Introduction to RNA-seq for Differential Expression Analysis – 4

by Jiajia Li

Biological Data Science Institute

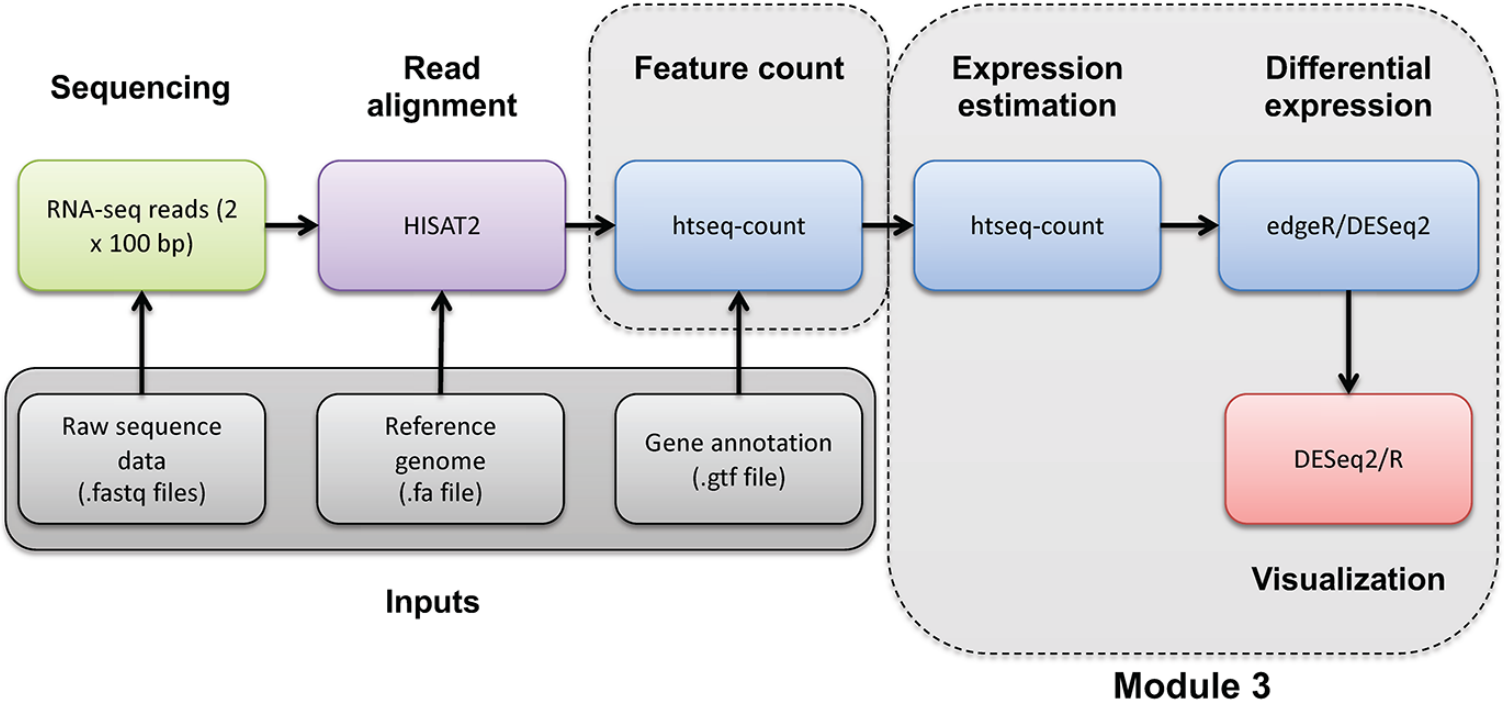
13 May 2025

Australian National University

Learning Objectives of Today

- Differential Expression with edgeR
- Compare with Ballgown result using Venn Diagram
- PCA Plot with Ballgown data
- Volcano Plot with edgeR data

Differential Expression with edgeR



Differential Expression with edgeR

Walter+Eliza Hall Bioinformatics
Institute of Medical Research

edgeR: Empirical analysis of digital gene expression data in R

edgeR was developed by WEHI in Melbourne. So, it's **Australian-made**!!

Differential expression analysis of sequence **count data**!!

It implements a range of **statistical methodology** based on the negative binomial distributions, including empirical Bayes estimation, exact tests, generalised linear models, quasi-likelihood, and gene set enrichment.

Differential Expression with edgeR

Using count data to find differentially expressed gene is an alternative way to StringTie/Ballgown pipeline.

First, let's create a new folder to store the result:
`mkdir -p ~/RNAseq-Workshop/de/edger`

Also create a new R script "edgeR_Analysis.R" under "RNAseq-Workshop".

Note that the htseq-count results provide counts for each gene but uses only the Ensembl Gene ID (e.g. ENSG00000054611). This is not very convenient for biological interpretation.

Previously we have created a mapping file called "ENSG_ID2Name.txt", this will help us translate ENSD IDs to Symbols.

Differential Expression with edgeR

Let's copy the "ENSG_ID2Name" file to "de/edger", previously it was stored in "expression/htseq-counts" folder.

