The UK Crop Microbiome Cryobank

Payton Yau

2023-08-10

The UK Crop Microbiome Cryobank

The UK Crop Microbiome Cryobank integrates genomic (DNA) data with a cryobank collection of samples for the soil microbiomes of the UK major crop plant systems. For this project, the microbiomes are from the rhizosphere (the soil surrounding the crop plant roots) and from bulk soil (soil outside the rhizosphere). The Cryobank provides a facility for researchers to source data and samples, including cryo-preserved microbial material, as well as genomic and metagenomic sequences from different soil microbiome environments.

An integrated cryopreserved collection of samples (rhizophere and bulk soil, bacterial and fungal isolates and DNA) from crop plant systems (barley, oats, oil seed rape, sugar beet and wheat) An open access AgMicrobiomeBase of microbiome data and associated meta-data linked to current public resources such as MGnify.

Qiime2 to Phyloseq Qiime2 is a powerful tool for microbial community analysis which we used for sequencing analysis; Phyloseq is a package in R that provides a flexible and comprehensive framework for the analysis of high-throughput sequencing data. To convert Qiime2 data to Phyloseq, users can use an external package called Qiime2R for the conversion. Qiime2R facilitates the transformation of Qiime2 data structures into phyloseq data structures within the R programming environment.

In various programming languages, including R, features are enhanced through external packages. These packages encompass a range of functions, datasets, and documentation that can be imported into R to extend its capabilities. External R packages are available from diverse sources such as CRAN (The Comprehensive R Archive Network), Bioconductor, GitHub, and more. The process of acquiring packages varies. After installation, a package must be loaded into the R session using the **library()** function before its functions can be utilised.

Once the data is imported, users can manipulate, analyse, and visualise the data using the various functions available in Phyloseq. This conversion allows users to take advantage of the many analysis tools available in R, such as ggplot2 for data visualisation, dplyr for data manipulation, and vegan for ecological community analysis.

```
# Download qiime2R from Github
# if (!requireNamespace("devtools", quietly = TRUE)){install.packages("devtools")}
# devtools::install_github("jbisanz/qiime2R")
library("qiime2R")

# Download phyloseq from Bioconductor
# if (!require("BiocManager", quietly = TRUE))
#   install.packages("BiocManager")
# BiocManager::install("phyloseq")
library("phyloseq")
```

```
# Convert qiime2 results to phyloseq format
physeq <- qza_to_phyloseq(
  features = "~/GitHub/agmicrobiomebase/16s/[Qiime2]Silva_138/428_228_220_table_silva138-with-phyla-no-m
  taxonomy = "~/GitHub/agmicrobiomebase/16s/[Qiime2]Silva_138/428_228_220_taxonomy_silva138.qza",
  metadata = "~/GitHub/agmicrobiomebase/16s/meta-table.txt"
  #, tree = "rooted-tree.qza"
)

physeq ## confirm the object
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 266974 taxa and 332 samples ]
## sample_data() Sample Data:       [ 332 samples by 17 sample variables ]
## tax_table()   Taxonomy Table:    [ 266974 taxa by 7 taxonomic ranks ]
```

Remove unwanted (failed and controls) samples before the normalisation Removing unwanted samples before normalisation is a common step in microbiome data analysis pipelines. In many cases, some samples may fail during sequencing or quality control, while others may be controls or blanks that are not of interest. These samples can introduce noise and bias in downstream analyses if not removed.

By removing the unwanted samples before normalization, the remaining samples can be normalised based on their true biological variation, allowing for more accurate comparisons between samples.

```
# sample_names(physeq)
# rank_names(physeq) # "Kingdom" "Phylum" "Class" "Order" "Family" "Genus" "Species"

## unwanted samples removel (including failed samples)
physeq.ori <- subset_samples(physeq, Analysis == "Include")

## remove object
rm(physeq)
```

Bata diversity Beta diversity is a measure of the differences or similarities between two or more microbial communities or samples. It quantifies the degree of variation in microbial composition among different samples. Beta diversity analysis can help identify patterns in the distribution of microbial communities and reveal relationships between different factors such as environment, host, and other ecological variables.

Non-Metric Multidimensional Scaling (NMDS) and Principal Coordinates Analysis (PCoA) are both ordination techniques used in analyzing beta diversity, which captures the variation in species composition between different samples or sites. However, they have some differences in terms of methodology and interpretation:

NMDS (Non-Metric Multidimensional Scaling) NMDS is a method that aims to represent the pairwise dissimilarities between samples in a lower-dimensional space while preserving the rank order of these dissimilarities. It is considered a non-metric method because it doesn't assume a linear relationship between the original dissimilarities and the distances in the ordination space. NMDS emphasizes preserving the relative ranking of dissimilarities, making it more suitable for cases where the distances between samples are not well-preserved in a lower-dimensional space. NMDS produces an ordination plot where the positions of samples are arranged based on their pairwise dissimilarities. The distances between points on the plot are not directly interpretable but reflect the similarities or dissimilarities between samples.

PCoA (Principal Coordinates Analysis) PCoA is also used to visualise and analyse beta diversity by representing samples in a lower-dimensional space. It is a type of metric multidimensional scaling. PCoA is based on a metric approach, which means it attempts to preserve the actual distances between samples as closely as possible in the lower-dimensional space. PCoA produces an ordination plot where the distances

between points reflect the actual dissimilarities between samples. Unlike NMDS, PCoA may not perform as well with non-linear or rank-based dissimilarity measures.

bata diversity - before normalisation

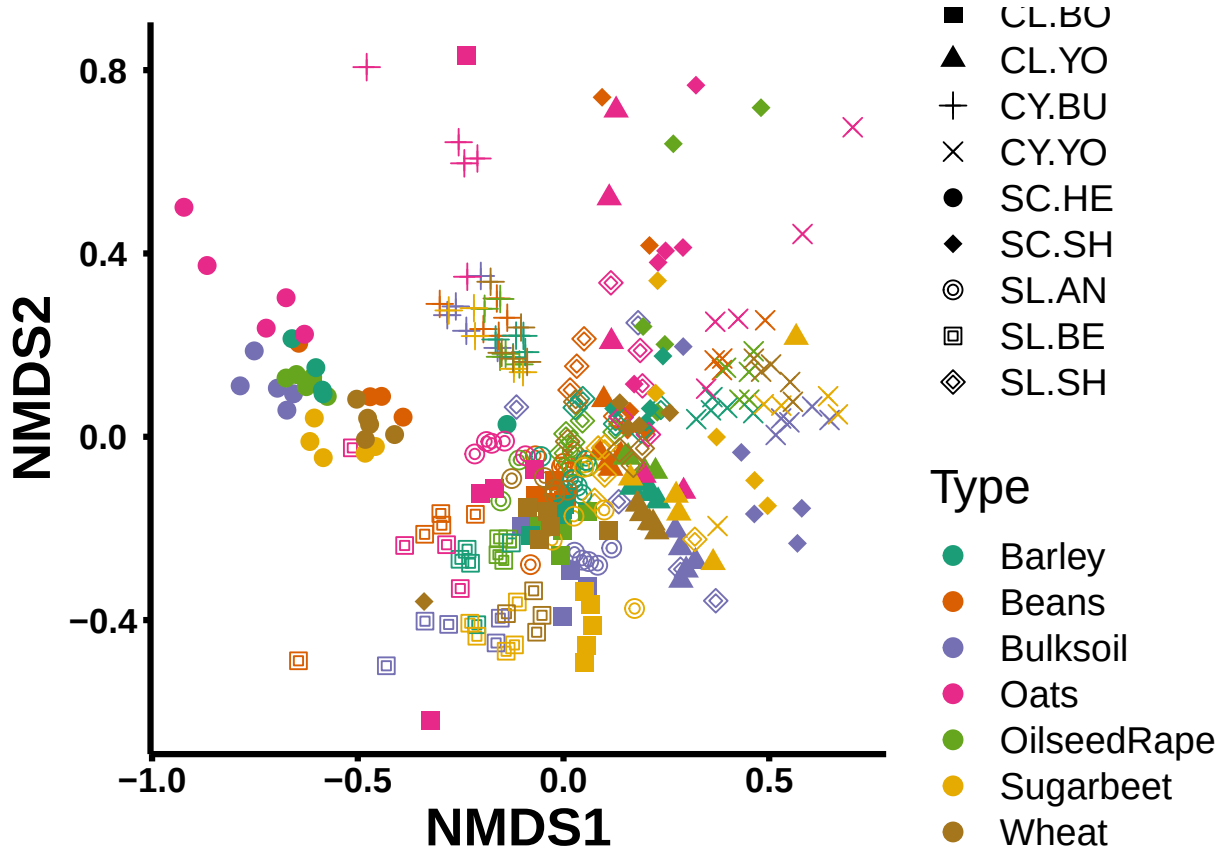
```
# install.packages("ggplot2")
library("ggplot2")
```

```
# Based on Type and Soil.Location
# method options: NMDS / PCoA
NMDS <- ordinate(physeq = physeq.ori, method = "NMDS", distance = "bray")
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.1872444
## Run 1 stress 0.1965942
## Run 2 stress 0.1882868
## Run 3 stress 0.1936358
## Run 4 stress 0.1907383
## Run 5 stress 0.1980526
## Run 6 stress 0.1956994
## Run 7 stress 0.2152232
## Run 8 stress 0.1970687
## Run 9 stress 0.1932784
## Run 10 stress 0.1917221
## Run 11 stress 0.2224317
## Run 12 stress 0.195442
## Run 13 stress 0.1908843
## Run 14 stress 0.1974398
## Run 15 stress 0.1901308
## Run 16 stress 0.1985515
## Run 17 stress 0.1943268
## Run 18 stress 0.19988
## Run 19 stress 0.1929727
## Run 20 stress 0.1930075
## *** Best solution was not repeated -- monoMDS stopping criteria:
##      2: no. of iterations >= maxit
##     13: stress ratio > sratmax
##      5: scale factor of the gradient < sfgrmin
```

```
# Plot ordination
# pdf(file = "bata_ori.pdf", width = 9,height = 8)
plot_ordination(
  physeq = physeq.ori,
  ordination = NMDS,
  color = "Type",
  shape = "Soil.Location"
) +
  theme_classic() +
  geom_point(aes(color = Type), alpha = 1, size = 3) +
  theme(
```

```
    text = element_text(size = 18, colour = "black"),
    axis.ticks = element_line(colour = "black", size = 1.1),
    axis.line = element_line(colour = 'black', size = 1.1),
    axis.text.x = element_text(colour = "black", angle = 0, hjust = 0.5, size = 13, face = "bold"),
    axis.text.y = element_text(colour = "black", angle = 0, hjust = 0.5, size = 13, face = "bold"),
    axis.title.y = element_text(color = "black", size = 20, face = "bold"),
    axis.title.x = element_text(color = "black", size = 20, face = "bold")
  ) +
  scale_color_brewer(palette = "Dark2") +
  scale_fill_brewer(palette = "Dark2") +
  scale_shape_manual(values = c(15, 17, 3, 4, 16, 18, 21, 22, 23)) # Set custom shapes
```




```
# Close the PDF device and save the plot to a file
# dev.off()
```

```
##### based on Plate and Type
# Plot ordination
# pdf(file = "bata_ori_plate.pdf", width = 9,height = 8)
plot_ordination(
  physeq = physeq.ori,
  ordination = pcoa,
  color = "Plate",
  shape = "Type"
) +
  theme_classic() +
```

```
  geom_point(aes(color = Plate), alpha = 1, size = 3) +
  theme(
    text = element_text(size = 18, colour = "black"),
    axis.ticks = element_line(colour = "black", size = 1.1),
    axis.line = element_line(colour = 'black', size = 1.1),
    axis.text.x = element_text(colour = "black", angle = 0, hjust = 0.5, size = 13, face = "bold"),
    axis.text.y = element_text(colour = "black", angle = 0, hjust = 0.5, size = 13, face = "bold"),
    axis.title.y = element_text(color = "black", size = 20, face = "bold"),
    axis.title.x = element_text(color = "black", size = 20, face = "bold")
  ) +
  scale_color_brewer(palette = "Dark2") +
  scale_fill_brewer(palette = "Dark2") +
  scale_shape_manual(values = c(15, 17, 3, 4, 16, 18, 21, 22, 23)) # Set custom shapes
```

```
## NULL
```

```
# Close the PDF device and save the plot to a file
# dev.off()
```

