PharmaSUG 2025 - Paper AI-299

Leveraging Gen AI (ChatGPT, Gemini) API for Advanced Biometric Analysis in SAS, Python and R

Kevin Lee, ClinVia, San Francisco, CA

ABSTRACT

Since its release, ChatGPT has rapidly gained popularity, reaching 100 million users within 2 months. Not only we could use ChatGPT for prompting, but we could build application / codes using API.

The paper explores the integration of Gen AI (e.g., OpenAI, Gemini) API into biometric data analysis workflows using SAS, Python, and R programming languages. The paper will display the workflow of Gen AI integration and provide sample codes for SAS, Python, and R programming. First, the paper will show how to create API in OpenAI (paid) and Gemini (free version). Secondly, it will help to create the input data containing prompts and the questions to Gen AI. Thirdly, it will create codes (e.g., SAS, Python, R) to read the input data and to generate output data. Fourth, it will show how to read output data to obtain the results from ChatGPT.

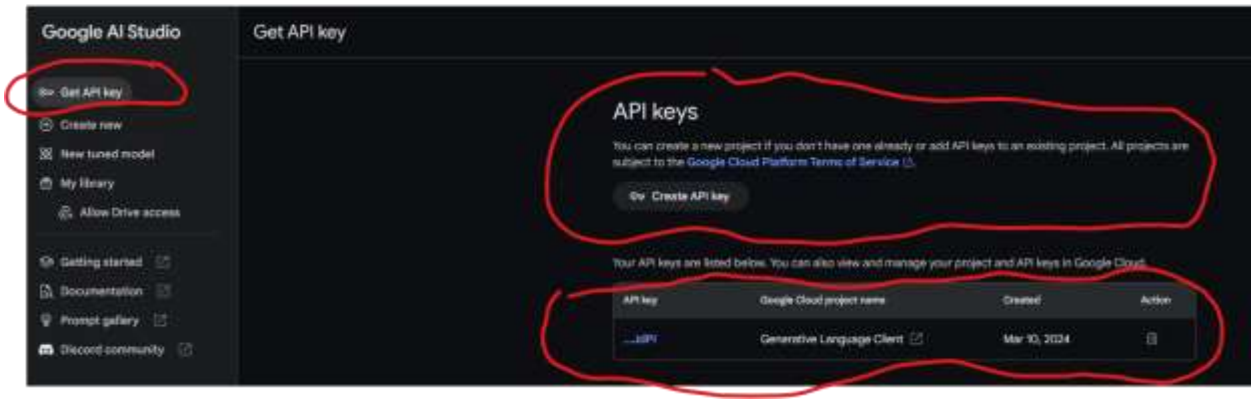
The paper also presents a document-based question-answering system leveraging LangChain, Retrieval-Augmented Generation (RAG), and OpenAI models. It will show how we could build a simple chatbot for ADaM IG.

Finally, the paper will explore the potential use cases of Gen-AI application to revolutionize biometric function, outlining innovative solutions and strategies for leading this technological advancement such as code conversion, code generation, prompt-driven data visualization, content development (e.g., SAP, CSR, Narratives), AI Agent driven automation and many more. Furthermore, it will address the inherent risks associated with Generative AI-generated output and propose mitigation strategies.

How to get API Key from Google AI Studio

The programmer can get a free version of the Gemini API through Google AI Studio. Here's how:

1. Go to Google AI Studio: Visit the Google AI Studio website in in https://aistudio.google.com/app/apikey
2. Get API Key: Look for the "Get API Key" button, usually in the top left corner. Click it.
3. Create API Key: You'll be prompted to create an API key. You can choose to create it in a new project or an existing one.
4. Copy Your Key: Once the key is generated, copy it and store it securely. You'll need this key to access the Gemini API.



