INTRODUCTION

This workflow involves processing RNA-seq data to identify differentially expressed genes (DEGs) between treatment and control groups. It utilizes the nf-core/rnaseq pipeline for pre-processing, including trimming and pseudo-alignment with Salmon, followed by statistical analysis with DESeq2 to evaluate gene expression changes and visualize results.

Background

Differential Gene Expression (DGE) is an analysis that aims to identify genes whose expression levels change significantly between two or more conditions or groups by comparing the total amount of gene expression. It looks for changes in the overall transcriptional output of a gene, and researchers use statistical packages such as edgeR12 or DESeq211 to count genes (Soneson et al. 2015). To identify differentially expressed genes (DEGs), we determine if the mean expression level differs between two or more groups (e.g., treatment vs. control) for each gene in the genome, and then we look for statistically significant differences in mean expression (after correcting for multiple tests) and genes that have a nontrivial difference in fold-difference.

References:

https://www.ebi.ac.uk/ena/data/view/PRJNA490376 http://www.rnajournal.org/cgi/doi/10.1261/rna.069773.118 Soneson C, Love MI, and Robinson MD.
Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences [version 1; peer review: 2 approved]. F1000Research 2015, 4 1521 (https://doi.org/10.12688/f1000research.7563.1)

METHODS

The first step is processing the single-end (SE) Illumina RNA-seq data using the nf-core/rnaseq workflow. This workflow allows you to perform various tasks, such as removing adapters with the default TrimGalore!, indexing the human reference genome for Salmon, trimming reads, and executing Salmon. In our example, the reference genome is provided in GTF format, which includes the start and end positions of features (e.g., genes, mRNAs, exons, CDSs, and often other feature types such as long non-coding RNAs) in the reference genome. Below is an example Nextflow script that runs Salmon based on a spreadsheet with SE data, reference genome FASTA, annotation, and a JSON file with parameters, where the data is corrected for GC bias.

Run the nf-core/rnaseq pipeline

```
#Run the nf-core/rnaseq pipeline to trim reads and execute Salmon to
create quant.sf files
#Use the default TrimGalore "trimmer"
#!/bin/bash
#SBATCH --job-name=<job name>
#SBATCH --mail-type=FAIL
```

```
#SBATCH --mail-user=<email>
#SBATCH --nodes=1
#SBATCH --tasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=8G
#SBATCH --time=48:00:00

# Load required modules
#module load nextflow/23.04.1

# Download reference genome and annotation from ENSEMBL
#nextflow run nf-core/rnaseq -r 3.14.0 \
#--input <spreadsheet with single-end data>
#--outdir res \
#--fasta <fa.gz reference genome>
#--gtf <gtf.gz reference genome>
#--extra_salmon_quant_args "--gcBias " \ #an argument to correct for GC
Bias.
#-profile <profile> \
#-params-file <json file with desired parameters> # note: we pass the JSON
or YAML file in to nextflow here. Some examples setting to put in the JSON
file is "skip_trimming": false, "skip_alignment": true, "pseudo_aligner":
"salmon", and"save_reference": true
```

To run this analysis these are the following packages I used.

Packages

```
library(clusterProfiler)
library(tximport)
library(DESeq2)
library(tidyverse)
```

Salmon generates a folder with various subfolders. From this folder, you can download the "Salmon" output, which includes a tx2gene.tsv file that maps transcript identifiers in the ENSEMBL annotation to their parent ENSEMBL genes, along with a folder for each sample. To process the data and perform statistical analysis to identify differentially expressed genes (DEGs) using the DESeq2 package, you first need to generate gene-level counts from Salmon's quant.sf files and the tx2gene.tsv file.

A dataframe must be created containing the columns you are testing to use the dds() command. This command stores count data, intermediate calculations, and results of a differential expression analysis. The structure of your data paths and dds() will vary depending on your data, the analysis you are conducting, and where your files are stored.

