**2. [25 points]**

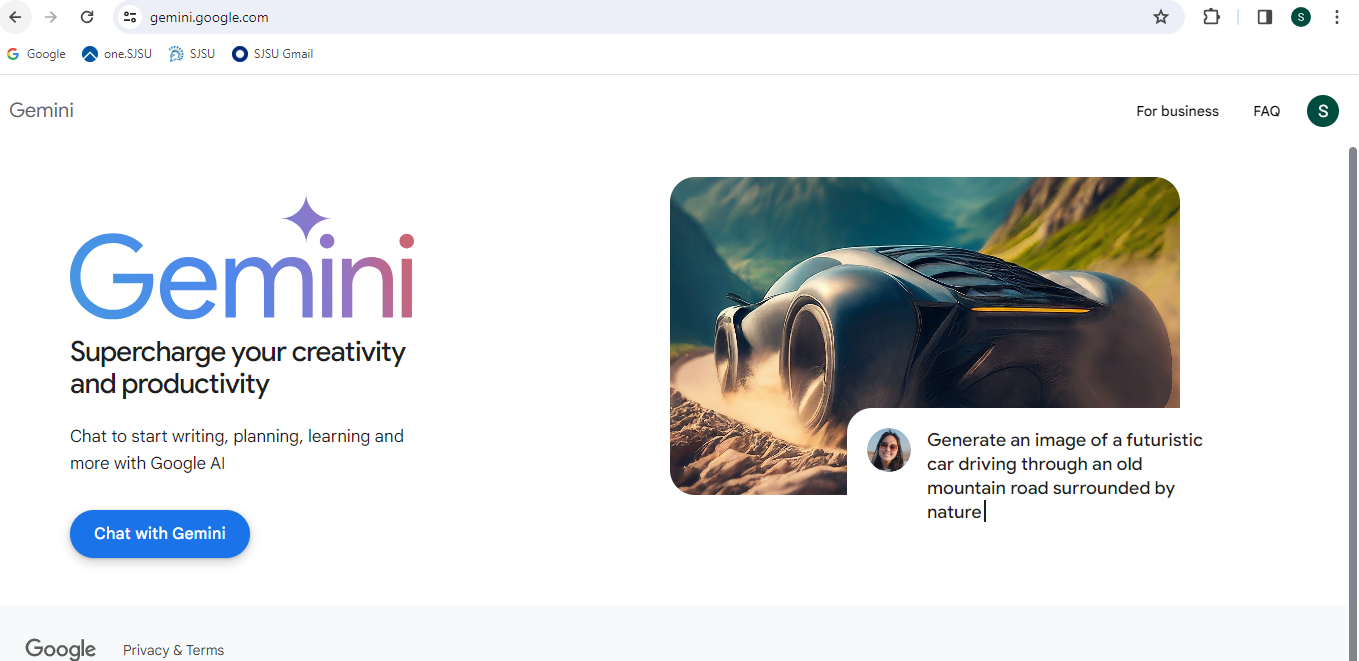
**Building a Code Query System with LLMs and LangChain for GitHub Repositories for Apache Spark, and other codebases that you manually examined earlier.**

**Objectives:**

* Gain practical experience with LLMs (Large Language Models) and LangChain for code comprehension.
* Develop a system that can answer user queries about the code within the Apache Spark and other GitHub repositories, in a way automating some of what you did in earlier HW.

**2.1 [5 points] Setup**

**Choose an LLM:**

Gemini is Google's newest family of Large Language Models (LLMs). The Gemini suite currently houses 3 different model sizes: Nano, Pro, and Ultra.

Advantage:

High-Quality Text Generation: Gemini, being a state-of-the-art language model, can generate high-quality text across various domains. It can understand context, generate coherent responses, and provide valuable insights.

Versatility: Gemini can be used for a wide range of **natural language processing (NLP) tasks, including text generation, summarization, translation, question answering, and more**. Its versatility makes it suitable for diverse applications.

Efficiency: Gemini is capable of **generating text quickly**, which can be advantageous in scenarios where real-time or near-real-time responses are required. It can handle large volumes of text efficiently.

Customization: Gemini allows fine-tuning on specific datasets or tasks, enabling users to customize its behavior for particular applications. **Fine-tuning can improve performance on domain-specific tasks.**

Access to Latest Research: OpenAI regularly updates its models based on the latest research advancements. Users of Gemini benefit from these updates, which may include improvements in text quality, efficiency, and capabilities.

Disadvantage:

Compute Resources: Training and fine-tuning large language models like **Gemini require significant computational resources, including powerful hardware and large-scale datasets**. Access to such resources may be limited for some users.

Environmental Impact: **Training and using large language models consume a considerable amount of energy,** contributing to carbon emissions and environmental impact. This aspect raises concerns about the sustainability of AI development.

Data Privacy: Large language models like Gemini are trained on vast amounts of text data, which may include sensitive or private information. There are concerns about data privacy and potential misuse of models for unethical purposes.

Bias and Fairness: Language models trained on diverse datasets may inherit biases present in the training data. Gemini may exhibit biases related to gender, race, or other sensitive attributes, which can lead to biased outputs and reinforce stereotypes.

Ethical Considerations: The widespread adoption of large language models raises ethical concerns related to misinformation, manipulation, and unintended consequences. Users of Gemini must be mindful of ethical considerations and responsible AI practices.

**Learning from LLM and Langchain:**

Language Understanding: Interacting with LLMs and LangChains can improve language comprehension skills as users formulate prompts and questions, and analyze responses to understand the model's reasoning and capabilities.

Creative Writing: Experimenting with LLMs and LangChains can foster creativity by generating stories, poems, or dialogues based on prompts. Users can explore different writing styles, genres, and narrative structures, enhancing their creative writing abilities.

Problem Solving: Using LLMs and LangChains to tackle complex problems can sharpen problem-solving skills. Users can formulate queries, analyze generated responses, and iterate on their approach to find solutions or insights across various domains.

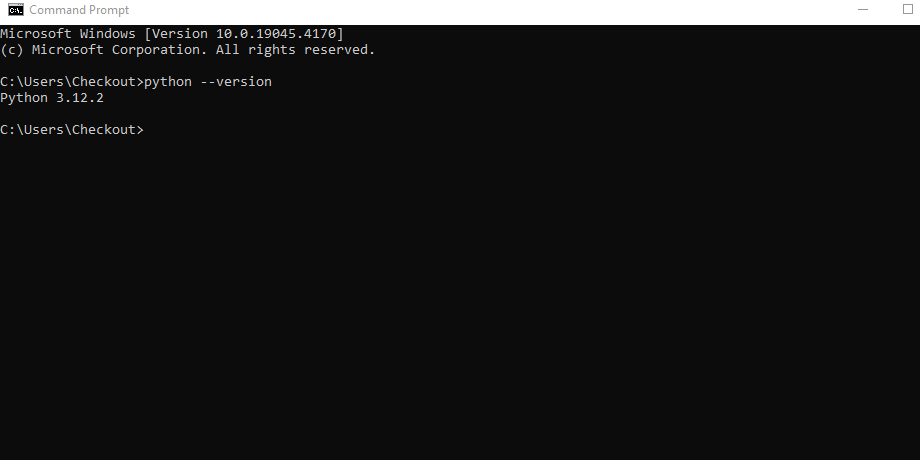
Critical Thinking: Evaluating the quality and relevance of responses generated by LLMs and LangChains encourages critical thinking. Users learn to discern between plausible and implausible answers, identify biases or inaccuracies, and refine their questions to elicit meaningful responses.

Programming and Automation: Leveraging LLMs and LangChains for code generation and automation tasks can enhance programming skills. Users can explore code snippets, generate scripts, and automate repetitive tasks, improving their coding proficiency and efficiency.

