Assessment of Cluster Analysis of Elastic Light Scatter Profiles for the Identification of Foodborne Bacteria

Python Codes:

1- Data amalgamation into Excel file

```
import os

import pandas as pd

# Define the path to the main folder

main_folder_path = r'C:\Users\sanab\OneDrive\Desktop\DATA-09-Copy'

# Function to get all CSV files from a subfolder

def get_csv_files(subfolder_path):

    csv_files = []

    for root, dirs, files in os.walk(subfolder_path):

        for file in files:

            if file.endswith('.csv'):

                csv_files.append(os.path.join(root, file))

    return csv_files


# Function to extract colony number from file path or filename

def extract_colony_id(file_path):

    file_name = os.path.basename(file_path)

    parts = file_name.split('colony')

    if len(parts) > 1:

        return parts[-1].strip().split('.')[0]

    else:

        return file_name.split('.')[0]


# Define lists for storing data

experimental_data = []

pseudo_zernike_data = []

zernike_data = []
```

```
# Iterate over the main subfolders
for folder_name in os.listdir(main_folder_path):
    folder_path = os.path.join(main_folder_path, folder_name)
    if os.path.isdir(folder_path):
        taxon = folder_name  # Extract the Taxon from the folder name

        # Iterate over each subfolder like RM 4463, RM 4826, etc.
        for subfolder_name in os.listdir(folder_path):
            subfolder_path = os.path.join(folder_path, subfolder_name)
            if os.path.isdir(subfolder_path):
                strain_id = subfolder_name.split('(')[0].strip()  # Extract the Strain ID

                # Get paths to Experimental, Pseudozernike, Zernike folders
                experimental_folder = os.path.join(subfolder_path, 'Experimental')
                pseudo_zernike_folder = os.path.join(subfolder_path, 'Pseudozernike')
                zernike_folder = os.path.join(subfolder_path, 'Zernike')

                # Get all CSV files for Experimental, Pseudozernike, Zernike data
                experimental_files = get_csv_files(experimental_folder)
                pseudo_zernike_files = get_csv_files(pseudo_zernike_folder)
                zernike_files = get_csv_files(zernike_folder)

                # Process each experimental file
                for file in experimental_files:
                    temp_df = pd.read_csv(file)

                    # Extract relevant columns: Column N (moment indices) and Y (moment values)
                    moments = temp_df['N'].astype(str).apply(lambda x: f"Experimental {x}").tolist()  #
Rename moments
                    moment_values = temp_df['Y'].tolist()
                    # Create a single row DataFrame for each colony
                    colony_df = pd.DataFrame([moment_values], columns=moments)
                    colony_df['Taxon'] = taxon
                    colony_df['Strain ID'] = strain_id
                    colony_df['Colony ID'] = extract_colony_id(file)  # Extract colony info

                    # Append the DataFrame to experimental_data
                    experimental_data.append(colony_df)

                # Process each pseudo-Zernike file
                for file in pseudo_zernike_files:
                    temp_df = pd.read_csv(file)

                    # Extract relevant columns: Column N (moment indices) and Y (moment values)
                    moments = temp_df['N'].astype(str).apply(lambda x: f"PseudoZernike {x}").tolist()
                    moment_values = temp_df['Y'].tolist()

                    # Create a single row DataFrame for each colony
                    colony_df = pd.DataFrame([moment_values], columns=moments)
                    colony_df['Taxon'] = taxon
                    colony_df['Strain ID'] = strain_id
                    colony_df['Colony ID'] = extract_colony_id(file)

                    # Append the DataFrame to pseudo_zernike_data
                    pseudo_zernike_data.append(colony_df)

                # Process each Zernike file
                for file in zernike_files:
                    temp_df = pd.read_csv(file)

                    # Extract relevant columns: Column N (moment indices) and Y (moment values)
```

```
            moments = temp_df['N'].astype(str).apply(lambda x: f"Zernike {x}").tolist()
            moment_values = temp_df['Y'].tolist()

            # Create a single row DataFrame for each colony
            colony_df = pd.DataFrame([moment_values], columns=moments)
            colony_df['Taxon'] = taxon
            colony_df['Strain ID'] = strain_id
            colony_df['Colony ID'] = extract_colony_id(file)

            # Append the DataFrame to zernike_data
            zernike_data.append(colony_df)

# Concatenate all the data for each category
experimental_df = pd.concat(experimental_data, ignore_index=True)
pseudo_zernike_df = pd.concat(pseudo_zernike_data, ignore_index=True)
zernike_df = pd.concat(zernike_data, ignore_index=True)

# Define the output path for the Excel file
output_excel_file_path = r'C:\Users\sanab\OneDrive\Desktop\Analysis for
Sana\Merged_Data_Corrected.xlsx'
# Write the data into separate sheets in an Excel file
with pd.ExcelWriter(output_excel_file_path, engine='xlsxwriter') as writer:
    # Write each DataFrame to a different sheet
    experimental_df.to_excel(writer, sheet_name='Experimental', index=False)
    zernike_df.to_excel(writer, sheet_name='Zernike Moments', index=False)
    pseudo_zernike_df.to_excel(writer, sheet_name='Pseudo-Zernike Moments', index=False)

print(f'Reorganized data has been saved to {output_excel_file_path}')
```

