#chunk it

import textwrap

# ✅ API Key - Do not change

api\_key = os.getenv("OPENAI\_API\_KEY") or "api key”

client = openai.OpenAI(api\_key=api\_key)

# ✅ Path to the TXT file containing OCR-extracted text

TEXT\_FILE\_PATH = "/content/PADDLE /201501750.paddle\_good.txt"

# Define Excel columns

COLUMNS = [

"Catalogue Identifier", "Keyword Person", "Artist City",

"Artist Address", "Entry Number", "Free Title",

"Additional Artwork Info", "Asterisk", "Amount Type",

"Currency", "Price", "Full Entry Quote"

]

def extract\_catalogue\_identifier(filename):

"""Extracts the 9-digit catalogue identifier from the filename."""

match = re.search(r"(\d{9})", filename)

return match.group(1) if match else "UNKNOWN"

def read\_txt\_file(txt\_path):

"""Reads extracted text from the specified .txt file."""

try:

with open(txt\_path, "r", encoding="utf-8") as file:

return file.read().strip()

except Exception as e:

print(f"❌ Error reading file {txt\_path}: {e}")

return ""

def split\_text\_into\_chunks(text, max\_chunk\_size=4000):

"""Splits large text into smaller chunks for API processing."""

return textwrap.wrap(text, max\_chunk\_size, break\_long\_words=False, break\_on\_hyphens=False)

def format\_text\_into\_table(raw\_text, txt\_name):

"""Formats extracted text into structured table using OpenAI."""

catalogue\_identifier = extract\_catalogue\_identifier(txt\_name)

text\_chunks = split\_text\_into\_chunks(raw\_text) # ✅ Break large text into smaller batches

extracted\_dfs = []

for chunk\_id, chunk\_text in enumerate(text\_chunks):

print(f"📦 Processing chunk {chunk\_id + 1}/{len(text\_chunks)}...")

try:

prompt = f"""Convert the following Dutch exhibition catalog text into a structured CSV table:

\*\*Rules:\*\*

- \*\*Catalogue Identifier:\*\* Use `{catalogue\_identifier}` for every row.

- \*\*Extract all entities from this chunk.\*\*

- \*\*Keyword Person:\*\* Extract the full name of the artist, formatting it as:

- First Name/Initials, Infixes, Surname.

- Example:

- "VAN GOGH, J. H." → "J. H. Van Gogh"

- "Mevr. Adriana van Haanen" → Keep full title and name.

- "T. van der Grient" → Preserve infixes and initials.

- \*\*Entry Number:\*\* Extract correctly.

- If "unknown", assign it based on order.

- Example: "1,2,3, unknown,5" → Assign "4" to the unknown entry.

- \*\*Include the full entry text exactly as it appears.\*\*

- \*\*Continue entity numbering sequentially across chunks.\*\*

- \*\*Do NOT stop at 65. Process all entries in this batch.\*\*

- \*\*Fill empty cells with `"unknown"`\*\*

\*\*Output Columns:\*\*

{','.join(COLUMNS)}

\*\*Extracted Text (Chunk {chunk\_id + 1}/{len(text\_chunks)}):\*\*

{chunk\_text}

"""

response = client.chat.completions.create(

model="gpt-4-turbo",

messages=[

{"role": "system", "content": "You are a data formatting assistant. Output CSV data only."},

{"role": "user", "content": prompt}

],

temperature=0.1,

max\_tokens=4000

)

if response.choices[0].message.content:

formatted\_data = response.choices[0].message.content.strip()

df = pd.read\_csv(StringIO(formatted\_data), delimiter=",", names=COLUMNS, quotechar='"', on\_bad\_lines="skip")

extracted\_dfs.append(df)

except Exception as e:

print(f"❌ Error processing chunk {chunk\_id + 1}: {e}")

# Combine all extracted data from chunks

if extracted\_dfs:

final\_df = pd.concat(extracted\_dfs, ignore\_index=True)

final\_df = fix\_missing\_entry\_numbers(final\_df) # ✅ Fix numbering issues

return final\_df

return pd.DataFrame(columns=COLUMNS)

def fix\_missing\_entry\_numbers(df):

"""Fills missing or 'unknown' entry numbers based on sequential order."""

df["Entry Number"] = pd.to\_numeric(df["Entry Number"], errors='coerce') # Convert to numbers

df = df.sort\_values(by="Entry Number", na\_position='last').reset\_index(drop=True)

missing\_indices = df[df["Entry Number"].isna()].index # Find missing numbers

if not missing\_indices.empty:

last\_known = df["Entry Number"].dropna().astype(int).tolist()

all\_possible = list(range(1, len(df) + 1)) # Expected sequence

missing\_numbers = [num for num in all\_possible if num not in last\_known]

for idx, missing\_num in zip(missing\_indices, missing\_numbers):

df.at[idx, "Entry Number"] = missing\_num # Fill in missing numbers

df["Entry Number"] = df["Entry Number"].astype(int) # Ensure integer format

return df

def process\_txt\_file(text\_file\_path):

"""Processes the single extracted text file and converts it into structured data."""

extracted\_data = []

print(f"\n📄 Processing TXT File: {text\_file\_path}")

# Read extracted text from .txt file

raw\_text = read\_txt\_file(text\_file\_path)

if not raw\_text:

print(f"⚠️ No text found in {text\_file\_path}")

return

# Format text into structured table

df\_entries = format\_text\_into\_table(raw\_text, os.path.basename(text\_file\_path))

if not df\_entries.empty:

extracted\_data.append(df\_entries)

# Combine all data into one DataFrame

final\_df = pd.concat(extracted\_data, ignore\_index=True) if extracted\_data else pd.DataFrame(columns=COLUMNS)

# Save extracted data to Excel

if not final\_df.empty:

output\_file = "chunked\_paddle.xlsx"

final\_df.to\_excel(output\_file, index=False)

print(f"\n✅ Extraction complete! Data saved to {output\_file}")

else:

print("❌ No data extracted! Check API responses.")

# Run the process

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

process\_txt\_file(TEXT\_FILE\_PATH)