CPSC 5310 – Machine LearningSeattle University – Winter 2026 Quarter

PROJECT PROGRESS REPORT

Prepared by:

Ben Tran, Rizvan Ahmed Rafsan

DeepDispatch

OPTIMIZING TAXI FLEET ALLOCATION VIA INTELLIGENT DEMAND PREDICTION

1. Team Members:

1. Ben Tran
2. Rizvan Ahmed Rafsan

2. Selected Dataset

NYC Yellow Taxi Trip Data (On Kaggle)

3. Problem Statement (Updated)

a. Motivation

The primary motivation for this project is to solve the persistent "spatial mismatch" in urban transportation where supply fails to meet passenger demand efficiently. While the NYC Yellow Taxi dataset is historical, the fundamental challenge of fleet relocation remains a cornerstone of modern logistics. By developing an intelligent dispatch system, we replace reactive intuition with proactive, data-driven repositioning. This work is highly relevant to the evolution of autonomous ride-sharing services like Waymo, where historical demand patterns serve as a blueprint for optimizing fleet utilization, reducing operational "dead mileage," and ensuring more sustainable, efficient urban transportation ecosystems.

b. The Problem and the Goal

We are trying to address the inefficiency of taxi fleet allocation caused by the "spatial mismatch" between supply (drivers) and demand (passengers). Drivers often circle empty streets, wasting fuel because they rely on intuition rather than data to find passengers. Specifically, drivers and fleet operators struggle with **revenue loss** (missed opportunities during demand surges), **increased operational costs** (wasted fuel and possible wear-and-tear from "dead mileage" without passengers). To resolve this, we propose DeepDispatch: an end-to-end intelligent dispatch system.

> *The goal: Replace reactive driver intuition with a spatiotemporal model of future potential revenue heatmap to optimize fleet repositioning strategy 1 hour in advance.*

As illustrated in the system architecture in **Figure 1**, the model uses historical trip data to generate a heatmap that shows the possible revenue potential of a taxi zone in advance. This output is fed into a dispatch logic algorithm, which sends proactive repositioning alerts to a driver's mobile app, allowing them to move to high-demand zones before the surge occurs.

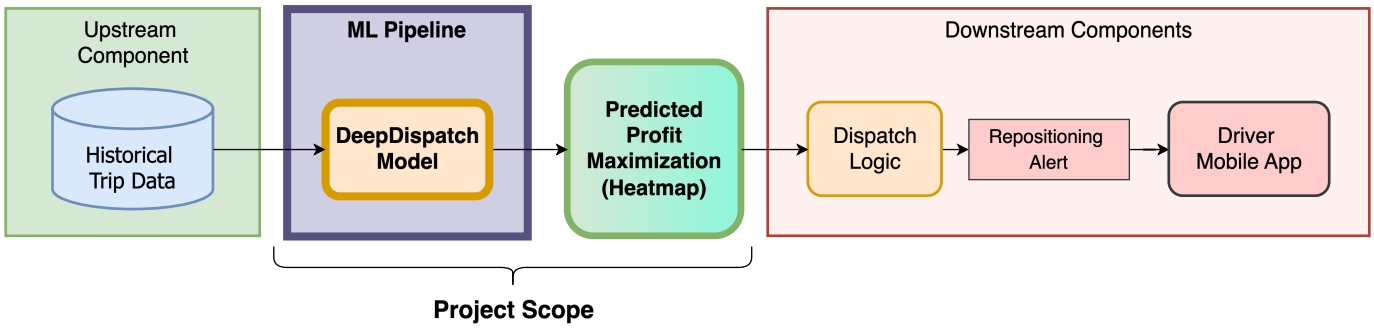


Figure 1: System architecture of DeepDispatch: an end-to-end intelligent dispatch system.

A key aspect of DeepDispatch is the transformation of continuous financial trip data into actionable revenue classes. Rather than predicting raw, noisy currency values or demands, our system categorizes future demand or potential revenue into discrete revenue potential rankings (on a scale of 1 to 5). By grouping potential earnings into these ranked "buckets" (ranging from "Quiet Level" to "Surge Level"), the model provides a stabilized and highly interpretable signal for drivers.

These predicted classes are then projected onto a geographic grid to generate a Future Potential Revenue Heatmap. As illustrated in the system architecture in **Figure 1**, this heatmap provides a high-resolution visual guide of the city's "profitability pulse." This output is fed into a dispatch logic algorithm, which sends proactive repositioning alerts to a driver's mobile app, allowing them to move to a "Class 5" surge zone before the surge occurs, effectively eliminating the inefficiency of dead mileage.

c. Performance Measurement

Model performance will be assessed through several key metrics beyond simple accuracy. We employ macro F1-score to treat all revenue classes equally and **weighted F1-score** to reflect real-world distributions. The confusion matrix reveals critical misclassification patterns: whether the model confuses *adjacent classes* (Class 3 vs. Class 4) or *distant ones* (Class 1 vs. Class 5).

Recognizing that revenue classes form an ordinal scale, we introduce a "**1-class tolerance accuracy**" metric where predictions within one class of truth are deemed acceptable[1]. For instance, predicting "High Value" instead of "Surge" still provides actionable guidance to drivers, whereas confusing "Quiet" areas with "Surge" would be operationally harmful. This tolerance better captures real-world utility since approximate revenue predictions remain valuable for dispatch decisions. We also calculate Mean Absolute Error (MAE) on class labels, treating them as ordinal values to quantify average prediction distance, a nuance that binary accuracy metrics overlook.

4. Data Preprocessing and Cleaning

d. Feature Derivation

Some features were derived from the dataset to help with data cleaning and preprocessing purposes:

- `trip_duration_min`: Calculated from `tpep_pickup_datetime` and `tpep_dropoff_datetime`.
- `speed_mph`: Calculated from `trip_distance` and `trip_duration_min`.

[1] Error-Sensitive Evaluation for Ordinal Target Variables (Chen et al., Eval4NLP 2021)

e. Handling Invalid and Impossible Values (Outlier Removal)

- Negative/Zero Values: Rows with `trip_duration_min` <= 0, `trip_distance` <= 0, `fare_amount` <= 0, or `passenger_count` <= 0 were removed. These most likely represent impossible data or a data input related issue.
- Upper Limits: Outliers beyond realistic NYC taxi operations were removed:
    - `trip_duration_min` > 240 minutes (4 hours, too long to be a typical NYC Taxi Ride)
    - `trip_distance` > 100 miles (too long to be a typical NYC Taxi Ride)
    - `speed_mph` > 100 MPH (Driving 100+ mph often triggers automatic reckless driving charges, which are criminal offenses, not just infractions)
    - `total_amount` > 1000 (in USD, to filter extreme fare glitches)

f. Spatial Filtering (NYC Bounding Box)

Trips with `pickup_latitude/longitude` or `dropoff_latitude/longitude` falling outside a defined NYC metropolitan bounding box (40.5 to 41 latitude, -74.3 to -73.7 longitude) were removed. This ensures the model focuses on relevant geographic areas.

g. Temporal Filtering

- All January 2015 data was archived, ensuring the model trains only on 2016 data to maintain consistency in pricing and operational patterns. Note that 2015 data do not contain any data after January.
- A check for missing dates in the timeline confirmed a continuous data stream, crucial for time-series analysis.

