Università di Catania

Fraud Detection Pipeline

A Real-time Fraud Detection System
for Financial Transactions

Simone Alfio Brancato

Big Data Final Project

a. y. 2025/26

University of Catania
Master's Degree in Artificial Intelligence and Machine Learning

Contents

1 Introduction and Source Code

The exponential growth of online transactions has led to a corresponding rise in **fraudulent activities**, and the development of intelligent, scalable fraud detection systems has become increasingly important. This project addresses that need by implementing **a real-time fraud detection pipeline** using modern big data technologies.

The primary objective of this work is to design and build a complete, modular, and extensible architecture capable of:

- **Ingesting** transactional data streams from a simulated source
- **Processing** and classifying transactions in real time using machine learning
- **Storing** processed results efficiently for further analysis
- **Real-time dashboards** for visualization and monitoring

By leveraging technologies such as **Apache Kafka**, **Apache Spark**, **ClickHouse**, and **Grafana**, the system is designed to ensure high performance and reliability when handling large volumes of streaming data.

The use of **Docker** facilitated the development process by providing a more manageable, extensible, and reliable environment for deploying and maintaining the system.

This **report** outlines the design decisions, implementation details, and evaluation results of the proposed fraud detection pipeline.

The source code is available at the following link: Fraud-Detection-Pipeline, Github

2 Prerequisites and Setup

To run the project, ensure the following **prerequisites** are met:

- A working installation of `Docker` on your system
- Download the dataset: Credit Card Transaction Fraud Detection Dataset
- Extract the test set file and place it in: `producer/src/main/resources/fraudTest.csv`

Once the dataset is in place, **follow these steps** to launch the project:

- Navigate to the root directory of the project
- To build and start all services run: `docker compose up --buid`
- Open your browser and go to: `http://localhost:3000`
- Log in to Grafana using the default credentials: `admin | admin`
- Access the pre-configured Fraud Detection Dashboard

3 Dataset

The dataset used in this project is the *Credit Card Transaction Fraud Detection Dataset* available on Kaggle. It contains over 1.7 million simulated credit card transactions collected between January 1, 2019, and December 31, 2020. The dataset includes both legitimate and fraudulent operations and is designed for developing and evaluating machine learning models for fraud detection.

The simulation represents transactions carried out by 1,000 customers across a pool of 800 merchants. It was generated using the open-source tool *Sparkov Data Generation, created by Brandon Harris*. This tool utilizes the `faker` Python library to synthetically generate realistic transaction profiles based on predefined customer demographics and transaction patterns. Notice that the dataset **is strongly imbalanced**, with only less than 0.5% transactions that are classified as fraud, just like a real scenario.

Table 1: Dataset Features

Feature	**Type**	**Description**
`index`	Integer	Row index
`trans_date_trans_time`	Datetime	Timestamp of the transaction
`cc_num`	Integer	Credit card number
`merchant`	String	Merchant name
`category`	String	Merchant category
`amt`	Float	Transaction amount
`first`	String	Customer's first name
`last`	String	Customer's last name
`gender`	Categorical (M/F)	Customer gender
`street`	String	Street address
`city`	String	City of residence
`state`	String	US state abbreviation
`zip`	Integer	Postal code
`lat`	Float	Latitude of residence
`long`	Float	Longitude of residence
`city_pop`	Integer	Population of the city
`job`	String	Customer's occupation
`dob`	Date	Date of birth
`trans_num`	String	Unique transaction ID
`unix_time`	Integer	Unix timestamp of the transaction
`merch_lat`	Float	Latitude of the merchant
`merch_long`	Float	Longitude of the merchant
`is_fraud`	Boolean (0/1)	Indicates if the transaction is fraudulent

In this project, the training set is used to train the fraud detection model, while the test set is employed to evaluate its performance. Additionally, the same test set is ingested into the fraud detection data pipeline to simulate **a realistic use case within an industrial environment**. We adopt the train-test split provided on the Kaggle dataset page, utilizing over 1.2 million records for training and more than 500,000 records for testing.

4 Fraud Detection Model

This section presents a comprehensive approach to set up a fraud detection model, including feature engineering, XGBoost classifier training, and performance evaluation on the train set and the test set.

4.1 Feature Engineering

The feature engineering phase constitutes a critical component of the fraud detection pipeline, transforming raw transactions data into meaningful predictive features. The preprocessing function implements several key transformations to enhance the model's ability to detect fraudulent patterns.

