Script RNA-seq

Ladan Ajdanian

2024-11-05

Contents

RNA-Seq Data Analysis of Cannabis Using R

This document outlines the steps taken to analyze RNA-seq data from Cannabis samples using R. The analysis includes data loading, normalization, differential expression analysis, PCA, heatmap generation, volcano plot creation, correlation matrix plotting, MA plot, gene network construction, and K-means clustering.

1. Load Required Libraries

Before starting the analysis, we need to load the necessary libraries.

Load necessary libraries

```
library(edgeR)      # For differential expression analysis
```

```
## Loading required package: limma
```

```
library(ggplot2)    # For data visualization
library(pheatmap)   # For generating heatmaps
library(igraph)     # For creating gene networks
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

2. Load Data

Next, we load the RNA-seq data into R. The data should be formatted with genes as rows and samples as columns, and the first column should contain gene names.

```
data <- read.csv("~/BVG-Assignment/2_Data_RNASeq_Cannabis_Sex.csv", row.names = 1)
```

3. Define Conditions

We define the experimental conditions, which indicate the two groups being compared: XX and XY. This factor will help in grouping the samples for further analysis. The number 69 represents the number of samples in each experimental group being compared. Specifically, it indicates that there are 69 samples for group XX and 69 samples for group XY.

```
condition <- factor(c(rep("XX", 69), rep("XY", 69))) # Adjust according to your sample labels
group <- data.frame(condition)
```

4. Create DGEList Object for edgeR Analysis

We create a DGEList object that holds the counts data and the grouping information. This object is essential for performing the edgeR analyses.

```
dge <- DGEList(counts = data, group = condition)
```

5. Filter Low-Count Genes

To improve the quality of our analysis, we filter out genes with low counts. This step helps to reduce noise and focus on biologically relevant genes.

```
keep <- filterByExpr(dge)
dge <- dge[keep, , keep.lib.sizes = FALSE]
```

6. Perform TMM Normalization

Next, we apply TMM (Trimmed Mean of M-values) normalization to adjust for compositional differences between libraries, allowing for accurate comparison of gene expression levels.

```
dge <- calcNormFactors(dge, method = "TMM")
normalized_counts <- cpm(dge, normalized.lib.sizes = TRUE)
```

7. Estimate Dispersions and Fit a GLM

We estimate the dispersions of the count data and fit a Generalized Linear Model (GLM) to the data. This model will help us to assess the differences in gene expression between the two conditions.

```
dge <- estimateDisp(dge)
```

```
## Using classic mode.
```

```
fit <- glmFit(dge, design = model.matrix(~condition))
lrt <- glmLRT(fit, coef = 2)  # Test for differences between "XX" and "XY"
```

8. Extract Significant Genes

We extract the genes that are significantly differentially expressed based on a specified p-value threshold. The results are saved to a CSV file for further analysis.

```
significant_genes <- topTags(lrt, n = Inf, p.value = 0.05)$table
```

Save Significant Genes to CSV

```
write.csv(significant_genes, file = "significant_genes.csv", row.names = TRUE)
```

9. PCA Analysis

PCA simplifies complex gene expression data by reducing dimensionality, identifies patterns and group differences, and minimizes noise, providing essential insights for further analysis.

9.1 Generate Normalized Counts for PCA

We prepare the normalized counts data for Principal Component Analysis (PCA), which will help visualize the variability in the data.

