Uber vs. Lyft Revenue Optimization

DNSC 4281: Pricing and Revenue Management Analytics

Tsion Temesgen (tgtemesgen03@gwu.edu), James O'Connell (joconnell10@gwu.edu), Isabel Barreiro (isabelnbarreiro@gwu.edu), Qingming Zeng (Qingmin.zeng@gwmail.gwu.edu)

Professor Zhengling Qi

George Washington University

December 7, 2025

Executive Summary

This project analyzes how rideshare platforms such as Uber and Lyft can strengthen pricing and revenue management strategies using real market data. Using the Uber vs Lyft Boston dataset from 2018, we built 4 predictive models: linear regression, multiple linear regression, random forest regression, and a neural network classifier. These models identify the factors that drive ride prices, estimate high demand periods, and simulate pricing scenarios. Results show that distance, ride type, and surge multiplier are the strongest price determinants. Weather and most time variables contribute very little. Scenario analysis reveals how prices change with distance and surge level, while elasticity and revenue modeling suggest a simulated revenue peak when rides are aggregated, though this peak is later identified as an artifact of using price per ride rather than a distance-adjusted pricing metric. Together, these tools offer a solid framework to support data-driven dynamic pricing.

1. Introduction

Rideshare pricing is a central part of revenue management because it directly influences profitability, driver availability, and customer experience. Demand for rides varies by time of day, day of week, weather conditions, and special events. Companies therefore depend on dynamic pricing to maintain balance between supply and demand.

This project evaluates 4 questions:

1. Which variables most strongly influence rideshare prices.
2. How accurately machine learning models can predict prices.
3. Whether surge periods can be predicted using classification models.
4. How analytic tools can guide revenue-maximizing pricing decisions.

The Uber vs Lyft Boston dataset includes detailed information on price, distance, surge levels, ride types, timestamps, locations, and weather. These attributes make it ideal for investigating pricing strategies. The goal is to produce insights that are both statistically sound and managerially actionable.

2. Methodology and Data

Dataset Summary

The dataset contains more than 690,000 observations. Each record includes information on price, distance, ride type, surge multiplier, timestamps, pickup and drop-off locations, and 20 plus weather features.

Tools Used

Python and Jupyter Notebook served as the main analysis platform.

1. Pandas and NumPy for data cleaning and structuring.
2. Matplotlib and Seaborn for visualization.
3. Scikit-learn for regression and random forest models.
4. TensorFlow and Keras for the neural network classifier.
5. ChatGPT for reasoning support, documentation, and model interpretation.

Linear Regression

Linear regression served as the starting point for understanding the fundamental drivers of rideshare pricing. Because linear regression estimates the marginal effect of each predictor on price, it provides a highly interpretable view of how factors such as distance, ride type, or time of day correlate with cost. This model was especially useful for checking the expected economic relationships, such as the positive association between trip distance and price or the premium attached to luxury ride categories. Although linear regression cannot capture the full complexity of the pricing environment, it acts as a baseline diagnostic tool. It helps verify data quality, identify anomalous patterns, and establish whether the direction and magnitude of relationships make economic sense before moving to more advanced models. Its simplicity also makes it a valuable reference against which more flexible machine learning models can be compared.

Multiple Linear Regression

Multiple linear regression extended the baseline model by allowing many predictors to influence price simultaneously. Rideshare pricing is affected by a combination of distance, ride type, weather, time, and location, which means that an additive multivariate model helps quantify each factor's contribution while holding others constant. This approach provides a more realistic representation of the price formation process and supports hypothesis testing regarding specific variables. The model was also used to evaluate goodness-of-fit improvements over the simple linear model and to assess whether additive effects were sufficient to explain pricing patterns. However, multiple linear regression still assumes linearity and additivity, which limits its ability to model surge pricing or nonlinear interactions among variables. Nevertheless, the model played a critical role in the methodological sequence by offering transparency and interpretability before introducing more sophisticated machine learning methods.

Random Forest Regression

Random forest regression was selected as the primary predictive model for estimating ride prices due to its ability to capture nonlinear relationships and complex interactions among features. Unlike linear models, random forests do not impose structural assumptions on the price function and can naturally handle high-dimensional categorical variables such as ride type and location. This makes them ideal for modeling real-world pricing systems where relationships are rarely additive. The model achieved high predictive accuracy, demonstrated by a low mean

absolute error and a strong R^2 value, and produced feature importance scores that identified the variables most responsible for price variation. These include ride category, distance, and surge multiplier, all of which are central to pricing strategy. Random forest also enables scenario simulation, which allows the team to explore how predicted prices change under different distances, surge conditions, and ride types. This capability directly supports revenue management by highlighting price sensitivities and providing a structured foundation for what-if analysis and dynamic pricing planning.

Neural Network for High-Demand Classification

The neural network model was designed to identify periods of high demand associated with surge pricing, a classification problem where the minority class represents only a small portion of rides. Neural networks were chosen because they outperform linear and tree-based classifiers when relationships are highly nonlinear or influenced by many interacting variables, such as time, weather, and geography. The model incorporated class weighting to address the severe class imbalance and was optimized using sigmoid activation and binary cross-entropy loss. It produced meaningful probability forecasts that enabled threshold tuning to balance recall and precision. In revenue management, failing to detect a true high-demand period is costly because it may result in missed surge pricing opportunities. For this reason, the neural network was evaluated across different decision thresholds to maximize recall while controlling false positives. The model ultimately provided a probabilistic early-warning system, helping determine not only when surge pricing is likely but also how aggressively the platform should respond.

Additional Revenue Management Analyses

Beyond predictive modeling, the study incorporated several analytical tools grounded in revenue management principles to interpret and apply model outputs. Scenario simulations were used to quantify how predicted prices respond to changes in distance, ride category, and surge multiplier, allowing the team to evaluate pricing under a range of operational conditions. The elasticity analysis explored the responsiveness of predicted prices to underlying variables, which helps determine where pricing leverage is strongest. High-demand probability forecasts from the neural network were used to examine threshold-based strategies that balance missed revenue opportunities against customer experience concerns. Together, these analyses bridge the gap between statistical modeling and business decision-making. They help translate model predictions into actionable insights regarding pricing optimization, demand forecasting, and capacity management strategies for platforms like Uber and Lyft.

3. LLM Application

Large Language Models, specifically OpenAI's ChatGPT, supported this project as an analytical assistant that enhanced the accuracy, efficiency, and rigor of the modeling process. While all coding, testing, and model execution were completed by the project team, ChatGPT was used to strengthen methodological decisions and improve the quality of implementation. For example,

ChatGPT provided guidance on best practices for handling missing values, recommending median imputation for skewed numeric features and placeholder imputation for categorical variables. It also suggested appropriate encoding and scaling techniques, including one-hot encoding for high-cardinality ride type categories and the use of standardized scaling prior to neural network training. These recommendations helped prevent data leakage, improve model stability, and ensure that preprocessing choices aligned with industry standards.

LLMs were especially valuable when addressing advanced modeling challenges. For the neural network surge classifier, ChatGPT explained how to address class imbalance through class weighting and why threshold tuning is essential when the minority class represents only about 3 percent of the dataset. It provided conceptual justification for model architecture decisions, including the use of ReLU activation functions, sigmoid output for binary classification, and cross-entropy loss. When errors arose during model evaluation, such as NaN issues in probability predictions or shape mismatches in encoded feature matrices, ChatGPT helped diagnose their causes and propose corrective strategies that were then implemented manually. These interactions improved the robustness of the modeling pipeline without automating any part of the statistical analysis.

