Flight Delay Study with Machine Learning

Nitya Sai Mounisha Korada
11710835
ADTA 5340
University of North Texas
Denton, Texas, USA

V S Sai Kanaka Raju Bonam
11704943
ADTA 5340
University of North Texas
Denton, Texas, USA

Bharadwaj Sakhamuri
11654364
ADTA 5340
University of North Texas
Denton, Texas, USA

Ranganadha Pawan Parankusam
11594074
ADTA 5340
University of North Texas
Denton, Texas, USA

***Abstract*—This paper explores the application of machine learning models to predict flight delays, leveraging a dataset containing various attributes related to flight performance. The data set includes features such as flight dates, departure delays, weather conditions, and airport-specific details. In order to ensure the reliability of the model, a series of pre-processing steps were performed, including handling missing data, outlier detection, and feature selection. Several machine learning models were evaluated to predict flight delays, including Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines (SVM). These models were assessed using performance metrics such as accuracy, precision, recall, F1-score, and ROC curves. Based on the results, Random Forests and Support Vector Machines exhibited strong performance in predicting delays. Additionally, feature importance analysis was conducted to understand which factors most significantly influenced delay predictions. The findings of this study provide insights into how machine learning can be effectively utilized to predict flight delays and assist in operational decision-making processes for airlines.**

***Index Terms*—Flight delay, machine learning, prediction models, Random Forest, Support Vector Machines, feature selection.**

I. Introduction and Problem Statement

Air travel has become an essential part of modern life, facilitating the global movement of passengers and goods. However, one of the most persistent challenges facing the aviation industry is flight delays. These delays disrupt travel plans, cause significant economic losses for airlines, and result in passenger dissatisfaction. The U.S. Department of Transportation reports that flight delays cost airlines billions of dollars annually, affecting millions of passengers each year [**?**]. While delays may be caused by various factors such as weather, mechanical issues, or airport congestion, accurately predicting these delays has proven to be a complex task. Traditionally, airlines have relied on historical data and rule-based systems for delay predictions, but these methods often fail to account for the complex and dynamic nature of flight operations. Recent advancements in machine learning present an opportunity to improve these predictions by leveraging large datasets that include numerous variables such as weather patterns, flight schedules, historical delays, and airport-specific data. The ability to predict flight delays with greater accuracy can lead to significant improvements in airline operations, customer service, and resource management. This research aims to address the problem of flight delay prediction using machine learning techniques, focusing on developing predictive models that can accurately forecast delays based on a wide array of influencing factors. By evaluating machine learning algorithms, such as Random Forests, Support Vector Machines, and Logistic Regression, we seek to identify the most effective models for predicting delays, improving prediction accuracy, and providing actionable insights for airline operators. In doing so, this paper aims to contribute to the growing body of knowledge in predictive analytics within the aviation industry. Furthermore, the study will examine the importance of various features, such as weather conditions, airport congestion, and historical performance, in predicting delays and assess their impact on model accuracy. The motivation behind this research is to offer practical solutions to the challenges posed by flight delays, providing airlines with tools to optimize scheduling, allocate resources more efficiently, and reduce operational costs. For passengers, accurate delay predictions can improve the travel experience by providing timely information, allowing them to make informed decisions and avoid unnecessary inconveniences. Ultimately, this research seeks to bridge the gap between the complexity of flight delay prediction and the need for real-time, reliable forecasting, thus helping both airlines and passengers better navigate the uncertainties of air travel.

II. Review of Literature

Flight delay prediction has been an important area of research due to its significant impact on airline operations and passenger satisfaction. Many machine learning (ML) techniques have been explored to enhance prediction accuracy by considering various factors such as weather conditions, flight characteristics, and airport operations. Kim et al. (2016) proposed a deep learning-based approach for flight delay prediction that utilized both historical flight data and weather conditions. Their model employed deep neural networks (DNNs) to process large amounts of flight data, including departure times, airlines, and historical delay patterns. The results showed that the model achieved high accuracy, particularly for short- and medium-haul flights, where consistent data patterns were available. However, the model struggled with incorporating real-time data, especially sudden weather changes that could disrupt predictions [1]. Gui et al. (2019) explored the use of aviation big data in combination with machine learning algorithms, such as Random Forest (RF) and Gradient Boosting Machines (GBM), to predict flight delays. Their ensemble approach combined multiple models to analyze variables like weather, airport congestion, and historical delay information. This approach significantly outperformed traditional methods, particularly at airports with large passenger volumes. One of the limitations was the high computational cost of ensemble models, which limited their applicability in real-time systems. Additionally, incomplete data and unpredictable events impacted prediction accuracy [2]. Yu et al. (2019) applied Convolutional Neural Networks (CNN) to predict flight delays by analyzing flight attributes like departure time, weather conditions, and other flight-specific variables. Their CNN-based model proved highly accurate in predicting both scheduled and unscheduled delays, demonstrating its ability to handle unpredicted events. Despite the model's effectiveness, it required large labeled datasets for training, limiting its scalability. Moreover, the model had difficulty predicting delays caused by human errors and maintenance issues [3]. Hatıpoğlu et al. (2022) focused on flight delay prediction using Decision Trees (DT) and Support Vector Machines (SVM), considering factors such as weather, flight characteristics, and airport operations. Their models demonstrated good balance between prediction accuracy and computational efficiency, especially in predicting delays based on airport-specific data and historical performance. However, the study highlighted the challenge of predicting real-time delays due to unforeseen disruptions like unexpected weather changes and maintenance issues, suggesting the need for continuous updates to the models as new data becomes available [4]. Chen and Li (2019) proposed a chained prediction method for flight delays, which aimed to predict delays across connected flight legs. By considering the dependencies between multiple legs of a journey, their model improved delay prediction accuracy for passengers on multi-leg flights. The model outperformed traditional methods in this context but still faced limitations when accounting for external factors such as extreme weather or airport congestion, which could disrupt multiple flights in the sequence [5]. Mokhtarimousavi and Mehrabi (2023) investigated the causal factors of flight delays by applying machine learning techniques combined with random parameter statistical analysis. Their study found that both flight-specific factors, like aircraft type and route, and weather conditions played significant roles in determining delays. However, the study's model did not account for unobserved factors such as human errors, maintenance issues, and operational inefficiencies, which are also crucial contributors to delays [6]. Choi et al. (2016) focused on weather-induced flight delays and applied machine learning models to predict delays caused by extreme weather conditions, such as wind speed, temperature, and precipitation. Their models successfully identified the strong correlation between these weather variables and flight delays. However, the granularity of weather data collected at airports varied, which reduced the accuracy of predictions in some cases. This limitation suggests that improving the precision of weather data could enhance model performance [7]. Thiagarajan et al. (2017) developed a machine learning model for predicting the on-time performance of flights, incorporating variables like flight-specific features, departure time, and airport congestion. Their model performed well under controlled conditions but faced challenges in integrating real-time data, such as last-minute cancellations and sudden maintenance issues. This highlighted the need for incorporating dynamic inputs to improve predictions under operational disruptions [8].Esmaeilzadeh and Mokhtarimousavi (2020) combined time-series analysis with machine learning techniques, such as Random Forest and XGBoost, to predict flight departure delays. The model performed well by capturing delay patterns during peak hours but struggled with predicting delays in real-time, particularly for unplanned cancellations and disruptions. This indicates that while the model performed well for specific time periods, it lacked flexibility in dynamic scenarios [9]. Yazdi et al. (2020) utilized deep learning models, specifically the Levenberg-Marquardt algorithm, to predict flight delays. Their study highlighted the superior performance of deep learning models, especially for large datasets, where the algorithm helped reduce training time and improve prediction accuracy. However, the study revealed that deep learning models struggled with predicting delays caused by external factors such as airport congestion and human error, pointing to the need for incorporating these variables into future models [10].

