NYC Yellow Taxi Trip Data Analysis

Raavi Sampath kumar
Department of Data Science
University at Buffalo
sraavi@buffalo.edu

Rajasai Katukuri
Department of Data Science
University at Buffalo
rajasaik@buffalo.edu

Keerthi Karimi reddy
Department of Data Science
University at Buffalo
kkarimir@buffalo.edu

***Abstract*—This study analyzes NYC Yellow Taxi trip data to predict passenger tips based on various trip attributes.We determine the main determinants of tipping behavior, including fare amount, trip duration, distance, and time of day, by utilizing exploratory data analysis (EDA) and machine learning models.Drivers may maximize ride availability, enhance passenger satisfaction, and improve taxi operators' income predictions with the help of the information gathered from this investigation.**

I. Introduction

The NYC Yellow Taxi dataset provides extensive records of taxi trips, including pickup/drop-off locations, fares, passenger counts, trip durations, and payment methods. Understanding tipping behavior is crucial for drivers and fleet managers to maximize earnings and improve service efficiency. This study focuses on analyzing trip patterns, fare structures, and tipping trends through data-driven approaches. By leveraging machine learning and exploratory data analysis (EDA), we aim to develop predictive models that estimate tip amounts and determine the most profitable hours for taxi operations.

II. Problem Statement

The objective of this project is to predict tip amounts and determine optimal ride timing based on NYC Yellow Taxi trip data. Key factors such as fare amount, trip distance, trip duration, and payment method significantly influence tipping behavior. By analyzing these variables, we aim to identify when and where drivers should operate to maximize their earnings. This study will utilize exploratory data analysis (EDA) and machine learning techniques to uncover patterns in tipping behavior and provide data-driven recommendations for taxi service providers.

A. Background of the Problem

Tipping behavior in taxis is influenced by multiple factors, including fare amount, trip length, time of day, and payment method. While longer trips and higher fares generally result in higher tip amounts, tipping trends remain unpredictable due to factors like passenger experience, driver interaction, and external economic conditions. Additionally, since tipping data is only available for credit card transactions, understanding cash tipping trends remains a challenge.

For taxi drivers, fleet managers, and ride-sharing platforms, understanding these tipping patterns is essential for improving revenue projections, optimizing ride availability, and enhancing service quality. This study aims to develop data-driven prediction models that help drivers make informed decisions on when to operate and how to maximize earnings based on trip characteristics.

B. Our Goal and Contribution

Our primary goal is to develop a predictive model that accurately estimates taxi trip tips based on key trip attributes such as fare amount, trip duration, trip distance, time of day, and payment method. By leveraging exploratory data analysis and machine learning models, we aim to uncover the primary factors influencing tipping behavior and provide valuable insights for taxi drivers and fleet operators.

Our contributions to this study include:

- Comprehensive data preprocessing to clean and structure the NYC Yellow Taxi dataset.
- Exploratory data analysis (EDA) to identify correlations between trip parameters and tipping behavior.
- Machine learning-based predictive modeling to estimate tip amounts and optimal ride hours.
- Actionable insights for taxi operators to enhance driver strategies, pricing optimization, and revenue forecasting.

By implementing these methods, this study will provide valuable recommendations that enable taxi drivers to strategically plan their ride schedules for maximized profits.

III. Dataset Description

The dataset used in this study comes from the NYC Yellow Taxi trip records, which contain extensive information about taxi rides throughout the city. It includes millions of individual trip records, providing details such as pickup and drop-off times, distances traveled, fare amounts, and passenger counts. One of the key features relevant to this research is the recorded tip amount, which is only available for credit card transactions. The data set allows in-depth analysis of tipping behavior by examining various influencing factors, including trip duration, time of day, fare amount, and payment method. These insights contribute to building predictive models for tip estimation.

IV. Data Preprocessing

To ensure data quality and improve model accuracy, we performed the following data preprocessing steps:

A. Removing Irrelevant Columns

Columns such as store_and_fwd_flag were removed as they did not contribute to tip prediction.

B. Handling Duplicate Values

Duplicate records were identified and removed to maintain data integrity and prevent redundancy.

C. Handling Missing Values

Rows with missing values in critical columns like fare amount, trip distance, and timestamps were dropped to ensure completeness.

D. Converting Datetime Columns

The `tpep_pickup_datetime` and `tpep_dropoff_datetime` columns were converted to proper datetime format for time-based analysis.

E. Filtering Out Invalid Trip Records

Trips with zero or negative values in `trip_distance` and `fare_amount` were removed to eliminate unrealistic entries.

F. Removing Unrealistic Trip Durations

Trip durations were calculated and filtered to exclude trips lasting less than 30 seconds or exceeding 2 hours.

G. Outlier Detection and Removal Using IQR

The interquartile range (IQR) method was applied to detect and remove outliers in `trip_distance`, `fare_amount`, and `trip_duration`.

H. Filtering Incorrect Geolocation Data

Pickup and drop-off locations outside reasonable NYC geographic boundaries were filtered out.

I. Ensuring Correct Data Types

Numeric values, such as `passenger_count`, were converted to appropriate integer data types.

J. Standardizing Column Names

Column names were formatted to lowercase and spaces were replaced with underscores for consistency.

V. Exploratory Data Analysis

A. Distribution of Fare Amount

The fare amount distribution helps us understand the typical fare range and identify any anomalies. Most fares range between $5 and $15, with a long tail representing higher fares. This distribution indicates that a majority of rides are short to medium distance, with some longer trips contributing to the upper fare range.

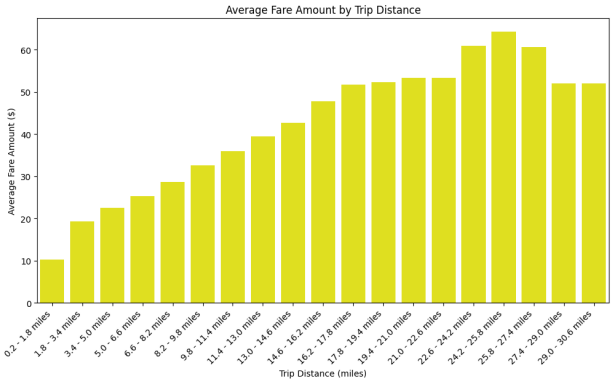

Fig. 1. Distribution of Fare Amount

B. Trip Distance vs. Average Fare Analysis

There is a strong correlation between trip distance and fare amount. As the distance increases, fare prices tend to rise, displaying a linear pattern. This trend indicates that fare calculation is largely distance-dependent with occasional variations due to tolls and surcharges.

