**PDF Parsing into Structured JSON Extraction**

1. **Problem Statement:**

Building a Python program that parses a PDF file and extracts its content into a well-structured JSON format. The extracted JSON must preserve the hierarchical organization of the document (e.g., sections, sub-sections, and content blocks) while clearly identifying different data types.

1. **Dependencies to Install:**

Install Python 3.8+ and install the required libraries:

* **Pdfplumber:** Extracts text, characters, and tables from PDFs.
* **fitz (PyMuPDF):** Handles images/charts detection inside PDFs.
* **Json:** Saves extracted content in structured JSON format.
* **Tkinter:** GUI dialogs for file selection and saving.

**Code:**

!pip install pdfplumber PyMuPDF Tkinter

1. **How to Run?**
2. Install Python version 3.8+ from the official website.
3. Install the dependencies using the code above and restart the kernel.
4. Save the below python script as ‘pdf\_to\_json.py’.
5. Run the below script:

‘python pdf\_to\_json.py’

1. A dialog box will open, where select the desired PDF file to parse and select the location to save the JSON file.
2. Once processed, you will see a success message with the JSON file path.
3. **Python Script:**

import pdfplumber

import fitz

import json

import tkinter as tk

from tkinter import filedialog, messagebox

def extract\_sections(page):

lines = {}

for char in page.chars:

y0 = round(char['top'], 1)

lines.setdefault(y0, []).append(char)

line\_infos = []

for y0 in sorted(lines.keys()):

chars = lines[y0]

text = ''.join(c['text'] for c in chars).strip()

if not text:

continue

font\_sizes = [c['size'] for c in chars]

max\_font\_size = max(font\_sizes)

is\_bold = any('Bold' in c['fontname'] or 'bold' in c['fontname'] for c in chars)

line\_infos.append({

'text': text,

'font\_size': max\_font\_size,

'is\_bold': is\_bold,

'y0': y0

})

return line\_infos

def classify\_sections(line\_infos):

if not line\_infos:

return []

font\_sizes = sorted(set([line['font\_size'] for line in line\_infos]), reverse=True)

section\_size = font\_sizes[0] if len(font\_sizes) > 0 else None

sub\_section\_size = font\_sizes[1] if len(font\_sizes) > 1 else None

classified = []

for line in line\_infos:

if line['font\_size'] == section\_size and line['is\_bold']:

classified.append({'type': 'section', 'text': line['text'], 'y0': line['y0']})

elif sub\_section\_size and line['font\_size'] == sub\_section\_size and line['is\_bold']:

classified.append({'type': 'sub\_section', 'text': line['text'], 'y0': line['y0']})

else:

classified.append({'type': 'paragraph', 'text': line['text'], 'y0': line['y0']})

return classified

def extract\_tables(page):

tables = []

for table in page.extract\_tables():

clean\_table = []

for row in table:

clean\_row = [cell.strip() if cell else "" for cell in row]

clean\_table.append(clean\_row)

tables.append(clean\_table)

return tables

def extract\_charts(doc, page\_number):

charts = []

page = doc.load\_page(page\_number - 1)

images = page.get\_images(full=True)

for img in images:

xref = img[0]

charts.append({

"description": "Chart/image detected on page",

"image\_xref": xref

})

return charts

def parse\_pdf(pdf\_path):

result = {"pages": []}

with pdfplumber.open(pdf\_path) as pdf:

doc = fitz.open(pdf\_path)

for i, page in enumerate(pdf.pages, start=1):

page\_content = []

line\_infos = extract\_sections(page)

classified = classify\_sections(line\_infos)

current\_section = None

current\_sub\_section = None

paragraph\_buffer = []

def flush\_paragraph():

nonlocal paragraph\_buffer

if paragraph\_buffer:

text = ' '.join(paragraph\_buffer)

page\_content.append({

"type": "paragraph",

"section": current\_section,

"sub\_section": current\_sub\_section,

"text": text

})

paragraph\_buffer = []

for item in classified:

if item['type'] == 'section':

flush\_paragraph()

current\_section = item['text']

current\_sub\_section = None

elif item['type'] == 'sub\_section':

flush\_paragraph()

current\_sub\_section = item['text']

else:

paragraph\_buffer.append(item['text'])

flush\_paragraph()

tables = extract\_tables(page)

for table in tables:

page\_content.append({

"type": "table",

"section": current\_section,

"description": None,

"table\_data": table

})

charts = extract\_charts(doc, i)

for chart in charts:

page\_content.append({

"type": "chart",

"section": current\_section,

"description": chart.get("description"),

"table\_data": None

})

result["pages"].append({

"page\_number": i,

"content": page\_content

})

return result

def select\_pdf\_and\_save\_json():

root = tk.Tk()

root.withdraw()

pdf\_path = filedialog.askopenfilename(

title="Select PDF file",

filetypes=[("PDF files", "\*.pdf")]