In addition to technical support, LLMs contributed by helping interpret statistical results in the context of revenue management. ChatGPT clarified how specific performance metrics, such as high recall with lower precision, map to pricing strategies where the cost of missing surge events outweighs the cost of occasional false alarms. It also helped connect elasticity estimates, scenario simulations, and pricing curves to the broader objectives of dynamic pricing. Throughout the project, ChatGPT served as a tool for reasoning, validation, and problem solving, ensuring that the final models were implemented correctly and interpreted through a revenue management lens. The project relied on human judgment, coding, and decision-making, while LLMs provided supplemental expertise that enriched the analytical process.

4. Analysis and Results

4.1 Linear Regression Results

For our Linear Regression analysis, our goal was to evaluate which predictors had the strongest influence on ride price. From the original 53 predictors, we narrowed the list to 10 meaningful variables. Many features, such as humidity or population density, showed minimal explanatory power and were removed to focus on factors that more directly affect pricing.

The key predictors we analyzed in depth were Hour, Distance, Surge Multiplier, and Ride Type, allowing us to explore both positive and negative relationships with price.

Hour of day showed a slope of only 0.0008 and an R^2 close to 0%, indicating that time of day does not meaningfully affect ride price. In other words, prices remain relatively stable across hours, and hour alone is too weak to be a meaningful predictor.

Distance, however, showed a clear positive relationship with price. The slope of 2.8337 suggests that each additional mile adds approximately $2.83 to the ride total. Its R^2 of 11.9% shows that distance explains more variation than hour, but still leaves a large portion of price unexplained, implying that additional predictors are needed.

Next, we evaluated Surge Multiplier and Ride Type, where we found the strongest effects. Premium ride types such as Lux Black XL and Black SUV were approximately $10–12 more expensive than the baseline Lyft, while economy and shared options like UberX, UberPool, and Shared were $10–14 cheaper. Ride type alone creates clear price tiers.

However, we realized that surge multiplier is not best captured through a traditional linear (additive) model, since linear regression assumes each predictor contributes a fixed dollar amount. In reality, surge pricing behaves multiplicatively. For this reason we decided to use a log-linear model that provides a more accurate interpretation. It shows proportional price changes rather than fixed increases. For example, instead of adding a dollar amount, a +1 increase in surge might increase the price by 35%. This model also reveals important interaction effects, such as premium rides amplifying surge prices more than economy rides. Overall, the log-linear approach better reflects how Uber and Lyft actually calculate fares, showing that surge impacts price as a scaling factor, not a fixed adjustment.

All of these predictors helped us determine the level of importance when trying to predict price.

4.2 Multiple Linear Regression

After constructing several simple linear regressions, we selected the strongest predictors, distance, surge multiplier, and ride type, and used them to build a multiple linear regression model predicting ride price. This model performed extremely well, yielding an R^2 of 93% and an RMSE of $2.52.

The variance (R^2) indicates that 93% of the variation in ride prices can be explained using only these three predictors, demonstrating that they capture nearly all of the meaningful price-driving information. Meanwhile, the RMSE of $2.52 shows that the model's predictions are, on average, within about $2.50 of the actual ride price.

From the coefficients, we found that for every +1 increase in surge multiplier, the expected ride price increases by approximately $18.32, holding distance and ride type constant. This reveals the strong multiplicative impact of surge, even in an additive model.

These findings reinforce that shared rides remain the cheapest options while luxury rides are consistently the most expensive. They also reveal that surge and ride type are the dominant drivers of price, with distance contributing as expected but producing smaller changes compared to the large jumps created by different ride categories.

4.3 Random Forest Regression

The random forest regression model produced the strongest overall performance of all price prediction methods explored in this project. With an MAE of 1.17 dollars and an RMSE of 1.74 dollars, the model consistently predicted ride prices within a narrow error band relative to true values. More importantly, the model achieved an R squared value of 0.963, meaning it explained 96.3 percent of all variation in price across more than 690,000 rides. This level of accuracy is unusually high for real-world transportation data, which often contain noise associated with unpredictable rider decisions, supply fluctuations, and platform adjustments.

The feature importance results from the random forest provide valuable insight into the underlying structure of rideshare pricing. Ride type emerged as the dominant driver of price, reflecting the platform's reliance on product segmentation and willingness to pay. For example, UberX, UberXL, Lux, and Black each represent distinct positioning strategies, and these categories have well-established fare structures that influence total price even before considering distance or surge. The model's ability to capture this segmentation reinforces the idea that pricing algorithms are built upon tiered service levels and not purely distance-based rules.

Distance was the second most important feature. This makes sense because the majority of fare revenue comes from a per-mile and per-minute formula. The interaction between distance and ride type is also critical. Luxury ride types accumulate distance-based charges at higher rates, which magnifies the effect of distance for certain categories. The random forest was able to learn these interactions, unlike the linear model which assumes strictly additive effects.

Surge multiplier was also a strong predictor but less important than ride type and distance. Surge only occurs during periods of tight supply and elevated demand, which means it affects a small portion of trips. For the trips that do experience surge, however, the multiplier meaningfully shifts prices upward. The random forest captured these nonlinear and conditional relationships without requiring explicit equations.

Weather variables, including temperature, humidity, precipitation probability, and cloud cover, contributed almost nothing to predictive performance. This does not imply that weather has no effect on demand. Instead, it means that weather affects price indirectly through its influence on rider behavior and driver availability. That influence is already embedded in the surge multiplier itself. Therefore, once surge is included as a feature, explicit weather effects become redundant.

The success of the random forest demonstrates that rideshare pricing algorithms are structured, predictable, and dominated by relatively few factors. This model provides a practical tool for simulating how price responds to changes in ride type distribution, demand patterns, and distance configurations across different neighborhoods. From a revenue management perspective, the model can be repurposed to evaluate alternative fare structures, explore the impact of minimum fare adjustments, and test hypothetical pricing strategies before implementation.

4.4 Neural Network: High Demand Classifier

The neural network classifier was designed to forecast whether a given ride would fall into a high demand period, defined as any instance in which the surge multiplier exceeded 1. This is an important forecasting task in rideshare revenue management because surge pricing is central to balancing market supply and demand. Surge conditions reflect periods in which the platform faces pressure to maintain service quality, incentivize drivers, and avoid long wait times or service failures.

The dataset contained significant class imbalance. Only about 3 percent of rides experienced surge pricing. Without correction, the model would learn to predict the majority class and ignore the surge class entirely. To address this issue, class weights were implemented so that the model received a stronger penalty for missing surge events. This approach enabled the neural network to detect rare high-demand conditions with much greater sensitivity. The final neural network consisted of 70 input features, two hidden layers with 64 and 32 units, and a sigmoid output layer for binary classification. The model achieved a test accuracy of about 75 percent and an ROC AUC of 0.883. The ROC AUC value is especially meaningful because it reflects the model's ability to rank rides by likelihood of surge, independent of any particular threshold. A value near 0.9 indicates that the neural network is highly capable of distinguishing surge events from non-surge events even in the presence of substantial imbalance.

The Precision Recall curve provides additional insight. Since only 3 percent of rides are surge rides, a naive classifier would achieve a precision equal to the base rate of 3 percent. The neural network achieved a precision of 9 to 10 percent depending on the threshold used. Although this appears low in absolute terms, it represents a performance that is roughly 3 times better than random chance. In an imbalanced environment, precision values must be interpreted relative to prevalence rather than in isolation. Recall values were much higher and ranged from 92 to 98 percent. Recall is more important than precision in this context because failing to identify surge periods leads to missed revenue opportunities and potential operational failures such as long wait times. False positives are less harmful because activating surge prematurely

may temporarily raise prices but does not disrupt system function. For this reason, threshold selection focused on maximizing recall while containing false positives to an acceptable level.