Objectives of the Study

The primary objectives of this study are to develop a predictive model for flight delays using machine learning techniques. The study involves a comprehensive process that includes data preprocessing, feature engineering, exploratory data analysis (EDA), model training, and performance evaluation. The key objectives are as follows:

1) **Import and Inspect the Dataset:**
    - The initial objective is to import the dataset and review its structure. This includes examining the dataset's columns, types, and first few records to understand the format of the data.
    - The data inspection will help identify important columns relevant to the study and understand the overall data structure.
2) **Handle Missing Values and Remove Irrelevant Columns:**
    - Identify and handle any missing values within the dataset by using appropriate techniques such as imputation or removal.
    - Remove any irrelevant columns that do not contribute meaningfully to the analysis, such as identifiers or columns with excessive missing values.
    - Preprocess categorical variables, converting them into appropriate numerical formats using encoding techniques like one-hot encoding or label encoding.
3) **Create New Features:**
    - Generate new features that may improve model performance, such as breaking down the date/time

column into smaller components like year, month, day, hour, and day of the week.
    - Transform existing features to enhance their utility in predicting flight delays (e.g., creating a binary variable for weekends or holidays).
4) **Visualize Distributions and Relationships:**
    - Perform exploratory data analysis (EDA) to understand the distributions of the key features using visualizations such as histograms, box plots, and bar plots.
    - Explore relationships between features and target variables (flight delay) using scatter plots, correlation matrices, and heatmaps.
    - Identify any patterns, trends, or anomalies in the data that could inform feature selection or model choices.
5) **Split the Data and Train Multiple Models:**
    - Split the dataset into training and testing sets to evaluate the model's performance on unseen data by splitting data with 80/20 split.
    - Train multiple machine learning models (e.g., Logistic Regression, Decision Trees, Random Forest, KNN, and SVM) on the training dataset.
    - Apply cross-validation techniques to ensure the models are robust and can generalize well to unseen data.
6) **Assess Model Performance:**
    - Evaluate the performance of each trained model using several metrics, including accuracy, precision, recall, F1-score, and AUC-ROC.
    - Analyze the confusion matrix to better understand the misclassifications and the model's ability to predict flight delays.
    - Compare the performance of different models and choose the best-performing model based on the evaluation metrics.

III. DATA COLLECTION

The dataset used in this study is publicly available from the U.S. Department of Transportation. Specifically, the "Reporting Carrier On-Time Performance" dataset has been selected as it contains comprehensive details on flight schedules, performance outcomes, and delay causes, which are crucial for analyzing flight delays. The dataset spans from 1987 to the present and includes more than 100,000 records, offering a substantial amount of data for analysis. The data was collected by the Bureau of Transportation Statistics (BTS) and is made available through their official website. This dataset includes multiple variables relevant to understanding the time performance of flights, such as departure and arrival delays, flight cancellations, and other time-related data points.

Dataset Link:

- https://www.transtats.bts.gov/

A. Dataset Overview

The data set contains more than 100,000 records and includes more than 10 variables, each providing specific details on flight performance. The data set has 115,782 records and 31 features.

B. Data Collection Method

The data was collected by the Bureau of Transportation Statistics (BTS) from various sources, including airlines and airports, and are made available for public access through the TranStats system. Data are updated regularly and provide information on airline on-time performance, cancellations, delays, and the factors that contribute to these delays. The data is made available in a structured format, which can be accessed via the official U.S. Department of Transportation. The data is collected and stored in a manner that ensures it can be used for further analysis and research in the field of transportation, particularly for understanding and improving flight delays and related factors.

C. Dataset Variables and Descriptions

FL_DATE:
This variable represents the date of the flight. It is recorded in the format of YYYY-MM-DD. This allows tracking and analysis of flights over time and comparing performance on different days.
OP_UNIQUE_CARRIER:
The unique code identifying the airline that operates the flight. Each airline has a specific code that is used in the data set to distinguish one carrier from another. This is critical for analyzing the performance of different airlines.
TAIL_NUM:
The tail number of the aircraft, which is a unique identifier for each plane. It is used to track individual aircrafts across flights and can help in identifying performance trends related to specific aircraft.
OP_CARRIER_FL_NUM:
This is the flight number assigned by the carrier. It helps identify the specific flight operated by the airline and can be used to track the performance of particular flights.
ORIGIN:
The airport code of the departure airport. This variable is important for analyzing the performance of flights departing from different airports and regions.
ORIGIN_CITY_NAME:
This variable gives the name of the city where the departure airport is located. It provides additional geographical context for the origin of the flight.
ORIGIN_STATE_ABR:
The state abbreviation of the departure airport. This can be useful for understanding regional patterns in flight delays and cancellations.
DEST:
The airport code of the destination airport. Similar to the origin, this helps to track where the flight is headed and allows comparisons across different destination airports.