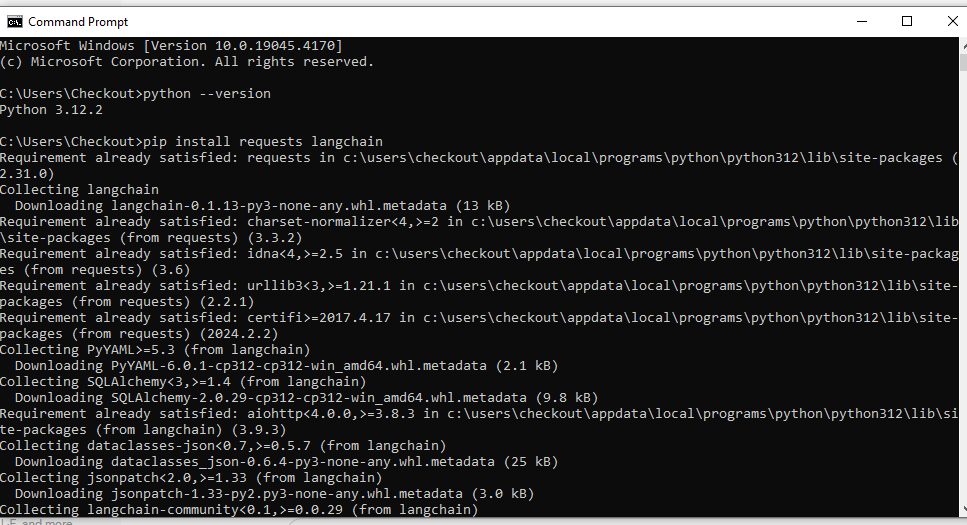
Research and Knowledge Acquisition: Engaging with LLMs and LangChains can facilitate research and knowledge acquisition across diverse topics. Users can pose questions, explore explanations, and delve into complex concepts, expanding their understanding and expertise in various domains.

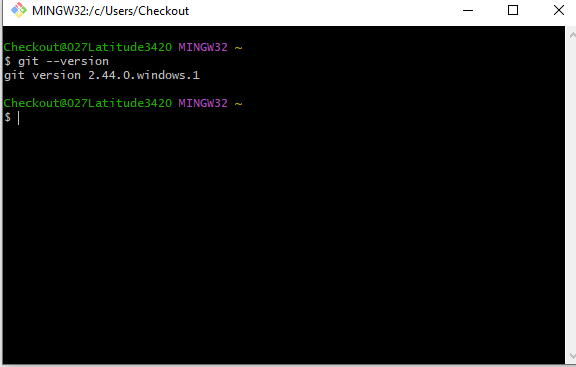
**Set Up Your Development Environment:**

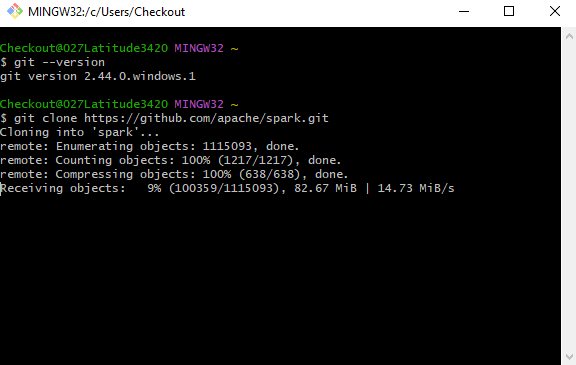
* Install Python (version 3.6 or later recommended).
* Install required libraries using pip:

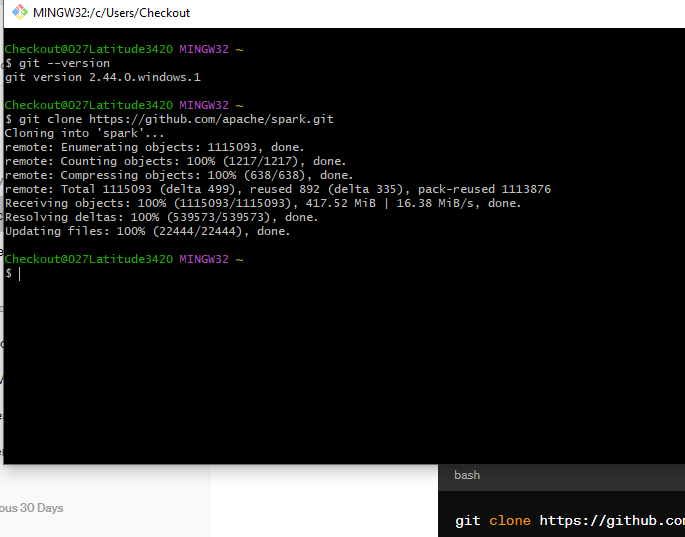


pip install requests langchain

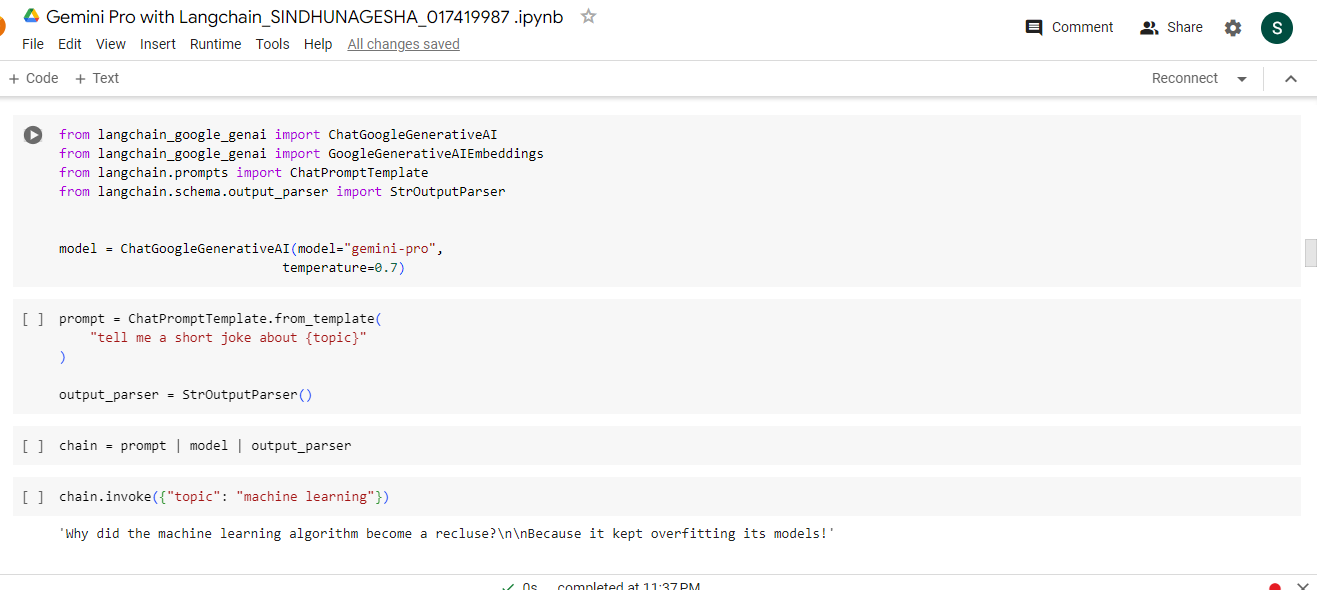








* Create a Python script to interact with the LLM and process queries.



**Prepare the Apache Spark GitHub Repository:**

* Clone the Apache Spark and other repositories locally using Git:

git clone https://github.com/apache/spark.git



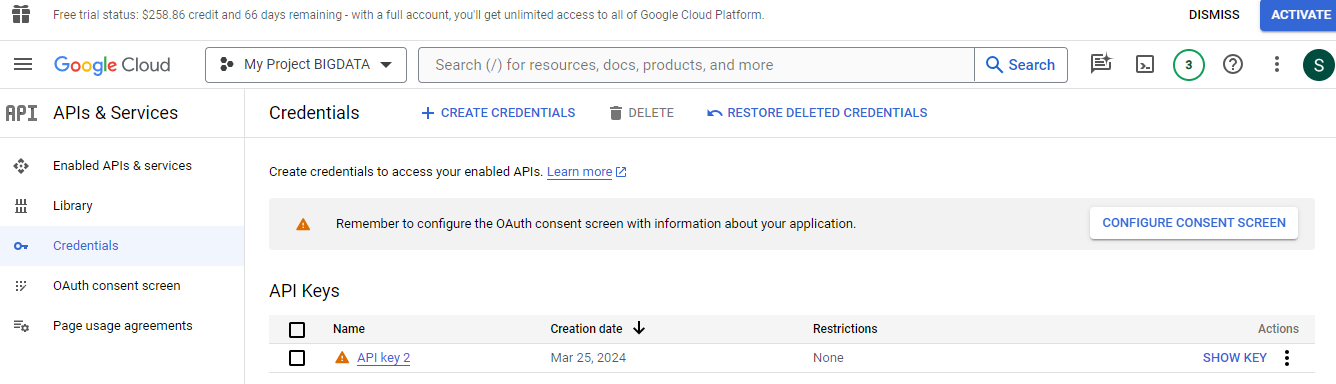
**Step 1: Importing required libraries:**

