RNASeq Data Analysis Workshop

Heshmat Borhani

2025-01-27

Deferentially Expression Analysis

Import the data for the analysis

Import the libraries you need:

First, we need to install and load the required libraries:

```
# Install required packages (if not already installed)
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("DESeq2")

if (!require("dplyr")) install.packages("dplyr")
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("ggrepel")) install.packages("ggrepel")
```

Now, load the libraries:

```
# Load libraries
library(DESeq2)
library(dplyr)
library(ggplot2)
library(ggrepel)
```

Set Working Directory

Define a variable `PATH` to direct R to your working directory. For example, if there's a folder named "RNASeq Workshop" on your desktop:

```
PATH <- "c:/Users/mbzhab/Desktop/RNASeq Workshop"
setwd (PATH)
```

Note: Adjust the `PATH` based on your directory structure. Windows uses backslashes, while Linux/Mac uses forward slashes.

Import Data Files

Import your "Raw-counts.tsv" and "Experiment-design.tsv" files into data frames using `read.delim`:

```
# Read the 'Raw-counts.tsv' file
counts <- read.delim(paste(PATH, "/DATA/Raw-counts.tsv", sep=""))

# Read the 'Experiment-design.tsv' file
design <- read.delim(paste(PATH, "/DATA/Experiment-design.tsv", sep=""))
```

Inspect the data

If the data were imported correctly, we can see how many samples you have in the experiment (`design`) and how many genes you have in the expression matrix (`counts`). Check the dimensions of the imported data frames:

```
# Check the dimension of the dataframes
dim(design)

dim(counts)
```

To verify the "Raw-counts.tsv" files was imported correctly, we can use the indexing approach `[ ]` to display the first 5 rows and columns:

```
#Index specific rows and columns in a dataframe
counts[1:5, 1:5]
```

Alternatively, we use the head command to inspect the first rows:

```
# Inspect the first rows of the data
head(design)
```

From the head output, you can see the columns of the Experiment-design file. The "Experiment-design.tsv" file contains the extra information which are needed for **DESeq analysis**. Here's a description of the columns:

Column	Description
Run	ID for the samples
Tissue	Sample tissue ("Normal" or "Primary Tumour")
Disease	Disease information
Patient	Patient ID
Organism_Part	Sample origin
Type	Sample type ("Normal" or "Cancer")
Therapy	Therapy info ("YES" or "NOT")
Age	Patient age

We can get the values stored in your dataframe in different ways. A useful command is `table` that gives you an overview of the values and occurrences in a specific column:

```r
# Check the values in a coulumn
table(design$Tissue)
```

We can also combine more columns:

```r
# Check the values in various columns
table(design$Tissue, design$Therapy)
```

Preparing input files

Now that all the data is confirmed, validated and examined, the data analysis part can be started.

metadata Firstly, we need to create the metadata dataframe:

```r
# Create the metadata dataframe
metadata <- data.frame(Sample_ID = design$Run, Type = as.factor(metadata$Type))
```

With this command, we created a new dataframe called metadata, which has the Sample IDs along with the Type of the sample.

We can examine the content of `metadata` dataframe, just running:

```r
head(metadata)
```

Then, we need to set Sample_ID as row names and remove the column:

```r
# Chang row names of the metadata with Sample_IDs
row.names (metadata) <- metadata$Sample_ID

head (metadata)
```

Now, we need to remove the Sample ID column from the `metadata` dataframe.

```r
# Remove the Sample ID from the dataframe
metadata <- metadata [, -1]

head (metadata)
```

counts Let's again have a look at the `counts` dataframe:

```r
head(counts)
```

For doing Differential Expression Analysis using DESeq2 packages, we need to ensure columns represent samples, and row names are gene IDs in counts dataframe:

```r
# Put the counts in a suitable format for the DESeq2 packages
row.names(counts) <- counts$ENS_ID

# Remove the ENS_ID from the datafram
counts <- counts %>% select(-ENS_ID)
```

We should verify that samples in metadata and counts dataframe match, using the following command identical:

```
# Check if the Sample IDs in two dataframe are the saame
identical (colnames(counts), rownames(metadata))
```

You can see the difference between the `metadata` and `counts` dataframes, using the following commands:

```
# Check if there are any differences between two variables
setdiff (colnames (counts), rownames(metadata))

# Check if there are every single Sample IDs are included in both variables
intersect (colnames (counts), rownames(metadata))
```

For DESeq2 packages, the samples must be presented in both dataframes in the same order. So we should present them in the same order:

```
# Check if the Sample IDs in both dataframe is in the same order
all (colnames (counts) == rownames (metadata))

# Change the order of the Sample IDs in an ascending format
metadata <- metadata [order(rownames(metadata)), ]
counts <- counts [, order(colnames(counts))]

all (colnames (counts) == rownames (metadata))
```

