# Syracuse Flight Arrival Delay Prediction Using Machine Learning Models

Akshaya Ganesan
Syracuse University
Syracuse, New York
aganes02@syr.edu

Sanjay Santhanam
Syracuse University
Syracuse, New York
ssanthan@syr.edu

Uday Vempalli
Syracuse University
Syracuse, New York
uvempall@syr.edu

Nimeesh Nilesh Bagwe
Syracuse University
Syracuse, New York
nibagwe@syr.edu

*Abstract* - This study addresses the cascading effect of arrival delays, where an earlier flight's status can significantly impact subsequent flights on the same route and day. We proposed a robust machine learning framework using a dataset that merges historical flight data with detailed weather information, including specifics like flight timings, durations, and weather conditions such as temperature and precipitation for both arrival and destination airports. Using Supervised ML Models like Random Forest, Gradient Boosting and XGBoost Classifiers, known for handling complex data interactions and mitigating overfitting. This ensures the models can be applied effectively to unseen datasets and real-world scenarios. The models were trained on a segmented dataset to optimize their accuracy in predicting flight arrival statuses (Early, On-time, or Late).

*Keywords - Flight arrival delay prediction, Machine learning, XGBoost Algorithm, Prediction Accuracy, Flight Status, Model Training and Testing*

## I. Introduction

Air travel is a critical component of global connectivity, allowing millions of travelers to move every day. However, flight delays remain a common problem, producing considerable disruptions for passengers, airlines, and the general efficiency of the aviation sector. Adverse weather conditions, air traffic control limits, and operational issues all contribute to these delays, which pose continuous challenges despite technological developments in the field.

This project addresses the crucial requirement for improved flight delay prediction mechanisms by concentrating on aircraft arriving at Syracuse Hancock International Airport (SYR) from major hubs such as Chicago O'Hare (ORD), New York JFK (JFK), and Orlando (MCO).

We intend to create a strong prediction model that not only anticipates the arrival statuses of flights into SYR, but also investigates the interrelated impacts of sequential flights from the same origin. Understanding these dynamics can help airlines improve resource management, optimize schedules, and limit the cascade impacts of delays, resulting in increased customer satisfaction and operational dependability.

The approach employed in this research leverages advanced machine learning techniques such as gradient boosting, random forest and xgboost classifiers to assess integrated datasets comprising both historical flight data and real-time meteorological variables. This approach allows for a more detailed knowledge of the features which contribute to flight delays, as well as the forecast of flight statuses (Early, On-time, or Late) for both incoming and following aircraft. Detailed data preparation, model training and testing, and model performance evaluation are all key components of our implementation and will be completely discussed in the methodology section. Our results seek to improve airlines' strategic planning capacities and give passengers with more trustworthy and timely information, resulting in a more efficient and less stressful air travel experience.

## II. Data Collection

For the purpose of this project on predicting flight delays, we utilized two primary sources for collecting data. The flight data was obtained from the Bureau of Transportation Statistics (BTS) [1]. This repository provided comprehensive information on flight operations across the United States, including details such as scheduled and actual departure and arrival times, flight numbers, and airline information. Weather data was sourced from Open Meteo's historical weather API [2] and forecast weather API [3] which provided hourly updates on weather conditions at the airports.

The dataset encompasses records from the years 2017 to 2023. Notably, data from the year 2020 was deliberately omitted due to the anomalous impact of the COVID-19 pandemic on air travel, which led to unprecedented changes in flight schedules and frequencies, thus distorting typical operational patterns.

### *A. Weather Data Retrieval*

To enhance the accuracy of our flight delay prediction model, comprehensive hourly weather data was obtained for each airport involved in the study (Syracuse - SYR, New York - JFK, Orlando - MCO, and Chicago - ORD). This level of temporal resolution allows us to account for sudden changes or developments in weather phenomena, such as thunderstorms, fog, or high winds, which can significantly affect flight operations. This data was sourced from the Open Meteo's historical and forecast weather API. The weather information includes hourly data on temperature, precipitation, snowfall, cloud cover, and wind conditions.

The process of retrieving and storing weather data was automated through a Python script utilizing the *openmeteo requests* module, which was tailored to fetch historical and forecast data for specified locations. Parameters such as latitude, longitude, and time range (start_date and

end_date) are specified along with desired data metrics like temperature, wind speed, rain, snow, cloud cover, wind gust.

```
def fetch_weather_data(latitude, longitude, timezone, excel_file_name):
    # Setup the Open-Meteo API client with cache and retry on error
    cache_session = requests_cache.CachedSession('.cache', expire_after=-1)
    retry_session = retry(cache_session, retries=5, backoff_factor=0.2)
    openmeteo = openmeteo_requests.Client(session=retry_session)

    # Define API parameters
    url = "https://archive-api.open-meteo.com/v1/archive"
    params = {
        "latitude": latitude,
        "longitude": longitude,
        "start_date": "2017-01-01",
        "end_date": "2024-01-31",
        "hourly": [
            "temperature_2m", "rain", "snowfall",
            "snow_depth", "cloud_cover", "wind_speed_10m",
            "wind_gusts_10m"
        ],
        "temperature_unit": "fahrenheit",
        "wind_speed_unit": "mph",
        "precipitation_unit": "inch",
        "timezone": timezone
    }
```

