Lang Chain - Retrieval

Venkata Reddy AI Classes

https://www.youtube.com/@VenkataReddyAIClasses/playlists

Contents

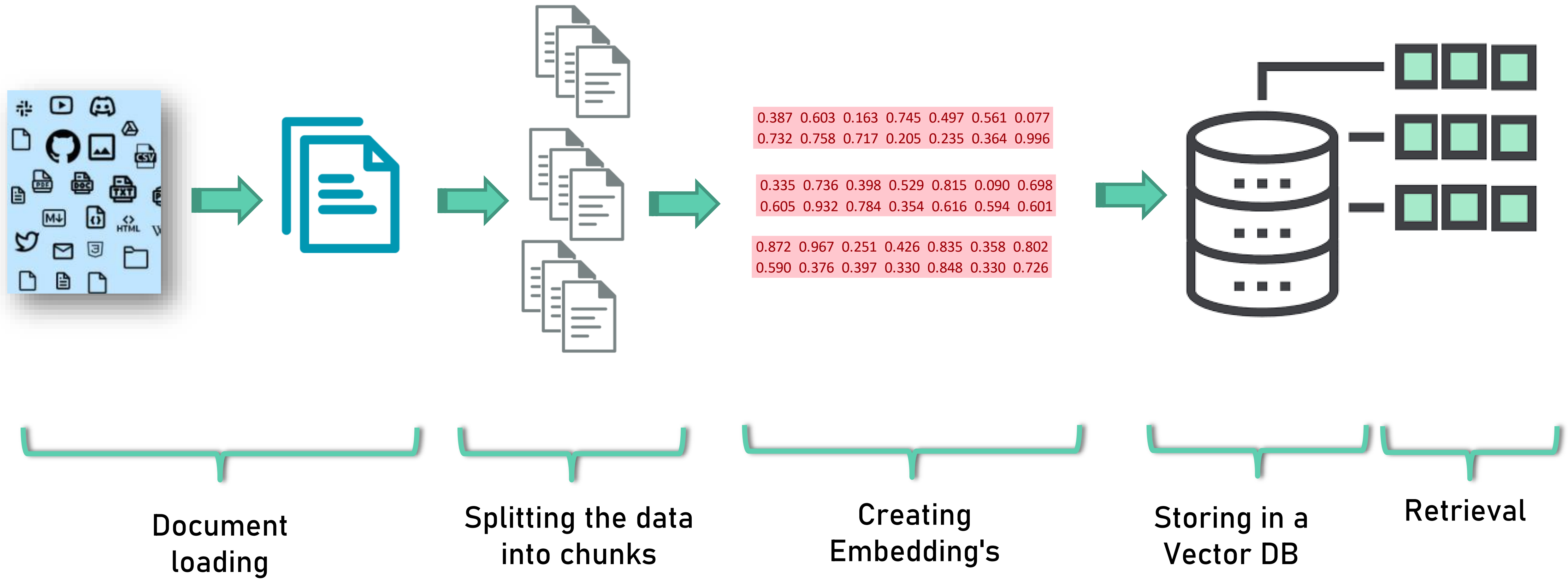
Information Retrieval from Documents

- LLM applications often need user-specific data.
- Retrieval Augmented Generation (RAG) enables this.
- External data is retrieved for LLM's generation step.
- LangChain supports all RAG application needs.
- RAG has 5 Steps

RAG has 5 Steps

1. Data Loading
2. Split the data into Chunks
3. Creating embedding's
4. Storing in Vector Stores
5. Finally Retrieve the relevant information.

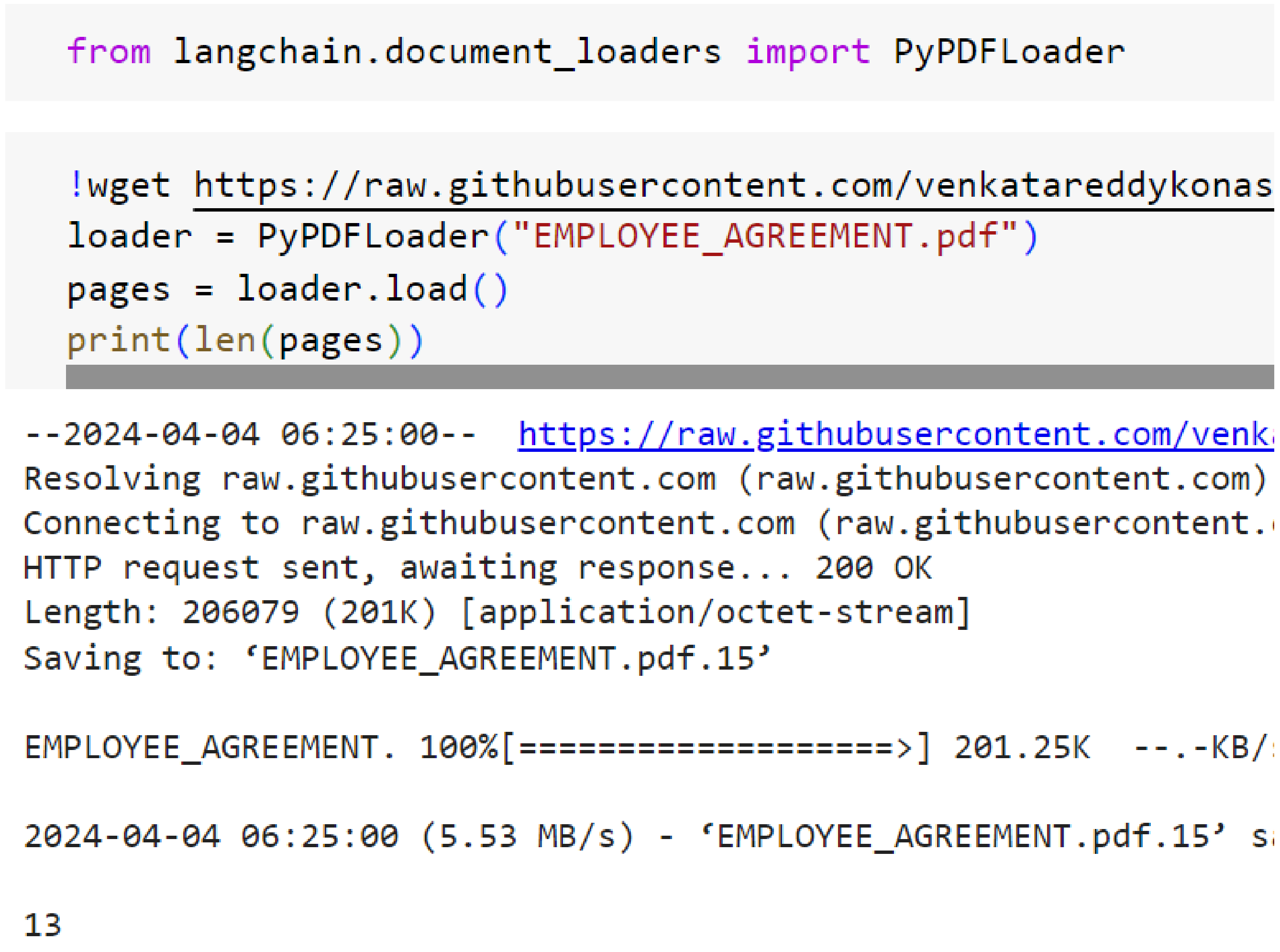
5 Steps in RAG



Step-1: Document Loading

- Document loaders import documents from various sources.
- LangChain offers over 100 different document loaders.
- Integrates with major providers such as AirByte and Unstructured.
- Supports loading various document types like HTML, PDF, and code.
- Capable of importing from diverse locations, including private S3 buckets and public websites.

