BBCC2025 Tutorial: Introductory bulk RNA-seq analysis in R:

From raw counts to biological interpretation.

Claudia Angelini

Istituto per le Applicazioni del Calcolo "M. Picone"
Consiglio Nazionale delle Ricerche

2025-12-03

Contents

INTRODUCTION

(Bulk) RNA sequencing (RNA-seq) is a high-throughput technique used to quantify the RNA molecules present in a biological sample at the gene-wide scale. By capturing and sequencing RNA at a specific time point and under a particular condition, bulk RNA-seq provides a comprehensive snapshot of transcriptional activity, allowing researchers to investigate cellular states, molecular responses, and regulatory mechanisms with high sensitivity and accuracy. It aims to answer questions like:

- Which genes are active?
- How much is each gene expressed?
- How does gene expression change between conditions (e.g., healthy vs. diseased, treated vs. untreated)?
- Which biological pathways are involved, activated, or deactivated by the treatment, or affected by the experimental condition?

A bulk RNA-seq experiment typically begins with RNA extraction from biological samples, followed by enrichment of relevant RNA species and conversion to complementary DNA (cDNA). We use the term bulk RNA-seq experiment to distinguish it from single-cell RNA-seq experiments. In the first, gene expression is averaged across pools of cells (usually several thousand), whereas in the latter, it is observed for each cell. However, when it is clear from the context, we might omit the word 'bulk'.

Sequencing libraries are then generated and processed on high-throughput platforms such as Illumina, producing millions of short reads. These sequencing reads undergo computational processing to assess

quality, align to a reference genome or transcriptome, and quantify the abundance of each gene or transcript. The resulting data include raw sequence reads, alignment files, gene-level count matrices, and normalized expression values.

Ultimately, bulk RNA-seq produces quantitative data on gene expression averaged across many cells and enables downstream analyses such as differential gene expression, pathway and functional enrichment, and comparative transcriptomics across conditions, treatments, developmental stages, or disease states. Moreover, bulk RNA-seq data can be used to train machine learning models to address more complex questions, such as disease classification, patient stratification, and survival analysis. Through these applications, RNA-seq has become an essential tool in functional genomics, molecular biology, and biomedical research.

THE RNA-SEQ DATA

The primary output from a sequencer consists of millions of short reads (e.g., 50–150 bases) plus quality scores. Short reads are small DNA fragments sequenced from RNA-derived cDNA, representing pieces of transcripts that, when assembled or mapped, reveal how much each gene is expressed in the sample. Such reads are stored in FASTQ files.

For each sample, you usually have one or two FASTQ files containing the sequences, depending on the library. There are two main library types:

- **Paired-end reads**: two FASTQ files per sample (usually denoted as R1 and R2)
- **Single-end reads**: one FASTQ file per sample.

A bulk RNA-seq experiment usually involves collecting data from several samples in two or more experimental conditions. The analysis requires setting up computational pipelines.

GENERAL OVERVIEW OF THE RNA-SEQ COMPUTATIONAL WORKFLOW

The analysis workflow of bulk RNA-seq experiments consists of a series of steps:

1. Quality control (QC) of raw sequencing reads (i.e., **Fastq files** and **sample metadata**).
2. Filtering, trimming, adapter removal, and other sequence processing ⇒ Fastq files.
3. Alignment to a reference genome or to transcripts ⇒ typical output is SAM/BAM alignment files.
4. Gene-level quantification or Isoform-level quantification ⇒ typical output is a **raw count matrix**.
5. Quality control, Normalization, Batch correction

6. Differential Expression (DE) Analysis.
7. Functional Enrichment and Pathway Analysis.
8. Other post-processing analysis, such as Clustering, Machine learning algorithms, etc.

Running the entire workflow is typically computationally intensive and requires combining different tools, written in various languages and environments. In particular, steps 1-4 refer to the main pre-processing, steps 5-6 to the primary statistical analysis, and steps 7-8 to other downstream analysis.

This tutorial is entirely based on the R language and covers steps 5, 6, and 7 of the workflow above, assuming a gene expression count matrix and the corresponding sample metadata are available. In particular, we will use **DESeq2** (*Love, Huber, and Anders 2014*,https://doi.org/10.1186/s13059-014-0550-8) for differential expression analysis, which is one of the most popular and robust methods for RNA-seq analysis.

For more information about the DESeq2 analysis, see https://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html

INSTALL AND LOAD THE R PACKAGES

1. Install R (https://www.r-project.org).
2. R Studio (https://posit.co/download/rstudio-desktop/).
3. Install the following R packages (listed below).

4. Load the packages in the R environment.

```
# Load libraries
  library(airway) ## For the example dataset
  library(ggplot2)
  library(dplyr)
  library(pheatmap)
  library(RColorBrewer)
  library(DESeq2)
  library(AnnotationDbi)
  library(org.Hs.eg.db)
  library(clusterProfiler)
  library(msigdbr)
  library(apeglm)
  library(ashr)
```

READ THE DATA

For this tutorial, we will assume that you have obtained a **read count matrix** from raw fastq reads obtained from an Illumina sequencing run or another high-throughput sequencing experiment.

- A **read count matrix X** contains un-normalized counts, where the value in the i-th row and the j-th column of the matrix tells how many reads (or fragments, for paired-end RNA-seq) can be assigned to gene i in sample j.
- Moreover, to proceed with the analysis, you also need a **sample metadata table** with sample IDs, condition, batch, etc., that describes the experimental design.

Raw count data can be easily downloaded from the NCBI Gene Expression Omnibus (GEO), Sequence Read Archive (SRA), or other public repositories, provided an accession number is available. Alternatively, you can obtain novel experimental data after suitable preprocessing.

To download RNA-seq data from GEO in R, you can use the **GEOquery** Bioconductor package to access processed count data. In fact, many GEO submissions for RNA-seq data include a processed gene count matrix as a supplementary file, which is ideal for immediate differential gene expression analysis. In such cases, you can use the *getGEO()* function to retrieve both the raw count matrix and metadata. The *getGEO()* function downloads the data and creates an **ExpressionSet** object in the standard Bioconductor format. The function *exprs()* extracts the counts; the function *pData()* extracts sample metadata.

SRA stores the raw sequence data. Downloading sequences from SRA usually involves command-line tools (like the **SRA Toolkit**) outside of R.

There are other online data repositories, such as **recount3**, which is an online resource containing RNA-seq gene, exon, and exon-exon junction counts, as well as coverage bigWig files for more than 10.000 studies in human and mouse. There are also specific repositories, such as the Genomic Data Commons (GDC), TGCA, and Linkedomics, among others. For many of these repositories, specific R packages are available for downloading the data in R.

For this illustrative example, the **count data** and **metadata** are available directly from the R package **airway**.

The study examined four primary human airway smooth muscle cell lines treated with 1 micromolar dexamethasone for 18 hours, with both treated and untreated samples for each cell line. It consists of 4 cell lines $\times$ 2 conditions (treated/untreated) = 8 samples.

```
# Dataset: Human airway smooth muscle cells treated with dexamethasone
# Study: Glucocorticoid effects on asthma-related gene expression
# Reference: Himes et al., PLoS ONE, 2014
# GEO Accession: GSE52778
```

```r
# Samples: 4 cell lines × 2 conditions (treated/untreated) = 8 samples
# Organism: Homo sapiens (Human)

data("airway")
print(airway)
```

```
## class: RangedSummarizedExperiment
## dim: 63677 8
## metadata(1): ''
## assays(1): counts
## rownames(63677): ENSG00000000003 ENSG00000000005 ... ENSG00000273492
##   ENSG00000273493
## rowData names(10): gene_id gene_name ... seq_coord_system symbol
## colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
## colData names(9): SampleName cell ... Sample BioSample
```

```r
# Extract count matrix
countData <- assay(airway)
head(countData)
```

```
##                 SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516
## ENSG00000000003        679        448        873        408       1138
## ENSG00000000005          0          0          0          0          0
## ENSG00000000419        467        515        621        365        587
## ENSG00000000457        260        211        263        164        245
## ENSG00000000460         60         55         40         35         78
## ENSG00000000938          0          0          2          0          1
##                 SRR1039517 SRR1039520 SRR1039521
## ENSG00000000003       1047        770        572
## ENSG00000000005          0          0          0
## ENSG00000000419        799        417        508
## ENSG00000000457        331        233        229
## ENSG00000000460         63         76         60
## ENSG00000000938          0          0          0
```

```r
dim(countData)
```

```
## [1] 63677     8
```

```r
# Extract sample information
colData <- colData(airway)
head(colData)
```

```
## DataFrame with 6 rows and 9 columns
##             SampleName     cell      dex    albut        Run avgLength
##               <factor> <factor> <factor> <factor>   <factor> <integer>
## SRR1039508 GSM1275862  N61311     untrt    untrt SRR1039508       126
## SRR1039509 GSM1275863  N61311     trt      untrt SRR1039509       126
## SRR1039512 GSM1275866  N052611    untrt    untrt SRR1039512       126
## SRR1039513 GSM1275867  N052611    trt      untrt SRR1039513        87
## SRR1039516 GSM1275870  N080611    untrt    untrt SRR1039516       120
## SRR1039517 GSM1275871  N080611    trt      untrt SRR1039517       126
##             Experiment    Sample    BioSample
##               <factor>  <factor>     <factor>
## SRR1039508  SRX384345 SRS508568 SAMN02422669
## SRR1039509  SRX384346 SRS508567 SAMN02422675
```

```
## SRR1039512  SRX384349 SRS508571 SAMN02422678
## SRR1039513  SRX384350 SRS508572 SAMN02422670
## SRR1039516  SRX384353 SRS508575 SAMN02422682
## SRR1039517  SRX384354 SRS508576 SAMN02422673
```

```
## select the metadata of interest
colData<-as.data.frame(colData[, c("SampleName","cell", "dex")])
```

In this example,

- the object **countData** contains the raw counts. In our case, it contains 63677 genes across eight samples.
- the object **colData** contains the sample metadata. In our case, it contains the information about the eight samples.

CREATE DESeq2 OBJECT and DEFINE THE DESIGN FORMULA

To create the **DESeqDataSet** object, we will need - the **count matrix**, - the **metadata** table as input. Moreover, - We will also need to specify a **design formula**.

The **design formula** is a mathematical formula, such as ~ *condition* or ~ *treatment + genotype*, that you must provide to define the variables for the statistical model.

The **design matrix** (as in regression models) is the numerical matrix automatically generated from this formula, which specifies how to group the samples and identify coefficients for each factor of interest. More generally, a design formula instructs statistical software on the known sources of variation to control for, as well as the factor of interest to test in differential expression analyses. The last factor entered in the formula should be the condition of interest.

There are different ways to define a design formula.

- Example (Single factor): ~ *condition*: Analyzes the difference between groups defined by the condition variable. The condition variable can have two or more levels.
- Example (Two factors): * ~ treatment + genotype* : Analyzes the main effects of treatment and genotype separately. While * ~ treatment * genotype*: Includes also the interaction effect.
- Example (batch correction): ~ *batch + condition*: Analyzes the difference between groups defined by the condition variable while correcting for some batch variables.

More complex formulas can be defined for experiments involving multiple factors.
For more information on how you can create design matrices for gene expression experiments, you can look at *Law et al 2020* "A guide to creating design matrices for gene expression experiments" (https://doi.org/10.12688/f1000research.27893.1).

Define the design matrix or formula:

For the airway experiment, you can use two different design formulas: *design= ~ cell + dex* or *design= ~ dex*.

Both formulas allow testing the dexamethasone effect (i.e., treated versus untreated), with or without controlling for cell lines.

- When using *design= ~ dex*, you assume the experiment has exactly two conditions (e.g., treated vs untreated) and that the cell type is not relevant. In this case, you can create a simple sample metadata table with a column dex containing the two levels of the condition.
- When using *design= ~ cell + dex*, you assume the experiment has two conditions (e.g., treated vs untreated), but you want to correct for the cell type. In this case, the sample metadata table must have a column named *cell* containing the cell type and a column named *dex* containing the two levels of the condition.

Note that, by default, R will choose a reference level for factors based on alphabetical order. However, you can specify the reference level using the *relevel()* function.

