**Master of Science**

**<<Artificial Intelligence and Visual Computing>>**

# UNIVERSITY OF WEST ATTICA
# & UNIVERSITY OF LIMOGES

## FACULTY OF ENGINEERING

### DEPARTMENT OF INFORMATICS AND COMPUTER ENGINEERING

***Master Thesis***

# Real-Time Fraud Detection Using Big Data and ML Techniques

**Student: Kastrinos Theodoros (aivc22006)**

**Supervisors: Prof. Tsolakidis Anastasios, Prof. Andritsos Periklis**

**Athens, February 2025**

**Members of the Examination Committee, including the Supervisor**
**The thesis was successfully examined by the following Examination Committee.**

| S/N | NAME | RANK/ROLE | DIGITAL SIGNATURE |
|---|---|---|---|
| **1** | Mastorocostas Paris | Professor | PARIS MASTOROKOSTAS Digitally signed by PARIS MASTOROKOSTAS Date: 2025.02.10 18:04:29 +02'00' |
| **2** | Kesidis Anastasios | Professor | ANASTASIOS KESIDIS Digitally signed by ANASTASIOS KESIDIS Date: 2025.02.10 20:24:24 +02'00' |
| **3** | Tsolakidis Anastasios | Assistant Professor | ANASTASIOS TSOLAKIDIS Digitally signed by ANASTASIOS TSOLAKIDIS Date: 2025.02.11 13:37:49 +02'00' |

## Abstract

Fraud detection in financial transactions finds great importance in the fight against financial crime. In this thesis, two models are developed: CatBoost and Light Gradient Boosting Machine for classifying transactions into "Fraud," "No Fraud," or "Suspicious." Later, the suspicious transactions will be relabeled after investigation and reintegrated into the training data, which will enhance the model performance. Apache Kafka allows doing real-time processing of data to efficiently handle live transactions. Challenges regarding dataset imbalance were addressed by employing class weights proportional to the inverse of class frequencies, further adjusted by a scaling factor to ensure optimal balance during training. In developing the adaptive accurate detection of frauds, this work designed a real-time pipeline, using a feedback loop iteratively in model refinements. Both the models are yielding good results; LGBM gave the best regarding precision and recall. The reintegration of relabeled data greatly increased accuracy, and the optimization performed with a focus on loss ensured that detection was better compared to traditional metrics. This thesis aims to contribute to the domain of fraud detection by presenting an adaptive and scalable framework in combination with real-time processing and continuous learning coupled with machine learning models. It addresses the challenges related to handling imbalanced datasets and evolving fraud patterns in real-world scenarios.

---

*Keywords:* Fraud detection, financial transactions, machine learning, Catboost, LightGBM, imbalanced datasets, real-time processing, Apache Kafka, continuous learning, Apache Superset, Optuna

## Acknowledgments

First, I would like to express my deepest gratitude to my advisor, Professor Periklis Andritsos, for his exceptional guidance, support, and mentorship throughout my research journey. Professor Andritsos has been an invaluable mentor, consistently encouraging me to explore new academic disciplines and pushing me to unlock my full potential. His profound expertise and insightful advice have not only guided me through the complexities of this thesis but also shaped my approach to research and critical thinking. His unwavering support and motivation played an instrumental role in the successful completion of this work, and for that, I am deeply appreciative.

I would also like to extend my sincere thanks to Professor Anastasios Tsolakidis, whose assistance and encouragement were pivotal in helping me finalize this thesis. His constructive feedback and invaluable guidance throughout the final stages of my work have been immensely helpful. I deeply appreciate his willingness to provide insights and direction when I needed it most, contributing significantly to the successful completion of this research.

In addition, I extend my heartfelt thanks to my family, whose constant support and understanding have been a cornerstone of this journey. Their belief in my abilities, coupled with their encouragement during moments of difficulty, provided me with the strength to persevere and continue striving toward my goals. The countless sacrifices they made to support me in this endeavor have not gone unnoticed, and I am profoundly grateful for their presence and unconditional love throughout this process. Without their encouragement and belief in me, this achievement would not have been possible.

