RNA-seq data cleaning
Counts to voom

Kim Dill-McFarland, kadm@uw.edu

version April 27, 2022

Contents

Overview

This document covers the Hawn lab recommended data cleaning pipeline for RNA-seq gene count tables. This pipeline includes quality assessment, filtering, and normalization of count tables generated using our SEAsnake pipeline. The example data are human, bulk, paired-end RNA-seq, but this pipeline can be applied to other organisms or single-read libraries.

0. Setup

Software

This pipeline should be completed in R and RStudio. You should also install the following packages.

```r
#CRAN packages
install.packages(c("tidyverse", "DiagrammeR", "ggrepel", "scales"))

#Bioconductor packages
install.packages("BiocManager")
BiocManager::install(c("edgeR", "limma", "biomaRt", "patchwork"))

#GitHub packages
install.packages("devtools")
devtools::install_github("BIGslu/BIGpicture", force=TRUE)
devtools::install_github("BIGslu/RNAetc", force=TRUE)
devtools::install_github("zhangyuqing/sva-devel")
```

And load them into your current R session.

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5      v purrr   0.3.4
## v tibble  3.1.6      v dplyr   1.0.8
## v tidyr   1.2.0      v stringr 1.4.0
## v readr   2.1.2      v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(BIGpicture)
library(RNAetc)
library(scales)
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```r
library(sva)
```

```
## Loading required package: mgcv
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

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

```
## This is mgcv 1.8-38. For overview type 'help("mgcv-package")'.

## Loading required package: genefilter

##
## Attaching package: 'genefilter'

## The following object is masked from 'package:readr':
##
##     spec

## Loading required package: BiocParallel
```

```r
library(ggrepel)
library(patchwork)
```

```
##
## Attaching package: 'patchwork'

## The following object is masked from 'package:genefilter':
##
##     area
```

```r
library(limma)
library(edgeR)
#Note we do not load biomaRt because it has conflicts with the tidyverse.
# We instead call its functions with biomaRt::

set.seed(651)
```

Directory structure

To directly use the code in this pipeline, you must organize your files as follows.

The `data_raw/` directory should contain outputs from SEAsnake pipeline including a counts table (`combined_feature_counts.tsv`) and samtools flagstat (`combined_flagstat.tsv`) and Picard tables (`combined_picard.tsv`). Additional sample or patient metadata may also be present in tables

Example data

Example data were obtained from virus-stimulated human plasmacytoid dendritic cells. Actual patient identifiers and metadata have been altered for this tutorial.

> Dill-McFarland KA, Schwartz JT, Zhao H, Shao B, Fulkerson PC, Altman MC, Gill MA. 2022. Eosinophil-mediated suppression and Anti-IL-5 enhancement of plasmacytoid dendritic cell interferon responses in asthma. J Allergy Clin Immunol. Epub ahead of print. doi: 10.1016/j.jaci.2022.03.025. — GitHub

1. Format data

1.1: Alignment metrics

The SEAsnake pipeline outputs library quality metrics from samtools `flagstat` and `picard`. For more details, see flagstat and Picard RnaSeqMetrics.

Let's load in the metrics files.

```
flagstat <- read_tsv("data_raw/combined_flagstat.tsv")
```

```
## Rows: 12 Columns: 17
## -- Column specification ---------------------------------------------------
## Delimiter: "\t"
## chr  (1): libID
## dbl (16): QC_pass, primary, secondary, supplementary, duplicate, primary_dup...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
picard <- read_tsv("data_raw/combined_picard.tsv")
```

```
## Rows: 12 Columns: 31
## -- Column specification ---------------------------------------------------
## Delimiter: "\t"
## chr  (1): libID
## dbl (27): PF_BASES, PF_ALIGNED_BASES, RIBOSOMAL_BASES, CODING_BASES, UTR_BAS...
## lgl  (3): SAMPLE, LIBRARY, READ_GROUP
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

1.2: Sample and library metadata

Next, we read in and combine any patient and/or sample metadata. You may have only one of these data types. In general, patient data include things like age, sex, and any variable that is assigned to each person in the study. In contrast, sample data are variables assigned to each library, like experimental treatment, quality, etc. We combine these along with the above metrics tables to make a single data frame with all metadata.

```
patient <- read_csv("data_raw/P259_patient_meta.csv")
```

```
## Rows: 5 Columns: 4
## -- Column specification ---------------------------------------------------
## Delimiter: ","
## chr (3): ptID, asthma, sex
## dbl (1): age
##
```

```
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
sample <- read_csv("data_raw/P259_sample_meta.csv") %>%
  #format any variables that need to be factors
  mutate(virus = fct_relevel(factor(virus), ref="none"))
```

```
## Rows: 12 Columns: 4
## -- Column specification -----------------------------------------------------
## Delimiter: ","
## chr (4): libID, ptID, virus, batch
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
meta <- full_join(sample, patient, by = "ptID") %>%
  full_join(flagstat, by = "libID") %>%
  full_join(picard, by = "libID")
```

This adds the following variable to the library metrics.

```
## # A tibble: 10 x 6
##     ptID   virus batch asthma     age sex
##     <chr>  <fct> <chr> <chr>    <dbl> <chr>
##  1 donor1 none  A     healthy     35 F
##  2 donor1 HRV   A     healthy     35 F
##  3 donor2 none  A     healthy     47 M
##  4 donor2 HRV   A     healthy     47 M
##  5 donor3 none  B     healthy     22 M
##  6 donor3 HRV   A     healthy     22 M
##  7 donor4 none  A     asthma      25 F
##  8 donor4 HRV   A     asthma      25 F
##  9 donor5 none  B     asthma      26 M
## 10 donor5 HRV   B     asthma      26 M
```

1.3: Counts table

Output by Subread `featureCounts` in SEAsnake, the counts table can be read directly into R.