Feature	**Type**	**Description**
`age`	Integer	Customer age computed from date of birth
`hour`	Integer	Hour of day when transaction occurs
`day_of_week`	Integer	Day of week (0=Monday, 6=Sunday)
`is_night`	Boolean (0/1)	Binary flag for transactions between 22:00-06:00
`is_weekend`	Boolean (0/1)	Binary flag for weekend transactions
`log_distance`	Float	Logarithmic version of Haversine distance
`log_amt`	Float	Logarithmic version of the transaction amount
`log_city_pop`	Float	Logarithmic version of the city population
`tx_count_user`	Integer	Cumulative transactions count per user
`amt_mean_user`	Float	Transactions window average of amounts per user
`gender`	Categorical (M/F)	Customer gender
`category`	Categorical	Transaction category
`state`	Categorical	US state abbreviation
`job`	Categorical	Customer occupation

Table 2: Additional Engineered Features for Fraud Detection Model

The feature engineering phase contributed to **improve the model performances** on both the training and test sets. It enhanced the model's ability to generalize and effectively handle skewed features through techniques such as logarithmic transformations.

4.2 Training an XGBoost Machine Learning Model

The model training phase employs **XGBoost (eXtreme Gradient Boosting)**, a powerful machine learning algorithm particularly effective for imbalanced classification tasks such as fraud detection.

The XGBoost classifier is configured with **logistic loss** as the evaluation metric, which is appropriate for binary classification problems.

The algorithm builds a collection of decision trees using gradient boosting, where each subsequent tree corrects the errors of the previous trees in the collection.

To define the **hyperparameters** we implemented a **grid search strategy** with **cross-validation** using k-folds. After the search, we met the optimal results with the following setting shown in the following table:

Hyperparameter	**Value**	**Description**
`n_estimators`	200	Number of boosting rounds
`max_depth`	12	Maximum tree depth
`learning_rate`	0.2	Step size shrinkage
`subsample`	0.8	Fraction of samples used for tree training
`colsample_bytree`	1.0	Fraction of features used per tree
`scale_pos_weight`	10	Balancing factor for positive class
`reg_alpha`	1	L1 regularization term
`reg_lambda`	10	L2 regularization term

Table 3: Optimal XGBoost Hyperparameters

The optimization criterion is **recall score**, prioritizing the detection of fraudulent transactions over false positive minimization.

4.3 Evaluations and Results

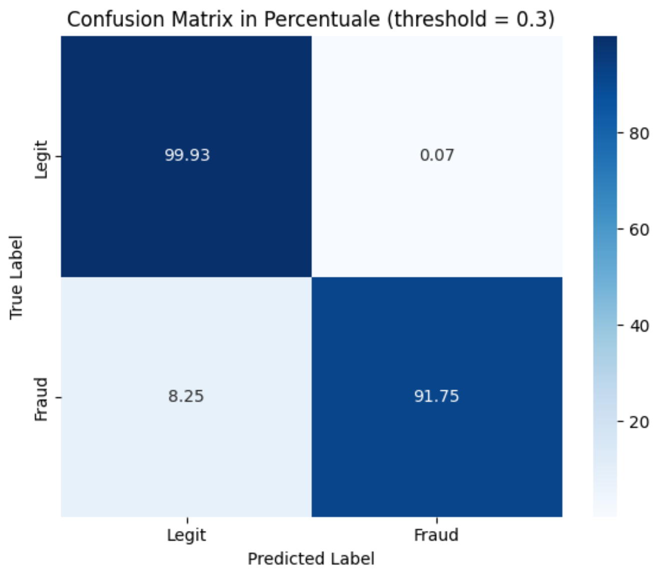
The model evaluation consists in various performance analysis using multiple metrics and **custom threshold optimization** to balance precision and recall according to business requirements. In fact, rather than using the default 0.5 classification threshold, a custom threshold of 0.3 is applied to the predicted probabilities.

This adjustment **increases sensitivity to fraud detection**, prioritizing recall over precision in accordance with the typical cost structure of fraud detection systems where false negatives are more costly than false positives.

The model demonstrates **exceptional performances** on both training and test datasets. This can be observed through the precision, recall and F1-score metrics, as discussed in the following table.

