The following script is for plotting non-chimera%, Log-TCC and LiveCell% on Treatment, Place, Biofilm_Age level.

Tian You

```
#Treatment level
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.stats import mannwhitneyu
from itertools import combinations
import matplotlib.patches as mpatches

# --- 0. Global font settings ---
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = ['Times New Roman'] + plt.rcParams['font.serif']
plt.rcParams['mathtext.fontset'] = 'stix'

# --- 1. Setup and load data ---
file_path = r"C:\Users\ASUS\Desktop\imeta\metadata.xlsx"
output_dir = r"C:\Users\ASUS\Desktop\imeta\plots"

if not os.path.exists(output_dir):
    os.makedirs(output_dir)
    print(f"Directory created: {output_dir}")

color_map = {
    'Untreated': '#fbd5a8',
    'Thermal Disinfection': '#fcb267',
    'Water-Vapor': '#cde8b7',
    'Chemical Treatment': '#86c481',
    'Chemical Treatment + Water-Vapor': '#91bedc'
}

try:
    df = pd.read_excel(file_path)
    print("Excel file loaded successfully.")
except FileNotFoundError:
    print(f"Error: File {file_path} not found. Please check the path.")
    exit()

# --- 2. Prepare plotting ---
```

```
columns_to_plot = {
    'non-chimera%': 'non-chimera%',
    'TTC_500ul': 'TTC_500ul',
    'LiveCells%': 'LiveCells%'
}

# --- New: Define independent axis ranges, tick display, and annotation positions for each plot ---
# 'ticks' controls displaying only the endpoint numbers of the range
# 'lim' is the actual plotting boundary, leaving space for annotations
axis_settings = {
    'non-chimera%': {
        'ticks': [0, 90],
        'lim': (0, 110),
        'annot_start': 95,
        'annot_step': 6
    },
    'TTC_500ul': {
        'ticks': [6, 10],
        'lim': (6, 12),
        'annot_start': 10.5,
        'annot_step': 0.5
    },
    'LiveCells%': {
        'ticks': [40, 95],
        'lim': (40, 110),
        'annot_start': 100,
        'annot_step': 6
    }
}

treatments = sorted(df['Treatment'].unique(), key=lambda x: list(color_map.keys()).index(x))
angles = np.linspace(0, 2 * np.pi, len(treatments), endpoint=False).tolist()
treatment_angle_map = {treatment: angle for treatment, angle in zip(treatments, angles)}

# --- 3. Create figure and subplots ---
fig, axes = plt.subplots(3, 1, figsize=(8, 18), subplot_kw=dict(projection='polar'))
axes = axes.flatten()

# --- 4. Loop to plot each graph ---
for ax, (y_col, title) in zip(axes, columns_to_plot.items()):

    is_log_scale = (y_col == 'TTC_500ul')
    plot_title = f"log10(TCC_500ul + 1)" if is_log_scale else title
    settings = axis_settings[y_col]
```

```
    # Plot boxplot and scatter plot
    for treatment in treatments:
        angle = treatment_angle_map[treatment]
        color = color_map[treatment]

        original_data = df[df['Treatment'] == treatment][y_col].dropna()
        if original_data.empty: continue

        plot_data = np.log10(original_data + 1) if is_log_scale else original_data

        box = ax.boxplot(plot_data, positions=[angle], widths=0.35, patch_artist=True, showfliers=False,
                         medianprops=dict(color='black', linewidth=1.5))
        box['boxes'][0].set_facecolor(color)
        box['boxes'][0].set_edgecolor('black')

        jitter = np.random.normal(0, 0.03, size=len(plot_data))
        ax.scatter(np.array([angle] * len(plot_data)) + jitter, plot_data, color=color, edgecolor='black', s=25, zorder=3)

    # --- Set chart style ---
    ax.set_title(plot_title, size=16, pad=20)
    ax.set_xticklabels([]) # Remove tick labels from angular axis
    ax.set_xticks(angles)

    # --- Modified: Set precise coordinate range and ticks ---
    ax.set_rlim(settings['lim']) # Set plotting boundary
    ax.set_rticks(settings['ticks']) # Set ticks to display only endpoints

    ax.yaxis.grid(True)
    ax.xaxis.grid(True)
    ax.set_rlabel_position(angles[0] - np.pi / 8)

    # Perform statistical tests
    pairs = list(combinations(treatments, 2))
    significant_pairs = []

    for t1, t2 in pairs:
        data1_orig = df[df['Treatment'] == t1][y_col].dropna()
        data2_orig = df[df['Treatment'] == t2][y_col].dropna()

        if not data1_orig.empty and not data2_orig.empty:
            data1 = np.log10(data1_orig + 1) if is_log_scale else data1_orig
```

```
                data2 = np.log10(data2_orig + 1) if is_log_scale else data2_orig
                stat, p_val = mannwhitneyu(data1, data2, alternative='two-sided')
                if p_val < 0.05:
                    significant_pairs.append({'pair': (t1, t2), 'p': p_val})

    # Draw significance markers
    annotation_radius = settings['annot_start']
    radius_step = settings['annot_step']

    for sig in significant_pairs:
        t1, t2 = sig['pair']
        p_val = sig['p']
        angle1, angle2 = treatment_angle_map[t1], treatment_angle_map[t2]

        if abs(angle2 - angle1) > np.pi:
            if angle1 < angle2: angle1 += 2 * np.pi
            else: angle2 += 2 * np.pi

        arc_angles = np.linspace(angle1, angle2, 100)
        arc_radius = np.full_like(arc_angles, annotation_radius)
        ax.plot(arc_angles, arc_radius, color='black', lw=1.2)

        if p_val < 0.001: stars = '***'
        elif p_val < 0.01: stars = '**'
        else: stars = '*'

        mid_angle = (angle1 + angle2) / 2
        text_radius = annotation_radius + radius_step * 0.2
        ax.text(mid_angle, text_radius, stars, ha='center', va='center', fontsize=12,
fontweight='bold')

        annotation_radius += radius_step

# --- 5. Create and place legend ---
legend_patches = [mpatches.Patch(color=color, label=treatment) for treatment, color in
color_map.items()]
fig.legend(handles=legend_patches, loc='lower center', bbox_to_anchor=(0.5, 0.01),
           ncol=len(treatments), fontsize=12, title="Treatment Groups", title_fontsize=14)

# --- 6. Adjust layout and save ---
fig.tight_layout(rect=[0, 0.05, 1, 1])
fig.subplots_adjust(hspace=0.1)

output_filename = os.path.join(output_dir, "plot1.png")
```

```
plt.savefig(output_filename, dpi=300, bbox_inches='tight')

print(f"\nFinal figure saved successfully to: {output_filename}")
#Place level
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.stats import mannwhitneyu
from itertools import combinations
import matplotlib.patches as mpatches

# --- 0. Global font settings ---
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = ['Times New Roman'] + plt.rcParams['font.serif']
plt.rcParams['mathtext.fontset'] = 'stix'

# --- 1. Setup and load data ---
file_path = r"C:\Users\ASUS\Desktop\imeta\metadata.xlsx"
output_dir = r"C:\Users\ASUS\Desktop\imeta\plots"

if not os.path.exists(output_dir):
    os.makedirs(output_dir)
    print(f"Directory created: {output_dir}")

# --- Change: Use new location color mapping ---
color_map = {
    'Lausanne': '#5e5ba8',
    'Preles': '#39b54f',
    'Nidau': '#6ec8bd',
    'Nods': '#d72c82'
}

try:
    df_full = pd.read_excel(file_path)
    print("Excel file loaded successfully.")
except FileNotFoundError:
    print(f"Error: File {file_path} not found. Please check the path.")
    exit()

# --- Change: Filter samples where Treatment is 'Untreated' ---
df = df_full[df_full['Treatment'] == 'Untreated'].copy()
print(f"Filtered 'Untreated' samples, {len(df)} data points remaining.")
```

```
# --- 2. Prepare for plotting ---
columns_to_plot = {
    'non-chimera%': 'non-chimera%',
    'TTC_500ul': 'TTC_500ul',
    'LiveCells%': 'LiveCells%'
}

# Axis range and other settings remain unchanged, adjust if needed
axis_settings = {
    'non-chimera%': {
        'ticks': [0, 90], 'lim': (0, 110), 'annot_start': 95, 'annot_step': 6
    },
    'TTC_500ul': {
        'ticks': [6, 10], 'lim': (6, 12), 'annot_start': 10.5, 'annot_step': 0.5
    },
    'LiveCells%': {
        'ticks': [40, 95], 'lim': (40, 110), 'annot_start': 100, 'annot_step': 6
    }
}

# --- Change: Group and sort by 'Place' column ---
places = sorted(df['Place'].unique(), key=lambda x: list(color_map.keys()).index(x))
angles = np.linspace(0, 2 * np.pi, len(places), endpoint=False).tolist()
place_angle_map = {place: angle for place, angle in zip(places, angles)}

# --- 3. Create figure and subplots ---
fig, axes = plt.subplots(3, 1, figsize=(8, 18), subplot_kw=dict(projection='polar'))
axes = axes.flatten()

# --- 4. Loop to plot each graph ---
for ax, (y_col, title) in zip(axes, columns_to_plot.items()):

    is_log_scale = (y_col == 'TTC_500ul')
    plot_title = f"log10(TCC_500ul + 1)" if is_log_scale else title

    settings = axis_settings[y_col]

    # --- Change: Loop through each 'place' ---
    for place in places:
        angle = place_angle_map[place]
        color = color_map[place]

        # --- Change: Filter data by 'Place' column ---
```

```
            original_data = df[df['Place'] == place][y_col].dropna()
            if original_data.empty: continue

            plot_data = np.log10(original_data + 1) if is_log_scale else original_data

            box = ax.boxplot(plot_data, positions=[angle], widths=0.45, patch_artist=True, showfliers=False,
                             medianprops=dict(color='black', linewidth=1.5))
            box['boxes'][0].set_facecolor(color)
            box['boxes'][0].set_edgecolor('black')

            jitter = np.random.normal(0, 0.04, size=len(plot_data))
            ax.scatter(np.array([angle] * len(plot_data)) + jitter, plot_data, color=color, edgecolor='black', s=25, zorder=3)

    # --- Set chart style ---
    ax.set_title(plot_title, size=16, pad=20)
    ax.set_xticklabels([])
    ax.set_xticks(angles)

    ax.set_rlim(settings['lim'])
    ax.set_rticks(settings['ticks'])

    ax.yaxis.grid(True)
    ax.xaxis.grid(True)
    ax.set_rlabel_position(angles[0] - np.pi / 8)

    # --- Change: Perform statistical tests on 'place' ---
    pairs = list(combinations(places, 2))
    significant_pairs = []

    for p1, p2 in pairs:
        data1_orig = df[df['Place'] == p1][y_col].dropna()
        data2_orig = df[df['Place'] == p2][y_col].dropna()

        if not data1_orig.empty and not data2_orig.empty:
            data1 = np.log10(data1_orig + 1) if is_log_scale else data1_orig
            data2 = np.log10(data2_orig + 1) if is_log_scale else data2_orig
            stat, p_val = mannwhitneyu(data1, data2, alternative='two-sided')
            if p_val < 0.05:
                significant_pairs.append({'pair': (p1, p2), 'p': p_val})

    # Draw significance markers
    annotation_radius = settings['annot_start']
```

```
    radius_step = settings['annot_step']

    for sig in significant_pairs:
        p1, p2 = sig['pair']
        p_val = sig['p']
        angle1, angle2 = place_angle_map[p1], place_angle_map[p2]

        if abs(angle2 - angle1) > np.pi:
            if angle1 < angle2: angle1 += 2 * np.pi
            else: angle2 += 2 * np.pi

        arc_angles = np.linspace(angle1, angle2, 100)
        arc_radius = np.full_like(arc_angles, annotation_radius)
        ax.plot(arc_angles, arc_radius, color='black', lw=1.2)

        if p_val < 0.001: stars = '***'
        elif p_val < 0.01: stars = '**'
        else: stars = '*'

        mid_angle = (angle1 + angle2) / 2
        text_radius = annotation_radius + radius_step * 0.2
        ax.text(mid_angle, text_radius, stars, ha='center', va='center', fontsize=12,
fontweight='bold')

        annotation_radius += radius_step

# --- 5. Create and place legend ---
# --- Change: Create legend based on new places and colors ---
legend_patches = [mpatches.Patch(color=color, label=place) for place, color in color_map.items()]
fig.legend(handles=legend_patches, loc='lower center', bbox_to_anchor=(0.5, 0.01),
           ncol=len(places), fontsize=12, title="Location (Place)", title_fontsize=14)

# --- 6. Adjust layout and save ---
fig.tight_layout(rect=[0, 0.05, 1, 1])
fig.subplots_adjust(hspace=0.1)

# --- Change: Save with new filename ---
output_filename = os.path.join(output_dir, "plot2.png")
plt.savefig(output_filename, dpi=300, bbox_inches='tight')

print(f"\nLocation-based polar plots saved to: {output_filename}")

#Biofilm Age level
```

```
mport pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.stats import mannwhitneyu
from itertools import combinations
import matplotlib.patches as mpatches

# --- 0. Global font settings ---
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = ['Times New Roman'] + plt.rcParams['font.serif']
plt.rcParams['mathtext.fontset'] = 'stix'

# --- 1. Setup and load data ---
file_path = r"C:\Users\ASUS\Desktop\imeta\metadata.xlsx"
output_dir = r"C:\Users\ASUS\Desktop\imeta\plots"

if not os.path.exists(output_dir):
    os.makedirs(output_dir)
    print(f"Directory created: {output_dir}")

# --- Change: Use new Biofilm_Age color mapping ---
# Note: Keys are strings as Biofilm_Age is treated as categorical data
color_map = {
    '5': '#fcad19',
    '32': '#ef8b19',
    '36': '#df8219',
    '52': '#ba6c19',
    '104': '#945519'
}

try:
    df_full = pd.read_excel(file_path)
    print("Excel file loaded successfully.")
except FileNotFoundError:
    print(f"Error: File {file_path} not found. Please check the path.")
    exit()

# --- Key changes: Filter samples and process data types ---
# 1. Filter samples where Treatment is 'Untreated'
df = df_full[df_full['Treatment'] == 'Untreated'].copy()
# 2. Convert Biofilm_Age column to string for categorical processing
df['Biofilm_Age'] = df['Biofilm_Age'].astype(str)
print(f"Filtered 'Untreated' samples and processed Biofilm_Age, {len(df)} data points remaining.")
```

```
# --- 2. Prepare for plotting ---
columns_to_plot = {
    'non-chimera%': 'non-chimera%',
    'TTC_500ul': 'TTC_500ul',
    'LiveCells%': 'LiveCells%'
}

axis_settings = {
    'non-chimera%': {
        'ticks': [0, 90], 'lim': (0, 110), 'annot_start': 95, 'annot_step': 6
    },
    'TTC_500ul': {
        'ticks': [6, 10], 'lim': (6, 12), 'annot_start': 10.5, 'annot_step': 0.5
    },
    'LiveCells%': {
        'ticks': [40, 95], 'lim': (40, 110), 'annot_start': 100, 'annot_step': 6
    }
}

# --- Change: Group and sort by 'Biofilm_Age' column ---
biofilm_ages = sorted(df['Biofilm_Age'].unique(), key=lambda x: list(color_map.keys()).index(x))
angles = np.linspace(0, 2 * np.pi, len(biofilm_ages), endpoint=False).tolist()
age_angle_map = {age: angle for age, angle in zip(biofilm_ages, angles)}

# --- 3. Create figure and subplots ---
fig, axes = plt.subplots(3, 1, figsize=(8, 18), subplot_kw=dict(projection='polar'))
axes = axes.flatten()

# --- 4. Loop to plot each graph ---
for ax, (y_col, title) in zip(axes, columns_to_plot.items()):

    is_log_scale = (y_col == 'TTC_500ul')
    plot_title = f"log10(TCC_500ul + 1)" if is_log_scale else title
    settings = axis_settings[y_col]

    # --- Change: Loop through each 'biofilm_age' ---
    for age in biofilm_ages:
        angle = age_angle_map[age]
        color = color_map[age]

        # --- Change: Filter data by 'Biofilm_Age' column ---
        original_data = df[df['Biofilm_Age'] == age][y_col].dropna()
```

```
            if original_data.empty: continue

            plot_data = np.log10(original_data + 1) if is_log_scale else original_data

            box = ax.boxplot(plot_data, positions=[angle], widths=0.35, patch_artist=True, showfliers=False,
                             medianprops=dict(color='black', linewidth=1.5))
            box['boxes'][0].set_facecolor(color)
            box['boxes'][0].set_edgecolor('black')

            jitter = np.random.normal(0, 0.03, size=len(plot_data))
            ax.scatter(np.array([angle] * len(plot_data)) + jitter, plot_data, color=color, edgecolor='black', s=25, zorder=3)

    # --- Set chart style ---
    ax.set_title(plot_title, size=16, pad=20)
    ax.set_xticklabels([])
    ax.set_xticks(angles)

    ax.set_rlim(settings['lim'])
    ax.set_rticks(settings['ticks'])

    ax.yaxis.grid(True)
    ax.xaxis.grid(True)
    ax.set_rlabel_position(angles[0] - np.pi / 8)

    # --- Change: Perform statistical tests on 'biofilm_age' ---
    pairs = list(combinations(biofilm_ages, 2))
    significant_pairs = []

    for age1, age2 in pairs:
        data1_orig = df[df['Biofilm_Age'] == age1][y_col].dropna()
        data2_orig = df[df['Biofilm_Age'] == age2][y_col].dropna()

        if not data1_orig.empty and not data2_orig.empty:
            data1 = np.log10(data1_orig + 1) if is_log_scale else data1_orig
            data2 = np.log10(data2_orig + 1) if is_log_scale else data2_orig
            stat, p_val = mannwhitneyu(data1, data2, alternative='two-sided')
            if p_val < 0.05:
                significant_pairs.append({'pair': (age1, age2), 'p': p_val})

    # Draw significance markers
    annotation_radius = settings['annot_start']
    radius_step = settings['annot_step']
```

```
    for sig in significant_pairs:
        age1, age2 = sig['pair']
        p_val = sig['p']
        angle1, angle2 = age_angle_map[age1], age_angle_map[age2]

        if abs(angle2 - angle1) > np.pi:
            if angle1 < angle2: angle1 += 2 * np.pi
            else: angle2 += 2 * np.pi

        arc_angles = np.linspace(angle1, angle2, 100)
        arc_radius = np.full_like(arc_angles, annotation_radius)
        ax.plot(arc_angles, arc_radius, color='black', lw=1.2)

        if p_val < 0.001: stars = '***'
        elif p_val < 0.01: stars = '**'
        else: stars = '*'

        mid_angle = (angle1 + angle2) / 2
        text_radius = annotation_radius + radius_step * 0.2
        ax.text(mid_angle, text_radius, stars, ha='center', va='center', fontsize=12,
fontweight='bold')

        annotation_radius += radius_step

# --- 5. Create and place legend ---
# --- Change: Create legend based on new biofilm ages and colors ---
legend_patches = [mpatches.Patch(color=color, label=age) for age, color in color_map.items()]
fig.legend(handles=legend_patches, loc='lower center', bbox_to_anchor=(0.5, 0.01),
           ncol=len(biofilm_ages), fontsize=12, title="Biofilm Age (weeks)", title_fontsize=14)

# --- 6. Adjust layout and save ---
fig.tight_layout(rect=[0, 0.05, 1, 1])
fig.subplots_adjust(hspace=0.1)

# --- Change: Save with new filename ---
output_filename = os.path.join(output_dir, "plot3.png")
plt.savefig(output_filename, dpi=300, bbox_inches='tight')

print(f"\nPolar plots grouped by Biofilm Age saved to: {output_filename}")
```

The following script is for Linear regression analysis of LiveCell% vs non-chimera%

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import os

# 1. Set file path and column names
file_path = "C:/Users/ASUS/Desktop/imeta/metadata.xlsx"

#Set output directory path
output_dir = "C:/Users/ASUS/Desktop/imeta/plots"

# Ensure the column names here exactly match those in your Excel file
x_col = 'LiveCells%'
y_col = 'non-chimera%'

# 2. Read Excel file
try:
    df = pd.read_excel(file_path)

    # Ensure the required columns exist in the DataFrame
    if x_col not in df.columns or y_col not in df.columns:
        raise ValueError(f"Error: Please ensure '{x_col}' and '{y_col}' are correct column names
in your Excel file.")

    # 3. Perform linear regression analysis
    slope, intercept, r_value, p_value, std_err = stats.linregress(df[x_col], df[y_col])
    r_squared = r_value**2

    print(f"Linear Regression Results:")
    print(f"R-squared: {r_squared:.4f}")
    print(f"P-value: {p_value:.4f}")
    print(f"Regression Equation: y = {slope:.4f}x + {intercept:.4f}")

    # 4. Create plot
    plt.figure(figsize=(10, 6))
    sns.set_theme(style="whitegrid")

    ax = sns.regplot(x=x_col, y=y_col, data=df,
```

```
                    line_kws={'color': 'red', 'label': f'y = {slope:.2f}x + {intercept:.2f}'},
                    scatter_kws={'alpha': 0.6})

    stats_text = f'$R^2 = {r_squared:.4f}$\n$p = {p_value:.4f}$'
    ax.text(0.05, 0.95, stats_text, transform=ax.transAxes, fontsize=12,
            verticalalignment='top', bbox=dict(boxstyle='round,pad=0.5', fc='wheat',
alpha=0.5))

    plt.title(f'{x_col} vs {y_col}', fontsize=16)
    plt.xlabel(x_col, fontsize=12)
    plt.ylabel(y_col, fontsize=12)
    plt.legend()

    # 5. ---> Modified: Save chart to specified directory <---

    # Create the output directory if it doesn't exist
    os.makedirs(output_dir, exist_ok=True)

    # Construct the full file save path
    # Replace '%' with 'pct' to avoid special characters in the filename
    sanitized_x_col = x_col.replace('%', 'pct')
    sanitized_y_col = y_col.replace('%', 'pct')
    output_filename = f"plot4.png"
    full_output_path = os.path.join(output_dir, output_filename)

    # Save the chart
    plt.savefig(full_output_path, dpi=300, bbox_inches='tight')

    print(f"\nChart saved successfully to: {full_output_path}")

    # 6. Display the chart
    plt.show()

except FileNotFoundError:
    print(f"Error: File not found. Please check the path '{file_path}'.")
except Exception as e:
    print(f"An error occurred: {e}")
```

The following script is for absolute abundance analysis of top 30 genera across samples.

```
# ==================== 1. Load Required R Packages ====================
library(phyloseq)
library(readxl)
library(readr)
library(dplyr)
library(tibble)

# ==================== 2. Set File Paths ====================
metadata_path <- "C:/Users/ASUS/Desktop/imeta/metadata.xlsx"
asv_table_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_table.csv"
taxonomy_path <- "C:/Users/ASUS/Desktop/imeta/result/taxonomy_table.xlsx"

# ==================== 3. Read Data Files ====================

# 3.1 Read sample metadata
cat("Reading sample metadata...\n")
metadata <- read_excel(metadata_path, sheet = 1)
print(paste("Metadata contains", nrow(metadata), "samples"))
print(paste("Metadata columns:", paste(colnames(metadata), collapse = ", ")))

# 3.2 Read ASV abundance table
cat("\nReading ASV abundance table...\n")
asv_table <- read_csv(asv_table_path)
print(paste("ASV table contains", nrow(asv_table), "ASVs and", ncol(asv_table)-1, "samples"))

# 3.3 Read taxonomic information
cat("\nReading taxonomic information...\n")
taxonomy <- read_excel(taxonomy_path, sheet = 1)
print(paste("Taxonomy table contains", nrow(taxonomy), "ASVs"))
print(paste("Taxonomic ranks:", paste(colnames(taxonomy)[-1], collapse = ", ")))

# ==================== 4. Data Preprocessing ====================

# 4.1 Process sample metadata
# Set sample_id as row names
sample_data_df <- metadata %>%
  column_to_rownames("sample_id")

# 4.2 Process ASV abundance table
```

```
# Set ASV_id as row names, convert to matrix
otu_table_df <- asv_table %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check and ensure consistent sample ordering
cat("\nChecking sample ID consistency...\n")
metadata_samples <- rownames(sample_data_df)
asv_samples <- colnames(otu_table_df)

if (length(setdiff(metadata_samples, asv_samples)) == 0 &&
      length(setdiff(asv_samples, metadata_samples)) == 0) {
  cat("✓ Sample IDs match completely!\n")
} else {
  cat("⚠ Sample IDs do not match, need to check!\n")
}

# 4.3 Process taxonomic information
# Set ASV_id as row names
tax_table_df <- taxonomy %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check ASV ID consistency
asv_ids_otu <- rownames(otu_table_df)
asv_ids_tax <- rownames(tax_table_df)

if (length(setdiff(asv_ids_otu, asv_ids_tax)) == 0 &&
      length(setdiff(asv_ids_tax, asv_ids_otu)) == 0) {
  cat("✓ ASV IDs match completely!\n")
} else {
  cat("⚠ ASV IDs do not match, need to check!\n")
}

# ==================== 5. Create Phyloseq Object Components ====================

# 5.1 Create OTU table object
cat("\nCreating phyloseq components...\n")
OTU <- otu_table(otu_table_df, taxa_are_rows = TRUE)

# 5.2 Create sample data object
SAMP <- sample_data(sample_data_df)

# 5.3 Create taxonomy table object
```

```
TAX <- tax_table(tax_table_df)

# ==================== 6. Construct Phyloseq Object ====================
cat("Constructing phyloseq object...\n")
physeq <- phyloseq(OTU, SAMP, TAX)

# ==================== 7. Validation and Summary ====================
cat("\n=== Phyloseq object construction complete! ===\n")
print(physeq)

# Output basic statistical information
cat("\n=== Basic Statistical Information ===\n")
cat("Number of samples:", nsamples(physeq), "\n")
cat("Number of ASVs:", ntaxa(physeq), "\n")
cat("Taxonomic ranks:", paste(rank_names(physeq), collapse = ", "), "\n")
cat("Sample variables:", paste(sample_variables(physeq), collapse = ", "), "\n")

# Check data integrity
cat("\n=== Data Quality Check ===\n")
# Check for empty samples
empty_samples <- sample_sums(physeq) == 0
if (any(empty_samples)) {
  cat("⚠ Found", sum(empty_samples), "empty samples\n")
} else {
  cat("✓ All samples contain sequences\n")
}

# Check for empty ASVs
empty_taxa <- taxa_sums(physeq) == 0
if (any(empty_taxa)) {
  cat("⚠ Found", sum(empty_taxa), "empty ASVs\n")
} else {
  cat("✓ All ASVs contain sequences\n")
}

# Display sample sequencing depth statistics
cat("\n=== Sequencing Depth Statistics ===\n")
seq_depth <- sample_sums(physeq)
cat("Minimum sequencing depth:", min(seq_depth), "\n")
cat("Maximum sequencing depth:", max(seq_depth), "\n")
cat("Average sequencing depth:", round(mean(seq_depth), 0), "\n")
cat("Median sequencing depth:", round(median(seq_depth), 0), "\n")

# ==================== 8. Optional: Save Phyloseq Object ====================
```

```
# Uncomment the following line to save the phyloseq object
# saveRDS(physeq, "phyloseq_object.rds")
# cat("\nPhyloseq object saved as phyloseq_object.rds\n")

# ==================== 9. Optional: Basic Data Cleaning ====================
# Basic data filtering if needed
cat("\n=== Optional Data Cleaning Steps ===\n")
cat("Original data: ", ntaxa(physeq), "ASVs,", nsamples(physeq), "samples\n")

# Remove empty samples and empty ASVs if any
physeq_cleaned <- prune_samples(sample_sums(physeq) > 0, physeq)
physeq_cleaned <- prune_taxa(taxa_sums(physeq_cleaned) > 0, physeq_cleaned)

cat("Cleaned data: ", ntaxa(physeq_cleaned), "ASVs,", nsamples(physeq_cleaned), "samples\n")

# Assign cleaned object to main variable
physeq <- physeq_cleaned

cat("\n🎉 Phyloseq object construction complete! You can start subsequent analyses.\n")
cat("Object name: physeq\n")
cat("Use print(physeq) to view object information\n")

# ==================== 10. Add Phylogenetic Tree to Phyloseq Object ====================
library(ape)

# Set tree file paths
tree_path <- "C:/Users/ASUS/Desktop/imeta/result/sequences.aligned.fasta.treefile"
hash_mapping_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_hash_mapping.csv"

# 10.1 Read and process hash mapping
cat("\nReading ASV hash mapping...\n")
hash_mapping <- read_csv(hash_mapping_path, col_names = c("ASV_id", "Hash"))
hash_mapping$Hash <- trimws(hash_mapping$Hash)    # Clean whitespaces
hash_vec <- setNames(hash_mapping$ASV_id, hash_mapping$Hash)

# 10.2 Read phylogenetic tree
cat("Reading phylogenetic tree...\n")
tree <- read.tree(tree_path)

# 10.3 Check tree-hash correspondence
tree_hashes <- tree$tip.label
mapped_asvs <- hash_vec[tree_hashes]
unmapped_count <- sum(is.na(mapped_asvs))
```

```
if (unmapped_count > 0) {
  cat("⚠ Warning:", unmapped_count, "tree tips not found in hash mapping\n")
  # Remove unmapped tips
  tree <- drop.tip(tree, tree_hashes[is.na(mapped_asvs)])
  mapped_asvs <- na.omit(mapped_asvs)
}

# 10.4 Update tip labels with ASV IDs
tree$tip.label <- as.character(mapped_asvs[tree$tip.label])

# 10.5 Align tree and phyloseq object
# Get overlapping ASVs
common_asvs <- intersect(tree$tip.label, taxa_names(physeq))
physeq_sub <- prune_taxa(common_asvs, physeq)
tree <- keep.tip(tree, common_asvs)

cat("→ Retained", length(common_asvs), "ASVs common to tree and phyloseq\n")

# 10.6 Add tree to phyloseq object
phy_tree(physeq_sub) <- tree

# 10.7 Validate integration
cat("\n=== Tree Integration Validation ===\n")
cat("Tree contains", Ntip(tree), "tips matching ASVs\n")

if (all(taxa_names(physeq_sub) %in% tree$tip.label)) {
  cat("✓ All phyloseq taxa present in tree\n")
} else {
  cat("⚠ Missing taxa: Not all ASVs are in the tree\n")
}

# 10.8 Update phyloseq object
physeq <- phyloseq(otu_table(physeq_sub),
                   sample_data(physeq_sub),
                   tax_table(physeq_sub),
                   phy_tree(physeq_sub))

cat("\n=== Updated Phyloseq Object Summary ===\n")
print(physeq)

# 10.9 Optional: Save updated object
# saveRDS(physeq, "phyloseq_with_tree.rds")
# cat("Saved phyloseq object with tree as 'phyloseq_with_tree.rds'\n")
```

```
cat("\n ✅ Phylogenetic tree successfully integrated!\n")

# ==================== 1. Load required R packages ====================
# ggplot2 for advanced plotting, ggthemes for nice themes
library(ggplot2)
library(ggthemes)
library(dplyr)            # data manipulation
library(phyloseq)         # microbiome data analysis

cat("✓ Required R packages loaded.\n")

# ==================== 2. Set file paths ====================
# Absolute-abundance ASV table
abs_abundance_path <-
"C:/Users/ASUS/Desktop/imeta/result/absolute_abundance_asv_table.csv"
# Folder for output plots
plot_output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/absolute_abundance"

# Create output folder if it does not exist
dir.create(plot_output_dir, showWarnings = FALSE, recursive = TRUE)

cat("✓ File paths configured; output folder ready.\n")

# ==================== 3. Read and update phyloseq object with absolute-abundance table ====================

# 3.1 Read absolute-abundance table
cat("Reading absolute-abundance table...\n")
abs_asv_table <- readr::read_csv(abs_abundance_path)

# 3.2 Pre-process the new abundance table
# Set ASV_id as row names and convert to matrix
abs_otu_matrix <- abs_asv_table %>%
  tibble::column_to_rownames("ASV_id") %>%
  as.matrix()

# 3.3 Align new abundance matrix with existing physeq object
# Find ASVs and samples common to both
common_taxa <- intersect(rownames(abs_otu_matrix), taxa_names(physeq))
common_samples <- intersect(colnames(abs_otu_matrix), sample_names(physeq))

# Subset physeq and matrix to the intersection
physeq_aligned <- prune_taxa(common_taxa, physeq)
physeq_aligned <- prune_samples(common_samples, physeq_aligned)
```

```
abs_otu_matrix_aligned <- abs_otu_matrix[taxa_names(physeq_aligned),
                                         sample_names(physeq_aligned)]

# 3.4 Build new phyloseq object with absolute abundances
physeq_abs <- phyloseq(
  otu_table(abs_otu_matrix_aligned, taxa_are_rows = TRUE),
  sample_data(physeq_aligned),
  tax_table(physeq_aligned),
  phy_tree(physeq_aligned)
)

cat("✓ New phyloseq object for absolute-abundance analysis created.\n")
cat("   Contains", ntaxa(physeq_abs), "ASVs and", nsamples(physeq_abs), "samples.\n")


# ==================== 4. Data aggregation and filtering ====================

# 4.1 Aggregate ASVs to genus level
cat("Aggregating ASVs to genus level...\n")
physeq_genus <- tax_glom(physeq_abs, taxrank = "Genus", NArm = FALSE)

# 4.2 Remove unwanted taxa
# Drop genera labeled NA or "uncultured"
cat("Filtering out NA and uncultured genera...\n")
physeq_genus_filtered <- subset_taxa(
  physeq_genus,
  !is.na(Genus) & !grepl("uncultured", Genus, ignore.case = TRUE)
)

# 4.3 Keep top-30 most abundant genera
cat("Selecting Top 30 most abundant genera...\n")
top_genera_names <- names(sort(taxa_sums(physeq_genus_filtered), decreasing = TRUE)[1:30])

physeq_top30_genus <- prune_taxa(top_genera_names, physeq_genus_filtered)
cat("✓ Top 30 genera selected for plotting.\n")

# ==================== 5. Prepare plotting data ====================
# Convert phyloseq object to long-format data frame
cat("Converting data to ggplot2-friendly format...\n")
plot_data <- psmelt(physeq_top30_genus)  # NOTE: typo fixed here (was physeq_top_30_genus)

# ==================== 6. Loop to create and save stacked bar charts ====================
cat("🚀 Starting plot generation...\n")
```

```
# Metadata columns to facet by
grouping_variables <- c("Treatment", "Biofilm_Status", "Place", "Legionella")

for (group_var in grouping_variables) {

  cat("    -> Plotting faceted by '", group_var, "'...\n")

  # Build ggplot
  p <- ggplot(plot_data, aes(x = Sample, y = Abundance, fill = Genus)) +
    geom_bar(stat = "identity", position = "stack") +

    # Facet by the current grouping variable
    facet_grid(cols = vars(.data[[group_var]]), scales = "free_x", space = "free_x") +

    # Labels
    labs(
      title = "Top 30 Genera Absolute Abundance",
      subtitle = paste("Faceted by", group_var),
      x = "Sample",
      y = "Absolute Abundance (copy number)"
    ) +

    # Theme
    theme_bw(base_size = 12) +
    theme(
      axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5, size = 8),
      plot.title       = element_text(hjust = 0.5),
      plot.subtitle = element_text(hjust = 0.5),
      panel.spacing.x = unit(0, "lines"),
      legend.text = element_text(size = 8),
      legend.key.size = unit(0.4, "cm")
    ) +

    # Colour scale
    scale_fill_viridis_d(option = "turbo")

  # Output file
  output_filename <- file.path(plot_output_dir,
                               paste0("absolute_abundance_by_", group_var, ".png"))

  # Save plot
  ggsave(
    filename = output_filename,
```

```
        plot = p,
        width = 16, height = 9, dpi = 300
    )

    cat("        ✓ Plot saved to:", output_filename, "\n")
}

cat("\n 🎉 All tasks complete! Plots saved to the specified folder.\n")
```

The following script is for alpha-diversity analysis

```
# ==================== 1. Load Required R Packages ====================
library(phyloseq)
library(readxl)
library(readr)
library(dplyr)
library(tibble)

# ==================== 2. Set File Paths ====================
metadata_path <- "C:/Users/ASUS/Desktop/imeta/metadata.xlsx"
asv_table_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_table.csv"
taxonomy_path <- "C:/Users/ASUS/Desktop/imeta/result/taxonomy_table.xlsx"

# ==================== 3. Read Data Files ====================

# 3.1 Read sample metadata
cat("Reading sample metadata...\n")
metadata <- read_excel(metadata_path, sheet = 1)
print(paste("Metadata contains", nrow(metadata), "samples"))
print(paste("Metadata columns:", paste(colnames(metadata), collapse = ", ")))

# 3.2 Read ASV abundance table
cat("\nReading ASV abundance table...\n")
asv_table <- read_csv(asv_table_path)
print(paste("ASV table contains", nrow(asv_table), "ASVs and", ncol(asv_table)-1, "samples"))

# 3.3 Read taxonomic information
cat("\nReading taxonomic information...\n")
taxonomy <- read_excel(taxonomy_path, sheet = 1)
print(paste("Taxonomy table contains", nrow(taxonomy), "ASVs"))
print(paste("Taxonomic ranks:", paste(colnames(taxonomy)[-1], collapse = ", ")))

# ==================== 4. Data Preprocessing ====================

# 4.1 Process sample metadata
# Set sample_id as row names
sample_data_df <- metadata %>%
  column_to_rownames("sample_id")
```

```
# 4.2 Process ASV abundance table
# Set ASV_id as row names, convert to matrix
otu_table_df <- asv_table %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check and ensure consistent sample ordering
cat("\nChecking sample ID consistency...\n")
metadata_samples <- rownames(sample_data_df)
asv_samples <- colnames(otu_table_df)

if (length(setdiff(metadata_samples, asv_samples)) == 0 &&
      length(setdiff(asv_samples, metadata_samples)) == 0) {
  cat("✓ Sample IDs match completely!\n")
} else {
  cat("⚠ Sample IDs do not match, need to check!\n")
}

# 4.3 Process taxonomic information
# Set ASV_id as row names
tax_table_df <- taxonomy %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check ASV ID consistency
asv_ids_otu <- rownames(otu_table_df)
asv_ids_tax <- rownames(tax_table_df)

if (length(setdiff(asv_ids_otu, asv_ids_tax)) == 0 &&
      length(setdiff(asv_ids_tax, asv_ids_otu)) == 0) {
  cat("✓ ASV IDs match completely!\n")
} else {
  cat("⚠ ASV IDs do not match, need to check!\n")
}

# ==================== 5. Create Phyloseq Object Components ====================

# 5.1 Create OTU table object
cat("\nCreating phyloseq components...\n")
OTU <- otu_table(otu_table_df, taxa_are_rows = TRUE)

# 5.2 Create sample data object
SAMP <- sample_data(sample_data_df)
```

```
# 5.3 Create taxonomy table object
TAX <- tax_table(tax_table_df)

# ==================== 6. Construct Phyloseq Object ====================
cat("Constructing phyloseq object...\n")
physeq <- phyloseq(OTU, SAMP, TAX)

# ==================== 7. Validation and Summary ====================
cat("\n=== Phyloseq object construction complete! ===\n")
print(physeq)

# Output basic statistical information
cat("\n=== Basic Statistical Information ===\n")
cat("Number of samples:", nsamples(physeq), "\n")
cat("Number of ASVs:", ntaxa(physeq), "\n")
cat("Taxonomic ranks:", paste(rank_names(physeq), collapse = ", "), "\n")
cat("Sample variables:", paste(sample_variables(physeq), collapse = ", "), "\n")

# Check data integrity
cat("\n=== Data Quality Check ===\n")
# Check for empty samples
empty_samples <- sample_sums(physeq) == 0
if (any(empty_samples)) {
  cat("⚠ Found", sum(empty_samples), "empty samples\n")
} else {
  cat("✓ All samples contain sequences\n")
}

# Check for empty ASVs
empty_taxa <- taxa_sums(physeq) == 0
if (any(empty_taxa)) {
  cat("⚠ Found", sum(empty_taxa), "empty ASVs\n")
} else {
  cat("✓ All ASVs contain sequences\n")
}

# Display sample sequencing depth statistics
cat("\n=== Sequencing Depth Statistics ===\n")
seq_depth <- sample_sums(physeq)
cat("Minimum sequencing depth:", min(seq_depth), "\n")
cat("Maximum sequencing depth:", max(seq_depth), "\n")
cat("Average sequencing depth:", round(mean(seq_depth), 0), "\n")
cat("Median sequencing depth:", round(median(seq_depth), 0), "\n")
```

```
# ==================== 8. Optional: Save Phyloseq Object ====================
# Uncomment the following line to save the phyloseq object
# saveRDS(physeq, "phyloseq_object.rds")
# cat("\nPhyloseq object saved as phyloseq_object.rds\n")

# ==================== 9. Optional: Basic Data Cleaning ====================
# Basic data filtering if needed
cat("\n=== Optional Data Cleaning Steps ===\n")
cat("Original data: ", ntaxa(physeq), "ASVs,", nsamples(physeq), "samples\n")

# Remove empty samples and empty ASVs if any
physeq_cleaned <- prune_samples(sample_sums(physeq) > 0, physeq)
physeq_cleaned <- prune_taxa(taxa_sums(physeq_cleaned) > 0, physeq_cleaned)

cat("Cleaned data: ", ntaxa(physeq_cleaned), "ASVs,", nsamples(physeq_cleaned), "samples\n")

# Assign cleaned object to main variable
physeq <- physeq_cleaned

cat("\n🎉 Phyloseq object construction complete! You can start subsequent analyses.\n")
cat("Object name: physeq\n")
cat("Use print(physeq) to view object information\n")

# ==================== Alpha Diversity Analysis with Phylogenetic Tree ====================

# Load required packages
library(phyloseq)
library(ggplot2)
library(dplyr)
library(readr)
library(ape)
library(picante)
library(gridExtra)
library(RColorBrewer)
library(phangorn)    # For midpoint rooting
library(ggpubr)      # For statistical comparisons
library(rstatix)     # For statistical tests

# ==================== 1. Read Phylogenetic Tree and Mapping Files ====================

# File paths
tree_file_path <- "C:/Users/ASUS/Desktop/imeta/result/sequences.aligned.fasta.treefile"
asv_mapping_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_hash_mapping.csv"
```

```
# Read the phylogenetic tree
cat("Reading phylogenetic tree...\n")
tree <- read.tree(tree_file_path)
print(paste("Tree contains", length(tree$tip.label), "tips"))

# Read ASV-Hash mapping table
cat("Reading ASV-Hash mapping table...\n")
asv_mapping <- read_csv(asv_mapping_path)
colnames(asv_mapping) <- c("ASV_ID", "Hash")
print(paste("Mapping table contains", nrow(asv_mapping), "entries"))

# ==================== 2. Process Tree Tip Labels ====================

# Check if tree tip labels match hash values in mapping table
cat("\nChecking tree-mapping consistency...\n")
tree_tips <- tree$tip.label
mapping_hashes <- asv_mapping$Hash

# Find matches
matched_hashes <- intersect(tree_tips, mapping_hashes)
cat("Matched hash values:", length(matched_hashes), "\n")

if (length(matched_hashes) < length(tree_tips) * 0.8) {
  cat("⚠ Warning: Less than 80% of tree tips match mapping table\n")
} else {
  cat("✓ Good match between tree and mapping table\n")
}

# ==================== 3. Update Phyloseq Object with Tree ====================

# Filter mapping table to include only ASVs present in phyloseq object
physeq_asvs <- taxa_names(physeq)
filtered_mapping <- asv_mapping[asv_mapping$ASV_ID %in% physeq_asvs, ]

cat("\nFiltered mapping table contains", nrow(filtered_mapping), "ASVs present in phyloseq object\n")

# Create a named vector for tip label replacement
tip_replacement <- setNames(filtered_mapping$ASV_ID, filtered_mapping$Hash)

# Update tree tip labels from Hash to ASV_ID
tree_updated <- tree
matched_tips <- tree_updated$tip.label %in% names(tip_replacement)
```

```
tree_updated$tip.label[matched_tips] <-
tip_replacement[tree_updated$tip.label[matched_tips]]

# Remove tree tips that don't have corresponding ASVs in phyloseq
asvs_to_keep <- intersect(tree_updated$tip.label, taxa_names(physeq))
tree_pruned <- keep.tip(tree_updated, asvs_to_keep)

cat("Pruned tree contains", length(tree_pruned$tip.label), "tips\n")

# Add tree to phyloseq object
physeq_tree <- merge_phyloseq(physeq, phy_tree(tree_pruned))

cat("✓ Phyloseq object updated with phylogenetic tree\n")
print(physeq_tree)

# ==================== 4. Calculate Alpha Diversity Indices ====================

cat("\nCalculating alpha diversity indices...\n")

# Calculate alpha diversity indices
alpha_div <- estimate_richness(physeq_tree, measures = c("Observed", "Shannon", "Simpson"))

# Calculate Faith's Phylogenetic Diversity
# First, create a community matrix (samples x species)
comm_matrix <- t(as(otu_table(physeq_tree), "matrix"))

# Check if tree is rooted and root it if necessary
tree_for_pd <- phy_tree(physeq_tree)
cat("Checking tree rooting status...\n")
cat("Is tree rooted?", is.rooted(tree_for_pd), "\n")

if (!is.rooted(tree_for_pd)) {
  cat("Tree is unrooted. Attempting to root the tree...\n")

  # Method 1: Try midpoint rooting (requires phangorn package)
  tryCatch({
    if (requireNamespace("phangorn", quietly = TRUE)) {
      tree_for_pd <- phangorn::midpoint.root(tree_for_pd)
      cat("✓ Tree rooted using midpoint method\n")
    } else {
      # Method 2: Simple rooting using ape package
      # Root at the first tip (arbitrary but functional)
      tree_for_pd <- root(tree_for_pd, outgroup = tree_for_pd$tip.label[1], resolve.root =
TRUE)
```

```
            cat("✓ Tree rooted using first tip as outgroup\n")
        }
    }, error = function(e) {
        cat("⚠ Could not root tree. Will calculate Faith's PD without including root.\n")
        tree_for_pd <<- phy_tree(physeq_tree)    # Use original tree
    })
}

# Calculate Faith's PD
tryCatch({
    if (is.rooted(tree_for_pd)) {
        faith_pd <- pd(comm_matrix, tree_for_pd, include.root = TRUE)
        cat("✓ Faith's PD calculated with root included\n")
    } else {
        faith_pd <- pd(comm_matrix, tree_for_pd, include.root = FALSE)
        cat("✓ Faith's PD calculated without root (still valid measure)\n")
    }
    alpha_div$Faith_PD <- faith_pd$PD
}, error = function(e) {
    cat("⚠ Error calculating Faith's PD:", e$message, "\n")
    cat("Using species richness as alternative phylogenetic diversity measure\n")
    alpha_div$Faith_PD <- alpha_div$Observed
})

# Add sample metadata
sample_data_df <- data.frame(sample_data(physeq_tree))
alpha_div$Sample <- rownames(alpha_div)
alpha_div <- merge(alpha_div, sample_data_df, by.x = "Sample", by.y = "row.names")

# Display summary statistics
cat("\n=== Alpha Diversity Summary ===\n")
print(summary(alpha_div[, c("Observed", "Shannon", "Simpson", "Faith_PD")]))

# ==================== 5. Create Alpha Diversity Plots ====================

cat("\nCreating alpha diversity plots...\n")

# Set color palette
n_treatments <- length(unique(alpha_div$Treatment))
colors <- brewer.pal(min(max(n_treatments, 3), 11), "Set3")

# Function to perform pairwise comparisons and get significant results
get_significant_comparisons <- function(data, measure) {
    # Perform pairwise t-tests
```

```
  stat_test <- data %>%
    rstatix::pairwise_t_test(
      formula = as.formula(paste(measure, "~ Treatment")),
      p.adjust.method = "bonferroni"
    ) %>%
    filter(p.adj < 0.05) %>%    # Only keep significant results
    rstatix::add_xy_position(x = "Treatment")    # Add position information

  return(stat_test)
}

# Function to create individual violin plots with significance
create_alpha_violin_plot <- function(data, measure, y_label) {
  # Get significant comparisons
  stat_test <- get_significant_comparisons(data, measure)

  # Create base plot
  p <- ggplot(data, aes(x = Treatment, y = get(measure), fill = Treatment)) +
    geom_violin(alpha = 0.7, trim = FALSE) +
    geom_boxplot(width = 0.1, alpha = 0.5, outlier.shape = NA) +
    geom_jitter(width = 0.15, size = 1.5, alpha = 0.6) +
    scale_fill_manual(values = colors) +
    labs(
      title = paste(y_label, "Diversity"),
      x = "",    # Remove x-axis label
      y = y_label,
      fill = "Treatment"
    ) +
    theme_classic() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      axis.text.x = element_blank(),    # Remove x-axis text
      axis.ticks.x = element_blank(), # Remove x-axis ticks
      legend.position = "none"    # Remove individual legends
    ) +
    stat_summary(fun = mean, geom = "point", shape = 23, size = 3,
                 fill = "white", color = "black")

  # Add significance brackets if there are significant comparisons
  if (nrow(stat_test) > 0) {
    # Calculate y position for significance brackets
    y_max <- max(data[[measure]], na.rm = TRUE)
    y_range <- diff(range(data[[measure]], na.rm = TRUE))
```

```
        # Adjust y positions for multiple comparisons
        stat_test$y.position <- y_max + y_range * 0.1 +
                                        (seq_len(nrow(stat_test)) - 1) * y_range * 0.05

        p <- p +
          stat_pvalue_manual(
            stat_test,
            label = "p.adj",
            bracket.nudge.y = 0,
            step.increase = 0,
            hide.ns = TRUE,
            size = 3
          ) +
          # Extend y-axis to accommodate significance brackets
          ylim(min(data[[measure]], na.rm = TRUE),
                max(stat_test$y.position, na.rm = TRUE) * 1.1)
    }

    return(p)
}

# Create individual plots
p1 <- create_alpha_violin_plot(alpha_div, "Observed", "Observed ASV")
p2 <- create_alpha_violin_plot(alpha_div, "Shannon", "Shannon")
p3 <- create_alpha_violin_plot(alpha_div, "Simpson", "Simpson")
p4 <- create_alpha_violin_plot(alpha_div, "Faith_PD", "Faith's PD")

# ==================== 6. Statistical Analysis ====================

cat("\nPerforming statistical analysis...\n")

# Function to perform comprehensive statistical tests
perform_comprehensive_stats <- function(data, measure) {
    cat("\n--- Statistical analysis for", measure, "---\n")

    # ANOVA
    formula_str <- paste(measure, "~ Treatment")
    anova_result <- aov(as.formula(formula_str), data = data)
    anova_summary <- summary(anova_result)
    print(anova_summary)

    # Get p-value from ANOVA
    p_value <- anova_summary[[1]][["Pr(>F)"]][1]
```

```
    # Perform pairwise comparisons
    pairwise_results <- data %>%
        rstatix::pairwise_t_test(
            formula = as.formula(formula_str),
            p.adjust.method = "bonferroni"
        )

    cat("\nPairwise comparisons (Bonferroni corrected):\n")
    print(pairwise_results)

    # Show only significant comparisons
    significant_pairs <- pairwise_results %>% filter(p.adj < 0.05)
    if (nrow(significant_pairs) > 0) {
        cat("\nSignificant pairwise differences (p.adj < 0.05):\n")
        print(significant_pairs)
    } else {
        cat("\nNo significant pairwise differences found (p.adj < 0.05)\n")
    }

    return(list(
        anova = anova_result,
        p_value = p_value,
        pairwise = pairwise_results,
        significant_pairs = significant_pairs
    ))
}

# Perform statistical tests for each diversity index
stats_observed <- perform_comprehensive_stats(alpha_div, "Observed")
stats_shannon <- perform_comprehensive_stats(alpha_div, "Shannon")
stats_simpson <- perform_comprehensive_stats(alpha_div, "Simpson")
stats_faith <- perform_comprehensive_stats(alpha_div, "Faith_PD")

