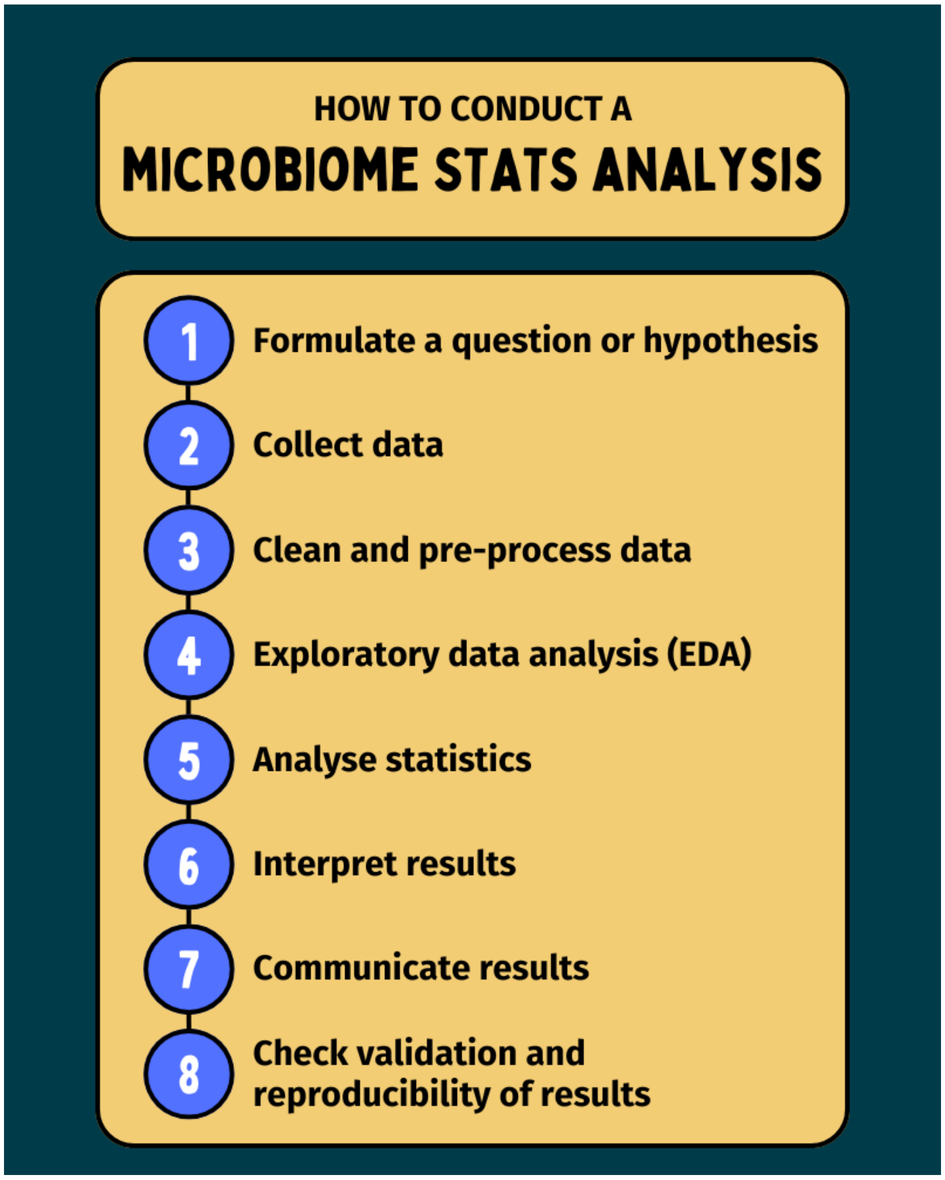
Microbiome Machine Learning


HOW TO CONDUCT A
MICROBIOME STATS ANALYSIS
1 Formulate a question or hypothesis
2 Collect data
3 Clean and pre-process data
4 Exploratory data analysis (EDA)
5 Analyse statistics
6 Interpret results
7 Communicate results
8 Check validation and reproducibility of results

Last updated: May 26, 2025

Contents

Part I

OVERVIEW

About This Guide

Part II

MICROBIOME DATA EDA

Q&A: 1

What are the essential tools for microbiome read quality control?

1.1 Explanation

Every microbiome analysis begins with raw sequencing data, often in the form of FASTQ files. These files contain both the nucleotide reads and quality scores. However, raw reads are rarely perfect — they may contain adapter sequences, low-quality regions, or even contaminant DNA.

Before proceeding to any taxonomic or functional profiling, it's essential to clean and assess these reads. This is the foundation of your analysis pipeline — ensuring that only high-quality data moves forward.

Several tools have been developed for this exact purpose. Most are installable via **Bioconda**, and they can be used independently or as part of an automated pipeline.

Here's a breakdown of what each tool does:

- **Seqkit**: Provides basic statistics about your FASTQ files (e.g., length distribution, GC content).
- **FastQC**: Generates per-base quality score plots to detect poor-quality cycles.
- **MultiQC**: Aggregates FastQC outputs across samples into a single report.
- **BBMap / Trimmomatic**: Trim adapters, remove artifacts, and perform quality filtering.
- **Kneaddata**: Specialized for metagenomics, it removes contaminant reads (e.g., host DNA) using alignment-based filtering.

1.2 Shell Code

```
# Install individual tools using mamba and bioconda
mamba install -c bioconda seqkit fastqc multiqc bbmap trimmomatic
# Install kneaddata from Biobakery channel (for metagenomics)
mamba install -c biobakery kneaddata
```

1.3 R Note

```
# These tools are primarily used from the command line, but their output files
# (e.g., FastQC or MultiQC reports) can be imported into R for downstream summarizatio
```

Q&A: 2

How do I obtain example microbiome sequencing data for analysis?

2.1 Explanation

Before performing any analysis, you need access to microbiome sequencing data. This data typically comes in the form of **FASTQ** files (either single-end or paired-end), which contain raw reads from amplicon sequencing.

There are several sources for publicly available datasets: - **QIIME2 Tutorials**: Include curated sample data for testing pipelines - **NCBI SRA / EBI ENA**: Provide raw sequencing data from published studies - **Qiita**: A microbiome database for submitting and reusing 16S/18S/ITS data - **Mock communities**: Simulated or synthetic datasets used to benchmark tools

This example uses the classic *Moving Pictures* tutorial dataset from QIIME2.

2.2 Shell Code

```
# Download paired-end FASTQ data from QIIME2 tutorial
wget https://data.qiime2.org/2024.2/tutorials/moving-pictures/emp-paired-end-sequences/b
wget https://data.qiime2.org/2024.2/tutorials/moving-pictures/emp-paired-end-sequences/f
wget https://data.qiime2.org/2024.2/tutorials/moving-pictures/emp-paired-end-sequences/r
```

2.3 Python Note

```
# Although QIIME2 is Python-based, raw sequencing data is usually downloaded externall
# Python/QIIME2 will be used later to import and process these FASTQ files.
```

2.4 R Note

```
# Most raw sequencing workflows do not begin in R. However, after generating feature t
# from tools like QIIME2 or mothur, R will be used for downstream analysis and visuali
```

Q&A: 3

How do I process raw sequencing data into a feature table using QIIME2?

3.1 Explanation

After obtaining your raw FASTQ data, the first analytical step is transforming it into a structured format for analysis — the **feature table**. This is a matrix of counts (samples × ASVs or OTUs).

QIIME2 is a powerful, Python-based platform for this entire workflow. It uses `.qza` files (QIIME artifacts) to structure data at each step and generates a `.qzv` summary for inspection.

This pipeline includes: - Importing FASTQ data - Demultiplexing reads - Denoising to generate ASVs using `DADA2` or `Deblur` - Creating a feature table

3.2 Shell Code (QIIME2 CLI)

```
# 1. Import paired-end reads
qiime tools import \
  --type EMPPairedEndSequences \
  --input-path emp-paired-end-sequences \
  --output-path emp-paired-end-sequences.qza

# 2. Demultiplex (barcode + sample mapping required)
```

```
qiime demux emp-paired \
  --i-seqs emp-paired-end-sequences.qza \
  --m-barcodes-file sample-metadata.tsv \
  --m-barcodes-column BarcodeSequence \
  --o-per-sample-sequences demux.qza \
  --o-error-correction-details demux-details.qza

# 3. Visualize quality
qiime demux summarize \
  --i-data demux.qza \
  --o-visualization demux.qzv

# 4. Denoise with DADA2
qiime dada2 denoise-paired \
  --i-demultiplexed-seqs demux.qza \
  --p-trim-left-f 0 \
  --p-trim-left-r 0 \
  --p-trunc-len-f 240 \
  --p-trunc-len-r 200 \
  --o-table table.qza \
  --o-representative-sequences rep-seqs.qza \
  --o-denoising-stats denoising-stats.qza
```

3.3 Python Note

```
# Although QIIME2 is Python-based, its workflow is run via CLI.
# Feature table (table.qza) can be exported and summarized later using Python or R.
```

Q&A: 4

How do I process raw sequencing data into a feature table using Mothur?

