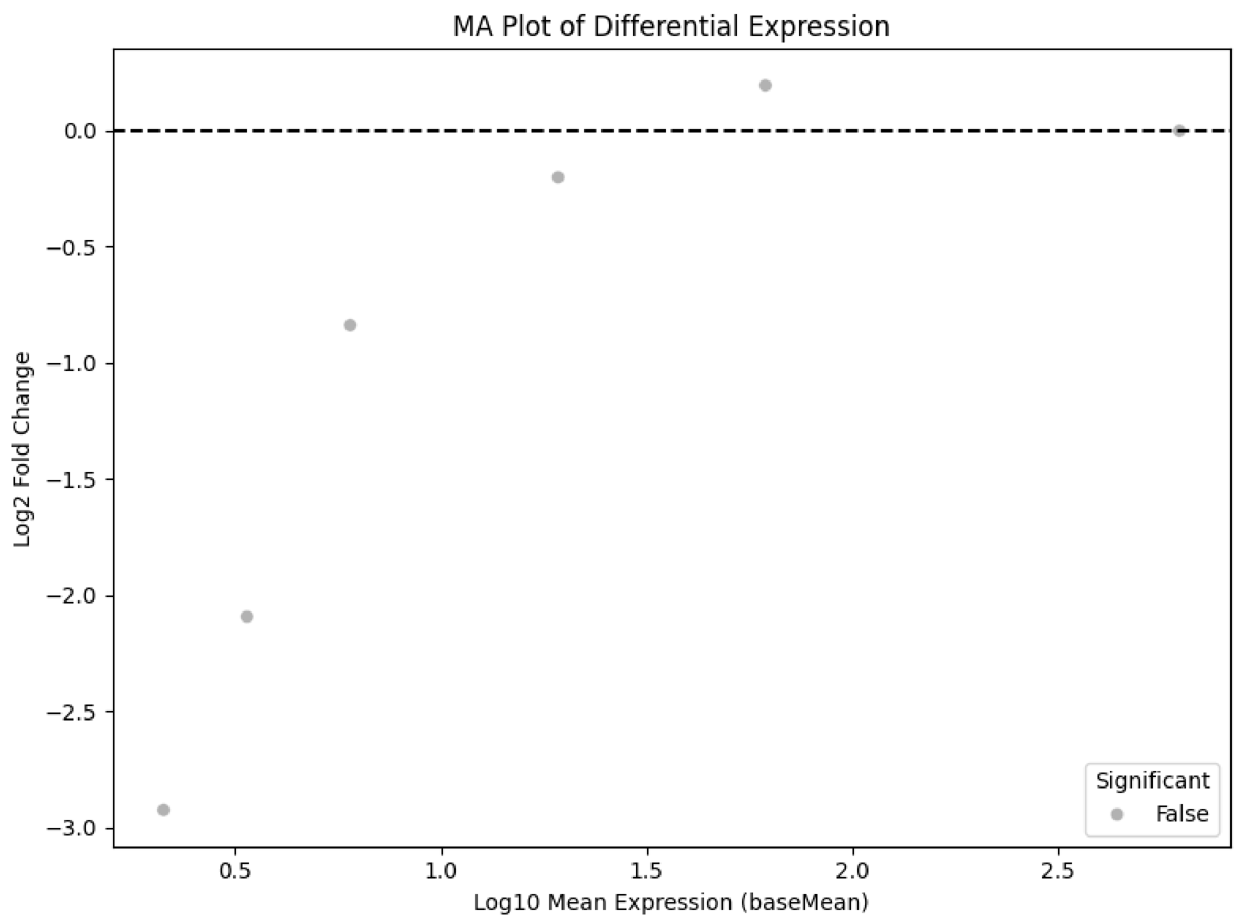
- □ Genes with **large fold changes**
- □ Low-abundance genes with unstable variance
- □ Systematic biases or asymmetries in the DE results

It's especially useful after running DESeq2, as the result object already contains both `baseMean` (A) and `log2FoldChange` (M).

7.2 Python Code

□ Make sure `data/deseq2_results.csv` is available, saved from the R step.

