Differential Expression and Visualization in R

Taylor Reiter N. Tessa Pierce

Learning objectives:

- Create a gene-level count matrix of Salmon quantification using tximport
- Perform differential expression of a single factor experiment in DESeq2
- Perform quality control and exploratory visualization of RNA-seq data in R

Tximeta

We first need to read our data into R. To do that, we will use a package called tximeta. We use tximeta for two main reasons: first, it facilitates summarizing transcript-level counts from Salmon to the gene-level for differential expression analysis. Second, tximeta enhances incorporates metadata (e.g. transcript locations, transcript and genome source and version, appropriate chromosome lengths, etc) for each transcriptome. This ensures computational reproducibility by attaching critical annotation information to the data object, such that exact quantifications can be reproduced from raw data (all software versions are also attached to the data object).

For more information, see the tximeta vignette

Since we're working in binder, all of the software is already installed. If you'd like to do a similar analysis on your own system, you can use the following installation commands:

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("readr")
install.packages("pheatmap")
#install.packages("RSQlite")
install.packages("data.table")
install.packages("kableExtra")

if (!requireNamespace("BiocManager", quietly = TRUE))
 install.packages("BiocManager", version = "3.11")
BiocManager::install("tximeta")
BiocManager::install("DESeq2")
BiocManager::install("SummarizedExperiment")
```

```
library(SummarizedExperiment)
library(tximeta)
library(DESeq2)
library(dplyr)
library(ggplot2)
library(readr)
library(pheatmap)
```

This lesson is executed in binder, so the relevant paths should work if we're running this from within the 2020-ggg-201b-rnaseq folder.

names	files	condition
ERR458493	rnaseq/quant/ERR458493_quant/quant.sf	wt
ERR458494	rnaseq/quant/ERR458494_quant/quant.sf	wt
ERR458495	rnaseq/quant/ERR458495_quant/quant.sf	wt
ERR458500	rnaseq/quant/ERR458500_quant/quant.sf	snf2
ERR458501	rnaseq/quant/ERR458501_quant/quant.sf	snf2
ERR458502	rnaseq/quant/ERR458502_quant/quant.sf	snf2

First let's read in our `rnaseq_samples.csv` file. This file designates the names of our samples, their conditions, and the path to their quantification files.

```
samples <- read_csv("rnaseq_samples.csv")
```

```
## Rows: 6 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): names, files, condition
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Next we need to define our reference files. We will use ensembl files, as this simplifies the way we interact with this information.

```
indexDir <- file.path("rnaseq", "quant", "sc_ensembl_index")
fastaFTP <- c("ftp://ftp.ensembl.org/pub/release-99/fasta/saccharomyces_cerevisiae/cdna/Saccharomyces_c
gtfPath <- "ftp://ftp.ensembl.org/pub/release-99/gtf/saccharomyces_cerevisiae/Saccharomyces_cerevisiae.
gtfLocal <- "yeast_ref/Saccharomyces_cerevisiae.R64-1-1.99.gtf.gz"
fastaLocal <- "yeast_ref/Saccharomyces_cerevisiae.R64-1-1.cdna.all.fa.gz"
```

With this information set, we can now build an object that saves this information.

```
makeLinkedTxome(indexDir = indexDir,
                source   = "Ensembl",
                organism = "Saccharomyces cerevisiae",
                release  = "99",
                genome   = "GCA_000146045.2",
                fasta    = fastaLocal,
                gtf      = gtfLocal,
                write    = FALSE)
```

```
## saving linkedTxome in bfc (first time)
```

To create an `se` object for using our `tximeta` information:

```
se <- tximeta(samples)
```

```
## importing quantifications
```

```
## reading in files with read_tsv
```

```
## 1 2 3 4 5 6
## found matching linked transcriptome:
## [ Ensembl - Saccharomyces cerevisiae - release 99 ]
## useHub=TRUE: checking for EnsDb via 'AnnotationHub'
## using temporary cache /tmp/RtmptZJghq/BiocFileCache
## snapshotDate(): 2020-10-27
## found matching EnsDb via 'AnnotationHub'
```

```
## downloading 1 resources
## retrieving 1 resource
## loading from cache
## require("ensembldb")
## generating transcript ranges
```

