Class 12: Transcriptomics and Analysis of RNA-Seq Data

Kira Jung

2023-06-10

1. Bioconductor and DESeq2 setup

First installed the core Bioconductor packages. Then, installed the DESeq2 bioconductor package.

After installing, the packages should then be able to load.

```
library(BiocManager)
```

```
## Bioconductor version '3.16' is out-of-date; the current release version '3.17'
##   is available with R version '4.3'; see https://bioconductor.org/install
```

```
library(DESeq2)
```

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

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

```
## 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, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min
```

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

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

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

```
##
## Attaching package: 'IRanges'
```

```
## The following object is masked from 'package:grDevices':
##
##     windows
```

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

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

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

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

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

```
##
## 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
```

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

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

```
##
## Attaching package: 'Biobase'
```

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

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

2. Import countData and colData

Using airway_scaledcounts.csv and airway_metadata.csv from the course website:

```
counts <- read.csv("airway_scaledcounts.csv", row.names = 1)
metadata <-  read.csv("airway_metadata.csv")
```

Now viewing each of the datasets:

```
head(counts)
```

```
##                 SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516
## ENSG00000000003        723        486        904        445       1170
## ENSG00000000005          0          0          0          0          0
## ENSG00000000419        467        523        616        371        582
## ENSG00000000457        347        258        364        237        318
## ENSG00000000460         96         81         73         66        118
## ENSG00000000938          0          0          1          0          2
##                 SRR1039517 SRR1039520 SRR1039521
## ENSG00000000003       1097        806        604
## ENSG00000000005          0          0          0
## ENSG00000000419        781        417        509
## ENSG00000000457        447        330        324
## ENSG00000000460         94        102         74
## ENSG00000000938          0          0          0
```

```
head(metadata)
```

```
##           id     dex celltype     geo_id
## 1 SRR1039508 control   N61311 GSM1275862
## 2 SRR1039509 treated   N61311 GSM1275863
## 3 SRR1039512 control  N052611 GSM1275866
## 4 SRR1039513 treated  N052611 GSM1275867
## 5 SRR1039516 control  N080611 GSM1275870
## 6 SRR1039517 treated  N080611 GSM1275871
```

Q1: How many genes are in this dataset (metadata)?

```
nrow(counts)
```

```
## [1] 38694
```

Answer: There are 38694 genes in the dataset.

Q2: How many 'control' cell lines do we have?

```
n.control <- sum(metadata$dex == "control")
```

Answer: There are 4 control cell lines.

3. Toy Differential Gene Expression

Analysis is for demonstration only

First seeing which samples in the metadata are 'control' and which are drug 'treated'. Then calculating the mean counts per gene across the samples:

```
control <- metadata[metadata$dex == "control", ]
control.counts <- counts[ , control$id]
control.mean <- rowMeans(control.counts)
head(control.mean)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
##          900.75            0.00          520.50          339.75           97.25
## ENSG00000000938
##            0.75
```

Alternatively, this can be done using the dplyr package from tidyverse.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

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

```
## The following object is masked from 'package:matrixStats':
##
##     count
```

```
## The following objects are masked from 'package:GenomicRanges':
##
##     intersect, setdiff, union
```

```
## The following object is masked from 'package:GenomeInfoDb':
##
##     intersect
```

```
## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union
```

```
## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union
```

```
## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
control <- metadata %>% filter(dex == "control")
control.counts <- counts %>% select(control$id)
control.mean <- rowSums(control.counts) / 4
head(control.mean)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
##          900.75            0.00          520.50          339.75           97.25
## ENSG00000000938
##            0.75
```

Q3: How would you make the above code in either approach more robust?

Answer: This can be done by isolating the 'control' column in the `counts` dataset. The mean value for each gene can be determined with `rowMeans()`.