normalisation using ConQUR Constrained Quantile Normalisation (ConQUR) https://www.nature.com/articles/s41467-022-33071-9 is a normalisation technique used in high-throughput sequencing data, particularly in microbiome studies. It is a type of **quantile normalisation** approach that preserves the relative abundances of taxa between samples while simultaneously removing systematic technical variations that can arise due to differences in sequencing depth, PCR amplification bias, or other factors. ConQUR uses kernel density estimation to model the distribution of taxon abundances across all samples, and then constrains the normalisation process to maintain the relative position of each taxon within that distribution.

```
# devtools::install_github("wdl2459/ConQuR")
library(ConQuR)

# Download phyloseq from CRAN
# install.packages("doParallel")
library(doParallel)
```

```
# Convert ASV table to a data frame and transpose
B <- as.data.frame(physeq.ori@otu_table) # taxa
B <- t(B)
B <- as.data.frame(B)

# Extract batch ID from sample data
batchid = physeq.ori@sam_data$Plate # batchid

# Extract covariates
D = physeq.ori@sam_data[, c('Type', 'Soil', 'Location')] #covar
summary(D)
```

```
##           Type    Soil     Location
##  Barley     :44   CL: 68   AN:35
##  Beans      :44   CY: 69   BE:32
##  Bulksoil   :45   SC: 70   BO:35
##  Oats       :44   SL:102   BU:35
```

```
##  OilseedRape:43            HE:35
##  Sugarbeet  :45            SH:70
##  Wheat      :44            YO:67
```

```
# Correct for batch effects using ConQuR package
options(warn=-1) # required to call
taxa_correct1 = ConQuR(tax_tab = B, batchid = batchid, covariates = D, batch_ref="1") #  warning messag

# Transpose the corrected matrix and convert it to a data frame
taxa_correct2 <- t(taxa_correct1)
taxa_correct2 <- as.data.frame(taxa_correct2)

# Create new ASV table, taxonomy table, and sample data
ASV = otu_table(taxa_correct2, taxa_are_rows = TRUE)
TAXA = tax_table(physeq.ori)
sampledata = sample_data(physeq.ori)

# repack the objects into a level 4 phyloseq structural data
physeq.norm = phyloseq(ASV, TAXA, sampledata)

# remove
rm(B, D, batchid, taxa_correct1, taxa_correct2, ASV, TAXA, sampledata, to_skip)
```

```
# install.packages("tidyverse")
library("tidyverse")
```

```
## (1) Sub-samples for Sugarbeet and Bulk Soil groups for the comparisons
physeq.SU <- physeq.norm %>% subset_samples(Type %in% c("Sugarbeet", "Bulksoil"))

## (2) Subset the samples in the phyloseq object that belong to "Sugarbeet"
## or other crops "Barley", "Beans", "Bulksoil", "Oats", "OilseedRape", "Sugarbeet"
physeq.Sugarbeet <- subset_samples(physeq.norm, Type=="Sugarbeet")

## (3A) Merge the replicate samples for each Group
physeq.Sugarbeet.group = merge_samples(physeq.Sugarbeet, "Group") # Sum between replicate samples

# (3B) repair factors in the sample metadata
# sample_data(physeq.Sugarbeet.group)$Group <- levels(sample_data(physeq.norm)$Group)[get_variable(phys
 # or another option
sample_data(physeq.Sugarbeet.group)$Group <- rownames(sample_data(physeq.Sugarbeet.group))

sample_data(physeq.Sugarbeet.group)$Soil <- levels(sample_data(physeq.norm)$Soil)[get_variable(physeq.Su

## (4) Further subgroup for SU.CL.BO vs SU.CL.YO
physeq.Sugarbeet.vs <- physeq.Sugarbeet %>% subset_samples(Group %in% c("SU.CL.BO", "SU.CL.YO"))
```

Subgroupping

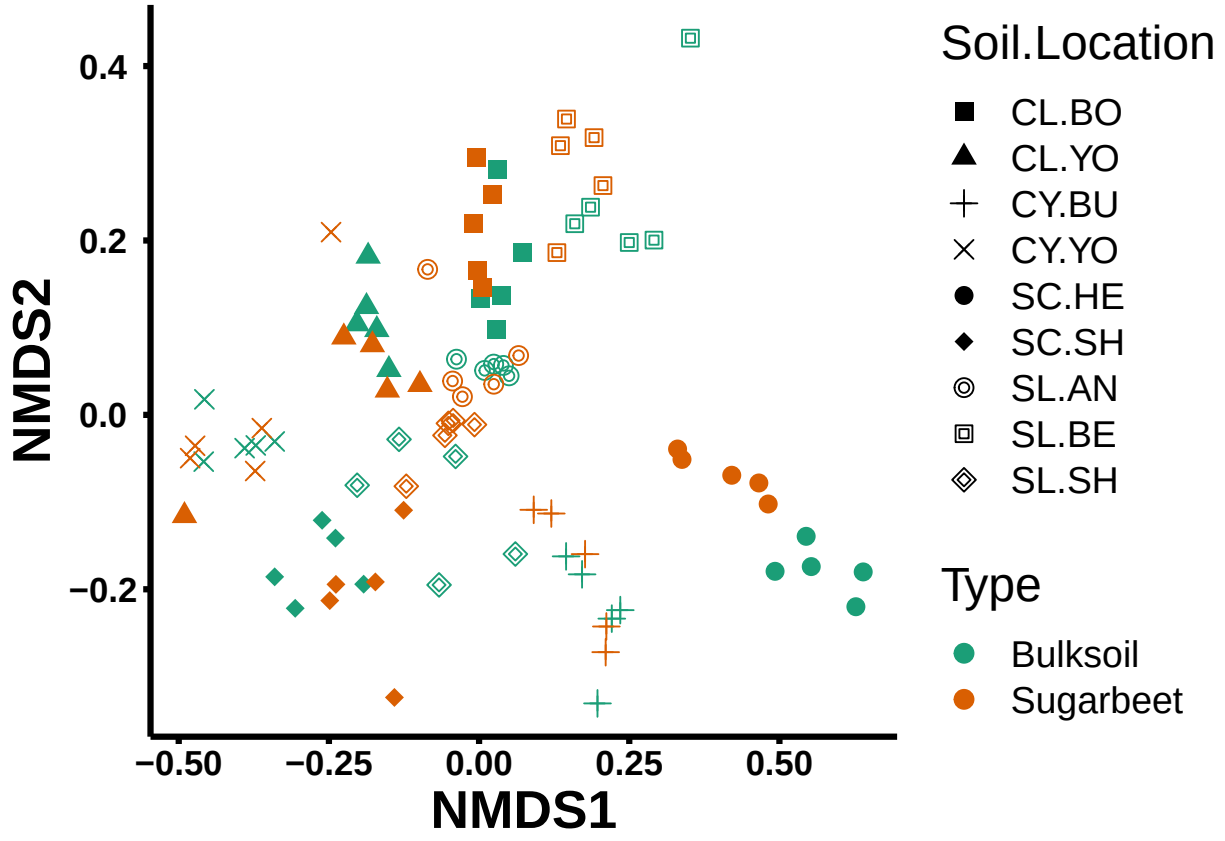
bata-diversity - after the normalisation

```
### based on Type and Soil.Location
# method options: NMDS / PCoA
pcoa <- ordinate(physeq = physeq.SU, method = "NMDS", distance = "bray")
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.1677301
## Run 1 stress 0.177103
## Run 2 stress 0.1661339
## ... New best solution
## ... Procrustes: rmse 0.03157689  max resid 0.150841
## Run 3 stress 0.1824044
## Run 4 stress 0.1679346
## Run 5 stress 0.1684949
## Run 6 stress 0.1702912
## Run 7 stress 0.1670884
## Run 8 stress 0.1836765
## Run 9 stress 0.1807332
## Run 10 stress 0.1668777
## Run 11 stress 0.1834423
## Run 12 stress 0.1818967
## Run 13 stress 0.1667808
## Run 14 stress 0.1701585
## Run 15 stress 0.1800431
## Run 16 stress 0.1790037
## Run 17 stress 0.1688595
## Run 18 stress 0.1688596
## Run 19 stress 0.1685488
## Run 20 stress 0.1806948
## *** Best solution was not repeated -- monoMDS stopping criteria:
##     17: stress ratio > sratmax
##      3: scale factor of the gradient < sfgrmin
```

```
# Plot ordination
# pdf(file = "bata_SU.pdf", width = 9,height = 8)
plot_ordination(
  physeq = physeq.SU,
  ordination = pcoa,
  color = "Type",
  shape = "Soil.Location"
) +
  theme_classic() +
  geom_point(aes(color = Type), alpha = 1, size = 3) +
  theme(
    text = element_text(size = 18, colour = "black"),
    axis.ticks = element_line(colour = "black", size = 1.1),
    axis.line = element_line(colour = 'black', size = 1.1),
    axis.text.x = element_text(colour = "black", angle = 0, hjust = 0.5, size = 13, face = "bold"),
    axis.text.y = element_text(colour = "black", angle = 0, hjust = 0.5, size = 13, face = "bold"),
    axis.title.y = element_text(color = "black", size = 20, face = "bold"),
```

```
    axis.title.x = element_text(color = "black", size = 20, face = "bold")
  ) +
  scale_color_brewer(palette = "Dark2") +
  scale_fill_brewer(palette = "Dark2") +
  scale_shape_manual(values = c(15, 17, 3, 4, 16, 18, 21, 22, 23)) # Set custom shapes
```



