Lecture 15

RNA-seq: Differential analysis

R and R studio

Programming language: R (https://www.r-project.org)

Integrated Development Environment (IDE) for R: R studio (https://www.rstudio.com/)

DESeq2

Love *et al. Genome Biology* (2014) 15:550
DOI 10.1186/s13059-014-0550-8

Genome Biology

METHOD Open Access

Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2

Michael I Love[1,2,3], Wolfgang Huber[2] and Simon Anders[2*]

Abstract

In comparative high-throughput sequencing assays, a fundamental task is the analysis of count data, such as read counts per gene in RNA-seq, for evidence of systematic changes across experimental conditions. Small replicate numbers, discreteness, large dynamic range and the presence of outliers require a suitable statistical approach. We present *DESeq2*, a method for differential analysis of count data, using shrinkage estimation for dispersions and fold changes to improve stability and interpretability of estimates. This enables a more quantitative analysis focused on the strength rather than the mere presence of differential expression. The *DESeq2* package is available at http://www.bioconductor.org/packages/release/bioc/html/DESeq2.html.

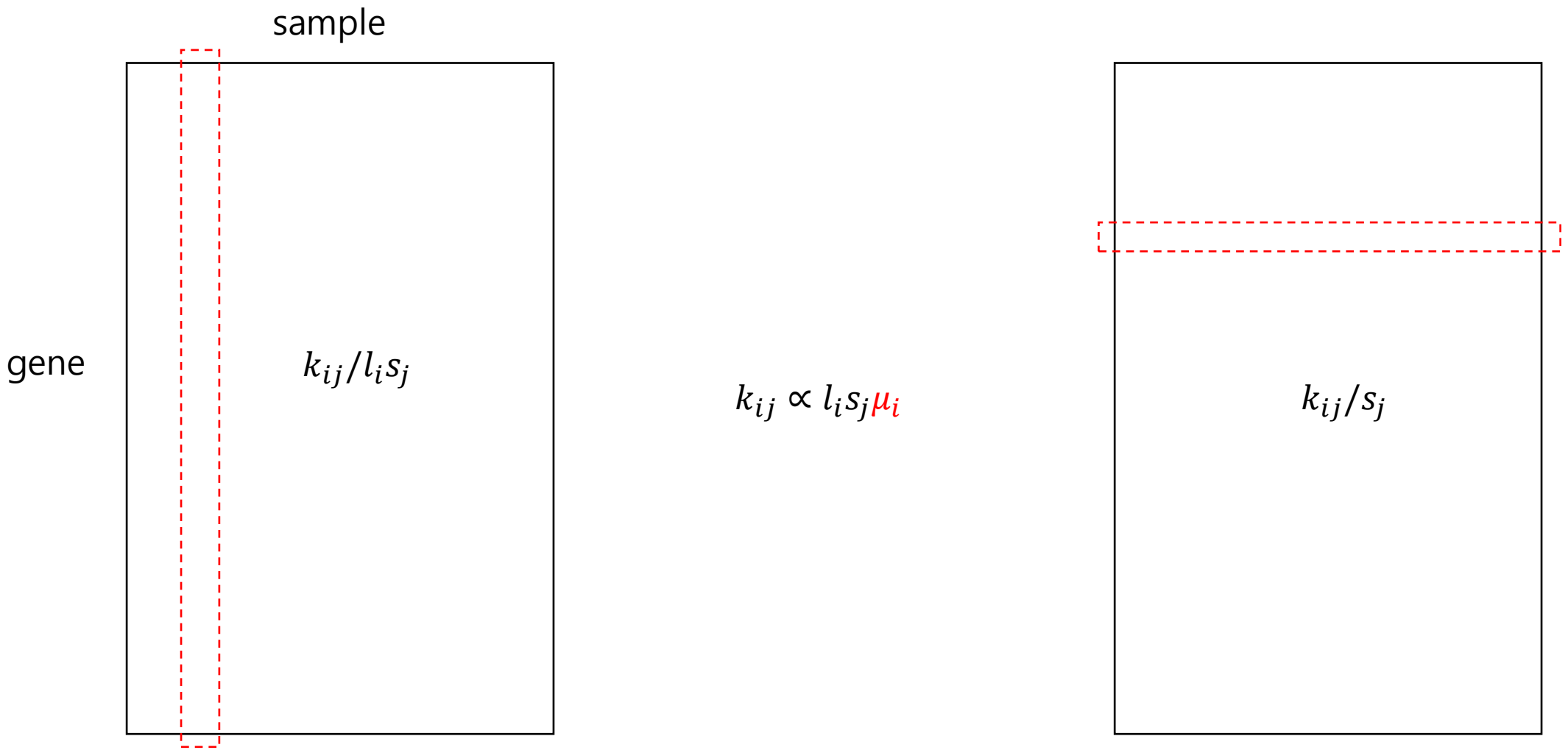
DESeq2 tutorial

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

DESeq2 installation

```
install.packages(c("BiocManager", "ggplot2", "hexbin"))
BiocManager::install(c("DESeq2", "apeglm", "vsn", "pheatmap"))
```

Normalization



Why un-normalized counts?

- As input, the DESeq2 package expects count data as obtained, e.g., from RNA-seq or another high-throughput sequencing experiment, in the form of a matrix of integer values. The value in the i-th row and the j-th column of the matrix tells how many reads can be assigned to gene i in sample j.

- The values in the matrix should be un-normalized counts or estimated counts of sequencing reads (for single-end RNA-seq) or fragments (for paired-end RNA-seq).

- It is important to provide count matrices as input for DESeq2's statistical model (Love, Huber, and Anders 2014) to hold, as only the count values allow assessing the measurement precision correctly.

Sample annotation

directory="your_working_directory"
setwd(directory)

```
colData = read.csv("zika_annot.csv")
head(colData)
```

```
##          Run Layout   Annot Condition
## 1 SRR3191542 PAIRED Mock1-1      Mock
## 2 SRR3191543 PAIRED Mock2-1      Mock
## 3 SRR3191544 PAIRED ZIKV1-1      ZIKA
## 4 SRR3191545 PAIRED ZIKV2-1      ZIKA
## 5 SRR3194428 SINGLE Mock1-2      Mock
## 6 SRR3194429 SINGLE Mock2-2      Mock
```

htseq-count input

```
sampleFiles = grep("htseqcount", list.files(directory), value=TRUE)
sampleTable = data.frame(sampleName = sampleFiles,
                         fileName = sampleFiles,
                         condition = colData[,"Condition"],
                         type = colData[,"Layout"])
head(sampleTable)
```

```
##                   sampleName                     fileName condition   type
## 1 SRR3191542.htseqcount.txt SRR3191542.htseqcount.txt      Mock PAIRED
## 2 SRR3191543.htseqcount.txt SRR3191543.htseqcount.txt      Mock PAIRED
## 3 SRR3191544.htseqcount.txt SRR3191544.htseqcount.txt      ZIKA PAIRED
## 4 SRR3191545.htseqcount.txt SRR3191545.htseqcount.txt      ZIKA PAIRED
## 5 SRR3194428.htseqcount.txt SRR3194428.htseqcount.txt      Mock SINGLE
## 6 SRR3194429.htseqcount.txt SRR3194429.htseqcount.txt      Mock SINGLE
```

htseq-count input

You can use the function *DESeqDataSetFromHTSeqCount* if you have used *htseq-count* from the HTSeq python package.

```
library(DESeq2)
dds = DESeqDataSetFromHTSeqCount(sampleTable = sampleTable,
                                 directory = directory,
                                 design = ~ condition)
dds
```

```
## class: DESeqDataSet
## dim: 58233 8
## metadata(1): version
## assays(1): counts
## rownames(58233): ENSG00000000003 ENSG00000000005 ...
##   ENSG00000284596 ENSG00000284600
## rowData names(0):
## colnames(8): SRR3191542.htseqcount.txt SRR3191543.htseqcount.txt
##   ... SRR3194430.htseqcount.txt SRR3194431.htseqcount.txt
## colData names(1): condition
```

Pre-filtering

- There are two reasons which make pre-filtering useful: by removing rows in which there are no reads or nearly no reads, we reduce the memory size of the `dds` data object and we increase the speed of the transformation and testing functions within DESeq2. Here we perform a minimal pre-filtering to remove rows that have only 0 or 1 read.

```
dds <- dds[ rowSums(counts(dds)) > 1, ]
```

Factor levels

- A factor is a type of vector for which the elements are categorical values.

```
x=c("high", "medium", "low", "high", "medium") # character vector
x
```

```
## [1] "high"   "medium" "low"    "high"   "medium"
```

```
typeof(x)
```

```
## [1] "character"
```

```
class(x)
```

```
## [1] "character"
```

Factor levels

```
xf = factor(x) # convert a character vector to a factor
xf
```

```
## [1] high   medium low    high   medium
## Levels: high low medium
```

```
typeof(xf)
```

```
## [1] "integer"
```

```
class(xf)
```

```
## [1] "factor"
```

```
as.integer(xf) # convert a factor to a numeric vector
```

```
## [1] 1 3 2 1 3
```

Factor levels: reordering

```
xfo = factor(xf, levels=c("low", "medium", "high"), ordered=TRUE)
xfo
```

```
## [1] high   medium low    high   medium
## Levels: low < medium < high
```

Factor levels: relabeling

```
xfol = factor(xf, levels=c("low", "medium", "high"), labels=c("Bottom", "Middle", "Top"), ordered=TRUE)
xfol
```

```
## [1] Top    Middle Bottom Top    Middle
## Levels: Bottom < Middle < Top
```

Factor levels

- By default, R will choose a reference level for factors based on alphabetical order. Then, if you never tell the DESeq2 functions which level you want to compare against (e.g. which level represents the control group), the comparisons will be based on the alphabetical order of the levels.

