ML_DESeq2_workshop

2026-02-09

DESeq2 Workshop - Machine Learning in Practice

Welcome to this practical workshop on analyzing bulk RNA-seq data using DESeq2. This tutorial demonstrates how the machine learning concepts we've discussed today are applied in real bioinformatics workflows.

Workshop Overview

In this workshop, we will apply several ML tools to analyze RNA-sequencing data:

- **Data transformation** with Variance Stabilizing Transformation (VST) to make data suitable for PCA and clustering
- **PCA** for quality control, visualizing sample relationships, and checking for batch effects
- **Hierarchical clustering** on gene expression heatmaps to identify clustering of genes and samples
- **Statistical testing** with DESeq2 for differentially expressed genes (based on generalized linear models)
- **Visualization** of the results with volcano plots and heatmaps

Getting Started

Please download the DESeq2_workshop.Rmd from GitHub (https://github.com/GustawEriksson/Machine_Learning_introduction_2023 (https://github.com/GustawEriksson/Machine_Learning_introduction_2023)) and open it in RStudio.

If R and RStudio are not installed: - Install R from: https://www.r-project.org/ (https://www.r-project.org/) - Install RStudio from: https://posit.co/download/rstudio-desktop/ (https://posit.co/download/rstudio-desktop/)

After opening DESeq2_workshop.Rmd, install BiocManager (done automatically) and the required packages by changing the `install.packages.flag` from TRUE to FALSE.

Thereafter, run all the code sections to generate output.

About DESeq2

DESeq2 is the most widely used method for analyzing bulk RNA-sequencing data. Other methods include limma and edgeR, but all share the common aim of identifying differentially expressed genes (and can be extended to proteins, lipids, etc.).

For a comprehensive introduction, see the excellent DESeq2 vignette: http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html (http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html)

The Dataset

The data used in this vignette is from an experiment on *Drosophila melanogaster* cell cultures investigating the effect of RNAi knock-down of the splicing factor pasilla (Brooks et al. 2011). This is a classic dataset perfect for learning the DESeq2 workflow.

Loading the Packages

The first thing is to load the required packages.

```
## Warning: package 'readr' was built under R version 4.4.3
```

```
## Loading required package: DEXSeq
```

```
## Loading required package: BiocParallel
```

```
## Loading required package: Biobase
```

```
## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, table,
##     tapply, union, unique, unsplit, which.max, which.min
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
## Loading required package: SummarizedExperiment
```

```
## Loading required package: MatrixGenerics
```

```
## Loading required package: matrixStats
```

```
## Warning: package 'matrixStats' was built under R version 4.4.1
```

```
##
## Attaching package: 'matrixStats'
```

```
## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians
```

```
##
## Attaching package: 'MatrixGenerics'
```

```
## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars
```

```
## The following object is masked from 'package:Biobase':
##
##     rowMedians
```

```
## Loading required package: GenomicRanges
```

```
## Warning: package 'GenomicRanges' was built under R version 4.4.1
```

```
## Loading required package: stats4
```

```
## Loading required package: S4Vectors
```

```
## Warning: package 'S4Vectors' was built under R version 4.4.1
```

```
##
## Attaching package: 'S4Vectors'
```

```
## The following object is masked from 'package:utils':
##
##     findMatches
```

```
## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
```

```
## Loading required package: IRanges
```

```
## Warning: package 'IRanges' was built under R version 4.4.1
```

```
## Loading required package: GenomeInfoDb
```

```
## Loading required package: DESeq2
```

```
## Loading required package: AnnotationDbi
```

```
## Loading required package: RColorBrewer
```

```
## Warning: package 'pheatmap' was built under R version 4.4.1
```

```
## Warning: package 'ggplot2' was built under R version 4.4.3
```

```
## Loading required package: ggrepel
```

```
## Warning: package 'ggrepel' was built under R version 4.4.1
```

Importing the Data and Generating the DESeq Dataset

DESeq2 allows for quick and easy import of test data. We'll load the pasilla dataset that comes with the package.

Let's examine the count matrix (cts) and the column data (i.e., sample metadata/groups):

```
head(cts,10)
```

```
##             untreated1 untreated2 untreated3 untreated4 treated1 treated2
## FBgn0000003          0          0          0          0        0        0
## FBgn0000008         92        161         76         70      140       88
## FBgn0000014          5          1          0          0        4        0
## FBgn0000015          0          2          1          2        1        0
## FBgn0000017       4664       8714       3564       3150     6205     3072
## FBgn0000018        583        761        245        310      722      299
## FBgn0000022          0          1          0          0        0        0
## FBgn0000024         10         11          3          3       10        7
## FBgn0000028          0          1          0          0        0        1
## FBgn0000032       1446       1713        615        672     1698      696
##             treated3
## FBgn0000003        1
## FBgn0000008       70
## FBgn0000014        0
## FBgn0000015        0
## FBgn0000017     3334
## FBgn0000018      308
## FBgn0000022        0
## FBgn0000024        5
## FBgn0000028        1
## FBgn0000032      757
```

```
nrow(cts)
```

```
## [1] 14599
```

```
coldata
```

```
##              condition        type
## treated1fb     treated single-read
## treated2fb     treated  paired-end
## treated3fb     treated  paired-end
## untreated1fb untreated single-read
## untreated2fb untreated single-read
## untreated3fb untreated  paired-end
## untreated4fb untreated  paired-end
```

Important Note: The samples are not in the same order in the count matrix and metadata!

It is **absolutely critical** that the columns of the count matrix and the rows of the column data are in the same order. If they are not in the correct order, we need to re-arrange one or the other so that they are consistent in terms of sample order (if we do not, later functions would produce an error). We additionally need to chop off the "fb" prefix from the row names of coldata to make the naming consistent.

```
rownames(coldata) <- sub("fb", "", rownames(coldata))
all(rownames(coldata) %in% colnames(cts))
```

```
## [1] TRUE
```

```
all(rownames(coldata) == colnames(cts))
```

```
## [1] FALSE
```

```
cts <- cts[, rownames(coldata)]
all(rownames(coldata) == colnames(cts))
```

```
## [1] TRUE
```

Let us look at the formatted cts and coldata

```
head(cts,10)
```

```
##             treated1 treated2 treated3 untreated1 untreated2 untreated3
## FBgn0000003        0        0        1          0          0          0
## FBgn0000008      140       88       70         92        161         76
## FBgn0000014        4        0        0          5          1          0
## FBgn0000015        1        0        0          0          2          1
## FBgn0000017     6205     3072     3334       4664       8714       3564
## FBgn0000018      722      299      308        583        761        245
## FBgn0000022        0        0        0          0          1          0
## FBgn0000024       10        7        5         10         11          3
## FBgn0000028        0        1        1          0          1          0
## FBgn0000032     1698      696      757       1446       1713        615
##             untreated4
## FBgn0000003          0
## FBgn0000008         70
## FBgn0000014          0
## FBgn0000015          2
## FBgn0000017       3150
## FBgn0000018        310
## FBgn0000022          0
## FBgn0000024          3
## FBgn0000028          0
## FBgn0000032        672
```

```
nrow(cts)
```

```
## [1] 14599
```

```
coldata
```

```
##            condition        type
## treated1     treated single-read
## treated2     treated  paired-end
## treated3     treated  paired-end
## untreated1 untreated single-read
## untreated2 untreated single-read
## untreated3 untreated  paired-end
## untreated4 untreated  paired-end
```

The count matrix is now in the correct order, and we can generate the DESeq dataset object:

```
dds <- DESeqDataSetFromMatrix(countData = cts,
                              colData = coldata,
                              design = ~ condition)
dds
```

```
## class: DESeqDataSet
## dim: 14599 7
## metadata(1): version
## assays(1): counts
## rownames(14599): FBgn0000003 FBgn0000008 ... FBgn0261574 FBgn0261575
## rowData names(0):
## colnames(7): treated1 treated2 ... untreated3 untreated4
## colData names(2): condition type
```

Prefiltering the Data

While not strictly required, prefiltering is good practice. It reduces memory usage and computing time, and can also help in visualization by removing “noise” from genes with very low counts.

```
keep <- rowSums(counts(dds)) >= 10
dds <- dds[keep,]
dds
```

```
## class: DESeqDataSet
## dim: 9921 7
## metadata(1): version
## assays(1): counts
## rownames(9921): FBgn0000008 FBgn0000014 ... FBgn0261574 FBgn0261575
## rowData names(0):
## colnames(7): treated1 treated2 ... untreated3 untreated4
## colData names(2): condition type
```

Setting Factor Levels

R chooses the reference level for differential gene expression analysis by alphabetical order. Therefore, you should set the factor level manually to ensure the comparison is in the direction you want (treated vs. untreated, not untreated vs. treated).

```
dds$condition <- factor(dds$condition, levels = c("untreated","treated"))
```

Data Transformation and Visualization

Important: Differential gene expression analysis is performed on **raw counts**. However, for visualization and some quality control steps, we need to use **transformed data**.

Why Transform?

Raw count data has variance that depends on the mean - genes with higher expression have higher variance. This heteroscedasticity can make visualization and clustering misleading. The transformations below stabilize the variance across the range of expression levels.

Transformation Methods

We'll demonstrate three transformation methods:

1. **normTransform** - Simple log2(n+1) transformation with pseudocount
2. **VST** (Variance Stabilizing Transformation) - Stabilizes variance across mean
3. **rlog** (Regularized log transformation) - Similar to VST but more robust for small sample sizes

The point of VST and rlog is to remove the dependence of variance on the mean, particularly the high variance of log-transformed count data when the mean is low. Both use the experiment-wide trend of variance over mean to transform the data and remove this trend.

For most applications with moderate to large sample sizes (n ≥ 30), VST is recommended as it's faster than rlog.

