FLIGHTS ANALYTICS DASHBOARD

PROJECT: Airline Flight Delay Dataset: Filtering and Grouping Operations.

PROBLEM STATEMENT: An airline company maintains flight records including departure/arrival times and delays. The dataset contains delays caused by weather, airline operations, or technical issues. The HR Analytics team needs to analyze delays by filtering and grouping operations using Pandas (no visualization).

NAME: AISHWARYA S (4GW23CI003)

EMAIL: aishwarya161005@gmail.com

DATE: 01-09-2025

INDEX PAGE

1.INTODUCTION	**3**
2.PROJECT OVERVIEW	**3**
3.DATASET DESCRIPTION	**3**
3.1. FILE DETAILS.	
4.TECHNOLOGY USAGE IN THE PROJECT	**4**
4.1. FLASK USAGE. 4.2. FORNT-END BASED TECHNOLOGIES. 4.3. THUNDERCLIENT USAGE.	
5.FEATURES IMPLEMENTED	**5-6**
5.1. DATA CLEANING AND EXPLORATION. 5.2. FILTERING OPERATIONS. 5.3. GROUPING AND AGGREGRATING OPERATIONS. 5.4. EXPORTING PROCCESSED RESULTS.	
6. TECHNICAL OVERVIEW	**6-7**
6.1. FRONT-END LAYER 6.2. BACKEND LAYER 6.3. DATASET LAYER	
7. SYSTEM FEATURES	**7-8**
8. UML DIAGRAMS	**9**
9. DESIGN	**10-15**
9.1. PROJECT STRUCTURE 9.2. CODE AND IMPLEMENTATIONS	
10.OUTPUTS SCREENSHOTS	**16-24**
11.CONCLUSION & BIBLIOGRAPHY	**25**

1.INTRODUCTION

The Flight Analytics Dashboard project aims to provide airline analysts, airport authorities, and passengers with a centralized platform for analyzing flight performance. The system will enable users to:

- Understand patterns of delays and cancellations.
- Identify top-performing airlines and airports.
- Track seasonal and weather-related impacts on flights.
- Make data-driven decisions to improve efficiency.

2.PROJECT OVERVIEW

The **Flight Analytics Dashboard** is a web-based application designed to analyze and visualize flight data, focusing on delays, cancellations, and overall flight performance. With the rapid increase in air travel, there is a growing need for tools that can help airlines, airports, and passengers gain insights into flight operations. This project aims to bridge that gap by providing an interactive and user-friendly dashboard.

The system collects and processes historical and real-time flight datasets, enabling users to explore flight patterns and trends. By leveraging powerful data analytics libraries such as **Pandas, Flask** the dashboard presents complex flight information in the form of **summary statistics/tables** with a **bootstrap-based front end.**

3.DATASET DESCRIPTION

The dataset used in this project is named **flights.csv**, which contains detailed records of flight operations. It is a **comma-separated values (CSV) file** that is loaded into the system using the **Pandas library** in Python for further analysis and visualization.

3.1 File Details

- **File Name:** flights.csv
- **Format:** CSV (Comma-Separated Values)
- **Loading Method:** pandas.read_csv("flights.csv")
- **Size:** The dataset may range from thousands to millions of rows depending on the records included.
- **Usage:** Primary data source for all analytics in the Flight Analytics Dashboard.

4. TECHNOLOGY USAGE IN THE PROJECT

4.1. FLASK Usage

Flask is a **lightweight Python web framework** used as the backend for the **Flight Analytics Dashboard**. It provides routing, request handling, and integration with data analysis modules.

Role of Flask in the Project:

- Acts as the **backend server** to handle requests and responses.
- Connects the **frontend (dashboard UI)** with the **data analytics logic (Pandas)**.
- Provides **REST APIs** for data access (e.g., flight delays, cancellations, and summary statistics).
- Manages **file uploads** (CSV datasets like flights.csv) and integrates them with Pandas for analysis

4.2. Frontend Technologies Used

Bootstrap: Bootstrap is an open-source CSS framework that provides pre-styled components, grid systems, and responsive layouts. In this project, Bootstrap was used to design a **clean and responsive dashboard interface**.

Jinja2 Templates: Jinja2 is the default templating engine for Flask. It allows embedding Python-like expressions directly inside **HTML**

4.3. Thunder Client Usage

Thunder Client is a **Visual Studio Code (VS Code) extension** used for testing APIs in a simple and lightweight way (similar to Postman).

Role of Thunder Client in the Project:

- Helps in **testing REST APIs** built with Flask without needing to run the entire frontend.
- Allows sending **GET, POST, PUT, DELETE requests** to check API endpoints.
- Provides an easy interface to check responses in **JSON/HTML format**.
- Validates the API output for uploaded datasets (e.g., verifying if /api/delays returns the correct delay statistics)
- Speeds up debugging and development by testing API endpoints directly inside **VS Code**.

5.FEATURES IMPLEMENTED

As part of the Flight Analytics Dashboard, the following data processing and analysis tasks were carried out on the dataset flights.csv using Python (Pandas):

5.1. Data Cleaning and Exploration

- The dataset was checked for missing values and duplicate records to ensure accuracy of analysis.
- Date and time-related columns (such as Scheduled_Departure, Actual_Departure, Scheduled_Arrival, Actual_Arrival) were converted into standard datetime formats for consistency.
- Outliers and invalid records (e.g., negative delay times) were handled appropriately to maintain data quality.

5.2. Filtering Operations

- Extracted all flights delayed by more than 30 minutes to study significant delays.
- Implemented filters to analyze flights operated by specific airlines.
- Retrieved flight records based on departure airport to study airport-specific performance.

5.3. Grouping and Aggregation Operations

- By Airline: Computed average delay times and total number of flights for each airline to compare performance.
- By Origin Airport: Calculated total flights and percentage of delayed flights for each airport.
- By Date: Derived daily average delay trends to analyze seasonal and date-wise variations.
- Top Routes: Identified the Top 5 Origin–Destination routes with the highest average delay, providing insight into problematic routes.

5.4. Exporting Processed Results

- Final summarized datasets were exported into CSV files for further use in reporting and visualization.
- These CSV summaries act as cleaned and preprocessed datasets that can be directly loaded into the dashboard for generating charts and reports.

6. TECHNICAL OVERVIEW

The **Flight Analytics Dashboard** is developed using a **modular architecture** that integrates the frontend, backend, and dataset seamlessly. Each component has a distinct role in the overall system.

6.1. Frontend (User Interface Layer)

- Built using **HTML, CSS, Bootstrap, and Jinja2 templates**.
- Provides a **dashboard-style layout** with:
    - Sidebar navigation (Dashboard, Upload Data, Reports).
    - Data upload form for flights.csv.
    - Tables and charts to display processed results.
- **Bootstrap** ensures the interface is **responsive** across devices.
- **Jinja2 templating** allows dynamic integration of Python/Flask results into HTML pages (e.g., showing filtered flight data or grouped summaries).
- Users interact with the system through this layer.

6.2. Backend (Application & Processing Layer)

- Implemented using **Python** and the **Flask web framework**.
- Key responsibilities:
  - Handle **file uploads** (accept flights.csv).
  - Process dataset using **Pandas** (cleaning, filtering, grouping, aggregation).
  - Integrate business logic such as identifying delayed flights, top routes, and average delays.
- **Thunder Client** (VS Code extension) was used to test and validate the backend APIs.
- Backend acts as the **bridge between dataset and frontend**.

6.3. Dataset (Data Layer)

- Dataset used: **flights.csv**
- Format: **CSV file** loaded using **Pandas**.
- Contains flight records with attributes such as:
  - Flight ID, Airline, Origin Airport, Destination Airport, Departure Time, Arrival Time, Delay, etc.
- Dataset preprocessing steps:
  - Check and handle missing/duplicate records.
  - Extract meaningful subsets (e.g., flights delayed > 30 minutes).
- Processed outputs are saved as **summary CSV files**, which can be reused and downloaded by the user.

7. SYSTEM FEATURES

7.1 Airline Summary

- Shows average delay, total flights by airline.
- Accessible via /summary/airline/table and /summary/airline/json.

7.2 Airport Summary

- Summarizes flights from and to each airport.
- Accessible via /summary/airport/table.

7.3 Route Summary

- Groups flights by origin-destination route.
- Accessible via /summary/route/table.

7.4 Date Summary

- Shows **daily average delay** grouped by date.
- Accessible via /summary/date/table.

7.5 Flights by Airport

- List flights from a specific airport.
- Available in both JSON and table format.

7.6 Flights by Airline

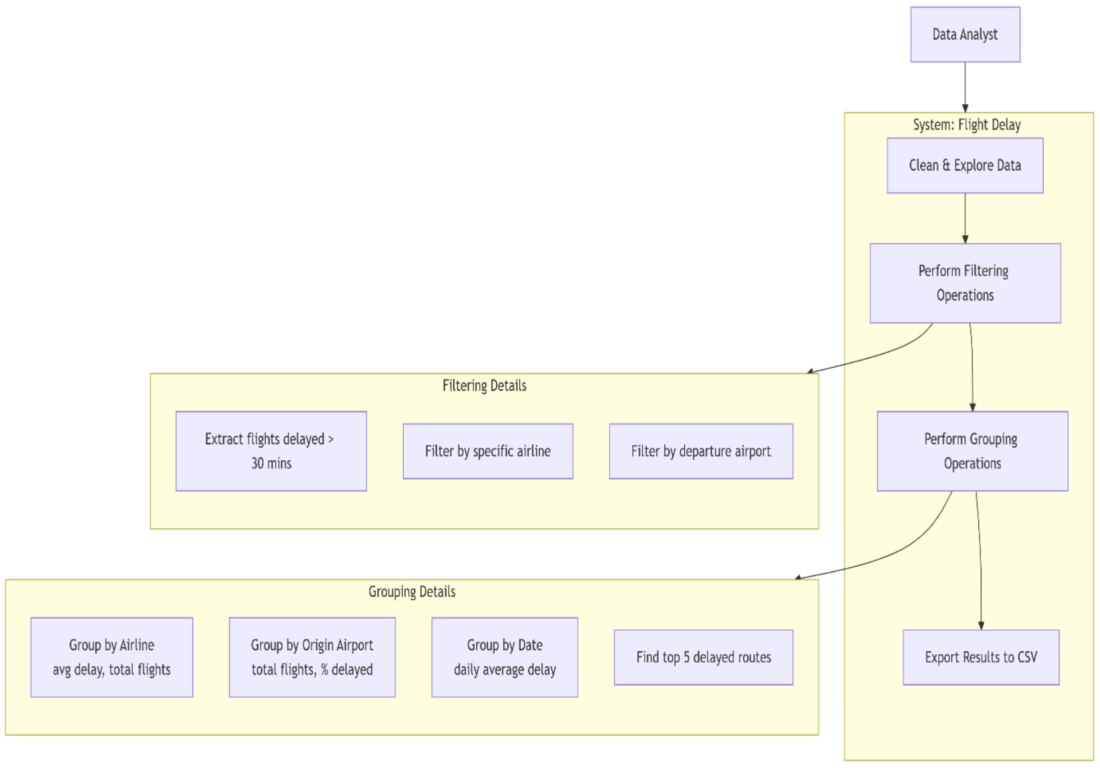
- Lists all flights operated by a specific airline.
- Accessible via:
  - /flights/from_airline/<airline>/table