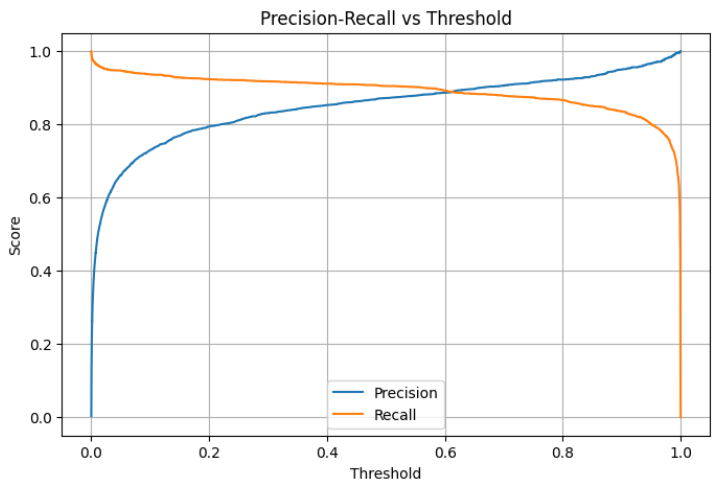
Dataset	**Class**	**Precision**	**Recall**	**F1-Score**	**Support**
Training	Legitimate	1.00	1.00	1.00	1,289,169
	Fraud	0.99	1.00	1.00	7,506
Test	Legitimate	1.00	1.00	1.00	553,574
	Fraud	0.83	0.92	0.87	2,145

Table 4: Model Performance on Training and Test Sets



The **confusion matrix** of the test set reveals the model's classification performance in detail. The model achieves:

- **True Negatives:** Nearly 100% of legitimate transactions correctly classified
- **True Positives:** 92% of fraudulent transactions correctly identified
- **False Positives:** Minimal misclassification of legitimate transactions as fraud
- **False Negatives:** 8% of fraudulent transactions missed



The comparison between training and test performance indicates **good generalization capability**. While the training set shows near-perfect performance, the test set results demonstrate robust real-world applicability with high recall for fraud detection and acceptable precision minimizing false alarms when it is possible.

The model's performance translates to **significant business value**, since 92% of fraudulent transactions are detected, and only 17% of fraud alerts are false positives. These results can be further optimized by adjusting the classification threshold for the specific operational costs and business requirements of different organizations.

5 Fraud Detection Pipeline Architecture

This section discusses the architecture designed for real-time fraud detection in production environments. The pipeline leverages modern distributed computing technologies to handle high-throughput transaction streams while maintaining low-latency detection capabilities.

5.1 Transactions Data Producer

The transaction data producer serves as the initial component of the fraud detection pipeline, responsible for **ingesting financial transaction data** from the test dataset and streaming it to the Apache Kafka cluster. This component has been implemented as a Java application that simulates real-time transaction flow by reading from a CSV file and publishing each transaction as a message to the Kafka Topic.

The Kafka Producer is configured with minimal yet essential properties, maintaining the flexibility required for different deployment scenarios.

```
KAFKA_BOOTSTRAP_SERVERS = "kafka1:9092,kafka2:9092,kafka3:9092"
KAFKA_TOPIC = "fraud-transactions"
TEST_DATA_PATH = "fraudTest.csv"
```

Once the properties are set up, it reads the transactions test set using Java NIO file operations:

```
Path path = Paths.get(TEST_DATA_PATH);
try (BufferedReader reader = Files.newBufferedReader(path)) {
    reader.readLine(); // Skip the first line (header)

    int i = 0;
    String line;
    while ((line = reader.readLine()) != null) {
        String key = "key-" + i;
        i++;
        producer.send(new ProducerRecord<>(KAFKA_TOPIC,key,line));
        Thread.sleep(ThreadLocalRandom.current().nextInt(50, 300));
    }
}
```

A critical aspect of the producer implementation is the **simulation of real-time transaction flow**. The application uses `ThreadLocalRandom` to introduce variable delays between message publications. The **random delays** simulate the irregular nature of real-world transaction patterns.

5.2 Data Streaming with Apache Kafka

The data streaming infrastructure of the fraud detection pipeline is built upon an **Apache Kafka cluster** with three nodes that provides high-throughput, fault-tolerant message streaming capabilities.

