Day 7 - Introduction to Gene Differential Expression Analysis using DESeq2

Authors:

Jacob Stanley (jacob.stanley@colorado.edu)

Edited and updated by:
Daniel Ramirez (daniel.ramirezhernandez@colorado.edu) - 2022
Rutendo Sigauke (rutendo.sigauke@colorado.edu) – 2023
Samuel Hunter (sahu0957@colorado.edu) - 2024

Additional DESeq2 resources:

https://bioconductor.org/packages/release/bioc/html/DESeq2.html
https://bioconductor.org/packages/release/bioc/vignettes/DESeq2/inst/doc/DESeq2.html

Introduction:

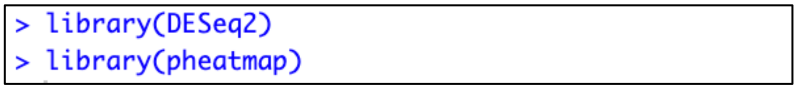
Here we'll use those gene counts tables as input for the software DESeq2 to answer the question: What genes are statistically significantly changed upon an experimental condition? In particular, we will explore a dataset from a real experiment published (Andrysik 2017 et al. doi: 10.1101/gr.220533.117). You will use a gene count table that we already prepared for you, from an experiment where Human colon cancer cells (HCT116) were treated with either the vehicle DMSO, or with the p53-activator drug Nutlin.

The purpose of DESeq2 is to identify which genomic loci demonstrate a statistically significant difference in expression level between two or more conditions (referred to as "gene differential expression analysis"). It does so by modeling the variance in expression level across the full range of baseline expression levels present in the data and determines if the differential expression level for each locus is significantly greater than this variance. DESeq2 takes as an input the unnormalized count values for each (non-overlapping) loci in each sample. We recommend that you use featureCounts() to compute count values. DESeq2 performs best when provided multiple replicates per experimental condition (preferably 5+ replicates), in order to get an accurate estimation of within condition variance. DESeq2 is only to be used for non-overlapping, unique genomic loci. If one's aim is to compute differential expression of transcripts, DESeq2 is not appropriate.

Note: All commands are executed within the R environment. We will be executing them manually, from the R command line, but they can also be compiled into a single script to be executed together.

--- FILES ARE IN THE DATA FOLDER ON GITHUB FOR DAY7---

We will use DESeq2 with RStudio on your local computer. First, we need to tell RStudio to load the DESeq2 and pheatmap libraries (if you have not yet installed DESeq2, let a class helper know. Be aware though that *installing DESeq2 takes some time* as it also needs several dependencies installed as well).

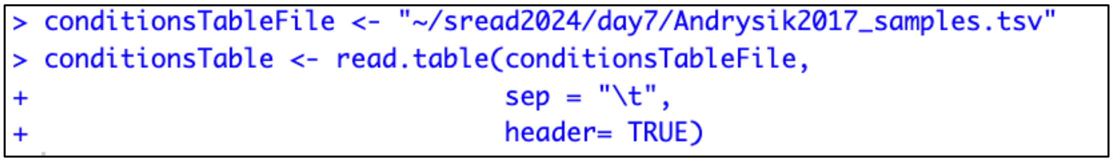


```
> library(DESeq2)
> library(pheatmap)
```

Below are the featureCounts output files as well as the condition table we will be using. Get these files onto your own local machine from GitHub. We will use your local machine installation of RStudio and use these files as inputs.

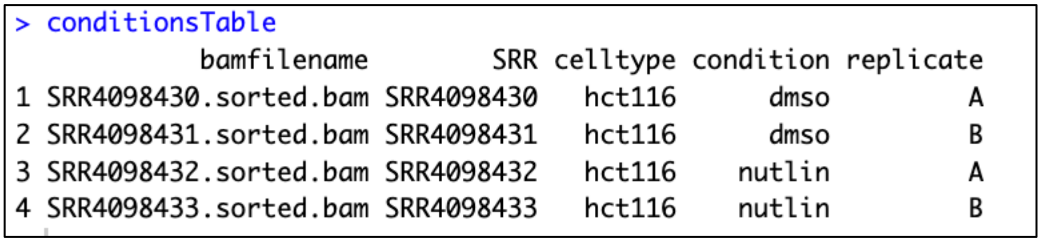
srworkshop/projectB/day07/data/
Andrysik2017_counts.tsv
Andrysik2017_samples.tsv

Read in these files using **read.table().** Replace the path with your own file path to these files from GitHub Let's start with the Conditions Table:



```
> conditionsTableFile <- "~/sread2024/day7/Andrysik2017_samples.tsv"
> conditionsTable <- read.table(conditionsTableFile,
+                               sep = "\t",
+                               header= TRUE)
```

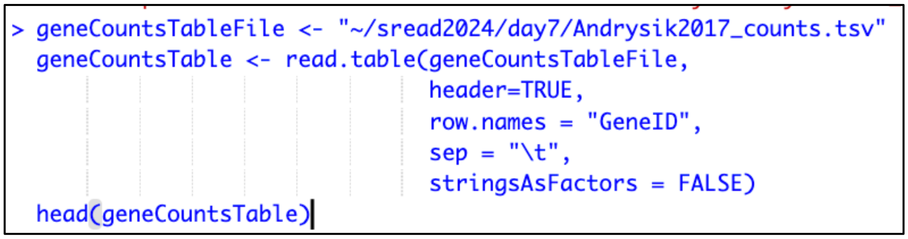
This file is a tab-separated data frame containing experimental information. Let's explore its contents:



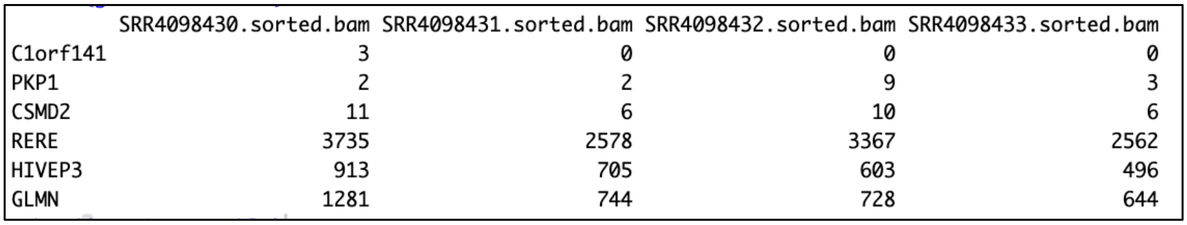
```
> conditionsTable
             bamfilename        SRR celltype condition replicate
1 SRR4098430.sorted.bam SRR4098430   hct116      dmso         A
2 SRR4098431.sorted.bam SRR4098431   hct116      dmso         B
3 SRR4098432.sorted.bam SRR4098432   hct116    nutlin         A
4 SRR4098433.sorted.bam SRR4098433   hct116    nutlin         B
```

Here, we've recorded information about each sample, namely, **bamfilename, SRR, celltype, condition,** and **replicate**. Note that this is a file that we write ourselves describing design elements of our experiment. We could even write in more columns with more experimental details we might want to test (e.g., genotype, dosage, treatment time, etc.).

Next, load in the Counts Table and print the first few lines:



```
> geneCountsTableFile <- "~/sread2024/day7/Andrysik2017_counts.tsv"
  geneCountsTable <- read.table(geneCountsTableFile,
                                header=TRUE,
                                row.names = "GeneID",
                                sep = "\t",
                                stringsAsFactors = FALSE)
  head(geneCountsTable)
```



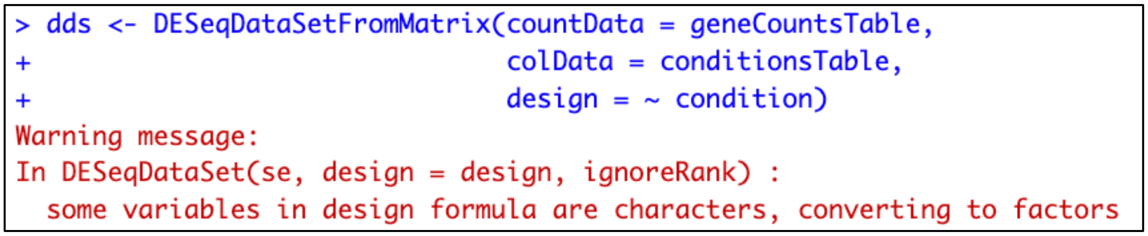
```
        SRR4098430.sorted.bam SRR4098431.sorted.bam SRR4098432.sorted.bam SRR4098433.sorted.bam
C1orf141                    3                     0                     0                     0
PKP1                        2                     2                     9                     3
CSMD2                      11                     6                    10                     6
RERE                     3735                  2578                  3367                  2562
HIVEP3                    913                   705                   603                   496
GLMN                     1281                   744                   728                   644
```

Here, we have counts information generated for each gene. The sample columns of this Counts Table **must** be in the same order as the rows of our Conditions Table. Note also that we've stored the gene names as rownames in the Counts Table. The Counts Table must only consist of numeric entries, otherwise DESeq2 will fail to run.

