

Differential Expression with DESeq2

Augustus Pendleton

Why is differential abundance/expression/analysis?

One of the most common questions we ask as scientists is whether a variable of interest changes between two conditions. Perhaps we're measuring growth rate between wild-type and a mutant, or protein production in two types of media. Then, we need to assess statistical significance of this relationship: is our dependent variable **significantly** different between the two conditions?

If we're measuring a single dependent variable across two conditions, we can use common statistical tests like t-tests or Wilcoxon tests. However, our methods get more complicated if we want to measure the outcome of *multiple* variables from a single experiment. If there's only a few outcome variables, we could still do t-tests, and correct our p-values for multiple comparisons. We could also consider multivariate ANOVAs (MANOVAs). But what if we want to measure the outcome of **thousands** of variables? And what if these outcomes might be dependent on each other?

This statistical challenge (assessing the significance of a treatment on hundreds to thousands of outcomes) has become increasingly prevalent as 'omics methods have expanded. Metabarcoding, RNASeq, TN-Seq, and other sequencing technologies all allow us to simultaneously measure the levels of thousands of outcomes (species abundance, gene expression, strain enrichment) over two or more conditions. What's challenging is to then determine which outcomes are statistically significant.

There are many tools which have been developed to address these issues, and many papers have been published comparing different methods (e.g. [this for RNA-Seq](#) and [this one for 16S](#)). Different tools will have different levels of sensitivity (minimizing false negatives) and specificity (minimizing false positives) and no one tool is the “best” - I encourage you to read comparisons of tools and choose one that’s best for your study design!

Today we’ll be working with RNASeq data, and we’ll use one of the most popular tools for gene expression analysis, DESeq2.

Our Data

Today, we’ll be working with previously published data looking at gene expression in two strains of *Staphylococcus aureus*. In [Waters et al., 2016](<https://onlinelibrary.wiley.com/doi/full/10.1111/mmi.13404>), they performed RNASeq, comparing gene expression in wild-type and a $\Delta codY$ mutant. CodY is a global transcriptional regulator which responds to depletions in branched-chain amino acids to modify gene expression.

There’s been a few steps which I’ve done, before we get started in this tutorial. We received sequencing data for two technical replicates of each condition. This data was cleaned and trimmed, and aligned to a reference genome (MRSA252). This produces .bam files. Then, I used the `featureCounts` command in the `subread` package to summarize reads for each feature (gene). We’ll load and work with those featureCounts today to start the lesson.

Loading packages and feature counts

First, we’ll load necessary packages for today. Next, we’ll load our featureCounts data, and explore a bit about the object.

```
library(tidyverse)
library(DESeq2)
library(ggrepel)
library(ggmagnify)
```

```
load("data/feature_counts.RData") # Load feature counts
```

```
glimpse(feature_counts) # Look at list
```

List of 4

```
$ counts      : int [1:2659, 1:4] 9849 11802 1144 7491 14959 27728 1903 244 7866 614 ...
..- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:2659] "gene-SAR0001" "gene-SAR0002" "gene-SAR0003" "gene-SAR0004" ...
.. ..$ : chr [1:4] "codY_R61K_1.bam" "codY_R61K_2.bam" "wt_1.bam" "wt_2.bam"
$ annotation:'data.frame': 2659 obs. of  7 variables:
..$ GeneID: chr [1:2659] "gene-SAR0001" "gene-SAR0002" "gene-SAR0003" "gene-SAR0004" ...
..$ Chr   : chr [1:2659] "BX571856.1" "BX571856.1" "BX571856.1" "BX571856.1" ...
..$ Start : chr [1:2659] "517" "2156" "3670" "3912" ...
..$ End   : chr [1:2659] "1878" "3289" "3915" "5024" ...
..$ Strand: chr [1:2659] "+" "+" "+" "+" ...
..$ Length: int [1:2659] 1362 1134 246 1113 1932 2661 831 1515 1287 696 ...
..$ Name  : chr [1:2659] "dnaA" "dnaN" "SAR0003" "recF" ...
$ targets  : chr [1:4] "codY_R61K_1.bam" "codY_R61K_2.bam" "wt_1.bam" "wt_2.bam"
$ stat      :'data.frame': 14 obs. of  5 variables:
..$ Status      : chr [1:14] "Assigned" "Unassigned_Unmapped" "Unassigned_Read_Type" "Un
..$ codY_R61K_1.bam: int [1:14] 15274105 93722 0 0 0 0 0 0 0 0 ...
..$ codY_R61K_2.bam: int [1:14] 14431023 76346 0 0 0 0 0 0 0 0 ...
..$ wt_1.bam      : int [1:14] 18335478 128682 0 0 0 0 0 0 0 0 ...
..$ wt_2.bam      : int [1:14] 17360453 98704 0 0 0 0 0 0 0 0 ...
```

