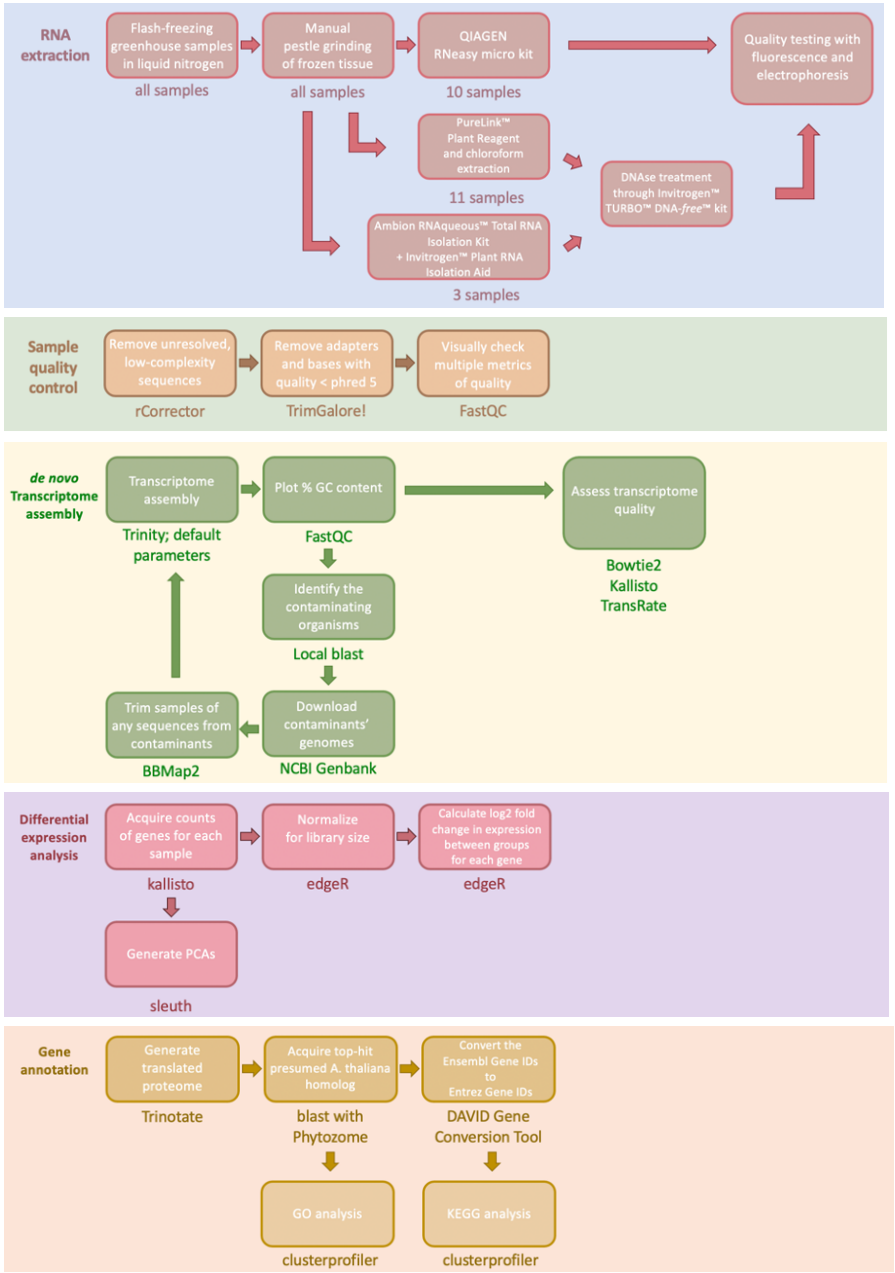
Abagail Burrus
Kramer Lab, July 7, 2022

Differential expression (DE) guide

The basis of all this guide's edgeR code was provided directly to me by Molly Edwards, who in turn received it from Anji Ballerini - it's a bit of a Kramer Lab tradition at this point!

This guide covers the purple background/pink-boxed portion of this RNA Seq workflow:



We use differential expression often to identify genes that may play a role in a trait of interest.