Document Loading



```python
from langchain.document_loaders import PyPDFLoader
```

```python
!wget https://raw.githubusercontent.com/venkatareddykonas
loader = PyPDFLoader("EMPLOYEE_AGREEMENT.pdf")
pages = loader.load()
print(len(pages))
```

```
--2024-04-04 06:25:00--  https://raw.githubusercontent.com/venk
Resolving raw.githubusercontent.com (raw.githubusercontent.com)
Connecting to raw.githubusercontent.com (raw.githubusercontent.
HTTP request sent, awaiting response... 200 OK
Length: 206079 (201K) [application/octet-stream]
Saving to: ‘EMPLOYEE_AGREEMENT.pdf.15’

EMPLOYEE_AGREEMENT. 100%[===================>] 201.25K  --.-KB/

2024-04-04 06:25:00 (5.53 MB/s) - ‘EMPLOYEE_AGREEMENT.pdf.15’ s

13
```

Document Loading

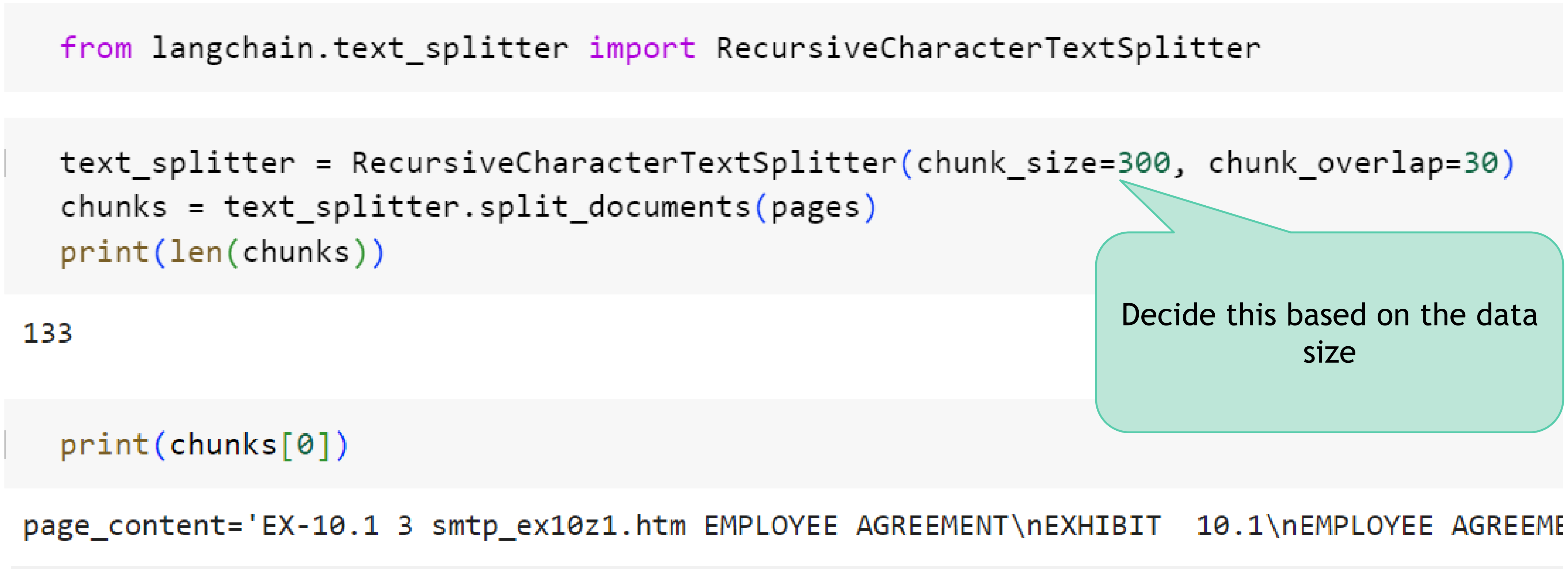
```
full_text =""
for page in pages:
    full_text += page.page_content

print("Pages", len(pages))
print("Lines" , len(full_text.split("\n")))
print("Words" , len(full_text.split(" ")))
print("Charecters", len(full_text))
```

```
Pages 13
Lines 369
Words 4481
Charecters 29817
```

Step-2: Split the data into Chunks

- Retrieval involves fetching relevant document parts.
- Documents undergo transformation steps before retrieval.
- Splitting documents into smaller chunks is a primary step.
- LangChain offers various algorithms for document splitting.
- Provides optimized logic for specific document types, like code and markdown.

Split the data into Chunks



```python
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

```python
text_splitter = RecursiveCharacterTextSplitter(chunk_size=300, chunk_overlap=30)
chunks = text_splitter.split_documents(pages)
print(len(chunks))
```

```
133
```

```python
print(chunks[0])
```

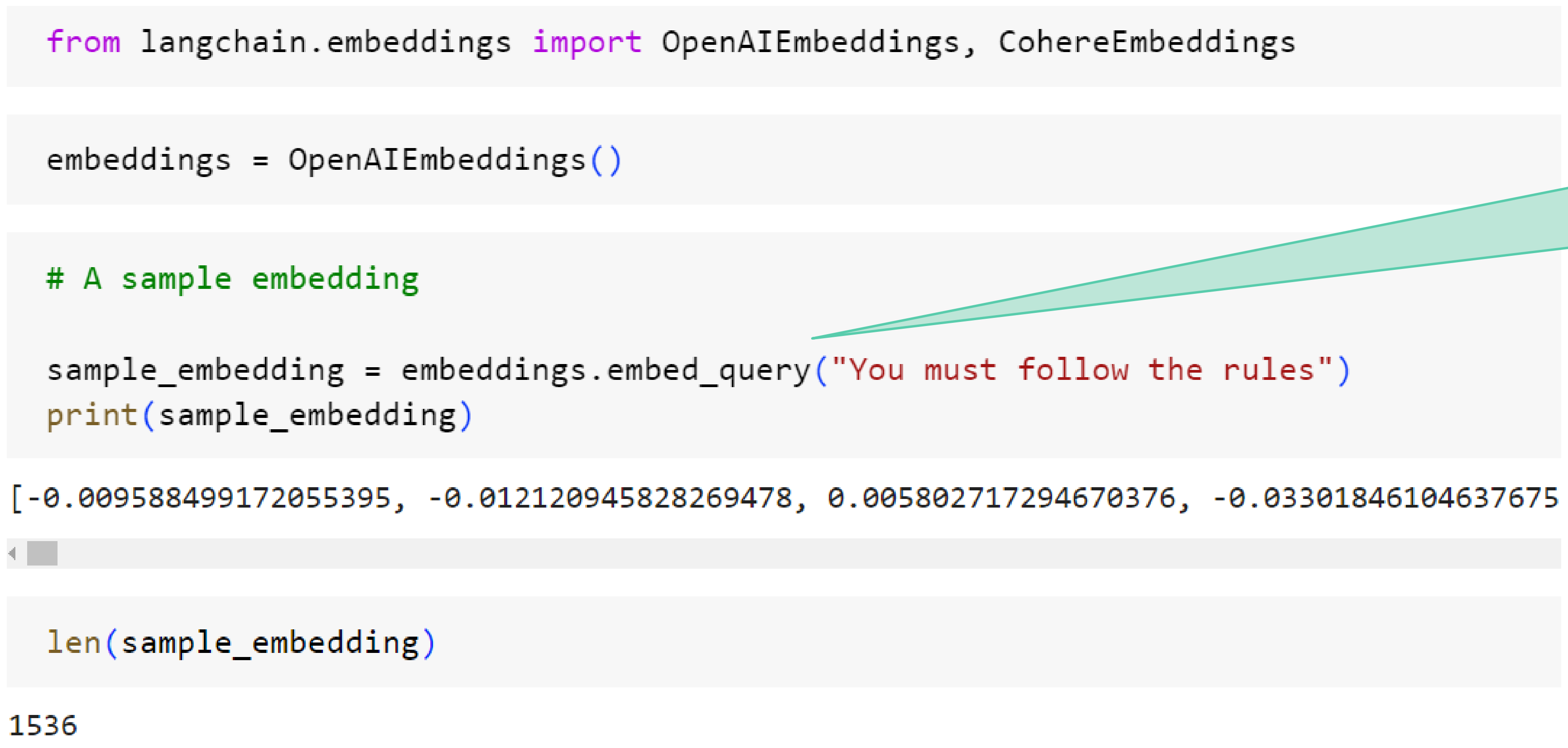
```
page_content='EX-10.1 3 smtp_ex10z1.htm EMPLOYEE AGREEMENT\nEXHIBIT  10.1\nEMPLOYEE AGREEME
```

Decide this based on the data size

Step-3: Creating embedding's

- Embedding's capture the semantic meaning of texts for efficient retrieval.
- LangChain integrates with over 25 embedding providers and methods.
- Offers both open-source and proprietary API options for embeddings.
- Provides a standard interface for easy swapping between embedding models.

Creating embedding's



