```
%%writefile app.py
import streamlit as st




import os
import requests
from bs4 import BeautifulSoup
import re
from sklearn.feature_extraction.text import TfidfVectorizer
import sqlite3
import pickle
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
from transformers import GPT2Tokenizer, GPT2LMHeadModel, AutoTokenizer, AutoModelForSeq2SeqLM
import streamlit as st
from langchain_core.messages import AIMessage, HumanMessage


def scrape_website(url):
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')
        paragraphs = soup.find_all('p')
        extracted_text = [paragraph.get_text() for paragraph in paragraphs]

        joined_text = " ".join(extracted_text)
        cleaned_text = re.sub(r'\s+', ' ', joined_text).strip()
        return cleaned_text
    except Exception as e:
        st.error(f"An error occurred while scraping the website: {e}")
        return None


def split_text(text):
    sentences = re.split(r'(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s', text)
    return sentences


def vectorize_data(sentences):
    try:
        vectorizer = TfidfVectorizer()
        tfidf_vectors = vectorizer.fit_transform(sentences)
        return tfidf_vectors, vectorizer
    except Exception as e:
        st.error(f"An error occurred during vectorization: {e}")
        return None, None


def store_vectors_in_database(tfidf_vectors, sentences):
    try:
        with sqlite3.connect('vector_database.db') as conn:
            cursor = conn.cursor()
            cursor.execute('''DROP TABLE IF EXISTS tfidf_vectors''')
            cursor.execute('''CREATE TABLE IF NOT EXISTS tfidf_vectors (
                                id INTEGER PRIMARY KEY,
                                sentence TEXT,
                                vector BLOB
                            )''')
            for i, vector in enumerate(tfidf_vectors):
                sentence = sentences[i]
                vector_serialized = pickle.dumps(vector)
                cursor.execute('''INSERT INTO tfidf_vectors (sentence, vector)
                                  VALUES (?, ?)''', (sentence, vector_serialized))
        st.write("TF-IDF vectors stored in database successfully.")
    except Exception as e:
        st.error(f"An error occurred while storing vectors in the database: {e}")


def keyword_matching(question, sentences):
    matching_sentences = []
    for sentence in sentences:
        if re.search(r'\b{}\b'.format(re.escape(question)), sentence, re.IGNORECASE):
            matching_sentences.append(sentence)
    return matching_sentences
```

```
def semantic_search(question, vectorizer):
    try:
        question_vector = vectorizer.transform([question])
        with sqlite3.connect('vector_database.db') as conn:
            cursor = conn.cursor()
            cursor.execute('''SELECT * FROM tfidf_vectors''')
            rows = cursor.fetchall()
            tfidf_vectors = [pickle.loads(row[2]) for row in rows]

        similarities = [cosine_similarity(question_vector, v)[0][0] for v in tfidf_vectors]
        max_similarity_index = np.argmax(similarities)
        most_similar_sentence = rows[max_similarity_index][1]

        return most_similar_sentence
    except Exception as e:
        st.error(f"An error occurred during semantic search: {e}")
        return None


def use_llm(question, max_response_length):
    try:
        tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
        model = GPT2LMHeadModel.from_pretrained("gpt2")

        inputs = tokenizer.encode(question, return_tensors="pt", max_length=tokenizer.model_max_length, truncation=True)
        max_length = min(tokenizer.model_max_length, len(inputs[0]) + max_response_length)

        output = model.generate(inputs, max_length=max_length, pad_token_id=tokenizer.eos_token_id)
        response = tokenizer.decode(output[0], skip_special_tokens=True)

        return response
    except Exception as e:
        st.error(f"An error occurred during GPT-2 response generation: {e}")
        return None


def give_answer(answer):
    tokenizer = AutoTokenizer.from_pretrained("Mr-Vicky-01/Bart-Finetuned-conversational-summarization")
    model = AutoModelForSeq2SeqLM.from_pretrained("Mr-Vicky-01/Bart-Finetuned-conversational-summarization")

    def generate_summary(answer):
        inputs = tokenizer([answer], max_length=1024, return_tensors='pt', truncation=True)
        summary_ids = model.generate(inputs['input_ids'], max_new_tokens=100, do_sample=False)
        summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
        return summary

    summary = generate_summary(answer)
    return summary


st.set_page_config(page_title="Webquery: Chat with Website", page_icon=":books:")
st.title("Webquery: Chat with Website")

if "chat_history" not in st.session_state:
    st.session_state.chat_history = [
        AIMessage(content="Hello, I am a bot. How can I help you?"),
    ]

if "sentences" not in st.session_state:
    st.session_state.sentences = []

if "vectorizer" not in st.session_state:
    st.session_state.vectorizer = None

with st.sidebar:
    st.header("Website URL !!!")
    website_url = st.text_input("Enter the website URL .......")
    submit = st.button("Submit")

if submit:
    try:
        if os.path.exists('vector_database.db'):
            os.remove('vector_database.db')

        text = scrape_website(website_url)
        st.session_state.sentences = split_text(text)
```

```
        st.session_state.sentences = split_text(text)
        tfidf_vectors, st.session_state.vectorizer = vectorize_data(st.session_state.sentences)
        store_vectors_in_database(tfidf_vectors, st.session_state.sentences)
    except Exception as e:
        st.error(f"An error occurred: {e}")

user_query = st.text_input("Type your question here....")

if user_query and user_query.strip() and st.session_state.vectorizer:
    matching_sentences = keyword_matching(user_query, st.session_state.sentences)
    most_similar_sentence = semantic_search(user_query, st.session_state.vectorizer)

    if most_similar_sentence:
        response = use_llm(most_similar_sentence, max_response_length=len(user_query.split()) * 5)
        answer = give_answer(response)
        st.session_state.chat_history.append(HumanMessage(content=user_query))
        st.session_state.chat_history.append(AIMessage(content=answer))

for message in st.session_state.chat_history:
    if isinstance(message, AIMessage):
        with st.chat_message("AI"):
            st.write(message.content)
    elif isinstance(message, HumanMessage):
        with st.chat_message("Human"):
            st.write(message.content)
```

```
Writing app.py
```

```
!npm install localtunnel
```

```
npm WARN saveError ENOENT: no such file or directory, open '/content/package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open '/content/package.json'
npm WARN content No description
npm WARN content No repository field.
npm WARN content No README data
npm WARN content No license field.

+ localtunnel@2.0.2
added 22 packages from 22 contributors and audited 22 packages in 4.587s

3 packages are looking for funding
  run `npm fund` for details

found 1 moderate severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details
```

```
!streamlit run /content/app.py &>/content/logs.txt & curl ipv4.icanhazip.com
```

```
35.243.173.45
```

```
!npx localtunnel --port 8501
```

```
npx: installed 22 in 2.968s
your url is: https://honest-doors-stick.loca.lt
```

Start coding or generate with AI.