To look at our `se` object, we can use the following commands:

```
colData(se)
assayNames(se)
rowRanges(se)
seqinfo(se)
```

And to summarize to gene level, we can use `summarizeToGene()`.

```
gse <- summarizeToGene(se)
```

```
## loading existing EnsDb created: 2023-03-10 14:01:04
## obtaining transcript-to-gene mapping from database
## generating gene ranges
## summarizing abundance
## summarizing counts
## summarizing length
```

And now let's take a look at the data

```
rowRanges(gse)
mcols(gse)
```

Why do we need to normalize and transform read counts

Given a uniform sampling of a diverse transcript pool, the number of sequenced reads mapped to a gene depends on:

- its own expression level,
- its length,
- the sequencing depth,
- the expression of all other genes within the sample.

In order to compare the gene expression between two conditions, we must therefore calculate the fraction of the reads assigned to each gene relative to the total number of reads and with respect to the entire RNA repertoire which may vary drastically from sample to sample. While the number of sequenced reads is known, the total RNA library and its complexity is unknown and variation between samples may be due to contamination as well as biological reasons. **The purpose of normalization is to eliminate systematic effects that are not associated with the biological differences of interest.**

Normalization aims at correcting systematic technical biases in the data, in order to make read counts comparable across samples. The normalization proposed by DESeq2 relies on the hypothesis that most features are not differentially expressed. It computes a scaling factor for each sample. Normalized read counts are obtained by dividing raw read counts by the scaling factor associated with the sample they belong to.

Differential Expression with `DESeq2`

Image credit: Paul Pavlidis, UBC

Differential expression analysis with DESeq2 involves multiple steps as displayed in the flowchart below. Briefly,

- DESeq2 will model the raw counts, using normalization factors (size factors) to account for differences in library depth.
- Then, it will estimate the gene-wise dispersions and shrink these estimates to generate more accurate estimates of dispersion to model the counts.
- Finally, DESeq2 will fit the negative binomial model and perform hypothesis testing using the Wald test or Likelihood Ratio Test.

We're now ready to use `DESeq2`, the package that will perform differential expression.

We'll start with a function called `DESeqDataSetFromTximport` which will transform our `txi` object into something that other functions in `DESeq2` can work on. This is where we also give information about our samples contain in the `samples` data.frame, and where we provide our experimental design. **A design formula tells the statistical software the known sources of variation to control for, as well as, the factor of interest to test for during differential expression testing. Here our experimental design has one factor with two levels.**

```
dds <- DESeqDataSet(se = gse, design = ~condition)
```

```
## using counts and average transcript lengths from tximeta
```

```
## Warning in DESeqDataSet(se = gse, design = ~condition): some variables in design
## formula are characters, converting to factors
```

Note: DESeq stores virtually all information associated with your experiment in one specific R object, called DESeqDataSet. This is, in fact, a specialized object of the class "SummarizedExperiment". This, in turn,is a container where rows (rowRanges()) represent features of interest (e.g. genes, transcripts, exons) and columns represent samples (colData()). The actual count data is stored in theassay()slot.

The first thing we notice is that both our counts and average transcript length were used to construct the `DESeq` object. We also see a warning message, where our condition was converted to a factor. Both of these messages are ok to see!

Now that we have a DESeq2 object, we can can perform differential expression.

```
dds <- DESeq(dds)
```

```
## estimating size factors
```

```
## using 'avgTxLength' from assays(dds), correcting for library size
```

```
## estimating dispersions
```

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

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

```
## final dispersion estimates
```

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

Everything from normalization to linear modeling was carried out by the use of a single function! This function prints out a message for the various steps it performs.

