# Pre-processing of *Strongyloides ratti* bulk RNAseq via an alignment-free analysis pipeline

## Contents



# 1 Introduction

The goal of this file is to pre-process the *Strongyloides ratti* RNAseq dataset originally published by Hunt *et al* 2016.

# 2 Pre-processing Methods Overview

Raw reads are aligned to the *S. ratti* reference transcriptome (PRJEB125.WBPS14.mRNA_transcripts, downloaded from WormBase Parasite on 17 August 2020), using Kallisto. Kallisto alignments are imported into the R environment and annotated with information imported via the Wormbase ParaSite BioMaRT. Annotation information includes: *C. elegans* homologs/percent homology, UniProtKB number, Interpro terms, GO terms, and general Description information. Hunt *et al* 2016 establishes two distinct subclades from the four sequenced *Strongyloides* species: *S. venezuelensis-S. papillosus* and *S. ratti-S. stercoralis*. Thus, we also include annotation information for the appropriate in-group (here, *S. stercoralis*), and a reference member of the out-group (*S. papillosus*). Annotation information is saved as an R object that is passed to a Shiny Web App for downstream browsing and on-demand analysis. Note: Raw count data could be saved as a digitial gene expression list if desired (not currently done).

Raw reads were quantified as counts per million using the EdgeR package, then filtered to remove transcripts with low counts (less than 1 count-per-million in at least 1 sample). A list of discarded genes and their expression values across life stages is saved. Non-discarded gene values are normalized using the trimmed

mean of M-values method (TMM, Robinson and Oshlack ) to permit between-samples comparisons. The mean-variance relationship was modeled using a precision weights approach Law *et al* 2014. This dataset includes technical replicates; after voom modeling, data are condensed by replacing within-experiment technical replicates with their average, using the `limma:avearrays` function.

A variance-stabilized, condensed DGEList object is saved; this file is passed to a Shiny Web App for downstream browsing and on-demand analysis.

Note: samples included in this database were collected in two separate experiments, with FLM and iL3s in one, and FLF and PF in another. Due to the lack of sample overlap between the experiments, it is not possible to correct for batch effects.

# 3 Results/Analysis

## 3.1 Kallisto read mapping

This script checks the qualitiy of the fastq files and performs an alignment to the Strongyloides ratti cDNA transcriptome reference with Kallisto. To run this 'shell script' you will need to open your terminal and navigate to the directory where this script resides on your computer. For addditional instructions, see comments at head of script.

## 3.2 Import Kallisto reads into R

Import Kallisto transcript counts into R using Tximport. Counts are generated from abundance files using the `lengthScaledTPM` option. This code chunk is not evaluated each time, instead it was run once and an object containing the transcripts per million data is saved. In subsequent chunks, that file is loaded, and analysis progresses. The point of this is so that folks attempting to rerun this analysis do not need to have abundance files loaded on their local machines (and we do not have to upload abundance files to github).

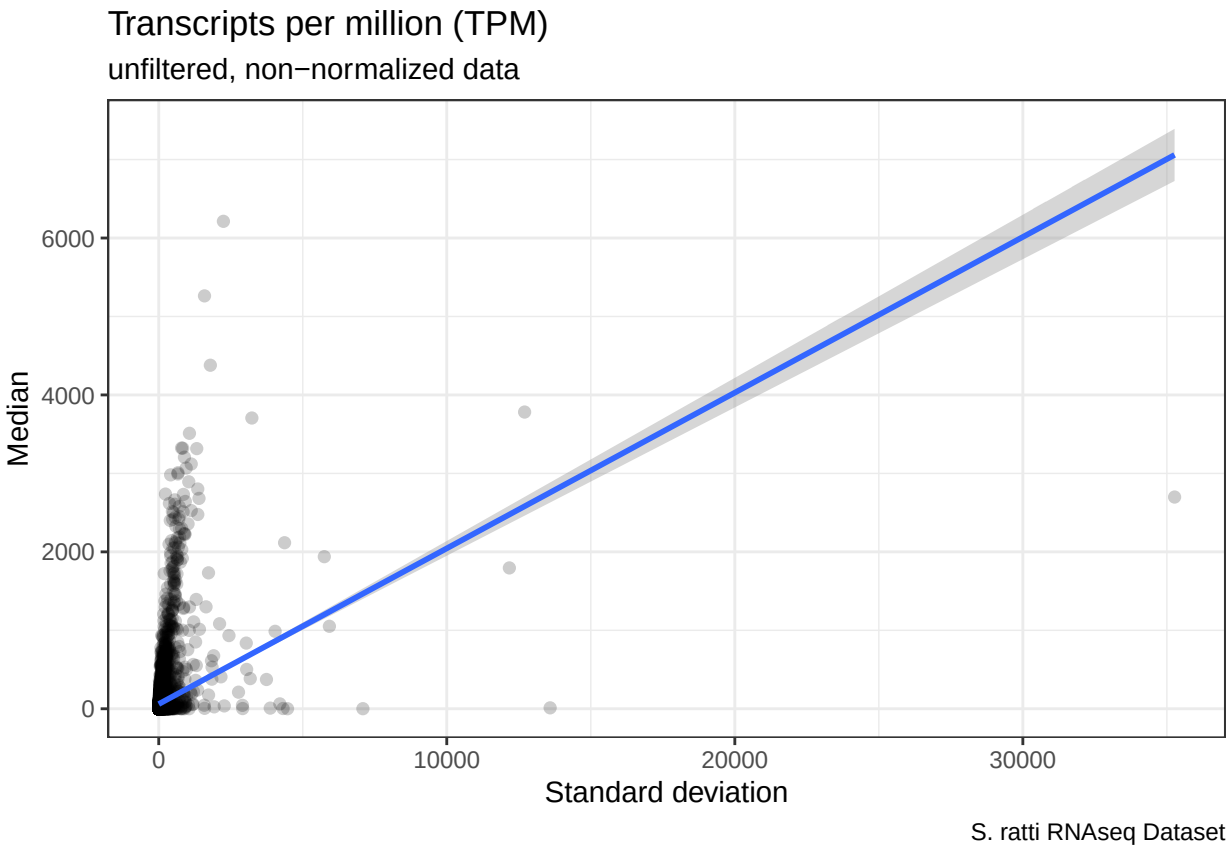
## 3.3 Gene Annotation

Import gene annotation information for *S. ratti* genes, including:

- *C. elegans* homologs/percent homology
- *S. stercoralis* homologs/percent homology
- *S. papillosus* homologs/percent homology
- UniProtKB number
- Interpro terms
- GO terms
- general Description information using biomart.

## 3.4 Generate Digital Gene Expression List

Next we generate a digital gene expression list that can be easily shared/loaded for downstream filtering/normalization. This code chunk generates a scatter plot of unfiltered and non-normalized transcripts per million data.