4.1 Explanation

Mothur is an open-source software package for analyzing 16S rRNA gene sequences. It supports raw data preprocessing, OTU clustering, and taxonomic assignment. Its workflow is particularly popular for microbial community studies and works well with single- or paired-end FASTQ data.

Mothur pipelines are usually run using command files or interactively within the Mothur console. Key steps include: - Merging paired-end reads - Quality filtering - Aligning and clustering reads into OTUs - Generating `.shared` (feature table) and `.taxonomy` files

4.2 Shell Code

```
# Launch Mothur
mothur

# Inside Mothur console (example workflow)
mothur > make.file(inputdir=./raw_data, type=fastq, prefix=stability)
mothur > make.contigs(file=stability.files, processors=8)
mothur > screen.seqs(fasta=stability.trim.contigs.fasta, group=stability.contigs.groups,
mothur > unique.seqs(fasta=stability.trim.contigs.good.fasta)
mothur > count.seqs(name=stability.trim.contigs.good.names, group=stability.contigs.good
mothur > align.seqs(fasta=stability.trim.contigs.good.unique.fasta, reference=silva.seed
mothur > screen.seqs(...)
mothur > pre.cluster(...)
mothur > chimera.vsearch(...)
mothur > remove.seqs(...)
mothur > cluster.split(...)
mothur > make.shared(...)
mothur > classify.otu(...)
```

4.3 R Note

```
# Mothur outputs a `.shared` file (feature/OTU table) and `.taxonomy` file.
# These can be imported into R using packages like phyloseq or tidyverse for analysis.
```

4.4 Python Note

```
# Mothur is not Python-based, but output files can be parsed with pandas or biom-forma
```

Q&A: 5

How do I explore and summarize a microbiome OTU table?

5.1 Explanation

After generating an OTU (or feature) table from raw sequencing data, it's essential to inspect and summarize it before moving into alpha or beta diversity analysis.

The OTU table is typically a matrix of **samples** × **features** (ASVs/OTUs), where each cell contains the abundance count of a feature in a sample.

Key summary steps include: - Calculating **sample richness** (how many OTUs each sample contains) - Measuring **OTU prevalence** (in how many samples each OTU occurs) - Assessing **abundance distribution** (e.g., sparse vs dominant OTUs) - Identifying **sparse or noisy features** that may need filtering

5.2 Python Code

```
import pandas as pd

# Load OTU table (OTUs as rows, samples as columns)
otu_df = pd.read_csv("data/otu_table.tsv", sep="\t", index_col=0)
```

```
# Number of OTUs per sample (richness)
sample_richness = (otu_df > 0).sum(axis=0)

# Number of samples per OTU (prevalence)
otu_prevalence = (otu_df > 0).sum(axis=1)

# Distribution of total counts per OTU
otu_abundance_summary = otu_df.sum(axis=1).describe()

print("Sample Richness:", sample_richness.head())
print("OTU Prevalence:", otu_prevalence.head())
print("Abundance Summary:", otu_abundance_summary)
```

5.3 R Code

```
otu_df <- read.delim("data/otu_table.tsv", row.names = 1)

# Sample richness: number of OTUs per sample
colSums(otu_df > 0)
```

```
 Sample_1  Sample_2  Sample_3  Sample_4  Sample_5  Sample_6  Sample_7  Sample_8
       44        44        43        40        40        42        45        43
 Sample_9 Sample_10
       45        41
```

```
# OTU prevalence: number of samples each OTU appears in
rowSums(otu_df > 0)
```

```
 OTU_1  OTU_2  OTU_3  OTU_4  OTU_5  OTU_6  OTU_7  OTU_8  OTU_9 OTU_10 OTU_11
    10      7      9      9      8      8      7      8      9      8     10
OTU_12 OTU_13 OTU_14 OTU_15 OTU_16 OTU_17 OTU_18 OTU_19 OTU_20 OTU_21 OTU_22
     9     10      9      9      7      9     10      9     10      7      8
```

```
OTU_23 OTU_24 OTU_25 OTU_26 OTU_27 OTU_28 OTU_29 OTU_30 OTU_31 OTU_32 OTU_33
     8      9      8      9      9     10      8      8      7      9      9
OTU_34 OTU_35 OTU_36 OTU_37 OTU_38 OTU_39 OTU_40 OTU_41 OTU_42 OTU_43 OTU_44
     8     10     10      9     10      7      8      8      9      9      8
OTU_45 OTU_46 OTU_47 OTU_48 OTU_49 OTU_50
     8      8      8      7      7      9
```

```
# Distribution of total counts per OTU
summary(rowSums(otu_df))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   36.0    41.0    48.0    47.4    52.0    64.0 
```

Q&A: 6

How do I filter out low-abundance or low-prevalence OTUs?

6.1 Explanation

OTU tables are often sparse, with many OTUs occurring in only a few samples or at very low abundances. These low-abundance and low-prevalence OTUs can introduce noise, inflate diversity metrics, and complicate downstream analysis.

Filtering such OTUs is a critical EDA step before diversity analysis or visualization. Common criteria include: - **Prevalence**: Removing OTUs that appear in fewer than X samples - **Abundance**: Removing OTUs with total counts below a threshold

This step helps reduce dimensionality and improves interpretability.

6.2 Python Code

```
import pandas as pd

# Load OTU table
otu_df = pd.read_csv("data/otu_table.tsv", sep="\t", index_col=0)

# Filter: keep OTUs present in at least 3 samples
```

```
otu_filtered = otu_df[(otu_df > 0).sum(axis=1) >= 3]

# Further filter: keep OTUs with total count   10
otu_filtered = otu_filtered[otu_filtered.sum(axis=1) >= 10]

# Save filtered table
otu_filtered.to_csv("data/otu_table_filtered.tsv", sep="\t")
```

6.3 R Code

```r
otu_df <- read.delim("data/otu_table.tsv", row.names = 1)

# Filter OTUs with prevalence   3 samples
keep_rows <- rowSums(otu_df > 0) >= 3
otu_df <- otu_df[keep_rows, ]

# Further filter by total abundance   10
keep_abundant <- rowSums(otu_df) >= 10
otu_df_filtered <- otu_df[keep_abundant, ]

# Write filtered table
write.table(otu_df_filtered, file = "data/otu_table_filtered.tsv", sep = "\t")
```

Part III

MICROBIOME DATA VISUALIZATION

Q&A: 7

How do I visualize total OTU abundance per sample?

7.1 Explanation

Before diving into deeper microbiome comparisons, it's helpful to visualize the **sequencing depth** — the total number of OTU counts per sample. This allows you to check: - Sample variability - Potential outliers - Overall library size distribution

Using modern tools like **ggplot2** in R or **seaborn** in Python helps create clearer, more elegant plots.

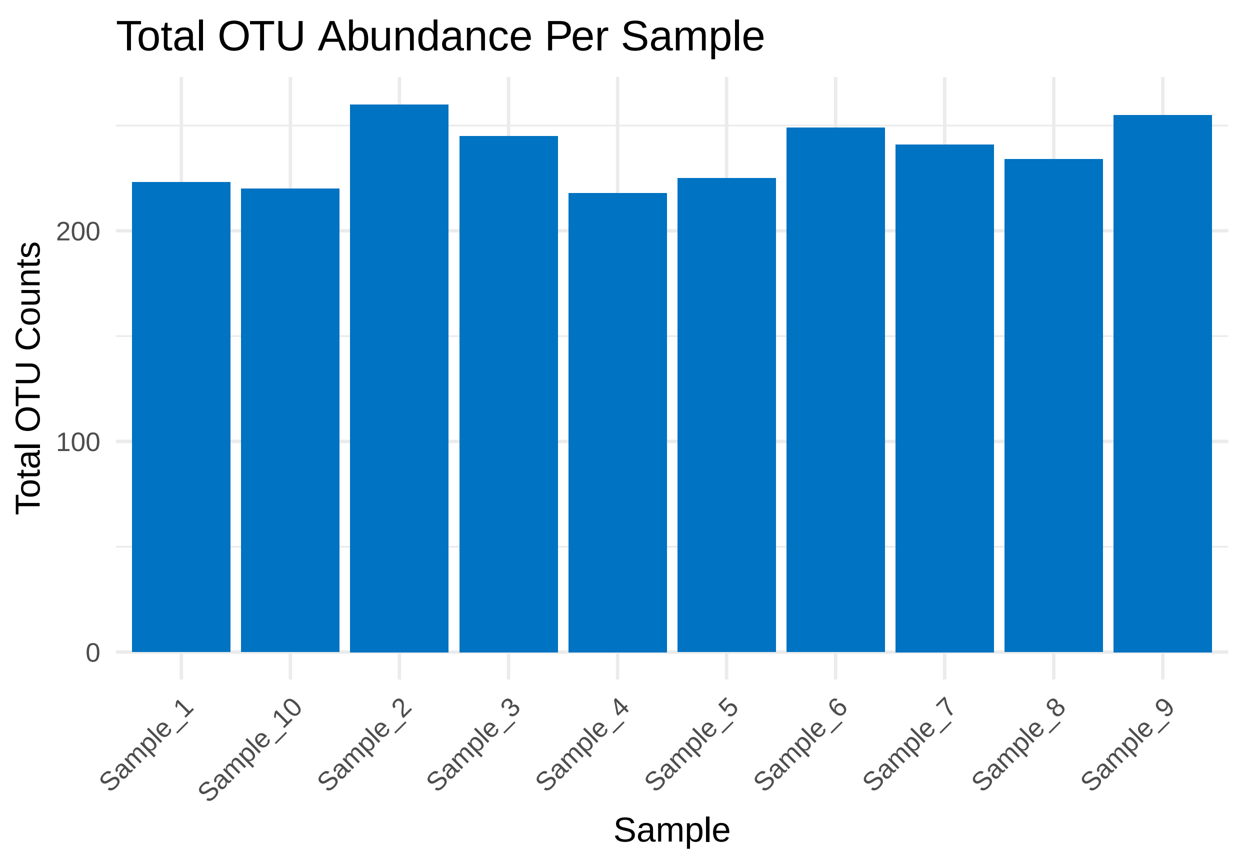
7.2 Python Code

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load OTU table
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0)

