**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.](https://d.docs.live.net/9d8ba3b24409dca1/Desktop/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



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.



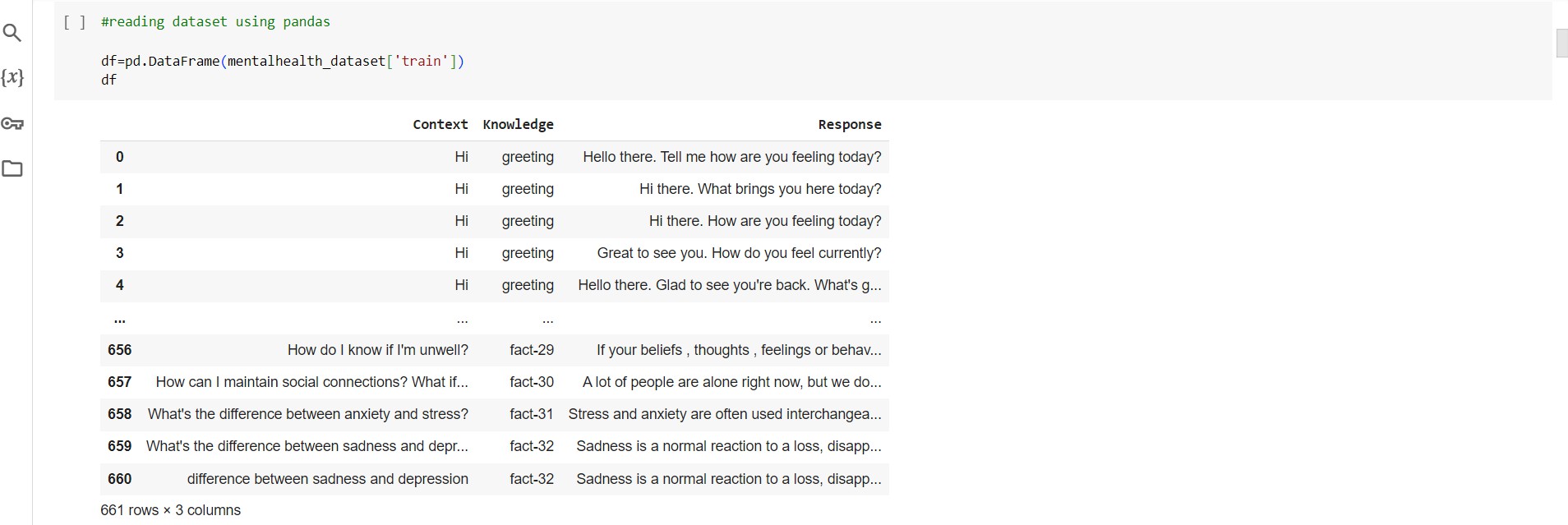
# import pandas as pd

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.

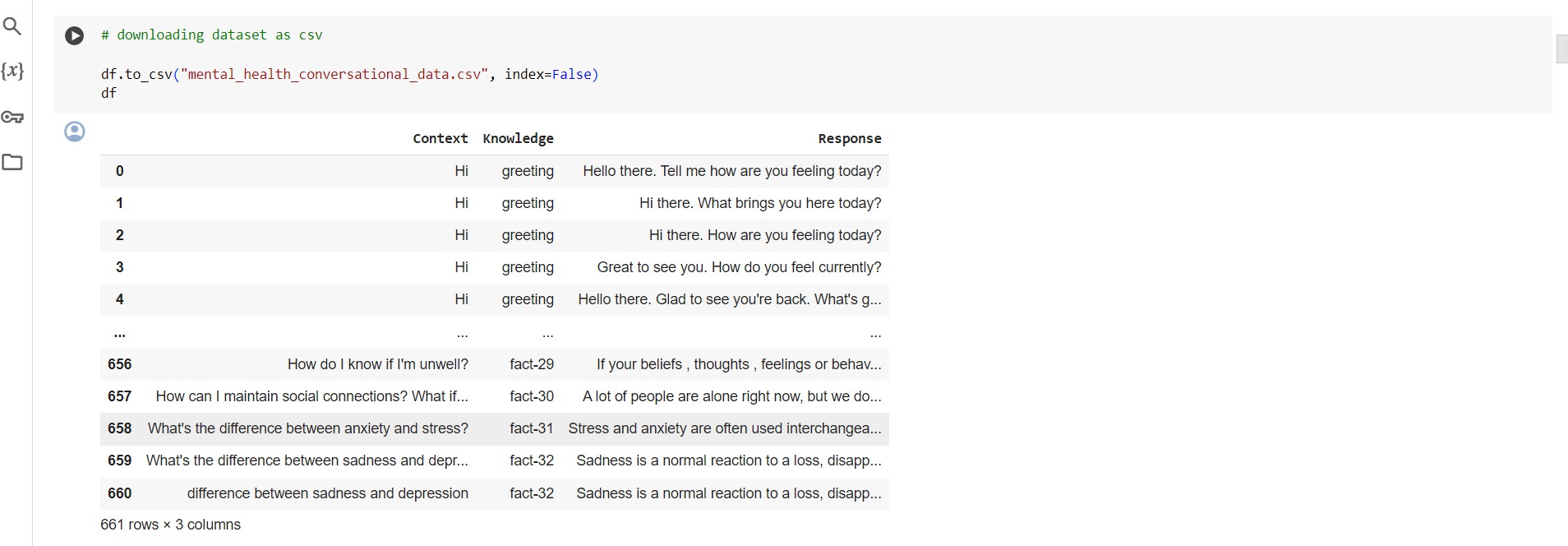
# from datasets import load\_dataset

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".

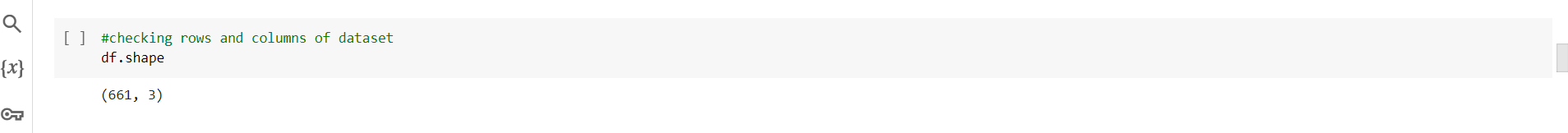
# Reading the dataset:

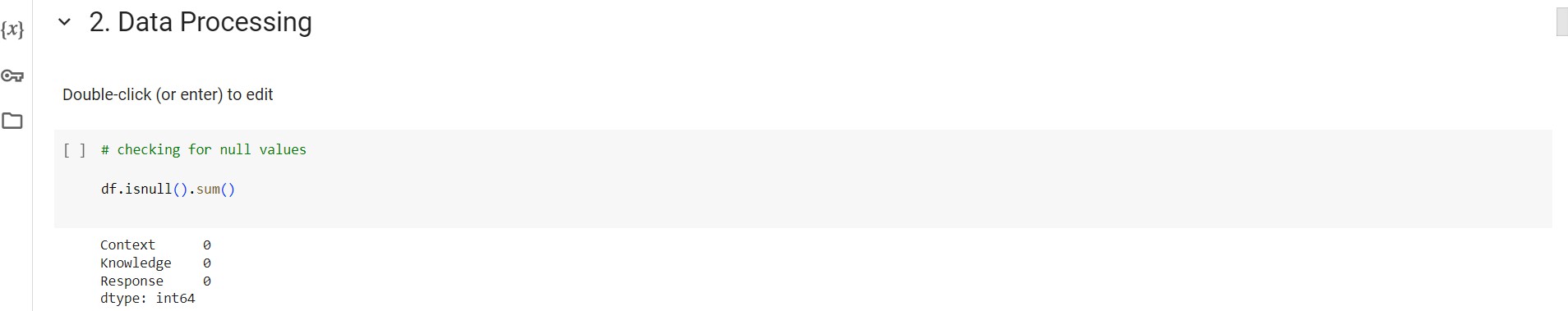


**Downloading the dataset:**



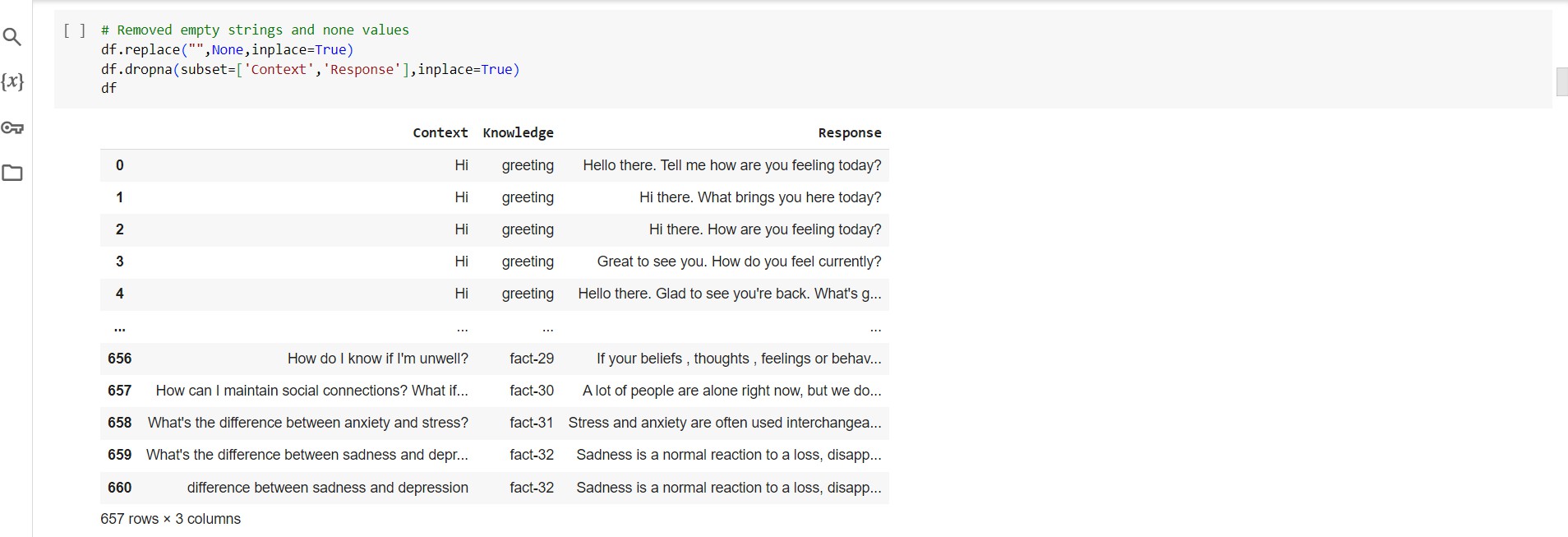
***Here the dataset has 661 rows and 3columns***





# 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.

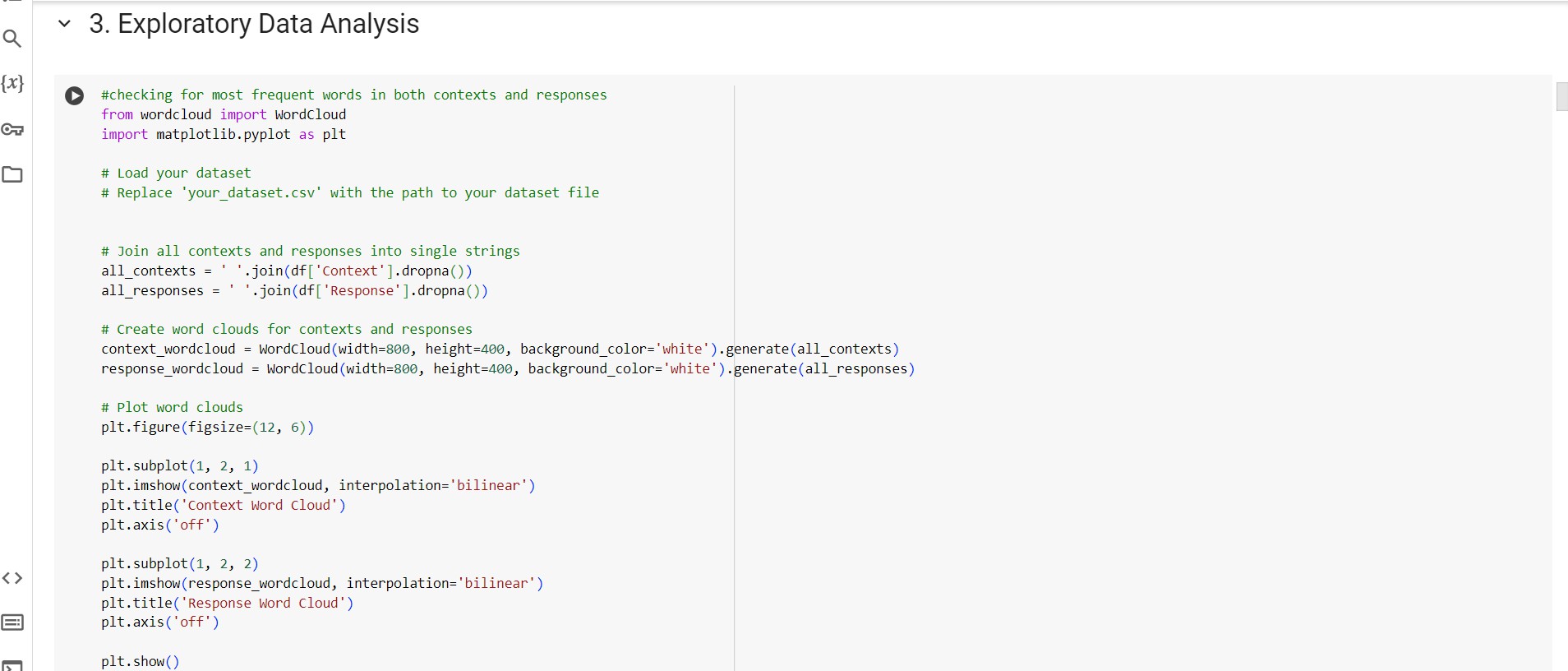


# df.replace()

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.