```
from langchain.embeddings import OpenAIEmbeddings, CohereEmbeddings
```

```
embeddings = OpenAIEmbeddings()
```

```
# A sample embedding

sample_embedding = embeddings.embed_query("You must follow the rules")
print(sample_embedding)
```

```
[-0.009588499172055395, -0.012120945828269478, 0.005802717294670376, -0.03301846104637675
```

```
len(sample_embedding)
```

```
1536
```

Embed a single piece of text for the purpose of comparing to other embedded pieces of texts.

Creating embedding's

- The Embeddings class interfaces with various text embedding models.
- It's designed to standardize access across providers like OpenAI and Hugging Face.
- Embeddings represent text as vectors, facilitating semantic searches.
- The base class in LangChain includes methods for embedding documents and queries.
- Embedding documents and queries are distinct to cater to provider-specific methods.

Creating embedding's

```
sample_docs=["You must follow the rules",
             "You must not disclose the rules"]

embeded_vectors1= embeddings.embed_documents(sample_docs)
print(embeded_vectors1)
```

```
[[-0.009588499172055395, -0.012120945828269478, 0.005802717294670376,
```

```
print(sample_embedding)
print(embeded_vectors1[0])
```

```
[-0.009588499172055395, -0.012120945828269478, 0.005802717294670376, -0.03301846104637675,
[-0.009588499172055395, -0.012120945828269478, 0.005802717294670376, -0.03301846104637675,
```

Step-4: Storing in Vector DB

- Embeddings require databases for efficient storage and search.
- LangChain integrates with over 50 vector stores, both open-source and proprietary.
- Offers options from local to cloud-hosted vector stores.
- Provides a standard interface for easy swapping between vector stores.

Storing in Vector DB

