CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO PROJECT:

In the dynamic environment of the aviation industry, providing exceptional customer experiences is essential for airlines to remain competitive and build long-lasting relationships with passengers. Airlines gather large volumes of data from multiple sources, including ticket bookings, passenger demographics, flight schedules, and customer feedback. However, the true value of this data can only be realized when it is transformed into meaningful insights that support decision-making and service improvement.

The project, **Passenger Pulse: Harnessing Data Analytics for Exceptional Customer Experiences in Aviation**, is centered on the effective use of data visualization to bridge the gap between raw data and actionable knowledge. By leveraging Power BI, a leading business intelligence and data visualization platform, this project converts complex and extensive datasets into interactive dashboards and visual reports. Power BI enables users to create real-time, customizable visualizations such as bar charts, line graphs, pie charts, and heat maps, making it easier to identify trends, patterns, and anomalies within the data.

The project workflow includes data extraction from various sources, data cleaning to ensure accuracy, and data integration to combine information into a unified dataset. Once the data is prepared, it is loaded into Power BI, where visually appealing and informative dashboards are designed. These dashboards allow airline managers and staff to monitor key performance indicators (KPIs) such as on-time performance, passenger satisfaction scores, and service quality metrics. Drill-down features and interactive filters provide users with the flexibility to explore specific aspects of the data, such as analyzing feedback by flight route or time period.

By presenting data in a clear and accessible format, the dashboards empower stakeholders to make informed decisions, prioritize improvements, and respond promptly to emerging issues. The use of technical features like data modeling, DAX (Data Analysis Expressions), and custom visuals further enhances the analytical capabilities of the dashboards. Ultimately, **Passenger Pulse** demonstrates how advanced data visualization using Power BI can transform airline operations, drive customer-centric strategies, and elevate the overall travel experience for passengers.

CHAPTER 2

LITERATURE SURVEY

2.1) SURVEY REVIEW 1:

Title: Exploratory Data Analysis and Prediction of Passenger Satisfaction with Airline services

Abstract:

The airline industry has seen rapid growth after the COVID-19 pandemic, making customer satisfaction more important than ever. This project explores the key factors that influence how passengers feel about their airline experiences. By analyzing data on passenger feedback, flight details, and service ratings, we identify what makes customers satisfied or unsatisfied. We use clear visualizations and dashboards in Power BI to highlight important trends and problem areas, making it easy for airline staff to understand and act on the findings. The results show that aspects like the boarding process and Wi-Fi services have a strong impact on satisfaction. Our project demonstrates how airlines can use data to better understand their passengers, improve their services, and create a more enjoyable flying experience. This approach helps airlines make smarter decisions and keep passengers coming back.

2.2) SURVEY REVIEW 2:

TITLE: Prediction of US airline passenger satisfaction using machine learning algorithms

Abstract:

The U.S. airline industry has faced major challenges due to the COVID-19 pandemic and increased competition, making customer satisfaction a top priority.

This project aims to predict passenger satisfaction using machine learning models and to identify which airline services most influence customer opinions. Using a dataset of over 129,000 passenger records with 22 different features, we analyze factors such as online boarding, inflight entertainment, seat comfort, onboard service, leg room, cleanliness, flight distance, and inflight Wi-Fi. Data preparation steps include cleaning, exploratory analysis, and feature selection. Several machine learning algorithms-such as K-Nearest Neighbors, Decision Tree, Logistic Regression, Random Forest, Naïve Bayes, and AdaBoost-are used to classify passengers as satisfied or unsatisfied. The Random Forest model performed best, achieving an accuracy of 89.2%. These results can help airlines focus on improving key services, attract more passengers, and enhance overall customer satisfaction through data-driven decisions.

2.3) SURVEY REVIEW 3:

TITLE: Are customers really satisfied with the Service Quality offered by the Aviation sector

ABSTRACT:

The proverb “By the inch it's a cinch; by the yard it's mighty hard” aptly captures the essence of this research, which methodically explores customer satisfaction with the service quality in the aviation sector at Swami Vivekananda Airport, Chhattisgarh. Notably, this airport experienced an 82% surge in passenger traffic in 2006, marking the highest growth in India. As one of the 35 non-metro airports being modernized by the Airport Authority of India, it serves as a significant case study for service quality assessment. This paper evaluates the range of services and facilities provided by Air India, Indigo, and Jet Airways through a structured passenger survey using a Likert Scale. By breaking down the research problem into manageable components, the study offers a detailed understanding of both

operational challenges and consumer satisfaction levels. The findings highlight key areas for improvement and provide actionable insights for enhancing service quality in the aviation sector.

CHAPTER -3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In the existing system, airlines often rely on traditional feedback forms and manual data collection to understand passenger satisfaction. This approach is slow, lacks real-time insights, and makes it difficult to identify key service issues quickly. Data is usually stored in separate databases, making analysis and reporting inefficient. Without advanced data analytics or visualization tools, airlines struggle to recognize patterns or predict customer needs. As a result, service improvements are reactive rather than proactive. The lack of integrated dashboards and automated reporting limits the ability of airline staff to make informed, data-driven decisions to enhance the passenger experience.

3.1.1 Limitations of Existing System:

1. **Delayed Insights:** Manual data collection causes slow analysis, making it hard to respond quickly to passenger issues.
2. **Data Silos:** Information is stored in separate databases, which prevents a unified view of passenger feedback and service quality.
3. **Limited Visualization:** Lack of advanced dashboards means staff cannot easily spot trends or problem areas.
4. **Reactive Approach:** Airlines only fix issues after complaints, instead of predicting and preventing them.
5. **Poor Decision Support:** Without real-time analytics, management cannot make fast, data-driven decisions to improve customer experience.

3.2 PROPOSED SYSTEM

Passenger Pulse is an advanced analytics platform that transforms airline passenger experience management. It aggregates data from booking systems, feedback channels, and service evaluations through an interconnected architecture, enabling seamless data sharing across domains. Power BI dashboards provide real-time visualization of key metrics like satisfaction scores, punctuality, and service quality, allowing teams to quickly identify trends and operational issues. Automated reporting minimizes manual effort and reduces errors. By leveraging distributed data access and predictive analytics, airlines can shift from reactive responses to proactive service management-anticipating passenger needs and addressing concerns before they escalate. This evidence-driven approach empowers management to make informed decisions, personalize services, and continuously enhance the passenger journey, ensuring a competitive advantage in the aviation sector.

3.2.1 Advantages of Proposed System:

1. **Distributed Data Integration:** Passenger information and feedback are accessed across multiple domains, ensuring robust data availability and reducing single points of failure.
2. **Real-Time Visualization:** Power BI dashboards provide instant access to operational KPIs, supporting dynamic monitoring of service quality and passenger satisfaction.
3. **Trend and Pattern Recognition:** Advanced analytics and interactive dashboards reveal operational trends, enabling rapid identification of strengths and weaknesses.
4. **Automated Reporting:** The platform automates report generation, reducing reliance on manual processes and minimizing errors.

5. **Predictive Decision Support:** Airlines can anticipate and address service disruptions proactively, enhancing overall service reliability.
6. **Personalized Passenger Services:** Insights derived from analytics enable the customization of services to align with the preferences and requirements of distinct passenger segments.
7. **Operational Competitiveness:** Enhanced passenger experience and operational efficiency drive customer loyalty and distinguish airlines in a crowded marketplace.

3.3 DOMAIN KNOWLEDGE:

Data Analytics in Aviation

Data analytics has emerged as a transformative force in the aviation industry, empowering organizations to harness vast amounts of operational and passenger data for informed decision-making. In this project, the focus is on leveraging data visualization tools, specifically Power BI, to analyze and present critical aviation metrics. The use of dashboards facilitates the interpretation of complex datasets, enabling stakeholders to monitor performance, identify trends, and drive strategic improvements without the direct application of artificial intelligence or machine learning techniques.

Key Components of Data Analytics in Aviation:

1. Operational Performance Monitoring:

Dashboards in Power BI allow for real-time tracking of key performance indicators (KPIs) such as on-time departures, turnaround times, and baggage handling efficiency. Visual representations of these metrics help airline managers quickly pinpoint operational bottlenecks and implement corrective actions to enhance service quality.

2. Passenger Satisfaction Analysis:

By visualizing survey results and feedback data, airlines can assess passenger satisfaction across various touchpoints, including check-in, boarding, in-flight services, and baggage claim. Power BI dashboards facilitate the segmentation of data by airline, flight, or time period, making it easier to identify areas requiring improvement.

3. Revenue and Load Factor Analysis:

Data analytics dashboards provide insights into revenue streams, ticket sales, and load factors. Visual tools help airlines monitor occupancy rates, identify peak travel periods, and optimize pricing strategies to maximize profitability.

4. Safety and Compliance Tracking:

Power BI dashboards can be used to visualize safety incidents, compliance checks, and maintenance schedules. This ensures that regulatory requirements are met and that safety standards are consistently upheld.

5. Resource Allocation:

Dashboards enable efficient monitoring of resource utilization, such as ground staff deployment, gate assignments, and equipment usage. This aids in optimizing workforce management and minimizing operational delays.

6. Trend Identification and Reporting:

Through interactive charts and graphs, Power BI dashboards help aviation professionals detect emerging trends, seasonal fluctuations, and recurring issues. These insights support data-driven decision-making and continuous process improvement.

CHAPTER 4

FEASIBILITY REPORT

4.1 INTRODUCTION

A feasibility report is a critical document that assesses the practicality and viability of a proposed project before significant resources are committed. For this report evaluates the technical, operational, and economic aspects to ensure successful implementation. The primary objective is to utilize data analytics and visualization tools to optimize service quality, enhance passenger satisfaction, and provide actionable insights for continuous improvement within the aviation sector.

4.2 Technical Feasibility

1. **Data Integration Capabilities:**
   The project leverages Power BI, a robust data visualization platform, which can seamlessly integrate with various aviation data sources such as passenger feedback systems, operational databases, and ticketing platforms.

2. **Scalability:**
   The technical architecture is designed to accommodate increasing data volumes as passenger traffic grows, ensuring long-term sustainability and adaptability to future requirements.

3. **User-Friendly Dashboards:**
   Power BI offers intuitive, interactive dashboards that enable aviation stakeholders to easily interpret complex datasets and make informed decisions in real time.

4. **Data Security and Compliance:**
   The system incorporates advanced security protocols and adheres to

industry standards, ensuring the confidentiality and integrity of sensitive passenger and operational data.

5. **Customizable Analytics:**
   The platform supports customizable reporting and analytics, allowing for tailored insights based on specific airline needs, operational metrics, and customer experience parameters.

4.3 Operational Feasibility

1. **Stakeholder Engagement:**
   The project is designed for seamless adoption by airline staff, management, and ground personnel, ensuring stakeholder buy-in and effective utilization.
2. **Minimal Disruption:**
   Implementation of the analytics system is planned to minimize disruption to existing workflows, with phased rollouts and comprehensive training sessions.
3. **Process Optimization:**
   By providing real-time insights into passenger satisfaction and operational efficiency, the system enables continuous process improvements and swift issue resolution.
4. **Resource Allocation:**
   Dashboards facilitate optimal allocation of resources such as staff, gates, and equipment, enhancing operational effectiveness and reducing bottlenecks.
5. **Scalable Support Structure:**
   The project includes ongoing technical support and system updates,

ensuring smooth operations and prompt troubleshooting as the system evolves.

4.4 Economic Feasibility

1. **Cost-Effective Implementation:**
   Utilizing Power BI reduces the need for expensive custom software development, lowering initial investment and ongoing maintenance costs.
2. **Return on Investment (ROI):**
   Enhanced passenger experiences and operational efficiencies are expected to drive higher customer retention, increased ticket sales, and improved brand reputation, resulting in measurable ROI.
3. **Operational Savings:**
   Data-driven decision-making leads to optimized staffing, reduced delays, and lower resource wastage, contributing to significant cost savings.
4. **Scalability of Costs:**
   The modular nature of the solution allows for incremental investment, aligning costs with business growth and evolving requirements.
5. **Competitive Advantage:**
   By harnessing advanced analytics, airlines can differentiate themselves in a competitive market, attracting more passengers and generating additional revenue streams.

CHAPTER 5

REQUIREMENTS SPECIFICATIONS

5.1 INTRODUCTION

Requirement specification is a critical phase in the development of the *Passenger Pulse* system. This stage outlines the essential functions, features, and technical needs required to build an effective data analytics platform for the aviation industry. The goal is to ensure that the system meets both user and business expectations by clearly defining what is needed for successful implementation. Key requirements include data integration from multiple sources, secure data storage, real-time visualization through Power BI dashboards, and automated reporting capabilities. The system must also support user-friendly interfaces for airline staff and management, as well as advanced filtering and drill-down features for in-depth analysis. By establishing clear and detailed requirements, the *Passenger Pulse* project aims to deliver a reliable, scalable, and efficient solution that empowers airlines to enhance customer experiences and maintain a strong competitive edge in the aviation market.

5.2 HARDWARE REQUIREMENTS

SI NO	NAME	DESCRIPTION
1	Processor	Intel I5 8th gen+
2	RAM	16GB and above
3	Storage	SSD 128GB+

5.3 SOFTWARE REQUIREMENTS

SI NO	NAME	DESCRIPTION
1	**Python (Version 3.9)**	Latest stable release, enhanced performance and security
2	**PostgreSQL (Version 17.1)**	Latest stable release, advanced data management and reliability.
3	**PowerBI Desktop**	Standalone application for creating interactive dashboards locally
4	**Pycharm (Version 2024.1.1)**	Recent IDE version, improved Python development experience.
5	**Docker (Version 4.41)**	Updated containerization platform for consistent deployment.
6	**Git/Github (Version 2.49.0)**	Latest version for version control and collaborative development.
7	**Apache Airflow (Version 2.10.5)**	Current supported version, robust workflow orchestration

CHAPTER 6

SYSTEM DEVELOPMENT ENVIRONMENT

6.1 System Configuration:

1. **OS Name:** Microsoft Windows 11 Home Single Language
2. **Version:** 11.0.22000 Build ueI14EE.
3. **BIOS Version/Date:** AMI F.01, 15-08-2023
4. **SMBIOS Version**: 4.0
5. **BIOS Mode:** UEFI
6. **Network Adapter –** Bridged Adapting Connection
7. **Browsing Modus as Chrome, Firefox, Brave etc.**
8. **Adequate Bandwidth.**
9. **SaaS –** Software as a Service Infrastructure support Driver.

6.2 IMPLEMENTATION:

The Passenger Pulse project leverages a modern data analytics stack to transform aviation data into actionable insights, visualized in Power BI dashboards. Below is a clear, detailed, and practical step-by-step guide to connecting, configuring, and using each tool and software version in your workflow, with a focus on seamless integration and output visualization in Power BI Desktop.

6.2.1 Data Extraction and Storage

Tools:

1. **Python (Version 3.9)**
2. **PostgreSQL (Version 17.1)**
3. **PyCharm (Version 2024.1.1)**

4. **Git/GitHub (Version 2.49.0)**

Process:

- Write Python scripts in PyCharm to extract passenger, operational, and feedback data from sources like CSV, APIs, and Excel files.
- Clean and preprocess data using Python libraries such as Pandas and SQL Alchemy. Handle missing values, standardize formats, and remove duplicates.
- Store structured data in PostgreSQL 17.1, creating separate tables for passenger satisfaction, flight status, and other domains.
- Use Git for version control, pushing code and configuration files to GitHub for collaboration, backup, and change tracking.

