```
import streamlit as st
import pandas as pd
import numpy as np
import json
from sklearn.neighbors import NearestNeighbors

# PAGE CONFIG
st.set_page_config(
    page_title="EV Assistant ",
    layout="wide",
    page_icon=" ⚡ "
)
# GREEN THEME + HEADER CSS
st.markdown("""
<style>
.stApp { background-color: #F4FFF7; }

h1, h2, h3 { color: #1B5E20; }

.stButton > button {
    background-color: #2E7D32;
    color: white;
    border-radius: 10px;
    padding: 0.5em 1.5em;
    font-weight: bold;
}
.stButton > button:hover {
    background-color: #1B5E20;
}

.stTabs [data-baseweb="tab"] {
```

```
    background-color: #C8E6C9;
    border-radius: 8px;
    font-weight: bold;
}
.stTabs [aria-selected="true"] {
    background-color: #2E7D32;
    color: white;
}

/* HEADER CARD */
.header-card {
    background: linear-gradient(135deg, #E8F5E9, #F1FFF4);
    padding: 28px;
    border-radius: 18px;
    box-shadow: 0 8px 22px rgba(0,0,0,0.10);
    text-align: center;
    margin-bottom: 25px;
}
.header-title {
    font-size: 42px;
    font-weight: 800;
    color: #1B5E20;
}
.header-subtitle {
    font-size: 18px;
    color: #4E6E5D;
    margin-top: 6px;
}
.header-features {
    margin-top: 14px;
    font-size: 16px;
```

```
    font-weight: 600;
    color: #2E7D32;
}
.header-badge {
    display: inline-block;
    margin-top: 12px;
    background-color: #2E7D32;
    color: white;
    padding: 6px 14px;
    border-radius: 20px;
    font-size: 14px;
}
</style>
""", unsafe_allow_html=True)

# APP HEADER

st.markdown("""
<div class="header-card">
    <div class="header-title">🌱 EV Assistant </div>
    <div class="header-subtitle">
        Smart, Sustainable & User-Friendly Electric Vehicle Assistant
    </div>
    <div class="header-features">
        🔋 EV Recommendation   |  
        📍 Charging Stations   |  
        ⚡ Charging Time   |  
        💬 FAQ Assistant
    </div>
    <div class="header-badge">
        🌍 Green AI | Data-Driven EV Solution
```

```
    </div>
</div>
""", unsafe_allow_html=True)

# LOAD DATA

cars = pd.read_csv("cars_data_cleaned.csv")
stations = pd.read_csv("ev-charging-stations-india.csv")

with open("ev_faq_100.json", "r", encoding="utf-8") as f:
    faq_data = json.load(f)

faq = pd.DataFrame(faq_data)
faq["question"] = faq["question"].str.lower()

# CLEAN STATION DATA

stations.columns = stations.columns.str.lower()
if "lattitude" in stations.columns:
    stations.rename(columns={"lattitude": "latitude"}, inplace=True)

stations["latitude"] = pd.to_numeric(stations["latitude"], errors="coerce")
stations["longitude"] = pd.to_numeric(stations["longitude"], errors="coerce")
stations_clean = stations.dropna(subset=["latitude", "longitude"])

# FAQ CHATBOT LOGIC

def faq_chatbot(user_query):
    user_query = user_query.lower()
    for _, row in faq.iterrows():
        if row["question"] in user_query:
```

```
        return row["answer"]
    return "🤖 I can answer questions related to EVs, charging, battery, and range."

# EV RECOMMENDATION LOGIC

features = ["Battery", "Range", "Efficiency", "Estimated_US_Value"]
X = cars[features].fillna(0)
X_scaled = (X - X.mean()) / X.std()

def recommend_car(index, k=3):
    distances = np.linalg.norm(X_scaled - X_scaled.iloc[index], axis=1)
    rec_idx = np.argsort(distances)[1:k+1]
    return cars.iloc[rec_idx][
        ["Brand", "Model", "Battery", "Range", "Estimated_US_Value"]
    ]

# CHARGING STATION LOGIC

coords = np.radians(stations_clean[["latitude", "longitude"]])
knn = NearestNeighbors(n_neighbors=5, metric="haversine")
knn.fit(coords)

def nearby_stations(lat, lon):
    dist, idx = knn.kneighbors(np.radians([[lat, lon]]))
    result = stations_clean.iloc[idx[0]].copy()
    result["distance_km"] = dist[0] * 6371
    return result[["name", "city", "state", "type", "distance_km"]]

# TABS

tab1, tab2, tab3, tab4 = st.tabs([
```

```
    " 🚗 EV Recommendation",
    " 📍 Charging Stations",
    " ⚡ Charging Time",
    " 💬 EV FAQ Chat"
])

# TAB 1 – EV RECOMMENDATION

with tab1:
    st.header(" 🔋 EV Car Recommendation")

    cars["car_name"] = cars["Brand"] + " - " + cars["Model"]
    cars_sorted = cars.sort_values("car_name")

    selected_car = st.selectbox("Select EV Car", cars_sorted["car_name"].unique())
    car_index = cars_sorted[cars_sorted["car_name"] == selected_car].index[0]

    st.subheader(" 📌 Selected Car Details")
    st.dataframe(
        cars.loc[car_index][
            ["Brand", "Model", "Battery", "Range", "Efficiency", "Estimated_US_Value"]
        ].to_frame(name="Value")
    )

    if st.button("Recommend Similar Cars"):
        st.success("Similar EVs")
        st.dataframe(recommend_car(car_index))

# TAB 2 – CHARGING STATIONS

with tab2:
```

```
    st.header("📍 Find EV Charging Stations")

    city_list = stations_clean["city"].dropna().str.title().unique()
    city_list.sort()

    selected_city = st.selectbox("Select City", city_list)
    city_data = stations_clean[stations_clean["city"].str.title() == selected_city]

    if st.button("Find Nearby Stations"):
        if not city_data.empty:
            avg_lat = city_data["latitude"].mean()
            avg_lon = city_data["longitude"].mean()
            st.dataframe(nearby_stations(avg_lat, avg_lon))
            st.map(city_data[["latitude", "longitude"]])
        else:
            st.warning("No stations found for this city.")

# TAB 3 – CHARGING TIME

with tab3:
    st.header("⚡ Charging Time Predictor (Demo)")

    battery_percent = st.slider("Battery Percentage (%)", 0, 100, 30)
    planned_distance = st.slider("Planned Distance (km)", 0, 500, 120)

    if st.button("Predict Charging Time"):
        avg_range = 300
        full_charge_time = 8

        available_range = (battery_percent / 100) * avg_range
```

```
        if planned_distance <= available_range:
            st.success("✅ No charging required.")
        else:
            remaining = planned_distance - available_range
            time_needed = (remaining / avg_range) * full_charge_time
            st.success(f"🔌 Estimated Charging Time: {time_needed:.2f} hours")

# TAB 4 – FAQ CHATBOT

with tab4:
    st.header("💬 EV Assistant – FAQ")

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

    user_question = st.text_input("Ask your EV question 👇")

    if st.button("Ask"):
        if user_question.strip():
            answer = faq_chatbot(user_question)
            st.session_state.faq_chat.append(("You", user_question))
            st.session_state.faq_chat.append(("Bot", answer))

    for sender, msg in st.session_state.faq_chat:
        if sender == "You":
            st.markdown(f"👱 **You:** {msg}")
        else:
            st.markdown(f"🤖 **Bot:** {msg}")
```