--- USING DESEQ2 WITHIN RSTUDIO ---

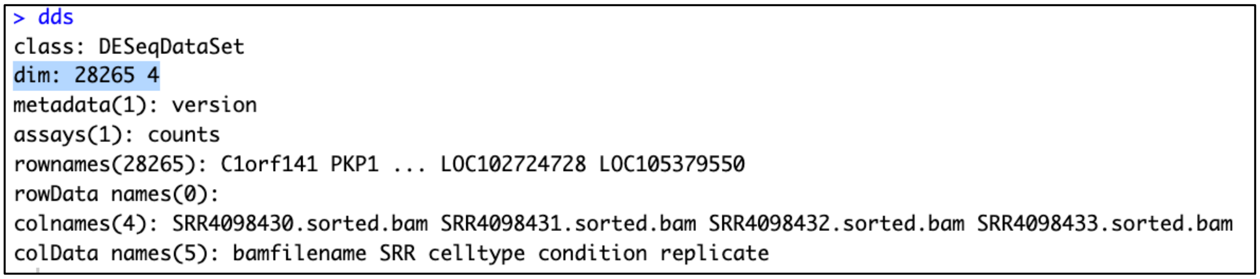
Next, load the two inputs onto DESeq2 using the **DESeqDataSetFromMatrix** function. The parameters should be as follows:

countData should point to our Counts Table object
colData should point to our Conditions Table object
design is a formula describing the experimental elements you want to test. It can only contain column names from your Conditions Table. In this case, we want to test whether there is a significant difference in gene expression between Nutlin- and DMSO-treated samples. The **condition** column of our Conditions Table specifies this information, so we tell DESeq2 to focus only on this column.



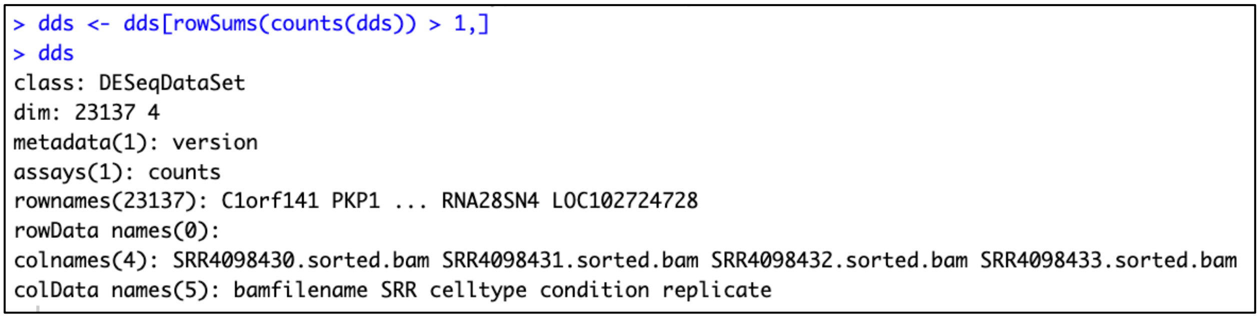
```
> dds <- DESeqDataSetFromMatrix(countData = geneCountsTable,
+                               colData = conditionsTable,
+                               design = ~ condition)
Warning message:
In DESeqDataSet(se, design = design, ignoreRank) :
  some variables in design formula are characters, converting to factors
```

The new **dds** variable is a special DESeq2 object containing all of the information from our Counts Table and Conditions Table. You can print a description of its contents:



```
> dds
class: DESeqDataSet
dim: 28265 4
metadata(1): version
assays(1): counts
rownames(28265): C1orf141 PKP1 ... LOC102724728 LOC105379550
rowData names(0):
colnames(4): SRR4098430.sorted.bam SRR4098431.sorted.bam SRR4098432.sorted.bam SRR4098433.sorted.bam
colData names(5): bamfilename SRR celltype condition replicate
```

The **dim** value tells you how many genes (rows) and samples (columns) are being tested. Later, DESeq2 will automatically filter genes with low average counts as part of its pipeline. You can also specify your own threshold:

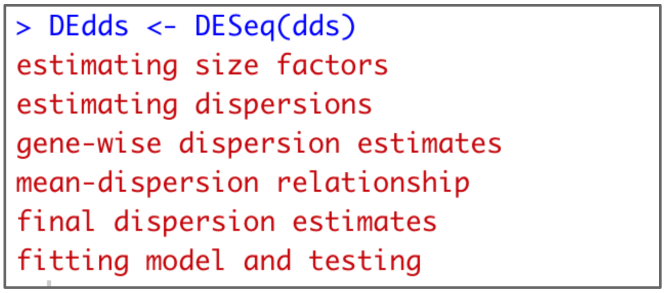


```
> dds <- dds[rowSums(counts(dds)) > 1,]
> dds
class: DESeqDataSet
dim: 23137 4
metadata(1): version
assays(1): counts
rownames(23137): C1orf141 PKP1 ... RNA28SN4 LOC102724728
rowData names(0):
colnames(4): SRR4098430.sorted.bam SRR4098431.sorted.bam SRR4098432.sorted.bam SRR4098433.sorted.bam
colData names(5): bamfilename SRR celltype condition replicate
```

Here, we require that the sum of counts for each gene across all 4 samples is greater than 1. Notice that printing the **dds** variable again gives us a smaller number of genes (e.g. 23,137).

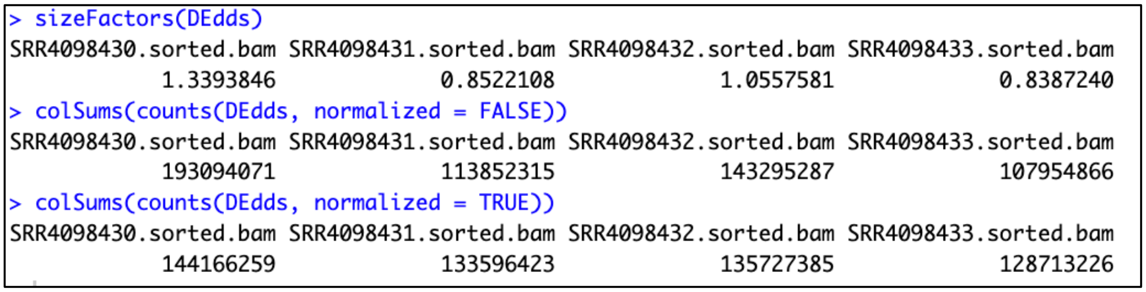
Run DESeq2's main function **DESeq** on the **dds** variable you created. DESeq2 will internally do several actions:

a. it will estimate size factors for normalizing each dataset
b. it will estimate dispersion for variance calculation,
c. it will fit a generalized linear model to calculate fold-changes
d. it will perform a statistical test for each gene



```
> DEdds <- DESeq(dds)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
```

Let's first check how well the normalization worked. We first check the **sizeFactors**, which are normalizing factors to account for depth and composition differences between the libraries. We can also check the total number of counts for each sample to see how the normalization performed using **colSums()**.



```
> sizeFactors(DEdds)
SRR4098430.sorted.bam SRR4098431.sorted.bam SRR4098432.sorted.bam SRR4098433.sorted.bam
            1.3393846             0.8522108             1.0557581             0.8387240
> colSums(counts(DEdds, normalized = FALSE))
SRR4098430.sorted.bam SRR4098431.sorted.bam SRR4098432.sorted.bam SRR4098433.sorted.bam
            193094071             113852315             143295287             107954866
> colSums(counts(DEdds, normalized = TRUE))
SRR4098430.sorted.bam SRR4098431.sorted.bam SRR4098432.sorted.bam SRR4098433.sorted.bam
            144166259             133596423             135727385             128713226
```

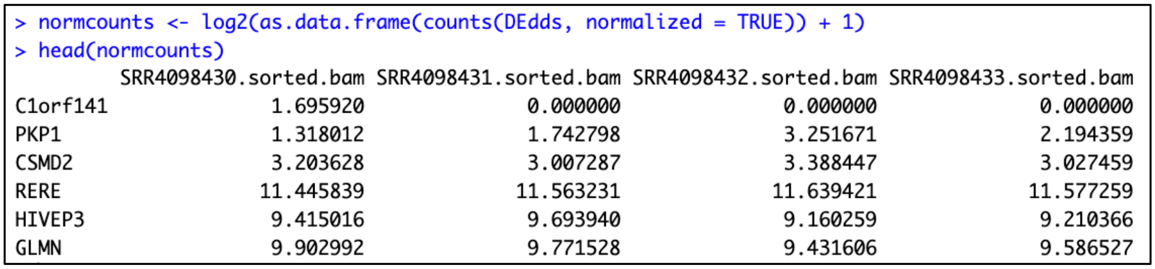
Notice that after setting **normalized** to TRUE, the sample counts are much more homogenous (e.g. between 12 and 14 million counts).

