**PREDICTING SCHOOL BUS BREAKDOWNS AND DELAYS IN NYC**

Team B

Nasim Aalemi; Shobha Panthi; James Gilmore

FINAL PROJECT REPORT

for

DATA 606 Capstone in Data Science

University of Maryland Baltimore County

2025

Copyright ©2025

ALL RIGHTS RESERVED

# ABSTRACT

The New York City (NYC) school bus system faces frequent breakdowns and delays, affecting student transportation and operational efficiency. This project used data from the NYC Open Data portal to predict school bus incidents through exploratory data analysis (EDA), machine learning (ML), and interactive visualization tools. EDA revealed temporal trends, borough-specific patterns, and main delay causes, guiding feature selection for ML models. An XGBoost classifier predicted incident types with 99% accuracy for delays (F1-score: 99%) and 73% for breakdowns (F1-score: 77%), with “Reason” as the top feature (“Won’t Start” for breakdowns, “Heavy Traffic” for delays). A Random Forest Regressor estimated delay duration with a Root Mean Squared Error (RMSE) of 9.6 minutes and an R² of 82%, where “Bus Company Name” was most influential. Deliverables include a Power BI dashboard showing delay density, trends, and key performance indicators, and a Streamlit web app for real-time predictions. These tools help officials identify high-risk routes, optimize resources, and improve planning, ultimately enhancing reliability and equity in NYC student transportation.

# LIST OF ABBREVIATIONS AND SYMBOLS

*NYC* New York City

DOE Department of Education

*EDA* Exploratory Data Analysis

*ML* Machine Learning

*RSME* Root Square Mean Error

*R2* R-Square

*ODP* Open Data Portal

*SMOTE* Synthetic Minority Oversampling Technique

*CSV* Comma-Separated Values

*XGBoost Extreme Gradient Boosting*

*OPT* Office of Pupil Transportation

BI Business Intelligence

LLC Limited Liability Company

# ACKNOWLEDGMENTS

Team B gratefully acknowledges and thanks Professor Unal Sakoglu for his invaluable guidance, expertise, and support throughout this capstone project. His insights and feedback were instrumental in shaping our approach and ensuring the success of our work. We also extend our appreciation to our colleagues who provided constructive feedback and encouragement at various stages of this project. Their collective contributions were essential to the completion of this endeavor.

# TEAM MEMBERS’ CONTRIBUTIONS

This capstone project involved significant contributions from all members of Team B, with each member taking on specific responsibilities while collaborating on shared tasks. The following table summarizes the primary tasks and individual contributions to the project.

|  |  |  |  |
| --- | --- | --- | --- |
| Tasks | James Gilmore | Shobha Panthi | Nasim Aalemi |
| Data Cleaning | Cleaned and standardized the Bus Company Name column, a challenging task spanning weeks. | Cleaned and standardized the Bus Route column, a highly complex task taking weeks. | Cleaned and standardized the How Long Delayed column, a challenging task spanning weeks. |
| EDA | Contributed equally to analyzing temporal trends, borough patterns, and delay reasons. | Contributed equally to analyzing temporal trends, borough patterns, and delay reasons. | Contributed equally to analyzing temporal trends, borough patterns, and delay reasons. |
| ML Models | Assisted with data preparation and preprocessing. | Developed regression model, including feature selection and evaluation. | Developed classification and regression models, including feature selection and evaluation. |
| Power BI Dashboard | Designed and developed the Power BI dashboard for visualizing delay density, trends, and KPIs. | Assisted in revising and enhancing the report in collaboration with the team. | Assisted in revising and enhancing the report in collaboration with the team. |
| Streamlit | Assisted with data preparation and preprocessing. | Initiated development of the Streamlit web app for real-time incident and delay prediction. | Refined and finalized the Streamlit web app, leveraging prior experience. |
| Project Report | Assisted with the Method section. | Assisted with the Method section. | Authored the full final project report with all detailed sections and analysis. |
| Project Presentations | Contributed equally to preparing and delivering project presentations. | Contributed equally to preparing and delivering project presentations. | Contributed equally to preparing and delivering project presentations. |

**TABLE OF CONTENTS**

[ABSTRACT I](#_Toc206087767)

[LIST OF ABBREVIATIONS AND SYMBOLS II](#_Toc206087768)

[ACKNOWLEDGMENTS III](#_Toc206087769)

[TEAM MEMBERS’ CONTRIBUTIONS III](#_Toc206087770)

[LIST OF TABLES V](#_Toc206087771)

[LIST OF FIGURES V](#_Toc206087772)

[I. INTRODUCTION 1](#_Toc206087773)

[I.1 The description of the problem and the data 1](#_Toc206087774)

[I.2 Background & Literature Review/Survey 2](#_Toc206087775)

[II. METHODS 2](#_Toc206087776)

[II.1 Description of Data Processing and Feature Engineering Employed 2](#_Toc206087777)

[Cleaning the Bus\_Company\_Name Column 2](#_Toc206087778)

[Cleaning the Route\_Number Column 3](#_Toc206087779)

[Cleaning the How\_Long\_Delayed Column 4](#_Toc206087780)

[Feature Engineering 5](#_Toc206087781)

[Handling Class Imbalance with SMOTE 5](#_Toc206087782)

[II.2 Description of Analyses Employed for (Classification and Regression) 5](#_Toc206087783)

[Classification: XGBoost Classifier 5](#_Toc206087784)

[Regression Model: Predicting Delay Duration 6](#_Toc206087785)

[III. RESULTS 6](#_Toc206087786)

[Classification Model Performance 7](#_Toc206087787)

[7](#_Toc206087788)

[Regression Model Performance 7](#_Toc206087789)

[IV. DISCUSSIONS AND CONCLUSIONS 8](#_Toc206087790)

[Conclusion 8](#_Toc206087791)

[V. FUTURE WORK 8](#_Toc206087792)

[VI. REFERENCES 9](#_Toc206087793)

[VII. APPENDICES 10](#_Toc206087794)

[VII.1 Programming Codes 10](#_Toc206087795)

# LIST OF TABLES

Table 1: Dataset Feature and Input Attribute 1

Table 2: Model Evaluation Report 6

2.1 Validation Report 6

2.2 Test Report 6

# LIST OF FIGURES

Figure 1: Top 10 School Bus Company Names 3

Figure 2: Average delay minutes distribution and counts 4

Figure 3: Most important features after running the regression model 7

Figure 4: Most important features after running the classification model 7

# I. INTRODUCTION

## I.1 The description of the problem and the data

The New York City (NYC) school bus system faces ongoing logistical challenges due to frequent vehicle breakdowns and delays, impacting over 150,000 students each day (School Bus Delay, 2025). These incidents result in missed instructional time, disrupted family routines, and increased operational strain for the Department of Education (DOE). The primary goal of our project was to predict school bus breakdowns and delays using machine learning models trained on real operational data, and to identify the key factors driving these incidents. Our capstone project addressed two key predictive questions:

1. What operational factors contribute most to school bus delays and breakdowns in NYC?
2. Can we accurately predict future incidents based on route, contractor, and historical patterns?

To answer these questions, we implemented an integrated data science workflow combining exploratory data analysis (EDA), supervised machine learning, and interactive visual analytics tools. We evaluated multiple modeling techniques and ultimately delivered a predictive framework alongside practical tools: a Power BI dashboard for interactive visual exploration and a Streamlit web app for real-time predictions. We used a large-scale public dataset from the NYC Open Data Portal (ODP), titled Bus Breakdown and Delays (NYC ODP, 2025). This dataset includes 228,427 records spanning the last three academic years: 2022–2023, 2023–2024, and 2024–2025. Each row in the dataset represents a unique incident (delay or breakdown) reported by contracted bus vendors. The dataset includes both categorical and numerical features relevant to time, location, route, vendor, and cause of the incident. The number of records totaled 228,427, with 21 original features and 6 additional engineered columns derived from datetime and categorical data. The file format was CSV, and it was loaded into a Pandas DataFrame for processing. We selected the following input attributes in Table 1.

Table 1: Dataset Feature and Input Attribute

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Type** | **Description** |
| Bus\_Company\_Name | Categorical | Contractor or vendor name |
| Route\_Number | Categorical | Specific bus route. |
| Delay\_Reason | Categorical | Cause of the incident |
| Breakdown\_or\_Running\_Late | Binary | Target variable for classification |
| How\_Long\_Delayed | Numeric | Delay duration (target for regression) |
| Occurred\_On | Timestamp | Used to extract Hour, Day, and Month |
| Has\_Contractor\_Notified\_Schools | Binary | Whether the contractor notified schools |
| Has\_Contractor\_Notified\_Parents | Binary | Whether the contractor notified parents |
| Have\_You\_Alerted\_Opt | Binary | Whether the contractor notified OPT |
| Borough | Categorical | Where incident occurred |
| School\_Age\_or\_PreK | Categorical | Indicates student age group |
| Number\_of\_Students\_on\_the\_Bus | Numeric | Number of students on the bus |
| School\_Year | Categorical | To filter the most recent three years |

## I.2 Background & Literature Review/Survey

The NYC school bus system experiences frequent delays and breakdowns that disrupt classroom instruction, complicate family logistics, and strain DOE’s operations (School Bus Delay, 2025). In response, our capstone project aimed to predict these incidents using machine learning and identify the operational factors contributing to them. We utilized a public dataset from the NYC Open Data Portal containing over 228,000 reported incidents across three academic years. To prepare the data, we addressed inconsistencies in labeling, handled missing values, encoded categorical variables, and engineered new features such as hour of day, day of week, and school year. The feature importance rankings later helped us identify the strongest predictors of bus incidents. We built supervised learning models and delivered two interactive tools: a Power BI dashboard that visualizes delay trends by route, vendor, and time, and a Streamlit web app that predicts both the type of incident and the estimated delay duration based on user input. These tools were developed to assist DOE officials and school transportation coordinators in identifying high-risk patterns, anticipating disruptions, and making data-driven decisions. Our project highlights the potential of machine learning to improve efficiency, reliability, and equity of student transportation in New York City.