DE is most often pairwise comparison between two treatment groups. In my case, I made comparisons between tissue with glands and highly similar tissue without glands - by identifying what genes were significantly expressed in the glandular tissue compared to the eglandular tissue (thereby being *differentially* expressed in the glandular tissue) I tried to identify candidate genes for the glands' development.

DE analysis can play an important role in identifying candidates for functional studies, helping you find specific genes you might want to try targeting in knockdowns, knockouts, and/or overexpression studies. (It can also be followed-up with confirmatory experiments such as qRT-PCR and *in situ* hybridizations, which would be conducted before the functional work and might be as far as you can go with some plant groups, like the Malpighiaceae I studied.)

First we will work on the Cluster, and then we will work locally in RStudio (I guess you could also do all this R work on the Cluster as well, but I love the plotting ease that local RStudio gives me, and no line of code's processing took longer than 10 minutes.)

The R/R Studio package edgeR lets us do the following:

- Normalize our compared samples for library size
- Set a threshold for a gene's difference in expression between your treatment groups that should be met for that difference to be considered "significant" (such as a certain value of log2-fold expression change between your compared groups, or at least how many samples in each group must show this difference)
- Set a threshold of expression level a gene must have to be considered "present" (such as 0.5TPM /transcripts per million)
- Customize what type of comparison test we want to make (the most basic, and the type that I used, was "exact" tests)

Before we can use edgeR, we need counts of our genes expressed in our various samples. We accomplish this with the package kallisto.

Kallisto

Running kallisto as an interactive job, as I originally tried, repeatedly failed for me. Maybe it would go better for you. But this worked for me:
(This is all going faster now, as you're probably a lot more familiar with the Cluster by now.)

```
cd /n/holyscratch01/davis_lab/aburrus/testdata/
mkdir kallisto
```
^^^ Made a directory my kallisto analysis will go in

```
cd /n/holyscratch01/davis_lab/aburrus/testdata/kallisto
mkdir output
```
^^^ Made a directory for kallisto's output specifically

I used *nano* to make a .txt file of my sample names, with one sample per line
nano list.txt

I made a list of all my samples in copylist.txt (one sample per line)
Example of what I pasted into 'list.txt':
/n/holyscratch01/davis_lab/aburrus/testdata/output/NES31_1clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/NGS33_1clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/OES12_1clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/OGS31_1clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/OGS32_1clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/NES31_2clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/NGS33_2clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/OES12_2clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/OGS31_2clean01.fq
/n/holyscratch01/davis_lab/aburrus/testdata/output/OGS32_2clean01.fq

Copy all the files in the list, using this below, into the directory specified:
cp `cat copylist.txt` /n/holyscratch01/davis_lab/aburrus/testdata/kallisto

Now /n/holyscratch01/davis_lab/aburrus/testdata/kallisto has a copy of all my good, clean sample files in it

Here I copied my new transcriptome to where I wanted to do my kallisto analysis:
cp /n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fasta
/n/holyscratch01/davis_lab/aburrus/testdata/kallisto

The next major task was building the index kallisto uses for its counts.
For whatever reason, you first need to use *nano glands.idx* and put just a space in the file, writing it out/saving it that way. Make sure you're doing that in whatever directory you want to run your kallisto script.

Now make the following script as anything you like- I used kallisto0.sh:

```
#!/bin/bash
#SBATCH -n 32
#SBATCH -N 1
#SBATCH --mem 35000
#SBATCH -p serial_requeue
#SBATCH -o kallisto_%A_%a.out
#SBATCH -e kallisto_%A_%a.err
#SBATCH -J kallisto_arr

module load kallisto/0.45.1-fasrc01
export KALLISTO_DIR=/n/holyscratch01/davis_lab/aburrus/testdata/kallisto
kallisto index -i $TRANS_DATA/glands.idx
$TRANS_DATA/n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fasta
```

(Don't forget to change the file paths to your own.)

Next, run kallisto based on the index you built. I actually figured out array jobs by this point.