DEST_CITY_NAME:
This variable provides the name of the city where the destination airport is located. It complements the DEST variable by offering geographical context for flight destinations.

DEST_STATE_ABR:
The state abbreviation of the destination airport. Like the origin state, this can help to identify any regional performance trends for arrivals.

CRS_DEP_TIME:
Scheduled departure time (local time). This is the time when the flight was originally planned to depart. It is used to compare the actual departure time with the scheduled time and to track delays.

DEP_TIME:
The actual departure time of the flight. This allows the comparison with the scheduled departure time (CRS_DEP_TIME) and helps to track delays in departure.

DEP_DELAY:
This variable represents the departure delay in minutes. Positive values indicate that the flight was delayed. It provides insight into how much longer a flight took to depart compared to the scheduled time.

DEP_DEL15:
This is a binary indicator where a value of 1 indicates that the flight departed 15 minutes later than the scheduled departure time, and 0 indicates that the flight was not delayed by 15 minutes or more.

TAXI_OUT:
The time spent taxiing from the gate to the runway in minutes. This helps in understanding the time delays associated with ground operations.

TAXI_IN:
The time spent taxiing from the runway to the gate in minutes. Like TAXI_OUT, this variable provides information about ground operations but for the arrival phase.

CRS_ARR_TIME:
Scheduled arrival time (local time). This is the time when the flight was originally planned to land. It is used to compare with the actual arrival time to assess delays.

ARR_TIME:
The actual arrival time of the flight. This variable allows the comparison with the scheduled arrival time (CRS_ARR_TIME) to track arrival delays.

ARR_DELAY:
The arrival delay in minutes, which measures how late the flight was upon arrival compared to the scheduled arrival time. Positive values indicate delays.

ARR_DEL15:
This is a binary indicator where a value of 1 indicates that the arrival was delayed by 15 minutes or more, and 0 indicates that the arrival was on time or delayed less than 15 minutes.

CANCELLED:
A binary indicator where 1 means the flight was cancelled and 0 means it was not. This variable is important for tracking flight cancellations.

CANCELLATION_CODE:
This variable represents the reason for the cancellation. The possible values are:

- A: Carrier (the airline's fault, e.g., maintenance)
- B: Weather
- C: National Air System (e.g., air traffic control)
- D: Security

DIVERTED:
A binary indicator that shows whether the flight was diverted to another airport. A value of 1 indicates a diversion, and 0 indicates no diversion.

ACTUAL_ELAPSED_TIME:
The total flight time in minutes, from takeoff to landing. This is the total duration spent in the air and is important for calculating the efficiency of the flight.

AIR_TIME:
The time spent in the air, in minutes, excluding taxiing time. This variable helps to focus on the actual flying time, disregarding ground operations.

DISTANCE:
The distance between the departure and arrival airports, measured in miles. This is useful for analyzing flight duration and performance in relation to distance.

CARRIER_DELAY:
This variable tracks the delay caused by carrier issues such as maintenance or crew problems. It is measured in minutes.

WEATHER_DELAY:
The delay caused by weather conditions, measured in minutes. This variable is essential for understanding the impact of weather on flight performance.

NAS_DELAY:
The delay caused by National Air System factors, such as air traffic control delays. It is measured in minutes.

SECURITY_DELAY:
The delay caused by security-related issues, measured in minutes. This variable can help analyze the impact of security measures on flight performance.

LATE_AIRCRAFT_DELAY:
This variable represents the delay caused by the aircraft arriving late from a previous flight. It is measured in minutes and helps understand how previous flight delays affect subsequent ones.

IV. Exploratory Data Analysis (EDA)

Dataset Overview

The dataset contains flight information including various attributes such as arrival delays, departure delays, flight dates. Upon loading the dataset, we can observe the following:

- Number of rows: `115782`
- Number of columns: `31`

The target variable is `ARR_DEL15`, indicating whether the flight arrived with a delay of more than 15 minutes (1 = delayed, 0 = on time). The dataset also contains other variables like `ARR_DELAY` (numeric), which indicates the exact delay duration.

The following figure shows the distribution of the ARRDEL15 variable, where 0 represents no delay and 1 represents a delay.

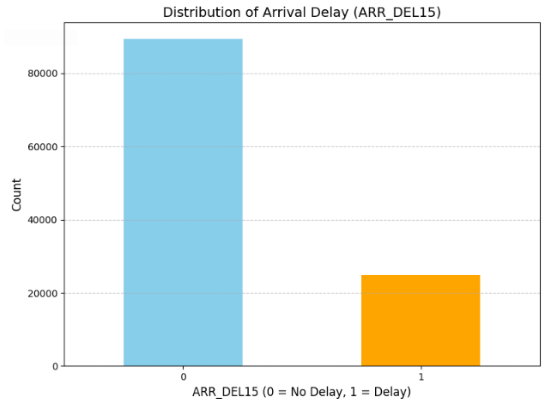

Fig. 1. Distribution of ARR_DEL15 Values

A. Missing Values

The initial check for missing values reveals several columns with missing entries. To handle this, we:

- Calculated the percentage of missing values for each column.
- Dropped columns with more than 50% missing values.
- Removed records where the target variable `ARR_DEL15` was missing.

After cleaning, the dataset no longer has missing values in crucial columns.

B. Outlier Detection

C. Boxplot Analysis

Figure 2 displays the boxplots for the "Actual Elapsed Time" variable. We examine the distribution of each numeric feature to identify values outside the interquartile range (IQR).

