EECS 3401

Final Report

Flight-Status Dataset

Group 20

Raj Panjabi (217847328)

Siddhrajsinh Parmar (219091321)

Aum Patel (218153338)

Nishiket Singh (216521197)

Problem Description

The aim of our project was to delve into a dataset concerning flight statuses to identify patterns, trends, and ultimately, to predict flight delays. The main objective of this project is to predict flight delays (more than 15 minutes) upon arrival using machine learning algorithms. Given the complexity and the dynamic nature of flight operations, this problem is both challenging and relevant to the airline industry and its consumers. Through EDA, data preprocessing, and machine learning, we sought to predict whether a flight would be delayed. Forecasting delays enable us to mitigate the adverse effects on scheduling, costs, and passenger satisfaction.

Through highlighting the significance of accurately predicting flight delays for various stakeholders, including airlines, passengers, and airport operators. Our predictions can facilitate better decision-making and operational efficiency.

Dataset (Flight Status Prediction (kaggle.com))

This dataset comprises detailed records of flights from 2018 to 2022, aimed at facilitating comprehensive analysis of flight delays and operational patterns. It includes a wide range of attributes from basic flight details to more specific data on delays, cancellations, and diversions. We are using the dataset for the year of 2022.

The Dataset includes :

Temporal Details: Year, Quarter, Month, DayOfMonth, DayOfWeek, and FlightDate, providing complete date-time information for each flight.

Airline and Flight Information: Including Marketing_Airline_Network, Operated_or_Branded_Code_Share_Partners, various unique carrier codes (DOT_ID, IATA Code), and Flight Numbers.

Destination: Details such as OriginAirportID, DestAirportID, along with city, state, and world area codes (WAC).

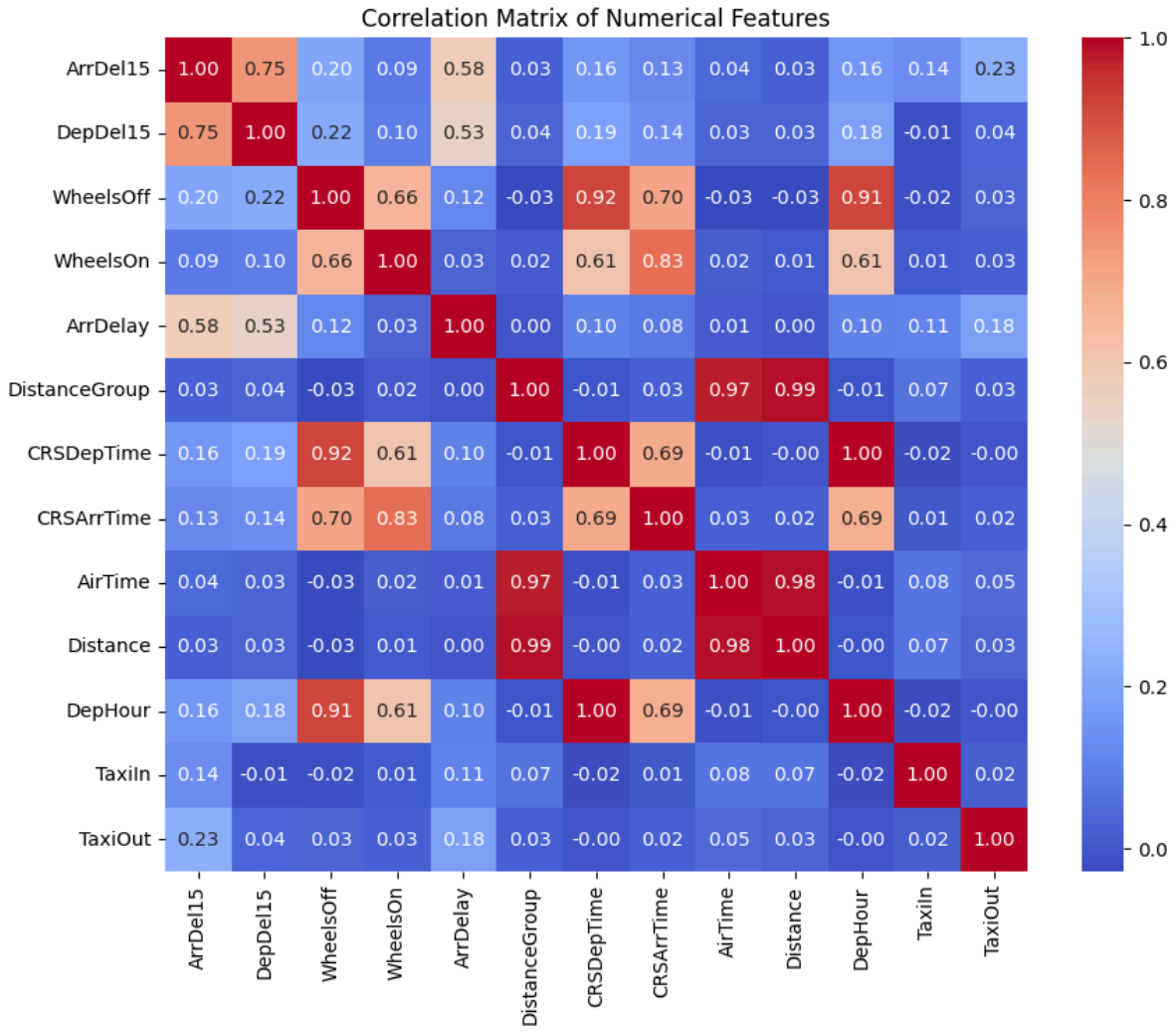
Timing and Delays: Scheduled (CRS) and actual times for departure (DepTime) and arrival (ArrTime), delay durations (DepDelay, ArrDelay), and specific reasons for delays.

