

tutorial_RNASeq_DESeq2.Rmd

Contents

Load Data and libraries	1
View the data	2
Data cleanup	2
DESeq2 wet toes - Step 1	3
Build DESeq2 object	3
Plot PCA	3
DESeq2 wet toes - Step 2	5
Differential Expression Analysis	5
P-value versus Adjusted P-value	6
Data distribution	6
Adding gene names	7
Saving results file	8

Load Data and libraries

- i) First let us download the data

- **Instructions: Forthcoming**

We will be using an in-house dataset of Inbred Long Sleep, ILS and Inbred Short Sleep, ISS mice. The ILS strain is selected for ‘longer recovery’ from ethanol consumption. The ISS strain is selected for ‘quicker recovery’ from ethanol consumption (they don’t get hangovers). This is because the ILS strain has a particular allele of the *Lore2* gene that exhibits an increase in the *loss of righting response time*. You can read more about it here. Our dataset has 3 ILS samples treated with saline (controls) and 4 ISS samples treated with saline (treatment). The counts have been calculated using HTSeq.

We will be loading our data into the *object data_raw*. We will be loading our covariate information into the *object covariates*. By default, these objects are dataframes, a type of table in R.

– **data_raw** is a data matrix, where each row is a gene and each column is a sample.

– **covariates** contains information for each sample, defining experimental groups.

```
data_raw = read.table("ILS_ISS_saline_HTSeq_nostats.txt", header = TRUE, stringsAsFactors = FALSE)
covariates = read.table("ILS_ISS_saline_HTSeq_nostats_covars.txt", header = TRUE,
stringsAsFactors = FALSE)
```

- ii) Load libraries

If you do not have DESeq2 installed, you will need to run these two commands in your RStudio console:

```
source("http://bioconductor.org/biocLite.R")
biocLite("DESeq2")
```

```
library(DESeq2)
library(ggplot2)
```

```
library(biomaRt)
library(knitr)
```

View the data

DATA The values for each gene are raw **counts**. There should be 38,293 genes (rows), across 7 samples (columns). The *dim* function lets us view the dimensions of a dataframe.

```
print(dim(data_raw))
```

```
## [1] 38293      7
```

Let us take a look at the first 6 rows of the dataframe with the raw counts. The *head* command lets us do that. The *kable* command makes the output table look pretty in the pdf :) If you are running this in RStudio **Console**, you can just say `head(data_raw)`.

```
kable(head(data_raw))
```

	ILS_S_1_A	ILS_S_2_B	ILS_S_3_C	ILS_S_3_H	ISS_S_1_A	ISS_S_2_B	ISS_S_3_F
ENSMUSG000000000001	486	364	474	683	881	660	
ENSMUSG000000000003	0	0	0	0	0	0	
ENSMUSG000000000028	18	15	19	26	26	21	
ENSMUSG000000000031	5	1	6	6	11	13	
ENSMUSG000000000037	28	15	28	50	54	42	
ENSMUSG000000000049	4	2	2	3	6	7	

COVARIATES Let us also take a look at the covariates dataframe.

```
kable(covariates)
```

	Strain	Treatment
ILS_S_1_A	ILS	Saline
ILS_S_2_B	ILS	Saline
ILS_S_3_C	ILS	Saline
ILS_S_3_H	ILS	Saline
ISS_S_1_A	ISS	Saline
ISS_S_2_B	ISS	Saline
ISS_S_3_F	ISS	Saline

Data cleanup

- Remove all genes where there are zero counts for all samples. You'll be left with 25,600 genes.

```
data_clean = data_raw[as.logical(rowSums(data_raw != 0)), ]
dim(data_clean)
```

```
## [1] 25600      7
```

- Subset the data by the samples that have covariate information. We will also *relevel* our covariate column with information on the experimental type.

- We need to set the default ‘reference’ experimental strain to ILS, so that any fold changes are calculated as treatment (ISS) vs control (ILS).
- Our experimental groups are defined in the column covariates\$Strain

```
data = data_clean[, row.names(covariates)]
covariates$Strain = factor(covariates$Strain, levels = c("ILS", "ISS"))
```

DESeq2 wet toes - Step 1

Build DESeq2 object

We will use the count data in our dataframe **data**, and our covariate information in our dataframe **covariates**, to fit a **Strain** based model for the samples.

```
dds_counts <- DESeqDataSetFromMatrix(countData = data, colData = covariates, design = ~Strain)
ds_fit <- DESeq(dds_counts)
```