)

if not pdf\_path:

messagebox.showinfo("Cancelled", "No PDF file selected. Exiting.")

return

try:

data = parse\_pdf(pdf\_path)

except Exception as e:

messagebox.showerror("Error", f"Failed to extract PDF content:\n{e}")

return

json\_path = filedialog.asksaveasfilename(

title="Save JSON file",

defaultextension=".json",

filetypes=[("JSON files", "\*.json")]

)

if not json\_path:

messagebox.showinfo("Cancelled", "No save location selected. Exiting.")

return

try:

with open(json\_path, "w", encoding="utf-8") as f:

json.dump(data, f, indent=4, ensure\_ascii=False)

messagebox.showinfo("Success", f"JSON saved successfully:\n{json\_path}")

except Exception as e:

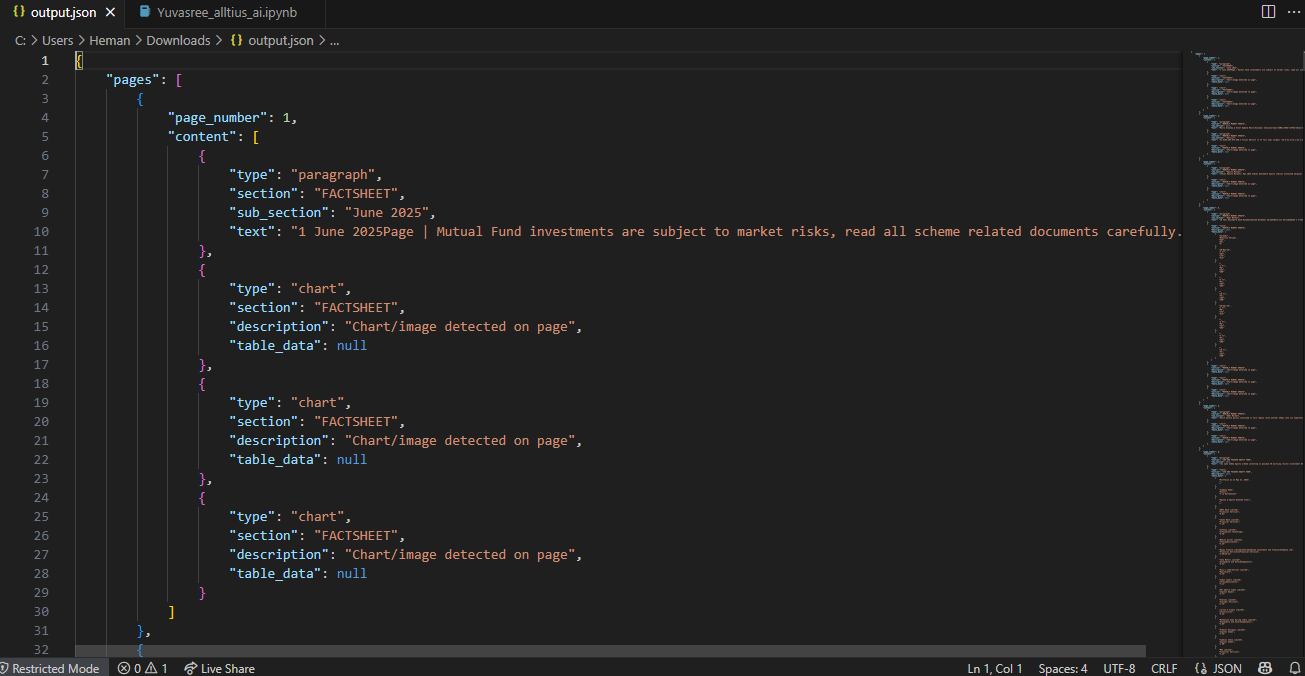
messagebox.showerror("Error", f"Failed to save JSON file:\n{e}")