Then, open Rstudio.

First, let's read in the mapping file and our count data.

```
# read in mapping file as a dataframe
mapping <- read.table("de/edger/ENSG_ID2Name.txt", header = FALSE)
# read in count data
rawdata <- read.table("expression/htseq_counts/gene_read_counts_table_all_final.tsv",
                      header = TRUE, row.names = 1)
```

Filter out low count gene

Our count table looks like:

	UHR_Rep1	UHR_Rep2	UHR_Rep3	HBR_Rep1	HBR_Rep2
ENSG00000008735	13	17	8	397	531
ENSG00000015475	103	64	100	40	42
ENSG00000025708	46	16	26	13	11
ENSG00000025770	188	116	154	73	78
ENSG00000040608	18	8	11	69	91
ENSG00000054611	70	47	66	36	43
ENSG00000056487	12	13	7	11	10

Check dimension of the table:

```
> dim(rawdata)
[1] 1410    6
```

So, we have 1410 genes.

Filter out low count gene

We want genes that **at least expressed in one sample**, and **count >= 10**.

Let's write some R code to do this.

```
# filtering low count gene
filtered_data <- rawdata[apply(rawdata, 1, function(x) any(x >= 10)), ]
```

Type your code by hand, so at least you get familiar with brackets () and square brackets [] for extracting certain things from a data frame.

`apply(array/matrix/dataframe, row-wise/column-wise, function to apply)`

Now we filter down to 449 genes.

```
> dim(filtered_data)
[1] 449   6
```

Filter out low count gene

Import edgeR:
`library(edgeR)`

First, let's create the class names.

```
# make class labels
class <- c(rep("UHR", 3), rep("HBR", 3))
```

```
> class
[1] "UHR" "UHR" "UHR" "HBR" "HBR" "HBR"
```

Then, let's get the symbol names for the 449 genes.

```
# get common gene names
gene_ids <- rownames(filtered_data)
gene_annotations <- mapping[mapping[,1] %in% gene_ids, ]
```

Filter out low count gene

The gene_annotation data frame should look like:

	V1	V2
1	ENSG00000008735	MAPK8IP2
2	ENSG00000015475	BID
3	ENSG00000025708	TYMP
4	ENSG00000025770	NCAPH2
5	ENSG00000040608	RTN4R
6	ENSG00000054611	TBC1D22A

```
gene_annotations                    449 obs. of 2 variables
  $ V1: chr  "ENSG00000008735" "ENSG00000015475" "ENSG0000
  $ V2: chr  "MAPK8IP2" "BID" "TYMP" "NCAPH2" ...
```

Create a DGEList object

```
# create a DGEList
y <- DGEList(counts = filtered_data, genes = gene_annotations, group = class)
```

A DGEList (Differential Gene Expression List) is a **special object** in the edgeR package designed to store **RNA-seq count data** and **associated information** needed for analysis.

It's like a container that keeps:

- your raw counts
- your sample information
- your gene annotations
- normalisation factors
- library sizes

Create a DGEList object

A DGEList object is essentially a list of objects, you can access these objects using $, such as:

```
ENSG00000054611 ENSG00
ENS  samples   {y}
ENS  genes     {y}
ENS  counts
ENS
> y$
```

```
> y$samples
         group lib.size norm.factors
UHR_Rep1   UHR   182653           1
UHR_Rep2   UHR   120619           1
UHR_Rep3   UHR   149797           1
HBR_Rep1   HBR    93757           1
HBR_Rep2   HBR   115203           1
HBR_Rep3   HBR   102508           1
```

`lib.size` is the total count for each sample.

```
> y$genes
                     gene_ids          symbols
ENSG00000008735 ENSG00000008735         MAPK8IP2
ENSG00000015475 ENSG00000015475              BID
ENSG00000025708 ENSG00000025708             TYMP
ENSG00000025770 ENSG00000025770           NCAPH2
```

TMM Normalisation

```
# TMM Normalisation
y <- calcNormFactors(y)
```

Because the sequence depth for each sample if different, e.g., `lib.size` is different.

The `calcNormFactors()` function **computes normalisation factors** to account for **composition bias** between samples.

It adjusts for the fact that some samples may have more counts just because:

- They were sequenced deeper (higher library size), or
- They had more highly expressed genes that skew the count distribution

Then we can compare gene expression across samples fairly!

TMM Normalisation

```
# TMM Normalisation
y <- calcNormFactors(y)
```

By default, the function uses the **TMM (Trimmed Mean of M-values)** method.

This updates the `y` object by adding normalisation factors.

```
> y$samples
         group lib.size norm.factors
UHR_Rep1   UHR   182653    1.0429273
UHR_Rep2   UHR   120619    1.0289273
UHR_Rep3   UHR   149797    0.9840335
HBR_Rep1   HBR    93757    0.9854177
HBR_Rep2   HBR   115203    0.9844642
HBR_Rep3   HBR   102508    0.9761827
```

Estimate Dispersion

In RNA-seq data, dispersion accounts for biological variability between replicates. It's a measure of how much a gene's expression varies across samples beyond what you'd expect by chance.