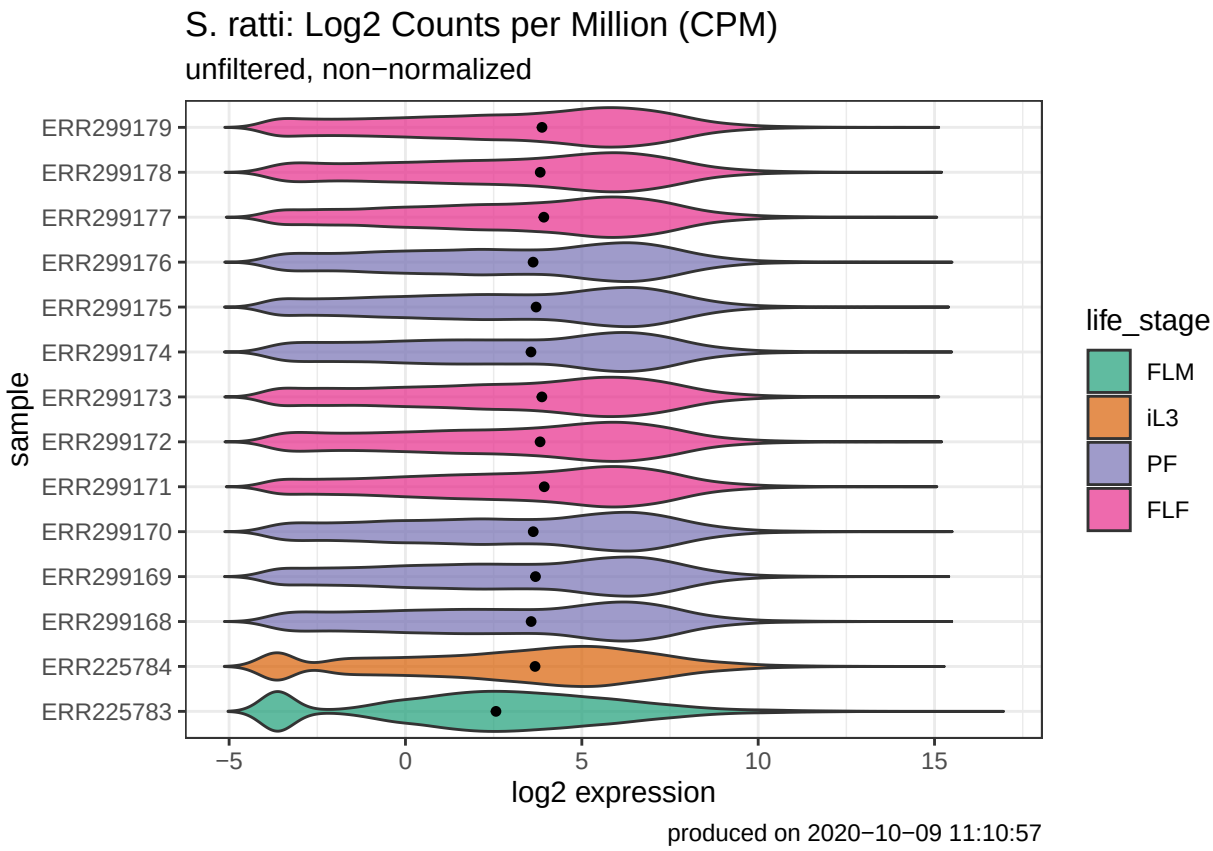



### 3.5 Data Filtering and Normalization

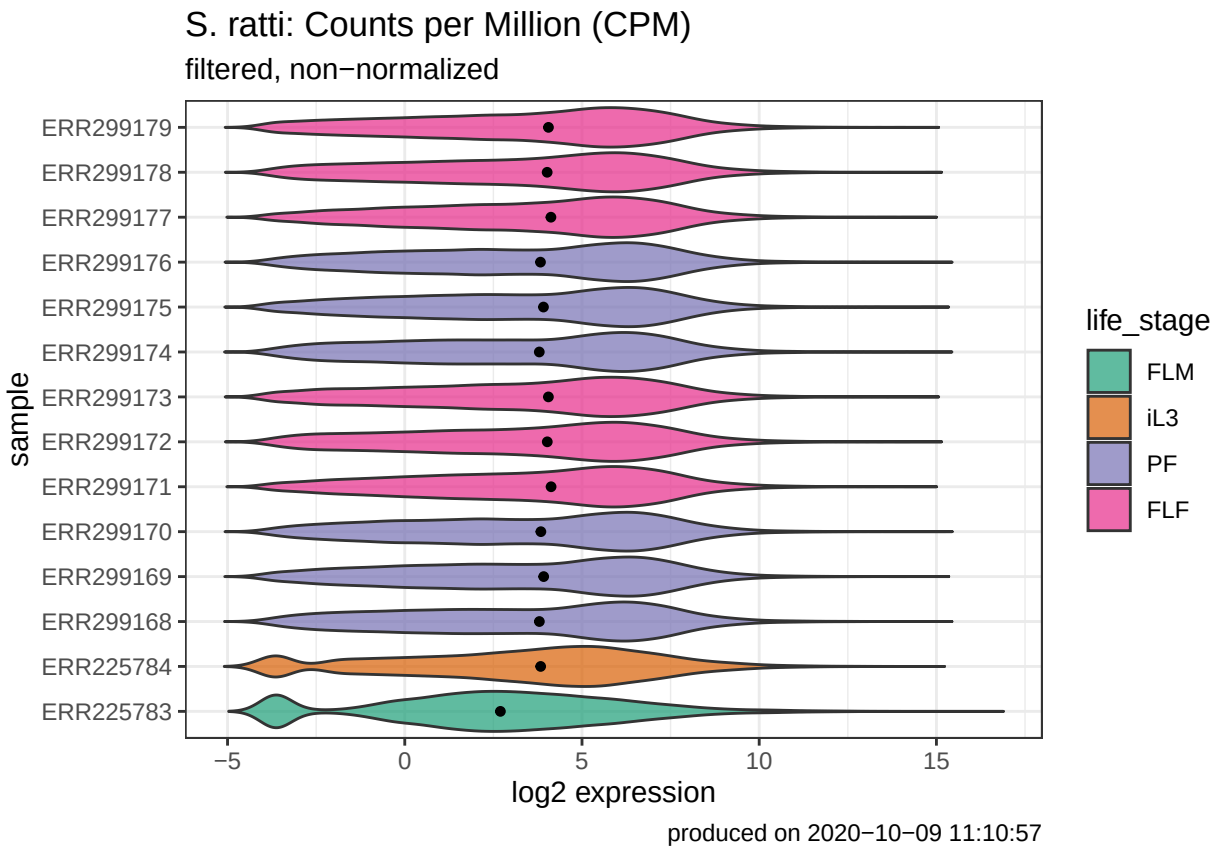
The goal of this chunk is to:

1. Filter and normalize data
2. Use `ggplot2` to visualize the impact of filtering and normalization on the data.

### 3.5.1 Plot of unfiltered, non-normalized log2CPM data by life stage

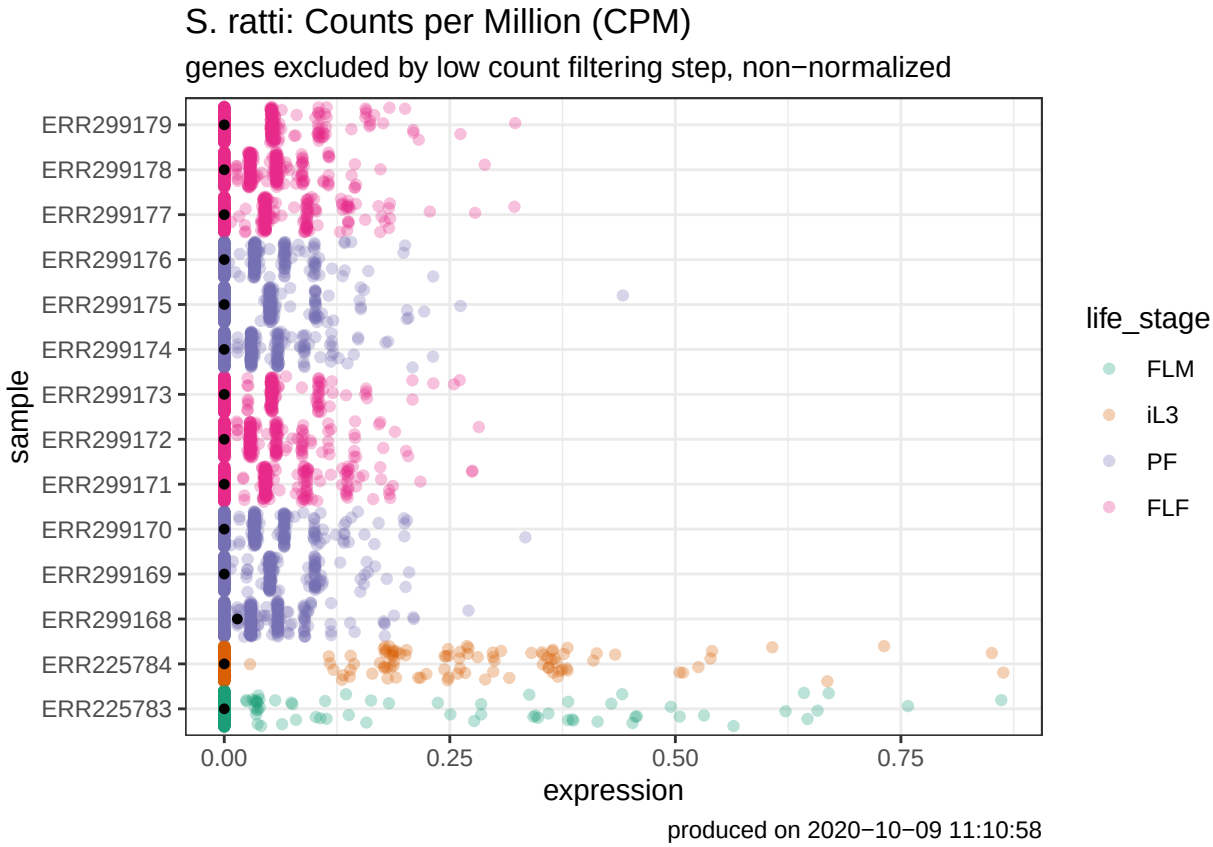


### 3.5.2 Plot of filtered, non-normalized log2CPM data by life stage

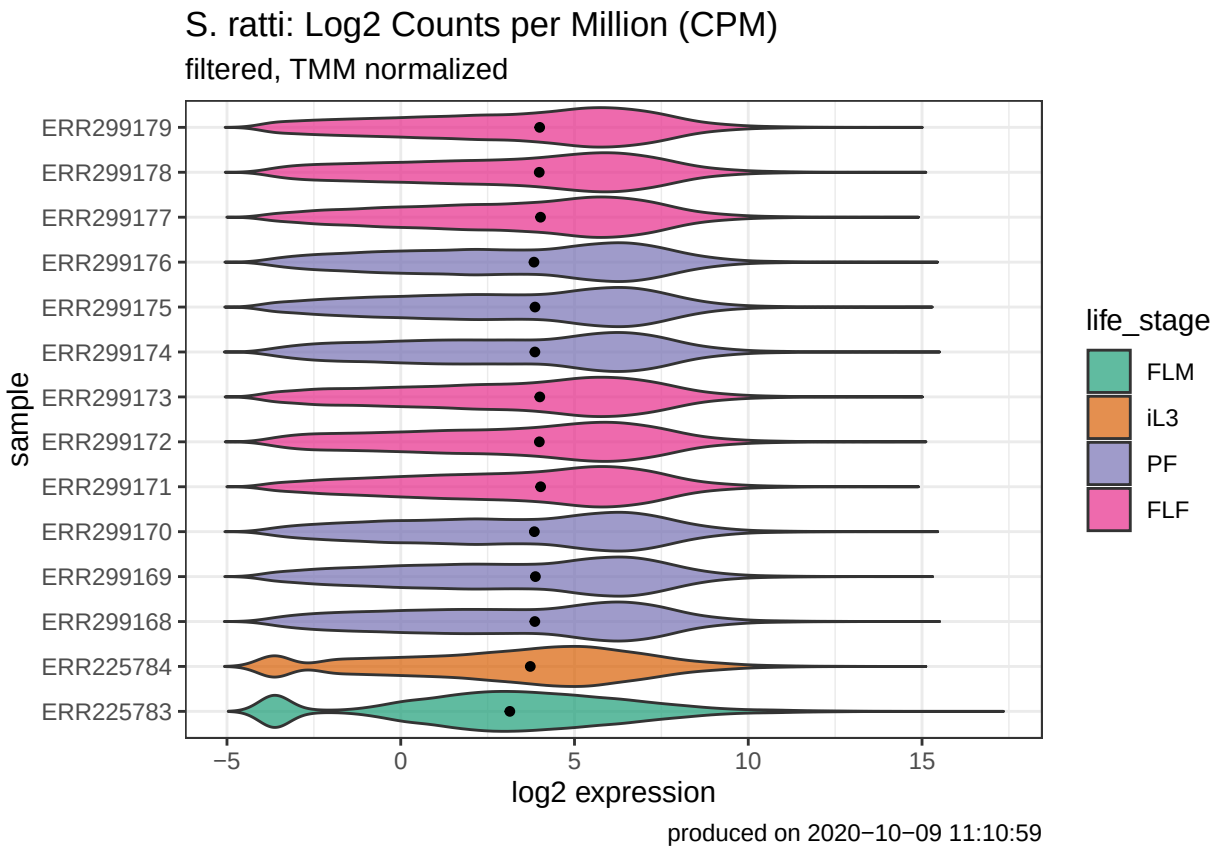



### 3.5.3 Plot of genes discarded by low-copy filtering step

The low copy number filtering step excluded a total of `dim(myDGEList.discarded)[[1]]` genes.