```r
count <- read_tsv("data_raw/combined_feature_counts.tsv")
```

```
## Rows: 19118 Columns: 13
## -- Column specification -----------------------------------------------------
## Delimiter: "\t"
## chr  (1): Geneid
## dbl (12): lib1, lib2, lib3, lib4, lib5, lib6, lib7, lib8, lib9, lib10, lib11...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
count[1:3,1:5]
```

```
## # A tibble: 3 x 5
##   Geneid          lib1  lib2  lib3  lib4
##   <chr>          <dbl> <dbl> <dbl> <dbl>
## 1 ENSG00000284662    0     0     0     0
## 2 ENSG00000186827  240   126   595   180
## 3 ENSG00000186891  185    50   255    37
```

2. Quality-filter data

2.1: Filter poor-quality libraries

We assess sample quality using several metrics from samtools `flagstat` and `picard`. Our standard assessment includes:

- `QC_pass`: total sequences passing quality control (QC), adapter and primer removal with `AdapterRemoval`
- `paired_mapped`: total QC pass sequences where both read 1 and 2 aligned to the genome. If you had non-paired-end data, you would use `mapped`
- `pct_align`: percent alignment i.e. `paired_mapped`/`QC_pass`
- `MEDIAN_CV_COVERAGE`: median coefficent of variation of coverage i.e. how variable sequence coverage is across the genome

Ideal libraries have high total sequences, high percent alignment, and low CV coverage. Cutoffs for sample removal will vary by data set but our starting recommendations are to remove libraries with:

- sequences < 1,000,000
- CV coverage > 1
- alignment < 75%

Set your cutoffs here.

```
seq_cutoff <- 1E6
cv_cutoff <- 1
align_cutoff <- 75
```

For the example data, we see that we do not need to remove any libraries due to total sequences.

```
ggplot(meta, aes(x = reorder(libID, QC_pass), y = QC_pass)) +
  geom_col() +
  #Add cutoff line
  geom_hline(yintercept = seq_cutoff) +
  #Log scale y-axis
  scale_y_continuous(trans = 'log10',
    breaks = trans_breaks("log10", function(x) 10^x),
    labels = trans_format("log10", math_format(10^.x))) +
  #Beautify
  theme_classic() +
  labs(x = "Library", y = "Pass-filter sequences (log scale)") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```
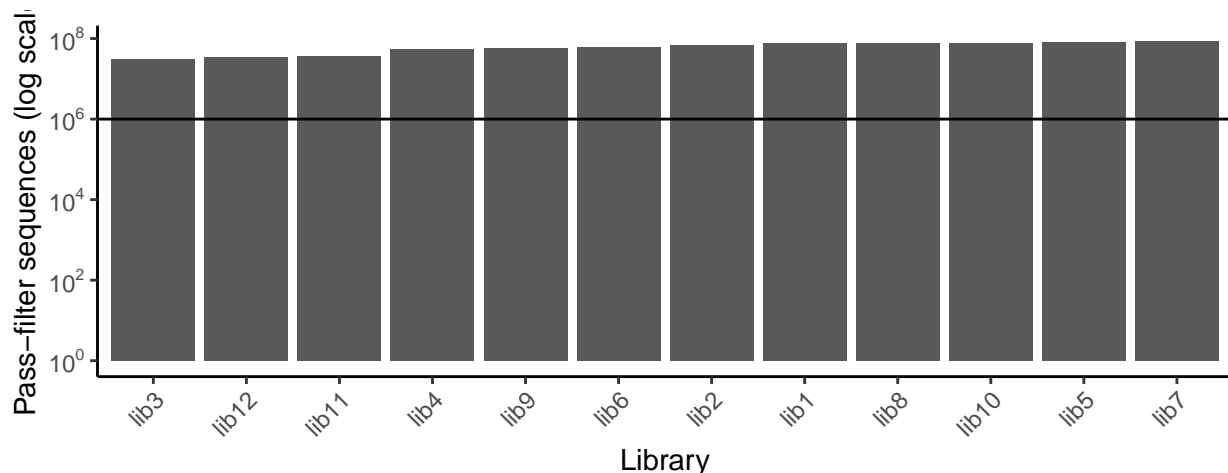
However, there are two libraries with median CV coverage near our cutoff. Note that we use `geom_text_repel` to label these libraries.

```r
#Set x-axis max to the larger of 1 or the highest data value
CV_max <- max(1, max(meta$MEDIAN_CV_COVERAGE))

#Set cutoffs to label with libID
cv_cutoff_label <- 0.9
align_cutoff_label <- 75

ggplot(meta, aes(x = MEDIAN_CV_COVERAGE, y = paired_mapped/QC_pass*100)) +
  geom_point() +
  #Rescale axis limits
  lims(x = c(0,CV_max), y=c(0,100)) +
  #Add cutoff lines
  geom_hline(yintercept = align_cutoff, lty="dashed") +
  geom_vline(xintercept = cv_cutoff, lty="dashed") +
  #Label points outside cutoffs
  geom_text_repel(data=filter(meta, MEDIAN_CV_COVERAGE > cv_cutoff_label |
                                paired_mapped/QC_pass*100 < align_cutoff_label),
          aes(label=libID), show.legend = FALSE, max.overlaps = Inf,
          box.padding = 1) +
  #Beautify
  theme_classic() +
  labs(x = "Median CV coverage", y="Percent alignment")
```
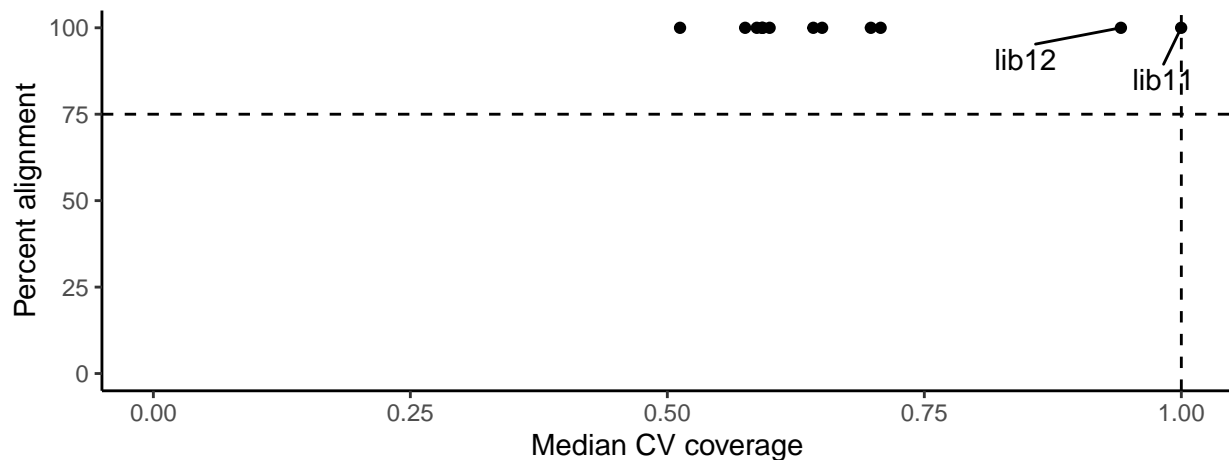

We now filter by our cutoffs. In this case, this leaves the questionable libraries (as we will see why they differ in a later step).

```r
meta.filter <- meta %>%
  filter(MEDIAN_CV_COVERAGE < cv_cutoff & QC_pass > seq_cutoff &
           paired_mapped/QC_pass*100 > align_cutoff)

count.filter <- count %>%
  select(1, all_of(meta.filter$libID))
```

This removes 0 libraries. However, you may choose to remove low-quality libraries at this stage.