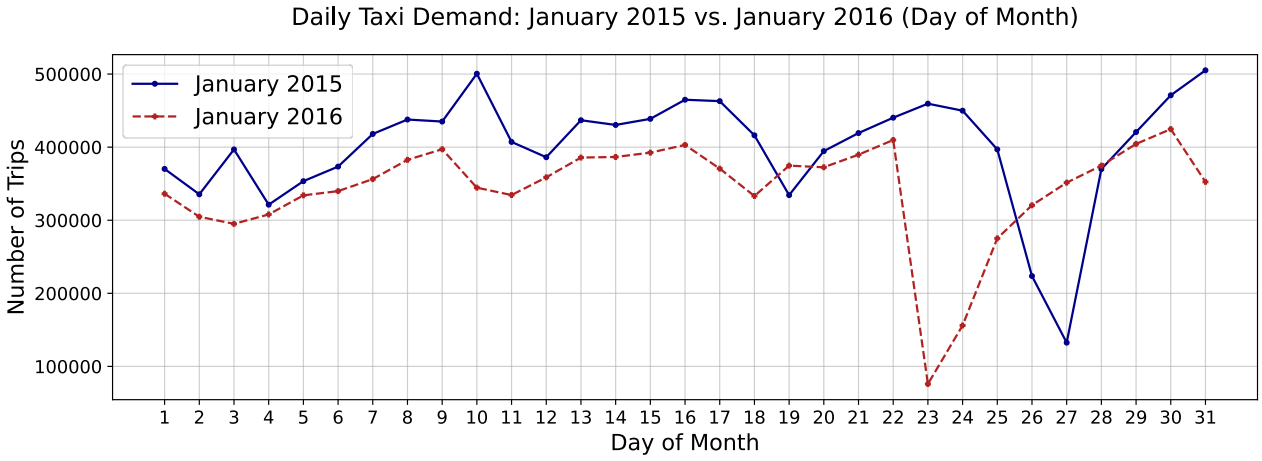


Figure 2: Number of trips comparison for January data in the dataset.

5. Summary of EDA Findings

Exploratory Data Analysis was conducted at multiple stages to inform modeling decisions and feature engineering.

a. Temporal Patterns

- **Hourly Seasonality:** Demand shows clear daily peaks (e.g., evening rush hour around 7 PM) and troughs (e.g., early morning around 4 AM).

- **Weekly Seasonality:** Significant surges in demand and revenue were observed on weekends, particularly Friday and Saturday evenings.

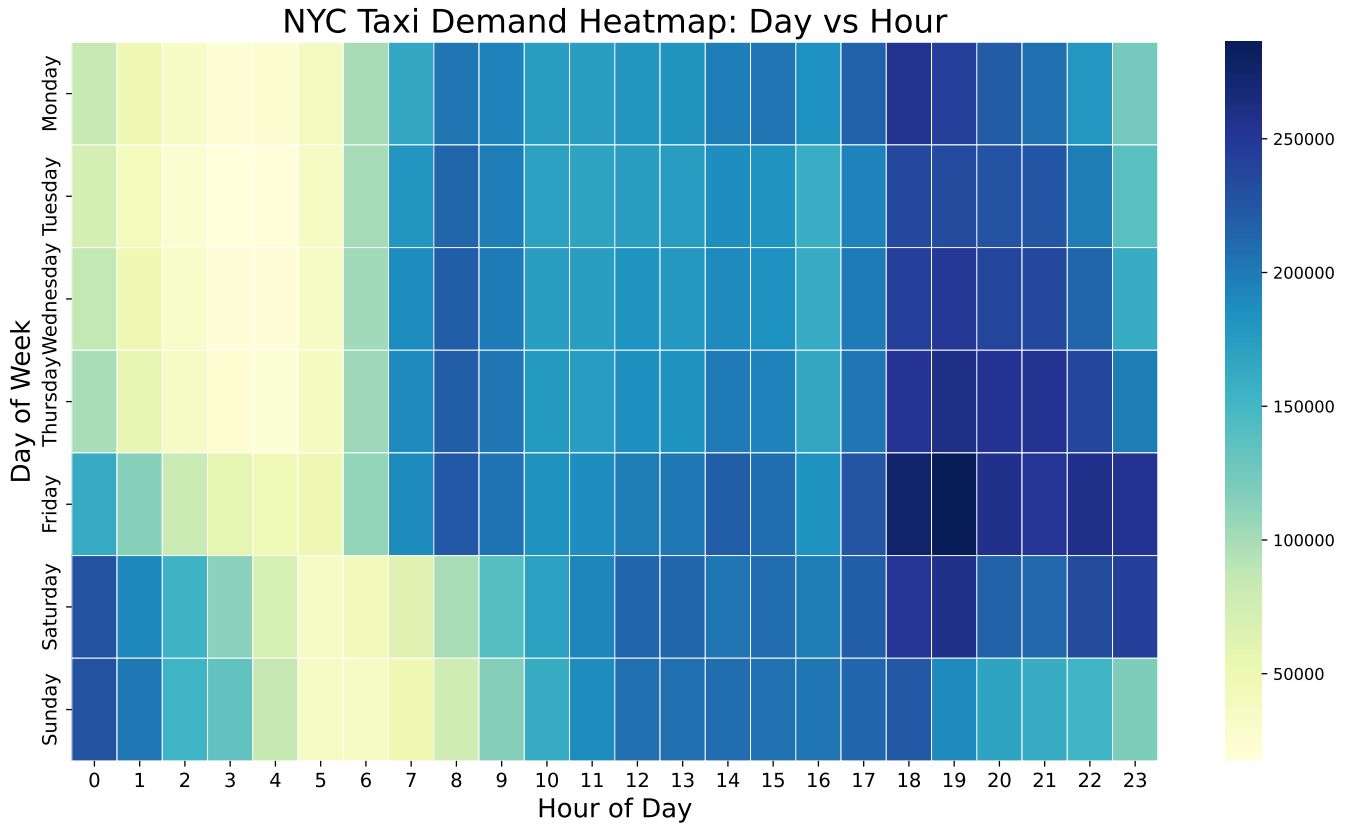


Figure 3: Hourly taxi demand heatmap for different days of the week revealing significant temporal patterns.

These patterns underscore the necessity of lag features and cyclical time encoding.

b. Feature Distribution

The vast majority of records are clustered around short, low-fare "neighborhood hops," while a small "long tail" represents highly profitable airport runs and long-distance journeys.

If left in their raw state, these extreme outliers would disproportionately skew a regression model's loss function, leading to poor generalization. To better visualize the underlying structure of this data, we applied a logarithmic transformation, as shown in **Figure 4**. This transformation revealed that when viewed on a log scale, the features follow a near-normal distribution, confirming that the "surge" events are not random noise but distinct, predictable statistical occurrences.

This discovery directly informed our decision to shift from a regression-based approach to a quantile-based classification strategy (1-5 classes). By using quantiles (equal-frequency binning) rather than equal-width bins, we can mitigate potential class imbalance and outlier issues. In addition, from a user-experience perspective, a 1-5 "Heatmap" provides a significantly clearer and more actionable signal for taxi drivers than a raw, fluctuating currency prediction.

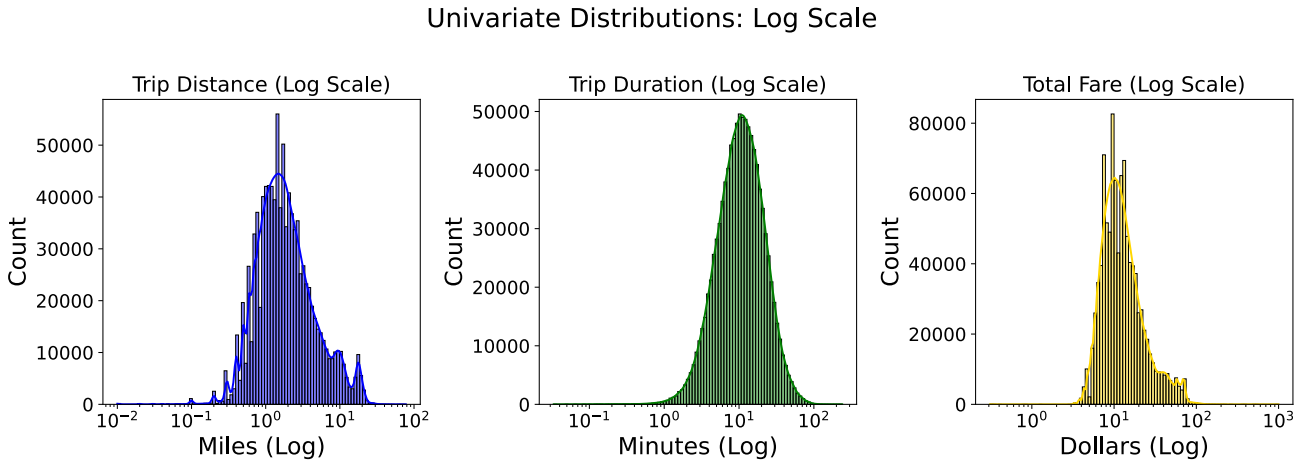


Figure 4: Univariate Distributions of target related features.

