End to End Document Q & A chatbot RAG_APP using Google open sources that is called as Gemma & Groq API

Definition of Gemma :

Gemma is a family of lightweight, state of the art open models builds from the same research and technology used to create the Gemini models.

Bigger companies like Meta and Google they are specially working on open sources models like Lama 2, Lama 3, why google will behind that so is why it open sources model and that is why Gemma and its variant models are there.

1.Here we will see how to use Gemma?

2.We are also using amazing inferencing engine which is called as “Groq”. Why Groq is used for specifically, if we really want to do faster inferencing we use Groq API. So, we are using this one also into our project.

3.If I will talk about the Gemma Model variants, there are 4 to 5 models :

(a) Gemma 2 – which was recently launched

(b) Gemma 1

(C) Recurrent Gemma - For to improve memory efficiency.

(d) Pali Gemma - This specifically open vision language model

(e) Code Gemma - Lets say if we really want to work with model which will be able to provide with us a amazing code assistance then we specifically used the Code Gemma.

These all models we are talking about will be developing this and as of now for these project we will be using the **[Gemma 1]** model for the practical purposes for this project.

- ➢ Here in Gemma 1 model, we will be seeing that gemma usually different kind of model with respect the parameters. There is 2 Billions there is 3 billions, might be there more billions parameters model that are available.
- ➢ Gemma is also available in Hugging face and also in Kaggle.

Like inside Kaggle we can use this model or else we can specifically locally execute and download the entire model locally.

But we will not go that way to implement this project. Instead of that what we will going to do is that we will going to use this Groq engine or Groq Cloud.

Now with the help of this Groq what particular inferencing engine what so special about this “Groq”

Go lets read about the [“Groq”]

Definition : Groq is on a mission to set the standard for “GenAI” inference Speed, helping real-time AI applications come to life today.

One problem is with working with this LLMs is Specifically with respect to inferencing how quickly we are able to get the response.

If I will consider this “Groq” Platform, it uses something called as – LPU Inferencing Engine.

LPU Inferencing Engine is nothing but LPU Stand for – Language Processing unit. It is a new type of end to end processing unit system.

That provide the faster inference for computationally intensive application with a sequential component to them such as AI language application.

What is the main this of LPU over here is that, it is pretty much faster for the Inferencing purpose, it is much more faster then **GPUs**.

Q. Why it is so much faster then the GPU for LLM and GenAI?

Answer :

LPU is designed to overcome the two LLM bottlenecks compute density and memory bandwidth.

LPU has a great compute capacity than a GPU and a CPU in regards to LLM this reduces the amounts of time per word calculated. Allowing sequences of text to be generated much faster.

Additionally, eliminating external memory bottlenecks enables the LPU Inference Engine to deliver orders of magnitude better performance on LLMs compared to GPUs.

That is the region why LPU is very much important, there is also a research paper , we can refer if we want to go deep dive into it.

If we specifically called with respect to “Groq” it provides us API and if we see top right corner, this platform has almost every open sources model.

For example :

(A) Gemma-7b-it - Here Gemma 7 Billion parameter.

Note :

If we want to use 2 Billion parameters, for that we probably use it from hugging face.

But right now,

(B) Llama3-70b-8192   -> this “Groq” platform provides the Lama3 **70 billion**

(C) Llama3-*b-8192 -> This provides the lama3 **8 Billion**

(d) Mixtral-8x7b-32768

Which all are open sources model. We can use any of these particular open source model based on the project that I am currently doing, I am use any one of them that is fine.

For example :

