The UK Crop Microbiome Cryobank

Payton Yau

2024-01-23

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 and genomic and metagenomic sequences from different soil microbiome environments.

An integrated cryopreserved collection of samples (rhizosphere 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.

Convert Qiime2 objects to Phyloseq objects `Qiime2` is a microbial community analysis tool used for sequencing analysis, while `Phyloseq` is an R package for analyzing high-throughput sequencing data. The `Qiime2R` package allows conversion of `Qiime2` data to `Phyloseq` within R. R enhances features through external packages from sources like CRAN, Bioconductor, and GitHub. After installation, packages must be loaded into the R session using the **library()** function.Once data is imported, Phyloseq enables data manipulation, analysis, and visualization. This conversion leverages R's analysis tools like ggplot2 for visualization, 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")

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

```
# 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)
```

Batch effects correction using Constrained Quantile Normalisation (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
```

```r
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:33
##  Bulksoil   :45   SC: 70   BO:35
##  Oats       :45   SL:101   BU:35
##  OilseedRape:41            HE:35
##  Sugarbeet  :45            SH:68
##  Wheat      :44            YO:67
```

```r
# 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 messages may appear & it can be ignored

# 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)
```

```r
# install.packages("tidyverse")
library("tidyverse")
library("reshape2")
```

```r
## (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)$Soil)[get_variable(physeq.Su
sample_data(physeq.Sugarbeet.group)$Soil.Location <- levels(sample_data(physeq.norm)$Soil.Location)[get_

## (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"))
```

Subgrouping

Bata diversity - before and after the normalisation

Beta diversity quantifies variation in microbial composition among samples, aiding in identifying patterns in microbial distribution. Non-Metric Multidimensional Scaling (NMDS) and Principal Coordinates Analysis (PCoA) are ordination techniques used for beta diversity analysis.

NMDS preserves the rank order of pairwise dissimilarities between samples in a lower-dimensional space, making it suitable for cases where distances between samples are not well-preserved. The distances on the NMDS plot reflect the similarities or dissimilarities between samples but are not directly interpretable.

PCoA, a metric multidimensional scaling technique, attempts to preserve the actual distances between samples in a lower-dimensional space. The distances on the PCoA plot reflect the actual dissimilarities between samples. Unlike NMDS, PCoA may not perform as well with non-linear or rank-based dissimilarity measures.

Here, we employ NMDS to analyze Beta diversity, allowing us to draw comparisons between the states before and after normalisation.

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

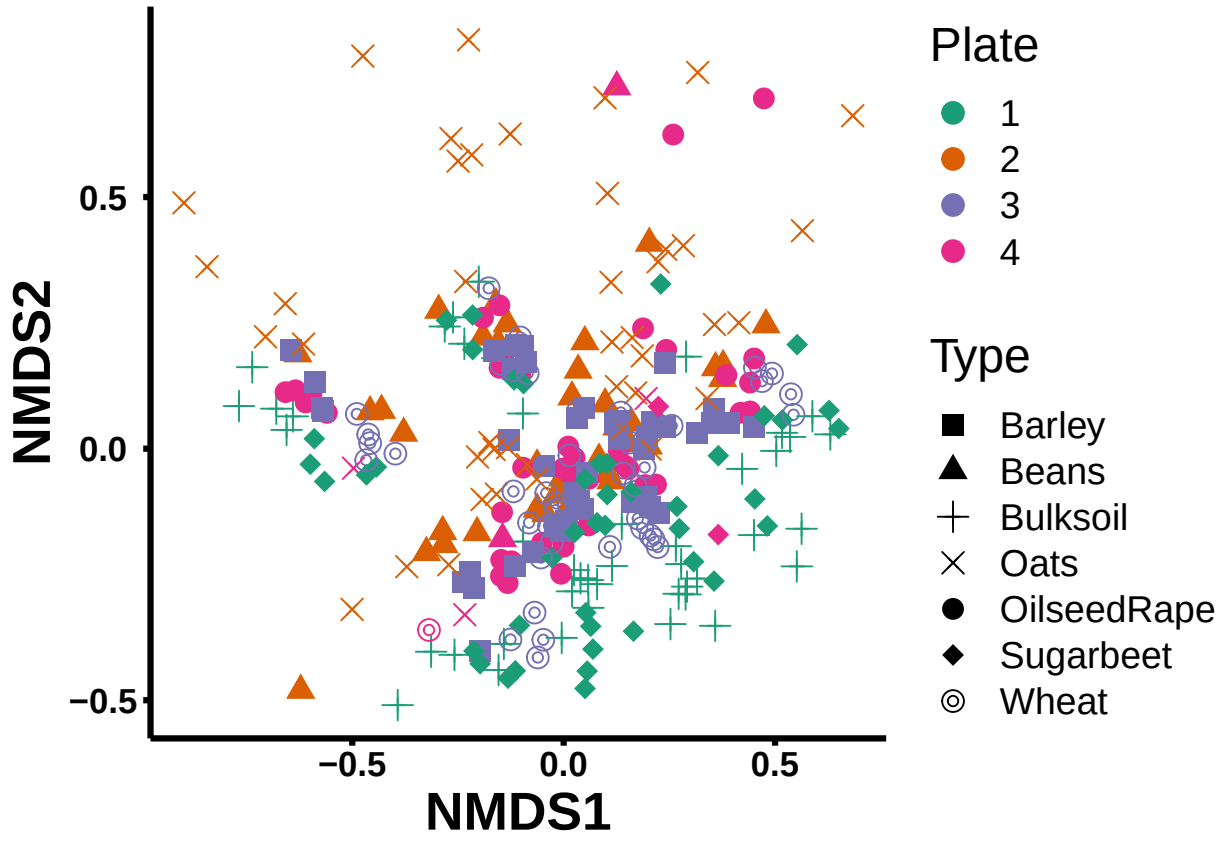
```
##### based on Plate and Type - before the normalisation
# method options: NMDS / PCoA

