CHAPTER 4

INITIAL RESULTS

4.1 Introduction

The purpose of this chapter is to present the main findings from the analysis of flight data that was used to predict airline delays. This section begins with an overview of the dataset structure and the distribution of flights that are delayed versus those that are on time. Identifying trends and patterns relating to delays is accomplished through exploratory analysis. After that, the data preprocessing steps will be summarized, including the handling of missing values, encoding, and feature scaling. By using standard evaluation metrics, three models are compared. The chapter concludes with hey insights that inform the effectiveness of each model and guide further improvements.

4.2 Exploratory Data Analysis (EDA)

To gain a better understanding of the structure, distribution, and patterns of the dataset related to flight delays, exploratory data analysis was conducted. Our first step was to ensure the dataset, which includes information such as departure carrier, weather conditions, and delay duration time, was complete.

4.2.1 Data Collection

Column Name	Description	Relevance to Delay Prediction
MONTH	Month of the flight (Jan - Dec)	Weather and demand can influence delay in different seasons

DAY_OF_MONTH	Day of Month (1 -31)	In some cases such as holiday, delays may be more noticeable
DAY_OF_WEEK	Day of the week (1-Monday,...,7-Sunday)	The weekend travel pattern may affect congestion and delays
OP_UNIQUE_CARRIER	Airline code (AA, UA, DL)	Depending on the airline, on-time performance can vary
TAIL_NUM	Aircraft tail number (unique identifier)	Indirectly relevant. Often omitted to reduce noise
DEST	Destination airport code	Congestion causes delays at some destinations
DEP_DELAY	Departure delay in minutes	Label created with target variable (is_delayed)
CRS_ELAPSED_TIME	Scheduled duration of the flight	The timing might be stricter on longer flights due to more delay buffer
DISTANCE	Distance between destination and origin	Schedules and delays may be related
CRS_DEP_M	Scheduled departure time in minutes from midnight	Suitable for extracting trends by time of day
DEP_TIME_M	Actual departure time in minutes from midnight	Use to calculate duration delay
CRS_ARR_M	Scheduled arrival time in minutes from midnight	Planned flight duration can be calculated with CRS_DEP_M
Temperature	Temperature at departure time	Temperatures can affect aircraft performance and cause delays
Dew Point	Measure of moisture (°F)	Visibility and fog conditions may be affected
Humidity	Relative Humidity (%)	Delay associated with weather may be caused by high humidity
Wind	Wind direction	The use of runways may be

		affected; only standardized data is useful
Wind Speed	Speed of wind at the departure location	Take-off and landing can be delayed due to high wind
Wind Gust	Maximum recorded wind gust	Flights may be temporarily grounded by sudden gusts of wind
Pressure	Atmospheric pressure	Storms or bad weather may be indicated by low pressure
Condition	Weather condition	Impact delay risk directly
sch_dep	Scheduled departure timestamp	Time-based features are often dropped after extraction
sch_arr	Scheduled arrival timestamp	Time-based features are often dropped after extraction
TAXI_OUT	Time taken to taxi from gate to runway (in minutes)	Congestion and delays often correlate with longer taxi wait times

Table 4.1: Data Description

4.2.2 Basic Inspection

A basic inspection of the dataset was conducted to understand its structure, data types, and completeness. Summary information, such as the number of rows and columns, the type of column data, and any missing values, was analyzed to identify any potential data quality issues. Using the DEP_DELAY column, the binary classification target is_delayed was created, where 1 indicates a flight delayed by more than 15 minutes and 0 indicates an on-time flight. As a result of this transformation, the analysis focused on a clear classification task. Figure 4.1 shows the Delay Class Distribution based on the initial dataset.

```
 Delay Class Distribution:
is_delayed
0    86.557946
1    13.442054
```

Figure 4.1: Initial Delay Class Distributions

4.2.3 Class Distribution Analysis

Class distribution analysis was used to visualize the proportion of delayed flights versus those that arrived on time, based on Figure 4.2.

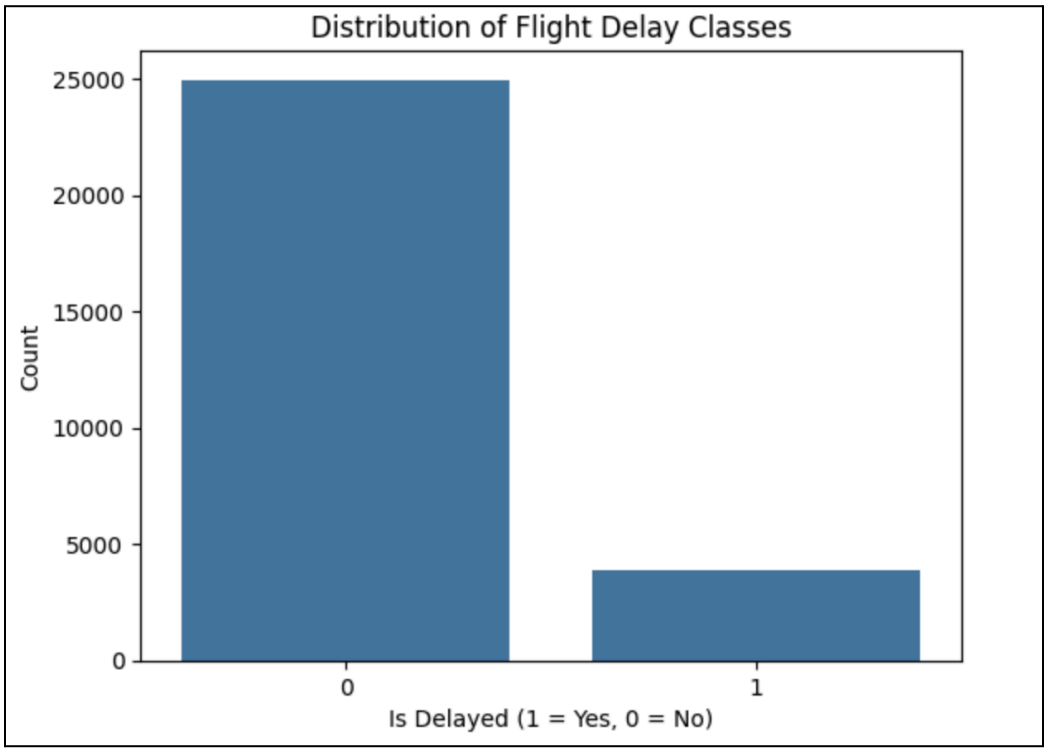

Figure 4.2: Bar Chart Distribution of Flight Delay Classes

There was a noticeable imbalance in the results, with more on-time flights, suggesting further evaluation should consider F1 score and recall metrics. Data from the DEP_TIME column was used to plot the frequency of delays throughout the day to examine temporal patterns. Traffic congestion or weather conditions may have contributed to delays earlier in the morning and later in the evening.

