Email Search AI - Langchain

Dataset:

https://www.kaggle.com/code/xokent/email-thread-summary-nlp

Problem Statement

Above Kaggle Dataset contains different email threads between different individuals. So it's very hard to get the insight of a full conversation as there are about 21864 rows and each thread has multiple rows. For an outside person it's very hard to understand what these conversations are all about.

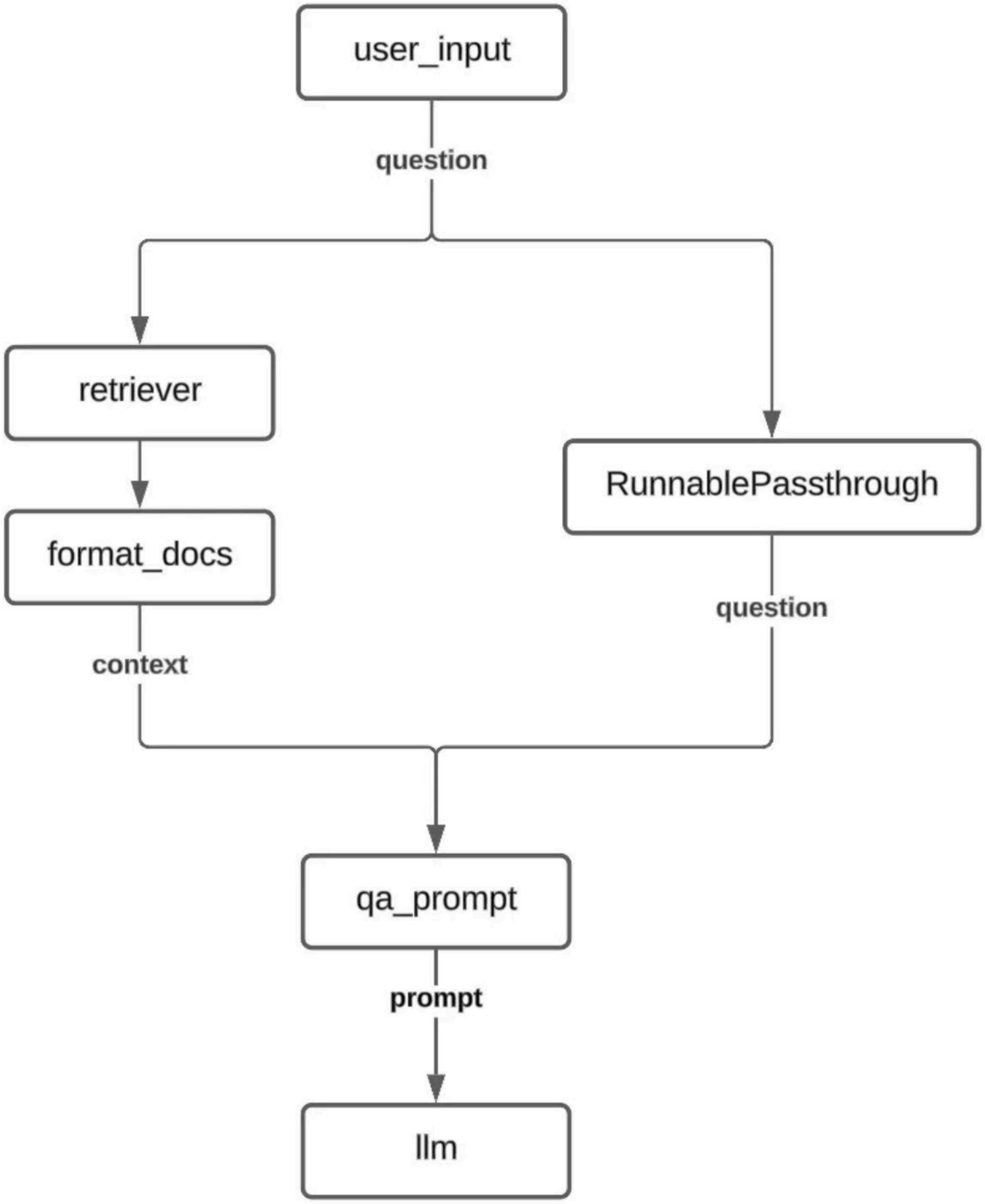
Solution

So based on the problem, building a QAChain or Conversational Chain will be an appropriate application to get required information about the email conversations happening among different individuals.