Create and extract information from a DESeqDataSet object

The **DESeqDataSet** object contains all information about the experimental setup, read counts, and design formulas. Let *dds* be a DESeqDataSet object; the following functions can be used to access this information separately:

- *rownames(dds)* shows which features are used in the study (e.g., genes),
- *colnames(dds)* displays the studied samples,
- *counts(dds)* displays the count table, and
- *colData(dds)* displays the experimental setup (i.e., the metadata associated with the samples).

```
colData$dex <- relevel(colData$dex, ref = "untrt")  # Set untreated as reference

## create the DESeqDataSet object
#dds <- DESeqDataSet(airway, design = ~  dex)
#
dds <- DESeqDataSetFromMatrix(countData,colData, design = ~  dex)

dds
```

```
## class: DESeqDataSet
## dim: 63677 8
## metadata(1): version
## assays(1): counts
## rownames(63677): ENSG00000000003 ENSG00000000005 ... ENSG00000273492
##   ENSG00000273493
## rowData names(0):
## colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
## colData names(3): SampleName cell dex
```

The dds object contains 63677 genes across 8 samples. Moreovere you can note that gene names are given as ENSEMBL IDs.

Filtering

Remove genes that have almost no or limited expression in any of the given samples.

```
## Filtering low expressed genes
keep <- rowSums(counts(dds)) >= 10
dds <- dds[keep, ]
dds
```

```
## class: DESeqDataSet
## dim: 22369 8
## metadata(1): version
## assays(1): counts
## rownames(22369): ENSG00000000003 ENSG00000000419 ... ENSG00000273487
##   ENSG00000273488
## rowData names(0):
## colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
## colData names(3): SampleName cell dex
```

After filtering for lowly expressed genes, we have 22369 genes in the dds object.

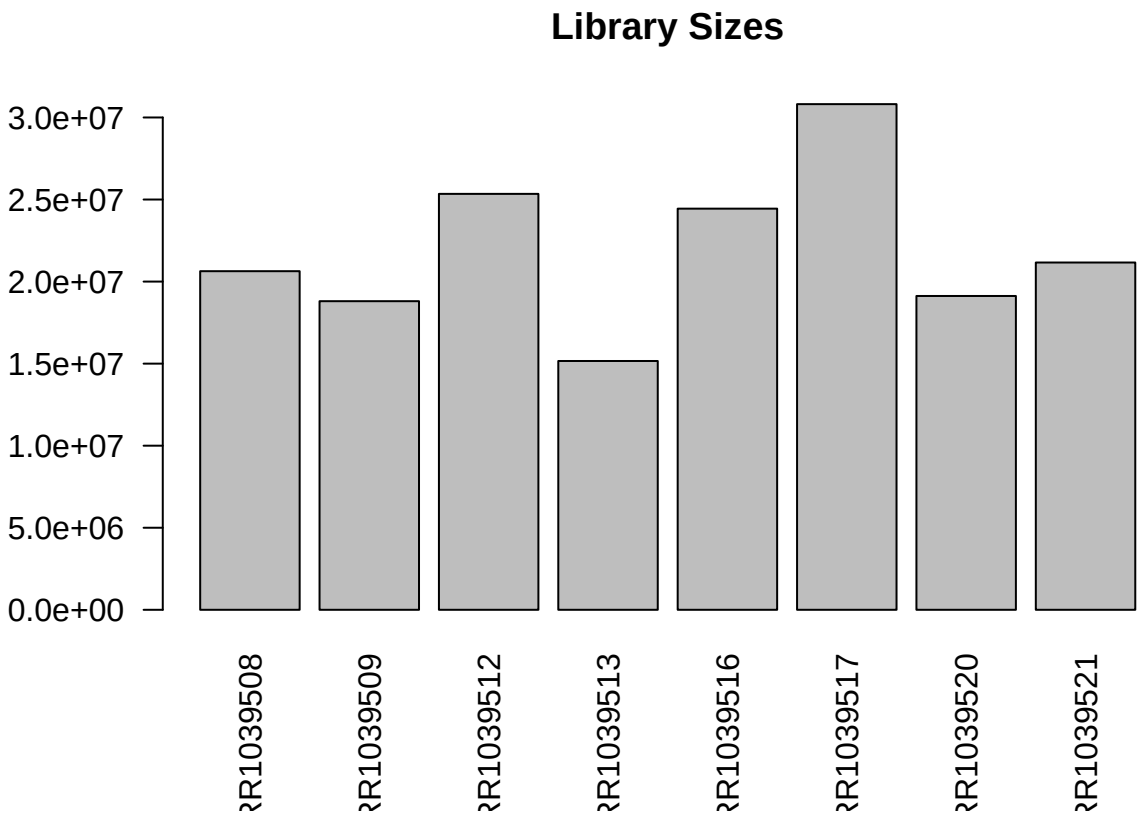
NORMALIZATION, QUALITY CONTROL AND VISUALIZATION

Normalization is necessary for bulk RNA-seq data to correct for technical variations, such as sequencing depth and library size, which can obscure true biological differences. It allows meaningful comparisons between samples by adjusting read counts to account for factors such as total RNA content, ensuring that observed differences are biological rather than technical artifacts. Without normalization, differences in sequencing depth could incorrectly suggest that a gene is up-regulated when it is not, leading to unreliable results.

In particular, when looking at raw RNA-seq count data, two important factors influence the observed values and need to be taken into account.

- The first factor is the sequencing depth or **library size**, that is, the total number of reads mapped to the genome.

```
# QC: Library sizes
barplot(colSums(counts(dds)), las = 2, main = "Library Sizes")
```



The *estimateSizeFactors()* function allows estimating the library size. Then, normalized counts can be obtained using the Median of Ratios (RLE - Relative Log Expression) method.

```
dds <- estimateSizeFactors(dds)
sizeFactors(dds)
```

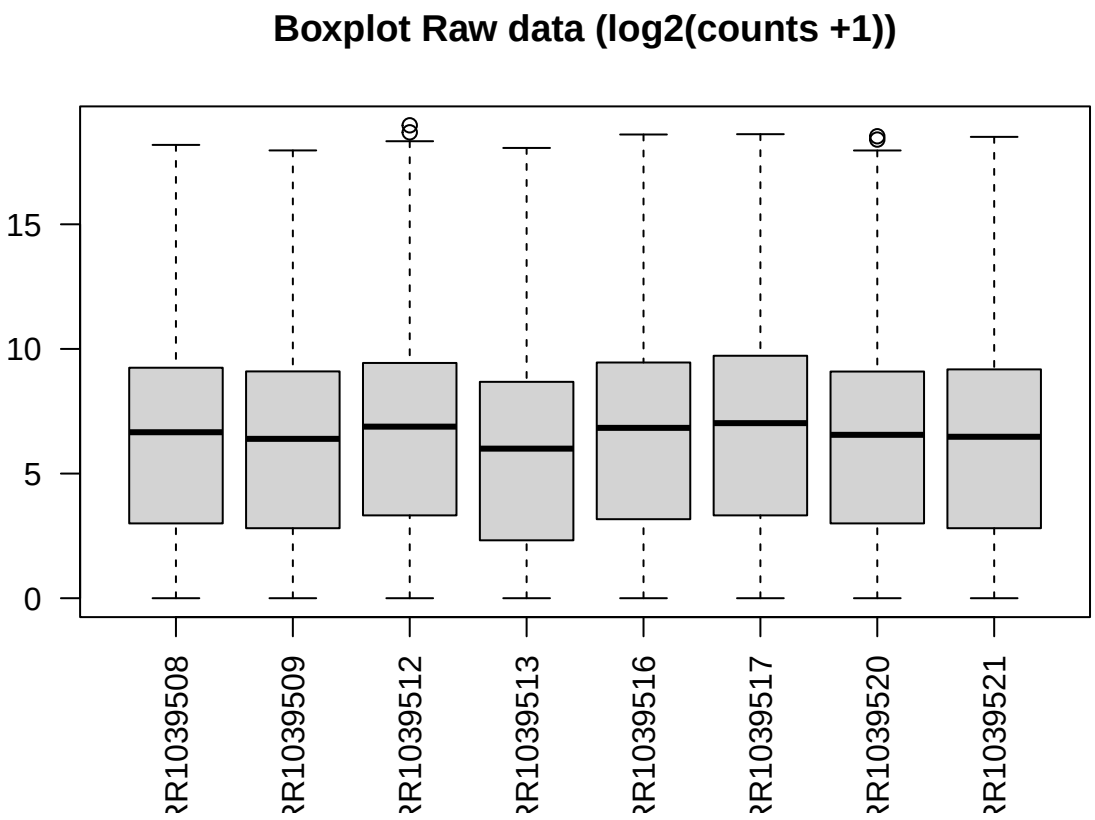
```
## SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516 SRR1039517 SRR1039520
##  1.0236476  0.8961327  1.1795171  0.6700291  1.1776714  1.3990527  0.9208046
## SRR1039521
##  0.9445520
```

```
normalized_counts<-counts(dds, normalized=TRUE)
head(normalized_counts)
```

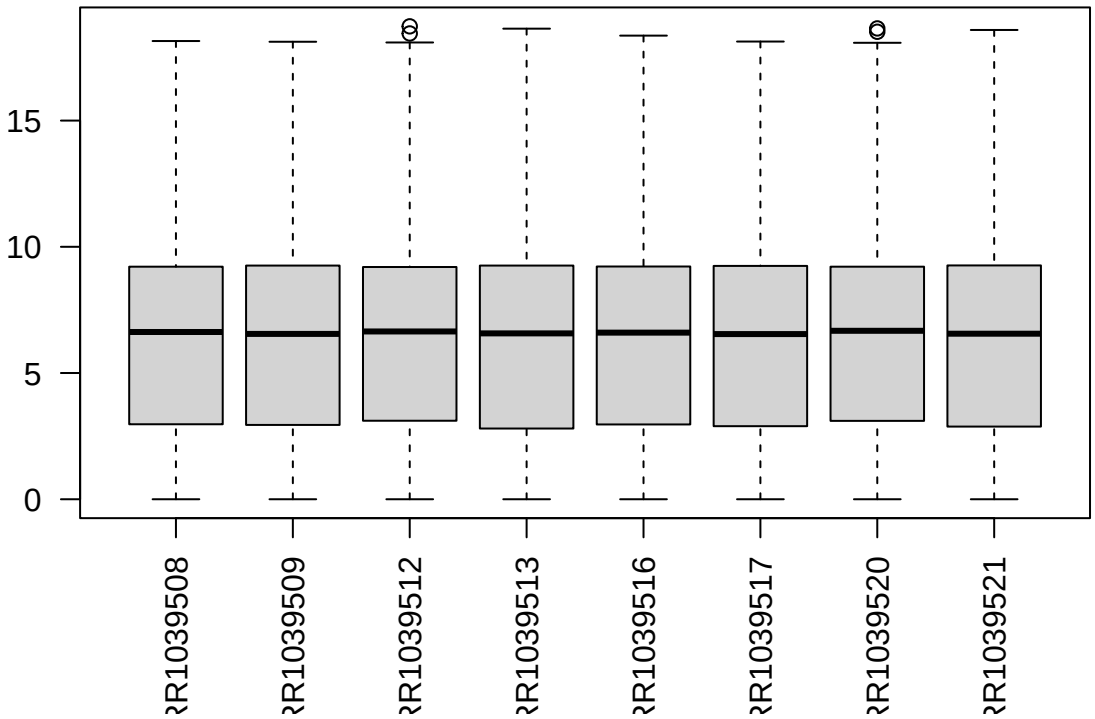
```
##                 SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516
```

```
## ENSG00000000003  663.31418  499.92596  740.13337  608.92882   966.3137
## ENSG00000000419  456.21167  574.69167  526.48662  544.75250   498.4413
## ENSG00000000457  253.99365  235.45620  222.97260  244.76551   208.0377
## ENSG00000000460   58.61392   61.37484   33.91218   52.23654    66.2324
## ENSG00000000971 3175.89749 4105.41876 5236.88869 6345.99347  5707.0251
## ENSG00000001036 1399.89576 1185.09234 1469.24528 1314.86835  1209.1659
##                 SRR1039517 SRR1039520 SRR1039521
## ENSG00000000003  748.36352  836.22522  605.57809
## ENSG00000000419  571.10072  452.86483  537.82110
## ENSG00000000457  236.58866  253.03958  242.44298
## ENSG00000000460   45.03047   82.53652   63.52218
## ENSG00000000971 7881.76172 5621.17109 8464.33018
## ENSG00000001036 1028.55311 1475.88321 1174.10159
```

```
boxplot(log2(counts(dds)+1), main="Boxplot Raw data (log2(counts +1))", las=2)
```




```
boxplot(log2(normalized_counts+1), main="Boxplot Normalized data (RLE)  (log2(counts +1))", las=2)
```



- The second factor is the **gene length**, i.e., the number of bases covering a gene. It is expected that larger genes, for a given level of transcription, will have a higher number of gene counts.

In general, various other factors affect transcript quantification in RNA-seq data, including library preparation, sequencing runs, GC-content, and other batch-related variability.

