Laboratory Report Cover Sheet

SRM Institute of Science and Technology College of Engineering and Technology Department of Electronics and Communication Engineering
21ECE201J-Python and Scientific Python Lab **Fourth Semester, 2024-25 (even semester)**

Mini Project report

On

Flight Path Estimation

Team Members:

Register Number & Name:

1.RA2311004010380 - Abinandhan P

2.RA2311004010384 - Dhilip Raj T

3.RA2311004010387 - Nikhil V

Mark Split up			
Novelty of the project (5)			
Understanding (5)			
Execution &Demonstration (15)			
Report preparation (15)			
Total (40)			

REPORT VERIFICATION

Date :

Staff Name :

Signature :

FLIGHT PATH ESTIMATION

Introduction:

The Flight Path Estimation and Visualization Tool is a Python-based application designed to calculate and display potential flight routes between airports. It offers functionalities to fetch airport coordinates via an external API, calculate the great-circle distance, estimate basic flight metrics like time and fuel consumption, and visualize the path on an interactive map. The system requires user input for origin and destination airport IATA codes and a specific aircraft type to perform estimations based on simplified performance data. Users interact with the system through a console-based interface for input and receive feedback, including calculated metrics and a generated map file. Additionally, the project utilizes data visualization tools (Folium library) to provide a clear graphical representation of the flight path and key airport information. This project demonstrates the integration of fundamental programming concepts such as API interaction, geographical calculations, data handling using dictionaries, user input processing, and map-based data representation using Python libraries like requests, geopy, and folium. It serves as a practical model for applications involving geospatial data processing and visualization.

Software used:

- Python 3.x
- requests library (for API calls)
- geopy library (for distance calculation)
- folium library (for map generation)
- webbrowser library (to open the map)
- json library (for handling API errors)
- Aviationstack API (external data source)
- Jupyter Notebook

Theory/Explanation:

The Flight Path Estimation and Visualization Tool is a Python application that simulates basic flight planning calculations and visualization. It operates by taking user inputs for origin airport, destination airport (using 3-letter IATA codes), and an aircraft type. Key data points like airport coordinates (latitude, longitude) are fetched dynamically using the requests library to call the Aviationstack API. This external API provides access to real-world airport data.

The core calculation involves determining the shortest path between the two airport coordinates on the Earth's surface using the great_circle function from the geopy.distance module. This provides the distance in kilometers. Based on this distance and pre-defined, simplified performance characteristics (average cruise speed and fuel burn rate in kg/hour) stored in a Python dictionary (AIRCRAFT_PERFORMANCE) for the selected aircraft type, the script estimates the flight duration (Time = Distance / Speed) and total fuel consumption (Fuel = Time * Fuel Burn Rate).

User interaction occurs via the console for inputting the IATA codes and aircraft type. The script provides textual feedback on the console, displaying the fetched airport names, the calculated distance, and the estimated time and fuel.

A key output is an interactive HTML map generated using the folium library. This map visually plots the origin and destination airports as markers. Clicking these markers reveals pop-ups containing airport details and, for the destination, the estimated flight metrics. A line representing the great-circle path connects the two airports. The webbrowser library is used to automatically open this generated HTML file for immediate viewing.

This system demonstrates practical Python application development involving API integration, geospatial calculations, data manipulation, user interaction, and visualization, providing a foundation for more complex geospatial analysis tools. It highlights how different libraries can be combined to create a functional application, while acknowledging the simplifications made in the flight model (e.g., ignoring wind, climb/descent phases, actual air routes).