```
norm_counts <- cpm(dge, log = TRUE)
```

9.2 Perform PCA

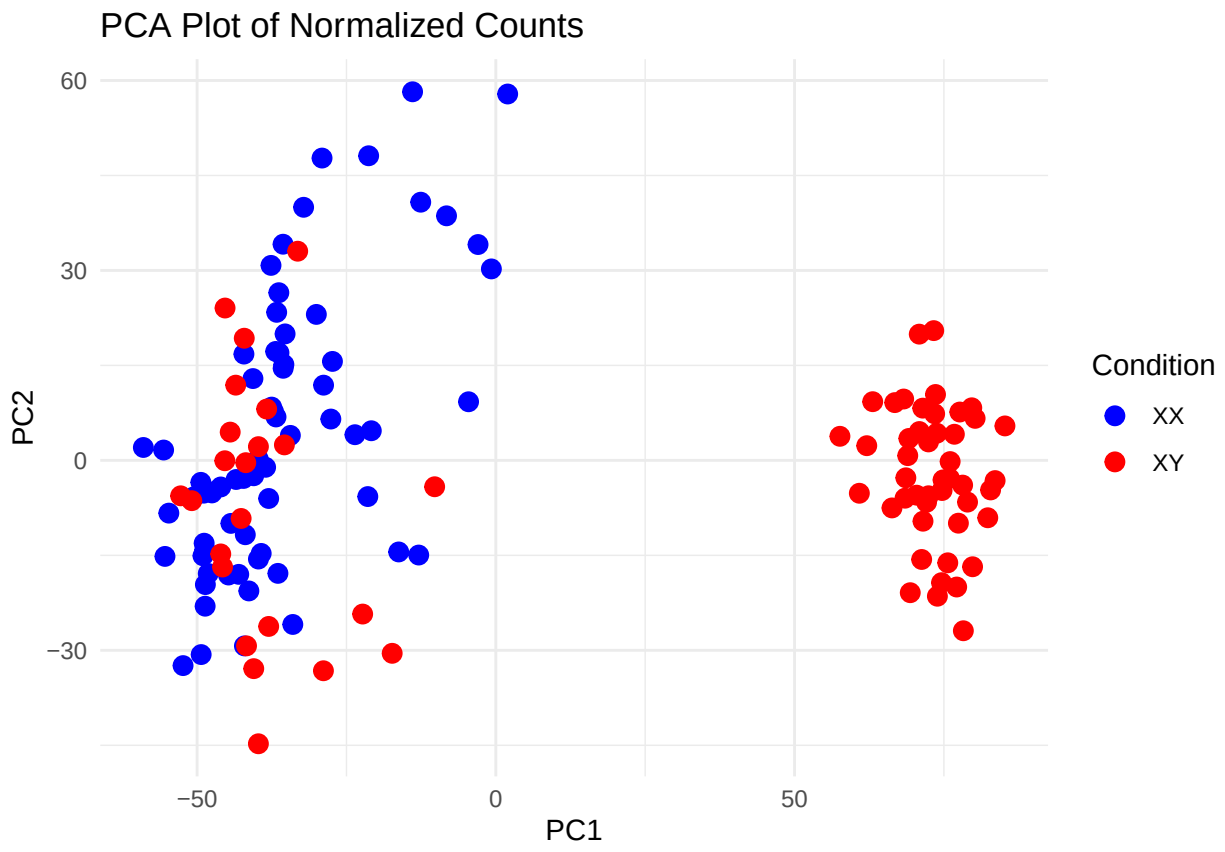
Using the PCA function, we transform the data into principal components.

```
pca <- prcomp(t(norm_counts), scale. = TRUE)
pca_data <- data.frame(PC1 = pca$x[,1], PC2 = pca$x[,2], Condition = condition)
```

9.3 Plot PCA

We visualize the PCA results, coloring the points by condition.

```
ggplot(pca_data, aes(x = PC1, y = PC2, color = Condition)) +
geom_point(size = 3) + labs(title = "PCA Plot of Normalized Counts", x =
"PC1", y = "PC2") + theme_minimal() + scale_color_manual(values =
c("blue", "red"))
```



10. Heatmap of Top Variable Genes

The heatmap of the top variable genes is created to visualize the expression patterns of genes that exhibit the highest variability across samples. This visualization aids in identifying clusters of co-expressed genes and highlighting differences between the experimental conditions (XX and XY), thereby providing insights into the underlying biological processes and potential functional relationships among genes.

10.1 Select Top 500 Most Variable Genes

We identify the top 500 genes with the highest variability across samples to visualize their expression in a heatmap.