- Setting the factor levels can be done in two ways, either using factor, or using `relevel()`, just specifying the reference level.

Factor levels

```
dds$condition <- relevel(dds$condition, ref="Mock")
```

DESeq()

- The standard differential expression analysis steps are wrapped into a single function, `DESeq()`.

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

```
## estimating size factors
```

```
## estimating dispersions
```

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

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

```
## final dispersion estimates
```

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

DESeq size factor

$$k_{ij} \propto s_j \mu_i$$

$$\hat{s}_j = \underset{i}{\text{median}} \frac{k_{ij}}{\left(\prod_{v=1}^{m} k_{iv} \right)^{1/m}}$$

Pseudo-reference

results()

- Details about the comparison are printed to the console, above the results table. The text, condition ZIKA vs MOCK, tells you that the estimates are of the logarithmic fold change log2(ZIKA/MOCK).

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

```
## log2 fold change (MLE): condition ZIKA vs Mock
## Wald test p-value: condition ZIKA vs Mock
## DataFrame with 37745 rows and 6 columns
##                    baseMean log2FoldChange     lfcSE       stat
##                   <numeric>      <numeric> <numeric>  <numeric>
## ENSG00000000003 1421.552108     -0.3976904 0.1131562  -3.514527
## ENSG00000000005    2.055105     -0.3114701 1.0882837  -0.286203
## ENSG00000000419  500.457677      0.3174325 0.1077202   2.946825
## ENSG00000000457  304.202948     -0.2473457 0.1245411  -1.986057
## ENSG00000000460  200.481694     -0.4503035 0.1363872  -3.301654
## ...                     ...            ...       ...        ...
## ENSG00000284526   2.3336259     -0.8277006 1.0130100 -0.8170704
## ENSG00000284543   0.1752070     -0.7117568 2.9785790 -0.2389585
## ENSG00000284575   0.4585063     -0.8341866 1.9367736 -0.4307094
## ENSG00000284584   1.1371808      0.8942097 1.5881039  0.5630675
## ENSG00000284600   8.9757314     -0.2565871 0.5162987 -0.4969742
```

Ordering results

- We can order our results table by the smallest adjusted *p* value:

```
resOrdered <- res[order(res$padj),]
resOrdered
```

```
## log2 fold change (MLE): condition ZIKA vs Mock
## Wald test p-value: condition ZIKA vs Mock
## DataFrame with 37745 rows and 6 columns
##                   baseMean log2FoldChange      lfcSE       stat
##                  <numeric>      <numeric>  <numeric>  <numeric>
## ENSG00000070669   4745.372       2.209231 0.06202700   35.61724
## ENSG00000100219   2498.051       2.492401 0.07109604   35.05682
## ENSG00000103257   5290.152       2.492995 0.07395630   33.70903
## ENSG00000112715   2031.744       2.094566 0.06683689   31.33848
## ENSG00000051108   2344.426       2.417485 0.07718640   31.32008
## ...                    ...            ...        ...        ...
## ENSG00000284518 0.07205413     -0.9447827   3.471065 -0.2721882
## ENSG00000284522 1.21336990     -1.3915898   1.409322 -0.9874180
## ENSG00000284543 0.17520702     -0.7117568   2.978579 -0.2389585
## ENSG00000284575 0.45850629     -0.8341866   1.936774 -0.4307094
## ENSG00000284584 1.13718080      0.8942097   1.588104  0.5630675
##                       pvalue          padj
##                    <numeric>     <numeric>
## ENSG00000070669 7.577963e-278 1.751191e-273
## ENSG00000100219 3.069546e-269 3.546706e-265
## ENSG00000103257 4.262237e-249 3.283201e-245
## ENSG00000112715 1.396689e-215 8.069019e-212
## ENSG00000051108 2.486590e-215 1.149252e-211
## ...                       ...           ...
```

More info on results

- Information about which variables and tests were used can be found by calling the function *mcols* on the results object.

```
mcols(res)$description
```

```
## [1] "mean of normalized counts for all samples"
## [2] "log2 fold change (MLE): condition ZIKA vs Mock"
## [3] "standard error: condition ZIKA vs Mock"
## [4] "Wald statistic: condition ZIKA vs Mock"
## [5] "Wald test p-value: condition ZIKA vs Mock"
## [6] "BH adjusted p-values"
```

Note on p-values set to NA

- Some values in the results table can be set to NA for one of the following reasons:
- If within a row, all samples have zero counts, the baseMean column will be zero, and the log2 fold change estimates, p value and adjusted p value will all be set to NA.
- If a row contains a sample with an extreme count outlier then the p value and adjusted p value will be set to NA. These outlier counts are detected by Cook's distance.
- If a row is filtered by automatic independent filtering, for having a low mean normalized count, then only the adjusted p value will be set to NA (https://bioconductor.org/packages/release/bioc/vignettes/DESeq2/inst/doc/DESeq2.html#indfilt).

How many adjusted p-values?

- How many adjusted p-values were less than 0.1?

```
sum(res$padj < 0.1, na.rm=TRUE)
```

```
## [1] 9748
```

summary()

- We can summarize some basic tallies using the summary function.

```
summary(res)
```

```
##
## out of 37745 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)      : 4951, 13%
## LFC < 0 (down)    : 4797, 13%
## outliers [1]      : 0, 0%
## low counts [2]    : 14636, 39%
## (mean count < 2)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

summary()

- Note that the results function automatically performs independent filtering based on the mean of normalized counts for each gene, optimizing the number of genes which will have an adjusted p value below a given FDR cutoff, alpha.

- By default the argument alpha is set to 0.1. If the adjusted p value cutoff will be a value other than 0.1, alpha should be set to that value.

summary()

```
summary(results(dds, alpha=0.05))
```

```
##
## out of 37745 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)      : 4412, 12%
## LFC < 0 (down)    : 4190, 11%
## outliers [1]      : 0, 0%
## low counts [2]    : 15368, 41%
## (mean count < 3)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Exploring results

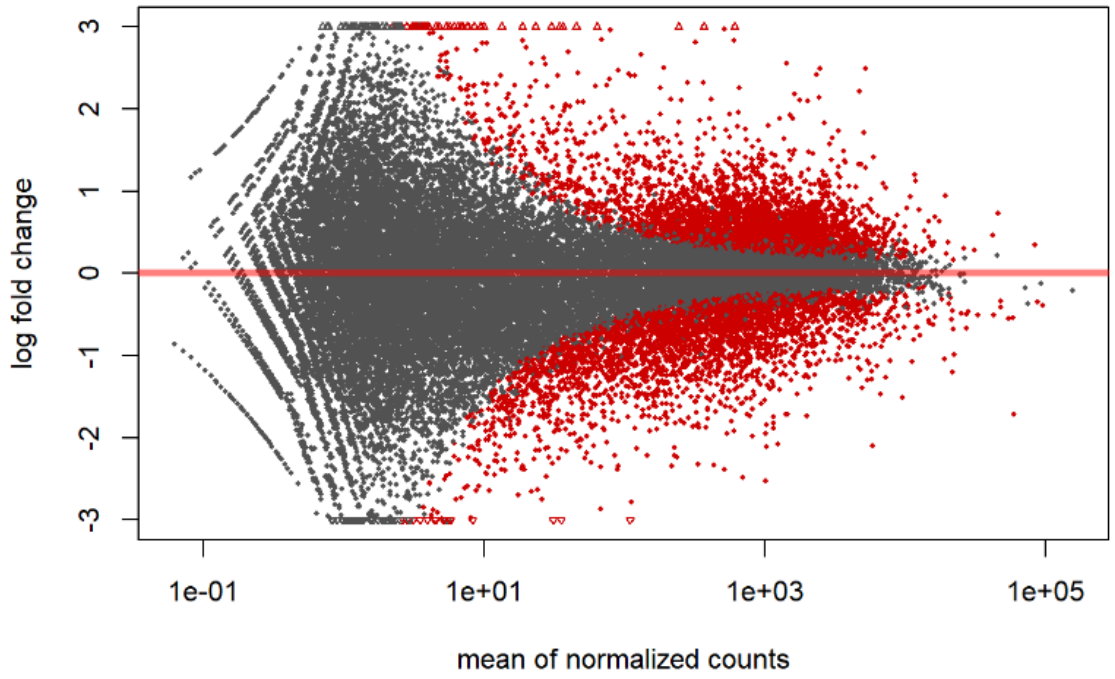
MA-plot

- In DESeq2, the function `plotMA` shows the log2 fold changes attributable to a given variable over the mean of normalized counts for all the samples.