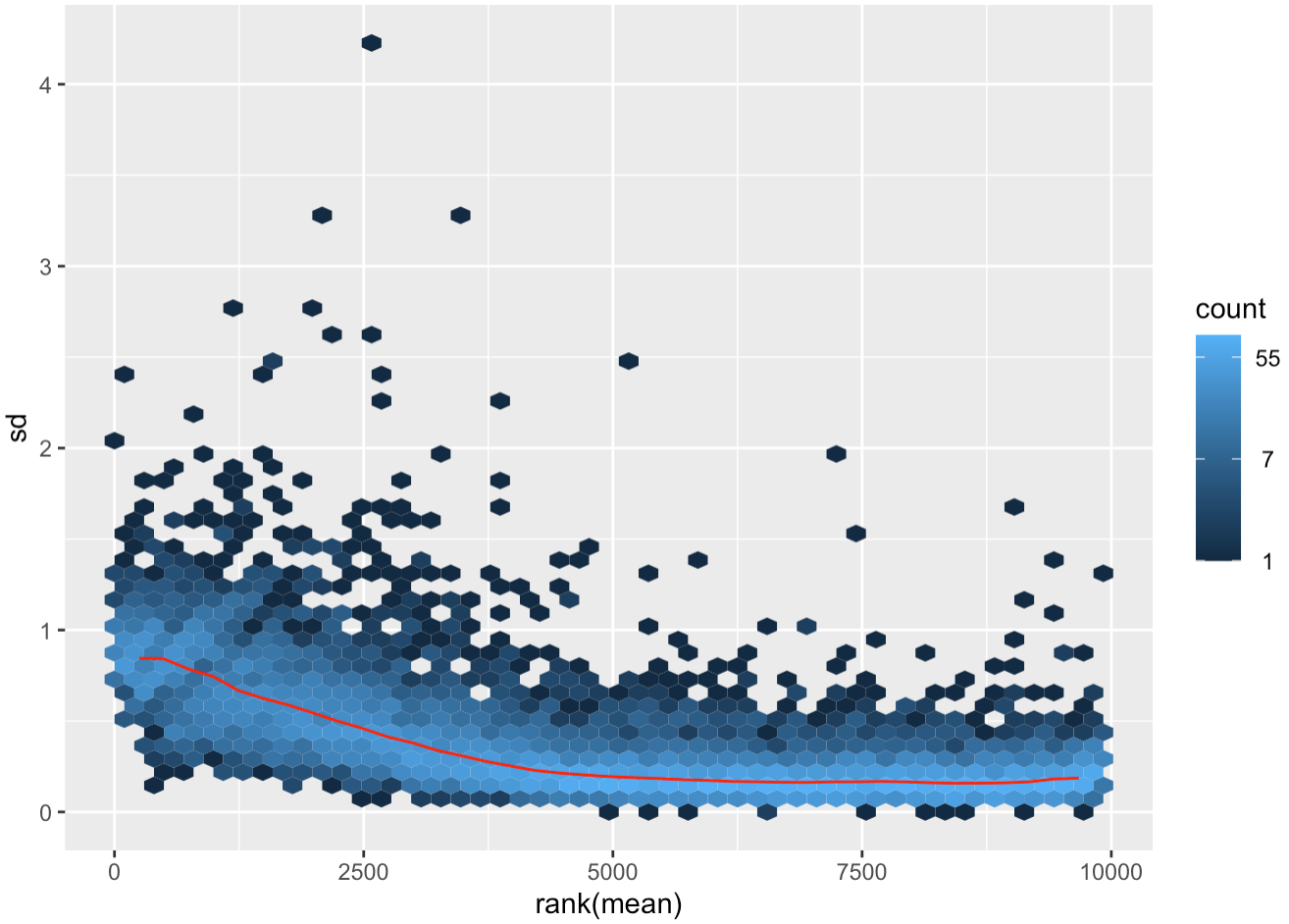
```
##              treated1  treated2  treated3 untreated1 untreated2 untreated3
## FBgn0000008  7.607917  7.834912  7.595052   7.567298   7.642174   7.844603
## FBgn0000014  6.318818  6.041221  6.041221   6.412782   6.173921   6.041221
## FBgn0000017 11.938311 12.024557 12.013565  12.045721  12.284647  12.455939
##             untreated4
## FBgn0000008   7.669147
## FBgn0000014   6.041221
## FBgn0000017  12.077404
```

```
##             treated1 treated2 treated3 untreated1 untreated2 untreated3
## FBgn0000008      140       88       70         92        161         76
## FBgn0000014        4        0        0          5          1          0
## FBgn0000017     6205     3072     3334       4664       8714       3564
##             untreated4
## FBgn0000008         70
## FBgn0000014          0
## FBgn0000017       3150
```

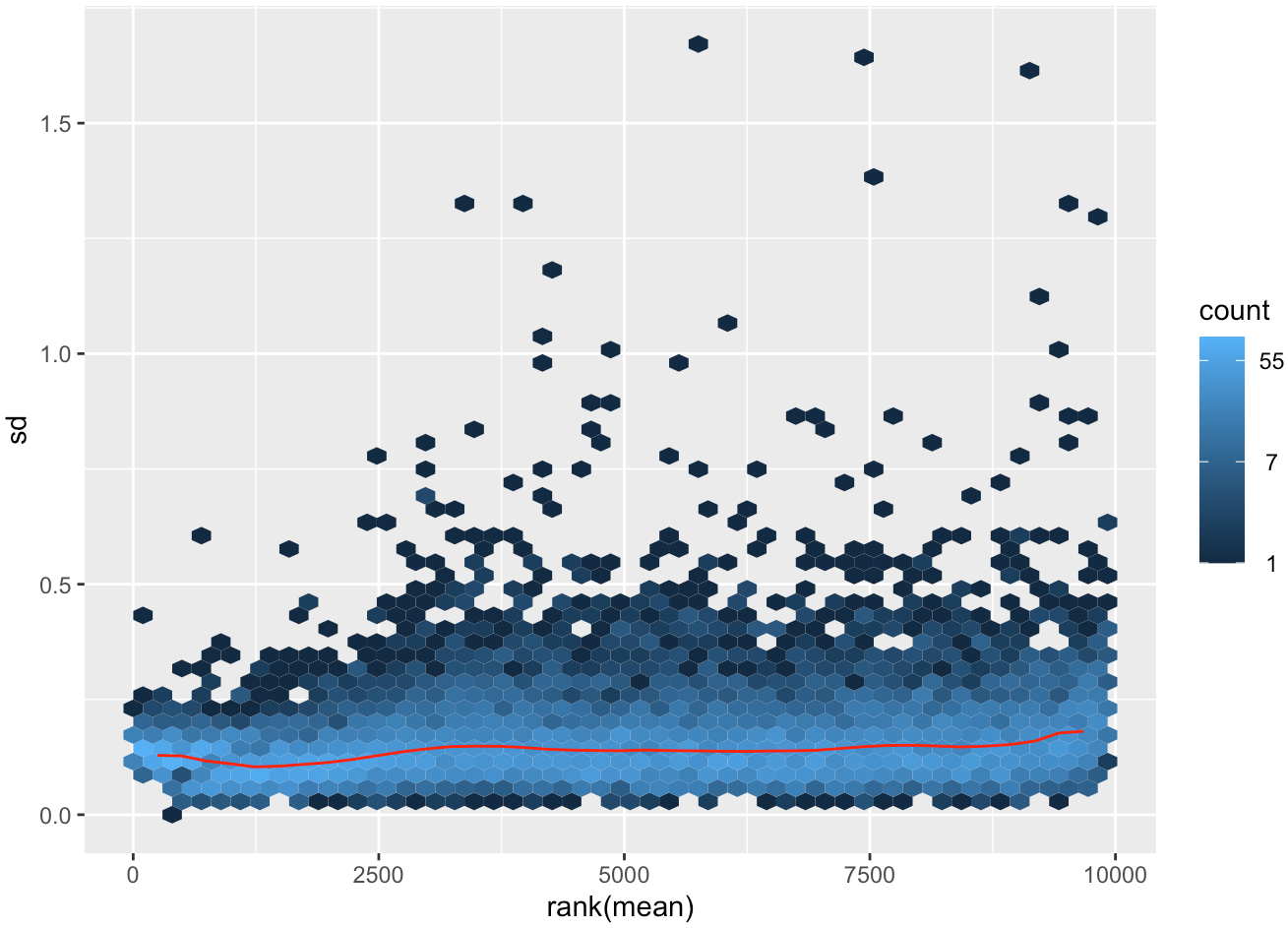
Let's visualize how well each transformation stabilizes variance by plotting the standard deviation of transformed data across all samples against the mean:

```
meanSdPlot(assay(ntd))
```

