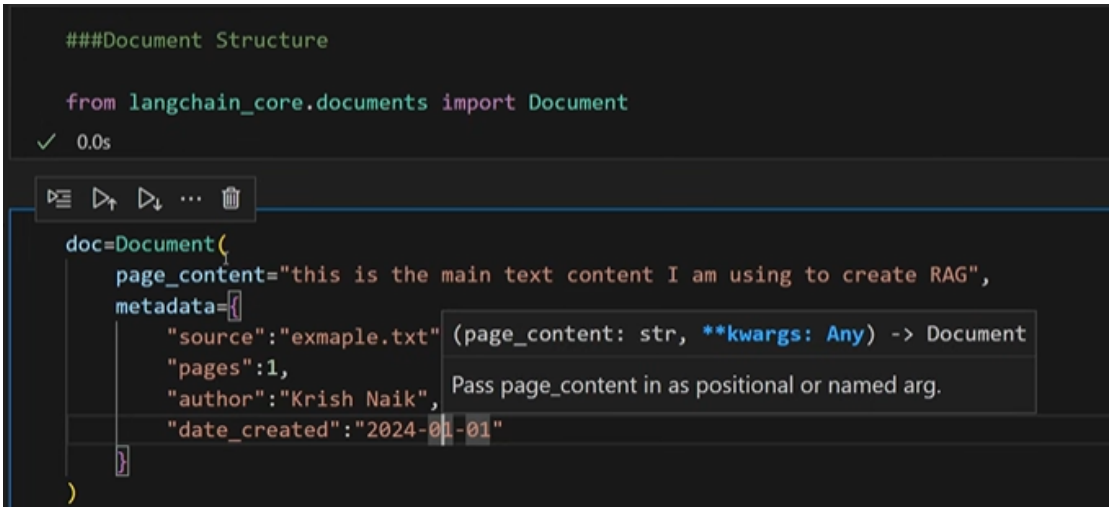
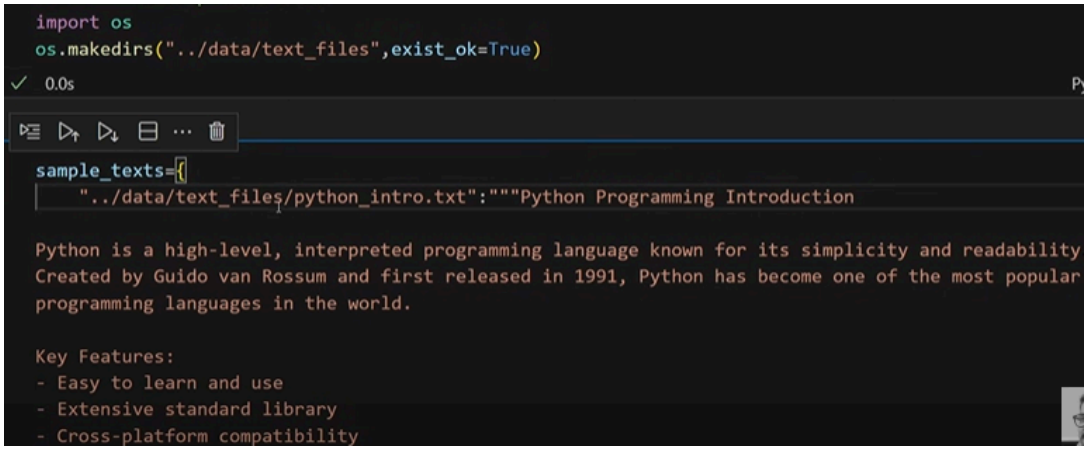
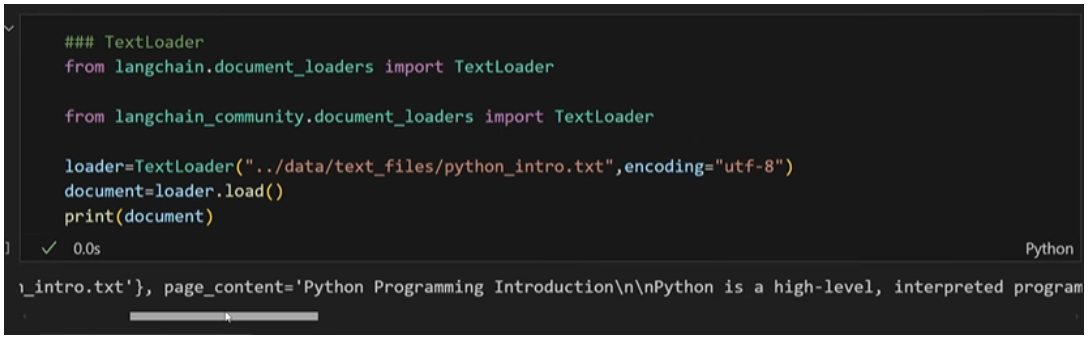
Basic of RAG



