Bioinformatics for RNAseq Workshop - day 2

Rebecca Batorsky, Sr. Bioinformatics Specialist at Tufts

April 2019

While you are waiting, please:

- Download day 1 slides from: https://sites.tufts.edu/biotools/
  - Tutorials -> bioinformatics_rnaseq_workshop_day_2
- In a **Chrome** web browser: https://ondemand.cluster.tufts.edu
  - Login with your tufts credentials

Today's Schedule

1. Viewing alignment in IGV
2. R and Rstudio Introduction
3. Differential Expression Analysis with DESeq2

We'll follow closely some courses from Harvard Chan School Bioinformatics Core (developers of BCBIO)

https://github.com/hbc/NGS_Data_Analysis_Course/
https://hbctraining.github.io/DGE_workshop/

See their pages for many useful tutorials

Summary

Why switch to R programming language?

Rstudio on the Tufts HPC cluster via "On Demand"

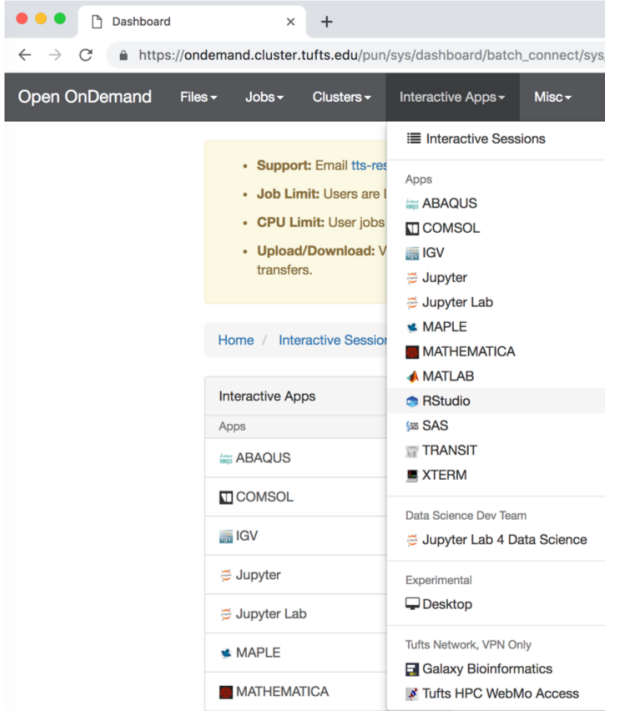
1. **Chrome** web browser
   https://ondemand.cluster.tufts.edu

2. Choose Rstudio from the Interactive Apps Menu

3. Choose

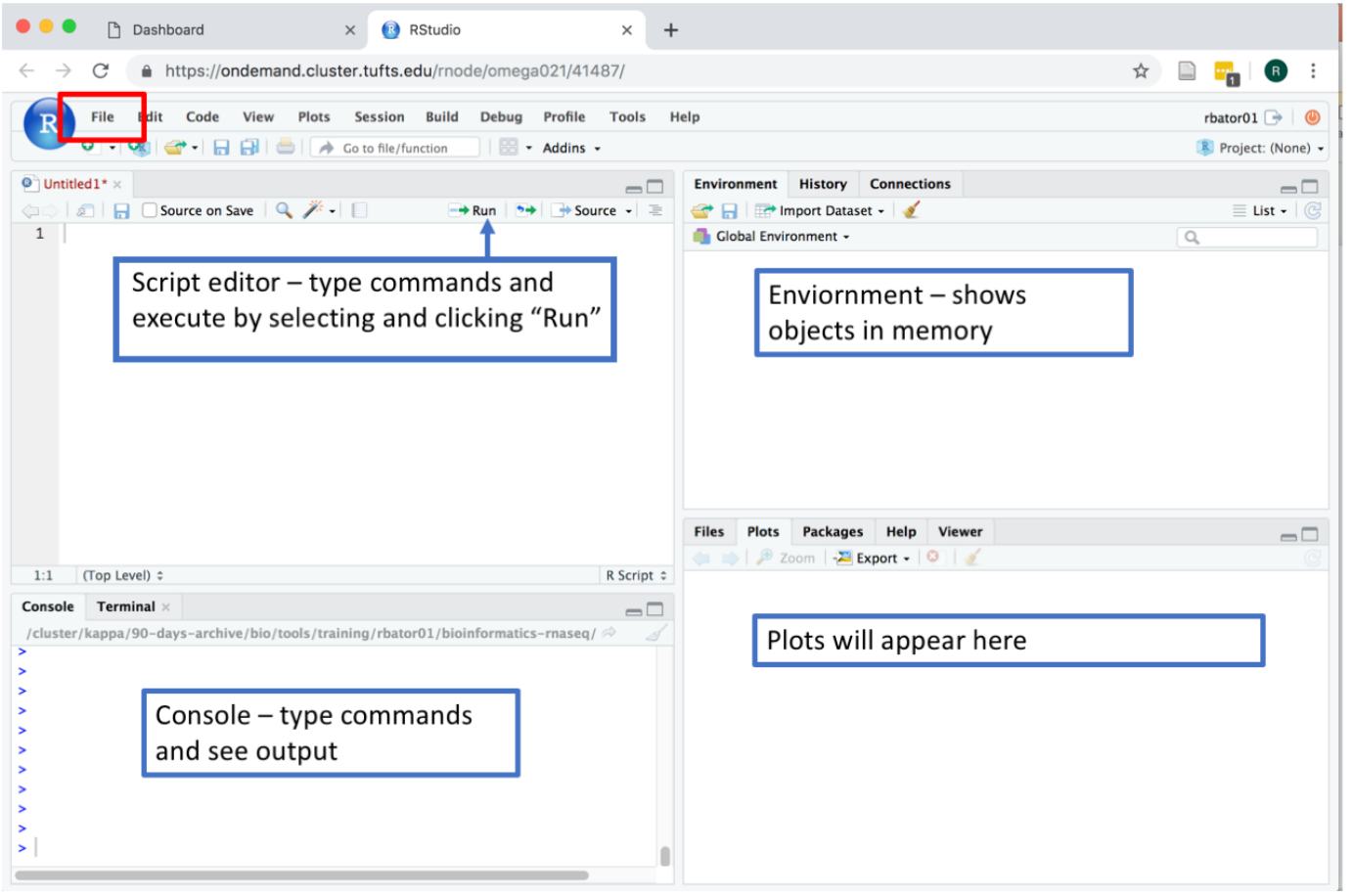
   - Number of hours: 4
   - Number of cores: 1
   - Amount of Memory: 32 Gb
   - R version: 3.5.0

4. Press "Connect to Rstudio"



Rstudio Interface

Go to the File menu -> New File -> R Script, you should see:



File menu -> Save your file as intro.R

Testing some commands

In the script editor type the following and click "Run". To run a block of code, highlight it and click "Run"

- To see your current working directory

```
getwd()
```

```
"/cluster/kappa/90-days-archive/bio/tools/training/<your user name>/bioinformatics-rnaseq"
```

- To change to our workshop directory

```
setwd('~/bioinformatics-rnaseq/')
```

- Set some variables

```
x = 3
y = 5
z = x + y
```

Check that the variables appear in the Environment panel

R libraries

- Check which paths on the cluster R will use to find library locations:

```
.libPaths()
```

  Will output the library paths:

```
[1] "/opt/shared/R/3.5.0/lib64/R/library"
```

- Add a custom library path, which contains libraries that we'll use:

```
.libPaths('/cluster/tufts/bio/tools/R_libs/3.5')
```

- Load a library:

```
library(tidyverse)
```

Vectors

A vector is the most basic data structure in R

```
numbers     <- c(1,2,3,4)
letters     <- c("A","B","C","D")
```

Vectors must contain all the same type of data If you give a combination, R will coerce it into one type

```
combination <- c("A","B",1,2)
```

```
combination
```

```
[1] "A" "B" "1" "2"
```

Data Frames

Data frames are the most common data structure for tables

- Read in the metadata for our experiment:

```
meta <- read.table("data/sample_info.txt", header=TRUE)
```

View data frame

```
meta
```

```
          condition
SNF2_rep1      SNF2
SNF2_rep2      SNF2
SNF2_rep3      SNF2
SNF2_rep4      SNF2
SNF2_rep5      SNF2
WT_rep1          WT
WT_rep2          WT
WT_rep3          WT
WT_rep4          WT
WT_rep5          WT
```

- View in a new tab

```
View(meta)
```

Manipulating Data Frames

The dollar sign `$` can be used to select columns in the data frame:

```
meta$condition
```

```
 [1] SNF2 SNF2 SNF2 SNF2 SNF2 WT   WT   WT   WT   WT
Levels: SNF2 WT
```

To add a column to a data frame

1. create a vector with 10 values

   ```
   number <- c(1,2,3,4,5,9,8,7,6,10)
   ```

2. use `$` on the left side:

   ```
   meta$number <- number
   ```

View the new table:

```
meta
```

	condition	number
SNF2_rep1	SNF2	1
SNF2_rep2	SNF2	2
SNF2_rep3	SNF2	3
SNF2_rep4	SNF2	4
SNF2_rep5	SNF2	5
WT_rep1	WT	6
WT_rep2	WT	7
WT_rep3	WT	8
WT_rep4	WT	9

Sorting and filtering data frames

Another way to select from a data frame is using the sytax

```
dataframe[rows, columns]
```

For example, to select the first row and first two columns:

```
meta[1,1:2]
```

```
          condition number
SNF2_rep1      SNF2      1
```

To select the whole first column just least the row number blank (this is the same as `meta$condition`):

```
meta[ ,1]
```

```
 [1] SNF2 SNF2 SNF2 SNF2 SNF2 WT   WT   WT   WT   WT
Levels: SNF2 WT
```

Sorting and filtering data frames

We can check for rows of our data frame that have a condition satisfied:

```
meta$number > 5
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Then, we can use this to filter our data frame for these rows using **which**:

```
meta[which(meta$number > 5), ]
```

We can filter for multiple conditions by using the `&` operator:

```
meta[which(meta$number > 5 & condition = "WT"), ]
```

```
        condition number
WT_rep1        WT      9
WT_rep2        WT      8
WT_rep3        WT      7
WT_rep4        WT      6
WT_rep5        WT     10
```

Data frame recap:

Data frames have row names and columns

```
          condition number
SNF2_rep1      SNF2      1
SNF2_rep2      SNF2      2
SNF2_rep3      SNF2      3
SNF2_rep4      SNF2      4
SNF2_rep5      SNF2      5
WT_rep1          WT      9
WT_rep2          WT      8
WT_rep3          WT      7
WT_rep4          WT      6
WT_rep5          WT     10
```

Filtering can done by selecting a row and column `dataframe[rows, columns]`, where `rows` and `columns` can be specified as either numbers or binary strings.

Exercise (5 minutes) Load the metadata for our experiment that we saw in day 1:

```
meta_2 <- read.table('data/ERP004763_info.txt', header=T)
```

Look at the column names and values using View().

Then, repeat the exercise we did yesterday where you filter the data frame for only rows corresponding to WT replicate 1.

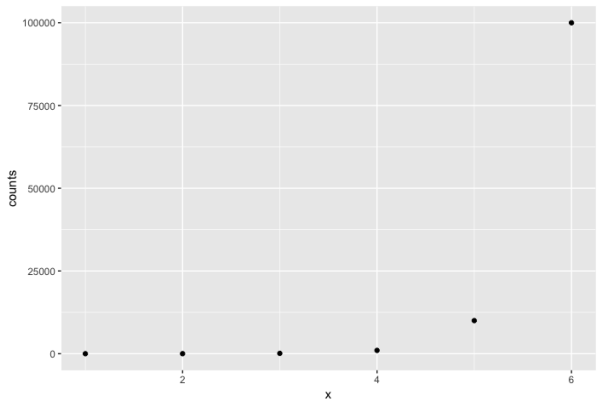
Plotting example with ggplot2

Create a data frame with count values that span 6 orders of magnitude

```
library(ggplot2)
x        <-c(1,2,3,4,5,6)
counts   <-c(1,10,100,1000,10000, 100000)
genotype <-c('WT','WT','WT','SNF2','SNF2','SNF2')
df       <-data.frame(x,counts,genotype)
```

To generate a basic dot plot:

```
ggplot(df) + geom_point(aes(x=x,y=counts))
```



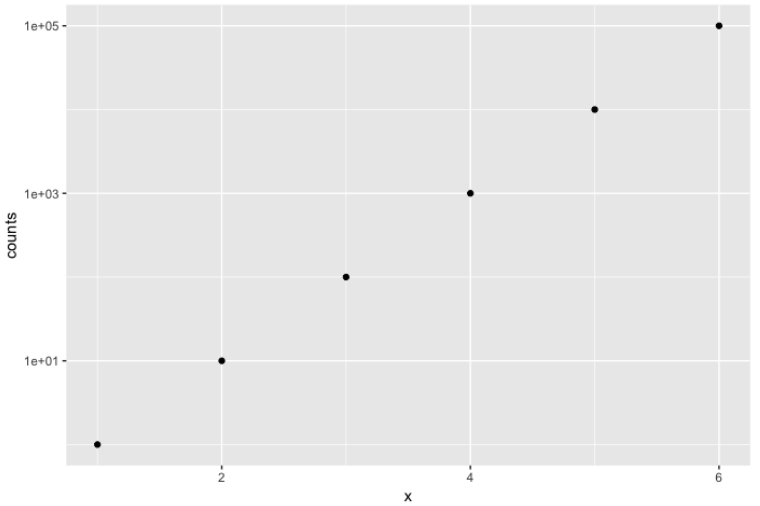
ggplot = "Grammar of Graphics"

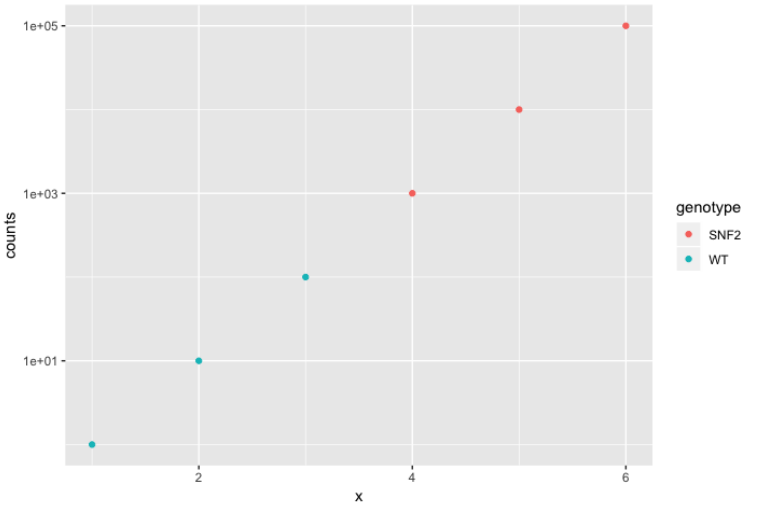
- ggplot(): create data object
- aes (): "how" the data will look
- geom_point/bar/line: "what" the plot will be

Plotting example with ggplot2

To see the relationships more clearly, we'll set the y log scale:

```
ggplot(df) + geom_point(aes(x=x,y=counts)) + scale_y_log10()
```



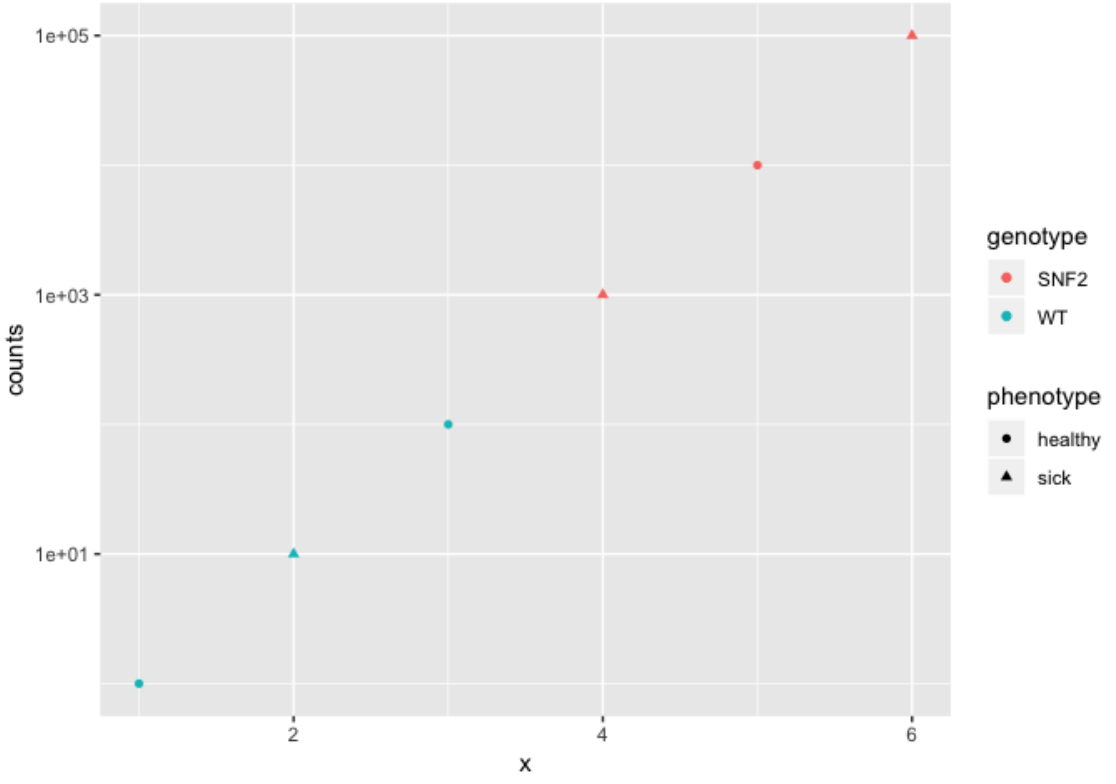


To color by the genotype value:

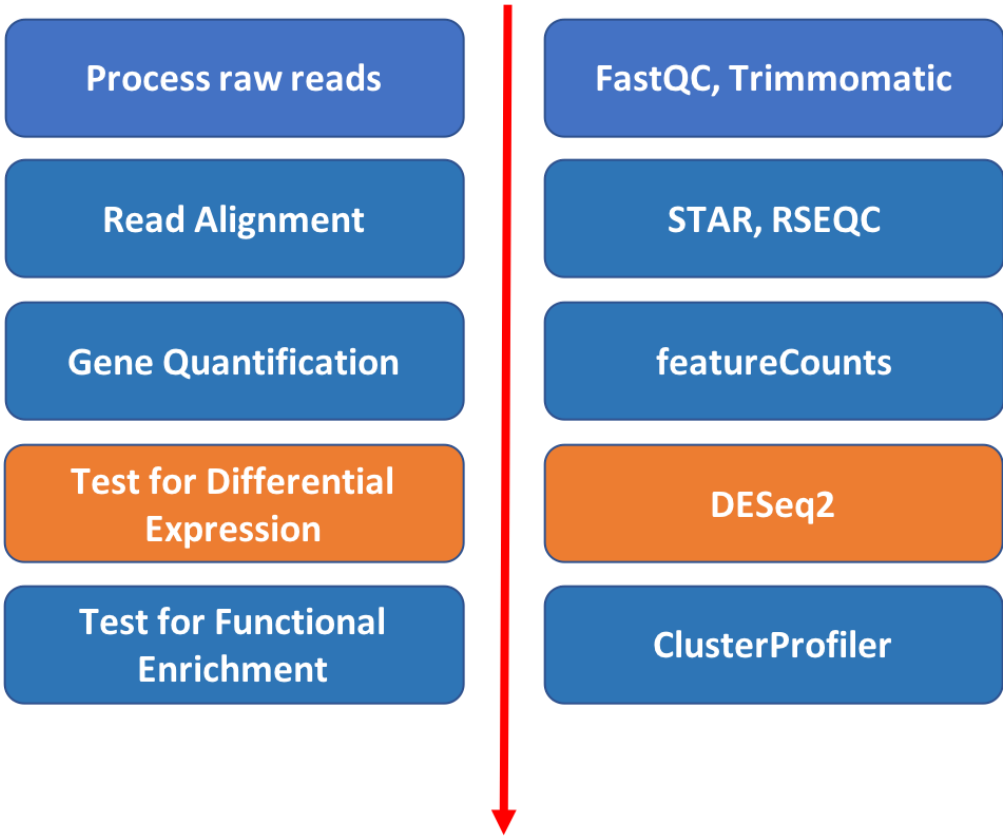
```
ggplot(df) + geom_point(aes(x=x, y=counts, color=genotype)) + scale_y_log10()
```

Exercise (5 minutes)

Construct another data frame which has an additional column called "phenotype" which can be either "sick" or "healthy". Make the same plot but add another argument shape=phenotype to the aesthetic (aes).

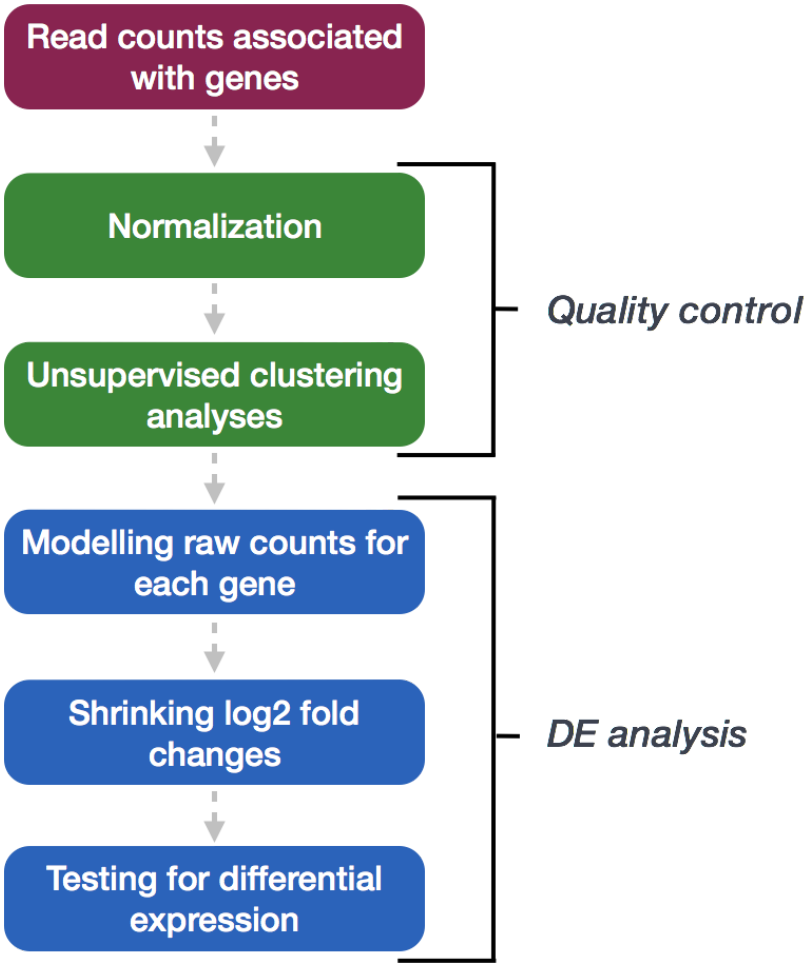


Differential Expression with DESeq2



A common goal for RNAseq analysis is to identify genes that are differentially expressed between conditions

DESeq2: Workflow overview



Getting set up: Loading libraries

File->"Open File" the R file 'bioinformatics_rnaseq/scripts/deseq2.R'

From now on commands are already in the script deseq2.R
You only need to **Select and run**:

```
# Put HPC biotools R libraries on your R path
.libPaths(c('', '/cluster/tufts/bio/tools/R_libs/3.5/'))

# load required libraries
library(DESeq2)
library(vsn)
library(ggplot2)
library(dplyr)
library(tidyverse)
library(ggrepel)
library(DEGreport)
library(pheatmap)
library(org.Sc.sgd.db)
library(clusterProfiler)
```

Reading in data

Load preprocessed data set of 5 WT replicates and 5 SNF2 knockouts

```
## Read in preprocessed count and meta data
course_data_path="~/bioinformatics-rnaseq/data/"
setwd(course_data_path)
data <- read.table("sacCerfeatureCounts_gene_results.formatted.txt",header=TRUE)
meta <- read.table("sample_info.txt", header=TRUE)

## View table in new window
View(data)
View(meta)
```

View the first few lines of "data" and "meta" using **head()** or open the files in another tab using **View()**

```
> head(meta)
          condition
SNF2_rep1      SNF2
SNF2_rep2      SNF2
SNF2_rep3      SNF2
SNF2_rep4      SNF2
SNF2_rep5      SNF2
WT_rep1          WT
```

```
> head(data)
          SNF2_rep1 SNF2_rep2 SNF2_rep3 SNF2_rep4 SNF2_rep5 WT_rep1 WT_rep2 WT_rep3 WT_rep4 WT_rep5
YAL069W           0         0         0         0         0       0       0       0       0       0
YAL068W-A         0         0         0         0         0       0       0       0       0       0
YAL068C           2         2         2         1         0       0       0       0       2       2
YAL067W-A         0         0         0         0         0       0       0       0       0       0
YAL067C         103        51        44        90        53      12      23      21      30      29
YAL066W           2         0         0         0         0       0       0       0       0       0
```

Create DESeq2 Dataset object

Check to make sure that all rows labels in meta are columns in data!