There are three main RNA-seq quantification measures that you could consider:

- **CPM**: Counts per million (CPM) mapped reads are the number of raw reads mapped to a transcript, scaled by the number of sequencing reads in your sample, multiplied by a million.
- **FPKM/RPKM**: FPKM (fragments per kilobase of transcript per million fragments mapped) for paired-end data and RPKM (reads per kilobase of transcript per million reads mapped) for single-end data, correct for variations in library size and gene length
- **TPM**: Transcripts per million (TPM) represents the relative number of transcripts you would detect for a gene if you had sequenced one million full-length transcripts

To compute one of the above measures, you need information about the **gene length**.

We will not cover this part in this tutorial.

For this part, we suggest reading

- The edgeR package: https://bioconductor.org/packages/release/bioc/html/edgeR.html
- The EDAseq package: https://bioconductor.org/packages/release/bioc/html/EDASeq.html
- The NOISeq package: https://www.bioconductor.org/packages/release/bioc/html/NOISeq.html

More sophisticated approaches also include the removal of batch effects. For this part, we suggest reading

- The SVA package: https://bioconductor.org/packages/release/bioc/html/sva.html.
- The RUVseq package: https://bioconductor.org/packages/release/bioc/html/RUVSeq.html.

Other transformation (i.e., variance stabilizing transform and rlog)

To test for differential expression, DESeq2 operates on raw counts and uses discrete distributions as described in the next section on differential expression. However, for other downstream analyses – e.g., visualization or clustering – it may be helpful to work with transformed count data and apply a suitable transformation.

RNA-seq data are usually counts, modelled as a negative binomial distribution. For this type of data, the mean and dispersion are related, i.e., the dispersion depends on the mean. Therefore, they are a classic example of heteroscedasticity (i.e., non-equal variance).

This can be easily visualized estimating the disepersion and plotting the mean versus the dispersion.

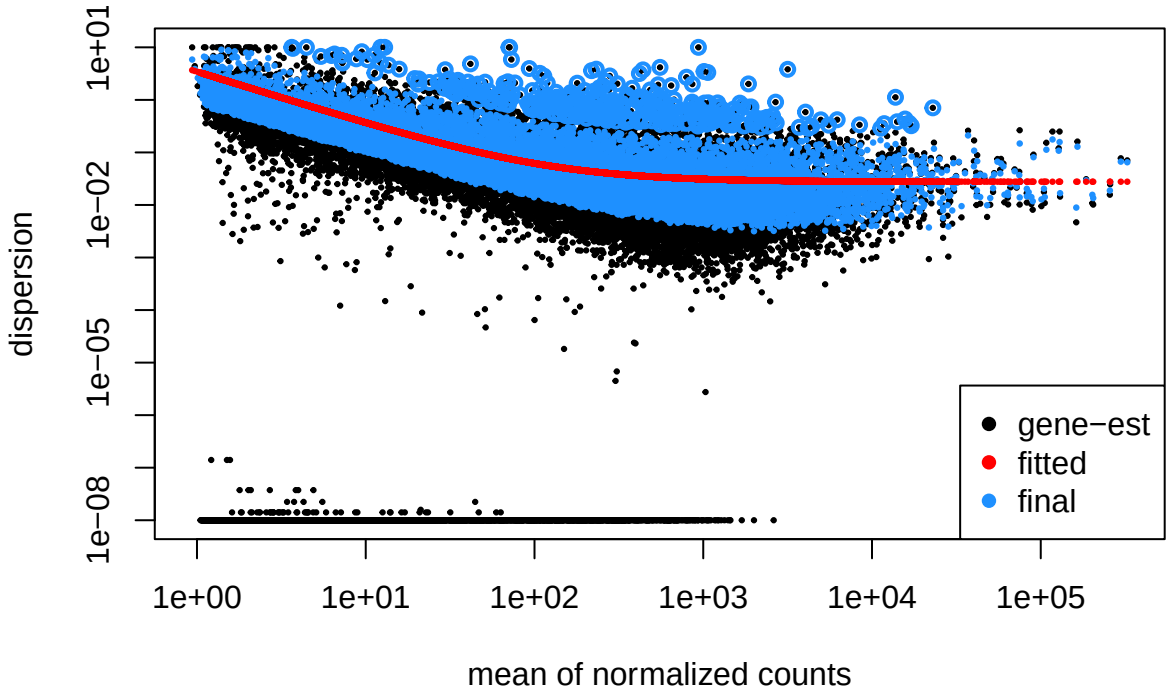
```
dds <- estimateDispersions(dds)
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

```
plotDispEsts(dds)
```



The mean-dispersion plot in DESeq2, generated using *plotDispEsts()**, visualizes the dispersion (variance) of gene expression levels against the mean of normalized counts for each gene. The plot shows how DESeq2 shrinks the gene-wise dispersion estimates toward a fitted dispersion-mean curve (red line) to provide more accurate final estimates (blue dots), which is essential for reliable differential expression analysis.

The DESeq2 package contains two transformations: the variance-stabilizing transformation (vst) and the regularized log transformation (rlog).

- **vst** fits dispersion-mean relation(s) and then transforms the count data (normalized by division by the size factors or normalization factors), yielding a matrix of values which are now approximately homoskedastic (having constant variance along the range of mean values). The transformation also normalizes with respect to library size.
- The **rlog** applies a 'regularized log' transformation. It transforms count data to the log2 scale to minimize differences between samples for rows with small counts and normalizes by library size.

These transformations are useful for detecting outliers or as input to machine learning techniques such as clustering or linear discriminant analysis.

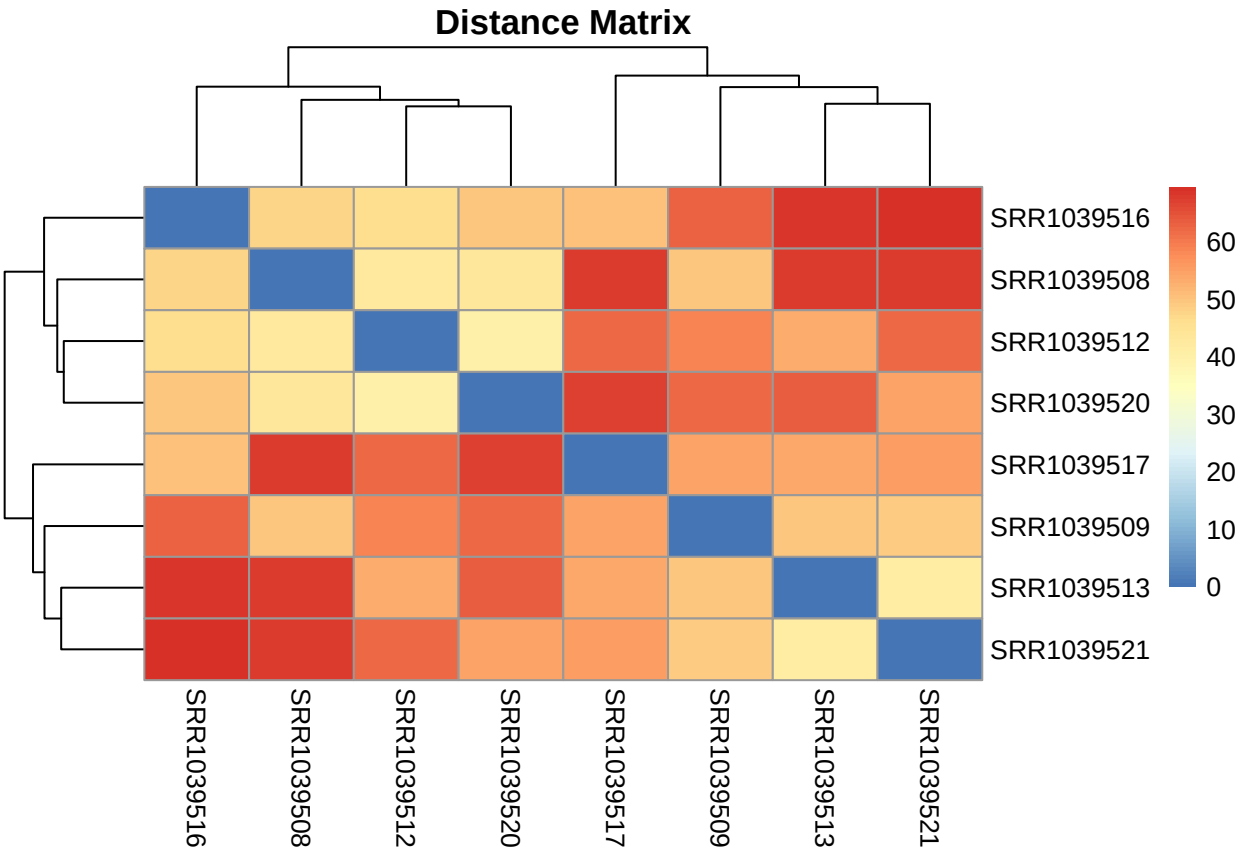
```
# Variance stabilization transform
vsd <- vst(dds, blind = FALSE)
# rlog
rld <- rlog(dds)
```

- Note that, the vst is less sensitive to size factors, which can be an issue when size factors vary widely.
- Note that neither the **rlog** transformation nor the **VST** matrices are used by the differential expression estimation in DESeq2, which always uses the raw count data, through generalized linear modeling, which incorporates knowledge of the variance-mean dependence. Instead, the rlog transformation and VST are proposed as separate functionalities for visualization, clustering, and other machine learning tasks.

Visualization (e.g., PCA, sample distance matrix, etc)

In the following explorative plots we will use the vst normalized counts.

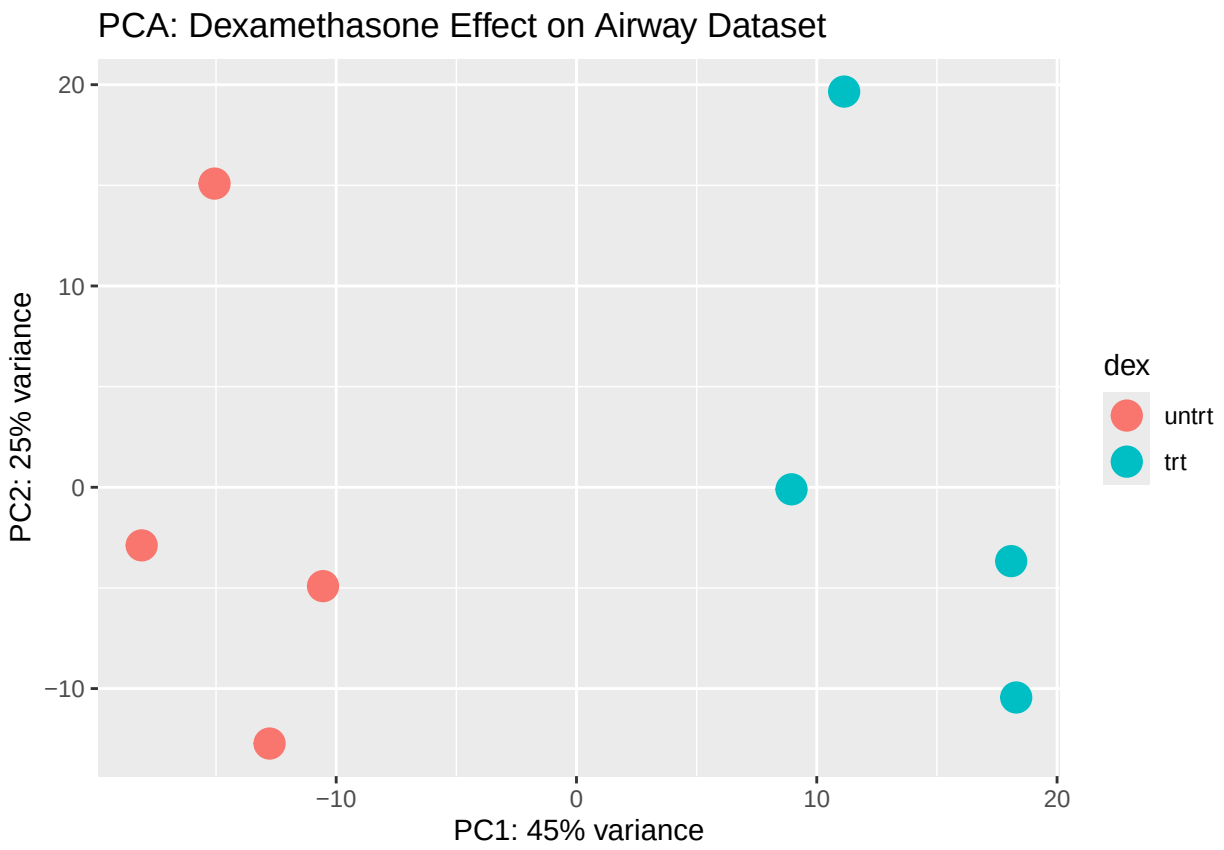
```
mat <- assay(vsd)
# sample distance matrix
sampleDists <- dist(t(mat)) ## matrix of dimension nsamples x nsamples with pairwise distances