Array Jobs

My samples were named things like OGS11, OGS12, OGS13 etc- if your numbers have a consistent character chain, like sample1, sample2, sample3, you can run an array job that serializes work to each sample# variation in order you specify.
#Here's an example of one I did:

kallistoarray12.sh:

```
#!/bin/bash
#SBATCH -n 32
#SBATCH -N 1
#SBATCH --mem 7500
#SBATCH -p serial_requeue
#SBATCH -o kallisto_%A_%a.out
#SBATCH -e kallisto_%A_%a.err
#SBATCH -J kallisto_arr
#SBATCH -t 120
#SBATCH --array=11-13

source new-modules.sh
module load gcc openmpi kallisto

TRANS_DATA=/n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fasta
KALLISTO_DIR=/n/holyscratch01/davis_lab/aburrus/testdata/kallisto

dataset=OGS${SLURM_ARRAY_TASK_ID}

kallisto quant -i /n/holyscratch01/davis_lab/aburrus/testdata/kallisto/glands.idx -o
$KALLISTO_DIR/${dataset}.kallisto_out -b 100 -t 32  $KALLISTO_DIR/${dataset}_1clean01.fq
$KALLISTO_DIR/${dataset}_2clean01.fq
```

More explanations on the next page.

My dataset being OGS11, OGS12, OGS13, I called the constant part of that (OGS) below,

```
#!/bin/bash
#SBATCH -n 32
#SBATCH -N 1
#SBATCH --mem 7500
#SBATCH -p serial_requeue
#SBATCH -o kallisto_%A_%a.out
#SBATCH -e kallisto_%A_%a.err
#SBATCH -J kallisto_arr
#SBATCH -t 120
#SBATCH --array=11-13

source new-modules.sh
module load gcc openmpi kallisto

TRANS_DATA=/n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fasta
KALLISTO_DIR=/n/holyscratch01/davis_lab/aburrus/testdata/kallisto

dataset=OGS${SLURM_ARRAY_TASK_ID}

kallisto quant -i /n/holyscratch01/davis_lab/aburrus/testdata/kallisto/glands.idx -o
$KALLISTO_DIR/${dataset}.kallisto_out -b 100 -t 32  $KALLISTO_DIR/${dataset}_1clean01.fq
$KALLISTO_DIR/${dataset}_2clean01.fq
```

—–----------

I also called the sequence for it to run the job on below:

```
#!/bin/bash
#SBATCH -n 32
#SBATCH -N 1
#SBATCH --mem 7500
#SBATCH -p serial_requeue
#SBATCH -o kallisto_%A_%a.out
#SBATCH -e kallisto_%A_%a.err
#SBATCH -J kallisto_arr
#SBATCH -t 120
#SBATCH --array=11-13

source new-modules.sh
module load gcc openmpi kallisto

TRANS_DATA=/n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fasta
KALLISTO_DIR=/n/holyscratch01/davis_lab/aburrus/testdata/kallisto

dataset=OGS${SLURM_ARRAY_TASK_ID}

kallisto quant -i /n/holyscratch01/davis_lab/aburrus/testdata/kallisto/glands.idx -o
$KALLISTO_DIR/${dataset}.kallisto_out -b 100 -t 32  $KALLISTO_DIR/${dataset}_1clean01.fq
$KALLISTO_DIR/${dataset}_2clean01.fq
```

—–----------

I had weird trailing characters on my sample names, so the full sample names were files like “OGS11_1clean01.fq” and “OGS11_2clean01.fq” - but I got the rest of that name covered in the actual job line:

```
#!/bin/bash
#SBATCH -n 32
#SBATCH -N 1
#SBATCH --mem 7500
#SBATCH -p serial_requeue
#SBATCH -o kallisto_%A_%a.out
#SBATCH -e kallisto_%A_%a.err
#SBATCH -J kallisto_arr
#SBATCH -t 120
#SBATCH --array=11-13

source new-modules.sh
module load gcc openmpi kallisto

TRANS_DATA=/n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fasta
KALLISTO_DIR=/n/holyscratch01/davis_lab/aburrus/testdata/kallisto

dataset=OGS${SLURM_ARRAY_TASK_ID}

kallisto quant -i /n/holyscratch01/davis_lab/aburrus/testdata/kallisto/glands.idx -o
$KALLISTO_DIR/${dataset}.kallisto_out -b 100 -t 32  $KALLISTO_DIR/${dataset}_1clean01.fq
$KALLISTO_DIR/${dataset}_2clean01.fq
```

