**GitHub Link:** <https://github.com/SoeWunna29/-Machine-Learning---ChatGPT-Use-ChatGPT-to-create-customer-support-website.git>

**web\_based\_interface.py file**

from flask import Flask, render\_template, request  
import openai  
import os  
  
# Set up OpenAI API credentials  
openai.api\_key = os.environ["OPENAI\_API\_KEY"]  
  
# Set up Flask app  
app = Flask(\_\_name\_\_)  
  
# Define route for home page  
@app.route('/')  
def home():  
 return render\_template('index.html')  
  
# Define route for chat API  
@app.route('/chat', methods=['POST'])  
def chat():  
 # Get user's message from POST request  
 message = request.form['message']  
  
 # Call OpenAI API to generate response  
 response = openai.Completion.create(  
 engine="davinci",  
 prompt=message,  
 max\_tokens=60,  
 n=1,  
 stop=None,  
 temperature=0.5  
 )  
  
 # Extract text from response and return it  
 text = response.choices[0].text.strip()  
 return text  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

**homepage\_template.html file**

<!doctype html>  
<html>  
 <head>  
 <title>Chat with ChatGPT</title>  
 </head>  
 <body>  
 <h1>Chat with ChatGPT</h1>  
 <form id="chat-form">  
 <input type="text" name="message" id="chat-input">  
 <button type="submit">Send</button>  
 </form>  
 <div id="chat-output"></div>  
 <script>  
 const ***chatForm*** = ***document***.getElementById('chat-form');  
 const ***chatInput*** = ***document***.getElementById('chat-input');  
 const ***chatOutput*** = ***document***.getElementById('chat-output');  
  
 ***chatForm***.addEventListener('submit', async (event) => {  
 event.preventDefault();  
 const response = await fetch('/chat', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/x-www-form-urlencoded'  
 },  
 body: new URLSearchParams(new FormData(***chatForm***))  
 });  
 const text = await response.text();  
 ***chatInput***.value = '';  
 ***chatOutput***.innerHTML += '<p>You: ' + event.target.elements.message.value + '</p>';  
 ***chatOutput***.innerHTML += '<p>ChatGPT: ' + text + '</p>';  
 });  
 </script>  
 </body>  
</html>

**Jupyter Notebook File**

# python -m venv env  
#  
# source env/bin/activate  
#  
# pip install -r requirements.txt  
  
def remove\_newlines(serie):  
 serie = serie.str.replace('\n', ' ')  
 serie = serie.str.replace('\\n', ' ')  
 serie = serie.str.replace(' ', ' ')  
 serie = serie.str.replace(' ', ' ')  
 return serie  
  
  
import pandas as pd  
  
# Create a list to store the text files  
texts=[]  
  
# Get all the text files in the text directory  
for file in os.listdir("text/" + domain + "/"):  
  
 # Open the file and read the text  
 with open("text/" + domain + "/" + file, "r", encoding="UTF-8") as f:  
 text = f.read()  
  
 # Omit the first 11 lines and the last 4 lines, then replace -, \_, and #update with spaces.  
 texts.append((file[11:-4].replace('-',' ').replace('\_', ' ').replace('#update',''), text))  
  
# Create a dataframe from the list of texts  
df = pd.DataFrame(texts, columns = ['fname', 'text'])  
  
# Set the text column to be the raw text with the newlines removed  
df['text'] = df.fname + ". " + remove\_newlines(df.text)  
df.to\_csv('processed/scraped.csv')  
df.head()  
  
import tiktoken  
  
# Load the cl100k\_base tokenizer which is designed to work with the ada-002 model  
tokenizer = tiktoken.get\_encoding("cl100k\_base")  
  
df = pd.read\_csv('processed/scraped.csv', index\_col=0)  
df.columns = ['title', 'text']  
  
# Tokenize the text and save the number of tokens to a new column  
df['n\_tokens'] = df.text.apply(lambda x: len(tokenizer.encode(x)))  
  
# Visualize the distribution of the number of tokens per row using a histogram  
df.n\_tokens.hist()  
  
max\_tokens = 500  
  
  
# Function to split the text into chunks of a maximum number of tokens  
def split\_into\_many(text, max\_tokens=max\_tokens):  
 # Split the text into sentences  
 sentences = text.split('. ')  
  
 # Get the number of tokens for each sentence  
 n\_tokens = [len(tokenizer.encode(" " + sentence)) for sentence in sentences]  
  
 chunks = []  
 tokens\_so\_far = 0  
 chunk = []  
  
