Conestoga College

School of Applied Computer Science & Information Technology

SENG8080 - Case Studies Big Data

Flight Analysis

Nilaykumar Patel (8892425)

Rajatkumar Maisuriya (8870363)

Drashti Bhavsar (8890040)

November 29, 2023

**Abstract:**

In the ever-changing world of modern air travel, effective flight information management is critical for both passengers and airlines. In this report, a Python-based technique that simplifies the management of critical flight information—such as airline data, flight numbers, status, operational information, departure, and arrival times—is shown. The suggested method presents an extensive and current source of flight information by combining real-time data from an Application Programming Interface (API) with a carefully selected data set.

The primary processing unit of the system architecture is a Python backend. Data is combined from two main sources: an internal dataset that offers additional details and historical context, and an external API that offers real-time updates.

A complete database of airline information, exact flight number management, real-time flight status updates, carrier monitoring, and precise departure and arrival time recording are some of the system's key features. By the smooth integration of data from the dataset and the API, the system ensures a stable, dependable, and continuously updated repository of flight-related data.

This flight information management system, which is based on Python, offers a comprehensive solution to meet the constantly evolving needs of the aviation sector, meeting the requirements of both airlines and passengers.

The following are the main points of emphasis for this Python code project:

**Data Retrieval:** The code retrieves flight information from two different sources: a dataset containing historical records and an API with real-time data. This combination guarantees that the system meets different analytical and operational needs by providing both historical and current flight information.

**Data management:** It is simple to query and obtain flight details since the retrieved data is arranged and kept in a structured database. The airline details, flight numbers, flight statuses, operational data, departure and arrival times, and other details are all supported by the database structure.

**Contents**

[Introduction 4](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960212)

[System Diagram](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960213) 6

[Data Research and Integration 6](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960213)

[Data Collection 8](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960214)

[Data Storage and Maintenance 9](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960215)

[Data Quality 11](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960216)

[Data Analysis and Visualization 14](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960217)

Extension 18

[Project Timeline 19](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960218)

[References 20](file:///C:\Users\Drasti\Downloads\SENG8080%20Major%20Final%20Project%20Template.docx#_Toc147960219)

* **Introduction:**

This project attempts to conduct a thorough analysis of airline operations, with particular attention to historical performance metrics, dynamic real-time tracking between airports, and schedule efficiency. The project's goals are to offer insightful information about airline scheduling, performance trends, and potential predictive analytics for maximizing efficiency in operations. To this end, it will investigate and preprocess the Kaggle dataset, apply modelling techniques, and create an intuitive dashboard.

1. **Data Retrieval:**

**API Integration for Real-Time Data:** The Python code is designed to interact with a specific flight data API, offering a quick and easy way to get the most recent flight data. This API is a crucial channel for obtaining up-to-date flight information, including key factors like the flight's status, arrival and departure times, and operating details. With this integration, the system is guaranteed to be ready to quickly satisfy the urgent needs of both travelers and airline employees. This API integration allows for real-time flight monitoring, which improves the system's responsiveness to dynamic and quickly changing conditions in the aviation landscape.

**Dataset for Historical Records:** Apart from the real-time API, the system includes an extensive dataset of historical flight records. The mentioned dataset functions as a store for relevant information obtained from prior flights, including but not limited to airline names, flight numbers, status updates, departure, and arrival times. Through the incorporation of past flight data, the system becomes capable of handling various analytical needs, such as investigating past flight patterns and detecting delays-related patterns. By adding this historical dataset, the system's analytical powers are enhanced and a more comprehensive understanding of flight operations throughout time is made possible.

**Data Synchronization:** The Python code has mechanisms for regular synchronization with the API to preserve the correctness and relevancy of real-time data. By doing this, the system is guaranteed to be updated with the most recent data, keeping up with the aviation industry's rapid changes. In the case of historical data, the system takes an accommodating stance. It can smoothly integrate new data as it becomes available, or it can update old records on a regular basis to maintain previous data current. Because of its flexibility, the system's historical dataset is guaranteed to continue being a valuable and dynamic resource, reflecting the ongoing changes in flight operations and related data.

1. **Data Management:**

**Database Structure:** Data from historical datasets and real-time API sources are arranged in the flight information management system using a well-structured relational database. With tables for flight numbers, statuses, operational information, departure and arrival times, and airline information, the schema makes sure that the data is arranged logically and effectively.