In the example below, we generate gene-level counts from Salmon's quant.sf files and tx2gene.tsv, storing the results in the txi variable—a simple list object containing gene counts and other information. Additionally, we have a dataframe containing metadata corresponding to our data. Here’s an example of what the metadata might look like:

```
   patient condition
1 Control1   control
2 Control2   control
3 Control3   control
4 Treated1 treatment
5 Treated2 treatment
6 Treated3 treatment
```

Once your variables are defined, the next step is to create a DESeqDataSet object with gene-level counts derived from Salmon-inferred TPMs per transcript. The "design" argument specifies the statistical design. In the example below, colData = metadata.df and the design is set as design = ~ condition, ensuring gene expression levels between the conditions specified in the condition variable are compared, allowing for differential gene expression testing.

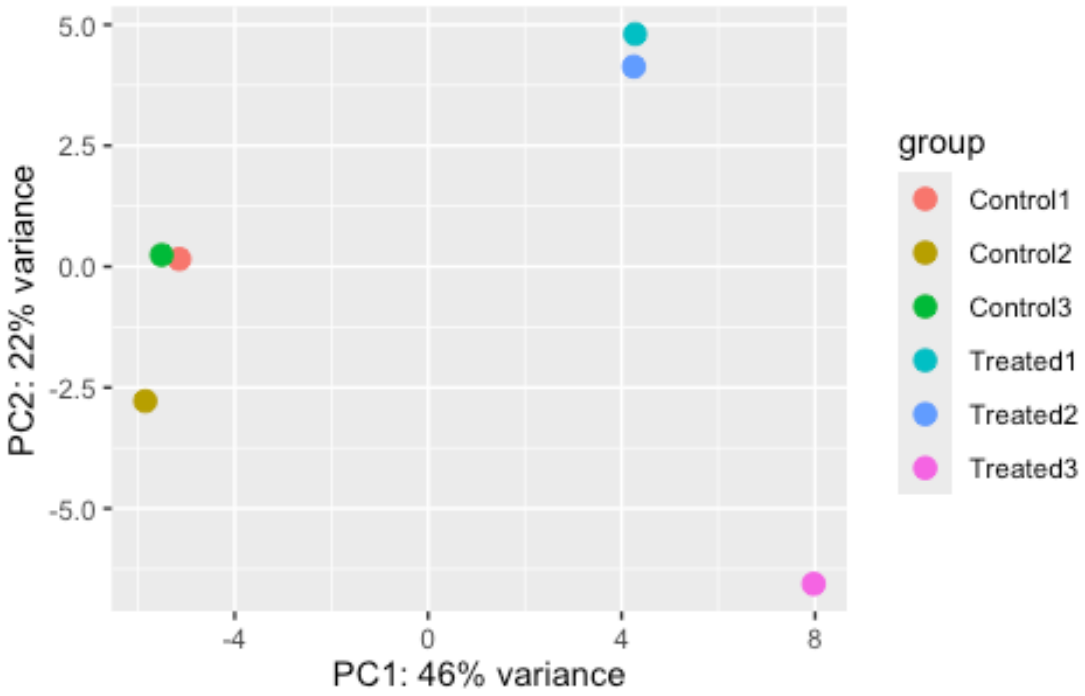
Creating a DESeqDataSet object with gene-level counts derived from Salmon-inferred TPMs per transcript.

```
dds <- DESeqDataSetFromTximport(txi,
                                colData = metadata.df,
                                design = ~ condition) #
```

After creating the dds variable, run the DESeq function. This function estimates size factors to normalize the raw count data, fits a Generalized Linear Model (GLM) to estimate dispersion for each gene (a key quantity in the statistical test for DGE), calculates the log2 fold-change between groups, and conducts a Wald Test for differential gene expression. You can also use "rlog" to transform the normalized counts.
#Creating plots to visualize results

At this stage, many scientists prefer to analyze their data using visualizations. A commonly used method is the Principal Component Analysis (PCA) plot, which allows one to examine the variation in their data. Higher principal components (PCs) explain less variation and are, therefore, of less interest than PC1 and PC2. Outliers in the PCA plot may indicate poor library quality or contamination. PCA plots provide insight into the similarity of samples in their genome-wide expression profiles by visualizing their relative positions along the PCs. Below is an example of code one might use to generate a PCA plot. An example of a PCA plot is down below.

```
plotPCA(rld,intgroup="patient")
```