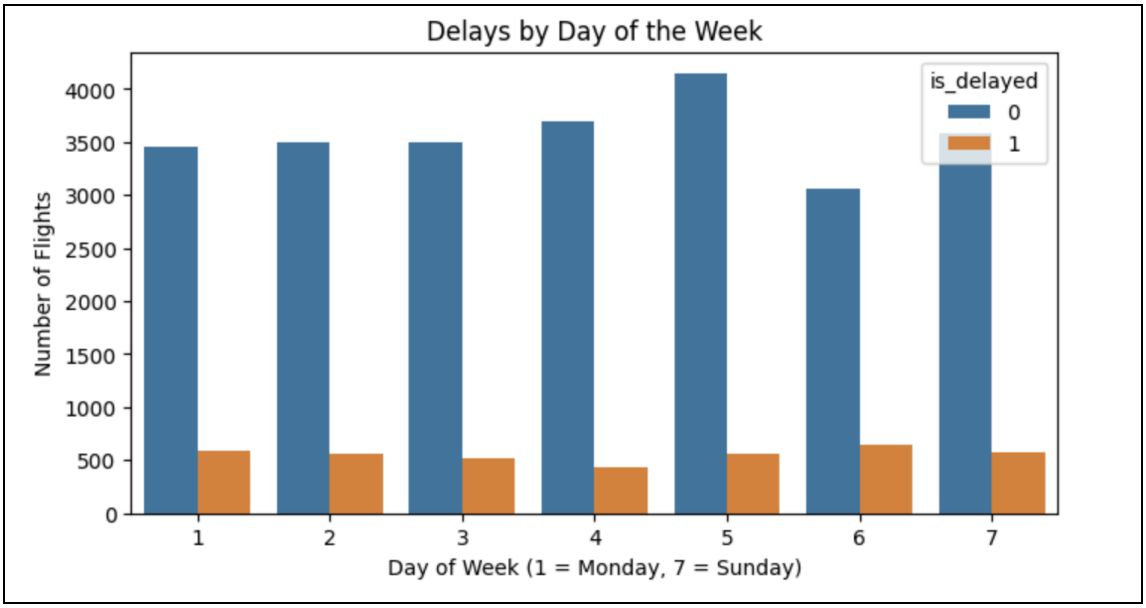

Figure 4.3: Clustered Bar Chart Delays by Day of the Week

Based on Figure 4.3, it was also observed that delays were more frequent on Saturdays and Sundays, which was possibly due to heavier traffic on weekends.

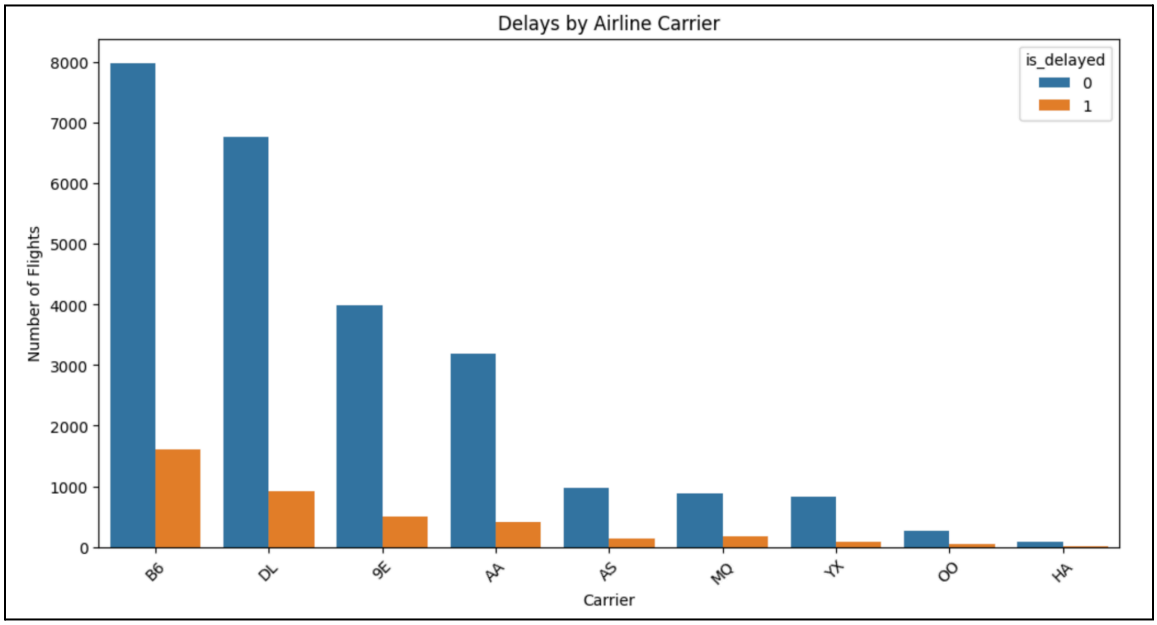

Figure 4.4: Clustered Bar Chart Delays by Airline Carrier

Moreover, categorical trends were analyzed, such as delays by airline carrier. There were significantly higher delays with some carriers compared to others in the plot based on Figure 4.4, which suggests that carrier performance and scheduling efficiency may contribute to this.