2 – Code For Generation of Similarity Matrices for Each Feature Type (Zernike, PseudoZernike, Patsekin elements)

```
#similarity matrices formation

import pandas as pd

import numpy as np

import os

# File path for the data

absolute_file_path = = r'C:\Users\sanab\OneDrive\Desktop\Analysis for
Sana\Merged_Data_Corrected.xlsx'

# Load the Excel file

excel_data = pd.ExcelFile(absolute_file_path)

# Initialize a dictionary to store similarity matrices for each sheet

similarity_matrices = {}

# Iterate over the sheets

for sheet_name in excel_data.sheet_names:

  print(f"Processing sheet: {sheet_name}")


 # Read the sheet into a DataFrame

  data = excel_data.parse(sheet_name)


 # Calculate the Pearson correlation matrix between rows (each colony with itself and every
other colony)

  row_correlation_matrix = np.corrcoef(numerical_data)


  # Store the similarity matrix with labels

  similarity_matrices[sheet_name] = pd.DataFrame(

    row_correlation_matrix, index=labels, columns=labels

  )

 # Save the similarity matrix to a new Excel file using openpyxl engine

  output_file_path = os.path.join(

    os.path.dirname(absolute_file_path), f"{sheet_name}_Pearson_Similarity_Matrix.xlsx"
  )
  similarity_matrices[sheet_name].to_excel(output_file_path, engine='openpyxl')
  print(f"Saved similarity matrix for sheet: {sheet_name} to {output_file_path}")

print("Processing complete!")
```

3-Code for averaging the similarity matrices, formation of distant matrix and generating the dendogram at 0.25 distant threshold

```
import os
import numpy as np
import pandas as pd
import scipy.cluster.hierarchy as sch
from scipy.spatial.distance import squareform
from scipy.cluster.hierarchy import fcluster
import matplotlib.pyplot as plt

# Function to load the similarity matrix from a CSV file with strain labels
def load_similarity_matrix(file_path):
    matrix = pd.read_csv(file_path, index_col=0)
    print(f"Loaded matrix from {file_path}, shape: {matrix.shape}")
    return matrix

# Function to find common rows and columns between all matrices
def find_common_indices(*matrices):
    common_indices = set(matrices[0].index)
    for matrix in matrices[1:]:
        common_indices = common_indices.intersection(set(matrix.index))
    common_indices = sorted(list(common_indices))
    print(f"Common indices found: {len(common_indices)}")
    return common_indices
```

```
# Function to trim matrices to only include common rows and columns
def trim_matrices(common_indices, *matrices):
    trimmed_matrices = []
    for matrix in matrices:
        trimmed_matrix = matrix.loc[common_indices, common_indices]
        print(f"Trimmed matrix shape: {trimmed_matrix.shape}")
        trimmed_matrices.append(trimmed_matrix)
    return trimmed_matrices

# Function to combine multiple similarity matrices by averaging them
def combine_similarity_matrices(*matrices):
    matrix_arrays = [matrix.values for matrix in matrices]
    combined = np.mean(matrix_arrays, axis=0)
    combined_symmetric = (combined + combined.T) / 2
    return combined_symmetric

# Function to convert similarity matrix to distance matrix (1 - similarity)
def similarity_to_distance_matrix(similarity_matrix):
    distance_matrix = 1 - similarity_matrix
    np.fill_diagonal(distance_matrix, 0)
    return distance_matrix
pseudozernike_path = 'C:/Users/sanab/Desktop/similarity matrix for pseudozernike.csv'
experimental_path   = 'C:/Users/sanab/Desktop/ similarity matrix for experimental.csv'

# 1. Load matrices
pseudozernike_matrix = load_similarity_matrix(pseudozernike_path)
experimental_matrix  = load_similarity_matrix(experimental_path)

# 3. Find common indices and trim the matrices accordingly
common_indices = find_common_indices(pseudozernike_matrix, experimental_matrix)
```

```
pseudozernike_matrix, experimental_matrix = trim_matrices(common_indices,
pseudozernike_matrix, experimental_matrix)


# Combine the similarity matrices and convert to a distance matrix

combined_similarity = combine_similarity_matrices(pseudozernike_matrix,
experimental_matrix)

distance_matrix = similarity_to_distance_matrix(combined_similarity)


# 5. Get the original labels (full strain names) from one of the matrices

labels = pseudozernike_matrix.index.tolist()


# 6. Create an interactive dendrogram using Plotly.

fig = ff.create_dendrogram(

    distance_matrix,

    orientation='right',

    labels=labels,

    linkagefun=lambda x: sch.linkage(squareform(distance_matrix), method='average')

)

fig.update_layout(

    title="Full Main Dendrogram with Original Labels",

    height=20000,   # Increase height (e.g. 12000 pixels) to allow vertical scrolling.

    width=1300,     # Adjust width as needed.

    font=dict(size=12)  # Font size set to 12 pt.

)


# 7. Save the dendrogram as an interactive HTML file.

output_html = "C:/Users/sanab/Desktop/full_main_dendrogram.html"

fig.write_html(output_html)

print(f"Full main dendrogram saved to: {output_html}")
```

