Full_Workflow

Jacob Westaway

Last updated on 2021-03-17

Contents

1 About.

This document contains the workflow for the manuscript *BLANK_BLANK*, and is based on the workflow from the paper Characterising the bacterial gut microbiome of probiotic-supplmented very-preterm infants,and includes the bioinformatics pipeline to go from raw reads to interpretable abundances, based largely around this DADA2 workflow developed by *Callahan, et al.*, with removal of contamination with MicroDecon, and the analysis using a combination of the packages phyloseq, DESeq2, lme4 and more.

2 Bioinformatics Pipeline.

2.1 About.

Creating an ASV table from raw reads, using DADA2.

2.2 Load required packages.

```
sapply(c("dada2", "phyloseq", "DECIPHER", "phangorn", "BiocManager", "BiocStyle",
        "Biostrings", "ShortRead", "ggplot2", "gridExtra", "tibble", "tidyverse"),
        require, character.only = TRUE)
```

2.3 Read quality.

2.3.1 Organise forward and reverse fastq filenames into own lists (check file format).

- First define the file path to the directory containing the fastq files (we will use this several times).

```
path <-"Data"

fnFs <- sort(list.files(path, pattern = "_R1_001.fastq.gz", full.names = TRUE))

fnRs <- sort(list.files(path, pattern = "_R2_001.fastq.gz", full.names = TRUE))
```

2.3.2 Extract sample names.

```
sample.names <- sapply(strsplit(basename(fnFs), "_"), `[`, 1)
```

2.3.3 Check quality of Forward and Reverse Reads (used to define truncLen in filtering).

```
plotQualityProfile(fnFs[1:2])
```

```
plotQualityProfile(fnRs[1:2])
```

2.3.4 Assign names for filtered reads.

```
filtFs <- file.path(path, "filtered", paste0(sample.names, "_F_filt.fastq.gz"))

filtRs <- file.path(path, "filtered", paste0(sample.names, "_R_filt.fastq.gz"))
```

2.3.5 Filter and trim the reads.

- Paremeters based on data and quality plots.
- `truncLean` defined by when quality plots begin to drop off, but ensuring it is large enough to maintain read overlap (=>20bp) downstream.
- `trimLeft` is not needed as primers/barcodes already removed.
- `maxEE = c(2,2)` is for filtering, where the higher the value the more relaxed filtering,allowing more reads to get through.
- Good quality data should allow for more stringent parameters (2 is stringent).
- The number of reads filtered is checked. If reads are too low, can alter parameters.

```
out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs, truncLen = c(280,200),
                     trimLeft = c(16,21),
                     maxN = 0,
                     maxEE = c(2,2),
                     truncQ = 2,
                     rm.phix = TRUE,
                     compress = TRUE,
                     multithread = FALSE)# windows can't support multithread
head(out)
```

2.4 Infer sequence variants.

2.4.1 Calculate Error Rates.

- Error rates are used for sample ineference downstream.

```
errF <- learnErrors(filtFs, multithread = TRUE)

errR <- learnErrors(filtRs, multithread = TRUE)
```

2.4.2 Plot error rates.

- Estimated error rates (black line) should be a good fit to observed rates (points) and error should decrease.

```
plotErrors(errF, nominalQ = TRUE)
```

```
plotErrors(errR, nominalQ = TRUE)
```

2.4.3 Dereplication.

- Combine indentical sequences into unique sequence bins.
- Name the derep-class objects by the sample name.

```
derepFs <- derepFastq(filtFs, verbose = TRUE)

derepRs <- derepFastq(filtRs, verbose = TRUE)

names(derepFs) <- sample.names

names(derepRs) <- sample.names
```

2.4.4 Sequence Inference.

```
dadaFs <- dada(derepFs, err = errF, multithread = F)

dadaRs <- dada(derepRs, err = errR, multithread = F)
```

2.4.5 Inspect denoised data.

```
dadaFs[[1]]

dadaRs[[1]]
```

2.4.6 Merge Paired Reads and inspect merged data.

- Removes paired reads that do not perfectly overlap.
- Arguments represent infered samples AND denoised reads.

```
mergers <- mergePairs(dadaFs, derepFs, dadaRs, derepRs, verbose = TRUE)
```

2.5 Construct amplicon sequence variance (ASV) table and remove chimeras.

2.5.1 Construct ASV table.

- Check dimentions and inspect distribution of sequence lengths.

```
seqtab <- makeSequenceTable(mergers)

dim(seqtab)

seqtab %>%
  getSequences() %>%
  nchar() %>%
  table()

# SHOULD GET THE SAME OUTPUT AS: table(nchar(getSequences(seqtab)))
```

2.5.2 Remove chimeras.

```
seqtab.nochim <- removeBimeraDenovo(seqtab, method = "consensus",
                                    multithread = TRUE, verbose = TRUE)

rm(seqtab)
```

2.5.3 Track reads through pipeline.

```
getN <- function(x) sum(getUniques(x))

track <- cbind(out, sapply(dadaFs, getN), sapply(dadaRs, getN),
               sapply(mergers, getN), rowSums(seqtab.nochim))

colnames(track) <- c("input", "filtered", "denoisedF", "denoisedR", "merged", "nonchim")

rownames(track) <- sample.names

head(track)
rm(track)
```

2.6 Contamination removal with *MicroDecon*.

```
library(microDecon)
```

2.6.1 Read in metadata (needed for MicroDecon)

```
Metadata  <- readxl::read_excel("Data/New_metadata.xlsx")
```

2.6.2 Reformat data for *MicroDecon*.

- **REFORMAT FUNCTION** for your data.
- Transpose sequencing table (post chimera removal) and convert to a dataframe.