EdgeR models this using the **negative binomial distribution**, where the dispersion controls the spread of counts. You can't do differential expression reliably without it.

```
# estimate dispersion
design <- model.matrix(~ class)
y <- estimateDisp(y, design, robust = TRUE)
```

The `design <- model.matrix(~ class)` creates a design matrix that tells edgeR how your samples are grouped. Before we have created a vector called `class`.

```
> class
[1] "UHR" "UHR" "UHR" "HBR" "HBR" "HBR"
```

Estimate Dispersion

It estimates 3 dispersions:

- **Common dispersion**: one value for all genes (rough overall variability)
- **Trended dispersion**: smooth curve of dispersion vs. expression level
- **Tagwise dispersion**: a unique dispersion value for each gene

These help edgeR decide how much trust to place in the count differences between groups.

You can access them by:
`y$common.dispersion`
`y$trended.dispersion`
`y$tagwise.dispersion`

Interpreting common.dispersion

In edgeR, `common.dispersion` is the **average biological variability across all genes**, assuming each gene shares the same level of variability between replicates. It's a single value used in the negative binomial model to estimate variance.

We have 6 samples:

- Group1: UHR_Rep1, UHR_Rep2, UHR_Rep3
- Group2: HBR_Rep1, HBR_Rep2, HBR_Rep3

```
> y$common.dispersion
[1] 0.002395129
```

Each group has **3 biological replicates**.

Our common.dispersion is 0.002395129.

Then, the BCV (Biological Coefficient of Variation) = square root of common.dispersion =

```
> sqrt(y$common.dispersion)
[1] 0.04894006
```

Interpreting common.dispersion

```
> sqrt(y$common.dispersion)
[1] 0.04894006
```

Meaning, on average, gene expression varies ~4.89% between replicates due to biological noise. Which is very low… compare to the normal range.

Because! our sample is not actually biological replicates… 😂 they are technical replicates because we are using commercial cells.

📊 Typical `common.dispersion` Ranges

Type of Replicates	Common Dispersion	BCV (√dispersion)	Notes
Technical replicates	0.001 – 0.01	3% – 10%	Very low variability; sequencing noise only
Homogeneous cell lines	0.01 – 0.04	10% – 20%	Clean, well-controlled lab experiments
Biological replicates	0.04 – 0.1	20% – 30%	Real tissue or organism-level variation
Heterogeneous samples	0.1 – 0.4+	30% – 60%+	e.g. human patient samples, tumors, complex tissues, batch effects

Differential Expression Test - exactTest()

```
# differential expression test
et <- exactTest(y)
```

The `exactTest()` function is used to perform differential expression analysis between **two groups** of samples using an **exact test** analogous to Fisher's exact test but adapted for **over-dispersed count data** modelled by the negative binomial distribution.

When to use exactTest()?

- When comparing two groups only (e.g., control vs treated)
- When using a simpler design (i.e., no covariates, no blocking factors)
- It's a quicker and simpler alternative to the `glmFit()` + `glmLRT()` pipeline, which is more flexible for complex designs.

Top expressed genes

```r
# print top genes
topTags(et)
```

```
> topTags(et)
Comparison of groups:  UHR-HBR
                     gene_ids    symbols     logFC   logCPM        PValue           FDR
ENSG00000211677 ENSG00000211677      IGLC2 12.596302 11.70735  0.000000e+00  0.000000e+00
ENSG00000100321 ENSG00000100321     SYNGR1 -4.818765 11.97929  0.000000e+00  0.000000e+00
ENSG00000100167 ENSG00000100167      SEPT3 -4.675601 11.94871  0.000000e+00  0.000000e+00
ERCC-00130           ERCC-00130 ERCC-00130  2.050612 16.82133  0.000000e+00  0.000000e+00
ENSG00000008735 ENSG00000008735   MAPK8IP2 -5.770890 11.00691 6.623006e-317 5.947459e-315
ENSG00000225783 ENSG00000225783       MIAT -4.097371 11.59226 1.325894e-307 9.922106e-306
ERCC-00004           ERCC-00004 ERCC-00004  2.375640 13.91644 1.019912e-297 6.542008e-296
ENSG00000100095 ENSG00000100095      SEZ6L -5.416941 10.96208 4.167025e-297 2.338743e-295
ENSG00000185686 ENSG00000185686      PRAME 11.992739 11.13350 2.051426e-278 1.023434e-276
ENSG00000128245 ENSG00000128245      YWHAH -2.656493 12.62772 5.882462e-252 2.641226e-250
```

Top expressed genes

logFC: Log2 fold change between two groups

- Positive - gene is upregulated ↑ in the second group
- Negative - gene is upregulated ↑ in the first group

logCPM: Average log2 counts per million

- Gives a sense of how abundantly expressed the gene is
- Higher logCPM = more reads = higher expression

PValue: raw p-value

- < 0.05, statistically significant

FDR (False Discovery Rate): adjusted p-value (Benjamini-Hochberg)

- < 0.05, statistically significant

Summary up/down significant genes at FDR=0.05

```
# get up/down regulated genes at FDR=0.05
de <- decideTests(et, adjust.method = "BH", p = 0.05)
summary(de)
```

```
> summary(de)
       UHR-HBR
Down       168
NotSig     118
Up         163
```

We have 168 genes that are downregulated in HBR, and 163 upregulated in HBR.