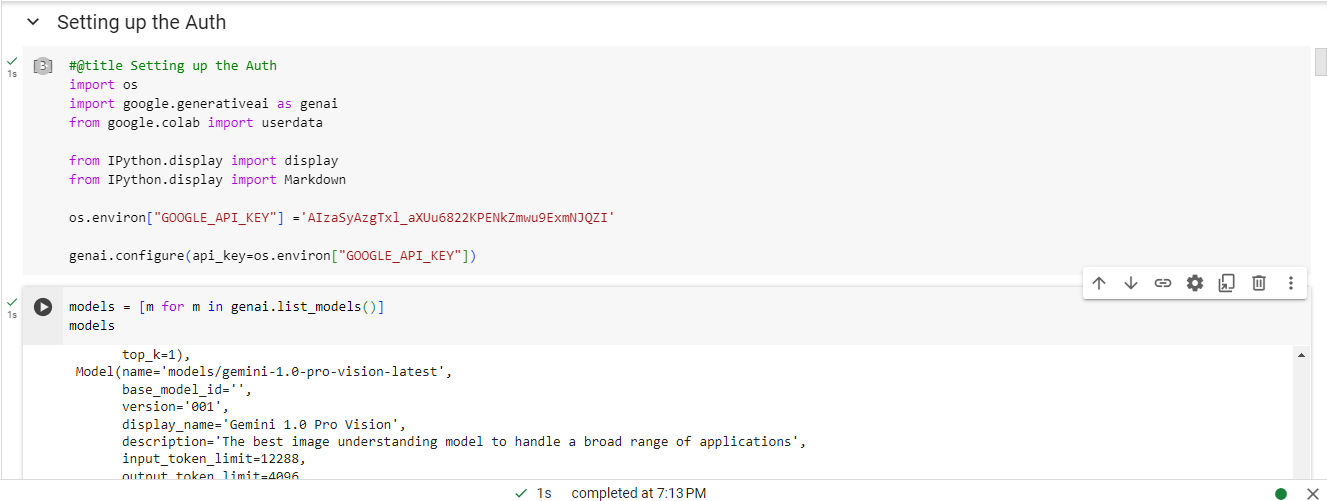




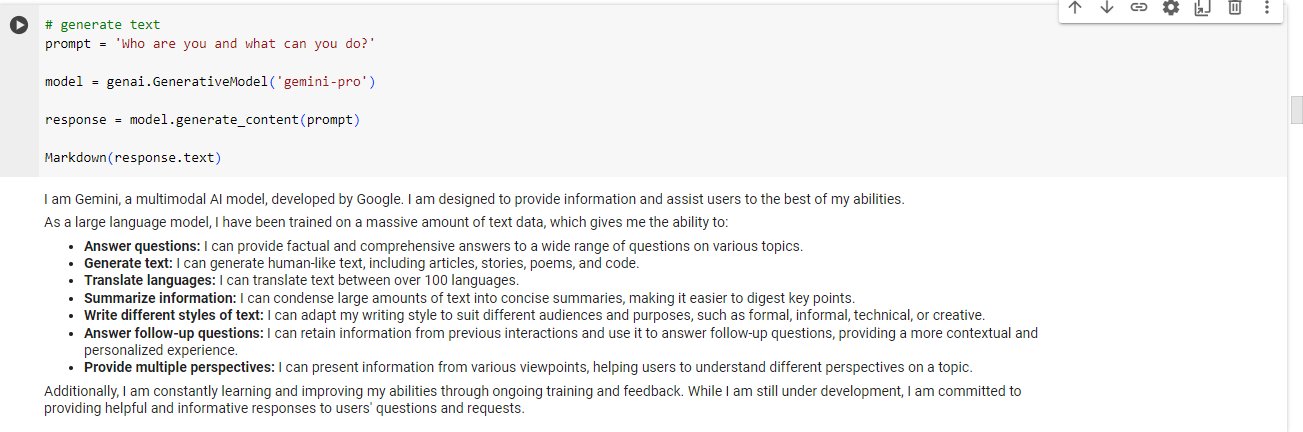
and many others according to the requirement.