2.2: Filter non-protein-coding genes

Standard differential expression analyses focus on protein-coding genes as these RNA products are the most likely to result in a measurable phenotype.

We annotate the counts table genes to their biotypes as well as additional identifiers with `biomaRt`.

```
#Get database
ensembl <- biomaRt::useEnsembl(biomart="ensembl", dataset="hsapiens_gene_ensembl",
                               mirror = "uswest")


#Format gene key
key <- biomaRt::getBM(attributes=c("ensembl_gene_id", "entrezgene_id", "hgnc_symbol",
                                   "gene_biotype", "chromosome_name",
                                   "start_position", "end_position"), mart=ensembl) %>%
  #Filter protein coding genes
  filter(gene_biotype == "protein_coding")

key.filter <- key %>%
  #Filter protein coding genes in count table
  filter(ensembl_gene_id %in% count$Geneid) %>%
  #collapse multiannotations.
  group_by(ensembl_gene_id, hgnc_symbol, gene_biotype,
           chromosome_name, start_position, end_position) %>%
  summarise(entrezgene_id = list(unique(entrezgene_id)), .groups = "drop") %>%
  group_by(ensembl_gene_id, entrezgene_id, gene_biotype,
           chromosome_name, start_position, end_position) %>%
  summarise(hgnc_symbol = list(unique(hgnc_symbol)), .groups = "drop")
```

Then, we filter the count table to genes in the protein-coding key.

```
count.filter.pc <- count.filter %>%
  filter(Geneid %in% key.filter$ensembl_gene_id)
```

This removes 104 genes from this data set. These data were pre-filtered so you will see A LOT more genes removed in this step in your data. In the current human genome (GRCh38 release 106), there are 23036 protein-coding genes, of which 19014 were found in this data set.

2.3: Correct batch effects

If your libraries were obtained from multiple experiments and/or sequenced on multiple runs, batch effects may exist in the data. These effects will mask true biological signals and should be corrected prior to modeling. There are a number of ways to batch correct. Our preferred method is with ComBat-Seq implemented in `sva`.

First, we look for batch effects in PCA. These effects may be apparent in an overall shift of libraries in a batch.

```
BIGpicture::plot_pca(count.filter.pc, meta=meta.filter,
                     vars="batch", transform_logCPM=TRUE)
```

```
## Joining, by = "libID"
```

```
## $batch
```

8
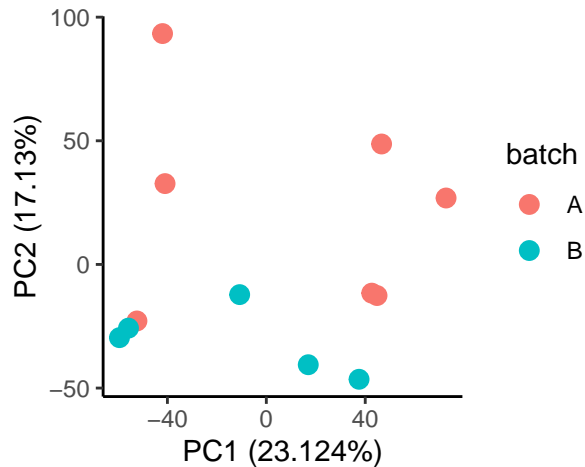
Or by highlighting libraries sequenced in multiple batches. In your analysis, you will need to change the `unique.ID` variables to those that uniquely identify a sample. For example, these samples can be uniquely identified by patient and virus stimulation.

```r
unique.ID <- c("ptID", "virus")

#Identify duplicate libraries
dups <- meta.filter %>%
  unite("dupID", all_of(unique.ID), remove=FALSE)  %>%
  count(dupID) %>%
  filter(n>1)

#Create duplicate ID variable
meta.filter.dups <- meta.filter %>%
  unite("dupID",unique.ID,remove=FALSE)  %>%
  mutate(duplicate = ifelse(dupID %in% dups$dupID, dupID, NA))
```
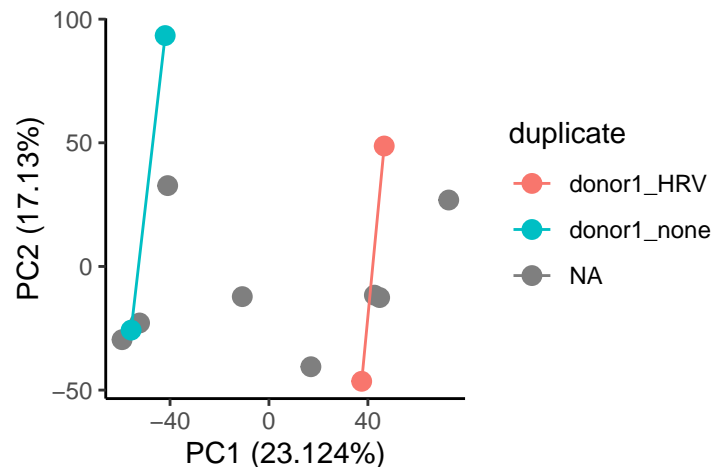
```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(unique.ID)` instead of `unique.ID` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```r
BIGpicture::plot_pca(count.filter.pc, meta=meta.filter.dups,
                     vars="duplicate", transform_logCPM=TRUE)
```

```
## Joining, by = "libID"
```

```
## $duplicate
```

Here, we see apparent batch effects where batch A tends to have lower PC2 values than batch B. This is re-enforced by the duplicate samples which group on PC2 more by batch than by duplicate samples.

To correct batch effects, we use negative binomial normalization with `sva` ComBat-Seq including any main `group` effects (virus stimulation) and co-variates `covar` (sex) we want to retain in our later models. This ensures that we don't over-correct the data and lose our biological signals of interest.

