AI – Chatbot using Ollama and Python

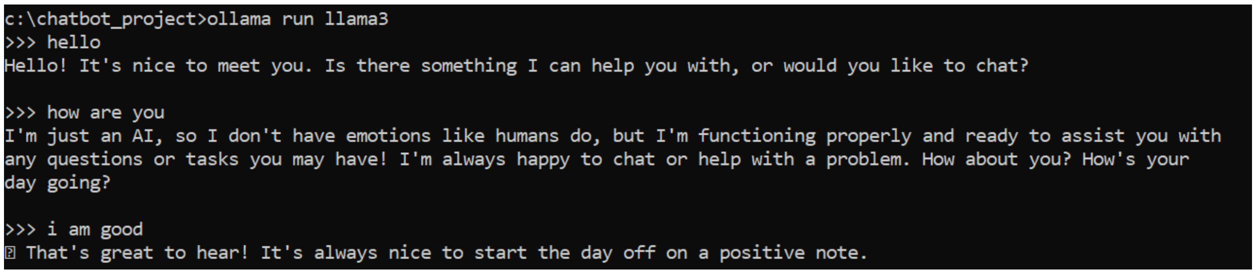
Ollama is a groundbreaking platform that democratizes access to large language models (LLMs) by enabling users to run them locally on their machines. Developed with a vision to empower individuals and organizations, Ollama provides a user-friendly interface and seamless integration capabilities, making it easier than ever to leverage the power of LLMs for various applications and use cases.

Key Features of Ollama

- **Local Execution:** One of the distinguishing features of Ollama is its ability to run LLMs locally, mitigating privacy concerns associated with cloud-based solutions. By bringing AI models directly to users' devices, Ollama ensures greater control and security over data while providing faster processing speeds and reduced reliance on external servers.
- **Extensive Model Library**: Ollama offers access to an extensive library of pre-trained LLMs, including popular models like Llama 3. Users can choose from a range of models tailored to different tasks, domains, and hardware capabilities, ensuring flexibility and versatility in their AI projects.
- **Seamless Integration:** Ollama seamlessly integrates with a variety of tools, frameworks, and programming languages, making it easy for developers to incorporate LLMs into their workflows. Whether it's Python, LangChain, or LlamaIndex, Ollama provides robust integration options for building sophisticated AI applications and solutions.
- **Customization and Fine-tuning:** With Ollama, users have the ability to customize and fine-tune LLMs to suit their specific needs and preferences. From prompt engineering to few-shot learning and fine-tuning processes, Ollama empowers users to shape the behavior and outputs of LLMs, ensuring they align with the desired objectives.

Download and install Ollama from https://ollama.com/

- Ollama allows you to run various open-source LLMs. Here, we'll use Llama 3 as an example.
- Use the following command to download the Llama 3 model: **ollama pull llama3**
- Run using **ollama run llama3** and interact with it.it used llama2 LLM to answer



```
c:\chatbot_project>ollama run llama3
>>> hello
Hello! It's nice to meet you. Is there something I can help you with, or would you like to chat?

>>> how are you
I'm just an AI, so I don't have emotions like humans do, but I'm functioning properly and ready to assist you with
any questions or tasks you may have! I'm always happy to chat or help with a problem. How about you? How's your
day going?

>>> i am good
� That's great to hear! It's always nice to start the day off on a positive note.
```

Type **/bye** to come out of chat mode.

How to execute ollama LLM using Python –

Set Up a Virtual Environment

Using a virtual environment is like giving your Python project its own little workspace. It keeps all the necessary libraries separate from your system's main setup, preventing any conflicts between different projects.

If you're new to coding, think of it like a dedicated folder where your chatbot's tools stay organized and don't interfere with anything else on your computer. This way, you avoid headaches down the line when installing or updating different Python packages.

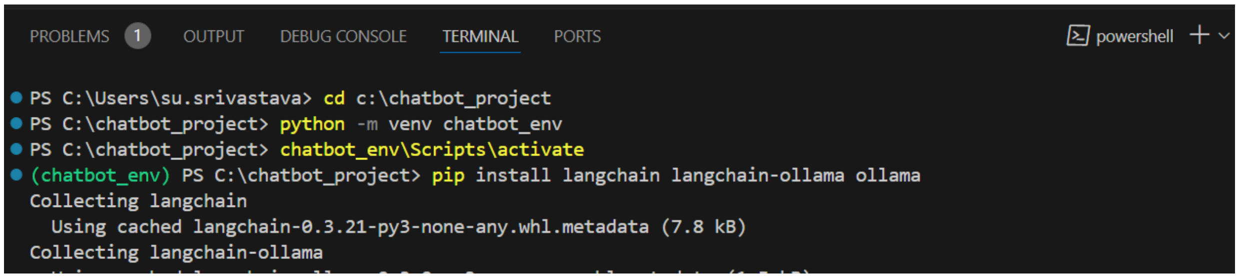
Run below in Visual studio code' s Terminal

Go to directory - PS C:\Users\su.srivastava> **cd c:\chatbot_project**

Create virtual env - PS C:\chatbot_project> **python -m venv chatbot_env**

Activate virtual env - PS C:\chatbot_project> **chatbot_env\Scripts\activate**

Install libraries – (chatbot_env) PS C:\chatbot_project> pip install langchain langchain-ollama ollama