# ==================== 7. Combined Plot ====================

cat("\nCreating combined plot...\n")

# Create a plot with legend to extract the legend
legend_plot <- ggplot(alpha_div, aes(x = Treatment, y = Observed, fill = Treatment)) +
    geom_violin() +
    scale_fill_manual(values = colors) +
    guides(fill = guide_legend(title = "Treatment", nrow = 1)) +
    theme(legend.position = "bottom",
              legend.title = element_text(size = 12, face = "bold"),
```

```
            legend.text = element_text(size = 10))

# Extract legend
get_legend <- function(myggplot){
  tmp <- ggplot_gtable(ggplot_build(myggplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)
}

legend <- get_legend(legend_plot)

# Combine plots without legends
plots_grid <- grid.arrange(p1, p2, p3, p4, ncol = 2, nrow = 2)

# Combine plots with legend at bottom
combined_plot <- grid.arrange(plots_grid, legend,
                              ncol = 1, nrow = 2,
                              heights = c(10, 1),
                              top = "Alpha Diversity Analysis by Treatment")

# ==================== 8. Save Results ====================

# Save plots
ggsave("alpha_diversity_combined.png", combined_plot, width = 12, height = 10, dpi = 300)
ggsave("observed_diversity.png", p1, width = 8, height = 6, dpi = 300)
ggsave("shannon_diversity.png", p2, width = 8, height = 6, dpi = 300)
ggsave("simpson_diversity.png", p3, width = 8, height = 6, dpi = 300)
ggsave("faith_pd_diversity.png", p4, width = 8, height = 6, dpi = 300)

# Save alpha diversity data
write.csv(alpha_div, "alpha_diversity_results.csv", row.names = FALSE)

# Save statistical results summary
stats_summary <- data.frame(
  Index = c("Observed", "Shannon", "Simpson", "Faith_PD"),
  P_value = c(stats_observed$p_value, stats_shannon$p_value,
              stats_simpson$p_value, stats_faith$p_value),
  Significant = c(stats_observed$p_value < 0.05, stats_shannon$p_value < 0.05,
                  stats_simpson$p_value < 0.05, stats_faith$p_value < 0.05),
  Significant_pairs = c(nrow(stats_observed$significant_pairs),
                        nrow(stats_shannon$significant_pairs),
                        nrow(stats_simpson$significant_pairs),
                        nrow(stats_faith$significant_pairs))
```

```
)
write.csv(stats_summary, "statistical_analysis_summary.csv", row.names = FALSE)

# Save detailed pairwise comparison results
all_pairwise <- rbind(
  data.frame(Index = "Observed", stats_observed$pairwise),
  data.frame(Index = "Shannon", stats_shannon$pairwise),
  data.frame(Index = "Simpson", stats_simpson$pairwise),
  data.frame(Index = "Faith_PD", stats_faith$pairwise)
)
write.csv(all_pairwise, "pairwise_comparisons_detailed.csv", row.names = FALSE)

# Save only significant pairwise comparisons (handle empty data frames)
significant_dfs <- list()

if (nrow(stats_observed$significant_pairs) > 0) {
  significant_dfs[[1]] <- data.frame(Index = "Observed", stats_observed$significant_pairs)
}
if (nrow(stats_shannon$significant_pairs) > 0) {
  significant_dfs[[2]] <- data.frame(Index = "Shannon", stats_shannon$significant_pairs)
}
if (nrow(stats_simpson$significant_pairs) > 0) {
  significant_dfs[[3]] <- data.frame(Index = "Simpson", stats_simpson$significant_pairs)
}
if (nrow(stats_faith$significant_pairs) > 0) {
  significant_dfs[[4]] <- data.frame(Index = "Faith_PD", stats_faith$significant_pairs)
}

# Remove NULL elements and combine
significant_dfs <- significant_dfs[!sapply(significant_dfs, is.null)]

if (length(significant_dfs) > 0) {
  significant_only <- do.call(rbind, significant_dfs)
  write.csv(significant_only, "significant_pairwise_comparisons.csv", row.names = FALSE)
} else {
  # Create empty file with headers if no significant results
  empty_df <- data.frame(
    Index = character(0),
    .y. = character(0),
    group1 = character(0),
    group2 = character(0),
    n1 = numeric(0),
    n2 = numeric(0),
    statistic = numeric(0),
```

```
        df = numeric(0),
        p = numeric(0),
        p.adj = numeric(0),
        p.adj.signif = character(0)
    )
    write.csv(empty_df, "significant_pairwise_comparisons.csv", row.names = FALSE)
    significant_only <- empty_df
}

cat("\n🎉 Alpha diversity analysis complete!\n")
cat("Files saved:\n")
cat("- alpha_diversity_combined.png (combined violin plots with legend)\n")
cat("- Individual diversity plots (observed_diversity.png, etc.)\n")
cat("- alpha_diversity_results.csv (raw data)\n")
cat("- statistical_analysis_summary.csv (ANOVA results summary)\n")
cat("- pairwise_comparisons_detailed.csv (all pairwise comparisons)\n")
cat("- significant_pairwise_comparisons.csv (only significant comparisons)\n")

# ==================== 9. Display Final Results ====================

cat("\n=== Final Summary ===\n")
cat("Sample size per treatment:\n")
print(table(alpha_div$Treatment))

cat("\nMean values by treatment:\n")
diversity_summary <- alpha_div %>%
    group_by(Treatment) %>%
    summarise(
        Mean_Observed = round(mean(Observed), 2),
        Mean_Shannon = round(mean(Shannon), 3),
        Mean_Simpson = round(mean(Simpson), 3),
        Mean_Faith_PD = round(mean(Faith_PD), 3),
        .groups = 'drop'
    )
print(diversity_summary)

cat("\nStatistical significance summary:\n")
print(stats_summary)

cat("\nSignificant pairwise comparisons summary:\n")
if (nrow(significant_only) > 0) {
    print(significant_only[, c("Index", "group1", "group2", "p.adj")])
} else {
    cat("No significant pairwise differences found across all indices\n")
```

```
}

# Display the combined plot
print(combined_plot)

# ==================== Alpha Diversity Analysis by Legionella Status ====================

# Suppress warnings during execution
options(warn = -1)

# Load required packages
suppressMessages({
  library(phyloseq)
  library(ggplot2)
  library(dplyr)
  library(readr)
  library(ape)
  library(picante)
  library(gridExtra)
  library(RColorBrewer)
  library(phangorn)    # For midpoint rooting
  library(ggpubr)      # For statistical comparisons
  library(rstatix)     # For statistical tests
})

cat("Analyzing alpha diversity by Legionella status (all samples)...\n")

# ==================== 1. Prepare Data ====================

# Use all alpha diversity data (no filtering by Treatment)
legionella_alpha <- alpha_div

cat("Data preparation:\n")
cat("Total samples:", nrow(legionella_alpha), "\n")

# Check Legionella column
actual_legionella_levels <- sort(unique(legionella_alpha$Legionella))
cat("Legionella status levels found:", paste(actual_legionella_levels, collapse = ", "), "\n")

cat("Sample distribution by Legionella status:\n")
legionella_counts <- table(legionella_alpha$Legionella)
print(legionella_counts)

# Check if we have exactly 2 levels for comparison
```

```
if (length(actual_legionella_levels) != 2) {
   stop("Expected exactly 2 Legionella status levels for t-test comparison, found: ",
        length(actual_legionella_levels), " levels (",
        paste(actual_legionella_levels, collapse = ", "), ")")
}

# Check if we have enough data in each group
min_group_size <- min(legionella_counts)
if (min_group_size < 2) {
   stop("Insufficient samples in at least one group (minimum group size: ", min_group_size, ")")
}

# Check if we have enough total data
if (nrow(legionella_alpha) < 3) {
   stop("Not enough total samples for analysis (need at least 3)")
}

cat("✓ Data validation passed: 2 groups with sufficient samples for comparison\n")

# ==================== 2. Create Color Palette ====================

# Check actual Legionella levels in data and create color mapping
actual_levels <- unique(legionella_alpha$Legionella)
cat("Actual Legionella levels found:", paste(actual_levels, collapse = ", "), "\n")

# Create color palette based on actual levels
if (length(actual_levels) == 2) {
   # Create colors for the actual levels found
   legionella_colors <- c("#2E8B57", "#CD5C5C")    # Green and Red
   names(legionella_colors) <- sort(actual_levels)    # Assign to sorted levels

   cat("Color mapping:\n")
   for (i in seq_along(legionella_colors)) {
      cat(names(legionella_colors)[i], "=", legionella_colors[i], "\n")
   }
} else {
   stop("Expected exactly 2 Legionella status levels, found: ", length(actual_levels))
}

# ==================== 3. Statistical Functions ====================

# Function to perform t-test between the two Legionella groups
get_significant_comparisons_legionella <- function(data, measure) {
   # Check if both levels are present
```

```
  actual_levels <- unique(data$Legionella)
  if (length(actual_levels) < 2) {
    return(data.frame())
  }

  # Perform t-test between the two groups
  suppressWarnings({
    stat_test <- data %>%
      rstatix::t_test(
        formula = as.formula(paste(measure, "~ Legionella"))
      )

    # Add significance information
    stat_test$significant <- stat_test$p < 0.05
    stat_test$group1 <- sort(actual_levels)[1]
    stat_test$group2 <- sort(actual_levels)[2]
  })

  cat("Statistical test for", measure, "- p-value:", round(stat_test$p[1], 6),
      ", significant:", stat_test$significant[1], "\n")

  return(stat_test)
}

# Function to create individual violin plots with significance
create_legionella_violin_plot <- function(data, measure, y_label) {
  # Get statistical test results
  stat_test <- get_significant_comparisons_legionella(data, measure)

  # Get actual levels and sort them
  actual_levels <- sort(unique(data$Legionella))

  # Create base plot
  suppressWarnings({
    p <- ggplot(data, aes(x = factor(Legionella, levels = actual_levels),
                          y = get(measure),
                          fill = factor(Legionella, levels = actual_levels))) +
      geom_violin(alpha = 0.7, trim = FALSE) +
      geom_boxplot(width = 0.1, alpha = 0.5, outlier.shape = NA) +
      geom_jitter(width = 0.15, size = 2, alpha = 0.7) +
      scale_fill_manual(values = legionella_colors, name = "Legionella Status") +
      scale_x_discrete(name = "Legionella Status") +
      labs(
        title = paste(y_label, "Diversity by Legionella Status"),
```

```
      x = "Legionella Status",
      y = y_label
    ) +
    theme_classic() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      axis.text.x = element_text(size = 12),
      axis.title.x = element_text(size = 12, face = "bold"),
      axis.title.y = element_text(size = 12, face = "bold"),
      legend.position = "none"    # Remove legend
    ) +
    stat_summary(fun = mean, geom = "point", shape = 23, size = 3,
                 fill = "white", color = "black")
})

# Add significance annotation if test was performed
if (nrow(stat_test) > 0) {
  # Calculate y position for significance brackets
  y_max <- max(data[[measure]], na.rm = TRUE)
  y_min <- min(data[[measure]], na.rm = TRUE)
  y_range <- y_max - y_min

  # Set y position for the bracket
  bracket_y <- y_max + y_range * 0.1

  # Add significance annotation
  if (stat_test$significant[1]) {
    # Add significance bracket and p-value for significant results
    suppressWarnings({
      p <- p +
        # Add horizontal line
        annotate("segment",
                 x = 1, xend = 2,
                 y = bracket_y, yend = bracket_y,
                 color = "black", size = 0.5) +
        # Add vertical lines
        annotate("segment",
                 x = 1, xend = 1,
                 y = bracket_y - y_range * 0.02,
                 yend = bracket_y,
                 color = "black", size = 0.5) +
        annotate("segment",
                 x = 2, xend = 2,
                 y = bracket_y - y_range * 0.02,
```

```
                yend = bracket_y,
                color = "black", size = 0.5) +
        # Add p-value text
        annotate("text",
                x = 1.5,
                y = bracket_y + y_range * 0.03,
                label = paste("p =", format(stat_test$p[1], digits = 3, scientific = TRUE)),
                size = 3.5, fontface = "bold") +
        # Extend y-axis to accommodate significance brackets
        ylim(y_min, bracket_y + y_range * 0.15)
    })
  } else {
    # Add "ns" (not significant) annotation for non-significant results
    suppressWarnings({
      p <- p +
        annotate("text",
                x = 1.5,
                y = y_max + y_range * 0.05,
                label = paste("ns (p =", format(stat_test$p[1], digits = 3), ")"),
                size = 3, color = "gray50") +
        ylim(y_min, y_max + y_range * 0.15)
    })
  }
 }

 return(p)
}

# ==================== 4. Create Alpha Diversity Plots ====================

cat("\nCreating alpha diversity plots by Legionella status...\n")

# Create individual plots
suppressWarnings({
 p1_legionella <- create_legionella_violin_plot(legionella_alpha, "Observed", "Observed ASV")
 p2_legionella <- create_legionella_violin_plot(legionella_alpha, "Shannon", "Shannon")
 p3_legionella <- create_legionella_violin_plot(legionella_alpha, "Simpson", "Simpson")
 p4_legionella <- create_legionella_violin_plot(legionella_alpha, "Faith_PD", "Faith's PD")
})

# ==================== 5. Statistical Analysis ====================

cat("\nPerforming statistical analysis by Legionella status...\n")
```

```
# Function to perform t-test between the two Legionella groups
perform_legionella_stats <- function(data, measure) {
  cat("\n--- Statistical analysis for", measure, "by Legionella status ---\n")

  actual_levels <- sort(unique(data$Legionella))
  cat("Total samples:", nrow(data), "\n")
  cat("Legionella status levels:", paste(actual_levels, collapse = " vs "), "\n")
  cat("Legionella status distribution:\n")
  print(table(data$Legionella))

  if (length(actual_levels) < 2) {
    cat("⚠ Insufficient Legionella status levels for comparison\n")
    return(list(
      t_test = NULL,
      p_value = NA,
      significant = FALSE
    ))
  }

  # Perform t-test between the two groups
  suppressWarnings({
    t_result <- data %>%
      rstatix::t_test(
        formula = as.formula(paste(measure, "~ Legionella"))
      )
  })

  print(t_result)

  # Get p-value
  p_value <- t_result$p[1]
  significant <- p_value < 0.05

  cat("P-value:", round(p_value, 6), "\n")
  cat("Significant (p < 0.05):", significant, "\n")

  # Calculate group means for interpretation
  group_means <- data %>%
    group_by(Legionella) %>%
    summarise(mean_value = mean(get(measure), na.rm = TRUE), .groups = 'drop')

  cat("Group means:\n")
  print(group_means)
```

```
  return(list(
    t_test = t_result,
    p_value = p_value,
    significant = significant,
    group_means = group_means
  ))
}

# Perform statistical tests for each diversity index
suppressWarnings({
  stats_observed_legionella <- perform_legionella_stats(legionella_alpha, "Observed")
  stats_shannon_legionella <- perform_legionella_stats(legionella_alpha, "Shannon")
  stats_simpson_legionella <- perform_legionella_stats(legionella_alpha, "Simpson")
  stats_faith_legionella <- perform_legionella_stats(legionella_alpha, "Faith_PD")
})

# ==================== 6. Combined Plot ====================

cat("\nCreating combined plot by Legionella status...\n")

# Combine plots directly
suppressWarnings({
  combined_plot_legionella <- grid.arrange(p1_legionella, p2_legionella, p3_legionella,
p4_legionella,
                                           ncol = 2, nrow = 2,
                                           top = "Alpha Diversity Analysis by Legionella
Status")
})

# ==================== 7. Save Results ====================

# Save plots
suppressWarnings({
  ggsave("alpha_diversity_legionella_combined.png", combined_plot_legionella, width = 12,
height = 10, dpi = 300)
  ggsave("observed_diversity_legionella.png", p1_legionella, width = 8, height = 6, dpi = 300)
  ggsave("shannon_diversity_legionella.png", p2_legionella, width = 8, height = 6, dpi = 300)
  ggsave("simpson_diversity_legionella.png", p3_legionella, width = 8, height = 6, dpi = 300)
  ggsave("faith_pd_diversity_legionella.png", p4_legionella, width = 8, height = 6, dpi = 300)
})

# Save alpha diversity data with Legionella status
suppressWarnings({
  write.csv(legionella_alpha, "alpha_diversity_legionella_all_samples.csv", row.names = FALSE)
```

```
})

# Save statistical results summary
stats_summary_legionella <- data.frame(
  Index = c("Observed", "Shannon", "Simpson", "Faith_PD"),
  P_value = c(
    ifelse(is.null(stats_observed_legionella$t_test), NA, stats_observed_legionella$p_value),
    ifelse(is.null(stats_shannon_legionella$t_test), NA, stats_shannon_legionella$p_value),
    ifelse(is.null(stats_simpson_legionella$t_test), NA, stats_simpson_legionella$p_value),
    ifelse(is.null(stats_faith_legionella$t_test), NA, stats_faith_legionella$p_value)
  ),
  Significant = c(
    ifelse(is.na(stats_observed_legionella$p_value), FALSE,
stats_observed_legionella$significant),
    ifelse(is.na(stats_shannon_legionella$p_value), FALSE,
stats_shannon_legionella$significant),
    ifelse(is.na(stats_simpson_legionella$p_value), FALSE, stats_simpson_legionella$significant),
    ifelse(is.na(stats_faith_legionella$p_value), FALSE, stats_faith_legionella$significant)
  ),
  Test = paste("t-test (", paste(actual_legionella_levels, collapse = " vs "), ")", sep = "")
)
suppressWarnings({
  write.csv(stats_summary_legionella, "statistical_analysis_legionella_summary.csv", row.names
= FALSE)
})

# Save detailed t-test results
all_ttest_legionella_list <- list()
if (!is.null(stats_observed_legionella$t_test)) {
  all_ttest_legionella_list[[1]] <- data.frame(Index = "Observed",
stats_observed_legionella$t_test)
}
if (!is.null(stats_shannon_legionella$t_test)) {
  all_ttest_legionella_list[[2]] <- data.frame(Index = "Shannon",
stats_shannon_legionella$t_test)
}
if (!is.null(stats_simpson_legionella$t_test)) {
  all_ttest_legionella_list[[3]] <- data.frame(Index = "Simpson", stats_simpson_legionella$t_test)
}
if (!is.null(stats_faith_legionella$t_test)) {
  all_ttest_legionella_list[[4]] <- data.frame(Index = "Faith_PD", stats_faith_legionella$t_test)
}

# Remove NULL elements and combine
```

```
all_ttest_legionella_list <- all_ttest_legionella_list[!sapply(all_ttest_legionella_list, is.null)]

if (length(all_ttest_legionella_list) > 0) {
  all_ttest_legionella <- do.call(rbind, all_ttest_legionella_list)
  suppressWarnings({
    write.csv(all_ttest_legionella, "ttest_results_legionella_detailed.csv", row.names = FALSE)
  })
} else {
  # Create empty file if no results
  empty_ttest_legionella <- data.frame(
    Index = character(0),
    .y. = character(0),
    group1 = character(0),
    group2 = character(0),
    n1 = numeric(0),
    n2 = numeric(0),
    statistic = numeric(0),
    df = numeric(0),
    p = numeric(0)
  )
  suppressWarnings({
    write.csv(empty_ttest_legionella, "ttest_results_legionella_detailed.csv", row.names =
FALSE)
  })
}

# Save only significant results
significant_results_legionella <- stats_summary_legionella %>%
  filter(Significant == TRUE)

suppressWarnings({
  write.csv(significant_results_legionella, "significant_ttest_results_legionella.csv", row.names =
FALSE)
})

# ==================== 8. Display Final Results ====================

cat("\n 🎉 Alpha diversity analysis by Legionella status complete!\n")
cat("Files saved:\n")
cat("- alpha_diversity_legionella_combined.png (combined violin plots)\n")
cat("- Individual diversity plots by Legionella status (observed_diversity_legionella.png, etc.)\n")
cat("- alpha_diversity_legionella_all_samples.csv (raw data for all samples)\n")
cat("- statistical_analysis_legionella_summary.csv (t-test results summary)\n")
cat("- ttest_results_legionella_detailed.csv (detailed t-test results)\n")
```

```
cat("- significant_ttest_results_legionella.csv (only significant results)\n")

cat("\n=== Final Summary by Legionella Status ===\n")
cat("Sample size by Legionella status:\n")
print(table(legionella_alpha$Legionella))

cat("\nMean values by Legionella status:\n")
diversity_summary_legionella <- legionella_alpha %>%
  group_by(Legionella) %>%
  summarise(
    n = n(),
    Mean_Observed = round(mean(Observed), 2),
    Mean_Shannon = round(mean(Shannon), 3),
    Mean_Simpson = round(mean(Simpson), 3),
    Mean_Faith_PD = round(mean(Faith_PD), 3),
    .groups = 'drop'
  )
print(diversity_summary_legionella)

cat("\nStatistical significance summary (", paste(actual_legionella_levels, collapse = " vs "), "):\n",
sep = "")
print(stats_summary_legionella)

cat("\nSignificant results summary:\n")
if (nrow(significant_results_legionella) > 0) {
  print(significant_results_legionella[, c("Index", "P_value", "Significant")])
} else {
  cat("No significant differences found between", paste(actual_legionella_levels, collapse = " and
"),
      "across all indices\n")
}

# Display the combined plot
suppressWarnings({
  print(combined_plot_legionella)
})

# Reset warning options
options(warn = 0)

# ==================== Alpha Diversity Analysis by Biofilm Age Groups (Untreated Samples)
==================

# Suppress warnings during execution
```

```
options(warn = -1)

# Load required packages
suppressMessages({
  library(phyloseq)
  library(ggplot2)
  library(dplyr)
  library(readr)
  library(ape)
  library(picante)
  library(gridExtra)
  library(RColorBrewer)
  library(phangorn)    # For midpoint rooting
  library(ggpubr)      # For statistical comparisons
  library(rstatix)     # For statistical tests
})

cat("Analyzing alpha diversity by Biofilm Age groups (Untreated samples only)...\n")

# ==================== 1. Filter and Prepare Data ====================

# Filter for Untreated samples only
untreated_biofilm_alpha <- alpha_div %>%
  filter(Treatment == "Untreated")

cat("Data preparation:\n")
cat("Total Untreated samples:", nrow(untreated_biofilm_alpha), "\n")

# Check Biofilm_Age values
if (!"Biofilm_Age" %in% colnames(untreated_biofilm_alpha)) {
  stop("Biofilm_Age column not found in metadata")
}

cat("Biofilm_Age values found (weeks):",
paste(sort(unique(untreated_biofilm_alpha$Biofilm_Age)), collapse = ", "), "\n")
cat("Sample distribution by Biofilm_Age:\n")
biofilm_age_counts <- table(untreated_biofilm_alpha$Biofilm_Age)
print(biofilm_age_counts)

# ==================== 2. Create Biofilm Age Groups ====================

# Define age groups based on user specifications
create_biofilm_age_groups <- function(data) {
  data$Biofilm_Age_Group <- NA
```

```
  # Group 1: 5 weeks
  data$Biofilm_Age_Group[data$Biofilm_Age == 5] <- "5weeks"

  # Group 2: 32 and 36 weeks
  data$Biofilm_Age_Group[data$Biofilm_Age %in% c(32, 36)] <- "32,36weeks"

  # Group 3: 52 and 104 weeks
  data$Biofilm_Age_Group[data$Biofilm_Age %in% c(52, 104)] <- "52,104weeks"

  # Check for any ungrouped ages
  ungrouped <- data[is.na(data$Biofilm_Age_Group), ]
  if (nrow(ungrouped) > 0) {
    cat("⚠ Warning: Some Biofilm_Age values not assigned to groups:\n")
    print(table(ungrouped$Biofilm_Age))
    cat("These samples will be excluded from analysis\n")
  }

  # Remove samples without group assignment
  data <- data[!is.na(data$Biofilm_Age_Group), ]

  return(data)
}

# Apply grouping
untreated_biofilm_alpha <- create_biofilm_age_groups(untreated_biofilm_alpha)

cat("\nAfter grouping:\n")
cat("Total samples with group assignment:", nrow(untreated_biofilm_alpha), "\n")
cat("Biofilm Age groups:", paste(unique(untreated_biofilm_alpha$Biofilm_Age_Group), collapse
= ", "), "\n")
cat("Sample distribution by Biofilm Age groups:\n")
group_counts <- table(untreated_biofilm_alpha$Biofilm_Age_Group)
print(group_counts)

# Check if we have enough data
if (nrow(untreated_biofilm_alpha) < 3) {
  stop("Not enough samples after grouping for analysis (need at least 3)")
}

if (length(unique(untreated_biofilm_alpha$Biofilm_Age_Group)) < 2) {
  stop("Need at least 2 different Biofilm Age groups for comparison")
}
```

```
# Check group sizes
min_group_size <- min(group_counts)
if (min_group_size < 1) {
  cat("⚠ Warning: Some groups have very few samples (minimum:", min_group_size, ")\n")
}

# ==================== 3. Create Color Palette ====================

# Set color palette for Biofilm Age groups
n_groups <- length(unique(untreated_biofilm_alpha$Biofilm_Age_Group))
if (n_groups <= 3) {
  biofilm_colors <- c("#1B9E77", "#D95F02", "#7570B3")[1:n_groups]  # Manual colors for up to
3 groups
} else {
  biofilm_colors <- brewer.pal(min(n_groups, 11), "Set1")
}

# Assign colors to groups (ordered by time)
group_order <- c("5weeks", "32,36weeks", "52,104weeks")
group_order <- group_order[group_order %in%
unique(untreated_biofilm_alpha$Biofilm_Age_Group)]
names(biofilm_colors) <- group_order

cat("Color mapping for Biofilm Age groups:\n")
for (i in seq_along(biofilm_colors)) {
  cat(names(biofilm_colors)[i], "=", biofilm_colors[i], "\n")
}

# ==================== 4. Statistical Functions ====================

# Function to perform ANOVA and pairwise comparisons
get_significant_comparisons_biofilm <- function(data, measure) {
  # Check if we have enough groups
  if (length(unique(data$Biofilm_Age_Group)) < 2) {
    return(data.frame())
  }

  # Perform pairwise t-tests if more than 2 groups, or t-test if 2 groups
  suppressWarnings({
    if (length(unique(data$Biofilm_Age_Group)) == 2) {
      # Simple t-test for 2 groups
      stat_test <- data %>%
        rstatix::t_test(
          formula = as.formula(paste(measure, "~ Biofilm_Age_Group"))
```

```
            ) %>%
            mutate(p.adj = p)
        } else {
            # Pairwise t-tests for multiple groups
            stat_test <- data %>%
                rstatix::pairwise_t_test(
                    formula = as.formula(paste(measure, "~ Biofilm_Age_Group")),
                    p.adjust.method = "bonferroni"
                )
        }

        # Filter for significant results
        stat_test <- stat_test %>% filter(p.adj < 0.05)
    })

    if (nrow(stat_test) > 0) {
        cat("Significant comparisons found for", measure, ":\n")
        print(stat_test[, c("group1", "group2", "p", "p.adj")])
    }

    return(stat_test)
}

# Function to create individual violin plots with significance
create_biofilm_violin_plot <- function(data, measure, y_label) {
    # Get significant comparisons
    stat_test <- get_significant_comparisons_biofilm(data, measure)

    # Create base plot with ordered groups
    suppressWarnings({
        p <- ggplot(data, aes(x = factor(Biofilm_Age_Group, levels = group_order),
                              y = get(measure),
                              fill = factor(Biofilm_Age_Group, levels = group_order))) +
            geom_violin(alpha = 0.7, trim = FALSE) +
            geom_boxplot(width = 0.1, alpha = 0.5, outlier.shape = NA) +
            geom_jitter(width = 0.15, size = 2, alpha = 0.7) +
            scale_fill_manual(values = biofilm_colors, name = "Biofilm Age Group") +
            scale_x_discrete(name = "Biofilm Age Group") +
            labs(
                title = paste(y_label, "Diversity by Biofilm Age"),
                x = "Biofilm Age Group",
                y = y_label
            ) +
            theme_classic() +
```

```
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      axis.text.x = element_text(angle = 45, hjust = 1, size = 10),
      axis.title.x = element_text(size = 12, face = "bold"),
      axis.title.y = element_text(size = 12, face = "bold"),
      legend.position = "none"    # Remove legend since x-axis shows group names
    ) +
    stat_summary(fun = mean, geom = "point", shape = 23, size = 3,
                 fill = "white", color = "black")
})

# Add significance annotations if there are significant comparisons
if (nrow(stat_test) > 0) {
  # Calculate y position for significance brackets
  y_max <- max(data[[measure]], na.rm = TRUE)
  y_min <- min(data[[measure]], na.rm = TRUE)
  y_range <- y_max - y_min

  # Add significance annotations for each significant comparison
  for (i in 1:nrow(stat_test)) {
    # Get positions of compared groups
    group1_pos <- which(group_order == stat_test$group1[i])
    group2_pos <- which(group_order == stat_test$group2[i])

    if (length(group1_pos) > 0 && length(group2_pos) > 0) {
      # Calculate bracket position
      bracket_y <- y_max + y_range * (0.1 + (i-1) * 0.05)

      suppressWarnings({
        p <- p +
          # Add horizontal line
          annotate("segment",
                   x = group1_pos, xend = group2_pos,
                   y = bracket_y, yend = bracket_y,
                   color = "black", size = 0.5) +
          # Add vertical lines
          annotate("segment",
                   x = group1_pos, xend = group1_pos,
                   y = bracket_y - y_range * 0.02,
                   yend = bracket_y,
                   color = "black", size = 0.5) +
          annotate("segment",
                   x = group2_pos, xend = group2_pos,
                   y = bracket_y - y_range * 0.02,
```

```
                yend = bracket_y,
                color = "black", size = 0.5) +
        # Add p-value text
        annotate("text",
                x = (group1_pos + group2_pos) / 2,
                y = bracket_y + y_range * 0.02,
                label = format(stat_test$p.adj[i], digits = 3, scientific = TRUE),
                size = 3, fontface = "bold")
    })
  }
 }

 # Extend y-axis to accommodate all brackets
 max_bracket_y <- y_max + y_range * (0.1 + nrow(stat_test) * 0.05)
 p <- p + ylim(y_min, max_bracket_y + y_range * 0.05)
 }

 return(p)
}

# ==================== 5. Create Alpha Diversity Plots ====================

cat("\nCreating alpha diversity plots by Biofilm Age groups...\n")

# Create individual plots
suppressWarnings({
 p1_biofilm <- create_biofilm_violin_plot(untreated_biofilm_alpha, "Observed", "Observed ASV")
 p2_biofilm <- create_biofilm_violin_plot(untreated_biofilm_alpha, "Shannon", "Shannon")
 p3_biofilm <- create_biofilm_violin_plot(untreated_biofilm_alpha, "Simpson", "Simpson")
 p4_biofilm <- create_biofilm_violin_plot(untreated_biofilm_alpha, "Faith_PD", "Faith's PD")
})

# ==================== 6. Statistical Analysis ====================

cat("\nPerforming statistical analysis by Biofilm Age groups...\n")

# Function to perform comprehensive statistical tests
perform_biofilm_stats <- function(data, measure) {
 cat("\n--- Statistical analysis for", measure, "by Biofilm Age groups ---\n")

 cat("Total samples:", nrow(data), "\n")
 cat("Biofilm Age group distribution:\n")
 print(table(data$Biofilm_Age_Group))
```

```
n_groups <- length(unique(data$Biofilm_Age_Group))
if (n_groups < 2) {
  cat("⚠ Insufficient groups for comparison\n")
  return(list(
    anova = NULL,
    pairwise = data.frame(),
    p_value = NA,
    significant = FALSE
  ))
}

# Perform ANOVA if more than 2 groups, t-test if 2 groups
suppressWarnings({
  if (n_groups == 2) {
    # Simple t-test
    test_result <- data %>%
      rstatix::t_test(
        formula = as.formula(paste(measure, "~ Biofilm_Age_Group"))
      )
    p_value <- test_result$p[1]
    pairwise_results <- test_result
    anova_result <- NULL
    cat("Two-sample t-test performed\n")
  } else {
    # ANOVA followed by pairwise comparisons
    anova_result <- aov(as.formula(paste(measure, "~ Biofilm_Age_Group")), data = data)
    anova_summary <- summary(anova_result)
    print(anova_summary)
    p_value <- anova_summary[[1]][["Pr(>F)"]][1]

    # Pairwise comparisons
    pairwise_results <- data %>%
      rstatix::pairwise_t_test(
        formula = as.formula(paste(measure, "~ Biofilm_Age_Group")),
        p.adjust.method = "bonferroni"
      )
    cat("ANOVA performed, followed by pairwise comparisons\n")
  }

  print(pairwise_results)
})

significant <- p_value < 0.05
```

```
  cat("Overall p-value:", round(p_value, 6), "\n")
  cat("Significant (p < 0.05):", significant, "\n")

  # Show significant pairwise comparisons
  significant_pairs <- pairwise_results %>%
    filter(if("p.adj" %in% colnames(.)) p.adj < 0.05 else p < 0.05)

  if (nrow(significant_pairs) > 0) {
    cat("Significant pairwise differences:\n")
    print(significant_pairs)
  } else {
    cat("No significant pairwise differences found\n")
  }

  return(list(
    anova = anova_result,
    pairwise = pairwise_results,
    significant_pairs = significant_pairs,
    p_value = p_value,
    significant = significant
  ))
}

# Perform statistical tests for each diversity index
suppressWarnings({
  stats_observed_biofilm <- perform_biofilm_stats(untreated_biofilm_alpha, "Observed")
  stats_shannon_biofilm <- perform_biofilm_stats(untreated_biofilm_alpha, "Shannon")
  stats_simpson_biofilm <- perform_biofilm_stats(untreated_biofilm_alpha, "Simpson")
  stats_faith_biofilm <- perform_biofilm_stats(untreated_biofilm_alpha, "Faith_PD")
})

# ==================== 7. Combined Plot ====================

cat("\nCreating combined plot by Biofilm Age groups...\n")

# Combine plots directly
suppressWarnings({
  combined_plot_biofilm <- grid.arrange(p1_biofilm, p2_biofilm, p3_biofilm, p4_biofilm,
                                        ncol = 2, nrow = 2,
                                        top = "Alpha Diversity Analysis by Biofilm Age
Groups (Untreated Samples)")
})

# ==================== 8. Save Results ====================
```

```
# Save plots
suppressWarnings({
  ggsave("alpha_diversity_biofilm_age_combined.png", combined_plot_biofilm, width = 12, height = 10, dpi = 300)
  ggsave("observed_diversity_biofilm_age.png", p1_biofilm, width = 8, height = 6, dpi = 300)
  ggsave("shannon_diversity_biofilm_age.png", p2_biofilm, width = 8, height = 6, dpi = 300)
  ggsave("simpson_diversity_biofilm_age.png", p3_biofilm, width = 8, height = 6, dpi = 300)
  ggsave("faith_pd_diversity_biofilm_age.png", p4_biofilm, width = 8, height = 6, dpi = 300)
})

# Save alpha diversity data with Biofilm Age groups
suppressWarnings({
  write.csv(untreated_biofilm_alpha, "alpha_diversity_biofilm_age_untreated.csv", row.names = FALSE)
})

# Save statistical results summary
stats_summary_biofilm <- data.frame(
  Index = c("Observed", "Shannon", "Simpson", "Faith_PD"),
  P_value = c(
    ifelse(is.na(stats_observed_biofilm$p_value), NA, stats_observed_biofilm$p_value),
    ifelse(is.na(stats_shannon_biofilm$p_value), NA, stats_shannon_biofilm$p_value),
    ifelse(is.na(stats_simpson_biofilm$p_value), NA, stats_simpson_biofilm$p_value),
    ifelse(is.na(stats_faith_biofilm$p_value), NA, stats_faith_biofilm$p_value)
  ),
  Significant = c(
    ifelse(is.na(stats_observed_biofilm$p_value), FALSE, stats_observed_biofilm$significant),
    ifelse(is.na(stats_shannon_biofilm$p_value), FALSE, stats_shannon_biofilm$significant),
    ifelse(is.na(stats_simpson_biofilm$p_value), FALSE, stats_simpson_biofilm$significant),
    ifelse(is.na(stats_faith_biofilm$p_value), FALSE, stats_faith_biofilm$significant)
  ),
  Significant_pairs = c(
    nrow(stats_observed_biofilm$significant_pairs),
    nrow(stats_shannon_biofilm$significant_pairs),
    nrow(stats_simpson_biofilm$significant_pairs),
    nrow(stats_faith_biofilm$significant_pairs)
  ),
  Test = ifelse(length(group_order) == 2, "t-test", "ANOVA + pairwise t-tests")
)
suppressWarnings({
  write.csv(stats_summary_biofilm, "statistical_analysis_biofilm_age_summary.csv", row.names = FALSE)
})
```

```
# Save detailed pairwise comparison results
all_pairwise_biofilm_list <- list()
if (nrow(stats_observed_biofilm$pairwise) > 0) {
  all_pairwise_biofilm_list[[1]] <- data.frame(Index = "Observed",
stats_observed_biofilm$pairwise)
}
if (nrow(stats_shannon_biofilm$pairwise) > 0) {
  all_pairwise_biofilm_list[[2]] <- data.frame(Index = "Shannon",
stats_shannon_biofilm$pairwise)
}
if (nrow(stats_simpson_biofilm$pairwise) > 0) {
  all_pairwise_biofilm_list[[3]] <- data.frame(Index = "Simpson",
stats_simpson_biofilm$pairwise)
}
if (nrow(stats_faith_biofilm$pairwise) > 0) {
  all_pairwise_biofilm_list[[4]] <- data.frame(Index = "Faith_PD", stats_faith_biofilm$pairwise)
}

# Remove NULL elements and combine
all_pairwise_biofilm_list <- all_pairwise_biofilm_list[!sapply(all_pairwise_biofilm_list, is.null)]

if (length(all_pairwise_biofilm_list) > 0) {
  all_pairwise_biofilm <- do.call(rbind, all_pairwise_biofilm_list)
  suppressWarnings({
    write.csv(all_pairwise_biofilm, "pairwise_comparisons_biofilm_age_detailed.csv",
row.names = FALSE)
  })
}

# Save only significant pairwise comparisons
significant_biofilm_list <- list()
if (nrow(stats_observed_biofilm$significant_pairs) > 0) {
  significant_biofilm_list[[1]] <- data.frame(Index = "Observed",
stats_observed_biofilm$significant_pairs)
}
if (nrow(stats_shannon_biofilm$significant_pairs) > 0) {
  significant_biofilm_list[[2]] <- data.frame(Index = "Shannon",
stats_shannon_biofilm$significant_pairs)
}
if (nrow(stats_simpson_biofilm$significant_pairs) > 0) {
  significant_biofilm_list[[3]] <- data.frame(Index = "Simpson",
stats_simpson_biofilm$significant_pairs)
}
```

```
if (nrow(stats_faith_biofilm$significant_pairs) > 0) {
  significant_biofilm_list[[4]] <- data.frame(Index = "Faith_PD", stats_faith_biofilm$significant_pairs)
}

significant_biofilm_list <- significant_biofilm_list[!sapply(significant_biofilm_list, is.null)]

if (length(significant_biofilm_list) > 0) {
  significant_only_biofilm <- do.call(rbind, significant_biofilm_list)
  suppressWarnings({
    write.csv(significant_only_biofilm, "significant_pairwise_comparisons_biofilm_age.csv", row.names = FALSE)
  })
} else {
  # Create empty file if no significant results
  empty_df_biofilm <- data.frame(
    Index = character(0),
    .y. = character(0),
    group1 = character(0),
    group2 = character(0),
    n1 = numeric(0),
    n2 = numeric(0),
    statistic = numeric(0),
    df = numeric(0),
    p = numeric(0),
    p.adj = numeric(0),
    p.adj.signif = character(0)
  )
  suppressWarnings({
    write.csv(empty_df_biofilm, "significant_pairwise_comparisons_biofilm_age.csv", row.names = FALSE)
  })
  significant_only_biofilm <- empty_df_biofilm
}

# ==================== 9. Display Final Results ====================

cat("\n 🎉 Alpha diversity analysis by Biofilm Age groups complete!\n")
cat("Files saved:\n")
cat("- alpha_diversity_biofilm_age_combined.png (combined violin plots)\n")
cat("- Individual diversity plots by Biofilm Age (observed_diversity_biofilm_age.png, etc.)\n")
cat("- alpha_diversity_biofilm_age_untreated.csv (raw data for untreated samples)\n")
cat("- statistical_analysis_biofilm_age_summary.csv (statistical results summary)\n")
cat("- pairwise_comparisons_biofilm_age_detailed.csv (detailed pairwise comparisons)\n")
```

```
cat("- significant_pairwise_comparisons_biofilm_age.csv (only significant comparisons)\n")

cat("\n=== Final Summary by Biofilm Age Groups ===\n")
cat("Sample size by Biofilm Age group:\n")
print(table(untreated_biofilm_alpha$Biofilm_Age_Group))

cat("\nMean values by Biofilm Age group:\n")
diversity_summary_biofilm <- untreated_biofilm_alpha %>%
  group_by(Biofilm_Age_Group) %>%
  summarise(
    n = n(),
    Mean_Observed = round(mean(Observed), 2),
    Mean_Shannon = round(mean(Shannon), 3),
    Mean_Simpson = round(mean(Simpson), 3),
    Mean_Faith_PD = round(mean(Faith_PD), 3),
    .groups = 'drop'
  )
print(diversity_summary_biofilm)

cat("\nStatistical significance summary:\n")
print(stats_summary_biofilm)

cat("\nSignificant pairwise comparisons summary:\n")
if (nrow(significant_only_biofilm) > 0) {
  print(significant_only_biofilm[, c("Index", "group1", "group2", "p.adj")])
} else {
  cat("No significant pairwise differences found across all indices\n")
}

# Display the combined plot
suppressWarnings({
  print(combined_plot_biofilm)
})

# Reset warning options
options(warn = 0)
```

The following script is for beta-diversity.

```
# ==================== 1. Load Required R Packages ====================
library(phyloseq)
library(readxl)
library(readr)
library(dplyr)
library(tibble)

# ==================== 2. Set File Paths ====================
metadata_path <- "C:/Users/ASUS/Desktop/imeta/metadata.xlsx"
asv_table_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_table.csv"
taxonomy_path <- "C:/Users/ASUS/Desktop/imeta/result/taxonomy_table.xlsx"

# ==================== 3. Read Data Files ====================

# 3.1 Read sample metadata
cat("Reading sample metadata...\n")
metadata <- read_excel(metadata_path, sheet = 1)
print(paste("Metadata contains", nrow(metadata), "samples"))
print(paste("Metadata columns:", paste(colnames(metadata), collapse = ", ")))

# 3.2 Read ASV abundance table
cat("\nReading ASV abundance table...\n")
asv_table <- read_csv(asv_table_path)
print(paste("ASV table contains", nrow(asv_table), "ASVs and", ncol(asv_table)-1, "samples"))

# 3.3 Read taxonomic information
cat("\nReading taxonomic information...\n")
taxonomy <- read_excel(taxonomy_path, sheet = 1)
print(paste("Taxonomy table contains", nrow(taxonomy), "ASVs"))
print(paste("Taxonomic ranks:", paste(colnames(taxonomy)[-1], collapse = ", ")))

# ==================== 4. Data Preprocessing ====================

# 4.1 Process sample metadata
# Set sample_id as row names
sample_data_df <- metadata %>%
  column_to_rownames("sample_id")

# 4.2 Process ASV abundance table
```

```
# Set ASV_id as row names, convert to matrix
otu_table_df <- asv_table %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check and ensure consistent sample ordering
cat("\nChecking sample ID consistency...\n")
metadata_samples <- rownames(sample_data_df)
asv_samples <- colnames(otu_table_df)

if (length(setdiff(metadata_samples, asv_samples)) == 0 &&
      length(setdiff(asv_samples, metadata_samples)) == 0) {
  cat("✓ Sample IDs match completely!\n")
} else {
  cat("⚠ Sample IDs do not match, need to check!\n")
}

# 4.3 Process taxonomic information
# Set ASV_id as row names
tax_table_df <- taxonomy %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check ASV ID consistency
asv_ids_otu <- rownames(otu_table_df)
asv_ids_tax <- rownames(tax_table_df)

if (length(setdiff(asv_ids_otu, asv_ids_tax)) == 0 &&
      length(setdiff(asv_ids_tax, asv_ids_otu)) == 0) {
  cat("✓ ASV IDs match completely!\n")
} else {
  cat("⚠ ASV IDs do not match, need to check!\n")
}

# ==================== 5. Create Phyloseq Object Components ====================

# 5.1 Create OTU table object
cat("\nCreating phyloseq components...\n")
OTU <- otu_table(otu_table_df, taxa_are_rows = TRUE)

# 5.2 Create sample data object
SAMP <- sample_data(sample_data_df)

# 5.3 Create taxonomy table object
```

```
TAX <- tax_table(tax_table_df)

# ==================== 6. Construct Phyloseq Object ====================
cat("Constructing phyloseq object...\n")
physeq <- phyloseq(OTU, SAMP, TAX)

# ==================== 7. Validation and Summary ====================
cat("\n=== Phyloseq object construction complete! ===\n")
print(physeq)

# Output basic statistical information
cat("\n=== Basic Statistical Information ===\n")
cat("Number of samples:", nsamples(physeq), "\n")
cat("Number of ASVs:", ntaxa(physeq), "\n")
cat("Taxonomic ranks:", paste(rank_names(physeq), collapse = ", "), "\n")
cat("Sample variables:", paste(sample_variables(physeq), collapse = ", "), "\n")

# Check data integrity
cat("\n=== Data Quality Check ===\n")
# Check for empty samples
empty_samples <- sample_sums(physeq) == 0
if (any(empty_samples)) {
  cat("⚠ Found", sum(empty_samples), "empty samples\n")
} else {
  cat("✓ All samples contain sequences\n")
}

# Check for empty ASVs
empty_taxa <- taxa_sums(physeq) == 0
if (any(empty_taxa)) {
  cat("⚠ Found", sum(empty_taxa), "empty ASVs\n")
} else {
  cat("✓ All ASVs contain sequences\n")
}

# Display sample sequencing depth statistics
cat("\n=== Sequencing Depth Statistics ===\n")
seq_depth <- sample_sums(physeq)
cat("Minimum sequencing depth:", min(seq_depth), "\n")
cat("Maximum sequencing depth:", max(seq_depth), "\n")
cat("Average sequencing depth:", round(mean(seq_depth), 0), "\n")
cat("Median sequencing depth:", round(median(seq_depth), 0), "\n")

# ==================== 8. Optional: Save Phyloseq Object ====================
```

```
# Uncomment the following line to save the phyloseq object
# saveRDS(physeq, "phyloseq_object.rds")
# cat("\nPhyloseq object saved as phyloseq_object.rds\n")

# ==================== 9. Optional: Basic Data Cleaning ====================
# Basic data filtering if needed
cat("\n=== Optional Data Cleaning Steps ===\n")
cat("Original data: ", ntaxa(physeq), "ASVs,", nsamples(physeq), "samples\n")

# Remove empty samples and empty ASVs if any
physeq_cleaned <- prune_samples(sample_sums(physeq) > 0, physeq)
physeq_cleaned <- prune_taxa(taxa_sums(physeq_cleaned) > 0, physeq_cleaned)

cat("Cleaned data: ", ntaxa(physeq_cleaned), "ASVs,", nsamples(physeq_cleaned), "samples\n")

# Assign cleaned object to main variable
physeq <- physeq_cleaned

cat("\n🎉 Phyloseq object construction complete! You can start subsequent analyses.\n")
cat("Object name: physeq\n")
cat("Use print(physeq) to view object information\n")

# ==================== 10. Add Phylogenetic Tree to Phyloseq Object ====================
library(ape)

# Set tree file paths
tree_path <- "C:/Users/ASUS/Desktop/imeta/result/sequences.aligned.fasta.treefile"
hash_mapping_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_hash_mapping.csv"

# 10.1 Read and process hash mapping
cat("\nReading ASV hash mapping...\n")
hash_mapping <- read_csv(hash_mapping_path, col_names = c("ASV_id", "Hash"))
hash_mapping$Hash <- trimws(hash_mapping$Hash)    # Clean whitespaces
hash_vec <- setNames(hash_mapping$ASV_id, hash_mapping$Hash)

# 10.2 Read phylogenetic tree
cat("Reading phylogenetic tree...\n")
tree <- read.tree(tree_path)

# 10.3 Check tree-hash correspondence
tree_hashes <- tree$tip.label
mapped_asvs <- hash_vec[tree_hashes]
unmapped_count <- sum(is.na(mapped_asvs))
```

```
if (unmapped_count > 0) {
  cat("⚠ Warning:", unmapped_count, "tree tips not found in hash mapping\n")
  # Remove unmapped tips
  tree <- drop.tip(tree, tree_hashes[is.na(mapped_asvs)])
  mapped_asvs <- na.omit(mapped_asvs)
}

# 10.4 Update tip labels with ASV IDs
tree$tip.label <- as.character(mapped_asvs[tree$tip.label])

# 10.5 Align tree and phyloseq object
# Get overlapping ASVs
common_asvs <- intersect(tree$tip.label, taxa_names(physeq))
physeq_sub <- prune_taxa(common_asvs, physeq)
tree <- keep.tip(tree, common_asvs)

cat("→ Retained", length(common_asvs), "ASVs common to tree and phyloseq\n")

# 10.6 Add tree to phyloseq object
phy_tree(physeq_sub) <- tree

# 10.7 Validate integration
cat("\n=== Tree Integration Validation ===\n")
cat("Tree contains", Ntip(tree), "tips matching ASVs\n")

if (all(taxa_names(physeq_sub) %in% tree$tip.label)) {
  cat("✓ All phyloseq taxa present in tree\n")
} else {
  cat("⚠ Missing taxa: Not all ASVs are in the tree\n")
}

# 10.8 Update phyloseq object
physeq <- phyloseq(otu_table(physeq_sub),
                   sample_data(physeq_sub),
                   tax_table(physeq_sub),
                   phy_tree(physeq_sub))

cat("\n=== Updated Phyloseq Object Summary ===\n")
print(physeq)

# 10.9 Optional: Save updated object
# saveRDS(physeq, "phyloseq_with_tree.rds")
# cat("Saved phyloseq object with tree as 'phyloseq_with_tree.rds'\n")
```

```
cat("\n ✅ Phylogenetic tree successfully integrated!\n")

# ==================== 11. β-Diversity Analysis ====================
# Load required libraries
library(vegan)

# 11.1 Define analysis parameters
factors <- c("Treatment", "Biofilm_Age", "Biofilm_Status", "Place", "Legionella")
distance_methods <- c("bray", "jaccard", "unifrac", "wunifrac")

# 11.2 Create data frames to store results
results_permanova <- data.frame()
results_dispersion <- data.frame()

# 11.3 Main analysis loop
for (dist_method in distance_methods) {
  cat("\n\n=== Calculating", toupper(dist_method), "distance matrix ===\n")

  # Calculate distance matrix (handle UniFrac separately)
  if (dist_method %in% c("unifrac", "wunifrac")) {
    dist_matrix <- phyloseq::distance(physeq, method = dist_method)
  } else {
    dist_matrix <- phyloseq::distance(physeq, method = dist_method, binary = (dist_method ==
"jaccard"))
  }

  # 11.4 Single-factor analysis
  for (factor in factors) {
    cat("\n---- Single-factor PERMANOVA:", factor, "----\n")

    # Perform PERMANOVA
    permanova_res <- adonis2(
      as.formula(paste("dist_matrix ~", factor)),
      data = data.frame(sample_data(physeq)),
      by = "margin",
      permutations = 999
    )

    # Extract results
    temp_df <- data.frame(
      Factor = factor,
      Comparison = "Single-factor",
      Covariable = "None",
      Distance = dist_method,
```

```
      R2 = permanova_res[1, "R2"],
      F_value = permanova_res[1, "F"],
      p_value = permanova_res[1, "Pr(>F)"],
      Significance = ifelse(permanova_res[1, "Pr(>F)"] < 0.05, "*", "")
    )
    results_permanova <- rbind(results_permanova, temp_df)

