Fraud Detection in Banking Transactions Using Hadoop

Prepared By: Sifatul Islam Tamim

University Name: VIT-AP University

Registration Number: 22BCE20246

Abstract

The "Fraud Detection in Banking Transactions Using Hadoop" project presents a comprehensive solution for identifying fraudulent credit card transactions by leveraging the Hadoop ecosystem's distributed computing capabilities. Designed to address the growing challenge of financial fraud, the system integrates batch and real-time data processing to deliver scalable, high-performance fraud detection. Batch data, including cardholder metadata (card_member), credit scores (member_score), and historical transactions (card_transactions), is ingested from AWS RDS and flat files using Apache Sqoop and stored in Hadoop Distributed File System (HDFS) with optimized formats like Parquet. Real-time transactions from point-of-sale (POS) terminals are streamed via Apache Kafka and processed using Apache Spark Streaming. The fraud detection methodology employs three rule-based techniques: an Upper Control Limit (UCL) rule to flag high-amount outliers, a credit score threshold rule to identify risky users, and a zip code distance check to detect geographically impossible transactions. Data is managed using Apache Hive for querying and Apache HBase for low-latency access, ensuring efficient storage and retrieval. The system achieves near real-time detection, reduces financial losses, and scales seamlessly with increasing transaction volumes. By combining robust data integration, rule-based analytics, and Hadoop's distributed architecture, this project enhances banking security and operational efficiency, offering a cost-effective solution for modern financial institutions.

Introduction

The "Fraud Detection in Banking Transactions Using Hadoop" project is designed to address the critical challenge of identifying fraudulent credit card transactions in the banking sector by leveraging the robust capabilities of the Hadoop ecosystem for large-scale data processing. The primary purpose is to develop an end-to-end pipeline that detects suspicious activities in both real-time and batch processing scenarios, ensuring rapid response to potential fraud while maintaining accuracy across high-volume transaction data. Financial institutions face increasing threats from sophisticated fraud schemes, necessitating advanced systems to safeguard customer assets and maintain trust. Hadoop is selected as the core technology due to its distributed computing framework, which excels in handling massive datasets across clusters of commodity hardware. Its ecosystem, including Apache Sqoop for data ingestion, Apache Hive and HBase for data storage and querying, Apache Kafka for real-time streaming, and Apache Spark Streaming for processing, provides a scalable and fault-tolerant environment. This enables the system to process diverse data sources efficiently, integrate structured and unstructured data, and deliver near real-time fraud detection, aligning with the demands of modern banking systems for high-throughput and low-latency analytics.

Objectives

- Real-Time Fraud Detection: Build a high-performance system to analyze point-of-sale (POS) transactions as they occur, utilizing streaming data from Kafka topics to identify and flag fraudulent activities instantaneously. This ensures immediate action to prevent unauthorized transactions, minimizing financial losses and enhancing customer security.
- Batch Fraud Detection: Process large volumes of historical transaction data stored in flat files and relational databases to uncover patterns of fraudulent behavior. This retrospective analysis supports the refinement of detection rules, improves model accuracy, and identifies trends that may not be evident in real-time data alone.
- Seamless Data Integration: Integrate heterogeneous data sources, including structured tables from AWS Relational Database Service (RDS) and unstructured flat files, into a unified pipeline. This enables comprehensive fraud analysis by combining cardholder metadata, credit scores, and transaction histories, ensuring a holistic view of customer activity.
- Scalability and Performance Optimization: Design a scalable architecture capable of handling exponentially growing transaction volumes without compromising performance. The system leverages Hadoop's distributed processing and storage capabilities to ensure consistent, low-latency fraud detection, even under high data loads, while maintaining reliability and fault tolerance for enterprise-grade banking applications.