*Figure 1: Weather API Parameters for Historical Data*

```
def fetch_forecast_weather_data(latitude, longitude, excel_file_name):
    # Setup the Open-Meteo API client with cache and retry on error
    cache_session = requests_cache.CachedSession('.cache', expire_after=-1)
    retry_session = retry(cache_session, retries=5, backoff_factor=0.2)
    openmeteo = openmeteo_requests.Client(session=retry_session)

    # Define API parameters
    url = "https://api.open-meteo.com/v1/forecast"
    params = {
        "latitude": latitude,
        "longitude": longitude,
        "hourly": [
            "temperature_2m", "rain", "snowfall",
            "snow_depth", "cloud_cover", "wind_speed_10m",
            "wind_gusts_10m"
        ],
        "temperature_unit": "fahrenheit",
        "wind_speed_unit": "mph",
        "precipitation_unit": "inch"
    }
```

*Figure 2: Weather API Parameters for Forecast Data*

Data from the historical and forecast API responses are processed into hourly intervals. Each weather variable (e.g., temperature, rain, snow, wind and cloud cover) is extracted and structured into a pandas Data Frame. The Data Frame organizes data with timestamps as rows and weather metrics as columns, providing a clear, hourly overview of weather conditions.

### B. *Flight Data*

The foundational dataset for this research was procured from the Bureau of Transportation Statistics (BTS), which maintains extensive records of flight operations within the United States. This data is publicly accessible through their On-Time Performance database, which updates monthly and provides detailed metrics on flight punctuality and delays for domestic flights. The dataset was filtered to include only the required origin and destination airports.

To analyze flight data effectively, the dataset includes important information such as the "Carrier Code," which tells us the airline's name, and the "Date (MM/DD/YYYY)," showing the day the flight took place. Each flight is uniquely marked by a "Flight Number." The "Origin Airport" column notes where the flight started. The "Scheduled Arrival Time" and "Actual Arrival Time" let us see when the flight was supposed to arrive versus when it actually did. The "Arrival Delay (Minutes)" measures how late the flight was. This set of columns helps us understand how timely and efficient the flights are.

To enhance the comprehensiveness of the flight data analysis, it was imperative to merge the arrival and departure information for each flight route. This integration was achieved using the XLOOKUP function in Microsoft Excel, which facilitates the seamless matching of corresponding data points between two distinct sets of data—arrival and departure logs in this case.

## III. Data Merging

To improve how we analyze flight data with weather conditions, we created a Python script that automatically combines flight and weather information (historical and forecast) into one dataset.

### A. *Combining Date and Time Columns into Datetime Objects:*

This process starts by changing the date and time details in our data into a single format that includes both the date and the specific time. This makes everything consistent and easier to work with, especially when we round these times to match with the hourly weather records.

### B. *Merging with Weather Data for Destination Airport (SYR):*

The flight data is merged with weather data for the destination airport (SYR) based on the rounded scheduled arrival time and the airport code. This ensures that the weather data corresponds to the time and location of the flights' arrival.

### C. *Merging with Weather Data for Origin Airports (JFK, ORD, MCO):*

Similar to the previous step, the flight data is merged with weather data for each origin airport (JFK, ORD, MCO) based on the rounded scheduled departure time and airport code. This step incorporates weather information relevant to the flights' departure locations.

### D. *Selecting and Renaming Columns:*

Relevant weather columns (e.g., temperature, rain, snow etc.,) are selected for merging and renamed to distinguish them as destination airport data and origin airport data. This step helps organize the merged dataset and avoids column name conflicts. The merged datasets for each airport (SYR, JFK, ORD, MCO) are concatenated along the rows to create a single comprehensive dataset containing flight details and corresponding weather information for both origin and destination airports.

```
Carrier Code
Date (MM/DD/YYYY)
Flight Number
Tail Number
Origin Airport
Destination Airport
Scheduled departure time
Actual departure time
Scheduled Arrival Time
Actual Arrival Time
Scheduled elapsed time (Minutes)
Actual elapsed time (Minutes)
Departure delay (Minutes)
Arrival Delay (Minutes)
Destination_temperature_2m
Destination_rain
Destination_snowfall
Destination_snow_depth
Destination_cloud_cover
Destination_wind_speed_10m
Destination_wind_gusts_10m
Origin_temperature_2m
Origin_rain
Origin_snowfall
Origin_snow_depth
Origin_cloud_cover
Origin_wind_speed_10m
Origin_wind_gusts_10m
```

*Figure 3: Features in Weather Integrated Flight Dataset*

## IV. EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is vital for our project as it provides a deep understanding of dataset structure and characteristics, aiding informed decision-making. It helps identify patterns, outliers, and trends crucial for feature selection and model development. EDA also ensures data quality by detecting missing values and inconsistencies, enhancing model reliability. Additionally, it validates assumptions and communicates insights effectively, guiding stakeholders in decision-making processes. Overall, EDA serves as a foundational step, facilitating the development of accurate prediction models for flight delays.