Flight Status: Including indicators for cancellations (Canceled, CancellationCode) and diversions (Diverted, DivAirportLandings).

Operational Metrics: Taxi times (TaxiOut, TaxiIn), elapsed flight times (CRSElapsedTime, ActualElapsedTime), and distances covered.

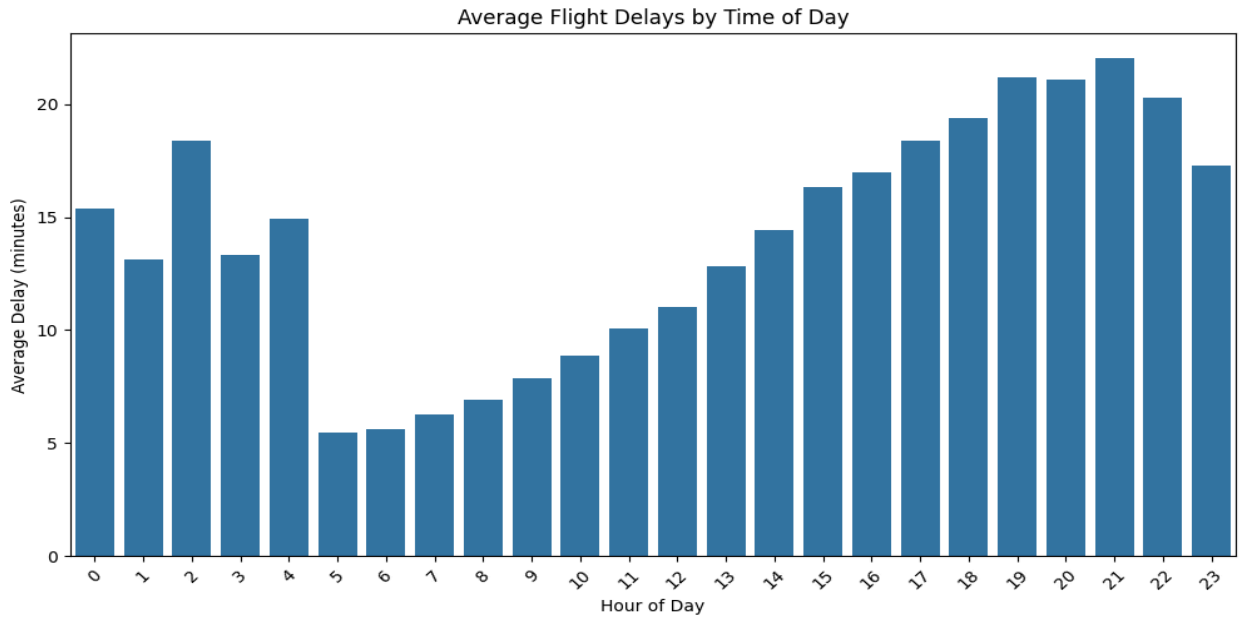
EDA

Correlation Matrix



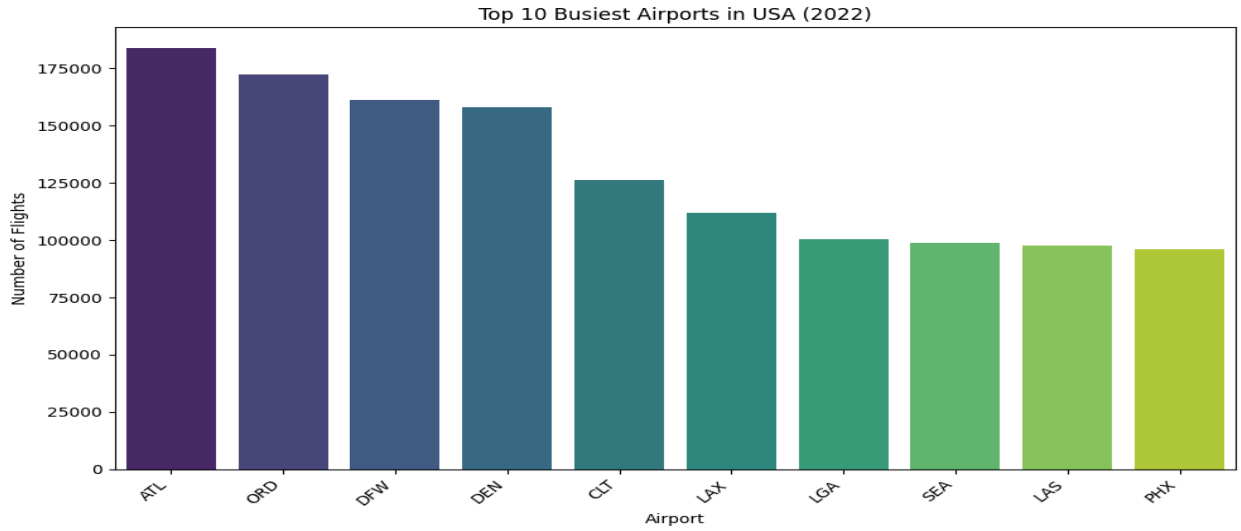
- Arrival and departure delays (ArrDel15 and DepDel15) have a strong positive correlation, indicating that when a flight departs late, it is also likely to arrive late.
- There is a significant positive correlation between CRSDepTime and DepHour, as well as CRSArrTime and WheelsOn, which is expected since these are directly related to the scheduled times and actual times of flights.

Average Flight Delays by Time of Day



From the above bar chart graph it is evident that the flights tend to experience higher average delays later in the day, with delays gradually increasing from the early afternoon hours and peaking in the evening. Early morning flights show lower average delays, suggesting that flying earlier might reduce the likelihood of encountering delays.

Top 10 Busiest Airport in USA (2022)



ATL (Atlanta) is the USA's busiest airport, followed by ORD (Chicago O'Hare), DFW (Dallas/Fort Worth), and DEN (Denver), highlighting their key positions in the air travel network. The flight numbers show a tiered airport busyness, with a notable decline after the top few. LAX (Los Angeles) and SEA (Seattle) are major international gateways, while LGA (LaGuardia) and LAS (Las Vegas), despite their high traffic, have more regional significance compared to the national prominence of the leading airports.

