Title: Airline Flight Delays and Weather Analysis

Author: Zhijie Xu

Date: April 20th

GitHub Repository Link: https://github.com/Charlie-ZHIJIE/SI507_Project

Video link: https://drive.google.com/file/d/1Np4nfa_6CGRzOc9MkB7jgt4pg48wZxGV/view?usp=sharing

Introduction

This project aims to analyze the relationship between airline flight delays and weather conditions by combining flight delay data with historical and live weather data. The application will allow users to explore correlations between weather patterns and flight delays through an interactive network graph, as well as view average flight delay times and weather conditions for specific airports.

Project code

Readme file containing special instructions for running your code.
Required Python Packages: os, time, datetime, json, flask, mplcursors, requests, network, matplotlib, diskcache.

Data Sources

Flight Delay Data and Live weather Data(challenge score 4)

Origin: https://developer.flightstats.com/
Documentation: https://developer.flightstats.com/api-docs/delayindex/v1/
Format: JSON
Access method: It needs to create an application for using data from the data source. It needs a AppID and AppKey to get the useful data.
Number of records available: huge, it provides delay information and weather information in every airport in real-time.
Number of records retrieved: for each time, it provides the current delay information and weather for one airport.
Description of records:
For delay information, several important fields are: 'delayindex 'and 'airport', using them to specify the airport. And get the delay information. For weather information, several important fields are: 'weather', and 'airport', using them to specify the airport and get the weather information.
Cache using: Yes

Historical Weather weather Data(challenge score 4 + 2)

Origin: https://www.ncei.noaa.gov
Documentation: https://www.ncei.noaa.gov/access/
The NCEI Data Access application offers a wide variety of download and subsetting options for a growing collection of environmental data. While current offerings are limited primarily to

weather and climate information, the application has a data-agnostic infrastructure designed to accommodate a broad spectrum of observation formats from across science disciplines.
Format: JSON
Access method: It needs AppKey to get useful data.
Number of records available: 1480000
Number of records retrieved: large than 1000
Description of records:
For historical weather information, several important fields are: 'station_id', 'start_date ' and 'end_date, using them to specify the station number for each airport and get the historical weather data between the start date and end date they have been specific.
Cache using: Yes

Airport coordinates Data(challenge score 2)

Origin: https://ourairports.com/data/airports.csv
Source: OurAirports
Purpose: To convert airport names to their respective coordinates.
Format: CSV
Access method: To access this dataset, download the CSV file from the provided source and load it into your desired programming environment or software.
Number of records available: large than 1000
Number of records retrieved: large than 1000
Description of records:
The dataset contains fields for airport names and their corresponding coordinates.
Fields: Airport Name, Latitude, Longitude

Mapping airport to weather station:

Origin: https://www.flightsfrom.com/top-100-airports-in-north-america &
https://www.ncei.noaa.gov
Format: JSON file
Access method: load it from local JSON file
Number of records available: 100
Number of records retrieved: 100
Description of records:
Convert the airport to the nearest weather station.

Data Access and Storage

The raw data from all three sources will be cached to ensure efficient, save local and responsible access. A separate JSON file will store non-dynamic data, such as airport codes and city locations. A script will be used to populate the cache and update it periodically.=

Data Structure

The data is organized into a network graph with the following components:
Vertices: Airports (nodes representing the airports)
Edges: Flight connections (weighted by average delay time)

Vertices attributes: Weather conditions (historical or live, depending on user preference)
A Python file (graph_constructor.py) constructs the network graph from the stored data using classes, and a JSON file (graph.json) stores the graph structure for use in the application.

The picture above is the screenshot of the data.
That graph uses portly to show this graph and we can easily analyze the relation between weather and flight delay. What's more, we can find the minimum path to find the most efficient way to buy airline tickets.