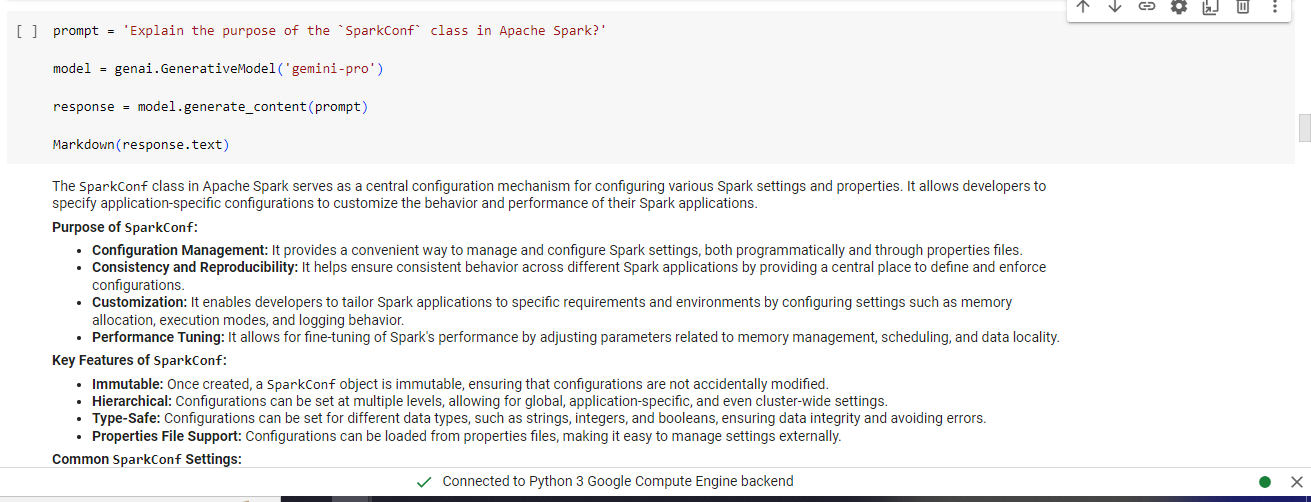
**Step 2: Setting up Authentication**

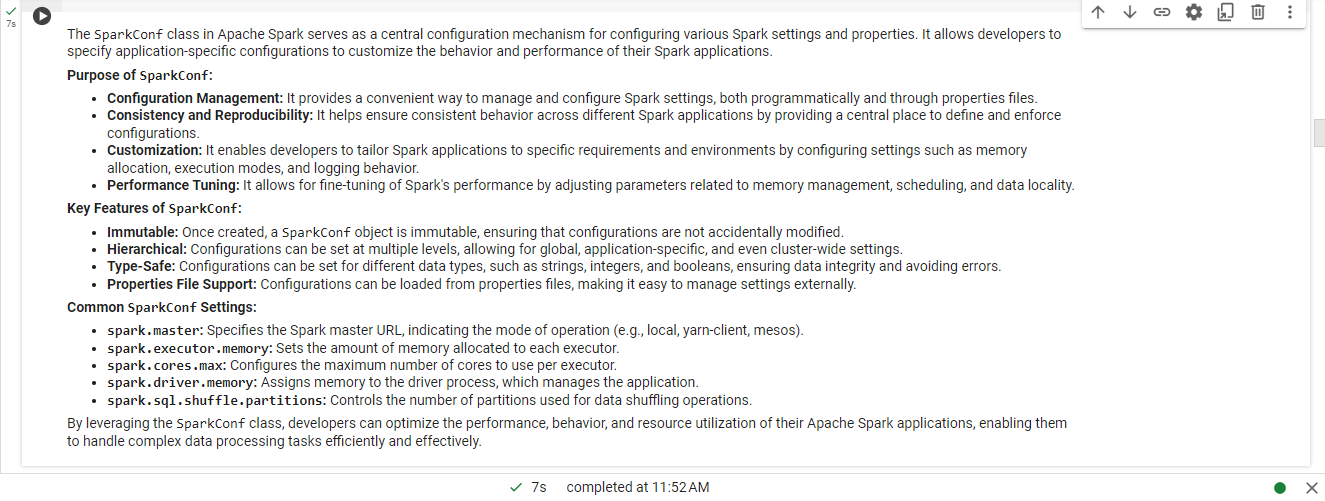


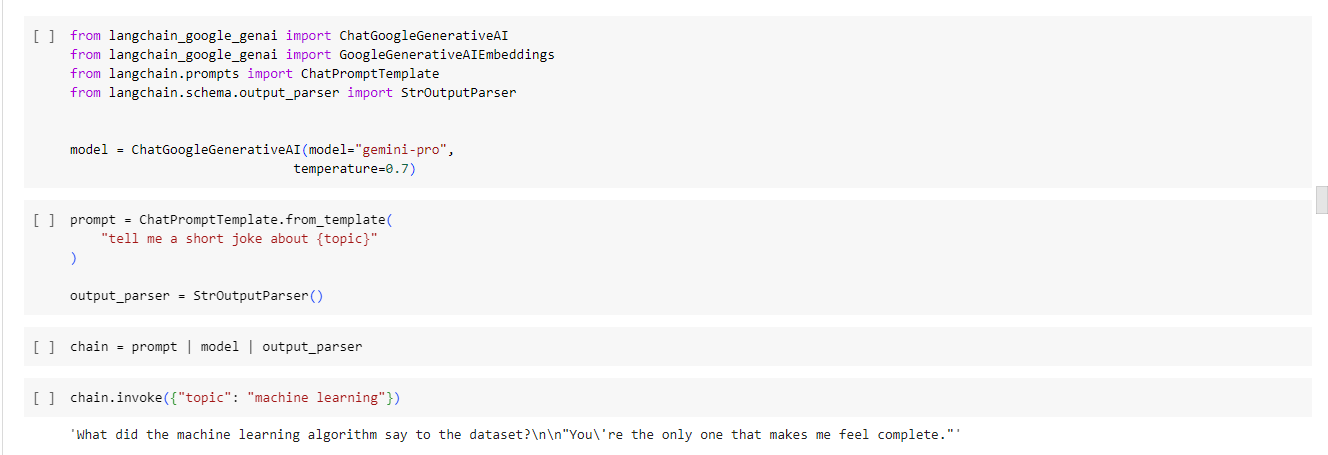


Step 3: Basic Interaction with LLM









**2.2 [10 points] Interact with the LLM:**

* Attempt to generate the takeaways that you manually extracted from the codebase(s) that you analyzed in the previous homework. Analyze the code for Apache Spark as well. Compare the insights generated with the ones that you manually generated earlier.

import requests

import google.generativeai as genai

def get\_repo\_stats(repo\_url):

"""

Function to get statistics for a given GitHub repository.

"""

# Extracting owner and repo name from the URL

parts = repo\_url.split('/')

owner = parts[-2]

repo\_name = parts[-1]

# Constructing the GitHub API URL

api\_url = f"https://api.github.com/repos/{owner}/{repo\_name}"

# Sending a GET request to the GitHub API

response = requests.get(api\_url)

if response.status\_code == 200:

repo\_data = response.json()

stars = repo\_data['stargazers\_count']

forks = repo\_data['forks\_count']

issues = repo\_data['open\_issues\_count']

return stars, forks, issues

else:

return None, None, None

def interact\_with\_codebase():

"""

Function to interact with a codebase and provide insights.

"""

repo\_url = "https://github.com/apache/spark"

stars, forks, issues = get\_repo\_stats(repo\_url)

if stars is not None:

print(f"The Apache Spark repository has {stars} stars, {forks} forks, and {issues} open issues.")

else:

print("Failed to fetch repository statistics. Please check the URL and try again.")

if \_\_name\_\_ == "\_\_main\_\_":

interact\_with\_codebase()

prompt = 'Explain the purpose of the `SparkConf` class in Apache Spark?'

model = genai.GenerativeModel('gemini-pro')

response = model.generate\_content(prompt)

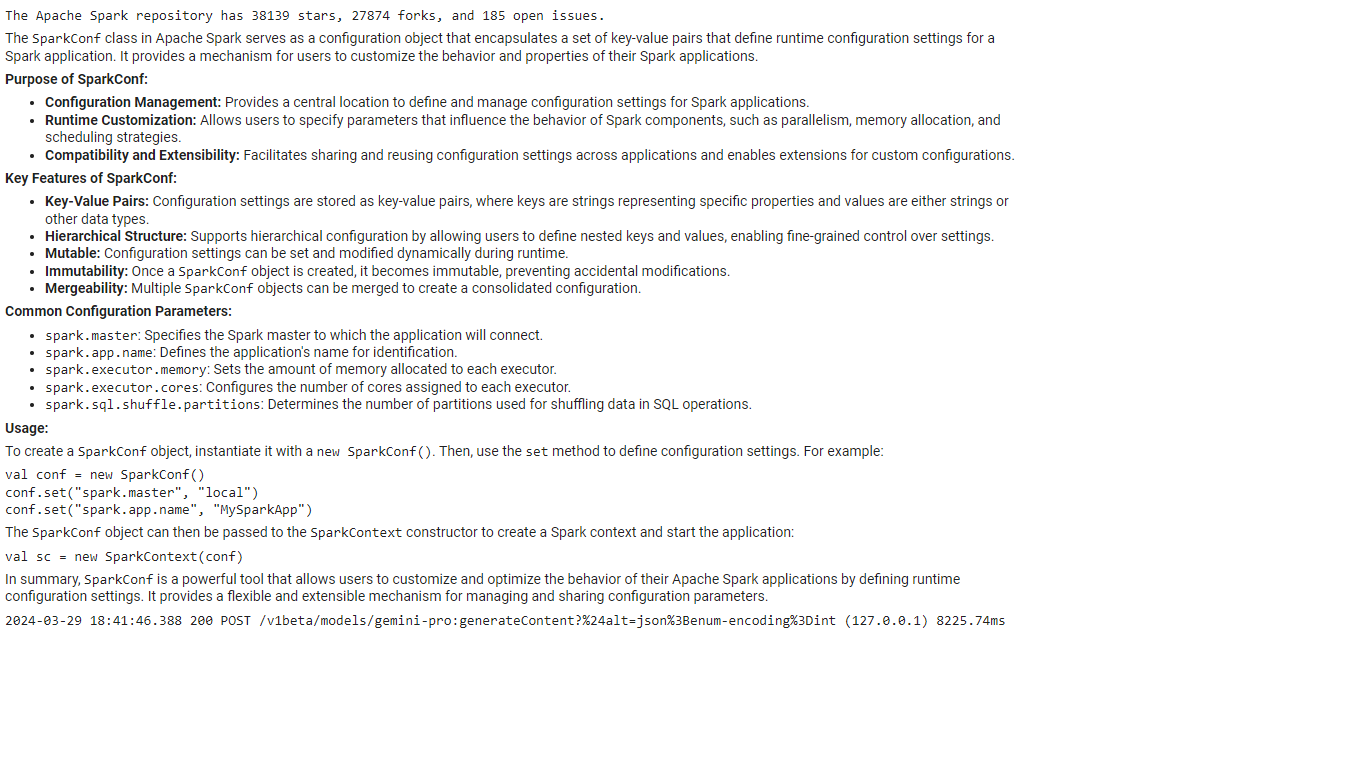
Markdown(response.text)

**Manual takeaways:**

Spark provides nice user APIs for computations on small datasets. Spark can scale these same code examples to large datasets on distributed clusters. It’s fantastic how Spark can handle both large and small datasets.

Spark also has an expansive API compared with other query engines. Spark allows you to perform DataFrame operations with programmatic APIs, write SQL, perform streaming analyses, and do machine learning. Spark saves you from learning multiple frameworks and patching together various libraries to perform an analysis.

**Gemini takeaways:**



**Manual Takeaways focuses on :**

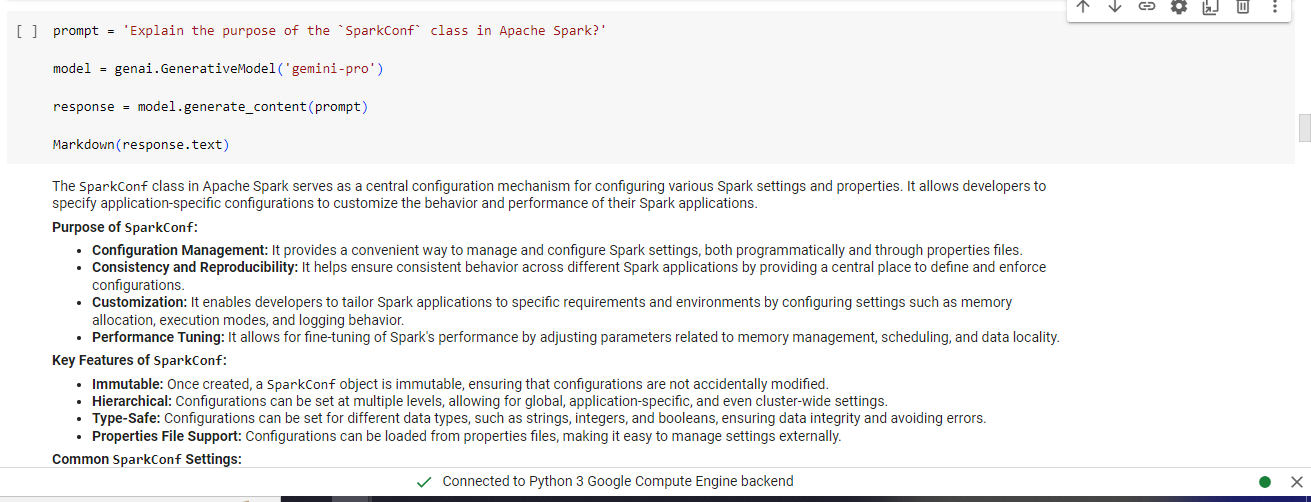
* Configuration Management: Provides a central location to define and manage configuration settings for Spark applications.
* Runtime Customization: Allows users to specify parameters influencing Spark components' behavior, such as parallelism, memory allocation, and scheduling strategies.
* Compatibility and Extensibility: Facilitates sharing and reusing configuration settings across applications and enables extensions for custom configurations.
* Key Features:
  + Key-Value Pairs
  + Hierarchical Structure
  + Mutable
  + Immutability
  + Mergeability
* Common Configuration Parameters: Examples such as spark.master, spark.app.name, spark.executor.memory, etc.
* Usage: Demonstrated instantiation and usage of SparkConf objects.