- Reorder sequencing table by a prior grouping (days).
- Move blank sample columns to the start of the sequencing table.
- Turn row names into their own column as *MicroDecon* requires that the OTUs have a unique ID in column 1.

```
wrangle_microdecon <- function(seqtab.nochim){

# transpose data
microdecon.df <- t(seqtab.nochim) %>%
  as.data.frame()

# a prior grouping
  Metadata_microdecon <- Metadata %>%
  arrange(Date) %>%
  select(Sample, Date) %>% # select key columns
  mutate(Sample = paste0("AM", Sample)) # add AM into cell values to be the same as the count table

microdecon.df <- microdecon.df %>%
  relocate(any_of(Metadata_microdecon$Sample)) # rearrange the columns by the ordered metadata

# blanks to first columns
Metadata_microdecon <- Metadata %>%
  filter(Type == "negative control") %>%  # filter for negative controls
  select(Sample, Type, Date) %>%
  mutate(Sample = paste0("AM", Sample))

microdecon.df <- microdecon.df %>%
  relocate(any_of(Metadata_microdecon$Sample)) %>%
  tibble::rownames_to_column(var = "ID") # turn the rownames into the first column
}

microdecon.df <- wrangle_microdecon(seqtab.nochim)

rm(seqtab.nochim)
```

2.6.3 Decontaminate data using `decon()`.

- get the counts for each of the a priori groups for `numd.ind`, not including the blanks.

```
numb_ind_vector <- Metadata %>%
  filter(Type == "Microbiome") %>% # remove blanks
  group_by(Date) %>%
  summarise(n()) %>% # get the counts for each date.
  select("n()") %>%
  add_row("n()" = 35) %>% # 26 is the number of samples there are no metadata for.
  rename("n" = "n()")
```

- `(ncol(microdecon.df) - 26) - (nrow(Metadata))` should be equal to 1 (ID column).
- `numb.ind` is the number of columns for each priori grouping.
- `taxa = F` as there is no taxonomy in the dataframe.

```
decontaminated <- decon(data = microdecon.df, numb.blanks = 7,
                        numb.ind = numb_ind_vector$n, taxa = F)

rm(microdecon.df)
rm(numb_ind_vector)
```

2.6.3.1 Check *MicroDecon* Outputs.

```
decontaminated$decon.table
decontaminated$reads.removed
decontaminated$OTUs.removed
decontaminated$mean.per.group
decontaminated$sum.per.group
```

2.6.4 Reformat decon.table.

- Convert column 1 to row names.
- Remove blank average column (1).
- Save rownames as seperate vector to be added back, as row names are removed during apply().
- Convert numeric values to integers (for downstream analysis).
- Transpose data.

```
seqtab.microdecon <- decontaminated$decon.table %>%
  remove_rownames() %>%
  column_to_rownames(var = "ID") %>%
  select(-1) %>% # remove mean blank
  as.matrix() %>%
  t()

rm(decontaminated)
```

2.6.5 Merging multiple sequence runs.

2.7 Assign taxonomy.

- With optional species addition (there is an agglomeration step downstream, so you can add species now for curiosities sake, and remove later for analysis).

```
taxa <- assignTaxonomy(seqtab.microdecon, "~/Aquaculture_Microbiome/SILVA/silva_nr_v132_train_set.fa.gz

taxa.print <- taxa # Removes sequence rownames for display only
rownames(taxa.print) <- NULL
```

3 Need to run species assignment on hpc

```
write.csv(taxa, "taxa.csv")

taxa <- addSpecies(taxa, "~/Aquaculture_Microbiome/SILVA/silva_species_assignment_v132.fa.gz")
```

3.0.1 Calculate percentage of NA taxa

```
sum(is.na(taxa))/prod(dim(taxa)) * 100

apply(taxa, 2, function(col)sum(is.na(col))/length(col)) * 100
```

4 Preprocessing: Creating a Phyloseq Object.

4.1 About.

Creating a phyloseq object to be used for analysis, and create different objects to be used for different types of analysis downstream.

4.2 Load required packages.

```
sapply(c("caret", "pls", "e1071", "ggplot2",
    "randomForest", "tidyverse", "ggrepel", "nlme", "devtools",
    "reshape2", "PMA", "structSSI", "ade4","ggnetwork",
    "intergraph", "scales", "readxl", "genefilter", "impute",
    "phyloseq", "phangorn", "dada2", "DECIPHER", "gridExtra", "stringi", "janitor"),
    require, character.only = TRUE)
```

4.3 Import metadata.

```
Metadata <- readxl::read_excel("Data/New_metadata.xlsx") %>%
  select(-c(3, 18, 21)) %>%
  add_column("Primary_Group" = "SCN", "Type" = "Discharge") %>%
  rbind(
    readxl::read_excel("Data/Old_metadata.xlsx") %>%
      separate(DOB, into = c("DOB", "Time"), sep = "\\s") %>%
      mutate(DOB = as.Date(DOB)) %>%
      select(1, 3:4, 9, 17:18, 20:35))
```

