AIR QUALITY HACKATHON - Respirer Living Sciences

Team Krish - Krish Joshi

PROBLEM STATEMENT

A challenge of living in an Indian city is using public transport, especially buses, to commute between Location A and Location B. Naturally, we tend to prefer the route with the least amount of time to reach, however commuters usually encounter high traffic and high air pollution.

GOAL

Find out the cleanest (least pollution) route through the city ie. avoiding areas with the high aqi index without affecting travel time significantly.

PROPOSED METHODOLOGY

- Download data from respective websites and load them into the working directory.
- Clean and preprocess the bus stops data and route data into a dataframe.
- Perform geospatial analysis to calculate proximity (distance) of each bus stop to the nearest air quality monitoring system and add that value to it.
- Use trips data to get the routes and the stops in the trip route in a dictionary.
- Utilize Djikstar's algorithm with aqi being the parameter for the greedy algorithm.
- Create visualization for each stop showing routes and cleanest routes.

CODE - python3

Joining dataframes using attributes values and removing irrelevant fields :

```
import pandas as pd

# Load the CSV file with AQI data
aqi_data = pd.read_csv("/Users/krishjoshi/Desktop/Python/HackAQI/stopsdata_calculated.csv")

# Load the CSV file with stop data
stop_data = pd.read_csv("/Users/krishjoshi/Desktop/Python/HackAQI/GTFS/stops.csv")

# Create a dictionary mapping 'stop_name' to 'stop_id'
stop_name_to_id = stop_data.set_index('stop_name')['stop_id'].to_dict()

# Add 'stop_id' to the AQI data based on 'stop_name'
aqi_data['stop_id'] = aqi_data['stop_name'].map(stop_name_to_id)

# Save the modified AQI data to a new CSV file
aqi_data.to_csv("aqi_data_with_stop_id.csv", index=False)
```

Removing duplicates from the dataframe :

```
import pandas as pd

# Replace 'input.csv' with the path to your CSV file.
csv_file_path = 'routesdata.csv'

# Read the CSV file into a Pandas DataFrame.
df = pd.read_csv(csv_file_path)
```

```
# Remove duplicates based on the 'stop_name' column.
df_cleaned = df.drop_duplicates()

# Save the cleaned DataFrame to a new CSV file if needed.
# Replace 'output.csv' with the desired output file path.
output_csv_file = 'routesdata2.csv'
df_cleaned.to_csv(output_csv_file, index=False)

# Print the cleaned DataFrame (optional).
print(df_cleaned)
```

Code to create basic streamlit interactive graph and finding out closest aqi station :

```
import csv
import json
import pandas as pd

# Define a list to store the lat-long tuples for stops
stops_list = []

# Open the CSV file for reading
with open('/Users/krishjoshi/Desktop/Python/HackAQI/GTFS/stops.csv', 'r') as csvfile:
    # Create a CSV reader object
    csvreader = csv.reader(csvfile)

    # Skip the header row
    next(csvreader)
```

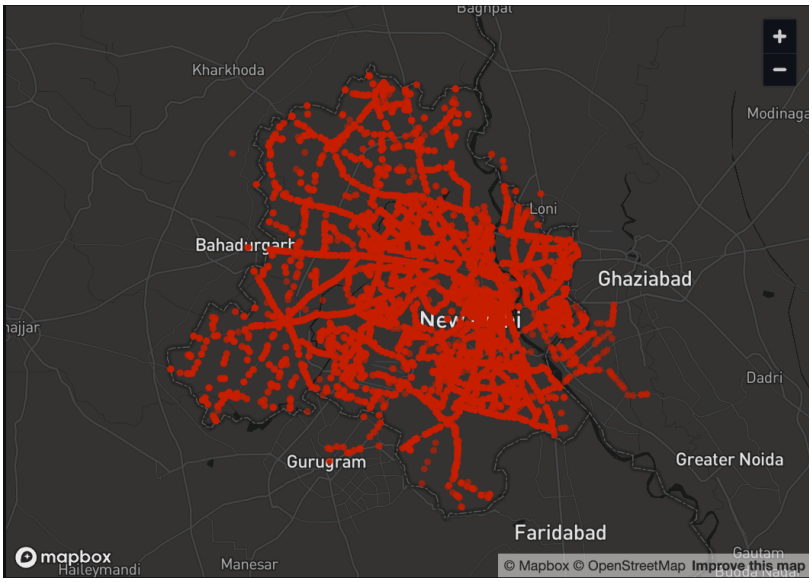
```
    # Iterate over each row in the CSV
    for row in csvreader:
        # Extract latitude and longitude values from the row
        stop_lat = float(row[2])
        stop_lon = float(row[3])
        stop_name = row[4]

        # Append the lat-long tuple to the list
        stops_list.append((stop_lat, stop_lon,stop_name))

# Print the list of lat-long tuples
#print(stops_list)



# Define a list to store the lat-long tuples for aqi monitoring stations
aqi_list = []

# Open the JSON file for reading
with open('delhi.json', 'r') as jsonfile:
    # Load the JSON data
    data = json.load(jsonfile)

    # Iterate over each item in the JSON array
    for item in data:
        # Extract latitude and longitude values from each item
        lat = item["lat"]
        lon = item["long"]
        aqi = item["stationAqi"]
        stationName = item["stationName"]
```

