Analytics Vidhya

Mastering RAG

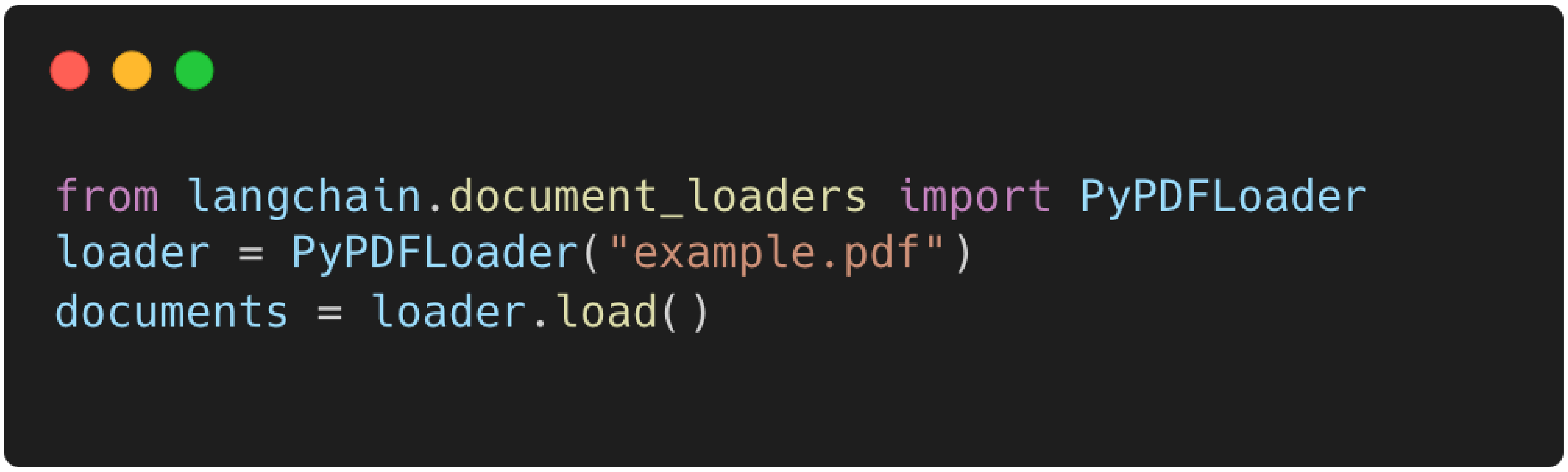
RAG with LangChain

```python
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.schema.output_parser import StrOutputParser
from langchain.schema.runnable import RunnablePassthrough

# Initialize components
llm = OpenAI()
prompt_template = """Answer the question based on the context below:

Context: {context}

Question: {question}
Answer: """
prompt = PromptTemplate.from_template(prompt_template)

# Create LCEL chain
qa_chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

# Invoke the chain
response = qa_chain.invoke("Explain RAG")
print(response)
```

LangChain provides several modules that facilitate RAG implementation:

1. **Document Loaders**

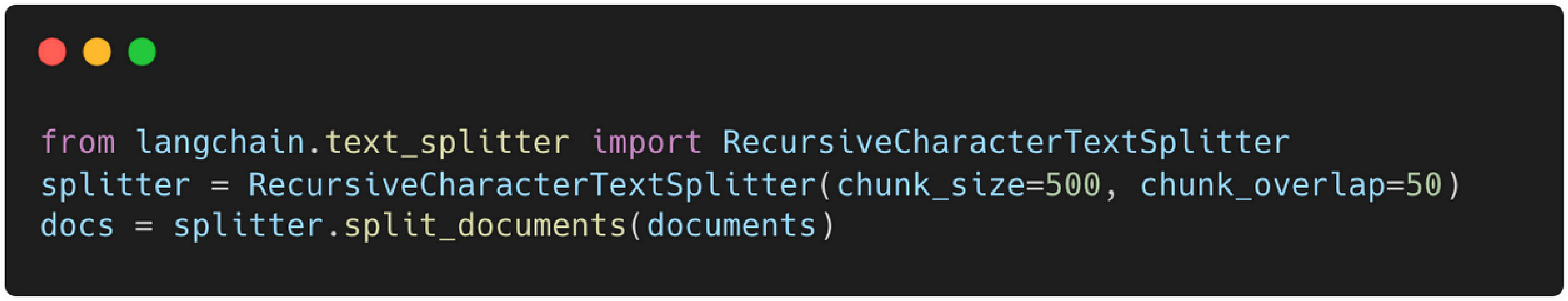
These modules help fetch data from various sources such as PDFs, web pages, databases, and APIs.



```python
from langchain.document_loaders import PyPDFLoader
loader = PyPDFLoader("example.pdf")
documents = loader.load()
```

2. Text Splitting

To efficiently process large documents, text must be split into manageable chunks.



```python
from langchain.text_splitter import RecursiveCharacterTextSplitter
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
docs = splitter.split_documents(documents)
```

