Pipeline

Jacob Westaway

Last updated on 2021-07-31

About.

This document contains a pipeline to go from raw Illumina MiSeq reads to a phyloseq object (along with some exploratory analysis) and is based on the workflow from the paper Characterising the bacterial gut microbiome of probiotic-supplmented very-preterm infants, which was based largely around this DADA2 workflow developed by *Callahan, et al.*.

Load required packages.

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

Read quality.

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/new_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))
```

Extract sample names.

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

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

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

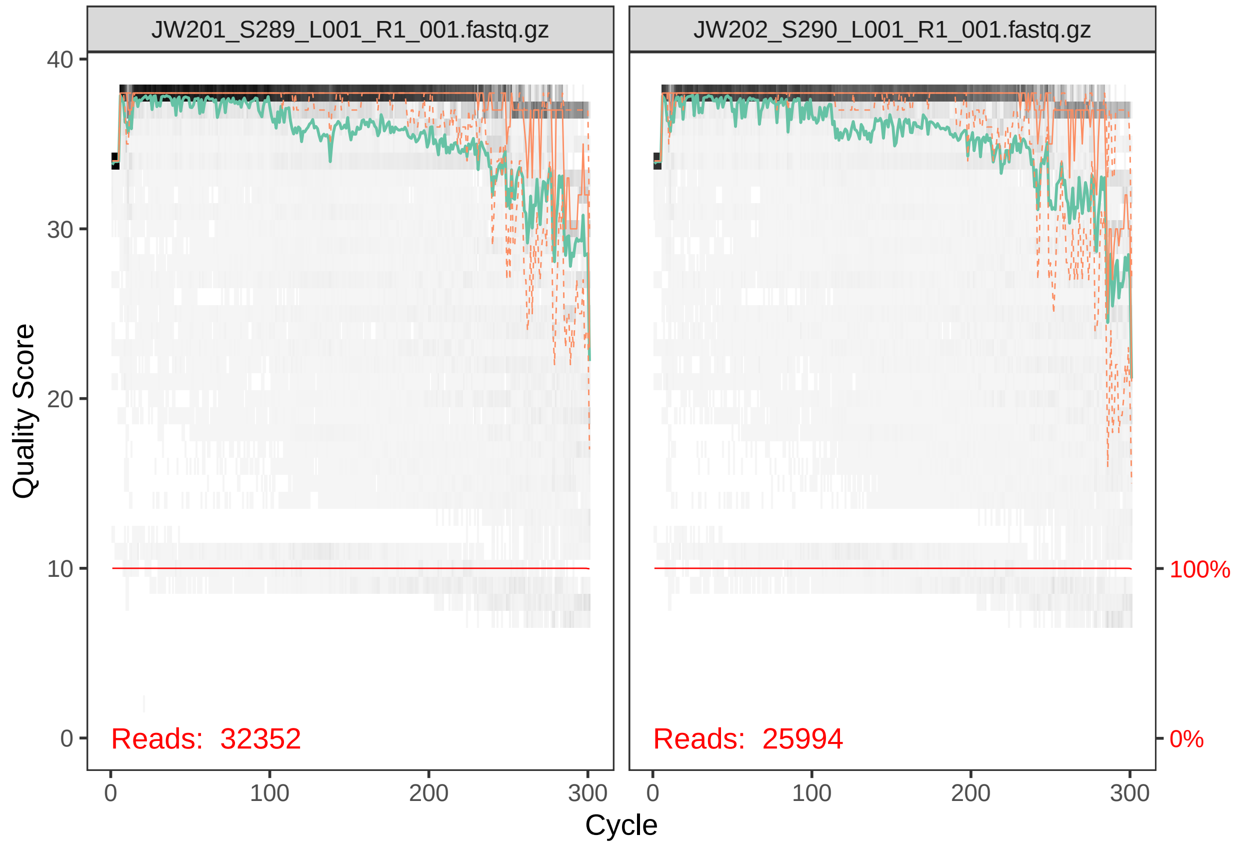


Figure 1: Quality of forward reads.

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

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"))
```

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.

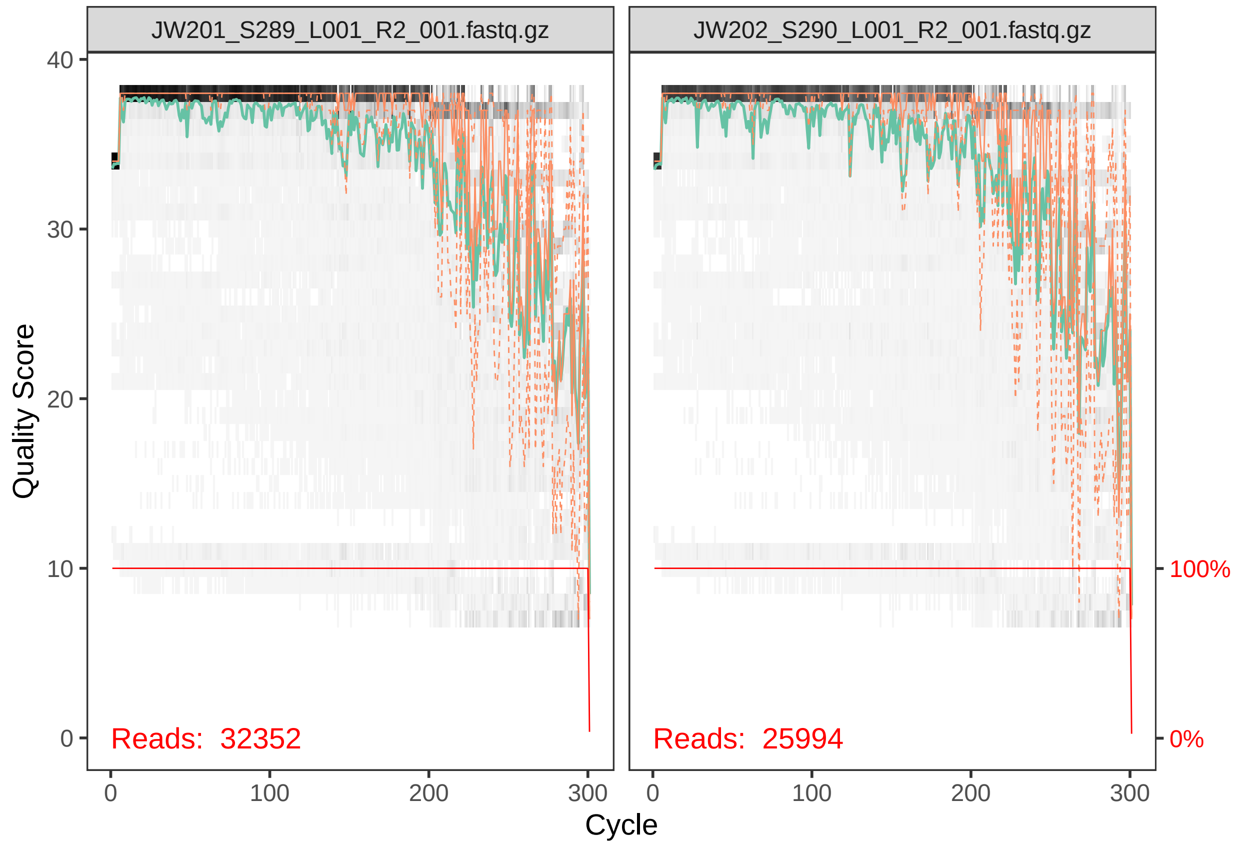


Figure 2: Quality of reverse reads.

