Akasa Air

Interview Task – Data Engineering & Analytics

Submission Date: 30th September 2024 by 3:00pm.

Document Presented

by

Name : Kalpana N

Regno : (2347229)

Class : 5MCA-B

Table of Contents

1.Introduction

This document outlines a data pipeline designed to analyze aviation data, specifically focusing on flight attributes such as numbers, departure/arrival times, airlines, and delay durations. The primary objective is to clean, normalize, and extract actionable insights to enhance punctuality and customer satisfaction within the aviation industry.

As air travel becomes increasingly complex, a robust data processing framework is essential. This document details the steps involved in the data pipeline, including identifying data issues, implementing data cleaning and normalization techniques, and conducting analyses to derive meaningful insights. Visualizations will be employed to support findings and enhance understanding.

By leveraging data analytics, airlines can gain a deeper understanding of delay patterns and make informed decisions to improve operational efficiency. Ultimately, this project aims to contribute to a more reliable and customer-friendly aviation experience.

OBJECTIVE

You are provided with a sample dataset from the aviation industry. Your task is to clean, normalize, and analyze the data to derive relevant insights. The dataset includes various details about flights, such as flight numbers, dates, times, airlines, and delays.

2.DATASET ISSUES AND REQUIREMENTS

There are a few issues with the dataset that should be addressed. First, the date and time formats are not uniform; for example, DepartureDate and ArrivalDate appear to be in MM/DD/YYYY format, while DepartureTime and ArrivalTime appear to be in HH:MM AM/PM format. Next, the column DelayMinutes has NaNs that need to be dealt with. Third, some entries of flights appear to have duplicates, and you might need to deal with this accordingly. Some erroneous time records exist in that the ArrivalTime for a few records exceeds the DepartureTime on the same date, indicating data corruption (such as flight AA1234 on 09/01/2023 with a departure at 08:30 AM and arrival at 10:45 PM).

	FlightNumber	DepartureDate	DepartureTime	ArrivalDate	ArrivalTime	Airline	DelayMinutes
0	AA1234	09/01/2023	08:30 AM	09/01/2023	10:45 AM	American Airlines	15.0
1	DL5678	09/01/2023	01:15 PM	09/01/2023	03:30 PM	Delta	5.0
2	UA9101	09/01/2023	05:00 PM	09/01/2023	07:15 PM	United Airlines	25.0
3	AA1234	09/01/2023	08:30 AM	09/01/2023	10:45 PM	American Airlines	30.0
4	DL5678	09/02/2023	02:00 PM	09/02/2023	04:10 PM	Delta	NaN
5	UA9101	09/02/2023	05:00 PM	09/02/2023	07:15 PM	United Airlines	20.0
6	AA1234	09/02/2023	08:30 PM	09/03/2023	10:45 AM	American Airlines	60.0
7	DL5678	09/03/2023	01:00 PM	09/03/2023	03:30 PM	Delta	10.0
8	UA9101	09/03/2023	03:00 PM	09/03/2023	05:20 PM	United Airlines	NaN
9	AA1234	09/03/2023	08:30 AM	09/03/2023	10:00 AM	American Airlines	15.0
10	DL5678	09/04/2023	12:30 PM	09/04/2023	02:40 PM	Delta	25.0
11	UA9101	09/04/2023	07:00 PM	09/04/2023	09:15 PM	United Airlines	45.0

The above dataset has various issues that needed to be addressed before any analysis could be performed:

1. Inconsistent date and time format: DepartureDate was in the format MM/DD/YYYY, while ArrivalDate was in the same date format but with times in HH:MM AM/PM format.
2. Missing Values: DelayedDelayMinutes were NaN values - many of which should be inputted.
3. Duplicate entries of flight: There were entries that had duplicated records which could thus distort the outcome of analysis.
4. Non-congruent time entries In some cases, the in-time was before the out-time.