Programme:

```
import requests

import folium

from geopy.distance import great_circle

import json # Used for pretty printing API errors if needed

import webbrowser



# --- Configuration ---

AVIATIONSTACK_API_KEY = '33ecd3b0dbedda04539b663e40ed1fba' # Replace with your key

AVIATIONSTACK_BASE_URL = 'http://api.aviationstack.com/v1/'


# Simplified Aircraft Performance Data

# Speeds are approximate cruise speeds in km/h

# Fuel burn is approximate average cruise burn in kg/hour

AIRCRAFT_PERFORMANCE = {

    'A320': {'cruise_speed_kmh': 830, 'fuel_burn_kgh': 2500},

    'B737': {'cruise_speed_kmh': 830, 'fuel_burn_kgh': 2750},

    # Add more aircraft types as needed
```

```
}


# --- Functions ---


def get_airport_coords(api_key, iata_code):
    """Fetches airport coordinates from aviationstack API."""
    params = {
        'access_key': api_key,
        'iata_code': iata_code
    }
    try:
        response = requests.get(f"{AVIATIONSTACK_BASE_URL}airports", params=params)
        response.raise_for_status() # Raise an exception for bad status codes (4xx or 5xx)
        data = response.json()

        # Check if 'data' key exists and is a non-empty list
        if data.get('data') and isinstance(data['data'], list) and len(data['data']) > 0:
            # * FIXED: Access the first dictionary inside the list *
            airport_info = data['data'][0]

            # Now extract details from the airport_info dictionary
            lat = airport_info.get('latitude')
            lon = airport_info.get('longitude')
            name = airport_info.get('airport_name', 'N/A')
            city = airport_info.get('city', airport_info.get('city_iata_code', 'N/A'))
            country = airport_info.get('country_name', 'N/A')

            if lat is not None and lon is not None:
                print(f"Successfully retrieved coordinates for {iata_code}: {name}, {city},
{country}")
                return float(lat), float(lon), name
```

```
            else:
                print(f"Error: Latitude or Longitude missing for {iata_code} in API response.")
                print("Airport Info Received:", json.dumps(airport_info, indent=2))
                return None, None, None
        else:
                print(f"Error: No data found or unexpected format for airport IATA code: {iata_code}")
            print("API Response:", json.dumps(data, indent=2)) # Print response for debugging
            return None, None, None

    except requests.exceptions.RequestException as e:
        print(f"Error fetching data for {iata_code} from aviationstack: {e}")
        try:
            error_details = response.json()
            print("API Error Details:", json.dumps(error_details, indent=2))
        except Exception:
            pass
        return None, None, None
    except json.JSONDecodeError:
        print(f"Error: Could not decode JSON response from aviationstack for {iata_code}.")
        print("Raw Response Text:", response.text)
        return None, None, None
    except IndexError:
        print(f"Error: API returned an empty list for {iata_code}.")
        print("API Response:", json.dumps(data, indent=2))
        return None, None, None

def calculate_great_circle_distance(coord1, coord2):
    """Calculates Great Circle distance between two coordinates."""
    if coord1 and coord2:
        # Ensure coordinates are tuples of floats
```

```
        coord1_float = (float(coord1[0]), float(coord1[1]))
        coord2_float = (float(coord2[0]), float(coord2[1]))
        distance = great_circle(coord1_float, coord2_float).km
        return distance
    return 0

def estimate_flight_metrics(distance_km, cruise_speed_kmh, fuel_burn_kgh):
    """Estimates flight time and fuel burn."""
    if cruise_speed_kmh <= 0:
        print("Error: Cruise speed must be positive.")
        return 0, 0
    if distance_km <= 0:
         print("Warning: Distance is zero or negative.")
         return 0, 0

    estimated_time_hr = distance_km / cruise_speed_kmh
    estimated_fuel_kg = estimated_time_hr * fuel_burn_kgh
    return estimated_time_hr, estimated_fuel_kg

def create_flight_map(origin_coord, dest_coord, origin_iata, dest_iata,
                      origin_name, dest_name, distance_km, time_hr, fuel_kg):
    """Creates and saves a Folium map with the flight path."""
    if not origin_coord or not dest_coord:
        print("Error: Cannot create map without valid coordinates.")
        return

    # Ensure coordinates are tuples of floats for Folium
    origin_coord_float = (float(origin_coord[0]), float(origin_coord[1]))
    dest_coord_float = (float(dest_coord[0]), float(dest_coord[1]))

    # Create map centered roughly between origin and destination
```

```
map_center = [
    (origin_coord_float[0] + dest_coord_float[0]) / 2,
    (origin_coord_float[1] + dest_coord_float[1]) / 2
]
flight_map = folium.Map(location=map_center, zoom_start=4)

# Add markers for origin and destination
tooltip_origin = f"Origin: {origin_iata} ({origin_name})"
popup_origin = f"""
<b>Origin: {origin_iata}</b><br>
Name: {origin_name}<br>
Coords: ({origin_coord_float[0]:.4f}, {origin_coord_float[1]:.4f})
"""
folium.Marker(
    location=origin_coord_float,
    popup=popup_origin,
    tooltip=tooltip_origin,
    icon=folium.Icon(color='blue', icon='plane', prefix='fa')
).add_to(flight_map)

tooltip_dest = f"Destination: {dest_iata} ({dest_name})"
popup_dest = f"""
<b>Destination: {dest_iata}</b><br>
Name: {dest_name}<br>
Coords: ({dest_coord_float[0]:.4f}, {dest_coord_float[1]:.4f})<br>
---<br>
<b>Estimated Metrics:</b><br>
Distance: {distance_km:,.2f} km<br>
Time: {time_hr:.2f} hours<br>
Fuel: {fuel_kg:,.2f} kg
"""
```