First, isolating 'control':

```
control.inds <- metadata$dex == "control"
control <- metadata[control.inds, ]
control$id
```

```
## [1] "SRR1039508" "SRR1039512" "SRR1039516" "SRR1039520"
```

```
control.counts <- counts[ , control$id]
head(control.counts)
```

```
##                 SRR1039508 SRR1039512 SRR1039516 SRR1039520
## ENSG00000000003        723        904       1170        806
## ENSG00000000005          0          0          0          0
## ENSG00000000419        467        616        582        417
## ENSG00000000457        347        364        318        330
## ENSG00000000460         96         73        118        102
## ENSG00000000938          0          1          2          0
```

And then determining the mean for each gene:

```
control.mean <- rowMeans(control.counts)
head(control.mean)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
##          900.75            0.00          520.50          339.75           97.25
## ENSG00000000938
##            0.75
```

Q4: Follow the same procedure for the treated samples (i.e. calculate the mean per gene across drug treated samples and assign to a labeled vector called treated.mean)

Extract and summarize the drug 'treated' samples:

```
treated <- metadata[metadata$dex == "treated", ]
treated.counts <- counts[, treated$id]
treated.mean <- rowMeans(treated.counts)
head(treated.mean)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
##          658.00            0.00          546.00          316.50           78.75
## ENSG00000000938
##            0.00
```

We will combine our meancount data for bookkeeping purposes.

```
meancounts <- data.frame(control.mean, treated.mean)
```

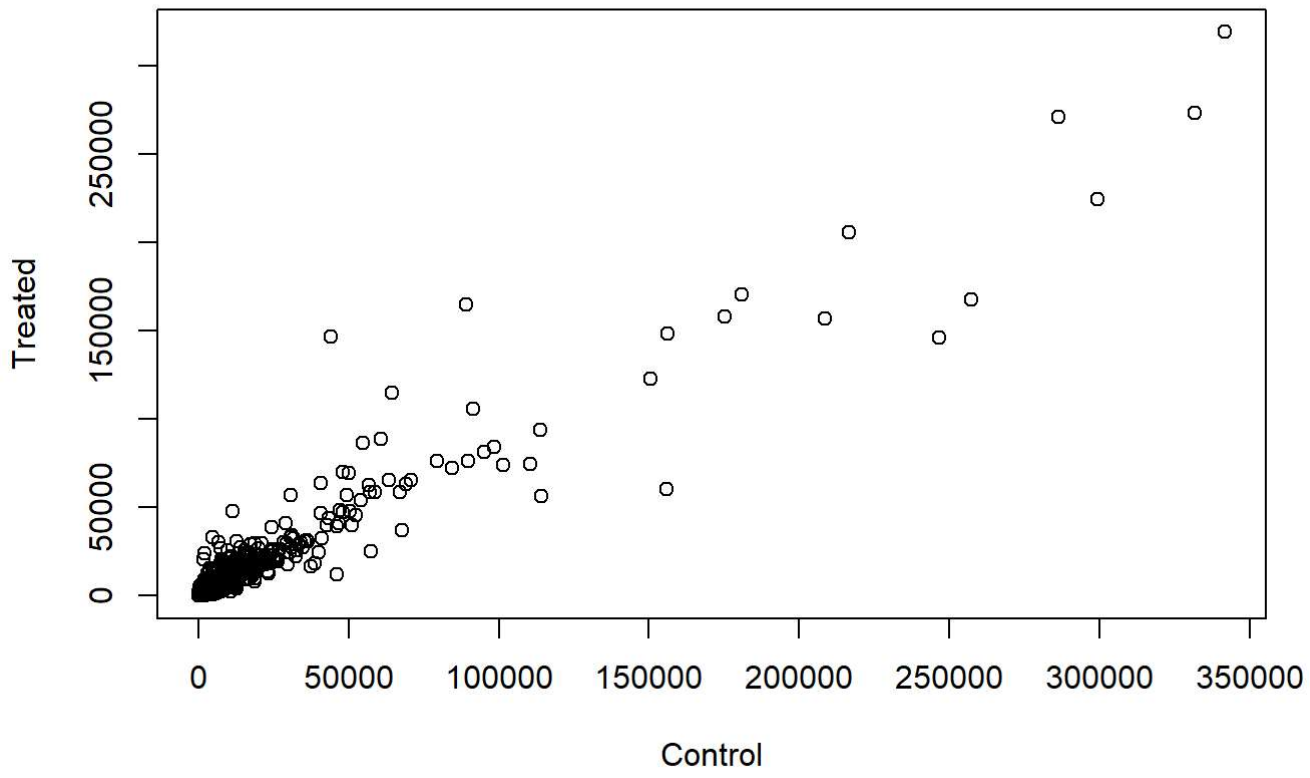
`colSums()` the data to show the sum of the mean counts across all genes for each group.

```
colSums(meancounts)
```

```
## control.mean treated.mean
##     23005324     22196524
```