```
!pip install chromadb -q
```

```
from langchain.vectorstores import Chroma
```

```
emp_rules_db= Chroma.from_documents(chunks,
                                    embeddings,
                                    persist_directory="emp_rules_db"
                                 )
emp_rules_db.persist()
```

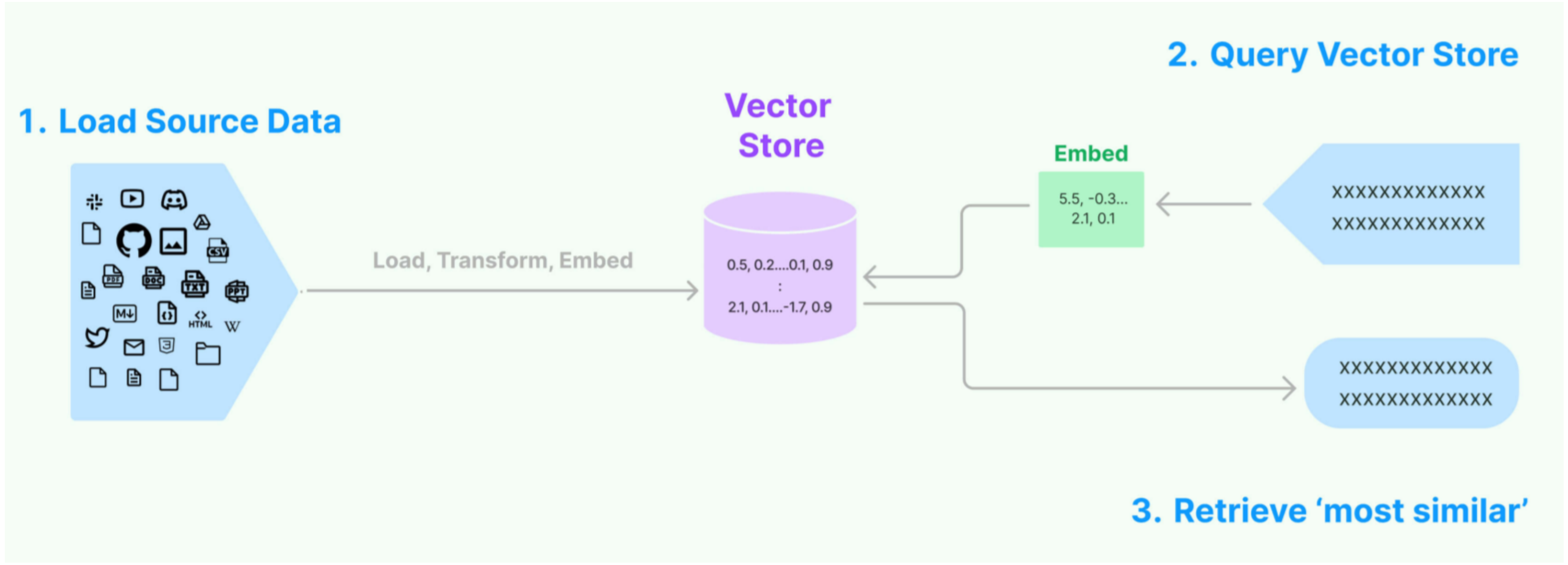
Local DB Directory name

Step-5: Retrieval

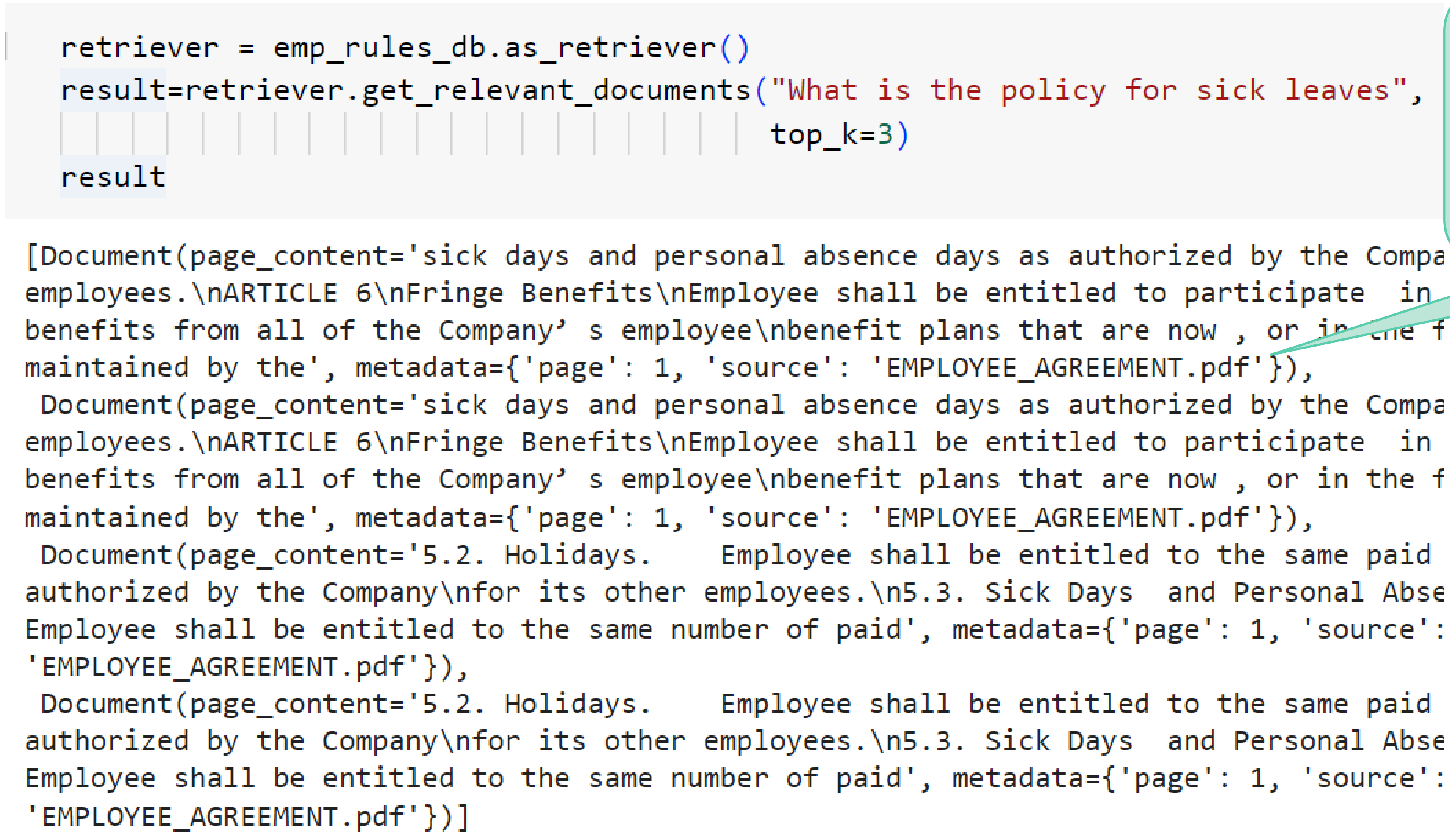
- LangChain enhances data retrieval with various algorithms.
- Supports simple semantic search for beginners.
- Adds advanced algorithms to improve performance.
- Parent Document Retriever enhances context by embedding chunks of documents.
- Self Query Retriever separates semantic queries from metadata filters.
- Ensemble Retriever combines multiple sources and algorithms for retrieval.

Step-5: Retrieval - Note

- Prompt is also converted into embedding.
- At query time, unstructured queries are embedded for similarity searches
- Vector stores handle the storage and retrieval of embedded data.



Retrieval



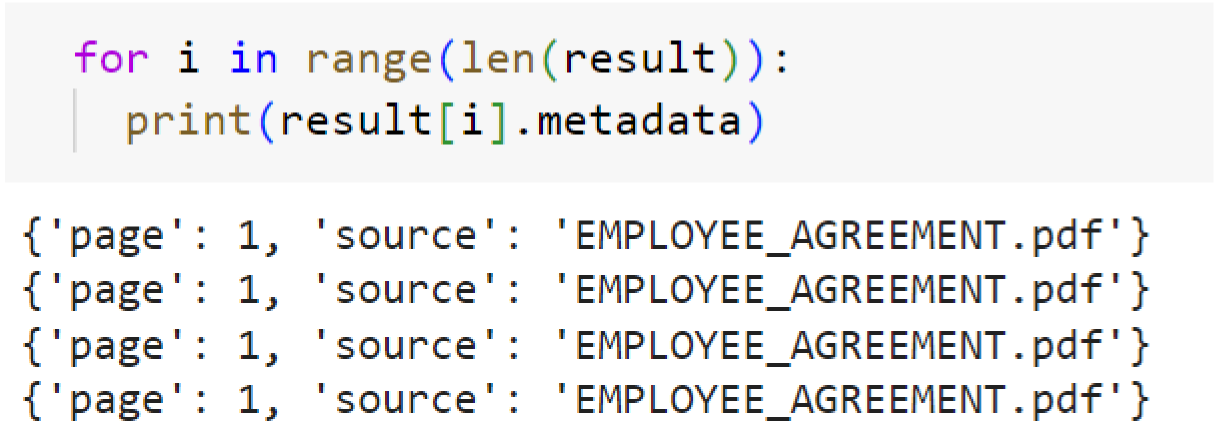
```
retriever = emp_rules_db.as_retriever()
result=retriever.get_relevant_documents("What is the policy for sick leaves",
                                        top_k=3)
result
```

```
[Document(page_content='sick days and personal absence days as authorized by the Compa
employees.\nARTICLE 6\nFringe Benefits\nEmployee shall be entitled to participate  in
benefits from all of the Company’ s employee\nbenefit plans that are now , or in the f
maintained by the', metadata={'page': 1, 'source': 'EMPLOYEE_AGREEMENT.pdf'}),
 Document(page_content='sick days and personal absence days as authorized by the Compa
employees.\nARTICLE 6\nFringe Benefits\nEmployee shall be entitled to participate  in
benefits from all of the Company’ s employee\nbenefit plans that are now , or in the f
maintained by the', metadata={'page': 1, 'source': 'EMPLOYEE_AGREEMENT.pdf'}),
 Document(page_content='5.2. Holidays.    Employee shall be entitled to the same paid
authorized by the Company\nfor its other employees.\n5.3. Sick Days  and Personal Abse
Employee shall be entitled to the same number of paid', metadata={'page': 1, 'source':
'EMPLOYEE_AGREEMENT.pdf'}),
 Document(page_content='5.2. Holidays.    Employee shall be entitled to the same paid
authorized by the Company\nfor its other employees.\n5.3. Sick Days  and Personal Abse
Employee shall be entitled to the same number of paid', metadata={'page': 1, 'source':
'EMPLOYEE_AGREEMENT.pdf'})]
```

These are relevant documents only, this is not the answer to the question.

For precise answers we need to use RetrievalQA Chain

Retrieval



