Exploring Mental Health using LLMs: Comparision between Chatgpt and Gemini

Google Colab was used extensively in our project because it is highly compatible with Jupyter Notebooks for the analysis and modeling of data. It is a hosted Jupyter Notebook service that does not require setup and provides free access to computing resources. Short for Google Colaboratory, Google Colab is a cloud-based platform by Google that allows its users to write and execute Python codes within a web-based interactive environment. Users can write code, run it, and view the results in web browsers.

Besides, Google Colab can access popular machine learning libraries like TensorFlow, PyTorch, and sci-kit-learn, hence we used them in data analysis and collective research. We can also share our notebooks anywhere in the world with internet access. It is affordable, with low search costs, and can be executed on free online services. Summing up, with Google Colab, one can make a setup that effortlessly creates an atmosphere for coding, experimenting, and collaborating with communities in many different areas.

INSTALLATION GUIDE/TOOL USED

Here are the steps to use Google Colab:

1. Open the browser and visit the website for Google Colab by typing in the URL colab.research.google.com.
2. If you are not already signed in, sign in with your Google account.
3. Click on "New Notebook" after you are signed in. You will be directed to a new notebook in which you can write and execute code. You have a cell where you can run Python code on the notebook.
4. Write the code in the cell and hit Shift + Enter to execute. We can add more cells using the "+" button.
5. Add Text and Markdown. A text cell can be used to explain and comment on the code. Click "+" and select "Text". Use Markdown syntax to format your text.
6. Your Google Colab notebook will automatically save as you work, but you can also do this manually by selecting "File," then "Save, or "Save a Copy in Drive."
7. The share button up at the top right corner allows one to share a notebook with others. It is possible to limit access only to some particular people or make it public.
8. You can directly access data in your Google Drive, or Google Sheets or simply upload to your Colab, with files residing in your Google account.

DOCUMENTATION OF CODE

The following modules and packages were installed and utilized in our code:

pip install datasets

```
#installing datasets
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.18.0-py3-none-any.whl (510 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 3.9 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets) (3.13.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.25.2)
Requirement already satisfied: pyarrow>=12.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (14.0.2)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/dist-packages (from datasets) (0.6)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 14.0 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (2.0.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.2)
Collecting xxhash (from datasets)
  Downloading xxhash-3.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 18.9 MB/s eta 0:00:00
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 15.0 MB/s eta 0:00:00
Requirement already satisfied: fsspec[http]<=2024.2.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.9.5)
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.20.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (24.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.19.4->datasets) (4.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.7)
```

```
Installing collected packages: xxhash, dill, multiprocess, datasets
Successfully installed datasets-2.18.0 dill-0.3.8 multiprocess-0.70.16 xxhash-3.4.1
```

It is used to install the datasets library from pip, which is a package for Python. It provides high-level access to various datasets for natural language processing (NLP) and machine learning. We will import and use the datasets library in our code easily after running this command in the Python environment to load and manipulate the data for the project.

1. Data Collection

```
# Loading dataset

import pandas as pd

from datasets import load_dataset

mentalhealth_dataset = load_dataset("alexandreteles/mental-health-conversational-data")

mentalhealth_dataset
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
Downloading readme: 100% 274/274 [00:00<00:00, 18.4kB/s]
Downloading data: 100% 21.5k/21.5k [00:00<00:00, 92.4kB/s]
Generating train split: 100% 661/661 [00:00<00:00, 11352.20 examples/s]
DatasetDict({
    train: Dataset({
        features: ['Context', 'Knowledge', 'Response'],
        num_rows: 661
    })
})
```

import pandas as pd

Used to import the pandas library(pd) for ease of use. Pandas is a powerful library used in Python for data manipulation and analysis. By importing pandas as pd, we can use the pd prefix to access pandas functions and classes throughout the code.

from datasets import load_dataset

Used to import load_dataset function from library dataset in Python using Hugging Face Datasets library. After importing this function, we can import the datasets and start using it. We imported the dataset from Hugging Face which is "alexandreteles/mental-health-conversational-data".

Reading the dataset:

```
#reading dataset using pandas

df=pd.DataFrame(mentalhealth_dataset['train'])
df
```

	Context	Knowledge	Response
0	Hi	greeting	Hello there. Tell me how are you feeling today?
1	Hi	greeting	Hi there. What brings you here today?
2	Hi	greeting	Hi there. How are you feeling today?
3	Hi	greeting	Great to see you. How do you feel currently?
4	Hi	greeting	Hello there. Glad to see you're back. What's g...
...	...	...	...
656	How do I know if I'm unwell?	fact-29	If your beliefs , thoughts , feelings or behav...
657	How can I maintain social connections? What if...	fact-30	A lot of people are alone right now, but we do...
658	What's the difference between anxiety and stress?	fact-31	Stress and anxiety are often used interchangea...
659	What's the difference between sadness and depr...	fact-32	Sadness is a normal reaction to a loss, disapp...
660	difference between sadness and depression	fact-32	Sadness is a normal reaction to a loss, disapp...

661 rows × 3 columns

Downloading the dataset:

```
# downloading dataset as csv

df.to_csv("mental_health_conversational_data.csv", index=False)
df
```

	Context	Knowledge	Response
0	Hi	greeting	Hello there. Tell me how are you feeling today?
1	Hi	greeting	Hi there. What brings you here today?
2	Hi	greeting	Hi there. How are you feeling today?
3	Hi	greeting	Great to see you. How do you feel currently?
4	Hi	greeting	Hello there. Glad to see you're back. What's g...
...	...	...	...
656	How do I know if I'm unwell?	fact-29	If your beliefs , thoughts , feelings or behav...
657	How can I maintain social connections? What if...	fact-30	A lot of people are alone right now, but we do...
658	What's the difference between anxiety and stress?	fact-31	Stress and anxiety are often used interchangea...
659	What's the difference between sadness and depr...	fact-32	Sadness is a normal reaction to a loss, disapp...
660	difference between sadness and depression	fact-32	Sadness is a normal reaction to a loss, disapp...