Q5(a): Create a scatter plot showing the mean of the treated samples against the mean of the control samples.

```
plot(meancounts[ , 1], meancounts[ , 2], xlab = "Control", ylab = "Treated")
```

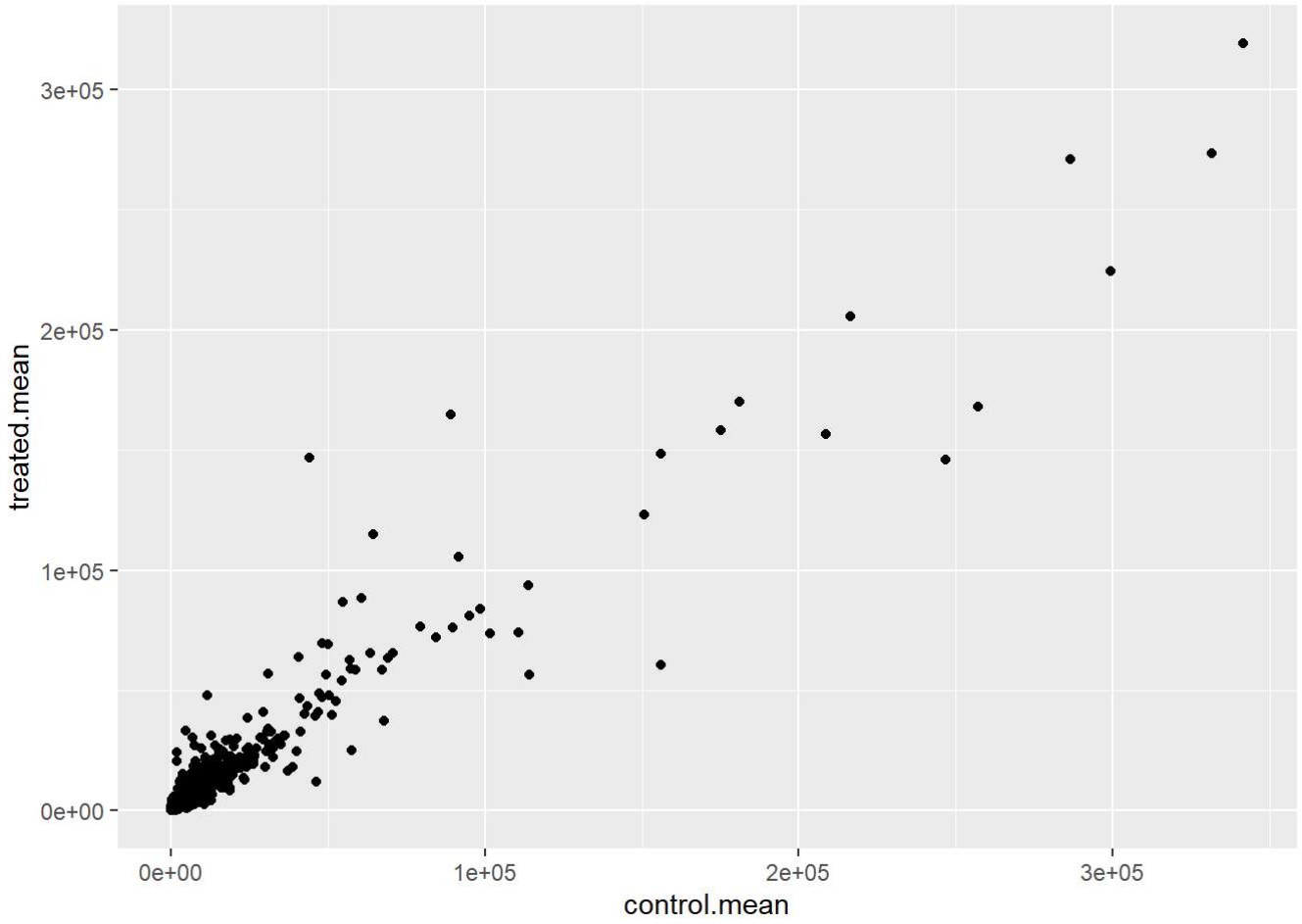


Q5(b): You could also use the ggplot2 package to make this figure producing the plot below. What geom_?() function would you use for this plot?

