import pandas as pd

import folium

from math import radians, sin, cos, sqrt, atan2

import networkx as nx

# Haversine formula to calculate distance between two points given their coordinates

def calculate\_distance(lat1, lon1, lat2, lon2):

   R = 6371.0  # Radius of the Earth in kilometers

   lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])

   dlon = lon2 - lon1

   dlat = lat2 - lat1

   a = sin(dlat / 2)\*\*2 + cos(lat1) \* cos(lat2) \* sin(dlon / 2)\*\*2

   c = 2 \* atan2(sqrt(a), sqrt(1 - a))

   distance = R \* c

   return distance

# Read data from Excel file

data = pd.read\_excel('/content/ai\_city.xlsx')

# User input: City name for which the user wants to calculate distances

user\_city\_name = input("Enter the city name: ")

# Find the city in the DataFrame

city\_row = data[data['city'] == user\_city\_name]

if not city\_row.empty:

   # City found, use its coordinates as the starting point

   start\_lat = city\_row['lat'].iloc[0]

   start\_lon = city\_row['lng'].iloc[0]

   # Calculate distances

   data['distance\_to\_target'] = data.apply(lambda row: calculate\_distance(row['lat'], row['lng'], start\_lat, start\_lon), axis=1)

   # Display the table in a neat format

   print(data[['city', 'distance\_to\_target']].to\_markdown(index=False))

   # Create a folium map centered around the starting city

   m = folium.Map(location=[start\_lat, start\_lon], zoom\_start=6)

   # Add markers for each city

   for \_, row in data.iterrows():

       folium.Marker([row['lat'], row['lng']], popup=row['city'] + f" ({row['distance\_to\_target']:.2f} km)").add\_to(m)

   # Create a graph using NetworkX

   G = nx.Graph()

   # Add nodes and edges to the graph

   for \_, row in data.iterrows():

       G.add\_node(row['city'], pos=(row['lat'], row['lng']))

   for \_, row in data.iterrows():

       if row['city'] != user\_city\_name:

           distance = calculate\_distance(start\_lat, start\_lon, row['lat'], row['lng'])

           G.add\_edge(user\_city\_name, row['city'], weight=distance)

   # Find the shortest path to each city using Dijkstra's algorithm

   for \_, row in data.iterrows():

       if row['city'] != user\_city\_name:

           shortest\_path = nx.shortest\_path(G, source=user\_city\_name, target=row['city'], weight='weight')

           # Add the shortest path to the map in blue

           shortest\_path\_coordinates = [(data[data['city'] == city]['lat'].iloc[0], data[data['city'] == city]['lng'].iloc[0]) for city in shortest\_path]

           folium.PolyLine(shortest\_path\_coordinates, color="blue", weight=2.5, opacity=1).add\_to(m)

           # Print the shortest path for each city

           print(f"Shortest path from {user\_city\_name} to {row['city']}: {shortest\_path}")

   # Save the map as an HTML file

   map\_html\_path = 'table\_and\_map\_with\_shortest\_paths.html'

   m.save(map\_html\_path)

   print(f"Table and map with shortest paths saved as '{map\_html\_path}'. Open the file in a web browser to view both.")

else:

   print(f"City '{user\_city\_name}' not found in the data.")