```python
for i in range(len(result)):
    print(result[i].metadata)
```

```
{'page': 1, 'source': 'EMPLOYEE_AGREEMENT.pdf'}
{'page': 1, 'source': 'EMPLOYEE_AGREEMENT.pdf'}
{'page': 1, 'source': 'EMPLOYEE_AGREEMENT.pdf'}
{'page': 1, 'source': 'EMPLOYEE_AGREEMENT.pdf'}
```

Retrieval

```
retriever = emp_rules_db.as_retriever()
result=retriever.get_relevant_documents("What is the policy for insurance?",
                                        top_k=3)
result
```

```
[Document(page_content='Company for its employees, including, without limitation, the Company’ s
insurance plan. No amounts\npaid to Employee from an employee benefit plan shall count as compen
due Employee as base salary or\nadditional compensa tion.  Nothing in this Employee Agreement sh
prohibit the Company from modifying\nor terminating any of its employee benefit plans in a manne
does not discriminate between Employee\nand other Company employees.\nARTICLE 7\nTermination of
Employment\n7.1. Termination of Employment.  Employe e’s employment hereunder shall automaticall
terminate\nupon (i) his death; (ii) Employee voluntarily leaving the employ of the Company; (iii
Company’ s sole\ndiscretion, upon fifteen (15) days prior written notice to Employee if the Comp
terminates his\nemployment hereun der without "cause;" (iv) at the Company’ s sole discretio n,
(2) days prior\nwritten notice to Employee if the Company terminates his employment hereunder fo
"cause." For purposes', metadata={'page': 2, 'source': 'EMPLOYEE_AGREEMENT.pdf'}),
 Document(page_content='Company for its employees, including, without limitation, the Company’ s
insurance plan. No amounts\npaid to Employee from an employee benefit plan shall count as compen
due Employee as base salary or\nadditional compensa tion.  Nothing in this Employee Agreement sh
prohibit the Company from modifying\nor terminating any of its employee benefit plans in a manne
does not discriminate between Employee\nand other Company employees.\nARTICLE 7\nTermination of
```

Retrieval

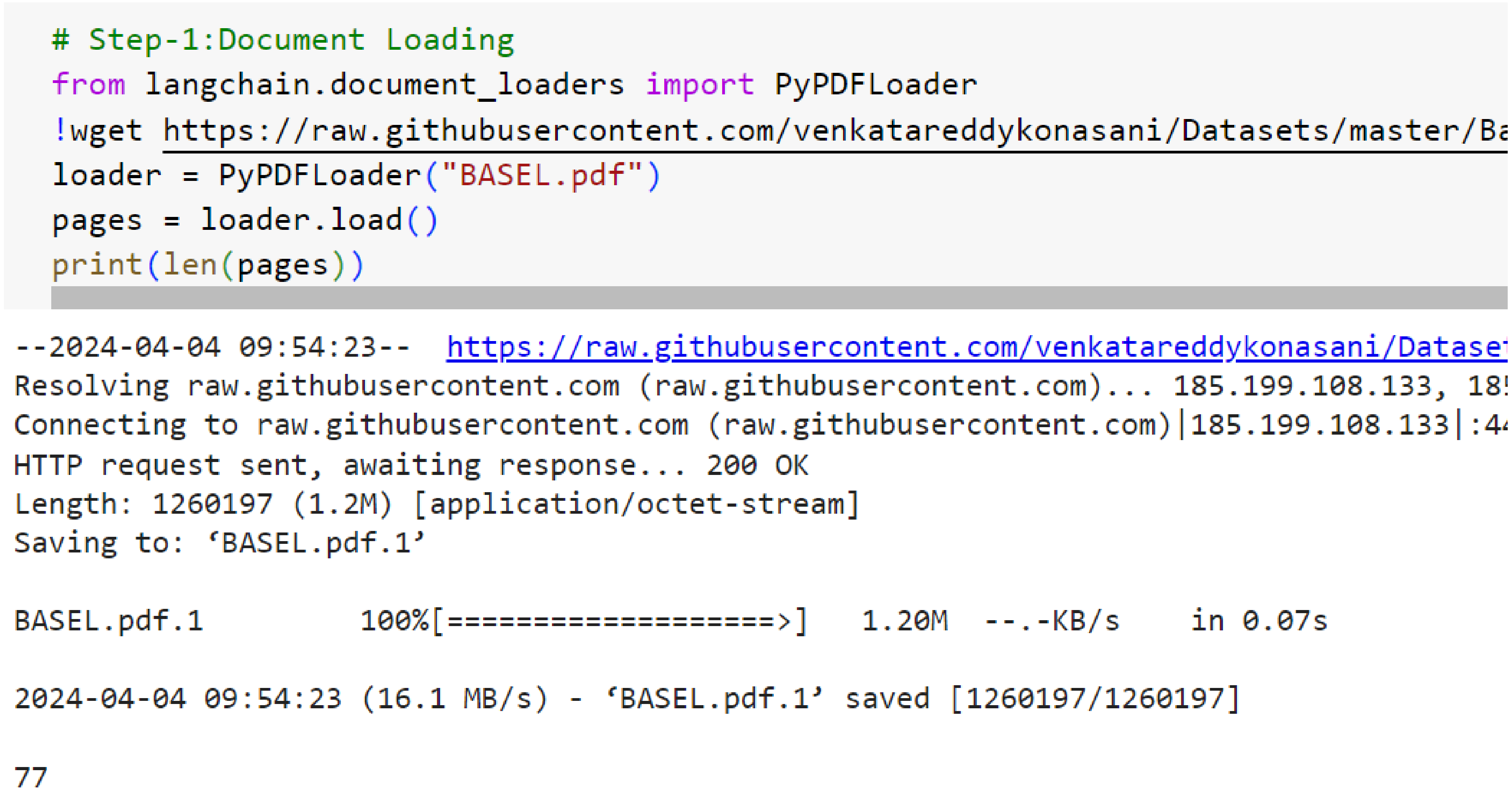
```
for i in range(len(result)):
    print(result[i].metadata)
```

