# NYC Rideshare Demand Forecasting

A Machine Learning Approach to Zone-Level Prediction

*AI-generated technical companion to the project notebooks. Provides an expanded narrative of the methodology, decisions, and results documented in the analysis.*

Technical Analysis Report

Data Acquisition • Validation • Exploration • Forecasting

K Flowers

January 2026

# Table of Contents

## Executive Summary

This report documents a complete data science pipeline for forecasting daily rideshare demand across New York City taxi zones. The project processed 684 million For-Hire Vehicle High Volume (FHVHV) trip records spanning January 2022 through December 2024—primarily Uber and Lyft trips—and developed zone-level, one-day-ahead demand forecasts using machine learning. The work was organized as a four-notebook pipeline covering data acquisition, validation, exploratory analysis, and model development, with explicit data handoffs between each stage to ensure full reproducibility.

The analysis revealed that weekly seasonality is the dominant demand signal across NYC, with weekend ridership averaging 13% higher than weekdays and summer months showing modest lows. After aggregating trip records to daily zone-level counts and computing zone-to-global correlation coefficients, 195 of 256 active zones (76%) were identified as sharing consistent demand patterns. These high-correlation zones accounted for 82% of total trip volume, supporting a shared modeling approach rather than requiring zone-specific models.

Three forecasting models were compared on a pilot zone (Central Harlem, Zone 41): a Seasonal Naive baseline using 7-day lag, Facebook Prophet configured for weekly seasonality, and XGBoost with engineered lag and rolling features. XGBoost achieved the best performance at 5.1% MAPE, outperforming the Seasonal Naive baseline (7.1% MAPE) and substantially outperforming Prophet (13.2% MAPE). When scaled across all 195 high-correlation zones, XGBoost achieved an average 6.4% MAPE, with 97% of zones meeting the sub-10% MAPE performance target.

The project demonstrates proficiency in large-scale data engineering (DuckDB for memory-efficient processing of 18GB datasets), systematic validation pipelines, time-series analysis, and scalable machine learning deployment. Key limitations include the one-day-ahead forecast horizon and the exclusion of external demand drivers such as weather and events.

# 1. Introduction

## 1.1 Business Context

Demand forecasting is foundational to operations planning in transportation. For rideshare platforms like Uber and Lyft, accurate short-term forecasts enable driver allocation, surge pricing calibration, and capacity planning. From a driver's perspective, reliable demand forecasts reduce uncertainty about when and where to position for pickups, contributing to schedule stability and earnings predictability. From an operations standpoint, the difference between a 15% MAPE forecast and a 6% MAPE forecast translates directly into fewer empty vehicles circling neighborhoods, shorter passenger wait times, and more efficient fleet utilization.

New York City represents a particularly challenging forecasting environment. The city's 263 taxi zones span dramatically different demand profiles: Manhattan's Midtown can see over 11,000 trips per day, while some outer-borough zones average fewer than 20. Airport zones, entertainment districts, and transit hubs exhibit demand patterns driven by flight schedules, event calendars, and commuter flows rather than the general weekly rhythms that dominate residential and commercial areas. Any forecasting approach must account for this structural heterogeneity rather than assuming a single model fits all zones.

## 1.2 Dataset Overview

The project used publicly available data from the NYC Taxi and Limousine Commission (TLC), specifically the For-Hire Vehicle High Volume (FHVHV) trip records. This dataset captures every trip dispatched by high-volume for-hire services—in practice, Uber (license HV0003, 73.1% of trips) and Lyft (license HV0005, 26.9%). Each record includes pickup and dropoff timestamps, pickup and dropoff zone IDs, trip distance, trip duration, and fare components. The analysis period spanned 36 months (January 2022 through December 2024), encompassing 684,376,551 trip records across 24 columns in a combined Parquet file of 18.4 GB.

## 1.3 Pipeline Architecture

The analysis followed a four-notebook pipeline, a deliberate structural choice that separates concerns and enforces reproducibility:

**Notebook 00 (Data Download)** acquires 36 monthly Parquet files from the TLC's public CDN, consolidates them into a single 18GB dataset using DuckDB, and downloads zone metadata. It produces the raw input for all subsequent work.

**Notebook 01 (Data Validation)** flags records against duration, distance, and fare thresholds, produces both a flagged dataset (all 684M records with quality flags) and a clean dataset (683.8M valid records), and generates a validation report.

**Notebook 02 (Exploratory Analysis)** aggregates the 683.8M trip records to daily zone counts, analyzes global and zone-level demand patterns, computes zone-to-global correlations, and segments zones into high-correlation (195 zones) and low-correlation (61 zones) groups.

**Notebook 03 (Demand Forecasting)** trains and compares three models on a pilot zone, selects XGBoost as the best performer, scales it across all 195 high-correlation zones, and exports forecast results and zone-level summaries.

Each notebook reads explicit file outputs from its predecessor rather than relying on shared variables or random seeds. This means any notebook can be re-run independently against saved artifacts, and the full pipeline can be audited step by step. Constants (project years, date ranges, file paths) are defined at the top of each notebook and kept consistent across the pipeline.