A threshold of 0.30 was identified as the best operating point. At this threshold, the model detected about 96.6 percent of surge events while maintaining a precision of 9.4 percent. Lowering the threshold to 0.20 boosted recall to nearly 98 percent but increased false positives to the point of diminishing operational usefulness. A higher threshold of 0.50 reduced false positives but missed too many surge events. Therefore, the 0.30 threshold offers the most favorable trade-off between missed revenue and operational noise.

From a revenue management perspective, the neural network can serve as an early warning system for upcoming demand spikes. When used alongside historical demand patterns or real time rider volume data, the model can help platforms prepare for peak periods, adjust incentives for drivers, balance supply geographically, and activate surge pricing with better timing. This forecasting capability is essential for managing the two-sided marketplace structure inherent to the rideshare industry.

5. Recommendations and Business Impact

The findings from this project highlight a number of actionable strategies that the rideshare platforms can implement to enhance pricing efficacy, operational efficiency, and revenue performance overall. Distance, ride type, and surge multiplier proved to be strong determinants of price, suggesting fare structures need to be carefully optimized across service categories and trip lengths rather than relying on distance-based formulas. Strong performance of the random forest model supports using data driven stimulations in order to evaluate alternative fare structures, minimum fare adjustments, and ride-type configurations before actual implementation.

Therefore, the neural network surge classifier provides a valuable early warning system for upcoming spikes in demand, and platforms can proactively switch on surge pricing, reposition drivers, and minimize disruptions in service. Simulations of various scenarios and elasticity analyses also reveal a simulated revenue peak when prices are aggregated at the ride level; however, this benchmark is not directly actionable because ride prices are fundamentally determined on a distance-adjusted (per-mile) basis rather than as a single price per ride.

The collection of further streams of data, including real-time ride request volume, driver availability, traffic conditions, and event schedules, can further improve the accuracy of surge predictions and operational planning.

With these insights, the platforms will be able to align driver incentives with predicted demand more effectively, reduce wait times for their customers, enhance the balance between supply and demand, and improve short-term profitability along with long-term market growth. This analytics-driven framework reinforces a more responsive and strategically grounded pricing system that can scale across diverse markets.

6. Limitations and Future Work

While this project demonstrates the strong potential of machine learning and revenue optimization tools for rideshare pricing, several important limitations should be acknowledged. These limitations also point directly to meaningful directions for future research and model enhancement.

6.1 Data Scope and Generalizability

The analysis is based exclusively on the Boston Uber vs. Lyft dataset from 2018. Although the dataset is large and rich in features, it represents only one geographic market and one specific time period. Rideshare demand patterns, pricing structures, and rider behavior may differ substantially across cities due to differences in population density, public transportation systems, regulatory environments, and income levels. In addition, the rideshare market has evolved significantly since 2018, particularly following the COVID-19 pandemic, changes in labor policies, and platform-specific pricing reforms.

As a result, the conclusions drawn in this study may not fully generalize to other cities, time periods, or current market conditions. Future research should extend this framework to multi-city datasets and more recent observations in order to capture regional heterogeneity and changing demand behavior over time.

6.2 Definition of High Demand and Surge Events

In this project, high demand periods were defined by a surge multiplier greater than 1. While this definition is intuitive and operationally convenient, it represents a simplified proxy for true demand-supply imbalance. Surge pricing may be influenced not only by rider demand, but also by driver availability, platform-specific pricing policies, and algorithmic adjustments that are not directly observable in the data.

This limitation affects the neural network classifier, especially explaining why precision is low even when recall is very high. Some instances flagged as surge by the model may not represent genuine market-wide demand spikes, but localized or temporary pricing adjustments.

Future work could incorporate additional indicators of demand pressure such as ride request counts, driver supply levels, wait times, or cancellation rates. These variables would allow for a more economically grounded and precise definition of high-demand conditions.

6.3 Potential Omitted Variables

Although the dataset contains extensive weather variables and detailed ride information, it lacks several key demand drivers that are known to influence rideshare pricing. These include major events, concerts, sports games, holidays, traffic congestion levels, airport activity, and public

transit disruptions. Such factors can create sudden, localized demand spikes that are not captured by standard time or weather variables.

The omission of these variables may partially explain why timestamp features and weather features show very low importance in the random forest results. In reality, time-of-day and day-of-week effects may interact strongly with unobserved event activity.

Future extensions could integrate external data sources, such as event calendars, traffic sensors, and airport arrival volumes, to strengthen both price prediction accuracy and surge detection capability.

6.4 Model Structure and Interpretability Trade-Offs

While random forest regression achieved excellent predictive performance, it is inherently less interpretable than linear models. Although feature importance rankings provide useful insights, they do not clearly quantify causal relationships between variables and price. Likewise, the neural network classifier offers strong detection capability but functions as a black-box model.

From a managerial perspective, decision-makers often prefer transparent pricing rules that can be clearly justified to regulators, drivers, and customers. The reliance on complex nonlinear models may therefore reduce interpretability even as predictive accuracy improves.

Future research could explore hybrid modeling approaches that balance accuracy and explainability, such as generalized additive models (GAMs), SHAP-based interpretability for tree models, or rule-based approximations of neural network behavior.

6.5 Revenue Optimization Based on Model-Implied Demand

Revenue maximization in this project was conducted using model-implied demand rather than observed market demand under controlled pricing experiments. A major limitation of this analysis is that the revenue peak was reported in terms of price per ride (approximately $27.50), rather than being expressed in a distance-adjusted metric such as price per mile, which is the true economic driver of fares in the rideshare pricing structure.

Because ride prices in this market are fundamentally determined by distance-based pricing formulas, aggregating optimization results into a single dollar amount per ride mixes together trips of very different lengths and ride types. As a result, the $27.50 “optimal price per ride” does not represent a stable or actionable pricing recommendation, and this interpretation was therefore a conceptual mistake in the original analysis. The value reflects an artifact of aggregating heterogeneous trips rather than a true structural optimum in pricing behavior.

Moreover, optimizing revenue on a per-ride basis ignores the fact that longer trips naturally generate higher revenue even if their marginal profitability remains unchanged. This makes the $27.50 figure highly sensitive to the trip length distribution in the dataset and limits its external validity and managerial usefulness.

Future research should correct this limitation by conducting revenue optimization in price-per-mile or marginal price terms, which would directly align with how fares are actually constructed in practice. In addition, future studies should use observed demand responses under controlled pricing experiments or dynamic pricing simulations to estimate revenue-maximizing price functions more realistically. These improvements would allow optimization results to translate into actionable pricing policies rather than aggregated observational benchmarks.

6.6 Future Research Directions

Building on the limitations above, several promising extensions can enhance the analytical framework developed in this project:

1. Expanding the dataset to include multiple metropolitan areas and more recent time periods.

2. Integrating real-time demand indicators such as ride request volume, driver availability, and wait time.

3. Incorporating event and traffic data to improve surge prediction accuracy.

4. Developing explainable machine learning models for pricing transparency.

5. Applying dynamic optimization and reinforcement learning to simulate adaptive surge strategies over time.

Together, these improvements would allow future research to move from predictive and diagnostic analytics toward fully prescriptive and real-time pricing systems.