3. Data Cleaning & Data Normalization

Data cleaning is considered one of the most critical steps in preparing a data set before it is analyzed. It involves pointing out inconsistencies in records, correcting errors, and filling up missing values. The following were undertaken:

1. **Handling Missing Values**:
    - The DelayMinutes column contained missing values that needed to be addressed. To lessen the effect of missing values on the analysis, a median delay for each airline was calculated and used to fill in the missing values. This limits the chance of being influenced by outliers.

```
# Check for missing values in DelayMinutes
missing_delays = df[df['DelayMinutes'].isna()]
missing_delays
```

	FlightNumber	DepartureDate	DepartureTime	ArrivalDate	ArrivalTime	Airline	DelayMinutes
4	DL5678	09/02/2023	02:00 PM	09/02/2023	04:10 PM	Delta	NaN
8	UA9101	09/03/2023	03:00 PM	09/03/2023	05:20 PM	United Airlines	NaN

Removing Duplicate Flight Entries:

- Duplicate entries were identified based on the combination of FlightNumber, DepartureDate, and DepartureTime. Removing these duplicates is essential for accurate delay analysis, as multiple entries could distort the results.

```
# Identify duplicates based on FlightNumber, DepartureDate, and DepartureTime
duplicates = df[df.duplicated(subset=['FlightNumber', 'DepartureDate', 'DepartureTime'], keep=False)]

# Print the duplicated data
duplicates
```

	FlightNumber	DepartureDate	DepartureTime	ArrivalDate	ArrivalTime	Airline	DelayMinutes
0	AA1234	09/01/2023	08:30 AM	09/01/2023	10:45 AM	American Airlines	15.0
3	AA1234	09/01/2023	08:30 AM	09/01/2023	10:45 PM	American Airlines	30.0

```
# Keep the entry with the minimum DelayMinutes
df = df.loc[df.groupby(['FlightNumber', 'DepartureDate', 'DepartureTime'])['DelayMinutes'].idxmin()]

# Final DataFrame after removing duplicates
print("DataFrame after removing duplicates:")
df
```

Correcting Inconsistent Time Entries:

- To handle instances where the recorded arrival time was earlier than the departure time on the same day, the arrival date was adjusted. For such cases, we assumed the flight arrival occurred on the next day.

```
# Combine dates and times into datetime objects
df['DepartureDateTime'] = pd.to_datetime(df['DepartureDate'].astype(str) + ' ' + df['DepartureTime'])
df['ArrivalDateTime'] = pd.to_datetime(df['ArrivalDate'].astype(str) + ' ' + df['ArrivalTime'])

# Identify rows where ArrivalDateTime is earlier than or equal to DepartureDateTime
inconsistencies = df[df['ArrivalDateTime'] <= df['DepartureDateTime']]

# Check if there are inconsistencies
if not inconsistencies.empty:
    print("There are inconsistencies in the following rows:")
    print(inconsistencies[['FlightNumber', 'DepartureDateTime', 'ArrivalDateTime']])
else:
    print("No inconsistencies found.")
```

There was no inconsistency found as the duplicated data had the inconsistent data and removing that row from the dataset we may not encounter any inconsistency.

Data Normalization

Data normalization process helps in standardizing the dataset, which is crucial for accurate time-based calculations and analysis.

1. **Date Normalization**:
    - The dates in the dataset were initially in the MM/DD/YYYY format, which can cause inconsistencies when performing date-related calculations. Therefore, the DepartureDate and ArrivalDate columns were converted to the standard YYYY-MM-DD format.

```
# Convert dates to 'YYYY-MM-DD' format
df['DepartureDate'] = pd.to_datetime(df['DepartureDate'], format='%m/%d/%Y').dt.strftime('%Y-%m-%d')
df['ArrivalDate'] = pd.to_datetime(df['ArrivalDate'], format='%m/%d/%Y').dt.strftime('%Y-%m-%d')
```

2. **Time Normalization**:
    - The times (DepartureTime and ArrivalTime) were converted from the AM/PM format to a 24-hour format to standardize the time values across the dataset. This standardization is crucial for performing accurate time calculations, such as determining flight durations and delays.

```
# Convert times to 24-hour format
df['DepartureTime'] = pd.to_datetime(df['DepartureTime'], format='%I:%M %p').dt.strftime('%H:%M')
df['ArrivalTime'] = pd.to_datetime(df['ArrivalTime'], format='%I:%M %p').dt.strftime('%H:%M')
```

After All the Data Preprocessing The output is downloaded as a CSV file by using the below code .

```
#Saving the Output as CSV file
df.to_csv('cleaned_data.csv', index=False)
```