NMDS1 <- ordinate(physeq = physeq.ori,
                  method = "NMDS",
                  distance = "bray"
                  )
```

colour by different sequence runs

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.1860312
```

```
## Run 1 stress 0.1931827
## Run 2 stress 0.1998788
## Run 3 stress 0.185703
## ... New best solution
## ... Procrustes: rmse 0.01175969  max resid 0.1791125
## Run 4 stress 0.1991221
## Run 5 stress 0.1908918
## Run 6 stress 0.1920496
## Run 7 stress 0.1981843
## Run 8 stress 0.1949982
## Run 9 stress 0.1952726
## Run 10 stress 0.1899819
## Run 11 stress 0.1875538
## Run 12 stress 0.1883131
## Run 13 stress 0.1920649
## Run 14 stress 0.1938736
## Run 15 stress 0.1971078
## Run 16 stress 0.1849942
## ... New best solution
## ... Procrustes: rmse 0.004384641  max resid 0.04927772
## Run 17 stress 0.1964405
## Run 18 stress 0.1871834
## Run 19 stress 0.1905859
## Run 20 stress 0.1907448
## *** Best solution was not repeated -- monoMDS stopping criteria:
##      1: no. of iterations >= maxit
##     10: stress ratio > sratmax
##      9: scale factor of the gradient < sfgrmin
```

```r
# Plot ordination
# pdf(file = "bata_ori_plate.pdf", width = 9,height = 8)
plot_ordination(physeq = physeq.ori,
                      ordination = NMDS1,
                      color = "Plate",
                      shape = "Type"
                      ) +
  theme_classic() +
  geom_point(aes(color = Plate), alpha = 1, size = 3.5) +
  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
```

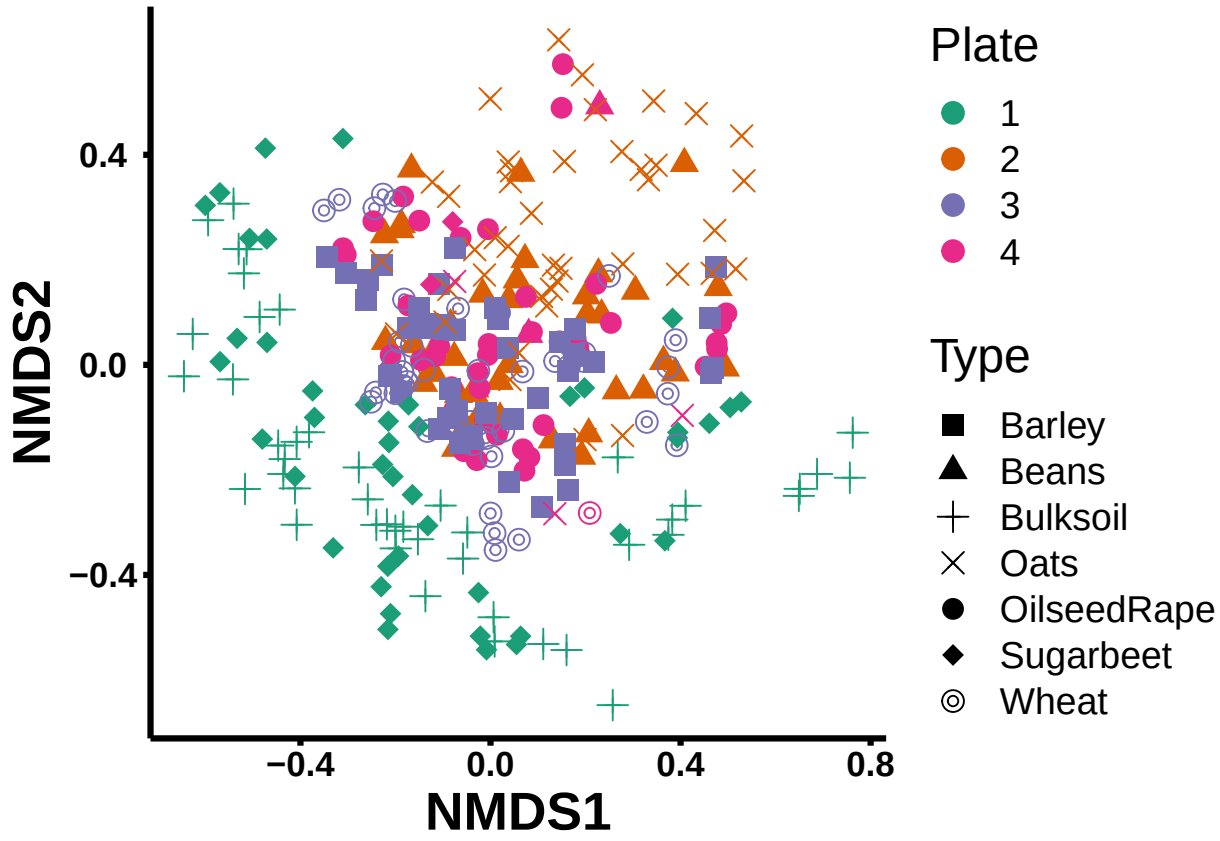



```
# Close the PDF device and save the plot to a file
# dev.off()


##### based on Plate and Type - after the normalisation
# method options: NMDS / PCoA
NMDS2 <- ordinate(physeq = physeq.norm,
                  method = "NMDS",
                  distance = "bray"
                  )
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.2013146
## Run 1 stress 0.2137589
## Run 2 stress 0.2115352
## Run 3 stress 0.2124221
## Run 4 stress 0.1986449
## ... New best solution
## ... Procrustes: rmse 0.01441996  max resid 0.08556403
## Run 5 stress 0.2069555
## Run 6 stress 0.2038399
## Run 7 stress 0.2173298
## Run 8 stress 0.1996355
## Run 9 stress 0.224785
## Run 10 stress 0.1985069
```

```
## ... New best solution
## ... Procrustes: rmse 0.00418695  max resid 0.05805782
## Run 11 stress 0.2039797
## Run 12 stress 0.2237555
## Run 13 stress 0.2039802
## Run 14 stress 0.2177519
## Run 15 stress 0.2053995
## Run 16 stress 0.2482932
## Run 17 stress 0.2328967
## Run 18 stress 0.4182954
## Run 19 stress 0.1989009
## ... Procrustes: rmse 0.005689797  max resid 0.07579884
## Run 20 stress 0.1988948
## ... Procrustes: rmse 0.005740886  max resid 0.07594417
## *** Best solution was not repeated -- monoMDS stopping criteria:
##      1: no. of iterations >= maxit
##     18: stress ratio > sratmax
##      1: scale factor of the gradient < sfgrmin
```