# Table of Contents

## List of Figures

## List of Tables

## Introduction

The increasing adoption of e-commerce, online and mobile banking, and other digital financial services has resulted in a manifold increase in electronic transactions and digital payments. While this growth has increased convenience and reach for consumers, it has also seen a corresponding rise in fraudulent activities. These frauds cause immense economic loss to businesses and individuals, which erodes consumer confidence and the integrity of the financial system.

The dynamic and sophisticated nature of modern fraud schemes makes traditional methods increasingly ineffective in fraud detection. Traditional approaches generally rely on a static, rules-based system, which is by design rigid and bound to predefined rules and thresholds, thus being very prone to false positives and false negatives and unable to adapt to fast-changing fraudster tactics. Therefore, there is a huge need for far superior, adaptable approaches that are data-driven towards timely and effective fraud detection and mitigation.

More advanced and proactive methods for detecting fraud must be adopted by organizations, with fraudulent tactics constantly evolving, making real-time fraud detection crucial in allowing the prompt detection and addressing of suspicious activities and minimizing potential risks and financial losses. There is an increasing potential to create systems that can learn from historical data, adapt to new fraud patterns, and provide continuous monitoring because of the growing prevalence of artificial intelligence (AI) and machine learning (ML), with this process being essential for organizations that wish to protect their financial transactions and preserve consumer confidence.

Various industries have been transformed by the inclusion of fraud detection systems by the emergence of machine learning and big data technologies, with big data enabling the gathering and examination of large volumes of information from multiple sources, providing a comprehensive dataset to detect patterns and identify anomalies. Machine learning algorithms, like LightGBM and CatBoost, provide powerful tools for analyzing this data, possibly enhancing the accuracy and efficiency of fraud detection systems by uncovering intricate relationships and patterns that conventional statistical methods may miss, being able to process and analyze data streams in real-time applications, enabling the immediate identification of suspicious transactions.

The objectives of this research are presented below.

- **Develop a real-time fraud detection pipeline using a CatBoost-based** approach by designing and implementing a CatBoost model capable of analyzing transaction data in real-time and detecting potential fraud.
- **Develop a LightGBM model to establish a real-time fraud detection pipeline**, by employing the LightGBM algorithm, which is renowned for its robustness and capacity to handle various datasets, classify transactions, and detect fraudulent activities.
- **Evaluate the metrics of these two pipelines by** examining them by testing the CatBoost and LightGBM models to see which performs best based on their recall, processing speed, and ability to handle increasing amounts of data.

- **Evaluate the environmental impact of the training processes regarding CO2 emissions and electricity consumption**, evaluating the environmental impact of the training processes by assessing CO2 emissions and electricity consumption, ensuring the sustainability of the machine learning models in light of their computational requirements.

# 1 Literature Overview

## 1.1 Overview of Existing Fraud Detection Methods

Enormous evolution in fraud detection has been caused by the complexity and sophistication of fraudulent actions with rule-based systems intended to identify transactions based on predetermined conditions, including anomalous transaction amounts or suspect geographic areas, being a significant component of traditional techniques and, although initially successful, the ever-evolving strategies used by fraudsters posed a challenge to these approaches. Traditional systems revealed flaws as fraud grew more dynamic, especially their inability to adjust to novel and unexpected patterns. (Ngai et al., 2011).

### 1.1.1 Rule Based

Rule-based systems have been a critical aspect of fraud detection since the early days of digital financial transactions, by applying static rules established by subject-matter experts to prevent transactions from coming from specific regions or exceeding a predetermined threshold and despite being simple to understand and comprehend, frequently leading to significant false-positive rates—incorrectly marking everyday transactions as fraudulent. Moreover, (Phua et al., 2010) there is severe non-scalability and requirement of continual human updates in order to handle novel fraud techniques of rule-based systems as they lack the flexibility needed for ongoing learning and improvement, they cannot handle more complex and nuanced forms of fraud.