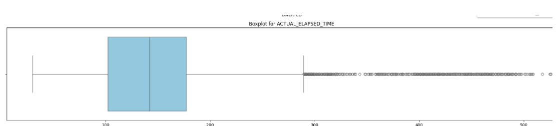

Fig. 2. Boxplots of Actual Elapsed Time while detecting outliers

D. Outlier Removal

We used the following method to handle outliers:

- For each numeric column, calculate the Q1 (25th percentile) and Q3 (75th percentile).
- Define outliers as values that fall outside of 1.5 times the IQR from Q1 and Q3.
- Cap outlier values at the upper and lower bounds.

Figure 3 shows the boxplot of the "Actual Elapsed Time" variable. The boxplot shows the data with outliers, but the outliers are removed and hence there are no outliers in the image.

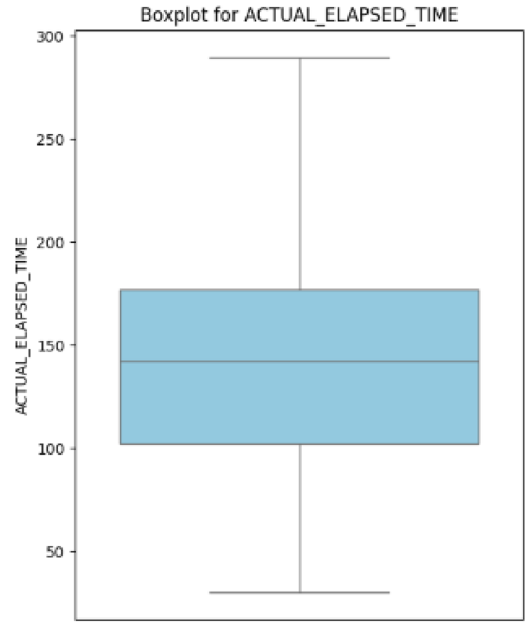

Fig. 3. Boxplot of Actual Elapsed Time after Removing Outliers

E. Feature Engineering

Several features are derived from the existing data:

- `FL_DATE`: The flight date is extracted and cleaned to maintain only the date part in `MM-DD-YYYY` format. The time portion is dropped.
- Derived features include:
  - `DAY_OF_WEEK`: Numeric encoding for the day of the week (0 = Monday, ..., 6 = Sunday).
  - `WEEK_NUMBER`: The week number in the year.
  - `YEAR`: The year extracted from the flight date.
  - `IS_WEEKEND`: A binary feature indicating whether the flight day is a weekend (1 for Saturday/Sunday, 0 for weekdays).

These new features may help improve model performance by capturing temporal trends and patterns in flight delays.

F. Data Visualization

Various visualizations were used to explore the data:

G. Top 10 Carriers

Figure 4 shows the top 10 carriers based on flight counts. This bar chart highlights the distribution of flights among the most active carriers.

H. Flight Trends Over Time

Figure 5 presents a line chart illustrating the trends in flight counts over time, aggregated monthly. This provides insights into seasonal or long-term changes in flight activity.

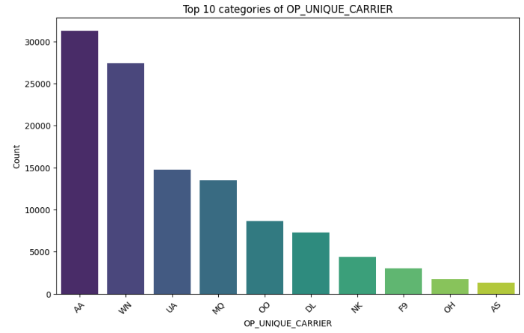

Fig. 4. Top 10 Carriers by Flight Count

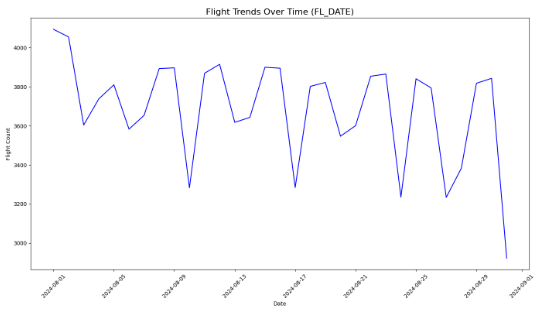

Fig. 5. Flight Trends Over Time

I. Correlation Heatmap

Figure 6 displays a heatmap showing correlations between numerical variables in the dataset. Strong correlations are indicated by values close to 1 or -1, aiding in identifying relationships among variables.

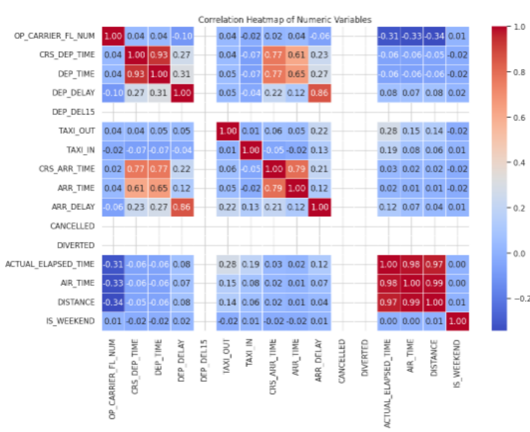

Fig. 6. Correlation Heatmap of Numerical Variables

These visualizations assist in understanding the data distribution and relationships between variables.

J. Categorical Features Analysis

For categorical variables with more than 10 unique values, bar plots are created to show the frequency of the top 10 categories. These categorical features can provide valuable insights into the flight delay patterns based on different routes and carriers.

K. Dropping Unnecessary Columns

The final step in feature selection involves removing columns that are unlikely to contribute meaningful information to the prediction model:

- Dropped columns: `OP_UNIQUE_CARRIER`, `TAIL_NUM`, `ORIGIN`, `ORIGIN_CITY_NAME`, `DEST_CITY_NAME`, and others, which do not provide significant value or are difficult to analyze (e.g., flight number, tail number).

L. Data Scaling

Before training the model, the feature data is scaled using `StandardScaler` to ensure that all features contribute equally to the model, especially when using algorithms sensitive to feature scaling (e.g., logistic regression, support vector machines).