References

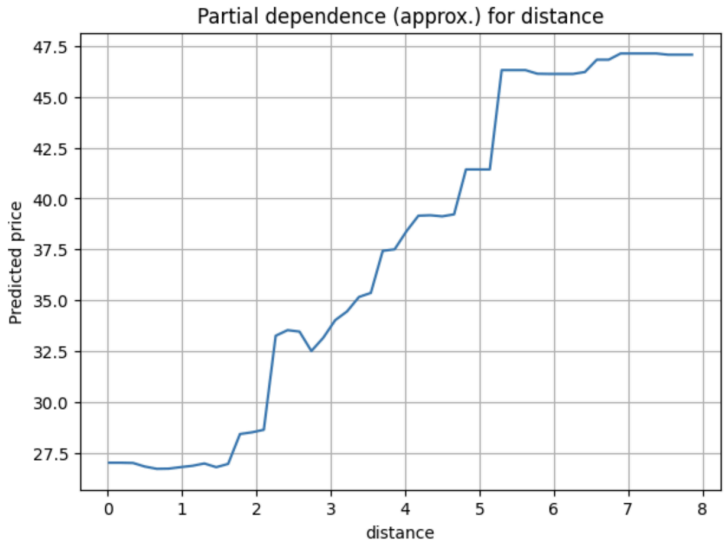
- Uber vs Lyft Boston Dataset
- Python libraries including Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, TensorFlow, Keras
- Course material from Revenue Management Analytics

Appendices

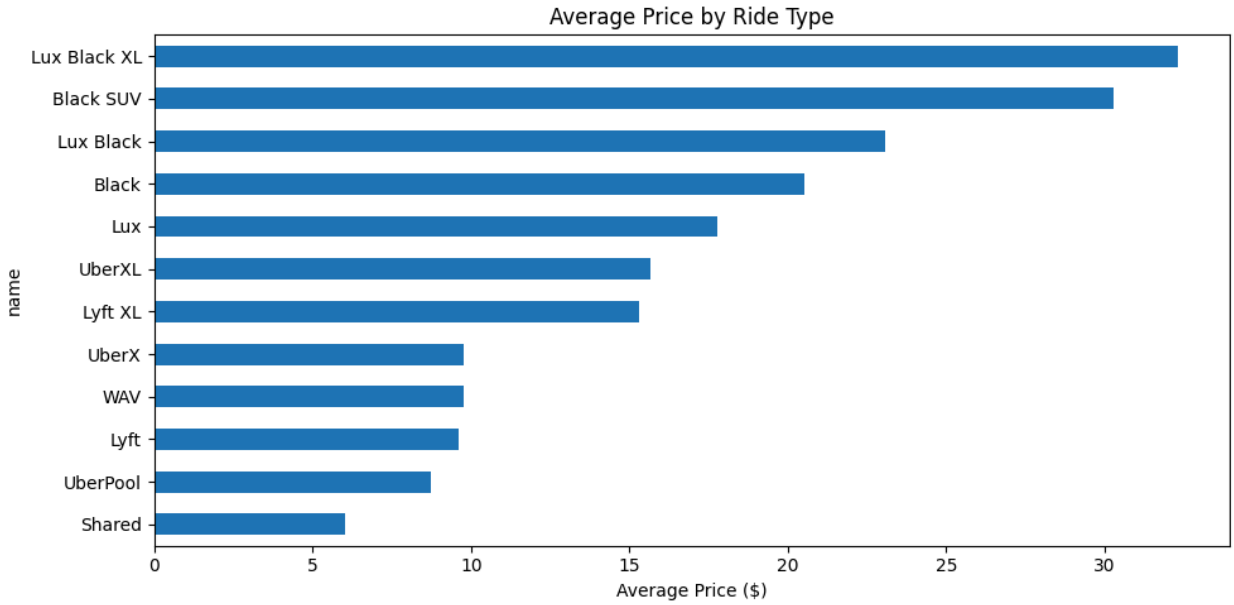
Additional Figures

Random Forest Partial Dependence Plot (Predicted Price vs. Distance)

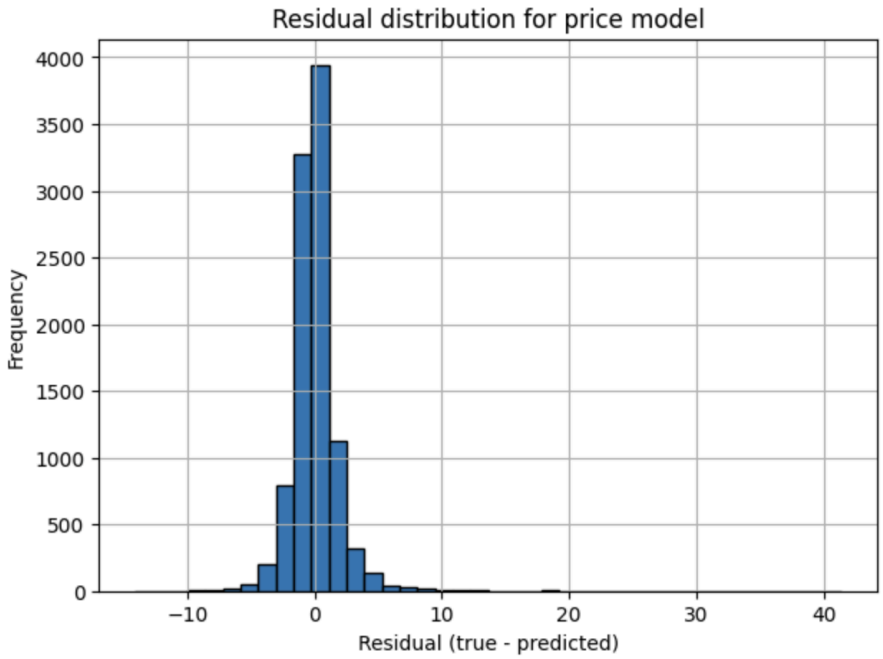
```
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.0s finished
```