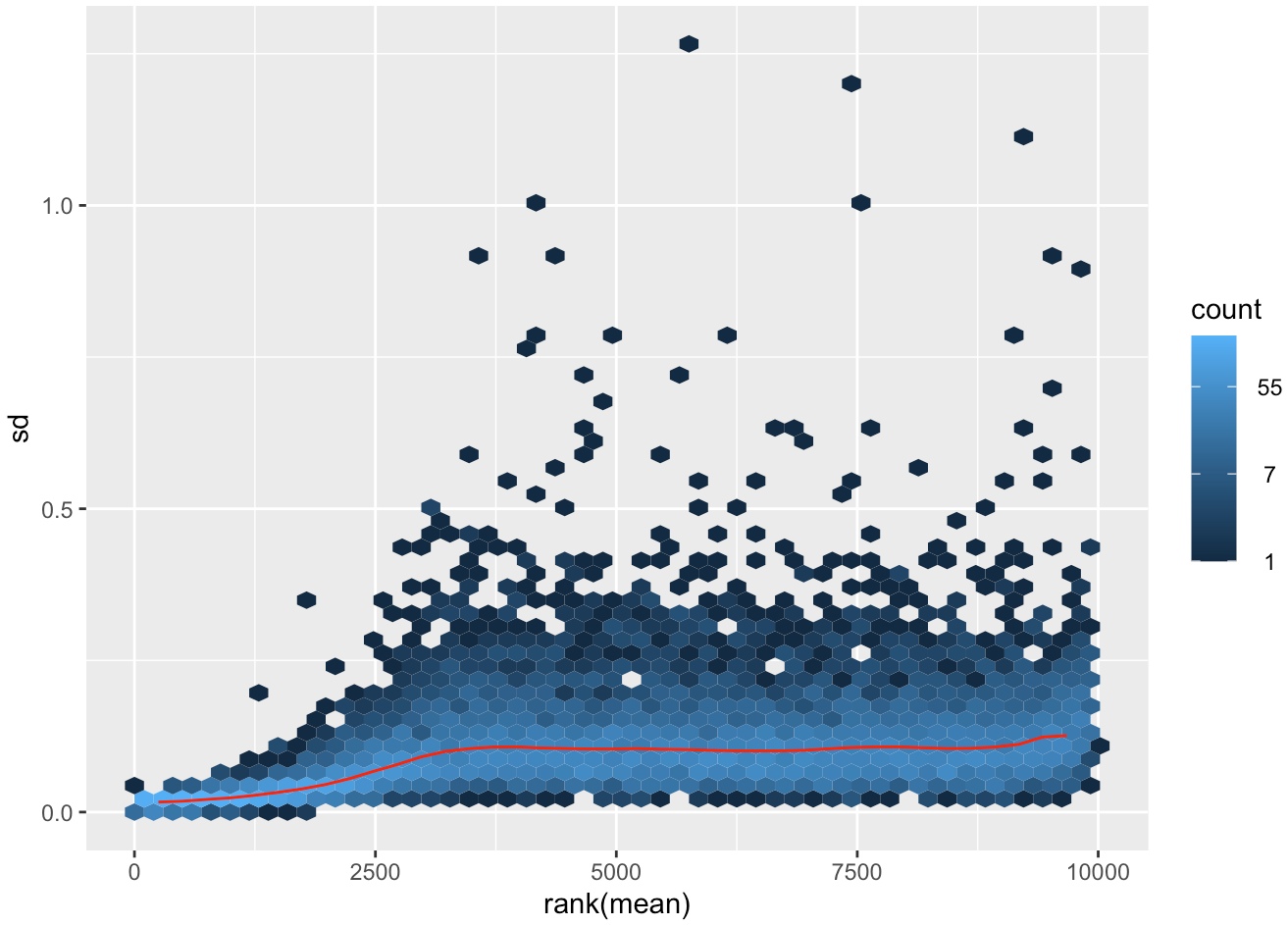
```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## ℹ Please use tidy evaluation idioms with `aes()`.
## ℹ See also `vignette("ggplot2-in-packages")` for more information.
## ℹ The deprecated feature was likely used in the vsn package.
##   Please report the issue to the authors.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




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




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



Observation: In VST and rlog, the variance is stable across the range of mean expression values, which is ideal for downstream visualization and clustering methods.

Quality Control by Sample Clustering and Visualization

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 dataset, preceding or in parallel to the differential expression testing.

Quality as Fitness for Purpose

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 abnormality that renders the data points obtained from these particular samples detrimental to our purpose.

Hierarchical Clustering with Heatmaps

Hierarchical clustering groups similar data points into clusters and is commonly used to visualize gene expression patterns in heatmaps. The clustering creates a dendrogram where: - **Long branches** indicate dissimilar samples/genes - **Short branches** indicate similar samples/genes

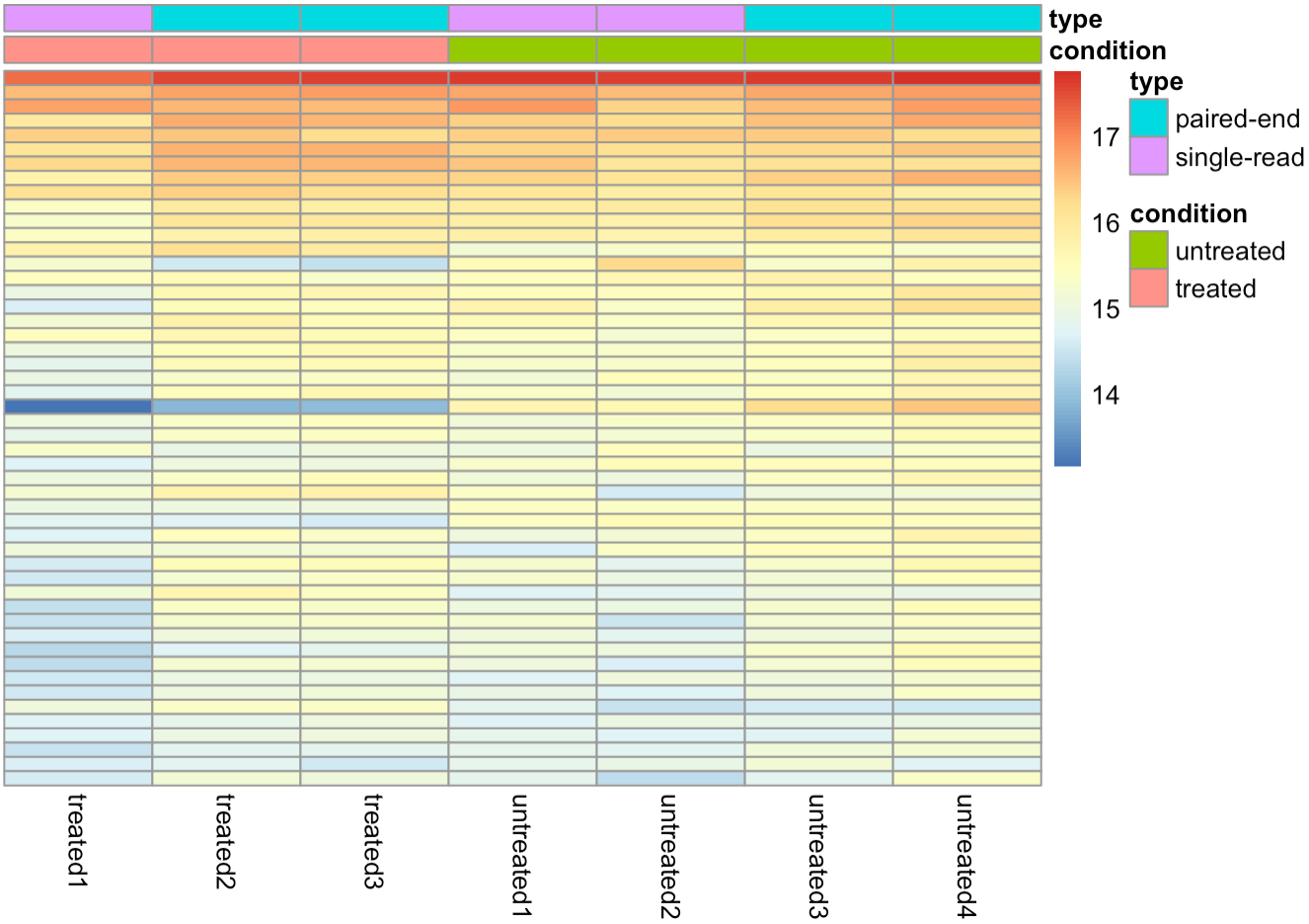
We will generate several heatmaps to assess data quality. First, let's select the top 50 most highly expressed

genes by mean expression across all samples:

```
select <- order(rowMeans(counts(dds,normalized=FALSE)),
                decreasing=TRUE)[1:50]
df <- as.data.frame(colData(dds)[,c("condition","type")])
```

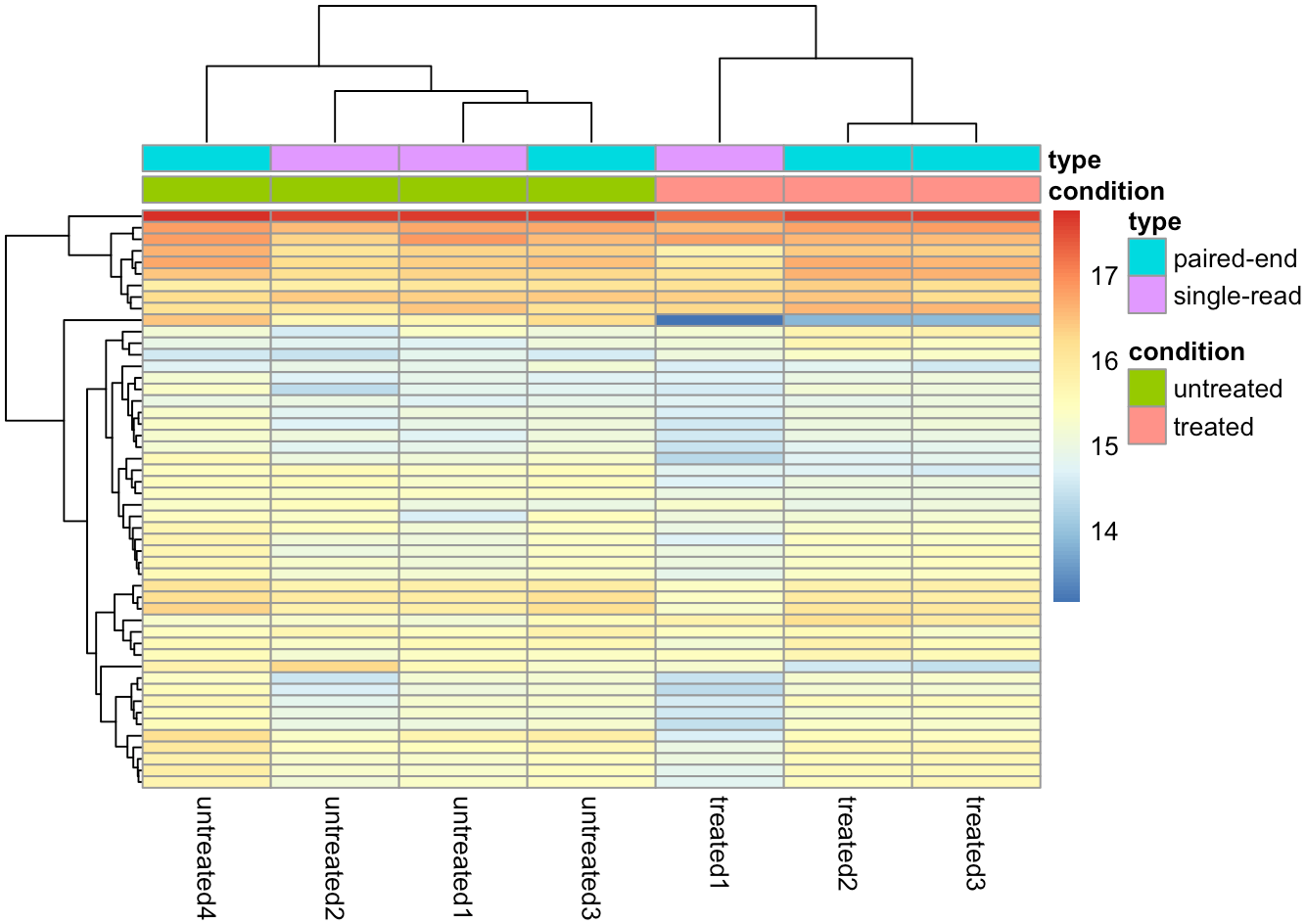
Heatmap 1: Without clustering (original order)

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



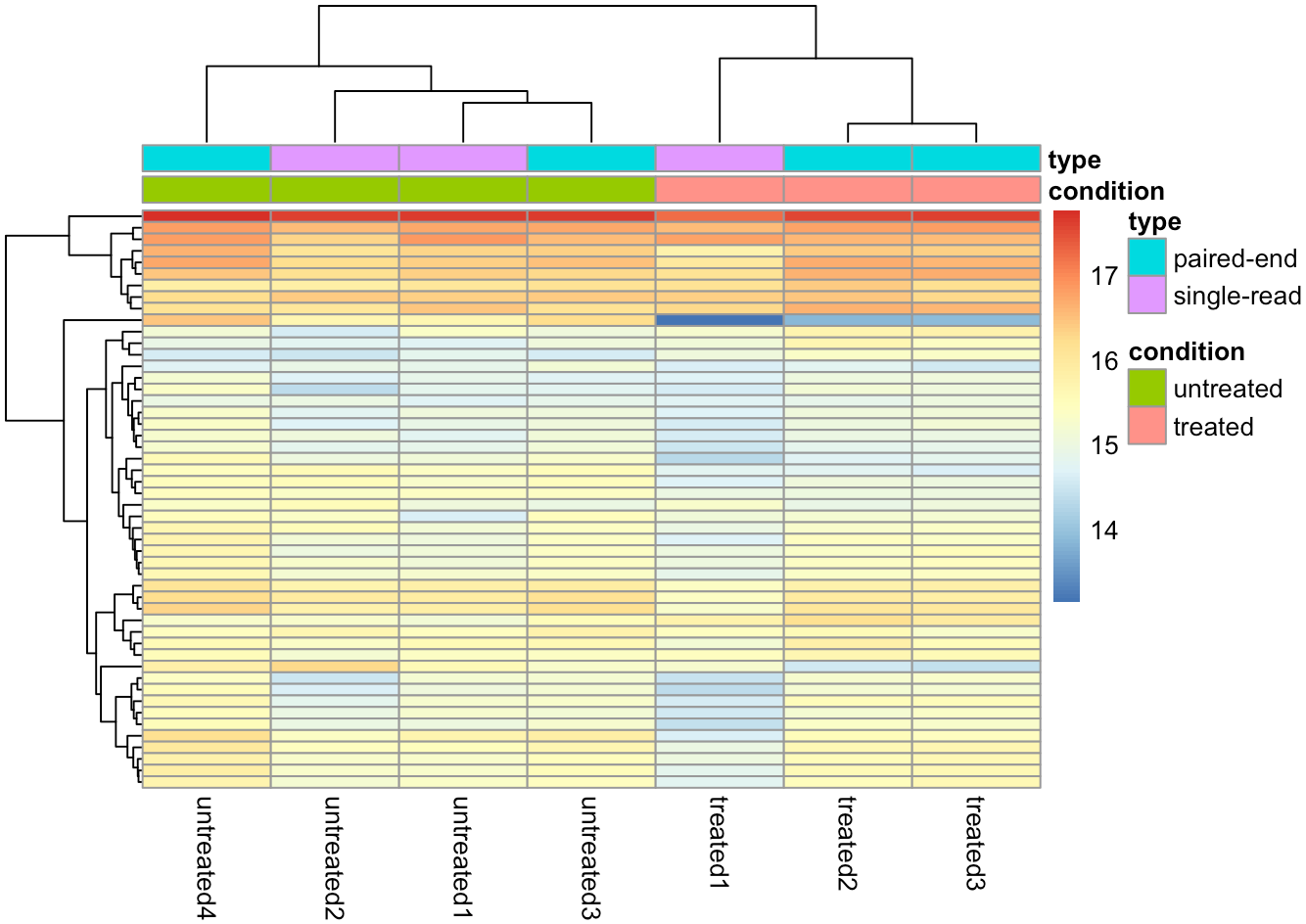
Heatmap 2: With hierarchical clustering

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



Heatmap 3: Comparing with simple log2 transformation (no variance stabilization)

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

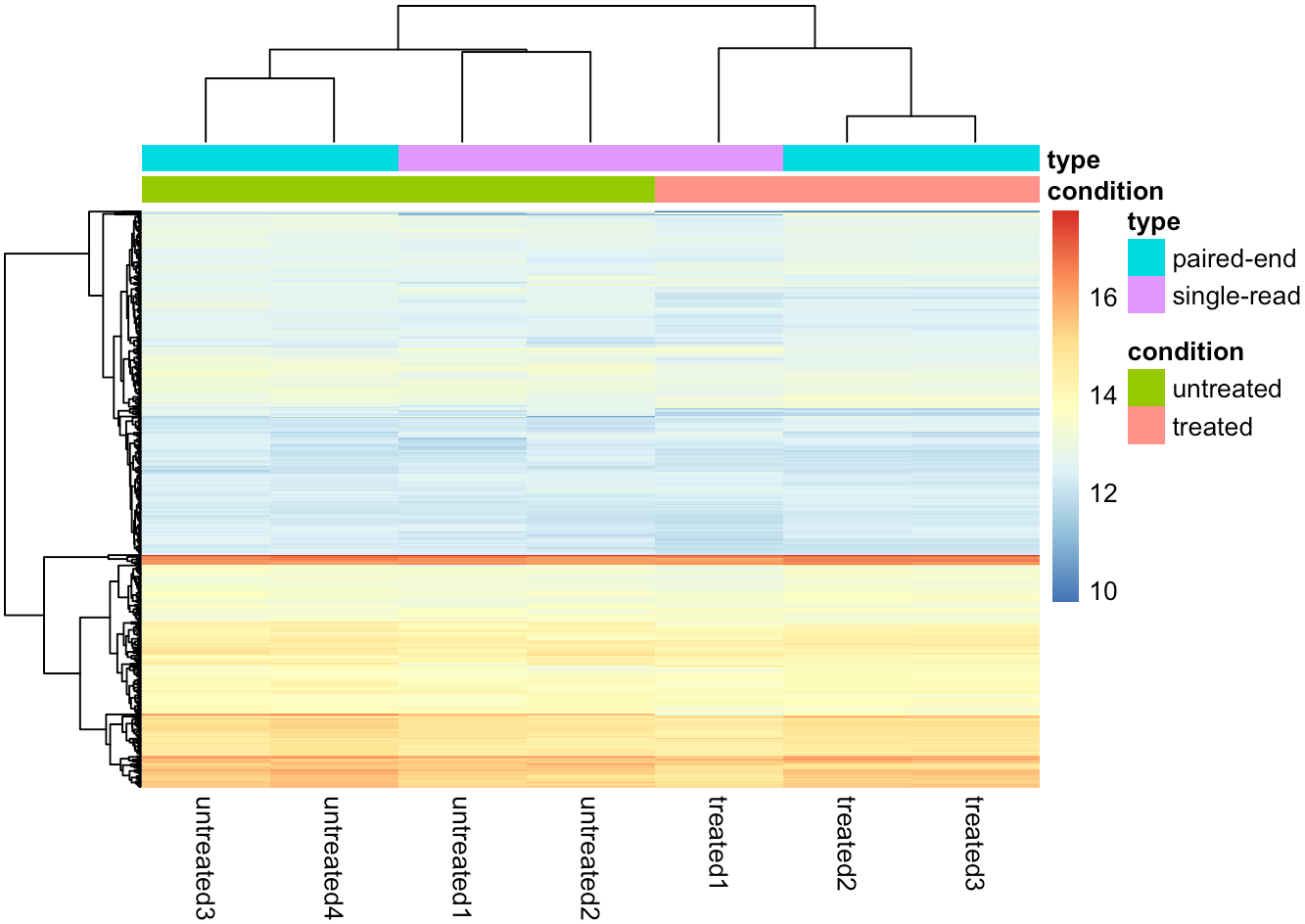


In this case, the data is not dramatically different between transformations, but with real datasets with more noise, the difference can be substantial.

Larger Heatmap

For fun, let's generate a heatmap with the top 500 genes:

```
select_500 <- order(rowMeans(counts(dds,normalized=FALSE)),
                    decreasing=TRUE)[1:500]
pheatmap(assay(vsd)[select_500,], cluster_rows=TRUE, show_rownames=FALSE,
         cluster_cols=TRUE, annotation_col=df)
```



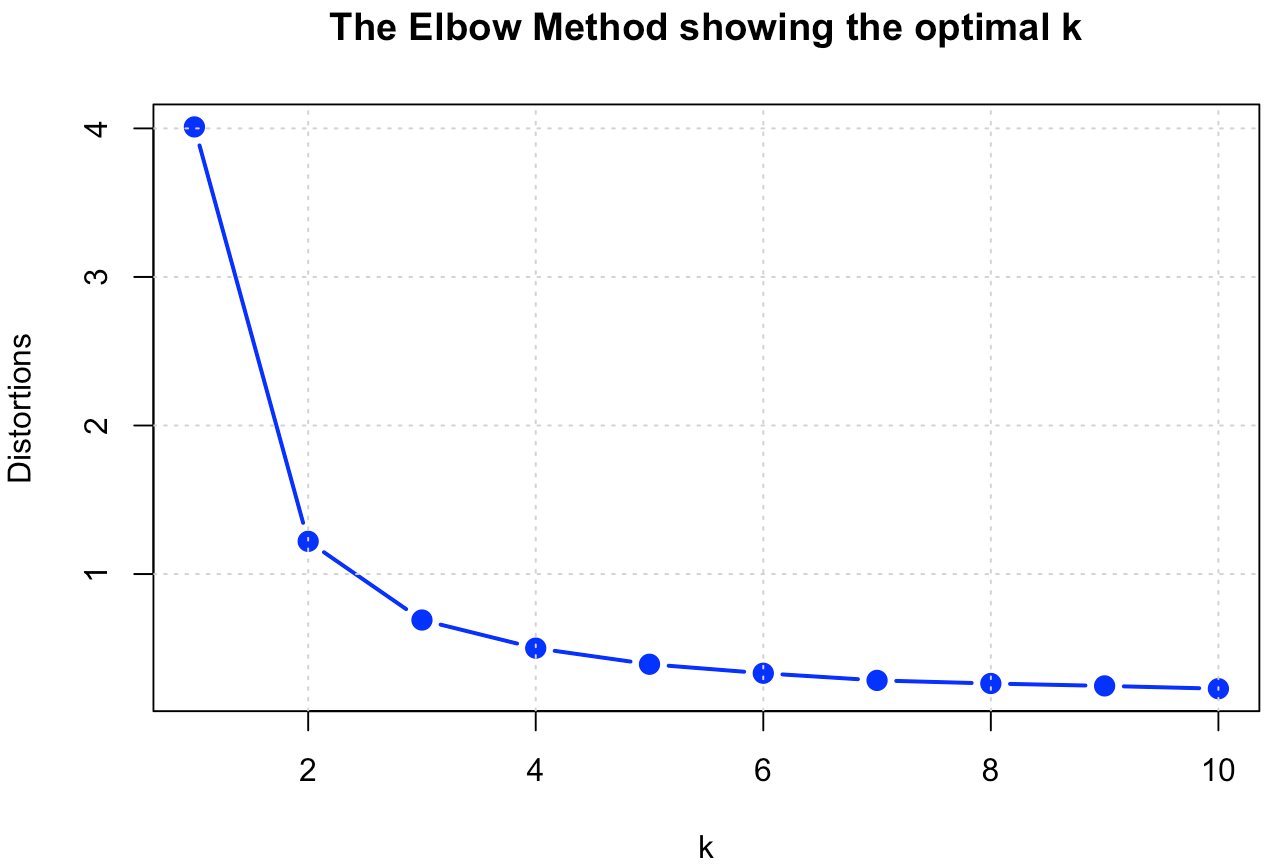
K-means Clustering in Heatmaps

We can also use **K-means clustering** to group genes into a predefined number of clusters (K). K-means groups similar genes together before generating the heatmap, which can help identify patterns.

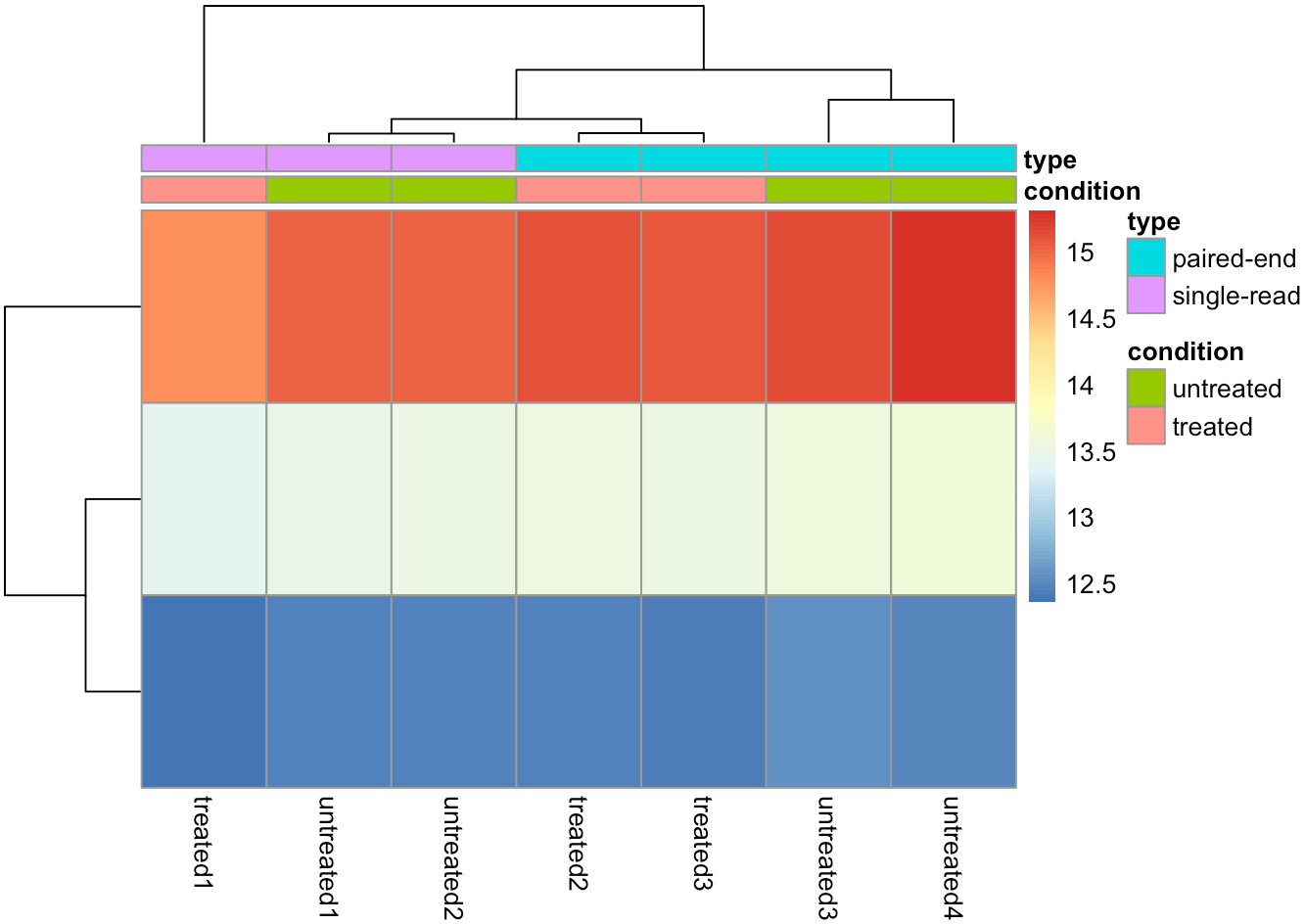
Reminder: K-means clustering is an unsupervised ML method that: 1. Creates K random centroids 2. Assigns each data point to the closest centroid 3. Recalculates centroids 4. Repeats until convergence

To set the number of K's to use, let us first generate a Elbow plot