Components used:

1. LLM's (Large language models)
2. Chains (Retrieval QA, ConversationalRetrievalChain)
3. Memory
4. Embeddings
5. Vector Stores
6. Retrievers

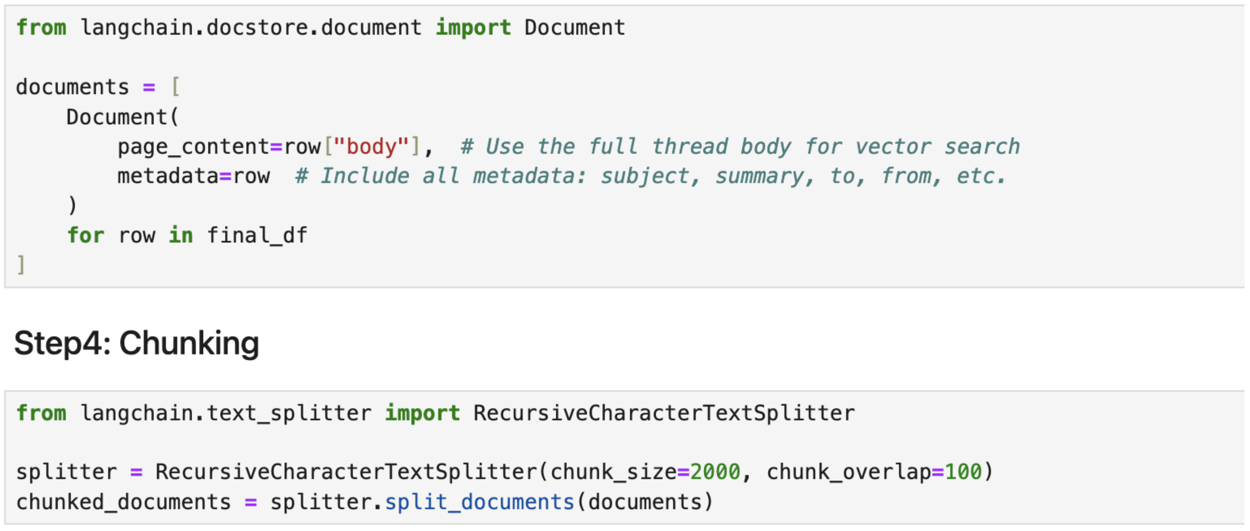
Please find below the simple flowchart:



Step1 - Data loading and metadata creation:

1. Loaded both datasets df1 = **email_thread_details.csv** and df2 = **email_thread_summaries.csv**.
2. Created a metadata of the important information in a new column as metadat_dict in the form of key value pairs in df1.
3. Dropped the other columns and then merged df1 and df2 and included summary columns as well in metada_dict.
4. Now we will utilize this metadata to build the documents and then chunking the same.

Step2 - Langchin Document Object and Chunking:



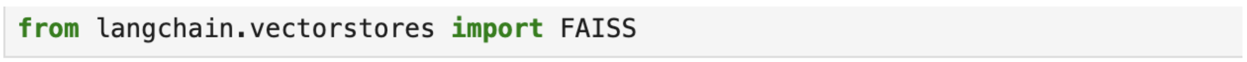
```python
from langchain.docstore.document import Document

documents = [
    Document(
        page_content=row["body"],  # Use the full thread body for vector search
        metadata=row  # Include all metadata: subject, summary, to, from, etc.
    )
    for row in final_df
]
```