In the existing literature, machine learning has been widely applied to traffic delay prediction and public transit optimization, but few studies focus specifically on school bus operations, and even fewer at the city-wide scale. Previous work has largely emphasized route optimization and fleet management rather than predictive modeling based on incident reports. Our project did not propose a novel algorithm but instead implemented and compared established methodologies (Transportation, 2025). We explored several supervised learning models for both classification (predicting breakdown vs. delay) and regression (estimating delay duration), including, XGBoost Classifier, Random Forest Classifier & Regressor, and Logistic and Linear Regression (as baselines). We evaluated these models using standard performance metrics such as accuracy, F1-score, precision, recall, R², and RMSE. Our approach also involved handling class imbalance using SMOTE, encoding categorical variables via Target Encoding, and applying Standard Scaler for numerical features. This structured approach allowed us to assess predictive performance while also maintaining interpretability for real-world use.

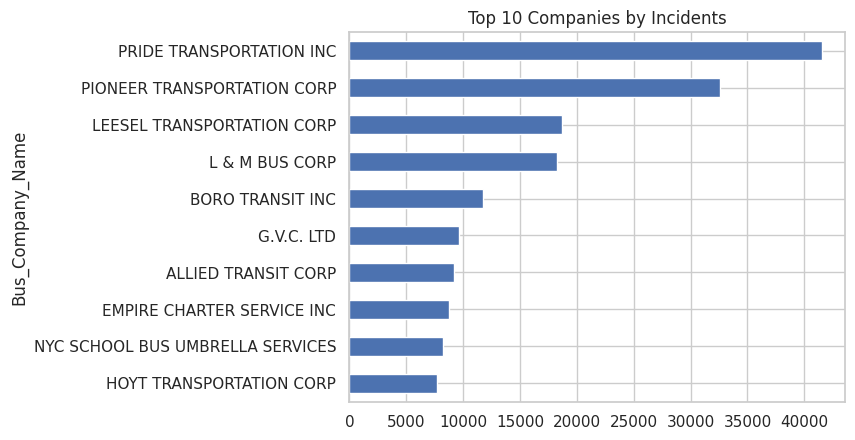
# II. METHODS

Our dataset which consisted of 228,427 school bus incident records spanned over three recent school years (2022–2023, 2023–2024, 2024–2025) and contained a mix of categorical, numerical, and datetime features related to bus breakdowns and delays. To prepare the data for modeling, thorough cleaning and preprocessing were required, especially for key columns critical to predictive accuracy: Bus\_Company\_Name, Route\_Number, and How\_Long\_Delayed.

## II.1 Description of Data Processing and Feature Engineering Employed

### *Cleaning the Bus\_Company\_Name Column*

The Bus\_Company\_Name column, which recorded the name of the contracted vendor responsible for each bus, showed significant inconsistencies due to manual entry errors, use of abbreviations, suffix variations (e.g., “Inc.”, “LLC”), and inconsistent casing or spacing. Common examples like “Consolidated Bus Co.,” “consolidated bus company,” “Consolidated Bus LLC,” and “Conso Bus” frequently appeared but all referred to the same vendor. To standardize this column, we implemented a multi-step cleaning process. First, we compiled a list of invalid names and removed them from the column. Second, we applied a normalization technique using Python’s unicodedata library to group similar names by removing punctuation, special characters, and common suffixes, and by trimming leading or trailing spaces. All entries were then converted to lowercase to eliminate casing discrepancies. This grouping step was semi-automated and manually reviewed by looking up each company online to verify accuracy. We then created a mapping dictionary to consolidate all identified variants under standardized vendor names, for example, mapping all variations to “consolidated bus company.” Finally, we cross-referenced the mapping by comparing vendor frequency counts before and after cleaning to ensure classification accuracy and consistency. This rigorous cleaning ensured that the Bus\_Company\_Name field accurately represented distinct vendors, which is essential for meaningful feature importance analysis and operational insights. After cleaning, the ten bus companies with the highest number of reported incidents were identified, as shown in Figure 1.



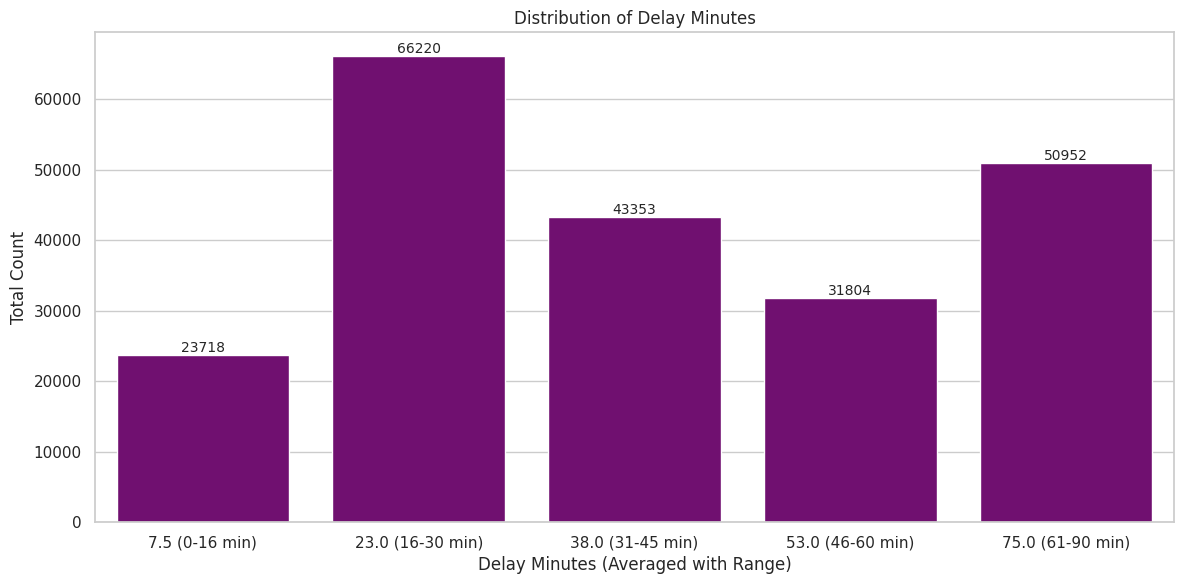
*Figure 1: Top 10 School Bus Company Names providing services to NYC Public School by incident*

### *Cleaning the Route\_Number Column*

The Route\_Number column identified the specific bus route associated with each incident and consisted of alphanumeric codes. However, this field contained inconsistent formatting that compromised data quality. Issues included mixed casing, such as “bx120” vs. “BX120,” leading zeros in the numeric portion (e.g., “BX0120”), extraneous whitespace (e.g., “ BX120 ”), and occasional non-standard characters introduced by manual entry errors. To address these inconsistencies, we applied a structured cleaning process. First, we stripped all leading and trailing whitespace from each entry. Next, we converted all route numbers to uppercase to enforce uniform casing. Using regular expressions, we removed leading zeros from the numeric portion of route codes, standardizing entries such as “BX0120” to “BX120.” To further ensure consistency, we validated each entry against the expected alphanumeric pattern and removed special characters like #, :, and spaces, allowing us to flag or exclude entries that did not conform to the required format.

This cleaning process enabled accurate grouping and analysis of route-specific data, which was essential for improving the models’ ability to learn patterns influencing bus delays and breakdowns.

### *Cleaning the How\_Long\_Delayed Column*

 The How\_Long\_Delayed column recorded the duration of each bus delay and was originally treated as a categorical field due to the presence of non-numeric and inconsistent entries. Although the delay values were fundamentally quantitative, the column contained a wide variety of formats that prevented it from being immediately usable for regression analysis. In addition to its categorical data type, approximately 5% of the entries were missing or null, and many others included non-numeric strings such as “unknown,” “N/A,” or delay ranges like “10–15 minutes.” Some cells even contained unrelated text descriptions or operational notes. To transform this column into a usable continuous variable, we undertook an extensive cleaning process. First, we treated rows with missing values as breakdowns. For entries that expressed delay ranges (e.g., “10–15”), we used regular expressions to extract the lower and upper bounds and replaced them with their mean (e.g., 12.5 minutes). Next, we removed any units or additional text, such as “min” or “minutes” and converted all remaining values to floating-point numbers. We also addressed extreme values by capping outliers: specifically, we excluded delay durations above the 99th percentile, which was around 180 minutes, to avoid skewing the model training. After these transformations, the How\_Long\_Delayed column was successfully converted from a messy categorical variable into a well-structured continuous numerical field. This cleaned and converted column was then ready to serve as the target variable in our regression model for predicting bus delay durations. The distribution of this column is displayed in Figure 2.

*Figure 2: Average delay minutes distribution and counts*

### *Feature Engineering*

To maximize model performance, we engineered additional features from the dataset:

* Extracted hour of day, day of week, and month from the Occurred\_On timestamp to capture temporal patterns (e.g., rush hours, weekdays vs. weekends).
* Encoded categorical features (e.g., Bus\_Company\_Name, Route\_Number) using label and target encoding, which replaced each category with the mean of the target variable within that category, reducing dimensionality while preserving predictive power.
* Applied Standard Scaler to numerical features to normalize scales, facilitating faster and more stable model convergence.

### *Handling Class Imbalance with SMOTE*

The classification target, Breakdown\_or\_Running\_Late, was highly imbalanced:

* Approximately 92% of records were “Running Late” (delays).
* Only 8% were “Breakdown.”

Imbalanced classes can bias models towards the majority class, reducing detection of breakdowns. To counter this, we applied SMOTE (Synthetic Minority Over-sampling Technique) during model training, which synthetically generates new minority class samples based on nearest neighbors in feature space. SMOTE was only applied to the training set to prevent data leakage, ensuring the validation and test sets reflected real-world class distributions.

