Day 8: Advanced DESeq2 Experimental Design (Worksheet)
By: Samuel Hunter

We will be working through some “advanced” DESeq2 designs. On your first day with DESeq2, you learned how to run a pairwise comparison and interpret/visualize your results. But our experimental designs are often more complex than just a simple pairwise comparison. How can we incorporate more design elements into DESeq2?

Download all files in /scratch/Shares/public/sread2022/day8/data_files from AWS to your local computer. You’ll need to replace all of the paths to these files in the instructions with the local path you stored them to. So, for the first step, I would replace “/path/to/your/files/batch_example_counts.txt “ with “/Users/samuelhunter/sread2022/day8/data_files/batch_example_counts.txt”

Part 1: Batch Effect Correction

1. Load in the counts file for batch correction, along with our DESeq2 library:

```
> library(DESeq2)
> countdata <- read.delim("/path/to/your/files/batch_example_counts.txt",
+                         sep="\t", header=TRUE)
```

2. Load in the metadata file for batch correction:

```
> metadata <- read.csv("/path/to/your/files/batch_example_metadata.csv",
+                      sep=",",header=TRUE)
```

3. Organize the counts file into a DESeq2 compatible matrix, and check the metadata file:

```
> head(countdata)
      GeneID TranscriptID Length D21_V_1 D21_IFN_1 D21_V_2 D21_IFN_2 D21_V_3 D21_IFN_3
1 ERCC-00002     DQ459430   1061  274895    310070  338941    319945  224632    304592
2 ERCC-00003     DQ516784   1023   19636     23099   25111      3883   16106     21587
3 ERCC-00004     DQ516752    523   67338     80546   84636     36315   54069     73629
4 ERCC-00009     DQ668364    984    9171     10620   11229     10056    7321     10037
5 ERCC-00012     DQ883670    994       1         5       2         0       2         5
6 ERCC-00013     EF011062    808       9        15      19        27      12        16
> rownames(countdata) <- countdata[,2]
> countdata <- countdata[,-c(1,2,3)]
> head(countdata)
         D21_V_1 D21_IFN_1 D21_V_2 D21_IFN_2 D21_V_3 D21_IFN_3
DQ459430  274895    310070  338941    319945  224632    304592
DQ516784   19636     23099   25111      3883   16106     21587
DQ516752   67338     80546   84636     36315   54069     73629
DQ668364    9171     10620   11229     10056    7321     10037
DQ883670       1         5       2         0       2         5
EF011062       9        15      19        27      12        16
```

```
> head(metadata)
       Name batch treatmentIFN
1   D21_V_1     1      control
2 D21_IFN_1     1          IFN
3   D21_V_2     2      control
4 D21_IFN_2     2          IFN
5   D21_V_3     3      control
6 D21_IFN_3     3          IFN
```

4. View the model matrix. The rows of the model matrix match up with the rows of your metadata:

```
> model.matrix(~treatmentIFN,data = metadata)
  (Intercept) treatmentIFNIFN
1           1               0
2           1               1
3           1               0
4           1               1
5           1               0
6           1               1
attr(,"assign")
[1] 0 1
attr(,"contrasts")
attr(,"contrasts")$treatmentIFN
[1] "contr.treatment"
```

5. For now we aren't using batch information, only sample treatment information.

6. Run DESeq2:

```
> dds <- DESeqDataSetFromMatrix(countData=countdata,colData = metadata,design = ~treatmentIFN)
> dds <- DESeq(dds)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
```

7. View "NM_016817" in results file. Take note of the Log2FoldChange and LfcSE values (This is a strong IFN response gene)

```
> res <- results(dds)
> res[rownames(res)=="NM_016817",]
log2 fold change (MLE): treatmentIFN IFN vs control
Wald test p-value: treatmentIFN IFN vs control
DataFrame with 1 row and 6 columns
                 baseMean     log2FoldChange             lfcSE              stat               pvalue                padj
                <numeric>          <numeric>         <numeric>         <numeric>            <numeric>           <numeric>
NM_016817 17829.2724560213 0.534463855509179 0.113379616531156 4.71393246741414 2.42981222179023e-06 0.000190503029223199
```

Visualize the results with an MA plot:

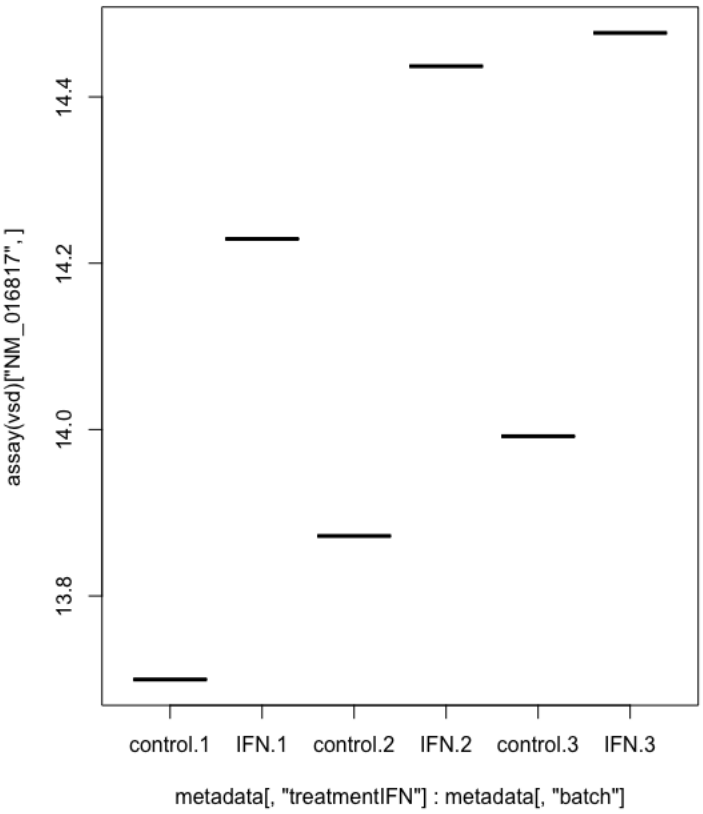
```
> plotMA(res)
>
```

And get a summary printout of the results

```
> summary(res)
```

8. Now fetch the normalized counts for each sample, and make a plot of this gene. You'll notice some substantial batch effects

```
> vsd <- vst(dds)
> boxplot(assay(vsd)['NM_016817',] ~ metadata[,'treatmentIFN'] + metadata[,'batch'])
```



9. You can also use a PCA plot to view this on a global scale:
   ```
   plotPCA(vsd,intgroup="batch")
   ```
10. We can correct for these effects by including another term in our design matrix. Take a look at our new model matrix:

```
> model.matrix(~batch+treatmentIFN,data = metadata)
  (Intercept) batchB batchC treatmentIFNIFN
1           1      0      0               0
2           1      0      0               1
3           1      1      0               0
4           1      1      0               1
5           1      0      1               0
6           1      0      1               1
attr(,"assign")
[1] 0 1 1 2
attr(,"contrasts")
attr(,"contrasts")$batch
[1] "contr.treatment"

attr(,"contrasts")$treatmentIFN
[1] "contr.treatment"
```

11. Notice that we've included a new term which additionally accounts for the batch effect. Now run DESeq2 again with this new design formula. View "NM_016817" again.

```
> dds <- DESeqDataSetFromMatrix(countData=countdata,colData = metadata,design = ~batch+treatmentIFN)
> dds <- DESeq(dds)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
> res <- results(dds)
> res[rownames(res)=="NM_016817",]
log2 fold change (MLE): treatmentIFN IFN vs control
Wald test p-value: treatmentIFN IFN vs control
DataFrame with 1 row and 6 columns
                 baseMean    log2FoldChange             lfcSE             stat              pvalue               padj
                <numeric>         <numeric>         <numeric>        <numeric>           <numeric>          <numeric>
NM_016817 17829.2724560213 0.535563860873358 0.0863779947657732 6.20023493628926 5.63789371097974e-10 8.52226140858758e-08
```

You'll notice that the same gene now has a smaller lfcSE but a similar Log2FoldChange. This is because we've explained some of that standard error by the differences in basal levels in each batch. But, each batch responded similarly, so the log2FC doesn't shift much. We can now more confidently call this gene as significant (padj is much smaller now)

Batch correction will help with finding significant calls (more likely true positives) while avoiding batch-specific significant calls (which are likely false positives).