### 1.1.2 Anomaly Detection Techniques

Anomaly detection techniques aim to spot changes in regular transaction patterns that might indicate fraud, being divided into two categories: supervised and unsupervised, with models like decision trees, logistic regression, and support vector machines (SVMs) being examples of supervised techniques that operate on labeled data, which have already been used to identify fraudulent and non-fraudulent transactions (Bolton & Hand, 2002), relying however significantly on the availability of labeled datasets, which are frequently lacking or insufficient in fraud detection scenarios.

On the other hand, unsupervised techniques look for outliers in the dataset rather than requiring labeled data with unsupervised anomaly detection often using autoencoders, isolation forests, and k-means clustering, being more flexible and adaptive than rule-based systems since they identify novel,

previously undiscovered fraud patterns. (Zhang et al., 2017). However, their interpretability issues often make it challenging for companies to understand the logic behind flagged transactions.

### 1.1.3 Machine Learning-Based Approaches

Machine learning (ML) has become a potent tool for fraud detection in recent years as it can learn from enormous volumes of data and eventually adapt to new fraud trends with labeled transaction data being readily available, making supervised learning the most widely utilized strategy in fraud detection in machine learning models. (Bahnsen et al., 2015). Unsupervised learning models are classified into two categories that have all been used in fraud detection with differing degrees of effectiveness based on algorithms like decision trees, random forests, gradient boosting, and neural networks. Supervised models usually perform well when accessing copious amounts of labeled data, being able to apply the lessons from past fraud cases to new information with class imbalance issues frequently impeding them however, as the quantity of legal transactions greatly outweigh the fraudulent ones.

Fraud detection also uses unsupervised machine learning models, especially when labeled data is few or nonexistent, by identifying patterns in the data that deviate from the norm and may indicate fraudulent activity with autoencoders, being an unsupervised neural network, compress and reconstruct input data to detect such anomalies. A transaction might be reported as fraudulent if it differs noticeably from the anticipated reconstruction (Hawkins et al., 2002), with unsupervised models possibly having difficulty with accuracy, unlike supervised models trained on large datasets, still being accommodating for identifying new fraud schemes.

### 1.1.4 Hybrid Methods

Hybrid approaches are used by many companies in an effort to overcome the drawbacks of rule-based and machine-learning approaches, as they combine the interpretability of rule-based techniques with the flexibility of machine learning models with a hybrid system for example using a machine learning model to handle more subtle, complicated fraud patterns, while a rule-based system handles simpler, well-known patterns (Zheng et al., 2020), making it possible to detect fraud more thoroughly, lowering the number of false positives and improving the system's capacity to identify new kinds of fraud.

### 1.1.5 Challenges in Fraud Detection

Despite the advancements in fraud detection techniques, there are still a few issues with the quick development of fraud techniques being one significant problem that calls for constant model updating and retraining and the data privacy problem moreover making collecting enough labeled data for machine learning model training difficult, and fraud detection systems lastly being required to process data in real time, deciding quickly and accurately without interfering with valid transactions.

The field of fraud detection has changed throughout time, moving from more static rule-based systems to more dynamic machine learning techniques, with machine learning models being more flexible and accurate than traditional methods when identifying complex fraud patterns, offering transparency and simplicity. At the same time there is a trend toward hybrid approaches, including both techniques, and is expected to become more prevalent as fraud becomes more sophisticated in the upcoming years.

## 1.2 Review of CatBoost in Fraud Detection

CatBoost has grown in popularity for fraud detection because it is a gradient boosting framework designed for categorical data; it can handle complex nonlinear relationships in data while providing robust performance even in the case of an imbalanced dataset. Unlike traditional methods, which rely on predefined rules or linear models. It excels in adaptive learning, making it well-suited for the dynamic nature of fraud detection in financial systems, where fraudsters constantly evolve their tactics to evade existing detection mechanisms (Prokhorenkova et al., 2018).

A great advantage of CatBoost comes from how naturally it can use categorical features without aggressively preprocessing or manually encoding them. Banking systems generate hundreds of millions of transactions with timestamps, merchant identifiers, customer demographics, among many other variables. The CatBoost approach robustly processes such high-dimensional data at train time by converting the categorical variables to numerical representations such that critical relationships are preserved and complex patterns indicative of fraud can be detected. This native ability to manage categorical data is especially important in the financial domain, where often the interaction of features drives fraudulent behavior.