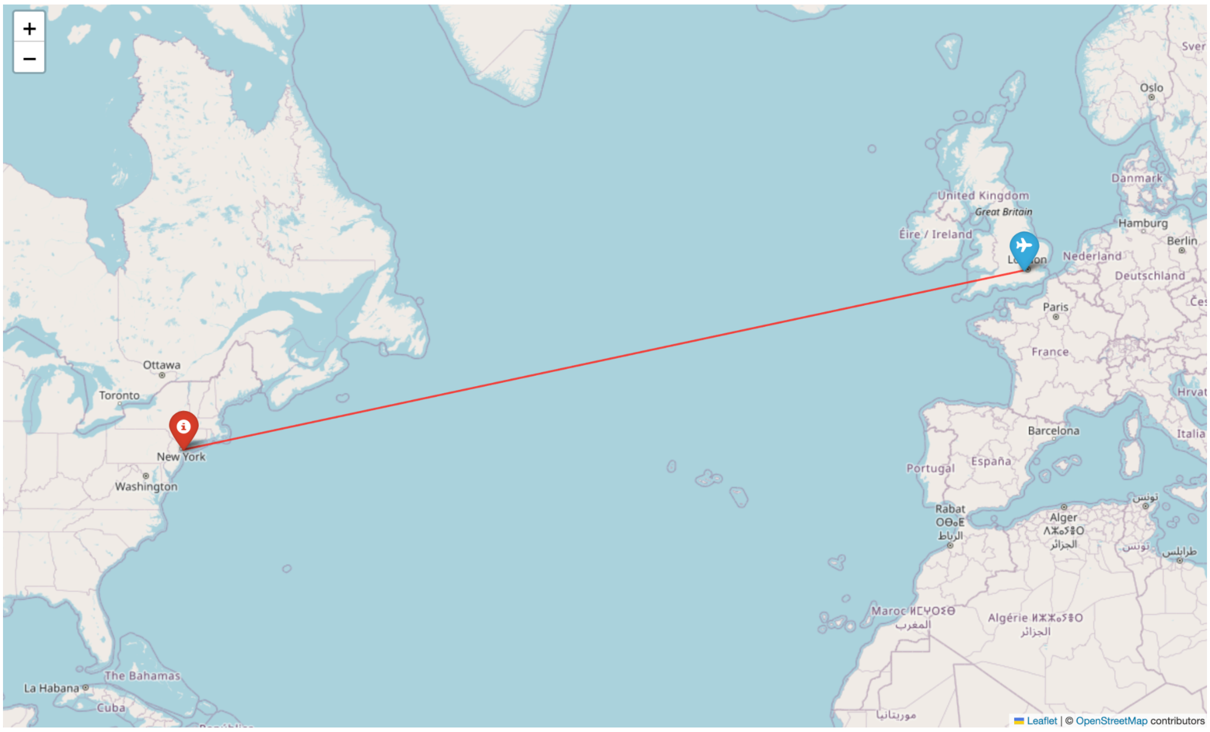
```
    folium.Marker(
        location=dest_coord_float,
        popup=popup_dest,
        tooltip=tooltip_dest,
        icon=folium.Icon(color='red', icon='info-sign')
    ).add_to(flight_map)

    # Add PolyLine for the flight path
    points = [origin_coord_float, dest_coord_float]
    folium.PolyLine(
        locations=points,
        color='red',
        weight=2,
        opacity=0.8,
        tooltip=f"Great Circle Path: {distance_km:,.2f} km"
    ).add_to(flight_map)

    # Save map to HTML file
    filename = f"flight_map_{origin_iata}to{dest_iata}.html"
    flight_map.save(filename)
    print(f"\nMap saved as {filename}")
    webbrowser.open(filename)


if _name_ == "_main_":
    print("Flight Path Estimator")
    print("-" * 20)

    if AVIATIONSTACK_API_KEY == 'YOUR_AVIATIONSTACK_API_KEY':
        print("\nError: Please replace 'YOUR_AVIATIONSTACK_API_KEY' with your actual key in the script.")
        exit()
```

```
    origin_iata = input("Enter Origin Airport IATA code (e.g., LHR): ").strip().upper()
    dest_iata = input("Enter Destination Airport IATA code (e.g., JFK): ").strip().upper()
    aircraft_type = input(f"Enter Aircraft Type ({', '.join(AIRCRAFT_PERFORMANCE.keys())}): ").strip().upper()

    if aircraft_type not in AIRCRAFT_PERFORMANCE:
        print(f"\nError: Aircraft type '{aircraft_type}' not found in performance data.")
        print(f"Available types: {', '.join(AIRCRAFT_PERFORMANCE.keys())}")
        exit()

    print("\nFetching airport data...")
    origin_lat, origin_lon, origin_name = get_airport_coords(AVIATIONSTACK_API_KEY, origin_iata)
    dest_lat, dest_lon, dest_name = get_airport_coords(AVIATIONSTACK_API_KEY, dest_iata)

    if origin_lat is None or dest_lat is None:
        print("\nCould not retrieve coordinates for one or both airports. Exiting.")
        exit()

    origin_coord = (origin_lat, origin_lon)
    dest_coord = (dest_lat, dest_lon)

    print("\nCalculating distance...")
    distance_km = calculate_great_circle_distance(origin_coord, dest_coord)
    if distance_km > 0:
        print(f"Great Circle Distance: {distance_km:,.2f} km")
    else:
        print("Could not calculate distance.")
        exit()
```

```
    print("\nEstimating flight metrics...")
    perf = AIRCRAFT_PERFORMANCE[aircraft_type]
    est_time_hr, est_fuel_kg = estimate_flight_metrics(
        distance_km,
        perf['cruise_speed_kmh'],
        perf['fuel_burn_kgh']
    )
    print(f"Aircraft Type: {aircraft_type}")
    print(f"  Estimated Cruise Speed: {perf['cruise_speed_kmh']} km/h")
    print(f"  Estimated Fuel Burn: {perf['fuel_burn_kgh']} kg/hr")
    print(f"Estimated Flight Time: {est_time_hr:.2f} hours")
    print(f"Estimated Total Fuel Burn: {est_fuel_kg:,.2f} kg")

    print("\nGenerating map...")
    create_flight_map(origin_coord, dest_coord, origin_iata, dest_iata,
                      origin_name, dest_name, distance_km, est_time_hr, est_fuel_kg)
    print("\nProcess Complete.")
```