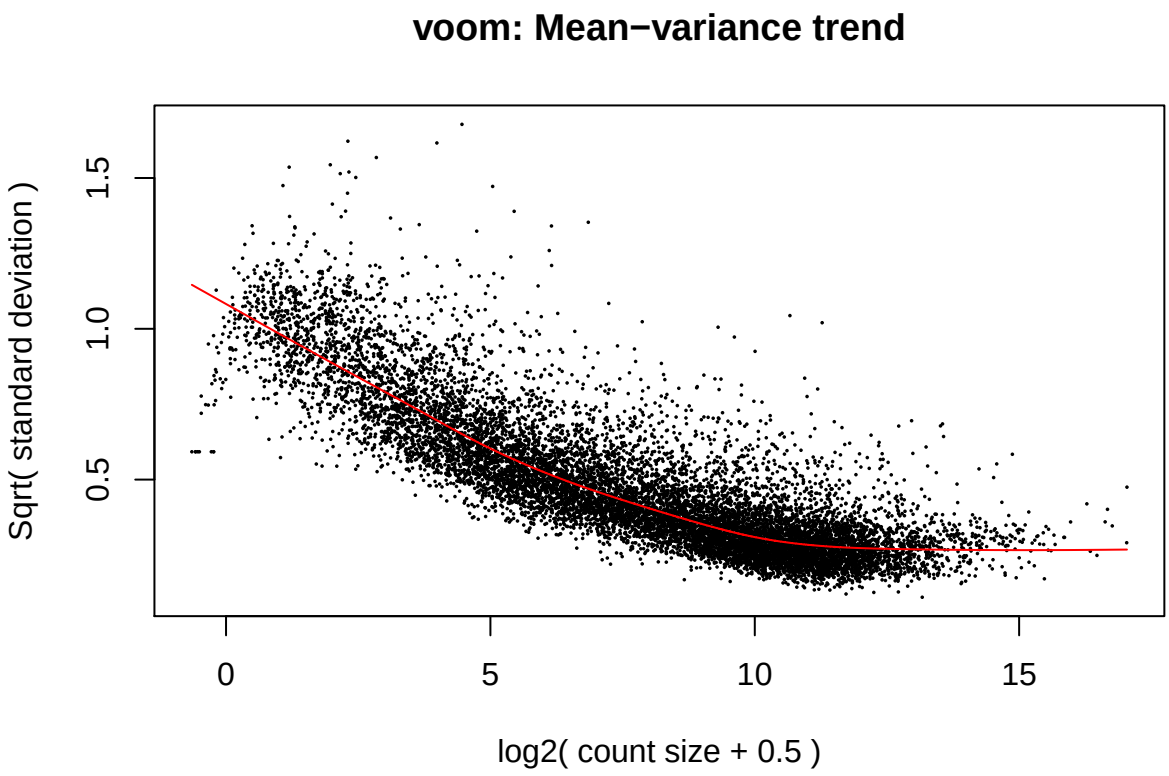

S. ratti: Counts per Million (CPM)
genes excluded by low count filtering step, non−normalized
ERR299179
ERR299178
ERR299177
ERR299176
ERR299175
ERR299174
ERR299173
ERR299172
ERR299171
ERR299170
ERR299169
ERR299168
ERR225784
ERR225783
sample
0.00
0.25
0.50
0.75
expression
life_stage
FLM
iL3
PF
FLF
produced on 2020−10−09 11:10:58

#### 3.5.4 Plot of filtered, normalized log2CPM data by life stage




### 3.6 Compute Variance-Stabilized DGEList Object

This chunk uses a DGEList of filtered and normalized abundance data. It will fit data to a linear model for responsively detecting differentially expressed genes (DEGs).




### 3.7 Save Data and Annotations

Finally, we save data and annotations generated in code chunks above. We can separate these saving actions into two groups:

1. Data saved for downstream offline analyses, including the `SrRNAseq_data_preprocessed` file which saves filtered, normalized (but not voom adjusted) log2CPM values, gene annotation information, and sample information.

2. Files that are required inputs to the *Strongyloides* RNAseq Browser App, including:

   i) a gene annotation R object (`Sr_geneAnnotations`)
   ii) the variance-stabilized vDGEList, saved as an R object (`Sr_vDGEList`)
   iii) a matrix of discarded genes and their raw counts (`SrRNAseq_discardedGene_counts.csv`) - this data is downloadable from within the Browser App
   iv) a matrix of variance-stabilized gene expression data, extracted from the vDGEList (`SrRNAseq_log2cpm_filtered_norm_voom.csv`) - this data is downloadable from within the Browser App

By default, this code chunk will not be evaluated. To save the files described above, change the chunk option from `eval = FALSE` to `eval = TRUE`. These files are saved in an Outputs folder; in order to make them accessible to a local version of the Shiny browser they need to be moved to appropriate subfolders within the App folder - the www sub folder (for .csv files) or the Data subfolder (for R objects). Stable copies are already located within those folders and do not need to be replaced unless the pre-processing steps change.

