Data Bootcamp Final Project

Can We Predict NYC Subway Delays?

Modeling Additional Platform Time (APT) with MTA Open Data

Will Wu – John Yue

December 16, 2025

Abstract

We predict next month's NYC subway delay level using *Additional Platform Time (APT)* at the subway-line level. We merge two MTA open datasets (APT metrics and delay incidents) and build simple time-series features like lags, rolling averages, and calendar indicators. We compare a few regression models with a chronological train/test split; our best model (HistGB) reaches **MAE = 0.193** on the held-out months. Overall, this is a clean baseline that shows what recent history helps when forecasting APT.

1 Introduction

NYC subway delays affect real people: they change commute time, crowding, and how reliable the system feels. This project asks:

> **Can we predict next month's APT for a given subway line using recent delay history and incident patterns?**

APT is useful because it compresses "delay" into one number (extra minutes spent waiting on the platform).

2 Dataset and API Access

We use two public datasets from New York State Open Data (Socrata exports), downloaded as CSVs and read with `pandas`:

- **Customer journey metrics (Beginning 2023):** monthly APT by subway line (our target).
- **Subway delay incidents (Beginning 2020):** monthly incident counts by line (and sometimes category/day type).

3 Setup and Cleaning

We standardize both datasets into the same keys: **month** and **line**. Main cleaning steps were: parse dates into month buckets, keep the columns we actually use, and handle missing values (mainly by dropping months with insufficient history for lag features). After that, we merge APT with incident totals so each row represents "one line in one month."

4 Building Monthly Incident Features

Incident data can show up at different levels (line, category, day type), so we aggregate to a monthly total at the line level. To keep prediction leakage-free, we also create lagged incident features (e.g., last month and two months ago) before merging them into the modeling table.

5 Feature Engineering for Prediction

Our features are intentionally simple:

- **APT history:** `apt_lag_1`, `apt_lag_2`, and a 3-month rolling mean.
- **Incidents:** lagged incident counts.
- **Calendar/line info:** month/quarter/season and one-hot encoded line labels.

All lag/rolling features are computed within each line (and shifted) so they only use past months.

6 Exploratory Data Analysis

Before modeling, we visualize APT to understand its scale and how it changes across time and lines.

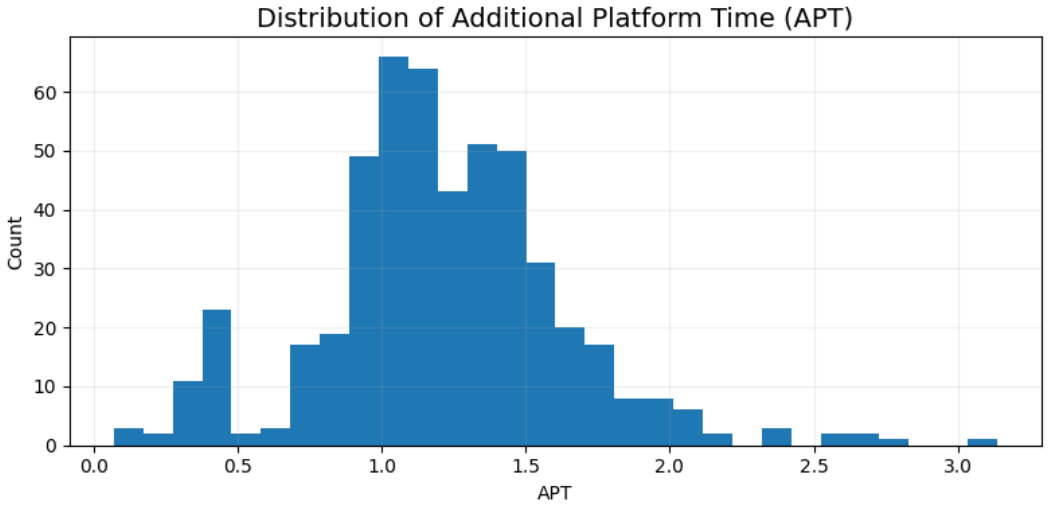

Figure 1: Distribution of Additional Platform Time (APT).

Figure 1 shows that APT is mostly concentrated around roughly **1.0 to 1.5 minutes**, with a right tail that stretches past **2.0** and up to around **3.1**. This shape matters for modeling: most months are "normal," but a small number of higher-delay months can noticeably increase RMSE.

