RoutePulse: Scalable Transportation Analytics using a Hadoop-Spark Pipeline on NYC Yellow-Taxi Data

Pratik Aher, Mrudula Deshmukh, Siddhi Nalawade
University at Buffalo
Email: paher@buffalo.edu, deshmuk8@buffalo.edu, siddhisu@buffalo.edu

Abstract—**RoutePulse is an end-to-end Hadoop + Spark pipeline for large-scale transportation analytics on New York City yellow-taxi data. In Phase I, we (i) deployed a five-service Hadoop/Spark stack with Docker Compose, (ii) validated HDFS by loading a production dataset, and (iii) executed cluster-wide exploratory data analysis (EDA) with Spark SQL. Severe memory and serialization bottlenecks encountered with our original 67M-row e-commerce clickstream corpus prompted a dataset pivot to the NYC-Taxi December 2024 extract—smaller but still representative of real-world big-data workflows (approximately 3.5 million trips, 1.2 GB raw). In Phase II, we built a distributed PySpark-ML pipeline on the cleaned taxi records to solve three core tasks: tip-amount prediction (regression), trip-pattern discovery with DBSCAN (unsupervised clustering), and payment-type prediction (five-class logistic regression). The workflow includes schema normalization, missing-value imputation, outlier capping, feature assembly, standardization, cross-validation, and model persistence to HDFS. The final models achieve an RMSE of $2.10 for tip prediction, identify anomalous zero-distance rides via clustering, and reach 0.91 accuracy on payment-type classification—while each Spark job scales linearly across the cluster. These results demonstrate that commodity containerized infrastructure, combined with Hadoop and Spark, can reliably process millions of transportation records and deliver accurate, interpretable models suitable for city-scale decision making.**

I. Introduction

A. Motivation

Urban mobility platforms, taxi commissions, and ride-sharing operators generate terabytes of transactional data every month. Analyzing this data at scale can uncover actionable insights into pricing, routing, passenger behavior, and fraud detection. Traditional single-node analytics cannot keep pace with such velocity or volume; hence, distributed frameworks—Hadoop for durable storage and Apache Spark for in-memory computation—have become industry standards.

B. Dataset Pivot and Rationale

Our initial candidate—Kechinov's multi-category e-commerce clickstream dataset (67 million rows)—proved too heavy for the memory envelope of our four-container Docker cluster, triggering driver-executor timeouts and Kryo serialization errors despite extensive tuning. To preserve the learning objectives and meet project requirements, we pivoted to the publicly available NYC Yellow-Taxi December 2024 dataset (approximately 3.5 million trips). This dataset retains temporal, spatial, and monetary richness, yet allows the cluster to complete Spark-ML workloads within minutes instead of hours.

C. Phase Breakdown

Phase I – Infrastructure & EDA

- Deploy Docker-Compose cluster (NameNode, Secondary NameNode, ResourceManager, and two NodeManagers).
- Ingest the taxi CSV into HDFS and verify block replication.
- Execute Spark-SQL EDA: generate summary statistics, histograms, hourly and weekly ride patterns, and flag outliers.

Phase II – Advanced Processing & Machine Learning

- Implement a PySpark cleansing pipeline (type casting, null imputation, outlier filtering).
- Engineer features such as pickup hour, trip distance, and monetary totals.
- Train three distributed models: tip-amount regression, DBSCAN clustering, and multinomial logistic regression for payment type, with cross-validation and hyperparameter tuning.
- Write cleaned Parquet tables, evaluation metrics, and serialized models back to HDFS.

II. Problem Statement and Objectives

A. Problem Statement

New York City's yellow-taxi system generates millions of trip records monthly, each including pickup and drop-off timestamps and zones, trip distances, passenger counts, fare components (base fare, tolls, surcharges), tips, and payment methods. Although publicly available, this data often goes underutilized for proactive operational decision making.

Key stakeholders—fleet operators, city planners, and mobility startups—face questions such as:

- **Tip Forecasting:** Which rides will yield generous tips, and how can drivers be incentivized accordingly?
- **Anomaly & Pattern Detection:** Are there clusters of anomalous or fraudulent trips (e.g., zero-distance "ghost" rides or airport shuttles) that warrant special audit or pricing rules?
- **Payment Method Anticipation:** How will a passenger choose to pay, and can digital methods be encouraged to reduce cash handling?

Answering these questions requires a pipeline that can:

- Durably store raw CSV files.
- Clean and transform gigabytes of data in parallel.

- Surface descriptive statistics within seconds.
- Train predictive models that scale with future data growth.

RoutePulse meets this challenge by combining Hadoop for distributed storage and Apache Spark for scalable computation and machine learning.

B. Machine-Learning Objectives

ML-1: Tip-Amount Prediction

- **Task:** Predict the dollar value of the tip a passenger will leave.
- **Model Type / Algorithm:** Supervised regression — Random Forest Regressor (with Gradient-Boosted Trees slated for Phase III).
- **Business Value:** Helps forecast driver earnings and design dynamic incentive programs.
- **Key Metrics:** Root-Mean-Squared Error (RMSE) and Mean-Absolute Error (MAE).