here the document looks like this, that contains the page_contents, metadata which contains the source pages etc . this meta data helps for thw filtering of the data from other data , so retrival may be fast. means if u ask what is the source content written by " Krish Naik " , then it filters all the krishnaik page docs



creating the manual text for the files , files are creating using the os.makedirs



so using the langchain TextLoader , we are reading the manual text we created in .txt file

```
## load all the text files from the directory
dir_loader=DirectoryLoader(
    "../data/text_files",
    glob="**/*.txt", ## Pattern to match files
    loader_cls= TextLoader, ##loader class to use
    loader_kwargs={'encoding': 'utf-8'},
    show_progress=False

)

documents=dir_loader.load()
documents
```

```
[Document(metadata={'source': '..\\data\\text_files\\machine_learning.txt'}, page_content='Machine Learn
 Document(metadata={'source': '..\\data\\text_files\\python_intro.txt'}, page_content='Python Programmin
```

same like above here also we are reading the text from .txt files , but all the *.txt

```
from langchain_community.document_loaders import PyPDFLoader, PyMuPDFLoader

## load all the text files from the directory
dir_loader=DirectoryLoader(
    "../data/pdf",
    glob="**/*.pdf", ## Pattern to match files
    loader_cls= PyMuPDFLoader, ##loader class to use
    show_progress=False

)

pdf_documents=dir_loader.load()
pdf_documents
```

here reading the data from the PDF

```
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
Document(metadata={'producer': 'pdfTeX-1.40.25', 'creator': 'LaTeX with hyperref', 'creationdate': '2024
```

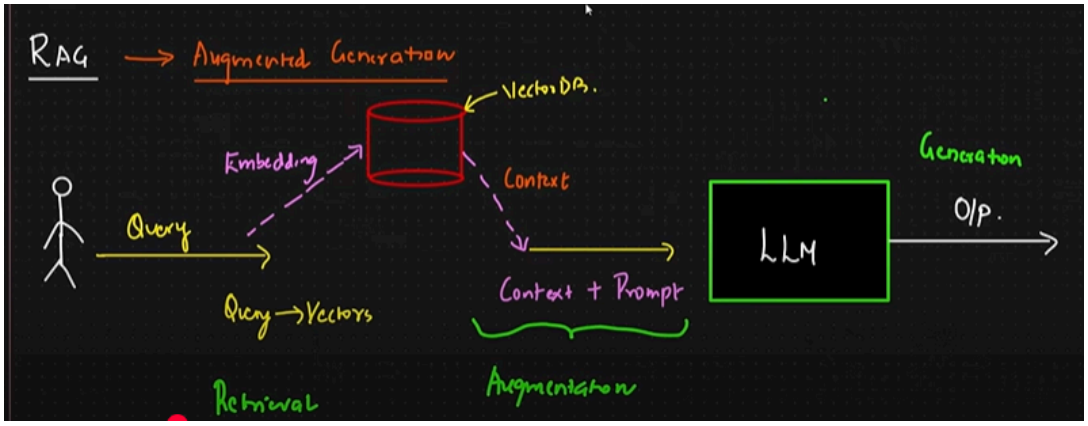
the data looks likes this , here it creates like “data content” and “metadata”