We also used a trip distance vs duration plot colored by speed to get a better insight on revenue potential. The plot shown in **Figure 5**, by coloring points based on speed, allowed us to visually identify regions where trips tend to be faster (e.g., specific combinations of distance and duration that appear in high-speed colors). These faster trips generally correspond to better revenue per minute. This also shows the potential room for improvements over the traditional reactive driver intuition-based approaches.

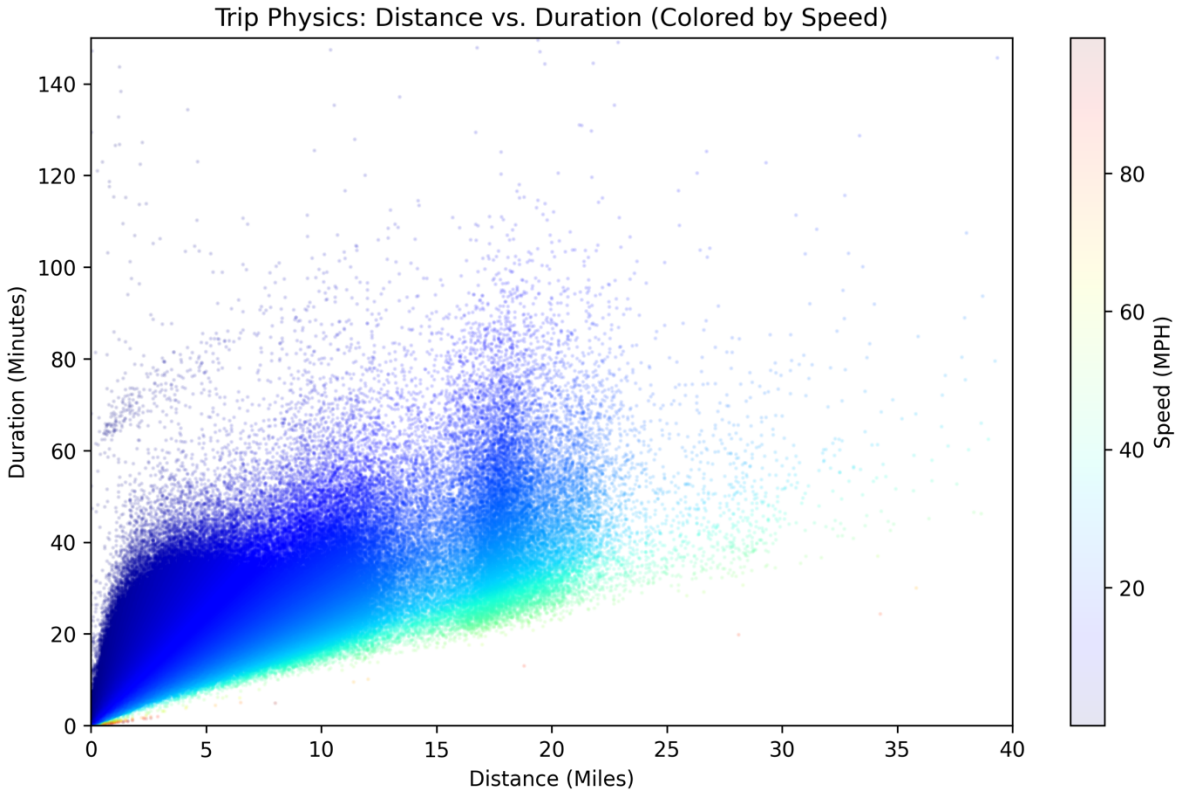


Figure 5: Low speeds (e.g., dark blue/green colors) trips indicate congestion. Trips in these zones, even if they cover a reasonable distance, take a long time, significantly reducing the potential revenue per minute for the driver.

c. Fare Rate

NYC has some fixed taxi fare rates enforced based on different types of rides, one of which considers if it's an "airport" ride. The fare structure analysis based on the `RateCodeID` feature shown in **Figure 6,** shows this pattern. Notice the clearly visible horizontal lines indicating fixed rate.

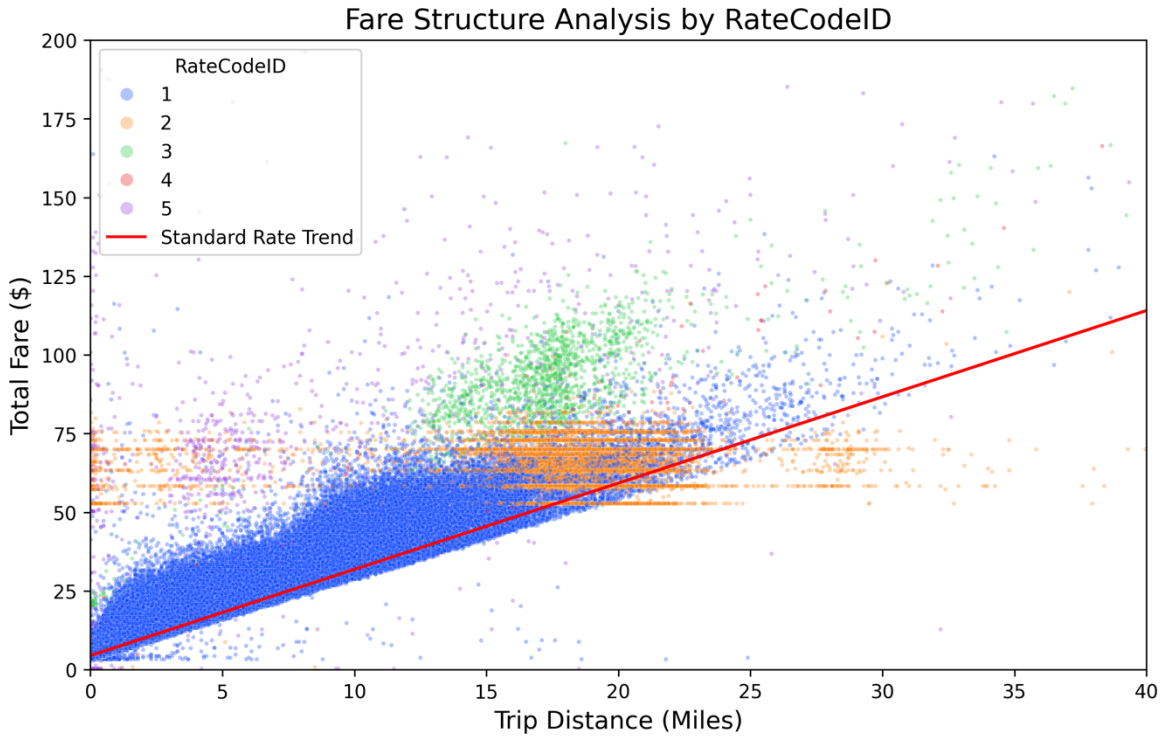


Figure 6: Scatterplot of Trip Distance and Total Trip Fare colored by RateCodeID revealed a fixed fare rate for airport trips.

This pattern has informed us to separately indicate the airport rides using a new feature.

d. Pickup Zones

The pickup hotspots show several different clusters or zones. While the Manhattan area has a lot more pickups and a small number of clusters might work for that, the outer areas are dispersed. While a density-based clustering algorithm might be useful for this, we decided to use K-means with a high number of clusters that reflects the average relocation distance for a driver.

A distance of approximately 0.5 miles between cluster centroids provides the ideal balance between spatial granularity and operational feasibility for the DeepDispatch system. At this scale, each cluster represents a distinct neighborhood "cell", roughly a few minutes' drive in NYC traffic. This resolution is granular enough to differentiate between a high-profit commercial corridor and a nearby quiet residential street, yet large enough to ensure each zone contains sufficient historical data for stable forecasting. For drivers, a 0.5-mile threshold ensures that "repositioning alerts" trigger meaningful shifts in location rather than redundant movements to the next block, maximizing fleet utility while minimizing fuel waste. By further integrating revenue-per-minute (RPM) into this spatial framework, these cells transcend simple geography to become "economic zones" that highlight the true value of a neighborhood. This approach allows the system to partition the city into non-overlapping, actionable pixels that serve as the fundamental units for our deep learning demand-prediction model.