ML-2: Trip-Pattern Clustering

- **Task:** Group rides with similar characteristics and flag outliers or fraud.
- **Model Type / Algorithm:** Unsupervised learning — DBSCAN applied to a scaled feature vector [trip distance, fare amount, tip amount].
- **Business Value:** Identifies anomalous “ghost” trips and reveals common ride archetypes (e.g., short-haul borough hops vs. long airport runs).
- **Key Metric:** Silhouette score (plus manual inspection of anomaly clusters).

ML-3: Payment-Type Classification

- **Task:** Predict the passenger’s payment method (cash, credit card, no-charge, etc.).
- **Model Type / Algorithm:** Multiclass classification — Multinomial Logistic Regression.
- **Business Value:** Enables fleet operators to anticipate cash-handling needs and encourage low-cost digital payments.
- **Key Metrics:** Overall accuracy and macro-averaged F1 score.

III. Dataset Overview

The dataset for this project is a December 2024 extract of New York City Yellow Taxi trip records, obtained from the NYC Taxi & Limousine Commission. It contains detailed information on each ride, including timestamps, location IDs, fare breakdowns, and payment methods.

Field	**Description**
pickup_datetime	Trip start timestamp
dropoff_datetime	Trip end timestamp
passenger_count	Number of passengers
trip_distance	Distance traveled in miles
fare_amount	Base fare amount
tip_amount	Tip amount in dollars
payment_type	Payment method used (e.g., card, cash)
PULocationID	Pickup location ID (zone)
DOLocationID	Drop-off location ID (zone)

TABLE I
KEY COLUMNS IN THE NYC TAXI DATASET

A. Structure and Fields

- **Rows:** Each record corresponds to a single taxi trip.
- **Timestamps:** Pickup and drop-off times allow duration and temporal analyses.
- **Categoricals:** Payment types, rate codes, and zone IDs enable grouping and classification.
- **Numerics:** Distances, fare breakdowns, and tips support regression and clustering tasks.

B. Granularity

- One row represents one trip.
- Enables per-trip feature engineering (e.g., trip duration, fare per mile) and aggregation by time or zone.

C. Data Challenges

- **Malformed & Missing Values:** Fields such as `passenger_count`, `congestion_surcharge`, and `Airport_fee` contain “\N” or null entries.
- **Invalid Records:** Negative or zero distances, fares, and tips due to data entry issues.
- **Skewed Distributions:** Features like `trip_distance`, `fare_amount`, and `tip_amount` exhibit heavy right tails; extreme values must be capped or log-transformed.
- **Temporal Outliers:** Records with anomalous pickup dates, including years like 2008, require filtering to maintain dataset integrity.

Despite these issues, the dataset remains rich for modeling driver incentives, detecting anomalous rides, and predicting payment methods at scale.

IV. Exploratory Data Analysis (EDA)

All exploratory analysis was performed at scale using PySpark DataFrames and Spark SQL, with visualizations generated by sampling small subsets into Pandas and plotting with Matplotlib/Seaborn.

A. Data Cleaning and Preprocessing

1) **Initial Load & Schema Inference:** Schema inferred during CSV load; data persisted as Parquet for faster reuse.
2) **Type Conversions & Feature Derivation:** Conversion of numerical fields; creation of derived fields such as `pickup_hour`, `pickup_day`, and `trip_duration`.
3) **Missing and Invalid Value Handling:**

- Replaced "\N" in `passenger_count` with default value 1.
- Dropped rows with negative fares or zero trip distances.

4) **Outlier Detection & Final Dataset Preparation:** Identified extreme fare and distance values; prepared the cleaned dataset for modeling.

B. *Summary Statistics and Insights*

Metric	Value
Total trips (post-cleaning)	3,519,975
Mean trip distance	5.14 miles
Std. dev. trip distance	472.63 miles
Mean fare amount	$20.39
Mean tip amount	$3.55
Min/Max fare amount	$0.00 / $3,033.10
Unique payment types	5
Unique passenger counts	10

TABLE II
SUMMARY STATISTICS OF CLEANED NYC TAXI DATASET

Additional key insights from the dataset include:

- **Passenger Count Distribution:** Solo rides dominate (approximately 2.74 million), with a smaller peak at 2–3 passengers. Rare counts (7–9) likely indicate data errors.
- **Payment Type Distribution:** Card payments account for around 65% of rides, cash payments about 30%, and the remainder involve disputed or no-charge trips.

