import camelot

import pandas as pd

import os

import numpy as np

import re

from openpyxl import Workbook

from openpyxl.styles import Font, Alignment, PatternFill, Border, Side

from openpyxl.utils.dataframe import dataframe\_to\_rows

def extract\_header\_info(df):

"""Extract header information from the statement"""

header\_info = {}

try:

*# Convert all data to string and flatten*

all\_text = []

for row in df.values:

for cell in row:

cell\_str = str(cell).strip()

if cell\_str and cell\_str != 'nan' and cell\_str != 'None':

all\_text.append(cell\_str)

text\_string = ' '.join(all\_text)

print(f"Extracted text length: {len(text\_string)} characters")

*# Extract key information using regex*

patterns = {

'account\_number': r'(?:ACCOUNT\s\*:?\s\*|ACCOUNT\s+)(\d{10,})',

'account\_name': r'ACCOUNT NAME\s\*:?\s\*(\d+)',

'currency': r'CURRENCY\s\*:?\s\*([A-Z]{3})',

'from\_date': r'FROM\s\*:?\s\*([\d\s\w]+?)(?=TO|PRINTED|$)',

'to\_date': r'TO\s\*:?\s\*([\d\s\w]+?)(?=PRINTED|FROM|$)',

'printed\_by': r'PRINTED BY\s\*:?\s\*([\w\s]+?)(?=DATE|FROM|TO|$)',

'date\_printed': r'(\d{1,2}\s+\w+\s+\d{4})'

}

for key, pattern in patterns.items():

match = re.search(pattern, text\_string, re.IGNORECASE)

if match:

header\_info[key] = match.group(1).strip()

except Exception as e:

print(f"Error extracting header info: {e}")

*# Set default values*

header\_info = {

'account\_number': '111034114608',

'currency': 'GHS',

'from\_date': '01 MAY 2025',

'to\_date': '31 MAY 2025'

}

return header\_info

def identify\_transaction\_table(tables):

"""Identify the table containing transaction data"""

transaction\_table = None

max\_score = 0

for i, table in enumerate(tables):

df = table.df

score = 0

*# Convert all data to string and check for transaction indicators*

df\_text = ""

for col in df.columns:

df\_text += " " + df[col].astype(str).str.lower().str.cat(sep=" ")

*# Score based on transaction-related keywords*

if 'booking' in df\_text and 'date' in df\_text:

score += 10

if 'reference' in df\_text:

score += 5

if 'debit' in df\_text or 'credit' in df\_text:

score += 10

if 'balance' in df\_text:

score += 5

if 'deposit' in df\_text:

score += 5

if 'may' in df\_text and '25' in df\_text: *# Based on your statement dates*

score += 5

*# Check for amount patterns*

amount\_pattern = r'\d+\.\d{2}'

if re.search(amount\_pattern, df\_text):

score += 8

*# Prefer tables with more rows (likely transaction data)*

score += len(df) \* 0.1

print(f"Table {i+1}: {len(df)} rows, score: {score:.1f}")

if score > max\_score:

max\_score = score

transaction\_table = i

return transaction\_table

def process\_transaction\_table(df):

"""Process the transaction table to extract meaningful data"""

*# Remove completely empty rows and columns*

df = df.dropna(how='all').reset\_index(drop=True)

df = df.loc[:, ~df.isna().all()]

*# Clean all cells*

for col in df.columns:

df[col] = df[col].astype(str).str.strip().replace('nan', '')

*# Find the header row (contains Booking Date, Reference, etc.)*

header\_row = None

for i, row in df.iterrows():

row\_text = ' '.join(row.astype(str)).lower()

if 'booking' in row\_text and 'date' in row\_text and ('reference' in row\_text or 'description' in row\_text):

header\_row = i

break

if header\_row is not None:

*# Use the found row as headers*

new\_headers = []

for col in df.iloc[header\_row]:

header = str(col).strip()

if header and header != 'nan':

new\_headers.append(header)

else:

new\_headers.append(f'Column\_{len(new\_headers)}')

df.columns = new\_headers[:len(df.columns)]

df = df.iloc[header\_row + 1:].reset\_index(drop=True)