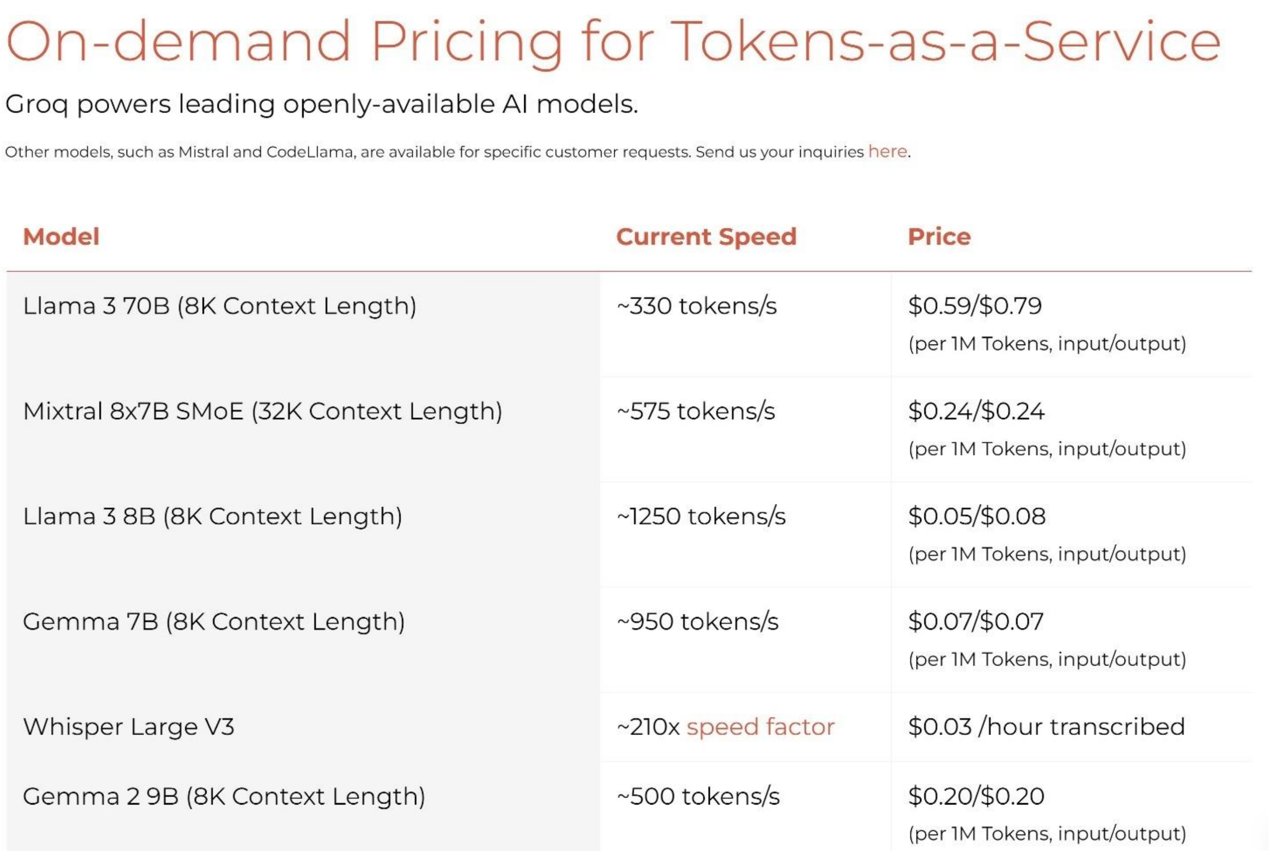
Lets select this (A) Gemma-7b-it and

I will just asked one question, [What is Generative AI] ?

- Just hit that by default question, the answer I am getting was so fast, within a sec its **788.30 T/S or Token per second.**

Lets go and check the pricing part also.

Here we can see pricing with respect to the different different model.


On-demand Pricing for Tokens-as-a-Service

Groq powers leading openly-available AI models.

Other models, such as Mistral and CodeLlama, are available for specific customer requests. Send us your inquiries here.

Model	Current Speed	Price
Llama 3 70B (8K Context Length)	~330 tokens/s	$0.59/$0.79 (per 1M Tokens, input/output)
Mixtral 8x7B SMoE (32K Context Length)	~575 tokens/s	$0.24/$0.24 (per 1M Tokens, input/output)
Llama 3 8B (8K Context Length)	~1250 tokens/s	$0.05/$0.08 (per 1M Tokens, input/output)
Gemma 7B (8K Context Length)	~950 tokens/s	$0.07/$0.07 (per 1M Tokens, input/output)
Whisper Large V3	~210x speed factor	$0.03 /hour transcribed
Gemma 2 9B (8K Context Length)	~500 tokens/s	$0.20/$0.20 (per 1M Tokens, input/output)

As we can 1st one **Llama3 70b(8k Context length)** | Speed current is 300 Tokens per sec | price is $0.59 (per 1M tokens, input / output)

Similar others also we can see all these all pricing parts.

But as of now, for our this project, I will going to use only Gemma 7B (8k Context Length) to create this end to end project.

Next Step,

Now lets see how to create a API KEY.

For that just down below we can see the (Groq Cloude developer console). If we hit into that

New window show where I am see a playground

Right corner there is alots of model options are there to select and use it. So but I will use as for my project - (Gemma-7b-it).

But here what I will do is that,

I will create a API KEYs -> I will name it as a **(Test)** and submit or save it.

Once I submit it. One pop up window open with my API KEYS – just copy it save it somewhere in word file to use.

[GROQ_API_KEY="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"]

This API KEY I will going to use to my project. End to end document Q/A .

So till so far I have cover all the things how we will do the entire projects and the model that I am going to use over here.

STEPS to Create a Model from scratch :

Step 1:

Open the Pycharm or VS code and go to the terminal section and I will create a new environment.

For that, I will paste this code inside my terminal.

```
C:\Users\Public\Music > conda create -p venv python==3.12
```

[conda create -p venv python==3.12] - this is my recent version with respect to the Python.

Press – **YES**

Step2 :

Folders need to create :

GEMMA inside this –

1.Us_census

2.venv

3.vnv

4. .env

5.app.py

6.requiredments.txt

1. (Venv)

2. (.env) – (Now I will create new folder called as .env this is basically for my environment variable. Inside my .env folder, I am going to create 2 environment variable.

[GROQ_API_KEY="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"]
[GOOGLE_API_KEY="AIzaSyA2kz7TMs1lMyVqBkEnK7xMvNm5K_GDj0Q"]

Note :

I am going to use my API key, everybody has different, so and create it from the **Groq Cloud.**

Now, my GOOGLE API KEY –

To get this, we have to go the google and search for the **[ai.google studio API Key]**

Very 1st select it, inside go and get the GOOGLE API KEY –

Select the get API Key -> create API Keys -> Copy that code and come back and paste it to the my folder **(.env).**

Now, we have 2 API Keys, one is from GOOGLE and another is GROQ.

Question, Why I am using GOOGLE API Keys?

Because, I am going to use some of the embedding techniques which is completely freely available from the google itself.

Suggested not to use it, later I will delete it.

3. (Requiredments.txt)

Now I am going to create a requiredment.txt because for this project, I need to have some libraries.

Now go to the terminal before this Requirements.

Paste this code below my **terminal.**

[conda activate venv/]

Now come to the requirement folder and I will write here all the libraries that I am going to use it over here.

1. faiss-cpu (This is for my vector embedding)
2. groq
3. langchain-groq
   (I will install and use this langchain_groq over here, because this is amazing libraries that specifically used for the framework to create amazing GenAI Application)
4. PyPDF2 (This library specifically used to read PDFs and probably document from the PDFs.
5. langchain_google_genai
6. langchain
7. streamlit (because we are going to develop some streamlit application)
8. langchain_community
9. Python-dotenv (So that we will install to call the environment variable)
10. Pypdf (I will import PyPDF)

These all requirements, I am going to use into my project.

Step3 :

Now I am go to my terminal section and

[pip install -r requiredments.txt]

Once this installation take place, Now lets start my coding part.

Step 4 :

Create a folder name it as a **[app.py]** Inside this I am going to write all my codes.

Note :

Now go to the terminal section open new terminal and check the

[python app.py]

Next,

[conda activate venv/]

Next,

[cls] - clear the screen.

