RNA-seq data cleaning

Counts to voom

Kim Dill-McFarland, kadm@uw.edu

version October 05, 2021

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 in the fastq to counts 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.

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

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

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

And load them into your current R session.

```
library(tidyverse)
```

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

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

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

```
library(BIGverse)
```

```
## -- Attaching packages ---------------------------------------- BIGverse 0.2.0 --
```

```
## v kimma      1.0.0     v BIGpicture 0.0.1
## v RNAetc     0.1.0
```

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

```
library(scales)
```

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

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

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

```
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 1.8-36. 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
```

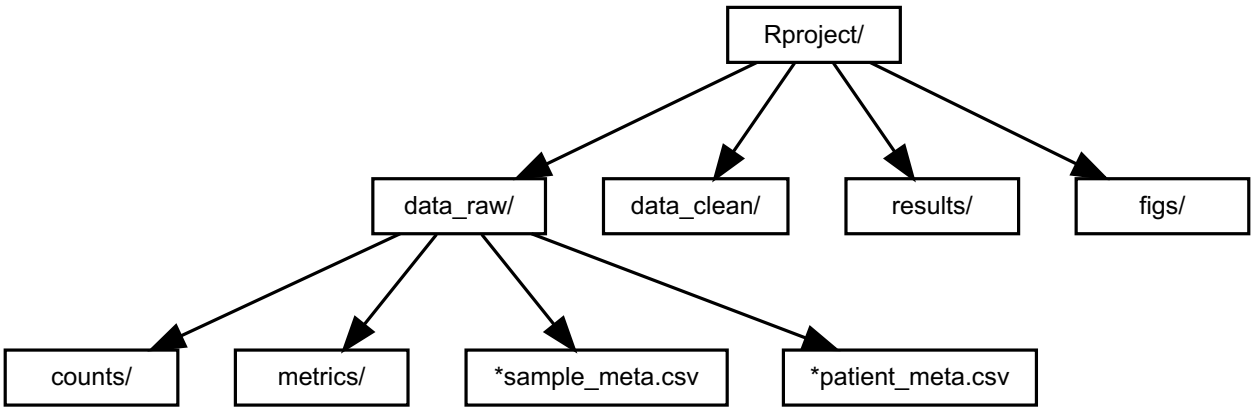
```
library(ggrepel)
library(patchwork)
```

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

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

```
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/` sub-directories should contain outputs from the fastq to counts pipeline including a counts table (`counts/`) and samtools `flagstat` and Picard tables (`metrics/`).

Example data

Example data were obtained from virus-stimulated human plasmacytoid dendritic cells. For full methods, see Dill-McFarland *et al.* 2021. Eosinophil-mediated suppression and Anti-IL-5 enhancement of plasmacytoid dendritic cell interferon responses in asthma. J Allergy Clin Immunol. In revision GitHub. Actual patient identifiers and metadata have been altered for this tutorial. Of note, these data were obtained from the same data set as the `kimma` package.

1. Format data

1.1: Alignment metrics

Our fastq pipeline outputs one `.tsv` per library per quality assessment. Thus, each library has three files in `data_raw/metrics/`: `*Aligned_flagstat.tsv`, `*filter_flagstat.tsv`, and `*Aligned_picard.tsv`. We use our custom function to extract and merge all of these files.

```
metric <- RNAetc::clean_metrics(dir = "data_raw/metrics/", flagstat = TRUE, picard = TRUE)
```

This results in a data frame with the following metrics for each sequencing library.

```
##  [1] "to.align"                          "secondary.align"
##  [3] "chimeric.align"                    "PCR.dups"
##  [5] "align"                             "paired"
##  [7] "R1.paired"                         "R2.paired"
##  [9] "properly paired "                  "both.align.paired"
## [11] "one.align.paired"                  "both.align.paired.diffCHR"
## [13] "both.align.paired.diffCHR.mapq"    "align.filtered"
## [15] "PF_BASES"                          "PF_ALIGNED_BASES"
## [17] "RIBOSOMAL_BASES"                   "CODING_BASES"
## [19] "UTR_BASES"                         "INTRONIC_BASES"
## [21] "INTERGENIC_BASES"                  "IGNORED_READS"
## [23] "CORRECT_STRAND_READS"              "INCORRECT_STRAND_READS"
## [25] "NUM_R1_TRANSCRIPT_STRAND_READS"    "NUM_R2_TRANSCRIPT_STRAND_READS"
## [27] "NUM_UNEXPLAINED_READS"             "PCT_R1_TRANSCRIPT_STRAND_READS"
## [29] "PCT_R2_TRANSCRIPT_STRAND_READS"    "PCT_RIBOSOMAL_BASES"
## [31] "PCT_CODING_BASES"                  "PCT_UTR_BASES"
## [33] "PCT_INTRONIC_BASES"                "PCT_INTERGENIC_BASES"
## [35] "PCT_MRNA_BASES"                    "PCT_USABLE_BASES"
## [37] "PCT_CORRECT_STRAND_READS"          "MEDIAN_CV_COVERAGE"
## [39] "MEDIAN_5PRIME_BIAS"                "MEDIAN_3PRIME_BIAS"
## [41] "MEDIAN_5PRIME_TO_3PRIME_BIAS"
```

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 table 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.
```

```
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.
```

