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

Contents

1 Introduction **1**

2 Pre-processing Methods Overview **2**

3 Results/Analysis **2**
- 3.1 Libraries . . . 2
- 3.2 Kallisto read mapping . . . 2
- 3.3 Import Kallisto reads into R . . . 2
- 3.4 Gene Annotation . . . 3
- 3.5 Generate Digital Gene Expression List . . . 3
- 3.6 Data Filtering and Normalization . . . 3
  - 3.6.1 Plot of unfiltered, non-normalized log2CPM data by life stage . . . 4
  - 3.6.2 Plot of filtered, non-normalized log2CPM data by life stage . . . 5
  - 3.6.3 Plot of genes discarded by low-copy filtering step . . . 5
  - 3.6.4 Plot of filtered, normalized log2CPM data by life stage . . . 7
- 3.7 Compute Variance-Stabilized DGEList Object . . . 7
- 3.8 Save Data and Annotations . . . 8

4 Appendix I: Replicate above chunks for Group iL3_extended **9**
- 4.1 Import Kallisto Reads . . . 9
- 4.2 Data Filtering and Normalization. . . . 10
  - 4.2.1 Plot of unfiltered, non-normalized log2CPM data by life stage . . . 10
  - 4.2.2 Plot of filtered, non-normalized log2CPM data by life stage . . . 11
  - 4.2.3 Plot of genes discarded by low-copy filtering step . . . 11
  - 4.2.4 Plot of filtered, normalized log2CPM data by life stage . . . 14
- 4.3 Save Data . . . 15

5 Appendix II: Process all life stages together **15**
- 5.1 Import Kallisto Reads . . . 15
- 5.2 Data Filtering and Normalization . . . 15
- 5.3 Save Data and Annotations . . . 15

6 Appendix III: All code for this report **16**

7 Appendix IV: Session Info **36**

1 Introduction

The goal of this file is to pre-process the *Strongyloides venezuelensis* RNAseq dataset originally published by Hunt *et al* 2018.

2 Pre-processing Methods Overview

Raw reads are aligned to the *S. venezuelensis* reference transcriptome (PRJEB530.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. papillosus*), and a reference member of the out-group (*S. stercoralis*). Annotation information is saved as an R object that is passed to a Shiny Web App for downstream browsing and on-demand analysis. Raw count data is saved as an R object.

Samples included in study PRJDB3457 were prepared using different libary construction methods (amplified vs non-amplified), sequencing run batches, and machines (Hunt *et al* 2018). Dividing the experiments based on sequencing instrument produces two batches, both of which contain data from Free-living females and thereotically permit batch correction. However, following limma-based batch correction there were still substantial differences between FLF groups from the two batches. We therefore take the conservative approach of treating these two batches separately.
Thus, we define two functional groups for processing and analysis:
1. Group FLF_PF: Free-living females and parasitic females (samples DRR106346 - DRR106357; aka SAMD00096905-SAMD00096910). This set includes 3 biological replicates and two technical replicates per life stage.
2. Group iL3_extended: Egg, L1, iL3s, activated iL3s (1 and 5 day), iL3_lung, Young_FLF, FLF (samples DRR029282, DRR029433 - DRR029445). This set includes 1 biological replicated and two technical replicates per life stage, except for activated iL3s, which have a single technical replicate at 1 and 5 days.

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 3 samples (Group FLF_PF) or 1 sample (Group iL3_extended)). 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 for each analysis group; this file is passed to a Shiny Web App for downstream browsing and on-demand analysis.

3 Results/Analysis

Note: Code chunks are collated and echoed at the end of the document in Appendix III.

3.1 Libraries

3.2 Kallisto read mapping

This shell script checks the qualitiy of the fastq files and performs an alignment to the *Strongyloides venezuelensis* cDNA transcriptome reference with Kallisto; to work, it needs to be saved as an .sh file in a folder containing required raw files.This script is reproduced here to demonstrate the QC and alignment process.

3.3 Import Kallisto reads into R

Imports study design file and Kallisto reads into the R environment. At this stage, we separate the two experimental groups, such that differences in library size and transcript length between the two groups are not corrected when calculating counts from abudnance (we use the lengthScaledTPM option in `tximport`).

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).

3.4 Gene Annotation

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

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

3.5 Generate Digital Gene Expression List

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

3.6 Data Filtering and Normalization

Goals of this chunk:

1. Filter and normalize data

2. Use ggplot2 to visualize the impact of filtering and normalization on the data. In this chunk the two experimental groups are separated.