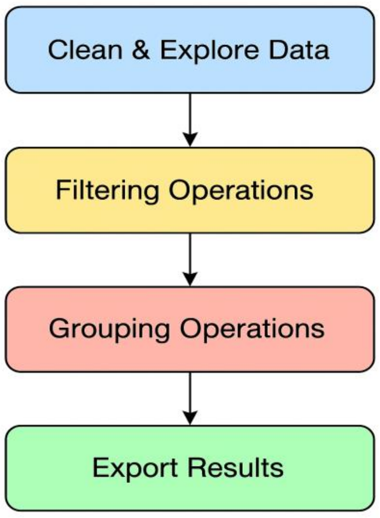
7.7 Delayed Flights

- Displays flights with delays beyond the scheduled time.
- Accessible via:
  - /flights/delayed/table
  - /flights/delayed/json

8.UML DIAGRAMS:

USE CASE AND FLOW DIAGRAM


Data Analyst
System: Flight Delay
Clean & Explore Data
Perform Filtering Operations
Perform Grouping Operations
Export Results to CSV
Filtering Details
Extract flights delayed > 30 mins
Filter by specific airline
Filter by departure airport
Grouping Details
Group by Airline avg delay, total flights
Group by Origin Airport total flights, % delayed
Group by Date daily average delay
Find top 5 delayed routes



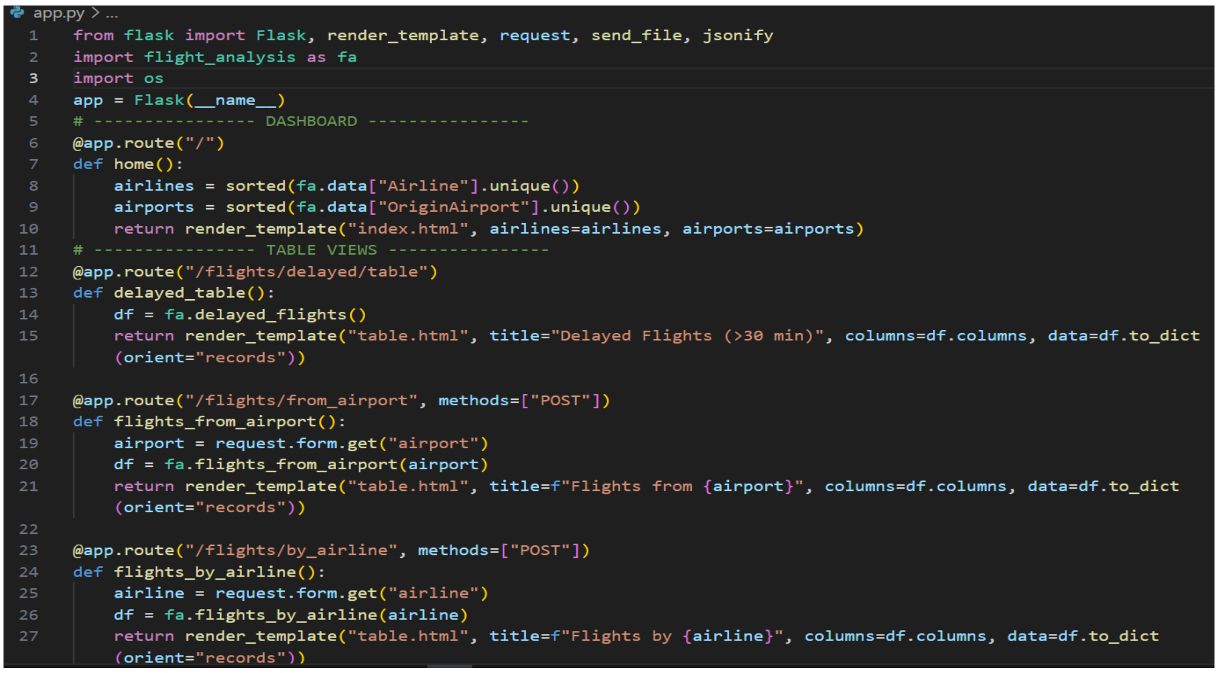
9.DESIGN

9.1. PROJECT STRUCTURE:

```
project/
|
├── app.py
├── flight_analysis.py
├── data/
|    └── flights.csv
└── templates/
     ├── index.html
     └── table.html
```

9.2. CODE AND IMPLEMENTATION

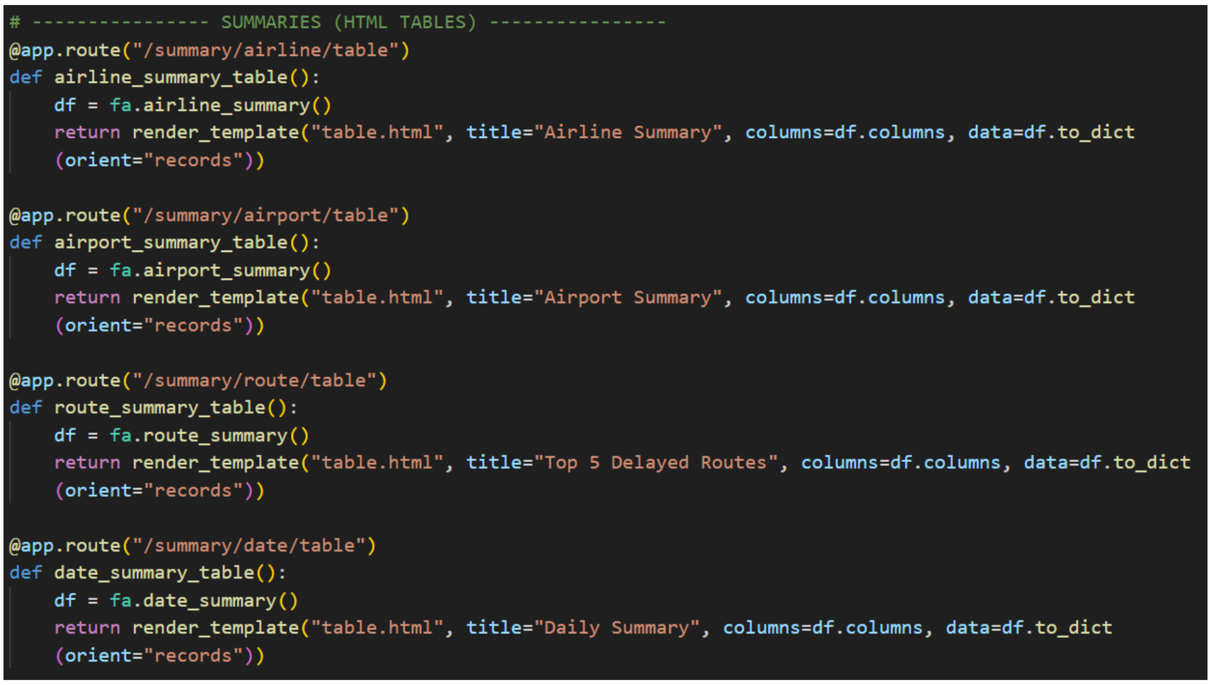
APP.PY



```python
from flask import Flask, render_template, request, send_file, jsonify
import flight_analysis as fa
import os
app = Flask(__name__)
# ---------------- DASHBOARD ----------------
@app.route("/")
def home():
    airlines = sorted(fa.data["Airline"].unique())
    airports = sorted(fa.data["OriginAirport"].unique())
    return render_template("index.html", airlines=airlines, airports=airports)
# ---------------- TABLE VIEWS ----------------
@app.route("/flights/delayed/table")
def delayed_table():
    df = fa.delayed_flights()
    return render_template("table.html", title="Delayed Flights (>30 min)", columns=df.columns, data=df.to_dict
    (orient="records"))

@app.route("/flights/from_airport", methods=["POST"])
def flights_from_airport():
    airport = request.form.get("airport")
    df = fa.flights_from_airport(airport)
    return render_template("table.html", title=f"Flights from {airport}", columns=df.columns, data=df.to_dict
    (orient="records"))

@app.route("/flights/by_airline", methods=["POST"])
def flights_by_airline():
    airline = request.form.get("airline")
    df = fa.flights_by_airline(airline)
    return render_template("table.html", title=f"Flights by {airline}", columns=df.columns, data=df.to_dict
    (orient="records"))
```

This code defines a **Flask web application** for a Flight Analytics Dashboard. It:

- Loads airlines and airports into the homepage (index.html).
- Provides routes to display tables for:
  - **Delayed flights** (>30 min).
  - **Flights from a specific airport** (via form input).
  - **Flights by a specific airline** (via form input).