```
{'page': 2, 'source': 'EMPLOYEE_AGREEMENT.pdf'}
{'page': 2, 'source': 'EMPLOYEE_AGREEMENT.pdf'}
{'page': 11, 'source': 'EMPLOYEE_AGREEMENT.pdf'}
{'page': 11, 'source': 'EMPLOYEE_AGREEMENT.pdf'}
```

DV Analytics
Transforming You

RAG Application

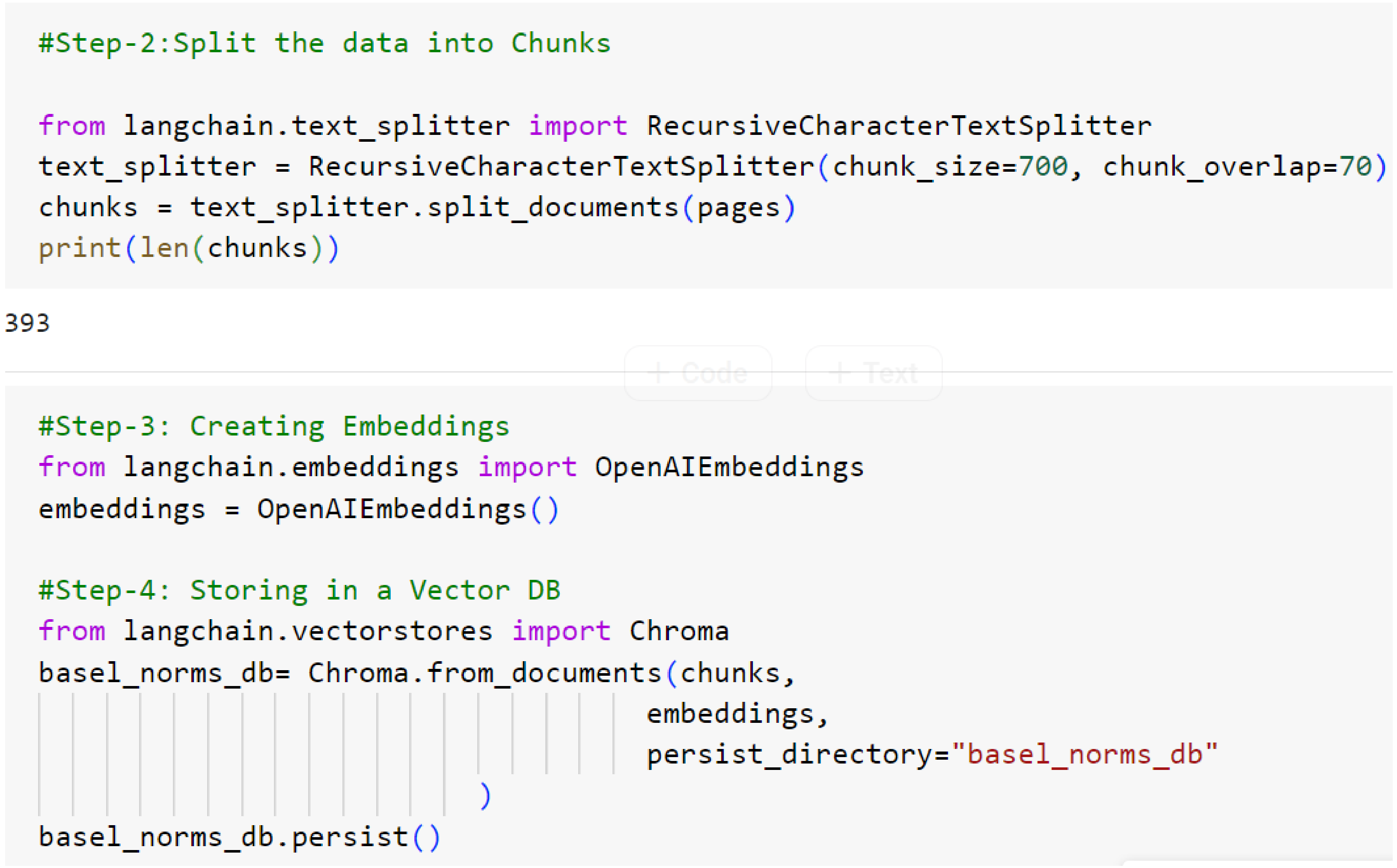
RAG on BASEL Norms



```
# Step-1:Document Loading
from langchain.document_loaders import PyPDFLoader
!wget https://raw.githubusercontent.com/venkatareddykonasani/Datasets/master/Ba
loader = PyPDFLoader("BASEL.pdf")
pages = loader.load()
print(len(pages))
```

```
--2024-04-04 09:54:23--  https://raw.githubusercontent.com/venkatareddykonasani/Dataset
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 18
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:4
HTTP request sent, awaiting response... 200 OK
Length: 1260197 (1.2M) [application/octet-stream]
Saving to: 'BASEL.pdf.1'

BASEL.pdf.1         100%[===================>]   1.20M  --.-KB/s    in 0.07s

2024-04-04 09:54:23 (16.1 MB/s) - 'BASEL.pdf.1' saved [1260197/1260197]

77
```

RAG on BASEL Norms

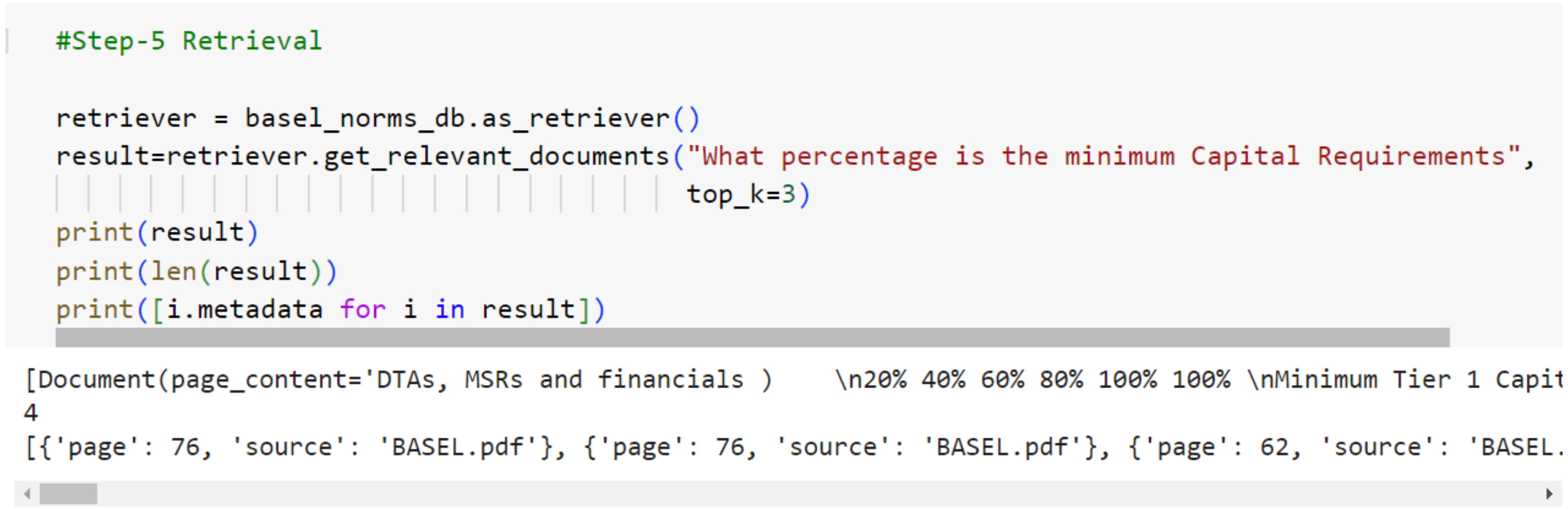


```
#Step-2:Split the data into Chunks

from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(chunk_size=700, chunk_overlap=70)
chunks = text_splitter.split_documents(pages)
print(len(chunks))
```

