**INFOSYS SPRINGBOARD**

**TASK SUBMISSION**

**(ASSIGNMENT-1)**

**Submitted by:**

Shriya Vuppu

Batch 4

Under the Esteemed Guidance of

Mr . Bhargavesh Dakka

**TASK1: INTEGRATION OF LLM’S**

To develop an intelligent and responsive conversational system, I integrated a Large Language Model (LLM) with a Python-based application that processes user inputs and generates relevant responses. For this project, I selected the Groq API as the primary platform for integrating the LLM due to its high-performance capabilities and ease of use.

The Groq API is a powerful tool for deploying and interacting with machine learning models, offering a straightforward interface for integrating sophisticated AI models into applications. This enabled me to build a robust and scalable system that handles natural language processing tasks effectively.

Why I Chose Groq API

The decision to use the Groq API was motivated by the following factors:

1. Optimized for High Performance: Groq’s specialized hardware and software design allow for faster computation, which is critical when integrating AI models like LLMs. This results in improved response time and seamless user interactions, even when processing complex queries.
2. Access to Advanced Language Models: The Groq API offers access to powerful models such as Llama3, which are designed to handle a wide range of conversational inputs and generate contextually accurate responses. This made Groq an ideal choice for creating an interactive sales assistant.
3. Simple and Flexible Integration: The Groq API’s simple RESTful interface and comprehensive documentation made it easy to integrate into my Python project. This reduced the time required for setup and allowed me to focus on implementing the core functionalities of the conversational assistant.

OBJECTIVE:

The objective of this task was to integrate the Groq API with a Python script to communicate with a Large Language Model (LLM) for handling user input and generating responses. By completing this task, I aimed to connect with the Groq API and implement a conversational flow, ensuring the model generates responses based on user queries.

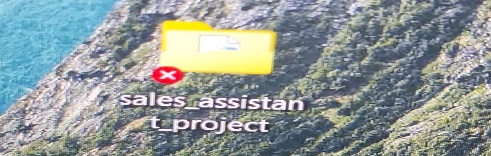
OUTCOMES:

1. Successful integration of the Groq API with a Python application.
2. Ability to send requests to the Groq API and receive LLM-generated responses.
3. A structured and reusable project setup with a virtual environment.

STEPS INVOLVED:

1. Project Setup

I created a project folder named **sales\_assistant\_project** and set up a virtual environment within it to manage dependencies efficiently.



2. Virtual Environment Configuration

To isolate project dependencies, I created a virtual environment using the following command:

**python -m venv sales\_assistant/activate/env**

Then, I activated the environment:

**sales\_assistant/activate/env/Scripts/activate**

3. Dependencies Installation

I installed the necessary libraries using pip:

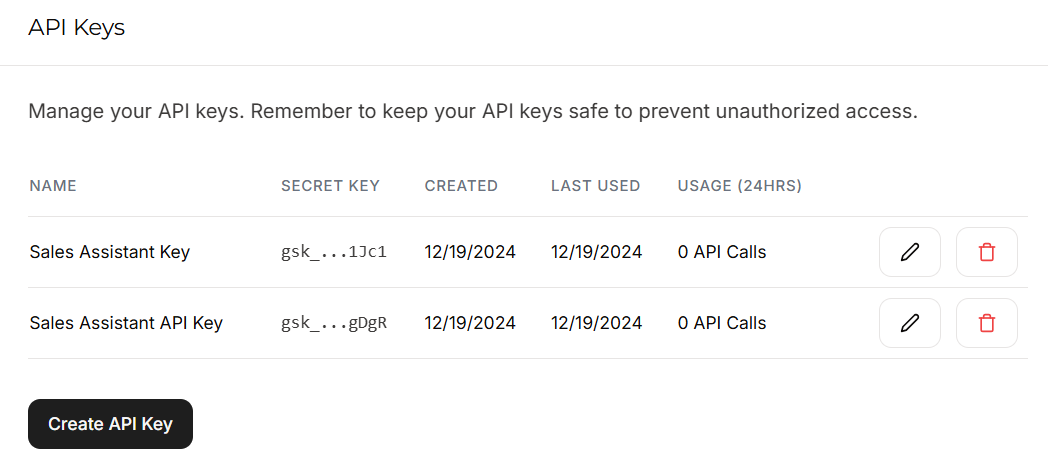
**pip install requests python-dot env**

These modules were installed to facilitate HTTP requests and manage environment variables securely.

4. API Key Configuration

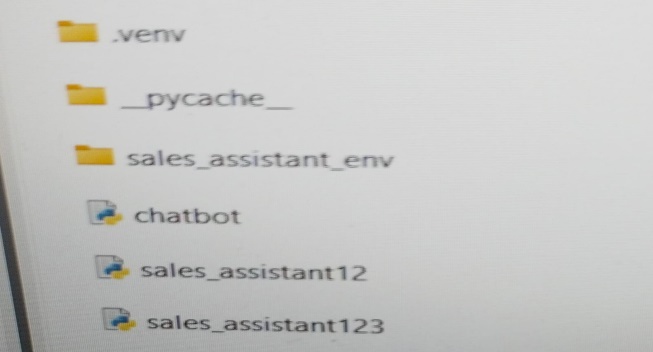
I set the Groq API key in a .env file for security. The .env file contains:

GROQ\_API\_KEY=gsk\_Sz………………………….1cv



5. Integration Code

I wrote a Python script named sales\_assistant123.py to interact with the Groq API and return the model's response based on a user query.