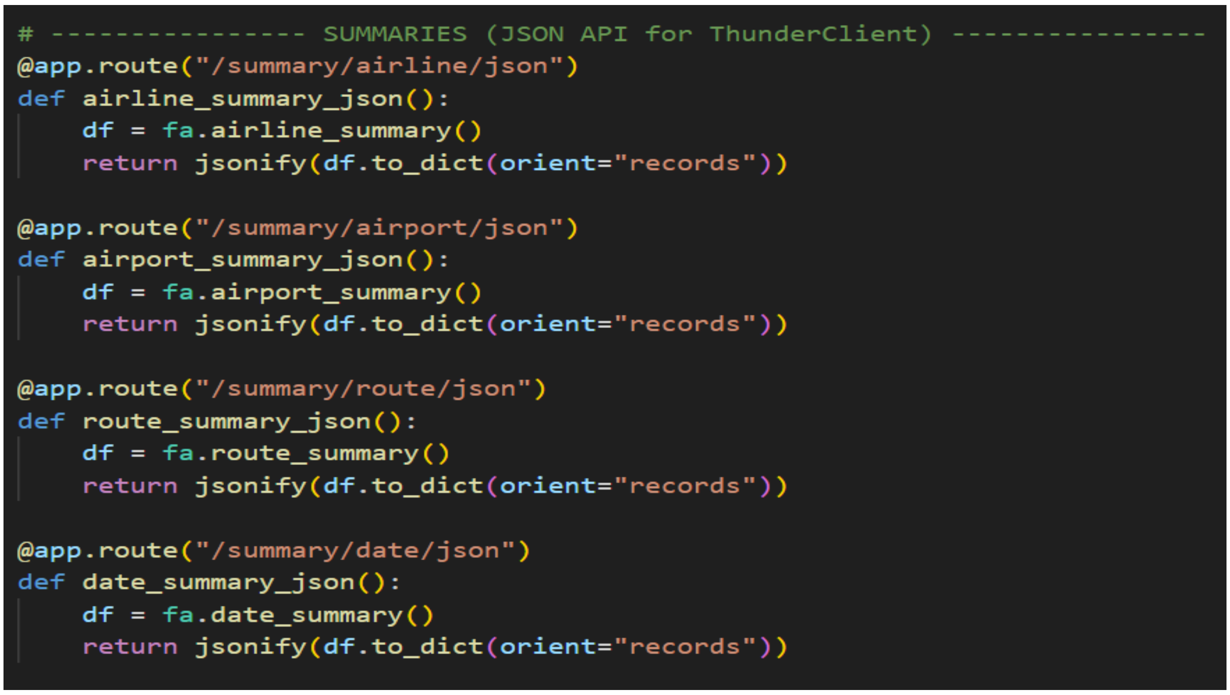
```
# --------------- SUMMARIES (HTML TABLES) ---------------
@app.route("/summary/airline/table")
def airline_summary_table():
    df = fa.airline_summary()
    return render_template("table.html", title="Airline Summary", columns=df.columns, data=df.to_dict
    (orient="records"))

@app.route("/summary/airport/table")
def airport_summary_table():
    df = fa.airport_summary()
    return render_template("table.html", title="Airport Summary", columns=df.columns, data=df.to_dict
    (orient="records"))

@app.route("/summary/route/table")
def route_summary_table():
    df = fa.route_summary()
    return render_template("table.html", title="Top 5 Delayed Routes", columns=df.columns, data=df.to_dict
    (orient="records"))

@app.route("/summary/date/table")
def date_summary_table():
    df = fa.date_summary()
    return render_template("table.html", title="Daily Summary", columns=df.columns, data=df.to_dict
    (orient="records"))
```

This code defines **Flask routes to display summary tables** in the Flight Analytics Dashboard. It:

- Shows **airline-wise summary**, **airport-wise summary**, **top 5 delayed routes**, and **daily flight summary**.
- Uses functions from flight_analysis.py to generate the data.

This code defines **API routes** that return **flight summary data in JSON format**. It:

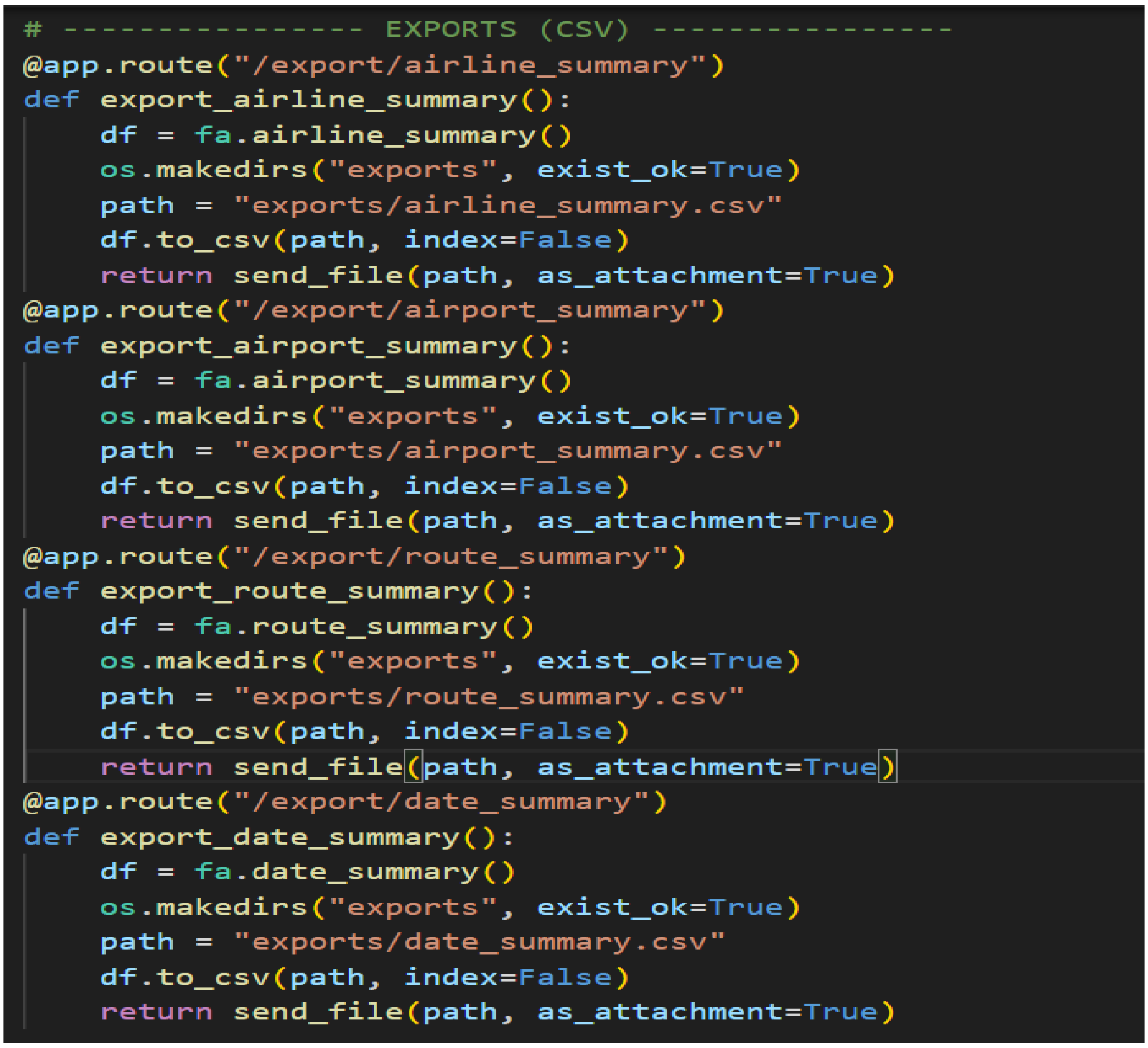
- Provides airline, airport, top delayed routes, and daily summaries as **JSON responses**.
- Uses flight_analysis.py functions to generate the data.



```python
# ---------------- SUMMARIES (JSON API for ThunderClient) ----------------
@app.route("/summary/airline/json")
def airline_summary_json():
    df = fa.airline_summary()
    return jsonify(df.to_dict(orient="records"))

@app.route("/summary/airport/json")
def airport_summary_json():
    df = fa.airport_summary()
    return jsonify(df.to_dict(orient="records"))

@app.route("/summary/route/json")
def route_summary_json():
    df = fa.route_summary()
    return jsonify(df.to_dict(orient="records"))

@app.route("/summary/date/json")
def date_summary_json():
    df = fa.date_summary()
    return jsonify(df.to_dict(orient="records"))
```

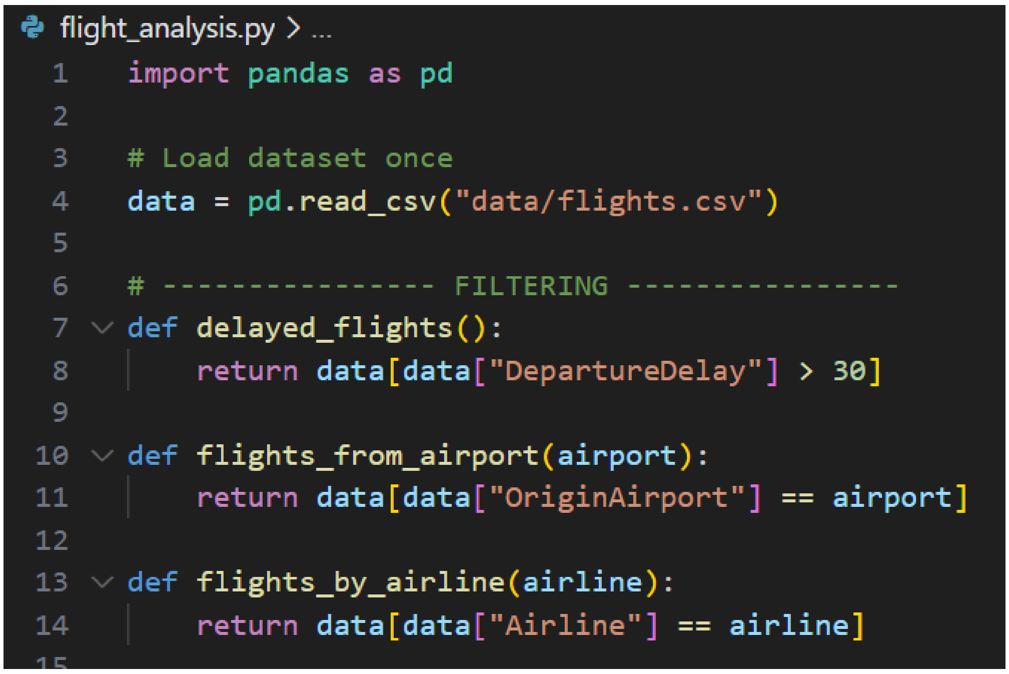
This code defines **Flask routes to export flight summary data as CSV files**. It:

- Generates airline, airport, top delayed routes, and daily summaries using flight_analysis.py.
- Saves the CSV files in an **exports** folder (created if it doesn't exist).
- Sends the CSV files to the user as **downloadable attachments** via the browser.



```python
# ---------------- EXPORTS (CSV) ----------------
@app.route("/export/airline_summary")
def export_airline_summary():
    df = fa.airline_summary()
    os.makedirs("exports", exist_ok=True)
    path = "exports/airline_summary.csv"
    df.to_csv(path, index=False)
    return send_file(path, as_attachment=True)
@app.route("/export/airport_summary")
def export_airport_summary():
    df = fa.airport_summary()
    os.makedirs("exports", exist_ok=True)
    path = "exports/airport_summary.csv"
    df.to_csv(path, index=False)
    return send_file(path, as_attachment=True)
@app.route("/export/route_summary")
def export_route_summary():
    df = fa.route_summary()
    os.makedirs("exports", exist_ok=True)
    path = "exports/route_summary.csv"
    df.to_csv(path, index=False)
    return send_file(path, as_attachment=True)
@app.route("/export/date_summary")
def export_date_summary():
    df = fa.date_summary()
    os.makedirs("exports", exist_ok=True)
    path = "exports/date_summary.csv"
    df.to_csv(path, index=False)
    return send_file(path, as_attachment=True)
```

FLIGHTS_ANALYSIS.PY

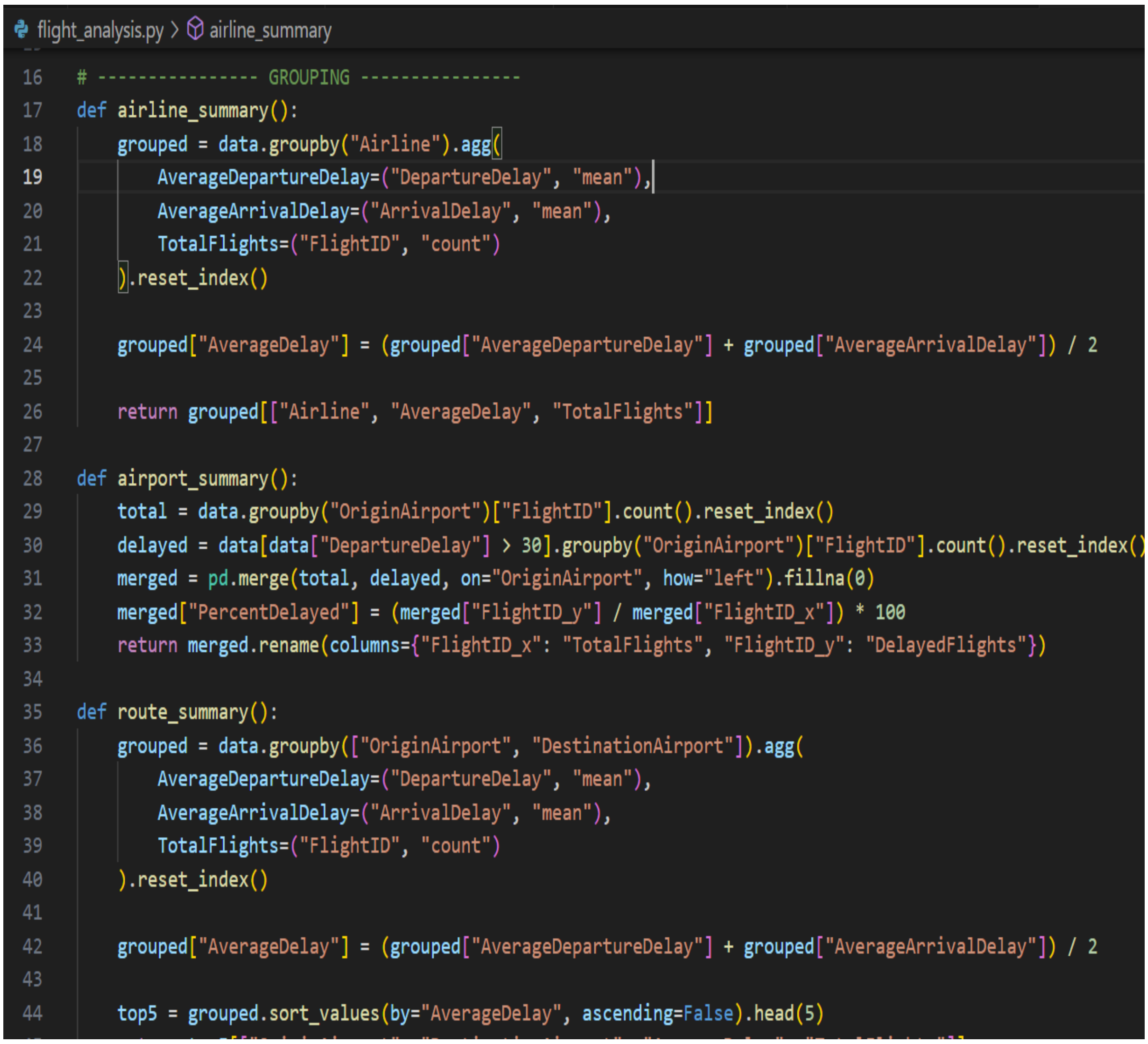


flight_analysis.py > ...

```python
import pandas as pd

# Load dataset once
data = pd.read_csv("data/flights.csv")

# ---------------- FILTERING ----------------
def delayed_flights():
    return data[data["DepartureDelay"] > 30]

def flights_from_airport(airport):
    return data[data["OriginAirport"] == airport]

def flights_by_airline(airline):
    return data[data["Airline"] == airline]
```

This code defines **data filtering functions** using **Pandas** on the flights.csv dataset. It:

- Loads the dataset once into a DataFrame.
- Provides functions to extract:
  - Flights **delayed more than 30 minutes**.
  - Flights **departing from a specific airport**.
  - Flights **operated by a specific airline**.



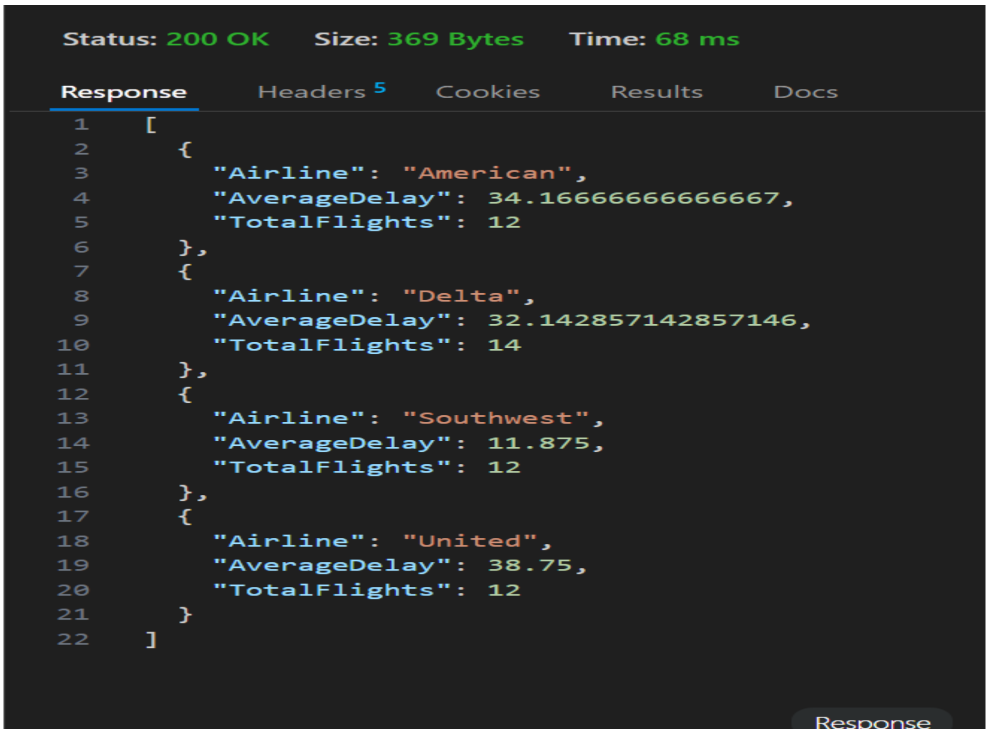
```python
# --------------- GROUPING ---------------
def airline_summary():
    grouped = data.groupby("Airline").agg(
        AverageDepartureDelay=("DepartureDelay", "mean"),
        AverageArrivalDelay=("ArrivalDelay", "mean"),
        TotalFlights=("FlightID", "count")
    ).reset_index()

    grouped["AverageDelay"] = (grouped["AverageDepartureDelay"] + grouped["AverageArrivalDelay"]) / 2

    return grouped[["Airline", "AverageDelay", "TotalFlights"]]

def airport_summary():
    total = data.groupby("OriginAirport")["FlightID"].count().reset_index()
    delayed = data[data["DepartureDelay"] > 30].groupby("OriginAirport")["FlightID"].count().reset_index()
    merged = pd.merge(total, delayed, on="OriginAirport", how="left").fillna(0)
    merged["PercentDelayed"] = (merged["FlightID_y"] / merged["FlightID_x"]) * 100
    return merged.rename(columns={"FlightID_x": "TotalFlights", "FlightID_y": "DelayedFlights"})

def route_summary():
    grouped = data.groupby(["OriginAirport", "DestinationAirport"]).agg(
        AverageDepartureDelay=("DepartureDelay", "mean"),
        AverageArrivalDelay=("ArrivalDelay", "mean"),
        TotalFlights=("FlightID", "count")
    ).reset_index()

    grouped["AverageDelay"] = (grouped["AverageDepartureDelay"] + grouped["AverageArrivalDelay"]) / 2

    top5 = grouped.sort_values(by="AverageDelay", ascending=False).head(5)
```

This code defines **data aggregation functions** using **Pandas** on the flights.csv dataset. It:

- **airline_summary()** → Calculates average departure & arrival delays and total flights per airline.
- **airport_summary()** → Calculates total flights, delayed flights, and percentage of delays per origin airport.
- **route_summary()** → Calculates average delays and total flights per route and returns the **top 5 routes with highest average delay**.
- **date_summary()** → Aggregates flight data **by date** to calculate metrics such as:
  - **->** Total flights per day
  - **->** Average departure and arrival delays per day.