> ***Expansion Opportunity:*** *A pipeline orchestrator (e.g., DVC, Prefect, or a simple Makefile) could automate the end-to-end flow and ensure notebooks execute in the correct order. A shared config.yaml file could centralize all constants across notebooks. Adding automated tests (e.g., pytest checks on output schemas between notebooks) would further strengthen the pipeline's production readiness.*

# 2. Technical Decisions and Methodology Rationale

## 2.1 Why DuckDB Instead of Pandas

The combined dataset was 18.4 GB—far larger than the available RAM on a typical data science workstation. Loading this into a Pandas DataFrame would either fail with a MemoryError or require substantial workarounds (chunked reading, downsampling, or cloud compute). DuckDB was selected as the processing engine because it operates directly on Parquet files using columnar scans, meaning it only reads the columns needed for each query rather than loading the entire dataset into memory. This allowed the full 684M-record dataset to be queried, validated, and aggregated on a standard laptop.

Several DuckDB configuration choices further optimized performance. Thread count was set to 4 (matching CPU cores), and preserve_insertion_order was set to false, which allows DuckDB to process rows in whatever order is most efficient rather than maintaining the original file order. Together, these settings reduced the combination step from 30+ minutes to approximately 6 minutes. The COPY statement with Snappy compression wrote the combined file as a single Parquet output, maintaining compatibility with Pandas for downstream analysis.

> ***Expansion Opportunity:*** *Profiling DuckDB's memory usage during the combination and validation steps (e.g., with memory_limit settings or system monitoring) would quantify the memory efficiency gains versus a Pandas-based approach. This data could support a cost-benefit analysis showing that the same analysis could run on a $500 laptop rather than a cloud instance, reducing infrastructure costs.*

## 2.2 Why MAPE as the Primary Metric

Three evaluation metrics were tracked: MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), and MAPE (Mean Absolute Percentage Error). MAPE was chosen as the primary comparison metric for two specific reasons.

First, MAPE normalizes errors by actual demand, making it directly comparable across zones with vastly different volumes. A zone averaging 10,000 trips/day and a zone averaging 100 trips/day might both have acceptable forecasts, but their MAE values would differ by two orders of magnitude. MAPE places both on the same percentage scale: a 6% error is a 6% error regardless of zone size.

Second, MAPE communicates intuitively to business stakeholders. Telling an operations manager that the forecast is off by 6.4% on average is immediately actionable, whereas an MAE of 170 trips requires additional context (what's the zone's mean? is 170 a lot?). The 10% MAPE threshold was set as the performance target because it represents a meaningful improvement over simple heuristics while remaining achievable with the available features.

MAE was used as the model selection metric (choosing between Seasonal Naive, Prophet, and XGBoost) because it avoids division-by-zero issues that MAPE encounters on zero-demand days. RMSE was included for completeness, providing sensitivity to occasional large forecast misses.

## 2.3 Why a Pilot-Zone-First Approach

Rather than immediately training models across all 195 zones, the analysis first developed and compared all three models on a single pilot zone (Central Harlem, Zone 41). This was chosen because it had the highest correlation to the global demand pattern (r = 0.947), meaning it was the most representative of the dominant demand signal. If a model worked well on this zone, it was likely to generalize to other high-correlation zones.

The pilot approach served two purposes. First, it allowed rapid iteration: training one XGBoost model takes under a second, while training 195 takes about 18 seconds. When experimenting with feature sets, hyperparameters, and evaluation approaches, the pilot zone provided fast feedback loops. Second, it created a clear narrative: the pilot zone established the ranking (XGBoost > Baseline > Prophet) and the rationale, and the scaling step confirmed that the ranking held across the full zone set.

## 2.4 Why a Temporal Train/Test Split

The dataset was split temporally: January 2022 through June 2024 for training (912 days per zone) and July through December 2024 for testing (184 days). This is a critical design choice for time-series forecasting that differs fundamentally from the random train/test splits used in cross-sectional problems.

In cross-sectional problems random splitting is appropriate because observations are independent. In time-series data, observations are temporally dependent—today's demand is correlated with yesterday's and last week's. A random split would scatter future observations into the training set, allowing the model to “see” future values through lag features. This data leakage would produce unrealistically optimistic performance estimates. A temporal split ensures the model is always predicting forward in time, mirroring how the model would actually be deployed.

The 70/30 split ratio (912 vs. 184 days) was chosen to provide a full 6-month test period covering both summer and fall/winter, ensuring the model is evaluated across seasonal variation. The training period of 2.5 years provides sufficient history for the model to learn annual patterns and long-term trends.

> ***Expansion Opportunity:*** *Time-series cross-validation (rolling or expanding window) would provide more robust performance estimates than a single temporal split. For example, scikit-learn's TimeSeriesSplit could create 5 folds where the training window expands and the test window slides forward, producing 5 independent performance estimates that could be averaged and compared with confidence intervals. This is listed as a next step in the project README.*

## 2.5 Why Lag Features Were Created on the Full Dataset Before Splitting

The XGBoost feature engineering step created lag features (lag_1, lag_7, rolling_mean_7, rolling_mean_28) on the full dataset before splitting into train and test sets. This might initially appear to risk data leakage—if lag_7 for a test observation uses a value from the training period, hasn't training data leaked into the test set?