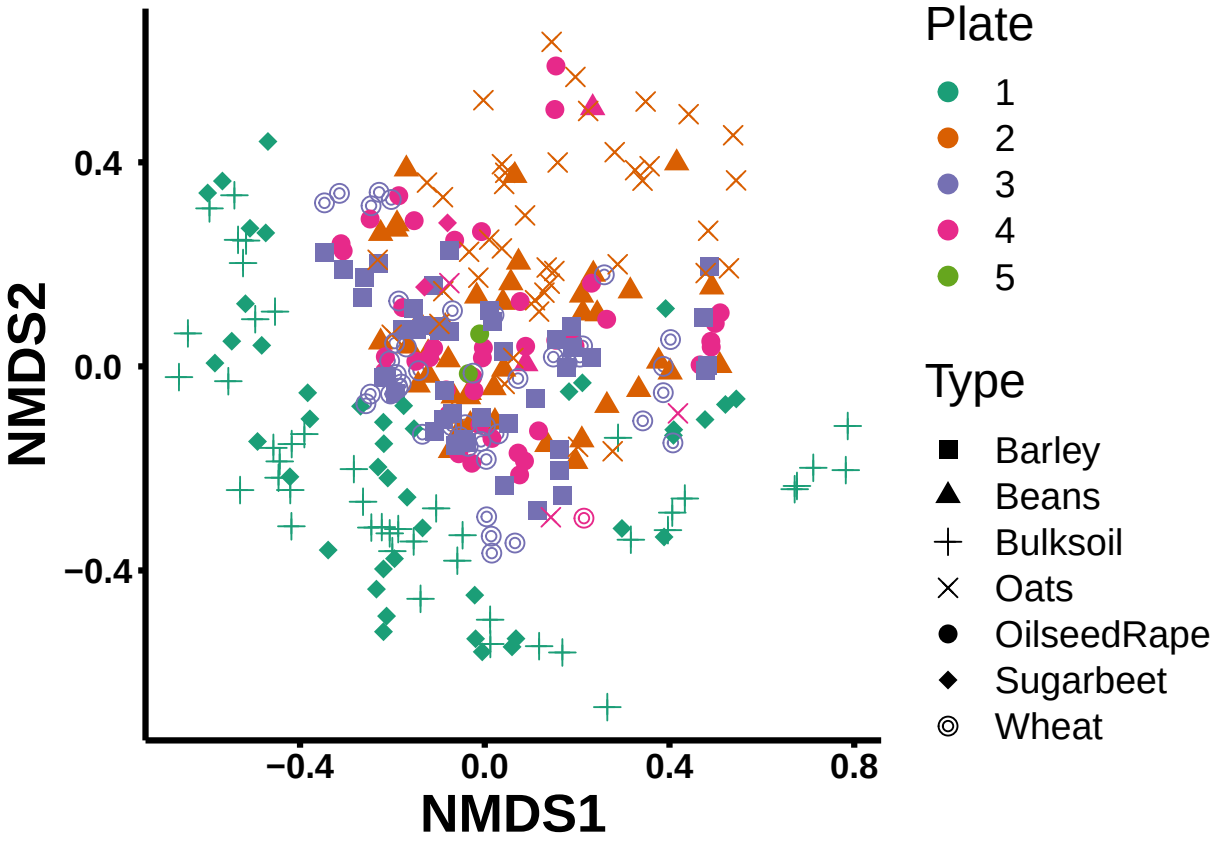

```
# Close the PDF device and save the plot to a file
# dev.off()

# Clean up by removing objects that are no longer needed
# rm(pcoa, physeq.SU)
```

```
#### Based on Plate and Type
# method options: NMDS / PCoA
NMDS <- ordinate(physeq = physeq.norm, method = "NMDS", distance = "bray")
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.2015008
## Run 1 stress 0.1989038
## ... New best solution
## ... Procrustes: rmse 0.01263709  max resid 0.08151516
## Run 2 stress 0.2015203
## Run 3 stress 0.1990112
```

```
## ... Procrustes: rmse 0.00487789  max resid 0.07287942
## Run 4 stress 0.2240136
## Run 5 stress 0.2052083
## Run 6 stress 0.250509
## Run 7 stress 0.200345
## Run 8 stress 0.1988978
## ... New best solution
## ... Procrustes: rmse 0.001081417  max resid 0.01810747
## Run 9 stress 0.1990114
## ... Procrustes: rmse 0.005402602  max resid 0.07335024
## Run 10 stress 0.200042
## Run 11 stress 0.2000202
## Run 12 stress 0.2295575
## Run 13 stress 0.2001903
## Run 14 stress 0.2137723
## Run 15 stress 0.2103611
## Run 16 stress 0.2067543
## Run 17 stress 0.1988681
## ... New best solution
## ... Procrustes: rmse 0.002692439  max resid 0.04377655
## Run 18 stress 0.1988683
## ... Procrustes: rmse 0.0001019873  max resid 0.001420499
## ... Similar to previous best
## Run 19 stress 0.2097036
## Run 20 stress 0.2089129
## *** Best solution repeated 1 times
```

```r
# Plot ordination
# pdf(file = "bata_norm_plate.pdf", width = 9,height = 8)
plot_ordination(
  physeq = physeq.norm,
  ordination = NMDS,
  color = "Plate",
  shape = "Type"
) +
  theme_classic() +
  geom_point(aes(color = Plate), alpha = 1, size = 3) +
  theme(
    text = element_text(size = 18, colour = "black"),
    axis.ticks = element_line(colour = "black", size = 1.1),
    axis.line = element_line(colour = 'black', size = 1.1),
    axis.text.x = element_text(colour = "black", angle = 0, hjust = 0.5, size = 13, face = "bold"),
    axis.text.y = element_text(colour = "black", angle = 0, hjust = 0.5, size = 13, face = "bold"),
    axis.title.y = element_text(color = "black", size = 20, face = "bold"),
    axis.title.x = element_text(color = "black", size = 20, face = "bold")
  ) +
  scale_color_brewer(palette = "Dark2") +
  scale_fill_brewer(palette = "Dark2") +
  scale_shape_manual(values = c(15, 17, 3, 4, 16, 18, 21, 22, 23)) # Set custom shapes
```




```
# Close the PDF device and save the plot to a file
# dev.off()
```

Alpha diversity Alpha diversity refers to the diversity of species within a single ecosystem or habitat. It is measured by analysing the number and distribution of species within a specific area or sample. Alpha diversity indices take into account the richness (number of species) and evenness (relative abundance of species) of a community. It provides insights into the complexity and stability of ecosystems. A high level of alpha diversity indicates a more complex ecosystem with a greater number of species, which is often associated with greater ecological resilience and stability. In contrast, a low level of alpha diversity can be an indicator of ecosystem disturbance, degradation, or vulnerability.

Please noted that different types of alpha diversity metrics capture various aspects of biodiversity within a specific community. Here's a brief explanation of each:

Observed: This metric simply counts the number of unique species (or operational taxonomic units) present in a sample. It provides a basic measure of species richness.

Chao1: Chao1 estimates the total number of species by considering the number of rare or singleton species. It takes into account the number of singletons (species observed only once) and doubletons (species observed only twice).

ACE (Abundance-based Coverage Estimator): Similar to Chao1, ACE also estimates species richness by accounting for rare species, but it also considers their abundance in the community.

Shannon Diversity Index: This index takes into account both species richness and evenness in the community. It considers the number of species present as well as their relative abundances.