661 rows × 3 columns

Here the dataset has 661 rows and 3columns

```
#checking rows and columns of dataset
df.shape

(661, 3)
```

2. Data Processing

Double-click (or enter) to edit

```
[ ] # checking for null values

    df.isnull().sum()
```

```
Context      0
Knowledge    0
Response     0
dtype: int64
```

df.isnull().sum()

Checks for the number of missing values in the DataFrame, df, after the previous operation. Method isnull() will return a DataFrame of the same shape as df; however, each element is True if the corresponding element in df is NaN, otherwise False. Then we have the sum() method, and after it has been called, it will return the sum of these boolean values for each column, thus giving the count of missing values for each column of the DataFrame.

```
[ ] # Removed empty strings and none values
    df.replace("",None,inplace=True)
    df.dropna(subset=['Context','Response'],inplace=True)
    df
```

	Context	Knowledge	Response
0	Hi	greeting	Hello there. Tell me how are you feeling today?
1	Hi	greeting	Hi there. What brings you here today?
2	Hi	greeting	Hi there. How are you feeling today?
3	Hi	greeting	Great to see you. How do you feel currently?
4	Hi	greeting	Hello there. Glad to see you're back. What's g...
...	...	...	...
656	How do I know if I'm unwell?	fact-29	If your beliefs , thoughts , feelings or behav...
657	How can I maintain social connections? What if...	fact-30	A lot of people are alone right now, but we do...
658	What's the difference between anxiety and stress?	fact-31	Stress and anxiety are often used interchangea...
659	What's the difference between sadness and depr...	fact-32	Sadness is a normal reaction to a loss, disapp...
660	difference between sadness and depression	fact-32	Sadness is a normal reaction to a loss, disapp...

657 rows × 3 columns

df.replace()

The df.replace() function in pandas is used to replace values in a DataFrame.

df = df.dropna()

This method call will remove any rows from the DataFrame df containing missing values (NaN). It drops all the rows in which at least a single element is missing. The resulting value is then assigned back to the variable df.

3. Exploratory Data Analysis

```python
#checking for most frequent words in both contexts and responses
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Load your dataset
# Replace 'your_dataset.csv' with the path to your dataset file


# Join all contexts and responses into single strings
all_contexts = ' '.join(df['Context'].dropna())
all_responses = ' '.join(df['Response'].dropna())

# Create word clouds for contexts and responses
context_wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_contexts)
response_wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_responses)

# Plot word clouds
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(context_wordcloud, interpolation='bilinear')
plt.title('Context Word Cloud')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(response_wordcloud, interpolation='bilinear')
plt.title('Response Word Cloud')
plt.axis('off')

plt.show()
```

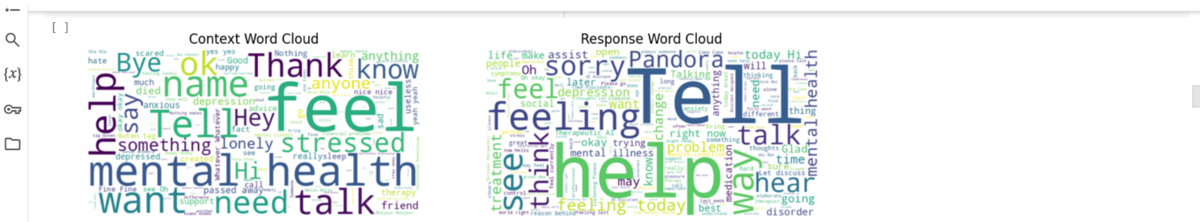
from wordcloud import WordCloud

Import wordCloud from the wordcloud library. The library is used to display words in a word cloud, which is a visual representation to define the significance and frequency of the displayed word size.

import matplotlib.pyplot as plt

Used to create visualizations such as plots and charts. The plt is commonly used for convenience.

These are the visualizations formed for context and response for most frequent words



Finding the value count for the knowledge column and then checking it to >20

```
# value count  for knowledge column

knowledge_count = df['Knowledge'].value_counts()
knowledge_count
```

```
Knowledge
casual              66
greeting            60
about               48
default             40
goodbye             32
                    ..
neutral-response     1
skill                1
pandora-useful       1
morning              1
fact-11              1
Name: count, Length: 79, dtype: int64
```

```
# knowledge_count gretaer than 20

knowledge_count[knowledge_count > 20]
```

```
Knowledge
casual      66
greeting    60
about       48
default     40
goodbye     32
sad         32
done        25
help        21
happy       21
Name: count, dtype: int64
```

Here is the response generated by the chatgpt

4. ChatGPT Response Generation

```
#installing openai
!pip install openai
```

```
Collecting openai
  Downloading openai-1.22.0-py3-none-any.whl (310 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.0/311.0 kB 2.6 MB/s eta 0:00:00
Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from openai) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from openai) (1.7.0)
Collecting httpx<1,>=0.23.0 (from openai)
  Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 7.0 MB/s eta 0:00:00
Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from openai) (2.7.0)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-packages (from openai) (4.66.2)
Requirement already satisfied: typing-extensions<5,>=4.7 in /usr/local/lib/python3.10/dist-packages (from openai) (4.11.0)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->openai) (3.7)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->openai) (1.2.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->openai) (2024.2.2)
Collecting httpcore==1.* (from httpx<1,>=0.23.0->openai)
  Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 8.2 MB/s eta 0:00:00
Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->openai)
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 5.7 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0->openai) (0.6.0)
Requirement already satisfied: pydantic-core==2.18.1 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1.9.0->openai) (2.18.1)
Installing collected packages: h11, httpcore, httpx, openai
Successfully installed h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 openai-1.22.0
```

pip install openai

This a shell command using Python's package manager, pip. Basically, this tries to install a package from the Python Package Index (PyPI), whose package name is "openai".

```
#checking response for one context
import os

from openai import OpenAI

client = OpenAI(
    # This is the default and can be omitted
    api_key="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",
)

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "What's the difference between anxiety and stress?",
        }
    ],
    model="gpt-3.5-turbo",
)
```

import os

The built-in Python module os is interfacing software for the operating system. This module enables to work with many aspects of the operating system, from file paths and environment variables to even running system commands.

from openai import OpenAI

imports a class named OpenAI from a module or package named openai.

