Summary of RNA-seq data cleaning

J. Shah chimeric mouse collaboration

Kim Dill-McFarland, kadm@uw.edu

version May 13, 2020

Contents

Pipeline overview

Command line tools

1. Combine `.fastq.gz` files per read per sample
2. Remove sequencing adapters
3. Quality filter sequences
4. Align to reference genome
5. Quality filter alignments
6. Count reads in genes

R

7. Filter protein coding genes
8. Filter low coverage samples
9. Filter PCA outliers
10. Filter rare genes
11. Normalize counts

RNA-seq data cleaning

1. Combine files

Samples were sequenced as 50 bp paired-end reads on a multi-lane Illumina sequencer. Therefore, there are multiple files for each sample and read. These are combined by sample and read (R1, R2) using the standard Unix function `cat`. The resulting files are in `data_raw/fastq_merge/`

For example:

```
cat Lu1*R1_00[0-9].fastq.gz > Lu01_R1.fastq.gz
cat Lu1*R2_00[0-9].fastq.gz > Lu01_R2.fastq.gz
```

2. Adapters and 3. quality filtering

Sequence assessment 1

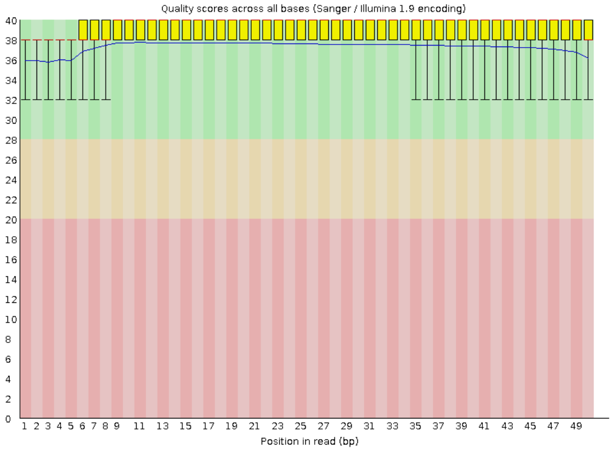
Sequences were assessed using FastQC to visualize sequence quality and determine if adapters exist.

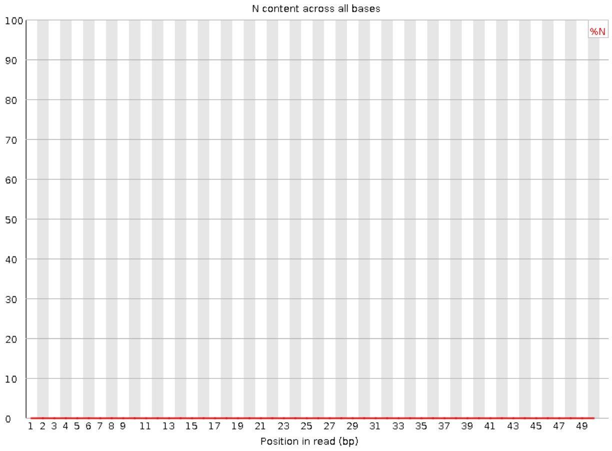
Results can be found in `results/results_fastqc/fastqc_raw/`

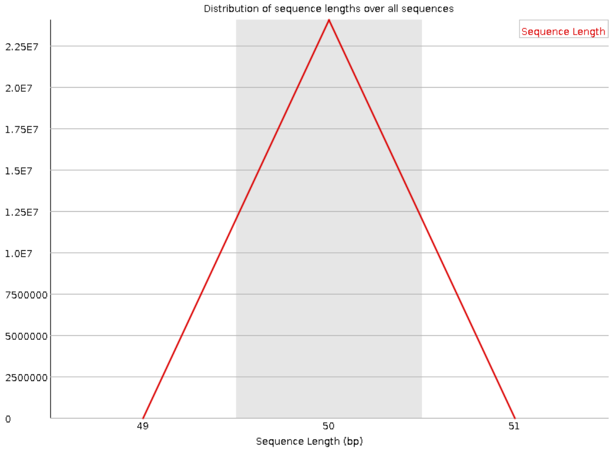
For example:

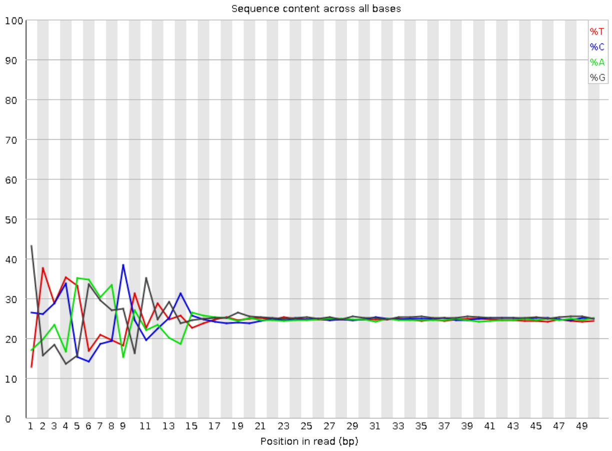
```
fastqc Lu01_R1.fastq.gz \
      -o ~/results/results_fastqc/fastqc_raw/ \
      -t 15
```

Resulting in:







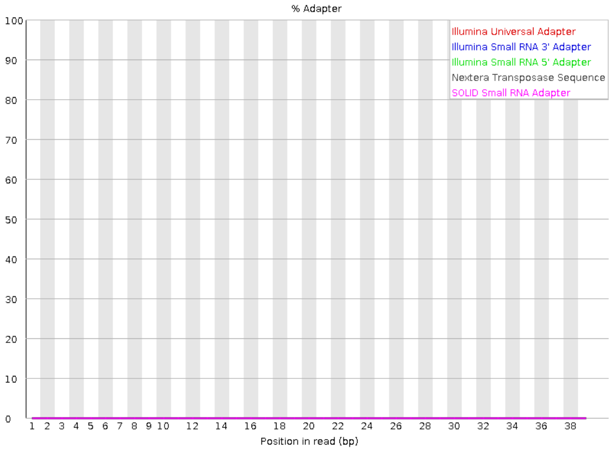


Results are similar for all samples.

Overall, read quality is high with Phred scores > 30 (upper left), minimal ambiguous bases pairs (upper right), and full length 50 bp reads (lower left).

However, there appear to be adapters present in these reads (lower right). RNA-seq randomly samples pieces of RNA expressed from the genome. Thus, if you stack all reads from a sample, each position should contain roughly 25% of each bp (ACTG). In these samples, we see that the first 15 bp are not at the expected 25%. This most commonly indicates sequencing adapters, because these sequences are the same for all reads, thus skewing the apparent bp proportions. The adapters appear in the results, because pieces of sample DNA are < 50 bp and the machine sequenced past the end of the “real data” into the adapter. The proportions along adapters are not 100%, because there is some variation in sample DNA length that causes the adapters on some sequences to be a few bp shifted relative to other sequences.