```
select_500 <- order(rowMeans(counts(dds,normalized=FALSE)),
                decreasing=TRUE)[1:500]

# Extract the expression matrix: transpose so genes become rows for kmeans
gene_matrix_t <- t(assay(vsd)[select_500,])

# Test K values from 1 to 10
k_values <- 1:10
distortions <- numeric(length(k_values))

# Calculate distortions for clustering GENES
set.seed(123)
for (i in k_values) {
  # Cluster the genes (columns of original, rows after transpose)
  kmeans_result <- kmeans(t(gene_matrix_t), centers = i, nstart = 25)
  distortions[i] <- kmeans_result$tot.withinss
}

# Scale distortions (divide by 1000)
distortions_scaled <- distortions / 1000

# Create the elbow plot
plot(k_values, distortions_scaled, type = "b", pch = 19,
     xlab = "k",
     ylab = "Distortions",
     main = "The Elbow Method showing the optimal k",
     col = "blue", lwd = 2, cex = 1.2)
grid()
```




```
Set_K = 3
pheatmap(assay(vsd)[select_500,], cluster_rows=TRUE, show_rownames=FALSE,
         cluster_cols=TRUE, annotation_col=df, kmeans_k = Set_K)
```



By setting K-means, the rows (genes) are aggregated into N clusters before generating the heatmap, making it easier to see groups of genes with similar expression patterns.

Sample Distance Matrix

Another way to assess sample similarity is to calculate Euclidean distances between samples and visualize them in a distance matrix heatmap.

Reminder: Euclidean distance is also used in K-nearest neighbors (KNN) classification to determine similarity between data points.

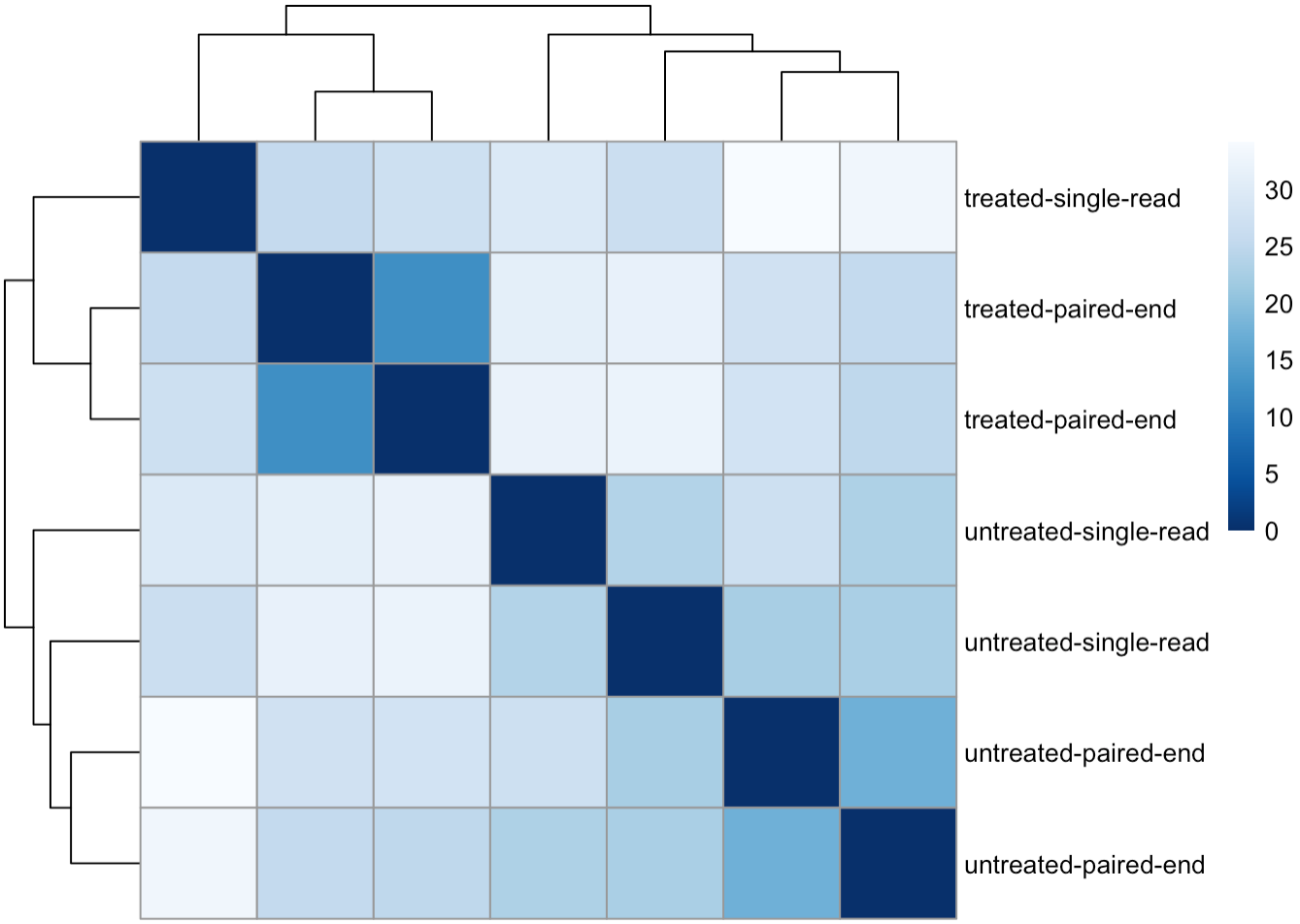
Calculating the distances:

```
sampleDists <- dist(t(assay(vsd)), method = "euclidean")
head(sampleDists)
```

```
## [1] 25.50386 26.95680 29.40565 26.55427 34.19351 33.10003
```

Plotting the distance matrix:

```
sampleDistMatrix <- as.matrix(sampleDists)
rownames(sampleDistMatrix) <- paste(vsd$condition, vsd$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)
```



Darker blue indicates smaller distances (more similar samples), while lighter colors indicate larger distances (more dissimilar samples).

Principal Component Analysis (PCA)

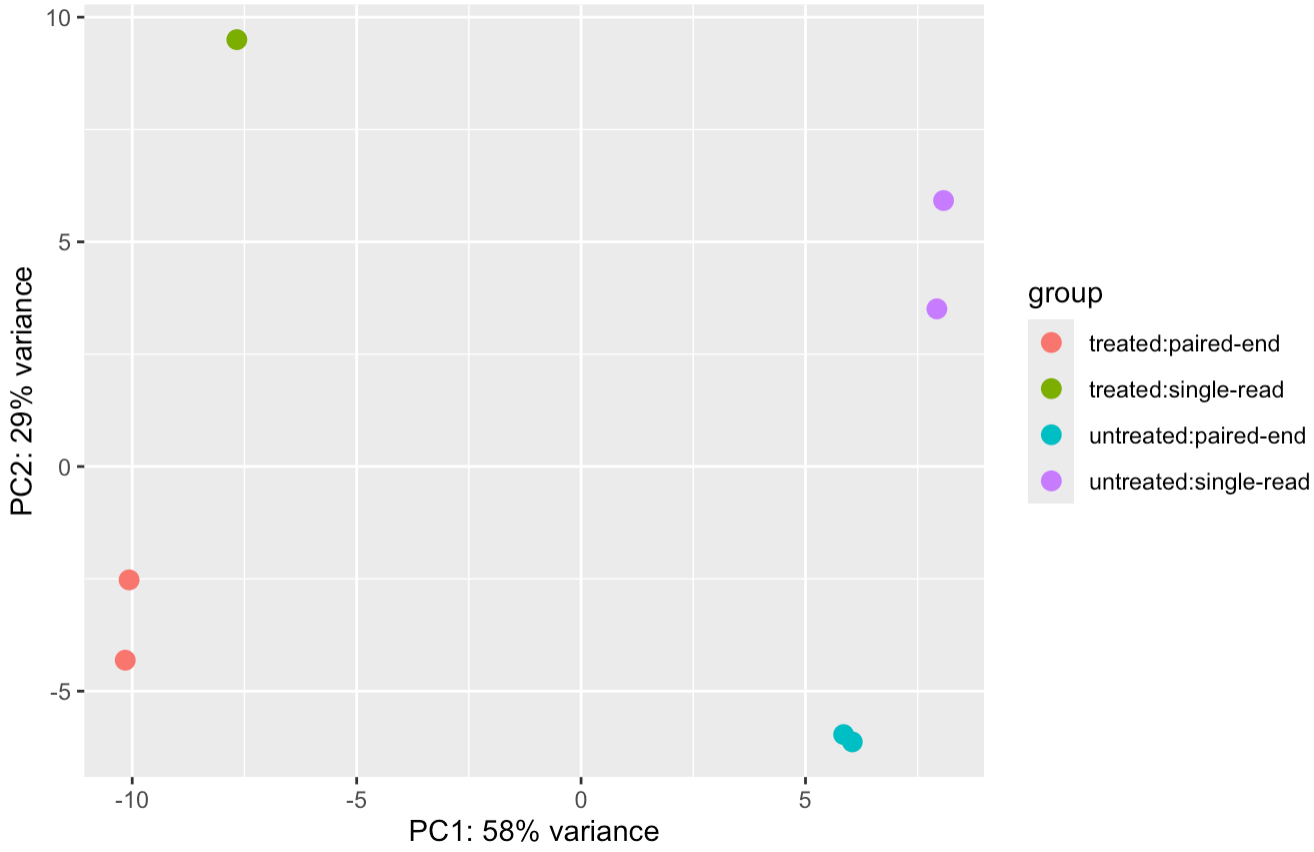
We leave the best for last - **PCA**, loved by most bioinformaticians!

PCA Refresher: - PCA is a dimensionality reduction technique that summarizes high-dimensional data (thousands of genes) into a smaller number of dimensions (typically 2 for visualization) - **PC1** (Principal Component 1) explains the most variance in the data - **PC2** explains the second most variance, and is perpendicular to PC1 - Each PC is a linear combination of all genes, with different genes having different “loadings” (weights/contributions) - The loadings are eigenvectors of the covariance matrix - PCA is commonly used in transcriptomics, proteomics, lipidomics, metabolomics, and other -omics fields for quality control

Key questions PCA helps answer: - Do replicates cluster together? - Do experimental conditions separate? - Is there a batch effect?

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

