**AI PHASE 4**

**CREATE A CHATBOT USING PYTHON**

SYNOPSIS:

# **1. INTRODUCTION**

# **2. TECHNOLOGIES USED**

# **3. LOADING AND PRE-PROCESSING THE DATASET (PREPROCESSING, TOKENIZATION, ENCODING & DECODING ,DATA SPLIT)**

# **4. INTEGRATION WITH FLASK**

# **5. WEB INTERFACE**

# **6. HOW TO RUN AND INTERACT WITH CHATBOT**

1.Introduction:

* This document presents an overview of a Python web application that implements a chatbot using Flask, SpaCy, and the GPT-2 language model.
* The application allows users to interact with a chatbot either by querying a pre-processed dataset or generating responses using GPT-2 for unseen queries.

2.Framework Used:

* Flask: A Python web framework used for building the web application.
* SpaCy: An open-source software library for advanced natural language processing in Python. GPT-2 (Generative Pre-trained Transformer 2): A state-of-the-art language processing AI model developed by OpenAI.
* Transformers Library: A library for natural language processing using pre-trained models like GPT-2. Pandas: A data manipulation and analysis library for Python.
* HTML and CSS: For designing and structuring the web interface.

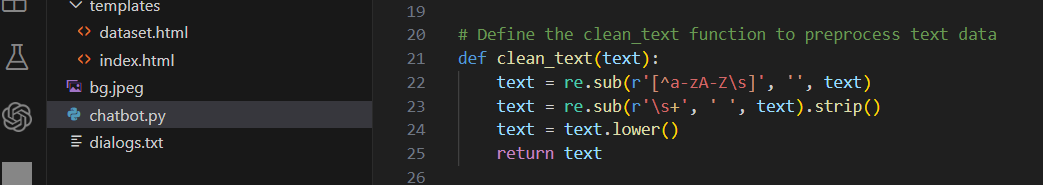
3.loading and pre-processing the dataset:

The dataset is loaded from the file **'dialogs.txt'** using Pandas. The dataset is assumed to have a tab-separated format with two columns: 'question' and 'answer'. To prepare the text data for processing, the clean text function is defined. Removes non-alphabetic characters from the text, Replaces multiple spaces with a single space and strips leading/trailing spaces, Converts the text to lowercase for uniformity.

**1.Preprocessing:**

Purpose: Preprocessing is the initial step to clean and transform raw text data into a format suitable for analysis or modelling. It involves removing unnecessary characters, converting text to lowercase, and handling other forms of noise in the data.

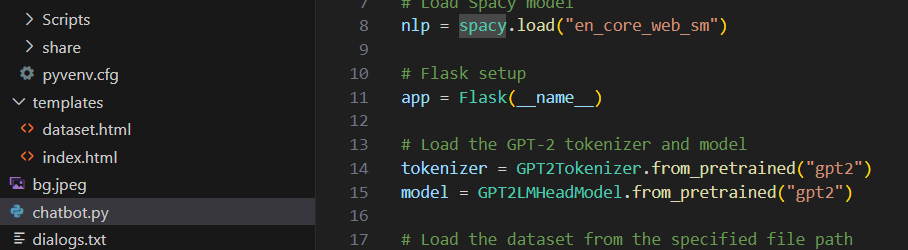
Implementation: In the provided code, the clean text function is responsible for preprocessing. It uses regular expressions to remove non-alphabetic characters, replaces multiple spaces with a single space, and converts the text to lowercase. This ensures consistency and uniformity in the textual data.



**2. Tokenization:**

Purpose: Tokenization is the process of breaking down text into smaller units, such as words or sub-words. Tokens are the building blocks for language processing tasks.