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

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

3.6.3 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. venezuelensis: Counts per Million (CPM)
genes excluded by low count filtering step, non−normalized, Group FLF_PF
DRR106357
DRR106356
DRR106355
DRR106354
DRR106353
DRR106352
DRR106351
DRR106350
DRR106349
DRR106348
DRR106347
DRR106346
sample
0.0
0.5
1.0
1.5
expression
life_stage
FLF
PF
S. venezuelensis RNAseq Dataset: Group FLF_PF

3.6.4 Plot of filtered, normalized log2CPM data by life stage

3.7 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).

3.8 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 `SvRNAseq_group_FLF_PF_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 (`Sv_geneAnnotations`)
   ii) the variance-stabilized vDGEList, saved as an R object (`Sv_vDGEList`)
   iii) a matrix of discarded genes and their raw counts (`SvRNAseq_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 (`SvRNAseq_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: Replicate above chunks for Group iL3_extended

4.1 Import Kallisto Reads

Collecting code for reading in Kallisto reads through Generating a DGEList. Gene annotations are not re-imported. 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).

4.2 Data Filtering and Normalization.

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

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

4.2.3 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. venezuelensis: Counts per Million (CPM)
genes excluded by low count filtering step, non−normalized, Group iL3_extended
DRR029445
DRR029444
DRR029443
DRR029442
DRR029441
DRR029440
DRR029439
DRR029438
DRR029437
DRR029436
DRR029435
DRR029434
DRR029433
DRR029282
sample
0.0
0.2
0.4
0.6
expression
RNAseq Dataset: Group iL3_extended
life_stage
L1
iL3
L3_lung
Young_FLF
FLF
iL3a_1d
iL3a_5d
Egg

4.2.4 Plot of filtered, normalized log2CPM data by life stage

4.3 Save Data

This code chunk saves data generated in Appendix I. However, it only save data required for downstream offline analyses, as this data is not included in the *Strongyloides* RNAseq browser. Thus, code for generating elements necessary for the browser are commented out.

5 Appendix II: Process all life stages together

Saving critical elements needed for demonstrating the existence of substantial batch effects during downstream analyses.

5.1 Import Kallisto Reads

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).

5.2 Data Filtering and Normalization

```
## [1] "Low copy number filtering step excluded a total of 737 genes"
```

5.3 Save Data and Annotations

This code chunk saves data, generated in Appendix II. This code chunk saves data generated in Appendix I. However, it only save data required for downstream offline analyses, as this data is not included in the

Strongyloides RNAseq browser. Thus, code for generating elements necessary for the browser are commented out.

6 Appendix III: All code for this report

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

suppressPackageStartupMessages({
  library(tidyverse)
  library(tximport)
  library(ensembldb)
  library(biomaRt)
  library(magrittr)
  library(biomaRt)
  library(edgeR)
  library(matrixStats)
  library(cowplot)
  library(ggthemes)
  library(RColorBrewer)
  library(gprofiler2)

})
# This script checks the qualitiy of the fastq files and performs
# an alignment to the Strongyloides venezuelensis 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.sh' (without the quotes)
# at the terminal prompt
# Then type './readMapping.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_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index
strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.fa

# map reads to the indexed reference host transcriptome

# Parasitic Females: Biological Replicates 1-6, Technical replicate set 1
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106346 -t 14 DRR106346_1.fastq.gz DRR106346_2.fastq.gz&> DRR106346.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106348 -t 14 DRR106348_1.fastq.gz DRR106348_2.fastq.gz&> DRR106348.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106350 -t 14 DRR106350_1.fastq.gz DRR106350_2.fastq.gz&> DRR106350.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106352 -t 14 DRR106352_1.fastq.gz DRR106352_2.fastq.gz&> DRR106352.log
```

```
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106354 -t 14 DRR106354_1.fastq.gz DRR106354_2.fastq.gz&> DRR106354.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106356 -t 14 DRR106356_1.fastq.gz DRR106356_2.fastq.gz&> DRR106356.log