The cluster consists of the following components:

- **Kafka Brokers**: `kafka1`, `kafka2`, and `kafka3`
- **Kafka UI**: interface for cluster monitoring at `http://localhost:8080`
- **Topic Initialization Service**: Automated topic creation and configuration

The cluster utilizes **Kafka's KRaft mode**, which represents a significant architectural improvement over traditional ZooKeeper-based deployments and reducing network overhead and latency. Each broker is configured to serve dual roles as both broker and controller:

```yaml
KAFKA_ENABLE_KRAFT: yes
KAFKA_CFG_PROCESS_ROLES: broker,controller
KAFKA_CFG_CONTROLLER_LISTENER_NAMES: CONTROLLER
```

Each Kafka broker utilizes dedicated persistent volumes to ensure data durability and enable recovery from container restarts. Now let's see a comprehensive configuration for Broker 1, as an example:

```yaml
kafka_1:
  image: bitnami/kafka:3.7.0
  container_name: kafka1
  ports:
    - 9092:9092
  environment:
    KAFKA_ENABLE_KRAFT: yes
    KAFKA_CFG_NODE_ID: 1
    KAFKA_CFG_PROCESS_ROLES: broker,controller
    KAFKA_CFG_CONTROLLER_LISTENER_NAMES: CONTROLLER
    KAFKA_CFG_LISTENERS: PLAINTEXT://:9092,CONTROLLER://:19092
    KAFKA_CFG_ADVERTISED_LISTENERS: PLAINTEXT://kafka1:9092
    KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP: CONTROLLER:PLAINTEXT,
 PLAINTEXT:PLAINTEXT
    KAFKA_CFG_CONTROLLER_QUORUM_VOTERS: 1@kafka1:19092,2@kafka2
 :19093,3@kafka3:19094
    KAFKA_KRAFT_CLUSTER_ID: Nru2U1JQQT-RKA0UapBN3w
    ALLOW_PLAINTEXT_LISTENER: yes
  volumes:
    - kafka1_data:/bitnami/kafka
```

The three-node cluster configuration provides robust **high availability** and **fault tolerance** capabilities. The cluster supports configurable **replication factor** for topics, ensuring data redundancy across multiple brokers, and **data partitions** for handle parallel processing.

The `fraud-transactions` topic is configured with three partitions and a replication factor of **three** to ensure data durability and maximize parallelism across our three-node Kafka and Spark clusters. Further details will be discussed in the next chapter.

5.3 Data Processing with Apache Spark

The fraud detection system leverages Apache Spark's distributed computing capabilities through a containerized cluster architecture. The Spark cluster consists of one master node and three worker nodes, each equipped with GPU resources for enhanced computational performance.

- **Spark Master**: Coordinates job execution and resource management, exposing the Spark UI on port 8080 and accepting worker connections on port 7077
- **Spark Workers**: Three identical worker nodes, each with 3GB memory allocation and single-core processing capability

The cluster utilizes the **Bitnami Spark 3.5** image, which provides a production-ready Spark environment with optimized configurations for containerized deployments. It consumes transaction data from Kafka Topic and applies the trained machine learning models for fraud detection.

The **feature engineering** process transforms raw transaction data into ML-ready features through a comprehensive preprocessing pipeline. Since we are using the same model trained in the previous chapter, we will calculate the same additional engineered features.

Feature	**Type**	**Description**
`age`	Integer	Customer age computed from date of birth
`hour`	Integer	Hour of day when transaction occurs
`day_of_week`	Integer	Day of week (0=Monday, 6=Sunday)
`is_night`	Boolean (0/1)	Binary flag for transactions between 22:00-06:00
`is_weekend`	Boolean (0/1)	Binary flag for weekend transactions
`log_distance`	Float	Logarithmic version of Haversine distance
`log_amt`	Float	Logarithmic version of the transaction amount
`log_city_pop`	Float	Logarithmic version of the city population
`tx_count_user`	Integer	Cumulative transactions count per user
`amt_mean_user`	Float	Transactions window average of amounts per user
`gender`	Categorical (M/F)	Customer gender
`category`	Categorical	Transaction category
`state`	Categorical	US state abbreviation
`job`	Categorical	Customer occupation

Table 5: Additional Engineered Features for Fraud Detection Model

The trained XGBoost model pipeline is loaded using `joblib` and applied to processed feature vectors:

```python
    threshold = 0.3
    df['fraud_prob'] = model_pipeline.predict_proba(processed_df)[:, 1]
    df['fraud_pred'] = (df['fraud_prob'] > threshold).astype(int)
```

Processed results are persisted to **ClickHouse**, a columnar database optimized for analytical workloads. The table for storing data will be created by Spark.