- Points will be colored red if the adjusted *p* value is less than alpha. Points which fall out of the window are plotted as open triangles pointing either up or down.

MA-plot

```
plotMA(res, ylim=c(-3,3), alpha=0.01)
```



MA-plot

- It is also useful to visualize the MA-plot for the shrunken log2 fold changes, which remove the noise associated with log2 fold changes from low count genes without requiring arbitrary filtering thresholds.

- Here we provide the `dds` object and the number of the coefficient we want to moderate.

MA-plot

```
resultsNames(dds)
```

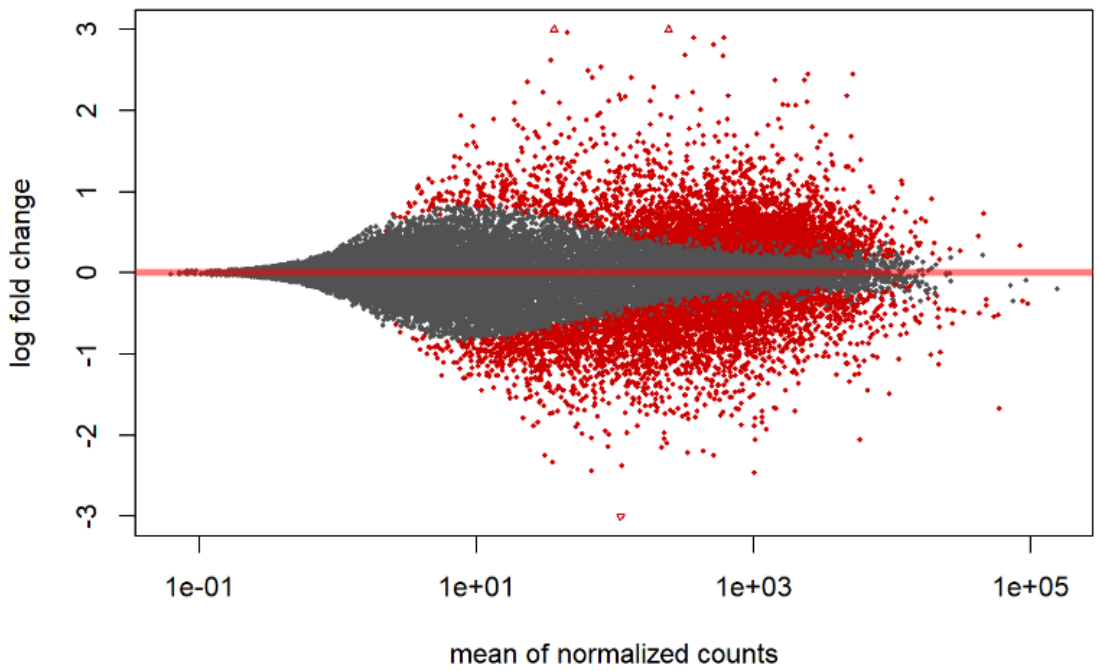
```
## [1] "Intercept"             "condition_ZIKA_vs_Mock"
```

```
resLFC <- lfcShrink(dds, coef=2, res=res)
resLFC
```

```
## log2 fold change (MAP): condition ZIKA vs Mock
## Wald test p-value: condition ZIKA vs Mock
## DataFrame with 37745 rows and 5 columns
##                    baseMean log2FoldChange       stat       pvalue
##                   <numeric>      <numeric>  <numeric>    <numeric>
## ENSG00000000003 1421.552108    -0.38221503  -3.514527 0.0004405386
## ENSG00000000005    2.055105    -0.06327595  -0.286203 0.7747226345
## ENSG00000000419  500.457677     0.30623802   2.946825 0.0032105485
## ENSG00000000457  304.202948    -0.23582646  -1.986057 0.0470270340
## ENSG00000000460  200.481694    -0.42540807  -3.301654 0.0009611646
## ...                     ...            ...        ...          ...
## ENSG00000284526   2.3336259    -0.18513594 -0.8170704    0.4138882
## ENSG00000284543   0.1752070    -0.02107810 -0.2389585    0.8111378
## ENSG00000284575   0.4585063    -0.06806745 -0.4307094    0.6666797
## ENSG00000284584   1.1371808     0.08413480  0.5630675    0.5733889
## ENSG00000284600   8.9757314    -0.13860902 -0.4969742    0.6192073
##                        padj
##                   <numeric>
```

MA-plot

```
plotMA(resLFC, ylim=c(-3,3), alpha=0.01)
```



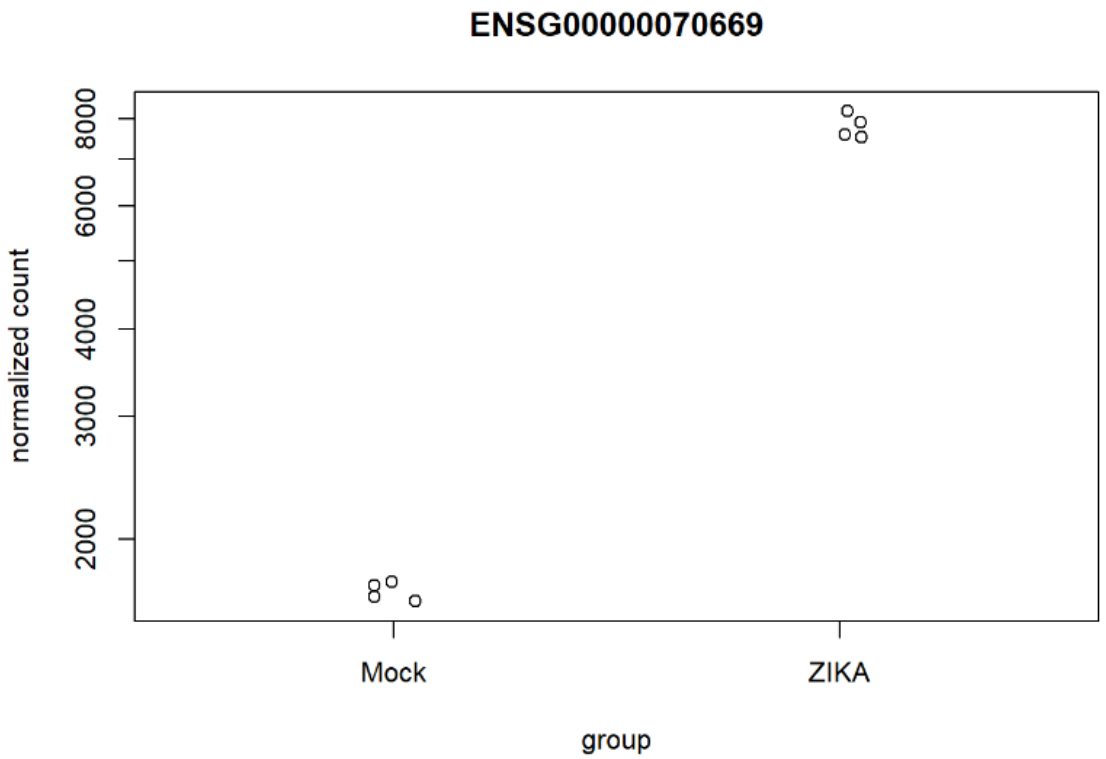
plotCounts()

- It can also be useful to examine the counts of reads for a single gene across the groups. A simple function for making this plot is plotCounts, which normalizes counts by sequencing depth and adds a pseudocount of 1/2 to allow for log scale plotting. The counts are grouped by the variables in intgroup, where more than one variable can be specified.

