Introduction and Objectives

This project presents a distributed, scalable fraud detection pipeline developed using Amazon Web Services (AWS) and Apache Spark. The primary goal is to process large-scale financial transaction data, engineer meaningful features, train a machine learning model for fraud detection, and generate useful insights and visualizations. Key AWS services include EMR (Elastic MapReduce), S3, Step Functions, Lambda, Athena, and Glue. Apache Spark is leveraged for data cleaning, transformation, and modeling at scale. The architecture is fully automated using Step Functions, ensuring efficient orchestration of all services.

Objectives:

- Efficient processing of large volumes of financial transaction data.
- Deployment of a machine learning model to predict fraudulent transactions.
- Automation of the ETL and ML workflow via AWS Step Functions.
- Visualization and storage of transformed data using Amazon Athena and S3.

Data Source and Justification

The dataset used is the "Financial Transactions Dataset for Fraud Detection" from Kaggle. It contains approximately 5 million synthetically generated transaction records, each with metadata, behavioral scores, device information, and fraud indicators.

https://www.kaggle.com/datasets/aryan208/financial-transactions-dataset-for-fraud-detection

Key features:

- ~5 million records with ~20 fields per record
- Includes timestamped transaction info, velocity scores, geo-anomalies, and fraud labels
- Suitable for supervised ML and behavioral analytics

Reason for Selection:

The dataset mirrors real-world fraud detection scenarios, includes diverse features, and is large enough to justify distributed processing using Spark. Its rich structure enables building robust machine learning models

