Credit Card Fraud Detection

1. Introduction and Objectives

Credit card fraud poses a major threat to financial institutions, costing billions of dollars annually. This project aims to build a scalable, end-to-end pipeline that can detect fraudulent credit card transactions with high accuracy. By leveraging Apache Spark and AWS services (S3, Athena), we handle large volumes of data, perform feature engineering in PySpark, and apply machine learning using Spark MLlib. This approach can help reduce financial losses and provide real-time or near-real-time alerts on potentially fraudulent transactions.

2. Data Source

- **Dataset:** A synthetic credit card transaction dataset that simulates real-world usage.
- **Size:** Over 1 million records, exceeding 100 MB of data, ensuring a sufficiently large and complex dataset for distributed processing.
- **Features:** Key fields include:
    - step: Represents a unit of time (e.g., hours or discrete time steps).
    - nameOrig: The originator (customer) identifier.
    - amount: Transaction amount.
    - oldbalanceOrg / newbalanceOrig: Balances before and after the transaction.
    - isFraud: A binary label indicating fraudulent (1) or legitimate (0) transactions.

Reason for Choice: Fraud detection is a critical real-world problem for financial institutions. The dataset's scale and diversity of features make it ideal for exploring advanced distributed data processing and predictive modeling.

3. Data Processing and Transformation

All transformations are performed in PySpark on an EMR cluster for scalability:

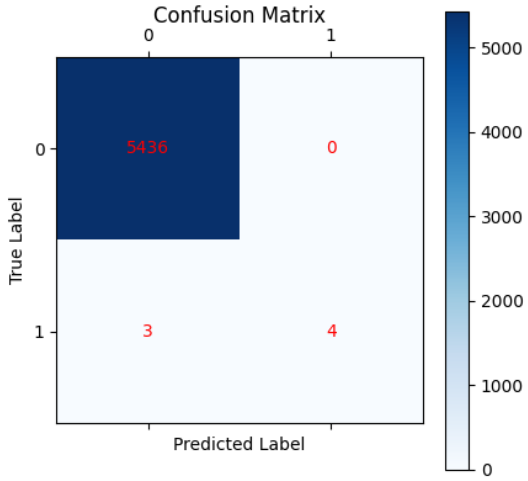
1. **Data Ingestion**
    - Raw data is uploaded to Amazon S3.
    - A Spark session reads the CSV file(s) into a DataFrame, using spark.read.csv() with header=True and inferSchema=True.
2. **Data Cleaning**
    - **Duplicate Removal:** Rows that are exact duplicates across all columns are dropped (df.dropDuplicates()).
    - **Null Filtering:** Transactions missing critical fields such as amount are removed (df.filter(col("amount").isNotNull())).
3. **Feature Engineering**
    - **Transaction Frequency:** For each originator (nameOrig), compute how many transactions occurred.
    - **Total/Avg/StdDev Amount:** Aggregate amount per originator to derive total, average, and standard deviation of transaction amounts.
    - **Balance Difference:** Calculate balance_diff = oldbalanceOrg - newbalanceOrig to see how balances shift with each transaction.
    - **Unique Transaction ID:** Generate an identifier using Spark's monotonically_increasing_id().
4. **Data Storage**
    - The cleaned and enriched DataFrame is written to S3 in Parquet format, facilitating efficient queries via AWS Athena.

4. Machine Learning Model Development

- **Preprocessing:**
    - Assemble features using VectorAssembler, combining columns like amount, transaction_frequency, and balance_diff into a single features vector.
    - Filter out rows where the label (isFraud) is null.
- **Modeling (Logistic Regression):**
    - A Logistic Regression model is trained on a 70/30 train-test split using Spark MLlib.
    - Hyperparameters (e.g., maxIter) are tuned for performance.
- **Evaluation:**
    - **ROC-AUC:** Measures the discriminative power of the classifier.
    - **Confusion Matrix:** Offers a detailed breakdown of true positives, false positives, true negatives, and false negatives.
    - **Precision, Recall, F1 Score:** Particularly important in fraud detection to assess performance on the minority (fraud) class.