Since the adapters do not represent real biological data, they need to be removed. Also, note that these adapters are not flagged in the “Adapter Content” plot (below), because the FastQC default list does not contain all current Illumina adapter sequences.



Sequence filtering

Using AdapterRemoval, sequences are filtered to

- hard cut the 5' end of sequences by 15 bp, thus removing adapters (`trim5p`)
- remove reads with > 1 ambiguous base (`maxns`)
- trim 5' and 3' ends until reach base with quality of 30+ (`trimqualities minquality`)
- remove reads < 15 bp long after both end trimming steps (`minlength`)

Results can be found in `data_raw/fastq_trim/`. Note that this program changes read names from R1 > pair1 and R2 > pair2.

For example:

```
AdapterRemoval \
    --file1 Lu01_R1.fastq.gz \
    --file2 Lu01_R2.fastq.gz \
    --basename data_raw/fastq_trim/Lu01 --gzip \
    --trim5p 15 --maxns 1 --minlength 15 \
    --trimqualities --minquality 30 \
    --threads 15
```

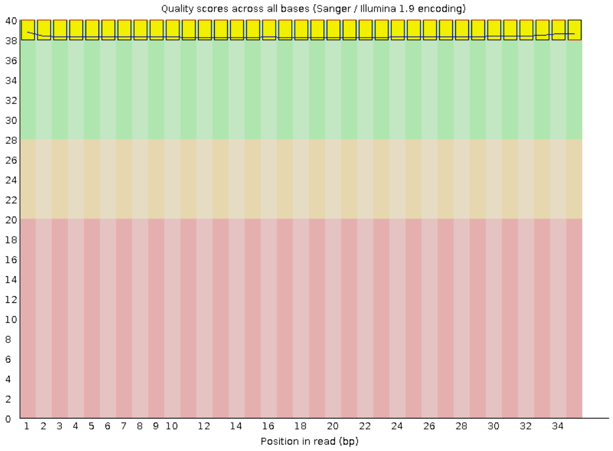
Sequence assessment 2

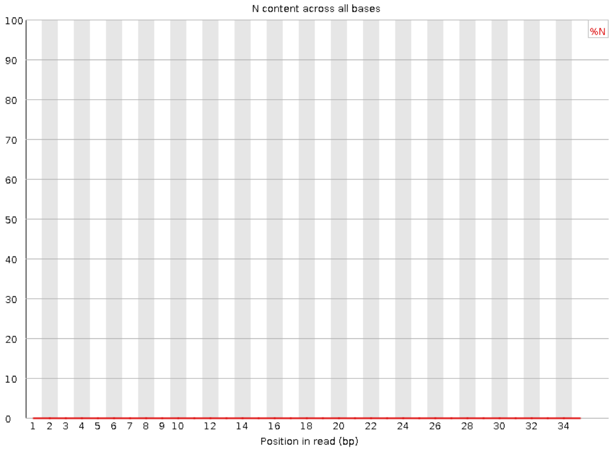
Using FastQC again, assess the quality of the filtered reads. Results can be found in `results/results_fastqc/fastqc_trim/`

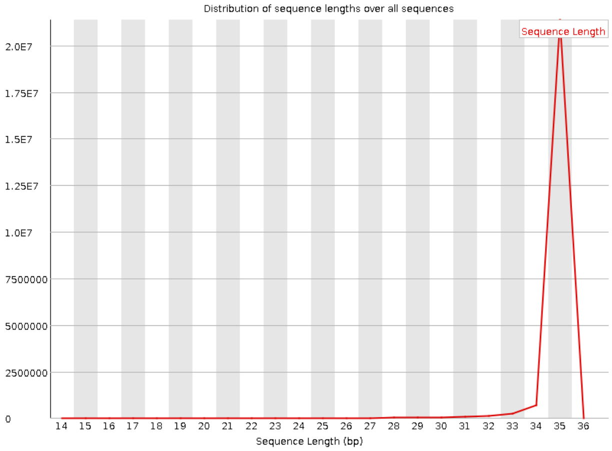
For example:

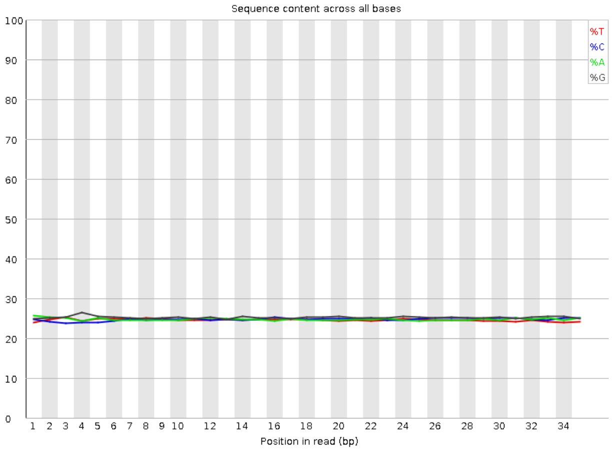
```
fastqc Lu01.pair1.truncated.gz \
      -o ~/results/results_fastqc/fastqc_trim/ \
      -t 15
```

Resulting in:









Results are similar for all samples.

Filtering achieved the desired outcomes as seen in the above plots.

4. Alignment

Sequences were aligned to the mouse reference genome (GRCm38 release 99 aka mm10) using STAR. This aligner is optimized for short-read data such as RNA-seq. Therefore, defaults were maintained.

Results can be found in `data_raw/bam`

For example:

```
#Download reference
sudo curl -O ftp://ftp.ensembl.org/pub/release-99/gtf/mus_musculus/Mus_musculus.GRCm38.99.gtf.gz
gunzip Mus_musculus.GRCm38.99.gtf.gz

sudo curl -O ftp://ftp.ensembl.org/pub/release-99/fasta/mus_musculus/dna/Mus_musculus.GRCm38.dna.primar
gunzip Mus_musculus.GRCm38.dna.primary_assembly.fa.gz

#Index reference
STAR --runMode genomeGenerate \
     --genomeDir STARindex.mouse \
     --genomeFastaFiles Mus_musculus.GRCm38.dna.primary_assembly.fa \
     --sjdbGTFfile Mus_musculus.GRCm38.99.gtf \
     --sjdbOverhang 99 \
     --runThreadN 15

#Align
STAR --genomeDir STARindex.mouse \
     --readFilesIn Lu01.pair1.truncated.gz Lu01.pair2.truncated.gz \
     --readFilesCommand zcat \
     --outFileNamePrefix Lu01 \
     --outSAMtype BAM SortedByCoordinate \
     --runThreadN 15 \
     --runRNGseed 8756
```

5. Alignment filtering

Alignment assessment

Alignment quality was assessed with Picard and samtools flagstat.