CatBoost is very suitable for real-time fraud detection because of the efficiency of model training and prediction. Unlike neural networks, which often require high computational resources, the optimized training of CatBoost minimizes overfitting while keeping speeds high, even on very large datasets. Its applicability to real-time systems is enhanced by its built-in mechanisms for handling missing data and its ability to adapt to evolving fraud patterns with minimal latency, ensuring that financial institutions can quickly identify and respond to suspicious transactions (Dorogush et al., 2018

Besides that, CatBoost incorporates state-of-the-art techniques for handling the class imbalance problem typical in fraud detection, where actual fraudulent transactions constitute a small fraction of the total volume. CatBoost reduces bias towards the majority class by employing custom loss functions and dynamic class weight to achieve a better balance between precision and recall. This means there will be fewer false negatives; hence, probable fraudulent transactions are flagged for further investigation with minimal disruption to legitimate transactions.

Another important advantage of CatBoost over complex models like neural networks is that it is interpretable; it contains feature importance scores, including SHAP values for clear model insights into its decisions. In banking systems, this interpretability becomes important due to regulatory requirements that will soon put pressure on explaining, for example, flagged transactions or denied

approvals. This makes CatBoost balanced regarding both demands: performance and transparency of the model.

While CatBoost has a number of strengths, it also has certain drawbacks, such as sensitivity to hyperparameter tuning and possible high computational demands in the case of really large datasets. These disadvantages are partly compensated for by the efficiency of CatBoost in categorical data processing and the reduced need for feature engineering. Besides that, CatBoost is scalable architecture-wise, thus allowing fitting into distributed systems, enabling it to process millions of transactions with minimal latency in real-time environments.

Overall, CatBoost presents a compelling solution for fraud detection in financial systems, combining high accuracy, interpretability, and efficiency. Its ability to process categorical data natively, address data imbalance, and operate effectively in real-time systems positions it as a leading choice for adaptive fraud detection in dynamic and high-stakes environments (Prokhorenkova et al., 2018; Dorogush et al., 2018).

## 1.3 Review of LightGBM in Fraud Detection

LightGBM (Light Gradient Boosting Machine) is a gradient boosting framework known for its exceptional efficiency and speed, mainly when dealing with large datasets, and as a result has garnered significant attention in machine learning, giving it an edge over its competitors regarding performance and memory use, making it the best choice for demanding tasks like finding fraud, by utilizing it’s unique decision tree learning method, in which trees grow leaf-wise instead of level-wise,. (Ke et al., 2017).

LightGBM's ability to manage extensive transaction data, including high-dimensional features such as transaction quantities, timestamps, locations, and customer details is highly respected by the banking sector, valuing it for its ability to efficiently handle large-scale transaction data, including high-dimensional features such as transaction amounts, timestamps, locations, and customer details, especially in fraud detection. The capability of natively managing missing values common in transaction data and handling categorical variables makes LightGBM highly effective, simplifying the data preprocessing pipeline, facilitating the handling of complex banking datasets that contain numerical and categorical features, unlike many machine learning models that require extensive preprocessing.

The scalability and rapidity is one of LightGBM's most noteworthy benefits in fraud detection, by achieving superior accuracy and quicker training times by growing trees leaf-wise and concentrating on the nodes with the highest loss reduction when being compared to conventional methods such as Random Forests or XGBoost, which is crucial in fraud detection scenarios, enabling prompt decision-making on potentially fraudulent transactions and flagging or blocking suspicious activities before they inflict significant damage.

Another advantage is its capacity to manage imbalanced datasets, a prevalent fraud detection problem with fraudulent transactions comprising only a minor proportion of the data and LightGBM resolving this through various mechanisms, including a "weighted" metric prioritizing the minority class (fraudulent transactions). The adaptation of the loss function to balance precision and recall or assigning a higher weight for the minority class to enhance the model's sensitivity to fraud without compromising accuracy for the majority class (non-fraudulent transactions), along with other parameters are also provided by the LightGBM algorithm.

