**Leveraging RAG (Retrieval-Augmented Generation) for Excel Spreadsheets**

Retrieval-Augmented Generation (RAG) is revolutionizing the way we interact with data by combining retrieval-based search with generative AI. When paired with Excel, this approach unlocks powerful insights, allowing users to query, analyze, and generate responses based on structured and unstructured data.

**Why Use RAG with Excel?**

Excel is a widely used tool for data management, but it lacks advanced AI-driven capabilities for contextual search and synthesis. By integrating RAG using **LlamaIndex**, we can bridge this gap, enabling users to:

* Retrieve relevant data from large Excel sheets quickly.
* Generate natural language summaries and insights.
* Answer complex queries based on structured tables and external knowledge sources.

**How LlamaIndex Enhances Excel with RAG**

LlamaIndex serves as the data framework that connects Excel spreadsheets with LLMs. Here’s how it works:

1. **Ingesting Data** — Excel sheets are loaded into LlamaIndex, converting them into query able indices.
2. **Retrieval Layer** — When a user inputs a query, LlamaIndex fetches the most relevant data from the Excel sheet.
3. **Generation with Context** — The retrieved data is passed to an LLM, which generates an informed response.

**Getting Started**

To implement this, you can use Python with libraries like llama-index,llama-parse, pandas, and openpyxl to load Excel data, create an index, and enable intelligent querying.

**API Keys Setup**

I will be using APIs from **OpenAI** (GPT-4o model) and **LlamaCloud**. You can generate your API keys by creating accounts:

* **LlamaIndex**: [Llama Cloud](https://cloud.llamaindex.ai/)
* **OpenAI**: [OpenAI Platform](https://platform.openai.com/)

The complete Python notebook can be found here: [GitHub Repository](https://github.com/deepikakaush/AI/tree/main)

I am using the publicly available dataset **Food Sales**: [Download Dataset](https://www.contextures.com/excelsampledatafoodsales.html)

**Explanation of Various Elements in the Code**

* **MarkdownElementNodeParser**: In LlamaIndex, this parser processes and structures markdown-formatted text into nodes that can be indexed and retrieved efficiently. It is particularly useful for structured content such as headings, tables, lists, and code blocks.
* **VectorStoreIndex**: A crucial component that enables efficient semantic search within data sources like Excel spreadsheets. It transforms text-based data into vector embeddings, making it possible to retrieve relevant information using similarity-based search.
* **top\_k Parameter**: I have set top\_k = 3 to have a more focused response.

**Code Implementation**

*Importing Required Packages*

#Importing the packages  
from llama\_index.llms.openai import OpenAI  
from llama\_index.core import VectorStoreIndex  
from llama\_parse import LlamaParse  
from llama\_index.core.node\_parser import MarkdownElementNodeParser  
import pandas as pd  
import os  
import nest\_asyncio  
nest\_asyncio.apply()

*Setting API Keys for OpenAI and LlamaParse*

#Loading the API Keys from OpenAI and Llamacloud  
os.environ["OPENAI\_API\_KEY"]='sk..'  
llm\_gpt4o = OpenAI(model="gpt-4o")  
parser=LlamaParse(api\_key='llx..',result\_type="markdown",)

*Parsing the excel spreadsheet*

documents = parser.load\_data("sampledatafoodsales.xlsx")

*View the content of the document*

print(documents[0].get\_content())

*Parsing the sheet*

node\_parser = MarkdownElementNodeParser(llm=llm\_gpt4o, num\_workers=4)  
nodes = node\_parser.get\_nodes\_from\_documents(documents[:5])

Dumping Indexed Table into Vector Index for Retrieval

recursive\_index = VectorStoreIndex(nodes=base\_nodes + objects, llm=OpenAI(model="gpt-4o"))  
  
  
recursive\_query\_engine\_gpt4o= recursive\_index.as\_query\_engine(  
 similarity\_top\_k=5, llm=llm\_gpt4o  
)

*Here are we finally prompting the pull the required data basis our query*

query = "Calculate the total price of product Potato Chips on 20 March 2024. Give reason?"  
response\_recursive\_gpt4o = recursive\_query\_engine\_gpt4o.query(query)  
print("Here is the output from our GPT model")  
display((f"{response\_recursive\_gpt4o}"))

**Conclusion**

This tutorial demonstrates how to integrate **Retrieval-Augmented Generation (RAG)** with **Excel Spreadsheets** using **LlamaIndex** and **GPT-4o** for intelligent data retrieval and analysis. This approach enhances Excel’s capabilities, making it a more powerful tool for d**ata-driven decision-making**.