4.1 Model Evaluation Visualizations

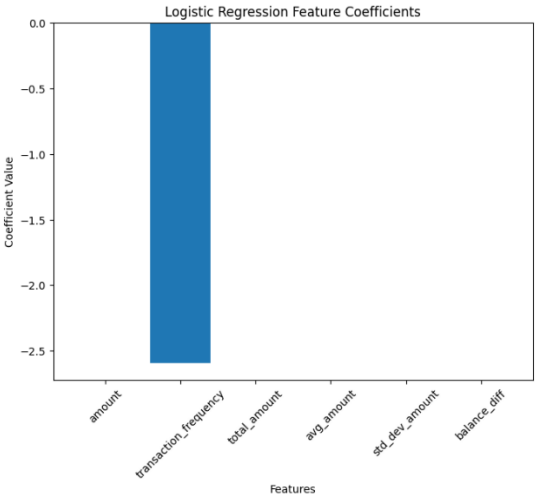
Confusion Matrix



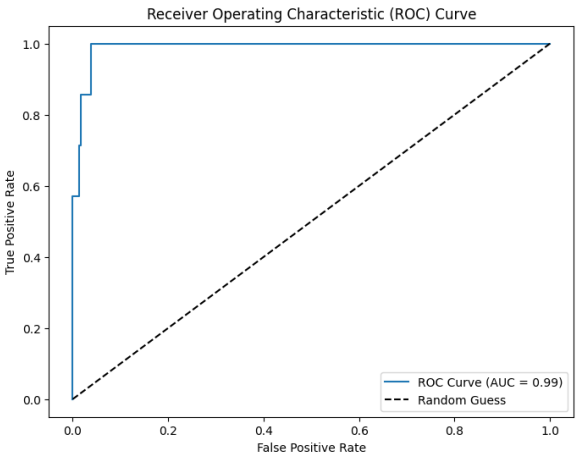
- Shows 5436 true negatives, 0 false positives, 3 false negatives, and 4 true positives.
- Indicates the model is highly accurate but also suggests a strong imbalance in the dataset.

Feature Coefficients

- transaction_frequency has a large negative coefficient, suggesting higher frequency might lower fraud probability in this dataset.
- Other features have relatively smaller magnitudes.



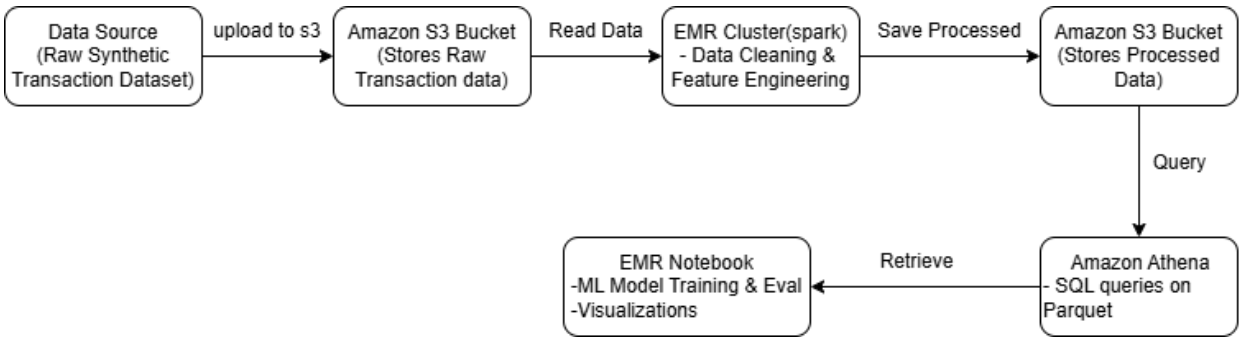
ROC Curve



- An AUC of ~0.99 implies the model discriminates very well between fraudulent and legitimate transactions.

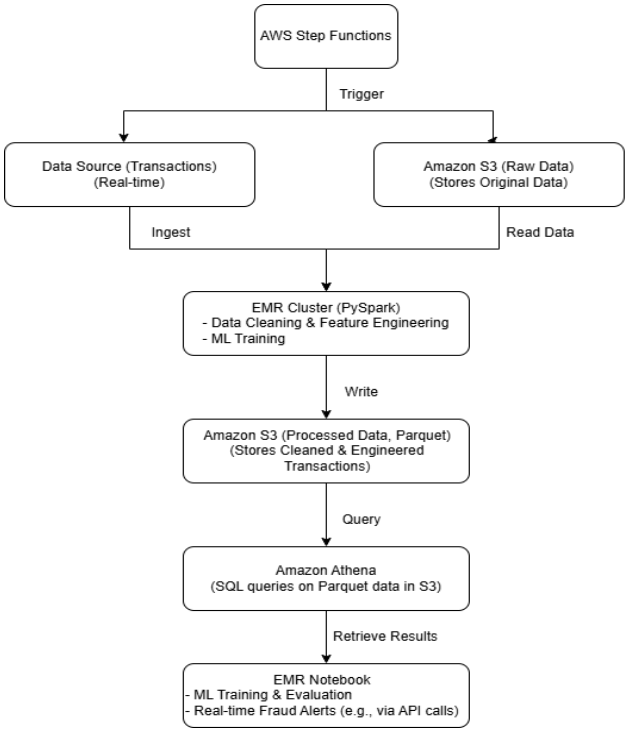
5. Architecture Diagrams

5.1 Current State Architecture



5.2 Future State Architecture

(For Real-Time or Batch Orchestration)



