import streamlit as st

from geopy.geocoders import Nominatim

import subprocess

import pandas as pd

# Function to update coordinates in code

def update\_coordinates\_in\_code(code\_file, new\_latitude, new\_longitude):

    try:

        with open(code\_file, 'r', encoding='utf-8') as f:

            lines = f.readlines()

        for i in range(len(lines)):

            if "latitude =" in lines[i]:

                lines[i] = f"latitude = {new\_latitude}\n"

            elif "longitude =" in lines[i]:

                lines[i] = f"longitude = {new\_longitude}\n"

        with open(code\_file, 'w', encoding='utf-8') as f:

            f.writelines(lines)

        st.success(f"Coordinates updated in {code\_file}")

    except Exception as e:

        st.error(f"An error occurred: {e}")

# Function to run code

def run\_code(code\_file):

    try:

        # Run the code and collect any output, including paths or data of images

        result = subprocess.run(['python', code\_file], capture\_output=True, text=True)

        code\_output = result.stdout

        # Extract image paths or data from the code output

        image\_paths = extract\_image\_paths(code\_output)

        # Display images at the end

        for image\_path in image\_paths:

            st.image(image\_path, caption='Image')

        # Display any other code output if needed

        st.code(code\_output, language='python')

    except Exception as e:

        st.error(f"An error occurred: {e}")

# Function to extract image paths

def extract\_image\_paths(code\_output):

    # Example function to extract image paths from code output

    # You should customize this based on how your code generates or saves images

    image\_paths = []

    # Your logic here to extract image paths from the code\_output

    return image\_paths

# Function to generate cluster information

def generate\_cluster\_info(df\_evaluate):

    # Your logic for generating cluster information goes here...

    # You can customize this based on the characteristics of each cluster

    # Example: Generating sample cluster information

    cluster\_info = {

        'Cluster 0 (Green)': 'Abundant availability of both restaurants and groceries.',

        'Cluster 1 (Orange)': 'Plentiful restaurants, with groceries available to a lesser extent.',

        'Cluster 2 (Red)': 'Relatively limited availability of both restaurants and groceries.'

    }

    return cluster\_info

# Function to get coordinates from location name

def get\_coordinates(location\_name):

    geolocator = Nominatim(user\_agent="geo\_app")

    location = geolocator.geocode(location\_name)

    if location:

        return location.latitude, location.longitude

    else:

        return None

# Function to produce color based on cluster label

def color\_producer(cluster\_label):

    # Define a color mapping logic based on your clusters

    color\_mapping = {

        0: 'green',

        1: 'orange',

        2: 'red',

        # Add more cluster labels and corresponding colors if needed

    }

    # Return the color for the given cluster label

    return color\_mapping.get(cluster\_label, 'gray')  # Default to gray if label not found

# Streamlit code

st.title("🌍Geolocational Data Analysis App")

# Sidebar menu

menu\_selection = st.sidebar.radio("Menu", ["About the Project", "Enter Location Name", "Enter Coordinates", "Home", "EDA"])

if menu\_selection == "About the Project":

    st.header("About the Project")

    about\_the\_project\_text = """

    Welcome to our Geolocational Data Exploratory Analysis application, a powerful tool that integrates k-means, agglomerative hierarchical, and affinity propagation clustering algorithms. This application streamlines the exploration of spatial datasets, empowering users to uncover meaningful patterns and make informed decisions.

    Upon entering user-provided coordinates, the application dynamically updates geographical details for all three clustering algorithms. The user-friendly interface facilitates a seamless exploration process. Marked apartments on an embedded map in HTML format are color-coded into three clusters, providing users with immediate insights into spatial distributions without the need for separate descriptions:

    - \*\*Cluster 0 (Green):\*\* Abundant availability of both restaurants and groceries.

    - \*\*Cluster 1 (Orange):\*\* Plentiful restaurants, with groceries available to a lesser extent.

    - \*\*Cluster 2 (Red):\*\* Relatively limited availability of both restaurants and groceries.

    The visual representation of marked apartments on the map enhances decision-making by offering insights into the amenities available in different regions. Running each clustering algorithm generates box plots, visually representing relationships within the dataset. After each algorithm's execution, the application automatically calculates silhouette scores, enabling users to assess the quality of clusters and choose results based on the clustering method with the highest silhouette score.

    Our application is designed not only for data exploration but also to empower users to make location-based decisions based on the unique characteristics of their geolocational data. With customizable coordinates, dynamic visualizations, embedded maps, and automatic silhouette score calculations, this application ensures a comprehensive and user-friendly geospatial analysis experience. Your journey to uncover spatial insights and make data-driven decisions begins here—effortlessly explore and understand the nuances of your data.