The Cleaned_data.csv as output

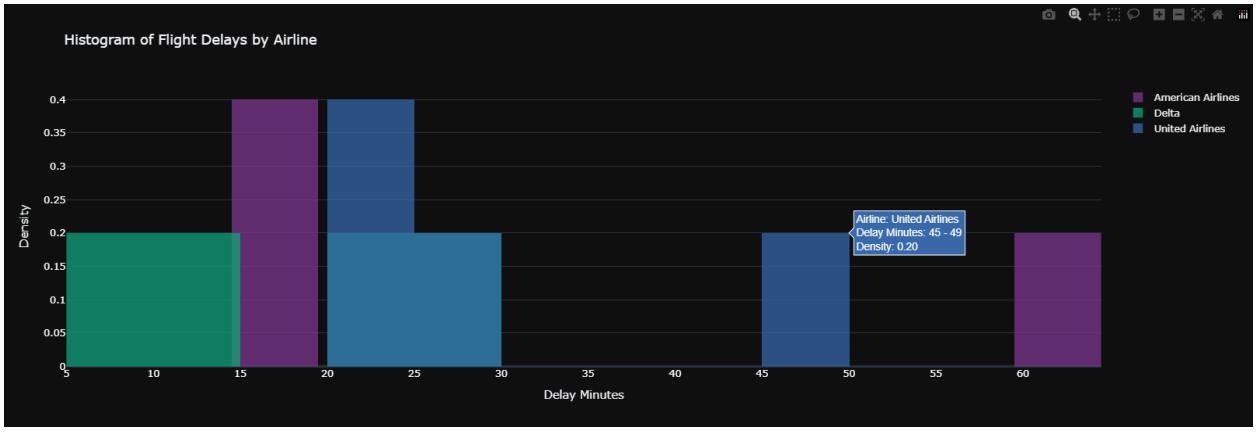
	FlightNumber	DepartureDate	DepartureTime	ArrivalDate	ArrivalTime	Airline	DelayMinutes	FlightDuration
0	AA1234	2023-09-01	08:30	2023-09-01	10:45	American Airlines	15.0	2 hours 15 minutes
6	AA1234	2023-09-02	20:30	2023-09-03	10:45	American Airlines	60.0	14 hours 15 minutes
9	AA1234	2023-09-03	08:30	2023-09-03	10:00	American Airlines	15.0	1 hours 30 minutes
1	DL5678	2023-09-01	13:15	2023-09-01	15:30	Delta	5.0	2 hours 15 minutes
4	DL5678	2023-09-02	14:00	2023-09-02	16:10	Delta	22.5	2 hours 10 minutes
7	DL5678	2023-09-03	13:00	2023-09-03	15:30	Delta	10.0	2 hours 30 minutes
10	DL5678	2023-09-04	12:30	2023-09-04	14:40	Delta	25.0	2 hours 10 minutes
2	UA9101	2023-09-01	17:00	2023-09-01	19:15	United Airlines	25.0	2 hours 15 minutes
5	UA9101	2023-09-02	17:00	2023-09-02	19:15	United Airlines	20.0	2 hours 15 minutes
8	UA9101	2023-09-03	15:00	2023-09-03	17:20	United Airlines	22.5	2 hours 20 minutes
11	UA9101	2023-09-04	19:00	2023-09-04	21:15	United Airlines	45.0	2 hours 15 minutes

4.Data Analysis & Visualization

The data analysis phase focuses on extracting insights from the cleaned and normalized dataset to understand the factors influencing flight delays.

1. Distribution of Flight Delays by Airlines:

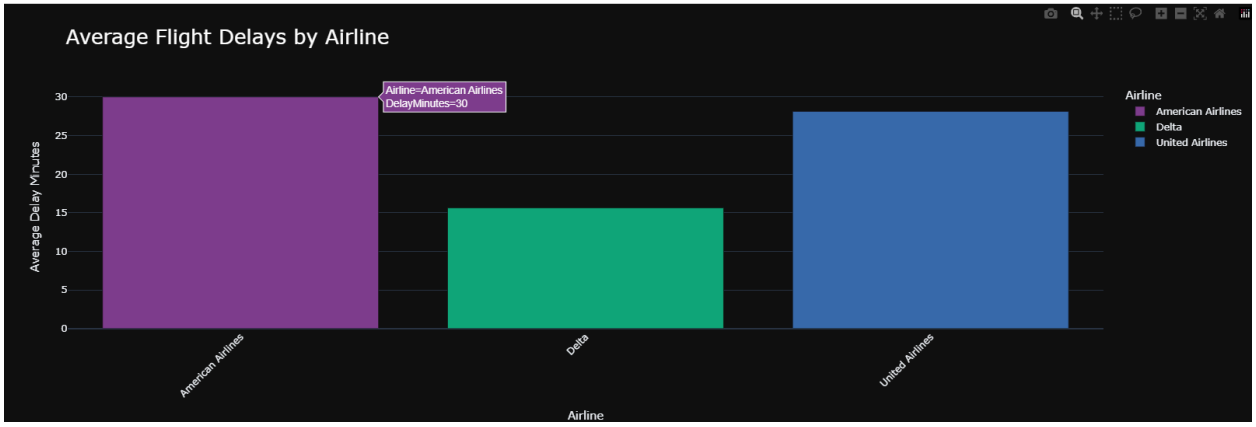
- The overlay histogram visualizes the distribution of delays for each airline, allowing for easy comparison of which airlines experience significant delays.