Get the **exact test table** for all genes at any p-value:

```
# get the exact test table for all genes
out <- topTags(et, n = "Inf", adjust.method = "BH", sort.by = "none", p.value = 1)$table
```

Also, add the raw count to it:

```
# extract raw counts of our gene
counts <- getCounts(y)
# add raw counts back to the DE test table
out2 <- cbind(out, counts)
```

out2 table

gene_ids	symbols	logFC	logCPM	PValue	FDR	UHR_Rep1
ENSG00000008735	MAPK8IP2	-5.77088987	11.006911	6.623006e-317	5.947459e-315	13
ENSG00000015475	BID	0.44200907	9.051675	1.078758e-02	1.562459e-02	103
ENSG00000025708	TYMP	0.42722203	7.536501	1.965261e-01	2.353072e-01	46
ENSG00000025770	NCAPH2	0.44744739	9.806050	7.851007e-04	1.268022e-03	188
ENSG00000040608	RTN4R	-3.28375160	8.589611	6.716005e-51	4.568919e-50	18
ENSG00000054611	TBC1D22A	0.02717722	8.676087	9.140345e-01	9.285101e-01	70

Then, we filter out those are not significant DE.

```
# limit to significant DE genes
out3 <- out2[as.logical(de), ]
```

out2	449 obs. of 12 variables
out3	331 obs. of 12 variables

We have 331 genes left.

Sort by Pvalue

Sort the out3 table by Pvalue:

```
# sort the out table by PValue
sorted_out <- out3[order(out3$PValue, decreasing = TRUE), ]
```

Finally, we can save the result:

```
# save the table
write.table(sorted_out, file = "de/edger/DE_genes.txt", quote = FALSE,
            row.names = FALSE, sep = "\t")
```

Compare with Ballgown result

Previously we had our Ballgown result saved in the "de/ballgown/DE_genes.txt", we can compare it with the edgeR result by plotting a Venn-Diagram.

Create a new R script called "Venn_Diagram.R".

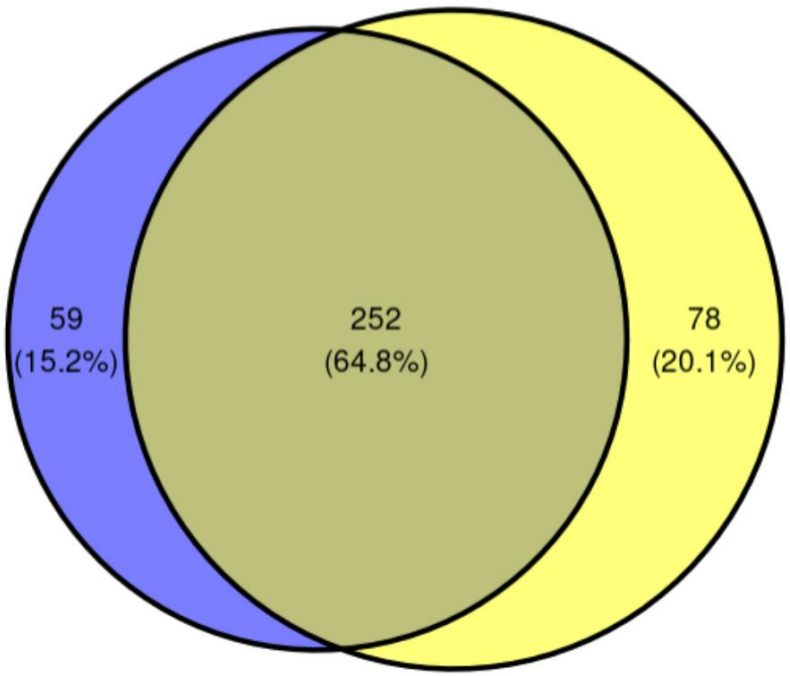
Then:
`install.packages("ggvenn")`
`library(ggvenn)`

Exercise: read in both the ballgown and edgeR DE result into R.