pheatmap(as.matrix(sampleDists), clustering_distance_rows=sampleDists, clustering_distance_cols=sampleDi
```



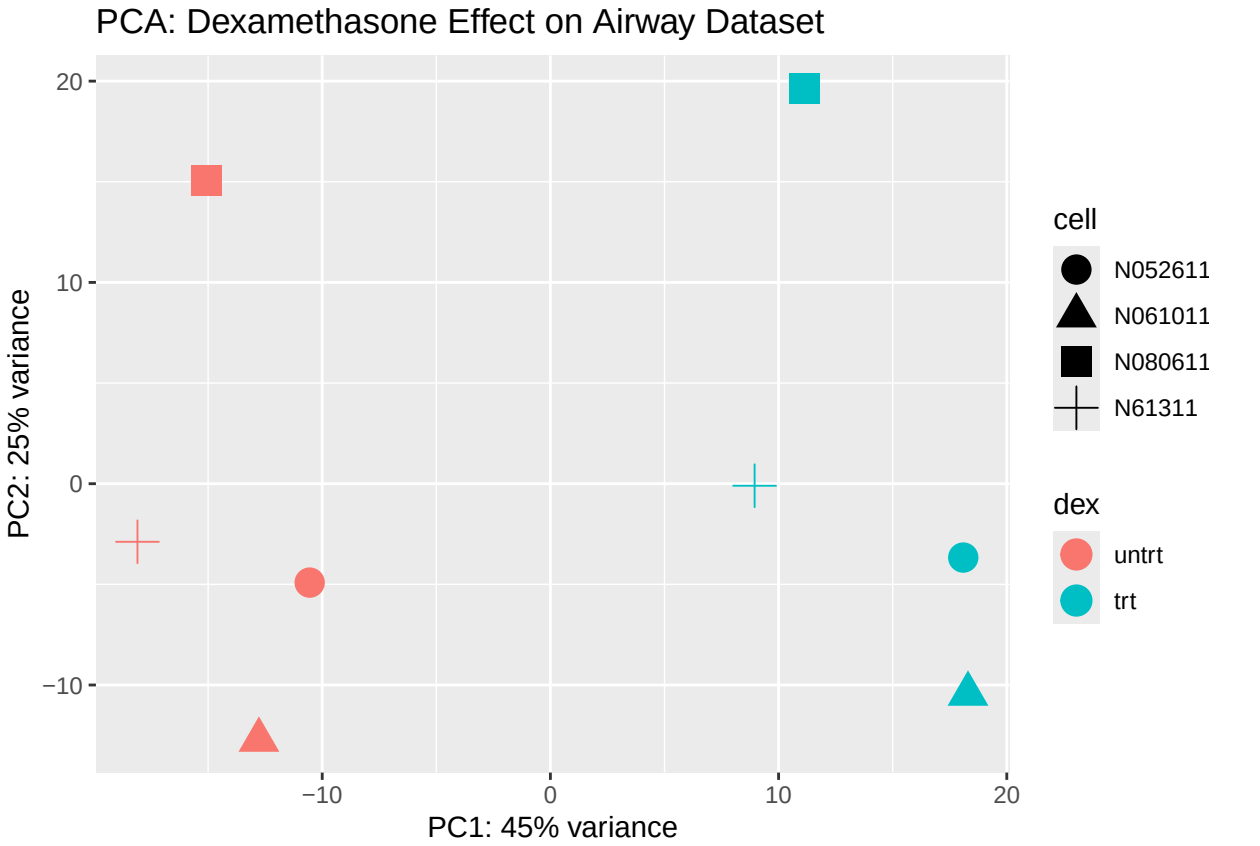

```
pcaData <- plotPCA(vsd, intgroup = "dex", returnData = TRUE)
```

```
## using ntop=500 top features by variance
```

```
percentVar <- round(100 * attr(pcaData, "percentVar"))

pca_plot <- ggplot(pcaData, aes(x = PC1, y = PC2, color = dex)) +
  geom_point(size = 5) +
  xlab(paste0("PC1: ", percentVar[1], "% variance")) +
  ylab(paste0("PC2: ", percentVar[2], "% variance")) +
  ggtitle("PCA: Dexamethasone Effect on Airway Dataset")

print(pca_plot)
```




```
pca_plot_new <- ggplot(pcaData, aes(x = PC1, y = PC2, color = dex, shape=cell)) +
  geom_point(size = 5) +
  xlab(paste0("PC1: ", percentVar[1], "% variance")) +
  ylab(paste0("PC2: ", percentVar[2], "% variance")) +
  ggtitle("PCA: Dexamethasone Effect on Airway Dataset")

print(pca_plot_new)
```



As you can see, the first principal component captures the treatment effect by distinguishing between treated and untreated samples. However, the second principal component captures the effect of the cell type.

We might consider and correct for cell types in the differential expression analysis (see Exercises).

DIFFERENTIAL EXPRESSION ANALYSIS

DESeq2 is a popular method for identifying differentially expressed genes from RNA-seq data.

It models the raw gene expression counts using the **negative binomial distribution to account** for accounting the extra variability in sequencing data.

It performs normalization, dispersion estimation, and correction for batch or confounding variables (in provided in the formula).

It fits a Generalized Linear Model (GLM) to identify genes with significant expression changes associated with the factor of interest. It returns a table that includes the genes, their fold changes, p-values, and adjusted p-values.

To run the differential expression analysis, we will use a single call to the function *DESeq()*. This function prints a message for each step it performs. Recall that DESeq2 does not use normalized counts; instead, it utilizes the raw counts and models the normalization within the GLM.

In particular, the *DESeq()* function corrects for sequencing depth and overdispersion. If you have already estimated size factors and dispersion, it will use the available estimates; otherwise, it will run the estimate before fitting the model. Moreover, by providing a suitable design formula, it can correct for batch effects or other confounding variables.

In our case, we have that the design formula includes only the treatment effect, therefore no cell type correction is applied (try the Exercise for correcting for cell types.).

```
# Run DESeq2 analysis
dds <- DESeq(dds)
```

```
## using pre-existing size factors
```

```
## estimating dispersions
```

```
## found already estimated dispersions, replacing these
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

```
## fitting model and testing
```

After fitting the model, we can explore the results and look and the contrast of interest. A **contrast** is a specific comparison between sample groups to find differentially expressed genes.

```
# Get results for dexamethasone effect
res <- results(dds, contrast = c("dex", "trt", "untrt"))
head(res)
```

```
## log2 fold change (MLE): dex trt vs untrt
## Wald test p-value: dex trt vs untrt
## DataFrame with 6 rows and 6 columns
##                   baseMean log2FoldChange     lfcSE      stat    pvalue
##                  <numeric>      <numeric> <numeric> <numeric> <numeric>
## ENSG00000000003  708.5979     -0.3788229  0.173155 -2.187769 0.0286865
## ENSG00000000419  520.2963      0.2037893  0.100742  2.022878 0.0430857
## ENSG00000000457  237.1621      0.0340631  0.126476  0.269325 0.7876795
## ENSG00000000460   57.9324     -0.1171564  0.301583 -0.388472 0.6976669
## ENSG00000000971 5817.3108      0.4409793  0.258776  1.704099 0.0883626
## ENSG00000001036 1282.1007     -0.2419097  0.119713 -2.020751 0.0433055
##                       padj
##                  <numeric>
## ENSG00000000003  0.138470
## ENSG00000000419  0.182998
## ENSG00000000457  0.929805
## ENSG00000000460  0.894231
## ENSG00000000971  0.297042
## ENSG00000001036  0.183498
```

```
summary(res, alpha = 0.05)  #Display a brief summary
```

```
##
## out of 22369 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)       : 1548, 6.9%
## LFC < 0 (down)     : 1146, 5.1%
## outliers [1]       : 51, 0.23%
## low counts [2]     : 3903, 17%
## (mean count < 4)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

In this example, genes are written as ENSEMBL IDs. It can be helpful to add gene symbols and ENTREZ IDs. Several packages support converting gene IDs. Here, we use the *mapIds()* function from the **AnnotationDbi**

packages.

Among other useful conversion tools, we recall the *bitr()* function from the **clusterProfiler** package and the *getBM()* function from the **biomaRt** package.

Note that not all ENSEMBL IDs have gene symbols and/or ENTRZID. Therefore, for some analyses, you should filter out observations with NA values.

```
# Add gene symbols and descriptions
res$symbol <- mapIds(org.Hs.eg.db,
                     keys = rownames(res),
                     column = "SYMBOL",
                     keytype = "ENSEMBL",
                     multiVals = "first")
```

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

```
res$entrez <- mapIds(org.Hs.eg.db,
                     keys = rownames(res),
                     column = "ENTREZID",
                     keytype = "ENSEMBL",
                     multiVals = "first")
```

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

It is possible to order results by statistical significance and/or to extract a list of DE genes based on different criteria.

```
# Order by adjusted p-value
resOrdered <- res[order(res$padj), ]
head(as.data.frame(resOrdered), 7)
```

```
##                   baseMean log2FoldChange      lfcSE     stat        pvalue
## ENSG00000152583   997.4447       4.602555 0.21174194 21.73662 9.245855e-105
## ENSG00000148175 11193.7232       1.451494 0.08489578 17.09736  1.552859e-65
## ENSG00000179094   776.6002       3.183883 0.20150941 15.80017  3.102848e-56
## ENSG00000134686  2737.9816       1.387168 0.09165870 15.13406  9.655331e-52
## ENSG00000125148  3656.2674       2.203468 0.14741913 14.94696  1.630150e-50
## ENSG00000120129  3409.0384       2.949013 0.20154644 14.63193  1.757388e-48
## ENSG00000189221  2341.7807       3.305227 0.22803072 14.49466  1.309529e-47
##                          padj  symbol entrez
## ENSG00000152583 1.702624e-100 SPARCL1   8404
## ENSG00000148175  1.429795e-61    STOM   2040
## ENSG00000179094  1.904631e-52    PER1   5187
## ENSG00000134686  4.445073e-48    PHC2   1912
## ENSG00000125148  6.003843e-47    MT2A   4502
## ENSG00000120129  5.393715e-45   DUSP1   1843
## ENSG00000189221  3.444997e-44    MAOA   4128
```

In general, you can identify DE genes accorditi to some thresholds-

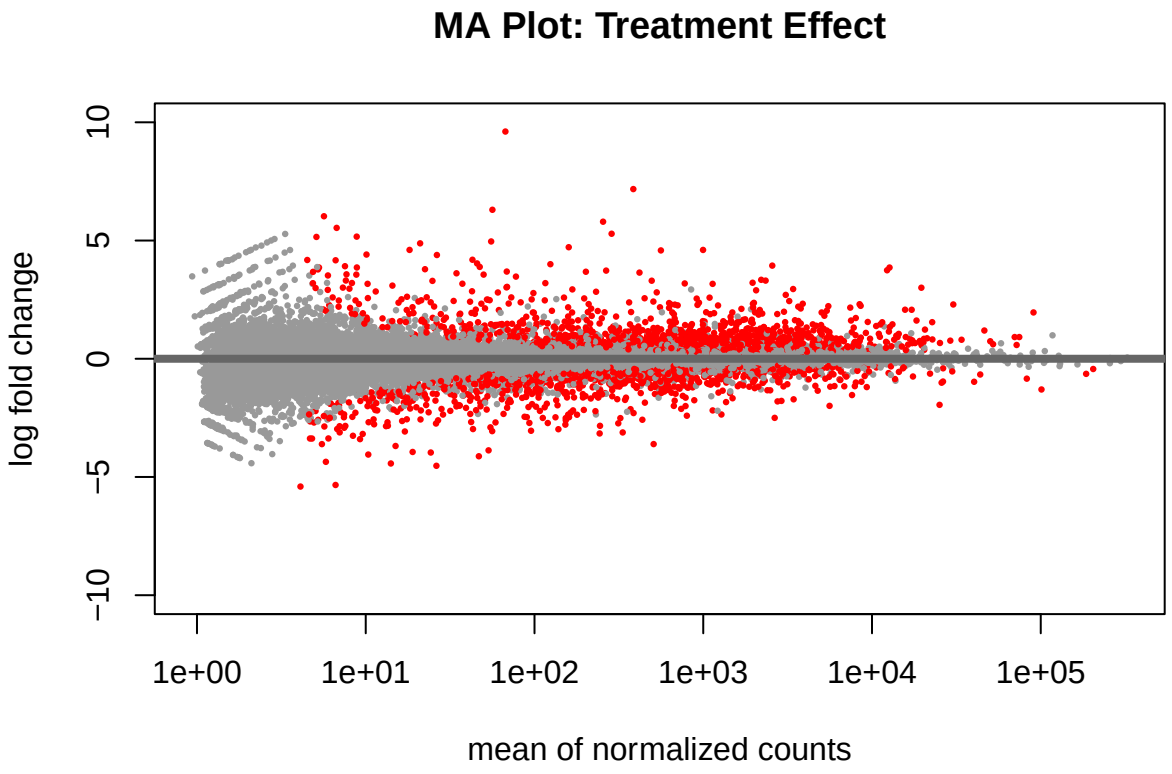
```
# Define thresholds
alpha <- 0.05
lfc_threshold <- 1