Picard results can be found in `results/results_cleaning/bam.metrics.tsv` and flagstat in `results/results_cleaning/summary.alignment.tsv`.

For example:

```
#Download Picard reference
sudo curl -O http://hgdownload.cse.ucsc.edu/goldenpath/mm10/database/refFlat.txt.gz
gunzip -c refFlat.txt.gz > refFlat.mouse.txt
## Remove chr in chromosome name to match ensembl alignment
sed 's/chr//' refFlat.mouse.txt > refFlat.mouse.ensembl.txt

#Assess alignment with Picard
java -XX:ParallelGCThreads=15 \
    -jar picard-2.22.1-0/picard.jar \
    CollectRnaSeqMetrics \
    REF_FLAT=refFlat.mouse.ensembl.txt \
    INPUT=Lu01_Aligned.sortedByCoord.out.bam \
    OUTPUT=results/results_cleaning/bam.metrics.tsv \
    ASSUME_SORTED=true \
    STRAND_SPECIFICITY=NONE \
    MINIMUM_LENGTH=500

#Assess alignment with samtools
samtools flagstat -@ 15 Lu01_Aligned.sortedByCoord.out.bam \
    >> results/results_cleaning/summary.alignment.tsv
```

Alignment filtering

Alignments were quality filtered using samtools view to:

- keep the file header (`h`)
- keep paired reads where both reads mapped to the genome (`-f 3`)
- remove unmapped reads, non-primary alignments, and PCR duplicates (`-F 1284`)
- remove reads with a mapping quality (MAPQ) < 30 (`-q 30`)

See this explanation of flags for more details on the numbers used here.

Results can be found in `data_raw/bam_filter_paired`

For example:

```
samtools view Lu01_Aligned.sortedByCoord.out.bam \
    -h -f 3 -F 1284 -q 30 \
    -@ 15 \
    > Lu01_filter_paired.bam
```

Alignment assessment 2

Filtering alignments were then re-assessed with samtools flagstat.

Results can be found in `results/results_cleaning/summary.align.filter.paired.tsv`

For example:

```
samtools flagstat -@ 15 Lu01_filter_paired.bam \
    >> results/results_cleaning/summary.align.filter.paired.tsv
```

6. Count reads in genes

Total reads in genes were quantified using SUBREAD featureCounts. Only reads in exons of known genes were counted.

Results can be found in `data_raw/counts`

For example:

```
featureCounts -T 14 -g gene_id -t exon -p \
  -a Mus_musculus.GRCm38.99.gtf \
  -o Shah.featurecounts.paired.tsv \
  bam_filter_paired/*filter_paired.bam
```

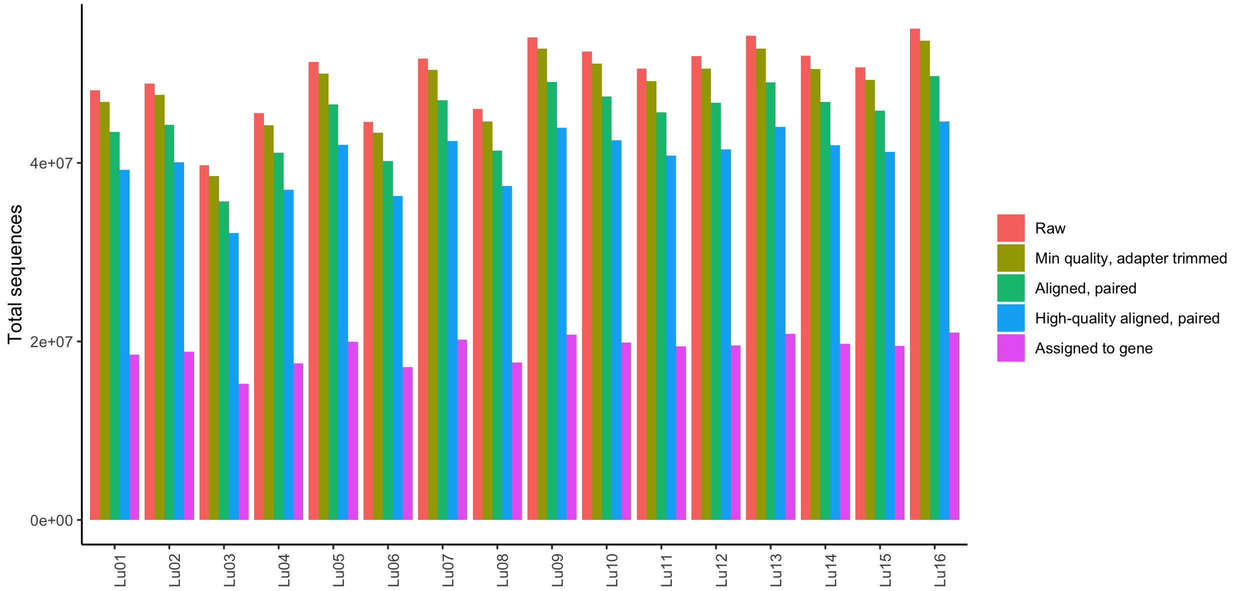
Counts were further formatted using `scripts/RNAseq_counts_formatting.R` to results in `data_clean/Shah.counts.clean.c`

```
## # A tibble: 55,471 x 17
##    geneName  Lu01  Lu02  Lu03  Lu04  Lu05  Lu06  Lu07  Lu08  Lu09  Lu10  Lu11
##    <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 ENSMUSG~     0     0     0     0     0     0     0     0     0     0     0
##  2 ENSMUSG~     0     0     0     0     0     0     0     0     0     0     0
##  3 ENSMUSG~     0     0     0     0     0     0     0     0     0     0     0
##  4 ENSMUSG~     0     0     0     0     0     0     0     0     0     0     0
##  5 ENSMUSG~     0     0     0     0     0     0     0     0     0     0     0
##  6 ENSMUSG~     1     0     0     0     0     0     0     0     0     0     0
##  7 ENSMUSG~     0     0     0     0     0     0     0     0     0     0     0
##  8 ENSMUSG~     0     0     0     0     0     0     0     0     0     0     0
##  9 ENSMUSG~     0     0     0     0     0     0     0     0     0     0     0
## 10 ENSMUSG~     1     0     1     1     1     0     0     0     0     0     0
## # ... with 55,461 more rows, and 5 more variables: Lu12 <dbl>, Lu13 <dbl>,
## #   Lu14 <dbl>, Lu15 <dbl>, Lu16 <dbl>
```