6.2.2 Workflow Orchestration and Automation

Tools:

1. **Apache Airflow (Version 2.10.5)**
2. **Docker (Version 4.41)**

Process:

- Build ETL (Extract, Transform, Load) pipelines in Python and schedule them using Apache Airflow.
- Define Directed Acyclic Graphs (DAGs) in Airflow to automate extraction, transformation, and loading of data into PostgreSQL.
- Containerize Python, Airflow, and PostgreSQL environments using Docker. This ensures consistency and portability across development, testing, and production.

- Deploy containers with Docker Compose for seamless orchestration and scaling. This setup simplifies dependency management and eliminates conflicts.

6.2.3 Connecting PostgreSQL to Power BI Desktop

Tools:

1. **Power BI Desktop (latest, May 2025)**

Process:

- Open Power BI Desktop and click "Get Data" → "PostgreSQL database"[9].
- If prompted, install the Npgsql driver for PostgreSQL connectivity.
- Enter the PostgreSQL server address, port (default 5432), database name, username, and password.
- For remote connections, edit the pg_hba.conf and postgresql.conf files in PostgreSQL to allow external access, then restart the database service.
- In the Navigator window, select the required tables and click "Load" to import data into Power BI.
- Use Power Query to further clean, filter, and transform data as needed for analysis. You can change data types, remove columns, and group or summarize data directly in Power BI.

6.2.4 Data Transformation with Python in Power BI

Tools:

1. **Python (Version 3.9)**
2. **Power BI Desktop**

Process:

- In Power BI Desktop, go to “Get Data” → “Other” → “Python script” to run Python scripts for advanced data transformation.
- Paste your Python script into the script field and click “OK.” Power BI will execute the script and present the results in the Navigator window.
- Select the resulting dataframe and load it into Power BI for visualization.
- This allows for advanced data manipulation and integration of Python analytics directly within Power BI.

6.2.5 Dashboard Design and Publishing

Tools:

1. **Power BI Desktop**
2. **Power BI Service (optional, for sharing)**

Process:

- Design interactive dashboards in Power BI Desktop, visualizing operational metrics, passenger satisfaction, and performance trends.
- Use slicers, filters, and drill-down features for detailed and dynamic analysis.
- Create calculated columns and measures using DAX for advanced insights.
- Save and publish dashboards to Power BI Service for secure web access and sharing with stakeholders.
- Set up scheduled data refreshes to keep dashboards updated with the latest data from PostgreSQL, using the Power BI Gateway if needed for on-premises data sources.

6.2.6 Monitoring, Collaboration, and Continuous Improvement

Tools:

1. **Docker**
2. **Git/GitHub**
3. **Power BI Service**

Process:

- Monitor ETL pipelines and container health using Airflow and Docker dashboards.
- Track user feedback and dashboard usage in Power BI Service to identify areas for improvement.
- Update code and configurations in GitHub, redeploy containers with Docker, and refresh Power BI datasets as needed for continuous enhancement.
- Document all processes and maintain clear version control for smooth collaboration and troubleshooting.

Best Practices:

- **Security:** Use strong authentication for PostgreSQL and GitHub. Restrict database access by IP and use SSL for data transfers.
- **Documentation:** Maintain clear documentation in GitHub for code, ETL processes, and dashboard usage.
- **Backup:** Regularly back up PostgreSQL databases and versioned code in GitHub.

CHAPTER 7

SYSTEM IMPLEMENTATION

7.1 SYSTEM MODULES

SI.NO	MODULES
1	**Customer Types & Travel Class**
2	**Travels by Age Groups**
3	**Our Services Rating**
4	**Travels by Gender and Type of Travel**
5	**Neutral or Dissatisfied vs. Satisfied Customers**
6	**General Insights and Recommendations**

7.1.1 Customer Types & Travel Class

This module categorizes passengers based on their customer type (such as business, leisure, or frequent flyer) and travel class (economy, premium economy, business, or first class). Data is extracted from booking records and enriched using Python scripts to segment passengers. PostgreSQL stores these segments in normalized tables, supporting efficient querying and aggregation. Power BI Desktop visualizes the distribution of customer types across travel classes using clustered bar charts and heatmaps. This module enables analysis of class-wise occupancy, revenue contribution, and customer behavior patterns. Airlines can identify high-value segments, monitor upgrade trends, and tailor loyalty programs. Insights from this module inform targeted service enhancements and personalized marketing strategies, helping maximize customer lifetime value and optimize seat allocation. Integration with the ETL pipeline ensures real-time updates, while role-based dashboard access allows airline managers to drill down into specific customer cohorts for deeper analysis.

7.1.2 Travels by Age Groups

This module segments passengers into predefined age brackets (e.g., <18, 18–30, 31–45, 46–60, >60) using demographic data collected during booking or check-in. Python scripts preprocess and validate age data, flagging anomalies or missing values. Data is stored in PostgreSQL with indexed columns for efficient age-based queries. Power BI Desktop visualizes age group distributions using pie charts, histograms, and stacked columns, enabling trend analysis over time and across routes. This module supports cohort analysis, allowing airlines to tailor services and marketing campaigns to specific age groups-such as special amenities for seniors or entertainment for younger travelers. Airlines can also track shifts in age demographics, assess the impact of promotions, and forecast future demand by age segment. Integration with other modules enables cross-analysis, such as correlating age with satisfaction or travel class preferences, providing a comprehensive view of passenger demographics.

7.1.3 Our Services Rating

This module aggregates and 21nalyses customer feedback on core service dimensions-check-in, boarding, in-flight experience, baggage handling, and customer support. Feedback data is collected via post-travel surveys and real-time mobile app inputs, then ingested into PostgreSQL through automated ETL pipelines. Python scripts calculate average ratings, standard deviations, and Net Promoter Scores (NPS) for each service category. Power BI Desktop visualizes these metrics using gauge charts, trend lines, and heatmaps, allowing stakeholders to monitor service quality in real time. Drill-down features enable root-cause analysis of low ratings, while time-series analysis tracks improvements after operational changes. This module helps airlines identify strengths, prioritize service enhancements, and benchmark against industry standards. Automated alerts can flag service areas falling below threshold ratings, prompting immediate corrective actions and supporting a culture of continuous improvement.

7.1.4 Travels by Gender and Type of Travel

This module classifies passengers by gender (male, female, non-binary, undisclosed) and type of travel (business, leisure, group, solo). Data is extracted from booking profiles and travel itineraries, validated for accuracy using Python scripts, and stored in PostgreSQL with categorical encoding for efficient querying. Power BI Desktop displays gender and travel type distributions using donut charts, stacked bars, and cross-tab visuals. Airlines can analyze travel patterns, identify gender-specific preferences, and assess the impact of targeted campaigns. This module supports diversity and inclusion initiatives by highlighting demographic trends and service utilization across gender groups. Integration with satisfaction and service rating modules enables deeper insights, such as gender-based satisfaction scores or travel type correlations with ancillary service uptake. Airlines can use these insights to personalize offerings, improve inclusivity, and optimize resource allocation for different traveler profiles.

7.1.5 Neutral or Dissatisfied vs. Satisfied Customers

This module segments customer feedback into three categories: satisfied, neutral, and dissatisfied, based on survey scores or sentiment analysis of open-text comments. Python scripts classify responses using predefined thresholds or NLP techniques, and results are stored in PostgreSQL for historical tracking. Power BI Desktop visualizes satisfaction segmentation using diverging bar charts and sentiment trend lines, providing a clear overview of customer sentiment across time periods, routes, or service touchpoints. Airlines can identify root causes of dissatisfaction, track the effectiveness of service recovery actions, and monitor the impact of operational changes on customer sentiment. This module supports closed-loop feedback processes, enabling targeted follow-ups with dissatisfied customers and proactive engagement with neutral respondents. Insights drive continuous service improvement, helping airlines reduce churn, increase loyalty, and enhance overall customer experience.

7.1.6 General Insights and Recommendations

This module synthesizes findings from all other modules, generating actionable insights and strategic recommendations for airline management. Python scripts aggregate key performance indicators (KPIs), uncover hidden patterns, and highlight correlations between demographic segments, service ratings, and satisfaction levels. PostgreSQL supports multi-dimensional queries, enabling comprehensive cross-module analysis. Power BI Desktop presents executive summaries, trend dashboards, and predictive forecasts using advanced visuals and AI-powered analytics. Recommendations may include targeted service enhancements, new product offerings, process optimizations, or marketing strategies based on data-driven evidence. This module also supports scenario analysis, allowing management to evaluate the potential impact of proposed changes. The integration of automated reporting ensures that insights are delivered to stakeholders in a timely manner, supporting agile decision-making and fostering a culture of continuous improvement within the aviation organization.