Data Cleaning and Preprocessing

- Only using data from the busiest airport

  We printed the airports from which the most flights flew and selected only that airport(ATL) for our analysis since our models wouldn't run on the whole dataset due to lack of computing power. Initially we decided to run the model on the 5 busiest airports but unfortunately we weren't able to run the model due to the sheer size of the data even while using google colab on the best non premium runtime.

- Selecting relevant columns

  We dropped the unnecessary columns that represented unique ID codes as they are unrelated to the data analysis. We also dropped the origin column since we are only doing analysis of flights from ATL. The selected columns are crucial for predicting flight delays and understanding the factors influencing arrival delays. For example, columns selected are TaxiIn, TaxiOut, and Airtime columns contribute to overall flight duration, DepDel15 tells us if the flight was delayed at departure, DepTime, WheelsOff, and WheelsOn capture key events during a flight, CRSDepTime and CRSArrTime are the scheduled departure and arrival times, serving as reference points for delay calculations and there few other features as well. Lastly, we also created a new column 'DepHour' which enables us to get insights about average flight delays on an hourly basis.

- Handling columns with value in 'hhmm' format

  We converted the columns in 'hhmm' format into 2 columns, one being column_sin and column_cos. We converted 'hhmm' to hours and minutes, which we then normalized to range [0,1), which was then converted to [0,2pi). This helped us create the sine and cosine transformations which encode the time information into two continuous features that capture both the cyclicality and the phase (shift) within the day. The resulting features (column_sin and column_cos) are used as inputs to our machine learning model for predicting flight cancellations.

- Handling missing values in the dataset

For starters we checked for duplicate values and dropped them as they would not provide any extra information and unnecessarily increase size of the selected dataset.The missing numerical values were replaced by the mean of all the other present values, while the missing categorical values were replaced by the most frequent values. Initially the data set was segregated into numerical and categorical columns to apply distinct pre-processing techniques.

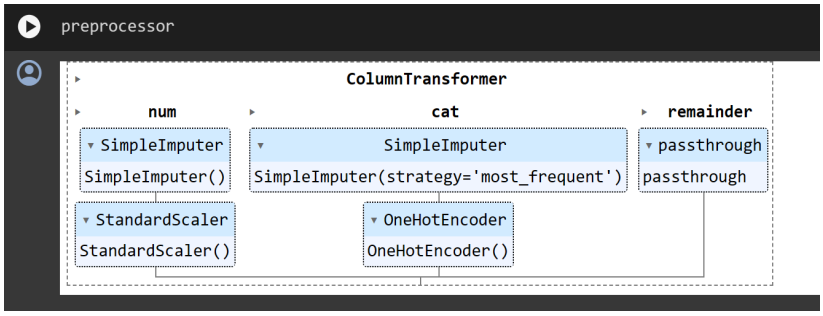
- Preprocessing Pipeline

  We started by excluding columns ('DepDel15' and 'ArrDel15') from the numerical columns list. Subsequently two separate pipelines are created, one for processing numerical data and another one for categorical data, each performing specific pre-processing techniques. Numerical data was imputed with mean values and standardization, while categorical data were filled with most frequent values and applied one-hot encoding to those columns.

  'ColumnTransformer' was used to fit the preprocessing steps across the dataset. The column header was reinstated to preserve the original dataset after the step of applying preprocessing pipeline.

  The missing values in the Target columns were dealt by dropping them to ensure data integrity for model training. Thus, after applying all those steps the dataset was ready for model training.