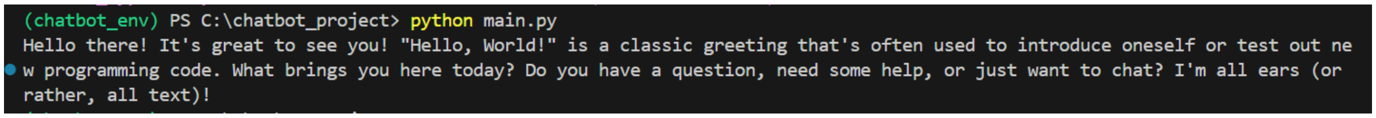
```
PS C:\Users\su.srivastava> cd c:\chatbot_project
PS C:\chatbot_project> python -m venv chatbot_env
PS C:\chatbot_project> chatbot_env\Scripts\activate
(chatbot_env) PS C:\chatbot_project> pip install langchain langchain-ollama ollama
Collecting langchain
  Using cached langchain-0.3.21-py3-none-any.whl.metadata (7.8 kB)
Collecting langchain-ollama
```

Create a python script – main.py

```
from langchain_ollama import OllamaLLM

model = OllamaLLM(model="llama3")

result = model.invoke(input="hello world")

print(result)
```

Run python script - **(chatbot_env) PS C:\chatbot_project> python main.py**

Interact same way as we were interacting with llama3 using Ollama. Input will be used as question to LLM here.



```
(chatbot_env) PS C:\chatbot_project> python main.py
Hello there! It's great to see you! "Hello, World!" is a classic greeting that's often used to introduce oneself or test out ne
w programming code. What brings you here today? Do you have a question, need some help, or just want to chat? I'm all ears (or
rather, all text)!
```

How to use Template – main.py

Lang chain is something that allows us to more easily interact with LMS. One of the things we can do is create a template that will pass to the LLM that contains our specific query or prompt. And this way we can give it some more description and instructions on what it should do.

Triple quotes are multiline string. Variables are inside curly braces.

```
from langchain_ollama import OllamaLLM

from langchain_core.prompts import ChatPromptTemplate

template = """

Answer the question below.

Here is the conversation history {context}

Question: {question}

Answer:

"""
```

model = OllamaLLM(model="llama3")

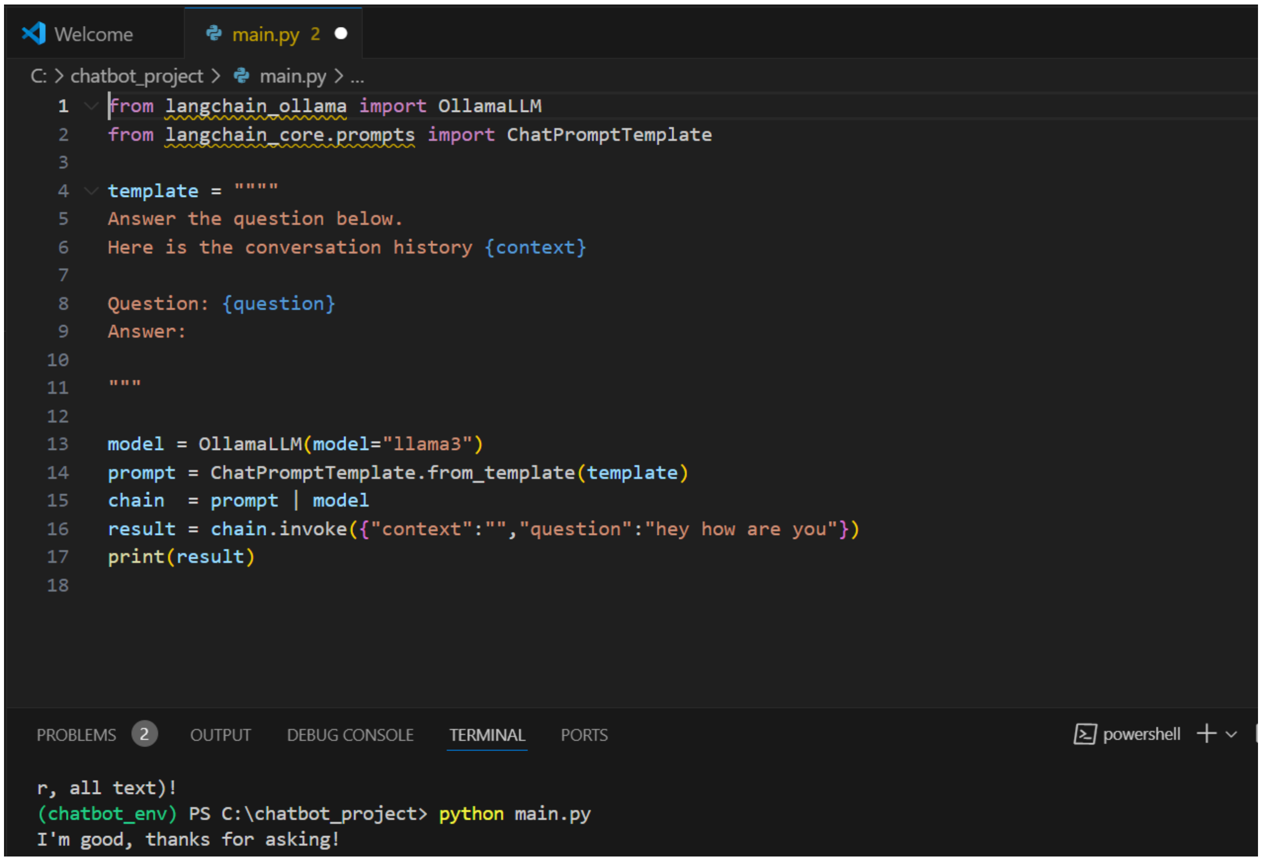
prompt = ChatPromptTemplate.from_template(template)

chain = prompt | model

result = chain.invoke({"context":"","question":"hey how are you"})

print(result)