The answer is no, and understanding why is important. Lag features by definition look backward in time. A lag_7 feature for July 8 uses the actual value from July 1. In a real deployment, when predicting demand for tomorrow, yesterday's actual demand is already known. The lag features use only information that would genuinely be available at prediction time. What would constitute leakage would be using a future value as a feature—for example, using July 9's demand to predict July 8—and the lag construction explicitly prevents this through the shift() operation.

Creating lag features before splitting (rather than separately on train and test) also avoids a subtle problem: if lag features were computed only within the test set, the first 7 days of the test period would have no lag_7 values (since there would be no preceding data within the test set). By computing on the full timeline, the first test observation (July 1) correctly uses June 24's value as its lag_7 feature.

## 2.6 Why These Three Models

The three models were chosen to represent fundamentally different forecasting philosophies, each testing a specific hypothesis about the demand structure identified during EDA.

**Seasonal Naive (Baseline)** tested the hypothesis that weekly patterns alone explain most of the demand variation. By simply predicting that today's demand equals the same day last week, it provides a principled baseline: any model that cannot beat this benchmark is adding complexity without value. The 7-day lag directly encodes the dominant weekly seasonality found in EDA.

**Prophet** tested whether a decomposition-based approach—separately modeling trend, weekly seasonality, and holiday effects—would capture patterns that a simple lag misses. Prophet was designed by Facebook for business time series with "strong multiple seasonality patterns" (Taylor & Letham, 2018), making it a natural candidate. It was configured with additive seasonality (matching the constant-variance finding from EDA), weekly seasonality enabled, and daily seasonality disabled (since the data was already aggregated to daily counts).

**XGBoost** tested whether a feature-engineered gradient boosting approach could outperform both the simple lag and the decomposition model. By providing multiple lag features (lag_1, lag_7) and rolling averages (7-day, 28-day), XGBoost could learn how to weight recent demand trends against weekly patterns—something the Seasonal Naive cannot do (it always looks back exactly one week) and Prophet handles implicitly through its trend component.

> ***Expansion Opportunity:*** *Adding LightGBM (often faster than XGBoost with comparable accuracy), an ARIMA/SARIMAX model (the classical time-series approach), and a*

> *simple neural network (e.g., a feedforward MLP or LSTM) would broaden the model comparison. ARIMA would test whether autoregressive modeling with explicit seasonal differencing captures additional signal. An LSTM would test whether sequence modeling from deep learning adds value on top of manually engineered lag features.*

# 3. Phase 1: Data Acquisition

The data acquisition phase downloaded, consolidated, and verified 36 months of NYC TLC FHVHV trip records. While conceptually straightforward, this phase involved engineering decisions around scale that are worth documenting.

## 3.1 Source Data and Download Strategy

The NYC TLC publishes trip data as monthly Parquet files on a public CDN (d37ci6vzurychx.cloudfront.net). Each monthly file contains approximately 19 million trip records and ranges from 300–500 MB in size. The download script generated a list of 36 file URLs programmatically from the project years (2022, 2023, 2024) and month numbers (1–12), then downloaded each file with two important features: it skipped files that already existed on disk (enabling safe re-runs without re-downloading), and it tracked failures separately to distinguish between transient network errors and missing files.

A zone metadata file was also downloaded from the TLC, providing the mapping between numeric zone IDs and human-readable names (e.g., Zone 41 = Central Harlem) and borough assignments. The LocationID column was immediately renamed to zone_id for consistency with the trip data, a small but important normalization that prevented join mismatches downstream.

## 3.2 Data Consolidation

The 36 monthly files were consolidated into a single Parquet file using DuckDB’s COPY statement with Snappy compression. The result was an 18.4 GB file containing 684,376,551 records across 24 columns. The consolidation took approximately 6 minutes with the optimized DuckDB settings, compared to 30+ minutes without the preserve_insertion_order=false optimization. This single-file output simplified all downstream processing: rather than managing 36 separate files with glob patterns, each subsequent notebook could point to one input file.

## 3.3 Schema and Coverage Verification

After consolidation, the dataset was verified against three criteria. First, all eight critical columns needed for downstream analysis (hvfhs_license_num, pickup_datetime, dropoff_datetime, PULocationID, DOLocationID, trip_miles, trip_time, base_passenger_fare) were confirmed present via an assertion check. Second, the date range was verified to span the full project period (2022-01-01 to 2024-12-31) with no gaps. Third, a sample of records was previewed to confirm data types and reasonable values. This multi-layer verification approach—schema check, coverage check, and visual inspection—provides defense-in-depth against data quality issues that might not be caught by any single check alone.