Data Sources

- **card_member:** Hosted in AWS RDS, this dataset contains detailed metadata about credit cards and their associated members. Key fields include card ID (a unique identifier for each credit card), member ID (a unique identifier for the cardholder), and joining date (the date when the member enrolled in the card program). This dataset is critical for linking transactions to cardholder profiles and providing contextual information for fraud detection, such as verifying the legitimacy of card usage based on membership details.
- **member_score:** Also stored in AWS RDS, this dataset captures credit score information for cardholders, with fields including member ID and credit score (a numerical value reflecting creditworthiness). Updated incrementally every four hours via Apache Sqoop, this dataset supports the credit score-based fraud detection rule, enabling the system to flag transactions associated with low credit scores (e.g., below 200) as potentially fraudulent. It is stored in HBase for real-time access during streaming analysis.
- **card_transactions:** This dataset consists of historical transaction records stored in CSV flat files, containing detailed information such as transaction amount (the monetary value of the transaction), transaction date (when the transaction occurred), status (e.g., approved, declined, or pending), and location (represented by postcode). Loaded into HDFS using commands like `hdfs dfs -put` and processed via Hive-HBase integration, this dataset is used for batch processing to compute statistical metrics like moving averages and standard deviations for anomaly detection.
- **Kafka Stream:** Real-time transaction data is ingested from POS terminals through Apache Kafka topics, providing a continuous stream of live transaction records. Each record includes fields such as card ID, member ID, transaction amount, POS ID (a unique identifier for the point-of-sale terminal), postcode (indicating the geographic location of the transaction), and transaction date/time (a timestamp of when the transaction occurred). This stream is processed by Apache Spark Streaming to apply fraud detection rules in near real-time, enabling immediate identification of suspicious activities based on predefined criteria.

System Architecture

The system architecture for this fraud detection project leverages Hadoop's distributed computing framework to manage large-scale data processing for both batch and real-time transaction analysis. Data ingestion occurs through two primary methods: Apache Sqoop handles batch data extraction from AWS RDS, importing tables like `card_member` and `member_score` into Hadoop Distributed File System (HDFS) using incremental jobs with configurations such as `--incremental append` or `--lastmodified`, ensuring efficient updates based on specified check columns and last values. Real-time data from point-of-sale (POS) terminals is ingested via Apache Kafka, which streams transaction records into the system for immediate processing. Data storage is optimized using HDFS for scalable, fault-tolerant storage of raw and processed datasets in formats like ORC or Parquet, enhancing query performance. Apache Hive serves as a data warehousing layer, enabling SQL-like queries on HDFS data and supporting external tables for historical transaction data from CSV files. Apache HBase provides low-latency, random-access storage for transactional data, using composite row keys (e.g., `card_id + transaction_dt`) for fast lookups during real-time analysis. Data processing is handled by Apache Spark Streaming, which consumes Kafka streams, applies fraud detection rules, and writes results to Hive or HBase. This architecture ensures Hadoop's scalability and reliability, seamlessly integrating batch and real-time pipelines for comprehensive fraud detection.

Fraud Detection Methodology

The fraud detection methodology employs three rule-based techniques to identify suspicious transactions, leveraging Hadoop's processing power for efficient computation across large datasets:

- Upper Control Limit (UCL) Rule: This statistical method flags high-amount transactions as outliers by calculating the UCL as the moving average of the last 10 genuine transactions for a given card plus three times their standard deviation. Implemented using Hive's window functions, the system extracts recent transaction data, computes the moving average and standard deviation, and stores UCL values in a Hive table, with HBase updates for real-time access. Transactions exceeding the UCL are marked as potentially fraudulent, ensuring detection of unusual spending patterns.

