# Excel + Template to PDF Certificate Generator (Clean Version)

# Upload Excel data + Word template → Generate filled PDF certificates

# Install required packages

!pip install pandas openpyxl python-docx --quiet

!apt-get update > /dev/null 2>&1

!apt-get install -y libreoffice > /dev/null 2>&1

import pandas as pd

from docx import Document

from docx.shared import Inches

from docx.enum.table import WD\_TABLE\_ALIGNMENT

import os

import re

from google.colab import files

import zipfile

from datetime import datetime

import subprocess

import warnings

warnings.filterwarnings('ignore')

def convert\_word\_to\_pdf(word\_filename, output\_folder):

"""Convert Word document to PDF using LibreOffice"""

try:

base\_name = os.path.splitext(os.path.basename(word\_filename))[0]

cmd = [

'libreoffice', '--headless', '--convert-to', 'pdf',

'--outdir', output\_folder, word\_filename

]

result = subprocess.run(cmd, capture\_output=True, text=True, timeout=30)

pdf\_filename = os.path.join(output\_folder, f"{base\_name}.pdf")

if os.path.exists(pdf\_filename):

if os.path.exists(word\_filename):

os.remove(word\_filename)

return pdf\_filename

else:

return word\_filename

except:

return word\_filename

def upload\_both\_files():

"""Upload both Excel file and Word template"""

print("Hey there! Welcome back user.")

print("The code is generated by udit!")

print("\n📊 Step 1: Upload Excel file")

excel\_files = files.upload()

if not excel\_files:

print("❌ No Excel file uploaded!")

return None, None

excel\_filename = list(excel\_files.keys())[0]

print(f"✅ Excel: {excel\_filename}")

print("\n📄 Step 2: Upload Word template")

word\_files = files.upload()

if not word\_files:

print("❌ No template uploaded!")

return None, None

word\_filename = list(word\_files.keys())[0]

print(f"✅ Template: {word\_filename}")

return excel\_filename, word\_filename

def analyze\_template\_placeholders(word\_filename):

"""Analyze the Word template to find placeholders"""

try:

doc = Document(word\_filename)

all\_text = []

for paragraph in doc.paragraphs:

all\_text.append(paragraph.text)

for table in doc.tables:

for row in table.rows:

for cell in row.cells:

for paragraph in cell.paragraphs:

all\_text.append(paragraph.text)

full\_text = ' '.join(all\_text)

placeholder\_patterns = [r'\[([^\]]+)\]', r'\{([^}]+)\}', r'\{\{([^}]+)\}\}', r'<<([^>]+)>>']

found\_placeholders = set()

for pattern in placeholder\_patterns:

matches = re.findall(pattern, full\_text)

for match in matches:

found\_placeholders.add(match.strip())

placeholders\_list = sorted(list(found\_placeholders))

if not placeholders\_list:

placeholders\_list = ["Invoice Date", "Address Line 1", "City postal code", "Address line 3",

"Customer Postal Code", "Customer No", "Local invoice Number", "SKF Designation", "Invoice Quantity"]

return doc, placeholders\_list

except Exception as e:

print(f"❌ Template error: {str(e)}")

return None, []

def read\_excel\_data(excel\_filename):

"""Read and process Excel data"""

try:

df = pd.read\_excel(excel\_filename)

# Convert all data to strings

for col in df.columns:

df[col] = df[col].astype(str).replace('nan', 'N/A')

# Find invoice column

invoice\_col = None

for col in df.columns:

if any(inv\_col.lower() in col.lower() for inv\_col in ['Local invoice Number', 'Invoice Number', 'Invoice No']):

invoice\_col = col

break

if not invoice\_col:

print("❌ No invoice number column found!")

return None, None

# Group by invoice

grouped\_data = {}

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

inv\_no = str(row[invoice\_col]).strip()

if inv\_no not in grouped\_data:

grouped\_data[inv\_no] = []

grouped\_data[inv\_no].append(row.to\_dict())

print(f"📊 Loaded: {len(df)} rows → {len(grouped\_data)} invoices")

return df, grouped\_data

except Exception as e:

print(f"❌ Excel error: {str(e)}")

return None, None

def map\_placeholders\_to\_excel(placeholders, excel\_columns):

"""Map template placeholders to Excel columns"""

mapping = {}

common\_mappings = {

'Invoice Date': ['Invoice Date', 'Date', 'Inv Date'],

'Address Line 1': ['Address Line 1', 'Address 1', 'Addr Line 1'],

'City postal code': ['City postal code', 'Postal Code', 'PIN Code', 'ZIP'],

'Address line 3': ['Address line 3', 'Address 3', 'Addr Line 3'],

'Customer Postal Code': ['Customer Postal Code', 'Postal Code', 'PIN Code', 'ZIP'],