Visualize the results with an MA plot and summarize results as before:

```
> summary(res)
```

```
> plotMA(res)
>
```

Part 2: Within-group contrasts

1. Load in the counts file for contrast

```
> countdata <- read.delim("/path/to/your/files/contrast_example_counts.txt",
+                         sep="\t", header=TRUE)
```

2. Load in the metadata file:

```
> metadata <- read.csv("/path/to/your/files/contrast_example_metadata.csv",
+                      sep=",",header=TRUE)
```

3. View the metadata and counts file. Notice that we have multiple values under the "Person" column.

```
> (metadata)
          Name batch    person
1        Eli_A     A       Eli
2        Eli_B     B       Eli
3        Eli_C     C       Eli
4  Elizabeth_A     A Elizabeth
5  Elizabeth_B     B Elizabeth
6  Elizabeth_C     C Elizabeth
7       Eric_A     A      Eric
8       Eric_B     B      Eric
9       Eric_C     C      Eric
10     Ethan_A     A     Ethan
11     Ethan_B     B     Ethan
12     Ethan_C     C     Ethan
```

4. Run DESeq2 (We'll leave batch correction out for now):

```
> rownames(countdata) <- countdata[,1]
> countdata <- countdata[,-c(1,2,3,4,5,6)]
> dds <- DESeqDataSetFromMatrix(countData=countdata,colData = metadata,design = ~person)
> dds <- DESeq(dds)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
```

5. Generate results file using a contrast. Compare "Ethan" and "Elizabeth" and get some summary results:

```
> res <- results(dds,contrast = c("person","Ethan","Elizabeth"))
> plotMA(res)
> summary(res)
```

```
> res <- results(dds,contrast = c("person","Ethan","Elizabeth"))
> head(res)
log2 fold change (MLE): person Ethan vs Elizabeth
Wald test p-value: person Ethan vs Elizabeth
DataFrame with 6 rows and 6 columns
                 baseMean     log2FoldChange             lfcSE               stat            pvalue              padj
                <numeric>          <numeric>         <numeric>          <numeric>         <numeric>         <numeric>
NR_046018 1.07449047357762  0.107998448013946  1.82689250517053 0.0591159292121926 0.952859771153804                NA
NR_024540 75.0618625564708 0.0657629529542215 0.448914355158083  0.146493317040537 0.883531965236386 0.999809615249678
```

Notice in the first lines of the results printout, it says that the log2 fold change and Wald test are performed comparing Ethan and Elizabeth, as specified in our contrast

6. Regenerate the results file comparing another two people. You can do whichever one you like, but here I'll use "Ethan" and "Eric"

```
> res <- results(dds,contrast = c("person","Ethan","Eric"))
> head(res)
log2 fold change (MLE): person Ethan vs Eric
Wald test p-value: person Ethan vs Eric
DataFrame with 6 rows and 6 columns
                 baseMean     log2FoldChange             lfcSE              stat            pvalue              padj
                <numeric>          <numeric>         <numeric>         <numeric>         <numeric>         <numeric>
NR_046018 1.07449047357762  1.20651410025299  1.93256742982961 0.624306340689683 0.532426420584249 0.999688681823917
NR_024540 75.0618625564708 0.174444303961561 0.447858085262212  0.38950799304969 0.696900399001311 0.999688681823917
```

You do not need to re-run DESeq2 to get these other comparisons! Just use the results() command with a different contrast.

Part 2.1: Numeric contrasts

1. You can equivalently use a "numeric" vector to get the same results. This is usually easiest to understand when no control sample is indicated. If you don't want to designate a control, just specify a "0" as the intercept term.
2. View the model matrix:

```
> model.matrix(~0+person,data = metadata)
   personEli personElizabeth personEric personEthan
1          1               0          0           0
2          1               0          0           0
3          1               0          0           0
4          0               1          0           0
5          0               1          0           0
6          0               1          0           0
7          0               0          1           0
8          0               0          1           0
9          0               0          1           0
10         0               0          0           1
11         0               0          0           1
12         0               0          0           1
```

3. Notice that we now are no longer assigning each column an intercept value. We'll instead be looking at changes relative to a "0" line. In other words, none of these samples are being used as controls
4. With numeric matrices, we can easily answer unique questions. Such as "Is Ethan different than the average of Elizabeth and Eli (his parents)?"
5. Run DESeq2 with the new equation

```
> dds <- DESeqDataSetFromMatrix(countData=countdata,colData = metadata,design = ~0+person)
> dds <- DESeq(dds)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
```