# 4 Appendix I: All code for this report

```
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE)

# This script checks the qualitiy of the fastq files and performs an alignment
# to the Strongyloides ratti cDNA transcriptome reference with Kallisto.
# To run this 'shell script' you will need to open your terminal and navigate
# to the directory where this script resides on your computer.
# This should be the same directory where you fastq files and
# reference fasta file are found.
# Change permissions on your computer so that you can run a shell script by
# typing: 'chmod u+x readMapping_sratti.sh' (without the quotes)
# at the terminal prompt
# Then type './readMapping_sratti.sh' (without the quotes) at the prompt.
# This will begin the process of running each line of code in the shell script.

# first use fastqc to check the quality of the fastq files:
fastqc *.gz -t 14

# build index from the reference fasta file
kallisto index -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index
strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.fa

# map reads to the indexed reference host transcriptome

# Parasitic Females: Biological Replicates A-C, Technical replicate set 1
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299168 -t 14 ERR299168_1.fastq.gz ERR299168_2.fastq.gz&> ERR299168.log
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299169 -t 14 ERR299169_1.fastq.gz ERR299169_2.fastq.gz&> ERR299169.log
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299170 -t 14 ERR299170_1.fastq.gz ERR299170_2.fastq.gz&> ERR299170.log

# Free-living Females: Biological Replicates A-C, Technical replicate set 1
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299171 -t 14 ERR299171_1.fastq.gz ERR299171_2.fastq.gz&> ERR299171.log
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299172 -t 14 ERR299172_1.fastq.gz ERR299172_2.fastq.gz&> ERR299172.log
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299173 -t 14 ERR299173_1.fastq.gz ERR299173_2.fastq.gz&> ERR299173.log

# Parasitic Females: Biological Replicates A-C, Technical replicate set 2
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299174 -t 14 ERR299174_1.fastq.gz ERR299174_2.fastq.gz&> ERR299174.log
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299175 -t 14 ERR299175_1.fastq.gz ERR299175_2.fastq.gz&> ERR299175.log
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299176 -t 14 ERR299176_1.fastq.gz ERR299176_2.fastq.gz&> ERR299176.log

# Free-living Females: Biological Replicates A-C, Technical replicate set 2
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299177 -t 14 ERR299177_1.fastq.gz ERR299177_2.fastq.gz&> ERR299177.log
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
```

```
ERR299178 -t 14 ERR299178_1.fastq.gz ERR299178_2.fastq.gz&> ERR299178.log
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR299179 -t 14 ERR299179_1.fastq.gz ERR299179_2.fastq.gz&> ERR299179.log

# Free-living Males
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR225783 -t 14 ERR225783_1.fastq.gz ERR225783_2.fastq.gz&> ERR225783.log

# iL3s
kallisto quant -i strongyloides_ratti.PRJEB125.WBPS14.mRNA_transcripts.index -o
ERR225784 -t 14 ERR225784_1.fastq.gz ERR225784_2.fastq.gz&> ERR225784.log

# summarize fastqc and kallisto mapping results using MultiQC
multiqc -d .

echo "Finished"
# load packages ----
suppressPackageStartupMessages({
  library(tidyverse)
  library(tximport)
  library(ensembldb)
  library(biomaRt)
  library(magrittr)
})
# read in the study design ----
targets <- read_tsv("../Data/S_ratti/Study_Design/SrattiRNAseq_study_design.txt",
                    na = c("", "NA", "na"))
# create file paths to the abundance files generated by Kallisto
# using the 'file.path' function
path <- file.path("../Data/S_ratti/Reads",targets$sample, "abundance.tsv")
# check to make sure this path is correct by seeing if the files exist
#all(file.exists(path))

# get annotations using organism-specific package ----
Tx.Sr <- getBM(attributes=c('wbps_transcript_id',
                            'wbps_gene_id'),
               # grab the ensembl annotations for Wormbase Parasite genes
               mart = useMart(biomart="parasite_mart",
                              dataset = "wbps_gene",
                              host="https://parasite.wormbase.org",
                              port = 443),
               filters = c('species_id_1010'),
               value = list('strattprjeb125')) %>%
  as_tibble() %>%
  #we need to rename the columns retreived from biomart
  dplyr::rename(target_id = wbps_transcript_id,
                WB_geneID = wbps_gene_id) %>%
  dplyr::mutate(gene_name = str_remove_all(target_id, "\\.[0-9]$")) %>%
  dplyr::mutate(gene_name = str_remove_all(gene_name, "[a-c]$")) %>%
  dplyr::select(!WB_geneID)


# import Kallisto transcript counts into R using Tximport ----
```

```r
# copy the abundance files to the working directory and rename
# so that each sample has a unique name
Txi_gene <- tximport(path,
                     type = "kallisto",
                     tx2gene = Tx.Sr[,1:2],
                     txOut = FALSE,
                     countsFromAbundance = "lengthScaledTPM",
                     ignoreTxVersion = FALSE)

# Save the raw transcript counts ----
save(Txi_gene,
     file = file.path("../Data/S_ratti",
                      "SrRNAseq_TPM"))

# Introduction to this chunk -----------
# This chunk imports gene annotation information for S. ratti genes, including:
# C. elegans homologs/percent homology,
# S. stercoralis homologs/percent homology,
# S. papillosus homologs/percent homology,
# UniProtKB number, Interpro terms, GO terms,
# and general Description information using biomart.
# It will generate a table that can be saved.

# Load packages ------
suppressPackageStartupMessages({
  library(tidyverse)
  library(tximport)
  library(ensembldb)
  library(biomaRt)
  library(magrittr)
})

# Load data & study design ----
load(file = file.path("../Data/S_ratti",
                      "SrRNAseq_TPM"))

targets <- read_tsv("../Data/S_ratti/Study_Design/SrattiRNAseq_study_design.txt",
                    na = c("", "NA", "na"))

# Get In-subclade group homologs for S. ratti
# genes from BioMart and filter -----
Annt.temp.1 <- getBM(attributes=c('wbps_transcript_id',
                                  'ststerprjeb528_gene',
                                  'ststerprjeb528_homolog_perc_id'),
                     # grab the ensembl annotations for Wormbase Parasite genes
                     mart = useMart(biomart="parasite_mart",
                                    dataset = "wbps_gene",
                                    host="https://parasite.wormbase.org",
                                    port = 443),
                     filters = c('species_id_1010'),
                     value = list('strattprjeb125')) %>%
  as_tibble() %>%
  #rename columns
```

```
  dplyr::rename(geneID = wbps_transcript_id,
                In.subclade_geneID = ststerprjeb528_gene,
                In.subclade_percent_homology= ststerprjeb528_homolog_perc_id
  ) %>%
  dplyr::group_by(geneID)

# Get Out-subclade group homologs for S. ratti
# genes from BioMart and filter -----
Annt.temp.2 <- getBM(attributes=c('wbps_transcript_id',
                                  'stpapiprjeb525_gene',
                                  'stpapiprjeb525_homolog_perc_id'
                                  ),
                     # grab the ensembl annotations for Wormbase Parasite genes
                     mart = useMart(biomart="parasite_mart",
                                    dataset = "wbps_gene",
                                    host="https://parasite.wormbase.org",
                                    port = 443),
                     filters = c('species_id_1010'),
                     value = list('strattprjeb125')) %>%
  as_tibble() %>%
  #rename columns
  dplyr::rename(geneID = wbps_transcript_id,