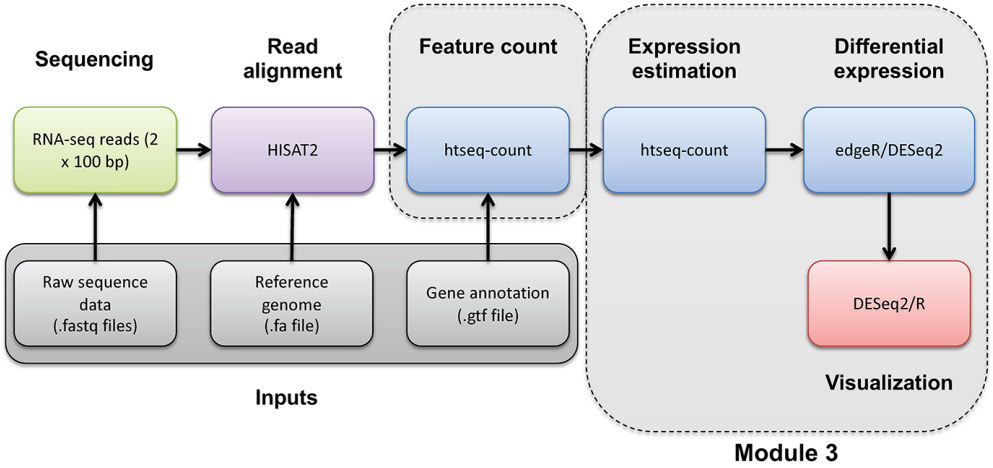
Use the below code to generate a Venn Diagram.

```
# create a venn diagram
my_sets <- list(Ballgown = ballgown_DE$V1, edgeR = edgeR_DE$symbols)
ggvenn(my_sets, auto_scale = TRUE)
```

Compare with Ballgown result



DE with DESeq2



We suppose to cover the htseq-count/DESeq2 pipeline.

Due to time limit we will skip this. For anyone who is interested in: https://rnabio.org/module-03-expression/0003/03/03/Differential_Expression-DESeq2/

PCA plot with Ballgown data

- Create a new script "Ballgown_Visualisation.R"
- Load the libraries

```
#load libraries
library(ballgown)
library(genefilter)
library(dplyr)
library(devtools)
```

- Load the previously saved ballgown data

```
# Load the ballgown object from file
load("de/ballgown/bg.rda")
```

- print the summary of the ballgown object

```
> bg
ballgown instance with 4564 transcripts and 6 samples
>
```

Visualisation with Ballgown

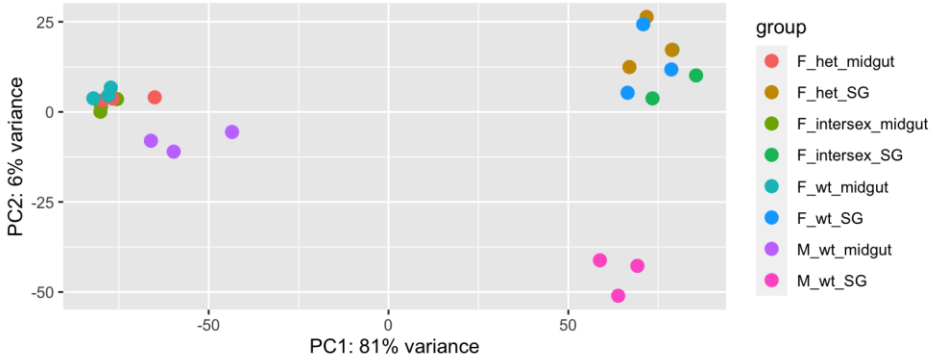
```
# load ballgown object to table
bg_table <- texpr(bg, meas = "all")
```

texpr() ???

meas = "all" ???

PCA Plot (Principal Component Analysis)

- Purpose: to see sample clustering and variability
- Use: detect batch effects or outliers
- Axes: principal components (PC1 vs. PC2)



Batch Effects in RNA-seq Data

Batch effects in RNA-seq data refer to **unwanted variation** that arises from **differences in experimental conditions** not related to the biological variables of interest.

These can significantly distort the results of differential expression analysis or other downstream analysis.

Common sources of batch effects include:

- Different sequencing runs or machines
- Library preparation dates
- Personnel or lab conditions
- Reagent lots or kits
- RNA extraction dates

PCA Plot

To create a PCA plot, we need a **normalised expression matrix**, where:

- Rows = Genes
- Column = Samples
- Values = normalised expression values, ideally log-transformed.

Preferred matrix for PCA:

- log2(TPM + 1)
- log2(FPKM + 1)
- VST or rlog, from DESeq2
- log2-CPM, from edgeR

Never use raw counts for PCA - they are highly skewed and violate assumptions of PCA.

PCA Plot with FPKM - gene level

From the ballgown object, we can extract the expression matrix for FPKM, but it is not log-transformed, let's transform it first.

First, we need to extract the matrix from the ballgown object:

```
# extract gene-level expression, FPKM
gene_exp_matrix <- gexpr(bg)
```

	FPKM.UHR_Rep1	FPKM.UHR_Rep2
ENSG00000008735	9.733280	19.257822
ENSG00000015475	101.648194	92.512731
ENSG00000025708	65.150466	35.587802
ENSG00000025770	302.197952	253.872392

Then, we need to change the gene IDs to symbols for easier interpretation.

PCA Plot with FPKM - gene level

```
# make gene symbols the row names
gene_mapping <- unique(bg_table[, 9:10])
id_to_symbol <- setNames(gene_mapping$gene_name, gene_mapping$gene_id)
new_rownames <- id_to_symbol[rownames(gene_exp_matrix)]
# check if any IDs not mapped
sum(is.na(new_rownames))
# substitute row names
rownames(gene_exp_matrix) <- new_rownames
```

Then, our gene expression table looks like:

	FPKM.UHR_Rep1	FPKM.UHR_Rep2	FPKM.UHR_Rep3
MAPK8IP2	9.733280	19.257822	6.9885366
BID	101.648194	92.512731	118.9674979
TYMP	65.150466	35.587802	39.0739550
NCAPH2	302.197952	253.872392	272.7187762

Then we can log-transform it.