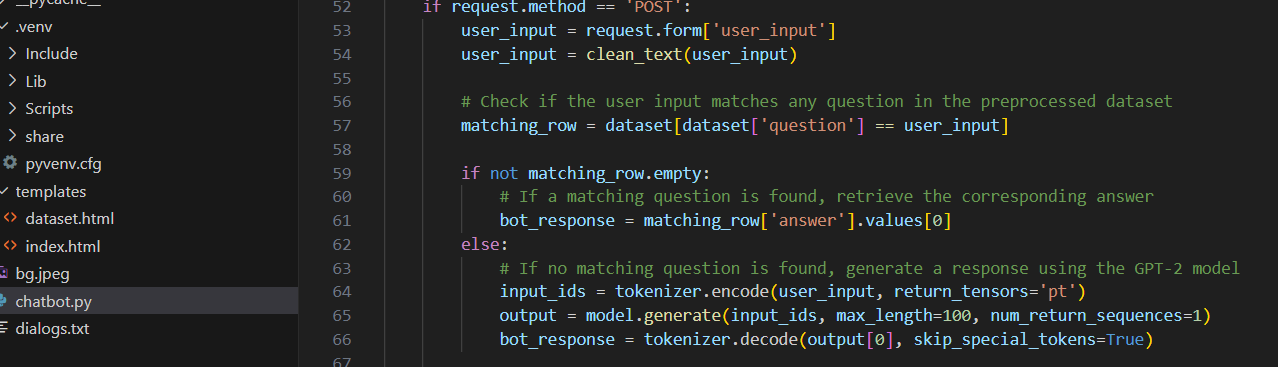
Implementation: The GPT-2 tokenizer (GPT2Tokenizer.from\_pretrained("gpt2")) is used to tokenize the user's input. Tokenization converts a sentence into a sequence of integers (token IDs) which can be fed into the neural network for processing.



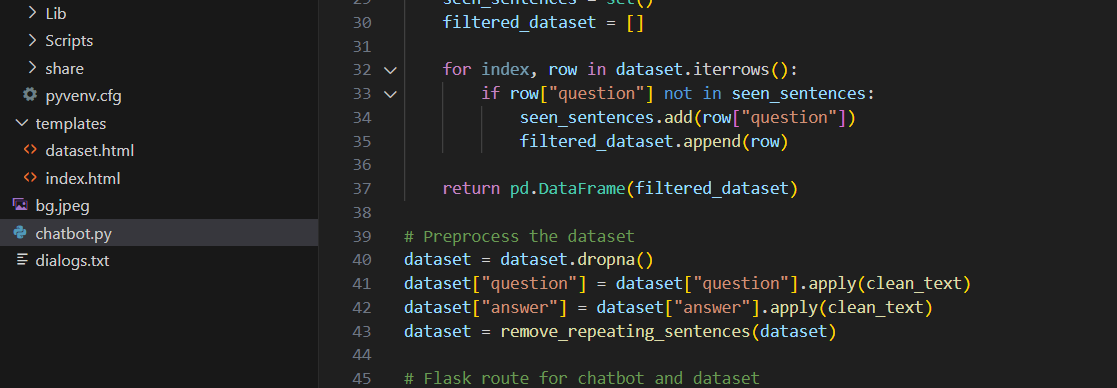
**3. Encoding & Decoding:**

Encoding refers to converting text tokens into numerical values. Neural networks process numerical data, so text data needs to be encoded into a format that the model can understand.