Answer: geom_point() would be used for this plot.

```
library(ggplot2)

ggplot(meancounts) + aes(control.mean, treated.mean) + geom_point()
```



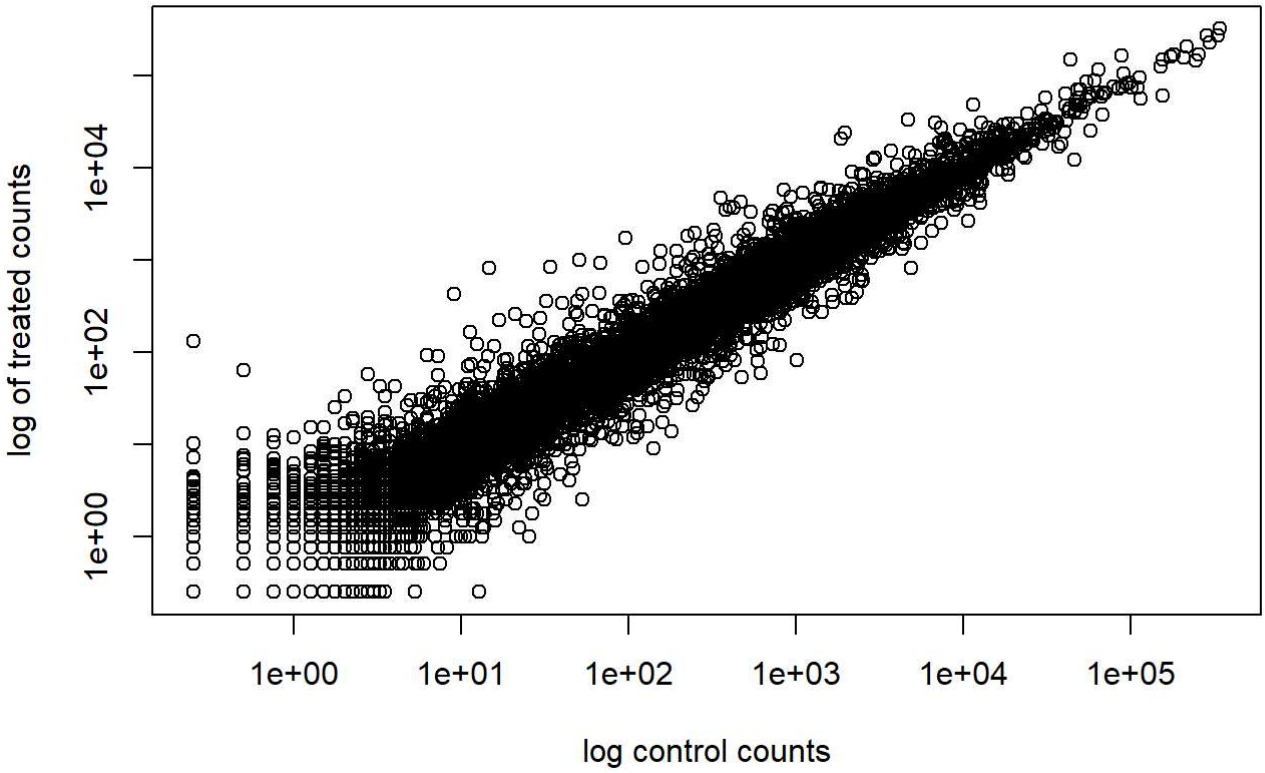
Q6: Try plotting both axes on a log scale. What is the argument to plot() that allows you to do this?