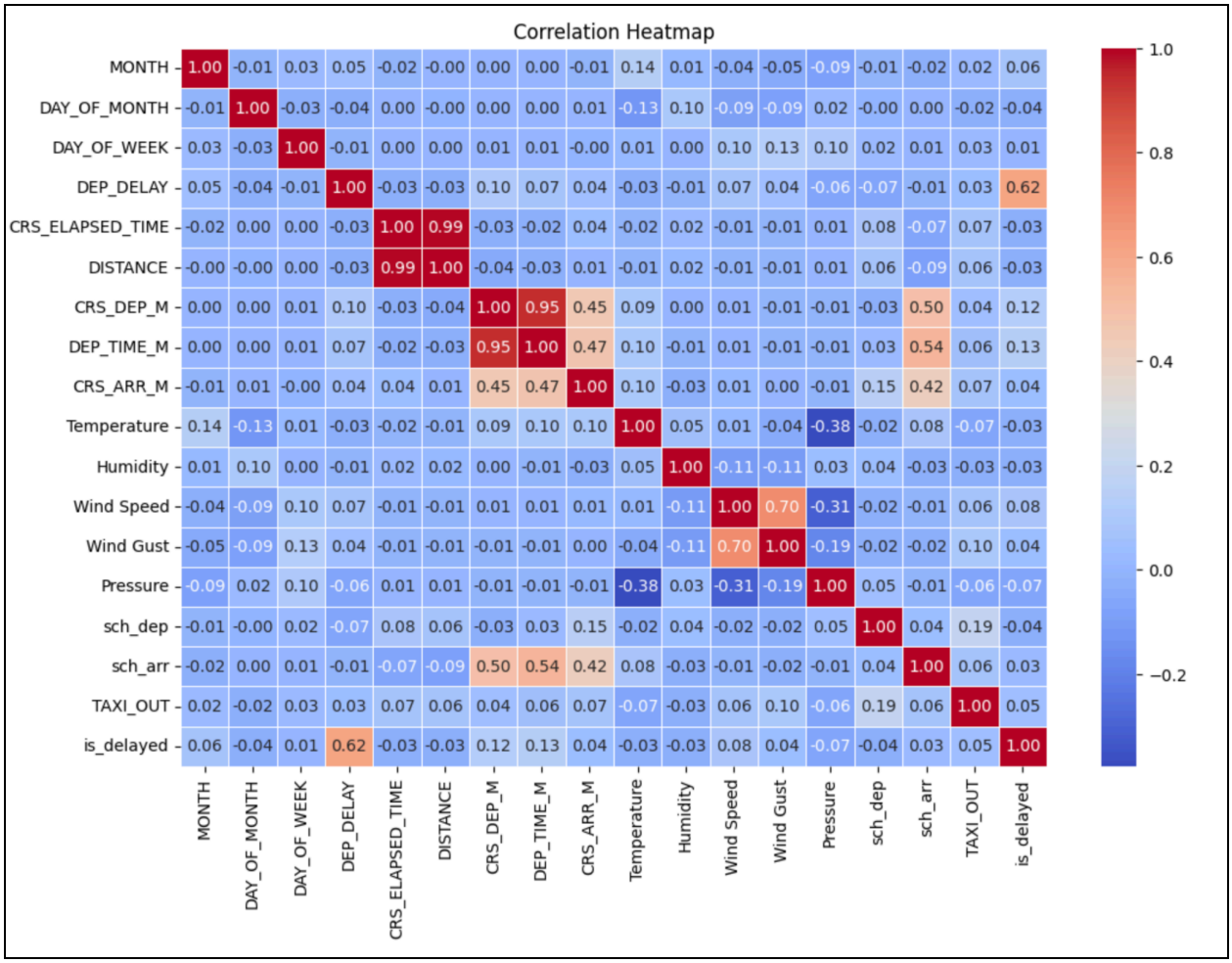

Figure 4.5: Correlation Heatmap

Figure 4.5 shows correlation heatmaps were generated to analyze relationships between numerical features. Some features, such as taxi-out time, humidity, and wind speed, showed moderate correlations with delays, suggesting their potential to be used in predictive modelling.

4.2.4 Boxplot

Boxplots were used to visualize the relationship between weather variables and flight delays.

