Airline Flight Delay Dataset: Filtering and Grouping Operation

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 : AMULYA U

Email : amulyau19@gmail.com

Date : 03-09-25

Table of Contents

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)

1. Clean & Explore Data

Check for missing or duplicate entries.
Convert date/time columns to proper formats.

2. Filtering Operations

Extract flights delayed more than 30 minutes.
Filter flights operated by a specific airline.
Find all flights departing from a specific airport.

3. Grouping Operations

Group by **Airline**: average delay, total flights.
Group by **OriginAirport**: total flights, % delayed.
Group by **Date**: daily average delay.
Identify **top 5 routes** (Origin–Destination) with highest average delay

Project Overview

This project focuses on analysing airline flight records to understand delays arising from weather, operational issues, and technical faults. The dataset includes detailed departure and arrival times along with delay information. Using **Python and Pandas**, the system performs data cleaning, filtering, and grouping to uncover meaningful insights.

The analysis highlights the frequency and distribution of delays, identifies recurring patterns, and classifies delays by their root causes. By summarizing these findings in tabular formats (without visualization), the project provides a clear view of operational inefficiencies.

The results will assist the HR Analytics team in workforce planning, scheduling adjustments, and developing strategies to minimize delays. Overall, the system acts as a data-driven tool for improving airline efficiency and enhancing decision-making.

Dataset Description

The dataset contains flight operation records maintained by the airline company. Each row represents a single flight, with details about its schedule and delay information. The attributes are:

- **FlightID** – A unique identifier assigned to each flight.
- **Airline** – The airline carrier operating the flight.
- **OriginAirport** – The airport where the flight departs.
- **DestinationAirport** – The airport where the flight arrives.
- **Date** – The date of the flight operation.
- **DepartureDelay** – Delay in minutes at the point of departure.
- **ArrivalDelay** – Delay in minutes upon arrival at the destination.
- **Dis** – Distance of the flight route (in miles/kilometers).

This dataset enables analysis of **delays across airlines, routes, and dates**, helping the HR Analytics team identify frequent causes of delays, measure their impact, and support operational improvements.

Features Implemented

Data Loading and Cleaning

- Loaded the flight records dataset (CSV) using Pandas for efficient data handling.
- Handled missing values by imputing or removing incomplete records to ensure reliability.
- Removed duplicate entries and standardized column names for consistency.
- Converted date and time columns into proper datetime objects for accurate delay calculations.
- Normalized categorical values like airline codes and airport names to avoid mismatches.

Filtering Operations

- Filtered flights dynamically based on airline, origin, destination, or specific travel dates.
- Isolated records with specific types of delays (departure or arrival) to focus on targeted analysis.
- Applied multi-level filtering to combine conditions (e.g., flights from a certain airline at a specific airport).
- Created reusable filter functions to simplify repeated queries.

Grouping and Aggregation

- Grouped flight data by airline, airport, or date to calculate aggregated statistics such as average delays.
- Summarized delays by category (weather, operations, technical issues) to identify key causes.
- Calculated airline-wise and airport-wise performance metrics to highlight operational strengths and weaknesses.
- Extended grouping to analyze seasonal or monthly averages for trend monitoring.

Delay Analysis

- Computed total, average, minimum, and maximum delays for different airlines and routes.
- Identified the flights or routes most frequently experiencing delays.
- Highlighted peak travel periods (time of day, weekdays, holidays) with maximum delay frequency.

- Classified delays into short, medium, and long durations for granular insights.

Comparative Analysis

- Compared delays across multiple airlines to identify top and underperforming carriers.
- Evaluated delays over different time intervals (daily, weekly, monthly, seasonal) to capture fluctuations.
- Benchmarked airline performance against industry averages for contextual understanding.
- Provided rankings of airlines and airports based on their delay patterns.

Distance vs Delay Insights

- Analyzed how flight distance correlates with departure and arrival delays.
- Differentiated between short-haul and long-haul flights to detect delay susceptibility.
- Verified if longer flights tend to recover from departure delays during the journey.
- Provided insights into efficiency of scheduling for different distance categories.

Tabular Reporting

- Generated structured tables using Pandas DataFrames for clear presentation of insights.
- Created summary tables for airline delays, airport delays, and delay reasons.
- Ensured reports are easy to interpret for non-technical stakeholders.
- Enabled exporting of results into CSV/Excel formats for sharing with HR and operations teams.