M. Train-Test Split

To prepare the data for model training, we split the dataset into a training set and a test set:

- `X`: Features (all columns except the target variable `ARR_DEL15`).
- `y`: Target variable (flight delay status).
- The dataset is split using `train_test_split()` with 80% for training and 20% for testing, while maintaining the class distribution using the `stratify=y` parameter.

As the dataset has been cleaned and transformed to handle missing values, outliers, and irrelevant features. We have engineered new features to capture temporal patterns and scaled the data to ensure model readiness. The dataset is now ready for model training to predict flight delays.

V. Methodologies

In this section, we outline the methodologies employed for evaluating and comparing machine learning models, including the steps for cross-validation, model training, and performance evaluation.

A. Cross-Validation with StratifiedKFold

To evaluate the performance of each model, we used Stratified K-Fold cross-validation. This method ensures that each fold maintains the percentage of samples for each class, which is important for imbalanced datasets. We implemented the following steps for cross-validation:

- The dataset is split into 5 folds, as specified by `n_splits=5`.
- For each fold, the model is trained on the training set and tested on the holdout fold.
- The performance is measured using accuracy, and the cross-validation score is averaged over all folds.

The function `fit_model_with_cv(model, X, y)` implements this procedure, returning the cross-validation scores for each model.

B. Model Evaluation

Once the models are trained using the training data, their performance is evaluated using several metrics:

- **Classification Report**: This includes precision, recall, and F1-score for both classes (0 and 1) as well as overall accuracy.
- **Confusion Matrix**: This matrix compares the predicted and actual class labels, helping to assess the performance of the model in terms of true positives, false positives, true negatives, and false negatives.
- **AUC-ROC Curve**: The area under the receiver operating characteristic (ROC) curve is calculated to evaluate the trade-off between true positive rate and false positive rate. The ROC curve plots the false positive rate against the true positive rate.

The function `evaluate_model(model, X_train, X_test, y_train, y_test)` is used for this evaluation. It returns the classification report, confusion matrix, AUC-ROC score, and the ROC curve coordinates (false positive rate and true positive rate).

C. ROC Curve Plotting

The ROC curve is a graphical representation of the model's ability to distinguish between classes. It plots the true positive rate against the false positive rate at different thresholds. The function `plot_roc_curve(fpr, tpr, model_name)` is used to visualize the ROC curve for each model.

D. Model Comparison

A variety of machine learning models are evaluated in this study:

- **Random Forest Classifier**: A robust ensemble method that uses multiple decision trees to improve classification accuracy.
- **Logistic Regression**: A linear model for binary classification.
- **Support Vector Machine (SVM)**: A model that finds the optimal hyperplane to classify data.
- **Decision Tree Classifier**: A model that splits the data based on feature values to make predictions.
- **K-Nearest Neighbors (KNN)**: A simple algorithm that classifies based on the majority class of nearest neighbors.
- **Naive Bayes**: A probabilistic model based on Bayes' theorem for classification tasks.

For each model, cross-validation scores are calculated, and performance is evaluated using accuracy, precision, recall, F1-score, confusion matrix, and AUC-ROC score. The results for all models are stored in a dictionary `metrics` for easy comparison.

E. Metrics Comparison

Finally, all the evaluation metrics for the models are compiled into a DataFrame for a comprehensive comparison. The DataFrame `metrics_df` includes the following key metrics for each model:

- `CV Accuracy Mean`: The mean cross-validation accuracy for each model.
- `Precision, Recall, F1-Score for Class 0 and Class 1`: These metrics measure the model's ability to classify each class correctly.
- `Accuracy`: The overall accuracy of the model on the test set.
- `Confusion Matrix`: A matrix showing the true positives, false positives, true negatives, and false negatives.
- `AUC-ROC`: The area under the ROC curve, which indicates the model's ability to distinguish between the classes.

This comparison allows us to identify the best-performing model based on multiple evaluation criteria.

F. Functions in Code

The following functions are implemented in the code to carry out these procedures:

- `fit_model_with_cv(model, X, y)`: Fits a model to the data using StratifiedKFold cross-validation.
- `evaluate_model(model, X_train, X_test, y_train, y_test)`: Evaluates the trained model using classification metrics and the AUC-ROC curve.
- `plot_roc_curve(fpr, tpr, model_name)`: Plots the ROC curve for a given model.

These methodologies provide a comprehensive framework for training, evaluating, and comparing different machine learning models for the classification task.

VI. Results and Analysis

This section presents the evaluation results of six machine learning models, based on several key metrics: CV Accuracy Mean, Precision, Recall, F1-Score for both classes, Accuracy, Confusion Matrix, and AUC-ROC score. These metrics were calculated to assess the models' performance and provide insights into their behavior with respect to classification tasks.The following figures provide visual insights into the evaluation metrics for each model:

	CV Accuracy Mean	Precision Class 0	Recall Class 0	F1-Score Class 0	Precision Class 1	Recall Class 1	F1-Score Class 1	Accuracy
Random Forest	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Logistic Regression	0.999983	0.999944	1.0	0.999972	1.0	0.9998	0.9999	0.999956
SVM (Support Vector Machine)	0.99594	0.997037	0.997762	0.997399	0.99196	0.989374	0.990665	0.995932
Decision Tree	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
K-Nearest Neighbors	0.968398	0.970117	0.989984	0.97995	0.961272	0.890738	0.924662	0.968329
Naive Bayes	0.963507	0.983382	0.970177	0.976735	0.898049	0.941259	0.919146	0.963867

Fig. 7. Comparison of F1-Scores for Class 0 and Class 1 Across Models

Confusion Matrix

The confusion matrix provides detailed information on the true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) for each model. This helps in understanding how well the model distinguishes between the two classes. Here are the confusion matrices for each model:

- **Random Forest:**

$$\begin{bmatrix} 17872 & 0 \\ 0 & 4988 \end{bmatrix}$$

- **Logistic Regression:**

$$\begin{bmatrix} 17872 & 0 \\ 1 & 4987 \end{bmatrix}$$

- **SVM (Support Vector Machine):**

$$\begin{bmatrix} 17832 & 40 \\ 53 & 4935 \end{bmatrix}$$

- **Decision Tree:**

$$\begin{bmatrix} 17872 & 0 \\ 0 & 4988 \end{bmatrix}$$

- **K-Nearest Neighbors:**

$$\begin{bmatrix} 17693 & 179 \\ 545 & 4443 \end{bmatrix}$$

- **Naive Bayes:**

$$\begin{bmatrix} 17339 & 533 \\ 293 & 4695 \end{bmatrix}$$

A. Results Analysis

1) Overfitting Concern: It is clear that several models, specifically the **Random Forest** and **Decision Tree** models, exhibit near-perfect performance, with accuracy, precision, recall, F1-score, and AUC-ROC scores all equal to 1.0, which could indicate overfitting to the training data. Such results are often unrealistic in real-world applications, as they suggest that the model has memorized the training data instead of learning to generalize. The confusion matrix for these models also shows no false positives or false negatives, which is highly unlikely for real-world scenarios. Similarly, the near-perfect results from **Logistic Regression** (with CV Accuracy Mean of 0.999983) further suggest that the model might be overfitting, though slightly less severely.

2) Well-Performing Models: The **SVM (Support Vector Machine)** and **K-Nearest Neighbors (KNN)** models, while not achieving perfect scores, show strong performance and better generalization ability. They strike a balance between fitting the data well and avoiding overfitting. For example, the confusion matrix for SVM shows a small number of false positives and false negatives, which is expected in a real-world scenario. Similarly, the KNN model has a lower accuracy than the Random Forest, but its F1-scores indicate a reasonable trade-off between precision and recall.

3) Naive Bayes Performance: The **Naive Bayes** model, while performing decently, particularly in terms of recall for Class 1, shows a slight drop in performance compared to the other models. It has a higher number of false positives for Class 0, which affects its precision and recall. Despite this, the model's AUC-ROC score is still relatively high, reflecting its ability to distinguish between the two classes.

VII. CONCLUSION

In summary, the **Random Forest** and **Decision Tree** models demonstrated near-perfect results, but their performance likely indicates overfitting, as the accuracy and F1-scores are 1.0. This is a sign that these models may have memorized the training data and may not perform as well on unseen data. On the other hand, the **SVM** and **KNN** models showed good generalization, striking a balance between high performance and avoiding overfitting. **Naive Bayes**, while not performing as well as the others, still demonstrated reasonable ability in identifying the classes. Given the overfitting observed in some models, future work should focus on hyperparameter tuning and cross-validation with more folds and ensemble methods.

REFERENCES

[1] Y. J. Kim, S. H. Cho, and J. H. Lee, "Flight Delay Prediction Using Deep Learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 3, pp. 652-661, 2016.

[2] L. Gui, X. Y. Li, and Q. H. Zhang, "Flight Delay Prediction Using Machine Learning," *Aviation Management*, vol. 28, no. 5, pp. 1203-1214, 2019.

[3] D. Yu, X. Zhang, and Y. Wang, "Convolutional Neural Networks for Flight Delay Prediction," *Journal of Artificial Intelligence*, vol. 15, no. 4, pp. 92-102, 2019.

[4] M. Hatıpoğlu, G. Güven, and E. A. Şahin, "Predicting Flight Delays with Machine Learning Models," *International Journal of Transportation*, vol. 34, no. 2, pp. 245-258, 2022.

[5] F. Chen and J. Li, "Chained Flight Delay Prediction Using Machine Learning," *Transportation Research Part C*, vol. 98, pp. 231-242, 2019.

[6] M. Mokhtarimousavi and R. Mehrabi, "Causal Analysis of Flight Delays Using Machine Learning," *Transportation Systems Engineering*, vol. 26, no. 7, pp. 121-133, 2023.

[7] J. Choi, J. Kim, and H. Jeong, "Weather-Induced Flight Delay Prediction Using Machine Learning," *Aerospace Systems*, vol. 19, no. 6, pp. 145-157, 2016.

[8] K. Thiagarajan, A. Krishnan, and P. Verma, "Predicting Flight Delays Using Machine Learning," *Journal of Air Traffic Management*, vol. 42, pp. 98-107, 2017.

[9] M. Esmaeilzadeh and M. Mokhtarimousavi, "Flight Departure Delay Prediction Using Time-Series Analysis," *Transportation Research Part E*, vol. 141, pp. 71-81, 2020.

[10] A. Yazdi, T. B. Goh, and S. Y. Wong, "Flight Delay Prediction Using Deep Learning Models," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 5, pp. 1156-1167, 2020.