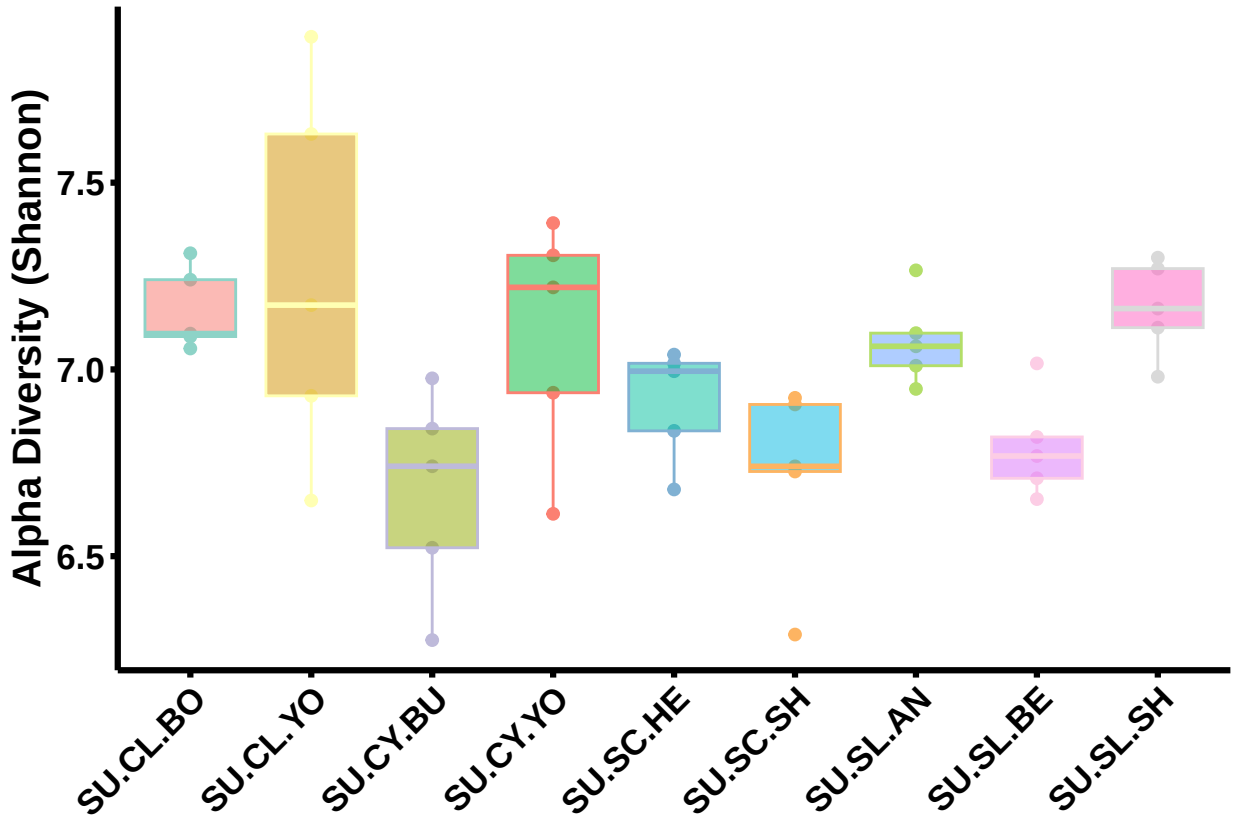
Simpson Diversity Index: Simpson's index gives more weight to dominant species in the community. It reflects the probability that two randomly selected individuals belong to different species.

Inverse Simpson Diversity Index (InvSimpson): This index is the reciprocal of the Simpson index and is useful for emphasizing the dominance of a few species.

Fisher's Alpha: Fisher's alpha is a measure of species richness that takes into account the distribution of individuals among species. It's particularly useful for comparing species diversity between different communities.

Here, we pick Shannon as an example for the work

```
# available measurements: "Observed" "Chao1" "ACE" "Shannon" "Simpson" "InvSimpson" "Fisher"
# Calculate alpha diversity (Shannon) and store it in alpha.object
alpha.object <- cbind(
  x = sample_data(physeq.Sugarbeet),
  y = estimate_richness(physeq.Sugarbeet, measures = 'Shannon')
)

# Create a plot for alpha diversity
ggplot(data = alpha.object, aes(x = x.Group, y = Shannon, color = x.Group, fill = x.Group)) +
  theme_classic() +
  labs(
    x = element_blank(),                    # No x-axis label
    y = "Alpha Diversity (Shannon)"         # y-axis label
  ) +
  geom_point(size = 1.75) +                 # Add points
  geom_boxplot(alpha = 0.5) +               # Add boxplot with transparency
  scale_color_brewer(palette = "Set3") +
  theme(
    text = element_text(size = 18, colour = "black"),
    axis.ticks = element_line(colour = "black", size = 1.1),
    axis.line = element_line(colour = 'black', size = 1.1),
    axis.text.x = element_text(colour = "black", angle = 45, hjust = 1, size = 13, face = "bold"),
    axis.text.y = element_text(angle = 0, hjust = 0, colour = "black", size = 13, face = "bold"),
    axis.title.y = element_text(color = "black", size = 15, face = "bold"),
    legend.position = "none"                # Hide legend
  )
```




```
# Clean up by removing the alpha.object
rm(alpha.object)
```

Calculate the number of taxa on each level and each group The purpose of this process is to visualise the distribution of the number of matched abundance across different groups and to identify any patterns in the distribution of the processed abundance within individual group.

```
# Create an empty list to store genus-level abundance data for each taxonomic level
gentab_levels <- list()

# Set observation threshold
observationThreshold <- 1

# Define the taxonomic levels
genus_levels <- c("Kingdom", "Phylum", "Class", "Order", "Family", "Genus", "Species")

# loop through all the taxonomic levels
for (level in genus_levels) {

  # create a factor variable for each level
  genfac <- factor(tax_table(physeq.Sugarbeet.group)[, level])

  # calculate the abundance of each genus within each sample
  gentab <- apply(otu_table(physeq.Sugarbeet.group), MARGIN = 1, function(x) {
    tapply(x, INDEX = genfac, FUN = sum, na.rm = TRUE, simplify = TRUE)
```

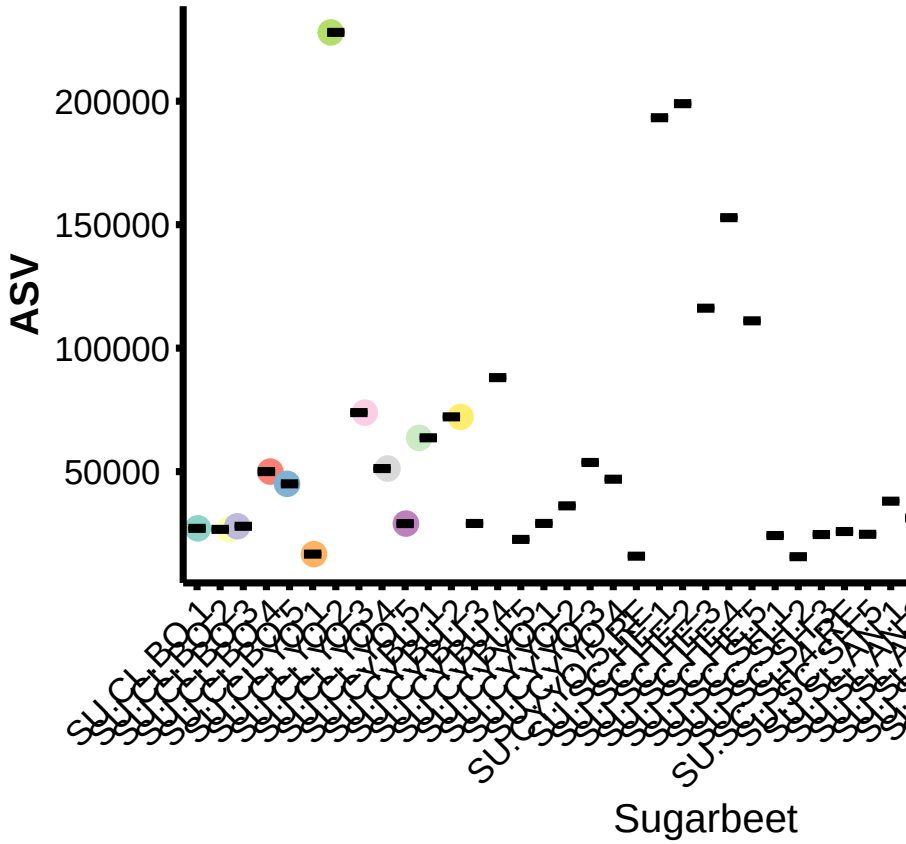
```
  })

  # calculate the number of samples in which each genus is observed above the threshold
  level_counts <- apply(gentab > observationThreshold, 2, sum)

  # create a data frame of level counts with genus names as row names
  BB <- as.data.frame(level_counts)
  BB$name <- row.names(BB)

  # add the data frame to the gentab_levels list
  gentab_levels[[level]] <- BB
}

# Combine all level counts data frames into one data frame
B2 <- gentab_levels %>% reduce(full_join, by = "name")

# Set row names and column names
rownames(B2) <- B2$name
B2$name <- NULL
colnames(B2)[1:7] <- genus_levels

# Print the resulting data frame
print(B2)
```

```
##          Kingdom Phylum Class Order Family Genus Species
## SU.CL.BO       2     40   120   265    389   593     304
## SU.CL.YO       2     44   127   304    441   716     386
## SU.CY.BU       2     36   101   235    344   549     277
## SU.CY.YO       2     42   121   282    408   626     313
## SU.SC.HE       2     38   107   245    361   548     306
## SU.SC.SH       2     38   109   234    323   476     220
## SU.SL.AN       2     40   120   272    400   598     289
## SU.SL.BE       2     41   112   250    356   532     254
## SU.SL.SH       2     38   112   254    370   569     300
```

```
# Clean up by removing unnecessary objects
rm(gentab_levels, BB)
```

```
# Calculate the sum of ASVs across samples and create a data frame
ASV <- sample_sums(physeq.Sugarbeet)
ASV <- as.data.frame(ASV)
ASV$Sugarbeet <- rownames(ASV)

# Remove the suffix from the ASV names
new_names <- sub("-\\d+$", "", ASV$Sugarbeet)

# Update the row names of the data frame
ASV$Sugarbeet <- new_names

# Create a scatter plot with jittered points and overlaid boxplots
```