## II.2 Description of Analyses Employed for (Classification and Regression)

### *Classification: XGBoost Classifier*

We developed a classification model to predict whether an incident was a breakdown or a delay. To evaluate performance, we compared several classification algorithms:

* Random Forest Classifier: An ensemble method robust to non-linearities.
* XGBoost Classifier: A gradient boosting model known for accuracy and speed.

We developed a classification model to predict whether an incident was a breakdown or a delay. Our dataset was highly imbalanced, with only 8% of incidents labeled as breakdowns and 92% as delays. To address this imbalance, we applied SMOTE (Synthetic Minority Over-sampling Technique) during model training. We compared several classification algorithms, and XGBoost outperformed the others, achieving superior accuracy, precision, recall, and F1 scores, particularly on the minority breakdown class. Its built-in support for handling sparse data, missing values, and regularization made it especially well-suited to our complex dataset. We further optimized model performance by tuning hyperparameters through grid search and 5-fold cross-validation, adjusting values such as learning rate, maximum depth, and number of estimators to minimize overfitting.

Final classification metrics on the held-out validation set included:

* Accuracy: 99% for delays, 73% for breakdowns
* F1-Score: 99% for delays, 77% for breakdowns

Additional detailed information provided in classification evaluation Table 2 below.

Table 2: Model Evaluation Report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Validation Report** | | | | |
| Class | Precision | Recall | F1-Score | Support |
| 0 | 0.99 | 0.98 | 0.99 | 43207 |
| 1 | 0.73 | 0.79 | 0.76 | 2475 |
| Accuracy |  |  | 0.97 | 45682 |
| Macro Avg | 0.86 | 0.89 | 0.87 | 45682 |
| Weighted Avg | 0.97 | 0.97 | 0.97 | 45682 |
| **Test Report** | | | | |
| Class | Precision | Recall | F1-Score | Support |
| 0 | 0.99 | 0.98 | 0.99 | 43208 |
| 1 | 0.73 | 0.82 | 0.77 | 2475 |
| Accuracy |  |  | 0.97 | 45683 |
| Macro Avg | 0.86 | 0.90 | 0.88 | 45683 |
| Weighted Avg | 0.98 | 0.97 | 0.97 | 45683 |

Feature importance analysis revealed the top three predictors included delay Reason (e.g., “Won’t Start” or “Heavy Traffic”), Route\_Number\_Clean, and School\_Year.

### *Regression Model: Predicting Delay Duration*

For the regression task, we tested:

* Linear Regression: Baseline model assuming linear relationships.
* Gradient Boosting Regressor: Powerful boosting method for non-linearities.
* Random Forest Regressor: Ensemble of decision trees capturing complex interactions.

The Random Forest Regressor provided the best balance of accuracy and robustness, handling skewness in delay durations and minimizing RMSE.

Hyperparameters such as tree depth, minimum samples per leaf, and number of estimators were tuned through grid search with cross-validation.

Performance metrics on validation data:

* Root Mean Squared Error (RMSE): 9.6 minutes
* R² Score: 0.82, indicating strong explanatory power

The feature Bus\_Company\_Name was the most influential predictor of delay duration, followed by Route\_Number\_Clean, and Run\_Type.

Our final deliverables included two interactive tools: a Power BI dashboard for exploring delay patterns by route, vendor, time, and location, and a Streamlit app that provides real-time predictions of incident type and expected delay based on user inputs. These tools support data-driven planning and help DOE officials proactively manage school bus disruptions.

# III. RESULTS

Our predictive models performed well in estimating both the type of incident and the delay duration. The regression model accurately predicted how long a bus would be delayed, while the classification model effectively distinguished between delays and breakdowns, despite the class imbalance in the data.

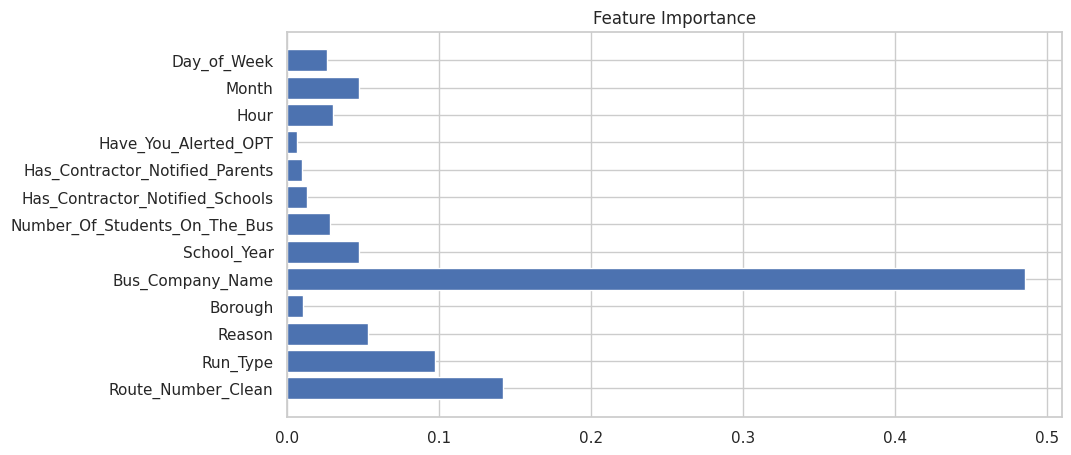
### *Classification Model Performance*

The classification model was designed to differentiate between two incident types: Running Late and Breakdown. Because the dataset was highly imbalanced, 92% delays and only 8% breakdowns, special attention was given to improving performance in the minority class. By combining XGBoost with SMOTE to rebalance the classes, the model achieved 99% accuracy for delays and 73% accuracy for breakdowns, with corresponding F1-scores of 99% and 77%, respectively, as shown in Table 2. Figure 4 presents the model’s feature importance.

### 

*Figure 4: Most important features after running the classification model (Reason is biggest).*

### *Regression Model Performance*

The regression model was designed to predict the length of school bus delays in minutes. After training and evaluation, it achieved a RMSE of 9.6 minutes, with an R² score of 0.82, indicating strong predictive accuracy. Among all features, Bus\_Company\_Name emerged as the most influential predictor of delay duration, highlighting the role of vendor performance in operational reliability. Figure 3 shows results.

*Figure 3: Most important features after running the regression model (Bus\_Company\_Name is biggest).*

# IV. DISCUSSIONS AND CONCLUSIONS

Our analysis showed that machine learning can effectively predict school bus incidents using historical operational data. The regression model achieved strong performance with an R² score of 0.82 and a RMSE of 9.6 minutes. This indicates that the model can reliably estimate delay durations, with Bus\_Company\_Name emerging as the most influential predictor. This suggests that contractor performance plays a key role in determining the severity of delays. For classification, predicting whether an incident was a Breakdown or Running Late, XGBoost yielded the best performance when paired with SMOTE to address class imbalance. Only 8% of incidents were breakdowns, yet the model achieved an F1-score of 77% for the minority class and 99% for the majority class. The most important features included Delay\_Reason, Route Number, and School Year. These findings demonstrate that historical patterns and operational metadata can be leveraged to flag high-risk scenarios in advance.

Overall, our deployed tools, a Power BI dashboard and a Streamlit prediction app, translate these insights into practical applications. They enable the DOE to identify vulnerable routes, evaluate vendor reliability, and anticipate disruptions before they occur.

### *Conclusion*

This project demonstrated that machine learning models trained on NYC’s school bus incident data can reliably predict both the type and duration of future disruptions. By integrating these models into user-friendly tools, we provide actionable insights to enhance the reliability and equity of student transportation across New York City.

# V. FUTURE WORK

Given more time and resources, several promising directions could enhance the impact and depth of this work. The dataset itself is incredibly rich, spanning over 10 years and continuously updated in real time, making it ideal for a range of extended analyses. First, incorporating external contextual data, such as local weather conditions, traffic congestion, and public holidays, could significantly improve model performance by accounting for situational influences on bus delays and breakdowns. The addition of GPS tracking and real-time ridership data would also allow for more precise, personalized predictions that reflect actual bus activity and student load. Beyond predictive modeling, unsupervised learning methods like clustering could uncover hidden operational patterns or risk profiles among vendors, routes, or time windows. Time-series forecasting could be applied to analyze seasonal trends and anticipate future spikes in incidents, allowing for better resource planning. Simulation studies could test the effects of proposed interventions (e.g., re-routing, vendor reassignment) before implementation, providing decision-makers with low-risk evaluation tools. Furthermore, equity-focused analyses could examine whether certain student populations or neighborhoods face disproportionate service disruptions, enabling the Department of Education to address disparities. In all, the real-time and historical nature of this data opens the door for multidisciplinary studies that blend machine learning, transportation planning, and public policy, with the goal of making NYC’s school transportation system more efficient, equitable, and resilient.

# VI. REFERENCES

*Bus breakdown and delays | NYC Open Data*. (2025, June 15).

<https://data.cityofnewyork.us/Transportation/Bus-Breakdown-and-Delays/ez4e-fazm/about_data>

*OPT\_Open\_Data\_Descriptions\_V0.2 | NYC Open Data. (2025, June 16).*

<https://data.cityofnewyork.us/Transportation/Bus-Breakdown-and-Delays/ez4e-fazm/about_data>

*School Bus Delay* | *NYC Public Schools* (2025, June 15).

<https://www.opt-osfns.org/opt/vendors/busbreakdowns/public/default.aspx?search=YES>

Transportation status | Baltimore County Public Schools (2025, June 16).

<https://www.bcps.org/parents/transportation_status>

# VII. APPENDICES

The following section contains some of the relevant source code used for this project report and beyond. The code is written in Python (version 3.x) and can be run in development environments such as VS Code, Jupyter Notebook, or Google Colab. Any required environment variables, library dependencies, and configuration steps are noted within the code to ensure reproducibility. All scripts are provided in a copy-paste-ready format for easy execution and validation.