chat completion:

```
[ ] chat_completion

ChatCompletion(id='chatcmpl-9FRbI4u8nc6eraDMkfXIJeiQ6C0QD', choices=[Choice(finish_reason='stop', index=0, logprobs=None, message=ChatCompletionMessage(content='Anxiety and stress are often used interchangeably, but they are actually separate experiences with distinct characteristics.\n\nStress is a response to a specific external factor or event that is perceived as overwhelming or difficult to handle. It is a natural and necessary part of life and can be both positive (eustress) and negative (distress). Stress can be caused by a variety of factors such as work deadlines, financial problems, or relationship issues.\n\nAnxiety, on the other hand, is a more generalized feeling of unease, fear, or worry that is often not linked to a specific event or trigger. Anxiety is typically more long-lasting and pervasive than stress and can interfere with daily functioning. It is often irrational and disproportionate to the situation at hand.\n\nIn summary, stress is a response to a specific external factor, while anxiety is a more general feeling of unease or fear that may not be linked to a specific cause. Stress is usually temporary, while anxiety can be chronic and ongoing.', role='assistant', function_call=None, tool_calls=None))], created=1713468044, model='gpt-3.5-turbo-0125', object='chat.completion', system_fingerprint='fp_d9767fc5b9', usage=CompletionUsage(completion_tokens=198, prompt_tokens=16, total_tokens=214))
```

The task of generating or completing a conversational response based on an input prompt or context. This is done through natural language processing (NLP) techniques, using machine learning models, more specifically language models such as OpenAI's GPT (Generative Pre-trained Transformer) series. This chat completion has checked for only one response.

Here we are using gpt 3.5 turbo for generating the reponse for each context

```python
# using gpt-3.5-turbo generating chatgpt response for each context

# Create a new OpenAI client instance
client = OpenAI(api_key="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")

# Initialize an empty list to store the chat responses
chatgpt_responses = []

# Loop through each context in the random_responses list
for context in df['Response']:
    # Create a chat completion request with the current context as the user's message
    chat_completion = client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": context,
            }
        ],
        model="gpt-3.5-turbo",
    )

    # Extract the chat response from the chat completion
    chatgpt_response = chat_completion.choices[0].message.content

    # Append the chat response to the chat_responses list
    chatgpt_responses.append(chatgpt_response)

# Print the chat responses
for chatgpt_response in chatgpt_responses:
    print(chatgpt_response)
```

We have purchased the API key which is mentioned above to authenticate with chatgpt and we get the responses for each and every context.

The response is as follows for each of them:

```
[ ] Hello! I'm just a computer program so I don't have feelings, but I'm here and ready to help you with anything you need. How can I assist you today?
    Hello! I'm here to help answer any questions you may have or assist you with whatever you need. How can I assist you today?
    Hello! I'm just a computer program so I don't have feelings, but I'm here to help you with anything you need. How can I assist you today?
    Thank you for asking. I'm just a computer program, so I don't have feelings in the way humans do, but I'm here and ready to assist you with anything you need. How can I help you tod
    Hello! Thank you for checking in. Currently, I am here and ready to assist you with any questions or tasks you may have. How can I help you today?
    Hello! I'm just a computer program so I don't have feelings, but I'm here and ready to assist you. How can I help you today?
    Hello! I'm an AI assistant here to help you with any questions or tasks you may have. How can I assist you today?
    Hello! I'm just a computer program so I don't have feelings, but I'm here to help. How can I assist you today?
    I'm an AI assistant, so I don't have feelings in the same way humans do. But I'm here and ready to assist you with anything you need. How can I help you today?
    Hello! I'm just here to help answer any questions you may have or have a chat with you. What's on your mind today?
    Hello! As an AI, I don't have feelings, but I'm here and ready to assist you. How can I help you today?
    Hello! I'm here to assist you with any questions or tasks you may have. How can I help you today?
    Hello! I'm just a computer program so I don't have feelings, but I'm here to help you. How can I assist you today?
    Hello! As an AI, I don't have feelings, but I'm here and ready to assist you with anything you need. How can I help today?
    Hello! I'm always here, ready to chat. Right now, I'm just here to answer any questions or have a conversation with you. How can I assist you today?
    Hello! As an AI, I do not have feelings or emotions, but I am here and ready to assist you. How can I help you today?
    Hello! I am an AI assistant here to help answer any questions or assist with any tasks you may have. How can I assist you today?
    Hello! I'm just a computer program, so I don't have feelings, but I'm here and ready to help you with anything you need. How can I assist you today?
    I'm just a computer program, so I don't have feelings in the way that humans do. But I'm here and ready to assist you with anything you need. How can I help you today?
    Hello! Right now, I am here to assist you with any questions or tasks you may have. How can I help you today?
    Hello! I'm just a computer program, so I don't have feelings, but I'm here and ready to help. How can I assist you today?
    Hello! I'm here to assist you with any questions or tasks you might have. How can I help you today?
    Hello! I'm just a computer program, so I don't have feelings in the same way humans do. How can I assist you today?
    Hello! Thank you for asking. I am just a computer program, so I don't have feelings, but I'm here to assist you with anything you need. How can I help you today?
    Hello! Thank you for asking. Right now, I am here and ready to assist you with anything you may need. How can I help you today?
    Hello! I'm just a language model AI, so I don't have feelings, but I'm here and ready to help you with anything you need. How can I assist you today?
    Hello! I am here to assist you with any questions or tasks you may have. How can I help you today?
    Hello! I'm just a computer program, so I don't have feelings, but I'm here to help you with anything you need. How can I assist you today?
    I'm just a computer program, so I don't have feelings in the same way humans do. I'm here to help you with any questions or tasks you have. How can I assist you today?
    Hello! Thank you for the warm welcome. Right now, I am here ready to assist you with any questions or tasks you may have. What can I help you with today?
    Hello! I'm just a computer program, so I don't have feelings in the same way humans do. But I'm here and ready to help with anything you need! How can I assist you today?
    Hello! I'm here to assist you with any questions or tasks you may have. How can I help you today?
    Hello! As an AI, I don't have feelings, but I'm here and ready to assist you. How can I help you today?
```

Converting original responses to embeddings. We are using BERT here:

5. Converting original Responses to Embeddings

```python
[ ]
# Convert 'original responses' to a list of strings
standard_response_list = df['Response'].tolist()

print(standard_response_list)
```

```
['Hello there. Tell me how are you feeling today?', 'Hi there. What brings you here today?', 'Hi there. How are you feeling today?', 'Great to see you. How do you feel currently?', "He
```

```python
[ ] # converting original responses to embeddings using bert
import transformers
import torch
# Load the BERT tokenizer and model
tokenizer = transformers.AutoTokenizer.from_pretrained("bert-base-uncased")
model = transformers.AutoModel.from_pretrained("bert-base-uncased")

# Encode the text in the responses using BERT
standard_encoded_responses = tokenizer(standard_response_list, padding=True, truncation=True, return_tensors="pt")

# Get the BERT embeddings for the encoded text
with torch.no_grad():
    model_output = model(**standard_encoded_responses)
    standard_embeddings = model_output.pooler_output

# Print the shape of the embeddings
print(standard_embeddings)
```

```
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 3.28kB/s]
config.json: 100%  570/570 [00:00<00:00, 46.9kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 4.98MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 26.0MB/s]
model.safetensors: 100%  440M/440M [00:02<00:00, 210MB/s]
tensor([[-0.9050, -0.3170, -0.7775,  ..., -0.5701, -0.6867,  0.8886],
        [-0.9451, -0.4878, -0.9316,  ..., -0.8420, -0.7578,  0.9403],
        [-0.9454, -0.4603, -0.9175,  ..., -0.7808, -0.7406,  0.9266],
        ...,
        [-0.6793, -0.5712, -0.9712,  ..., -0.9615, -0.4867,  0.3869],
        [-0.6200, -0.6038, -0.9815,  ..., -0.9313, -0.6690,  0.3167],
        [-0.6200, -0.6038, -0.9815,  ..., -0.9313, -0.6690,  0.3167]])
```

import transformers

The line import transformers imports the entire transformers library into our Python script or environment. Transformers is an open-source library by Hugging Face, offering a large collection of pre-trained models in natural language processing (NLP), including those of a transformer type, such as BERT, GPT, RoBERTa, and many others. Here we are using BERT.

import torch

This imports the “PyTorch” library into the Python environment. "PyTorch" is one of the most popular and powerful open-source deep learning frameworks developed by Facebook's AI Research lab (FAIR). It provides an effective and flexible platform for building and training deep neural networks.

tokenizer = transformers.AutoTokenizer.from_pretrained("bert-base-uncased")

This line initializes the BERT tokenizer using the `AutoTokenizer` class from the `transformers` library, loading in particular the pre-trained tokenizer named "bert-base-uncased". The tokenizer is able to encode the input text into tokens in respect to converting words to tokens before feeding to the BERT model for different NLP-based tasks.

model = transformers.AutoModel.from_pretrained("bert-base-uncased")

This line of code loads BERT model using the AutoModel class from the transformers library. The model is specified by its name. It's designed to take tokenized input text and returns a list of token representations. These representations can be used for various natural language processing tasks.

Now we are converting the GPT responses to Embeddings:

6. Converting ChatGPT Responses to Embeddings

```
# adding chatgpt_responses to dataframe

df['chatgpt_responses'] = chatgpt_responses
df
```

	Context	Knowledge	Response	chatgpt_responses
0	Hi	greeting	Hello there. Tell me how are you feeling today?	Hello! I'm just a computer program so I don't ...
1	Hi	greeting	Hi there. What brings you here today?	Hello! I'm here to help answer any questions y...
2	Hi	greeting	Hi there. How are you feeling today?	Hello! I'm just a computer program so I don't ...
3	Hi	greeting	Great to see you. How do you feel currently?	Thank you for asking. I'm just a computer prog...
4	Hi	greeting	Hello there. Glad to see you're back. What's g...	Hello! Thank you for checking in. Currently, I...
...	...	...	...	...
656	How do I know if I'm unwell?	fact-29	If your beliefs , thoughts , feelings or behav...	\n\nSeeking help from a mental health professi...
657	How can I maintain social connections? What if...	fact-30	A lot of people are alone right now, but we do...	Remember, it's important to take care of your...
658	What's the difference between anxiety and stress?	fact-31	Stress and anxiety are often used interchangea...	\n\nStress is a normal part of life and can e...
659	What's the difference between sadness and depr...	fact-32	Sadness is a normal reaction to a loss, disapp...	If you suspect that you or someone you know is...
660	difference between sadness and depression	fact-32	Sadness is a normal reaction to a loss, disapp...	If you or someone you know is experiencing sym...

657 rows × 4 columns

Now we will convert chatgpt responses to embeddings using Bert

```
# chatgpt responses embedding using bert

# Import the necessary libraries
import transformers
import torch
# Load the BERT tokenizer and model
tokenizer = transformers.AutoTokenizer.from_pretrained("bert-base-uncased")
model = transformers.AutoModel.from_pretrained("bert-base-uncased")

# Encode the text in the responses using BERT
chatgpt_encoded_responses = tokenizer(chatgpt_responses_list, padding=True, truncation=True, return_tensors="pt")

# Get the BERT embeddings for the encoded text
with torch.no_grad():
    model_output = model(**chatgpt_encoded_responses)
    chatgpt_embeddings = model_output.pooler_output

# Print the shape of the embeddings
print(chatgpt_embeddings)
```

```
tensor([[-0.9202, -0.5752, -0.9858,  ..., -0.9274, -0.8333,  0.9200],
        [-0.9330, -0.6038, -0.9842,  ..., -0.9477, -0.8343,  0.9154],
        [-0.9257, -0.5818, -0.9872,  ..., -0.9343, -0.8337,  0.9253],
        ...,
        [-0.7306, -0.4614, -0.9372,  ..., -0.9183, -0.5282,  0.2532],
        [-0.8799, -0.5702, -0.9902,  ..., -0.9286, -0.7631,  0.6544],
        [-0.9041, -0.6419, -0.9910,  ..., -0.9387, -0.7999,  0.7178]])
```

For the produced original and chatgpt responses embeddings we are finding the cosine similarity.