CHAPTER 8

SOFTWARE DESIGN

8.1 ARCHITECTURE DIAGRAM

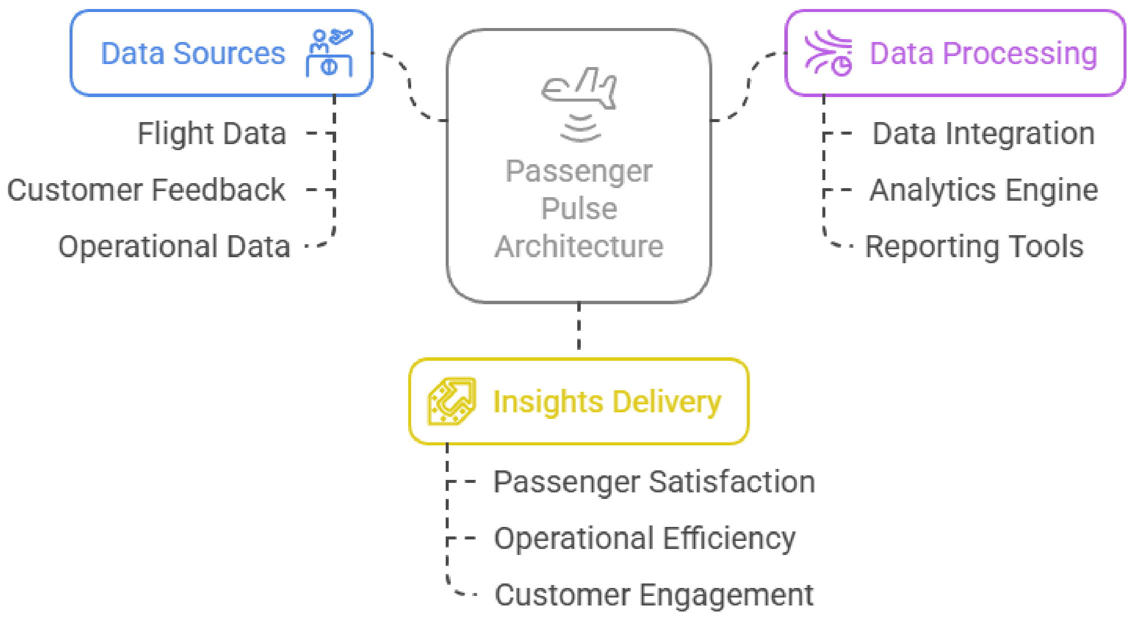


FIGURE: 8.1 Architecture Diagram

8.2 USE CASE DIAGRAM

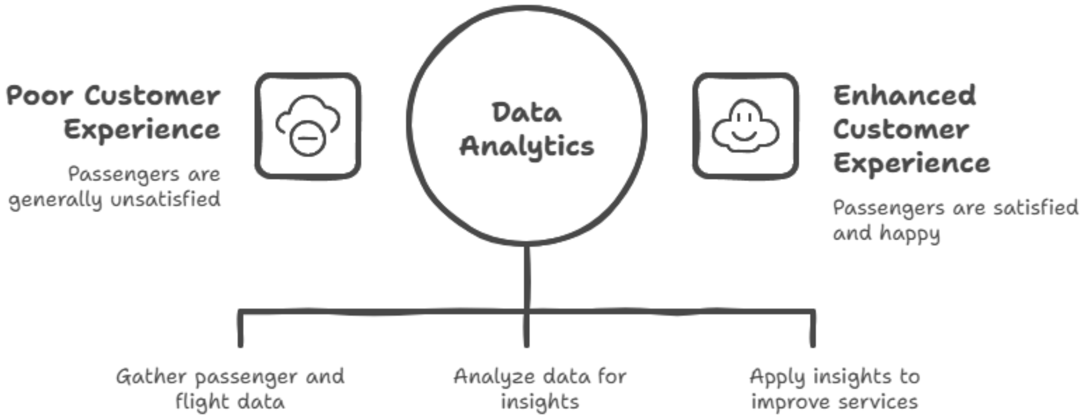


FIGURE: 8.2 Use Case Diagram

8.3 PROCESS - FLOW DIAGRAM

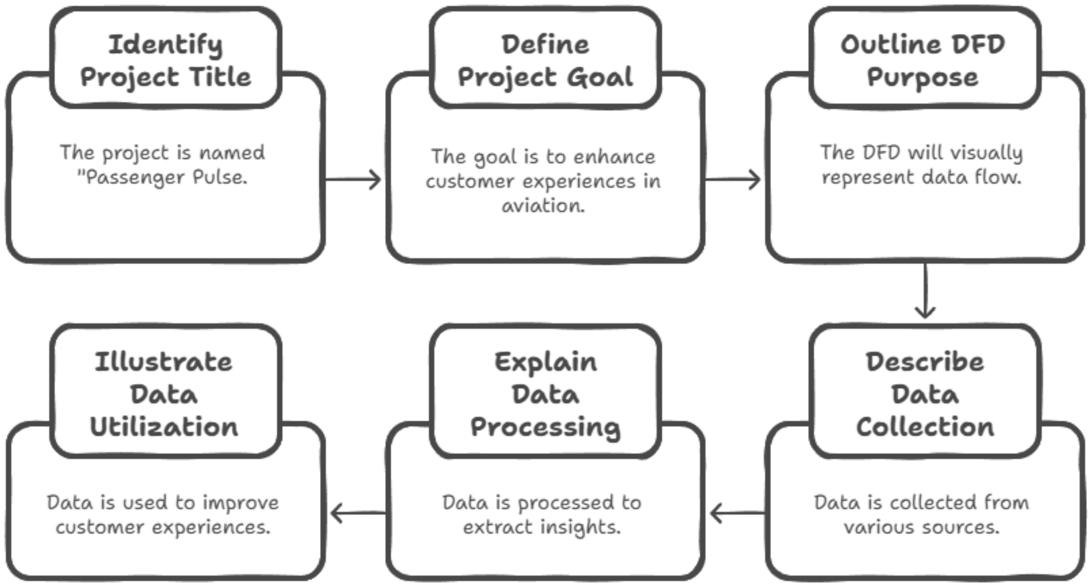


FIGURE: 8.3 Process – Flow Diagram

8.4 DATA FLOW DIAGRAM

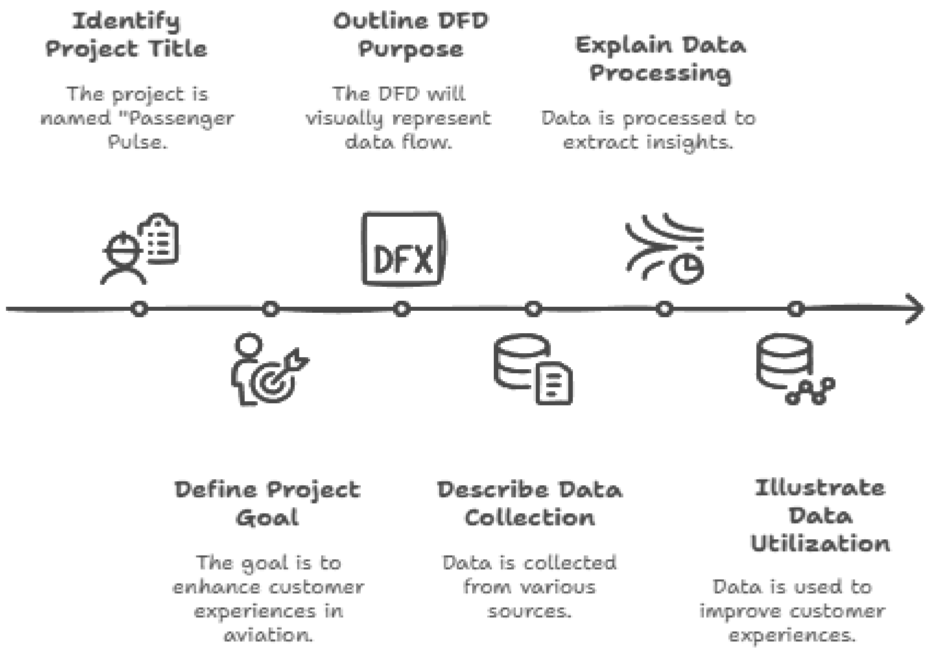


FIGURE: 8.4 Data Flow Diagram

8.5 CLASS DIAGRAM

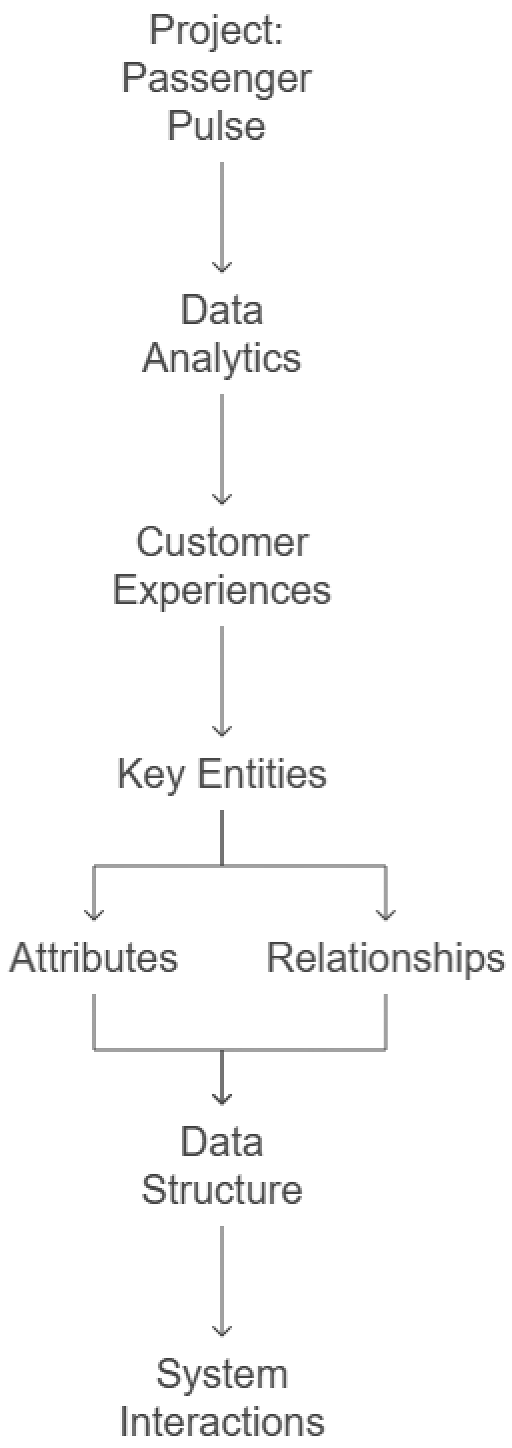


FIGURE: 8.5 Class Diagram

CHAPTER 9

CODING

Docker-compose.yaml

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.
#

# Basic Airflow cluster configuration for CeleryExecutor with Redis and PostgreSQL.
#
# WARNING: This configuration is for local development. Do not use it in a production
deployment.
#
# This configuration supports basic configuration using environment variables or an .env file
# The following variables are supported:
#
# AIRFLOW_IMAGE_NAME           - Docker image name used to run Airflow.
#                                Default: apache/airflow:2.10.5
# AIRFLOW_UID                  - User ID in Airflow containers
#                                Default: 50000
# AIRFLOW_PROJ_DIR             - Base path to which all the files will be volumed.
#                                Default: .
# Those configurations are useful mostly in case of standalone testing/running Airflow in
test/try-out mode
#
# _AIRFLOW_WWW_USER_USERNAME   - Username for the administrator account (if
requested).
#   Default: airflow
```

```
# _AIRFLOW_WWW_USER_PASSWORD   - Password for the administrator account (if
requested).
#                          Default: airflow
# _PIP_ADDITIONAL_REQUIREMENTS - Additional PIP requirements to add when
starting all containers.
# Use this option ONLY for quick checks. Installing requirements at container
# startup is done EVERY TIME the service is started.
# A better way is to build a custom image or extend the official image
# as described in https://airflow.apache.org/docs/docker-stack/build.html.
# Default: ''
#
# Feel free to modify this file to suit your needs.
---
x-airflow-common:
  &airflow-common
  # In order to add custom dependencies or upgrade provider packages you can use your
extended image.
  # Comment the image line, place your Dockerfile in the directory where you placed the
docker-compose.yaml
  # and uncomment the "build" line below, Then run `docker-compose build` to build the
images.
  image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.10.5}
  # build: .
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: CeleryExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN:
postgresql+psycopg2://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__RESULT_BACKEND:
db+postgresql://airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
    AIRFLOW__API__AUTH_BACKENDS:
'airflow.api.auth.backend.basic_auth,airflow.api.auth.backend.session'
    # yamllint disable rule:line-length
    # Use simple http server on scheduler for health checks
    # See https://airflow.apache.org/docs/apache-airflow/stable/administration-and-
deployment/logging-monitoring/check-health.html#scheduler-health-check-server
    # yamllint enable rule:line-length
    AIRFLOW__SCHEDULER__ENABLE_HEALTH_CHECK: 'true'
    # WARNING: Use _PIP_ADDITIONAL_REQUIREMENTS option ONLY for a quick
```

```
checks
    # for other purpose (development, test and especially production usage) build/extend
Airflow image.
    _PIP_ADDITIONAL_REQUIREMENTS: ${_PIP_ADDITIONAL_REQUIREMENTS:-}
    # The following line can be used to set a custom config file, stored in the local config folder
    # If you want to use it, outcomment it and replace airflow.cfg with the name of your config
file
    # AIRFLOW_CONFIG: '/opt/airflow/config/airflow.cfg'
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/transform:/opt/airflow/transform
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/config:/opt/airflow/config
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
    - ${AIRFLOW_PROJ_DIR:-.}/data:/opt/airflow/data
    - "K:/Data_analytics_project/Cyber1/Dataset/V1:/opt/airflow/data"
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    redis:
      condition: service_healthy
    postgres:
      condition: service_healthy

services:
  postgres:
    image: postgres:13
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "airflow"]
      interval: 10s
      retries: 5
      start_period: 5s
    restart: always

  pgadmin:
```

```yaml
  image: dpage/pgadmin4
  restart: always
  environment:
    PGADMIN_DEFAULT_EMAIL: admin@admin.com
    PGADMIN_DEFAULT_PASSWORD: admin
  ports:
    - "5050:80"
  depends_on:
    - postgres

redis:
  # Redis is limited to 7.2-bookworm due to licencing change
  # https://redis.io/blog/redis-adopts-dual-source-available-licensing/
  image: redis:7.2-bookworm
  expose:
    - 6379
  healthcheck:
    test: ["CMD", "redis-cli", "ping"]
    interval: 10s
    timeout: 30s
    retries: 50
    start_period: 30s
  restart: always

airflow-webserver:
  <<: *airflow-common
  command: webserver
  ports:
    - "8080:8080"
  healthcheck:
    test: ["CMD", "curl", "--fail", "http://localhost:8080/health"]
    interval: 30s
    timeout: 10s
    retries: 5
    start_period: 30s
  restart: always
  depends_on:
    <<: *airflow-common-depends-on
    airflow-init:
      condition: service_completed_successfully

airflow-scheduler:
  <<: *airflow-common
```

```
    command: scheduler
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:8974/health"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-worker:
    <<: *airflow-common
    command: celery worker
    healthcheck:
      # yamllint disable rule:line-length
      test:
        - "CMD-SHELL"
        - 'celery --app airflow.providers.celery.executors.celery_executor.app inspect ping -d
"celery@$${HOSTNAME}" || celery --app airflow.executors.celery_executor.app inspect
ping -d "celery@$${HOSTNAME}"'
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    environment:
      <<: *airflow-common-env
      # Required to handle warm shutdown of the celery workers properly
      # See https://airflow.apache.org/docs/docker-stack/entrypoint.html#signal-propagation
      DUMB_INIT_SETSID: "0"
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-triggerer:
    <<: *airflow-common
    command: triggerer
    healthcheck:
      test: ["CMD-SHELL", 'airflow jobs check --job-type TriggererJob --hostname
```

```
"$${HOSTNAME}"']
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-init:
    <<: *airflow-common
    entrypoint: /bin/bash
    # yamllint disable rule:line-length
    command:
      - -c
      - |
        if [[ -z "${AIRFLOW_UID}" ]]; then
          echo
          echo -e "\033[1;33mWARNING!!!: AIRFLOW_UID not set!\e[0m"
          echo "If you are on Linux, you SHOULD follow the instructions below to set "
          echo "AIRFLOW_UID environment variable, otherwise files will be owned by root."
          echo "For other operating systems you can get rid of the warning with manually
created .env file:"
          echo "    See: https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-
compose/index.html#setting-the-right-airflow-user"
          echo
        fi
        one_meg=1048576
        mem_available=$$(($$(getconf _PHYS_PAGES) * $$(getconf PAGE_SIZE) /
one_meg))
        cpus_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)
        disk_available=$$(df / | tail -1 | awk '{print $$4}')
        warning_resources="false"
        if (( mem_available < 4000 )) ; then
          echo
          echo -e "\033[1;33mWARNING!!!: Not enough memory available for Docker.\e[0m"
          echo "At least 4GB of memory required. You have $$(numfmt --to iec
$$((mem_available * one_meg)))"
          echo
          warning_resources="true"
        fi
```

```
        if (( cpus_available < 2 )); then
          echo
          echo -e "\033[1;33mWARNING!!!: Not enough CPUS available for Docker.\e[0m"
          echo "At least 2 CPUs recommended. You have $${cpus_available}"
          echo
          warning_resources="true"
        fi
        if (( disk_available < one_meg * 10 )); then
          echo
          echo -e "\033[1;33mWARNING!!!: Not enough Disk space available for
Docker.\e[0m"
          echo "At least 10 GBs recommended. You have $$(numfmt --to iec $$((disk_available
* 1024 )))"
          echo
          warning_resources="true"
        fi
        if [[ $${warning_resources} == "true" ]]; then
          echo
          echo -e "\033[1;33mWARNING!!!: You have not enough resources to run Airflow (see
above)!\e[0m"
          echo "Please follow the instructions to increase amount of resources available:"
          echo "   https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-
compose/index.html#before-you-begin"
          echo
        fi
        mkdir -p /sources/logs /sources/dags /sources/plugins
        chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,plugins}
        exec /entrypoint airflow version
    # yamllint enable rule:line-length
    environment:
      <<: *airflow-common-env
      _AIRFLOW_DB_MIGRATE: 'true'
      _AIRFLOW_WWW_USER_CREATE: 'true'
      _AIRFLOW_WWW_USER_USERNAME:
${_AIRFLOW_WWW_USER_USERNAME:-airflow}
      _AIRFLOW_WWW_USER_PASSWORD:
${_AIRFLOW_WWW_USER_PASSWORD:-airflow}
      _PIP_ADDITIONAL_REQUIREMENTS: "
    user: "0:0"
    volumes:
      - ${AIRFLOW_PROJ_DIR:-.}:/sources

  airflow-cli:
```

```
    <<: *airflow-common
    profiles:
      - debug
    environment:
      <<: *airflow-common-env
      CONNECTION_CHECK_MAX_COUNT: "0"
    # Workaround for entrypoint issue. See: https://github.com/apache/airflow/issues/16252
    command:
      - bash
      - -c
      - airflow

  # You can enable flower by adding "--profile flower" option e.g. docker-compose --profile flower up
  # or by explicitly targeted on the command line e.g. docker-compose up flower.
  # See: https://docs.docker.com/compose/profiles/
  flower:
    <<: *airflow-common
    command: celery flower
    profiles:
      - flower
    ports:
      - "5555:5555"
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:5555/"]
      interval: 30s
      timeout: 10s
      retries: 5
      start_period: 30s
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

volumes:
  postgres-db-volume:
```

SQL for creating flight_ext:

```
CREATE TABLE flight_ext (

  flight_id VARCHAR,

  airline VARCHAR,
```

```
  source_airport_code VARCHAR,
  destination_airport_code VARCHAR,
  status VARCHAR,
  delay_code VARCHAR,
  scheduled_departure VARCHAR,
  scheduled_arrival VARCHAR,
  travel_duration_hours VARCHAR
);
```

Flights_ext DAG:

```
import csv
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from datetime import datetime

def insert_data_from_csv_into_postgres(**kwargs):

    pg_hook = PostgresHook(postgres_conn_id='postgres_default')
    csv_file_path = '/opt/airflow/data/flights_data.csv'

    insert_query = """
    INSERT INTO flight_ext (flight_id, airline, source_airport_code,
destination_airport_code,
    status, delay_code, scheduled_departure, scheduled_arrival, travel_duration_hours)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
    """

    with open(csv_file_path, mode='r') as file:
        reader = csv.reader(file)
        next(reader)
        for row in reader:
            pg_hook.run(insert_query, parameters=(row[0], row[1], row[2], row[3], row[4],
row[5], row[6], row[7], row[8]))

default_args = {
    'owner': 'airflow',
    'retries': 1,
    'retry_delay': 300,
```

```
}

with DAG(
    'flights_ext_dag',
    default_args=default_args,
    description='A simple DAG to insert data from a CSV file into PostgreSQL',
    schedule_interval=None,
    start_date=datetime(2023, 1, 1),
    catchup=False,
) as dag:

    insert_data_from_csv_task = PythonOperator(
        task_id='insert_data_from_csv_into_postgres_task',
        python_callable=insert_data_from_csv_into_postgres,
        provide_context=True
    )

start_task = EmptyOperator(
    task_id='start_task'
)

end_task = EmptyOperator(
     task_id='end_task'
)

start_task>>insert_data_from_csv_task>>end_task
```

SQL for creating passenger_ext:

```
CREATE TABLE PASSENGERS_EXT (

    First_Name VARCHAR(100),

    Last_Name VARCHAR(100),

    Gender VARCHAR(100),

    DOB VARCHAR(100),

    Age_Group VARCHAR(100),

    Passenger_Type VARCHAR(100),

    Flight_ID VARCHAR(100),

    Payment_Method VARCHAR(100),

    Travel_Class VARCHAR(100),
```

```
    Meal_Preference VARCHAR(100),
    Total_Fare_Amount VARCHAR(100),
    Reward_Points VARCHAR(100),
    Service_Quality_Feedback VARCHAR(100),
    Cleanliness_Feedback VARCHAR(100),
    Timeliness_Feedback VARCHAR(100),
    Overall_Experience_Feedback VARCHAR(100),
    Luggage_Status VARCHAR(100),
    Luggage_Weight VARCHAR(100),
    Meal_Feedback VARCHAR(100),
    Gate_Location_Feedback VARCHAR(100),
    Other_Services_Feedback VARCHAR(100),
    Before_Boarding_Services_Feedback VARCHAR(100)
);
```

Passenger_ext.DAG:

```
import csv
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from datetime import datetime

def insert_data_from_csv_into_postgres(**kwargs):
    pg_hook = PostgresHook(postgres_conn_id='postgres_default')
    csv_file_path = '/opt/airflow/data/passengers_data.csv'

    insert_query = """
   INSERT INTO PASSENGERS_EXT (First_Name, Last_Name, Gender, DOB,
Age_Group,
                        Passenger_Type, Flight_ID, Payment_Method, Travel_Class,
                        Meal_Preference, Total_Fare_Amount, Reward_Points,
                        Service_Quality_Feedback, Cleanliness_Feedback,
                        Timeliness_Feedback, Overall_Experience_Feedback,
                        Luggage_Status, Luggage_Weight, Meal_Feedback,
                        Gate_Location_Feedback, Other_Services_Feedback,
                        Before_Boarding_Services_Feedback)
```

```
   VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s);
   """

    check_query = """
   SELECT 1 FROM PASSENGERS_EXT
   WHERE First_Name = %s AND Last_Name = %s AND DOB = %s AND Flight_ID = %s;
   """

    with open(csv_file_path, mode='r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip header
        for row in reader:
            if len(row) < 22:
                continue

            first_name = row[0]
            last_name = row[1]
            dob = row[3]
            flight_id = row[6]

            result = pg_hook.get_records(check_query, parameters=(first_name, last_name,
dob, flight_id))
            if not result:
                pg_hook.run(insert_query, parameters=row)


default_args = {
    'owner': 'airflow',
    'retries': 1,
    'retry_delay': 300,
}

with DAG(
        'passengers_ext_dag',
        default_args=default_args,
        description='A DAG to insert new data from CSV into PostgreSQL, skipping existing
records',
        schedule_interval=None,
        start_date=datetime(2023, 1, 1),
        catchup=False,
) as dag:
    insert_data_from_csv_task = PythonOperator(
```

```
        task_id='insert_data_from_csv_into_postgres_task',
        python_callable=insert_data_from_csv_into_postgres,
        provide_context=True
    )


start_task = EmptyOperator(
    task_id='start_task'
)

end_task = EmptyOperator(
     task_id='end_task'
)

start_task>>insert_data_from_csv_task>>end_task
```

SQL for operations_ext:

```
CREATE TABLE OPERATIONS_EXT (

    Operation_Id VARCHAR(250),

    Airport_Code VARCHAR(100),

    Operation_Type VARCHAR(100),

    Flight_ID VARCHAR(100),

    Operation_Time VARCHAR(100),

    Operator_Name VARCHAR(100),

    Operator_Contact VARCHAR(100)

);
```

Operations_ext.DAG:

```
import csv
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from datetime import datetime

def insert_data_from_csv_into_postgres(**kwargs):
    pg_hook = PostgresHook(postgres_conn_id='postgres_default')
    csv_file_path = '/opt/airflow/data/Airport_Operations_Data.csv'
```

```
    insert_query = """
    INSERT INTO operations_ext (operation_id, airport_code, operation_type, flight_id,
    operation_time, operator_name, operator_contact)
    VALUES ( %s, %s, %s, %s, %s, %s, %s)
    """

    with open(csv_file_path, mode='r') as file:
        reader = csv.reader(file)
        next(reader)
        for row in reader:
            pg_hook.run(insert_query, parameters=(row[0], row[1], row[2], row[3], row[4],
row[5], row[6]))

default_args = {
    'owner': 'airflow',
    'retries': 1,
    'retry_delay': 300,
}

with DAG(
    'operation_ext_dag',
    default_args=default_args,
    description='A simple DAG to insert data from a CSV file into PostgreSQL',
    schedule_interval=None,
    start_date=datetime(2023, 1, 1),
    catchup=False,
) as dag:

    insert_data_from_csv_task = PythonOperator(
        task_id='insert_data_from_csv_into_postgres_task',
        python_callable=insert_data_from_csv_into_postgres,
        provide_context=True
    )

start_task = EmptyOperator(
    task_id='start_task'
)

end_task = EmptyOperator(
    task_id='end_task'
)

start_task >> insert_data_from_csv_task >> end_task
```

SQL for delay_ext:

```
CREATE TABLE DELAY_EXT (
   Delay_Code VARCHAR(100),
   Description VARCHAR(250)
);
```

Delay_ext.DAG:

```
import csv
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from datetime import datetime

def insert_data_from_csv_into_postgres(**kwargs):

   pg_hook = PostgresHook(postgres_conn_id='postgres_default')
   csv_file_path = '/opt/airflow/data/delay_reasons_data.csv'

   insert_query = """
   INSERT INTO delay_ext (delay_code, description)
   VALUES ( %s, %s)
   """

   with open(csv_file_path, mode='r') as file:
      reader = csv.reader(file)
      next(reader)
      for row in reader:
         pg_hook.run(insert_query, parameters=(row[0], row[1]))

default_args = {
   'owner': 'airflow',
   'retries': 1,
   'retry_delay': 300,
}

with DAG(
   'delay_ext_dag',
   default_args=default_args,
   description='A simple DAG to insert data from a CSV file into PostgreSQL',
   schedule_interval=None,
```

```
    start_date=datetime(2023, 1, 1),
    catchup=False,
) as dag:

    insert_data_from_csv_task = PythonOperator(
        task_id='insert_data_from_csv_into_postgres_task',
        python_callable=insert_data_from_csv_into_postgres,
        provide_context=True
    )

start_task = EmptyOperator(
    task_id='start_task'
)

end_task = EmptyOperator(
     task_id='end_task'
)

start_task>>insert_data_from_csv_task>>end_task
```

SQL for creating flight_sfm:

```
CREATE TABLE flight_sfm (
  flight_id VARCHAR,
  airline VARCHAR,
  source_airport_code VARCHAR,
  destination_airport_code VARCHAR,
  status VARCHAR,
  delay_code VARCHAR,
  scheduled_departure TIMESTAMP,
  scheduled_arrival TIMESTAMP,
  travel_duration_hours DECIMAL
);
```

Flight_sfm.DAG:

```
from airflow import DAG
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
```

```
from datetime import datetime
import pandas as pd

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2023, 1, 1),
    'retries': 1,
}

start_task = EmptyOperator(
    task_id='start_task'
)

def extract_flight_data():
    hook = PostgresHook(postgres_conn_id='postgres_default')
    conn = hook.get_conn()
    df = hook.get_pandas_df("SELECT * FROM extract.flight_ext")
    conn.close()
    return df


def transform_flight_data(df):
    date_format = '%d-%m-%Y %H:%M'

    for col in ['scheduled_departure', 'scheduled_arrival']:
        df[col] = pd.to_datetime(
            df[col],
            format=date_format,
            errors='coerce'
        )
        df[col] = df[col].apply(lambda x: x if pd.notnull(x) else None)
        df[col] = df[col].astype(object)

    df['travel_duration_hours'] = pd.to_numeric(
        df['travel_duration_hours'].replace('', pd.NA),
        errors='coerce'
    )
    df['travel_duration_hours'] = df['travel_duration_hours'].apply(
        lambda x: x if pd.notnull(x) else None
    )

    return df
```

```
def load_flight_data(df):
    hook = PostgresHook(postgres_conn_id='postgres_default')
    conn = hook.get_conn()
    cursor = conn.cursor()

    columns = df.columns.tolist()
    values_placeholder = ','.join(['%s'] * len(columns))
    insert_sql = f"""
      INSERT INTO transform.flight_sfm
      ({','.join(columns)})
      VALUES ({values_placeholder})
   """

    data = [tuple(None if pd.isna(x) else x for x in record)
           for record in df.to_records(index=False)]

    cursor.executemany(insert_sql, data)
    conn.commit()

    cursor.close()
    conn.close()

def etl_process():
    raw_df = extract_flight_data()

    transformed_df = transform_flight_data(raw_df)

    load_flight_data(transformed_df)

end_task = EmptyOperator(
    task_id='end_task'
)

with DAG(
        'flight_sfm_dag',
        default_args=default_args,
        description='Transform flight data using Python',
        schedule_interval=None,
        catchup=False,
) as dag:
    transform_task = PythonOperator(
```

```
        task_id='transforming_and_loading_data',
        python_callable=etl_process
    )

start_task>>transform_task>>end_task
```

SQL for creating passenger_sfm:

```
CREATE TABLE PASSENGERS_SFM (
   First_Name VARCHAR(100),
   Last_Name VARCHAR(100),
   Gender VARCHAR(100),
   DOB VARCHAR(100),
   Age_Group VARCHAR(100),
   Passenger_Type VARCHAR(100),
   Flight_ID VARCHAR(100),
   Payment_Method VARCHAR(100),
   Travel_Class VARCHAR(100),
   Meal_Preference VARCHAR(100),
   Total_Fare_Amount DECIMAL,
   Reward_Points INT,
   Service_Quality_Feedback INT,
   Cleanliness_Feedback INT,
   Timeliness_Feedback INT,
   Overall_Experience_Feedback INT,
   Luggage_Status VARCHAR(100),
   Luggage_Weight DECIMAL,
   Meal_Feedback INT,
   Gate_Location_Feedback INT,
   Other_Services_Feedback INT,
   Before_Boarding_Services_Feedback INT
);
```

Passenger_sfm.DAG:

```
from airflow import DAG
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime
import pandas as pd

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2023, 1, 1),
    'retries': 1,
}

FEEDBACK_MAPPING = {
    'Poor': 1,
    'Average': 2,
    'Good': 3,
    'Excellent': 4
}

start_task = EmptyOperator(
    task_id='start_task'
)

def extract_data():
    hook = PostgresHook(postgres_conn_id='postgres_default')
    conn = hook.get_conn()
    df = hook.get_pandas_df("SELECT * FROM extract.passengers_ext")
    conn.close()
    return df


def transform_data(df):
    df['dob'] = pd.to_datetime(df['dob']).dt.date

    # Convert numeric columns
    df['total_fare'] = df['total_fare'].astype(float)
    df['reward_points'] = df['reward_points'].astype(int)
    df['luggage_weight'] = df['luggage_weight'].astype(float)

    feedback_columns = [
```

```
        'service_quality_feedback',
        'cleanliness_feedback',
        'timeliness_feedback',
        'overall_experience_feedback',
        'meal_feedback',
        'gate_location_feedback',
        'other_services_feedback',
        'before_boarding_services_feedback'
    ]

    for col in feedback_columns:
        df[col] = df[col].map(FEEDBACK_MAPPING)

    return df


def load_data(df):
    hook = PostgresHook(postgres_conn_id='postgres_default')
    conn = hook.get_conn()

    hook.insert_rows(
        table='transform.passengers_sfm',
        rows=df.values.tolist(),
        target_fields=df.columns.tolist()
    )
    conn.close()


def etl_process():
    raw_df = extract_data()

    transformed_df = transform_data(raw_df)

    load_data(transformed_df)

end_task = EmptyOperator(
    task_id='end_task'
)

with DAG(
        'passengers_sfm_dag',
        default_args=default_args,
        description='Transform passenger data using pure Python',
```

```
        schedule_interval=None,
        catchup=False,
) as dag:
    transform_task = PythonOperator(
        task_id='transforming_and_loading_data',
        python_callable=etl_process
    )

start_task>>transform_task>>end_task
```

SQL for operations_sfm:

```
CREATE TABLE OPERATIONS_SFM (
   Operation_Id VARCHAR(250),
   Airport_Code VARCHAR(100),
   Operation_Type VARCHAR(100),
   Flight_ID VARCHAR(100),
   Operation_Time TIMESTAMP,
   Operator_Name VARCHAR(100),
   Operator_Contact VARCHAR(100)
);
```

Operations_sfm DAG:

```
from airflow import DAG
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime
import pandas as pd
import re

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2023, 1, 1),
    'retries': 1,
}


start_task = EmptyOperator(
```

```
    task_id='start_task'
)

def clean_operator_contact(contact):
    if pd.isna(contact) or contact.strip() == "":
        return None
    contact = contact.split('x')[0]
    contact = re.sub(r'\D', '', contact)
    return f"+{contact}" if contact else None


def extract_operation_data():
    hook = PostgresHook(postgres_conn_id='postgres_default')
    return hook.get_pandas_df('SELECT * FROM "extract"."operations_ext"')


def transform_operation_data(df):
    df['operator_contact'] = df['operator_contact'].apply(clean_operator_contact)
    return df.where(pd.notnull(df), None)


def load_operation_data(df):
    hook = PostgresHook(postgres_conn_id='postgres_default')
    hook.insert_rows(
        table='transform.operations_sfm',
        rows=df.values.tolist(),
        target_fields=df.columns.tolist()
    )


def etl_process():
    raw_df = extract_operation_data()

    transformed_df = transform_operation_data(raw_df)

    load_operation_data(transformed_df)


with DAG(
        'operation_sfm_dag',
        default_args=default_args,
        description='Transform operations data',
        schedule_interval=None,
```

```
        catchup=False,
) as dag:
    transform_task = PythonOperator(
        task_id='transforming_and_loading_data',
        python_callable=etl_process,
    )

end_task = EmptyOperator(
    task_id='end_task'
)

start_task>>transform_task>>end_task
```

SQL for delay_sfm:

```
CREATE TABLE DELAY_SFM (

   Delay_Code VARCHAR(100),

   Description VARCHAR(250)

);
```

Delay_sfm DAG:

```
from airflow import DAG
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2023, 1, 1),
    'retries': 1,
}

start_task = EmptyOperator(
    task_id='start_task'
)


# Single function to handle both extraction and loading
def etl_process():
    hook = PostgresHook(postgres_conn_id='postgres_default')
```

```
    conn = hook.get_conn()
    df = hook.get_pandas_df("SELECT * FROM extract.delay_ext")

    hook.insert_rows(
        table='transform.delay_sfm',
        rows=df.values.tolist(),
        target_fields=df.columns.tolist()
    )

    conn.close()


with DAG(
        'delay_sfm_dag',
        default_args=default_args,
        description='Transform passenger data using pure Python',
        schedule_interval=None,
        catchup=False,
) as dag:
    transform_task = PythonOperator(
        task_id='transforming_and_loading_data',
        python_callable=etl_process
    )

end_task = EmptyOperator(
    task_id='end_task'
)

start_task >> transform_task >> end_task
```

SQL for creating dim_passenger:

```
CREATE TABLE dimension.dim_passenger(

   passenger_sk serial,

   FIRST_NAME_TXT VARCHAR,

   LAST_NAME_TXT VARCHAR,

   GENDER_TXT VARCHAR,

   DOB DATE,

   REWARD_POINTS INTEGER,

   ETL_LOAD_NBR INTEGER,
```

```
    ETL_LOADED_DATE TIMESTAMP,
    FILENAME_TXT VARCHAR
);
```

SQL for creating dim_flight:

```
CREATE TABLE dimension.dim_flight
(
    flight_sk serial,
    flight_id_txt varchar,
    airline_txt varchar,
    status_txt varchar,
    delay_code_txt varchar,
    description_txt varchar,
    source_airport_code_txt varchar,
    destination_airport_code_txt varchar,
    date date,
    scheduled_departure timestamp without time zone,
    scheduled_arrival timestamp without time zone,
    travel_duration_hours_dc numeric,
    etl_load_nbr int,
    etl_loaded_date timestamp without time zone,
    filename_txt varchar
)
```

SQL for dim_operations:

```
CREATE TABLE IF NOT EXISTS dimension.dim_operations
(
    operation_sk integer NOT NULL DEFAULT
nextval('dimension.dim_operations_operation_sk_seq'::regclass),
    operation_id_txt character varying COLLATE pg_catalog."default",
    flight_id_txt character varying COLLATE pg_catalog."default",
```

```
    airport_code_txt character varying COLLATE pg_catalog."default",
    operation_type_txt character varying COLLATE pg_catalog."default",
    operation_time timestamp without time zone,
    operator_contact_txt character varying COLLATE pg_catalog."default",
    operator_name_txt character varying COLLATE pg_catalog."default",
    etl_load_nbr integer,
    etl_loaded_date timestamp without time zone,
    filename_txt character varying COLLATE pg_catalog."default",
    CONSTRAINT dim_operations_pkey PRIMARY KEY (operation_sk)
)
```

SQL for dim_date:

```
CREATE TABLE dimension.dim_date
(
    date_id integer NOT NULL,
    date date,
    year integer,
    month integer,
    day integer,
    day_of_the_week_txt character varying(10),
    week_nbr integer,
    etl_load_nbr character varying(255),
    etl_loaded_date timestamp
)
```

Dim_date DAG:

```
from datetime import datetime
from airflow import DAG
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago

default_args = {
```

```
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(1),
    'email_on_failure': False,
    'retries': 0
}

def validate_input(**context):
    input_year = context['dag_run'].conf.get('input_year')
    if not input_year:
        raise ValueError("input_year must be provided in configuration!")
    if not isinstance(input_year, int):
        raise TypeError("input_year must be an integer")
    return input_year

def execute_load(**context):
    hook = PostgresHook(postgres_conn_id='postgres_default')
    print(f"Connection Details: {hook.get_uri()}")
    conn = hook.get_conn()
    cursor = conn.cursor()

    input_year = context['dag_run'].conf['input_year']

    try:
        cursor.execute("SELECT load_date_range(%s);", (input_year,))
        conn.commit()
    except Exception as e:
        conn.rollback()
        raise e
    finally:
        cursor.close()
        conn.close()

with DAG(
    'date_dimension_loader',
    default_args=default_args,
    description='DAG to load dates using PostgresHook',
    schedule_interval=None,
    catchup=False,
    tags=['dimension', 'date'],
    params={
        "input_year": 0
    }
```

```
) as dag:

    validate_params = PythonOperator(
        task_id='validate_parameters',
        python_callable=validate_input,
        provide_context=True
    )

    execute_load_task = PythonOperator(
        task_id='execute_date_load',
        python_callable=execute_load,
        provide_context=True
    )

    validate_params >> execute_load_task
```

Dim_DAG.py:

```
from airflow import DAG
from airflow.decorators import task
from airflow.operators.empty import EmptyOperator
from datetime import datetime
from transform.dim_logic import *

default_args = {
    'owner': 'airflow',
    'start_date': datetime(2023, 1, 1),
    'retries': 1,
    'retry_delay': 300,
}

with DAG(
    'dim_loader',
    default_args=default_args,
    schedule_interval=None,
    catchup=False,
    tags=['dimension_table'],
) as dag:

    start = EmptyOperator(task_id='start')


    @task(task_id="load_data_operation")
```

```
    def load_data_operation():
        transform_and_load_operations()

    load_data_operation()


    @task(task_id="load_data_flight")
    def load_data_flight():
        transform_and_load_flight()

    load_data_flight()

    @task(task_id="load_data_passenger")
    def load_data_passenger():
        transform_and_load_passengers()

    load_data_passenger()

    end = EmptyOperator(task_id='end')
```

Dim_logic.py:

```
from airflow.providers.postgres.hooks.postgres import PostgresHook
from io import StringIO
import pandas as pd
from datetime import datetime
from contextlib import closing


def transform_and_load_operations():
    hook = PostgresHook(postgres_conn_id='postgres_default')
    conn = hook.get_conn()

    source_table = 'transform.operations_sfm'
    target_table = 'dimension.dim_operations'

    columns_mapping = {
        'source_columns': [
            'OPERATION_ID',
            'FLIGHT_ID',
            'AIRPORT_CODE',
            'OPERATION_TYPE',
            'OPERATION_TIME',
            'OPERATOR_CONTACT',
```

```
            'OPERATOR_NAME'
        ],
        'target_columns': [
            'OPERATION_ID_TXT',
            'FLIGHT_ID_TXT',
            'AIRPORT_CODE_TXT',
            'OPERATION_TYPE_TXT',
            'OPERATION_TIME',
            'OPERATOR_CONTACT_TXT',
            'OPERATOR_NAME_TXT',
            'ETL_LOAD_NBR',
            'ETL_LOADED_DATE',
            'FILENAME_TXT'
        ]
    }

    with closing(conn.cursor()) as cursor:
        # Build dynamic insert statement
        select_columns = ', '.join(columns_mapping['source_columns'])
        insert_columns = ', '.join(columns_mapping['target_columns'])
        static_values = " 1, CURRENT_TIMESTAMP, 'Operations_data'"

        cursor.execute(
            f"INSERT INTO {target_table} ({insert_columns}) "
            f"SELECT {select_columns}, {static_values} "
            f"FROM {source_table}"
        )
        conn.commit()


def transform_and_load_passengers():
    """Data transformation and loading logic"""
    hook = PostgresHook(postgres_conn_id='postgres_default')
    conn = hook.get_conn()

    source_table = 'transform.passengers_sfm'
    target_table = 'dimension.dim_passenger'

    columns_mapping = {
        'source_columns': [
            'first_name',
            'last_name',
            'gender',
```

```
            'dob',
            'reward_points'
        ],
        'target_columns': [
            'first_name_txt',
            'last_name_txt',
            'gender_txt',
            'dob',
            'reward_points',
            'etl_load_nbr',
            'etl_loaded_date',
            'filename_txt'
        ]
    }

    with closing(conn.cursor()) as cursor:
        # Build dynamic insert statement
        select_columns = ', '.join(columns_mapping['source_columns'])
        insert_columns = ', '.join(columns_mapping['target_columns'])
        static_values = " 1, CURRENT_TIMESTAMP, 'Passengers_data'"

        cursor.execute(
            f"INSERT INTO {target_table} ({insert_columns}) "
            f"SELECT {select_columns}, {static_values} "
            f"FROM {source_table}"
        )
        conn.commit()

def transform_and_load_flight():
    source_hook = PostgresHook(postgres_conn_id='postgres_default')
    dest_hook = PostgresHook(postgres_conn_id='postgres_default')

    # Extract from source
    flight_data = source_hook.get_pandas_df("SELECT * FROM transform.flight_sfm")
    delay_mapping = source_hook.get_pandas_df("SELECT * FROM transform.delay_sfm")

    # Merge + transform
    merged_data = pd.merge(flight_data, delay_mapping, on='delay_code', how='left')

    final_data = merged_data[[
        'flight_id', 'airline', 'source_airport_code', 'destination_airport_code',
        'status', 'delay_code', 'description', 'scheduled_departure', 'scheduled_arrival',
        'travel_duration_hours'
```

```
    ]].rename(columns={
        'flight_id': 'flight_id_txt',
        'airline': 'airline_txt',
        'source_airport_code': 'source_airport_code_txt',
        'destination_airport_code': 'destination_airport_code_txt',
        'status': 'status_txt',
        'delay_code': 'delay_code_txt',
        'description': 'description_txt',
        'travel_duration_hours': 'travel_duration_hours_dc'
    })

    final_data['date'] = pd.to_datetime(final_data['scheduled_departure']).dt.date

    # ETL metadata
    final_data['etl_load_nbr'] = 1
    final_data['etl_loaded_date'] = datetime.now()
    final_data['filename_txt'] = 'flight_data, delay_reasons_data'

    # Buffer to CSV
    buffer = StringIO()
    final_data.to_csv(buffer, index=False, header=False)
    buffer.seek(0)

    # Load into Postgres
    dest_conn = dest_hook.get_conn()
    cursor = dest_conn.cursor()
    cursor.copy_expert(
        sql="""
      COPY dimension.dim_flight (
          flight_id_txt, airline_txt, source_airport_code_txt, destination_airport_code_txt,
          status_txt, delay_code_txt, description_txt, scheduled_departure, scheduled_arrival,
          travel_duration_hours_dc, date, etl_load_nbr, etl_loaded_date, filename_txt
      )
      FROM STDIN WITH CSV
      """,
        file=buffer
    )
    dest_conn.commit()
```

SQL for fact:

```
WITH PassengerOrder AS (
  SELECT
    p.*,
```

```
    dp.passenger_sk,
    ROW_NUMBER() OVER (
      PARTITION BY p.flight_id, p.travel_class, p.dob
      ORDER BY p.first_name, p.last_name, p.dob  -- Replace with actual priority column if
available
    ) AS rn
  FROM transform.passengers_sfm p
  JOIN dimension.dim_passenger dp
    ON p.first_name = dp.first_name_txt
   AND p.last_name = dp.last_name_txt
   AND p.dob = dp.dob
   AND p.reward_points = dp.reward_points
),

FlightDates AS (
  SELECT
    flight_id,
    scheduled_departure,
    DATE(scheduled_departure) AS flight_date,
    ROW_NUMBER() OVER (
      PARTITION BY flight_id
      ORDER BY scheduled_departure
    ) AS date_order
  FROM transform.flight_sfm
),

PassengerBuckets AS (
  SELECT
    *,
    CASE
      WHEN travel_class = 'Economy' THEN ((rn - 1) / 200) + 1
      WHEN travel_class = 'First Class' THEN ((rn - 1) / 120) + 1
      WHEN travel_class = 'Business' THEN ((rn - 1) / 80) + 1
    END AS bucket
```

```
  FROM PassengerOrder
)

INSERT INTO dimension.fact (
  flight_id, passenger_sk, passenger_type_txt, luggage_status_txt,
  travel_class_txt, meal_preference_txt, service_quality_feedback_nbr,
  cleanliness_feedback_nbr, timeliness_feedback_nbr, overall_experience_feedback_nbr,
  meal_feedback_nbr, gate_location_feedback_nbr, other_service_feedback_nbr,
  before_boarding_services_feedback_nbr, payment_method_txt, total_fare_amount,
  luggage_weight, etl_load_nbr, etl_loaded_date
)
SELECT
  pb.flight_id,
  pb.passenger_sk,
  pb.passenger_type AS passenger_type_txt,
  pb.luggage_status AS luggage_status_txt,
  pb.travel_class AS travel_class_txt,
  pb.meal_preference AS meal_preference_txt,
  pb.service_quality_feedback AS service_quality_feedback_nbr,
  pb.cleanliness_feedback AS cleanliness_feedback_nbr,
  pb.timeliness_feedback AS timeliness_feedback_nbr,
  pb.overall_experience_feedback AS overall_experience_feedback_nbr,
  pb.meal_feedback AS meal_feedback_nbr,
  pb.gate_location_feedback AS gate_location_feedback_nbr,
  pb.other_services_feedback AS other_service_feedback_nbr,
  pb.before_boarding_services_feedback AS before_boarding_services_feedback_nbr,
  pb.payment_method AS payment_method_txt,
  pb.total_fare AS total_fare_amount,
  pb.luggage_weight,
  1 AS etl_load_nbr,
  CURRENT_TIMESTAMP AS etl_loaded_date
FROM PassengerBuckets pb
JOIN FlightDates fd
  ON pb.flight_id = fd.flight_id
```

```
  AND pb.bucket = fd.date_order
ORDER BY pb.passenger_sk ASC;
```

Fact DAG:

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.utils.dates import days_ago
from datetime import timedelta

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

def run_fact_insert():
    pg_hook = PostgresHook(postgres_conn_id='postgres_default')
    conn = pg_hook.get_conn()
    cursor = conn.cursor()

    sql = """
            WITH PassengerOrder AS (
              SELECT
                p.*,
                dp.passenger_sk,
                ROW_NUMBER() OVER (
                  PARTITION BY p.flight_id, p.travel_class, p.dob
                  ORDER BY p.first_name, p.last_name, p.dob  -- Replace with actual priority
column if available
                ) AS rn
              FROM transform.passengers_sfm p
              JOIN dimension.dim_passenger dp
```

```
    ON p.first_name = dp.first_name_txt
    AND p.last_name = dp.last_name_txt
    AND p.dob = dp.dob
    AND p.reward_points = dp.reward_points
),

FlightDates AS (
  SELECT
    flight_id,
    scheduled_departure,
    DATE(scheduled_departure) AS flight_date,
    ROW_NUMBER() OVER (
      PARTITION BY flight_id
      ORDER BY scheduled_departure
    ) AS date_order
  FROM transform.flight_sfm
),

PassengerBuckets AS (
  SELECT
    *,
    CASE
      WHEN travel_class = 'Economy' THEN ((rn - 1) / 200) + 1
      WHEN travel_class = 'First Class' THEN ((rn - 1) / 120) + 1
      WHEN travel_class = 'Business' THEN ((rn - 1) / 80) + 1
    END AS bucket
  FROM PassengerOrder
)

INSERT INTO dimension.fact (
  flight_id, passenger_sk, passenger_type_txt, luggage_status_txt,
  travel_class_txt, meal_preference_txt, service_quality_feedback_nbr,
  cleanliness_feedback_nbr, timeliness_feedback_nbr,
overall_experience_feedback_nbr,
```

```
        meal_feedback_nbr, gate_location_feedback_nbr, other_service_feedback_nbr,
        before_boarding_services_feedback_nbr, payment_method_txt, total_fare_amount,
        luggage_weight, etl_load_nbr, etl_loaded_date
      )
      SELECT
        pb.flight_id,
        pb.passenger_sk,
        pb.passenger_type AS passenger_type_txt,
        pb.luggage_status AS luggage_status_txt,
        pb.travel_class AS travel_class_txt,
        pb.meal_preference AS meal_preference_txt,
        pb.service_quality_feedback AS service_quality_feedback_nbr,
        pb.cleanliness_feedback AS cleanliness_feedback_nbr,
        pb.timeliness_feedback AS timeliness_feedback_nbr,
        pb.overall_experience_feedback AS overall_experience_feedback_nbr,
        pb.meal_feedback AS meal_feedback_nbr,
        pb.gate_location_feedback AS gate_location_feedback_nbr,
        pb.other_services_feedback AS other_service_feedback_nbr,
        pb.before_boarding_services_feedback AS before_boarding_services_feedback_nbr,
        pb.payment_method AS payment_method_txt,
        pb.total_fare AS total_fare_amount,
        pb.luggage_weight,
        1 AS etl_load_nbr,
        CURRENT_TIMESTAMP AS etl_loaded_date
      FROM PassengerBuckets pb
      JOIN FlightDates fd
        ON pb.flight_id = fd.flight_id
        AND pb.bucket = fd.date_order
      ORDER BY pb.passenger_sk ASC;
        """

cursor.execute(sql)
conn.commit()
cursor.close()
```