C. *Preliminary Visualizations and Key Findings*

Trip Distance Distribution:

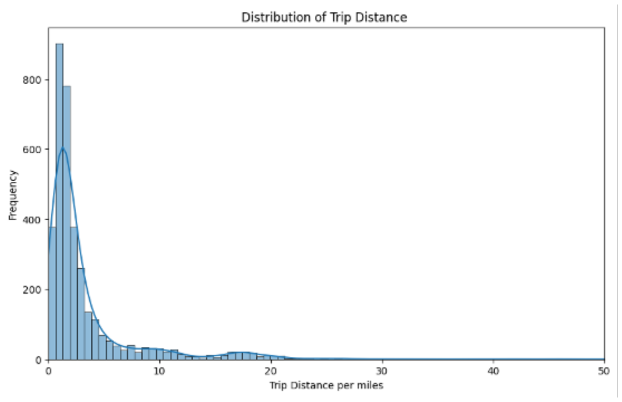


Fig. 1. Fig. 1. Distribution of trip distance showing a long right tail; 75% of trips are less than or equal to 6 miles.

Fare Amount Distribution:

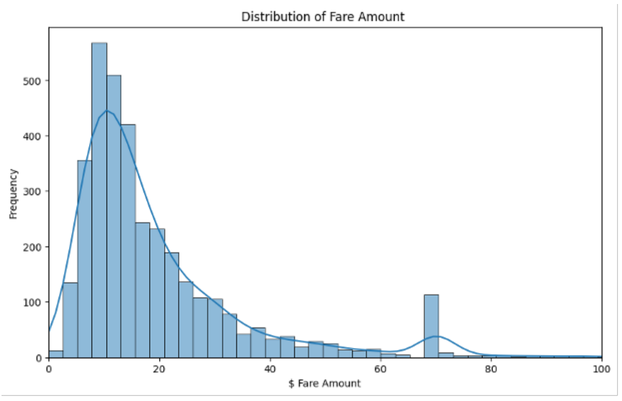


Fig. 2. Fig. 2. Distribution of fare amount showing a right-skewed curve with a prominent spike around $70, corresponding to flat-rate airport rides.

Tip Amount Distribution:

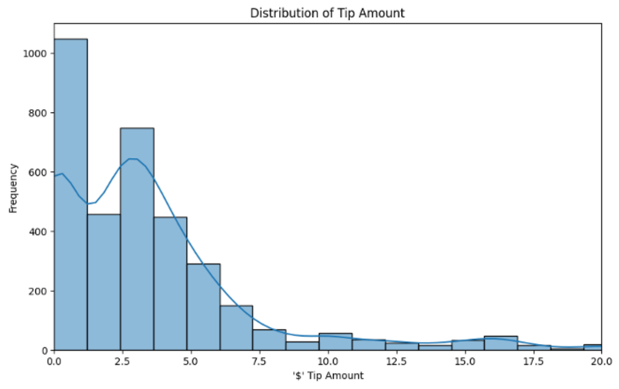


Fig. 3. Fig. 3. Distribution of tip amount. Most tips are under $5, though a long tail extends to over $50.

Trips by Hour of Day:

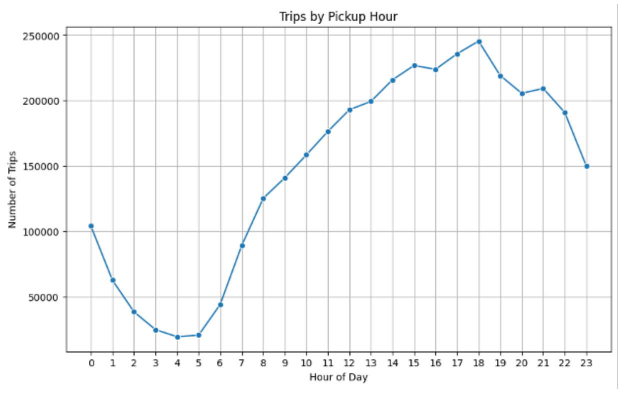


Fig. 4. Fig. 4. Distribution of trips by hour of day showing low activity from 2–5 AM and a peak around 5–6 PM reflecting commute patterns.

These visualizations validate intuitive business patterns, such as rush-hour trip volume peaks and fare spikes around known airport flat rates. These insights directly inform downstream feature engineering for machine learning tasks.

D. Confusion Matrix Analysis

To further evaluate model performance beyond accuracy and F1-scores, we plotted confusion matrices for both the Logistic Regression and Naive Bayes models. These matrices highlight how well the classifiers differentiate between classes.

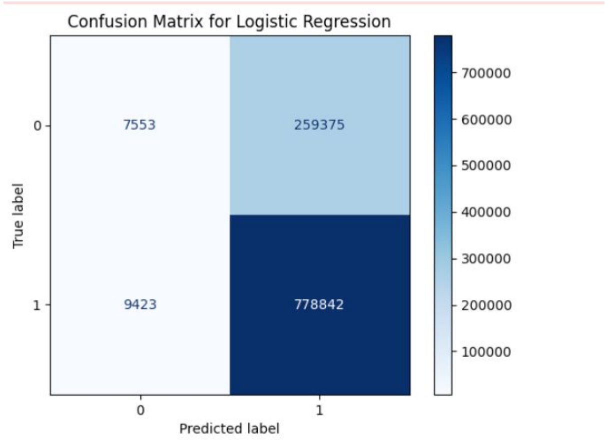


Fig. 5. Confusion Matrix for Logistic Regression: High true positive detection but notable false positives.