> ***Expansion Opportunity:*** *Adding checksum verification (e.g., MD5 hashes of downloaded files compared against published values, if available) would detect corrupted downloads. Row count validation per monthly file—comparing against published TLC statistics—would catch incomplete files. These checks would be*

> *especially important if the pipeline were scheduled to run automatically (e.g., monthly updates with new data).*

# 4. Phase 2: Data Validation

The validation phase applied systematic quality checks to all 684 million records, flagging invalid entries based on duration, distance, and fare thresholds. The design philosophy was to be conservative—catching clear data errors while preserving legitimate edge cases.

## 4.1 Missing Value Assessment

A column-by-column null check revealed that all core analysis fields (pickup_datetime, PULocationID, trip_time, trip_miles, base_passenger_fare) had zero missing values. Two non-essential columns showed high null rates: originating_base_num (26.88% null) and on_scene_datetime (26.87% null). These columns were not used in the demand forecasting analysis, so they were noted but not treated. The decision to document rather than drop these columns preserved the full dataset schema for potential future analyses (e.g., dispatch timing analysis using on_scene_datetime).

## 4.2 Validation Threshold Design

Thirteen validation flags were applied across three domains: duration, distance, and fare. The thresholds were deliberately set wide to capture clear errors without removing legitimate edge cases:

**Duration:** Minimum 60 seconds (filters GPS glitches and cancelled rides), maximum 43,200 seconds (12 hours, covering trips to Philadelphia which falls within the NYC TLC service area). A separate extreme threshold of 604,800 seconds (7 days) flagged obvious data corruption.

**Distance:** Minimum 0.1 miles (filters GPS noise), maximum 200 miles (covers the NYC-to-Philadelphia corridor).

**Fare:** Minimum $0 (zero fares are allowed for promotional rides), maximum $500 (accommodates surge pricing; the 99.9th percentile was approximately $150). Negative fares were flagged as errors.

The threshold choices reflect domain knowledge about rideshare operations. A trip under 60 seconds is almost certainly a dispatch error or immediate cancellation, not a completed ride. A trip over 12 hours exceeds any reasonable driving scenario within the service area. The $0 fare floor (rather than excluding zero fares) preserves promotional and corporate-account rides that generate legitimate demand even without direct fare revenue.

## 4.3 Flag-and-Filter Implementation

The validation used a flag-and-filter pattern identical in philosophy to the UHPC concrete analysis: each check created a boolean flag column, and a master is_valid flag combined all checks. This produced two output files: a flagged dataset containing all 684M records with their quality flags (for auditability), and a clean dataset containing only valid records (for analysis).

The flagged dataset preserves the complete audit trail—anyone can query it to understand exactly why specific records were excluded.

The master validity flag required all checks to pass simultaneously: valid duration AND valid distance AND valid fare. A record that passed duration checks but had a negative fare was still excluded. This conservative approach ensures the clean dataset contains only records that are plausible across all dimensions.

## 4.4 Validation Results

Of 684,376,551 total records, 683,780,462 passed all validation checks—a 99.91% retention rate with only 596,089 records excluded (0.09%). The largest single exclusion category was negative fares (320,373 records, 0.047%), followed by trips below the 0.1-mile distance minimum (251,660 records, 0.037%) and trips below the 60-second duration minimum (60,163 records, 0.009%). No individual flag exceeded 0.1% of total records, confirming that the TLC data is generally high quality.

A critical zone-level check verified that no geographic area was disproportionately affected by validation exclusions. Of 263 zones, only one (Zone 1, Newark Airport) exceeded a 1% invalid rate—and it had only 67 total trips, making the 22.4% rate a small-sample artifact rather than a systematic data quality issue. This zone was later filtered out during EDA due to incomplete temporal coverage, not because of its invalid rate.

> ***Expansion Opportunity:*** *Statistical outlier detection (e.g., IQR-based or Z-score methods applied to trip_time, trip_miles, and fare within each zone) could identify records that pass the absolute thresholds but are unusual relative to their zone's distribution. For example, a 3-hour trip in a zone where the median trip is 15 minutes might warrant investigation even if it falls below the 12-hour maximum. Speed-based validation (distance/time) could catch trips with implausible speeds, which might indicate GPS errors.*

# 5. Phase 3: Exploratory Analysis

The exploratory analysis transformed 683 million individual trip records into zone-level daily time series and then systematically characterized the demand patterns that would inform model selection and zone segmentation.

## 5.1 Aggregation to Daily Zone Counts

The first analytical step was a DuckDB aggregation that grouped the 683.8M trip records by pickup zone and date, computing daily trip counts along with auxiliary metrics (total and average trip duration, distance, and fare). This reduced the dataset from 683 million rows to 283,528 rows—a 2,400-fold compression that made the data manageable in Pandas while preserving all the temporal structure needed for forecasting. The aggregated dataset contained 263 unique zones with dates spanning the full project period.

Calendar features were added as columns: year, month, day_of_week, day_name, is_weekend, season (meteorological), and is_holiday (using the USFederalHolidayCalendar). These features served dual purposes: they supported the EDA visualizations (e.g., average demand by day of week) and were later used as input features for the XGBoost model.

