import pandas as pd

import numpy as np

from fuzzywuzzy import fuzz, process

from fuzzywuzzy.utils import full\_process

import re

import itertools

from collections import defaultdict

class AdvancedNameMatcher:

def \_\_init\_\_(self, threshold=80):

self.threshold = threshold

def clean\_name(self, name):

"""Clean and standardize names for better matching"""

if pd.isna(name) or name == '':

return ''

# Convert to string and uppercase

name = str(name).upper().strip()

# Remove extra spaces and special characters

name = re.sub(r'[^\w\s]', ' ', name)

name = re.sub(r'\s+', ' ', name).strip()

return name

def generate\_name\_variants(self, name):

"""Generate different variants of a name for matching"""

if not name:

return [name]

variants = set()

name\_parts = name.split()

if len(name\_parts) <= 1:

variants.add(name)

return list(variants)

# Original name

variants.add(name)

# Reversed order (Ama Atta -> Atta Ama)

variants.add(' '.join(reversed(name\_parts)))

# All permutations for names with 2-3 parts

if len(name\_parts) <= 3:

for perm in itertools.permutations(name\_parts):

variants.add(' '.join(perm))

# First name + last name combinations

if len(name\_parts) >= 2:

variants.add(f"{name\_parts[0]} {name\_parts[-1]}")

variants.add(f"{name\_parts[-1]} {name\_parts[0]}")

# Remove middle initials/names and try combinations

if len(name\_parts) >= 3:

# First and last only

variants.add(f"{name\_parts[0]} {name\_parts[-1]}")

# All combinations without middle parts

for i in range(len(name\_parts)):

remaining = [name\_parts[j] for j in range(len(name\_parts)) if j != i]

if len(remaining) >= 2:

variants.add(' '.join(remaining))

return list(variants)

def fuzzy\_match\_names(self, name1, name2):

"""Advanced fuzzy matching between two names"""

if not name1 or not name2:

return 0

# Clean both names

clean\_name1 = self.clean\_name(name1)

clean\_name2 = self.clean\_name(name2)

if clean\_name1 == clean\_name2:

return 100

# Generate variants for both names

variants1 = self.generate\_name\_variants(clean\_name1)

variants2 = self.generate\_name\_variants(clean\_name2)

max\_score = 0

# Compare all variants

for v1 in variants1:

for v2 in variants2:

if v1 and v2:

# Try different fuzzy matching methods

scores = [

fuzz.ratio(v1, v2),

fuzz.partial\_ratio(v1, v2),

fuzz.token\_sort\_ratio(v1, v2),

fuzz.token\_set\_ratio(v1, v2)

]

max\_score = max(max\_score, max(scores))

return max\_score

def find\_best\_matches(self, stallion\_df, master\_df):

"""Find best matches between STALLION\_EMPTY and Master files"""

results = []

print(f"Processing {len(stallion\_df)} names from STALLION\_EMPTY...")

print(f"Matching against {len(master\_df)} names from Master...")

for idx, stallion\_row in stallion\_df.iterrows():

stallion\_name = stallion\_row.get('Name', '')

if pd.isna(stallion\_name) or stallion\_name == '':

continue

best\_match = None

best\_score = 0

best\_ssnit = None

# Compare with all master names

for master\_idx, master\_row in master\_df.iterrows():

master\_name = master\_row.get('Name', '')

ssnit\_number = master\_row.get('SSNIT', '')

if pd.isna(master\_name) or master\_name == '':

continue

score = self.fuzzy\_match\_names(stallion\_name, master\_name)

if score > best\_score and score >= self.threshold:

best\_score = score

best\_match = master\_name

best\_ssnit = ssnit\_number

results.append({

'STALLION\_Name': stallion\_name,

'Matched\_Master\_Name': best\_match if best\_match else 'No Match',

'SSNIT\_Number': best\_ssnit if best\_ssnit else 'No Match',

'Match\_Score': best\_score,

'Match\_Quality': self.get\_match\_quality(best\_score)

})

# Progress indicator

if (idx + 1) % 100 == 0:

print(f"Processed {idx + 1} names...")

return pd.DataFrame(results)

def get\_match\_quality(self, score):

"""Categorize match quality based on score"""

if score >= 95:

return 'Excellent'

elif score >= 85:

return 'Good'

elif score >= self.threshold:

return 'Fair'

else:

return 'Poor'

def main():