**AI-generated Takeaways focuses on :**

* Ease of Scaling: Spark's ability to scale from small to large datasets on distributed clusters.
* Expansive API: Highlights Spark's comprehensive API, covering DataFrame operations, SQL queries, streaming analysis, and machine learning.
* Unified Framework: Emphasizes Spark's advantage of providing a single framework for various types of analyses, saving users from having to learn multiple frameworks and libraries.

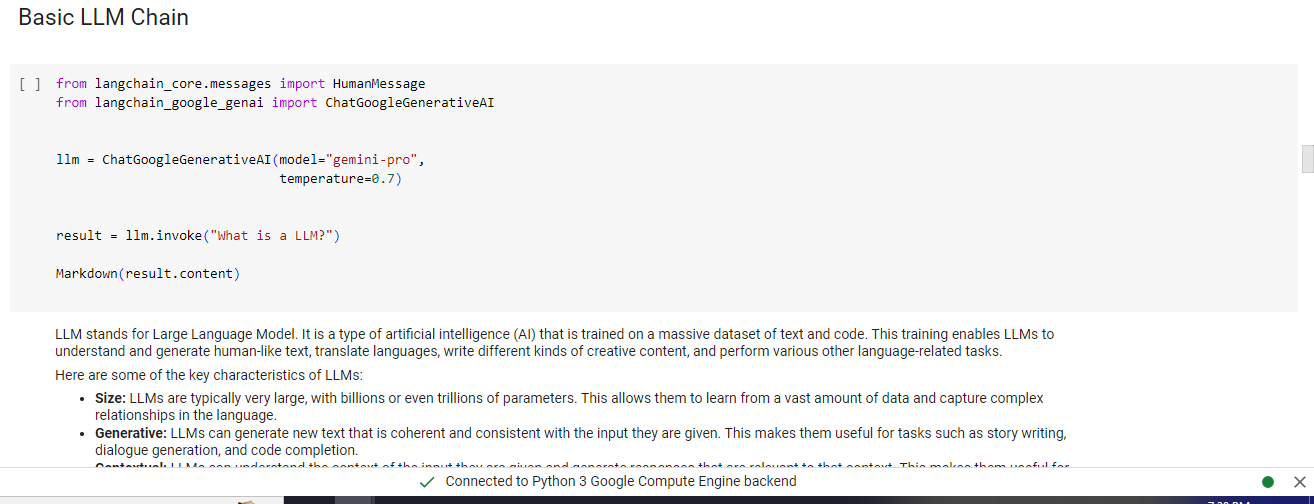
**Comparison:**

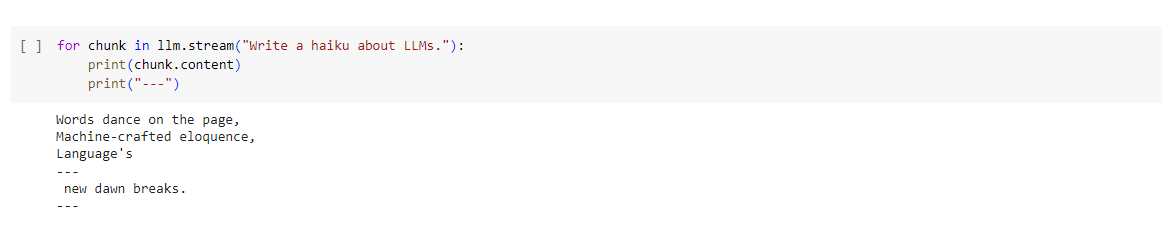
* Content Coverage: Both the manual and AI-generated takeaways cover aspects of SparkConf's purpose, key features, common parameters, and usage.
* Depth of Explanation: The manual takeaways delve deeper into the technical aspects of SparkConf, such as its hierarchical structure, immutability, and manageability. In contrast, the AI-generated takeaway focuses more on the broader advantages and capabilities of Spark, highlighting its scalability, expansive API, and unified framework.
* Technical vs. Practical Focus: The manual takeaways provide more technical insights into SparkConf's workings and usage. In contrast, the AI-generated takeaway emphasizes practical benefits and advantages of using Spark in data analysis.



**Process User Queries:**

* Design a function to handle user queries:

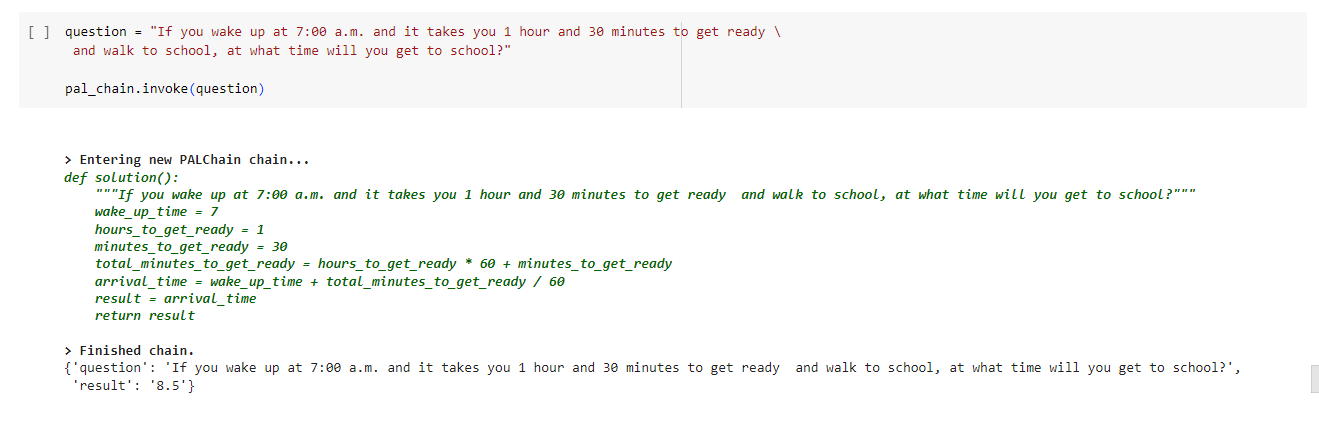


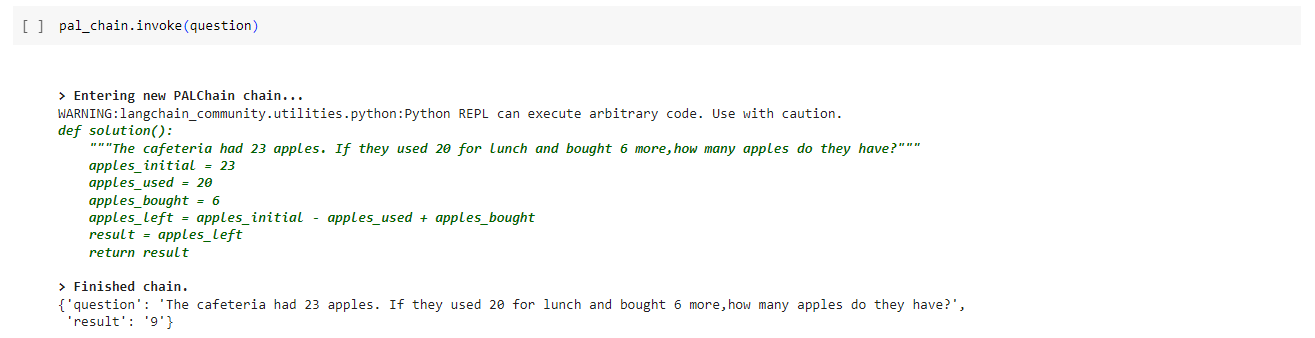


**2.3 [5 points] LangChain:**

* Assess the value addition of using LangChain

Math solutions:

****

****

* create a pipeline to process queries in a structured manner, potentially involving:
  + Tokenization (breaking down the query into words)

****

* + Named entity recognition (identifying code elements)

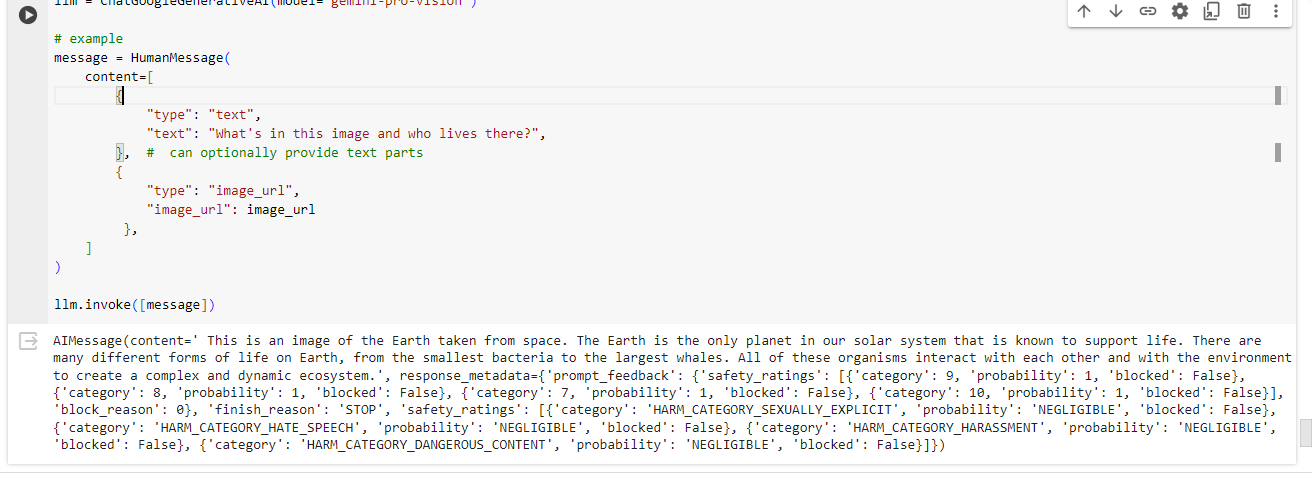
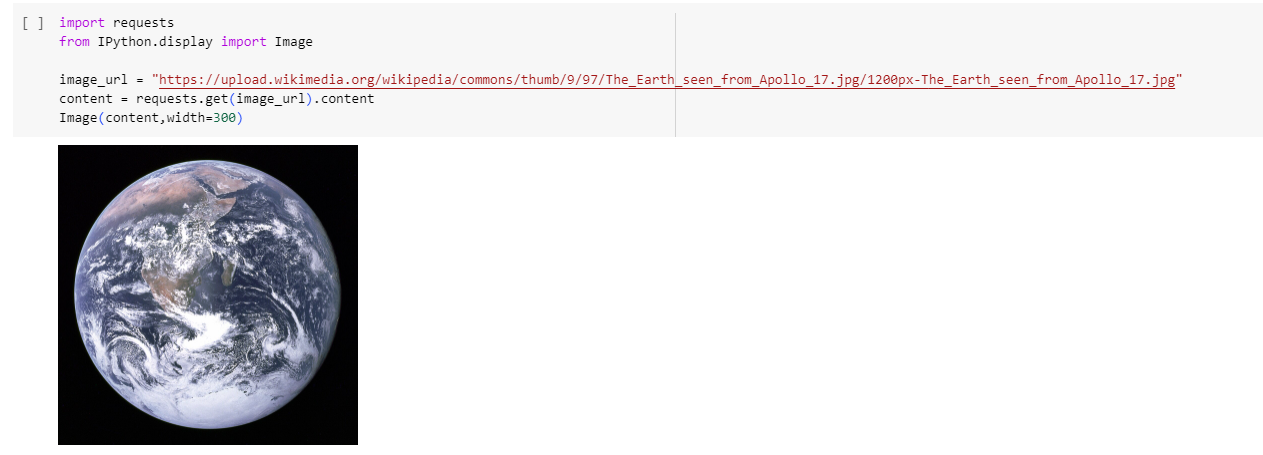
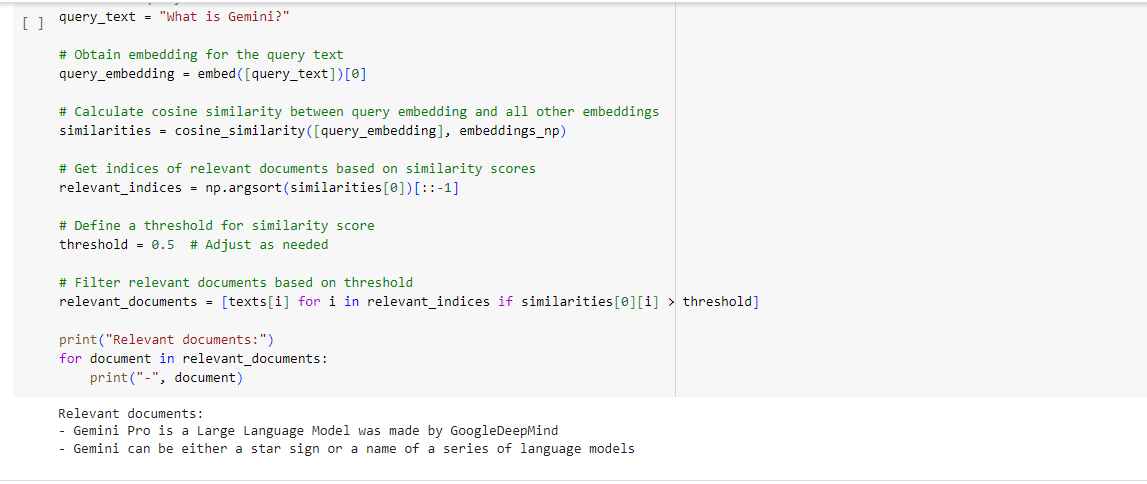
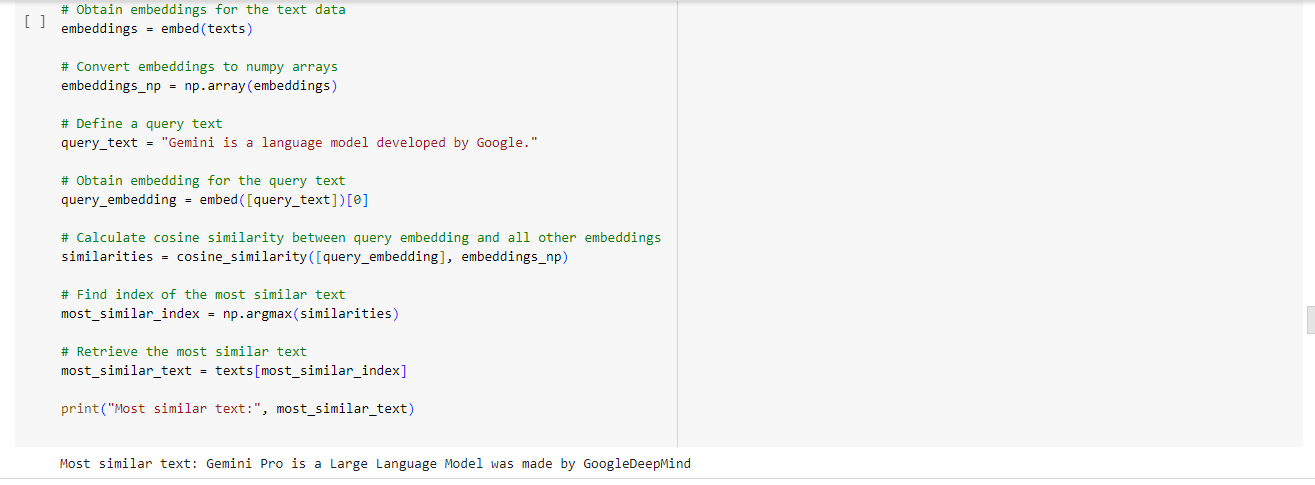


* + Relation extraction (understanding relationships between entities)