Interaction and Presentation

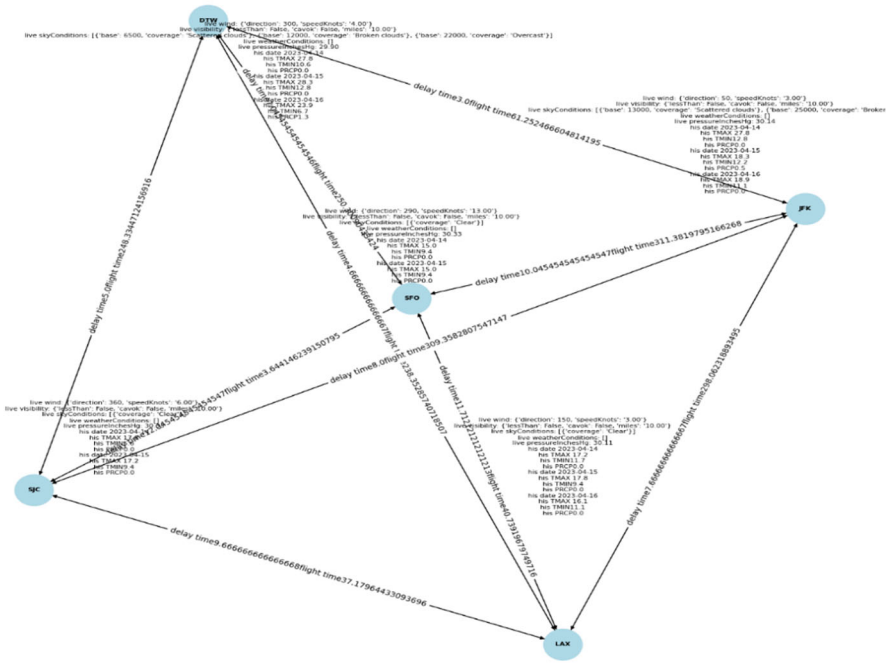
Web Application (Flask)

The Flask web application allows users to enter an airport code and view the average flight delay time and weather conditions for the selected airport.
What's more, the graph picture also shows the Flask web Application.
In this Flask web application, you have built a tool that allows users to input one or multiple airport codes, and then retrieves and displays the average flight delay time and weather conditions for the selected airports. The application also generates a network graph illustrating the relationships between the airports when more than one airport code is provided.
The application consists of several routes and functions:
The root route ("/") renders the 'index.html' template, which is used to display the form where users can input the airport codes.
The '/get_data' route is triggered when the form is submitted. This route fetches the data for the selected airports using several functions:
load_airport_station_mapping() get_historical_weather_data() get_flight_delay_data() get_flight_weatherdata_andForecast()
If more than one airport code is provided, the application creates a network graph using the 'create_network_graph()' function, and then draws the graph using the 'draw_graph()' function. The resulting image is stored as 'graph_filename'.
The results are then rendered as an HTML page using the 'render_template_string()' function. This page displays the flight delay data, historical weather data, live weather data, and the graph image when applicable.
The '/graph_image/' route serves the graph image generated by the application. It uses the 'send_from_directory()' function to return the image file from the server to the user.
Overall, this Flask web application provides users with a convenient way to visualize flight delay and weather information for selected airports, making it easier to understand potential impacts on flight schedules.

Interactive Network Graph (Plotly with Flask)

The Plotly network graph displays weather and flight delay relationships in an interactive way, allowing users to zoom in and explore the correlations between weather and flight delays.

The picture below is the graph generated by plotly, the vertices is airport with weather conditions including current and historical. The edge is the delay weight, the high value of the delay weight represents more flights have been delayed or the flights have been delayed more time.



Command-Line Interface

The command-line interface provides a simple way for users to view airport information, flight delay times, and weather conditions from the terminal by running the Python script.

```
Welcome to SI507 final project
Enter airports(at least two airports Example:LAX,SFO) LAX,SFO,JFK,SJC,DTW
historical_weather_data:
{'LAX': [{'DATE': '2023-04-13', 'STATION': 'USW00023174', 'PRCP_ATTRIBUTES': ',,W,2400', 'TMAX': '15.6', 'TMAX_ATTRIBUTES': ',,W',
flight_delay_data:
{'LAX': {'flights': 110, 'observations': 101, 'canceled': 6, 'onTime': 67, 'delayed15': 8, 'delayed30': 6, 'delayed45': 14, 'averag
flight_weatherdata_andForecast:
{'LAX': {'conditions': {'wind': {'direction': 250, 'directionIsVariable': False, 'speedKnots': '11.00'}, 'visibility': {'lessThan':
```

Add Flight Time and help Analysis

Add Flight time to Edge, Make the graph has more information. Customer can find an efficient path from source to destination. Below is the Graph example:

Conclusion

The Airline Flight Delays and Weather Analysis project demonstrates the ability to access data from multiple web APIs that require authentication, use caching to efficiently and responsibly access data, analyze and process data using advanced data structures, and present data to the user in various ways using Flask, Plotly, and a command-line interface.