Average price by ride type

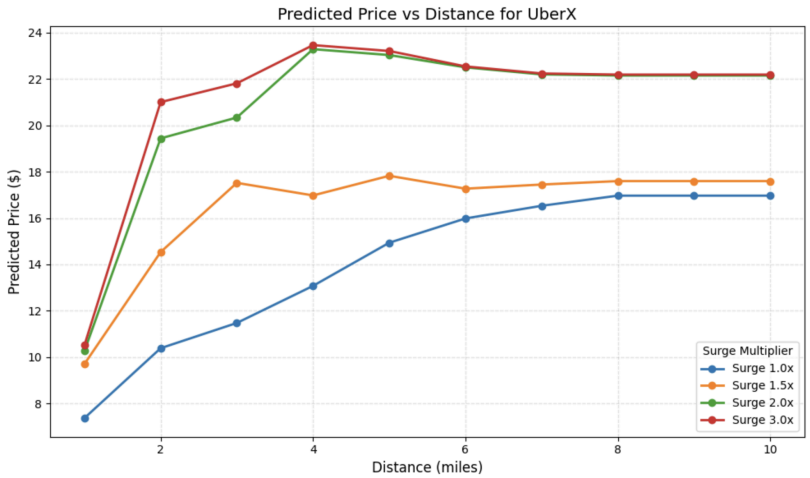


Random Forest - Residual Distribution

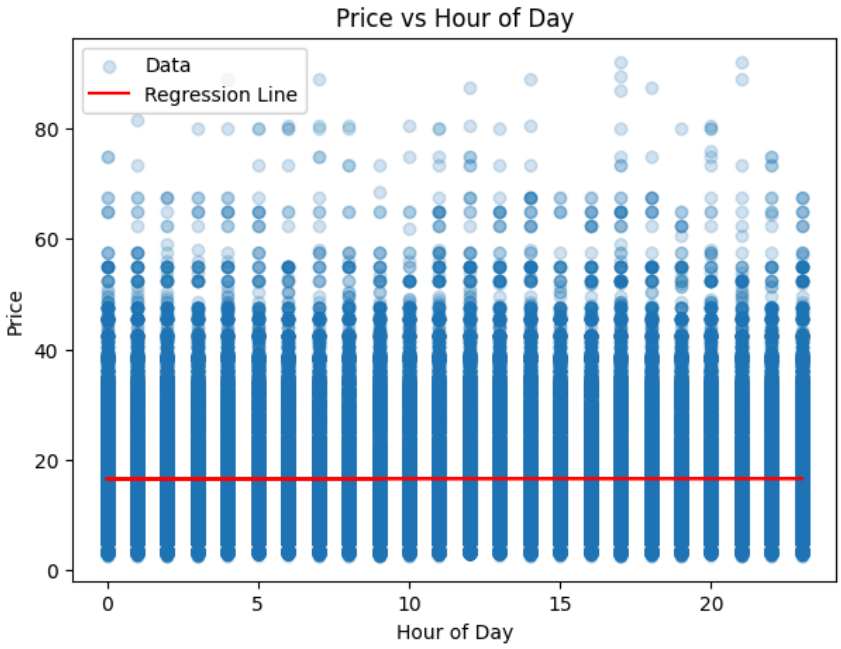


Random Forest - Residuals vs Predicted Price

Random Forest Scenario Analysis - Predicted Price vs Distance for UberX



Simple Linear regression: Price - Hour



Simple Linear Regression Price vs Distance

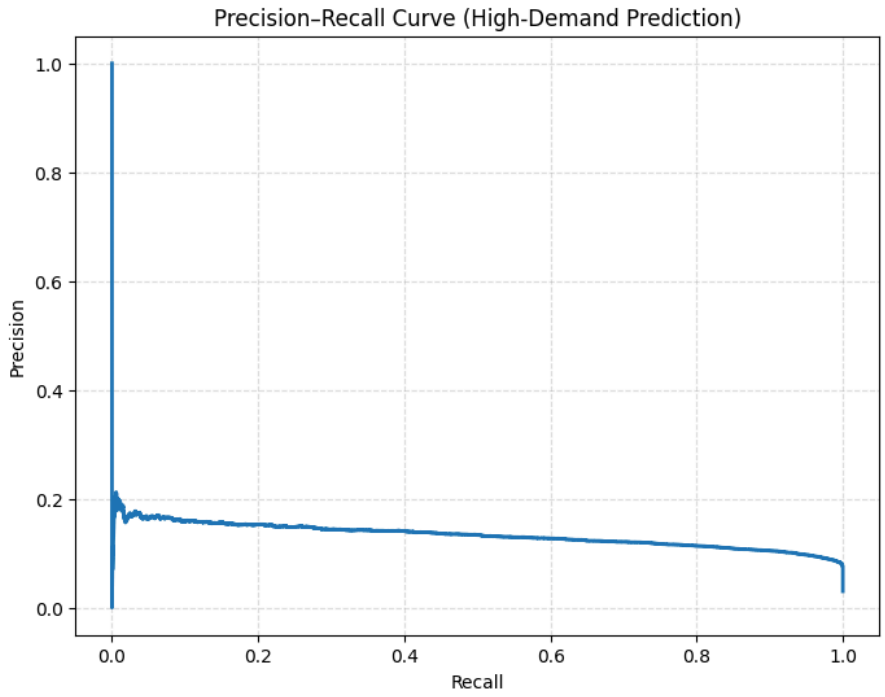
Multiple Linear Regression Model for Car type, Surge, Distance

```
=== Interaction Model Results ===
RMSE: 2.4540379334917897
R²: 0.9309213926758941

=== Coefficients ===
                    Feature  Coefficient
6         name_Lux Black XL    11.135978
3            name_Black SUV     9.762539
1          surge_multiplier     6.053397
2   dist_surge_interaction      5.098488
5           name_Lux Black      1.876800
0                 distance     -2.367814
4                 name_Lux     -3.411133
12              name_UberXL    -4.836176
8              name_Lyft XL    -5.887086
11               name_UberX   -10.748724
13                 name_WAV   -10.750853
7                 name_Lyft   -11.596073
10             name_UberPool   -11.761642
9                name_Shared   -14.479880
```

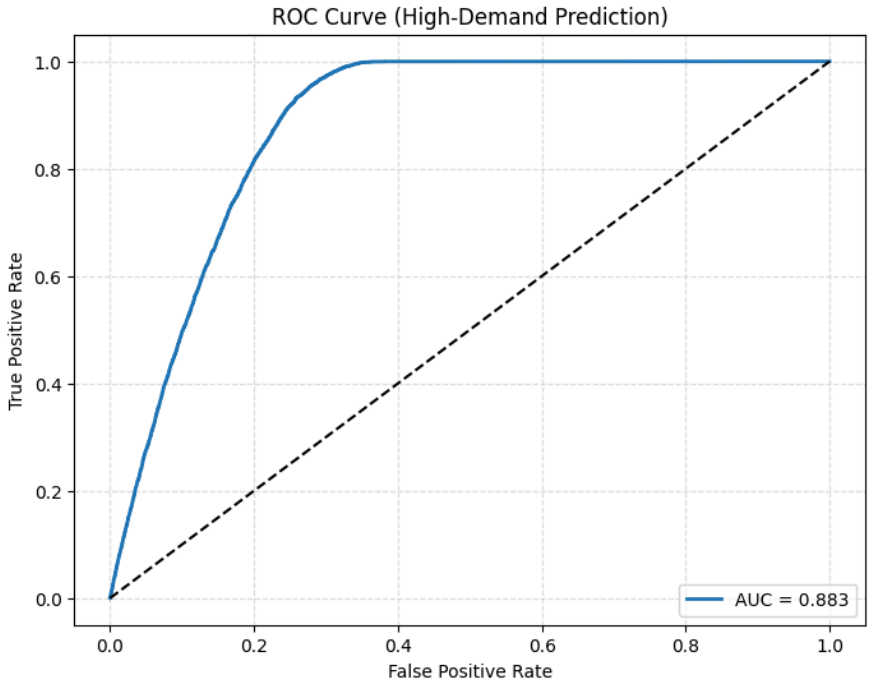
Neural Network - Threshold Tuning

```
=== Neural Network High Demand Model (with class weights) ===
Test Accuracy: 0.747
4332/4332 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step

Number of NaNs in predictions: 0
Number of finite predictions: 138615 out of 138615
ROC AUC       : 0.883

--- Threshold = 0.50 ---
Confusion Matrix:
[[99666 34754]
 [  287  3908]]

Classification Report:
              precision    recall  f1-score   support

           0      0.997     0.741     0.850    134420
           1      0.101     0.932     0.182      4195

    accuracy                          0.747    138615
   macro avg      0.549     0.837     0.516    138615
weighted avg      0.970     0.747     0.830    138615


--- Threshold = 0.30 ---
Confusion Matrix:
[[95141 39279]
 [  138  4057]]


--- Threshold = 0.30 ---
Confusion Matrix:
[[95141 39279]
 [  138  4057]]

Classification Report:
              precision    recall  f1-score   support

           0      0.999     0.708     0.828    134420
           1      0.094     0.967     0.171      4195

    accuracy                          0.716    138615
   macro avg      0.546     0.837     0.500    138615
weighted avg      0.971     0.716     0.808    138615


--- Threshold = 0.20 ---
Confusion Matrix:
[[93336 41084]
 [  101  4094]]

Classification Report:
              precision    recall  f1-score   support

           0      0.999     0.694     0.819    134420
           1      0.091     0.976     0.166      4195

    accuracy                          0.703    138615
   macro avg      0.545     0.835     0.493    138615
weighted avg      0.971     0.703     0.799    138615
```