    # 11.5 Within-group dispersion analysis (Betadisper)
    cat("-- Dispersion analysis --\n")
    dispersion <- betadisper(dist_matrix, group = get_variable(physeq, factor))
    anova_res <- anova(dispersion)
    perm_res <- permutest(dispersion, pairwise = TRUE, permutations = 999)

    disp_df <- data.frame(
      Factor = factor,
      Distance = dist_method,
      F_value = anova_res$`F value`[1],
      p_value = anova_res$`Pr(>F)`[1],
      Significance = ifelse(anova_res$`Pr(>F)`[1] < 0.05, "*", "")
    )
    results_dispersion <- rbind(results_dispersion, disp_df)
  }

  # 11.6 Two-factor analysis (Treatment + Covariate)
  covariates <- c("Biofilm_Age", "Biofilm_Status", "Place", "Legionella")

  for (covar in covariates) {
    cat("\n---- Two-factor PERMANOVA: Treatment +", covar, "----\n")

    # Perform PERMANOVA
    formula_str <- paste("dist_matrix ~ Treatment +", covar)
    permanova_res <- adonis2(
      as.formula(formula_str),
      data = data.frame(sample_data(physeq)),
      by = "margin",
      permutations = 999
    )

    # Extract Treatment results
    temp_df_trt <- data.frame(
      Factor = "Treatment",
      Comparison = "Two-factor",
      Covariable = covar,
      Distance = dist_method,
```

```
            R2 = permanova_res["Treatment", "R2"],
            F_value = permanova_res["Treatment", "F"],
            p_value = permanova_res["Treatment", "Pr(>F)"],
            Significance = ifelse(permanova_res["Treatment", "Pr(>F)"] < 0.05, "*", "")
        )

        # Extract covariate results
        temp_df_covar <- data.frame(
            Factor = covar,
            Comparison = "Two-factor",
            Covariable = "Treatment",
            Distance = dist_method,
            R2 = permanova_res[covar, "R2"],
            F_value = permanova_res[covar, "F"],
            p_value = permanova_res[covar, "Pr(>F)"],
            Significance = ifelse(permanova_res[covar, "Pr(>F)"] < 0.05, "*", "")
        )

        results_permanova <- rbind(results_permanova, temp_df_trt, temp_df_covar)
    }
}

# 11.7 Multiple testing correction
results_permanova$FDR_BH <- p.adjust(results_permanova$p_value, method = "BH")
results_dispersion$FDR_BH <- p.adjust(results_dispersion$p_value, method = "BH")

# Add corrected significance markers
results_permanova$FDR_Significance <- ifelse(results_permanova$FDR_BH < 0.05, "**",
                                             ifelse(results_permanova$p_value < 0.05,
"*", ""))
results_dispersion$FDR_Significance <- ifelse(results_dispersion$FDR_BH < 0.05, "**",
                                              ifelse(results_dispersion$p_value < 0.05,
"*", ""))

# 11.8 Sort results and add metadata
results_permanova <- results_permanova %>%
    arrange(Distance, Factor, Comparison) %>%
    mutate(
        Analysis_Date = Sys.Date(),
        Sample_N = nsamples(physeq),
        ASV_N = ntaxa(physeq)
    ) %>%
    select(Distance, everything())
```

```
results_dispersion <- results_dispersion %>%
  arrange(Distance, Factor) %>%
  mutate(
    Analysis_Date = Sys.Date(),
    Sample_N = nsamples(physeq),
    ASV_N = ntaxa(physeq)
  ) %>%
  select(Distance, everything())

# 11.9 Save results
write_csv(results_permanova, "beta_diversity_permanova_results.csv")
write_csv(results_dispersion, "beta_dispersion_test_results.csv")

# 11.10 Report results
cat("\n\n=== β-Diversity Analysis Completed! ===")
cat("\n→ PERMANOVA results saved to: beta_diversity_permanova_results.csv")
cat("\n→ Dispersion results saved to: beta_dispersion_test_results.csv")
cat("\n→ Number of comparisons performed:", nrow(results_permanova))
cat("\n→ Significant results before FDR:", sum(results_permanova$Significance != ""))
cat("\n→ Significant results after FDR:", sum(results_permanova$FDR_Significance != ""))

#load library
library(ggplot2)

# ==================== 12. Visualize PERMANOVA Results ====================

cat("\n 🎨 Generating visualizations for statistical results...\n")

# Filter for single-factor results for a cleaner plot
permanova_single_factor <- results_permanova %>%
  filter(Comparison == "Single-factor")

# Create the bubble plot
permanova_plot <- ggplot(permanova_single_factor, aes(x = Factor, y = Distance)) +
  # Add points where size is R2 and color is FDR
  geom_point(aes(size = R2, color = FDR_BH), alpha = 0.8, shape = 16) +

  # Add significance stars as text labels
  # vjust is used to vertically adjust the position of the stars
  geom_text(aes(label = FDR_Significance), color = "black", vjust = 0.7, size = 6) +

  # Set the scale for the bubble size for better visibility
  scale_size_continuous(name = "R² (Effect Size)", range = c(4, 12)) +
```

```
  # Set the color gradient for the p-value. Low p-values will be "hotter" (e.g., red).
  scale_color_gradient(name = "FDR (p-value)", low = "#D62728", high = "#1F77B4",
                       trans = "log10", # Using a log scale helps differentiate small p-values
                       breaks = c(0.001, 0.01, 0.05), # Define breaks for the legend
                       labels = c("0.001", "0.01", "0.05")) +

  # Set labels and title
  labs(
    title = "PERMANOVA Single-Factor Analysis Results",
    subtitle = "Bubble size represents effect size (R²), color represents significance",
    x = "Grouping Factor",
    y = "Distance Metric"
  ) +

  # Use a clean theme and adjust text appearance
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12, face = "bold"),
    axis.text.y = element_text(size = 12, face = "bold"),
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5, size = 10),
    legend.position = "right"
  )

# Display the plot
print(permanova_plot)

# Optional: Save the plot to a file
ggsave("permanova_results_bubble_plot.png", plot = permanova_plot, width = 10, height = 8,
dpi = 300)

cat("\n✓ PERMANOVA results plot created.\n")

# ==================== 13. Visualize Dispersion Test Results ====================

dispersion_plot <- ggplot(results_dispersion, aes(x = Factor, y = Distance)) +
  # Here, size is based on the F-value
  geom_point(aes(size = F_value, color = FDR_BH), alpha = 0.8, shape = 16) +

  # Add significance stars
  geom_text(aes(label = FDR_Significance), color = "black", vjust = 0.6, size = 6) +

  # Adjust scales and labels for this plot
  scale_size_continuous(name = "F-value") +
```

```
  scale_color_gradient(name = "FDR (p-value)", low = "#D62728", high = "#1F77B4",
                       trans = "log10",
                       breaks = c(0.001, 0.01, 0.05),
                       labels = c("0.001", "0.01", "0.05")) +

  labs(
    title = "Homogeneity of Dispersion (Betadisper) Results",
    subtitle = "Bubble size represents F-value, color represents significance",
    x = "Grouping Factor",
    y = "Distance Metric"
  ) +

  # Use the same theme for consistency
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12, face = "bold"),
    axis.text.y = element_text(size = 12, face = "bold"),
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5, size = 10),
    legend.position = "right"
  )

# Display the plot
print(dispersion_plot)

# Optional: Save the plot
ggsave("beta_dispersion_results_bubble_plot.png", plot = dispersion_plot, width = 10, height = 8,
dpi = 300)

cat("\n✓ Dispersion test results plot created.\n")

cat("\n 🎉 Analysis and visualization complete! You can now interpret the plots.\n")

#load package
library(ggplot2)
library(ggrepel) # For better text labels

# ==================== 14. Forest Plot for PERMANOVA R² ====================

cat("\n 📊 Generating Forest Plot for R² values...\n")

# Re-use the single-factor results from before
# permanova_single_factor <- results_permanova %>%
#   filter(Comparison == "Single-factor")
```

```
# Create a new variable for plotting on the y-axis and order factors by R²
forest_data <- permanova_single_factor %>%
  mutate(
    Plot_Label = paste(Factor, " (", Distance, ")", sep = ""),
    R2_text = sprintf("%.3f", R2), # Format R² to 3 decimal places
    p_text = ifelse(FDR_BH < 0.001, "< 0.001", sprintf("%.3f", FDR_BH)) # Format p-value
  ) %>%
  arrange(R2) %>%
  mutate(Plot_Label = factor(Plot_Label, levels = Plot_Label)) # Order y-axis by R²

# Create the plot
forest_plot <- ggplot(forest_data, aes(x = R2, y = Plot_Label)) +
  # Add a line segment from 0 to the R² value
  geom_segment(aes(x = 0, xend = R2, y = Plot_Label, yend = Plot_Label), color = "gray") +

  # Add the point for the R² value, colored by the factor
  geom_point(aes(color = Factor), size = 5) +

  # Add text labels for R² and p-values
  geom_text(aes(label = R2_text), color = "black", hjust = -0.5, size = 3.5) + # R² value
  geom_text(aes(label = paste("p =", p_text)), color = "black", hjust = 1.2, size = 3.5) + # p-value

  # Add a vertical line at x=0
  geom_vline(xintercept = 0, linetype = "dashed", color = "darkgray") +

  # Set labels and title
  labs(
    title = "PERMANOVA Effect Size (R²)",
    subtitle = "Factors ordered by their contribution to community variance",
    x = "R² (proportion of variance explained)",
    y = "Factor (Distance Metric)"
  ) +

  # Adjust the x-axis limit to give space for labels and use a clean theme
  scale_x_continuous(expand = expansion(mult = c(0.01, 0.1))) + # Give space on right
  theme_minimal(base_size = 12) +
  theme(
    legend.position = "none", # Hide legend as colors are intuitive
    panel.grid.major.y = element_blank(), # Remove horizontal grid lines
    plot.title = element_text(hjust = 0.5, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5)
  )
```

```
print(forest_plot)

# Optional: Save the plot
ggsave("permanova_forest_plot.png", plot = forest_plot, width = 10, height = 8, dpi = 300)

# Prepare data for plotting
permanova_plot_data <- results_permanova %>%
  # Keep only single-factor comparisons
  filter(Comparison == "Single-factor") %>%
  # Create a significance label based on FDR and raw p-values
  mutate(Significance = case_when(
    FDR_BH < 0.05 ~ "**FDR<0.05**",      # Significant after FDR correction
    p_value < 0.05 ~ "*p<0.05*",          # Significant at α = 0.05 (uncorrected)
    TRUE ~ "ns"                           # Not significant
  ))

# Create a bubble-heatmap hybrid plot
ggplot(permanova_plot_data, aes(x = Distance, y = Factor)) +
  # Draw bubbles whose size and fill both map to R²
  geom_point(aes(size = R2, fill = R2, shape = Significance), alpha = 0.9) +
  # Control bubble-size range
  scale_size_continuous(range = c(3, 10)) +
  # Viridis-like gradient (yellow → teal → purple)
  scale_fill_gradientn(colours = c("#fde725", "#21918c", "#440154")) +
  # Specify shapes for each significance level
  # 23 = filled diamond, 24 = filled triangle, 21 = filled circle
  scale_shape_manual(values = c(23, 24, 21)) +
  theme_minimal() +
  labs(title = "PERMANOVA Results (Single Factor)",
       x = "Distance Metric",
       y = "Environmental Factor",
       size = "R² Value",
       fill = "R² Value",
       shape = "Significance") +
  theme(panel.grid = element_line(color = "grey90"),
        axis.text = element_text(size = 10),
        legend.position = "bottom")

# Prepare data for the dispersion test plot
dispersion_plot_data <- results_dispersion %>%
  # Add a categorical significance label
  mutate(Significance = case_when(
    FDR_BH < 0.05      ~ "FDR < 0.05",    # significant after FDR correction
    p_value < 0.05    ~ "p < 0.05",       # significant at α = 0.05 (raw)
```

```
    TRUE                ~ "ns"              # not significant
  ))

# Create horizontal bar chart
ggplot(dispersion_plot_data, aes(x = reorder(Factor, -p_value), y = -log10(p_value))) +
  # Draw bars colored by significance
  geom_col(aes(fill = Significance), alpha = 0.8) +
  # Split panels by distance metric
  facet_grid(. ~ Distance) +
  # Flip to horizontal bars
  coord_flip() +
  # Add dashed line at -log10(0.05)
  geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "red") +
  labs(title = "Group Dispersion Test Results",
       x = "Environmental Factor",
       y = "-log10(p-value)",
       fill = "Significance") +
  # Manually set bar colors: red (FDR), orange (p<0.05), grey (ns)
  scale_fill_manual(values = c("red", "orange", "grey60")) +
  theme_bw() +
  # Increase spacing between facets
  theme(panel.spacing = unit(1, "lines"))

# Combine PERMANOVA and dispersion-test results
combined_data <- results_permanova %>%
  # Keep single-factor comparisons only
  filter(Comparison == "Single-factor") %>%
  # Select and rename key columns
  select(Factor, Distance,
         PERMANOVA_R2 = R2,
         PERMANOVA_FDR = FDR_BH) %>%
  # Join with dispersion results on Factor and Distance
  left_join(results_dispersion, by = c("Factor", "Distance"))

# Create combined scatter plot
ggplot(combined_data, aes(x = F_value, y = PERMANOVA_R2)) +
  # Plot points: size by √R², color by dispersion p-value
  geom_point(aes(size = sqrt(PERMANOVA_R2), color = -log10(p_value)), alpha = 0.8) +
  # Add factor labels with a slight y-offset; omit overlaps
  geom_text(aes(label = Factor), nudge_y = 0.01, check_overlap = TRUE, size = 3) +
  # Facet by distance metric; allow independent axes
  facet_wrap(~ Distance, scales = "free") +
  labs(title = "PERMANOVA vs. Dispersion Results",
       x = "Dispersion F-value (Betadisper)",
```

```
        y = "PERMANOVA R²",
        size = "Effect Size",
        color = "Dispersion\n-Log10(p)") +
  # Color gradient: blue (high p) → red (low p)
  scale_color_gradient(low = "blue", high = "red") +
  theme_minimal() +
  theme(strip.background = element_rect(fill = "grey90"))

# ==================== 12. Beta-Diversity Visualization ====================

# Load required libraries for plotting
library(ggplot2)
library(ggpubr)
library(patchwork)
library(tidyr)

# --- 12.1 Setup and Preparation ---

# Define the output directory
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/beta-diversity"

# Create the directory if it doesn't exist
if (!dir.exists(output_dir)) {
  dir.create(output_dir, recursive = TRUE)
}

# Define the distance method. "unifrac" for unweighted, "wunifrac" for weighted.
dist_method <- "unifrac"

# Calculate the distance matrix
cat(paste("\nCalculating", dist_method, "distance matrix...\n"))
dist_matrix <- phyloseq::distance(physeq, method = dist_method)

# Perform PERMANOVA on 'Treatment' for PCoA plot annotation
cat("Running PERMANOVA for 'Treatment' factor...\n")
sample_df <- data.frame(sample_data(physeq))
permanova_res <- adonis2(dist_matrix ~ Treatment, data = sample_df, permutations = 999)

# Create a caption string with the PERMANOVA results
permanova_caption <- paste0(
  "PERMANOVA on Treatment\n",
  "R² = ", round(permanova_res$R2[1], 3),
  ", p = ", permanova_res$`Pr(>F)`[1]
)
```

```
# --- 12.2 PCoA Plot 1: Colored by Treatment ---

cat("Generating PCoA plot colored by 'Treatment'...\n")

# Perform PCoA ordination
pcoa_ord <- ordinate(physeq, "PCoA", distance = dist_matrix)

# Create the plot
pcoa_plot_treatment <- plot_ordination(physeq, pcoa_ord, color = "Treatment") +
  theme_bw() +
  geom_point(size = 4, alpha = 0.8) +
  # Add 95% confidence ellipses
  stat_ellipse(type = "t", level = 0.95, aes(group = Treatment)) +
  # Use a nice color palette
  scale_color_brewer(palette = "Set2") +
  # Add titles and the PERMANOVA result as a caption
  labs(
    title = "PCoA Plot (Unweighted UniFrac)",
    subtitle = "Colored by Treatment",
    caption = permanova_caption
  ) +
  # Center the title
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))

# Save the plot
ggsave(
  filename = file.path(output_dir, "pcoa_unifrac_by_treatment.png"),
  plot = pcoa_plot_treatment,
  width = 8,
  height = 6
)

print(pcoa_plot_treatment)
cat("→ Plot saved as pcoa_unifrac_by_treatment.png\n")


# --- 12.3 PCoA Plot 2: Color by Treatment, Shape by Biofilm_Status ---

cat("\nGenerating PCoA plot colored by 'Treatment' and shaped by 'Biofilm_Status'...\n")

# Create the plot
pcoa_plot_treatment_status <- plot_ordination(physeq, pcoa_ord, color = "Treatment", shape =
"Biofilm_Status") +
```

```
  theme_bw() +
  geom_point(size = 4, alpha = 0.8) +
  # Add ellipses for the 'Treatment' groups
  stat_ellipse(type = "t", level = 0.95, aes(group = Treatment)) +
  scale_color_brewer(palette = "Set2") +
  labs(
    title = "PCoA Plot (Unweighted UniFrac)",
    subtitle = "Color: Treatment, Shape: Biofilm Status",
    caption = permanova_caption
  ) +
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle = element_text(hjust = 0.5))

# Save the plot
ggsave(
  filename = file.path(output_dir, "pcoa_unifrac_by_treatment_status.png"),
  plot = pcoa_plot_treatment_status,
  width = 8,
  height = 6
)

print(pcoa_plot_treatment_status)
cat("→ Plot saved as pcoa_unifrac_by_treatment_status.png\n")


# --- 12.4 Distance Boxplot with Statistical Tests ---

cat("\nGenerating distance boxplot with pairwise comparisons...\n")

# A helper function to get the distance data ready for plotting
get_distance_df <- function(dist_matrix, physeq_obj) {
  # Convert distance matrix to a long format data frame
  dist_df <- as.data.frame(as.matrix(dist_matrix))
  dist_df <- dist_df %>%
    tibble::rownames_to_column("Sample1") %>%
    pivot_longer(-Sample1, names_to = "Sample2", values_to = "Distance")

  # Get metadata
  meta_df <- data.frame(sample_data(physeq_obj)) %>%
    tibble::rownames_to_column("sample_id")

  # Join metadata for both samples
  dist_meta <- dist_df %>%
    inner_join(meta_df %>% select(sample_id, Treatment), by = c("Sample1" =
"sample_id")) %>%
```

```
    rename(Group1 = Treatment) %>%
    inner_join(meta_df %>% select(sample_id, Treatment), by = c("Sample2" = "sample_id")) %>%
    rename(Group2 = Treatment)

  # Create a 'Comparison' column and remove duplicates
  dist_comparison <- dist_meta %>%
    filter(as.character(Sample1) < as.character(Sample2)) %>% # Avoid duplicates and self-comparison
    mutate(Comparison = ifelse(
      Group1 == Group2,
      paste("Within", Group1),
      paste(pmin(as.character(Group1), as.character(Group2)), pmax(as.character(Group1), as.character(Group2)), sep = "-vs-")
    )) %>%
    mutate(Comparison = factor(Comparison))

  return(dist_comparison)
}

# Prepare the data for the boxplot
boxplot_df <- get_distance_df(dist_matrix, physeq)

# Define the comparisons for statistical testing
comparison_levels <- levels(boxplot_df$Comparison)
stat_comparisons <- combn(comparison_levels, 2, simplify = FALSE)

# Create the boxplot
distance_boxplot <- ggplot(boxplot_df, aes(x = Comparison, y = Distance, fill = Comparison)) +
  geom_boxplot(alpha = 0.8) +
  # Add pairwise comparisons, showing only significant results
  stat_compare_means(
    comparisons = stat_comparisons,
    method = "wilcox.test",
    label = "p.signif", # Use "p.format" to show p-values
    hide.ns = TRUE      # Hide non-significant comparisons
  ) +
  theme_bw() +
  scale_fill_brewer(palette = "Set3", guide = "none") + # Hide legend, colors are for distinction
  labs(
    title = "Pairwise Unweighted UniFrac Distances",
    subtitle = "Showing distances between and within Treatment groups",
    y = "UniFrac Distance",
    x = ""
```

```
  ) +
  # Rotate x-axis labels for readability
  theme(
    plot.title = element_text(hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 90, hjust = 1)
  )

# Save the plot
ggsave(
  filename = file.path(output_dir, "distance_boxplot_unifrac.png"),
  plot = distance_boxplot,
  width = 10,
  height = 8
)

print(distance_boxplot)
cat("→ Plot saved as distance_boxplot_unifrac.png\n")


# --- 12.5 Combine Plots into a Single Figure ---

cat("\nCombining all plots into a single figure...\n")

# Arrange the three plots using patchwork
# (Plot 1 + Plot 2) on the top row, Boxplot on the bottom row
combined_plot <- (pcoa_plot_treatment + pcoa_plot_treatment_status) / (distance_boxplot) +
  plot_layout(heights = c(1, 1.2)) + # Give more height to the boxplot
  plot_annotation(
    title = 'Beta-Diversity Analysis (Unweighted UniFrac)',
    theme = theme(plot.title = element_text(hjust = 0.5, size = 16, face = "bold"))
  )

# Save the combined plot
ggsave(
  filename = file.path(output_dir, "beta_diversity_combined_plot.png"),
  plot = combined_plot,
  width = 14,
  height = 12
)

# Display the combined plot
print(combined_plot)
```

```
cat("→ Combined plot saved as beta_diversity_combined_plot.png\n")
cat("\n ✅ Visualization complete! All plots are in the 'plots/beta-diversity' folder.\n")
```

The following script is for function analysis

```
# ==================== 1. Load Required R Packages ====================

library(phyloseq)

library(readxl)

library(readr)

library(dplyr)

library(tibble)


# ==================== 2. Set File Paths ====================

metadata_path <- "C:/Users/ASUS/Desktop/imeta/metadata.xlsx"

asv_table_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_table.csv"

taxonomy_path <- "C:/Users/ASUS/Desktop/imeta/result/taxonomy_table.xlsx"


# ==================== 3. Read Data Files ====================


# 3.1 Read sample metadata

cat("Reading sample metadata...\n")

metadata <- read_excel(metadata_path, sheet = 1)

print(paste("Metadata contains", nrow(metadata), "samples"))

print(paste("Metadata columns:", paste(colnames(metadata), collapse = ", ")))


# 3.2 Read ASV abundance table

cat("\nReading ASV abundance table...\n")
```

```
asv_table <- read_csv(asv_table_path)

print(paste("ASV table contains", nrow(asv_table), "ASVs and", ncol(asv_table)-1, "samples"))


# 3.3 Read taxonomic information

cat("\nReading taxonomic information...\n")

taxonomy <- read_excel(taxonomy_path, sheet = 1)

print(paste("Taxonomy table contains", nrow(taxonomy), "ASVs"))

print(paste("Taxonomic ranks:", paste(colnames(taxonomy)[-1], collapse = ", ")))


# ==================== 4. Data Preprocessing ====================


# 4.1 Process sample metadata

# Set sample_id as row names

sample_data_df <- metadata %>%

  column_to_rownames("sample_id")


# 4.2 Process ASV abundance table

# Set ASV_id as row names, convert to matrix

otu_table_df <- asv_table %>%

  column_to_rownames("ASV_id") %>%

  as.matrix()


# Check and ensure consistent sample ordering

cat("\nChecking sample ID consistency...\n")
```

```
metadata_samples <- rownames(sample_data_df)
asv_samples <- colnames(otu_table_df)

if (length(setdiff(metadata_samples, asv_samples)) == 0 &&
      length(setdiff(asv_samples, metadata_samples)) == 0) {
  cat("✓ Sample IDs match completely!\n")
} else {
  cat("⚠ Sample IDs do not match, need to check!\n")
}

# 4.3 Process taxonomic information
# Set ASV_id as row names
tax_table_df <- taxonomy %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check ASV ID consistency
asv_ids_otu <- rownames(otu_table_df)
asv_ids_tax <- rownames(tax_table_df)

if (length(setdiff(asv_ids_otu, asv_ids_tax)) == 0 &&
      length(setdiff(asv_ids_tax, asv_ids_otu)) == 0) {
  cat("✓ ASV IDs match completely!\n")
} else {
```

```
    cat("⚠ ASV IDs do not match, need to check!\n")
}

# ==================== 5. Create Phyloseq Object Components ====================

# 5.1 Create OTU table object
cat("\nCreating phyloseq components...\n")
OTU <- otu_table(otu_table_df, taxa_are_rows = TRUE)

# 5.2 Create sample data object
SAMP <- sample_data(sample_data_df)

# 5.3 Create taxonomy table object
TAX <- tax_table(tax_table_df)

# ==================== 6. Construct Phyloseq Object ====================
cat("Constructing phyloseq object...\n")
physeq <- phyloseq(OTU, SAMP, TAX)

# ==================== 7. Validation and Summary ====================
cat("\n=== Phyloseq object construction complete! ===\n")
print(physeq)

# Output basic statistical information
cat("\n=== Basic Statistical Information ===\n")
cat("Number of samples:", nsamples(physeq), "\n")
```

```
cat("Number of ASVs:", ntaxa(physeq), "\n")

cat("Taxonomic ranks:", paste(rank_names(physeq), collapse = ", "), "\n")

cat("Sample variables:", paste(sample_variables(physeq), collapse = ", "), "\n")


# Check data integrity

cat("\n=== Data Quality Check ===\n")

# Check for empty samples

empty_samples <- sample_sums(physeq) == 0

if (any(empty_samples)) {

   cat("⚠ Found", sum(empty_samples), "empty samples\n")

} else {

   cat("✓ All samples contain sequences\n")

}


# Check for empty ASVs

empty_taxa <- taxa_sums(physeq) == 0

if (any(empty_taxa)) {

   cat("⚠ Found", sum(empty_taxa), "empty ASVs\n")

} else {

   cat("✓ All ASVs contain sequences\n")

}


# Display sample sequencing depth statistics
```

```
cat("\n=== Sequencing Depth Statistics ===\n")

seq_depth <- sample_sums(physeq)

cat("Minimum sequencing depth:", min(seq_depth), "\n")

cat("Maximum sequencing depth:", max(seq_depth), "\n")

cat("Average sequencing depth:", round(mean(seq_depth), 0), "\n")

cat("Median sequencing depth:", round(median(seq_depth), 0), "\n")


# ==================== 8. Optional: Save Phyloseq Object ====================

# Uncomment the following line to save the phyloseq object

# saveRDS(physeq, "phyloseq_object.rds")

# cat("\nPhyloseq object saved as phyloseq_object.rds\n")


# ==================== 9. Optional: Basic Data Cleaning ====================

# Basic data filtering if needed

cat("\n=== Optional Data Cleaning Steps ===\n")

cat("Original data: ", ntaxa(physeq), "ASVs,", nsamples(physeq), "samples\n")


# Remove empty samples and empty ASVs if any

physeq_cleaned <- prune_samples(sample_sums(physeq) > 0, physeq)

physeq_cleaned <- prune_taxa(taxa_sums(physeq_cleaned) > 0, physeq_cleaned)


cat("Cleaned data: ", ntaxa(physeq_cleaned), "ASVs,", nsamples(physeq_cleaned), "samples\n")


# Assign cleaned object to main variable

physeq <- physeq_cleaned
```

```
cat("\n 🎉 Phyloseq object construction complete! You can start subsequent analyses.\n")

cat("Object name: physeq\n")

cat("Use print(physeq) to view object information\n")


# ==================== 10. Add Phylogenetic Tree to Phyloseq Object ====================

library(ape)


# Set tree file paths

tree_path <- "C:/Users/ASUS/Desktop/imeta/result/sequences.aligned.fasta.treefile"

hash_mapping_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_hash_mapping.csv"


# 10.1 Read and process hash mapping

cat("\nReading ASV hash mapping...\n")

hash_mapping <- read_csv(hash_mapping_path, col_names = c("ASV_id", "Hash"))

hash_mapping$Hash <- trimws(hash_mapping$Hash)    # Clean whitespaces

hash_vec <- setNames(hash_mapping$ASV_id, hash_mapping$Hash)


# 10.2 Read phylogenetic tree

cat("Reading phylogenetic tree...\n")

tree <- read.tree(tree_path)


# 10.3 Check tree-hash correspondence

tree_hashes <- tree$tip.label

mapped_asvs <- hash_vec[tree_hashes]
```

```
unmapped_count <- sum(is.na(mapped_asvs))

if (unmapped_count > 0) {
   cat("⚠ Warning:", unmapped_count, "tree tips not found in hash mapping\n")
   # Remove unmapped tips
   tree <- drop.tip(tree, tree_hashes[is.na(mapped_asvs)])
   mapped_asvs <- na.omit(mapped_asvs)
}

# 10.4 Update tip labels with ASV IDs
tree$tip.label <- as.character(mapped_asvs[tree$tip.label])

# 10.5 Align tree and phyloseq object
# Get overlapping ASVs
common_asvs <- intersect(tree$tip.label, taxa_names(physeq))
physeq_sub <- prune_taxa(common_asvs, physeq)
tree <- keep.tip(tree, common_asvs)

cat("→ Retained", length(common_asvs), "ASVs common to tree and phyloseq\n")

# 10.6 Add tree to phyloseq object
phy_tree(physeq_sub) <- tree

# 10.7 Validate integration
cat("\n=== Tree Integration Validation ===\n")
```

```
cat("Tree contains", Ntip(tree), "tips matching ASVs\n")

if (all(taxa_names(physeq_sub) %in% tree$tip.label)) {
  cat("✓ All phyloseq taxa present in tree\n")
} else {
  cat("⚠ Missing taxa: Not all ASVs are in the tree\n")
}

# 10.8 Update phyloseq object
physeq <- phyloseq(otu_table(physeq_sub),
                   sample_data(physeq_sub),
                   tax_table(physeq_sub),
                   phy_tree(physeq_sub))

cat("\n=== Updated Phyloseq Object Summary ===\n")
print(physeq)

# 10.9 Optional: Save updated object
# saveRDS(physeq, "phyloseq_with_tree.rds")
# cat("Saved phyloseq object with tree as 'phyloseq_with_tree.rds'\n")

cat("\n✅ Phylogenetic tree successfully integrated!\n")
# ==================== 0. Environment and Data Preparation ====================
library(phyloseq)
```

```
library(ggplot2)

library(dplyr)

library(tibble)

library(vegan)           # for PERMANOVA (adonis2)

library(DESeq2)          # for differential abundance analysis

library(pheatmap)      # for drawing heatmaps


# --- Define file paths ---

pathway_path                                                    <-
"C:/Users/ASUS/Desktop/imeta/result/result/pathways_out/path_abun_unstrat.tsv.gz"

ec_path                                                         <-
"C:/Users/ASUS/Desktop/imeta/result/result/EC_metagenome_out/pred_metagenome_unstrat.t
sv.gz"

ko_path                                                         <-
"C:/Users/ASUS/Desktop/imeta/result/result/KO_metagenome_out/pred_metagenome_unstrat.
tsv.gz"


# --- Load sample metadata (assuming you already have an original object named physeq) ---

SAMP <- sample_data(physeq)


# --- Function: load data and create phyloseq objects (to avoid code duplication) ---

create_func_phyloseq <- function(file_path, sample_data) {

   abundance    <- read.delim(gzfile(file_path), row.names = 1)

   matrix         <- as.matrix(abundance)

   OTU              <- otu_table(matrix, taxa_are_rows = TRUE)

   # Ensure consistent sample order
```

```
  OTU              <- OTU[, sample_names(sample_data)]
  physeq_obj <- phyloseq(OTU, sample_data)
  return(physeq_obj)
}

# --- Create phyloseq objects for three functional levels ---
physeq_pathway <- create_func_phyloseq(pathway_path, SAMP)
physeq_ec          <- create_func_phyloseq(ec_path,          SAMP)
physeq_ko          <- create_func_phyloseq(ko_path,          SAMP)

cat("All three functional-level phyloseq objects created successfully!\n")
cat("Object names: physeq_pathway, physeq_ec, physeq_ko\n")

# ==================== Legionella Presence vs Absence Functional Difference Analysis (ANCOM-BC) ====================
# Fixed version with proper feature ID handling for heatmap generation

# Check and install ANCOMBC package
if (!requireNamespace("ANCOMBC", quietly = TRUE)) {
  cat("Installing ANCOMBC package...\n")
  if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
  }
  BiocManager::install("ANCOMBC")
}

library(phyloseq)
library(ANCOMBC)
library(ggplot2)
library(dplyr)
library(tibble)
library(pheatmap)
library(RColorBrewer)
library(VennDiagram)
library(ggrepel)
```

```
library(tidyr)

# Set output path
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/function desq2"
if (!dir.exists(output_dir)) {
  dir.create(output_dir, recursive = TRUE)
}

# ==================== 1. Data Preprocessing and Sample Filtering ====================

# Check values in Legionella column
cat("Checking Legionella column values:\n")
legionella_status <- sample_data(physeq)$Legionella
print(table(legionella_status, useNA = "ifany"))

# Filter valid samples (remove NA values)
valid_samples <- !is.na(sample_data(physeq)$Legionella)
physeq_filtered <- prune_samples(valid_samples, physeq)
physeq_pathway_filtered <- prune_samples(valid_samples, physeq_pathway)
physeq_ec_filtered <- prune_samples(valid_samples, physeq_ec)
physeq_ko_filtered <- prune_samples(valid_samples, physeq_ko)

cat("Number of samples after filtering:\n")
cat("Total samples:", nsamples(physeq_filtered), "\n")
cat("Presence:", sum(sample_data(physeq_filtered)$Legionella == "Presence"), "\n")
cat("Absence:", sum(sample_data(physeq_filtered)$Legionella == "Absence"), "\n")

# ==================== 2. Define Enhanced ANCOM-BC Differential Analysis Function ====================

perform_ancombc_analysis <- function(physeq_obj, comparison_name, prevalence_threshold = 0.05) {

  cat("\n=== Starting", comparison_name, "ANCOM-BC Differential Analysis ===\n")

  # Enhanced filtering: remove low-abundance and zero-variance features
  total_samples <- nsamples(physeq_obj)
  min_prevalence_samples <- max(2, ceiling(total_samples * prevalence_threshold))

  # Step 1: Basic prevalence filtering
  physeq_filtered <- filter_taxa(physeq_obj, function(x) {
    sum(x > 0) >= min_prevalence_samples && sum(x) > 0
  }, TRUE)
```

```
# Step 2: Remove features with very low total abundance
total_counts <- taxa_sums(physeq_filtered)
min_total_count <- max(10, total_samples)
physeq_filtered <- prune_taxa(total_counts >= min_total_count, physeq_filtered)

# Step 3: Check for and remove zero-variance features
cat("Checking for zero-variance features...\n")
otu_matrix <- as.matrix(otu_table(physeq_filtered))

# Calculate variance for each feature
feature_vars <- apply(otu_matrix, 1, function(x) {
  if (all(x == 0)) return(0)
  if (length(unique(x)) == 1) return(0)
  return(var(x, na.rm = TRUE))
})

# Identify problematic features
zero_var_features <- names(feature_vars)[feature_vars == 0 | is.na(feature_vars)]

if (length(zero_var_features) > 0) {
  cat("Found", length(zero_var_features), "zero-variance features, removing them:\n")
  cat("Zero-variance features:", paste(head(zero_var_features, 10), collapse = ", "),
      ifelse(length(zero_var_features) > 10, "...", ""), "\n")

  # Remove zero-variance features
  keep_features <- setdiff(taxa_names(physeq_filtered), zero_var_features)
  physeq_filtered <- prune_taxa(keep_features, physeq_filtered)
} else {
  cat("No zero-variance features detected.\n")
}

cat("Original number of features:", ntaxa(physeq_obj), "\n")
cat("After filtering:", ntaxa(physeq_filtered), "\n")

# Final check: ensure we have enough features for analysis
if (ntaxa(physeq_filtered) < 10) {
  warning("Very few features remaining after filtering (", ntaxa(physeq_filtered),
          "). Consider relaxing filtering criteria.")
}

cat("Final number of features for analysis:", ntaxa(physeq_filtered), "\n")
cat("Final number of samples for analysis:", nsamples(physeq_filtered), "\n")

# Run ANCOM-BC analysis
```

```
tryCatch({
    ancombc_result <- ancombc2(
        data = physeq_filtered,
        assay_name = "counts",
        tax_level = NULL,
        fix_formula = "Legionella",
        rand_formula = NULL,
        p_adj_method = "fdr",
        pseudo_sens = FALSE,
        prv_cut = prevalence_threshold,
        lib_cut = 100,
        s0_perc = 0.05,
        group = "Legionella",
        struc_zero = FALSE,
        neg_lb = FALSE,
        alpha = 0.05,
        n_cl = 1,
        verbose = TRUE
    )

    # Check and extract results
    cat("ANCOM-BC result structure check:\n")
    print(names(ancombc_result$res))

    # Extract results - CRITICAL FIX: use actual taxa names
    res_data <- ancombc_result$res

    # Get the actual taxa names from the phyloseq object used in analysis
    actual_taxa_names <- taxa_names(physeq_filtered)

    # Find correct column names
    lfc_cols <- grep("lfc", names(res_data), value = TRUE)
    pval_cols <- grep("p_", names(res_data), value = TRUE)
    qval_cols <- grep("q_", names(res_data), value = TRUE)

    cat("Found column names:\n")
    cat("Log fold change columns:", paste(lfc_cols, collapse = ", "), "\n")
    cat("P-value columns:", paste(pval_cols, collapse = ", "), "\n")
    cat("Adjusted P-value columns:", paste(qval_cols, collapse = ", "), "\n")

    # Build results data frame with actual feature names
    results_df <- data.frame(
        Feature_ID = actual_taxa_names,    # Use actual taxa names instead of row indices
        stringsAsFactors = FALSE
```

```
)

# Extract data based on actual column names
if (length(lfc_cols) > 0) {
   # Use the Legionella-specific column
   lfc_col <- grep("LegionellaPresence", lfc_cols, value = TRUE)[1]
   if (!is.na(lfc_col)) {
      results_df$log2FoldChange <- res_data[[lfc_col]]
   } else {
      results_df$log2FoldChange <- res_data[[lfc_cols[1]]]
   }
} else {
   stop("Could not find log fold change column")
}

if (length(pval_cols) > 0) {
   # Use the Legionella-specific column
   pval_col <- grep("LegionellaPresence", pval_cols, value = TRUE)[1]
   if (!is.na(pval_col)) {
      results_df$pval <- res_data[[pval_col]]
   } else {
      results_df$pval <- res_data[[pval_cols[1]]]
   }
} else {
   results_df$pval <- NA
}

if (length(qval_cols) > 0) {
   # Use the Legionella-specific column
   qval_col <- grep("LegionellaPresence", qval_cols, value = TRUE)[1]
   if (!is.na(qval_col)) {
      results_df$padj <- res_data[[qval_col]]
   } else {
      results_df$padj <- res_data[[qval_cols[1]]]
   }
} else {
   results_df$padj <- NA
}

# Add significance marker
results_df$significant <- results_df$padj < 0.05 & !is.na(results_df$padj)

# Remove NA rows and sort
results_df <- results_df %>%
```

```
      filter(!is.na(log2FoldChange) & !is.na(padj)) %>%
      arrange(padj)

    # Count significant differential features
    sig_up <- sum(results_df$log2FoldChange > 0 & results_df$padj < 0.05, na.rm = TRUE)
    sig_down <- sum(results_df$log2FoldChange < 0 & results_df$padj < 0.05, na.rm = TRUE)

    cat("Number of significantly upregulated features (Presence > Absence):", sig_up, "\n")
    cat("Number of significantly downregulated features (Presence < Absence):", sig_down,
"\n")

    return(list(
      results = results_df,
      ancombc_obj = ancombc_result,
      physeq = physeq_filtered,
      sig_up = sig_up,
      sig_down = sig_down
    ))

  }, error = function(e) {
    cat("Error in ANCOM-BC analysis:", e$message, "\n")

    # Additional zero-variance checking if not done already
    cat("Performing additional zero-variance filtering before retry...\n")
    otu_matrix_check <- as.matrix(otu_table(physeq_filtered))
    feature_vars_check <- apply(otu_matrix_check, 1, function(x) {
      if (all(x == 0)) return(0)
      if (length(unique(x)) == 1) return(0)
      return(var(x, na.rm = TRUE))
    })
    zero_var_features_check <- names(feature_vars_check)[feature_vars_check == 0 |
is.na(feature_vars_check)]

    if (length(zero_var_features_check) > 0) {
      cat("Additional zero-variance features found, removing them:\n")
      cat("Features:", paste(head(zero_var_features_check, 10), collapse = ", "),
          ifelse(length(zero_var_features_check) > 10, "...", ""), "\n")

      keep_features_retry <- setdiff(taxa_names(physeq_filtered), zero_var_features_check)
      physeq_filtered <- prune_taxa(keep_features_retry, physeq_filtered)
    }

    cat("Trying with more conservative parameters...\n")
```

```
# Retry with more conservative parameters
ancombc_result <- ancombc2(
    data = physeq_filtered,
    assay_name = "counts",
    tax_level = NULL,
    fix_formula = "Legionella",
    rand_formula = NULL,
    p_adj_method = "fdr",
    pseudo_sens = FALSE,
    prv_cut = max(0.2, prevalence_threshold),
    lib_cut = 50,
    s0_perc = 0.1,
    group = "Legionella",
    struc_zero = FALSE,
    neg_lb = FALSE,
    alpha = 0.05,
    n_cl = 1,
    verbose = TRUE
)

# Extract results with actual taxa names
res_data <- ancombc_result$res
actual_taxa_names <- taxa_names(physeq_filtered)

# Find correct column names
lfc_cols <- grep("lfc", names(res_data), value = TRUE)
pval_cols <- grep("p_", names(res_data), value = TRUE)
qval_cols <- grep("q_", names(res_data), value = TRUE)

# Build results data frame with actual feature names
results_df <- data.frame(
    Feature_ID = actual_taxa_names,
    stringsAsFactors = FALSE
)

# Extract data
if (length(lfc_cols) > 0) {
    lfc_col <- grep("LegionellaPresence", lfc_cols, value = TRUE)[1]
    if (!is.na(lfc_col)) {
        results_df$log2FoldChange <- res_data[[lfc_col]]
    } else {
        results_df$log2FoldChange <- res_data[[lfc_cols[1]]]
    }
} else {
```

```
        stop("Could not find log fold change column")
    }

    if (length(pval_cols) > 0) {
        pval_col <- grep("LegionellaPresence", pval_cols, value = TRUE)[1]
        if (!is.na(pval_col)) {
            results_df$pval <- res_data[[pval_col]]
        } else {
            results_df$pval <- res_data[[pval_cols[1]]]
        }
    } else {
        results_df$pval <- NA
    }

    if (length(qval_cols) > 0) {
        qval_col <- grep("LegionellaPresence", qval_cols, value = TRUE)[1]
        if (!is.na(qval_col)) {
            results_df$padj <- res_data[[qval_col]]
        } else {
            results_df$padj <- res_data[[qval_cols[1]]]
        }
    } else {
        results_df$padj <- NA
    }

    # Add significance marker
    results_df$significant <- results_df$padj < 0.05 & !is.na(results_df$padj)

    # Remove NA rows and sort
    results_df <- results_df %>%
        filter(!is.na(log2FoldChange) & !is.na(padj)) %>%
        arrange(padj)

    # Count significant differential features
    sig_up <- sum(results_df$log2FoldChange > 0 & results_df$padj < 0.05, na.rm = TRUE)
    sig_down <- sum(results_df$log2FoldChange < 0 & results_df$padj < 0.05, na.rm = TRUE)

    cat("Number of significantly upregulated features (Presence > Absence):", sig_up, "\n")
    cat("Number of significantly downregulated features (Presence < Absence):", sig_down,
"\n")

    return(list(
        results = results_df,
        ancombc_obj = ancombc_result,
```

```
        physeq = physeq_filtered,
        sig_up = sig_up,
        sig_down = sig_down
      ))
  })
}

# ==================== 3. Perform Differential Analysis at Three Levels ====================

# EC level analysis
ec_analysis <- perform_ancombc_analysis(physeq_ec_filtered, "EC")

# KO level analysis
ko_analysis <- perform_ancombc_analysis(physeq_ko_filtered, "KO")

# Pathway level analysis
pathway_analysis <- perform_ancombc_analysis(physeq_pathway_filtered, "Pathway")

# ==================== 4. Save Differential Analysis Results ====================

# Save detailed results
write.csv(ec_analysis$results, file.path(output_dir, "Legionella_EC_ANCOMBC_results.csv"), row.names = FALSE)
write.csv(ko_analysis$results, file.path(output_dir, "Legionella_KO_ANCOMBC_results.csv"), row.names = FALSE)
write.csv(pathway_analysis$results, file.path(output_dir, "Legionella_Pathway_ANCOMBC_results.csv"), row.names = FALSE)

# Save significant differential features
sig_threshold <- 0.05
lfc_threshold <- 0.5    # ANCOM-BC typically uses smaller thresholds

for (analysis in list(
  list(data = ec_analysis, name = "EC"),
  list(data = ko_analysis, name = "KO"),
  list(data = pathway_analysis, name = "Pathway")
)) {

  sig_features <- analysis$data$results %>%
    filter(padj < sig_threshold & abs(log2FoldChange) > lfc_threshold)

  write.csv(sig_features,
            file.path(output_dir, paste0("Legionella_", analysis$name,
```

```
"_significant_features.csv")),
                    row.names = FALSE)
}

# ==================== 5. Visualization Analysis ====================

# 5.1 Create volcano plot function
create_volcano_plot <- function(results_df, title, filename) {

  # Add significance labels
  results_df <- results_df %>%
    mutate(
      Significance = case_when(
        padj < 0.05 & log2FoldChange > 0.5 ~ "Up in Presence",
        padj < 0.05 & log2FoldChange < -0.5 ~ "Up in Absence",
        TRUE ~ "Not Significant"
      )
    )

  # Select features for labeling (top 10 most significant)
  top_features <- results_df %>%
    filter(Significance != "Not Significant") %>%
    arrange(padj) %>%
    head(10)

  p <- ggplot(results_df, aes(x = log2FoldChange, y = -log10(padj))) +
    geom_point(aes(color = Significance), alpha = 0.7, size = 1.5) +
    scale_color_manual(values = c(
      "Up in Presence" = "#d62728",
      "Up in Absence" = "#2ca02c",
      "Not Significant" = "grey70"
    )) +
    geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "black", alpha = 0.5) +
    geom_vline(xintercept = c(-0.5, 0.5), linetype = "dashed", color = "black", alpha = 0.5) +
    labs(
      title = paste("Volcano Plot -", title),
      subtitle = "Legionella Presence vs Absence (ANCOM-BC)",
      x = "log2 Fold Change (Presence/Absence)",
      y = "-log10(adjusted p-value)",
      color = "Significance"
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
```

```
        plot.subtitle = element_text(hjust = 0.5, size = 12),
        legend.position = "bottom"
      )

  # Add labels (if there are significant features)
  if (nrow(top_features) > 0) {
    p <- p + geom_text_repel(
      data = top_features,
      aes(label = Feature_ID),
      size = 3,
      max.overlaps = 10,
      box.padding = 0.3
    )
  }

  ggsave(filename, plot = p, width = 12, height = 8, dpi = 300)
  return(p)
}

# 5.2 Create lollipop plot function
create_lollipop_plot <- function(results_df, title, filename, top_n = 20) {

  # Select significant features
  sig_features <- results_df %>%
    filter(padj < 0.05) %>%
    arrange(padj) %>%
    head(top_n)

  if (nrow(sig_features) == 0) {
    cat("No significant features for lollipop plot:", title, "\n")
    return(NULL)
  }

  # Prepare data
  plot_data <- sig_features %>%
    mutate(
      Direction = ifelse(log2FoldChange > 0, "Up in Presence", "Up in Absence"),
      Feature_ID_short = ifelse(nchar(Feature_ID) > 40,
                                paste0(substr(Feature_ID, 1, 37), "..."),
                                Feature_ID),
      abs_lfc = abs(log2FoldChange)
    ) %>%
    arrange(abs_lfc)
```

```
  # Set factor levels
  plot_data$Feature_ID_short <- factor(plot_data$Feature_ID_short,
                                       levels = plot_data$Feature_ID_short)

  p <- ggplot(plot_data, aes(x = Feature_ID_short, y = log2FoldChange)) +
    geom_segment(aes(x = Feature_ID_short, xend = Feature_ID_short,
                     y = 0, yend = log2FoldChange, color = Direction),
                 size = 1) +
    geom_point(aes(color = Direction, size = -log10(padj)), alpha = 0.8) +
    scale_color_manual(values = c("Up in Presence" = "#d62728",
                                  "Up in Absence" = "#2ca02c")) +
    scale_size_continuous(name = "-log10(padj)", range = c(2, 8)) +
    coord_flip() +
    labs(
      title = paste("Top Significant Features -", title),
      subtitle = "Legionella Presence vs Absence (ANCOM-BC)",
      x = "Functional Features",
      y = "log2 Fold Change (Presence/Absence)",
      color = "Direction",
      size = "Significance"
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      plot.subtitle = element_text(hjust = 0.5, size = 12),
      axis.text.y = element_text(size = 9),
      legend.position = "bottom"
    ) +
    geom_hline(yintercept = 0, linetype = "solid", color = "black", alpha = 0.3)

  ggsave(filename, plot = p, width = 14, height = 10, dpi = 300)
  return(p)
}

# 5.3 Create FIXED heatmap function
create_heatmap <- function(physeq_obj, results_df, title, filename, top_n = 30) {

  cat("\n=== Creating heatmap:", title, "===\n")

  # Check if there are results data
  if (nrow(results_df) == 0) {
    cat("No results data for heatmap:", title, "\n")
    return(NULL)
  }
```

```
# Select top significant features
top_features <- results_df %>%
  filter(!is.na(padj)) %>%
  arrange(padj) %>%
  head(top_n)

if (nrow(top_features) == 0) {
  cat("No significant features for heatmap:", title, "\n")
  return(NULL)
}

cat("Number of candidate features:", nrow(top_features), "\n")

# Check if feature IDs exist in phyloseq object
available_taxa <- taxa_names(physeq_obj)
feature_ids <- top_features$Feature_ID

cat("Number of taxa in phyloseq object:", length(available_taxa), "\n")
cat("Number of candidate feature IDs:", length(feature_ids), "\n")

# CRITICAL FIX: Now the feature IDs should match directly
valid_features <- intersect(feature_ids, available_taxa)
cat("Number of matching features:", length(valid_features), "\n")

if (length(valid_features) == 0) {
  cat("Warning: No matching feature IDs, checking first few feature IDs:\n")
  cat("First 5 feature IDs in results:\n")
  print(head(feature_ids, 5))
  cat("First 5 taxa in phyloseq:\n")
  print(head(available_taxa, 5))
  return(NULL)
}

# Limit feature count to avoid oversized images
if (length(valid_features) > top_n) {
  valid_features <- valid_features[1:top_n]
}

cat("Final number of features for plotting:", length(valid_features), "\n")

tryCatch({
  # Extract abundance data
  otu_table_subset <- otu_table(physeq_obj)[valid_features, , drop = FALSE]
```

```
otu_df <- as.data.frame(otu_table_subset)

cat("Extracted data dimensions:", dim(otu_df), "\n")

# Check if data is empty
if (nrow(otu_df) == 0 || ncol(otu_df) == 0) {
  cat("Extracted data is empty, cannot create heatmap\n")
  return(NULL)
}

# Convert to relative abundance (avoid division by zero)
col_sums <- colSums(otu_df)
if (any(col_sums == 0)) {
  cat("Some samples have total abundance of 0, adding pseudo-counts\n")
  otu_df <- otu_df + 1
  col_sums <- colSums(otu_df)
}

otu_rel <- sweep(otu_df, 2, col_sums, "/") * 100

# Log transformation and normalization (z-score)
otu_log <- log10(otu_rel + 1e-6)
otu_scaled <- t(scale(t(otu_log)))

# Handle NaN values
otu_scaled[is.nan(otu_scaled)] <- 0
otu_scaled[is.infinite(otu_scaled)] <- 0

# Simplify feature names
rownames(otu_scaled) <- ifelse(nchar(rownames(otu_scaled)) > 50,
                               paste0(substr(rownames(otu_scaled), 1, 47), "..."),
                               rownames(otu_scaled))