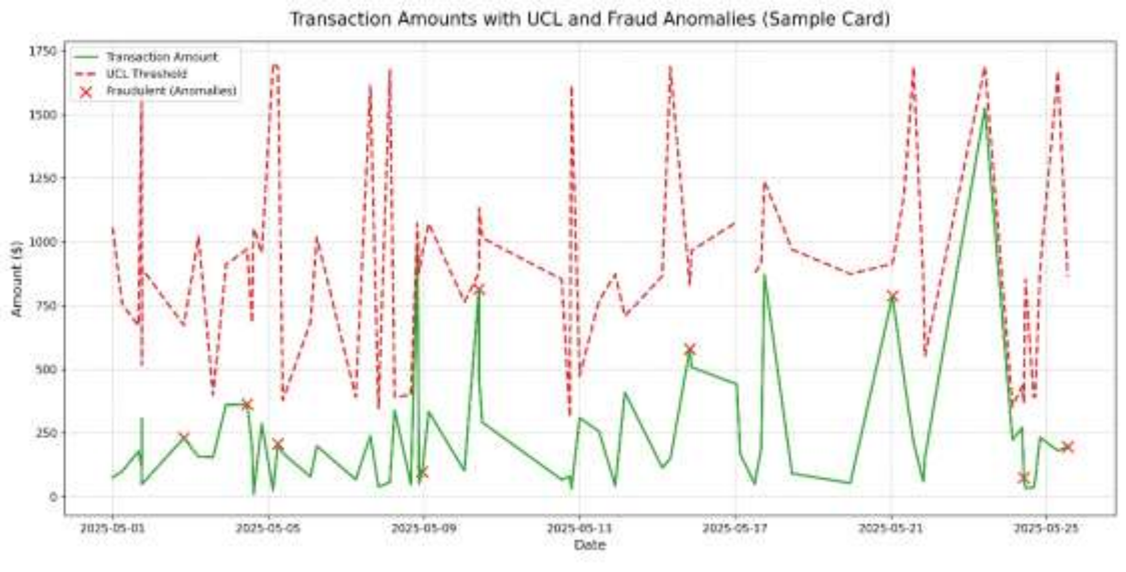

Figure 1: Transaction Amounts with UCL Threshold and Detected Fraud Anomalies for a Sample Card, Illustrating the UCL Rule"

- Credit Score Threshold Rule: Transactions linked to members with poor credit history are flagged by checking if the credit score is below 200. Credit scores, sourced from the `member_score` dataset in AWS RDS, are incrementally updated every four hours using Sqoop and stored in HBase for quick retrieval during streaming analysis. This rule leverages Hadoop's ability to handle frequent data updates and ensures rapid identification of high-risk transactions based on creditworthiness.
- Zip Code Distance Check: This geospatial rule identifies impossible travel patterns by calculating the speed (distance divided by time) between consecutive transactions using a custom postcode distance library. For each new transaction, the system retrieves the previous transaction's postcode and timestamp from HBase, computes the distance using the library, and determines the speed. If the calculated speed exceeds humanly feasible limits (e.g., implying unrealistic travel), the transaction is flagged as fraudulent. Genuine transactions trigger updates to HBase with the new postcode and timestamp, enabling continuous tracking within Hadoop's ecosystem.