We also reduce the data set with `shrink = TRUE, gene.subset.n = 2` to run models on 2 random genes. This significantly reduces computational time for this tutorial. However, a real analysis should be run on a larger subset. We recommend anywhere from 10,000 to all genes in the data.

```
count.filter.pc.combat <- count.filter.pc %>%
  #Convert count df to matrix
  column_to_rownames("Geneid") %>%
  as.matrix() %>%
  #Batch correction
  sva::ComBat_seq(., batch = meta.filter$batch,
                  group = meta.filter$virus,
                  covar_mod = model.matrix(~ sex, meta.filter),
              shrink = TRUE, gene.subset.n = 2) %>%
  as.data.frame()
```

```
## Found 2 batches
## Using full model in ComBat-seq.
## Adjusting for 2 covariate(s) or covariate level(s)
## Estimating dispersions
## Fitting the GLM model
## Apply shrinkage - computing posterior estimates for parameters
## Using 2 random genes for Monte Carlo integration
## Apply shrinkage to mean only
## Adjusting the data
```

Note that the total "co-variates" listed in the above message includes non-reference levels of the main effects (HRV) plus variables listed as `covar` (sex).

This new data set should have reduced batch effects. However, some batch effects likely still exist as this is a difficult correction and we err on the side of less correction to preserve as much of the original data as possible. In this example, we did a rather poor correction based on only 2 genes. So not surprisingly, there is still apparent splitting by batch.
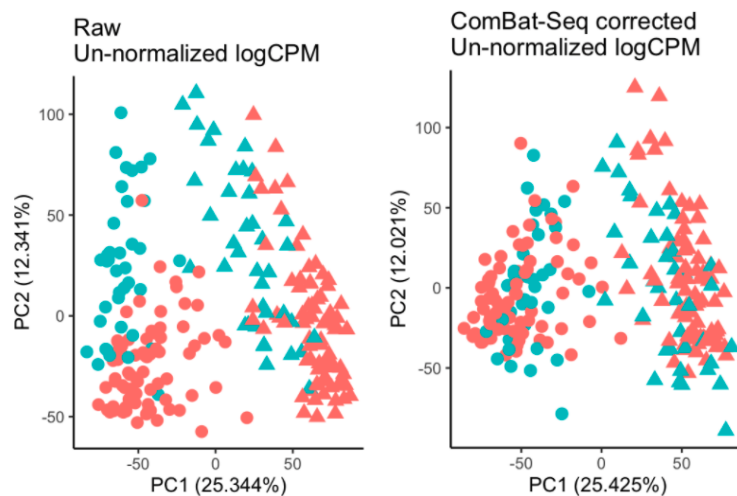
```
BIGpicture::plot_pca(count.filter.pc.combat, meta=meta.filter.dups,
                     vars=c("batch","duplicate"), transform_logCPM=TRUE) %>%
```

```
wrap_plots(ncol=2)
```

```
## Joining, by = "libID"
```
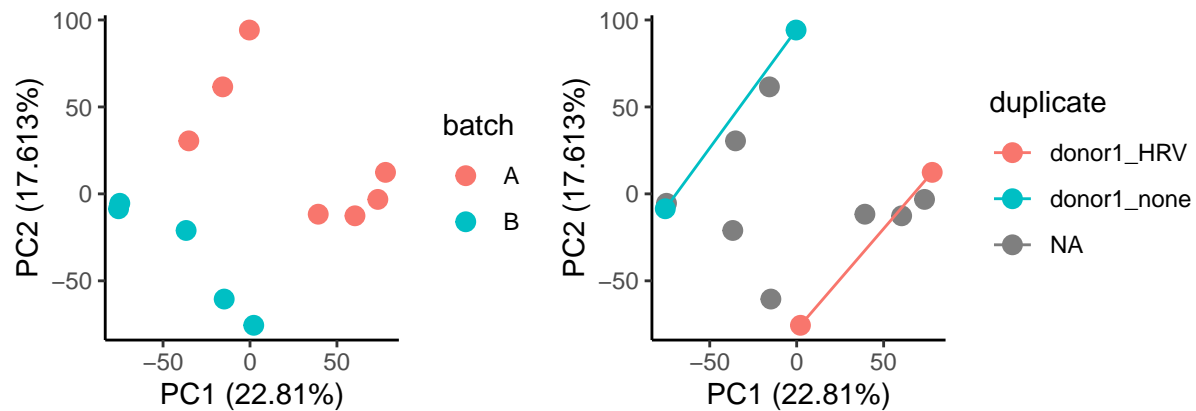

In another data set corrected using all genes, we can more readily see a reduction in the batch effects shown by color below.

Other batch correction methods   If ComBat-Seq does not appear to reduce the batch effects in your data, you may also consider:

- `scBatch` quantile normalization with `QuantNorm( )`
- `limma` batch correction based only on duplicate samples with `removeBatchEffect( )`. This would be applied during linear modeling as seen in the voom to DEG tutorial
- Including batch as a variable in linear modeling. You may do this in addition to batch correction or on its own with the uncorrected data

Choosing not to correct batch effects   Even if you have measurable batch effects, you may not wish to perform batch correction. This correction directly alters count values and can lead to lose of signal for your actual variables of interest. Some things to consider include:

- How strong are the batch effects? This can be roughly estimated based on the PC to which the batch variable best aligns. In the above data, this is PC2 which explains a substantial percent of variation (17%). Thus, the batch effects are also likely substantial. If the batch PC explains only a small amount of variation, you may wish to skip batch correction as the effects were minimal.

11

- Does batch correction improve the PCA? After running batch correction, you may not see much improvement in the PCA. Thus, this method did not improve your data and you will likely need to include batch in your models and carefully consider any genes of interest that are also significant for batch.
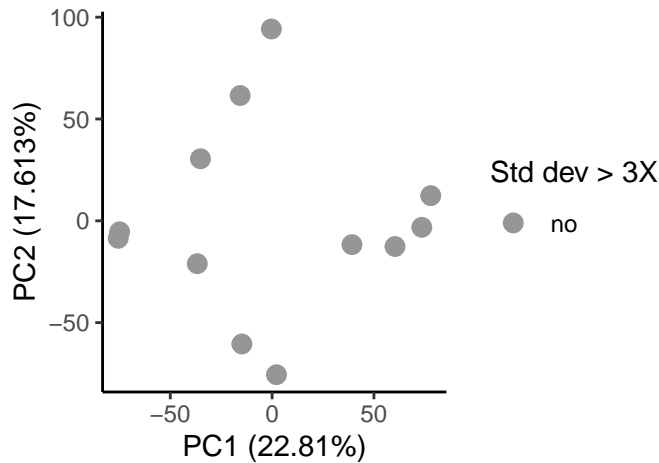
2.4: Filter PCA outliers

Sometimes one or more libraries will appear as outliers in PCA. We define this as any library more than 3 standard deviations away from the mean on PC1 and/or PC2. Here, we have no outliers.

```
pca3 <- BIGpicture::plot_pca(count.filter.pc.combat, meta=meta.filter,
                             vars="outlier", transform_logCPM=TRUE)
```

```
## Joining, by = "libID"
```

```
pca3
```

```
## $outlier
```