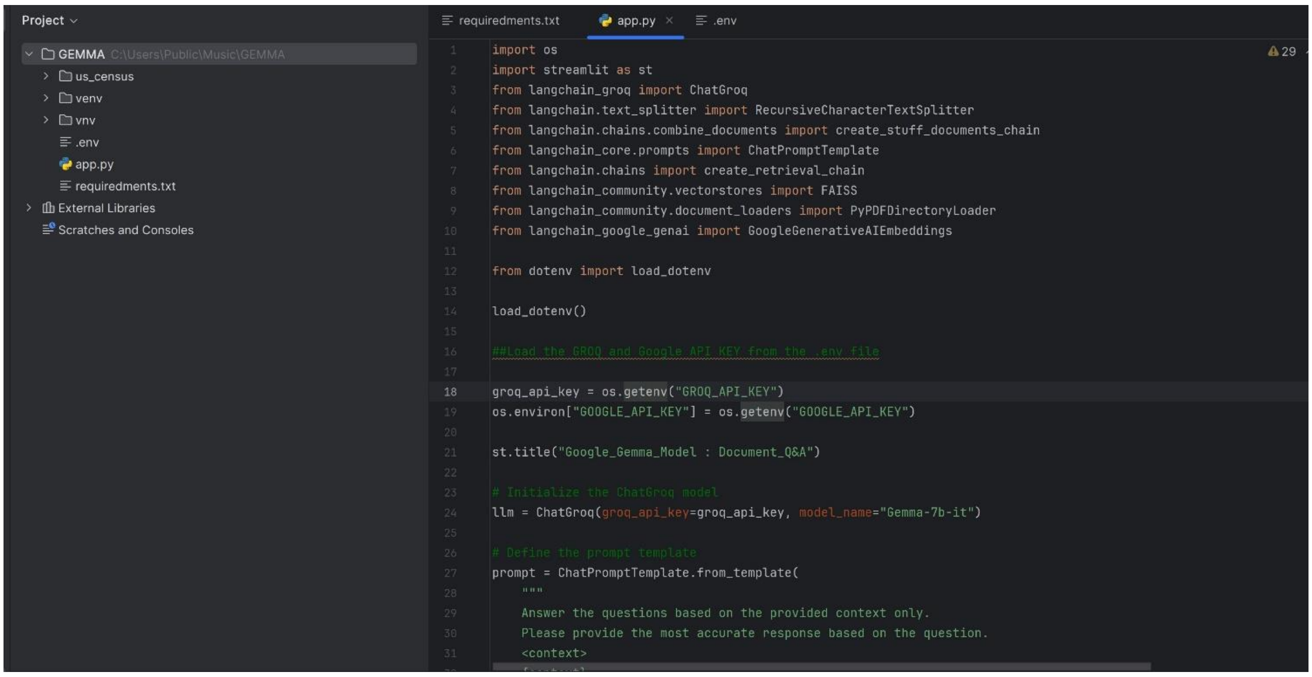
Next,

[python app.py]

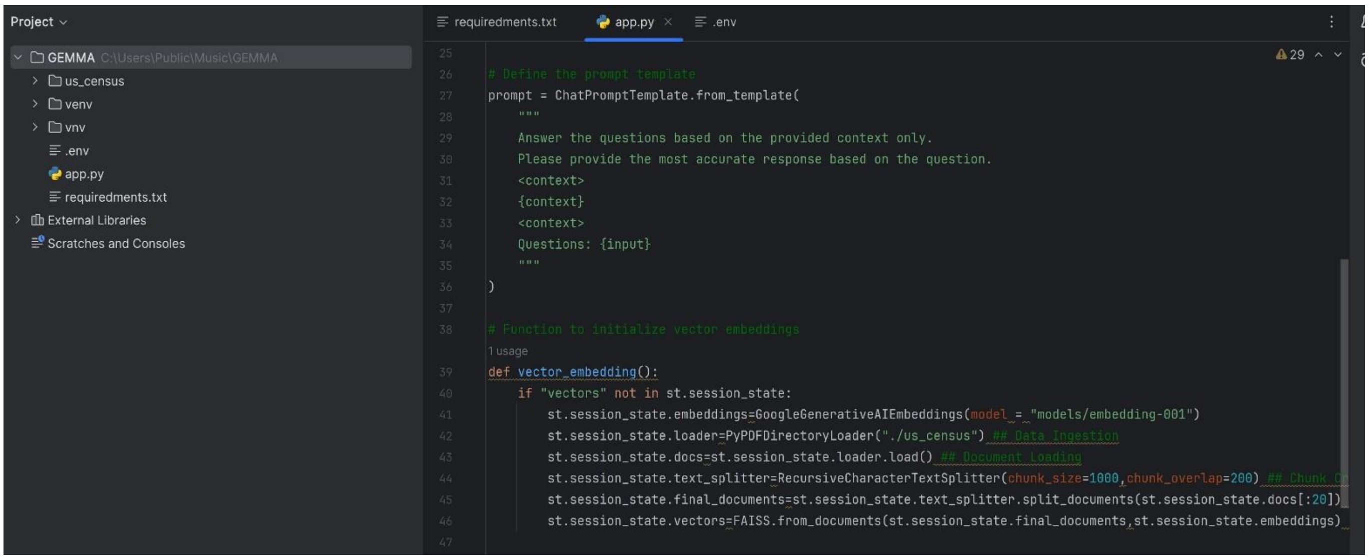
Why I am writing this, because my code should be ready and it should be able to highlight these all libraries.