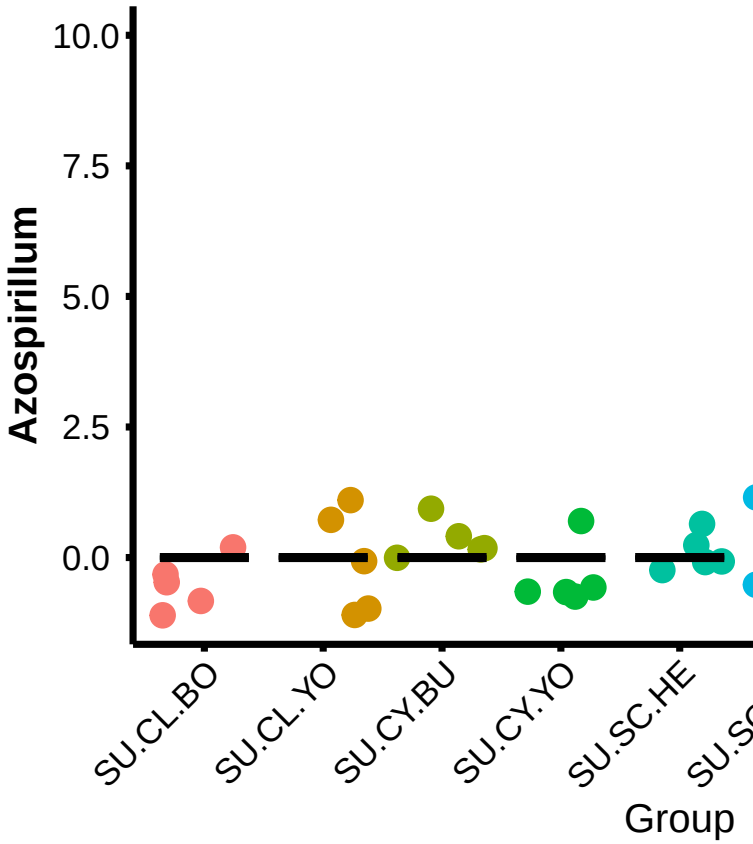
```
ggplot(ASV, aes(x = Sugarbeet, y = ASV,  colour = Sugarbeet)) +
  geom_point(alpha = 1, position = "jitter", size = 4) +
  geom_boxplot(alpha = 0, colour = "black", size = 0.8) +
  theme_classic() + scale_color_brewer(palette = "Set3") +
  theme(text = element_text(size=15, colour = "black"),
        axis.ticks = element_line(colour = "black", size = 1.25),
        axis.line = element_line(colour = 'black', size = 1.25),
        axis.text.x = element_text(angle=45, hjust=1, colour = "black", size = 13),
        axis.text.y = element_text(angle=0, hjust=0.5, colour = "black", size = 13),
        axis.title.y = element_text(color="black", size=15, face="bold"),
        legend.position = "none")
```



Calculate the sum of ASV (Sugarbeet)

```
# Subset the taxa to Genus from physeq.Sugarbeet
physeq.Sugarbeet.genus <- subset_taxa(physeq.Sugarbeet, Genus == "Azospirillum")

# Calculate the total abundance of Azospirillum for each sample
meta = physeq.Sugarbeet.genus@sam_data
otudf = as.data.frame(t(as.data.frame(physeq.Sugarbeet.genus@otu_table)))
meta$Azospirillum = rowSums(otudf)

# Plot a graph of the abundance of Azospirillum for each sample grouped by Group:
ggplot(subset(meta, Group %in% c("SU.CL.BO","SU.CL.YO",
```

```
                                  "SU.CY.BU","SU.CY.YO",
                                  "SU.SC.HE","SU.SC.SH",
                                  "SU.SL.AN","SU.SL.BE",
                                  "SU.SL.SH")),
       aes(x = Group, y = Azospirillum,  colour = interaction(Group))) +
  geom_point(alpha = 1, position = "jitter", size = 4) +
  geom_boxplot(alpha = 0, colour = "black", size = 0.8)+
  theme_classic() +
  theme(text = element_text(size=15, colour = "black"),
        axis.ticks = element_line(colour = "black", size = 1.25),
        axis.line = element_line(colour = 'black', size = 1.25),
        axis.text.x = element_text(angle=45, hjust=1, colour = "black", size = 13),
        axis.text.y = element_text(angle=0, hjust=0.5, colour = "black",size = 13),
        axis.title.y = element_text(color="black", size=15,face="bold"), legend.position = "none")
```



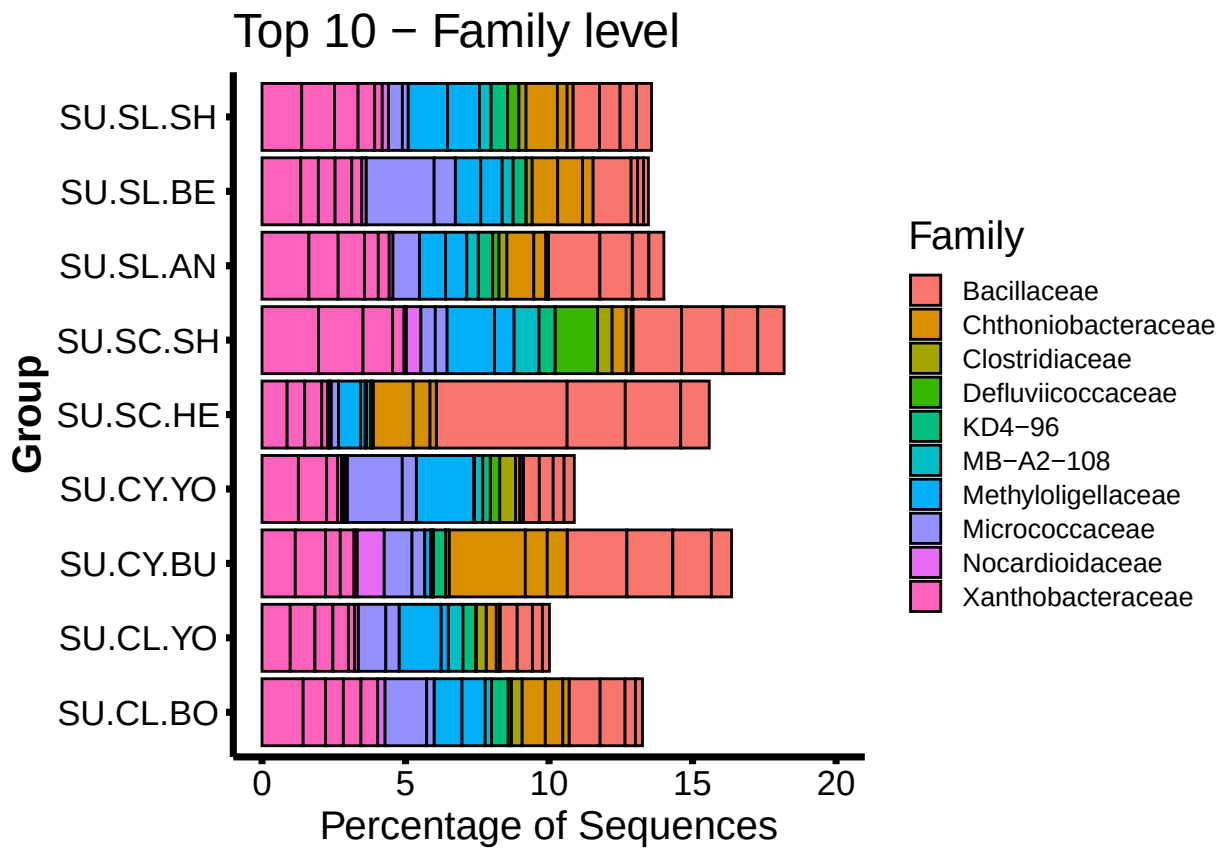
Plotting for the abundance of one specific bacteria

```
# Clean up by removing unnecessary objects
rm(physeq.Sugarbeet.genus, meta, otudf)
```

Plotting the top 10 taxa at family level

```
## Transform normalised ASVs to proportions
proportions = transform_sample_counts(physeq.Sugarbeet.group, function(x) 100 * x/sum(x))
```

```r
##
Top10ASVs = names(sort(taxa_sums(proportions), TRUE)[1:21])
Taxtab10 = cbind(tax_table(proportions), Family10 = NA)
Taxtab10[Top10ASVs, "Family10"] <- as(tax_table(proportions)[Top10ASVs, "Family"], "character")
tax_table(proportions) <- tax_table(Taxtab10)

Rgsm10 = prune_taxa(Top10ASVs, proportions)

# plotting
title = "Top 10 - Family level"
plot_bar(Rgsm10, "Group", fill = "Family", title = title) +
  coord_flip() +
  ylab("Percentage of Sequences") + ylim(0, 20) +
  scale_color_brewer(palette = "Set3") +
  theme_classic() +
  theme(text = element_text(size=15, colour = "black"),
        axis.ticks = element_line(colour = "black", size = 1.25),
        axis.line = element_line(colour = 'black', size = 1.25),
        axis.text.x = element_text(angle=0, hjust=0.5, colour = "black", size = 13),
        axis.text.y = element_text(angle=0, hjust=0.5, colour = "black",size = 13),
        axis.title.y = element_text(color="black", size=15,face="bold"),
        legend.position = "right",
        legend.text = element_text(size = 9.5),
        legend.key.height= unit(0.45, 'cm'),
        legend.key.width= unit(0.45, 'cm')
        )
```




```
# Clean up by removing unnecessary objects
rm(proportions, Top10ASVs, Taxtab10, Rgsm10, title)
```