## 5.2 Temporal Coverage Validation

A critical check verified whether each zone had complete data for all 1,096 days in the project period. Of 263 zones, 256 (97.3%) had complete temporal coverage. Seven zones had missing days, ranging from Zone 253 (missing just 1 day) to Zone 264 (with data for only 1 day out of 1,096). These incomplete zones were excluded from further analysis because time-series forecasting models require continuous, regular observations. The 7 excluded zones accounted for just 2,952 records (1.04% of the aggregated data), so the exclusion had negligible impact on coverage.

This completeness check is particularly important for lag-based features. If a zone is missing data on a Tuesday, the lag_7 feature for the following Tuesday would be null, and the rolling_mean_7 would be distorted. Rather than imputing missing days (which could introduce bias), the analysis opted for clean exclusion of incomplete zones.

## 5.3 Global Demand Patterns

### Trend and Stationarity

Summing daily trip counts across all zones produced a global demand time series showing stable demand with visible weekly oscillations. An Augmented Dickey-Fuller test confirmed stationarity (ADF statistic = -4.25, p = 0.0005), meaning the series fluctuates around a stable mean without persistent upward or downward trends. This was an important finding: stationary data does not require differencing, detrending, or decomposition before modeling. The slight upward trend visible in early 2022 likely reflects post-COVID recovery, which stabilized by 2023.

### Variance Stability

A 30-day rolling standard deviation plotted alongside raw demand showed constant volatility throughout the analysis period—the spread of daily fluctuations did not increase or decrease with the demand level. This constant-variance (homoscedastic) pattern directly informed the Prophet configuration: additive seasonality was chosen because the seasonal component adds a fixed amount to the demand level rather than multiplying it by a percentage. In a multiplicative model, seasonal swings would scale with the demand level, which the data did not support.

### Weekly Seasonality

Average daily demand by day of week showed a consistent pattern: demand builds through the workweek, peaks on Saturday, and dips slightly on Sunday. Weekend demand averaged 13% higher than weekday demand. This weekly cycle was the single strongest temporal pattern in the data—stronger than monthly variation, seasonal variation, or year-over-year trends. It directly motivated the choice of lag_7 as the primary forecasting feature: if the strongest signal is weekly, then the best single predictor of today’s demand is the same day last week.

### Monthly and Seasonal Variation

Monthly demand showed a modest pattern: summer months (June–August) exhibited consistent lows across all three years, while fall and spring showed slightly higher demand. However, the magnitude of seasonal variation was small compared to the weekly cycle—the difference between the highest and lowest months was approximately 10–15%, versus the 30–40% swings seen within individual weeks. This informed the decision to prioritize lag-based features (which capture weekly patterns directly) over seasonal decomposition approaches.

> ***Expansion Opportunity:*** *Fourier analysis or periodogram decomposition could quantify the exact contribution of each frequency component (daily, weekly, monthly, annual) to total demand variance. This would put hard numbers on the qualitative observation that weekly patterns dominate. Autocorrelation and partial autocorrelation plots (ACF/PACF) would identify the optimal lag structure for ARIMA-class models and could validate the choice of lag_1 and lag_7 as the key features.*

## 5.4 Zone-Level Characteristics

### Volume Distribution

Zone-level demand varied enormously, from 13 trips per day (quiet outer-borough zones) to 11,800 trips per day (Manhattan's busiest zones). A cumulative distribution plot showed that demand was broadly distributed rather than concentrated: the top 10 zones accounted for only 13.5% of total trips, and the top 100 zones covered 72.5%. This broad distribution meant that modeling only high-volume zones would miss significant demand; a scalable approach covering the majority of zones was needed.

### Within-Zone Stability

The coefficient of variation (CV = standard deviation / mean) was computed for each zone as a measure of demand predictability. The median CV was 0.173, meaning the typical zone's daily demand fluctuates by about 17% around its mean. The most stable zone had a CV of 0.113 (very predictable), while the most volatile had a CV of 1.334 (highly variable, likely a low-volume zone where small absolute changes create large percentage swings). The low median CV provided confidence that lag-based forecasting would work well: if demand is stable within zones, then recent historical values are strong predictors of near-term demand.

### Zone-to-Global Correlation

The most consequential analysis in the EDA was measuring how closely each zone's demand pattern tracked the citywide aggregate. For each zone, the Pearson correlation between its daily demand series and the global daily total was computed. The results revealed a bimodal structure: a large cluster of zones with correlations above 0.6 (sharing the global weekly rhythm) and a smaller group with weak or negative correlations (marching to their own beat).

Setting a threshold of $r \geq 0.6$, 195 zones (76.2%) qualified as high-correlation, collectively accounting for 82% of total trip volume. The remaining 61 zones (23.8%, 18% of trips) showed low correlation to the global pattern. These low-correlation zones likely include airports (where demand follows flight schedules), entertainment districts (event-driven demand), and transit hubs (commuter-driven demand). The key insight was that a single modeling approach could serve the majority of zones and the majority of trips, while the low-correlation zones would need specialized treatment in future work.