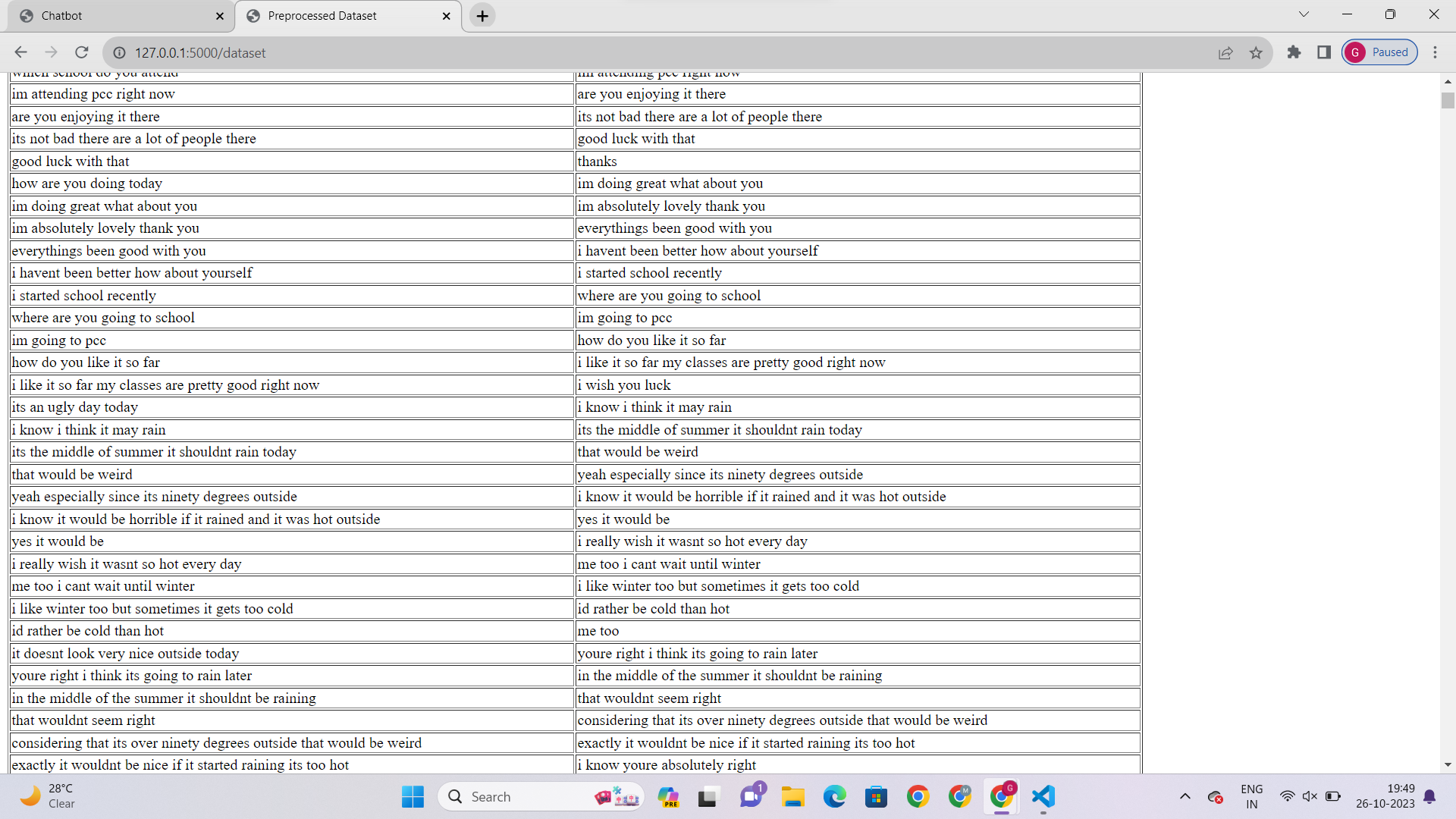
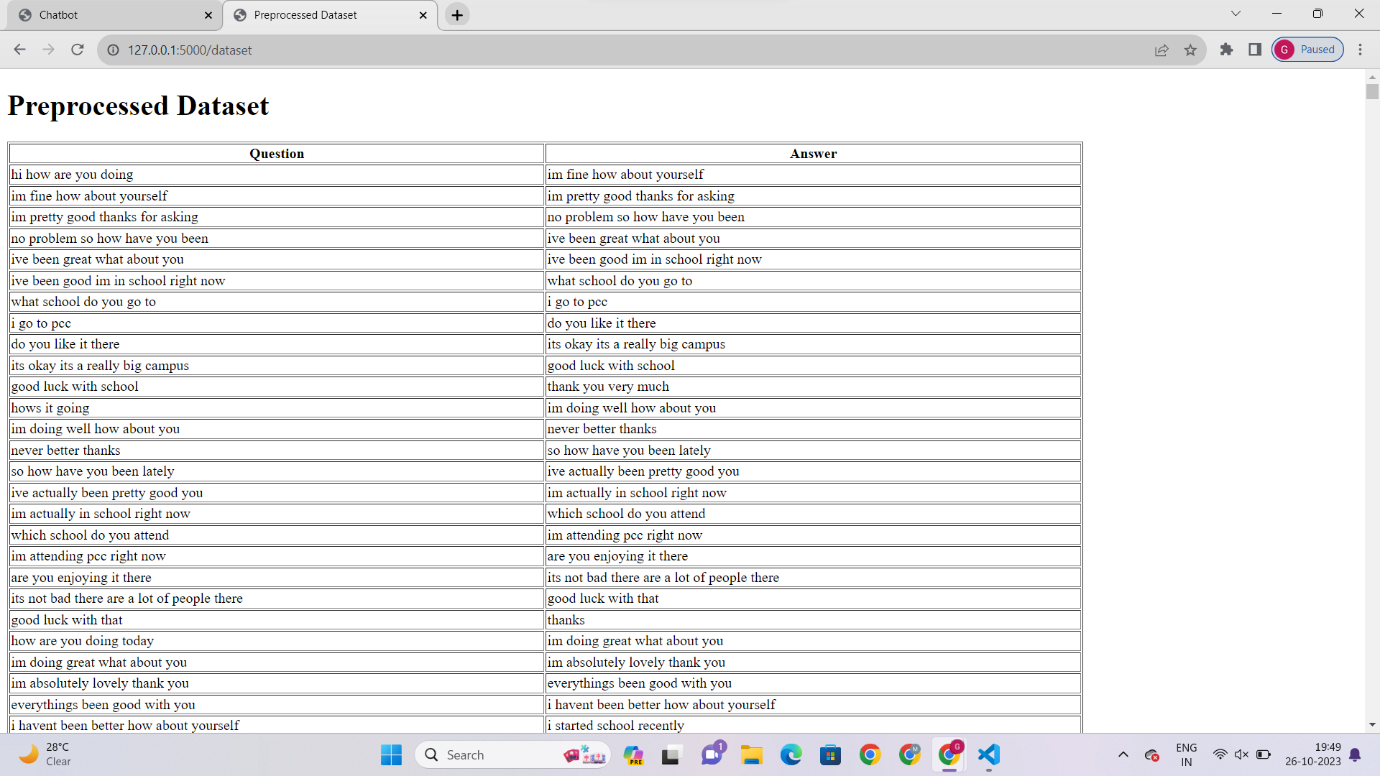
Decoding is the inverse process of encoding. It converts numerical values back into human-readable text. After the model generates predictions, the output needs to be decoded to obtain the final textual response.