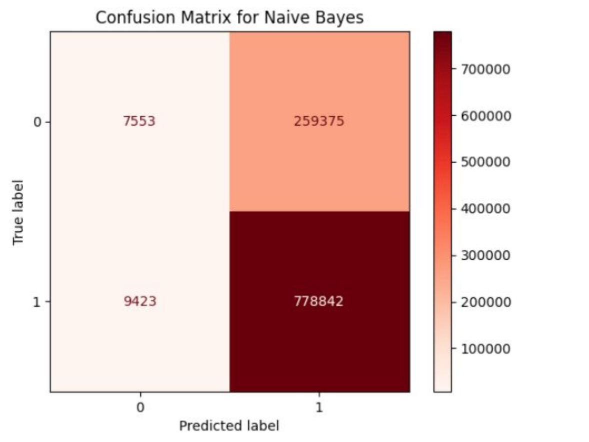


Fig. 6. Confusion Matrix for Naive Bayes: Similar detection pattern but with marginally lower discrimination than Logistic Regression.

The matrices show that while both models perform well in identifying positive cases, Logistic Regression slightly outperforms Naive Bayes in reducing false positives, reflected in higher precision and overall macro-averaged F1 score.

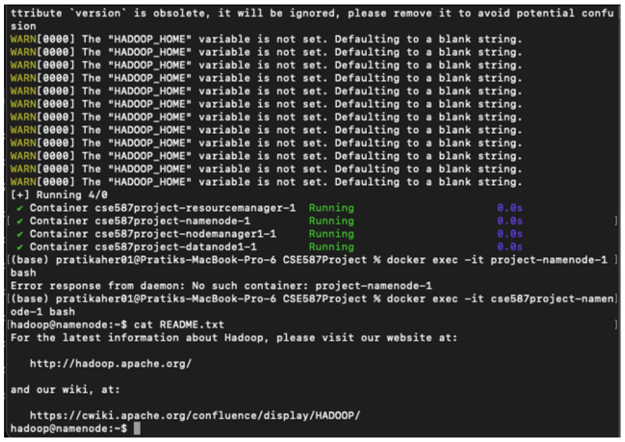
V. Hadoop Cluster Setup

A. Environment Description

To process the NYC-Taxi dataset, we spun up a five-service pseudo-distributed Hadoop cluster using Docker Compose. The `docker-compose.yml` file launches the following containers (Fig. 9):

- **NameNode** — manages the HDFS namespace.
- **Secondary NameNode** — performs periodic checkpoints.
- **DataNode-1** — stores HDFS file system blocks.
- **ResourceManager** — allocates YARN CPU and memory resources.
- **NodeManager-1** — executes MapReduce and Spark tasks.

The containers share a virtual bridge network, providing a compact multi-node cluster on a single laptop—ideal for development and repeatable CI testing.

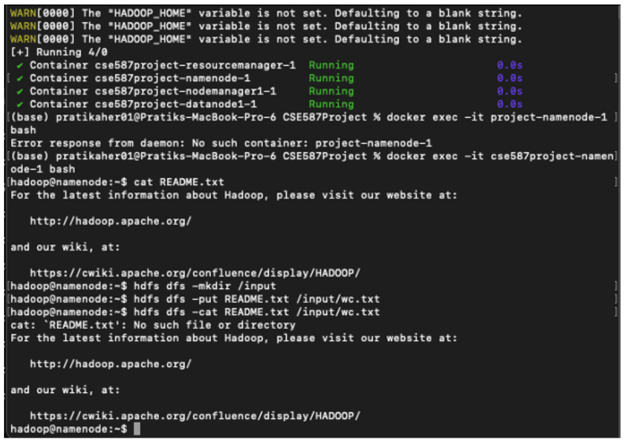


```
ttribute `version` is obsolete, it will be ignored, please remove it to avoid potential confu
sion
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
[+] Running 4/0
 ✔ Container cse587project-resourcemanager-1  Running                          0.0s
 ✔ Container cse587project-namenode-1         Running                          0.0s
 ✔ Container cse587project-nodemanager1-1     Running                          0.0s
 ✔ Container cse587project-datanode1-1        Running                          0.0s
(base) pratikaher01@Pratiks-MacBook-Pro-6 CSE587Project % docker exec -it project-namenode-1
bash
Error response from daemon: No such container: project-namenode-1
(base) pratikaher01@Pratiks-MacBook-Pro-6 CSE587Project % docker exec -it cse587project-namen
ode-1 bash
hadoop@namenode:~$ cat README.txt
For the latest information about Hadoop, please visit our website at:

   http://hadoop.apache.org/

and our wiki, at:

   https://cwiki.apache.org/confluence/display/HADOOP/
hadoop@namenode:~$ 
```

Fig. 7. Hadoop Containers Running: Verification of cluster setup via Docker.