'Customer No': ['Customer No.', 'Customer No', 'Customer Number', 'Cust No'],

'Local invoice Number': ['Local invoice Number', 'Invoice Number', 'Invoice No'],

'SKF Designation': ['SKF Designation', 'Product Code', 'SKF Code', 'Designation'],

'Invoice Quantity': ['Invoice Quantity', 'Quantity', 'Qty']

}

for placeholder in placeholders:

found\_mapping = None

if placeholder in excel\_columns:

found\_mapping = placeholder

else:

for common\_name, possible\_cols in common\_mappings.items():

if placeholder.lower().strip() == common\_name.lower().strip():

for excel\_col in excel\_columns:

for possible\_col in possible\_cols:

if possible\_col.lower() in excel\_col.lower():

found\_mapping = excel\_col

break

if found\_mapping:

break

break

mapping[placeholder] = found\_mapping

return mapping

def fill\_template\_with\_excel\_data(template\_path, invoice\_data, invoice\_number, mapping):

"""Fill the template with Excel data"""

doc = Document(template\_path)

first\_record = invoice\_data[0]

excel\_columns\_list = list(first\_record.keys())

# Prepare replacements

single\_value\_replacements = {}

for placeholder, excel\_column in mapping.items():

if excel\_column and placeholder not in ['SKF Designation', 'Invoice Quantity']:

value = first\_record.get(excel\_column, 'N/A')

single\_value\_replacements[placeholder] = str(value)

# Get City postal code from Column V (index 21)

city\_postal\_code = first\_record.get(excel\_columns\_list[21] if len(excel\_columns\_list) > 21 else '', 'N/A')

single\_value\_replacements['City postal code'] = str(city\_postal\_code)

single\_value\_replacements['Address Line 2'] = str(city\_postal\_code)

single\_value\_replacements['Local invoice Number'] = str(invoice\_number)

single\_value\_replacements['Invoice Number'] = str(invoice\_number)

# Replace in paragraphs

for paragraph in doc.paragraphs:

for placeholder, value in single\_value\_replacements.items():

placeholder\_formats = [f'[{placeholder}]', f'{{{placeholder}}}', f'{{{{{placeholder}}}}}', f'<<{placeholder}>>']

for placeholder\_format in placeholder\_formats:

if placeholder\_format in paragraph.text:

paragraph.text = paragraph.text.replace(placeholder\_format, str(value))

# Replace in existing tables

for table in doc.tables:

for row in table.rows:

for cell in row.cells:

for paragraph in cell.paragraphs:

for placeholder, value in single\_value\_replacements.items():

placeholder\_formats = [f'[{placeholder}]', f'{{{placeholder}}}', f'{{{{{placeholder}}}}}', f'<<{placeholder}>>']

for placeholder\_format in placeholder\_formats:

if placeholder\_format in paragraph.text:

paragraph.text = paragraph.text.replace(placeholder\_format, str(value))

# Create product table data

table\_data = []

for i, record in enumerate(invoice\_data, 1):

excel\_columns\_list = list(record.keys())

skf\_designation = record.get(excel\_columns\_list[8] if len(excel\_columns\_list) > 8 else '', 'N/A')

invoice\_quantity = record.get(excel\_columns\_list[9] if len(excel\_columns\_list) > 9 else '', 'N/A')

table\_data.append([str(i), str(skf\_designation), str(invoice\_quantity)])

# Find "Dear Sir," and insert table before it

for i, paragraph in enumerate(doc.paragraphs):

if "Dear Sir," in paragraph.text:

p = paragraph.\_element

# Create table

new\_table = doc.add\_table(rows=1, cols=3)

new\_table.style = 'Table Grid'

new\_table.alignment = WD\_TABLE\_ALIGNMENT.LEFT

# Set column widths

new\_table.columns[0].width = Inches(0.8)

new\_table.columns[1].width = Inches(3.0)

new\_table.columns[2].width = Inches(1.2)

# Add headers

header\_cells = new\_table.rows[0].cells

header\_cells[0].text = "Sr. No."

header\_cells[1].text = "SKF Product Designation"

header\_cells[2].text = "Quantity Supplied"