5.4 Data Storage with ClickHouse

ClickHouse serves as the primary database, optimized for high-performance queries on large volumes of transaction and prediction data. The column-oriented architecture provides high performance for fraud analytics and reporting. To keep the architecture simple, we implement a single table with every transaction along with its prediction.

```sql
CREATE TABLE IF NOT EXISTS fraud_predictions (
    idx Int32,
    trans_date_trans_time DateTime,
    cc_num String,
    merchant String,
    category String,
    amt Float64,
    first String,
    last String,
    gender String,
    street String,
    city String,
    state String,
    zip String,
    lat Float64,
    long Float64,
    city_pop Int64,
    job String,
    dob Date32,
    trans_num String,
    unix_time Int64,
    merch_lat Float64,
    merch_long Float64,
    age Int64,
    hour Int64,
    day_of_week Int32,
    is_night UInt8,
    is_weekend UInt8,
    distance_user_to_merch Float64,
    log_amt Float64,
    log_city_pop Float64,
    log_distance Float64,
    user_id String,
    tx_count_user Int32,
    amt_mean_user Float64,
    fraud_probability Float64,
    fraud_prediction Int32,
    inserted_at DateTime DEFAULT now()
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(trans_date_trans_time)
PRIMARY KEY (trans_date_trans_time, idx)
ORDER BY (trans_date_trans_time, idx)
SETTINGS index_granularity = 8192;
```

5.5 Data Visualization with Grafana

The implementation of effective data visualization is crucial for monitoring fraud detection systems. **Grafana** provides a robust solution for creating interactive dashboards that can perfectly integrate with ClickHouse, maintaining minimal operational overhead while delivering optimal analytical capabilities.

```yaml
grafana_service:
  build:
    context: ./grafana
    dockerfile: Dockerfile
  container_name: grafana
  ports:
    - 3000:3000
  environment:
    GF_SECURITY_ADMIN_USER: admin
    GF_SECURITY_ADMIN_PASSWORD: admin
  volumes:
    - grafana_data:/var/lib/grafana
    - ./grafana/provisioning:/etc/grafana/provisioning
  depends_on:
    - clickhouse
```

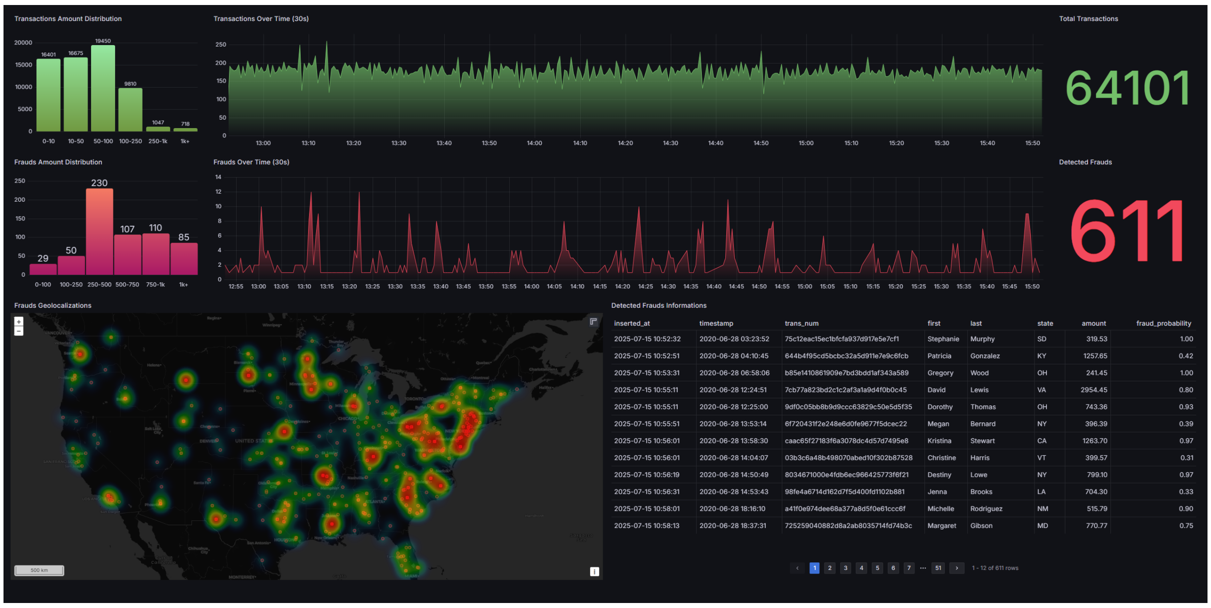
The **connection between Grafana and ClickHouse** is established through the provisioning directory, which contains both the dashboard configurations and the data source definitions. The ClickHouse data source configuration is defined as follows:

```yaml
apiVersion: 1

datasources:
  - name: ClickHouse
    type: vertamedia-clickhouse-datasource
    access: proxy
    url: http://clickhouse:8123
    basicAuth: true
    basicAuthUser: default
    basicAuthPassword: password
    withCredentials: true
    isDefault: true
    editable: true
    jsonData:
      defaultDatabase: default
      addCorsHeader: false
      usePOST: false
    secureJsonData:
      basicAuthPassword: password
    version: 1
    readOnly: false
```