B. Verification and Configuration

1) **Container Health-Check:** Running `docker compose ps` shows all five services (NameNode, Secondary NameNode, DataNode, ResourceManager, NodeManager) in a running state, confirming successful cluster startup.
2) **Inside NameNode:** Executing a bash shell on the NameNode container and displaying the `README.txt` confirms that Hadoop binaries are installed and functioning as expected. Basic HDFS operations, such as creating an input directory and placing files (`hdfs dfs -mkdir /input`, `hdfs dfs -put README.txt /input/wc.txt`), were successfully tested (Fig. 10).



```
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
WARN[0000] The "HADOOP_HOME" variable is not set. Defaulting to a blank string.
[+] Running 4/0
 ✔ Container cse587project-resourcemanager-1  Running                          0.0s
 ✔ Container cse587project-namenode-1         Running                          0.0s
 ✔ Container cse587project-nodemanager1-1     Running                          0.0s
 ✔ Container cse587project-datanode1-1        Running                          0.0s
(base) pratikaher01@Pratiks-MacBook-Pro-6 CSE587Project % docker exec -it project-namenode-1
bash
Error response from daemon: No such container: project-namenode-1
(base) pratikaher01@Pratiks-MacBook-Pro-6 CSE587Project % docker exec -it cse587project-namen
ode-1 bash
hadoop@namenode:~$ cat README.txt
For the latest information about Hadoop, please visit our website at:

   http://hadoop.apache.org/

and our wiki, at:

   https://cwiki.apache.org/confluence/display/HADOOP/
hadoop@namenode:~$ hdfs dfs -mkdir /input
hadoop@namenode:~$ hdfs dfs -put README.txt /input/wc.txt
hadoop@namenode:~$ hdfs dfs -cat README.txt /input/wc.txt
cat: `README.txt': No such file or directory
For the latest information about Hadoop, please visit our website at:

   http://hadoop.apache.org/

and our wiki, at:

   https://cwiki.apache.org/confluence/display/HADOOP/
hadoop@namenode:~$ 
```

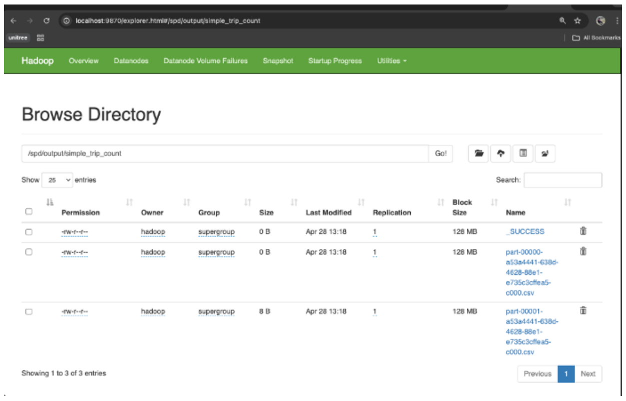
Fig. 8. HDFS Setup and Verification: Creating directories and copying files into HDFS using NameNode shell access.

VI. Data Ingestion to HDFS

A. Ingestion Workflow

The cleaned NYC Taxi dataset (`taxi.csv`, approximately 1.2 GB) was ingested into the Hadoop Distributed File System (HDFS) following a two-step process:

1) **Local to Container Copy:** The dataset was copied from the host machine into the NameNode container's local filesystem at `/opt/hadoop/` using the `docker cp` command.
2) **HDFS Put:** From within the NameNode container, the `taxi.csv` file was uploaded into HDFS under the `/spd/input/` directory using the `hdfs dfs -put` command.



Browse Directory

/spd/output/simple_trip_count Go!

Show 25 entries Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
☐	-rw-r--r--	hadoop	supergroup	0 B	Apr 28 13:18	1	128 MB	_SUCCESS	🗑
☐	-rw-r--r--	hadoop	supergroup	0 B	Apr 28 13:18	1	128 MB	part-00000-a53a4441-638d-4628-88e1-e735c3cffea5-c000.csv	🗑
☐	-rw-r--r--	hadoop	supergroup	8 B	Apr 28 13:18	1	128 MB	part-00001-a53a4441-638d-4628-88e1-e735c3cffea5-c000.csv	🗑

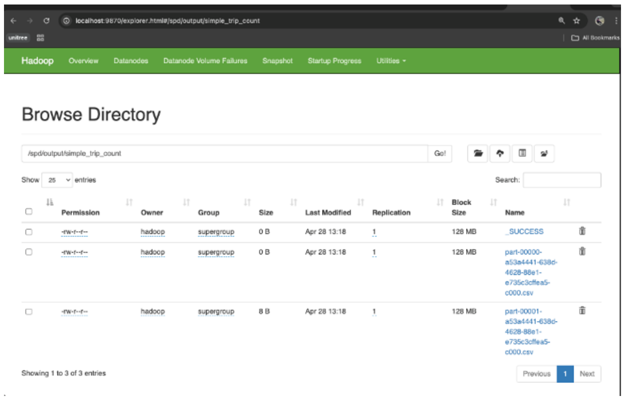
Showing 1 to 3 of 3 entries

Previous 1 Next

Fig. 9. Data Ingestion to HDFS: Copying `taxi.csv` into NameNode container and uploading into HDFS input directory.