> ***Expansion Opportunity:*** *Clustering the low-correlation zones by their demand profile (e.g., using k-means or hierarchical clustering on the daily demand time series) could identify subgroups that share patterns with each other, even if they don't match the global pattern. For example, airports might cluster together, entertainment districts might form another cluster, and each cluster could be modeled with its own approach. Additionally, the 0.6 correlation threshold was set based on domain judgment; a sensitivity analysis varying this threshold from 0.5 to 0.8 and measuring the impact on zone count, trip coverage, and downstream model performance would validate or refine the choice.*

# 6. Phase 4: Demand Forecasting

## 6.1 Feature Engineering

XGBoost requires explicit features—unlike Prophet, which internally models trend and seasonality, XGBoost treats each observation as a flat row of numbers and has no inherent notion of temporal ordering. The feature engineering step bridged this gap by creating time-aware features that encode the temporal patterns identified during EDA:

**lag_1 (yesterday's demand):** Captures short-term momentum. If demand spiked yesterday, it may remain elevated today. This feature is especially useful for capturing multi-day events (concerts, conferences, weather events) where elevated demand persists beyond a single day.

**lag_7 (same day last week):** Captures the dominant weekly pattern. This is the single most important feature, directly encoding the finding that weekly seasonality is the strongest demand signal. A lag of exactly 7 ensures day-of-week alignment: Monday predicts Monday, Saturday predicts Saturday.

**rolling_mean_7 (7-day trailing average):** Smooths recent demand into a level indicator. Where lag_7 looks back to a single point, rolling_mean_7 averages the entire preceding week, providing a more stable estimate of the current demand regime. This helps the model adjust when demand has gradually shifted (e.g., a new subway construction zone redirecting rideshare demand).

**rolling_mean_28 (28-day trailing average):** Provides a monthly baseline. This feature captures slower-moving trends such as seasonal shifts or gradual growth. By comparing lag_7 against rolling_mean_28, the model can detect whether the most recent week was above or below the longer-term norm.

**Calendar features (month, day_of_week, is_holiday):** Encode structural time patterns. While lag_7 implicitly captures day-of-week effects, the explicit day_of_week feature allows the model to learn systematic differences (e.g., Saturdays are consistently the peak day). The month feature captures seasonal variation, and is_holiday flags days where demand may deviate sharply from typical patterns.

All rolling features used a shift(1) before the rolling window, ensuring they only used information available at prediction time (up through yesterday, not including today). Rows with null lag values (the first 7–28 days of the series, depending on the feature) were dropped rather than imputed, losing a small number of training observations but preserving data integrity.

> ***Expansion Opportunity:*** *Additional feature candidates include: weather data (temperature, precipitation—both strongly correlated with rideshare demand), event indicators (major events at Madison Square Garden, Yankee Stadium, etc.), subway service disruptions (which redirect commuters to rideshare), and interaction features (e.g., is_weekend × is_holiday, lag_7 × month). Feature importance analysis could then*

> *quantify which additions provide the most lift. Recursive feature elimination or SHAP-based selection could prune the feature set to the most informative subset.*

## 6.2 Pilot Zone Model Comparison

### Seasonal Naive Baseline

The Seasonal Naive model predicted each day's demand as the actual demand from the same day of the previous week (lag_7). On the pilot zone (Central Harlem, Zone 41, mean demand 3,992 trips/day), it achieved an MAE of 297 trips/day and a MAPE of 7.1%. This strong baseline performance confirmed the EDA finding: weekly patterns alone explain most of the day-to-day variation. The Seasonal Naive's limitation is that it cannot adjust for trends or recent shifts—if demand has been gradually increasing over the past month, the Naive forecast still looks back to the same day last week without adjustment.

### Prophet

Prophet was configured with additive seasonality, weekly seasonality enabled, yearly seasonality disabled (to focus on the dominant weekly signal), and a changepoint_prior_scale of 0.05 (a moderate setting for trend flexibility). On the pilot zone, Prophet achieved an MAE of 528 trips/day and a MAPE of 13.2%—substantially worse than the Seasonal Naive baseline.

This underperformance deserves explanation because Prophet is widely regarded as a strong time-series tool. The key issue is forecast horizon. Prophet was designed for medium-to-long-range forecasting where trend extrapolation and seasonal decomposition add value. For one-day-ahead forecasting, these advantages become liabilities: Prophet's forecast for tomorrow is driven primarily by its estimated trend and seasonal components, which are fit to the entire training period. It does not directly use yesterday's actual value. In contrast, the Seasonal Naive and XGBoost models directly leverage the most recent observations (lag_1 and lag_7), giving them a substantial informational advantage for short-horizon prediction.

Tuning experiments confirmed this diagnosis. Adding yearly seasonality and US holidays, and testing multiplicative seasonality, did not improve Prophet's performance. The additive configuration produced an MAE of 525, and the multiplicative configuration produced 527—both still worse than the 297 MAE baseline. This suggests that Prophet's underperformance was structural (not using recent actuals) rather than configurational.