—----------

You can replicate my work here with this summarized guide. DON’T change the $KALLISTO_DIR/$ parts or any of that. Only the parts colored to match the parts of your own samples. Full layout is on the next page to keep it all together/easy to see:

Example sample names:

constantcharactersvariablenumberforsequence_1restoffilenameforforwardreads.fq
constantcharactersvariablenumberforsequence_2restoffilenameforreversereads.fq
constantcharactersvariablenumberforsequence_1restoffilenameforforwardreads.fq
constantcharactersvariablenumberforsequence_2restoffilenameforreversereads.fq

sepal144_1clean.fq
sepal144_2clean.fq
sepal145_1clean.fq
sepal145_2clean.fq
sepal146_1clean.fq
sepal146_2clean.fq

The constant characters would be these below: dataset=sepal${SLURM_ARRAY_TASK_ID}
The array here would be these numbers below: -- array= 144-146
The trailing sequence would be:
$KALLISTO_DIR/${dataset}_1clean.fq $KALLISTO_DIR/${dataset}_2clean.fq

(You could also say in that specific example that sepal14 is the constant characters and the array is just 4-6, but if you have many more samples, being able to capture more of them in an array like 144-168 or however far your samples go is more efficient.)

Basically, take your samples and customize the code below to match your samples (or tweak your sample names to fit an array):
Don't forget to edit the path names in green!

```
#!/bin/bash
#SBATCH -n 32
#SBATCH -N 1
#SBATCH --mem 7500
#SBATCH -p serial_requeue
#SBATCH -o kallisto_%A_%a.out
#SBATCH -e kallisto_%A_%a.err
#SBATCH -J kallisto_arr
#SBATCH -t 120
#SBATCH --array=11-13

source new-modules.sh
module load gcc openmpi kallisto

TRANS_DATA=/n/path/to/your/Trinity.fasta
KALLISTO_DIR=/n/path/to/your/kallisto

dataset=constantcharacters${SLURM_ARRAY_TASK_ID}

kallisto quant -i /n/holyscratch01/davis_lab/aburrus/testdata/kallisto/glands.idx -o
$KALLISTO_DIR/${dataset}.kallisto_out -b 100 -t 32  $KALLISTO_DIR/${dataset}_1restoffilenameforforwardreads.fq
$KALLISTO_DIR/${dataset}_2restoffilenameforreversereads.fq
```

With these arrays completed, submit them as whatever you made- I would do
sbatch kallistoarray12.sh
Or something like that.

Then, with those jobs completed, you need to take all your kallisto output and bring the results under one single umbrella folder. This is necessary for edgeR (and makes it much easier to download to our home computer systems).

```
cp -r sampleone.kallisto_out/ /n/holyscratch01/davis_lab/aburrus/testdata/kallisto/output
cp -r sampletwo.kallisto_out/ /n/holyscratch01/davis_lab/aburrus/testdata/kallisto/output
cp -r samplethree.kallisto_out/ /n/holyscratch01/davis_lab/aburrus/testdata/kallisto/output
```
You might need to dig around in your kallisto results a bit to find the names of your output.

Now, download that /output folder with your results to your local system. Rename the folders as necessary so it's easily interpreted. (For example, I changed folder NES12.kallisto_out to just be NES12).

EdgeR for DE

From now on, we work in RStudio.

All of the R code I have below is likely more accessible to you in R script format. I have it available at ___ in that format.
In case that's unavailable, however, it's all here too and you can copy+paste it all directly into an RStudio script with the same effect. The # denotes a commentary line rather than a code line. These commentary lines contain the instructions of what I did.

```
#Install necessary packages:
install.packages("dplyr")

#Have to install bioconductor:
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(version = "3.14")

#As an alternative to that bit, just use:
install.packages("BiocManager")
##Agree to restart, and if it prompts you again, just say "no" to restarting and it should proceed with installation
#
#Installing needed packages:
BiocManager::install(c("limma", "edgeR"))
#Go ahead and update any packages prompted to update by typing "a" at the [a/s/n] question if it's given to you. Otherwise:
BiocManager::install("devtools")

devtools::install_github("pachterlab/sleuth")

BiocManager::install("biomaRt")
#Say 'Yes' if prompted to install packages needing compilation (might be Mac-specific)

#We'll also grab sleuth for this - this will let us easily perform PCA on our samples
#To download sleuth, you need to work around a bug in the current (Feb 2022) coding for it:
#Download it to your computer somewhere with "git clone https://github.com/pachterlab/sleuth"
```

```
#Then open the NAMESPACE file in it and delete the last line (mentions .default)
#Then return to R and:
devtools::install('~/Downloads/sleuth/')


library("sleuth")
library("limma")
library("edgeR")
library("dplyr")
library("biomaRt")

BiocManager::install("rhdf5")
library("rhdf5")

install.packages("readr")
library("readr")

library("ggplot2")

setwd("~/Downloads/kallisto/")
#This needs to be wherever you put all your sample output folders from Kallisto on your home computer system.

#make a list of all the full file paths and turn it into a dataframe
Kallisto_paths <- list.files()
Kallisto_paths <- as.data.frame(Kallisto_paths)
Kallisto_paths$Kallisto_paths <- paste0("~/Downloads/kallisto/", Kallisto_paths$Kallisto_paths)

paths_list <- as.list(Kallisto_paths$Kallisto_paths)
paths <- unlist(paths_list, recursive = T, use.names = T)

Kallisto.output <- catchKallisto(paths, verbose = TRUE)

counts.kallisto <- as.data.frame(Kallisto.output$counts, header = F)
setwd("~/Downloads/deworktables/")
sample.info <- read.csv("sleuthsampleinfo.csv")
#This document of sample information is very important - it's how you tell the program to divide your samples' categories. Because I want to compare glandular and eglandular samples, for example, I have:
```

Sample	Rep	Tissue	Stage	Glands
NES11	1	Leaf	1	Eglandular
NES12	2	Leaf	1	Eglandular

NES13	3	Leaf	1	Eglandular
NES31	1	Leaf	3	Eglandular
NES32	2	Leaf	3	Eglandular
NES33	3	Leaf	3	Eglandular
NGS11	1	Leaf	1	Glandular
NGS12	2	Leaf	1	Glandular
NGS13	3	Leaf	1	Glandular
NGS31	1	Leaf	3	Glandular
NGS32	2	Leaf	3	Glandular
NGS33	3	Leaf	3	Glandular
OES11	1	Sepal	1	Eglandular
OES12	2	Sepal	1	Eglandular
OES13	3	Sepal	1	Eglandular
OES31	1	Sepal	3	Eglandular
OES32	2	Sepal	3	Eglandular
OES33	3	Sepal	3	Eglandular
OGS11	1	Sepal	1	Glandular
OGS12	2	Sepal	1	Glandular
OGS13	3	Sepal	1	Glandular
OGS31	1	Sepal	3	Glandular
OGS32	2	Sepal	3	Glandular
OGS33	3	Sepal	3	Glandular

#Stage is 1= early, 3= late; Glands is whether they have glands or not; Rep is replicates 1-3 for each group. The sample names are just coded combinations of all that information that I used for ease in my study - don't worry about that.