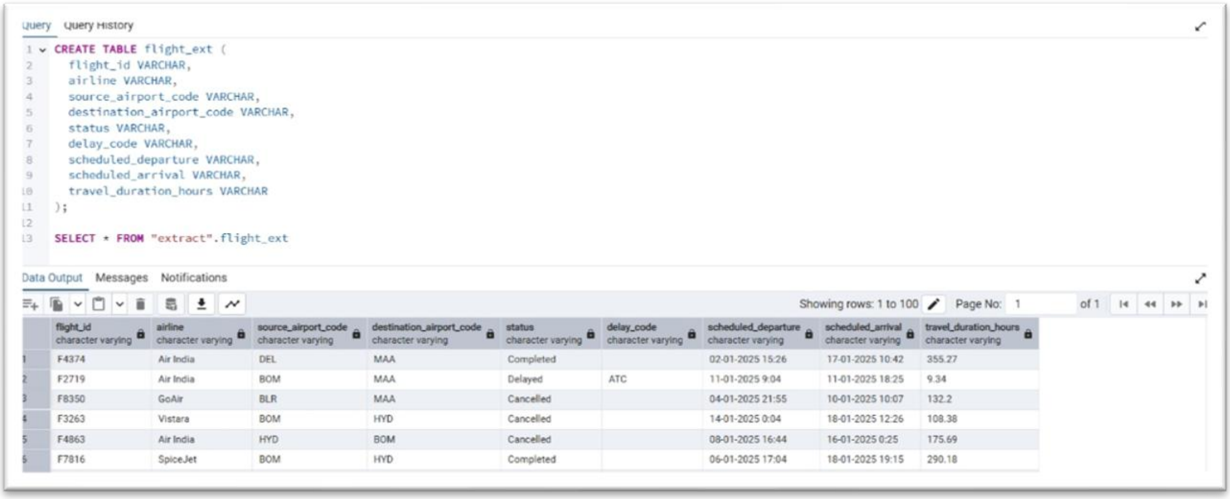
```
    conn.close()

with DAG(
    dag_id='fact_dag',
    default_args=default_args,
    description='ETL DAG to insert into fact table from CTE joins',
    schedule_interval=None,
    start_date=days_ago(1),
    catchup=False,
    tags=['ETL', 'fact_table', 'passenger_data'],
) as dag:


    insert_fact_data = PythonOperator(
        task_id='insert_fact_table_data',
        python_callable=run_fact_insert
    )

    insert_fact_data
```

CHAPTER 10

OUTPUT

EXTRACT CODE:

FLIGHT:



Query Query History

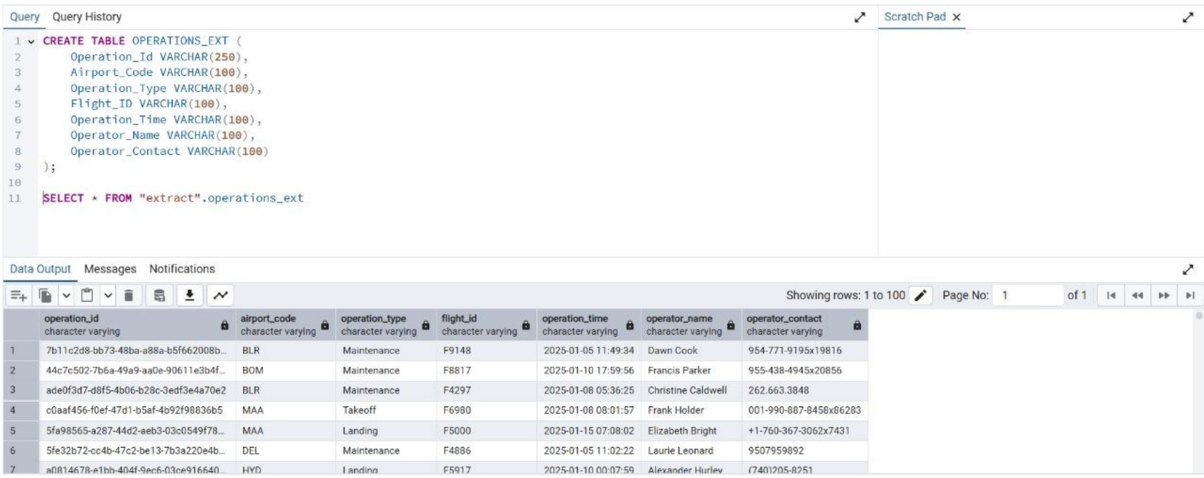
```sql
CREATE TABLE flight_ext (
  flight_id VARCHAR,
  airline VARCHAR,
  source_airport_code VARCHAR,
  destination_airport_code VARCHAR,
  status VARCHAR,
  delay_code VARCHAR,
  scheduled_departure VARCHAR,
  scheduled_arrival VARCHAR,
  travel_duration_hours VARCHAR
);

SELECT * FROM "extract".flight_ext
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	flight_id character varying	airline character varying	source_airport_code character varying	destination_airport_code character varying	status character varying	delay_code character varying	scheduled_departure character varying	scheduled_arrival character varying	travel_duration_hours character varying
1	F4374	Air India	DEL	MAA	Completed		02-01-2025 15:26	17-01-2025 10:42	355.27
2	F2719	Air India	BOM	MAA	Delayed	ATC	11-01-2025 9:04	11-01-2025 18:25	9.34
3	F8350	GoAir	BLR	MAA	Cancelled		04-01-2025 21:55	10-01-2025 10:07	132.2
4	F3263	Vistara	BOM	HYD	Cancelled		14-01-2025 0:04	18-01-2025 12:26	108.38
5	F4863	Air India	HYD	BOM	Cancelled		08-01-2025 16:44	16-01-2025 0:25	175.69
6	F7816	SpiceJet	BOM	HYD	Completed		06-01-2025 17:04	18-01-2025 19:15	290.18

OPERATIONS:



Query Query History Scratch Pad

```sql
CREATE TABLE OPERATIONS_EXT (
    Operation_Id VARCHAR(250),
    Airport_Code VARCHAR(100),
    Operation_Type VARCHAR(100),
    Flight_ID VARCHAR(100),
    Operation_Time VARCHAR(100),
    Operator_Name VARCHAR(100),
    Operator_Contact VARCHAR(100)
);

SELECT * FROM "extract".operations_ext
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	operation_id character varying	airport_code character varying	operation_type character varying	flight_id character varying	operation_time character varying	operator_name character varying	operator_contact character varying
1	7b11c2d8-bb73-48ba-a88a-b5f662008b...	BLR	Maintenance	F9148	2025-01-05 11:49:34	Dawn Cook	954-771-9195x19816
2	44c7c502-7b6a-49a9-aa0e-90611e3b4f...	BOM	Maintenance	F8817	2025-01-10 17:59:56	Francis Parker	955-438-4945x20856
3	ade0f3d7-d8f5-4b06-b28c-3edf3e4a70e2	BLR	Maintenance	F4297	2025-01-08 05:36:25	Christine Caldwell	262.663.3848
4	c0aaf456-f0ef-47d1-b5af-4b92f98836b5	MAA	Takeoff	F6980	2025-01-08 08:01:57	Frank Holder	001-990-887-8458x86283
5	5fa98565-a287-44d2-aeb3-03c0549f78...	MAA	Landing	F5000	2025-01-15 07:08:02	Elizabeth Bright	+1-760-367-3062x7431
6	5fe32b72-cc4b-47c2-be13-7b3a220e4b...	DEL	Maintenance	F4886	2025-01-05 11:02:22	Laurie Leonard	9507959892
7	a0814678-e1bb-404f-9ec6-03ce916640...	HYD	Landing	F5917	2025-01-10 00:07:59	Alexander Hurley	(740)205-8251

PASSENGER:

Query Query History

Scratch Pad

```sql
CREATE TABLE PASSENGERS_EXT (
    First_Name VARCHAR(100),
    Last_Name VARCHAR(100),
    Gender VARCHAR(100),
    DOB VARCHAR(100),
    Age_Group VARCHAR(100),
    Passenger_Type VARCHAR(100),
    Flight_ID VARCHAR(100),
    Payment_Method VARCHAR(100),
    Travel_Class VARCHAR(100),
    Meal_Preference VARCHAR(100),
    Total_Fare_Amount VARCHAR(100),
    Reward_Points VARCHAR(100),
    Service_Quality_Feedback VARCHAR(100),
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	first_name character varying	last_name character varying	gender character varying	dob character varying	age_group character varying	passenger_type character varying	flight_id character varying	payment_method character varying	travel_class character varying	meal_preference character varying	total_fare character varying	reward_p… character…
1	Brooke	French	Female	2001-07-04	60+	Repeat	F9907	Wallet	Economy	Vegetarian	13269.59	1326
2	Tamara	Matthews	Female	1999-08-07	41-60	First-time	F3006	UPI	Business	Vegetarian	18891.93	1889
3	Marissa	Byrd	Other	1978-09-10	18-25	First-time	F7654	Debit Card	Economy	Non-Vegetarian	5798.53	579
4	Thomas	Rodriguez	Male	2006-12-28	41-60	First-time	F5069	Cash	First Class	Gluten-Free	11884.93	1188
5	Regina	Gutierrez	Other	1992-06-18	60+	Repeat	F3402	Cash	Business	Non-Vegetarian	13963.32	1396
6	Jacqueline	Smith	Female	1972-09-05	18-25	Repeat	F6255	UPI	First Class	Vegetarian	9648.9	964

DELAY:

Query Query History

```sql
CREATE TABLE DELAY_EXT (
    Delay_Code VARCHAR(100),
    Description VARCHAR(250)
);

SELECT * FROM "extract".delay_ext
```

Data Output Messages Notifications

	delay_code character varying	description character varying
1	WTH	Weather-related delay
2	TEC	Technical issues
3	ATC	Air Traffic Control restrictions

TRANSFORM CODE:

FLIGHT:

Query Query History

```sql
CREATE TABLE flight_sfm (
    flight_id VARCHAR,
    airline VARCHAR,
    source_airport_code VARCHAR,
    destination_airport_code VARCHAR,
    status VARCHAR,
    delay_code VARCHAR,
    scheduled_departure TIMESTAMP,
    scheduled_arrival TIMESTAMP,
    travel_duration_hours DECIMAL
);

SELECT * FROM transform.flight_sfm
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	flight_id character varying	airline character varying	source_airport_code character varying	destination_airport_code character varying	status character varying	delay_code character varying	scheduled_departure timestamp without time zone	scheduled_arrival timestamp without time zone	travel_duration_hours numeric
1	F4374	Air India	DEL	MAA	Completed		2025-01-02 15:26:00	2025-01-17 10:42:00	355.27
2	F2719	Air India	BOM	MAA	Delayed	ATC	2025-01-11 09:04:00	2025-01-11 18:25:00	9.34
3	F8350	GoAir	BLR	MAA	Cancelled		2025-01-04 21:55:00	2025-01-10 10:07:00	132.2
4	F3263	Vistara	BOM	HYD	Cancelled		2025-01-14 00:04:00	2025-01-18 12:26:00	108.38
5	F4863	Air India	HYD	BOM	Cancelled		2025-01-08 16:44:00	2025-01-16 00:25:00	175.69
6	F7816	SpiceJet	BOM	HYD	Completed		2025-01-06 17:04:00	2025-01-18 19:15:00	290.18
7	F6176	Air India	HYD	MAA	In-Flight		2025-01-02 19:23:00	2025-01-04 03:34:00	32.19

OPERATIONS:

Query Query History

```sql
CREATE TABLE OPERATIONS_SFM (
    Operation_Id VARCHAR(250),
    Airport_Code VARCHAR(100),
    Operation_Type VARCHAR(100),
    Flight_ID VARCHAR(100),
    Operation_Time TIMESTAMP,
    Operator_Name VARCHAR(100),
    Operator_Contact VARCHAR(100)
);

SELECT * FROM transform.operations_sfm
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	operation_id character varying	airport_code character varying	operation_type character varying	flight_id character varying	operation_time timestamp without time zone	operator_name character varying	operator_contact character varying
1	7b11c2d8-bb73-48ba-a88a-b5f662008b...	BLR	Maintenance	F9148	2025-01-05 11:49:34	Dawn Cook	+9547719195
2	44c7c502-7b6a-49a9-aa0e-90611e3b4f...	BOM	Maintenance	F8817	2025-01-10 17:59:56	Francis Parker	+9554384945
3	ade0f3d7-d8f5-4b06-b28c-3edf3e4a70e2	BLR	Maintenance	F4297	2025-01-08 05:36:25	Christine Caldwell	+2626633848
4	c0aaf456-f0ef-47d1-b5af-4b92f98836b5	MAA	Takeoff	F6980	2025-01-08 08:01:57	Frank Holder	+0019908878458
5	5fa98565-a287-44d2-aeb3-03c0549f78...	MAA	Landing	F5000	2025-01-15 07:08:02	Elizabeth Bright	+17603673062
6	5fe32b72-cc4b-47c2-be13-7b3a220e4b...	DEL	Maintenance	F4886	2025-01-05 11:02:22	Laurie Leonard	+9507959892
7	a0814678-e1bb-404f-9ec6-03ce916640...	HYD	Landing	F5917	2025-01-10 00:07:59	Alexander Hurley	+7402058251

PASSENGER:

Query Query History

```sql
CREATE TABLE PASSENGERS_SFM (
    First_Name VARCHAR(100),
    Last_Name VARCHAR(100),
    Gender VARCHAR(100),
    DOB VARCHAR(100),
    Age_Group VARCHAR(100),
    Passenger_Type VARCHAR(100),
    Flight_ID VARCHAR(100),
    Payment_Method VARCHAR(100),
    Travel_Class VARCHAR(100),
    Meal_Preference VARCHAR(100),
    Total_Fare_Amount DECIMAL,
    Reward_Points INT,
    Service_Quality_Feedback INT,
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	first_name character varying	last_name character varying	gender character varying	dob date	age_group character varying	passenger_type character varying	flight_id character varying	payment_method character varying	travel_class character varying	meal_preference character varying	total_fare numeric	reward_points integer	servic intege
1	Brooke	French	Female	2001-07-04	60+	Repeat	F9907	Wallet	Economy	Vegetarian	13269.59	1326	
2	Tamara	Matthews	Female	1999-08-07	41-60	First-time	F3006	UPI	Business	Vegetarian	18891.93	1889	
3	Marissa	Byrd	Other	1978-09-10	18-25	First-time	F7654	Debit Card	Economy	Non-Vegetarian	5798.53	579	
4	Thomas	Rodriguez	Male	2006-12-28	41-60	First-time	F5069	Cash	First Class	Gluten-Free	11884.93	1188	
5	Regina	Gutierrez	Other	1992-06-18	60+	Repeat	F3402	Cash	Business	Non-Vegetarian	13963.32	1396	
6	Jacqueline	Smith	Female	1972-09-05	18-25	Repeat	F6255	UPI	First Class	Vegetarian	9648.9	964	

DELAY:

Query Query History

```sql
CREATE TABLE DELAY_SFM (
    Delay_Code VARCHAR(100),
    Description VARCHAR(250)
);

SELECT * FROM transform.delay_sfm
```

Data Output Messages Notifications

	delay_code character varying	description character varying
1	WTH	Weather-related delay
2	TEC	Technical issues
3	ATC	Air Traffic Control restrictions

DIMENSION:

FLIGHT:

Query Query History

```
CREATE TABLE dimension.dim_flight
(
    flight_sk serial,
    flight_id_txt varchar,
    airline_txt varchar,
    status_txt varchar,
    delay_code_txt varchar,
    description_txt varchar,
    source_airport_code_txt varchar,
    destination_airport_code_txt varchar,
    date date,
    scheduled_departure timestamp without time zone,
    scheduled_arrival timestamp without time zone,
    travel_duration_hours_dc numeric,
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	flight_sk integer	flight_id_txt character varying	airline_txt character varying	status_txt character varying	delay_code_txt character varying	description_txt character varying	source_airport_code_txt character varying	destination_airport_code_txt character varying	date date	scheduled_departure timestamp without time zone	scheduled timestamp
1	1	F4374	Air India	Completed	[null]	[null]	DEL	MAA	2025-01-02	2025-01-02 15:26:00	2025-01-1
2	2	F2719	Air India	Delayed	ATC	Air Traffic Control restrictions	BOM	MAA	2025-01-11	2025-01-11 09:04:00	2025-01-1
3	3	F8350	GoAir	Cancelled	[null]	[null]	BLR	MAA	2025-01-04	2025-01-04 21:55:00	2025-01-1
4	4	F3263	Vistara	Cancelled	[null]	[null]	BOM	HYD	2025-01-14	2025-01-14 00:04:00	2025-01-1
5	5	F4863	Air India	Cancelled	[null]	[null]	HYD	BOM	2025-01-08	2025-01-08 16:44:00	2025-01-1
6	6	F7816	SpiceJet	Completed	[null]	[null]	BOM	HYD	2025-01-06	2025-01-06 17:04:00	2025-01-1

OPERATIONS:

Query Query History

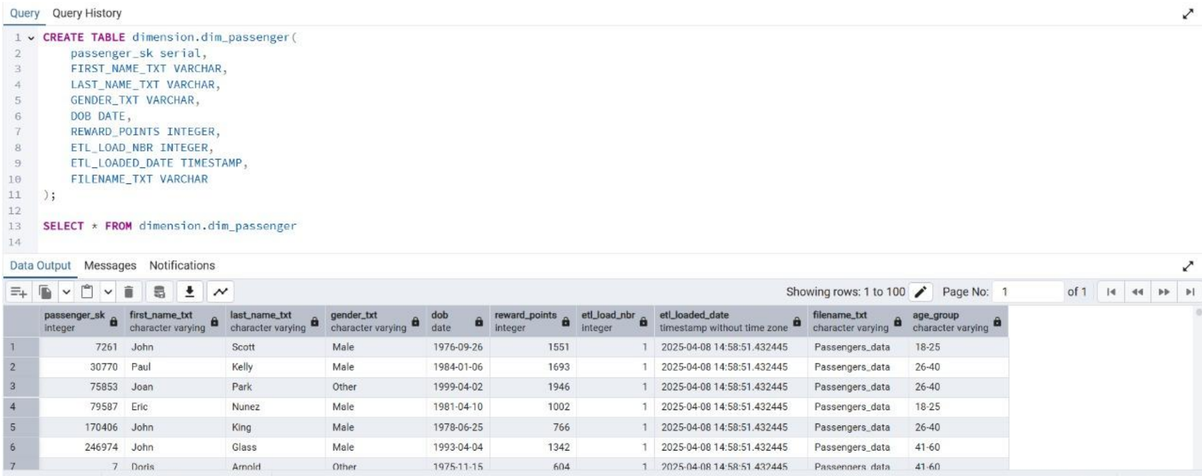
```
CREATE TABLE DIMENSION.DIM_OPERATIONS (
    OPERATION_SK SERIAL PRIMARY KEY,
    OPERATION_ID_TXT VARCHAR,
    FLIGHT_ID_TXT VARCHAR,
    AIRPORT_CODE_TXT VARCHAR,
    OPERATION_TYPE_TXT VARCHAR,
    OPERATION_TIME TIMESTAMP,
    OPERATOR_CONTACT_TXT VARCHAR,
    OPERATOR_NAME_TXT VARCHAR,
    ETL_LOAD_NBR INT,
    ETL_LOADED_DATE TIMESTAMP,
    FILENAME_TXT VARCHAR
);
select * from dimension.dim_operations
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	operation_sk [PK] integer	operation_id_txt character varying	flight_id_txt character varying	airport_code_txt character varying	operation_type_txt character varying	operation_time timestamp without time zone	operator_contact_txt character varying	operator_name_txt character varying	etl_load_nbr integer	etl_loaded_date timestamp without time zon
1	1	7b11c2d8-bb73-48ba-a88a-b5f662008b...	F9148	BLR	Maintenance	2025-01-05 11:49:34	+9547719195	Dawn Cook	1	2025-04-08 14:58:51.43250
2	2	44c7c502-7b6a-49a9-aa0e-90611e3b4f...	F8817	BOM	Maintenance	2025-01-10 17:59:56	+9554384945	Francis Parker	1	2025-04-08 14:58:51.43250
3	3	ade0f3d7-d8f5-4b06-b28c-3edf3e4a70e2	F4297	BLR	Maintenance	2025-01-08 05:36:25	+2626633848	Christine Caldwell	1	2025-04-08 14:58:51.43250
4	4	c0aaf456-f0ef-47d1-b5af-4b92f98836b5	F6980	MAA	Takeoff	2025-01-08 08:01:57	+0019908878458	Frank Holder	1	2025-04-08 14:58:51.43250
5	5	5fa98565-a287-44d2-aeb3-03c0549f78...	F5000	MAA	Landing	2025-01-15 07:08:02	+17603673062	Elizabeth Bright	1	2025-04-08 14:58:51.43250
6	6	5fe32b72-cc4b-47c2-be13-7b3a220e4b...	F4886	DEL	Maintenance	2025-01-05 11:02:22	+9507959892	Laurie Leonard	1	2025-04-08 14:58:51.43250

PASSENGER:



Query Query History

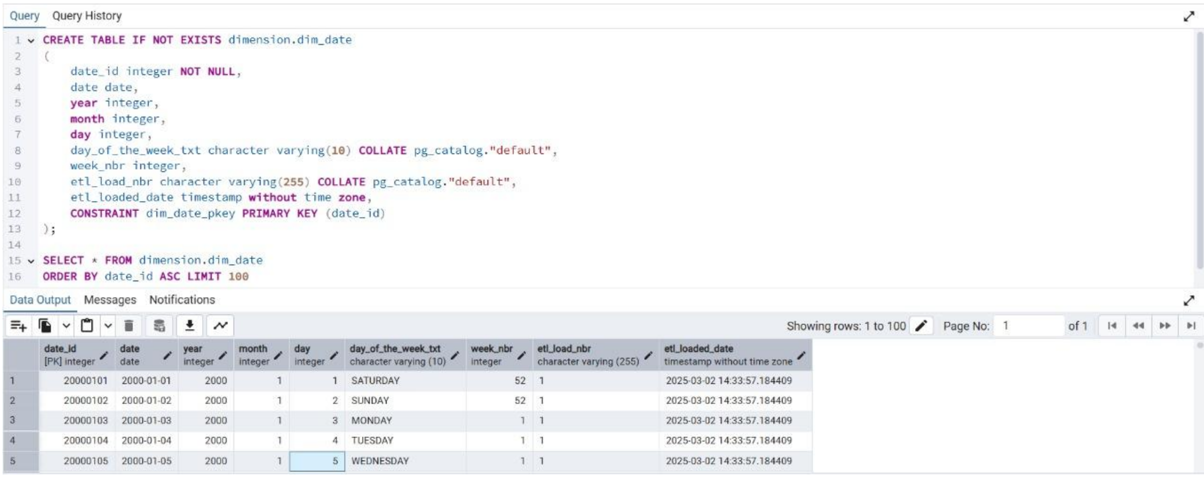
```sql
CREATE TABLE dimension.dim_passenger(
    passenger_sk serial,
    FIRST_NAME_TXT VARCHAR,
    LAST_NAME_TXT VARCHAR,
    GENDER_TXT VARCHAR,
    DOB DATE,
    REWARD_POINTS INTEGER,
    ETL_LOAD_NBR INTEGER,
    ETL_LOADED_DATE TIMESTAMP,
    FILENAME_TXT VARCHAR
);

SELECT * FROM dimension.dim_passenger
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	passenger_sk integer	first_name_txt character varying	last_name_txt character varying	gender_txt character varying	dob date	reward_points integer	etl_load_nbr integer	etl_loaded_date timestamp without time zone	filename_txt character varying	age_group character varying
1	7261	John	Scott	Male	1976-09-26	1551	1	2025-04-08 14:58:51.432445	Passengers_data	18-25
2	30770	Paul	Kelly	Male	1984-01-06	1693	1	2025-04-08 14:58:51.432445	Passengers_data	26-40
3	75853	Joan	Park	Other	1999-04-02	1946	1	2025-04-08 14:58:51.432445	Passengers_data	26-40
4	79587	Eric	Nunez	Male	1981-04-10	1002	1	2025-04-08 14:58:51.432445	Passengers_data	18-25
5	170406	John	King	Male	1978-06-25	766	1	2025-04-08 14:58:51.432445	Passengers_data	26-40
6	246974	John	Glass	Male	1993-04-04	1342	1	2025-04-08 14:58:51.432445	Passengers_data	41-60
7	7	Doris	Arnold	Other	1975-11-15	604	1	2025-04-08 14:58:51.432445	Passengers_data	41-60

DATE:



Query Query History

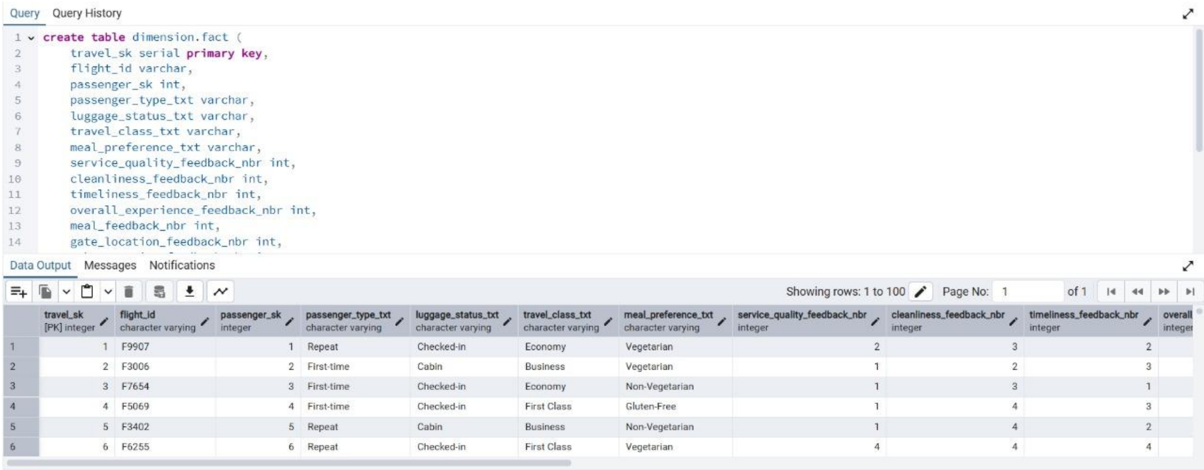
```sql
CREATE TABLE IF NOT EXISTS dimension.dim_date
(
    date_id integer NOT NULL,
    date date,
    year integer,
    month integer,
    day integer,
    day_of_the_week_txt character varying(10) COLLATE pg_catalog."default",
    week_nbr integer,
    etl_load_nbr character varying(255) COLLATE pg_catalog."default",
    etl_loaded_date timestamp without time zone,
    CONSTRAINT dim_date_pkey PRIMARY KEY (date_id)
);