10.OUTPUTS (SCREENSHOTS OF BOTH JSON AND BROWSER)

http://127.0.0.1:5000/summary/airline/json

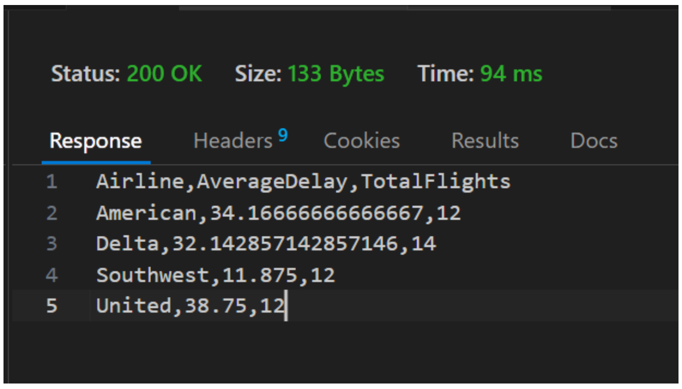


```
[
  {
    "Airline": "American",
    "AverageDelay": 34.166666666666667,
    "TotalFlights": 12
  },
  {
    "Airline": "Delta",
    "AverageDelay": 32.142857142857146,
    "TotalFlights": 14
  },
  {
    "Airline": "Southwest",
    "AverageDelay": 11.875,
    "TotalFlights": 12
  },
  {
    "Airline": "United",
    "AverageDelay": 38.75,
    "TotalFlights": 12
  }
]
```

The output represents average flight delays by airline.
It shows, for each airline:

- The average delay across all flights.
- The total number of flights considered.

http://127.0.0.1:5000/export/airline_summary

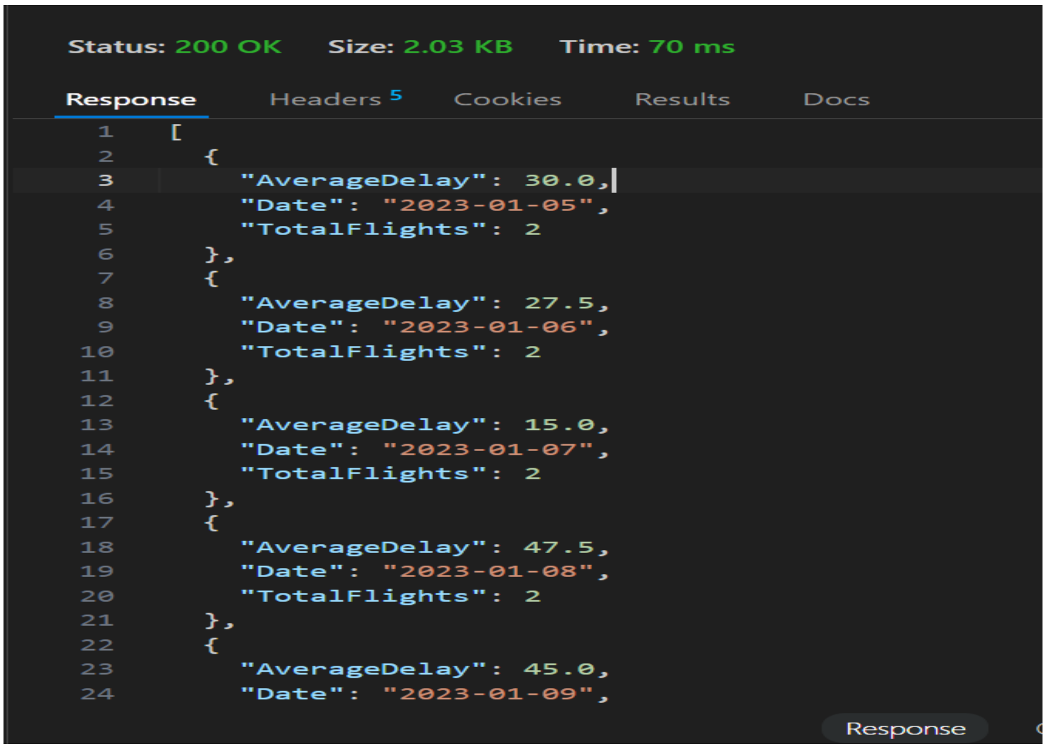


```
Airline,AverageDelay,TotalFlights
American,34.16666666666667,12
Delta,32.142857142857146,14
Southwest,11.875,12
United,38.75,12
```

The output shows average flight delays by airline.
For each airline, it lists:

- Average delay across all flights
- Total number of flights considered

http://127.0.0.1:5000/summary/date/json

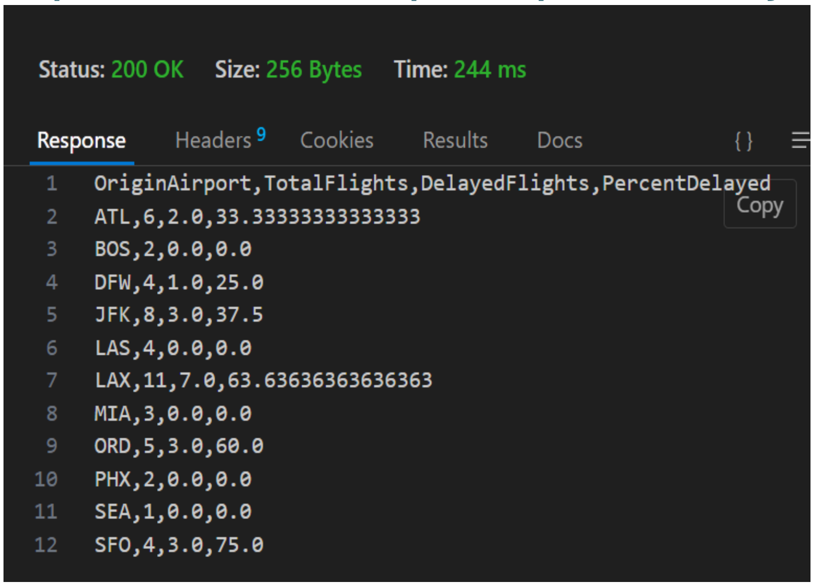


```
[
  {
    "AverageDelay": 30.0,
    "Date": "2023-01-05",
    "TotalFlights": 2
  },
  {
    "AverageDelay": 27.5,
    "Date": "2023-01-06",
    "TotalFlights": 2
  },
  {
    "AverageDelay": 15.0,
    "Date": "2023-01-07",
    "TotalFlights": 2
  },
  {
    "AverageDelay": 47.5,
    "Date": "2023-01-08",
    "TotalFlights": 2
  },
  {
    "AverageDelay": 45.0,
    "Date": "2023-01-09",
```

The output represents daily average flight delays over a period of time.
It shows, for each day:

- The average delay of all flights that day.
- The total number of flights considered.
- The date corresponding to those flights.

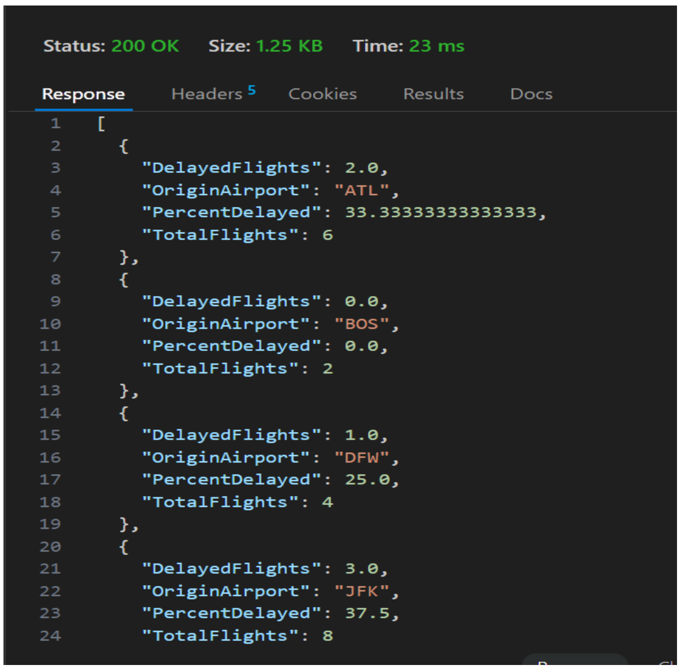
http://127.0.0.1:5000/export/airport_summary



```
OriginAirport,TotalFlights,DelayedFlights,PercentDelayed
ATL,6,2.0,33.333333333333333
BOS,2,0.0,0.0
DFW,4,1.0,25.0
JFK,8,3.0,37.5
LAS,4,0.0,0.0
LAX,11,7.0,63.63636363636363
MIA,3,0.0,0.0
ORD,5,3.0,60.0
PHX,2,0.0,0.0
SEA,1,0.0,0.0
SFO,4,3.0,75.0
```