## VII.1 Programming Codes

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, TargetEncoder

from google.colab import drive

import pandas as pd

# Mounting Google Drive

drive.mount("/content/drive", force\_remount=True)

# File path

file\_path = "/content/drive/My Drive/Data\_606\_Capstone\_Project"

# Loading Bus Breakdown CSV into DataFrame

df = pd.read\_csv(f'{file\_path}/Bus\_Breakdown\_and\_Delays\_20250603.csv')

"""Filtering the data for three School Years (SY) 2022-2023, 2023-2024, and 2024-2025"""

"""#EDA Process"""

#Filtering the data to only include the most recent 3 years after Covid-19

three\_recent\_SY\_Yrs = ['2022-2023', '2023-2024', '2024-2025']

df = df[df['School\_Year'].isin(three\_recent\_SY\_Yrs)]

df.columns

pd.set\_option('display.max\_columns', None) #Showing all columns

df.head()#this is how our dataset look like

#Distribution of Boroughs in NYC

df.rename(columns={'Boro': 'Borough'}, inplace=True)

not\_the\_city= ['Nassau County', 'Westchester', 'New Jersey', 'All Boroughs', 'Rockland County', 'Connecticut']

df = df[~df['Borough'].isin(not\_the\_city)]

borough\_counts = df['Borough'].value\_counts()

colors = sns.color\_palette('pastel')[0:len(borough\_counts)]

plt.figure(figsize=(5, 5), dpi=120)

plt.pie(

    borough\_counts,

    labels=borough\_counts.index,

    autopct='%1.1f%%',

    startangle=100,

    colors=colors,

    wedgeprops={'edgecolor': 'black'}

)

plt.title('Distribution of Boroughs in NYC', fontsize=14, fontweight='bold')

plt.tight\_layout()

plt.show()

"""This right here takes the column and sort the value for the num of students that'll realistically be on the bus. The number we're focusing on according to google is 72 students that can fit on a school bus(YELLOW BUS) in NYC."""

df['Run\_Type']

#a visualization to show the outliers

sns.set(style="whitegrid")

plt.figure(figsize=(3, 2), dpi=120)

sns.boxplot(

    y=df['Number\_Of\_Students\_On\_The\_Bus'],

    color='skyblue',

    linewidth=1.5,

    fliersize=4

)

plt.title('Distribution of Students on the Bus During Accidents', fontsize=12, fontweight='bold')

plt.ylabel('Number of Students', fontsize=10)

plt.xlabel('')

plt.tight\_layout()

plt.show()

#Before I replaced the rows with 0, I wanted to see how many rows were over our target. Based off of this numbr, I realised it's better to get rid of the 105 columns.

(df['Number\_Of\_Students\_On\_The\_Bus'] > 72).sum()

#Now what Im doing is im filtering the dataframe for us to not see the 105 columns that are over the count 72

df = df[df['Number\_Of\_Students\_On\_The\_Bus'] <= 72]

#This is just a quick sort check

df['Number\_Of\_Students\_On\_The\_Bus'].sort\_values(ascending=False)

df.describe().round(0)

df.isnull().sum()

"""These column are not too pertinent becasue according to the data dictionary it is: "Some reports of bus breakdowns or delays originate from calls to the OPT Customer Service line who records incidents. When this happens, the record will have the Incident reference number." More importantly, it contains too many missing values."""

#dropping records for where there is null values

df.dropna(subset=['Borough','Run\_Type','Bus\_No',

                  'Route\_Number', 'Reason'], inplace=True)

#counting the dataframe

df.count()

df.describe().round(1)

"""After dropping the null values, the dataset still remains substantially large at 689K records, which is quite sizable."""

df.dtypes

"""##Cleaning the How Delayed Column"""

#Standardized the How\_Long\_Delayed column and convert everything to minute

import re

import numpy as np

def standardize\_delay\_time(text):

    if pd.isna(text) or text in ['?', '??', '???', '????', 'unknown', 'unk', 'UNSURE', 'TBD', 'N/A', 'NI0634', 'NI0627', 'NI0933', 'NI932', 'NI2134', 'NI0306']:

        return np.nan

    original = str(text).strip()

    # Clean the text

    cleaned = original.lower()

    cleaned = re.sub(r'[^a-z0-9\s/.-]', '', cleaned)  # Remove special chars except those used in ranges/dates

    cleaned = re.sub(r'\s+', ' ', cleaned).strip()

    # Handle date-like entries (e.g., "15-Oct")

    if re.match(r'\d{1,2}-[a-z]{3}', cleaned):

        return np.nan

    # Handle special cases

    special\_cases = {

        'half hour': 30,

        'halfhour': 30,

        '1/2 hour': 30,

        '1/2hour': 30,

        '1/2 hr': 30,

        'one hour': 60,

        'hour': 60,

        '1 hour': 60,

        '1hr': 60,

        '1 hrs': 60,

        '1h': 60,

        '1.5 hrs': 90,

        '1.5 hour': 90,

        '1 1/2 hr': 90,

        '1hr30min': 90,

        '1 hour 30min': 90,

        '1hr 30min': 90,

        '1hr 45min': 105,

        '1hr20min': 80,

        '1hr40min': 100,

        '2hr': 120,

        '2 hrs': 120,

        '2 hours': 120,

        '2hrs': 120,

        '3 hr': 180,

        '3 hrs': 180,

        '3 hours': 180,

        '4 hours': 240

    }

    for case, minutes in special\_cases.items():

        if case in cleaned:

            return minutes

    # Extract all numbers

    numbers = re.findall(r'\d+', cleaned)

    if not numbers:

        return np.nan

    # Handle ranges (e.g., 15-20, 10/15) and convert to Avg

    if '-' in cleaned or '/' in cleaned or 'to' in cleaned:

        try:

            if len(numbers) >= 2:

                num1, num2 = int(numbers[0]), int(numbers[1])

                return (num1 + num2) / 2  # Return midpoint

            else:

                return int(numbers[0])

        except:

            return np.nan

    # Handle simple numeric values

    if len(numbers) == 1 and not any(x in cleaned for x in ['hr', 'hour', 'min', 'mns', 'mins', 'm']):

        return int(numbers[0])

    # Handle time units

    total\_minutes = 0

    # Hours first

    hour\_matches = re.findall(r'(\d+)\s\*(?:hr|hour|h|ho)', cleaned)

    for match in hour\_matches:

        total\_minutes += int(match) \* 60

    # Then minutes

    min\_matches = re.findall(r'(\d+)\s\*(?:min|mins|mns|m|mn|mi|minutes|minute|minut|minu|minuets|minites|minuts|minuites|minnutes|minte|minuate|minuste|minuets|minuites|minuets|minuites|minnutes|minte|minuate|minuste|minuets|minuites)', cleaned)

    for match in min\_matches:

        total\_minutes += int(match)

    # If we found any time units, return the total

    if hour\_matches or min\_matches:

        return total\_minutes if total\_minutes > 0 else np.nan

    # If no units found but we have numbers, assume minutes

    if numbers:

        return int(numbers[0])

    return np.nan

# Apply the function to column

df['Delay\_Minutes'] = df['How\_Long\_Delayed'].apply(standardize\_delay\_time)

# For values that seem too large to be minutes (e.g., > 300), we make them blank and then mark them for breakdown instead of delay

df.loc[df['Delay\_Minutes'] > 300, 'Delay\_Minutes'] = np.nan

#count of 'Delay\_Minutes' where values are null

df['Delay\_Minutes'].isnull().sum()

# Dropping rows where 'Delay\_Minutes' is 0

df.drop(df[df['Delay\_Minutes'] == 0].index, inplace=True)

# Marking  'Breakdown\_or\_Running\_Late' as "Breakdown" where 'Delay\_Minutes' is Null

df.loc[df['Delay\_Minutes'].isnull(), 'Breakdown\_or\_Running\_Late'] = 'Breakdown'

# getting count of both breakdown and running late

df['Breakdown\_or\_Running\_Late'].value\_counts()

df['Delay\_Minutes'].describe().round(1)

# Distribution of Delay Minutes by creating bins first

bins = [0, 16, 30, 45, 60, 90]

avg\_labels = [7.5, 23.0, 38.0, 53.0, 75.0]

range\_labels = ['0-16 min', '16-30 min', '31-45 min', '46-60 min', '61-90 min']

display\_labels = [f"{avg} ({rng})" for avg, rng in zip(avg\_labels, range\_labels)]

df['Delay\_Bin'] = pd.cut(df['Delay\_Minutes'], bins=bins, labels=display\_labels, include\_lowest=True)

# Counting occurrences in each bin

delay\_counts = df['Delay\_Bin'].value\_counts().reindex(display\_labels)

#Plotting

plt.figure(figsize=(12, 6))

ax = sns.barplot(

    x=delay\_counts.index,

    y=delay\_counts.values,

    color='purple'

)

for bar, count in zip(ax.patches, delay\_counts.values):

    height = bar.get\_height()

    ax.text(

        bar.get\_x() + bar.get\_width() / 2,

        height,

        f'{int(count)}',

        ha='center',

        va='bottom',

        fontsize=10

    )

plt.title('Distribution of Delay Minutes')

plt.xlabel('Delay Minutes (Averaged with Range)')

plt.ylabel('Total Count')

plt.xticks(rotation=0)

plt.tight\_layout()

plt.show()

# Average Delay Minutes by Borough

df.rename(columns={'Boro': 'Borough'}, inplace=True)

not\_the\_city = ['Nassau County', 'Westchester', 'New Jersey', 'All Boroughs', 'Rockland County', 'Connecticut']

df = df[~df['Borough'].isin(not\_the\_city)]

# average delay per borough

borough\_avg\_delay = df.groupby('Borough')['Delay\_Minutes'].mean().sort\_values(ascending=False)

total\_avg\_sum = borough\_avg\_delay.sum()

# labels

labels = [

    f"{borough}\n{(avg/total\_avg\_sum)\*100:.1f}% ({avg:.1f} min)"

    for borough, avg in borough\_avg\_delay.items()

]