6. Generate a results file. Use a regular contrast comparing Ethan and Eric.

```
> res <- results(dds,contrast = c("person","Ethan","Eric"))
> head(res)
log2 fold change (MLE): person Ethan vs Eric
Wald test p-value: person Ethan vs Eric
DataFrame with 6 rows and 6 columns
                 baseMean    log2FoldChange             lfcSE              stat            pvalue             padj
                <numeric>         <numeric>         <numeric>         <numeric>         <numeric>        <numeric>
NR_046018 1.07449047357762  1.20651472879382  1.93256761931939 0.624306604712096 0.532426247225463 0.99968867941443
NR_024540 75.0618625564708 0.174444336715847 0.447858096441416 0.389508056462401 0.696900352101449 0.99968867941443
```

7. Next, look at the model matrix. Use a numeric contrast to generate the same comparison. You'll notice each of the values in every column are the same.

```
> res <- results(dds,contrast = c(0,0,-1,1))
> head(res)
log2 fold change (MLE): 0,0,-1,+1
Wald test p-value: 0,0,-1,+1
DataFrame with 6 rows and 6 columns
                 baseMean    log2FoldChange             lfcSE              stat            pvalue             padj
                <numeric>         <numeric>         <numeric>         <numeric>         <numeric>        <numeric>
NR_046018 1.07449047357762  1.20651472879382  1.93256761931939 0.624306604712096 0.532426247225463 0.99968867941443
NR_024540 75.0618625564708 0.174444336715847 0.447858096441416 0.389508056462401 0.696900352101449 0.99968867941443
```

The values in the numeric contrast refer to the column index of the design matrix (so the first position in the numeric matrix refers to personEli, the second position is personElizabeth, and so on). The numeric contrast can be used to pick which columns from your design matrix you want to compare. You can contrast two effects by making one column negative. In effect, we are looking at the difference between Ethan and Eric (thus, we "subtract" Ethan and Eric by using a -1 in the numeric contrast vector for Ethan, and a +1 for Eric)

8. Now, use a partial value to answer the following question- what is the difference between Ethan and the average of Eli and Elizabeth (the parents)?:

```
> res <- results(dds,contrast = c(.5,.5,-1,0))
> head(res)
log2 fold change (MLE): +0.5,+0.5,-1,0
Wald test p-value: +0.5,+0.5,-1,0
DataFrame with 6 rows and 6 columns
                 baseMean   log2FoldChange             lfcSE              stat            pvalue             padj
                <numeric>        <numeric>         <numeric>         <numeric>         <numeric>        <numeric>
NR_046018 1.07449047357762 1.38662672064284  1.72919328470719 0.801892265547191  0.42261528054417 0.999777384883477
NR_024540 75.0618625564708 0.28971614323549 0.390673543039882 0.741581170255788 0.458341135579838 0.999777384883477
```

Here, the .5 means we're adding half of the effect from Eli, and half of the effect from Elizabeth, then finding the difference between that value and Ethan. This is the equivalent of taking the average effect between Eli and Elizabeth.

Alternatively, you could also run the two comparisons separately and find the average log2FC (i.e., Ethan vs Elizabeth and Ethan vs Eli).

9. Partial values can be used to average datasets or effects, or to make more specific comparisons between different groups. These can get very complex- be aware of what your design means

Part 2.2: Contrast list

There's one more way to generate these comparisons: contrast lists. In this format, the first vector contains the columns which will be added together, and the last vector contains the columns which will be subtracted. These columns must be specified exactly as they're named in the design matrix.

```
> res <- results(dds,contrast = list(c("personEthan"),c("personEric")))
> head(res)
log2 fold change (MLE): personEthan vs personEric
Wald test p-value: personEthan vs personEric
DataFrame with 6 rows and 6 columns
                 baseMean     log2FoldChange             lfcSE              stat            pvalue             padj
                <numeric>         <numeric>         <numeric>         <numeric>         <numeric>        <numeric>
NR_046018 1.07449047357762  1.20651472879382  1.93256761931939 0.624306604712096 0.532426247225463 0.99968867941443
NR_024540 75.0618625564708 0.174444336715847 0.447858096441416 0.389508056462401 0.696900352101449 0.99968867941443
```

All of these methods are equivalent, but some might be easier to understand/utilize than others depending on your type of analysis.