  Pipeline



Machine Learning Algorithms:

We picked 3 classification models for our project: Random Forest Classification, Logistic Regression and KNN. We split the dataset into 80:20 for training and testing respectively

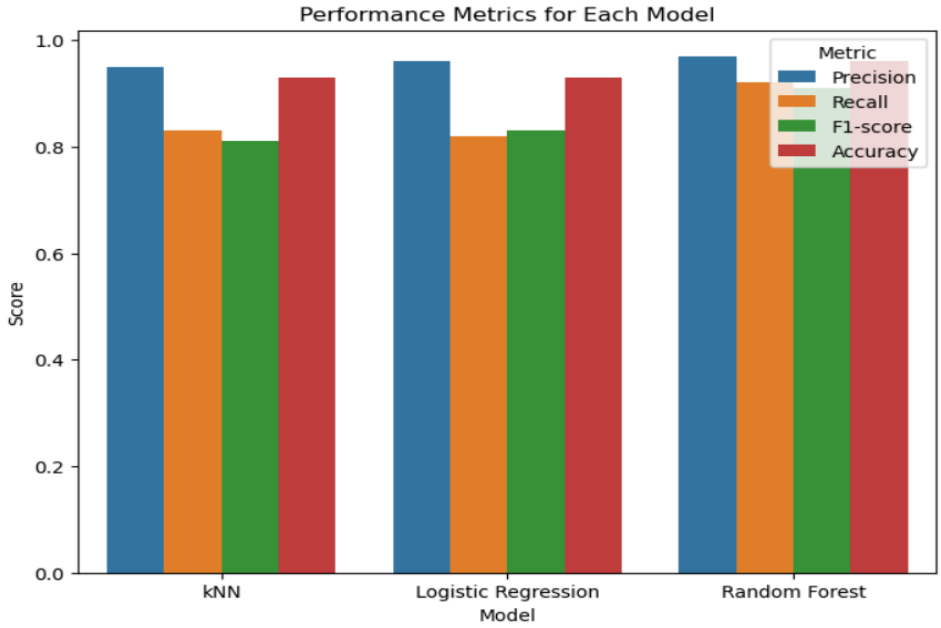
So we use the Random Forest model for classification . Random Forest grows multiple decision trees which are merged together for a greater accurate prediction.

The logic behind the Random Forest model is that multiple individual decision trees perform much better as a group than they do alone. When using Random Forest for classification, each tree gives a classification or a “vote”, so in our case the arrival flights are delayed by 15 minutes or more, the “vote” will be yes or no . The forest chooses the classification with the majority of the “votes.” When using Random Forest for regression, the forest picks the average of the outputs of all trees. As it is dependent on every vote we use random forest to apply overfitting to our training set.

Logistic regression models the probability that the "AirDel15" variable belongs to a particular category (categorical type). It does this by fitting the data to a logistic function, which is an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1 uses. Then these probabilities are used to classify each observation. If the predicted probability is greater than 0.5, the model predicts that the observation belongs to one class. If it’s less than 0.5, it predicts that the observation belongs to the other class. This threshold of 0.5 is called the decision boundary.

Unlike other learning algorithms, k-NN doesn’t create a model using the training data. Instead, it uses the training instances themselves in the prediction phase. This is why it’s called an instance-based learning algorithm.When a prediction is needed for an unseen data instance, the k-NN algorithm will search through the training dataset for the k-most similar instances. The similarity of instances is determined based on a distance measure, such as Euclidean distance .Once the k-most similar instances have been identified, the algorithm uses them to make a prediction. For classification problems, this is typically done by majority voting.