# chart

colors = sns.color\_palette('pastel')[0:len(borough\_avg\_delay)]

explode = [0.03] \* len(borough\_avg\_delay)

plt.figure(figsize=(7, 7), dpi=120)

wedges, texts = plt.pie(

    borough\_avg\_delay,

    labels=labels,

    colors=colors,

    startangle=140,

    wedgeprops={'edgecolor': 'black'},

    explode=explode

)

centre\_circle = plt.Circle((0, 0), 0.50, fc='white')

plt.gca().add\_artist(centre\_circle)

# plot

plt.title('Average Delay Minutes by Borough', fontsize=14, fontweight='bold')

plt.tight\_layout()

plt.show()

"""## Cleaning the Route Number"""

# Ensure Route\_Number is string

df['Route\_Number'] = df['Route\_Number'].astype(str).str.strip()

# Replace exact '0' or '0.0' with NaN BEFORE cleaning

df['Route\_Number'] = df['Route\_Number'].replace(['0', '0.0'], np.nan)

# Function to clean route numbers

def clean\_route(route):

    if pd.isna(route):

        return None

    route\_str = str(route).strip().upper()

    # Remove special characters like #, :, spaces

    route\_str = re.sub(r'[^A-Z0-9]', '', route\_str)

    # Remove values that are still empty or zero after cleaning

    if route\_str in {"", "0", "0.0"}:

        return None

    # Remove time formats like 08:15AM, 7:00, etc.

    if re.match(r'^\d{1,2}:\d{2}(:\d{2})?(AM|PM)?$', route\_str, flags=re.IGNORECASE):

        return None

    # Remove values like '8AM' or '10PM'

    if re.search(r'(AM|PM)', route\_str, flags=re.IGNORECASE):

        return None

    # Remove leading zeros from numeric strings

    route\_str = re.sub(r'^0+', '', route\_str)

    return route\_str

# Apply the cleaning function

df['Route\_Number\_Clean'] = df['Route\_Number'].apply(clean\_route)

df['Route\_Number\_Clean'].value\_counts()

#This groups each route by incident type and counts how many were breakdowns vs just running late.

status\_by\_route = df.groupby(['Route\_Number\_Clean', 'Breakdown\_or\_Running\_Late']).size().unstack(fill\_value=0)

status\_by\_route.head(10).plot(kind='bar', stacked=True, title='Breakdown vs Running Late by Route')

plt.xlabel('Route Number')

plt.ylabel('Count')

plt.tight\_layout()

plt.show()

#This calculates how many students are typically on the bus for each route. You see which routes carry the most kids.

avg\_students = df.groupby('Route\_Number\_Clean')['Number\_Of\_Students\_On\_The\_Bus'].mean().sort\_values(ascending=False).head(10)

avg\_students.plot(kind='bar', title='Average Students on Bus by Route')

plt.xlabel('Route Number')

plt.ylabel('Average Number of Students')

plt.tight\_layout()

plt.show()

"""## Cleaning the Company Names"""

# Define list of invalid values to exclude

invalid\_names = ['1967', '1992', '1997', 'COMPANY', '`', 'PL1800', 'IY', 'BUS COMPANY', 'MS', 'MR', 'GUILLEN RODRIGUEZ']

# Filter out

df = df[~df['Bus\_Company\_Name'].isin(invalid\_names)]

import re

import unicodedata

def normalize\_company(name):

    name = str(name)

    name = unicodedata.normalize("NFKD", name)

    name = re.sub(r'[^\w\s&.-]', '', name)      # Remove special chars except useful punctuation

    name = re.sub(r'\s+', ' ', name).strip()    # Collapse spaces

    return name.upper()

# Apply normalization

df['Bus\_Company\_Name'] = df['Bus\_Company\_Name'].apply(normalize\_company)

# Extract possible alpha+number codes at the end (B2192, B2, etc.)

df['Company\_Code'] = df['Bus\_Company\_Name'].str.extract(r'([A-Z]\d{1,4})')

# Rule: Keep codes only if 4 digits (e.g., B2192), remove if short like B2

df['Company\_Code'] = df['Company\_Code'].apply(lambda x: x if (isinstance(x, str) and re.match(r'[A-Z]\d{4}$', x)) else None)

# Remove any suffix codes from the company name (even short ones)

df['Bus\_Company\_Name'] = df['Bus\_Company\_Name'].str.replace(r'\s\*[A-Z]\d{1,4}$', '', regex=True).str.strip()

# Re-normalize after removal

df['Bus\_Company\_Name'] = df['Bus\_Company\_Name'].apply(normalize\_company)