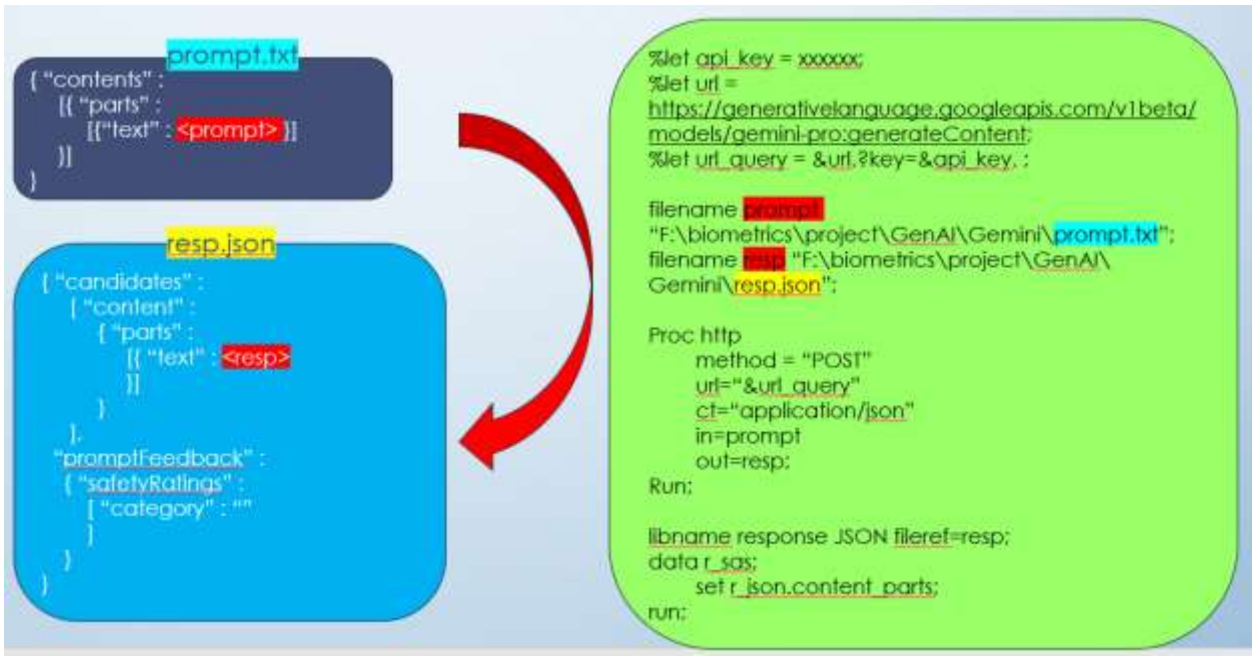
You can create a new project if you don't have one already or add API keys to an existing project. All projects are subject to the Google Cloud Platform Terms of Service.

Your API keys are listed below. You can also view and manage your project and API keys in Google Cloud.

API key	Google Cloud project name	Created	Action
...jdPI	Generative Language Client	Mar 10, 2024	

Workflow of Gemini API integration in the SAS® codes

Below diagram will show how SAS codes will use Gemini API for Gen-AI driven analysis.



```
{ "contents" :
    [{ "parts" :
        [{"text" : <prompt> }]
    }]
}
```

```
{ "candidates" :
    [ "content" :
        { "parts" :
            [{ "text" : <resp>
            }]
        }
    ],
  "promptFeedback" :
    { "safetyRatings" :
        [ "category" : ""
        ]
    }
}
```

```
%let api_key = xxxxxx;
%let url =
https://generativelanguage.googleapis.com/v1beta/
models/gemini-pro:generateContent;
%let url_query = &url.?key=&api_key. ;

filename prompt
"F:\biometrics\project\GenAI\Gemini\prompt.txt";
filename resp "F:\biometrics\project\GenAI\
Gemini\resp.json";

Proc http
      method = "POST"
      url="&url_query"
      ct="application/json"
      in=prompt
      out=resp;
Run;

libname response JSON fileref=resp;
data r_sas;
      set r_json.content_parts;
run;
```