- `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)
out
```

Infer sequence variants.

Calculate Error Rates.

- Error rates are used for sample ineference downstream.

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

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

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)
```

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
```

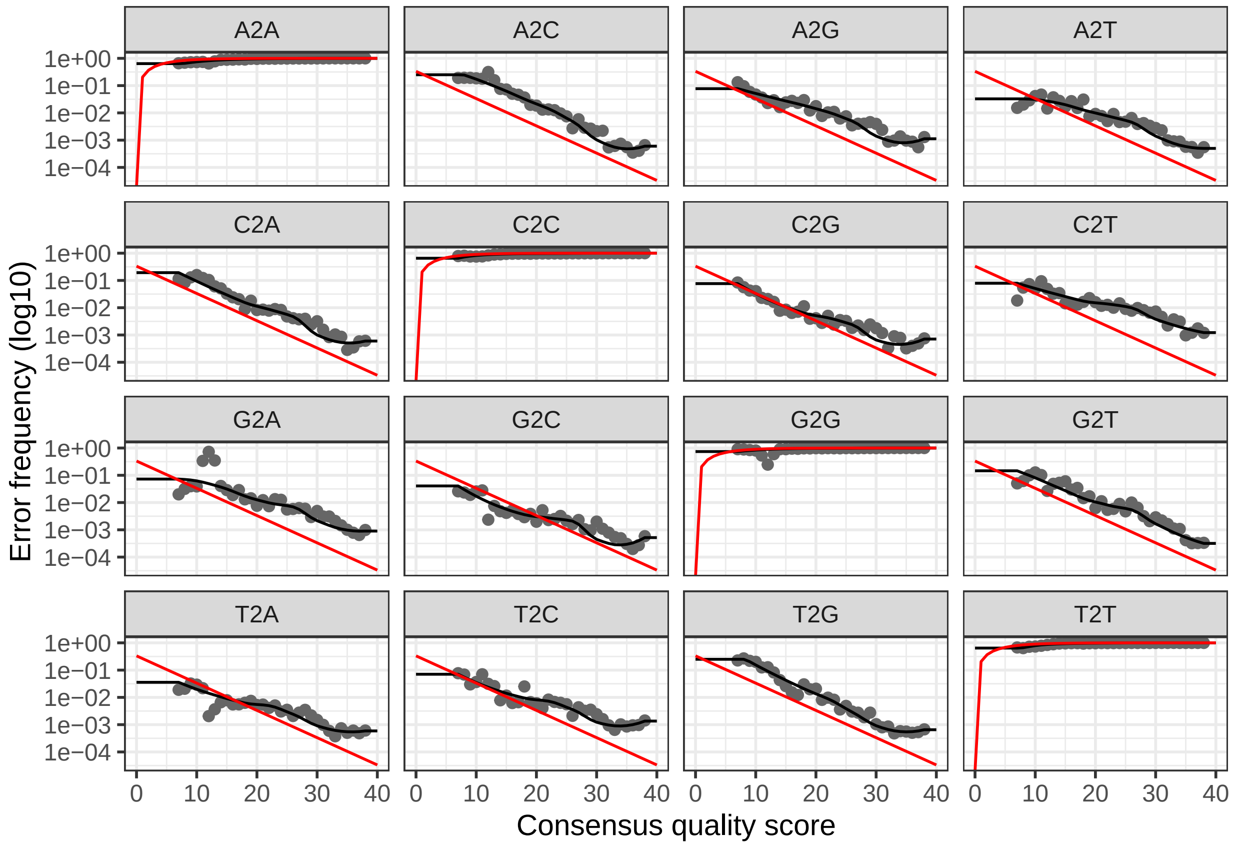


Figure 3: Error rates for forward reads

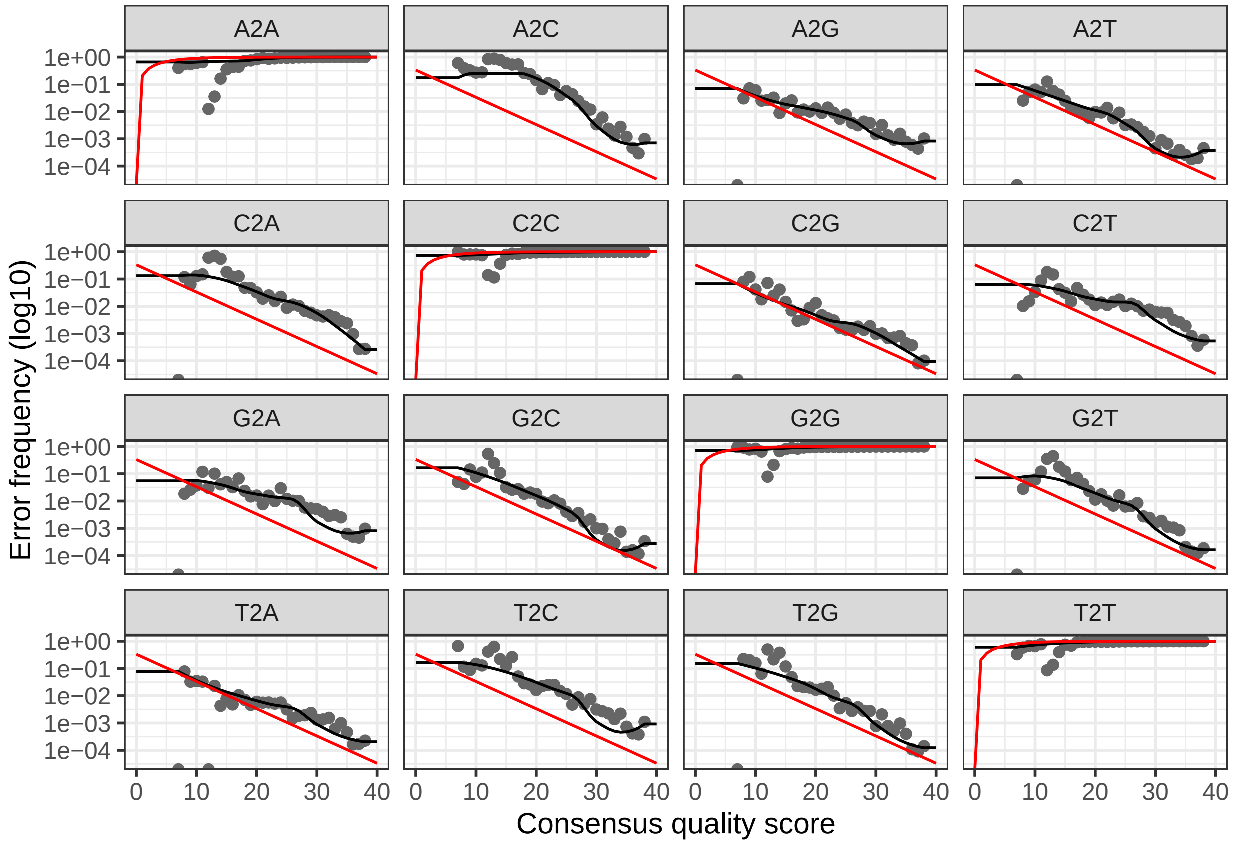


Figure 4: Error rates for reverse reads.

Sequence Inference.

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

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

Inspect denoised data.