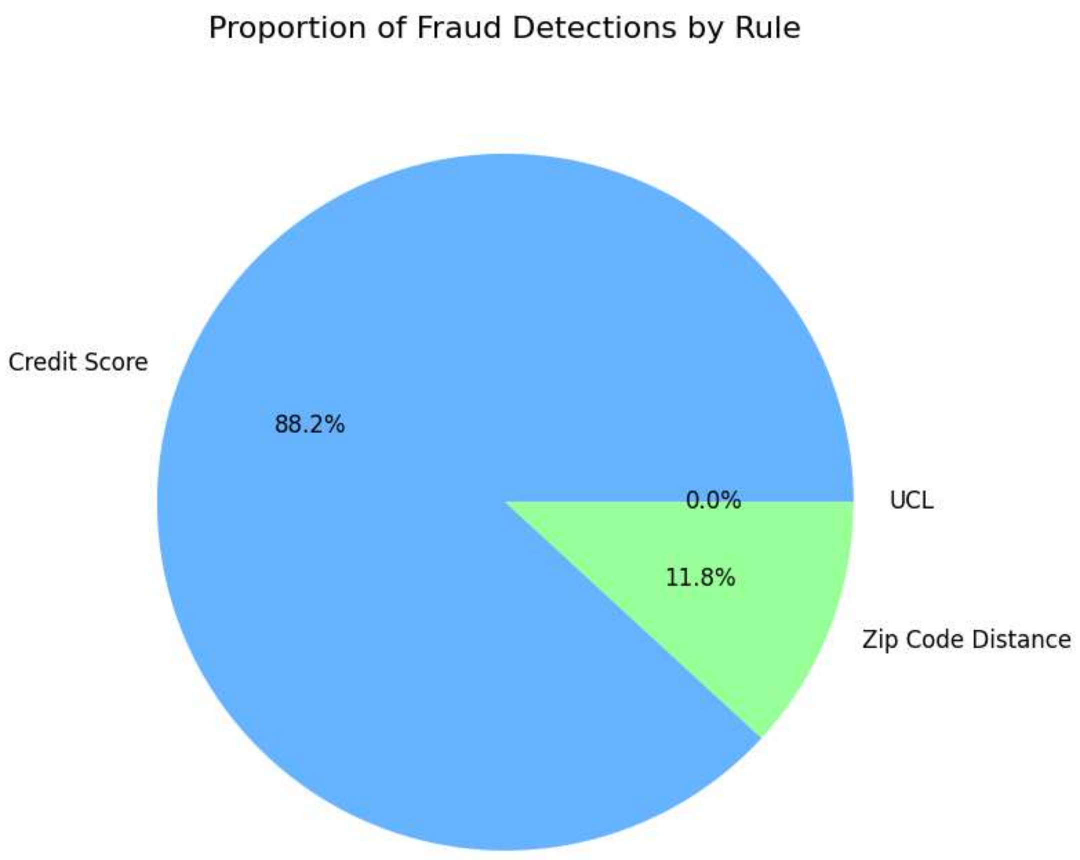

Figure 2: Proportion of Fraud Detections by Rule.

Technologies and Tools

The project utilizes a robust set of Hadoop ecosystem tools and custom components to enable scalable fraud detection:

- Apache Sqoop: Facilitates batch data ingestion by transferring structured data from AWS RDS tables (`card_member`, `member_score`) to HDFS, with incremental imports for efficient updates.
- HDFS (Hadoop Distributed File System): Provides scalable, fault-tolerant storage for raw and processed datasets, using optimized formats like ORC or Parquet to support high-performance querying.
- Apache Hive: Acts as a data warehouse, enabling SQL-like queries on HDFS data and supporting external tables for historical transaction data, integrated with HBase for transactional storage.
- Apache HBase: Offers low-latency, random-access storage for real-time transaction data, using composite row keys for efficient lookups during fraud detection.
- Apache Kafka: Streams real-time transaction data from POS terminals, enabling immediate processing of live data within the Hadoop ecosystem.
- Apache Spark Streaming: Processes Kafka streams in near real-time, applying fraud detection rules and writing results to Hive or HBase, leveraging Hadoop's distributed computing for scalability.
- Custom Postcode Distance Library: A specialized library for calculating distances between postcodes, used in the zip code distance rule to detect geographically impossible transactions, complementing Hadoop's processing capabilities.