# Filter significant genes according to padj and log_fc
sig_genes <- subset(res, padj < alpha & abs(log2FoldChange) > lfc_threshold)
sig_df <- as.data.frame(sig_genes)
```

```r
# Get moderately significant genes (padj < 0.05, any fold change)
sig_genes_moderate <- subset(res, padj < alpha)
```

RESULTS and VISUALIZATION

MA Plot

```r
plotMA(res, ylim = c(-10, 10), alpha = 0.05,
       main = "MA Plot: Treatment Effect",
       colSig = "red", colLine = "grey40")
```



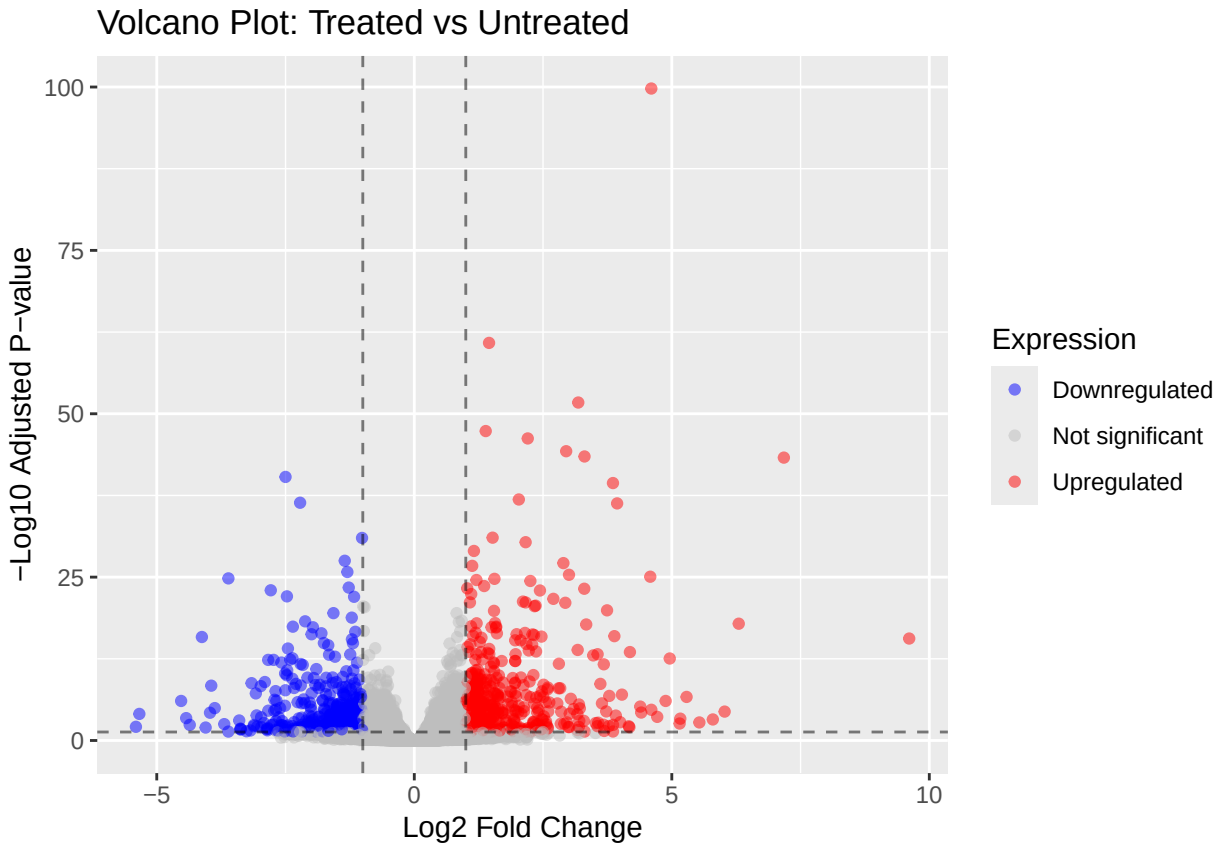
Volcano Plot

Convert DESeq2 result object into a data.frame

```r
res_df <- as.data.frame(res) # converts the object res into a dataframe

res_df$diffexpressed <- "Not Significant"
res_df$diffexpressed[res_df$log2FoldChange > lfc_threshold &
                       res_df$padj < alpha] <- "UP-regulated"
res_df$diffexpressed[res_df$log2FoldChange < -lfc_threshold &
                       res_df$padj < alpha] <- "DOWN-regulated"
```

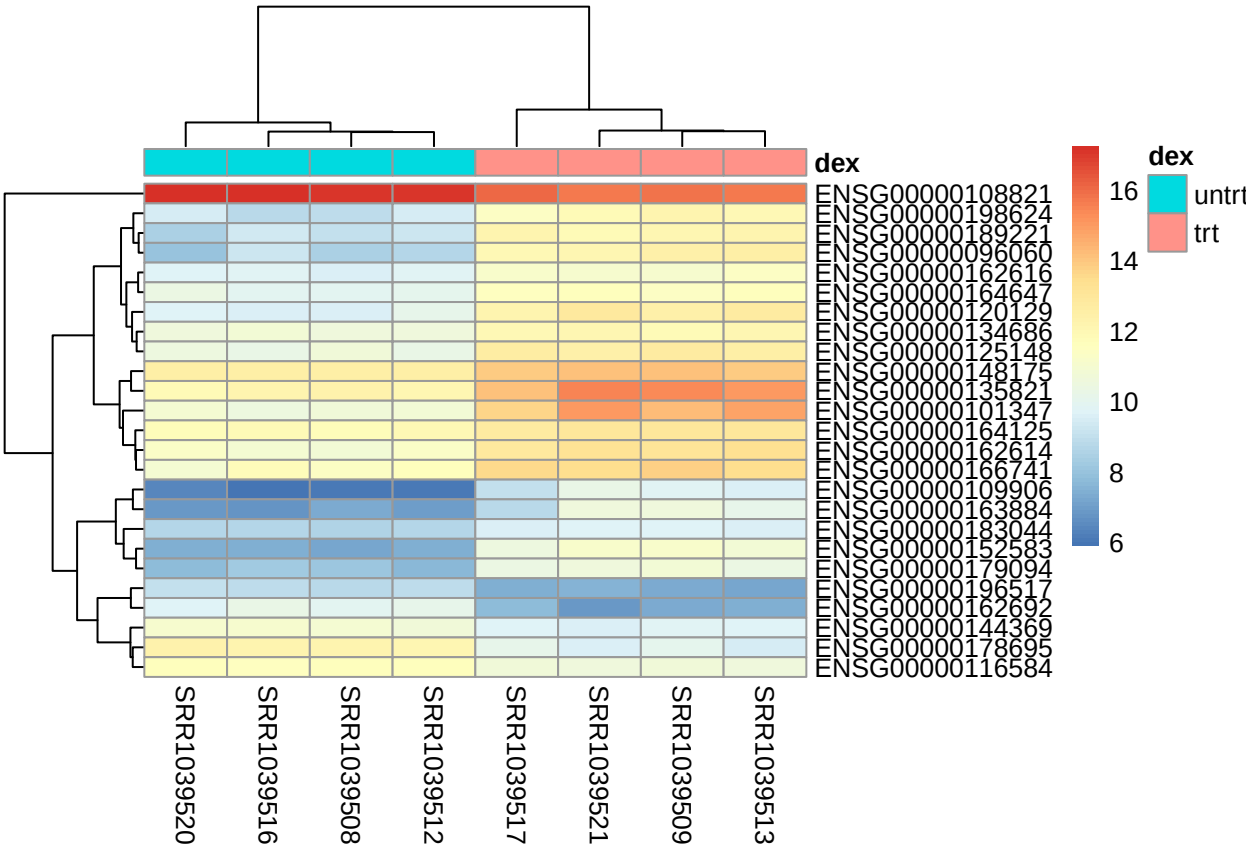
```r
volcano <- ggplot(res_df, aes(x = log2FoldChange, y = -log10(padj),
                              col = diffexpressed)) +
  geom_point(alpha = 0.5, size = 1.5) +
  scale_color_manual(values = c("blue", "grey", "red"),
                     labels = c("Downregulated", "Not significant", "Upregulated"),
                     name = "Expression") +
  geom_vline(xintercept = c(-lfc_threshold, lfc_threshold),
```

```
            linetype = "dashed", col = "black", alpha = 0.5) +
  geom_hline(yintercept = -log10(alpha),
             linetype = "dashed", col = "black", alpha = 0.5) +
  labs(title = "Volcano Plot: Treated vs Untreated",
       x = "Log2 Fold Change",
       y = "-Log10 Adjusted P-value")

print(volcano)
```



Heatmap

```
select <- head(order(res$padj), 25) ## take the top 25 genes ordered by adjussted pvalues

pheatmap(assay(vsd)[select, ], cluster_rows = TRUE, cluster_cols = TRUE,
         show_rownames = TRUE, annotation_col = as.data.frame(colData(dds)[, "dex", drop=FALSE]))
```



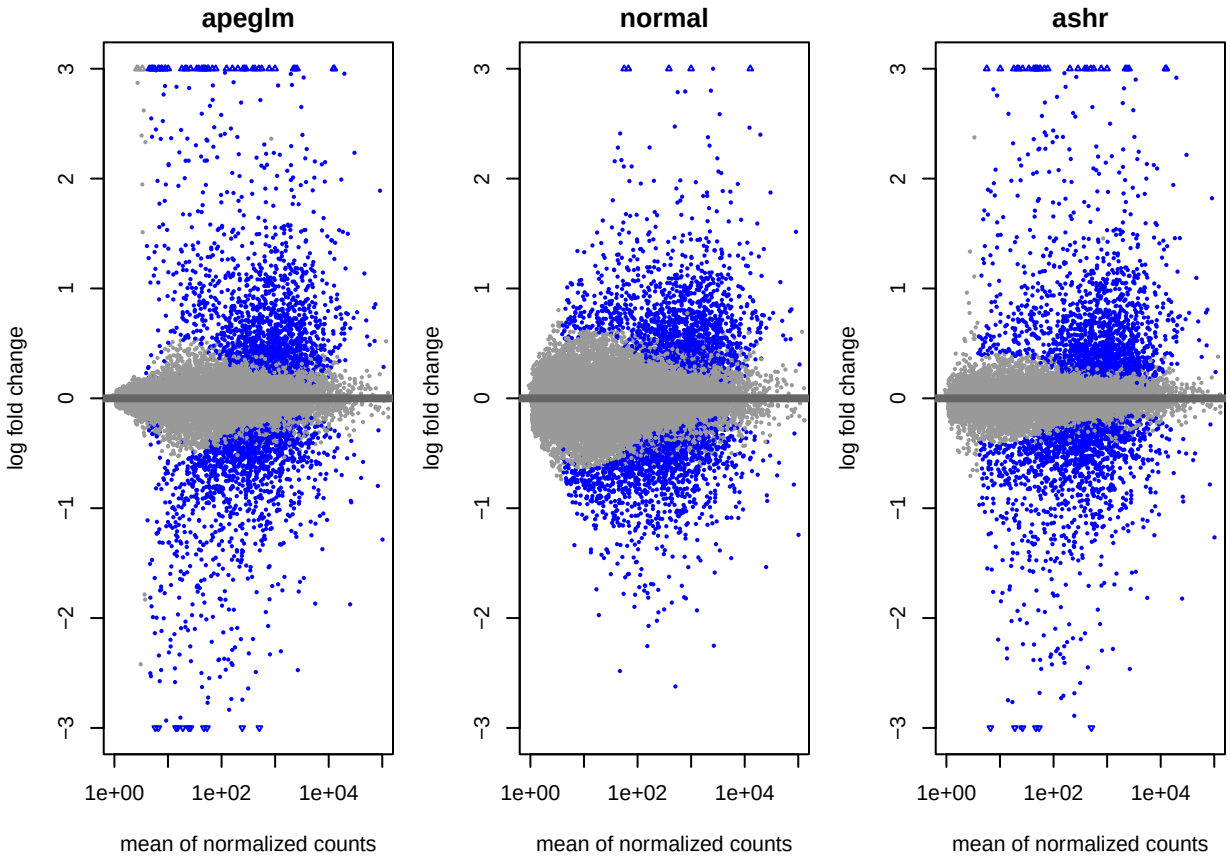
Shrinkage (optional)

The shrunken log fold changes are useful for ranking and visualization, without the need for arbitrary filters on low-count genes. The normal prior can sometimes lead to excessive shrinkage in certain datasets.

DESeq2 provides thee shrinkage options:

- **apeglm** is the adaptive t prior shrinkage estimator from the apeglm package (Zhu, Ibrahim, and Love 2018). As of version 1.28.0, it is the default estimator.
- **ashr** is the adaptive shrinkage estimator from the ashr package (Stephens 2016). Here, DESeq2 uses the ashr option to fit a mixture of Normal distributions as the prior, with method="shrinkage".
- **normal** is the original DESeq2 shrinkage estimator, an adaptive Normal distribution as prior.