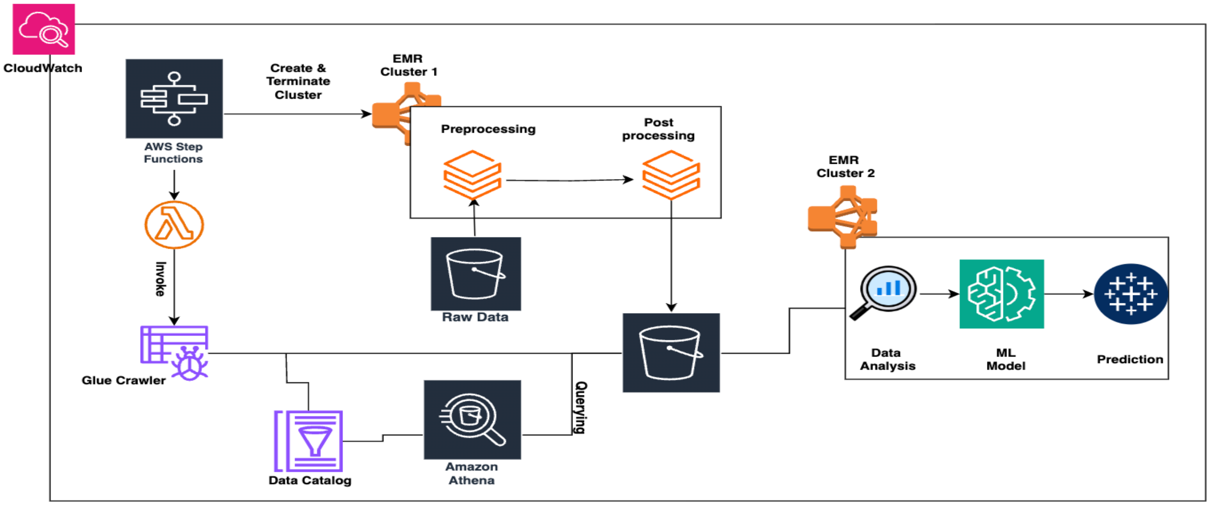
Data Processing and Transformation

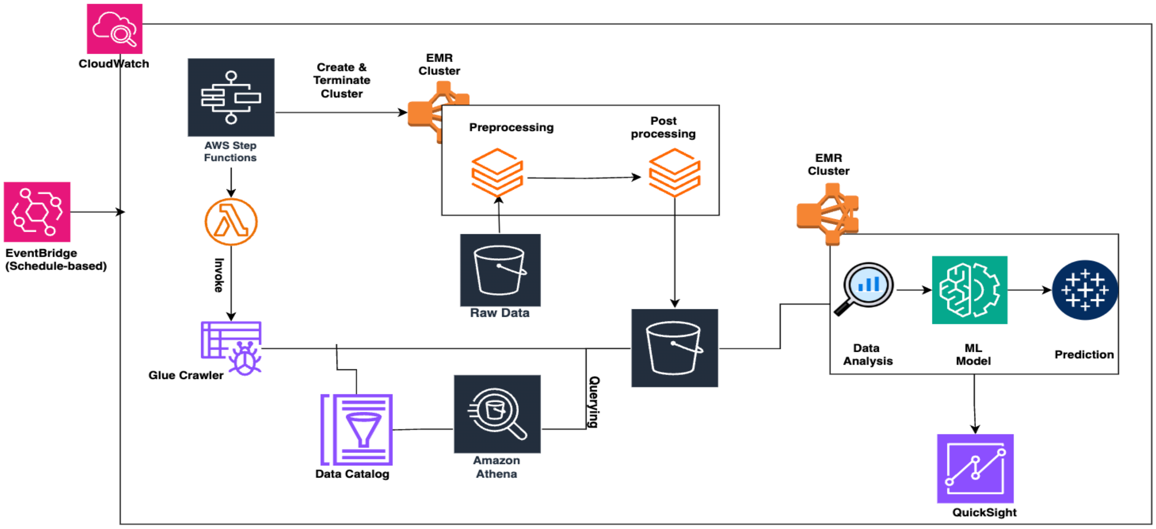
- Reading CSV from S3 using Spark
- Dropping null and invalid entries
- Filtering out negative or zero transaction amounts
- Casting categorical fields (like is_fraud) to appropriate types
- Engineeringfeatures: velocity_score, geo_anomaly_score, spending_deviation_score
- Saving transformed data to S3 in Parquet format

ML Model Development and Pipeline Orchestration

- Machine learning was performed using Spark MLlib. The model pipeline included:
- VectorAssembler for combining numerical features
- Splitting data into training and test sets (80/20)
- Logistic Regression classifier trained to predict fraud labels
- Model evaluation using AUC (Area Under ROC)