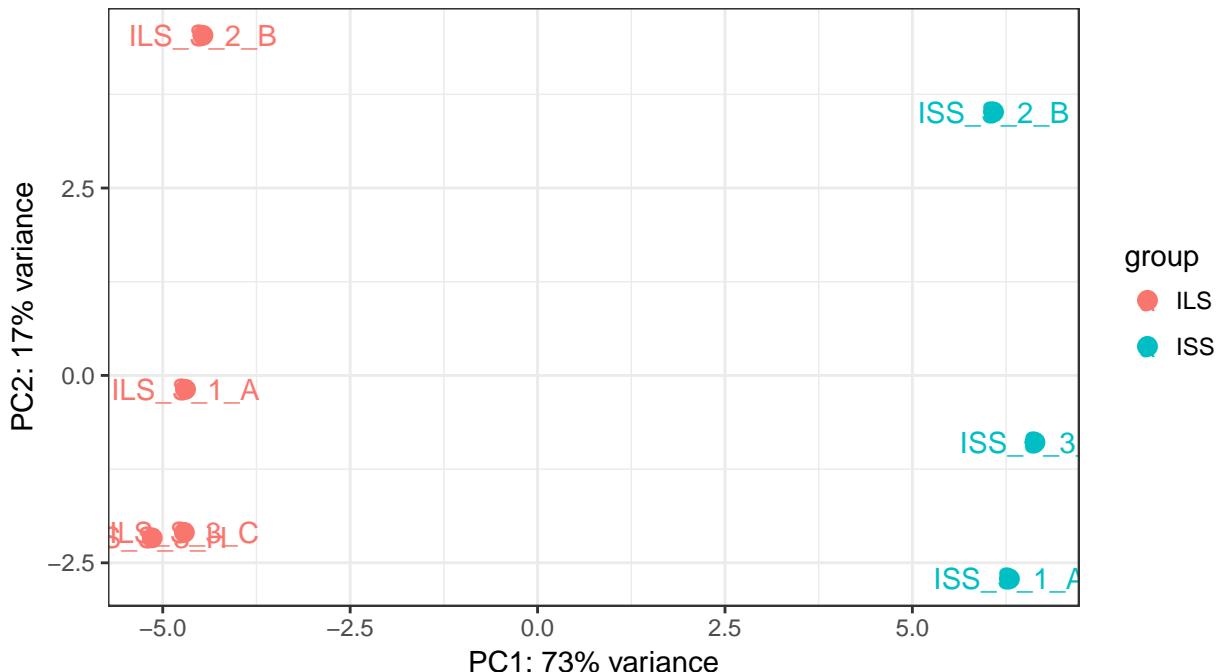
Plot PCA

In order to make sure our data looks sensible, and to check for any outliers, we can plot the first 2 Principal Components of the data. For this, we will first need to retrieve the log transformed counts from our fit object, which we will do with the **rlogTransformation** function in DESeq2.

```
rld <- rlogTransformation(ds_fit, blind = TRUE)
```

Then we use the **plotPCA** function in DESeq2 to plot the first 2 PC’s.

```
DESeq2::plotPCA(rld, intgroup = c("Strain") + theme_bw() + geom_text(aes(label = colnames(rld)))
```



As we can see, the ILS_S_2_B is quite far away from all the other samples. We will exclude this sample from future analysis. We will start by removing it from our covariates table.

```
covariates_new = covariates[rownames(covariates) != "ILS_S_2_B", ]
```

What are the samples in the covariates table now?

```
kable(covariates_new)
```

	Strain	Treatment
ILS_S_1_A	ILS	Saline
ILS_S_3_C	ILS	Saline
ILS_S_3_H	ILS	Saline
ISS_S_1_A	ISS	Saline
ISS_S_2_B	ISS	Saline
ISS_S_3_F	ISS	Saline

Next, we will again make sure we are keeping the data for the samples we have the covariate information for (no more *ILS_S_2_B*!). We will also remove any genes that have ‘zero’ counts in all of our samples.

```
data_new = data_clean[, rownames(covariates_new)]
data_new = data_new[as.logical(rowSums(data_new != 0)), ]
```