4- Code for clustering Arcobacters Only

```
import os

import numpy as np

import pandas as pd

import scipy.cluster.hierarchy as sch

from scipy.spatial.distance import squareform

from scipy.cluster.hierarchy import fcluster

import matplotlib.pyplot as plt


# Function to load the similarity matrix from a CSV file with strain labels

def load_similarity_matrix(file_path):

    matrix = pd.read_csv(file_path, index_col=0)

    print(f"Loaded matrix from {file_path}, shape: {matrix.shape}")

    return matrix


# Function to filter Arcobacter strains based on the specified row ranges

def filter_arcobacter_by_row_range(matrix, start_row, end_row):

    arcobacter_matrix = matrix.iloc[start_row-1:end_row, start_row-1:end_row]

    arcobacter_labels = matrix.index[start_row-1:end_row]

    print(f"Filtered matrix for Arcobacter strains, new shape: {arcobacter_matrix.shape}")

    return arcobacter_matrix, arcobacter_labels


# Function to find and remove rows/columns that are not common between two matrices

def keep_common_rows_and_columns(matrix1, matrix2):

    common_indices = matrix1.index.intersection(matrix2.index)

    matrix1_common = matrix1.loc[common_indices, common_indices]

    matrix2_common = matrix2.loc[common_indices, common_indices]

    print(f"After removing uncommon rows/columns: Matrix 1 shape: {matrix1_common.shape},
Matrix 2 shape: {matrix2_common.shape}")

    return matrix1_common, matrix2_common
```

```
# Function to combine multiple similarity matrices by averaging them and enforce symmetry
def combine_similarity_matrices(*matrices):
    matrix_arrays = [matrix.values for matrix in matrices]
    combined = np.mean(matrix_arrays, axis=0)
    combined_symmetric = (combined + combined.T) / 2
    return combined_symmetric

# Function to convert similarity matrix to distance matrix (1 - similarity)
def similarity_to_distance_matrix(similarity_matrix):
    if similarity_matrix.shape[0] == 0:
        raise ValueError("The filtered similarity matrix is empty. No clustering can be performed.")

    distance_matrix = 1 - similarity_matrix
    np.fill_diagonal(distance_matrix, 0)
    return distance_matrix

# Function to generate dendrograms, calculate cophenetic coefficient, and save results
def generate_dendrogram_and_results(distance_matrix, labels, output_path, threshold):
    if not os.path.exists(output_path):
        os.makedirs(output_path)

    linkage_matrix = sch.linkage(squareform(distance_matrix), method='average')
    cluster_labels = fcluster(linkage_matrix, t=threshold, criterion='distance')


    plt.figure(figsize=(22, 10))
    dendrogram = sch.dendrogram(linkage_matrix, labels=labels, leaf_rotation=90,
leaf_font_size=8, color_threshold=threshold)


    plt.title(f'Dendrogram at threshold {threshold}')
    plt.tight_layout()
    main_dendrogram_path = os.path.join(output_path,
f'Arcobacters_only_dendrogram_threshold_{threshold}.png')
```

```
    plt.savefig(main_dendrogram_path, bbox_inches='tight')
    plt.show()
    print(f"Main dendrogram saved at {main_dendrogram_path}")

    return cluster_labels, dendrogram, linkage_matrix

# Function to generate and display sub-dendrograms with colors preserved from the main
dendrogram
def generate_sub_dendrograms_with_colors_preserved(linkage_matrix, cluster_labels,
dendrogram, labels, distance_matrix):
    original_colors = dendrogram['leaves_color_list']
    unique_clusters = np.unique(cluster_labels)[::-1]

    for i, cluster in enumerate(unique_clusters):
        cluster_indices = np.where(cluster_labels == cluster)[0]
        if len(cluster_indices) > 1:
            cluster_distance_matrix = distance_matrix[np.ix_(cluster_indices, cluster_indices)]