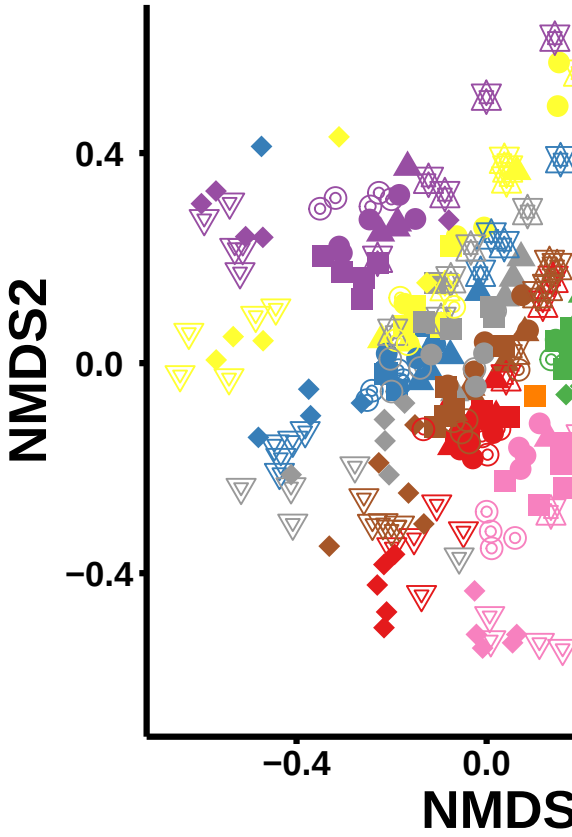
```
# Plot ordination
# pdf(file = "bata_norm_plate.pdf", width = 9,height = 8)
plot_ordination(physeq = physeq.norm,
                ordination = NMDS2,
                color = "Plate",
                shape = "Type"
                ) +
  theme_classic() +
  geom_point(aes(color = Plate), alpha = 1, size = 3.5) +
  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()
```

```
# Plot ordination
# pdf(file = "bata_norm_plate.pdf", width = 9,height = 8)
plot_ordination(physeq = physeq.norm,
                ordination = NMDS2,
                color = "Soil.Location",
                shape = "Type"
) +
  theme_classic() +
  geom_point(aes(color = Soil.Location), alpha = 1, size = 3.5) +
  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 = "Set1") +
  scale_fill_brewer(palette = "Set1") +
  scale_shape_manual(values = c(15, 17, 25, 11, 16, 18, 21, 22, 23) #,
                     # guide = guide_legend(ncol = 2)
                     ) # Set custom shapes
```



colour by different crop types & Soil.Location (normalised data)

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

Than, we examine sugarbeet and bulk soil

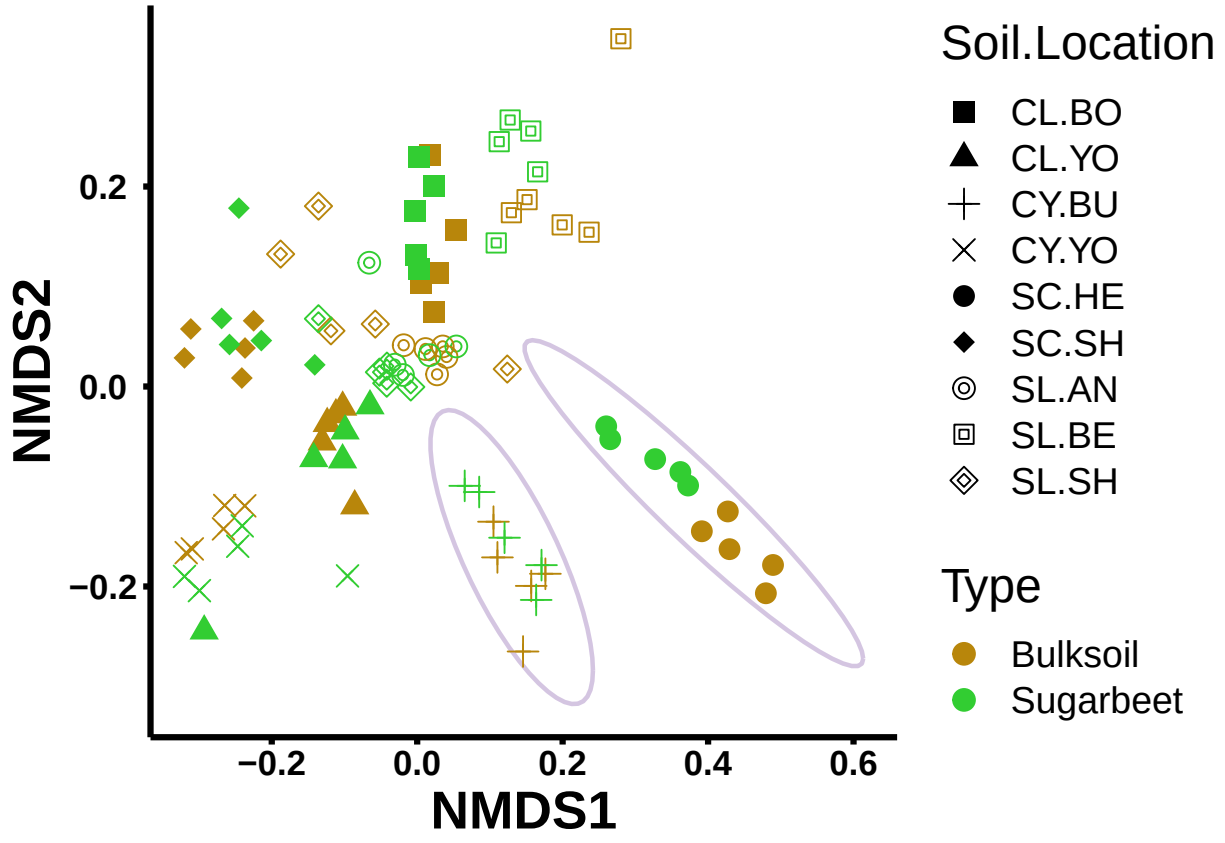
```
# method options: c("DCA", "CCA", "RDA", "CAP", "DPCoA", "NMDS", "MDS", "PCoA")
NMDS <- ordinate(physeq = physeq.SU,
                 method = "NMDS",
                 distance = "bray"
                 )
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.1677248
## Run 1 stress 0.1920378
## Run 2 stress 0.1682628
## Run 3 stress 0.1672579
## ... New best solution
```

```
## ... Procrustes: rmse 0.02584134  max resid 0.1292272
## Run 4 stress 0.1684117
## Run 5 stress 0.1807298
## Run 6 stress 0.1942418
## Run 7 stress 0.1820625
## Run 8 stress 0.1798731
## Run 9 stress 0.2015104
## Run 10 stress 0.1792748
## Run 11 stress 0.1713194
## Run 12 stress 0.179236
## Run 13 stress 0.1828264
## Run 14 stress 0.1687469
## Run 15 stress 0.1652487
## ... New best solution
## ... Procrustes: rmse 0.02013843  max resid 0.1855828
## Run 16 stress 0.1698419
## Run 17 stress 0.1667611
## Run 18 stress 0.168128
## Run 19 stress 0.1679293
## Run 20 stress 0.1652514
## ... Procrustes: rmse 0.000569174  max resid 0.003700737
## ... Similar to previous best
## *** Best solution repeated 1 times
```

```
groups_to_ellipse <- c("SC.HE", "CY.BU") # Replace with your actual group names

physeq.SU.subset <- subset_samples(physeq.SU, Soil.Location %in% groups_to_ellipse)

# Convert physeq.SU.subset to a data frame
df <- sample_data(physeq.SU.subset)

my_palette <- c("darkgoldenrod", "limegreen")

# Create the ordination plot
plot_ordination <- plot_ordination(physeq = physeq.SU,
                                   ordination = NMDS,
                                   color = "Type",
                                   shape = "Soil.Location")

# Extract ordination scores from the plot
df <- plot_ordination$data

# Subset the data for the groups to ellipse
df_subset <- df[df$Soil.Location %in% groups_to_ellipse, ]

# Add the ellipse for the subsetted data

plot_ordination <- plot_ordination +
  stat_ellipse(data = df_subset,
               type="norm",
               alpha=0.25,
               aes(group = Soil.Location),
               linetype = 1,
               size = 0.8,
```

```
                colour = "purple4"
                ) +
  theme_classic() +
  geom_point(aes(color = Type), alpha = 1, size = 3.5) +
  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_manual(values = my_palette) +
  scale_fill_manual(values = my_palette) +
  scale_shape_manual(values = c(15, 17, 3, 4, 16, 18, 21, 22, 23)) # Set custom shapes

# Plot ordination
# pdf(file = "bata_SU.pdf", width = 9,height = 6)
plot_ordination
```