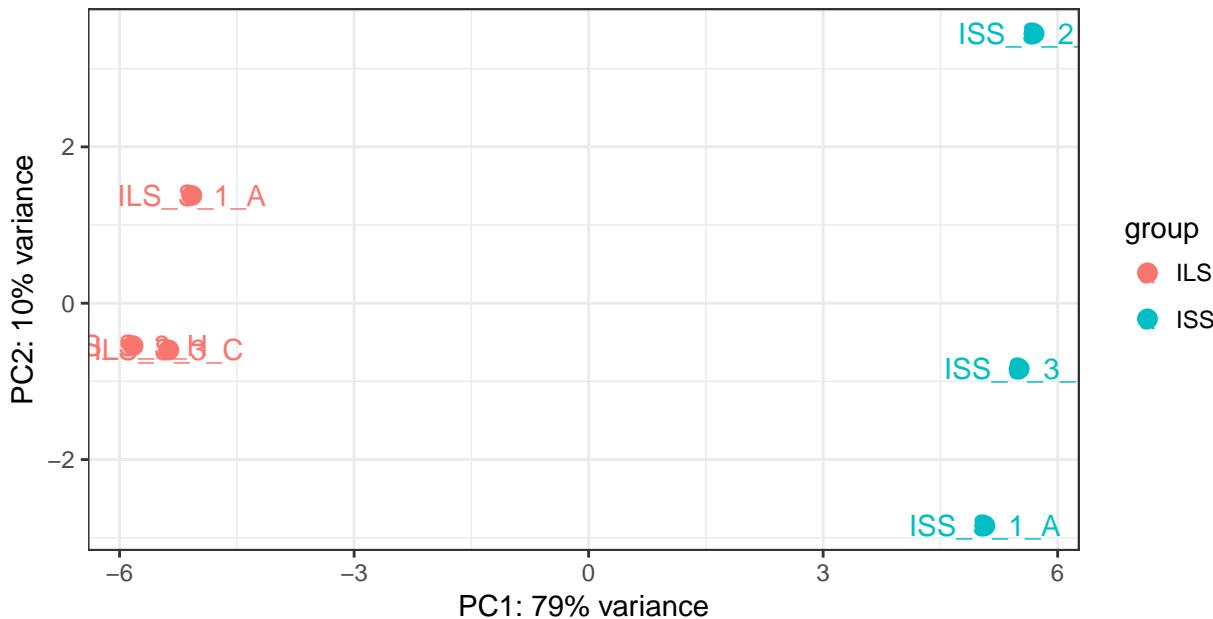
What are the dimensions of our new dataset? (We should have 25,394 genes, and 6 samples now).

```
print(dim(data_new))
```

```
## [1] 25394      6
```

Let us review our PCA plot, to make sure everything looks good and we have removed the outlier.

```
dds_new_counts <- DESeqDataSetFromMatrix(countData = data_new, colData = covariates_new,
                                             design = ~Strain)
ds_new_fit <- DESeq(dds_new_counts)
rld_new <- rlogTransformation(ds_new_fit, blind = TRUE)
```



Now let us compare the two Strain types in the model that we fit.

We will see log2 fold change results for Strain ISS vs ILS

```
res = results(ds_new_fit)
kable(head(res))
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
ENSMUSG000000000001	654.882728	-0.0966413	0.0980852	-0.9852794	0.3244869	0.7903037
ENSMUSG000000000028	22.091736	-0.1991458	0.1491001	-1.3356518	0.1816631	0.6549837
ENSMUSG000000000031	9.467788	0.1349042	0.1229589	1.0971485	0.2725765	0.7518182
ENSMUSG000000000037	40.647109	-0.0956190	0.1547720	-0.6178054	0.5367036	0.9017844
ENSMUSG000000000049	4.354693	0.0387544	0.0949037	0.4083552	0.6830129	0.9479047
ENSMUSG000000000056	826.979761	-0.0856696	0.0968421	-0.8846316	0.3763553	0.8292743

DESeq2 wet toes - Step 2

Differential Expression Analysis

Lastly, let us try and identify differentially expressed genes in our results object, `res`.

```
## [1] "Total genes are: 25394"
## [1] "Gene with p-value 'NA': 7"
```

Whatever is an NA p-value?

Sometimes with DESeq, a gene with a p-value of NA mean that the gene's counts were below DESeq2's internal threshold for assessing any sort of substantial differential expression. This is called *independent filtering*, and we can remove this by setting the `independentFiltering` flag to FALSE, like so:

```
res_nofilter = results(ds_new_fit, independentFiltering = FALSE)
```

How many genes with NA p-value are there in this new results data object?

```
## [1] "Total genes are: 25394"
## [1] "Gene with p-value 'NA': 7"
```

What? What are these genes then? Let us take a look at the genes in the `results` datafram which have a pvalue of NA.

