Warsaw University of Technology

Faculty of Mathematics and Information Science

# Real-time fraudulent transactions detection

Big Data Analytics

Salveen Singh Dutt (317298)
Karina Tiurina (335943)
Patryk Prusak (305794)

Supervisor
mgr inz. Jakub Abelski

Warsaw 2025

# Contents

# Introduction

This project aims to plan and implement a financial transaction processing system that identifies suspicious and fraudulent activity in real time. Given the enormous volume of incoming data, the project utilizes big data technologies and advanced machine learning algorithms for anomaly detection. The project is available on the GitHub repository: https://github.com/salveendutt/Big-Data-Analytics.

## Updates from Milestone 3

The serving layer has been introduced with Apache Superset and TrinoDB. The initial model is now being trained locally with Spark.

# 1 High level description

The main idea is to implement automatic transaction processing so that the transaction is blocked for further manual review when a fraudulent activity occurs. The aim is to reduce the end-users financial losses and enhance online payment security, ensuring a safer experience for all customers.

The project's main end-users are financial institutions (we will call them 'Managers') and their customers who are executing the payments. Although both categories can benefit from the solution, in our implementation, we will mainly focus on Managers to limit additional data in storage.

The list below contains the main features that we expect to implement for Managers:

1. Fraudulent transactions are automatically highlighted so that it is easier to identify suspicious activity;

2. The history of transactions is stored and available for later review;

3. A dashboard with statistics of fraudulent activity is available and customizable for better localization of issues (e.g., too large amount, unusual location);

4. Anomaly-detection model is continuously updated so that fraud detection utilizes new historical data and is more accurate on future transactions;

5. Data streaming processing and batch jobs are customizable so that the testing of the model's performance is simplified.

## 2 Data sources

Due to strict security regulations on personal and financial data, finding open-source actual transaction data for model training and streaming is challenging. Therefore, available synthetic and anonymized datasets will be used. The table below contains a description of the data sources. Each data source is described in more detail in the dedicated subsections.

| Data Source | Content | Volume | Fraud, % | Link |
|---|---|---|---|---|
| 1. Fraudulent Transactions Data | Dataset for predicting fraudulent transactions for a financial company. | 6,362,620 rows and 10 columns (493.53 MB) | 0.13% | Kaggle |
| 2. Credit Card Fraud | Contains features with transactional context. | 1,000,000 transactions (58.9 MB) | 8.7% | OpenML |
| 3. Credit Card Transactions Synthetic Data Generation | A collection of synthetic credit card transaction data. | 1,785,308 transactions; 5,000 customers; (153.66 MB) | 3% | Kaggle |
| 4. Credit Card Fraud Detection | Transactions made by credit cards in September 2013 by European cardholders. | 284,807 transactions (150.83 MB) | 0.17% | Kaggle |

Table 1: Data sources

Data-streaming API was implemented from scratch. The assumption was that it would use the above datasets; with a specified time frame, it would choose a random transaction not used for training and push it for further processing. The probability of a fraudulent transaction will be set manually to some high enough constant value for testing purposes. A detailed description of the implemented streaming API can be found in Chapter 3.

The content of the datasets is exceptionally different, which makes it impossible to combine them into a single dataset. Therefore, each dataset will be treated separately for streaming and ML training.

Complete Exploratory Data Analysis can be found in the 'eda' subfolder of our repository. In the report we included fraud distribution and the amount distribution for each dataset where applicable.

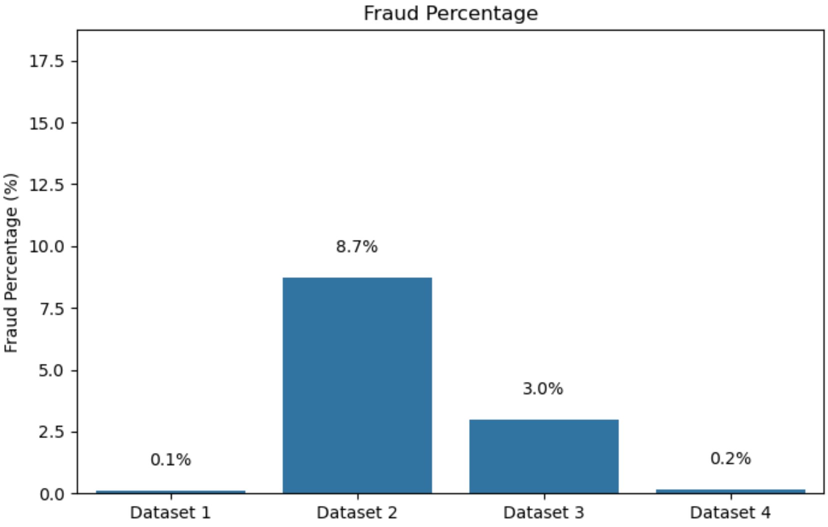


Figure 1: Fraud distribution across datasets

## 2.1 Fraudulent Transactions Data (Kaggle)

The dataset contains transactions for a financial company, indicating whether they are fraudulent. Data for the case is available in CSV format, which has 6362620 rows and ten columns. The entire column description is presented in Table 2.

The following transformation should be done to pass the dataset to the ML models:

1. 'type' column values CASH-IN, CASH-OUT, DEBIT, PAYMENT, and TRANSFER transformed to 1, 2, 3, 4, and 5, respectively;

2. a new attribute 'isMerchant' is calculated. 1 if 'nameDest' starts with 'M', 0 - otherwise.

Generally, the dataset and clean and structured, no other preprocessing, except for the described above is necessary. The potential issue is that it is highly unbalanced (less than 1% of fraud transactions) and contains quite a small number of features (6) for model training.

## 2.2 Credit Card Fraud (OpenML)

This dataset captures transaction patterns and behaviors that could indicate potential fraud in card transactions. The data comprises several features that reflect the transactional context, such as geographical location, transaction medium, and spending behavior relative to the user's history.