This configuration establishes a secure connection to the ClickHouse database using basic authentication and proxy access mode. The data source is configured as the default connection, enabling immediate availability for data analysis along with the preconfigured dashboard.

The implemented **Grafana dashboard** provides comprehensive monitoring capabilities for the fraud detection system, presenting multiple analytical visualizations carefully designed. The dashboard architecture incorporates real-time data streams and historical analysis to support both operational monitoring and strategic decision-making processes. You can check it accessing the Grafana UI as discussed in the second chapter `Prerequisites and Setup`.



inserted_at	timestamp	trans_num	first	last	state	amount	fraud_probability
2025-07-15 10:52:32	2020-06-28 03:23:52	75c12eac15ec1bfcfa937d917e5e7cf1	Stephanie	Murphy	SD	319.53	1.00
2025-07-15 10:52:51	2020-06-28 04:10:45	644b4f95cd5bcbc32a5d911e7e9c6fcb	Patricia	Gonzalez	KY	1257.65	0.42
2025-07-15 10:53:31	2020-06-28 06:58:06	b85e1410861909e7bd3bdd1af343a589	Gregory	Wood	OH	241.45	1.00
2025-07-15 10:55:11	2020-06-28 12:24:51	7cb77a823bd2c1c2af3a1a9d4f0b0c45	David	Lewis	VA	2954.45	0.80
2025-07-15 10:55:11	2020-06-28 12:25:00	9df0c05bb8b9d9ccc63829c50e5d5f35	Dorothy	Thomas	OH	743.36	0.93
2025-07-15 10:55:51	2020-06-28 13:53:14	6f720431f2e248e6d0fe9677f5dcec22	Megan	Bernard	NY	396.39	0.39
2025-07-15 10:56:01	2020-06-28 13:58:30	caac65f27183f6a3078dc4d57d7495e8	Kristina	Stewart	CA	1263.70	0.97
2025-07-15 10:56:01	2020-06-28 14:04:07	03b3c6a48b498070abed10f302b87528	Christine	Harris	VT	399.57	0.31
2025-07-15 10:56:19	2020-06-28 14:50:49	8034671000e4fdb6ec966425773f6f21	Destiny	Lowe	NY	799.10	0.97
2025-07-15 10:56:31	2020-06-28 14:53:43	98fe4a6714d162d7f5d400fd1102b881	Jenna	Brooks	LA	704.30	0.33
2025-07-15 10:58:01	2020-06-28 18:16:10	a41f0e974dee68a377a8d5f0e61ccc6f	Michelle	Rodriguez	NM	515.79	0.90
2025-07-15 10:58:13	2020-06-28 18:37:31	725259040882d8a2ab8035714fd74b3c	Margaret	Gibson	MD	770.77	0.75

1 - 12 of 611 rows

The upper section displays **transaction volume metrics** through two complementary visualizations. The `Transactions Amount Distribution` presents a histogram showing the distribution of transaction amounts across predefined ranges, enabling analysts to understand transaction patterns. Adjacent to this, the `Transactions Over Time` time series chart tracks transaction frequency over a 30-second interval, providing real-time insights into transaction velocity patterns.

The **fraud analytics** are presented through parallel visualizations that mirror the transaction analytics structure. The `Frauds Amount Distribution` histogram reveals the distribution of fraudulent transaction amounts. The `Frauds Over Time` time series visualization tracks fraud detection events with 30-second granularity, facilitating real-time monitoring of fraud patterns and system effectiveness.