```
393
```

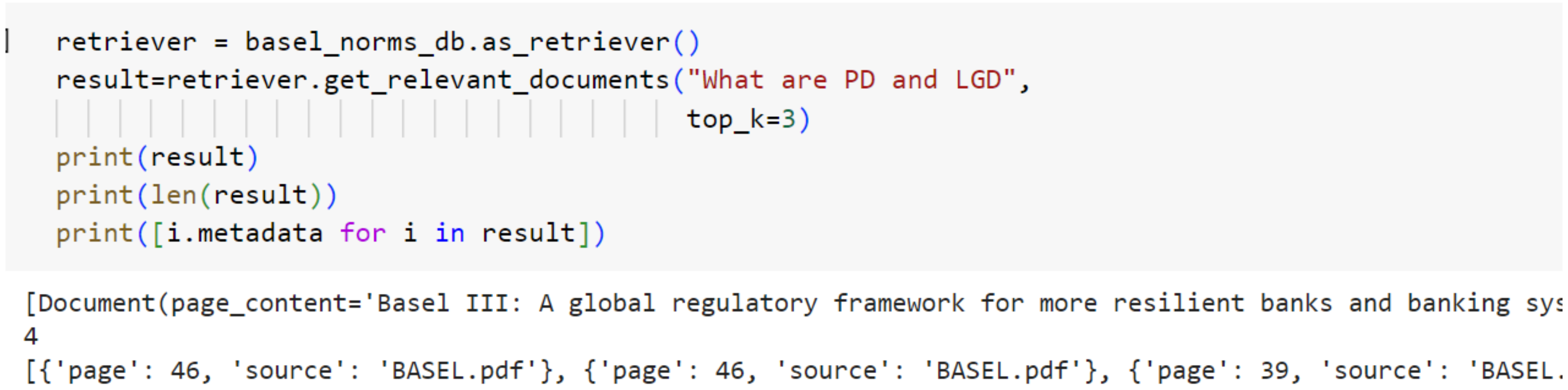
```
#Step-3: Creating Embeddings
from langchain.embeddings import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()

#Step-4: Storing in a Vector DB
from langchain.vectorstores import Chroma
basel_norms_db= Chroma.from_documents(chunks,
                                    embeddings,
                                    persist_directory="basel_norms_db"
                          )
basel_norms_db.persist()
```

RAG on BASEL Norms



```
#Step-5 Retrieval

retriever = basel_norms_db.as_retriever()
result=retriever.get_relevant_documents("What percentage is the minimum Capital Requirements",
                                        top_k=3)
print(result)
print(len(result))
print([i.metadata for i in result])
```

```
[Document(page_content='DTAs, MSRs and financials )    \n20% 40% 60% 80% 100% 100% \nMinimum Tier 1 Capit
4
[{'page': 76, 'source': 'BASEL.pdf'}, {'page': 76, 'source': 'BASEL.pdf'}, {'page': 62, 'source': 'BASEL.
```



```
retriever = basel_norms_db.as_retriever()
result=retriever.get_relevant_documents("What are PD and LGD",
                                        top_k=3)
print(result)
print(len(result))
print([i.metadata for i in result])
```

```
[Document(page_content='Basel III: A global regulatory framework for more resilient banks and banking sys
4
[{'page': 46, 'source': 'BASEL.pdf'}, {'page': 46, 'source': 'BASEL.pdf'}, {'page': 39, 'source': 'BASEL.
```

Multi-Query Retriever

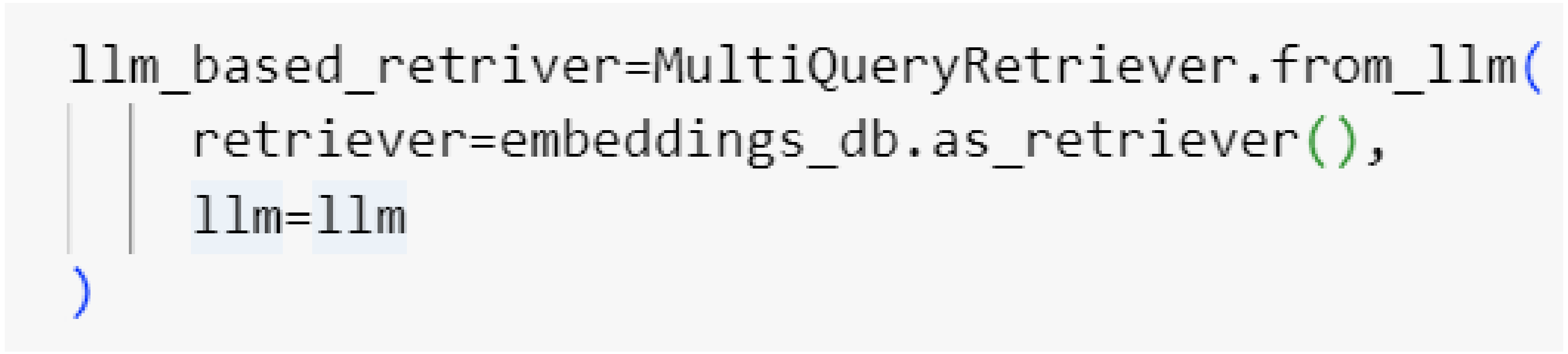
The need of Multi-Query Retriever

- Distance-based retrieval uses high-dimensional space for query representation.
- Similarity found based on "distance" between embedded queries and documents.
- Subtle query wording changes can alter retrieval results significantly.
- Retrieval accuracy depends on the embeddings' semantic capture of data.
- Prompt engineering/tuning manually addresses retrieval discrepancies.

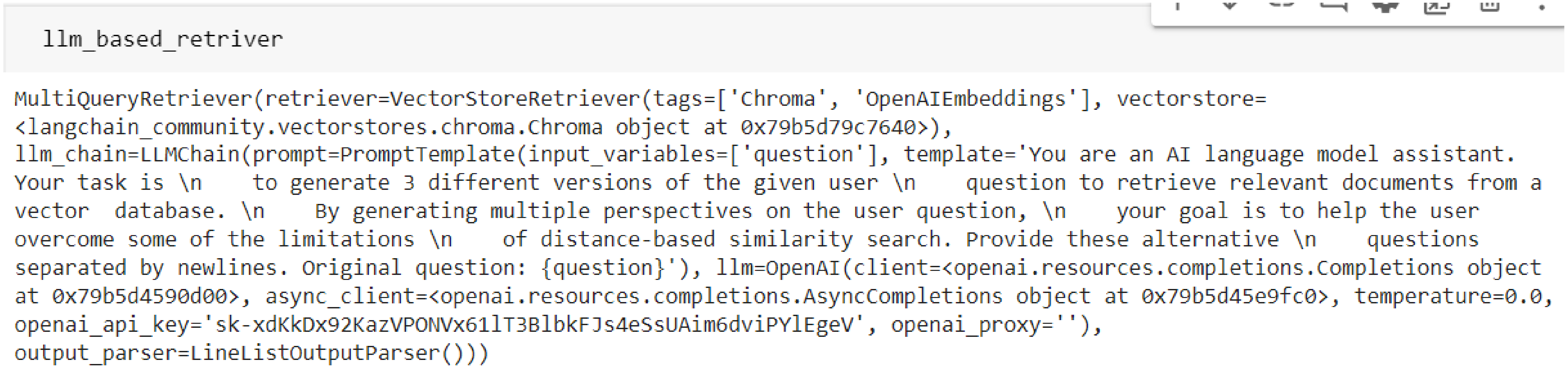
MultiQueryRetriever

- MultiQueryRetriever automates prompt tuning with an LLM for diverse queries.
- Generates multiple queries from different perspectives for one user input.
- Retrieves relevant documents for each query, combining results into a unique set.
- Aims to overcome limitations of distance-based retrieval.
- Provides a richer set of potentially relevant documents through varied queries.