**Code Snippet:**

import requests

import json

import os

from dotenv import load\_dotenv

load\_dotenv()

api\_key = os.getenv("GROQ\_API\_KEY")

headers = {

"Authorization": f"Bearer {api\_key}",

"Content-Type": "application/json"

}

url = "https://api.groq.com/openai/v1/chat/completions"

payload = {

"model": "llama3-8b-8192",

"messages": [

{"role": "user", "content": "Hello, how can I assist with sales?"}

]

}

response = requests.post(url, headers=headers, json=payload)

if response.status\_code == 200:

print("Response:", response.json()["choices"][0]["message"]["content"])

else:

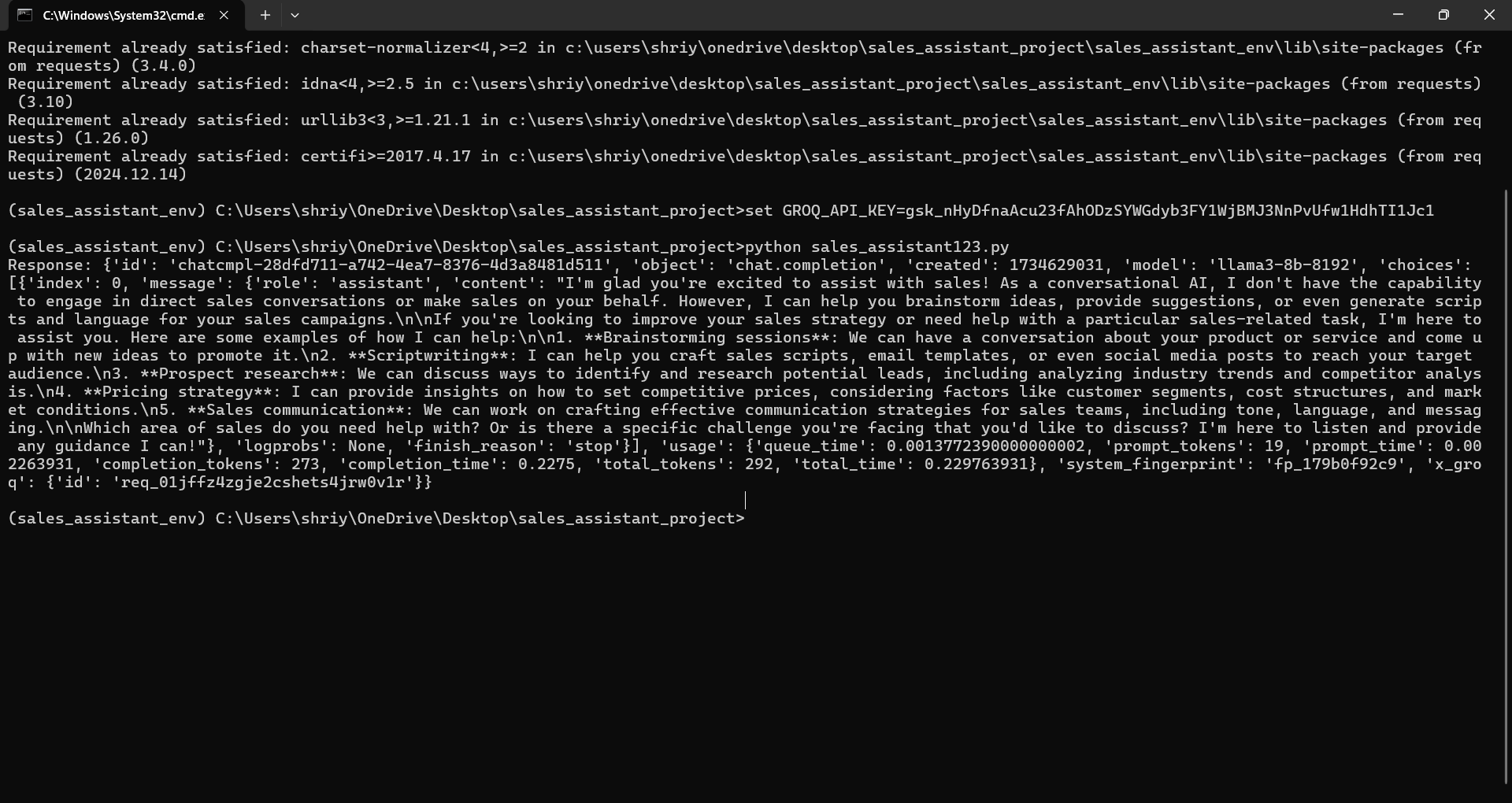
print("Error:", response.status\_code, response.text)

6. Execution

To run the script, I used the following command:

**python sales\_assistant123.py**

OUTPUT:



RESPONSE :

"id": "chatcmpl-abc123",

"object": "chat.completion",

"created": 1712345678,

"model": "llama3-8b-8192",

"choices": [

{

"index": 0,

"message": {

"role": "assistant",

"content": "Hello! How can I assist you with your sales inquiries today?" {

},

"finish\_reason": "stop"

},

{

"index": 1,

"message": {

"role": "assistant",

"content": "Hi there! What sales-related support do you need?"

},

"finish\_reason": "stop"

},

{

"index": 2,

"message": {

"role": "assistant",

"content": "Good day! How may I help you with your sales queries?"

},

"finish\_reason": "stop"

}

],

"usage": {

"prompt\_tokens": 12,

"completion\_tokens": 27,

"total\_tokens": 39

}

}