# Prepare data
total_counts = otu_df.sum(axis=0).reset_index()
```

```
total_counts.columns = ["Sample", "Total_OTUs"]

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(data=total_counts, x="Sample", y="Total_OTUs", palette="viridis")
plt.title("Total OTU Abundance Per Sample")
plt.xticks(rotation=45)
plt.ylabel("Total OTU Counts")
plt.tight_layout()
plt.show()
```

7.3 R Code

```r
library(tidyverse)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
otu_long <- colSums(otu_df) %>%
  enframe(name = "Sample", value = "Total_OTUs")

ggplot(otu_long, aes(x = Sample, y = Total_OTUs)) +
  geom_col(fill = "#0073C2FF") +
  labs(title = "Total OTU Abundance Per Sample", y = "Total OTU Counts") +
  theme_minimal(base_size = 13) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```


Total OTU Abundance Per Sample
Total OTU Counts
0
100
200
Sample_1
Sample_10
Sample_2
Sample_3
Sample_4
Sample_5
Sample_6
Sample_7
Sample_8
Sample_9
Sample

Q&A: 8

How do I create a stacked bar plot of top genera across samples?

8.1 Explanation

Stacked bar plots are widely used in microbiome studies to show **relative abundance** of microbial taxa across samples. This visual helps assess: - Community composition - Dominant vs rare genera - Variability between sample groups

Here we simulate a relative abundance plot using the **Genus** column from the taxonomy file merged with the OTU table.

8.2 Python Code

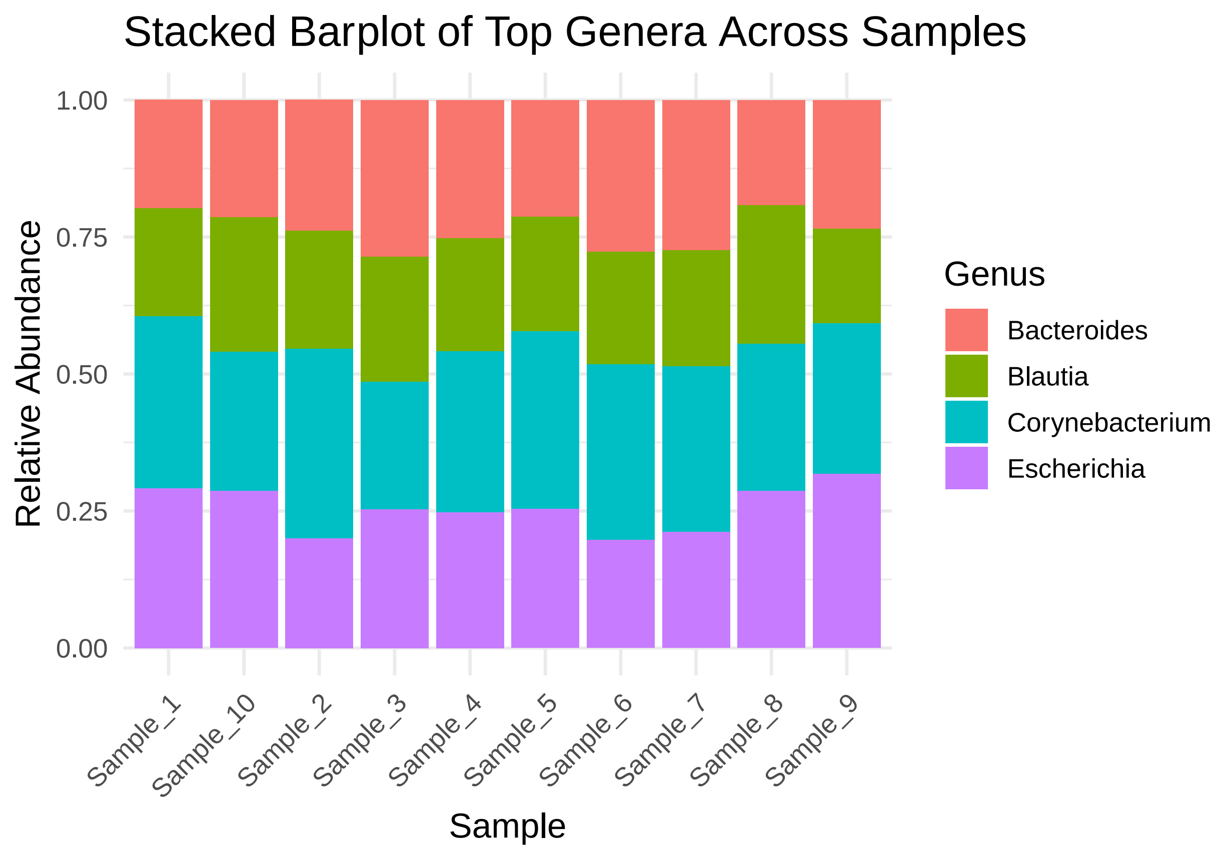
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load OTU table and taxonomy
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0)
tax_df = pd.read_csv("data/otu_taxonomy.tsv")
```

```
# Merge taxonomy info (Genus) with OTU table
merged = otu_df.merge(tax_df[["OTU_ID", "Genus"]], left_index=True, right_on="OTU_ID")
melted = merged.drop("OTU_ID", axis=1).melt(id_vars="Genus", var_name="Sample", value_na

# Summarize top 8 genera, lump rest as 'Other'
top_genera = melted.groupby("Genus")["Abundance"].sum().nlargest(8).index
melted["Genus"] = melted["Genus"].where(melted["Genus"].isin(top_genera), "Other")

# Normalize per sample
melted = melted.groupby(["Sample", "Genus"])["Abundance"].sum().reset_index()
melted["RelativeAbundance"] = melted.groupby("Sample")["Abundance"].transform(lambda x:

# Plot
plt.figure(figsize=(12, 5))
sns.barplot(data=melted, x="Sample", y="RelativeAbundance", hue="Genus")
plt.title("Stacked Barplot of Top Genera Across Samples")
plt.ylabel("Relative Abundance")
plt.xticks(rotation=45)
plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left")
plt.tight_layout()
plt.show()
```

8.3 R Code

```
library(tidyverse)

# Load OTU and taxonomy tables
otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
tax_df <- read.delim("data/otu_taxonomy.tsv")

# Merge by rownames (OTUs)
otu_df$OTU_ID <- rownames(otu_df)
```

```r
merged_df <- left_join(otu_df, tax_df, by = "OTU_ID")

# Convert to long format and summarize
long_df <- merged_df %>%
  pivot_longer(cols = starts_with("Sample"), names_to = "Sample", values_to = "Abundanc
  group_by(Sample, Genus) %>%
  summarise(Abundance = sum(Abundance), .groups = "drop") %>%
  group_by(Sample) %>%
  mutate(RelativeAbundance = Abundance / sum(Abundance))

# Keep top 8 genera
top_genera <- long_df %>%
  group_by(Genus) %>%
  summarise(Total = sum(RelativeAbundance), .groups = "drop") %>%
  top_n(8, Total) %>%
  pull(Genus)

long_df <- long_df %>%
  mutate(Genus = if_else(Genus %in% top_genera, Genus, "Other"))

