## 1 Introduction

Flight Delays is an essential subject in the context of air transportation systems. It brings negative impacts to passengers, airlines and airports. In order to prevent any loss caused by the uncertainty of flights, passengers usually choose the flights which are a day ahead of their appointments. That increases their trip costs and wastes their time. On the other hand, airlines and airports also suffer penalties, fines and additional operation costs, such as crew and aircrafts retention in airports. Therefore, the estimation of flight delays can help airports and airlines make better tactical and operational decisions and also can help passengers arrange their plans more precisely.

## 2 Flight Delays Dataset

### 2. 1 Data Source

In this project, I use the `Combined_Flights_2022.csv` provided by Rob Mulla from Kaggle. This dataset contains all flight information including cancellation and delays by airlines for 2022. The data was extracted from the Marketing Carrier On-Time Performance data table of the “On-Time” database from the TranStats data library.

### 2.2 Data Detail

The dataset contains 1.42 GB raw data with totally 61 features and 4078318 instances. There are 5 feature types: Boolean, Floats, Integer, Object. In the dataset, there are 133402 rows that have missing values or null.

### 2.3 Task and Expected output

My program is designed to train a DecisionTree model for predicting whether the flight will arrive at the destination over 15 minutes late. After running the program on the Hadoop, you are expected to receive a HDFS output file with training accuracy (decimal < 1), test accuracy(decimal < 1) and running time (seconds). The evaluated metrics are based on the number of instances that get the same value of their “ArrDel15”(0=No, 1=Yes) as the actual label values.

### 2.4 Data Preprocess

I applied 4 preprocessing steps to the raw data (DataFrame) before putting it in the program.

- Data Cleaning: Since I learned the existence of missing values or null during data exploration, the first step is to clean the data by dropping the rows with missing values or null.
- Drop inapplicable columns from the DataFrame:
    - Redundant/duplicate columns: FlightDate, Year, State/City names, Airline names.
    - Leakage columns: features record actual arrival/delay time and real time events .
    - Irrelevant columns for delay predictions: features for cancellation prediction like “Cancelled”, “Diverted”.

This step needs precise exploratory data analysis and enough domain knowledge. I use correlation matrix as reference to decide which columns should drop. Also, after knowing every features’ description, I learnt that there are many duplicate features in the dataset, which represent the same meaning. For example, in the raw data, “Dest”, “DeatAirportID” and “DestAirpotSeqID” all represent destination airports. However, in the correlation matrix, all three correlate with the target in the same range. Therefore, I filtered out two of them to improve the running speed and reduce the risk of noise.

- Handle categorical features: I converted string features to numeric indices using `StringIndexer`.

- Assemble all features: After previous steps, the dataset is ready to be used. Before putting into the program, I assembled indexed categorical columns and numeric columns into a single vector column, as the algorithms required, using VectorAssembler.

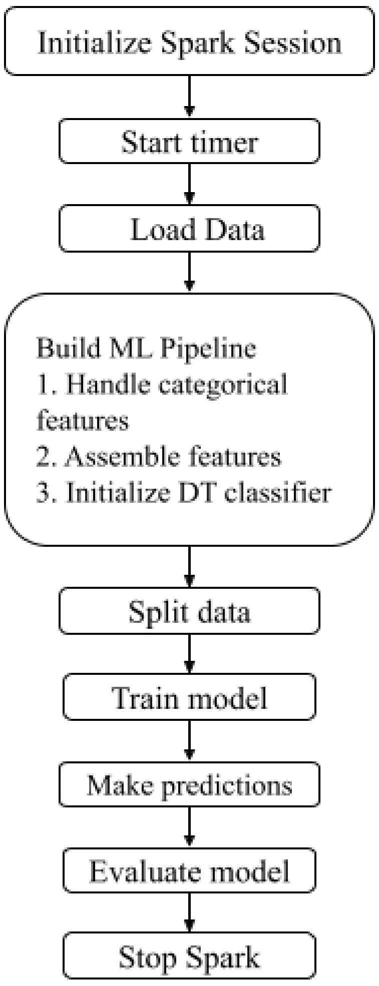
## 3 Program Description

### 3.1 Pseudo-code

```
FUNCTION Build ML Pipeline(df):

    StringIndexer categorical features

    VectorAssembler categorical features + numeric features

    Initialize DecisionTreeClassifier with feature vector as
    input and "ArrDel15" as label

    Pipeline(indexer + [assembler, DecisionTreeClassifier])

FUNCTION main():
    Initialize Spark session and read the data from HDFS input
path
    Record the start_time
    Create DataFrame by reading csv file from hdfs data path with
infer schema and first row as header

    Data preprocess on raw data:
      ● Drop rows with missing(null) values
      ● Drop redundant, irrelevant and leakage-prone columns

    Call preset function to build pipeline with cleaned df

    Split cleaned df into train_data (80%) and test_data (20%)
using seed

    Train the pipeline on train_data

    Use the trained model to predict on both train_data and
test_data

    Initialize an evaluator for accuracy

    Compute accuracy on train_predictions
    Compute accuracy on test_predictions

    Create an RDD with strings containing:
      - training accuracy
      - test accuracy
```

```
        - total running time

    Save the RDD to HDFS output path
    Stop the Spark session

IF script is run as main:
    Call main()
```