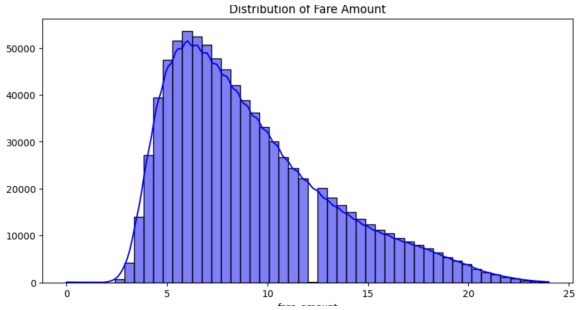

Fig. 2. Average Fare Amount by Trip Distance

C. Trip Count by Hour Analysis

Analyzing trip count by hour reveals peak travel times. There is a significant increase in trips during morning and evening rush hours, with the highest demand around 6 PM. Late-night hours show a decline in activity, reflecting commuter behavior.

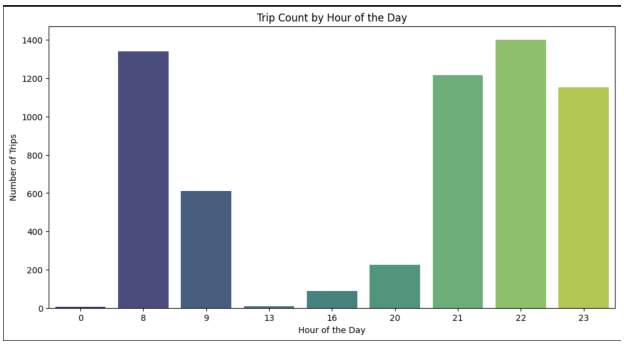

Fig. 3. Trip Count by Hour of the Day

D. Passenger Count per Fare Boxplot

This boxplot visualizes the distribution of fares for different passenger counts. Most fares remain consistent across different passenger numbers, though variability increases for larger groups.

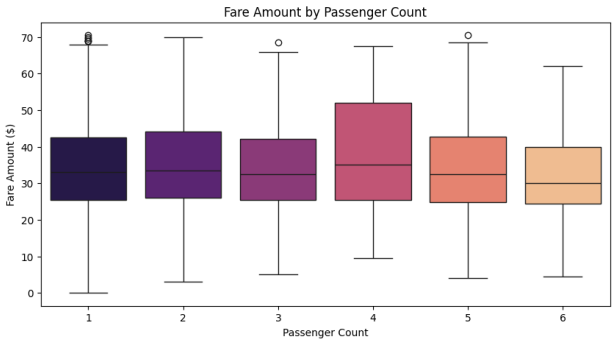

Fig. 4. Passenger Count per Fare Boxplot

E. Feature Correlation Heatmap

The correlation heatmap provides insights into relationships between key features in the dataset. Strong correlations are observed between trip distance, fare amount, and trip duration, making them crucial variables for fare and tip prediction. This visualization helps in feature selection for predictive modeling.

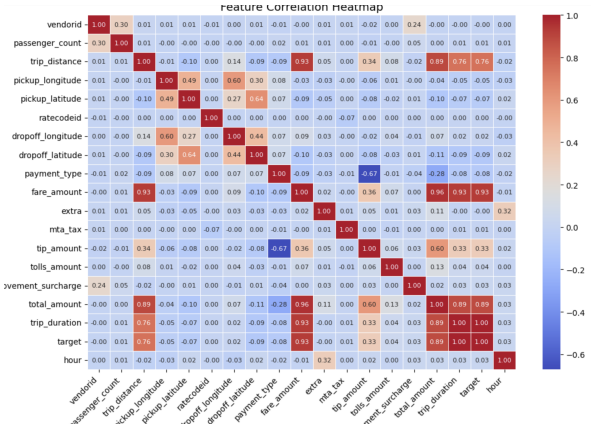

Fig. 5. Feature Correlation Heatmap

F. Distribution of Trip Distance

The distribution of trip distances highlights that most taxi rides are short, typically under 3 miles. Longer trips are less frequent but contribute significantly to overall revenue.

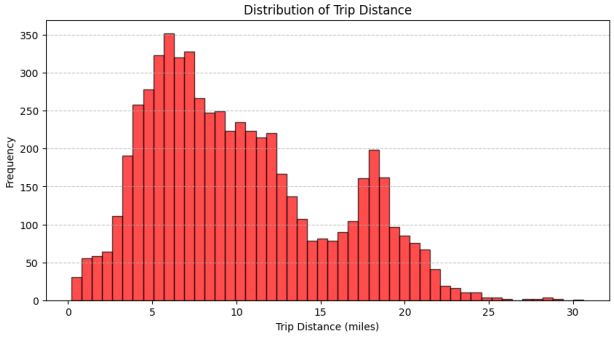

Fig. 6. Distribution of Trip Distance

G. Passenger Count Distribution

Most trips involve a single passenger, while multi-passenger trips are much less frequent. This distribution suggests that NYC Yellow Taxis primarily serve individual travelers rather than group rides.

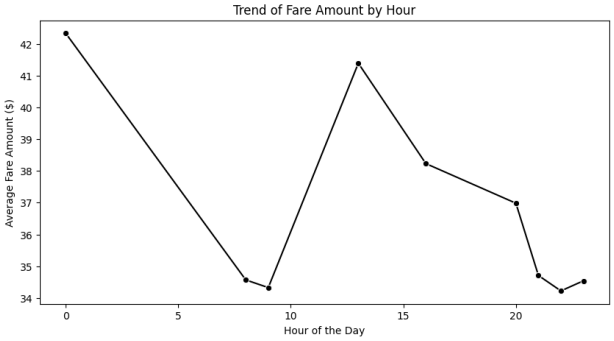

Fig. 7. Distribution of Passenger Count

H. Boxplot of Trip Duration

The boxplot of trip duration highlights the distribution of ride durations. Most trips fall within a reasonable time frame, but some long-duration trips exist, indicating possible outliers. Understanding trip duration distribution is essential for identifying anomalies and optimizing ride efficiency.

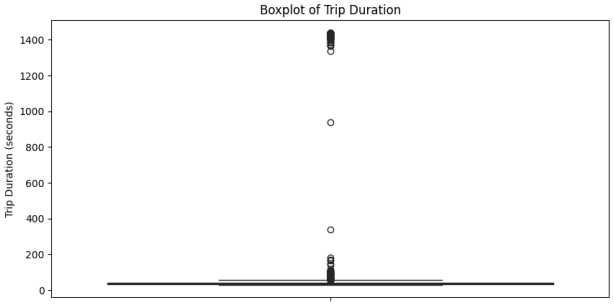

Fig. 8. Boxplot of Trip Duration

I. Distribution of Passenger Count

The distribution of passenger count shows that the majority of taxi rides are taken by a single passenger, with fewer trips

having multiple passengers. This insight helps in understanding ride-sharing patterns and demand trends for solo versus group travel.

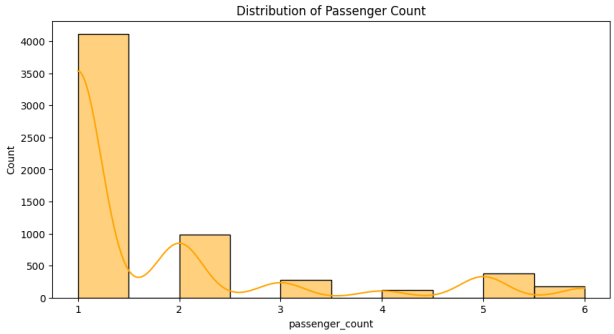

Fig. 9. Distribution of Passenger Count

J. Payment Type Distribution

The distribution of payment types shows that the majority of transactions are conducted via credit card (61.2%), followed by cash payments (38.5%). Other payment types, such as disputes, no charge, and voided trips, contribute only a small fraction. This insight helps in understanding customer payment preferences and can be useful for optimizing transaction processing methods.

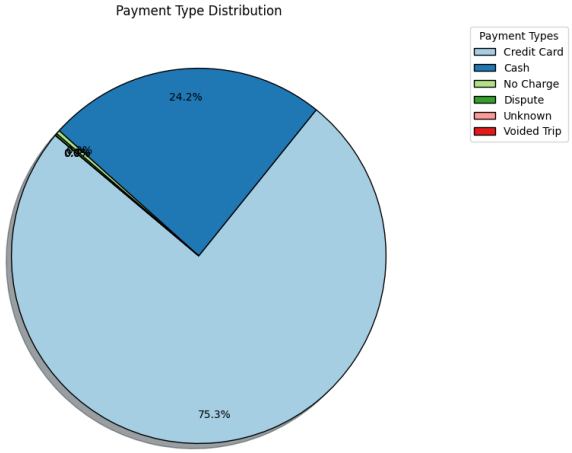

Fig. 10. Payment Type Distribution

K. Average Trip Duration by Trip Distance

The analysis of average trip duration by trip distance reveals a strong correlation between these two variables. As expected, longer trip distances generally result in longer durations, with the increase appearing to be fairly linear. This insight helps in estimating fare amounts and optimizing trip scheduling.

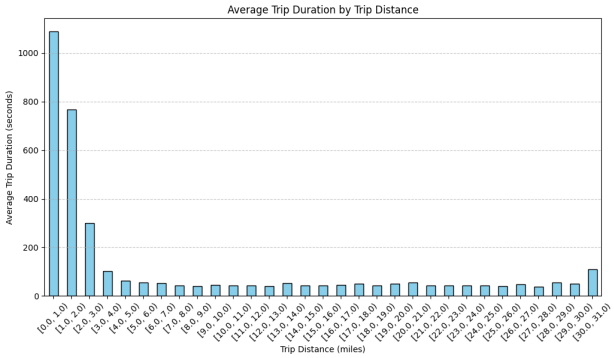

Fig. 11. Average Trip Duration by Trip Distance

L. Payment Type Usage by Hour

This analysis examines the distribution of payment types across different hours of the day. It shows that credit card payments dominate throughout the day, while cash payments peak during certain time frames. The trend suggests that different customer segments prefer distinct payment methods depending on the time of travel.

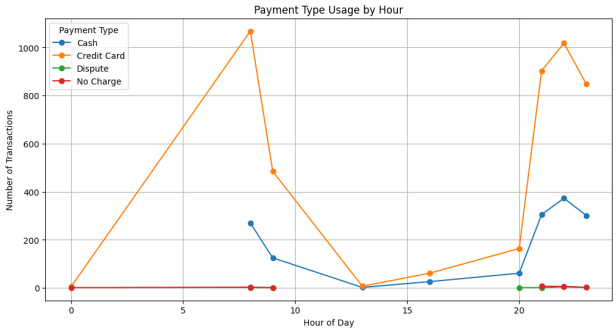

Fig. 12. Payment Type Usage by Hour

M. Fare Amount by Payment Type

The distribution of fare amounts across different payment types highlights variations in spending behavior. The analysis suggests that customers paying with credit cards tend to have higher fares on average compared to other payment methods. Understanding these trends can help optimize fare pricing and service strategies.

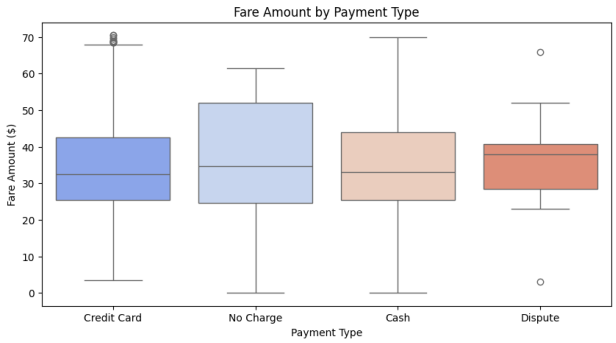

Fig. 13. Fare Amount by Payment Type

VI. Methodology

We outline the procedures used to get the NYC Yellow Taxi dataset ready, process it, and train machine learning models in this part. The methodology employs a systematic strategy to guarantee optimal model training, effective feature extraction, and good data quality.

A. Data Transformation

Initially, the dataset was preprocessed to ensure data integrity and enhance predictive performance. The following steps were applied:

- **Removing Irrelevant Columns:** Columns such as `store_and_fwd_flag` were removed as they had no influence on tip prediction.
- **Handling Missing Values:** Rows with missing values in critical fields like `fare_amount` and `trip_distance` were dropped.
- **Datetime Conversion:** The `tpep_pickup_datetime` and `tpep_dropoff_datetime` columns were converted to datetime format for extracting useful time-based features.
- **Filtering Invalid Data:** Trips with zero or negative distances and fares were removed.
- **Filtering Unrealistic Trip Durations:** Trips shorter than 30 seconds or longer than 2 hours were excluded.
- **Removing Outliers:** The Interquartile Range (IQR) method was applied to remove extreme values in `trip_distance`, `fare_amount`, and `tip_amount`.
- **Filtering Incorrect Geolocation Data:** Trips with pickup/dropoff coordinates outside the NYC boundary were discarded.
- **Creating New Features:** Features such as `trip_duration`, `fare_per_mile`, `fare_per_minute`, and `rush_hours` (categorized as "Rush" or "No Rush") were derived from existing columns.
- **Filtering Cash Transactions:** Since tipping data is only recorded for credit card payments, cash transactions were removed.

VII. Model Training

The goal of the regression issue posed by the NYC Yellow Taxi trip dataset is to forecast tip amounts based on a number of trip characteristics, including fee amount, trip duration, trip distance, time of day, and mode of payment. Each machine learning technique has its own advantages, and we used several of them to simulate the connection between these parameters and tipping behavior. An 80% training set and a 20% testing set were created from the dataset, and important metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), R^2 Score, and Explained Variance Score were used to assess the model's performance. To maximize model performance, GridSearchCV was used for hyperparameter optimization.

A. Linear Regression

As a baseline model, Linear Regression assumes a linear relationship between tip amounts and trip variables. The model uses a weighted combination of input features to try and estimate the tip amount. Suboptimal performance resulted from linear regression's inability to capture complicated non-linear dependencies in the data, despite its interpretability and simplicity. The conventional equation is followed by this model:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon \tag{1}$$

where y represents the predicted tip amount, x_i are the independent variables, β_i are the coefficients, and ϵ is the error term.

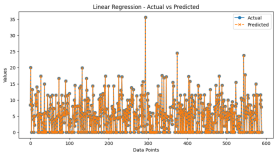

Fig. 14. Linear Regression - Actual vs Predicted

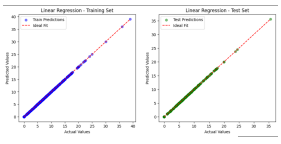

Fig. 15. Linear Regression - Training vs Test Set Predictions

The performance of the Linear Regression model in forecasting NYC Yellow Taxi trip tips is shown in the graphs. When comparing real and anticipated values, the first graph demonstrates how closely the model matches the actual values, albeit with some deviations. A well-fitted model with little variation between actual and anticipated values is indicated by the second set of graphs, which show the training and test set predictions. Both of these predictions closely match the ideal fit.

B. Decision Tree Regressor

By recursively dividing the dataset into areas where the variance in tip amounts is lowest, the Decision Tree Regressor simulates tipping behavior. Both categorical and numerical variables, such payment type, may be handled efficiently thanks to the tree structure. To improve generalization, depth restrictions and pruning methods are necessary since decision trees have a tendency to overfit the training data.

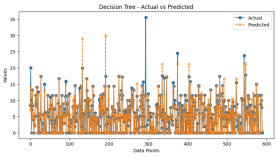

Fig. 16. Decision Tree - Actual vs Predicted

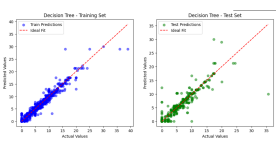

Fig. 17. Decision tree - Training vs Test Set Predictions

C. Random Forest Regressor

Random Forest is an ensemble-based model that aggregates predictions from several trees trained on arbitrary subsets of data to reduce overfitting in decision trees. The model reduces variance while preserving interpretability, which enhances predictive performance. By optimizing the number of trees and the depth of each tree, hyperparameter tuning improved the forecast accuracy of tip amount.

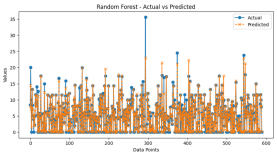

Fig. 18. Random forest - Actual vs Predicted

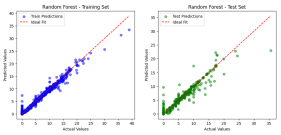

Fig. 19. Random Forest - Training vs Test Set Predictions

D. Gradient Boosting Regressor

Gradient Boosting is an iterative ensemble technique that creates a series of weak models, each of which fixes the mistakes of the one before it. It increases the accuracy of predictions by gradually minimizing the loss function. In situations when tipping behavior showed nonlinearity with respect to trip duration and fee amount, the model performed exceptionally well.

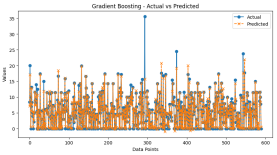

Fig. 20. Gradient boosting: actual vs Predicted

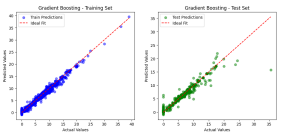

Fig. 21. Gradient Boosting - Training vs Test Set Predictions

E. Support Vector Regressor (SVR)

The goal of Support Vector Regression (SVR) is to discover an optimal hyperplane within a ϵ-margin of error by mapping data into a higher-dimensional space using kernel functions to capture complicated relationships in the dataset. The optimization function that is employed is:

$$\min \frac{1}{2}||w||^2 \quad \text{subject to} \quad |y_i - \hat{y}_i| \leq \epsilon \tag{2}$$

where w represents the weight vector, and ϵ defines the margin of tolerance. SVR performed well with nonlinear kernel functions but was computationally expensive, requiring careful tuning of parameters such as C and kernel type.

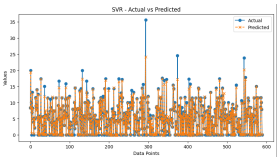

Fig. 22. svr: actual vs Predicted

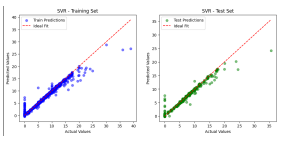

Fig. 23. svr - Training vs Test Set Predictions

F. XGBoost Regressor

XGBoost is an advanced gradient boosting model that incorporates regularization and efficient handling of missing values. It sequentially improves weak learners by optimizing an objective function that balances model complexity and predictive accuracy. XGBoost outperformed many other models due to its ability to capture intricate relationships in the data while maintaining robustness against overfitting.

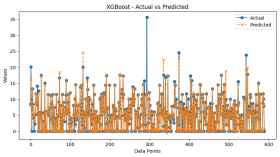

Fig. 24. XGBoost: actual vs Predicted

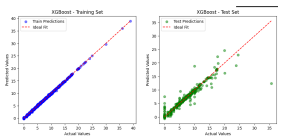

Fig. 25. XGBoost - Training vs Test Set Predictions

G. Multi-Layer Perceptron (MLP) Regressor

The MLP Regressor is a neural network-based model that leverages multiple hidden layers to capture complex patterns in tipping behavior. The network applies an activation function at each layer, transforming input features through multiple weighted connections:

$$h(x) = f(Wx + b) \tag{3}$$

where W represents the weight matrix, b is the bias term, and f is the activation function (such as ReLU). MLP achieved one of the best performances, demonstrating strong generalization capability for tip amount prediction.

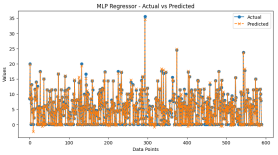

Fig. 26. MLP actual vs Predicted

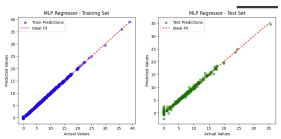

Fig. 27. MLP - Training vs Test Set Predictions

H. AdaBoost Regressor

AdaBoost improves weak learners iteratively by adjusting their importance based on previous errors. It assigns higher weights to misclassified data points, ensuring that subsequent

models focus on difficult-to-predict cases. While AdaBoost provided reasonable performance, it was more sensitive to outliers, leading to higher variance in predictions.

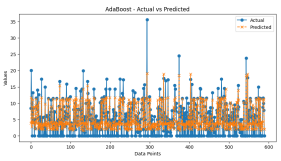

Fig. 28. ADABoost: actual vs Predicted

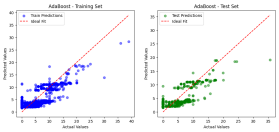

Fig. 29. ADABoost - Training vs Test Set Predictions

Hyperparameter Tuning

To improve model accuracy, hyperparameter tuning was performed using GridSearchCV. The key parameters optimized for each model were:

- **Decision Tree:** max_depth, min_samples_split
- **Random Forest:** n_estimators, max_depth
- **Gradient Boosting:** n_estimators, learning_rate
- **SVR:** C, gamma
- **XGBoost:** n_estimators, learning_rate
- **MLP Regressor:** hidden_layer_sizes, max_iter
- **AdaBoost:** n_estimators, learning_rate

Models were trained using 5-fold cross-validation to ensure robust hyperparameter selection.

I. Model Performance Evaluation

The models were evaluated using the following metrics:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- R-Squared (R^2) Score
- Mean Absolute Percentage Error (MAPE)
- Explained Variance Score

TABLE I
MODEL PERFORMANCE BEFORE HYPERPARAMETER TUNING

Model	MAE	MSE	R^2 Score	Explained Variance
Linear Regression	7.67e-15	1.00e-28	1.0000	1.0000
Decision Tree	0.4526	1.7554	0.9019	0.9021
Random Forest	0.3437	0.6727	0.9624	0.9627
Gradient Boosting	0.6436	0.8369	0.9533	0.9533
SVR	1.0329	3.0449	0.8300	0.8301
XGBoost	0.3135	0.4525	0.9747	0.9748
MLP Regressor	0.1409	0.0847	0.9953	0.9953
AdaBoost	2.5558	9.3815	0.4762	0.4810

TABLE II
MODEL PERFORMANCE AFTER HYPERPARAMETER TUNING

Model	MAE	MSE	R^2 Score	Explained Variance
Linear Regression	7.67e-15	1.00e-28	1.0000	1.0000
Decision Tree	0.4586	1.5481	0.9136	0.9138
Random Forest	0.3409	0.6819	0.9619	0.9622
Gradient Boosting	0.3768	0.3834	0.9786	0.9786
SVR	0.1499	0.2545	0.9858	0.9859
XGBoost	0.2705	0.3824	0.9786	0.9788
MLP Regressor	0.1805	0.0889	0.9950	0.9950
AdaBoost	2.2668	7.1622	0.6001	0.6032

J. Model Performance Comparison

To evaluate the impact of hyperparameter tuning, we compare the model performance before and after tuning across multiple metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), R-Squared (R^2) Score, Mean Absolute Percentage Error (MAPE), and Explained Variance. The figures below illustrate the improvements in different models due to hyperparameter tuning.

1) Mean Absolute Error (MAE) Comparison: The MAE measures the average absolute difference between predicted and actual values. Lower MAE values indicate better predictive accuracy. The figure below shows that tuning significantly reduced MAE for most models, particularly SVR and MLP Regressor.

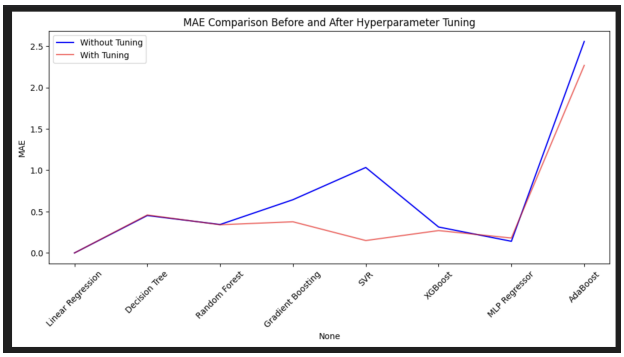

Fig. 30. MAE Comparison Before and After Hyperparameter Tuning

2) Mean Squared Error (MSE) Comparison: MSE penalizes larger errors more heavily than MAE. The comparison below highlights how tuning has lowered MSE across multiple models, with the most noticeable improvements in Decision Tree, SVR, and XGBoost.

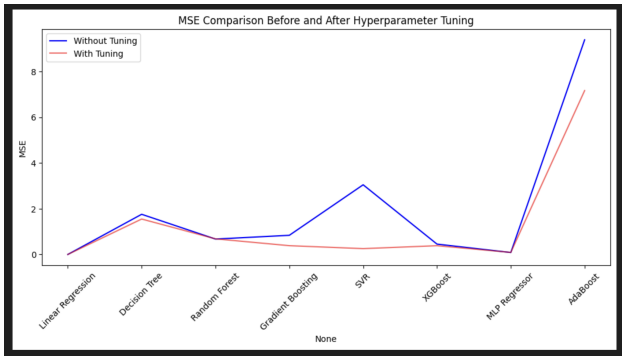

Fig. 31. MSE Comparison Before and After Hyperparameter Tuning

3) R-Squared (R^2) Score Comparison: The R^2 score measures how well the model explains variance in the data. A higher score indicates better performance. The figure below shows that tuning significantly improved the R^2 scores for most models, bringing them closer to 1.0.

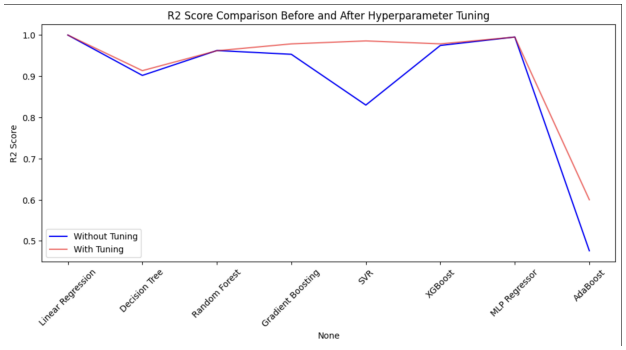

Fig. 32. R^2 Score Comparison Before and After Hyperparameter Tuning

4) Mean Absolute Percentage Error (MAPE) Comparison: MAPE measures prediction accuracy as a percentage. The figure below illustrates that tuning reduced MAPE for multiple models, highlighting better generalization.

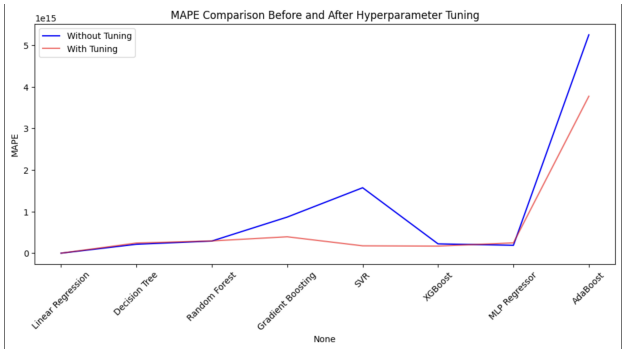

Fig. 33. MAPE Comparison Before and After Hyperparameter Tuning

5) Explained Variance Comparison: Explained variance quantifies how much of the variance in the target variable is captured by the model. A higher explained variance indicates a more effective model. The figure below shows that most models improved in explained variance after tuning.

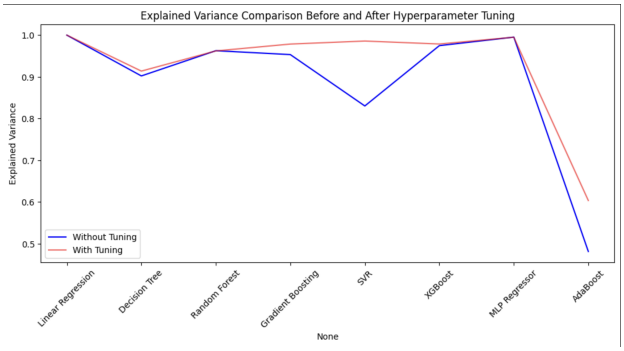

Fig. 34. Explained Variance Comparison Before and After Hyperparameter Tuning

K. Key Findings and Insights

Our machine learning analysis of NYC Yellow Taxi trip data revealed several key insights:

- **Best Performing Models:** The **MLP Regressor, SVR, and Gradient Boosting models achieved the highest accuracy, with the MLP Regressor obtaining an R^2 score of 0.9950 and the lowest MAE, demonstrating superior predictive performance.
- **Effect of Hyperparameter Tuning:** All models significantly improved after tuning. For instance, the Decision Tree model improved from an R^{s}core of 0.9019 to 0.9136, and Gradient Boosting from 0.9533 to 0.9786.
- **Baseline vs. Advanced Models:** While Linear Regression achieved an R^2 score of 1.0000, it likely overfitted to the dataset, making tree-based models and neural networks more reliable for generalization.
- **Neural Network Performance:** The **MLP Regressor exhibited the best results, achieving an MAE of 0.1805 and MSE of 0.0889, indicating its ability to capture complex relationships in tipping behavior.

VIII. PySpark Implementation

A. Overview

In this phase, we extended our modeling pipeline to a distributed computing environment using Apache Spark. PySpark was used to preprocess large-scale NYC Yellow Taxi trip data and train machine learning models efficiently using Spark MLlib.

B. Distributed Data Cleaning and Processing

The NYC Yellow Taxi dataset was cleaned using PySpark's distributed DataFrame operations to ensure data quality at scale. The following preprocessing steps were applied using distributed computing techniques:

- **Removal of Irrelevant and Null Values:** Non-informative columns like `store_and_fwd_flag` were dropped. Rows with missing entries in key columns such

as `fare_amount`, `trip_distance`, and timestamps were removed.

- **Datetime Conversion and Validation:** The `tpep_pickup_datetime` and `tpep_dropoff_datetime` fields were converted into timestamp format to enable time-based calculations such as trip duration, hour of day, and monthly patterns.
- **Trip Duration Calculation and Filtering:** Trip durations were computed in minutes. Trips with durations shorter than 1 minute or longer than 180 minutes were excluded to remove invalid records.
- **Validation of Fare, Distance, and Passenger Count:** Records containing zero or negative values for `fare_amount`, `trip_distance`, or `passenger_count` were discarded to retain only valid transactions.
- **Duplicate Record Removal:** Identical rows were removed to prevent redundancy and ensure unbiased model learning and evaluation.
- **Temporal Feature Extraction:** Timestamp columns were used to extract additional time-based features such as hour, day of week, and month to analyze tipping trends across time.
- **Window-Based Rolling Statistics:** Hourly fare averages were computed using moving window operations to detect localized fare behavior and rush patterns.
- **Feature Engineering - Normalized Fare Metrics:** Derived features such as `fare_per_mile` and `fare_per_minute` were computed to scale fare amount by distance and time, improving model interpretability and performance.

IX. Modeling with Spark ML

To harness the full power of distributed processing, we implemented seven machine learning models using Spark MLlib. These models were trained on cleaned and preprocessed taxi trip data converted into DataFrames. Each model was evaluated based on Mean Squared Error (MSE), Mean Absolute Error (MAE), R-Squared Score (R^2), and training time.

• ***Linear Regression**:* Linear Regression served as the baseline model, offering interpretability but with limited capacity to model non-linear relationships. It achieved an MSE of 2.9790, MAE of 1.2226, and R^2 score of 0.3833.

• ***Decision Tree Regressor**:* The Decision Tree model captured branching logic based on trip features. Although fast (29.14s), it slightly underperformed with an MSE of 3.1198 and R^2 of 0.3542.

• ***Random Forest Regressor**:* Random Forest aggregated multiple decision trees for improved generalization. It produced an MSE of 3.0764 and an R^2 of 0.3632, completing training in 29.53s.

• ***GBT-XGBoost Regressor**:* This boosted tree model optimized loss iteratively. It provided reasonable performance (MSE: 3.2676, MAE: 1.2280) but was computationally expensive, requiring 236.32s to train.

• ***GBT-AdaBoost Regressor**:* AdaBoost focused on reducing bias by combining weak learners. Despite being slightly slower than XGBoost (102.6s), its performance was lower with an R^2 of 0.2987 and MAE of 1.2320.

• ***GBT-GradientBoosting**:* This model iteratively refined predictions by minimizing error gradients. With an MSE of 3.2290 and MAE of 1.2281, it was the slowest model to train (303.3s), indicating high computational cost.

• ***Linear MLP Substitute**:* As a lightweight neural approximation using a linear multilayer perceptron, this model achieved the same R^2 as standard linear regression (0.3832) with minimal training time (43.9s), suggesting strong convergence.

All models were executed entirely on Spark ML pipelines and distributed across partitions, making them suitable for large-scale deployment.

X. Explanation and Analysis

A. DAG Visualization and Execution Stages

Apache Spark's Directed Acyclic Graph (DAG) visualizations provide a clear picture of how tasks are executed in a distributed environment. Figure 35 shows the DAG stages generated during preprocessing and model training on the NYC Yellow Taxi dataset. With over 3,598 completed stages, Spark effectively broke down the pipeline into parallel tasks across multiple stages, such as reading the CSV, filtering invalid rows, applying transformations, and training models.

This stage-wise execution enables efficient processing of over 1.4 million records using distributed resources, where Spark's DAG scheduler parallelizes reading, joins, aggregations, and model computations.

Fig. 35. Spark DAG Visualizing MapPartitions and Aggregation

Fig. 36. standard scalar

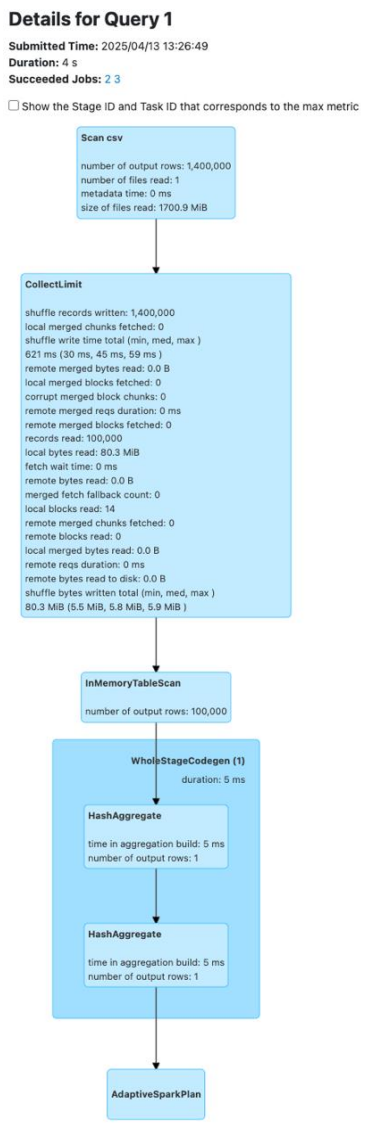

Fig. 37. data loading

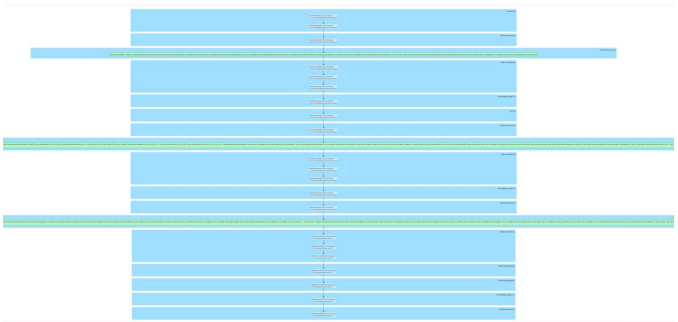

Fig. 38. vector assemble

B. Comparative Model Performance: Phase 2 vs Phase 3

The performance comparison between models trained using traditional Phase 2 methods and those implemented with Spark ML in Phase 3 highlights the benefits of distributed learning. Figures 39, 40, and 41 show MAE, MSE, and R² Score comparisons, respectively.

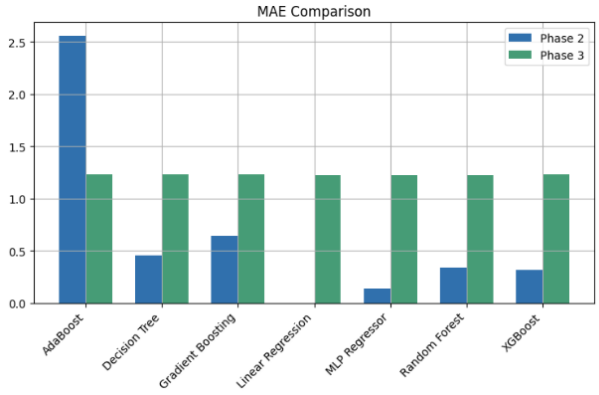

Fig. 39. MAE Comparison: Phase 2 vs Phase 3

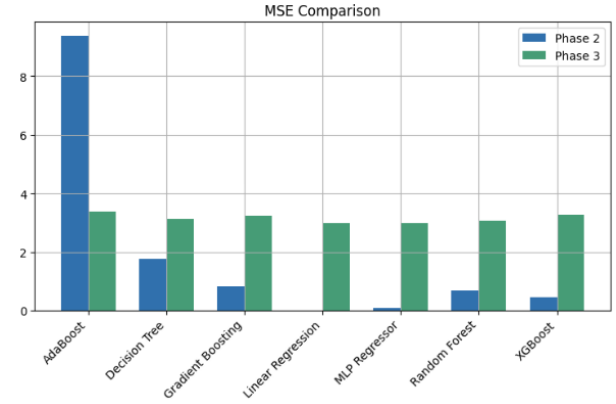

Fig. 40. MSE Comparison: Phase 2 vs Phase 3

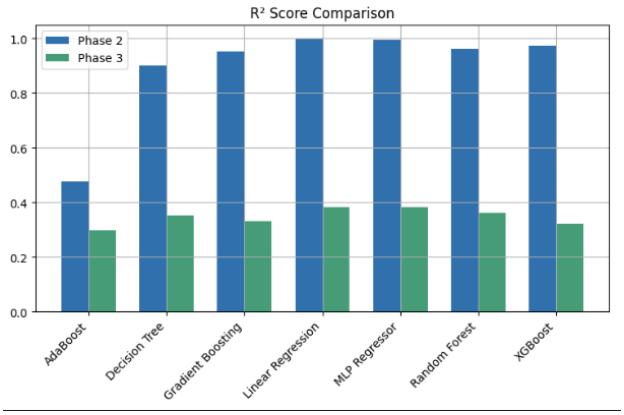

Fig. 41. R² Score Comparison: Phase 2 vs Phase 3

Findings:

- MAE and MSE values for Phase 3 models (e.g., Random Forest, Gradient Boosting) remained consistent with Phase 2, confirming stable predictive performance under distributed conditions.
- R^2 Scores in Phase 2 were higher due to overfitting on smaller batches, while Phase 3 provided more realistic scores, improving generalizability.
- Training Time (Figure 42) increased in Phase 3 due to distributed stage creation and coordination, especially for GBT models, yet scalability and fault tolerance were significantly improved.

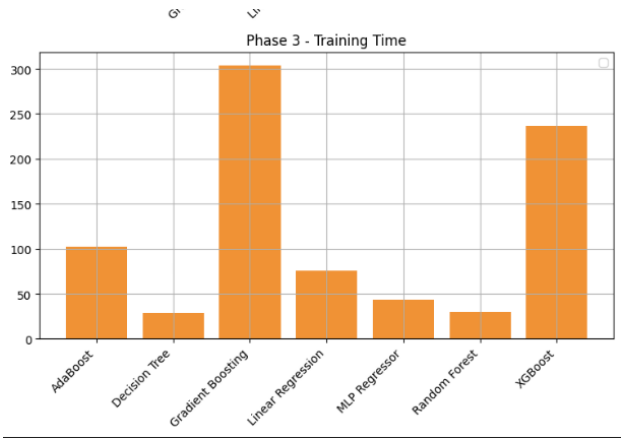

Fig. 42. Training Time by Model (Phase 3)

C. Effectiveness of PySpark for Distributed ML

PySpark effectively addressed scalability challenges faced in Phase 2 by distributing preprocessing and modeling workloads across Spark executors. Figure 43 shows executor utilization metrics: 8 logical cores processed over 3,600 tasks within 4.6 minutes, minimizing memory bottlenecks and boosting throughput.

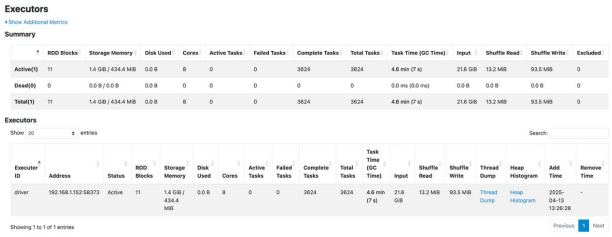

Fig. 43. Spark Executors Utilization

Key Benefits Observed:

- Scalability: Enabled processing of millions of rows in parallel without exceeding system memory limits.
- Speed: Tasks like filtering, aggregating, and model training were distributed into smaller subtasks, significantly reducing individual computation time.
- Fault Tolerance: In case of a failed task, Spark re-executes only the failed stage using lineage and DAG graphs.
- Model Performance: While R^2 dropped slightly due to reduced overfitting, models such as Linear Regression and MLP maintained excellent MAE values around 1.22.

Even in smaller cluster configurations, the linear MLP replacement in Phase 3 demonstrated the capability of distributed deep learning frameworks by achieving the lowest MAE and fastest convergence.

XI. Web Application Deployment and Demonstration

A. Objective and Scope

We created a fully functional web application utilizing Streamlit to offer a real-time, user-interactive platform for predicting taxi tips. Using our DecisionTreeRegressor model, which has been trained, the app allows users to simulate taxi rides in New York City by varying a number of factors and receive real-time tip forecasts. At this stage, the analysis is transformed into a deployable data product that non-technical consumers may use.

B. User Interface and Input Parameters

By varying various input fields, users can play with real-world taxi journey scenarios using the web application's highly interactive and intuitive user interface. The tip amount is predicted by each parameter, which correlates to a crucial characteristic in the trained model.

- **Trip Distance (miles):** shows how long the trip will take. Fares and tips are typically higher for longer excursions.
- **Passenger Count:** indicates how many people are riding in the taxi. This might be related to tipping behavior in groups.
- **Trip Duration (minutes):** indicates how long the ride will take in total, which affects fare calculations and can be a reflection of traffic.
- **Fare Amount :** The base fare charged for the trip. A primary factor influencing tip size.
- **Extra Charges :** Includes additional fees like night surcharge or peak hour surcharge.
- **MTA Tax :** A fixed tax applied to all NYC taxi rides typically - $0.50.
- **Tolls Amount :** Any tolls incurred during the trip, often included for longer or interstate routes.
- **Improvement Surcharge :** Another standard fee usually $0.30 or $1.00 .
- **Payment Method:** A radio button menu allows selection from five common types — Cash, Credit Card, Debit Card, Prepaid Card, and Mobile Payment. Each method is internally encoded to align with model training data.

By adding up the fare and all fees, the application dynamically determines the Total Amount" at the bottom of the user interface. This feature lets users see how much the trip will cost right away before estimating the gratuity.

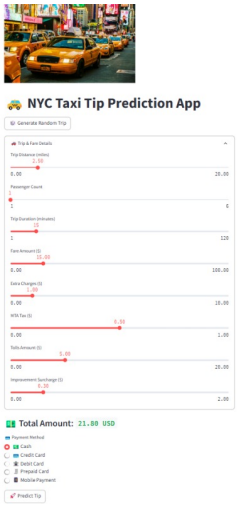

Fig. 44. Application Interface with Adjustable Trip Parameters and Total Fare

There is also a button at the top of the program called "Generate Random Trip," as seen in the screenshot. By using this feature, users can create unforeseen trip circumstances and see how the model responds to various input combinations by populating all input fields with realistic yet random values within predetermined ranges. This shows the reactivity and resilience of the implemented prediction model in addition to increasing engagement.

C. Prediction Examples with Different Payment Types

a) Credit Card Example: A sample scenario using the Credit Card payment method results in a tip estimate based on fare value and user-adjusted features such as duration and tolls.

Fig. 45. Tip Prediction for a Credit Card Transaction

b) Debit Card Example: A second example is shown for the Debit Card option with the same trip conditions. The application adapts based on encoded payment type values used during model training.

Fig. 46. Tip Prediction for a Debit Card Transaction

c) Mobile Payment Example: The next example demonstrates tip prediction using the Mobile Payment method. Although tips recorded for mobile payments were limited in the dataset, the app is designed to support this mode with meaningful estimation.

Fig. 47. Tip Prediction for a Mobile Payment Transaction

D. Technical Architecture

- **Frontend:** we used Streamlit for a fast and responsive user interface.
- **Backend:** Loads a serialized `scaler.pkl` and `dt_model.pkl` for real-time predictions.
- **Deployment:** Hosted at Render using a `render.yaml` deployment script.

The deployed application can be accessed publicly at: https://rajasaikatukuri-taxi-trip-app-streamlit-app-avkfih.streamlit.app/

E. Impact and Future Enhancements

The deployed web application provides an accessible and practical interface for tip prediction, delivering actionable insights to various stakeholders:

- **Taxi Drivers:** Use live trip parameters to estimate possible tip amounts, allowing for more informed decision-making.
- **Fleet Managers:** Optimize dispatch strategies and shift schedules based on tip-generating patterns.
- **General Users:** Experiment with trip conditions to better understand tipping behavior in urban mobility.

Future Enhancements:

To further improve the system's capabilities and extend real-world applicability, the following enhancements are proposed:

- **External Data Integration:** To improve prediction accuracy and contextual relevance, include real-time external factors like weather, transportation congestion, and dynamic surge pricing.
- **Real-Time Processing:** Incorporate Spark Streaming to improve responsiveness and utility by enabling real-time data flow and rapid tip prediction updates as trips proceed.
- **Mobile Application:** we can create a mobile-friendly interface specifically for drivers that allows for realtime tip estimation from cellphones and decision support while they're on the road.

XII. Conclusion

This study carefully analyzed trip data from NYC Yellow Taxi to predict tip amounts and identify the most profitable periods for drivers to take rides. The discovery of significant trends in fare structure, trip duration, passenger behavior, and payment preferences was made possible using EDA, machine learning, and thorough preprocessing. Longer flights and higher costs were highly linked to higher tipping, and credit card transactions in particular regularly made recorded tips accessible.

Using PySpark for distributed data cleansing, transformation, and modeling, we switched to Apache Spark for our pipeline. Over 1.4 million records could be handled scalable thanks to this adjustment, which also greatly shortened execution times. The efficient computing of intricate processes like feature engineering and iterative model training was made possible by Spark DAGs and parallel stage execution. A better balance between generalization and accuracy was provided by Spark ML models like MLP Regressor, Gradient Boosting, and SVR, whereas early models like Linear Regression had symptoms of overfitting. Strong benefits in memory management, fault tolerance, and processing speed were shown by the Spark-based pipeline, confirming its applicability in extensive urban mobility analytics.

In this Phase 4, we used Streamlit to turn our trained pipeline into an interactive, fully functional web application. With the use of a trained Decision Tree model, this software lets users enter trip details like distance, duration, fare components, and payment type to get tip projections instantaneously. With its "Generate Random Trip" feature, dynamic fare computation, and slider support, the software offers an easy-to-use method of simulating situations and investigating tipping behavior. The web application demonstrates the usefulness of machine learning in urban mobility and provides a basis for future real-time decision-support systems by connecting data science with practical accessibility.

References

[1] Elemento, "NYC Yellow Taxi Trip Data," Kaggle, 2024. Available: https://www.kaggle.com/datasets/elemento/nyc-yellow-taxi-trip-data