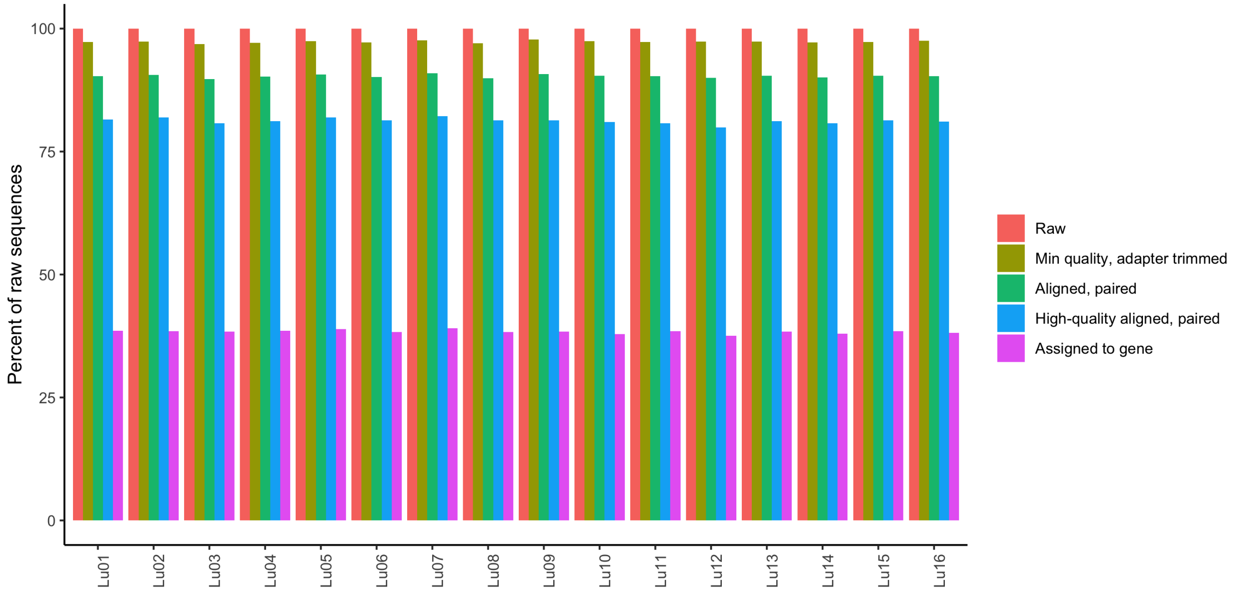
Summary 1

See `scripts/RNAseq_summary_formatting.R` for details on plot creation.

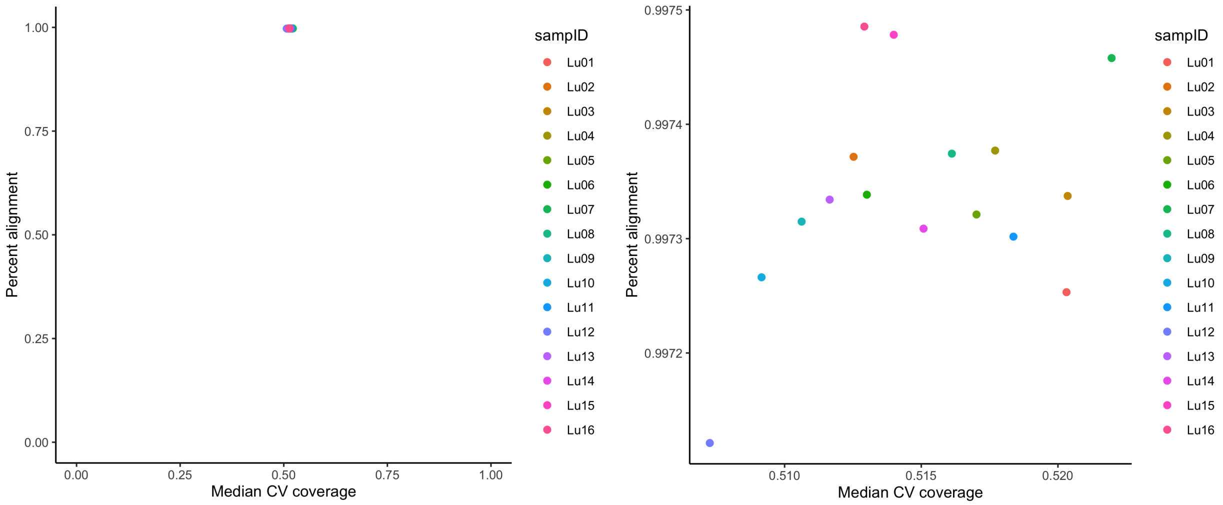
Total sequences retained during cleaning



Percent of raw sequences retained during cleaning



Final quality



Overall, these samples are very high-quality with > 32 million clean, aligned sequences per sample and low variability in gene coverage (*e.g.* median CV coverage). As expected, ~20% of raw sequences were removed for sequence or alignment quality.

A minimum of 15 million sequences were assigned to genes in each sample. This is more than sufficient for desired analyses.

Further data cleaning in R

Setup

Load packages

```
# Data manipulation and figures
library(tidyverse)
  # Modify ggplot data order within facets
  library(drlib)
  # Plot log scales
  library(scales)
  #Multi-panel figures
  library(cowplot)

# Empirical analysis of digital gene expression data
## Data normalization
library(edgeR)

# Print pretty table to knit file
library(knitr)
library(kableExtra)
  options(knitr.kable.NA = '')

#Create 'not in' operator
`%notin%` <- Negate(`%in%`)
```

Set seed

```
set.seed(4389)
```

Set variable names and cutoffs for this workflow.

```r
# Define cutoffs
#Median CV coverage MAXIMUM
CV.cut <- 0.65
#Alignment percentage with duplicates MINIMUM
align.cut <- 0.8
#Total sequences MINIMUM
count.cut <- 1E6
```

Read in and format data

Counts

```r
counts <- read_csv("data_clean/Shah.counts.clean.csv")
```

```
## Parsed with column specification:
## cols(
##   geneName = col_character(),
##   Lu01 = col_double(),
##   Lu02 = col_double(),
##   Lu03 = col_double(),
##   Lu04 = col_double(),
##   Lu05 = col_double(),
##   Lu06 = col_double(),
##   Lu07 = col_double(),
##   Lu08 = col_double(),
##   Lu09 = col_double(),
##   Lu10 = col_double(),
##   Lu11 = col_double(),
##   Lu12 = col_double(),
##   Lu13 = col_double(),
##   Lu14 = col_double(),
##   Lu15 = col_double(),
##   Lu16 = col_double()
## )
```