```r
var_genes <- apply(norm_counts, 1, var)
top_var_genes <- names(sort(var_genes, decreasing = TRUE))[1:500]
top_var_data <- norm_counts[top_var_genes, ]
```

10.2 Prepare Annotation Data

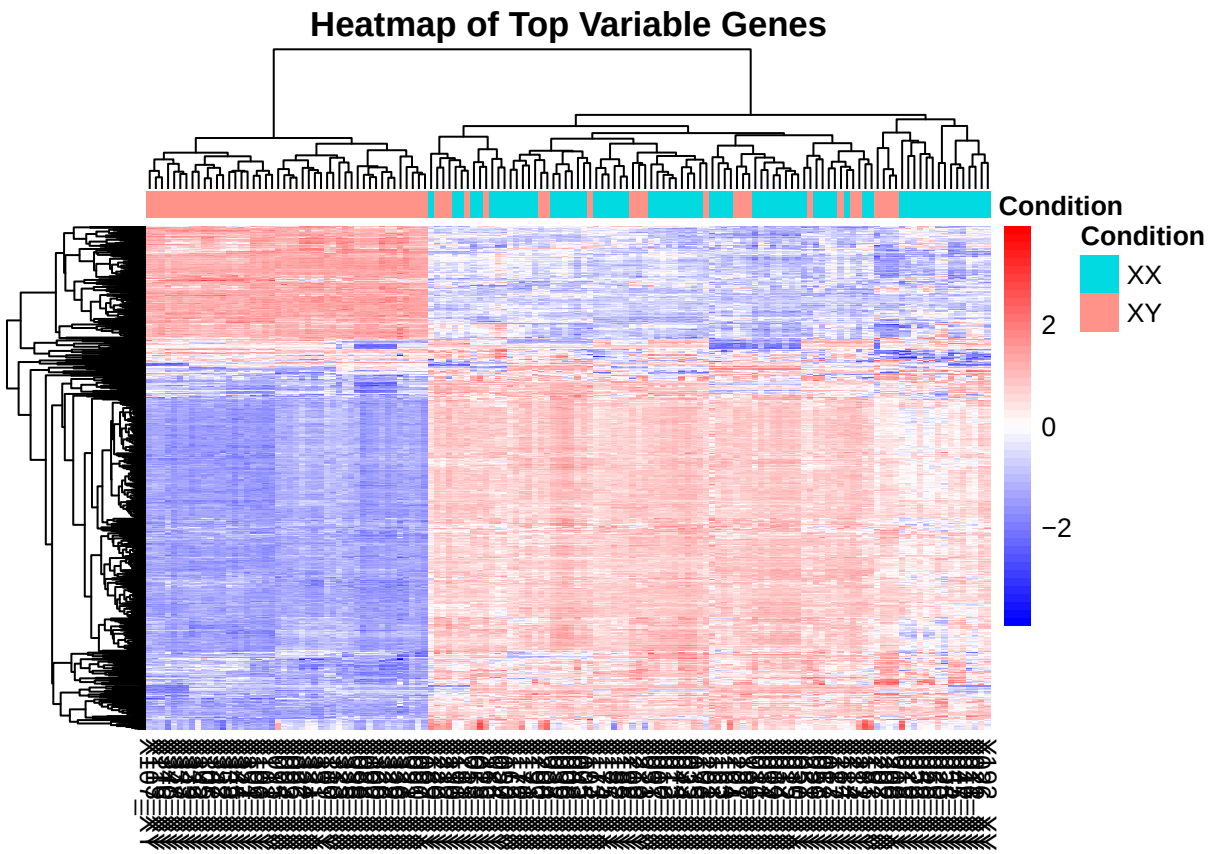
We prepare annotation data for the heatmap.

```r
annotation_data <- data.frame(Condition = condition)
rownames(annotation_data) <- colnames(top_var_data)
```

10.3 Generate Heatmap

We create a heatmap of the top variable genes, scaling the rows to visualize relative expression levels.