1. then we need to create the embeddings using any openAi/HuggingFace model
2. build vectorStore either using Pinecone,ChromaDb,Fiass
3. then RAG retriever

a. the query we give will be converted to embeddings

b. then similarity search score

c. if similarity search score is greater than threshold then retrive that as ans



→ RAGS implementation types

1. we can do RAG using the normal python functions , but large code
2. using simple RAG pipeline

```
### Simple RAG pipeline with Groq LLM
from langchain_groq import ChatGroq
import os
from dotenv import load_dotenv
load_dotenv()

### Initialize the Groq LLM (set your GROQ_API_KEY in environment)
groq_api_key = "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"

llm=ChatGroq(groq_api_key=groq_api_key,model_name="gemma2-9b-it",temperature=0.1,max_tokens=1024)

## 2. Simple RAG function: retrieve context + generate response
def rag_simple(query,retriever,llm,top_k=3):
    ## retriever the context
    results=retriever.retrieve(query,top_k=top_k)
    context="\n\n".join([doc['content'] for doc in results]) if results else ""
    if not context:
        return "No relevant context found to answer the question."

    ## generate the answwer using GROQ LLM
    prompt=f"""Use the following context to answer the question concisely.
```

- loading the groq llm model , used its API key
- we are retrieving the context first (simple Retrieve RAG)

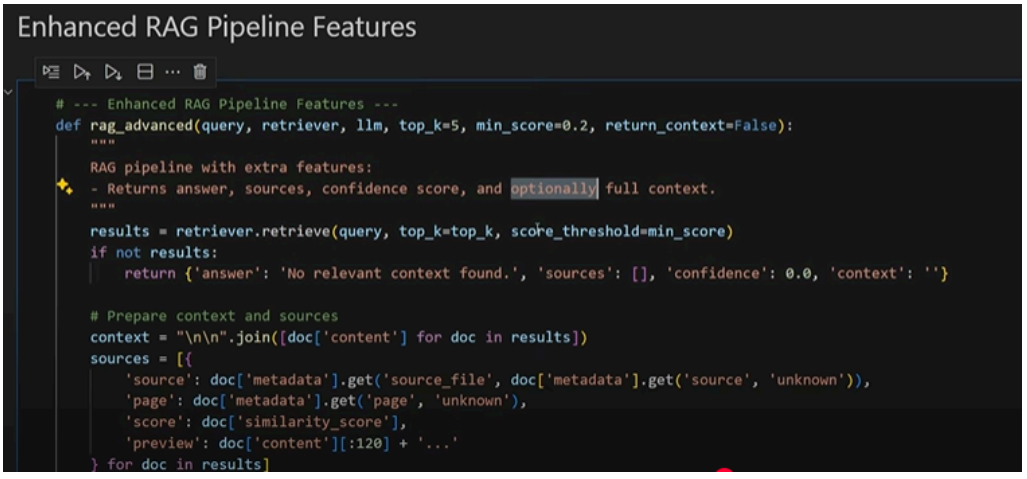
```
## generate the answwer using GROQ LLM
prompt=f"""Use the following context to answer the question concisely.
    Context:
    {context}

    Question: {query}

    Answer:"""

response=llm.invoke([prompt.format(context=context,query=query)])
return response.content
```