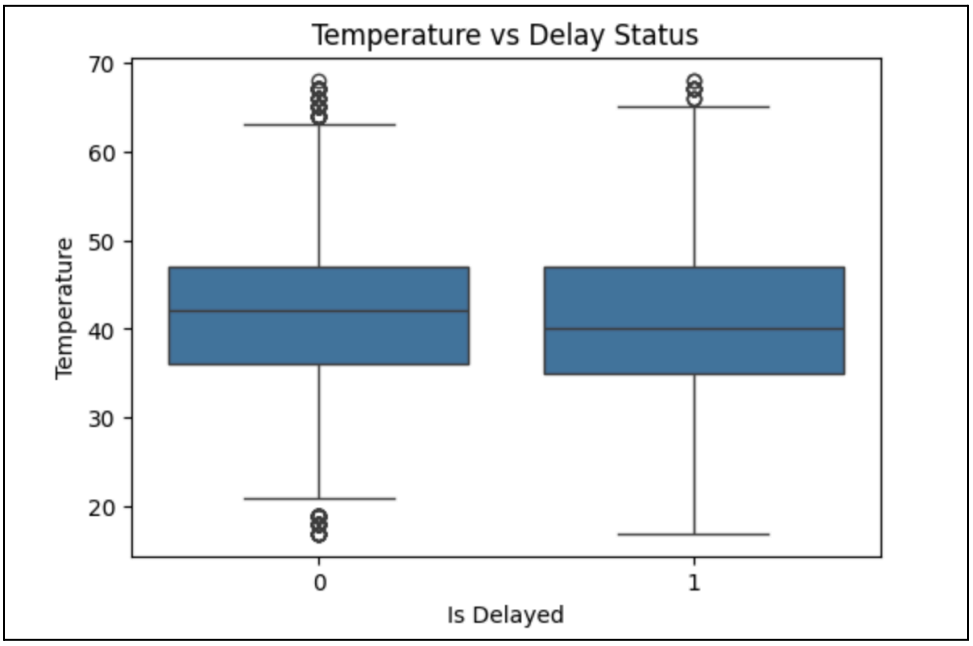

Figure 4.6: Boxplot Temperature vs Delay Status

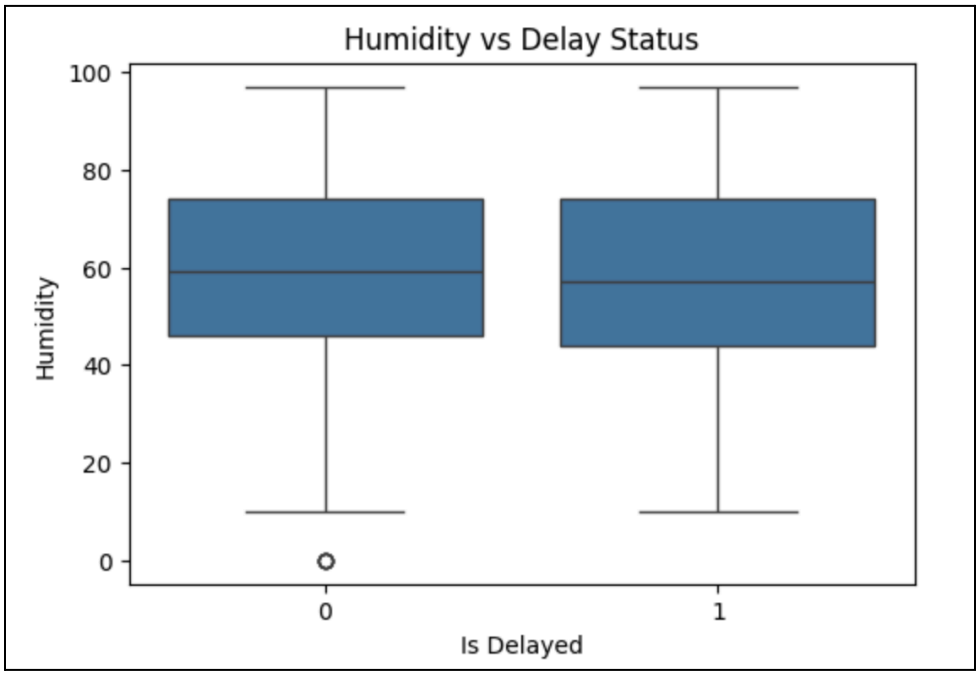

Figure 4.7: Boxplot Humidity vs Delay Status

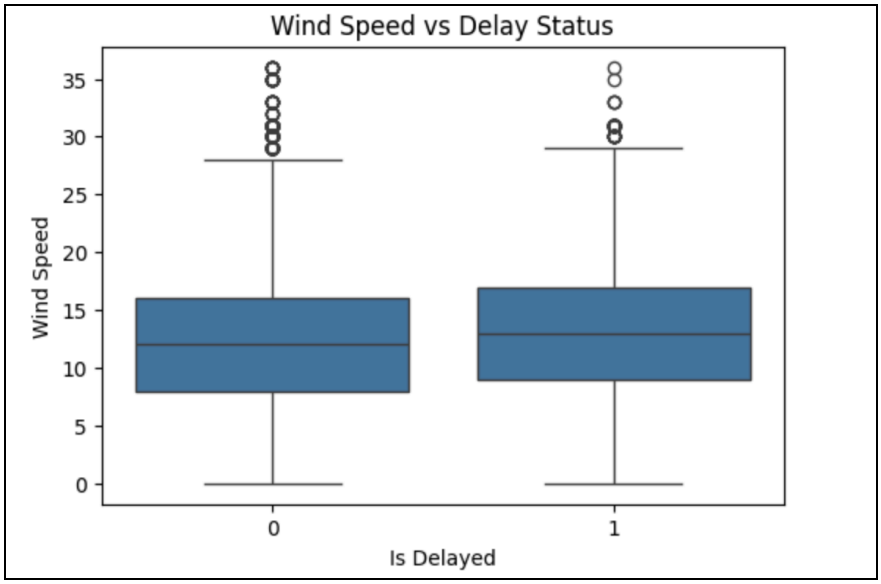

Figure 4.8: Boxplot Wind Speed vs Delay Status

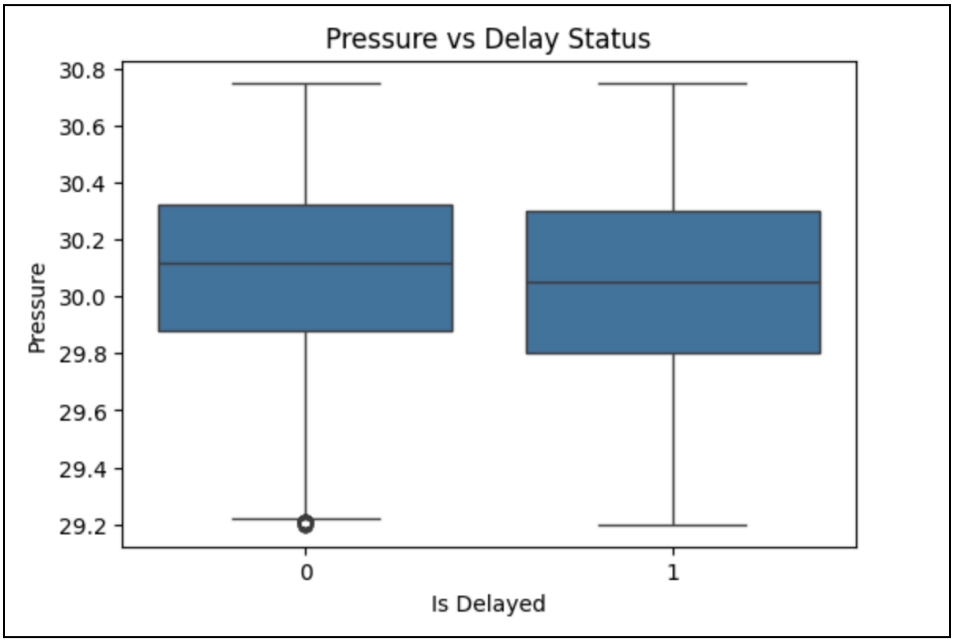

Figure 4.9: Boxplot Pressure vs Delay Status

According to comparisons of temperature, humidity, wind speed, and pressure between delayed flights and non-delayed flights, delays are more likely to occur in cool, humid, and windy environments. However, these differences were not pronounced, suggesting that while weather plays a role in flight delays, it is likely only one of several contributors. A key outcome of the EDA was the identification of relevant features and patterns that could be employed to enhance the accuracy of prediction models.

4.3 Data Preparation and Cleaning Data

Initially, we examined the raw dataset's structure and dimensions to confirm its structure and dimensions. Next, both numerical and categorical features were cleaned and prepared for model development. We converted key numerical columns, such as temperature, dew point, humidity, wind speed, wind gust, pressure, and taxi-out time, to a numeric format to handle any nonstandard values. In these columns, missing values were filled using the mean of each column to maintain statistical consistency while preserving data volume.

```
def clean_flight_data(filepath):
    # Load dataset
    df = pd.read_csv(filepath)
    print(" Loaded data with shape:", df.shape)

    # Identify numeric weather/operational columns
    numeric_cols = ['Temperature', 'Dew Point', 'Humidity', 'Wind Speed', 'Wind Gust', 'Pressure', 'TAXI_OUT']
    for col in numeric_cols:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')
            df[col].fillna(df[col].mean(), inplace=True)
```

Figure 4.10: Data Cleaning Code

As part of this study, categorical features were also addressed, with a special emphasis on the condition column, which represents weather conditions. For completeness, all missing values were filled with the placeholder "Unknown". In addition, non-relevant columns such as TAIL_NUM, SCH_DEP, and SCH_ARR were removed from the dataset, since they were unlikely to contribute meaningful predictive power.

```
# Fill missing values for 'Condition' (categorical)
if 'Condition' in df.columns:
    df['Condition'].fillna('Unknown', inplace=True)

# Optional: Drop high-cardinality or irrelevant columns
drop_cols = ['TAIL_NUM', 'sch_dep', 'sch_arr']
df.drop(columns=[col for col in drop_cols if col in df.columns], inplace=True)
```

Figure 4.11: Missing Value and Irrelevant Columns

To make machine-readable variables, categorical variables such as the operating carrier (OP_UNIQUES_CARRIER), the destination airport (DEST), and the weather condition

(Condition) were label-encoded. In this step, text labels are converted into numerical values that are suitable for use in machine learning algorithms. In addition, a new binary classification target variable was created named is_delayed. Using this variable, flight delays be more than 15 minutes are assigned a value of 1 and all other flights receive a value of 0, simplifying the modelling task into a binary classification problem.

```
# Encode categorical columns
cat_cols = ['OP_UNIQUE_CARRIER', 'DEST', 'Condition']
for col in cat_cols:
    if col in df.columns:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col].astype(str))

# Create delay classification label
if 'DEP_DELAY' in df.columns:
    df['is_delayed'] = df['DEP_DELAY'].apply(lambda x: 1 if x > 15 else 0)
```

Figure 4.12: Encode and Delay Label Code

A final step was to remove any rows that contained missing values to ensure that model training would not be disrupted by missing values. A new CSV file named cleaned_flight_data.csv was created from the cleaned dataset. Through the cleaning process, the dataset was transformed into a structured, complete, and machine-readable one and ready for predictive modeling.

```
    # Drop remaining rows with missing values (if any)
    df.dropna(inplace=True)

    # Final report
    print("Final cleaned shape:", df.shape)
    print("Missing values per column:\n", df.isnull().sum())
    return df

# Usage
cleaned_df = clean_flight_data('M1_final.csv')

# Optional: Save to new CSV
cleaned_df.to_csv('cleaned_flight_data.csv', index=False)
print("Cleaned dataset saved as 'cleaned_flight_data.csv'")
```

Figure 4.13: Final Steps of Data Cleaning Code

4.4 Model Development

4.4.1 Random Forest

Using the Random Forest model, it was possible to predict whether flights would be delayed by more than 15 minutes. Based on Figure 4.14, after loading the cleaned dataset, the original DEP_DELAY column was removed, since it wasn't needed after creating the binary target variable is_delayed. After that, the dataset was divided into features (X) and target labels (Y). With LabelEncoder, several categorical columns, including OP_UNIQUE_CARRIER, DEST, Condition, and Wind, were converted into numerical form for model input.

```
# Load the cleaned dataset
df = pd.read_csv('cleaned_flight_data.csv')

# Drop raw delay column (optional) and select features + target
if 'DEP_DELAY' in df.columns:
    df = df.drop(columns=['DEP_DELAY'])

# Separate features and label
X = df.drop('is_delayed', axis=1)
y = df['is_delayed']

# List of categorical columns that need encoding
cat_cols = ['OP_UNIQUE_CARRIER', 'DEST', 'Condition', 'Wind']

# Encode each categorical column with LabelEncoder
le = LabelEncoder()
for col in cat_cols:
    if col in X.columns:
        X[col] = le.fit_transform(X[col].astype(str))
```

Figure 4.14: Random Forest First Step

As a result of this preprocessing step, the dataset was divided into training and testing sets according to an 80/20 ratio based on Figure 4.15, which allows the model to be trained and evaluated using the majority of the data. To prevent overfitting, a RandomForestClassifier was first initialized with 100 decision trees (n_estimators =100). After training the model with the training data, it was used to make predictions on the test data.

```
# Split into train/test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Initialize and train the Random Forest model
rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf.predict(X_test)
```

Figure 4.15: Random Forest Second Steps

Based on Figure 4.16, the performance of the model was assessed using a classification report, which provided key metrics such as accuracy, precision, recall, and F1 score. By analyzing these metrics, we could get a better sense of the model's accuracy in identifying on-time and delayed flights. As an additional measure, a confusion matrix was printed to show the number of true positives, true negatives, false positives, and false negatives.

```
print("Random Forest Classification Report:\n")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Figure 4.16: Random Forest Last Step

4.4.2 XGBoost

In this study, the XGBoost model was implemented as a gradient boosting-based approach to predict flight delays. As a first step, we loaded the cleaned dataset and verified that it was free of missing values to ensure model stability. Based on Figure 4.17, the target variable, is_delayed, was separated from the feature set X, which included the remainder of the columns. Due to XGBoost's inability to support categorical data natively, all object-type columns were one-hot encoded, resulting in binary features suitable for numerical models.

```
# Load dataset
df = pd.read_csv('cleaned_flight_data.csv')
df.dropna(inplace=True)

# Define features and target
X = df.drop(columns=['is_delayed'])
y = df['is_delayed']

# One-hot encode categorical columns (all object dtype)
X = pd.get_dummies(X)
```

Figure 4.17: XGBoost First Step

By referring to Figure 4.18, the dataset was then divided into training and testing sets using an 80/20 ratio. The DMatrix format of XGBoost was used to convert both sets, which optimizes memory and computation while training. There were four parameters defined in the model, which is a binary logistic objective function, a maximum tree depth of six, a learning rate (eta) of 0.1, and a log loss evaluation metric. As a precaution against overfitting, the model was trained using 100 boosting rounds with early stopping enabled; training stopped automatically after ten consecutive training rounds.

