**Restaurant Recommendation System and Employee Route Optimization**

**Abstract:**

This project aims to develop a restaurant recommendation system and an employee route optimization tool using Streamlit, a Python library for building interactive web applications. The restaurant recommendation system helps users find nearby restaurants based on their location and preferences. The employee route optimization tool calculates the shortest paths between different employee locations to optimize travel routes.

**Algorithm used:**

Graph: Used graph Structure to use prims algorithm

Haversine formula: To calculate the distance between two points on Earth's surface, taking into account the curvature of the Earth.

Prim's algorithm and Priority Queue: To find the minimum spanning tree (MST) of a graph, which is used to find the shortest path connecting all restaurants in the recommendation system.

Dijkstra's algorithm: To find the shortest path between pairs of nodes in the employee route optimization tool.

**Implementation:**

import streamlit as st

import pandas as pd

from geopy.distance import geodesic

import numpy as np

import requests

import streamlit as st

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

from PIL import Image

import webbrowser

import geocoder

import networkx as nx

# --- Data Loading and Preprocessing ---

# Read data from Excel file

file\_path = r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\ADSA\ADSA PR\Employee.xlsx"

df\_employee = pd.read\_excel(file\_path)

# Load the CSV data for Chennai and Coimbatore

file\_path\_chennai = r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\ADSA\ADSA PR\Zomato Chennai Listing 2020.csv"

file\_path\_coimbatore = r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\ADSA\ADSA PR\Coimbatore Restaraunts.csv"

df\_chennai = pd.read\_csv(file\_path\_chennai)

df\_coimbatore = pd.read\_csv(file\_path\_coimbatore)

# --- Helper Functions ---

# Calculate the haversine distance between two points on Earth

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

R = 6371 # Radius of the Earth in kilometers

dlat = radians(lat2 - lat1)

dlon = radians(lon2 - lon1)

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

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

distance = R \* c

return distance

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = np.zeros((vertices, vertices))

def add\_edge(self, u, v, weight):

self.graph[u][v] = weight

self.graph[v][u] = weight

# Prim's algorithm to find minimum spanning tree (MST)

def prim\_mst(graph):

parent = [-1] \* graph.V

key = [float('inf')] \* graph.V

mst\_set = [False] \* graph.V

key[0] = 0 # Starting node

parent[0] = -1

for \_ in range(graph.V):

u = min\_key(key, mst\_set)

mst\_set[u] = True

for v in range(graph.V):

if graph.graph[u][v] > 0 and not mst\_set[v] and key[v] > graph.graph[u][v]:

key[v] = graph.graph[u][v]

parent[v] = u

return parent

def min\_key(key, mst\_set):

min\_val = float('inf')

min\_index = -1

for v in range(len(key)):

if key[v] < min\_val and not mst\_set[v]:

min\_val = key[v]

min\_index = v

return min\_index

def create\_graph\_from\_df(data):

num\_vertices = len(data)

graph = Graph(num\_vertices)

for i in range(num\_vertices):

for j in range(i+1, num\_vertices):

distance = haversine\_distance(data.iloc[i]['lat'], data.iloc[i]['lng'], data.iloc[j]['lat'], data.iloc[j]['lng'])

graph.add\_edge(i, j, distance)

return graph

def get\_user\_location():

g = geocoder.ip('me')

location = g.latlng

if location:

return location[0], location[1]

return None

def chennai\_restaurants\_page():

st.title("Chennai Restaurants")

image = Image.open(r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\Predictive analysis\PA Project\fac.png")

st.image(image, caption="Chennai Restaurants")

st.sidebar.header('Filter Options')

locations = df\_chennai['Location'].unique()

selected\_location = st.sidebar.selectbox('Select Location', locations)

cuisines = df\_chennai['Cuisine'].unique()

selected\_cuisine = st.sidebar.selectbox('Select Cuisine', cuisines)

max\_price = st.sidebar.number\_input('Maximum Price for 2', value=1000)

filtered\_df = df\_chennai[(df\_chennai['Location'] == selected\_location) &

(df\_chennai['Cuisine'] == selected\_cuisine) &

(df\_chennai['Price for 2'] <= max\_price)]

# Drop 'S.No', 'Zomato URL', and 'Location' columns if they exist

columns\_to\_drop = ['S.No', 'Zomato URL', 'Location']

filtered\_df = filtered\_df.drop(columns=[col for col in columns\_to\_drop if col in filtered\_df.columns])