# 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.

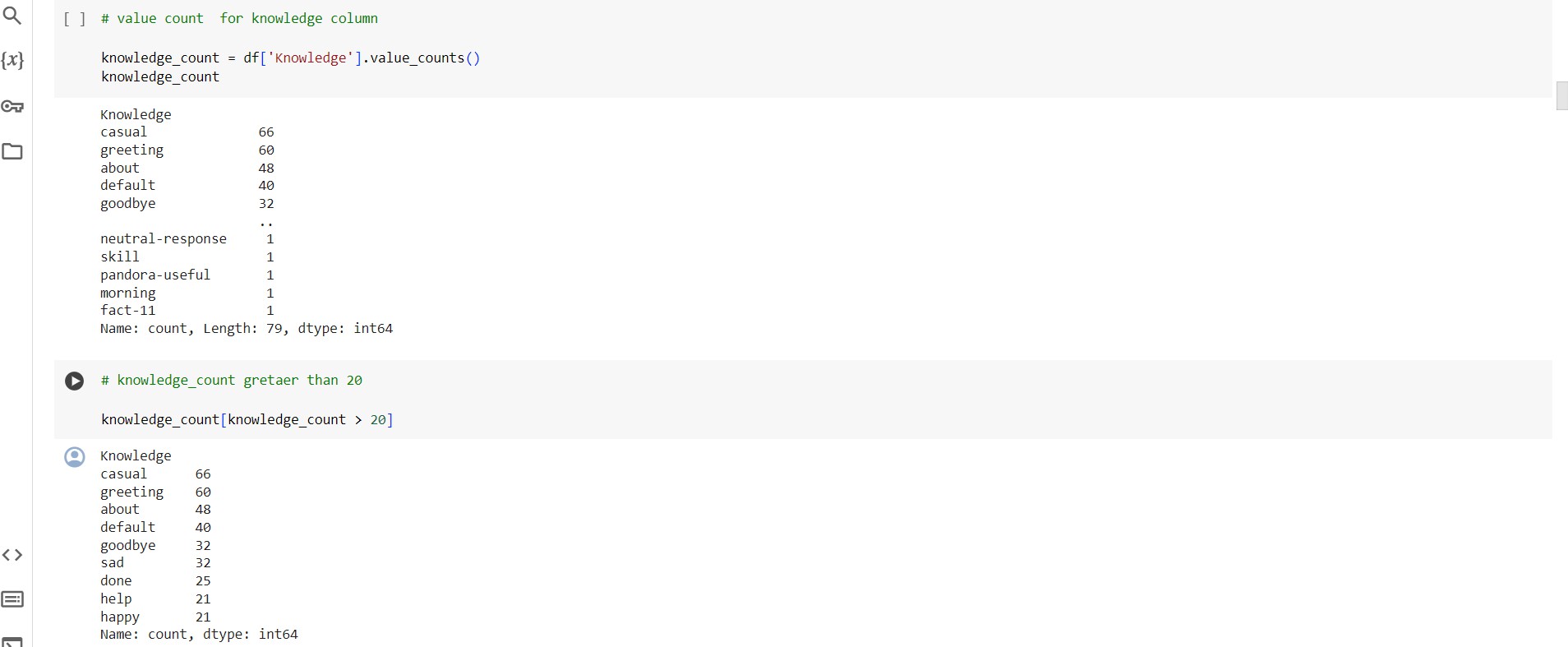
# import matplotlib.pyplot as plt

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

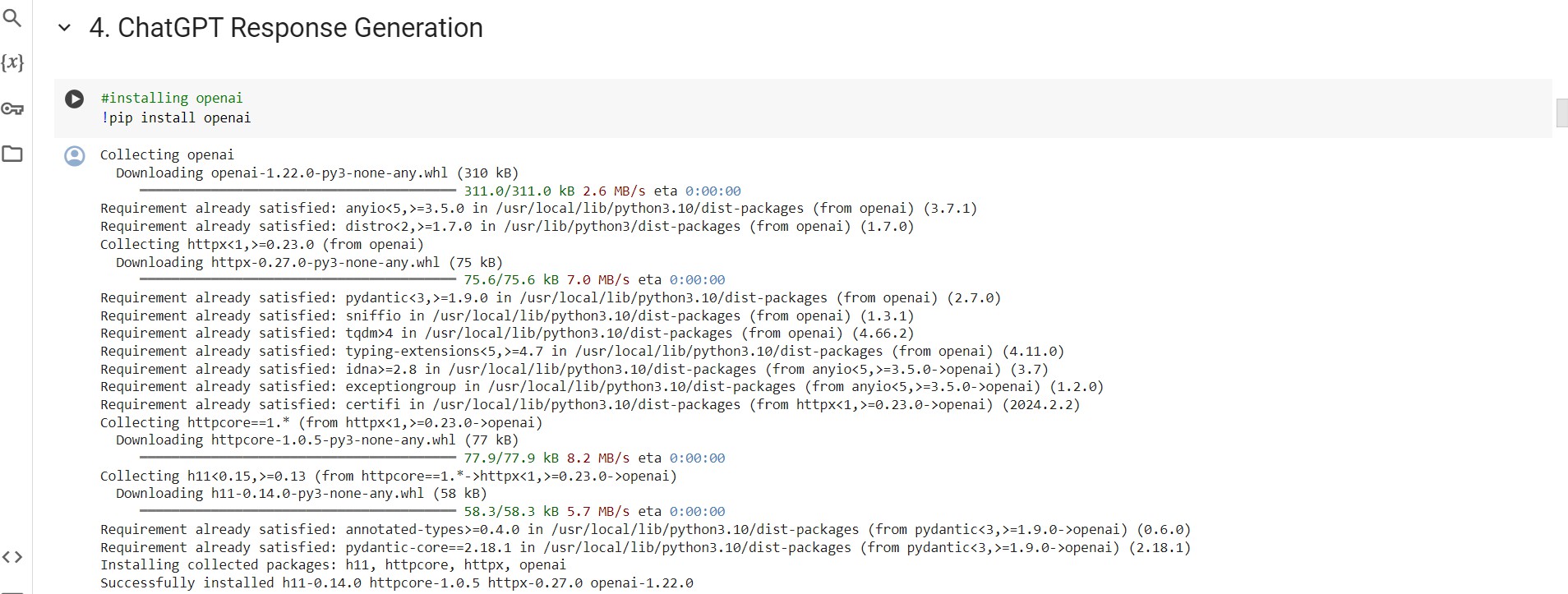
***These are the visualizations formed for context and response for most frequent words***



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



***Here is the response generated by the chatgpt***



# 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”.



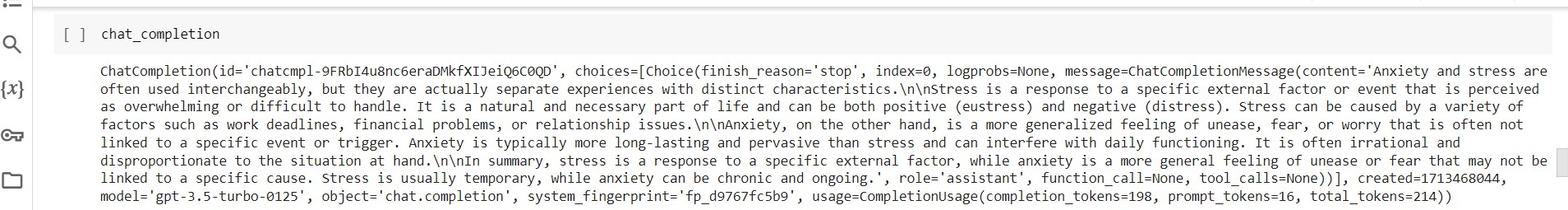
# import os

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

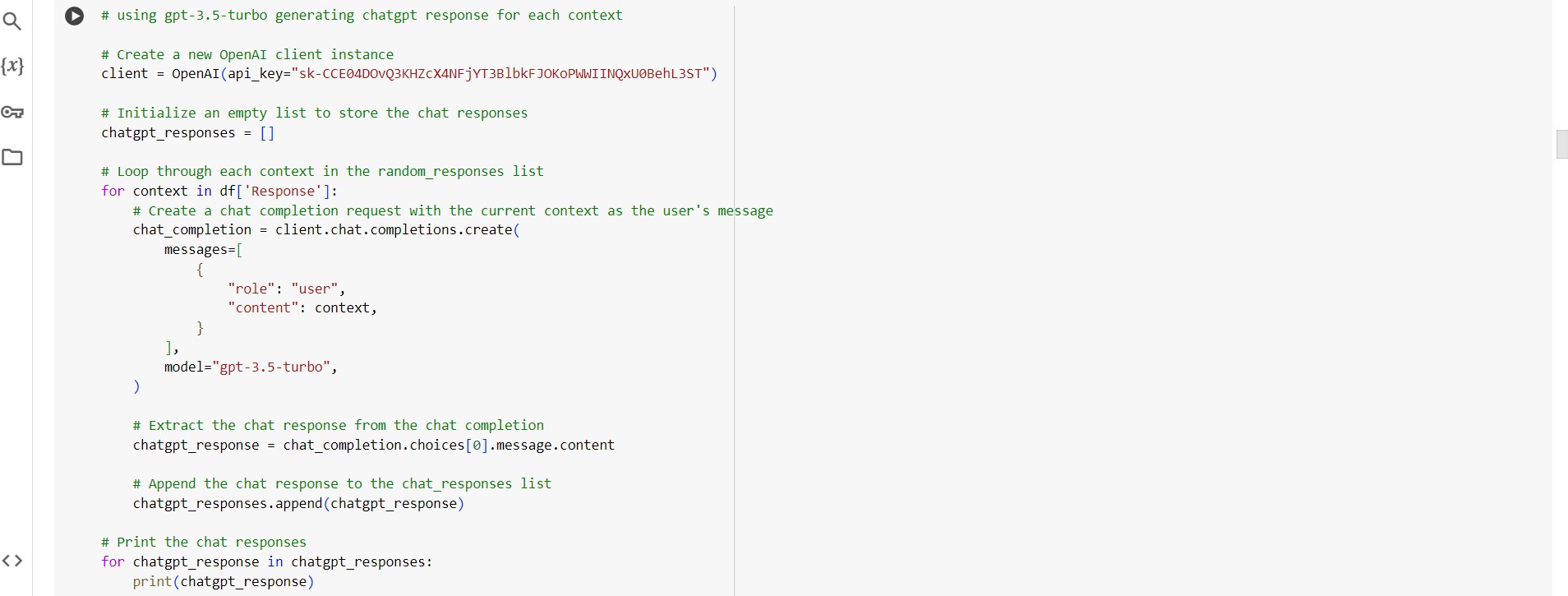
imports a class named OpenAI from a module or package named openai.

# chat completion:



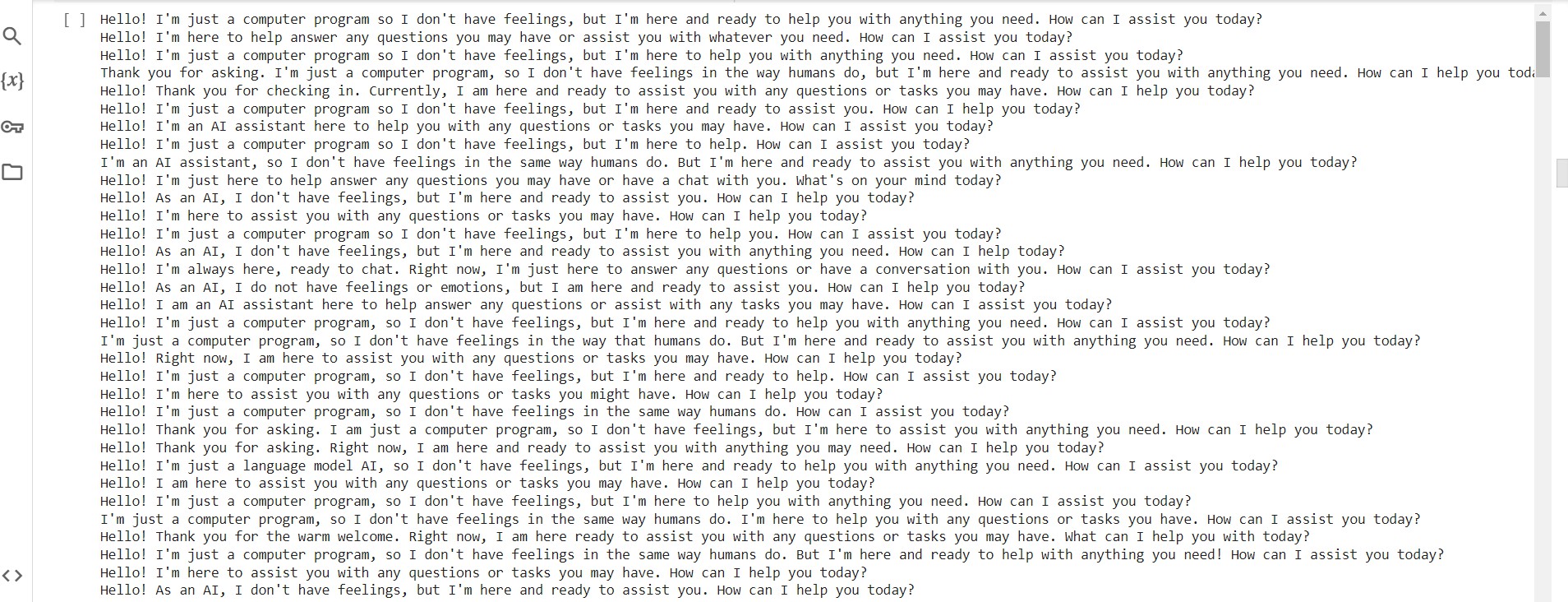
The task of generating or completing a conversational response based on an input prompt or context. This is done through natural language processing (NLP) techniques, using machine learning models, more specifically language models such as OpenAI's GPT (Generative Pre- trained Transformer) series. This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