Perform Differential Expression Analysis

Once, we have the `metadata` and the `counts` dataframes, we can perform the differential expression analysis using the DESeq2 package. We need to create a `DESeqDataSet` object:

```
# Perform DESeq analysis
dds <- DESeqDataSetFromMatrix (counts, metadata, ~ Type)

dds <- DESeq (dds)
```

The `DESeq` code is very simple, and it takes only TWO steps:

1. Create a `DESeqDataSet` object (`dds`) containing **expression data**, **metadata** and the **variable of interest** to consider for the comparison in the differential expression analysis
2. Perform normalisation of the expression values and the differential expression analysis, using the `DESeq()` function on the `DESeqDataSet` object.

A this point, we need to extract DESeq results with Benjamini-Hochberg correction:

```
# Create DESeq results object using Benjamini-Hochberg correction
res <- results (object = dds, contrast = c('Type','cancer','normal'),
pAdjustMethod = 'BH', alpha = 0.05)

head(res)
```

In this command, we have used the Benjamini-Hochberg correction (BH) and a threshold of 0.05 (alpha) to extract the genes that are differentially expressed between cancer and normal tissues. **Note:** Specify the contrast explicitly (e.g., contrast = c('Type', 'cancer', 'normal')) to define comparisons and interpret log2 fold changes. We can see a summary of the results using the following command:

```
# Get summary of the result of DESeq
summary (res)
```

Out of 40166 with nonzero total read count, we identified **3814** UP-regulated genes (`logFoldChange (LFC) > 0`) and **2943** DOWN-regulated genes (`logFoldChange (LFC) < 0`). The first conclusion is that there are much more UP-regulated genes of interest as potential causal factors for this type of cancer. Obviously, the DOWN-regulated are **NOT** something to ignore either.

Note: Multiple testing correction is crucial in high-throughput gene analysis. When we perform tests on over 40,000 genes, it means that the analysis will include over 40,000 hypothesis tests. Consequently, there is a high risk of **false positive results** (a high probability of finding significant results by chance) if a multiple correction procedure is not implemented. To avoid this, DESeq (and all the tools used for Differential Expression Analysis) always uses a multiple correction procedure.

Variance Stabilizing Transformation (VST)

VST is used in RNA-Seq data to reduce the dependence of variance on the mean. Raw count data often show higher variance for genes with higher expression levels, making it difficult to compare genes. VST applies a transformation, such as a log, to stabilise the variance across the range of expression levels. This makes the data more suitable for downstream analyses like visualisation, as it helps highlight biological differences rather than technical noise.Let's apply VST to stabilise variance:

```
# Performe Varience Stabilizing Transformatio
vst <- vst (dds)
```

Visualisation

Principle-Component Analysis (PCA)

A good practice and a useful first step in an RNA-Seq analysis is to assess overall similarity between samples. To visualise sample-to-sample similarities, we can use a **Principal-Components Analysis (PCA)**. In this method, the data points (the samples) are projected onto the 2D space such that they spread out optimally. Now, we visualise sample similarities using PCA:

```
# Define the plots content
pcaData <- plotPCA(vst, intgroup = "Type", returnData = TRUE)
percentVar <- round(100 * attr(pcaData, "percentVar"))

# Draw the Priciple Component Analysis
ggplot(pcaData, aes(PC1, PC2, color = Type, label = rownames(pcaData))) +
  geom_point(size = 4, shape = 19) +
  geom_text_repel(size = 3, max.overlaps = 10, show.legend = FALSE) +
  xlab(paste0("PC1: ", percentVar[1], "% variance")) +
  ylab(paste0("PC2: ", percentVar[2], "% variance")) +
  theme_classic(base_size = 14) +
  ggtitle("PCA Plot of Samples")+
  theme(
```

```
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8),
    plot.title = element_text(hjust = 0.5),
    panel.border = element_rect(color = "black", fill = NA, size = 0.8))
```

Extract deferentially expressed genes The last step is to create an appropriate output, containing the Symbol IDs, and the genes ranked by their adjusted-p value.

```
# Import annotation with Symbol names, gene position, and biotype
rel_ENS_to_SYM <- read.delim (paste (PATH, "/DATA/rel_ENS_SYM_chr.tsv", sep=""))

# Create a new dataframe containing the results
output <- as.data.frame (res)
head(output)

# Merge the results with the annotation dataframe
output <- merge(output, rel_ENS_to_SYM, by.x="row.names", by.y= "ENS_ID")
head(output)

# Sort the output on padj
output <- output [order (output$padj), ]
```

We will obtain a new dataframe output that looks like this:

```
head(output)
```