4.4 Constuct a phylogenetic tree (for Phyloseq object downstream, required for distance measures).

- Peform multiple-allignment.
- `pml` calculates the likelihood of a given tree, and then `optim.pml()` optimizes the tree topology and branch length for the selected model (GTR+G+I max tree).

```
build_tree <- function(ASV_table){

seqs <- getSequences(ASV_table)

names(seqs) <- seqs
```

```
alignment <- AlignSeqs(DNAStringSet(seqs), anchor=NA, verbose=FALSE)

phangAlign <- phyDat(as(alignment, "matrix"), type = "DNA")

fitGTR <- phangAlign %>%
  dist.ml() %>%
  NJ() %>%
  pml(data = phangAlign) %>%
  update(k = 4, inv = 0.2) %>%
  optim.pml(model = "GTR", optInv = TRUE, optGamma = TRUE,
  rearrangement = "NNI", control = pml.control(trace = 0))

detach("package:phangorn", unload = TRUE) # conflicts downstream

return(fitGTR)
}

fitGTR <- build_tree(seqtab.microdecon)
```

4.5 Constrcut the Phyloseq object.

- Includes: metadata, ASV table, taxonomy table and phylogenetic tree.

```
ps <- phyloseq(otu_table(seqtab.microdecon, taxa_are_rows=FALSE),
               sample_data(Metadata),
               tax_table(taxa),
               phy_tree(fitGTR$tree))
```

4.6 Wrangling the metadata.

- And do some additional wrangling.
- Convert chraracters to factors.

Change `ID`.

```
sample_data(ps) <- sample_data(ps) %>%
  unclass() %>%
  as.data.frame() %>%
  mutate_if(is.character, as.factor) %>%
  mutate("Sample" = ID) %>% # need to redo the rownames to save it back into the original ps object
  column_to_rownames("Sample")
```

4.6.1 Subset phyloseq object for data to be analyzed.

Change `Type` and `Discharge`.

```
ps <- subset_samples(ps, Type == "Discharge")
```

4.7 Filtering and normalisation.

4.7.1 Taxonomy filtering.

- Can check the number of phyla before and after transformation with `table(tax_table(ps)[, "Phylum"], exclude = NULL)`.
- Remove features with ambiguous and NA phylum annotation.

```
ps1 <- subset_taxa(ps, !is.na(Phylum) & !Phylum %in% c("", "uncharacterized"))
```

4.7.1.1 Check percentages of NA values left.

```
sum(is.na(tax_table(ps1)))/prod(dim(tax_table(ps1))) * 100

apply(tax_table(ps1), 2, function(col)sum(is.na(col))/length(col)) * 100
```

4.7.2 Prevelance filtering.

- Using an unsupervised method (relying on the data in this experiment) explore the prevelance of features in the dataset.
- Calculate the prevalence of each feature and store as a dataframe.
- Add taxonomy and total read counts.

```
prevdf = apply(X = otu_table(ps1),
               MARGIN = ifelse(taxa_are_rows(ps1), yes = 1, no = 2),
               FUN = function(x){sum(x > 0)})

prevdf = data.frame(Prevalence = prevdf,
                    TotalAbundance = taxa_sums(ps1),
                    tax_table(ps1))
```

- Plot the relationship between prevelance and total read count for each feature. This provides information on outliers and ranges of features.

```
prevdf %>%
  subset(Phylum %in% get_taxa_unique(ps1, "Phylum")) %>%
  ggplot(aes(TotalAbundance, Prevalence / nsamples(ps1),color=Phylum)) +
  geom_hline(yintercept = 0.05, alpha = 0.5, linetype = 1) +
  geom_point(size = 2, alpha = 0.7) +
  scale_x_log10() +
  xlab("Total Abundance") + ylab("Prevalence [Frac. Samples]") +
  facet_wrap(~Phylum) + theme(legend.position="none")
```

- Define prevalence threshold based on the plot (~1% is standard) and apply to ps object (if prevelance is too low don't designate a threshold).

```
prevalenceThreshold = 0.01 * nsamples(ps1)

keepTaxa = rownames(prevdf)[(prevdf$Prevalence >= prevalenceThreshold)]

ps2 = prune_taxa(keepTaxa, ps1)
```

- Explore the relationship on the filtered data set.

```
prevdf %>%
  subset(Phylum %in% get_taxa_unique(ps2, "Phylum")) %>%
  ggplot(aes(TotalAbundance, Prevalence / nsamples(ps2),color=Phylum)) +
  geom_hline(yintercept = 0.05, alpha = 0.5, linetype = 1) +
  geom_point(size = 2, alpha = 0.7) +
  scale_x_log10() +
  xlab("Total Abundance") + ylab("Prevalence [Frac. Samples]") +
  facet_wrap(~Phylum) + theme(legend.position="none")
```

4.7.3 Aggolmerate taxa.

- Combine features that descend from the same genus as most species have not been identified due to the poor sequencing depth in 16S.
- Can check how many genera would be present after filtering by running `length(get_taxa_unique(ps2, taxonomic.rank = "Genus"))`, and `ntaxa(ps3)` will give the number of post agglomeration taxa.

```
ps3 = tax_glom(ps2, "Genus", NArm = TRUE)
```

- Create tree plots to observe pre and post agglomeration.

```
grid.arrange(nrow = 1,
    (plot_tree(ps2, method = "treeonly",
      ladderize = "left", title = "Before Agglomeration") +
      theme(plot.title = element_text(size = 15))),
    (plot_tree(ps3, method = "treeonly",
      ladderize = "left", title = "Post Genus Agglomeration") +
      theme(plot.title = element_text(size = 15))))
```

4.7.4 Normalisation.

- Plot a refraction curve to see if total sum scaling will surfice.
- Define colours and lines.
- Step = step size for sample sizes in rarefaction curve.

```
vegan::rarecurve(t(otu_table(ps3)), step = 20, label = FALSE, main = "Rarefaction Curve",
        col = c("black", "darkred", "forestgreen", "orange", "blue", "yellow", "hotpink"))
```

- Perform total sum scaling on agglomerated dataset.

```
ps4 <- transform_sample_counts(ps3, function(x) x / sum(x))
```

- Explore normalisation with violin plots.
- Compares differences in scale and distribution of the abundance values before and after transformation.
- Using arbitrary subset, based on Phylum = Firmicutes, for plotting (ie. can explore any taxa to observe transformation).