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

This adds the following variable to the alignment metrics.

```
## # A tibble: 10 x 7
##    libID ptID   virus batch asthma    age sex
##    <chr> <chr>  <fct> <chr> <chr>   <dbl> <chr>
##  1 lib1  donor1 none  A     healthy    35 F
##  2 lib2  donor1 HRV   A     healthy    35 F
##  3 lib3  donor2 none  A     healthy    47 M
##  4 lib4  donor2 HRV   A     healthy    47 M
##  5 lib5  donor3 none  B     healthy    22 M
##  6 lib6  donor3 HRV   A     healthy    22 M
##  7 lib7  donor4 none  A     asthma     25 F
##  8 lib8  donor4 HRV   A     asthma     25 F
##  9 lib9  donor5 none  B     asthma     26 M
## 10 lib10 donor5 HRV   B     asthma     26 M
```

1.3: Counts table

Output by Subread `featureCounts`, the counts table needs some basic cleaning to become a numeric data frame with genes as rows and libraries as columns. We use a custom function in `RNAetc` to read in and format these data.

```
count <- RNAetc::read_fcounts("data_raw/counts/P259.featurecounts.tsv")

count
```

```
## # A tibble: 19,118 x 13
##    ensemblID    lib1  lib2  lib3  lib4  lib5  lib6  lib7  lib8  lib9 lib10 lib11
##    <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 ENSG000002~     0     0     0     0     0     0     0     0     0     0     0
##  2 ENSG000001~   240   126   595   180   332   186   521   202   294   170   362
##  3 ENSG000001~   185    50   255    37    99    50   329    64   392    74   395
##  4 ENSG000001~    62    52   101    81   138   113   166    94    80   132   185
##  5 ENSG000000~    17    56    45   106    46    85    92   132   104    76    43
##  6 ENSG000001~     0     0     0     0     3     0     2     4     0     0     0
##  7 ENSG000000~     0     8     8     0    34     8    26     0    23    21    31
##  8 ENSG000001~    43     9    74    22    35    14    83    37    96    50    85
```

```
##  9 ENSG000001~    15     6    83    29    49    22    64    25    41    30    51
## 10 ENSG000001~     6    15    34    15    40    25    77    36    60    50    50
## # ... with 19,108 more rows, and 1 more variable: lib12 <dbl>
```

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:

- Pass-filter sequences (`to.align`)
- Percent alignment (pass-filter alignments/sequences, `align.filtered`/`to.align`)
- Median coefficient of variation (CV) of coverage (`MEDIAN_CV_COVERAGE`)

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:

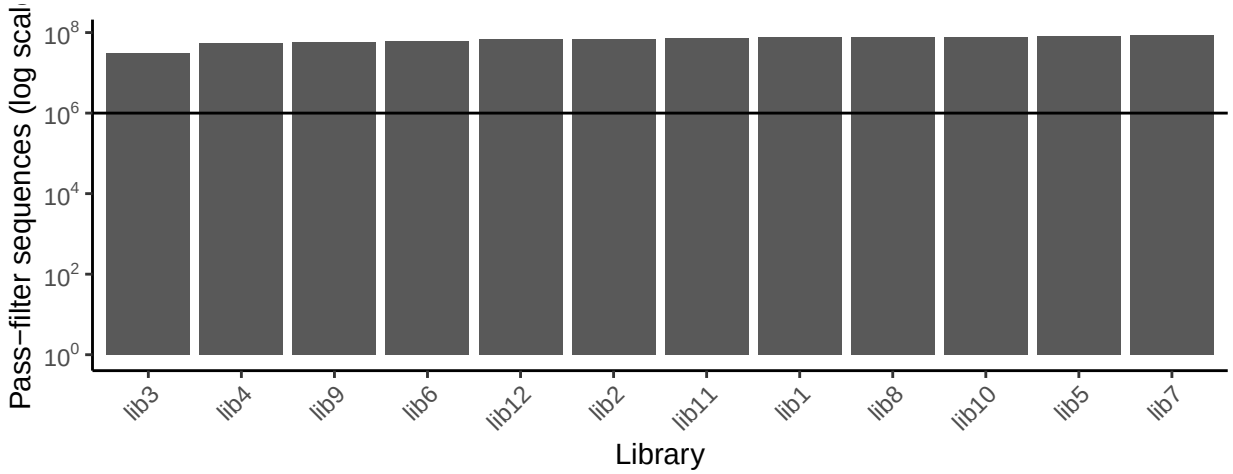
- sequences $<$ 1,000,000
- CV coverage $>$ 1
- alignment $<$ 75%

Set your cutoffs here.