# Create combined rating

filtered\_df['combined\_rating'] = filtered\_df[['Dining Rating', 'Delivery Rating']].mean(axis=1)

# Sort the restaurants by combined rating

sorted\_restaurants = filtered\_df.sort\_values(by='combined\_rating', ascending=False)

# Display filtered results

st.write('## Filtered Results')

st.write(sorted\_restaurants)

# Recommend restaurants based on user's location

def restaurant\_recommendation(user\_lat, user\_lon, data):

try:

# Create graph from dataframe

graph = create\_graph\_from\_df(data)

# Run Prim's algorithm to find MST

parent = prim\_mst(graph)

connected\_restaurants = []

for i in range(1, len(parent)):

if parent[i] == 0:

connected\_restaurants.append((data.iloc[i]['name'], haversine\_distance(user\_lat, user\_lon, data.iloc[i]['lat'], data.iloc[i]['lng'])))

# Sort restaurants by distance

connected\_restaurants.sort(key=lambda x: x[1])

return connected\_restaurants

except FileNotFoundError:

st.error("Error: CSV file not found.")

except ValueError:

st.error("Error: Invalid data in CSV file.")

def recommend\_nearby\_restaurants\_page():

st.title("Recommend Nearby Restaurants")

image = Image.open(r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\Predictive analysis\PA Project\fac.png")

st.image(image, caption="Restaurant Recommendation")

user\_location = get\_user\_location()

if user\_location:

user\_lat, user\_lon = user\_location

st.success(f"Your location is: Latitude: {user\_lat}, Longitude: {user\_lon}")

recommended\_restaurants = restaurant\_recommendation(user\_lat, user\_lon, df\_coimbatore)

if recommended\_restaurants:

st.write("Recommended restaurants near your location:")

for restaurant, distance in recommended\_restaurants:

st.write(f"- {restaurant} ({distance:.2f} km away)")

else:

st.warning("No restaurants found near your location.")

else:

st.warning("Unable to detect your location.")

def sort\_data\_by\_rating(data):

sorted\_data = data.sort\_values(by='star\_count', ascending=False, ignore\_index=True)

return sorted\_data.to\_dict('records')

def review\_rating\_page():

st.title("Review and Rating")

image = Image.open(r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\Predictive analysis\PA Project\fac.png")

st.image(image, caption="Review and Rating")

data = df\_coimbatore

sorted\_data = sort\_data\_by\_rating(data)

for restaurant in sorted\_data:

name = restaurant['name']

star\_count = restaurant['star\_count']

st.write(f"Name: {name}, Star Rating Out of 5: {star\_count}")

def get\_restaurant\_details(data, restaurant\_name):

selected\_restaurant = data[data['name'] == restaurant\_name]

return (selected\_restaurant.to\_dict('records')[0] if not selected\_restaurant.empty else None)

def select\_restaurant\_page():

st.title("Select Restaurant")

image = Image.open(r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\Predictive analysis\PA Project\fac.png")

st.image(image, caption="Select Restaurant")

data = df\_coimbatore

# Display a list of all restaurants

restaurants = data['name'].tolist()

selected\_restaurant = st.selectbox("Choose a restaurant", restaurants)

if selected\_restaurant:

details = get\_restaurant\_details(data, selected\_restaurant)

if details:

st.markdown(

f"""

<div>

<h3>{details['name']}</h3>

<p><strong>Address:</strong> {details['address']}</p>

<p><strong>Phone:</strong> {details['phone']}</p>

<p><strong>Primary Category:</strong> {details['primary\_category\_name']}</p>

<p><strong>Category:</strong> {details['category\_name']}</p>

<p><strong>Cuisine:</strong> {details['Cuisine']}</p>

<p><strong>Top Dishes:</strong> {''.join(details['Top Dishes'])}</p>

<p><strong>Price for 2:</strong> {details['Price for 2']}</p>

<p><strong>Dining Rating:</strong> {details['Dining Rating']} / 5</p>

<p><strong>Dining Rating Count:</strong> {details['Dining Rating Count']}</p>

<p><strong>Delivery Rating:</strong> {details['Delivery Rating']} / 5</p>

<p><strong>Delivery Rating Count:</strong> {details['Delivery Rating Count']}</p>

<p><strong>Features:</strong> {''.join(details['Features'])}</p>

</div>

""",

unsafe\_allow\_html=True

)

def find\_location\_page():

st.title("Find Your Location")

image = Image.open(r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\Predictive analysis\PA Project\fac.png")

st.image(image, caption="Find Your Location")

user\_location = get\_user\_location()

if user\_location:

user\_lat, user\_lon = user\_location

st.success(f"Your location is: Latitude: {user\_lat}, Longitude: {user\_lon}")

else:

st.warning("Unable to detect your location.")