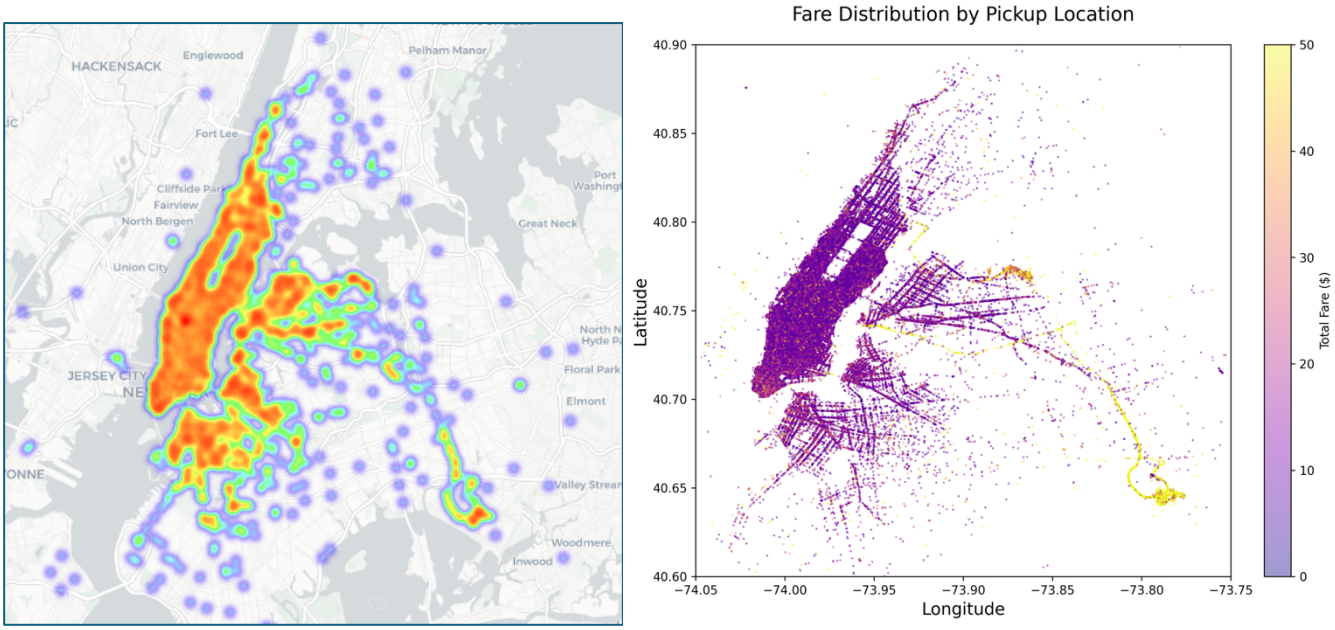


Figure 7: Pickup location heatmap by pickup count and Fare Distribution showing potential clustering necessity.

e. "The Jonas Blizzard" Effect

A time-series plot of the number of daily trips across the dataset has revealed a massive drop in daily taxi trips on January 23rd, with a decrease of 81.47% compared to the previous day.

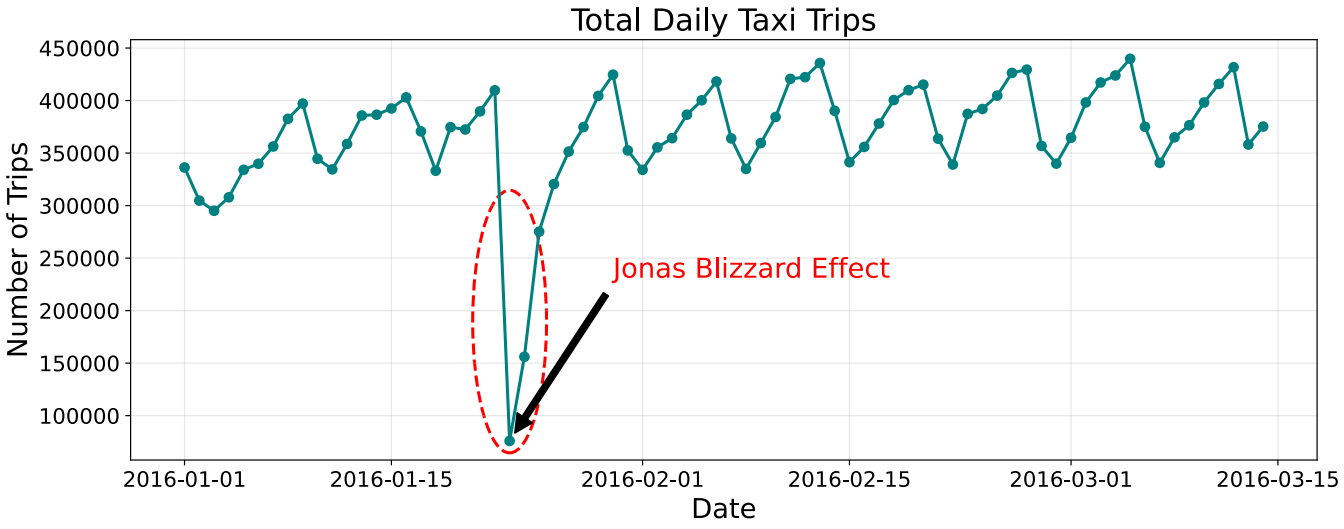


Figure 8: Time series plot of the total number of taxi trips per day revealed a massive drop in daily trip count.

A comprehensive study revealed that this significant reduction is attributed to Winter Storm Jonas[2] (Blizzard of 2016), which brought heavy snowfall and hazardous conditions to the East Coast of the United States around that date, severely impacting travel and daily activities in New York City. The blizzard led to widespread cancellations of flights, public transportation shutdowns, and a general advisory for people to stay indoors,

[2] January 2016 United States blizzard on Wikipedia.

directly causing the sharp decline in taxi ridership. To capture this effect, we also used a tag or feature that indicated this drop for the duration January 23-25.

f. Feature Correlation

The correlation heatmap reveals significant multicollinearity among financial and distance-based features, with `fare_amount`, `total_amount`, and `trip_distance` showing near-perfect correlations (0.94–0.98). This redundancy confirms that keeping all three would introduce noise and potential overfitting. To optimize the "DeepDispatch" pipeline, these should be consolidated into a single high-fidelity "Economic Signal," such as your proposed `profit_index` or Revenue Per Minute (RPM). Conversely, `passenger_count` and extra show negligible correlation with revenue potential, marking them as low-value features. Removing these redundant and low-impact variables will streamline model training and improve forecasting stability.