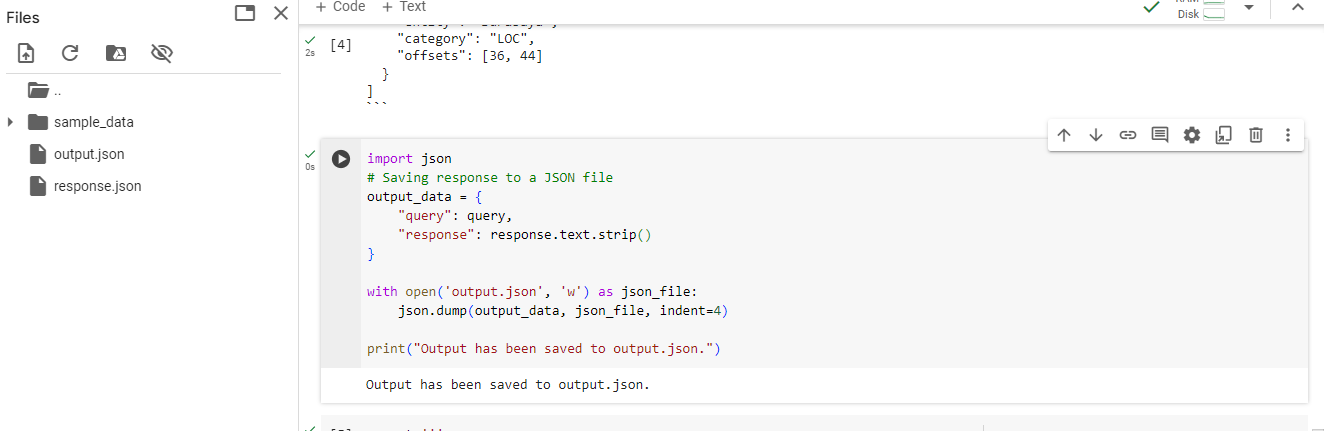
****

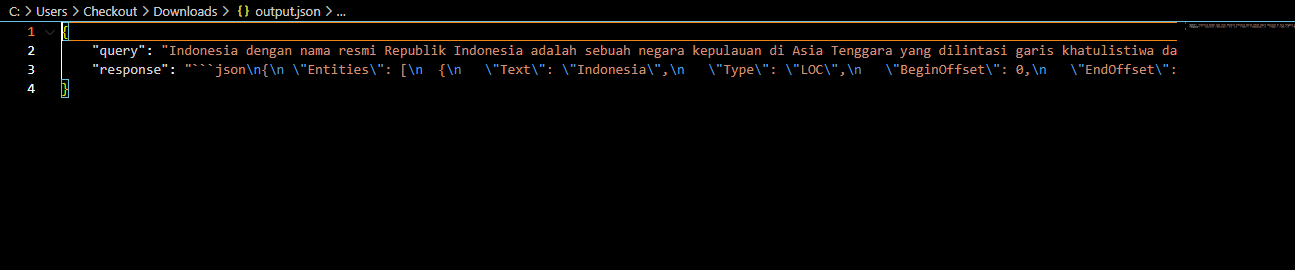
**Interacting with one of my old databases and providing solutions.**

****

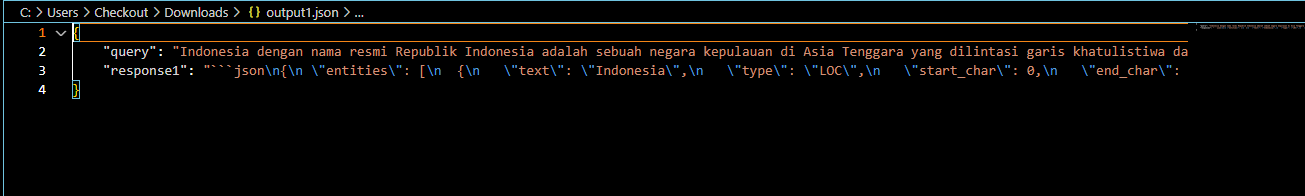
****

Storing result in JSON:

****

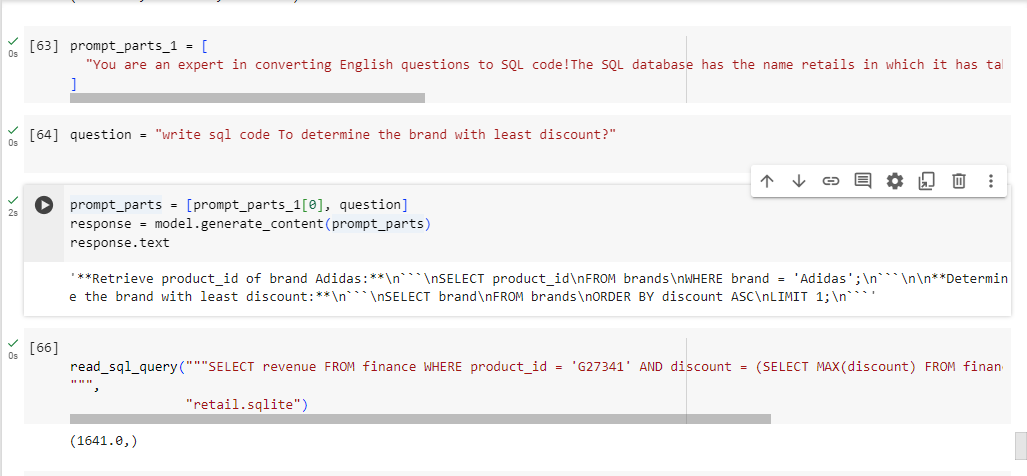
****

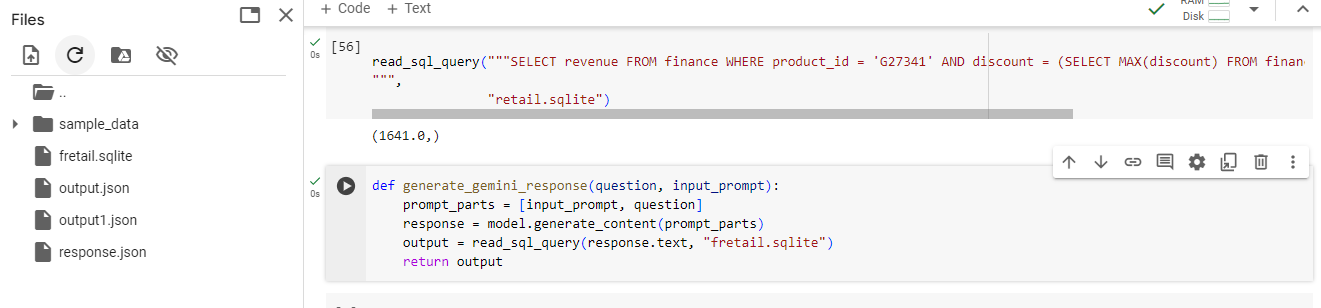
****

****

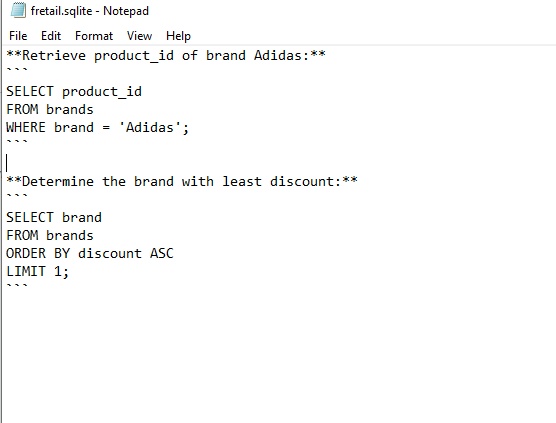
**Storing the preprocessed code in a database ( SQLite )**

Asking the LLM to go through the retail.sqlite from a previous work and answering the questions.



