Deliverable 1: Flight Data Extraction and CO_2 Emissions Calculation

Overview

This deliverable focuses on data extraction and processing to generate a CSV file containing flight data with calculated CO_2 emissions. The process uses the AviationStack API for real-time flight data and includes hardcoded scraped data from airline sustainability reports (simulating HTML scraping). The output includes two CSV files: `flight_emissions.csv` (full dataset) and `flight_emissions_clean.csv` (filtered for complete emissions data). Notebooks implement the API calls and scraping logic.

Stepwise Explanation

1. API Key and Initialization

1. Obtain an API key from AviationStack (e.g., `034659c48f30f9832c05aea458a29eb5`).
2. Initialize the `FlightCarbonCalculator` class with the API key. This sets up:
   - Base URL: `https://api.aviationstack.com/v1`
   - Aircraft emissions factors (e.g., `A320`: 0.095 kg CO_2/km/pax)
   - Aircraft capacities (e.g., `A320`: 180 seats)
   - Airport coordinates for 100 major airports
   - Scraped airline efficiency data (hardcoded from reports, e.g., Delta: 0.095 kg CO_2e/pkm)

2. Fetching Flight Data via API

1. Use `get_flights_data()` to query the `/flights` endpoint with parameters: `access_key`, `limit` (e.g., 100), `offset` (for pagination).
2. Handle requests with timeouts and errors. Parse JSON to extract flight details: aircraft type, departure/arrival airports (IATA), airline, status, etc.
3. Batch fetch in loops (e.g., for 500 flights, fetch in batches of 100 with offsets 0, 100, ..., 400) to avoid API limits. Include a sleep of 1 second between batches.

3. Processing Flight Data

1. In `process_flight_data()`, for each flight:
    - Extract keys: flight number, airline, aircraft ICAO code, origin/destination IATA, scheduled times, status.
    - Calculate passenger capacity using `get_aircraft_capacity()` (match ICAO code or fallback `DEFAULT`: 200).
    - Compute geodesic distance (km) using `calculate_distance()` with `geopy.geodesic` on airport coordinates.
    - Calculate CO_2 emissions per passenger in `calculate_co2_emissions()`:
        - Base: distance × emission factor (from `get_aircraft_emission_factor()`, fallback `DEFAULT`: 0.095)
        - Adjust for load factor (0.8): base / 0.8
        - Add penalties: +95 km equivalent for flights <1000 km, +50 km for flights <3000 km
    - Add derived fields: `co2_per_passenger_kg`, `co2_per_passenger_per_km`, extraction timestamp.
2. Handle errors (e.g., missing airports) with warnings and None values.

4. HTML Scraping for Airline Data (Simulated)

1. In a real notebook, use libraries like `BeautifulSoup` or `Scrapy` to scrape airline sustainability reports (e.g., `delta.com/sustainability`, `lufthansa.com/reports`).
2. Extract metrics: emissions intensity (kg CO_2/pkm), notes (e.g., fuel efficiency improvements).
3. Hardcoded in code as `airline_efficiency_from_reports` dictionary.
4. Notebook includes `requests.get()` for URLs, parsing HTML tables/divs, storing data in dict or CSV.

5. Saving to CSV

1. In `extract_and_save_flights()`:
    - Compile processed flights into a Pandas DataFrame.
    - Save full DF to `flight_emissions.csv`.
    - Clean DF: drop rows without origin/destination; filter for non-null CO_2.
    - Save to `flight_emissions_clean.csv`.
    - Display stats: total flights, with distance, with CO_2.

6. Notebooks Structure

- **API Notebook** (`flight_api_extraction.ipynb`): imports requests, pandas, geopy. Defines class/methods for API fetch and processing. Runs extraction loop and saves CSVs.
- **HTML Scraping Notebook** (`airline_scraping.ipynb`): imports requests, BeautifulSoup. Defines functions to scrape URLs for airlines. Parses and stores data; exports to JSON/CSV for integration.

7. Extraction Documentation

- Document API limits (e.g., 1000 queries/month free), error handling, and data sources (AviationStack for flights, geopy for distances, manual scraping for reports).
- Assumptions: Load factor 0.8, emission factors from industry averages (ICAO data).
- Limitations: Airport list limited to 100; no real-time scraping (data hardcoded).