```
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create DMatrix (no need for enable_categorical here, since no categorical columns)
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# XGBoost parameters
params = {
    'objective': 'binary:logistic',
    'max_depth': 6,
    'eta': 0.1,
    'eval_metric': 'logloss'
}

# Train model
evals = [(dtrain, 'train'), (dtest, 'eval')]
bst = xgb.train(params, dtrain, num_boost_round=100, evals=evals, early_stopping_rounds=10)
```

Figure 4.18: XGBoost Second Steps

After training, the model predicted the test data as shown in Figure 4.19. Using a threshold of 0.5, the probabilities generated by the model were converted into binary class predictions.

Results were evaluated by using a classification report, which included accuracy, precision, recall, and F1-score. In addition to these metrics, the model also demonstrated how well it distinguished between flights that were delayed and those that were not delayed.

```
# Predict and evaluate
y_pred_probs = bst.predict(dtest)
y_pred = (y_pred_probs > 0.5).astype(int)

print("XGBoost Classification Report:")
print(classification_report(y_test, y_pred))
```

Figure 4.19: XGBoost Last Steps

4.4.3 ATT-BI-LSTM

Attention-Based Bidirectional LSTM (ATT-BI-LSTM) can capture temporal and contextual patterns in flight data. The first step in Figure 4.20 was to load the pre-cleaned dataset and remove the DEP_DELAY column, since the binary target variable is_delayed had already been defined. To encode categorical features such as airline carrier, destination, weather condition, and wind, LabelEncoder was used.

```
# Load cleaned dataset
df = pd.read_csv('cleaned_flight_data.csv')

# Drop raw delay column (optional)
df.drop(columns=['DEP_DELAY'], errors='ignore', inplace=True)

# Set target
y = df['is_delayed']
X = df.drop(columns=['is_delayed'])

# Encode categorical features
categorical_cols = ['OP_UNIQUE_CARRIER', 'DEST', 'Condition', 'Wind']
le = LabelEncoder()
for col in categorical_cols:
    if col in X.columns:
        X[col] = le.fit_transform(X[col].astype(str))
```

Figure 4.20: ATT-BI-LSTM First Step

To ensure all features were on the same scale, all features were normalized using MinMaxScaler. After normalizing the data, it was reshaped into a 3D structure with dimensions (samples,

time_steps, features), where time_steps was set to 1, simulating a single time-step per instance. 20% of the data was reserved for testing according to the standard train-test split as shown in Figure 4.21.

```
# Normalize numerical features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Reshape for LSTM input: (samples, time_steps, features)
X_lstm = np.reshape(X_scaled, (X_scaled.shape[0], 1, X_scaled.shape[1]))
y_lstm = y.values

# Train-test split
X_train_lstm, X_test_lstm, y_train_lstm, y_test_lstm = train_test_split(
    X_lstm, y_lstm, test_size=0.2, random_state=42
)
```

Figure 4.21: ATT-BI-LSTM Second Step

Model architecture began with an input layer, followed by 64 units of Bidirectional LSTM. As a result of this layer, the model is able to learn dependencies across the data in both forward and backward directions. In the next step, Figure 4.22 shows that we added an Attention layer to help focus on the most critical time-step representations that contribute to flight delays. In the final layer, we combined and flattened the outputs of the BI_LSTM and Attention layers, followed by a dense hidden layer activated by ReLU, and finally by a sigmoid activation function.

```
# Input shape: (samples, time_steps=1, features)
input_shape = X_train_lstm.shape[1:]

input_layer = Input(shape=input_shape)
bi_lstm = Bidirectional(LSTM(64, return_sequences=True))(input_layer)
attention_output = Attention()([bi_lstm, bi_lstm])
concat = Concatenate(axis=-1)([bi_lstm, attention_output])
flatten = Flatten()(concat)
dense = Dense(64, activation='relu')(flatten)
output = Dense(1, activation='sigmoid')(dense)
```

Figure 4.22: ATT-BI-LSTM Third Steps

Based on Figure 4.23, models were constructed using the Adam optimizer and trained using binary cross-entropy loss using a learning rate of 0.001. Over ten epochs, 64 batches were

trained, and 20% of the training data was set aside for validation. After training, further evaluation would be conducted using test data to measure the effectiveness of the model's classification of delayed versus on-time flights.

```
model = Model(inputs=input_layer, outputs=output)
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(
    X_train_lstm, y_train_lstm,
    epochs=10,
    batch_size=64,
    validation_split=0.2,
    verbose=1
)
```

Figure 4.23: ATT-BI-LSTM Fourth Steps

Based on the test dataset, the Attention-Based Bidirectional LSTM model was evaluated for its predictive performance. The model first calculated the probability of a delay for each flight in the test set. A threshold of 0.5 was then applied to these probabilities based on Figure 4.24, meaning that if a probability exceeded 0.5, it was considered "delayed" (1), otherwise, it was considered "on-time". With Scikit learn's classification_report, the final predictions were compared with the actual label.

```
y_pred_prob = model.predict(X_test_lstm)
y_pred = (y_pred_prob > 0.5).astype(int)