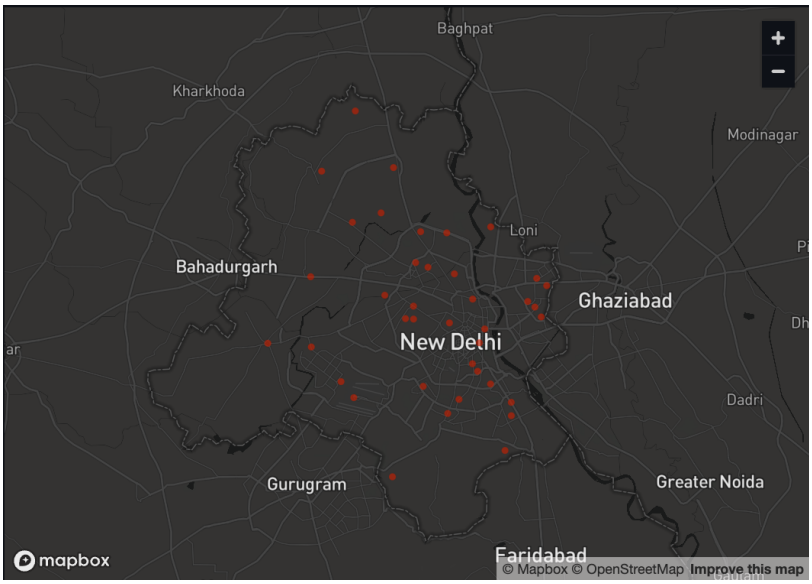
```
        # Append the lat-long tuple to the list
        aqi_list.append((lat, lon, aqi, stationName))

# Print the list of lat-long tuples
#print(aqi_list)

# Create a list of dictionaries with "latitude" and "longitude" keys
stops_coordinates_dict_list = [{"latitude": lat, "longitude": lon, "type": "stops", "stop_name": stop_name} for lat, lon, stop_name in stops_list]
aqi_coordinates_dict_list = [{"latitude": lat, "longitude": lon, "type": "aqi", "aqi": aqi, "stationName": stationName} for lat, lon, aqi, stationName in aqi_list]

# Plotting these points on an interactive map using streamlit
import streamlit as st

# Create a Streamlit app
st.title("Interactive Map")

# Map showing stops data
st.map(stops_coordinates_dict_list)

# Map showing aqi data
st.map(aqi_coordinates_dict_list)

'''

# Calculating closest aqi station
import streamlit as st
```

```
from geopy.distance import geodesic

# Calculate the minimum distances and store AQI values
for stop_data in stops_coordinates_dict_list:
    stop_location = (stop_data["latitude"], stop_data["longitude"])
    min_distance = float("inf")  # Initialize with a large value
    closest_aqi = None

    for aqi_data in aqi_coordinates_dict_list:
        aqi_location = (aqi_data["latitude"], aqi_data["longitude"])
        distance = geodesic(stop_location, aqi_location).kilometers

        if distance < min_distance:
            min_distance = distance
            closest_aqi = aqi_data["aqi"]
            aqi_stationName = aqi_data["stationName"]

    stop_data["closest_aqi"] = closest_aqi  # Store the closest AQI value
    stop_data["aqi_stationName"] = aqi_stationName # Store the AQI station name

# print(stops_coordinates_dict_list)

# Create a DataFrame from the list of dictionaries
df = pd.DataFrame(stops_coordinates_dict_list)

# Specify the path where you want to save the CSV file
csv_file_path = "stopsdata_calculated.csv"

# Save the DataFrame to a CSV file
```

```
df.to_csv(csv_file_path, index=False)


# Print a message to confirm the file has been saved
print(f"Data has been saved to {csv_file_path}")
'''
```

Interactive Map plots the stops and the aqi stations :