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

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

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.

(https://docs.cosmosid.com/docs/alpha-diversity)

Here, we use Shannon as an example for the work

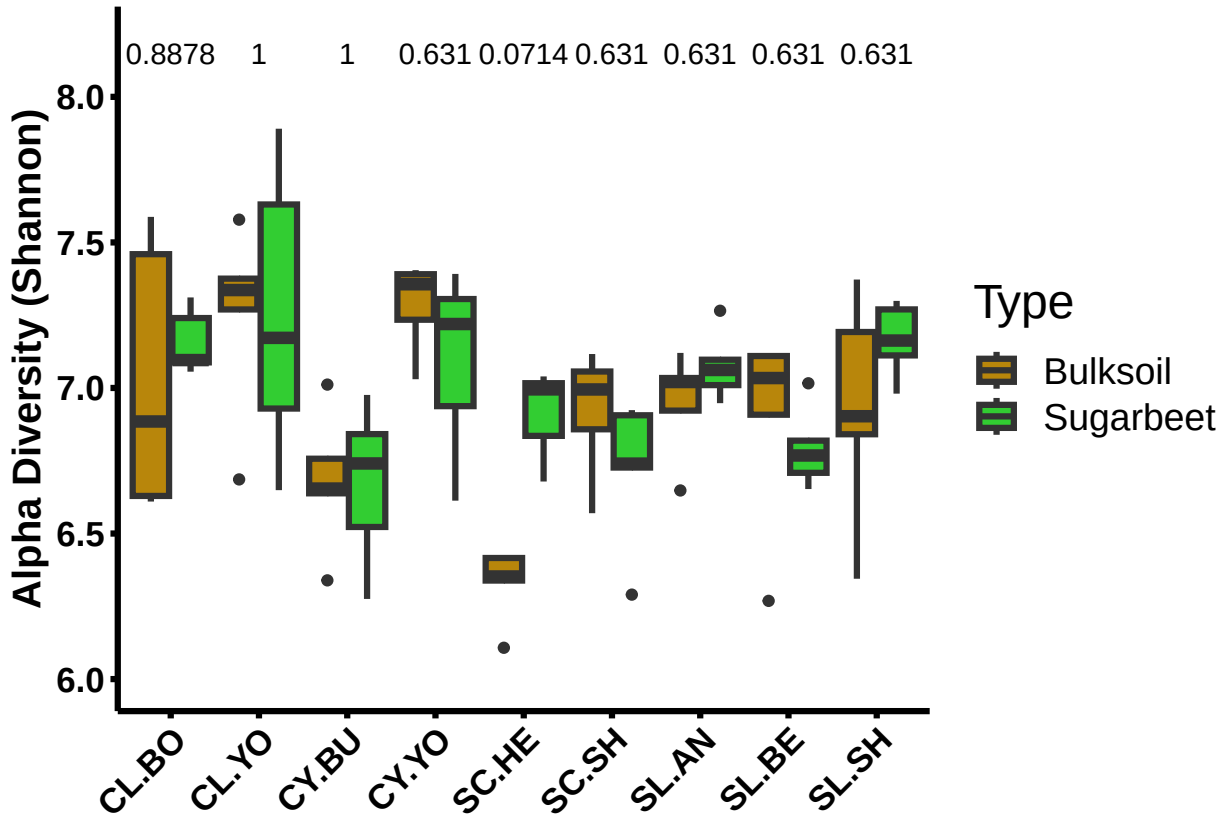
```
# available measurements: "Observed" "Chao1" "ACE" "Shannon" "Simpson" "InvSimpson" "Fisher"
# Calculate alpha diversity (Shannon) and store it in physeq.SU object
alpha.object <- cbind(
  x = sample_data(physeq.SU),
  y = estimate_richness(physeq.SU, measures = 'Shannon')
)

# Data preparation (formatting)
selected_columns <- alpha.object[, c("x.Soil.Location", "x.Group", "x.Type", "Shannon")]
selected_columns2 <- melt(selected_columns)
names(selected_columns2) <- c("Soil_Location","Group" , "Type", "variable", "Shannon")

# Define the comparisons
my_comparisons <- list(
```

```
  c("SU.CL.BO", "CO.CL.BO"),
  c("SU.CL.YO", "CO.CL.YO"),
  c("SU.CY.BU", "CO.CY.BU"),
  c("SU.CY.YO", "CO.CY.YO"),
  c("SU.SC.HE", "CO.SC.HE"),
  c("SU.SC.SH", "CO.SC.SH"),
  c("SU.SL.AN", "CO.SL.AN"),
  c("SU.SL.BE", "CO.SL.BE"),
  c("SU.SL.SH", "CO.SL.SH")
)

# Initialise an empty data frame to store the results
results <- data.frame()

# Perform t-tests for each pair of groups
for (i in seq_along(my_comparisons)) {
  group1_data <- selected_columns2$Shannon[selected_columns2$Group == my_comparisons[[i]][1]]
  group2_data <- selected_columns2$Shannon[selected_columns2$Group == my_comparisons[[i]][2]]

  wilcox_test_result <- wilcox.test(group1_data, group2_data)

  results <- rbind(results, data.frame(
    group1 = my_comparisons[[i]][1],
    group2 = my_comparisons[[i]][2],
    p.value = wilcox_test_result$p.value
  ))
}

# Adjust the p-values for multiple comparisons using the Benjamini-Hochberg procedure
results$p.adjusted <- p.adjust(results$p.value, method = "BH")

# Add significance levels based on the adjusted p-values
results$p.signif <- symnum(results$p.adjusted, corr = FALSE, na = FALSE,
                           cutpoints = c(0, 0.001, 0.01, 0.05, 0.1, 1),
                           symbols = c("***", "**", "*", ".", " "))

# print result
print(results)
```

```
##     group1   group2     p.value p.adjusted p.signif
## 1 SU.CL.BO CO.CL.BO 0.690476190 0.88775510
## 2 SU.CL.YO CO.CL.YO 1.000000000 1.00000000
## 3 SU.CY.BU CO.CY.BU 1.000000000 1.00000000
## 4 SU.CY.YO CO.CY.YO 0.309523810 0.63095238
## 5 SU.SC.HE CO.SC.HE 0.007936508 0.07142857        .
## 6 SU.SC.SH CO.SC.SH 0.222222222 0.63095238
## 7 SU.SL.AN CO.SL.AN 0.420634921 0.63095238
## 8 SU.SL.BE CO.SL.BE 0.222222222 0.63095238
## 9 SU.SL.SH CO.SL.SH 0.420634921 0.63095238
```

```
# Defind colour
my_palette <- c("darkgoldenrod", "limegreen")