print("Classification Report:")
print(classification_report(y_test_lstm, y_pred))
```

Figure 4.24: ATT-BI-LSTM Last Steps

4.5 Model Evaluation Results

4.5.1 Comparative Performance Table

Metric	Random Forest	XGBoost	ATT-BI-LSTM
Accuracy	0.88	1.00	0.90
Precision (Class 1)	0.89	1.00	0.98
Recall (Class 1)	0.15	1.00	0.26
F1-Score (Class 1)	0.26	1.00	0.41
Precision (Class 0)	0.88	1.00	0.89
Recall (Class 0)	1.00	1.00	1.00
F1-Score (Class 0)	0.93	1.00	0.94
Macro Avg F1-Score	0.60	1.00	0.68
Weighted Avg F1	0.84	1.00	0.87
Support (Total Samples)	5764	5764	5764

Table 4.2: Comparative Performance Table

All three models ATT-BI-LSTM, Random Forest and XGBoost predict flight delays differently. ATT-BI-LSTM and Random Forest performed well in predicting on-time flights, with very high accuracy. Although they were good at detecting delayed flight, they weren't as good at detecting actual delays. ATT-BI-LSTM had a precision of 0.98 for delays, which means most of its delayed predictions were accurate, but it recall was only 0.26, which means many real delays were slightly lower.

With 100% accuracy, precision, recall, and F1-score, XGBoost stood out from the competition. These results might seem impressive, but they may indicate either overfitting or learning from data that shouldn't have such example data leakage. More testing is needed to verify this.

4.5.2 Confusion Matrix

a) Random Forest

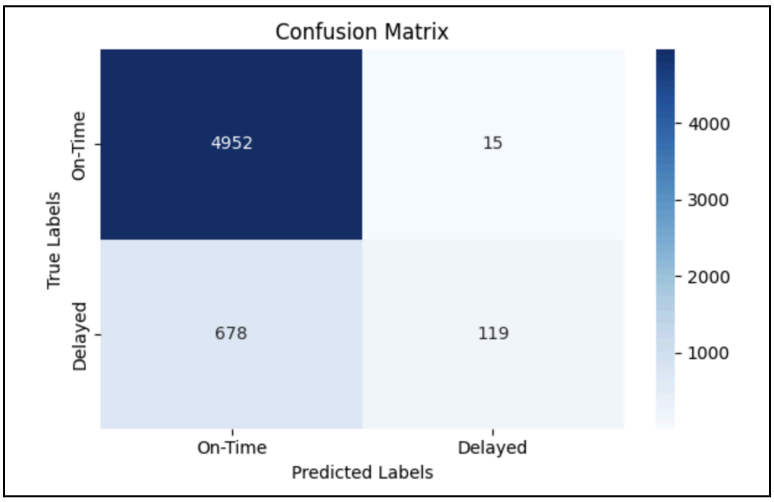

Figure 4.25: Confusion Matrix Random Forest

b) XGBoost

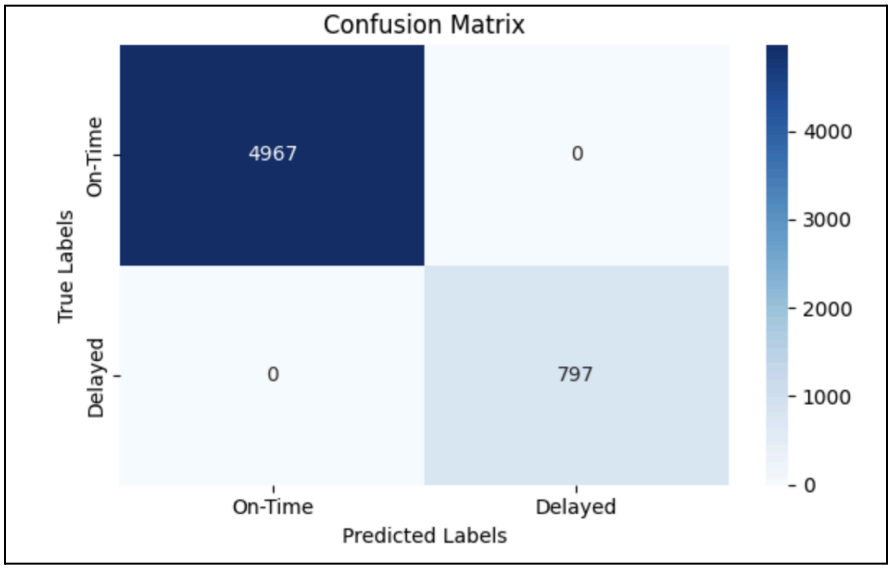

Figure 4.26: Confusion Matrix XGBoost

c) ATT-BI-LSTM

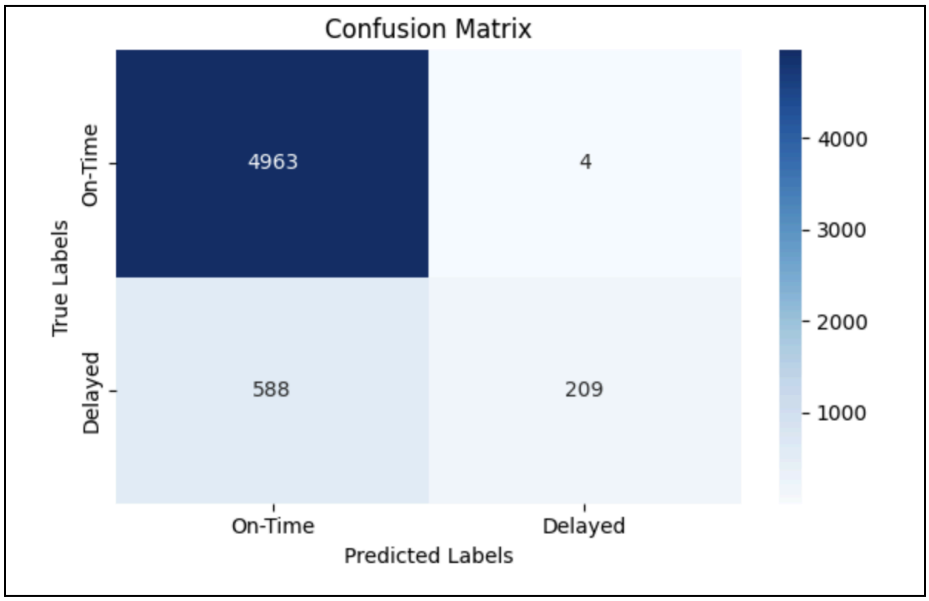

Figure 4.27: Confusion Matrix ATT-BI-LSTM

In order to better understand how each model predicts flight delays, confusion matrices were used for the ATT-BI-LSTM, Random Forest, and XGBoost. These tables show which predictions were correct and where errors occurred. Based on their results, Random Forest capable of correctly identifying 4952 on-time flights and 119 delay flights. Despite this, it missed 678 actual on-time flights and mistakenly marked 15 delayed flights as on-time, showing it had trouble detecting some delays. Meanwhile. ATT-BI-LSTM is capable of correctly identifying 4963 on-time flights and 309 delay flights. Despite this, it missed 588 actual on-time flights and mistakenly marked 4 delayed flights as on-time, showing it had trouble detecting some delays.