standardize\_map = {

    # CHILDREN

    'CHILDREN`S TRANS INC': 'CHILDREN TRANSIT INC',

    'CHILDREN`S TRANS INC. (B2321)': 'CHILDREN TRANSIT INC',

    'CHILDRENS TRANS INC': 'CHILDREN TRANSIT INC',

    'CHILDRENS TRANS INC.': 'CHILDREN TRANSIT INC',

    # CONSOLIDATED BUS

    'CONSOLIDATED BUS TRANS I': 'CONSOLIDATED BUS TRANSIT INC',

    'CONSOLIDATED BUS TRANS INC': 'CONSOLIDATED BUS TRANSIT INC',

    'CONSOLIDATED BUS TRANSIT INC': 'CONSOLIDATED BUS TRANSIT INC',

    'CONSOLIDATED BUS TRANS. I': 'CONSOLIDATED BUS TRANSIT INC',

    'CONSOLIDATED BUS TRANS. INC.': 'CONSOLIDATED BUS TRANSIT INC',

    'CONSOLIDATED BUS TRANSIT INC.': 'CONSOLIDATED BUS TRANSIT INC',

    # DON THOMAS

    'DON THOMAS BUSES, INC.': 'DON THOMAS BUSES INC',

    'DON THOMAS BUSES, INC. (B2321)': 'DON THOMAS BUSES INC',

    'DON THOMAS BUSES INC.': 'DON THOMAS BUSES INC',

    # FIRST STEPS

    'FIRST STEPS TRANS INC. (B2192)': 'FIRST STEPS TRANSIT INC',

    'FIRST STEPS TRANS, INC': 'FIRST STEPS TRANSIT INC',

    'FIRST STEPS TRANSP INC': 'FIRST STEPS TRANSIT INC',

    'FIRST STEPS TRANSP INC.': 'FIRST STEPS TRANSIT INC',

    'FIRST STEPS TRANS INC': 'FIRST STEPS TRANSIT INC',

    'FIRST STEPS TRANS INC.': 'FIRST STEPS TRANSIT INC',

    # LORINDA

    'LORINDA ENT LTD': 'LORINDA ENTERPRISES LTD',

    'LORINDA ENTERPRISES LTD': 'LORINDA ENTERPRISES LTD',

    'LORINDA ENTERPRISES LTD.': 'LORINDA ENTERPRISES LTD',

    # MJT

    'MJT BUS': 'MJT BUS COMPANY INC',

    'MJT BUS COMPANY INC': 'MJT BUS COMPANY INC',

    # MONTAUK

    'MONTAUK STUDENT TRANS IN': 'MONTAUK STUDENT TRANS INC',

    'MONTAUK STUDENT TRANS INC': 'MONTAUK STUDENT TRANS INC',

    # SMART PICK

    'SMART PICK': 'SMART PICK INC',

    'SMART PICK INC': 'SMART PICK INC',

    # THIRD AVENUE

    'THIRD AVENUE TRANSIT': 'THIRD AVENUE TRANSIT INC',

    'THIRD AVENUE TRANSIT INC': 'THIRD AVENUE TRANSIT INC',

    # TWENTY FIRST AVE

    'TWENTY FIRST AV TRANSP B': 'TWENTY FIRST AV TRANSIT',

    'TWENTY FIRST AV TRANSIT': 'TWENTY FIRST AV TRANSIT',

    # LEESEL

    'LEESEL TRANSP CORP': 'LEESEL TRANSPORTATION CORP',

    'LEESEL TRANSPORTATION CORP (B2192)': 'LEESEL TRANSPORTATION CORP',

    'LEESEL TRANSPORTATION COR': 'LEESEL TRANSPORTATION CORP',

    # PHILLIP

    'PHILLIPS BUS SERVICE': 'PHILLIP BUS SERVICE INC',

    'Phillip Bus Service, Inc.E': 'PHILLIP BUS SERVICE INC',

    'PHILLIPBUSSERVICE': 'PHILLIP BUS SERVICE INC',

    'PHILLIP BUS CORP (B2192)': 'PHILLIP BUS SERVICE INC',

    'PHILLIPBUSSERVICE': 'PHILLIP BUS SERVICE INC',

    'PHILLIPSBUSSERVICE': 'PHILLIP BUS SERVICE INC',

    'PHILLIP BUS CORP': 'PHILLIP BUS SERVICE INC',

    'PHILLIP BUS SERVICE INC.': 'PHILLIP BUS SERVICE INC',

    # G.V.C.

    'G.V.C. LTD. (B2192)': 'G.V.C. LTD',

    'G.V.C., LTD.': 'G.V.C. LTD',

    'GVC LTD': 'G.V.C. LTD',

    'G.V.C. LTD.': 'G.V.C. LTD',

    # RELIANT

    'RELIANT TRANS INC': 'RELIANT TRANSPORTATION INC',

    'RELIANT TRANSPORTATION INC': 'RELIANT TRANSPORTATION INC',

    # CAREFUL

    'CAREFUL BUS': 'CAREFUL BUS SERVICE INC',

    'CAREFUL BUS SERVICE INC': 'CAREFUL BUS SERVICE INC',

    # PRIDE

    'PRIDE TRANSPORTATION (SCH': 'PRIDE TRANSPORTATION INC',

    'PRIDE TRANSPORTATION (SCH AGE)': 'PRIDE TRANSPORTATION INC',

    'PRIDE TRANSPORTATION SCH': 'PRIDE TRANSPORTATION INC',

    'PRIDE TRANSPORTATION SCH AGE': 'PRIDE TRANSPORTATION INC',

    # THOMAS

    'THOMAS BUSES, INC. (B2321)': 'THOMAS BUSES INC',

    'THOMAS BUSES INC': 'THOMAS BUSES INC',

    'THOMAS BUSES INC (B2192)': 'THOMAS BUSES INC',

    'THOMAS BUSES, INC. (B2321': 'THOMAS BUSES INC',

    'THOMAS BUSES INC.': 'THOMAS BUSES INC',

    # ALINA

    'ALINA': 'ALINA SERVICES CORP',

    'ALINA SERVICES CORP': 'ALINA SERVICES CORP',

    'Alina Services CORP.': 'ALINA SERVICES CORP',

    'ALINA SERVICES CORP.': 'ALINA SERVICES CORP',

    # ALL AMERICAN SCHOOL BUS

    'ALL AMERICAN SCHOOL BUS C': 'ALL AMERICAN SCHOOL BUS COMPANY',

    'ALL AMERICAN SCHOOL BUS CORP.': 'ALL AMERICAN SCHOOL BUS COMPANY',

    'ALL AMERICAN SCHOOL BUS COMPANY': 'ALL AMERICAN SCHOOL BUS COMPANY',

    # SELBY

    'SELBY TRANS CORP': 'SELBY TRANSPORTATION CORP',

    'SELBY TRANSPORTATION CORP': 'SELBY TRANSPORTATION CORP',

    'SELBY TRANSPORTATION': 'SELBY TRANSPORTATION CORP',

    # L & M

    'L & M BUS CORP.': 'L & M BUS CORP',

    'L & M BUS CORP (A)': 'L & M BUS CORP',

    'L & M BUS CORP': 'L & M BUS CORP',

    'L & M BUS CORP A': 'L & M BUS CORP',

    # PIONEER

    'PIONEER TRANSPORTATION CO': 'PIONEER TRANSPORTATION CORP',

    'PIONEER TRANSPORTATION CORP': 'PIONEER TRANSPORTATION CORP',

    # Additional companies - normalize punctuation

    'ALL COUNTY BUS LLC': 'ALL COUNTY BUS LLC',

    'ALLIED TRANSIT CORP.': 'ALLIED TRANSIT CORP',

    'ANOTHER RIDE INC.': 'ANOTHER RIDE INC',

    'B & F SKILLED INC.': 'B & F SKILLED INC',

    'BOBBYS BUS CO. INC.': 'BOBBYS BUS CO INC',

    'BORO TRANSIT INC.': 'BORO TRANSIT INC',

    'EMPIRE CHARTER SERVICE INC': 'EMPIRE CHARTER SERVICE INC',

    'EMPIRE STATE BUS CORP.': 'EMPIRE STATE BUS CORP',

    'GRANDPAS BUS CO. INC.': 'GRANDPAS BUS CO INC',

    'HOYT TRANSPORTATION CORP.': 'HOYT TRANSPORTATION CORP',

    'I & Y TRANSIT CORP': 'I & Y TRANSIT CORP',

    'IC BUS INC.': 'IC BUS INC',

    'JOFAZ TRANSPORTATION INC.': 'JOFAZ TRANSPORTATION INC',

    'LITTLE LINDA BUS CO.INC.': 'LITTLE LINDA BUS CO INC',

    'LITTLE LISA BUS CO. INC.': 'LITTLE LISA BUS CO INC',

    'LITTLE RICHIE BUS SERVICE': 'LITTLE RICHIE BUS SERVICE',

    'LOGAN BUS COMPANY INC.': 'LOGAN BUS COMPANY INC',

    'LOGAN TRANSPORTATION SYSTEMS': 'LOGAN TRANSPORTATION SYSTEMS',

    'LORISSA BUS SERVICE INC.': 'LORISSA BUS SERVICE INC',

    'MAR-CAN TRANSPORT CO. INC': 'MAR-CAN TRANSPORT CO INC',

    'NYC SCHOOL BUS UMBRELLA SERVICES': 'NYC SCHOOL BUS UMBRELLA SERVICES',

    'PENNY TRANSPORTATION': 'PENNY TRANSPORTATION',

    'QUALITY TRANSPORTATION CORP.': 'QUALITY TRANSPORTATION CORP',

    'SNT BUS INC': 'SNT BUS INC',

    'VAN TRANS LLC': 'VAN TRANS LLC',

    'VINNYS BUS SERVICES': 'VINNYS BUS SERVICES',

    'Y & M TRANSIT CORP': 'Y & M TRANSIT CORP',

}

# Apply map

df['Bus\_Company\_Name'] = df['Bus\_Company\_Name'].replace(standardize\_map)

df['Bus\_Company\_Name'] = df['Bus\_Company\_Name'].str.upper()

#dropping the 'Incident\_Number' and 'Schools\_Serviced' columns

df.drop(['Incident\_Number', 'Schools\_Serviced', 'Bus\_No', 'Company\_Code', 'Informed\_On','Last\_Updated\_On','Created\_On'], axis=1, inplace=True)

#showing the Bus\_Company\_Name value\_counts ascending

df['Bus\_Company\_Name'].value\_counts().head(10).sort\_values(ascending=True).plot(kind='barh', title='Top 10 Companies by Incidents')

"""Understanding the bus companies. Ensuring that there are no duplicates in the values"""

data = df['Bus\_Company\_Name'].value\_counts().head(20).sort\_values()

data.plot(kind='barh', figsize=(10, 6))

plt.title('Bus Company Name Distribution')

plt.xlabel('Count')

plt.ylabel('Bus Company Name')

plt.tight\_layout()

plt.show()

df['Breakdown\_or\_Running\_Late'].value\_counts().plot(kind='bar', title='Breakdown vs Running Late')

df['Reason'].value\_counts().head(10).plot(kind='bar', title='Top 10 Delay Reasons')

plt.xticks(rotation=45, ha='right')   # rotate x labels 45° and right-align

plt.tight\_layout()

plt.show()

"""According to the dataset data dictionary category "Other" means: "to be selected by the reporting bus vendor when the delay cannot be classified within the available categories"

"""

df.shape

"""Even after extensive cleaning and standardization, the dataset remains substantial in size, with more than 228K records.

Adjusting Borough information. Since the data set includes data from New York State (City + Counties), we will only focus on New York City. This means removing the counties from the dataset.

"""

#Incidents Trend by Hour of Day

df['Occurred\_On'] = pd.to\_datetime(df['Occurred\_On'], errors='coerce')

military\_hrs = df['Occurred\_On'].dt.hour.value\_counts().sort\_index()

plt.figure(figsize=(12, 6), facecolor='lightgray')

ax = plt.gca()

ax.set\_facecolor('lightgray')

plt.plot(military\_hrs.index, military\_hrs.values, marker='o', linestyle='-', color='Red')

plt.title('Incidents Trend by Hour of Day', fontsize=16, color='black')

plt.xlabel('Hour of Day', fontsize=12, color='black')

plt.ylabel('Number of Incidents', fontsize=12, color='black')

plt.xticks(range(0, 24), color='black')

plt.yticks(color='black')

plt.grid(True, color='white')

plt.tight\_layout()

plt.show()

df = df.loc[:, ~df.columns.isin(['Delay\_Bin'])]

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

import seaborn as sns

#Create a copy of the main DataFrame

df\_encoded = df.copy()

#Exclude unwanted columns

columns\_to\_exclude = ['Busbreakdown\_ID', 'School\_Year', 'Route\_Number', 'How\_Long\_Delayed','Breakdown\_or\_Running\_Late']

df\_encoded = df\_encoded.drop(columns=columns\_to\_exclude, errors='ignore')

#Encode categorical columns

for col in df\_encoded.columns:

    if df\_encoded[col].dtype == 'object':

        df\_encoded[col] = LabelEncoder().fit\_transform(df\_encoded[col].astype(str))

#Compute correlation matrix

corr\_matrix = df\_encoded.corr()

#Plot correlation heatmap

plt.figure(figsize=(20, 8))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Heatmap')

plt.tight\_layout()

plt.show()

"""Getting a copy of the dataframe for classification modeling given we don't want re-use trained data.

#Modeling

##Modling (Regression and Classification)

Here we use random forest regressor model because it handles numerical and categorical data very well.

"""

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

df['Occurred\_On'] = pd.to\_datetime(df['Occurred\_On'], errors='coerce')

df['Hour'] = df['Occurred\_On'].dt.hour

df['Month'] = df['Occurred\_On'].dt.month

df['Day\_of\_Week'] = df['Occurred\_On'].dt.dayofweek

# Selected Features and Target

features = ['Route\_Number\_Clean', 'Run\_Type', 'Reason', 'Borough', 'Bus\_Company\_Name',

            'School\_Year', 'Number\_Of\_Students\_On\_The\_Bus',

            'Has\_Contractor\_Notified\_Schools', 'Has\_Contractor\_Notified\_Parents',

            'Have\_You\_Alerted\_OPT', 'Hour', 'Month', 'Day\_of\_Week']

target = 'Delay\_Minutes'

# Filter dataset

df\_model = df[features + [target]].dropna(subset=[target])

df\_encoded = df\_model.copy()

le\_dict = {}

for col in features:

    if df\_encoded[col].dtype == 'object':

        le = LabelEncoder()

        df\_encoded[col] = le.fit\_transform(df\_encoded[col].astype(str))

        le\_dict[col] = le # Store the fitted encoder

X = df\_encoded[features]

y = df\_encoded[target]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = RandomForestRegressor(random\_state=42, n\_estimators=100)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

print("Root Mean Squared Error:", rmse)

print("R2 Score:", r2\_score(y\_test, y\_pred))

# Feature importance

plt.figure(figsize=(10,5))

plt.barh(features, model.feature\_importances\_)

plt.title("Feature Importance")

plt.show()

"""###XGBoost Only Model (Higher Breakdown Predictions)"""

df1 = df.copy()