```

```r
# Create the boxplot
p <- ggplot(data=selected_columns2, aes(x=Soil_Location, y= Shannon, fill = Type)) +
  geom_boxplot(size = 1.1,
               width = 0.825,
               color = "grey20",
               position = position_dodge(0.9)
  ) +
  scale_fill_manual(values = my_palette) +
  labs(x = element_blank(),
       y = "Alpha Diversity (Shannon)"
  ) +
  theme_classic() +
  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 = "right") +
  scale_y_continuous(breaks = seq(6, 8, by = 0.5), limits = c(6, 8.2))

# Add the results of the comparisons to the plot
for (i in seq_len(nrow(results))) {

  # Set the y position for the label
  y_position <- 8.15

  # Add the label to the plot
p <- p + annotate("text", x = i,
                  y=y_position ,
                  label=round(results$p.adjusted[i], 4),
                  size= 3.75, face = "bold")
}

# Create a plot for alpha diversity
# pdf(file = "Fig7B_alpha.pdf", width = 8,height = 5)

# Print the plot
print(p)
```




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

# Clean up by removing the alpha.object
# rm(alpha.object, selected_columns, selected_columns2, my_comparisons, results, my_palette, group1_dat
```

Determine the count of taxa within each level and 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)
```

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")
```

```r
metdat = as.data.frame(as.matrix(physeq.SU@sam_data))
dat = as.data.frame(t(as.data.frame(physeq.SU@otu_table)))

# Define the pairs for comparison
pairs <- list(
  c("SU.CL.BO", "CO.CL.BO"),
  c("SU.CL.YO", "CO.CL.YO"),
  c("SU.CY.BU", "CO.CY.BU"),
  c("SU.CY.YO", "CO.CY.YO"),
  c("SU.SC.HE", "CO.SC.HE"),
  c("SU.SC.SH", "CO.SC.SH"),
  c("SU.SL.AN", "CO.SL.AN"),
  c("SU.SL.BE", "CO.SL.BE"),
  c("SU.SL.SH", "CO.SL.SH")
)

# Initialize an empty list to store the results
results <- list()

# Loop over each pair
for(i in seq_along(pairs)) {
  pair <- pairs[[i]]

  dat$Group = metdat$Group
  dat_subset <- dat[dat$Group %in% pair, ]
  dat_subset$Group <- NULL

  metdat_subset <- metdat[metdat$Group %in% pair, ]

  # Perform the pairwise comparison
  results[[i]] <- pairwise.adonis(dat_subset,
                                  metdat_subset$Group,
                                  sim.function = "vegdist",
                                  sim.method = "bray",
                                  reduce = NULL, perm = 100000)
}

# Convert the list of results to a data frame
results_df <- do.call(rbind, lapply(results, function(x) data.frame(t(unlist(x)))))

# Add the adjusted p-value
# "bonferroni", "holm", "hochberg", "hommel", "BH" or "BY"
results_df$p.adjusted <- p.adjust(results_df$p.value, method = "BH")