```r
if (nrow(top_var_data) > 1 && ncol(top_var_data) > 1) {
  pheatmap(
    top_var_data, scale = "row", show_rownames = FALSE,
    annotation_col = annotation_data,
    main = "Heatmap of Top Variable Genes",
    color = colorRampPalette(c("blue", "white", "red"))(50) # Custom color palette
  )
} else {
  message("Insufficient variable genes to generate a heatmap.")
}
```



11. Create a Volcano Plot

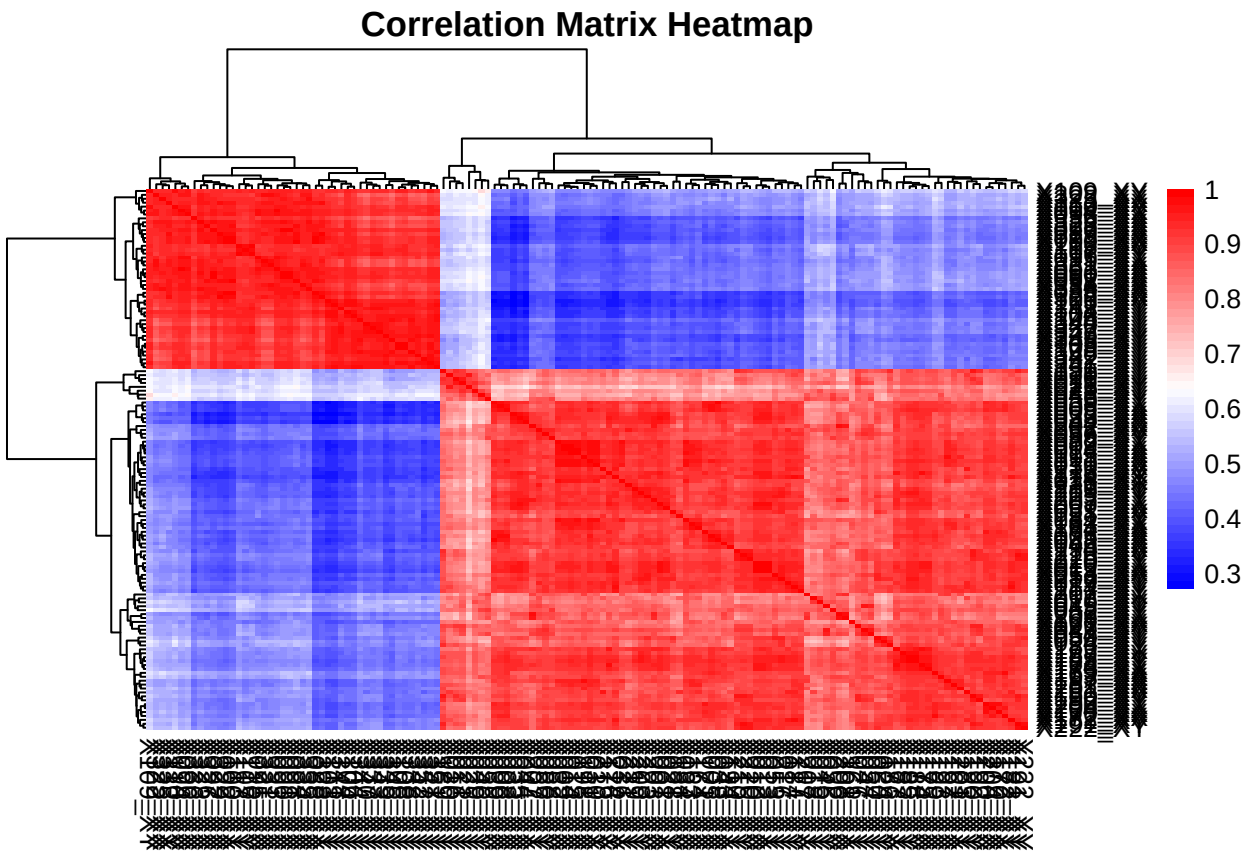
We generate a volcano plot to visualize the relationship between fold change and significance.

```
significant_genes$logP <- -log10(significant_genes$PValue)
ggplot(significant_genes, aes(x = logFC, y = logP)) +
geom_point(aes(color = FDR < 0.05)) + scale_color_manual(values =
c("pink", "red")) + labs(title = "Volcano Plot", x = "Log Fold Change",
y = "-Log10 P-Value") + theme_minimal()
```

12. Compute and Plot Correlation Matrix

We compute the correlation matrix of the normalized counts and visualize it using a heatmap.

```
correlation_matrix <- cor(normalized_counts)
pheatmap(correlation_matrix, main = "Correlation Matrix Heatmap", color
= colorRampPalette(c("blue", "white", "red"))(50))
```



13. MA Plot

We create an MA plot to visualize the relationship between mean expression levels and log fold changes.

Log fold change (logFC) is a statistical measure used to quantify the relative difference in gene expression between two groups (e.g., different conditions or treatments). It is often used in transcriptomic studies to compare gene expression levels.

```
plotMD(lrt, column = 1, main = "MA Plot", xlab = "Average Log CPM", ylab = "Log Fold Change")
```