Using a list and specifying the column name also allows you to compare across multiple groups. This will become useful in the next section.

Part 3: Interaction Coefficients

1. Load in the counts file containing multiple treatments

```
> countdata <- read.delim("/path/to/your/filess/ploidy_treatment_counts.txt",
+                         sep="\t", header=TRUE)
```

2. Load in the metadata. Notice we have multiple columns of interest. How can you load in all of the metadata information into the design matrix?

```
> metadata <- read.csv("/path/to/your/files/ploidy_treatment.csv",
+                      sep=",",header=TRUE)
```

3. View the model matrix with an interaction term (again, for simplicity, we'll leave out batch correction):

```
> model.matrix(~treatmentIFN+ploidy+treatmentIFN:ploidy,data=metadata)
   (Intercept) treatmentIFNIFN ploidyT21 treatmentIFNIFN:ploidyT21
1            1               0         0                        0
2            1               1         0                        0
3            1               0         1                        0
4            1               1         1                        1
5            1               0         0                        0
6            1               1         0                        0
7            1               0         1                        0
8            1               1         1                        1
9            1               0         0                        0
10           1               1         0                        0
11           1               0         1                        0
12           1               1         1                        1
```

4. Which samples are used for the genotype:treatment term? All of the samples that have both ploidyT21 AND treatmentIFNIFN values of 1. In other words, samples that are both T21 and IFN-treated.

5. Run DESeq2 with the new model:

```
> rownames(countdata) <- countdata[,2]
> countdata <- countdata[,-c(1,2,3)]
> dds <- DESeqDataSetFromMatrix(countData = countdata, colData = metadata, design=~treatmentIFN+ploidy+treatmentIFN:ploidy)
> dds <- DESeq(dds)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
```

6. Find the results for the interaction term for the interferon-response gene "NM_016817"

```
> res[rownames(res)=="NM_016817",]
log2 fold change (MLE): treatmentIFNIFN.ploidyT21
Wald test p-value: treatmentIFNIFN.ploidyT21
DataFrame with 1 row and 6 columns
                baseMean     log2FoldChange             lfcSE             stat            pvalue              padj
               <numeric>          <numeric>         <numeric>        <numeric>         <numeric>         <numeric>
NM_016817 25684.954620474 0.0248278861258059 0.204549587137594 0.12137832431363 0.903391384623132 0.999656960433803
```

How do we interpret these results? Put short, the interaction term here is the treatment effect of IFN in T21 minus the treatment effect of IFN in D21. Since our log2FoldChange here is small, it means that this gene responds similarly to IFN for both T21 and D21.

7. What if we just want the condition effect in T21 (i.e., the effect of IFN in T21, not accounting for the baseline effect in D21)? We can use a contrast:

```
> resultsNames(dds)
[1] "Intercept"                  "treatmentIFN_IFN_vs_control" "ploidy_T21_vs_D21"          "treatmentIFNIFN.ploidyT21"
> res <- results(dds,contrast=list(c("treatmentIFNIFN.ploidyT21","treatmentIFN_IFN_vs_control")))
> head(res)
log2 fold change (MLE): treatmentIFNIFN.ploidyT21+treatmentIFN_IFN_vs_control effect
Wald test p-value: treatmentIFNIFN.ploidyT21+treatmentIFN_IFN_vs_control effect
DataFrame with 6 rows and 6 columns
                baseMean      log2FoldChange             lfcSE               stat            pvalue              padj
               <numeric>           <numeric>         <numeric>          <numeric>         <numeric>         <numeric>
DQ459430 326636.681168512 -0.0599061958275214 0.180211080500315  -0.332422377476488 0.739570343591367 0.974740612382292
DQ516784  21561.436724912 -0.0148325507686184 0.437217320979903 -0.0339248928550574 0.972937042982389 0.997595529456313
DQ516752 76762.5774909045 -0.0369141620560472 0.163140443516299  -0.226272291900194 0.820989647418344 0.982665571214653
DQ668364  10892.583104162  -0.100404700084239 0.168831270382741  -0.594704404324039 0.552041077278195 0.931767198778404
DQ883670 2.78972972874319  -0.851853815345591  1.11239571168187  -0.765783080966437 0.443805382897581                NA
EF011062 17.9221863913968  -0.708636792081533 0.533612088589211   -1.32799988462604 0.184178145029285 0.687137848520213
```

Remember to check your design matrix column names to make sure they match. Use the resultsNames() command to check the design matrix within the DESeq2 object itself.