We are now ready to start exploring our data!

Exploratory Data Analysis

Prior to examining the differential analysis results, it's always a good idea to look at features of the data itself. This is called Exploratory Data Analysis, or EDA. EDA is vital for ensuring your data were generated correctly, and can often reveal interesting biology prior to differential analysis.

Start by storing our normalized counts in a new data frame:

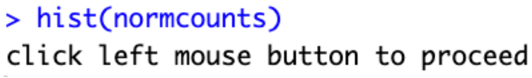


```
> normcounts <- log2(as.data.frame(counts(DEdds, normalized = TRUE)) + 1)
> head(normcounts)
         SRR4098430.sorted.bam SRR4098431.sorted.bam SRR4098432.sorted.bam SRR4098433.sorted.bam
C1orf141              1.695920              0.000000              0.000000              0.000000
PKP1                  1.318012              1.742798              3.251671              2.194359
CSMD2                 3.203628              3.007287              3.388447              3.027459
RERE                 11.445839             11.563231             11.639421             11.577259
HIVEP3                9.415016              9.693940              9.160259              9.210366
GLMN                  9.902992              9.771528              9.431606              9.586527
```

Because gene counts data often have an extremely wide range, we also rescale our counts to a log2-scale. And, because log-scaling can't handle anything with a value of 0, we also add a pseudocount of 1 to all entries.

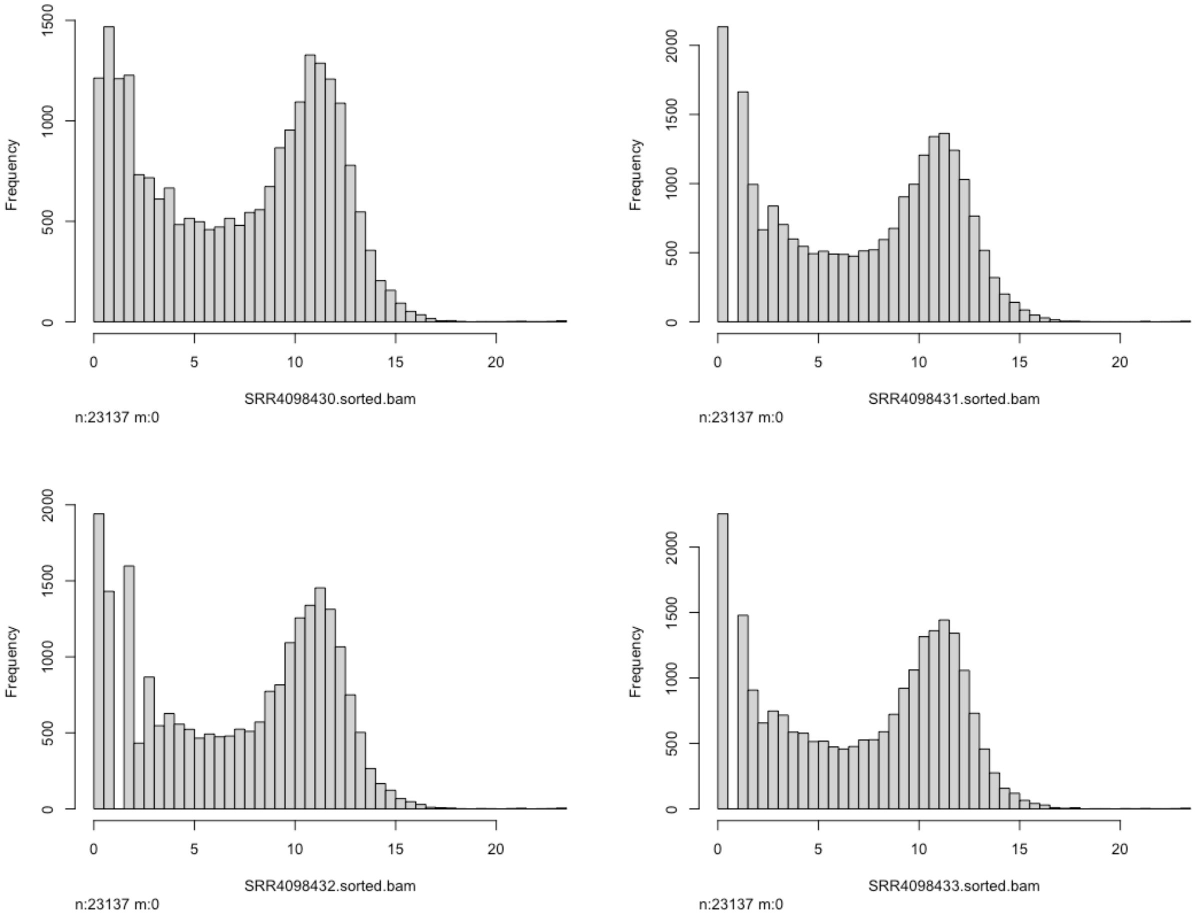
Histograms

Let's generate histograms for each sample:



```
> hist(normcounts)
click left mouse button to proceed
```

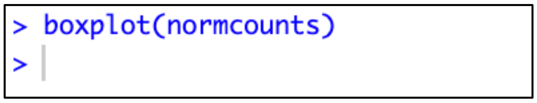
Click on the image to return to the console. The histograms should be printed in the RStudio viewer:



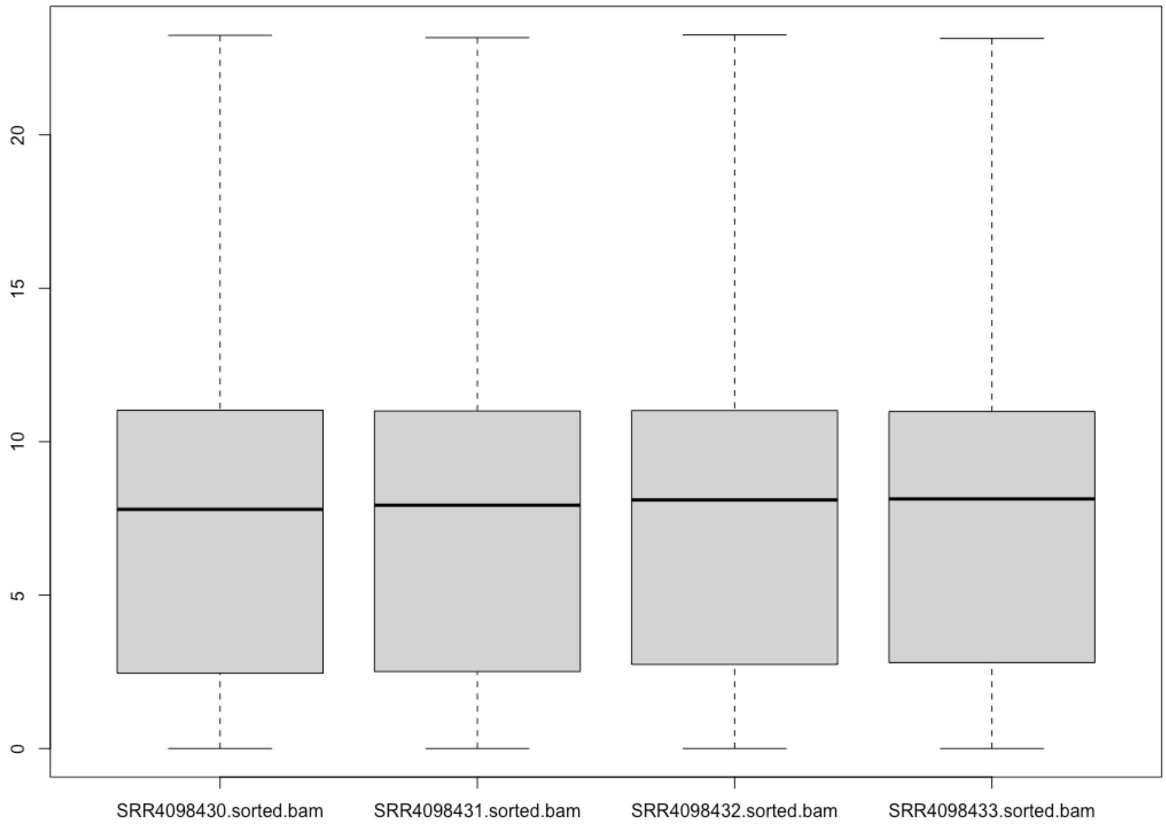
We can see that even after our filtering, each sample has many very low-expression genes, demonstrated by the large peaks on the left side of each graph. Furthermore, we can see another peak occurring around 10-12 (remember, this is log2-scaled) for each sample. We could use this plot to inform future read count thresholds if we want to refine our analysis.