```
## Warning in plot.window(...): "column" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "column" is not a graphical parameter
```

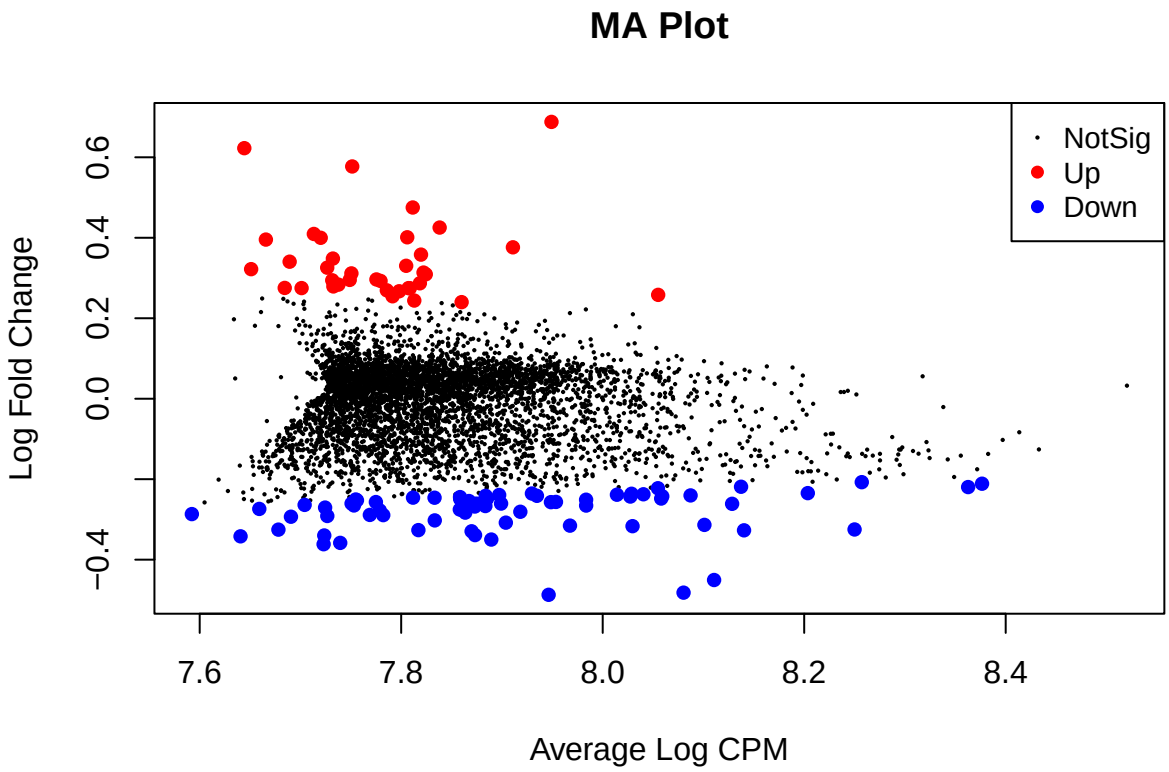
```
## Warning in axis(side = side, at = at, labels = labels, ...): "column" is not a
## graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "column" is not a
## graphical parameter
```

```
## Warning in box(...): "column" is not a graphical parameter
```

```
## Warning in title(...): "column" is not a graphical parameter
```

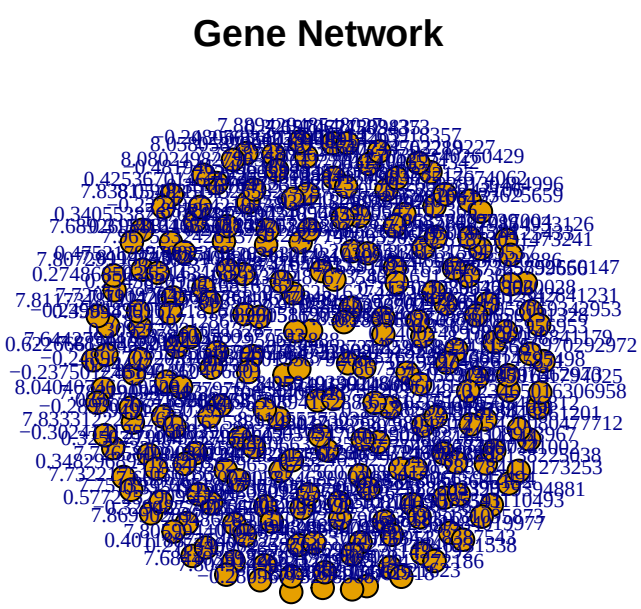
```r
abline(h = c(-1, 1), col = "blue")
```



#14. Create Gene Network

We construct a gene network using significant genes, visualizing their relationships.