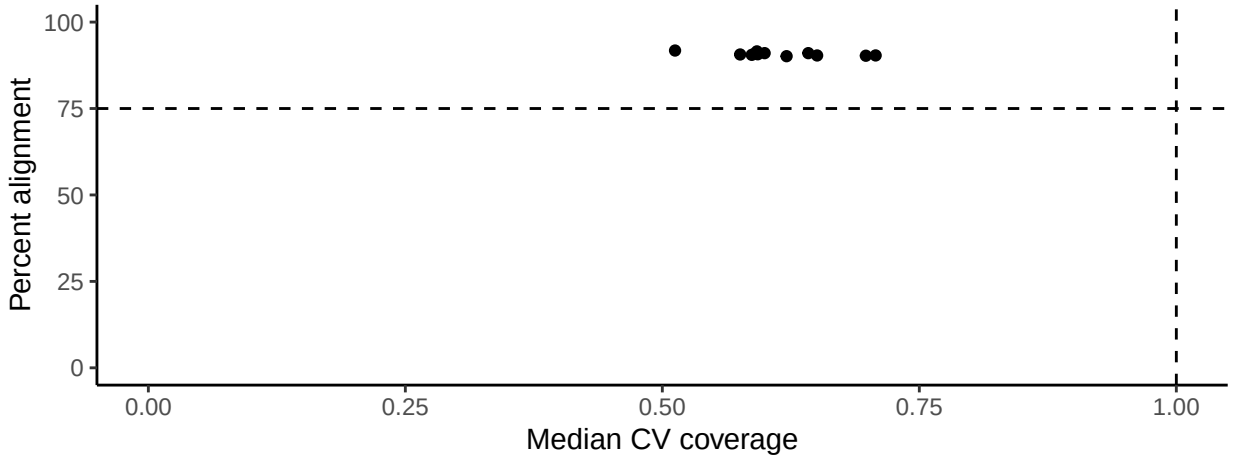
```
seq_cutoff <- 1E6
cv_cutoff <- 1
align_cutoff <- 75
```

For the example data, we see that we do not need to remove any libraries as all of them pass the above cutoffs.

```
ggplot(metric, aes(x = reorder(libID, to.align), y = to.align)) +
  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))
```



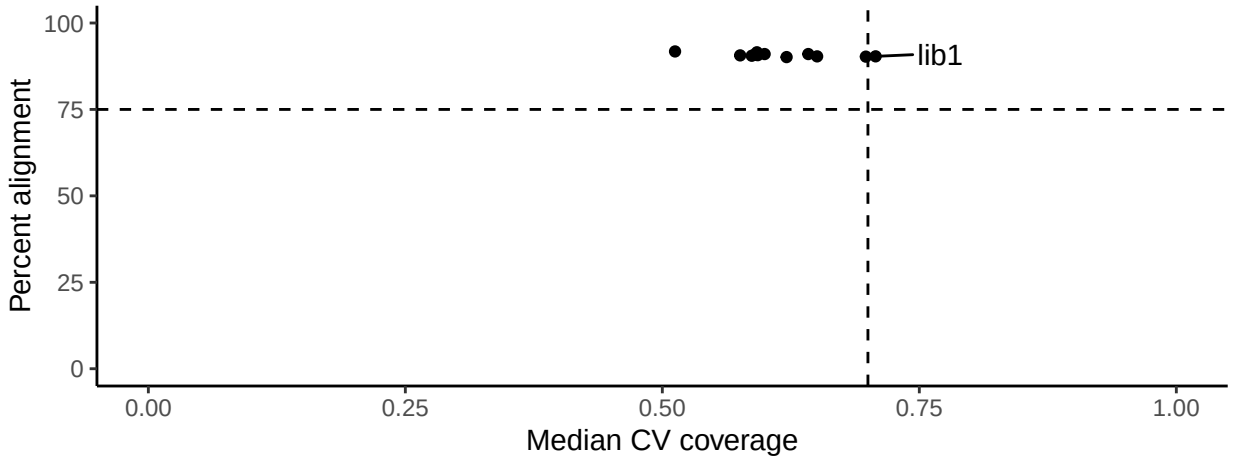

```
#Set CV max to the largest of 1 or max in dataset
CV_max <- max(1, max(metric$MEDIAN_CV_COVERAGE))

ggplot(metric, aes(x = MEDIAN_CV_COVERAGE, y = align.filtered/to.align*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") +
  #Beautify
  theme_classic() +
  labs(x = "Median CV coverage", y="Percent alignment")
```



If we set harsher cutoffs such as CV coverage > 0.7, 1 library would fail as shown here. Note the use of `geom_text_repel` to label outlying libraries.

```
cv_cutoff2 <- 0.7

ggplot(metric, aes(x = MEDIAN_CV_COVERAGE, y = align.filtered/to.align*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_cutoff2, lty="dashed") +
  #Label points outside cutoffs
  geom_text_repel(data=filter(metric, MEDIAN_CV_COVERAGE > cv_cutoff2 |
                                align.filtered/to.align*100 < align_cutoff),
            aes(label=libID), show.legend = FALSE, max.overlaps = Inf,
            box.padding = 1) +
  #Beautify
  theme_classic() +
  labs(x = "Median CV coverage", y="Percent alignment")
```



However, these stricter cutoffs are not necessary for these data. If libraries needed to be removed, you can use the following code to do so.

```
meta.filter <- meta %>%
  filter(MEDIAN_CV_COVERAGE < cv_cutoff & to.align > seq_cutoff &
           align.filtered/to.align*100 > align_cutoff)

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

Here, this removes 0 libraries.

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$ensemblID) %>%
  #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(ensemblID %in% key.filter$ensembl_gene_id)
```

This removes 90 genes from this data set. These data were somewhat pre-filtered so you will likely see many more genes removed in this step. In the current human genome (GRCh38 release 104), there are 23079 protein-coding genes, of which 19028 were found in this data set.