Terminal result

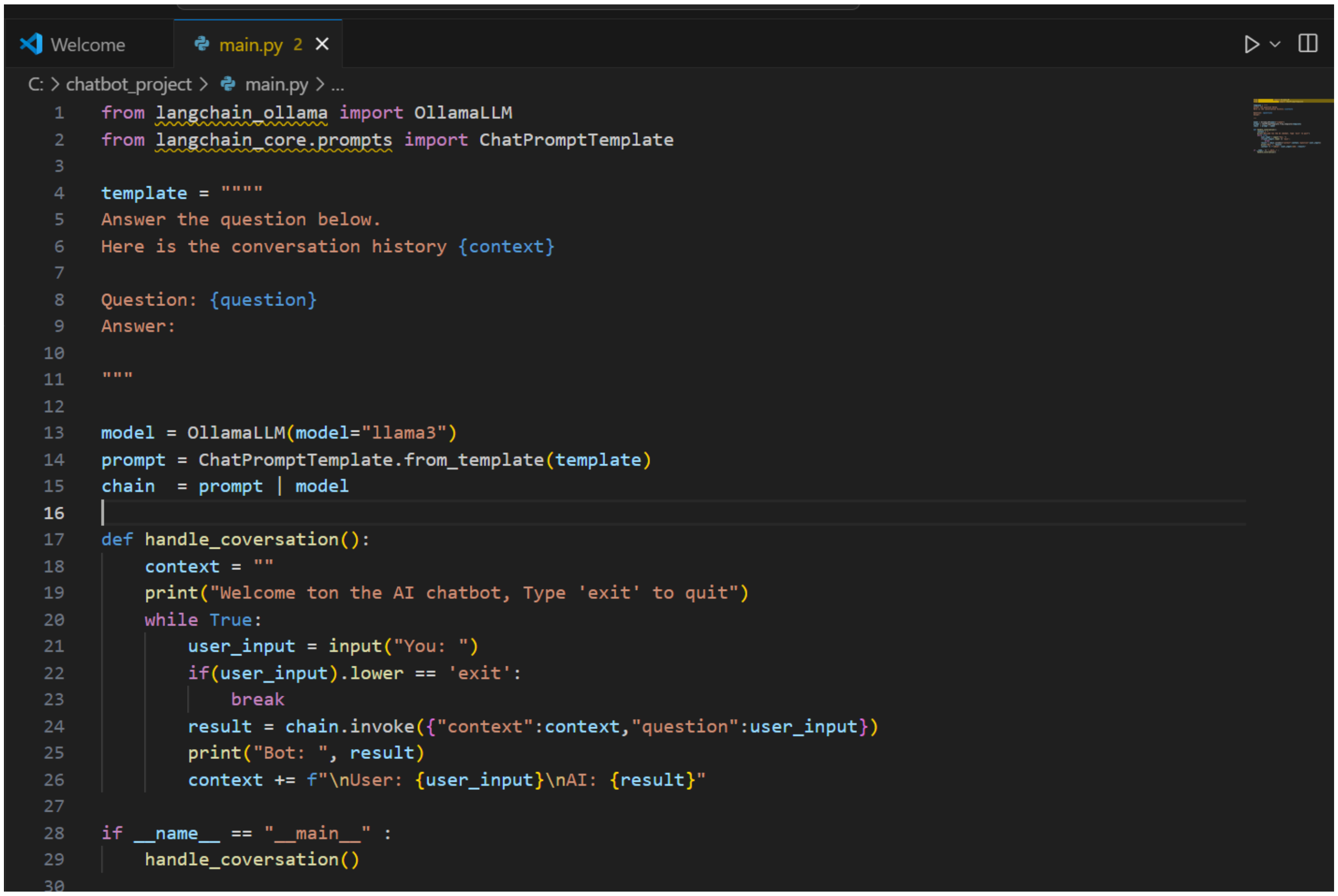


```python
from langchain_ollama import OllamaLLM
from langchain_core.prompts import ChatPromptTemplate

template = """
Answer the question below.
Here is the conversation history {context}

Question: {question}
Answer:

"""

model = OllamaLLM(model="llama3")
prompt = ChatPromptTemplate.from_template(template)
chain  = prompt | model
result = chain.invoke({"context":"","question":"hey how are you"})
print(result)
```

```
r, all text)!
(chatbot_env) PS C:\chatbot_project> python main.py
I'm good, thanks for asking!
```

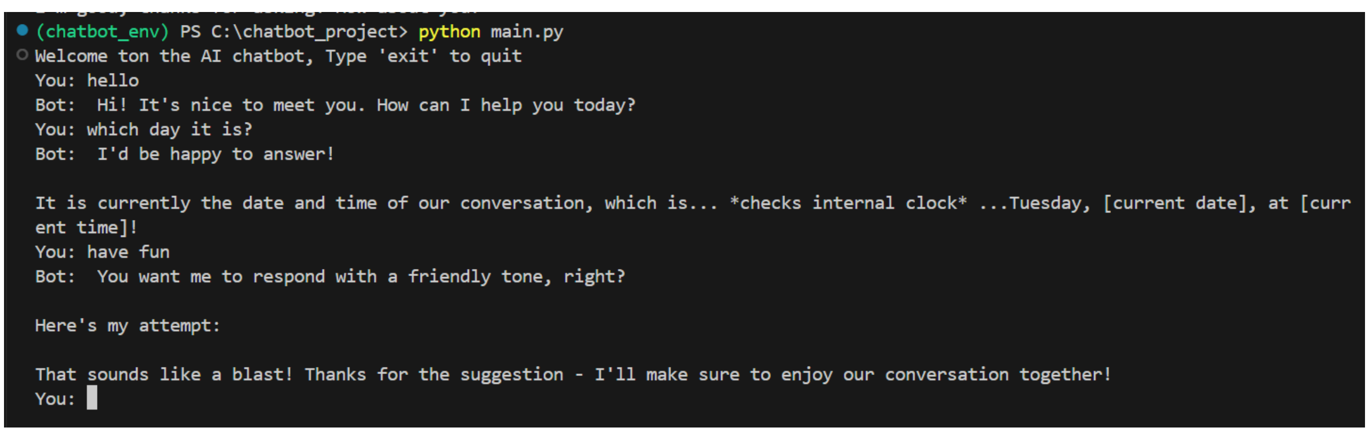
How to continually talk with the model and store a conversation history

We need to create a function called handle and conversation. This is where we'll kind of put all of our main code inside.



```python
from langchain_ollama import OllamaLLM
from langchain_core.prompts import ChatPromptTemplate

template = """
Answer the question below.
Here is the conversation history {context}

Question: {question}
Answer:

"""

model = OllamaLLM(model="llama3")
prompt = ChatPromptTemplate.from_template(template)
chain  = prompt | model

def handle_coversation():
    context = ""
    print("Welcome ton the AI chatbot, Type 'exit' to quit")
    while True:
        user_input = input("You: ")
        if(user_input).lower == 'exit':
            break
        result = chain.invoke({"context":context,"question":user_input})
        print("Bot: ", result)
        context += f"\nUser: {user_input}\nAI: {result}"

if __name__ == "__main__" :
    handle_coversation()
```

Chatbot interaction in terminal



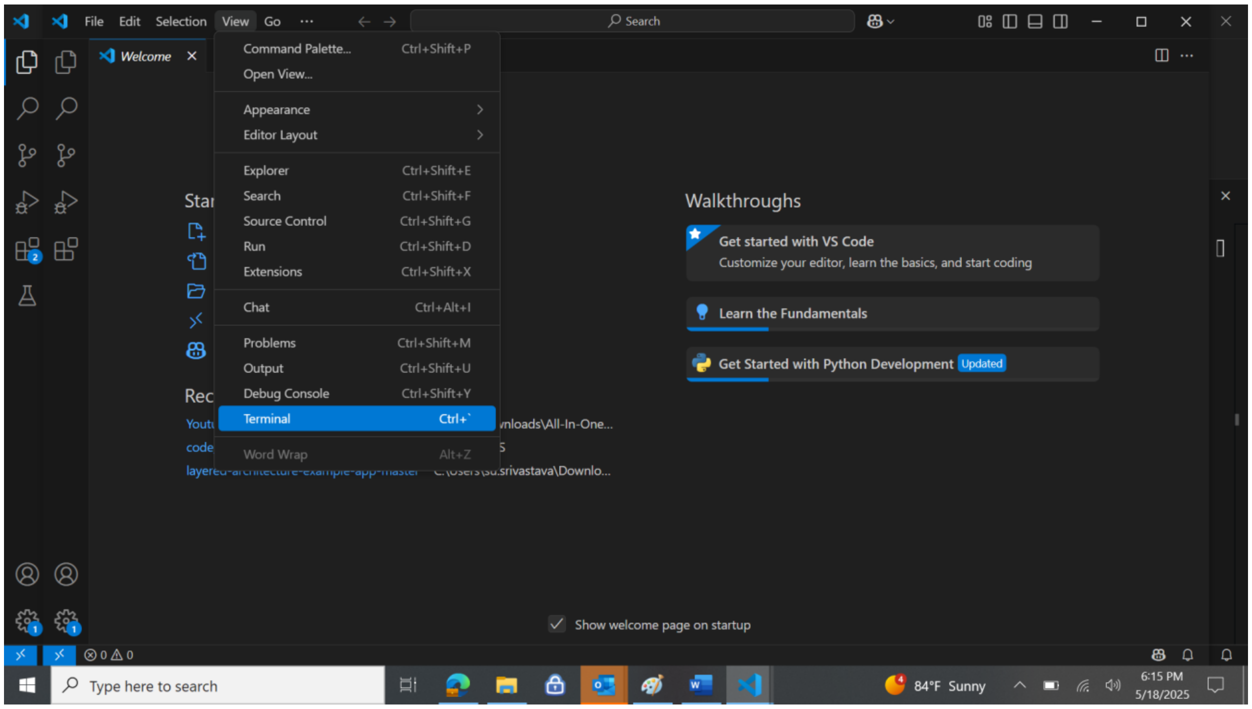
```
(chatbot_env) PS C:\chatbot_project> python main.py
Welcome ton the AI chatbot, Type 'exit' to quit
You: hello
Bot:  Hi! It's nice to meet you. How can I help you today?
You: which day it is?
Bot:  I'd be happy to answer!

It is currently the date and time of our conversation, which is... *checks internal clock* ...Tuesday, [current date], at [curr
ent time]!
You: have fun
Bot:  You want me to respond with a friendly tone, right?

Here's my attempt:

That sounds like a blast! Thanks for the suggestion - I'll make sure to enjoy our conversation together!
You: 
```

if __name__ == "__main__" : - this means whenever main.py is invoked.