Metadata

```r
meta <- read_csv("data_clean/Shah.metadata.csv") %>%
  full_join(read_csv("data_clean/Shah.data.cleaning.metrics.csv"),
            by="sampID") %>%
  mutate(batch = as.character(batch),
         cell = factor(cell, levels=c("WT","TKO")),
         status = factor(status, levels=c("Uninfected",
                                          "Infected"))) %>%
  rename(libID=sampID)
```

```
## Parsed with column specification:
## cols(
##   sampID = col_character(),
##   batch = col_double(),
##   cell = col_character(),
##   status = col_character()
## )
```

Table 1: Total libraries

cell	status	n
WT	Uninfected	4
WT	Infected	4
TKO	Uninfected	4
TKO	Infected	4

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   sampID = col_character()
## )

## See spec(...) for full column specifications.
```

Check samples *all libraries in the count data are found in the metadata. And vice versa.*

```
##
##        TRUE
##   TRUE   16
```

Summarize samples

7. Filter protein coding genes

Load key. Filter gene key to protein coding (pc) genes that occur in the count data set and have valid MGI symbols.

```
#get key
library(biomaRt)
#List all genes in dataset
gene.names <- counts$geneName

#Get mouse reference genome
ensembl <- useEnsembl(biomart = "ensembl",
                      dataset = "mmusculus_gene_ensembl",
                      mirror = "uswest")

#Extract key from ref genome
key <- getBM(attributes=c("ensembl_gene_id",
                           "mgi_symbol",
                          "gene_biotype"),
              values=list(gene.names),
              mart=ensembl)

write_tsv(key, path="data_clean/ensembl.key.txt")
```

```
key.pc <- read_tsv("data_clean/ensembl.key.txt") %>%
  rename(geneName=ensembl_gene_id) %>%
  # Keep only valid MGI symbols
  filter(!is.na(mgi_symbol) & !is.na(geneName)) %>%
  # Keep protein coding genes only
  filter(gene_biotype == "protein_coding") %>%
  # Remove duplicate entries
  distinct(geneName, .keep_all=TRUE) %>%
  # Keep only genes found in dataset count.filter2
  filter(geneName %in% counts$geneName) %>%
  arrange(geneName)
```

Filter the count data to pc genes as well.