                Out.subclade_geneID = stpapiprjeb525_gene,
                Out.subclade_percent_homology= stpapiprjeb525_homolog_perc_id
  ) %>%
  dplyr::group_by(geneID)

# Get C. elegans homologs and gene information for S. ratti
# genes from BioMart and filter -----
Annt.temp.3 <- getBM(attributes=c('wbps_transcript_id',
                                  'wbps_gene_id',
                                  'caelegprjna13758_gene_name',
                                  'caelegprjna13758_homolog_perc_id',
                                  'description',
                                  'interpro_short_description',
                                  'go_name_1006',
                                  'uniprot_sptrembl'),
                     # grab the ensembl annotations for Wormbase Parasite genes
                     mart = useMart(biomart="parasite_mart",
                                    dataset = "wbps_gene",
                                    host="https://parasite.wormbase.org",
                                    port = 443),
                     filters = c('species_id_1010',
                                 'wbps_transcript_id'),
                     value = list('strattprjeb125',
                                  rownames(Txi_gene$counts))) %>%
  as_tibble() %>%
  #rename columns
  dplyr::rename(geneID = wbps_transcript_id,
                WBgeneID = wbps_gene_id,
                Ce_geneID = caelegprjna13758_gene_name,
                Ce_percent_homology = caelegprjna13758_homolog_perc_id,
                Description = description,
```

```
                GO_term = go_name_1006,
                UniProtKB = uniprot_sptrembl
  ) %>%
  dplyr::group_by(geneID)

Annt.import <- full_join(Annt.temp.1, Annt.temp.2, by = "geneID") %>%
  full_join(Annt.temp.3, by = "geneID")


Annt.import$geneID <- str_remove_all(Annt.import$geneID, "\\.[0-9]$")
Annt.import$geneID <- str_remove_all(Annt.import$geneID, "[a-z]$")

# Replace empty string values (mostly in Ce_geneID column) with NAs
Annt.import[Annt.import == ""]<-NA

# Remove any duplications in the possible homolog matches.
# Select based on highest % homology.
# #Give fake value here to make sure genes
#without a homologs aren't filtered out
Annt.import$Ce_percent_homology[
  is.na(Annt.import$Ce_percent_homology)] <- 1000
Annt.import$In.subclade_percent_homology[
  is.na(Annt.import$In.subclade_percent_homology)] <- 1000
Annt.import$Out.subclade_percent_homology[
  is.na(Annt.import$Out.subclade_percent_homology)] <- 1000

Annt.logs <-Annt.import %>%
  dplyr::select(!c(interpro_short_description:GO_term))%>%
  group_by(geneID) %>%
  slice_max(n = 1, order_by = Ce_percent_homology,
            with_ties = FALSE) %>%
  slice_max(n = 1, order_by = In.subclade_percent_homology,
            with_ties = FALSE) %>%
  slice_max(n = 1, order_by = Out.subclade_percent_homology,
            with_ties = FALSE) %>%
  group_by(geneID, Ce_geneID)

# Remove source code to shorten the description
Annt.logs$Description<- Annt.logs$Description %>%
  str_replace_all(string = .,
                  pattern = "  \\[Source:.*\\]",
                  replacement = "") %>%
  cbind()

Annt.logs$Ce_percent_homology[
  Annt.logs$Ce_percent_homology == 1000] <- NA
Annt.logs$In.subclade_percent_homology[
  Annt.logs$In.subclade_percent_homology == 1000]<- NA
Annt.logs$Out.subclade_percent_homology[
  Annt.logs$Out.subclade_percent_homology == 1000]<- NA

# Clean up interprotKB terms, removing duplications and collapsing to one line
Annt.interpro<-Annt.import %>%
```

```
  dplyr::select(geneID, Ce_geneID, interpro_short_description) %>%
  group_by(geneID, Ce_geneID) %>%
  dplyr::distinct(interpro_short_description, .keep_all = TRUE) %>%
  dplyr::summarise(InterPro = paste(interpro_short_description,
                                    collapse = ', '))
# Clean up GO terms, removing duplications and collapsing to one line
Annt.goterms<-Annt.import %>%
  dplyr::select(geneID, Ce_geneID, GO_term) %>%
  group_by(geneID, Ce_geneID) %>%
  dplyr::distinct(GO_term, .keep_all = TRUE) %>%
  dplyr::summarise(GO_term = paste(GO_term, collapse = ', '))

annotations<-dplyr::left_join(Annt.logs, Annt.interpro) %>%
  dplyr::left_join(.,Annt.goterms) %>%
  ungroup() %>%
  dplyr::relocate(WBgeneID,In.subclade_geneID,
                  In.subclade_percent_homology,
                  Out.subclade_geneID,
                  Out.subclade_percent_homology,
                  .after = geneID) %>%
  column_to_rownames(var = "geneID")

# List of S. ratti genes is longer than the number of genes
# in the RNAseq dataset. So subset the annotations by the geneIDs in Txi_gene
annotations<-annotations[rownames(annotations) %in% rownames(Txi_gene$counts),]


# Goals of this chunk:
# Generate and save a digitial gene expression list
# that could be easily shared/loaded for downstream filtering/normalization

# Load packages -----
suppressPackageStartupMessages({
  library(tidyverse)
  library(edgeR)
  library(matrixStats)
  library(cowplot)
  library(ggthemes)
  library(RColorBrewer)
  library(gprofiler2)
})

# Generate and plot summary stats for the data ----
myTPM.stats <- transform(Txi_gene$abundance,
                         SD=rowSds(Txi_gene$abundance),
                         AVG=rowMeans(Txi_gene$abundance),
                         MED=rowMedians(Txi_gene$abundance))

# produce a scatter plot of the transformed data
p1<-ggplot(myTPM.stats) +
  aes(x = SD, y = MED) +
  geom_point(shape=16, size=2, alpha = 0.2) +
  geom_smooth(method=lm) +
```

```
  #geom_hex(show.legend = FALSE) +
  labs(y="Median", x = "Standard deviation",
       title="Transcripts per million (TPM)",
       subtitle="unfiltered, non-normalized data",
       caption="S. ratti RNAseq Dataset") +
  theme_bw()
p1

# make a Digital Gene Expression list using the raw counts and plot ----
myDGEList <- DGEList(Txi_gene$counts,
                     samples = data.frame(samples = targets$source,
                                          source = targets$sample),
                     group = targets$group,
                     genes = annotations)

# Goals of this chunk:
# 1 - Filter and normalize data
# 2 - use ggplot2 to visualize the impact of filtering and
# normalization on the data.

# Notes:
# recall that abundance data are TPM, while the counts are
# read counts mapping to each gene or transcript

# Load packages -----
suppressPackageStartupMessages({
  library(tidyverse)
  library(edgeR)
  library(matrixStats)
  library(cowplot)
  library(ggthemes)
  library(RColorBrewer)
  library(gprofiler2)
})

# calculate and plot log2 counts per million ----

# Generate life stage IDs
ids <- rep(cbind(targets$group),
           times = nrow(myDGEList$counts)) %>%
  as_factor()

# use the 'cpm' function from EdgeR to get log2 counts per million
# then coerce into a tibble
# add sample names to the dataframe
# tidy up the dataframe into a tibble
log2.cpm.df.pivot <-cpm(myDGEList, log=TRUE) %>%
  as_tibble(rownames = "geneID") %>%
  setNames(nm = c("geneID", targets$sample)) %>%
  pivot_longer(cols = -geneID,
               names_to = "samples",
               values_to = "expression") %>%
  add_column(life_stage = ids)
```

```
# plot the pivoted data
p2 <- ggplot(log2.cpm.df.pivot) +
  aes(x=samples, y=expression, fill=life_stage) +
  geom_violin(trim = FALSE, show.legend = T, alpha= 0.7) +
  stat_summary(fun = "median",
               geom = "point",
               shape = 20,
               size = 2,
               color = "black",
               show.legend = FALSE) +
  labs(y="log2 expression", x = "sample",
       title="S. ratti: Log2 Counts per Million (CPM)",
       subtitle="unfiltered, non-normalized",
       caption=paste0("produced on ", Sys.time())) +
  theme_bw() +
  scale_fill_brewer(palette = "Dark2") +
  coord_flip()

p2