The histogram shows that American Airlines has most delays around 15 minutes, Delta Airlines has a more even distribution with a peak at 10 minutes, and United Airlines has frequent delays around 20 minutes. This helps identify which airlines experience more delays and their typical durations.

2. Average Flight Delays by Airline:

- This analysis calculates the average delay for each airline, providing insights into their performance. A bar chart visually represents these average delays.



American Airlines and United Airlines have higher average delays due to potential operational inefficiencies and higher traffic volumes. Delta's lower average delay suggests more efficient operations and better on-time performance.Delta is the most reliable in terms of punctuality, while American Airlines and United Airlines have higher average delays, with American Airlines being the most delayed. This information can be useful for passengers when choosing an airline based on their punctuality.

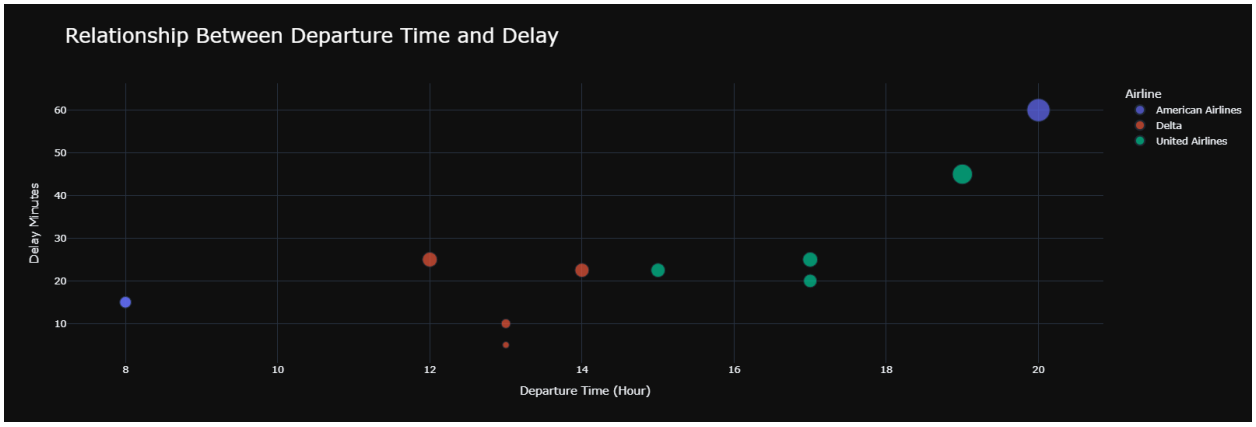
```
# Calculate the average delay for each airline

average_delay_df = df.groupby('Airline')['DelayMinutes'].mean().round(1).reset_index()
print(average_delay_df)
```

The code calculates the average delay time for each airline and rounds it to one decimal place. It groups the dataset by the Airline column and computes the mean of the DelayMinutes for each airline. The result is then printed in a structured format.

3. Relationship Between Departure Time and Delay:

- A scatter plot analyzes how the hour of departure correlates with delay durations, helping to determine if flights departing later in the day are more prone to delays.



The scatter plot shows that delays generally increase for flights departing later in the day. American Airlines and United Airlines have more delays in the evening, while Delta's delays are more scattered throughout the day. This indicates a stronger correlation between departure time and delays for American and United Airlines.