Next step,

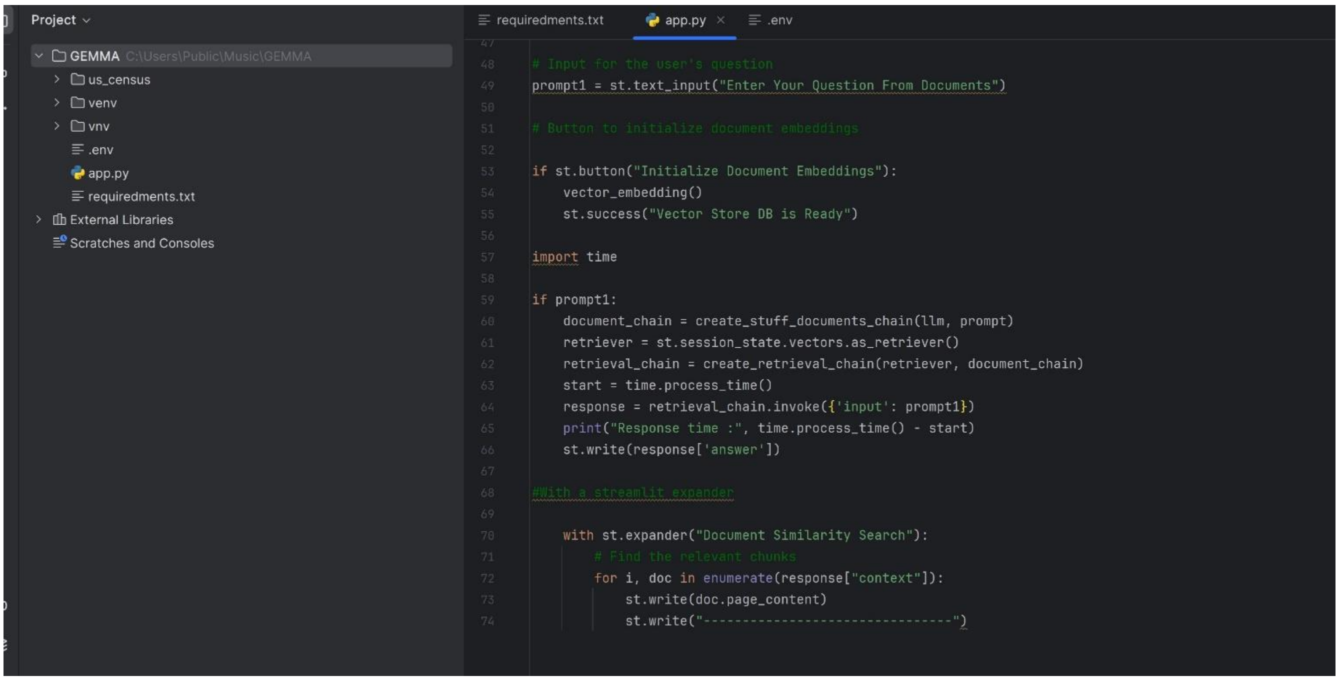
Now go back to the folder **[app.py]**   below is my all codes.



```
import os
import streamlit as st
from langchain_groq import ChatGroq
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain_core.prompts import ChatPromptTemplate
from langchain.chains import create_retrieval_chain
from langchain_community.vectorstores import FAISS
from langchain_community.document_loaders import PyPDFDirectoryLoader
from langchain_google_genai import GoogleGenerativeAIEmbeddings

from dotenv import load_dotenv

load_dotenv()

##Load the GROQ and Google API KEY from the .env file

groq_api_key = os.getenv("GROQ_API_KEY")
os.environ["GOOGLE_API_KEY"] = os.getenv("GOOGLE_API_KEY")

st.title("Google_Gemma_Model : Document_Q&A")

# Initialize the ChatGroq model
llm = ChatGroq(groq_api_key=groq_api_key, model_name="Gemma-7b-it")

# Define the prompt template
prompt = ChatPromptTemplate.from_template(
    """
    Answer the questions based on the provided context only.
    Please provide the most accurate response based on the question.
    <context>
```



```

# Define the prompt template
prompt = ChatPromptTemplate.from_template(
    """
    Answer the questions based on the provided context only.
    Please provide the most accurate response based on the question.
    <context>
    {context}
    <context>
    Questions: {input}
    """
)

# Function to initialize vector embeddings
def vector_embedding():
    if "vectors" not in st.session_state:
        st.session_state.embeddings=GoogleGenerativeAIEmbeddings(model = "models/embedding-001")
        st.session_state.loader=PyPDFDirectoryLoader("./us_census") ## Data Ingestion
        st.session_state.docs=st.session_state.loader.load() ## Document Loading
        st.session_state.text_splitter=RecursiveCharacterTextSplitter(chunk_size=1000,chunk_overlap=200) ## Chunk C
        st.session_state.final_documents=st.session_state.text_splitter.split_documents(st.session_state.docs[:20])
        st.session_state.vectors=FAISS.from_documents(st.session_state.final_documents,st.session_state.embeddings)

```



```python
# Input for the user's question
prompt1 = st.text_input("Enter Your Question From Documents")

# Button to initialize document embeddings

if st.button("Initialize Document Embeddings"):
    vector_embedding()
    st.success("Vector Store DB is Ready")

import time

if prompt1:
    document_chain = create_stuff_documents_chain(llm, prompt)
    retriever = st.session_state.vectors.as_retriever()
    retrieval_chain = create_retrieval_chain(retriever, document_chain)
    start = time.process_time()
    response = retrieval_chain.invoke({'input': prompt1})
    print("Response time :", time.process_time() - start)
    st.write(response['answer'])

#With a streamlit expander

    with st.expander("Document Similarity Search"):
        # Find the relevant chunks
        for i, doc in enumerate(response["context"]):
            st.write(doc.page_content)
            st.write("--------------------------------")
```

BELOW IS THE DETAIL CODE EXPLANATION STEP BY STEP :

Step1 _Code :

import os
import streamlit as st

# Since I am going to use "Groq" so, in Groq we have to chat groq. So, that we will be able to create a chatbot.
from langchain_groq import ChatGroq

# Once I probably read any documents, I should be able to convert that into chunks. So, for that I will be using from (langchain.text_splitter) and [import RecursiveCharacterTextSplitter].

from langchain.text_splitter import RecursiveCharacterTextSplitter

# along with this I am using one more libraries called [langchain.chains.combine_documents] and I am going to create [import create_stuff_documents_chain]. So, in Langchain libraries, we use this create stuff documents chain for the relevant documents and Q & A. This is will basically help us to set up the context.

from langchain.chains.combine_documents import create_stuff_documents_chain

# Here, we will also going to use [ChatPromptTemplate]. So, that we will be able to create a our own custom prompt template. So, here we are writing from [langchain_core.prompts import ChatPromptTemplate]. This is my vector store DB.

from langchain_core.prompts import ChatPromptTemplate
from langchain.chains import create_retrieval_chain

# Now, here we will be embedding our vector store DB that we are specifically going to use and for that we are going to import from the [langchain_community.vectorstores import FAISS] .