The whole dataset is in the numeric form. Therefore, no additional preprocessing is necessary. The only transformation is to rename the target variable to 'isFraud' to match the format of other datasets. The target feature balance is much better than the previous dataset: 8.7% of fraud. An additional complication is that the feature 'amount' is not available. A ratio to the median purchase of the same customer is provided instead.

| Column | Content | Type |
|---|---|---|
| step | maps a unit of time in the real world. In this case, 1 step is 1 hour. Total steps 744 (30 days simulation) | int |
| type | type of the transaction. Available values: CASH-IN, CASH-OUT, DEBIT, PAYMENT, and TRANSFER | str |
| amount | amount of the transaction in local currency | float |
| nameOrig | customer who started the transaction | str |
| oldbalanceOrg | initial balance before the transaction | float |
| newbalanceOrig | new balance after the transaction | float |
| nameDest | customer who is the recipient of the transaction | str |
| oldbalanceDest | initial balance recipient before the transaction. Note that there is no information for customers that start with M (Merchants) | float |
| newbalanceDest | new balance recipient after the transaction. Note that there is no information for customers that start with M (Merchants) | float |
| isFraud | these are the transactions made by the fraudulent agents inside the simulation. In this specific dataset, the fraudulent behavior of the agents aims to profit by taking control of customers' accounts and trying to empty the funds by transferring them to another account and then cashing out of the system | int |
| isFlaggedFraud | the business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction | int |

Table 2: Columns description. Dataset 1: 'Fraudulent Transactions Data' from Kaggle

## 2.3 Credit Card Transactions Synthetic Data Generation (Kaggle)

This dataset is a collection of synthetic credit card transaction data. The data is designed to mimic the characteristics of actual credit card transactions while ensuring privacy and compliance with data protection regulations such as the General Data Protection Regulation (GDPR). It contains 1,785,308 transactions for 5000 customers.

Like other datasets, it is quite unbalanced, with 3% of fraudulent transactions. The following preprocessing should be done before passing rows to the ML task:

1. 'entry_mode' column values Contactless, Chip, and Swipe transformed to 1, 2, and 3, respectively;

2. 'amt' renamed to 'amount',

3. 'fraud' renamed to 'isFraud',

4. transaction data itself contains only customer ID. Therefore, an additional step is to find the customer and add related features to the output.

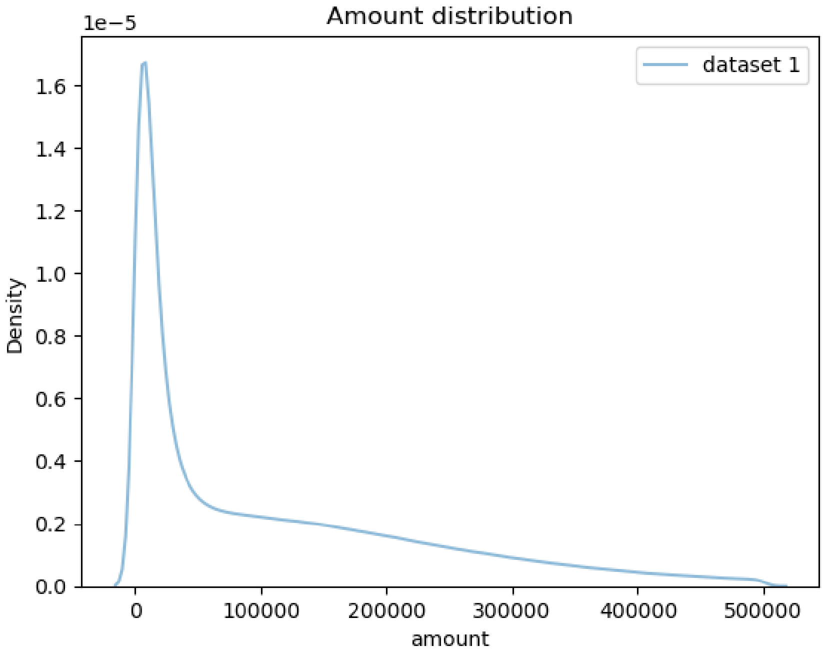


Figure 2: Amount distribution of dataset 1

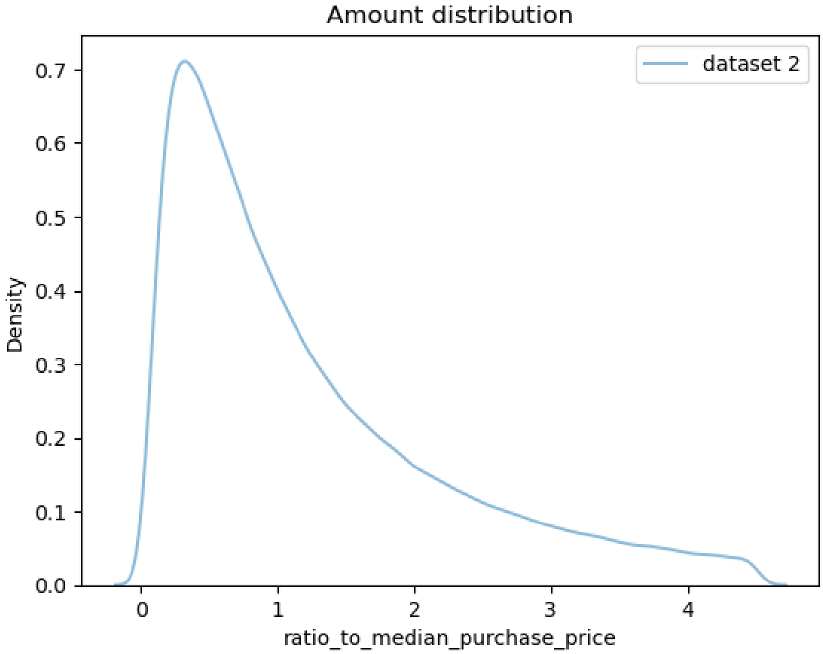


Figure 3: Amount distribution of dataset 2

## 2.4 Credit Card Fraud Detection (Kaggle)

The dataset contains transactions made by credit cards in September 2013 by European cardholders. It includes 284,807 transactions, with only 492 (less than 1%) of fraud ones. Due to confidentiality issues, the dataset contains only numerical input variables resulting from a PCA transformation.