Change `Firmicutes`.

```
plot_abundance = function(physeq, Title = "Abundance", Facet = "Order", Color = "Phylum", variable = "Ty
    subset_taxa(physeq, Phylum %in% c("Firmicutes")) %>%
    psmelt() %>%
    subset(Abundance > 0) %>%
    ggplot(mapping = aes_string(x = variable, y = "Abundance", color = Color, fill = Color)) +
      geom_violin(fill = NA) +
      geom_point(size = 1, alpha = 0.3, position = position_jitter(width = 0.3)) +
      facet_wrap(facets = Facet) +
      scale_y_log10()+
      theme(legend.position="none") +
      labs(title = Title)
}

grid.arrange(nrow = 2, (plot_abundance(ps3, Title = "Abundance", Color = "Type", variable = "Type")),
             plot_abundance(ps4, Title = "Relative Abundance", Color = "Type", variable = "Type"))
```

- Explore normalisation with tree plots.

Change Type.

```
plot_tree(ps4.NICU_no_na, size = "Abundance", color = "Type",
    justify = "yes please", ladderize = "left") +
    labs(title = "Phylogenetic Tree and Relative Abundance") +
    scale_size_continuous(range = c(.5, 3))
```

5 Data Exploration and Univariate Analysis.

5.1 About.

This section again uses the phyloseq package (along with several others) to explore the data using bar, violin and ordination plot. This then leads into a collection of univariate analyses, including; alpha and beta diversity, and also taxonomic differential abundance.

5.2 Load required packages.

```
sapply(c("BiocManager", "ggplot2", "ggforce", "vegan", "knitr", "dplyr",
         "phyloseq", "phyloseqGraphTest", "igraph", "ggnetwork", "nlme",
         "reshape2", "tidyverse", "plyr", "DESeq2", "sjPlot", "ggpubr",
         "gridExtra", "grid", "gtable", "lazyeval"), require, character.only = TRUE)
```

5.3 Taxanomic distribution.

5.3.1 Bar charts

- Use `plot_bar_auto()` function wrapped around phyloseq's `plot_bar()` to explore the distribution of taxa at the genus and phylum levels.
- Subset transformed data (relative abundance) to only the top20 taxa.

Change `Primary_Group`.

```
top20 <- names(sort(taxa_sums(ps4), decreasing=TRUE))[1:20]
ps.top20 <- prune_taxa(top20, ps4)

plot_bar_auto <- function(ps, taxonomy){
plot_bar(ps, fill = taxonomy) +
    facet_wrap(~Primary_Group, scales = "free_x") +
    labs(title = paste0("Level:", taxonomy), y = "Abundance") +
    theme(legend.position = "bottom", legend.title = element_blank(),
    axis.title.x = element_blank(), axis.text.x = element_blank(),
    axis.ticks = element_blank())
}

plot_bar_auto(ps.top20, "Phylum")
```

5.3.2 Calculate the number samples containing a given taxa by creating a `samples_with_taxa()` function.

- Define a function takes the phyloseq object, taxonomy level and taxanomic name (with the later two as strings).
- It then gets the ASV name from the *phyloseq* `tax_table()` by filtering with *dply* and *lazyeval*. (*lazyeval* is needed because of two concepts;non-standard evaluation and lazy evaluation.
- `paste()` is then used to concatenate the ASVs and `collapse` to insert the 'or' symbol.
- The function then matches the ASV names to the `otu_table()` of the *phyloseq* object to select the desired column(s) that represent the taxa of interest, and then counts the number of rows that have any of the selected taxa with counts greater than 0 to get the number of samples with that taxa present.

```
samples_with_taxa <- function(ps_object, taxonomy_level, taxa){
 ASV <- tax_table(ps_object) %>%
        unclass() %>%
        as.data.frame() %>%
        filter_(interp(~y == x, .values=list(y = as.name(taxonomy_level), x = taxa))) %>%
        row.names() %>%
        paste(collapse = " | ")

   otu_table(ps_object) %>%
        as.data.frame() %>%
        select(matches(ASV)) %>%
        filter_all(any_vars( . > 0)) %>%
        nrow()
}

samples_with_taxa(ps4.microbiome, "Genus", "Bifidobacterium")
```

5.4 Beta diversity

- Use distance and ordination methods to explore the relationship between metadata.
- We calculate the distances using pruned, transformed (TSS) and non-agglomerated data.

```
ps2.TSS <- ps2 %>%
        transform_sample_counts(function(x) x / sum(x))
```

- We can then create distance matrices and plots for this data subset using several methods:
- e.g. bray-curtis or weighted unifrac distances with PCoA, NMDS, etc.
- Define a function that ordindates the previously transformed data, extarcts the eigenvalues, and creates a dissimilarity plot.
- Extract eigenvalues from ordination.

```
ordination_plots <- function(filtered_ps, variable, vis_method, dist_method){
# ordinate
ps_ordination <- ordinate(filtered_ps, method = vis_method, distance = dist_method)
# get eignenvalues
evals <- ps_ordination$values$Eigenvalues
# generate plot
plot_ordination(filtered_ps, ps_ordination, color = variable,
  title = "PCoA (Bray-Curtis)") +
  labs(col = variable) +
  coord_fixed(sqrt(evals[2] / evals[1])) +
  geom_point(size = 2)
 }

ordination_plots(ps2, "Primary_Group", "PCoA", "bray")
```