Key Features

- The "Fraud Detection in Banking Transactions Using Hadoop" project incorporates several critical features that enhance its effectiveness and scalability:
- Real-Time Fraud Detection: Utilizes Apache Kafka and Spark Streaming to process point-of-sale transactions instantly, enabling rapid identification and flagging of suspicious activities to prevent unauthorized transactions.
- Robust Rule-Based Analytics: Employs three distinct rules—Upper Control Limit (UCL) for outlier detection, credit score thresholds for risk assessment, and zip code distance checks for geospatial validation—ensuring comprehensive fraud detection.
- Scalable Data Processing: Leverages Hadoop's distributed architecture, including HDFS for storage and Spark for processing, to handle millions of transactions daily with high throughput and low latency.
- Seamless Data Integration: Integrates diverse data sources (AWS RDS tables, flat files, and Kafka streams) into a unified pipeline, providing a holistic view of transaction and cardholder data for accurate analysis.
- Low-Latency Storage and Querying: Uses Apache HBase for real-time transaction access and Apache Hive for efficient batch querying, optimizing performance across both real-time and historical data analysis.

Implementation Details

The implementation of the fraud detection pipeline leverages Hadoop's ecosystem to ensure efficient data ingestion, storage, and processing for both batch and real-time scenarios. Below are the detailed configurations and setups for key components:

- Apache Sqoop Configurations: Sqoop is used to ingest batch data from AWS RDS tables (`card_member` and `member_score`) into HDFS. Incremental imports are configured to handle updates efficiently. For example, the Sqoop command for `card_member` is structured as:

```bash
sqoop import \
 --connect jdbc:mysql://aws-rds-endpoint:3306/banking_db \
 --username <user> --password <pass> \
 --table card_member \
 --target-dir /hdfs/card_member \
 --incremental append \
 --check-column joining_date \
 --last-value <last_imported_date> \
 --as-parquetfile
```

Similarly, `member_score` is imported with `--incremental lastmodified` to capture updated credit scores every four hours. The `--as-parquetfile` option ensures data is stored in Parquet format for optimized querying in Hive. Sqoop jobs are scheduled using a workflow manager like Apache Oozie to automate periodic imports, ensuring data freshness without manual intervention.

- HBase Table Design: HBase is used for low-latency storage and retrieval of transactional data, critical for real-time fraud detection. The `card_transactions` table in HBase uses a composite row key combining `card_id` and `transaction_dt` (e.g., `12345_2025-05-26T12:00:00`) to ensure unique identification and efficient range queries. The table schema is defined as:

```
CREATE TABLE card_transactions (
 row_key STRING,
 cf:amount DOUBLE,
 cf:pos_id STRING,
  cf:postcode STRING,

  cf:transaction_dt STRING,

  cf:status STRING

) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'

WITH SERDEPROPERTIES ('hbase.columns.mapping' =
':key,cf:amount,cf:pos_id,cf:postcode,cf:transaction_dt,cf:status')

TBLPROPERTIES ('hbase.table.name' = 'card_transactions');
```

The column family `cf` stores transaction details, and the composite key prevents hot-spotting by distributing data evenly across HBase regions. Historical transaction data from CSV files is loaded into HBase via Hive using `INSERT INTO` statements, ensuring seamless integration with batch processing.

- Spark Streaming Setup: Apache Spark Streaming processes real-time transactions from Kafka topics for immediate fraud detection. The setup involves configuring a Spark Streaming job to subscribe to the Kafka topic `pos_transactions`. A sample configuration in Scala (compatible with Spark) is:

```
import org.apache.spark.streaming.{Seconds, StreamingContext}

import org.apache.spark.streaming.kafka010._

import org.apache.kafka.common.serialization.StringDeserializer


val sparkConf = new SparkConf().setAppName("FraudDetection")

val ssc = new StreamingContext(sparkConf, Seconds(1))

val kafkaParams = Map[String, Object](

  "bootstrap.servers" -> "kafka-broker:9092",

  "key.deserializer" -> classOf[StringDeserializer],

  "value.deserializer" -> classOf[StringDeserializer],

  "group.id" -> "fraud_detection_group",

  "auto.offset.reset" -> "latest"

)

val topics = Array("pos_transactions")

val stream = KafkaUtils.createDirectStream[String, String](

  ssc,
 LocationStrategies.PreferConsistent,
 ConsumerStrategies.Subscribe[String, String](topics, kafkaParams)
)
stream.map(record => parseTransaction(record.value())).foreachRDD { rdd =>
 // Apply fraud detection rules and write results to HBase/Hive
}
ssc.start()
ssc.awaitTermination()
```