Creating a **RAG (Retrieval-Augmented Generation)** chatbot project using **Ollama** (to run local LLMs like LLaMA2, Mistral, etc.) and retrieving knowledge from a **website** involves these key steps:

We'll:

1. **Scrape** a website.
2. **Split** the content.
3. **Embed** it using a local embedding model.
4. **Store** in a local vector DB (like Chroma).
5. **Retrieve** relevant chunks.
6. **Use Ollama** for generating responses.
7. Wrap it into a **chatbot** interface (CLI or optionally Streamlit).



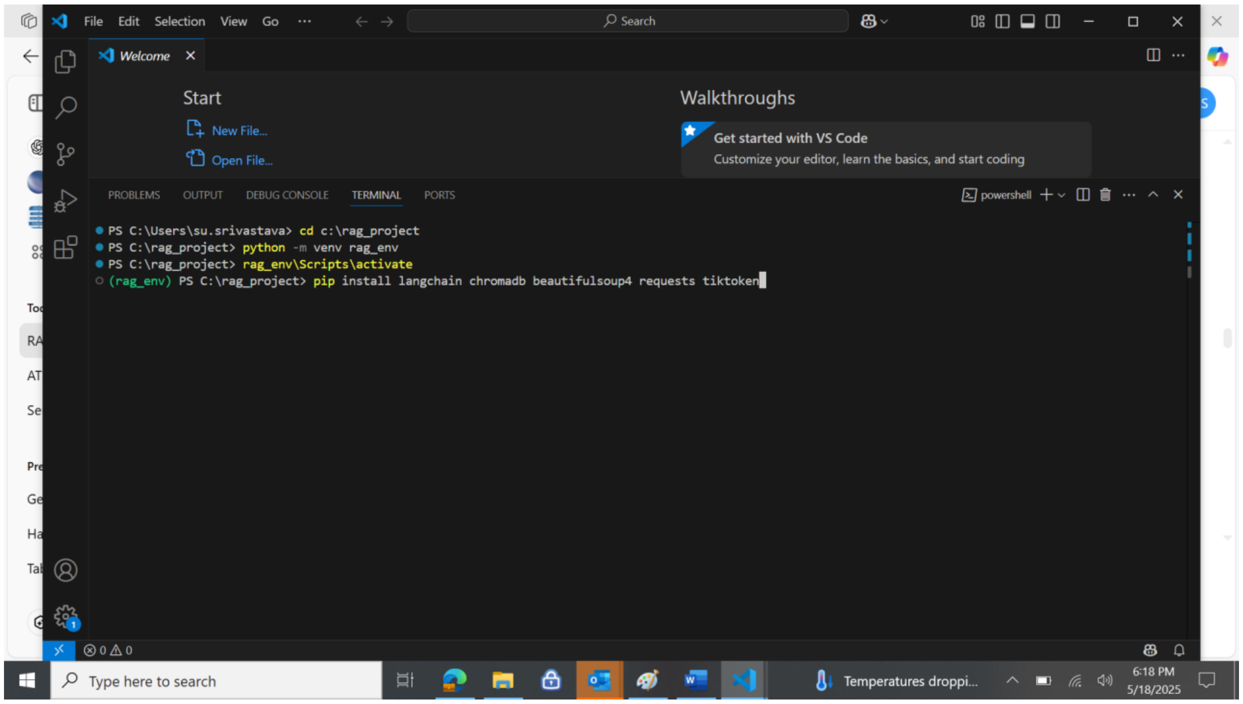
Run below in Visual studio code' s Terminal

Go to directory - PS C:\Users\su.srivastava> **cd c:\rag_project**

Create virtual env - PS C:\chatbot_project> **python -m venv rag_env**

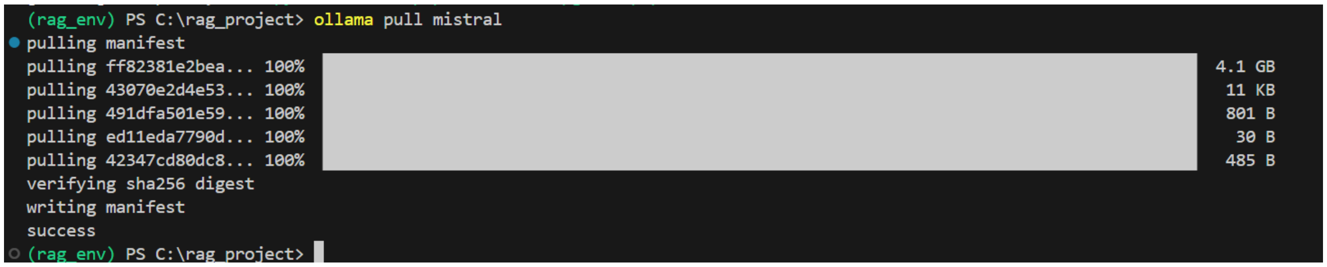
Activate virtual env - PS C:\chatbot_project> **rag_env\Scripts\activate**