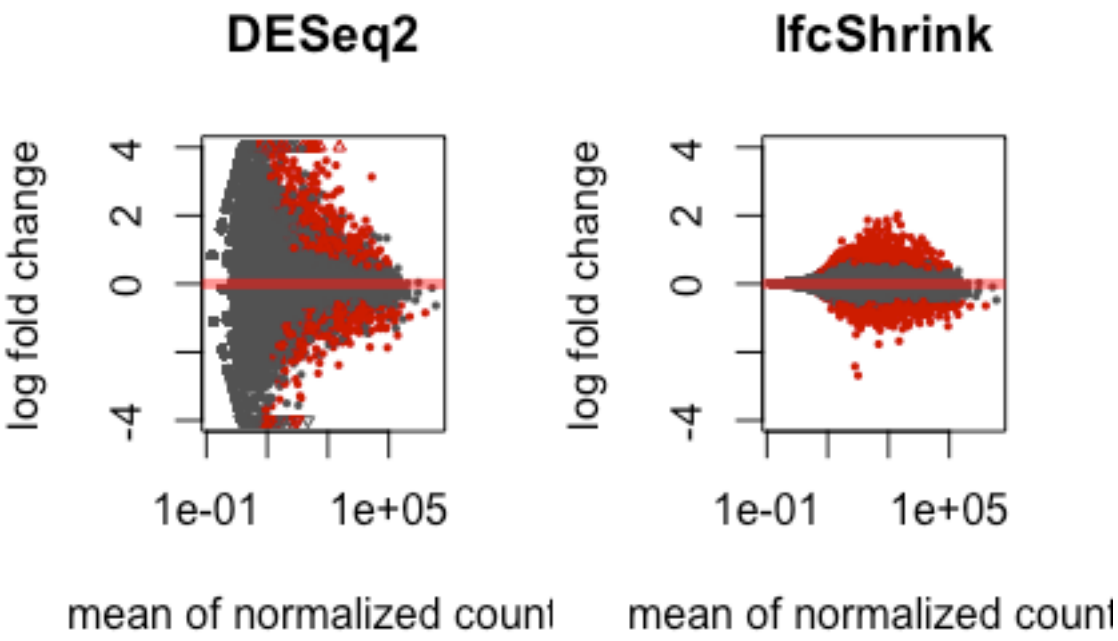
Scientists also use MA plots to illustrate the log2 fold-change (y-axis) versus the mean expression (x-axis) for each gene. The difference in mean expression between two conditions is estimated as the log2 foldchange (LFC). In the example, shrinkage log fold-change estimates are used in the analysis with lfcShrink, which applies a shrinkage estimator and returns a results table with shrunken log2FC estimates. Blue points represent statistically significant DEGs, while gray points indicate genes that are not significantly different between the treatment and control groups. Scientists can create this plot before and after LFC shrinkage to visualize the differences in fold-change estimates more clearly. An example of MA plot with and without shrinkage is below.

Using the results function on non shrinkage data and creating MA plot

```
res <- results(dds, contrast = c('condition','control','treatment') ) # set
numerator as treatment, denominator as normal (control) in LFC plotMA(res)
# no LFC shrinkage was applied to the `res` object
```