And look at the results! The results() function lets you extract the base means across samples, log2-fold changes, standard errors, test statistics etc. for every gene.

```
res <- results(dds)
```

```
head(res)
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
Q0010	0.0000000	NA	NA	NA	NA	NA
Q0017	0.0000000	NA	NA	NA	NA	NA
Q0032	0.0000000	NA	NA	NA	NA	NA
Q0045	0.1130105	0.1518424	4.080473	0.0372120	0.9703160	NA
Q0050	0.3409296	-1.3316013	2.720271	-0.4895106	0.6244803	NA
Q0055	0.1130742	0.1518424	4.080473	0.0372120	0.9703160	NA

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
YAL005C	1817.23835	1.417483	0.0443141	31.987155	0.0000000	0.0000000
YAL025C	62.79625	1.056928	0.2293775	4.607812	0.0000041	0.0000396
YAL026C-A	15.71888	-1.470673	0.5105664	-2.880475	0.0039708	0.0197910
YAL038W	8204.96119	1.594199	0.0227086	70.202541	0.0000000	0.0000000
YAL044C	200.60392	1.732260	0.1378569	12.565634	0.0000000	0.0000000
YAL054C	62.63189	-1.179931	0.2461361	-4.793815	0.0000016	0.0000168

The first thing that prints to the screen is information about the "contrasts" in the differential expression experiment. By default, DESeq2 selects the alphabetically first factor to the be "reference" factor. Here that doesn't make that much of a difference. However, it does change how we interpret the log2foldchange values. We can read these results as, "Compared to *SNF2* mutant, WT had a decrease of -0.2124 in log2fold change of gene expression.

Speaking of log2fold change, what do all of these columns mean?

baseMean	giving means across all samples
log2FoldChange	log2fold changes of gene expression from one condition to another. Reflects how different the expression of a gene in one condition is from the expression of the same gene in another condition.
lfcSE	standard errors (used to calculate p value)
stat	test statistics used to calculate p value)
pvalue	p-values for the log fold change
padj	adjusted p-values

We see that the default differential expression output is sorted the same way as our input counts. Instead, it can be helpful to sort and filter by adjusted p value or log2 Fold Change:

```
res_sig <- subset(res, padj<.05)
res_lfc <- subset(res_sig, abs(log2FoldChange) > 1)
```

```
head(res_lfc)
```

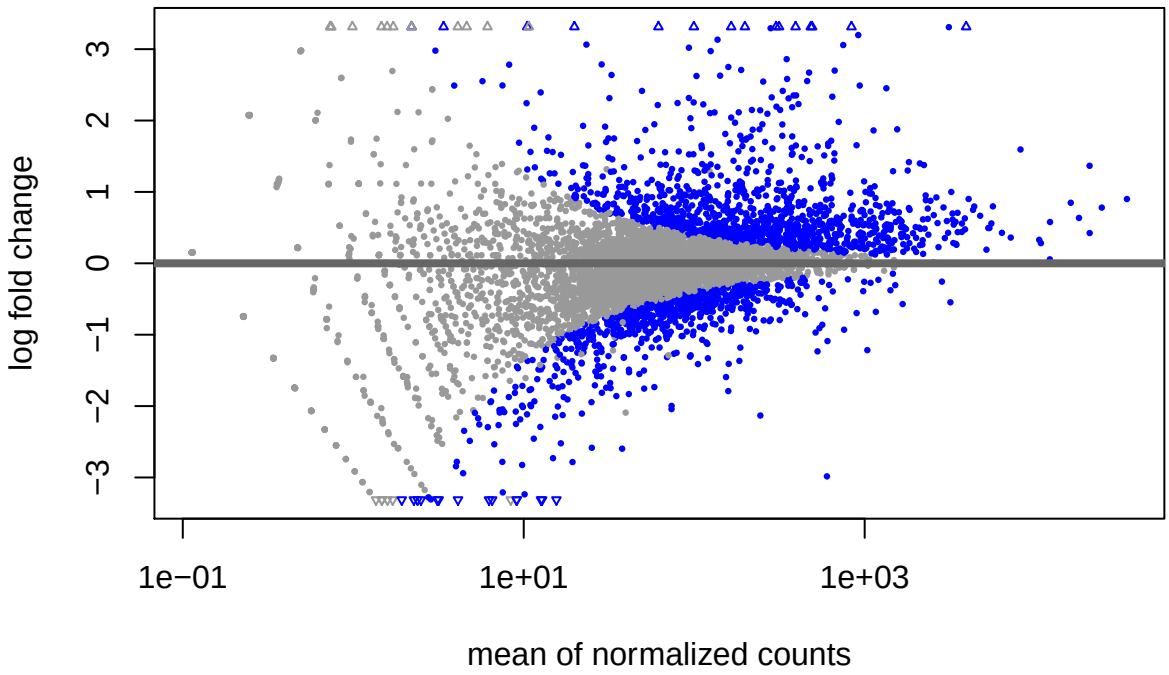
Visualization of RNA-seq and Differential Expression Results

Looking at our results is great, but visualizing them is even better!

MA Plot

The MA plot provides a global view of the relationship between the expression change between conditions (log ratios, M), the average expression strength of the genes (average mean, A) and the ability of the algorithm to detect differential gene expression: genes that pass the significance threshold are colored in red