Install libraries – (chatbot_env) PS C:\chatbot_project> **pip install langchain chromadb beautifulsoup4 requests tiktoken**



```
PS C:\Users\su.srivastava> cd c:\rag_project
PS C:\rag_project> python -m venv rag_env
PS C:\rag_project> rag_env\Scripts\activate
(rag_env) PS C:\rag_project> pip install langchain chromadb beautifulsoup4 requests tiktoken
```

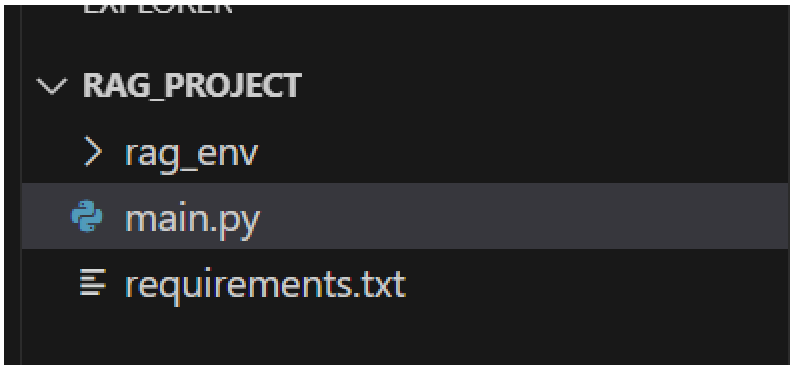
Install **Ollama**: https://ollama.com/download
Once installed, pull a model: **ollama pull mistral**



```
(rag_env) PS C:\rag_project> ollama pull mistral
pulling manifest
pulling ff82381e2bea... 100%                                      4.1 GB
pulling 43070e2d4e53... 100%                                       11 KB
pulling 491dfa501e59... 100%                                      801 B
pulling ed11eda7790d... 100%                                       30 B
pulling 42347cd80dc8... 100%                                      485 B
verifying sha256 digest
writing manifest
success
(rag_env) PS C:\rag_project>
```

Install --pip install -U langchain-community

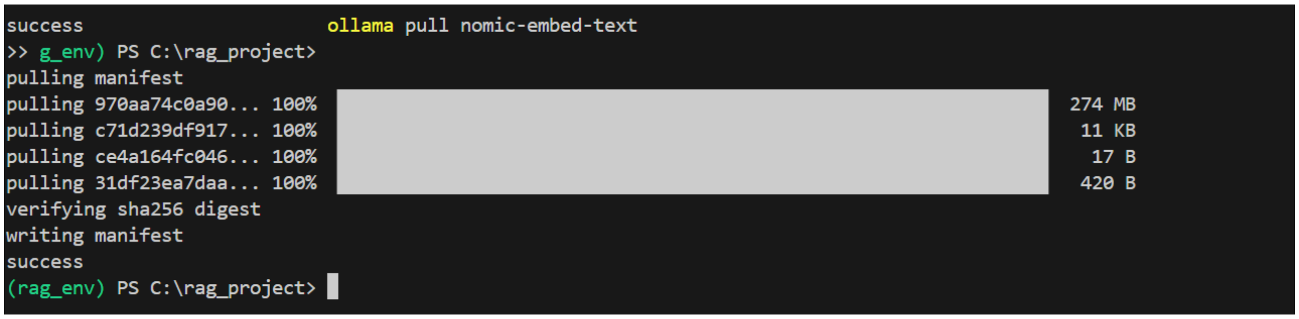
Folder structure



```
RAG_PROJECT
  > rag_env
    main.py
    requirements.txt
```

Before running, pull the embedding model:

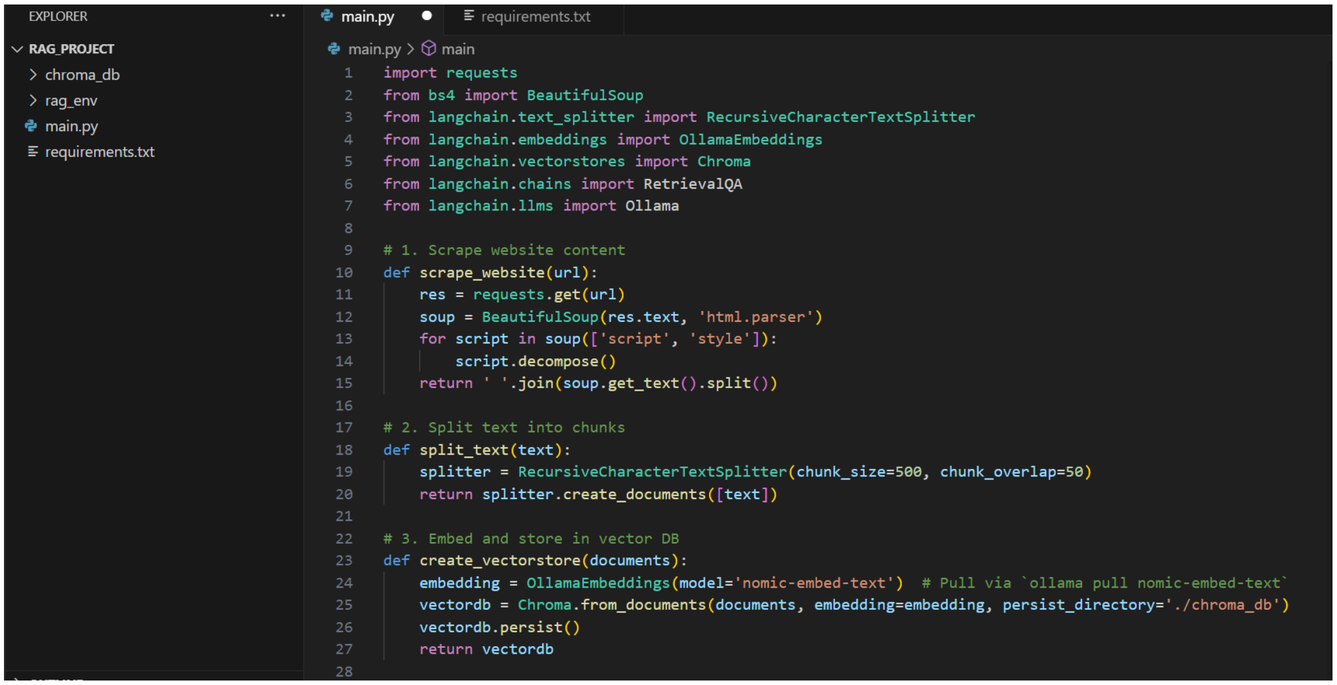
ollama pull nomic-embed-text



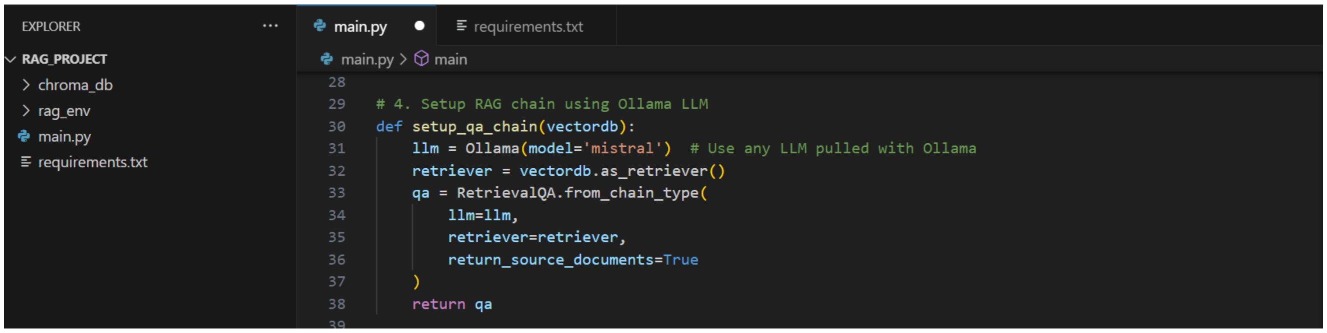
```
success                        ollama pull nomic-embed-text
>> g_env) PS C:\rag_project>
pulling manifest
pulling 970aa74c0a90... 100%                                                 274 MB
pulling c71d239df917... 100%                                                  11 KB
pulling ce4a164fc046... 100%                                                  17 B
pulling 31df23ea7daa... 100%                                                 420 B
verifying sha256 digest
writing manifest
success
(rag_env) PS C:\rag_project>
```

Run chatbot python main.py

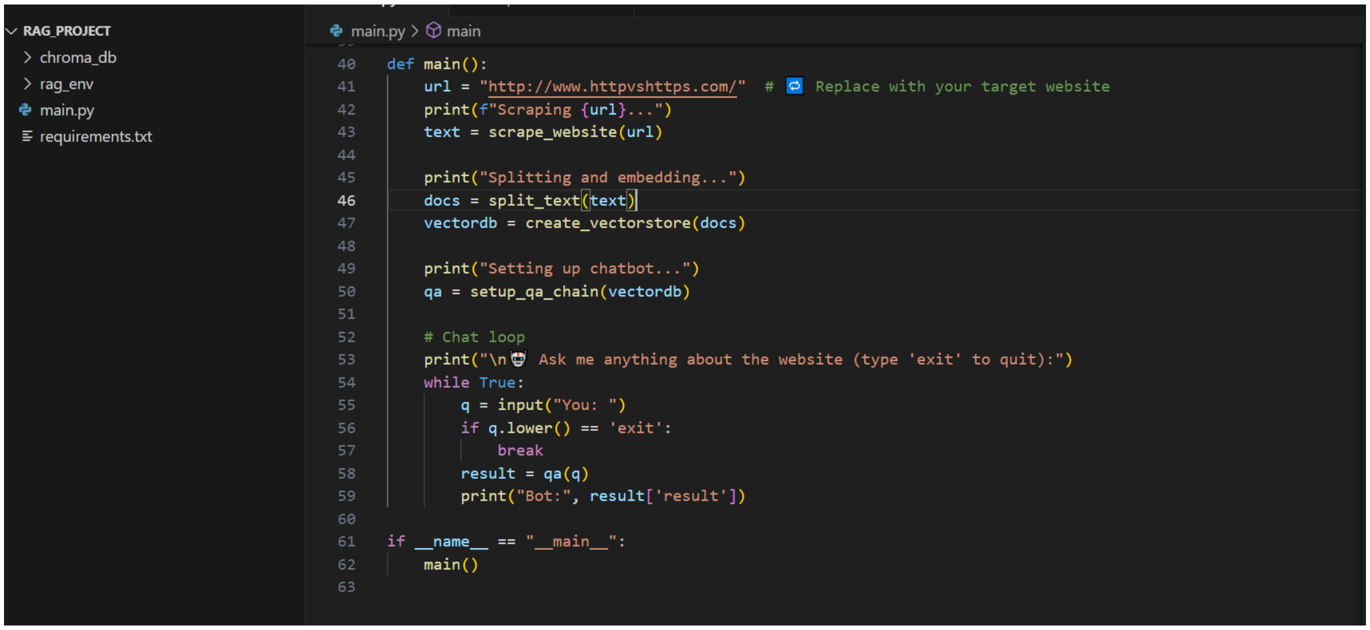
Complete code



```python
import requests
from bs4 import BeautifulSoup
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import OllamaEmbeddings
from langchain.vectorstores import Chroma
from langchain.chains import RetrievalQA
from langchain.llms import Ollama

# 1. Scrape website content
def scrape_website(url):
    res = requests.get(url)
    soup = BeautifulSoup(res.text, 'html.parser')
    for script in soup(['script', 'style']):
        script.decompose()
    return ' '.join(soup.get_text().split())

# 2. Split text into chunks
def split_text(text):
    splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
    return splitter.create_documents([text])

# 3. Embed and store in vector DB
def create_vectorstore(documents):
    embedding = OllamaEmbeddings(model='nomic-embed-text')  # Pull via `ollama pull nomic-embed-text`
    vectordb = Chroma.from_documents(documents, embedding=embedding, persist_directory='./chroma_db')
    vectordb.persist()
    return vectordb