“FAISS” is kind of a vector store that has been created, by Meta and internally it is basically used to store vectors and it internally performs semantic search and similarities search to give us the result based on the information that I have asked.

from langchain_community.vectorstores import FAISS

# Here, I will also importing my libraries which is [langchain_community.document_loaders],

# we are using [PyPDF] directory because our main aim is that we will be reading some PDF files that from our folder and then we will be reading all the documents and then, we will be dividing chunk using this recursive character Text plater.

from langchain_community.document_loaders import PyPDFDirectoryLoader

# Now, finally we also going to use one more library, that is our embedding technique for that we are going to import from the [langchain_google_genai] because Google Already provide this, Google GenAI embedding which we can completely use it. Its completely freely available, just by using the “GOOGLE_API_KEY”.

# These [GoogleGenerativeAIEmbeddings] will be responsible to converting my text, a chunks of text into the vectors. This is my vector Embedding techniques.

from langchain_google_genai import GoogleGenerativeAIEmbeddings

Next,

Step code2 :

# Now from dotenv we are going to import load_ underscore dotenv.

# I am using this libraries region is that,so that we can actually go ahead and load all our environment variables.

```
from dotenv import load_dotenv

load_dotenv()
```

Next,

Step code 3 :

## Now, Load the GROQ and Google API KEY from the [.env] or environment variable file.

# How do I load it, for that [groq_api_key = os.getenv("GROQ_API_KEY")]

# Here , I am using my Amazon Q as my code assistance over here.So, automatically I am able to get the suggestion.

# So, the both below environment variable is created in my [.env] file or folder.

```
groq_api_key = os.getenv("GROQ_API_KEY")
os.environ["GOOGLE_API_KEY"] = os.getenv("GOOGLE_API_KEY")
```

Next,

Step code 4 :

# lets st.title here I ("Google_Gemma_Model : Document_Q&A"). Here specifically we are using "Groq".

st.title("Google_Gemma_Model : Document_Q&A")

# Here, I am just calling my LLM model, here I am going to use my "ChaGroq", with respect to that, I am going to use my GROQ_API_KEY and specifically say model_name, and here I am going to give my Model name, which is – [Gemma-7b-it].

# This particular model, I am using it and with the help of GROQ_API_KEY, I will be able to call it.

```
llm = ChatGroq(groq_api_key=groq_api_key, model_name="Gemma-7b-it")
```

Next,

Step code 5 :

# Now here what we will do is, we will going to set up our Prompt template.To set up Prompt template, I will be using this same [ChatPromptTemplate.from_template] & I am going to write these prompt as –

""""
# Answer the questions based on the provided context only.
Please provide the most accurate response based on the question.
<context>
{context}   # The context will be this, from this particular context,
<context>
Questions: {input}   # It will going to take this particular input over here.
""""

# By taking this all information, it is going to give us the entire information.

```
prompt = ChatPromptTemplate.from_template(
""""
Answer the questions based on the provided context only.
Please provide the most accurate response based on the question.
<context>
{context}
<context>
Questions: {input}
""""
```

Next,

Step code 6 :

# So, now we are going to create a function which is called as Vector Embedding & this vector Embedding function, It is going to do is that. We will be reading all the documents from our PDF Files.

# From those PDF files, we also going to convert these into chunks then apply embeddings.

# We are going to apply "GOOGLE_GENERTIVE_AI Embedding and then finally, we will be storing it in a vector store DB, that is called as "FAISS".

# We will going to keep this vector store DB, even in our session state so that we will be able to use it anywhere when it is required.

# So 1st of all that, I am going to do over here is that, I am going to make sure that one type of PDFs, I should be able to upload over here. I will create a new folder name it as a – **[us_census]**

# I will save this folder with **4 PDF files** inside it and save it inside my repository **directory,** where is

# my all files stores + codes and every available.

# This is my Dirctory inside my system – [C:\Users\Public\Music\GEMMA>]

# This 4 PDF specifically we are going to use it.

[def vector_embedding():]

# Now here, 1st vector store DB, that I am going to create in a variable that variable I will just go ahead and write it somewhere like this.

# So, lets go and write it.

# Next Line 1 :

# If vectors, since I also need to use session states, not in St.Session_States, So I will not

[if "vectors" not in st.session_state:]

# So, I will make code very simple, because I need to save everything in session states with respect to different different variables.

# If the vector is not in the session states, 1st things I am going to write over here is that,

# go ahead & write the st.session_state.embeddings, and 1st of all I am going to define my # embeddings for that, I am going to use my [**GoogleGenerativeAIEmbeddings]** inside this I am going to use use my model, which is basically called as **[models/embeddings-001].** This is the one of the model, which is available in Google. For the embedding purpose.

# Now I am actually going to stay share in the form of the session states with this variable that is called as "Embeddings".

[st.session_state.embeddings=GoogleGenerativeAIEmbeddings(model = "models/embedding-001")]

Next line 2 -

# we will write [st.session_state.loader= here I am specifically use the PyPDF Directory and my folder name is where I have kept my all 4 PDF files. Name is – **[us_census].**

# This is basically my Data injection phase, once I read it I am storing entire loader in a session states with this particular variable name. This is my basically **"Data Ingetsion".**

[st.session_state.loader=PyPDFDirectoryLoader("./us_census")]

Next line 3 -

# St.session_states.docs= & I am going to create another variable called as docs. Let write it st.session_underscore state.loader.load.

# When we use this loader.load in short, it is going to load all the documents.This is basically loading all the documents.