```
plotMA(res)
```

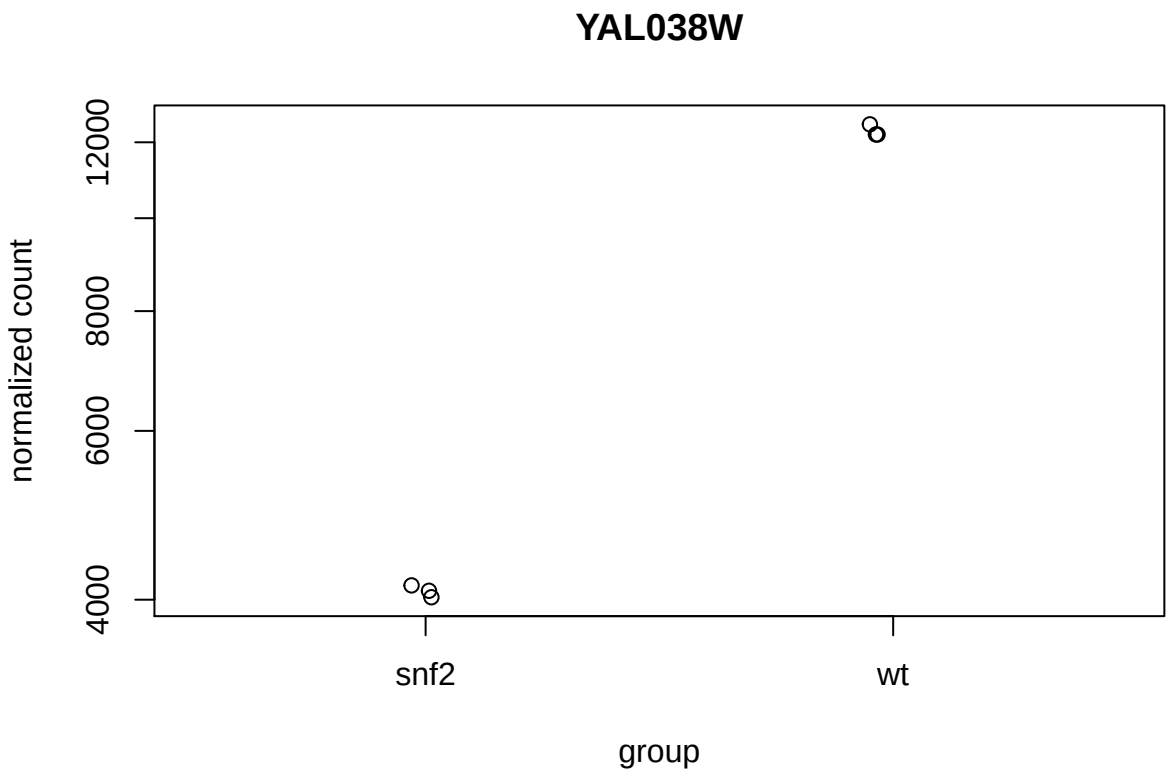


Question
Why are more genes grey on the left side of the axis than on the right side?

Plotting individual genes

Although it's helpful to plot many (or all) genes at once, sometimes we want to see how counts change for a specific gene. We can use the following code to produce such a plot:

```
plotCounts(dds, gene=which.min(res$padj), intgroup="condition")
```



Question
What gene is plotted here (i.e., what criteria did we use to select a single gene to plot)?

Normalization & Transformation

`DESeq2` automatically normalizes our count data when it runs differential expression. However, for certain plots, we need to normalize our raw count data. One way to do that is to use the `vst()` function. It perform variance stabilized transformation on the count data, while controlling for library size of samples.

```
vsd <- vst(dds)
```

MDS Plot

An MDS (multi-dimensional scaling) plot gives us an idea of how our samples relate to each other. The closer two samples are on a plot, the more similar all of their counts are. To generate this plot in DESeq2, we need to calculate "sample distances" and then plot them.

```
# calculate sample distances
sample_dists <- assay(vsd) %>%
  t() %>%
  dist() %>%
  as.matrix()
```

```
head(sample_dists)
```

Next, let's calculate the MDS values from the distance matrix.

	ERR458493	ERR458494	ERR458495	ERR458500	ERR458501	ERR458502
ERR458493	0.00000	20.40146	20.29342	36.36871	36.50662	36.61701
ERR458494	20.40146	0.00000	19.80574	36.33875	36.30391	36.39905
ERR458495	20.29342	19.80574	0.00000	36.39983	36.56026	36.79698