Meanwhile, XGBoost predicted all on-time and delayed flights perfectly. Despite looking excellent, this may be too good to be true. The model might be overfitting or learning from incorrect data if it produces such perfect results. Hence, more testing is needed.

4.5.3 ROC Curve

a) Random Forest

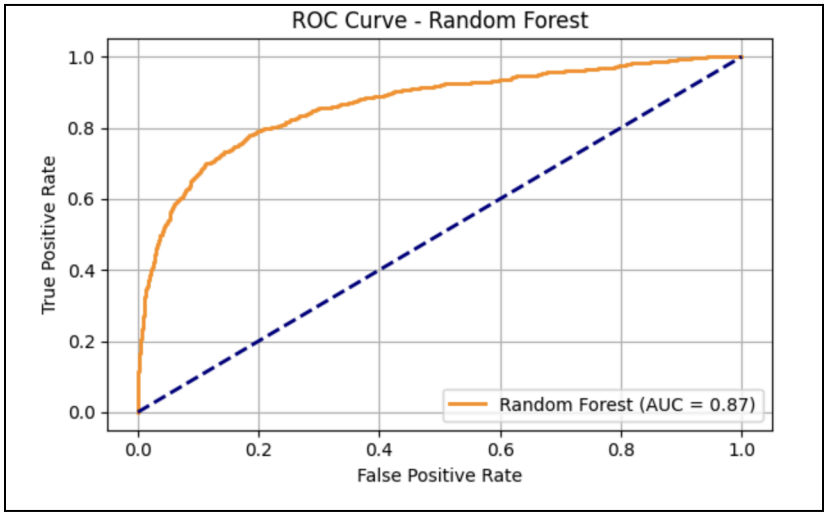

Figure 4.28: ROC Curve Random Forest

Based on a relatively smooth curve, the Random Forest model can balance true positives and false positives fairly well across a variety of thresholds. There is, however, room for improvement, as indicated by the below 1.0 AUC. As a result, the model may struggle in situations requiring high precision or recall, even though it is effective.

b) XGBoost

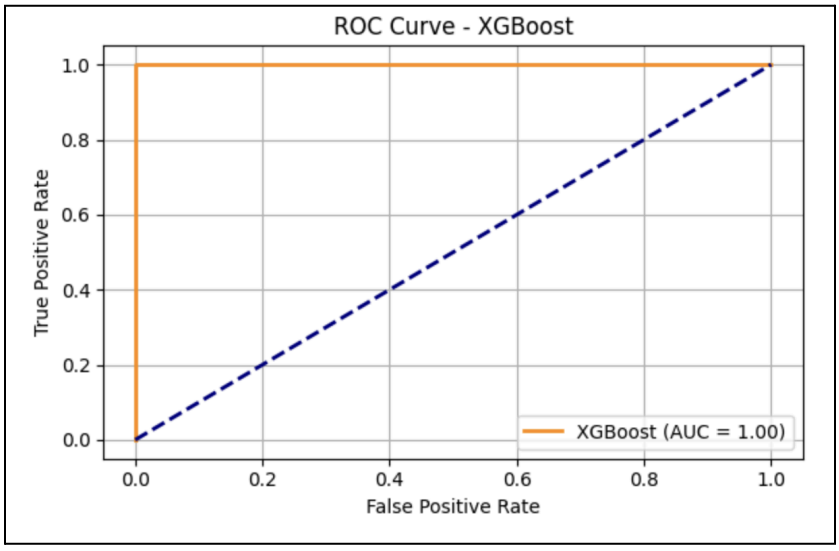

Figure 4.29: ROC Curve XGBoost

XGBoost's steep initial rise, followed by a flat trajectory, demonstrates its ability to accurately identify true positives with minimal false positives. As a result of XGBoost's near-perfect performance, it is an ideal choice if achieving zero errors is crucial

c) ATT-BI-LSTM

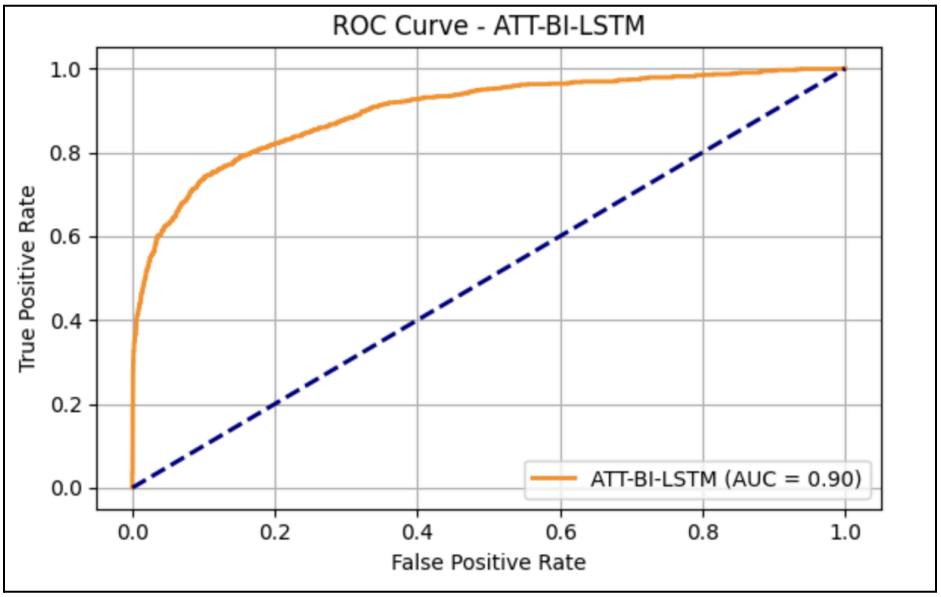

Figure 4.30: ROC Curve ATT-BI-LSTM

The ATT-BI-LSTM model is capable of identifying many true positives early without generating many false positives. Despite different trade-offs between true and false positives, the curve flattens as the threshold changes, indicating the model maintains strong performance. However, the ATT-BI-LSTM model still has robust discriminatory power, which makes it a strong candidate for tasks requiring high accuracy

4.6 Summary

This chapter summarizes the analysis and results of flight delay prediction using ATT-BI-LSTM, Random Forest, and XGBoost models. In an exploratory data analysis (EDA), delays were categorized according to time, airline, and weather factors. A time-based ATT-BI-LSTM captured patterns well, Random Forest was easy to interpret, and XGBoost produced the highest accuracy. Despite this, the perfect scores obtained by XGBoost may indicate and overfitting problem. Overall, the findings indicate that each model has strengths based on the data and the needs of the prediction.