```r
dadaFs[[1]]

dadaRs[[1]]
```

Merge Paired Reads and inspect merged data.

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

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

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

Construct ASV table.

- Check dimentions and inspect distribution of sequence lengths.

```r
seqtab <- makeSequenceTable(mergers)

dim(seqtab)

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

Remove chimeras.

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

Track reads through pipeline.

```r
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)
```

```
##        input filtered denoisedF denoisedR merged nonchim
## JW201  32352    27583     27342     27406  26036   14276
## JW202  25994    21958     21868     21908  21762   20325
## JW203     31        6         1         3      0       0
## JW203b 30571    25513     24856     25197  22249    7368
## JW204  25523    21198     20751     21100  18947   10893
## JW205  31796    26653     25969     26433  23889   11348
```

Contamination removal with MicroDecon.

```
library(microDecon)
```

Read in metadata (needed for MicroDecon)

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

Reformat data for *MicroDecon*.

- Function is **data specific**.
- 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()

microdecon.df <- microdecon.df %>%
  relocate("JW219B", "JW220") %>%
  tibble::rownames_to_column(var = "ID") # turn the rownames into the first column
}

microdecon.df <- wrangle_microdecon(seqtab.nochim)
```

Decontaminate data using `decon()`.

- `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 = 2,
                  numb.ind = c(21,3), taxa = F)
```

```


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

Check *MicroDecon* Outputs.

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()
```

Remove non-amplified samples and rename all.

```
seqtab.microdecon <- seqtab.microdecon %>%
  as.data.frame() %>%
  rownames_to_column("ID") %>%
  filter(ID != c("JW203", "JW211")) %>%
  mutate(ID = str_remove(ID, "b")) %>%
  mutate(ID = str_remove(ID, "JW")) %>%
  column_to_rownames("ID") %>%
  as.matrix()
```

Merging multiple sequence runs.

```
# Reading in pilot data for rmd.
seqtab.combined <- read_csv("Data/seqtab_original.csv") %>%
  column_to_rownames("ID") %>%
  as.matrix() %>%
  mergeSequenceTables(seqtab.microdecon)
```

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, "Data/silva_nr_v132_train_set.fa.gz")

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

Calculate percentage of NA taxa

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

```
## [1] 19.73255
```

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

```
##   Kingdom    Phylum     Class     Order    Family     Genus   Species
##  0.000000  1.369863  2.054795  4.566210  9.589041 32.876712 87.671233
```

Preprocessing: Creating a Phyloseq Object.

About.

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

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)
```

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)) %>%
  add_row(URN = "ZymoDNA1", ID = "ZymoDNA1", Type = "Control") %>%
  add_row(URN = "ZymoDNA2", ID = "ZymoDNA2", Type = "Control") %>%
  add_row(URN = "ZymoDNA4", ID = "ZymoDNA4", Type = "Control") %>%
  mutate(Sample_ID = ID) %>%
  column_to_rownames("Sample_ID") %>%
  mutate(Mode_of_Delivery = str_replace(Mode_of_Delivery, "Ceaserean", "Cesarean")) %>%
  mutate(Batch = 1:nrow(.)) %>%
  mutate(Batch = if_else(Batch <= 20, "Run2", "Run1"))
```

Constrcut the Phyloseq object.

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

```
ps <- phyloseq(otu_table(seqtab.combined, taxa_are_rows=FALSE),
               sample_data(Metadata),
               tax_table(taxa))
```

Wrangling the metadata.

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

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

Getting read counts

```
sample_data(ps) %>%
  unclass() %>%
  as.data.frame() %>%
  mutate(TotalReads = sample_sums(ps)) %>%
  ggplot(aes(TotalReads)) +
    geom_histogram() +
    ggtitle("Sequencing Depth")
```

Filtering and normalisation.

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"))
```

Remove problematic samples

- samples that produce NA values during transformations downstream in the otu_table (for some reason it won't allow all within the same function, hence the pipe)

```
ps1 <- subset_samples(ps1, ID != "219") %>%
  subset_samples(ID != "141") %>%
  subset_samples(ID != "118")
```

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

Check percentages of NA values left.

```
## [1] 18.81614
```

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

```
##    Kingdom     Phylum      Class      Order     Family      Genus    Species
##  0.0000000  0.0000000  0.6944444  3.2407407  8.3333333 31.9444444 87.5000000
```

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")
```