Boxplots

In histogram format, it's difficult to see whether there are large read distribution differences between the samples. We can instead compress the information from the histograms into a boxplot:



```
> boxplot(normcounts)
> 
```



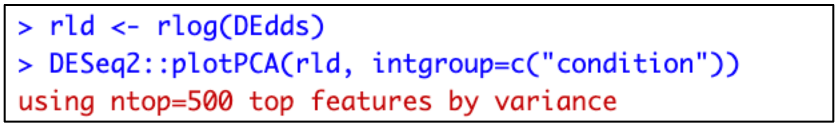
Here we see that the counts data are extremely similar among all four samples. This is in line with DESeq2's baseline assumptions. If there were radically different read distributions between the replicates, for example, we should question whether our replication was good or if our normalization was successful.

We can also see potential outliers in our data. DESeq2 will automatically apply outlier detection so these can also be filtered later if we deem it necessary.

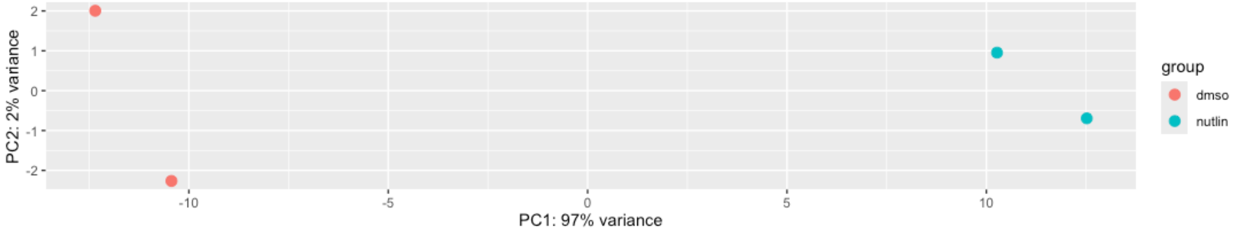
PCA

While the overall distributions of our samples are very similar, it's possible there are many differences on a gene-by-gene basis. Hopefully, these differences are a result of our experimental intervention, and not just a random result or a "batch" effect from our experimental setup. We can explore this possibility with Principal Component Analysis, or PCA.

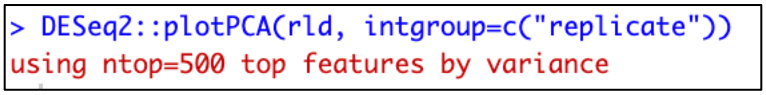
PCA can be used to graphically compress and represent genome-wide differences between samples as a 2-D plot. We can then see whether these differences line up with our experimental setup. First, we apply a specialized transformation (called rlog, or regularized log-transformation) to our DESeq2 object, then generate the PCA plot:



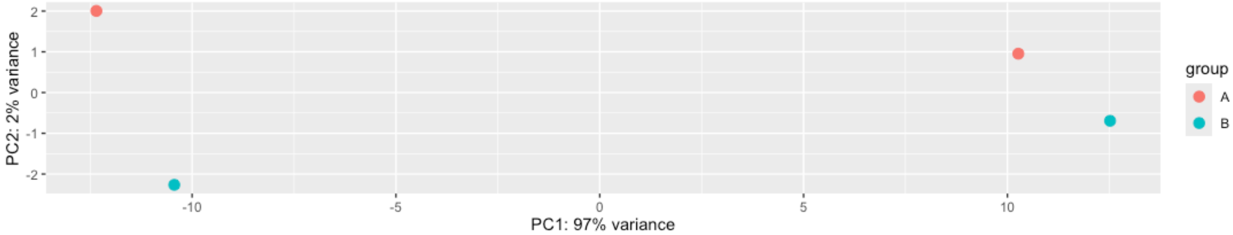
```
> rld <- rlog(DEdds)
> DESeq2::plotPCA(rld, intgroup=c("condition"))
using ntop=500 top features by variance
```



DESeq2 automatically uses the top 500 genes which show the most variance between the samples for PCA. We use the **intgroup** parameter to colorize the samples by their values in the **condition** column from our Conditions Table. We can see that the Nutlin-treated samples group together away from the DMSO treated samples. This separation occurs along the x-axis (PC1). Let's color the samples by replicate instead:



```
> DESeq2::plotPCA(rld, intgroup=c("replicate"))
using ntop=500 top features by variance
```



Notice that the colors representing **replicate** groupings are now separated along the y-axis (PC2). This is potentially indicative of a "batch" effect, a technical error associated with non-biological factors (e.g., if two libraries were prepped on different days or in different labs). Let's explore this further.

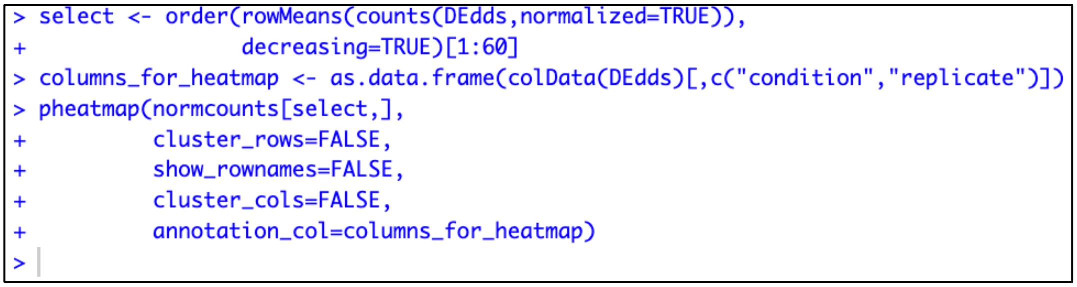
PCA provides a percent value for the variance represented along each axis. PC1 explains a very large amount of variance between the samples, at 97% (likely because our experiment caused many distinct effects between our sample groups). PC2 only explains 2% of the variance, indicating whatever batch effect might be there, it's very small compared to the variance from our Nutlin treatment. Because it's so small, we don't need to worry about a batch effect getting in the way of our conclusions about our Nutlin-treated samples, and can safely ignore it.

Sometimes, batch effects may be so severe that they show even stronger effects than the experimental treatment! Visit the reference links on page 1 to find out more info on how to combat batch effects.

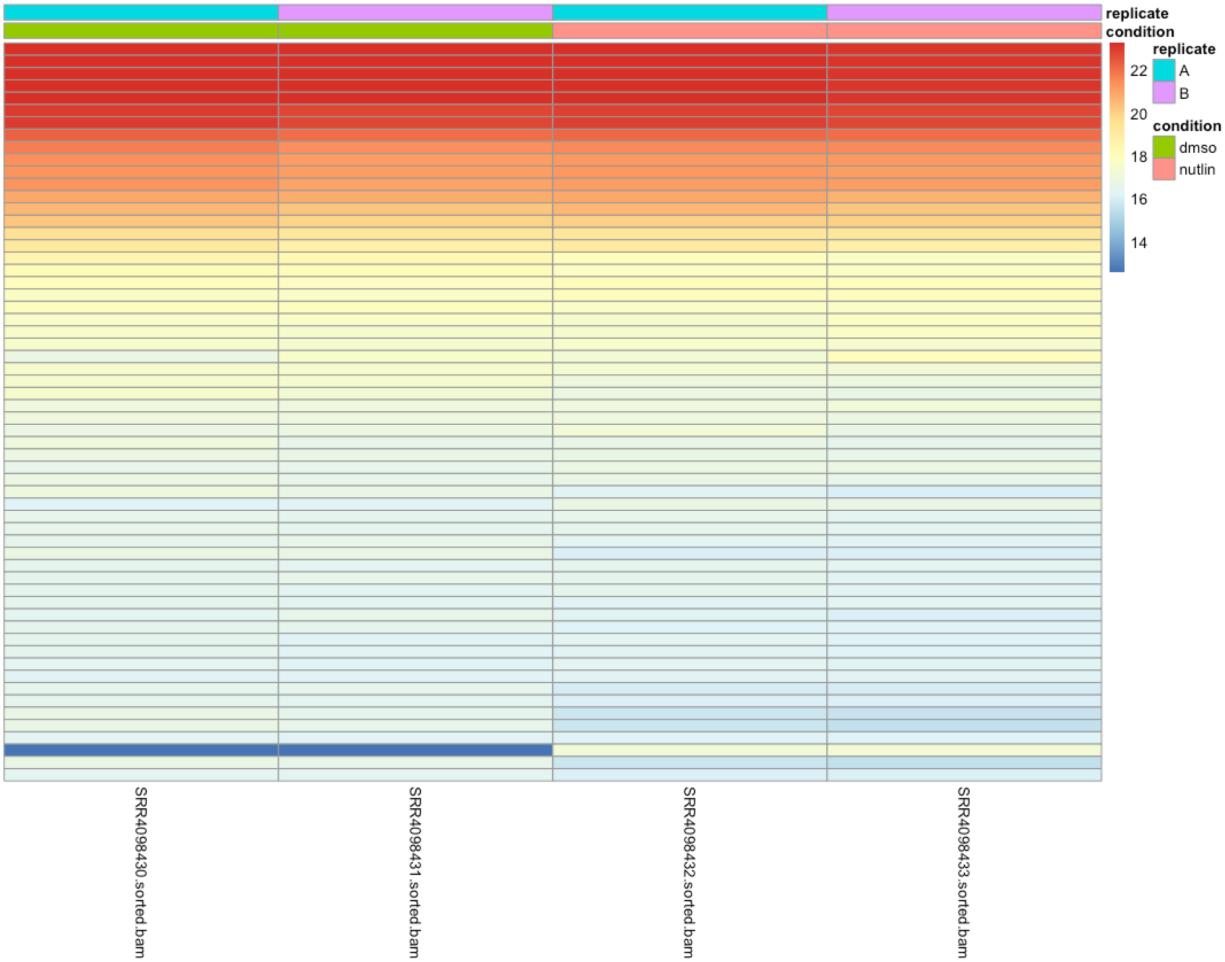
Note that our PCA plot only examines variance as a percentage explained along each axis. It should NOT be used to determine whether two samples' gene counts are truly significantly different- that requires statistics and differential analysis. Very similar samples could still show large percentage values for PC1 and PC2, but show little to no statistical differences.