Code continues on next page

```
colnames(counts.kallisto) <- sample.info[,1]
k.gene.list <- rownames(counts.kallisto)
k.gene.list <- as.data.frame(k.gene.list)
```

```
###Sepal glandular vs eglandular for early tissue

data <- DGEList(counts.kallisto, genes=k.gene.list)
cpm <- cpm(data)
data <- calcNormFactors(data)
data.normalized <- cpm(data, normalized.lib.sizes=TRUE)


write.table(data.normalized, file="edgeR_sample_normalized_data.txt", quote=FALSE, sep="\t",
row.names=TRUE, col.names=TRUE)
#Making subsets for comparisons:
counts.leaf.egl.1 <- counts.kallisto[,sample.info$Tissue == "Leaf" & sample.info$Glands ==
"Eglandular" & sample.info$Stage == "1"]
counts.leaf.egl.3 <- counts.kallisto[,sample.info$Tissue == "Leaf" & sample.info$Glands ==
"Eglandular" & sample.info$Stage == "3"]

counts.leaf.gla.1 <- counts.kallisto[,sample.info$Tissue == "Leaf" & sample.info$Glands ==
"Glandular" & sample.info$Stage == "1"]
counts.leaf.gla.3 <- counts.kallisto[,sample.info$Tissue == "Leaf" & sample.info$Glands ==
"Glandular" & sample.info$Stage == "3"]

counts.sepal.egl.1 <- counts.kallisto[,sample.info$Tissue == "Sepal" & sample.info$Glands ==
"Eglandular" & sample.info$Stage == "1"]
counts.sepal.egl.3 <- counts.kallisto[,sample.info$Tissue == "Sepal" & sample.info$Glands ==
"Eglandular" & sample.info$Stage == "3"]

counts.sepal.gla.1 <- counts.kallisto[,sample.info$Tissue == "Sepal" & sample.info$Glands ==
"Glandular" & sample.info$Stage == "1"]
counts.sepal.gla.3 <- counts.kallisto[,sample.info$Tissue == "Sepal" & sample.info$Glands ==
"Glandular" & sample.info$Stage == "3"]

#This is comparing the glandular and eglandular leaves for each stage in each tissue type
counts.sepal.gla.egl.1 <- cbind(counts.sepal.gla.1, counts.sepal.egl.1)
counts.sepal.gla.egl.3 <- cbind(counts.sepal.gla.3, counts.sepal.egl.3)
counts.leaf.gla.egl.1 <- cbind(counts.leaf.gla.1, counts.leaf.egl.1)
counts.leaf.gla.egl.3 <- cbind(counts.leaf.gla.3, counts.leaf.egl.3)

#Prepping to do lots of edgeR runs
group=factor(c(1,1,1,2,2,2))
#^^You have to change this to the number of replicates you have that you're comparing. For
Molly's data of 5 replicates, for example, it was this:
#group=factor(c(1,1,1,1,1,2,2,2,2,2))
```

```
run.edgeR.kallisto  <- function(data, c.p.m, model, output) {

  all <- DGEList(data, genes=k.gene.list, group=model)

  dim(all)
  cpm <- cpm(all)
  all <- all[rowSums(cpm >= c.p.m) >= 3,]  #keep rows where cpms for a transcript are greater than whatever minimum we set, if at least three samples of that transcript have a greater cpm than that (we set it below when we call the function). ***IF YOU HAVE OTHER THAN 3 REPS YOU NEED TO CHANGE THIS NUMBER***

  all <- calcNormFactors(all)

  design <- model.matrix(~group)
  #design <- model.matrix(~model)

  rownames(design) <- colnames(all)
  #estimate dispersions of every transcript
  all <- estimateGLMCommonDisp(all, design, verbose=TRUE)
  all <- estimateGLMTrendedDisp(all, design)
  all <- estimateGLMTagwiseDisp(all, design)
  #different tests that edgeR can do - QLFit, QLF, exactTest - other options for more complex models. Anji wound up using exact Test for most of it
  fit.ql <- glmQLFit(all, design)
  qlf <- glmQLFTest(fit.ql)

  et <- exactTest(all)
  topTags(et) #printing in R window

  fit <- glmFit(all, design)
  lrt <- glmLRT(fit)
  topTags(lrt)
  #summary command will print "this many genes up, this many genes down" etc
  sum.de.lrt <- summary(de <- decideTestsDGE(lrt))
  sum.de.et <- summary(de <- decideTestsDGE(et))
  sum.de.qlf <- summary(de <- decideTestsDGE(qlf))

  stats.lrt <- c(all$common.dispersion, sum.de.lrt)
  stats.et <- c(all$common.dispersion, sum.de.et)
  stats.qlf <- c(all$common.dispersion, sum.de.qlf)
  #write.table(stats.lrt, file=paste("sum.de.lrt", c.p.m, output, "txt", sep="."), quote=FALSE, sep="\t")
  #write table, paste command will tell it to start with sum.de.et and string together cpm, etc etc separated by .'s
```

```
  write.table(stats.et, file=paste("sum.de.et", c.p.m, output, "txt", sep="."), quote=FALSE,
sep="\t")
  #write.table(stats.qlf, file=paste("sum.de.qlf", c.p.m, output, "txt", sep="."), quote=FALSE,
sep="\t")

  #passed.filter.lrt <- summary(de <- decideTestsDGE(lrt))[2,1]
  passed.filter.et <- summary(de <- decideTestsDGE(et))[2,1]
  #passed.filter.qlf <- summary(de <- decideTestsDGE(qlf))[2,1]

  #results.all.lrt <- topTags(lrt, n=passed.filter.lrt)$table
  #results.significant.all.lrt <- results.all.lrt[(results.all.lrt$FDR <= 0.05), ]
  results.all.et <- topTags(et, n=passed.filter.et)$table
  results.significant.all.et <- results.all.et[(results.all.et$FDR <= 0.05), ]
  #results.all.qlf <- topTags(qlf, n=passed.filter.qlf)$table
  #results.significant.all.qlf <- results.all.qlf[(results.all.qlf$FDR <= 0.05), ]

  #write.table(results.all.lrt, file=paste("results.all.lrt.cpm", c.p.m, output, "txt", sep="."),
quote=FALSE, sep="\t")
  #write.table(results.significant.all.lrt, file=paste("results.sig.lrt.cpm", c.p.m, output, "txt",
sep="."), quote=FALSE, sep="\t")

  write_csv(results.all.et, file=paste("results.all.et.cpm", c.p.m, output, "csv", sep="."))
  write_csv(results.significant.all.et, file=paste("results.sig.et.cpm", c.p.m, output, "csv", sep="."))
  #write.table(results.all.qlf, file=paste("results.all.qlf.cpm", c.p.m, output, "txt", sep="."),
quote=FALSE, sep="\t")
  #write.table(results.significant.all.qlf, file=paste("results.sig.qlf.cpm", c.p.m, output, "txt",
sep="."), quote=FALSE, sep="\t")

  #mds plot is sort of a principal component plot - hopefully your data will cluster as you expect
  pdf(file=paste("MDS.cpm", c.p.m, output, "pdf", sep="."))
  plotMDS(all, method="bcv")
  dev.off()

  #pdf(file=paste("volcano.cpm.lrt", c.p.m, output, "pdf", sep="."))
  #plot(results.all.lrt$logFC, -log(results.all.lrt$FDR))
  #dev.off()

  #volcano plot log fold change on x axis, y axis is significance of DE (tighter clustering of three
reps- consistent pattern across reps)
  pdf(file=paste("volcano.cpm.et", c.p.m, output, "pdf", sep="."))
  plot(results.all.et$logFC, -log(results.all.et$FDR))
  dev.off()

  #pdf(file=paste("volcano.cpm.qlf", c.p.m, output, "pdf", sep="."))
```