PCA Plot with FPKM - gene level

```
# make gene symbols the row names
gene_mapping <- unique(bg_table[, 9:10])
id_to_symbol <- setNames(gene_mapping$gene_name, gene_mapping$gene_id)
new_rownames <- id_to_symbol[rownames(gene_exp_matrix)]
# check if any IDs not mapped
sum(is.na(new_rownames))
# substitute row names
rownames(gene_exp_matrix) <- new_rownames
```

Then, our gene expression table looks like:

	FPKM.UHR_Rep1	FPKM.UHR_Rep2	FPKM.UHR_Rep3
MAPK8IP2	9.733280	19.257822	6.9885366
BID	101.648194	92.512731	118.9674979
TYMP	65.150466	35.587802	39.0739550
NCAPH2	302.197952	253.872392	272.7187762

Then we can log-transform it:

```
# log-transform
log_matrix <- log2(gene_exp_matrix + 1)
```

PCA Plot with FPKM - gene level

Then, we have to transpose our matrix because PCA expects **samples to be rows**.

```
# transpose for PCA
t_expr_matrix <- t(log_matrix)
```

	MAPK8IP2	BID	TYMP	NCAPH2	RTN4R
FPKM.UHR_Rep1	3.424019	6.681564	6.047679	8.244116	4.624956
FPKM.UHR_Rep2	4.340407	6.547091	5.193291	7.993631	4.297397
FPKM.UHR_Rep3	2.997931	6.906500	5.324593	8.096551	4.158044
FPKM.HBR_Rep1	9.146930	6.301641	5.085186	7.757949	7.540291
FPKM.HBR_Rep2	9.272775	6.048294	4.833630	7.505566	7.559774
FPKM.HBR_Rep3	9.266826	6.444587	5.589118	7.574249	7.591458

Then, we can finally run PCA…

```
# run PCA
pca <- prcomp(t_expr_matrix, scale. = TRUE)
```

and receive an error!!! 🤯

```
> pca <- prcomp(t_expr_matrix, scale. = TRUE)
Error in prcomp.default(t_expr_matrix, scale. = TRUE) :
  cannot rescale a constant/zero column to unit variance
```

PCA Plot with FPKM - gene level

```
> pca <- prcomp(t_expr_matrix, scale. = TRUE)
Error in prcomp.default(t_expr_matrix, scale. = TRUE) :
  cannot rescale a constant/zero column to unit variance
```

This error means at lease one column (gene) in our matrix has **zero variance** - it's **constant across all samples**. PCA can't rescale a constant column because it has no variability to analyse.

So, we can remove these genes.

```
# remove zero variance genes
nonzero_v_g <- apply(t_expr_matrix, 2, var) != 0
t_matrix_filtered <- t_expr_matrix[, nonzero_v_g]
```

Then, the number of genes get down to 884.

```
> dim(t_matrix_filtered)
[1]   6 884
>
```

PCA Plot with FPKM - gene level

Now, let's run PCA again!! ✊

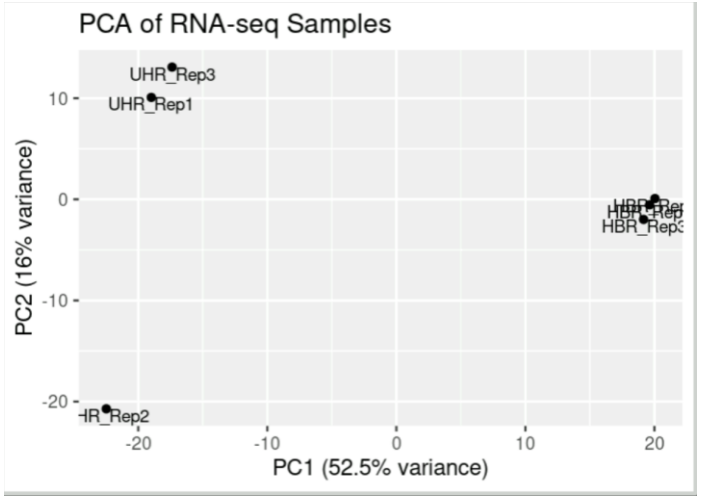
```
# run PCA again
pca <- prcomp(t_matrix_filtered, scale. = TRUE)
```

No error, good. We can proceed to plotting.

```
# PCA plot
pca_df <- as.data.frame(pca$x)
pca_df$sample <- c("UHR_Rep1", "UHR_Rep2", "UHR_Rep3",
                   "HBR_Rep1", "HBR_Rep2", "HBR_Rep3")

ggplot(pca_df, aes(x = PC1, y = PC2)) +
  geom_point() +
  geom_text(aes(label = sample), vjust = 1, size = 3) +
  labs(title = "PCA of RNA-seq Samples",
       x = paste0("PC1 (", round(summary(pca)$importance[2,1] * 100, 1), "% variance)"),
       y = paste0("PC2 (", round(summary(pca)$importance[2,2] * 100, 1), "% variance)"))
```

PCA Plot with FPKM - gene level



From the plot, we can see that HBR samples are clustered together and UHR_Rep2 is further away from the other two UHR samples.