- Here we specify the gene which had the smallest p value from the results table created above. You can select the gene to plot by rowname or by numeric index.

plotCounts()

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



Exporting results to CSV files

- A plain-text file of the results can be exported using the base R functions *write.csv.*

```
setwd(directory)
write.csv(as.data.frame(resOrdered),
          file="mock_zika_deseq2_results.csv")
```

Data transformations

Count data transformations

- In order to test for differential expression, we operate on raw counts and use discrete distributions differential expression.

- For other downstream analyses – e.g. for visualization or clustering – it might be useful to work with transformed versions of the count data.

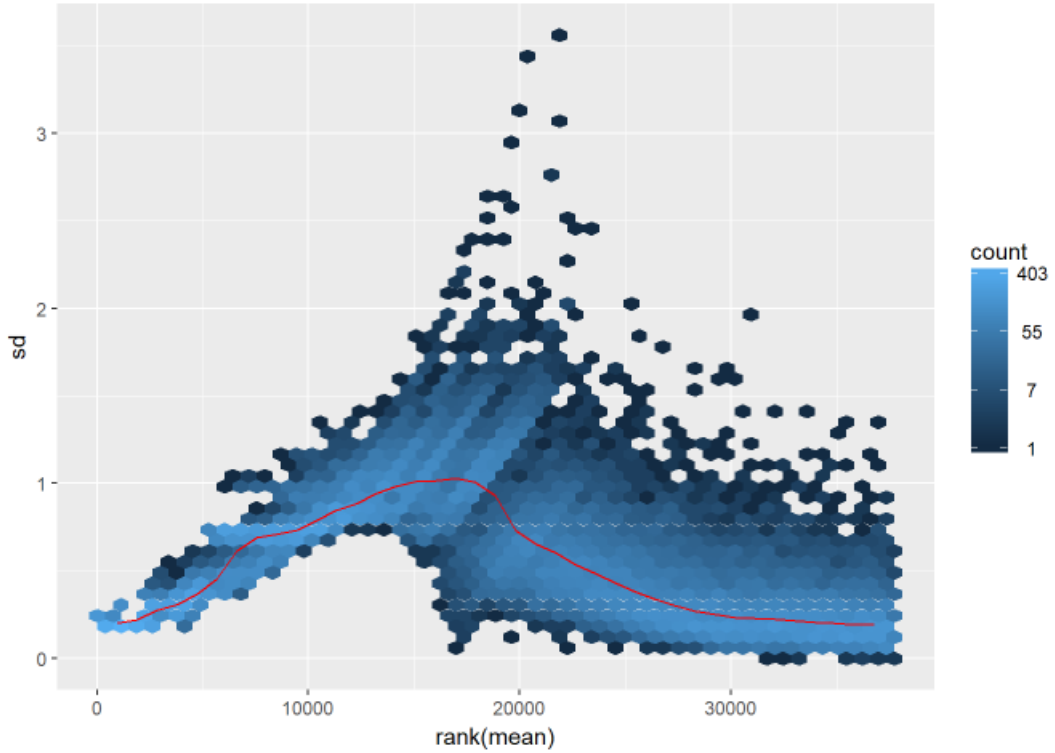
- The most obvious choice of transformation is the logarithm. Since count values for a gene can be zero in some conditions, some advocate the use of *pseudocounts*, i.e. transformations of the form:

$$y = \log_2(n + n_0)$$

where n represents the count values and n_0 is a positive constant.

normTransform(), $\log_2(n+1)$

```
ntd <- normTransform(dds)
library("vsn")
meanSdPlot(assay(ntd))
```



rlog and vst

- We discuss two alternative approaches that offer more theoretical justification and a rational way of choosing the parameter equivalent to n_0.

- The regularized logarithm or rlog incorporates a prior on the sample differences (Love, Huber, and Anders 2014), and the other uses the concept of variance stabilizing transformations (VST) (Tibshirani 1988; Huber et al. 2003; Anders and Huber 2010).

- Both transformations produce transformed data on the log2 scale which has been normalized with respect to library size.

rlog and vst

- The point of these two transformations is to remove the dependence of the variance on the mean, particularly the high variance of the logarithm of count data when the mean is low.

- Both *rlog* and VST use the experiment-wide trend of variance over mean, in order to transform the data to remove the experiment-wide trend.

- Note that we do not require or desire that all the genes have *exactly* the same variance after transformation. Indeed, you will see that after the transformations the genes with the same mean do not have exactly the same standard deviations, but that the experiment-wide trend has flattened.

- It is those genes with variance above the trend which will allow us to cluster samples into interesting groups.

Blind dispersion estimation

- The two functions, rlog and vst have an argument blind, for whether the transformation should be blind to the sample information specified by the design formula.

- When blind equals TRUE (the default), the functions will re-estimate the dispersions using only an intercept. This setting should be used in order to compare samples in a manner wholly unbiased by the information about experimental groups, for example to perform sample QA (quality assurance).

Blind dispersion estimation

- However, blind dispersion estimation is not the appropriate choice if one expects that many or the majority of genes will have large differences in counts which are explainable by the experimental design, and one wishes to transform the data for downstream analysis.

- In this case, using blind dispersion estimation will lead to large estimates of dispersion, as it attributes differences due to experimental design as unwanted noise, and will result in overly shrinking the transformed values towards each other.

- By setting blind to FALSE, the dispersions already estimated will be used to perform transformations, or if not present, they will be estimated using the current design formula.

Extracting transformed values

- The *assay* function is used to extract the matrix of normalized values.

```
rld <- rlog(dds, blind=FALSE)
vsd <- varianceStabilizingTransformation(dds, blind=FALSE)
head(assay(rld), 3)
```

```
##                 SRR3191542.htseqcount.txt SRR3191543.htseqcount.txt
## ENSG00000000003                10.4922343                10.5097223
## ENSG00000000005                 0.9822675                 0.9371416
## ENSG00000000419                 8.7688848                 8.8314214
##                 SRR3191544.htseqcount.txt SRR3191545.htseqcount.txt
## ENSG00000000003                10.1010207                10.254780
## ENSG00000000005                 0.9378057                 1.030696
## ENSG00000000419                 8.9989000                 8.936819
##                 SRR3194428.htseqcount.txt SRR3194429.htseqcount.txt
## ENSG00000000003                10.7971481                10.694273
## ENSG00000000005                 0.9672037                 0.993749
## ENSG00000000419                 8.9122243                 8.854269
##                 SRR3194430.htseqcount.txt SRR3194431.htseqcount.txt
## ENSG00000000003                10.3510408                10.4410640
## ENSG00000000005                 0.9331875                 0.9518496
## ENSG00000000419                 9.1470867                 9.2289348
```

rlog

- The function *rlog*, stands for *regularized log*, transforming the original count data to the log2 scale by fitting a model with a term for each sample and a prior distribution on the coefficients which is estimated from the data.

- The rlog-transformed values are defined as:

$$\log_2 q_{ij} = \beta_{i0} + \beta_{ij}$$

rlog

- q_{ij}: a parameter proportional to the expected true concentration of fragments for gene i and sample j.

- β_{i0}: an intercept which does not undergo shrinkage.