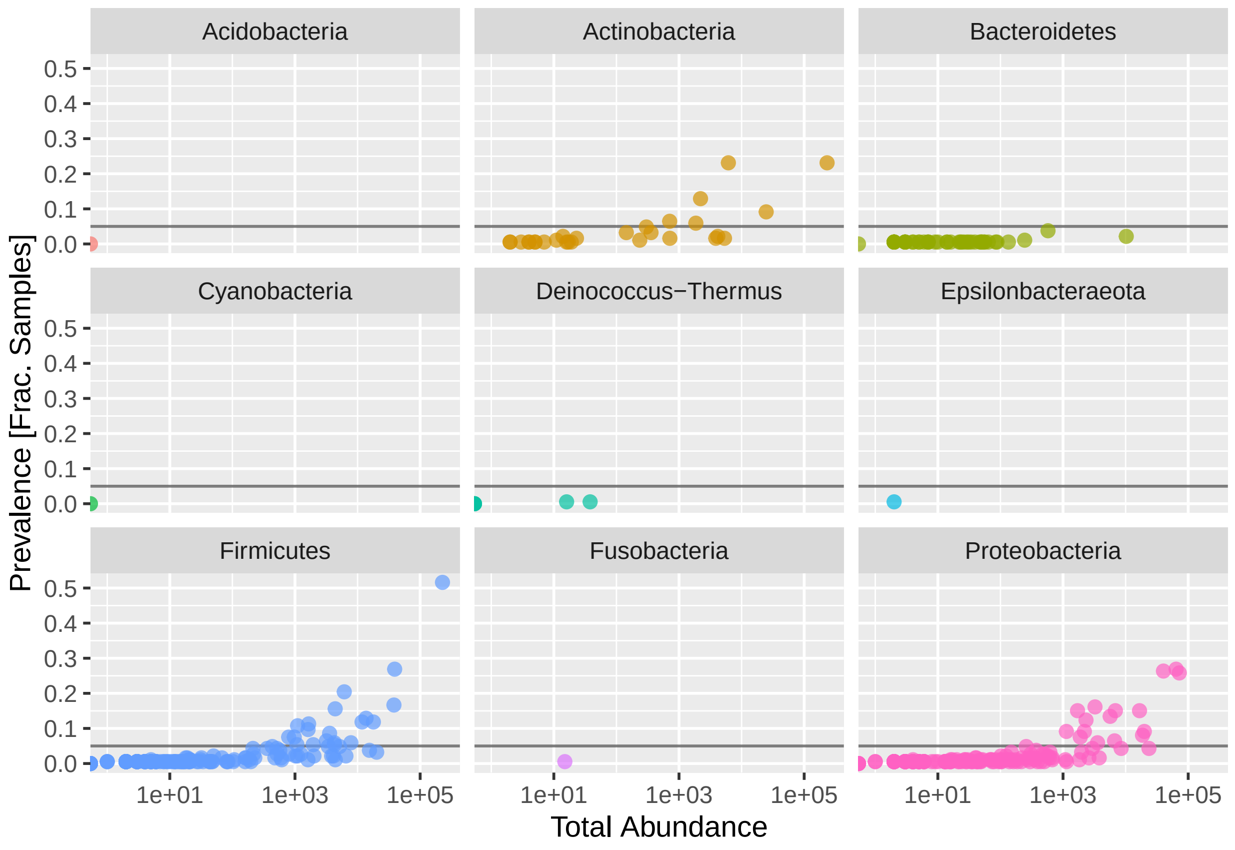


Figure 5: Scatterplot exploring the relationship between prevelance and abundance of phyla.

- 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)
#ps2 = ps1
```

Subset phyloseq object for data to be analyzed.

```
ps2 <- subset_samples(ps2, Type == "Discharge")
```

- 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")
```

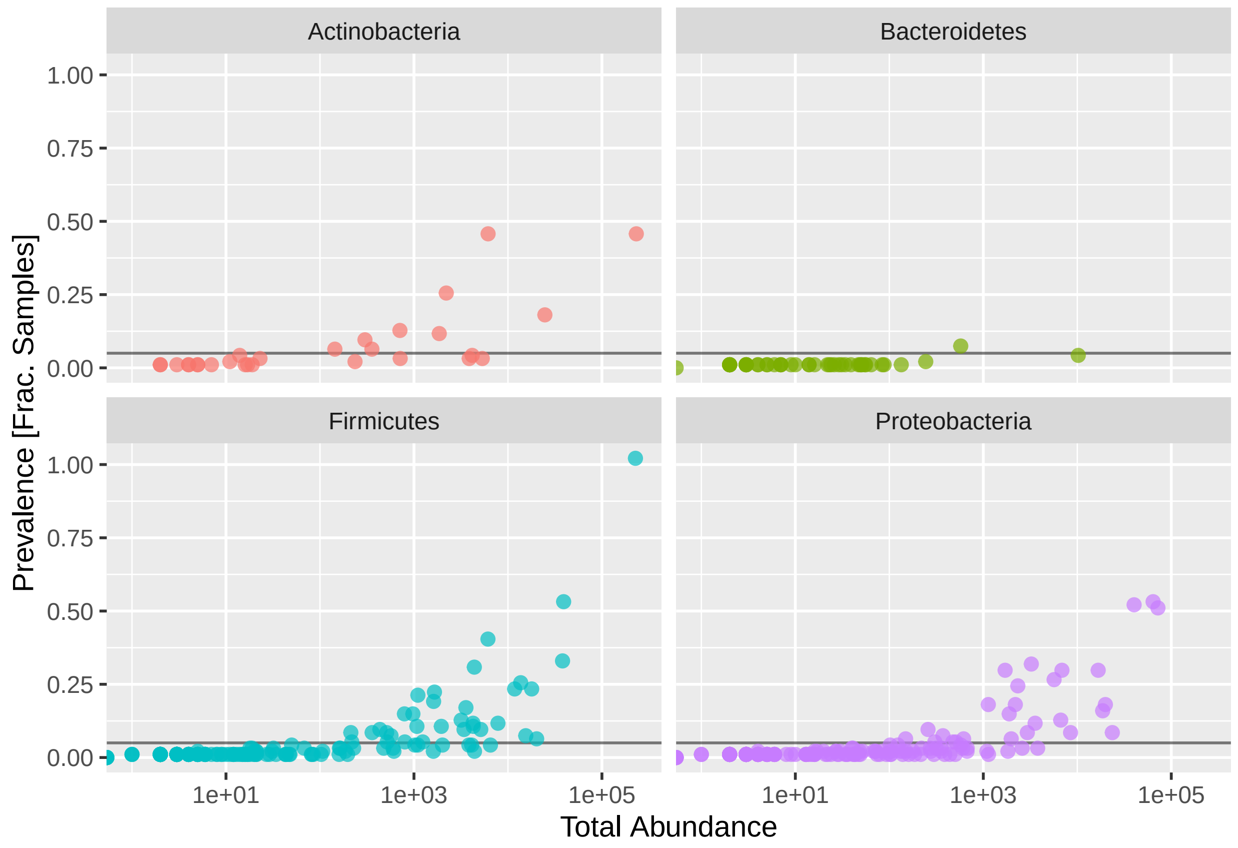


Figure 6: Scatterplot exploring the relationship between prevelance and abundance of phyla on data passed through a prevalence threshold.

Aggolmerate taxa.

- Combine features that descend from the same genus as most species have not been identified due to the poor taxonomic depth in 16S, a result of the length of the fragment amplified from the 16SrRNA gene.

- Can check how many genera would be present after filtering by running `length(get_taxa_unique(ps2, taxonomic.rank = "Genus"))` and/or `ntaxa(ps3)` will give the number of post agglomeration taxa.

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

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).

```
plot_abundance = function(physeq, Title = "Abundance",
                          Facet = "Order", Color = "Phylum", variable = "Type"){

    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"))
```

Data exploration and univariate analysis.