# Print the dataframe to check the new column
print(results_df)
```

```
##                  pairs Df         SumsOfSqs           F.Model                R2
## 1 CO.CL.BO vs SU.CL.BO  1  0.19234498806295 0.982889882772245 0.109418004183406
## 2 CO.CL.YO vs SU.CL.YO  1 0.213461169617495 0.922512086881703 0.103391520000071
## 3 CO.CY.BU vs SU.CY.BU  1 0.179825627125348  1.15125484611348 0.125802948936824
## 4 CO.CY.YO vs SU.CY.YO  1  0.21153126052888   1.1109025342993 0.121931118252791
## 5 CO.SC.HE vs SU.SC.HE  1 0.907826586326208  10.9195252783035 0.577156409459479
```

```
## 6 CO.SC.SH vs SU.SC.SH  1 0.270828638514029  1.61501960608062 0.167968415275957
## 7 CO.SL.AN vs SU.SL.AN  1 0.222511174445787  1.51768215168303 0.159459217853231
## 8 CO.SL.BE vs SU.SL.BE  1 0.289600010447189  1.29444922906478 0.139271214158327
## 9 CO.SL.SH vs SU.SL.SH  1 0.292493472968061  1.64486102559631 0.170542739934878
##               p.value p.adjusted sig
## 1   0.396846031539685 0.44645179
## 2   0.523494765052349 0.52349477
## 3   0.228677713222868 0.29401420
## 4   0.174108258917411 0.26116239
## 5 0.00748992510074899 0.05024950   *
## 6   0.015829841701583 0.05024950   .
## 7   0.016749832501675 0.05024950   .
## 8   0.047719522804772 0.08684913   .
## 9   0.048249517504825 0.08684913   .
```

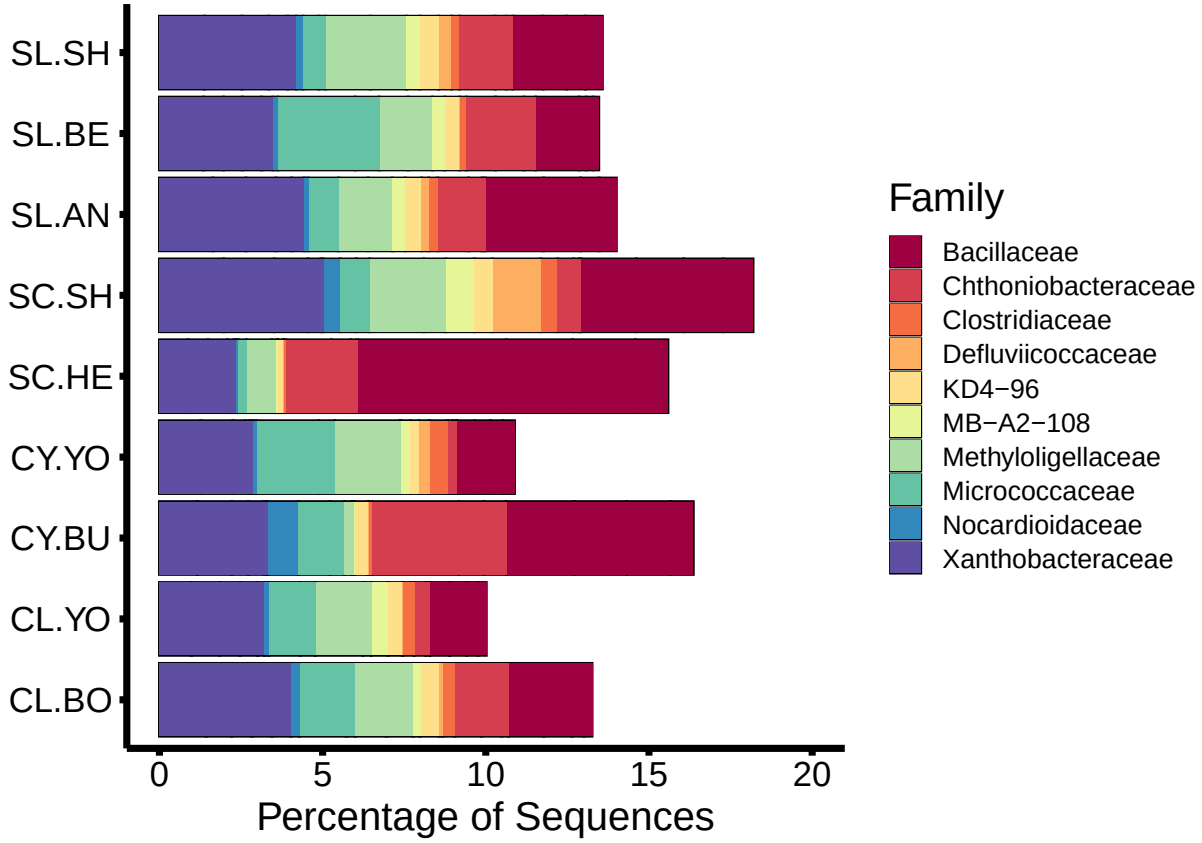
```
# Clean up by removing unnecessary objects
# rm(metdat, dat, pairs, i, dat_subset, metdat_subset, results, results_df)
```

Plotting the top 10 taxa at family level

Plotting the top 10 taxa at the family level offers a clear and concise overview of the microbial composition. This method not only highlights the most prevalent families in the sample but also simplifies complex microbiome data. It facilitates comparative analysis across different sample groups and allows for easy interpretation of trends and patterns. This approach enables quick identification of the most prevalent families within the corresponding sample groups.

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

##
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)

my_palette <- brewer.pal(n = 10, name = "Spectral")

# plotting

p <- plot_bar(Rgsm10, "Soil.Location", fill = "Family") + coord_flip() +
  ylab("Percentage of Sequences") + ylim(0, 20) +
  geom_col() +  coord_flip() +
  scale_fill_manual(values = my_palette) +
  labs(x = element_blank()) +
  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')
        )


# Create a plot for alpha diversity
# pdf(file = "Fig08A_TOP10.pdf", width = 8,height = 5)
print(p)
```




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

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

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.