Heatmaps

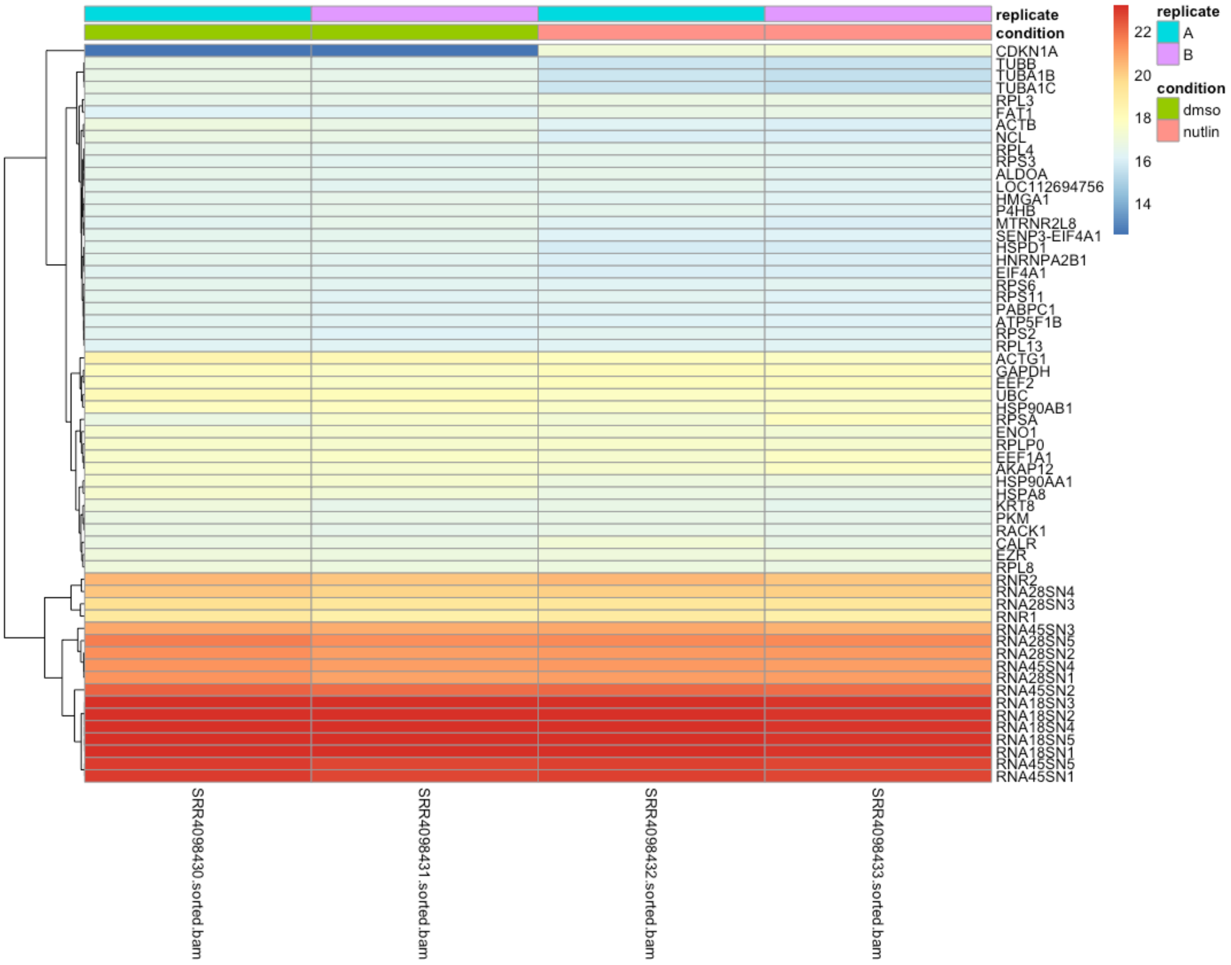
Finally, heatmaps can be used to visualize patterns between samples across genes, and for clustering genes together with similar expression patterns. Let's focus on just those genes with the highest average expression across all samples:



```
> select <- order(rowMeans(counts(DEdds,normalized=TRUE)),
+                 decreasing=TRUE)[1:60]
> columns_for_heatmap <- as.data.frame(colData(DEdds)[,c("condition","replicate")])
> pheatmap(normcounts[select,],
+          cluster_rows=FALSE,
+          show_rownames=FALSE,
+          cluster_cols=FALSE,
+          annotation_col=columns_for_heatmap)
>
```



Most genes look very similar across all data sets, but there is one gene near the bottom that seems very different. Let's cluster the results using **cluster_rows = TRUE** and show the gene names using **show_rownames=TRUE:**

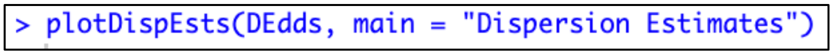


Now, the order is determined by how the genes are clustered. The top gene shows the most unique profile, and to our luck, that gene is CDKN1A. This gene is known to be upregulated upon p53 activation.

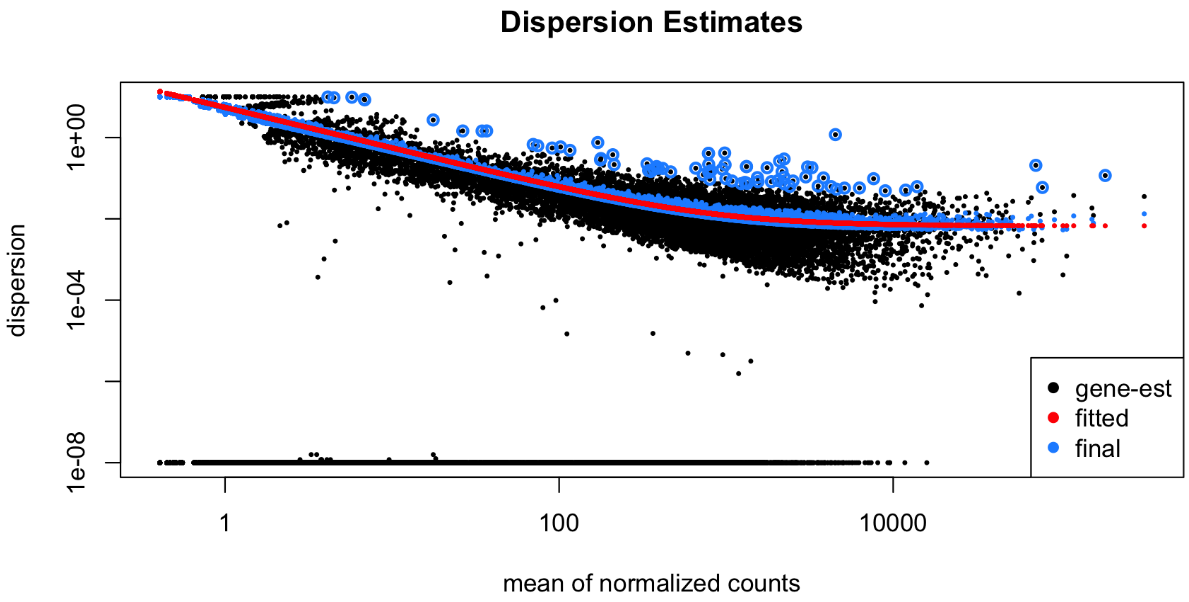
Notice that our potential outliers from the boxplot are all shown near the bottom of the graph here. These are all rRNA genes, which often make up a large proportion in RNA-seq libraries depending on the quality of the library prep. For genes which are subject to technical variation like this, we could filter these out from our Counts Table prior to analysis.