**Normalization and Data Consistency:** The normalizing approaches are employed by the system to minimize redundancy and increase efficiency. For instance, flight records and airline information are connected, reducing data duplication, and enhancing overall data consistency. This method lowers the possibility of errors and improves database reliability.

**Query and Retrieval:** Users can access precise flight information based on several parameters by utilizing the advanced queries supported by the database. The system's structured indexing and structure allow for effective and efficient searches, whether one is looking for historical delay information, airline flight information, or operational specifics for companies.

In summary, the flight information management system built on Python effectively arranges data in a relational database from both historical and real-time sources. The system provides a flexible answer to operational and analytical requirements in the aviation industry, with strong querying capabilities and standardization ensuring data consistency.

* **System Diagram**

A diagram of a diagram of a diagram

Description automatically generated

* **Data Research and Integration:**

**Data Sources:**

1. **Flight API Real-Time Data:** The Flight API is essential for receiving the most recent knowledge possible on flights, including their statuses, times of departure, and times of arrival. To meet urgent passenger demands, follow flights in real-time, and keep your system current with information, you must have access to this data source.

**API Link:** <https://docs.flightapi.io/track-flights-between-airports>

Key Steps for Utilizing the Flight API:

* Acquire an API key by registering for access to the Flight API.
* To get real-time flight information, use Python libraries like requests or specialized Python SDKs that the API provides.
* To make sure that your system always has the most recent data, use recurring or event-driven data synchronization with the API.
* Put the real-time flight data into your database in a way that follows the schema intended for historical data.

1. **Kaggle Airline Flight Dataset:** The Kaggle dataset includes historical flight information about individual flights, such as the names of the airlines, the flight numbers, the airports of departure and arrival, the times of the departure and arrival, and more. This dataset is useful for gathering historical flight data for trends, delays, and other statistical analysis. It can act as the system's starting point for managing flight details.

**Kaggle Dataset Link:**

[https://www.kaggle.com/datasets/arunjangir245/airline-flight-dataset- schedule-performance-etc](https://www.kaggle.com/datasets/arunjangir245/airline-flight-dataset-%20schedule-performance-etc)

Key Steps for Utilizing the Kaggle Dataset:

* Utilizing a library like Pandas, download the dataset from Kaggle and import it into your Python environment.
* Make a database table or structure that precisely fits the dataset's structure to hold the historical flight data.
* Add the historical flight information to the database.

**Data Management and Integration:**

With data from both sources now available in your system, we can integrate them as follows:

**Database Schema:** Design a database schema that can handle both past and present flight information. Tables for airline information, flight numbers, flight statuses, operational data, departure times, and arrival times are all possible.

**Synchronization:** The system uses scheduled synchronization with the Flight API to maintain the most recent version of the real-time data. To make sure that the most recent information is provided, this calls for recurring updates. Depending on the speed of change in flight data, the frequency of synchronization can be adjusted to meet operational needs, for example, every few minutes or hourly.

To maintain synchronization, the Flight API requests the most recent data, and the relevant database entries are updated. This guarantees that the system will continue to react to modifications in real-time flight data, giving users current and accurate data for operational and analytical needs.

In conclusion, the system is designed to support both historical and real-time flight data, and the scheduled synchronization with the Flight API guarantees that the system continuously keeps the most recent data necessary for efficient decision-making in the aviation industry.

* **Data Collection:**

**Step 1: Data Retrieval from Kaggle (Historical Data):**

**Data Retrieval:** The first step is to obtain the Kaggle dataset with historical flight data from the link that is provided above. This dataset may be accessible in CSV file format. We can directly retrieve it using Python's requests library, or you can manually download and save it to your project directory.

**Pandas Data Loading:** The historical flight data from the Kaggle dataset can be loaded into a Pandas DataFrame using the Pandas package.

**Pyodbc database connection:** Using the pyodbc package, establish a connection to your SQL Server database so that you may save the historical flight information.

**Database Table Creation:** Create the database tables in your SQL Server database using SQL commands, defining their structure to correspond to the Kaggle dataset.

**Data Insertion:** Embed the Kaggle Data Frame’s historical flight data in your SQL Server database. Insert records into the database that are consistent with the Data Frame’s table structure.