# Parasitic Females: Biological Replicates 1-6, Technical replicate set 2
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106347 -t 14 DRR106347_1.fastq.gz DRR106347_2.fastq.gz&> DRR106347.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106349 -t 14 DRR106349_1.fastq.gz DRR106349_2.fastq.gz&> DRR106349.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106351 -t 14 DRR106351_1.fastq.gz DRR106351_2.fastq.gz&> DRR106351.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106353 -t 14 DRR106353_1.fastq.gz DRR106353_2.fastq.gz&> DRR106353.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106355 -t 14 DRR106355_1.fastq.gz DRR106355_2.fastq.gz&> DRR106355.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR106357 -t 14 DRR106357_1.fastq.gz DRR106357_2.fastq.gz&> DRR106357.log

# L1s: Technical Replicates 1-2
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029433 -t 14 DRR029433_1.fastq.gz DRR029433_2.fastq.gz&> DRR029433.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029434 -t 14 DRR029434_1.fastq.gz DRR029434_2.fastq.gz&> DRR029434.log

# iL3s: Technical Replicates 1-2
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029435 -t 14 DRR029435_1.fastq.gz DRR029435_2.fastq.gz&> DRR029435.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029436 -t 14 DRR029436_1.fastq.gz DRR029436_2.fastq.gz&> DRR029436.log

# L3s from lung: Technical Replicates 1-2
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029437 -t 14 DRR029437_1.fastq.gz DRR029437_2.fastq.gz&> DRR029437.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029438 -t 14 DRR029438_1.fastq.gz DRR029438_2.fastq.gz&> DRR029438.log

# Young Adult FLF: Technical Replicates 1-2
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029439 -t 14 DRR029439_1.fastq.gz DRR029439_2.fastq.gz&> DRR029439.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029440 -t 14 DRR029440_1.fastq.gz DRR029440_2.fastq.gz&> DRR029440.log

# Free-living Females: Technical Replicates 1-2
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029441 -t 14 DRR029441_1.fastq.gz DRR029441_2.fastq.gz&> DRR029441.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029442 -t 14 DRR029442_1.fastq.gz DRR029442_2.fastq.gz&> DRR029442.log

# Activated iL3s, 1 day and 5 day (treat as Biological Replicates)
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029443 -t 14 DRR029443_1.fastq.gz DRR029443_2.fastq.gz&> DRR029443.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
```

```
DRR029444 -t 14 DRR029444_1.fastq.gz DRR029444_2.fastq.gz&> DRR029444.log

# Eggs: Technical Replicates 1-2
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029445 -t 14 DRR029445_1.fastq.gz DRR029445_2.fastq.gz&> DRR029445.log
kallisto quant -i strongyloides_venezuelensis.PRJEB530.WBPS14.mRNA_transcripts.index -o
DRR029282 -t 14 DRR029282_1.fastq.gz DRR029282_2.fastq.gz&> DRR029282.log


# summarize fastqc and kallisto mapping results using MultiQC
multiqc -d .
# load packages ----
suppressPackageStartupMessages({
  library(tidyverse)
  library(tximport)
  library(ensembldb)
  library(biomaRt)
  library(magrittr)
})
# read in the study desig for Group FLF_PF ----
targets <- read_tsv("../Data/S_venezuelensis/Study_Design/Svene_Group_FLF_PF_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_venezuelensis/Reads",
                  targets$sample,
                  "abundance.tsv")

# get annotations using organism-specific package ----
Tx.Sv <- 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('stveneprjeb530')) %>%
  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 ----
# 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.Sv[,1:2],
```

```r
                      txOut = FALSE,
                      countsFromAbundance = "lengthScaledTPM",
                      ignoreTxVersion = FALSE)

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

# Load packages ------
suppressPackageStartupMessages({
  library(tidyverse)
  library(tximport)
  library(ensembldb)
  library(biomaRt)
  library(magrittr)
})
# Load data & study design for group FLF_PF----
load(file = file.path("../Data/S_venezuelensis",
                       "SvRNAseq_Group_FLF_PF_TPM"))

targets <- read_tsv("../Data/S_venezuelensis/Study_Design/Svene_Group_FLF_PF_study_design.txt",
                    na = c("", "NA", "na"))
# Get In-subclade group homologs for S. venezuelensis genes
# from BioMart and filter -----
Annt.temp.1 <- getBM(attributes=c('wbps_gene_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('stveneprjeb530')) %>%
  as_tibble() %>%
  #rename columns
  dplyr::rename(geneID = wbps_gene_id,
                In.subclade_geneID = stpapiprjeb525_gene,
                In.subclade_percent_homology= stpapiprjeb525_homolog_perc_id
  ) %>%
  dplyr::group_by(geneID)