Technical Architecture

The system follows a **data-processing pipeline architecture** designed for analysing flight delays. The architecture is organized into the following layers:

1. Data Source Layer

- Input dataset in **CSV format** containing flight records.
- Attributes include: FlightID, Airline, OriginAirport, DestinationAirport, Date, DepartureDelay, ArrivalDelay, and Distance.

2. Data Ingestion Layer

- Dataset is imported into the system using **Pandas**.
- Data validation is performed to check missing or inconsistent values.

3. Data Processing Layer

- **Filtering**: Extract specific subsets of flights by airline, route, or date.
- **Grouping and Aggregation**: Summarize data based on airline, airport, or delay cause.
- **Computation**: Calculate metrics like average, maximum, and minimum delays.
- **Classification**: Categorize delays into weather, operational, or technical groups.

4. Analysis Layer

- Produces tabular summaries for:
  - Delay trends across airlines and airports.
  - Comparison of departure vs. arrival delays.
  - Distance vs. delay relationships.
- Supports HR Analytics team in identifying operational inefficiencies.

5. Output Layer

- Final results are presented as **structured tables and reports** generated via Pandas.
- No visualizations are included (as per requirements).
- Results can be exported to CSV/Excel for further use by analysts

Outcome and Benefits:

Outcomes

1. **Cleaned and Structured Dataset** – The dataset is standardized with consistent formats, duplicate entries removed, missing values handled, and delays categorized by cause (weather, airline operations, technical). This ensures data reliability for further analysis.
2. **Filtered Views** – HR Analytics can easily narrow down insights to specific airlines, routes, airports, or seasonal trends. For example, delays during monsoon months or festive travel peaks can be studied in isolation.
3. **Grouped Insights** – Grouping by airline, airport, or delay category uncovers recurring patterns, such as specific airports facing congestion or airlines with frequent operational delays.
4. **Delay Statistics** – Provides quantitative insights like averages, totals, and extremes, along with comparisons across multiple dimensions (airline vs. airport, seasonal vs. off-season).
5. **Data-Driven Decisions** – By relying on factual insights, HR and operations can optimize scheduling, identify weak links in the workflow, and improve passenger service reliability.

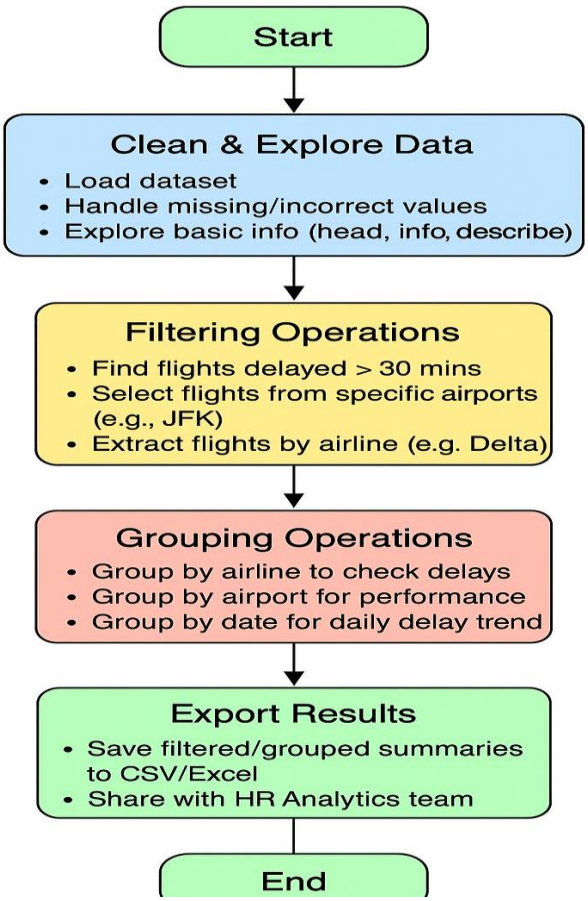
Benefits

1. **Improved Operational Efficiency** – Pinpointing frequent delay causes enables smoother scheduling, reduced turnaround times, and higher on-time performance.
2. **Better Resource Allocation** – Insights help HR plan workforce shifts, align crew schedules, and optimize aircraft usage based on high-delay zones.
3. **Cost Reduction** – Minimizing delays reduces operational expenses such as fuel wastage, crew overtime, and passenger compensation, while also improving airline profitability.
4. **Strategic Planning** – Reliable analysis supports future decision-making such as preventive maintenance scheduling, employee training, and policy changes.
5. **Enhanced Accountability** – Clear categorization of delays creates transparency, making it easier to assign responsibility and monitor improvement across departments.