```
kable(res[is.na(res$pvalue), ])
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
ENSMUSG00000020713	422.04367	0.0818306	0.0508294	1.6099057	NA	NA
ENSMUSG00000028298	25.98245	0.1891820	0.0760347	2.4880994	NA	NA
ENSMUSG00000046341	776.05051	1.8320542	0.1529313	11.9795901	NA	NA
ENSMUSG00000075014	83.86589	0.0245023	0.0416450	0.5883615	NA	NA
ENSMUSG00000096385	38.41533	0.0369702	0.0464921	0.7951926	NA	NA
ENSMUSG00000097312	80.27601	-0.0008426	0.0594809	-0.0141658	NA	NA
ENSMUSG00000097346	23.13011	0.1333450	0.0889143	1.4997030	NA	NA

In our case, it appears that there are some genes with count outliers. That is, a single sample has a count that is disproportionately impacting the log fold changes and resulting p-values. These are genes whose counts do not fit to a negative binomial distribution, but sadly this discussion lies outside the scope of this tutorial.

For now, we will simply fix for this by adjusting the *Cook's cutoff* that is used to determine count outliers (we can set the flag `cooksCutoff` to FALSE), like so:

```
res_nofilter = results(ds_new_fit, independentFiltering = FALSE, cooksCutoff = FALSE)
```

How many genes with NA p-value are there in this new results data object?

```
## [1] "Total genes are: 25394"
## [1] "Gene with p-value 'NA': 0"
```

P-value versus Adjusted P-value

- If our **null hypothesis** is that no gene is affected by the treatment with ethanol, then by random chance we would expect up to 1% of the genes to have a p-value below 0.01.
- To adjust for multiple testing, we can use the Benjamini-Hochberg test correction method. Nicely enough, DESeq2 does that for us automatically, and we can find these values in the `padj` column of our new results object, `res_nofilter`.

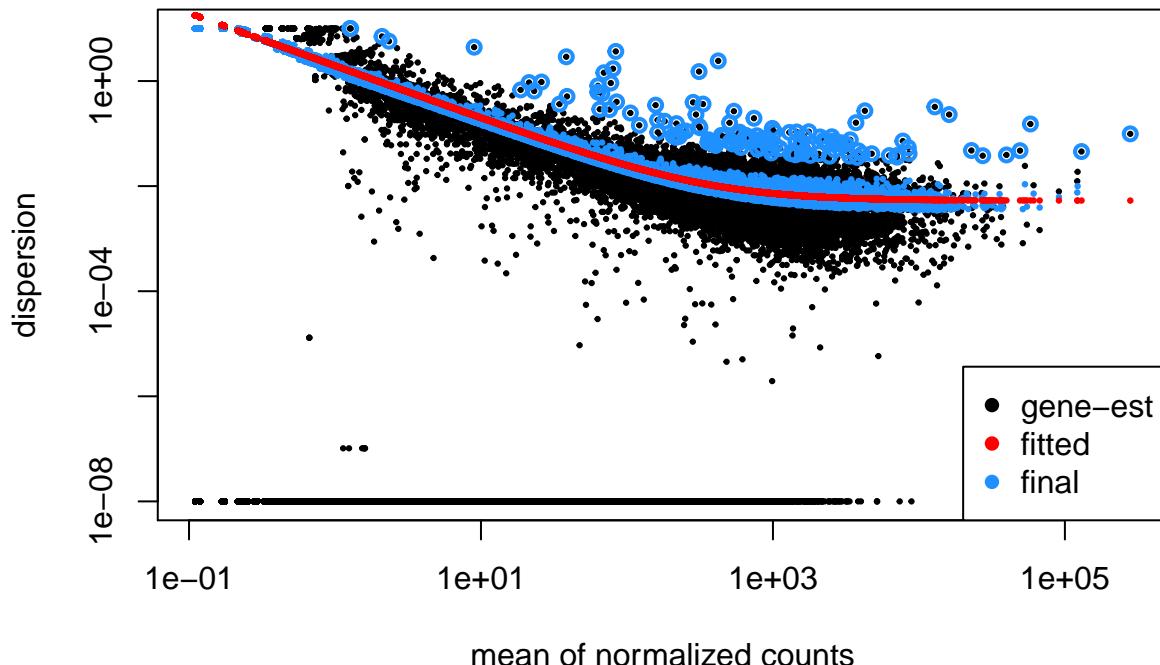
```
## [1] "Number of genes with p-value < 0.01: 1368"
## [1] "Number of genes with adjusted p-value < 0.01: 580"
## [1] "1% of total number of genes is 253.94"
```