SAS codes using Gemini API
Below SAS codes will read the prompt from the input data and generate the response to the output data.

```
%let api_key = xxxxxx;
%let url = https://generativelanguage.googleapis.com/v1/models/gemini-p1.5-flash:generateContent;
%let url_query = &url.?key=&api_key. ;

filename prompt "F:\biometrics\project\GenAI\Gemini\prompt.txt";
filename resp "F:\biometrics\project\GenAI\ Gemini\resp.json";

proc http
        method = "POST"
        url="&url_query"
        ct="application/json"
        in=prompt
        out=resp;
run;

libname response JSON fileref=resp;

data r_sas;
        set r_json.content_parts;
run;
```

Its input data will be "prompt.txt", and above SAS codes will read yellow-highlighted <prompt>.

```
{ "contents" :
        [{ "parts" :
                [{"text" : <prompt> }]
        }]
}
```

Finally, output data will be "resp.json" and above SAS codes will read yellow-highlighted <response>.

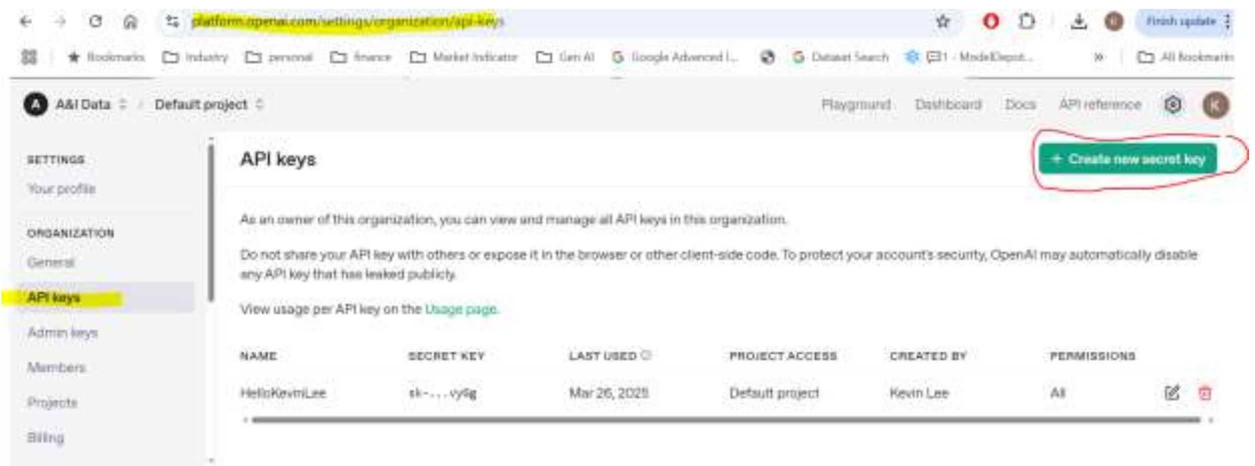
```
{ "candidates" :
        [ "content" :
                { "parts" :
                        [{ "text" : <response>
```

```
                                        }]
                            }
                    ],
  “promptFeedback” :
    { “safetyRatings” :
                    [ “category” : “”
                    ]
    }
}
```

How to get API Key from OpenAI API

The programmer can get paid version of the OpenAI API through Google AI Studio. Here's how:

1. Go to OpenAI API in https://platform.openai.com/settings/organization/api-keys
2. Get API Key: Look for the "Create new secret key" button, usually in the top right corner. Click it.
3. Copy Your Key: Once the key is generated, copy it and store it securely. You'll need this key to access the OpenAI API.