Using lfcShrink to produce shrunken log2FC

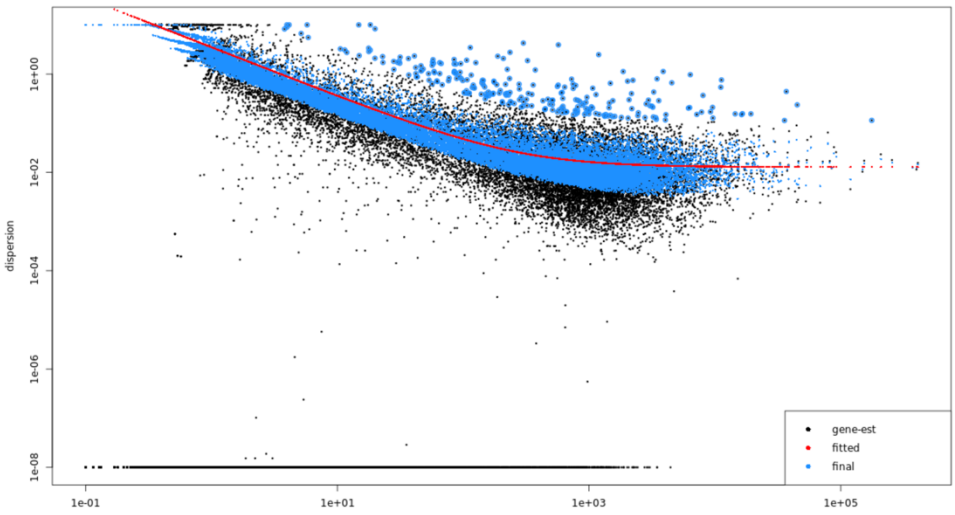
```
# note: lfcShrink is analogous to results function, but it produces
shrunken log2FC to the table, but without updating `dds` res.lfcShrink
<- lfcShrink(dds, coef =
'condition_treatment_vs_control',type = 'apeglm')
plotMA(res.lfcShrink) # LFC shrinkage was applied to this object
```



Another commonly used plot is the dispersion-by-mean plot, which estimates dispersion using Maximum Likelihood and then shrinks these estimates toward the fitted line to evaluate the relationship between gene-wise dispersion and the mean of normalized counts. The x-axis represents the mean of normalized counts, while the y-axis represents dispersion. The black points show the original gene-wise dispersion estimates, while the blue points represent the shrunken values, adjusted closer to the red fitted line. Bluecircled points indicate high-dispersion genes that DESeq does not shrink, as they may reflect cases where biological dispersion is higher. An example of a dispersion plot is down below.

Plotting gene-wise dispersion to the mean of normalized counts using the `plotDispEsts` function

```
plotDispEsts(dds)
```



To more easily analyze which genes show significant changes in expression, a table can be created containing the mean of normalized counts for all samples, log2 fold-change (MAP) for the treatment versus control condition, the posterior

standard deviation for the comparison, the Wald test p-value, and BHadjusted p-values. This can be done using the lfcShrink() function.

Sorting the genes based on the adjusted p-value applied by default is the false discovery rate (FDR) and is calculated using the Benjamin-Hochberg ("BH") method

```
# Here we "tidy" the data using a series of pipes to return a tibble
(tbl_df) with feature_id column with ENSEMBL gene ids and remaining columns
from the results object
# note: the biobroom package has functionality to streamline this
operation res.lfcShrink.tbl_df <- res.lfcShrink %>%
as.data.frame() %>%   rownames_to_column(var = "feature_id") %>%
as_tibble()

res.lfcShrink.tbl_df %>%
  arrange(padj) # sort tibble in ascending order on adjusted p-value (FDR)
```

Sorting a table of DEGs that are signficant at a False Discovery Rate (FDR) of 0.05.

Below is an example of a modified table where the genes are sorted based on the adjusted p-value (padj), which are p-values corrected for multiple testing using the false discovery rate (FDR) approach, calculated via the Benjamini-Hochberg (BH) method. This helps identify the number of genes with significantly higher or lower expression between groups and the number of statistically significant genes with an FDR of 0.05. You can also use the summary function to output the number of genes with significantly higher or lower expression in each group, along with percentages.

This table contains rows representing different genes, all of which are significantly affected by the conditions. The first column, feature_id, indicates the gene ID based on the Ensembl database. The baseMean value represents the total number of reads for the gene, averaged across samples. The log2FoldChange column shows the base-2 logarithm of the fold change in expression, where positive values indicate upregulation and negative values indicate downregulation. The lfcSE column shows the Log Fold Change Standard Error, which measures the variability and uncertainty of the log2FoldChange estimate. Finally, the pvalue and padj (adjusted p-value) columns show the probability of observing the results if the null hypothesis is true (i.e., no difference in expression between groups). Padj differs from pvalue as it accounts for multiple testing corrections.

```
res.lfcShrink.tbl_df %>%
  filter(padj < 0.05) %>%
  arrange(padj)
summary(res.lfcShrink)
```