Finally, LightGBM excels is feature significance by offering financial institutions valuable insights into the features that are most influential in predicting fraud, thereby assisting them in identifying critical indicators of fraudulent behavior, with LightGBM's feature importance metrics possibly emphasizing specific transaction patterns or client behaviors that may consistently indicate fraud, being especially beneficial in the banking industry, where regulatory requirements frequently require explicit explanations for decisions regarding whether to approve, deny, or mark a transaction for further investigation.

The detection of fraud using LightGBM, despite having numerous benefits, also presents some obstacles, with the need for fine tuning of the model's many hyperparameters, including the learning rate, number of leaves, and feature fraction, in an effort to guarantee optimal performance. While LightGBM can train rapidly, the optimal balance of precision and recall frequently necessitates meticulous hyperparameter tuning, which can be both computationally costly and time-consuming, with another obstacle being the potential for overfitting, mainly when dealing with high-dimensional datasets and although LightGBM's leaf-wise growth strategy enhances accuracy, it may also result in overfitting if not adequately regularized. Lastly, another critical factor in LightGBM's efficacy is data quality, with inconsistent or biased data possibly adversely affecting its predictions, even though the model can accommodate absent values, potentially disrupting legitimate transactions or allow fraud to go undetected, making it imperative to maintain high-quality, well-represented training data.

## 1.4 Real-Time Data Processing and Kafka

Real-time data processing has become essential in fraud detection, where quick choices are critical, with conventional batch processing techniques working well for analyzing historical data, but falling short for applications that need to analyze data instantly and take immediate action and real-time data processing making continuous analysis of incoming data streams possible, enabling systems to react swiftly to dynamic events and make timely choices.

Designed to handle high-throughput, low-latency data streams, Apache Kafka is a distributed streaming platform and one of the leading real-time data processing platforms with Kafka being the first software LinkedIn created, being evolved into a standard for processing data in real-time in a variety of businesses. Because of the distributed commit log at the center of its architecture, producers can transmit data to topics, and consumers can subscribe to these topics to access and process data

with large volumes of streaming data being handled by Kafka with efficiency thanks to its ability to grow horizontally over several brokers (Kreps et al., 2011).

Kafka is essential to fraud detection because transaction data can be continuously ingested and processed in real-time, enhancing fraud detection systems, which must evaluate transactions almost instantly to prevent fraudulent activity from escalating, ensuring low-latency data streaming, allowing machine learning models to categorize transactions as suspicious, fraudulent, or non-fraudulent quickly.

Fault tolerance and durability are further benefits of Kafka's distributed architecture, being essential for mission-critical applications like fraud detection, with the risk of data loss being reduced via data replication over numerous brokers in the event of hardware failures, enabling companies to set up retention policies maximizing storage and maintain responsiveness (Narkhede et al., 2017). The integration of Apache Spark and Flink which are data processing frameworks interfacing easily with Kafka, is frequently used by organizations, as a more sophisticated real-time analytics pipeline, with real-time fraud detection models depending on ongoing data processing and decision-making often requiring these frameworks' extensive stream processing characteristics, such as windowing and stateful computations. (Zaharia et al., 2016).

## 1.5 Related Work in Big Data Analytics for Fraud Detection

Big data analytics have emerged as an essential tool in the struggle against fraud with traditional fraud detection methods, frequently dependent on rule-based systems or basic statistical models, being unable to keep up with the escalating scope and complexity of fraudulent activities, facilitating a substantial transition to the utilization of big data analytics, enabling the processing and analysis of extensive datasets to detect patterns, anomalies, and trends that suggest fraudulent behavior.

One of the most significant developments in the field of fraud detection is the incorporation of machine learning algorithms with big data platforms, including Apache Hadoop and Apache Spark, facilitating large-scale datasets being processed across distributed computing environments. This solution enables the application of complex models such as Random Forests, LightGBM, Neural Networks, and Support Vector Machines (SVMs) with Aggarwal (2016) demonstrating that applying machine learning models to big data, potentially enhancing the accuracy and speed of fraud detection systems, identifying subtle patterns that conventional methods may overlook.