7. Cosine Similarity score between Original responses and Chatgpt responses

```
## finding similarity score between standard responses and chatgpt_responses

from sklearn.metrics.pairwise import cosine_similarity

# Calculate the cosine similarity between the two sets of embeddings
cosine_similarity_scores_1 = cosine_similarity(standard_embeddings, chatgpt_embeddings)

# Print the cosine similarity scores
print(cosine_similarity_scores_1)
```

```
[[0.95022655 0.9458135  0.9495053  ... 0.76162994 0.9209132  0.92042327]
 [0.9806452  0.9793805  0.98091155 ... 0.77022505 0.95213324 0.95432645]
 [0.9762755  0.9745792  0.97639644 ... 0.7722233  0.948799   0.9505161 ]
 ...
 [0.8369154  0.8376525  0.8352724  ... 0.9835917  0.89566034 0.89081085]
 [0.8494019  0.8503853  0.8480708  ... 0.9804302  0.90609187 0.90409875]
 [0.849402   0.8503854  0.84807086 ... 0.9804301  0.90609187 0.90409875]]
```

```
#calculating overall (average) similarity for standard and chatgpt responses
mean_cosine_similarity_1 = cosine_similarity_scores_1.mean().item()
mean_cosine_similarity_1
```

```
0.9355708360671997
```

from sklearn.metrics.pairwise import cosine_similarity

This imports the cosine_similarity function from the metrics module of the scikit-learn library. The idea of cosine similarity is essentially applied to compute the similarity measure for word embeddings in two different documents or the feature vectors of two different documents.

We are generating responses for Gemini here

8. Gemini Response Generation

```
#installing generative AI
!pip install -q -U google-generativeai
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 142.2/142.2 kB 1.6 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.5/664.5 kB 8.2 MB/s eta 0:00:00
```

```
#configuring gemini api key
import google. generativeai as genai
API_Key= 'AIzaSyCo3FbsU15Iorp-2LGfrVrG7pQZtsgwC6I'
genai.configure(api_key=API_Key)
```

pip install -q -U google-generativeai

This command installs the Python "google-generativeai" package using pip. This command adds package to bring its features into your Python environment. During the installation process, I have added the -q option so that there is no unnecessary output, while the -U option is used to force package updating with the latest version in case it is already installed.

import google. generativeai as genai

This line imports a module named generativeai from the google package and assigns it .The alias for it is genai.

We have generated gemini responses using Gemini-pro

```
# generating gemini responses using gemini-pro
generated_responses = []
for context in df['Context']:
    # Check if the context is not empty
    if context.strip():
        response = model.generate_content(context, safety_settings=safety_settings)
        generated_responses.append(response)
    else:
        continue
        # generated_responses.append("Empty context")
df['generated_response'] = generated_responses
```

```
df['generated_response']
```

```
0      response:\nGenerateContentResponse(\n    done=...
1      response:\nGenerateContentResponse(\n    done=...
2      response:\nGenerateContentResponse(\n    done=...
3      response:\nGenerateContentResponse(\n    done=...
4      response:\nGenerateContentResponse(\n    done=...
                              ...
656    response:\nGenerateContentResponse(\n    done=...
657    response:\nGenerateContentResponse(\n    done=...
658    response:\nGenerateContentResponse(\n    done=...
659    response:\nGenerateContentResponse(\n    done=...
660    response:\nGenerateContentResponse(\n    done=...
Name: generated_response, Length: 657, dtype: object
```

We have looped all the responses from the Gemini

```
#looping all the responses from gemini
for each in generated_responses:
  print(each)
```

```
response:
GenerateContentResponse(
    done=True,
    iterator=None,
    result=glm.GenerateContentResponse({'candidates': [{'content': {'parts': [{'text': 'Hi there! How can I help you today?'}], 'role': 'model'}, 'finish_reason': 1, 'index': 0, 'sa
)
response:
GenerateContentResponse(
    done=True,
    iterator=None,
    result=glm.GenerateContentResponse({'candidates': [{'content': {'parts': [{'text': 'Hello! How can I assist you today?'}], 'role': 'model'}, 'finish_reason': 1, 'index': 0, 'safe
)
response:
GenerateContentResponse(
    done=True,
    iterator=None,
    result=glm.GenerateContentResponse({'candidates': [{'content': {'parts': [{'text': 'Hello! How can I help you today?'}], 'role': 'model'}, 'finish_reason': 1, 'index': 0, 'safety
)
response:
GenerateContentResponse(
    done=True,
    iterator=None,
    result=glm.GenerateContentResponse({'candidates': [{'content': {'parts': [{'text': 'Hello there! How can I assist you today?'}], 'role': 'model'}, 'finish_reason': 1, 'index': 0
)
response:
GenerateContentResponse(
    done=True,
    iterator=None,
    result=glm.GenerateContentResponse({'candidates': [{'content': {'parts': [{'text': 'Hello there! How can I assist you today?'}], 'role': 'model'}, 'finish_reason': 1, 'index': 0
```

So finally the gemini responses are as the below:

```
[ ] gemini_responses

['Hi there! How can I help you today?',
 'Hello! How can I assist you today?',
 'Hello! How can I help you today?',
 'Hello there! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Hello! How can I assist you today?',
 'Hey there! How can I help you today?',
 'Hello! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Greetings! How can I assist you today?',
 'Yes, I am the AI assistant, here to help.',
 'Yes, I am here. Is there anything I can help you with?',
 'Yes, I am here. I am Gemini, a multi-modal AI model, developed by Google.',
 'Yes, I am here. How can I help you?',
 'Yes, I am here. Is there anything you need help with?',
 'Hello! How can I help you today?',
 'Hello! How can I help you today?',
 'Hello! How can I help you today?',
 'Hello! How can I assist you?',
 'Hello! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Greetings! How may I assist you today?',
 'Hello there! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Hello there! How can I help you?',
 'Heya! How can I help you today?',
 'Hey there! How can I help you today?',
 'Greetings! How may I assist you today?',
 'Greetings! How may I assist you today?',
 'Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy!
Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy!
Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy!
```

Converting Gemini responses to list

```
[ ] #converting gemini responses to list
    gemini_responses_list = df['gemini_responses'].tolist()
    gemini_responses_list

['Hi there! How can I help you today?',
 'Hello! How can I assist you today?',
 'Hello! How can I help you today?',
 'Hello there! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Hello! How can I assist you today?',
 'Hey there! How can I help you today?',
 'Hello! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Greetings! How can I assist you today?',
 'Yes, I am the AI assistant, here to help.',
 'Yes, I am here. Is there anything I can help you with?',
 'Yes, I am here. I am Gemini, a multi-modal AI model, developed by Google.',
 'Yes, I am here. How can I help you?',
 'Yes, I am here. Is there anything you need help with?',
 'Hello! How can I help you today?',
 'Hello! How can I help you today?',
 'Hello! How can I help you today?',
 'Hello! How can I assist you?',
 'Hello! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Greetings! How may I assist you today?',
 'Hello there! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Hello there! How can I assist you today?',
 'Hello there! How can I help you?',
 'Heya! How can I help you today?',
 'Hey there! How can I help you today?',
 'Greetings! How may I assist you today?',
 'Greetings! How may I assist you today?',
 'Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy! Howdy!
```

The datatype, and length of Gemini responses are generated and then adding the Gemini responses to the dataset

```
[ ] #checking datatype of gemini responses

    print(type(gemini_responses))

    <class 'list'>

[ ] #checking length of gemini response

    print(len(gemini_responses))

    657

[ ] # Adding gemini responses to dataset

    df['gemini_responses'] = gemini_responses
    df.head()
```

	Context	Knowledge	Response	chatgpt_responses	generated_response	gemini_responses
0	Hi	greeting	Hello there. Tell me how are you feeling today?	Hello! I'm just a computer program so I don't ...	response:\nGenerateContentResponse(\n done=...	Hi there! How can I help you today?
1	Hi	greeting	Hi there. What brings you here today?	Hello! I'm here to help answer any questions y...	response:\nGenerateContentResponse(\n done=...	Hello! How can I assist you today?
2	Hi	greeting	Hi there. How are you feeling today?	Hello! I'm just a computer program so I don't ...	response:\nGenerateContentResponse(\n done=...	Hello! How can I help you today?
3	Hi	greeting	Great to see you. How do you feel currently?	Thank you for asking. I'm just a computer prog...	response:\nGenerateContentResponse(\n done=...	Hello there! How can I assist you today?
4	Hi	greeting	Hello there. Glad to see you're back. What's g...	Hello! Thank you for checking in. Currently, I...	response:\nGenerateContentResponse(\n done=...	Hello there! How can I assist you today?

```
[ ] #converting gemini responses to list
    gemini_responses_list = df['gemini_responses'].tolist()
    gemini_responses_list
```

Gemini responses to embedding conversion

9. Converting Gemini responses to embeddings

```python
#embedding gemini responses
import transformers
import torch
# Load the BERT tokenizer and model
tokenizer = transformers.AutoTokenizer.from_pretrained("bert-base-uncased")
model = transformers.AutoModel.from_pretrained("bert-base-uncased")

# Encode the text in the responses using BERT
gemini_encoded_responses = tokenizer(gemini_responses_list, padding=True, truncation=True, return_tensors="pt")

# Get the BERT embeddings for the encoded text
with torch.no_grad():
    model_output = model(**gemini_encoded_responses)
    gemini_embeddings = model_output.pooler_output
```

```python
# gemini embeddings
print(gemini_embeddings)
```

```
tensor([[-0.9275, -0.4614, -0.8960,  ..., -0.8221, -0.7765,  0.9319],
        [-0.9135, -0.4201, -0.8027,  ..., -0.7391, -0.7680,  0.9198],
        [-0.9394, -0.4716, -0.8803,  ..., -0.7862, -0.7819,  0.9427],
        ...,
        [-0.6133, -0.5756, -0.9921,  ..., -0.9838, -0.6007,  0.3235],
        [-0.6307, -0.5328, -0.9723,  ..., -0.8748, -0.6059,  0.1585],
        [-0.7046, -0.7110, -0.9938,  ..., -0.9713, -0.7661,  0.4232]])
```

```python
# shape(rows and columns) of the embeddings
print(gemini_embeddings.shape)
```

```
torch.Size([657, 768])
```

Cosine similarity between original and gemini responses

10. Cosine Similarity score between original and gemini responses

```python
# finding similarity score between responses and gemini_responses

from sklearn.metrics.pairwise import cosine_similarity

# Calculate the cosine similarity between the two sets of embeddings
cosine_similarity_scores_2 = cosine_similarity(standard_embeddings, gemini_embeddings)

# Print the cosine similarity scores
print(cosine_similarity_scores_2)
```

```
[[0.9855076  0.9912153  0.9858933  ... 0.7476143  0.75175774 0.79527736]
 [0.99678826 0.99047625 0.9967525  ... 0.801755   0.7768905  0.8443443 ]
 [0.99596167 0.9907683  0.99619496 ... 0.79363465 0.77621305 0.8383176 ]
 ...
 [0.7870321  0.765682   0.77280855 ... 0.95770055 0.9852272  0.9599835 ]
 [0.7974235  0.7743323  0.78386915 ... 0.9588957  0.9925051  0.9743445 ]
 [0.7974234  0.7743324  0.78386915 ... 0.95889574 0.9925051  0.9743445 ]]
```

```python
#overall(average) similarity of gemini embeddings
mean_cosine_similarity_2 = cosine_similarity_scores_2.mean().item()
mean_cosine_similarity_2
```

```
0.8861517310142517
```

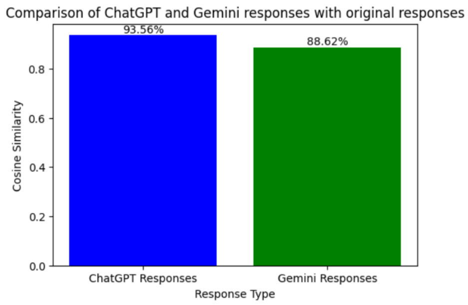
from sklearn.metrics.pairwise import cosine_similarity

First, we are importing the `cosine_similarity` function from the scikit-learn machine learning library in Python. This computes the cosine similarity between samples in a space. It computes the similarity between embeddings; hence, it has become useful in tasks that involve clustering.

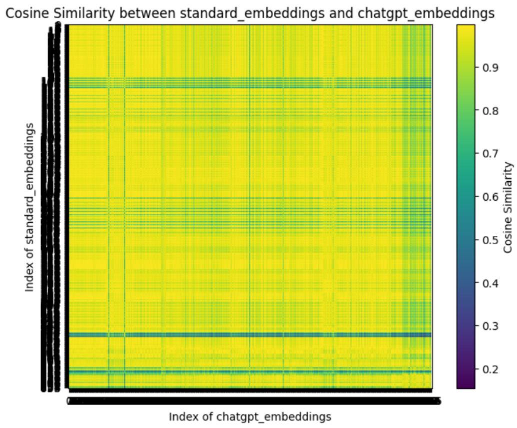
Data Visualizations for the performance of Chatgpt and Gemini

11. Data Visualization of comparing performance of ChatGPT and Gemini

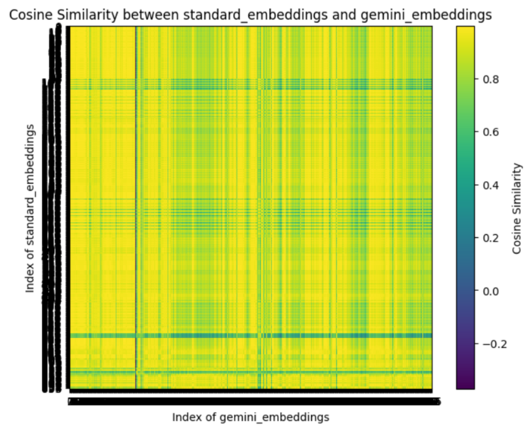
```python
# plotting graph for comparing  of chatgpt and gemini using cosine similarities

import matplotlib.pyplot as plt

# Prepare data
labels = ['ChatGPT Responses', 'Gemini Responses']
cosine_similarities = [mean_cosine_similarity_1, mean_cosine_similarity_2]

# Create bar chart
plt.figure(figsize=(6, 4))
plt.bar(labels, cosine_similarities, color=['blue', 'green'])

# Add labels and title
plt.xlabel('Response Type')
plt.ylabel('Cosine Similarity')
plt.title('Comparison of ChatGPT and Gemini responses with original responses')
# Add percentage labels to the bars
for i, v in enumerate(values):
    plt.text(i, v, str(round(v * 100, 2)) + '%', ha='center', va='bottom')

# Show plot
plt.show()
```



Finding the plot for cosine similarity between the original and chatgpt embeddings



Finding the plot for cosine similarity between the original and Gemini embeddings



Finding the optimal number of clusters and plotting scatter plot

12. Finding optimal number of clusters for Original, ChatGPT and Gemini Embeddings and plotting scatter plot

```
#finding optimal number of clusters for standard response embeddings using silhouette_scores and elbow chart

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Assuming you have an embedding matrix named 'embedding_matrix'
# Each row of 'embedding_matrix' corresponds to the embedding of a response

# Define the range of clusters you want to test
k_range = range(2, 21)  # Adjust the range as needed

# Initialize a list to store silhouette scores
silhouette_scores = []

# Iterate through each value of k
for k in k_range:
    # Initialize KMeans with the current value of k
    kmeans = KMeans(n_clusters=k, random_state=42)
    # Fit KMeans to the embedding matrix
    kmeans.fit(standard_embeddings)
    # Compute the silhouette score
    score = silhouette_score(standard_embeddings, kmeans.labels_)
    # Append the silhouette score to the list
    silhouette_scores.append(score)

# Find the value of k that maximizes the silhouette score
optimal_k = k_range[silhouette_scores.index(max(silhouette_scores))]
print("Optimal number of clusters:", optimal_k)
```

from sklearn.cluster import KMeans

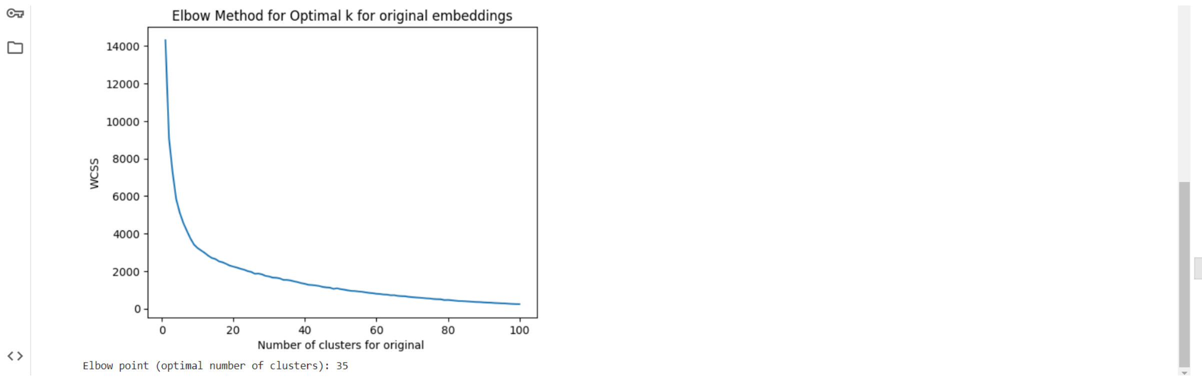
KMeans is a class imported and is an implementation of the K-means clustering algorithm provided by scikit-learn. K-means is a simple machine-learning algorithm of the unsupervised type used to cluster features into a predetermined number of clusters based on feature similarity.