### 3.2 Flowchart



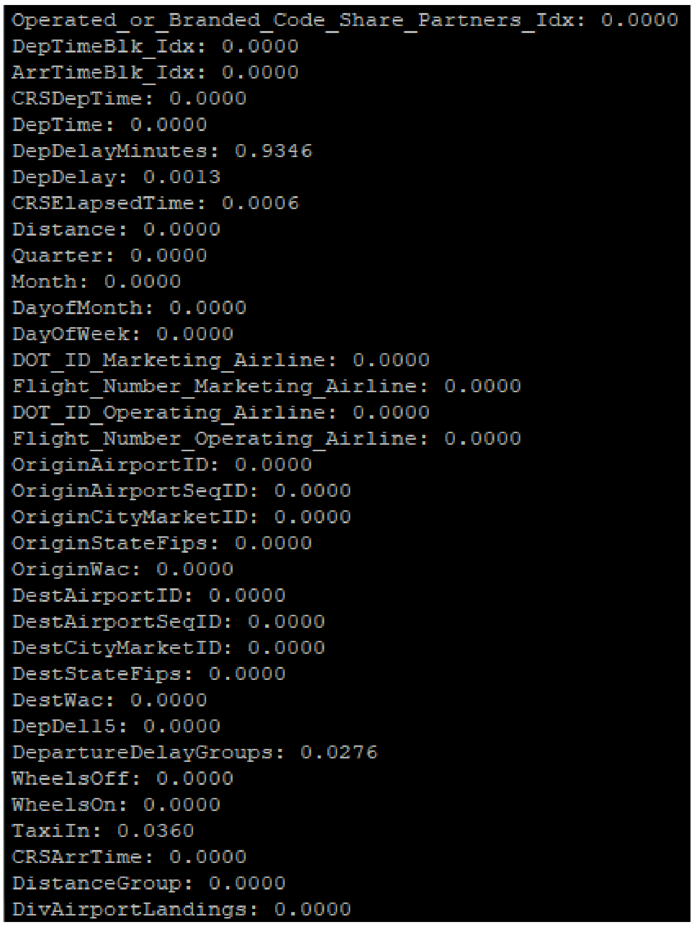
## 4 Comparison and Discussion

### 4.1 Normalization

| | **Without Normalization** | **With Normalization** |
|---|---|---|
| **Training Accuracy** | 0.9302 | 0.9302 |
| **Test Accuracy** | 0.9303 | 0.9303 |
| **Running Time** | 192.224 seconds | 202.098 seconds |

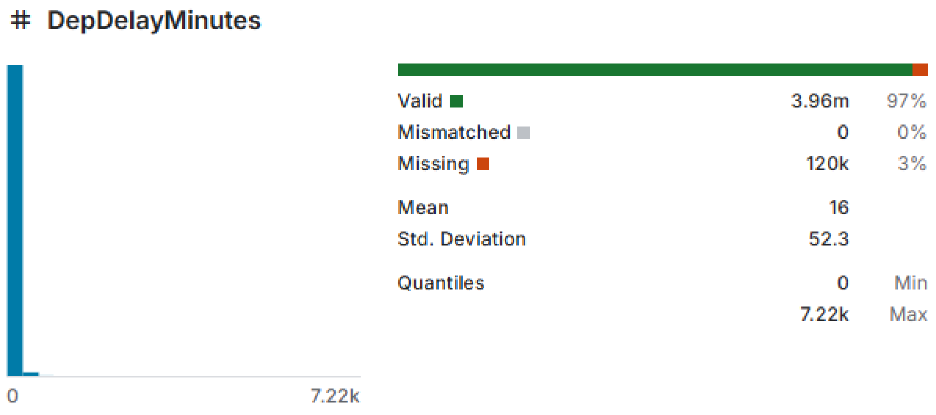
The table above records the results of two models: Decision Tree with normalization vs Decision Tree without normalization. Compared to the standard model, the model with normalization on the numerical features doesn't have any improvement on prediction performance. By contrast, it took longer to get the results. There are several reasons why normalization doesn't improve the performance:

1. Model characteristics
    a. Scale Invariance: Decision trees train the model based on threshold-based splitting. Even if the feature values are scaled, the relative rank of the data points still remains the same, and the model will still split on the same positions.
    b. Independence on feature magnitude: Decision trees won't be affected by the magnitude of feature values. They focus on identifying the most appropriate threshold for splitting of features, regardless of the scale of features.
2. Data characteristics
    a. Intrinsic data types: In our dataset, there are 4 types of data: float, int, boolean, object. Besides boolean, most of the features in other data types actually represent ID, Date or Time (format: hhmm). So, these features should be indexed as categorical features instead of normalized like numeric features.
    b. Outliers: I will use the feature with the highest importance rank below as reference. "DepDelayMinutes" is the most informative feature in model learning. It represents the difference in minutes between scheduled and actual departure time. Early departures set to 0.



```
Operated_or_Branded_Code_Share_Partners_Idx: 0.0000
DepTimeBlk_Idx: 0.0000
ArrTimeBlk_Idx: 0.0000
CRSDepTime: 0.0000
DepTime: 0.0000
DepDelayMinutes: 0.9346
DepDelay: 0.0013
CRSElapsedTime: 0.0006
Distance: 0.0000
Quarter: 0.0000
Month: 0.0000
DayofMonth: 0.0000
DayOfWeek: 0.0000
DOT_ID_Marketing_Airline: 0.0000
Flight_Number_Marketing_Airline: 0.0000
DOT_ID_Operating_Airline: 0.0000
Flight_Number_Operating_Airline: 0.0000
OriginAirportID: 0.0000
OriginAirportSeqID: 0.0000
OriginCityMarketID: 0.0000
OriginStateFips: 0.0000
OriginWac: 0.0000
DestAirportID: 0.0000
DestAirportSeqID: 0.0000
DestCityMarketID: 0.0000
DestStateFips: 0.0000
DestWac: 0.0000
DepDel15: 0.0000
DepartureDelayGroups: 0.0276
WheelsOff: 0.0000
WheelsOn: 0.0000
TaxiIn: 0.0360
CRSArrTime: 0.0000
DistanceGroup: 0.0000
DivAirportLandings: 0.0000
```

From the feature information, it’s obvious that there is the existence of outliers and the value of outliers is very extreme. Also, we can see that excluding the outlets, the value interval is actually not wide-ranged. Since normalization is very sensitive to outliers, the feature distribution may be distorted by strong outliers and that could hurt model performance.


# DepDelayMinutes

| | | |
|---|---|---|
| Valid 🟩 | 3.96m | 97% |
| Mismatched ▨ | 0 | 0% |
| Missing 🟥 | 120k | 3% |
| Mean | 16 | |
| Std. Deviation | 52.3 | |
| Quantiles | 0 | Min |
| | 7.22k | Max |

Let me explain how normalization distorts the feature distribution. Assume “DepDelayMinutes” with values mostly between 0-60 minutes, but a few flights have extreme delays of 1000 minutes.

- Without normalization: Trees may choose a meaningful split like <15 minutes.
- With normalization: Because outliers stretch the scale, most values are compressed very close to 0. Now, trees may choose unstable thresholds like < 0.145 minutes, which are hard to interpret.

In summary, this dataset doesn’t support Decision Tree to do normalization. Apply normalization to features with strong outliers could introduce noise rather than help.

## 4.2 PCA

| | **Without PCA** | **With PCA** |
|---|---|---|
| **Training Accuracy** | 0.9302 | 0.9140 |
| **Test Accuracy** | 0.9303 | 0.9138 |
| **Running Time** | 192.224 seconds | 415.7938 seconds |

The table above records the results of two models: Decision Tree with PCA vs Decision Tree without PCA. Compared to the standard model, the model with PCA apparently got lower accuracy and slower running time. There are several reasons why PCA hurts model performance:

1. PCA distorts feature interpretability: As the mandatory requirement, the program normalized all features before putting into PCA, including indexed categorical features and disguised features(ID, Date and Time). Then, PCA transformed scaled features into linear combinations(principal components). Until now, the original data has been double encrypted and Decision trees can’t find the intuitive and meaningful thresholds from abstract

components (e.g. PC1 = 0.8 * feature1 + 0.5 * feature2 + … - 0.3 * featureN) easily. That caused the reduction of accuracy and longer running time.

2. Loss of information: PCA is used to reduce the dimensionality. When you reduce the number of components (k=10), some information will also be thrown away. According to the feature importance rank in Section 4.1, we can see that the dataset only contains few informative features and some of informative features may contain very small predictive signal to the label. Therefore, under such an information lack situation, every discarded component may throw some information and the model performance will drop for sure.
3. Nonlinearity Mismatch: PCA converted the features into linear expressions. However, our dataset seems nonlinear based on its underlying data types. Therefore, PCA may remove some important interactions, which Decision trees are good at capturing, among the features.

In summary, although PCA may help in some models, the Decision tree model won't benefit from it.

## Citation

1. Mulla, R. (2022). *Flight Status Prediction (Flight Delay Dataset 2018–2022)* [Data set]. Kaggle. https://www.kaggle.com/datasets/robikscube/flight-delay-dataset-20182022
2. Arya, N. (2022, July 27). *Does the Random Forest Algorithm Need Normalization?* KDnuggets. https://www.kdnuggets.com/2022/07/random-forest-algorithm-need-normalization.html
3. Sternberg, A., De Abreu Soares, J., Carvalho, D., & Ogasawara, E. (2017, March). *A Review on Flight Delay Prediction* [Preprint]. ResearchGate. https://doi.org/10.48550/arXiv.1703.06118