You may define outliers at a more or less strict level (`outlier_sd`) or wish to determine them separately within each group of interest (`outlier_group`). For example here, there is one outlier more than 1.5 sd away from the mean within batch.

```
BIGpicture::plot_pca(count.filter.pc.combat, meta=meta.filter,
                     vars="outlier", transform_logCPM=TRUE,
                     outlier_sd = 1.5, outlier_group = "batch")
```

```
## Joining, by = "libID"
```

```
## $outlier
```
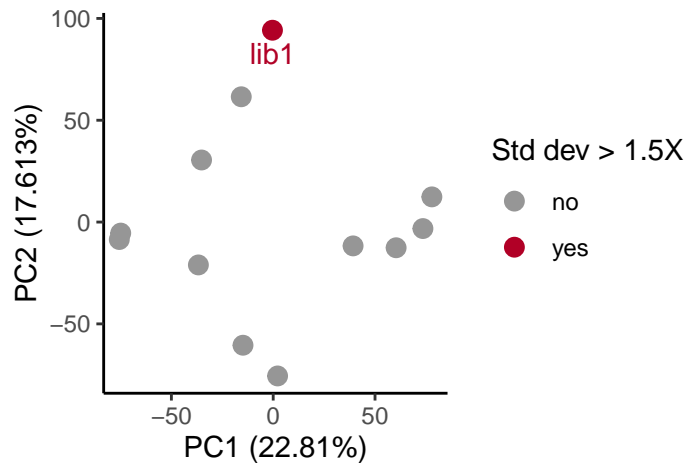
We recommend that you initially remove dramatic outliers but leave those that are borderline or questionable. Then, you can re-assess outliers after gene filtering and normalization. You may find that some are no longer outliers after these steps. If they are, you can return to this step and remove them before repeating subsequent steps.

If outlier libraries need to be removed, you can use the following code to do so.

```
not.outlier <- pca3$outlier$data %>%
  filter(col.group == "no")

meta.filter.out <- meta.filter %>%
  filter(libID %in% not.outlier$libID)

count.filter.pc.combat.out <- count.filter.pc.combat %>%
  select(1, all_of(meta.filter.out$libID))
```

Here, this removes 0 libraries because we do not have any at std dev > 3X.

2.5: Filter duplicate libraries

Your data may have have one or more libraries that represent the same sample. This is common when multiple sequencing batches are completed and a couple of samples are re-sequenced to aid in batch effect detection. Unless the majority of samples have multiple libraries (*i.e.* technical replicates), the duplicates should be removed or they will skew downstream analysis.

We usually keep the library with the highest number of sequences like so.

```
#Using the name unique ID variables as seen in batch correction
unique.ID
```

```
## [1] "ptID"  "virus"
```

```
#Find libraries with highest seqs
meta.filter.out.dedup <- meta.filter.out %>%
  group_by_at(unique.ID) %>%
  slice_max(order_by = QC_pass)

#Filter lower seq libraries
count.filter.pc.combat.out.dedup <- count.filter.pc.combat.out %>%
  select(1, all_of(meta.filter.out.dedup$libID))
```

Here, this removes 2 libraries, which matches what we expect from the duplicate PCA. Note that these two

libraries were the ones with questionable median CV coverage. So we have resolved this quality issue without removing them in step 2.1.

2.6: Create DGEList

At this stage, we've completed sample filtering and can collapse our count and meta data into a single list object. This allows us to shorten our long object names as well as works efficiently with the remaining cleaning steps.

First, let's ensure that all the data are in the same order.

```
meta.filter.out.dedup.ord <- meta.filter.out.dedup %>%
  arrange(libID)

count.filter.pc.combat.out.dedup.ord <- count.filter.pc.combat.out.dedup %>%
  select(1, all_of(meta.filter.out.dedup.ord$libID))

#check
identical(meta.filter.out.dedup.ord$libID,
          colnames(count.filter.pc.combat.out.dedup.ord))
```

```
## [1] TRUE
```

Then, we merge into the DEGList object, `edgeR` format.

```
dat <- DGEList(
  #count table
  # Note that if you did not do batch correction, you may need to
  # move gene names from a variable in the df to rownames
  counts=as.matrix(count.filter.pc.combat.out.dedup.ord),
  #metadata
  samples=meta.filter.out.dedup.ord,
  #gene info
  genes=key.filter)
```

2.7: Filter low abundance genes

Low abundance (small counts) and rare genes (many 0 counts) are removed from the data because they:

- are unlikely to be significantly differentially expressed
- greatly inflate multiple comparison correction
- often do not meet linear modeling assumptions regarding mean variance trends (*e.g.* because of small N, they show lower variance than what is expected for their mean expression)

Our goal is to remove genes in the lower left of the mean-variance plot because counts (x) and variance (y) are low *e.g.* these genes break the red mean variance trend.

```
BIGpicture::plot_mv(dat, design = "~ virus")
```

voom mean−variance trend
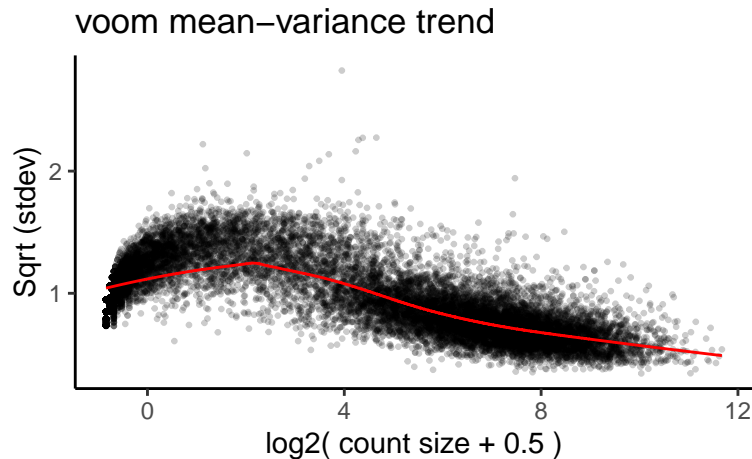
We use our custom function to retain only genes that are at least `min.CPM` counts per million in at least `min.sample` number of samples OR in at least `min.pct` percent of samples. Here, we use 0.1 CPM in at least 3 samples.

```
dat.abund <- RNAetc::filter_rare(dat, min.CPM = 0.1, min.sample = 3,
                                 gene.var="ensembl_gene_id")
```

```
## 5713 (30.05%) of 19014 genes removed.
```