# Open Google Maps

def open\_maps\_page():

st.title("Open Maps")

image = Image.open(r"C:\Users\srikr\Desktop\COLLEGE\Sem 4\Predictive analysis\PA Project\fac.png")

st.image(image, caption="Open Maps")

map\_url = ("https://www.google.com/maps/d/u/0/"

"embed?mid=1nqmVv1\_AgLWHO10knGd26DFtuUQ1Gfk&ehbc=2E312F")

if st.button("Open Google Maps"):

webbrowser.open(map\_url)

# --- Main Streamlit Application ---

def main():

st.set\_page\_config(page\_title="Restaurant Recommendation System", page\_icon=":fork\_and\_knife:")

pages = {

"Find Your Location": find\_location\_page,

"Open Maps": open\_maps\_page,

"Recommend Nearby Restaurants": recommend\_nearby\_restaurants\_page,

"Review and Rating": review\_rating\_page,

"Select Restaurant": select\_restaurant\_page,

"Chennai Restaurants": chennai\_restaurants\_page,

"Employee Route Optimization": employee\_route\_optimization\_page

}

st.sidebar.title("Navigation")

selection = st.sidebar.radio("Go to", list(pages.keys()))

pages[selection]()

# --- Employee Route Optimization ---

def employee\_route\_optimization\_page():

st.title("Employee Route Optimization")

uploaded\_file = st.file\_uploader("Choose an Excel file with employee data", type=["xlsx"])

if uploaded\_file is not None:

df\_employee = pd.read\_excel(uploaded\_file)

if 'Locations' in df\_employee.columns and 'Latitude' in df\_employee.columns and 'Longitude' in df\_employee.columns:

# Extract the location names, latitude, and longitude values

locations = df\_employee[['Locations', 'Latitude', 'Longitude']].values

num\_locations = len(locations)

# Initialize the distance matrix with infinities

distance\_matrix = np.full((num\_locations, num\_locations), np.inf)

for i in range(num\_locations):

for j in range(num\_locations):

if i != j:

loc1 = (locations[i][1], locations[i][2])

loc2 = (locations[j][1], locations[j][2])

distance\_matrix[i][j] = geodesic(loc1, loc2).kilometers

# Create a graph using NetworkX

G = nx.Graph()

for i, loc in enumerate(locations):

G.add\_node(i, pos=(loc[2], loc[1]), label=loc[0])

for i in range(num\_locations):

for j in range(num\_locations):

if i != j and distance\_matrix[i][j] < np.inf:

G.add\_edge(i, j, weight=distance\_matrix[i][j])

shortest\_paths = {}

for i in range(num\_locations):

for j in range(i + 1, num\_locations):

shortest\_path = nx.shortest\_path(G, source=i, target=j, weight='weight')

shortest\_distance = nx.shortest\_path\_length(G, source=i, target=j, weight='weight')

shortest\_paths[(i, j)] = {'path': shortest\_path, 'distance': shortest\_distance}

sorted\_paths = sorted(shortest\_paths.items(), key=lambda x: x[1]['distance'])

st.subheader("Top 5 Best Routes:")

for i, (path, info) in enumerate(sorted\_paths[:5], start=1):

start\_location = locations[path[0]][0]

end\_location = locations[path[1]][0]

st.write(f"Route {i}: {start\_location} -> {end\_location}, Distance: {info['distance']} km, Path: {info['path']}")

else:

st.error("The Excel file does not contain 'Locations', 'Latitude', and 'Longitude' columns.")

else:

st.info("Please upload an Excel file with employee data.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

The provided code is a complete implementation of the restaurant recommendation system and employee route optimization tool using Streamlit and NetworkX. The code reads data from Excel and CSV files, preprocesses the data, and creates interactive web pages for users to explore and interact with the system.

Restaurant recommendation page:

Employee route optimization page:

**Conclusion:**

The Restaurant Recommendation System and Employee Route Optimization tool developed using Streamlit and NetworkX provides an interactive and user-friendly interface for users to find nearby restaurants and optimize travel routes for employees. The system uses various algorithms such as the Haversine formula, Prim's algorithm, and Dijkstra's algorithm to calculate distances and find the shortest paths between different locations. The tool can be useful for businesses and individuals who want to optimize their travel routes and find the best restaurants in their area.