Tools & Technologies

1. **Programming Language**:
    - **Python** – for data manipulation and analysis.
2. **Data Analysis Library**:
    - **Pandas** – for filtering, grouping, and aggregating flight delay data.
3. **Development Environment**:
    - **VS Code** – for writing and running Python scripts interactively.
4. **Data Storage**:
    - **CSV files** – the flight records dataset can be stored in CSV or Excel format

UML Diagrams



File Structure:

PROJECT/

```
│
├── data/
│   ├── flights.csv                # Original dataset with all flight records
├── templates/
│   └── index.html                 # HTML template for the dashboard or web app
│
├── analysis.py                    # Python script for data cleaning, filtering, grouping, summary
├── app.py                         # Python script for Flask web app integration (dashboard/API)
└── requirements.txt               # Required Python libraries (pandas, Flask, etc.)
```

Code Used In Projects

App.py

1. **Imports & Setup**

```
1   from flask import Flask, render_template, request, send_file, jsonif
2   import pandas as pd
3   from io import BytesIO, StringIO
4   import os
5
6   app = Flask(__name__)
```

- Import Flask and Pandas.
- BytesIO and StringIO → for in-memory CSV handling.
- Os → to check if dataset file exists.
- Create a Flask app instance.

2. **Load Dataset**

```
CSV_PATH = "data/flights.csv"
if os.path.exists(CSV_PATH):
    df = pd.read_csv(CSV_PATH)
else:
    # fallback sample so app won't crash while you test
    sample = [
        {"FlightID": "AF001", "Airline": "Delta", "OriginAirport": "JFK", "DestinationAirport": "LAX", "Date": "2023-01-05", "De
        {"FlightID": "F002", "Airline": "United", "OriginAirport": "ORD", "DestinationAirport": "SFO", "Date": "2023-01-05", "De
        {"FlightID": "F003", "Airline": "Delta", "OriginAirport": "ATL", "DestinationAirport": "MIA", "Date": "2023-01-06", "Dep
        {"FlightID": "F004", "Airline": "American", "OriginAirport": "DFW", "DestinationAirport": "LAX", "Date": "2023-01-06", "
        {"FlightID": "F005", "Airline": "Southwest", "OriginAirport": "LAX", "DestinationAirport": "LAS", "Date": "2023-01-07",
    ]
    df = pd.DataFrame(sample)
```

- Tries to load a real dataset from data/flights.csv.
- If missing, uses a **fallback sample dataset** so the app still works.

3. **Clean & Normalize Data**

```
23  # normalize column names
24  df.columns = df.columns.str.strip()
25
26  # ensure required columns exist
27  for c in ["Airline", "OriginAirport", "DestinationAirport", "Date", "DepartureDelay", "ArrivalDelay"]:
28      if c not in df.columns:
29          if c in ("DepartureDelay", "ArrivalDelay"):
30              df[c] = 0
31          else:
32              df[c] = ""
33
34  # convert types
35  df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
36  df["DepartureDelay"] = pd.to_numeric(df["DepartureDelay"], errors="coerce").fillna(0)
37  df["ArrivalDelay"] = pd.to_numeric(df["ArrivalDelay"], errors="coerce").fillna(0)
```

- Strips spaces in column names.
- Ensures required columns exist.
- Converts Date → datetime, delays → numeric.

4. **Dropdown Choices**

```python
def _choices(series):
    return sorted([str(x).strip() for x in series.dropna().unique() if str(x).strip()])

AIRLINES = _choices(df["Airline"])
ORIGINS = _choices(df["OriginAirport"])
```

- Helper function _choices → extracts **unique values** for dropdowns.
- Saves available **airlines** and **origin airports** for frontend filters.