Plotting the filtered data, we see the red trend line no longer curves down on the left. There is still a bit of the downward tail of dots but this filtering is sufficient.

```
plot_mv(dat.abund, design = "~ virus")
```

voom mean−variance trend
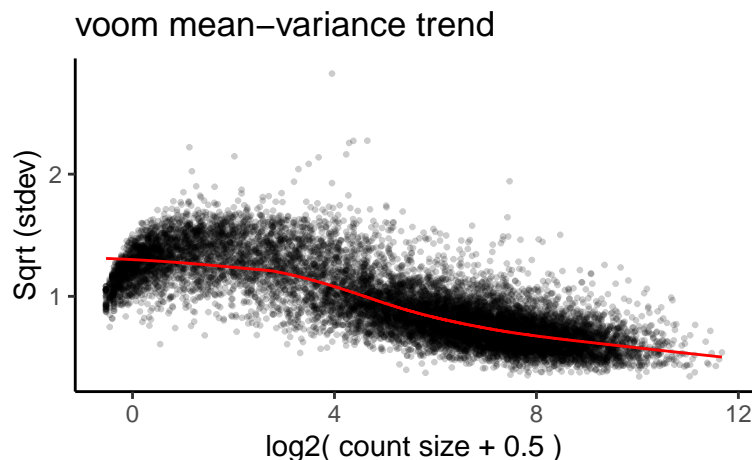
There is no exact cutoff for this filtering, and you should try several cutoffs to observe the effects. In general, we use minimum CPM from 0.1 - 1, minimum samples around 3 for small data sets, or minimum samples from 5 - 10% in larger data sets. It is important to keep the minimum sample cutoff larger than your smallest group of interest, otherwise you may filter genes specifically associated with one group. For example, if you have 10 libraries with 5 each of media vs stimulated, you're `min.sample` value should not be > 5 or else you will remove genes only expressed in one of media or stimulated groups.

This filtering generally results in the removal of around 25% of genes. Here, our filtering is on the stricter side (which is common in small data sets), removing 5713 or 30% of genes.

You may also wish to look for specific genes of interest and ensure they are not being filtered. This following calculates some statistics on filtered genes and saves a `csv` for your perusal.

```
rare <- as.data.frame(cpm(dat$counts)) %>%
  #Filter genes removed
  rownames_to_column("ensembl_gene_id") %>%
  filter(!(ensembl_gene_id %in% rownames(dat.abund$counts))) %>%
  #Add gene symbols
  left_join(dat$genes, by = "ensembl_gene_id") %>%
  #format
  select(-c(chromosome_name:end_position)) %>%
  pivot_longer(-c(ensembl_gene_id, gene_biotype, hgnc_symbol, entrezgene_id)) %>%
  group_by(ensembl_gene_id, gene_biotype, hgnc_symbol) %>%
  summarise(mean.CPM = mean(value, na.rm=TRUE),
            min.CPM = min(value, na.rm=TRUE),
            max.CPM = max(value, na.rm=TRUE),
            express.in.libs = length(value[value > 0]),
            .groups="drop")

write_csv(rare, file="data_clean/P259_rare_genes.csv")
rare
```

```
## # A tibble: 5,713 x 7
##    ensembl_gene_id gene_biotype   hgnc_symbol mean.CPM min.CPM max.CPM
##    <chr>           <chr>          <list>          <dbl>   <dbl>   <dbl>
##  1 ENSG00000000005 protein_coding <chr [1]>      0            0   0
##  2 ENSG00000001617 protein_coding <chr [1]>      0.141        0   0.916
##  3 ENSG00000001626 protein_coding <chr [1]>      0.0496       0   0.496
##  4 ENSG00000002745 protein_coding <chr [1]>      0.0377       0   0.377
##  5 ENSG00000002746 protein_coding <chr [1]>      0            0   0
##  6 ENSG00000003137 protein_coding <chr [1]>      0.0373       0   0.373
##  7 ENSG00000003249 protein_coding <chr [1]>      0            0   0
##  8 ENSG00000003989 protein_coding <chr [1]>      0            0   0
##  9 ENSG00000004776 protein_coding <chr [1]>      0            0   0
## 10 ENSG00000004799 protein_coding <chr [1]>      0.183        0   1.83
## # ... with 5,703 more rows, and 1 more variable: express.in.libs <int>
```

3. Normalize data

3.1: Trimmed mean of M (TMM)

RNA-seq counts are not independent within a library and not comparable across libraries. A library with 1 million sequences will have higher counts for most genes than one with 1 thousand sequences. We correct for this aspect of the data with the following normalization steps.

TMM defines a reference sample from your data set as the one with the most representative expression for the overall data set. Specifically, the reference sample is the one whose upper quartile is closest to the overall data set upper quartile. The upper quantile is the value (x) where 75% of genes have expression $< x$. Thus, the reference sample is the sample whose x is the closest to mean(x) across all samples.

All other samples are considered test samples. For each test sample, a scaling factor is calculated based on the weighted mean of log ratios of representative genes between the test and reference. These representative genes are a subset of the data set, removing the highest and lowest expressed genes as well as genes with the highest and lowest log ratios. The exact genes used as representative genes for scaling are dynamic and specific to each test sample.

The calculated scaling factors are then applied to the counts table automatically in the voom step.

```
dat.abund.norm <- calcNormFactors(dat.abund, method = "TMM")
```