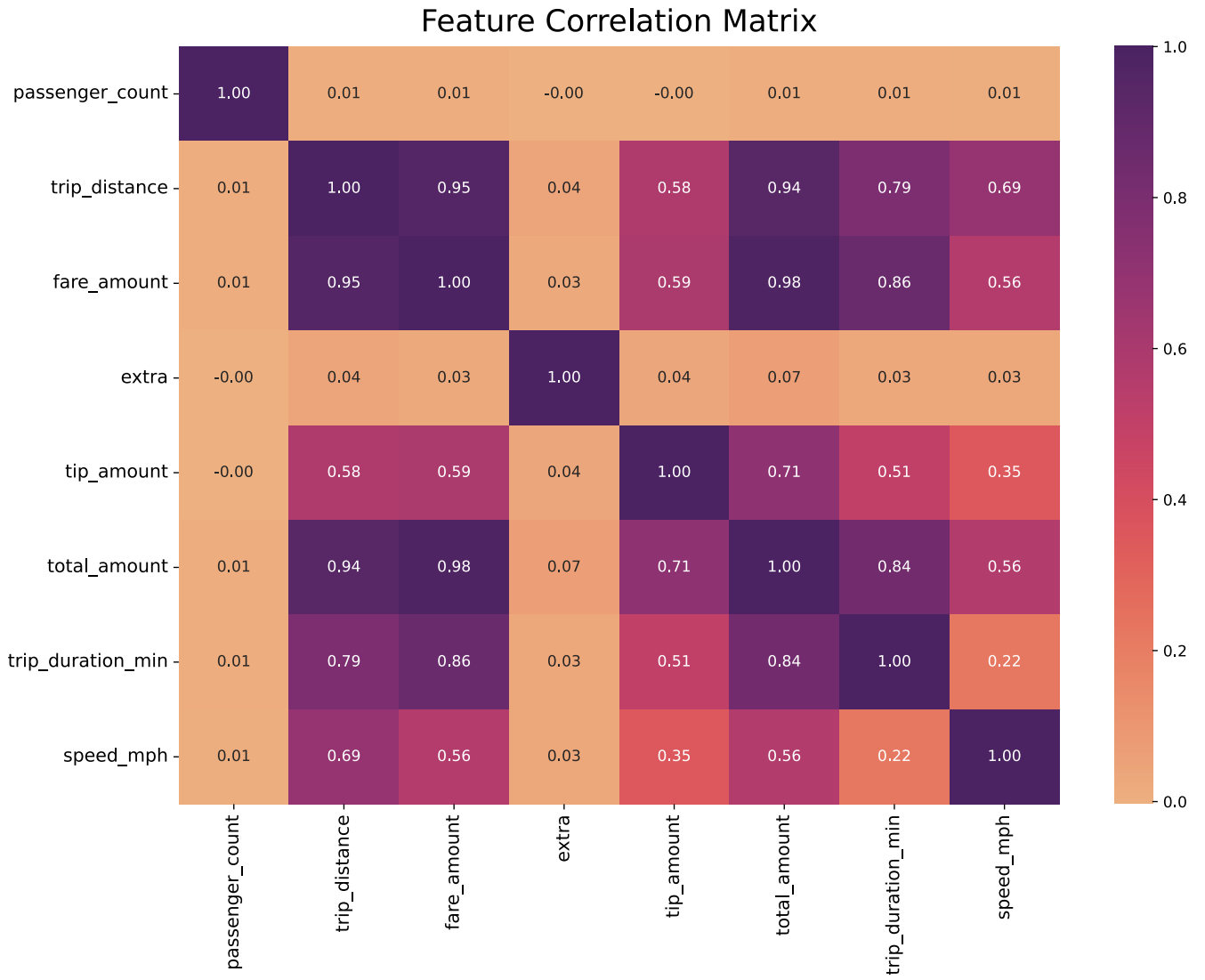


Figure 9: Correlation heatmap of the non-aggregated features.

6. The Machine Learning Pipeline

a. Clustering

`MiniBatchKMeans` (with final `k=150`) was fit on scaled `pickup_latitude` and `pickup_longitude` from the clean training data. The `StandardScaler` and `KMeans` model were saved to ensure consistency when processing new (test) data. The clustering metrics shown in **Appendix a**, revealed that while rate of inertia decrease is high with low number of clusters, after k=150, it the rate of decrease is a lot less. The silhouette score goes up after k=125. But the Davies-Bouldin Index is better at k=150 than k=125.

We also considered the practical operational granularity for choosing the number of clusters. Initially chosen, smaller number of clusters like k=40 was too coarse, with an average distance between zones of over 1 mile. This would too far for a driver to "hop" between trips, reducing the revenue potential. At k=150, the average distance to the nearest zone drops to 0.48 miles. This represents a few city blocks, an ideal distance for a "Proactive Dispatch" system that allows drivers to reposition quickly and efficiently. Also, the outer areas are divided in lower number of clusters indicating a more uniform cluster sizes compared to choosing a lower k value. **Figure 9** shows the cluster centroids and cluster zones for k=150.

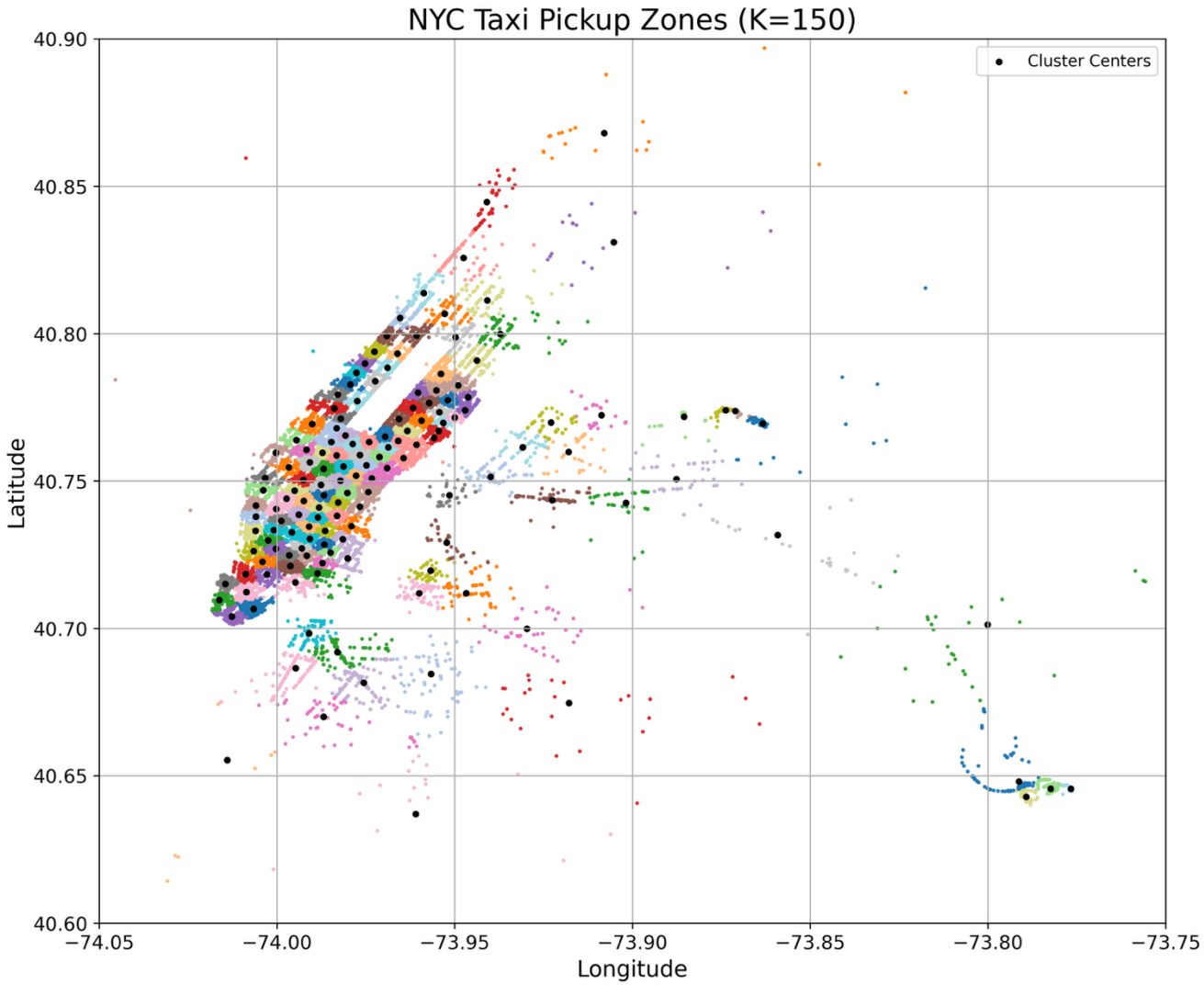


Figure 10: Demand zones and cluster centroids revealed at k=150. The demand zones are more practical and appropriate for dispatch or relocation decisions.

b. Aggregation/Time Binning

Raw trip data (with cluster IDs assigned) was aggregated into an hourly time-series grid for each of the 150 clusters. A "Master Grid" strategy was implemented to ensure every cluster has a continuous row for every hour (even zero-demand hours, which are critical signals).

Key aggregated features include:

`demand_count, total_revenue, trip_distance, total_engaged_minutes, speed_mph, airport_trip_count, passenger_count.`

c. Target Generation: Defining "Revenue Potential"

A core challenge of DeepDispatch was defining what "Revenue Potential" truly means for a taxi driver. Simply using total fare amount per hour could be misleading, as a high-revenue hour might also involve severe traffic or long waiting times, making it inefficient. To address this, we engineered a specific `profit_index` and subsequently a 5-class target variable.

We aimed to create a single metric that encapsulates both the total money earned and the efficiency of earning it. The `profit_index` serves this purpose by combining `total_revenue` with `rpm` (revenue per minute).