    """

    st.markdown(about\_the\_project\_text)

    # Add images after the text using image names

    st.image("clusters.jpg", use\_column\_width=True)

elif menu\_selection == "Enter Location Name":

    st.header("Enter Location Name")

    # Input field for location name

    location\_name = st.text\_input("Enter Location Name:")

    # Button to retrieve coordinates

    if st.button("Retrieve Coordinates"):

        if location\_name:

            # Display a loading GIF while retrieving coordinates

            loading\_mp4\_path = "coordinates.mp4"  # Replace with the name of your MP4 file

            st.video(loading\_mp4\_path, format="video/mp4", start\_time=0)

            # Retrieve coordinates

            coordinates = get\_coordinates(location\_name)

            # Display the result

            st.write("")  # Add an empty line for spacing

            if coordinates:

                st.success(f"Coordinates for {location\_name}: Latitude={coordinates[0]}, Longitude={coordinates[1]}")

            else:

                st.error("Unable to retrieve coordinates. Please check the location name.")

        else:

            st.warning("Please enter a location name.")

elif menu\_selection == "Enter Coordinates":

    st.header("Enter Coordinates")

    # Input fields for latitude and longitude

    user\_latitude = st.number\_input("Enter Latitude:")

    user\_longitude = st.number\_input("Enter Longitude:")

    # Display the user's input

    st.write(f"Coordinates - Latitude: {user\_latitude}, Longitude: {user\_longitude}")

    # Button to update coordinates in source codes

    if st.button("Update Coordinates"):

        # Update coordinates in all three source codes

        update\_coordinates\_in\_code("code\_file\_1.py", user\_latitude, user\_longitude)

        update\_coordinates\_in\_code("code\_file\_2.py", user\_latitude, user\_longitude)

        update\_coordinates\_in\_code("code\_file\_3.py", user\_latitude, user\_longitude)

        st.success("Coordinates updated in all source codes. To run the updated codes, use the respective 'Run Code' options.")

elif menu\_selection == "Home":

    st.header("Home")

    # Create a single row with three columns for the buttons

    col1, col2, col3 = st.columns(3)

    # Dropdown button for code 1

    selected\_option\_1 = col1.selectbox("Select an option for Code 1:", ["Select Option", "View Code", "Run Code"], key="code1")

    # Process the selected option for code 1

    if selected\_option\_1 == "View Code":

        st.code(open("code\_file\_1.py").read(), language='python')

    elif selected\_option\_1 == "Run Code":

        run\_code("code\_file\_1.py")

    # Dropdown button for code 2

    selected\_option\_2 = col2.selectbox("Select an option for Code 2:", ["Select Option", "View Code", "Run Code"], key="code2")

    # Process the selected option for code 2

    if selected\_option\_2 == "View Code":

        st.code(open("code\_file\_2.py").read(), language='python')

    elif selected\_option\_2 == "Run Code":

        run\_code("code\_file\_2.py")

    # Dropdown button for code 3

    selected\_option\_3 = col3.selectbox("Select an option for Code 3:", ["Select Option", "View Code", "Run Code"], key="code3")

    # Process the selected option for code 3

    if selected\_option\_3 == "View Code":

        st.code(open("code\_file\_3.py").read(), language='python')

    elif selected\_option\_3 == "Run Code":

        run\_code("code\_file\_3.py")

elif menu\_selection == "EDA":

    st.header("Exploratory Data Analysis (EDA)")

    eda\_option = st.sidebar.selectbox("Select an option for EDA:", ["Select Option", "Maps", "Cluster Info"])

    # Process the selected EDA option

    if eda\_option == "Maps":

        st.subheader("Maps")

        # File uploader for uploading multiple HTML files

        uploaded\_files = st.file\_uploader("Upload your HTML files", type=["html"], accept\_multiple\_files=True)

        if uploaded\_files is not None:

            for file\_number, uploaded\_file in enumerate(uploaded\_files, start=1):

                # Display each uploaded HTML file using st.components.v1.html

                st.subheader(f"Map {file\_number}")

                map\_html = uploaded\_file.read().decode('utf-8')

                st.components.v1.html(map\_html, height=600)

    elif eda\_option == "Cluster Info":

        st.subheader("Cluster Info")

        # Display cluster information in an expander

        with st.expander("Cluster Information"):

            # Define cluster information

            cluster\_data = {

                'Cluster 0 (Green)': 'Abundant availability of both restaurants and groceries.',

                'Cluster 1 (Orange)': 'Plentiful restaurants, with groceries available to a lesser extent.',

                'Cluster 2 (Red)': 'Relatively limited availability of both restaurants and groceries.'

            }

            # Display colored circle and cluster information

            for cluster, description in cluster\_data.items():

                color = color\_producer(int(cluster.split()[1]))

                st.markdown(f'<span style="color: {color}">&#11044;</span> \*\*{cluster} Info:\*\* {description}', unsafe\_allow\_html=True)