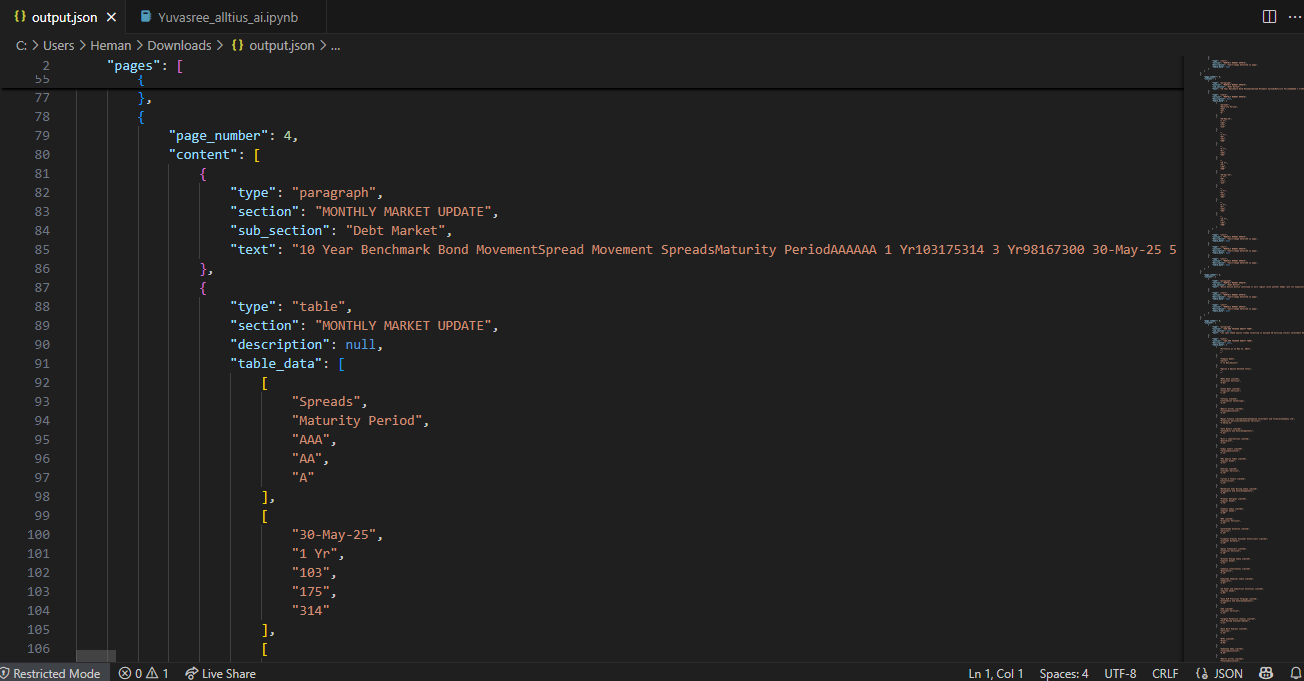
if \_\_name\_\_ == "\_\_main\_\_":

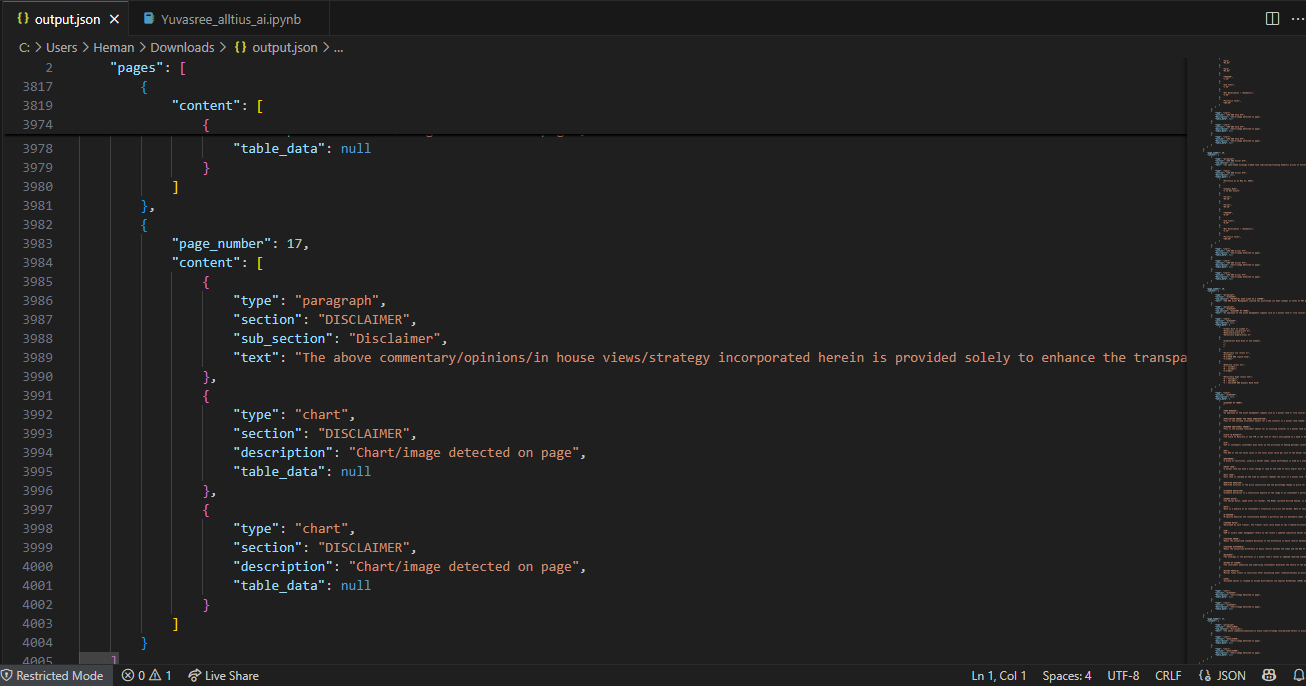
select\_pdf\_and\_save\_json()