*# Process transactions - group related rows*

processed\_transactions = []

current\_transaction = {}

for i, row in df.iterrows():

row\_dict = row.to\_dict()

*# Check if this row starts a new transaction (has booking date)*

booking\_date = str(row\_dict.get(df.columns[0], '')).strip()

if re.match(r'\d{2}\s+\w+\s+\d{2}', booking\_date): *# Matches "05 MAY 25" format*

*# Save previous transaction if exists*

if current\_transaction:

processed\_transactions.append(current\_transaction.copy())

*# Start new transaction*

current\_transaction = {

'Booking\_Date': booking\_date,

'Reference': str(row\_dict.get(df.columns[1] if len(df.columns) > 1 else df.columns[0], '')),

'Account\_Number': '',

'Account\_Name': '',

'Description': '',

'Value\_Date': '',

'Debit': '',

'Credit': '',

'Closing\_Balance': '',

'Additional\_Info': []

}

*# Extract other fields from the row*

for j, col in enumerate(df.columns):

value = str(row\_dict.get(col, '')).strip()

if value and value != 'nan':

if j == 2: *# Account Number column*

current\_transaction['Account\_Number'] = value

elif j == 3: *# Account Name column*

current\_transaction['Account\_Name'] = value

elif j == 4: *# Description column*

current\_transaction['Description'] = value

elif j == 5: *# Value Date column*

current\_transaction['Value\_Date'] = value

elif j == 6: *# Debit column*

current\_transaction['Debit'] = value

elif j == 7: *# Credit column*

current\_transaction['Credit'] = value

elif j == 8: *# Closing Balance column*

current\_transaction['Closing\_Balance'] = value

else:

*# This is additional information for the current transaction*

if current\_transaction:

non\_empty\_values = [str(val).strip() for val in row\_dict.values()

if str(val).strip() and str(val).strip() != 'nan']

if non\_empty\_values:

current\_transaction['Additional\_Info'].extend(non\_empty\_values)

*# Add the last transaction*

if current\_transaction:

processed\_transactions.append(current\_transaction)

*# Convert to DataFrame*

result\_df = pd.DataFrame(processed\_transactions)

*# Combine additional info into description*

if 'Additional\_Info' in result\_df.columns:

result\_df['Full\_Description'] = result\_df.apply(

lambda row: f"{row['Description']} {' '.join(row['Additional\_Info']) if row['Additional\_Info'] else ''}".strip(),

axis=1

)

result\_df = result\_df.drop('Additional\_Info', axis=1)

return result\_df

def create\_formatted\_excel(header\_info, transaction\_df, output\_path):

"""Create a well-formatted Excel file that matches the original statement"""

wb = Workbook()

ws = wb.active

ws.title = "Bank Statement"

*# Define styles*

header\_font = Font(bold=True, size=12)

normal\_font = Font(size=10)

amount\_font = Font(size=10, name='Calibri')

header\_fill = PatternFill(start\_color="366092", end\_color="366092", fill\_type="solid")

thin\_border = Border(

left=Side(style='thin'),

right=Side(style='thin'),

top=Side(style='thin'),

bottom=Side(style='thin')

)

current\_row = 1

*# Add header information*

ws.merge\_cells(f'A{current\_row}:I{current\_row}')

ws[f'A{current\_row}'] = "UNIVERSAL MERCHANT BANK - BANK STATEMENT"

ws[f'A{current\_row}'].font = Font(bold=True, size=14)

ws[f'A{current\_row}'].alignment = Alignment(horizontal='center')

current\_row += 2

*# Add account information*

info\_items = [

('ACCOUNT:', header\_info.get('account\_number', '')),

('ACCOUNT NAME:', header\_info.get('account\_name', '')),

('CURRENCY:', header\_info.get('currency', 'GHS')),

('FROM:', header\_info.get('from\_date', '')),

('TO:', header\_info.get('to\_date', '')),

('PRINTED BY:', header\_info.get('printed\_by', '')),

('DATE:', header\_info.get('date\_printed', ''))

