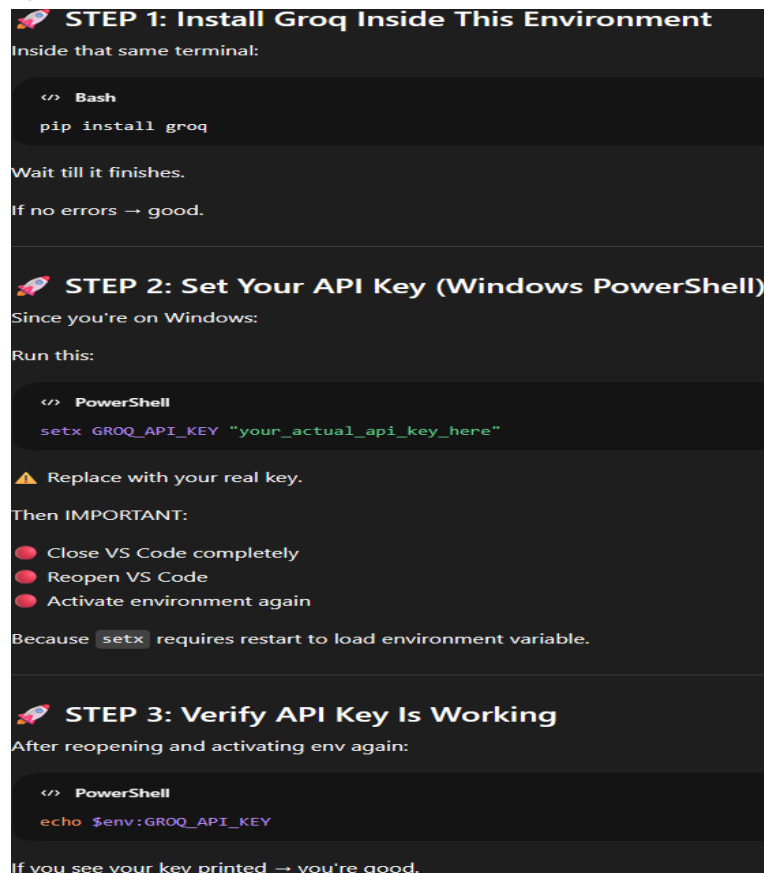


RESEARCH AGENT NOTES

1. Read the pdf in the folder to understand what the project is and how it is supposed to function.

Steps:

1. Firstly, open VS Code and create the folder structure for this project.
2. Activate the Anaconda environment. This is very important to do at each step, and also make sure to select the right Python interpreter(ml_dl_env).
3. Went on Groq and made a new account to get a free API Key to use Ollama LLM for the parts of the projects where it is needed.
4. Then we set up the API key we got from Groq to be able to work in our project. Here is a screenshot of that process.



5. Then, inside the [planner.py](#) file, we set up our LLM model using the API key. What is that model going to do?

- a. Firstly, we are using the **Ollama** model as our LLM.
 - b. In this file we set up a prompt for breaking our research topic into relevant subtopics, and then we have the LLM return that to us.
6. Now the Model has been setup and everything works. If we give a topic in the file and ask the LLM to do the work, it does so and returns the list of subtopics in the terminal.
7. From here, we will see what is in each file:
- a. [Orchestrator.py](#): This file is the big BOSS and controls every agent. We call each agent in here wherever they are needed(planner, writer, and more)
 - b. [Planner.py](#): This file contains the code that makes up our “RESEARCH ASSISTANT AGENT.” In this file, we give the LLM our topic of interest, and it breaks into multiple sub-topics that it will later on do research on.
 - c. [Writer.py](#): This file contains the code that makes up our “PROFESSIONAL RESEARCH ANALYST AGENT.” In this file, we pass in our sub-topics(and the research we did on the topics using our search **tool**) and the original topic that we have initially, and it generates a PDF-type report on each sub-topic and goes in depth. **This file also gets information from our search_tools.py file.** More details about the search_tools.py are below.
 - d. [Search_tools.py](#): This file contains one of our many “Tools” that we are going to use in this agent. In this file, we are doing nothing but using it as a search engine similar to Google. We are temporarily using the AI to do this search, but later in the future, we can replace it with an API of an actual search engine. In this file, WE ARE USING IT LIKE A SEARCH ENGINE TO LOOK UP MORE INFORMATION ON THE SUBTOPICS GENERATED FROM THE [PLANNER.PY](#) FILE.

Implementing RAG

8. First, we install the Transformer model. Using the command:

- a. pip install chromadb sentence-transformers
 - i. This transformer is similar to the one from the research paper but this one is a small encoder because in our case we are basically only using it to **convert text to vectors**.
 - 9. Then we create a new folder called “rag” and in here we create all the files regarding the RAG system we are implementing.
 - a. `__init__.py`
 - b. `Vector_stores.py`
 - 10. In the `vector_stores.py` file, we used the “SentenceTransformer” model to implement the RAG system that stores regular text into vectors. To learn more about this logic look in this file more.
-

Converting the report into a PDF file

- 11. Currently, the [writer.py](#) file is generating the report and printing it in the terminal. We don't want that.
 - 12. So we will use the built-in library called reportlab to generate a pdf instead of printing in the terminal.
 - 13. Command to install the library:
 - a. `pip install reportlab`
 - 14. Once we installed that, we then created a new folder called “utils” and inside that we created a file called “pdf_generator.py”
 - 15. Then, in the `pdf_generator.py` file, we added code that turns the report into a pdf file. It's basically a function we defined, and then we call that function inside the [Orchestrator.py](#) file to generate the PDF.
-

Agent Completed