The output represents flight delay statistics by origin airport.
->The total number of flights.
->The number of delayed flights.
-> The percentage of flights delayed.

http://127.0.0.1:5000/summary/airport/json



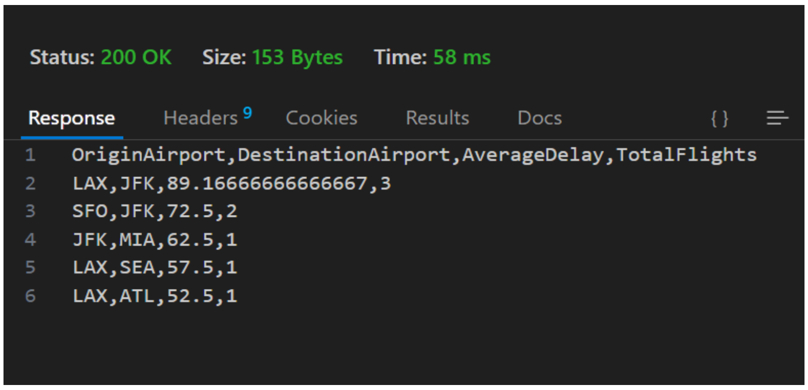
```
[
  {
    "DelayedFlights": 2.0,
    "OriginAirport": "ATL",
    "PercentDelayed": 33.333333333333333,
    "TotalFlights": 6
  },
  {
    "DelayedFlights": 0.0,
    "OriginAirport": "BOS",
    "PercentDelayed": 0.0,
    "TotalFlights": 2
  },
  {
    "DelayedFlights": 1.0,
    "OriginAirport": "DFW",
    "PercentDelayed": 25.0,
    "TotalFlights": 4
  },
  {
    "DelayedFlights": 3.0,
    "OriginAirport": "JFK",
    "PercentDelayed": 37.5,
    "TotalFlights": 8
```

The output shows flight delay details by origin airport.

For each airport, it lists:

- Total flights
- Number of delayed flights
- Percentage of delayed flights

http://127.0.0.1:5000/export/route_summary



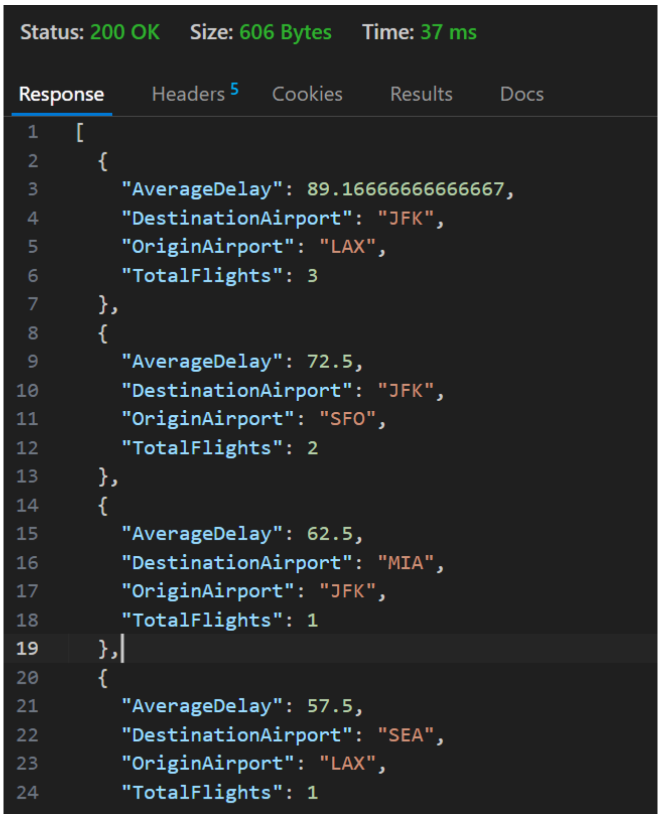
```
OriginAirport,DestinationAirport,AverageDelay,TotalFlights
LAX,JFK,89.16666666666667,3
SFO,JFK,72.5,2
JFK,MIA,62.5,1
LAX,SEA,57.5,1
LAX,ATL,52.5,1
```

The output represents average flight delays by route.

For each route, it shows:

- Origin and destination airports
- Average delay for flights on that route
- Total number of flights considered

http://127.0.0.1:5000/summary/route/json



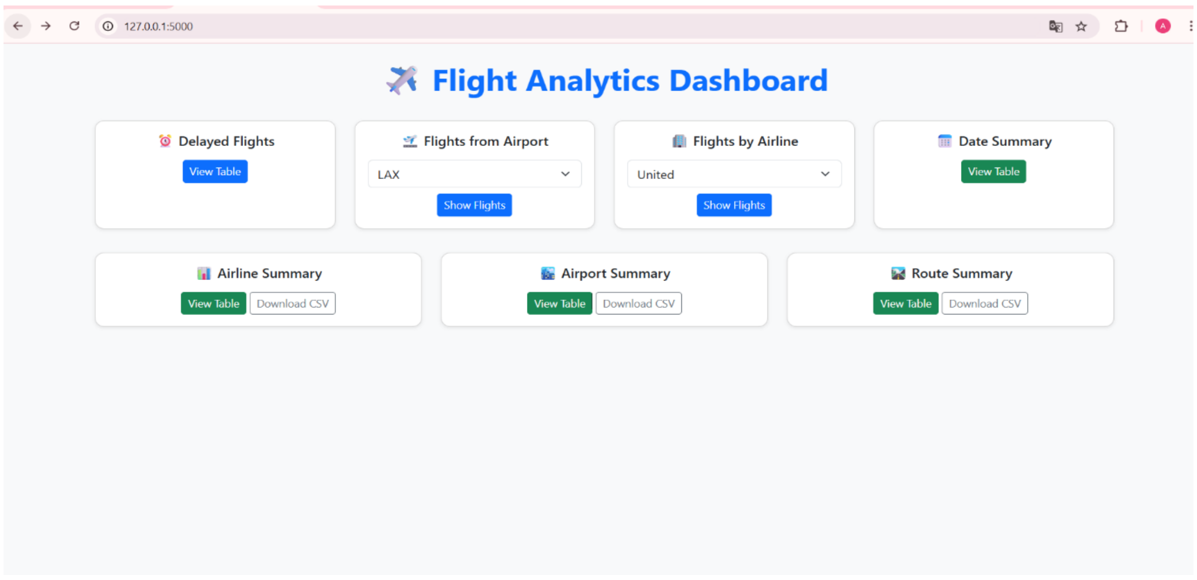
```
[
  {
    "AverageDelay": 89.16666666666667,
    "DestinationAirport": "JFK",
    "OriginAirport": "LAX",
    "TotalFlights": 3
  },
  {
    "AverageDelay": 72.5,
    "DestinationAirport": "JFK",
    "OriginAirport": "SFO",
    "TotalFlights": 2
  },
  {
    "AverageDelay": 62.5,
    "DestinationAirport": "MIA",
    "OriginAirport": "JFK",
    "TotalFlights": 1
  },
  {
    "AverageDelay": 57.5,
    "DestinationAirport": "SEA",
    "OriginAirport": "LAX",
    "TotalFlights": 1
```

The output shows average flight delays for specific routes.

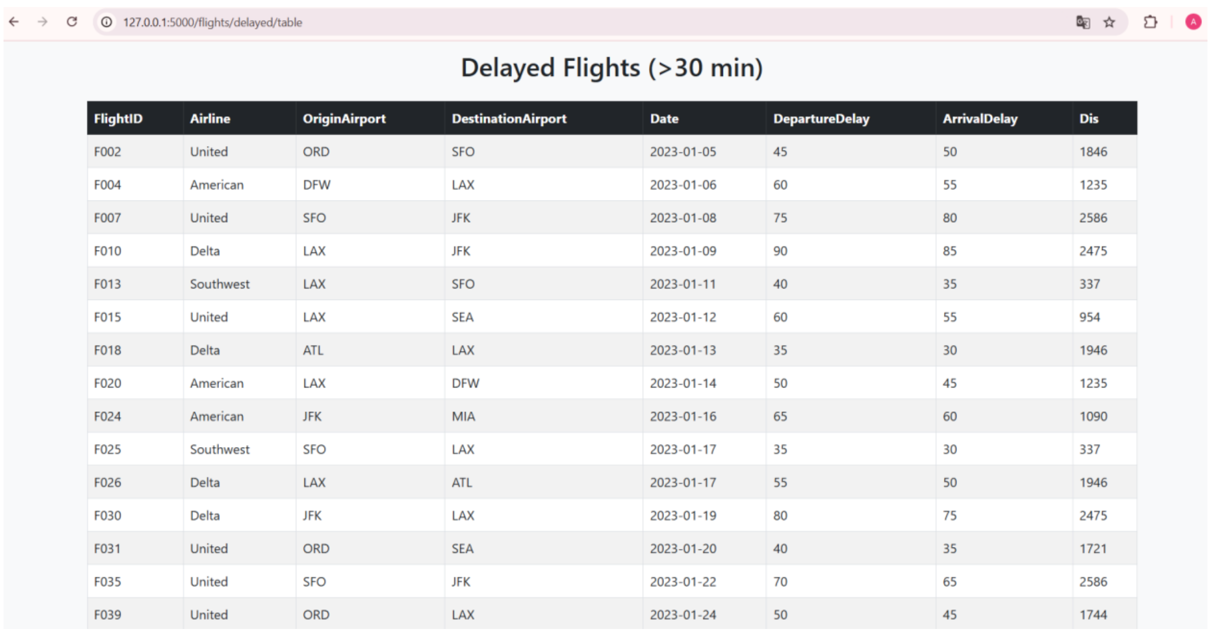
For each route, it provides:

- Origin and destination airports
- Average delay of flights on that route
- Total number of flights considered

FRONT-END DESIGN INTERFACE:



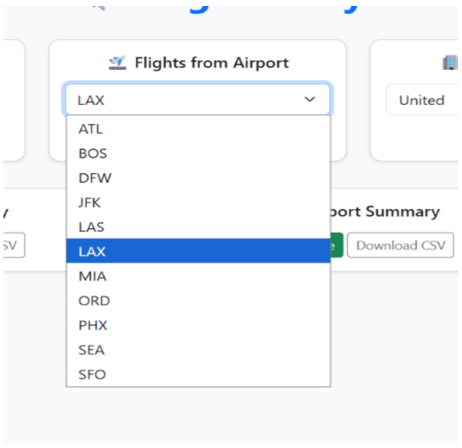
WEBPAGE DASHBOARD INTERFACE

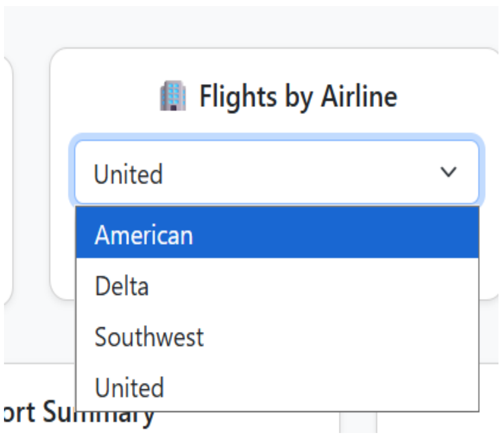


Delayed Flights (>30 min)

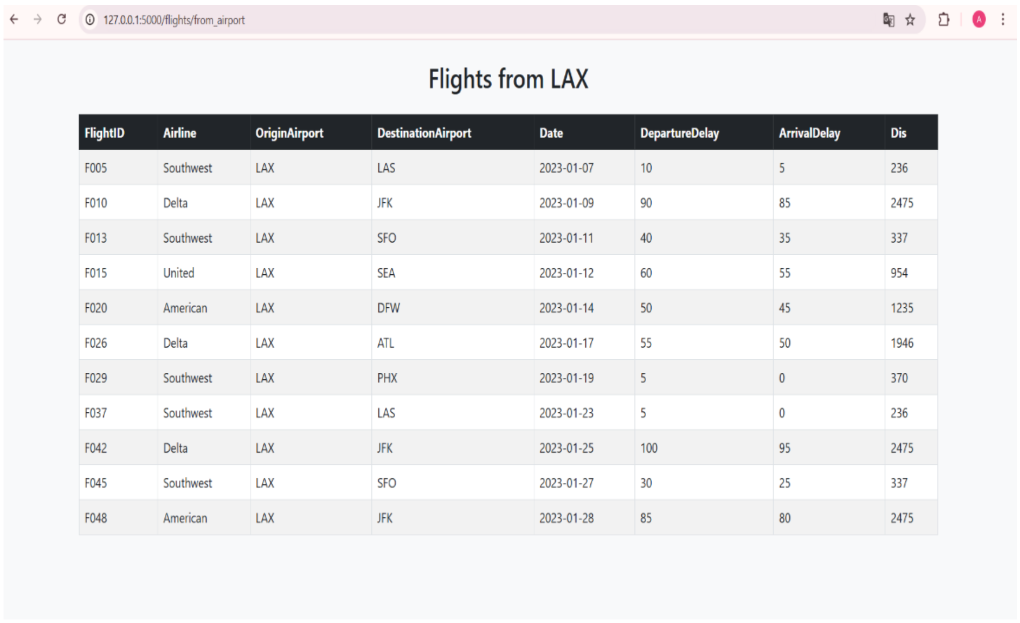
FlightID	Airline	OriginAirport	DestinationAirport	Date	DepartureDelay	ArrivalDelay	Dis
F002	United	ORD	SFO	2023-01-05	45	50	1846
F004	American	DFW	LAX	2023-01-06	60	55	1235
F007	United	SFO	JFK	2023-01-08	75	80	2586
F010	Delta	LAX	JFK	2023-01-09	90	85	2475
F013	Southwest	LAX	SFO	2023-01-11	40	35	337
F015	United	LAX	SEA	2023-01-12	60	55	954
F018	Delta	ATL	LAX	2023-01-13	35	30	1946
F020	American	LAX	DFW	2023-01-14	50	45	1235
F024	American	JFK	MIA	2023-01-16	65	60	1090
F025	Southwest	SFO	LAX	2023-01-17	35	30	337
F026	Delta	LAX	ATL	2023-01-17	55	50	1946
F030	Delta	JFK	LAX	2023-01-19	80	75	2475
F031	United	ORD	SEA	2023-01-20	40	35	1721
F035	United	SFO	JFK	2023-01-22	70	65	2586
F039	United	ORD	LAX	2023-01-24	50	45	1744

DELAYED FLIGHTS INTERFACE





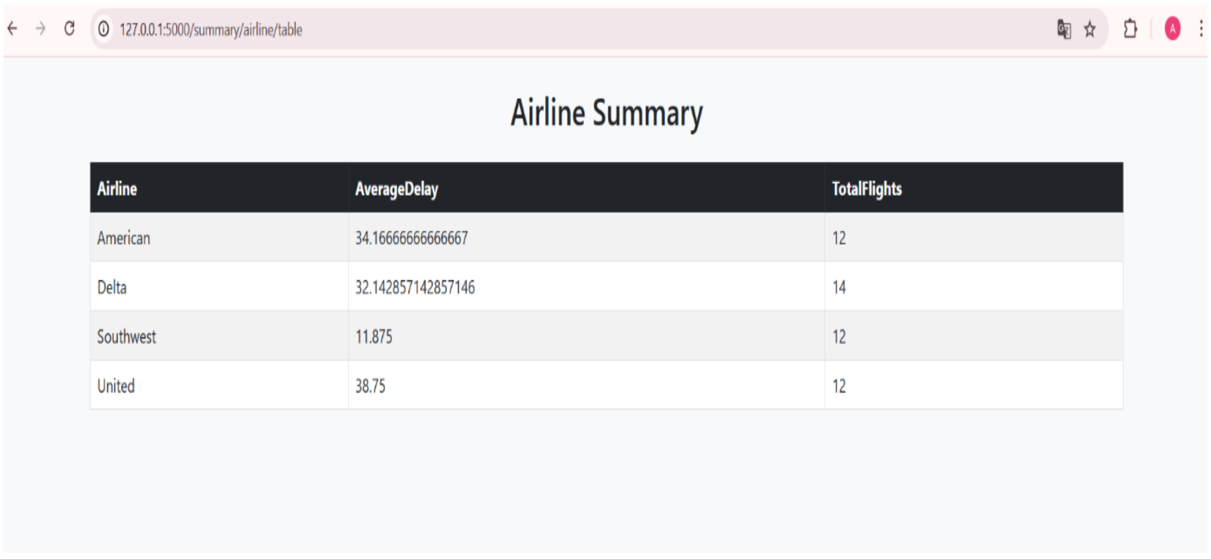
SELECTING AIRPORT AND AILINE



Flights from LAX

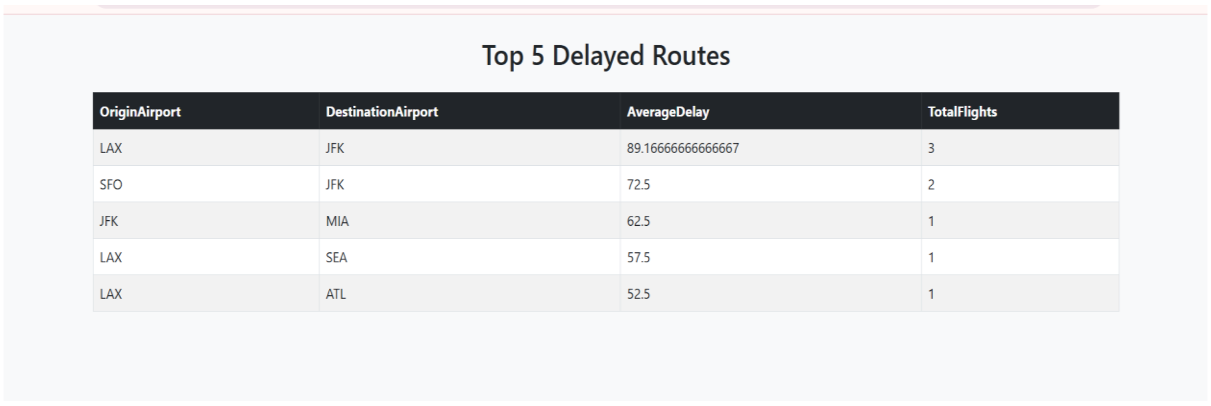
FlightID	Airline	OriginAirport	DestinationAirport	Date	DepartureDelay	ArrivalDelay	Dis
F005	Southwest	LAX	LAS	2023-01-07	10	5	236
F010	Delta	LAX	JFK	2023-01-09	90	85	2475
F013	Southwest	LAX	SFO	2023-01-11	40	35	337
F015	United	LAX	SEA	2023-01-12	60	55	954
F020	American	LAX	DFW	2023-01-14	50	45	1235
F026	Delta	LAX	ATL	2023-01-17	55	50	1946
F029	Southwest	LAX	PHX	2023-01-19	5	0	370
F037	Southwest	LAX	LAS	2023-01-23	5	0	236
F042	Delta	LAX	JFK	2023-01-25	100	95	2475
F045	Southwest	LAX	SFO	2023-01-27	30	25	337
F048	American	LAX	JFK	2023-01-28	85	80	2475

