## Team Members:

1. IMT2021007 -Sunny Kaushik
2. IMT2021087 –Shrey Salaria
3. IMT2021530 –Akash Perla

## Dataset Description and Analysis:

In this project, we are provided with a dataset related to a public bus transport management system. The dataset encompasses crucial information pertaining to bus operations, specifically focusing on predicting whether a bus will arrive on time. The dataset includes essential columns such as Index, Bus ID, Departure Time, Journey Time, Bus Operator, Departure Bus Stop, Arrival Bus Stop, and the day of travel. The objective of this project is to develop a predictive model capable of determining whether a bus is likely to be on time or not based on the provided features. The target variable for this prediction task is binary, where a value of 1 denotes that the bus will arrive on time, and a value of 0 indicates otherwise. To facilitate this predictive modeling task, we have been given two key datasets: the training dataset to train our models and the test dataset to evaluate their predictive performance.

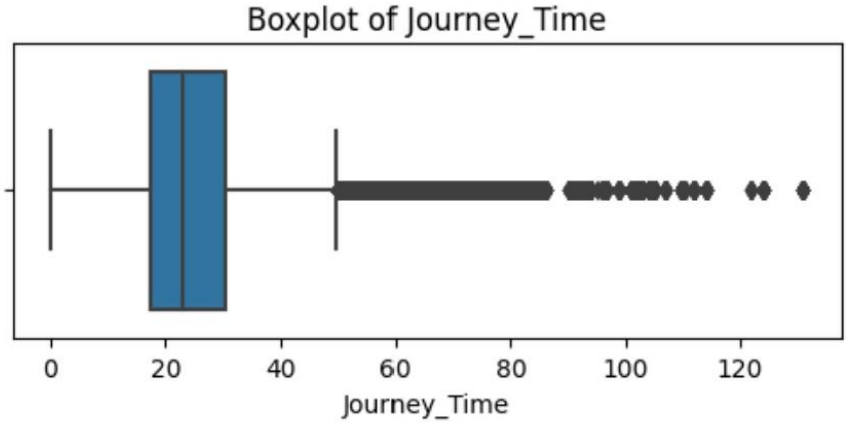
## Data Preprocessing and cleaning:

Firstly, we made sure that there are no duplicates in the given dataset and if any then removed them using the command: train_df.drop_duplicates(implace = true).

Then there were a lot of missing values to keep track of the missing values we ran: train_df.isna.sum(). ---> this command takes the training data(train_df) and returns the number of NAN values for each column respectively. By running this, we found that there are 69078 block of journey time with NAN values and 1258 blocks of days with NAN values, which needs to be handled.

We have used IterativeImputer to deal with NaN values. Iterative Imputer is based on an iterative approach to impute missing values. It treats each feature with missing data as a target variable and uses the other features as predictors to estimate the missing values.

For the outlier detection we plotted the box plot diagram for the Journey_time feature:



This box plot diagram can be used in the outlier removal, here instead of using Inter Quartile method we went with the simple restriction of values in the range of (12,80) as per the box plot diagram.

## Encoding Methods:

1. Now, since some of the columns had values in non-numeric format therefore, we need to use encoding for the non-numeric data. We first went with the label encoding as there were categorical variables with ordinal relationships as Journey_Time, Day.
2. Now, since the variation of the Journey_time was quite large, and ordinal relationships were already existing therefore applied Target encoding.

## Data Augmentation:

In classification problems with imbalanced classes, where some classes have significantly fewer samples than others, data augmentation can help balance the class distribution. Data augmentation can act as a form of regularization. It helps prevent overfitting by exposing the model to a wider range of variations and noise in the data. This can lead to a more generalizable model. Overfitting case was observed as the testing dataset showed an increased accuracy, but this was not the case with testing dataset (Kaggle).

Therefore, we augmented the training dataset with 5000 samples of augmented rows adding noise to Journey_time specifically. Since we have used 99% of the training dataset hence there were chances of overfitting.

## Models Used:

Since, the dataset was a binary classification, therefore we applied various models such as Logistic Regression, Random Forest classifier, Adaboost, Gradient Boosting and XG boost.

The model with the best accuracy was XG boost, therefore we went on to apply some advanced technique to predict the parameters for the XG boost, Grid Search.

Grid search is a hyperparameter tuning technique used in machine learning to systematically search for the optimal combination of hyperparameters for a given model. It's called "grid search" because it evaluates all possible combinations of hyperparameter values from a predefined grid or set of values.

We provided an array of parameters for each hyperparameter to get the best values for each hyperparameter.

## Submission Logs and Model Variations:

1. **XG Boost with Feature Scaling:**
    - Removed the 'day' and 'bus_id' columns.
    - Filled missing values with mode.
    - Applied feature scaling.
    - Achieved an accuracy of 0.65632.
2. **XG Boost with Data Expansion:**
    - Increased the training data size.
    - Achieved improved accuracy of 0.65756.

3. **Gradient Boost with Data Preprocessing:**
    - Filled missing values with the mode.
    - Did not apply feature scaling.
    - Achieved an accuracy of 0.66042.
4. **Outlier Removal and XG Boost:**
    - Removed outliers in the 'Journey_Time' feature.
    - Utilized XG Boost with 300 trees and a maximum depth of 8.
    - Achieved an accuracy of 0.65905.
5. **XG Boost with Feature Selection and Tuning:**
    - Dropped only the 'Bus_Id' column.
    - Utilized category encoders.
    - Set the maximum depth to 11.
    - Achieved an accuracy of 0.66375.
6. **XG Boost with Feature Selection and Hyperparameter Tuning:**
    - Dropped only the 'Bus_Id' column.
    - Utilized category encoders.
    - Set the maximum depth to 9 and used 270 trees.
    - Achieved an accuracy of 0.66669.

## Best Submission Stats:

**Our best accuracy stood at 0.66828 with the following statistics:**

1. Restricting Journey_Time feature between (12,80) both inclusive.
2. Didn’t use Bus_Index in the training data, dropping columns wasn’t beneficial in our case
3. Dropped duplicates if present.
4. Used iterative imputer with max_iteration=10000 and random_state=32
5. Used Category encoding
6. Augmented training data with 5000 values adding noise in Journey_Time column
7. Applied XG Boost model while splitting training dataset into 99% and 1% testing size.
8. Applied grid search to fine tune the parameters.

(We tried dropping columns of Bus_Id, Day but the accuracy also decreased hence avoided dropping any feature.)

## Steps to Run: ( Platforms: Google Colab, Kaggle)

1. Open the notebook Wall_Breakers.ipynb
2. Load the training dataset, augmented_train.csv
3. Load the testing dataset test.csv
4. Download the resulting csv file as output and submit the same in kaggle.

Link to competition: https://www.kaggle.com/competitions/bus-on-time-or-delayed/overview