**Step 2: Real-Time Data Retrieval from Flight API:**

**Real-time data fetching:** We used API key to retrieve current flight information from the Flight API using Python's requests module.

**Database Connection and Table**: Establish a connection to your SQL Server database and we ensure that you have a table to store the real-time flight data.

**Data Insertion:** Like how historical data is inserted, insert the real-time flight data into your SQL Server database.

* **Data Storage and Maintenance with SQL Server:**

**Data Storage:** Data storage is the important phase of any analysis process. It includes data storage process in reliable storage tools or database such as MySQL, MongoDB, SQL server, etc.

For this project, SQL server database is used to insert data into specified database and tables. And stored them for further step of analysis.

Here is the database structure and its insertion code.

**Database Structure:**

**Historical Data Structure:** We created a table in SQL Server first to hold historical flight information received from the Kaggle dataset. Define the columns in the dataset, such as airline, flight\_number, departure\_airport, arrival\_airport, departure\_time, and arrival\_time, to be included in the structure of this table.

**Real-time Data Table:** Created a table for holding real-time flight data that has been acquired from the Flight API in a similar manner. This table should have columns for airline, flight\_number, departure\_airport, arrival\_airport, departure\_time, arrival\_time, and status that correspond to the schema of the real-time data obtained from the API.

**Data Insertion:**

**Historical Data Insertion:** Insert the historical flight information into the database table that is made specifically for historical information. To populate the table, SQL INSERT commands can be executed using Python and pyodbc.

**SQL Query:**

**CREATE TABLE HistoricaDatal (**

**FlightID INT IDENTITY(1,1) PRIMARY KEY,**

**Date DATE,**

**DepTime TIME,**

**ArrTime TIME,**

**UniqueCarrier VARCHAR(255),**

**FlightNum VARCHAR(10)**

**);**

**INSERT INTO HistoricalData (Date, DepTime, ArrTime, UniqueCarrier, FlightNum)**

**VALUES (?, ?, ?, ?, ?);**

**Real-Time Data Insertion:** Insert real-time flight data into the real-time data table in a manner like that described above. Python may be used to carry out SQL INSERT statements.

**SQL Query:**

**CREATE TABLE Schedule (**

**Airline VARCHAR(255),**

**FlightNumber VARCHAR(10),**

**Status VARCHAR(50),**

**OperatedBy VARCHAR(255),**

**DepartureTime DATETIME,**

**ArrivalTime DATETIME**

**);**

**INSERT INTO Schedule (Airline, FlightNumber, Status, OperatedBy, DepartureTime, ArrivalTime)**

**VALUES (?, ?, ?, ?, ?, ?, ?);**

* **Data Maintenance:**

In order to maintain effective data in your airline operations project, we should clean historical data from the Kaggle dataset on a regular basis, continuously validate data standards, set up mechanisms for updating historical data, stream real-time data from flightapi.io with regular queries, monitor data quality, use version control for datasets, maintain thorough metadata, regularly backup datasets for data security, update documentation to reflect changes, and perform periodic reviews to ensure alignment with changing project requirements. The project's continued success will be bolstered by these procedures, which will improve data reliability.

**Data Cleanup:** Over time, databases may gather unnecessary or out-of-date data. Consider putting in place data cleanup procedures to preserve data quality and improve database performance. For instance, you can set up data retention policies to have records that are automatically deleted when they reach a particular age be automatically deleted records older than a certain threshold.

* **Data Quality:**

The reliability, consistency, accuracy, and completeness of data are referred to as data quality. To ensure high data quality in your airline operations project, historical data from Kaggle and real-time data from flightapi.io need to be cleaned, validated, and monitored on a regular basis. To gain valuable insights and make well-informed decisions, we need access to reliable data.

**Accuracy**: Providing accurate and correct information to improve the dependability of analyses and forecasts, both in real-time flight tracking data from the flightapi.io API and historical airline data from the Kaggle dataset.

**Completeness**: Ensuring that datasets contain every relevant information and reducing gaps or missing values in historical records and real-time updates from the flightapi.io API to present a comprehensive picture of airline operations.

**Consistency**: Keeping real-time data from the flightapi.io API and the Kaggle dataset consistent in terms of definitions, formats, and structures will promote coherence and make integration easier.