API keys

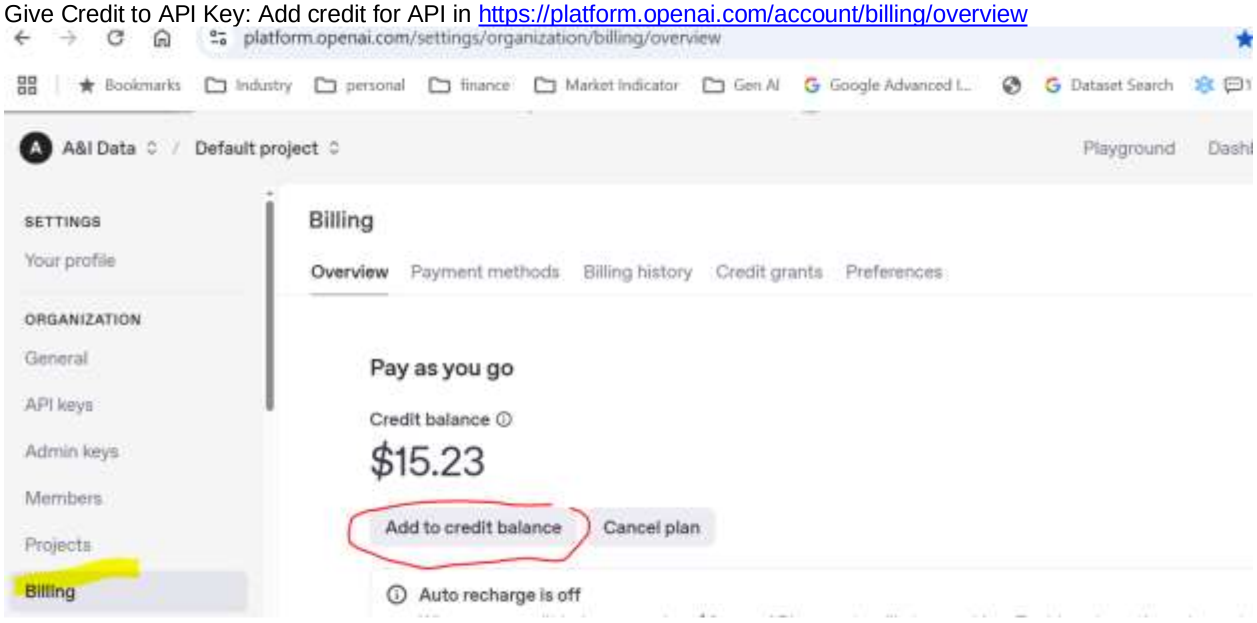
As an owner of this organization, you can view and manage all API keys in this organization.

Do not share your API key with others or expose it in the browser or other client-side code. To protect your account's security, OpenAI may automatically disable any API key that has leaked publicly.

View usage per API key on the Usage page.

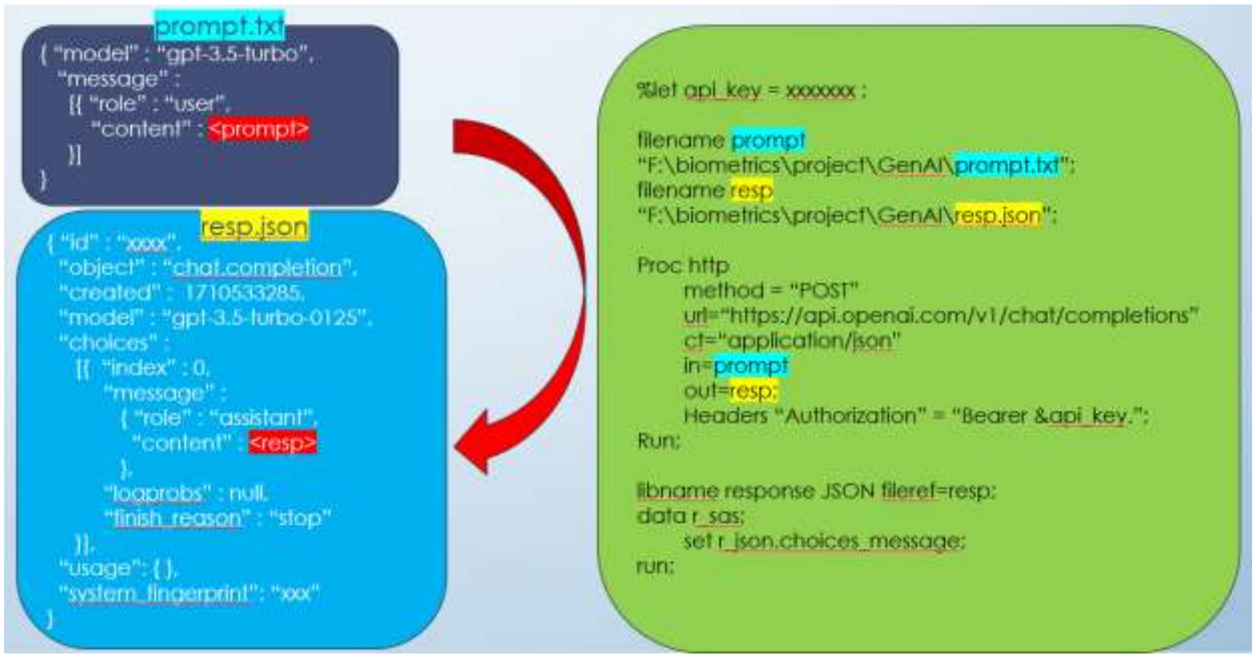
NAME	SECRET KEY	LAST USED	PROJECT ACCESS	CREATED BY	PERMISSIONS
HelloKevinLee	sk-....vy6g	Mar 26, 2025	Default project	Kevin Lee	All