ERR458500	36.36871	36.33875	36.39983	0.00000	13.37862	13.15368
ERR458501	36.50662	36.30391	36.56026	13.37862	0.00000	13.46619
ERR458502	36.61701	36.39905	36.79698	13.15368	13.46619	0.00000

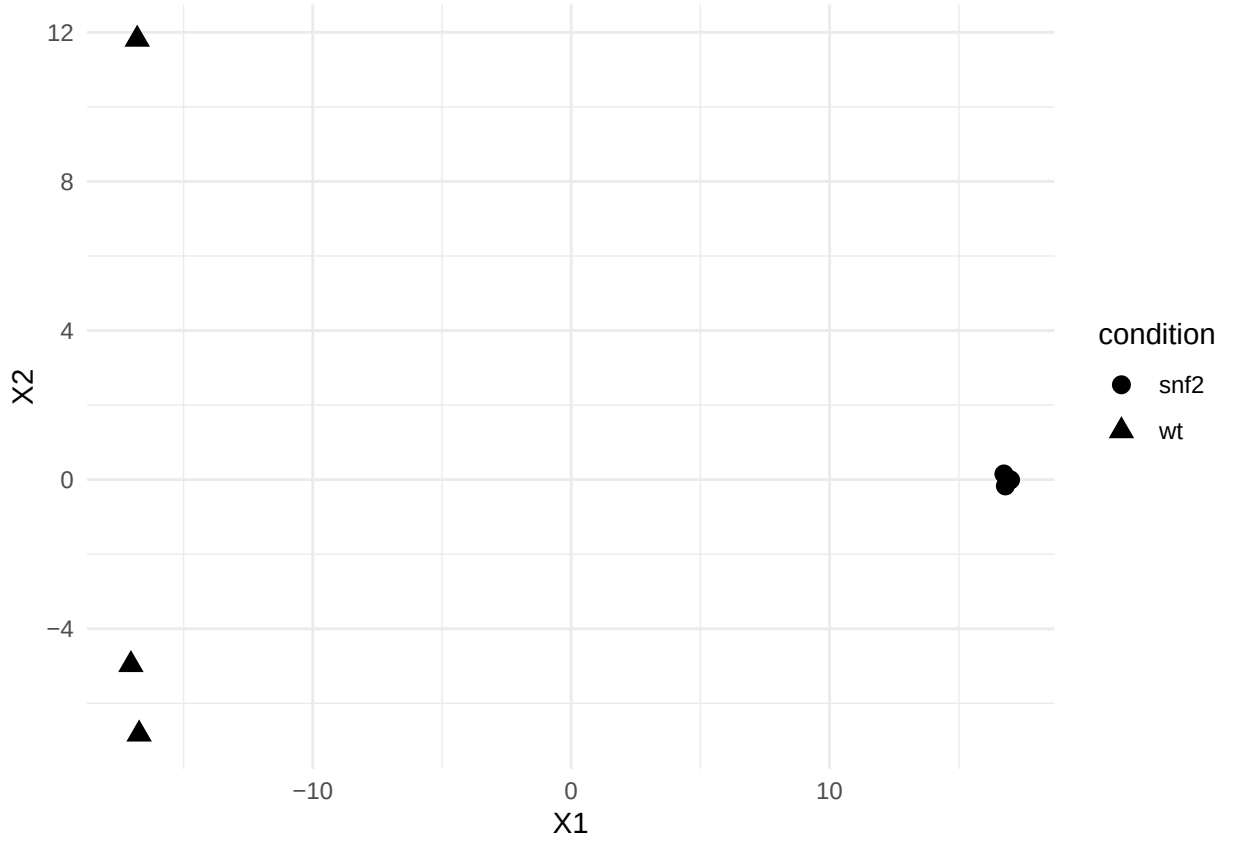
	X1	X2	names	condition
ERR458493	-16.79386	11.8104465	ERR458493	wt
ERR458494	-16.71968	-6.8263477	ERR458494	wt
ERR458495	-17.03719	-4.9635162	ERR458495	wt
ERR458500	16.74530	0.1529456	ERR458500	snf2
ERR458501	16.80243	-0.1679149	ERR458501	snf2
ERR458502	17.00299	-0.0056133	ERR458502	snf2

```
mdsData <- data.frame(cmdscale(sample_dists))
mds <- cbind(mdsData, as.data.frame(colData(vsd))) # combine with sample data
```

```
head(mds)
```

And plot with ggplot2!

```
ggplot(mds, aes(X1, X2, shape = condition)) +
  geom_point(size = 3) +
  theme_minimal()
```



Question
How similar are the samples between conditions?

Heatmap

Heatmaps are a great way to look at gene counts. To do that, we can use a function in the `pheatmap` package. Here we demonstrate how to install a package from the CRAN repository and then load it into our environment.

Next, we can select a subset of genes to plot. Although we could plot all ~6000 yeast genes, let's choose the 20 genes with the largest positive log2fold change.