1. **JSON Output:**

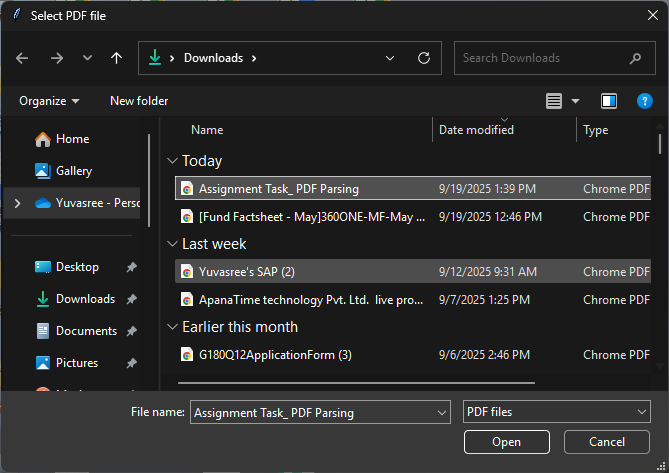
‘output.json’ file is already uploaded. Sample images are:



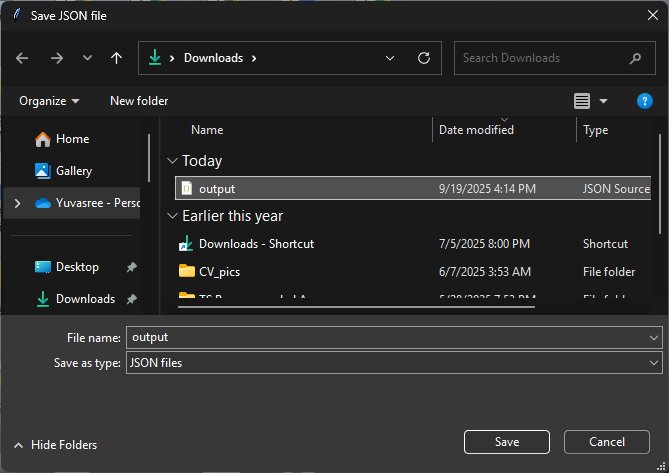




This is where I uploaded the file with the Tkinter message window-



This is where I saved the JSON File-



1. **Key Components:**
2. **‘extract\_sections(page)’**

This function helps to read all text characters from the page and groups them into lines by their vertical position (y0). It identifies text content, font size and bold content and returns the structured line information.

1. **‘classify\_sections(line\_infos)’**

This function determines the section headings and sub-sections based on font size:

* Largest= section
* Second largest= sub-section
* Least= Paragragh
* Bold formatting

1. **‘extract\_tables(page)’**

This function uses ‘pdfplumber’s’ built-in table extraction. Also, cleans and organizes the table rows while returning structured table data.

1. **‘extract\_charts(doc, page\_number)’**

This function uses ‘PyMuPDF (fitz)’ to find embedded images on a page. Each detected image is treated as a chart and returns the list of chart descriptions.

1. **‘parse\_pdf(pdf\_path)’**

This function coordinates extraction from each page such as text sections, tables and charts. Later, it returns the structures JSON object with per-page content.

1. **‘select\_pdf\_and\_save\_json()’**

Using Tkinter with GUI interface:

We can select a PDF file. Later it parses and extract the content. Now, we need to choose a save location for JSON file and we will get the output in that JSON file.

1. **Main Execution**

When the script is executed manually, it triggers Tkinter GUI interface and we need to upload the PDF file to start the workflow. Now, the other functions help to parse the document and we can save the JSON file wherever we want to.