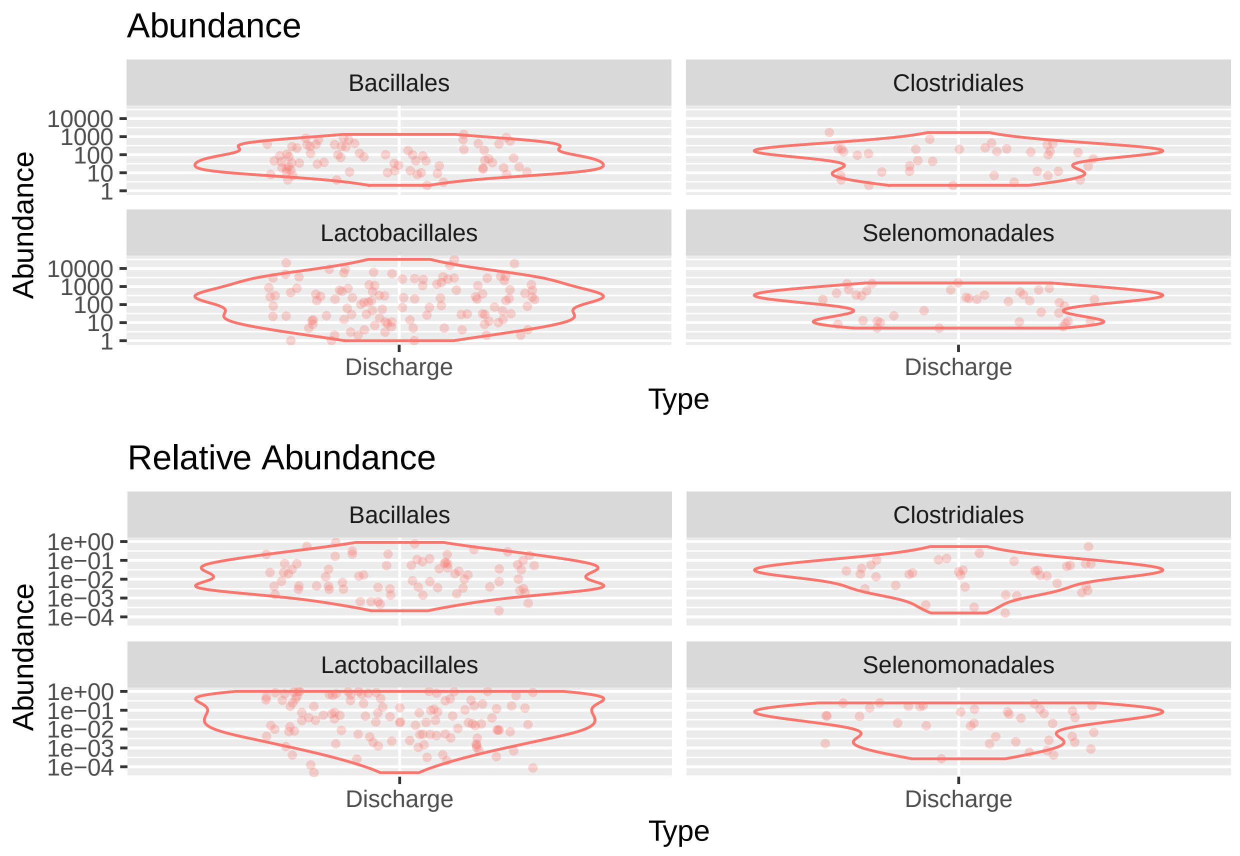


Figure 7: Violin plots exploring of distribution of abundance in Firmicutes before and after normalisation of data.

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.

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)
```

Taxanomic distribution.

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.

```
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, "Genus")
```

Investigate specific taxa

```
ps4 %>%
  subset_taxa(Genus == "Bifidobacterium") %>%
  plot_bar_auto(., "Genus")
```

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).

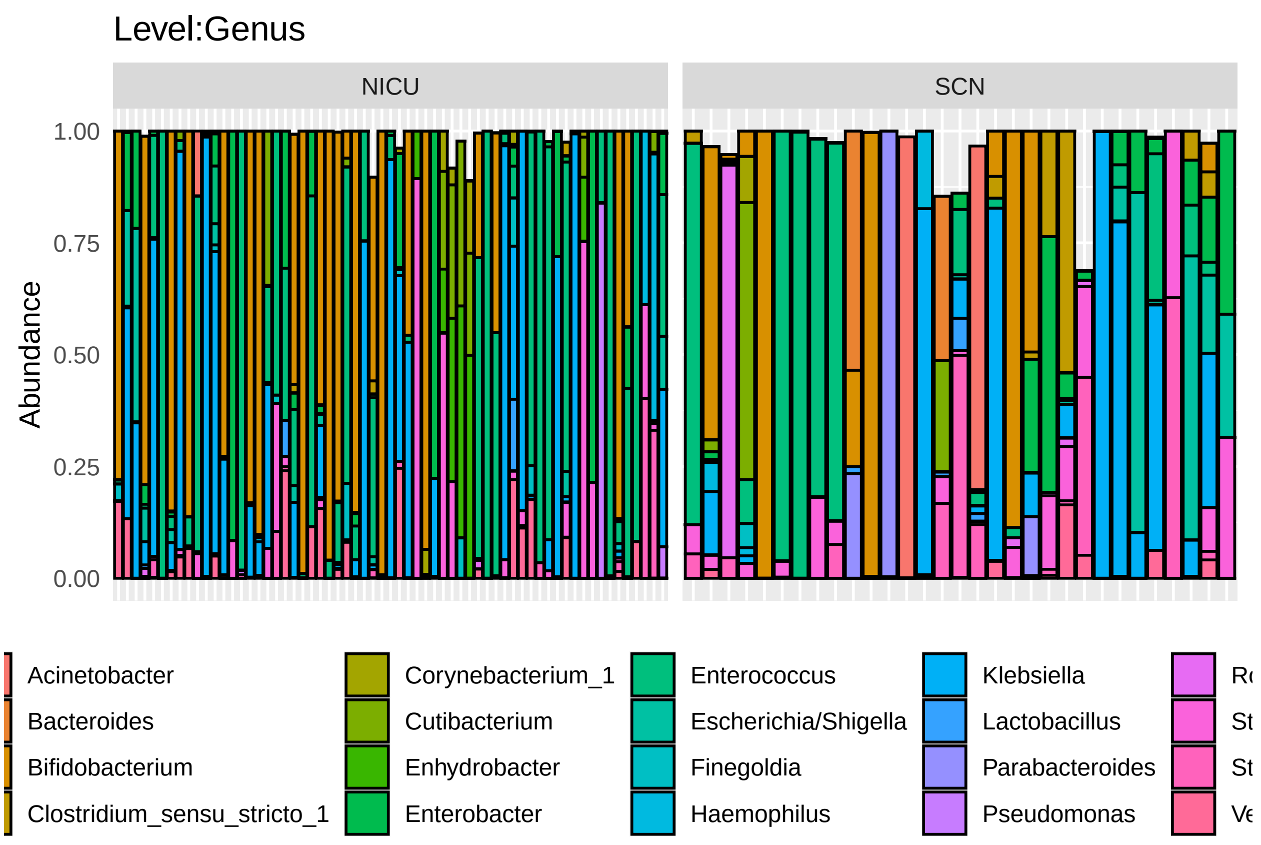


Figure 8: Bar plots of the taxonomic distribution (relative abundance) at the genus levels.

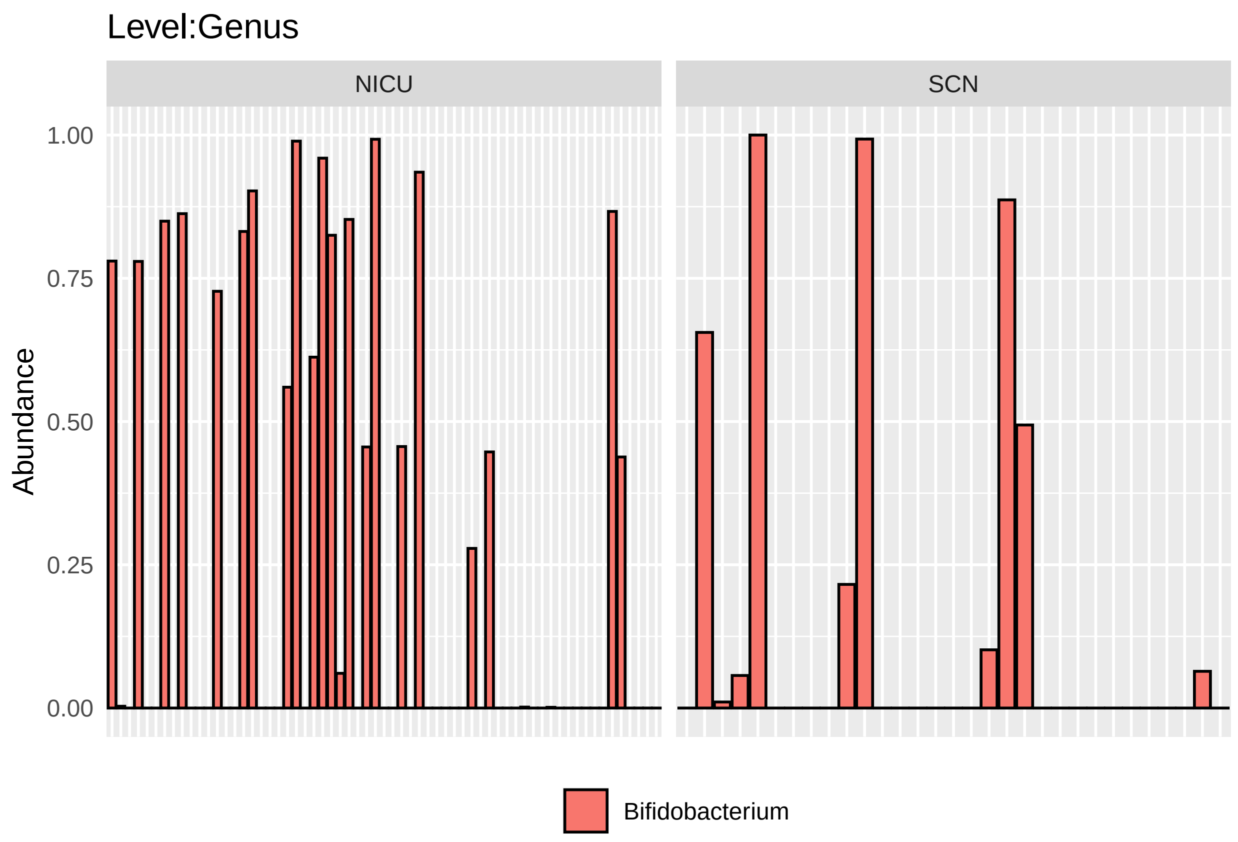


Figure 9: Bar plot of the taxonomic distribution of Bifidobacterium.

- 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()
}

ps4 %>%
  subset_samples(Primary_Group == "NICU") %>%
samples_with_taxa(., "Genus", "Bifidobacterium")
```

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 = 5) +
  stat_ellipse(typre = "norm", linetype = 2) +
  scale_color_hue(labels = c("Probiotic-treated", "Non-treated"))
 }

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

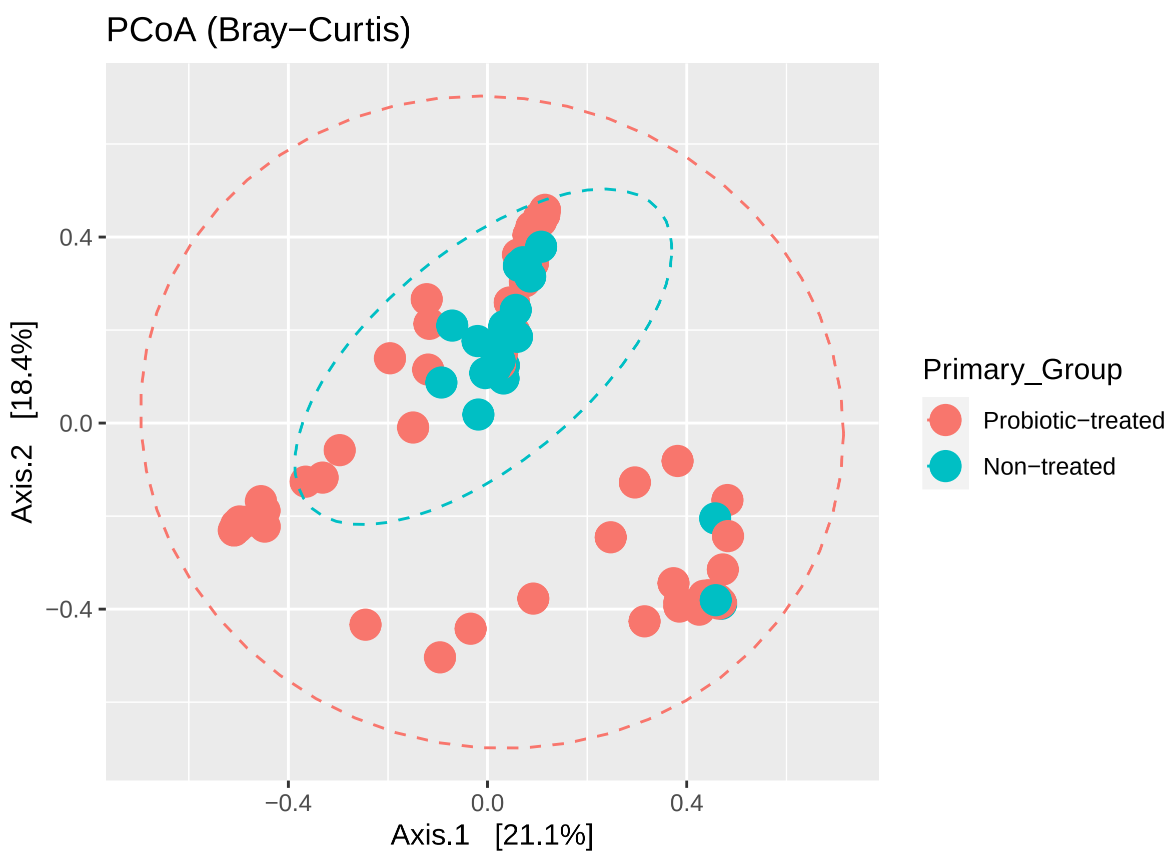


Figure 10: PCoA plot of Bray-Curtis distances coloured by date.

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.

```
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)
```

- Explore the major contributors to the differences.

```
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")
```

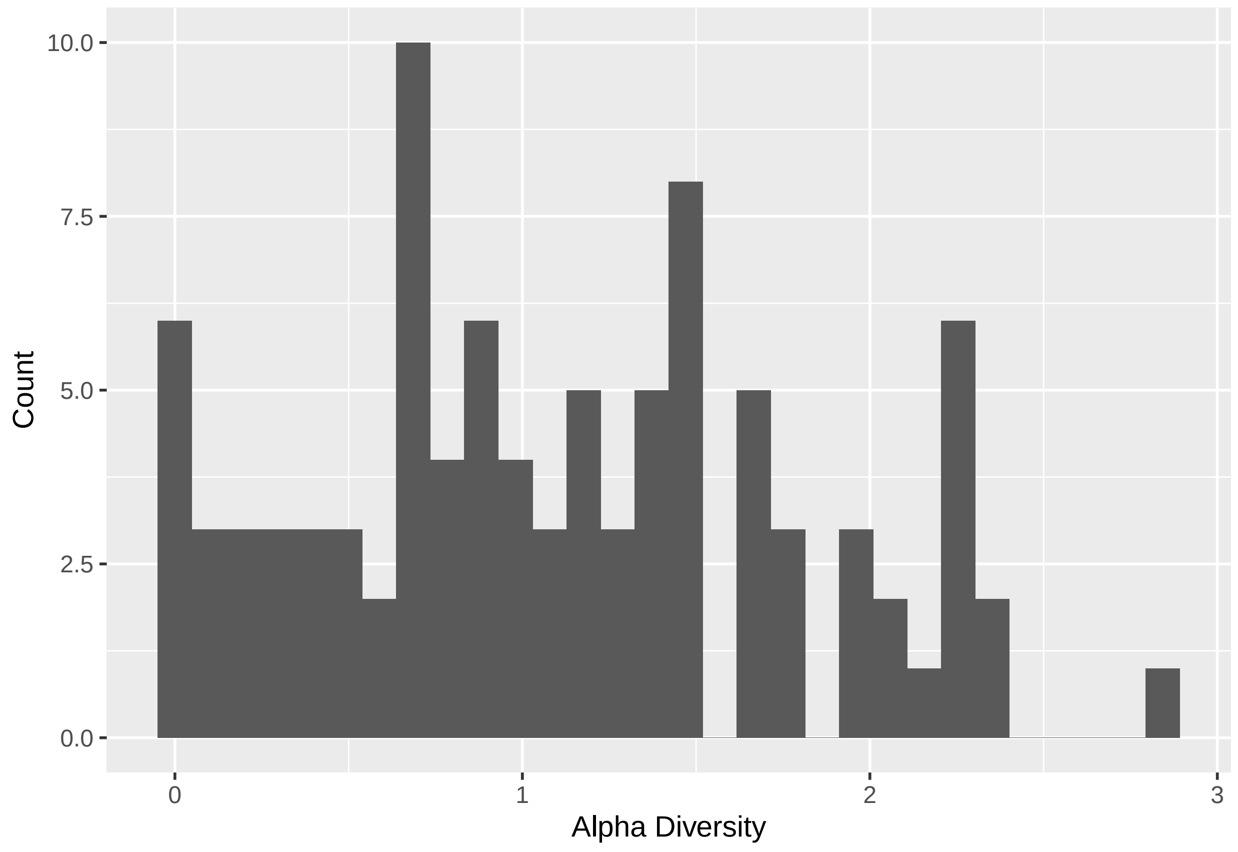
Alpha diversity.

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

```r
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("X", "", ID)))

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

ps_metadata <- calc_alpha_diversity(ps2)
```

- Create histogram to examine distribution.

```r
# 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)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  ps_metadata$Shannon
## W = 0.97086, p-value = 0.03382
```

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).

```
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)
```

```
## # A tibble: 3 x 8
##   `Diversity Measure` group1 group2     p p.adj p.format p.signif method
##   <chr>               <chr>  <chr>  <dbl> <dbl> <chr>    <chr>    <chr>
## 1 Shannon             SCN    NICU   0.278  0.28 0.28     ns       Wilcoxon
## 2 Observed            SCN    NICU   0.171  0.17 0.17     ns       Wilcoxon
## 3 Chao1               SCN    NICU   0.171  0.17 0.17     ns       Wilcoxon
```

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 = "Primary_Group", 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))
```

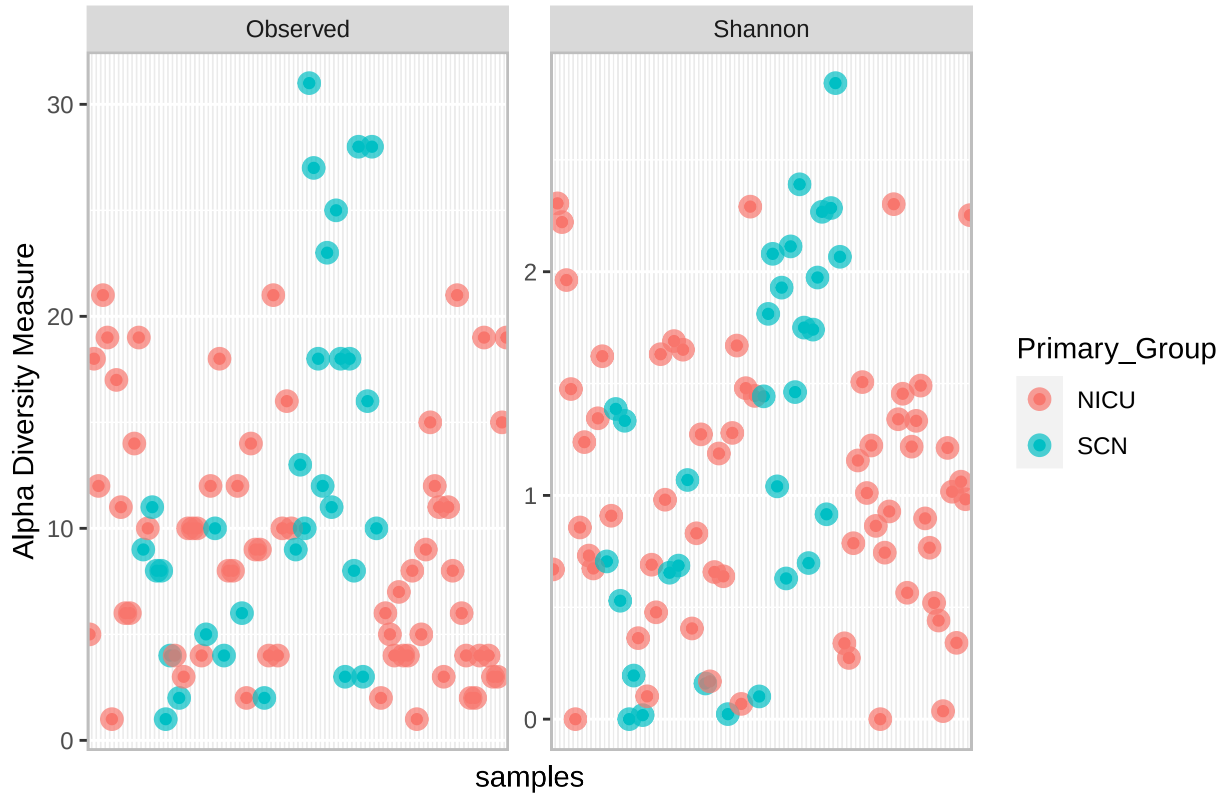


Figure 11: Scatterplot of richness and shannon diversity metrics coloured by ddPCR.

- 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")`.

```
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),
              axis.text.x = element_blank(), axis.ticks = element_blank())
```

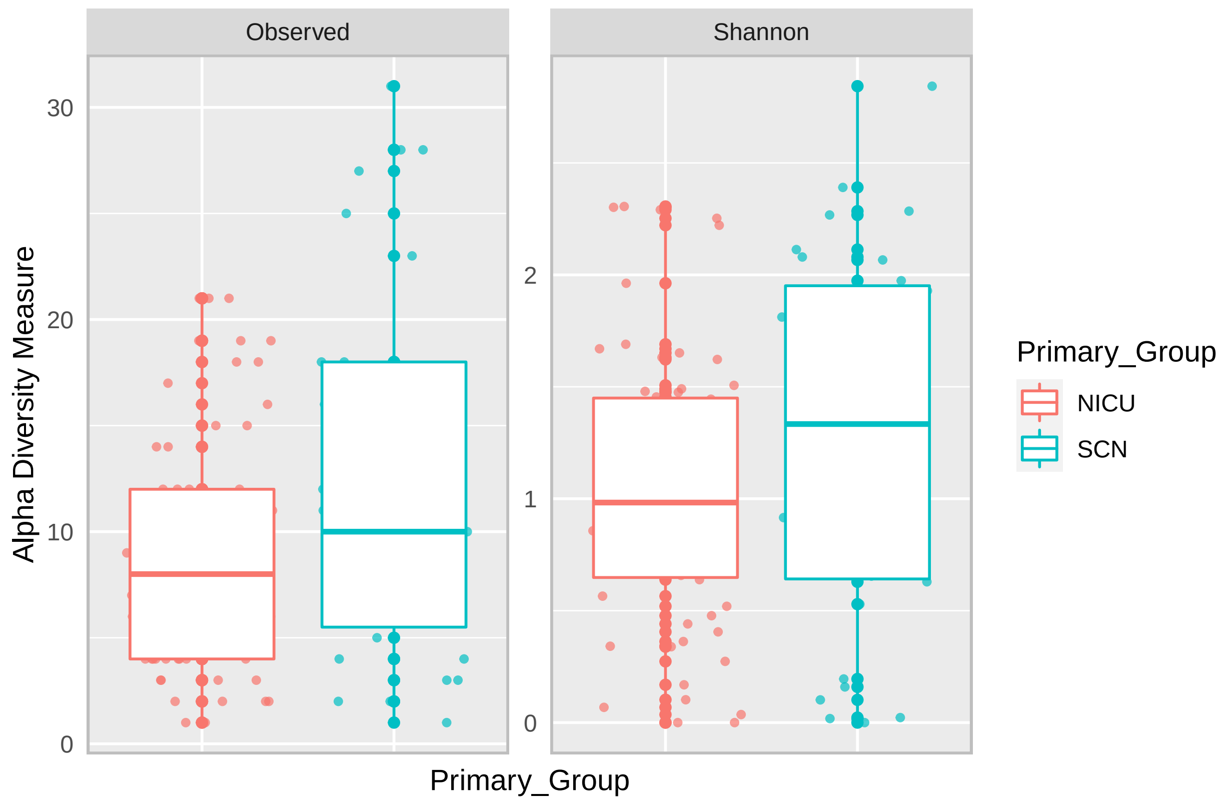
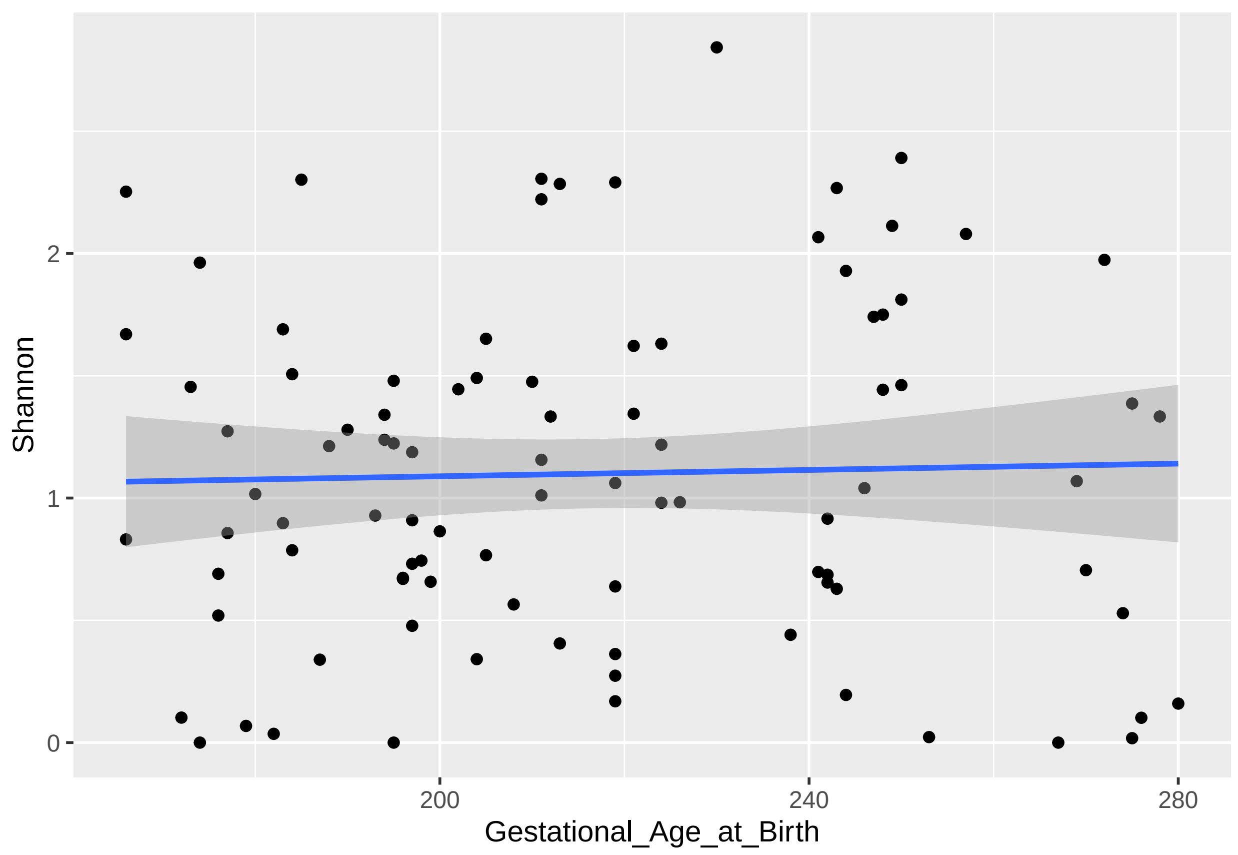


Figure 12: Boxplots of richness and shannon diversity metrics coloured by date.

- shannon diversity for continuous variable.

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

```
## `geom_smooth()` using formula 'y ~ x'
```



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 < 0.1), ]
  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 < 0.1), ]
  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$padj, na.last = NA), ]
  sigtab = res[(res$padj < 0.1), ]
  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.

```r
# LRT
phyloseq_to_deseq2(ps3, ~ Primary_Group) %>%
    calc_geo_means() %>%
    deseq_filter(10,30) %>%
    DESeq(fitType = "local", test = "LRT", reduced = ~ 1) %>%
    get_deseq_res_lrt() %>%
    remove_rownames()
```

```r
# Wald
phyloseq_to_deseq2(ps3, ~ Primary_Group) %>%
    calc_geo_means() %>%
    deseq_filter(10,30) %>%
    DESeq(fitType = "local", test = "Wald") %>%
    get_deseq_res_cat("Primary_Group", "NICU", "SCN") %>%
    remove_rownames()
```

```
##   log2FoldChange    lfcSE       padj           Genus          Variable
## 1       3.913908 1.501674 0.06405668 Bifidobacterium Primary_GroupNICU
```