Result and Screenshots:

```
Flight Path Estimator
--------------------
Enter Origin Airport IATA code (e.g., LHR):  LHR
Enter Destination Airport IATA code (e.g., JFK):  JFK
Enter Aircraft Type (A320, B737):  A320

Fetching airport data...
Successfully retrieved coordinates for LHR: Heathrow, LON, United Kingdom
Successfully retrieved coordinates for JFK: John F Kennedy International, NYC, United States

Calculating distance...
Great Circle Distance: 5,540.67 km

Estimating flight metrics...
Aircraft Type: A320
  Estimated Cruise Speed: 830 km/h
  Estimated Fuel Burn: 2500 kg/hr
Estimated Flight Time: 6.68 hours
Estimated Total Fuel Burn: 16,688.77 kg

Generating map...

Map saved as flight_map_LHRtoJFK.html

Process Complete.
```



Discussion of Results:

The Flight Path Estimation and Visualization Tool was successfully developed to take user inputs and fetch required airport data via the Aviationstack API. The system correctly calculates the great-circle distance using geopy and estimates flight time and fuel based on the selected aircraft's simplified performance data. Furthermore, using folium, a key visualization was created: an interactive HTML map showing the origin and destination airports as markers, connected by the calculated flight path. Popups on the markers provide relevant details, including the estimated metrics. The results from the console output and the interactive map provide clear feedback on the calculated distance and estimated flight parameters, making the system a useful tool for basic flight path visualization, suitable for extensions like adding more aircraft types or refining the flight model.

Conclusion:

The developed Flight Path Estimation and Visualization Tool effectively meets the core objective of calculating and displaying basic flight route information between two airports. The integration of external API calls, geographical calculations, and interactive map visualization enhances data interpretation by clearly showing the route and associated estimates. The system is designed with clear functions and user interaction via the console, providing a solid foundation for further enhancements such as incorporating wind data, using more complex flight phase models (climb, cruise, descent), or allowing data export.