import pandas as pd

import numpy as np

import ast

def process\_finger\_data(data, frequency, finger\_index, data\_index):

"""

Process data for a specific finger at a given frequency and index in the CSV file.

"""

finger\_key = f'resp\_{frequency}Hz\_{finger\_index}'

if f'{finger\_key}.rt' in data.columns and not pd.isna(data.loc[data\_index, f'{finger\_key}.rt']):

response\_started = data.loc[data\_index, f'{finger\_key}.started']

pressed\_raw = ast.literal\_eval(data.loc[data\_index, f'{finger\_key}.rt'])

pressed\_raw = [float(x) for x in pressed\_raw if x is not None]

pressed = np.array(pressed\_raw) - response\_started

duration\_raw = ast.literal\_eval(data.loc[data\_index, f'{finger\_key}.duration'])

# Replace None with 30.0 only if the last value in duration\_raw is None

if duration\_raw and duration\_raw[-1] is None:

duration\_raw[-1] = 30.0

# Replace any remaining None values with 30.0

duration = [float(x) if x is not None else 30.0 for x in duration\_raw]

# Ensure `pressed` and `duration` have the same length

if len(pressed) != len(duration):

print(f"Warning: Length mismatch for frequency {frequency}Hz, finger {finger\_index} at index {data\_index}")

print(f"Pressed Raw: {pressed\_raw}, Duration Raw: {duration\_raw}")

return None, None

released = np.add(pressed, duration)

return pressed, released

else:

print(f"Finger {finger\_index}\_{frequency}Hz was not pressed; skipping this part of the code.")

return None, None

def calculate\_time\_intervals(pressed, released):

"""

Calculate time intervals based on pressed and released times.

"""

if pressed is not None and pressed.size > 0:

intervals = [(0, pressed[0])]

if released is not None and released.size > 0:

for i in range(len(released) - 1):

intervals.append((released[i], pressed[i + 1]))

intervals.append((released[-1], 30))

else:

print("Released data is empty or undefined; skipping some time interval calculations.")

return intervals

else:

print("Pressed data is empty or undefined; skipping time interval calculation.")

return []

def process\_frequency\_block(data, frequency, data\_index):

"""

Process the block of data for a given frequency and CSV index.

"""

formatted\_frequency = str(frequency).replace('.', '\_') # Format frequency for column names

all\_intervals = {}

for finger\_index in range(1, 6):

if finger\_index == 2: # Skip Finger 2

continue

pressed, released = process\_finger\_data(data, formatted\_frequency, finger\_index, data\_index)

intervals = calculate\_time\_intervals(pressed, released)

all\_intervals[f'Finger{finger\_index}\_intervals'] = intervals

play\_times\_raw = data.loc[data\_index, f'sound{formatted\_frequency}Hz\_play\_times']

play\_times\_original = ast.literal\_eval(play\_times\_raw)

play\_times = play\_times\_original[:-1]

return all\_intervals, play\_times

# Load the data from the CSV file

file\_path = r'C:\Users\12035\Desktop\1. phd\24 motor task\data\0200.csv'

data = pd.read\_csv(file\_path)

# Define frequencies and indices

frequency\_index\_mapping = {

1: 3,

2: 7,

2.5: 11,

3: 15,

3.5: 19,

4: 23,

4.5: 27,

5: 31,

5.5: 35,

6: 39

}

# Process each frequency block

for frequency, data\_index in frequency\_index\_mapping.items():

intervals, sound\_play\_times = process\_frequency\_block(data, frequency, data\_index)

print(f"Intervals for {frequency}Hz:", intervals)

print(f"Sound play times for {frequency}Hz:", sound\_play\_times)

def calculate\_reaction\_time\_and\_accuracy(finger\_pressed, sound\_play\_times, frequency):

"""

Calculate reaction time and accuracy for a specific finger and frequency.