### *A. Statistical Summary:*

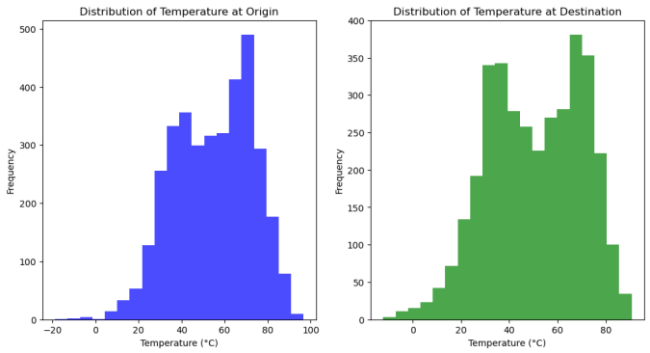


*Figure 4: Distribution of Temperature*

- The temperature variables at both the destination and origin have a broad range (e.g., -12.24 to 90.72 °C at the destination and -18.78 to 96.69 °C at the origin), reflecting diverse climatic conditions across the flight data.

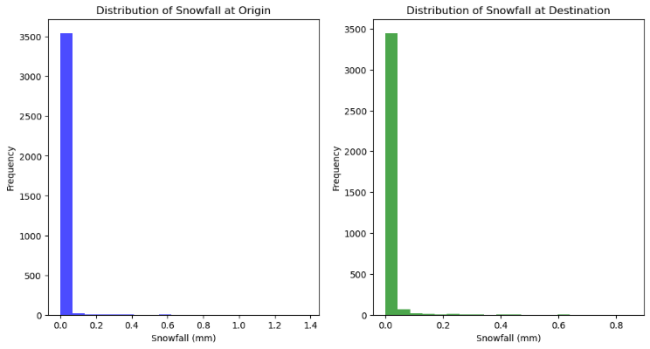


*Figure 5: Distribution of Snowfall*

- Precipitation (rain and snowfall) and snow depth are generally low, suggesting that most of the data points represent clear or mildly adverse weather conditions.
- Wind speeds and gusts also vary, with gusts reaching up to 50.33 m/s at the destination and 55.48 m/s at the origin, highlighting some flights encountering significant wind conditions.

### *B. Correlation Between Features:*

The correlation matrix heatmap illustrates the relationships between arrival delays and various weather-related variables at both the origin and destination, revealing generally weak correlations. Notably, snowfall and snow depth at both origin and destination present weak negative correlations (e.g., -0.19 for destination snowfall and -0.07 for origin snow depth), suggesting that more severe snow conditions could slightly decrease the likelihood of delays, potentially due to better preparation for known adverse conditions. Wind speed and gusts also show very low negative correlations, implying minimal impact on flight arrival times. A strong negative correlation between Temperature and Snow Depth (-0.70), highlighting where snowfall increases with depth, and higher temperatures reduce snow accumulation.

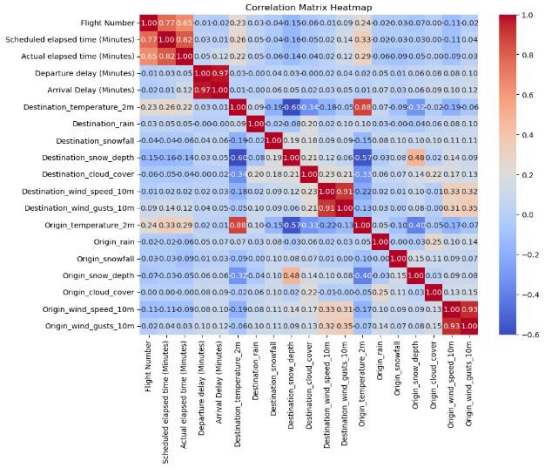


*Figure 6: Correlation Between Features*

### *C. Distribution of Arrival Delay with respect to SYR's weather conditions:*

The histograms illustrate the distribution of arrival delays in relation to different weather conditions at the destination: wind speed, rain, and snowfall.

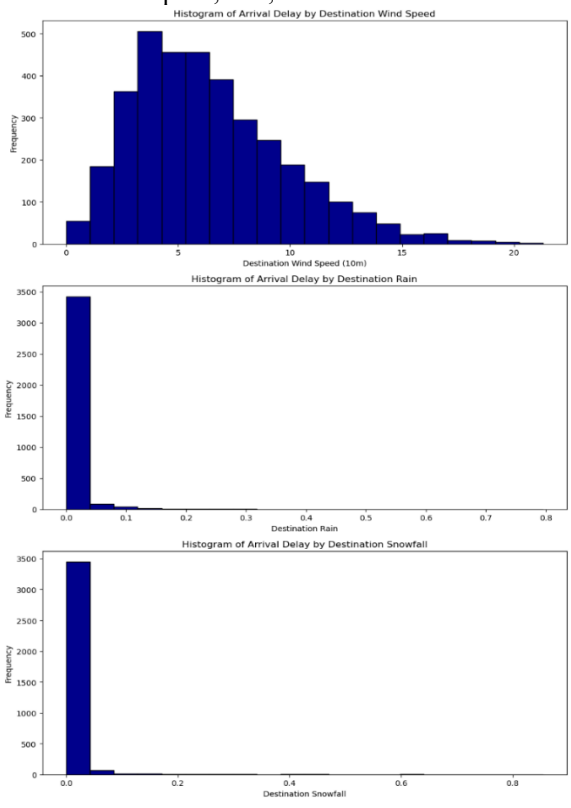


*Figure 7: Distribution of Arrival Delay with respect to Destination Weather conditions*

- *Wind Speed*: Most flights encounter low to moderate wind speeds (0-10 m/s), with a high frequency of arrivals under these conditions. As wind speed increases beyond 10 m/s, the number of flights experiencing delays increases significantly.
- *Rain*: The vast majority of flights arrive during conditions of no or very light rain (0-0.1 units), with a sharp drop in the number of flights as rainfall intensity increases. This indicates that while airports are generally well-equipped to handle light rain without significant delays, heavier rain is potentially disruptive.
- *Snowfall*: Similar to rain, the frequency of flights arriving with no snowfall is extremely high, and it decreases dramatically as snowfall intensity increases. This distribution implies that heavy snowfall might cause flights to be delayed.