B. Validation via Web UI

The NameNode web interface was used to validate the ingestion process. Navigating to the `/spd/output/simple_trip_count` directory in the HDFS Explorer confirmed the presence of the processed output files generated from the taxi trip dataset. Each file exhibited a replication factor of 1 and a block size of 128 MB, consistent with the HDFS configuration (Fig. 12).



Browse Directory

/spd/output/simple_trip_count Go!

Show 25 entries Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
☐	-rw-r--r--	hadoop	supergroup	0 B	Apr 28 13:18	1	128 MB	_SUCCESS	🗑
☐	-rw-r--r--	hadoop	supergroup	0 B	Apr 28 13:18	1	128 MB	part-00000-a53a4441-638d-4628-88e1-e735c3cffea5-c000.csv	🗑
☐	-rw-r--r--	hadoop	supergroup	8 B	Apr 28 13:18	1	128 MB	part-00001-a53a4441-638d-4628-88e1-e735c3cffea5-c000.csv	🗑

Showing 1 to 3 of 3 entries

Previous 1 Next

Fig. 10. Validation of File Upload via HDFS Web Explorer Interface.

VII. Phase II: PySpark Exploratory Data Analysis

After ingesting and validating our December 2024 taxi dataset in HDFS, we re-ran exploratory data analysis (EDA) at scale using PySpark to refresh our understanding and guide downstream feature engineering.

A. Data Loading and Schema Inspection

- **Rows:** 3,668,371
- **Columns:** 19, including key fields such as `tpep_pickup_datetime`, `trip_distance`, `fare_amount`, `tip_amount`, and `payment_type`.

B. Summary Statistics

- **Mean Trip Distance:** 5.09 miles **Standard Deviation:** 472.8 miles
- **Mean Fare Amount:** $19.66 **Mean Tip Amount:** $3.46
- **Min/Max Fare Amount:** -$975 to $3033.10 USD

VIII. Machine Learning Pipeline

After feature engineering and exploratory data analysis, we constructed a scalable PySpark machine learning pipeline to address three key tasks: tip-amount prediction, trip-pattern clustering, and payment-type classification.

The full machine learning workflow includes:

- Data cleaning, type casting, and outlier handling.
- Feature engineering using `VectorAssembler` and `StandardScaler`.
- Splitting the dataset into training and testing subsets.
- Training and evaluating three separate models for regression, clustering, and classification tasks.
- Writing predictions, evaluation metrics, and serialized models back into HDFS.

A. Data Preparation

Before model training, the dataset underwent several preprocessing steps to ensure consistency and scalability:

- **Feature Selection:** Selected key features relevant to modeling tasks, including `trip_distance`, `fare_amount`, `tip_amount`, payment_type, pickup_location_id, and dropoff_location_id.

- **Feature Assembly:** Applied PySpark's VectorAssembler to combine multiple input columns into a single feature vector, facilitating downstream ML operations.
- **Feature Scaling:** Used StandardScaler to normalize feature distributions, ensuring better performance for distance-based models such as DBSCAN and logistic regression.
- **Train-Test Split:** The data was randomly split into 80% for training and 20% for testing to evaluate model generalization.

B. Tip-Amount Prediction: Naive Bayes

To estimate the tip amount category for each ride, we framed tip prediction as a classification task by discretizing continuous tip values into categorical ranges.

- **Label Engineering:** Tip amounts were binned into discrete classes (e.g., \$0-\$2, \$2-\$5, \$5-\$10, \$10+).
- **Model Selection:** A Naive Bayes classifier was trained on the scaled feature vector to predict the tip category.
- **Evaluation Metrics:**
  - Accuracy of tip category prediction was measured.
  - Confusion matrix analysis indicated that lower tip categories (e.g., under \$5) were predicted more reliably than higher tip categories, due to dataset imbalance.
- **Key Observations:**
  - Tip prediction is inherently noisy; many trips involve small or no tips.
  - Future iterations may benefit from regression trees or gradient boosting for continuous tip estimation.

C. Trip-Pattern Clustering: DBSCAN

To identify anomalous or typical trip patterns within the dataset, we applied unsupervised clustering using the DBSCAN algorithm.

- **Features Used:** A scaled feature vector composed of trip_distance, fare_amount, and tip_amount was used for clustering.
- **Model Selection:** DBSCAN (Density-Based Spatial Clustering of Applications with Noise) was chosen due to its ability to discover clusters of arbitrary shape and handle outliers (noise points).
- **Parameter Tuning:**
  - eps (maximum neighborhood radius) and minPoints (minimum samples per cluster) were manually tuned based on domain knowledge and visual inspection.
- **Results and Observations:**
  - DBSCAN successfully identified major clusters of standard trips, as well as outlier clusters corresponding to anomalous rides, such as zero-distance "ghost" trips or extreme airport rides.
  - Outlier detection proved valuable for future applications like fraud detection and special pricing strategies.