# Filter the data ----
# filter genes/transcripts with low counts
# how many genes had more than 1 CPM (TRUE) in at least n samples
# Note: The cutoff "n" is adjusted for the number of
# samples in the smallest group of comparison.
keepers <- cpm(myDGEList) %>%
  rowSums(.>1)>=1

myDGEList.filtered <- myDGEList[keepers,]

ids.filtered <- rep(cbind(targets$group),
                    times = nrow(myDGEList.filtered)) %>%
  as_factor()

log2.cpm.filtered.df.pivot <- cpm(myDGEList.filtered, log=TRUE) %>%
  as_tibble(rownames = "geneID") %>%
  setNames(nm = c("geneID", targets$sample)) %>%
  pivot_longer(cols = -geneID,
               names_to = "samples",
               values_to = "expression") %>%
  add_column(life_stage = ids.filtered)

p3 <- ggplot(log2.cpm.filtered.df.pivot) +
  aes(x=samples, y=expression, fill=life_stage) +
  geom_violin(trim = FALSE, show.legend = T, alpha= 0.7) +
  stat_summary(fun = "median",
               geom = "point",
               shape = 20,
               size = 2,
               color = "black",
               show.legend = FALSE) +
  labs(y="log2 expression", x = "sample",
```

```
       title="S. ratti: Counts per Million (CPM)",
       subtitle="filtered, non-normalized",
       caption=paste0("produced on ", Sys.time())) +
  theme_bw() +
  scale_fill_brewer(palette = "Dark2") +
  coord_flip()
p3

# Look at the genes excluded by the filtering step ----
# just to check that there aren't any with high expression that are in few samples
# Discarded genes
myDGEList.discarded <- myDGEList[!keepers,]

ids.discarded <- rep(cbind(targets$group),
                     times = nrow(myDGEList.discarded)) %>%
    as_factor()

log2.cpm.discarded.df.pivot <- cpm(myDGEList.discarded, log=F) %>%
    as_tibble(rownames = "geneID") %>%
    setNames(nm = c("geneID", targets$sample)) %>%
    pivot_longer(cols = -geneID,
                 names_to = "samples",
                 values_to = "expression") %>%
    add_column(life_stage = ids.discarded)


p.discarded <- ggplot(log2.cpm.discarded.df.pivot) +
    aes(x=samples, y=expression, color=life_stage) +
    #geom_violin(trim = FALSE, show.legend = T, alpha= 0.7) +
    geom_jitter(alpha = 0.3, show.legend = T)+
    stat_summary(fun = "median",
                 geom = "point",
                 shape = 20,
                 size = 2,
                 color = "black",
                 show.legend = FALSE) +
    labs(y="expression", x = "sample",
         title="S. ratti: Counts per Million (CPM)",
         subtitle="genes excluded by low count filtering step, non-normalized",
         caption=paste0("produced on ", Sys.time())) +
    theme_bw() +
    scale_color_brewer(palette = "Dark2") +
    coord_flip()
p.discarded

# # Carry out GO enrichment of discarded gene set using gProfiler2 ----
# discarded.geneID <- unique(log2.cpm.discarded.df.pivot$geneID)
# gost.res <- gost(list(Discarded_genes = discarded.geneID),
#                  organism = "strattprjeb125", correction_method = "fdr")
# gostplot(gost.res, interactive = T, capped = T)

# Generate a matrix of discarded genes and their raw counts ----
discarded.gene.df <- log2.cpm.discarded.df.pivot %>%
```

```
  pivot_wider(names_from = c(life_stage, samples),
              names_sep = "-",
              values_from = expression,
              id_cols = geneID)

# Normalize the data using a between samples normalization ----
# Source for TMM sample normalization here:
# https://genomebiology.biomedcentral.com/articles/10.1186/gb-2010-11-3-r25
myDGEList.filtered.norm <- calcNormFactors(myDGEList.filtered, method = "TMM")

log2.cpm.filtered.norm <- cpm(myDGEList.filtered.norm, log=TRUE)

log2.cpm.filtered.norm.df<- cpm(myDGEList.filtered.norm, log=TRUE) %>%
  as_tibble(rownames = "geneID") %>%
  setNames(nm = c("geneID", targets$sample))

log2.cpm.filtered.norm.df.pivot<-log2.cpm.filtered.norm.df %>%
  pivot_longer(cols = -geneID,
               names_to = "samples",
               values_to = "expression") %>%
  add_column(life_stage = ids.filtered)

p4 <- ggplot(log2.cpm.filtered.norm.df.pivot) +
  aes(x=samples, y=expression, fill=life_stage) +
  geom_violin(trim = FALSE, show.legend = T, alpha = 0.7) +
  stat_summary(fun = "median",
               geom = "point",
               shape = 20,
               size = 2,
               color = "black",
               show.legend = FALSE) +
  labs(y="log2 expression", x = "sample",
       title="S. ratti: Log2 Counts per Million (CPM)",
       subtitle="filtered, TMM normalized",
       caption=paste0("produced on ", Sys.time())) +
  theme_bw() +
  scale_fill_brewer(palette = "Dark2") +
  coord_flip()
p4

# Introduction to this chunk ----
# This chunk uses a DGEList of filtered and normalized abundance data
# It will fit data to a linear model for responsively detecting
# differentially expressed genes (DEGs)

# Load packages ----
suppressPackageStartupMessages({
  library(tidyverse)
  library(limma) # differential gene expression using linear modeling
  library(edgeR)
})

# Set up the design matrix ----
```

```r
group <- factor(targets$group)
biolrep <- factor(targets$source)
#block <- factor(targets$block)
# ideally, I would use a blocking design to handle possibility of batch effects
# between "PRJEB1376" and "PRJEB3187", comparisons across group.
# However, because there aren't any overlaping samples, between the two groups,
# I don't think I can use the (~block + group) setup for the model matrix.
# At this point, the batch is likely a confounding variable.
design <- model.matrix(~0 + group)
colnames(design) <- levels(group)

# Model mean-variance trend and fit linear model to data ----
colnames(myDGEList.filtered.norm$counts) <- targets$group

v.DEGList.filtered.norm.withtechreplicates <- voom(counts = myDGEList.filtered.norm,
                                design = design,
                                plot = T)
colnames(v.DEGList.filtered.norm.withtechreplicates$E) <- targets$group

# Condense data by replacing within-experiment technical
# replicates with their average
v.DEGList.filtered.norm <- avearrays(v.DEGList.filtered.norm.withtechreplicates,
                                     biolrep)
colnames(v.DEGList.filtered.norm$E) <- paste(v.DEGList.filtered.norm$targets$group,
                                             levels(biolrep),sep = '-')


# Check for presence of output folder, generate if it doesn't exist
output.path <- "../Outputs"
if (!dir.exists(output.path)){
  dir.create(output.path)
}

# Save full gene annotations ----
# This object is required for the Shiny Browser
save(annotations,
     file = file.path(output.path,
                      "Sr_geneAnnotations"))


# Save a matrix of discarded genes and their raw counts ----
discarded.gene.df %>%
write.csv(file = file.path(output.path,
                      "SrRNAseq_discardedGene_counts.csv"))

# Save matrix of genes and their filtered, normalized, voom-transformed counts ----
# This is the count data that underlies the differential expression analyses in the Shiny app.
# Saving it here so that users of the app can access the input information.
write.csv(v.DEGList.filtered.norm$E,
          file = file.path(output.path,
                           "SrRNAseq_log2cpm_filtered_norm_voom.csv"))

# Save v.DEGList ----
```

```
# This file will can be imported into Shiny App

save(v.DEGList.filtered.norm,
     file = file.path(output.path,
                      "Sr_vDGEList"))

# This data is required for downstream analyses in this file.
# It enables users to not have to re-import and
# re-align raw read files every time the code is run.
#
SrRNAseq.preprocessed.data <- list(targets = targets,
                                   annotations = annotations,
                                   log2.cpm.filtered.norm = log2.cpm.filtered.norm,
                                   myDGEList.filtered.norm = myDGEList.filtered.norm
)
save(SrRNAseq.preprocessed.data,
     file = file.path(output.path,
                      "SrRNAseq_data_preprocessed"))


sessionInfo()
```