## 3.2: voom aka log2 counts per million (CPM)

We continue normalization by converting counts to CPM within each sample, thus accounting for differential sampling depth. We also perform log2 transformation, because RNA-seq data are heavily right-skewed and thus, violate assumptions of normality.

```
as.data.frame(dat.abund$counts) %>%
  rownames_to_column() %>%
  pivot_longer(-rowname) %>%
ggplot() +
  geom_histogram(aes(x=value), bins = 100) +
  theme_classic() +
  labs(x = "count", y = "occurences") +
  lims(x=c(0,1000))
```
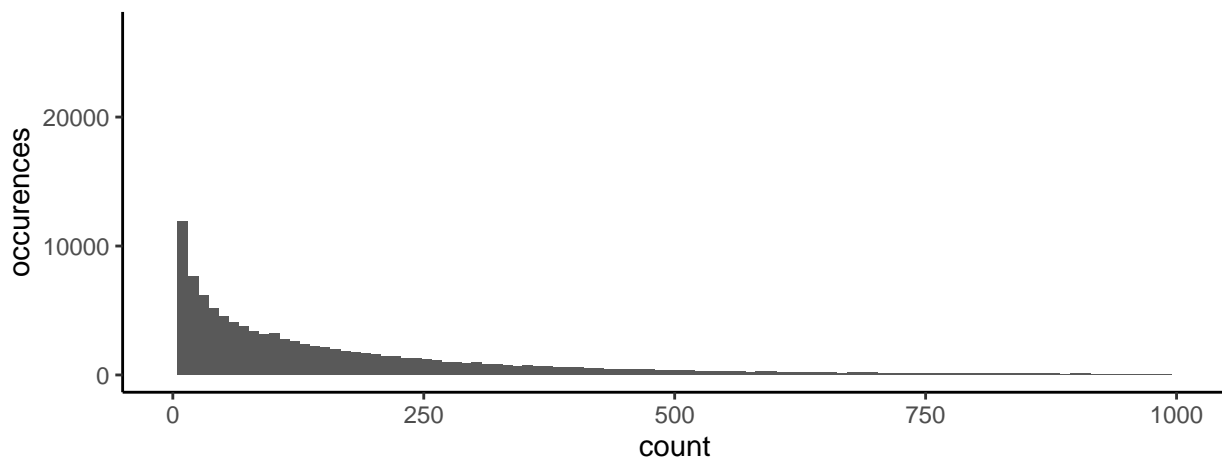

voom performs both of these steps! We use voomWithQualityWeights to additionally calculate sample specific quality weights that can be of use as co-variates in linear modeling.

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

voom: Mean–variance trend

Sample–specific weights
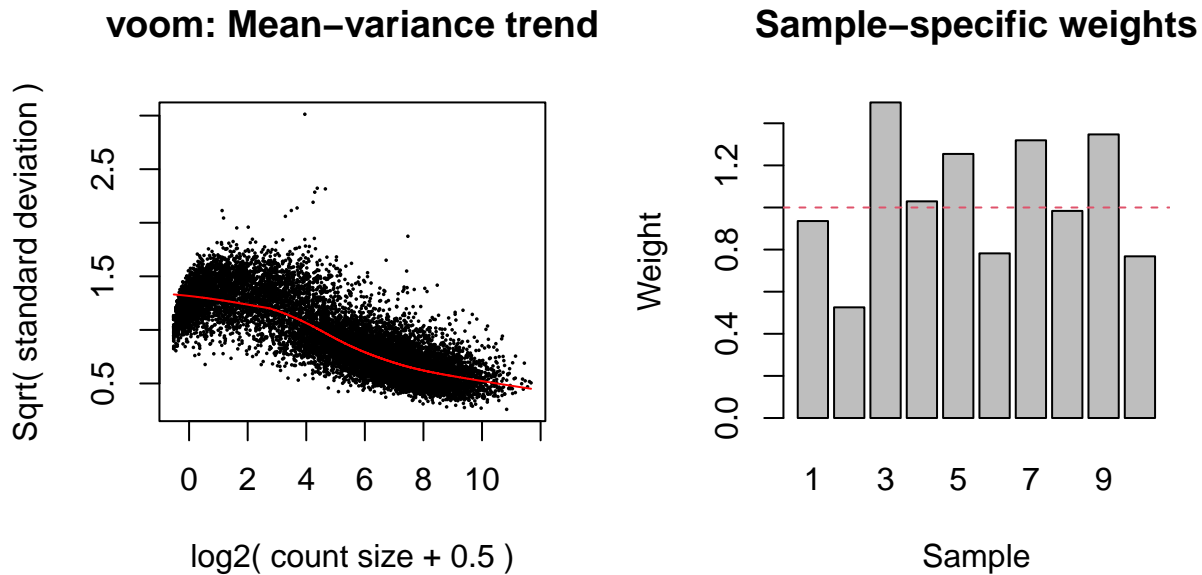
4. Summary and save

Filtering   The following summarizes libraries removed from analysis. In this tutorial, all libraries pass-filter and move on to analysis.

```
## # A tibble: 12 x 2
##    libID filter
##    <chr> <chr>
##  1 lib1  pass-filter
##  2 lib2  pass-filter
##  3 lib3  pass-filter
##  4 lib4  pass-filter
##  5 lib5  pass-filter
##  6 lib6  pass-filter
##  7 lib7  pass-filter
##  8 lib8  pass-filter
##  9 lib9  pass-filter
## 10 lib10 pass-filter
## 11 lib11 duplicate sample
## 12 lib12 duplicate sample
```