# 2 Methodology

## 2.1 Tools and System

The project is using Ubuntu 24.04 with the specifications listed below:

| | |
|---|---|
| **CPU Model on constant consumption mode** | AMD Ryzen 7 5800H with Radeon Graphics |
| **Platform system** | Linux-6.8.0-39-generic-x86_64-with-glibc2.39 |
| **Python version** | 3.12.3 |
| **CodeCarbon version** | 2.5.0 |
| **Available RAM** | 30 GB |
| **CPU count** | 16 |
| **GPU model** | 1 x NVIDIA GeForce RTX 3070 Laptop GPU |

***Table 1 – Hardware Specifications***

Python is the main programming language used to create the algorithms for this project; it is the best option for this task because of its many libraries and frameworks. With libraries like TensorFlow, Scikit-learn, and Pandas, among others, this language offers strong tools for creating and improving machine learning models, making it ideal for data science, machine learning, and deep learning applications. The above- mentioned packages give Python its adaptability and extensive ecosystem, making it perfect for complex projects including evaluation, model training, data preprocessing, and real-time decision-making.

By encapsulating the code, libraries, and environment configurations required for various phases of the machine learning pipeline, from training to real-time evaluation, a Docker is used, ensuring a streamlined and consistent deployment process, potentially preventing problems caused by disparities between environments and enabling seamless transitions from development to production. In classifying transaction data into three groups, suspicious, no fraud, and fraud, the likely alternatives of the model evaluations, Docker containers supervise a PostgreSQL database that houses the outcomes of the training processes.

| Container ID | Image | Command | Created | Status | Ports | Names |
|---|---|---|---|---|---|---|
| **2958a8ebf44b** | redis | “docker-entrypoint.s…” | 3 months ago | Up 3 minutes | 0.0.0.0:6379/tcp, :::6379->6379/tcp | redis-container |
| **8d099417a3d7** | dpage/pgadmin4 | “/entrypoint.sh” | 3 months ago | Up 3 minutes | 443/tcp, 0.0.0.0:8081->80/tcp, [::]:8081->80/tcp | pgadmin-container |
| **11fbf147ece3** | postgres | “docker-entrypoint.s…” | 3 months ago | Up 3 minutes | 0.0.0.0:5432/tcp, :::5432->5432/tcp | postgres-container |

| | | | | | | |
|---|---|---|---|---|---|---|
| **bb8074cf52e3** | confluentic/cp-kafka:latest | "/etc/confluent/dock…" | 3 months ago | Up 10 seconds | 0.0.0.0:9092/tcp, :::9092->9092/tcp | kafka |
| **bce2736d55e3** | zookeeper:latest | "docker-entrypoint.…" | 3 months ago | Up 3 minutes | 2181/tcp, 2888/tcp, 3888/tcp, 8080/tcp | zookeeper |
| **15d65433a286** | apache/superset | "/usr/bin/run-server…" | 40 hours ago | Up 40 hours | 0.0.0.0:8088->8088/tcp, :::8088->8088/tcp | superset-container |

***Table 2 - Docker configuration***

Because of the requirement for real-time data processing, Apache Kafka was chosen, facilitating smooth read, train, and write operations, making it possible to handle large volumes of data with low latency and high throughput. The aforementioned is essential in fraud detection systems where quick decisions are needed to reduce risks as they appear with Redis, an in-memory data structure store, being one of Kafka's two main component structures, involving managing Kafka offsets and tracks the status of data processing, guaranteeing dependable and scalable processing, with Redis, being a high-speed storage layer serving as the system's backbone and ensuring fault-tolerant and real-time performance.

In addition to providing robust, interactive dashboards that offer profound insights into transaction data and the effectiveness of machine learning models, Apache Superset is used for data visualization and performance monitoring of fraud detection models, being essential for monitoring the real-time flow of data, identifying trends, and identifying anomalies. Furthermore, it facilitates the analysis of model performance over time, enabling for adjustments and improvements based on the processed data, allowing scientists and decision-makers to access important metrics and insights in an interactive, user-friendly way.

Each container needed to be part of the same network for proper service communications. The arrangement that was adopted for running all the containers on the same network, here referred to as pgnetwork, forms the basis for ensuring that the connections are effective for smooth transactions of data among different parts with least hindrance possible in the overall system.

