import pandas as pd

import ast

import matplotlib.pyplot as plt

import numpy as np

from pathlib import Path

from IPython.display import HTML, display

def process\_block(data, block\_index):

fingers\_data = []

for i in range(1, 6):

rt = ast.literal\_eval(data.loc[block\_index, f'resp\_{i}.rt'])

rt = [float(x) for x in rt if x is not None]

duration = ast.literal\_eval(data.loc[block\_index, f'resp\_{i}.duration'])

duration = [10 if x is None else x for x in duration]

released = [x + y for x, y in zip(rt, duration)]

intervals = [(released[j], rt[j + 1]) for j in range(len(rt) - 1)]

fingers\_data.append(intervals)

return tuple(fingers\_data)

def find\_overlapping\_intervals(data, labels):

overlaps = []

for i, finger\_data in enumerate(data):

for j in range(len(finger\_data)):

for k in range(j + 1, len(finger\_data)):

start1, end1 = finger\_data[j]

start2, end2 = finger\_data[k]

if max(start1, start2) < min(end1, end2):

overlaps.append((labels[i], labels[i], (start1, end1), (start2, end2)))

for j in range(i + 1, len(data)):

for interval1 in finger\_data:

for interval2 in data[j]:

start1, end1 = interval1

start2, end2 = interval2

if max(start1, start2) < min(end1, end2):

overlaps.append((labels[i], labels[j], interval1, interval2))

return overlaps

def remove\_overlapping\_intervals(data, overlaps, labels):

updated\_data = [finger\_data[:] for finger\_data in data]

for overlap in overlaps:

finger1, finger2, interval1, interval2 = overlap

index1 = labels.index(finger1)

index2 = labels.index(finger2)

if interval1 in updated\_data[index1]:

updated\_data[index1].remove(interval1)

if interval2 in updated\_data[index2]:

updated\_data[index2].remove(interval2)

return updated\_data

def print\_colored\_sequence(timeline\_marks):

"""Print the sequence with deviations from the pattern colored in red using HTML."""

pattern = [1, 2, 3, 4, 5]

current\_position = 0

html\_parts = ['<div style="font-family: monospace;">Timeline Marks: [']

for i, mark in enumerate(timeline\_marks):

expected\_number = pattern[current\_position]

if mark != expected\_number:

# Number doesn't match pattern - display in red

html\_parts.append(f'<span style="color: red;">{mark}</span>')

else:

# Number matches pattern - display normally

html\_parts.append(str(mark))

# Move to next position in pattern

current\_position = (current\_position + 1) % 5

# Add comma and space if not the last number

if i < len(timeline\_marks) - 1:

html\_parts.append(', ')

html\_parts.append(']</div>')

display(HTML(''.join(html\_parts)))

def analyze\_pattern(fingers, pattern, pattern\_name=""):

"""Analyze pattern occurrences and calculate errors."""

pattern\_length = len(pattern)

count = 0

for i in range(len(fingers) - pattern\_length + 1):

if fingers[i:i + pattern\_length] == pattern:

count += 1

return count

def analyze\_sequence(timeline\_marks):

"""Analyze the sequence using both methods."""

fingers = timeline\_marks # Our sequence of finger presses

print("\n=== Pattern Analysis ===")

# 1st line of analysis

pattern = [1, 2, 3, 4, 5]

count = analyze\_pattern(fingers, pattern)

# Calculate errors for first method

errors = 0

expected\_index = 0

for finger in fingers:

if finger == pattern[expected\_index]:

expected\_index = (expected\_index + 1) % len(pattern)

else:

errors += 1

total\_taps = len(fingers)

error\_rate = errors / total\_taps

print("\n1st line of analyses")

print(f"The pattern {pattern} appears {count} times in the list.")

print(f"Total Errors: {errors}")

print(f"Total Taps: {total\_taps}")

print(f"Error Rate: {error\_rate:.2%}")

# 2nd line of analysis

patterns = {

"pattern1": [2, 3, 4, 5],

"pattern2": [1, 3, 4, 5],

"pattern3": [1, 2, 4, 5],

"pattern4": [1, 2, 3, 5]

}

counts = {name: analyze\_pattern(fingers, p) for name, p in patterns.items()}

print("\n2nd line of analyses")

for name, p in patterns.items():

print(f"The pattern {p} appears {counts[name]} times in the list.")