**Reliability**:  In order to ensure dependability for crucial decision-making processes in airline operations, reliability must be established by assuring data integrity, security, and the consistent application of validation checks across historical and real-time data.

* **Images:FlightInfo.py**

A screenshot of a computer

Description automatically generated

**Database**

A screenshot of a computer

Description automatically generated

**HistoricalData.py**

A screenshot of a computer

Description automatically generated

**Database**

A screenshot of a computer

Description automatically generated

* **Data Analysis & Visualization:**

**Visualization of historical data & Analysis**

**1st visualizations (Historical Data): Average Fly time by Airline with flight number**

A screenshot of a computer screen

Description automatically generated

**Analysis:** From the above tableau graph it can be observed that, there are many Airlines and their flights. And this graph compares airlines, their flights based on their average fly time. And at that time most of the flight’s average fly time was around 20 hours to 30 hours.

**2nd Visualization (Historical Data): Busiest hour of the day**

A graph on a white background

Description automatically generated

**Busiest hour of the day (real-time data)**

A graph with lines and dots

Description automatically generated

**Analysis:** Comparing the above two graphs, the busiest day for flights in historical data is at 7 PM. So, in this period most flights were in the air. while in the current situation during afternoon is the busiest time of the day.

**3rd visualization: Airline average timeline per day for historical**.

A graph of blue bars

Description automatically generated with medium confidence

**Visualization of current time**

A graph of green bars

Description automatically generated

**Analysis:** The above two graphs describe the average Airline timeline for a day. It shows the number of flights for every hour per day for selected airlines. Both graphs compare past and current average timelines for particular airline.

**4th visualization:**

A colorful lines on a white background

Description automatically generated

**Analysis:** The above graph describes average Airline and flight number timeline for a day. It shows the number of flights with its average fly time.

**Usefulness 1**: From the above graph it can be recommended when a particular flight will arrive or land. Based on their average fly time.

**Usefulness 2**: It can be observed which flight is delayed as per its average fly time and it helps to know reasons behind it.

**Recommendation:** It helps to recommend flight management office to optimize their flight scheduling, minimize flight delays, it helps to monitoring real-time, it also leads to use best flight routes. In the end it helps to improve the customer experience.

**5th visualization:**

A screenshot of a computer

Description automatically generated

**Analysis:** The above graph describes Operators and their number flights it operates. And from the graph American Airline operates highest number of flights.

Usefulness: It describes the collaboration of different operators to manage flights between them.

* **Extension:**

To enhance the project, we think about adding Machine Learning models to predict flight delays, putting in place a recommendation system for customized travel experiences, connecting IoT devices to monitor aircraft health and passenger comfort in real time, and investigating blockchain technology for safe and transparent management of airline records and transactions. The project's capacity for prediction, customer interaction, operational monitoring, and general technological innovation can all be improved by these extensions.

* **Project Timeline:**

|  |  |  |
| --- | --- | --- |
| **Date** | **Deliverable** | **Responsible** |
| Oct 05 | Resource finding (API and dataset) | Drashti |
| Oct 05 | Collected data from API and planned, cleaning | Nilay, Drashti, Rajat |
| Oct 18 | Stored data in the database | Nilay, Rajat, Drashti |
| Oct 21 | Midterm report | Drashti, Nilay, Rajat |
| Nov 18 | Further cleaning of data | Nilay |
| Nov 20 | Testing and solved errors | Nilay, Drashti, Rajat |
| Nov 29 | System Diagram | Rajat |
| Nov 29 | Analysis & visualization | Drashti, Rajat |
| Nov 29 | Final Report | Nilay, Drashti |

**References:**

# By Arun Jangir. (Updated 3 months ago). Airline Flight Dataset: Schedule, Performance etc.

# <https://www.kaggle.com/datasets/arunjangir245/airline-flight-dataset-schedule-performance-etc>

# By FlightAPI. Track Flights between Airports.

# <https://docs.flightapi.io/track-flights-between-airports>

Abstract:

<https://lib.conestogac.on.ca/writing-services/abstracts>

For more information on making a table of contents please see this:

<http://www.mit.edu/course/21/21.guide/contents.htm>

word processing software’s documentation on how to generate a table of contents.

<https://support.office.com/en-ie/article/insert-a-table-of-contents-882e8564-0edb-435e-84b5-1d8552ccf0c0>