Pairwise comparison using PERMANOVA Pairwise PERMANOVA is a statistical method used to compare multiple groups or treatments in ecological and microbial community studies. It assesses dissimilarity between samples and provides a p-value to determine the significance of observed differences. This approach is valuable for targeted group comparisons, allowing researchers to investigate the effects of specific factors on microbial communities and uncover significant variations in community composition. By considering within- and between-group variation, pairwise PERMANOVA provides robust statistical analysis and insights into microbial community dynamics and functioning.

```
# devtools::install_github("pmartinezarbizu/pairwiseAdonis/pairwiseAdonis")
library("pairwiseAdonis")
library("GGally")
```

```
metdat = as.data.frame(as.matrix(physeq.Sugarbeet@sam_data))
dat = as.data.frame(t(as.data.frame(physeq.Sugarbeet@otu_table)))
perma = pairwise.adonis(dat, metdat$Group, sim.function = "vegdist",
                sim.method = "bray", p.adjust.m = "BH",
                reduce = NULL, perm = 100000)

# print(perma)
# Keep only the necessary columns
df <- perma[, c("pairs", "p.adjusted")]
```

```r
# Split the "pairs" column into two separate columns
df_split <- strsplit(df$pairs, " vs ")
df$A <- sapply(df_split, "[[", 1)
df$B <- sapply(df_split, "[[", 2)

# Create a matrix
cor_matrix <- matrix(NA, nrow = length(unique(df$A)), ncol = length(unique(df$B)))
rownames(cor_matrix) <- unique(df$A)
colnames(cor_matrix) <- unique(df$B)

# Fill in the correlation matrix with p.adjusted sig values
for (i in 1:nrow(cor_matrix)) {
  for (j in 1:ncol(cor_matrix)) {
    a <- rownames(cor_matrix)[i]
    b <- colnames(cor_matrix)[j]
    value <- df[df$A == a & df$B == b, "p.adjusted"]
    if (length(value) > 0) {
      cor_matrix[i, j] <- value
    }
  }
}

# Clean up by removing unnecessary objects generated from the loop
rm(a, b, i, j, value)

print(cor_matrix)
```

```
##             SU.CL.YO    SU.CY.BU    SU.CY.YO    SU.SC.HE    SU.SC.SH
## SU.CL.BO 0.008876483 0.008876483 0.008876483 0.008876483 0.008876483
## SU.CL.YO          NA 0.008876483 0.055209448 0.008876483 0.008876483
## SU.CY.BU          NA          NA 0.008876483 0.008876483 0.008876483
## SU.CY.YO          NA          NA          NA 0.008876483 0.008876483
## SU.SC.HE          NA          NA          NA          NA 0.008876483
## SU.SC.SH          NA          NA          NA          NA          NA
## SU.SL.AN          NA          NA          NA          NA          NA
## SU.SL.BE          NA          NA          NA          NA          NA
##             SU.SL.AN    SU.SL.BE    SU.SL.SH
## SU.CL.BO 0.008876483 0.008876483 0.008876483
## SU.CL.YO 0.008876483 0.008876483 0.008876483
## SU.CY.BU 0.008876483 0.008876483 0.008876483
## SU.CY.YO 0.008876483 0.008876483 0.008876483
## SU.SC.HE 0.008876483 0.008876483 0.008876483
## SU.SC.SH 0.008876483 0.008876483 0.008876483
## SU.SL.AN          NA 0.008876483 0.008876483
## SU.SL.BE          NA          NA 0.008876483
```

```r
# Clean up by removing unnecessary objects
rm(metdat, dat, perma, df, cor_matrix)
```

Upset plot using UpsetR

When it comes to representing sets visually, the go-to option is usually a Venn diagram. These diagrams work well when dealing with up to five sets, providing a clear visualisation. However, as the dataset expands, such as when dealing with five sets, deriving the desired insights from the diagram becomes more complex. As a result, considering an UpSet graph for data visualisation becomes an appealing choice. UpSet graphs offer a more streamlined way to display intersections and complements, particularly when dealing with larger datasets or multiple sets. This option ensures a more intuitive and informative representation of the data.

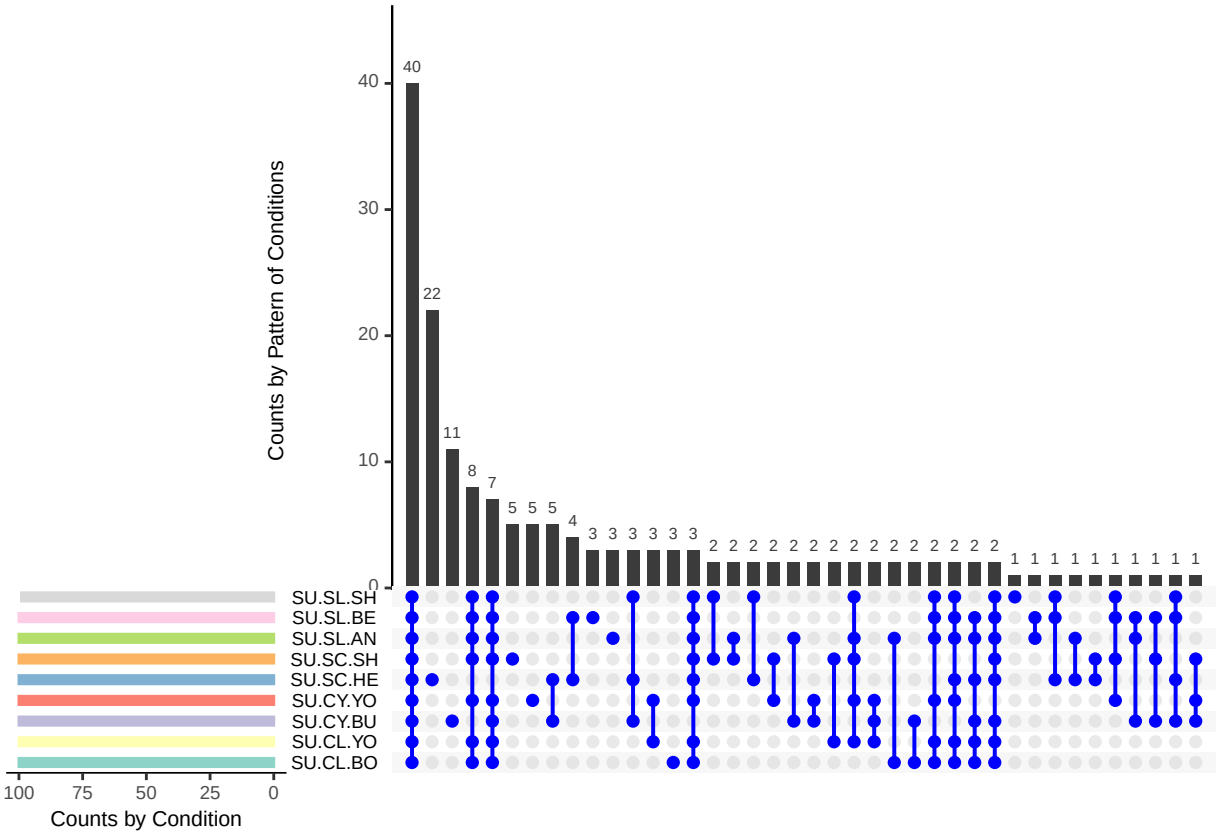
```
# BiocManager::install("microbiome")
library("microbiome")

# devtools::install_github("mikemc/speedyseq")
library("speedyseq")

library("UpSetR")
library("plyr")
library("reshape2")
# install.packages("RColorBrewer")
library("RColorBrewer")
```

```
# Aggregate taxa at the genus level
B <- aggregate_taxa(physeq.Sugarbeet.group, "Genus", verbose = TRUE)
```

```
## [1] "Remove taxonomic information below the target level"
## [1] "Mark the potentially ambiguous taxa"
## [1] "-- split"
## [1] "-- sum"
## [1] "Create phyloseq object"
## [1] "Remove ambiguous levels"
## [1] "-- unique"
## [1] "-- Rename the lowest level"
## [1] "-- rownames"
## [1] "-- taxa"
## [1] "Convert to taxonomy table"
## [1] "Combine OTU and Taxon matrix into Phyloseq object"
## [1] "Add the metadata as is"
```

```
# Remove undesired genera
# B2 <- subset_taxa(B, !get("Genus") %in% c("uncultured", "Unknown"))

# Remove unwanted taxon names
taxa_to_remove <- c("uncultured", "Unknown")
B2 <- subset_taxa(B, !get("Genus") %in% taxa_to_remove)

# Convert to tibble, rename columns, select relevant columns, group by Sample, and keep top 100 abundan
# Convert data to a tibble and perform necessary operations
D <- as_tibble(B2) %>%
  mutate(Sample = .sample, ASV = .otu, Abundance = .abundance) %>%
  select(Sample, Abundance, Genus) %>%
  group_by(Sample) %>%
  filter(rank(desc(Abundance)) <= 100) %>% # Filter <= 100
  ungroup()
```