```
all(colnames(data) == rownames(meta))
```

Create the dataset and run the analysis

```
dds  <- DESeqDataSetFromMatrix(countData = data, colData = meta, design = ~ condition)
dds  <- DESeq(dds)
```

Behind the scenes these steps were run:

- estimating size factors
- gene-wise dispersion estimates
- mean-dispersion relationship
- final dispersion estimates
- fitting model and testing

DESeq2: Design formula

```
dds  <- DESeqDataSetFromMatrix(countData = data, colData = meta, design = ~ condition)
```

The design formula `design = ~condition`

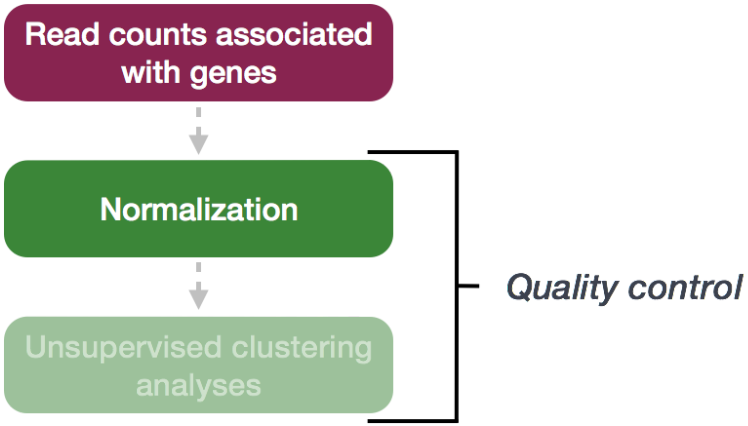
- Tells DESeq2 which factors in the metadata to test
- The design can include multiple factors that are columns in the metadata
- The factor that you are testing for comes **last**, and factors that you want to account for come first

E.g. To test for differences in condition while accounting for sex and age:

```
design = ~ sex + age + condition
```

It's even possible to include time series data and interactions

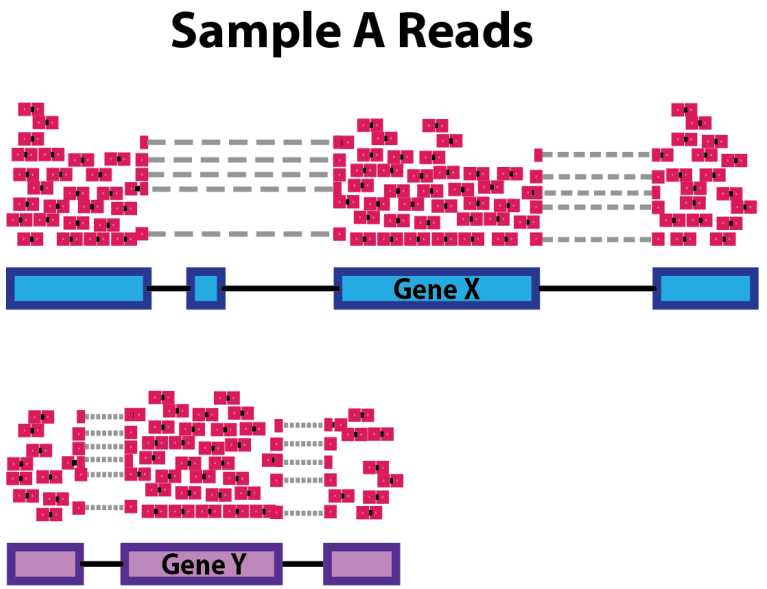
Normalization



Normalization

The number of sequenced reads mapped to a gene depends on

- Gene Length



Normalization

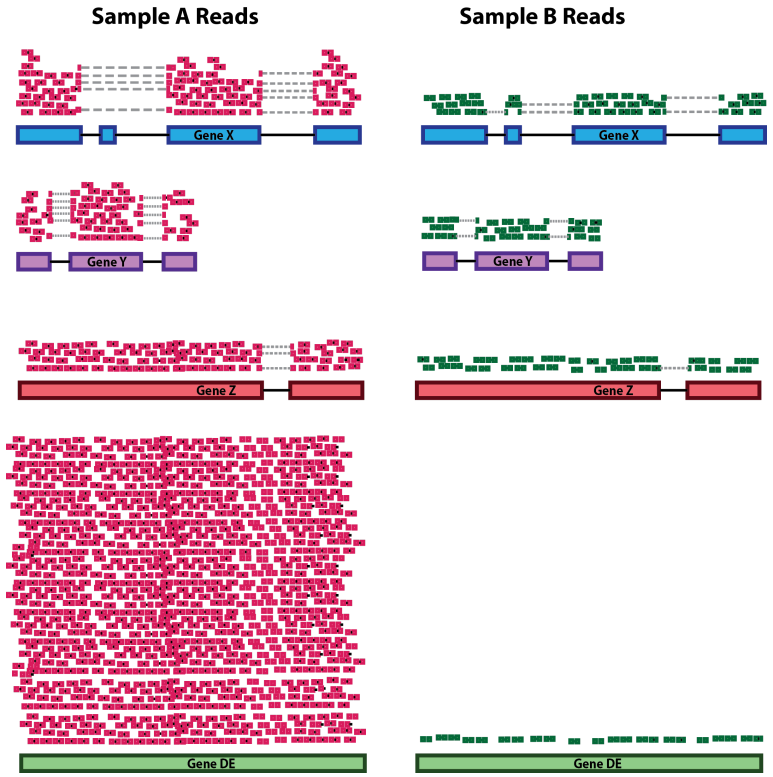
The number of sequenced reads mapped to a gene depends on:

- Gene Length
- Sequencing depth

Normalization:

The number of sequenced reads mapped to a gene depends on:

- Gene Length
- Sequencing depth
- The expression level of other genes in the sample
- **It's own expression level**

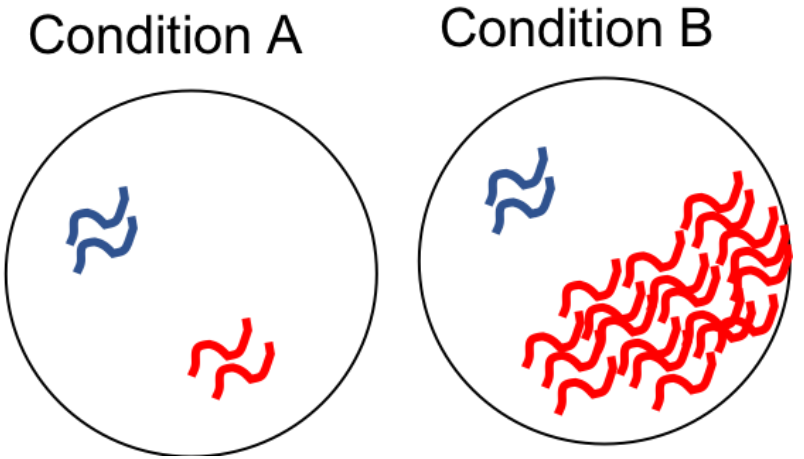


Normalization eliminates the factors that are not of interest!

Normalization:

The **naive** approach: divide by total library size (CPM, TPM) for each sample is NOT reccomended for comparison between samples

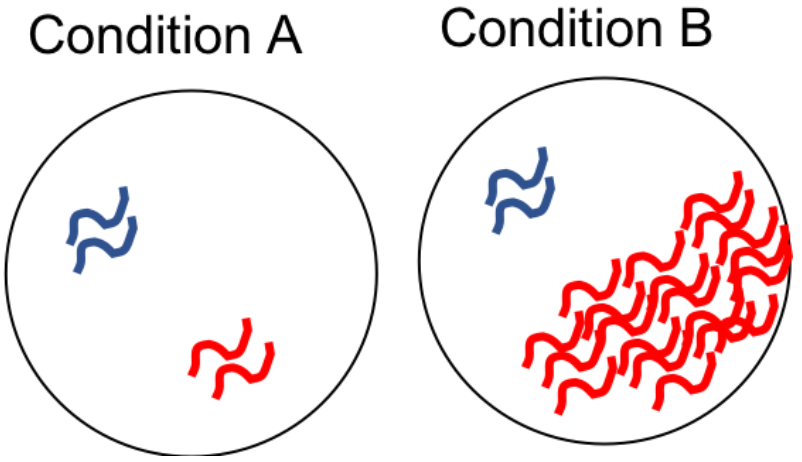
Why not? Reads are a finiate resource and **Composition** matters!



Normalization:

The **naive** approach: divide by total library size (CPM, TPM) for each sample is NOT reccomended for comparison between samples

Why not? Reads are a finiate resource and **Composition** matters!



If you only got 10 reads per condition...

	condition A	condition B
gene 1	5 reads	1 reads
gene 2	5 reads	9 reads

Normalization with DESeq2: Median of ratios method

Accounts for both sequencing depth and composition

Step 1: creates a pseudo-reference sample (row-wise geometric mean)

For each gene, a pseudo-reference sample is created that is equal to the geometric mean across all samples.

gene	sampleA	sampleB	pseudo-reference sample
1	1000	1000	$\sqrt{(1000 * 1000)}$ = **1000**
2	10	1	$\sqrt{(10 * 1)}$ = **3.16**
…	…	…	…

Normalization with DESeq2: Median of ratios method

Step 2: calculates ratio of each sample to the reference

Calculate the ratio of each sample to the pseudo-reference. Since most genes aren't differentially expressed, ratios should be similar.

gene	sampleA	sampleB	pseudo-reference sample	ratio of sampleA/ref	ratio of sampleB/ref
1	1000	1000	1000	1000/1000 = **1.00**	1000/1000 = **1.00**
2	10	1	3.16	10/3.16 = **3.16**	1/3.16 = **0.32**
…	…	…	…		

Normalization with DESeq2: Median of ratios method

Step 2: calculates ratio of each sample to the reference

Calculate the ratio of each sample to the pseudo-reference.

gene	sampleA	sampleB	pseudo-reference sample	ratio of sampleA/ref	ratio of sampleB/ref
1	1000	1000	1000	1000/1000 = **1.00**	1000/1000 = **1.00**
2	10	1	3.16	10/3.16 = **3.16**	1/3.16 = **0.32**
...	...	...	...		

Step 3: calculate the normalization factor for each sample (size factor)

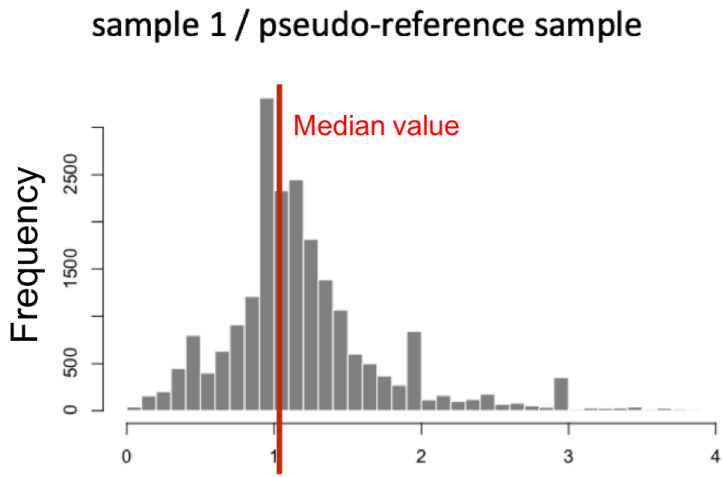
The median value of all ratios for a given sample is taken as the normalization factor (size factor) for that sample:

```
normalization_factor_sampleA <- median(c(1.00, 3.16)) = 2.08
normalization_factor_sampleB <- median(c(1.00, 0.32)) = 0.66
```

Normalization with DESeq2: Median of ratios method

Visualization of normalization factor for a sample:

- Median should be ~1 for each sample
- **This method is robust to imbalance in up-/down-regulation**



Median of ratios method

Step 4: calculate the normalized count values using the normalization factor

This is performed by dividing each raw count value in a given sample by that sample's size factor to generate normalized count values.

SampleA normalization factor = 2.08

SampleB normalization factor = 0.66

Raw Counts

gene	sampleA	sampleB
1	1000	1000
2	10	1

Normalized Counts

gene	sampleA	sampleB
1	1000/2.08 = **480.77**	1000 / 0.66 = **1515.16**
2	10/2.08 = **4.81**	1 / 0.66 = **1.52**

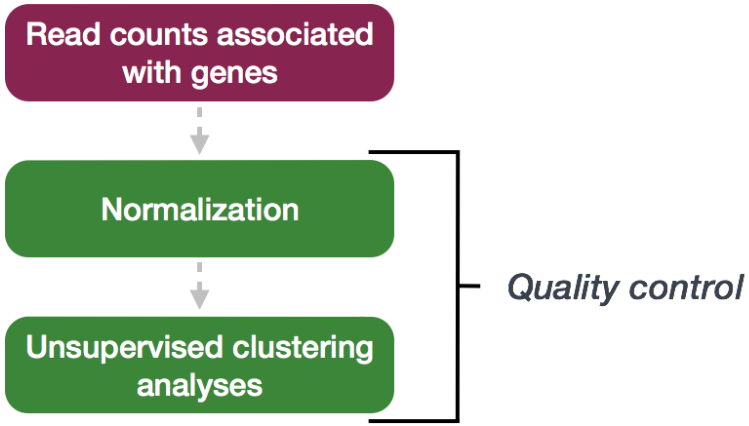
Exercise (1 minutes)

View the size factors that were calculated by DESeq2 using your R script:

```
sizeFactors(dds)
```

Are they roughly the expected value? Based on these, which sample has the fewest reads and which the largest amount?

Unsupervised Clustering



Unsupervised Clustering

Quality Control step to asses overall similarity between samples:

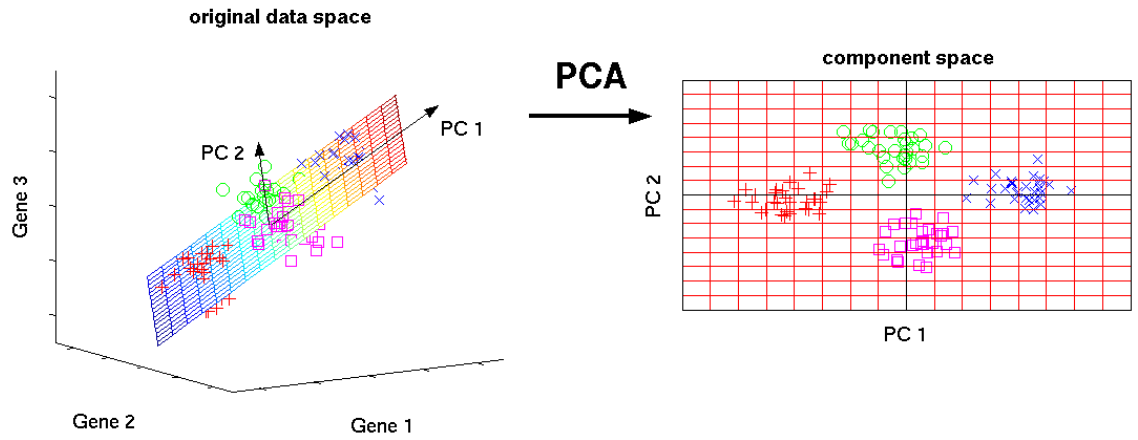
- Which samples are similar to each other, which are different?
- Does this fit to the expectation from the experiment's design?
- What are the major sources of variation in the dataset?

Principle Components Analysis

Here is an example with three genes measured in many samples:

gene	sampleA	sampleB	sampleC	sampleD	..
1	1000	1000	100	10	..
2	10	1	5	6	..
3	10	1	10	20	..

Each gene that we measure is a "dimension" and we can visualize up to 3 PCA can help us visualize relationships in out data in a lower number of dimensions



http://www.nlpca.org/fig_pca_principal_component_analysis.png

Make a PCA plot

- This uses the built in function plotPCA from DESeq2 (built on top of ggplot)
- The regularized log transform (rlog) improves clustering by log transforming the data

```
rld <- rlog(dds, blind=TRUE)
plotPCA(rld, intgroup="condition") + geom_text(aes(label=name))
```

Exercise (5 minutes)

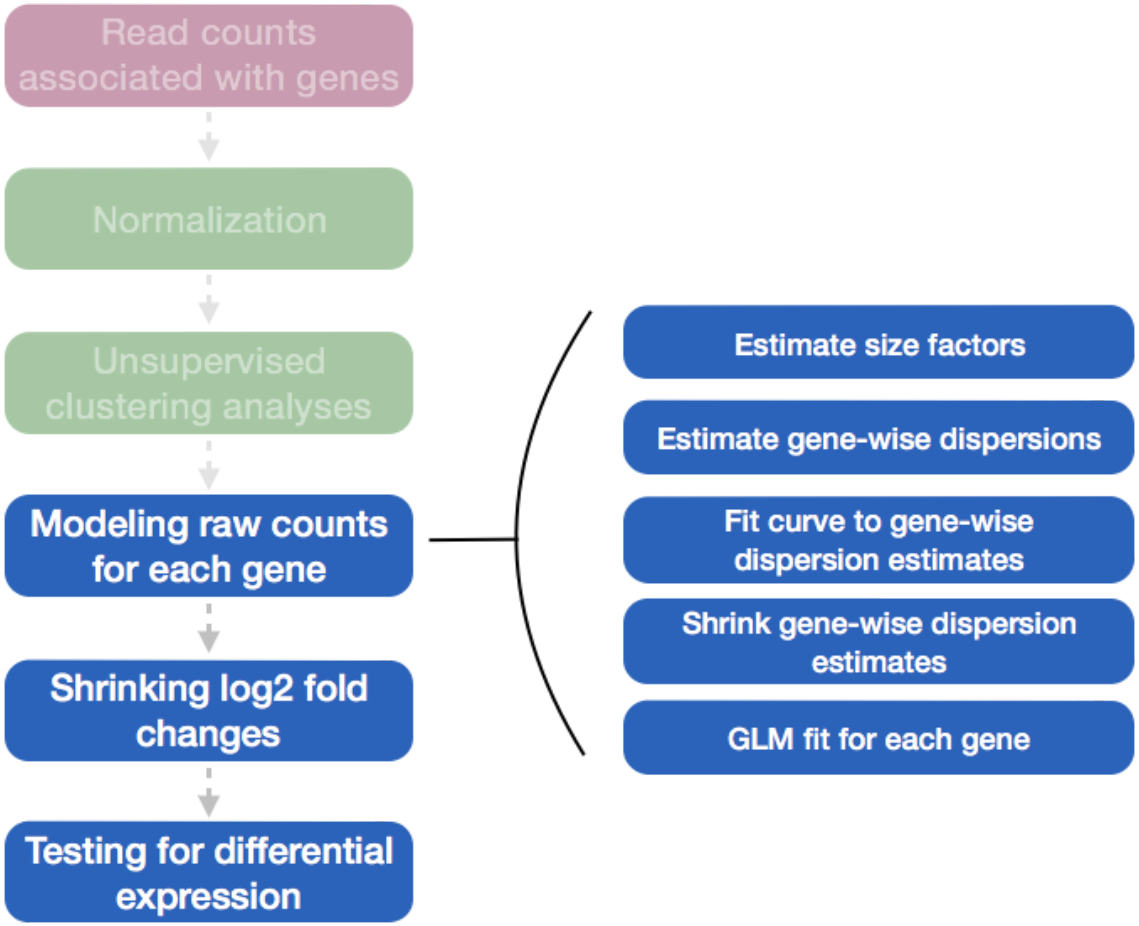
Does something look wrong with the PCA plot?

Suppose we go back over the data and verify that somewhere in the processing steps, the data for WT_rep1 and SNF_rep5 columns were switched! Go back and load the corrected data frame