# Plot
ggplot(long_df, aes(x = Sample, y = RelativeAbundance, fill = Genus)) +
  geom_col() +
  theme_minimal(base_size = 13) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Stacked Barplot of Top Genera Across Samples", y = "Relative Abundance")
```



Stacked Barplot of Top Genera Across Samples
Relative Abundance
1.00
0.75
0.50
0.25
0.00
Sample_1
Sample_10
Sample_2
Sample_3
Sample_4
Sample_5
Sample_6
Sample_7
Sample_8
Sample_9
Sample
Genus
Bacteroides
Blautia
Corynebacterium
Escherichia

Q&A: 9

How do I visualize alpha diversity (richness) across groups?

9.1 Explanation

Alpha diversity measures **within-sample diversity** — often captured by the number of observed OTUs or ASVs (richness).

Visualizing alpha diversity across experimental groups (e.g., Control vs Treatment) helps detect differences in microbial complexity. Boxplots are commonly used for this purpose.

To generate the plot, we: - Sum OTUs per sample - Merge with metadata - Group by condition (e.g., Treatment group)

9.2 Python Code

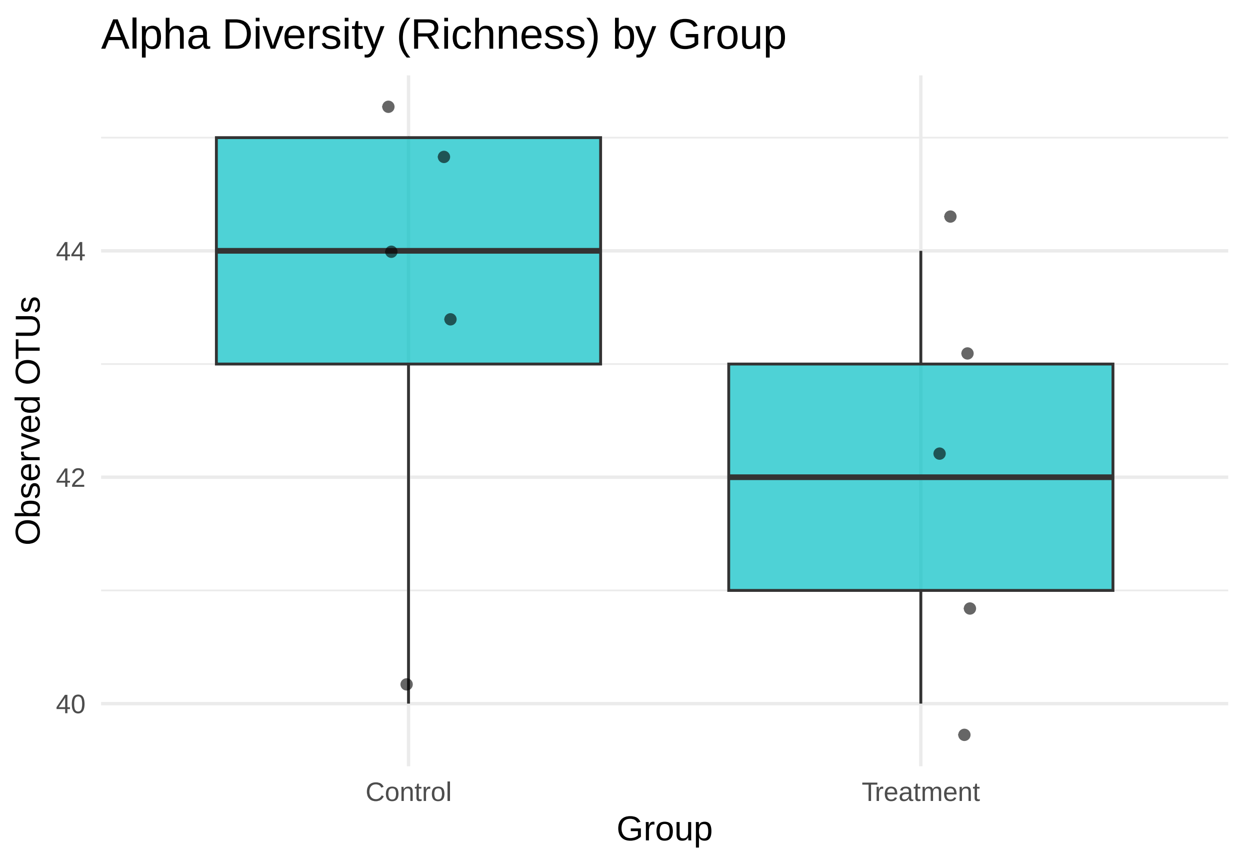
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load OTU table and metadata
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0)
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
```

```python
# Compute richness
richness = pd.DataFrame({
    "sample_id": otu_df.columns,
    "richness": (otu_df > 0).sum(axis=0).values
})

# Merge with metadata
merged = pd.merge(richness, meta_df, on="sample_id")

# Plot
plt.figure(figsize=(8, 5))
sns.boxplot(data=merged, x="group", y="richness", palette="Set2")
sns.stripplot(data=merged, x="group", y="richness", color='black', alpha=0.5)
plt.title("Alpha Diversity (Richness) by Group")
plt.ylabel("Observed OTUs")
plt.xlabel("Group")
plt.tight_layout()
plt.show()
```

9.3 R Code

```r
library(tidyverse)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")

# Compute richness
richness <- colSums(otu_df > 0)
richness_df <- data.frame(sample_id = names(richness), richness = richness)

# Merge with metadata
merged <- left_join(richness_df, meta_df, by = "sample_id")
```

```
# Plot
ggplot(merged, aes(x = group, y = richness)) +
  geom_boxplot(fill = "#00BFC4", alpha = 0.7) +
  geom_jitter(width = 0.1, color = "black", alpha = 0.6) +
  theme_minimal(base_size = 13) +
  labs(title = "Alpha Diversity (Richness) by Group", y = "Observed OTUs", x = "Group")
```



Q&A: 10

How do I perform ordination (e.g., PCA) to visualize sample clustering?

10.1 Explanation

Ordination techniques like **PCA**, **NMDS**, or **PCoA** help reduce the complexity of high-dimensional OTU tables, making it easier to visualize **sample relationships**.

These methods project samples into 2D or 3D based on similarity in microbial composition. Samples that cluster together share similar community profiles.

In this example, we'll perform PCA on centered log-ratio (CLR) transformed data — a common preprocessing step in microbiome compositional analysis.

10.2 Python Code

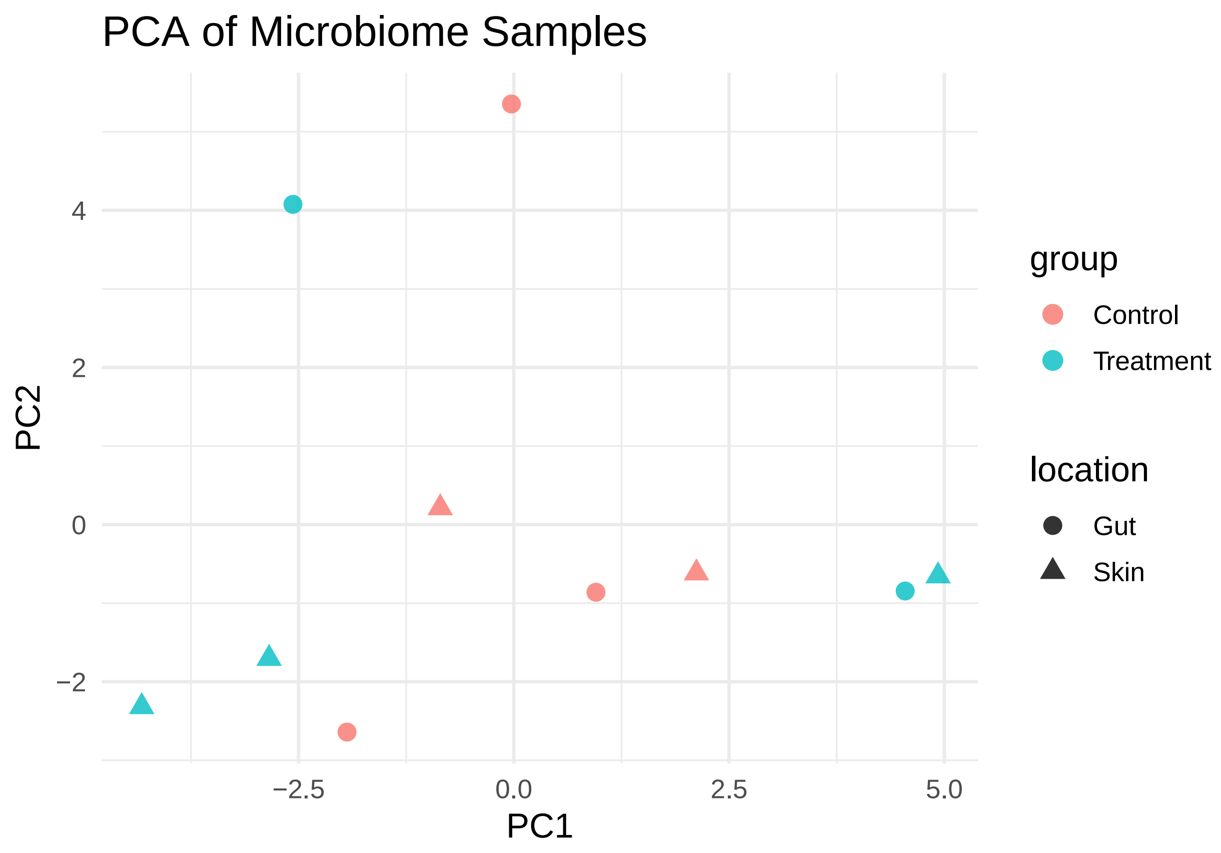
```python
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt

# Load OTU and metadata
```

```python
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")

# Replace 0s with pseudocount for log transformation
otu_df += 1
otu_log = np.log(otu_df)

# Standardize (optional)
otu_scaled = StandardScaler().fit_transform(otu_log)

# PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(otu_scaled)
pca_df = pd.DataFrame(pca_result, columns=["PC1", "PC2"])
pca_df["sample_id"] = otu_df.index

# Merge with metadata
pca_df = pd.merge(pca_df, meta_df, on="sample_id")

# Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(data=pca_df, x="PC1", y="PC2", hue="group", style="location", s=100)
plt.title("PCA of Microbiome Samples")
plt.xlabel(f"PC1 ({pca.explained_variance_ratio_[0]:.2%})")
plt.ylabel(f"PC2 ({pca.explained_variance_ratio_[1]:.2%})")
plt.tight_layout()
plt.show()
```