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

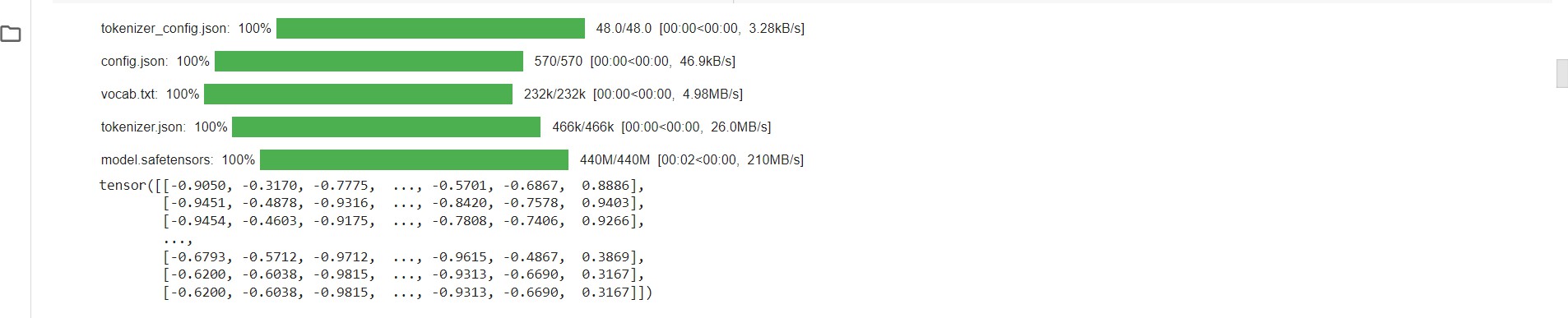


We have purchased the API key which is mentioned above to authenticate with chatgpt and we get the responses for each and every context.

***The response is as follows for each of them:***



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



# 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")

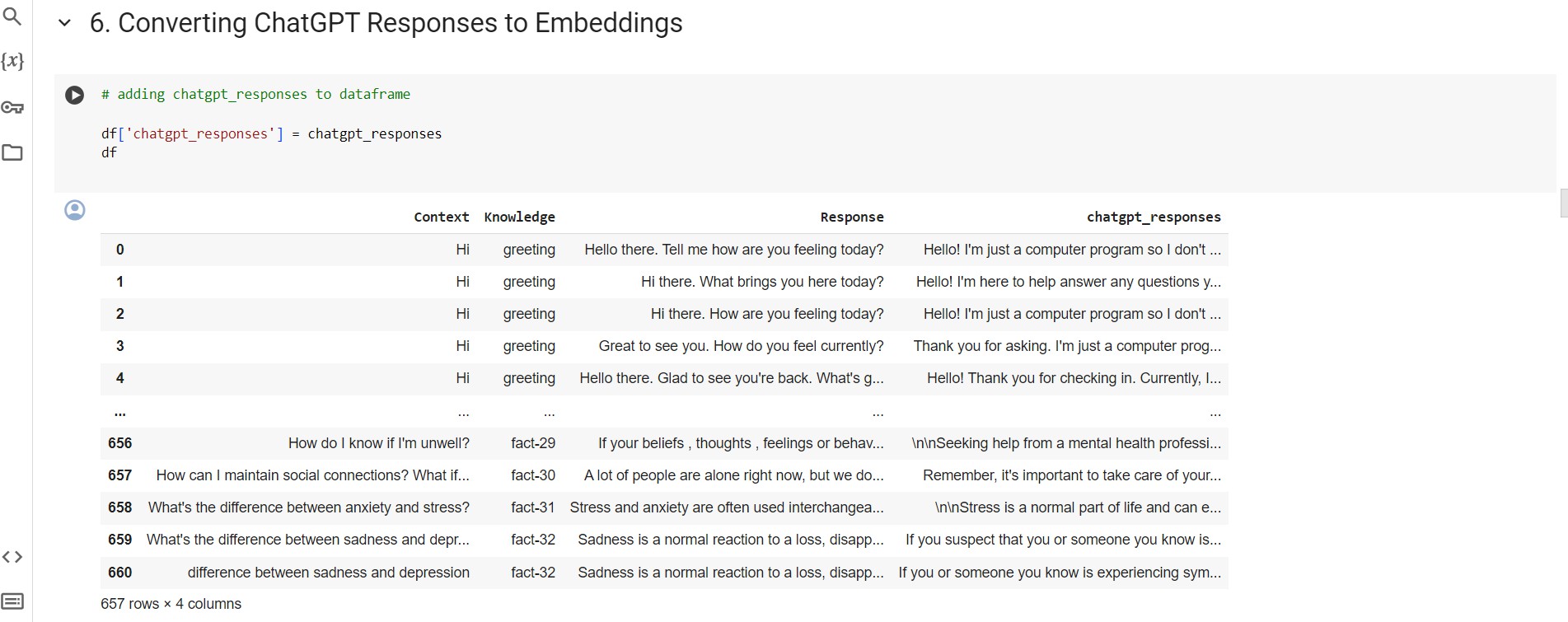
This line initializes the BERT tokenizer using the `AutoTokenizer` class from the

`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.

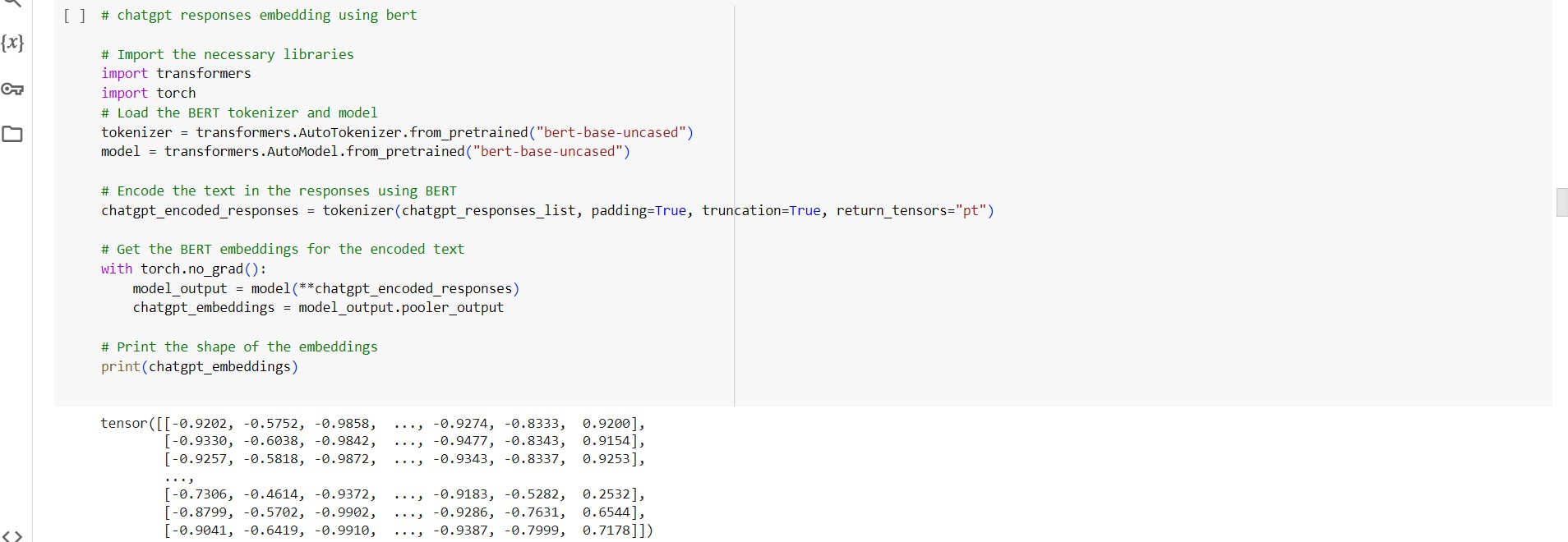
# model = transformers.AutoModel.from\_pretrained("bert-base-uncased")