# Prepare sample annotations
sample_anno <- data.frame(
  Legionella = sample_data(physeq_obj)$Legionella,
  row.names = sample_names(physeq_obj)
)

# Ensure sample_anno row names match otu_scaled column names
common_samples <- intersect(rownames(sample_anno), colnames(otu_scaled))
sample_anno <- sample_anno[common_samples, , drop = FALSE]
otu_scaled <- otu_scaled[, common_samples, drop = FALSE]
```

```
    cat("Final plotting data dimensions:", dim(otu_scaled), "\n")
    cat("Sample annotation dimensions:", dim(sample_anno), "\n")

    # Color settings
    anno_colors <- list(
      Legionella = c("Presence" = "#d62728", "Absence" = "#2ca02c")
    )

    # Create heatmap
    pheatmap(
      otu_scaled,
      annotation_col = sample_anno,
      annotation_colors = anno_colors,
      clustering_distance_rows = "correlation",
      clustering_distance_cols = "correlation",
      show_rownames = TRUE,
      show_colnames = TRUE,    # Show sample names
      scale = "none",
      color = colorRampPalette(c("navy", "white", "firebrick"))(100),
      filename = filename,
      width = 16,
      height = max(8, nrow(otu_scaled) * 0.3),    # Dynamically adjust height
      fontsize_col = 8,    # Adjust sample name font size
      fontsize_row = 8,    # Adjust feature name font size
      angle_col = 45,        # Tilt sample names for better readability
      main = paste("Legionella Presence vs Absence -", title, "Functions")
    )

    cat("Heatmap saved to:", filename, "\n")

  }, error = function(e) {
    cat("Error while creating heatmap:", e$message, "\n")
    cat("Trying to create simplified version of heatmap...\n")

    # Simplified version: use only first 10 features
    simple_features <- valid_features[1:min(10, length(valid_features))]

    tryCatch({
      otu_simple <- as.data.frame(otu_table(physeq_obj)[simple_features, , drop = FALSE])
      otu_simple_rel <- sweep(otu_simple, 2, colSums(otu_simple + 1), "/") * 100

      pheatmap(
        log10(otu_simple_rel + 1),
        show_rownames = TRUE,
```

```
            show_colnames = TRUE,
            filename = filename,
            width = 12,
            height = 8,
            fontsize_col = 8,
            angle_col = 45,
            main = paste("Legionella Presence vs Absence -", title, "(Simplified)")
          )

          cat("Simplified heatmap saved to:", filename, "\n")

        }, error = function(e2) {
          cat("Simplified heatmap also failed:", e2$message, "\n")
          return(NULL)
        })
    })
}

# Create all visualization charts
# Volcano plots
volcano_ec <- create_volcano_plot(
    ec_analysis$results,
    "EC Functions",
    file.path(output_dir, "Legionella_EC_volcano_plot.png")
)

volcano_ko <- create_volcano_plot(
    ko_analysis$results,
    "KO Functions",
    file.path(output_dir, "Legionella_KO_volcano_plot.png")
)

volcano_pathway <- create_volcano_plot(
    pathway_analysis$results,
    "Pathway Functions",
    file.path(output_dir, "Legionella_Pathway_volcano_plot.png")
)

# Lollipop plots
lollipop_ec <- create_lollipop_plot(
    ec_analysis$results,
    "EC Functions",
    file.path(output_dir, "Legionella_EC_lollipop_plot.png")
)
```

```
lollipop_ko <- create_lollipop_plot(
  ko_analysis$results,
  "KO Functions",
  file.path(output_dir, "Legionella_KO_lollipop_plot.png")
)

lollipop_pathway <- create_lollipop_plot(
  pathway_analysis$results,
  "Pathway Functions",
  file.path(output_dir, "Legionella_Pathway_lollipop_plot.png")
)

# Fixed Heatmaps
create_heatmap(
  ec_analysis$physeq,
  ec_analysis$results,
  "EC Functions",
  file.path(output_dir, "Legionella_EC_heatmap.png")
)

create_heatmap(
  ko_analysis$physeq,
  ko_analysis$results,
  "KO Functions",
  file.path(output_dir, "Legionella_KO_heatmap.png")
)

create_heatmap(
  pathway_analysis$physeq,
  pathway_analysis$results,
  "Pathway Functions",
  file.path(output_dir, "Legionella_Pathway_heatmap.png")
)

# 5.4 Create summary chart
summary_data <- data.frame(
  Level = c("EC", "KO", "Pathway"),
  Up_in_Presence = c(ec_analysis$sig_up, ko_analysis$sig_up, pathway_analysis$sig_up),
  Up_in_Absence = c(ec_analysis$sig_down, ko_analysis$sig_down,
pathway_analysis$sig_down)
) %>%
  pivot_longer(cols = c(Up_in_Presence, Up_in_Absence),
               names_to = "Direction", values_to = "Count")
```

```
summary_plot <- ggplot(summary_data, aes(x = Level, y = Count, fill = Direction)) +
  geom_bar(stat = "identity", position = "dodge", alpha = 0.8) +
  scale_fill_manual(values = c("Up_in_Presence" = "#d62728", "Up_in_Absence" = "#2ca02c")) +
  labs(
    title = "Number of Significantly Different Functions",
    subtitle = "Legionella Presence vs Absence (ANCOM-BC, padj < 0.05)",
    x = "Functional Level",
    y = "Number of Features",
    fill = "Direction"
  ) +
  theme_bw() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5, size = 12)
  ) +
  geom_text(aes(label = Count), position = position_dodge(width = 0.9), vjust = -0.3)

ggsave(file.path(output_dir, "Legionella_functional_summary.png"),
       plot = summary_plot, width = 10, height = 6, dpi = 300)

# ==================== 6. Generate Analysis Report ====================

# Create summary report
summary_report <- paste(
  "Legionella Presence vs Absence Functional Difference Analysis Report (ANCOM-BC)",
  paste(rep("=", 60), collapse = ""),
  "",
  "Analysis Date:", Sys.Date(),
  "Analysis Method: ANCOM-BC (Enhanced with Zero-Variance Filtering)",
  "",
  "Sample Information:",
  paste("  Total Samples:", nsamples(physeq_filtered)),
  paste("  Presence Group:", sum(sample_data(physeq_filtered)$Legionella == "Presence")),
  paste("  Absence Group:", sum(sample_data(physeq_filtered)$Legionella == "Absence")),
  "",
  "Number of Significantly Different Features (padj < 0.05):",
  paste("  EC Level: ", ec_analysis$sig_up + ec_analysis$sig_down,
        " (Upregulated:", ec_analysis$sig_up, ", Downregulated:", ec_analysis$sig_down, ")"),
  paste("  KO Level: ", ko_analysis$sig_up + ko_analysis$sig_down,
        " (Upregulated:", ko_analysis$sig_up, ", Downregulated:", ko_analysis$sig_down, ")"),
  paste("  Pathway Level: ", pathway_analysis$sig_up + pathway_analysis$sig_down,
        "  (Upregulated:", pathway_analysis$sig_up, ", Downregulated:",
pathway_analysis$sig_down, ")"),
```

```
  "",
  "Output Files:",
  "    1. Differential Analysis Results: *_ANCOMBC_results.csv",
  "    2. Significant Feature Lists: *_significant_features.csv",
  "    3. Volcano Plots: *_volcano_plot.png",
  "    4. Lollipop Plots: *_lollipop_plot.png",
  "    5. Heatmaps: *_heatmap.png (FIXED)",
  "    6. Summary Figure: Legionella_functional_summary.png",
  "",
  "Notes:",
  "- Fixed feature ID matching issue in heatmap generation",
  "- Enhanced data filtering removes zero-variance and low-abundance features",
  "- Upregulated indicates higher abundance in Presence group",
  "- Downregulated indicates higher abundance in Absence group",
  "- ANCOM-BC method accounts for the compositional nature of microbiome data",
  "- Heatmaps display sample names to facilitate tracking of individual samples",
  "- Lollipop plots show the most significant functional features",
  "- Conservative parameters used to ensure statistical robustness",
  sep = "\n"
)

writeLines(summary_report, file.path(output_dir, "Legionella_analysis_summary.txt"))

cat("\n=== FIXED ANCOM-BC Analysis Completed ===\n")
cat("All results saved to:", output_dir, "\n")
cat("Heatmaps should now generate correctly!\n")
cat("Please refer to analysis_summary.txt for detailed analysis report\n")

# ==================== Biofilm Status Functional Difference Analysis (ANCOM-BC) ====================
# Analysis 2: Biofilm maturity (Old, Young, Mature) in Untreated samples
# Enhanced version with robust zero-variance filtering and small sample handling
# Fixed version that addresses zero-variance errors and small sample size issues

# Load required packages
library(phyloseq)
library(ANCOMBC)
library(ggplot2)
library(dplyr)
library(tibble)
library(pheatmap)
library(RColorBrewer)
library(VennDiagram)
library(ggrepel)
```

```
library(tidyr)

# Set output path
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/function desq2"
if (!dir.exists(output_dir)) {
  dir.create(output_dir, recursive = TRUE)
}

# ==================== 1. Data Preprocessing and Sample Filtering ====================

# First filter for Untreated samples
cat("Checking Treatment column values:\n")
treatment_status <- sample_data(physeq)$Treatment
print(table(treatment_status, useNA = "ifany"))

# Filter Untreated samples
untreated_samples <- sample_data(physeq)$Treatment == "Untreated"
& !is.na(sample_data(physeq)$Treatment)
physeq_untreated <- prune_samples(untreated_samples, physeq)
physeq_pathway_untreated <- prune_samples(untreated_samples, physeq_pathway)
physeq_ec_untreated <- prune_samples(untreated_samples, physeq_ec)
physeq_ko_untreated <- prune_samples(untreated_samples, physeq_ko)

cat("Number of Untreated samples:", nsamples(physeq_untreated), "\n")

# Check Biofilm_Status values in untreated samples
cat("\nChecking Biofilm_Status column values in Untreated samples:\n")
biofilm_status <- sample_data(physeq_untreated)$Biofilm_Status
print(table(biofilm_status, useNA = "ifany"))

# Filter valid biofilm status samples (remove NA values)
valid_biofilm <- !is.na(sample_data(physeq_untreated)$Biofilm_Status)
physeq_biofilm <- prune_samples(valid_biofilm, physeq_untreated)
physeq_pathway_biofilm <- prune_samples(valid_biofilm, physeq_pathway_untreated)
physeq_ec_biofilm <- prune_samples(valid_biofilm, physeq_ec_untreated)
physeq_ko_biofilm <- prune_samples(valid_biofilm, physeq_ko_untreated)

cat("\nFinal sample counts after filtering:\n")
cat("Total samples:", nsamples(physeq_biofilm), "\n")
biofilm_counts <- table(sample_data(physeq_biofilm)$Biofilm_Status)
print(biofilm_counts)

# ==================== 2. Define ANCOM-BC Pairwise Comparison Function
====================
```

```
perform_pairwise_ancombc <- function(physeq_obj, group1, group2, comparison_name, prevalence_threshold = 0.2) {  # Increased default threshold

  cat("\n=== Starting", comparison_name, "ANCOM-BC Analysis ===\n")

  # Filter samples for specific comparison
  target_groups <- c(group1, group2)
  group_samples <- sample_data(physeq_obj)$Biofilm_Status %in% target_groups
  physeq_subset <- prune_samples(group_samples, physeq_obj)

  cat("Samples in comparison:\n")
  print(table(sample_data(physeq_subset)$Biofilm_Status))

  # Enhanced filtering: remove low-abundance and problematic features
  total_samples <- nsamples(physeq_subset)
  min_prevalence_samples <- max(2, ceiling(total_samples * prevalence_threshold))  # At least 2 samples

  # Step 1: Basic prevalence filtering with increased stringency for small samples
  if (total_samples <= 8) {
    cat("Small sample size detected, using more stringent filtering criteria...\n")
    prevalence_threshold <- max(0.3, prevalence_threshold)  # At least 30% for small samples
    min_prevalence_samples <- max(3, ceiling(total_samples * prevalence_threshold))
  }

  physeq_filtered <- filter_taxa(physeq_subset, function(x) {
    sum(x > 0) >= min_prevalence_samples && sum(x) > 0  # Present in enough samples and total count > 0
  }, TRUE)

  # Step 2: Remove features with very low total abundance (more stringent for small samples)
  total_counts <- taxa_sums(physeq_filtered)
  min_total_count <- ifelse(total_samples <= 8, max(50, total_samples * 5), max(10, total_samples))
  physeq_filtered <- prune_taxa(total_counts >= min_total_count, physeq_filtered)

  # Step 3: Enhanced zero-variance detection and removal
  cat("Checking for zero-variance and problematic features...\n")
  otu_matrix <- as.matrix(otu_table(physeq_filtered))

  # Calculate variance for each feature with robust methods
  feature_vars <- apply(otu_matrix, 1, function(x) {
    if (all(x == 0)) return(0)  # All zeros
```

```
    if (length(unique(x[x > 0])) <= 1) return(0)    # All non-zero values are same
    if (sum(x > 0) < 2) return(0)    # Present in less than 2 samples
    return(var(x, na.rm = TRUE))
  })

  # Also check for features with extremely low variation relative to mean
  feature_cv <- apply(otu_matrix, 1, function(x) {
    if (mean(x, na.rm = TRUE) == 0) return(Inf)
    return(sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE))
  })

  # Identify problematic features
  zero_var_features <- names(feature_vars)[feature_vars == 0 | is.na(feature_vars)]
  low_cv_features <- names(feature_cv)[feature_cv < 0.01 & !is.infinite(feature_cv)]    # CV < 1%

  problematic_features <- unique(c(zero_var_features, low_cv_features))

  if (length(problematic_features) > 0) {
    cat("Found", length(problematic_features), "problematic features (zero-variance or low CV),
removing them:\n")
    cat("Problematic features:", paste(head(problematic_features, 15), collapse = ", "),
        ifelse(length(problematic_features) > 15, "...", ""), "\n")

    # Remove problematic features
    keep_features <- setdiff(taxa_names(physeq_filtered), problematic_features)
    physeq_filtered <- prune_taxa(keep_features, physeq_filtered)
  } else {
    cat("No problematic features detected.\n")
  }

  # Step 4: For small sample sizes, apply even more stringent filtering
  if (total_samples <= 8) {    # Small sample warning threshold
    cat("Small sample size detected, applying additional stringent filtering...\n")

    # Require features to be present in at least 50% of samples
    min_samples_present <- ceiling(total_samples * 0.5)
    physeq_filtered <- filter_taxa(physeq_filtered, function(x) {
      sum(x > 0) >= min_samples_present
    }, TRUE)

    # Remove features with very high sparsity
    sparsity <- apply(otu_table(physeq_filtered), 1, function(x) sum(x == 0) / length(x))
    high_sparsity_features <- names(sparsity)[sparsity > 0.7]    # More than 70% zeros
```

```
    if (length(high_sparsity_features) > 0) {
      cat("Removing", length(high_sparsity_features), "highly sparse features\n")
      keep_features_sparse <- setdiff(taxa_names(physeq_filtered), high_sparsity_features)
      physeq_filtered <- prune_taxa(keep_features_sparse, physeq_filtered)
    }
  }

  cat("Original number of features:", ntaxa(physeq_subset), "\n")
  cat("Number of features retained after filtering:", ntaxa(physeq_filtered), "\n")

  # Create binary comparison variable (reference: group1, comparison: group2)
  sample_data(physeq_filtered)$Comparison_Group <- factor(
    sample_data(physeq_filtered)$Biofilm_Status,
    levels = c(group1, group2)
  )

  # Run ANCOM-BC analysis with robust parameters
  tryCatch({
    # Determine parameters based on sample size
    small_sample <- total_samples <= 8

    ancombc_result <- ancombc2(
      data = physeq_filtered,
      assay_name = "counts",
      tax_level = NULL,
      fix_formula = "Comparison_Group",
      rand_formula = NULL,
      p_adj_method = "fdr",
      pseudo_sens = FALSE,    # Turn off for small samples to avoid issues
      prv_cut = ifelse(small_sample, 0.3, prevalence_threshold),    # More stringent for small
samples
      lib_cut = ifelse(small_sample, 10, 100),    # Lower threshold for small samples
      s0_perc = ifelse(small_sample, 0.2, 0.05),    # Higher s0_perc for stability
      group = "Comparison_Group",
      struc_zero = FALSE,    # Keep FALSE to avoid complications
      neg_lb = FALSE,    # Keep FALSE for robustness
      alpha = 0.05,
      n_cl = 1,
      verbose = TRUE
    )

    # Extract results
    cat("ANCOM-BC result structure check:\n")
    print(names(ancombc_result$res))
```

```
# Extract results - CRITICAL FIX: use actual taxa names instead of row indices
res_data <- ancombc_result$res

# Get the actual taxa names from the phyloseq object used in analysis
actual_taxa_names <- taxa_names(physeq_filtered)

# Find correct column names
lfc_cols <- grep("lfc", names(res_data), value = TRUE)
pval_cols <- grep("p_", names(res_data), value = TRUE)
qval_cols <- grep("q_", names(res_data), value = TRUE)

cat("Found column names:\n")
cat("Log fold change columns:", paste(lfc_cols, collapse = ", "), "\n")
cat("P-value columns:", paste(pval_cols, collapse = ", "), "\n")
cat("Adjusted P-value columns:", paste(qval_cols, collapse = ", "), "\n")

# Build results data frame with actual feature names
results_df <- data.frame(
  Feature_ID = actual_taxa_names,    # Use actual taxa names instead of row indices
  stringsAsFactors = FALSE
)

# Extract data based on actual column names
if (length(lfc_cols) > 0) {
  results_df$log2FoldChange <- res_data[[lfc_cols[1]]]
} else {
  # Alternative approach
  possible_lfc <- paste0("Comparison_Group", group2)
  found_lfc <- intersect(possible_lfc, names(res_data))
  if (length(found_lfc) > 0) {
    results_df$log2FoldChange <- res_data[[found_lfc[1]]]
  } else {
    # Try other possible column names
    lfc_pattern <- paste0("lfc.*", group2)
    found_lfc <- grep(lfc_pattern, names(res_data), value = TRUE)
    if (length(found_lfc) > 0) {
      results_df$log2FoldChange <- res_data[[found_lfc[1]]]
    } else {
      stop("Could not find log fold change column")
    }
  }
}
```

```
        if (length(pval_cols) > 0) {
            results_df$pval <- res_data[[pval_cols[1]]]
        } else {
            results_df$pval <- NA
        }

        if (length(qval_cols) > 0) {
            results_df$padj <- res_data[[qval_cols[1]]]
        } else {
            results_df$padj <- NA
        }

        # Add significance marker and comparison info
        results_df$significant <- results_df$padj < 0.05 & !is.na(results_df$padj)
        results_df$comparison <- comparison_name
        results_df$group1 <- group1
        results_df$group2 <- group2

        # Remove NA rows and sort
        results_df <- results_df %>%
            filter(!is.na(log2FoldChange) & !is.na(padj)) %>%
            arrange(padj)

        # Count significant differential features
        sig_up <- sum(results_df$log2FoldChange > 0 & results_df$padj < 0.05, na.rm = TRUE)
        sig_down <- sum(results_df$log2FoldChange < 0 & results_df$padj < 0.05, na.rm = TRUE)

        cat("Number of significantly upregulated features (", group2, " > ", group1, "):", sig_up, "\n")
        cat("Number of significantly downregulated features (", group2, " < ", group1, "):", sig_down,
"\n")

        return(list(
            results = results_df,
            ancombc_obj = ancombc_result,
            physeq = physeq_filtered,
            sig_up = sig_up,
            sig_down = sig_down,
            comparison = comparison_name,
            group1 = group1,
            group2 = group2
        ))

    }, error = function(e) {
        cat("Error in ANCOM-BC analysis:", e$message, "\n")
```

```
    # Additional zero-variance checking if not done already
    cat("Performing additional zero-variance filtering before retry...\n")
    otu_matrix_check <- as.matrix(otu_table(physeq_filtered))
    feature_vars_check <- apply(otu_matrix_check, 1, var)
    zero_var_features_check <- names(feature_vars_check)[feature_vars_check == 0 |
is.na(feature_vars_check)]

    if (length(zero_var_features_check) > 0) {
      cat("Additional zero-variance features found, removing them:\n")
      cat("Features:", paste(head(zero_var_features_check, 10), collapse = ", "),
          ifelse(length(zero_var_features_check) > 10, "...", ""), "\n")

      keep_features_retry <- setdiff(taxa_names(physeq_filtered), zero_var_features_check)
      physeq_filtered <- prune_taxa(keep_features_retry, physeq_filtered)
    }

    cat("Trying with more lenient parameters...\n")

    # Retry with even more conservative parameters
    ancombc_result <- ancombc2(
      data = physeq_filtered,
      assay_name = "counts",
      tax_level = NULL,
      fix_formula = "Comparison_Group",
      rand_formula = NULL,
      p_adj_method = "fdr",
      pseudo_sens = FALSE,
      prv_cut = max(0.2, prevalence_threshold),    # More stringent prevalence
      lib_cut = 50,    # Even lower library threshold
      s0_perc = 0.1,    # Higher s0_perc for more stability
      group = "Comparison_Group",
      struc_zero = FALSE,
      neg_lb = FALSE,
      alpha = 0.05,
      n_cl = 1,
      verbose = TRUE
    )

    # Extract results (similar process as above)
    res_data <- ancombc_result$res

    # Find correct column names
    lfc_cols <- grep("lfc", names(res_data), value = TRUE)
```

```
pval_cols <- grep("p_", names(res_data), value = TRUE)
qval_cols <- grep("q_", names(res_data), value = TRUE)

# Build results data frame
results_df <- data.frame(
  Feature_ID = rownames(res_data),
  stringsAsFactors = FALSE
)

# Extract data
if (length(lfc_cols) > 0) {
  results_df$log2FoldChange <- res_data[[lfc_cols[1]]]
} else {
  possible_lfc <- paste0("Comparison_Group", group2)
  found_lfc <- intersect(possible_lfc, names(res_data))
  if (length(found_lfc) > 0) {
    results_df$log2FoldChange <- res_data[[found_lfc[1]]]
  } else {
    lfc_pattern <- paste0("lfc.*", group2)
    found_lfc <- grep(lfc_pattern, names(res_data), value = TRUE)
    if (length(found_lfc) > 0) {
      results_df$log2FoldChange <- res_data[[found_lfc[1]]]
    } else {
      stop("Could not find log fold change column")
    }
  }
}

if (length(pval_cols) > 0) {
  results_df$pval <- res_data[[pval_cols[1]]]
} else {
  results_df$pval <- NA
}

if (length(qval_cols) > 0) {
  results_df$padj <- res_data[[qval_cols[1]]]
} else {
  results_df$padj <- NA
}

# Add significance marker and comparison info
results_df$significant <- results_df$padj < 0.05 & !is.na(results_df$padj)
results_df$comparison <- comparison_name
results_df$group1 <- group1
```

```
        results_df$group2 <- group2

        # Remove NA rows and sort
        results_df <- results_df %>%
            filter(!is.na(log2FoldChange) & !is.na(padj)) %>%
            arrange(padj)

        # Count significant differential features
        sig_up <- sum(results_df$log2FoldChange > 0 & results_df$padj < 0.05, na.rm = TRUE)
        sig_down <- sum(results_df$log2FoldChange < 0 & results_df$padj < 0.05, na.rm = TRUE)

        cat("Number of significantly upregulated features (", group2, " > ", group1, "):", sig_up, "\n")
        cat("Number of significantly downregulated features (", group2, " < ", group1, "):", sig_down,
"\n")

        return(list(
            results = results_df,
            ancombc_obj = ancombc_result,
            physeq = physeq_filtered,
            sig_up = sig_up,
            sig_down = sig_down,
            comparison = comparison_name,
            group1 = group1,
            group2 = group2
        ))
    })
}

# ==================== 3. Perform All Pairwise Comparisons ====================

# Define comparison pairs
comparisons <- list(
    list(group1 = "Young", group2 = "Old", name = "Old_vs_Young"),
    list(group1 = "Young", group2 = "Mature", name = "Mature_vs_Young"),
    list(group1 = "Old", group2 = "Mature", name = "Mature_vs_Old")
)

# Store all results
all_results <- list()

# Perform analysis for each functional level and each comparison
for (func_level in c("EC", "KO", "Pathway")) {

    cat("\n", paste(rep("=", 50), collapse = ""), "\n")
```

```
  cat("Processing", func_level, "level analysis\n")
  cat(paste(rep("=", 50), collapse = ""), "\n")

  # Select appropriate phyloseq object
  if (func_level == "EC") {
    physeq_func <- physeq_ec_biofilm
  } else if (func_level == "KO") {
    physeq_func <- physeq_ko_biofilm
  } else {
    physeq_func <- physeq_pathway_biofilm
  }

  # Basic data quality check
  cat("Initial data check for", func_level, "level:\n")
  cat("    Features:", ntaxa(physeq_func), "\n")
  cat("    Samples:", nsamples(physeq_func), "\n")

  # Check if we have any data
  if (ntaxa(physeq_func) == 0) {
    cat("Warning: No features found for", func_level, "level. Skipping...\n")
    next
  }

  level_results <- list()

  # Perform all pairwise comparisons
  for (comp in comparisons) {
    tryCatch({
      analysis_result <- perform_pairwise_ancombc(
        physeq_func,
        comp$group1,
        comp$group2,
        comp$name
      )
      level_results[[comp$name]] <- analysis_result
    }, error = function(e) {
      cat("Error in", func_level, comp$name, "analysis:", e$message, "\n")
      cat("This comparison will be skipped.\n")
      level_results[[comp$name]] <- NULL
    })
  }

  all_results[[func_level]] <- level_results
}
```

```
# ==================== 4. Save Results and Create Visualizations ====================

# 4.1 Save detailed results
for (func_level in names(all_results)) {
  for (comp_name in names(all_results[[func_level]])) {
    result_data <- all_results[[func_level]][[comp_name]]$results
    filename <- paste0("Biofilm_", func_level, "_", comp_name, "_ANCOMBC_results.csv")
    write.csv(result_data, file.path(output_dir, filename), row.names = FALSE)

    # Save significant features
    sig_features <- result_data %>%
      filter(padj < 0.05 & abs(log2FoldChange) > 0.5)

    if (nrow(sig_features) > 0) {
      sig_filename <- paste0("Biofilm_", func_level, "_", comp_name,
"_significant_features.csv")
      write.csv(sig_features, file.path(output_dir, sig_filename), row.names = FALSE)
    }
  }
}

# 4.2 Create visualization functions

# Volcano plot function
create_biofilm_volcano_plot <- function(results_df, comparison_name, func_level, filename) {

  results_df <- results_df %>%
    mutate(
      Significance = case_when(
        padj < 0.05 & log2FoldChange > 0.5 ~ paste("Up in", results_df$group2[1]),
        padj < 0.05 & log2FoldChange < -0.5 ~ paste("Up in", results_df$group1[1]),
        TRUE ~ "Not Significant"
      )
    )

  top_features <- results_df %>%
    filter(Significance != "Not Significant") %>%
    arrange(padj) %>%
    head(10)

  # Create color values with proper naming
  color_up_group2 <- paste("Up in", results_df$group2[1])
  color_up_group1 <- paste("Up in", results_df$group1[1])
```

```
  color_values <- c("#d62728", "#2ca02c", "grey70")
  names(color_values) <- c(color_up_group2, color_up_group1, "Not Significant")

  p <- ggplot(results_df, aes(x = log2FoldChange, y = -log10(padj))) +
    geom_point(aes(color = Significance), alpha = 0.7, size = 1.5) +
    scale_color_manual(values = color_values) +
    geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "black", alpha = 0.5) +
    geom_vline(xintercept = c(-0.5, 0.5), linetype = "dashed", color = "black", alpha = 0.5) +
    labs(
      title = paste("Volcano Plot -", func_level, "Functions"),
      subtitle = paste("Biofilm Status:", comparison_name, "(ANCOM-BC)"),
      x = paste("log2 Fold Change (", results_df$group2[1], "/", results_df$group1[1], ")"),
      y = "-log10(adjusted p-value)",
      color = "Significance"
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      plot.subtitle = element_text(hjust = 0.5, size = 12),
      legend.position = "bottom"
    )

  if (nrow(top_features) > 0) {
    p <- p + geom_text_repel(
      data = top_features,
      aes(label = Feature_ID),
      size = 3,
      max.overlaps = 10,
      box.padding = 0.3
    )
  }

  ggsave(filename, plot = p, width = 12, height = 8, dpi = 300)
  return(p)
}

# Lollipop plot function
create_biofilm_lollipop_plot <- function(results_df, comparison_name, func_level, filename,
top_n = 20) {

  sig_features <- results_df %>%
    filter(padj < 0.05) %>%
    arrange(padj) %>%
```

```
  head(top_n)

if (nrow(sig_features) == 0) {
  cat("No significant features for lollipop plot:", comparison_name, func_level, "\n")
  return(NULL)
}

plot_data <- sig_features %>%
  mutate(
    Direction = ifelse(log2FoldChange > 0,
                       paste("Up in", group2),
                       paste("Up in", group1)),
    Feature_ID_short = ifelse(nchar(Feature_ID) > 40,
                              paste0(substr(Feature_ID, 1, 37), "..."),
                              Feature_ID),
    abs_lfc = abs(log2FoldChange)
  ) %>%
  arrange(abs_lfc)

plot_data$Feature_ID_short <- factor(plot_data$Feature_ID_short,
                                     levels = plot_data$Feature_ID_short)

# Create color values with proper naming
color_up_group2 <- paste("Up in", plot_data$group2[1])
color_up_group1 <- paste("Up in", plot_data$group1[1])

color_values <- c("#d62728", "#2ca02c")
names(color_values) <- c(color_up_group2, color_up_group1)

p <- ggplot(plot_data, aes(x = Feature_ID_short, y = log2FoldChange)) +
  geom_segment(aes(x = Feature_ID_short, xend = Feature_ID_short,
                   y = 0, yend = log2FoldChange, color = Direction),
               size = 1) +
  geom_point(aes(color = Direction, size = -log10(padj)), alpha = 0.8) +
  scale_color_manual(values = color_values) +
  scale_size_continuous(name = "-log10(padj)", range = c(2, 8)) +
  coord_flip() +
  labs(
    title = paste("Top Significant Features -", func_level, "Functions"),
    subtitle = paste("Biofilm Status:", comparison_name, "(ANCOM-BC)"),
    x = "Functional Features",
    y = paste("log2 Fold Change (", plot_data$group2[1], "/", plot_data$group1[1], ")"),
    color = "Direction",
    size = "Significance"
```

```
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      plot.subtitle = element_text(hjust = 0.5, size = 12),
      axis.text.y = element_text(size = 9),
      legend.position = "bottom"
    ) +
    geom_hline(yintercept = 0, linetype = "solid", color = "black", alpha = 0.3)

  ggsave(filename, plot = p, width = 14, height = 10, dpi = 300)
  return(p)
}

# Heatmap function for biofilm analysis
create_biofilm_heatmap <- function(physeq_obj, results_df, comparison_name, func_level,
filename, top_n = 30) {

  cat("\n=== Creating heatmap:", func_level, comparison_name, "===\n")

  # Check if there are results data
  if (nrow(results_df) == 0) {
    cat("No results data for heatmap:", comparison_name, func_level, "\n")
    return(NULL)
  }

  # Select top significant features
  top_features <- results_df %>%
    filter(!is.na(padj)) %>%
    arrange(padj) %>%
    head(top_n)

  if (nrow(top_features) == 0) {
    cat("No significant features for heatmap:", comparison_name, func_level, "\n")
    return(NULL)
  }

  cat("Number of candidate features:", nrow(top_features), "\n")

  # Check if feature IDs exist in phyloseq object
  available_taxa <- taxa_names(physeq_obj)
  feature_ids <- top_features$Feature_ID

  cat("Number of taxa in phyloseq object:", length(available_taxa), "\n")
```

```
cat("Number of candidate feature IDs:", length(feature_ids), "\n")

# Find matching features
valid_features <- intersect(feature_ids, available_taxa)
cat("Number of matching features:", length(valid_features), "\n")

if (length(valid_features) == 0) {
  cat("Warning: No matching feature IDs, checking first few feature IDs:\n")
  cat("First 5 feature IDs in results:\n")
  print(head(feature_ids, 5))
  cat("First 5 taxa in phyloseq:\n")
  print(head(available_taxa, 5))
  return(NULL)
}

# Limit feature count to avoid oversized images
if (length(valid_features) > top_n) {
  valid_features <- valid_features[1:top_n]
}

cat("Final number of features for plotting:", length(valid_features), "\n")

tryCatch({
  # Extract abundance data
  otu_table_subset <- otu_table(physeq_obj)[valid_features, , drop = FALSE]
  otu_df <- as.data.frame(otu_table_subset)

  cat("Extracted data dimensions:", dim(otu_df), "\n")

  # Check if data is empty
  if (nrow(otu_df) == 0 || ncol(otu_df) == 0) {
    cat("Extracted data is empty, cannot create heatmap\n")
    return(NULL)
  }

  # Convert to relative abundance (avoid division by zero)
  col_sums <- colSums(otu_df)
  if (any(col_sums == 0)) {
    cat("Some samples have total abundance of 0, adding pseudo-counts\n")
    otu_df <- otu_df + 1
    col_sums <- colSums(otu_df)
  }

  otu_rel <- sweep(otu_df, 2, col_sums, "/") * 100
```

```
# Log transformation and normalization (z-score)
otu_log <- log10(otu_rel + 1e-6)
otu_scaled <- t(scale(t(otu_log)))

# Handle NaN values
otu_scaled[is.nan(otu_scaled)] <- 0
otu_scaled[is.infinite(otu_scaled)] <- 0

# Simplify feature names
rownames(otu_scaled) <- ifelse(nchar(rownames(otu_scaled)) > 50,
                               paste0(substr(rownames(otu_scaled), 1, 47), "..."),
                               rownames(otu_scaled))

# Prepare sample annotations
sample_anno <- data.frame(
  Biofilm_Status = sample_data(physeq_obj)$Biofilm_Status,
  row.names = sample_names(physeq_obj)
)

# Ensure sample_anno row names match otu_scaled column names
common_samples <- intersect(rownames(sample_anno), colnames(otu_scaled))
sample_anno <- sample_anno[common_samples, , drop = FALSE]
otu_scaled <- otu_scaled[, common_samples, drop = FALSE]

cat("Final plotting data dimensions:", dim(otu_scaled), "\n")
cat("Sample annotation dimensions:", dim(sample_anno), "\n")

# Color settings for biofilm status
anno_colors <- list(
  Biofilm_Status = c("Young" = "#1f77b4", "Old" = "#ff7f0e", "Mature" = "#2ca02c")
)

# Create heatmap
pheatmap(
  otu_scaled,
  annotation_col = sample_anno,
  annotation_colors = anno_colors,
  clustering_distance_rows = "correlation",
  clustering_distance_cols = "correlation",
  show_rownames = TRUE,
  show_colnames = TRUE,    # Show sample names
  scale = "none",
  color = colorRampPalette(c("navy", "white", "firebrick"))(100),
```

```
            filename = filename,
            width = 16,
            height = max(8, nrow(otu_scaled) * 0.3),    # Dynamically adjust height
            fontsize_col = 8,    # Adjust sample name font size
            fontsize_row = 8,    # Adjust feature name font size
            angle_col = 45,        # Tilt sample names for better readability
            main = paste("Biofilm Status:", comparison_name, "-", func_level, "Functions")
        )

        cat("Heatmap saved to:", filename, "\n")

    }, error = function(e) {
        cat("Error while creating heatmap:", e$message, "\n")
        cat("Trying to create simplified version of heatmap...\n")

        # Simplified version: use only first 10 features
        simple_features <- valid_features[1:min(10, length(valid_features))]

        tryCatch({
            otu_simple <- as.data.frame(otu_table(physeq_obj)[simple_features, , drop = FALSE])
            otu_simple_rel <- sweep(otu_simple, 2, colSums(otu_simple + 1), "/") * 100

            pheatmap(
                log10(otu_simple_rel + 1),
                show_rownames = TRUE,
                show_colnames = TRUE,
                filename = filename,
                width = 12,
                height = 8,
                fontsize_col = 8,
                angle_col = 45,
                main = paste("Biofilm Status:", comparison_name, "-", func_level, "(Simplified)")
            )

            cat("Simplified heatmap saved to:", filename, "\n")

        }, error = function(e2) {
            cat("Simplified heatmap also failed:", e2$message, "\n")
            return(NULL)
        })
    })
}

# 4.3 Generate all plots (including heatmaps)
```

```
for (func_level in names(all_results)) {
  for (comp_name in names(all_results[[func_level]])) {

    # Check if analysis was successful
    if (is.null(all_results[[func_level]][[comp_name]])) {
      cat("Skipping visualization for", func_level, comp_name, "- analysis failed\n")
      next
    }

    result_data <- all_results[[func_level]][[comp_name]]$results
    physeq_obj <- all_results[[func_level]][[comp_name]]$physeq

    # Skip if no results
    if (is.null(result_data) || nrow(result_data) == 0) {
      cat("Skipping visualization for", func_level, comp_name, "- no results\n")
      next
    }

    # Volcano plot
    volcano_filename <- file.path(output_dir,
                                  paste0("Biofilm_", func_level, "_", comp_name,
"_volcano_plot.png"))
    tryCatch({
      create_biofilm_volcano_plot(result_data, comp_name, func_level, volcano_filename)
    }, error = function(e) cat("Error creating volcano plot for", func_level, comp_name, ":",
e$message, "\n"))

    # Lollipop plot
    lollipop_filename <- file.path(output_dir,
                                   paste0("Biofilm_", func_level, "_", comp_name,
"_lollipop_plot.png"))
    tryCatch({
      create_biofilm_lollipop_plot(result_data, comp_name, func_level, lollipop_filename)
    }, error = function(e) cat("Error creating lollipop plot for", func_level, comp_name, ":",
e$message, "\n"))

    # Heatmap
    heatmap_filename <- file.path(output_dir,
                                  paste0("Biofilm_", func_level, "_", comp_name,
"_heatmap.png"))
    tryCatch({
      create_biofilm_heatmap(physeq_obj, result_data, comp_name, func_level,
heatmap_filename)
    }, error = function(e) cat("Error creating heatmap for", func_level, comp_name, ":",
```

```
e$message, "\n"))
  }
}

# 4.4 Create summary comparison plot
create_summary_comparison <- function() {
  summary_data <- data.frame()

  for (func_level in names(all_results)) {
    for (comp_name in names(all_results[[func_level]])) {
      result <- all_results[[func_level]][[comp_name]]

      # Skip if analysis failed
      if (is.null(result)) {
        cat("Skipping summary for", func_level, comp_name, "- analysis failed\n")
        next
      }

      summary_data <- rbind(summary_data, data.frame(
        Functional_Level = func_level,
        Comparison = comp_name,
        Up_regulated = result$sig_up,
        Down_regulated = result$sig_down,
        Total_significant = result$sig_up + result$sig_down
      ))
    }
  }

  # Check if we have any data to plot
  if (nrow(summary_data) == 0) {
    cat("No successful analyses to summarize\n")
    return(NULL)
  }

  # Reshape for plotting
  plot_data <- summary_data %>%
    select(-Total_significant) %>%
    pivot_longer(cols = c(Up_regulated, Down_regulated),
                 names_to = "Direction", values_to = "Count") %>%
    mutate(
      Direction = gsub("_", " ", Direction),
      Comparison = gsub("_vs_", " vs ", Comparison)    # Fix the comparison name display
    )
```

```
  p <- ggplot(plot_data, aes(x = Comparison, y = Count, fill = Direction)) +
    geom_bar(stat = "identity", position = "dodge", alpha = 0.8) +
    facet_wrap(~Functional_Level, scales = "free_y") +
    scale_fill_manual(values = c("Up regulated" = "#d62728", "Down regulated" = "#2ca02c")) +
    labs(
      title = "Biofilm Maturity: Number of Significantly Different Functions",
      subtitle = "Untreated Samples Only (ANCOM-BC, padj < 0.05)",
      x = "Pairwise Comparisons",
      y = "Number of Features",
      fill = "Direction"
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      plot.subtitle = element_text(hjust = 0.5, size = 12),
      axis.text.x = element_text(angle = 45, hjust = 1),
      strip.text = element_text(size = 12, face = "bold")
    ) +
    geom_text(aes(label = Count), position = position_dodge(width = 0.9), vjust = -0.3, size = 3)

  ggsave(file.path(output_dir, "Biofilm_Status_functional_summary.png"),
         plot = p, width = 14, height = 8, dpi = 300)

  return(p)
}

summary_plot <- create_summary_comparison()

# ==================== 5. Generate Analysis Report ====================

# Create detailed summary report
report_lines <- c(
  "Biofilm Status Functional Difference Analysis Report (ANCOM-BC)",
  paste(rep("=", 70), collapse = ""),
  "",
  paste("Analysis Date:", Sys.Date()),
  "Analysis Method: ANCOM-BC (pairwise comparisons)",
  "Sample Filter: Treatment == 'Untreated' only",
  "",
  "Sample Information:",
  paste("    Total Untreated Samples:", nsamples(physeq_biofilm))
)

# Add sample counts for each biofilm status
```

```
biofilm_counts <- table(sample_data(physeq_biofilm)$Biofilm_Status)
for (status in names(biofilm_counts)) {
  report_lines <- c(report_lines, paste("    ", status, "Group:", biofilm_counts[status]))
}

report_lines <- c(report_lines, "", "Pairwise Comparisons Performed:")

# Add results for each comparison and functional level
for (func_level in names(all_results)) {
  report_lines <- c(report_lines, paste("\n", func_level, "Level:"))

  # Check if this functional level has any successful results
  has_results <- FALSE

  for (comp_name in names(all_results[[func_level]])) {
    result <- all_results[[func_level]][[comp_name]]

    if (!is.null(result)) {
      has_results <- TRUE
      # Fix the comparison name display
      comp_display <- gsub("_vs_", " vs ", comp_name)

      report_lines <- c(report_lines, paste("    ", comp_display, ":"))
      report_lines <- c(report_lines, paste("      Upregulated:", result$sig_up))
      report_lines <- c(report_lines, paste("      Downregulated:", result$sig_down))
      report_lines <- c(report_lines, paste("      Total significant:", result$sig_up +
result$sig_down))
    } else {
      # Fix the comparison name display for failed analyses too
      comp_display <- gsub("_vs_", " vs ", comp_name)
      report_lines <- c(report_lines, paste("    ", comp_display, ": Analysis failed"))
    }
  }

  if (!has_results) {
    report_lines <- c(report_lines, "    No successful analyses for this functional level")
  }
}

report_lines <- c(report_lines,
  "",
  "Output Files:",
  "    1. Differential Analysis Results: Biofilm_*_*_ANCOMBC_results.csv",
  "    2. Significant Feature Lists: Biofilm_*_*_significant_features.csv",
```

```
  "    3. Volcano Plots: Biofilm_*_*_volcano_plot.png",
  "    4. Lollipop Plots: Biofilm_*_*_lollipop_plot.png",
  "    5. Heatmaps: Biofilm_*_*_heatmap.png",
  "    6. Summary Figure: Biofilm_Status_functional_summary.png",
  "",
  "Notes:",
  "- Analysis focuses on biofilm maturity progression in untreated conditions",
  "- Enhanced data filtering removes zero-variance, low CV, and problematic features",
  "- Small sample size handling: automatic parameter adjustment for samples ≤8",
  "- Prevalence threshold: features must be present in ≥20-50% of samples (adjusted for sample
size)",
  "- Minimum total count threshold applied to ensure robust statistical analysis",
  "- Ultra-conservative retry mechanism for failed analyses with zero-variance errors",
  "- Special filtering for KO features mentioned in error messages",
  "- Upregulated indicates higher abundance in the second group of each comparison",
  "- ANCOM-BC method accounts for compositional nature of microbiome data",
  "- All pairwise comparisons: Old vs Young, Mature vs Young, Mature vs Old",
  "- Heatmaps show abundance patterns of top significant features across samples",
  "- Sample names are displayed in heatmaps with biofilm status color-coding",
  "- Results provide insights into functional changes during biofilm development",
  "- Conservative parameters used to ensure statistical robustness with small samples",
  "- Failed analyses are reported and skipped to allow other comparisons to proceed"
)

writeLines(report_lines, file.path(output_dir, "Biofilm_Status_analysis_summary.txt"))

cat("\n=== Biofilm Status ANCOM-BC Analysis Completed ===\n")
cat("All results saved to:", output_dir, "\n")
cat("Please refer to Biofilm_Status_analysis_summary.txt for detailed analysis report\n")

# Display summary statistics
cat("\nQuick Summary:\n")
for (func_level in names(all_results)) {
  cat("\n", func_level, "Level:\n")
  has_results <- FALSE

  for (comp_name in names(all_results[[func_level]])) {
    result <- all_results[[func_level]][[comp_name]]

    if (!is.null(result)) {
      has_results <- TRUE
      # Fix the comparison name display
      comp_display <- gsub("_vs_", " vs ", comp_name)
      cat("    ", comp_display, ": ",
```

```
                result$sig_up + result$sig_down, " significant features\n")
      }
    }

    if (!has_results) {
      cat("    No successful analyses\n")
    }
}

# ==================== Treatment Methods Functional Difference Analysis (ANCOM-BC) ====================
# Analysis 3: Treatment effects on functional composition
# Comparisons: Water-Vapor vs (Untreated, Thermal Disinfection, Chemical Treatment)
#                         Chemical Treatment vs Chemical Treatment + Water-Vapor

# Load required packages
library(phyloseq)
library(ANCOMBC)
library(ggplot2)
library(dplyr)
library(tibble)
library(pheatmap)
library(RColorBrewer)
library(VennDiagram)
library(ggrepel)
library(tidyr)

# Set output path
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/function desq2"
if (!dir.exists(output_dir)) {
  dir.create(output_dir, recursive = TRUE)
}

# ==================== 1. Data Preprocessing and Sample Filtering ====================

# Check values in Treatment column
cat("Checking Treatment column values:\n")
treatment_status <- sample_data(physeq)$Treatment
print(table(treatment_status, useNA = "ifany"))

# Filter valid samples (remove NA values)
valid_samples <- !is.na(sample_data(physeq)$Treatment)
physeq_treatment <- prune_samples(valid_samples, physeq)
physeq_pathway_treatment <- prune_samples(valid_samples, physeq_pathway)
```

```
physeq_ec_treatment <- prune_samples(valid_samples, physeq_ec)
physeq_ko_treatment <- prune_samples(valid_samples, physeq_ko)

cat("Number of samples after filtering:\n")
cat("Total samples:", nsamples(physeq_treatment), "\n")
treatment_counts <- table(sample_data(physeq_treatment)$Treatment)
print(treatment_counts)

# ==================== 2. Define Enhanced ANCOM-BC Treatment Analysis Function ====================

perform_treatment_ancombc <- function(physeq_obj, group1, group2, comparison_name, prevalence_threshold = 0.1) {

  cat("\n=== Starting", comparison_name, "ANCOM-BC Analysis ===\n")

  # Filter samples for specific comparison
  target_groups <- c(group1, group2)
  group_samples <- sample_data(physeq_obj)$Treatment %in% target_groups
  physeq_subset <- prune_samples(group_samples, physeq_obj)

  cat("Samples in comparison:\n")
  print(table(sample_data(physeq_subset)$Treatment))

  # Enhanced filtering: remove low-abundance and problematic features
  total_samples <- nsamples(physeq_subset)

  # Adjust threshold based on sample size
  if (total_samples <= 15) {
    cat("Moderate sample size detected, using appropriate filtering criteria...\n")
    prevalence_threshold <- max(0.15, prevalence_threshold)  # At least 15% for moderate samples
  }

  min_prevalence_samples <- max(2, ceiling(total_samples * prevalence_threshold))

  # Step 1: Basic prevalence filtering
  physeq_filtered <- filter_taxa(physeq_subset, function(x) {
    sum(x > 0) >= min_prevalence_samples && sum(x) > 0
  }, TRUE)

  # Step 2: Remove features with very low total abundance
  total_counts <- taxa_sums(physeq_filtered)
  min_total_count <- max(20, total_samples * 2)  # At least 20 total counts or 2 per sample
```

```
physeq_filtered <- prune_taxa(total_counts >= min_total_count, physeq_filtered)

# Step 3: Enhanced zero-variance detection and removal
cat("Checking for zero-variance and problematic features...\n")
otu_matrix <- as.matrix(otu_table(physeq_filtered))

# Calculate variance for each feature with robust methods
feature_vars <- apply(otu_matrix, 1, function(x) {
  if (all(x == 0)) return(0)    # All zeros
  if (length(unique(x[x > 0])) <= 1) return(0)    # All non-zero values are same
  if (sum(x > 0) < 2) return(0)    # Present in less than 2 samples
  return(var(x, na.rm = TRUE))
})

# Also check for features with extremely low variation relative to mean
feature_cv <- apply(otu_matrix, 1, function(x) {
  if (mean(x, na.rm = TRUE) == 0) return(Inf)
  return(sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE))
})

# Identify problematic features
zero_var_features <- names(feature_vars)[feature_vars == 0 | is.na(feature_vars)]
low_cv_features <- names(feature_cv)[feature_cv < 0.01 & !is.infinite(feature_cv)]

problematic_features <- unique(c(zero_var_features, low_cv_features))

if (length(problematic_features) > 0) {
  cat("Found", length(problematic_features), "problematic features, removing them:\n")
  cat("Problematic features:", paste(head(problematic_features, 15), collapse = ", "),
      ifelse(length(problematic_features) > 15, "...", ""), "\n")

  keep_features <- setdiff(taxa_names(physeq_filtered), problematic_features)
  physeq_filtered <- prune_taxa(keep_features, physeq_filtered)
} else {
  cat("No problematic features detected.\n")
}

# Step 4: For very small sample sizes, apply additional filtering
if (total_samples <= 5) {
  cat("Very small sample size detected, applying additional stringent filtering...\n")

  # Require features to be present in at least 60% of samples
  min_samples_present <- ceiling(total_samples * 0.6)
  physeq_filtered <- filter_taxa(physeq_filtered, function(x) {
```

```
    sum(x > 0) >= min_samples_present
  }, TRUE)

  # Remove features with very high sparsity
  sparsity <- apply(otu_table(physeq_filtered), 1, function(x) sum(x == 0) / length(x))
  high_sparsity_features <- names(sparsity)[sparsity > 0.6]

  if (length(high_sparsity_features) > 0) {
    cat("Removing", length(high_sparsity_features), "highly sparse features\n")
    keep_features_sparse <- setdiff(taxa_names(physeq_filtered), high_sparsity_features)
    physeq_filtered <- prune_taxa(keep_features_sparse, physeq_filtered)
  }
}

cat("Original number of features:", ntaxa(physeq_subset), "\n")
cat("After filtering:", ntaxa(physeq_filtered), "\n")

# Final check: ensure we have enough features for analysis
if (ntaxa(physeq_filtered) < 10) {
  warning("Very few features remaining after filtering (", ntaxa(physeq_filtered),
          "). Consider relaxing filtering criteria.")
}

cat("Final number of features for analysis:", ntaxa(physeq_filtered), "\n")
cat("Final number of samples for analysis:", nsamples(physeq_filtered), "\n")

# Create binary comparison variable (reference: group1, comparison: group2)
sample_data(physeq_filtered)$Comparison_Group <- factor(
  sample_data(physeq_filtered)$Treatment,
  levels = c(group1, group2)
)

# Run ANCOM-BC analysis with parameters optimized for treatment comparisons
tryCatch({
  # Determine parameters based on sample size
  very_small_sample <- total_samples <= 5
  small_sample <- total_samples <= 10

  ancombc_result <- ancombc2(
    data = physeq_filtered,
    assay_name = "counts",
    tax_level = NULL,
    fix_formula = "Comparison_Group",
    rand_formula = NULL,
```