Part 4 (If we make it here): Likelihood Ratio Tests, Time Series, and Heatmaps

Oftentimes we are working with data that isn't amenable to a simple pairwise comparison. Instead, we want to group the values together into a single model, and test how well each element of the model predicts the data.

1. Load in counts file containing a time series

```
> countdata <- read.delim("/path/to/local/files/timeseries_simple_example_counts.txt",
+                         sep="\t", header=TRUE)
```

2. Load in the time series metadata.

```
> metadata <- read.csv("/path/to/local/files/timeseries_simple_example_metadata.csv",
+                      sep=",",header=TRUE)
```

3. Run DESeq2, with a likelihood ratio test (LRT). Use a full model with batch and time, and a reduced model with just batch

   Note that we also have to relevel our metadata, because otherwise the times are sorted alphanumerically by default. This won't affect the test itself, but rather how we visualize the data.

```
> rownames(countdata) <- countdata[,1]
> countdata <- countdata[,-c(1)]
> metadata$time <- factor(metadata$time,levels=c("0m","60m","120m","180m","240m","300m"))
> dds <- DESeqDataSetFromMatrix(countData = countdata, colData = metadata,
+                               design=~batch+time)
> dds <- DESeq(dds,test="LRT",reduced=~batch)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
```

4. Find the results. Notice that we still have a reported log2FC. The p-value is NOT generated from this value in a LRT.

```
> res <- results(dds)
> head(res)
log2 fold change (MLE): time 300m vs 0m
LRT p-value: '~ batch + time' vs '~ batch'
DataFrame with 6 rows and 6 columns
               baseMean     log2FoldChange             lfcSE              stat            pvalue              padj
              <numeric>          <numeric>         <numeric>         <numeric>         <numeric>         <numeric>
gene1   128.55032705123  0.150491931513462 0.638783580331034 0.672661788821202 0.984435302354209 0.998986440956293
gene2  24.9641618619594  0.128802922262277 0.976156515917486  2.55687526915447 0.767906238029474                NA
gene3  21.0215737074764  0.439832915451598  1.10727665433593  6.07013344947407 0.299454335894977                NA
gene4  5.89794525268679   0.48873489790734  1.52697228424526  1.95435177736707 0.855424613331512                NA
gene5 0.876067595017242 0.0083942237645078  4.29728287912438  2.74856139065366 0.738681756452033                NA
gene6  12.3549536202008  -3.55465463865471  1.23439654483279  9.05089737232706  0.10705117334082                NA
```

So how do we interpret these results? This is similar to an analysis of variance (ANOVA) test. Here, we're asking whether the full model is significantly better at explaining the data than the reduced model (this is formally known as an analysis of deviance).

So, any significant genes were much better explained by ~batch+time than by ~batch alone. In other words, we need the time information to explain the deviance in the counts we see for that gene, suggesting that the duration of the treatment is an important element to that gene's expression level.

This is NOT the same as a fold change! The deviance in expression levels we see might not actually be significant if we were to do a pairwise Wald (log2FoldChange) test, but the model better explains the counts across all the samples. Visualize gene8151 to get a better sense of this.

```
> vsd <- vst(dds)
> boxplot(assay(vsd)['gene8151',] ~ metadata[,'time'])
```

Pop quiz: How could you tell which term is the most important for explaining the data? How about which terms are least important?

One method would be to compare the stat/p-values of each term when compared to an "empty" model, which only contains an intercept fit. To do this, use "reduced=~1". For example:

```
> dds <- DESeqDataSetFromMatrix(countData = countdata, colData = metadata,
+                               design=~time)
> dds <- DESeq(dds,test="LRT",reduced=~1)
```

Part 4.1 Heatmaps and Clustering

One common method for visualization of counts data is a heatmap, combined with clustering. Clustering is a necessary first step before an enrichment analysis (gene ontology, gene set enrichment analysis, etc.)