4. Give Credit to API Key: Add credit for API in https://platform.openai.com/account/billing/overview



Workflow of OpenAI API integration in the SAS codes

Below diagram will show how SAS codes will use OpenAI API for Gen-AI driven analysis.



```
{ "model" : "gpt-3.5-turbo",
  "message" :
   [{ "role" : "user",
      "content" : <prompt>
   }]
}
```

```
{ "id" : "xxxx",
  "object" : "chat.completion",
  "created" :  1710533285,
  "model" : "gpt-3.5-turbo-0125",
  "choices" :
   [{ "index" : 0,
      "message" :
        { "role" : "assistant",
          "content" : <resp>
        },
      "logprobs" : null,
      "finish_reason" : "stop"
   }],
  "usage": { },
  "system_fingerprint": "xxx"
}
```

```
%let api_key = xxxxxxx ;

filename prompt
"F:\biometrics\project\GenAI\prompt.txt";
filename resp
"F:\biometrics\project\GenAI\resp.json";

Proc http
      method = "POST"
      url="https://api.openai.com/v1/chat/completions"
      ct="application/json"
      in=prompt
      out=resp;
      Headers "Authorization" = "Bearer &api_key.";
Run;

libname response JSON fileref=resp;
data r_sas;
      set r_json.choices_message;
run;
```

SAS codes using OpenAI API
Below SAS codes will read the prompt from the input data and generate the response to the output data.

```
%let api_key = xxxxxxx ;
filename prompt "F:\biometrics\project\GenAI\OpenAI\prompt.txt";
filename resp "F:\biometrics\project\GenAI\OpenAI\resp.json";

proc http
            method = "POST"
            url="https://api.openai.com/v1/chat/completions"
            ct="application/json"
            in=prompt
            out=resp;
            Headers "Authorization" = "Bearer &api_key.";
run;

libname response JSON fileref=resp;
data r_sas;
            set r_json.choices_message;
run;
```

Its input data will be "prompt.txt", and above SAS codes will read yellow-highlighted <prompt>.

```
{ "model" : "gpt-3.5-turbo",
   "message" :
            [{ "role" : "user",
               "content" : <prompt>
            }]
}
```

Finally, output data will be "resp.json" and above SAS codes will read yellow-highlighted <resp>.

```
{ "id" : "xxxx",
  "object" : "chat.completion",
  "created" :  1710533285,
  "model" : "gpt-3.5-turbo-0125",
  "choices" :
```

```
            [{  “index” : 0,
                        “message” :
                          { “role” : “assistant”,
                                    “content” : <resp>
                          },
                        “logprobs” : null,
                        “finish_reason” : “stop”
            }],
  “usage”: { },
  “system_fingerprint”: “xxx”
}
```