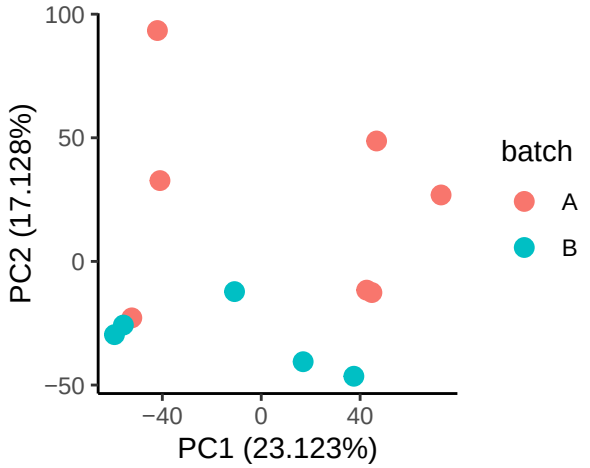
2.3: Correct batch effects

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)
```

```
## $batch
```



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.

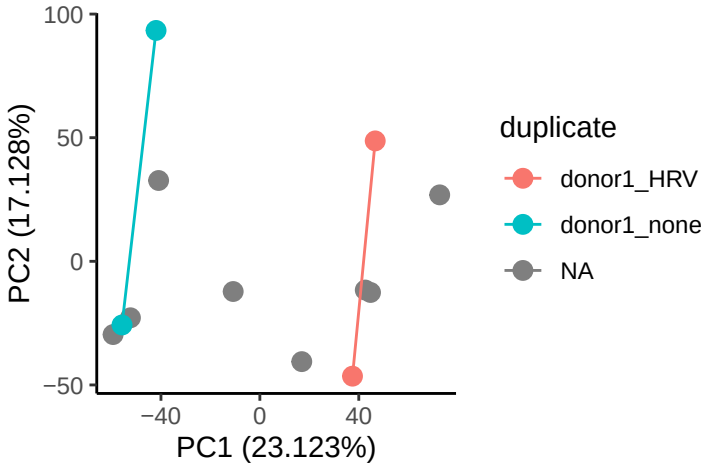
```
unique.ID <- c("ptID", "virus")

#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.
```

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

```
## $duplicate
```



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

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, though a real analysis should be run on a larger subset. We recommend anywhere from 10,000 to all genes in the data.

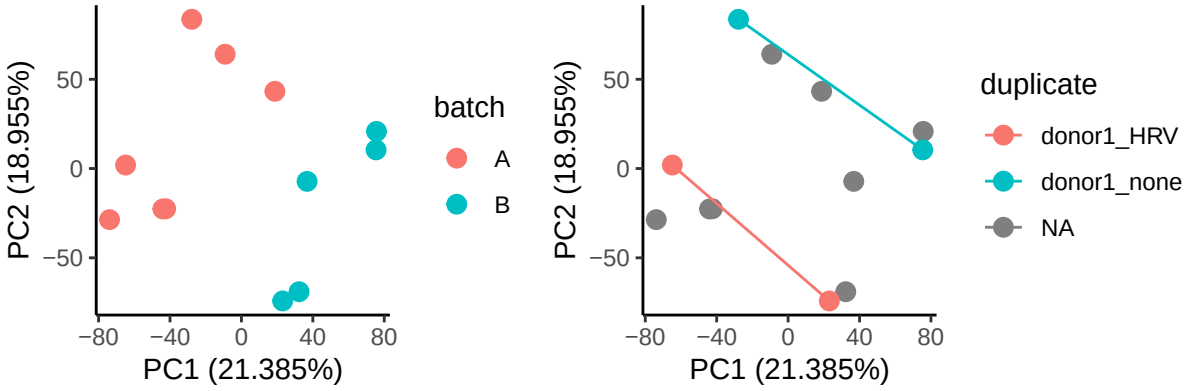
```
count.filter.pc.combat <- count.filter.pc %>%
  #Convert count df to matrix
  column_to_rownames("ensemblID") %>%
  as.matrix() %>%
  #Batch correction
  sva::ComBat_seq(., batch = meta.filter$batch,
                  group = meta.filter$virus,
                  covar_mod = model.matrix(~ sex, meta.filter),
              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
```

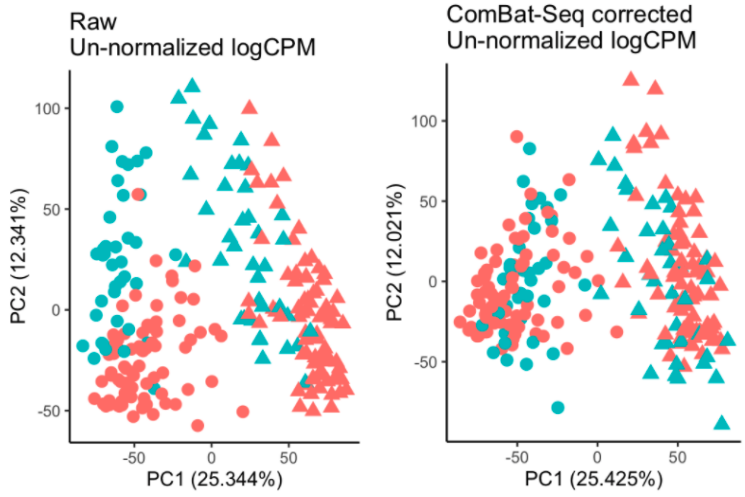
Note that the total "co-variates" listed in the above message includes non-reference levels of the main effects (HRV) plus variables listed as `covar` (sex).

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 some 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)
```