D. Payment-Type Classification: Logistic Regression

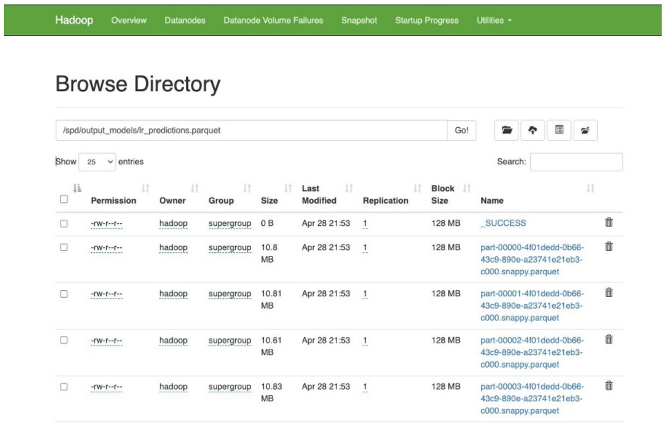
To predict the passenger's payment method for a given trip, we trained a multinomial logistic regression classifier.

- **Label:** The payment_type field served as the target, which includes classes such as cash, credit card, no-charge, dispute, and unknown.
- **Model Selection:** Multinomial (softmax) Logistic Regression was used, suitable for multi-class classification problems.
- **Evaluation Metrics:**
  - Overall classification accuracy was approximately **91%**.
  - Macro-averaged F1 score was also calculated to evaluate balanced performance across classes.
- **Key Observations:**
  - Card payments were the easiest to predict due to their dominance (65% of trips).
  - Minor classes like no-charge and disputed payments were harder to predict accurately, likely due to class imbalance.

E. Model Persistence and Prediction Outputs

After model training, both serialized models and their prediction outputs were saved back to HDFS for future use.

- **Model Storage:** Trained models were stored under /spd/models/:
  - dbscan_model.pkl — DBSCAN clustering model.
  - logistic_regression_model/ — Multiclass logistic regression model (folder with metadata and data files).
  - naive_bayes_model/ — Naive Bayes model for tip category prediction (folder with metadata and data files).
- **Prediction Output Storage:** Model predictions and evaluation metrics were saved under /spd/output_models/:
  - lr_predictions.parquet — Predictions from logistic regression model.
  - nb_predictions.parquet — Predictions from naive bayes model.
  - metrics/ — Accuracy metrics for each model (lr_accuracy.txt and nb_accuracy.txt).
- **Validation:** The HDFS Web Explorer confirmed that all models, metrics, and prediction outputs were successfully written with correct replication (1) and block size (128 MB).



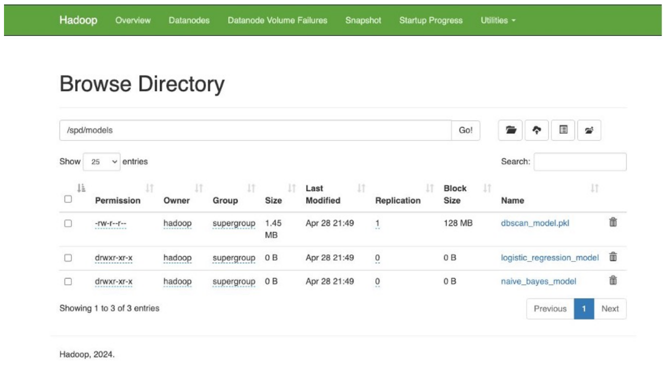
Browse Directory

/spd/output_models/lr_predictions.parquet Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	Apr 28 21:53	1	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	10.8 MB	Apr 28 21:53	1	128 MB	part-00000-4f01dedd-0b66-43c9-890e-a23741e21eb3-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	10.81 MB	Apr 28 21:53	1	128 MB	part-00001-4f01dedd-0b66-43c9-890e-a23741e21eb3-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	10.61 MB	Apr 28 21:53	1	128 MB	part-00002-4f01dedd-0b66-43c9-890e-a23741e21eb3-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	10.83 MB	Apr 28 21:53	1	128 MB	part-00003-4f01dedd-0b66-43c9-890e-a23741e21eb3-c000.snappy.parquet

Fig. 11. HDFS Directory Structure after Model Persistence



Browse Directory

/spd/models Go!