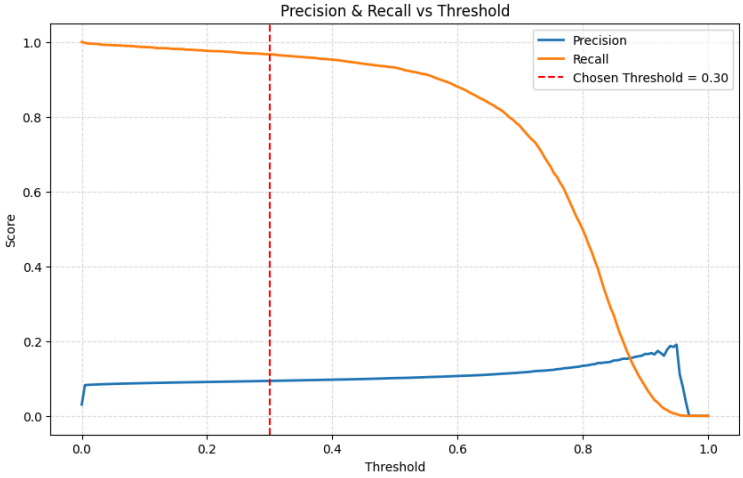
Neural Network - Precision/Recall Curve



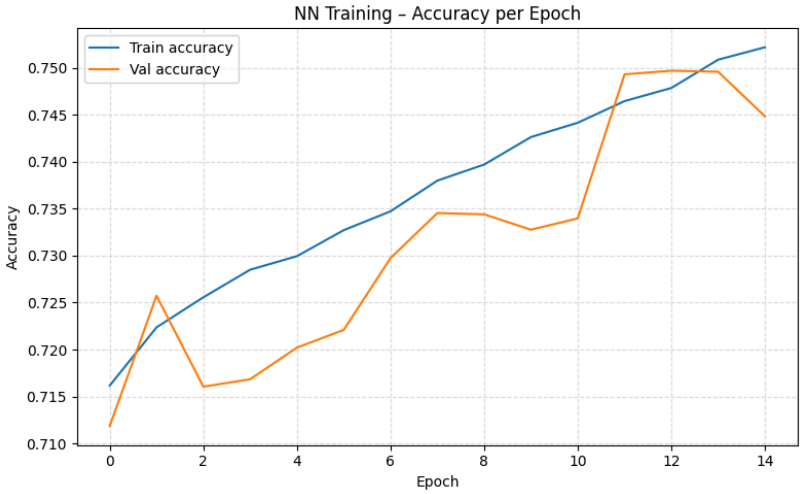
Neural Network - ROC Curve for High-Demand Prediction



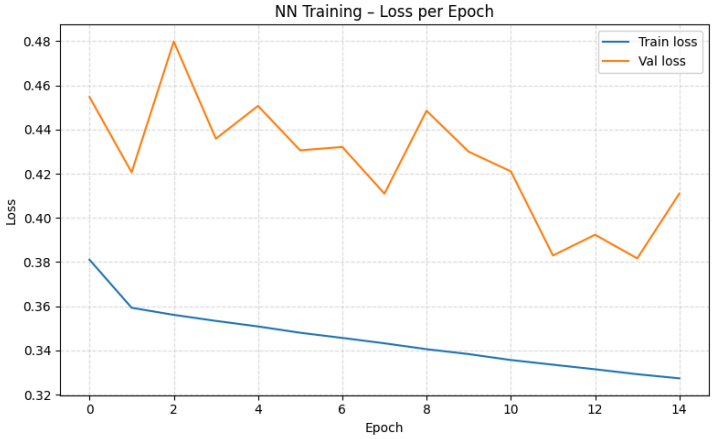
Neural Network - Precision/Recall vs Threshold