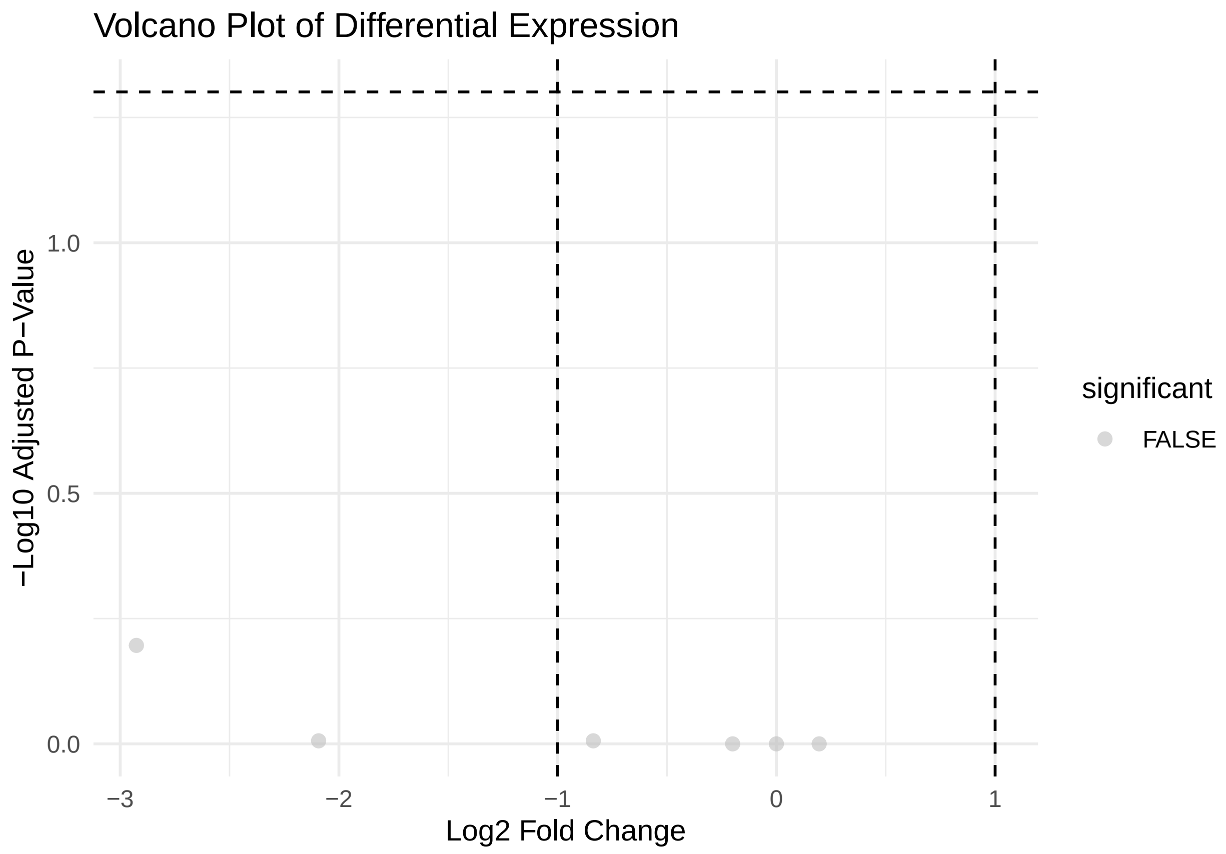
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#  Load DE results
res_df = pd.read_csv("data/deseq2_results.csv")

#  Add significance flag
res_df["significant"] = (res_df["padj"] < 0.05) & (abs(res_df["log2FoldChange"]) > 1)

#  MA plot
plt.figure(figsize=(8, 6))
sns.scatterplot(data=res_df,
                x=np.log10(res_df["baseMean"] + 1),
                y=res_df["log2FoldChange"],
                hue="significant",
                palette={True: "red", False: "gray"},
                alpha=0.6)
plt.axhline(0, linestyle="--", color="black")
plt.title("MA Plot of Differential Expression")
plt.xlabel("Log10 Mean Expression (baseMean)")
plt.ylabel("Log2 Fold Change")
plt.legend(title="Significant")
plt.tight_layout()
plt.show()
```



7.3 R Code

```
library(tidyverse)

#  Load DE results saved during initial processing
res_df <- read_csv("data/deseq2_results.csv") %>%
  drop_na(log2FoldChange, padj) %>%
  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")) +
```

```r
  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:** MA plots reveal expression trends across the full dynamic range of genes. Use them to verify if your DE analysis is symmetric and highlight low-abundance noise.