[st.session_state.docs=st.session_state.loader.load() ## Document Loading]

# Next Line 4 –

# I am going to use this st.session_states.text_Splitter, because when we write the loader.load, we will specifically get all the docments.Inside my particular variable Docs.

# So, just by using this [st.session_state.text_splitter] This is my variable name. Here, we will be going to use [RecursiveCharacterTextSplitter] where in all these documents will be Splitted into chunks. Here we will going to use a chunk size of [Chunk_size = 1000 and chunk_overlap = 200].

# In short, these [RecursiveCharacterTextSplitter] function, which is going to take out the documents and Splits based on this Chunk_Size and Chunk_overlap, Chunk_overlap is basically means there is a overlap of characters.

[st.session_state.text_splitter=RecursiveCharacterTextSplitter(chunk_size=1000,chunk_overlap=200)]

# Finally, I will go ahead and write the st.session_State. And basically I am going to create my final_documents.

# And lets use this text Splitter [st.session_state.text_splitter.split_documents] and here we are going to use entire docs – [st.session_state.docs[:20])]

# That is the region, we are going have given the **[st.session_state.docs[:20])]**

# Region why we are saving this all in this **session_States**, because we should be able to use it anywhere that we required and finally after doing this

[st.session_state.final_documents=st.session_state.text_splitter.split_documents(st.session_state.docs[:20])]

# Next Line 5 –

# Finally, after doing this, we will finally create our vector_Store a Vector OpenAI Embeddings

# st.session_state.vectors and I am going to use this “FAISS”. From documents and here I am going to use this – [st.session_state.final_documents] and the Embeddings techniques it will be the same Embedding techniques that we have initial over here that is nothing, but **Google Generative_AI** Embedding.

And this embedding **–** [st.session_state.embeddings] Will be responsible for the converting this all final documents into vectors, and this “FAISS” will be responsible in storing all those embeddings into this particular vectors.

# This is the Function that is probably doing all these things.

[
st.session_state.vectors=FAISS.from_documents(st.session_state.final_documents,st.session_state.embeddings) #vector OpenAI embeddings]

Next,

Step code 6 :

# After close the function, Here I have crated the field, so lets write.

# prompt1 and lets create the st.text_input("Enter Your Question From Documents").

# Here basically, what I want to asked from the documents.

[prompt1 = st.text_input("Enter Your Question From Documents")]

# Here, I am creating the Button, This Button will be responsible for on - **Initialize Document Embeddings**.

# This basically means, If I click this button then my entire process of this vector embedding should happened. So, here what I am calling my vector embedding,

# when this entire embedding is created, I will just go-ahead and write, my vector store my vector DB is ready. Because from these vector store, I am going to do is that, quey anything.

# Before that we required this vector_states Variable, this vector DB should be there.

# So, If I will click this below button, it automatically that vector embedding will created.

```
[ if st.button("Initialize Document Embeddings"):
    vector_embedding()
    st.success("Vector Store DB is Ready")    ]
```

Next,

Step code 8 :

# Here, lets try to create something with respect to the **"time"**

```
[ if prompt1:
    document_chain = create_stuff_documents_chain(llm, prompt)
    retriever = st.session_state.vectors.as_retriever()
    retrieval_chain = create_retrieval_chain(retriever, document_chain)
    start = time.process_time()
    response = retrieval_chain.invoke({'input': prompt1})
    print("Response time :", time.process_time() - start)
    st.write(response['answer'])    ]
```

# Because, I am saying that, this is really really very fast, This "Goq_LPU" is really very fast.

# So, lets record the "time" also, with that we will be able to understand the importance of it.

# So, if prompt1 :

# when basically, I am writing any text and press enter, I should take this particular input and create my document chain.For that, I am using this **[create_stuff_documents_chain]**

# inside this, there is 2 parameters will be given, one is **(llm, prompt)**. The LLM model, I am using here is nothing but - **[Gemma-7b-it]**.

# And the 2nd parameters, that I am using is basically nothing but – [Prompt].Both these things are available.

[import time]

```
[ if prompt1:
    document_chain = create_stuff_documents_chain(llm, prompt)    ]
```

# Next Step 9 :

# Now, I will go-ahead and create a [retriever] and this is basically be **[st.session_state.vectors.as_retriever()]** . what is the main functionality of this particular retriever,

This vector is a vector database, now to retriever information from this vectors, vector database, with the help of this particular function as retriever, it create a interface. So, whatever question basically we will ask through this interface, it will be able to take this particular response and it will be able to give to the end user. That is the region we specifically used the “Retriver”.

[retriever = st.session_state.vectors.as_retriever()]

# Next Step 10 :

# After creating this retriever, I really need to run this in the form of chain, where I have my retriever, where I have my documents chain.Boths needs to be combined. So basically we are creating a retriever_chain.

[retrieval_chain = create_retrieval_chain(retriever, document_chain)]

# Next Step 11 :

# Here, I am going to start my time, I am going to say - [time.process_time()].

[start = time.process_time()]

# Then, I will go ahead and write my response, which is nothing but it will be “Retriver_chain”

# And, we will going to call the **invoke function**. Inside this Invoke function, we can keep our variable as **({'input': prompt1})** input and this will be equal to my prompt1.

# Because I am going to sent, whatever input, I am giving in this particular prompt with respect to my question, it should be able to retrieve, from this entire chain. Then finally I will get my response.

[response = retrieval_chain.invoke({'input': prompt1})]

# Next Step 12 :

# After get my entire response, basically I am going to do is display that entire response.In my streamlit APPs.

# Note :

# When this Gemma Model provide a response, it will also provide some kind of context in return.

# I will try to display that content over here

```
  [ print("Response time :", time.process_time() - start)
   st.write(response['answer'])   ]
```

# Next Step 13 :   [StreamLit]

# This below is my final code for Streamlit, step by step explanation:

# With a streamlit expander

```
   [ with st.expander("Document Similarity Search"):
       for i, doc in enumerate(response["context"]):
           st.write(doc.page_content)
           st.write("--------------------------------")     ]
```

# Here,this line creates an interactive expander in a **Streamlit app**. An expander is a collapsible container that can be expanded or collapsed by the user. Initially, it shows as a heading labeled **"Document Similarity Search,"** and users can click to expand or collapse it.