# Make headers bold and centered

for cell in header\_cells:

for paragraph in cell.paragraphs:

paragraph.alignment = 1

for run in paragraph.runs:

run.bold = True

# Add data rows

for row\_data in table\_data:

data\_row = new\_table.add\_row()

data\_cells = data\_row.cells

data\_cells[0].text = row\_data[0]

data\_cells[1].text = row\_data[1]

data\_cells[2].text = row\_data[2]

data\_cells[0].paragraphs[0].alignment = 1

data\_cells[2].paragraphs[0].alignment = 1

# Insert table before Dear Sir

table\_element = new\_table.\_element

p.addprevious(table\_element)

break

return doc

def generate\_certificates\_from\_template(template\_path, grouped\_data, mapping):

"""Generate PDF certificates for all invoices using the template"""

generated\_files = []

output\_folder = "SKF\_PDF\_Certificates"

if not os.path.exists(output\_folder):

os.makedirs(output\_folder)

total = len(grouped\_data)

print(f"🏭 Generating {total} certificates...")

for i, (invoice\_number, invoice\_records) in enumerate(grouped\_data.items(), 1):

try:

# Show progress every 10 files or for small batches

if i % 10 == 0 or total <= 20:

print(f" 📄 Processing {i}/{total}: {invoice\_number}")

# Fill template

doc = fill\_template\_with\_excel\_data(template\_path, invoice\_records, invoice\_number, mapping)

# Save as Word first

clean\_invoice = re.sub(r'[<>:"/\\|?\*\s]', '\_', str(invoice\_number))

word\_filename = os.path.join(output\_folder, f"{clean\_invoice}.docx")

doc.save(word\_filename)

# Convert to PDF

pdf\_filename = convert\_word\_to\_pdf(word\_filename, output\_folder)

generated\_files.append(pdf\_filename)

except Exception as e:

print(f" ❌ Error processing {invoice\_number}: {str(e)}")

continue

pdf\_count = len([f for f in generated\_files if f.endswith('.pdf')])

word\_count = len([f for f in generated\_files if f.endswith('.docx')])

print(f"✅ Generated: {pdf\_count} PDFs, {word\_count} Word files")

return generated\_files, output\_folder

def create\_download\_package(generated\_files):

"""Create ZIP package and download"""

if not generated\_files:

print("❌ No files to package!")

return

timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

zip\_filename = f"SKF\_PDF\_Certificates\_{timestamp}.zip"

try:

with zipfile.ZipFile(zip\_filename, 'w', zipfile.ZIP\_DEFLATED) as zipf:

for file\_path in generated\_files:

if os.path.exists(file\_path):

arcname = os.path.basename(file\_path)

zipf.write(file\_path, arcname)

file\_size = os.path.getsize(zip\_filename) / (1024 \* 1024)

print(f"📦 Created: {zip\_filename} ({file\_size:.1f} MB)")

# Download

files.download(zip\_filename)

print(f"✅ Download started!")

except Exception as e:

print(f"❌ Error creating package: {str(e)}")

def main():

"""Main function"""

print("🎯 SKF PDF CERTIFICATE GENERATOR")

print("=" \* 50)

# Step 1: Upload files

excel\_filename, word\_filename = upload\_both\_files()

if not excel\_filename or not word\_filename:

return

# Step 2: Analyze template

print("\n🔍 Analyzing template...")

template\_doc, placeholders = analyze\_template\_placeholders(word\_filename)

if template\_doc is None:

return

print(f"✅ Found {len(placeholders)} placeholders")

# Step 3: Read Excel data

print("\n📊 Processing Excel data...")

df, grouped\_data = read\_excel\_data(excel\_filename)

if df is None or grouped\_data is None:

return

# Step 4: Map placeholders

print("\n🔗 Mapping fields...")

mapping = map\_placeholders\_to\_excel(placeholders, list(df.columns))

mapped\_count = len([v for v in mapping.values() if v])

print(f"✅ Mapped {mapped\_count}/{len(placeholders)} fields")

# Step 5: Generate certificates

print("\n🏭 Processing certificates...")

generated\_files, output\_folder = generate\_certificates\_from\_template(word\_filename, grouped\_data, mapping)

if not generated\_files:

return

# Step 6: Package and download

print("\n📦 Creating download package...")

create\_download\_package(generated\_files)

# Final summary

pdf\_count = len([f for f in generated\_files if f.endswith('.pdf')])

word\_count = len([f for f in generated\_files if f.endswith('.docx')])

print(f"\n🎉 COMPLETED!")

print(f"📊 Input: {len(df)} records → {len(grouped\_data)} invoices")

print(f"📄 Output: {pdf\_count} PDFs, {word\_count} Word files")

print(f"✅ Ready for download!")

# Run the system

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

main()