```
resLFC <- lfcShrink(dds,  coef=2, type="apeglm")
resNorm <- lfcShrink(dds,  coef=2, type="normal")
resAsh <- lfcShrink(dds,   coef=2,type="ashr")

par(mfrow=c(1,3), mar=c(4,4,2,1))
xlim <- c(1,1e5); ylim <- c(-3,3)
plotMA(resLFC, xlim=xlim, ylim=ylim, main="apeglm")
plotMA(resNorm, xlim=xlim, ylim=ylim, main="normal")
plotMA(resAsh, xlim=xlim, ylim=ylim, main="ashr")
```



Exporting results to Tab delimited files

A plain-text file of the results can be exported using the base R functions write.table, write.csv or write.delim. We suggest using a descriptive file name indicating the variable and levels which were tested.

```
write.table(as.data.frame(resOrdered),
          file="condition_treated_vs_control_results.txt", quote = FALSE, sep = "\t", row.names = FALSE)
```

PATHWAY ENRICHMENT ANALYSIS

There are at least two broad categories of methods:

- **Over-Representation Analysis (ORA)**: It uses a list of DE genes, or up-regulated genes, down-regulated genes, and tests whether specific pathways contain more genes than expected by chance. It is fast and straightforward.

Example of Tools: Enrichr, clusterProfiler (enrichGO/enrichKEGG), gProfileR2

- **Gene Set Enrichment Analysis (GSEA)**: It uses the entire ranked gene list (no DE cutoff) and detects subtle but coordinated changes in pathways. It is often more biologically informative.

Example of Tools: GSEA, fgsea, clusterProfiler (gseGO/gseKEGG).

Both approaches use Curated Gene Sets such as

- GO (Gene Ontology) — biological processes (BP), molecular functions (MF), cellular components (Cc).
- KEGG — metabolic and signaling pathways.
- Reactome — curated molecular pathways.

- MSigDB — hallmark and curated gene sets.

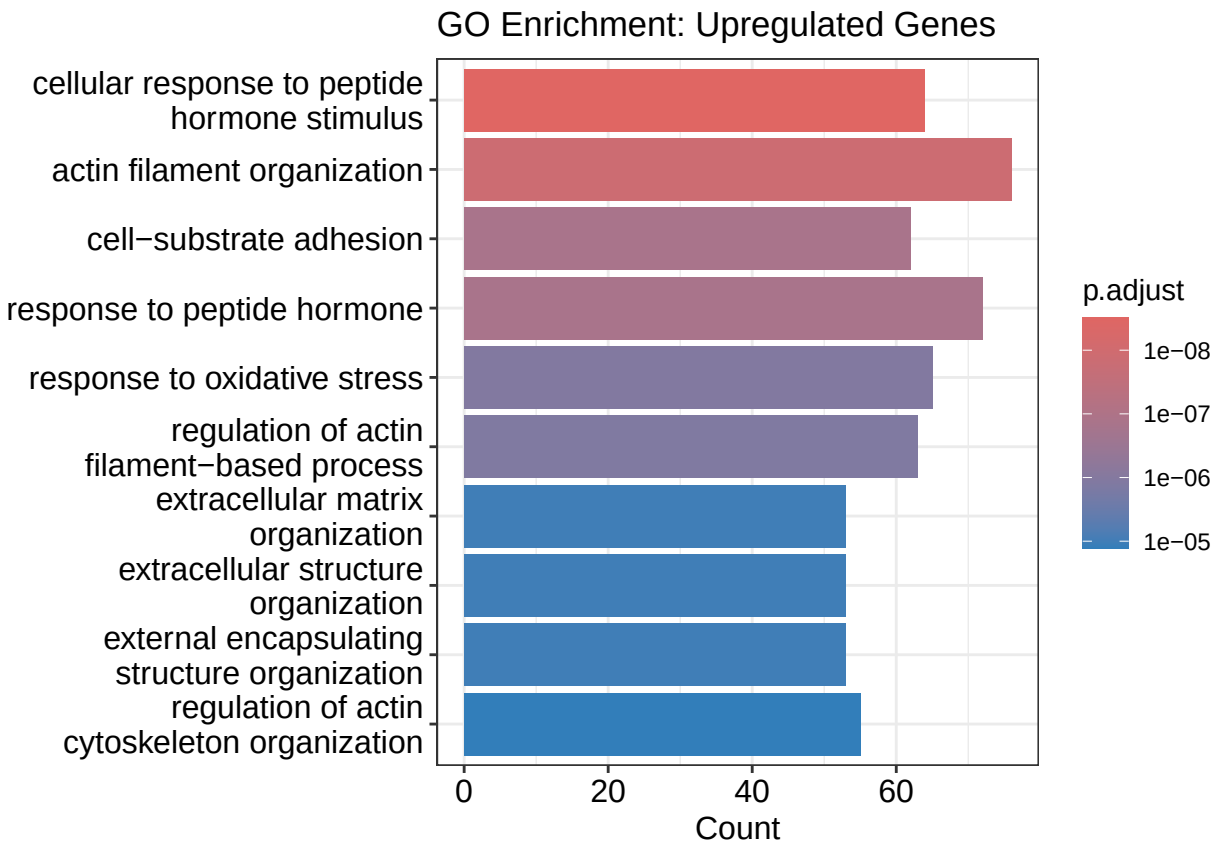
Gene Ontology Enrichment Analysis

In this example, we will perform GO enrichment using the list of up-regulated and down-regulated genes separately. However the same analysis can be done also with the list of DE genes.

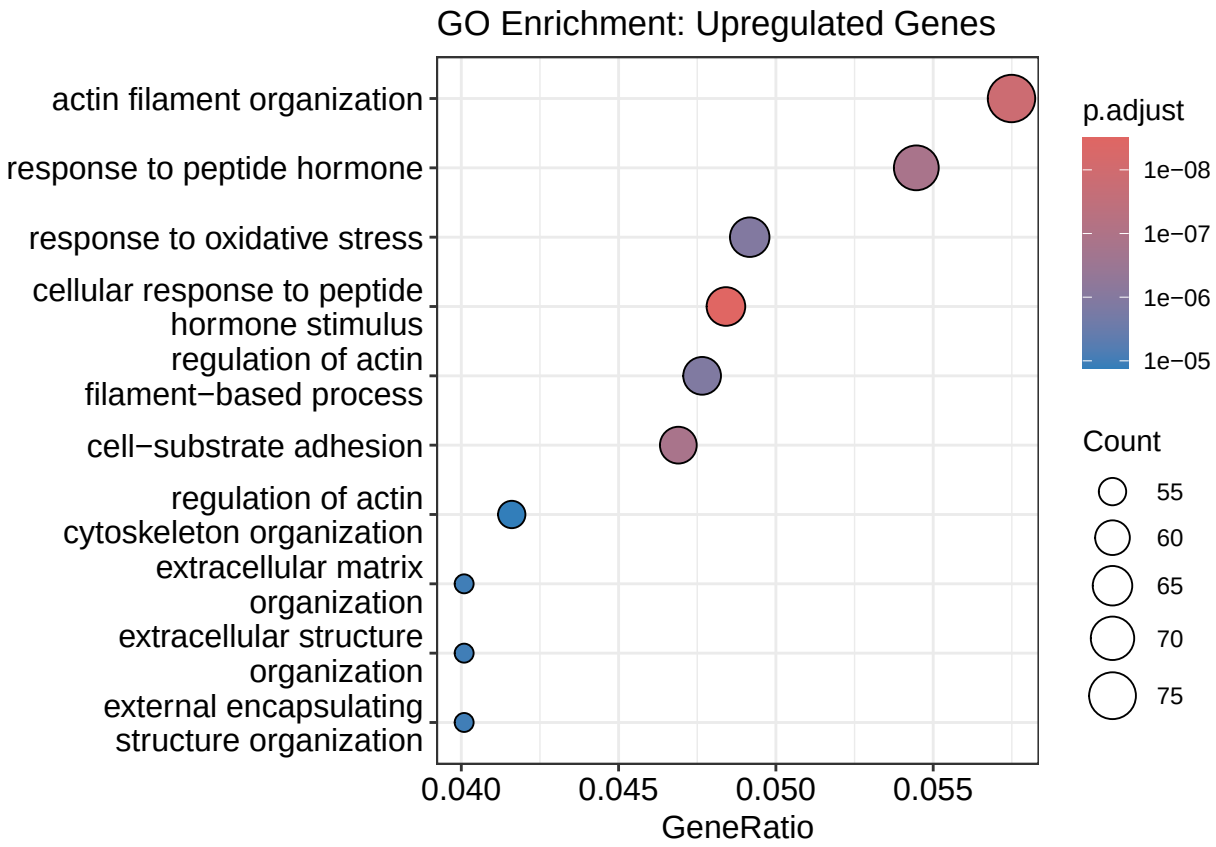
```
# Get significant upregulated genes
sig_up_genes <- rownames(res_df[res_df$log2FoldChange > 0 & res_df$padj < 0.05, ])
sig_up_entrez <- na.omit(res_df$entrez[rownames(res_df) %in% sig_up_genes]) ##remove those with NA in t

  # Get significant downregulated genes
sig_down_genes <- rownames(res_df[res_df$log2FoldChange < 0 & res_df$padj < 0.05, ])
sig_down_entrez <- na.omit(res_df$entrez[rownames(res_df) %in% sig_down_genes]) ##remove those with NA
```

Note in the GO, there are different types of Ontologies. You must choose the one you are interested in. Here, we used the Biological Process as an example.

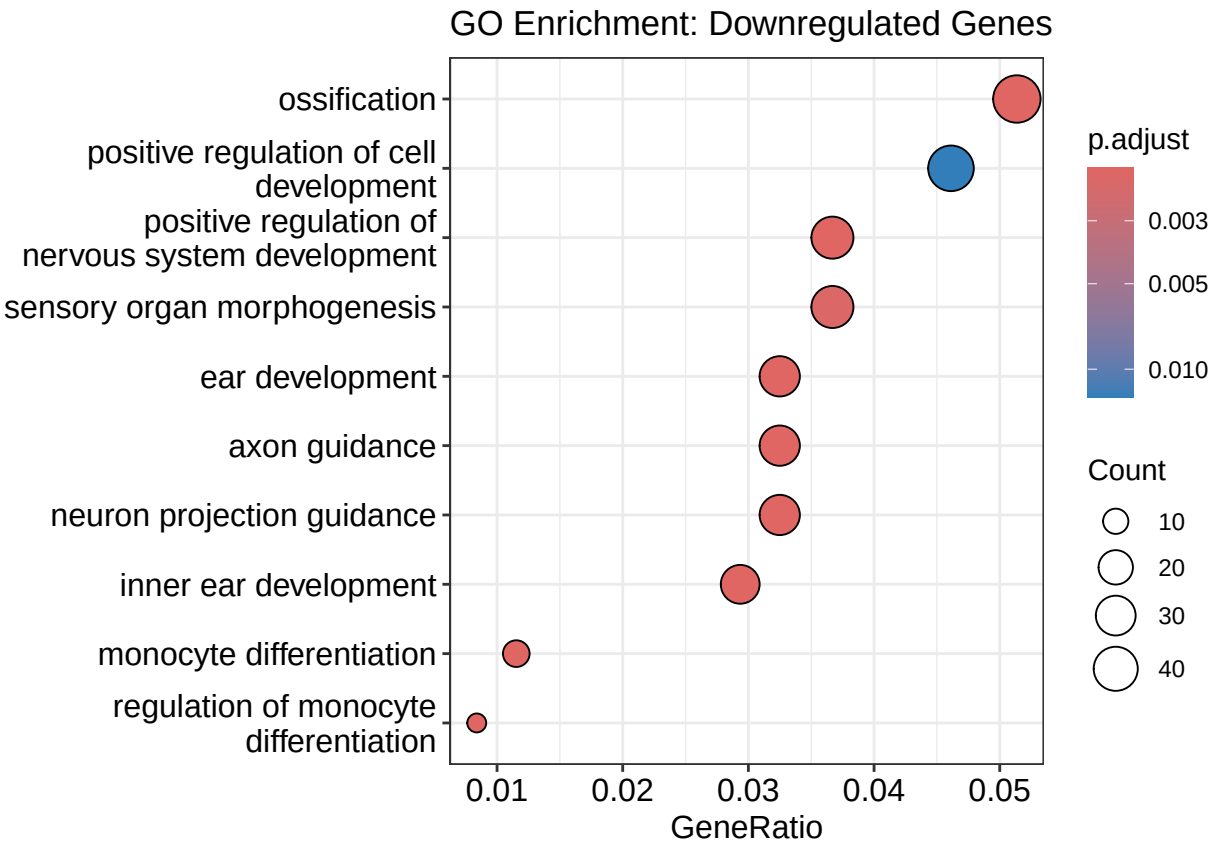
```
ego_up <- enrichGO(gene = sig_up_entrez,
                   OrgDb = org.Hs.eg.db,
                   keyType = "ENTREZID",
                   ont = "BP",  # Biological Process
                   pAdjustMethod = "BH",
                   pvalueCutoff = 0.05,
                   qvalueCutoff = 0.2)

barplot(ego_up, showCategory = 10, title = "GO Enrichment: Upregulated Genes")
```