Step4: Chunking

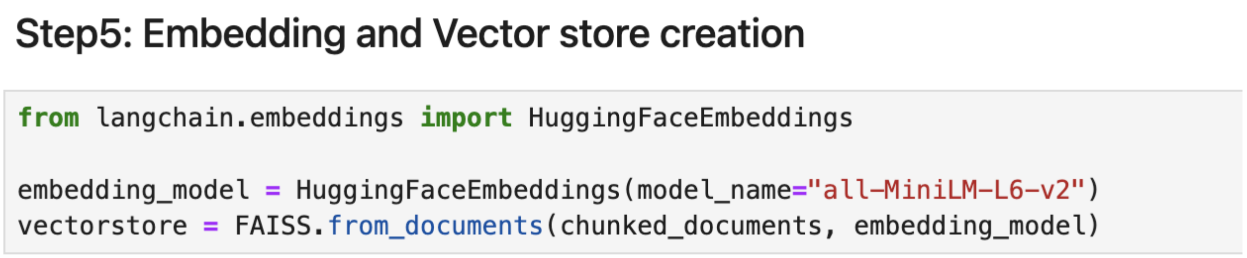
```python
from langchain.text_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(chunk_size=2000, chunk_overlap=100)
chunked_documents = splitter.split_documents(documents)
```

Created lanchain document object and used recursive character splitter for chunking with a chunk size of ‘2000’ having an overlap of ‘100’. Splitted the documents accordingly.

Step3 - Embedding and Vector Store creation:



```python
from langchain.vectorstores import FAISS
```

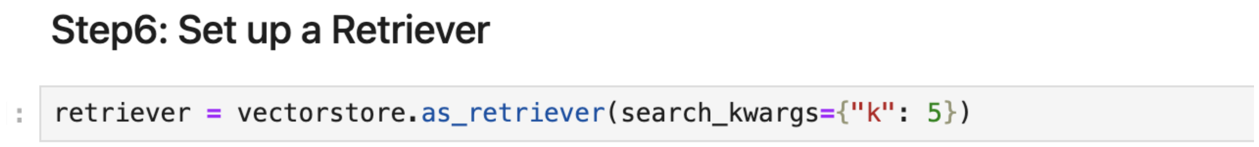


Step5: Embedding and Vector store creation

```python
from langchain.embeddings import HuggingFaceEmbeddings

embedding_model = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vectorstore = FAISS.from_documents(chunked_documents, embedding_model)
```

Created vector store using **FAISS** and used HuggingFaceEmbeddings model - **“all-MiniLM-L6-v2”.**
Embeddings converts text to numeric vectors for semantic search, helps in similarity search and RAG pipelines.
Vector Stores Stores vector embeddings and enables similarity search. Retrieve relevant chunks of text.

Step4 - Retriever

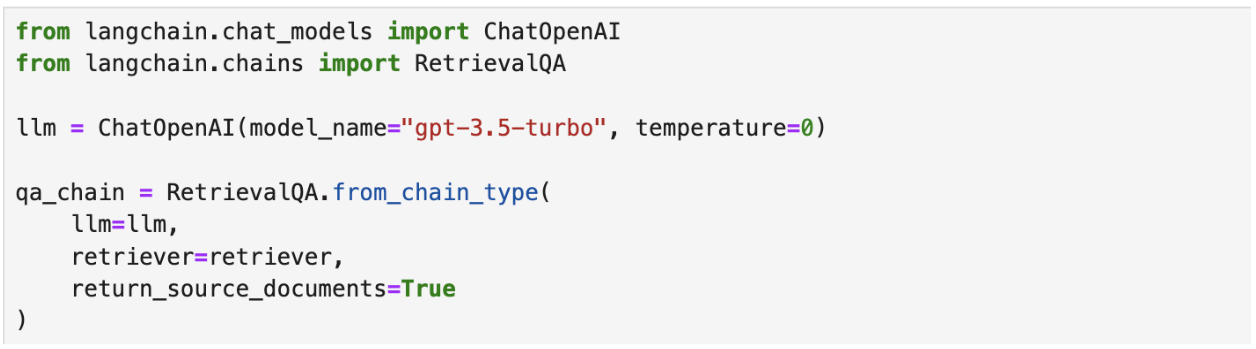


Step6: Set up a Retriever

```python
retriever = vectorstore.as_retriever(search_kwargs={"k": 5})
```

Set up the retriever which is a interface to fetch the documents acts a bridge between vector store and chain