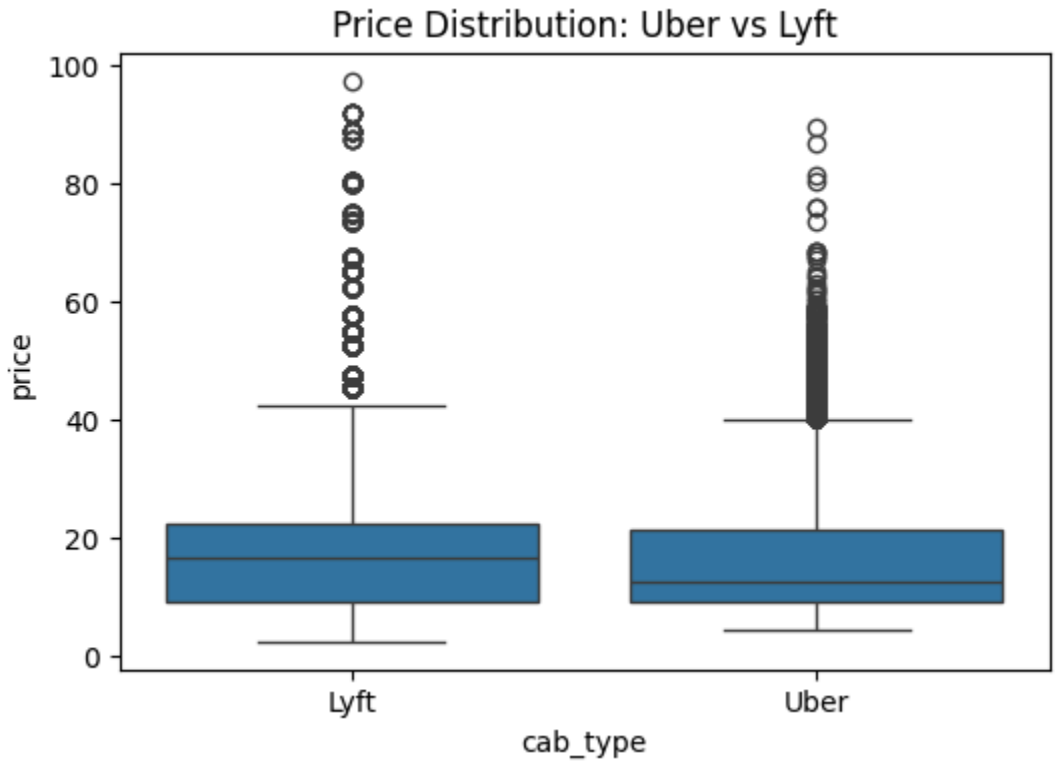
Neural Network - Accuracy per Epoch



Neural Network - Loss per Epoch

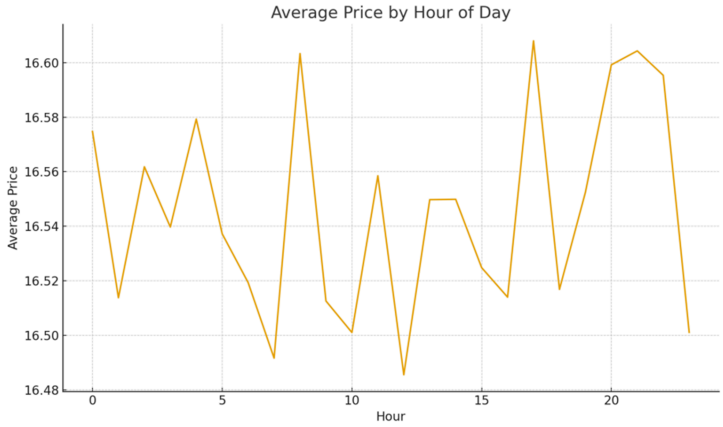


Price Distribution Uber vs Lyft

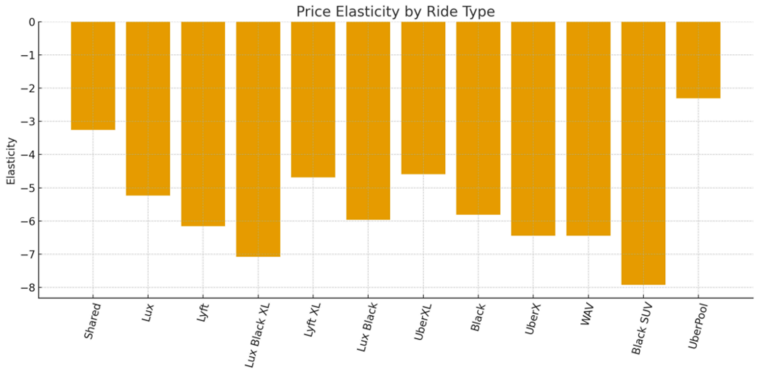


Demand Curve - Price vs Ride

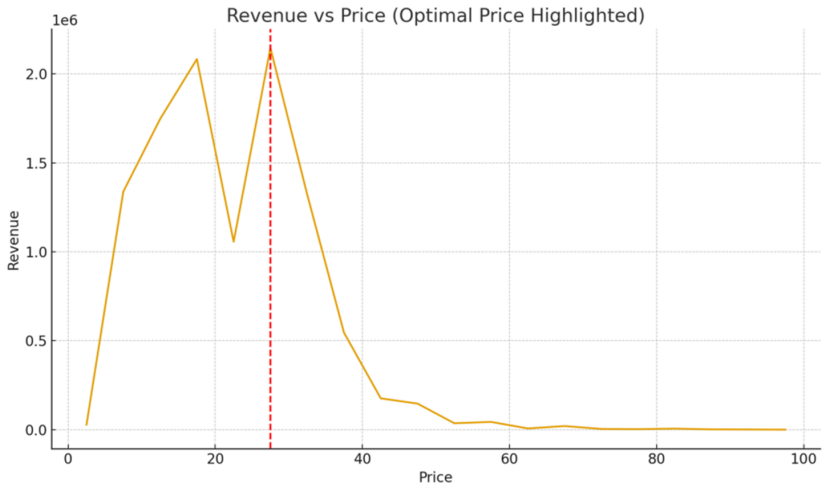
Price Sensitivity - Hour of Day



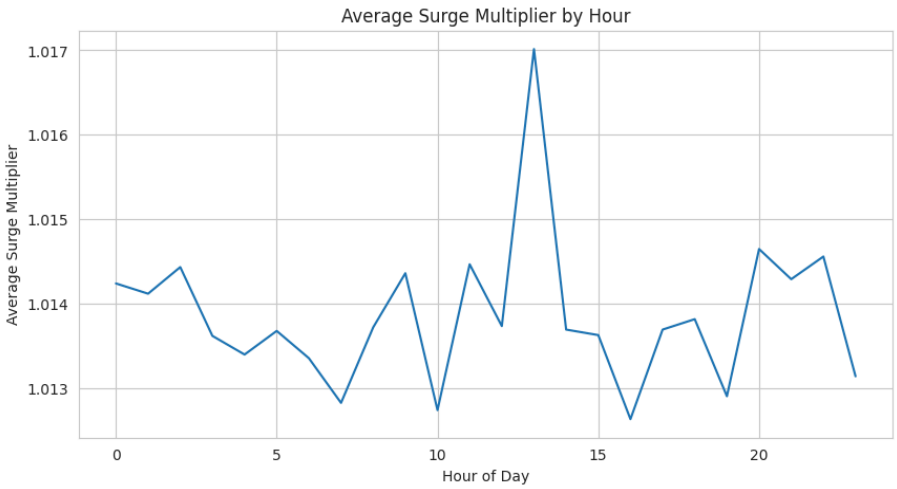
Price Elasticity - Ride Type



Optimal Price Analysis - Revenue Maximization



Surge Pricing - Average Surge Multiplier by Hour of day



Code Snippets

Random Forest Model

```
df_price = df[df["price"].notna()].copy()
print("Rows with non-missing price:", df_price.shape)

# 2) Defining features for this model
numeric_features_price = [
    col for col in numeric_features
    if col in df_price.columns and col != "price"
]

categorical_features_price = [
    col for col in categorical_features
    if col in df_price.columns and col not in ["short_summary", "long_summary"]
]

print("Numeric features used for price model:", numeric_features_price)
print("Categorical features used for price model:", categorical_features_price)

X_price = df_price[numeric_features_price + categorical_features_price]
y_price = df_price["price"]

print("X_price shape (full):", X_price.shape)

# 3) Sample to prevent runtime/memory issues
max_rows = 50000

if len(df_price) > max_rows:
    sample = df_price.sample(n=max_rows, random_state=42)
    X_price = sample[numeric_features_price + categorical_features_price]
    y_price = sample["price"]
    print("Using sampled data for training:", X_price.shape)
else:
    print("Using full data for training:", X_price.shape)
```

```
# 4) Train-test split
X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(
    X_price, y_price, test_size=0.2, random_state=42
)

# 5) Preprocessor: impute + scale numeric, impute + one-hot encode categoricals
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor_price = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features_price),
        ("cat", categorical_transformer, categorical_features_price)
    ]
)

# 6) Random Forest (reduced complexity for speed)
rf_regressor = RandomForestRegressor(
    n_estimators=100,
    max_depth=20,
    n_jobs=-1,
    random_state=42,
    verbose=1
)

price_model = Pipeline(steps=[
    ("preprocessor", preprocessor_price),
```

```python
# 7) Train the model
price_model.fit(X_train_p, y_train_p)

# 8) Evaluate model
y_pred_p = price_model.predict(X_test_p)

mae = mean_absolute_error(y_test_p, y_pred_p)

# Manual RMSE (older sklearn does not support squared=False)
mse = mean_squared_error(y_test_p, y_pred_p)
rmse = mse ** 0.5

r2 = r2_score(y_test_p, y_pred_p)

print("\n=== Random Forest Price Model Performance ===")
print(f"MAE  : {mae:.2f}")
print(f"RMSE : {rmse:.2f}")
print(f"R^2  : {r2:.3f}")
```

```
Rows with non-missing price: (637976, 39)
Numeric features used for price model: ['hour', 'day', 'month', 'distance', 'surge_multiplier', 'latitude', 'longitude', 'temperature', 'apparentTem
Categorical features used for price model: ['timezone', 'source', 'destination', 'cab_type', 'name', 'icon']
X_price shape (full): (637976, 36)
Using sampled data for training: (50000, 36)
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   44.5s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.6min finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.2s

=== Random Forest Price Model Performance ===
MAE  : 1.17
RMSE : 1.87
R^2  : 0.959
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.4s finished
```