```



```python

# 4. Setup RAG chain using Ollama LLM
def setup_qa_chain(vectordb):
    llm = Ollama(model='mistral')  # Use any LLM pulled with Ollama
    retriever = vectordb.as_retriever()
    qa = RetrievalQA.from_chain_type(
        llm=llm,
        retriever=retriever,
        return_source_documents=True
    )
    return qa
```

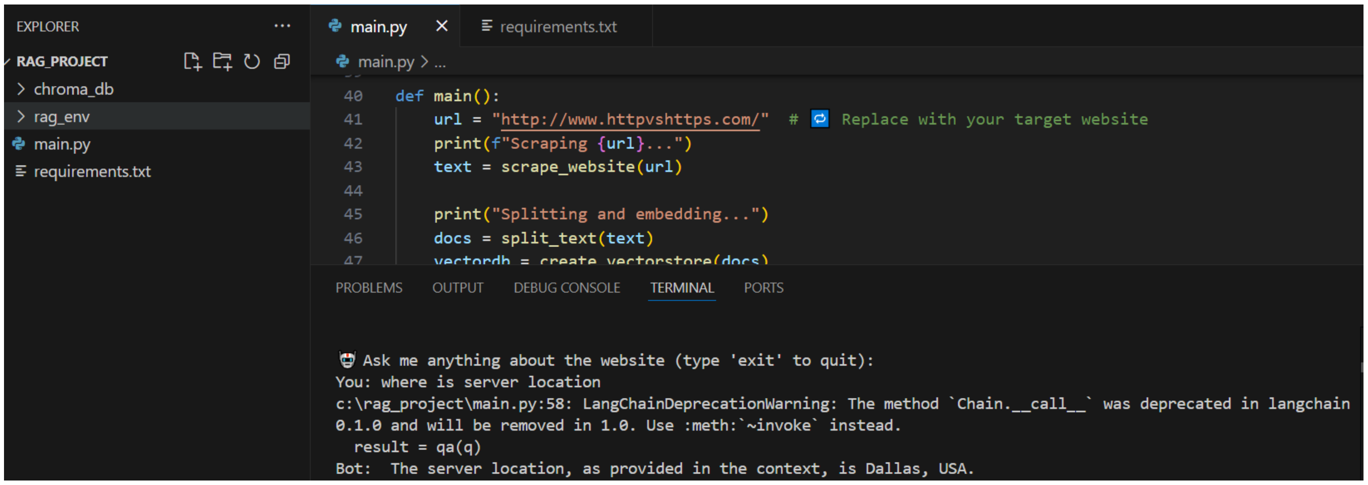


RAG_PROJECT
chroma_db
rag_env
main.py
requirements.txt

main.py > main

```python
def main():
    url = "http://www.httpvshttps.com/"  # 🔁 Replace with your target website
    print(f"Scraping {url}...")
    text = scrape_website(url)

    print("Splitting and embedding...")
    docs = split_text(text)
    vectordb = create_vectorstore(docs)

    print("Setting up chatbot...")
    qa = setup_qa_chain(vectordb)

    # Chat loop
    print("\n🤖 Ask me anything about the website (type 'exit' to quit):")
    while True:
        q = input("You: ")
        if q.lower() == 'exit':
            break
        result = qa(q)
        print("Bot:", result['result'])

if __name__ == "__main__":
    main()

```

Terminal interaction with bot



EXPLORER
RAG_PROJECT
chroma_db
rag_env
main.py
requirements.txt

main.py × requirements.txt

main.py > ...

```python
def main():
    url = "http://www.httpvshttps.com/"  # 🔁 Replace with your target website
    print(f"Scraping {url}...")
    text = scrape_website(url)

    print("Splitting and embedding...")
    docs = split_text(text)
    vectordb = create_vectorstore(docs)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
🤖 Ask me anything about the website (type 'exit' to quit):
You: where is server location
c:\rag_project\main.py:58: LangChainDeprecationWarning: The method `Chain.__call__` was deprecated in langchain
0.1.0 and will be removed in 1.0. Use :meth:`~invoke` instead.
  result = qa(q)
Bot:  The server location, as provided in the context, is Dallas, USA.
```