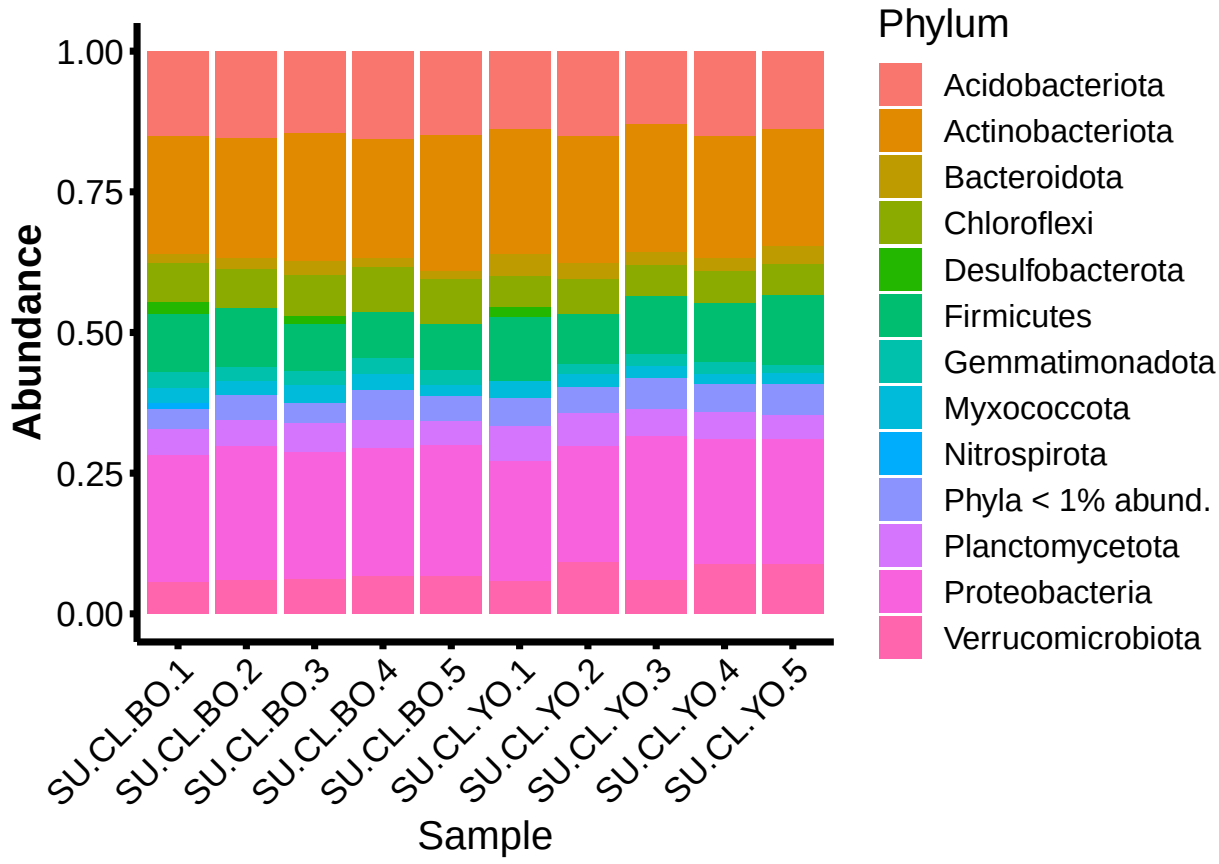
```r
# Remove the Abundance column
D$Abundance <- NULL

# Rename the second column to "ASV"
names(D)[2] <- "ASV"

# Convert data from long to wide format
E <- dcast(D, ASV ~ Sample)

# Define a binary function
binary_fun <- function(x) {
  x[is.na(x)] <- 0
  ifelse(x > 0, 1, 0)
}

col = brewer.pal(n = 9, name = "Set3")

# Apply the binary function to columns 2 to 10
temp_df <- apply(E[2:10], 2, binary_fun)
temp_df <- as.data.frame(temp_df)
rownames(temp_df) = E$ASV

# Create an UpSet plot
upset_plot <- upset(temp_df,
                    sets = colnames(temp_df),
                    sets.bar.color = (col),
                    order.by = "freq",
                    empty.intersections = "on",
                    mainbar.y.label = "Counts by Pattern of Conditions",
                    sets.x.label = "Counts by Condition",
                    matrix.color="blue",
                    point.size=2
                    )

# Print the UpSet plot
print(upset_plot)
```




```
# Clean up by removing unnecessary objects
# rm(B, B2, D, E, binary_fun, temp_df, upset_plot, physeq.Sugarbeet.group)
```

```
# https://github.com/joey711/phyloseq/issues/901
# Filter taxa with mean abundance > 0.1
physeq2 = filter_taxa(physeq.Sugarbeet.vs, function(x) mean(x) > 0.1, TRUE)

# Transform sample counts to relative abundances
physeq3 = transform_sample_counts(physeq2, function(x) x / sum(x))

# Aggregate taxa at the Phylum level
glom <- tax_glom(physeq3, taxrank = 'Phylum')

# Convert the aggregated phyloseq object to a dataframe
data <- psmelt(glom)
# Convert Phylum column to character
data$Phylum <- as.character(data$Phylum)

# Rename phyla with abundance < 1% to "< 1% abund."
data$Phylum[data$Abundance < 0.01] <- "< 1% abund."

# Calculate medians of abundance for each Phylum
medians <- ddply(data, ~Phylum, function(x) c(median = median(x$Abundance)))

# Extract phyla with median abundance <= 0.01
remainder <- medians[medians$median <= 0.01, ]$Phylum
```

```r
# Rename selected phyla to "Phyla < 1% abund."
data[data$Phylum %in% remainder, ]$Phylum <- "Phyla < 1% abund."

# Rename phyla with abundance < 1% to "Phyla < 1% abund."
data$Phylum[data$Abundance < 0.01] <- "Phyla < 1% abund."

# Plot the data with condensed phyla categories
p <- ggplot(data = data, aes(x = Sample, y = Abundance, fill = Phylum))
p + geom_bar(aes(), stat = "identity", position = "stack") +
    scale_color_brewer(palette = "Paired") +
  theme_classic() +
  theme(text = element_text(size=15, colour = "black"),
        axis.ticks = element_line(colour = "black", size = 1.25),
        axis.line = element_line(colour = 'black', size = 1.25),
        axis.text.x = element_text(angle=45, hjust=1, colour = "black", size = 13),
        axis.text.y = element_text(angle=0, hjust=0.5, colour = "black",size = 13),
        axis.title.y = element_text(color="black", size=15,face="bold"), legend.position = "right") # +
```




```r
rm(p, data, remainder, medians,glom, physeq3, physeq2)
```

Linear discriminant analysis effect size (LEFSe) Linear discriminant analysis effect size (LEFSe) is a powerful computational tool commonly used in microbiome analysis to identify microbial biomarkers that are differentially abundant between different experimental groups. Microbiome analysis involves studying the composition and diversity of microbial communities present in various environments, such as the human

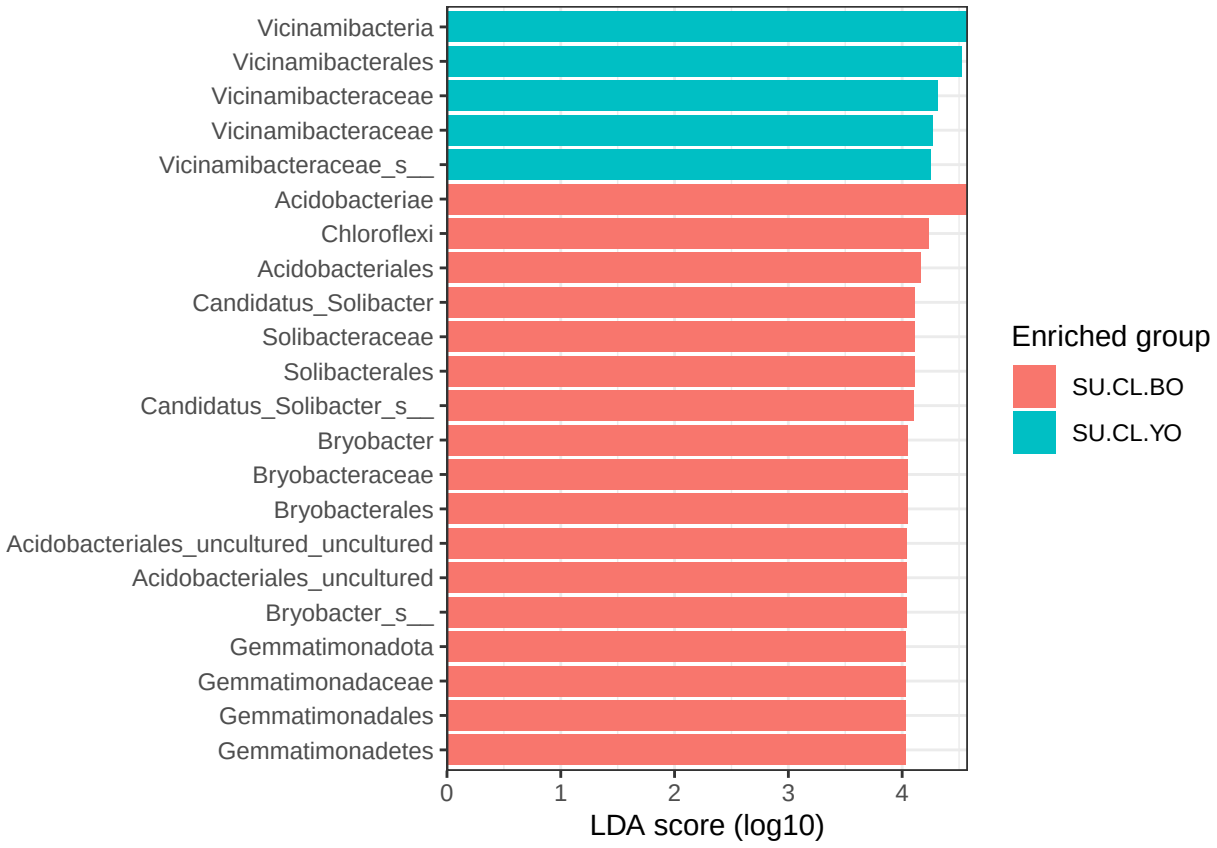
gut, soil, water, and more. LEFSe is specifically designed to address the challenge of identifying microbial taxa that drive differences between different experimental conditions or groups.