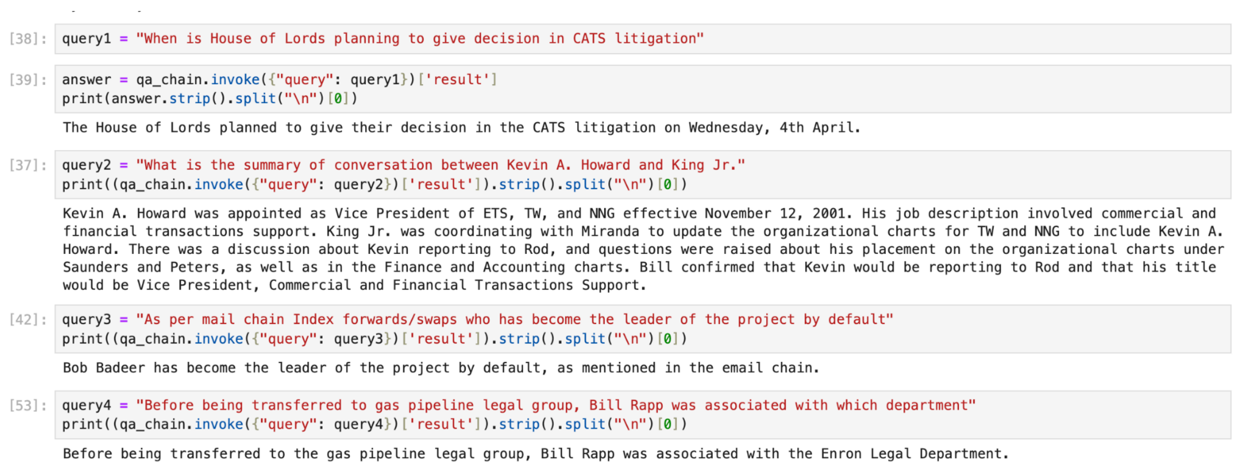
Step5: Generate QA Chain



```python
from langchain.chat_models import ChatOpenAI
from langchain.chains import RetrievalQA

llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)

qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever,
    return_source_documents=True
)
```

Using LLM “gpt-3.5-turbo” to generate QA Chain.

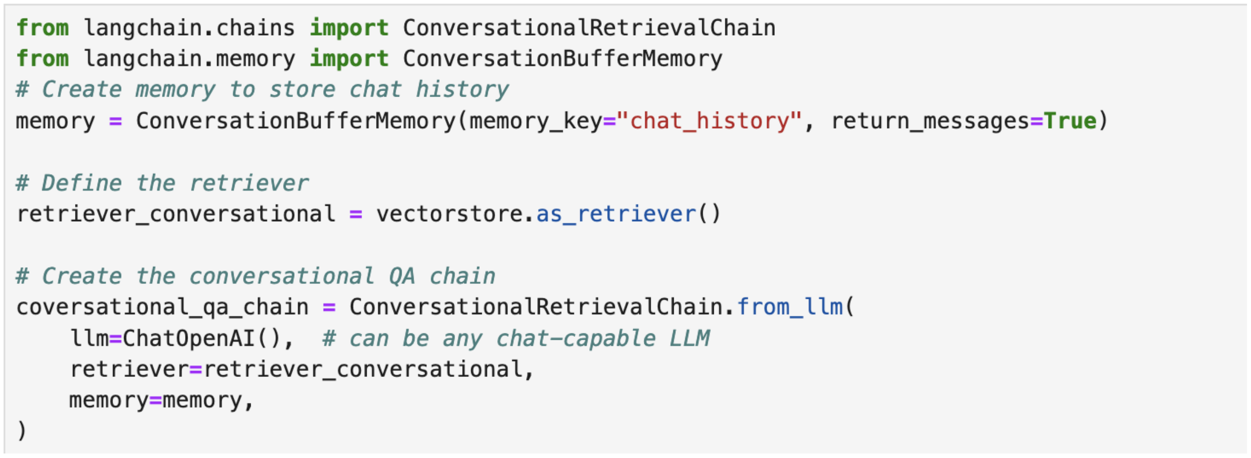
Few Outputs:



```
[38]: query1 = "When is House of Lords planning to give decision in CATS litigation"

[39]: answer = qa_chain.invoke({"query": query1})['result']
      print(answer.strip().split("\n")[0])

      The House of Lords planned to give their decision in the CATS litigation on Wednesday, 4th April.

[37]: query2 = "What is the summary of conversation between Kevin A. Howard and King Jr."
      print((qa_chain.invoke({"query": query2})['result']).strip().split("\n")[0])

      Kevin A. Howard was appointed as Vice President of ETS, TW, and NNG effective November 12, 2001. His job description involved commercial and
      financial transactions support. King Jr. was coordinating with Miranda to update the organizational charts for TW and NNG to include Kevin A.
      Howard. There was a discussion about Kevin reporting to Rod, and questions were raised about his placement on the organizational charts under
      Saunders and Peters, as well as in the Finance and Accounting charts. Bill confirmed that Kevin would be reporting to Rod and that his title
      would be Vice President, Commercial and Financial Transactions Support.

[42]: query3 = "As per mail chain Index forwards/swaps who has become the leader of the project by default"
      print((qa_chain.invoke({"query": query3})['result']).strip().split("\n")[0])

      Bob Badeer has become the leader of the project by default, as mentioned in the email chain.

[53]: query4 = "Before being transferred to gas pipeline legal group, Bill Rapp was associated with which department"
      print((qa_chain.invoke({"query": query4})['result']).strip().split("\n")[0])

      Before being transferred to the gas pipeline legal group, Bill Rapp was associated with the Enron Legal Department.
```