**4.Data split:**

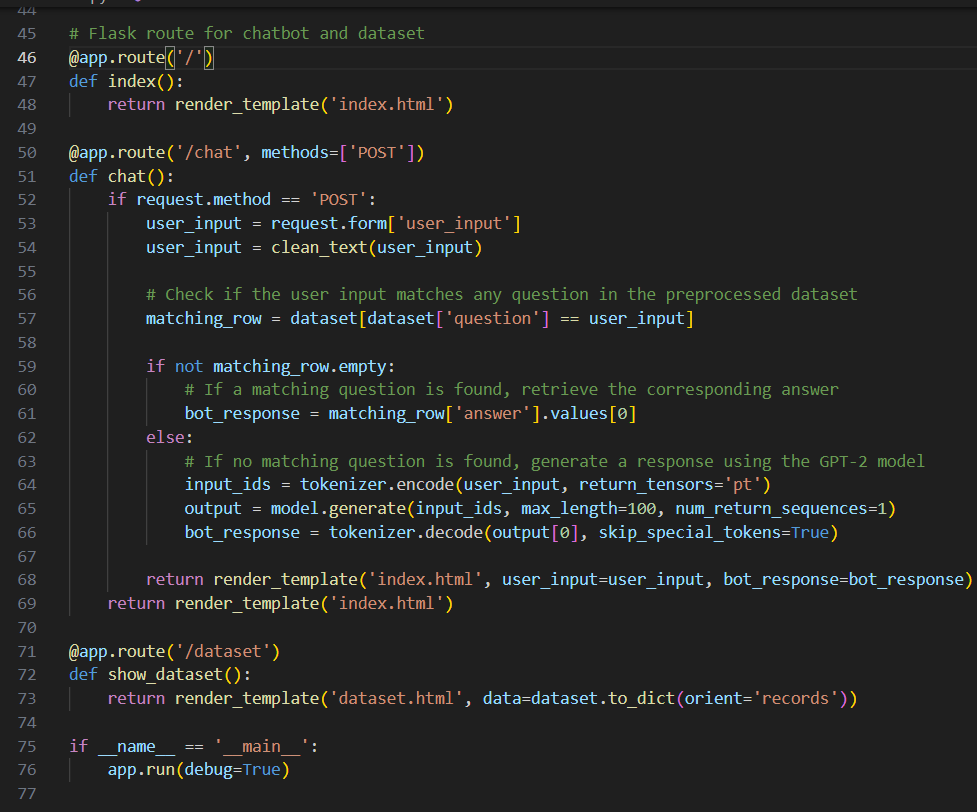
Data split in this occurred in the per-processed dataset (). It is in the form of Question and Answer table. The chatbot only accepted the data from this dataset only.

NOTE: You can access the dataset with the help of this address <http://127.0.0.1:5000/dataset>



**4. Integration with flask:**

The pre-processed dataset is integrated with the Flask web application. Flask routes are set up to handle user input and responses. When a user submits a query, it is cleaned using the clean text function. The cleaned query is then compared with the pre-processed dataset to find a matching question. If no match is found in the pre-processed dataset, the cleaned user query is tokenized using the GPT-2 tokenizer. The tokenized input is fed into the GPT-2 model, which generates a response. The response is then decoded using the tokenizer to obtain the final bot response. The generated response is displayed to the user through the web interface. This model and tokenizer are used in GPT-3 also.