10.3 R Code

```r
library(tidyverse)
library(vegan)
```

```r
otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")

# CLR transformation (log + pseudocount)
otu_clr <- log(otu_df + 1)
otu_scaled <- scale(t(otu_clr))  # samples as rows

# PCA
pca_res <- prcomp(otu_scaled, center = TRUE, scale. = TRUE)
pca_df <- as.data.frame(pca_res$x[, 1:2])
pca_df$sample_id <- rownames(pca_df)

# Merge
merged <- left_join(pca_df, meta_df, by = "sample_id")

# Plot
ggplot(merged, aes(x = PC1, y = PC2, color = group, shape = location)) +
  geom_point(size = 3, alpha = 0.8) +
  theme_minimal(base_size = 13) +
  labs(title = "PCA of Microbiome Samples", x = "PC1", y = "PC2")
```


PCA of Microbiome Samples
PC2
4
2
0
−2
−2.5
0.0
2.5
5.0
PC1
group
Control
Treatment
location
Gut
Skin

Q&A: 11

How do I visualize OTU or Genus abundance using a heatmap?

11.1 Explanation

Heatmaps are excellent for visualizing microbial abundance patterns across samples. They help identify: - Co-occurring OTUs or genera - Sample clusters with similar profiles - High- or low-abundance taxa patterns

Heatmaps often include clustering on rows (features) and columns (samples), with scaling or log-transformation to improve interpretability.

In this example, we visualize the **top 20 most abundant OTUs** across all samples.

11.2 Python Code

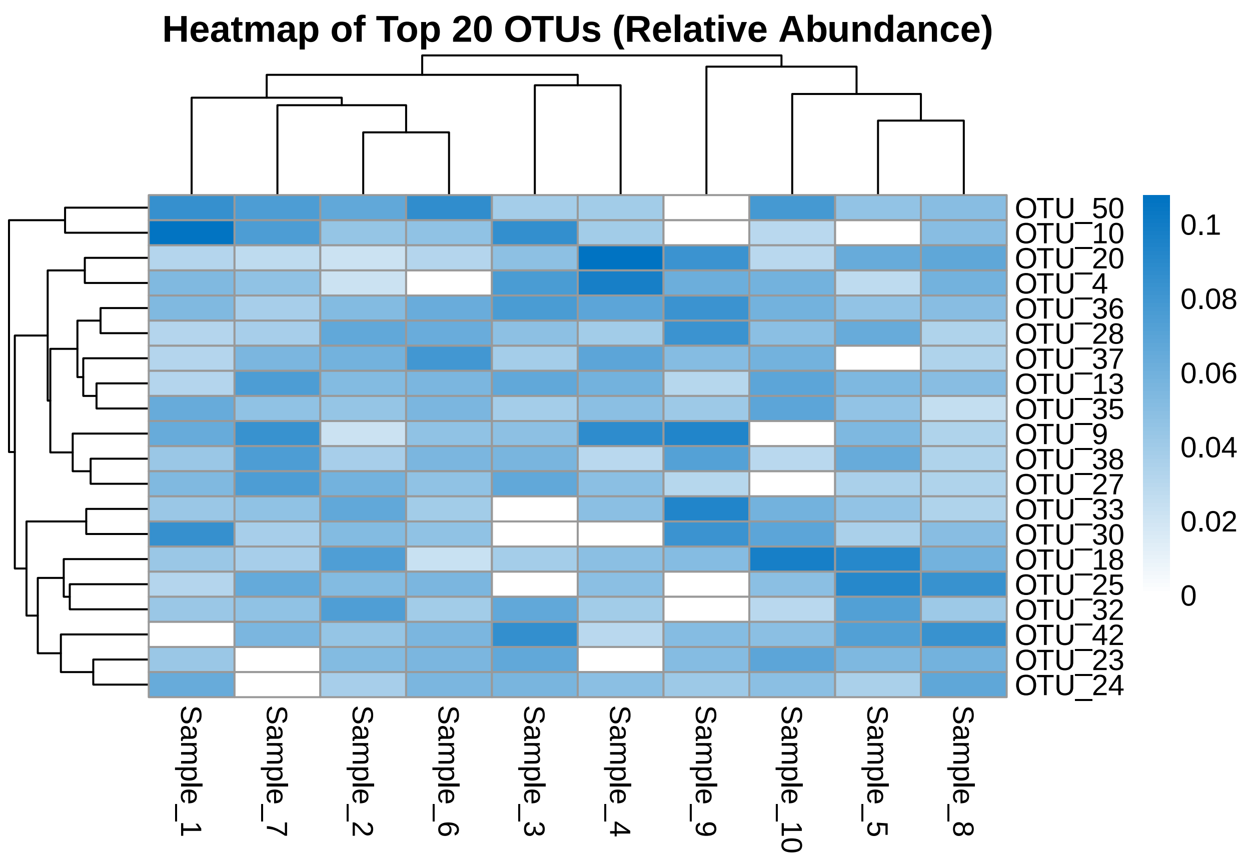
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load OTU table
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0)
```

```python
# Select top 20 OTUs by total abundance
top_otus = otu_df.sum(axis=1).nlargest(20).index
top_otu_df = otu_df.loc[top_otus]

# Normalize (relative abundance per sample)
rel_abund = top_otu_df.div(top_otu_df.sum(axis=0), axis=1)

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(rel_abund, cmap="YlGnBu", linewidths=0.5)
plt.title("Heatmap of Top 20 OTUs (Relative Abundance)")
plt.xlabel("Samples")
plt.ylabel("OTUs")
plt.tight_layout()
plt.show()
```

11.3 R Code

```r
library(tidyverse)
library(pheatmap)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)

# Select top 20 OTUs by abundance
top_otus <- rowSums(otu_df) %>%
  sort(decreasing = TRUE) %>%
  head(20) %>%
  names()

top_otu_df <- otu_df[top_otus, ]

# Convert to relative abundance
```

```r
rel_abund <- sweep(top_otu_df, 2, colSums(top_otu_df), FUN = "/")

# Plot heatmap
pheatmap(rel_abund,
         color = colorRampPalette(c("white", "#0073C2FF"))(100),
         fontsize = 11,
         main = "Heatmap of Top 20 OTUs (Relative Abundance)")
```



Part IV

MICROBIOME STATS

Q&A: 12

How do I statistically compare OTU richness between groups?

12.1 Explanation

Once you've calculated **alpha diversity** (e.g., observed OTUs per sample), it's common to test whether **groups differ significantly**. This can help you determine if an experimental condition affects microbial richness.

Typical tests include: - **T-test** (for two groups, normally distributed data) - **Wilcoxon rank-sum test** (non-parametric) - **ANOVA or Kruskal-Wallis** (for 3+ groups)

Here we demonstrate group comparison for **richness** using appropriate statistical tests.

12.2 Python Code

```
import pandas as pd
from scipy.stats import ttest_ind, mannwhitneyu

# Load richness and metadata
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0)
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
```

```python
# Compute richness
richness = pd.DataFrame({
    "sample_id": otu_df.columns,
    "richness": (otu_df > 0).sum(axis=0).values
})
data = pd.merge(richness, meta_df, on="sample_id")

# Split by group
control = data[data["group"] == "Control"]["richness"]
treatment = data[data["group"] == "Treatment"]["richness"]

# T-test (assumes normality)
t_stat, t_pval = ttest_ind(control, treatment)

# Wilcoxon (non-parametric)
w_stat, w_pval = mannwhitneyu(control, treatment)

print(f"T-test p-value: {t_pval:.4f}")
print(f"Wilcoxon test p-value: {w_pval:.4f}")
```

12.3 R Code

```r
library(tidyverse)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")

# Calculate richness
richness <- colSums(otu_df > 0)
richness_df <- data.frame(sample_id = names(richness), richness = richness)
merged <- left_join(richness_df, meta_df, by = "sample_id")
```

```r
# T-test
t_test <- t.test(richness ~ group, data = merged)

# Wilcoxon test
wilcox_test <- wilcox.test(richness ~ group, data = merged)

t_test$p.value
```

```
[1] 0.2666018
```

```r
wilcox_test$p.value
```

```
[1] 0.2447901
```

Q&A: 13

How do I test for correlation between alpha diversity and age?

13.1 Explanation

In many studies, you may want to examine whether microbial diversity is associated with continuous metadata like **age**, **BMI**, or **pH**.

Correlation tests help assess linear or monotonic relationships between variables: - **Pearson correlation**: for linear relationships (assumes normality) - **Spearman correlation**: for monotonic (rank-based) associations (non-parametric)

This Q&A demonstrates testing correlation between **richness** and **age**.

13.2 Python Code

```
import pandas as pd
from scipy.stats import pearsonr, spearmanr

# Load OTU and metadata
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0)
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
```