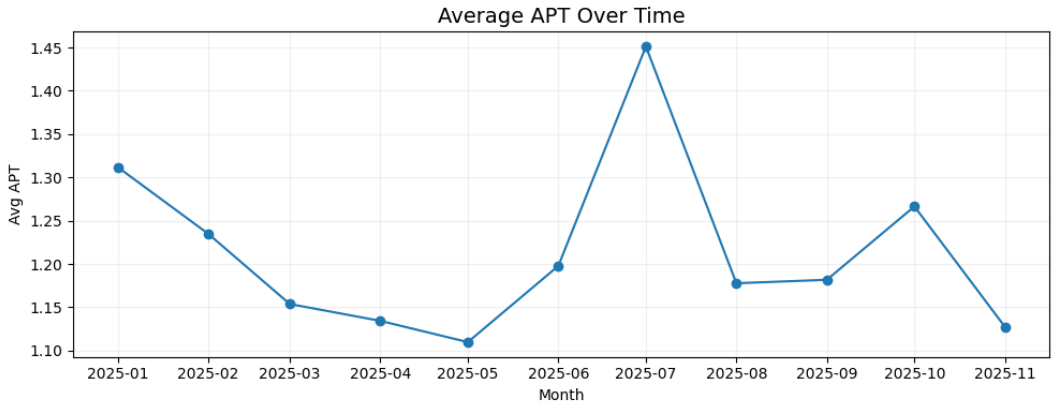

Figure 2: Average APT over time (monthly mean).

In Figure 2, average APT falls from January to May (about 1.31 down to about 1.11), then rises in early summer and peaks in **July** (around **1.45**). After July, it drops back near 1.18 in August/September, bumps up in October, and then declines again in November. Even with limited months, this suggests there may be *seasonal or operational* effects that a model can partially capture using month/season features and recent lags.

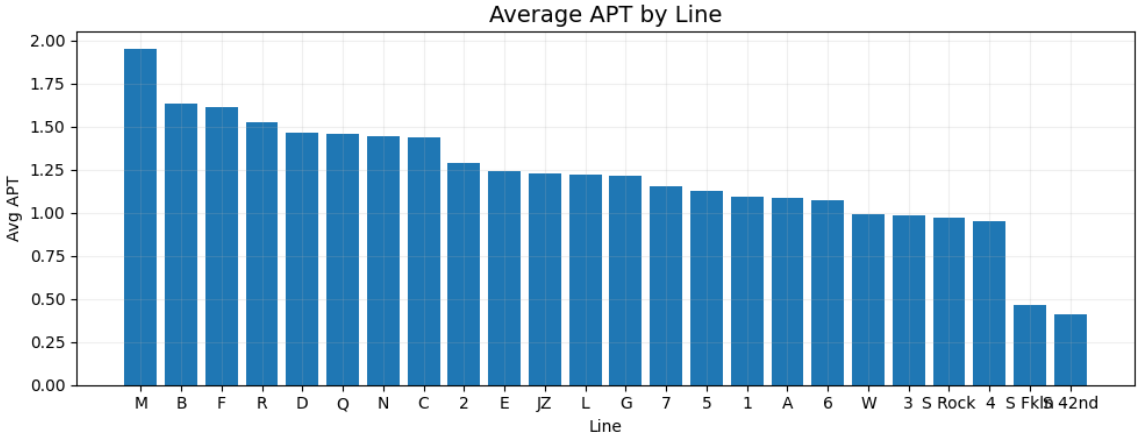

Figure 3: Average APT by subway line.

Figure 3 shows clear differences across lines. For example, the **M** line has the highest average APT (close to 2.0), while several shuttle-type services (e.g., **SFkln** and **H 42nd**) sit far lower (well under 0.6). This supports including the line identifier as a categorical feature, because "typical" delay levels differ by service pattern.

7 Modeling and Evaluation

Train/test split. We split the dataset chronologically by month: the first 80% of months are used for training and the last 20% are held out for testing. This mimics a real deployment setting where we predict future performance from past history.

Preprocessing. We use a single preprocessing pipeline for all models:

- Numeric features: median imputation.

- Categorical features: most-frequent imputation + one-hot encoding (with `handle_unknown = ''ignore''` so unseen line labels do not crash the model).