```
counts.pc <- counts %>%
  filter(geneName %in% key.pc$geneName) %>%
  arrange(geneName)
```

Check genes *all genes in the key are found in the data. And vice versa.*

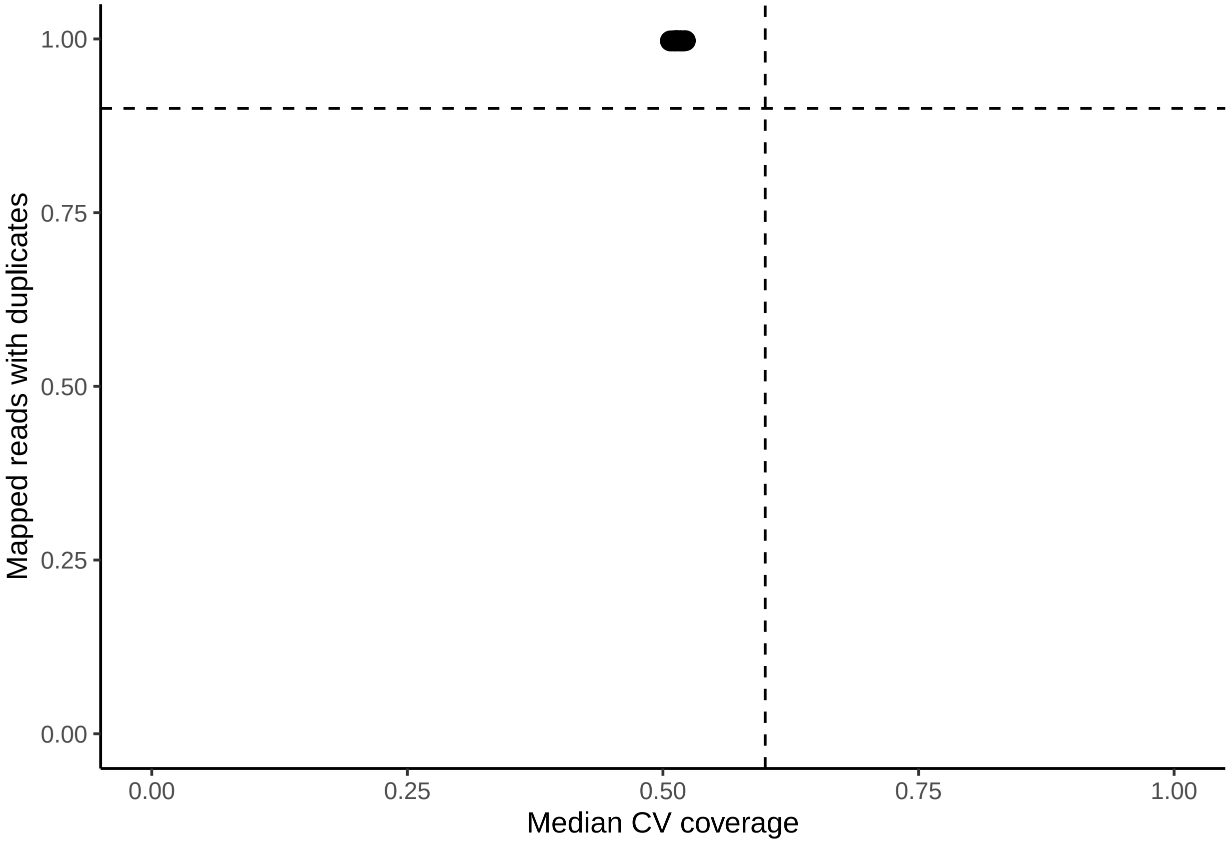
```
##
##          TRUE
##   TRUE 21850
```

8. Filter low coverage samples

Assess median CV coverage vs. mapped duplicate reads

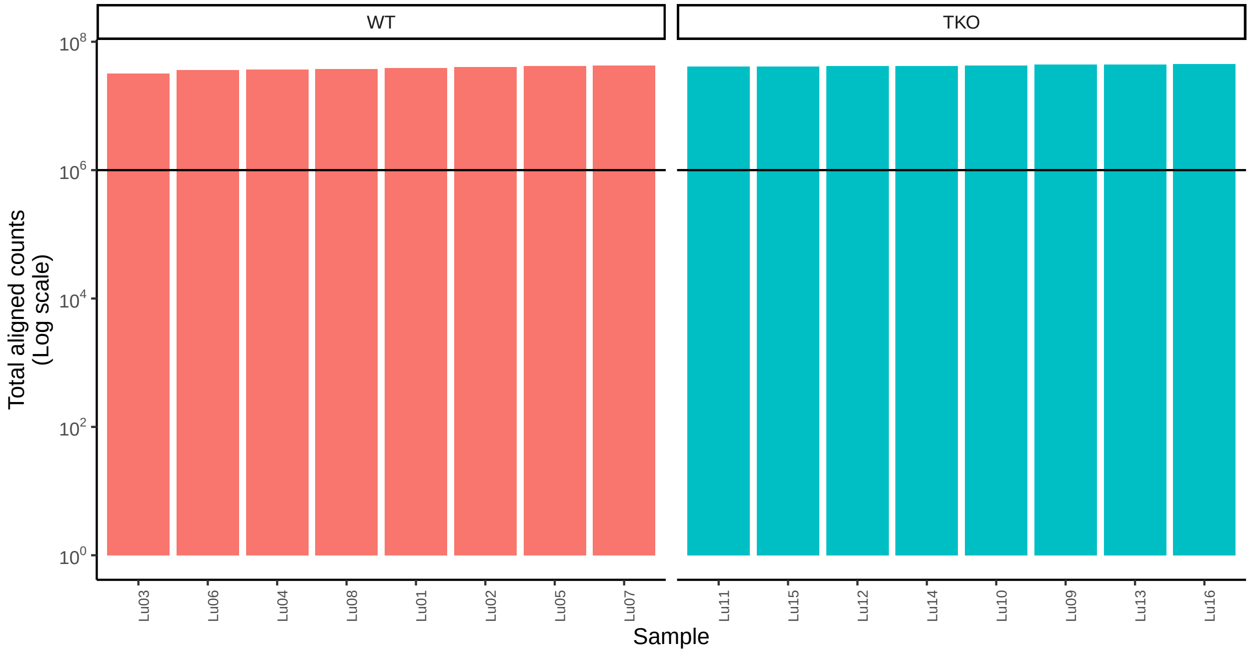
Compare the median coefficient of variation (CV) coverage (`median_cv_coverage`) and percent alignment of reads with duplicates (`mapped_reads_w_dups`). Ideally, you want libraries with LOW median CV coverage and HIGH percent aligned duplicates, indicating low variability in coverage over the length of genes and high coverage across the genome, respectively.

Plot CV coverage vs alignment.



Assess total aligned counts

Assess aligned counts per library. Higher counts indicate high coverage and are preferred. The minimum total sequences cutoff set above is indicated by a horizontal line.



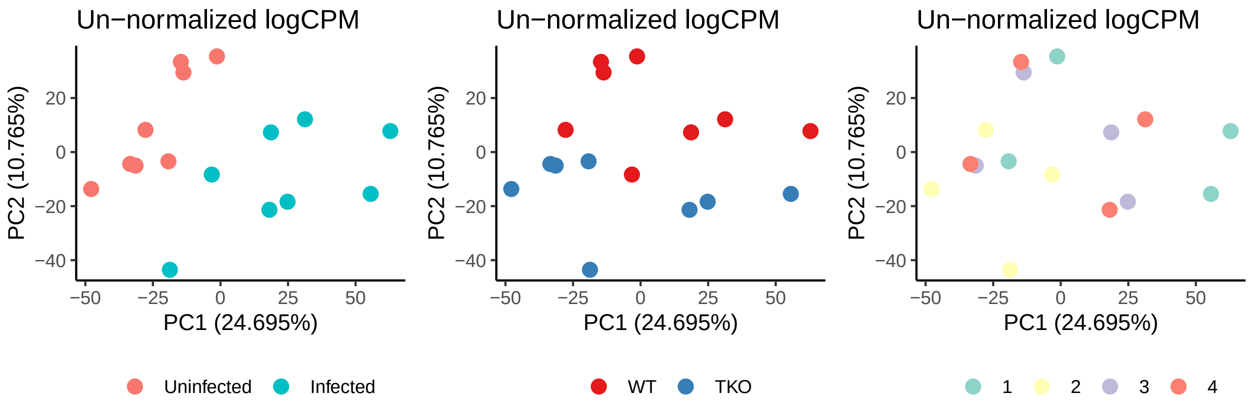
Filter by library coverage

No samples need to be removed.

9. Filter PCA outliers

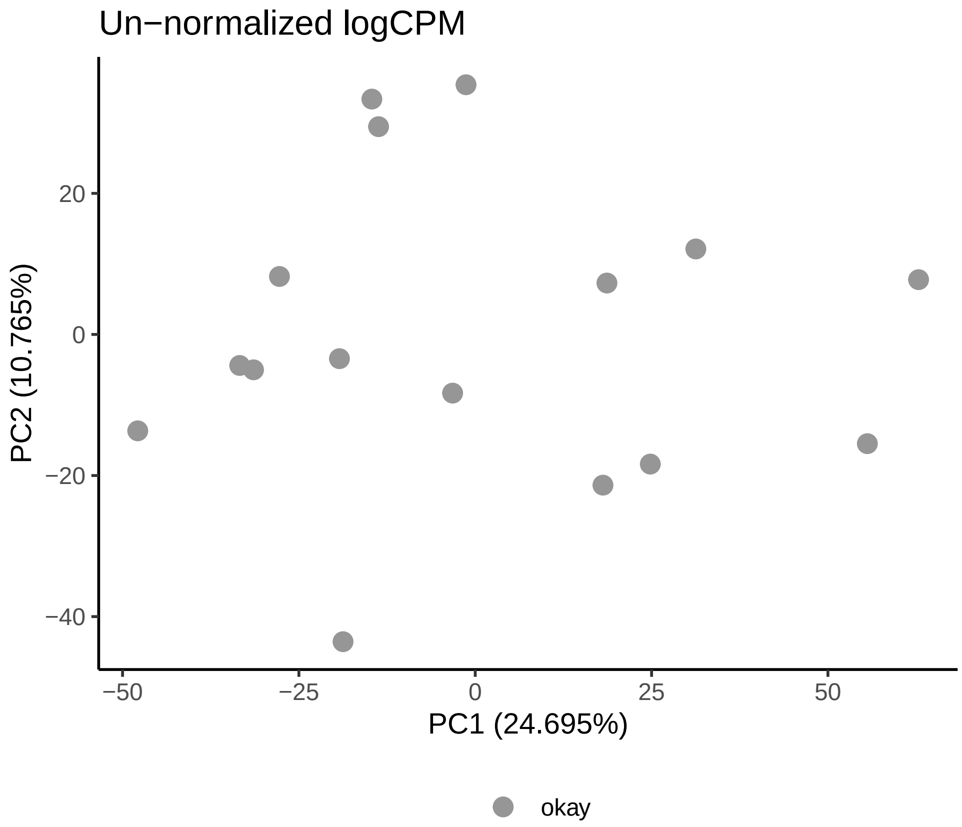
PCA of variables of interest

If one or more variables of interest represents significant variation in the data, consider further filtering and normalizing separately. This is most common when multiple sample or cell types exist in the data set.



PCA of potential outliers

Visualize PCA outliers defined as any library with PC1 and/or PC2 values greater than 3 standard deviations away from the PC mean.



Filter PCA outliers

No PCA outliers exist in the data.

10. Filter rare genes

Create DGEList object

For use in gene filtering with edgeR