Data distribution

This section is optional

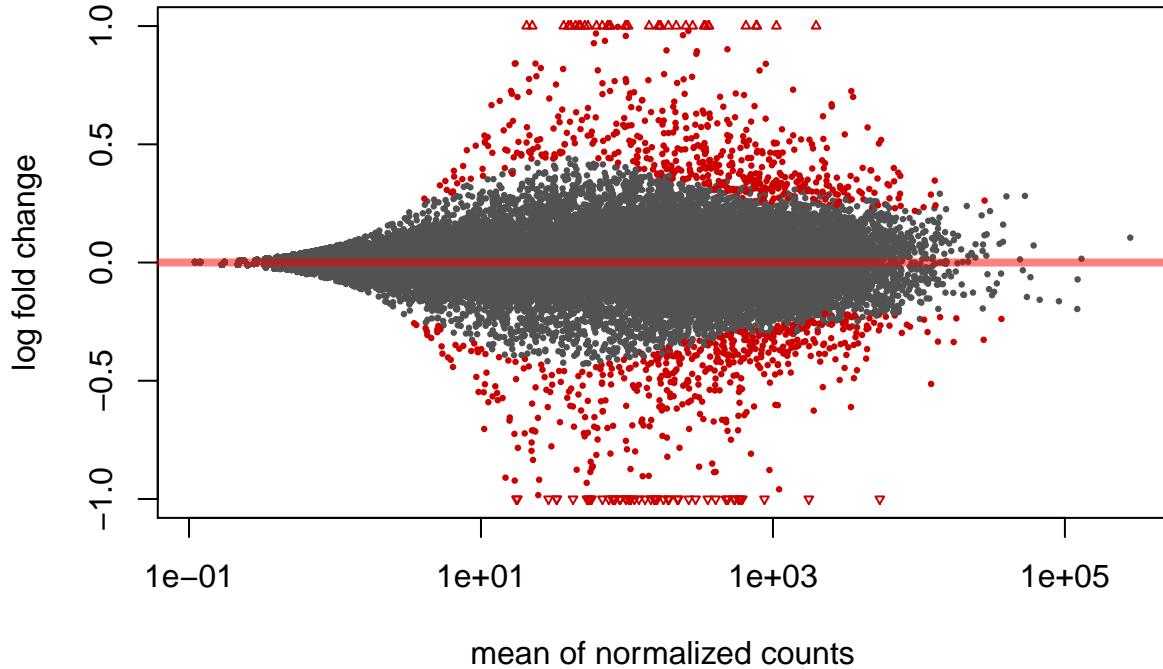
We can look at the distribution of gene values after log transformation, to assess how variable different genes are within the control and experimental group.

```
DESeq2::plotDispEsts(ds_new_fit)
```



We can also have an overview of the comparison with an *MA* plot.

```
plotMA(res_nofilter, ylim = c(-1, 1))
```



Adding gene names

optional section

We can replace our ensemble gene names with their official ‘mgi’ gene names, without using Google! Remember these are mice genes, that’s why we need to refer to the *M.musculus* ensembl dataset, and map over the ensemble IDs to the ‘MGI’ symbol.

```
ensembl = useMart("ensembl", dataset = "mmusculus_gene_ensembl")
genemap <- getBM(attributes = c("ensembl_gene_id", "mgi_symbol"), filters = "ensembl_gene_id",
                  values = rownames(res_nofilter), mart = ensembl)
idx <- match(rownames(res_nofilter), genemap$ensembl_gene_id)
res_nofilter$mgi_symbol <- genemap$mgi_symbol[idx]
```

The resulting results table looks like this:

```
kable(head(res_nofilter))
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj	mgi_symbol
ENSMUSG000000000001	654.882728	-0.0966413	0.0980852	-0.9852794	0.3244869	0.9932374	Gnai3
ENSMUSG000000000028	22.091736	-0.1991458	0.1491001	-1.3356518	0.1816631	0.9057523	Cdc45
ENSMUSG000000000031	9.467788	0.1349042	0.1229589	1.0971485	0.2725765	0.9932374	H19
ENSMUSG000000000037	40.647109	-0.0956190	0.1547720	-0.6178054	0.5367036	0.9932374	Scml2
ENSMUSG000000000049	4.354693	0.0387544	0.0949037	0.4083552	0.6830129	0.9932374	Apoh
ENSMUSG000000000056	826.979761	-0.0856696	0.0968421	-0.8846316	0.3763553	0.9932374	Narf

Saving results file

We can save our results object to a csv file, to analyze in Excel or other tools of our choice.

```
write.csv(as.data.frame(res_nofilter), file = "results.csv")
```