This indicates **UHR_Rep2 is likely an outlier** - different from its biological replicates.

PCA Plot with FPKM - gene level

Possible reasons:

- Technical issues
  - RNA degradation
  - Low library complexity
  - Mapping/alignment errors
  - Batch effects
- Data processing problems
  - Normalisation artifacts
  - Mislabelling
- Biological heterogeneity (less likely if it's a technical replicate)

There are also ways you can check which genes drive the difference.

Volcano Plot

A volcano plot is a type of scatter plot that displays the results of differential gene expression (DGE) analysis. It combines **effect size** (fold change) with **statistical significance** (p-values), allowing you to see both at once.

X Axis: log2(Fold Change)

- Measures how much a gene's expression changes between two groups.
- Positive = upregulated in second group
- Negative = downregulated

Y Axis: -log10(p-value/adjusted p-value)

- Represents significance of the change.
- Higher = more statistically significant

Volcano Plot

To create a volcano plot, we need to perform **Differential Expression test** first to generate the **Fold Change** values and **p-values**.

Previously we have done this with both Ballgown and edgeR.

Unfortunately, the `ballgown::stattest()` function doesn't generate FC values at **log scale**, so we can't create volcano plot from it.

We will use the edgeR DE result as the example.

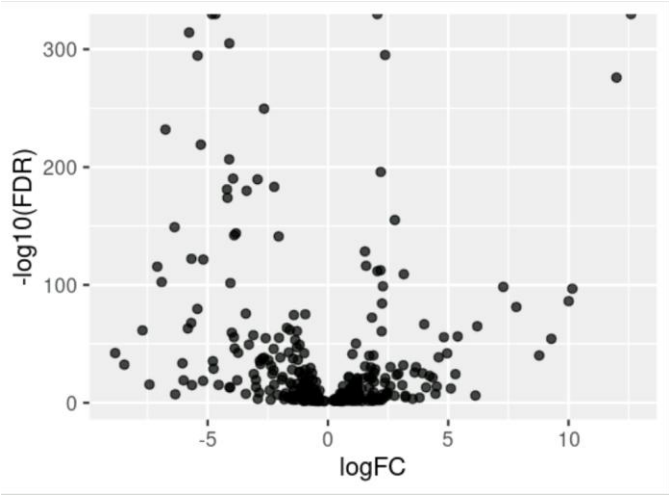
Create a new file called "volcano_plot.R".

Volcano Plot

Read in the previously saved "DE_gene.txt" from edgeR.

Then create a scatter plot, x-axis = logFC, y-axis = -log10(FDR).

```
ggplot(de_table, aes(x = logFC, y = -log10(FDR))) +
  geom_point(alpha = 0.7)
```



This is a most basic one. We can also add colours for significant up/down regulated genes and lines to separate them.

Volcano Plot

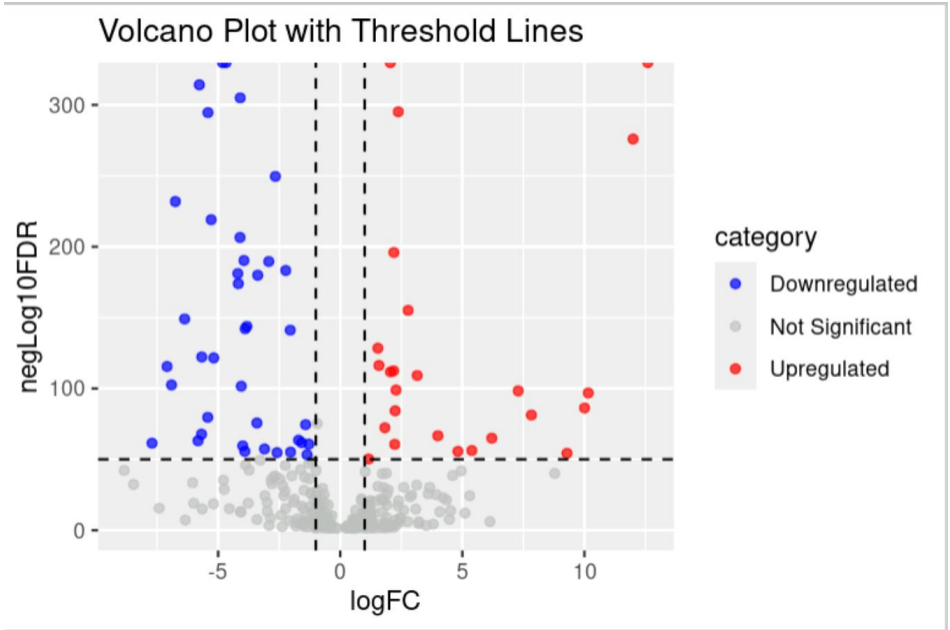
Add a column "category" and decide if this gene is upregulated or downregulated or not significant.

```
de_table <- de_table |>
  mutate(
    negLog10FDR = -log10(FDR),
    category = case_when(
      logFC > 1 & negLog10FDR > 50 ~ "Upregulated",
      logFC < -1 & negLog10FDR > 50 ~ "Downregulated",
      TRUE ~ "Not Significant"
    )
  )
```