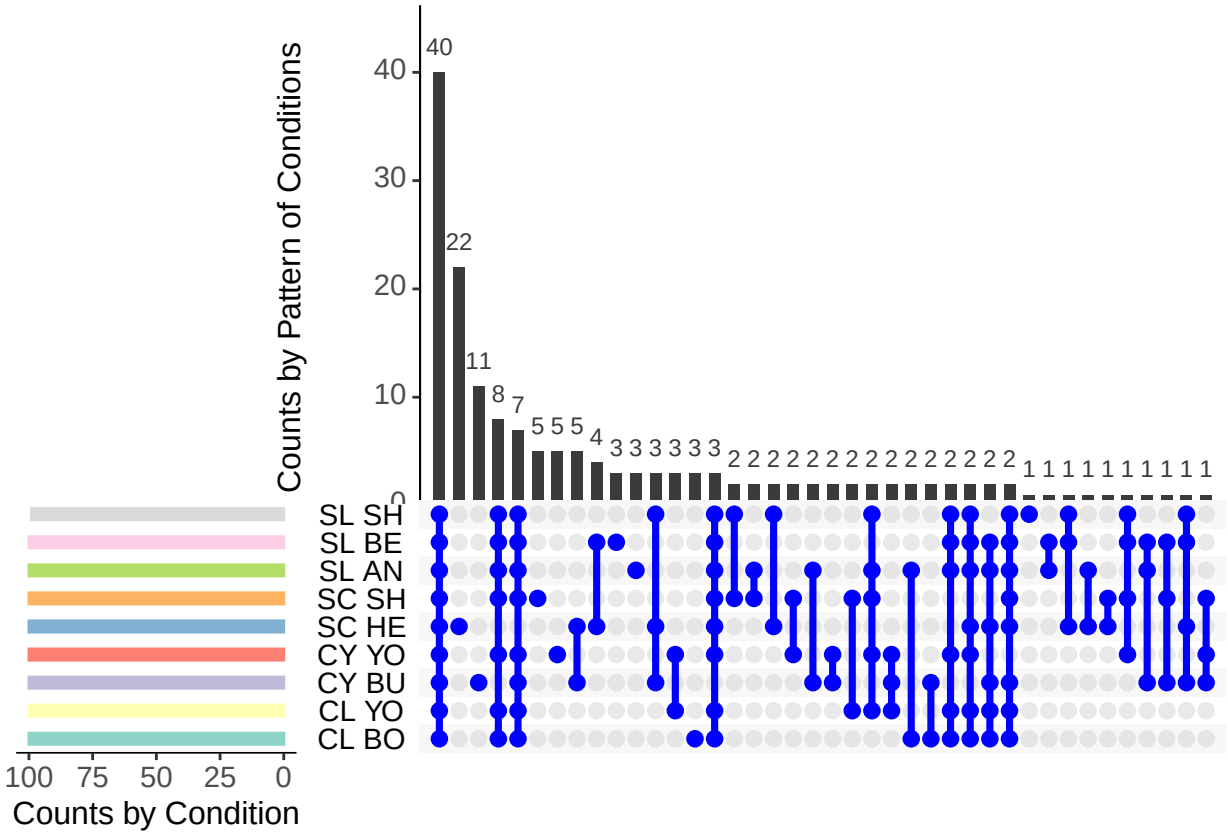
```r
# BiocManager::install("microbiome")
library("microbiome")

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

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

```r
# 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"
```

```r
# 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 = Soil.Location, ASV = .otu, Abundance = .abundance) %>%
  select(Sample, Abundance, Genus) %>%
  group_by(Sample) %>%
  filter(rank(desc(Abundance)) <= 100) %>% # Filter <= 100
  ungroup()

# Remove the Abundance column
D$Abundance <- NULL

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

```r
names(D)[1] <- "Soil.Location"

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

# 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",
                    mb.ratio = c(0.6, 0.4),
                    point.size= 2.75,
                    line.size = 1.25,
                    text.scale = 1.5
)

# Create a plot for alpha diversity
# Open a new PDF graphics device
# pdf(file = "Fig08B_UpSet.pdf", width=8,height=5)
# Print the ggtree plot
print(upset_plot)
```

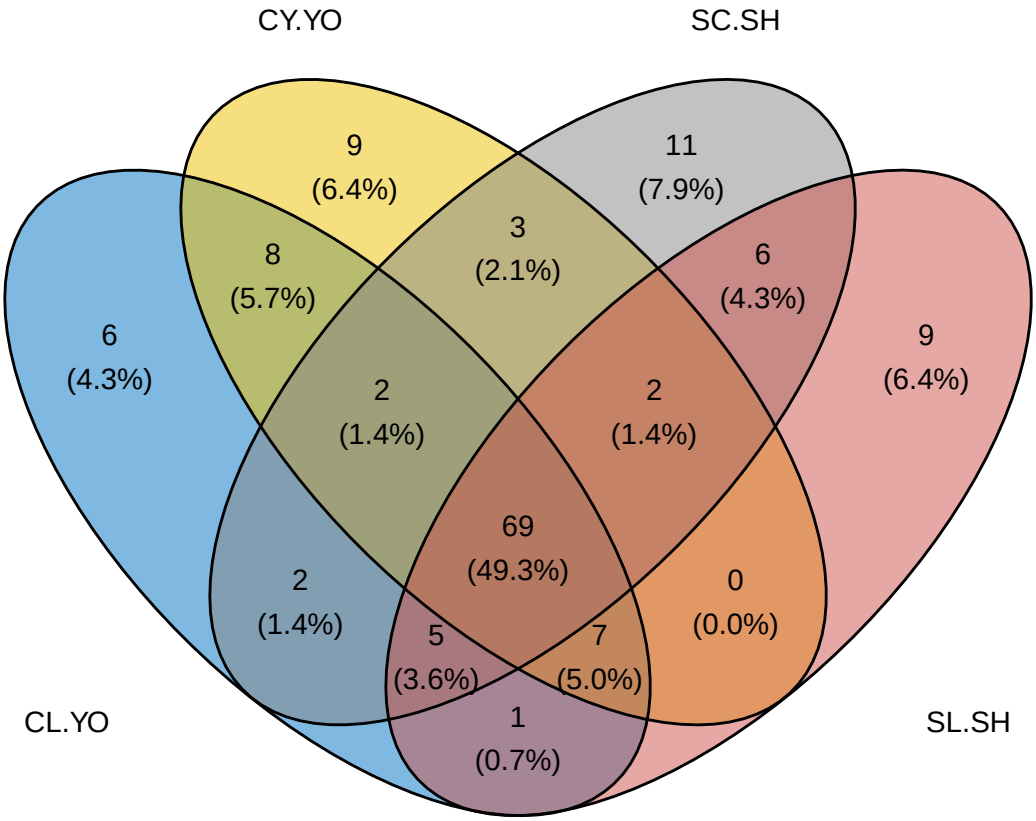



```
# Close the PDF device and save the plot to a file
# dev.off()

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