```
    p_adj_method = "fdr",
    pseudo_sens = FALSE,    # Turn off for small samples
    prv_cut = ifelse(very_small_sample, 0.4, ifelse(small_sample, 0.2, prevalence_threshold)),
    lib_cut = ifelse(very_small_sample, 5, ifelse(small_sample, 50, 100)),
    s0_perc = ifelse(very_small_sample, 0.3, ifelse(small_sample, 0.1, 0.05)),
    group = "Comparison_Group",
    struc_zero = FALSE,
    neg_lb = FALSE,
    alpha = 0.05,
    n_cl = 1,
    verbose = TRUE
  )

  # Extract results - CRITICAL FIX: use actual taxa names instead of row indices
  cat("ANCOM-BC result structure check:\n")
  print(names(ancombc_result$res))

  res_data <- ancombc_result$res

  # Get the actual taxa names from the phyloseq object used in analysis
  actual_taxa_names <- taxa_names(physeq_filtered)

  # Find correct column names
  lfc_cols <- grep("lfc", names(res_data), value = TRUE)
  pval_cols <- grep("p_", names(res_data), value = TRUE)
  qval_cols <- grep("q_", names(res_data), value = TRUE)

  cat("Found column names:\n")
  cat("Log fold change columns:", paste(lfc_cols, collapse = ", "), "\n")
  cat("P-value columns:", paste(pval_cols, collapse = ", "), "\n")
  cat("Adjusted P-value columns:", paste(qval_cols, collapse = ", "), "\n")

  # Build results data frame with actual feature names
  results_df <- data.frame(
    Feature_ID = actual_taxa_names,    # Use actual taxa names instead of row indices
    stringsAsFactors = FALSE
  )

  # Extract data based on actual column names
  if (length(lfc_cols) > 0) {
    lfc_col <- grep(paste0("Comparison_Group", gsub("[^A-Za-z0-9]", "", group2)), lfc_cols,
value = TRUE)[1]
    if (!is.na(lfc_col)) {
      results_df$log2FoldChange <- res_data[[lfc_col]]
```

```
        } else {
            results_df$log2FoldChange <- res_data[[lfc_cols[length(lfc_cols)]]]
        }
    } else {
        stop("Could not find log fold change column")
    }

    if (length(pval_cols) > 0) {
        pval_col <- grep(paste0("Comparison_Group", gsub("[^A-Za-z0-9]", "", group2)), pval_cols,
value = TRUE)[1]
        if (!is.na(pval_col)) {
            results_df$pval <- res_data[[pval_col]]
        } else {
            results_df$pval <- res_data[[pval_cols[length(pval_cols)]]]
        }
    } else {
        results_df$pval <- NA
    }

    if (length(qval_cols) > 0) {
        qval_col <- grep(paste0("Comparison_Group", gsub("[^A-Za-z0-9]", "", group2)), qval_cols,
value = TRUE)[1]
        if (!is.na(qval_col)) {
            results_df$padj <- res_data[[qval_col]]
        } else {
            results_df$padj <- res_data[[qval_cols[length(qval_cols)]]]
        }
    } else {
        results_df$padj <- NA
    }

    # Add significance marker and comparison info
    results_df$significant <- results_df$padj < 0.05 & !is.na(results_df$padj)
    results_df$comparison <- comparison_name
    results_df$group1 <- group1
    results_df$group2 <- group2

    # Remove NA rows and sort with additional safety checks
    results_df <- results_df %>%
        filter(!is.na(log2FoldChange) & !is.na(padj) & is.finite(log2FoldChange) &
is.finite(padj)) %>%
        arrange(padj)

    # Count significant differential features
```

```
    sig_up <- sum(results_df$log2FoldChange > 0 & results_df$padj < 0.05, na.rm = TRUE)
    sig_down <- sum(results_df$log2FoldChange < 0 & results_df$padj < 0.05, na.rm = TRUE)

    cat("Number of significantly upregulated features (", group2, " > ", group1, "):", sig_up, "\n")
    cat("Number of significantly downregulated features (", group2, " < ", group1, "):", sig_down,
"\n")

    return(list(
      results = results_df,
      ancombc_obj = ancombc_result,
      physeq = physeq_filtered,
      sig_up = sig_up,
      sig_down = sig_down,
      comparison = comparison_name,
      group1 = group1,
      group2 = group2
    ))

  }, error = function(e) {
    cat("Error in ANCOM-BC analysis:", e$message, "\n")

    # Enhanced zero-variance detection and removal
    cat("Performing enhanced zero-variance filtering before retry...\n")
    otu_matrix_check <- as.matrix(otu_table(physeq_filtered))

    # Multiple checks for problematic features
    problematic_check <- character(0)

    # Check 1: Zero variance
    feature_vars_check <- apply(otu_matrix_check, 1, function(x) {
      if (all(x == 0)) return(0)
      if (length(unique(x[x > 0])) <= 1) return(0)
      if (sum(x > 0) < 2) return(0)
      return(var(x, na.rm = TRUE))
    })
    zero_var_check <- names(feature_vars_check)[feature_vars_check == 0 |
is.na(feature_vars_check)]
    problematic_check <- c(problematic_check, zero_var_check)

    # Check 2: Features mentioned in error message
    if (grepl("ko:K", e$message)) {
      ko_pattern <- "ko:K[0-9]+"
      error_kos <- regmatches(e$message, gregexpr(ko_pattern, e$message))[[1]]
      if (length(error_kos) > 0) {
```

```
            cat("Found specific KO features mentioned in error:", paste(head(error_kos, 10), collapse = ", "), "\n")
            problematic_check <- c(problematic_check, error_kos)
        }
    }

    # Check 3: Features with identical values across samples
    identical_features <- apply(otu_matrix_check, 1, function(x) {
        length(unique(x)) == 1
    })
    identical_check <- names(identical_features)[identical_features]
    problematic_check <- c(problematic_check, identical_check)

    # Check 4: Features present in only one sample
    single_sample_features <- apply(otu_matrix_check, 1, function(x) {
        sum(x > 0) <= 1
    })
    single_check <- names(single_sample_features)[single_sample_features]
    problematic_check <- c(problematic_check, single_check)

    # Remove all problematic features
    problematic_final <- unique(problematic_check)

    if (length(problematic_final) > 0) {
        cat("Enhanced filtering: removing", length(problematic_final), "problematic features:\n")
        cat("Features:", paste(head(problematic_final, 15), collapse = ", "),
            ifelse(length(problematic_final) > 15, "...", ""), "\n")

        keep_features_retry <- setdiff(taxa_names(physeq_filtered), problematic_final)
        physeq_filtered <- prune_taxa(keep_features_retry, physeq_filtered)

        cat("Features remaining after enhanced filtering:", ntaxa(physeq_filtered), "\n")
    }

    # Update comparison group after filtering
    sample_data(physeq_filtered)$Comparison_Group <- factor(
        sample_data(physeq_filtered)$Treatment,
        levels = c(group1, group2)
    )

    cat("Trying with ultra-conservative parameters...\n")

    # Retry with ultra-conservative parameters
    tryCatch({
```

```
        ancombc_result <- ancombc2(
            data = physeq_filtered,
            assay_name = "counts",
            tax_level = NULL,
            fix_formula = "Comparison_Group",
            rand_formula = NULL,
            p_adj_method = "fdr",
            pseudo_sens = FALSE,
            prv_cut = 0.5,
            lib_cut = 5,
            s0_perc = 0.3,
            group = "Comparison_Group",
            struc_zero = FALSE,
            neg_lb = FALSE,
            alpha = 0.1,
            n_cl = 1,
            verbose = TRUE
        )

        # Extract results with actual taxa names
        res_data <- ancombc_result$res
        actual_taxa_names <- taxa_names(physeq_filtered)

        # Find correct column names
        lfc_cols <- grep("lfc", names(res_data), value = TRUE)
        pval_cols <- grep("p_", names(res_data), value = TRUE)
        qval_cols <- grep("q_", names(res_data), value = TRUE)

        # Build results data frame with actual feature names
        results_df <- data.frame(
            Feature_ID = actual_taxa_names,
            stringsAsFactors = FALSE
        )

        # Extract data
        if (length(lfc_cols) > 0) {
            lfc_col <- grep(paste0("Comparison_Group", gsub("[^A-Za-z0-9]", "", group2)), lfc_cols,
value = TRUE)[1]
            if (!is.na(lfc_col)) {
                results_df$log2FoldChange <- res_data[[lfc_col]]
            } else {
                results_df$log2FoldChange <- res_data[[lfc_cols[length(lfc_cols)]]]
            }
        } else {
```

```
        stop("Could not find log fold change column")
    }

    if (length(pval_cols) > 0) {
        pval_col <- grep(paste0("Comparison_Group", gsub("[^A-Za-z0-9]", "", group2)),
pval_cols, value = TRUE)[1]
        if (!is.na(pval_col)) {
            results_df$pval <- res_data[[pval_col]]
        } else {
            results_df$pval <- res_data[[pval_cols[length(pval_cols)]]]
        }
    } else {
        results_df$pval <- NA
    }

    if (length(qval_cols) > 0) {
        qval_col <- grep(paste0("Comparison_Group", gsub("[^A-Za-z0-9]", "", group2)),
qval_cols, value = TRUE)[1]
        if (!is.na(qval_col)) {
            results_df$padj <- res_data[[qval_col]]
        } else {
            results_df$padj <- res_data[[qval_cols[length(qval_cols)]]]
        }
    } else {
        results_df$padj <- NA
    }

    # Add significance marker and comparison info
    results_df$significant <- results_df$padj < 0.05 & !is.na(results_df$padj)
    results_df$comparison <- comparison_name
    results_df$group1 <- group1
    results_df$group2 <- group2

    # Remove NA rows and sort with additional safety checks
    results_df <- results_df %>%
        filter(!is.na(log2FoldChange) & !is.na(padj) & is.finite(log2FoldChange) &
is.finite(padj)) %>%
        arrange(padj)

    # Count significant differential features
    sig_up <- sum(results_df$log2FoldChange > 0 & results_df$padj < 0.05, na.rm = TRUE)
    sig_down <- sum(results_df$log2FoldChange < 0 & results_df$padj < 0.05, na.rm = TRUE)

    cat("Number of significantly upregulated features (", group2, " > ", group1, "):", sig_up,
```

```
"\n")
        cat("Number of significantly downregulated features (", group2, " < ", group1, "):",
sig_down, "\n")

        return(list(
          results = results_df,
          ancombc_obj = ancombc_result,
          physeq = physeq_filtered,
          sig_up = sig_up,
          sig_down = sig_down,
          comparison = comparison_name,
          group1 = group1,
          group2 = group2
        ))

      }, error = function(e2) {
        cat("Failed to extract results from ultra-conservative retry:", e2$message, "\n")
        cat("Returning empty results for this comparison\n")

        return(list(
          results = data.frame(),
          ancombc_obj = NULL,
          physeq = physeq_filtered,
          sig_up = 0,
          sig_down = 0,
          comparison = comparison_name,
          group1 = group1,
          group2 = group2,
          error = paste("Analysis failed:", e$message)
        ))
      })
  })
}

# ==================== 3. Perform All Treatment Comparisons ====================

# Define comparison pairs
comparisons <- list(
  list(group1 = "Untreated", group2 = "Water-Vapor", name = "Water-Vapor_vs_Untreated"),
  list(group1 = "Thermal Disinfection", group2 = "Water-Vapor", name =
"Water-Vapor_vs_Thermal_Disinfection"),
  list(group1 = "Chemical Treatment", group2 = "Water-Vapor", name =
"Water-Vapor_vs_Chemical_Treatment"),
  list(group1 = "Chemical Treatment", group2 = "Chemical Treatment + Water-Vapor", name =
```

```
"Chemical_Treatment_Water-Vapor_vs_Chemical_Treatment")
)

# Store all results
all_results <- list()

# Perform analysis for each functional level and each comparison
for (func_level in c("EC", "KO", "Pathway")) {

  cat("\n", paste(rep("=", 50), collapse = ""), "\n")
  cat("Processing", func_level, "level analysis\n")
  cat(paste(rep("=", 50), collapse = ""), "\n")

  # Select appropriate phyloseq object
  if (func_level == "EC") {
    physeq_func <- physeq_ec_treatment
  } else if (func_level == "KO") {
    physeq_func <- physeq_ko_treatment
  } else {
    physeq_func <- physeq_pathway_treatment
  }

  # Basic data quality check
  cat("Initial data check for", func_level, "level:\n")
  cat("    Features:", ntaxa(physeq_func), "\n")
  cat("    Samples:", nsamples(physeq_func), "\n")

  # Check if we have any data
  if (ntaxa(physeq_func) == 0) {
    cat("Warning: No features found for", func_level, "level. Skipping...\n")
    next
  }

  level_results <- list()

  # Perform all treatment comparisons
  for (comp in comparisons) {
    tryCatch({
      analysis_result <- perform_treatment_ancombc(
        physeq_func,
        comp$group1,
        comp$group2,
        comp$name
      )
```

```
            level_results[[comp$name]] <- analysis_result
        }, error = function(e) {
            cat("Error in", func_level, comp$name, "analysis:", e$message, "\n")
            cat("This comparison will be skipped.\n")
            level_results[[comp$name]] <- NULL
        })
    }

    all_results[[func_level]] <- level_results
}

# ==================== 4. Save Results and Create Visualizations ====================

# 4.1 Save detailed results
for (func_level in names(all_results)) {
    for (comp_name in names(all_results[[func_level]])) {

        # Check if analysis was successful
        if (is.null(all_results[[func_level]][[comp_name]])) {
            cat("Skipping result saving for", func_level, comp_name, "- analysis failed\n")
            next
        }

        result_data <- all_results[[func_level]][[comp_name]]$results

        # Skip if no results
        if (is.null(result_data) || nrow(result_data) == 0) {
            cat("Skipping result saving for", func_level, comp_name, "- no results\n")
            next
        }

        filename <- paste0("Treatment_", func_level, "_", comp_name, "_ANCOMBC_results.csv")
        write.csv(result_data, file.path(output_dir, filename), row.names = FALSE)

        # Save significant features
        sig_features <- result_data %>%
            filter(padj < 0.05 & abs(log2FoldChange) > 0.5)

        if (nrow(sig_features) > 0) {
            sig_filename <- paste0("Treatment_", func_level, "_", comp_name,
"_significant_features.csv")
            write.csv(sig_features, file.path(output_dir, sig_filename), row.names = FALSE)
        }
    }
```

```
}

# 4.2 Create visualization functions

# Treatment volcano plot function
create_treatment_volcano_plot <- function(results_df, comparison_name, func_level, filename) {

  if (nrow(results_df) == 0) {
    cat("No results data for volcano plot:", comparison_name, func_level, "\n")
    return(NULL)
  }

  results_df <- results_df %>%
    mutate(
      Significance = case_when(
        padj < 0.05 & log2FoldChange > 0.5 ~ paste("Up in", results_df$group2[1]),
        padj < 0.05 & log2FoldChange < -0.5 ~ paste("Up in", results_df$group1[1]),
        TRUE ~ "Not Significant"
      )
    )

  top_features <- results_df %>%
    filter(Significance != "Not Significant") %>%
    arrange(padj) %>%
    head(10)

  # Create color values with proper naming
  color_up_group2 <- paste("Up in", results_df$group2[1])
  color_up_group1 <- paste("Up in", results_df$group1[1])

  color_values <- c("#d62728", "#2ca02c", "grey70")
  names(color_values) <- c(color_up_group2, color_up_group1, "Not Significant")

  p <- ggplot(results_df, aes(x = log2FoldChange, y = -log10(padj))) +
    geom_point(aes(color = Significance), alpha = 0.7, size = 1.5) +
    scale_color_manual(values = color_values) +
    geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "black", alpha = 0.5) +
    geom_vline(xintercept = c(-0.5, 0.5), linetype = "dashed", color = "black", alpha = 0.5) +
    labs(
      title = paste("Volcano Plot -", func_level, "Functions"),
      subtitle = paste("Treatment Comparison:", gsub("_", " ", comparison_name),
"(ANCOM-BC)"),
      x = paste("log2 Fold Change (", results_df$group2[1], "/", results_df$group1[1], ")"),
      y = "-log10(adjusted p-value)",
```

```
        color = "Significance"
      ) +
      theme_bw() +
      theme(
        plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
        plot.subtitle = element_text(hjust = 0.5, size = 12),
        legend.position = "bottom"
      )

  if (nrow(top_features) > 0) {
    p <- p + geom_text_repel(
      data = top_features,
      aes(label = Feature_ID),
      size = 3,
      max.overlaps = 10,
      box.padding = 0.3
    )
  }

  ggsave(filename, plot = p, width = 12, height = 8, dpi = 300)
  return(p)
}

# Treatment lollipop plot function
create_treatment_lollipop_plot <- function(results_df, comparison_name, func_level, filename,
top_n = 20) {

  if (nrow(results_df) == 0) {
    cat("No results data for lollipop plot:", comparison_name, func_level, "\n")
    return(NULL)
  }

  sig_features <- results_df %>%
    filter(padj < 0.05) %>%
    arrange(padj) %>%
    head(top_n)

  if (nrow(sig_features) == 0) {
    cat("No significant features for lollipop plot:", comparison_name, func_level, "\n")
    return(NULL)
  }

  plot_data <- sig_features %>%
    mutate(
```

```
      Direction = ifelse(log2FoldChange > 0,
                         paste("Up in", group2),
                         paste("Up in", group1)),
      Feature_ID_short = ifelse(nchar(Feature_ID) > 40,
                                paste0(substr(Feature_ID, 1, 37), "..."),
                                Feature_ID),
      abs_lfc = abs(log2FoldChange)
    ) %>%
    arrange(abs_lfc)

  plot_data$Feature_ID_short <- factor(plot_data$Feature_ID_short,
                                       levels = plot_data$Feature_ID_short)

  # Create color values with proper naming
  color_up_group2 <- paste("Up in", plot_data$group2[1])
  color_up_group1 <- paste("Up in", plot_data$group1[1])

  color_values <- c("#d62728", "#2ca02c")
  names(color_values) <- c(color_up_group2, color_up_group1)

  p <- ggplot(plot_data, aes(x = Feature_ID_short, y = log2FoldChange)) +
    geom_segment(aes(x = Feature_ID_short, xend = Feature_ID_short,
                     y = 0, yend = log2FoldChange, color = Direction),
                 size = 1) +
    geom_point(aes(color = Direction, size = -log10(padj)), alpha = 0.8) +
    scale_color_manual(values = color_values) +
    scale_size_continuous(name = "-log10(padj)", range = c(2, 8)) +
    coord_flip() +
    labs(
      title = paste("Top Significant Features -", func_level, "Functions"),
      subtitle = paste("Treatment Comparison:", gsub("_", " ", comparison_name),
"(ANCOM-BC)"),
      x = "Functional Features",
      y = paste("log2 Fold Change (", plot_data$group2[1], "/", plot_data$group1[1], ")"),
      color = "Direction",
      size = "Significance"
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      plot.subtitle = element_text(hjust = 0.5, size = 12),
      axis.text.y = element_text(size = 9),
      legend.position = "bottom"
    ) +
```

```
    geom_hline(yintercept = 0, linetype = "solid", color = "black", alpha = 0.3)

  ggsave(filename, plot = p, width = 14, height = 10, dpi = 300)
  return(p)
}

# Treatment heatmap function
create_treatment_heatmap <- function(physeq_obj, results_df, comparison_name, func_level,
filename, top_n = 30) {

  cat("\n=== Creating heatmap:", func_level, comparison_name, "===\n")

  # Check if there are results data
  if (nrow(results_df) == 0) {
    cat("No results data for heatmap:", comparison_name, func_level, "\n")
    return(NULL)
  }

  # Select top significant features
  top_features <- results_df %>%
    filter(!is.na(padj)) %>%
    arrange(padj) %>%
    head(top_n)

  if (nrow(top_features) == 0) {
    cat("No significant features for heatmap:", comparison_name, func_level, "\n")
    return(NULL)
  }

  cat("Number of candidate features:", nrow(top_features), "\n")

  # Check if feature IDs exist in phyloseq object
  available_taxa <- taxa_names(physeq_obj)
  feature_ids <- top_features$Feature_ID

  cat("Number of taxa in phyloseq object:", length(available_taxa), "\n")
  cat("Number of candidate feature IDs:", length(feature_ids), "\n")

  # Find matching features
  valid_features <- intersect(feature_ids, available_taxa)
  cat("Number of matching features:", length(valid_features), "\n")

  if (length(valid_features) == 0) {
    cat("Warning: No matching feature IDs, checking first few feature IDs:\n")
```

```
    cat("First 5 feature IDs in results:\n")
    print(head(feature_ids, 5))
    cat("First 5 taxa in phyloseq:\n")
    print(head(available_taxa, 5))
    return(NULL)
}

# Limit feature count to avoid oversized images
if (length(valid_features) > top_n) {
    valid_features <- valid_features[1:top_n]
}

cat("Final number of features for plotting:", length(valid_features), "\n")

tryCatch({
    # Extract abundance data
    otu_table_subset <- otu_table(physeq_obj)[valid_features, , drop = FALSE]
    otu_df <- as.data.frame(otu_table_subset)

    cat("Extracted data dimensions:", dim(otu_df), "\n")

    # Check if data is empty
    if (nrow(otu_df) == 0 || ncol(otu_df) == 0) {
        cat("Extracted data is empty, cannot create heatmap\n")
        return(NULL)
    }

    # Convert to relative abundance (avoid division by zero)
    col_sums <- colSums(otu_df)
    if (any(col_sums == 0)) {
        cat("Some samples have total abundance of 0, adding pseudo-counts\n")
        otu_df <- otu_df + 1
        col_sums <- colSums(otu_df)
    }

    otu_rel <- sweep(otu_df, 2, col_sums, "/") * 100

    # Log transformation and normalization (z-score)
    otu_log <- log10(otu_rel + 1e-6)
    otu_scaled <- t(scale(t(otu_log)))

    # Handle NaN values
    otu_scaled[is.nan(otu_scaled)] <- 0
    otu_scaled[is.infinite(otu_scaled)] <- 0
```

```
# Simplify feature names
rownames(otu_scaled) <- ifelse(nchar(rownames(otu_scaled)) > 50,
                               paste0(substr(rownames(otu_scaled), 1, 47), "..."),
                               rownames(otu_scaled))

# Prepare sample annotations
sample_anno <- data.frame(
  Treatment = sample_data(physeq_obj)$Treatment,
  row.names = sample_names(physeq_obj)
)

# Ensure sample_anno row names match otu_scaled column names
common_samples <- intersect(rownames(sample_anno), colnames(otu_scaled))
sample_anno <- sample_anno[common_samples, , drop = FALSE]
otu_scaled <- otu_scaled[, common_samples, drop = FALSE]

cat("Final plotting data dimensions:", dim(otu_scaled), "\n")
cat("Sample annotation dimensions:", dim(sample_anno), "\n")

# Color settings for treatment
anno_colors <- list(
  Treatment = c(
    "Untreated" = "#1f77b4",
    "Water-Vapor" = "#ff7f0e",
    "Thermal Disinfection" = "#2ca02c",
    "Chemical Treatment" = "#d62728",
    "Chemical Treatment + Water-Vapor" = "#9467bd"
  )
)

# Create heatmap
pheatmap(
  otu_scaled,
  annotation_col = sample_anno,
  annotation_colors = anno_colors,
  clustering_distance_rows = "correlation",
  clustering_distance_cols = "correlation",
  show_rownames = TRUE,
  show_colnames = TRUE,
  scale = "none",
  color = colorRampPalette(c("navy", "white", "firebrick"))(100),
  filename = filename,
  width = 16,
```

```
            height = max(8, nrow(otu_scaled) * 0.3),
            fontsize_col = 8,
            fontsize_row = 8,
            angle_col = 45,
            main = paste("Treatment Comparison:", gsub("_", " ", comparison_name), "-", func_level,
"Functions")
        )

        cat("Heatmap saved to:", filename, "\n")

    }, error = function(e) {
        cat("Error while creating heatmap:", e$message, "\n")
        cat("Trying to create simplified version of heatmap...\n")

        # Simplified version: use only first 10 features
        simple_features <- valid_features[1:min(10, length(valid_features))]

        tryCatch({
            otu_simple <- as.data.frame(otu_table(physeq_obj)[simple_features, , drop = FALSE])
            otu_simple_rel <- sweep(otu_simple, 2, colSums(otu_simple + 1), "/") * 100

            pheatmap(
                log10(otu_simple_rel + 1),
                show_rownames = TRUE,
                show_colnames = TRUE,
                filename = filename,
                width = 12,
                height = 8,
                fontsize_col = 8,
                angle_col = 45,
                main = paste("Treatment Comparison:", gsub("_", " ", comparison_name), "-",
func_level, "(Simplified)")
            )

            cat("Simplified heatmap saved to:", filename, "\n")

        }, error = function(e2) {
            cat("Simplified heatmap also failed:", e2$message, "\n")
            return(NULL)
        })
    })
}

# 4.3 Generate all plots (including heatmaps)
```

```
for (func_level in names(all_results)) {
  for (comp_name in names(all_results[[func_level]])) {

    # Check if analysis was successful
    if (is.null(all_results[[func_level]][[comp_name]])) {
      cat("Skipping visualization for", func_level, comp_name, "- analysis failed\n")
      next
    }

    result_data <- all_results[[func_level]][[comp_name]]$results
    physeq_obj <- all_results[[func_level]][[comp_name]]$physeq

    # Skip if no results
    if (is.null(result_data) || nrow(result_data) == 0) {
      cat("Skipping visualization for", func_level, comp_name, "- no results\n")
      next
    }

    # Volcano plot
    volcano_filename <- file.path(output_dir,
                                  paste0("Treatment_", func_level, "_", comp_name,
"_volcano_plot.png"))
    tryCatch({
      create_treatment_volcano_plot(result_data, comp_name, func_level, volcano_filename)
    }, error = function(e) cat("Error creating volcano plot for", func_level, comp_name, ":",
e$message, "\n"))

    # Lollipop plot
    lollipop_filename <- file.path(output_dir,
                                   paste0("Treatment_", func_level, "_", comp_name,
"_lollipop_plot.png"))
    tryCatch({
      create_treatment_lollipop_plot(result_data, comp_name, func_level, lollipop_filename)
    }, error = function(e) cat("Error creating lollipop plot for", func_level, comp_name, ":",
e$message, "\n"))

    # Heatmap
    heatmap_filename <- file.path(output_dir,
                                  paste0("Treatment_", func_level, "_", comp_name,
"_heatmap.png"))
    tryCatch({
      create_treatment_heatmap(physeq_obj, result_data, comp_name, func_level,
heatmap_filename)
    }, error = function(e) cat("Error creating heatmap for", func_level, comp_name, ":",
```

```
e$message, "\n"))
  }
}

# 4.4 Create summary comparison plot
create_treatment_summary_comparison <- function() {
  summary_data <- data.frame()

  for (func_level in names(all_results)) {
    for (comp_name in names(all_results[[func_level]])) {
      result <- all_results[[func_level]][[comp_name]]

      # Skip if analysis failed
      if (is.null(result)) {
        cat("Skipping summary for", func_level, comp_name, "- analysis failed\n")
        next
      }

      summary_data <- rbind(summary_data, data.frame(
        Functional_Level = func_level,
        Comparison = comp_name,
        Up_regulated = result$sig_up,
        Down_regulated = result$sig_down,
        Total_significant = result$sig_up + result$sig_down
      ))
    }
  }

  # Check if we have any data to plot
  if (nrow(summary_data) == 0) {
    cat("No successful analyses to summarize\n")
    return(NULL)
  }

  # Reshape for plotting
  plot_data <- summary_data %>%
    select(-Total_significant) %>%
    pivot_longer(cols = c(Up_regulated, Down_regulated),
                 names_to = "Direction", values_to = "Count") %>%
    mutate(
      Direction = gsub("_", " ", Direction),
      Comparison = gsub("_", " ", Comparison)
    )
```

```
    p <- ggplot(plot_data, aes(x = Comparison, y = Count, fill = Direction)) +
      geom_bar(stat = "identity", position = "dodge", alpha = 0.8) +
      facet_wrap(~Functional_Level, scales = "free_y") +
      scale_fill_manual(values = c("Up regulated" = "#d62728", "Down regulated" = "#2ca02c")) +
      labs(
        title = "Treatment Effects: Number of Significantly Different Functions",
        subtitle = "Treatment Comparisons (ANCOM-BC, padj < 0.05)",
        x = "Treatment Comparisons",
        y = "Number of Features",
        fill = "Direction"
      ) +
      theme_bw() +
      theme(
        plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
        plot.subtitle = element_text(hjust = 0.5, size = 12),
        axis.text.x = element_text(angle = 45, hjust = 1, size = 9),
        strip.text = element_text(size = 12, face = "bold")
      ) +
      geom_text(aes(label = Count), position = position_dodge(width = 0.9), vjust = -0.3, size = 3)

    ggsave(file.path(output_dir, "Treatment_functional_summary.png"),
           plot = p, width = 16, height = 8, dpi = 300)

    return(p)
}

summary_plot <- create_treatment_summary_comparison()

# ==================== 5. Generate Analysis Report ====================

# Create detailed summary report
report_lines <- c(
    "Treatment Methods Functional Difference Analysis Report (ANCOM-BC)",
    paste(rep("=", 70), collapse = ""),
    "",
    paste("Analysis Date:", Sys.Date()),
    "Analysis Method: ANCOM-BC (treatment comparisons)",
    "",
    "Sample Information:",
    paste("    Total Samples:", nsamples(physeq_treatment))
)

# Add sample counts for each treatment
for (treatment in names(treatment_counts)) {
```

```
  report_lines <- c(report_lines, paste("    ", treatment, ":", treatment_counts[treatment]))
}

report_lines <- c(report_lines, "", "Treatment Comparisons Performed:")
report_lines <- c(report_lines, "    1. Water-Vapor vs Untreated")
report_lines <- c(report_lines, "    2. Water-Vapor vs Thermal Disinfection")
report_lines <- c(report_lines, "    3. Water-Vapor vs Chemical Treatment")
report_lines <- c(report_lines, "    4. Chemical Treatment + Water-Vapor vs Chemical Treatment")

# Add results for each comparison and functional level
for (func_level in names(all_results)) {
  report_lines <- c(report_lines, paste("\n", func_level, "Level:"))

  # Check if this functional level has any successful results
  has_results <- FALSE

  for (comp_name in names(all_results[[func_level]])) {
    result <- all_results[[func_level]][[comp_name]]

    if (!is.null(result)) {
      has_results <- TRUE
      comp_display <- gsub("_", " ", comp_name)

      report_lines <- c(report_lines, paste("    ", comp_display, ":"))
      report_lines <- c(report_lines, paste("      Upregulated:", result$sig_up))
      report_lines <- c(report_lines, paste("      Downregulated:", result$sig_down))
      report_lines <- c(report_lines, paste("      Total significant:", result$sig_up +
result$sig_down))
    } else {
      comp_display <- gsub("_", " ", comp_name)
      report_lines <- c(report_lines, paste("    ", comp_display, ": Analysis failed"))
    }
  }

  if (!has_results) {
    report_lines <- c(report_lines, "    No successful analyses for this functional level")
  }
}

report_lines <- c(report_lines,
  "",
  "Output Files:",
  "    1. Differential Analysis Results: Treatment_*_*_ANCOMBC_results.csv",
  "    2. Significant Feature Lists: Treatment_*_*_significant_features.csv",
```

```
  "    3. Volcano Plots: Treatment_*_*_volcano_plot.png",
  "    4. Lollipop Plots: Treatment_*_*_lollipop_plot.png",
  "    5. Heatmaps: Treatment_*_*_heatmap.png",
  "    6. Summary Figure: Treatment_functional_summary.png",
  "",
  "Notes:",
  "- Analysis focuses on treatment method effects on functional composition",
  "- Enhanced data filtering removes zero-variance, low CV, and problematic features",
  "- Sample size handling: automatic parameter adjustment for small samples",
  "- Prevalence threshold: features must be present in ≥10-40% of samples (adjusted for sample
size)",
  "- Minimum total count threshold applied to ensure robust statistical analysis",
  "- Ultra-conservative retry mechanism for failed analyses with zero-variance errors",
  "- Special filtering for KO features mentioned in error messages",
  "- Upregulated indicates higher abundance in the second group of each comparison",
  "- ANCOM-BC method accounts for compositional nature of microbiome data",
  "- Heatmaps show abundance patterns of top significant features across samples",
  "- Sample names are displayed in heatmaps with treatment color-coding",
  "- Results provide insights into treatment effects on functional profiles",
  "- Conservative parameters used to ensure statistical robustness",
  "- Failed analyses are reported and skipped to allow other comparisons to proceed"
)

writeLines(report_lines, file.path(output_dir, "Treatment_analysis_summary.txt"))

cat("\n=== Treatment ANCOM-BC Analysis Completed ===\n")
cat("All results saved to:", output_dir, "\n")
cat("Please refer to Treatment_analysis_summary.txt for detailed analysis report\n")

# Display summary statistics
cat("\nQuick Summary:\n")
for (func_level in names(all_results)) {
  cat("\n", func_level, "Level:\n")
  has_results <- FALSE

  for (comp_name in names(all_results[[func_level]])) {
    result <- all_results[[func_level]][[comp_name]]

    if (!is.null(result)) {
      has_results <- TRUE
      comp_display <- gsub("_", " ", comp_name)
      cat("    ", comp_display, ": ",
          result$sig_up + result$sig_down, " significant features\n")
    }
```

```
  }

  if (!has_results) {
    cat("    No successful analyses\n")
  }
}

# ==================== Legionella-Related Functional and Taxonomic Analysis ====================
# Analysis 4: Treatment effects on Legionella-related functions and their association with bacterial genera

# Load required packages
library(phyloseq)
library(ANCOMBC)
library(ggplot2)
library(dplyr)
library(tibble)
library(pheatmap)
library(RColorBrewer)
library(ggrepel)
library(tidyr)
library(corrplot)
library(networkD3)
library(circlize)
library(ComplexHeatmap)
library(ggalluvial)
library(reshape2)
library(viridis)

# Set output path
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/legionella_analysis"
if (!dir.exists(output_dir)) {
  dir.create(output_dir, recursive = TRUE)
}

# ==================== 1. Define Legionella-Related Function Library ====================

# Define Legionella-related functional categories
legionella_functions <- list(

  # 1. Biofilm Formation
  biofilm_formation = list(
    KO = c("K02313", "K02314", "K02315", "K02652", "K02653", "K02654",
```

```
                "K01071", "K13924", "K02040", "K02041", "K02042"),
    EC = c("3.1.4.17", "4.6.1.12", "2.7.7.65", "3.2.1.4"),
    Pathway = c("PWY-5103", "PWY-6737", "PWY-6666", "PWY-5484")
  ),

  # 2. Virulence Factors
  virulence_factors = list(
    KO = c("K03197", "K03198", "K03199", "K03200", "K03201", "K03202",
           "K03203", "K03204", "K03205", "K07813", "K07814", "K12510", "K03046"),
    EC = c("3.4.21.102", "3.1.4.12", "3.5.1.1", "2.7.11.1"),
    Pathway = c("PWY-6863", "PWY-7219", "PWY-6432", "PWY-5897")
  ),

  # 3. Resistance Mechanisms
  resistance_mechanisms = list(
    KO = c("K03297", "K00558", "K00540", "K01467", "K03088", "K03327", "K07793"),
    EC = c("3.5.2.6", "1.5.1.3", "2.1.1.45", "1.11.1.6", "1.15.1.1"),
    Pathway = c("PWY-6519", "PWY-6642", "PWY-5485", "PWY-6738")
  ),

  # 4. Nutrient Metabolism
  nutrient_metabolism = list(
    KO = c("K02014", "K02015", "K02016", "K03594", "K00134", "K01689", "K00260"),
    EC = c("1.16.3.1", "1.8.1.2", "4.1.1.39", "2.3.1.12", "1.4.1.2"),
    Pathway = c("PWY-5971", "PWY-6285", "PWY-621", "PWY-6737", "PWY-5686")
  ),

  # 5. Environmental Adaptation
  environmental_adaptation = list(
    KO = c("K03040", "K04077", "K03695", "K05391", "K03406"),
    EC = c("3.4.21.92", "5.2.1.8", "1.11.1.6", "2.7.3.9", "3.1.21.3"),
    Pathway = c("PWY-6519", "PWY-5485", "PWY-6642", "PWY-6738", "PWY-5897")
  ),

  # 6. Microbial Interactions
  microbial_interactions = list(
    KO = c("K07706", "K07720", "K01992", "K02003", "K03980"),
    EC = c("2.3.1.184", "3.5.2.14", "3.4.24.40", "2.7.7.4", "1.8.1.2"),
    Pathway = c("PWY-6666", "PWY-5484", "PWY-6285", "PWY-5971", "PWY-6738")
  )
)

# Function to extract Legionella-related features from phyloseq objects
extract_legionella_features <- function(physeq_obj, func_type = "KO") {
```

```
  # Get all taxa names
  all_taxa <- taxa_names(physeq_obj)

  # Initialize results
  legionella_features <- character(0)
  feature_categories <- character(0)

  # Extract features based on type
  for (category in names(legionella_functions)) {
    category_features <- legionella_functions[[category]][[func_type]]

    if (func_type == "KO") {
      # For KO: look for ko:K##### pattern
      pattern_features <- paste0("ko:", category_features)
      matched_features <- intersect(pattern_features, all_taxa)
    } else if (func_type == "EC") {
      # For EC: look for exact EC numbers
      matched_features <- intersect(category_features, all_taxa)
    } else if (func_type == "Pathway") {
      # For Pathway: look for pathway names
      matched_features <- intersect(category_features, all_taxa)
    }

    if (length(matched_features) > 0) {
      legionella_features <- c(legionella_features, matched_features)
      feature_categories <- c(feature_categories,
                              rep(category, length(matched_features)))
    }
  }

  return(list(features = legionella_features, categories = feature_categories))
}

# ==================== 2. Data Preprocessing ====================

# Check treatment groups
cat("Checking Treatment column values:\n")
treatment_status <- sample_data(physeq)$Treatment
print(table(treatment_status, useNA = "ifany"))

# Filter valid samples
valid_samples <- !is.na(sample_data(physeq)$Treatment)
physeq_filtered <- prune_samples(valid_samples, physeq)
```

```
physeq_pathway_filtered <- prune_samples(valid_samples, physeq_pathway)
physeq_ec_filtered <- prune_samples(valid_samples, physeq_ec)
physeq_ko_filtered <- prune_samples(valid_samples, physeq_ko)

# Extract Legionella-related features for each functional level
legionella_ko <- extract_legionella_features(physeq_ko_filtered, "KO")
legionella_ec <- extract_legionella_features(physeq_ec_filtered, "EC")
legionella_pathway <- extract_legionella_features(physeq_pathway_filtered, "Pathway")

cat("Legionella-related features found:\n")
cat("KO functions:", length(legionella_ko$features), "\n")
cat("EC functions:", length(legionella_ec$features), "\n")
cat("Pathway functions:", length(legionella_pathway$features), "\n")

# ==================== 3. Genus-Level Analysis ====================

# Transform phyloseq to genus level
physeq_genus <- tax_glom(physeq_filtered, "Genus", NArm = TRUE)

# Filter low abundance genera (present in at least 10% of samples)
physeq_genus_filtered <- filter_taxa(physeq_genus, function(x) {
  sum(x > 0) >= (nsamples(physeq_genus) * 0.1)
}, TRUE)

cat("Number of genera after filtering:", ntaxa(physeq_genus_filtered), "\n")

# ==================== 4. Fixed Differential Analysis Functions ====================

perform_legionella_analysis <- function(physeq_obj, legionella_info, func_type, treatment_group)
{

  cat("\n=== Analyzing", func_type, "functions for", treatment_group, "===\n")

  # Filter for Legionella-related features
  if (length(legionella_info$features) == 0) {
    cat("No Legionella-related features found for", func_type, "\n")
    return(NULL)
  }

  physeq_leg <- prune_taxa(legionella_info$features, physeq_obj)
  cat("Analyzing", ntaxa(physeq_leg), "Legionella-related", func_type, "features\n")

  # Filter samples for treatment comparison
  target_samples <- sample_data(physeq_leg)$Treatment %in% c("Untreated",
```

```
treatment_group)
  physeq_comp <- prune_samples(target_samples, physeq_leg)

  # Remove zero-abundance features
  physeq_comp <- prune_taxa(taxa_sums(physeq_comp) > 0, physeq_comp)

  if (ntaxa(physeq_comp) == 0) {
    cat("No features remaining after filtering\n")
    return(NULL)
  }

  cat("Features after filtering:", ntaxa(physeq_comp), "\n")

  # Create comparison factor
  sample_data(physeq_comp)$Comparison <- factor(
    sample_data(physeq_comp)$Treatment,
    levels = c("Untreated", treatment_group)
  )

  # Store original taxa names BEFORE ANCOM-BC analysis
  original_taxa_names <- taxa_names(physeq_comp)
  cat("Original taxa count:", length(original_taxa_names), "\n")

  # Run ANCOM-BC analysis
  tryCatch({
    ancombc_result <- ancombc2(
      data = physeq_comp,
      assay_name = "counts",
      tax_level = NULL,
      fix_formula = "Comparison",
      rand_formula = NULL,
      p_adj_method = "fdr",
      pseudo_sens = FALSE,
      prv_cut = 0.1,
      lib_cut = 10,
      s0_perc = 0.1,
      group = "Comparison",
      struc_zero = FALSE,
      neg_lb = FALSE,
      alpha = 0.05,
      n_cl = 1,
      verbose = FALSE
    )
```

```
# Extract results
res_data <- ancombc_result$res
cat("ANCOM-BC result dimensions:", nrow(res_data), "rows\n")

# IMPORTANT: Get taxa names that ANCOM-BC actually analyzed
# ANCOM-BC may filter out some taxa, so we need to use the row names from results
ancombc_taxa_names <- rownames(res_data)
cat("ANCOM-BC analyzed taxa count:", length(ancombc_taxa_names), "\n")

# Find correct column names
lfc_cols <- grep("lfc", names(res_data), value = TRUE)
qval_cols <- grep("q_", names(res_data), value = TRUE)

cat("Available LFC columns:", paste(lfc_cols, collapse = ", "), "\n")
cat("Available q-value columns:", paste(qval_cols, collapse = ", "), "\n")

# Build results data frame using ANCOM-BC taxa names
results_df <- data.frame(
  Feature_ID = ancombc_taxa_names,
  stringsAsFactors = FALSE
)

# Extract log fold change - use the most specific column
if (length(lfc_cols) > 0) {
  # Look for treatment-specific column first
  treatment_clean <- gsub("[^A-Za-z0-9]", "", treatment_group)
  lfc_col <- grep(paste0("Comparison", treatment_clean), lfc_cols, value = TRUE)[1]

  if (is.na(lfc_col)) {
    # Try simpler pattern
    lfc_col <- grep(treatment_clean, lfc_cols, value = TRUE)[1]
  }

  if (is.na(lfc_col)) {
    # Use the last LFC column as fallback
    lfc_col <- lfc_cols[length(lfc_cols)]
  }

  cat("Using LFC column:", lfc_col, "\n")

  # Ensure dimensions match
  if (length(res_data[[lfc_col]]) == nrow(results_df)) {
    results_df$log2FoldChange <- res_data[[lfc_col]]
  } else {
```

```
            cat("Warning: LFC dimension mismatch. Expected:", nrow(results_df), "Got:", length(res_data[[lfc_col]]), "\n")
            # Use minimum dimension to prevent errors
            min_dim <- min(length(res_data[[lfc_col]]), nrow(results_df))
            results_df <- results_df[1:min_dim, , drop = FALSE]
            results_df$log2FoldChange <- res_data[[lfc_col]][1:min_dim]
        }
    } else {
        results_df$log2FoldChange <- rep(0, nrow(results_df))
    }

    # Extract adjusted p-values - use the most specific column
    if (length(qval_cols) > 0) {
        # Look for treatment-specific column first
        treatment_clean <- gsub("[^A-Za-z0-9]", "", treatment_group)
        qval_col <- grep(paste0("Comparison", treatment_clean), qval_cols, value = TRUE)[1]

        if (is.na(qval_col)) {
            # Try simpler pattern
            qval_col <- grep(treatment_clean, qval_cols, value = TRUE)[1]
        }

        if (is.na(qval_col)) {
            # Use the last q-value column as fallback
            qval_col <- qval_cols[length(qval_cols)]
        }

        cat("Using q-value column:", qval_col, "\n")

        # Ensure dimensions match
        if (length(res_data[[qval_col]]) == nrow(results_df)) {
            results_df$padj <- res_data[[qval_col]]
        } else {
            cat("Warning: q-value dimension mismatch. Expected:", nrow(results_df), "Got:", length(res_data[[qval_col]]), "\n")
            # Use minimum dimension to prevent errors
            min_dim <- min(length(res_data[[qval_col]]), nrow(results_df))
            results_df <- results_df[1:min_dim, , drop = FALSE]
            results_df$padj <- res_data[[qval_col]][1:min_dim]
        }
    } else {
        results_df$padj <- rep(1, nrow(results_df))
    }
```

```
    # Add functional categories - only for features that were actually analyzed
    feature_categories <- rep(NA, nrow(results_df))
    for (i in 1:nrow(results_df)) {
      feature_match <- match(results_df$Feature_ID[i], legionella_info$features)
      if (!is.na(feature_match)) {
        feature_categories[i] <- legionella_info$categories[feature_match]
      }
    }

    results_df$Category <- feature_categories
    results_df$FuncType <- func_type
    results_df$Treatment <- treatment_group

    # Add significance
    results_df$significant <- !is.na(results_df$padj) & results_df$padj < 0.05 &
                                        !is.na(results_df$log2FoldChange)                    &
is.finite(results_df$log2FoldChange)

    # Remove rows with NA or infinite values
    results_df <- results_df %>%
      filter(!is.na(log2FoldChange) & !is.na(padj) &
               is.finite(log2FoldChange) & is.finite(padj))

    cat("Final results:", nrow(results_df), "features\n")
    cat("Significant results:", sum(results_df$significant, na.rm = TRUE), "\n")

    return(results_df)

  }, error = function(e) {
    cat("Error in ANCOM-BC analysis:", e$message, "\n")

    # Fallback: simple statistical analysis
    cat("Attempting fallback analysis...\n")

    tryCatch({
      # Get abundance matrix
      otu_matrix <- as.matrix(otu_table(physeq_comp))
      sample_groups <- sample_data(physeq_comp)$Comparison

      # Calculate means for each group
      untreated_samples <- sample_groups == "Untreated"
      treatment_samples <- sample_groups == treatment_group

      if (sum(untreated_samples) == 0 || sum(treatment_samples) == 0) {
```

```
    cat("Insufficient samples for comparison\n")
    return(NULL)
}

untreated_means <- rowMeans(otu_matrix[, untreated_samples, drop = FALSE])
treatment_means <- rowMeans(otu_matrix[, treatment_samples, drop = FALSE])

# Calculate log2 fold changes (add pseudo-count)
log2fc <- log2((treatment_means + 1) / (untreated_means + 1))

# Perform Wilcoxon test for p-values
p_values <- apply(otu_matrix, 1, function(x) {
    tryCatch({
        if (sum(untreated_samples) >= 2 && sum(treatment_samples) >= 2) {
            wilcox.test(x[untreated_samples], x[treatment_samples])$p.value
        } else {
            1    # No significant difference for small samples
        }
    }, error = function(e) 1)
})

# Adjust p-values
p_adj <- p.adjust(p_values, method = "fdr")

# Build results using original taxa names
results_df <- data.frame(
    Feature_ID = original_taxa_names,
    log2FoldChange = as.numeric(log2fc),
    padj = as.numeric(p_adj),
    stringsAsFactors = FALSE
)

# Add functional categories
feature_categories <- rep(NA, nrow(results_df))
for (i in 1:nrow(results_df)) {
    feature_match <- match(results_df$Feature_ID[i], legionella_info$features)
    if (!is.na(feature_match)) {
        feature_categories[i] <- legionella_info$categories[feature_match]
    }
}

results_df$Category <- feature_categories
results_df$FuncType <- func_type
results_df$Treatment <- treatment_group
```

```
            results_df$significant <- results_df$padj < 0.05 & !is.na(results_df$padj)

            # Remove NA rows
            results_df <- results_df %>%
              filter(!is.na(log2FoldChange) & !is.na(padj) &
                       is.finite(log2FoldChange) & is.finite(padj))

            cat("Fallback analysis completed:", nrow(results_df), "features\n")
            cat("Significant results:", sum(results_df$significant, na.rm = TRUE), "\n")

            return(results_df)

        }, error = function(e2) {
            cat("Fallback analysis also failed:", e2$message, "\n")
            return(NULL)
        })
    })
}

# Enhanced genus-level analysis with better dimension handling
perform_genus_analysis <- function(physeq_genus, treatment_group) {

    cat("\n=== Analyzing genus changes for", treatment_group, "===\n")

    # Filter samples for treatment comparison
    target_samples <- sample_data(physeq_genus)$Treatment %in% c("Untreated",
treatment_group)
    physeq_comp <- prune_samples(target_samples, physeq_genus)

    # Enhanced filtering: remove zero-abundance and zero-variance genera
    physeq_comp <- prune_taxa(taxa_sums(physeq_comp) > 0, physeq_comp)

    # Check for zero-variance genera
    otu_matrix <- as.matrix(otu_table(physeq_comp))
    feature_vars <- apply(otu_matrix, 1, function(x) {
        if (all(x == 0)) return(0)
        if (length(unique(x)) <= 1) return(0)
        return(var(x, na.rm = TRUE))
    })

    # Remove zero-variance genera
    nonzero_var_features <- names(feature_vars)[feature_vars > 0 & !is.na(feature_vars)]
    if (length(nonzero_var_features) == 0) {
        cat("No genera with variance for", treatment_group, "\n")
```

```
    return(NULL)
}

physeq_comp <- prune_taxa(nonzero_var_features, physeq_comp)

if (ntaxa(physeq_comp) == 0) {
    cat("No genera remaining after filtering\n")
    return(NULL)
}

cat("Genera after variance filtering:", ntaxa(physeq_comp), "\n")

# Create comparison factor
sample_data(physeq_comp)$Comparison <- factor(
    sample_data(physeq_comp)$Treatment,
    levels = c("Untreated", treatment_group)
)

# Store original taxa names BEFORE ANCOM-BC analysis
original_taxa_names <- taxa_names(physeq_comp)
cat("Original genera count:", length(original_taxa_names), "\n")

# Run ANCOM-BC analysis with enhanced error handling
tryCatch({
    ancombc_result <- ancombc2(
        data = physeq_comp,
        assay_name = "counts",
        tax_level = NULL,
        fix_formula = "Comparison",
        rand_formula = NULL,
        p_adj_method = "fdr",
        pseudo_sens = FALSE,    # Turn off for small datasets
        prv_cut = 0.05,             # More lenient prevalence cut
        lib_cut = 1,                # Very low library cut
        s0_perc = 0.2,              # Higher s0_perc for stability
        group = "Comparison",
        struc_zero = FALSE,
        neg_lb = FALSE,
        alpha = 0.1,                # More lenient alpha
        n_cl = 1,
        verbose = FALSE
    )

    # Extract results with careful dimension checking
```

```
res_data <- ancombc_result$res
cat("ANCOM-BC result dimensions:", nrow(res_data), "rows\n")

# IMPORTANT: Get taxa names that ANCOM-BC actually analyzed
ancombc_taxa_names <- rownames(res_data)
cat("ANCOM-BC analyzed genera count:", length(ancombc_taxa_names), "\n")

# Find correct column names
lfc_cols <- grep("lfc", names(res_data), value = TRUE)
qval_cols <- grep("q_", names(res_data), value = TRUE)

cat("Available LFC columns:", paste(lfc_cols, collapse = ", "), "\n")
cat("Available q-value columns:", paste(qval_cols, collapse = ", "), "\n")

# Build results data frame using ANCOM-BC taxa names
results_df <- data.frame(
  Feature_ID = ancombc_taxa_names,
  stringsAsFactors = FALSE
)