Answer: the 'log' argument.

```
plot(meancounts[ , 1], meancounts[ , 2], log = "xy", xlab = "log control counts", ylab = "log of treated counts")
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 15032 x values <= 0 omitted
## from logarithmic plot
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 15281 y values <= 0 omitted
## from logarithmic plot
```



We can find candidate differentially expressed genes by looking for genes with a large change between control and dex-treated samples. The log2 of the fold change has better mathematical properties.

Calculate log2foldchange and add it to the meancounts data.frame:

```
meancounts$log2fc <- log2(meancounts$treated.mean / meancounts$control.mean)

head(meancounts)
```

```
##                 control.mean treated.mean      log2fc
## ENSG00000000003       900.75       658.00 -0.45303916
## ENSG00000000005         0.00         0.00         NaN
## ENSG00000000419       520.50       546.00  0.06900279
## ENSG00000000457       339.75       316.50 -0.10226805
## ENSG00000000460        97.25        78.75 -0.30441833
## ENSG00000000938         0.75         0.00        -Inf
```

NaN is returned when you divide by zero and try to take the log. The -Inf is returned when you try to take the log of zero.

Filter the data to remove those genes:

```
zero.vals <- which(meancounts[ , 1:2] == 0, arr.ind = TRUE)

to.rm <- unique(zero.vals[ , 1])
mycounts <- meancounts[-to.rm, ]
```

```
head(mycounts)
```

```
##                 control.mean treated.mean      log2fc
## ENSG00000000003       900.75       658.00 -0.45303916
## ENSG00000000419       520.50       546.00  0.06900279
## ENSG00000000457       339.75       316.50 -0.10226805
## ENSG00000000460        97.25        78.75 -0.30441833
## ENSG00000000971      5219.00      6687.50  0.35769358
## ENSG00000001036      2327.00      1785.75 -0.38194109
```

Q7: What is the purpose of the arr.ind argument in the which() function call above? Why would we then take the first column of the output and need to call the unique() function?

Answer: `arr.ind = TRUE` causes which() to return both the row and column indices where there are TRUE values. Therefore, we know which genes (rows) and samples (columns) have zero counts. We take the first column of the output to exclude genes with zero counts in any sample - so we only want rows, not columns. unique() ensures we don't count a row twice if it has a zero count in both samples.

A common threshold used for calling something differentially expressed is a log2(FoldChange) of greater than 2 or less than -2.

Let's filter the dataset both ways to see how many genes are up or down-regulated.

```
up.ind <- mycounts$log2fc > 2
down.ind <- mycounts$log2fc < (-2)
```

Q8: Using the up.ind vector above can you determine how many up regulated genes we have at the greater than 2 fc level?

```
sum(up.ind)
```

```
## [1] 250
```

Answer: There are 250 up regulated genes at greater than 2 fc level.

Q9: Using the down.ind vector above can you determine how many down regulated genes we have at the greater than 2 fc level

```
sum(down.ind)
```

```
## [1] 367
```

Answer: There are 367 down regulated genes at greater than 2 fc level.

Q10: Do you trust these results? Why or why not?

Answer: Not fully because we don't know yet if these changes are significant.