Comparing F1 score, Recall, Precision and Accuracy

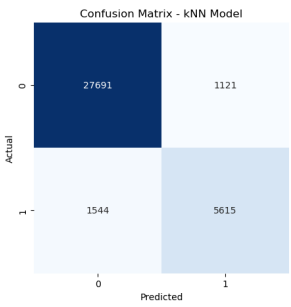


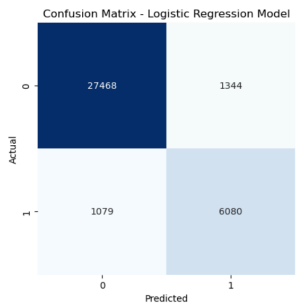
Results

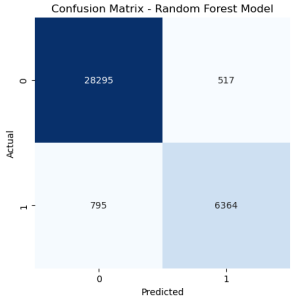
Classification report

```
Classification Report - kNN Model:
              precision    recall  f1-score   support

           0       0.95      0.96      0.95     28812
           1       0.83      0.78      0.81      7159

    accuracy                           0.93     35971
   macro avg       0.89      0.87      0.88     35971
weighted avg       0.92      0.93      0.92     35971


Classification Report - Logistic Regression Model:
              precision    recall  f1-score   support

           0       0.96      0.95      0.96     28812
           1       0.82      0.85      0.83      7159

    accuracy                           0.93     35971
   macro avg       0.89      0.90      0.90     35971
weighted avg       0.93      0.93      0.93     35971


Classification Report - Random Forest Model:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     28812
           1       0.92      0.89      0.91      7159

    accuracy                           0.96     35971
   macro avg       0.95      0.94      0.94     35971
weighted avg       0.96      0.96      0.96     35971
```

Confusion Matrix







Analyzing the classification reports for all three models

k-Nearest Neighbors (kNN) Model:The overall accuracy of the model is 93%, which means it correctly predicts whether an arrival flight is delayed or not 93% of the time. The precision for predicting an arrival flight is delayed is 0.83.

The recall is 0.78, which means the model correctly identified 78% of all delayed flights.

Logistic Regression Model:

The overall accuracy of the model is also 93%, the same as the kNN model.

However, the precision for an arrival flight being delayed is slightly lower than the kNN model at 0.82, but the recall is higher at 0.85. This means the Logistic Regression model is slightly more likely to miss delayed flights but also less likely to predict a delay when there isn't one.

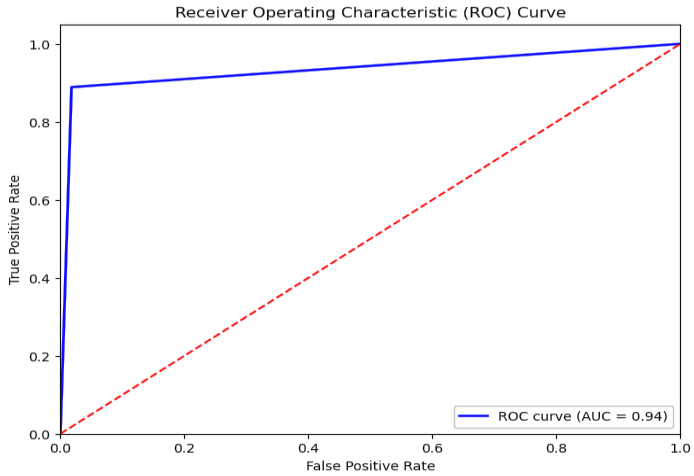
Random Forest Model:

The Random Forest model has the highest overall accuracy at 96%. The precision and recall for arrival flight being delayed are also the highest among the three models at 0.92 and 0.89, respectively. This means the Random Forest model is the most accurate at predicting flight delays and also the least likely to miss a delay or predict a delay when there isn't one.

In summary, while all three models perform well, the Random Forest model has the best performance in terms of accuracy, precision, and recall. However, it's important to consider other factors such as the training time, interpretability, and the specific costs of false positives and false negatives when choosing a model. For example, if interpretability is important, a logistic regression model might be preferred as it provides coefficients for each feature which can be interpreted as the effect of that feature on the odds of a delay. On the other hand, if the cost of missing a delay (false negative) is high, a model with a higher recall like the Random Forest might be preferred.