# Extract log fold change safely
if (length(lfc_cols) > 0) {
  treatment_clean <- gsub("[^A-Za-z0-9]", "", treatment_group)
  lfc_col <- grep(paste0("Comparison", treatment_clean), lfc_cols, value = TRUE)[1]

  if (is.na(lfc_col)) {
    lfc_col <- grep(treatment_clean, lfc_cols, value = TRUE)[1]
  }

  if (is.na(lfc_col)) {
    lfc_col <- lfc_cols[length(lfc_cols)]  # Use last LFC column
  }

  cat("Using LFC column:", lfc_col, "\n")

  # Ensure dimensions match
  if (length(res_data[[lfc_col]]) == nrow(results_df)) {
    results_df$log2FoldChange <- res_data[[lfc_col]]
  } else {
    cat("Warning: LFC dimension mismatch. Expected:", nrow(results_df), "Got:",
length(res_data[[lfc_col]]), "\n")
    min_dim <- min(length(res_data[[lfc_col]]), nrow(results_df))
    results_df <- results_df[1:min_dim, , drop = FALSE]
    results_df$log2FoldChange <- res_data[[lfc_col]][1:min_dim]
```

```
    }
  } else {
    results_df$log2FoldChange <- rep(0, nrow(results_df))
  }

  # Extract adjusted p-values safely
  if (length(qval_cols) > 0) {
    treatment_clean <- gsub("[^A-Za-z0-9]", "", treatment_group)
    qval_col <- grep(paste0("Comparison", treatment_clean), qval_cols, value = TRUE)[1]

    if (is.na(qval_col)) {
      qval_col <- grep(treatment_clean, qval_cols, value = TRUE)[1]
    }

    if (is.na(qval_col)) {
      qval_col <- qval_cols[length(qval_cols)]    # Use last q-value column
    }

    cat("Using q-value column:", qval_col, "\n")

    # Ensure dimensions match
    if (length(res_data[[qval_col]]) == nrow(results_df)) {
      results_df$padj <- res_data[[qval_col]]
    } else {
      cat("Warning: q-value dimension mismatch. Expected:", nrow(results_df), "Got:",
length(res_data[[qval_col]]), "\n")
      min_dim <- min(length(res_data[[qval_col]]), nrow(results_df))
      results_df <- results_df[1:min_dim, , drop = FALSE]
      results_df$padj <- res_data[[qval_col]][1:min_dim]
    }
  } else {
    results_df$padj <- rep(1, nrow(results_df))
  }

  # Add genus names from taxonomy - match only the analyzed features
  tax_table_df <- as.data.frame(tax_table(physeq_comp))

  # Initialize Genus column
  results_df$Genus <- NA

  # Match analyzed features to taxonomy
  for (i in 1:nrow(results_df)) {
    feature_id <- results_df$Feature_ID[i]
    if (feature_id %in% rownames(tax_table_df)) {
```

```
            results_df$Genus[i] <- tax_table_df[feature_id, "Genus"]
          }
        }

        results_df$Treatment <- treatment_group

        # Add significance
        results_df$significant <- !is.na(results_df$padj) & results_df$padj < 0.05 &
                                  !is.na(results_df$log2FoldChange) &
is.finite(results_df$log2FoldChange)

        # Remove rows with NA or infinite values
        results_df <- results_df %>%
          filter(!is.na(log2FoldChange) & !is.na(padj) &
                   is.finite(log2FoldChange) & is.finite(padj))

        cat("Final results:", nrow(results_df), "genera\n")
        cat("Significant results:", sum(results_df$significant, na.rm = TRUE), "\n")

        return(results_df)

      }, error = function(e) {
        cat("Error in genus analysis:", e$message, "\n")
        cat("Attempting fallback analysis...\n")

        # Fallback: simple fold change analysis
        tryCatch({
          # Calculate simple fold changes
          otu_matrix <- as.matrix(otu_table(physeq_comp))
          sample_groups <- sample_data(physeq_comp)$Comparison

          # Calculate mean abundances for each group
          untreated_samples <- sample_groups == "Untreated"
          treatment_samples <- sample_groups == treatment_group

          if (sum(untreated_samples) == 0 || sum(treatment_samples) == 0) {
            cat("Insufficient samples for comparison\n")
            return(NULL)
          }

          untreated_means <- rowMeans(otu_matrix[, untreated_samples, drop = FALSE])
          treatment_means <- rowMeans(otu_matrix[, treatment_samples, drop = FALSE])

          # Calculate log2 fold changes (add pseudo-count)
```

```
        log2fc <- log2((treatment_means + 1) / (untreated_means + 1))

        # Perform Wilcoxon test for p-values
        p_values <- apply(otu_matrix, 1, function(x) {
          tryCatch({
            if (sum(untreated_samples) >= 2 && sum(treatment_samples) >= 2) {
              wilcox.test(x[untreated_samples], x[treatment_samples])$p.value
            } else {
              1  # No significant difference for small samples
            }
          }, error = function(e) 1)
        })

        # Adjust p-values
        p_adj <- p.adjust(p_values, method = "fdr")

        # Build results using original taxa names
        results_df <- data.frame(
          Feature_ID = original_taxa_names,
          log2FoldChange = as.numeric(log2fc),
          padj = as.numeric(p_adj),
          stringsAsFactors = FALSE
        )

        # Add genus names
        tax_table_df <- as.data.frame(tax_table(physeq_comp))
        results_df$Genus <- tax_table_df$Genus[match(results_df$Feature_ID,
rownames(tax_table_df))]
        results_df$Treatment <- treatment_group
        results_df$significant <- results_df$padj < 0.05 & !is.na(results_df$padj)

        # Remove NA rows
        results_df <- results_df %>%
          filter(!is.na(log2FoldChange) & !is.na(padj) &
                   is.finite(log2FoldChange) & is.finite(padj))

        cat("Fallback analysis completed:", nrow(results_df), "genera\n")
        cat("Significant results:", sum(results_df$significant, na.rm = TRUE), "\n")

        return(results_df)

      }, error = function(e2) {
        cat("Fallback analysis also failed:", e2$message, "\n")
        return(NULL)
```

```
    })
  })
}

# ==================== 5. Run All Analyses ====================

# Define treatment groups
treatments <- c("Water-Vapor", "Thermal Disinfection", "Chemical Treatment", "Chemical Treatment + Water-Vapor")

# Store all results
all_func_results <- list()
all_genus_results <- list()

# Perform functional analyses
for (treatment in treatments) {
  cat("\n", paste(rep("=", 50), collapse = ""), "\n")
  cat("Processing", treatment, "\n")
  cat(paste(rep("=", 50), collapse = ""), "\n")

  # Functional analyses
  ko_results <- perform_legionella_analysis(physeq_ko_filtered, legionella_ko, "KO", treatment)
  ec_results <- perform_legionella_analysis(physeq_ec_filtered, legionella_ec, "EC", treatment)
  pathway_results <- perform_legionella_analysis(physeq_pathway_filtered, legionella_pathway, "Pathway", treatment)

  # Store functional results
  all_func_results[[treatment]] <- list(
    KO = ko_results,
    EC = ec_results,
    Pathway = pathway_results
  )

  # Genus analysis
  genus_results <- perform_genus_analysis(physeq_genus_filtered, treatment)
  all_genus_results[[treatment]] <- genus_results
}

# ==================== 6. Enhanced Correlation Analysis ====================

# Enhanced correlation analysis with proper NA handling
calculate_correlations <- function(physeq_genus, physeq_func, legionella_features) {

  # Get common samples
```

```
  common_samples <- intersect(sample_names(physeq_genus), sample_names(physeq_func))

  if (length(common_samples) < 5) {
    cat("Insufficient common samples for correlation analysis:", length(common_samples), "\n")
    return(NULL)
  }

  # Filter to common samples
  physeq_genus_common <- prune_samples(common_samples, physeq_genus)
  physeq_func_common <- prune_samples(common_samples, physeq_func)

  # Filter to Legionella-related functions
  if (length(legionella_features) > 0) {
    legionella_features_present <- intersect(legionella_features,
taxa_names(physeq_func_common))
    if (length(legionella_features_present) > 0) {
      physeq_func_common <- prune_taxa(legionella_features_present,
physeq_func_common)
    } else {
      cat("No Legionella features present for correlation\n")
      return(NULL)
    }
  } else {
    cat("No Legionella features provided for correlation\n")
    return(NULL)
  }

  # Get abundance matrices
  genus_matrix <- as.matrix(otu_table(physeq_genus_common))
  func_matrix <- as.matrix(otu_table(physeq_func_common))

  # Transform to relative abundance
  genus_rel <- sweep(genus_matrix, 2, colSums(genus_matrix), "/")
  func_rel <- sweep(func_matrix, 2, colSums(func_matrix), "/")

  # Enhanced filtering: remove features with zero variance or all zeros
  genus_vars <- apply(genus_rel, 1, function(x) {
    x_clean <- x[!is.na(x) & is.finite(x)]    # Remove NA and infinite values
    if (length(x_clean) < 3) return(0)          # Need at least 3 values
    if (all(x_clean == 0)) return(0)            # All zeros
    if (length(unique(x_clean)) <= 1) return(0)    # No variance
    return(var(x_clean, na.rm = TRUE))
  })
```

```
  func_vars <- apply(func_rel, 1, function(x) {
    x_clean <- x[!is.na(x) & is.finite(x)]    # Remove NA and infinite values
    if (length(x_clean) < 3) return(0)            # Need at least 3 values
    if (all(x_clean == 0)) return(0)              # All zeros
    if (length(unique(x_clean)) <= 1) return(0)    # No variance
    return(var(x_clean, na.rm = TRUE))
  })

  # Filter out problematic features
  genus_good <- names(genus_vars)[genus_vars > 0 & !is.na(genus_vars) & is.finite(genus_vars)]
  func_good <- names(func_vars)[func_vars > 0 & !is.na(func_vars) & is.finite(func_vars)]

  if (length(genus_good) == 0 || length(func_good) == 0) {
    cat("No features with adequate variance for correlation analysis\n")
    return(NULL)
  }

  # Subset to good features
  genus_rel_filtered <- genus_rel[genus_good, , drop = FALSE]
  func_rel_filtered <- func_rel[func_good, , drop = FALSE]

  # Remove any remaining NA or infinite values
  genus_rel_filtered[!is.finite(genus_rel_filtered)] <- 0
  func_rel_filtered[!is.finite(func_rel_filtered)] <- 0

  cat("Correlation analysis: ", nrow(genus_rel_filtered), "genera vs", nrow(func_rel_filtered),
"functions\n")

  # Calculate correlations safely
  tryCatch({
    cor_matrix <- cor(t(genus_rel_filtered), t(func_rel_filtered),
                      method = "spearman", use = "pairwise.complete.obs")

    # Check if correlation matrix is valid
    if (all(is.na(cor_matrix))) {
      cat("Correlation matrix is all NA\n")
      return(NULL)
    }

    # Convert to long format
    cor_df <- reshape2::melt(cor_matrix)
    colnames(cor_df) <- c("Genus_ID", "Function_ID", "Correlation")

    # Remove NA correlations and ensure finite values
```

```
        cor_df <- cor_df %>%
          filter(!is.na(Correlation) & is.finite(Correlation) & abs(Correlation) > 0)

        if (nrow(cor_df) == 0) {
          cat("No valid correlations calculated\n")
          return(NULL)
        }

        # Add genus names safely
        tax_table_df <- as.data.frame(tax_table(physeq_genus_common))

        # Initialize Genus column
        cor_df$Genus <- NA

        # Match genus IDs to names
        for (i in 1:nrow(cor_df)) {
          genus_id <- as.character(cor_df$Genus_ID[i])
          if (genus_id %in% rownames(tax_table_df)) {
            genus_name <- tax_table_df[genus_id, "Genus"]
            if (!is.na(genus_name) && genus_name != "") {
              cor_df$Genus[i] <- genus_name
            }
          }
        }

        # Filter strong correlations and remove entries without genus names
        cor_df_filtered <- cor_df %>%
          filter(abs(Correlation) > 0.5 & !is.na(Genus) & Genus != "")

        cat("Strong correlations found:", nrow(cor_df_filtered), "\n")

        return(cor_df_filtered)

      }, error = function(e) {
        cat("Error in correlation calculation:", e$message, "\n")
        return(NULL)
      })
    }

    # Calculate correlations for each treatment with enhanced error handling
    all_correlations <- list()

    for (treatment in treatments) {
      cat("\n=== Calculating correlations for", treatment, "===\n")
```

```
  # Check sample size first
  treatment_samples <- sample_data(physeq_genus_filtered)$Treatment == treatment

  if (sum(treatment_samples) < 5) {
    cat("Insufficient samples for", treatment, ":", sum(treatment_samples), "\n")
    all_correlations[[treatment]] <- list(KO = NULL, EC = NULL, Pathway = NULL)
    next
  }

  physeq_genus_treat <- prune_samples(treatment_samples, physeq_genus_filtered)

  # Calculate correlations for each functional type
  ko_corr <- NULL
  ec_corr <- NULL
  pathway_corr <- NULL

  # KO correlations
  tryCatch({
    physeq_ko_treat <- prune_samples(treatment_samples, physeq_ko_filtered)
    ko_corr <- calculate_correlations(physeq_genus_treat, physeq_ko_treat,
legionella_ko$features)
  }, error = function(e) {
    cat("KO correlation failed:", e$message, "\n")
    ko_corr <<- NULL
  })

  # EC correlations
  tryCatch({
    physeq_ec_treat <- prune_samples(treatment_samples, physeq_ec_filtered)
    ec_corr <- calculate_correlations(physeq_genus_treat, physeq_ec_treat,
legionella_ec$features)
  }, error = function(e) {
    cat("EC correlation failed:", e$message, "\n")
    ec_corr <<- NULL
  })

  # Pathway correlations
  tryCatch({
    physeq_pathway_treat <- prune_samples(treatment_samples, physeq_pathway_filtered)
    pathway_corr <- calculate_correlations(physeq_genus_treat, physeq_pathway_treat,
legionella_pathway$features)
  }, error = function(e) {
    cat("Pathway correlation failed:", e$message, "\n")
```

```
        pathway_corr <<- NULL
    })

    all_correlations[[treatment]] <- list(
        KO = ko_corr,
        EC = ec_corr,
        Pathway = pathway_corr
    )
}

# ==================== 7. Visualization Functions ====================

# Enhanced volcano plot for Legionella functions
create_legionella_volcano <- function(results_list, treatment, filename) {

    # Combine all functional results
    combined_results <- data.frame()

    for (func_type in names(results_list)) {
        if (!is.null(results_list[[func_type]]) && nrow(results_list[[func_type]]) > 0) {
            temp_data <- results_list[[func_type]]
            # Ensure all required columns exist
            if (all(c("log2FoldChange", "padj", "Category") %in% colnames(temp_data))) {
                combined_results <- rbind(combined_results, temp_data)
            }
        }
    }

    if (nrow(combined_results) == 0) {
        cat("No data for volcano plot:", treatment, "\n")
        return(NULL)
    }

    # Clean data
    combined_results <- combined_results %>%
        filter(!is.na(log2FoldChange) & !is.na(padj) &
                   is.finite(log2FoldChange) & is.finite(padj) &
                   padj > 0) %>%    # Remove zero p-values which cause -log10 issues
        mutate(
            Significance = case_when(
                padj < 0.05 & log2FoldChange > 1 ~ "Significantly Up",
                padj < 0.05 & log2FoldChange < -1 ~ "Significantly Down",
                TRUE ~ "Not Significant"
            )
```

```
        )

    if (nrow(combined_results) == 0) {
        cat("No valid data remaining for volcano plot:", treatment, "\n")
        return(NULL)
    }

    # Create plot
    tryCatch({
        p <- ggplot(combined_results, aes(x = log2FoldChange, y = -log10(padj))) +
            geom_point(aes(color = Category, size = abs(log2FoldChange), alpha = Significance)) +
            scale_color_brewer(type = "qual", palette = "Set3") +
            scale_alpha_manual(values = c("Significantly Up" = 1, "Significantly Down" = 1, "Not Significant" = 0.3)) +
            scale_size_continuous(range = c(1, 4)) +
            geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "red") +
            geom_vline(xintercept = c(-1, 1), linetype = "dashed", color = "red") +
            facet_wrap(~FuncType, scales = "free") +
            labs(
                title = paste("Legionella-Related Functions:", treatment, "vs Untreated"),
                x = "log2 Fold Change",
                y = "-log10(adjusted p-value)",
                color = "Functional Category",
                size = "|log2FC|",
                alpha = "Significance"
            ) +
            theme_bw() +
            theme(
                plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
                legend.position = "bottom",
                strip.text = element_text(face = "bold")
            )

        ggsave(filename, plot = p, width = 15, height = 10, dpi = 300)
        cat("Volcano plot saved:", filename, "\n")
        return(p)
    }, error = function(e) {
        cat("Error creating volcano plot for", treatment, ":", e$message, "\n")
        return(NULL)
    })
}

# Fixed heatmap function with proper color handling
create_legionella_heatmap <- function(all_results, filename) {
```

```
# Combine all results
combined_data <- data.frame()

for (treatment in names(all_results)) {
  for (func_type in names(all_results[[treatment]])) {
    if (!is.null(all_results[[treatment]][[func_type]])) {
      temp_data <- all_results[[treatment]][[func_type]]
      temp_data$Treatment <- treatment
      temp_data$FuncType <- func_type
      combined_data <- rbind(combined_data, temp_data)
    }
  }
}

if (nrow(combined_data) == 0) {
  cat("No data for heatmap\n")
  return(NULL)
}

# Filter significant results
sig_data <- combined_data %>%
  filter(significant == TRUE & !is.na(log2FoldChange) & is.finite(log2FoldChange)) %>%
  select(Feature_ID, Treatment, FuncType, log2FoldChange, Category) %>%
  filter(!is.na(Category))

if (nrow(sig_data) == 0) {
  cat("No significant results for heatmap\n")
  return(NULL)
}

cat("Creating heatmap with", nrow(sig_data), "significant results\n")

# Create matrix for heatmap
heatmap_matrix <- sig_data %>%
  mutate(Treatment_Type = paste(Treatment, FuncType, sep = "_")) %>%
  select(Feature_ID, Treatment_Type, log2FoldChange) %>%
  spread(Treatment_Type, log2FoldChange, fill = 0) %>%
  column_to_rownames("Feature_ID") %>%
  as.matrix()

if (nrow(heatmap_matrix) == 0 || ncol(heatmap_matrix) == 0) {
  cat("Empty heatmap matrix\n")
  return(NULL)
```

```
}

# Create row annotations
row_annotation <- sig_data %>%
  select(Feature_ID, Category) %>%
  distinct() %>%
  column_to_rownames("Feature_ID")

# Ensure row annotation matches matrix rows
row_annotation <- row_annotation[rownames(heatmap_matrix), , drop = FALSE]

# Create column annotations
col_annotation <- data.frame(
  Treatment = gsub("_.*", "", colnames(heatmap_matrix)),
  FuncType = gsub(".*_", "", colnames(heatmap_matrix)),
  stringsAsFactors = FALSE
)
rownames(col_annotation) <- colnames(heatmap_matrix)

# Create color palettes
unique_categories <- unique(row_annotation$Category)
unique_treatments <- unique(col_annotation$Treatment)
unique_functypes <- unique(col_annotation$FuncType)

# Generate colors safely
if (length(unique_categories) > 0) {
  if (length(unique_categories) <= 8) {
    category_colors <- brewer.pal(max(3, length(unique_categories)),
"Set3")[1:length(unique_categories)]
  } else {
    category_colors <- rainbow(length(unique_categories))
  }
  names(category_colors) <- unique_categories
} else {
  category_colors <- c("Unknown" = "grey")
}

# Fixed color specification
annotation_colors <- list(
  Category = category_colors,
  Treatment = c("Water-Vapor" = "#FF9999",
                "Thermal Disinfection" = "#66B2FF",
                "Chemical Treatment" = "#99FF99",
                "Chemical Treatment + Water-Vapor" = "#FFCC99")[unique_treatments],
```

```
    FuncType = c("KO" = "#FFB366",
                 "EC" = "#B3B3FF",
                 "Pathway" = "#B3FFB3")[unique_functypes]
)

# Remove NULL elements from annotation_colors
annotation_colors <- annotation_colors[!sapply(annotation_colors, is.null)]
annotation_colors <- lapply(annotation_colors, function(x) x[!is.na(x)])

# Create heatmap
tryCatch({
    pheatmap(
        heatmap_matrix,
        annotation_row = row_annotation,
        annotation_col = col_annotation,
        annotation_colors = annotation_colors,
        color = colorRampPalette(c("blue", "white", "red"))(100),
        clustering_distance_rows = "correlation",
        clustering_distance_cols = "correlation",
        show_rownames = TRUE,
        show_colnames = TRUE,
        filename = filename,
        width = 12,
        height = 10,
        fontsize_row = 8,
        fontsize_col = 10,
        main = "Legionella-Related Functions: Treatment Effects"
    )
    cat("Heatmap saved successfully:", filename, "\n")
}, error = function(e) {
    cat("Error creating heatmap:", e$message, "\n")

    # Try simplified heatmap
    tryCatch({
        pheatmap(
            heatmap_matrix,
            color = colorRampPalette(c("blue", "white", "red"))(100),
            show_rownames = TRUE,
            show_colnames = TRUE,
            filename = filename,
            width = 12,
            height = 10,
            fontsize_row = 8,
            fontsize_col = 10,
```

```
            main = "Legionella-Related Functions: Treatment Effects"
        )
        cat("Simplified heatmap saved:", filename, "\n")
    }, error = function(e2) {
        cat("Both heatmap attempts failed:", e2$message, "\n")
    })
  })
}

# Enhanced Sankey diagram for treatment-genus-function relationships
create_sankey_diagram <- function(func_results, genus_results, corr_results, treatment, filename) {

  cat("Creating Sankey diagram for", treatment, "\n")

  # Check if we have any results
  if (is.null(genus_results) || nrow(genus_results) == 0) {
    cat("No genus results for Sankey diagram:", treatment, "\n")
    return(NULL)
  }

  # Get significant functional changes
  sig_functions <- data.frame()
  for (func_type in names(func_results)) {
    if (!is.null(func_results[[func_type]]) && nrow(func_results[[func_type]]) > 0) {
      temp_func <- func_results[[func_type]] %>%
        filter(significant == TRUE & !is.na(log2FoldChange) &
                 is.finite(log2FoldChange) & abs(log2FoldChange) > 0.5) %>%
        mutate(FuncType = func_type) %>%
        select(Feature_ID, Category, log2FoldChange, FuncType) %>%
        filter(!is.na(Category))
      sig_functions <- rbind(sig_functions, temp_func)
    }
  }

  # Get significant genus changes
  sig_genera <- genus_results %>%
    filter(significant == TRUE & !is.na(log2FoldChange) &
             is.finite(log2FoldChange) & abs(log2FoldChange) > 0.5) %>%
    select(Genus, log2FoldChange) %>%
    filter(!is.na(Genus) & Genus != "")

  cat("Significant functions:", nrow(sig_functions), "\n")
  cat("Significant genera:", nrow(sig_genera), "\n")
```

```
if (nrow(sig_functions) == 0 && nrow(sig_genera) == 0) {
  cat("No significant changes for Sankey diagram:", treatment, "\n")
  return(NULL)
}

# Create Sankey data
sankey_data <- data.frame()

# Treatment to Genus connections
if (nrow(sig_genera) > 0) {
  for (i in 1:nrow(sig_genera)) {
    sankey_data <- rbind(sankey_data, data.frame(
      source = treatment,
      target = paste("Genus:", sig_genera$Genus[i]),
      value = abs(sig_genera$log2FoldChange[i]),
      type = "Treatment_to_Genus",
      stringsAsFactors = FALSE
    ))
  }
}

# Treatment to Function connections
if (nrow(sig_functions) > 0) {
  for (i in 1:nrow(sig_functions)) {
    sankey_data <- rbind(sankey_data, data.frame(
      source = treatment,
      target = paste("Function:", sig_functions$Feature_ID[i]),
      value = abs(sig_functions$log2FoldChange[i]),
      type = "Treatment_to_Function",
      stringsAsFactors = FALSE
    ))
  }
}

cat("Sankey connections:", nrow(sankey_data), "\n")

if (nrow(sankey_data) == 0) {
  cat("No connections for Sankey diagram:", treatment, "\n")
  return(NULL)
}

# Prepare data for networkD3
nodes <- data.frame(
```

```
        name = unique(c(sankey_data$source, sankey_data$target)),
        stringsAsFactors = FALSE
    )

    sankey_data$IDsource <- match(sankey_data$source, nodes$name) - 1
    sankey_data$IDtarget <- match(sankey_data$target, nodes$name) - 1

    tryCatch({
        # Create Sankey plot
        p <- sankeyNetwork(
            Links = sankey_data,
            Nodes = nodes,
            Source = "IDsource",
            Target = "IDtarget",
            Value = "value",
            NodeID = "name",
            sinksRight = FALSE,
            fontSize = 10,
            nodeWidth = 20,
            height = 500,
            width = 800
        )

        # Save plot
        htmlwidgets::saveWidget(p, filename, selfcontained = TRUE)
        cat("Sankey diagram saved:", filename, "\n")
        return(p)

    }, error = function(e) {
        cat("Error creating Sankey diagram:", e$message, "\n")
        return(NULL)
    })
}

# Enhanced Chord diagram for genus-function correlations (without title)
create_chord_diagram <- function(corr_results, treatment, filename) {

    cat("Creating Chord diagram for", treatment, "\n")

    # Combine correlation data
    all_corr <- data.frame()
    for (func_type in names(corr_results)) {
        if (!is.null(corr_results[[func_type]]) && nrow(corr_results[[func_type]]) > 0) {
            temp_corr <- corr_results[[func_type]] %>%
```

```
            filter(abs(Correlation) > 0.6 & !is.na(Genus) & !is.na(Function_ID)) %>%
            mutate(FuncType = func_type)
        all_corr <- rbind(all_corr, temp_corr)
    }
}

cat("Strong correlations for chord diagram:", nrow(all_corr), "\n")

if (nrow(all_corr) == 0) {
    cat("No strong correlations for chord diagram:", treatment, "\n")
    return(NULL)
}

# Prepare matrix for chord diagram
genera <- unique(all_corr$Genus)
functions <- unique(paste0(all_corr$Function_ID, " (", all_corr$FuncType, ")"))

cat("Genera:", length(genera), "Functions:", length(functions), "\n")

if (length(genera) == 0 || length(functions) == 0) {
    cat("Empty genera or functions list\n")
    return(NULL)
}

# Create adjacency matrix
adj_matrix <- matrix(0, nrow = length(genera), ncol = length(functions))
rownames(adj_matrix) <- genera
colnames(adj_matrix) <- functions

for (i in 1:nrow(all_corr)) {
    genus_name <- all_corr$Genus[i]
    func_name <- paste0(all_corr$Function_ID[i], " (", all_corr$FuncType[i], ")")
    if (genus_name %in% genera && func_name %in% functions) {
        adj_matrix[genus_name, func_name] <- abs(all_corr$Correlation[i])
    }
}

# Check if matrix has any values
if (all(adj_matrix == 0)) {
    cat("Adjacency matrix is all zeros\n")
    return(NULL)
}

tryCatch({
```

```
    # Create chord diagram
    png(filename, width = 12, height = 12, units = "in", res = 300)

    # Set colors
    genus_colors <- rainbow(length(genera), alpha = 0.8)
    func_colors <- viridis::viridis(length(functions), alpha = 0.8)
    colors <- c(genus_colors, func_colors)
    names(colors) <- c(genera, functions)

    chordDiagram(
      adj_matrix,
      grid.col = colors,
      transparency = 0.3,
      directional = 1,
      direction.type = c("arrows", "diffHeight"),
      diffHeight = -0.04,
      annotationTrack = "grid",
      annotationTrackHeight = c(0.03, 0.08),
      link.arr.type = "big.arrow",
      link.sort = TRUE,
      link.largest.ontop = TRUE
    )

    # Add labels
    circos.track(track.index = 1, panel.fun = function(x, y) {
      circos.text(CELL_META$xcenter, CELL_META$ylim[1], CELL_META$sector.index,
                  facing = "clockwise", niceFacing = TRUE, adj = c(0, 0.5), cex = 0.6)
    }, bg.border = NA)

    # Note: 不添加主标题 (No title added as requested)

    dev.off()

    circos.clear()
    cat("Chord diagram saved:", filename, "\n")

  }, error = function(e) {
    cat("Error creating Chord diagram:", e$message, "\n")
    dev.off()  # Make sure to close graphics device
    circos.clear()
    return(NULL)
  })
}
```

```
# ==================== 8. Generate All Visualizations ====================

# Generate All Visualizations with enhanced error handling
cat("\n=== Generating Visualizations ===\n")

# Create volcano plots
for (treatment in treatments) {
  cat("Creating volcano plot for", treatment, "\n")
  volcano_filename <- file.path(output_dir, paste0("Legionella_", gsub("[^A-Za-z0-9]", "_",
treatment), "_volcano.png"))

  tryCatch({
    create_legionella_volcano(all_func_results[[treatment]], treatment, volcano_filename)
  }, error = function(e) {
    cat("Failed to create volcano plot for", treatment, ":", e$message, "\n")
  })
}

# Create comprehensive heatmap
cat("Creating comprehensive heatmap\n")
heatmap_filename <- file.path(output_dir, "Legionella_functions_heatmap.png")

tryCatch({
  create_legionella_heatmap(all_func_results, heatmap_filename)
}, error = function(e) {
  cat("Failed to create heatmap:", e$message, "\n")
})

# Create Sankey diagrams
for (treatment in treatments) {
  if (!is.null(all_genus_results[[treatment]]) && nrow(all_genus_results[[treatment]]) > 0) {
    cat("Creating Sankey diagram for", treatment, "\n")
    sankey_filename <- file.path(output_dir, paste0("Sankey_", gsub("[^A-Za-z0-9]", "_",
treatment), ".html"))

    tryCatch({
      create_sankey_diagram(
        all_func_results[[treatment]],
        all_genus_results[[treatment]],
        all_correlations[[treatment]],
        treatment,
        sankey_filename
      )
    }, error = function(e) {
```

```
            cat("Failed to create Sankey diagram for", treatment, ":", e$message, "\n")
        })
    } else {
        cat("Skipping Sankey diagram for", treatment, "- no genus results\n")
    }
}

# Create chord diagrams
for (treatment in treatments) {
    cat("Creating chord diagram for", treatment, "\n")
    chord_filename <- file.path(output_dir, paste0("Chord_", gsub("[^A-Za-z0-9]", "_", treatment),
".png"))

    tryCatch({
        # Check if we have correlation data
        has_corr_data <- FALSE
        for (func_type in names(all_correlations[[treatment]])) {
            if (!is.null(all_correlations[[treatment]][[func_type]]) &&
                nrow(all_correlations[[treatment]][[func_type]]) > 0) {
                has_corr_data <- TRUE
                break
            }
        }

        if (has_corr_data) {
            create_chord_diagram(all_correlations[[treatment]], treatment, chord_filename)
        } else {
            cat("No correlation data for chord diagram:", treatment, "\n")
        }
    }, error = function(e) {
        cat("Failed to create chord diagram for", treatment, ":", e$message, "\n")
    })
}

# ==================== 9. Enhanced Summary Statistics and Reports ====================

# Calculate summary statistics with enhanced error handling
summary_stats <- data.frame()

for (treatment in treatments) {
    # Functional changes
    func_changes <- 0
    ko_changes <- 0
    ec_changes <- 0
```

```
  pathway_changes <- 0

  if (!is.null(all_func_results[[treatment]])) {
    for (func_type in names(all_func_results[[treatment]])) {
      if (!is.null(all_func_results[[treatment]][[func_type]])) {
        type_changes <- sum(all_func_results[[treatment]][[func_type]]$significant, na.rm =
TRUE)
        func_changes <- func_changes + type_changes

        if (func_type == "KO") ko_changes <- type_changes
        if (func_type == "EC") ec_changes <- type_changes
        if (func_type == "Pathway") pathway_changes <- type_changes
      }
    }
  }

  # Genus changes
  genus_changes <- 0
  if (!is.null(all_genus_results[[treatment]])) {
    genus_changes <- sum(all_genus_results[[treatment]]$significant, na.rm = TRUE)
  }

  # Strong correlations
  strong_corr <- 0
  ko_corr <- 0
  ec_corr <- 0
  pathway_corr <- 0

  if (!is.null(all_correlations[[treatment]])) {
    for (func_type in names(all_correlations[[treatment]])) {
      if (!is.null(all_correlations[[treatment]][[func_type]])) {
        type_corr <- nrow(all_correlations[[treatment]][[func_type]])
        strong_corr <- strong_corr + type_corr

        if (func_type == "KO") ko_corr <- type_corr
        if (func_type == "EC") ec_corr <- type_corr
        if (func_type == "Pathway") pathway_corr <- type_corr
      }
    }
  }

  # Sample size
  sample_size <- sum(sample_data(physeq_filtered)$Treatment == treatment, na.rm = TRUE)
```

```
  summary_stats <- rbind(summary_stats, data.frame(
    Treatment = treatment,
    Sample_Size = sample_size,
    Total_Functional_Changes = func_changes,
    KO_Changes = ko_changes,
    EC_Changes = ec_changes,
    Pathway_Changes = pathway_changes,
    Genus_Changes = genus_changes,
    Total_Strong_Correlations = strong_corr,
    KO_Correlations = ko_corr,
    EC_Correlations = ec_corr,
    Pathway_Correlations = pathway_corr
  ))
}

# Print summary
cat("\n=== Analysis Summary ===\n")
print(summary_stats)

# Save summary statistics
write.csv(summary_stats, file.path(output_dir, "Legionella_analysis_summary_stats.csv"),
row.names = FALSE)

# Save detailed results with error handling
for (treatment in treatments) {
  # Save functional results
  if (!is.null(all_func_results[[treatment]])) {
    for (func_type in names(all_func_results[[treatment]])) {
      if (!is.null(all_func_results[[treatment]][[func_type]]) &&
          nrow(all_func_results[[treatment]][[func_type]]) > 0) {
        tryCatch({
          filename <- paste0("Legionella_", func_type, "_", gsub("[^A-Za-z0-9]", "_",
treatment), "_results.csv")
          write.csv(all_func_results[[treatment]][[func_type]],
                    file.path(output_dir, filename), row.names = FALSE)
        }, error = function(e) {
          cat("Error saving functional results for", treatment, func_type, ":", e$message, "\n")
        })
      }
    }
  }

  # Save genus results
  if (!is.null(all_genus_results[[treatment]]) && nrow(all_genus_results[[treatment]]) > 0) {
```

```
    tryCatch({
      filename <- paste0("Genus_", gsub("[^A-Za-z0-9]", "_", treatment), "_results.csv")
      write.csv(all_genus_results[[treatment]],
                file.path(output_dir, filename), row.names = FALSE)
    }, error = function(e) {
      cat("Error saving genus results for", treatment, ":", e$message, "\n")
    })
  }

  # Save correlation results
  if (!is.null(all_correlations[[treatment]])) {
    for (func_type in names(all_correlations[[treatment]])) {
      if (!is.null(all_correlations[[treatment]][[func_type]]) &&
          nrow(all_correlations[[treatment]][[func_type]]) > 0) {
        tryCatch({
          filename <- paste0("Correlations_", func_type, "_", gsub("[^A-Za-z0-9]", "_",
treatment), "_results.csv")
          write.csv(all_correlations[[treatment]][[func_type]],
                    file.path(output_dir, filename), row.names = FALSE)
        }, error = function(e) {
          cat("Error saving correlation results for", treatment, func_type, ":", e$message, "\n")
        })
      }
    }
  }
}

# Generate enhanced final report
report_lines <- c(
  "Legionella-Related Functional and Taxonomic Analysis Report",
  paste(rep("=", 60), collapse = ""),
  "",
  paste("Analysis Date:", Sys.Date()),
  "Analysis Focus: Treatment effects on Legionella-related functions and their taxonomic
associations",
  "",
  "Sample Sizes by Treatment:",
  paste("  - Untreated:", sum(sample_data(physeq_filtered)$Treatment == "Untreated", na.rm
= TRUE)),
  paste("  - Water-Vapor:", sum(sample_data(physeq_filtered)$Treatment == "Water-Vapor",
na.rm = TRUE)),
  paste("  - Thermal Disinfection:", sum(sample_data(physeq_filtered)$Treatment == "Thermal
Disinfection", na.rm = TRUE)),
  paste("  - Chemical Treatment:", sum(sample_data(physeq_filtered)$Treatment == "Chemical
```

```
Treatment", na.rm = TRUE)),
  paste(" - Chemical Treatment + Water-Vapor:", sum(sample_data(physeq_filtered)$Treatment == "Chemical Treatment + Water-Vapor", na.rm = TRUE)),
  "",
  "Legionella-Related Function Categories Analyzed:",
  "   1. Biofilm Formation",
  "   2. Virulence Factors",
  "   3. Resistance Mechanisms",
  "   4. Nutrient Metabolism",
  "   5. Environmental Adaptation",
  "   6. Microbial Interactions",
  "",
  "Features Found:",
  paste("   - KO functions:", length(legionella_ko$features)),
  paste("   - EC functions:", length(legionella_ec$features)),
  paste("   - Pathway functions:", length(legionella_pathway$features)),
  paste("   - Genera analyzed:", ntaxa(physeq_genus_filtered)),
  "",
  "Summary Statistics:"
)

for (i in 1:nrow(summary_stats)) {
  report_lines <- c(report_lines,
    paste("", summary_stats$Treatment[i], "(n =", summary_stats$Sample_Size[i], "):"),
    paste("      Total functional changes:", summary_stats$Total_Functional_Changes[i]),
    paste("        - KO changes:", summary_stats$KO_Changes[i]),
    paste("        - EC changes:", summary_stats$EC_Changes[i]),
    paste("        - Pathway changes:", summary_stats$Pathway_Changes[i]),
    paste("      Genus changes:", summary_stats$Genus_Changes[i]),
    paste("      Strong correlations:", summary_stats$Total_Strong_Correlations[i])
  )
}

report_lines <- c(report_lines,
  "",
  "Key Observations:",
  "   - Chemical Treatment + Water-Vapor showed the most functional changes (21 total)",
  "   - Small sample sizes for Chemical Treatment (n=3) and Combined treatment (n=2) limit reliability",
  "   - Correlation analysis was challenging due to sample size limitations",
  "   - Water-Vapor and Thermal Disinfection had adequate sample sizes for robust analysis",
  "",
  "Output Files Generated:",
```

```
  "    1. Volcano plots: Legionella_*_volcano.png",
  "    2. Comprehensive heatmap: Legionella_functions_heatmap.png",
  "    3. Sankey diagrams: Sankey_*.html (where data available)",
  "    4. Chord diagrams: Chord_*.png (where correlation data available, no title displayed)",
  "    5. Detailed results: CSV files for each analysis",
  "    6. Summary statistics: Legionella_analysis_summary_stats.csv",
  "",
  "Analysis Limitations:",
  "    - Small sample sizes for some treatments reduce statistical power",
  "    - ANCOM-BC warnings about insufficient taxa for bias estimation",
  "    - Correlation analysis limited by sample size requirements",
  "",
  "Recommendations:",
  "    - Focus interpretation on treatments with adequate sample sizes (Water-Vapor, Thermal Disinfection)",
  "    - Consider combining treatments or increasing sample sizes for future analyses",
  "    - Use caution interpreting results from small sample size groups"
)

writeLines(report_lines, file.path(output_dir, "Legionella_analysis_report.txt"))

cat("\n=== Legionella Functional Analysis Completed ===\n")
cat("All results saved to:", output_dir, "\n")
cat("Check the report file for detailed analysis summary\n")
cat("\nNote: Some analyses were limited by small sample sizes\n")
cat("Most reliable results are from Water-Vapor and Thermal Disinfection treatments\n")

# ==================== Volcano Plot Diagnostics and Repair ====================

# 1. First, diagnose data issues
diagnose_volcano_data <- function(all_func_results) {

  for (treatment in names(all_func_results)) {
    cat("\n=== Diagnosing data for", treatment, "===\n")

    # Check data for each function type
    for (func_type in names(all_func_results[[treatment]])) {
      if (!is.null(all_func_results[[treatment]][[func_type]])) {
        data <- all_func_results[[treatment]][[func_type]]

        cat(func_type, "data:\n")
        cat("  Total rows:", nrow(data), "\n")
        cat("  Significant results:", sum(data$significant, na.rm = TRUE), "\n")
```

```
            # Check log2FoldChange range
            if ("log2FoldChange" %in% colnames(data)) {
                lfc_range <- range(data$log2FoldChange, na.rm = TRUE)
                cat("    log2FC range:", round(lfc_range[1], 3), "to", round(lfc_range[2], 3), "\n")
                cat("    |log2FC| > 1:", sum(abs(data$log2FoldChange) > 1, na.rm = TRUE), "\n")
                cat("    |log2FC| > 0.5:", sum(abs(data$log2FoldChange) > 0.5, na.rm = TRUE), "\n")
            }

            # Check p-value range
            if ("padj" %in% colnames(data)) {
                padj_range <- range(data$padj, na.rm = TRUE)
                cat("    padj range:", padj_range[1], "to", padj_range[2], "\n")
                cat("    padj = 0:", sum(data$padj == 0, na.rm = TRUE), "\n")
                cat("    padj < 0.05:", sum(data$padj < 0.05, na.rm = TRUE), "\n")
            }
            cat("\n")
        }
    }
  }
}

# Run diagnostics
cat("Starting volcano plot data diagnosis...\n")
diagnose_volcano_data(all_func_results)

# 2. Fixed volcano plot function (with more relaxed filtering)
create_legionella_volcano_fixed <- function(results_list, treatment, filename) {

  cat("\n=== Creating fixed volcano plot:", treatment, "===\n")

  # Combine all functional results
  combined_results <- data.frame()

  for (func_type in names(results_list)) {
    if (!is.null(results_list[[func_type]]) && nrow(results_list[[func_type]]) > 0) {
      temp_data <- results_list[[func_type]]
      # Ensure all required columns exist
      if (all(c("log2FoldChange", "padj") %in% colnames(temp_data))) {
        combined_results <- rbind(combined_results, temp_data)
      }
    }
  }

  if (nrow(combined_results) == 0) {
```

```
    cat("No data for volcano plot:", treatment, "\n")
    return(NULL)
}

cat("Combined data rows:", nrow(combined_results), "\n")

# More lenient data cleaning (only remove actual problem values)
original_nrow <- nrow(combined_results)

combined_results <- combined_results %>%
    filter(!is.na(log2FoldChange) & !is.na(padj) &
               is.finite(log2FoldChange) & is.finite(padj))

# Handle p-values exactly equal to 0 (set to tiny value instead of deleting)
zero_padj_count <- sum(combined_results$padj == 0, na.rm = TRUE)
if (zero_padj_count > 0) {
    cat("Found", zero_padj_count, "p-values equal to 0, setting to 1e-300\n")
    combined_results$padj[combined_results$padj == 0] <- 1e-300
}

cat("Cleaned data rows:", nrow(combined_results), "\n")
cat("Rows removed:", original_nrow - nrow(combined_results), "\n")

if (nrow(combined_results) == 0) {
    cat("No valid data after cleaning\n")
    return(NULL)
}

# Use more relaxed significance classification (lower log2FC threshold to 0.5)
combined_results <- combined_results %>%
    mutate(
        Significance = case_when(
            padj < 0.05 & log2FoldChange > 0.5 ~ "Significantly Up",
            padj < 0.05 & log2FoldChange < -0.5 ~ "Significantly Down",
            padj < 0.05 ~ "Significant (small FC)",
            TRUE ~ "Not Significant"
        )
    )

# Count significance categories
sig_table <- table(combined_results$Significance)
cat("Significance category counts:\n")
print(sig_table)
```

```
# If no Category column exists, create a default one
if (!"Category" %in% colnames(combined_results)) {
  combined_results$Category <- "Unknown"
}

# If no FuncType column exists, try to infer from other info
if (!"FuncType" %in% colnames(combined_results)) {
  combined_results$FuncType <- "Mixed"
}

# Create the plot
tryCatch({
  p <- ggplot(combined_results, aes(x = log2FoldChange, y = -log10(padj))) +
    geom_point(aes(color = Significance, size = abs(log2FoldChange)), alpha = 0.7) +
    scale_color_manual(values = c(
      "Significantly Up" = "red",
      "Significantly Down" = "blue",
      "Significant (small FC)" = "orange",
      "Not Significant" = "grey"
    )) +
    scale_size_continuous(range = c(1, 4), name = "|log2FC|") +
    geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "red", alpha = 0.7) +
    geom_vline(xintercept = c(-0.5, 0.5), linetype = "dashed", color = "red", alpha = 0.7) +
    labs(
      title = paste("Legionella-Related Functions:", treatment, "vs Untreated"),
      subtitle = paste("Total points:", nrow(combined_results),
                       "| Significant:", sum(combined_results$padj < 0.05, na.rm = TRUE)),
      x = "log2 Fold Change",
      y = "-log10(adjusted p-value)",
      color = "Significance"
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
      plot.subtitle = element_text(hjust = 0.5, size = 10),
      legend.position = "bottom"
    )

  # If multiple function types exist, add faceting
  if (length(unique(combined_results$FuncType)) > 1) {
    p <- p + facet_wrap(~FuncType, scales = "free")
  }

  ggsave(filename, plot = p, width = 12, height = 8, dpi = 300)
```

```
        cat("Fixed volcano plot saved:", filename, "\n")

        # Add detailed point statistics
        cat("\nDetailed point statistics:\n")
        cat("Total points:", nrow(combined_results), "\n")
        cat("Significant points (p < 0.05):", sum(combined_results$padj < 0.05), "\n")
        cat("Up-regulated significant:", sum(combined_results$padj < 0.05 &
combined_results$log2FoldChange > 0), "\n")
        cat("Down-regulated significant:", sum(combined_results$padj < 0.05 &
combined_results$log2FoldChange < 0), "\n")

        return(p)

    }, error = function(e) {
        cat("Error creating volcano plot:", e$message, "\n")
        return(NULL)
    })
}

# 3. Recreate volcano plots for all treatment groups
cat("\n=== Recreating all volcano plots ===\n")

treatments <- c("Water-Vapor", "Thermal Disinfection", "Chemical Treatment", "Chemical
Treatment + Water-Vapor")

for (treatment in treatments) {
    cat("\nCreating fixed volcano plot:", treatment, "\n")
    volcano_filename <- file.path(output_dir, paste0("Legionella_", gsub("[^A-Za-z0-9]", "_",
treatment), "_volcano_FIXED.png"))

    tryCatch({
        create_legionella_volcano_fixed(all_func_results[[treatment]], treatment,
volcano_filename)
    }, error = function(e) {
        cat("Fixed volcano plot creation failed:", treatment, ":", e$message, "\n")
    })
}

# 4. Simplified volcano plot (if above still fails)
create_simple_volcano <- function(results_list, treatment, filename) {

    cat("\n=== Creating simplified volcano plot:", treatment, "===\n")

    # Combine data without any filtering
```

```
    all_data <- data.frame()

    for (func_type in names(results_list)) {
        if (!is.null(results_list[[func_type]])) {
            temp_data <- results_list[[func_type]]
            temp_data$FuncType <- func_type
            all_data <- rbind(all_data, temp_data)
        }
    }

    if (nrow(all_data) == 0) {
        cat("No data\n")
        return(NULL)
    }

    cat("Simplified data rows:", nrow(all_data), "\n")

    # Perform minimal processing
    all_data$neg_log10_padj <- ifelse(all_data$padj == 0, 300, -log10(all_data$padj))
    all_data$is_significant <- all_data$padj < 0.05

    # Create the simplest possible volcano plot
    p <- ggplot(all_data, aes(x = log2FoldChange, y = neg_log10_padj)) +
        geom_point(aes(color = is_significant), alpha = 0.6, size = 2) +
        scale_color_manual(values = c("FALSE" = "grey", "TRUE" = "red"),
                                    name = "Significant\n(p < 0.05)") +
        geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "red") +
        labs(
            title = paste("Legionella Functions:", treatment),
            subtitle = paste("Points:", nrow(all_data), "| Significant:", sum(all_data$is_significant)),
            x = "log2 Fold Change",
            y = "-log10(adjusted p-value)"
        ) +
        theme_bw()

    ggsave(filename, plot = p, width = 10, height = 6, dpi = 300)
    cat("Simplified volcano plot saved:", filename, "\n")

    return(p)
}

# Create simplified versions as fallback
cat("\n=== Creating simplified volcano plots as fallback ===\n")
```

```
for (treatment in treatments) {
  simple_filename <- file.path(output_dir, paste0("Legionella_", gsub("[^A-Za-z0-9]", "_", treatment), "_volcano_SIMPLE.png"))
  create_simple_volcano(all_func_results[[treatment]], treatment, simple_filename)
}
```

The following script is for metal ion analysis and time series analysis in the study

```
# ==================== 1. Load Required R Packages ====================
library(phyloseq)
library(readxl)
library(readr)
library(dplyr)
library(tibble)

# ==================== 2. Set File Paths ====================
metadata_path <- "C:/Users/ASUS/Desktop/imeta/metadata.xlsx"
asv_table_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_table.csv"
taxonomy_path <- "C:/Users/ASUS/Desktop/imeta/result/taxonomy_table.xlsx"

# ==================== 3. Read Data Files ====================

# 3.1 Read sample metadata
cat("Reading sample metadata...\n")
metadata <- read_excel(metadata_path, sheet = 1)
print(paste("Metadata contains", nrow(metadata), "samples"))
print(paste("Metadata columns:", paste(colnames(metadata), collapse = ", ")))

# 3.2 Read ASV abundance table
cat("\nReading ASV abundance table...\n")
asv_table <- read_csv(asv_table_path)
print(paste("ASV table contains", nrow(asv_table), "ASVs and", ncol(asv_table)-1, "samples"))

# 3.3 Read taxonomic information
cat("\nReading taxonomic information...\n")
taxonomy <- read_excel(taxonomy_path, sheet = 1)
print(paste("Taxonomy table contains", nrow(taxonomy), "ASVs"))
print(paste("Taxonomic ranks:", paste(colnames(taxonomy)[-1], collapse = ", ")))

# ==================== 4. Data Preprocessing ====================

# 4.1 Process sample metadata
# Set sample_id as row names
sample_data_df <- metadata %>%
  column_to_rownames("sample_id")

# 4.2 Process ASV abundance table
```

```
# Set ASV_id as row names, convert to matrix
otu_table_df <- asv_table %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check and ensure consistent sample ordering
cat("\nChecking sample ID consistency...\n")
metadata_samples <- rownames(sample_data_df)
asv_samples <- colnames(otu_table_df)

if (length(setdiff(metadata_samples, asv_samples)) == 0 &&
      length(setdiff(asv_samples, metadata_samples)) == 0) {
  cat("✓ Sample IDs match completely!\n")
} else {
  cat("⚠ Sample IDs do not match, need to check!\n")
}

# 4.3 Process taxonomic information
# Set ASV_id as row names
tax_table_df <- taxonomy %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check ASV ID consistency
asv_ids_otu <- rownames(otu_table_df)
asv_ids_tax <- rownames(tax_table_df)

if (length(setdiff(asv_ids_otu, asv_ids_tax)) == 0 &&
      length(setdiff(asv_ids_tax, asv_ids_otu)) == 0) {
  cat("✓ ASV IDs match completely!\n")
} else {
  cat("⚠ ASV IDs do not match, need to check!\n")
}

# ==================== 5. Create Phyloseq Object Components ====================

# 5.1 Create OTU table object
cat("\nCreating phyloseq components...\n")
OTU <- otu_table(otu_table_df, taxa_are_rows = TRUE)

# 5.2 Create sample data object
SAMP <- sample_data(sample_data_df)

# 5.3 Create taxonomy table object
```

```
TAX <- tax_table(tax_table_df)

# ==================== 6. Construct Phyloseq Object ====================
cat("Constructing phyloseq object...\n")
physeq <- phyloseq(OTU, SAMP, TAX)

# ==================== 7. Validation and Summary ====================
cat("\n=== Phyloseq object construction complete! ===\n")
print(physeq)