This dataset contains a relatively small amount of data compared to others (about

| Column | Content | Type |
|---|---|---|
| distance_from_home | This is a numerical feature representing the geographical distance in kilometers between the transaction location and the cardholder's home address. | float |
| distance_from_last_transaction | This numerical attribute measures the distance in kilometers from the location of the last transaction to the current transaction location. | float |
| ratio_to_median_purchase_price | A numeric ratio that compares the transaction's price to the median purchase price of the user's transaction history. | float |
| repeat_retailer | A binary attribute where '1' signifies that the transaction was conducted at a retailer previously used by the cardholder, and '0' indicates a new retailer. | [0, 1] |
| used_chip | This binary feature indicates whether the transaction was made using a chip (1) or not (0). | [0, 1] |
| used_pin_number | Another binary feature, where '1' signifies the use of a PIN number for the transaction, and '0' shows no PIN number was used. | [0, 1] |
| online_order | This attribute identifies whether the purchase was made online ('1') or offline ('0'). | [0, 1] |
| fraud | A binary target variable indicating whether the transaction was fraudulent ('1') or not ('0'). | [0, 1] |

Table 3: Columns description. Dataset 2: 'Credit_Card_Fraud_' from OpenML

280k transactions). Therefore, we plan to keep it as an additional dataset in case of any issues with others.

Since the data has already been processed, the only necessary transormation is to rename columns 'Amount' and 'Class' to 'amount', 'isFraud' respectively.

| Column | Content | Type |
|---|---|---|
| transaction_id | Random string containing specific transactions id | str |
| post_ts | Date and time of the transaction | str |
| customer_id | Specific customer id | str |
| bin | Bank Identification Number | int |
| terminal_id | Specific terminal id | int |
| amt | Transaction amount | float |
| entry_mode | Mode of the transaction. Possible values are Contactless, Chip, and Swipe. | str |
| fraud | Target variable containing 1 for the fraudulent transaction and 0 otherwise | int |
| fraud_scenario | Additional label for the transaction. 97% of the dataset has a value of 0. No specific description for each scenario is provided. | int, [0, 1, 2] |
| mean_amount | Average transaction amount for a specific customer | float |
| std_amount | Standard deviation of the transaction amount for a specific customer | float |
| mean_nb_tx_per_day | Mean number of transactions per day for a specific customer | float |
| customer_bin | Bank Identification Number of a customer | int |

Table 4: Columns description. Dataset 3: 'Credit Card Transactions Synthetic Data Generation' from Kaggle

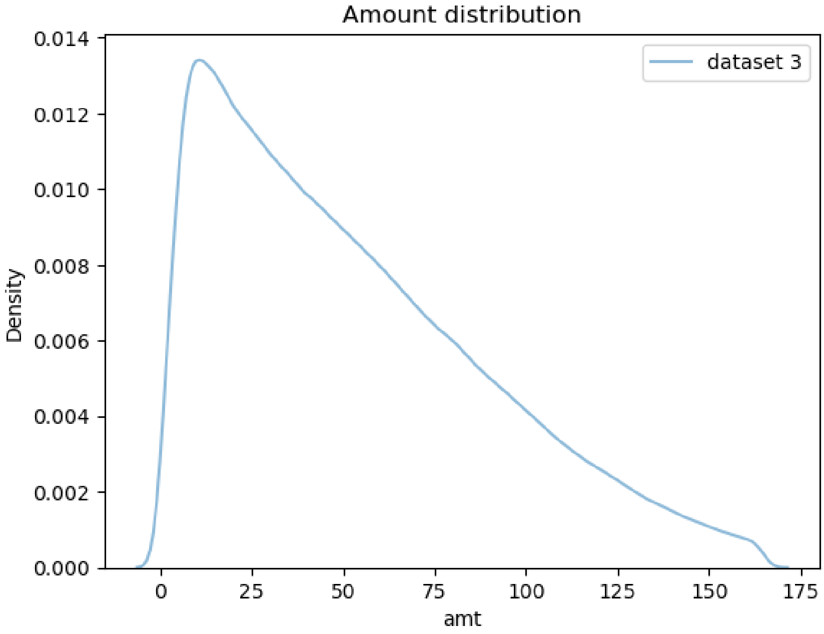


Figure 4: Amount distribution of dataset 3

| Column | Content | Type |
|---|---|---|
| Time | The seconds elapsed between the transaction and the first transaction in the dataset | int |
| V1 ... V28 | The principal components obtained with PCA. The original features and more background information about the data are not provided. | float |
| Amount | Transaction amount | float |
| Class | Target variable; 1 for fraudulent transaction and 0 otherwise | int, [0, 1] |

Table 5: Columns description. Dataset 4: 'Credit Card Fraud Detection' from Kaggle

## 3 Data acquisition strategy

Due to the lack of publicly available open streaming APIs, we designed and implemented a custom streaming API for real-time fraud detection. Stream API is connected to the NiFi for further data collection and preprocessing. The following technological stack is used for the data acquisition:

1. Python,
2. Flask - for the stream API,
3. Apache NiFi - for the data collection,
4. Docker - for deployment,
5. Apache Kafka - for the data streaming,
6. Kafdrop - for Kafka monitoring,
7. Apache HDFS and Hive - as a main storage solution
8. Zookeeper.