Models compared. We evaluate:

- Baseline (predict `apt_lag_1`)
- Ridge regression
- Random Forest
- Histogram-based Gradient Boosting (HistGB)

Table 1: Model performance on held-out future months. Lower MAE/RMSE is better; higher R^2 is better.

Model	**MAE**	**RMSE**	R^2
Baseline (lag1)	0.201279	0.285367	0.312922
Ridge	0.260231	0.337060	0.041455
RandomForest	0.197058	0.264868	0.408090
HistGB	**0.193131**	**0.262690**	**0.417782**

Two results stand out. First, the simple baseline is already strong, which tells us that APT is *highly autocorrelated* (last month is a good guess for next month). Second, the tree-based models improve on the baseline, with HistGB performing best overall. Ridge performs poorly here because the relationship between APT and the engineered features is not purely linear, and the interactions between line and time are better handled by non-linear models.

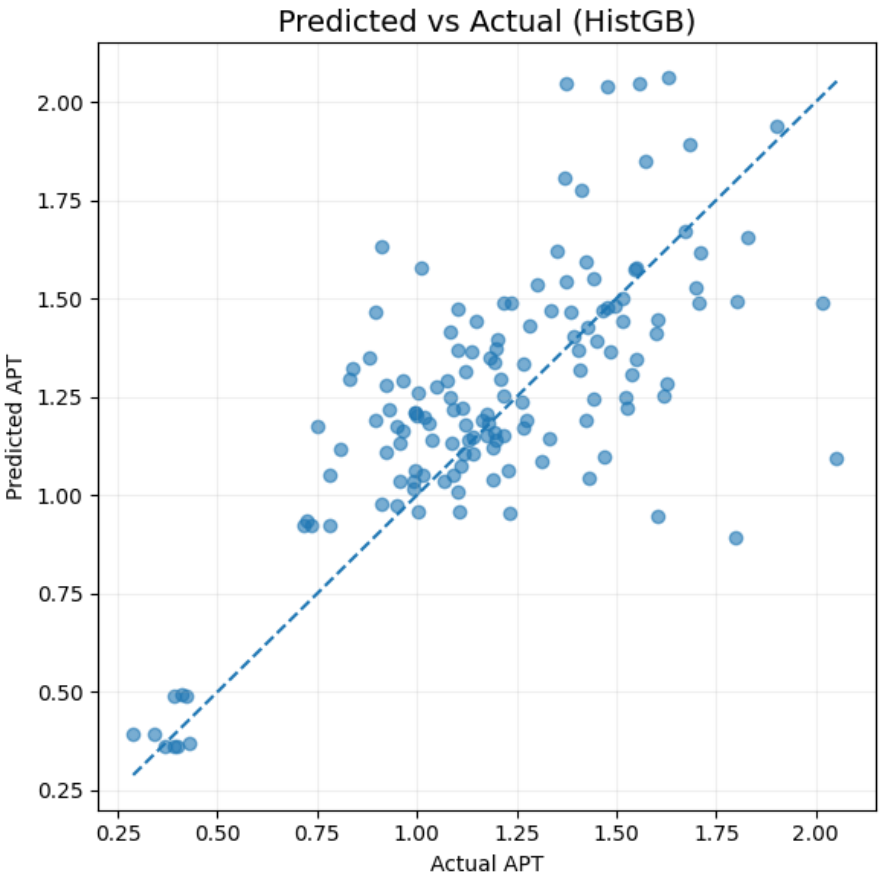

Figure 4: Predicted vs. actual APT for the best model (HistGB).

Figure 4 shows that most points cluster near the diagonal, which is what we want. The model generally tracks typical months well, but there are a few noticeable misses when actual APT is very high (around 1.8 to 2.1). In other words, extreme delay months are harder to predict, which matches the long right tail we saw in the distribution.