Then we can colour the dots with the category label.

```
ggplot(de_table, aes(x = logFC, y = negLog10FDR)) +
  geom_point(aes(colour = category), alpha = 0.7) +
  scale_color_manual(values = c(
    "Upregulated" = "red",
    "Downregulated" = "blue",
    "Not Significant" = "grey"
  )) +
  geom_vline(xintercept = c(-1, 1), linetype = "dashed", color = "black") +
  geom_hline(yintercept = 50, linetype = "dashed", color = "black") +
  labs(title = "Volcano Plot with Threshold Lines")
```

Volcano Plot



Then we can also label top significant genes.

Volcano Plot

Select top significant genes (up and down).

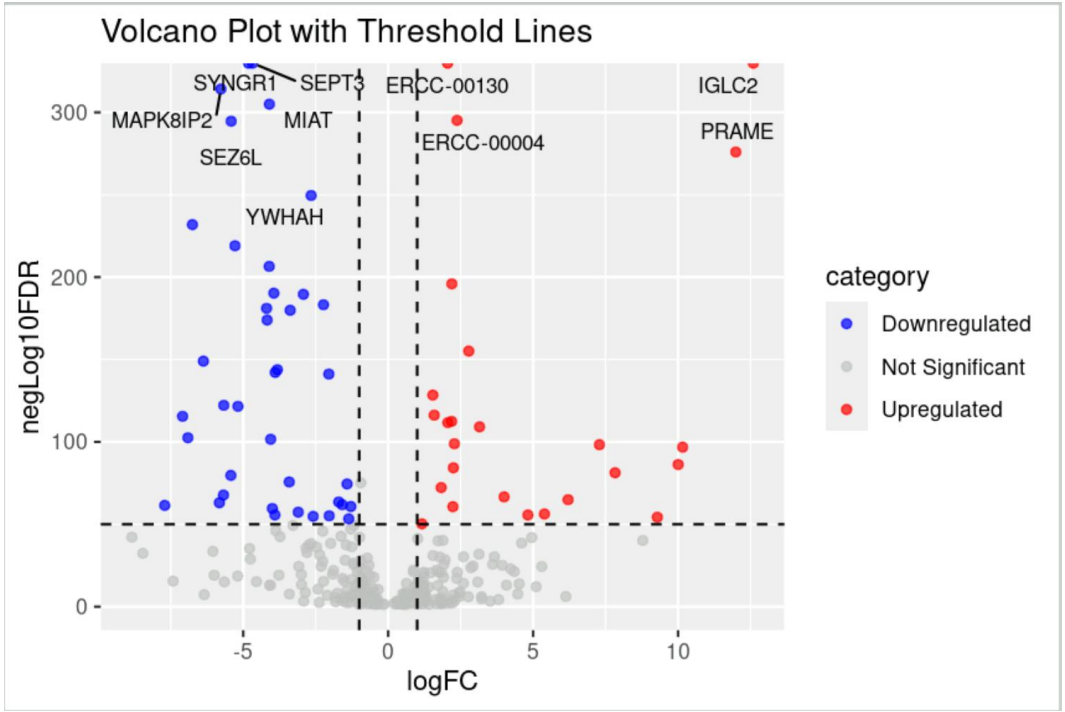
```
top_genes <- de_table %>%
  filter(category != "Not Significant") %>%
  arrange(desc(negLog10FDR)) %>%
  slice_head(n = 10)
```

And we need to install another package:

```
install.packages("ggrepel")
library(ggrepel)
```

```
ggplot(de_table, aes(x = logFC, y = negLog10FDR)) +
  geom_point(aes(colour = category), alpha = 0.7) +
  scale_color_manual(values = c(
    "Upregulated" = "red",
    "Downregulated" = "blue",
    "Not Significant" = "grey"
  )) +
  geom_vline(xintercept = c(-1, 1), linetype = "dashed", color = "black") +
  geom_hline(yintercept = 50, linetype = "dashed", color = "black") +
  geom_text_repel(data = top_genes,
                  aes(label = symbols),
                  size = 3,
                  max.overlaps = 10,
                  box.padding = 0.3,
                  point.padding = 0.2) +
  labs(title = "Volcano Plot with Threshold Lines")
```

Volcano Plot



Thank you

Contact us

Jiajia Li
Biological Data Science Institute

RN Robertson Building, 46 Sullivan's Creek Rd
The Australian National University
Canberra ACT 2600

E jiajia.li1@anu.edu.au
W https://bdsi.anu.edu.au/

Australian National University