3 out of 4 outputs are to the point and we are getting the right results for the queries.

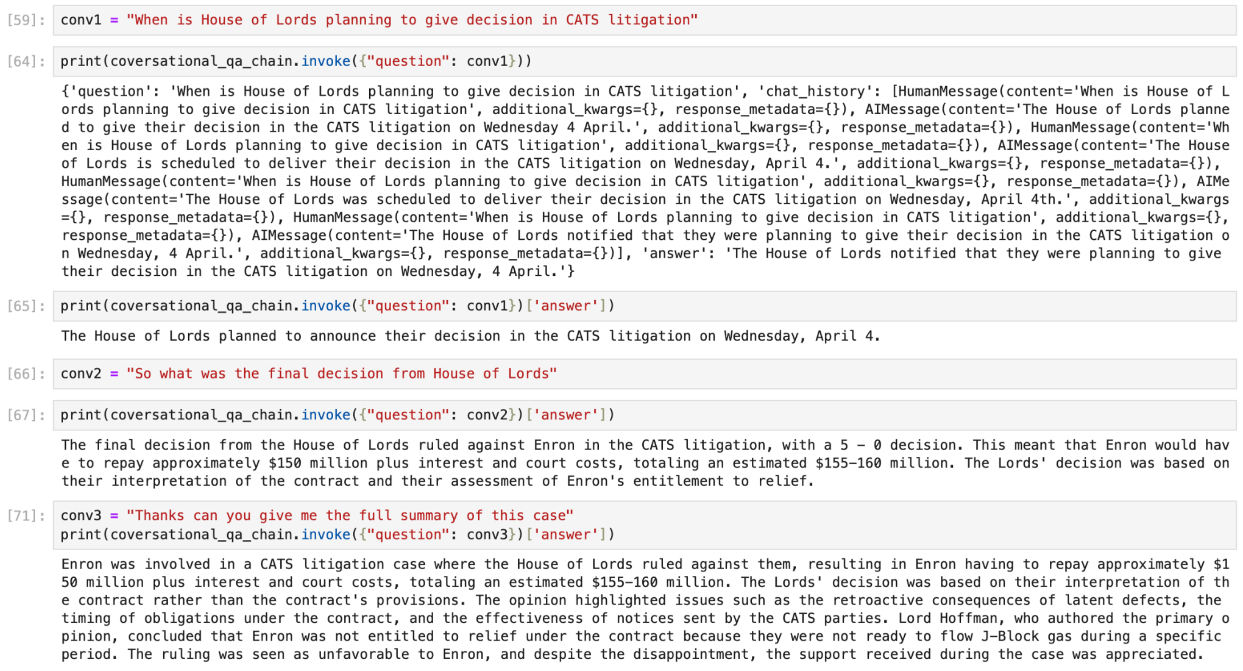
Step6: Generate Conversational QA chain



```python
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory
# Create memory to store chat history
memory = ConversationBufferMemory(memory_key="chat_history", return_messages=True)

# Define the retriever
retriever_conversational = vectorstore.as_retriever()

# Create the conversational QA chain
coversational_qa_chain = ConversationalRetrievalChain.from_llm(
    llm=ChatOpenAI(),  # can be any chat-capable LLM
    retriever=retriever_conversational,
    memory=memory,
)
```