When the server is started, stream API is available on localhost:5001/data/:dataset_id. Data frequency is configurable on the NiFi side and, at the moment, is set to 1 row per 2 seconds for each dataset. The format of each incoming transaction is JSON, containing attributes as described for each dataset in Chapter 2. The streaming API randomly (10%) returns a 500 error to simulate real-world conditions. Example screenshots of the data stream and NiFi are provided in the SKaMP_Tests.pdf file.

Below, we present the NiFi flows for the data acquisition. Processor failures are retried 10 times, and errors are logged with the LogMessage processor with a level of 'error.' The first flow is the overview of the whole process. The three processor groups are responsible for fetching and processing the data. Inside each of the processor groups, there are:

- InvokeHTTP - to fetch the data from the stream API,

- EvaluateJsonPath - to evaluate whether any data is missing,
- ReplaceText - to add year, month, and day that is used for data partitioning in Hive,
- PublishKafkaRecord - to send the data to Kafka,
- JoltTransformJSON - to preprocess the data into desired format,
- UpdateAttribute - to add the hdfs path attribute to the data that depends on the year, month, and day,
- ConvertRecord - to convert the data to the Parquet format,
- LogMessage - to log the errors,
- PutHDFS - to store the data in HDFS.

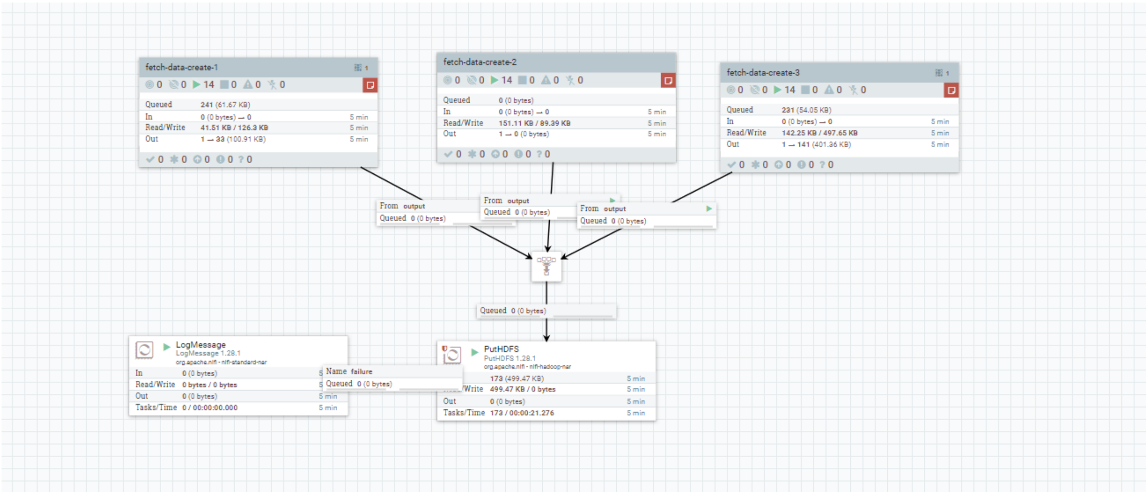


Figure 5: NiFi flow for the data acquisition

# 4 Data storage strategy

Our input data consists of structured data; there is no need to store particular data types, such as images, text files, audio, etc. Therefore, we decided to utilize an SQL-like (CQL) data warehouse system that enables analytics at a massive scale - Apache Hive.

As described in Chapter 2, our datasets contain entirely different sets of features, which makes it impossible to combine them in a single table. All incoming transactions are divided into four groups for each dataset. The code block below represents an example of table creation for dataset 1 of the project.

Listing 1: Apache Hive table creation

```
CREATE EXTERNAL TABLE if not exists dataset1 (
```