```python
# Compute richness
richness = pd.DataFrame({
    "sample_id": otu_df.columns,
    "richness": (otu_df > 0).sum(axis=0).values
})
data = pd.merge(richness, meta_df, on="sample_id")

# Pearson correlation
pearson_corr, pearson_pval = pearsonr(data["richness"], data["age"])

# Spearman correlation
spearman_corr, spearman_pval = spearmanr(data["richness"], data["age"])

print(f"Pearson r: {pearson_corr:.3f}, p = {pearson_pval:.4f}")
print(f"Spearman rho: {spearman_corr:.3f}, p = {spearman_pval:.4f}")
```

13.3 R Code

```r
library(tidyverse)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")

# Compute richness
richness <- colSums(otu_df > 0)
richness_df <- data.frame(sample_id = names(richness), richness = richness)
merged <- left_join(richness_df, meta_df, by = "sample_id")

# Pearson correlation
cor.test(merged$richness, merged$age, method = "pearson")
```

```

    Pearson's product-moment correlation

data:  merged$richness and merged$age
t = -0.099992, df = 8, p-value = 0.9228
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.6504867  0.6078167
sample estimates:
        cor 
-0.03533044 
```

```r
# Spearman correlation
cor.test(merged$richness, merged$age, method = "spearman")
```

```

    Spearman's rank correlation rho

data:  merged$richness and merged$age
S = 179.17, p-value = 0.8135
alternative hypothesis: true rho is not equal to 0
sample estimates:
        rho 
-0.08589604 
```

Q&A: 14

How do I compare alpha diversity across 3 or more groups?

14.1 Explanation

When comparing microbial richness across more than two groups, you can use: - **ANOVA**: if data are normally distributed - **Kruskal-Wallis test**: non-parametric alternative

This Q&A tests whether OTU richness differs by **body location** (e.g., gut, skin).

14.2 Python Code

```python
import pandas as pd
from scipy.stats import f_oneway, kruskal

# Load data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0)
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")

# Compute richness
richness = pd.DataFrame({
    "sample_id": otu_df.columns,
    "richness": (otu_df > 0).sum(axis=0).values
})
data = pd.merge(richness, meta_df, on="sample_id")

# Split richness by location
groups = [group["richness"].values for name, group in data.groupby("location")]

# ANOVA
f_stat, f_pval = f_oneway(*groups)

# Kruskal-Wallis
kw_stat, kw_pval = kruskal(*groups)

print(f"ANOVA p-value: {f_pval:.4f}")
print(f"Kruskal-Wallis p-value: {kw_pval:.4f}")
```

14.3 R Code

```
library(tidyverse)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")

# Compute richness
richness <- colSums(otu_df > 0)
richness_df <- data.frame(sample_id = names(richness), richness = richness)
merged <- left_join(richness_df, meta_df, by = "sample_id")

# ANOVA
anova_res <- aov(richness ~ location, data = merged)

# Kruskal-Wallis
```

```r
kruskal_res <- kruskal.test(richness ~ location, data = merged)

summary(anova_res)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
location     1    2.5     2.5   0.676  0.435
Residuals    8   29.6     3.7
```

```r
kruskal_res
```

```

	Kruskal-Wallis rank sum test

data:  richness by location
Kruskal-Wallis chi-squared = 0.71553, df = 1, p-value = 0.3976
```

Q&A: 15

How do I test for differences in community composition using PERMANOVA?

15.1 Explanation

PERMANOVA (Permutational Multivariate Analysis of Variance) is used to test whether **beta diversity** significantly differs between groups.

It operates on a dissimilarity matrix (e.g., Bray-Curtis) and partitions variation based on experimental factors like **treatment group** or **location**.

This Q&A applies PERMANOVA to Bray-Curtis distances computed from OTU abundances.

15.2 Python Code

```
import pandas as pd
from skbio.diversity import beta_diversity
from skbio.stats.distance import permanova
from skbio import DistanceMatrix

# Load data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
```

```python
# Compute Bray-Curtis distance matrix
bray_dm = beta_diversity("braycurtis", otu_df.values, ids=otu_df.index)

# Format metadata
meta_df = meta_df.set_index("sample_id").loc[otu_df.index]

# Run PERMANOVA
result = permanova(distance_matrix=bray_dm, grouping=meta_df["group"], permutations=999)
print(result)
```

15.3 R Code

```r
library(vegan)
library(tidyverse)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")

# Bray-Curtis distance
otu_t <- t(otu_df)
bray <- vegdist(otu_t, method = "bray")

# Match metadata
meta_df <- meta_df %>% filter(sample_id %in% rownames(otu_t)) %>% column_to_rownames("s

# Run PERMANOVA (adonis)
adonis_res <- adonis2(bray ~ group, data = meta_df, permutations = 999)

adonis_res
```

	Df	SumOfSqs	R2	F	Pr(>F)
Model	1	0.0374422	0.0811772	0.7067931	0.879
Residual	8	0.4237981	0.9188228	NA	NA
Total	9	0.4612403	1.0000000	NA	NA

Q&A: 16

How do I test for differential abundance of OTUs across groups?

16.1 Explanation

Differential abundance analysis identifies OTUs that significantly differ between groups (e.g., Control vs Treatment).

While tools like **DESeq2** are used for RNA-seq and microbiome count data, simpler methods like: - **Wilcoxon tests** - **t-tests** - **ANCOM / ALDEx2** (specialized tools)
can also be used with filtered OTU data.

Here we demonstrate how to test one OTU at a time between groups.

16.2 Python Code

```
import pandas as pd
from scipy.stats import mannwhitneyu

# Load OTU table and metadata
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0)
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t").set_index("sample_id")
otu_df = otu_df[meta_df.index]  # ensure matching
```

```python
# Perform Wilcoxon test for each OTU
results = []
for otu in otu_df.index:
    control = otu_df.loc[otu, meta_df["group"] == "Control"]
    treatment = otu_df.loc[otu, meta_df["group"] == "Treatment"]
    stat, pval = mannwhitneyu(control, treatment)
    results.append((otu, pval))

# Convert to DataFrame
df_results = pd.DataFrame(results, columns=["OTU", "p_value"])
df_results["adjusted_p"] = df_results["p_value"] * len(df_results)  # Bonferroni
df_results.sort_values("p_value").head()
```

16.3 R Code

```r
library(tidyverse)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")

# Ensure sample order matches
otu_df <- otu_df[, meta_df$sample_id]

# Run Wilcoxon test for each OTU
results <- apply(otu_df, 1, function(x) {
  group <- meta_df$group
  test <- wilcox.test(x[group == "Control"], x[group == "Treatment"])
  return(test$p.value)
})

df_results <- data.frame(OTU = rownames(otu_df), p_value = results)
df_results$adjusted_p <- p.adjust(df_results$p_value, method = "bonferroni")
```

```r
head(df_results[order(df_results$p_value), ])
```

	OTU	p_value	adjusted_p
OTU_14	OTU_14	0.0196244	0.9812207
OTU_37	OTU_37	0.0585526	1.0000000
OTU_38	OTU_38	0.1104920	1.0000000
OTU_16	OTU_16	0.1326223	1.0000000
OTU_26	OTU_26	0.1612376	1.0000000
OTU_9	OTU_9	0.1988289	1.0000000

Part V

MICROBIOME MACHINE LEARNING

Q&A: 17

How do I prepare microbiome data for machine learning?

17.1 Explanation

Before applying machine learning, you must structure your OTU table and metadata into a form suitable for modeling.

Typical steps include: - **Filtering**: Keep relevant OTUs/features - **Merging**: Align OTU table with sample metadata - **Encoding**: Set up group labels (e.g., Control = 0, Treatment = 1) - **Splitting**: Train-test split to evaluate generalizability

This Q&A sets up data for classification.

17.2 Python Code

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
```

```python
# Merge OTU table with metadata by sample
data = pd.merge(otu_df, meta_df, left_index=True, right_on="sample_id")

# Define features (X) and labels (y)
X = data[otu_df.columns]  # OTU features
y = data["group"].map({"Control": 0, "Treatment": 1})  # binary encoding

# Split into training/testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

# Check shape
print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

17.3 R Note

```r
# Most ML pipelines in microbiome analysis are performed in Python.
# In R, similar workflows can be built using caret, tidymodels, or mlr3.
```

Q&A: 18

How do I train and evaluate a Random Forest classifier on microbiome data?

18.1 Explanation

The **Random Forest** algorithm is a popular and robust model for microbiome classification tasks due to its: - Built-in feature importance - Resistance to overfitting - Non-linear modeling capability

This Q&A demonstrates how to: - Train a Random Forest classifier - Evaluate it using accuracy and confusion matrix - Inspect important OTUs

18.2 Python Code

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split

# Load and prepare data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
data = pd.merge(otu_df, meta_df, left_index=True, right_on="sample_id")
```

```
X = data[otu_df.columns]
y = data["group"].map({"Control": 0, "Treatment": 1})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

# Train Random Forest
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict and evaluate
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {acc:.2f}")
print("Confusion Matrix:")
print(cm)

# Top 5 important OTUs
feat_imp = pd.Series(clf.feature_importances_, index=X.columns)
print("Top OTUs:
", feat_imp.sort_values(ascending=False).head())
```

18.3 R Code (caret)

```
library(tidyverse)
library(caret)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")
otu_df <- otu_df[, meta_df$sample_id]
otu_df <- t(otu_df)
data <- cbind(as.data.frame(otu_df), group = meta_df$group)
```

```r
# Encode group and split
data$group <- as.factor(data$group)
set.seed(42)
trainIndex <- createDataPartition(data$group, p = .7, list = FALSE)
train <- data[trainIndex, ]
test  <- data[-trainIndex, ]

# Train Random Forest
rf_model <- train(group ~ ., data = train, method = "rf", trControl = trainControl(metho

# Predict and evaluate
pred <- predict(rf_model, newdata = test)
confusionMatrix(pred, test$group)
```

```
Confusion Matrix and Statistics

           Reference
Prediction  Control Treatment
  Control         0         1
  Treatment       1         0

               Accuracy : 0
                 95% CI : (0, 0.8419)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 1

                  Kappa : -1

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.0
            Specificity : 0.0
         Pos Pred Value : 0.0
         Neg Pred Value : 0.0
```

```
          Prevalence : 0.5
      Detection Rate : 0.0
Detection Prevalence : 0.5
   Balanced Accuracy : 0.0

    'Positive' Class : Control
```

Q&A: 19

How do I build a Logistic Regression model for microbiome classification?