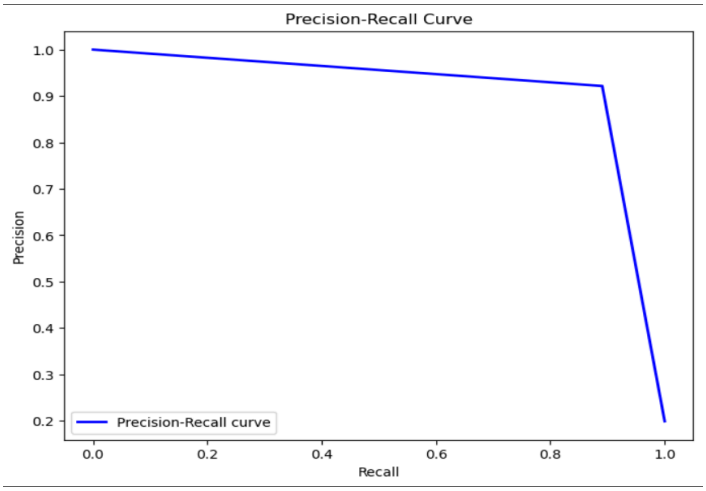
This ROC curve suggests that the binary classification model has excellent performance in distinguishing between the two classes: flights that are delayed by 15 minutes or more, and flights that are not. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test. Here, the curve is closer to the top left corner, which indicates a good measure of separability and a good model.

ROC curve



This Precision-Recall curve indicates that the model has perfect precision (no false positives) with increasing recall up to a certain point, after which precision drops. This suggests that for a certain range of thresholds, the model is able to identify all positive instances correctly. However, beyond this range, the model starts to misclassify some negative instances as positive, which reduces precision. This could be due to class imbalance or other factors.

Precision-Recall curve



Tuning

We applied RandomizedSearchCV technique for fine tuning hyperparameters. This technique systematically searches for the best hyperparameters, taking care of the risk of overfitting. The parameter grid defines some important hyperparameter change that will tune the Random Forest Classifier. So, it explores the hyperparameter space through randomized sampling. The accuracy we get after tuning is 0.96 which is similar to our first run. There could be few conclusions that we can think of like the model is already optimal, there is limited exploration of hyperparameter space.

Limitation

The biggest challenge we faced was the lack of computing power available to our group, since we couldn't run any models with all of the dataset and had to create a subset of the busiest airports and even then, it took a long time to run the models. There were an exorbitantly high number of columns and selecting useful features became time consuming and difficult.

Next Steps and Conclusion

1. There is potential for exploring additional features or engineering techniques that could enhance the predictive power of the model. This could involve incorporating external datasets, exploring more sophisticated time-series features, or leveraging advanced feature selection methods.

2. It could be beneficial to investigate ensemble methods such as stacking or boosting. These methods combine multiple models to improve overall performance and robustness.

3. We believe in the future some consideration should be given to the implementation mechanisms for seamless deployment and integration of the model into operational workflows or decision-support systems. This may involve building APIs, developing user interfaces, or integrating with existing business intelligence tools.

4. Overall we really liked doing this project , and would like to thank you for taking the time to go through our project report.

Appendix 1

Dataset: [Flight Status Prediction (kaggle.com)]

Github link: https://github.com/patel-aum-yorku/Flight-Status-Prediction

Youtube Video: https://www.youtube.com/watch?v=aJBHVN3gEB4

Appendix 2 source code:

```
time_columns = ['CRSDepTime','DepTime','CRSArrTime','WheelsOff',  'WheelsOn']

for column in time_columns:

  # Extract hour and minute

    df_filtered[column + '_hour'] = df_filtered[column] // 100

    df_filtered[column + '_minute'] = df_filtered[column] % 100

  # Apply trigonometric transformations

    df_filtered[column + '_sin'] = np.sin(2 * np.pi * (df_filtered[column + '_hour'] * 60 +
df_filtered[column + '_minute']) / (24 * 60))

    df_filtered[column + '_cos'] = np.cos(2 * np.pi * (df_filtered[column + '_hour'] * 60 +
df_filtered[column + '_minute']) / (24 * 60))

    # Drop the original columns

    df_filtered = df_filtered.drop(columns=[column, column + '_hour', column + '_minute'])

# Print data types of the updated DataFrame

print(df_filtered.dtypes)
```