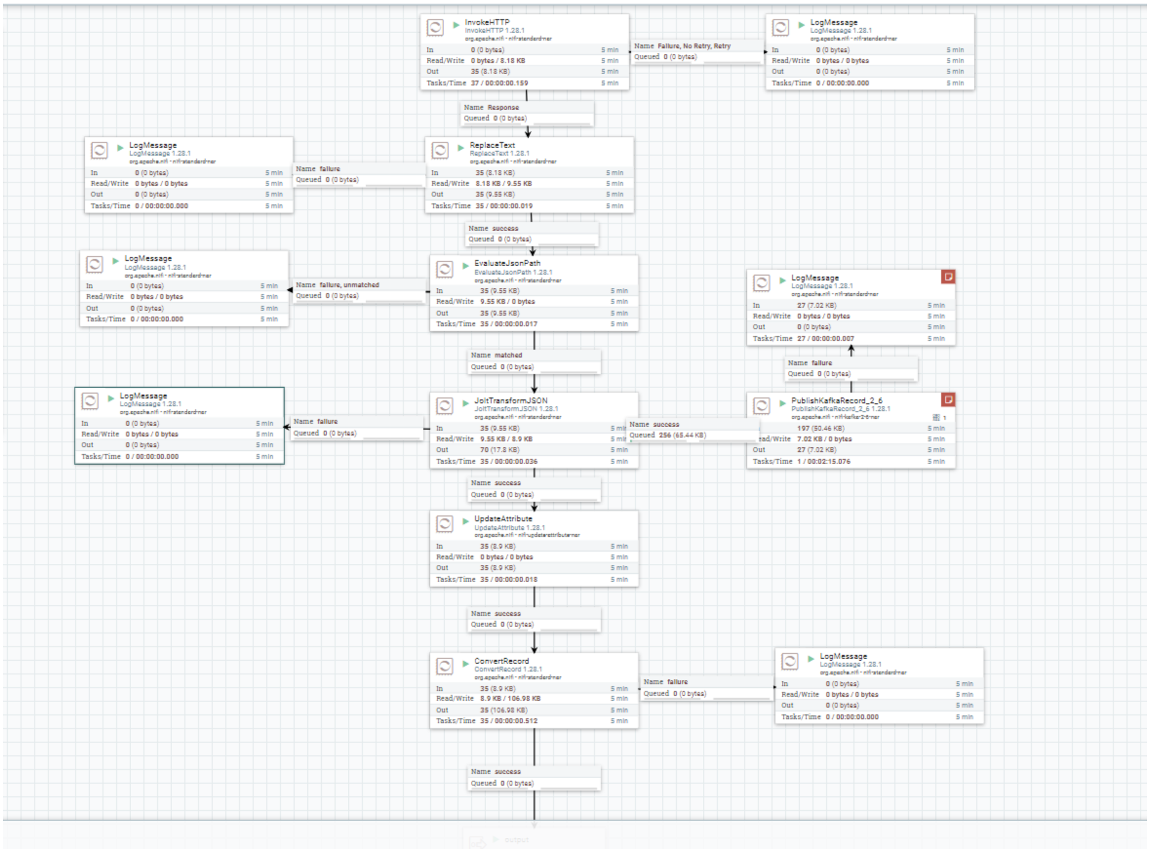

InvokeHTTP
InvokeHTTP 1.28.1
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 8.18 KB 5 min
Out 35 (8.18 KB) 5 min
Tasks/Time 37 / 00:00:00.159 5 min
Name Failure, No Retry, Retry
Queued 0 (0 bytes)
LogMessage
LogMessage 1.28.1
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name Response
Queued 0 (0 bytes)
LogMessage
LogMessage 1.28.1
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name failure
Queued 0 (0 bytes)
ReplaceText
ReplaceText 1.28.1
org.apache.nifi - nifi-standard-nar
In 35 (8.18 KB) 5 min
Read/Write 8.18 KB / 9.55 KB 5 min
Out 35 (9.55 KB) 5 min
Tasks/Time 35 / 00:00:00.019 5 min
Name success
Queued 0 (0 bytes)
LogMessage
LogMessage 1.28.1
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name failure, unmatched
Queued 0 (0 bytes)
EvaluateJsonPath
EvaluateJsonPath 1.28.1
org.apache.nifi - nifi-standard-nar
In 35 (9.55 KB) 5 min
Read/Write 9.55 KB / 0 bytes 5 min
Out 35 (9.55 KB) 5 min
Tasks/Time 35 / 00:00:00.017 5 min
LogMessage
LogMessage 1.28.1
org.apache.nifi - nifi-standard-nar
In 27 (7.02 KB) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 27 / 00:00:00.007 5 min
Name matched
Queued 0 (0 bytes)
Name failure
Queued 0 (0 bytes)
LogMessage
LogMessage 1.28.1
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name failure
Queued 0 (0 bytes)
JoltTransformJSON
JoltTransformJSON 1.28.1
org.apache.nifi - nifi-standard-nar
In 35 (9.55 KB) 5 min
Read/Write 9.55 KB / 8.9 KB 5 min
Out 70 (17.8 KB) 5 min
Tasks/Time 35 / 00:00:00.036 5 min
Name success
Queued 256 (65.44 KB)
PublishKafkaRecord_2_6
PublishKafkaRecord_2_6 1.28.1
org.apache.nifi - nifi-kafka-2-6-nar
197 (50.46 KB) 5 min
Read/Write 7.02 KB / 0 bytes 5 min
Out 27 (7.02 KB) 5 min
Tasks/Time 1 / 00:02:15.076 5 min
Name success
Queued 0 (0 bytes)
UpdateAttribute
UpdateAttribute 1.28.1
org.apache.nifi - nifi-update-attribute-nar
In 35 (8.9 KB) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 35 (8.9 KB) 5 min
Tasks/Time 35 / 00:00:00.018 5 min
Name success
Queued 0 (0 bytes)
ConvertRecord
ConvertRecord 1.28.1
org.apache.nifi - nifi-standard-nar
In 35 (8.9 KB) 5 min
Read/Write 8.9 KB / 106.98 KB 5 min
Out 35 (106.98 KB) 5 min
Tasks/Time 35 / 00:00:00.512 5 min
Name failure
Queued 0 (0 bytes)
LogMessage
LogMessage 1.28.1
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
output

Figure 6: Data processing flow in NiFi

```
    step INT,
    type STRING,
    amount FLOAT,
    nameOrig STRING,
    oldbalanceOrg FLOAT,
    newbalanceOrig FLOAT,
    nameDest STRING,
    oldbalanceDest FLOAT,
    newbalanceDest FLOAT,
    isFraud INT,
    isFlaggedFraud INT
)
PARTITIONED BY (year STRING, month STRING, day STRING)
STORED AS PARQUET
LOCATION '/user/hive/warehouse/dataset1';
```

The preliminary list of all tables in the storage is the following:

1. dataset1 - raw data for the dataset 1;

2. dataset2 - raw data for the dataset 2;

3. dataset3 - raw data for the dataset 3;
4. dataset4 - raw data for the dataset 4;

Additionally, we use Apache Cassandra to store prepared views for fast querying during the data presentation. Naturally, it replicates the processed data from batch and streaming layers. This includes predicted labels for new transactions and an indication of whether data was used for training.

## 5 Project architecture

The project is implemented based on Lambda Architecture. The main data processing is divided into three layers:

1. Speed Layer (streaming)
   - Data preprocessing including transformation to a specific format;
   - Real-time fraud detection on all of the incoming transactions.
2. Batch Layer
   - Data processing and filtering for the model training
   - ML model training with a fixed schedule (e.g., every 10 minutes)
3. Serving Layer
   - Stores processed real-time and batch data in NoSQL for fast querying
   - Client interface highlighting fraud transactions, accepting/blocking transactions;
   - Data visualization with customizable filters

Figure 1 shows an outline of the project architecture.
The following Big Data platforms will be used:

- Apache NiFi: to collect and distribute the data from different sources;
- Apache Hadoop HDFS and Hive: as the main storage solution;
- Apache Cassandra as a view storage for fast querying;
- Apache Kafka: to work with the streaming data;
- Apache Spark: to make the batch processing and model training;
- Apache Superset: for data analysis on the user interface.