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:

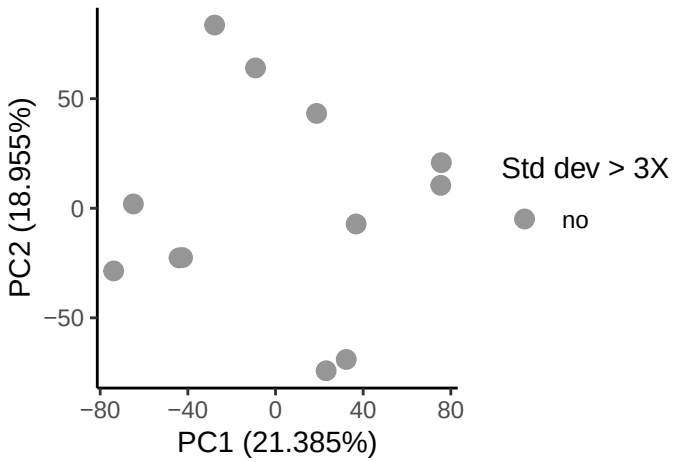
- How strong are the batch effects? This can be roughly estimated based on the PC to which the batch variable best aligns. In the above data, this is PC2 which explains a substantial percent of variation (17%). Thus, the batch effects are also likely substantial. If the batch PC explains only a small amount of variation, you may wish to skip batch correction as the effects were minimal.
- 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)
pca3
```

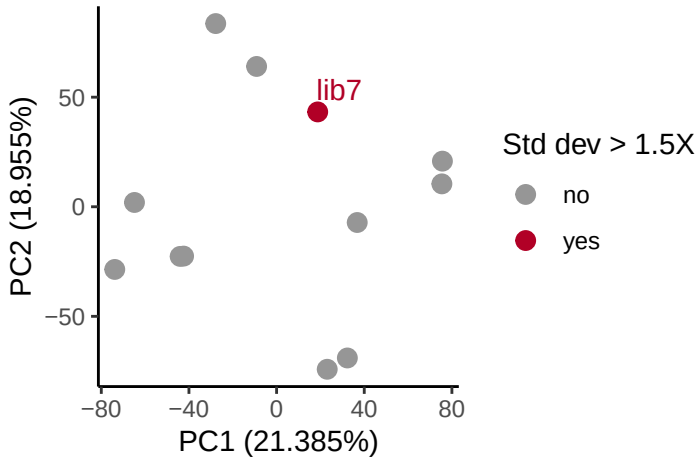
```
## $outlier
```



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 are two outliers 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")
```

```
## $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.

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 = to.align)

#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.

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.

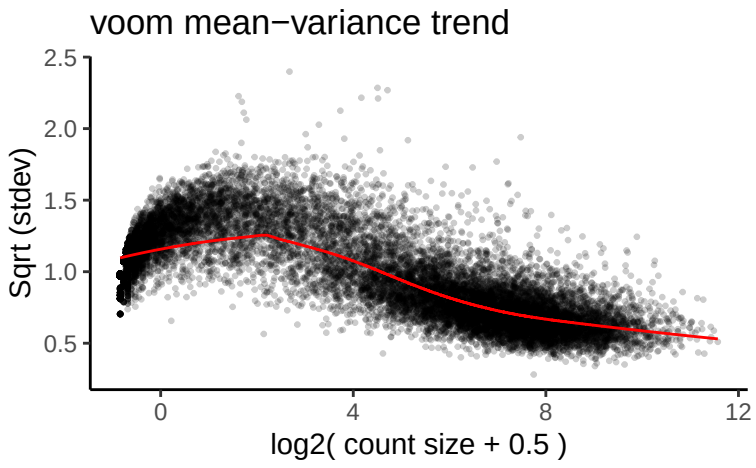
```
dat <- DGEList(
  #count table
  # Note that if you did not do batch correction, you may need to
  # move gene names from a variable in the df to rownames
  counts=as.matrix(count.filter.pc.combat.out.dedup.ord),
  #metadata
  samples=meta.filter.out.dedup.ord,
  #gene info
  genes=key.filter)
```

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 - see plot below)

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

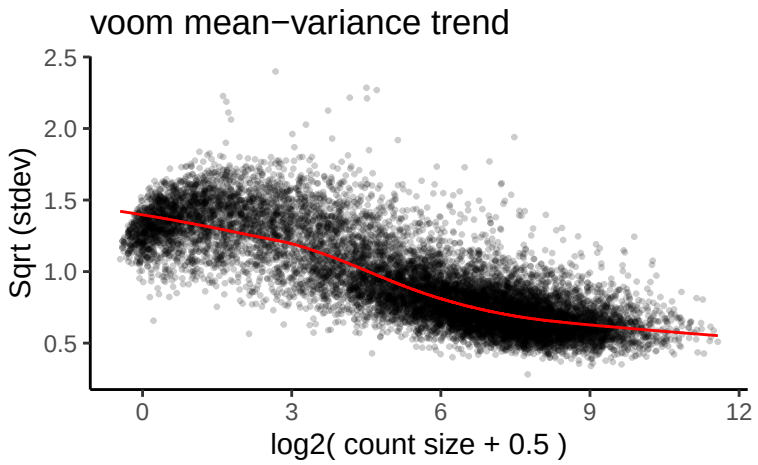


Our goal is to remove genes in the lower left where counts (x) and variance (y) are low *e.g.* where genes break the mean variance trend. We use our custom function to retain only genes that are at least `min.CPM` counts per million in at least `min.sample` number of samples OR in at least `min.pct` percent of samples. Here, we use 0.5 CPM in at least 3 samples.

