Airline Delay Prediction System Documentation

Team Members:

- Mostafa Mohamed Khafaga (231027979)
- Marwan Abdelaziz Mohamed (231017947)

Presented To:

- Dr. Ahmed Habashy
- TA. Amr Daba

---

1. Executive Summary

This project implements a Machine Learning system designed to predict flight delays. It addresses a critical challenge in the aviation industry: calculating the probability of a delay based on historical data and real-time environmental factors.

The system utilizes a Hybrid Intelligence approach:

1. Statistical Model: A Logistic Regression model trained on historical flight data to handle static factors (Airline, Route, Time).

2. Rule-Based Heuristic: A logic layer that overlays dynamic factors (Weather) which are often missing from training datasets but crucial for real-world accuracy.

The project includes a training pipeline (`algorithm`), a shared inference engine (`utils`), and a user-facing dashboard (`GUI`).

---

2. System Architecture

The project is structured into three main layers:

A. Data Processing & Training Layer (`airline_delay_prediction.py`)

- Responsibility: Ingests raw CSV data, cleans it, trains the model, and validates performance.
- Key Output: Generates the "Brain" of the system (`.pkl` model files) and configuration artifacts.

B. Inference & Logic Layer (`prediction_utils.py`)

- Responsibility: Acts as the bridge between the trained model and the application.
- Functionality: Serves multiple roles including loading saved model artifacts, transforming raw inputs into model-ready vectors, applying `Marginal Probability Contribution` math to explain predictions, and applying

Heuristic Logic (e.g., "If Snowing -> Risk +30%").

C. Presentation Layer (`delay_prediction_gui.py`)

• Responsibility: Provides an interactive dashboard for end-users.

• Features: Simulation scenarios, visual feedback (Red/Green indicators), and detailed explanation popups.

---

3. Technical Implementation

3.1 Algorithm Selection: Logistic Regression

The project focuses on Logistic Regression for binary classification (Delayed vs. On-Time). Two regularization techniques were implemented and compared:

- L2 Regularization (Ridge):

• Goal: Minimize overfitting by penalizing large coefficients.

• Outcome: Generally achieves higher stability and accuracy by keeping all features but reducing their impact if they are redundant.

- L1 Regularization (Lasso):

• Goal: Perform feature selection by forcing unimportant coefficients to zero.

• Outcome: Creates a sparse model. In this project, it was used to analyze which features are statistically irrelevant.

3.2 Data Preprocessing

Machine Learning models require formatted data. The pipeline (using `scikit-learn`) performs:

1. Imputation: Filling missing values (Median for numbers, "Missing" constant for categories).

2. Scaling: Using `StandardScaler` to normalize numerical ranges (e.g., converting Distance [0-5000] and Month [1-12] to the same scale).

3. Encoding: Using `OneHotEncoder` to convert categorical text (e.g., "AA", "UA") into mathematical vectors.

3.3 Mathematical Foundation

The prediction engine relies on Logistic Regression. While the name sounds complex, the logic is straightforward: it calculates a "score" for the flight and then converts that score into a probability (0% to 100%).

Step 1: calculating the Score (The Linear Part)

Imagine every feature adds or subtracts points from a "Delay Score" (z).

- Base Score: The starting point (Intercept).

- Feature Points: value * weight.

The formula is a simple sum:

`Score (z) = Base + (Weight1 * Feature1) + (Weight2 * Feature2) + ...`

- *Example*: If Snow has a weight of +2.0 and Distance has a weight of +0.001, a snowy flight adds 2 points, and a long flight adds a few fraction points.

Step 2: The Probability (The Sigmoid Part)

A "Score" can be anything (e.g., -5, +10, +100). To turn this into a probability between 0 and 1, we usage the Sigmoid Function.

`Probability = 1 / (1 + e^(-Score))`

- If the Score is 0, the Probability is 50%.
- If the Score is High Positive (e.g., +5), the Probability nears 100%.
- If the Score is High Negative (e.g., -5), the Probability nears 0%.

3.4 Explainable AI (XAI) Matrix

Traditional models give you a prediction but don't tell you *why*. We solved this by using Marginal Probability Contribution.

The Problem:

Weights (β) are in "Log-Odds" units. Telling a user "The coefficient is 0.4" is meaningless.

The Solution:

We approximate how much *actual percentage* probability each feature added. The math is:

`Impact ~= Weight * Feature_Value * (Probability * (1 - Probability))`

This tells us the specific contribution of that factor *at that specific moment*.

- *Interpretation*: "Because the probability is near 50%, a small change in weight has a huge impact. But if the probability is already 99%, adding more weight won't change much."

This allows the GUI to say: "Snow increased your risk by 15%."

---

4. Installation & Setup

Prerequisites

- Python 3.8 or higher.
- Libraries listed below.

Libraries Used

- pandas: The backbone of data manipulation. Used to load the CSV dataset, filter rows, and handle missing values.
- numpy: Performs high-performance numerical calculations and matrix operations required by the machine learning algorithms.
- scikit-learn: The core Machine Learning framework. It provides the Logistic Regression algorithm, data scalers (`StandardScaler`), category encoders (`OneHotEncoder`), and evaluation metrics.
- matplotlib & seaborn: Visualization libraries used to generate the "Exploratory Data Analysis" charts (Heatmaps, Bar plots).
- joblib: Handles model serialization, allowing us to "save" the trained brain of the AI to a file (`.pkl`) and load it later in the GUI.
- tkinter: The standard Python GUI toolkit used to build the interactive desktop dashboard.
- fpdf: A lightweight PDF generation library used to create this documentation file programmatically.

Setup Steps

1. Clone/Download the project folder.
2. Install Dependencies:

   ```bash

   pip install -r requirements.txt

   ```
3. Generate Data:

   If `filtered_flight_data.csv` is missing, run:

   ```bash

   python generate_data.py

   ```

---

5. Usage Guide

5.1 Training the Model

To retrain the system (e.g., after updating data), run:

```
python airline_delay_prediction.py
```

What happens:

- The script loads date.
- It performs EDA (Exploratory Data Analysis) and saves graphs.

- It trains both L1 and L2 models.
- It compares them and saves the best model to disk (`.pkl` files).
- Result: You will see a "Model Comparison Report" in the terminal.

5.2 Running the Dashboard

To use the application for predictions:

```
python delay_prediction_gui.py
```

Features:

- Scenario Table: Choose from pre-loaded flight scenarios (e.g., "Holiday Rush", "Winter Storm").
- Predict: Click the "Predict" button to analyze the selected flight.
- Analysis Popup: A window appears showing:
- Prediction: DELAYED or ON-TIME.
- Probability: The calculated % chance.
- Reasoning: A ranked list of factors (e.g., "Weather: Snow - Increases Risk [!]").

---

6. Code Reference

`airline_delay_prediction.py`

- `load_data()`: Reads CSV.
- `preprocess_data()`: Builds the Scikit-Learn pipeline.
- `train_model()`: Fits Logistic Regression.
- `evaluate_model()`: Calcs Recision, Recall, F1, AUC.

`prediction_utils.py`

- `calculate_probability_contribution()`: The core "Explainability" engine. It takes a raw flight row, runs it through the model components, calculates how much each feature contributed to the final score, and adds the weather heuristic.

`delay_prediction_gui.py`

- `DelayPredictionApp`: The main Tkinter class.
- `populate_dashboard()`: Loads scenarios and pre-calculates results for display.
- `show_prediction_popup()`: Renders the detailed analysis window.