"""Main function to run the name matching process"""

try:

# File paths

stallion\_path = r"C:\Users\USER\Desktop\STALLION\_COMPLETED\STALLION\_EMPTY.xlsx"

master\_path = r"C:\Users\USER\Desktop\STALLION\_COMPLETED\Member.xlsx"

# Load the Excel files with headers in row 1 (index 0)

print("Loading STALLION\_EMPTY file...")

stallion\_df = pd.read\_excel(stallion\_path, header=0) # Headers in row 1, data from row 2

print("Loading Member file...")

master\_df = pd.read\_excel(master\_path, header=0) # Headers in row 1, data from row 2

# Display column information for verification

print(f"\nSTALLION\_EMPTY columns: {list(stallion\_df.columns)}")

print(f"Member columns: {list(master\_df.columns)}")

# Extract relevant columns

# STALLION\_EMPTY: Names in column D (index 3)

# Member: Names in column F (index 5), SSNIT in column B (index 1)

# Get column D (index 3) from STALLION\_EMPTY

if stallion\_df.shape[1] >= 4: # Ensure column D exists

stallion\_names = stallion\_df.iloc[:, 3].to\_frame() # Column D

stallion\_names.columns = ['Name']

else:

print(f"Error: STALLION\_EMPTY doesn't have enough columns. Found {stallion\_df.shape[1]} columns, need at least 4.")

return None

# Get columns B and F (indices 1 and 5) from Member

if master\_df.shape[1] >= 6: # Ensure columns B and F exist

master\_data = master\_df.iloc[:, [1, 5]] # Columns B and F

master\_data.columns = ['SSNIT', 'Name']

else:

print(f"Error: Member doesn't have enough columns. Found {master\_df.shape[1]} columns, need at least 6.")

return None

print(f"STALLION\_EMPTY contains {len(stallion\_names)} names")

print(f"Member contains {len(master\_data)} names")

# Initialize the matcher

matcher = AdvancedNameMatcher(threshold=75) # Adjust threshold as needed

# Find matches

print("\nStarting fuzzy matching process...")

results\_df = matcher.find\_best\_matches(stallion\_names, master\_data)

# Display summary

total\_matches = len(results\_df[results\_df['SSNIT\_Number'] != 'No Match'])

print(f"\nMatching completed!")

print(f"Total names processed: {len(results\_df)}")

print(f"Successful matches: {total\_matches}")

print(f"Match rate: {(total\_matches/len(results\_df))\*100:.1f}%")

# Show match quality distribution

quality\_counts = results\_df['Match\_Quality'].value\_counts()

print(f"\nMatch Quality Distribution:")

for quality, count in quality\_counts.items():

print(f" {quality}: {count}")

# Save results to the same directory

output\_filename = r"C:\Users\USER\Desktop\STALLION\_COMPLETED\Name\_Matching\_Results.xlsx"

results\_df.to\_excel(output\_filename, index=False)

print(f"\nResults saved to {output\_filename}")

# Display first few results

print(f"\nFirst 10 results:")

print(results\_df.head(10).to\_string(index=False))

# Show some example matches

good\_matches = results\_df[results\_df['Match\_Score'] >= 80].head(5)

if not good\_matches.empty:

print(f"\nExample good matches:")

for \_, row in good\_matches.iterrows():

print(f" '{row['STALLION\_Name']}' -> '{row['Matched\_Master\_Name']}' (Score: {row['Match\_Score']}, SSNIT: {row['SSNIT\_Number']})")

return results\_df

except FileNotFoundError as e:

print(f"Error: Could not find the Excel file. Please make sure both files exist:")

print(" - C:\\Users\\USER\\Desktop\\STALLION\_COMPLETED\\STALLION\_EMPTY.xlsx")

print(" - C:\\Users\\USER\\Desktop\\STALLION\_COMPLETED\\Member.xlsx")

print(f"Error details: {e}")

except Exception as e:

print(f"An error occurred: {e}")

print("Please check your file formats and try again.")

# Installation instructions (run these in your Jupyter notebook first)

def install\_requirements():

"""

Run this function first to install required packages:

!pip install fuzzywuzzy python-levenshtein pandas openpyxl

"""

pass

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

# Uncomment the next line if you need to install packages

# install\_requirements()

results = main()