```
dat.abund <- RNAetc::filter_rare(dat, min.CPM = 0.5, min.sample = 3,
                                 gene.var="ensembl_gene_id")
```

Plotting the filtered data, we see the trend line (red) is roughly linear and the lower left tail is mostly removed.

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



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. This results in the removal of around 25% of genes. Here, our filtering is on the stricter side, removing 6198 or 33% 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: 6,198 x 7
##    ensembl_gene_id gene_biotype   hgnc_symbol mean.CPM min.CPM max.CPM
##    <chr>           <chr>          <list>         <dbl>   <dbl>   <dbl>
```

```
##  1 ENSG00000000003 protein_coding <chr [1]>     0.309        0   1.83
##  2 ENSG00000000005 protein_coding <chr [1]>     0            0   0
##  3 ENSG00000001617 protein_coding <chr [1]>     0.169        0   0.913
##  4 ENSG00000001626 protein_coding <chr [1]>     0.0775       0   0.775
##  5 ENSG00000002745 protein_coding <chr [1]>     0.0375       0   0.375
##  6 ENSG00000002746 protein_coding <chr [1]>     0            0   0
##  7 ENSG00000003137 protein_coding <chr [1]>     0.0372       0   0.372
##  8 ENSG00000003249 protein_coding <chr [1]>     0            0   0
##  9 ENSG00000003989 protein_coding <chr [1]>     0            0   0
## 10 ENSG00000004776 protein_coding <chr [1]>     0            0   0
## # ... with 6,188 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.

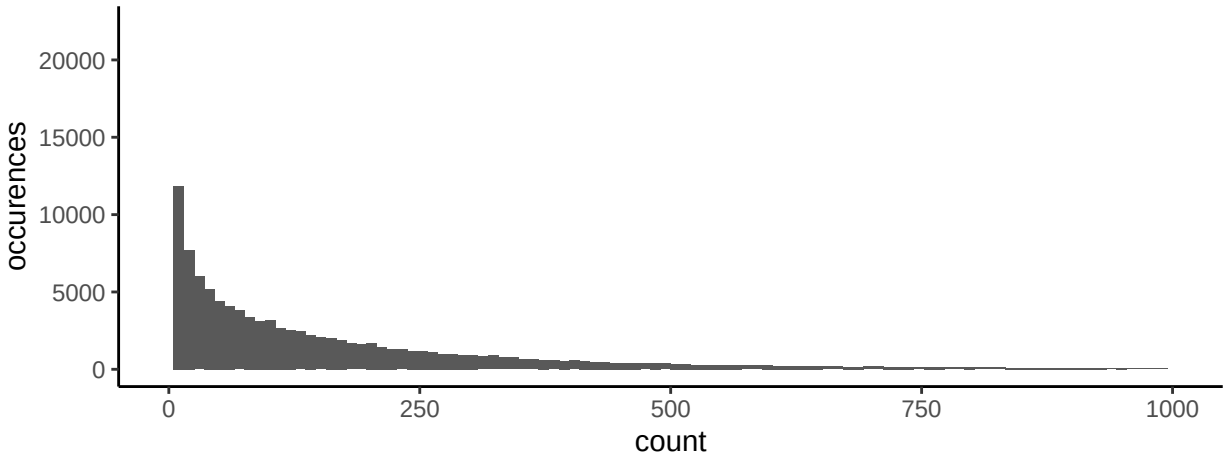
The calculated scaling factors are then applied to the counts table automatically in the `voom` step.

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

3.2: voom aka log2 counts per million (CPM)

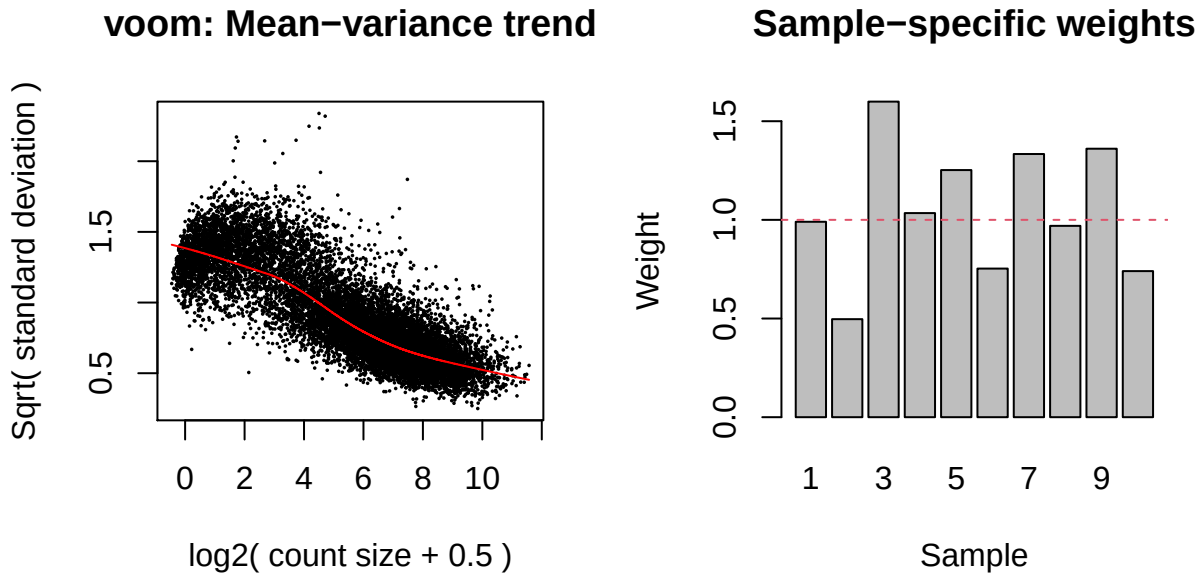
We continue normalization by converting counts to CPM within each sample, thus accounting for differential sampling depth. We also perform log2 transformation, because RNA-seq data are heavily right-skewed and thus, violate assumptions of normality.