### D. *Correlations between origin and destination weather variables:*

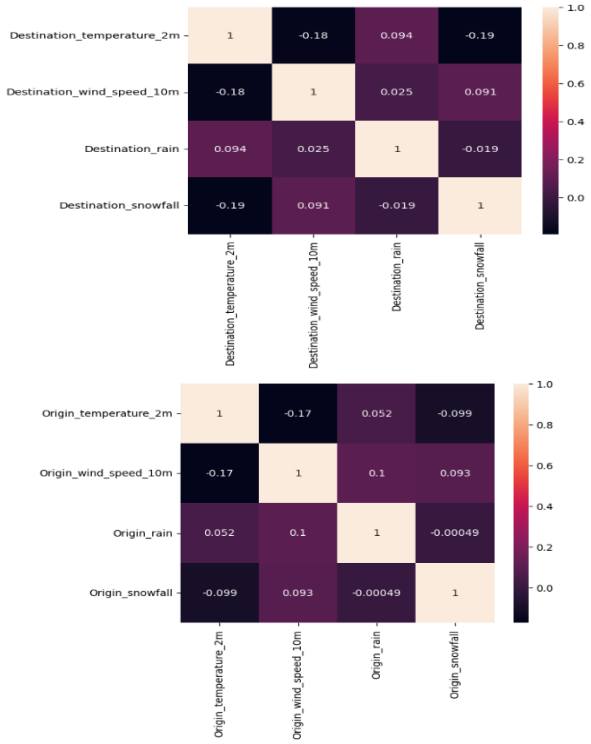


*Figure 8: Correlation Between Origin and Destination Weather Variables*

The two correlation matrices display relationships between weather conditions at the destination and origin of flights, respectively. For the destination, there's a notable negative correlation between temperature and snowfall (-0.19), suggesting that lower temperatures are associated with more snowfall, as expected. Wind speed and snowfall are somewhat positively correlated (0.091), indicating that higher wind speeds might coincide with snowfall events. . In the origin correlation matrix, there are similar patterns, with temperature and snowfall negatively correlated (-0.099), and wind speed and snowfall showing a slight positive relationship (0.093). These correlations help in understanding weather patterns that might impact flight conditions at both departure and arrival airports.

## V. Data Preprocessing

Data preprocessing is an important phase in data analysis, particularly for ensuring that the dataset is clean, consistent, and structured appropriately for analysis. This step enhances the accuracy and effectiveness of the predictive modeling, especially when dealing with complex datasets like those combining flight performance and weather conditions.

### A. *Handling NaN values:*

This step involves checking for NaN values in the dataset and filling them with appropriate values if needed. In our dataset, 'Tail Number' had few NaN values, and it is replaced with unknown since it does not hold much value in prediction.

```
# Replace NaN values with Unknown
first_flight_data['Tail Number'].fillna('Unknown', inplace=True)

# Check for missing values
missing_values = first_flight_data.isnull().sum()
print(missing_values)
```

```
Carrier Code                        0
Date (MM/DD/YYYY)                   0
Flight Number                       0
Tail Number                         0
Origin Airport                      0
Destination Airport                 0
Scheduled departure time            0
Actual departure time               0
Scheduled Arrival Time              0
Actual Arrival Time                 0
Scheduled elapsed time (Minutes)    0
Actual elapsed time (Minutes)       0
Departure delay (Minutes)           0
Arrival Delay (Minutes)             0
Destination_temperature_2m          0
Destination_rain                    0
Destination_snowfall                0
Destination_snow_depth              0
Destination_cloud_cover             0
Destination_wind_speed_10m          0
Destination_wind_gusts_10m          0
Origin_temperature_2m               0
Origin_rain                         0
Origin_snowfall                     0
Origin_snow_depth                   0
Origin_cloud_cover                  0
Origin_wind_speed_10m               0
Origin_wind_gusts_10m               0
dtype: int64
```

*Figure 9: Handling NaN values*

### B. *Combining date and Time Columns into Datetime Objects:*

This step involves converting separate date and time columns into datetime objects, which are easier to work with for time-based analysis. It uses the pd.to_datetime() function from the pandas library to combine the date and time information into a single datetime object for each flight.

### C. *Extracting Hour and Minute:*

After creating datetime objects, specific time components such as hours and minutes were extracted. This granularity is beneficial for scheduling and operational analysis, where details down to the minute can influence conclusions about peak times, delays, and operational efficiency.

### D. *Categorizing Flight Status:*

A custom function calculate_flight_status() was defined to assess each flight's punctuality based on the 'Arrival Delay (Minutes)' column. This categorization into *'Early'*, *'On-time'*, or *'Late'* allows for more nuanced analysis, such as determining the reliability of flight schedules or the impact of external factors like weather on flight timing.

```
def calculate_flight_status(arrival_delay_minutes):
    if arrival_delay_minutes < -5:
        return "Early"
    elif -5 <= arrival_delay_minutes <= 5:
        return "On-time"
    else:
        return "Late"

first_flight_data['Flight_Status'] = first_flight_data['Arrival Delay (Minutes)'].apply(calculate_flight_status)
```

*Figure 10: Categorizing Flight Status*

### E. *Feature Selection:*

Irrelevant and redundant features such as *'Carrier Code'*, *'Tail Number'*, and raw timestamps are dropped from the dataset to improve model efficiency and reduce overfitting. Dropping these columns is crucial for decluttering the dataset, focusing analyses on impactful variables, and ensuring efficient processing during data analysis.`

### F. *One-Hot Encoding:*

The categorical variable *'Origin Airport'* was transformed using one-hot encoding to facilitate its inclusion in classification models and other statistical analyses that require numerical input features. This encoding converts categorical data into a format that can be easily used by machine learning algorithms by creating binary columns for each category. In our case, the categories are *'Origin_MCO'*, *'Origin_JFK'* and *'Origin_ORD'*.

```
from sklearn.preprocessing import OneHotEncoder

def get_ohe(df, col):
    ohe = OneHotEncoder(drop='first', handle_unknown='error', sparse_output=False, dtype='int')
    ohe.fit(df[[col]])
    temp_df = pd.DataFrame(data=ohe.transform(df[[col]]), columns=ohe.get_feature_names_out())
    df.drop(columns=[col], axis=1, inplace=True)
    df = pd.concat([df.reset_index(drop=True), temp_df], axis=1)
    return df

first_flight_data = get_ohe(first_flight_data, 'Origin Airport')
first_flight_data.head()
```

*Figure 11: One-hot Encoding*

### G. *Scaling:*

The data is standardized using StandardScaler to scale numerical features to have mean 0 and standard deviation 1, which can improve the performance of machine learning algorithms. Here, we used the fit_transform() method on our training data (X_train) to compute the mean and standard deviation of each feature and then standardized the data accordingly. Subsequently, we applied the transform() method to standardize the test data (X_test), ensuring consistency with the scaling parameters obtained from the training data.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = pd.DataFrame(sc.fit_transform(X_train), columns = X_train.columns, index = X_train.index)
X_test = pd.DataFrame(sc.transform(X_test), columns = X_test.columns, index = X_test.index)
X_train
X_test
```

*Figure 12: Standard Scaler*

## VI. Model Training And Testing

In this analysis, we explore the performance of three popular classification algorithms—Random Forest, Gradient Boosting, and XGBoost—for predicting flight statuses. Each model undergoes rigorous training, hyperparameter tuning, and evaluation to assess its accuracy, precision, recall, and F1-score. Through this comparison, we aim to identify the most effective model for flight status prediction.

### A. *Random Forest Classifier:*

The Random Forest algorithm utilizes an ensemble of decision trees to improve classification accuracy and prevent overfitting.

We initiated our modeling by setting up a comprehensive grid of hyperparameters to optimize the Random Forest Classifier. This included variations in the number of trees (n_estimators at 100 and 200), the number of features considered for splitting at each leaf (max_features set to 'log2' and 'sqrt'), tree depth (max_depth with values of 2, 5, or unlimited), and the rules for splitting nodes (min_samples_split and min_samples_leaf) [5]. The selection and tuning of these parameters were systematically performed using GridSearchCV, which applies exhaustive search across the specified parameter grid with 3-fold cross-validation, ensuring the model's generalizability.

The model's performance was evaluated using precision, recall, and F1-score metrics, offering an understanding of its predictive accuracy across various flight status categories such as 'Early', 'Late', and 'On-time'. The analysis revealed a notably strong performance in predicting 'Early' status flights, with an F1-score of 0.68, indicating high precision and recall. The model exhibited an accuracy of 53.2%. However, the model exhibited less satisfactory performance in predicting 'Late' and 'On-time' statuses. This disparity in predictive accuracy across different categories suggests potential areas for enhancement, which could involve further fine-tuning of the model, incorporating additional features, or a combination of both strategies.

### B. *Gradient Boosting Classifier:*

Gradient Boosting model specifically aims to correct the errors of the previous ones, which makes this method particularly effective for complex classification tasks such as flight status prediction where data may contain various interacting patterns.

We began by establishing a rigorous grid of hyperparameters for the Gradient Boosting Classifier to optimize its performance. This included configurations for the number of trees (n_estimators), the rate of learning (learning_rate), the depth of each tree (max_depth), and the minimum number of samples required at leaves and for splits (min_samples_leaf and min_samples_split). These parameters were meticulously explored through GridSearchCV using 3-fold cross-validation, ensuring the selected model settings were not only optimal but also robust across different subsets of data.

The optimal settings identified via grid search were a learning rate of 0.1, a maximum depth of 2, and a relatively small tree ensemble of 100 estimators, suggesting a simpler model was effective within our data context. Despite achieving an accuracy of around 53% on the test set and 60% on the training set, there was a clear indication of the model's limitations in generalizing beyond the training data, a challenge often encountered in predictive modeling.

### C. *XGBoost Classifier:*

XGBoost Classifier, a sophisticated ensemble machine learning algorithm noted for its high efficiency and scalability. XGBoost stands out for its ability to handle large datasets with excellent accuracy and speed, making it a popular choice in many competitive machine learning environments.

To optimize the XGBoost model [4], we configured a comprehensive grid of hyperparameters, which included the number of boosting rounds (n_estimators with values 100 and 200), the learning rate (learning_rate at 0.01 and 0.1 to

manage the speed and quality of learning), the maximum depth of the trees (max_depth set at 3, 5, and 7 to control overfitting), and the subsampling rates by observations and features (subsample and colsample_bytree each tested at 0.6, 0.8, and 1.0) [6]. These parameters were meticulously tested using GridSearchCV that combined exhaustive search with 3-fold cross-validation to identify the best model parameters, ensuring the model's effectiveness and robustness.

```
# Hyperparameters setup
param_grid_xgb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}

# Initialize and fit the GridSearchCV
xgb_model = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='mlogloss')
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid_xgb, cv=3, n_jobs=-1, scoring='accuracy')
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.fit_transform(y_test)
grid_search_xgb.fit(X_train, y_train_encoded)

# Best estimator
best_xgb = grid_search_xgb.best_estimator_
```

*Figure 13: Hyperparameter tuning using GridSearchCV.*

The best performing model configuration included a learning rate of 0.01, a tree depth of 5, using 200 trees, and subsample and colsample_bytree rates of 0.6 and 0.8, respectively. The analysis revealed an accuracy of approximately 55% on the test set and 64% on the training set, indicating some overfitting but generally effective learning from the training data. The model showed strong performance in predicting 'Early' flights, with a notable F1-score, but its effectiveness was lower in accurately classifying 'Late' and 'On-time' flights.

*Feature Importance:*

.

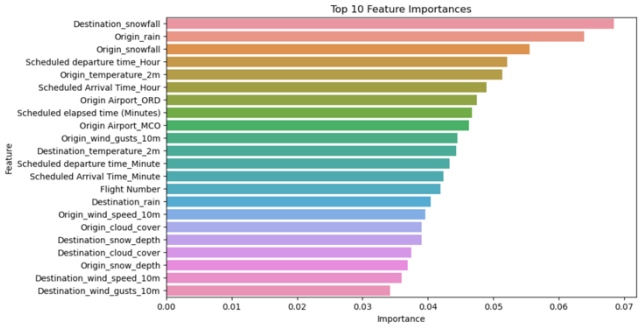


*Figure 14: Feature Importance*

This visualization is crucial for quickly identifying which variables are most influential in predicting flight status. For example, from the bar plot, we can observe that factors such as Destination_snowfall, Origin_rain, and Scheduled departure time_Hour are among the most influential features.

| | Accuracy | Precision (weighted avg) | Recall (weighted avg) | F1-score (weighted avg) |
|---|---|---|---|---|
| Random Forest Classifier | 0.532 | 0.46 | 0.53 | 0.45 |
| Gradient Boosting Classifier | 0.535 | 0.48 | 0.53 | 0.45 |
| XGBoost Classifier | 0.552 | 0.48 | 0.55 | 0.46 |

*Figure 15: Comparison of metrics across models*

XGBoost Classifier stands out as the most effective model for this particular task, demonstrating superior performance across all key metrics: accuracy, precision, recall, and F1-score. This suggests that XGBoost is better suited for scenarios where both the accuracy of the positive predictions and the balance between identifying as many positives as possible (without increasing false positives unduly) are crucial.

## VII. Predictions And Results

### A. *First Flight Arrival Status Predictions:*

The test flight data is merged with the forecast weather data for corresponding dates and times, ensuring that the forecast weather relevant to the scheduled departure and arrival times of flights is accurately paired. This process is replicated across different datasets for each airport.

| | Date (MM/DD/YYYY) | Flight Number | Origin Airport | Scheduled departure time | Scheduled Arrival Time | ELAPSED TIME |
|---|---|---|---|---|---|---|
| 0 | 4/19/2024 | 1400 | ORD | 18:52 | 21:47 | 2:55 |
| 1 | 4/19/2024 | 116 | JFK | 13:34 | 14:51 | 1:17 |
| 2 | 4/19/2024 | 5285 | MCO | 11:35 | 14:20 | 2:45 |
| 3 | 4/20/2024 | 1400 | ORD | 18:52 | 21:47 | 2:55 |
| 4 | 4/20/2024 | 116 | JFK | 13:25 | 14:41 | 1:16 |
| 5 | 4/21/2024 | 1400 | ORD | 18:52 | 21:47 | 2:55 |
| 6 | 4/21/2024 | 116 | JFK | 13:35 | 14:51 | 1:16 |
| 7 | 4/21/2024 | 5285 | MCO | 11:05 | 13:50 | 2:45 |
| 8 | 4/22/2024 | 1400 | ORD | 18:52 | 21:47 | 2:55 |
| 9 | 4/22/2024 | 116 | JFK | 13:35 | 14:51 | 1:16 |
| 10 | 4/22/2024 | 5285 | MCO | 11:35 | 14:20 | 2:45 |

*Figure 16: First Flight Test Data*

Before making predictions, the test dataset is prepared in a format compatible with the trained XGB model. This includes applying any preprocessing steps such as scaling features or encoding categorical variables that were used during the training phase. The pre-trained XGB model, which has already been trained on a dataset with known outcomes (e.g., flight statuses like "on-time, late, early"), is loaded into the environment.

The test dataset is fed into the XGB model. The model uses the information from the test data to make flight arrival status predictions based on the patterns it learned during training.

| | Flight Status | Date (MM/DD/YYYY) | Flight Number | Origin Airport | Scheduled departure time | Scheduled Arrival Time | ELAPSED TIME |
|---|---|---|---|---|---|---|---|
| 0 | Early | 4/19/2024 | 1400 | ORD | 18:52 | 21:47 | 2:55 |
| 1 | Early | 4/19/2024 | 116 | JFK | 13:34 | 14:51 | 1:17 |
| 2 | Early | 4/19/2024 | 5285 | MCO | 11:35 | 14:20 | 2:45 |
| 3 | Early | 4/20/2024 | 1400 | ORD | 18:52 | 21:47 | 2:55 |
| 4 | Early | 4/20/2024 | 116 | JFK | 13:25 | 14:41 | 1:16 |
| 5 | Early | 4/21/2024 | 1400 | ORD | 18:52 | 21:47 | 2:55 |
| 6 | Early | 4/21/2024 | 116 | JFK | 13:35 | 14:51 | 1:16 |
| 7 | Early | 4/21/2024 | 5285 | MCO | 11:05 | 13:50 | 2:45 |
| 8 | Late | 4/22/2024 | 1400 | ORD | 18:52 | 21:47 | 2:55 |
| 9 | Early | 4/22/2024 | 116 | JFK | 13:35 | 14:51 | 1:16 |
| 10 | Late | 4/22/2024 | 5285 | MCO | 11:35 | 14:20 | 2:45 |

*Figure 17: Predictions for First Flight Arrival Status*

### B. *Second Flight Arrival Status Predictions:*

Now, after predicting the arrival status of the first flight, we are going to predict the status of the subsequent flight based on the first one's status.

For the second flight dataset, we are going to follow the same steps as before as we did for the first flight dataset preparation. This involves:

- Data Loading – Loading the second flight dataset which we are going to use for model training.
- Data Merging – Merging the dataset with destination and origin weather dataset using scheduled departure time and schedule arrival time as a merging key.
- Data Preprocessing – Data preprocessing steps include extracting hour and minute from datetime columns, performing one-hot encoding, scaling the features, and removing unnecessary and irrelevant columns.

*Model Training for Second Flight Predictions:*

For the new model we are going to utilize the predicted arrival status from earlier flights as input features to enhance the accuracy of predicting subsequent flight statuses. The dataset containing subsequent flight details is then merged with these predictions. The key focus here is the 'Flight_Status' column from the first flight predictions, which is linked to the subsequent flight data using the date and origin. Post-merging, this column is aptly renamed to 'Prev_Flight_Status' to differentiate it from the current flight status and to indicate it as a historical data point. Any missing values in the 'Prev_Flight_Status' are filled with 'Unknown', which handles cases where the first flight data might be missing. Categorical data such as Origin Airport and Prev_Flight_Status are converted into numerical format using one-hot encoding.

```
Index(['Date (MM/DD/YYYY)', 'Flight Number',
       'Scheduled elapsed time (Minutes)', 'Destination_temperature_2m',
       'Destination_rain', 'Destination_snowfall', 'Destination_snow_depth',
       'Destination_cloud_cover', 'Destination_wind_speed_10m',
       'Destination_wind_gusts_10m', 'Origin_temperature_2m', 'Origin_rain',
       'Origin_snowfall', 'Origin_snow_depth', 'Origin_cloud_cover',
       'Origin_wind_speed_10m', 'Origin_wind_gusts_10m',
       'Scheduled departure time_Hour', 'Scheduled departure time_Minute',
       'Scheduled Arrival Time_Hour', 'Scheduled Arrival Time_Minute',
       'Flight_Status', 'Origin Airport_MCO', 'Origin Airport_ORD',
       'Prev_Flight_Status_Late', 'Prev_Flight_Status_On-time',
       'Prev_Flight_Status_Unknown'],
      dtype='object')
```

*Figure 18: Selected Features for Second Flight Predictions*

| Prev_Flight_Status_Late | Prev_Flight_Status_On-time | Prev_Flight_Status_Unknown |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 0 |

*Figure 19: Previous Flight Arrival Status(One-hot Encoded)*

For model training, we are using the previously found best parameters using GridSearchCV hyperparameter tuning for XGBoost Classifier. The prepared second flight dataset is given as input to the XGB classifier, and the flight status is predicted for the subsequent flight using weather data and also the previously predicted first flight dataset as a feature for this model training which is classified as 'Prev_Flight_Status' (Late, Early, On-time or Unknown).

```
# Define parameters
params = {
    'colsample_bytree': 0.8,
    'learning_rate': 0.01,
    'max_depth': 5,
    'n_estimators': 200,
    'subsample': 0.6,
    'random_state': 42
}
xgb_classifier = xgb.XGBClassifier(**params)

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train_second)
y_test_encoded = label_encoder.fit_transform(y_test_second)
xgb_classifier.fit(X_train_second, y_train_encoded)

# Evaluate the model
print('R-squared value for train data:')
print(xgb_classifier.score(X_train_second, y_train_encoded))
print('R-squared value for test data:')
print(xgb_classifier.score(X_test_second, y_test_encoded))
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=None,

R-squared value for train data:
0.656125733762545
R-squared value for test data:
0.5934897804693414
```

*Figure 20: Model Training for Second Flight Predictions*

The analysis revealed an accuracy of approximately 59% on the test set and 65% on the training set, indicating some overfitting but generally effective learning from the training data.

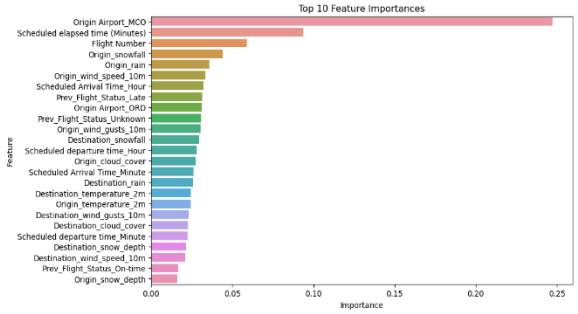


*Figure 21: Feature Importance for Subsequent Flight Prediction Model*

From the above image, we can infer that along with weather related features, the flight status of the previous flights also plays a major role in improving the model prediction and is of high importances.

*Second Flight Arrival StatusPredictions using XGB Model:*

The test flight data for the subsequent flight data is merged with the forecast weather data for corresponding dates and times, ensuring that the forecast weather relevant to the scheduled departure and arrival times of flights is accurately paired.

Before making predictions, the test dataset is prepared in a format compatible with the trained XGB model. This includes applying any preprocessing steps such as scaling features or encoding categorical variables that were used during the training phase. We have to make sure that the previous flight status is included in the test dataset as a feature and also the test dataset has same columns as the train dataset.

| | Date (MM/DD/YYYY) | Flight Number | Origin Airport | Scheduled departure time | Scheduled Arrival Time | ELAPSED TIME | Prev_Flight_Status_Unknown | Prev_Flight_Status_On-time | Prev_Flight_Status_Late |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2024-04-19 | 3402 | ORD | 19:59:00 | 22:52:00 | 02:53:00 | 0 | 0 | 0 |
| 1 | 2024-04-19 | 3402 | ORD | 19:59:00 | 22:52:00 | 02:53:00 | 0 | 1 | 0 |
| 2 | 2024-04-19 | 3402 | ORD | 19:59:00 | 22:52:00 | 02:53:00 | 0 | 0 | 1 |
| 3 | 2024-04-19 | 5182 | JFK | 14:55:00 | 16:21:00 | 01:26:00 | 0 | 0 | 0 |
| 4 | 2024-04-19 | 5182 | JFK | 14:55:00 | 16:21:00 | 01:26:00 | 0 | 1 | 0 |
| 5 | 2024-04-19 | 5182 | JFK | 14:55:00 | 16:21:00 | 01:26:00 | 0 | 0 | 1 |

*Figure 22: Subsequent Flight Test Data*

The second flight test dataset is fed into the pre-trained XGB model which includes the first flight's arrival status prediction as a feature.

| | Flight Status | Date (MM/DD/YYYY) | Flight Number | Origin Airport | Scheduled departure time | Scheduled Arrival Time | ELAPSED TIME | Prev_Flight_Status_Unknown | Prev_Flight_Status_On-time | Prev_Flight_Statu |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Early | 2024-04-19 | 3402 | ORD | 19:59:00 | 22:52:00 | 02:53:00 | 0 | 0 | |
| 1 | Early | 2024-04-19 | 3402 | ORD | 19:59:00 | 22:52:00 | 02:53:00 | 0 | 1 | |
| 2 | Early | 2024-04-19 | 3402 | ORD | 19:59:00 | 22:52:00 | 02:53:00 | 0 | 0 | |
| 3 | Early | 2024-04-19 | 5182 | JFK | 14:55:00 | 16:21:00 | 01:26:00 | 0 | 0 | |

*Figure 23: Predictions for Second Flight Arrival Status Based on First Flight Arrival Status*

## VIII. Conclusion

The Syracuse Flight Arrival Delay Prediction project successfully demonstrated the potential of advanced machine learning techniques to predict flight statuses with considerable accuracy. By integrating comprehensive datasets that included historical flight data and detailed weather conditions from both departure and arrival airports, our models were able to capture the complex dependencies that affect flight arrival timings. The utilization of XGBoost classifier provided robust framework for our predictive model, contributing to a deeper understanding of the factors influencing flight delays.

Our findings underline the significant role that preceding flight statuses play in predicting subsequent flight timings, emphasizing the cascading effects of delays within daily flight schedules. This insight is particularly valuable for airlines and airport operations, providing a predictive tool to enhance operational planning and passenger communication.

Future research could expand upon this work by incorporating real-time data feeds and more granular weather predictions to further refine the accuracy of flight status predictions.

## References


[1] https://www.transtats.bts.gov/ontime/.

[2] https://open-meteo.com/en/docs/historical-weather-api

[3] https://open-meteo.com/en/docs

[4] R. T. Reddy, P. Basa Pati, K. Deepa and S. T. Sangeetha, "Flight Delay Prediction Using Machine Learning," 2023 IEEE 8th International Conference for Convergence in Technology (I2CT), Lonavla, India, 2023, pp. 1-5, doi: 10.1109/I2CT57861.2023.10126220.

[5] N. L. Kalyani, G. Jeshmitha, B. S. Sai U., M. Samanvitha, J. Mahesh and B. V. Kiranmayee, "Machine Learning Model - based Prediction of Flight Delay," 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2020, pp. 577-581, doi: 10.1109/I-SMAC49090.2020.9243339.

[6] P. V. Kavitha, L. Manoranjani, V. Mithra and P. Monal, "Flight Delay Prediction using Machine Learning Model," 2022 International Conference on Futuristic Technologies (INCOFT), Belgaum, India, 2022, pp. 1-4, doi: 10.1109/INCOFT55651.2022.10094407.