- The `rpm` (revenue per minute) metric quantifies how much money a driver earns per minute of active engagement in a zone. During our aggregation step, we took the sum of the individual trip duration values and grouped them by `pickup_cluster` and hourly frequency, which was the `total_engaged_minutes`. The `rpm` is the then calculated as:

$$\text{rpm} = \frac{\text{total\_revenue}}{\text{total\_engaged\_minutes} + \epsilon}$$

  Where ϵ is a small number to prevent division by zero when `total_engaged_minutes` is 0.
- The `profit_index` combines the total value of a zone with its earning efficiency.

$$\text{profit\_index} = \text{total\_revenue} \times \text{rpm}$$

- Our model needs to predict the future state (1 hour advance in time). We define the raw target by shifting the profit_index forward in time.

$$\text{target\_profit\_next\_hour}_t = \text{profit\_index}_{t+1}$$

- To provide drivers with actionable, easily interpretable insights, the continuous `target_profit_next_hour` was transformed into a 5-class categorical variable from four thresholds. These thresholds divide the active (non-zero) profit hours into five groups:
  - Revenue Class 1 (*Quiet*): Assigned to any value that is <= 10th percentile.
  - Revenue Class 2 (*Steady*): Assigned to active profits <= 32.5th percentile.
  - Revenue Class 3 (*Busy*): Assigned to active profits <= 55th percentile.
  - Revenue Class 4 (*High Value*): Assigned to active profits <= 77.5th percentile.
  - Revenue Class 5 (*Surge*): Assigned to active profits > 77.5th percentile.

  The distribution of these 5 claseses are shown in **Figure 11.**

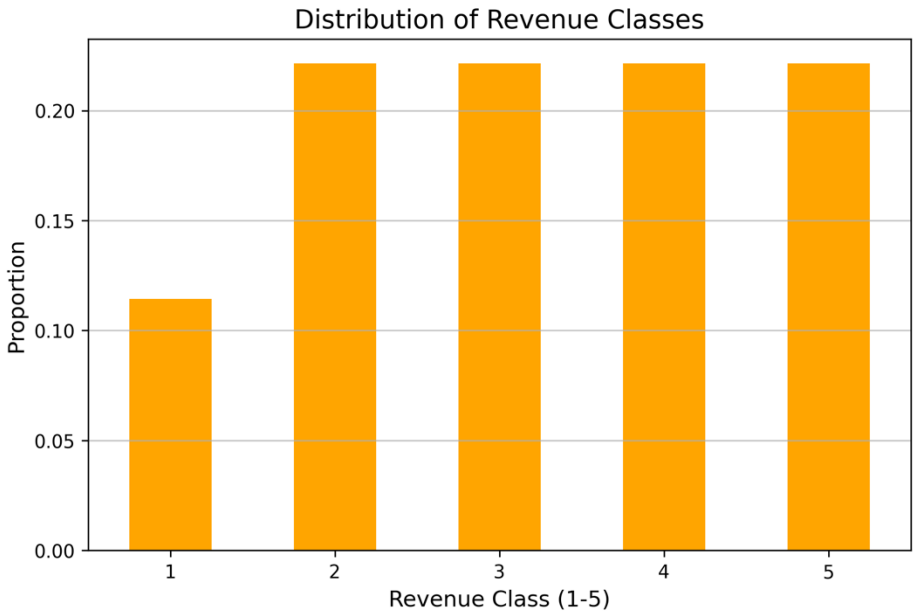


Figure 11: Distribution of the generated revenue classes.

d. Feature Engineering

- Lags (Memory): `_lag_1`, `_lag_2`, `_lag_24`, `_lag_168` were created for `demand_count`, `profit_index`, and `speed_mph` (grouped by `pickup_cluster` to prevent data leakage).
- Rolling Statistics (Momentum): `_rolling_mean_3`, `_rolling_mean_8` and `_rolling_mean_24` were created for `demand_count`, `profit_index`, and `speed_mph` (also grouped by `pickup_cluster` and critically shifted by 1 hour to prevent leakage).
- Cyclical Time Encoding: `hour` and `day_of_week` were extracted and transformed into sin and cos components[3].
- Contextual Flags: `is_weekend` and `is_blizzard` (for the Jan 2016 blizzard) were added.

e. Train/Test Data Split

- The dataset was split chronologically: Training data (Jan 1 - Mar 14, 2016) and Test data (Mar 15 - Mar 31, 2016). This split date was chosen since more than 20% of the dataset was timestamped after that time, so effectively this achieves an 80-20 split.
- The Bridge: A critical step involved concatenating the last 168 hours (7 days) of the aggregated training data with the test data before applying feature engineering to the test set. This ensured that lags and rolling stats could be correctly calculated for the initial hours of the test period without `NaN` values.
- Final datasets were saved, with identical column names and order, and non-essential columns were dropped.

7. Baseline Model(s) and Early Performance

A comprehensive performance evaluation of four distinct models, XGBoost, Random Forest, Logistic Regression, and a sophisticated Naive Baseline (Cluster + Hour Most Frequent), has been conducted.

[3] Is Your Model Time-Blind? The Case for Cyclical Feature Encoding - Gustavo Santos, Dec 24, 2025

a. Performance Summary & Current Leader

The following table summarizes the key performance metrics on the test set:

Model	Test Accuracy	Weighted F1-Score	Macro F1-Score	Mae (Ordinal)
XGBoost (Tuned)	79.12%	0.7908	0.79	0.2119
Random Forest	78.41%	0.7838	0.78	0.2194
Logistic Regression	75.36%	0.7522	0.75	0.2513
Naive Baseline (Cluster + Hour)	68.91%	0.69	0.69	N/A

XGBoost is presently the leading model, demonstrating superior performance across all evaluated metrics. Its test accuracy of 79.12% and a Mean Absolute Error (MAE) of 0.2119 indicate a more accurate and precise prediction capability, signifying that its errors are typically closer to the true revenue class compared to other models.

b. ML vs. Naive Benchmarking

The Machine Learning models have learned beyond mere historical recurrence. The strongest naive baseline, "Cluster+Hour Most Frequent," which predicts the most common revenue class for a given cluster and hour, achieved an accuracy of 68.91%. In contrast, the XGBoost model achieved 79.12% accuracy. This 10.2% absolute improvement suggests that the ML models effectively leverage dynamic features such as real-time lag values and rolling means to capture transient temporal dynamics, adjusting predictions based on prevailing conditions rather than solely on static historical averages.

c. Confusion Matrix Interpretation

Analysis of the confusion matrices (particularly for XGBoost and Random Forest) reveals a highly desirable pattern of misclassification. The "Adjacent Accuracy" metric, which quantifies predictions within one class of the truth, is exceptionally high at 99.74% for XGBoost. This is clearly illustrated in **Figure 12**, which shows the full distribution of ordinal prediction errors: 79.1% of all test predictions are exact matches (off by 0), while 20.6% are off by exactly one class. Meaning the vast majority of errors are "near misses" (e.g., predicting a Class 4 when the true class was a Class 5). Critically, errors of two or more classes are nearly nonexistent

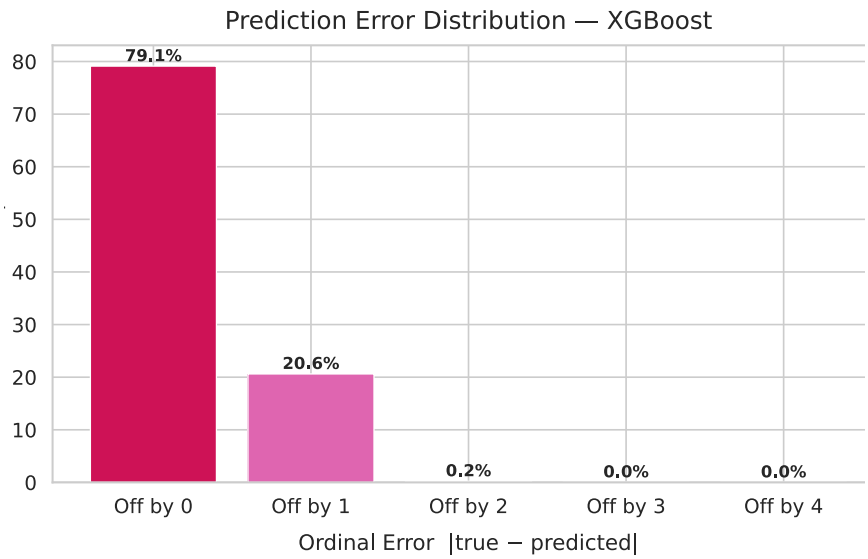


Figure 12: XGBoost prediction error distribution by ordinal distance; 79.1% exact matches, 20.6% off by one class, with near-zero errors beyond that.

at 0.2%, and catastrophic errors of three or four classes occur in essentially 0.0% of cases, confirming that the model never confuses extreme classes such as predicting “Quiet" when the truth is "Surge."