R codes using OpenAI API

Below R codes will read the prompt and generate the response.

```
## import OpenAI ChatGPT library
library(httr)
library(jsonlite)
library(dotenv)

# Load environment variables
curr_path = "C:/Users/kevin/OneDrive - Cornell University/training/Machine
Learning/Training/PharmaSUG/2025/.env"
dotenv::load_dot_env(file=curr_path)

# Set your OpenAI API key from .env file
# api_key <- "xxxx"
api_key <- Sys.getenv("OPENAI_API_KEY_R")

# Define the request URL
api_url <- "https://api.openai.com/v1/chat/completions"

prompt <- "Tell me about CDISC in 200 words"

# Define the request body
body <- list(
  model = "gpt-3.5-turbo",
  messages = list(
    list(role = "system", content = "You are a helpful assistant."),
    list(role = "user", content = prompt )
  )
)

# Make the API request
response <- POST(
  api_url,
  add_headers('Authorization' = paste("Bearer", api_key),
              'Content-Type' = "application/json"),
  body = jsonlite::toJSON(body, auto_unbox = TRUE),
  encode = "json"
)

# Parse the response
result <- content(response, as = "parsed")

# Print the response
print(result$choices[[1]]$message$content)
```

```
[1] "CDISC, or the Clinical Data Interchange Standards Consortium, is a global non-profit organization that develops
and promotes data standards to streamline research processes in the life sciences industry, particularly in clinical
trials. Established in 1997, CDISC aims to enhance data quality, efficiency, and interoperability in clinical
research.\n\nCDISC standards provide a common language and format for organizing and reporting data, making it
easier for different studies to be compared and integrated. They cover various aspects of clinical research, including
data collection, management, and analysis. Some of the key CDISC standards include CDASH (Clinical Data
Acquisition Standards Harmonization) for data collection, SDTM (Study Data Tabulation Model) for data submission,
and ADaM (Analysis Data Model) for data analysis and reporting.\n\nAdoption of CDISC standards by regulatory
```

authorities like the FDA and the EMA has become increasingly common, as they facilitate the submission and review of clinical trial data. Compliance with CDISC standards not only ensures data quality and consistency across different studies but also supports the efficient exchange of data between organizations, ultimately benefiting patients by accelerating the development of new therapies."

Python codes using OpenAI API
Below Python codes will read the prompt and generate the response.

```
## import OpenAI ChatGPT library
import openai

## Create Prompt message
message = [{"role" : "user", "content " : " Tell me about CDISC in 200 words" }]

## Set up OpenAI ChatGPT API
openai.api_key = XXXXXXXX

## Send Prompt message to OpenAI
## Get the response
response = openai.chat.completion.create(
        model="gpt-3.5-turbo",
        messages=message,
        temperature=0
)

##print response
Print(response.choices[0].message.content)
```

CDISC, or Clinical Data Interchange Standards Consortium, is a global non-profit organization that develops and promotes data standards for clinical research. These standards help streamline the collection, analysis, and sharing of data in clinical trials, ultimately improving efficiency and data quality. CDISC standards cover various aspects of clinical research, including data collection, data submission, and data analysis. By adopting CDISC standards, organizations can ensure that their data is consistent, interoperable, and compliant with regulatory requirements, leading to more efficient and effective clinical research processes.

Python codes using OpenAI API, and LangChain and RAG
Below Python codes will build a simple Q&A application for ADaM IG using LangChain, RAG and OpenAI API.

1. LangChain's Role - LangChain is a tool that helps organize and process documents. In the code:
   - Document Loading: LangChain's `PyPDFLoader` reads the PDF and splits it into pages.
   - Text Splitting: LangChain's `CharacterTextSplitter` breaks the pages into smaller chunks (like paragraphs) so the computer can handle them easily.
   - ChromaDB: LangChain connects to Chroma, a database that stores text chunks as "vectors" (numbers representing text meaning) for fast searching.
   - QA Chain: LangChain links the document search (ChromaDB) and the AI (OpenAI) to answer questions.