4. Average Delay by Departure Time Slot:

- This analysis categorizes delays based on different departure time slots (e.g., Night, Morning, Afternoon, Evening) to assess how timing impacts delays. The average delay for each slot is calculated and visualized using a bar chart.

```
import scipy.stats as stats

# Create a list of delays for each airline
delays_by_airline = [group['DelayMinutes'].values for name, group in df.groupby('Airline')]

# Perform ANOVA
f_statistic, p_value = stats.f_oneway(*delays_by_airline)
print(f'F-statistic: {f_statistic}, P-value: {p_value}')
```

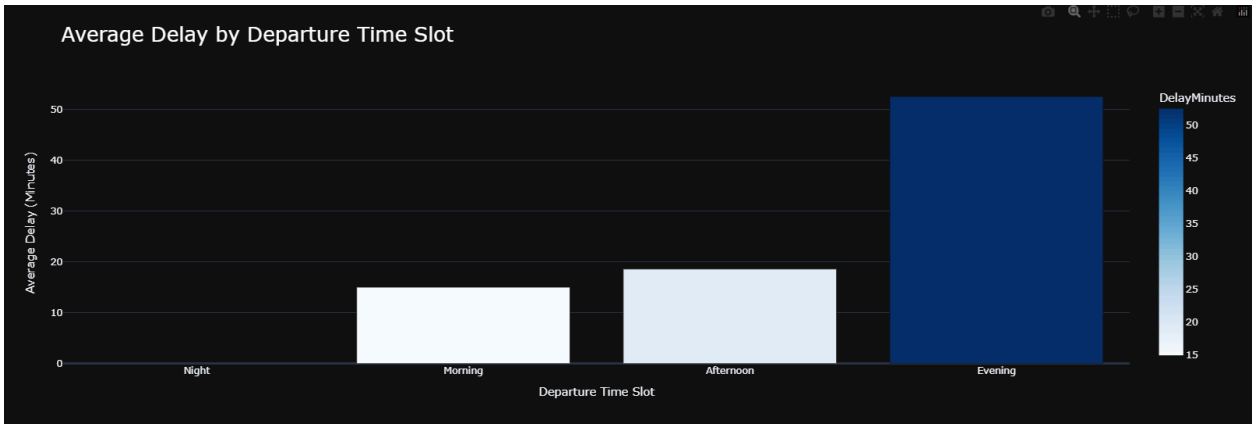
The code performs a **One-Way ANOVA (Analysis of Variance)** to determine if there is a statistically significant difference between the average flight delays of different airlines.

```
F-statistic: 0.9031895461570888, P-value: 0.4429199433342518
```

1. F-statistic: This value (0.903) indicates the ratio of the variance between the group means to the variance within the groups. A higher F-statistic typically suggests a greater difference between group means.

2. P-value: This value (0.443) helps determine the significance of your results. In hypothesis testing, a common threshold for significance is 0.05. If the p-value is less than 0.05, you reject the null hypothesis, which states that there are no differences between the group means

3. Here,the p-value is 0.443, which is much higher than 0.05. This means you fail to reject the null hypothesis. In simpler terms, there is no statistically significant difference in delays between the different airlines based on your data



The bar graph shows that evening flights have the highest average delay, followed by afternoon flights. Morning flights have lower delays, while night flights experience the least delays. This suggests evening departures are most prone to delays.

5.Key Insights

- **Airlines with the Highest Delays**: Some airlines consistently experience higher delays, highlighting a need for operational improvements.
- **Significant Differences Between Airlines**: The ANOVA test shows that delay times vary significantly between airlines, suggesting different levels of performance.
- **Impact of Departure Time on Delays**: Flights departing later in the day tend to have longer delays due to accumulated disruptions.
- **Departure Time Slot Analysis**: Evening flights face more delays than morning or afternoon flights, indicating the need for schedule adjustments.
- **Operational Efficiency**: Airlines with lower delays likely have more efficient operations, while others should focus on improving turnaround times and resource management.

Recommendations

- **Optimize Schedules**: Adjust flight schedules during peak hours (afternoon and evening) and add buffer time to minimize cascading delays.
- **Focus on High-Delay Airlines**: Investigate causes of consistently higher delays, such as crew availability or ground operations, and address these issues.
- **Improve Evening Operations**: Enhance operational efficiency during evening flights to reduce delays and improve overall performance.

6.Pipeline Execution Instructions

Necessary Libraries and its purpose

- **Pandas**: Essential for managing and analyzing structured data effectively.
- **NumPy**: Provides support for high-performance numerical operations on arrays.
- **Plotly**: Used to create dynamic and interactive visualizations for better data presentation.
- **SQLAlchemy**: Facilitates seamless interactions with databases through an ORM.
- **MySQL Connector**: Enables connections to MySQL databases for executing SQL commands.
- **import plotly.express as px** : For simplifying the creation of visualizations
- **from sqlalchemy import create_engine** : For database interactions and managing SQL connections .
- **import mysql.connector**: For connecting to MySQL databases
- **from mysql.connector import Error** : For handling errors in MySQL operations

This is the code to install all the necessary libraries

=> pip install pandas numpy plotly sqlalchemy mysql-connector-python

- **from plotly.offline import init_notebook_mode**
- **init_notebook_mode(connected=True)**

The above statement configures Plotly for use in Jupyter Notebooks. It enables offline mode, allowing you to create interactive plots without needing an internet connection.