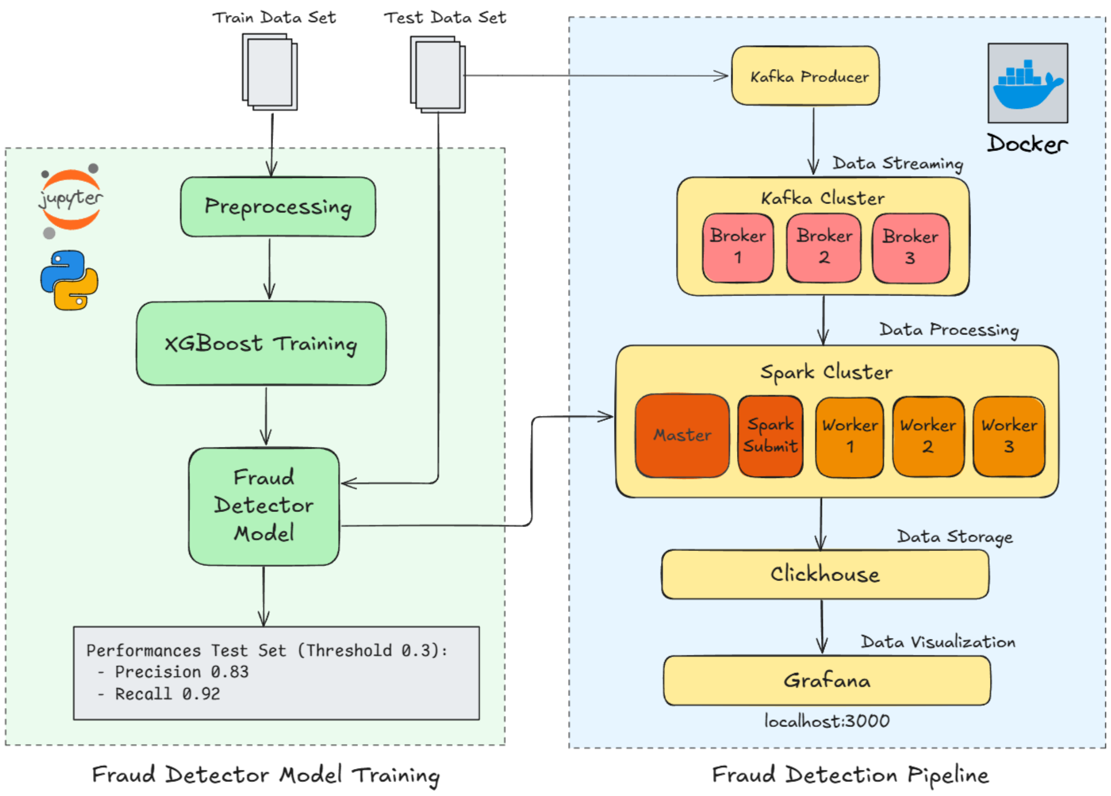
Along with the two time series graphs concerning the transactions and frauds volume over the time, we can also check a **count** for both of them with `Total Transactions` and `Detected Frauds` providing immediate visibility of the fraud rates for the considered time range.

The `Frauds Geolocalisations` heatmap visualization presents a **geographical representation of fraud incidents** across different regions, utilizing color intensity to indicate fraud concentration levels. This spatial analysis capability enables identification of geographical fraud patterns.

In the end, the `Detected Frauds Informations` table provides **granular visibility into individual fraud detection events**, displaying transaction details including timestamps, transaction numbers, customer information, geographical data, transaction amounts, and fraud probability scores, enableing manual verification of automated detection results.

6 Project Architecture Overview

The image below illustrates the complete structure of the project, including the model training phase and the data pipeline.



```
Performances Test Set (Threshold 0.3):
 - Precision 0.83
 - Recall 0.92
```

7 Future Work

The current implementation of the fraud detection data engineering pipeline establishes a solid foundation for real-time transaction monitoring and analysis. However, several areas present opportunities for improve system performance and analytical capabilities.

- **Advanced Machine Learning Integration:** enhance the current system with more advanced ML algorithms and implement larger and real production dataset.
- **Advanced Analytics**: integrate NLP tools for automated report generation and smart alert summarization.
- **Security Enhancements**: enforce end-to-end encryption for sensitive data.
- **Real-Time Alerting**: integrate with external communication systems.
- **Integration with External Data Sources**: integrations with banking platforms for real-time fraud blocking.

These future enhancements would transform the current fraud detection system from a reactive monitoring tool into a real intelligent fraud prevention and production-ready platform.