The `parseTransaction` function extracts fields like `card_id`, `amount`, `postcode`, and `transaction_dt` from the Kafka payload. Each transaction is processed by querying HBase for UCL values, credit scores, and previous transaction details, applying fraud rules, and writing results (Genuine or Fraudulent) to HBase or Hive. The batch interval is set to 1 second to balance latency and throughput, adjustable based on Kafka topic volume.

Outcomes

The fraud detection pipeline delivers significant benefits for banking institutions, leveraging Hadoop's scalability and performance:

- Faster Fraud Detection: Real-time processing with Spark Streaming and Kafka enables near-instantaneous identification of suspicious transactions, reducing the window for fraudulent activities to go undetected. This ensures rapid response, such as declining unauthorized transactions or alerting customers within seconds.
- Reduced Financial Losses: By flagging fraudulent transactions early through rules like UCL, credit score thresholds, and zip code distance checks, the system minimizes financial losses for both banks and customers. Batch processing further refines detection by identifying patterns that prevent recurring fraud.
- Scalability for High-Volume Transactions: Hadoop's distributed architecture, with HDFS for storage and Spark for processing, ensures the system can handle millions of transactions daily without performance degradation. This scalability supports growing transaction volumes as banking services expand.
- Improved Accuracy and Reliability: The rule-based approach, combined with efficient data storage in HBase and Hive, ensures accurate and interpretable fraud detection. Incremental updates and optimized formats like Parquet enhance reliability by maintaining data consistency and enabling fast queries.
- Cost-Effective Processing: Using Hadoop's commodity hardware-based framework reduces infrastructure costs compared to traditional monolithic systems, while still providing enterprise-grade performance for large-scale data analytics.