```
data <- read.table("featurecounts_results.formatted.fixed.txt",header=TRUE)
```

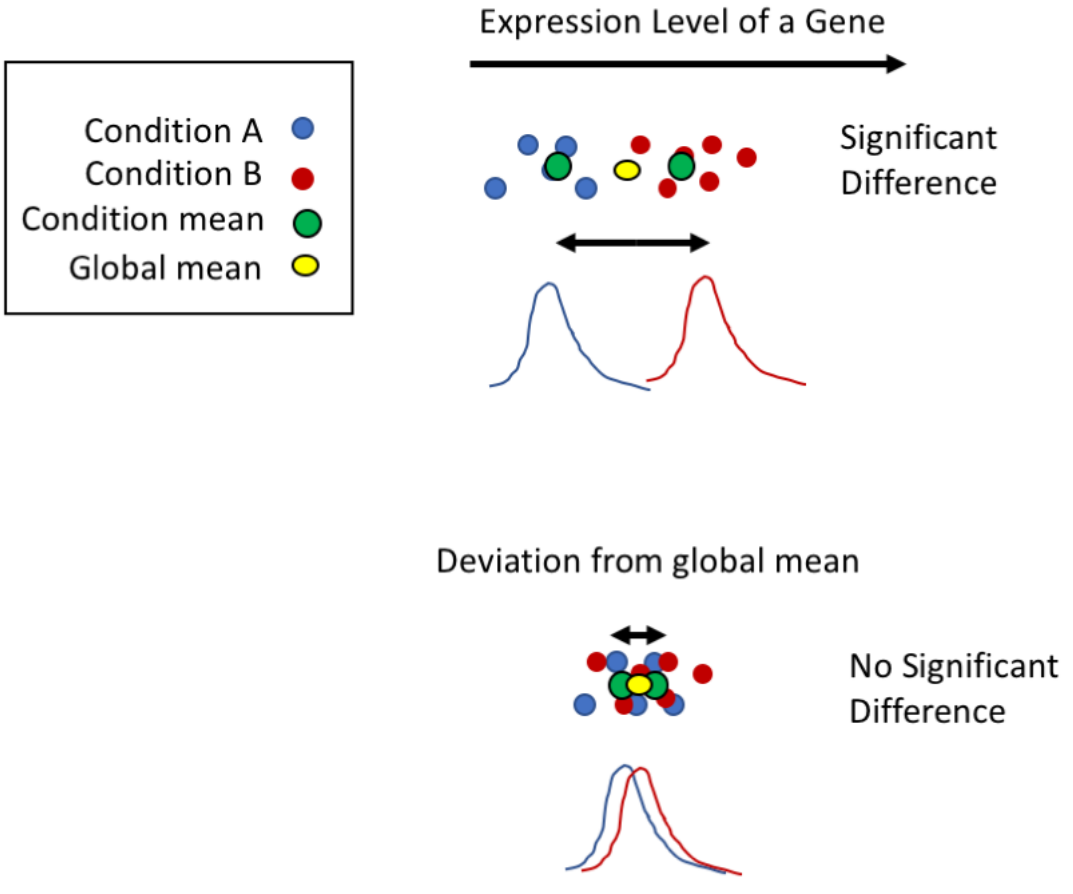
Rerun **all steps** in the analysis, including the PCA, and verify that things look as expected.

DESeq2 Workflow



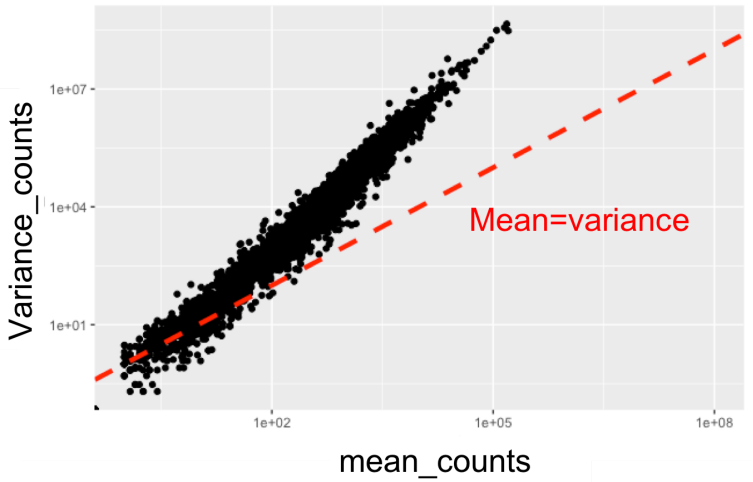
Modeling count data

DESeq2 will seek to fit a probability distibution to each gene we measured and perform a statistical test to determine whether there is a difference between conditions



Modeling count data

This plot shows the relationship between the mean and variance across WT replicates for *each gene* we measured:



Since our mean is not equal to the variance, a Negative binomial distribution is used

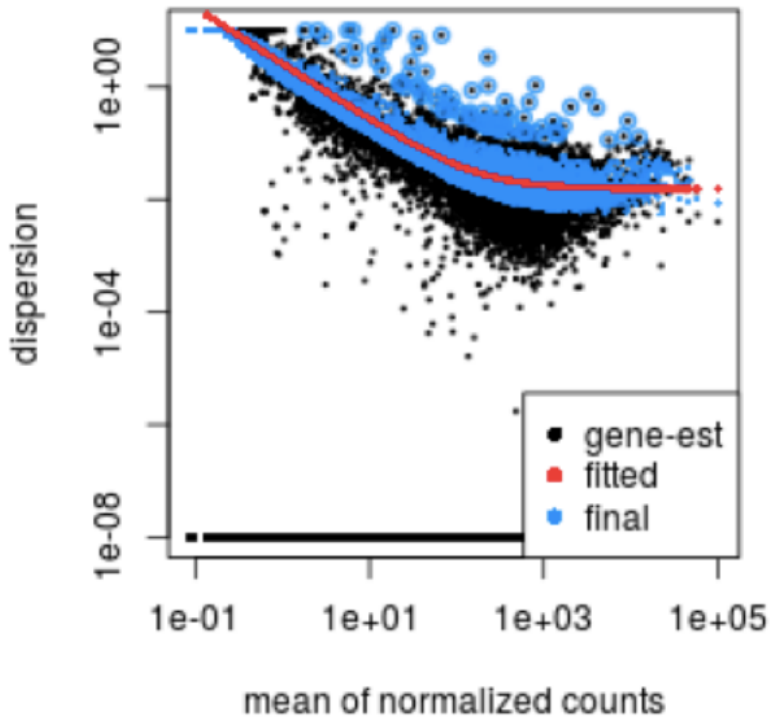
- 2 parameters NB(μ , α)
- μ mean, α dispersion
- allows extra source of variation

- Dispersion is a measure of spread or variability in the data between **biological repliates**
- Genes with high dispersion will have lower significance than genes with low dispersion for a given difference between conditions

Viewing the per-gene dispersion estimates

You can see the dispersion estimates

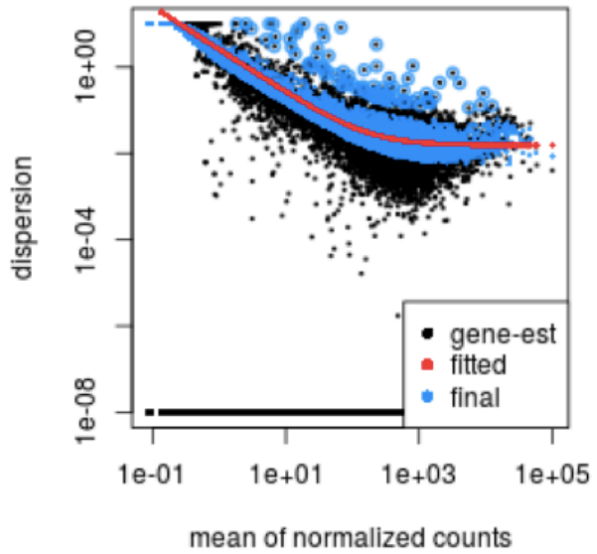
```
plotDispEsts(dds)
```

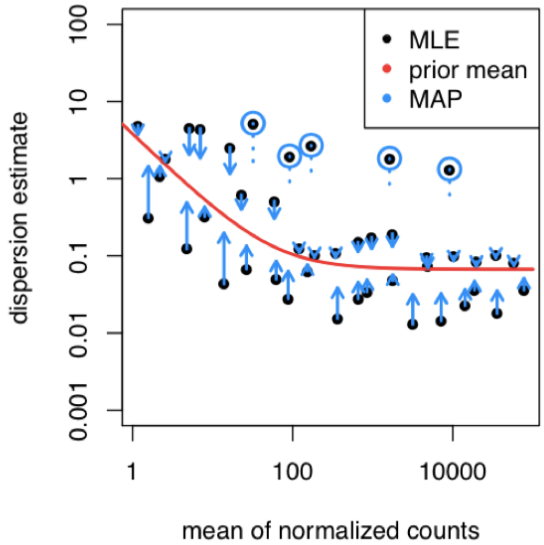


Viewing the gene-wise dispersion estimates

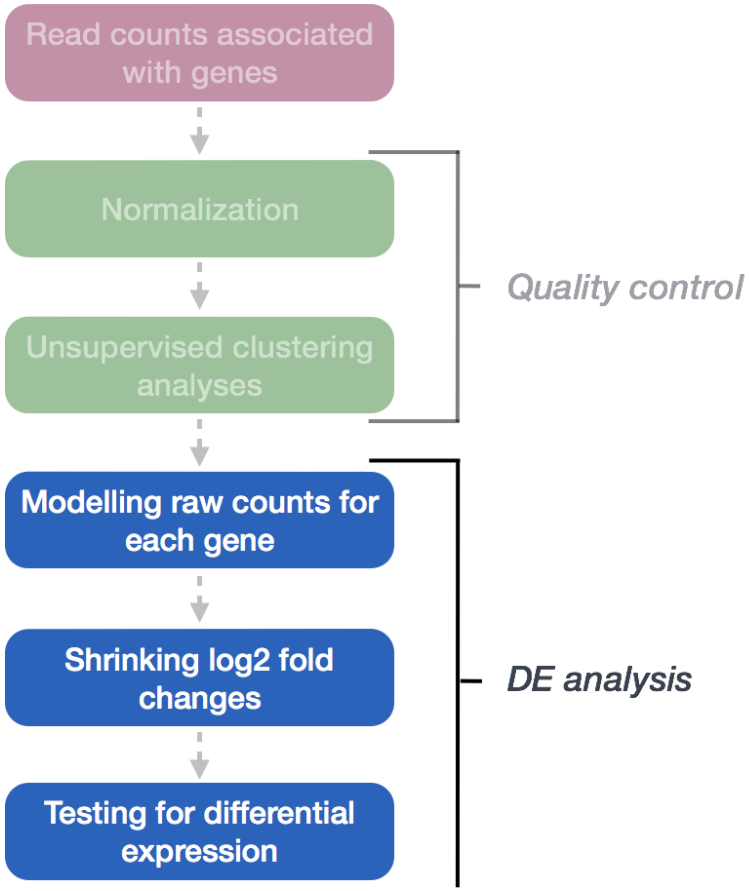
What DESeq is doing:

1. Estimates dispersion per gene using biological replicates
2. Fit a mean-dispersion curve
3. Change dispersion estimates for genes that are far from the curveto avoid false positives





Testing for Differential Expression



Creating contrasts and running a Wald test

The null hypothesis: log fold change = 0 for across conditions

P-values are the probability of rejecting the null hypothesis for a given gene, and adjusted p values take into account that we've made many comparisons:

```
## Creating contrasts
contrast <- c("condition", "SNF2", "WT")
res_unshrunken <- results(dds, contrast=contrast)
summary(res_unshrunken)
```

```
out of 6391 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 485, 7.6%
LFC < 0 (down)     : 637, 10%
outliers [1]       : 3, 0.047%
low counts [2]     : 371, 5.8%
(mean count < 5)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

Shrinkage of the log2 fold changes

One more step where information is used across genes to avoid overestimates of differences between genes with high dispersion

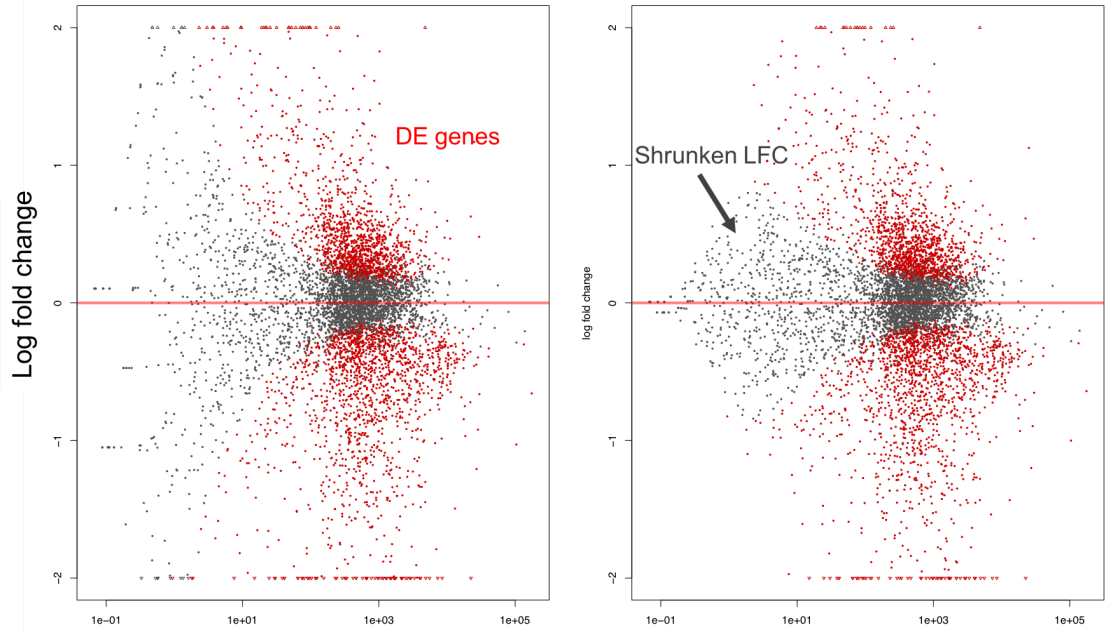
Shrinkage of the log2 fold changes

One more shrinking step - shrink the estimated log2 fold changes
This is not done by default, so we run the code:

```
res <- lfcShrink(dds, contrast=contrast, res=res_unshrunken)
```

MA plot: Log ratio vs. average for comparison

- Shows the mean of the normalized counts versus the log2 foldchanges for all genes tested
- Genes that are significantly DE are colored to be easily identified and should span the range of fold changes




```
plotMA(res_unshrunken, ylim=c(-2,2))
plotMA(res, ylim=c(-2,2))
```

Exploring results

A summary of the results:

```
summary(res)
```

```
out of 6391 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 1464, 23%
LFC < 0 (down)     : 1623, 25%
outliers [1]       : 0, 0%
low counts [2]     : 0, 0%
(mean count < 0)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

Exploring results