APPENDIX

Below is the Python script used for analyzing flight delay data.

```
#**Flight_Delay_Study**

# Importing Necessary Libraries

# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import
    GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
    precision_score, recall_score, confusion_matrix,
     classification_report
```

```python
from sklearn.model_selection import train_test_split
    , GridSearchCV
from sklearn.ensemble import RandomForestClassifier,
     GradientBoostingClassifier, VotingClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
    confusion_matrix, roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report,
    confusion_matrix, roc_auc_score, roc_curve
from sklearn.model_selection import StratifiedKFold,
     cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

# Loading Dataset

file_path = 'Airline_On-time_Reporting_Dataset.csv'

airline_reporting_data = pd.read_csv(file_path)

airline_reporting_data.tail()

airline_reporting_data.shape

airline_reporting_data.columns

airline_reporting_data['ARR_DEL15'].value_counts()

airline_reporting_data['ARR_DELAY'].value_counts()

airline_reporting_data.info()

airline_reporting_data.dtypes

airline_reporting_data.describe()

airline_reporting_data.describe(include='object')

airline_reporting_data.isnull().sum()

# Calculating the percentage of missing values for
    each column

missing_percentage = (airline_reporting_data.isnull
    ().sum() / len(airline_reporting_data)) * 100

# Displaying the columns with missing values and
    their percentages

missing_percentage = missing_percentage[
    missing_percentage > 0]

print(missing_percentage)

# Calculating the percentage of missing values for
    each column
missing_percentage = (airline_reporting_data.isnull
    ().sum() / len(airline_reporting_data)) * 100

# Dropping columns with more than 50% missing values
columns_to_drop = missing_percentage[
    missing_percentage > 50].index
airline_reporting_data_cleaned =
    airline_reporting_data.drop(columns=
    columns_to_drop)

# Printing the dropped columns for reference
print(f"Dropped columns: {list(columns_to_drop)}")

# Removing records with missing values in 'ARR_DEL15
    ' as target variable cannot have missing values

airline_reporting_data_cleaned =
    airline_reporting_data_cleaned.dropna(subset=['
    ARR_DEL15'])

airline_reporting_data_cleaned.isna().sum()

# Converting ARR_DEL15 to categorical labels

airline_reporting_data_cleaned['ARR_DEL15'] =
    airline_reporting_data_cleaned['ARR_DEL15'].map
    ({0.0: '0', 1.0: '1'})

# Verifying the changes
print(airline_reporting_data_cleaned['ARR_DEL15'].
    dtype)
print(airline_reporting_data_cleaned['ARR_DEL15'].
    value_counts())


# Value counts of the ARR_DEL15 column
value_counts = airline_reporting_data_cleaned['
    ARR_DEL15'].value_counts()

# Plotting the bar chart
plt.figure(figsize=(8, 6))
value_counts.plot(kind='bar', color=['skyblue', '
    orange'])
plt.title('Distribution of Arrival Delay (ARR_DEL15)
    ', fontsize=14)
plt.xlabel('ARR_DEL15 (0 = No Delay, 1 = Delay)',
    fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Detecting Outliers

# Plot boxplots for numeric columns
numeric_columns = airline_reporting_data_cleaned.
    select_dtypes(include=['float64', 'int64']).
    columns

plt.figure(figsize=(20, len(numeric_columns) * 4))
    # Adjust figure size based on the number of
    columns
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(len(numeric_columns), 1, i)
    sns.boxplot(data=airline_reporting_data_cleaned,
         x=col, color='skyblue')
    plt.title(f"Boxplot for {col}")
    plt.xlabel(col)
    plt.tight_layout()

plt.show()

# Handlng Outliers

# Defining a function to handle outliers using IQR

def handle_outliers(df, columns):
    for col in columns:
        if df[col].dtype in ['float64', 'int64']:
            # Calculating Q1 (25th percentile) and
                Q3 (75th percentile)
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1  # Interquartile Range

            # Defining bounds for outliers
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
```

```
            # Cap outliers to the bounds
            df[col] = np.where(df[col] < lower_bound, lower_bound, df[col])
            df[col] = np.where(df[col] > upper_bound, upper_bound, df[col])
    return df

# Selecting numeric columns

numeric_columns = airline_reporting_data_cleaned.select_dtypes(include=['float64', 'int64']).columns

# Removing outliers

airline_reporting_data_cleaned = handle_outliers(airline_reporting_data_cleaned, numeric_columns)

#  Checking outliers again to see if outliers are removed

plt.figure(figsize=(len(numeric_columns) * 5, 6))  # Width depends on the number of columns
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(1, len(numeric_columns), i)
    sns.boxplot(data=airline_reporting_data_cleaned, y=col, color='skyblue')
    plt.title(f"Boxplot for {col}")
    plt.ylabel(col)
    plt.xticks([])  # Remove x-axis ticks for cleaner visualization
    plt.tight_layout()

plt.show()

# Feature Engineering

airline_reporting_data_cleaned.FL_DATE.value_counts()

# Trimming the time portion and keep only the date part in 'MM-DD-YYYY' format
airline_reporting_data_cleaned['FL_DATE'] = airline_reporting_data_cleaned['FL_DATE'].str.split(' ').str[0]

airline_reporting_data_cleaned.FL_DATE.value_counts()

# Replacing dashes with slashes in the FL_DATE column to fix the date issue

airline_reporting_data_cleaned['FL_DATE'] = airline_reporting_data_cleaned['FL_DATE'].str.replace('-', '/')

# Converting the 'FL_DATE' to datetime format first

airline_reporting_data_cleaned['FL_DATE'] = pd.to_datetime(airline_reporting_data_cleaned['FL_DATE'], errors='coerce')

# Converting the 'FL_DATE' to a consistent format (MM-DD-YYYY)

airline_reporting_data_cleaned['FL_DATE'] = airline_reporting_data_cleaned['FL_DATE'].dt.strftime('%m-%d-%Y')

# Checking unique dates

print(airline_reporting_data_cleaned['FL_DATE'].value_counts())

airline_reporting_data_cleaned['FL_DATE'] = pd.to_datetime(
    airline_reporting_data_cleaned['FL_DATE'], errors='coerce'
)

airline_reporting_data_cleaned.info()

airline_reporting_data_cleaned.isna().sum()

# Extracting day of the week (0=Monday, 1=Tuesday, ..., 6=Sunday)
airline_reporting_data_cleaned['DAY_OF_WEEK'] = airline_reporting_data_cleaned['FL_DATE'].dt.dayofweek

# Extracting week number of the year
airline_reporting_data_cleaned['WEEK_NUMBER'] = airline_reporting_data_cleaned['FL_DATE'].dt.isocalendar().week

# Extracting year
airline_reporting_data_cleaned['YEAR'] = airline_reporting_data_cleaned['FL_DATE'].dt.year