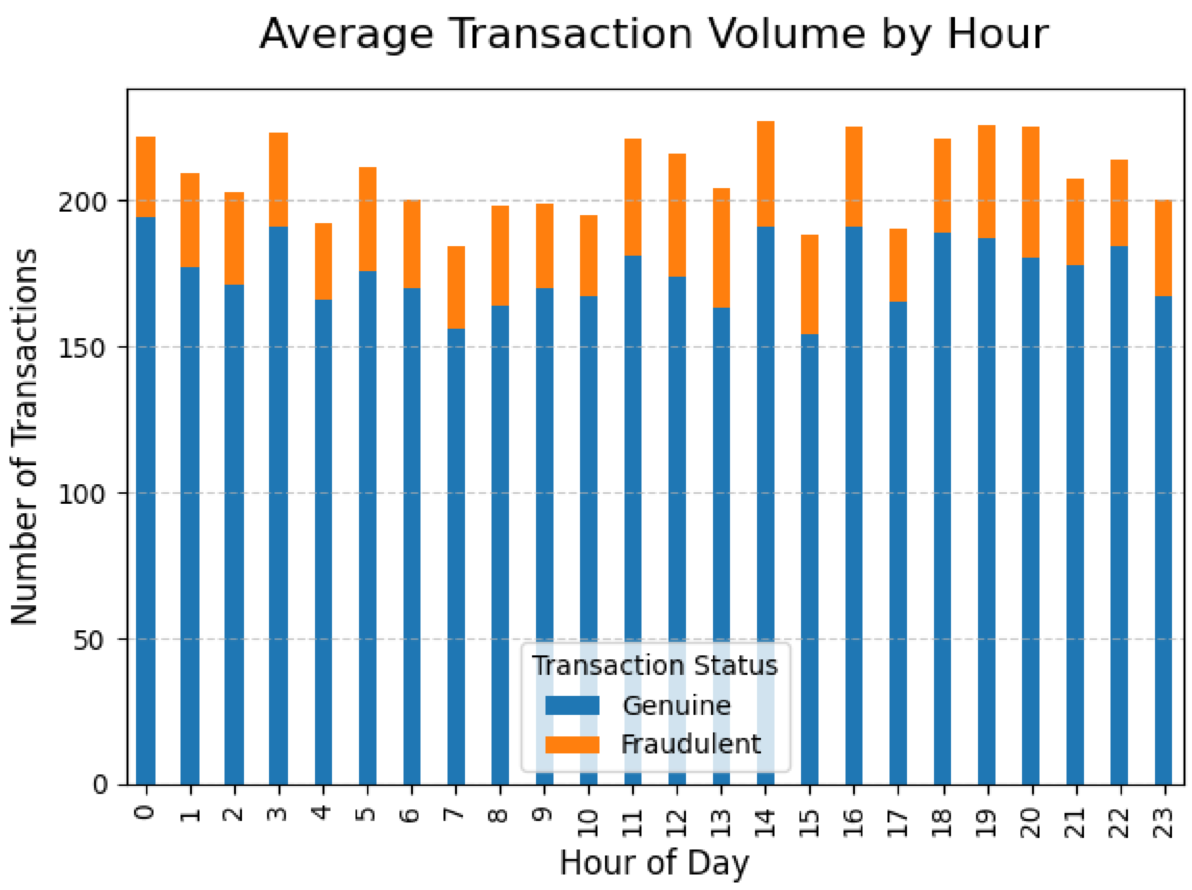

Figure 3: Average Transaction Volume by Hour

Conclusion

The "Fraud Detection in Banking Transactions Using Hadoop" project significantly enhances banking security and operational efficiency by delivering a scalable, high-performance solution for detecting fraudulent transactions. By integrating batch and real-time processing, the system addresses the dual needs of retrospective analysis and immediate fraud prevention, leveraging Hadoop's distributed computing capabilities to process massive datasets efficiently. The use of Sqoop, HDFS, Hive, HBase, Kafka, and Spark Streaming ensures seamless data ingestion, storage, and processing, while rule-based detection methods provide clear, actionable insights. This pipeline reduces financial losses, improves customer trust, and supports the growing demands of modern banking systems. The project's impact lies in its ability to deliver near real-time fraud detection, maintain scalability for high transaction volumes, and offer a cost-effective solution for financial institutions, ultimately strengthening the security and reliability of banking operations.