2. RAG (Retrieval-Augmented Generation) : RAG is a two-step process:
   - Retrieval: When you ask a question (e.g., What variables are required in ADSL?), ChromaDB searches the stored text chunks to find the most relevant parts of the PDF.
   - Generation: OpenAI's model reads those chunks and writes a clear answer using the retrieved information.

   This ensures answers are based on the document (not just the AI's general knowledge).

3. OpenAI's Role
   - Embeddings: OpenAI turns text chunks into numbers (vectors) so the computer can understand their meaning and find similar text.
   - Answer Generation: OpenAI API `gpt-3.5-turbo` reads the retrieved text chunks and writes a human-like answer.

Its Workflow is
1. Load PDF → Split into pages.

2. Chunk text → Break pages into smaller pieces.
3. Convert to vectors (OpenAI) → Store in ChromaDB.
4. Retrieve relevant chunks when a question is asked.
5. Generate answer (OpenAI) using those chunks.

The code automates answering questions from large documents by combining document search (LangChain/Chroma) and AI smarts (OpenAI).

```
## import PDF loader package
from langchain.document_loaders import PyPDFLoader

## loading ADaM IG pdf file
loader = PyPDFLoader("docs/ADaMIG v1.2-Final.pdf")
pages = loader.load() ## ADaM IG has 110 pages.

## Import Document Splitting Package
from langchain.text_splitter import CharacterTextSplitter

## Set tup Text Splitter
text_splitter = CharacterTextSplitter(
   separator="\n",
   chunk_size=1000,
   chunk_overlap=150,
   length_function=len
)

## Split Pages
docs_pdf = text_splitter.split_documents(pages)

## import OpenAI embedding package
from langchain.embeddings.openai import OpenAIEmbeddings

## Create OpenAI Embeddings instance
embedding = OpenAIEmbeddings( openai_api_key=os.getenv("chatgpt_api"))

## Import langchain chroma DB package
from langchain.vectorstores import Chroma

persist_directory = 'docs/chroma/'

## Store Vector Data in the local drive
vectordb = Chroma.from_documents(
   documents=docs_pdf,
   embedding=embedding,
   persist_directory=persist_directory
)

## Import OpenAI LLM model
from langchain.chat_models import ChatOpenAI

## Initialize OpenAI LLM
llm = ChatOpenAI(
   model_name="gpt-3.5-turbo",
   openai_api_key=os.getenv("chatgpt_api"),
   temperature=0)


## import Retrieval QA Chain package
from langchain.chains import RetrievalQA

## Initiate QA chain using openAI LLM and Vector Store DB
qa_chain = RetrievalQA.from_chain_type(
   llm,
   return_source_documents=True,
   retriever=vectordb.as_retriever() # defualt search_type=mmr, k=4
)

## Feed question into QA chain and get the result
question = "What variables are required in ADSL?"

result = qa_chain({"query": question})

result["result"]

'The required variables in ADSL are specified in Section 3.2, ADSL Variables.'
```

Gen-AI driven Biometric innovation

There are many applications that we could use Gen AI API.

- SAS code conversion to R or Python
- Metadata driven code development (e.g., SAS, R or Python)
- Prompt driven data visualization
- Contents development (e.g., SAP, Mock-up tables, CSR, Safety narratives)
- AI Agent driven workflow & automation
- Many more

Gen AI powered Programming - Pros

1. Rapid Prototyping and Development
   - Gen AI APIs enable programmers to quickly build sophisticated applications by leveraging pre-trained Gen AI models.
   - Significantly reduces development time compared to building AI capabilities from scratch

2. Advanced Functionality with Minimal Expertise
   - Provides access to cutting-edge Gen AI capabilities like natural language processing, image generation, and complex reasoning
   - Democratizes access to sophisticated Gen AI technologies across department

3. Scalability and Flexibility
   - Cloud-based AI APIs can easily scale to handle varying levels of computational demand
   - Flexible integration across different programming languages and frameworks
   - Can be quickly adapted to different use cases and business requirements

4. Cost-Effectiveness
   - Reduces infrastructure and computational costs associated with training and maintaining Gen AI models
   - Pay-as-you-go pricing models allow for more predictable budgeting
   - Eliminates the need for massive upfront investments in AI infrastructure