4. DESeq2 Analysis

First loading the package:

```
library(DESeq2)
citation("DESeq2")
```

```
##
## To cite package 'DESeq2' in publications use:
##
##   Love, M.I., Huber, W., Anders, S. Moderated estimation of fold change
##   and dispersion for RNA-seq data with DESeq2 Genome Biology 15(12):550
##   (2014)
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Moderated estimation of fold change and dispersion for RNA-seq data with DESeq
2},
##     author = {Michael I. Love and Wolfgang Huber and Simon Anders},
##     year = {2014},
##     journal = {Genome Biology},
##     doi = {10.1186/s13059-014-0550-8},
##     volume = {15},
##     issue = {12},
##     pages = {550},
##   }
```

Importing Data

DESeq works on a particular type of object called a DESeqDataSet. The DESeqDataSet is a single object that contains input values, intermediate calculations like how things are normalized, and all results of a differential expression analysis.

We will use the DESeqDataSetFromMatrix() function to build the required DESeqDataSet object and call it dds, short for our DESeqDataSet:

```
dds <- DESeqDataSetFromMatrix(countData = counts, colData = metadata, design=~dex)
```

```
## converting counts to integer mode
```

```
## Warning in DESeqDataSet(se, design = design, ignoreRank): some variables in
## design formula are characters, converting to factors
```

```
dds
```

```
## class: DESeqDataSet
## dim: 38694 8
## metadata(1): version
## assays(1): counts
## rownames(38694): ENSG00000000003 ENSG00000000005 ... ENSG00000283120
##   ENSG00000283123
## rowData names(0):
## colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
## colData names(4): id dex celltype geo_id
```

DESeq Analysis

The DESeq() function takes a DESeqDataSet and returns a DESeqDataSet, but with additional information filled in (including the differential expression results we are after). If we try to access these results before running the analysis, nothing exists.

Here we are running the DESeq pipeline on the dds object, and reassigning the whole thing back to dds, which will now be a DESeqDataSet populated with all those values.

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

```
## estimating size factors
```

```
## estimating dispersions
```

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

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

```
## final dispersion estimates
```

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

Getting results

We can get results out of the object simply by calling the results() function on the DESeqDataSet that has been run through the pipeline.

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

```
## log2 fold change (MLE): dex treated vs control
## Wald test p-value: dex treated vs control
## DataFrame with 38694 rows and 6 columns
##                   baseMean log2FoldChange     lfcSE      stat    pvalue
##                  <numeric>      <numeric> <numeric> <numeric> <numeric>
## ENSG00000000003  747.1942     -0.3507030  0.168246 -2.084470 0.0371175
## ENSG00000000005    0.0000             NA        NA        NA        NA
## ENSG00000000419  520.1342      0.2061078  0.101059  2.039475 0.0414026
## ENSG00000000457  322.6648      0.0245269  0.145145  0.168982 0.8658106
## ENSG00000000460   87.6826     -0.1471420  0.257007 -0.572521 0.5669691
## ...                    ...            ...       ...       ...       ...
## ENSG00000283115  0.000000             NA        NA        NA        NA
## ENSG00000283116  0.000000             NA        NA        NA        NA
## ENSG00000283119  0.000000             NA        NA        NA        NA
## ENSG00000283120  0.974916      -0.668258   1.69456 -0.394354  0.693319
## ENSG00000283123  0.000000             NA        NA        NA        NA
##                       padj
##                  <numeric>
## ENSG00000000003  0.163035
## ENSG00000000005        NA
## ENSG00000000419  0.176032
## ENSG00000000457  0.961694
## ENSG00000000460  0.815849
## ...                   ...
## ENSG00000283115        NA
## ENSG00000283116        NA
## ENSG00000283119        NA
## ENSG00000283120        NA
## ENSG00000283123        NA
```

Convert the res object to a data.frame with the as.data.frame() function and then pass it to View() to bring it up in a data viewer

