```
# -*- coding: utf-8 -*-
"""FinDoc Analyzer

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1mJo5Dl23sirJQdZ71TfX0ecC5-rYY0Bo
"""

#wget https://digitalassets.tesla.com/tesla-contents/image/upload/IR/TSLA-Q3-2023-Update-3.pdf

import os
print(os.getcwd())

!apt-get install -y poppler-utils

!apt-get install -y tesseract-ocr

pip install -q unstructured[all-docs]==0.11.0 fastapi==0.103.2 kaleido==0.2.1 uvicorn==0.24.0.post1 typing-extensions==4.5.0 pydantic==1.10.13 llama-index

from llama_index.core import SimpleDirectoryReader
from llama_index.readers.file import PDFReader

parser = PDFReader()
file_extractor = {".pdf": parser}
documents = SimpleDirectoryReader(
    "/content/financial Document", file_extractor=file_extractor
).load_data()

print(len(documents))

from unstructured.partition.pdf import partition_pdf

raw_pdf_elements = partition_pdf(
    filename="./financial Document/TSLA-Q3-2023-Update-3.pdf",
    # Use layout model (YOLOX) to get bounding boxes (for tables) and find titles
    # Titles are any sub-section of the document
    infer_table_structure=True,
    # Post processing to aggregate text once we have the title
    chunking_strategy="by_title",
    # Chunking params to aggregate text blocks
    # Attempt to create a new chunk 3800 chars
    # Attempt to keep chunks > 2000 chars
    # Hard max on chunks
    max_characters=4000,
    new_after_n_chars=3800,
    combine_text_under_n_chars=2000
)

from pydantic import BaseModel
from typing import Any

class Element(BaseModel):
  type: str
  text: Any

categorized_elements = []
for element in raw_pdf_elements:
    if "unstructured.documents.elements.Table" in str(type(element)):
        categorized_elements.append(Element(type="table", text=str(element)))
    elif "unstructured.documents.elements.CompositeElement" in str(type(element)):
        categorized_elements.append(Element(type="text", text=str(element)))

!pip install -q pdf2image==1.16.3

from pdf2image import convert_from_path

os.mkdir('./pages')
convertor = convert_from_path('/content/financial Document/TSLA-Q3-2023-Update-3.pdf')

for idx, image in enumerate( convertor ):
    image.save(f"./pages/page-{idx}.png")

pages_png = [file for file in os.listdir("./pages") if file.endswith('.png')]

headers = {
  "Content-Type": "application/json",
  "Authorization": "Bearer " + str( os.environ["OPENAI_API_KEY"] )
}

payload = {
  "model": "gpt-4-vision-preview",
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "type": "text",
          "text": "You are an assistant that find charts, graphs, or diagrams from an image and summarize their information. There could be multiple diagrams i
        },
        {
          "type": "text",
          "text": 'The response must be a JSON in following format {"graphs": [<chart_1>, <chart_2>, <chart_3>]} where <chart_1>, <chart_2>, and <chart_3> plac
        },
        {
          "type": "text",
          "text": 'If could not find a graph in the image, return an empty list JSON as follows: {"graphs": []}. Do not append or add anything other than the J
        },
        {
          "type": "text",
          "text": "Look at the attached image and describe all the graphs inside it in JSON format. ignore tables and be concise."
        }
      ]
    }
  ],
  "max_tokens": 1000
```

```
}

# Function to encode the image to base64 format
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')

pip install tqdm

from tqdm import tqdm
import base64
import copy  # Import the copy module
import json
import requests

graphs_description = []
for idx, page in tqdm( enumerate( pages_png ) ):
  # Getting the base64 string
  base64_image = encode_image(f"./pages/{page}")

  # Adjust Payload
  tmp_payload = copy.deepcopy(payload)
  tmp_payload['messages'][0]['content'].append({
    "type": "image_url",
    "image_url": {
      "url": f"{base64_image}"
    }
  })

  try:
    response = requests.post("https://api.openai.com/v1/chat/completions", headers=headers, json=tmp_payload)
    response = response.json()
    graph_data = json.loads( response['choices'][0]['message']['content'] )['graphs']

    desc = [f"{page}\n" + '\n'.join(f"{key}: {item[key]}" for key in item.keys()) for item in graph_data]

    graphs_description.extend( desc )

  except:
    # Skip the page if there is an error.
    print("skipping... error in decoding.")
    continue;

graphs_description = [Element(type="graph", text=str(item)) for item in graphs_description]

all_docs = categorized_elements + graphs_description

print( len( all_docs ) )

!pip install -q llama_index deeplake cohere

!pip install langchain

import os
os.environ['OPENAI_API_KEY'] = ''
os.environ['ACTIVELOOP_TOKEN'] =''

#index.storage_context.persist()

#from llama_index.core import BaseQueryEngine

# Commented out IPython magic to ensure Python compatibility.
# %pip install llama-index-vector-stores-deeplake

from llama_index.vector_stores.deeplake import DeepLakeVectorStore

my_activeloop_org_id = "ihamzakhan89"
my_activeloop_dataset_name = "tsla_q3"
dataset_path = f"hub://{my_activeloop_org_id}/{my_activeloop_dataset_name}"

vector_store = DeepLakeVectorStore( dataset_path=dataset_path,
                   runtime={"tensor_db": True},
                   overwrite=False)

!pip install storage

from llama_index.core import StorageContext

storage_context = StorageContext.from_defaults(vector_store=vector_store)

from llama_index.core import Document

documents = [Document(text=t.text, metadata={"category": t.type},) for t in categorized_elements]

from llama_index.core import VectorStoreIndex

index = VectorStoreIndex.from_documents(
    documents, storage_context=storage_context
)

query_engine = index.as_query_engine()
response = query_engine.query(
    "What are the trends in vehicle deliveries?",
)

print(response.response)

#Applying Prompt Engineering
from IPython.display import Markdown, display
def display_prompt_dict(prompts_dict):
    for k, p in prompts_dict.items():
        text_md = f"**Prompt Key**: {k}<br>" f"**Text:** <br>"
        display(Markdown(text_md))
        print(p.get_template())
        display(Markdown("<br><br>"))
```

```
prompts_dict = query_engine.get_prompts()

display_prompt_dict(prompts_dict)

from langchain import hub

langchain_prompt = hub.pull("rlm/rag-prompt")

!pip install langchainhub

from llama_index.core.prompts import LangchainPromptTemplate

lc_prompt_tmpl = LangchainPromptTemplate(
    template=langchain_prompt,
    template_var_mappings={"query_str": "question", "context_str": "context"},
)

query_engine.update_prompts(
    {"response_synthesizer:text_qa_template": lc_prompt_tmpl})

prompts_dict = query_engine.get_prompts()
display_prompt_dict(prompts_dict)

!pip install llama_index.llms.langchain

# Commented out IPython magic to ensure Python compatibility.
# %pip install llama-index-llms-openai

response = query_engine.query(
    "What are the trends in vehicle deliveries?",
)
print(str(response))



#pip install llama_index==0.9.8 deeplake
#!pip install llama-index==0.10.11
```