SELECT * FROM dimension.dim_date
ORDER BY date_id ASC LIMIT 100
```

Data Output Messages Notifications

Showing rows: 1 to 100 Page No: 1 of 1

	date_id [PK] integer	date date	year integer	month integer	day integer	day_of_the_week_txt character varying (10)	week_nbr integer	etl_load_nbr character varying (255)	etl_loaded_date timestamp without time zone
1	20000101	2000-01-01	2000	1	1	SATURDAY	52	1	2025-03-02 14:33:57.184409
2	20000102	2000-01-02	2000	1	2	SUNDAY	52	1	2025-03-02 14:33:57.184409
3	20000103	2000-01-03	2000	1	3	MONDAY	1	1	2025-03-02 14:33:57.184409
4	20000104	2000-01-04	2000	1	4	TUESDAY	1	1	2025-03-02 14:33:57.184409
5	20000105	2000-01-05	2000	1	5	WEDNESDAY	1	1	2025-03-02 14:33:57.184409

FACT:



Query Query History

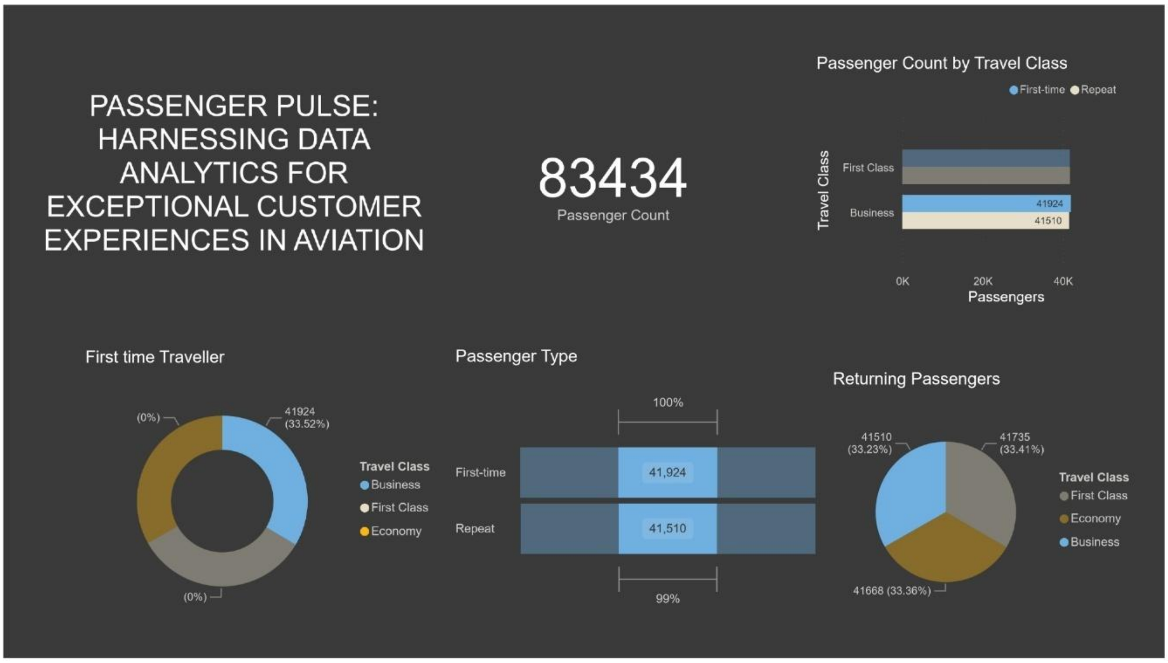
```
create table dimension.fact (
    travel_sk serial primary key,
    flight_id varchar,
    passenger_sk int,
    passenger_type_txt varchar,
    luggage_status_txt varchar,
    travel_class_txt varchar,
    meal_preference_txt varchar,
    service_quality_feedback_nbr int,
    cleanliness_feedback_nbr int,
    timeliness_feedback_nbr int,
    overall_experience_feedback_nbr int,
    meal_feedback_nbr int,
    gate_location_feedback_nbr int,
```

Data Output Messages Notifications

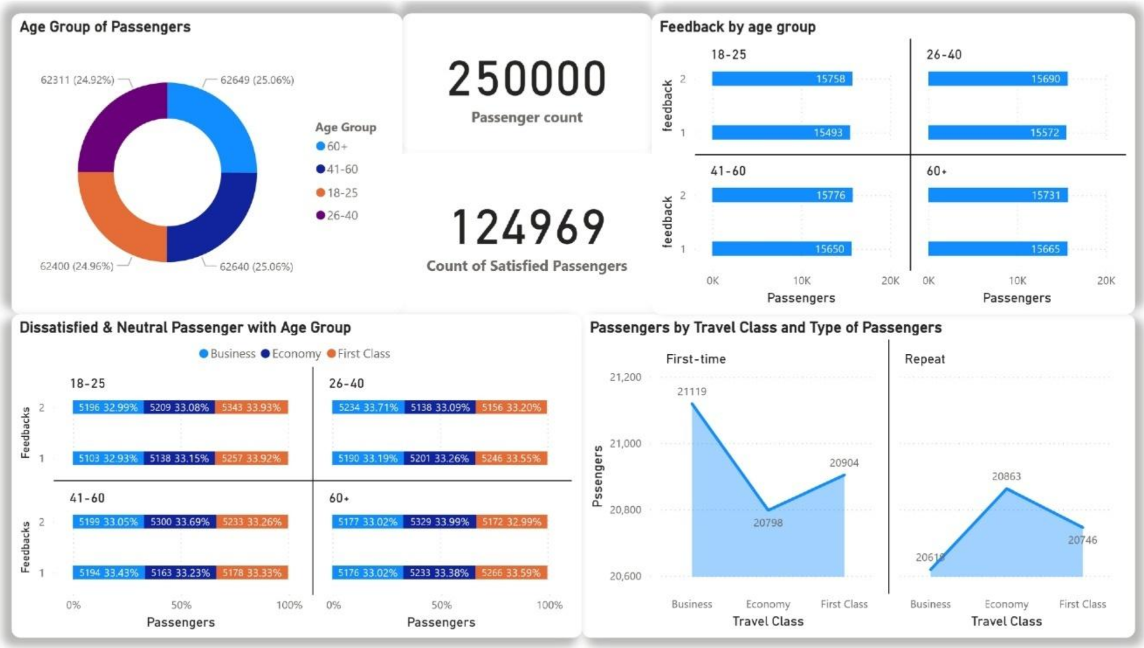
Showing rows: 1 to 100 Page No: 1 of 1

	travel_sk [PK] integer	flight_id character varying	passenger_sk integer	passenger_type_txt character varying	luggage_status_txt character varying	travel_class_txt character varying	meal_preference_txt character varying	service_quality_feedback_nbr integer	cleanliness_feedback_nbr integer	timeliness_feedback_nbr integer	overall integer
1	1	F9907	1	Repeat	Checked-in	Economy	Vegetarian	2	3	2	
2	2	F3006	2	First-time	Cabin	Business	Vegetarian	1	2	3	
3	3	F7654	3	First-time	Checked-in	Economy	Non-Vegetarian	1	3	1	
4	4	F5069	4	First-time	Checked-in	First Class	Gluten-Free	1	4	3	
5	5	F3402	5	Repeat	Cabin	Business	Non-Vegetarian	1	4	2	
6	6	F6255	6	Repeat	Checked-in	First Class	Vegetarian	4	4	4	

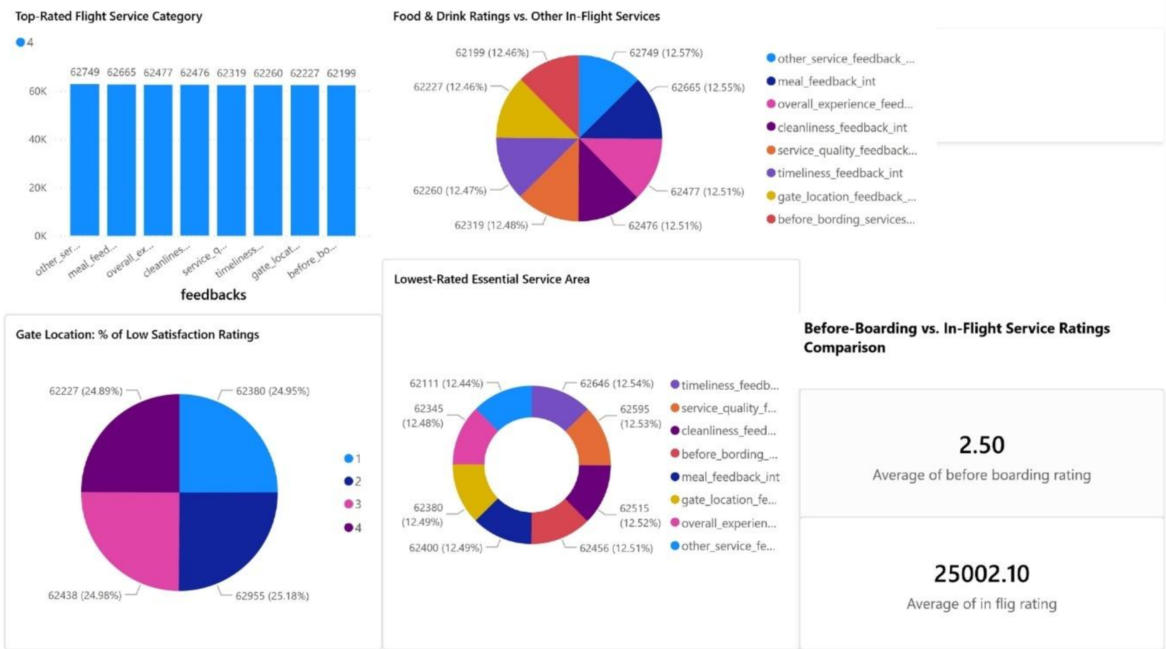
Customer Types & Travel Class:



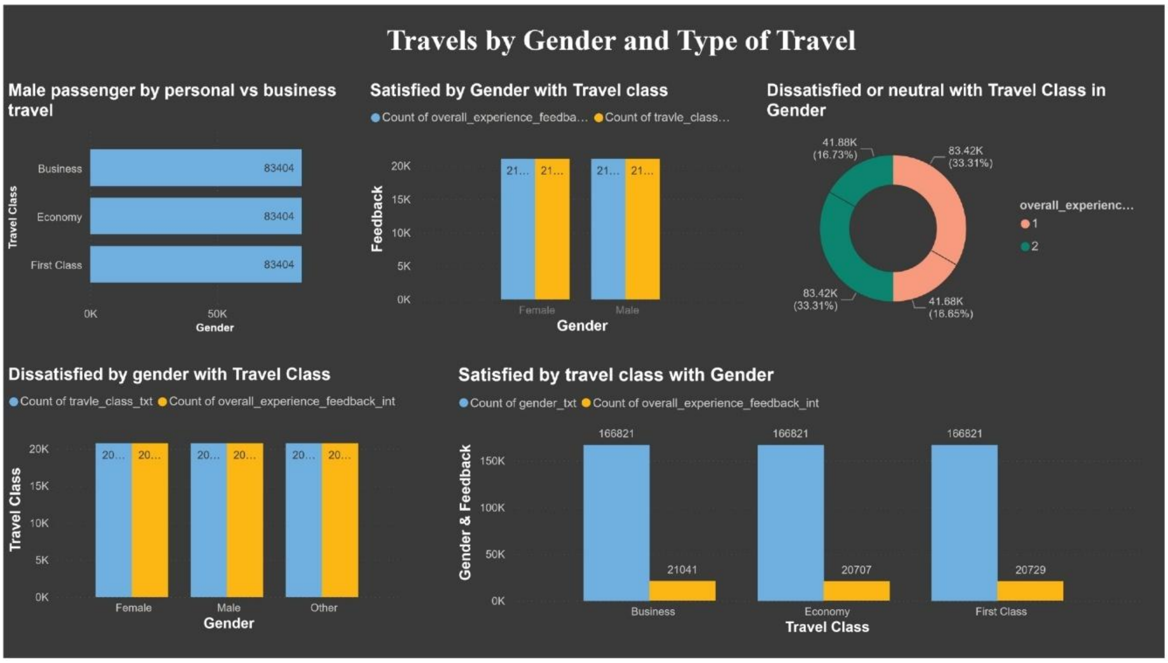
Travel by Age Groups:



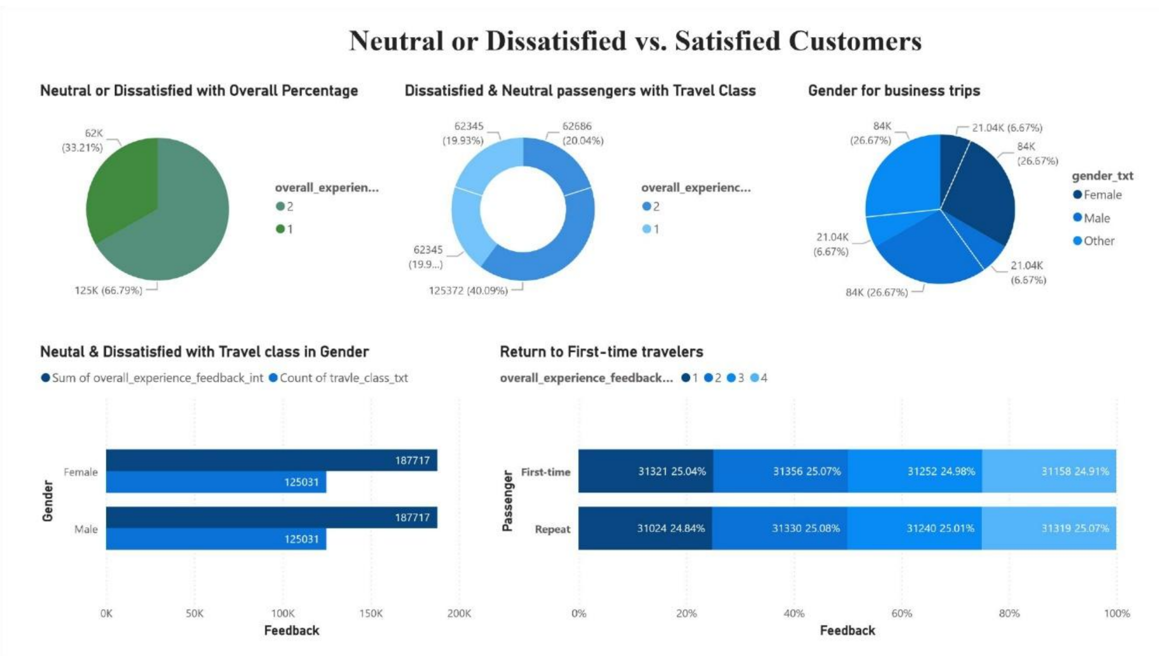
Our Services Rating:



Travel by Gender and Type of Travel:



Neutral or Dissatisfied Vs. Satisfied Customers



General Insights and Recommendations

CHAPTER – 11

CONCLUSION

In conclusion, the integration of data analytics into the aviation industry-exemplified by the "Passenger Pulse" initiative-has fundamentally transformed how airlines and airports understand and enhance the customer experience. By harnessing vast amounts of passenger data, airlines can now identify key trends, segment customer profiles, and deliver highly personalized services that cater to individual preferences, such as tailored flight recommendations, customized in-flight offerings, and proactive service recovery. This data-driven approach not only elevates passenger satisfaction but also fosters greater brand loyalty and repeat business.

Furthermore, data analytics empowers airlines to optimize operational efficiency, from resource allocation and flight scheduling to predictive maintenance and fuel management, resulting in fewer delays, cost reductions, and a smoother travel experience for passengers. Real-time data analysis enables airlines to anticipate disruptions, dynamically adjust operations, and maintain high safety standards, all of which are crucial in today’s competitive and high-demand aviation environment.

Ultimately, "Passenger Pulse" demonstrates that leveraging advanced analytics is no longer optional but essential for delivering exceptional customer experiences in aviation. As the industry continues to evolve, data analytics will remain at the core of innovation, driving smarter decisions, operational excellence, and a truly customer-centric future for air travel.

CHAPTER - 12

FUTURE ENHANCEMENT

This project has laid a strong foundation for leveraging data-driven insights to improve passenger satisfaction and operational efficiency. Looking ahead, several future enhancements can further elevate the impact and scope of this initiative. One promising avenue is the integration of artificial intelligence (AI) and machine learning (ML) algorithms to enable even more precise predictive analytics. By continuously learning from new data, these systems can anticipate passenger needs, forecast demand patterns, and suggest personalized services in real time. For example, AI-powered chatbots could offer instant support for itinerary changes, baggage tracking, and in-flight requests, making the travel experience smoother and more responsive.

Another enhancement involves expanding data sources to include biometric and IoT (Internet of Things) data. By incorporating facial recognition for seamless check-ins, wearable devices for real-time health monitoring, and smart sensors for tracking luggage or monitoring cabin conditions, airlines can create a more integrated and frictionless journey for passengers. This would not only improve convenience but also enhance safety and security throughout the travel process.

Additionally, future versions of Passenger Pulse could focus on advanced sentiment analysis by mining social media, customer feedback, and review platforms. This would allow airlines to proactively address emerging issues, gauge public perception, and adapt services to changing expectations more rapidly.

Finally, the project can be enhanced by fostering greater collaboration with airports, regulatory bodies, and partner airlines. Establishing secure data-sharing frameworks and industry-wide standards will enable a holistic view of the passenger journey, from booking to arrival, ensuring a consistently high level of service across all touchpoints.

By embracing these future enhancements, Passenger Pulse can continue to set new benchmarks for customer experience in aviation, driving innovation and building lasting loyalty in an increasingly competitive industry.

CHAPTER - 13

REFERENCES

1) Pereira, F., Costa, J.M., Ramos, R. and Raimundo, A., 2023. The impact of the COVID-19 pandemic on airlines' passenger satisfaction. *Journal of Air Transport Management*, *112*, p.102441.
2) Noviantoro, T. and Huang, J.P., 2022. Investigating airline passenger satisfaction: Data mining method. *Research in Transportation Business & Management*, *43*, p.100726.
3) Law, C.C., Zhang, Y. and Gow, J., 2022. Airline service quality, customer satisfaction, and repurchase Intention: Laotian air passengers' perspective. *Case Studies on Transport Policy*, *10*(2), pp.741-750.
4) Jiang, X., Zhang, Y., Li, Y. and Zhang, B., 2022. Forecast and analysis of aircraft passenger satisfaction based on RF-RFE-LR model. *Scientific reports*, *12*(1), p.11174.
5) Hayadi, B.H., Kim, J.M., Hulliyah, K. and Sukmana, H.T., 2021. Predicting airline passenger satisfaction with classification algorithms. *International Journal of Informatics and Information Systems*, *4*(1), pp.82-94.
6) Tahanisaz, S., 2020. Evaluation of passenger satisfaction with service quality: A consecutive method applied to the airline industry. *Journal of Air Transport Management*, *83*, p.101764.
7) Lestari, Y.D. and Murjito, E.A., 2020. Factor determinants of customer satisfaction with airline services using big data approaches. *Jurnal Pendidikan Ekonomi Dan Bisnis (JPEB)*, *8*(1), pp.34-42.
8) Park, E., Jang, Y., Kim, J., Jeong, N.J., Bae, K. and Del Pobil, A.P., 2019. Determinants of customer satisfaction with airline services: An analysis of customer feedback big data. *Journal of Retailing and Consumer Services*, *51*, pp.186-190.

9) Ban, H.J. and Kim, H.S., 2019. Understanding customer experience and satisfaction through airline passengers' online review. *Sustainability*, *11*(15), p.4066.
10) Kumar, S. and Zymbler, M., 2019. A machine learning approach to analyze customer satisfaction from airline tweets. *Journal of Big Data*, *6*(1), pp.1-16.