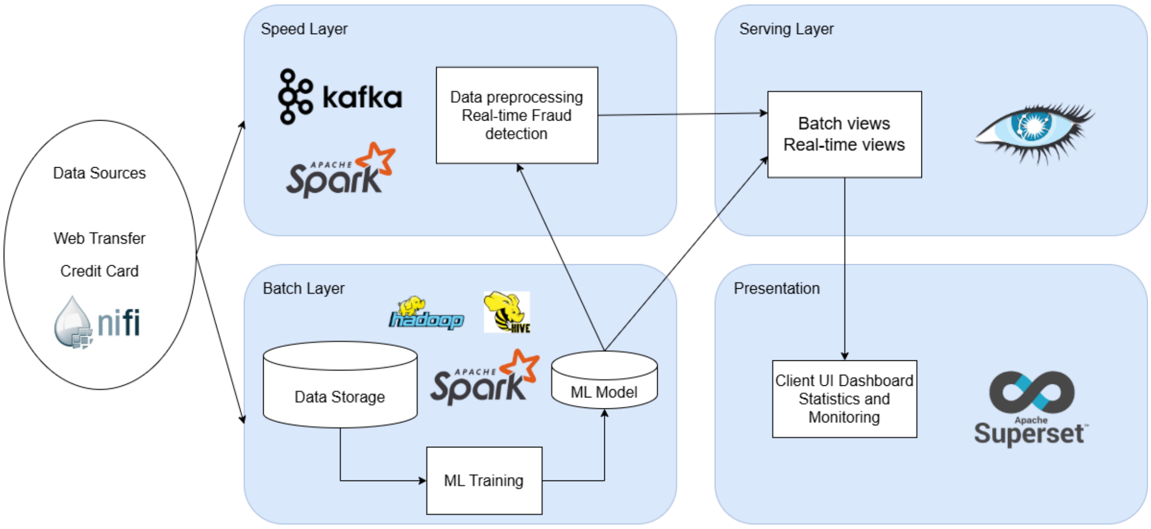


Figure 7: Project architecture

## 6 Speed Layer

The stream processing container is responsible for two main tasks: model training and message processing.

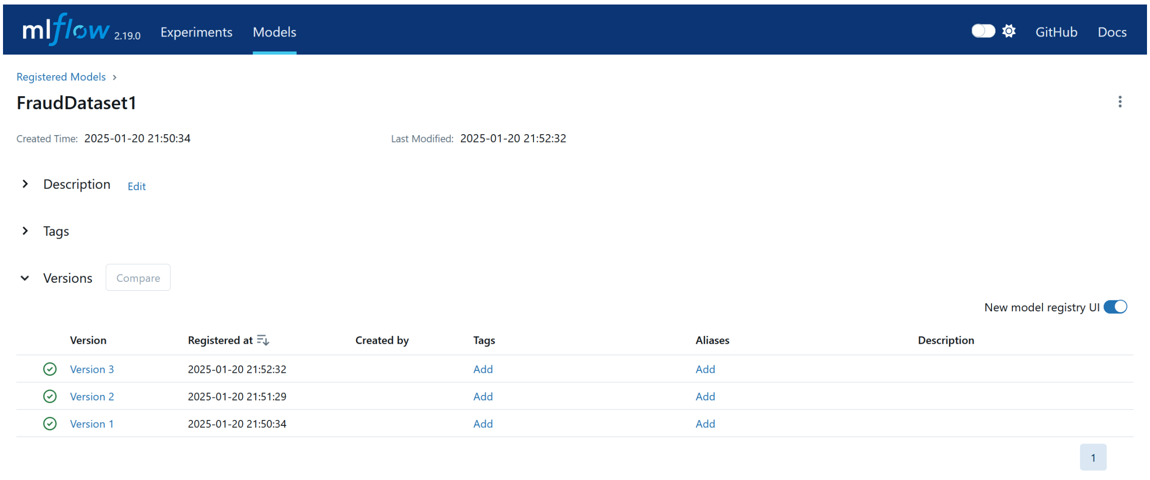
We have tested 4 models for fraud detection and selected Random Forest as it provided better performance on all of the datasets. A comparison of the model's performance is provided in the table below, including metrics precision (P) and recall (R).

| | LogisticRegression | GaussianNB | GradientBoosting | RandomForest |
|---|---|---|---|---|
| Dataset 1 | P: 0.35; R: 0.44 | P: 0.04; R: 0.18 | P: 0.9; R: 0.41 | P: 0.97; R: 0.77 |
| Dataset 2 | P: 0.89; R: 0.58 | P: 0.79; R: 0.59 | P: 1.0; R: 0.99 | P: 1.0; R: 0.99 |
| Dataset 3 | P: 0.0; R: 0.0 | P: 0.48; R: 0.1 | P: 1.0; R: 1.0 | P: 1.0; R: 1.0 |

Table 6: Performance metrics of fraud detection models

The initial model is being trained locally with a spark on the collected 'train' portion of the datasets. As the next step, the streaming layer uses Spark Streaming in micro-batches mode to consume three Kafka streams and run predictions on them. Moreover, this module also includes the joining of these streams to produce additional insights.

Additionally, we planned introduce periodic model retraining and model version menagement using MLflow. We have set up a separate container (services/mlflow) with MLflow server and a cron job to update the fraud classifier. The model artifacts are saved in a shared storage (Docker volume or, preferably, HDFS). The results of the job run are available in container logs or MLflow UI as shown on Figure 8. Unfortunately, we had issues connecting stream processing container to the artifacts storage, therefore, a localy pretrained model is used instread.