5. Summary Computation Function

```python
# compute summary helper (used for both view and download)
def compute_summary(option: str, airline_filter: str = None, origin_filter: str = None, top_n_rou
    base = df.copy()

    # apply filters (case-insensitive)
    if airline_filter:
        base = base[base["Airline"].astype(str).str.strip().str.casefold() == str(airline_filter)
    if origin_filter:
        base = base[base["OriginAirport"].astype(str).str.strip().str.casefold() == str(origin_fi

    if option == "airline":
        res = base.groupby("Airline", dropna=False).agg(
            AverageArrivalDelay=("ArrivalDelay", "mean"),
            TotalFlights=("Airline", "size")
        ).reset_index()
        res["AverageArrivalDelay"] = res["AverageArrivalDelay"].round(2)

    elif option == "airport":
        temp = base.groupby("OriginAirport", dropna=False).agg(
            TotalFlights=("OriginAirport", "size"),
            DelayedFlights=("ArrivalDelay", lambda x: (x > 30).sum())
        ).reset_index()
        temp["PercentDelayed"] = ((temp["DelayedFlights"] / temp["TotalFlights"]) * 100).fillna(0
        res = temp.drop(columns=["DelayedFlights"])
```

- Applies **filters** (airline/origin).
- Groups data differently depending on option:
  - "airline" → avg delay per airline.
  - "airport" → % delayed flights per airport.
  - "date" → delays per date.
  - "route" → delays per route.

6. Main Route (/)

```python
@app.route("/", methods=["GET", "POST"])
def index():
    records = []
    columns = []
    operation = None

    if request.method == "POST":
        choice = request.form.get("choice")

        # Filtering choices
        if choice == "delayed":
            result_df = df[df["ArrivalDelay"] > 30].reset_index(drop=True)
            operation = "Flights delayed more than 30 minutes"

        elif choice == "filter_airline":
            airline = request.form.get("value")
            result_df = df[df["Airline"].astype(str).str.strip().str.casefold() == str(airline).strip().casefold()].reset_index(c
            operation = f"Flights by Airline: {airline}"
```

- Handles **form submissions** (POST).
- Depending on the selected option (choice), it:
  - Filters flights (delayed, filter_airline, filter_origin).
  - Groups flights (group_airline, group_airport, group_date, group_route).
- Converts results into a list of dictionaries → so Jinja template can render them in a table.

7. Download CSV Summaries

```python
@app.route("/download/<summary_type>")
def download(summary_type):
    if summary_type not in ("airline", "airport", "date", "route"):
        return jsonify({"error": "invalid summary type"}), 400

    if summary_type == "route":
        df_summary = compute_summary("route", top_n_routes=None)  # full route summary
    else:
        df_summary = compute_summary(summary_type)

    buf = StringIO()
    df_summary.to_csv(buf, index=False)
    buf.seek(0)
    return send_file(
        BytesIO(buf.getvalue().encode("utf-8")),
        mimetype="text/csv",
        as_attachment=True,
        download_name=f"{summary_type}_summary.csv"
    )
```

- Lets users download grouped summaries as **CSV files**.
- Example: /download/airline → downloads airline summary.

8. Run the App

```
if __name__ == "__main__":
    app.run(debug=True)
```

- Starts Flask app in debug mode.

Analysis.py

1. Load Dataset

```
# --- 1. Load dataset ---
df = pd.read_csv("data/flights.csv", sep=",")

# Strip whitespace from column names
df.columns = df.columns.str.strip()
```

- Imports the pandas library for data manipulation.
- Reads the CSV file flights.csv into a DataFrame df.
- Strips any extra whitespace from column names to avoid key errors later.

2. Explore Data

```
# --- 2. Explore ---
print("\n--- First 5 Rows ---")
print(df.head())

print("\n--- Info ---")
df.info()  # don't wrap with print()

print("\n--- Describe ---")
print(df.describe())
```

- df.head() shows the first 5 rows to get a quick look at the data.
- df.info() displays column types, non-null counts, and memory usage.
- df.describe() provides summary statistics for numeric columns like mean, min, max, etc.