> ***Expansion Opportunity:*** *Prophet could be re-evaluated at longer forecast horizons (7-day, 14-day, 30-day). At these horizons, lag features become less reliable (lag_7 becomes a 14-day-old observation, etc.) and Prophet's trend/seasonality decomposition may outperform. This would test the hypothesis that Prophet and XGBoost excel at different forecast horizons, potentially supporting a hybrid approach where XGBoost handles 1–3 day horizons and Prophet handles longer-range planning.*

### XGBoost

XGBoost was trained with 100 estimators, max_depth=6, learning_rate=0.1, and random_state=42. On the pilot zone, it achieved an MAE of 219 trips/day and a MAPE of

5.1%—a 26.2% improvement in MAE over the Seasonal Naive baseline. Feature importance analysis showed day_of_week and lag_7 as the two most important features, directly confirming the EDA finding that weekly patterns dominate. The lag_1 and rolling mean features provided incremental value by allowing the model to adjust for recent demand shifts.

**Table 1:** *Pilot Zone Model Comparison — Central Harlem (Zone 41)*

| Model | MAE | RMSE | MAPE | vs. Baseline |
|---|---|---|---|---|
| Seasonal Naive | 297 | 393 | 7.1% | — |
| Prophet | 528 | 627 | 13.2% | -77.7% |
| **XGBoost** | **219** | **312** | **5.1%** | **+26.2%** |

## 6.3 Scaling to All Zones

Based on the pilot zone results, XGBoost was selected for scaling. A reusable forecast_zone_xgboost function was created that encapsulates the full train-predict-evaluate cycle for a single zone: it filters the data, fits an XGBoost model with the same hyperparameters as the pilot, generates predictions on the test period, and returns a DataFrame with dates, actuals, and forecasts. This function was then applied to all 195 high-correlation zones in a loop, with progress tracking via tqdm. The full scaling completed in approximately 18 seconds (about 10 zones per second).

An important architectural decision was to train a separate model per zone rather than a single pooled model across all zones. The per-zone approach was chosen because different zones may have different relationships between features and demand. For example, in a residential zone, lag_1 might be very predictive because demand is habit-driven, while in a commercial zone, day_of_week might dominate because demand follows business hours. Training separate models allows each zone to learn its own feature-demand relationships.

> ***Expansion Opportunity:*** *A pooled model (training a single XGBoost on all 195 zones simultaneously, with zone_id as a feature or with zone-level embeddings) could be compared against the per-zone approach. Pooled models can transfer learning across zones—if a new zone has limited history, the pooled model leverages patterns from all other zones. A hybrid approach (pooled model with zone-specific fine-tuning) could combine both advantages. Additionally, the current approach uses the same hyperparameters for all zones; per-zone or group-level hyperparameter tuning could further improve performance.*

## 6.4 Scaled Results

### Overall Performance

Across all 195 zones, XGBoost achieved an average MAPE of 6.4%, an average MAE of 170 trips/day (6.3% of mean demand of 2,713 trips/day), and 97% of zones met the sub-10% MAPE target. The performance was remarkably consistent: the MAPE histogram showed most zones clustered near the median (5.8%), with a small tail extending to 10–15% for a handful of structurally more variable zones.

### Residual Analysis

Residuals (actual minus forecast) were analyzed by month and day of week to check for unaddressed seasonal or weekly patterns. Monthly residuals ranged from -27 (August, slight overprediction) to +83 (December, underprediction), representing just 0.9% of mean demand. Day-of-week residuals ranged from -57 (Sunday, overprediction) to +47 (Tuesday, underprediction), at 1.0% of mean demand. December's slight underprediction likely reflects holiday-season demand increases not fully captured by the model. Sunday's overprediction suggests the model slightly overestimates weekend demand—possibly because the lag_7 feature preserves the previous Sunday's level even when a gradual decline is occurring.

Critically, no month or day showed bias large enough to warrant model modification. The residuals were effectively random noise around zero, confirming that the model has captured the main demand signals and is not systematically missing any temporal pattern.

> ***Expansion Opportunity:*** *A formal Ljung-Box test on the residuals would statistically test for remaining autocorrelation. If significant autocorrelation is detected (e.g., at lag 7), it would indicate the model has not fully captured the weekly pattern and might benefit from additional features. Plotting residual ACF/PACF would visualize any remaining temporal structure. Segmenting residual analysis by borough or zone volume tier could reveal whether certain zone types have systematically higher or lower forecast accuracy.*

# 7. Practical Implications

## 7.1 Operational Value

A 6.4% MAPE forecast translates to concrete operational benefits. For a zone averaging 3,000 trips per day, the model's typical forecast miss is about 192 trips—roughly the workload of 4–5 drivers over a typical shift. Overstaffing by this amount incurs modest idle-time cost; understaffing by this amount creates passenger wait-time delays but not service collapse. Compared to a 13% MAPE forecast (Prophet) or even a 7% MAPE forecast (Seasonal Naive), the XGBoost model's tighter accuracy enables meaningfully better resource allocation.