6. Challenges and Distributed Computing Benefits

- **Challenges:**
  - **Large Dataset:** Over 1 million records required memory-efficient processing.
  - **Class Imbalance:** Fraud is relatively rare, so standard accuracy metrics can be misleading.
- **Distributed Computing Benefits:**
  - **Scalability:** Apache Spark's distributed architecture speeds up data transformations.
  - **Cost-Effective Storage and Querying:** AWS S3 and Athena reduce the overhead of setting up dedicated data warehouses, enabling quick SQL queries on Parquet files.

7. Conclusion and Future Work

This project successfully demonstrates an end-to-end pipeline for credit card fraud detection, integrating Spark for large-scale data processing, AWS for storage and querying, and Spark MLlib for classification. The model shows high predictive performance, aided by robust feature engineering.

- **Future Enhancements:**
  Real-Time Deployment: Integrate a streaming solution (e.g., Kinesis) and a microservice approach for on-the-fly fraud detection.
  Advanced Algorithms: Explore ensemble methods (e.g., Random Forest, Gradient Boosting) or deep learning for potentially higher accuracy.
  Expanded Data Sources: Incorporate additional features (e.g., user demographics, external signals) to enrich the model.

Appendices

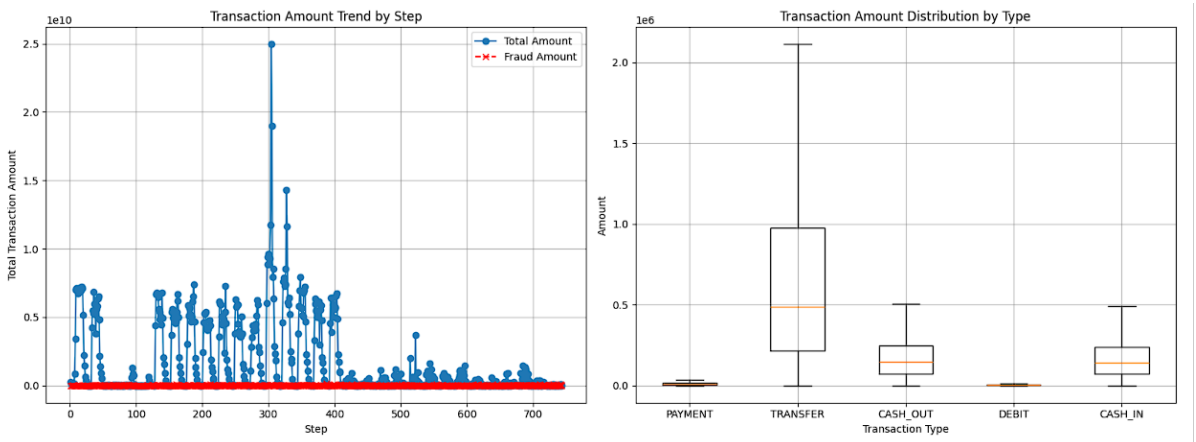
- **Codebase:** Python and PySpark scripts used for data cleaning, transformation, and modeling.
- **Athena Queries:**
  Query 1: Top 5 Originators by Total Transaction Amount

```
SELECT nameOrig AS Originator,
    SUM(amount) AS Total_Amount
FROM fraud_detection_table
GROUP BY nameOrig
ORDER BY Total_Amount DESC
LIMIT 5;
```

  Query 2: Fraud vs. Non-Fraud Counts Per Originator

```
SELECT nameOrig AS Originator,
isFraud,
COUNT(*) AS Transaction_Count
FROM fraud_detection_table
GROUP BY nameOrig, isFraud
ORDER BY Transaction_Count DESC;
```

- **Visualizations:**



Total vs. Fraud by Step:

- Compares total transaction amounts (blue) to fraudulent amounts (red) per "step," highlighting when large transaction spikes occur and the relative scale of fraud.

Distribution by Type:

- Groups transaction amounts by type, showing typical value ranges and which types involve larger sums, providing quick insight into potential risk areas.