# Get Out-subclade group homologs for S. venezuelensis genes
# from BioMart and filter -----
Annt.temp.2 <- getBM(attributes=c('wbps_gene_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('stveneprjeb530')) %>%
  as_tibble() %>%
  #rename columns
  dplyr::rename(geneID = wbps_gene_id,
                Out.subclade_geneID = ststerprjeb528_gene,
                Out.subclade_percent_homology= ststerprjeb528_homolog_perc_id
  ) %>%
  dplyr::group_by(geneID)

# Get C. elegans homologs and gene information for S. venezuelensis genes
# from BioMart and filter -----
Annt.temp.3 <- getBM(attributes=c('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'),
                     value = list('stveneprjeb530')) %>%
  as_tibble() %>%
  #rename columns
  dplyr::rename(geneID = 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 homolog aren't filtered out
Annt.import$Ce_percent_homology[
  is.na(Annt.import$Ce_percent_homology)] <- 1000
Annt.import$In.subclade_percent_homology[
```

```r
  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(In.subclade_geneID,
                  In.subclade_percent_homology,
                  Out.subclade_geneID,
                  Out.subclade_percent_homology,
                  .after = geneID) %>%
  column_to_rownames(var = "geneID")
```

```r
# List of S. venezuelensis 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),]

# 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="S. venezuelensis: Transcripts per million (TPM)",
       subtitle="unfiltered, non-normalized data, Group FLF_PF",
       caption="S. venezuelensis RNAseq Dataset: Group FLF_PF") +
  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$sample, batch = ta
                     group = targets$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)
```

```r
  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. venezuelensis: Log2 Counts per Million (CPM)",
       subtitle="unfiltered, non-normalized, Group FLF_PF",
       caption="S. venezuelensis RNAseq Dataset: Group FLF_PF") +
  theme_bw() +
  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)>=3
```

```r
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. venezuelensis: Log2 Counts per Million (CPM)",
       subtitle="filtered, non-normalized, Group FLF_PF",
       caption="S. venezuelensis RNAseq Dataset: Group FLF_PF") +
  theme_bw() +
  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) +
```

```r
    #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. venezuelensis: Counts per Million (CPM)",
         subtitle="genes excluded by low count filtering step, non-normalized, Group FLF_PF",
         caption="S. venezuelensis RNAseq Dataset: Group FLF_PF") +
    theme_bw() +
    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 = "stveneprjeb530", 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. venezuelensis: Log2 Counts per Million (CPM)",
       subtitle="filtered, TMM normalized, Group FLF_PF",
       caption="S. venezuelensis RNAseq Dataset: Group FLF_PF") +
  theme_bw() +
  coord_flip()
p4

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

# Set up the design matrix ----
# no intercept/blocking for matrix, comparisons across group
group <- factor(targets$group)
biolrep <- factor(targets$source)

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$group,
  v.DEGList.filtered.norm$targets$samples,
  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 ----
save(annotations,
```

```r
file = file.path(output.path,
                 "Sv_geneAnnotations"))

# Save a matrix of discarded genes and their raw counts ----
discarded.gene.df %>%
write.csv(file =
            file.path(output.path,
                      "SvRNAseq_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,
                           "SvRNAseq_log2cpm_filtered_norm_voom.csv"))
# Save v.DEGList ----
# This file can be imported into Shiny App

save(v.DEGList.filtered.norm,
     file = file.path(output.path, "Sv_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.
SvRNAseq.preprocessed.data <- list(
  targets = targets,
  annotations = annotations,
  log2.cpm.filtered.norm = log2.cpm.filtered.norm,
  myDGEList.filtered.norm = myDGEList.filtered.norm
)
save(SvRNAseq.preprocessed.data,
     file = file.path(output.path,
                      "SvRNAseq_group_FLF_PF_data_preprocessed"))


targets <- read_tsv("../Data/S_venezuelensis/Study_Design/Svene_Group_iL3_extended_study_design.txt",
                    na = c("", "NA", "na"))
path <- file.path("../Data/S_venezuelensis/Reads",targets$sample, "abundance.tsv")
Tx.Sv <- getBM(attributes=c('wbps_transcript_id',
                            'wbps_gene_id'),
               # grab the ensembl annotations for Wormbase Parasite genes
               mart = useMart(biomart="parasite_mart",
                              dataset = "wbps_gene",
                              host="https://parasite.wormbase.org",
                              port = 443),
               filters = c('species_id_1010'),
               value = list('stveneprjeb530')) %>%
    as_tibble() %>%
    #we need to rename the columns retreived from biomart
    dplyr::rename(target_id = wbps_transcript_id,
                  WB_geneID = wbps_gene_id) %>%
    dplyr::mutate(gene_name = str_remove_all(target_id, "\\.[0-9]$")) %>%
    dplyr::mutate(gene_name = str_remove_all(gene_name, "[a-c]$")) %>%
```