"""

# Dictionary to store responses assigned to each sound, initialize with empty lists

assigned\_responses = {sound\_time: [] for sound\_time in sound\_play\_times}

current\_sound\_index = 0

# Assign each response to the appropriate sound

for rt in finger\_pressed:

# Move to the next sound if the response time is beyond the current sound's interval

while current\_sound\_index < len(sound\_play\_times) - 1 and rt >= sound\_play\_times[current\_sound\_index + 1]:

current\_sound\_index += 1

# Assign the response to the current sound

assigned\_responses[sound\_play\_times[current\_sound\_index]].append(rt)

# Check for sounds with no responses, adjust assignments if the next sound has multiple responses

sound\_times = list(assigned\_responses.keys()) # List of sound times for ordered access

for i in range(1, len(sound\_times)):

prev\_sound = sound\_times[i - 1]

current\_sound = sound\_times[i]

# Condition: If previous sound has no response and current sound has multiple responses

if not assigned\_responses[prev\_sound] and len(assigned\_responses[current\_sound]) > 1:

# Move the first response from the current sound to the previous sound

assigned\_responses[prev\_sound].append(assigned\_responses[current\_sound].pop(0))

# Prepare data for the DataFrame

table\_data = {

f"sound\_play\_times\_{frequency}Hz": [],

f"first\_response\_{frequency}Hz": [],

f"reaction\_time\_{frequency}Hz": [],

f"additional\_responses\_{frequency}Hz": [],

f"accuracy\_{frequency}Hz": []

}

accuracy\_sum = 0 # Track sum of accuracies for total accuracy calculation

reaction\_time\_sum = 0 # Sum of all valid reaction times for average reaction time calculation

reaction\_time\_count = 0 # Count of valid reaction times

for sound\_time in sound\_play\_times:

responses = assigned\_responses[sound\_time]

# Add data to each column

table\_data[f"sound\_play\_times\_{frequency}Hz"].append(round(sound\_time, 2))

if responses:

first\_response = round(responses[0], 2)

reaction\_time = round(first\_response - sound\_time, 2)

table\_data[f"first\_response\_{frequency}Hz"].append(first\_response)

table\_data[f"reaction\_time\_{frequency}Hz"].append(reaction\_time)

# Update reaction time sum and count for average calculation

reaction\_time\_sum += reaction\_time

reaction\_time\_count += 1

# Determine accuracy

accuracy = 0.5 if len(responses) > 1 else 1

table\_data[f"accuracy\_{frequency}Hz"].append(accuracy)

accuracy\_sum += accuracy

else:

table\_data[f"first\_response\_{frequency}Hz"].append("None")

table\_data[f"reaction\_time\_{frequency}Hz"].append("N/A")

table\_data[f"accuracy\_{frequency}Hz"].append(0)

accuracy\_sum += 0

table\_data[f"additional\_responses\_{frequency}Hz"].append([round(r, 2) for r in responses[1:]] if len(responses) > 1 else "None")

# Calculate total accuracy and average reaction time

total\_accuracy = round((accuracy\_sum / len(sound\_play\_times)) \* 100, 2)

average\_reaction\_time = round((reaction\_time\_sum / reaction\_time\_count), 2) if reaction\_time\_count > 0 else "N/A"

return table\_data, total\_accuracy, average\_reaction\_time

# Example usage for multiple frequencies

frequency\_index\_mapping = {

1: 3,

2: 7,

2.5: 11,

3: 15,

3.5: 19,

4: 23,

4.5: 27,

5: 31,

5.5: 35,

6: 39

}

# Load the data from the CSV file

file\_path = r'C:\Users\12035\Desktop\1. phd\24 motor task\data\0200.csv'

data = pd.read\_csv(file\_path)

# Loop through frequencies

for frequency, data\_index in frequency\_index\_mapping.items():

formatted\_frequency = str(frequency).replace('.', '\_')

# Get the pressed data for the current frequency

pressed\_raw = ast.literal\_eval(data.loc[data\_index, f'resp\_{formatted\_frequency}Hz\_2.rt'])

pressed\_raw = [float(x) for x in pressed\_raw if x is not None]

pressed = np.array(pressed\_raw) - data.loc[data\_index, f'resp\_{formatted\_frequency}Hz\_2.started']

# Get sound play times for the current frequency

play\_times\_raw = data.loc[data\_index, f'sound{formatted\_frequency}Hz\_play\_times']

play\_times\_original = ast.literal\_eval(play\_times\_raw)

play\_times = play\_times\_original[:-1]

# Calculate reaction time and accuracy for Finger 2 (adjust as needed for other fingers)

table\_data, total\_accuracy, average\_reaction\_time = calculate\_reaction\_time\_and\_accuracy(pressed, play\_times, frequency)

# Create DataFrame

df = pd.DataFrame(table\_data)

