Title: Code Genei AI

Name: Ranjan Kumar

.....

Milestone 1: Creating UI

1. Project Overview

This is a **Streamlit** chatbot application designed for conversational data exploration. It leverages the **OpenAI API** to process natural language queries, providing an intuitive interface for interacting with a dataset.

2. Features

- **Natural Language Interaction:** Ask questions about the dataset in plain English.
- **Persistent Chat History:** The application saves conversations, allowing users to revisit past inquiries.
- **Simple UI:** The interface is clean, user-friendly, and responsive across different devices.

3. System Requirements & Setup

Requirements:

- **Python:** Python 3.13.5
- **Dependencies:** streamlit, pandas, openai
- **Tool:** An **OpenAI API Key** is required.

Setup:

1. Save the code as **Chat_botApp.py**.
2. Install dependencies : **pip install streamlit pandas opena**i.
3. Place your OpenAI API key and the **bengaluru_house_prices.csv** file in the same directory.
4. Run from the terminal: **streamlit run Chat_botApp.py**.

Source Code

```
import streamlit as st
import pandas as pd
from openai import OpenAI

# ------------------- CONFIG -------------------
st.set_page_config(page_title="Chatbot with Dataset", layout="wide")
client = OpenAI(api_key=" ")  # 👈 put your API key here
```

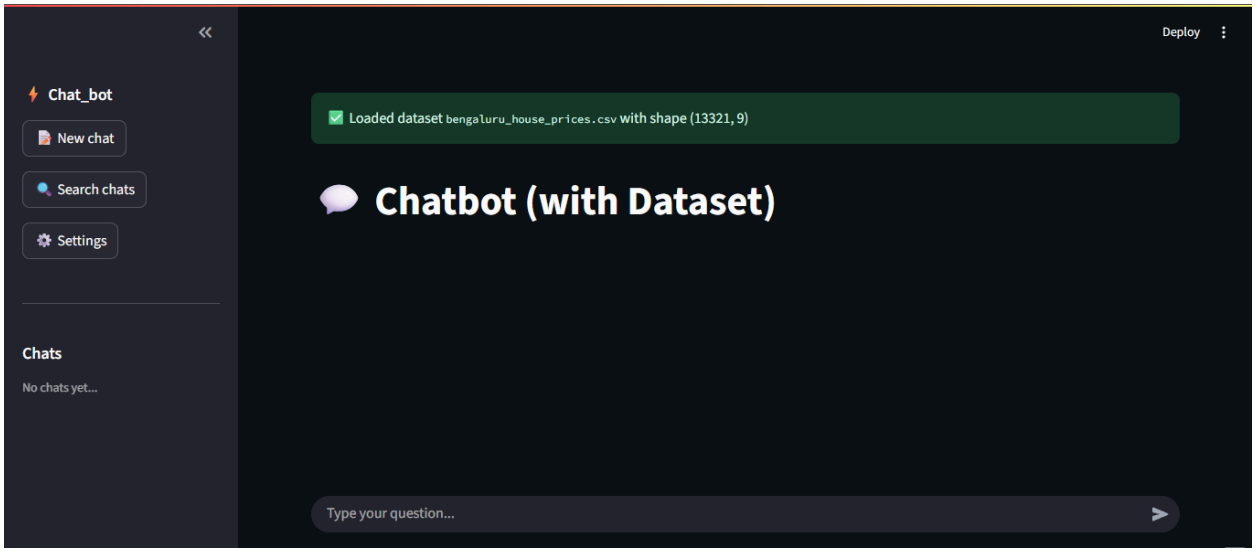
```
DATA_PATH = "bengaluru_house_prices.csv"  # 👈 dataset file path

# ------------------- SESSION STATE -------------------
if "messages" not in st.session_state:
    st.session_state.messages = []
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []
if "df" not in st.session_state:
    try:
        st.session_state.df = pd.read_csv(DATA_PATH)
        st.success(f"✅ Loaded dataset `{DATA_PATH}` with shape
{st.session_state.df.shape}")
    except Exception as e:
        st.error(f"❌ Failed to load dataset: {e}")
        st.stop()

# ------------------- SIDEBAR -------------------
with st.sidebar:
    st.markdown("### ⚡ Chat_bot")

    if st.button("📝 New chat"):
        if st.session_state.messages:

st.session_state.chat_history.append(st.session_state.messages)
        st.session_state.messages = []
        st.experimental_rerun()

    st.button("🔍 Search chats")
    st.button("⚙️ Settings ")

    st.markdown("---")
    st.subheader("Chats")
    if st.session_state.chat_history:
        for i, chat in enumerate(st.session_state.chat_history):
            if st.button(f"💬 Chat {i+1}", key=f"history_{i}"):
                st.session_state.messages = chat
                st.experimental_rerun()
```

```
    else:
        st.caption("No chats yet...")

# ------------------- MAIN CHAT WINDOW -------------------
st.title("💬 Chatbot (with Dataset)")

# Display previous messages (user → right, assistant → left)
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# ------------------- CHAT INPUT -------------------
if prompt := st.chat_input("Type your question..."):
    # Add user message
    st.session_state.messages.append({"role": "user", "content": prompt})
    with st.chat_message("user"):
        st.markdown(prompt)

    # ------------------- BOT RESPONSE -------------------
    with st.chat_message("assistant"):
        with st.spinner("Thinking..."):
            # Take a small dataset context
            df_sample = st.session_state.df.head(5).to_string()

            response = client.chat.completions.create(
                model="gpt-4o-mini",
                messages=[
                    {"role": "system", "content": f"You are a helpful
assistant. Here is some dataset context:\n{df_sample}"},
                    *st.session_state.messages,
                ]
            )

            reply = response.choices[0].message.content
            st.markdown(reply)

    # Save bot response
```