System Architecture





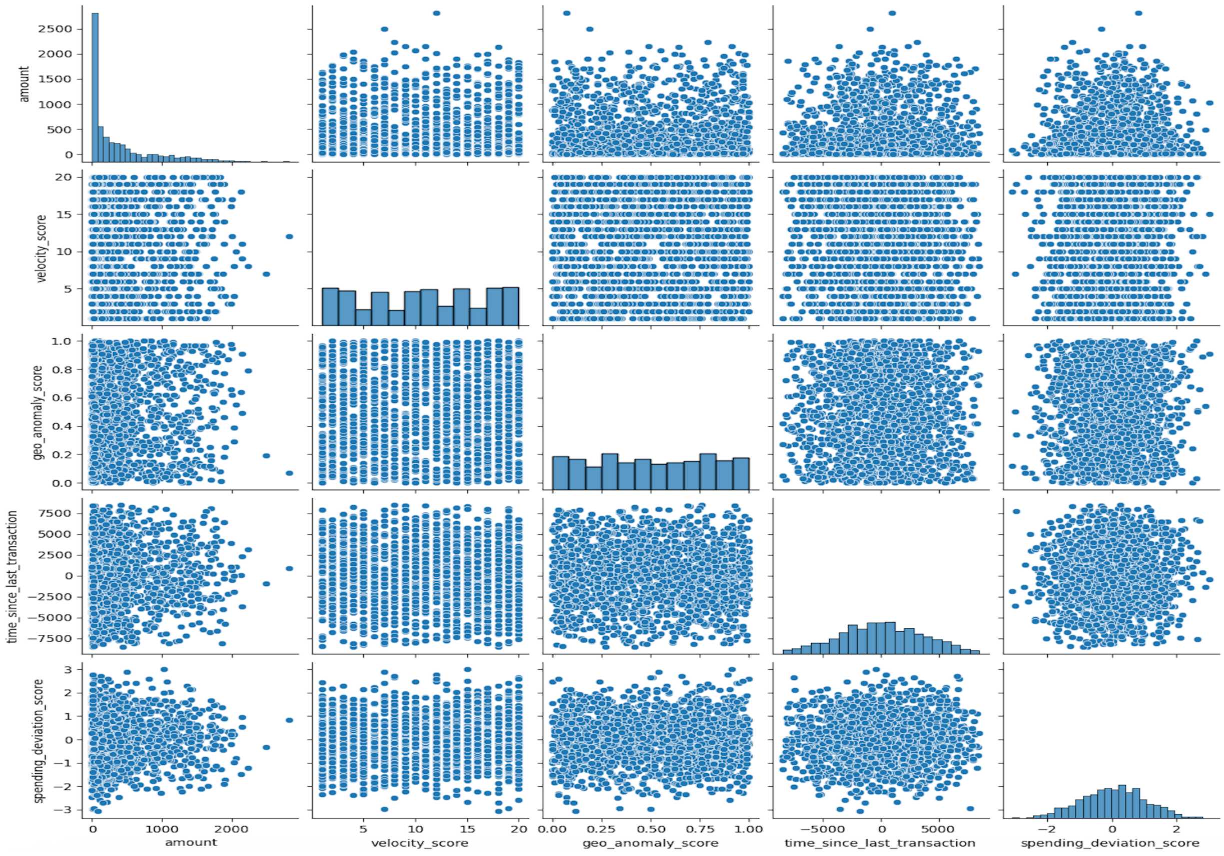
Evaluation and Results

The logistic regression model achieved the following:

- AUC Score: indicating good separation between fraud and non-fraud classes
- Prediction Summary: Tabulated actual vs. predicted values

The result distribution was visualized using stacked bar charts. Pairwise correlation of key features was analyzed via pairplots to understand relationships like velocity vs. amount.

Pairplot of Behavioral Features



Challenges and Benefits of Distributed Computing

Challenges:

- Cluster Resource Limits: EMR nodes (m4.large) at times failed to handle multiple jobs; fixed by adjusting shuffle partitions and node count
- File Path Issues: S3 path mismatches caused job failures; fixed via path verification in notebooks
- Library Installation: EMR Studio required %matplotlib inline and sc.install_pypi_package for visualizations

Distributed Computing Benefits:

- Enabled parallel transformation of millions of records in seconds
- Allowed scalable model training using Spark MLlib
- Simplified orchestration with AWS Step Functions and EMR job steps

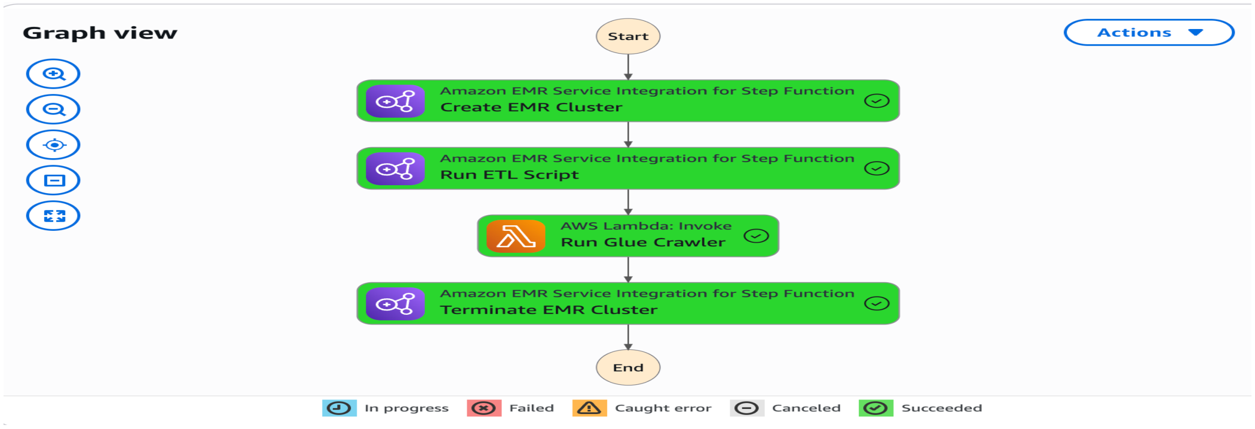
Conclusion and Future Work

This project successfully demonstrates how distributed computing and AWS can streamline the fraud detection process from ingestion to prediction. The pipeline is modular, scalable, and automatable.

