import pandas as pd

import numpy as np

from collections import defaultdict

import ast

# ANSI color codes

RED = '\033[91m'

RESET = '\033[0m'

def highlight\_non\_patterns(fingers, pattern):

"""Highlight all sequences that are NOT part of the main pattern."""

if not fingers:

return "[]"

pattern\_length = len(pattern)

highlighted\_fingers = fingers.copy()

pattern\_indices = set()

# Find all pattern occurrences first

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

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

# Mark indices that are part of the pattern

for j in range(i, i + pattern\_length):

pattern\_indices.add(j)

# Create the highlighted string - now highlighting indices NOT in pattern\_indices

result = ["["]

for i, finger in enumerate(fingers):

if i > 0:

result.append(", ")

# Add color if this index is NOT part of a pattern

if i not in pattern\_indices:

result.append(f"{RED}{finger}{RESET}")

else:

result.append(str(finger))

result.append("]")

return "".join(result)

def process\_finger\_data(data\_row, resp\_columns):

"""Process finger response data for a given row."""

finger\_data = []

for i, col in enumerate(resp\_columns, 1):

try:

values = ast.literal\_eval(data\_row[col])

finger\_data.extend((float(x), i) for x in values if x is not None)

except (ValueError, SyntaxError) as e:

print(f"Warning: Could not process data in row {data\_row.name}, column {col}")

continue

if not finger\_data:

return [], []

# Sort by time

finger\_data.sort(key=lambda x: x[0])

times, fingers = zip(\*finger\_data)

return list(times), list(fingers)

def count\_pattern\_occurrences(fingers, pattern):

"""Count occurrences of a given pattern in finger sequence."""

if not fingers:

return 0

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 calculate\_error\_metrics(fingers, main\_pattern):

"""Calculate error metrics using both methods."""

if not fingers:

return {

'sequential\_errors': 0,

'sequential\_error\_rate': 0,

'adjusted\_errors': 0,

'adjusted\_error\_rate': 0,

'total\_taps': 0,

'pattern\_counts': {

'main\_pattern': 0,

'sub\_patterns': dict(zip(['2345', '1345', '1245', '1235'], [0] \* 4))

}

}

# First method: Sequential pattern matching

errors = 0

expected\_index = 0

for finger in fingers:

if finger == main\_pattern[expected\_index]:

expected\_index = (expected\_index + 1) % len(main\_pattern)

else:

errors += 1

# Second method: Pattern occurrence analysis

sub\_patterns = [

[2, 3, 4, 5],

[1, 3, 4, 5],

[1, 2, 4, 5],

[1, 2, 3, 5]

]

main\_pattern\_count = count\_pattern\_occurrences(fingers, main\_pattern)

sub\_pattern\_counts = [count\_pattern\_occurrences(fingers, p) for p in sub\_patterns]

# Calculate adjusted errors

adjusted\_errors = (sub\_pattern\_counts[0] - main\_pattern\_count) + sum(sub\_pattern\_counts[1:])

total\_taps = len(fingers)

return {

'sequential\_errors': errors,

'sequential\_error\_rate': errors / total\_taps if total\_taps > 0 else 0,

'adjusted\_errors': adjusted\_errors,

'adjusted\_error\_rate': adjusted\_errors / total\_taps if total\_taps > 0 else 0,

'total\_taps': total\_taps,

'pattern\_counts': {

'main\_pattern': main\_pattern\_count,

'sub\_patterns': dict(zip(['2345', '1345', '1245', '1235'], sub\_pattern\_counts))

}

}

def analyze\_block(data, block\_number, row\_mapping, resp\_columns):

"""Analyze a single block of finger pattern data."""

row\_index = row\_mapping[block\_number]

main\_pattern = [1, 2, 3, 4, 5]

try:

times, fingers = process\_finger\_data(data.loc[row\_index], resp\_columns)

except KeyError:

print(f"\nBlock {block\_number}: Row index {row\_index} not found in data")

return None

if not fingers:

print(f"\nBlock {block\_number}: No valid data found")

return None

print(f"\nBlock {block\_number}")

print("Times:", times)

print("Fingers:", highlight\_non\_patterns(fingers, main\_pattern))

metrics = calculate\_error\_metrics(fingers, main\_pattern)

print("\n1st line of analyzes")

print(f"Total Errors: {metrics['sequential\_errors']}")

print(f"Total Taps: {metrics['total\_taps']}")

print(f"Error Rate: {metrics['sequential\_error\_rate']:.2%}")

print("\n2nd line of analyzes")

print(f"Pattern counts:")

print(f"Main pattern {main\_pattern}: {metrics['pattern\_counts']['main\_pattern']}")

for pattern\_name, count in metrics['pattern\_counts']['sub\_patterns'].items():

print(f"Pattern {pattern\_name}: {count}")

print(f"Adjusted Errors: {metrics['adjusted\_errors']}")

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

return metrics

def generate\_summary\_report(all\_metrics):

"""Generate a summary report across all blocks."""

if not all\_metrics:

return

print("\n=== SUMMARY REPORT ACROSS ALL BLOCKS ===")

# Calculate averages and totals

total\_blocks = len(all\_metrics)

avg\_error\_rate = np.mean([m['sequential\_error\_rate'] for m in all\_metrics])

avg\_adjusted\_error\_rate = np.mean([m['adjusted\_error\_rate'] for m in all\_metrics])

total\_taps = sum(m['total\_taps'] for m in all\_metrics)

print(f"\nTotal blocks analyzed: {total\_blocks}")

print(f"Total taps across all blocks: {total\_taps}")

print(f"Average error rate: {avg\_error\_rate:.2%}")

print(f"Average adjusted error rate: {avg\_adjusted\_error\_rate:.2%}")

# Find best and worst blocks

error\_rates = [(i, m['sequential\_error\_rate']) for i, m in enumerate(all\_metrics, 1)]

best\_block = min(error\_rates, key=lambda x: x[1])

worst\_block = max(error\_rates, key=lambda x: x[1])

print(f"\nBest performing block: Block {best\_block[0]} (Error rate: {best\_block[1]:.2%})")

print(f"Worst performing block: Block {worst\_block[0]} (Error rate: {worst\_block[1]:.2%})")

def main():

# Load the data from the CSV file

file\_path = r'C:\Users\12035\Desktop\1. phd\24 motor task\data\0300.csv'

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

# Define response columns

resp\_columns = [f'resp\_{i}.rt' for i in range(1, 6)]

# Define block to row index mapping

block\_row\_mapping = {

1: 3,

2: 4,

3: 5,

4: 6,

5: 7,

6: 8,

7: 9,

8: 10,

9: 11,

10: 12

}

all\_metrics = []

# Process each block using the mapping

for block\_num in sorted(block\_row\_mapping.keys()):

metrics = analyze\_block(data, block\_num, block\_row\_mapping, resp\_columns)

if metrics:

all\_metrics.append(metrics)

# Generate summary report

generate\_summary\_report(all\_metrics)

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

main()