```
    dplyr::select(!WB_geneID)
Txi_gene <- tximport(path,
                     type = "kallisto",
                     tx2gene = Tx.Sv[,1:2],
                     txOut = FALSE, #How does the result change if this =FALSE vs =TRUE?
                     countsFromAbundance = "lengthScaledTPM",
                     ignoreTxVersion = FALSE)
# Save the raw transcript counts ----
save(Txi_gene,
     file = file.path("../Data/S_venezuelensis",
                      "SvRNAseq_Group_iL3_extended_TPM"))
# Load data & study design ----
load(file = file.path("../Data/S_venezuelensis",
                      "SvRNAseq_Group_iL3_extended_TPM"))
targets <- read_tsv("../Data/S_venezuelensis/Study_Design/Svene_Group_iL3_extended_study_design.txt",
                    na = c("", "NA", "na"))

myTPM.stats <- transform(Txi_gene$abundance,
                         SD=rowSds(Txi_gene$abundance),
                         AVG=rowMeans(Txi_gene$abundance),
                         MED=rowMedians(Txi_gene$abundance))
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="S. venezuelensis: Transcripts per million (TPM)",
         subtitle="unfiltered, non-normalized data, Group iL3_extended",
         caption="S. venezuelensis RNAseq Dataset: Group iL3_extended") +
    theme_bw()
p1
myDGEList <- DGEList(Txi_gene$counts,
                     samples = data.frame(samples = targets$source, source = targets$sample, batch = ta
                     group = targets$group,
                     genes = annotations)


ids <- rep(cbind(targets$group),
           times = nrow(myDGEList$counts)) %>%
    as_factor()
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)
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. venezuelensis: Log2 Counts per Million (CPM)",
         subtitle="unfiltered, non-normalized, Group iL3_extended",
         caption="RNAseq Dataset: Group iL3_extended") +
    theme_bw() +
    coord_flip()
p2
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. venezuelensis: Log2 Counts per Million (CPM)",
         subtitle="filtered, non-normalized, Group iL3_extended",
         caption="RNAseq Dataset: Group iL3_extended") +
    theme_bw() +
    coord_flip()
p3

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. venezuelensis: Counts per Million (CPM)",
         subtitle="genes excluded by low count filtering step, non-normalized, Group iL3_extended",
         caption="RNAseq Dataset: Group iL3_extended") +
    theme_bw() +
    coord_flip()
p.discarded
# discarded.geneID <- unique(log2.cpm.discarded.df.pivot$geneID)
# gost.res <- gost(list(Discarded_genes = discarded.geneID),
# organism = "stveneprjeb530",
# 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)

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. venezuelensis: Log2 Counts per Million (CPM)",
         subtitle="filtered, TMM normalized, Group iL3_extended",
         caption="RNAseq Dataset: Group iL3_extended") +
    theme_bw() +
    coord_flip()
p4

# Compute Variance-Stabilized DGEList Object
SvRNAseq.preprocessed.data <- list(targets = targets,
                                   annotations = annotations,
                                   log2.cpm.filtered.norm = log2.cpm.filtered.norm,
                                   myDGEList.filtered.norm = myDGEList.filtered.norm
)


group <- factor(targets$group)
biolrep <- factor(targets$source)
design <- model.matrix(~0 + group)
colnames(design) <- levels(group)
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

v.DEGList.filtered.norm <- avearrays(v.DEGList.filtered.norm.withtechreplicates,
                                     biolrep)

colnames(v.DEGList.filtered.norm$E) <- paste(
  v.DEGList.filtered.norm$targets$group,
  v.DEGList.filtered.norm$targets$samples,
  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 a matrix of discarded genes and their raw counts ----
# discarded.gene.df %>%
# write.csv(
#   file = file.path(output.path,
#                    "SvRNAseq_Group_iL3_extended_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,
```

```r
#           file = file.path(output.path,
#                            "SvRNAseq_group_iL3_extended_log2cpm_filtered_norm_voom.csv"))