From a workforce perspective, accurate demand forecasts provide drivers with better information about when and where demand will be high, reducing the guesswork and uncertainty that characterize gig work. A driver who knows that Zone 41 typically sees 4,200 trips on Saturdays but only 3,500 on Mondays can plan their schedule accordingly. The zone-level granularity of this model—195 separate forecasts, not just a citywide number—adds geographic specificity that a single aggregate forecast cannot provide.

## 7.2 What the Model Does Not Capture

The model's forecast is based entirely on historical demand patterns and calendar features. It has no knowledge of: weather (a heavy snowstorm can cut demand in half), major events (a concert at Madison Square Garden can double demand in nearby zones), transit disruptions (a subway outage redirects commuters to rideshare), policy changes (congestion pricing adjustments alter rideshare economics), or competitor behavior (Uber vs. Lyft pricing changes). Any of these external factors can cause demand deviations that the model will miss by definition.

This limitation is inherent to the current feature set, not to the modeling approach. XGBoost could readily incorporate weather forecasts, event calendars, and transit status as additional input features. The incremental improvement from each external data source could be measured by comparing model performance with and without the feature, providing a clear return-on-investment calculation for data acquisition efforts.

## 7.3 Coverage and Scope

The model covers 195 of 263 zones (74%) and 82% of total trip volume. The 61 excluded low-correlation zones represent structurally different demand patterns that cannot be well-served by the shared modeling approach. These zones are not unimportant—they include high-profile locations like airports and entertainment districts—but they require specialized models that account for their unique demand drivers (flight schedules, event calendars, commuter flows). Developing these specialized models is a natural extension of the current work.

# 8. Conclusions, Limitations, and Future Work

## 8.1 Summary of Key Findings

This project demonstrated that gradient-boosted trees with engineered lag features can forecast daily rideshare demand with strong accuracy across a large, diverse set of urban zones. The final XGBoost model achieved a 6.4% average MAPE across 195 NYC taxi zones representing 82% of total trip volume, with 97% of zones meeting the sub-10% MAPE performance target.

The analysis confirmed that weekly seasonality is the dominant demand signal in NYC rideshare data, with lag_7 and day_of_week consistently ranking as the most important features. The EDA-driven zone segmentation strategy—identifying high-correlation zones that share consistent demand patterns—proved effective, enabling a scalable modeling approach without sacrificing per-zone accuracy. The pilot-first development strategy demonstrated that models validated on a single representative zone generalize well to the broader zone set.

Prophet's underperformance for one-day-ahead forecasting was an instructive finding, highlighting that model selection must match the forecast horizon. Prophet's design strengths (trend decomposition, holiday handling, long-range seasonal patterns) are better suited for medium-to-long-range planning.

## 8.2 Limitations

**Forecast horizon.** The current model is limited to one-day-ahead predictions. While useful for daily operational planning, most staffing and fleet decisions require 7–14 day forecasts. Extending the horizon will likely degrade accuracy as lag features become less informative over longer gaps.

**No external variables.** Weather, events, transit disruptions, and policy changes are not incorporated. These are likely the primary sources of remaining forecast error.

**Low-correlation zone exclusion.** 61 zones (18% of trip volume) including airports and entertainment districts are not covered by the current model.

**No cross-validation.** Results depend on a single temporal split. Time-series cross-validation with rolling windows would provide more robust performance estimates.

**Default hyperparameters.** XGBoost was used with standard hyperparameters (100 estimators, max_depth=6, lr=0.1) without systematic tuning. GridSearchCV or Bayesian optimization could potentially improve performance.

**Single random seed.** All XGBoost models used random_state=42. A multi-seed robustness analysis would quantify sensitivity to initialization.

## 8.3 Future Work

**Multi-day forecast horizons** (7–14 days) to support workforce scheduling and fleet planning.

**Time-series cross-validation** with expanding or rolling windows for more robust performance estimates.

**Hyperparameter optimization** via GridSearchCV, RandomizedSearchCV, or Optuna.

**External feature integration:** weather forecasts, event calendars, transit alerts, and congestion pricing data.

**Specialized models for low-correlation zones** using cluster-specific approaches.

**Prophet re-evaluation at longer horizons** where its trend/seasonality decomposition may outperform lag-based approaches.

**SHAP analysis** to interpret feature contributions at the zone level and identify zones where specific features are more or less important.

**Interactive dashboard** (Streamlit or Tableau) for zone-level forecast exploration and monitoring.

**Real-time updating pipeline** that retrains models as new monthly TLC data becomes available.

## References

Hyndman, R. J., & Athanasopoulos, G. (2021). Forecasting: Principles and Practice (3rd ed.). OTexts.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2022). M5 accuracy competition: Results, findings, and conclusions. International Journal of Forecasting, 38(4), 1346–1364.

NYC Taxi and Limousine Commission. (2024). TLC Trip Record Data. https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Taylor, S. J., & Letham, B. (2018). Forecasting at Scale. The American Statistician, 72(1), 37–45.