```r
dotplot(ego_up, showCategory = 10, title = "GO Enrichment: Upregulated Genes")
```



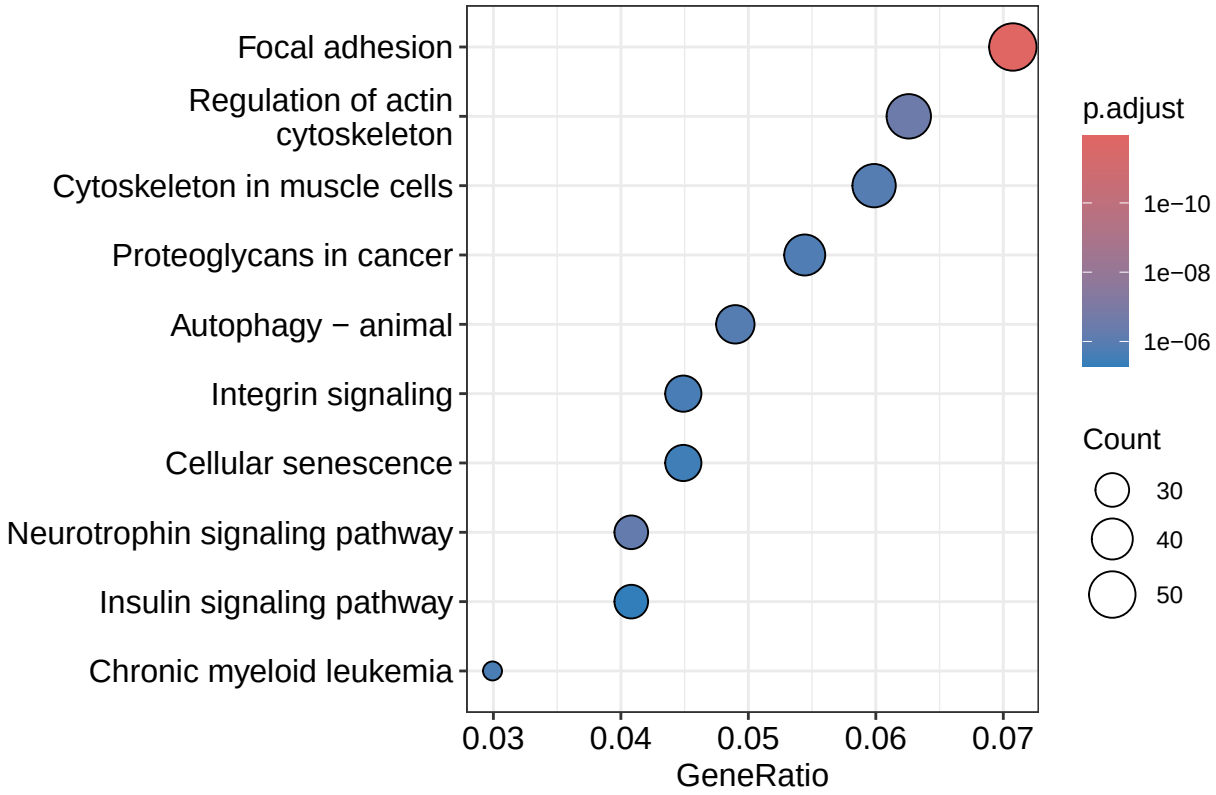

```r
ego_down <- enrichGO(gene = sig_down_entrez,
                     OrgDb = org.Hs.eg.db,
                     keyType = "ENTREZID",
                     ont = "BP", # Biological Process
                     pAdjustMethod = "BH",
                     pvalueCutoff = 0.05,
                     qvalueCutoff = 0.2)

dotplot(ego_down, showCategory = 10, title = "GO Enrichment: Downregulated Genes")
```



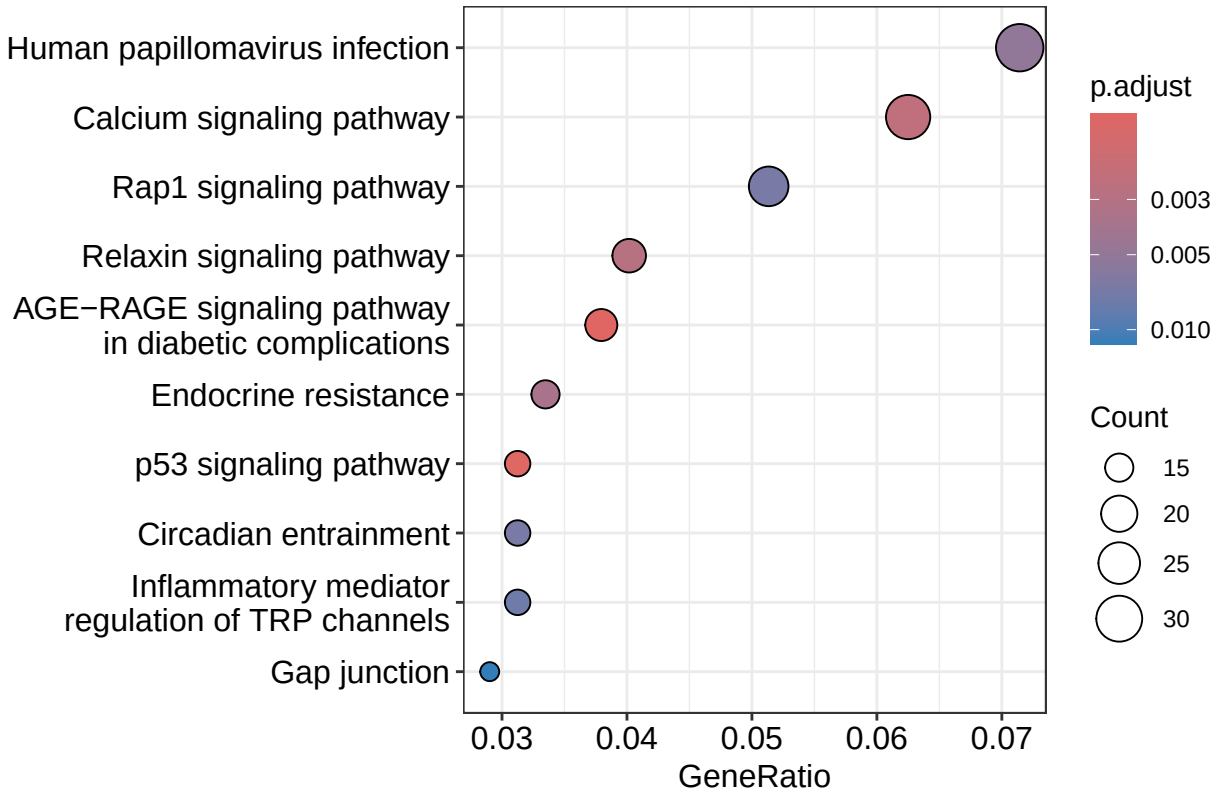
Kegg Pathway Analysis

In this example, we will perform KEGG enrichment using the list of up-regulated and down-regulated genes separately.

```
# need ENTREZ gene ids and sometimes organism code e.g. "hsa"
ekegg_up <- enrichKEGG(gene = sig_up_entrez, organism = 'hsa', pvalueCutoff=0.05)
dotplot(ekegg_up, showCategory=10)
```




```
ekegg_dw <- enrichKEGG(gene = sig_down_entrez, organism = 'hsa', pvalueCutoff=0.05)
dotplot(ekegg_dw, showCategory=10)
```



GSEA (optional)

To perform GSEA with DESeq2 results, you need to generate a **ranked list of genes** based on the DESeq2 output, typically using the log2 fold change, the stats, the sign(log2FoldChange) * -log10(pvalue), or other metric, and then use this list as input for GSEA software or packages like clusterProfiler.

You also need to create a list of pathways to test. The MSigDB Hallmark gene sets can be retrieved and prepared using the *msigdbr()* function.

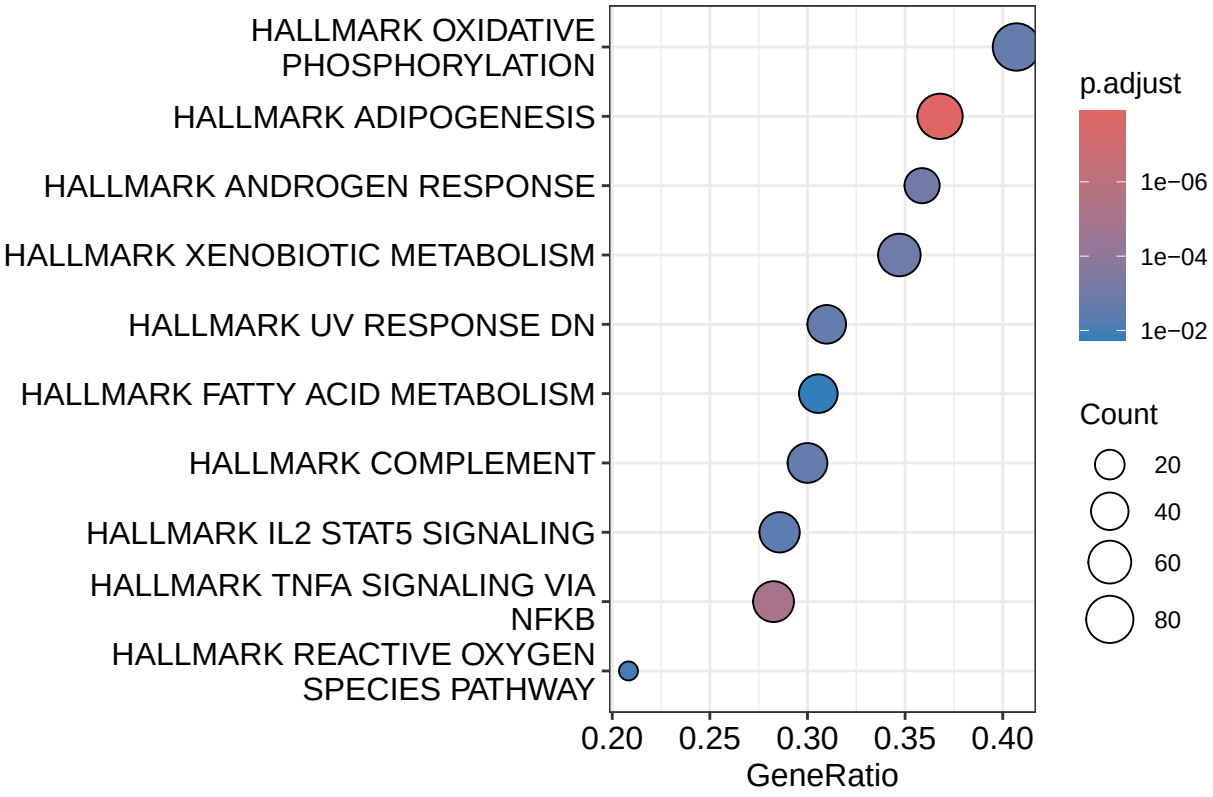
```
res_clean <- res_df %>%
  filter(!is.na(symbol), !is.na(stat))

# Keep only rows with symbol and stat
res_unique <- res_clean %>%
  group_by(symbol) %>%
  slice_max(order_by = abs(stat), n = 1, with_ties = FALSE) %>%
  ungroup()