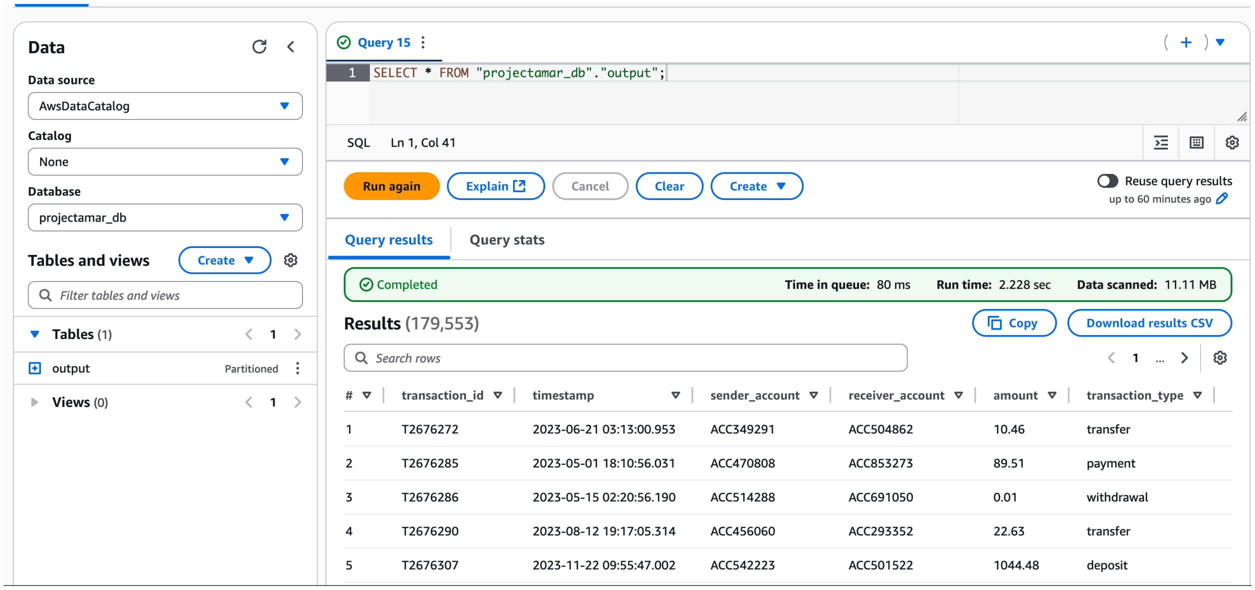
Future Enhancements:

- Adding SMOTE or class weighting to address class imbalance
- Evaluating advanced models like Random Forest or XGBoost in SageMaker
- Integrating real-time fraud alerting using Amazon Kinesis and Lambda
- Extend visualizations to include ROC curves, SHAP values, and time-based trend charts

Orchestration:



Athena: -



Data source: AwsDataCatalog

Catalog: None

Database: projectamar_db

```
SELECT * FROM "projectamar_db"."output";
```

Completed Time in queue: 80 ms Run time: 2.228 sec Data scanned: 11.11 MB

Results (179,553)

#	transaction_id	timestamp	sender_account	receiver_account	amount	transaction_type
1	T2676272	2023-06-21 03:13:00.953	ACC349291	ACC504862	10.46	transfer
2	T2676285	2023-05-01 18:10:56.031	ACC470808	ACC853273	89.51	payment
3	T2676286	2023-05-15 02:20:56.190	ACC514288	ACC691050	0.01	withdrawal
4	T2676290	2023-08-12 19:17:05.314	ACC456060	ACC293352	22.63	transfer
5	T2676307	2023-11-22 09:55:47.002	ACC542223	ACC501522	1044.48	deposit