AIRLINE SUMMARY



Airline Summary

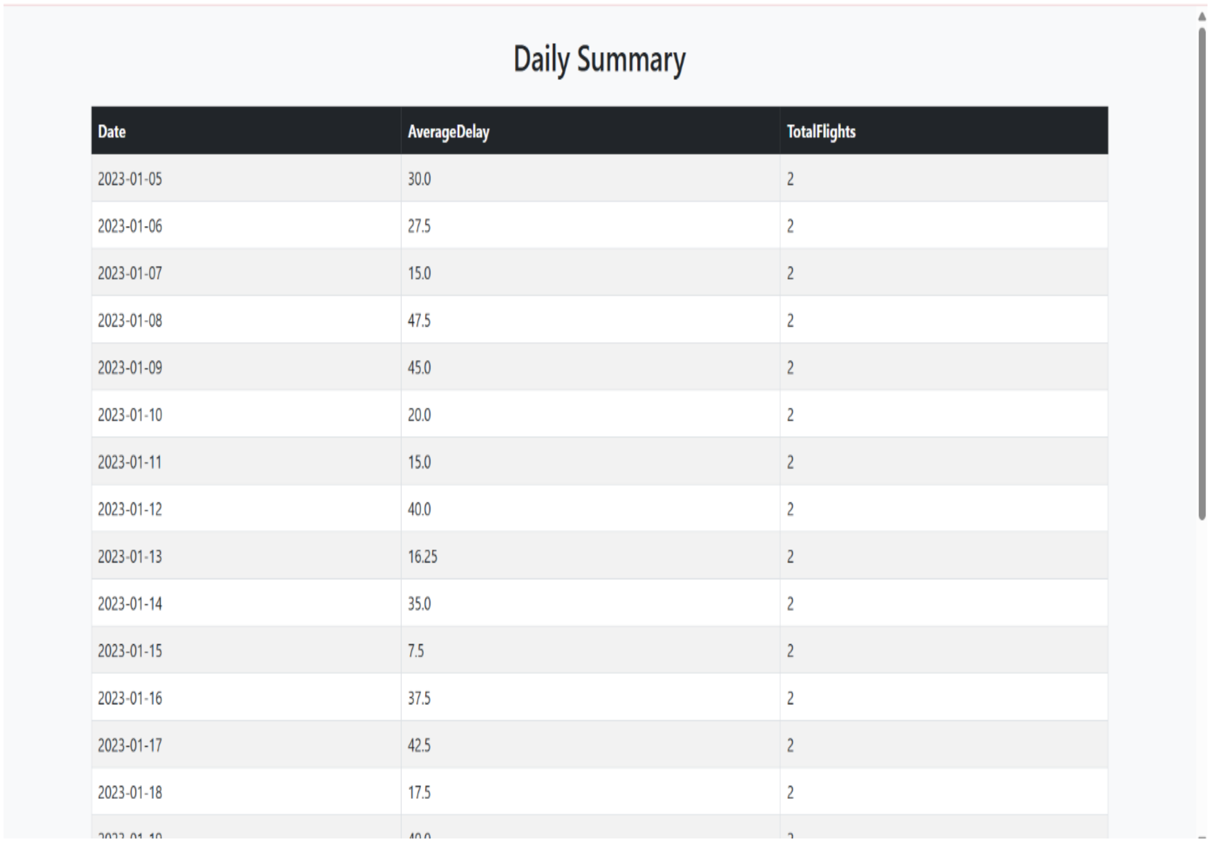
Airline	AverageDelay	TotalFlights
American	34.166666666666667	12
Delta	32.142857142857146	14
Southwest	11.875	12
United	38.75	12



Top 5 Delayed Routes

OriginAirport	DestinationAirport	AverageDelay	TotalFlights
LAX	JFK	89.16666666666667	3
SFO	JFK	72.5	2
JFK	MIA	62.5	1
LAX	SEA	57.5	1
LAX	ATL	52.5	1

TOP 5 DELAYED ROUTES INTERFACE

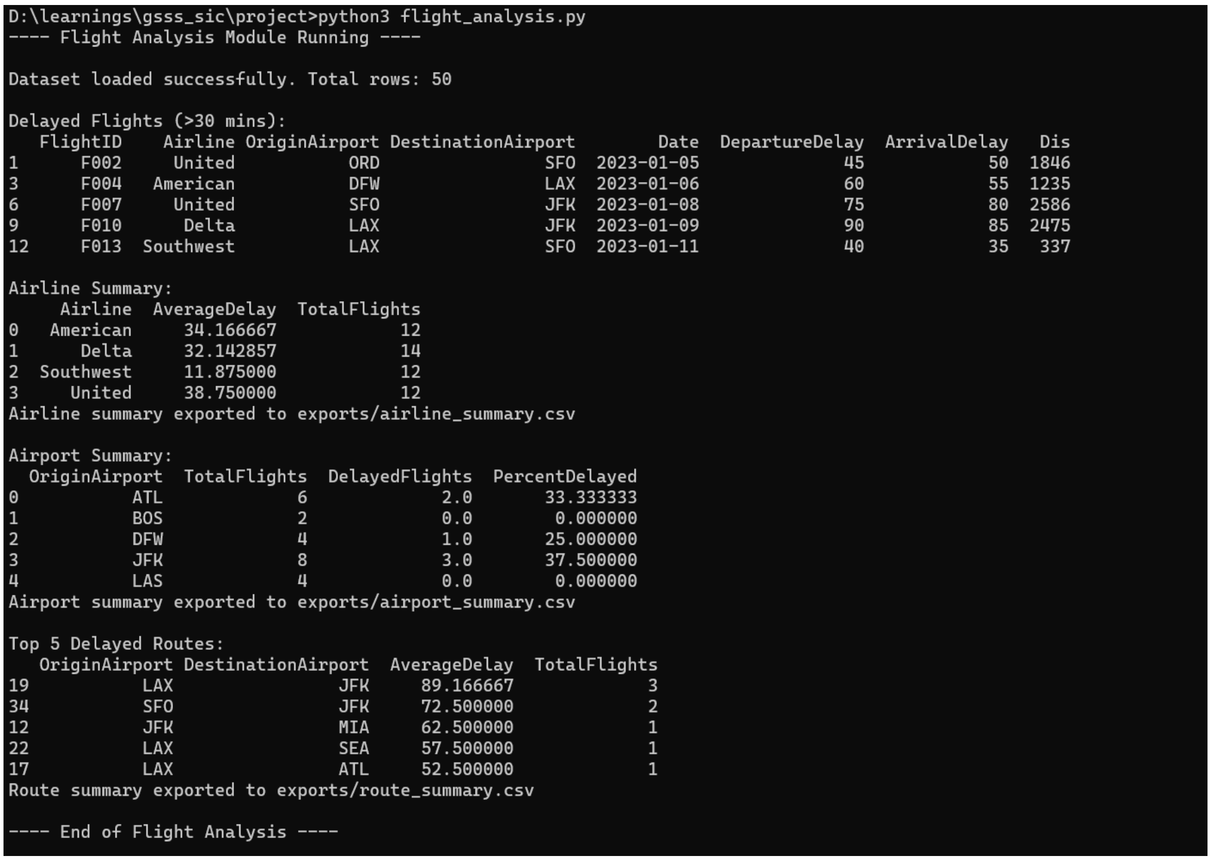


Daily Summary

Date	AverageDelay	TotalFlights
2023-01-05	30.0	2
2023-01-06	27.5	2
2023-01-07	15.0	2
2023-01-08	47.5	2
2023-01-09	45.0	2
2023-01-10	20.0	2
2023-01-11	15.0	2
2023-01-12	40.0	2
2023-01-13	16.25	2
2023-01-14	35.0	2
2023-01-15	7.5	2
2023-01-16	37.5	2
2023-01-17	42.5	2
2023-01-18	17.5	2

DAILY SUMMARY INTERFACE

COMMAND PROPMT OUTPUT:



```
D:\learnings\gsss_sic\project>python3 flight_analysis.py
---- Flight Analysis Module Running ----

Dataset loaded successfully. Total rows: 50

Delayed Flights (>30 mins):
   FlightID    Airline OriginAirport DestinationAirport        Date  DepartureDelay  ArrivalDelay   Dis
1      F002     United           ORD                SFO  2023-01-05              45            50  1846
3      F004   American           DFW                LAX  2023-01-06              60            55  1235
6      F007     United           SFO                JFK  2023-01-08              75            80  2586
9      F010      Delta           LAX                JFK  2023-01-09              90            85  2475
12     F013  Southwest           LAX                SFO  2023-01-11              40            35   337

Airline Summary:
     Airline  AverageDelay  TotalFlights
0   American     34.166667            12
1      Delta     32.142857            14
2  Southwest     11.875000            12
3     United     38.750000            12
Airline summary exported to exports/airline_summary.csv

Airport Summary:
  OriginAirport  TotalFlights  DelayedFlights  PercentDelayed
0           ATL             6             2.0       33.333333
1           BOS             2             0.0        0.000000
2           DFW             4             1.0       25.000000
3           JFK             8             3.0       37.500000
4           LAS             4             0.0        0.000000
Airport summary exported to exports/airport_summary.csv

Top 5 Delayed Routes:
   OriginAirport DestinationAirport  AverageDelay  TotalFlights
19           LAX                JFK     89.166667             3
34           SFO                JFK     72.500000             2
12           JFK                MIA     62.500000             1
22           LAX                SEA     57.500000             1
17           LAX                ATL     52.500000             1
Route summary exported to exports/route_summary.csv

---- End of Flight Analysis ----
```

The output shows a flight delay analysis summary.

It includes:

- Delayed flights (>30 mins) with details like airline, route, and delay times
- Airline summary showing average delays and total flights per airline
- Airport summary showing total flights, delayed flights, and percent delayed for each origin airport
- Top 5 delayed routes ranked by highest average delay

11.CONCLUSION AND BIBLIOGRAPHY

Conclusion:

The Flight Analytics Dashboard provides a **comprehensive and interactive solution** for analyzing flight data. It integrates **Python data analysis, Flask backend, and Bootstrap frontend** to deliver insights into delays, airline performance, airport efficiency, and top delayed routes. Users can **view, filter, and export data**, enabling **data-driven decision-making**.

Bibliography / References:

1. Python Documentation – https://docs.python.org/3/
2. Flask Documentation – https://flask.palletsprojects.com/
3. Pandas Documentation – https://pandas.pydata.org/
4. Bootstrap 5 Documentation – https://getbootstrap.com/
5. Flight data source (flights.csv)