```
genes <- order(res_lfc$log2FoldChange, decreasing=TRUE)[1:20]
```

We can also make a data.frame that contains information about our samples that will appear in the heatmap. We will use our samples data.frame from before to do this.

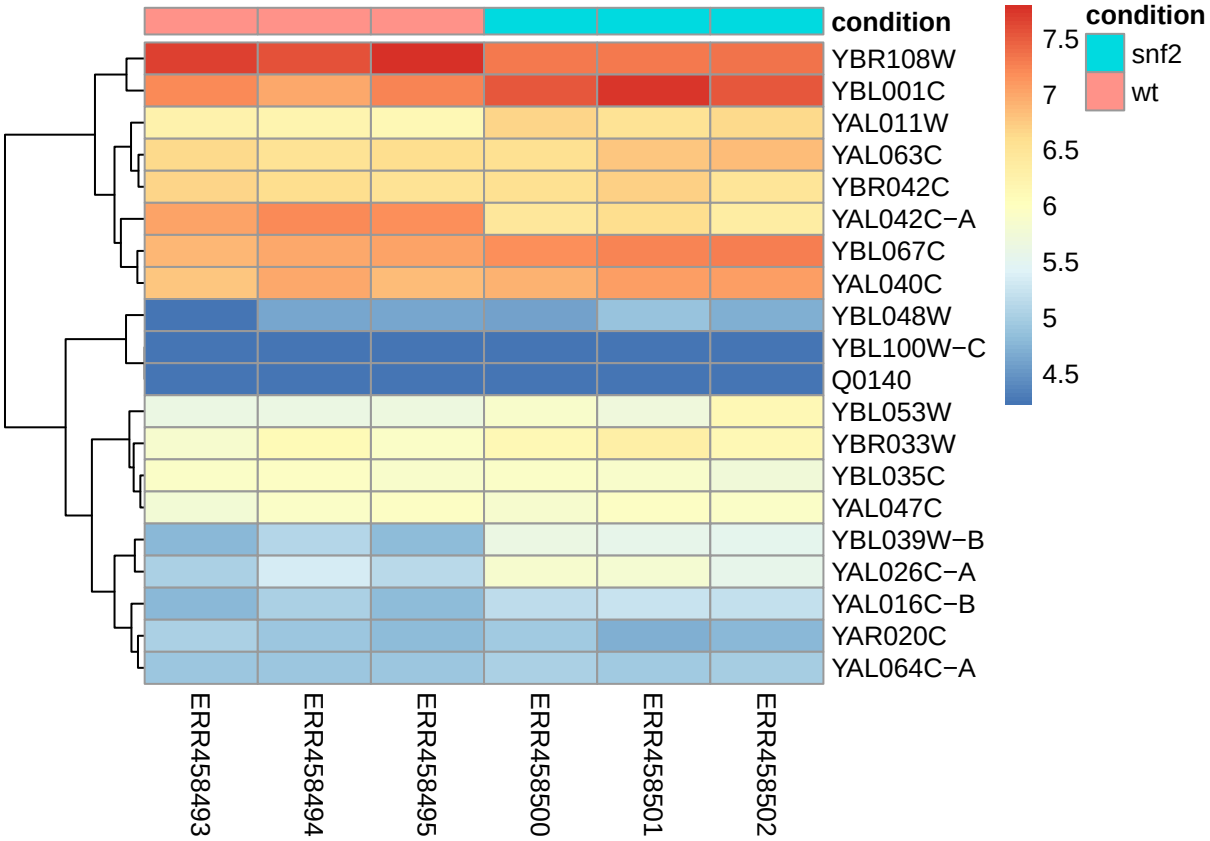
```
annot_col <- samples %>%
        tibble::column_to_rownames('names') %>%
        dplyr::select(condition) %>%
        as.data.frame()
```

```
head(annot_col)
```

And now plot the heatmap!

```
pheatmap(assay(vsd)[genes, ], cluster_rows=TRUE, show_rownames=TRUE,
         cluster_cols=FALSE, annotation_col=annot_col)
```

	condition
ERR458493	wt
ERR458494	wt
ERR458495	wt
ERR458500	snf2
ERR458501	snf2
ERR458502	snf2



We see that our samples do cluster by condition, but that by looking at just the counts, the patterns aren't very strong. How does this compare to our MDS plot?

Question
When are heatmaps useful?
What other types of heatmaps have you seen in the wild?

Further Notes

Here are some helpful notes or resources for anyone performing differential expression.

- Introduction to differential gene expression analysis using RNA-seq (Written by Friederike Dündar, Luce Skrabanek, Paul Zumbo). Click here
- Introduction to DGE - click here