The feature counts output is a list with four slots. The `counts` slot is the most important to us, as it contains the number of reads mapped to each feature (gene) in each bam file (condition). Another useful slot is `stat`, which shows how many reads were mapped for each replicate - this could be important if on condition or replicate had a very different number of

assigned reads!

Next, to run DESeq2, we'll need to extract the count data from our `feature_counts` object and create a `DESeqDataSet`.

```
just_counts <- feature_counts$counts[rowSums(feature_counts$counts) > 1000,]  
  
head(just_counts)
```

	codY_R61K_1.bam	codY_R61K_2.bam	wt_1.bam	wt_2.bam
gene-SAR0001	9849	9508	11457	10491
gene-SAR0002	11802	11294	14292	13737
gene-SAR0003	1144	1014	1503	1336
gene-SAR0004	7491	7356	8520	8190
gene-SAR0005	14959	14443	17735	16690
gene-SAR0006	27728	24750	32210	30508

```
strain_info <- data.frame(file = c("codY_R61K_1.bam",  
                                   "codY_R61K_2.bam",  
                                   "wt_1.bam",  
                                   "wt_2.bam"),  
                          strain = c("codY",  
                                     "codY",  
                                     "wt",  
                                     "wt"))  
  
deseq_ds <- DESeqDataSetFromMatrix(countData = just_counts,  
                                   colData = strain_info,  
                                   design = ~strain)
```

Next, we'll run DESeq2 to test for differential gene expression.

```
deseq_object <- DESeq(deseq_ds)
```

```
glimpse(deseq_object)
```

Formal class 'DESeqDataSet' [package "DESeq2"] with 8 slots

```
..@ design          :Class 'formula' language ~strain
.. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
..@ dispersionFunction:function (q)
.. ..- attr(*, "coefficients")= Named num [1:2] 0.00177 3.6679
.. ..- attr(*, "names")= chr [1:2] "asymptDisp" "extraPois"
.. ..- attr(*, "fitType")= chr "parametric"
.. ..- attr(*, "varLogDispEsts")= num 1.09
.. ..- attr(*, "dispPriorVar")= num 0.521
..@ rowRanges       :Formal class 'CompressedGRangesList' [package "GenomicRanges"] with
..@ colData         :Formal class 'DFrame' [package "S4Vectors"] with 6 slots
..@ assays          :Formal class 'SimpleAssays' [package "SummarizedExperiment"] with 1
..@ NAMES           : NULL
..@ elementMetadata :Formal class 'DFrame' [package "S4Vectors"] with 6 slots
..@ metadata        :List of 1
.. ..$ version:Classes 'package_version', 'numeric_version' hidden list of 1
..$ betaPrior       : logi FALSE
..$ modelMatrixType: chr "standard"
..$ betaPriorVar    : num [1:2] 1e+06 1e+06
..$ modelMatrix     : num [1:4, 1:2] 1 1 1 1 0 0 1 1
.. ..- attr(*, "dimnames")=List of 2
.. ..- attr(*, "assign")= int [1:2] 0 1
.. ..- attr(*, "contrasts")=List of 1
..$ test            : chr "Wald"
..$ dispModelMatrix: num [1:4, 1:2] 1 1 1 1 0 0 1 1
```

```
.. ..- attr(*, "dimnames")=List of 2
.. ..- attr(*, "assign")= int [1:2] 0 1
.. ..- attr(*, "contrasts")=List of 1
```