# Commented out IPython magic to ensure Python compatibility.

# %pip install category\_encoders

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import classification\_report, precision\_recall\_curve

from imblearn.over\_sampling import SMOTE

from category\_encoders import TargetEncoder

from xgboost import XGBClassifier

import numpy as np

import pandas as pd

# === Feature Engineering ===

df1['Occurred\_On'] = pd.to\_datetime(df1['Occurred\_On'], errors='coerce')

df1['Hour'] = df1['Occurred\_On'].dt.hour

df1['Month'] = df1['Occurred\_On'].dt.month

df1['Day\_of\_Week'] = df1['Occurred\_On'].dt.dayofweek

df1['Is\_Weekend'] = df1['Day\_of\_Week'].isin([5, 6]).astype('int8')

df1['Is\_Rush\_Hour'] = df1['Hour'].isin([7, 8, 9, 15, 16, 17]).astype('int8')

df1['Contract\_Notified\_Schools'] = df1['Has\_Contractor\_Notified\_Schools'].map({'Yes': 1, 'No': 0}).fillna(0).astype('int8')

df1['Contract\_Notified\_Parents'] = df1['Has\_Contractor\_Notified\_Parents'].map({'Yes': 1, 'No': 0}).fillna(0).astype('int8')

df1['Alerted\_OPT'] = df1['Have\_You\_Alerted\_OPT'].map({'Yes': 1, 'No': 0}).fillna(0).astype('int8')

# Map target to numeric, preserve original labels

df1 = df1[df1['Breakdown\_or\_Running\_Late'].isin(['Breakdown', 'Running Late'])]  # filter valid

df1['Breakdown\_Binary'] = df1['Breakdown\_or\_Running\_Late'].map({'Breakdown': 1, 'Running Late': 0}).astype('int8')

target = 'Breakdown\_Binary'

# === Selected Features ===

features = [

    'School\_Year', 'Number\_Of\_Students\_On\_The\_Bus','Run\_Type',

    'Reason','Borough', 'Hour', 'Month', 'Day\_of\_Week',

    'Is\_Weekend', 'Is\_Rush\_Hour','School\_Age\_or\_PreK',

    'Bus\_Company\_Name','Route\_Number\_Clean',

    'Contract\_Notified\_Schools','Contract\_Notified\_Parents','Alerted\_OPT',

]

df1 = df1.dropna(subset=features + [target])

X = df1[features]

y = df1[target]

# === Split Data ===

X\_temp, X\_test, y\_temp, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=42)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_temp, y\_temp, test\_size=0.25, stratify=y\_temp, random\_state=42)

# === Target Encoding ===

cat\_features = [col for col in features if df1[col].dtype == 'object']

encoder = TargetEncoder(cols=cat\_features)

X\_train\_enc = encoder.fit\_transform(X\_train, y\_train)

X\_val\_enc = encoder.transform(X\_val)

X\_test\_enc = encoder.transform(X\_test)

# === Scaling ===

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train\_enc)

X\_val\_scaled = scaler.transform(X\_val\_enc)

X\_test\_scaled = scaler.transform(X\_test\_enc)

# === Oversample with SMOTE ===

smote = SMOTE(sampling\_strategy=0.8, random\_state=42, k\_neighbors=3)

X\_train\_bal, y\_train\_bal = smote.fit\_resample(X\_train\_scaled, y\_train)

# === Train XGBoost Model ===

model = XGBClassifier(

    n\_estimators=200,

    max\_depth=6,

    learning\_rate=0.05,

    scale\_pos\_weight=(y\_train\_bal == 0).sum() / (y\_train\_bal == 1).sum(),

    evalin\_metric='logloss',

    random\_state=42

)

model.fit(X\_train\_bal, y\_train\_bal)

# === Validation Performance ===

y\_val\_pred = model.predict(X\_val\_scaled)

print("\nValidation Report:\n", classification\_report(y\_val, y\_val\_pred))

# === Test Set Prediction ===

y\_test\_pred = model.predict(X\_test\_scaled)

print("\nTest Report:\n", classification\_report(y\_test, y\_test\_pred))

val\_report = classification\_report(y\_val, y\_val\_pred, output\_dict=True)

test\_report = classification\_report(y\_test, y\_test\_pred, output\_dict=True)

val\_df = pd.DataFrame(val\_report).transpose().drop('accuracy')

test\_df = pd.DataFrame(test\_report).transpose().drop('accuracy')

styled\_val = val\_df.style.background\_gradient(cmap="Blues").format("{:.2f}")

styled\_test = test\_df.style.background\_gradient(cmap="Greens").format("{:.2f}")

display(styled\_val)

display(styled\_test)

# Feature importance and displaying result

feature\_importance = pd.DataFrame({

    'feature': X\_train.columns,

    'importance': model.feature\_importances\_

}).sort\_values('importance', ascending=False)

# Showing the top 10 features

top\_features = feature\_importance.nlargest(10, 'importance').copy()

top\_features.sort\_values('importance', ascending=False, inplace=True)

plt.figure(figsize=(10, 6))

ax = sns.barplot(

    x='importance',

    y='feature',

    data=top\_features,

    hue='feature',

    palette='viridis',

    dodge=False

)

# Add percentage labels to each bar

for i, (value, name) in enumerate(zip(top\_features['importance'], top\_features['feature'])):

    ax.text(value + 0.001, i, f"{value:.2%}", va='center')

plt.title('Top 10 Feature Importances')

plt.xlabel('Importance (%)')

plt.ylabel('Feature')

plt.tight\_layout()

plt.show()

#Breakdown Rate by Reason

if 'Reason' in df1.columns and 'Breakdown\_or\_Running\_Late' in df1.columns:

    # Calculate breakdown rate per Reason

    breakdown\_rate = (

        df1[df1['Breakdown\_or\_Running\_Late'] == 'Breakdown']

        .groupby('Reason')

        .size()

    )

    # Calculate total incidents per Reason

    total\_incidents = df1.groupby('Reason').size()

    # Proportion Breakdown per Reason

    breakdown\_prop = (breakdown\_rate / total\_incidents).fillna(0).sort\_values(ascending=False)

    # Plot Breakdown Rate

    plt.figure(figsize=(10, 6))

    ax = sns.barplot(

        x=breakdown\_prop.values,

        y=breakdown\_prop.index,

        hue=breakdown\_prop.index,

        palette=sns.color\_palette('Spectral', n\_colors=len(breakdown\_prop))

    )

    plt.title('Breakdown Rate by Reason')

    plt.xlabel('Proportion of Breakdowns')

    plt.ylabel('Reason')

    plt.xlim(0, 1)

    plt.gca().xaxis.set\_major\_formatter(plt.FuncFormatter(lambda x, \_: f'{x:.0%}'))

    # Add percentage labels

    for i, val in enumerate(breakdown\_prop.values):

        ax.text(val + 0.01, i, f"{val:.1%}", va='center')

    plt.tight\_layout()

    plt.show()

#Running Late Rate by Reason

if 'Reason' in df1.columns and 'Breakdown\_or\_Running\_Late' in df1.columns:

    # Calculate running late rate per Reason

    running\_late\_rate = (

        df1[df1['Breakdown\_or\_Running\_Late'] == 'Running Late']

        .groupby('Reason')

        .size()

    )

    # Calculate total incidents per Reason

    total\_incidents = df1.groupby('Reason').size()

    # Proportion Running Late per Reason

    running\_late\_prop = (running\_late\_rate / total\_incidents).fillna(0).sort\_values(ascending=False)

    # Plot Running Late Rate

    plt.figure(figsize=(10, 7))

    ax = sns.barplot(

        x=running\_late\_prop.values,

        y=running\_late\_prop.index,

        hue=running\_late\_prop.index,

        palette=sns.color\_palette('RdYlBu', n\_colors=len(running\_late\_prop))

    )

    plt.title('Running Late Rate by Reason')

    plt.xlabel('Proportion of Running Late')

    plt.ylabel('Reason')

    plt.xlim(0, 1)

    plt.gca().xaxis.set\_major\_formatter(plt.FuncFormatter(lambda x, \_: f'{x:.0%}'))

    # Add percentage labels

    for i, val in enumerate(running\_late\_prop.values):

        ax.text(val + 0.01, i, f"{val:.1%}", va='center')

    plt.tight\_layout()

    plt.show()

df1['Reason'].value\_counts()

"""Getting the counts for total incidents and then by breakdown and running late to see if the model is giving us the expected results"""

# Total reason counts

total\_counts = df1['Reason'].value\_counts()

# Breakdown counts

breakdown\_counts = df1[df1['Breakdown\_or\_Running\_Late'] == 'Breakdown']['Reason'].value\_counts()

# Running Late counts

late\_counts = df1[df1['Breakdown\_or\_Running\_Late'] == 'Running Late']['Reason'].value\_counts()

# Combine into one DataFrame

summary\_df = pd.DataFrame({

    'Total': total\_counts,

    'Breakdown': breakdown\_counts,

    'Running Late': late\_counts

}).fillna(0).astype(int)

# Sort by total in descending order

summary\_df = summary\_df.sort\_values(by='Total', ascending=False)

summary\_df

"""# Streamlit Frontend Application"""

import joblib

# Saving model, encoder, scaler

joblib.dump(model, 'xgb\_model.pkl')

joblib.dump(encoder, 'target\_encoder.pkl')

joblib.dump(scaler, 'scaler.pkl')

code = '''

# streamlit\_app.py

import streamlit as st

import numpy as np

import pandas as pd

import joblib

# Load saved artifacts

model = joblib.load('xgb\_model.pkl')

encoder = joblib.load('target\_encoder.pkl')

scaler = joblib.load('scaler.pkl')

st.title("🚌 Predict School Bus Delay and Breakdown")

st.markdown("Enter the following information:")

# === Input fields ===

school\_year = st.selectbox("School Year", ['2024-2025', '2023-2024', '2022-2023'])

num\_students = st.slider("Number of Students on the Bus", 0, 80, 25)

run\_type = st.selectbox("Run Type", [

    'Pre-K/EI', 'Special Ed AM Run', 'General Ed AM Run',

    'Special Ed PM Run', 'General Ed PM Run',

    'General Ed Field Trip', 'Special Ed Field Trip'