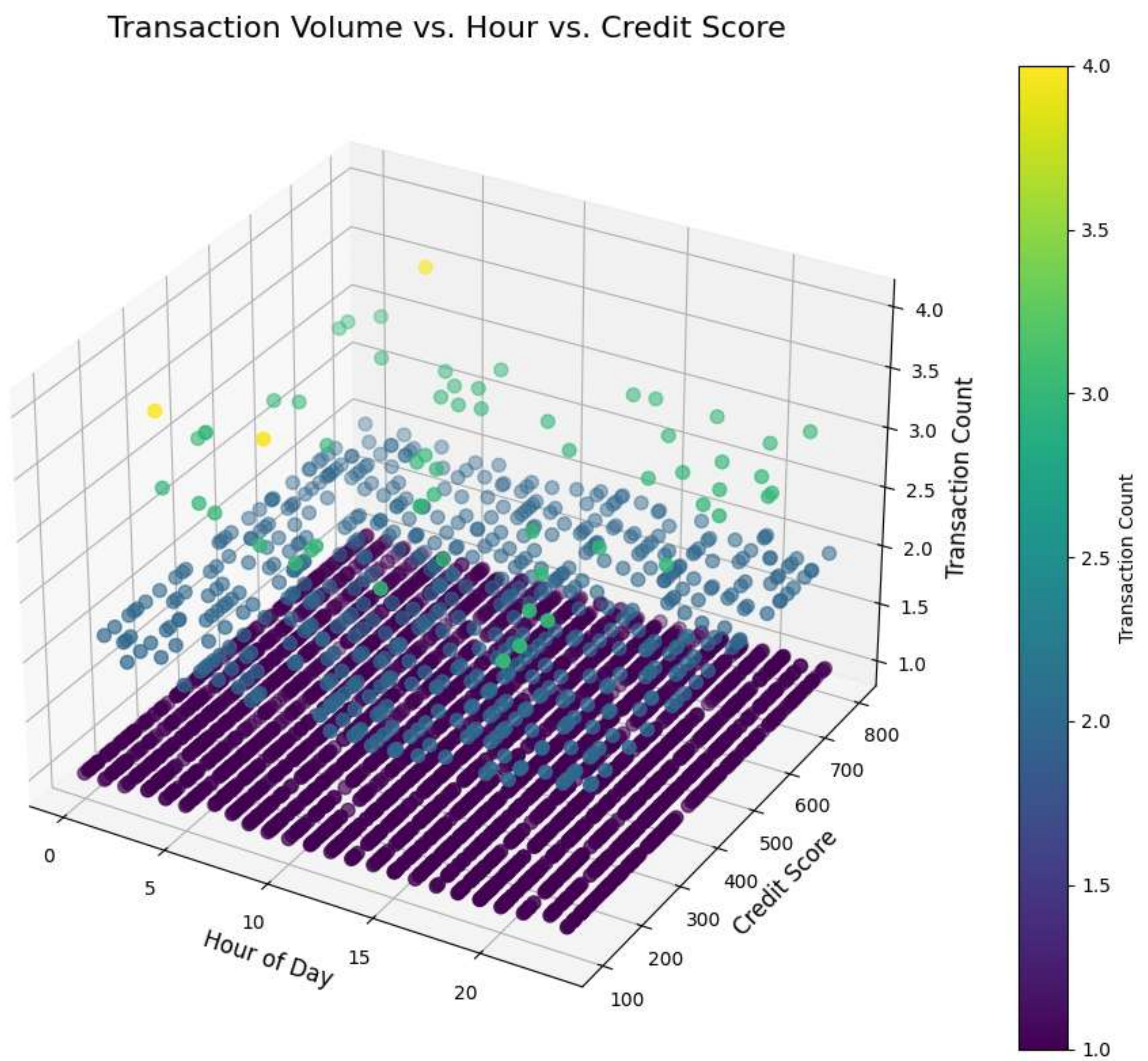

Figure 4: Transaction Volume vs. Hour vs. Credit Score

References

1. Apache Sqoop Documentation. Available at: https://sqoop.apache.org/docs/

2. Apache HBase Reference Guide. Available at: https://hbase.apache.org/book.html

3. Apache Kafka Documentation. Available at: https://kafka.apache.org/documentation/

4. Apache Spark Streaming Guide. Available at: https://spark.apache.org/docs/latest/streaming-programming-guide.html

5. Apache Hive Documentation. Available at: https://hive.apache.org/

6. Nagamangala Sreenivasan, A. (2023). How Hadoop Enhances Fraud Prevention Through Data Analytics. Available at: https://www.accionlabs.com/post/how-hadoop-enhances-fraud-prevention-through-data-analytics

7. Hexanika. (2015). Hadoop in Banking: The Game Changer. Available at: https://hexanika.com/hadoop-in-banking-the-game-changer/

8. Intellipaat. (2023). A Timeline of the Contribution of HADOOP on the Fraud Detection in the Banking Sectors. Available at: https://intellipaat.com/blog/hadoop-fraud-detection-banking/

9. Phung, N. D., Gaber, M. M., & Rohm, U. (2022). Financial Fraud: A Review of Anomaly Detection Techniques and Recent Advances. *Expert Systems with Applications*, 193, 116429. Available at: https://www.sciencedirect.com/science/article/pii/S0957417422000245