Streamlit Context: The with statement here is used to create a context where all the code inside it will be executed only if the expander is opened by the user. This is similar to how we would work with **st.sidebar or st.form**.

```
[ with st.expander("Document Similarity Search"):   ]
```

# Next Line 14 :

# This line is a for loop that iterates over the elements of response["context"].

# Enumerate function: The enumerate function adds a counter to the iteration. The variable I will represent the index (starting from 0) of each doc in the response["context"] list.

# Response["context"]: This is likely a list of documents or chunks of text that have been retrieved as part of a document similarity search. Each doc represents a chunk or document that has been identified as similar.

# response of ["context] through which I will be able to get my entire context information.

# **i, doc** basically have the page content, which will get display.

# The region I am using enumerate function over here is that, because there will be 2 values.

[for i, doc in enumerate(response["context"]):]

# Next Line 15 :

# This line writes the content of each document (doc.page_content) to the Streamlit app. The st.write function can display text, markdown or even more complex objects.

# doc.page_content: This likely refers to the actual content (text) of the document or chunk that is being displayed.

[st.write(doc.page_content)]

# Next Line 16 :

# This line writes a visual separator (a row of dashes) between the different documents being displayed. This helps to visually distinguish between different chunks or documents in the app.

# Why a separator? After each document's content is displayed, a line of dashes is added to separate it from the next document, making the output easier to read and understand.

[**st.write("--------------------------------")]**

# Next Line 17 :   [Go to the " Terminal" section to see the final outcome]

Finally, Lets go-ahead and run this entire code to see my final outcome, in my streamlit.

[streamlit run app.py]

Project Deployed successfully :

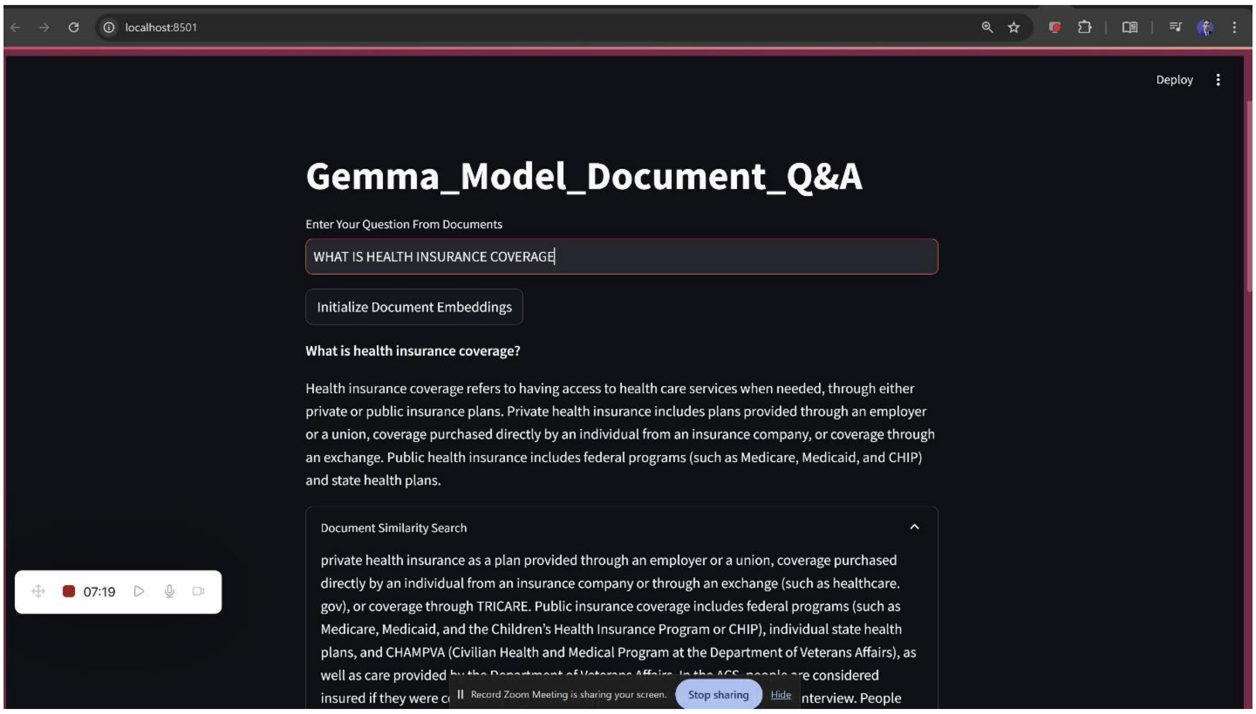
# If we run this above line of code inside my terminal, pop up open and in the browser we can see my final Apps has been Deployed successfully.

# This is my title of the project –

[Google Gemma Model Documenst Q & A   ]

[Documents Embedding]   - Here if we click, just wait few second, it will create the vector DB and as message it will show that -

[Vector store DB is Ready   ]