3. Data Cleaning / Type Conversion

```
# --- 3. Data cleaning / type conversion ---
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
df['DepartureDelay'] = pd.to_numeric(df['DepartureDelay'], errors='coerce').fillna(0)
df['ArrivalDelay'] = pd.to_numeric(df['ArrivalDelay'], errors='coerce').fillna(0)
df = df.dropna(subset=['Date'])  # remove invalid dates
```

- Converts the Date column to datetime format, invalid dates become NaT.
- Converts delay columns to numeric type; invalid entries are replaced with 0.
- Removes rows with missing Date values to avoid errors in analysis.

4. Filtering Operations

```
# --- 4. Filtering Operations ---
delayed_30 = df[df['ArrivalDelay'] > 30]
from_JFK = df[df['OriginAirport'] == 'JFK']
delta_flights = df[df['Airline'] == 'Delta']
```

- delayed_30 stores flights delayed more than 30 minutes.
- from_JFK stores flights originating from JFK airport.
- delta_flights stores all flights operated by Delta Airlines.

5.Grouping Operations

5.1 By Airline

```
# 5.1 By Airline
airline_summary = df.groupby('Airline').agg(
    AverageArrivalDelay=('ArrivalDelay', 'mean'),
    AverageDepartureDelay=('DepartureDelay', 'mean'),
    TotalFlights=('Airline', 'size')  # safer than FlightID
).reset_index()
```

- Groups flights by Airline.
- Calculates average arrival & departure delays.

- Counts total flights per airline.

5.2 By Origin Airport

```
# 5.2 By OriginAirport
airport_summary = df.groupby('OriginAirport').agg(
    TotalFlights=('OriginAirport', 'size'),
    DelayedFlights=('ArrivalDelay', lambda x: (x > 30).sum())
).reset_index()
airport_summary['PercentDelayed'] = (
    airport_summary['DelayedFlights'] / airport_summary['TotalFlights']
).replace([float("inf")], 0).round(2)
airport_summary.drop(columns=['DelayedFlights'], inplace=True)
```

- Groups flights by OriginAirport.
- Counts total flights per airport.
- Calculates the percentage of flights delayed over 30 minutes.

5.3 By Date

```
# 5.3 By Date
daily_summary = df.groupby('Date').agg(
    AverageArrivalDelay=('ArrivalDelay', 'mean'),
    TotalFlights=('Airline', 'size')
).reset_index()
```

- Groups flights by Date.
- Calculates daily average arrival delay and total flights.

5.4 By Route

```
# 5.4 By Route
route_summary = df.groupby(['OriginAirport', 'DestinationAirport']).agg(
    AverageArrivalDelay=('ArrivalDelay', 'mean'),
    TotalFlights=('OriginAirport', 'size')
).reset_index()
```

- Groups flights by origin-destination pairs (routes).
- Calculates average arrival delay and number of flights per route.

6.Export Summaries

```
# --- 6. Export summaries ---
airline_summary.to_csv('airline_summary.csv', index=False)
airport_summary.to_csv('airport_summary.csv', index=False)
route_summary.to_csv('route_summary.csv', index=False)

print("\n--- Summaries exported successfully! ---")
```

- Exports the grouped summary DataFrames to CSV files for further analysis or reporting.
- index=False ensures row indices are not included in the CSV.