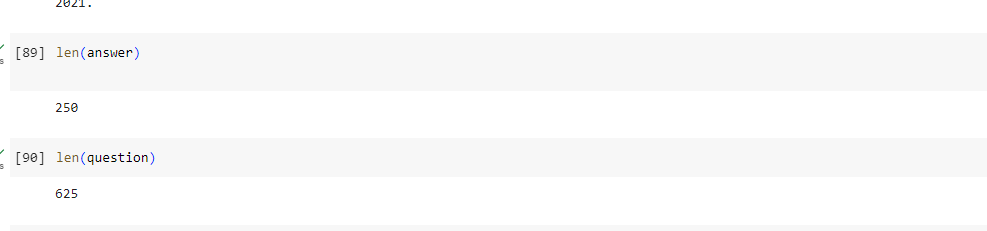
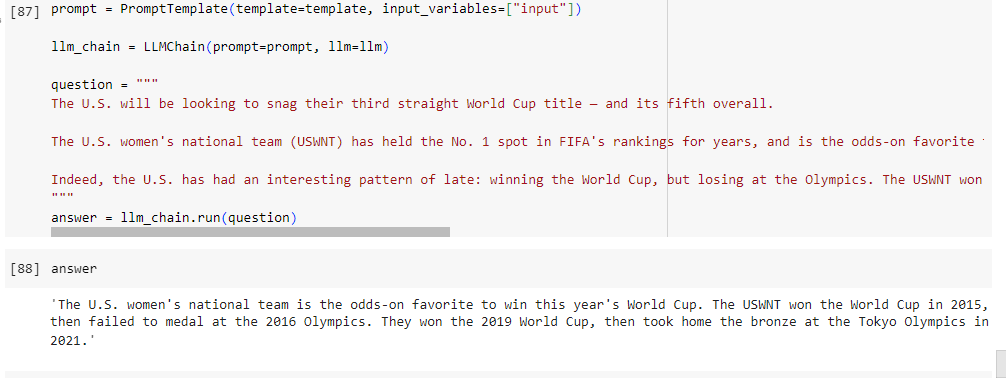


Storing the solution into sqlite file.

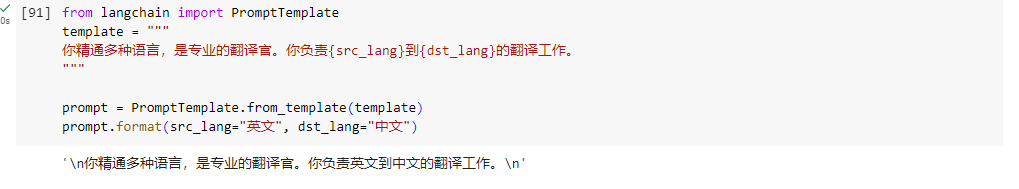


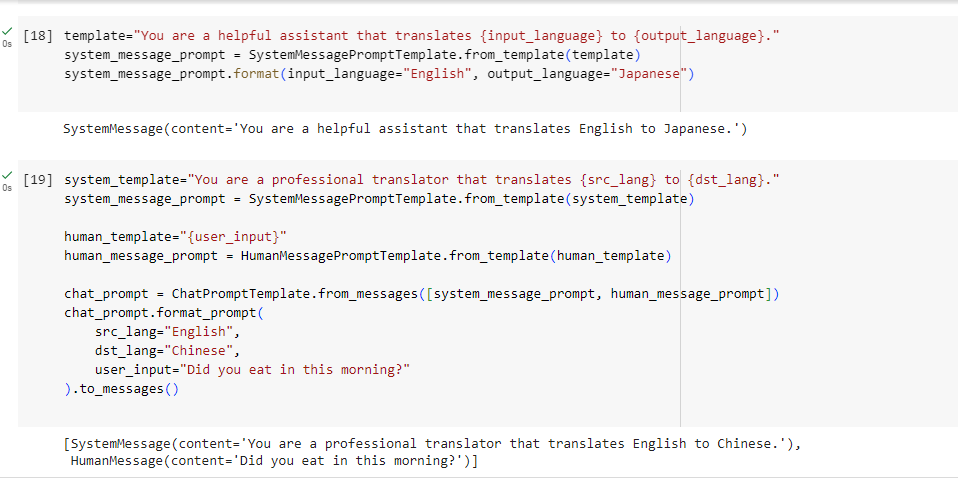
**LLM in summarization:**

Here we can see the length of the question and the summarizes answer.

****

**To see if our llm can offer the same performance when considered to interact with new languages.**

****

****

****

**2.4 [5 points]** **Reflect and Share**

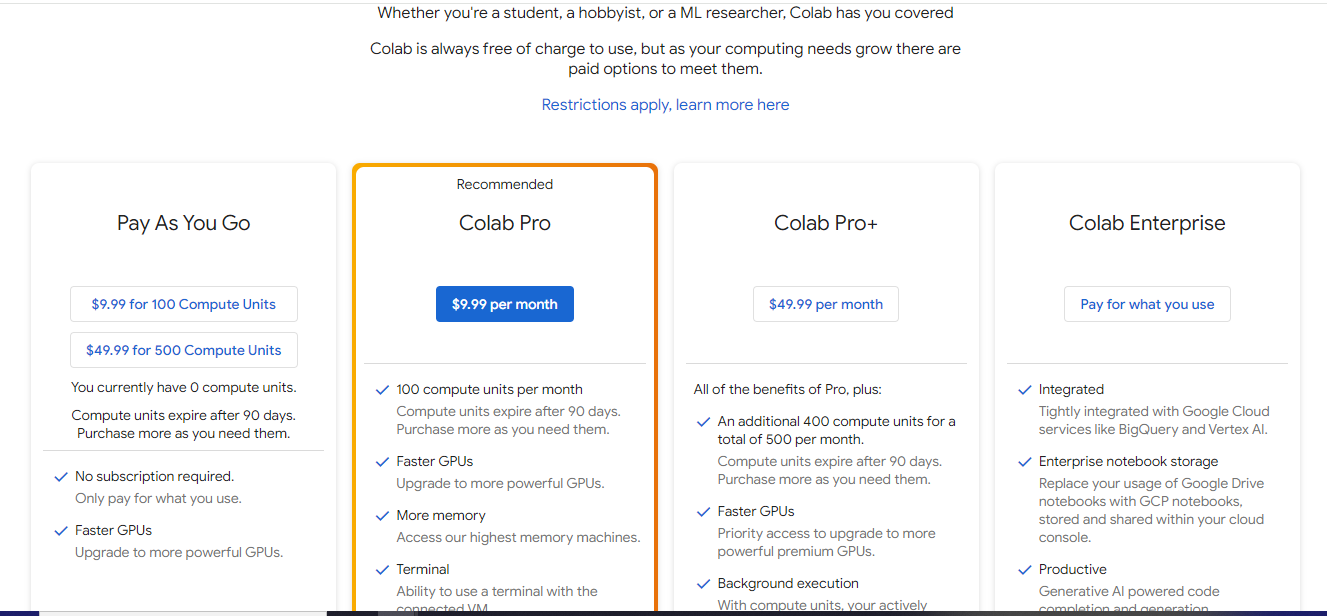
Summarize your learning experience, challenges faced, and insights gained. Document technical difficulties and possible best practices you discovered in the process.

* Challenges:

Gemini is not supported for university accounts, which was a main hindrance.

The free version had very few features to explore.

Working on all the subtasks took a lot of storage and time to install the dependencies.



* Learning Experiences:

API Integration: Integrating APIs such as GitHub API for fetching repository statistics and Google's AI Gemini for natural language generation provides practical experience in leveraging external services to enhance application functionality.

Data Processing: Understanding how to parse JSON data retrieved from web APIs (in this case, GitHub API) and extract relevant information demonstrates basic data processing skills essential for working with external data sources.

Configuration Management: Exploring the role of the SparkConf class in Apache Spark highlights the importance of configuration management in distributed computing environments. Learning to configure Spark applications dynamically can optimize performance and resource utilization.

* Insights

Great at understanding other languages.

Natural Language Understanding: Gemini Pro can help users gain insights into natural language understanding by analyzing various text inputs and generating relevant responses or summaries. This can be particularly useful for tasks such as text summarization, sentiment analysis, or question answering.

Content Generation: Users can gain insights into content generation by exploring the capabilities of Gemini Pro in generating human-like text based on prompts or input data. This can be valuable for tasks such as content creation, creative writing, or generating personalized responses.

Language Modeling: Gemini Pro can provide insights into language modeling techniques and approaches by showcasing how language models can generate coherent and contextually relevant text based on input prompts. Users can learn about language model architectures, training data, and fine-tuning techniques.

Creative Exploration: Gemini Pro can inspire creative exploration by generating diverse and imaginative text outputs based on user prompts. Users can experiment with different input scenarios, explore various writing styles, and generate novel ideas or narratives.

Link:

https://medium.com/@sindhukotegar/exploration-with-gemini-pro-14b94261892f

**References:**

Hugging Face Transformers (<https://huggingface.co/docs/transformers/en/index>

[Links to an external site.](https://huggingface.co/docs/transformers/en/index)

LangChain: <https://github.com/langchain-ai/langchain>

[Links to an external site.](https://github.com/langchain-ai/langchain)

OpenAI API (<https://openai.com/>

[Links to an external site.](https://openai.com/)

)

Google AI Platform (<https://cloud.google.com/vertex-ai>

[Links to an external site.](https://cloud.google.com/vertex-ai)

)