Using **ConversationalRetrievalChain** for building a conversational chain
Making use of memory → **ConversationalBufferMemory** to keep the chat history.

Output for the conversational chain:



```
[59]: conv1 = "When is House of Lords planning to give decision in CATS litigation"

[64]: print(coversational_qa_chain.invoke({"question": conv1}))

      {'question': 'When is House of Lords planning to give decision in CATS litigation', 'chat_history': [HumanMessage(content='When is House of L
      ords planning to give decision in CATS litigation', additional_kwargs={}, response_metadata={}), AIMessage(content='The House of Lords planne
      d to give their decision in the CATS litigation on Wednesday 4 April.', additional_kwargs={}, response_metadata={}), HumanMessage(content='Wh
      en is House of Lords planning to give decision in CATS litigation', additional_kwargs={}, response_metadata={}), AIMessage(content='The House
      of Lords is scheduled to deliver their decision in the CATS litigation on Wednesday, April 4.', additional_kwargs={}, response_metadata={}),
      HumanMessage(content='When is House of Lords planning to give decision in CATS litigation', additional_kwargs={}, response_metadata={}), AIMe
      ssage(content='The House of Lords was scheduled to deliver their decision in the CATS litigation on Wednesday, April 4th.', additional_kwargs
      ={}, response_metadata={}), HumanMessage(content='When is House of Lords planning to give decision in CATS litigation', additional_kwargs={},
      response_metadata={}), AIMessage(content='The House of Lords notified that they were planning to give their decision in the CATS litigation o
      n Wednesday, 4 April.', additional_kwargs={}, response_metadata={})], 'answer': 'The House of Lords notified that they were planning to give
      their decision in the CATS litigation on Wednesday, 4 April.'}

[65]: print(coversational_qa_chain.invoke({"question": conv1})['answer'])

      The House of Lords planned to announce their decision in the CATS litigation on Wednesday, April 4.

[66]: conv2 = "So what was the final decision from House of Lords"

[67]: print(coversational_qa_chain.invoke({"question": conv2})['answer'])

      The final decision from the House of Lords ruled against Enron in the CATS litigation, with a 5 - 0 decision. This meant that Enron would hav
      e to repay approximately $150 million plus interest and court costs, totaling an estimated $155-160 million. The Lords' decision was based on
      their interpretation of the contract and their assessment of Enron's entitlement to relief.

[71]: conv3 = "Thanks can you give me the full summary of this case"
      print(coversational_qa_chain.invoke({"question": conv3})['answer'])

      Enron was involved in a CATS litigation case where the House of Lords ruled against them, resulting in Enron having to repay approximately $1
      50 million plus interest and court costs, totaling an estimated $155-160 million. The Lords' decision was based on their interpretation of th
      e contract rather than the contract's provisions. The opinion highlighted issues such as the retroactive consequences of latent defects, the
      timing of obligations under the contract, and the effectiveness of notices sent by the CATS parties. Lord Hoffman, who authored the primary o
      pinion, concluded that Enron was not entitled to relief under the contract because they were not ready to flow J-Block gas during a specific
      period. The ruling was seen as unfavorable to Enron, and despite the disappointment, the support received during the case was appreciated.
```

Summary

1. We built a simple QA Chain (Question Answer) using Langchain where we can find answers for our query from the corpus of data, and we are getting more than 90% accuracy.
2. Also we can build a conversational chain just like chatbots and we were able to maintain the history of conversation going. And answers provided by the conversational chain were based on the previous instance.