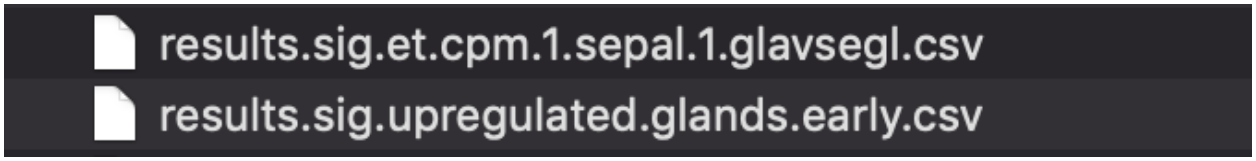
```
  #plot(results.all.qlf$logFC, -log(results.all.qlf$FDR))
  #dev.off()


  pdf(file=paste("cpm.FC", c.p.m, output, "pdf", sep="."))
  plot(results.all.et$logCPM, results.all.et$logFC)
  dev.off()

}

#setwd("wherever you want your function to output things")
setwd("~/Downloads/GLAND_TISSUES_PER_GROUP/sepalstage1/")
run.edgeR.kallisto(counts.sepal.gla.egl.1, c.p.m=1, group, "sepal.1.glavsegl")
#What I got for mine: Disp = 0.64884 , BCV = 0.8055
setwd("~/Downloads/GLAND_TISSUES_PER_GROUP/sepalstage3/")
run.edgeR.kallisto(counts.sepal.gla.egl.3, c.p.m=1, group, "sepal.3.glavsegl")
#What I got for mine: Disp = 0.83142 , BCV = 0.9118
setwd("~/Downloads/GLAND_TISSUES_PER_GROUP/leafstage1/")
run.edgeR.kallisto(counts.leaf.gla.egl.1, c.p.m=1, group, "leaf.1.glavsegl")
#What I got for mine: Disp = 1.00155 , BCV = 1.0008
setwd("~/Downloads/GLAND_TISSUES_PER_GROUP/leafstage3/")
run.edgeR.kallisto(counts.leaf.gla.egl.3, c.p.m=1, group, "leaf.3.glavsegl")
#What I got for mine: Disp = 0.93664 , BCV = 0.9678
```