- Export plot.

```
ggsave("PCoA_Bray.png",
  plot = (plot_ordination(ps2.TSS, ps_ordination,
  color = "Type", title = "PCoA (Weighted-Unifrac)") +
  labs(col = "Type") +
  coord_fixed(sqrt(evals[2] / evals[1])) +
  geom_point(size = 2)+
  stat_ellipse(type = "norm", linetype = 2)), dpi = 600, height = 5, width = 5)
```

5.4.1 Statistical test: PERMANOVA.

- ps2 transformed.
- Preforming permutational anova for group-level differences based on dissimilarity.
- Extract otu table and metadata from phyloseq object.
- Use `adonis()` from the *vegan* package to perform the PERMANOVA.
- Homogeneity Condition
- Significant PERMANOVA means one of three things:
- there is a difference in the location of the samples (i.e. the average community composition).
- there is a difference in the dispersion of the samples (i.e. the variability in the community composition).
- there is a difference in both the location and the dispersion.
- If you get a significant PERMANOVA you'll want to distinguish between the three options by checking the homogeneity condition using `permdisp()`. If you get a non-significant result the first option above is correct.
- `betadisper()` gives a measure of the dispersion within groups. Thus, if the PERMANOVA test is significant and the permdisp is not, the significant result in your communities is due to a mean shift in community composition and not from increased variance within groups.

Change `Primary_Group`.

```
permanova_func <- function(ps2.TSS){

# permanova
ps_otu <- data.frame(otu_table(ps2.TSS))
ps_metadata <- data.frame(sample_data(ps2.TSS))
permanova <- adonis(ps_otu ~Primary_Group, data = ps_metadata, method = "bray")
permanova <- tableGrob(as.data.frame(permanova$aov.tab)) %>%
  annotate_figure(fig.lab = "PERMANOVA", fig.lab.face = "bold", fig.lab.size = 15)

# homogeneity condition
dist <- vegdist(ps_otu)
homogeneity <- as.data.frame(anova(betadisper(dist, ps_metadata$Primary_Group))) %>%
  tableGrob() %>%
  annotate_figure(fig.lab = "Homogeneity Condition", fig.lab.face = "bold", fig.lab.size = 15)

# combine ouputs in a grid
grid.arrange(permanova, homogeneity, ncol = 1)

}

permanova_func(ps2.TSS)
```

- Export results.

```
tab_df((permanova_func(ps2.TSS)),
       alternate.rows = TRUE,
       file = "PERMANOVA.doc")
```

- Explore the major contributors to the differences.

Change `Primary_Group`.

```
major_contributors <- function(ps2.TSS, variable){
# perform permanova
ps_otu <- data.frame(otu_table(ps2.TSS))
ps_metadata <- data.frame(sample_data(ps2.TSS))
permanova <- adonis(ps_otu ~Primary_Group, data = ps_metadata, method = "bray")

# coefficients
coef <- coefficients(permanova)[paste0(variable, "1"),]
top.coef <- coef[rev(order(abs(coef)))[1:20]]

genus_contributors <- tax_table(ps2.TSS) %>%
    unclass() %>%
    as.data.frame() %>%
    select("Genus") %>%
    rownames_to_column(var = "ASV") %>%
    right_join((as.data.frame(top.coef) %>%
    rownames_to_column(var = "ASV"))) %>%
    select(!"ASV")
```

```
return(genus_contributors)
}

major_contributors(ps2.TSS, "Primary_Group")
```

- Export table.

```
tab_df(major_contributors(ps2.TSS), alternate.rows = TRUE,
       title = "Major Contriutors to PERMANOVA differences.",
       file = "Major_contributors_beta_diversity.doc")
```

5.5 Alpha diversity.

- Define a function that calculates Shannon Index, Obsverved (richness) & Chao1 diversity, and binds it to our original metadata dataframe, which can then be used for analysis.

Change `ID`.

```
calc_alpha_diversity <- function(ps2){
# calculate metrics
ps_alpha_div <- ps2 %>%
                estimate_richness(measures = c("Shannon", "Observed", "Chao1")) %>%
                select(-se.chao1)

# creat ID column based on rownames
ps_alpha_div <- rownames_to_column(ps_alpha_div, var = "ID") %>%
                mutate(ID = as.factor(gsub("AM", "", ID)))

# join alpha metrics with metadata by the ID column
Metadata %>%
  filter(Type == "Microbiome") %>%
  right_join(ps_alpha_div, by = "ID") %>%
  as.data.frame()
}

ps_metadata <- calc_alpha_diversity(ps2)
```

- Create histogram to examine distribution.

```
# To determine if diveristy is normally distributed
ggplot(ps_metadata, aes(x = Shannon)) + geom_histogram() +
       xlab("Alpha Diversity") + ylab("Count")
```

- Test for normality.

```
shapiro.test(ps_metadata$Shannon)
```

5.5.1 Statistical test: compare mean/median between groups.

- define a function that performs a Wilcoxin test on the three diversity metrics and binds them (Shannon Index, Richness & Chao1).