19.1 Explanation

Logistic Regression is a foundational classification model useful for: - Binary prediction (e.g., Control vs Treatment) - Interpreting OTU effects via coefficients - Establishing baselines before more complex models

This Q&A shows how to train and evaluate a Logistic Regression model.

19.2 Python Code

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split

# Load and prepare data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
data = pd.merge(otu_df, meta_df, left_index=True, right_on="sample_id")
```

```
X = data[otu_df.columns]
y = data["group"].map({"Control": 0, "Treatment": 1})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

# Train logistic regression model
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {acc:.2f}")
print("Confusion Matrix:")
print(cm)

# Inspect top OTUs by absolute coefficient magnitude
coef = pd.Series(model.coef_[0], index=X.columns)
print("Top OTUs:
", coef.abs().sort_values(ascending=False).head())
```

19.3 R Code (caret)

```
library(tidyverse)
library(caret)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")
otu_df <- otu_df[, meta_df$sample_id]
otu_df <- t(otu_df)
data <- cbind(as.data.frame(otu_df), group = meta_df$group)
```

```r
# Encode group and split
data$group <- as.factor(data$group)
set.seed(42)
trainIndex <- createDataPartition(data$group, p = .7, list = FALSE)
train <- data[trainIndex, ]
test  <- data[-trainIndex, ]

# Train logistic regression
lr_model <- train(group ~ ., data = train, method = "glm", family = "binomial", trContro

# Predict and evaluate
pred <- predict(lr_model, newdata = test)
confusionMatrix(pred, test$group)
```

```
Confusion Matrix and Statistics

           Reference
Prediction  Control Treatment
  Control         1         0
  Treatment       0         1

               Accuracy : 1
                 95% CI : (0.1581, 1)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 0.25

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0
            Specificity : 1.0
         Pos Pred Value : 1.0
         Neg Pred Value : 1.0
```

```
           Prevalence : 0.5
       Detection Rate : 0.5
 Detection Prevalence : 0.5
    Balanced Accuracy : 1.0

     'Positive' Class : Control
```

Q&A: 20

How do I train a Support Vector Machine (SVM) for microbiome classification?

20.1 Explanation

Support Vector Machines (SVM) are powerful classifiers for high-dimensional biological data. They find a decision boundary (hyperplane) that maximizes class separation.

SVMs are well-suited for microbiome data because: - They can handle many features (OTUs) - Work with kernel functions for non-linear separation - Often perform well with sparse data

20.2 Python Code

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split

# Load and prepare data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
data = pd.merge(otu_df, meta_df, left_index=True, right_on="sample_id")
```

```
X = data[otu_df.columns]
y = data["group"].map({"Control": 0, "Treatment": 1})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

# Train SVM
model = SVC(kernel='linear', probability=True, random_state=42)
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {acc:.2f}")
print("Confusion Matrix:")
print(cm)
```

20.3 R Code (caret)

```
library(tidyverse)
library(caret)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")
otu_df <- otu_df[, meta_df$sample_id]
otu_df <- t(otu_df)
data <- cbind(as.data.frame(otu_df), group = meta_df$group)

# Encode group and split
data$group <- as.factor(data$group)
set.seed(42)
trainIndex <- createDataPartition(data$group, p = .7, list = FALSE)
```

```r
train <- data[trainIndex, ]
test  <- data[-trainIndex, ]

# Train SVM
svm_model <- train(group ~ ., data = train, method = "svmLinear", trControl = trainContr

# Predict and evaluate
pred <- predict(svm_model, newdata = test)
confusionMatrix(pred, test$group)
```

```
Confusion Matrix and Statistics

           Reference
Prediction  Control Treatment
  Control         0         0
  Treatment       1         1

               Accuracy : 0.5
                 95% CI : (0.0126, 0.9874)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 0.75

                  Kappa : 0

 Mcnemar's Test P-Value : 1.00

            Sensitivity : 0.0
            Specificity : 1.0
         Pos Pred Value : NaN
         Neg Pred Value : 0.5
             Prevalence : 0.5
         Detection Rate : 0.0
   Detection Prevalence : 0.0
      Balanced Accuracy : 0.5
```

```
'Positive' Class : Control
```

Q&A: 21

How do I apply Gradient Boosting (XGBoost) for microbiome classification?

21.1 Explanation

XGBoost is an efficient, scalable gradient boosting algorithm widely used in biological classification tasks. It's known for: - High performance on structured/tabular data - Handling of missing values - Built-in feature importance metrics

This Q&A shows how to train and evaluate an XGBoost classifier.

21.2 Python Code

```
import pandas as pd
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split

# Load and prepare data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
data = pd.merge(otu_df, meta_df, left_index=True, right_on="sample_id")
```

```
X = data[otu_df.columns]
y = data["group"].map({"Control": 0, "Treatment": 1})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

# Train XGBoost classifier
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {acc:.2f}")
print("Confusion Matrix:")
print(cm)

# Top important OTUs
importances = pd.Series(model.feature_importances_, index=X.columns)
print("Top Features:
", importances.sort_values(ascending=False).head())
```

21.3 R Code (caret + xgboost)

```
library(tidyverse)
library(caret)
library(xgboost)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")
otu_df <- otu_df[, meta_df$sample_id]
otu_df <- t(otu_df)
```

```r
data <- cbind(as.data.frame(otu_df), group = meta_df$group)

# Encode group and split
data$group <- as.factor(data$group)
set.seed(42)
trainIndex <- createDataPartition(data$group, p = .7, list = FALSE)
train <- data[trainIndex, ]
test  <- data[-trainIndex, ]

# Train XGBoost
xgb_model <- train(group ~ ., data = train, method = "xgbTree", trControl = trainControl
```

```
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
[00:04:29] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_
```

```r
# Predict and evaluate
pred <- predict(xgb_model, newdata = test)
confusionMatrix(pred, test$group)
```

```
Confusion Matrix and Statistics
```

```
          Reference
Prediction  Control Treatment
  Control         1         1
  Treatment       0         0

               Accuracy : 0.5
                 95% CI : (0.0126, 0.9874)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 0.75

                  Kappa : 0

 Mcnemar's Test P-Value : 1.00

            Sensitivity : 1.0
            Specificity : 0.0
         Pos Pred Value : 0.5
         Neg Pred Value : NaN
             Prevalence : 0.5
         Detection Rate : 0.5
   Detection Prevalence : 1.0
      Balanced Accuracy : 0.5

       'Positive' Class : Control
```

Q&A: 22

How do I visualize ROC curves to compare classification models?

22.1 Explanation

Receiver Operating Characteristic (ROC) curves help visualize model performance across different thresholds. The **Area Under the Curve (AUC)** summarizes performance — closer to 1.0 is better.

Comparing ROC curves across models (e.g., Random Forest, Logistic Regression, XGBoost) provides insight into which performs best and where they differ.

This Q&A demonstrates ROC curve generation in Python and R.