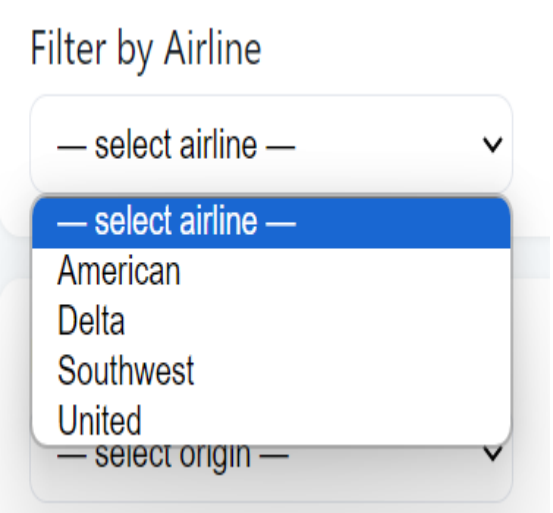
Webpage Dashboard Interface

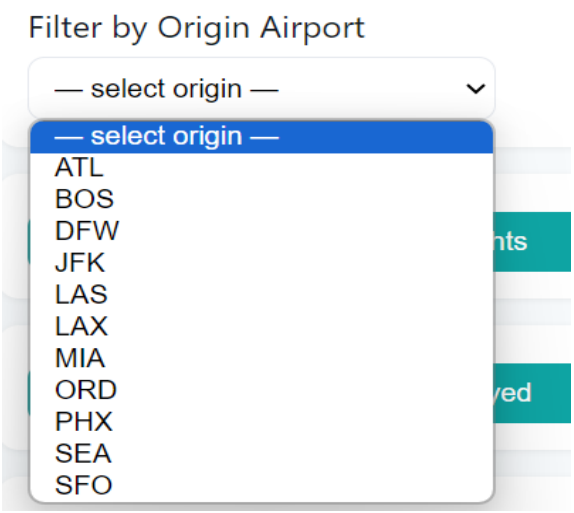
Delayed flights

Result: Flights delayed more than 30 minutes

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
F020	American	LAX	DFW	2023-01-14	50	45	1235
F024	American	JFK	MIA	2023-01-16	65	60	1090
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

Selecting Airport and Airline





Total Flights Delayed

Result: Grouped by OriginAirport → Total Flights & % Delayed

OriginAirport	TotalFlights	PercentDelayed
ATL	6	0.0
BOS	2	0.0
DFW	4	25.0
JFK	8	37.5
LAS	4	0.0
LAX	11	63.64
MIA	3	0.0
ORD	5	60.0
PHX	2	0.0
SEA	1	0.0
SFO	4	50.0

Top 5 Routes

Result: Top 5 Routes with Highest Average Delay

OriginAirport	DestinationAirport	AverageArrivalDelay	TotalFlights
LAX	JFK	86.67	3
SFO	JFK	72.5	2
JFK	MIA	60.0	1
LAX	SEA	55.0	1
LAX	ATL	50.0	1

Command Prompt Output

```
E:\Learning\project>python analysis.py

--- First 5 Rows ---
  FlightID    Airline OriginAirport DestinationAirport        Date  DepartureDelay  ArrivalDelay   Dis
0     F001      Delta           JFK                LAX  2023-01-05              15            10  2475
1     F002     United           ORD                SFO  2023-01-05              45            50  1846
2     F003      Delta           ATL                MIA  2023-01-06              -5             0   595
3     F004   American           DFW                LAX  2023-01-06              60            55  1235
4     F005  Southwest           LAX                LAS  2023-01-07              10             5   236

--- Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   FlightID            50 non-null     object
 1   Airline             50 non-null     object
 2   OriginAirport       50 non-null     object
 3   DestinationAirport  50 non-null     object
 4   Date                50 non-null     object
 5   DepartureDelay      50 non-null     int64
 6   ArrivalDelay        50 non-null     int64
 7   Dis                 50 non-null     int64
dtypes: int64(3), object(5)
memory usage: 3.3+ KB
```

- The dataset has **50 flight records** with 8 columns:
  FlightID, Airline, OriginAirport, DestinationAirport, Date, DepartureDelay, ArrivalDelay, Distance.
- Data types:
  - object → Categorical data (Airline, Airports, Date, FlightID).
  - int64 → Numerical data (DepartureDelay, ArrivalDelay, Distance).
- Sample entries show flight details like delays and distances.
- Dataset is **clean and complete (no missing values)**.

Closure of the project

In this project, we analyzed an airline’s flight records using Python and Pandas to uncover patterns and insights related to flight delays. The dataset was thoroughly cleaned, standardized, and converted into appropriate formats to ensure accuracy. This enabled efficient filtering, grouping, and aggregation operations. By examining delays across airlines, airports, routes, and dates, we identified critical trends, such as which airlines and airports consistently experience higher delays, peak times with frequent disruptions, and the overall percentage of flights affected.

The analysis highlights actionable insights for the HR Analytics team to improve decision-making. These include optimizing flight schedules, allocating resources more effectively, and addressing delay-prone routes or carriers. Additionally, comparative and distance-based analyses provided a deeper understanding of performance variations across short-haul and long-haul flights.

To ensure usability, summarized tabular reports were generated and exported in CSV format, making the results easily shareable with stakeholders. The findings not only help improve operational efficiency but also support strategies to enhance passenger satisfaction and reliability of services.