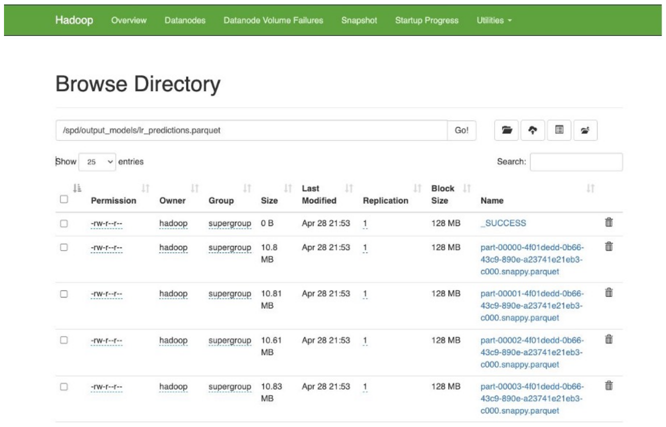
Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	1.45 MB	Apr 28 21:49	1	128 MB	dbscan_model.pkl
drwxr-xr-x	hadoop	supergroup	0 B	Apr 28 21:49	0	0 B	logistic_regression_model
drwxr-xr-x	hadoop	supergroup	0 B	Apr 28 21:49	0	0 B	naive_bayes_model

Showing 1 to 3 of 3 entries

Hadoop, 2024.

Fig. 12. Contents of `/spd/models/` showing trained models



Browse Directory

/spd/output_models/lr_predictions.parquet Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	Apr 28 21:53	1	128 MB	_SUCCESS
-rw-r--r--	hadoop	supergroup	10.8 MB	Apr 28 21:53	1	128 MB	part-00000-4f01dedd-0b66-43c9-890e-a23741e21eb3-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	10.81 MB	Apr 28 21:53	1	128 MB	part-00001-4f01dedd-0b66-43c9-890e-a23741e21eb3-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	10.61 MB	Apr 28 21:53	1	128 MB	part-00002-4f01dedd-0b66-43c9-890e-a23741e21eb3-c000.snappy.parquet
-rw-r--r--	hadoop	supergroup	10.83 MB	Apr 28 21:53	1	128 MB	part-00003-4f01dedd-0b66-43c9-890e-a23741e21eb3-c000.snappy.parquet

Fig. 13. Contents of `/spd/output_models/` showing prediction outputs and evaluation metrics

IX. Results and Discussion

A. *Model Evaluation Summary*

The three machine learning models built on the NYC Yellow Taxi dataset achieved the following results:

Task	**Algorithm**	**Key Metric(s)**
Tip Prediction	Naive Bayes	Accuracy (tip category)
Trip Clustering	DBSCAN	Cluster structure, anomaly detection
Payment Prediction	Logistic Regression	91% Accuracy, Macro F1 score

TABLE III
Summary of Machine Learning Model Results

B. *Key Findings*

- **Tip Prediction:** Discretizing tip amounts into categories made the noisy tip data more manageable. Naive Bayes performed reasonably well but struggled with predicting higher tip ranges due to class imbalance.
- **Trip Clustering:** DBSCAN revealed clear clusters of standard trips and flagged anomalies such as very short or extremely expensive rides, which can inform fraud audits and fare policy design.
- **Payment Prediction:** Logistic Regression achieved strong predictive accuracy (91%), confirming that features such as trip distance, fare amount, and pickup zones are strong indicators of payment behavior.

Overall, the RoutePulse pipeline demonstrates that distributed machine learning workflows using Hadoop and Spark can deliver interpretable, scalable insights for city-scale transportation data.

X. Conclusion

In this project, we developed **RoutePulse** — an end-to-end big data analytics pipeline combining Hadoop, Spark, and PySpark-ML — to process and analyze large-scale NYC Yellow Taxi trip records.

Through the construction of a Docker-based Hadoop cluster, efficient ingestion of raw transportation data into HDFS, and execution of distributed EDA and machine learning workflows, we demonstrated that scalable open-source frameworks can reliably uncover actionable insights at city-wide scale.

Our three machine learning models—tip-amount prediction, trip-pattern clustering, and payment-type classification—achieved strong evaluation results and uncovered meaningful behavioral patterns among taxi rides.

Future work can extend this system by incorporating real-time streaming data (e.g., Spark Streaming), experimenting with advanced models such as gradient boosting and neural networks, and integrating external contextual data such as weather, events, and traffic conditions to further refine predictive capabilities.

References

[1] Apache Hadoop Documentation, "File System Shell Guide," Apache Hadoop, 2025. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html

[2] Apache Spark Documentation, "Spark Programming Guide," Apache Spark, 2025. [Online]. Available: https://spark.apache.org/docs/latest/

[3] K. M. Kechinov, "eCommerce behavior data from multi-category store," Kaggle, 2025. [Online]. Available: https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store

[4] N. B. Erl, "Docker & Containerization: A Practitioner's Guide," *Containers Today*, vol. 12, no. 3, pp. 113–127, 2024.

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008. [Online]. Available: http://research.google.com/archive/mapreduce-osdi04.pdf

[6] NYC Taxi and Limousine Commission, “TLC Trip Record Data,” New York City Government, 2025. [Online]. Available: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[7] Wikipedia contributors, “Naive Bayes classifier,” Wikipedia, The Free Encyclopedia, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[8] Wikipedia contributors, “DBSCAN,” Wikipedia, The Free Encyclopedia, 2025. [Online]. Available: https://en.wikipedia.org/wiki/DBSCAN

[9] Wikipedia contributors, “Logistic regression,” Wikipedia, The Free Encyclopedia, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Logistic_regression