#Build the ranked list for GSEA
ranked <- res_unique %>% arrange(desc(stat))
ranks <- ranked$stat
names(ranks) <- ranked$symbol
```

Build the list of pathways to test

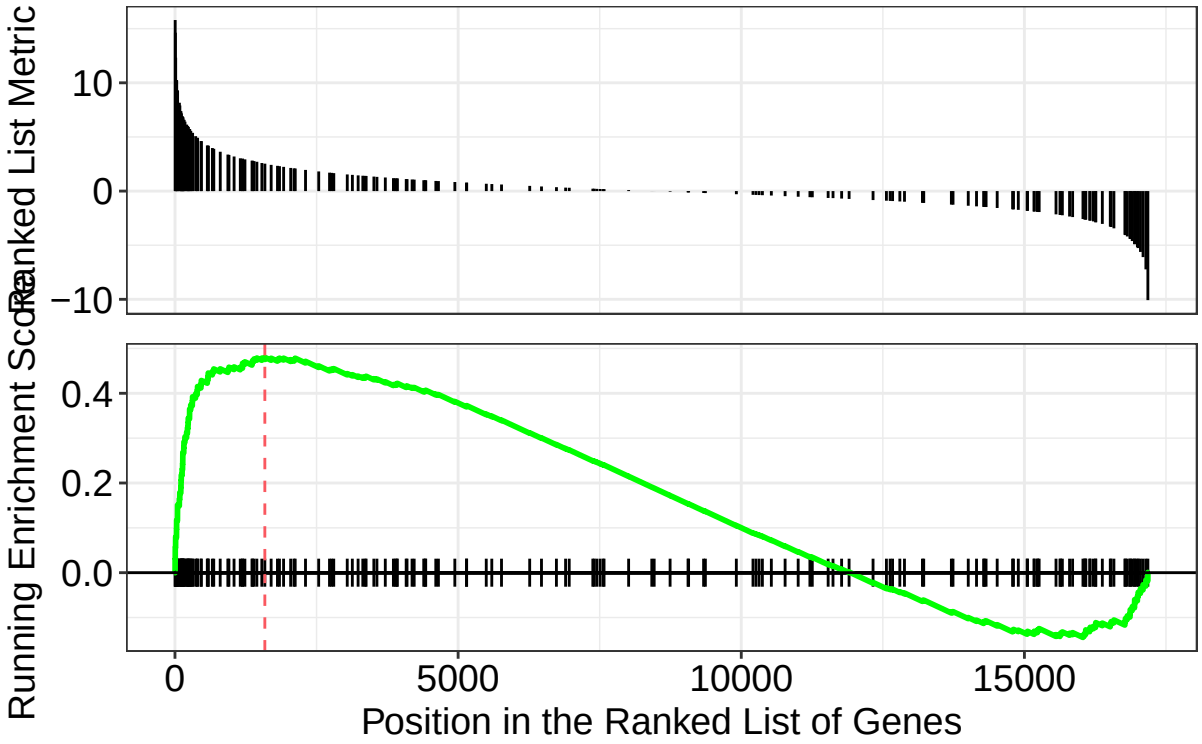
```
##  Load MSigDB Hallmark gene sets via msigdbr
msig <- msigdbr(species = "Homo sapiens", collection = "H") %>%
  select(gs_name, gene_symbol)

term2gene <- msig %>% select(gs_name, gene_symbol)
```

run GSEA using clusterProfiler::GSEA

```
gsea_cp <- GSEA(
  geneList = ranks,
  TERM2GENE = term2gene,
  pvalueCutoff = 1,
  verbose = FALSE
)

dotplot(gsea_cp, showCategory = 10)
```

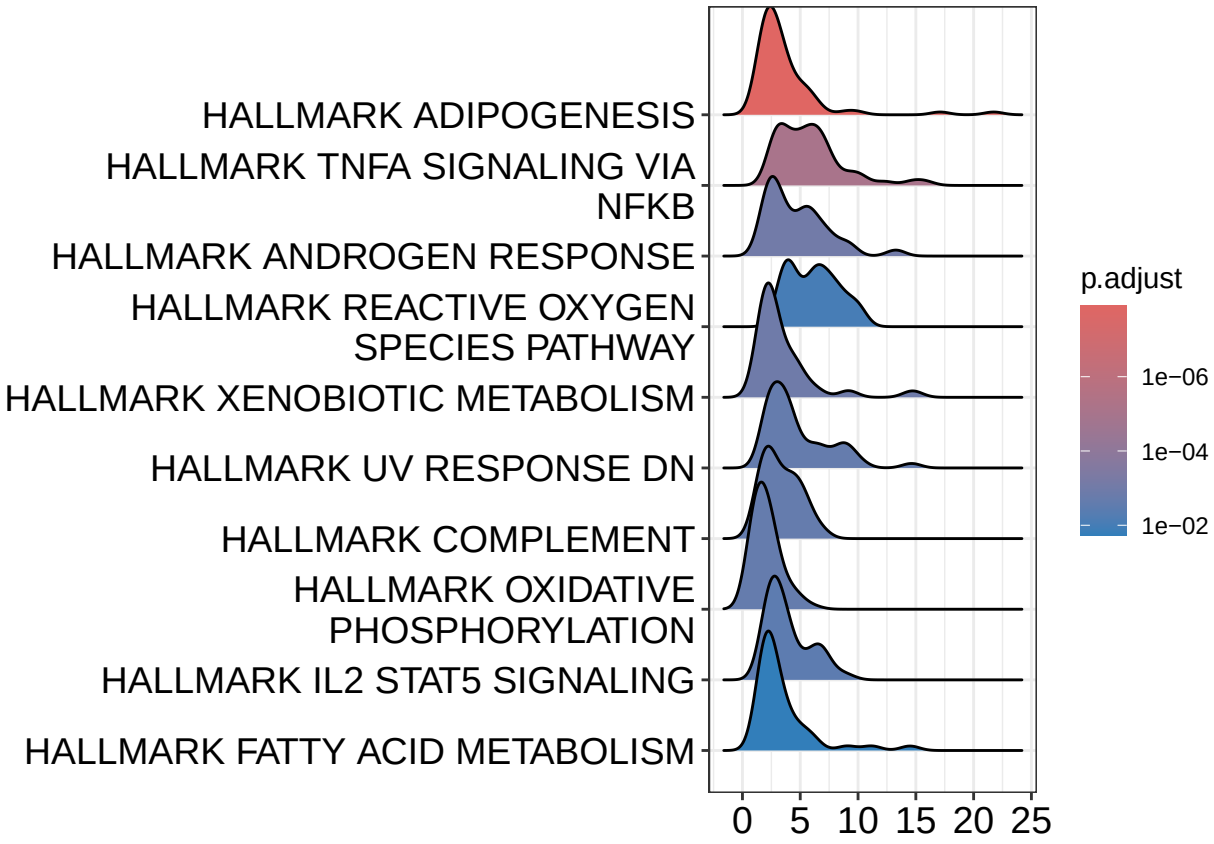



```
gseaplot(gsea_cp, geneSetID = "HALLMARK_TNFA_SIGNALING_VIA_NFKB")
```




```
ridgeplot(gsea_cp, showCategory = 10)
```

```
## Picking joint bandwidth of 0.815
```



QUESTION TIME

- Do you have any questions or comments?
- Report your experience.

EXERCISES

1. Try different choices for the pathway analysis by providing the list of DE genes.
2. Repeat the above analysis, including the cell type in the formula. Hint: use *dds <- DESeqDataSet(airway, design = ~ cell+dex)*
3. Try to find some raw data from a paper of your interest with a GEO accession number. Try importing the data into R and running a basic analysis.

Acknowledgments

- This tutorial presented at BBCC 2025 (https://www.bbcc-meetings.it/bbcc2025/) is part of the dissemination activities supported by the P2022BLN38 project *Computational approaches for the integration of multi-omics data* – funded by European Union – Next Generation EU within the PRIN 2022 PNRR program (D.D. 1409 del 14/09/2022 Ministero dell'Università e della Ricerca) CUP B53D23027810001.


```
sessionInfo()
```

```
## R version 4.5.2 (2025-10-31)
## Platform: aarch64-apple-darwin20
## Running under: macOS Tahoe 26.1
##
```

```
## Matrix products: default
## BLAS:   /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Versi
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/Rome
## tzcode source: internal
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] ashr_2.2-63                 apeglm_1.32.0
##  [3] msigdbr_25.1.1              clusterProfiler_4.18.2
##  [5] org.Hs.eg.db_3.22.0         AnnotationDbi_1.72.0
##  [7] DESeq2_1.50.2               RColorBrewer_1.1-3
##  [9] pheatmap_1.0.13             dplyr_1.1.4
## [11] ggplot2_4.0.1               airway_1.30.0
## [13] SummarizedExperiment_1.40.0 Biobase_2.70.0
## [15] GenomicRanges_1.62.0        Seqinfo_1.0.0
## [17] IRanges_2.44.0              S4Vectors_0.48.0
## [19] BiocGenerics_0.56.0         generics_0.1.4
## [21] MatrixGenerics_1.22.0       matrixStats_1.5.0
##
## loaded via a namespace (and not attached):
##   [1] rstudioapi_0.17.1       jsonlite_2.0.0          magrittr_2.0.4
##   [4] ggtangle_0.0.9          farver_2.1.2            rmarkdown_2.30
##   [7] fs_1.6.6                vctrs_0.6.5             memoise_2.0.1
##  [10] ggtree_4.0.1            SQUAREM_2021.1          tinytex_0.58
##  [13] mixsqp_0.3-54           htmltools_0.5.8.1       S4Arrays_1.10.0
##  [16] curl_7.0.0              truncnorm_1.0-9         SparseArray_1.10.2
##  [19] gridGraphics_0.5-1      htmlwidgets_1.6.4       plyr_1.8.9
##  [22] cachem_1.1.0            igraph_2.2.1            lifecycle_1.0.4
##  [25] pkgconfig_2.0.3         Matrix_1.7-4            R6_2.6.1
##  [28] fastmap_1.2.0           gson_0.1.0              digest_0.6.39
##  [31] numDeriv_2016.8-1.1     aplot_0.2.9             enrichplot_1.30.3
##  [34] patchwork_1.3.2         irlba_2.3.5.1           RSQLite_2.4.5
##  [37] labeling_0.4.3          invgamma_1.2            httr_1.4.7
##  [40] abind_1.4-8             compiler_4.5.2          bit64_4.6.0-1
##  [43] fontquiver_0.2.1        withr_3.0.2             S7_0.2.1
##  [46] BiocParallel_1.44.0     DBI_1.2.3               R.utils_2.13.0
##  [49] MASS_7.3-65             rappdirs_0.3.3          DelayedArray_0.36.0
##  [52] tools_4.5.2             ape_5.8-1               R.oo_1.27.1
##  [55] glue_1.8.0              nlme_3.1-168            GOSemSim_2.36.0
##  [58] grid_4.5.2              reshape2_1.4.5          fgsea_1.36.0
##  [61] gtable_0.3.6            R.methodsS3_1.8.2       tidyr_1.3.1
##  [64] data.table_1.17.8       XVector_0.50.0          ggrepel_0.9.6
##  [67] pillar_1.11.1           stringr_1.6.0           yulab.utils_0.2.2
##  [70] babelgene_22.9          emdbook_1.3.14          splines_4.5.2
##  [73] treeio_1.34.0           lattice_0.22-7          bit_4.6.0
##  [76] tidyselect_1.2.1        fontLiberation_0.1.0    GO.db_3.22.0
```

```
##  [79] locfit_1.5-9.12         Biostrings_2.78.0       knitr_1.50
##  [82] fontBitstreamVera_0.1.1 xfun_0.54               stringi_1.8.7
##  [85] lazyeval_0.2.2          ggfun_0.2.0             yaml_2.3.11
##  [88] evaluate_1.0.5          codetools_0.2-20        bbmle_1.0.25.1
##  [91] gdtools_0.4.4           tibble_3.3.0            qvalue_2.42.0
##  [94] ggplotify_0.1.3         cli_3.6.5               systemfonts_1.3.1
##  [97] Rcpp_1.1.0              coda_0.19-4.1           png_0.1-8
## [100] bdsmatrix_1.3-7         parallel_4.5.2          assertthat_0.2.1
## [103] blob_1.2.4              DOSE_4.4.0              mvtnorm_1.3-3
## [106] tidytree_0.4.6          ggiraph_0.9.2           ggridges_0.5.7
## [109] scales_1.4.0            purrr_1.2.0             crayon_1.5.3
## [112] rlang_1.1.6             cowplot_1.2.0           fastmatch_1.1-6
## [115] KEGGREST_1.50.0
```