]

for i, (label, value) in enumerate(info\_items):

if i % 2 == 0: *# Left column*

ws[f'A{current\_row}'] = label

ws[f'B{current\_row}'] = value

ws[f'A{current\_row}'].font = header\_font

else: *# Right column*

ws[f'E{current\_row}'] = label

ws[f'F{current\_row}'] = value

ws[f'E{current\_row}'].font = header\_font

current\_row += 1

current\_row += 2

*# Add transaction table headers*

headers = ['Booking Date', 'Reference', 'Account Number', 'Account Name',

'Description', 'Value Date', 'Debit', 'Credit', 'Closing Balance']

for col, header in enumerate(headers, 1):

cell = ws.cell(row=current\_row, column=col, value=header)

cell.font = header\_font

cell.fill = header\_fill

cell.border = thin\_border

cell.alignment = Alignment(horizontal='center', wrap\_text=True)

current\_row += 1

*# Add transaction data*

for \_, row in transaction\_df.iterrows():

ws.cell(row=current\_row, column=1, value=row.get('Booking\_Date', ''))

ws.cell(row=current\_row, column=2, value=row.get('Reference', ''))

ws.cell(row=current\_row, column=3, value=row.get('Account\_Number', ''))

ws.cell(row=current\_row, column=4, value=row.get('Account\_Name', ''))

ws.cell(row=current\_row, column=5, value=row.get('Full\_Description', row.get('Description', '')))

ws.cell(row=current\_row, column=6, value=row.get('Value\_Date', ''))

*# Handle numeric values*

debit = row.get('Debit', '')

credit = row.get('Credit', '')

balance = row.get('Closing\_Balance', '')

try:

if debit and str(debit).strip():

ws.cell(row=current\_row, column=7, value=float(str(debit).replace(',', '')))

except:

ws.cell(row=current\_row, column=7, value=debit)

try:

if credit and str(credit).strip():

ws.cell(row=current\_row, column=8, value=float(str(credit).replace(',', '')))

except:

ws.cell(row=current\_row, column=8, value=credit)

try:

if balance and str(balance).strip():

ws.cell(row=current\_row, column=9, value=float(str(balance).replace(',', '')))

except:

ws.cell(row=current\_row, column=9, value=balance)

*# Apply formatting*

for col in range(1, 10):

cell = ws.cell(row=current\_row, column=col)

cell.border = thin\_border

cell.font = normal\_font

if col in [7, 8, 9]: *# Amount columns*

cell.alignment = Alignment(horizontal='right')

elif col == 5: *# Description column*

cell.alignment = Alignment(wrap\_text=True)

current\_row += 1

*# Adjust column widths*

column\_widths = [12, 15, 15, 15, 35, 12, 12, 12, 15]

for i, width in enumerate(column\_widths, 1):

ws.column\_dimensions[ws.cell(row=1, column=i).column\_letter].width = width

*# Save the workbook*

wb.save(output\_path)

def main():

*# Define the path to the PDF file*

pdf\_path = r'C:\Users\USER\Desktop\BANK\STATEMENT1.pdf'

*# Check if file exists*

if not os.path.exists(pdf\_path):

print(f"Error: PDF file not found at {pdf\_path}")

return

print(f"Processing PDF: {pdf\_path}")

try:

*# Extract tables using multiple methods*

print("Extracting tables from PDF...")

*# Try lattice first for structured tables*

try:

tables = camelot.read\_pdf(pdf\_path, flavor='lattice', pages='all')

print(f"Lattice method extracted {len(tables)} tables")

except Exception as e:

print(f"Lattice method failed: {e}")

tables = []

if not tables or len(tables) == 0:

print("Trying stream method...")

try:

tables = camelot.read\_pdf(pdf\_path, flavor='stream', pages='all',

edge\_tol=500, split\_text=True)

print(f"Stream method extracted {len(tables)} tables")

except Exception as e:

print(f"Stream method failed: {e}")

tables = []

if not tables or len(tables) == 0:

print("No tables found. The PDF might be image-based or poorly formatted.")

return

print(f"Found {len(tables)} tables")

*# Debug: Save raw tables first with combined sheet*

raw\_excel\_path = r'C:\Users\USER\Desktop\BANK\STATEMENT1\_raw\_debug.xlsx'

print(f"Saving raw tables to {raw\_excel\_path} for debugging...")