This ordinal robustness is further reinforced by the misclassification rate matrix in Figure 13, which reveals that confusion is consistently concentrated along the diagonal and its immediate neighbors. For instance, Class 1 (Quiet) is most frequently confused with Class 2 (Steady) at 31.0%, and Class 5 (Surge) confusion is almost entirely limited to Class 4 (High Value) at 10.4%, and never reaching lower classes. This structured, proximity-bounded error pattern is operationally significant for a dispatch system: even an approximate revenue potential estimate provides actionable guidance for drivers, as a near-miss prediction still directs them toward the correct neighborhood of demand.

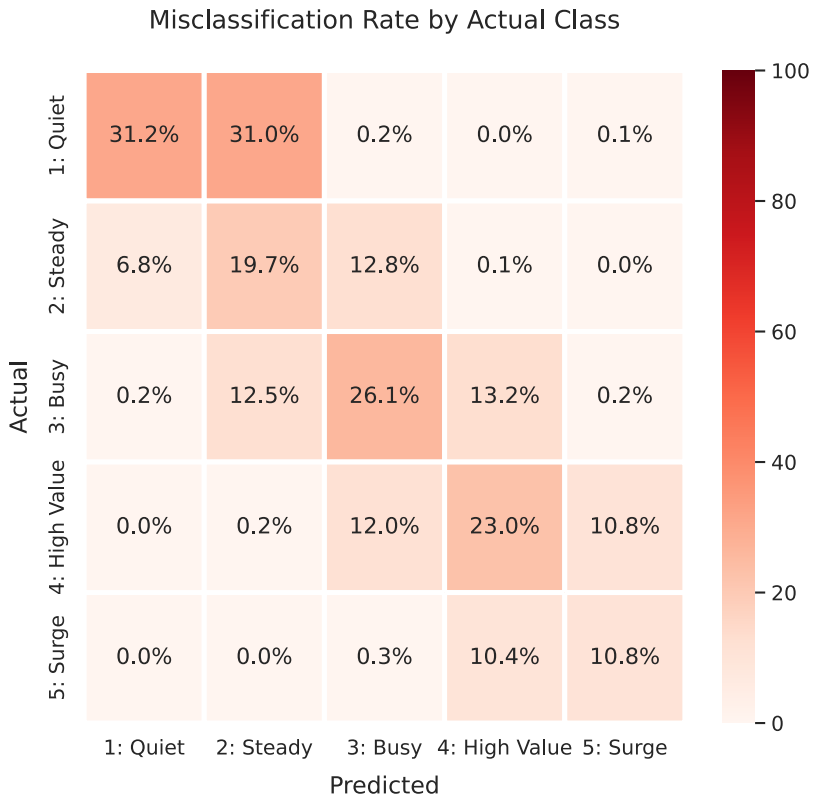
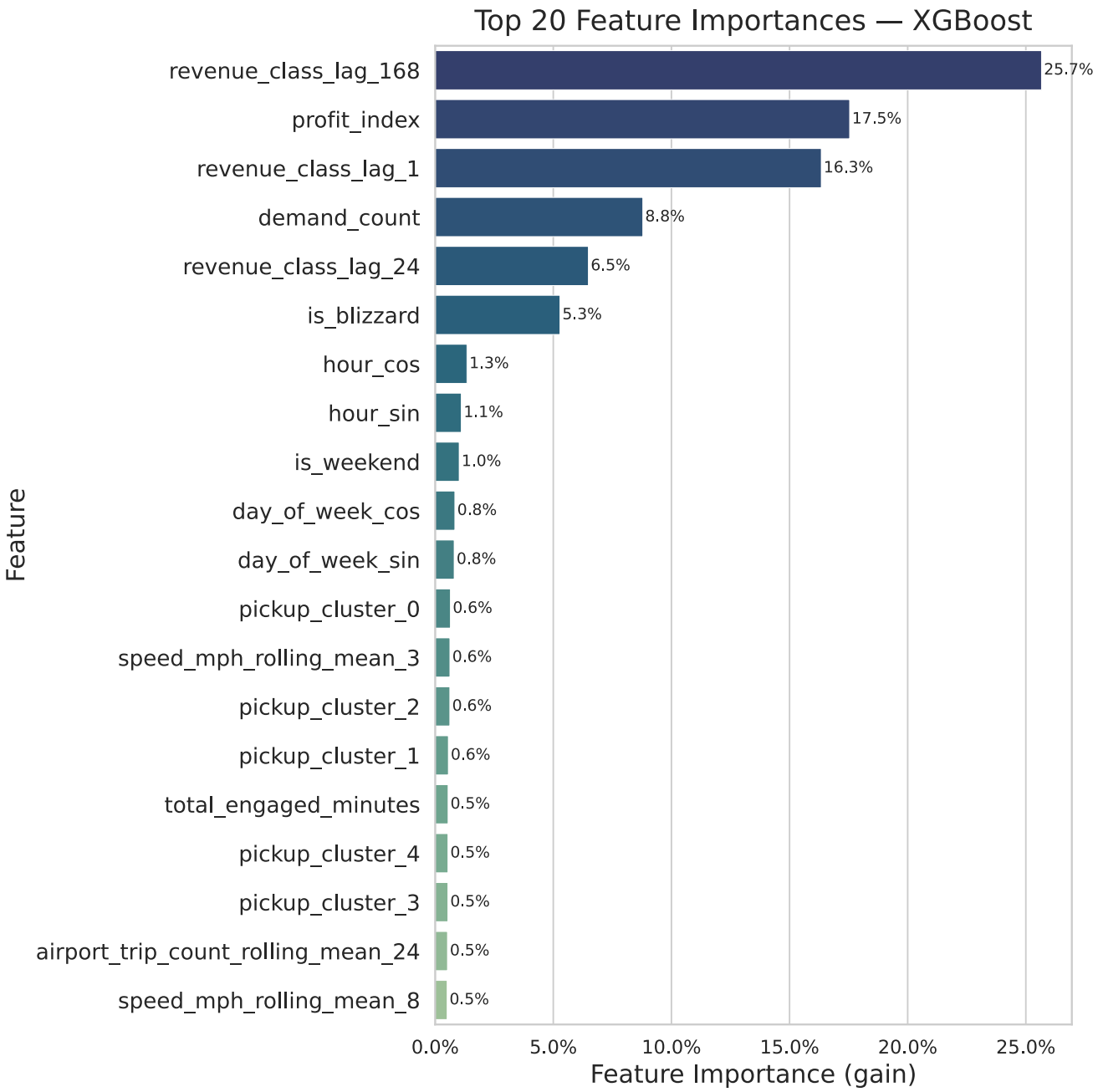


Figure 13: XGBoost misclassification rate matrix showing confusion is tightly bounded to adjacent classes, with no meaningful errors across distant revenue tiers.

d. Feature Importance Analysis:

The feature importance results validate several key engineering decisions made during preprocessing. **Figure 14** shows the top 20 most important features for the best performing XGBoost model. The dominance of revenue_class_lag_1 and revenue_class_lag_168 confirms that the lag features are working exactly as intended, revenue conditions are highly autocorrelated. Meaning the current hour's revenue class is strongly predictable from what happened an hour ago and at the same time last week. This weekly lag signal in particular validates the assumption that NYC ride demand follows consistent day-of-week rhythms worth capturing explicitly. The strong performance of `profit_index` and `demand_count` as the top non-lag features confirms that real-time market conditions carry genuine predictive signal beyond historical patterns alone. Notably, is_blizzard appearing in the top six despite being a simple binary flag demonstrates that

weather disruption events create distinct, learnable demand signatures. The spatial features (`pickup_cluster`) contribute meaningfully but modestly, suggesting location matters but is largely mediated by the temporal and demand features already captured. Overall, the model's reliance on engineered features rather than raw inputs affirms that the feature engineering pipeline added real predictive value.