Creating graph usings stops as nodes and calculating the cleanest route:

```
import pandas as pd
import networkx as nx
import math
import pickle

# Function to load or create the graph
def load_or_create_graph():
    try:
        with open('graph.pkl', 'rb') as file:
            G = pickle.load(file)
            print("Graph loaded from file.")
    except FileNotFoundError:
        G = create_graph()
        with open('graph.pkl', 'wb') as file:
            pickle.dump(G, file)
        print("Graph created and saved to file.")
    return G

# Function to create the graph
def create_graph(num_trip_ids=10):
    print("Creating the graph...")
    # Load trip data
    trip_data = pd.read_csv('routesdata.csv')
    trip_data['stop_id'] = trip_data['stop_id'].astype(int)
    trip_data['stop_name'] = trip_data['stop_name'].astype(str)

    # Load AQI data
    aqi_data = pd.read_csv('stopsdata_calculated.csv')
```

```
aqi_data['stop_name'] = aqi_data['stop_name'].astype(str)

# Create a dictionary to map stop names to AQI values
stop_aqi_mapping = dict(zip(aqi_data['stop_name'], aqi_data['closest_aqi']))

# Create a graph
G = nx.DiGraph()

count = 0
# Create nodes in the graph
for _, row in trip_data.iterrows():
    stop_name = row['stop_name']
    latitude = row['stop_lat']
    longitude = row['stop_lon']
    aqi = stop_aqi_mapping.get(stop_name, 0)  # Use 0 as default AQI if not found in AQI data
    G.add_node(stop_name, latitude=latitude, longitude=longitude, aqi=aqi)
    count += 1
    print("Node Added : " + str(count))

# Create edges in the graph based on trip data
trip_groups = trip_data.groupby('trip_id')
edges_to_add = []

count = 0
for _, group in trip_groups:
    stops = group['stop_name'].tolist()
    for i in range(len(stops) - 1):
        source = stops[i]
        target = stops[i + 1]
```

```
            source_data = trip_data[trip_data['stop_name'] == source].iloc[0]
            target_data = trip_data[trip_data['stop_name'] == target].iloc[0]
                        distance = math.sqrt((source_data['stop_lat'] - target_data['stop_lat']) ** 2 +
(source_data['stop_lon'] - target_data['stop_lon']) ** 2)
            edges_to_add.append((source, target, {'distance': distance}))
            count += 1
            print("Edge Added : " + str(count))

            if num_trip_ids is not None and count >= num_trip_ids:
               break

    G.add_edges_from(edges_to_add)
    print("Graph creation completed.")
    return G

# Define a function to find all stops in a path
def find_all_stops_in_path(G, path):
    stops = [node for node in path]
    return stops

# Define a function to find the shortest and cleanest path
def find_shortest_and_cleanest_path(G, source, target):
    print("Finding shortest and cleanest paths...")
    shortest_path = nx.shortest_path(G, source=source, target=target, weight='distance')
    cleanest_path = nx.shortest_path(G, source=source, target=target, weight='aqi')

    # Calculate the distance along the shortest path
    shortest_distance = sum(G.get_edge_data(shortest_path[i], shortest_path[i+1])['distance'] for i
in range(len(shortest_path) - 1))
```

```
    # Calculate the distance along the cleanest path
     cleanest_distance = sum(G.get_edge_data(cleanest_path[i], cleanest_path[i+1])['distance'] for
i in range(len(cleanest_path) - 1))

    avg_aqi_shortest = sum(G.nodes[node]['aqi'] for node in shortest_path) / len(shortest_path)
    avg_aqi_cleanest = sum(G.nodes[node]['aqi'] for node in cleanest_path) / len(cleanest_path)

    print("Paths found.")

    return {
        'shortest_path': shortest_path,
        'cleanest_path': cleanest_path,
        'shortest_distance': shortest_distance,
        'cleanest_distance': cleanest_distance,
        'avg_aqi_shortest': avg_aqi_shortest,
        'avg_aqi_cleanest': avg_aqi_cleanest
    }

# Example usage
source_stop = "Narela Terminal"
target_stop = "Sec A-9 Narela"

# Load or create the graph
G = load_or_create_graph()

result = find_shortest_and_cleanest_path(G, source_stop, target_stop)

# Get all stops in the shortest path
```

```
shortest_stops = find_all_stops_in_path(G, result['shortest_path'])

# Get all stops in the cleanest path
cleanest_stops = find_all_stops_in_path(G, result['cleanest_path'])

print("Shortest Path Stops:", shortest_stops)
print("Cleanest Path Stops:", cleanest_stops)
print("Shortest Distance:", result['shortest_distance'])
print("Cleanest Distance:", result['cleanest_distance'])
print("Average AQI for Shortest Path:", result['avg_aqi_shortest'])
print("Average AQI for Cleanest Path:", result['avg_aqi_cleanest'])
```