LEFSe combines linear discriminant analysis (LDA), which is a statistical technique used for classification and feature selection, with the concept of effect size. Effect size refers to the magnitude of a difference between two groups and provides insights into the biological relevance of the observed differences.

P,S.ALDEx2 and ANCOM-II produce the most consistent results (https://www.nature.com/articles/s41467-022-28034-z)

```
BiocManager::install("microbiomeMarker")
library("microbiomeMarker")
```

```
lef_out <- run_lefse(physeq.Sugarbeet.vs, group = "Group",
                     norm = "CPM",
                     # taxa_rank = "Genus",
                     kw_cutoff = 0.01, lda_cutoff = 4)
plot_ef_bar(lef_out)
```



Visualising the marker table as cladogram The default is 4 which is showing the kingdom, Phylum, Class on the cladogram.

```
# plot_cladogram(lef_out, color = c("red","blue"),
#                clade_label_level = 6)
# rm(lef_out)
```

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 11 x64 (build 22621)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.utf8
## [2] LC_CTYPE=English_United Kingdom.utf8
## [3] LC_MONETARY=English_United Kingdom.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.utf8
##
## time zone: Europe/London
## tzcode source: internal
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] microbiomeMarker_1.6.0 RColorBrewer_1.1-3     reshape2_1.4.4
##  [4] plyr_1.8.8             UpSetR_1.4.0           speedyseq_0.5.3.9018
##  [7] microbiome_1.22.0      GGally_2.1.2           pairwiseAdonis_0.4.1
## [10] cluster_2.1.4          vegan_2.6-4            lattice_0.21-8
## [13] permute_0.9-7          lubridate_1.9.2        forcats_1.0.0
## [16] stringr_1.5.0          dplyr_1.1.3            purrr_1.0.2
## [19] readr_2.1.4            tidyr_1.3.0            tibble_3.2.1
## [22] tidyverse_2.0.0        doParallel_1.0.17      iterators_1.0.14
## [25] foreach_1.5.2          ConQuR_2.0             ggplot2_3.4.3
## [28] phyloseq_1.44.0        qiime2R_0.99.6
##
## loaded via a namespace (and not attached):
##   [1] IRanges_2.34.1                gld_2.6.6
##   [3] nnet_7.3-19                   DT_0.29
##   [5] Biostrings_2.68.1             TH.data_1.1-2
##   [7] vctrs_0.6.3                   energy_1.7-11
##   [9] digest_0.6.33                 png_0.1-8
##  [11] shape_1.4.6                   proxy_0.4-27
##  [13] Exact_3.2                     ggrepel_0.9.3
##  [15] MASS_7.3-60                   reshape_0.8.9
##  [17] BiocGenerics_0.46.0           withr_2.5.0
##  [19] xfun_0.40                     ggfun_0.1.3
##  [21] survival_3.5-5                doRNG_1.8.6
##  [23] memoise_2.0.1                 ggbeeswarm_0.7.2
##  [25] MatrixModels_0.5-2            gmp_0.7-2
##  [27] tidytree_0.4.5                zoo_1.8-12
##  [29] GlobalOptions_0.1.2           gtools_3.9.4
##  [31] DEoptimR_1.1-2                Formula_1.2-5
##  [33] httr_1.4.7                    rhdf5filters_1.12.1
```

```
##  [35] rhdf5_2.44.0                   rstudioapi_0.15.0
##  [37] generics_0.1.3                 base64enc_0.1-3
##  [39] S4Vectors_0.38.1               zlibbioc_1.46.0
##  [41] ScaledMatrix_1.8.1             randomForest_4.7-1.1
##  [43] statip_0.2.3                   GenomeInfoDbData_1.2.10
##  [45] ade4_1.7-22                    evaluate_0.21
##  [47] S4Arrays_1.0.6                 hms_1.1.3
##  [49] glmnet_4.1-8                   GenomicRanges_1.52.0
##  [51] irlba_2.3.5.1                  colorspace_2.1-0
##  [53] ROCR_1.0-11                    readxl_1.4.3
##  [55] magrittr_2.0.3                 viridis_0.6.4
##  [57] ggtree_3.8.2                   robustbase_0.99-0
##  [59] SparseM_1.81                   DECIPHER_2.28.0
##  [61] scuttle_1.10.2                 matrixStats_1.0.0
##  [63] class_7.3-22                   Hmisc_5.1-1
##  [65] pillar_1.9.0                   nlme_3.1-162
##  [67] decontam_1.20.0                plotROC_2.3.0
##  [69] caTools_1.18.2                 compiler_4.3.1
##  [71] beachmat_2.16.0                stringi_1.7.12
##  [73] biomformat_1.28.0              DescTools_0.99.50
##  [75] stabledist_0.7-1               minqa_1.2.6
##  [77] SummarizedExperiment_1.30.2    crayon_1.5.2
##  [79] abind_1.4-5                    scater_1.28.0
##  [81] truncnorm_1.0-9                timeSeries_4031.107
##  [83] gridGraphics_0.5-1             cqrReg_1.2.1
##  [85] locfit_1.5-9.8                 bit_4.0.5
##  [87] mia_1.8.0                      rootSolve_1.8.2.3
##  [89] sandwich_3.0-2                 codetools_0.2-19
##  [91] multcomp_1.4-25                BiocSingular_1.16.0
##  [93] e1071_1.7-13                   lmom_3.0
##  [95] GetoptLong_1.0.5               multtest_2.56.0
##  [97] MultiAssayExperiment_1.26.0    metagenomeSeq_1.42.0
##  [99] splines_4.3.1                  circlize_0.4.15
## [101] Rcpp_1.0.11                    fastDummies_1.7.3
## [103] quantreg_5.97                  sparseMatrixStats_1.12.2
## [105] cellranger_1.1.0               knitr_1.44
## [107] blob_1.2.4                     utf8_1.2.3
## [109] clue_0.3-64                    lme4_1.1-34
## [111] fBasics_4022.94                checkmate_2.2.0
## [113] DelayedMatrixStats_1.22.6      Rdpack_2.5
## [115] expm_0.999-7                   gsl_2.1-8
## [117] ggplotify_0.1.2                Matrix_1.6-1.1
## [119] statmod_1.5.0                  tzdb_0.4.0
## [121] pkgconfig_2.0.3                tools_4.3.1
## [123] cachem_1.0.8                   rbibutils_2.2.15
## [125] RSQLite_2.3.1                  viridisLite_0.4.2
## [127] NADA_1.6-1.1                   DBI_1.1.3
## [129] numDeriv_2016.8-1.1            rmutil_1.1.10
## [131] fastmap_1.1.1                  rmarkdown_2.24
## [133] scales_1.2.1                   grid_4.3.1
## [135] patchwork_1.1.3                stable_1.1.6
## [137] BiocManager_1.30.22            rpart_4.1.19
## [139] farver_2.1.1                   mgcv_1.8-42
## [141] yaml_2.3.7                     MatrixGenerics_1.12.3
```

```
## [143] spatial_7.3-16                 foreign_0.8-84
## [145] bayesm_3.1-5                   cli_3.6.1
## [147] stats4_4.3.1                   lifecycle_1.0.3
## [149] Biobase_2.60.0                 mvtnorm_1.2-3
## [151] backports_1.4.1                modeest_2.4.0
## [153] BiocParallel_1.34.2            timechange_0.2.0
## [155] gtable_0.3.4                   rjson_0.2.21
## [157] ANCOMBC_2.2.1                  limma_3.56.2
## [159] ape_5.7-1                      CVXR_1.0-11
## [161] jsonlite_1.8.7                 bitops_1.0-7
## [163] bit64_4.0.5                    Rtsne_0.16
## [165] yulab.utils_0.0.9              BiocNeighbors_1.18.0
## [167] TreeSummarizedExperiment_2.8.0 timeDate_4022.108
## [169] lazyeval_0.2.2                 zCompositions_1.4.1
## [171] htmltools_0.5.6                glue_1.6.2
## [173] Wrench_1.18.0                  XVector_0.40.0
## [175] RCurl_1.98-1.12                treeio_1.24.3
## [177] gridExtra_2.3                  boot_1.3-28.1
## [179] igraph_1.5.1                   R6_2.5.1
## [181] DESeq2_1.40.2                  SingleCellExperiment_1.22.0
## [183] gplots_3.1.3                   Rmpfr_0.9-3
## [185] labeling_0.4.3                 rngtools_1.5.2
## [187] Rhdf5lib_1.22.1                aplot_0.2.1
## [189] GenomeInfoDb_1.36.3            nloptr_2.0.3
## [191] compositions_2.0-6             DirichletMultinomial_1.42.0
## [193] DelayedArray_0.26.7            tidyselect_1.2.0
## [195] vipor_0.4.5                    htmlTable_2.4.1
## [197] tensorA_0.36.2                 inline_0.3.19
## [199] GUniFrac_1.8                   rsvd_1.0.5
## [201] munsell_0.5.0                  KernSmooth_2.23-21
## [203] data.table_1.14.8              htmlwidgets_1.6.2
## [205] ComplexHeatmap_2.16.0          rlang_1.1.1
## [207] lmerTest_3.1-3                 fansi_1.0.4
## [209] beeswarm_0.4.0
```