e. Deep Learning Justification

Despite the strong performance of XGBoost at approximately 79%, a move to Deep Learning models such as LSTMs or Transformers is technically justified for the following reasons:

- **Sequential Pattern Learning:** Tree-based models process lag features as independent snapshots rather than as a continuous sequence. RNN-based LSTMs[4] and Transformers are inherently designed to learn long-term temporal dependencies and the trajectory of time-series data, potentially uncovering more nuanced patterns in demand evolution that static lags cannot fully capture.
- **Performance Issues:** We achieved a 79% accuracy with engineered features and best hyperparameters with sophisticated models. However, further gains with traditional ML algorithms often require increasingly complex and time-consuming feature engineering. Deep learning offers the potential to transcend this plateau by automatically learning hierarchical representations and complex non-linear relationships from raw sequential data.
- **Latent Spatial Relationships:** While encoding helps, current models treat geographic `pickup_cluster` IDs as discrete categories. Deep Learning can incorporate embedding layers for these cluster IDs, allowing the model to learn latent spatial relationships (e.g., that two geographically adjacent clusters might share similar demand dynamics), enriching the model's understanding of the urban environment.

8. Plans for Completion

a. Remaining Tasks

- **Deep Learning Implementation:** Transition from traditional ML baselines to training sequential models, specifically LSTM and Transformer architectures, to better capture complex temporal dependencies.
- **Final Evaluation:** Perform a rigorous final evaluation on the “unseen” March 15–31 Test Set to measure real-world generalizability using the best performing model using an ensemble approach.
- Heatmap Visualization: Develop the final visualization module to transform predicted revenue classes into the interactive "DeepDispatch Heatmap" for driver decision support.

b. Planned Improvements & Experiments

- Sequence Length Optimization: Experiment with different input windows (e.g., 12h vs. 24h vs. 168h) to determine the optimal historical memory for the LSTM.
- Hyperparameter Tuning: Utilize Bayesian optimization to refine model architecture, specifically focusing on dropout rates and hidden units to prevent overfitting.

c. Timeline & Next Steps

- Week 8: Complete Deep Learning training and hyperparameter optimization.
- Week 9: Finalize cross-model performance comparison and generate final heatmaps.
- Week 10: Complete project documentation and finalize the presentation.

9. Challenges

- The dataset contains around 47 millions of rows. Processing this requires efficient code and memory optimization, or it will be extremely slow and computationally expensive.
- The raw data is very noisy. Without aggressive outlier removal, models will try to learn from GPS errors, leading to poor generalization. So, extensive data cleaning and preprocessing was required.

---

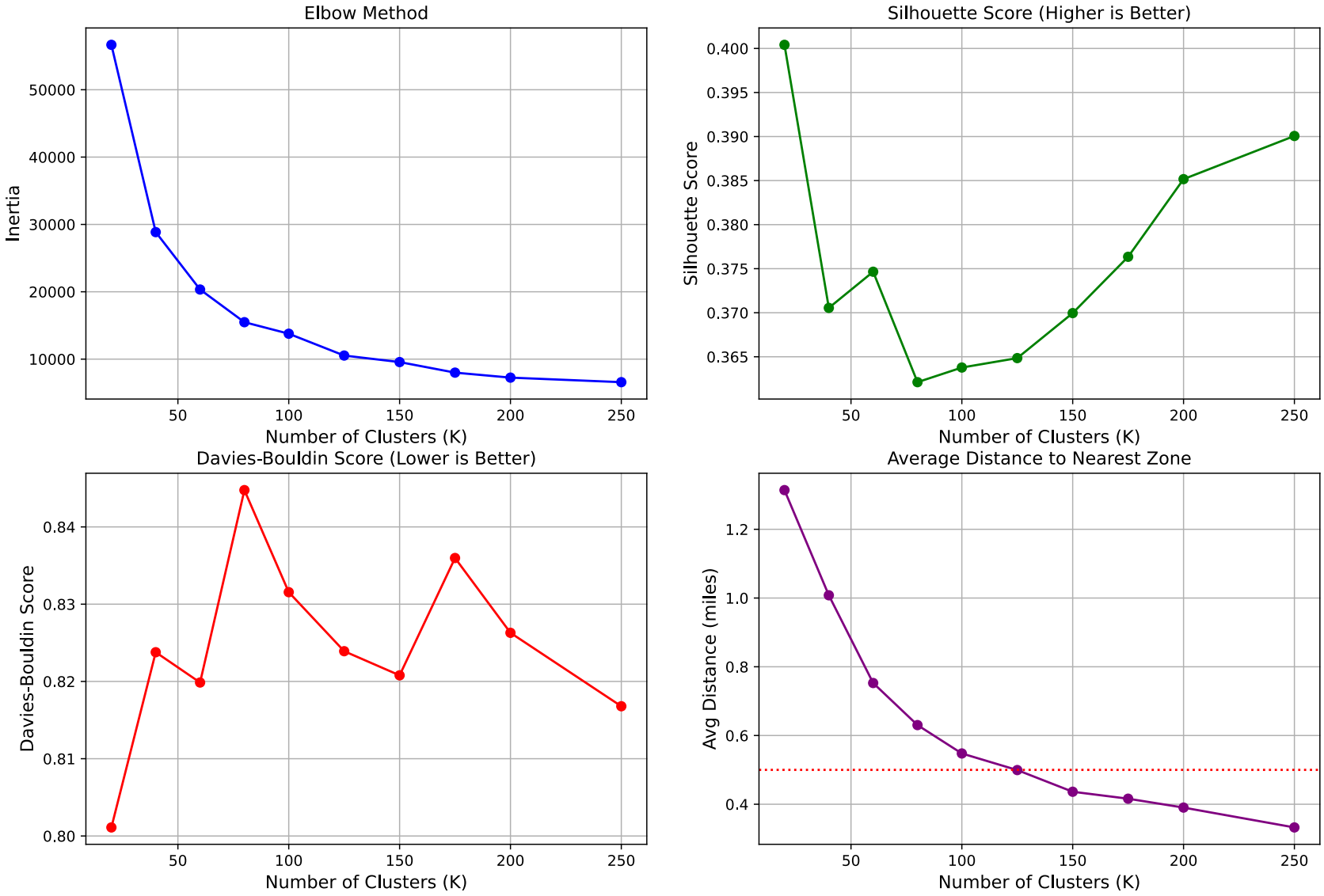
[4] Engineering Extreme Event Forecasting at Uber with Recurrent Neural Networks on Uber Blog.

- GPS coordinates are continuous, so we had to transform them into meaningful spatial regions using clustering before analysis. Defining optimal K=150 zones (approx. 0.48 miles apart) was challenging but crucial for actionable driver repositioning and overcoming overly large, uninformative clusters.
- For exploratory data analysis, using the full dataset was slow, so we used a random sample from the dataset to get insights from the dataset.
- Capturing 24-hour and 7-day demand rhythms via sin/cos encoding, alongside flagging extreme weather events (e.g., Jonas Blizzard), was difficult to figure out but turned out to be useful for accurate prediction.
- Creating lagged features (for example, demand 24 hours ago) is dangerous in a clustered dataset. We had to ensure that lags were generated strictly within each cluster to prevent the history of a high-demand neighborhood from "leaking" into the predictions for a quiet residential zone.
- To optimize hyperparameters using `RandomizedSearchCV` on time-series data required additional attention. We used Scikit-learn's `TimeSeriesSplit` to overcome this[5].
- While the baseline models like could use the current format of the dataset for training and testing, deep learning models like LSTM and Transformer require 3d tensor format `[Batch Size, Time Steps, Features]` as inputs. As a result these models would not require the cyclical encodings and lag features, meaning that we would have to update the training and testing sets.

[5] Scikit-learn's TimeSeriesSplit Documentation.

Appendix

Appendix A: Choosing the appropriate number of clusters



Appendix B: Detailed ML Pipeline Diagram