            cluster_linkage_matrix = sch.linkage(squareform(cluster_distance_matrix),
method='average')

            cluster_labels_subset = [labels[i] for i in cluster_indices]
            cluster_colors = [original_colors[dendrogram['leaves'].index(i)] for i in cluster_indices]

            plt.figure(figsize=(18, 10))
            sch.dendrogram(
                cluster_linkage_matrix,
                labels=cluster_labels_subset,
                leaf_rotation=90,
                leaf_font_size=6,
                link_color_func=lambda x: cluster_colors[x % len(cluster_colors)]
            )
            plt.title(f'Sub-Dendrogram for Cluster {cluster}')
        plt.show()

        print(f"Displayed sub-dendrogram for cluster {cluster}.")


# Updated file paths

pseudozernike_path = 'C:/Users/sanab/Desktop/similarity matrix for pseudozernike.csv'

experimental_path = 'C:/Users/sanab/Desktop/ similarity matrix for experimental.csv'


# Load similarity matrices

pseudozernike_matrix = load_similarity_matrix(pseudozernike_path)

experimental_matrix = load_similarity_matrix(experimental_path)


# Print initial shapes before any filtering

print(f"Initial pseudo-Zernike shape: {pseudozernike_matrix.shape}")

print(f"Initial experimental shape: {experimental_matrix.shape}")


# Step 1: Filter Arcobacter strains based on the provided row ranges

# Pseudozernike: Rows 1 to 527, Experimental: Rows 1 to 564 (before excluding RM 5557)

pseudozernike_matrix, pseudozernike_labels =
filter_arcobacter_by_row_range(pseudozernike_matrix, 1, 527)

experimental_matrix, experimental_labels =
filter_arcobacter_by_row_range(experimental_matrix, 1, 564)


# Step 3: Remove any rows/columns that are not common between the two matrices

pseudozernike_matrix, experimental_matrix =
keep_common_rows_and_columns(pseudozernike_matrix, experimental_matrix)


# Combine similarity matrices and convert to distance matrix

combined_similarity = combine_similarity_matrices(pseudozernike_matrix,
experimental_matrix)

distance_matrix = similarity_to_distance_matrix(combined_similarity)


# Set the output path to the desktop
```

```
output_path = 'C:/Users/sanab/Desktop/aberrant colonies removed'


# Step 4: Generate the main dendrogram

labels = pseudozernike_labels  # Use the Arcobacter labels for the dendrogram

threshold = 0.25

cluster_labels, dendrogram, linkage_matrix =
generate_dendrogram_and_results(distance_matrix, labels, output_path, threshold)


# Step 5: Display sub-dendrograms for each cluster, preserving colors

generate_sub_dendrograms_with_colors_preserved(linkage_matrix, cluster_labels,
dendrogram, labels, distance_matrix)
```

Click on following link to view clustering for *Arcobacter* species only:

https://sana-bari.github.io/clustering-analysis/Arcobacter_only_dendrogram_normalized.html

Click on following link to view clustering for *Yersinia* species only:

https://sana-bari.github.io/clustering-analysis/Yersinia_only_dendrogram_normalized.html

Links for Different Combinations of Features Used

Combined features_dendrogram	https://sana-bari.github.io/clustering-analysis/combined_dendrogram%20(3).html
Pseudozernike_dendrogram	https://sana-bari.github.io/clustering-analysis/pseudozernike_dendrogram.html
Zernike_dendrogram	https://sana-bari.github.io/clustering-analysis/zernike_dendrogram.html
Zernike_Experimental_dendrogram	https://sana-bari.github.io/clustering-analysis/zernike_experimental_dendrogram.html
Zernike_Pseudozernike_dendrogram	https://sana-bari.github.io/clustering-analysis/zernike_pseudozernike_dendrogram.html

Patsekin elements	https://sana-bari.github.io/clustering-analysis/experimental_dendrogram.html