# Display results

print(f"\nFrequency: {frequency}Hz")

print(df)

print(f"Total Accuracy\_{frequency}Hz: {total\_accuracy}%")

print(f"Average Reaction Time\_{frequency}Hz: {average\_reaction\_time}")

# Prepare a summary table for all frequencies

summary\_data = {

"Block": list(frequency\_index\_mapping.keys()),

"Accuracy (%)": [],

"Reaction Time (s)": []

}

for frequency, data\_index in frequency\_index\_mapping.items():

formatted\_frequency = str(frequency).replace('.', '\_')

# Get the pressed data for the current frequency

pressed\_raw = ast.literal\_eval(data.loc[data\_index, f'resp\_{formatted\_frequency}Hz\_2.rt'])

pressed\_raw = [float(x) for x in pressed\_raw if x is not None]

pressed = np.array(pressed\_raw) - data.loc[data\_index, f'resp\_{formatted\_frequency}Hz\_2.started']

# Get sound play times for the current frequency

play\_times\_raw = data.loc[data\_index, f'sound{formatted\_frequency}Hz\_play\_times']

play\_times\_original = ast.literal\_eval(play\_times\_raw)

play\_times = play\_times\_original[:-1]

# Calculate reaction time and accuracy for Finger 2

\_, total\_accuracy, average\_reaction\_time = calculate\_reaction\_time\_and\_accuracy(pressed, play\_times, frequency)

# Add results to the summary table

summary\_data["Accuracy (%)"].append(total\_accuracy)

summary\_data["Reaction Time (s)"].append(average\_reaction\_time)

# Create DataFrame for summary

summary\_df = pd.DataFrame(summary\_data)

# Display the summary table

print(summary\_df)

# Initialize lists to store accuracy and reaction time

accuracy\_list = []

reaction\_time\_list = []

# Loop through each frequency block

for frequency, data\_index in frequency\_index\_mapping.items():

formatted\_frequency = str(frequency).replace('.', '\_')

# Get the pressed data for the current frequency

pressed\_raw = ast.literal\_eval(data.loc[data\_index, f'resp\_{formatted\_frequency}Hz\_2.rt'])

pressed\_raw = [float(x) for x in pressed\_raw if x is not None]

pressed = np.array(pressed\_raw) - data.loc[data\_index, f'resp\_{formatted\_frequency}Hz\_2.started']

# Get sound play times for the current frequency

play\_times\_raw = data.loc[data\_index, f'sound{formatted\_frequency}Hz\_play\_times']

play\_times\_original = ast.literal\_eval(play\_times\_raw)

play\_times = play\_times\_original[:-1]

# Calculate reaction time and accuracy for Finger 2

\_, total\_accuracy, average\_reaction\_time = calculate\_reaction\_time\_and\_accuracy(pressed, play\_times, frequency)

# Append results to the lists

accuracy\_list.append(total\_accuracy)

reaction\_time\_list.append(average\_reaction\_time)

# Display the lists

print("Accuracy List:", accuracy\_list)

print("Reaction Time List:", reaction\_time\_list)

# Initialize lists to store accuracy and reaction time

accuracy\_list = []

reaction\_time\_list = []

# Loop through each frequency block

for frequency, data\_index in frequency\_index\_mapping.items():

formatted\_frequency = str(frequency).replace('.', '\_')

# Get the pressed data for the current frequency

pressed\_raw = ast.literal\_eval(data.loc[data\_index, f'resp\_{formatted\_frequency}Hz\_2.rt'])

pressed\_raw = [float(x) for x in pressed\_raw if x is not None]

pressed = np.array(pressed\_raw) - data.loc[data\_index, f'resp\_{formatted\_frequency}Hz\_2.started']

# Get sound play times for the current frequency

play\_times\_raw = data.loc[data\_index, f'sound{formatted\_frequency}Hz\_play\_times']

play\_times\_original = ast.literal\_eval(play\_times\_raw)

play\_times = play\_times\_original[:-1]

# Calculate reaction time and accuracy for Finger 2

\_, total\_accuracy, average\_reaction\_time = calculate\_reaction\_time\_and\_accuracy(pressed, play\_times, frequency)

# Append results to the lists

accuracy\_list.append(total\_accuracy)

reaction\_time\_list.append(average\_reaction\_time)

# Display the lists

print("Accuracy List:", accuracy\_list)

print("Reaction Time List:", reaction\_time\_list)