Neural Network Model

```python
# MODEL 2: Neural Network for High Demand

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score,
    roc_auc_score,
    confusion_matrix,
    classification_report
)
from sklearn.utils.class_weight import compute_class_weight

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# 1) Keep only rows where surge_multiplier is known
df_cls = df[df["surge_multiplier"].notna()].copy()
print("Rows with non-missing surge_multiplier:", df_cls.shape)

# 2) Define binary target: high demand if surge_multiplier > 1
df_cls["high_demand"] = (df_cls["surge_multiplier"] > 1).astype(int)
print("\nHigh_demand class balance:")
print(df_cls["high_demand"].value_counts(normalize=True))

# 3) Define features for the NN model
#    numeric: all numeric_features except 'price' and 'surge_multiplier'
numeric_features_cls = [
    col for col in numeric_features
    if col in df_cls.columns and col not in ["price", "surge_multiplier"]
]
```

```
#    categorical: lean subset to keep dimension manageable
categorical_features_cls = [
    col for col in ["timezone", "source", "destination", "cab_type", "name", "icon"]
    if col in df_cls.columns
]

print("\nNN numeric features:", numeric_features_cls)
print("NN categorical features:", categorical_features_cls)

# 4) Build X and y
X_cls = df_cls[numeric_features_cls + categorical_features_cls].copy()
y_cls = df_cls["high_demand"].astype(int)

print("\nX_cls shape (before encoding):", X_cls.shape)

# 5) IMPUTE MISSING VALUES BEFORE ENCODING

# Impute numeric columns with median
for col in numeric_features_cls:
    median_val = X_cls[col].median()
    X_cls[col] = X_cls[col].fillna(median_val)

# Impute categorical columns with a placeholder
for col in categorical_features_cls:
    X_cls[col] = X_cls[col].fillna("Missing")

# 6) One-hot encode categorical features
X_cls_encoded = pd.get_dummies(
    X_cls,
    columns=categorical_features_cls,
    drop_first=True
)

print("X_cls_encoded shape (after one-hot):", X_cls_encoded.shape)

# 7) Train-test split (stratified to preserve class imbalance)
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(
    X_cls_encoded,
    y_cls,
    test_size=0.2,
    random_state=42,
    stratify=y_cls
)

print("\nTrain shape:", X_train_c.shape)
print("Test shape :", X_test_c.shape)

# 8) Scale features for neural network
scaler = StandardScaler()
X_train_c_scaled = scaler.fit_transform(X_train_c)
X_test_c_scaled = scaler.transform(X_test_c)

# 9) Compute class weights to handle imbalance
classes = np.unique(y_train_c)
class_weights_array = compute_class_weight(
    class_weight='balanced',
    classes=classes,
    y=y_train_c
)
class_weights = {int(c): w for c, w in zip(classes, class_weights_array)}
print("\nClass weights used:", class_weights)

# 10) Build the neural network
input_dim = X_train_c_scaled.shape[1]
print("\nInput dimension for NN:", input_dim)

nn_model = keras.Sequential([
    layers.Input(shape=(input_dim,)),
    layers.Dense(64, activation="relu"),
```

```python
nn_model = keras.Sequential([
    layers.Input(shape=(input_dim,)),
    layers.Dense(64, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")  # binary classification
])

nn_model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

nn_model.summary()

# 11) Train the model WITH CLASS WEIGHTS
history = nn_model.fit(
    X_train_c_scaled,
    y_train_c,
    epochs=15,
    batch_size=256,
    validation_split=0.2,
    class_weight=class_weights,
    verbose=1
)
```

```
Rows with non-missing surge_multiplier: (693071, 39)

High_demand class balance:
high_demand
0    0.969736
1    0.030264
Name: proportion, dtype: float64

NN numeric features: ['hour', 'day', 'month', 'distance', 'latitude', 'longitude', 'temperature', 'apparentTemperature', 'precipIntensity', 'precip
NN categorical features: ['timezone', 'source', 'destination', 'cab_type', 'name', 'icon']

X_cls shape (before encoding): (693071, 35)
X_cls_encoded shape (after one-hot): (693071, 70)

Train shape: (554456, 70)
Test shape : (138615, 70)

Class weights used: {0: np.float64(0.5156041928596404), 1: np.float64(16.52133492252682)}

Input dimension for NN: 70
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 64)	4,544
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 1)	33

```
Total params: 6,657 (26.00 KB)
Trainable params: 6,657 (26.00 KB)
Non-trainable params: 0 (0.00 B)

Epoch 1/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.7082 - loss: 0.4172 - val_accuracy: 0.7118 - val_loss: 0.4547
Epoch 2/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7233 - loss: 0.3592 - val_accuracy: 0.7257 - val_loss: 0.4206
Epoch 3/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7270 - loss: 0.3565 - val_accuracy: 0.7160 - val_loss: 0.4798
Epoch 4/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.7260 - loss: 0.3540 - val_accuracy: 0.7168 - val_loss: 0.4358
Epoch 5/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7315 - loss: 0.3485 - val_accuracy: 0.7202 - val_loss: 0.4507
Epoch 6/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7293 - loss: 0.3491 - val_accuracy: 0.7221 - val_loss: 0.4306
Epoch 7/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.7349 - loss: 0.3450 - val_accuracy: 0.7297 - val_loss: 0.4321
Epoch 8/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.7376 - loss: 0.3423 - val_accuracy: 0.7345 - val_loss: 0.4110
Epoch 9/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7394 - loss: 0.3393 - val_accuracy: 0.7344 - val_loss: 0.4485
Epoch 10/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.7440 - loss: 0.3360 - val_accuracy: 0.7328 - val_loss: 0.4300
Epoch 11/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.7457 - loss: 0.3333 - val_accuracy: 0.7340 - val_loss: 0.4211
Epoch 12/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7460 - loss: 0.3341 - val_accuracy: 0.7493 - val_loss: 0.3830
Epoch 13/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7489 - loss: 0.3307 - val_accuracy: 0.7497 - val_loss: 0.3924
Epoch 14/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7532 - loss: 0.3255 - val_accuracy: 0.7496 - val_loss: 0.3817
Epoch 15/15
1733/1733 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7521 - loss: 0.3269 - val_accuracy: 0.7448 - val_loss: 0.4110
```

LLM Documentation (ChatGPT Helped Summarize its Contribution to the Project)

1. Assistance with Data Preprocessing and Error Resolution

Large Language Models were used to support the data cleaning and preprocessing workflow by helping identify the causes of technical issues and suggesting appropriate solutions. Examples include diagnosing parser errors when importing the dataset, explaining why certain columns required imputation, and recommending the proper handling of categorical variables through one-hot encoding. The LLM also clarified why certain scaling methods were necessary for neural network inputs and guided corrections for issues such as NaN values appearing in model predictions.

2. Support in Model Design Decisions

LLMs provided conceptual guidance on the selection and structure of the machine learning models used in the project. This included explaining the advantages of random forest regression for capturing nonlinear pricing patterns, clarifying why linear models may be limited for surge forecasting, and recommending class-weighting techniques to address the imbalance in the high-demand classification problem. The LLM also helped justify architectural choices for the neural network, such as activation functions, layer sizes, and the use of cross-entropy loss for binary classification.

3. Interpretation of Model Outputs

The project used LLMs to refine the interpretation of statistical and machine learning results. For the random forest model, the LLM helped explain the meaning of performance metrics such as MAE, RMSE, and R^2, and provided insight into how feature importance scores translate to real pricing behavior. For the neural network classifier, the LLM assisted in understanding precision-recall tradeoffs, threshold tuning, and the implications of high recall but low precision in surge detection from a revenue management perspective.

4. Guidance on Scenario Analysis and Simulation Design

The LLM supported the development of scenario-based pricing simulations by suggesting appropriate methods for constructing baseline conditions, identifying relevant variables to vary systematically, and interpreting the resulting price curves. This assistance helped strengthen the connection between predictive modeling and revenue management strategy by ensuring the simulations reflected realistic operational choices.

5. Clarification of Revenue Management Concepts

Throughout the project, the LLM provided explanations linking model outputs to revenue management principles, such as willingness-to-pay, dynamic pricing behavior, demand classification, and threshold-based surge activation strategies. These clarifications helped situate the modeling results within the theoretical framework of the course and assisted in translating technical findings into business implications.