# Report: Learning About Gemini API, Agents

## 1. Gemini API

### Overview

* **What I Learned**: I explored using the free Gemini API from Google, focused on generating content. The API allows developers to build applications leveraging language models for text generation, summaries, and other AI-powered tasks.
* **Experience**: I successfully ran the initial code in VS Code, which helped solidify my understanding of interacting with the Gemini API. This involved making API calls, processing responses, and handling potential errors.

### Code Example

    # Import necessary modules

import os

import getpass

from google.auth.exceptions import DefaultCredentialsError

from langchain\_google\_genai import ChatGoogleGenerativeAI

from langchain\_core.prompts import ChatPromptTemplate

from langchain\_core.output\_parsers import StrOutputParser

from langchain\_core.runnables import RunnablePassthrough

# Set the API key directly in the script

os.environ["GOOGLE\_API\_KEY"] = "YOUR\_API\_KEY"

# Prompt for Google API key if not set

if "GOOGLE\_API\_KEY" not in os.environ:

    os.environ["GOOGLE\_API\_KEY"] = getpass.getpass("Enter your Google AI API key: ")

try:

    # Initialize the ChatGoogleGenerativeAI model

    llm = ChatGoogleGenerativeAI(

        model="gemini-1.5-flash",  # Specify the Gemini model to use

        temperature=0.7,           # Control randomness (0.0 = deterministic, 1.0 = very random)

        max\_output\_tokens=100      # Limit the response length

    )

    # Create a chat prompt template

    prompt = ChatPromptTemplate.from\_messages([

        ("system", "You are a Python developer."),

        ("human", "{question}")

    ])

    # Create a chain combining the prompt and the language model

    chain = (

        {"question": RunnablePassthrough()}

        | prompt

        | llm

        | StrOutputParser()

    )

    # Get user input for the question

    user\_question = input("Enter your question: ")

    # Use the chain to generate a response

    response = chain.invoke(user\_question)

    # Print the generated content

    print("\nResponse:")

    print(response)

except DefaultCredentialsError as e:

    print("Error: Google API credentials not found or invalid.")

    print("Please make sure you've set up your Google API key correctly.")

    print("Error details:", str(e))

except Exception as e:

    print("An unexpected error occurred:")

    print(str(e))

## 2. Concepts About Agents

### What Are Agents?

* Agents use language models (LLMs) to take a series of actions based on user input. Unlike basic text responses, agents can execute commands, solve complex tasks, and interact with external systems.
* **Key Characteristics**:
  + They provide more than knowledge, enabling automation by performing actions.
  + Used in applications such as chatbots, recommendation systems, and automation tools.
  + Example Frameworks: Langraph, Crew AI, Microsoft AutoGen.

### How Do LLM Agents Work?

A typical LLM agent functions as follows:

1. **Input**: The user provides a prompt or question.
2. **Interpretation**: The agent processes the text and understands the task.
3. **Decision**: The agent decides which actions need to be taken.
4. **Action**: The agent performs the required task (e.g., fetching data, running a function).
5. **Output**: The agent returns the result in a human-readable format (text, table, graph, API response, etc.).

### Examples of LLM Agents

* **Personal Assistants**: Siri, Alexa
* **Customer Support Bots**
* **Business Assistants**

## 3. Exploring Tools for Agents and LLMs

### Using Tools with LLM Agents

LLM agents interact with tools to enhance their capabilities and perform complex tasks. These tools can include APIs, databases, and external systems.

### How LLM Agents Use Tools

1. **Interpret User Input**: The agent analyzes the user’s request.
2. **Select the Right Tool**: Based on the task, the agent chooses an appropriate tool (e.g., API, data processing).
3. **Execute the Task**: The agent provides input to the tool and waits for the result.
4. **Generate Output**: The agent processes the output and delivers it in a user-friendly format.

### Examples

* **Real-Time Information Retrieval**: A user asks, "What’s Tesla's stock price right now?" The agent fetches the stock price using an API.
* **Data Analysis**: A user provides a sales data CSV file and asks for the top 5 products. The agent processes the data and returns the top-selling items.

4. Fetching Yelp API Data

**Overview**  
• What I Learned: I explored the Yelp API to fetch business data based on location. The API provides a rich set of information on businesses, including name, rating, category, and location details.

**Code Example**

import requests

import pandas as pd

from tabulate import tabulate

import json

key = "ryDZoNBlIR7IhRrPnq6H79AtSRS4ZMpamzglMQQCFfjg3e-0DFKFG5HDLKkSu2t5czEq9D4KVvEw0-17mORgxfg68R8Vlgxjhfz3AibhvDPERAWXrBWgXfrG55YHZ3Yx"

url = "https://api.yelp.com/v3/businesses/search?sort\_by=best\_match&limit=20"

headers = {'Authorization': f'Bearer {key}', "accept": "application/json"}

response = requests.get(url, headers=headers, params={"location": '1067 Inez Drive, Smyrna, Tennessee 37167'})

if response.status\_code == 200:

    data = response.json()

    businesses = data['businesses']

    # Flatten nested dictionaries

    def flatten\_dict(d, parent\_key='', sep='\_'):

        items = []

        for k, v in d.items():

            new\_key = f"{parent\_key}{sep}{k}" if parent\_key else k

            if isinstance(v, dict):

                items.extend(flatten\_dict(v, new\_key, sep=sep).items())

            elif isinstance(v, list):

                items.append((new\_key, json.dumps(v)))

            else:

                items.append((new\_key, v))

        return dict(items)

    # Extract all information

    flattened\_businesses = [flatten\_dict(business) for business in businesses]

    # Create DataFrame

    df = pd.DataFrame(flattened\_businesses)

    # Format table

    table = tabulate(df, headers='keys', tablefmt='pretty', showindex=False)

    print("Output:-")

    print(table)

    # Optionally, save to Excel

    df.to\_excel('yelp\_businesses.xlsx', index=False)

    print("Data saved to yelp\_businesses.xlsx")

else:

    print(f"Error: {response.status\_code}")

    print(response.text)

## Summary

Today's learning focused on practical usage of the Gemini API and deepened my understanding of LLM agents and their role in automation. This knowledge will help me design intelligent systems that autonomously interact with data, APIs, and external systems.

I successfully ran code to retrieve and display business data from the Yelp API using Pandas and Tabulate. The results were structured and saved in Excel for further analysis, reinforcing my ability to process and analyze API-driven data in real-world scenarios.