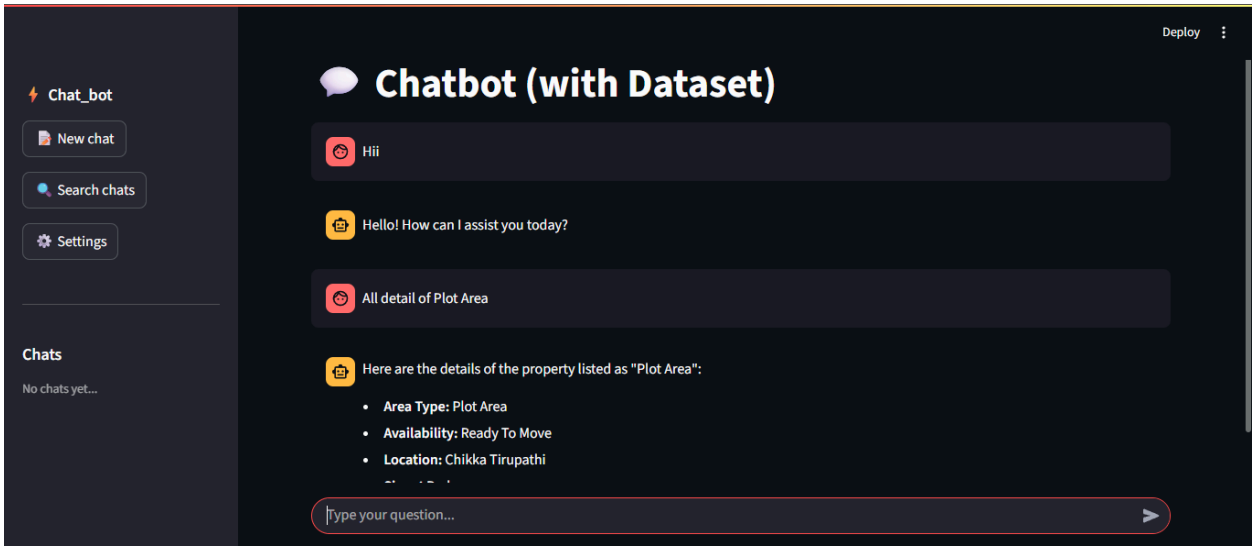
```
    st.session_state.messages.append({"role": "assistant", "content": reply})
```

UI Screenshots

Basic model



When interaction start



4. Approach & Methodology

Approach:

The core methodology is to create a thin, conversational layer on top of a powerful language model. This avoids the need to build a complex NLP engine from scratch, allowing the focus to remain on user experience and data integration.

Methodology:

1. **UI Scaffolding:** Utilized Streamlit's components for rapid UI development.
2. **State Management:** Leveraged **st.session_state** to maintain chat and data state across user interactions.
3. **Data Context:** Provided a small sample of the dataset in the API prompt to give the model the necessary context for basic queries.
4. **API Integration:** The OpenAI API is called with the user's prompt and a complete history of the conversation to generate a coherent response.

5. Limitations & Future Enhancements

- **Limitations:** The bot's ability is currently limited as it only uses a small sample of the dataset for context, preventing it from performing complex data analysis. The API key is also insecurely hardcoded.
- **Enhancements:** Future improvements could include full dataset analysis with pandas, secure API key management using Streamlit secrets, and the addition of data visualization features.