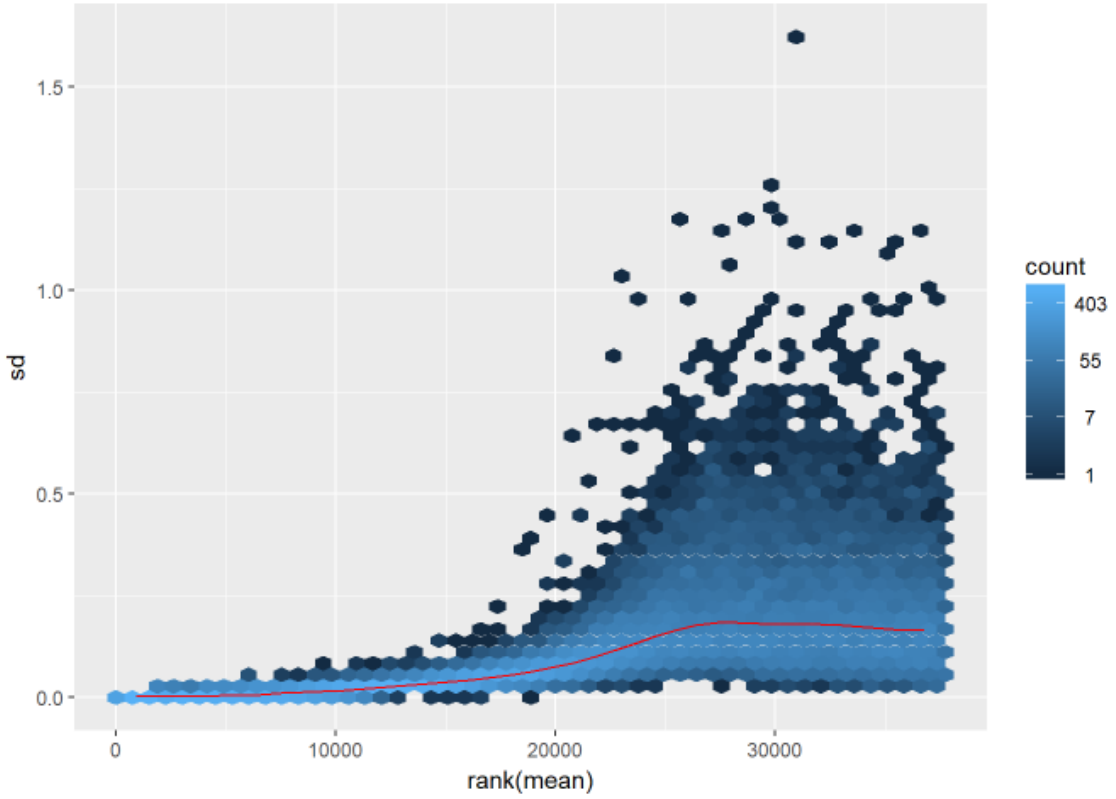
- β_{ij}: the sample-specific effect which is shrunk toward zero based on the dispersion-mean trend over the entire dataset (shrunken LFC).

VST

- The closed-form expression for the variance stabilizing transformation is used by *varianceStabilizingTransformation*.

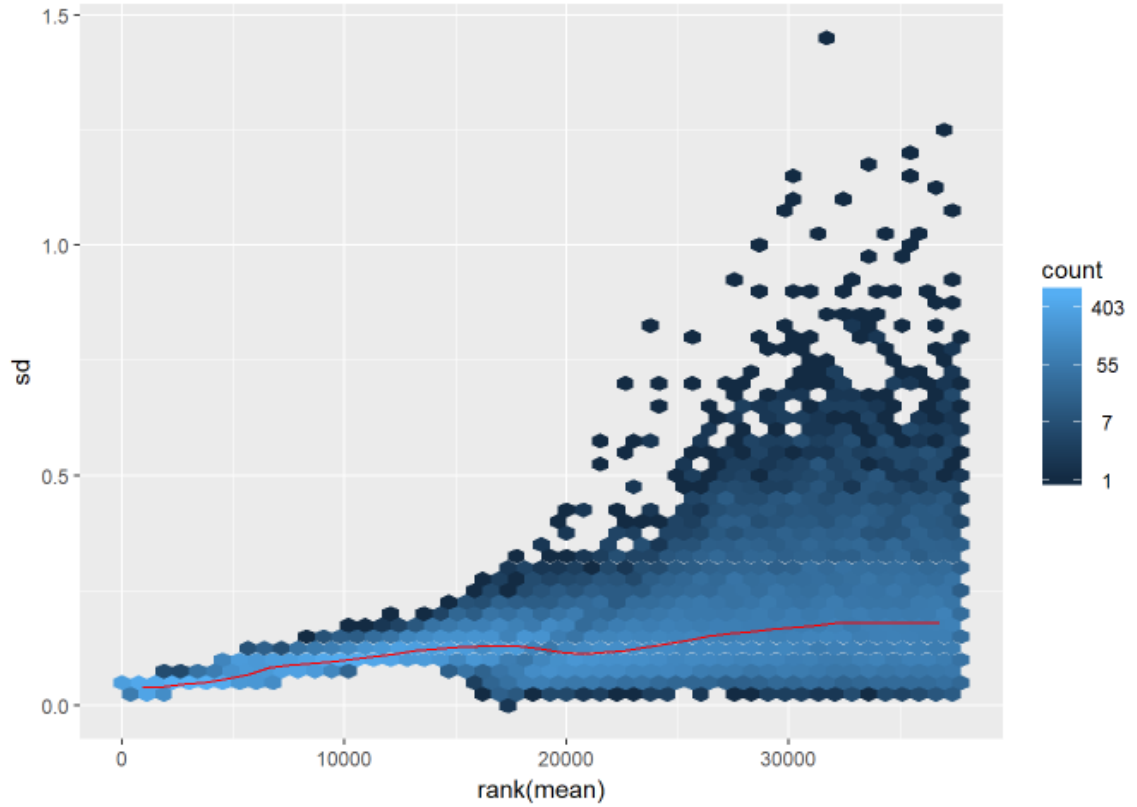
Effects of transformations on the variance

```
meanSdPlot(assay(rld))
```



Effects of transformations on the variance

```
meanSdPlot(assay(vsd))
```



Data visualization

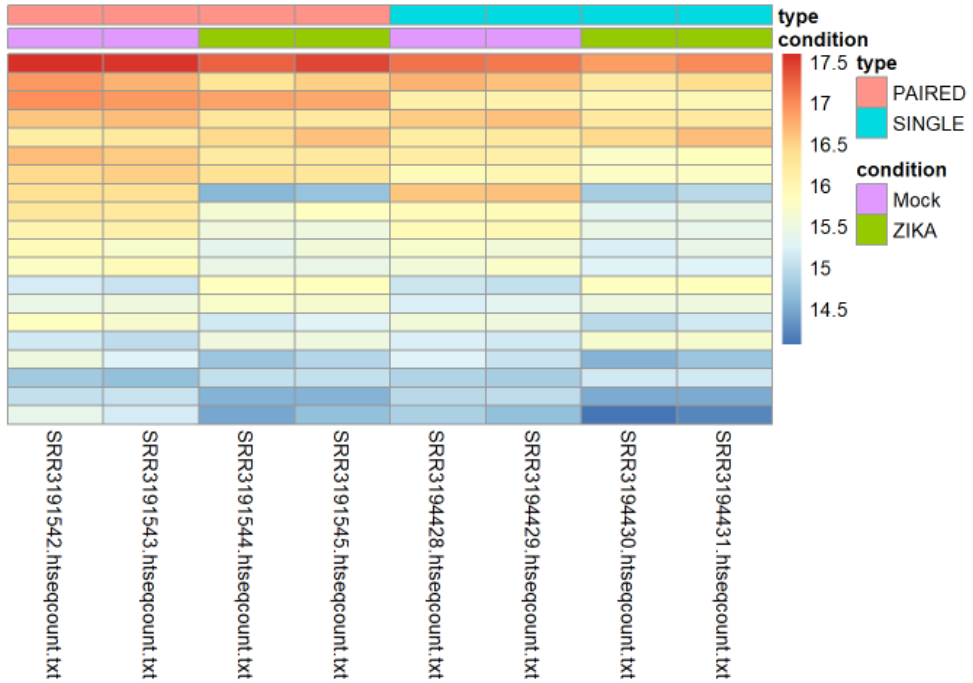
Quality assessment

- Data quality assessment and quality control (i.e. the removal of insufficiently good data) are essential steps of any data analysis. These steps should typically be performed very early in the analysis of a new data set, preceding or in parallel to the differential expression testing.

- We define the term *quality* as *fitness for purpose*. Our purpose is the detection of differentially expressed genes, and we are looking in particular for samples whose experimental treatment suffered from an anormality that renders the data points obtained from these particular samples detrimental to our purpose.

Heatmap of the count matrix

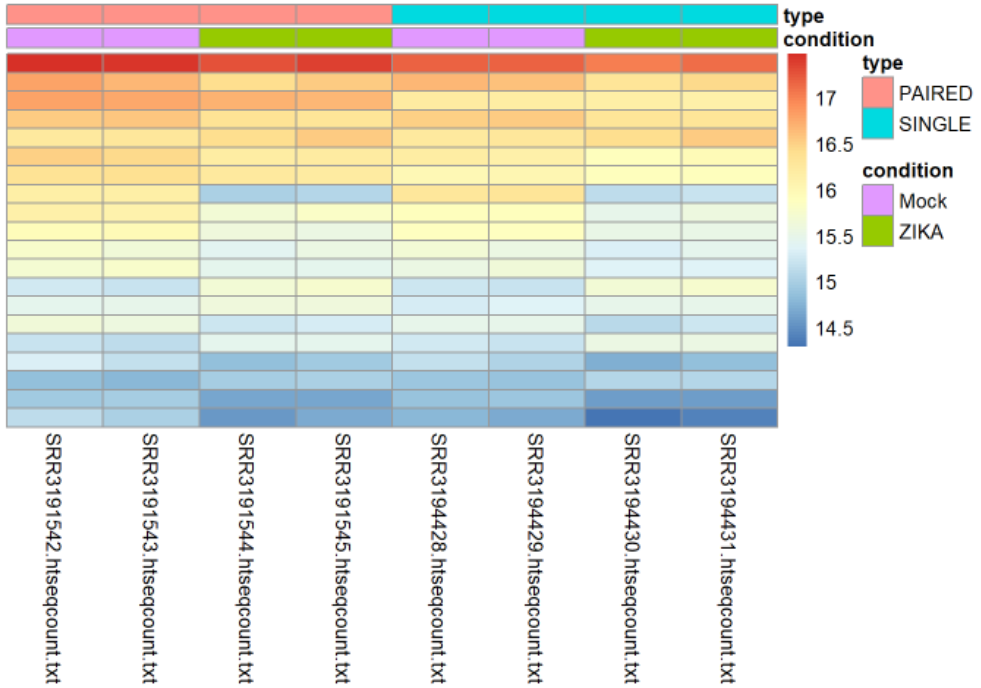
```
library("pheatmap")
select <- order(rowMeans(counts(dds,normalized=TRUE)),
                decreasing=TRUE)[1:20]
df <- as.data.frame(colData(dds)[,c("condition","type")])
pheatmap(assay(ntd)[select,], cluster_rows=FALSE, show_rownames=FALSE,
         cluster_cols=FALSE, annotation_col=df)
```



Heatmap of the count matrix

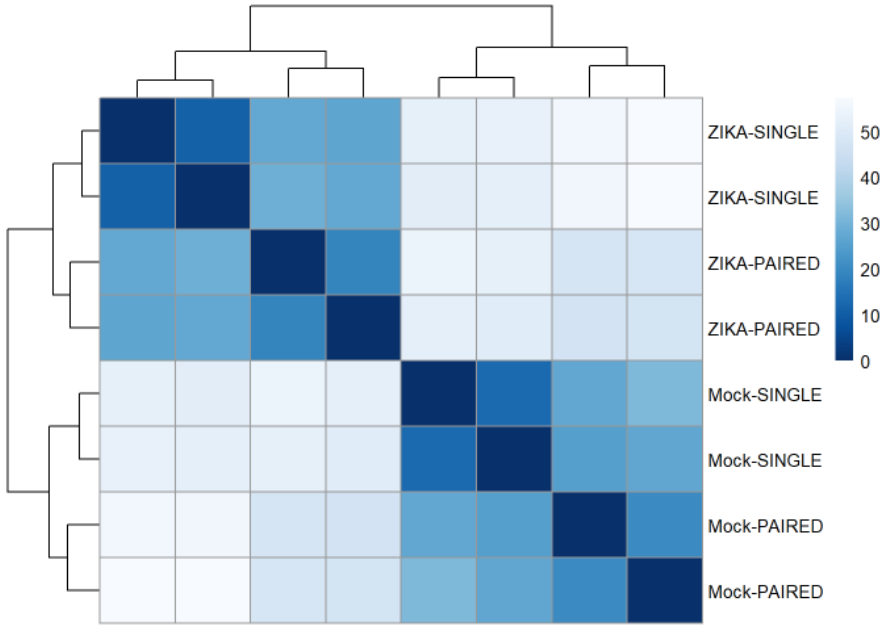
```
rld <- rlog(dds, blind=TRUE)
vsd <- varianceStabilizingTransformation(dds, blind=TRUE)
```

```
pheatmap(assay(rld)[select,], cluster_rows=FALSE, show_rownames=FALSE,
         cluster_cols=FALSE, annotation_col=df)
```



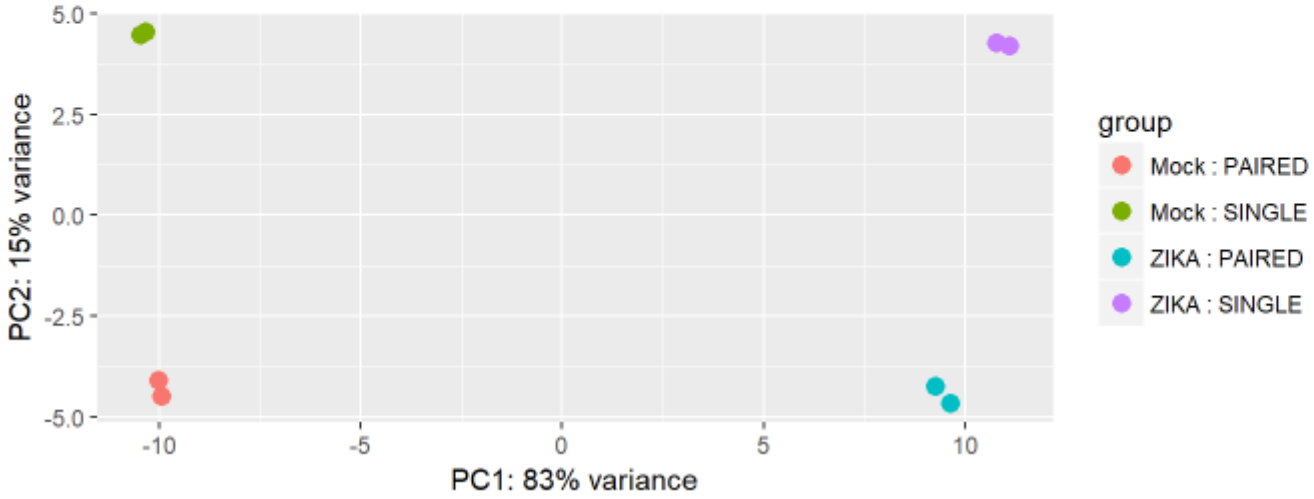
Heatmap of the sample-to-sample distances

```
sampleDists <- dist(t(assay(rld)))
library("RColorBrewer")
sampleDistMatrix <- as.matrix(sampleDists)
rownames(sampleDistMatrix) <- paste(rld$condition, rld$type, sep="-")
colnames(sampleDistMatrix) <- NULL
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)
pheatmap(sampleDistMatrix,
         clustering_distance_rows=sampleDists,
         clustering_distance_cols=sampleDists,
         col=colors)
```



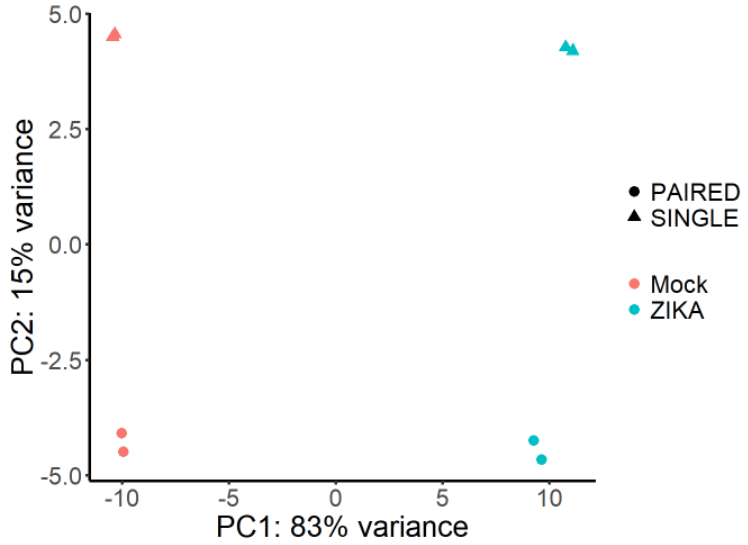
PCA plot of the samples

```
plotPCA(rld, intgroup=c("condition", "type"))
```



PCA plot of the samples

```
library(ggplot2)
pcaData <- plotPCA(rld, intgroup=c("condition", "type"), returnData=TRUE)
percentVar <- round(100 * attr(pcaData, "percentVar"))
ggplot(pcaData, aes(PC1, PC2, color=condition, shape=type)) +
  geom_point(size=3) +
  xlab(paste0("PC1: ",percentVar[1],"% variance")) +
  ylab(paste0("PC2: ",percentVar[2],"% variance")) +
  theme_bw() +
  theme(text = element_text(size=20),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border=element_blank(),
        axis.line=element_line(size=1),
        axis.ticks=element_line(size=1),
        legend.title=element_blank(),
        legend.key=element_blank())
```



Multiple factors

Multiple factors

- Experiments with more than one factor influencing the counts can be analyzed using design formula that include the additional variables.

- In fact, DESeq2 can analyze any possible experimental design that can be expressed with fixed effects terms (multiple factors, designs with interactions, designs with continuous variables, splines, and so on are all possible).

- By adding variables to the design, one can control for additional variation in the counts. For example, if the condition samples are balanced across experimental batches, by including the batch factor to the design, one can increase the sensitivity for finding differences due to condition. There are multiple ways to analyze experiments when the additional variables are of interest and not just controlling factors.

Including type

```
sampleFiles = grep("htseqcount", list.files(directory), value=TRUE)
sampleTable = data.frame(sampleName = sampleFiles,
                         fileName = sampleFiles,
                         condition = colData[,"Condition"],
                         type = colData[,"Layout"])
head(sampleTable)
```

```
##                   sampleName                   fileName condition   type
## 1 SRR3191542.htseqcount.txt SRR3191542.htseqcount.txt      Mock PAIRED
## 2 SRR3191543.htseqcount.txt SRR3191543.htseqcount.txt      Mock PAIRED
## 3 SRR3191544.htseqcount.txt SRR3191544.htseqcount.txt      ZIKA PAIRED
## 4 SRR3191545.htseqcount.txt SRR3191545.htseqcount.txt      ZIKA PAIRED
## 5 SRR3194428.htseqcount.txt SRR3194428.htseqcount.txt      Mock SINGLE
## 6 SRR3194429.htseqcount.txt SRR3194429.htseqcount.txt      Mock SINGLE
```

Accounting for type

- We can account for the different types of sequencing, and get a clearer picture of the differences attributable to the treatment.

- As condition is the variable of interest, we put it at the end of the formula.

- Thus the results function will by default pull the condition results unless contrast or name arguments are specified. Then we can re-run DESeq.

Accounting for type

```
ddsMF = dds
design(ddsMF) <- formula(~ type + condition)
ddsMF <- DESeq(ddsMF)
```

ddsMF$type <- as.factor(ddsMF$type)

```
## using pre-existing size factors
```

```
## estimating dispersions
```

```
## found already estimated dispersions, replacing these
```

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

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

```
## final dispersion estimates
```

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

Accounting for type

```
resMF <- results(ddsMF)
head(resMF)
```

```
## log2 fold change (MLE): condition ZIKA vs Mock
## Wald test p-value: condition ZIKA vs Mock
## DataFrame with 6 rows and 6 columns
##                    baseMean log2FoldChange      lfcSE       stat
##                   <numeric>      <numeric>  <numeric>  <numeric>
## ENSG00000000003 1421.552108     -0.4018334 0.05398398 -7.4435672
## ENSG00000000005    2.055105     -0.4506732 0.91840156 -0.4907147
## ENSG00000000419  500.457677      0.3381494 0.06538730  5.1714847
## ENSG00000000457  304.202948     -0.2213778 0.08730240 -2.5357586
## ENSG00000000460  200.481694     -0.4448332 0.10198927 -4.3615685
## ENSG00000000938    1.457083      2.0019295 1.35399198  1.4785387
##                       pvalue         padj
##                    <numeric>    <numeric>
## ENSG00000000003 9.800219e-14 6.322915e-13
## ENSG00000000005 6.236282e-01 7.459013e-01
## ENSG00000000419 2.322413e-07 9.718264e-07
## ENSG00000000457 1.122040e-02 2.647530e-02
## ENSG00000000460 1.291334e-05 4.602240e-05
## ENSG00000000938 1.392636e-01 2.393810e-01
```

Accounting for type

```
summary(resMF)
```

```
##
## out of 37745 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)      : 6468, 17%
## LFC < 0 (down)    : 6165, 16%
## outliers [1]      : 0, 0%
## low counts [2]    : 12441, 33%
## (mean count < 1)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Accounting for type

- It is also possible to retrieve the log2 fold changes, p values and adjusted p values of the type variable. The contrast argument of the function `results` takes a character vector of length three: the name of the variable, the name of the factor level for the numerator of the log2 ratio, and the name of the factor level for the denominator.

Accounting for type

```
resMFType <- results(ddsMF,
                     contrast=c("type", "SINGLE", "PAIRED"))
summary(resMFType)
```

```
##
## out of 37745 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)      : 4064, 11%
## LFC < 0 (down)    : 3140, 8.3%
## outliers [1]      : 0, 0%
## low counts [2]    : 13172, 35%
## (mean count < 2)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Gene Ontology

Annotating and exporting results

- Our result table only contains information about Ensembl gene IDs, but alternative gene names may be more informative for collaborators. Bioconductor's annotation packages help with mapping various ID schemes to each other.

```
library("AnnotationDbi")
library("org.Hs.eg.db")
```

BiocManager::install(c("topGO", "org.Hs.eg.db"))

```
##
```

```
columns(org.Hs.eg.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"
##  [5] "ENSEMBLTRANS" "ENTREZID"     "ENZYME"       "EVIDENCE"
##  [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"         "ONTOLOGY"
## [17] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"
## [21] "PROSITE"      "REFSEQ"       "SYMBOL"       "UCSCKG"
## [25] "UNIGENE"      "UNIPROT"
```

Annotating and exporting results

```
resMF$symbol <- mapIds(org.Hs.eg.db,
 keys=row.names(resMF),
 column="SYMBOL",
 keytype="ENSEMBL",
 multiVals="first")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
resMF$entrez <- mapIds(org.Hs.eg.db,
 keys=row.names(resMF),
 column="ENTREZID",
 keytype="ENSEMBL",
 multiVals="first")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

Annotating and exporting results

```
resOrdered <- resMF[order(resMF$padj),]
head(resOrdered)
```

```
## log2 fold change (MLE): condition ZIKA vs Mock
## Wald test p-value: condition ZIKA vs Mock
## DataFrame with 6 rows and 8 columns
##                  baseMean log2FoldChange      lfcSE      stat    pvalue
##                 <numeric>      <numeric>  <numeric> <numeric> <numeric>
## ENSG00000051108  2344.426       2.423564 0.05374969  45.08983         0
## ENSG00000070669  4745.372       2.217991 0.03070414  72.23752         0
## ENSG00000090861  3371.892       1.303009 0.03051029  42.70719         0
## ENSG00000092621  4669.460       1.323115 0.03184948  41.54277         0
## ENSG00000100219  2498.051       2.501056 0.03818504  65.49832         0
## ENSG00000103257  5290.152       2.481071 0.03140038  79.01402         0
##                      padj      symbol      entrez
##                 <numeric> <character> <character>
## ENSG00000051108         0     HERPUD1        9709
## ENSG00000070669         0        ASNS         440
## ENSG00000090861         0        AARS          16
## ENSG00000092621         0       PHGDH       26227
## ENSG00000100219         0        XBP1        7494
## ENSG00000103257         0      SLC7A5        8140
```

```
setwd(directory)
write.csv(as.data.frame(resOrdered),
          file="mock_zika_deseq2_genesymbol_results.csv")
```

Running topGO

```
library(topGO)
```

```
## Loading required package: graph
```

```
## Loading required package: GO.db
```

```
##
```

```
## Loading required package: SparseM
```

```
##
## Attaching package: 'SparseM'
```

```
## The following object is masked from 'package:base':
##
##     backsolve
```

```
##
## groupGOTerms:    GOBPTerm, GOMFTerm, GOCCTerm environments built.
```

```
##
## Attaching package: 'topGO'
```

```
## The following object is masked from 'package:IRanges':
##
##     members
```

Running topGO

```
backgroundGenes = rownames(resMF)
targetUPGenes = rownames(resMF[resMF$padj<0.1 & !is.na(resMF$padj) & resMF$log2FoldChange > 0,])
targetDOWNGenes = rownames(resMF[resMF$padj<0.1 & !is.na(resMF$padj) & resMF$log2FoldChange < 0,])
allGeneUP = factor( as.integer( backgroundGenes %in% targetUPGenes ) )
allGeneDOWN = factor( as.integer( backgroundGenes %in% targetDOWNGenes ) )
names(allGeneUP) = backgroundGenes
names(allGeneDOWN) = backgroundGenes
```

Running topGO

```
library(plyr)
onts = c("MF", "BP", "CC")
tab = as.list(onts)
names(tab) = onts
for (i in 1:length(onts)) {
  tgd = new("topGOdata", ontology=onts[i], allGenes=allGeneUP, nodeSize=5,
            annot=annFUN.org, mapping="org.Hs.eg.db", ID="ensembl")

  resultTopGO.elim = runTest(tgd, algorithm="elim", statistic="Fisher")

  tab[[i]] = GenTable(tgd, Fisher.elim = resultTopGO.elim,
                      orderBy="Fisher.classic", topNodes=200)
}
topGOResults = rbind.fill(tab)
setwd(directory)
write.csv(topGOResults, file="Upregulated_topGOResults.csv")
```

Running topGO

```
library(plyr)
onts = c("MF", "BP", "CC")
tab = as.list(onts)
names(tab) = onts
for (i in 1:length(onts)) {
  tgd = new("topGOdata", ontology=onts[i], allGenes=allGeneDOWN, nodeSize=5,
            annot=annFUN.org, mapping="org.Hs.eg.db", ID="ensembl")

  resultTopGO.elim = runTest(tgd, algorithm="elim", statistic="Fisher")

  tab[[i]] = GenTable(tgd, Fisher.elim = resultTopGO.elim,
                      orderBy="Fisher.classic", topNodes=200)
}
topGOResults = rbind.fill(tab)
setwd(directory)
write.csv(topGOResults, file="Downregulated_topGOResults.csv")
```

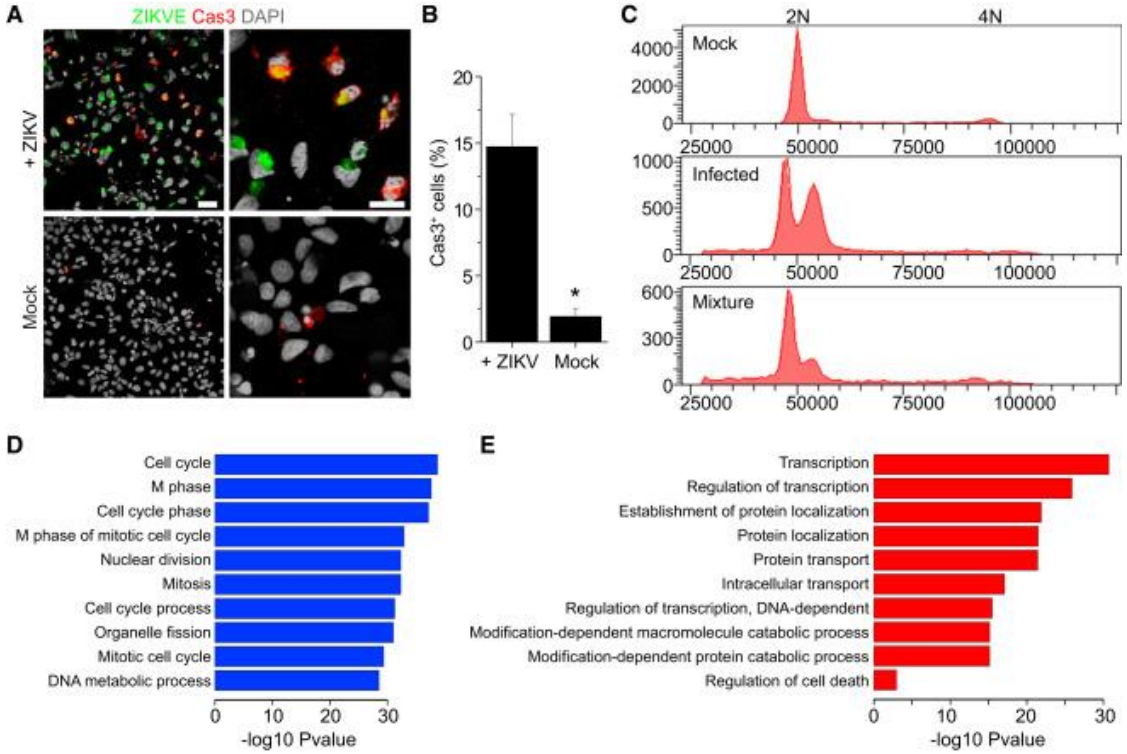
Downregulated GO

GO.ID	Term	Annotatec	Significan	Expected	Fisher.elim
GO:00070	mitotic nuclear division	449	212	117.75	1.10E-13
GO:00513	cell division	548	228	143.71	5.70E-13
GO:00007	condensed chromosome kinetochore	101	66	26.44	9.80E-12
GO:00070	sister chromatid cohesion	123	71	32.26	3.60E-11
GO:00062	DNA replication initiation	39	30	10.23	5.20E-11
GO:00977	ciliary basal body docking	96	54	25.18	4.20E-10
GO:00432	lysosomal lumen	89	49	23.3	6.70E-09
GO:00009	spindle pole	131	65	34.29	7.30E-09
GO:00007	double-strand break repair via homologou...	101	56	26.49	1.70E-08
GO:00000	G2/M transition of mitotic cell cycle	233	104	61.1	6.20E-08
GO:00007	telomere maintenance via recombination	36	25	9.44	7.00E-08
GO:00361	inner dynein arm assembly	12	12	3.15	1.00E-07
GO:00362	interstrand cross-link repair	49	30	12.85	2.60E-07
GO:00425	MCM complex	11	11	2.88	3.90E-07
GO:00055	proteinaceous extracellular matrix	341	149	89.26	6.70E-07
GO:00700	extracellular exosome	2531	761	662.52	9.20E-07
GO:00726	mitotic spindle	66	35	17.28	3.00E-06
GO:00431	single-stranded DNA-dependent ATPase act...	12	11	3.14	3.50E-06
GO:00340	CENP-A containing nucleosome assembly	39	24	10.23	3.60E-06
GO:00009	condensed chromosome outer kinetochore	12	11	3.14	3.60E-06
GO:00060	glycosaminoglycan biosynthetic process	116	53	30.42	4.60E-06
GO:00007	DNA synthesis involved in DNA repair	72	41	18.88	5.70E-06
GO:00062	DNA strand elongation involved in DNA re...	16	13	4.2	6.70E-06
GO:00056	basement membrane	91	47	23.82	8.20E-06
GO:00314	nucleosomal DNA binding	29	19	7.59	9.80E-06
GO:00427	DNA damage response, detection of DNA da...	36	22	9.44	1.10E-05
GO:00062	base-excision repair	41	24	10.75	1.20E-05
GO:00007	strand displacement	27	18	7.08	1.20E-05

Upregulated GO

GO.ID	Term	Annotatec	Significan	Expected	Fisher.elim
GO:0005654	nucleoplasm	3217	1429	1067.41	5.20E-25
GO:0005634	nucleus	6645	2718	2204.84	4.20E-20
GO:0016607	nuclear speck	357	192	118.45	4.70E-16
GO:0005515	protein binding	10235	3811	3471.89	2.50E-15
GO:0061630	ubiquitin protein ligase activity	197	120	66.83	6.40E-15
GO:0004674	protein serine/threonine kinase activity	442	227	149.93	1.50E-14
GO:0003723	RNA binding	1565	690	530.88	1.00E-12
GO:0006355	regulation of transcription, DNA-templat...	3376	1426	1137.93	1.90E-12
GO:0005524	ATP binding	1436	598	487.12	9.00E-11
GO:0048471	perinuclear region of cytoplasm	620	280	205.72	1.70E-10
GO:0008270	zinc ion binding	1039	443	352.45	8.80E-10
GO:0003677	DNA binding	2313	942	784.61	2.00E-09
GO:0000122	negative regulation of transcription fro...	713	314	240.33	2.80E-09
GO:0005730	nucleolus	886	370	293.98	2.50E-08
GO:0018105	peptidyl-serine phosphorylation	264	134	88.99	3.40E-08
GO:0004843	thiol-dependent ubiquitin-specific prote...	76	49	25.78	4.90E-08
GO:0007030	Golgi organization	101	69	34.04	6.40E-08
GO:0043161	proteasome-mediated ubiquitin-dependent ...	376	194	126.74	6.80E-08
GO:0014069	postsynaptic density	194	100	64.37	8.50E-08
GO:0032922	circadian regulation of gene expression	54	37	18.2	1.80E-07
GO:0006886	intracellular protein transport	1113	486	375.15	2.40E-07
GO:0016567	protein ubiquitination	803	386	270.66	2.50E-07
GO:0006511	ubiquitin-dependent protein catabolic pr...	542	281	182.69	2.70E-07
GO:0043130	ubiquitin binding	108	62	36.64	4.50E-07
GO:0000209	protein polyubiquitination	285	148	96.06	6.10E-07
GO:0006888	ER to Golgi vesicle-mediated transport	171	102	57.64	7.70E-07
GO:0043022	ribosome binding	50	34	16.96	8.80E-07
GO:0000151	ubiquitin ligase complex	286	137	94.9	8.80E-07
GO:0016239	positive regulation of macroautophagy	55	36	18.54	1.50E-06
GO:0003700	transcription factor activity, sequence-...	1135	473	385.01	1.70E-06
GO:0036498	IRE1-mediated unfolded protein response	66	41	22.25	2.10E-06

Published results



The top 10 most significant terms are shown for downregulated (D) and upregulated (E) genes, respectively.