The results of the DESeq command are a complex S4 object with lots of info about how the model was run. To access information that might be more relevant to us, we'll use a separate command, `results`

```
deseq_results <- results(deseq_object) %>%
  as.data.frame() %>% # Convert to dataframe
  mutate(gene = row.names(.)) # Add new column of gene name

head(deseq_results)
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
gene-SAR0001	10296.422	-0.06003818	0.05879963	-1.0210639	0.3072242	0.5972292
gene-SAR0002	12710.686	0.03875767	0.05280028	0.7340429	0.4629226	0.7425669
gene-SAR0003	1237.270	0.15514202	0.10084554	1.5384123	0.1239478	0.3838753
gene-SAR0004	7871.376	-0.07031979	0.05700853	-1.2334960	0.2173908	0.5062019
gene-SAR0005	15891.432	-0.01337841	0.05279550	-0.2534005	0.7999587	0.9234335
gene-SAR0006	28646.292	0.01743278	0.06035557	0.2888346	0.7727079	0.9143417

gene
gene-SAR0001
gene-SAR0002
gene-SAR0003
gene-SAR0004
gene-SAR0005
gene-SAR0006

This gives us a data frame containing two very important pieces of information: the `log2FoldChange` (how much that gene is enriched in one condition vs. the other) and `padj`

- the adjusted p-value. We'll use this information to make our volcano plot. One annoying thing we might notice is that our genes are annotated by the GeneID - a unique identifier for each gene in this genome. However, we often want to the gene name itself. We'll access that info from our `feature_counts` object and join it to our DESeq results.

```

annotations <- feature_counts$annotation %>%
  select(GeneID, Name)

clean_deseq_results <- deseq_results %>%
  left_join(annotations,
            by = c("gene" = "GeneID")) %>%
  filter(Name != "codY")

head(clean_deseq_results)

```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
1	10296.422	-0.06003818	0.05879963	-1.0210639	0.3072242	0.5972292
2	12710.686	0.03875767	0.05280028	0.7340429	0.4629226	0.7425669
3	1237.270	0.15514202	0.10084554	1.5384123	0.1239478	0.3838753
4	7871.376	-0.07031979	0.05700853	-1.2334960	0.2173908	0.5062019
5	15891.432	-0.01337841	0.05279550	-0.2534005	0.7999587	0.9234335
6	28646.292	0.01743278	0.06035557	0.2888346	0.7727079	0.9143417

	gene	Name
1	gene-SAR0001	dnaA
2	gene-SAR0002	dnaN
3	gene-SAR0003	SAR0003
4	gene-SAR0004	recF
5	gene-SAR0005	gyrB
6	gene-SAR0006	gyrA

Plotting our DESeq Results

```
sig <- clean_deseq_results %>%  
  filter(padj < 0.05, abs(log2FoldChange) > 1)
```

Making our Volcano Plot

```
start_plot <- ggplot(data = clean_deseq_results,  
  aes(x = log2FoldChange,  
      y = -log10(padj+1e-20),  
      color = padj<0.05&abs(log2FoldChange)>1)) +  
  geom_point(position = position_jitter(width = 0, height = .5)) +  
  coord_cartesian(xlim = c(-5, 5),  
    ylim = c(-2, 30)) +  
  scale_color_manual(values = c("grey90", "black")) +  
  labs(x = bquote(log[2](Fold~Change)),  
    y = bquote(-log[10](Adjusted~p~value))) +  
  theme_classic() +  
  theme(legend.position = "none") +  
  ggrepel::geom_text_repel(data = sig,  
    position = position_jitter(width = 0, height = 0.5),  
    aes(label = Name),  
    point.padding = 2,  
    seed = 10,  
    size = 2,  
    max.overlaps = 12)  
  
final_plot <- start_plot +  
  ggmagnify::geom_magnify(aes(from = log2FoldChange < -2.5 & padj < 1e-20),
```



```

                                to = c(-5,-2,22,30)) +
annotate(geom = "segment",
          x = c(-.2,.2),
          xend = c(-2,2),
          y = -1,
          yend = -1,
          arrow = arrow(type = "closed",
                        length = unit(0.02, "npc")))) +
annotate(geom = "text",
          x = c(-2,2),
          y = -1,
          label = c("High~'in'~Delta*italic(codY)",
                    "High~'in'~WT"),
          parse = TRUE,
          hjust = c(1.1,-.1))

ggsave(final_plot,
        filename = "final_volcano.png",
        height = 5, width = 5, units = "in")

```