Gemma_Model_Document_Q&A

Enter Your Question From Documents

WHAT IS HEALTH INSURANCE COVERAGE

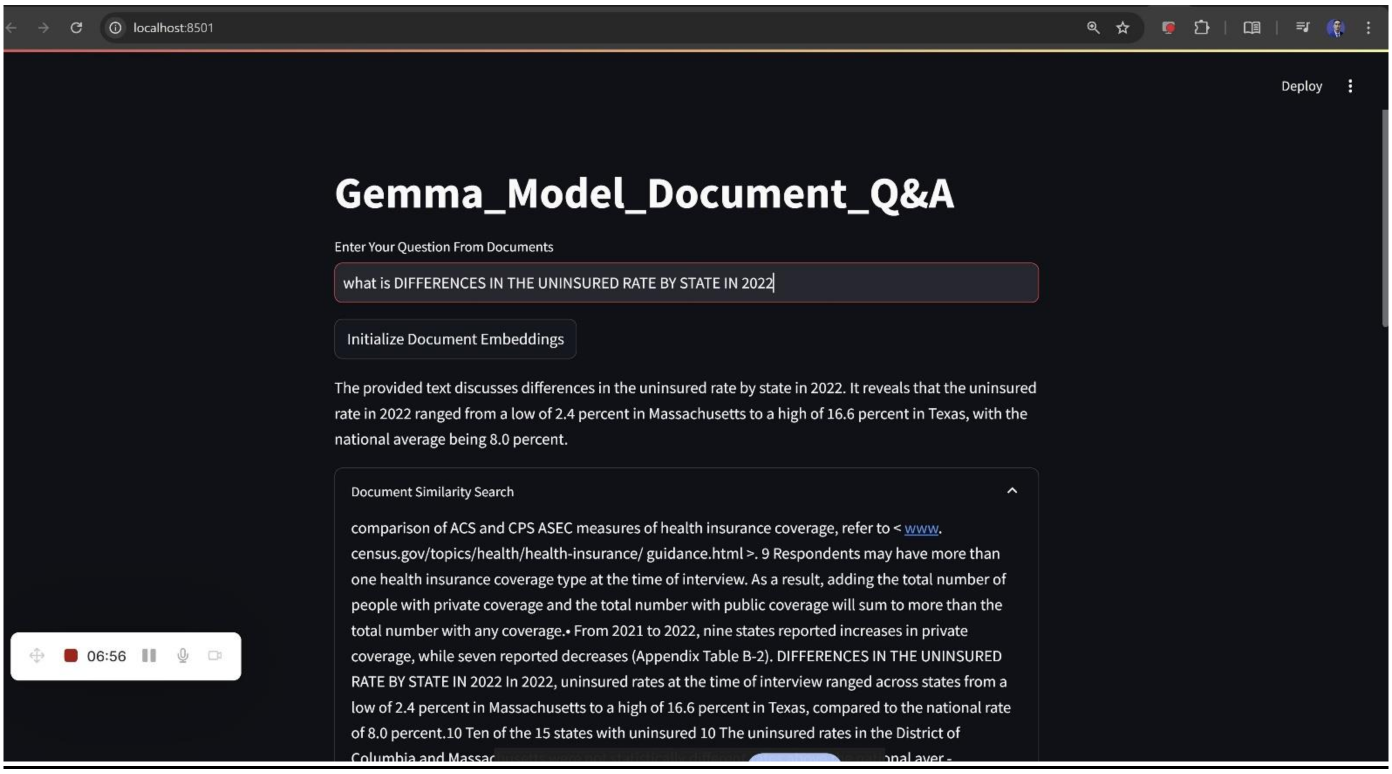
Initialize Document Embeddings

What is health insurance coverage?

Health insurance coverage refers to having access to health care services when needed, through either private or public insurance plans. Private health insurance includes plans provided through an employer or a union, coverage purchased directly by an individual from an insurance company, or coverage through an exchange. Public health insurance includes federal programs (such as Medicare, Medicaid, and CHIP) and state health plans.

Document Similarity Search

private health insurance as a plan provided through an employer or a union, coverage purchased directly by an individual from an insurance company or through an exchange (such as healthcare. gov), or coverage through TRICARE. Public insurance coverage includes federal programs (such as Medicare, Medicaid, and the Children's Health Insurance Program or CHIP), individual state health plans, and CHAMPVA (Civilian Health and Medical Program at the Department of Veterans Affairs), as well as care provided by the Department of Veterans Affairs. In the ACS, people are considered insured if they were c nterview. People



Gemma_Model_Document_Q&A

Enter Your Question From Documents

what is DIFFERENCES IN THE UNINSURED RATE BY STATE IN 2022

Initialize Document Embeddings

The provided text discusses differences in the uninsured rate by state in 2022. It reveals that the uninsured rate in 2022 ranged from a low of 2.4 percent in Massachusetts to a high of 16.6 percent in Texas, with the national average being 8.0 percent.

Document Similarity Search

comparison of ACS and CPS ASEC measures of health insurance coverage, refer to < www. census.gov/topics/health/health-insurance/ guidance.html >. 9 Respondents may have more than one health insurance coverage type at the time of interview. As a result, adding the total number of people with private coverage and the total number with public coverage will sum to more than the total number with any coverage.• From 2021 to 2022, nine states reported increases in private coverage, while seven reported decreases (Appendix Table B-2). DIFFERENCES IN THE UNINSURED RATE BY STATE IN 2022 In 2022, uninsured rates at the time of interview ranged across states from a low of 2.4 percent in Massachusetts to a high of 16.6 percent in Texas, compared to the national rate of 8.0 percent.10 Ten of the 15 states with uninsured 10 The uninsured rates in the District of Columbia and Massac onal aver -

Highlights:

After successfully deploying my project, I searched for **2 titles** from a single PDF file. Overall, I have 4 PDF files in my directory."

I have search these below 2 titles :

1. **[WHAT IS HEALTH INSURANCE COVERAGE]**
2. **[WHAT IS DIFFRENCES IN THE UNINSURED RATE BY STATE IN 2022]**

Steps :

# Now here I can asked any question from my PDF documents that I already have store inside folder - **[US_Census]**

Just open any Pdf File, lets say – I will open my 1st PDF file, which name as - **[acsbr-015],** it's a 1st PDF file name that I have given.

This is basically a – **[Health Insurance Coverage]** PDF file, similarly others 3 PDF files is there with me.

# Juts open the PDF and copy any Headline of the from my PDF.

# Lets copy this – **[WHAT IS HEALTH INSURANCE COVERAGE?]**

# Just copy it and paste it over the my Q & A section.

# As we can see the final result or outcome a quick response.

# Note :

# In my code I am also mention that, Time is = to 0.1 sec, pretty fast with the help of LPU Inferencing engine we can see all the information or context specifically got.

# It explained that copy & pasted title that I have pasted from my PDF files.

# Similar way, if we copy and paste others Titles from the PDF files, it will give me the information accordingly.