Change `Primary_Group`.

```
diversity_analysis <- function(ps_metadata){

Shannon <- compare_means(Shannon ~ Primary_Group, data = ps_metadata,
                         method = "wilcox.test", p.adjust.method = "fdr")

Observed <- compare_means(Observed ~ Primary_Group, data = ps_metadata,
                          method = "wilcox.test", p.adjust.method = "fdr")

Chao1 <- compare_means(Chao1 ~ Primary_Group, data = ps_metadata,
                       method = "wilcox.test", p.adjust.method = "fdr")

bind_rows(Shannon, Observed, Chao1) %>%
  rename(c(".y." = "Diversity Measure"))
}

diversity_analysis(ps_metadata)
```

- Export *Diversity_Analysis* results table.

```
tab_df(diversity_analysis(ps_metadata), alternate.rows = TRUE,
       title = "Diversity Analysis: Admission Vs Discharge",
       file = "Alpha_Diversity_Analysis_Type.doc")
```

5.5.2 Plot alpha diversity.

- Use `plot_richness()` from *phyloseq*, which estimates alpha diversity metrics using *vegan* and plots them, taking standard *ggplot2 geoms_* for the plot design.
- use ps2 non-transformed data for alpha.

```
plot_richness(ps2, measures = c("Shannon", "Observed"),
              color = "ddPCR", title = "") +
    geom_point(size = 3.5, alpha = 0.7) +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank(),
          panel.border = element_rect(colour = "grey", fill = NA, size = 1))
```

- Export scatterplot.

```
ggsave("Alpha_Point.png", dpi = 600, height = 5, width = 5)
```

- Use `plot_richness()` to create boxplots of alpha diversity.
- To add a layer with p values use `stat_compare_means(comparisons = list(c("Admission", "Discharge")), method = "wilcox.test")`.

Change `Primary_Group`.

```
plot_richness(ps2, measures = c("Shannon", "Observed"),
              x = "Primary_Group", color = "Primary_Group", title = "") +
              geom_jitter(size = 1, alpha = 0.7) +
              geom_boxplot() +
              theme(panel.border = element_rect(colour = "grey", fill = NA, size = 1), legend.position =
              axis.text.x = element_blank(), axis.ticks = element_blank())
```

- Export boxplot.

```
ggsave("Alpha_Box.png", dpi = 600, height = 5, width = 5)
```

- shannon diversity for continuous variable.

Change `Gestational_Age_at_Birth`.

```
ps_metadata %>%
ggplot(aes(x = Gestational_Age_at_Birth, y = Shannon)) +
  geom_point() +
  geom_smooth(method = "lm", se = T)
```

6 Differential abundance analysis with *DESeq2*.

- Define function for calculating geometric means and estimating size factors.
- Define function to filter out taxa with small counts and low occurance. *count* and *samples* arguments need to be applied as numerical values.

```
calc_geo_means <- function(deseq_object){
# geometric mean
  gm_mean = function(x, na.rm = TRUE){
    exp(sum(log(x[x > 0]), na.rm = na.rm) / length(x))
  }
  geoMeans <- apply(counts(deseq_object), 1, gm_mean)
# size factors
  estimateSizeFactors(deseq_object, geoMeans = geoMeans)
}

deseq_filter <- function(deseq_object, count, samples){
  nc <- counts(deseq_object, normalized = TRUE)
  filtered <- rowSums(nc >= count) >= samples # filter = abundance of 10 in 60 samples.
  deseq_object[filtered,]
}
```

- Define a function to extract the results.
- Extract the results, order by p value, selects significant (<0.05) results, binds this data to the *tax_table* from the *phyloseq* object to get the taxonomic information, and then select and order the desired columns.

```r
# function for liklihood ratio test
get_deseq_res_lrt <- function(deseq_object){
  res = results(deseq_object)
  res = res[order(res$padj, na.last = NA), ]
  sigtab = res[(res$padj < 0.05), ]
  sigtab = cbind(as(sigtab, "data.frame"),
          as(tax_table(ps3)[rownames(sigtab), ], "matrix"))
  sigtab %>%
  arrange(padj) %>%
  select("log2FoldChange", "lfcSE", "padj", "Genus")
}

# function for Walds test and continuous variables
get_deseq_res_cont <- function(deseq_object, contrast_variable){
  res = results(deseq_object, name = contrast_variable)
  res = res[order(res$padj, na.last = NA), ]
  sigtab = res[(res$padj < 0.05), ]
  sigtab = cbind(as(sigtab, "data.frame"),
          as(tax_table(ps3)[rownames(sigtab), ], "matrix"))
  sigtab %>%
  arrange(padj) %>%
  select("log2FoldChange", "lfcSE", "padj", "Genus")
}

# function for Walds test and categorical variables
get_deseq_res_cat <- function(desq_object, contrast_variable, level1, level2){
  res = results(desq_object, contrast = c(contrast_variable, level1, level2))
  res = res[order(res$padj, na.last = NA), ]
  alpha = 0.05
  sigtab = res[(res$padj < alpha), ]
  sigtab = cbind(as(sigtab, "data.frame"),
    as(tax_table(ps3)[rownames(sigtab), ], "matrix"))
  sigtab %>%
  arrange(padj) %>%
  select("log2FoldChange", "lfcSE", "padj", "Genus") %>%
  add_column(Variable = paste0(contrast_variable, level1)) # label the base level
}
```

- Convert from *phyloseq* to *deseq* object.
- Calculate geometric means and filter to the most abundant and frequent taxa.
- Use `Deseq()` to perform the normalisation and analysis.
- Extract the results.