Dispersion Plotting

Finally, before examining our results, we should examine the dispersion estimation for our data.



```
> plotDispEsts(DEdds, main = "Dispersion Estimates")
```

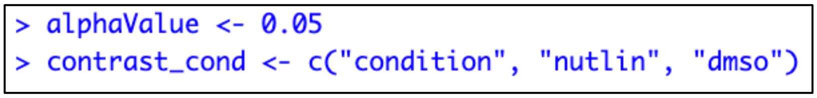


The math behind this plot is outside the scope of this workshop, but your data should show estimates that are monotonically descending, and most data points (blue) that are nearby the fitted line (red).

Differential Analysis

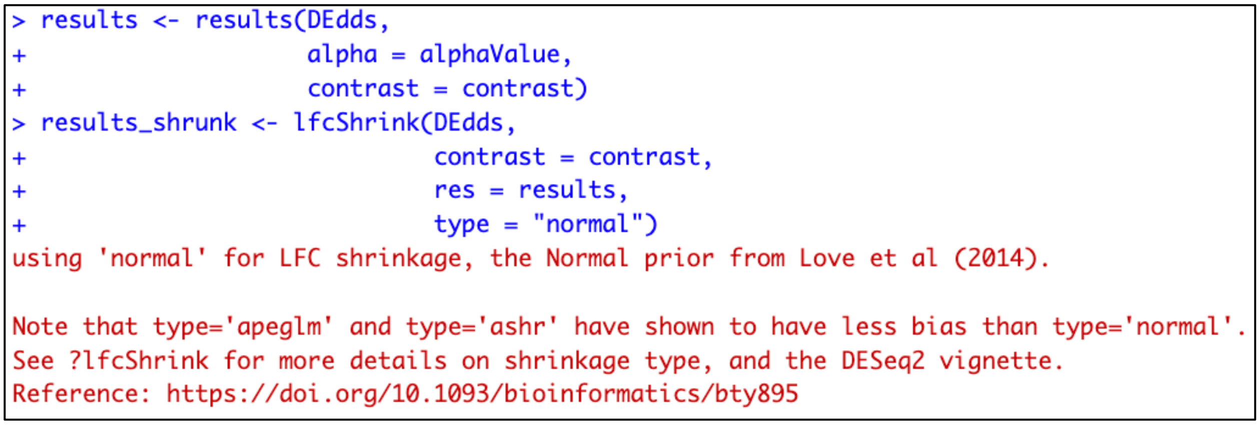
Everything we've done so far requires no statistical analysis, but to truly determine whether there are significant gene expression differences in our samples, we must apply DESeq2's differential analysis framework.

First, define the alpha value that DESeq2 will need to assign statistical significance, as well as the comparison of the two experimental conditions we want to compare against each other. This is called a "contrast":



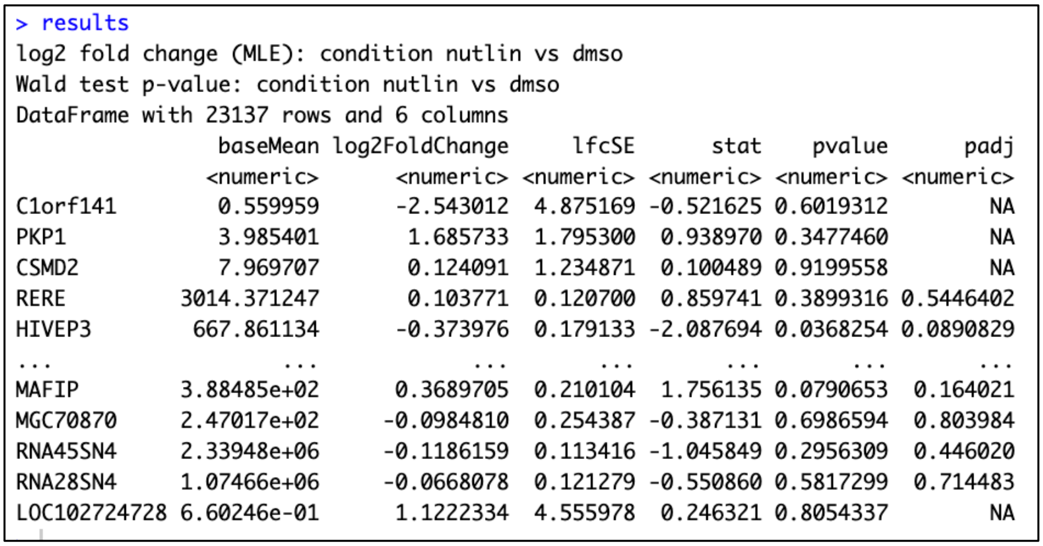
```
> alphaValue <- 0.05
> contrast_cond <- c("condition", "nutlin", "dmso")
```

Let's extract statistically significant results, and do DESeq2 special log fold-change shrinkage, which is useful for visualization purposes (DESeq2 encourages a few other algorithms for shrinkage, but we'll just stick with the "normal" option here)

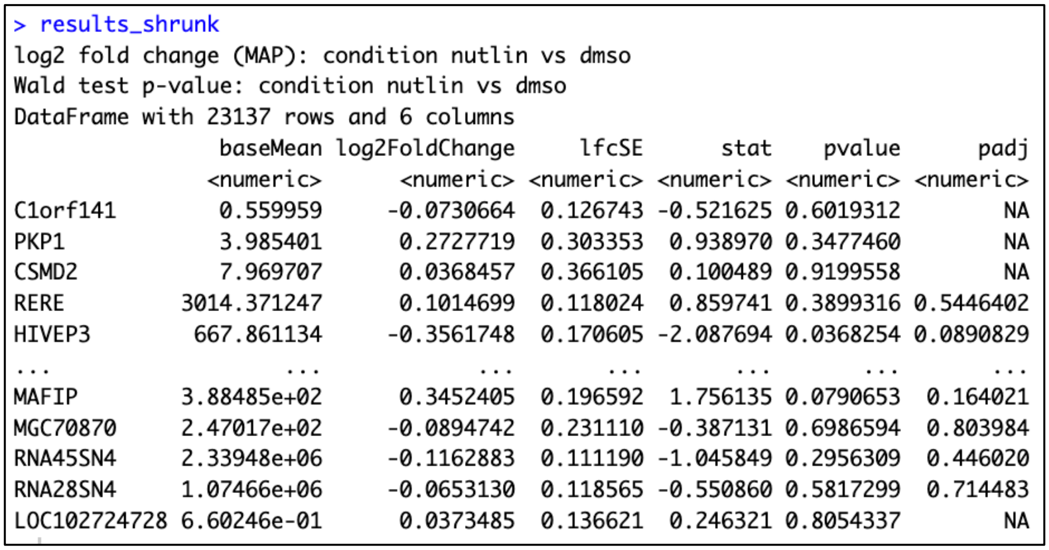


```
> results <- results(DEdds,
+                    alpha = alphaValue,
+                    contrast = contrast)
> results_shrunk <- lfcShrink(DEdds,
+                             contrast = contrast,
+                             res = results,
+                             type = "normal")
using 'normal' for LFC shrinkage, the Normal prior from Love et al (2014).

Note that type='apeglm' and type='ashr' have shown to have less bias than type='normal'.
See ?lfcShrink for more details on shrinkage type, and the DESeq2 vignette.
Reference: https://doi.org/10.1093/bioinformatics/bty895
```

Print the results. See that both unshrunk and shrunk results have identical nominal and adjusted p-values. The shrinkage only affects the fold-change estimation values. This keeps uncertain fold-changes for low-expression genes near zero.