```
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)
```



4. Summary and save

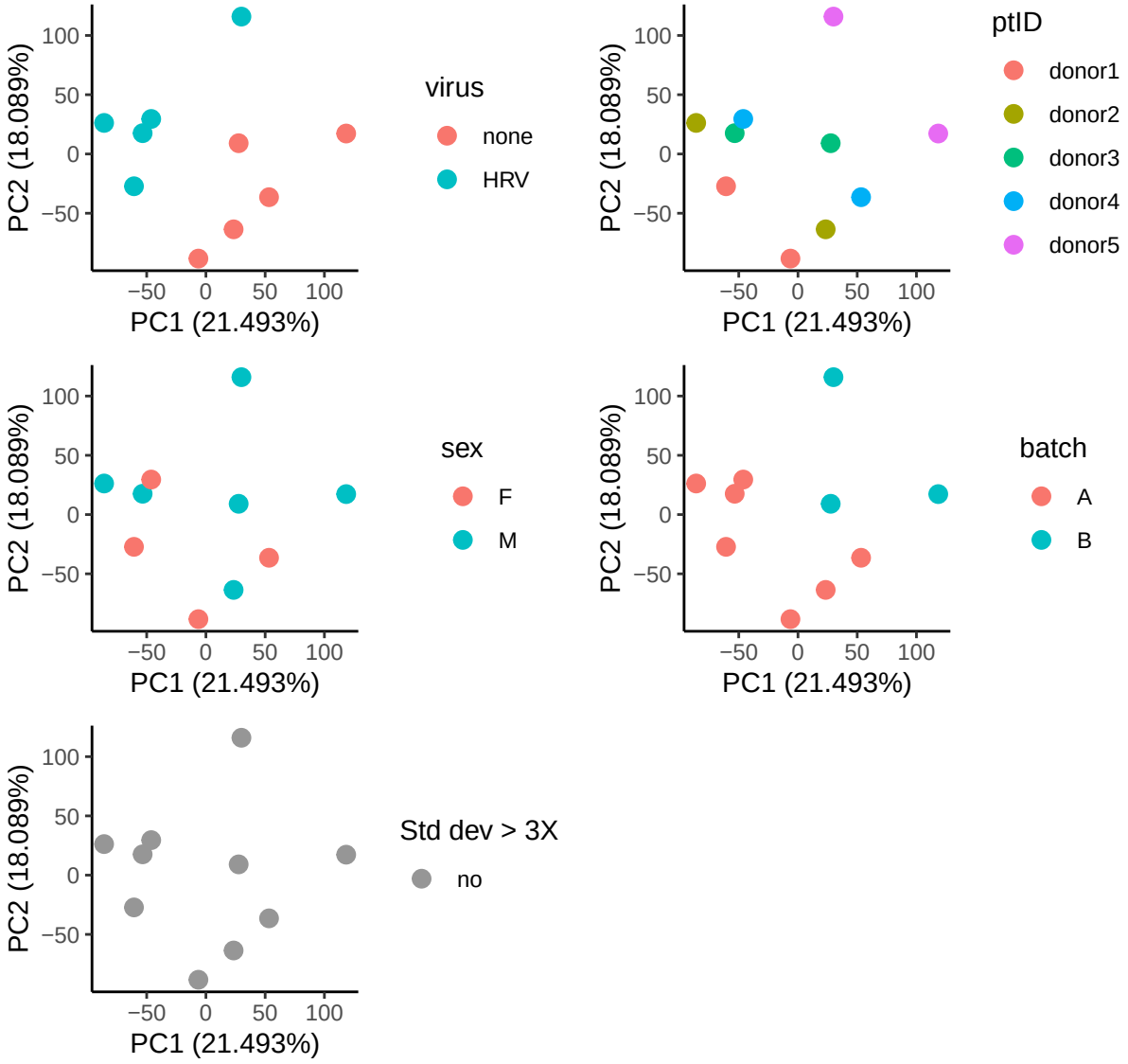
Filtering The following summarizes libraries removed from analysis. In this tutorial, all libraries pass-filter and move on to analysis.