Change `Primary_Group`.

```r
# LRT
phyloseq_to_deseq2(ps3, ~ Primary_Group) %>%
    calc_geo_means() %>%
    deseq_filter(10,10) %>%
    DESeq(fitType = "local", test = "LRT", reduced = ~ 1) %>%
    get_deseq_res_lrt() %>%
    remove_rownames()
```

Change `Primary_Group`.

```
# Wald
phyloseq_to_deseq2(ps3, ~ Primary_Group) %>%
    calc_geo_means() %>%
    deseq_filter(10,10) %>%
    DESeq(fitType = "local", test = "Wald") %>%
    get_deseq_res_cat("Primary_Group", "NICU", "SCN") %>%
    remove_rownames()
```

7 Multivariant Analysis

7.1 Mixed effects modelling with *DESeq2* for differential abundance testing.

```
sapply(c("DESeq2", "phyloseq", "dplyr", "ggplot2", "grid",
         "gridExtra", "ggpubr", "sjPlot", "pheatmap", "tidyverse"),
          require, character.only = TRUE)
```

7.1.1 Explore clustering of variables with PCoA and Bray-Curtis.

```
ordination_plots(ps2, "Primary_Group", "PCoA", "bray")
```

7.1.1.1 For multiple plots.

```
grid.arrange(ordination_plots(ps2, "Primary_Group", "PCoA", "bray"),
             ordination_plots(ps2, "Feeding_Type", "PCoA", "bray"),
             ncol=2)
```

7.1.2 Centre and scale continuous variables.

Change ID.

```
centre_and_scale <- function(data){
# get numeric variables
data2 <- data %>%
  select_if(is.numeric)
# entering and scaling over variables
data3 <- sapply(data2, function(x) scale(x, center=T, scale = 2*sd(x))) %>%
  as.data.frame() %>%
  rownames_to_column("RowID")
# join scaled/centred data to non-numeric data
data %>%
  select_if(negate(is.numeric)) %>%
  rownames_to_column("RowID") %>%
  left_join(data3, by = "RowID") %>%
  select(-RowID)
}
```

```r
sample_data(ps3) <- sample_data(ps3) %>%
  unclass() %>%
  as.data.frame() %>%
  centre_and_scale() %>%
  mutate("Sample" = ID) %>% # need to redo the rownames to save it back into the original ps object
  column_to_rownames("Sample")
```

7.1.3 Test for multicollinearity.

- Define the `corvif()`function that takes metadata and creates a linear model to see if any collinearity exists between variables.
- Then use this function on a defined a vector with all the variables to be included in the model.
- If GVIF < 3 = no collinearity.

```r
# myvif
myvif <- function(mod) {
  v <- vcov(mod)
  assign <- attributes(model.matrix(mod))$assign
  if (names(coefficients(mod)[1]) == "(Intercept)") {
    v <- v[-1, -1]
    assign <- assign[-1]
  } else warning("No intercept: vifs may not be sensible.")
  terms <- labels(terms(mod))
  n.terms <- length(terms)
  if (n.terms < 2) stop("The model contains fewer than 2 terms")
  if (length(assign) > dim(v)[1] ) {
    diag(tmp_cor)<-0
    if (any(tmp_cor==1.0)){
      return("Sample size is too small, 100% collinearity is present")
    } else {
      return("Sample size is too small")
    }
  }
  R <- cov2cor(v)
  detR <- det(R)
  result <- matrix(0, n.terms, 3)
  rownames(result) <- terms
  colnames(result) <- c("GVIF", "Df", "GVIF^(1/2Df)")
  for (term in 1:n.terms) {
    subs <- which(assign == term)
    result[term, 1] <- det(as.matrix(R[subs, subs])) * det(as.matrix(R[-subs, -subs])) / detR
    result[term, 2] <- length(subs)
  }
  if (all(result[, 2] == 1)) {
    result <- data.frame(GVIF=result[, 1])
  } else {
    result[, 3] <- result[, 1]^(1/(2 * result[, 2]))
  }
  invisible(result)
}

# corvif
```

```
corvif <- function(data) {
  data <- as.data.frame(data)

  form    <- formula(paste("fooy ~ ",paste(strsplit(names(data)," "),collapse = " + ")))
  data  <- data.frame(fooy = 1 + rnorm(nrow(data)) ,data)
  lm_mod  <- lm(form,data) # runs linear model with above formula and metadata

  cat("\n\nVariance inflation factors\n\n")
  print(myvif(lm_mod))
}
```

```
sample_data(ps3) %>%
  unclass %>%
  as.data.frame() %>%
  select(Feeding_Type, NEC, Sepsis, Died, Mode_of_Delivery, Antenatal_Antibiotics, Neonatal_Antibiotics
  corvif()
```

7.2 DESeq2

- Convert from *phyloseq* to *deseq* object.
- Use prevviously defined functions to calculate geometric means and filter to the most abundant and frequent taxa.
- Use `Deseq()` to perform the normalisation and analysis.
- Extract the results using appropriate previosuly defined function.

7.2.1 LRT

Change `Primary_Group` & `Feeding_Type`.

```
phyloseq_to_deseq2(ps3, ~ Primary_Group + Feeding_Type) %>%
    calc_geo_means() %>%
    deseq_filter() %>%
    DESeq(fitType = "local", test = "LRT", reduced = ~ Primary_Group) %>%
    get_deseq_res_lrt() %>%
    remove_rownames()
```

7.2.2 Wald

Change variables and arguments.

```
deseq_model <- phyloseq_to_deseq2(ps3, ~ Primary_Group + Feeding_Type + NEC) %>%
    calc_geo_means() %>%
    deseq_filter() %>%
    DESeq(fitType = "local", test = "Wald")

# rbind the results outputs to create one large output
deseq_model <- deseq_model %>%
  get_deseq_res_cat("Primary_Group", "Admission", "Discharge") %>%
  remove_rownames() %>%
  rbind(get_deseq_res_cat("Feeding_Type", "Breastmilk", "Formula") %>%
```

```
  remove_rownames()) %>%
  rbind(get_deseq_res_cat("Feeding_Type", "Breastmilk", "Breastmilk & Formula") %>%
  remove_rownames()) %>%
  rbind(get_deseq_res_cat("Feeding_Type", "Formula", "Breastmilk & Formula") %>%
  remove_rownames()) %>%
  rbind(get_deseq_res_cat("NEC") %>%
  remove_rownames())
```