```
## using ntop=500 top features by variance
```

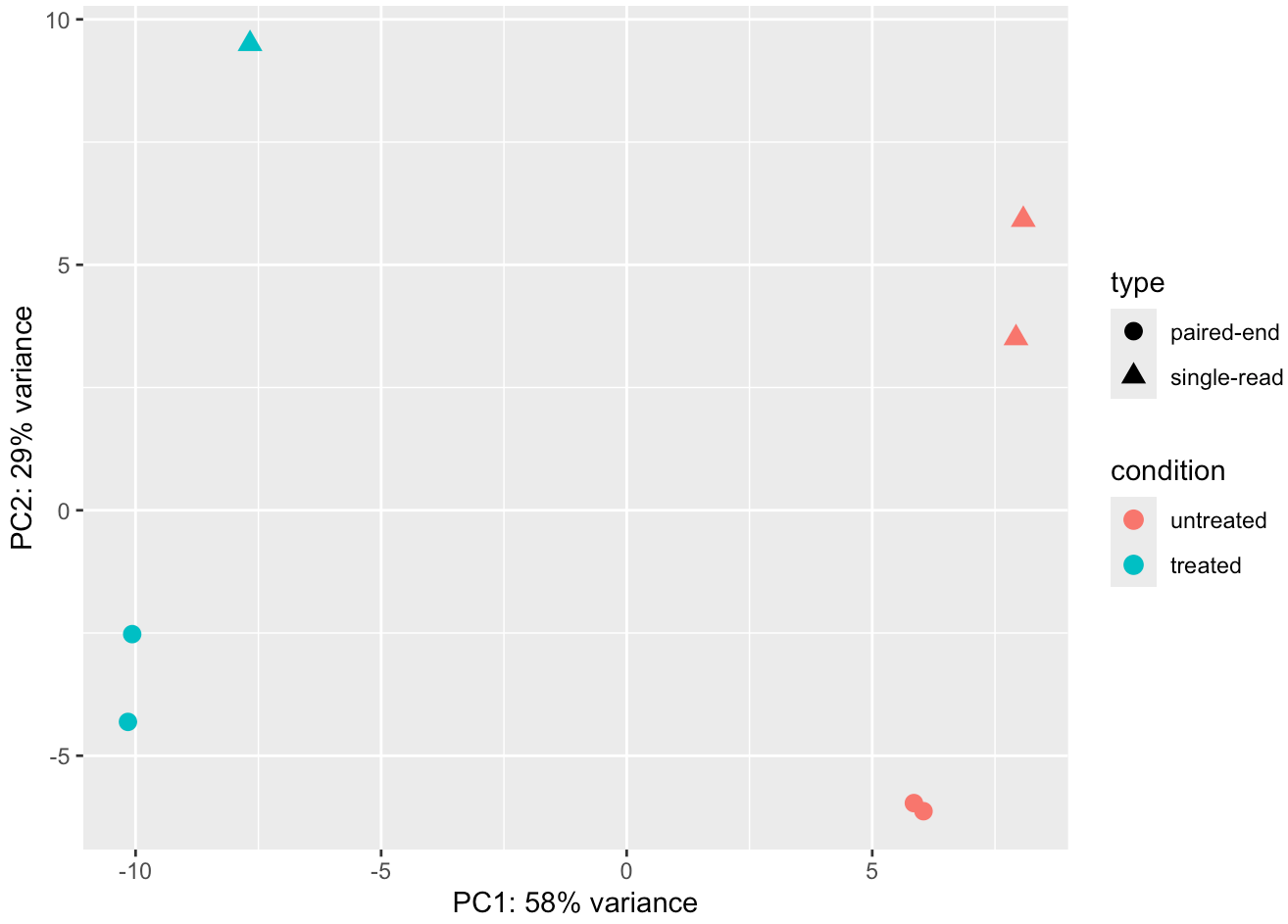


The PCA plot can be further customized with ggplot2:

```
pcaData <- plotPCA(vsd, intgroup=c("condition", "type"), returnData=TRUE)
```

```
## using ntop=500 top features by variance
```

```
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")) +
  coord_fixed()
```



Interpretation:

The data looks good! The majority of the variance is captured in PC1, where we see that samples cluster based on the experimental condition (untreated vs. treated). PC2 captures the technical difference between paired-end and single-read sequencing.

This is exactly what we want to see: - **Biological variation (treatment effect)** is the dominant source of variance (PC1) - **Technical variation (sequencing type)** is a secondary source (PC2) - Replicates from the same condition cluster together

Differential Gene Expression Analysis

Now we move to the main analysis: identifying differentially expressed genes using DESeq2.

The DESeq2 Statistical Model

DESeq2 uses a **generalized linear model (GLM)** based on the Negative Binomial distribution (also known as Gamma-Poisson distribution). This model is well-suited for count data and accounts for: - Mean-variance relationship in count data - Differences in sequencing depth between samples - Biological variability

For more details, see the DESeq2 publication (Love, Huber, and Anders 2014).

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

```
## estimating size factors
```

```
## estimating dispersions
```

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

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

```
## final dispersion estimates
```

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

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

```
## log2 fold change (MLE): condition treated vs untreated
## Wald test p-value: condition treated vs untreated
## DataFrame with 9921 rows and 6 columns
##               baseMean log2FoldChange     lfcSE       stat    pvalue      padj
##              <numeric>      <numeric> <numeric>  <numeric> <numeric> <numeric>
## FBgn0000008   95.14429     0.00227644  0.223729   0.010175 0.9918817  0.997211
## FBgn0000014    1.05652    -0.49512039  2.143186  -0.231021 0.8172987        NA
## FBgn0000017 4352.55357    -0.23991894  0.126337  -1.899041 0.0575591  0.288002
## FBgn0000018  418.61048    -0.10467391  0.148489  -0.704927 0.4808558  0.826834
## FBgn0000024    6.40620     0.21084779  0.689588   0.305759 0.7597879  0.943501
## ...                ...            ...       ...        ...       ...       ...
## FBgn0261570 3208.38861      0.2955329  0.127350  2.3206264  0.020307  0.144240
## FBgn0261572    6.19719     -0.9588230  0.775315 -1.2366888  0.216203  0.607848
## FBgn0261573 2240.97951      0.0127194  0.113300  0.1122634  0.910615  0.982657
## FBgn0261574 4857.68037      0.0153924  0.192567  0.0799327  0.936291  0.988179
## FBgn0261575   10.68252      0.1635705  0.930911  0.1757102  0.860522  0.967928
```

Log Fold Change Shrinkage

Raw log fold change (LFC) estimates can have high variance, especially for genes with low counts. **Shrinkage** helps by pulling unreliable LFC estimates toward zero, improving visualization and gene ranking.

Apeglm Method: - Based on a generalized linear model - Provides Bayesian shrinkage estimators - Uses approximation of the posterior for individual coefficients - Has lower bias than previous estimators while still reducing variance

We specify the apeglm method for effect size shrinkage (Zhu, Ibrahim, and Love 2018):

```
resLFC <- lfcShrink(dds, coef="condition_treated_vs_untreated", type="apeglm")
```

```
## using 'apeglm' for LFC shrinkage. If used in published research, please cite:
##     Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distributions for
##     sequence count data: removing the noise and preserving large differences.
##     Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895
```

```
resLFC
```

```
## log2 fold change (MAP): condition treated vs untreated
## Wald test p-value: condition treated vs untreated
## DataFrame with 9921 rows and 5 columns
##               baseMean log2FoldChange     lfcSE    pvalue      padj
##              <numeric>      <numeric> <numeric> <numeric> <numeric>
## FBgn0000008   95.14429     0.00117785  0.151896 0.9918817  0.997211
## FBgn0000014    1.05652    -0.00472053  0.205467 0.8172987        NA
## FBgn0000017 4352.55357    -0.19018294  0.120382 0.0575591  0.288002
## FBgn0000018  418.61048    -0.07001006  0.123898 0.4808558  0.826834
## FBgn0000024    6.40620     0.01752520  0.198633 0.7597879  0.943501
## ...                ...            ...       ...       ...       ...
## FBgn0261570 3208.38861     0.23521340 0.1240667  0.020307  0.144240
## FBgn0261572    6.19719    -0.06575858 0.2141337  0.216203  0.607848
## FBgn0261573 2240.97951     0.00975870 0.0993753  0.910615  0.982657
## FBgn0261574 4857.68037     0.01017595 0.1408931  0.936291  0.988179
## FBgn0261575   10.68252     0.00809101 0.2014704  0.860522  0.967928
```

Extracting and Summarizing Results

Let's extract the most significant differentially expressed genes:

```
resOrdered <- res[order(res$padj),]
head(resOrdered, 10)
```

```
## log2 fold change (MLE): condition treated vs untreated
## Wald test p-value: condition treated vs untreated
## DataFrame with 10 rows and 6 columns
##              baseMean log2FoldChange     lfcSE      stat       pvalue
##             <numeric>      <numeric> <numeric> <numeric>    <numeric>
## FBgn0039155   730.568       -4.61874 0.1691240  -27.3098 3.24447e-164
## FBgn0025111  1501.448        2.89995 0.1273576   22.7701 9.07165e-115
## FBgn0029167  3706.024       -2.19691 0.0979154  -22.4368 1.72030e-111
## FBgn0003360  4342.832       -3.17954 0.1435677  -22.1466 1.12417e-108
## FBgn0035085   638.219       -2.56024 0.1378126  -18.5777  4.86845e-77
## FBgn0039827   261.911       -4.16243 0.2325942  -17.8957  1.27485e-71
## FBgn0034736   225.871       -3.51132 0.2147628  -16.3498  4.36736e-60
## FBgn0029896   489.877       -2.44494 0.1522149  -16.0625  4.67705e-58
## FBgn0000071   342.246        2.67973 0.1824175   14.6901  7.46383e-49
## FBgn0051092   153.069        2.32791 0.1772255   13.1353  2.06706e-39
##                    padj
##               <numeric>
## FBgn0039155 2.71919e-160
## FBgn0025111 3.80147e-111
## FBgn0029167 4.80596e-108
## FBgn0003360 2.35542e-105
## FBgn0035085  8.16049e-74
## FBgn0039827  1.78075e-68
## FBgn0034736  5.22898e-57
## FBgn0029896  4.89979e-55
## FBgn0000071  6.95048e-46
## FBgn0051092  1.73240e-36
```

Summary of results at adjusted p-value < 0.1:

```
summary(res)
```

```
##
## out of 9921 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 518, 5.2%
## LFC < 0 (down)     : 536, 5.4%
## outliers [1]       : 1, 0.01%
## low counts [2]     : 1539, 16%
## (mean count < 6)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
print("Number of DEGs:")
```

```
## [1] "Number of DEGs:"
```

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

```
## [1] 1054
```

The parameters can also be adjusted to use adjusted p-value of 0.05:

```
res05 <- results(dds, alpha=0.05)
summary(res05)
```

```
##
## out of 9921 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)       : 407, 4.1%
## LFC < 0 (down)     : 431, 4.3%
## outliers [1]       : 1, 0.01%
## low counts [2]     : 1347, 14%
## (mean count < 5)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
print("Number of DEGs:")
```

```
## [1] "Number of DEGs:"
```

```
sum(res05$padj < 0.05, na.rm=TRUE)
```

```
## [1] 838
```

Comparing to the LFC-shrunken results:

```
summary(resLFC)
```

```
##
## out of 9921 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 518, 5.2%
## LFC < 0 (down)     : 536, 5.4%
## outliers [1]       : 1, 0.01%
## low counts [2]     : 1539, 16%
## (mean count < 6)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
print("Number of DEGs:")
```

```
## [1] "Number of DEGs:"
```

```
sum(resLFC$padj < 0.05, na.rm=TRUE)
```

```
## [1] 841
```

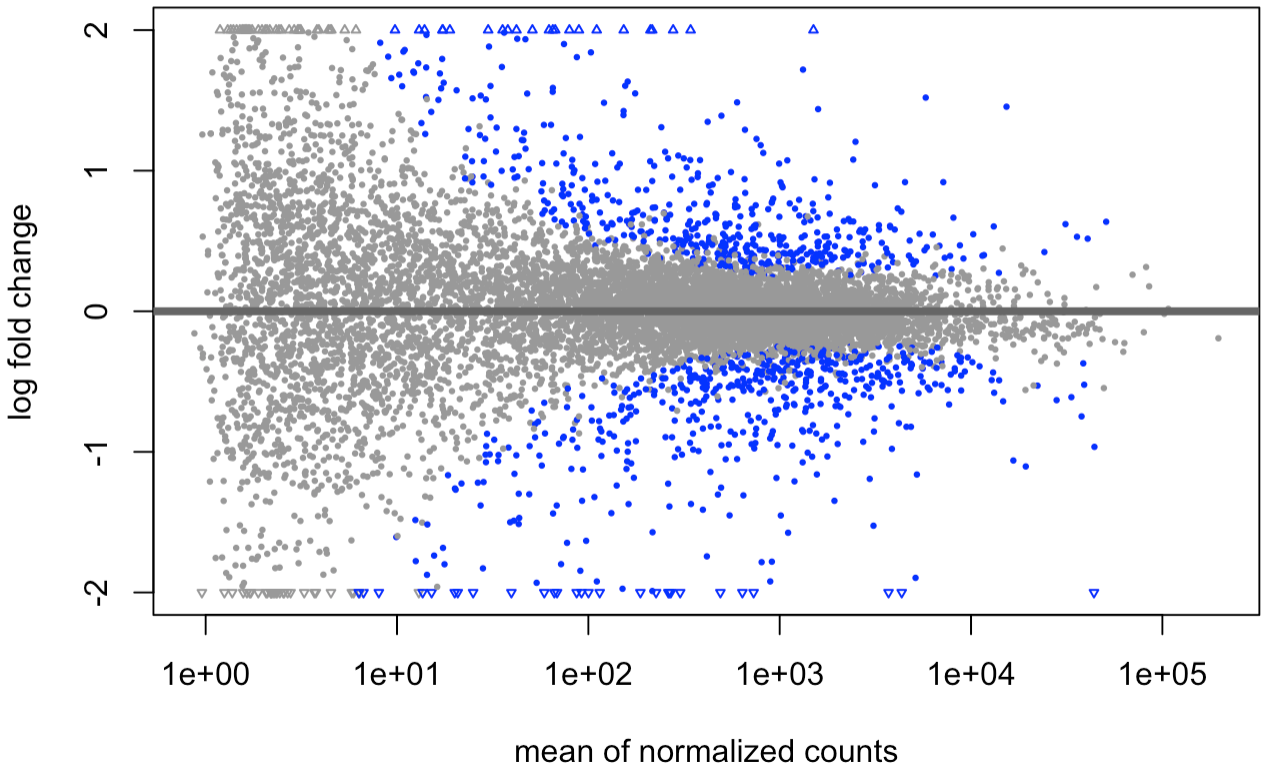
At adjusted p-value 0.1, there are no differences in the number of DEGs between shrunken and non-shrunken results. However, as we lower the cutoff, differences emerge.

Visualizing the Effect of Shrinkage

The non-shrunken and shrunken data can be compared visually using MA plots (M = log ratio, A = mean average):

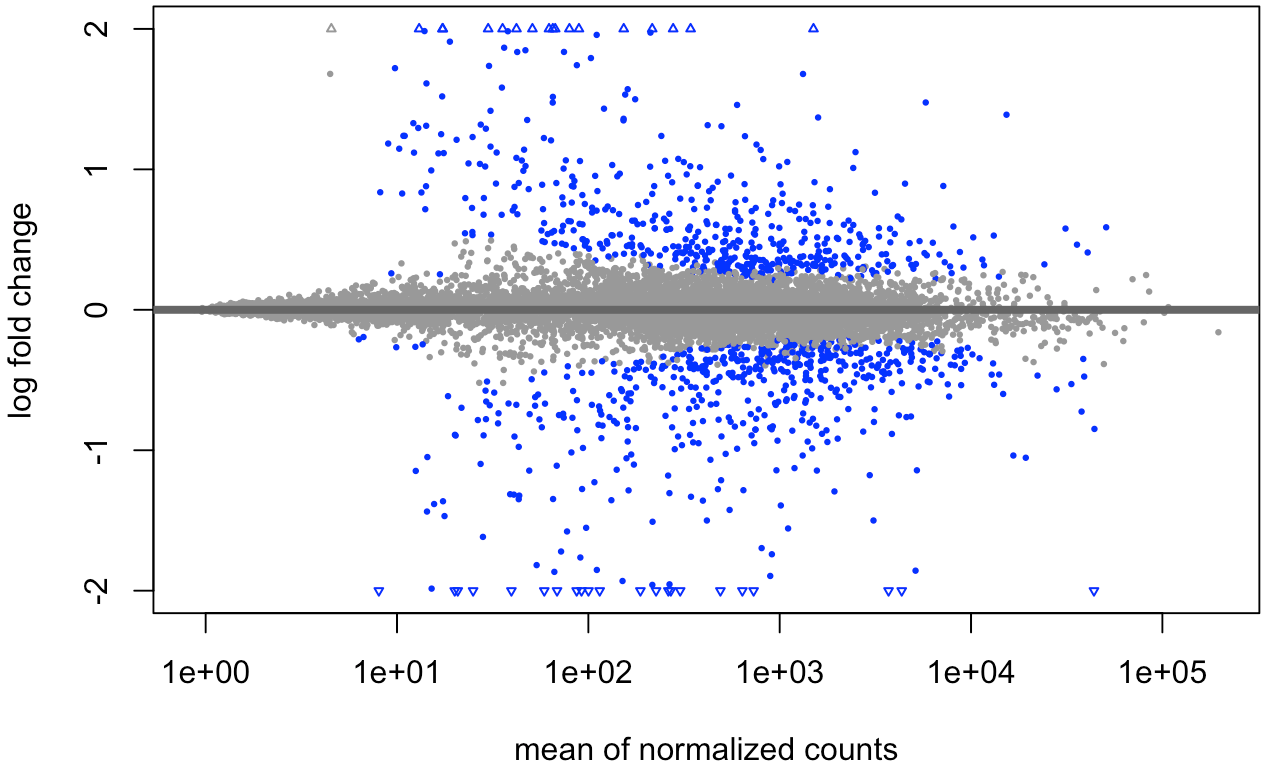
Without shrinkage:

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



With shrinkage:

```
plotMA(resLFC, ylim=c(-2,2))
```



Observation: In the shrunken plot, you can see that LFC estimates for genes with low mean expression (left side of the plot) are pulled toward zero, while high-confidence estimates (high expression genes) remain largely unchanged.

Understanding Shrinkage

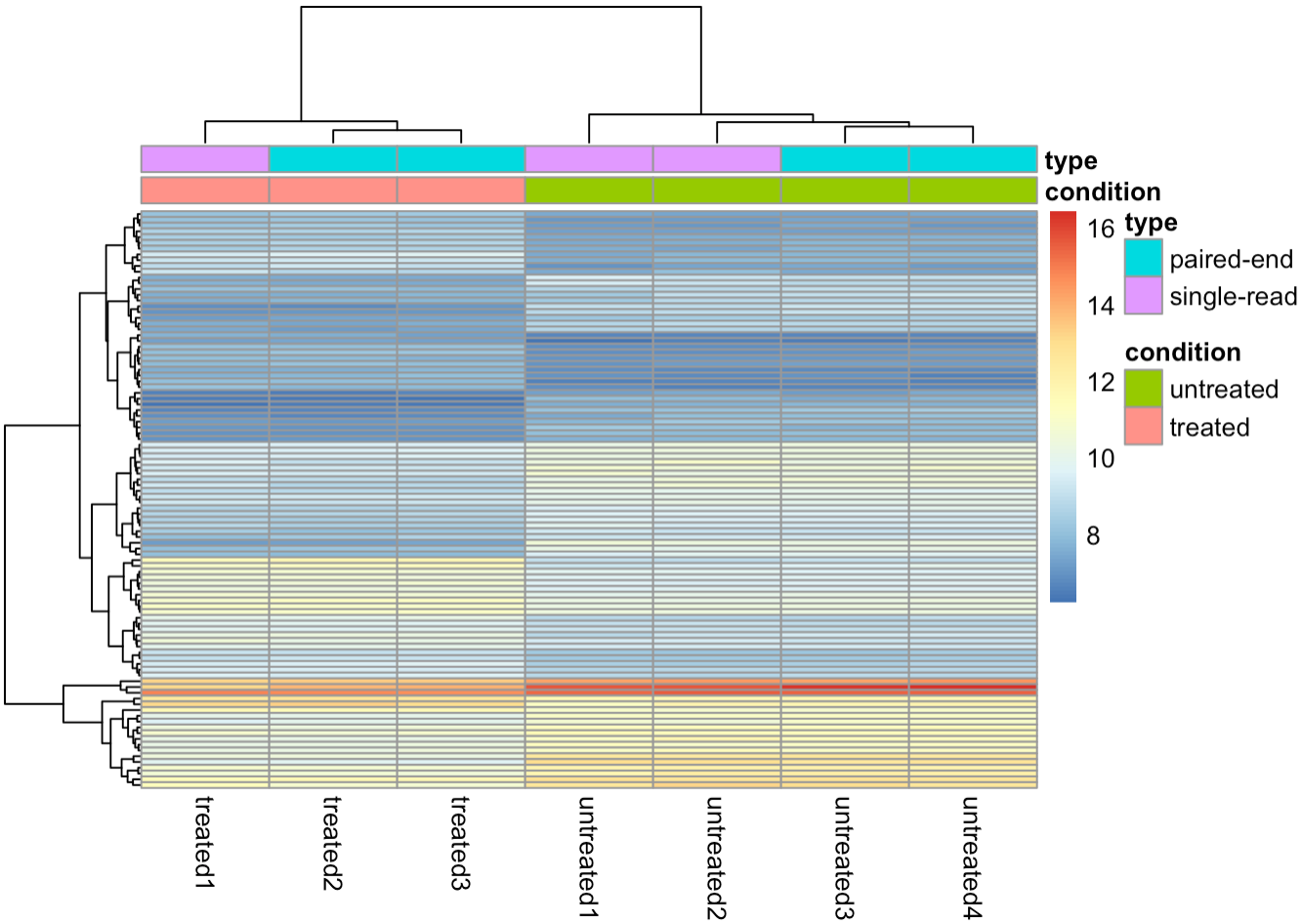
From the APEGLM paper:

> "When the read counts are low or highly variable, the maximum likelihood estimates for the LFCs has high variance, leading to large estimates not representative of true differences, and poor ranking of genes by effect size. One approach is to introduce filtering thresholds and pseudocounts to exclude or moderate estimated LFCs. Filtering may result in a loss of genes from the analysis with true differences in expression, while pseudocounts provide a limited solution that must be adapted per dataset. […] The proposed method, Approximate Posterior Estimation for generalized linear model (apeglm), has lower bias than previously proposed shrinkage estimators, while still reducing variance for those genes with little information for statistical inference."

Heatmap of Top Differentially Expressed Genes

Let's visualize the top 100 differentially expressed genes with hierarchical clustering:

```
#resLFC <- resLFC[!is.na(resLFC$padj),]
#DE.genes = rownames(resLFC[resLFC$padj < .05 & abs(resLFC$log2FoldChange) > 1,])

resOrdered.LFC <- resLFC[order(resLFC$padj, decreasing = FALSE),]
resOrdered.LFC.genes = rownames(resOrdered.LFC)[1:100]

pheatmap(assay(vsd)[resOrdered.LFC.genes,], cluster_rows=TRUE, show_rownames=FALSE,
         cluster_cols=TRUE, annotation_col=df)
```



This heatmap clearly shows: - Hierarchical clustering reveals groups of genes with similar expression patterns - Samples cluster by experimental condition - Some genes are upregulated in treated samples (red/yellow), while others are downregulated (blue)

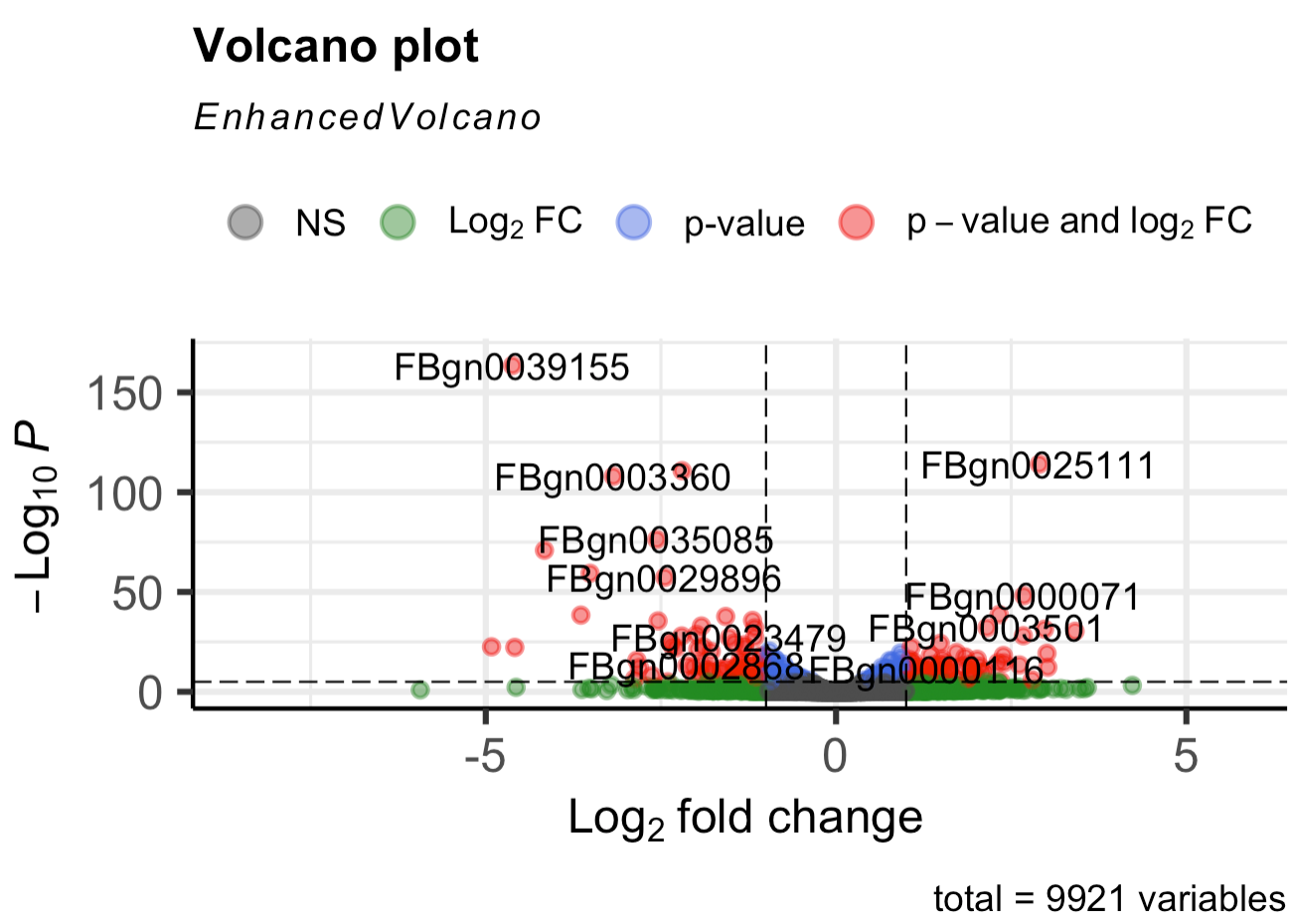
Volcano Plot

Finally, let's visualize all DEGs with a volcano plot, which shows both statistical significance (y-axis) and effect size (x-axis):

```
EnhancedVolcano(res,
    lab = rownames(res),
    x = 'log2FoldChange',
    y = 'pvalue')
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## ℹ The deprecated feature was likely used in the EnhancedVolcano package.
##   Please report the issue to the authors.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0
.
## ℹ Please use the `linewidth` argument instead.
## ℹ The deprecated feature was likely used in the EnhancedVolcano package.
##   Please report the issue to the authors.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



The volcano plot makes it easy to identify the most significant and biologically relevant genes: - Points in the upper corners are both statistically significant and have large fold changes - Points in the middle top are significant but have small effect sizes - Points at the bottom are not significant

Connecting to Machine Learning Concepts

Throughout this tutorial, we've applied several ML concepts:

1. **Unsupervised Learning:**
    - K-means clustering for grouping genes
    - Hierarchical clustering for creating dendrograms
    - PCA for dimensionality reduction and visualization
2. **Supervised Learning:**
    - Generalized Linear Models (GLMs) in the DESeq2 statistical test
    - Similar to linear/logistic regression concepts we discussed
3. **Validation:**
    - Quality control with PCA and clustering
    - Multiple testing correction (adjusted p-values)
    - Shrinkage to reduce overfitting of LFC estimates

ML Applications in RNA-seq

The ML concepts we've covered today have broader applications in transcriptomics:

- **Linear/Logistic Regression:** Studying gene expression relationships to clinical variables
- **KNN:** Cell type classification in single-cell RNA-seq (used in Leiden clustering)
- **Random Forest:** Disease prediction using gene expression and clinical variables
- **Consensus Clustering:** Subtyping diseases based on gene expression patterns

Summary and Next Steps

Congratulations! You've completed a full DESeq2 analysis workflow, applying:

- Data preprocessing and quality control
- Variance-stabilizing transformation
- Unsupervised ML (PCA, hierarchical clustering, K-means)
- Statistical testing with GLMs
- Visualization and interpretation

For Further Learning

- **DESeq2 Vignette:** http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html (http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html)
- **StatQuest YouTube:** Excellent tutorials on ML and statistics, including PCA: https://www.youtube.com/watch?v=FgakZw6K1QQ (https://www.youtube.com/watch?v=FgakZw6K1QQ)
- **The GitHub:** Additional bioinformatics tutorials and projects: https://github.com/GustawEriksson/Machine_Learning_introduction_2023 (https://github.com/GustawEriksson/Machine_Learning_introduction_2023)

Questions?

Feel free to explore the code, modify parameters, and experiment with different visualizations. Machine learning is best learned by doing!

Workshop created by: Gustaw Eriksson (gustaw.eriksson@ki.se (mailto:gustaw.eriksson@ki.se))
Date: February 9, 2026
Affiliation: Reproductive Endocrinology and Metabolism Group, Karolinska Institutet