5.Web interface:

The user interface is designed using HTML and CSS. The main page **(index.html)** includes an input field for user queries and displays both user inputs and bot responses. The dataset page **(dataset.html)** displays the pre-processed dataset. The web interface is designed to take user input, display user queries, and show bot responses. Users interact with the chatbot by typing questions into an input field and receiving responses in real-time.

6. How to run and interact with chatbot:

* Ensure all required libraries and dependencies are installed.
* Run the Python script in a local environment.
* Access the application through a web browser.
* Enter queries and interact with the chatbot.

Here is the codes for the chatbot and interfaces:

(dataset.html)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Preprocessed Dataset</title>

</head>

<body>

    <h1>Preprocessed Dataset</h1>

    <table border="1">

        <thead>

            <tr>

                <th>Question</th>

                <th>Answer</th>

            </tr>

        </thead>

        <tbody>

            {% for row in data %}

                <tr>

                    <td>{{ row.question }}</td>

                    <td>{{ row.answer }}</td>

                </tr>

            {% endfor %}

        </tbody>

</table>

</body>

</html>

(index.html)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Chatbot</title>

    <style>

        body {

            display: flex;

            background-color: #007bff;

            justify-content: center;

            align-items: center;

            height: 100vh;

            margin: 0;

        }

        .container {

            max-width: 400px;

            width: 100%;

            padding: 20px;

            background-color: #ffffff;

            border-radius: 10px;

            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

        }

        .message {

            margin-bottom: 10px;

        }

        .user-message strong {

            color: #007bff;

        }

        .bot-message strong {

            color: #28a745;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1 style="text-align: center;">Chatbot</h1>

        <div class="message user-message">

            <strong>You:</strong> {{ user\_input }}

        </div>

        <div class="message bot-message">

            <strong>Bot:</strong> {{ bot\_response }}

        </div>

        <form method="POST" action="/chat" style="text-align: center;">

            <label for="user\_input">You:</label>

            <input type="text" id="user\_input" name="user\_input" value="{{ user\_input }}">

            <input type="submit" value="Ask">

        </form>

    </div>

</body>

</html>

(Chatbot.py)

import re

import pandas as pd

import spacy

from flask import Flask, render\_template, request

from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load SpaCy model

nlp = spacy.load("en\_core\_web\_sm")

# Flask setup

app = Flask(\_\_name\_\_)

# Load the GPT-2 tokenizer and model

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

model = GPT2LMHeadModel.from\_pretrained("gpt2")

# Load the dataset from the specified file path

dataset = pd.read\_csv('dialogs.txt', delimiter="\t", header=None, names=["question", "answer"])

# Define the clean\_text function to preprocess text data

def clean\_text(text):

    text = re.sub(r'[^a-zA-Z\s]', '', text)

    text = re.sub(r'\s+', ' ', text).strip()

    text = text.lower()

    return text

# Define the remove\_repeating\_sentences function to remove repeating sentences from a dataset

def remove\_repeating\_sentences(dataset):

    seen\_sentences = set()

    filtered\_dataset = []

    for index, row in dataset.iterrows():

        if row["question"] not in seen\_sentences:

            seen\_sentences.add(row["question"])

            filtered\_dataset.append(row)

    return pd.DataFrame(filtered\_dataset)

# Preprocess the dataset

dataset = dataset.dropna()

dataset["question"] = dataset["question"].apply(clean\_text)

dataset["answer"] = dataset["answer"].apply(clean\_text)

dataset = remove\_repeating\_sentences(dataset)

# Flask route for chatbot and dataset

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/chat', methods=['POST'])

def chat():

    if request.method == 'POST':

        user\_input = request.form['user\_input']

        user\_input = clean\_text(user\_input)

        # Check if the user input matches any question in the preprocessed dataset

        matching\_row = dataset[dataset['question'] == user\_input]

        if not matching\_row.empty:

            # If a matching question is found, retrieve the corresponding answer

            bot\_response = matching\_row['answer'].values[0]

        else:

            # If no matching question is found, generate a response using the GPT-2 model

            input\_ids = tokenizer.encode(user\_input, return\_tensors='pt')

            output = model.generate(input\_ids, max\_length=100, num\_return\_sequences=1)

            bot\_response = tokenizer.decode(output[0], skip\_special\_tokens=True)

        return render\_template('index.html', user\_input=user\_input, bot\_response=bot\_response)

    return render\_template('index.html')

@app.route('/dataset')

def show\_dataset():

    return render\_template('dataset.html', data=dataset.to\_dict(orient='records'))

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

OUTPUT:

