Fare Prediction Using NYC Taxi Data

1. Introduction and Objectives

Taxi fare prediction is a real-world use of machine learning in transportation systems, where it can help with demand forecasting, driver suggestions, and passenger pricing transparency by calculating a journey's cost based on time, distance, and other trip aspects. The goal of this project is to develop an end-to-end pipeline that predicts cab fares in New York City by utilizing distributed computing technology.

The objective is to use PySpark on Amazon EMR to import and process a sizable real-world dataset, then use Spark MLlib to create a regression model and store and query the converted data. This algorithm estimates fare amounts based on trip-level features including duration, passenger count, distance, and pickup time. This project also demonstrates a scalable cloud-based process for data science and machine learning at scale through the integration of several AWS services.

2. Dataset Overview

The dataset utilized in this study is a CSV file that is over 300 MB in size and contains over 3.2 million records from the yellow taxi trip data in New York City. It contains comprehensive trip-level data, including the time of pickup and drop-off, the distance traveled, the number of passengers, the fare, the mode of payment, and the tip amount. The dataset was staged in an Amazon S3 bucket and was taken from a public NYC taxi dataset.

This dataset was chosen because it reflects a real-world situation appropriate for regression analysis and satisfies the project's requirement of having more than one million rows. It is also perfect for obtaining significant features that are essential to fare estimation, such as journey duration and pickup hour, thanks to its extensive structure and timestamped data. AWS Glue was used to catalog the dataset for Athena querying after it had been staged and loaded into Apache Spark for distributed transformation.

3. Data Processing and Transformation

After staging the cleaned dataset in Amazon S3, it was read into a PySpark DataFrame using Spark running on an EMR cluster. The processing workflow focused on preparing the data for machine learning by engineering new features and filtering out invalid records.

First, a new column *trip_duration* was computed by calculating the difference between pickup and drop-off timestamps in minutes. An additional feature, pickup_hour, was extracted from the pickup timestamp to capture time-of-day effects. Records with non-positive fare amounts or trip durations were filtered out to ensure data quality.

The transformed dataset included the following selected features:

- *trip_distance*
- *trip_duration*
- *passenger_count*
- *pickup_hour*
- *fare_amount (target variable)*

After processing, the data was stored in Parquet format for effective querying back to S3. The output path was then subjected to a Glue crawler, which produced a matching table in the AWS Glue Data Catalog. This allowed the ability to query the data in Athena for further analysis and validation.

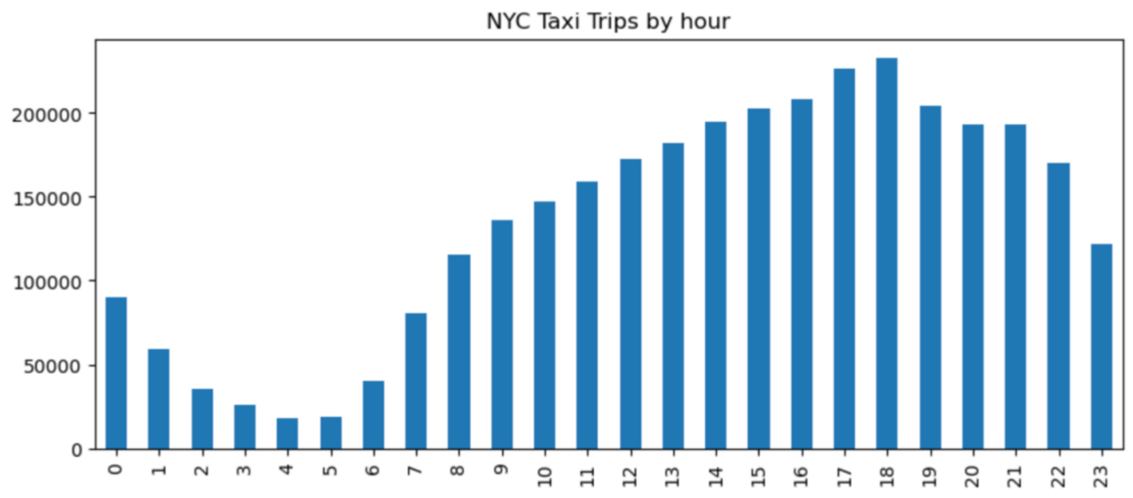
4. Exploratory Data Analysis

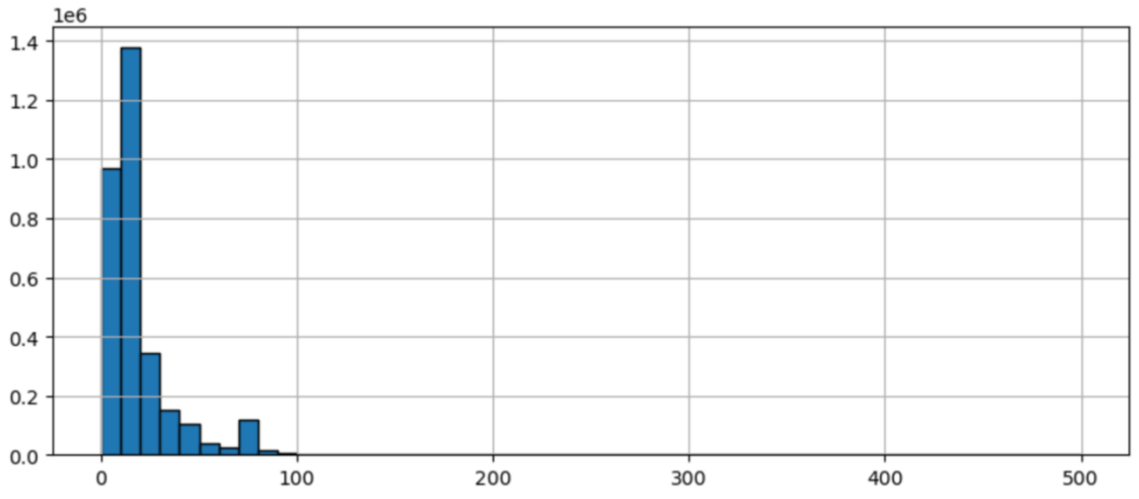
To better understand the distribution and characteristics of the data, exploratory analysis was performed using AWS Athena and PySpark. The focus was on identifying trends, patterns, and potential outliers that could impact model performance.

Key Findings of EDA

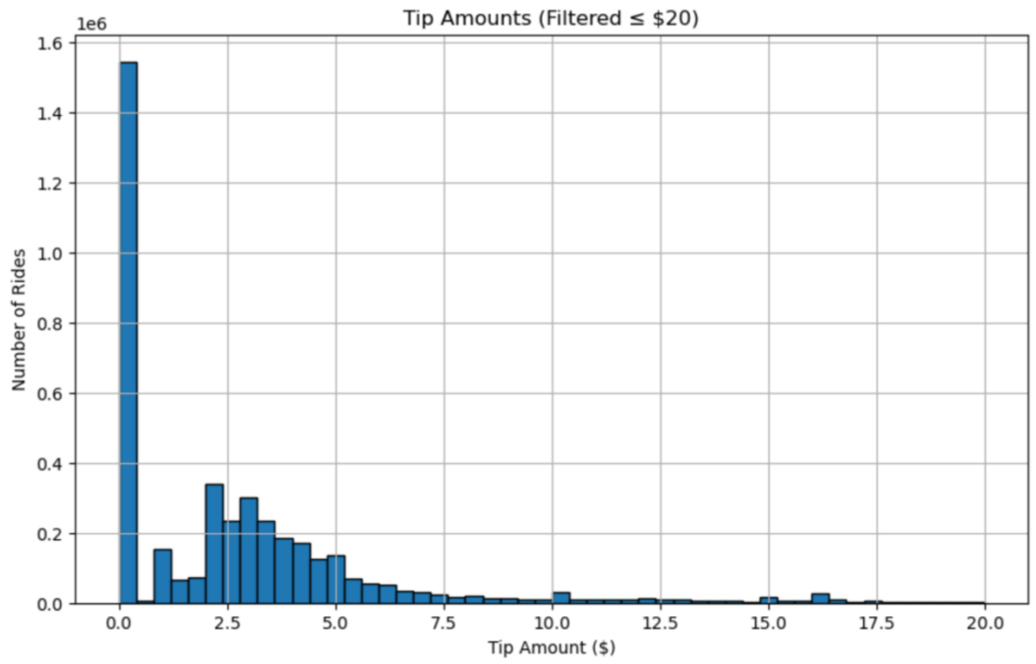
- Fare Amount Distribution: The majority of fares fell within the $5–$20 range, but a lengthy tail of higher costs indicated possible outliers or lengthier journeys.
- Trip Distance vs. Fare: A positive association was found, confirming *trip_distance* as a crucial predictive factor. Longer distances were typically associated with higher costs.
- Trip Duration and Pickup Hour: Temporal trends in fare pricing and trip characteristics were illustrated by calculating *trip_duration* and pickup_hour.
- Distribution of Passengers: Most trips carried one or two people, while very few carried more than four.
- Outliers: During preprocessing, a number of entries with zero or negative fare or duration data were eliminated.

Basic aggregations and data quality validation were done using Athena queries. Trends in taxi demand and behavior can be highlighted with additional visualizations, such as hourly ride volume and fare and distance histograms.





Fare Amount Distribution



5. Machine Learning Model Development

The machine learning phase's objective was to create a regression model that uses trip-related data to forecast the fare amount for taxi rides in New York City. This workflow was implemented in a PySpark environment on an Amazon EMR cluster using Spark MLlib.

Selected Features:

- *trip_distance*
- *trip_duration*
- *passenger_count*
- *pickup_hour (derived from timestamp)*

These features were assembled into a single feature vector using *VectorAssembler*, and the dataset was split into an 80/20 train-test split using *randomSplit*. A **Linear Regression model** was then trained using the *LinearRegressionclass* from Spark MLlib. The EMR cluster successfully finished the training, utilizing distributed resources to effectively handle millions of records. Among the steps in the workflow were:

- Input feature vectorization
- Using the training data to train the regression model
- Using the test set to apply the model
- Using regression metrics (RMSE, R^2) to assess predictions

This setup demonstrated a scalable machine learning process powered by cloud infrastructure, capable of handling large volumes of structured data with minimal manual configuration.

6. Model Evaluation and Results

Two common metrics were applied to the test dataset in order to assess the trained linear regression model:

R-squared (R^2), or root mean squared error (RMSE)

The accuracy of the model and its capacity to account for variation in the target variable are revealed by these indicators.

Findings:

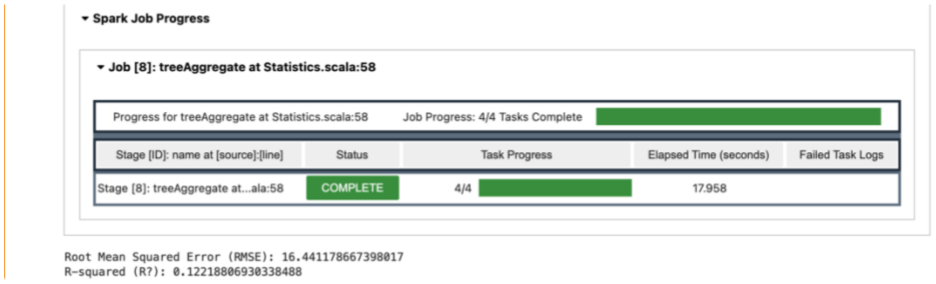
16.44 $\rightarrow$RMSE

0.122 $\rightarrow R^2$

The expected fare typically differs from the actual fare by roughly $16.44, according to the RMSE of 16.44. The model's input features account for about 12.2% of the variance in fare amount, according to the R^2 score of 0.122.

Despite the results' low predictive power, this is to be expected considering the model's simplicity and the small number of characteristics it uses. Numerous external factors, including as traffic, tolls, route efficiency, and real-time demand, that are not included in this dataset affect the fare amount.

PySpark's built-in *evaluate()* method was used to validate these results, and screenshots of the output were taken to bolster the conclusions in the report.



▾ Spark Job Progress

▾ Job [8]: treeAggregate at Statistics.scala:58

Progress for treeAggregate at Statistics.scala:58 Job Progress: 4/4 Tasks Complete

Stage [ID]: name at [source]:[line]	Status	Task Progress	Elapsed Time (seconds)	Failed Task Logs
Stage [8]: treeAggregate at...ala:58	COMPLETE	4/4	17.958	

```
Root Mean Squared Error (RMSE): 16.441178667398017
R-squared (R?): 0.12218806930338488
```

7. Architecture Diagrams

7.1 Core Architecture (Implemented)

The implemented pipeline uses AWS services integrated with Apache Spark running on EMR:

- Step 1: Data Staging
  The cleaned NYC Taxi CSV dataset was stored in an S3 bucket (s3://nyc-taxi-final-megha/cleaned/).
- Step 2: Distributed Processing
  An Amazon EMR cluster with PySpark was used to read, clean, and transform the dataset (e.g., derive trip duration, filter invalid values).

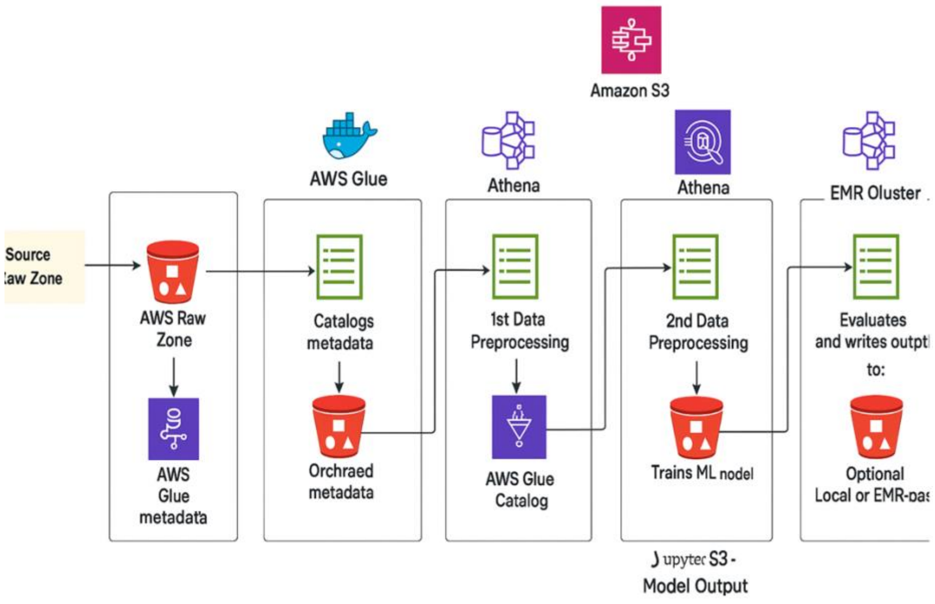
- Step 3: Transformed Data Storage

  The output was saved back to S3 in Parquet format at s3://nyc-taxi-final-megha/transformed/.
- Step 4: Cataloging

  AWS Glue was used to crawl the Parquet data and register it in the Glue Data Catalog.
- Step 5: Query Layer

  AWS Athena queried the data using SQL for validation and EDA.
- Step 6: Machine Learning

  The transformed data was used to train a Linear Regression model in Spark MLlib to predict fare amount.

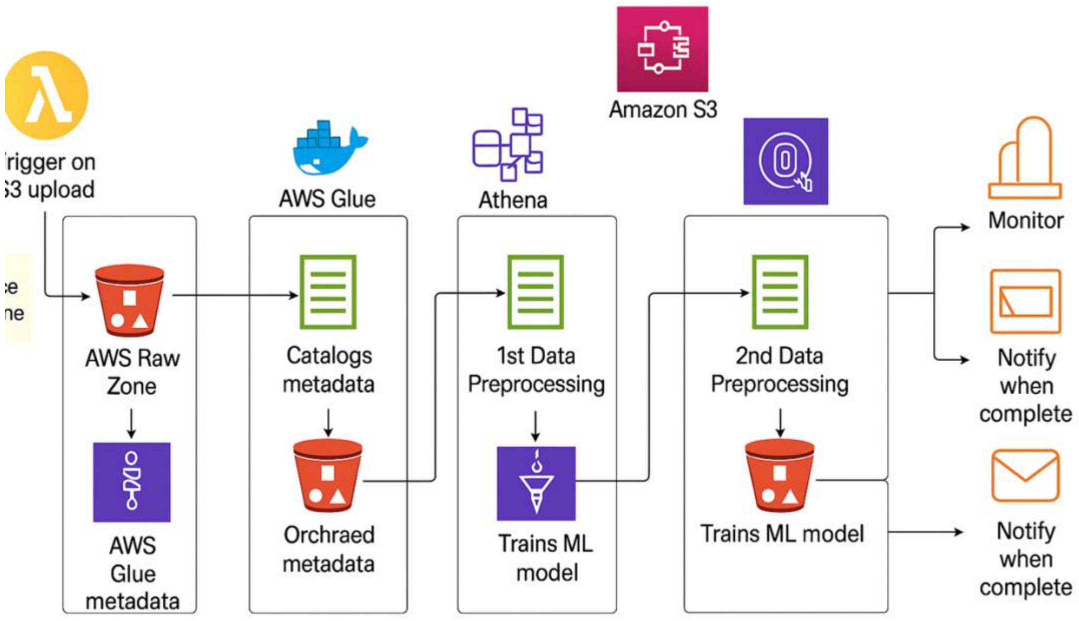


7.2 Full Architecture (Orchestrated Version)

The extended architecture represents a scalable, automated version of the solution suitable for production:

- AWS Lambda (or AWS Step Functions) is configured to automatically trigger when a new file is uploaded to the S3 raw zone, initiating the entire pipeline without manual intervention.

- AWS Glue catalogs metadata and prepares data for querying and further processing.
- Athena performs initial SQL-based data inspection and validation.
- Amazon EMR runs Spark jobs for advanced data transformation and machine learning using MLlib.
- Amazon S3 stores the output data and model results.
- Amazon CloudWatch monitors the pipeline execution, logs key events, and triggers notifications if errors occur.
- Amazon SNS sends notifications (e.g., email or SMS) upon job completion or failure, supporting operational awareness.



8. Challenges and Solutions

A significant obstacle encountered throughout this project was making sure Spark MLlib operated properly on the Amazon EMR cluster, especially because of missing Python requirements. NumPy, which was not pre-installed on the EMR nodes, is necessary for Spark's machine learning library. When training models, this resulted in runtime failures. Install_numpy.sh, a custom bootstrap script, was implemented and configured during the setup of the EMR cluster in order to fix the problem. At startup, this script made sure NumPy was installed on every node.

Connecting the appropriate EMR cluster to EMR Studio and making sure that networking and permissions settings were in line was another small difficulty. This was fixed by reattaching the appropriate cluster after halting and restarting the workspace.

Benefits of distributed computing

The effective processing of more than 3 million records was made possible by the use of distributed computing with Apache Spark on Amazon EMR. When compared to local processing, the execution speed of tasks like data transformation, feature engineering, and machine learning training was greatly increased by parallelizing them across several EC2 nodes. Large-scale data transformations and ML model training could be handled with little manual infrastructure management thanks to Spark's in-memory computation and EMR's scalability.

9. Conclusion and Future Work

Using distributed computing and machine learning, this study showed how to create a scalable, cloud-based pipeline for estimating the cost of cab fares in New York City. The solution processed, transformed, and analyzed more than 3 million cab trip records by utilizing AWS services like Amazon S3, EMR, Glue, and Athena. Key trip characteristics, such as distance, length, passenger count, and pickup time, were used to train a linear regression model constructed with Spark MLlib.

The entire end-to-end design was successfully validated by the model, despite its moderate predictive accuracy (R2 = 0.122). These findings suggest that trip-level characteristics can account for a portion of taxi fares, while pricing is probably influenced by other outside factors including tolls, traffic, and season.

Future Enhancements

- Incorporate geolocation features (pickup/dropoff zones)
- Include tolls, surcharges, and tip amount as predictors
- Experiment with more complex models (Random Forest, Gradient Boosted Trees)
- Automate the pipeline using orchestration tools like AWS Step Functions
- Deploy the model via a REST API for real-time fare estimation

□□□□