```
dat.pc <- DGEList(
  #count table. move gene names to column names
  counts=as.matrix(column_to_rownames(counts.pc,
                                      "geneName")),
  #metadata
  samples=meta,
  #keep genes in count table
  genes=key.pc)
```

Filter rare genes

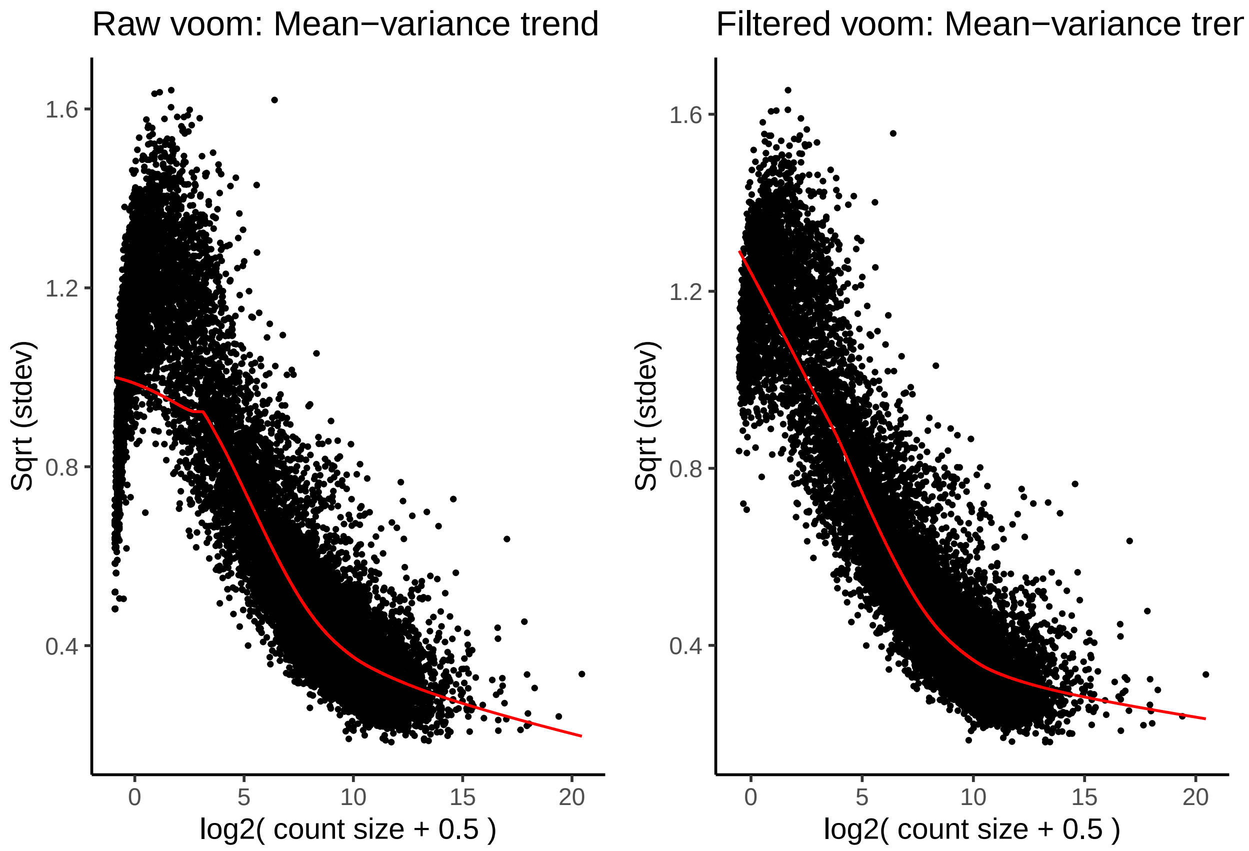
The raw gene sets contain highly variable, low abundance/rare genes.

Filters genes to those with `min.CPM` (counts per million) in at least `min.pct` (percent of samples) or `min.sample` (number of samples).

```
#Get Kim's function from GitHub
source("https://raw.githubusercontent.com/kdillmcfarland/R_bioinformatic_scripts/master/RNAseq_rare_gen