# Calculate adjusted errors

errors\_1 = (counts["pattern1"] - count) + counts["pattern2"] + counts["pattern3"] + counts["pattern4"]

error\_rate\_1 = errors\_1 / total\_taps

print(f"Adjusted Errors: {errors\_1}")

print(f"Total Taps: {total\_taps}")

print(f"Adjusted Error Rate: {error\_rate\_1:.2%}")

return {

'first\_analysis': {

'pattern\_count': count,

'errors': errors,

'total\_taps': total\_taps,

'error\_rate': error\_rate

},

'second\_analysis': {

'pattern\_counts': counts,

'adjusted\_errors': errors\_1,

'total\_taps': total\_taps,

'adjusted\_error\_rate': error\_rate\_1

}

}

def analyze\_all\_blocks(file\_path):

# Load the data

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

# Store overall results

all\_results = {}

# Process all blocks

for block\_num in range(1, 11):

print(f"\n{'='\*50}")

print(f"Processing Block {block\_num}")

print(f"{'='\*50}\n")

block\_index = block\_num + 2

# Get tapping start and stop times

tapping\_started = data.loc[block\_index, 'tapping.started']

tapping\_stopped = data.loc[block\_index, 'tapping.stopped']

# Process block data

time\_intervals = process\_block(data, block\_index)

finger\_intervals = list(time\_intervals)

finger\_labels = ["Thumb", "Index", "Middle", "Ring", "Pinky"]

# Plot original data

fig, ax = plt.subplots(figsize=(12, 6))

finger\_labels\_rev = finger\_labels[::-1]

finger\_intervals\_rev = finger\_intervals[::-1]

for i, finger\_data in enumerate(finger\_intervals\_rev):

for start, end in finger\_data:

ax.barh(i, end - start, left=start, color="green", edgecolor="black", height=0.4)

ax.set\_yticks(range(len(finger\_labels\_rev)))

ax.set\_yticklabels(finger\_labels\_rev)

ax.set\_xlabel("Time (s)")

ax.set\_title(f"Buttons Released for Each Finger - Block {block\_num}")

ax.set\_xticks(np.arange(int(tapping\_started), int(tapping\_stopped) + 1, 1))

plt.tight\_layout()

plt.savefig(fr'C:\Users\12035\Desktop\plots\finger\_press\_visualization\_block\_{block\_num}.png',

dpi=300, bbox\_inches='tight')

plt.close()

# Find overlaps

overlapping\_intervals = find\_overlapping\_intervals(finger\_intervals, finger\_labels)

# Print overlaps

for overlap in overlapping\_intervals:

print(f"Overlap detected between {overlap[0]} and {overlap[1]}: {overlap[2]} and {overlap[3]}")

# Remove overlaps and create cleaned data

finger\_intervals\_cleaned = remove\_overlapping\_intervals(finger\_intervals, overlapping\_intervals, finger\_labels)

# Print the cleaned intervals

for finger, intervals in zip(finger\_labels, finger\_intervals\_cleaned):

print(f"{finger}: {intervals}")

# Plot cleaned data

fig, ax = plt.subplots(figsize=(12, 6))

finger\_intervals\_cleaned\_rev = finger\_intervals\_cleaned[::-1]

for i, finger\_data in enumerate(finger\_intervals\_cleaned\_rev):

for start, end in finger\_data:

ax.barh(i, end - start, left=start, color="blue", edgecolor="black", height=0.4)

ax.set\_yticks(range(len(finger\_labels\_rev)))

ax.set\_yticklabels(finger\_labels\_rev)

ax.set\_xlabel("Time (s)")

ax.set\_title(f"Button Press Durations Without Overlaps - Block {block\_num}")

ax.set\_xticks(np.arange(int(tapping\_started), int(tapping\_stopped) + 1, 1))

plt.tight\_layout()

plt.savefig(fr'C:\Users\12035\Desktop\plots\finger\_press\_cleaned\_visualization\_block\_{block\_num}.png',

dpi=300, bbox\_inches='tight')

plt.close()

# Calculate timeline marks

finger\_marks = {finger: idx + 1 for idx, finger in enumerate(finger\_labels)}

combined\_timeline = []

for finger\_idx, finger\_data in enumerate(finger\_intervals\_cleaned):

mark = finger\_marks[finger\_labels[finger\_idx]]

for interval in finger\_data:

combined\_timeline.append((interval[0], mark))

combined\_timeline.sort(key=lambda x: x[0])

timeline\_marks = [mark for \_, mark in combined\_timeline]

# Print colored sequence

print\_colored\_sequence(timeline\_marks)

# Perform pattern analysis

pattern\_results = analyze\_sequence(timeline\_marks)

all\_results[block\_num] = pattern\_results

return all\_results

if \_\_name\_\_ == "\_\_main\_\_":

# Set your file path

file\_path = r"C:\Users\12035\Desktop\1. phd\24 motor task\data\0400.csv"

# Run analysis for all blocks

results = analyze\_all\_blocks(file\_path)

# Print summary of all blocks

print("\n\n" + "="\*50)

print("SUMMARY OF ALL BLOCKS")

print("="\*50)

for block\_num, block\_results in results.items():

print(f"\nBlock {block\_num}:")

print("First Analysis:")

print(f" Pattern Count: {block\_results['first\_analysis']['pattern\_count']}")

print(f" Error Rate: {block\_results['first\_analysis']['error\_rate']:.2%}")

print("Second Analysis:")

print(f" Adjusted Error Rate: {block\_results['second\_analysis']['adjusted\_error\_rate']:.2%}")