22.2 Python Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

```
# Load and prepare data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
data = pd.merge(otu_df, meta_df, left_index=True, right_on="sample_id")
X = data[otu_df.columns]
y = data["group"].map({"Control": 0, "Treatment": 1})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

# Define models
models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="logloss", random_stat
}

# Plot ROC curves
plt.figure(figsize=(8, 6))
for name, model in models.items():
    model.fit(X_train, y_train)
    probas = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, probas)
    auc_score = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")

plt.plot([0, 1], [0, 1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.tight_layout()
plt.show()
```

22.3 R Code (caret + pROC)

```r
library(tidyverse)
library(caret)
library(pROC)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")
otu_df <- otu_df[, meta_df$sample_id]
otu_df <- t(otu_df)
data <- cbind(as.data.frame(otu_df), group = as.factor(meta_df$group))

# Train/test split
set.seed(42)
trainIndex <- createDataPartition(data$group, p = .7, list = FALSE)
train <- data[trainIndex, ]
test  <- data[-trainIndex, ]

# Define fixed tuning for models that require it
rf_grid <- data.frame(mtry = floor(sqrt(ncol(train) - 1)))
svm_grid <- data.frame(C = 1)
xgb_grid <- data.frame(
  nrounds = 50,
  max_depth = 3,
  eta = 0.3,
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

# Train models using fixed tuneGrid
models <- list(
  rf = train(group ~ ., data = train, method = "rf", trControl = trainControl(method = 
  glm = train(group ~ ., data = train, method = "glm", family = "binomial", trControl = 
  svm = train(group ~ ., data = train, method = "svmLinear", trControl = trainControl(me
  xgb = train(group ~ ., data = train, method = "xgbTree", trControl = trainControl(meth
)

# ROC analysis
roc_list <- lapply(models, function(model) {
  probs <- predict(model, newdata = test, type = "prob")
  # Safely extract numeric probabilities for the "Treatment" class
  class_label <- "Treatment"
  if (!(class_label %in% colnames(probs))) {
    stop(paste("Class label", class_label, "not found in predicted probabilities"))
  }
  prob_values <- as.numeric(probs[, class_label])
  roc(response = test$group, predictor = prob_values)
})

# Plot ROC
plot(roc_list[[1]], col = "blue", legacy.axes = TRUE, main = "ROC Curves - Microbiome Cl
cols <- c("blue", "green", "red", "purple")
for (i in 2:length(roc_list)) {
  plot(roc_list[[i]], col = cols[i], add = TRUE)
}
legend("bottomright", legend = names(models), col = cols, lwd = 2)
```

Q&A: 23

How do I apply cross-validation strategies to evaluate model reliability?

23.1 Explanation

Cross-validation (CV) is critical for evaluating how well a machine learning model generalizes to unseen data. It reduces the risk of overfitting by testing the model on multiple train/test splits.

Common CV strategies: - **k-Fold**: Split into k subsets, rotate test set - **Repeated k-Fold**: More robust by repeating k-fold several times - **Stratified k-Fold**: Ensures balanced class distribution in folds

This Q&A shows how to apply CV in Python and R using common microbiome classifiers.

23.2 Python Code

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold

# Load and prepare data
otu_df = pd.read_csv("data/otu_table_filtered.tsv", sep="\t", index_col=0).T
meta_df = pd.read_csv("data/sample_metadata.tsv", sep="\t")
```

```
data = pd.merge(otu_df, meta_df, left_index=True, right_on="sample_id")
X = data[otu_df.columns]
y = data["group"].map({"Control": 0, "Treatment": 1})

# Define CV strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Evaluate Random Forest using cross-validation
model = RandomForestClassifier(random_state=42)
scores = cross_val_score(model, X, y, cv=cv, scoring="accuracy")

print("Cross-Validation Accuracy Scores:", scores)
print("Mean Accuracy:", scores.mean())
```

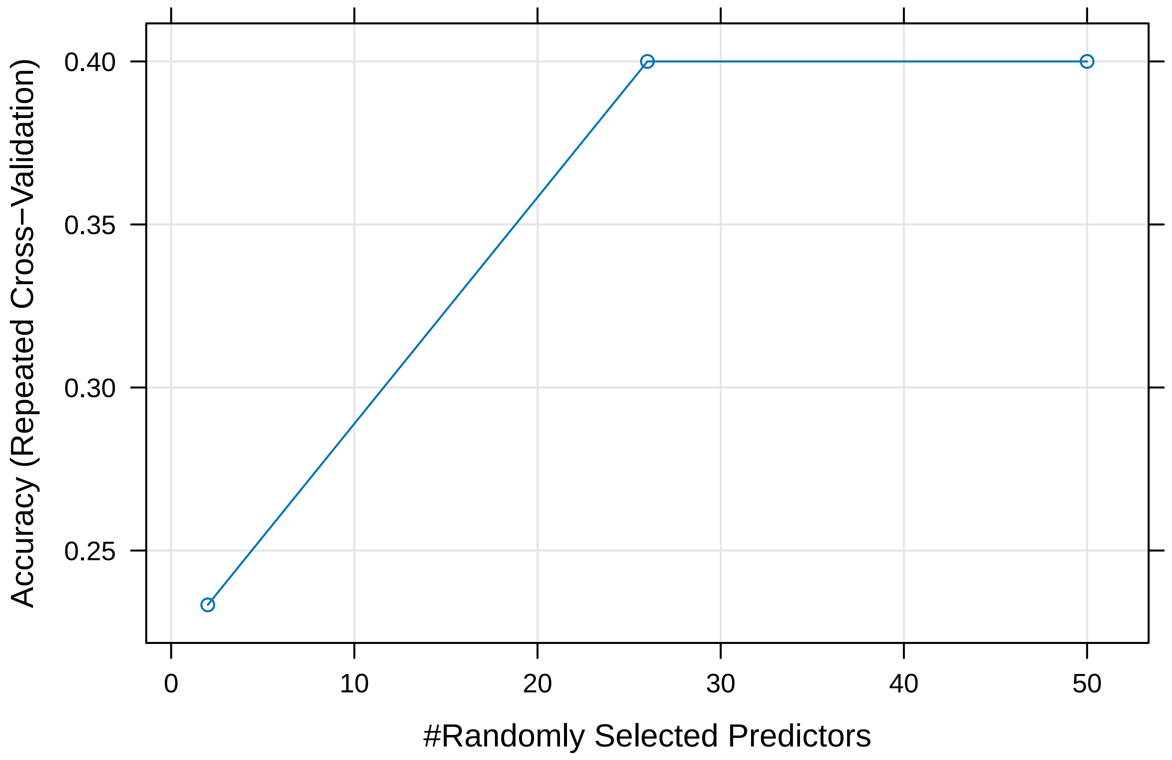
23.3 R Code (caret with repeated k-fold CV)

```r
library(tidyverse)
library(caret)

otu_df <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta_df <- read.delim("data/sample_metadata.tsv")
otu_df <- otu_df[, meta_df$sample_id]
otu_df <- t(otu_df)
data <- cbind(as.data.frame(otu_df), group = as.factor(meta_df$group))

# Define CV control
ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3)

# Train with CV
set.seed(42)
cv_model <- train(group ~ ., data = data, method = "rf", trControl = ctrl)
```

```r
# Results
print(cv_model)
```

```
Random Forest

10 samples
50 predictors
 2 classes: 'Control', 'Treatment'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 8, 8, 8, 8, 8, 8, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.2333333  -0.5333333
  26    0.4000000  -0.2000000
  50    0.4000000  -0.2000000

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 26.
```

```r
plot(cv_model)
```



Accuracy (Repeated Cross−Validation)
0.40
0.35
0.30
0.25
0
10
20
30
40
50
#Randomly Selected Predictors

Q&A: 24

How do I use `mikropml` in R for microbiome machine learning?

24.1 Explanation

`mikropml` is a microbiome-focused R package by Pat Schloss designed for: - End-to-end modeling workflows - Built-in cross-validation and hyperparameter tuning - Transparency in model reporting and evaluation

It simplifies the process of building, tuning, and interpreting microbiome ML models.

This Q&A introduces a basic pipeline using `mikropml` and prepares the OTU and metadata files as expected.

24.2 R Code

```
#  Ensure mikropml is installed
if (!requireNamespace("mikropml", quietly = TRUE)) {
  if (!requireNamespace("remotes", quietly = TRUE)) install.packages("remotes")
  remotes::install_github("SchlossLab/mikropml")
}

library(mikropml)
```

```r
library(tidyverse)

# Load OTU table and metadata
otu <- read.delim("data/otu_table_filtered.tsv", row.names = 1)
meta <- read.delim("data/sample_metadata.tsv")

# Transpose OTU so samples are rows
otu_t <- t(otu)
otu_df <- as.data.frame(otu_t)
otu_df$sample_id <- rownames(otu_t)

# Merge with metadata
data <- inner_join(otu_df, meta, by = "sample_id")

# Run mikropml using run_ml()
set.seed(42)
fit <- run_ml(
  dataset = data,
  outcome_colname = "group",
  method = "rf",         # Choose from rf, svm, glmnet, xgb
  seed = 42
)

# View model summary
summary(fit)
```

```
                   Length Class      Mode
trained_model      21     train      list
test_data          55     data.frame list
performance        17     tbl_df     list
feature_importance  1     -none-     character
```

```r
# Plot variable importance
fit$importance_plot
```

```
NULL
```

24.3 Notes

- mikropl supports additional tuning and export for reproducibility.
- `mikropml()` auto-detects classification vs regression tasks.

Appendix A

Dot Diagram

Appendix B

Modelframework

```r
library(DiagrammeR)
library(DiagrammeRsvg)

mermaid("graph TD
subgraph A
A[Data Preprocessing: Cleaning and Transformation] --> B[Exploratory Analysis]
B --> C[Features selection]
C --> D[Feature Balancing]
D --> |Multi-Model Testing| E[Model Selection]
E --> F[Parameters Tuning]
F --> G[Parameter cross Validation]
end

subgraph B

G--> QC{Model Evaluation}
QC --> H1[ROC: Receiver Operating <br> Characteristic Curve]
QC --> H2[Precision Recall Curve]
end
", height = 1000, width = 800)
```