#Filter
```

```
rare.gene.filter(dat = dat.pc,
                 min.sample = 3,
                 min.CPM = 0.1,
                 name = "dat.pc.abund")
```



This removes 7635 (~ 35%) genes.

11. Normalize counts

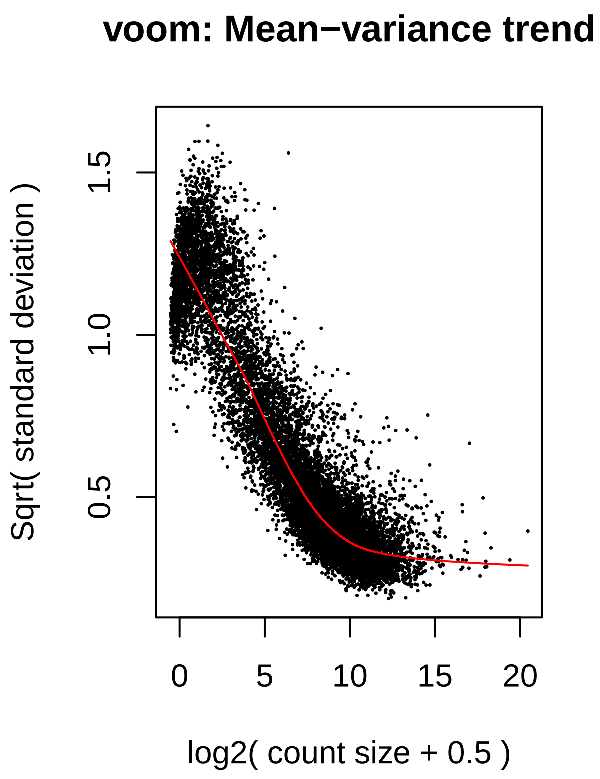
Normalize for RNA composition

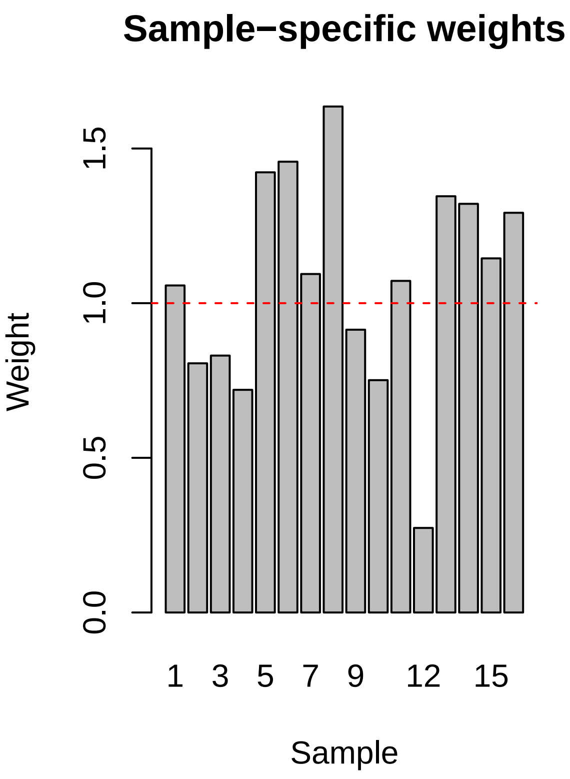
Calculate factors to scale library sizes.

```
dat.pc.abund.norm <- calcNormFactors(dat.pc.abund)
```

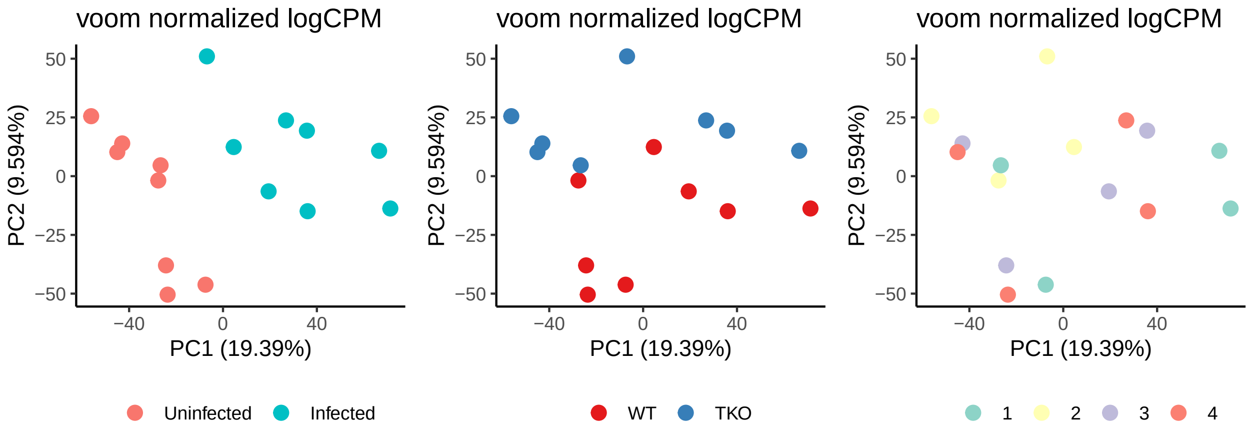
Normalize with voom

```
dat.pc.abund.norm.voom <- voomWithQualityWeights(
                          dat.pc.abund.norm,
                          design = model.matrix(VOI.model,
                                 data=  dat.pc.abund.norm$samples),
                          plot=TRUE)
```





Re-check for PCA outliers



Summary 2

No samples were removed due to low quality or coverage. In total, 7635 (~ 35%) genes were removed due to low abundance.

Save data

Write as RData

```
#Rename to short form
dat.voom <- dat.pc.abund.norm.voom

save(dat.voom, file="data_clean/Shah.clean.RData")
```

Write counts as table.

```
#Counts table
as.data.frame(dat.pc.abund.norm.voom$E) %>%
  rownames_to_column("geneName") %>%
write_csv("results/gene_level/Shah_gene_voom_counts.csv")
```

R session

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Catalina 10.15.4
## 
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
## 
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
## 
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base     
## 
## other attached packages:
##  [1] kableExtra_1.1.0 knitr_1.28       edgeR_3.26.8     limma_3.40.6    
##  [5] cowplot_1.0.0    scales_1.1.0     drlib_0.1.1      forcats_0.5.0   
##  [9] stringr_1.4.0    dplyr_0.8.5      purrr_0.3.4      tidyr_1.0.3     
## [13] tibble_3.0.1     ggplot2_3.3.0    tidyverse_1.3.0  readr_1.3.1     
## 
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.4.6       locfit_1.5-9.4     lubridate_1.7.8    lattice_0.20-41   
##  [5] assertthat_0.2.1   digest_0.6.25      utf8_1.1.4         R6_2.4.1          
##  [9] cellranger_1.1.0   backports_1.1.6    reprex_0.3.0       evaluate_0.14     
## [13] httr_1.4.1         pillar_1.4.4       rlang_0.4.6        readxl_1.3.1      
## [17] rstudioapi_0.11    Matrix_1.2-18      rmarkdown_2.1      labeling_0.3      
## [21] webshot_0.5.2      munsell_0.5.0      broom_0.5.6        compiler_3.6.1    
## [25] modelr_0.1.7       xfun_0.13          pkgconfig_2.0.3    htmltools_0.4.0   
## [29] tidyselect_1.0.0   fansi_0.4.1        viridisLite_0.3.0  crayon_1.3.4      
## [33] dbplyr_1.4.3       withr_2.2.0        grid_3.6.1         nlme_3.1-147      
## [37] jsonlite_1.6.1     gtable_0.3.0       lifecycle_0.2.0    DBI_1.1.0         
## [41] magrittr_1.5       cli_2.0.2          stringi_1.4.6      farver_2.0.3      
## [45] fs_1.4.1           xml2_1.3.2         ellipsis_0.3.0     generics_0.0.2    
## [49] vctrs_0.2.4        RColorBrewer_1.1-2 tools_3.6.1        glue_1.4.0        
## [53] hms_0.5.3          yaml_2.2.1         colorspace_1.4-1   rvest_0.3.5       
## [57] haven_2.2.0
```