We can summarize some basic tallies using the summary function.

```
summary(res)
```

```
##
## out of 25258 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 1563, 6.2%
## LFC < 0 (down)     : 1188, 4.7%
## outliers [1]       : 142, 0.56%
## low counts [2]     : 9971, 39%
## (mean count < 10)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

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:

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

```
##
## out of 25258 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)       : 1236, 4.9%
## LFC < 0 (down)     : 933, 3.7%
## outliers [1]       : 142, 0.56%
## low counts [2]     : 9033, 36%
## (mean count < 6)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

5. Adding Annotation Data

We will use one of Bioconductor's main annotation packages to help with mapping between various ID schemes. Here we load the AnnotationDbi package and the annotation data package for humans org.Hs.eg.db.

```
library("AnnotationDbi")
```

```
##
## Attaching package: 'AnnotationDbi'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

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

```
##
```

To get a list of all available key types that we can use to map between, use the columns() function:

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

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
##  [6] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"  "GENENAME"
## [11] "GENETYPE"     "GO"           "GOALL"        "IPI"          "MAP"
## [16] "OMIM"         "ONTOLOGY"     "ONTOLOGYALL"  "PATH"         "PFAM"
## [21] "PMID"         "PROSITE"      "REFSEQ"       "SYMBOL"       "UCSCKG"
## [26] "UNIPROT"
```

```
sessionInfo()
```

```
## R version 4.2.3 (2023-03-15 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22621)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] org.Hs.eg.db_3.16.0         AnnotationDbi_1.60.2
##  [3] ggplot2_3.4.2               dplyr_1.1.2
##  [5] DESeq2_1.38.3               SummarizedExperiment_1.28.0
##  [7] Biobase_2.58.0              MatrixGenerics_1.10.0
##  [9] matrixStats_1.0.0           GenomicRanges_1.50.2
## [11] GenomeInfoDb_1.34.9         IRanges_2.32.0
## [13] S4Vectors_0.36.2            BiocGenerics_0.44.0
## [15] BiocManager_1.30.21
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.6              sass_0.4.6              bit64_4.0.5
##  [4] jsonlite_1.8.5          bslib_0.5.0             highr_0.10
##  [7] blob_1.2.4              GenomeInfoDbData_1.2.9 yaml_2.3.7
## [10] pillar_1.9.0            RSQLite_2.3.1           lattice_0.21-8
## [13] glue_1.6.2              digest_0.6.31           RColorBrewer_1.1-3
## [16] XVector_0.38.0          colorspace_2.1-0        htmltools_0.5.5
## [19] Matrix_1.5-4.1          XML_3.99-0.14           pkgconfig_2.0.3
## [22] zlibbioc_1.44.0         xtable_1.8-4            scales_1.2.1
## [25] BiocParallel_1.32.6     tibble_3.2.1            annotate_1.76.0
## [28] KEGGREST_1.38.0         farver_2.1.1            generics_0.1.3
## [31] cachem_1.0.8            withr_2.5.0             cli_3.6.1
## [34] magrittr_2.0.3          crayon_1.5.2            memoise_2.0.1
## [37] evaluate_0.21           fansi_1.0.4             tools_4.2.3
## [40] lifecycle_1.0.3         munsell_0.5.0           locfit_1.5-9.7
## [43] DelayedArray_0.24.0     Biostrings_2.66.0       compiler_4.2.3
## [46] jquerylib_0.1.4         rlang_1.1.0             grid_4.2.3
## [49] RCurl_1.98-1.12         rstudioapi_0.14         labeling_0.4.2
## [52] bitops_1.0-7            rmarkdown_2.22          gtable_0.3.3
## [55] codetools_0.2-19        DBI_1.1.3               R6_2.5.1
## [58] knitr_1.43              fastmap_1.1.1           bit_4.0.5
## [61] utf8_1.2.3              parallel_4.2.3          Rcpp_1.0.10
```

```
## [64] vctrs_0.6.2            geneplotter_1.76.0     png_0.1-8
## [67] tidyselect_1.2.0       xfun_0.39
```