The `output` dataframe has been generated using the data from the DESeq2 results (`res`), which extracts a results table containing Log2 fold changes (log2FoldChange), **p values** (**pvalue**) and **adjusted p values** (**padj**). With no additional arguments to the results function, the Log2 fold change will refer to the last variable in the design formula, and if this is a factor, the comparison will be the last level of this variable over the reference level. However, the order of the variables of the design does not matter so long as the user specifies the comparison to build a results table for, using the name or contrast arguments of results. The other columns imported from the DESeq2 results are:

Column	Description
lfcSE	The standard error estimate for the log2 fold change
baseMean	The average of the normalized count values, dividing by size factors, taken over all samples
stat	The value of the test statistic for the gene or transcript

From the rel_ENS_to_SYM dataframe, we extract the Symbol name of the gene (SYMBOL), the biotype (Gene_type), and the gene location (Gene_start, Gene_end, Strand).

Since, in general, we are interested in protein-coding genes, it could be useful to extract only the protein-coding genes from the output. In the following commands, we also extract the top 10 deferentially expressed protein-coding genes and all the statistically significant deferentially expressed protein-coding genes. We put this data in two new dataframes, called top_deregulated and `signigficantDEGs.coding`, respectively.

```
# Select protein coding genes and put them in output.coding
output.coding <- subset(output, Gene_type == "protein_coding")
```

```r
# Identify significant Differentially Expressed Genes
signigficantDEGs <- subset (output.coding, padj < 0.05)
```

We can create a dataframe for the UP-regulated and one for the DOWN-regulated genes and then see their dimension and the top genes. To create the dataframes, you can use the commands:

```r
# Collect up-regulated and down-regulated genes' name
UP_regulated <- subset (signigficantDEGs, log2FoldChange > 0)

DOWN_regulated <- subset (signigficantDEGs, log2FoldChange < 0)
```

Volcano plot

Visualise the deferentially expressed genes using a Volcano plot. The last step is to create a nice visualisation of the results. A Volcano plot visualises the results of RNA-Seq or omics experiments by plotting statistical significance (p value) against a magnitude of change (fold change). This Scatter plot format allows for fast identification of biologically significant genes with large fold changes and statistical significance.

```r
# Anotate genes in Volcano Plot
output.coding$DiffExpressed <- "NO"
output.coding$DiffExpressed [output.coding$log2FoldChange > 0 &
  output.coding$padj < 0.0001] <- "UP"
output.coding$DiffExpressed [output.coding$log2FoldChange < 0 &
  output.coding$padj < 0.0001] <- "DOWN"

# Create a new column DElabel, which contains the name of DEGs
output.coding$DElabel <- NA
output.coding$DElabel [output.coding$SYMBOL %in%
  c(head(DOWN_regulated$SYMBOL, 5), head(UP_regulated$SYMBOL, 5))] <-
  output.coding$SYMBOL[output.coding$SYMBOL %in%
  c(head(DOWN_regulated$SYMBOL, 5), head(UP_regulated$SYMBOL, 5))]

# Draw the Volcano plot
ggplot(data = subset(output.coding, !is.na(padj)),
  aes(x = log2FoldChange, y = -log10(padj), color = DiffExpressed, label = DElabel)) +
  geom_point(alpha = 0.7) +
  theme_classic() +
  geom_text_repel(size = 3, max.overlaps = 10, show.legend = FALSE) +
  scale_color_manual(values = c("blue", "black", "red")) +
  labs(title = "Volcano Plot",
    x = "Log2 Fold Change", y = "-Log10 Adjusted p-value", color = "Expression") +
  theme(
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 9),
    plot.title = element_text(hjust = 0.5),
    panel.border = element_rect(color = "black", fill = NA, size = 0.8))
```

In the plot, the most UP-regulated genes are towards the right, the most DOWN-regulated genes are towards the left, and the most statistically significant genes are towards the top.

Density Plot

A density plot of log2 fold changes visualises the distribution of expression changes between two conditions in RNA-Seq data. Each value represents the log2-transformed ratio of expression levels, where values above 0 indicate up-regulation and below 0 indicate down-regulation. The plot helps assess the overall trend of differential expression, identify biases, or confirm symmetry in the data. It is often used after normalisation or filtering to ensure an accurate representation of biological changes.

```
# Draw density plot of log2 fold changes
ggplot(output.coding, aes(x = log2FoldChange, fill = DiffExpressed)) +
  geom_density(alpha = 0.6) +
  scale_fill_manual(values = c("blue", "red", "grey"), name = "Gene Expression") +
  theme_classic() +
  labs(title = "Density Plot of Log2 Fold Changes", x = "Log2 Fold Change", y = "Density", legend= "He"
  theme(
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8),
    plot.title = element_text(hjust = 0.5),
    panel.border = element_rect(color = "black", fill = NA, size = 0.8))
```