 # Loop through the sentences and tokens joined together in a tuple  
 for sentence, token in zip(sentences, n\_tokens):  
  
 # If the number of tokens so far plus the number of tokens in the current sentence is greater  
 # than the max number of tokens, then add the chunk to the list of chunks and reset  
 # the chunk and tokens so far  
 if tokens\_so\_far + token > max\_tokens:  
 chunks.append(". ".join(chunk) + ".")  
 chunk = []  
 tokens\_so\_far = 0  
  
 # If the number of tokens in the current sentence is greater than the max number of  
 # tokens, go to the next sentence  
 if token > max\_tokens:  
 continue  
  
 # Otherwise, add the sentence to the chunk and add the number of tokens to the total  
 chunk.append(sentence)  
 tokens\_so\_far += token + 1  
  
 return chunks  
  
  
shortened = []  
  
# Loop through the dataframe  
for row in df.iterrows():  
  
 # If the text is None, go to the next row  
 if row[1]['text'] is None:  
 continue  
  
 # If the number of tokens is greater than the max number of tokens, split the text into chunks  
 if row[1]['n\_tokens'] > max\_tokens:  
 shortened += split\_into\_many(row[1]['text'])  
  
 # Otherwise, add the text to the list of shortened texts  
 else:  
 shortened.append(row[1]['text'])  
  
df = pd.DataFrame(shortened, columns = ['text'])  
df['n\_tokens'] = df.text.apply(lambda x: len(tokenizer.encode(x)))  
df.n\_tokens.hist()  
  
import openai  
  
df['embeddings'] = df.text.apply(lambda x: openai.Embedding.create(input=x, engine='text-embedding-ada-002')['data'][0]['embedding'])  
  
df.to\_csv('processed/embeddings.csv')  
df.head()  
  
import numpy as np  
from openai.embeddings\_utils import distances\_from\_embeddings  
  
df=pd.read\_csv('processed/embeddings.csv', index\_col=0)  
df['embeddings'] = df['embeddings'].apply(eval).apply(np.array)  
  
df.head()  
  
  
def create\_context(  
 question, df, max\_len=1800, size="ada"  
):  
 *"""  
 Create a context for a question by finding the most similar context from the dataframe  
 """* # Get the embeddings for the question  
 q\_embeddings = openai.Embedding.create(input=question, engine='text-embedding-ada-002')['data'][0]['embedding']  
  
 # Get the distances from the embeddings  
 df['distances'] = distances\_from\_embeddings(q\_embeddings, df['embeddings'].values, distance\_metric='cosine')  
  
 returns = []  
 cur\_len = 0  
  
 # Sort by distance and add the text to the context until the context is too long  
 for i, row in df.sort\_values('distances', ascending=True).iterrows():  
  
 # Add the length of the text to the current length  
 cur\_len += row['n\_tokens'] + 4  
  
 # If the context is too long, break  
 if cur\_len > max\_len:  
 break  
  
 # Else add it to the text that is being returned  
 returns.append(row["text"])  
  
 # Return the context  
 return "\n\n###\n\n".join(returns)  
  
def answer\_question(  
 df,  
 model="text-davinci-003",  
 question="Am I allowed to publish model outputs to Twitter, without a human review?",  
 max\_len=1800,  
 size="ada",  
 debug=False,  
 max\_tokens=150,  
 stop\_sequence=None  
):  
 *"""  
 Answer a question based on the most similar context from the dataframe texts  
 """* context = create\_context(  
 question,  
 df,  
 max\_len=max\_len,  
 size=size,  
 )  
 # If debug, print the raw model response  
 if debug:  
 print("Context:\n" + context)  
 print("\n\n")  
  
 try:  
 # Create a completions using the question and context  
 response = openai.Completion.create(  
 prompt=f"Answer the question based on the context below, and if the question can't be answered based on the context, say \"I don't know\"\n\nContext: {context}\n\n---\n\nQuestion: {question}\nAnswer:",  
 temperature=0,  
 max\_tokens=max\_tokens,  
 top\_p=1,  
 frequency\_penalty=0,  
 presence\_penalty=0,  
 stop=stop\_sequence,  
 model=model,  
 )  
 return response["choices"][0]["text"].strip()  
 except Exception as e:  
 print(e)  
 return ""  
  
answer\_question(df, question="What day is it?", debug=False)  
  
answer\_question(df, question="What is our newest embeddings model?")  
  
answer\_question(df, question="What is ChatGPT?")  
  
"I don't know."  
  
'The newest embeddings model is text-embedding-ada-002.'  
  
'ChatGPT is a model trained to interact in a conversational way. It is able to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests.'