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:***



***Now we will convert chatgpt responses to embeddings using Bert***



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



# 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***



# pip install -q -U google-generativeai

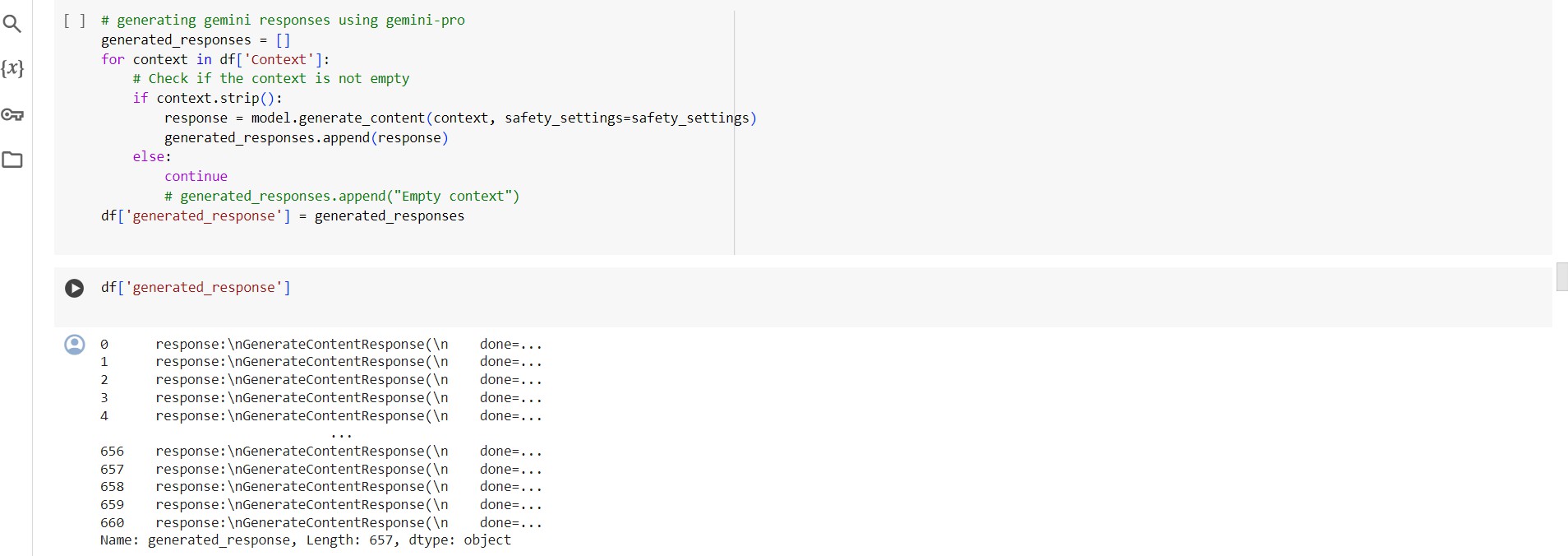
This command installs the Python "google-generativeai" package using pip. This command adds package to bring its features into your Python environment. During the installation

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***



***We have looped all the responses from the Gemini***



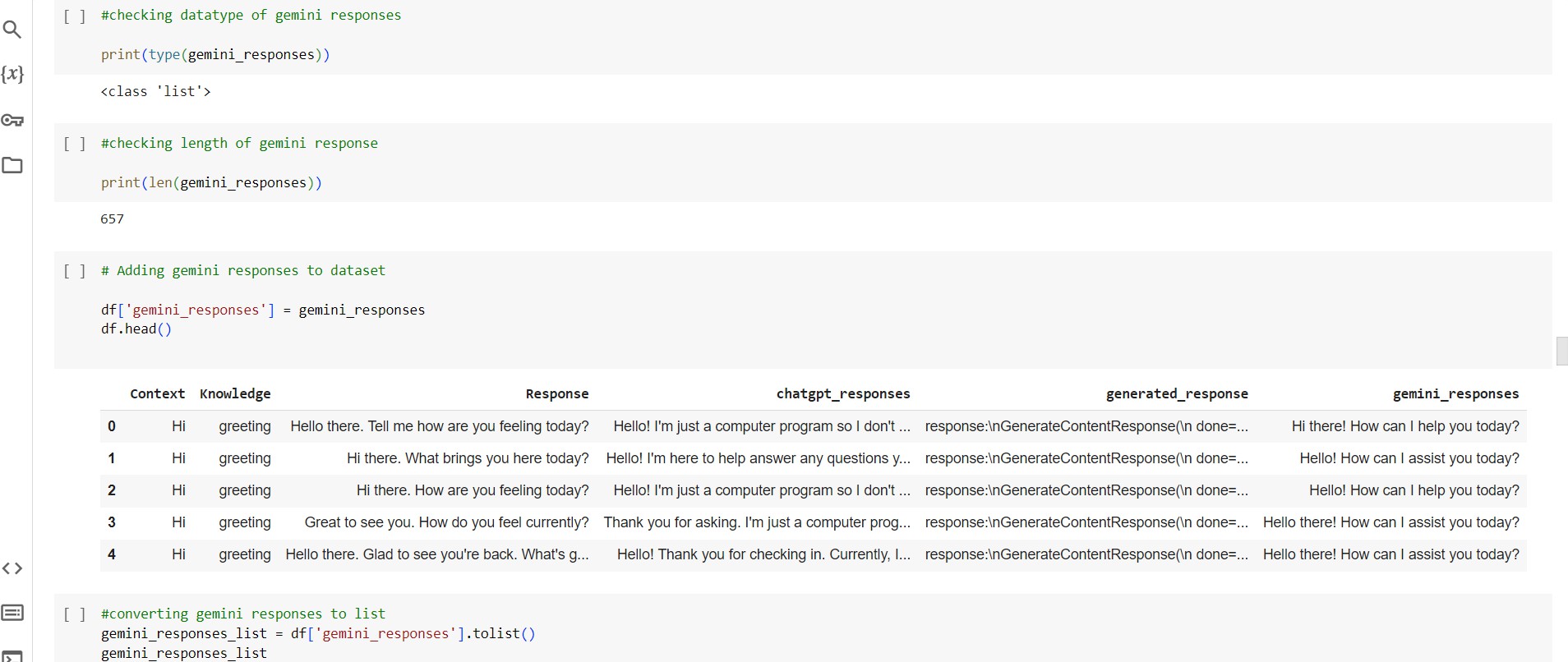
***So finally the gemini responses are as the below:***



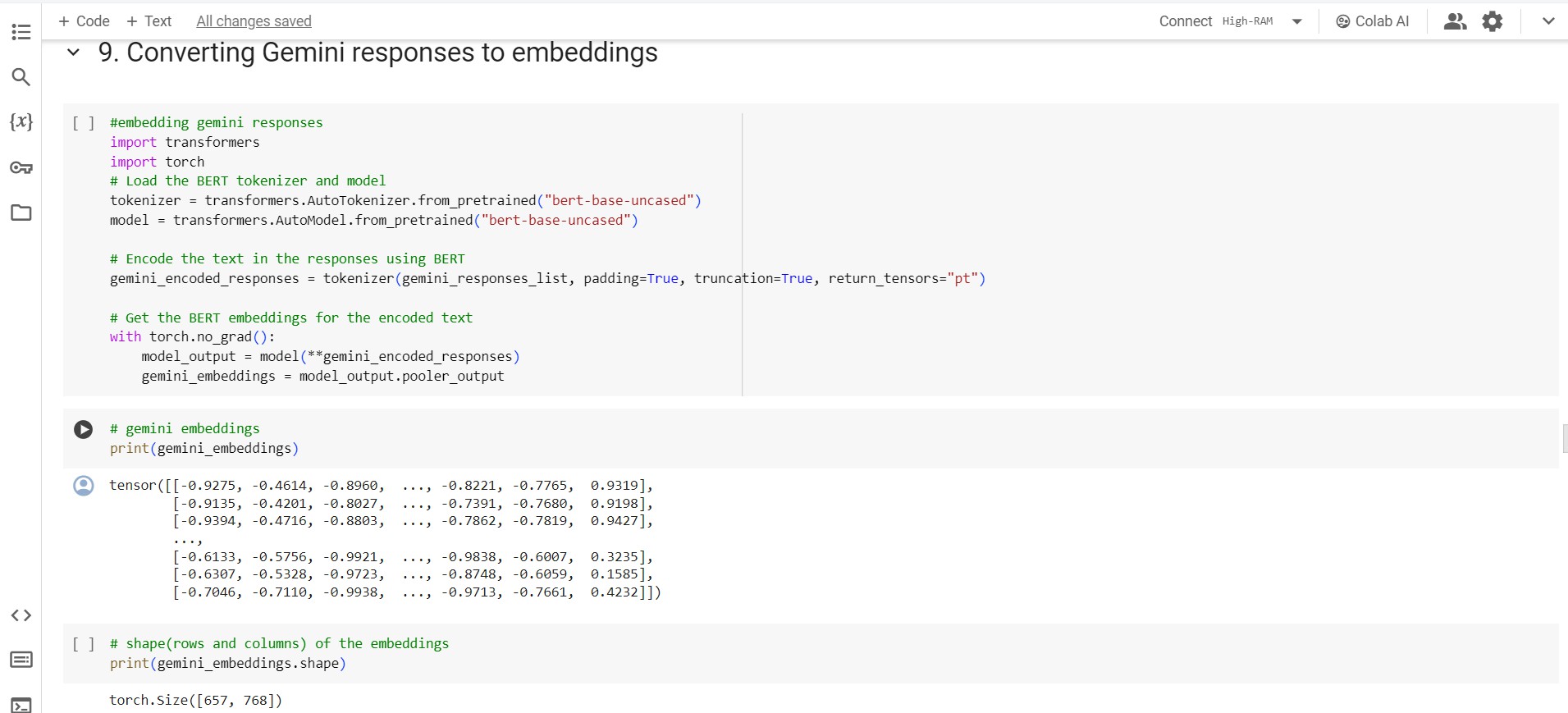
***Converting Gemini responses to list***



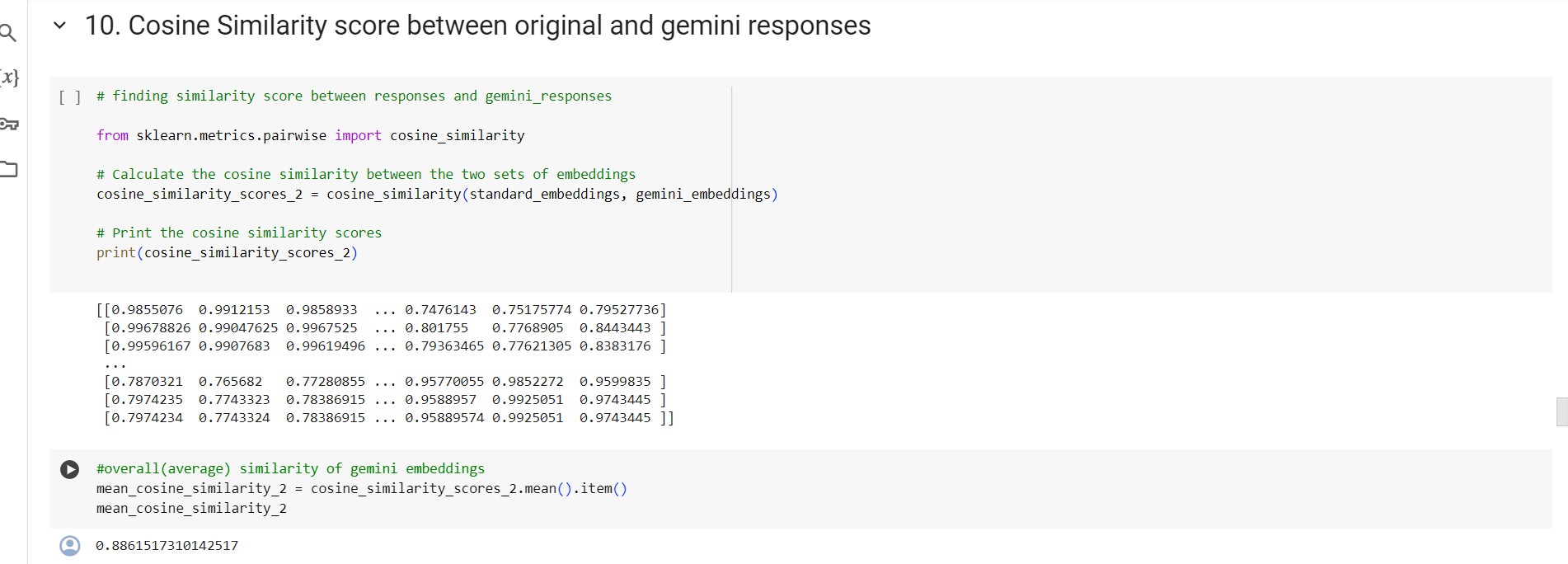
***The datatype, and length of Gemini responses are generated and then adding the Gemini responses to the dataset***



***Gemini responses to embedding conversion***



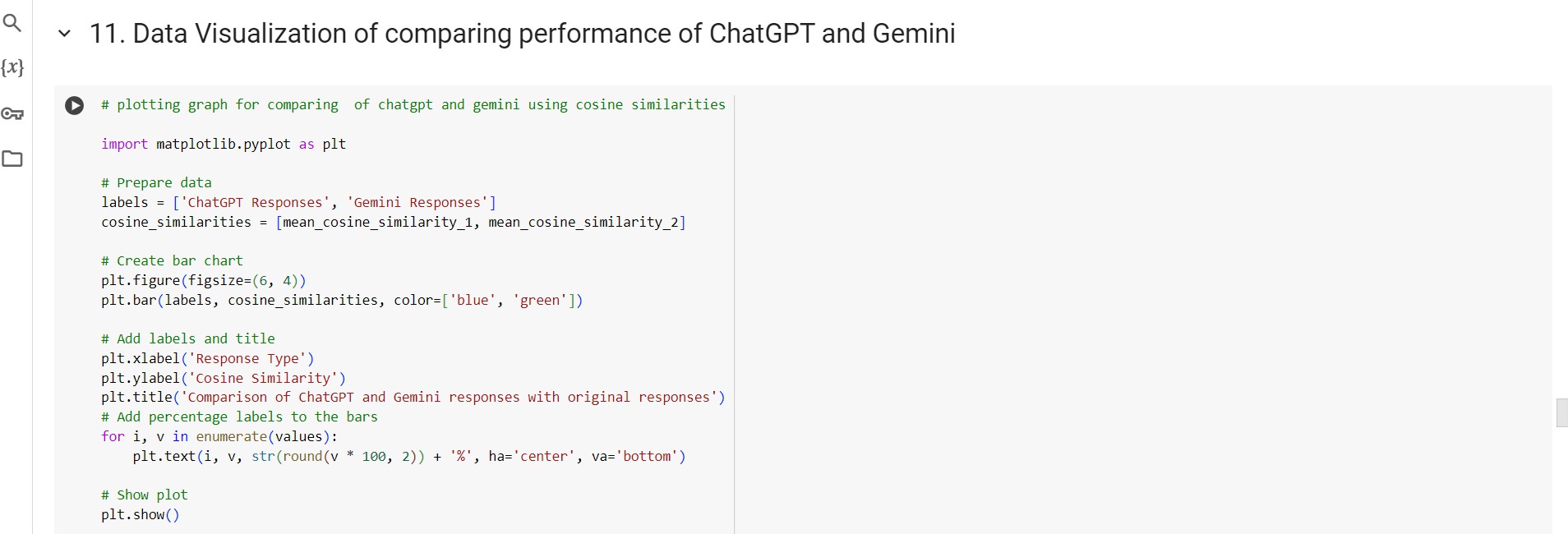
***Cosine similarity between original and gemini responses***

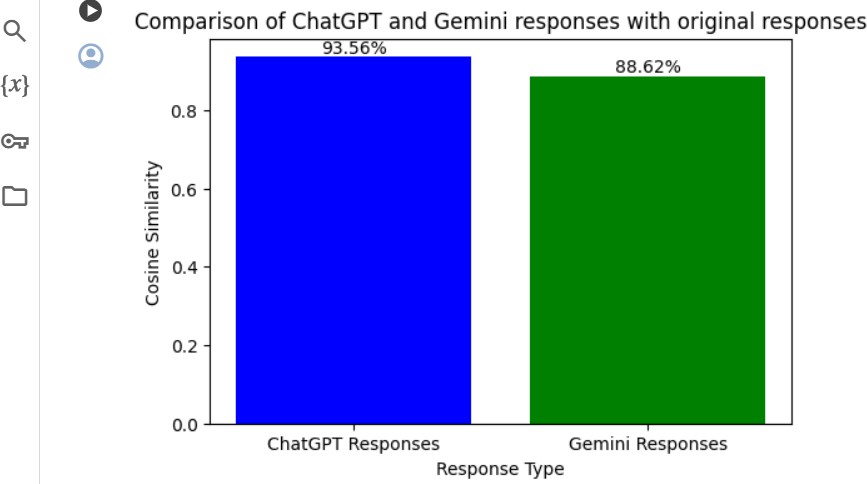


# from sklearn.metrics.pairwise import cosine\_similarity

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***

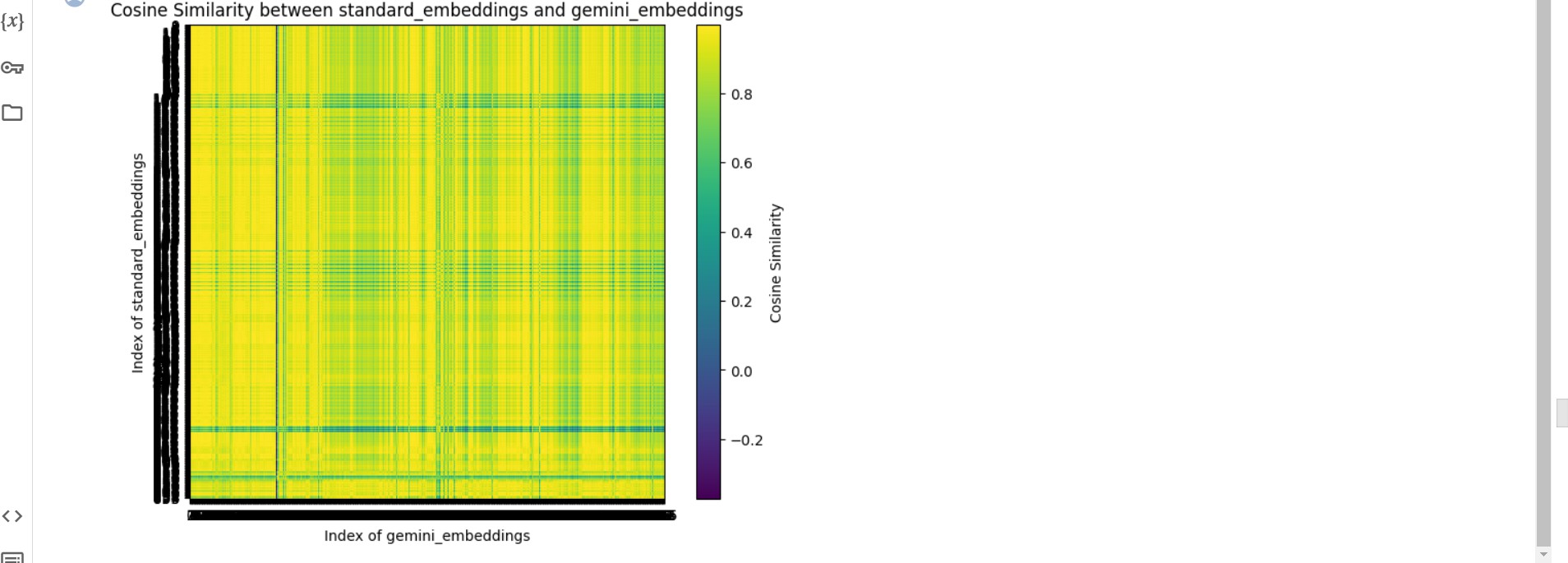




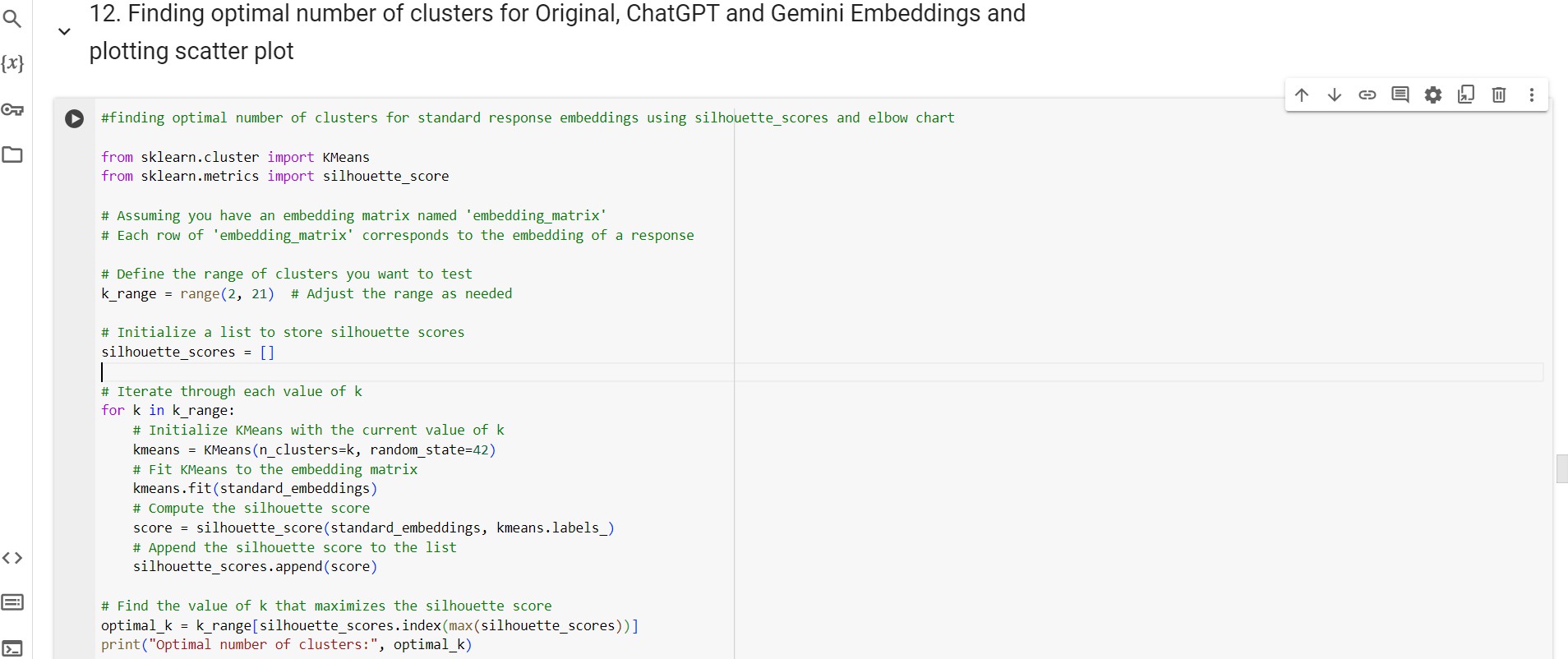
***Finding the plot for cosine similarity between the original and chatgpt embeddings***



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



***Finding the optimal number of clusters and plotting scatter plot***



# from sklearn.cluster import KMeans

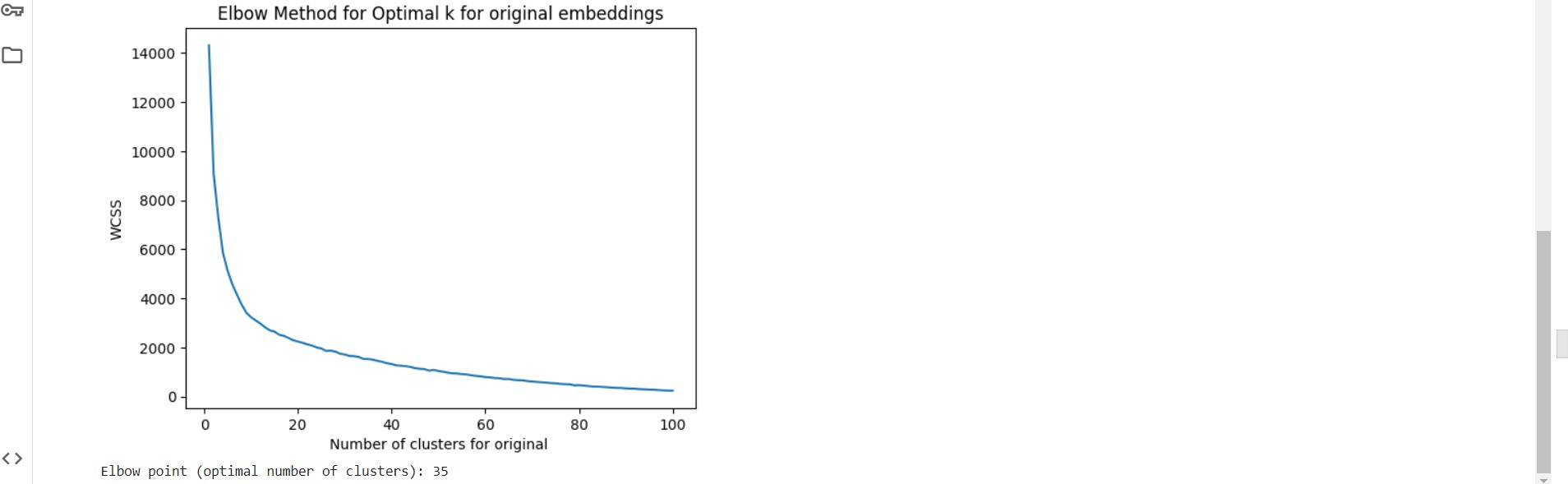
KMeans is a class imported and is an implementation of the K-means clustering algorithm provided by scikit-learn. K-means is a simple machine-learning algorithm of the unsupervised type used to cluster features into a predetermined number of clusters based on feature similarity.

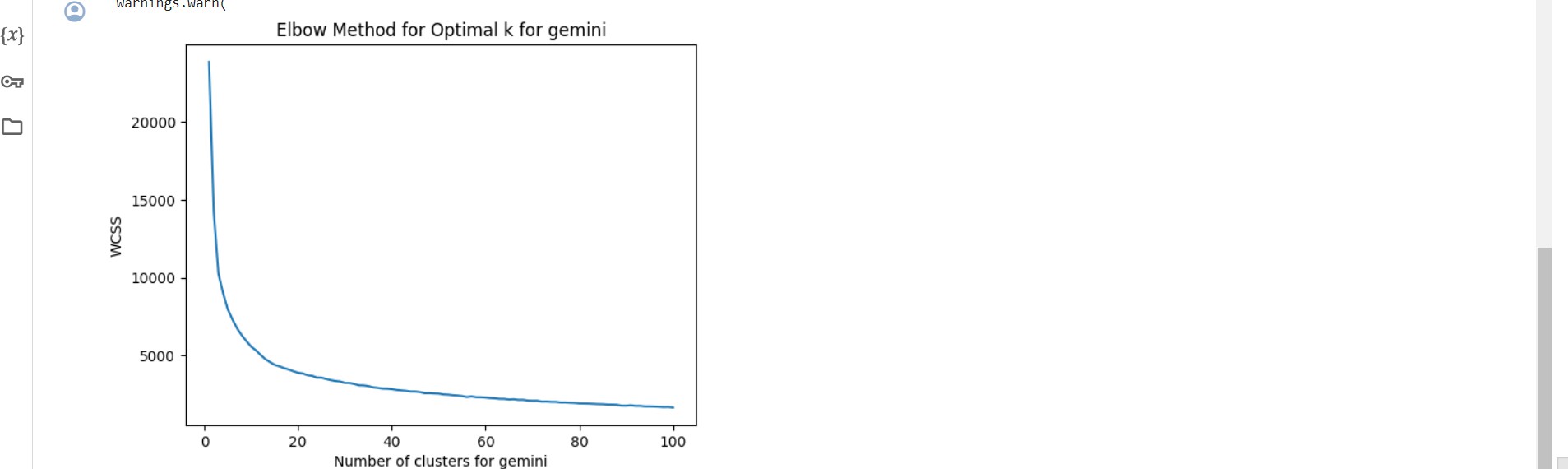
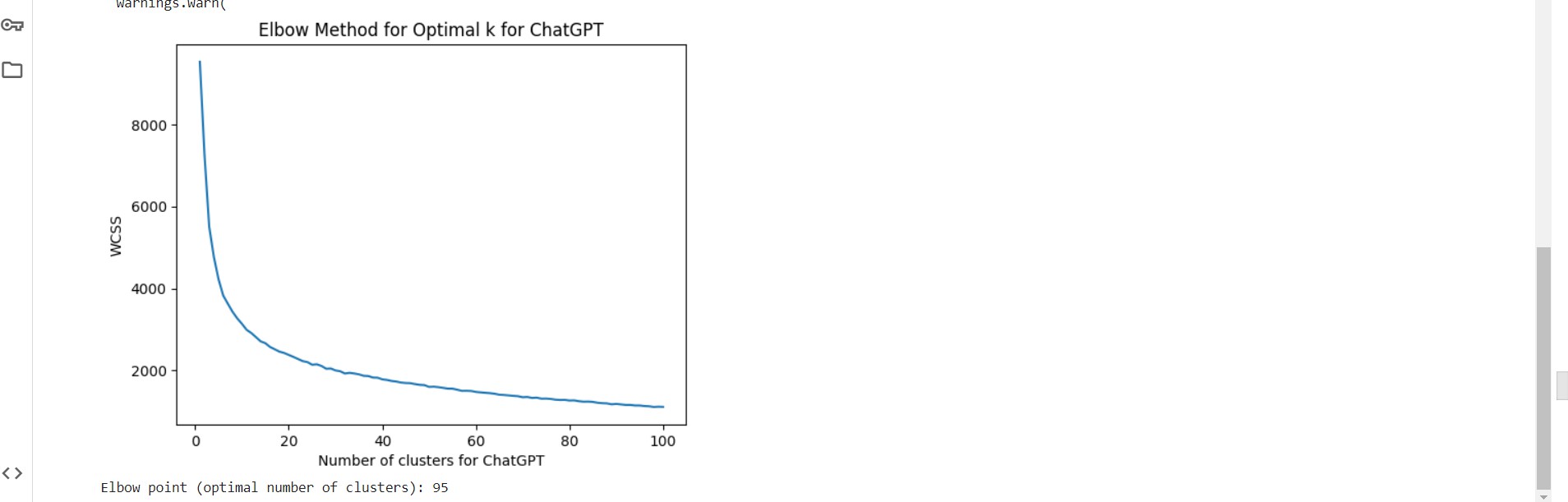
# from sklearn.metrics import silhouette\_score

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.

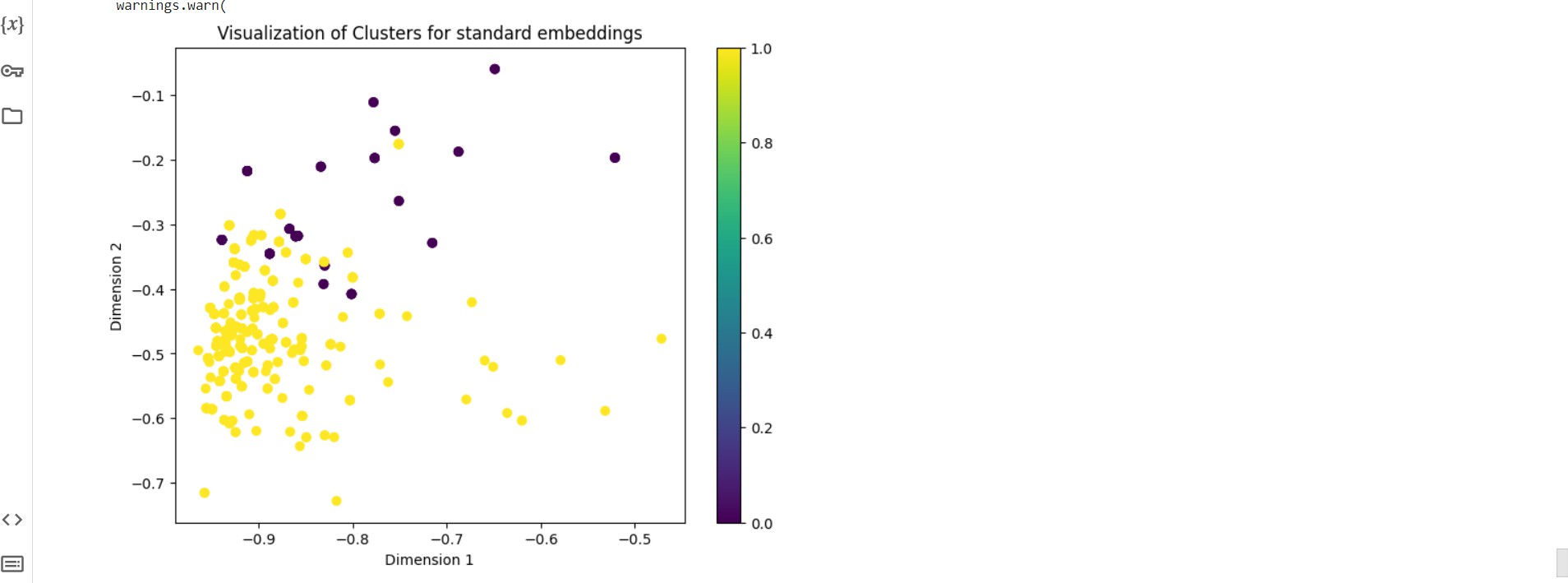
***Finding optimal number of clusters for original embeddings, chatgpt and gemini using elbow chart***



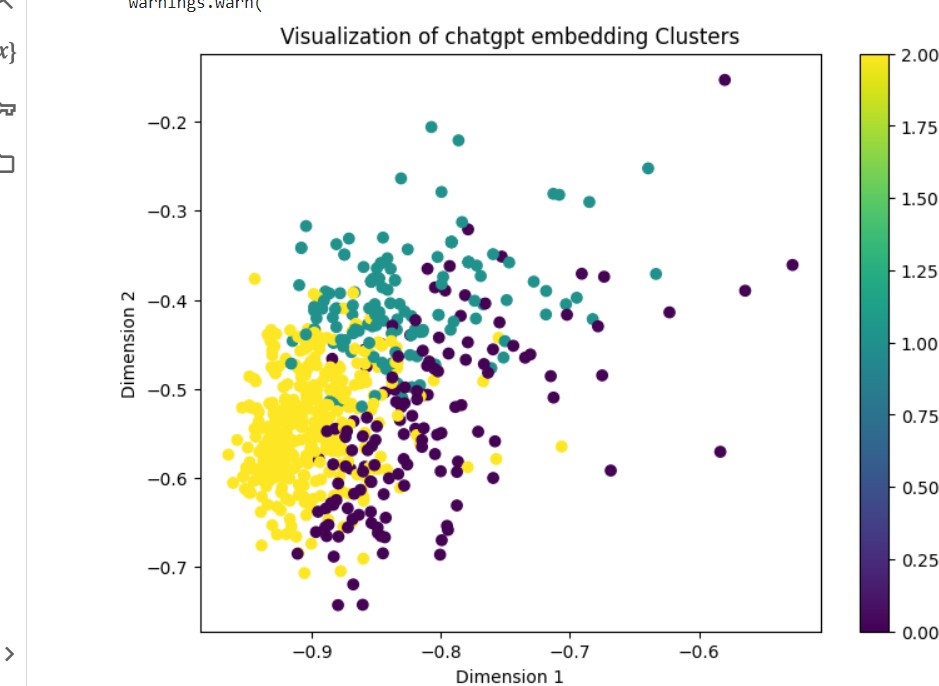


From the above elbow charts we can see that the line is getting straight at point 80 so we can take 80 as the optimal number of clusters

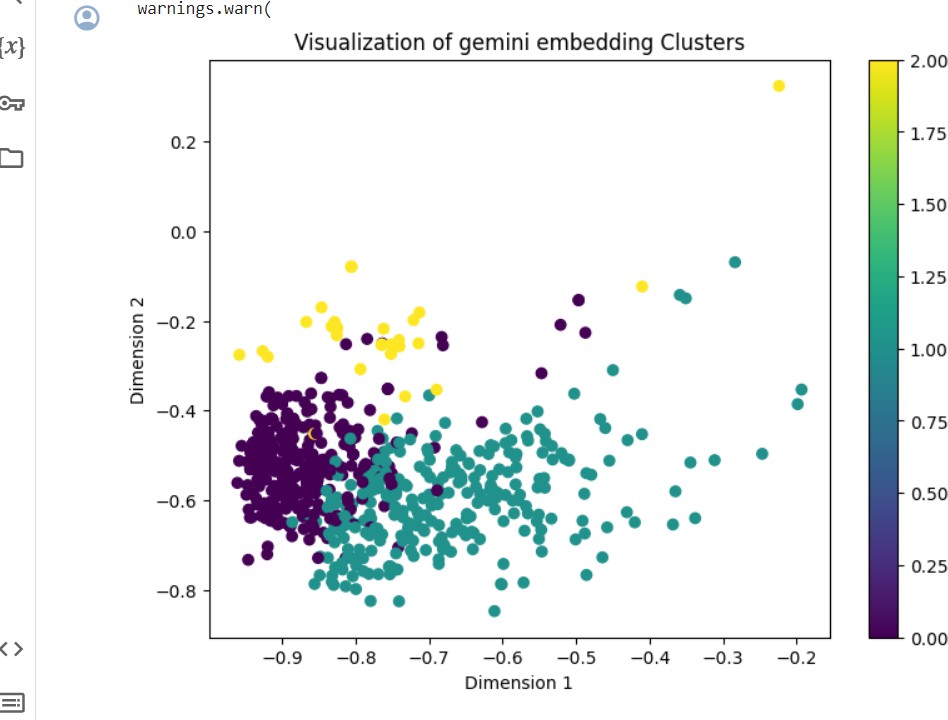
***Clustering and scatterplot for standard embeddings,chatgpt and gemini***



***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.