3. Vector Stores

Storing document embeddings in a vector database enables efficient similarity searches.

```
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings

embeddings = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(docs, embeddings)
```

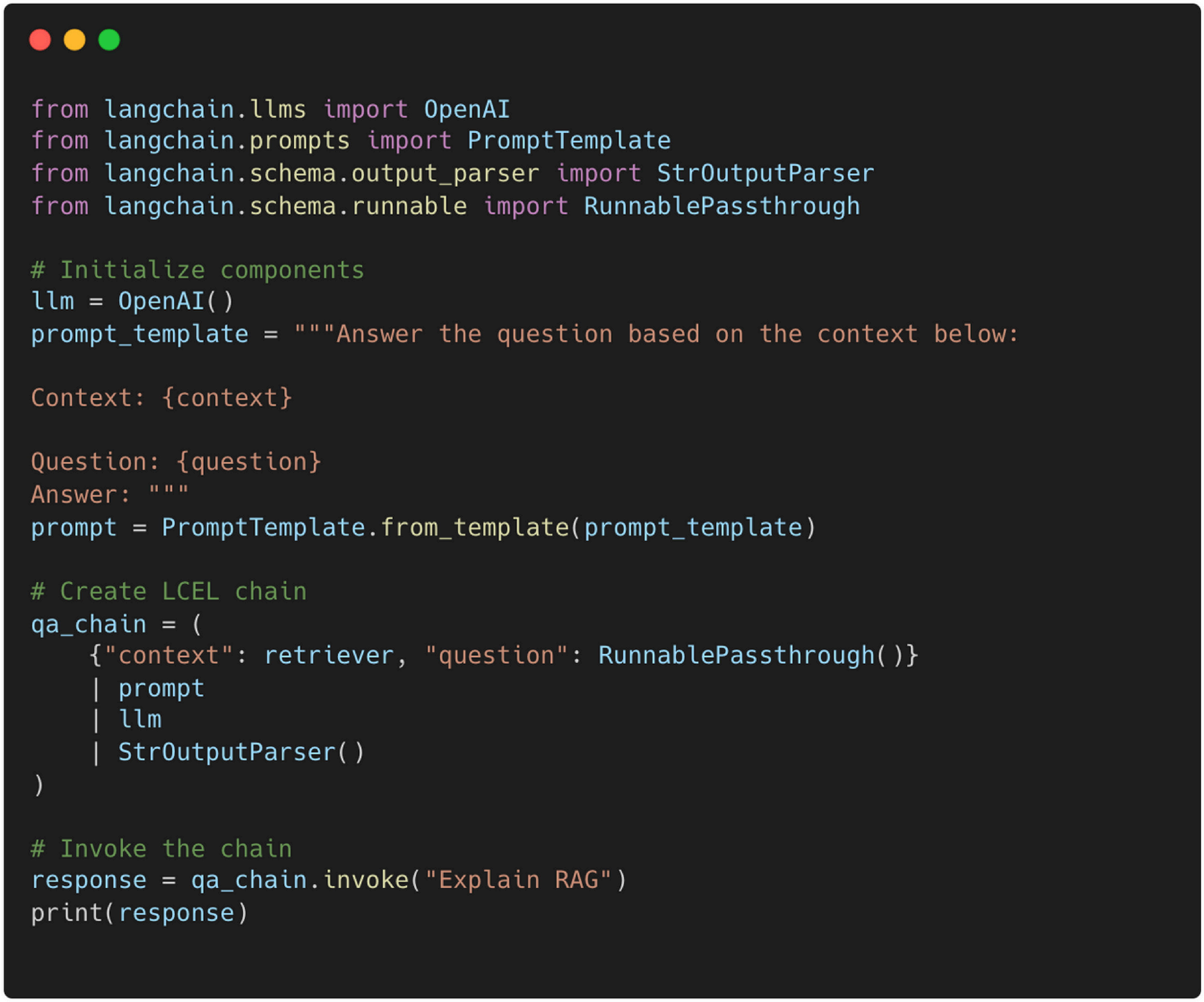
4. Retrieval Mechanism

The retriever fetches relevant documents based on user queries.

```
retriever = vectorstore.as_retriever()

# LCEL version using invoke()
retrieved_docs = retriever.invoke("What is RAG?")
```

5. LLM Integration

Using retrieved documents, the LLM generates responses enriched with external knowledge.



```python
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.schema.output_parser import StrOutputParser
from langchain.schema.runnable import RunnablePassthrough

# Initialize components
llm = OpenAI()
prompt_template = """Answer the question based on the context below:

Context: {context}

Question: {question}
Answer: """
prompt = PromptTemplate.from_template(prompt_template)

# Create LCEL chain
qa_chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

# Invoke the chain
response = qa_chain.invoke("Explain RAG")
print(response)
```