We will use the pheatmap package:

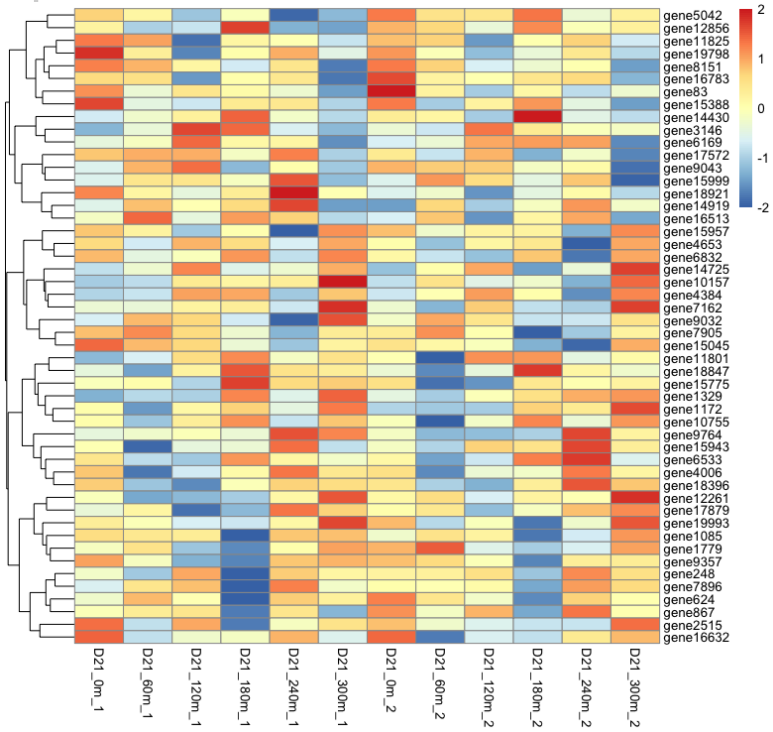
```
> install.packages("pheatmap")
> library(pheatmap)
```

First, we need to normalize our counts. This will take several steps:

```
> dds <- DESeqDataSetFromMatrix(countData = countdata, colData = metadata,
+                               design=~batch+time)
> dds <- DESeq(dds,test="LRT",reduced=~batch)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
> res <- results(dds)
> rlogcounts <- rlog(dds,blind=F)
> normcounts <- (assay(rlogcounts))
> topGenes <- head(resdata[order(resdata$padj),],50)
> normcounts <- normcounts[rownames(normcounts) %in% rownames(topGenes),]
```

We run DESeq2, as before, saving our results file. Then, we use an "rlog" transformation. This is a stabilizing transformation that makes counts more comparable to each other within and across samples. We then subset our top hits by taking the best 50 genes (lowest padj. This is to keep processing time down for the workshop- you could easily cluster on many more genes). Finally, we subset our normalized counts to only include those genes in the top 50.

```
> #Scale by Z-score (mean = 0)
> heat <- t(scale(t(normcounts)))
> # Max values to display (all heatmaps lie!)
> thr <- 2
> heat[heat < -thr] <- -thr
> heat[heat > thr] <- thr
> pheatmap(heat, breaks=seq(from=-thr, to=thr, length=101),cluster_cols = FALSE)
```



Lastly, we re-scale our counts by Z-score for each row (i.e., we set the mean of the row to zero, and each sample's value for that gene is set to its Z-score). This makes the visualization consistent across all genes and samples. (We also have to "transpose" our dataframe, so that our genes are the column values, for the scale function to work. We transpose it back once we're done. Just one example of many "R-isms").

Now for the visualization- we set a threshold value so that outliers don't blow out the heatmap. We convert all values beyond this threshold to the threshold value. This threshold value can obscure data, so be conscientious of this when you make or view heatmaps. All heatmaps lie!

We set the number of "breaks" for the color gradient in the heatmap. The more breaks, the smoother the gradient. Again, your choice of breaks can obscure the data, so be careful. We also specify to not cluster columns, as we want to preserve the order of the time series data and replicate information.

Pheatmap has built-in clustering (k-means) which can help with visualization and downstream analysis. You can specify the k as an argument in pheatmap:

```
> pheatmap(heat, breaks=seq(from=-thr, to=thr, length=101),cluster_cols = FALSE,
+          kmeans_k = 4)
```