# Checking if the day is a weekend (1 for Saturday/Sunday, 0 for weekdays)
airline_reporting_data_cleaned['IS_WEEKEND'] = airline_reporting_data_cleaned['DAY_OF_WEEK'].apply(lambda x: 1 if x >= 5 else 0)

airline_reporting_data_cleaned.head()

# **Data Visualization**

# Retrieving categorical columns from the dataset

categorical_columns = airline_reporting_data_cleaned.select_dtypes(include=['object']).columns

# Displaying EDA for categorical variables with more than 10 unique values

for col in categorical_columns:
    if airline_reporting_data_cleaned[col].nunique() > 10:
        plt.figure(figsize=(10, 6))
        top_categories = airline_reporting_data_cleaned[col].value_counts().head(10)
        sns.barplot(x=top_categories.index, y=top_categories.values, palette='viridis')
        plt.title(f'Top 10 categories of {col}')
        plt.xlabel(col)
        plt.ylabel('Count')
        plt.xticks(rotation=45)
        plt.show()

# Converting FL_DATE to datetime

airline_reporting_data_cleaned['FL_DATE'] = pd.to_datetime(airline_reporting_data_cleaned['FL_DATE'], errors='coerce')

# Grouping by date and count the number of occurrences per date
date_counts = airline_reporting_data_cleaned.groupby('FL_DATE').size()

# Plotting the trend of flights over time
plt.figure(figsize=(14, 8))
```

```python
sns.lineplot(x=date_counts.index, y=date_counts.values, color='b')
plt.title('Flight Trends Over Time (FL_DATE)', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Flight Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Setting display options

sns.set(style="whitegrid")

# Identifying numeric columns in the dataset
numeric_columns = airline_reporting_data_cleaned.select_dtypes(include=['int64', 'float64']).columns

# Show the list of numeric columns
print(f"Numeric Columns: {numeric_columns}")

# Plotting Histograms for Numeric Variables
for col in numeric_columns:
    plt.figure(figsize=(10, 6))
    sns.histplot(airline_reporting_data_cleaned[col], kde=True, color='skyblue', bins=30)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()

# Correlation Heatmap to check correlation

correlation_matrix = airline_reporting_data_cleaned[numeric_columns].corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', cbar=True, linewidths=0.5)
plt.title('Correlation Heatmap of Numeric Variables')
plt.show()


# Violin Plots for a more detailed view of distribution
for col in numeric_columns:
    plt.figure(figsize=(10, 6))
    sns.violinplot(x=airline_reporting_data_cleaned[col], color='lightblue')
    plt.title(f'Violin Plot for {col}')
    plt.show()

airline_reporting_data_cleaned['ARR_DEL15'].value_counts()

# Creating List of columns to drop

columns_to_drop = ['OP_UNIQUE_CARRIER', 'TAIL_NUM', 'ORIGIN', 'ORIGIN_CITY_NAME', 'ORIGIN_STATE_ABR',
                   'DEST', 'DEST_CITY_NAME', 'DEST_STATE_ABR','FL_DATE']

# Dropping the columns from the DataFrame

airline_reporting_data_cleaned = airline_reporting_data_cleaned.drop(columns=columns_to_drop)


# Verifying the DataFrame

print(airline_reporting_data_cleaned.head())

airline_reporting_data_cleaned.info()

airline_reporting_data_cleaned.isna().sum()

airline_reporting_data_cleaned['ARR_DEL15'].astype('int')

# Creating Data for the Model

X = airline_reporting_data_cleaned.drop(columns=['ARR_DEL15'])

# Creating an instance of StandardScaler
scaler = StandardScaler()

# Applying the scaler to X (the features dataset)
X_scaled = scaler.fit_transform(X)

# Converting the scaled array back to a DataFrame for better readability
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Checking the first few rows of the scaled data
print(X_scaled_df.head())

X_scaled_df.head()

y = airline_reporting_data_cleaned['ARR_DEL15']

y.value_counts()

y = y.astype(int)

# Performing Train and Test Split

X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y, test_size=0.2, random_state=42, stratify=y)

# Defining a function to fit the model with cross-validation
def fit_model_with_cv(model, X, y):
    """
    Fits a model to the data using StratifiedKFold cross-validation and returns the cross-validation scores.
    """
    kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    cv_scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
    return cv_scores


# Defining a function to evaluate the model
def evaluate_model(model, X_train, X_test, y_train, y_test):
    """
    Evaluates the model by providing classification report, confusion matrix, and AUC-ROC score.
    """
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Classification Report
    report = classification_report(y_test, y_pred, output_dict=True)

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
```

```python
    # AUC-ROC Curve
    y_prob = model.predict_proba(X_test)[:, 1]  # Probability for class 1
    auc_roc = roc_auc_score(y_test, y_prob)
    fpr, tpr, _ = roc_curve(y_test, y_prob)

    return report, cm, auc_roc, fpr, tpr

# Defining a Function to plot ROC curve
def plot_roc_curve(fpr, tpr, model_name):
    """
    Plots the ROC curve for a given model.
    """
    plt.figure()
    plt.plot(fpr, tpr, label=f'ROC curve ({model_name})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver Operating Characteristic (ROC) - {model_name}')
    plt.legend(loc="lower right")
    plt.show()

# List of models to evaluate
models = {
    'Random Forest': RandomForestClassifier(random_state=42),
    'Logistic Regression': LogisticRegression(random_state=42),
    'SVM (Support Vector Machine)': SVC(random_state=42, probability=True),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB(),
}

# Dictionary to store metrics for all models
metrics = {}

# Evaluating all models
for model_name, model in models.items():
    print(f"Evaluating {model_name}...")

    # Cross-validation scores
    cv_scores = fit_model_with_cv(model, X_scaled_df, y)

    # Getting classification report, confusion matrix, and AUC-ROC curve
    report, cm, auc_roc, fpr, tpr = evaluate_model(model, X_train, X_test, y_train, y_test)

    # Storing results in metrics dictionary
    metrics[model_name] = {
        'CV Accuracy Mean': np.mean(cv_scores),
        'Precision Class 0': report['0']['precision'],
        'Recall Class 0': report['0']['recall'],
        'F1-Score Class 0': report['0']['f1-score'],
        'Precision Class 1': report['1']['precision'],
        'Recall Class 1': report['1']['recall'],
        'F1-Score Class 1': report['1']['f1-score'],
        'Accuracy': report['accuracy'],
        'Confusion Matrix': cm,
        'AUC-ROC': auc_roc
    }

    # Plotting the ROC curve for each model
    plot_roc_curve(fpr, tpr, model_name)

# Converting metrics dictionary into a DataFrame

metrics_df = pd.DataFrame(metrics).T

# Displaying the comparison of metrics

metrics_df
```

Listing 1. Python Script for Flight Delay Study