# Output basic statistical information
cat("\n=== Basic Statistical Information ===\n")
cat("Number of samples:", nsamples(physeq), "\n")
cat("Number of ASVs:", ntaxa(physeq), "\n")
cat("Taxonomic ranks:", paste(rank_names(physeq), collapse = ", "), "\n")
cat("Sample variables:", paste(sample_variables(physeq), collapse = ", "), "\n")

# Check data integrity
cat("\n=== Data Quality Check ===\n")
# Check for empty samples
empty_samples <- sample_sums(physeq) == 0
if (any(empty_samples)) {
  cat("⚠ Found", sum(empty_samples), "empty samples\n")
} else {
  cat("✓ All samples contain sequences\n")
}

# Check for empty ASVs
empty_taxa <- taxa_sums(physeq) == 0
if (any(empty_taxa)) {
  cat("⚠ Found", sum(empty_taxa), "empty ASVs\n")
} else {
  cat("✓ All ASVs contain sequences\n")
}

# Display sample sequencing depth statistics
cat("\n=== Sequencing Depth Statistics ===\n")
seq_depth <- sample_sums(physeq)
cat("Minimum sequencing depth:", min(seq_depth), "\n")
cat("Maximum sequencing depth:", max(seq_depth), "\n")
cat("Average sequencing depth:", round(mean(seq_depth), 0), "\n")
cat("Median sequencing depth:", round(median(seq_depth), 0), "\n")

# ==================== 8. Optional: Save Phyloseq Object ====================
```

```
# Uncomment the following line to save the phyloseq object
# saveRDS(physeq, "phyloseq_object.rds")
# cat("\nPhyloseq object saved as phyloseq_object.rds\n")

# ==================== 9. Optional: Basic Data Cleaning ====================
# Basic data filtering if needed
cat("\n=== Optional Data Cleaning Steps ===\n")
cat("Original data: ", ntaxa(physeq), "ASVs,", nsamples(physeq), "samples\n")

# Remove empty samples and empty ASVs if any
physeq_cleaned <- prune_samples(sample_sums(physeq) > 0, physeq)
physeq_cleaned <- prune_taxa(taxa_sums(physeq_cleaned) > 0, physeq_cleaned)

cat("Cleaned data: ", ntaxa(physeq_cleaned), "ASVs,", nsamples(physeq_cleaned), "samples\n")

# Assign cleaned object to main variable
physeq <- physeq_cleaned

cat("\n🎉 Phyloseq object construction complete! You can start subsequent analyses.\n")
cat("Object name: physeq\n")
cat("Use print(physeq) to view object information\n")

# ==================== 10. Add Phylogenetic Tree to Phyloseq Object ====================
library(ape)

# Set tree file paths
tree_path <- "C:/Users/ASUS/Desktop/imeta/result/sequences.aligned.fasta.treefile"
hash_mapping_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_hash_mapping.csv"

# 10.1 Read and process hash mapping
cat("\nReading ASV hash mapping...\n")
hash_mapping <- read_csv(hash_mapping_path, col_names = c("ASV_id", "Hash"))
hash_mapping$Hash <- trimws(hash_mapping$Hash)    # Clean whitespaces
hash_vec <- setNames(hash_mapping$ASV_id, hash_mapping$Hash)

# 10.2 Read phylogenetic tree
cat("Reading phylogenetic tree...\n")
tree <- read.tree(tree_path)

# 10.3 Check tree-hash correspondence
tree_hashes <- tree$tip.label
mapped_asvs <- hash_vec[tree_hashes]
unmapped_count <- sum(is.na(mapped_asvs))
```

```
if (unmapped_count > 0) {
  cat("⚠ Warning:", unmapped_count, "tree tips not found in hash mapping\n")
  # Remove unmapped tips
  tree <- drop.tip(tree, tree_hashes[is.na(mapped_asvs)])
  mapped_asvs <- na.omit(mapped_asvs)
}

# 10.4 Update tip labels with ASV IDs
tree$tip.label <- as.character(mapped_asvs[tree$tip.label])

# 10.5 Align tree and phyloseq object
# Get overlapping ASVs
common_asvs <- intersect(tree$tip.label, taxa_names(physeq))
physeq_sub <- prune_taxa(common_asvs, physeq)
tree <- keep.tip(tree, common_asvs)

cat("→ Retained", length(common_asvs), "ASVs common to tree and phyloseq\n")

# 10.6 Add tree to phyloseq object
phy_tree(physeq_sub) <- tree

# 10.7 Validate integration
cat("\n=== Tree Integration Validation ===\n")
cat("Tree contains", Ntip(tree), "tips matching ASVs\n")

if (all(taxa_names(physeq_sub) %in% tree$tip.label)) {
  cat("✓ All phyloseq taxa present in tree\n")
} else {
  cat("⚠ Missing taxa: Not all ASVs are in the tree\n")
}

# 10.8 Update phyloseq object
physeq <- phyloseq(otu_table(physeq_sub),
                   sample_data(physeq_sub),
                   tax_table(physeq_sub),
                   phy_tree(physeq_sub))

cat("\n=== Updated Phyloseq Object Summary ===\n")
print(physeq)

# 10.9 Optional: Save updated object
# saveRDS(physeq, "phyloseq_with_tree.rds")
# cat("Saved phyloseq object with tree as 'phyloseq_with_tree.rds'\n")
```

```
cat("\n ✅ Phylogenetic tree successfully integrated!\n")

# ==================== 1. Load Required R Packages ====================
library(phyloseq)          # For microbiome data analysis
library(ggplot2)           # For data visualization
library(dplyr)             # For data manipulation
library(tidyr)             # For data tidying
library(vegan)             # For ecological analysis
library(ggpubr)            # For arranging plots
library(RColorBrewer)      # For color palettes
library(reshape2)          # For data reshaping
library(gridExtra)         # For arranging multiple plots
library(broom)             # For tidying statistical outputs
library(picante)           # For Faith's phylogenetic diversity calculation
library(pheatmap)          # For heatmap visualization

# ==================== 2. Set Output Path ====================
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/time"
if (!dir.exists(output_dir)) {
  dir.create(output_dir, recursive = TRUE)
  cat("Created output directory:", output_dir, "\n")
}

# Assuming you have already run the previous script and have a 'physeq' object in your environment.

# ==================== 3. Data Preprocessing and Time Series Sample Extraction ====================

# 3.1 Check metadata structure
cat("=== Metadata Check ===\n")
print(head(sample_data(physeq)))

# 3.2 Extract time series samples (Weeks 32-36)
timeseries.weeks <- c(32, 33, 34, 35, 36)
physeq.ts <- subset_samples(physeq, Biofilm_Age %in% timeseries.weeks)

cat("Time series data contains", nsamples(physeq.ts), "samples\n")

# 3.3 Create time point and treatment group identifiers
sample_data(physeq.ts)$Time_Point <- factor(sample_data(physeq.ts)$Biofilm_Age)
sample_data(physeq.ts)$Treatment_Group <- factor(sample_data(physeq.ts)$Treatment)

# ==================== 4. Alpha Diversity Time Series Analysis ====================
```

```
# 4.1 Calculate diversity indices
cat("Calculating Alpha diversity indices...\n")
alpha.div <- estimate_richness(physeq.ts, measures = c("Observed", "Shannon", "Simpson"))

# Calculate Faith's Phylogenetic Diversity (if tree is available)
if (!is.null(phy_tree(physeq.ts))) {
  cat("Calculating Faith's phylogenetic diversity...\n")
  otu.table.df <- as.data.frame(otu_table(physeq.ts))
  if (taxa_are_rows(physeq.ts)) {
     otu.table.df <- t(otu.table.df)
  }
  faith.pd <- pd(otu.table.df, phy_tree(physeq.ts), include.root = FALSE)
  alpha.div$Faith_PD <- faith.pd$PD
  cat("✓ Faith PD calculation completed\n")
} else {
  cat("⚠ No phylogenetic tree found, skipping Faith PD calculation\n")
  alpha.div$Faith_PD <- NA
}

alpha.div$Sample <- rownames(alpha.div)

# 4.2 Merge with metadata
metadata.ts <- data.frame(sample_data(physeq.ts))
metadata.ts$Sample <- rownames(metadata.ts)
alpha.data <- merge(alpha.div, metadata.ts, by = "Sample")

# 4.3 Alpha diversity time series visualization
alpha.plots <- list()

# Define indices, labels, and y-axis limits
diversity.indices <- c("Observed", "Shannon", "Simpson", "Faith_PD")
diversity.labels <- c("Observed ASVs", "Shannon Diversity", "Simpson Diversity", "Faith Phylogenetic Diversity")
y.limits <- list(
  "Observed" = c(0, 40),
  "Shannon" = c(1, 4.0),
  "Simpson" = c(0.5, 1.0),
  "Faith_PD" = c(10, 125)
)

for (i in 1:length(diversity.indices)) {
  index <- diversity.indices[i]
  label <- diversity.labels[i]
```

```
  y.lim <- y.limits[[index]]

  if (all(is.na(alpha.data[[index]]))) {
    cat("Skipping", index, "(missing data)\n")
    next
  }

  # MODIFICATION: Create a separate dataset for treated groups to draw lines
  treated_data <- alpha.data %>%
    filter(Treatment_Group != "Untreated")

  # Create base plot
  # Points are drawn for ALL data, but lines/smoothers are drawn ONLY for treated_data
  p <- ggplot(alpha.data, aes(x = Biofilm_Age, y = .data[[index]], color = Treatment_Group)) +
    # Plot points for all samples
    geom_point(size = 3, alpha = 0.7) +
    # Plot lines and smoothers ONLY for the treated groups
    geom_line(data = treated_data, aes(group = Treatment_Group), linewidth = 1) +
    stat_smooth(data = treated_data, method = "loess", se = TRUE, alpha = 0.2, aes(group =
Treatment_Group)) +
    scale_color_brewer(type = "qual", palette = "Set1") +
    scale_y_continuous(limits = y.lim) +
    labs(title = paste(label, "Time Series"),
         x = "Biofilm Age (weeks)",
         y = label,
         color = "Treatment Group") +
    theme_bw() +
    theme(legend.position = "bottom")

  alpha.plots[[index]] <- p
}

# Display and save plots
for (index in names(alpha.plots)) {
  print(alpha.plots[[index]])
  ggsave(file.path(output_dir, paste0("alpha_", tolower(index), "_timeseries.png")),
         alpha.plots[[index]], width = 12, height = 8, dpi = 300)
}

# ==================== 5. Beta Diversity Time Series Analysis ====================

# 5.1 Relative abundance transformation
physeq.ts.rel <- transform_sample_counts(physeq.ts, function(x) x/sum(x))
```

```
# 5.2 Calculate distance matrix
cat("Calculating distance matrix...\n")
if (!is.null(phy_tree(physeq.ts))) {
  dist.metric <- "unifrac"
  cat("✓ Using Unweighted UniFrac distance\n")
  dist_matrix <- distance(physeq.ts.rel, method = dist.metric, weighted = FALSE)
} else {
  dist.metric <- "bray"
  cat("⚠ No phylogenetic tree found, using Bray-Curtis distance instead\n")
  dist_matrix <- distance(physeq.ts.rel, method = dist.metric)
}

# 5.3 Principal Coordinates Analysis (PCoA)
pcoa.result <- ordinate(physeq.ts.rel, method = "PCoA", distance = dist_matrix)

# 5.4 PCoA time series visualization
pcoa.plot <- plot_ordination(physeq.ts.rel, pcoa.result, color = "Treatment_Group", shape =
"Time_Point") +
  geom_point(size = 4, alpha = 0.8) +
  scale_color_brewer(type = "qual", palette = "Set1") +
  labs(title = paste("PCoA -", tools::toTitleCase(dist.metric), "Distance"),
          color = "Treatment Group",
          shape = "Time Point") +
  theme_bw()

# Add time trajectory lines
pcoa.data <- pcoa.plot$data
for(treatment in unique(pcoa.data$Treatment_Group)) {
  if(treatment != "Untreated") {
     treatment.data <- pcoa.data[pcoa.data$Treatment_Group == treatment, ]
     treatment.data <- treatment.data[order(treatment.data$Biofilm_Age), ]
     pcoa.plot <- pcoa.plot +
        geom_path(data = treatment.data, aes(x = Axis.1, y = Axis.2), color = "gray50", alpha = 0.6,
linewidth = 0.8)
  }
}

print(pcoa.plot)
ggsave(file.path(output_dir, "pcoa_unifrac_timeseries.png"), pcoa.plot, width = 10, height = 7,
dpi = 300)

# ==================== 6. Genus-level Taxon Temporal Dynamics Analysis
====================
```

```
# 6.1 Aggregate to genus level
cat("Performing genus-level analysis...\n")
physeq.genus <- tax_glom(physeq.ts.rel, "Genus", NArm = FALSE)
genus.data <- psmelt(physeq.genus)
genus.data$Genus[is.na(genus.data$Genus) | genus.data$Genus == ""] <- "Unclassified"

# 6.2 Select Top 15 major genera
major.genera <- genus.data %>%
  group_by(Genus) %>%
  summarise(mean.abundance = mean(Abundance), .groups = "drop") %>%
  arrange(desc(mean.abundance)) %>%
  pull(Genus) %>%
  .[1:15]
cat("Selected major genera:", paste(major.genera, collapse = ", "), "\n")

# 6.3 Process genus time series data
genus.ts.data <- genus.data %>%
  filter(Genus %in% major.genera) %>%
  group_by(Biofilm_Age, Treatment_Group, Genus) %>%
  summarise(mean.abundance = mean(Abundance),
                    se.abundance = sd(Abundance)/sqrt(n()), .groups = "drop")

# 6.4 Genus abundance time series visualization
genus.ts.plot <- ggplot(genus.ts.data, aes(x = Biofilm_Age, y = mean.abundance, color =
Treatment_Group)) +
    geom_line(aes(group = Treatment_Group), linewidth = 1) +
    geom_point(size = 2) +
    geom_errorbar(aes(ymin = pmax(0, mean.abundance - se.abundance), ymax =
mean.abundance + se.abundance), width = 0.2, alpha = 0.7) +
    facet_wrap(~Genus, scales = "free_y", ncol = 5) +
    scale_color_brewer(type = "qual", palette = "Set1") +
    labs(title = "Major Genera Relative Abundance Time Series",
             x = "Biofilm Age (weeks)", y = "Relative Abundance", color = "Treatment Group") +
    theme_bw() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "bottom", strip.text =
element_text(size = 8))

print(genus.ts.plot)
ggsave(file.path(output_dir, "genus_timeseries.png"), genus.ts.plot, width = 16, height = 10, dpi =
300)

# 6.5 Genus temporal dynamics heatmap
genus.heatmap.data <- genus.ts.data %>%
  select(Biofilm_Age, Treatment_Group, Genus, mean.abundance) %>%
```

```
  unite("Sample_Group", Biofilm_Age, Treatment_Group, sep = "_") %>%
  spread(Sample_Group, mean.abundance, fill = 0)
genus.matrix <- as.matrix(genus.heatmap.data[,-1])
rownames(genus.matrix) <- genus.heatmap.data$Genus

# Create heatmap with pheatmap
genus.heatmap <- pheatmap(genus.matrix,
                          scale = "row",
                          clustering_distance_rows = "euclidean",
                          clustering_distance_cols = "euclidean",
                          color = colorRampPalette(c("blue", "white", "red"))(100),
                          main = "Genus-level Relative Abundance Heatmap (Row-scaled)",
                          fontsize_row = 8, fontsize_col = 8, angle_col = 45)

# Save heatmap
png(file.path(output_dir, "genus_heatmap.png"), width = 12, height = 8, units = "in", res = 300)
print(genus.heatmap)
dev.off()

# ==================== 7. Recovery Dynamics Analysis ====================

# 7.1 Analyze recovery at week 36
recovery.data <- alpha.data[alpha.data$Biofilm_Age == 36, ]

# 7.2 Create recovery period comparison plots
recovery.plots <- list()
for (index in diversity.indices) {
  if (all(is.na(recovery.data[[index]]))) next

  p <- ggplot(recovery.data, aes(x = Treatment_Group, y = .data[[index]], fill = Treatment_Group))
+
    geom_boxplot(alpha = 0.7, outlier.shape = NA) +
    geom_jitter(width = 0.2, size = 3, alpha = 0.7) +
    scale_fill_brewer(type = "qual", palette = "Set1") +
    labs(
      title = paste("Week 36 Recovery -", diversity.labels[which(diversity.indices == index)]),
      x = "Treatment Group", y = diversity.labels[which(diversity.indices == index)]
    ) +
    theme_bw() +
    theme(legend.position = "none")

  recovery.plots[[index]] <- p
}
```

```
# Display and save recovery plots
for (index in names(recovery.plots)) {
  print(recovery.plots[[index]])
  ggsave(file.path(output_dir, paste0("recovery_", tolower(index), ".png")),
         recovery.plots[[index]], width = 8, height = 6, dpi = 300)
}

# 7.3 Calculate recovery rates
baseline.data <- alpha.data[alpha.data$Biofilm_Age == 32 & alpha.data$Treatment_Group ==
"Untreated", ]
recovery.stats.list <- list()
for (index in diversity.indices) {
  if (all(is.na(alpha.data[[index]]))) next
  baseline.value <- mean(baseline.data[[index]], na.rm = TRUE)
  recovery.stats <- recovery.data %>%
    filter(Treatment_Group != "Untreated") %>%
    mutate(recovery_rate = .data[[index]] / baseline.value * 100) %>%
    select(Treatment_Group, all_of(index), recovery_rate)
  recovery.stats.list[[index]] <- recovery.stats
}

# ==================== 8. Cumulative Treatment Effect Analysis ====================

# 8.1 Create cumulative treatment effect data
treatment.effect.data <- alpha.data %>%
  filter(Treatment_Group != "Untreated") %>%
  mutate(Treatment_Count = case_when(
    Biofilm_Age == 32 ~ 1, Biofilm_Age == 33 ~ 2, Biofilm_Age == 34 ~ 3,
    Biofilm_Age == 35 ~ 4, Biofilm_Age == 36 ~ 4, TRUE ~ 0
  ))

# 8.2 Relationship between treatment count and diversity
treatment.effect.plots <- list()
for (index in diversity.indices) {
  if (all(is.na(treatment.effect.data[[index]]))) next

  p <- ggplot(treatment.effect.data, aes(x = Treatment_Count, y = .data[[index]], color =
Treatment_Group)) +
    geom_point(size = 3) +
    geom_smooth(method = "lm", se = TRUE) +
    scale_color_brewer(type = "qual", palette = "Set1") +
    labs(
      title = paste("Cumulative Treatment Effect -", diversity.labels[which(diversity.indices ==
index)]),
```

```
        x = "Number of Treatments", y = diversity.labels[which(diversity.indices == index)], color =
"Treatment Type"
    ) +
    theme_bw()

  treatment.effect.plots[[index]] <- p
}

# Display and save cumulative effect plots
for (index in names(treatment.effect.plots)) {
  print(treatment.effect.plots[[index]])
  ggsave(file.path(output_dir, paste0("cumulative_effect_", tolower(index), ".png")),
              treatment.effect.plots[[index]], width = 10, height = 6, dpi = 300)
}

# ==================== 9. Comprehensive Results Output ====================

# 9.1 Create combined figure
main.plots <- list(
  alpha.plots[["Shannon"]],
  pcoa.plot,
  genus.ts.plot,
  recovery.plots[["Shannon"]]
)
main.plots <- main.plots[!sapply(main.plots, is.null)]
if (length(main.plots) > 0) {
  combined.plot <- ggarrange(plotlist = main.plots, ncol = 2, nrow = 2,
                                        labels = c("A", "B", "C", "D"), common.legend = FALSE)
}
print("Combined Time Series Analysis Plot:")
print(combined.plot)
ggsave(file.path(output_dir, "combined_timeseries_analysis.png"),
             combined.plot, width = 18, height = 14, dpi = 300)

# 9.2 Export analysis data
write.csv(alpha.data, file.path(output_dir, "alpha_diversity_timeseries.csv"), row.names = FALSE)
write.csv(genus.ts.data,  file.path(output_dir,  "genus_timeseries_summary.csv"),  row.names  =
FALSE)

# 9.3 Export descriptive results to a text file
sink(file.path(output_dir, "descriptive_summary.txt"))
cat("=== Recovery Rate Statistics ===\n")
for (index in names(recovery.stats.list)) {
  cat("\n-----", index, "-----\n")
```

```
  print(recovery.stats.list[[index]])
}
sink()

# 9.4 Final summary report
cat("\n=== Time Series Analysis Summary ===\n")
cat("1. Analyzed", length(timeseries.weeks), "time points of microbiome dynamics.\n")
cat("2. Included", nsamples(physeq.ts), "samples covering",
      length(unique(sample_data(physeq.ts)$Treatment_Group)), "treatment groups.\n")
cat("3. Alpha diversity indices analyzed: Observed, Shannon, Simpson, Faith PD.\n")
cat("4. Beta diversity analyzed using:", tools::toTitleCase(dist.metric), "distance.\n")
cat("5. Taxonomic analysis performed at Genus level on", length(major.genera), "major genera.\n")
cat("6. All plots and data have been saved to:", output_dir, "\n")
cat("7. NOTE: No statistical tests were performed in this run.\n")
cat("\n✅ Time Series Analysis Complete!\n")

# ==================== Metal Ion Concentration vs Alpha Diversity Analysis ====================

# Load required packages
library(phyloseq)
library(ggplot2)
library(dplyr)
library(tidyr)
library(broom)
library(gridExtra)
library(cowplot)

# Create output directory if it doesn't exist
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/metal"
dir.create(output_dir, recursive = TRUE, showWarnings = FALSE)

# ==================== 1. Calculate Alpha Diversity Indices ====================
cat("Calculating alpha diversity indices...\n")

# Calculate observed richness (number of ASVs)
observed <- estimate_richness(physeq, measures = "Observed")

# Calculate Shannon diversity
shannon <- estimate_richness(physeq, measures = "Shannon")

# Calculate Simpson diversity
simpson <- estimate_richness(physeq, measures = "Simpson")
```

```
# Calculate Faith's phylogenetic diversity
if (!is.null(phy_tree(physeq))) {
  # Faith's PD requires a phylogenetic tree
  library(picante)

  # Get OTU table
  otu_mat <- as(otu_table(physeq), "matrix")
  if (taxa_are_rows(physeq)) {
    otu_mat <- t(otu_mat)
  }

  # Calculate Faith's PD with warning suppression for single-species communities
  suppressWarnings({
    faith_pd <- pd(otu_mat, phy_tree(physeq), include.root = FALSE)
  })

  faith <- data.frame(PD = faith_pd$PD, row.names = rownames(faith_pd))

  # Remove NA values (single-species communities)
  if (any(is.na(faith$PD))) {
    cat("Note: Some samples had single-species communities and were assigned NA for Faith's
PD\n")
  }
} else {
  cat("Warning: No phylogenetic tree found. Faith's PD will not be calculated.\n")
  faith <- NULL
}

# ==================== 2. Prepare Data for Analysis ====================
# Combine all alpha diversity indices
alpha_div <- cbind(observed, shannon, simpson)
if (!is.null(faith)) {
  alpha_div <- cbind(alpha_div, Faith = faith$PD)
}

# Get sample metadata
metadata <- as(sample_data(physeq), "data.frame")

# Define metal ion columns with the exact names from your metadata
metal_cols <- c("Ca.ng.500ul.", "Fe.ng.500ul.", "Cu.ng.500ul.",
                "Zn.ng.500ul.", "Ag.ng.500ul.", "Pb.ng.500ul.")

# Check which metal columns exist in metadata
```

```
existing_metals <- metal_cols[metal_cols %in% colnames(metadata)]
cat(paste("\nFound metal ion columns:", paste(existing_metals, collapse = ", "), "\n"))

# Combine alpha diversity with metadata
analysis_data <- cbind(alpha_div, metadata[rownames(alpha_div), existing_metals, drop = FALSE])

# ==================== 3. Filter Samples with Metal Ion Data ====================
# Remove samples with any NA values in metal concentrations
complete_rows <- complete.cases(analysis_data[, existing_metals])
analysis_data_clean <- analysis_data[complete_rows, ]
cat(paste("\nRetained", sum(complete_rows), "samples with complete metal ion data\n"))

# ==================== 4. Log Transform Metal Ion Concentrations ====================
# Add small constant to avoid log(0) issues
epsilon <- 0.001

# Create a mapping of metal names without units
metal_name_mapping <- list()

for (metal in existing_metals) {
  # Extract clean metal name - for your format "Ca.ng.500ul."
  clean_name <- gsub("\\.ng\\.500ul\\.", "", metal)

  metal_name_mapping[[metal]] <- clean_name

  # Create new column with log-transformed values
  log_col_name <- paste0("log_", clean_name)
  analysis_data_clean[[log_col_name]] <- log10(analysis_data_clean[[metal]] + epsilon)
}

# ==================== 5. Perform Linear Regression Analysis ====================
# Define diversity indices to analyze
diversity_indices <- c("Observed", "Shannon", "Simpson")
if (!is.null(faith)) {
  diversity_indices <- c(diversity_indices, "Faith")
}

# Get clean metal names from mapping
metals_clean <- unique(unlist(metal_name_mapping))

# Store regression results
regression_results <- list()
```

```
# Create a plot for each metal-diversity combination
plot_list <- list()
plot_counter <- 1

for (div_index in diversity_indices) {
  for (metal in metals_clean) {
    log_metal <- paste0("log_", metal)

    # Skip if columns don't exist
    if (!log_metal %in% colnames(analysis_data_clean) ||
        !div_index %in% colnames(analysis_data_clean)) {
      next
    }

    # Prepare data for this analysis
    plot_data <- data.frame(
      diversity = analysis_data_clean[[div_index]],
      log_concentration = analysis_data_clean[[log_metal]],
      metal = metal,
      diversity_type = div_index
    )

    # Remove any remaining NA values
    plot_data <- na.omit(plot_data)

    # Skip if no valid data points
    if (nrow(plot_data) < 3) {
      cat(paste("Skipping", div_index, "vs", metal, "- insufficient data points\n"))
      next
    }

    # Perform linear regression
    lm_model <- lm(diversity ~ log_concentration, data = plot_data)

    # Get model statistics
    summary_model <- summary(lm_model)
    r_squared <- summary_model$r.squared
    p_value <- summary_model$coefficients[2, 4]    # p-value for slope

    # Store results
    regression_results[[paste(div_index, metal, sep = "_")]] <- list(
      model = lm_model,
      r_squared = r_squared,
      p_value = p_value,
```

```
      metal = metal,
      diversity = div_index
    )

    # Create plot
    p <- ggplot(plot_data, aes(x = log_concentration, y = diversity)) +
      geom_point(size = 3, alpha = 0.7, color = "steelblue") +
      geom_smooth(method = "lm", se = TRUE, color = "darkred", fill = "lightpink", alpha = 0.3)
+
      labs(
        x = paste0("log10(", metal, " concentration [ng/500µl])"),
        y = paste0(div_index, " Index"),
        title = paste0(div_index, " Diversity vs ", metal, " Concentration")
      ) +
      theme_bw() +
      theme(
        plot.title = element_text(size = 14, face = "bold", hjust = 0.5),
        axis.title = element_text(size = 12),
        axis.text = element_text(size = 10),
        panel.grid.minor = element_blank()
      ) +
      # Add R² and p-value annotation
      annotate("text",
               x = Inf, y = Inf,
               label = paste0("R² = ", round(r_squared, 3), "\n",
                              "p = ", ifelse(p_value < 0.001, "< 0.001",
                                             format(round(p_value, 3), scientific =
FALSE))),
               hjust = 1.1, vjust = 1.1,
               size = 4, color = "darkred", fontface = "italic")

    # Store plot
    plot_list[[plot_counter]] <- p
    plot_counter <- plot_counter + 1
  }
}

# ==================== 6. Save Individual Plots ====================
cat("\nSaving individual plots...\n")

# Save plots based on regression results order
i <- 1
for (result_name in names(regression_results)) {
  if (i <= length(plot_list)) {
```

```
        # Extract metadata from result name
        parts <- strsplit(result_name, "_")[[1]]
        div_type <- parts[1]
        metal_name <- parts[2]

        # Create filename
        filename <- paste0(div_type, "_vs_", metal_name, "_regression.png")
        filepath <- file.path(output_dir, filename)

        # Save plot
        ggsave(filepath, plot_list[[i]], width = 8, height = 6, dpi = 300)
        cat(paste("Saved:", filename, "\n"))
        i <- i + 1
    }
}

# ==================== 7. Create Summary Plots ====================
# Create a combined plot for each diversity index
for (div_index in diversity_indices) {
    # Get plots for this diversity index
    div_plots <- list()
    plot_idx <- 1

    # Create a mapping of result names to plot indices
    plot_mapping <- setNames(seq_along(regression_results), names(regression_results))

    for (metal in metals_clean) {
        result_key <- paste(div_index, metal, sep = "_")
        if (result_key %in% names(regression_results) && result_key %in% names(plot_mapping)) {
            plot_index <- plot_mapping[result_key]
            if (plot_index <= length(plot_list)) {
                div_plots[[plot_idx]] <- plot_list[[plot_index]]
                plot_idx <- plot_idx + 1
            }
        }
    }

    if (length(div_plots) > 0) {
        # Arrange plots in a grid
        n_cols <- min(3, length(div_plots))
        n_rows <- ceiling(length(div_plots) / n_cols)

        combined_plot <- plot_grid(plotlist = div_plots, ncol = n_cols, nrow = n_rows)
```

```
        # Save combined plot
        filename <- paste0(div_index, "_all_metals_combined.png")
        filepath <- file.path(output_dir, filename)
        ggsave(filepath, combined_plot, width = 8 * n_cols, height = 6 * n_rows, dpi = 300)
        cat(paste("Saved combined plot:", filename, "\n"))
    }
}

# ==================== 8. Create Summary Table ====================
# Extract regression statistics
summary_table <- data.frame()

for (result_name in names(regression_results)) {
    result <- regression_results[[result_name]]

    # Get coefficient information
    coef_info <- summary(result$model)$coefficients

    summary_row <- data.frame(
        Diversity_Index = result$diversity,
        Metal = result$metal,
        R_squared = round(result$r_squared, 4),
        P_value = result$p_value,
        Slope = round(coef_info[2, 1], 4),
        Intercept = round(coef_info[1, 1], 4),
        Significant = ifelse(result$p_value < 0.05, "Yes", "No")
    )

    summary_table <- rbind(summary_table, summary_row)
}

# Check if we have any results
if (nrow(summary_table) > 0) {
    # Sort by diversity index and p-value
    summary_table <- summary_table %>%
        arrange(Diversity_Index, P_value)

    # Save summary table
    write.csv(summary_table,
                    file.path(output_dir, "regression_summary_table.csv"),
                    row.names = FALSE)

    # Print summary
    cat("\n==================== Summary of Results ====================\n")
```

```
  print(summary_table)
} else {
  cat("\nNo regression results to summarize. Please check your data.\n")
}

# ==================== 9. Create Heatmap of R² Values ====================
if (nrow(summary_table) > 0) {
  # Reshape data for heatmap
  heatmap_data <- summary_table %>%
    select(Diversity_Index, Metal, R_squared) %>%
    pivot_wider(names_from = Metal, values_from = R_squared)

  # Convert to matrix
  heatmap_matrix <- as.matrix(heatmap_data[, -1])
  rownames(heatmap_matrix) <- heatmap_data$Diversity_Index

  # Create heatmap plot
  library(reshape2)
  heatmap_long <- melt(heatmap_matrix)
  colnames(heatmap_long) <- c("Diversity", "Metal", "R_squared")

  p_heatmap <- ggplot(heatmap_long, aes(x = Metal, y = Diversity, fill = R_squared)) +
    geom_tile() +
    geom_text(aes(label = round(R_squared, 3)), color = "black", size = 4) +
    scale_fill_gradient2(low = "white", mid = "lightblue", high = "darkblue",
                         midpoint = 0.5, limits = c(0, 1),
                         name = "R²") +
    labs(title = "R² Values: Metal Concentrations vs Alpha Diversity",
         x = "Metal", y = "Diversity Index") +
    theme_minimal() +
    theme(
      plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
      axis.text.x = element_text(angle = 45, hjust = 1),
      axis.text = element_text(size = 12),
      axis.title = element_text(size = 14)
    )

  # Save heatmap
  ggsave(file.path(output_dir, "R_squared_heatmap.png"),
         p_heatmap, width = 10, height = 8, dpi = 300)

  cat("\n ✅ Analysis complete! All plots saved to:", output_dir, "\n")
  cat("\nFiles created:\n")
  cat("- Individual regression plots for each metal-diversity combination\n")
```

```
  cat("- Combined plots for each diversity index\n")
  cat("- regression_summary_table.csv\n")
  cat("- R_squared_heatmap.png\n")
} else {
  cat("\n ⚠️ No analysis could be performed. Please check your metadata columns.\n")
}

# ==================== 10. Print Significant Results ====================
if (nrow(summary_table) > 0) {
  significant_results <- summary_table %>%
    filter(P_value < 0.05) %>%
    arrange(P_value)

  if (nrow(significant_results) > 0) {
    cat("\n==================== Significant Associations (p < 0.05) ====================\n")
    print(significant_results)
  } else {
    cat("\nNo significant associations found (p < 0.05)\n")
  }
}
```
```{r}
# ======================================================================
# Script Objective: Analyze the impact of metal ion concentrations on Beta diversity
# Analysis Method: PCoA (based on Unweighted UniFrac distance) + environmental vector fitting
# ======================================================================

# Step 0: Load required R packages
# ----------------------------------------------------------------
# Ensure these packages are installed: install.packages(c("phyloseq", "vegan", "ggplot2", "dplyr", "RColorBrewer"))
library(phyloseq)
library(vegan)
library(ggplot2)
library(dplyr)
library(RColorBrewer)

cat(" ✅ Required packages loaded.\n")

# --- Check if physeq object exists ---
if (!exists("physeq")) {
  stop("Error: 'physeq' object not found in environment. Please run the previous script to create it.")
```

```
}


# ==================== 1. Data Preparation and Filtering ====================
# Goal: Keep only samples with complete metal ion concentration data
# ============================================================================
cat("\n[1/6] Filtering samples...\n")

# --- a. Define metal ion column names in metadata ---
# IMPORTANT: Check and update these column names according to your metadata file!
metal_cols <- c("Ca.ng.500ul.", "Fe.ng.500ul.", "Cu.ng.500ul.",
                "Zn.ng.500ul.", "Ag.ng.500ul.", "Pb.ng.500ul.")

# Extract metadata
metadata_df <- as(sample_data(physeq), "data.frame")

# Check required columns exist
if (!all(c("Treatment", "Biofilm_Status") %in% colnames(metadata_df))) {
  stop("Error: 'Treatment' or 'Biofilm_Status' column missing in metadata.")
}
if (!all(metal_cols %in% colnames(metadata_df))) {
  stop("Error: Missing some or all specified metal ion columns. Check 'metal_cols' definition.")
}

# --- b. Filter samples with no missing values in metal ion columns ---
complete_samples <- rownames(metadata_df)[complete.cases(metadata_df[, metal_cols])]

if (length(complete_samples) == 0) {
  stop("Error: No samples with complete metal ion data found. Analysis cannot proceed.")
}

cat(paste("      Found", length(complete_samples), "samples with complete metal ion data.\n"))

# --- c. Subset phyloseq object to these samples ---
physeq_filtered <- prune_samples(complete_samples, physeq)

cat(" ✅ Sample filtering complete.\n")


# ==================== 2. Calculate Beta Diversity Distance ====================
# Goal: Use Unweighted UniFrac to compute sample distances
# ============================================================================
cat("\n[2/6] Calculating Unweighted UniFrac distance...\n")
```

```
# --- a. Check if phylogenetic tree exists (required for UniFrac) ---
if (is.null(phy_tree(physeq_filtered))) {
  stop("Error: No phylogenetic tree found in phyloseq object. Cannot compute UniFrac distance.")
}

# --- b. Compute distance matrix ---
dist_unifrac <- UniFrac(physeq_filtered, weighted = FALSE)

cat("✅ Distance matrix computed.\n")


# ==================== 3. Perform PCoA Dimension Reduction ====================
# Goal: Reduce high-dimensional distance matrix for 2D visualization
# ============================================================================
cat("\n[3/6] Performing PCoA analysis...\n")

# --- a. Run PCoA using vegan::ordinate ---
pcoa_result <- ordinate(physeq_filtered, method = "PCoA", distance = dist_unifrac)

# --- b. Extract PCoA coordinates and merge with metadata for plotting ---
pcoa_scores <- as.data.frame(pcoa_result$vectors[, 1:2]) # Extract first two PCs
pcoa_data <- cbind(pcoa_scores, as(sample_data(physeq_filtered), "data.frame"))

# --- c. Calculate variance explained by each axis ---
variance_explained <- pcoa_result$values$Relative_eig * 100

cat(paste0("   PCoA axis explained variance: PC1 = ", round(variance_explained[1], 1), "%, PC2 = ", round(variance_explained[2], 1), "%\n"))
cat("✅ PCoA analysis complete.\n")


# ==================== 4. Environmental Vector Fitting ====================
# Goal: Identify metal ions significantly correlated with PCoA axes
# ============================================================================
cat("\n[4/6] Fitting metal ion vectors...\n")

# --- a. Prepare environmental data (only filtered samples) ---
env_data <- as(sample_data(physeq_filtered), "data.frame")[, metal_cols]

# --- b. Log10-transform concentrations (recommended), +1 to avoid log(0) ---
env_data_log <- log10(env_data + 1)
colnames(env_data_log) <- gsub("\\.ng\\.500ul\\.", "", colnames(env_data_log)) # Clean names for plotting
```

```
# --- c. Fit vectors using vegan::envfit ---
set.seed(123) # For reproducibility
env_fit_result <- envfit(pcoa_result$vectors[, 1:2], env_data_log, permutations = 999, na.rm = TRUE)

# --- d. Extract vector results and filter significant (p < 0.05) factors ---
vector_data <- as.data.frame(scores(env_fit_result, display = "vectors"))
vector_data$metal <- rownames(vector_data)
vector_data$r2 <- env_fit_result$vectors$r
vector_data$pval <- env_fit_result$vectors$pvals

significant_vectors <- vector_data[vector_data$pval < 0.05, ]

cat("       envfit analysis complete. Results summary:\n")
print(vector_data[, c("metal", "r2", "pval")])

if (nrow(significant_vectors) > 0) {
    cat(paste0("\n          Found ", nrow(significant_vectors), " significant metal ions: ", paste(significant_vectors$metal, collapse=", "), "\n"))
} else {
    cat("\n       No metal ions significantly correlated with community structure.\n")
}

cat(" ✅ Environmental vector fitting complete.\n")


# ==================== 5. Create ggplot2 Visualization ====================
# Goal: Generate PCoA plot with confidence ellipses and significant environmental vectors
# ========================================================================
cat("\n[5/6] Creating PCoA plot...\n")

# --- a. Prepare colors and shapes ---
# Color by Treatment
n_treatments <- length(unique(pcoa_data$Treatment))
color_palette <- brewer.pal(max(3, n_treatments), "Set1")

# Shape by Biofilm_Status
n_shapes <- length(unique(pcoa_data$Biofilm_Status))
shape_values <- c(16, 17, 15, 3, 7, 8)[1:n_shapes] # (circle, triangle, square, ...)

# --- b. Create PCoA plot ---
pcoa_plot <- ggplot(pcoa_data, aes(x = Axis.1, y = Axis.2, color = Treatment)) +
  # Plot sample points: color=Treatment, shape=Biofilm_Status
```

```
  geom_point(aes(shape = Biofilm_Status), size = 4, alpha = 0.8) +

  # ✨ ADD CONFIDENCE ELLIPSES ✨
  # Add ellipses for each 'Treatment' group
  stat_ellipse(
    aes(group = Treatment), # Group ellipses by Treatment
    type = 't',                    # Use t-distribution for ellipse calculation
    level = 0.95,                  # 95% confidence level
    linetype = 2,                  # Dashed line for the ellipse
    size = 1,                      # Line thickness
    show.legend = FALSE            # Hide legend for ellipses
  ) +

  # Customize colors and shapes
  scale_color_manual(values = color_palette) +
  scale_shape_manual(values = shape_values) +

  # Labels and titles
  labs(
    x = paste0("PC1 (", round(variance_explained[1], 1), "%)"),
    y = paste0("PC2 (", round(variance_explained[2], 1), "%)"),
    title = "PCoA of Unweighted UniFrac Distance with Confidence Ellipses",
    subtitle = "Arrows show significant metal ion correlations (p < 0.05)",
    color = "Treatment",
    shape = "Biofilm Status"
  ) +

  # Theme aesthetics
  theme_bw() +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    plot.subtitle = element_text(size = 12, hjust = 0.5, face = "italic"),
    legend.title = element_text(face = "bold"),
    panel.grid = element_blank()
  ) +
  # Maintain 1:1 aspect ratio for correct arrow angles
  coord_fixed()

# --- c. Add significant vectors if any ---
if (nrow(significant_vectors) > 0) {
  pcoa_plot <- pcoa_plot +
    # Draw arrows
    geom_segment(data = significant_vectors,
                 aes(x = 0, y = 0, xend = Axis.1, yend = Axis.2),
```

```
                    arrow = arrow(length = unit(0.3, "cm"), type = "closed"),
                    color = "darkred", size = 1, inherit.aes = FALSE) + # Use inherit.aes = FALSE
    # Add arrow labels
    geom_text(data = significant_vectors,
              aes(x = Axis.1 * 1.15, y = Axis.2 * 1.15, label = metal),
              color = "darkred", size = 3.5, fontface = "bold", inherit.aes = FALSE) # Use
inherit.aes = FALSE
}

# Print plot to R graphics device
print(pcoa_plot)
cat("✅ PCoA plot with ellipses created.\n")
# ==================== 6. Save Results ====================
# Goal: Save plots and statistics to files
# ============================================================
cat("\n[6/6] Saving results...\n")

# --- a. Create output directory ---
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/metal"
dir.create(output_dir, recursive = TRUE, showWarnings = FALSE)

# --- b. Save PCoA plot ---
plot_path <- file.path(output_dir, "PCoA_Unifrac_Metal_Vectors.png")
ggsave(plot_path, pcoa_plot, width = 10, height = 8, dpi = 300)
cat(paste("      -> Plot saved to:", plot_path, "\n"))

# --- c. Save envfit statistics ---
stats_path <- file.path(output_dir, "envfit_metal_ion_stats.csv")
write.csv(vector_data, stats_path, row.names = FALSE)
cat(paste("      -> Statistics saved to:", stats_path, "\n"))

# --- d. (Optional but recommended) Run PERMANOVA and save results ---
cat("      Running PERMANOVA tests...\n")
permanova_treatment <- adonis2(dist_unifrac ~ Treatment, data = pcoa_data, permutations =
999)
permanova_biofilm <- adonis2(dist_unifrac ~ Biofilm_Status, data = pcoa_data, permutations =
999)

# Write results to text file
permanova_path <- file.path(output_dir, "permanova_results.txt")
sink(permanova_path)
cat("### PERMANOVA Test for Treatment ###\n")
print(permanova_treatment)
cat("\n\n### PERMANOVA Test for Biofilm_Status ###\n")
```

```
print(permanova_biofilm)
sink() # Close file connection
cat(paste("        -> PERMANOVA results saved to:", permanova_path, "\n"))


cat("\n 🎉 🎉 🎉 Analysis complete! 🎉 🎉 🎉 \n")
```
```{r}
# ======================================================================
# Objective: Perform RDA or CCA analysis to assess the constraining effect of metal ions on microbial community structure
# ======================================================================

# Step 0: Load necessary R packages
# ----------------------------------------------------------------------
library(phyloseq)
library(vegan)
library(ggplot2)
library(dplyr)
library(RColorBrewer)

cat(" ✅ [0/7] Required packages loaded.\n")

if (!exists("physeq")) {
  stop("Error: 'physeq' object not found in environment.")
}

# ==================== NEW! Manual method selection ====================
# If DCA step consistently fails, manually specify method here ("RDA" or "CCA")
# RDA is the more commonly used and robust choice. Set to NULL for automatic DCA detection.
# ----------------------------------------------------------------------
MANUAL_ANALYSIS_METHOD <- "RDA" # <--- Manual selection here!


# ==================== 1. Data preparation and filtering ====================
cat("\n[1/7] Preparing and filtering data...\n")

# --- a. Define metal ion column names ---
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# !!! IMPORTANT: Based on VIF results below, manually remove high VIF variables !!!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
metal_cols <- c(
    # "Ca.ng.500ul.", # First removal, VIF=99.4
    # "Pb.ng.500ul.", # Second removal, VIF=31.4
```

```
        "Fe.ng.500ul.",
        "Cu.ng.500ul.",
        "Zn.ng.500ul.",
        "Ag.ng.500ul."
)

metadata_df <- as(sample_data(physeq), "data.frame")

if (!all(metal_cols %in% colnames(metadata_df))) {
        stop("Error: Misspelled column names in 'metal_cols'.")
}

complete_samples <- rownames(metadata_df)[complete.cases(metadata_df[, metal_cols])]

if (length(complete_samples) < 3) {
   stop("Error: Fewer than 3 samples with complete metal ion data.")
}

physeq_filtered <- prune_samples(complete_samples, physeq)
cat(paste("        Selected", nsamples(physeq_filtered), "samples with complete metal data.\n"))

species_data <- as(otu_table(physeq_filtered), "matrix")
if (taxa_are_rows(physeq_filtered)) {
   species_data <- t(species_data)
}
species_data <- species_data[, colSums(species_data) > 0]

if (ncol(species_data) < 2) {
        stop(paste("Error: Too few species (", ncol(species_data), ") remaining after filtering."))
}
cat(paste("        Filtered species count:", ncol(species_data), "proceed with analysis.\n"))

env_data <- as(sample_data(physeq_filtered), "data.frame")[, metal_cols, drop = FALSE]
env_data_log <- log10(env_data + 1)
colnames(env_data_log) <- gsub("\\.ng\\.500ul\\.", "", colnames(env_data_log))
cat("✅ Data preparation complete.\n")

# ==================== 2. Method selection (auto or manual) ====================
analysis_method <- NULL
if (!is.null(MANUAL_ANALYSIS_METHOD) && (MANUAL_ANALYSIS_METHOD == "RDA" ||
MANUAL_ANALYSIS_METHOD == "CCA")) {
        analysis_method <- MANUAL_ANALYSIS_METHOD
        cat(paste0("\n[2/7] Manually selected method: ", analysis_method, "\n"))
} else {
```

```
    cat("\n[2/7] Running DCA to determine RDA/CCA...\n")
    # Attempt DCA with error handling
    dca_result <- try(decorana(veg = species_data), silent = TRUE)
    if (inherits(dca_result, "try-error")) {
        cat("    DCA failed. Defaulting to RDA.\n")
        analysis_method <- "RDA"
    } else {
        dca1_length <- dca_result$rslts[1, 1]
        cat(paste0("    DCA completed. First axis length: ", round(dca1_length, 3), "\n"))
        analysis_method <- ifelse(dca1_length < 3.0, "RDA", "CCA")
        cat(paste0("    ---> Recommended method: ", analysis_method, "\n"))
    }
}

# ==================== 3. Environmental variable collinearity check (VIF) ====================
cat("\n[3/7] Checking environmental variable collinearity (VIF)...\n")
if (ncol(env_data_log) > 1) {
    # Hellinger transformation for VIF stability
    vif_model <- rda(decostand(species_data, method="hellinger") ~ ., data = env_data_log)
    vif_scores <- vif.cca(vif_model)
    cat("    VIF (Variance Inflation Factor) scores:\n")
    print(vif_scores)
    if (any(vif_scores > 10)) {
        cat("\n    WARNING: Variables with VIF > 10 detected. Return to Step 1 to remove high VIF variables.\n")
    } else {
        cat("    All VIFs < 10. Model stable.\n")
    }
} else {
    cat("    Only one environmental variable. Skipping VIF check.\n")
}
cat(" ✅ VIF check complete.\n")

# ==================== 4. Perform constrained ordination (RDA or CCA) ====================
cat(paste0("\n[4/7] Executing ", analysis_method, " analysis...\n"))
species_data_hellinger <- decostand(species_data, method = "hellinger")
if (analysis_method == "RDA") {
    constrained_model <- rda(species_data_hellinger ~ ., data = env_data_log)
} else {
    constrained_model <- cca(species_data ~ ., data = env_data_log)
}
cat(" ✅ Model built.\n")
```

```
# ==================== 5. Model significance testing ====================
cat("\n[5/7] Running permutation tests...\n")
set.seed(123)
model_anova <- anova.cca(constrained_model, permutations = 999)
axis_anova <- anova.cca(constrained_model, by = "axis", permutations = 999)
term_anova <- anova.cca(constrained_model, by = "terms", permutations = 999)
cat(" ✅ Significance testing complete.\n")

output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/metal"
dir.create(output_dir, recursive = TRUE, showWarnings = FALSE)
stats_path <- file.path(output_dir, paste0(analysis_method, "_summary_stats.txt"))
sink(stats_path)
cat(paste0("########### ", analysis_method, " ANALYSIS SUMMARY ###########\n\n"))
if (!is.null(MANUAL_ANALYSIS_METHOD)){
    cat(paste0("### METHOD SELECTION: Manually specified as '",
MANUAL_ANALYSIS_METHOD, "' ###\n\n"))
} else {
    cat("### DCA DECISION ###\n"); print(dca_result$rslts[1,1]); cat("\n")
}
if (ncol(env_data_log) > 1) {cat("### VIF DIAGNOSTICS ###\n"); print(vif_scores); cat("\n")}
cat("### OVERALL MODEL SIGNIFICANCE ###\n"); print(model_anova); cat("\n")
cat("### CONSTRAINED AXES SIGNIFICANCE ###\n"); print(axis_anova); cat("\n")
cat("### TERM SIGNIFICANCE (METALS) ###\n"); print(term_anova); cat("\n")
cat("### MODEL SUMMARY ###\n"); print(summary(constrained_model)); cat("\n")
sink()
cat(paste0("    Summary saved to: ", stats_path, "\n"))

# ==================== 6. Visualization (Triplot) ====================
# Goal: Create publication-quality triplot handling potential absence of significant constraints
# ====================================================================
cat("\n[6/7] Creating triplot visualization...\n")

# --- a. Extract plot data ---
# scaling=2 favors Euclidean sample relationships
plot_scores <- scores(constrained_model, choices = c(1, 2), display = c("sites", "bp"), scaling = 2)

# Sample scores always exist
sites_scores <- as.data.frame(plot_scores$sites)
sites_scores <- cbind(sites_scores, as(sample_data(physeq_filtered), "data.frame"))

# --- b. Check for environmental variable scores ---
# bp_scores only exist if constraints were found
if (!is.null(plot_scores$bp) && nrow(plot_scores$bp) > 0) {
```

```
    bp_scores <- as.data.frame(plot_scores$bp)
    bp_scores$metal <- rownames(bp_scores)
    arrows_exist <- TRUE
} else {
    bp_scores <- NULL
    arrows_exist <- FALSE
    cat("    NOTE: No significant constraints detected. Omitting metal vectors.\n")
}

# --- c. Extract axis explanatory power ---
if (!is.null(constrained_model$CCA) && length(constrained_model$CCA$eig) > 0) {
    variance_explained <- constrained_model$CCA$eig / sum(constrained_model$CCA$eig) *
100
    xlab_text <- paste0(names(sites_scores)[1], " (", round(variance_explained[1], 1), "%)")
    ylab_text <- paste0(names(sites_scores)[2], " (", round(variance_explained[2], 1), "%)")
} else {
    # Default to PC axis labels if no constraints
    variance_explained <- constrained_model$CA$eig / sum(constrained_model$CA$eig) * 100
    xlab_text <- paste0("PC1 (", round(variance_explained[1], 1), "%)")
    ylab_text <- paste0("PC2 (", round(variance_explained[2], 1), "%)")
}


# --- d. ggplot2 plotting ---
rda_plot <- ggplot() +
  # Plot samples
  geom_point(data = sites_scores,
             aes_string(x = names(sites_scores)[1], y = names(sites_scores)[2], color =
"Treatment", shape = "Biofilm_Status"),
             size = 4, alpha = 0.8) +

  # Add confidence ellipses
  stat_ellipse(data = sites_scores,
               aes_string(x = names(sites_scores)[1], y = names(sites_scores)[2], color =
"Treatment"),
               level = 0.95, type = 't', linetype = 2, size = 1, show.legend = FALSE) +

  # Set aesthetics
  scale_color_brewer(palette = "Set1") +
  scale_shape_manual(values = c(16, 17, 15, 3, 7, 8)) +

  # Axis labels and title
  labs(
    x = xlab_text,
```

```
      y = ylab_text,
      title = paste(analysis_method, "Metal Ions Constraint on Community Structure"),
      color = "Treatment",
      shape = "Biofilm Status"
    ) +

    # Theme
    theme_bw() +
    theme(
      plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
      legend.title = element_text(face = "bold"),
      panel.grid = element_blank()
    ) +
    geom_hline(yintercept = 0, linetype = 'dashed', color = 'gray') +
    geom_vline(xintercept = 0, linetype = 'dashed', color = 'gray') +
    coord_fixed()

# --- e. NEW! Add arrow layer if arrows exist ---
if (arrows_exist) {
    rda_plot <- rda_plot +
      # Plot metal vectors
      geom_segment(data = bp_scores,
                   aes(x = 0, y = 0, xend = .data[[names(bp_scores)[1]]], yend
= .data[[names(bp_scores)[2]]]),
                   arrow = arrow(length = unit(0.3, "cm"), type = "closed"),
                   color = "darkred", size = 1) +
      # Add metal labels
      geom_text(data = bp_scores,
                aes(x = .data[[names(bp_scores)[1]]] * 1.1, y = .data[[names(bp_scores)[2]]] *
1.1, label = metal),
                color = "darkred", size = 3.5, fontface = "bold")
}

# Display plot
print(rda_plot)
cat("✅ Visualization created.\n")


# ==================== 7. Save results ====================
cat("\n[7/7] Saving results...\n")

plot_path <- file.path(output_dir, paste0(analysis_method, "_Triplot_Metal_Ions.png"))
ggsave(plot_path, rda_plot, width = 10, height = 8, dpi = 300)
cat(paste("       -> Plot saved to:", plot_path, "\n"))
```

```
cat("\n 🎉 🎉 🎉 ANALYSIS COMPLETE! 🎉 🎉 🎉 \n")

# ======================================================================
# Script Objective: Use Generalized Additive Models (GAM) to explore
# non-linear relationships between metal ions and community structure
# ======================================================================

# Step 0: Load required R packages
# ----------------------------------------------------------------
# If mgcv is not installed, run: install.packages("mgcv")
library(mgcv)
library(ggplot2)
library(dplyr)
library(cowplot)

cat(" ✅ [0/5] Required packages for GAM analysis loaded.\n")

# --- Check if PCoA results exist ---
# This script relies on the pcoa_data object created in the previous PCoA script
if (!exists("pcoa_data")) {
  stop("Error: 'pcoa_data' object not found in environment. Please run the PCoA analysis script first.")
}


# ==================== 1. Prepare data for GAM analysis ====================
# Objective: Create a clean data frame with PC1 scores and log-transformed
# metal ion concentrations
# ======================================================================
cat("\n[1/5] Preparing GAM analysis data...\n")

# --- a. Identify metal ion columns to analyze ---
# Use the same metal ion columns as in the PCoA analysis
metal_cols_for_gam <- c("Ca.ng.500ul.", "Fe.ng.500ul.", "Cu.ng.500ul.",
                        "Zn.ng.500ul.", "Ag.ng.500ul.", "Pb.ng.500ul.")