```
## # 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 pass-filter
## 12 lib12 pass-filter
```

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)
```



Save Write as `RData`

```
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`

```
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 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/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_1.0.6.1  edgeR_3.32.1        limma_3.46.0
##  [4] patchwork_1.1.1     ggrepel_0.9.1       sva_3.36.0
##  [7] BiocParallel_1.24.1 genefilter_1.70.0   mgcv_1.8-36
## [10] nlme_3.1-152        scales_1.1.1        BIGpicture_0.0.1
## [13] RNAetc_0.1.0        kimma_1.0.0         BIGverse_0.2.0
## [16] forcats_0.5.1       stringr_1.4.0       dplyr_1.0.7
## [19] purrr_0.3.4         readr_2.0.2         tidyr_1.1.4
## [22] tibble_3.1.5        ggplot2_3.3.5       tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] colorspace_2.0-2     ellipsis_0.3.2       fs_1.5.0
##  [4] rstudioapi_0.13      farver_2.1.0         bit64_4.0.5
##  [7] AnnotationDbi_1.52.0 fansi_0.5.0          lubridate_1.7.10
## [10] xml2_1.3.2           codetools_0.2-18     splines_4.0.2
## [13] cachem_1.0.6         knitr_1.36           jsonlite_1.7.2
## [16] broom_0.7.9          annotate_1.66.0      dbplyr_2.1.1
## [19] compiler_4.0.2       httr_1.4.2           backports_1.2.1
## [22] assertthat_0.2.1     Matrix_1.3-4         fastmap_1.1.0
## [25] cli_3.0.1            visNetwork_2.0.9     htmltools_0.5.2
## [28] prettyunits_1.1.1    tools_4.0.2          gtable_0.3.0
## [31] glue_1.4.2           rappdirs_0.3.3       Rcpp_1.0.7
## [34] Biobase_2.50.0       cellranger_1.1.0     vctrs_0.3.8
## [37] iterators_1.0.13     xfun_0.26            ps_1.6.0
## [40] rvest_1.0.1          lifecycle_1.0.1      XML_3.99-0.6
```

```
## [43] vroom_1.5.5          hms_1.1.1            parallel_4.0.2
## [46] RColorBrewer_1.1-2   curl_4.3.2           yaml_2.2.1
## [49] memoise_2.0.0        biomaRt_2.44.4       stringi_1.7.4
## [52] RSQLite_2.2.8        highr_0.9            S4Vectors_0.28.1
## [55] foreach_1.5.1        BiocGenerics_0.36.1  rlang_0.4.11
## [58] pkgconfig_2.0.3      bitops_1.0-7         matrixStats_0.61.0
## [61] evaluate_0.14        lattice_0.20-44      htmlwidgets_1.5.4
## [64] labeling_0.4.2       bit_4.0.4            tidyselect_1.1.1
## [67] processx_3.5.2       magrittr_2.0.1       R6_2.5.1
## [70] IRanges_2.24.1       generics_0.1.0       DBI_1.1.1
## [73] pillar_1.6.3         haven_2.4.3          withr_2.4.2
## [76] survival_3.2-11      RCurl_1.98-1.3       modelr_0.1.8
## [79] crayon_1.4.1         utf8_1.2.2           BiocFileCache_1.12.1
## [82] tzdb_0.1.2           rmarkdown_2.9        progress_1.2.2
## [85] locfit_1.5-9.4       grid_4.0.2           readxl_1.3.1
## [88] blob_1.2.2           callr_3.7.0          reprex_2.0.0
## [91] digest_0.6.28        webshot_0.5.2        xtable_1.8-4
## [94] openssl_1.4.4        stats4_4.0.2         munsell_0.5.0
## [97] askpass_1.1
```