```
#Document Loading
loader=wikipedia.WikipediaLoader(query="MS Dhoni")
documents=loader.load()

#Splitting
text_splitter=RecursiveCharacterTextSplitter(chunk_size=500,chunk_overlap=50)
docs=text_splitter.split_documents(documents)
len(docs)

#Embeddings and VectorDB
from langchain.embeddings import OpenAIEmbeddings
embeddings=OpenAIEmbeddings()

embeddings_db=Chroma.from_documents(docs,embeddings,
                                    persist_directory="wiki_db")
embeddings_db.persist()
```



```python
llm_based_retriver=MultiQueryRetriever.from_llm(
    retriever=embeddings_db.as_retriever(),
    llm=llm
)
```



```python
llm_based_retriver
```

```
MultiQueryRetriever(retriever=VectorStoreRetriever(tags=['Chroma', 'OpenAIEmbeddings'], vectorstore=
<langchain_community.vectorstores.chroma.Chroma object at 0x79b5d79c7640>),
llm_chain=LLMChain(prompt=PromptTemplate(input_variables=['question'], template='You are an AI language model assistant.
Your task is \n    to generate 3 different versions of the given user \n    question to retrieve relevant documents from a
vector  database. \n    By generating multiple perspectives on the user question, \n    your goal is to help the user
overcome some of the limitations \n    of distance-based similarity search. Provide these alternative \n    questions
separated by newlines. Original question: {question}'), llm=OpenAI(client=<openai.resources.completions.Completions object
at 0x79b5d4590d00>, async_client=<openai.resources.completions.AsyncCompletions object at 0x79b5d45e9fc0>, temperature=0.0,
openai_api_key='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', openai_proxy=''),
output_parser=LineListOutputParser()))
```

```
question1="What is the DOB of Dhoni?"
question2= "What Sport does Dhoni Play?"
rel_docs1=llm_based_retriver.get_relevant_documents(question1)
rel_docs2=llm_based_retriver.get_relevant_documents(question2)
```

```
INFO:langchain.retrievers.multi_query:Generated queries:
['1. When was Dhoni born?', '2. What is the date of birth
for Dhoni?', "3. Can you tell me Dhoni's date of birth?"]

INFO:langchain.retrievers.multi_query:Generated queries:
["1. What is Dhoni's preferred sport?", '2. Which sport
is associated with Dhoni?', '3. Can you tell me the sport
that Dhoni plays?']
```

Multiple queries generated

Contextual compression

Contextual compression

- Retrieval systems often face unpredictable queries during data ingestion.
- Relevant information may be hidden within largely irrelevant documents.
- Processing full documents increases LLM call costs and degrades responses.
- Contextual compression optimizes retrieval by focusing on query relevance.
- Compresses documents to retain only information pertinent to the query.
- Involves content compression within documents and entire document filtering.

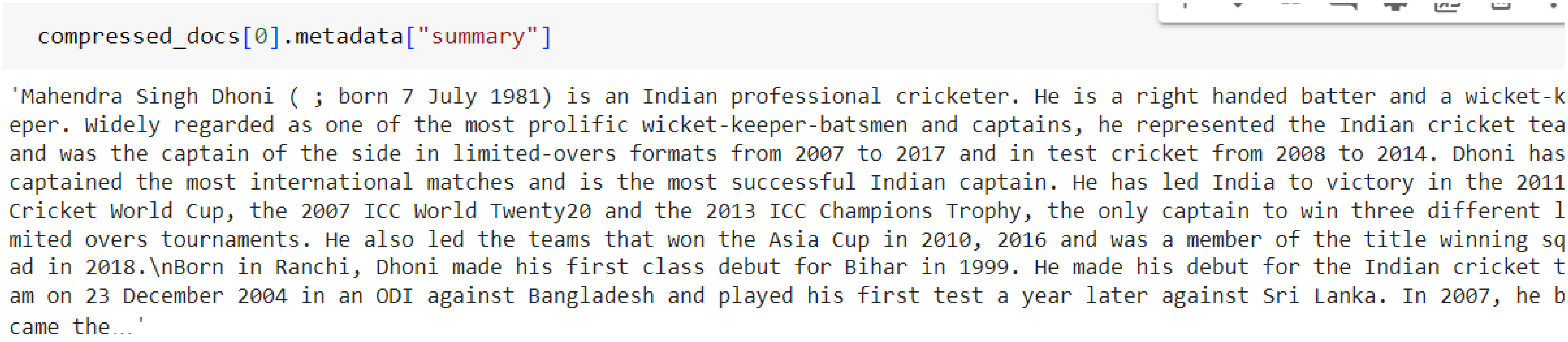
Two-Step Process of Contextual Document Compression

- Contextual Compression Retriever requires a base retriever and Document Compressor.
- Queries passed to the base retriever to fetch initial documents.
- Document Compressor shortens list by content reduction or document removal.

Code- Contextual compression

```
llm=OpenAI(temperature=0)

compressor=LLMChainExtractor.from_llm(llm)

compression_retriever=ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=embeddings_db.as_retriever()
)

compressed_docs=compression_retriever.get_relevant_documents
(question1)
```

Code- Contextual compression



```
compressed_docs[0].metadata["summary"]
```

```
'Mahendra Singh Dhoni ( ; born 7 July 1981) is an Indian professional cricketer. He is a right handed batter and a wicket-k
eper. Widely regarded as one of the most prolific wicket-keeper-batsmen and captains, he represented the Indian cricket tea
and was the captain of the side in limited-overs formats from 2007 to 2017 and in test cricket from 2008 to 2014. Dhoni has
captained the most international matches and is the most successful Indian captain. He has led India to victory in the 2011
Cricket World Cup, the 2007 ICC World Twenty20 and the 2013 ICC Champions Trophy, the only captain to win three different l
mited overs tournaments. He also led the teams that won the Asia Cup in 2010, 2016 and was a member of the title winning sq
ad in 2018.\nBorn in Ranchi, Dhoni made his first class debut for Bihar in 1999. He made his debut for the Indian cricket t
am on 23 December 2004 in an ODI against Bangladesh and played his first test a year later against Sri Lanka. In 2007, he b
came the...'
```

RetrievalQA Chain

RetrievalQA Chain

- RetrievalQAChain combines a Retriever and QA chain in Langchain.
- It retrieves documents using the Retriever component.
- Retrieved passages are fed to a large language model for answers.
- The QA chain then answers questions based on retrieved documents.

```python
from langchain.chains import RetrievalQA
llm=OpenAI(temperature=0)

Q_AChain=RetrievalQA.from_chain_type(
        llm=llm,
        chain_type="stuff",# It takes a list of documents,
        retriever=llm_based_retriver
)
```

```python
query="What is the DOB of Dhoni?"
docs=Q_AChain({"query":query})
docs["result"]
```

```
INFO:langchain.retrievers.multi_query:Ger
' 7 July 1981'
```

Chain type= “Stuff”: This is basic type of summarization. It takes a list of documents, inserts them all into a prompt

Prompt template behind Q&A Chain

```
print(Q_AChain.combine_documents_chain.llm_chain.prompt.template)
```

```
Use the following pieces of context to answer the question at the end. If
you don't know the answer, just say that you don't know, don't try to make
up an answer.

{context}

Question: {question}

Helpful Answer:
```

This is Woking in the backend of Q&A Chain

Thank you