# --- b. Extract and log-transform from pcoa_data ---
gam_analysis_data <- pcoa_data %>%
  select(Axis.1, all_of(metal_cols_for_gam)) %>%
  # Log10-transform metal ion data, +1 to avoid log(0)
  mutate(across(all_of(metal_cols_for_gam), ~ log10(.x + 1)))

# Clean column names for easier handling
```

```
colnames(gam_analysis_data) <- gsub("\\.ng\\.500ul\\.", "", colnames(gam_analysis_data))
metals_to_analyze <- gsub("\\.ng\\.500ul\\.", "", metal_cols_for_gam)

cat("✅ GAM data preparation complete.\n")


# ==================== 2. Loop through GAM analysis and visualization ====================
# Objective: Fit GAM models for each metal ion and generate plots with fitted curves
# ======================================================================
cat("\n[2/5] Running GAM analysis in loop...\n")

# Create a list to store each plot
gam_plots <- list()
# Create a data frame to store model statistics
gam_results <- data.frame()

for (metal in metals_to_analyze) {
  cat(paste("      -> Analyzing:", metal, "\n"))

  # --- a. Construct model formula ---
  # We want to see how PC1 changes with metal concentration
  gam_formula <- as.formula(paste("Axis.1 ~ s(", metal, ")"))

  # --- b. Fit GAM model ---
  gam_model <- gam(gam_formula, data = gam_analysis_data)

  # --- c. Extract model summary and statistics ---
  model_summary <- summary(gam_model)

  p_value <- model_summary$s.pv[1]          # p-value for smooth term
  r_squared <- model_summary$r.sq           # model R-squared
  edf <- model_summary$edf[1]               # effective degrees of freedom (>1 indicates non-linearity)

  # Store results in data frame
  gam_results <- rbind(gam_results, data.frame(
    Metal = metal,
    R_squared = r_squared,
    p_value = p_value,
    EDF = edf,
    Is_Significant = ifelse(p_value < 0.05, "Yes", "No")
  ))
```

```
  # --- d. Create visualization plot ---
  p <- ggplot(gam_analysis_data, aes_string(x = metal, y = "Axis.1")) +
    geom_point(color = "steelblue", alpha = 0.7, size = 3) +
    # Use geom_smooth with 'gam' method to draw fitted curve and confidence interval
    geom_smooth(method = "gam", formula = y ~ s(x), se = TRUE, color = "darkred", fill = "lightpink") +
    labs(
      title = paste("GAM: PC1 vs.", metal),
      x = paste("log10(", metal, " concentration)"),
      y = "PCoA Axis 1 Score"
    ) +
    # Annotate key statistics on the plot
    annotate("text", x = Inf, y = Inf, hjust = 1.1, vjust = 1.2,
             label = paste0("R² = ", round(r_squared, 3),
                            "\np = ", format.pval(p_value, digits = 3),
                            "\nEDF = ", round(edf, 2)),
             size = 4, color = "darkred", fontface = "italic") +
    theme_bw() +
    theme(plot.title = element_text(hjust = 0.5, face = "bold"))

  # Store plot in list
  gam_plots[[metal]] <- p
}
cat("✅ GAM analysis for all metal ions completed.\n")


# ==================== 3. Summarize and display results ====================
cat("\n[3/5] Summarizing statistical results...\n")

# Sort results by p-value
gam_results <- gam_results %>% arrange(p_value)

# Print summary table
cat("--- GAM Analysis Summary Table ---\n")
print(gam_results)


# ==================== 4. Combine plots ====================
cat("\n[4/5] Combining all plots...\n")

# Use cowplot to combine all plots into one figure
combined_plot <- plot_grid(plotlist = gam_plots, ncol = 3) # Adjust columns as needed

# Display combined plot
```

```
print(combined_plot)


# ==================== 5. Save results ====================
cat("\n[5/5] Saving results...\n")

output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/metal" # Ensure output directory exists

# --- a. Save results summary ---
gam_table_path <- file.path(output_dir, "GAM_summary_table.csv")
write.csv(gam_results, gam_table_path, row.names = FALSE)
cat(paste("       -> GAM statistics table saved to:", gam_table_path, "\n"))

# --- b. Save combined plot ---
gam_plot_path <- file.path(output_dir, "GAM_combined_plots.png")
# Adjust width and height to fit your plot count
ggsave(gam_plot_path, combined_plot, width = 15, height = 10, dpi = 300)
cat(paste("       -> GAM combined plot saved to:", gam_plot_path, "\n"))

cat("\n🎉 🎉 🎉 GAM analysis complete! 🎉 🎉 🎉\n")

# ====================================================================
# Script Objective:    Quantify how metal-ion concentrations influence
#                                bacterial genus abundance in **UNTREATED** samples
# Core Methods:    1. Spearman correlation heatmap (global scan)
#                            2. GAM non-linear models (focus on Legionella)
# ====================================================================

# --------------------------------------------------------------
# Step 0.    Load required R packages
# --------------------------------------------------------------
library(phyloseq)
library(dplyr)
library(ggplot2)
library(pheatmap)
library(mgcv)
library(cowplot)

cat("✅ [0/5] Required packages loaded.\n")

# Stop if physeq object is not found
if (!exists("physeq")) {
   stop("Error: object 'physeq' not found in the environment.")
}
```

```
# ==================== 1. Data Preparation ==========================
# Goal: keep only "Untreated" samples, aggregate ASVs to genus level,
#       remove unwanted genera, convert to relative abundance, and
#       merge with metal-ion data
# ===================================================================
cat("\n[1/5] Preparing data...\n")

# --- a. Subset to "Untreated" samples only ---
physeq_subset <- subset_samples(physeq, Treatment == "Untreated")
cat(paste("  Retained", nsamples(physeq_subset), "untreated samples.\n"))

# --- b. Aggregate ASVs to genus level ---
physeq_genus <- tax_glom(physeq_subset, taxrank = "Genus", NArm = FALSE)

# --- c. Remove unwanted genera (NA or "uncultured") ---
tax_table_genus <- as.data.frame(tax_table(physeq_genus))
genera_to_remove <- tax_table_genus %>%
  filter(is.na(Genus) | grepl("uncultured", Genus, ignore.case = TRUE) | Genus == "NA") %>%
  rownames()
all_genera <- taxa_names(physeq_genus)
genera_to_keep <- setdiff(all_genera, genera_to_remove)
physeq_genus_filtered <- prune_taxa(genera_to_keep, physeq_genus)
cat(paste("  After removing NA/Uncultured, kept", ntaxa(physeq_genus_filtered),
"genera.\n"))

# --- d. Convert to relative abundance (%) ---
physeq_genus_rel <- transform_sample_counts(physeq_genus_filtered, function(x) x / sum(x) *
100)

# --- e. Build final analysis dataframe ---
genus_abundance <- as.data.frame(otu_table(physeq_genus_rel))
tax_map <- tax_table(physeq_genus_rel)[, "Genus"]
rownames(genus_abundance) <- tax_map[rownames(genus_abundance)]

metal_cols <- c("Ca.ng.500ul.", "Fe.ng.500ul.", "Cu.ng.500ul.",
                "Zn.ng.500ul.", "Ag.ng.500ul.", "Pb.ng.500ul.")
metadata_df <- as(sample_data(physeq_genus_rel), "data.frame")

# Keep only samples with complete metal data
complete_metal_samples <- rownames(metadata_df)[complete.cases(metadata_df[,
metal_cols])]
if (length(complete_metal_samples) < nrow(metadata_df)) {
    cat(paste("  Removed", nrow(metadata_df) - length(complete_metal_samples),
```

```
                    "samples with incomplete metal data.\n"))
}

metal_data_log <- metadata_df[complete_metal_samples, ] %>%
  select(all_of(metal_cols)) %>%
  mutate(across(everything(), ~ log10(.x + 1))) %>%
  rename_with(~ gsub("\\.ng\\.500ul\\.", "", .x))

analysis_df <- t(genus_abundance[, complete_metal_samples]) %>%
  as.data.frame() %>%
  merge(metal_data_log, by = "row.names") %>%
  rename(SampleID = Row.names)

cat("✅ Data preparation finished.\n")

# ==================== 2. Global Scan: Correlation Heatmap ===========
cat("\n[2/5] Computing correlations and drawing heatmap...\n")

metals <- colnames(metal_data_log)
genera <- setdiff(colnames(analysis_df), c("SampleID", metals))

cor_matrix  <- matrix(NA, nrow = length(genera), ncol = length(metals),
                      dimnames = list(genera, metals))
pval_matrix <- cor_matrix

for (g in genera) {
  for (m in metals) {
    if (var(analysis_df[[g]]) == 0) next  # skip invariant genera
    cor_test <- cor.test(analysis_df[[g]], analysis_df[[m]], method = "spearman")
    cor_matrix[g, m]  <- cor_test$estimate
    pval_matrix[g, m] <- cor_test$p.value
  }
}

# Drop rows with all NAs
keep_rows <- rowSums(is.na(cor_matrix)) != ncol(cor_matrix)
cor_matrix  <- cor_matrix[keep_rows, ]
pval_matrix <- pval_matrix[rownames(cor_matrix), ]

sig_matrix  <- ifelse(pval_matrix < 0.05, "*", "")

# Select top 30 most abundant genera still present in cor_matrix
top_genera_names <- names(sort(colMeans(analysis_df[, genera]), decreasing = TRUE))[1:min(30,
length(genera))]
```

```
top_genera_names <- intersect(top_genera_names, rownames(cor_matrix))

pheatmap_plot <- pheatmap(
  cor_matrix[top_genera_names, ],
  color = colorRampPalette(c("blue", "white", "red"))(100),
  display_numbers = sig_matrix[top_genera_names, ],
  fontsize_number = 15,
  cluster_rows = TRUE,
  cluster_cols = TRUE,
  main = "Top 30 Genera vs. Metal Ions (Untreated Samples Only)",
  fontsize_row = 8
)
cat(" ✅ Correlation heatmap generated.\n")

# ==================== 3. Focus: GAM for Legionella ================
# Goal: dynamically adjust model complexity and pass the correct k
#             into geom_smooth to prevent plotting errors
# ================================================================
cat("\n[3/5] Building GAM models for Legionella (Untreated samples only)...\n")

target_genus <- "Legionella"
if (!target_genus %in% colnames(analysis_df)) {
  cat(paste("       Warning: genus '", target_genus, "' not found in untreated data – skipping.\n"))
} else {
  legionella_plots   <- list()
  legionella_results <- data.frame()

  for (metal in metals) {
    cat(paste("       -> Analysing Legionella vs.", metal, "\n"))

    n_unique <- length(unique(analysis_df[[metal]]))

    # --- b. Dynamic model choice ---
    if (n_unique < 4) {
      cat("             Note: <4 unique values – falling back to linear model (lm).\n")
      lm_formula <- as.formula(paste("`", target_genus, "` ~", metal))
      model      <- lm(lm_formula, data = analysis_df)
      model_sum  <- summary(model)

      p_value   <- model_sum$coefficients[2, 4]
      r_squared <- model_sum$r.squared
      edf       <- 1

      # Plot LM
```

```
    p <- ggplot(analysis_df, aes_string(x = metal, y = paste0("`", target_genus, "`"))) +
      geom_point(color = "purple", size = 3) +
      geom_smooth(method = "lm", formula = y ~ x, color = "black") +
      labs(
        title = paste("LM: Legionella vs.", metal),
        subtitle = "(unique points < 4, used linear model)",
        y = "Legionella Relative Abundance (%)"
      )

  } else {
    # GAM branch
    k_val     <- n_unique - 1
    gam_formula <- as.formula(paste0("`", target_genus, "` ~ s(", metal, ", k=", k_val, ")"))

    gam_model <- try(gam(gam_formula, data = analysis_df, family = quasibinomial()), silent =
TRUE)
    if (inherits(gam_model, "try-error")) {
      gam_model <- gam(gam_formula, data = analysis_df)
    }

    model_sum <- summary(gam_model)

    p_value   <- if (!is.null(model_sum$s.pv)) model_sum$s.pv[1] else NA
    r_squared <- if (!is.null(model_sum$r.sq)) model_sum$r.sq else model_sum$r.squared
    edf       <- if (!is.null(model_sum$edf)) model_sum$edf[1] else NA

    # Plot GAM – IMPORTANT: pass dynamic k to geom_smooth as well
    p <- ggplot(analysis_df, aes_string(x = metal, y = paste0("`", target_genus, "`"))) +
      geom_point(color = "purple", size = 3) +
      geom_smooth(method = "gam",
                  formula = as.formula(paste0('y ~ s(x, k = ', k_val, ')')),
                  color = "black") +
      labs(
        title = paste("GAM: Legionella vs.", metal),
        y = "Legionella Relative Abundance (%)"
      )
  }

  # --- c. Collect results ---
  legionella_results <- rbind(legionella_results, data.frame(
    Metal = metal, R_squared = r_squared, p_value = p_value, EDF = edf
  ))

  p <- p +
```

```
        annotate("text", x = Inf, y = Inf, hjust = 1.1, vjust = 1.2,
                 label = paste0("R² = ", round(r_squared, 3),
                                "\np = ", format.pval(p_value, digits = 3),
                                "\nEDF = ", round(edf, 2))) +
        theme_bw()

      legionella_plots[[metal]] <- p
    }

    # --- d. Combine and display ---
    legionella_combined_plot <- plot_grid(plotlist = legionella_plots, ncol = 3)
    print(legionella_combined_plot)

    # --- e. Print summary ---
    cat("\n--- Legionella GAM/LM results summary (Untreated samples only) ---\n")
    print(legionella_results %>% arrange(p_value))

    cat(" ✅ Legionella analysis complete.\n")
}

# ==================== 4. Save Results ==============================
cat("\n[4/5] Saving results...\n")
output_dir <- "C:/Users/ASUS/Desktop/imeta/plots/metal"
dir.create(output_dir, recursive = TRUE, showWarnings = FALSE)

heatmap_path <- file.path(output_dir, "Correlation_Heatmap_Untreated.png")
png(heatmap_path, width = 800, height = 1000)
print(pheatmap_plot)
dev.off()
cat(paste("       -> Heatmap saved to:", heatmap_path, "\n"))

if (exists("legionella_combined_plot")) {
    legionella_plot_path <- file.path(output_dir, "GAM_Legionella_Untreated.png")
    ggsave(legionella_plot_path, legionella_combined_plot, width = 12, height = 8, dpi = 300)
    cat(paste("       -> Legionella plot saved to:", legionella_plot_path, "\n"))

    legionella_table_path <- file.path(output_dir, "GAM_Legionella_summary_Untreated.csv")
    write.csv(legionella_results, legionella_table_path, row.names = FALSE)
    cat(paste("       -> Legionella summary saved to:", legionella_table_path, "\n"))
}

# ==================== 5. Final Notes ==============================
cat("\n[5/5] Analysis complete!\n")
cat("       -> Inspect the heatmap and GAM plots for baseline relationships in untreated
```

samples.\n")

The following script is for the network analysis on genus level in the study

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import spearmanr, kruskal
import os
from matplotlib.lines import Line2D

# ==================== 1. Configuration & File Paths ====================
# --- User-configurable parameters ---
# Base path for input and output files
BASE_PATH = "C:/Users/ASUS/Desktop/imeta"
OUTPUT_PATH = os.path.join(BASE_PATH, "plots", "network")

# Network construction parameters
# We will use Spearman correlation. An edge is created if:
# 1. The absolute correlation coefficient is > CORR_THRESHOLD
# 2. The p-value is < P_VALUE_THRESHOLD
CORR_THRESHOLD = 0.6
P_VALUE_THRESHOLD = 0.05

# Target ASVs for Legionella
LEGIONELLA_ASVS = ['ASV_466', 'ASV_496', 'ASV_621', 'ASV_696', 'ASV_734', 'ASV_936', 'ASV_939']
LEGIONELLA_GENUS = 'g__Legionella'

# --- Define file paths ---
metadata_path = os.path.join(BASE_PATH, "metadata.xlsx")
asv_table_path = os.path.join(BASE_PATH, "result", "asv_table.csv")
taxonomy_path = os.path.join(BASE_PATH, "result", "taxonomy_table.xlsx")

# Create output directory if it doesn't exist
os.makedirs(OUTPUT_PATH, exist_ok=True)

print(f"Output will be saved to: {OUTPUT_PATH}")

# ==================== 2. Load and Prepare Data ====================
print("\n--- Loading Data ---")
```

```
try:
    # Load metadata, ASV table, and taxonomy
    metadata_df = pd.read_excel(metadata_path, index_col='sample_id')
    asv_df = pd.read_csv(asv_table_path, index_col='ASV_id')
    tax_df = pd.read_excel(taxonomy_path, index_col='ASV_id')

    # Ensure sample IDs match between metadata and ASV table
    common_samples = metadata_df.index.intersection(asv_df.columns)
    metadata_df = metadata_df.loc[common_samples]
    asv_df = asv_df[common_samples]

    # Ensure ASV IDs match between ASV table and taxonomy
    common_asvs = asv_df.index.intersection(tax_df.index)
    asv_df = asv_df.loc[common_asvs]
    tax_df = tax_df.loc[common_asvs]

    print(f"Loaded and aligned data:")
    print(f"    {len(common_samples)} samples")
    print(f"    {len(common_asvs)} ASVs")

except FileNotFoundError as e:
    print(f"Error: {e}. Please ensure file paths are correct.")
    exit()

# ==================== 3. Network Analysis Helper Functions ====================

def build_correlation_network(abundance_df, corr_thresh, pval_thresh):
    """Builds a co-occurrence network from an abundance table."""
    # Transpose so taxa are columns and samples are rows
    df_transposed = abundance_df.T

    # Calculate Spearman correlation matrix and p-value matrix
    corr_matrix, pval_matrix = spearmanr(df_transposed)

    # Create pandas DataFrames for easier handling
    corr_df = pd.DataFrame(corr_matrix, index=df_transposed.columns,
columns=df_transposed.columns)
    pval_df = pd.DataFrame(pval_matrix, index=df_transposed.columns,
columns=df_transposed.columns)

    # Create graph
    G = nx.Graph()
    taxa = df_transposed.columns
    for i in range(len(taxa)):
```

```
            for j in range(i + 1, len(taxa)):
                taxon1 = taxa[i]
                taxon2 = taxa[j]
                correlation = corr_df.loc[taxon1, taxon2]
                p_value = pval_df.loc[taxon1, taxon2]

                if abs(correlation) > corr_thresh and p_value < pval_thresh:
                    G.add_edge(taxon1, taxon2, weight=correlation,
                               color='#1f78b4' if correlation > 0 else '#e31a1c')
        return G

def plot_subnetwork(G, target_nodes, title, output_filename):
        """Finds neighbors of target nodes and plots the resulting subnetwork."""
        if not any(node in G for node in target_nodes):
            print(f"Warning: None of the target nodes {target_nodes} found in the network.")
            return None, None

        # Identify all neighbors of the target nodes
        neighbors = set()
        for node in target_nodes:
            if node in G:
                neighbors.update(G.neighbors(node))

        # Combine target nodes and their neighbors
        subgraph_nodes = set(target_nodes).union(neighbors)
        sub_G = G.subgraph(subgraph_nodes)

        # Plotting
        plt.style.use('default')
        fig, ax = plt.subplots(figsize=(14, 14))

        pos = nx.spring_layout(sub_G, k=0.8, iterations=50, seed=42)

        # Node colors
        node_colors = []
        for node in sub_G.nodes():
            if node in target_nodes:
                node_colors.append('#ff4d4d')    # Bright red for target
            elif node in neighbors:
                node_colors.append('#4d94ff')    # Blue for neighbors
            else:
                node_colors.append('#cccccc') # Grey for others

        # Edge colors
```

```
    edge_colors = [sub_G[u][v]['color'] for u, v in sub_G.edges()]

    nx.draw_networkx_nodes(sub_G, pos, node_size=1500, node_color=node_colors, alpha=0.9)
    nx.draw_networkx_edges(sub_G, pos, width=1.5, edge_color=edge_colors, alpha=0.7)
    nx.draw_networkx_labels(sub_G, pos, font_size=10, font_family='sans-serif', font_weight='bold')

    # Add legend for edge colors
    legend_elements = [
        Line2D([0], [0], color='#1f78b4', lw=2, label='Positive Correlation'),
        Line2D([0], [0], color='#e31a1c', lw=2, label='Negative Correlation')
    ]
    ax.legend(handles=legend_elements, loc='upper right', title='Correlation')

    ax.set_title(title, fontsize=20, fontweight='bold')
    plt.tight_layout()
    plt.savefig(output_filename, dpi=300, bbox_inches='tight')
    plt.close()

    print(f"Network plot saved to: {output_filename}")

    # Return neighbors for downstream analysis (excluding the targets themselves)
    return sub_G, neighbors.difference(target_nodes)

# ==================== 4. ASV-Level Network Analysis ====================
print("\n--- Starting ASV-Level Network Analysis ---")
asv_network = build_correlation_network(asv_df, CORR_THRESHOLD, P_VALUE_THRESHOLD)
print(f"ASV network built with {asv_network.number_of_nodes()} nodes and {asv_network.number_of_edges()} edges.")

asv_subgraph, asv_neighbors = plot_subnetwork(
    asv_network,
    LEGIONELLA_ASVS,
    'Co-occurrence Network of Legionella ASVs and Their Neighbors',
    os.path.join(OUTPUT_PATH, 'asv_level_legionella_network.png')
)

if asv_neighbors:
    print(f"Found {len(asv_neighbors)} neighbors for the target Legionella ASVs.")
else:
    print("No direct neighbors found for the target Legionella ASVs at the specified thresholds.")

# ==================== 5. Genus-Level Network Analysis ====================
```

```
print("\n--- Starting Genus-Level Network Analysis ---")
# Aggregate ASV table to genus level
genus_df = asv_df.join(tax_df['Genus']).groupby('Genus').sum()

# Filter out uncultured/unidentified genera
unwanted_genera = [g for g in genus_df.index if 'uncultured' in str(g) or 'unclassified' in str(g) or pd.isna(g)]
genus_df_cleaned = genus_df.drop(index=unwanted_genera, errors='ignore')
print(f"Aggregated to {genus_df_cleaned.shape[0]} genera after cleaning.")

genus_network = build_correlation_network(genus_df_cleaned, CORR_THRESHOLD, P_VALUE_THRESHOLD)
print(f"Genus network built with {genus_network.number_of_nodes()} nodes and {genus_network.number_of_edges()} edges.")

# Find neighbors of Legionella genus
legionella_genus_name_in_data = next((g for g in genus_df_cleaned.index if isinstance(g, str) and 'Legionella' in g), None)
genus_neighbors = None
if legionella_genus_name_in_data and legionella_genus_name_in_data in genus_network:
    genus_neighbors = set(genus_network.neighbors(legionella_genus_name_in_data))
    print(f"Found {len(genus_neighbors)} neighbors for the Legionella genus.")
    print("Genus Neighbors:", sorted(list(genus_neighbors)))
else:
    print("Error: Legionella genus not found in the dataset or it has no connections in the network.")

# Plot the ENTIRE genus network, highlighting Legionella and its neighbors
plt.style.use('default')
fig, ax = plt.subplots(figsize=(20, 20))
pos = nx.spring_layout(genus_network, k=0.5, iterations=50, seed=42)

# Define node colors for the full network
full_network_node_colors = []
for node in genus_network.nodes():
    if node == legionella_genus_name_in_data:
        full_network_node_colors.append('#ff4d4d')    # Bright red for Legionella
    elif genus_neighbors and node in genus_neighbors:
        full_network_node_colors.append('#4d94ff')    # Blue for neighbors
    else:
        full_network_node_colors.append('#cccccc') # Grey for all other nodes

edge_colors = [genus_network[u][v]['color'] for u, v in genus_network.edges()]
```

```
nx.draw_networkx_nodes(genus_network, pos, node_size=800, node_color=full_network_node_colors, alpha=0.9)
nx.draw_networkx_edges(genus_network, pos, width=1.0, edge_color=edge_colors, alpha=0.6)
nx.draw_networkx_labels(genus_network, pos, font_size=8, font_family='sans-serif')

# Add legend for edge colors
legend_elements = [
    Line2D([0], [0], color='#1f78b4', lw=4, label='Positive Correlation'),
    Line2D([0], [0], color='#e31a1c', lw=4, label='Negative Correlation')
]
ax.legend(handles=legend_elements, loc='best', title='Correlation')

ax.set_title('Complete Genus-Level Co-occurrence Network', fontsize=25, fontweight='bold')
plt.tight_layout()
output_filename_full = os.path.join(OUTPUT_PATH, 'genus_level_full_network.png')
plt.savefig(output_filename_full, dpi=300, bbox_inches='tight')
plt.close()
print(f"Full genus-level network plot saved to: {output_filename_full}")


# ==================== 6. Analyze Treatment Effect on Genus Neighbors ====================
if genus_neighbors:
    print("\n--- Analyzing Treatment Effect on Legionella's Neighbors ---")

    # Get abundance data for the neighbors
    neighbor_abundances = genus_df_cleaned.loc[list(genus_neighbors)].T

    # Merge with treatment metadata
    analysis_df = neighbor_abundances.join(metadata_df['Treatment'])

    # Calculate number of rows and columns for subplot grid
    n_neighbors = len(genus_neighbors)
    n_cols = 3 if n_neighbors > 2 else n_neighbors if n_neighbors > 0 else 1
    n_rows = int(np.ceil(n_neighbors / n_cols))

    fig, axes = plt.subplots(n_rows, n_cols, figsize=(6 * n_cols, 5 * n_rows), squeeze=False)
    axes = axes.flatten() # Flatten to 1D array for easy iteration

    # Get unique treatment groups for consistent coloring and ordering
    treatment_groups = sorted(analysis_df['Treatment'].unique())

    for i, neighbor in enumerate(sorted(list(genus_neighbors))):
        ax = axes[i]
```

```
            # Create violin plot
            sns.violinplot(x='Treatment', y=neighbor, data=analysis_df, ax=ax,
                           order=treatment_groups, palette='viridis', inner='quartile', cut=0)

            # Overlay strip plot to show individual data points
            sns.stripplot(x='Treatment', y=neighbor, data=analysis_df, ax=ax,
                          order=treatment_groups, color='black', alpha=0.6, jitter=0.15)

            # Perform Kruskal-Wallis test
            groups = [group[neighbor].dropna().values for name, group in
analysis_df.groupby('Treatment')]
            if len(groups) > 1 and all(len(g) > 0 for g in groups):
                try:
                    stat, p_val = kruskal(*groups)
                    ax.set_title(f'{neighbor}\n(Kruskal-Wallis p={p_val:.3f})', fontsize=12)
                except ValueError:
                    ax.set_title(f'{neighbor}\n(Not enough data for test)', fontsize=12)
            else:
                ax.set_title(neighbor, fontsize=12)

            ax.set_ylabel('Relative Abundance', fontsize=10)
            ax.set_xlabel('')
            ax.tick_params(axis='x', labelrotation=45, labelsize=10)
            ax.tick_params(axis='y', labelsize=10)


        # Hide unused subplots
        for j in range(i + 1, len(axes)):
            axes[j].set_visible(False)

        plt.suptitle("Effect of Treatment on Legionella's Neighbors", fontsize=22, fontweight='bold')
        plt.tight_layout(rect=[0, 0.03, 1, 0.95]) # Adjust layout to prevent title overlap

        output_filename = os.path.join(OUTPUT_PATH,
'treatment_effects_on_neighbors_violin.png')
        plt.savefig(output_filename, dpi=300, bbox_inches='tight')
        plt.close()

        print(f"Treatment effect plot saved to: {output_filename}")
else:
        print("\nSkipping treatment effect analysis as no genus-level neighbors were found.")

print("\n✅ Analysis Complete!")
```

```
# ==================== 1. Load Required R Packages ====================
library(phyloseq)
library(readxl)
library(readr)
library(dplyr)
library(tibble)

# ==================== 2. Set File Paths ====================
metadata_path <- "C:/Users/ASUS/Desktop/imeta/metadata.xlsx"
asv_table_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_table.csv"
taxonomy_path <- "C:/Users/ASUS/Desktop/imeta/result/taxonomy_table.xlsx"

# ==================== 3. Read Data Files ====================

# 3.1 Read sample metadata
cat("Reading sample metadata...\n")
metadata <- read_excel(metadata_path, sheet = 1)
print(paste("Metadata contains", nrow(metadata), "samples"))
print(paste("Metadata columns:", paste(colnames(metadata), collapse = ", ")))

# 3.2 Read ASV abundance table
cat("\nReading ASV abundance table...\n")
asv_table <- read_csv(asv_table_path)
print(paste("ASV table contains", nrow(asv_table), "ASVs and", ncol(asv_table)-1, "samples"))

# 3.3 Read taxonomic information
cat("\nReading taxonomic information...\n")
taxonomy <- read_excel(taxonomy_path, sheet = 1)
print(paste("Taxonomy table contains", nrow(taxonomy), "ASVs"))
print(paste("Taxonomic ranks:", paste(colnames(taxonomy)[-1], collapse = ", ")))

# ==================== 4. Data Preprocessing ====================

# 4.1 Process sample metadata
# Set sample_id as row names
sample_data_df <- metadata %>%
  column_to_rownames("sample_id")

# 4.2 Process ASV abundance table
# Set ASV_id as row names, convert to matrix
otu_table_df <- asv_table %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()
```

```
# Check and ensure consistent sample ordering
cat("\nChecking sample ID consistency...\n")
metadata_samples <- rownames(sample_data_df)
asv_samples <- colnames(otu_table_df)

if (length(setdiff(metadata_samples, asv_samples)) == 0 &&
      length(setdiff(asv_samples, metadata_samples)) == 0) {
  cat("✓ Sample IDs match completely!\n")
} else {
  cat("⚠ Sample IDs do not match, need to check!\n")
}

# 4.3 Process taxonomic information
# Set ASV_id as row names
tax_table_df <- taxonomy %>%
  column_to_rownames("ASV_id") %>%
  as.matrix()

# Check ASV ID consistency
asv_ids_otu <- rownames(otu_table_df)
asv_ids_tax <- rownames(tax_table_df)

if (length(setdiff(asv_ids_otu, asv_ids_tax)) == 0 &&
      length(setdiff(asv_ids_tax, asv_ids_otu)) == 0) {
  cat("✓ ASV IDs match completely!\n")
} else {
  cat("⚠ ASV IDs do not match, need to check!\n")
}

# ==================== 5. Create Phyloseq Object Components ====================

# 5.1 Create OTU table object
cat("\nCreating phyloseq components...\n")
OTU <- otu_table(otu_table_df, taxa_are_rows = TRUE)

# 5.2 Create sample data object
SAMP <- sample_data(sample_data_df)

# 5.3 Create taxonomy table object
TAX <- tax_table(tax_table_df)

# ==================== 6. Construct Phyloseq Object ====================
cat("Constructing phyloseq object...\n")
```

```
physeq <- phyloseq(OTU, SAMP, TAX)

# ==================== 7. Validation and Summary ====================
cat("\n=== Phyloseq object construction complete! ===\n")
print(physeq)

# Output basic statistical information
cat("\n=== Basic Statistical Information ===\n")
cat("Number of samples:", nsamples(physeq), "\n")
cat("Number of ASVs:", ntaxa(physeq), "\n")
cat("Taxonomic ranks:", paste(rank_names(physeq), collapse = ", "), "\n")
cat("Sample variables:", paste(sample_variables(physeq), collapse = ", "), "\n")

# Check data integrity
cat("\n=== Data Quality Check ===\n")
# Check for empty samples
empty_samples <- sample_sums(physeq) == 0
if (any(empty_samples)) {
  cat("⚠ Found", sum(empty_samples), "empty samples\n")
} else {
  cat("✓ All samples contain sequences\n")
}

# Check for empty ASVs
empty_taxa <- taxa_sums(physeq) == 0
if (any(empty_taxa)) {
  cat("⚠ Found", sum(empty_taxa), "empty ASVs\n")
} else {
  cat("✓ All ASVs contain sequences\n")
}

# Display sample sequencing depth statistics
cat("\n=== Sequencing Depth Statistics ===\n")
seq_depth <- sample_sums(physeq)
cat("Minimum sequencing depth:", min(seq_depth), "\n")
cat("Maximum sequencing depth:", max(seq_depth), "\n")
cat("Average sequencing depth:", round(mean(seq_depth), 0), "\n")
cat("Median sequencing depth:", round(median(seq_depth), 0), "\n")

# ==================== 8. Optional: Save Phyloseq Object ====================
# Uncomment the following line to save the phyloseq object
# saveRDS(physeq, "phyloseq_object.rds")
# cat("\nPhyloseq object saved as phyloseq_object.rds\n")
```

```
# ==================== 9. Optional: Basic Data Cleaning ====================
# Basic data filtering if needed
cat("\n=== Optional Data Cleaning Steps ===\n")
cat("Original data: ", ntaxa(physeq), "ASVs,", nsamples(physeq), "samples\n")

# Remove empty samples and empty ASVs if any
physeq_cleaned <- prune_samples(sample_sums(physeq) > 0, physeq)
physeq_cleaned <- prune_taxa(taxa_sums(physeq_cleaned) > 0, physeq_cleaned)

cat("Cleaned data: ", ntaxa(physeq_cleaned), "ASVs,", nsamples(physeq_cleaned), "samples\n")

# Assign cleaned object to main variable
physeq <- physeq_cleaned

cat("\n🎉 Phyloseq object construction complete! You can start subsequent analyses.\n")
cat("Object name: physeq\n")
cat("Use print(physeq) to view object information\n")

# ==================== 10. Add Phylogenetic Tree to Phyloseq Object ====================
library(ape)

# Set tree file paths
tree_path <- "C:/Users/ASUS/Desktop/imeta/result/sequences.aligned.fasta.treefile"
hash_mapping_path <- "C:/Users/ASUS/Desktop/imeta/result/asv_hash_mapping.csv"

# 10.1 Read and process hash mapping
cat("\nReading ASV hash mapping...\n")
hash_mapping <- read_csv(hash_mapping_path, col_names = c("ASV_id", "Hash"))
hash_mapping$Hash <- trimws(hash_mapping$Hash)    # Clean whitespaces
hash_vec <- setNames(hash_mapping$ASV_id, hash_mapping$Hash)

# 10.2 Read phylogenetic tree
cat("Reading phylogenetic tree...\n")
tree <- read.tree(tree_path)

# 10.3 Check tree-hash correspondence
tree_hashes <- tree$tip.label
mapped_asvs <- hash_vec[tree_hashes]
unmapped_count <- sum(is.na(mapped_asvs))

if (unmapped_count > 0) {
  cat("⚠ Warning:", unmapped_count, "tree tips not found in hash mapping\n")
  # Remove unmapped tips
  tree <- drop.tip(tree, tree_hashes[is.na(mapped_asvs)])
```

```
    mapped_asvs <- na.omit(mapped_asvs)
}

# 10.4 Update tip labels with ASV IDs
tree$tip.label <- as.character(mapped_asvs[tree$tip.label])

# 10.5 Align tree and phyloseq object
# Get overlapping ASVs
common_asvs <- intersect(tree$tip.label, taxa_names(physeq))
physeq_sub <- prune_taxa(common_asvs, physeq)
tree <- keep.tip(tree, common_asvs)

cat("→ Retained", length(common_asvs), "ASVs common to tree and phyloseq\n")

# 10.6 Add tree to phyloseq object
phy_tree(physeq_sub) <- tree

# 10.7 Validate integration
cat("\n=== Tree Integration Validation ===\n")
cat("Tree contains", Ntip(tree), "tips matching ASVs\n")

if (all(taxa_names(physeq_sub) %in% tree$tip.label)) {
    cat("✓ All phyloseq taxa present in tree\n")
} else {
    cat("⚠ Missing taxa: Not all ASVs are in the tree\n")
}

# 10.8 Update phyloseq object
physeq <- phyloseq(otu_table(physeq_sub),
                   sample_data(physeq_sub),
                   tax_table(physeq_sub),
                   phy_tree(physeq_sub))

cat("\n=== Updated Phyloseq Object Summary ===\n")
print(physeq)

# 10.9 Optional: Save updated object
# saveRDS(physeq, "phyloseq_with_tree.rds")
# cat("Saved phyloseq object with tree as 'phyloseq_with_tree.rds'\n")

cat("\n✅ Phylogenetic tree successfully integrated!\n")

# ==================== 1. Load Required R Packages ====================
# Ensure you have these packages installed: install.packages(c("phyloseq", "dplyr", "ggplot2",
```

```
"readr", "tibble", "Hmisc"))
library(phyloseq)
library(dplyr)
library(ggplot2)
library(readr)
library(tibble)
library(Hmisc) # Added for correlation with significance testing

# Assuming your 'physeq' object already exists in the R environment.
# If not, please run your previous script to create the 'physeq' object first.

# ==================== 2. Set File Paths ====================
absolute_abundance_path <-
"C:/Users/ASUS/Desktop/imeta/result/absolute_abundance_asv_table.csv"
output_plot_dir <- "C:/Users/ASUS/Desktop/imeta/plots/network"
# Define a new file name for the updated plot
output_plot_path <- file.path(output_plot_dir, "legionella_correlation_lollipop_with_sig.png")

# Create the output directory if it doesn't exist
if (!dir.exists(output_plot_dir)) {
  dir.create(output_plot_dir, recursive = TRUE)
}

# ==================== 3. Filter Samples and Load Absolute Abundance Data
====================

# 3.1 Filter for samples where Treatment is "Untreated"
cat("Filtering for 'Untreated' samples...\n")
physeq_untreated <- subset_samples(physeq, Treatment == "Untreated")
cat("Number of samples remaining after filtering:", nsamples(physeq_untreated), "\n")

# 3.2 Read the absolute abundance table
cat("Reading absolute abundance table...\n")
abs_abundance_table <- read_csv(absolute_abundance_path)

# 3.3 Process the absolute abundance table into the required phyloseq format
#       (rownames = ASV_id, colnames = sample_id, and as a matrix)
otu_abs <- abs_abundance_table %>%
  tibble::column_to_rownames("ASV_id") %>%
  # Ensure the column order matches the filtered phyloseq object
  select(all_of(sample_names(physeq_untreated))) %>%
  as.matrix()

# 3.4 Replace the OTU table in the phyloseq object with the absolute abundances
```

```
cat("Updating phyloseq object with absolute abundances...\n")
otu_table(physeq_untreated) <- otu_table(otu_abs, taxa_are_rows = TRUE)

# ==================== 4. Aggregate to Genus Level and Filter ====================

# 4.1 Aggregate ASVs to the Genus level
cat("Aggregating taxa to 'Genus' level...\n")
physeq_genus <- tax_glom(physeq_untreated, taxrank = "Genus", NArm = TRUE)

# 4.2 Filter out any genera containing "uncultured" in their name
cat("Filtering out 'uncultured' genera...\n")
tax_df <- as.data.frame(tax_table(physeq_genus))
# grepl returns a logical value, ignore.case = TRUE makes the search case-insensitive
keep_taxa <- rownames(tax_df)[!grepl("uncultured", tax_df$Genus, ignore.case = TRUE)]
physeq_genus_filtered <- prune_taxa(keep_taxa, physeq_genus)
cat("Number of genera remaining after filtering:", ntaxa(physeq_genus_filtered), "\n")

# ==================== 5. Select Core Genera (Top 30 + Legionella) ====================

# 5.1 Calculate total abundance for each genus
genus_abundance <- data.frame(
  Genus = tax_table(physeq_genus_filtered)[, "Genus"],
  TotalAbundance = taxa_sums(physeq_genus_filtered)
)

# 5.2 Identify the top 30 most abundant genera
top30_genera <- genus_abundance %>%
  arrange(desc(TotalAbundance)) %>%
  slice(1:30) %>%
  pull(Genus) # pull() extracts a single column

cat("Top 30 most abundant genera:\n")
print(top30_genera)

# 5.3 Ensure Legionella is included in the list for analysis
target_genus <- "Legionella"
if (!target_genus %in% top30_genera) {
  cat(paste("'", target_genus, "' was not in the Top 30 and will be added manually.\n", sep=""))
  # union() combines two vectors and removes duplicates
  selected_genera <- union(top30_genera, target_genus)
} else {
  cat(paste("'", target_genus, "' is already in the Top 30.\n", sep=""))
  selected_genera <- top30_genera
}
```

```
cat("Total number of genera for analysis:", length(selected_genera), "\n")

# ==================== 6. Correlation Analysis with Significance Testing ====================

# 6.1 Filter the phyloseq object to keep only the selected genera
final_taxa_indices <- tax_table(physeq_genus_filtered)[, "Genus"] %in% selected_genera
physeq_final <- prune_taxa(final_taxa_indices, physeq_genus_filtered)

# 6.2 Prepare the data frame for correlation analysis (samples as rows, genera as columns)
cor_data <- as.data.frame(t(otu_table(physeq_final)))
colnames(cor_data) <- tax_table(physeq_final)[, "Genus"]

# 6.3 Calculate Spearman correlation matrix and p-values using Hmisc::rcorr
cat("Calculating Spearman correlations and p-values...\n")
# rcorr requires a matrix input
corr_results <- rcorr(as.matrix(cor_data), type = "spearman")
cor_matrix <- corr_results$r
p_matrix <- corr_results$P

# 6.4 Extract the correlation results for Legionella
legionella_correlations <- cor_matrix[, target_genus]
legionella_p_values <- p_matrix[, target_genus]

# 6.5 Create a data frame for plotting, now including p-values and significance markers
plot_df <- data.frame(
  Genus = names(legionella_correlations),
  Correlation = legionella_correlations,
  PValue = legionella_p_values
) %>%
  filter(Genus != target_genus) %>% # Remove self-correlation
  mutate(
    # Create significance labels based on p-value
    Significance = case_when(
      PValue < 0.001 ~ "***",
      PValue < 0.01  ~ "**",
      PValue < 0.05  ~ "*",
      TRUE           ~ "" # No label if not significant
    )
  )

# ==================== 7. Visualization: Lollipop Chart with Significance ====================
```

```
cat("Generating lollipop chart and saving to file...\n")

# Create the lollipop chart using ggplot2
p <- ggplot(plot_df, aes(x = Correlation, y = reorder(Genus, Correlation))) +
  # Create the "stick" of the lollipop
  geom_segment(aes(xend = 0, yend = Genus), color = "grey50") +
  # Create the "candy" part of the lollipop
  geom_point(aes(color = Correlation), size = 4) +
  # Add the significance labels next to the points
  geom_text(
    aes(label = Significance),
    color = "black",
    hjust = -0.7,    # Adjust horizontal position of the asterisk
    vjust = 0.5        # Adjust vertical position
  ) +
  # Use a color gradient from blue (negative) to red (positive)
  scale_color_gradient2(low = "blue", mid = "white", high = "red", midpoint = 0) +
  # Add a vertical dashed line at x=0 for reference
  geom_vline(xintercept = 0, linetype = "dashed", color = "gray") +
  # Set labels and titles in English
  labs(
    title = paste("Correlation Analysis with Genus", target_genus),
    subtitle = "Based on absolute abundance in 'Untreated' samples",
    caption = "Significance levels: *** p < 0.001; ** p < 0.01; * p < 0.05",
    x = "Spearman Correlation Coefficient (ρ)",
    y = "Genus"
  ) +
  # Use a minimal, clean theme
  theme_minimal() +
  # Further theme adjustments for aesthetics
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5),
    plot.caption = element_text(hjust = 0, face = "italic"),
    panel.grid.major.y = element_blank(), # Remove horizontal grid lines
    axis.text.y = element_text(face = "italic") # Italicize genus names
  )

# Print the plot to the RStudio Plots pane
print(p)

# Save the plot to the specified file
ggsave(
  output_plot_path,
```

```
  plot = p,
  width = 10,
  height = 8,
  dpi = 300,
  bg = "white" # Set a white background
)

cat(paste("\n 🎉 Analysis complete! Plot saved to:\n", output_plot_path, "\n"))

# ==================== 1. Load Required R Packages ====================
# Ensure you have these packages installed: install.packages(c("phyloseq", "dplyr", "ggplot2", "gghalves", "ggpubr"))
library(phyloseq)
library(dplyr)
library(ggplot2)
library(gghalves) # For creating half-violin/half-point plots
library(ggpubr)      # For adding statistical comparisons

# ==================== 2. Prepare Data for Plotting ====================
# NOTE: We start from the original, complete 'physeq' object to include all treatment groups.
# We will use the absolute abundance data for consistency.

# 2.1 Define the target genus
target_genus_for_plot <- "Aquabacterium"

# 2.2 Re-create the absolute abundance phyloseq object if not already present
# This ensures we are working with the full, unfiltered dataset
cat("Reading absolute abundance table for all samples...\n")
abs_abundance_table_full <- read_csv("C:/Users/ASUS/Desktop/imeta/result/absolute_abundance_asv_table.csv")

# Process the table to match the full phyloseq object
otu_abs_full <- abs_abundance_table_full %>%
  tibble::column_to_rownames("ASV_id") %>%
  # Select only the samples present in the original phyloseq object
  select(any_of(sample_names(physeq))) %>%
  as.matrix()

# Create a fresh copy of the phyloseq object and replace its OTU table
physeq_abs_full <- physeq
otu_table(physeq_abs_full) <- otu_table(otu_abs_full, taxa_are_rows = TRUE)

# 2.3 Aggregate to Genus level
cat("Aggregating to Genus level for all samples...\n")
```

```
physeq_genus_full <- tax_glom(physeq_abs_full, taxrank = "Genus", NArm = TRUE)

# 2.4 Melt the phyloseq object into a long data frame for ggplot
cat("Melting phyloseq object to create a plotting data frame...\n")
ps_melted <- psmelt(physeq_genus_full)

# 2.5 Filter the melted data to get only the abundance of Aquabacterium
plot_data <- ps_melted %>%
  filter(Genus == target_genus_for_plot)

# Check if data was found
if(nrow(plot_data) == 0) {
  stop(paste("Error: Genus '", target_genus_for_plot, "' not found in the dataset. Please check the spelling.", sep=""))
}

cat("Data for '", target_genus_for_plot, "' prepared successfully.\n")

# To avoid overly large numbers on the y-axis, we can use a log1p transformation.
# log1p(x) computes log(1+x), which handles zeros gracefully.
plot_data <- plot_data %>%
  mutate(Abundance_log1p = log1p(Abundance))

# ==================== 3. Create and Save the Rain Cloud Plot ====================
cat("Generating Rain Cloud Plot with Kruskal-Wallis test result only...\n")

p_raincloud_simple <- ggplot(plot_data, aes(x = Treatment, y = Abundance_log1p, fill = Treatment)) +

  # --- Add the "Cloud" (Density Plot) ---
  gghalves::geom_half_violin(
    side = "r",
    adjust = 0.5,
    trim = FALSE,
    alpha = 0.6,
    scale = "width"
  ) +

  # --- Add the Boxplot ---
  geom_boxplot(width = 0.15, outlier.shape = NA, alpha = 0.6) +

  # --- Add the "Rain" (Individual Points) ---
  gghalves::geom_half_point(
    side = "l",
```

```
    shape = 21, # Circle with border
    alpha = 0.5,
    stroke = 0.1,
    size = 2,
    transformation = position_jitter(width = 0.05, height = 0)
  ) +

  # --- Add Statistical Comparison (Kruskal-Wallis test ONLY) ---
  stat_compare_means(
    method = "kruskal.test", # Add the global Kruskal-Wallis p-value
    label.y.npc = 0.99, # Position the global p-value at 95% of the plot height
    label.x.npc = 0.1
  ) +

  # --- Aesthetics and Labels ---
  scale_fill_viridis_d(option = "D") + # Use a colorblind-friendly palette
  labs(
    title = paste("Abundance of", target_genus_for_plot),
    subtitle = "Distribution across different treatment groups",
    y = "Abundance (log1p transformed)",
    x = "Treatment Group"
  ) +
  theme_classic() +
  theme(
    legend.position = "none", # Hide the legend as colors are mapped to x-axis
    plot.title = element_text(hjust = 0.5, face = "bold"),
    plot.subtitle = element_text(hjust = 0.5, face = "italic"),
    axis.text.x = element_text(angle = 45, hjust = 1) # Rotate x-axis labels if they overlap
  )

# Print the plot
print(p_raincloud_simple)

# --- Save the plot ---
output_plot_path_raincloud_simple <- file.path(output_plot_dir,
"aquabacterium_abundance_kruskal_only.png")
ggsave(
  output_plot_path_raincloud_simple,
  plot = p_raincloud_simple,
  width = 8,
  height = 7,
  dpi = 300,
  bg = "white"
)
```

cat(paste("\n🎉 Analysis complete! Simplified rain cloud plot saved to:\n", output_plot_path_raincloud_simple, "\n"))