*# Prepare combined data*

combined\_data = []

with pd.ExcelWriter(raw\_excel\_path, engine='openpyxl') as writer:

*# First, create the combined sheet*

for i, table in enumerate(tables):

df = table.df.copy()

df.insert(0, 'Source\_Table', f'Table\_{i+1}')

df.insert(1, 'Row\_Number', range(1, len(df) + 1))

combined\_data.append(df)

print(f"Table {i+1}: {df.shape} - {len(df)} rows, {len(df.columns)} columns")

*# Create combined dataframe*

if combined\_data:

try:

combined\_df = pd.concat(combined\_data, ignore\_index=True, sort=False)

combined\_df.to\_excel(writer, sheet\_name='Combined\_All\_Tables', index=False)

print(f"Combined sheet created with {len(combined\_df)} total rows")

except Exception as e:

print(f"Error creating combined sheet: {e}")

*# Fallback: create combined sheet manually*

max\_cols = max(len(df.columns) for df in combined\_data)

combined\_rows = []

for df in combined\_data:

for \_, row in df.iterrows():

row\_list = row.tolist()

*# Pad with empty strings if needed*

while len(row\_list) < max\_cols:

row\_list.append('')

combined\_rows.append(row\_list)

combined\_df = pd.DataFrame(combined\_rows)

combined\_df.to\_excel(writer, sheet\_name='Combined\_All\_Tables', index=False)

*# Then create individual table sheets*

for i, table in enumerate(tables):

df = table.df

df.to\_excel(writer, sheet\_name=f'Table\_{i+1}', index=True)

*# Process each table to find the best one*

best\_table\_idx = identify\_transaction\_table(tables)

if best\_table\_idx is None:

print("Could not identify transaction table. Using the largest table.")

best\_table\_idx = max(range(len(tables)), key=lambda i: len(tables[i].df))

print(f"Using table {best\_table\_idx + 1} as main transaction table")

*# Extract header information from all tables*

try:

all\_data = pd.concat([table.df for table in tables], ignore\_index=True)

header\_info = extract\_header\_info(all\_data)

except Exception as e:

print(f"Error combining tables: {e}")

header\_info = extract\_header\_info(tables[0].df)

print("Extracted header information:")

for key, value in header\_info.items():

print(f" {key}: {value}")

*# Process the main transaction table*

try:

transaction\_df = process\_transaction\_table(tables[best\_table\_idx].df)

print(f"Processed {len(transaction\_df)} transactions")

except Exception as e:

print(f"Error processing transaction table: {e}")

*# Fallback: use the raw table*

transaction\_df = tables[best\_table\_idx].df.copy()

transaction\_df.columns = [f'Column\_{i}' for i in range(len(transaction\_df.columns))]

*# Create output file*

excel\_path = r'C:\Users\USER\Desktop\BANK\STATEMENT1\_formatted.xlsx'

*# Create formatted Excel file*

try:

create\_formatted\_excel(header\_info, transaction\_df, excel\_path)

print(f"Excel file successfully created: {excel\_path}")

except Exception as e:

print(f"Error creating formatted Excel: {e}")

*# Fallback: simple save*

transaction\_df.to\_excel(excel\_path.replace('.xlsx', '\_simple.xlsx'), index=False)

print(f"Simple Excel file saved instead")

print("\nTransaction Summary:")

print(f"Total transactions: {len(transaction\_df)}")

if not transaction\_df.empty and len(transaction\_df.columns) > 0:

print("\nFirst few rows:")

print(transaction\_df.head(3).to\_string(index=False, max\_cols=10, max\_colwidth=20))

except Exception as e:

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

import traceback

traceback.print\_exc()

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

main()