```
# if (!require(devtools)) install.packages("devtools")
# devtools::install_github("yanlinlin82/ggvenn")

library(ggvenn)
```

```
# Extract the rows where the value is 1 for each column
CL.YO <- rownames(temp_df)[temp_df$CL.YO == 1]
CY.YO <- rownames(temp_df)[temp_df$CY.YO == 1]
SC.SH <- rownames(temp_df)[temp_df$SC.SH == 1]
SL.SH <- rownames(temp_df)[temp_df$SL.SH == 1]

# Create a list with the extracted data
list_data <- list("CL.YO" = CL.YO, "CY.YO" = CY.YO, "SC.SH" = SC.SH, "SL.SH" = SL.SH)

# Use ggvenn to create the Venn diagram
Venn <- ggvenn(
  list_data,
  fill_color = c("#0073C2FF", "#EFC000FF", "#868686FF", "#CD534CFF"),
  stroke_size = 0.5, set_name_size = 4
```

```
  )

# Open a new PDF graphics device
# pdf(file = "Fig08C_Venn.pdf", width=5,height=5)

# Print the Venn plot
print(Venn)
```




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

```
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 11 x64 (build 22631)
##
## 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] grid      parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] ggvenn_0.1.10        plyr_1.8.9           UpSetR_1.4.0
##  [4] speedyseq_0.5.3.9018 microbiome_1.22.0    GGally_2.1.2
##  [7] pairwiseAdonis_0.4.1 cluster_2.1.4        vegan_2.6-4
## [10] lattice_0.21-9       permute_0.9-7        ggpubr_0.6.0
## [13] reshape2_1.4.4       lubridate_1.9.3      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] ggplot2_3.4.4        tidyverse_2.0.0      doParallel_1.0.17
## [25] iterators_1.0.14     foreach_1.5.2        ConQuR_2.0
## [28] RColorBrewer_1.1-3   phyloseq_1.44.0      qiime2R_0.99.6
##
## loaded via a namespace (and not attached):
##   [1] splines_4.3.2           bitops_1.0-7            rpart_4.1.21
##   [4] fastDummies_1.7.3       lifecycle_1.0.4         rstatix_0.7.2
##   [7] MASS_7.3-60             backports_1.4.1         magrittr_2.0.3
##  [10] Hmisc_5.1-1             rmarkdown_2.25          yaml_2.3.7
##  [13] bayesm_3.1-6            ade4_1.7-22             abind_1.4-5
##  [16] zlibbioc_1.46.0         Rtsne_0.16              BiocGenerics_0.46.0
##  [19] RCurl_1.98-1.12         nnet_7.3-19             tensorA_0.36.2
##  [22] GenomeInfoDbData_1.2.10 cqrReg_1.2.1            IRanges_2.34.1
##  [25] S4Vectors_0.38.2        ggrepel_0.9.4           rmutil_1.1.10
##  [28] inline_0.3.19           MatrixModels_0.5-2      spatial_7.3-17
##  [31] codetools_0.2-19        DT_0.30                 tidyselect_1.2.0
##  [34] shape_1.4.6             farver_2.1.1            stable_1.1.6
##  [37] matrixStats_1.0.0       stats4_4.3.2            base64enc_0.1-3
##  [40] jsonlite_1.8.7          multtest_2.56.0         Formula_1.2-5
##  [43] survival_3.5-7          tools_4.3.2             Rcpp_1.0.11
##  [46] glue_1.6.2              gridExtra_2.3           xfun_0.40
##  [49] mgcv_1.9-0              GenomeInfoDb_1.36.4     withr_2.5.2
##  [52] timeSeries_4031.107     fastmap_1.1.1           rhdf5filters_1.12.1
##  [55] fansi_1.0.4             SparseM_1.81            caTools_1.18.2
##  [58] digest_0.6.33           truncnorm_1.0-9         timechange_0.2.0
##  [61] R6_2.5.1                colorspace_2.1-0        gtools_3.9.4
##  [64] modeest_2.4.0           utf8_1.2.3              generics_0.1.3
##  [67] data.table_1.14.8       robustbase_0.99-0       htmlwidgets_1.6.2
##  [70] pkgconfig_2.0.3         gtable_0.3.4            timeDate_4022.108
##  [73] zCompositions_1.4.1     XVector_0.40.0          htmltools_0.5.6
##  [76] carData_3.0-5           biomformat_1.28.0       clue_0.3-65
##  [79] scales_1.3.0            Biobase_2.60.0          knitr_1.44
##  [82] rstudioapi_0.15.0       tzdb_0.4.0              statip_0.2.3
##  [85] checkmate_2.2.0         nlme_3.1-163            rhdf5_2.44.0
##  [88] KernSmooth_2.23-22      foreign_0.8-85          fBasics_4031.95
##  [91] pillar_1.9.0            reshape_0.8.9           vctrs_0.6.3
```

```
##  [94] gplots_3.1.3           randomForest_4.7-1.1   car_3.1-2
##  [97] htmlTable_2.4.1        evaluate_0.22          cli_3.6.1
## [100] compiler_4.3.2         rlang_1.1.1            crayon_1.5.2
## [103] ggsignif_0.6.4         labeling_0.4.3         stringi_1.7.12
## [106] munsell_0.5.0          Biostrings_2.68.1      glmnet_4.1-8
## [109] compositions_2.0-6     quantreg_5.97          Matrix_1.6-1.1
## [112] hms_1.1.3              stabledist_0.7-1       NADA_1.6-1.1
## [115] Rhdf5lib_1.22.1        statmod_1.5.0          ROCR_1.0-11
## [118] GUniFrac_1.8           igraph_1.5.1           broom_1.0.5
## [121] DEoptimR_1.1-3         ape_5.7-1
```