FraudDataset1

Created Time: 2025-01-20 21:50:34

Last Modified: 2025-01-20 21:52:32

| | Version | Registered at | Created by | Tags | Aliases | Description |
|---|---|---|---|---|---|---|
| ✓ | Version 3 | 2025-01-20 21:52:32 | | Add | Add | |
| ✓ | Version 2 | 2025-01-20 21:51:29 | | Add | Add | |
| ✓ | Version 1 | 2025-01-20 21:50:34 | | Add | Add | |

Figure 8: MLflow UI with trained model versions

## 7 Batch Layer

The batch layer is implemented using Apache Spark. On a regular basis (currently, every 5 minutes), the transaction data is taken from Hive to prepare some analytics on it. The resulting data is stored in Cassandra incrementally.

In particular, we are currently preparing data for the following views:

1. Fraud Statistics by Transaction Type for the Dataset 1
   - To analyze fraud patterns based on transaction types. We calculate fraud rates across specific transaction types to investigate whether certain types are more or less vulnerable to fraud.
2. Fraud Statistics by Transaction Amount for the Dataset 1
   - To analyze fraud patterns based on transaction amounts. Similar to the previous view, the fraud rate is calculated, but across several amount ranges (0-1000, 1000-10000, 10000-50000, 50000-100000, 100000-500000, 500000+) instead of types.
3. Hourly Fraud Statistics for the Dataset 2
   - To identify trends in fraudulent activities based on transaction hours. Given the timestamp of each transaction, we calculate the amount of fraud as well as the percentage of fraud transactions per time frame (e.g., hour);
4. High-Risk Customer Analysis for the Dataset 2
   - To identify customers with a high likelihood of fraud. The total amount of transactions per customer, as well as the fraud percentage, are calculated.

# 8 Serving layer

The serving layer relies on Apache Superset, TrinoDB, and Cassandra. Superset is an open-source data visualization and business intelligence platform well-suited for modern data pipelines due to its extensive support for a wide range of data sources and ease of integration into existing architectures.

## 8.1 Integration of Apache Superset with the Current Architecture

As illustrated in the architecture diagram:

- The Batch Layer, powered by Apache Hive, processes large-scale data and aggregates it into queryable formats.
- The Speed Layer, which uses Kafka for real-time data ingestion and Spark for processing, produces real-time views stored in Cassandra. Superset can connect to Apache Cassandra through TrinoDB to display these real-time analytics, provided an appropriate connector is configured.
- The Serving Layer facilitates the querying of both batch and real-time views, making this data accessible to Superset for visualization in unified dashboards.

By integrating Superset with batch and speed layers, the system can provide users with comprehensive dashboards that include historical trends and real-time updates. This combination is particularly beneficial for monitoring critical use cases such as fraud detection, as it allows stakeholders to view both immediate alerts and long-term patterns.

## 8.2 Advantages of Using Apache Superset

- Ease of Use: Superset provides an intuitive drag-and-drop interface for creating visualizations and dashboards, making it accessible to non-technical users.
- Real-Time and Batch Data Integration: Combining data from the speed and batch layers into unified dashboards enables comprehensive analytics and monitoring.
- Custom Visualization: Superset supports a variety of visualizations, allowing users to explore data in formats that best suit their needs, from basic bar charts to advanced geographic maps.
- Scalability: Being lightweight and web-based, Superset can handle a growing volume of data as the architecture scales.
- Open Source and Extensibility: Superset can be customized and extended to fit specific requirements as an open-source platform.

## 8.3 Disadvantages and Challenges

While Superset provides significant benefits, there are a few challenges and limitations to consider:

- Indirect Support for Kafka: Superset does not natively connect to Kafka. Before it can be visualized, real-time data from Kafka must first be stored in a queryable data store (e.g., MongoDB or a similar OLAP engine).
- Cassandra Support: Superset does not have native support for Cassandra. Integration through a Python library or TrinoDB is required, adding some complexity.
- Performance Considerations: For real-time monitoring, performance can be a bottleneck if the underlying data stores are not optimized for frequent querying by Superset.
- Learning Curve for Advanced Features: While the primary interface is user-friendly, advanced configurations (e.g., complex filters and custom SQL queries) may require technical expertise.

The serving layer consists of two dashboards created in Apache Superset presented in Figure 9 and 10. The dashboards present various charts created from the views generated by batch and streaming layers.