PCA   To get an initial sense of the cleaning data, we plot all our variables of interest as well as double-check any potential outliers. Here, we see a clear viral stimulation signal, some lingering batch effects, and no outliers.

```
plot_pca(dat.abund.norm.voom, vars = c("virus","ptID","sex","batch","outlier")) %>%
  wrap_plots(ncol=2)
```

```
## Joining, by = "libID"
```
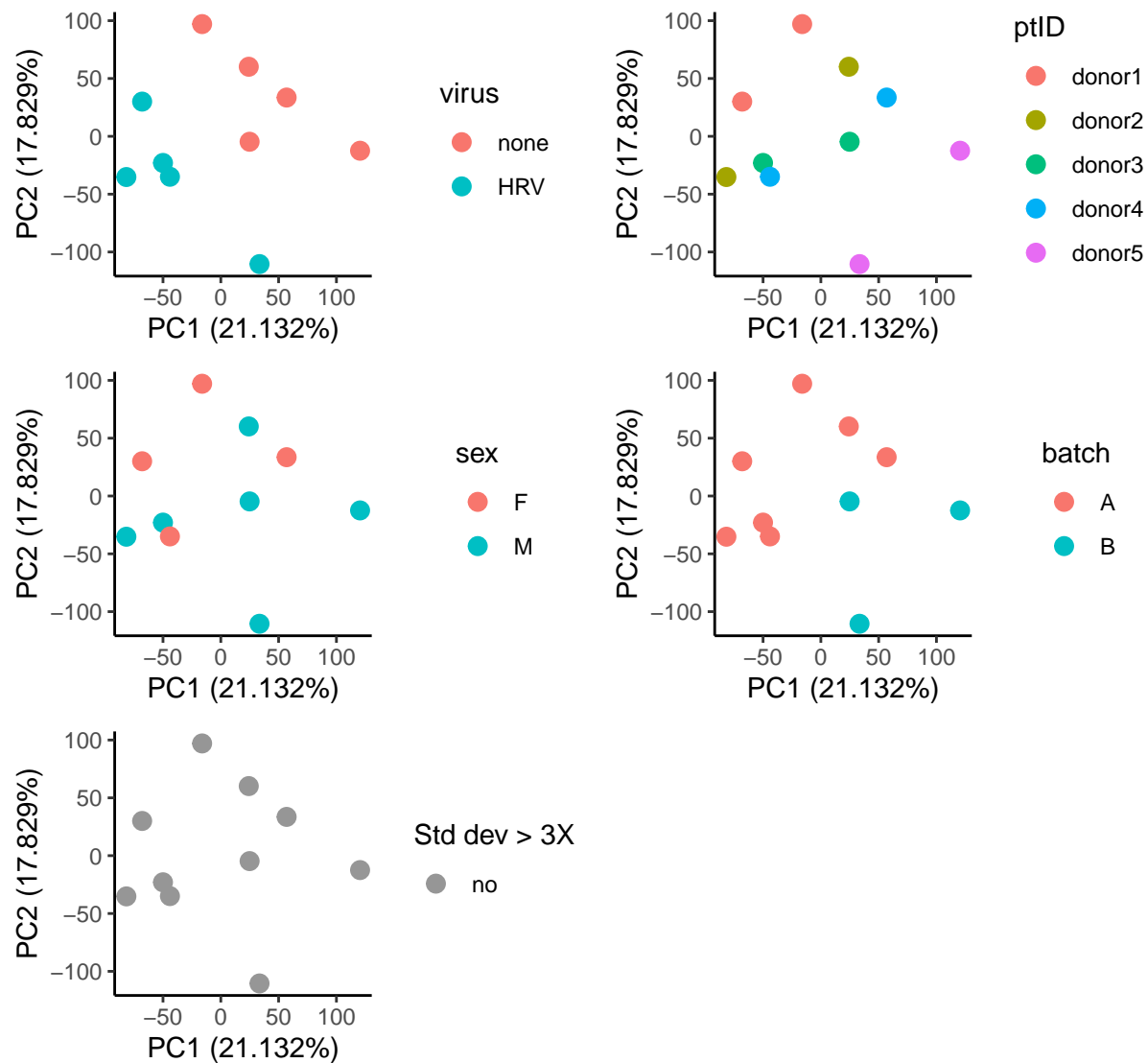
Save Write as `RData`

```r
save(dat.abund.norm, file = "data_clean/P259_dat.RData")
save(dat.abund.norm.voom, file = "data_clean/P259_voom.RData")
```

Write counts as `csv`

```r
as.data.frame(dat.abund.norm$counts) %>%
  rownames_to_column("ensembl_gene_id") %>%
  write_csv("data_clean/P259_counts.csv")

as.data.frame(dat.abund.norm.voom$E) %>%
  rownames_to_column("ensembl_gene_id") %>%
  write_csv("data_clean/P259_counts_voom.csv")
```

R session

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] DiagrammeR_1.0.8   edgeR_3.36.0      limma_3.50.3
##  [4] patchwork_1.1.1    ggrepel_0.9.1     sva_3.42.0
##  [7] BiocParallel_1.28.3 genefilter_1.76.0 mgcv_1.8-38
## [10] nlme_3.1-155       scales_1.2.0      RNAetc_0.1.0
## [13] BIGpicture_0.0.2   forcats_0.5.1     stringr_1.4.0
## [16] dplyr_1.0.8        purrr_0.3.4       readr_2.1.2
## [19] tidyr_1.2.0        tibble_3.1.6      ggplot2_3.3.5
## [22] tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##   [1] colorspace_2.0-3        ellipsis_0.3.2          XVector_0.34.0
##   [4] fs_1.5.2                rstudioapi_0.13         farver_2.1.0
##   [7] bit64_4.0.5             AnnotationDbi_1.56.2    fansi_1.0.3
##  [10] lubridate_1.8.0         xml2_1.3.3              codetools_0.2-18
##  [13] splines_4.1.2           cachem_1.0.6            knitr_1.39
##  [16] jsonlite_1.8.0          broom_0.8.0             annotate_1.72.0
##  [19] dbplyr_2.1.1            png_0.1-7               compiler_4.1.2
##  [22] httr_1.4.2              backports_1.4.1         assertthat_0.2.1
##  [25] Matrix_1.4-0            fastmap_1.1.0           cli_3.3.0
##  [28] prettyunits_1.1.1       visNetwork_2.1.0        htmltools_0.5.2
##  [31] tools_4.1.2             gtable_0.3.0            glue_1.6.2
##  [34] GenomeInfoDbData_1.2.7  rappdirs_0.3.3          Rcpp_1.0.8.3
##  [37] Biobase_2.54.0          cellranger_1.1.0        vctrs_0.4.1
##  [40] Biostrings_2.62.0       iterators_1.0.14        xfun_0.30
##  [43] ps_1.7.0                rvest_1.0.2             lifecycle_1.0.1
##  [46] XML_3.99-0.8            zlibbioc_1.40.0         vroom_1.5.7
##  [49] hms_1.1.1               parallel_4.1.2          RColorBrewer_1.1-3
##  [52] curl_4.3.2              yaml_2.3.5              memoise_2.0.1
##  [55] biomaRt_2.50.3          stringi_1.7.6           RSQLite_2.2.12
##  [58] highr_0.9               S4Vectors_0.32.4        foreach_1.5.2
##  [61] filelock_1.0.2          BiocGenerics_0.40.0     GenomeInfoDb_1.30.1
##  [64] rlang_1.0.2             pkgconfig_2.0.3         bitops_1.0-7
##  [67] matrixStats_0.62.0      evaluate_0.15           lattice_0.20-45
##  [70] htmlwidgets_1.5.4       labeling_0.4.2          bit_4.0.4
##  [73] tidyselect_1.1.2        processx_3.5.3          magrittr_2.0.3
##  [76] R6_2.5.1                IRanges_2.28.0          generics_0.1.2
```

```
##  [79] DBI_1.1.2              pillar_1.7.0           haven_2.4.3
##  [82] withr_2.5.0            survival_3.2-13        KEGGREST_1.34.0
##  [85] RCurl_1.98-1.6         modelr_0.1.8           crayon_1.5.1
##  [88] utf8_1.2.2             BiocFileCache_2.2.1    tzdb_0.3.0
##  [91] rmarkdown_2.11         progress_1.2.2         locfit_1.5-9.5
##  [94] grid_4.1.2             readxl_1.3.1           blob_1.2.3
##  [97] callr_3.7.0            reprex_2.0.1           digest_0.6.29
## [100] webshot_0.5.2          xtable_1.8-4           stats4_4.1.2
## [103] munsell_0.5.0
```