```
head(results)
```

```
log2 fold change (MAP): condition SNF2 vs WT
Wald test p-value: condition SNF2 vs WT
DataFrame with 6 rows and 6 columns
                  baseMean     log2FoldChange             lfcSE                stat              pvalue               padj
                 <numeric>          <numeric>         <numeric>           <numeric>           <numeric>          <numeric>
YAL069W                  0                 NA                NA                  NA                  NA                 NA
YAL068W-A                0                 NA                NA                  NA                  NA                 NA
YAL068C   0.967880032164114  0.130810394924328 0.392843011761567 0.328000845094124   0.742911023648992   0.819176044192669
YAL067W-A                0                 NA                NA                  NA                  NA                 NA
YAL067C      40.87932305681   1.01424015380012 0.212805370098983  4.74630178261535 2.07169548468324e-06 1.31612384121377e-05
YAL066W   0.140275942926642 0.0448864068979152 0.208358779835131 0.217860827508585    0.82753754913751   0.880584827928377
```

log2FoldChange = $log_2 \frac{SNF2Counts}{WTCounts}$ Estimated from the model

padj - Adjusted pvalue for the probability that the log2FoldChange is not zero

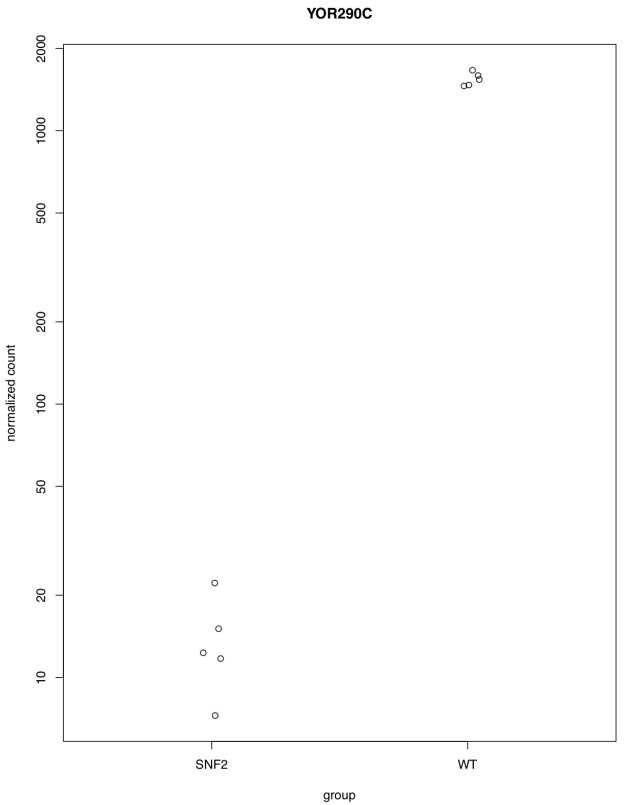
Review DESeq workflow

These comprise the full workflow:

```
# Setup DESeq2
dds  <- DESeqDataSetFromMatrix(countData = data, colData = meta, design = ~ condition)
# Run size factor estimation, dispersion estimation, dispersion shrinking, testing
dds  <- DESeq(dds)
# Tell DESeq2 which conditions you would like to output
contrast <- c("condition", "SNF2", "WT")
# Output results to table
res_unshrunken <- results(dds, contrast=contrast)
# Shrink log fold changes for genes with high dispersion
res <- lfcShrink(dds, contrast=contrast, res=res_unshrunken)
```

Plotting a single gene SNF2 (YOR290C open reading frame)

```
## simple plot for a single gene
plotCounts(dds, gene="YOR290C", intgroup="condition")
```



Filtering to find significant genes

Let's filter the results table for padj < 0.05

```
padj.cutoff <- 0.05
significant_results <- results[which(results$padj < padj.cutoff),]
```

We can export these results to a table:

```
file_name='results_pval_0.05.txt'
write.table(significant_results, file_name)
```

Plot multiple genes in a heatmap

Sort the rows from smallest to largest padj and take the top 50 genes:

```
significant_results_sorted <- significant_results[order(significant_results$padj), ]
significant_genes_50       <- rownames(significant_results_sorted[1:50, ])
```

We now have a list of genes

```
significant_genes_50
```

```
"YGR234W" "YDR033W" "YOR290C" "YML123C" ...
```

But we need to find the counts corresponding to these genes. To extract the counts from the rlog transformed object

```
rld_counts <- assay(rld)
```

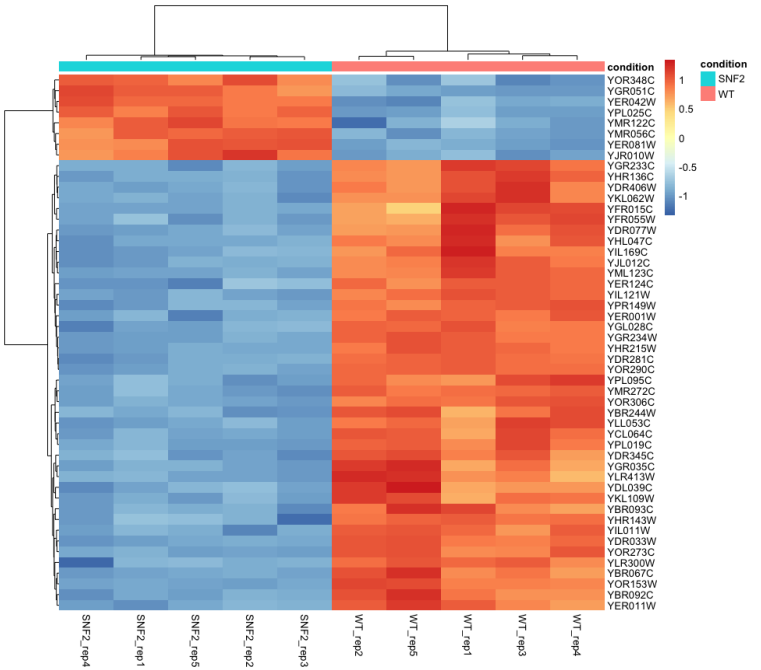
Select by row name using the list of genes:

```
rld_counts_sig <- rld_counts[significant_genes_50, ]
```

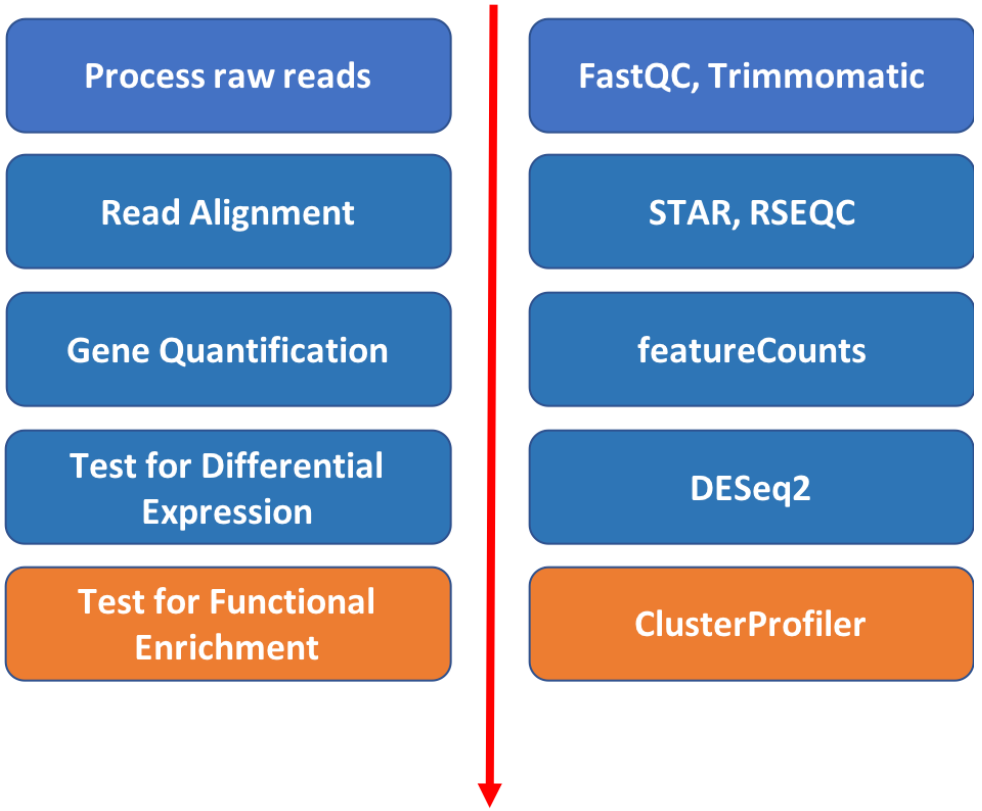
Plot multiple genes in a heatmap

pheatmap has many customizable options
`scale = "row"` - mean center each row and divides by the standard deviation to give [-1,1] values

```
pheatmap(rld_counts_sig,
         cluster_rows = T,
         show_rownames = T,
         annotation = meta,
         border_color = NA,
         fontsize = 10,
         scale = "row",
         fontsize_row = 8,
         height = 20)
```



Note on Functional Enrichment



A great tutorial to follow:
https://hbctraining.github.io/DGE_workshop/lessons/09_functional_analysis.html

What should I do with my files?

Files will remain in the training directory until the end of May, when they will be removed.

If you have a cluster lab storage space, e.g.:

```
/cluster/tufts/<your lab name>/<your user name>/
```

You can copy your work like this:

```
cd /cluster/tufts/bio/tools/training/your-user-name/
cp -r bioinformatics-rnaseq/ /cluster/tufts/your-lab-name/your-user-name/
```

Otherwise, you can try to copy your data into your home directory, if there's space:

```
rm ~/bioinformatics-rnaseq #no trailing slash
cp -r /cluster/tufts/bio/tools/training/your-user-name/bioinformatics-rnaseq/ ~/
```

We'd like to hear from you

tts-research@tufts.edu or rebecca.batorsky@tufts.edu

Short survey by email after course

Thanks for participating!