7.2.3 Export DESeq2 results

```
tab_df(deseq_model, alternate.rows = TRUE,
       title = "Significantly Differentially Abundant Taxa",
       file = "DESeq2_Mixed_Model.doc")
```

7.2.4 DESeq2 Plots

- Construct histograms to compare pre and post transformation.
- Call `estimateDispersions()` to calculate abundances with `getVarianceStabilizedData()`.
- **NB** piped to `calc_geo_means()` to calculate geometric means and estimate size factors, which is needed for the above.
- **NB.** the samples are in columns in the *deseq* object but in rows for the *phyloseq* object.

Change `Primary_Group` & `Feeding_Type`.

```
plot_deseq_transformation <- function(deseq_object){
multi.deseq <- estimateDispersions(deseq_object, fitType = "local")

abund_sums_trans <- data.frame(sum = colSums(getVarianceStabilizedData(multi.deseq) ),
                      sample = colnames(getVarianceStabilizedData(multi.deseq) ),
                      type = "DESeq2")

abund_sums_no_trans <- data.frame(sum = rowSums(otu_table(ps3)),
                        sample = rownames(otu_table(ps3)),
                        type = "None")

grid.arrange((ggplot(abund_sums_trans) +
  geom_histogram(aes(x = sum), binwidth = 1) +
  xlab("Abundance within sample") +
  ggtitle("DESeq2 transformation")),
  (ggplot(abund_sums_no_trans) +
  geom_histogram(aes(x = sum), binwidth = 200) +
  xlab("Abundance within sample") +
  ylim(0,4) +
  ggtitle("No transformation")),
  nrow = 2)
}

phyloseq_to_deseq2(ps3, ~ Primary_Group + Feeding_Type) %>% calc_geo_means() %>% plot_deseq_transformati
```

- Visualising deseq-transformed abundances with heat maps.

Change `Primary_Group` & `Feeding_Type`.

```
phyloseq_to_deseq2(ps3, ~ Primary_Group + Feeding_Type) %>%
    calc_geo_means() %>%
    deseq_filter() %>%
    DESeq(fitType = "local", test = "LRT", reduced = ~ Primary_Group) %>%
    varianceStabilizingTransformation() %>%
    assay() %>%
    cor() %>%
    pheatmap()
```

- Visualising deseq-transformed abundances with PCA (substitute in any variable).

Change `Mode.of.Delivery`.

```
multi.deseq.clean %>%
  varianceStabilizingTransformation() %>%
  plotPCA(intgroup = "Mode.of.Delivery") +
   xlim(-20,20)+
   ylim(-20,20)
```

8 Mixed effects modelling with glms for alpha diversity

```
sapply(c("lme4", "nlme", "dplyr", "plyr", "lmerTest", "aods3",
         "tidyverse", "ggplot2", "MuMIn", "sjPlot", "gridExtra",
         "grid", "car", "emmeans", "ggpubr"),
         require, character.only = TRUE)
```

8.1 Centre/Scale numerical values

```
glm_data <- centre_and_scale(ps_metadata)
```

8.1.1 Test for collinearity

Change variables.

```
glm_data %>%
  select(Feeding_Type, NEC, Sepsis, Died, Mode_of_Delivery, Antenatal_Antibiotics, Neonatal_Antibiotics
  corvif()
```

8.2 Fit Model

Change fixed and random effects.

```
global <- lme4::glmer(Shannon ~ Primary_Group + Feedin_Type + (1|URN), data = glm_data) %>% summary()
```

8.2.1 If convergence issues arise, use allFit to see why.

- If they all work then any non-convergence warning is a false positive.

Change fixed and random effects.

```
glmer_refit <- lme4::glmer(Shannon ~ Primary_Group + Feedin_Type + (1|URN), data = glm_data) %>%
  allFit() %>%
  summary()

glmer_refit$which.OK # which optimisers work
```

8.2.2 Calculate the goodness of fit and R2.

- Calculate this again post bakwards selection.

```
gof(global)
r.squaredGLMM(global)
```

8.3 Backwards Selection.

- Define a function that determines what variable is contributing least to the model, as determined by AIC score.
- Then apply that function to the model, and subsequent models, removing variables from the model that are not contributing (first from the interaction and then from the model entirely).

Create as many global iterations as needed.

```
dfun <- function(x) {
  x$AIC <- x$AIC-min(x$AIC)
  names(x)[2] <- "dAIC"
  x
}

dfun(drop1(global))

global2 <- # remove variables from inside interaction, or from model entirely

dfun(drop1(global2))
```

8.3.1 Calculate the goodness of fit and R2.

Replace model name with final model.

```
gof(global2)
r.squaredGLMM(global2)
```

8.3.2 QQ-Plot.

Replace model name with final model.

```
qqnorm(resid(global2))
```

8.4 Reintgeration

- Add variables from the first model back into the final model one at a time to calculate estimates, SE and p values for each.

Change fixed effects.

```
lme4::glmer(Shannon ~ Primary_Group + Feedin_Type + (1|URN), data = glm_data) %>%
  summary()
```

FINISIHED

Link to github repo.