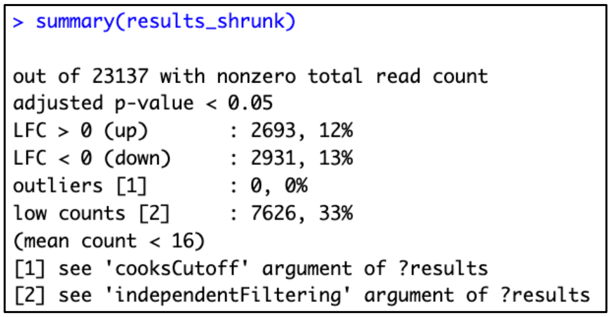


```
> results
log2 fold change (MLE): condition nutlin vs dmso
Wald test p-value: condition nutlin vs dmso
DataFrame with 23137 rows and 6 columns
                baseMean log2FoldChange     lfcSE      stat    pvalue      padj
               <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
C1orf141        0.559959      -2.543012  4.875169 -0.521625 0.6019312        NA
PKP1            3.985401       1.685733  1.795300  0.938970 0.3477460        NA
CSMD2           7.969707       0.124091  1.234871  0.100489 0.9199558        NA
RERE         3014.371247       0.103771  0.120700  0.859741 0.3899316 0.5446402
HIVEP3        667.861134      -0.373976  0.179133 -2.087694 0.0368254 0.0890829
...                  ...            ...       ...       ...       ...       ...
MAFIP        3.88485e+02      0.3689705  0.210104  1.756135 0.0790653  0.164021
MGC70870     2.47017e+02     -0.0984810  0.254387 -0.387131 0.6986594  0.803984
RNA45SN4     2.33948e+06     -0.1186159  0.113416 -1.045849 0.2956309  0.446020
RNA28SN4     1.07466e+06     -0.0668078  0.121279 -0.550860 0.5817299  0.714483
LOC102724728 6.60246e-01      1.1222334  4.555978  0.246321 0.8054337        NA
```



```
> results_shrunk
log2 fold change (MAP): condition nutlin vs dmso
Wald test p-value: condition nutlin vs dmso
DataFrame with 23137 rows and 6 columns
                baseMean log2FoldChange     lfcSE      stat    pvalue      padj
               <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
C1orf141        0.559959     -0.0730664  0.126743 -0.521625 0.6019312        NA
PKP1            3.985401      0.2727719  0.303353  0.938970 0.3477460        NA
CSMD2           7.969707      0.0368457  0.366105  0.100489 0.9199558        NA
RERE         3014.371247      0.1014699  0.118024  0.859741 0.3899316 0.5446402
HIVEP3        667.861134     -0.3561748  0.170605 -2.087694 0.0368254 0.0890829
...                  ...            ...       ...       ...       ...       ...
MAFIP        3.88485e+02      0.3452405  0.196592  1.756135 0.0790653  0.164021
MGC70870     2.47017e+02     -0.0894742  0.231110 -0.387131 0.6986594  0.803984
RNA45SN4     2.33948e+06     -0.1162883  0.111190 -1.045849 0.2956309  0.446020
RNA28SN4     1.07466e+06     -0.0653130  0.118565 -0.550860 0.5817299  0.714483
LOC102724728 6.60246e-01      0.0373485  0.136621  0.246321 0.8054337        NA
```

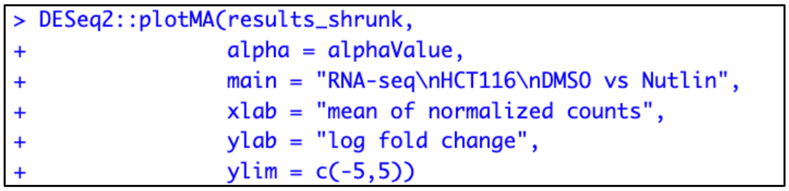
We can get a flyover view of our results with the **summary()** command:



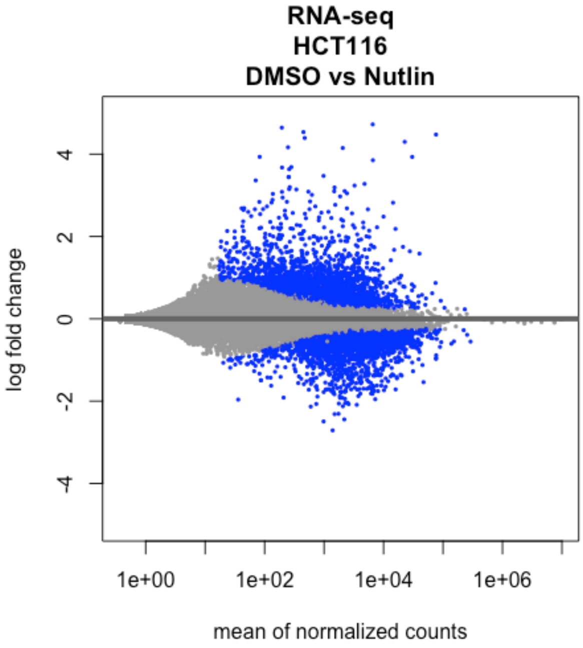
```
> summary(results_shrunk)

out of 23137 with nonzero total read count
adjusted p-value < 0.05
LFC > 0 (up)       : 2693, 12%
LFC < 0 (down)     : 2931, 13%
outliers [1]       : 0, 0%
low counts [2]     : 7626, 33%
(mean count < 16)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

You can check a global visualization of how the genes changed upon the Nutlin treatment using DESeq2 **plotMA** function.

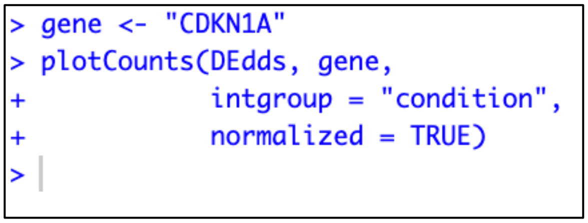
The resulting MA figure will plot each gene's shrunken fold-change in the Y-axis, and each gene's normalized counts in the X-axis. You can tell **plotMA** to color genes based on whether they are considered significant by specifying your **alphaValue**.



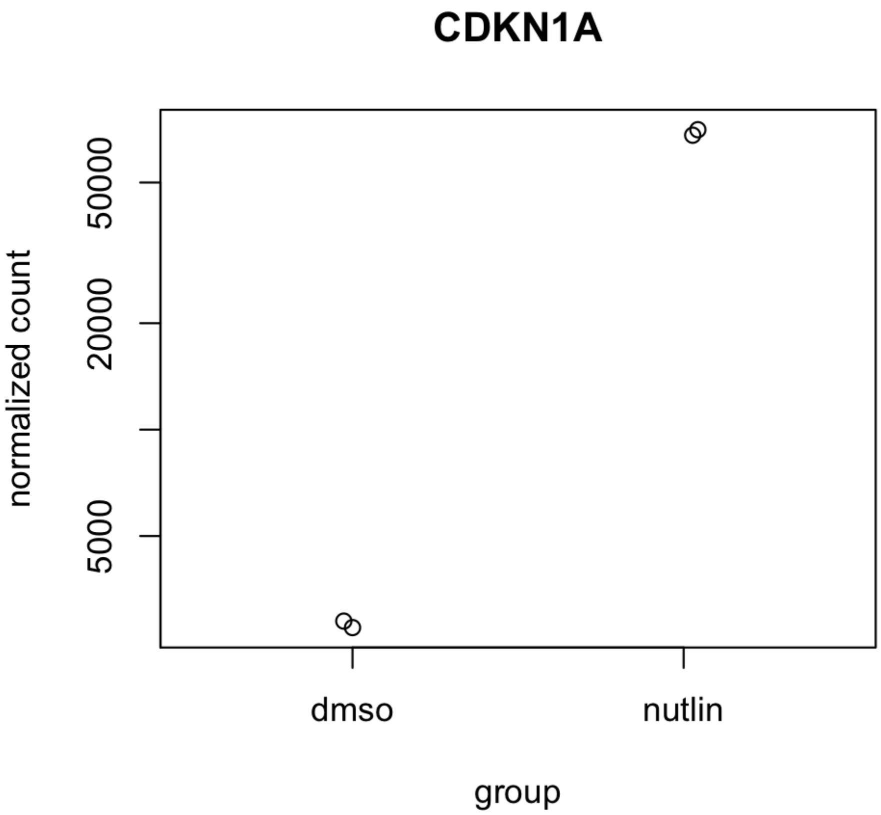
```
> DESeq2::plotMA(results_shrunk,
+                alpha = alphaValue,
+                main = "RNA-seq\nHCT116\nDMSO vs Nutlin",
+                xlab = "mean of normalized counts",
+                ylab = "log fold change",
+                ylim = c(-5,5))
```



You can plot the normalized read counts for a given gene using DESeq2 built-in function **plotCounts**. You need to tell it the name of the gene as it appears in the original annotation file, as well as the name of the column in the condition file that denotes either “Nutlin” or “DMSO” as the conditions.

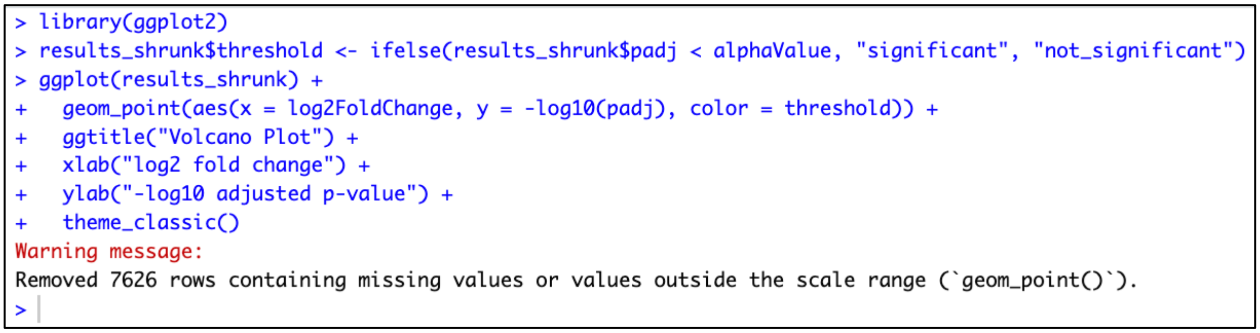