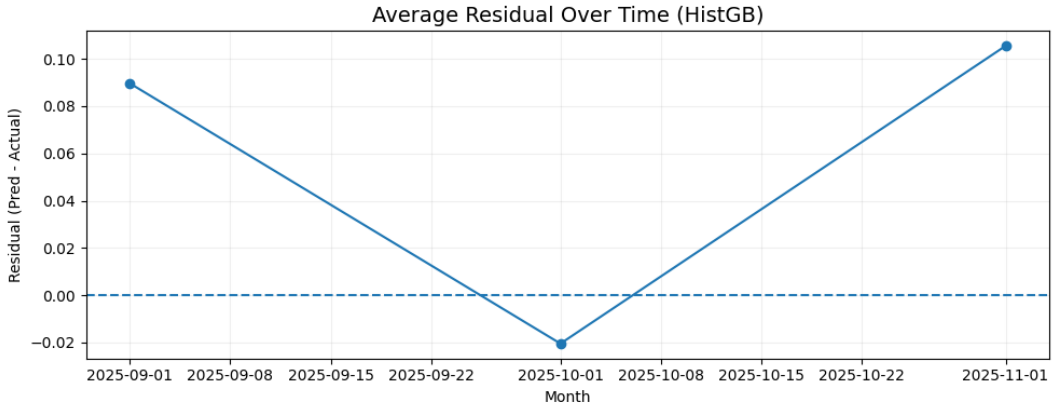

Figure 5: Average residual over time for the best model (HistGB).

In Figure 5, the mean residual changes slightly across the test months. The model overpredicts

in September and November (positive residuals), and is slightly under in October. This is not a huge effect, but it suggests there can be month-to-month shifts that a purely lag-based model will not perfectly capture.

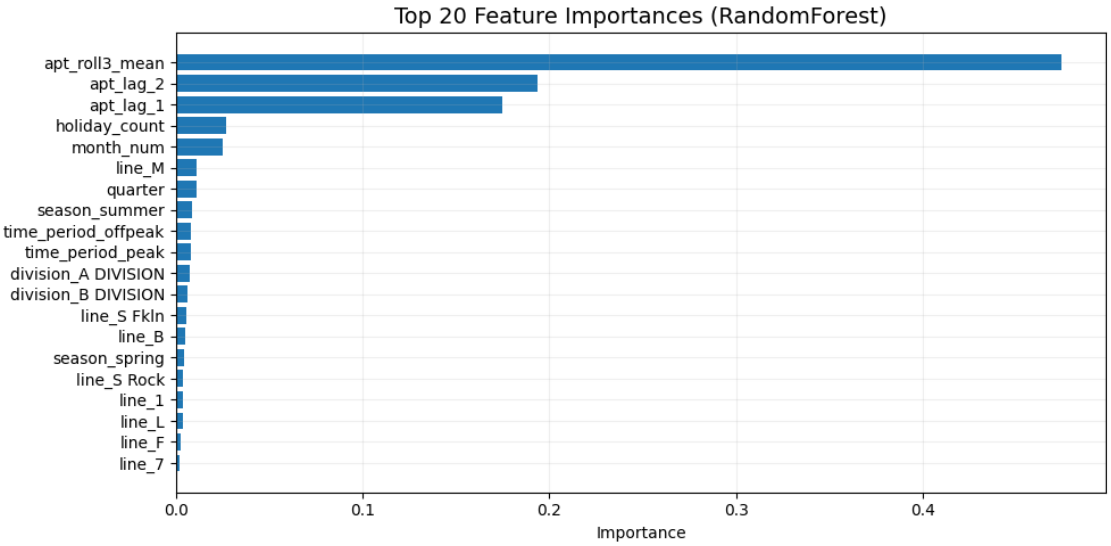

Figure 6: Top 20 feature importances (Random Forest).

Although our best model is HistGB, Figure 6 is still useful as a check. The Random Forest relies most heavily on **recent APT history** (3-month rolling mean and the 1–2 month lags), which is exactly what we expected. Holiday count and calendar features (month/quarter) matter a bit, and line indicators also show up, but they are secondary compared to what has been happening recently.

8 Limitations and Next Steps

- **Limited inputs.** We do not include weather, service changes, planned work, or special events, which could explain some spikes.
- **Small sample per line.** Monthly data means we have relatively few observations, so results can be sensitive to the time window.
- **Hard-to-predict extremes.** The long right tail in APT makes extreme-delay months the most difficult cases.

If we extended this project, the first upgrades would be: add external signals (weather/service advisories) and evaluate a model that predicts at a finer time granularity (weekly or daily).

9 Reproducibility and Code Readability

We kept the notebook easy to rerun and grade:

- Data downloads are cached (no repeated API calls).
- Modeling uses a single, consistent preprocessing pipeline so comparisons are fair.