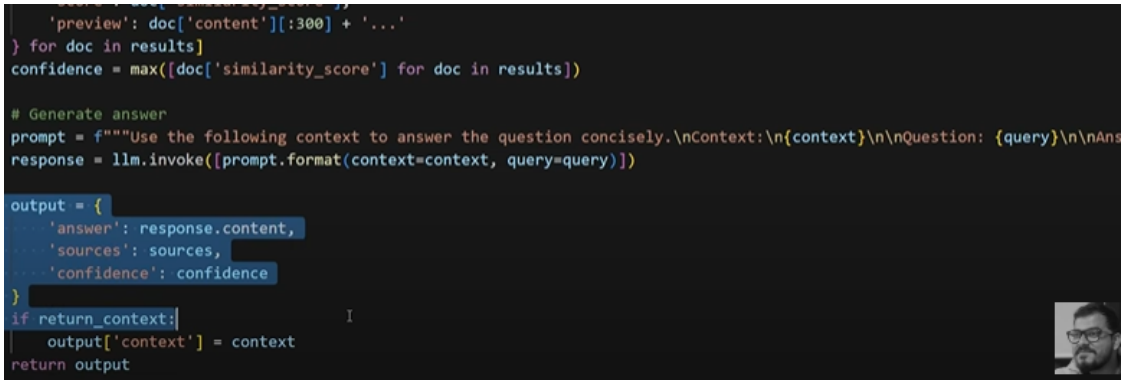
- then we gave the prompt inside that function only
- so when the quaetion is asked it sends to the function , there some answers are retrieved from the vector database
- and then sends that to the llm model to get the correct o/p based on the retrievers o/p

```
answer=rag_simple("What is attention mechanism?",rag_retriever,llm)
print(answer)
✓ 0.4s

Retrieving documents for query: 'What is attention mechanism?'
Top K: 3, Score threshold: 0.0
Generating embeddings for 1 texts...
Batches: 100%|██████████| 1/1 [00:00<00:00, 76.39it/s]
Generated embeddings with shape: (1, 384)
Retrieved 3 documents (after filtering)

An attention mechanism is a function that maps a query and a set of key-value pairs to an output vector, using a weighted
```

we are calling the function with the query as the parameter send to the function where we have retriever,LLM model

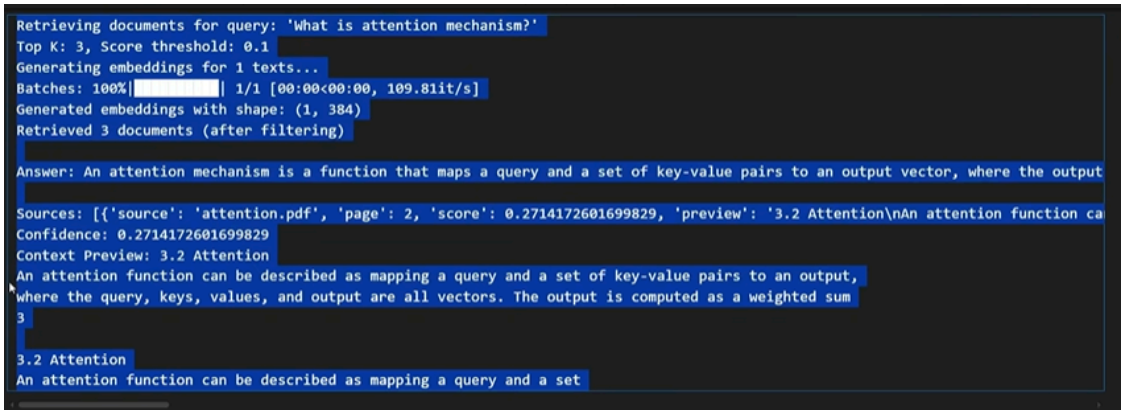
3. Enhanced RAG Pipeline Features





Defining the RAG and LLM model function

this result contains the answer(what we get after passing the context to LLM model) , sources,confidence score , context that psses through LLM



actually in second type we saw the simple code of the RAG where we get the contents only, but here we tried to get the Answer as well as the “Sources, Confidence Score” also

4. TypeSense-Lightning Fast
    - https://www.youtube.com/watch?v=MMS04bku3FE&list=PLZoTAELRMXVM8Pf4U67L4UuDRgV4TNX9D&index=4
    - this is the link of the video

- here instead of using the PineCone/ChromaDb/Fiass we used Typesense which has ability for fast retrieval of the data , It is a Open source model only , here we need to create a collections of data