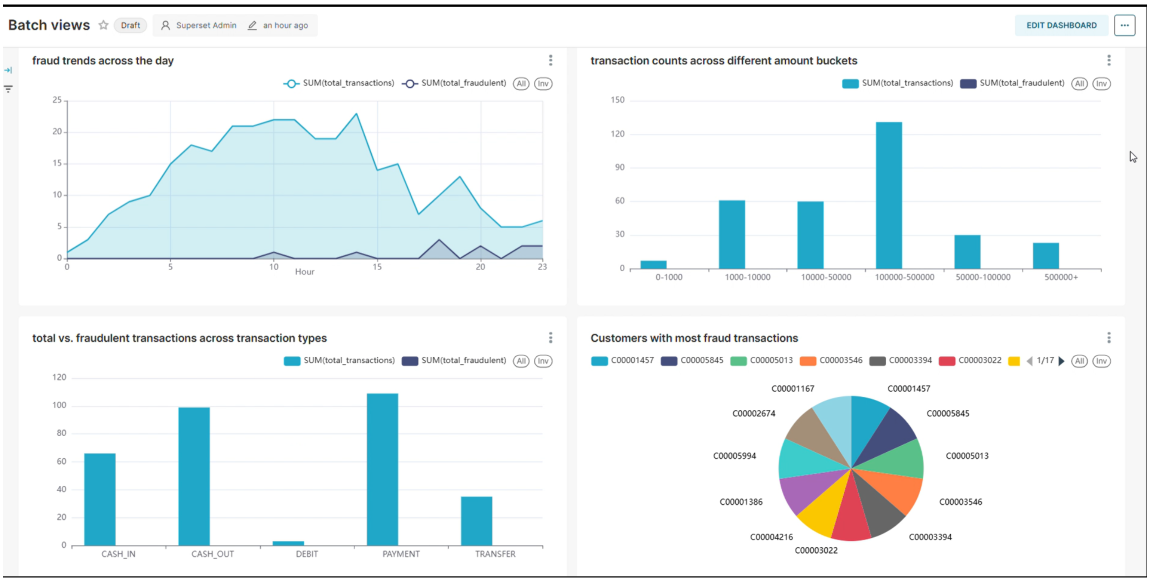


Figure 9: Batch layer dashboard

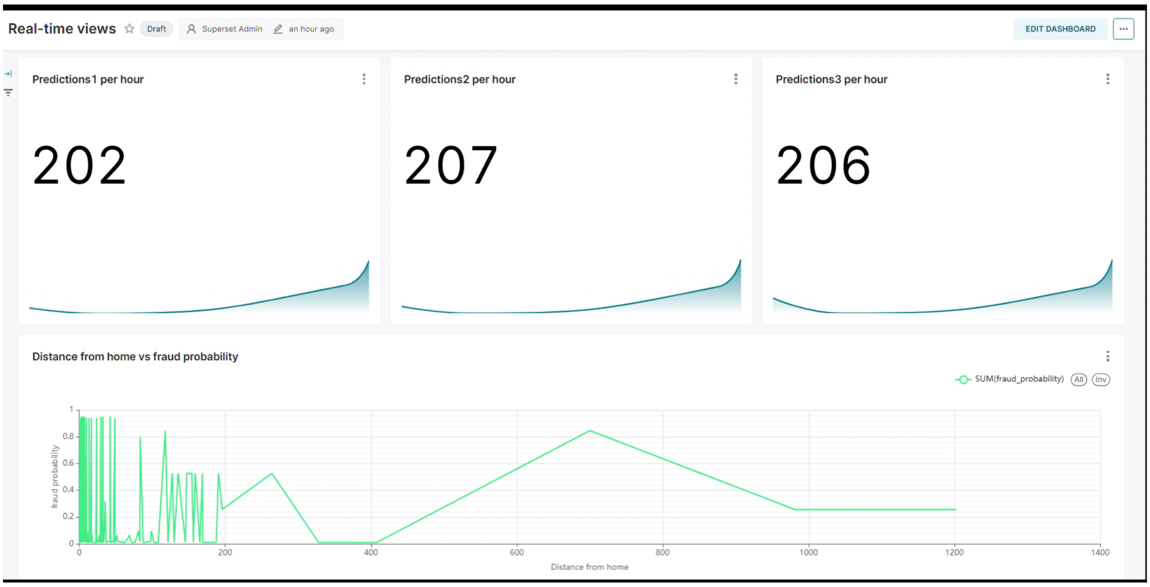


Figure 10: Speed layer dashboard

# 9 Project impact

The project of fraudulent transaction predictions has potential positive and negative impacts across various fields.

1. Fraudulent transactions cost businesses and individuals billions annually. Detecting and preventing them minimizes these losses, protecting bottom lines and investor confidence.

2. Reduced reliance on manual fraud assessment resulting in lower costs.

3. Increased customer trust.

4. Possibility to assist law enforcement in identifying and prosecuting fraudulent networks.

5. Potential harm resulting from poor predictions.

6. Some edge cases might have been missed, resulting in customer annoyance, such as false fraud predictions based on distance from the last transaction when going on holidays.

The project has been realized with Big Data technologies, primarily based on Apache software and Docker, allowing for high fault tolerance and scalability. It is ready to handle incoming data robustly, representing the 5 Vs characteristics. However, for production-level quality, it should be extended with tools such as Kubernetes, allowing for seamless container scalability. It is worth pointing out that the whole system is resource expensive in regards to RAM, one needs to take that into consideration before planned deployment.

## 10 Tasks assignment

The table below contains the list of team members and the allocation of tasks to team members.

| Team member | Tasks | Supporter |
|---|---|---|
| Salveen Singh Dutt | Batch processing of the historical data for up-to-date model training (Batch Layer). | Karina Tiurina |
| Karina Tiurina | Fraud detection model training and fine-tuning; Data stream processing (Speed Layer). | Salveen Singh Dutt |
| Patryk Prusak | Data ingestion, collection, and pre-processing. | Karina Tiurina, Salveen Singh Dutt |
| Patryk Prusak | Data visualization and configuration on the UI (Serving Layer). | Karina Tiurina, Salveen Singh Dutt |

Table 7: Tasks assignment

# References

[1] Chitwan Manchanda, Fraudulent Transactions Data, Kaggle, 2022. Available at: https://www.kaggle.com/datasets/chitwanmanchanda/fraudulent-transactions-data.

[2] Credit_Card_Fraud_, OpenML, 2024. Available at: https://www.openml.org/search?type=data&status=active&id=45955&sort=runs.

[3] Carlos José, Credit Card Transactions Synthetic Data Generation, Kaggle, 2024. Available at: https://www.kaggle.com/datasets/cgrodrigues/credit-card-transactions-synthetic-data-generation?select=transactions_df.csv.

[4] Machine Learning Group - ULB and Andrea, Credit Card Fraud Detection, Kaggle, 2018. Available at: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.

[5] Lambda Architecture, Cazton. Available at: https://cazton.com/consulting/enterprise/lambda-architecture.

[6] OpenAI, ChatGPT: Language Model, Available at: https://chatgpt.com.