```r
gene_network <- graph_from_data_frame(d = significant_genes, directed = FALSE)
plot(gene_network,
     vertex.size = 10,
     vertex.label.cex = 0.7,
     vertex.label.dist = 1,
     vertex.label.degree = -pi/2,
     main = "Gene Network")
```



15. K-means Clustering

K-means clustering is a popular unsupervised machine learning algorithm used to group similar data points into a specified number of clusters (K). It works by partitioning the data into K clusters where each data point belongs to the cluster with the nearest mean. This is useful for identifying patterns or groupings within the data.

15.1 Perform K-means Clustering

We perform K-means clustering on the transposed normalized counts data.

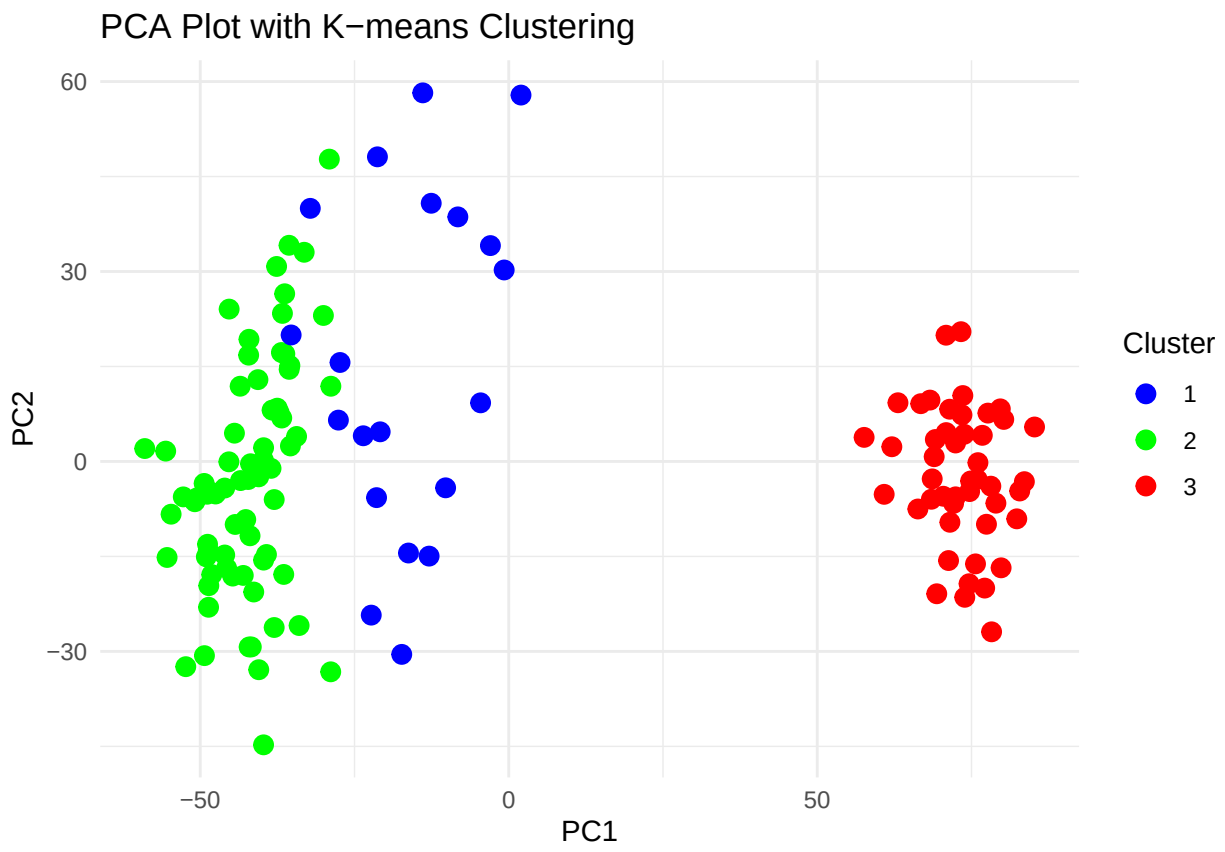
```
set.seed(123)
kmeans_result <- kmeans(t(norm_counts), centers = 3)
```

15.2 Clustering Visualization

Finally, we visualize the PCA results with K-means clustering results.

```
pca_data$Cluster <- as.factor(kmeans_result$cluster)
ggplot(pca_data, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point(size = 3) +
```

```
labs(title = "PCA Plot with K-means Clustering", x = "PC1", y = "PC2") +
theme_minimal() +
scale_color_manual(values = c("blue", "green", "red"))
```



16.Conclusion

This analysis provides insights into the differential gene expression and clustering patterns of samples, highlighting potential biological differences between the XX and XY groups. Each visualization helps in understanding the underlying structure of the data and the significant genes involved in the observed differences.

This markdown document gives a comprehensive overview of the code and its purpose, making it easy to understand the workflow and analysis conducted on the RNA-seq data.