# 5 Appendix II: Session Info

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Catalina 10.15.5
##
## 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] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] gprofiler2_0.1.9        RColorBrewer_1.1-2      ggthemes_4.2.0
##  [4] cowplot_1.0.0           matrixStats_0.56.0      edgeR_3.28.1
##  [7] limma_3.42.2            magrittr_1.5            biomaRt_2.42.1
## [10] ensembldb_2.10.2        AnnotationFilter_1.10.0 GenomicFeatures_1.38.2
## [13] AnnotationDbi_1.48.0    Biobase_2.46.0          GenomicRanges_1.38.0
## [16] GenomeInfoDb_1.22.1     IRanges_2.20.2          S4Vectors_0.24.4
## [19] BiocGenerics_0.32.0     tximport_1.14.2         forcats_0.5.0
## [22] stringr_1.4.0           dplyr_1.0.1             purrr_0.3.4
## [25] readr_1.3.1             tidyr_1.1.1             tibble_3.0.3
## [28] ggplot2_3.3.2           tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] colorspace_1.4-1            ellipsis_0.3.1
```

```
##  [3] XVector_0.26.0              fs_1.4.2
##  [5] rstudioapi_0.11             farver_2.0.3
##  [7] bit64_0.9-7                 fansi_0.4.1
##  [9] lubridate_1.7.9             xml2_1.3.2
## [11] splines_3.6.3               knitr_1.29
## [13] jsonlite_1.7.0              Rsamtools_2.2.3
## [15] broom_0.5.6                 dbplyr_1.4.4
## [17] compiler_3.6.3              httr_1.4.2
## [19] backports_1.1.8             assertthat_0.2.1
## [21] Matrix_1.2-18               lazyeval_0.2.2
## [23] cli_2.0.2                   htmltools_0.5.0
## [25] prettyunits_1.1.1           tools_3.6.3
## [27] gtable_0.3.0                glue_1.4.1
## [29] GenomeInfoDbData_1.2.2      rappdirs_0.3.1
## [31] Rcpp_1.0.5                  cellranger_1.1.0
## [33] vctrs_0.3.2                 Biostrings_2.54.0
## [35] nlme_3.1-148                rtracklayer_1.46.0
## [37] xfun_0.15                   rvest_0.3.5
## [39] lifecycle_0.2.0             XML_3.99-0.3
## [41] zlibbioc_1.32.0             scales_1.1.1
## [43] hms_0.5.3                   ProtGenerics_1.18.0
## [45] SummarizedExperiment_1.16.1 yaml_2.2.1
## [47] curl_4.3                    memoise_1.1.0
## [49] stringi_1.4.6               RSQLite_2.2.0
## [51] BiocParallel_1.20.1         rlang_0.4.7
## [53] pkgconfig_2.0.3             bitops_1.0-6
## [55] evaluate_0.14               lattice_0.20-41
## [57] labeling_0.3                GenomicAlignments_1.22.1
## [59] htmlwidgets_1.5.1.9001      bit_1.1-15.2
## [61] tidyselect_1.1.0            R6_2.4.1
## [63] generics_0.0.2              DelayedArray_0.12.3
## [65] DBI_1.1.0                   mgcv_1.8-31
## [67] pillar_1.4.6                haven_2.3.1
## [69] withr_2.2.0                 RCurl_1.98-1.2
## [71] modelr_0.1.8                crayon_1.3.4
## [73] BiocFileCache_1.10.2        plotly_4.9.2.9000
## [75] rmarkdown_2.3               progress_1.2.2
## [77] locfit_1.5-9.4              grid_3.6.3
## [79] readxl_1.3.1                data.table_1.12.8
## [81] blob_1.2.1                  reprex_0.3.0
## [83] digest_0.6.25               openssl_1.4.2
## [85] munsell_0.5.0               viridisLite_0.3.0
## [87] askpass_1.1
```