```
> gene <- "CDKN1A"
> plotCounts(DEdds, gene,
+            intgroup = "condition",
+            normalized = TRUE)
> 
```

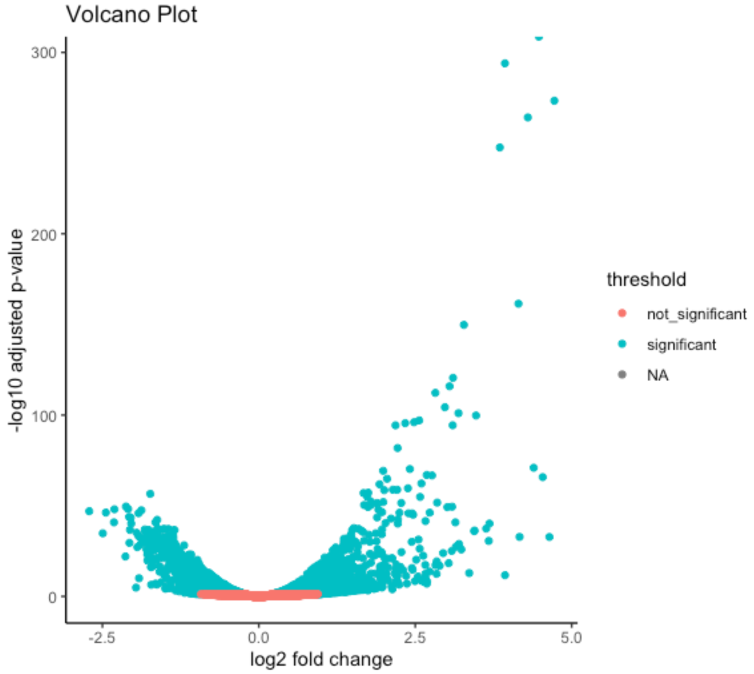


One of the most common visualizations of differential expression results is the volcano plot. While DESeq2 does not have built-in volcano plotting, we can use another package called **ggplot2.** We’ll go more into detail on ggplot’s functionality on a later day.

To color genes by whether or not they’re significant, we first make a new column in our data frame which labels each gene. Then we input the new results data frame into **ggplot2**:

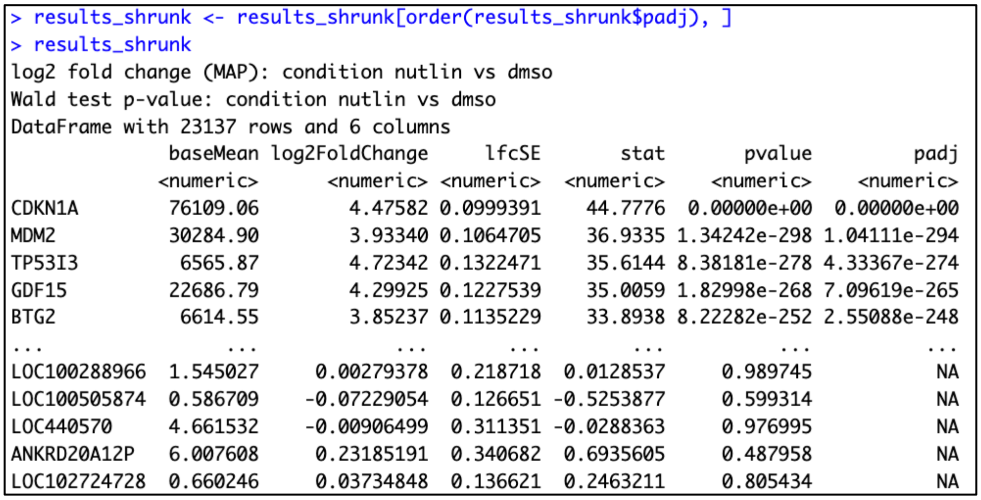


```
> library(ggplot2)
> results_shrunk$threshold <- ifelse(results_shrunk$padj < alphaValue, "significant", "not_significant")
> ggplot(results_shrunk) +
+   geom_point(aes(x = log2FoldChange, y = -log10(padj), color = threshold)) +
+   ggtitle("Volcano Plot") +
+   xlab("log2 fold change") +
+   ylab("-log10 adjusted p-value") +
+   theme_classic()
Warning message:
Removed 7626 rows containing missing values or values outside the scale range (`geom_point()`).
> 
```



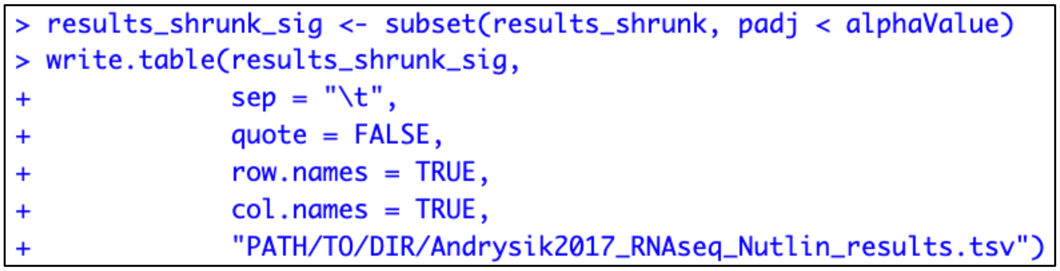
Storing Results

Order the gene results in descending order by their adjusted p-values, so that the most significant genes will be on the top of your results table. You can see again that the very top gene is CDKN1A or p21, a known gene that controls cell cycle progression directly controlled by the transcription factor p53, which itself is activated by the drug Nutlin.



```
> results_shrunk <- results_shrunk[order(results_shrunk$padj), ]
> results_shrunk
log2 fold change (MAP): condition nutlin vs dmso
Wald test p-value: condition nutlin vs dmso
DataFrame with 23137 rows and 6 columns
              baseMean log2FoldChange     lfcSE      stat       pvalue         padj
             <numeric>      <numeric> <numeric> <numeric>    <numeric>    <numeric>
CDKN1A        76109.06        4.47582 0.0999391   44.7776  0.00000e+00  0.00000e+00
MDM2          30284.90        3.93340 0.1064705   36.9335 1.34242e-298 1.04111e-294
TP53I3         6565.87        4.72342 0.1322471   35.6144 8.38181e-278 4.33367e-274
GDF15         22686.79        4.29925 0.1227539   35.0059 1.82998e-268 7.09619e-265
BTG2           6614.55        3.85237 0.1135229   33.8938 8.22282e-252 2.55088e-248
...                ...            ...       ...       ...          ...          ...
LOC100288966  1.545027     0.00279378  0.218718  0.0128537     0.989745           NA
LOC100505874  0.586709    -0.07229054  0.126651 -0.5253877     0.599314           NA
LOC440570     4.661532    -0.00906499  0.311351 -0.0288363     0.976995           NA
ANKRD20A12P   6.007608     0.23185191  0.340682  0.6935605     0.487958           NA
LOC102724728  0.660246     0.03734848  0.136621  0.2463211     0.805434           NA
```

Finally, let's store our significant genes onto a text output file. First, we'll filter our gene list to only the significant results, then we'll write it to our desired path



```
> results_shrunk_sig <- subset(results_shrunk, padj < alphaValue)
> write.table(results_shrunk_sig,
+             sep = "\t",
+             quote = FALSE,
+             row.names = TRUE,
+             col.names = TRUE,
+             "PATH/TO/DIR/Andrysik2017_RNAseq_Nutlin_results.tsv")
```

This command will save your results as a tab-separated file, which you can view with bash (or spreadsheet programs like Excel). We'll use our results in later analyses.