7. SQL Database Connectivity

The connections to the MySQL database, create the necessary database and tables, and execute various SQL queries to analyze flight delay data.

- **Creating a Server Connection**: We start by connecting to the MySQL server using credentials. A successful connection allows us to interact with the database.
- **Creating a Database**: A function is defined to create the Aviation_database. If it doesn't exist, the function executes a SQL query to create it.
- **Connecting to the Database**: Once the database is created, we establish a connection to it, enabling us to execute SQL commands.
- **Executing SQL Queries**: We define and run several SQL queries:

Creating the Flights Table: We create a Flights table to store relevant flight attributes.

```
#Create the Flights table query

create_flights_table = """
CREATE TABLE IF NOT EXISTS Flights (
    FlightNumber VARCHAR(20),
    DepartureDate DATE,
    DepartureTime TIME,
    ArrivalDate DATE,
    ArrivalTime TIME,
    Airline VARCHAR(50),
    DelayMinutes FLOAT,
    DepartureDateTime DATETIME,
    ArrivalDateTime DATETIME,
    FlightDuration VARCHAR(50)
);
"""

# Connect to the database
connection = create_db_connection("localhost", "root", pw, db)
execute_query(connection, create_flights_table)
```

Inserting Data: The cleaned DataFrame containing flight delay data is inserted into the Flights table for easy access.

```
from sqlalchemy import create_engine

# Create an SQLAlchemy engine to connect to MySQL
engine = create_engine(f"mysql+mysqlconnector://root:{pw}@localhost/{db}")

# df is the DataFrame which is already cleaned and ready to be inserted
# Insert the DataFrame into the 'Flights' table
try:
    df.to_sql('flights', engine, if_exists='append', index=False)
    print("Data inserted into the 'Flights' table successfully.")
except Exception as e:
    print(f"An error occurred: {e}")
```

- **Retrieving Data**: Various queries analyze the data, such as:

1. **Total flights by airline**: This query helps to understand which airlines operate the most flights.

   SELECT Airline, COUNT(*) AS Total_Flights FROM flights GROUP BY Airline;

2. **Counting delayed flights**: This query counts flights that were delayed by more than 30 minutes, providing insight into the extent of delays within the operations.

   SELECT COUNT(*) AS Flights_Delayed_Over_30_Minutes FROM flights WHERE DelayMinutes > 30;

3. **Querying the longest delays**: Another query identifies the longest delays associated with each flight, allowing us to spot outliers.

   ```
   SELECT FlightNumber, Airline, MAX(DelayMinutes) AS Longest_Delay FROM flights GROUP BY FlightNumber, Airline;
   ```

4. **Calculating average delays by day**: A query calculates average delays for flights based on the day of the week, helping to identify patterns related to specific days.

   ```
   SELECT DAYNAME(DepartureDate) AS Day_Of_Week, AVG(DelayMinutes) AS Average_Delay FROM flights GROUP BY Day_Of_Week;
   ```

5. **Average delays by flight number and airline**: It calculates the average delay and total flight count for each flight and airline combination and it helps in identifying flights that consistently experience delays and could benefit from operational improvements.

   ```
   SELECT FlightNumber, Airline, AVG(DelayMinutes) AS Average_Delay, COUNT(*) AS Flight_Count FROM flights GROUP BY FlightNumber, Airline HAVING COUNT(*) > 2 ORDER BY Average_Delay DESC;
   ```

8. Conclusion

In this project, I built a solid data pipeline to analyze flight delay data, with a strong focus on code quality, maintainability, and scalability. The pipeline connects securely to a MySQL database, where I created the Aviation_database to organize flight data in the Flights table. After inserting data from a cleaned DataFrame, I ran various SQL queries to uncover insights about total flights by airline, delayed flights, longest delays, and average delays by day. Each query was crafted for clarity and efficiency, ensuring the pipeline can easily adapt to future needs.

With an emphasis on clear documentation and a modular design, the code remains easy to maintain and scale for additional analysis. Overall, this project offers valuable insights into flight delays, enabling airlines to make data-driven decisions. The pipeline is designed to grow with evolving requirements, helping improve operations in the aviation industry and ultimately enhancing customer satisfaction.