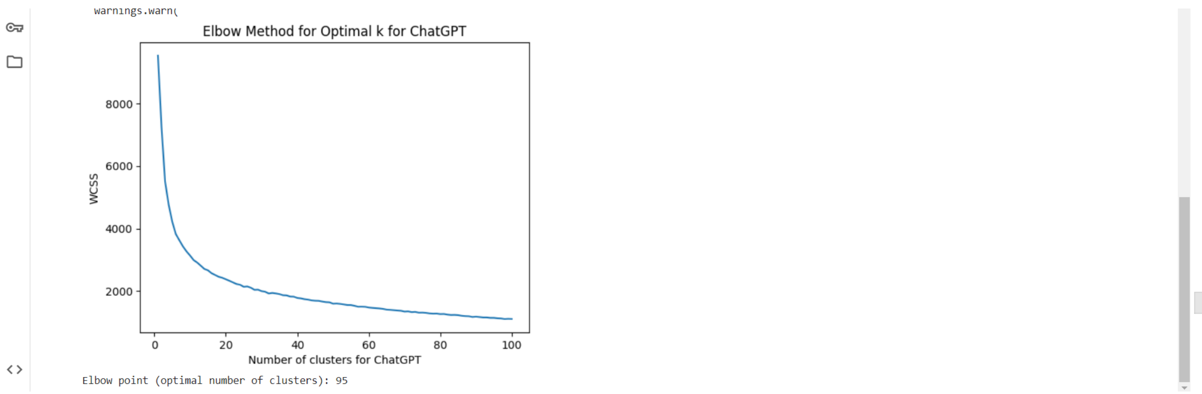
from sklearn.metrics import silhouette_score

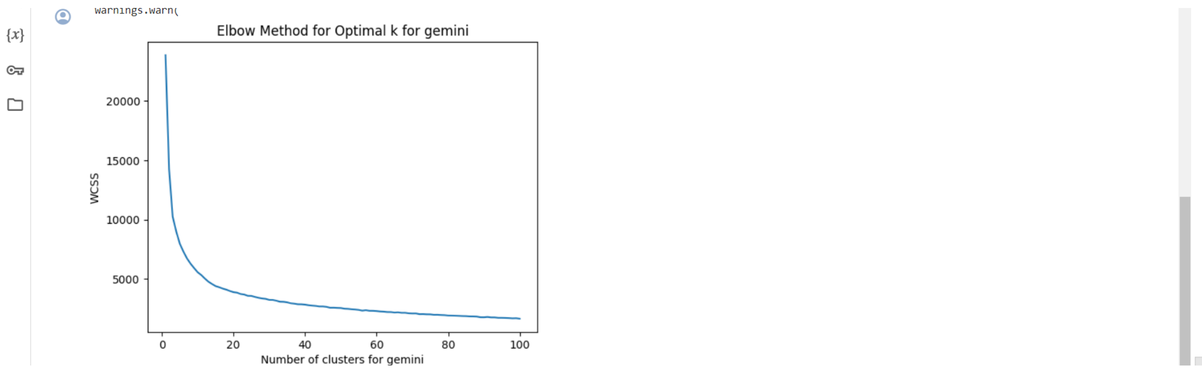
This imports the silhouette_score function which is a metric of the quality of the clustering structure. It measures how much an object is similar to its own cluster, compared to other clusters.

The silhouette value varies from [-1, 1]; a higher value indicates that an object is well-clustered and located in the correct cluster. This will mean better clusters when the silhouette score is higher.

Finding optimal number of clusters for original embeddings, chatgpt and gemini using elbow chart

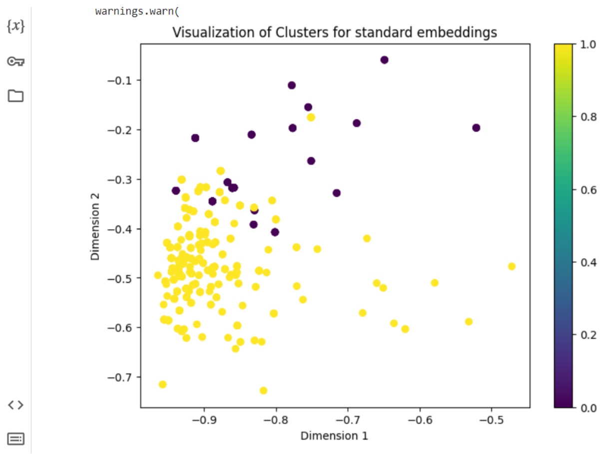




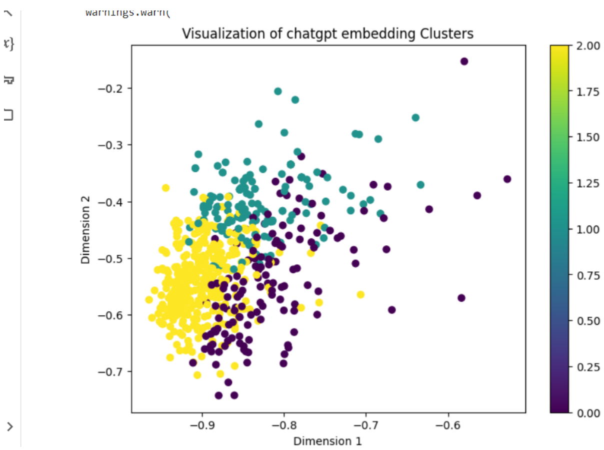


From the above elbow charts we can see that the line is getting straight at point 80 so we can take 80 as the optimal number of clusters

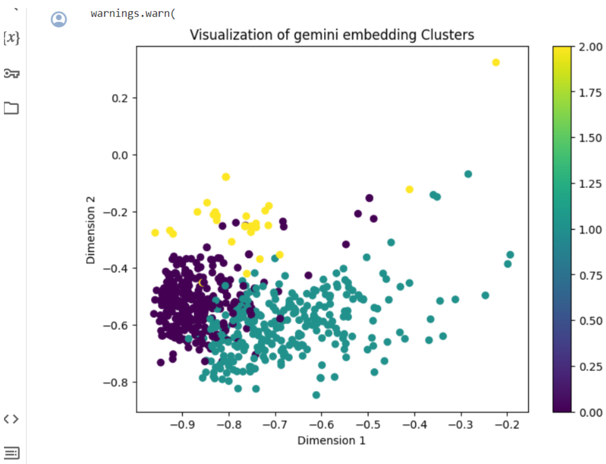
Clustering and scatterplot for standard embeddings,chatgpt and gemini



Clustering and scatterplot for chatgpt embeddings



Clustering and scatterplot for gemini embeddings



In the above plots 80 clusters and 661 data points were plotted with 80 different colors and we can see that the clusters are overlapping it means that datapoints belongs to more than one clusters.