])

reason = st.selectbox("Reason", [

    'Heavy Traffic', 'Other', 'Mechanical Problem', "Won`t Start", 'Flat Tire',

    'Problem Run', 'Accident', 'Late return from Field Trip',

    'Weather Conditions', 'Delayed by School'

])

borough = st.selectbox("Borough", ['Brooklyn', 'Bronx', 'Staten Island', 'Queens', 'Manhattan'])

hour = st.slider("Hour of Day", 0, 23, 8)

month = st.slider("Month", 1, 12, 5)

day\_of\_week = st.slider("Day of Week (0=Mon, 6=Sun)", 0, 6, 2)

is\_weekend = int(day\_of\_week in [5, 6])

is\_rush\_hour = int(hour in [7, 8, 9, 15, 16, 17])

school\_age\_or\_prek = st.selectbox("Student Type", ['Pre-K', 'School-Age'])

bus\_company = st.selectbox("Bus Company", ['L & M BUS CORP', 'BORO TRANSIT INC',

        'PIONEER TRANSPORTATION CORP', 'PRIDE TRANSPORTATION INC',

        'ALLIED TRANSIT CORP', 'CONSOLIDATED BUS TRANSIT INC',

        'LOGAN BUS COMPANY INC', 'LITTLE RICHIE BUS SERVICE',

        'SNT BUS INC', 'DON THOMAS BUSES INC', 'Other',

        'LORINDA ENTERPRISES LTD', 'HOYT TRANSPORTATION CORP',

        'G.V.C. LTD', 'PHILLIP BUS SERVICE INC', 'VAN TRANS LLC',

        'ALL AMERICAN SCHOOL BUS COMPANY', 'EMPIRE STATE BUS CORP',

        'CAREFUL BUS SERVICE INC', 'NYC SCHOOL BUS UMBRELLA SERVICES',

        'ALINA SERVICES CORP', 'EMPIRE CHARTER SERVICE INC',

        'LEESEL TRANSPORTATION CORP', 'QUALITY TRANSPORTATION CORP'

])

# === Route number ===

route\_options = [

    '3002A', 'K064', 'X184', 'R1218', 'L343', 'Q370', 'X028', 'P863',

    'Q742', 'R1203', 'M1084', 'M859', 'Q664', 'P102', 'K424', '3406A',

    'PS200', 'R349', 'L141', 'R1030', 'R1303', 'Q2901', 'M154', 'L535',

    'X2359', 'Q818', 'X049', 'M136', 'Q337', 'R1043', 'P584', 'P783',

    'L400', 'R007', 'X560', 'R1121', 'M124', 'M9045', 'M626', 'X2318',

    'X112', 'P671', 'R1324', 'R1318', 'R1078', 'K1436', 'M798', 'P677',

    'K032', 'K972', '3606A', 'X2269', 'L610', 'B0203A', 'M729', 'Q840',

    'Y136', 'X202', 'X744', 'X2138', 'N619', 'M884', 'PB4', '2B',

    'X174', 'M717', 'K1433', 'K880', 'N292', 'R1102', 'Q3082', 'L147',

    'B0230A', 'P893', 'Q121', 'R001', 'K008', 'R014', 'K1412', 'R1185',

    'R032', 'K243', 'EF4', 'B0919Z', 'R9091', 'P692', 'M111', 'K642',

    'P748', 'Q2994', 'P638', 'R9063', 'Y182', 'R9185', 'R9198',

    'R9227', 'R9316', 'R9164', 'R006', 'R023', 'L697', 'X175', 'R1068',

    'Q8375', 'M1184', 'X058', 'K012', 'M1121', 'K1429', 'Q388', 'X532',

    '3315A', '3318A', 'X631', 'M116', 'P885', 'PK1', 'Y234', 'X099',

    'Y200', 'M1036', 'K505', 'N210', 'Q906', 'K014', 'Y254', 'M783',

    'N035', 'B0221A', 'Q868', 'M956', 'K168', 'K144', 'K696', 'R9030',

    'R9043', 'R9117', 'N563', 'K981', 'X2034', '2', 'B0204A', 'R9130',

    'R9175', 'R9219', 'R1015', 'P704', 'L747', 'K109', 'Q954',

    'Other'  # allow manual entry if not in list

]

# Create searchable dropdown

route\_choice = st.selectbox("Route Number (type to search or select)", route\_options)

# Fallback to manual input

if route\_choice == "Other":

    route\_number\_clean = st.text\_input("Enter Route Number Manually")

else:

    route\_number\_clean = route\_choice

contract\_notified\_schools = st.checkbox("Contractor Notified Schools?")

contract\_notified\_parents = st.checkbox("Contractor Notified Parents?")

alerted\_opt = st.checkbox("Alerted OPT?")

# === Assemble feature input ===

input\_dict = {

    'School\_Year': school\_year,

    'Number\_Of\_Students\_On\_The\_Bus': num\_students,

    'Run\_Type': run\_type,

    'Reason': reason,

    'Borough': borough,

    'Hour': hour,

    'Month': month,

    'Day\_of\_Week': day\_of\_week,

    'Is\_Weekend': is\_weekend,

    'Is\_Rush\_Hour': is\_rush\_hour,

    'School\_Age\_or\_PreK': school\_age\_or\_prek,

    'Bus\_Company\_Name': bus\_company,

    'Route\_Number\_Clean': route\_number\_clean,

    'Contract\_Notified\_Schools': int(contract\_notified\_schools),

    'Contract\_Notified\_Parents': int(contract\_notified\_parents),

    'Alerted\_OPT': int(alerted\_opt)

}

input\_df = pd.DataFrame([input\_dict])

# === Encode, scale, and predict ===

input\_encoded = encoder.transform(input\_df)

input\_scaled = scaler.transform(input\_encoded)

proba = model.predict\_proba(input\_scaled)[0][1]  # Probability of Breakdown

# Determine which class has higher probability

if proba >= 0.5:

    label = "🚨 Breakdown"

    confidence = proba

else:

    label = "⏱️ Running Late"

    confidence = 1 - proba

# === Display result ===

st.markdown(f"### Prediction: \*\*{label}\*\*")

st.progress(float(confidence))

st.caption(f"Confidence: {confidence\*100:.1f}%")'''

# Save to a .py file

with open("streamlit\_app.py", "w") as f:

    f.write(code)

"""# This is for the PowerBI dashboard"""

# XGBoost Predicted Probablity Values --- left are 0 values and right are the 1 values predicted

pred\_prob\_xgboost = model.predict\_proba(X\_test\_scaled)

# XGBoost This is just the predicted probability to the whole number

y\_test\_pred

def Column(matrix, i):

    return [row[i] for row in matrix]

[col.item() for col in Column(pred\_prob\_xgboost, 0)]

output = X\_test.iloc[:43210].copy()

output['XGPredict - Running Late or Breakdown'] = pd.Series(y\_test\_pred[:43210], index=output.index)

output['XGPredict - Probability vs Running Late'] = pd.Series(pred\_prob\_xgboost[:43210, 0], index=output.index)

output['XGPredict -  Classification Results: Running Late or Breakdown'] = 'Empty'

output.loc[output['XGPredict - Running Late or Breakdown'] == 1, 'XGPredict -  Classification Results: Running Late or Breakdown'] = 'Breakdown'

output.loc[output['XGPredict - Running Late or Breakdown'] == 0, 'XGPredict -  Classification Results: Running Late or Breakdown'] = 'Running Late'

output\_class = output

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

df['Occurred\_On'] = pd.to\_datetime(df['Occurred\_On'], errors='coerce')

df['Hour'] = df['Occurred\_On'].dt.hour

df['Month'] = df['Occurred\_On'].dt.month

df['Day\_of\_Week'] = df['Occurred\_On'].dt.dayofweek

features = ['Route\_Number\_Clean', 'Run\_Type', 'Reason', 'Borough', 'Bus\_Company\_Name',

            'School\_Year', 'Number\_Of\_Students\_On\_The\_Bus',

            'Has\_Contractor\_Notified\_Schools', 'Has\_Contractor\_Notified\_Parents',

            'Have\_You\_Alerted\_OPT', 'Hour', 'Month', 'Day\_of\_Week']

target = 'Delay\_Minutes'

df\_model = df[features + [target]].dropna(subset=[target])

df\_encoded = df\_model.copy()

le\_dict = {}

for col in features:

    if df\_encoded[col].dtype == 'object':

        le = LabelEncoder()

        df\_encoded[col] = le.fit\_transform(df\_encoded[col].astype(str))

        le\_dict[col] = le

X = df\_encoded[features]

y = df\_encoded[target]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = RandomForestRegressor(random\_state=42, n\_estimators=100)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

print("Root Mean Squared Error:", rmse)

print("R2 Score:", r2\_score(y\_test, y\_pred))

plt.figure(figsize=(10,5))

plt.barh(features, model.feature\_importances\_)

plt.title("Feature Importance")

plt.show()

output = X\_test.copy()

output['True\_Delay\_Minutes'] = y\_test.values

output['Predicted\_Delay\_Minutes'] = y\_pred

output['Error'] = output['True\_Delay\_Minutes'] - output['Predicted\_Delay\_Minutes']

threshold = 3

output['Error\_Type'] = 'Accurate'

output.loc[output['Error'] > threshold, 'Error\_Type'] = 'Underestimated'

output.loc[output['Error'] < -threshold, 'Error\_Type'] = 'Overestimated'

output.reset\_index(drop=True, inplace=True)

reg\_output = output

output\_class = output\_class.reset\_index(drop=True)

reg\_output = reg\_output.reset\_index(drop=True)

combined = pd.concat([

    output\_class,

    reg\_output[['True\_Delay\_Minutes', 'Predicted\_Delay\_Minutes', 'Error', 'Error\_Type']]

], axis=1)

)