#You then navigate to where you told the results to go and interpret them from there. The “sig” list gives you what’s significantly differentially expressed. Here’s an example of what mine look like for my sepalstage1 results:



#For me, the way I set this code up, the genes upregulated in glandular tissue had a NEGATIVE value of log2fold change. The output there doesn’t tell you which group has it upregulated, so you have to judge for yourself.
#You can check if this directionality of what’s upregulated vs. downregulated for your treatment groups if you pull up on your computer this file in your results directory in a text editor: edgeR_sample_normalized_data.txt
This file will tell you the expression counts for each sample, and you can say then something like “oh the glandular samples here have really high count values and the eglandular samples don’t express it - so this must be upregulated in the glandular tissue, which is negative in my significant genes list.” Then you can sort your significantly upregulated genes list by the log2

fold change and all the negative values indicate genes upregulated in glandular tissue. (This is one long example but I hope you see how you can apply it to your data!)

#Bottom line is, most of the edgeR work is just customizing the edgeR function to your samples/the comparisons you specifically want to make.

Generating PCAs

Principal Component Analysis (PCA) is often used to judge the ordination of differences between your samples. Ideally, samples within a treatment group should be more closely related to each other than to samples in different treatment groups. Based on this, we hope to see clustering of our samples in sensible ways that let the variation of space between the clusters be indicative of treatment group difference, not random inter-sample variation. (In the latter case, our DE analysis might not be useful at all.)

We generate our PCAs with the R package sleuth, which we already installed.

```
#Set directory
setwd("~/Downloads/deworktables/")
sample_id2 <- dir(file.path("~/Downloads/sepalkallisto/"))
#Reading in the information like before for our samples:
sepp<-read.csv("sepalsampleinfo.csv",header = TRUE, stringsAsFactors=FALSE)
kal_dirs2 <- file.path("~/Downloads/sepalkallisto/", sample_id2)
sepp <- mutate(sepp, path = kal_dirs2)
se <- sleuth_prep(sepp, extra_bootstrap_summary = TRUE,read_bootstrap_tpm=TRUE)

#You can color by different groups you want to test the clustering of. For me, I cared about whether the samples separated based on whether they had glands or not, and if they separated into early vs. late stages
#Looking based on stages:
plot_pca(se, color_by = 'Stage')
#Looking based on glands:
plot_pca(se, color_by = 'Glands')

#This plots PC2 and PC3 instead of the default PC1 and PC2 we saw above
plot_pca(se, pc_x = 2L, pc_y = 3L, color_by = 'Glands')

plot_pca(se, pc_x = 2L, pc_y = 3L, color_by = 'Glands')

#If you want your samples labeled with their names instead of as circles or other shapes:
plot_pca(se, pc_x = 2L, pc_y = 3L, color_by = 'Glands', text_labels = TRUE)
```