5. Continuous Improvement
   - Gen AI providers continuously update and improve their models
   - Access to the latest Gen AI advancements with minimal effort

Gen AI powered Programming - Cons

1. Dependency on Gen AI company
   - Heavy reliance on third-party Gen AI API providers
   - Potential challenges in switching between different Gen AI service providers
   - Risk of service disruptions or significant pricing changes

2. Privacy and Data Security Concerns
   - Sending sensitive data through external APIs raises potential security risks
   - Compliance challenges with data protection regulations
   - Limited control over how data is processed and stored by API providers

3. Performance and Latency Issues
   - Network latency can impact real-time application responsiveness
   - Potential performance bottlenecks during high-traffic periods
   - Dependence on external service availability and response times

4. Limited Customization
    - Pre-trained Gen AI models might not perfectly align with specific use cases
    - Restricted ability to fine-tune models compared to custom-built solutions
    - Generic responses that may lack domain-specific nuance

5. Cost Unpredictability
    - Usage-based pricing can lead to unexpected expenses (e.g., Gen AI application not suitable for record-based programming)
    - Complex pricing structures might make budgeting challenging
    - Potential for high costs with increased API usage

6. Ethical and Bias Considerations
    - Inherit potential biases from pre-trained models
    - Limited transparency in model decision-making processes
    - Potential for generating inappropriate or controversial content

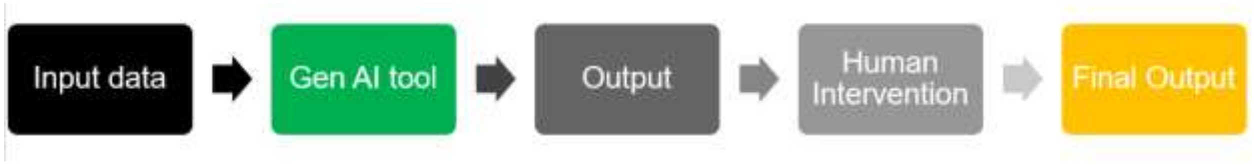
By understanding these pros and cons, we should make informed decisions about integrating Gen AI API into our programming based on company and Biometrics department strategy, balancing innovation with practical considerations and risk management.

Risk of Gen-AI output

The output of Gen-AI application should be considered a source of input, not a source of truth or final output. Below could be possible reasons.

- Gen-AI models lack clear traceability and explainability, operating as a "black box," hindering understanding why a model produced a given result.
- They are also susceptible to biases and hallucinations, generating factually incorrect or nonsensical outputs, and are highly sensitive to the quality and characteristics of input data.
- Furthermore, the evolving nature of these models means outputs can be inconsistent over time, and their performance is often highly context-dependent.

Therefore, while Gen-AI can be a powerful tool, its output should always be treated with a caution, requiring validation, ideally involving human review and comparison with established benchmarks, before being used as the final output.



CONCLUSION

In conclusion, the paper discusses the process of integrating Gen AI (e.g., Gemini and OpenAI) APIs into SAS, R and Python environments for Gen-AI driven analysis. The paper also explores how to obtain API keys, the workflow of API integration, and provides example code snippets for API integration. Furthermore, the paper showcases a practical application of OpenAI with LangChain and RAG for building a Q&A system for company document. Finally, the paper will discuss potential Gen-AI driven biometric innovations and highlight the crucial need for caution regarding Gen-AI outputs due to inherent risks like bias, hallucinations, and lack of explainability, emphasizing the importance of validation before final use.

REFERENCES

- OpenAI in https://platform.openai.com/settings/organization/billing/overview
- Google AI Studio in https://aistudio.google.com/app/apikey
- LangChain in https://www.langchain.com/
- RAG in LangChain in https://python.langchain.com/docs/tutorials/rag/

CONTACT INFORMATION

Your comments and questions are valued and welcomed. Please contact the author at

Kevin Lee
ClinVia LLC
kevin@clinvia.com

TRADEMARKS