# 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.
save(SvRNAseq.preprocessed.data,
     file = file.path(output.path,
                      "SvRNAseq_group_iL3_extended_data_preprocessed"))


save(v.DEGList.filtered.norm,
     file = file.path(output.path,
                      "Sv_group_iL3_extended_vDGEList"))


## Import Kallisto reads into R ----
suppressPackageStartupMessages({
  library(tidyverse)
  library(tximport)
  library(ensembldb)
  library(biomaRt)
  library(magrittr)
})
# read in the study design
targets <- read_tsv(
  "../Data/S_venezuelensis/Study_Design/PRJDB3457_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_venezuelensis/Reads",
                  targets$sample,
                  "abundance.tsv")

# get annotations using organism-specific package
Tx.Sv <- 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('stveneprjeb530')) %>%
  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)
```

```r
# import Kallisto transcript counts into R using Tximport
# 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.Sv[,1:2],
                     txOut = FALSE,
                     countsFromAbundance = "lengthScaledTPM",
                     ignoreTxVersion = FALSE)
# Save the raw transcript counts ----
save(Txi_gene,
     file = file.path("../Data/S_venezuelensis",
                      "SvRNAseq_allSamples_TPM"))

## Generate Digital Gene Expression List ----
# Load packages ------
suppressPackageStartupMessages({
  library(tidyverse)
  library(tximport)
  library(ensembldb)
  library(biomaRt)
  library(magrittr)
})

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

targets <- read_tsv(
  "../Data/S_venezuelensis/Study_Design/PRJDB3457_study_design.txt",
                    na = c("", "NA", "na"))

suppressPackageStartupMessages({
  library(tidyverse)
  library(edgeR)
  library(matrixStats)
  library(cowplot)
  library(ggthemes)
  library(RColorBrewer)
  library(gprofiler2)
})
# make a Digital Gene Expression list using the raw counts
myDGEList <- DGEList(Txi_gene$counts,
                     samples = data.frame(samples = targets$source,
                                          source = targets$sample,
                                          batch = targets$batch),
                     group = targets$group,
                     genes = annotations)



##  Data Filtering and Normalization ----
suppressPackageStartupMessages({
```

```r
  library(tidyverse)
  library(edgeR)
  library(matrixStats)
  library(cowplot)
  library(ggthemes)
  library(RColorBrewer)
  library(gprofiler2)
})
ids <- rep(cbind(targets$group),
           times = nrow(myDGEList$counts)) %>%
  as_factor()

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)

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)

myDGEList.discarded <- myDGEList[!keepers,]
print(paste('Low copy number filtering step excluded a total of',
            dim(myDGEList.discarded)[[1]], 'genes'))


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)
```

```r
# Save 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)

## Compute and Save Variance-Stabilized DGEList Object  ----
suppressPackageStartupMessages({
  library(tidyverse)
  library(limma) # differential gene expression using linear modeling
  library(edgeR)
})

# Set up the design matrix
group <- factor(targets$group)
biolrep <- factor(targets$source)
block <- factor (targets$batch)
#block <- factor(targets$block)
# trying to use a blocking design to handle possibility of
# batch effects between samples released in 2015 after sequencing
# on an Illumina HiSeq and those released in 2019 after sequencing on an
# Illumina MiSeq. There is a single overlapping sample, Free-living
# females; will use the (~block + group) setup.
design <- model.matrix(~block + 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
```

```r
# 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,
  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)
}

# discarded.gene.df %>%
# write.csv(file = file.path(output.path,
#   "SvRNAseq_allSamples_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,
#                            "SsRNAseq_allSamples_log2cpm_filtered_norm_voom.csv"))

# Save v.DEGList
# This file will be imported into Shiny App
save(v.DEGList.filtered.norm,
     file = file.path(output.path,
                      "Sv_allSamples_vDGEList"))

#
## Save Filtered Data and Annotations ----
SvRNAseq.preprocessed.data <- list(targets = targets,
                                   annotations = annotations,
                                   log2.cpm.filtered.norm =
                                     log2.cpm.filtered.norm,
                                   myDGEList.filtered.norm =
                                     myDGEList.filtered.norm
)
save(SvRNAseq.preprocessed.data,
     file = file.path(output.path,
                      "SvRNAseq_allSamples_data_preprocessed"))


sessionInfo()
```

7 Appendix IV: 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
```