Two primary machine learning pipelines, one dedicated to CatBoost and the other to LightGBM models are used in the proposed system, with each pipeline being responsible for a distinct aspect of the fraud detection system, working together to cover various transaction patterns. Managing the entire process for each step, from the data ingestion and preprocessing step to the model training step, real-time evaluation, and retraining when necessary is a series of scripts, integrating Random Forest to investigate further and refine suspicious data classification enhances model accuracy and fine-tunes the overall classification process, leading to more precise detection of fraudulent activities.

## 2.2 Data Collection and Preprocessing

A significant part of the data collected came from the IEEE-CIS Fraud Detection competition dataset, available on Kaggle (Kaggle, 2019), containing a wide range of features, such as transaction amounts, timestamps, product codes, device information, numerical features for identity, etc, with the diversity and richness of this dataset being crucial in the effective identification of potentially fraudulent activities. The rest of the data were sourced by various sources, including transaction records from financial institutions, e-commerce platforms, and public datasets.

Data Preprocessing with various preprocessing techniques being implemented to guarantee the integrity and consistency of the data.

Data Cleaning, ensuring consistent data quality by standardizing formats and addressing inconsistencies and missing values.

Feature Engineering was used for the improvement of prediction accuracy with new features being developed, such as simplifying device information and extracting specific details from existing data.

For both algorithms scale_pos_weight was dynamically set as a function of the inverse class frequencies with a multiplier to give more emphasis on the minority class.

Normalization and Scaling with continuous features being normalized and scaled, guaranteeing practical model training and uniformity.

Categorical Encoding was achieved by converting the machine learning models to numerical formats, facilitating the processing of categorical variables.

Feature Selection was achieved by using a correlation heatmap as the sole method for feature selection, analyzing the relationships between features for the identification and the removal of highly correlated variables, preventing multicollinearity, at the same time allowing the retention of the most informative and independent features, ensuring that redundant data did not influence the model, streamlining the feature selection process while improving the model's performance with a single move.

The aforementioned steps set the foundation for an accurate and reliable fraud detection, ensuring the dataset is well-prepared for model training and evaluation, with the implementation section providing a more detailed explanation of the specific implementation details of these phases.

## 2.3 Optuna Framework

Optimizing the hyper-patameters in our system is Optuna, a robust hyperparameter optimization library, optimizing the efficacy of the catboost and LightGBM models. The whole process works by trial and error, being particularly well-suited for tasks that necessitate the efficient exploration of extensive hyperparameter spaces. (Akiba et al., 2019), being employed in autonomous search of the

optimal combination of hyperparameters, maximizing the AUC, thereby enhancing the model's generalization to new, unseen data.

The ability of asynchronous optimization, enabling the evaluation of multiple trials in parallel, expediting the hyperparameter search procedure and mitigating potential bottlenecks, is probably Optuna's most noteworthy advantage. Optuna was used in this study, monitoring multiple trials, each of which investigated distinct configurations for hyperparameters, including the learning rate, dropout rate, and number of layers in the neural network, as well as the number of leaves, maximum depth, and feature fraction in LightGBM, with the dashboard being incorporated in tracking the advancement of the hyperparameter optimization process. The user-friendly interface offered by this tool worked flawlessly in assessing the performance of each trial and visualizing the optimization results, facilitating the identification of the most suitable configurations for subsequent training and observing the impact of various hyperparameters on the model's loss function.

The subsequent examples were used as a demonstration of Optuna's ability in generating studies for both the CatBoost and LightGBM models:

| |
|---|
| **study =** optuna.create_study(storage=storage,study_name=f"catboost_prd_{CU_DT}",direction=maximize) |
| **study =** optuna.create_study(storage=storage, study_name=f"lgbm_prd_{CU_DT}", direction= maximize) |

***Table 3 – Optuna Initialization***

The hyperparameter tuning process was streamlined, enabling optimal configurations for both models to be identified with greater efficiency and precision. (Akiba et al., 2019) by the use of the Optuna's dashboard, contributing in a significant manner to the improved performance of the fraud detection pipeline, as being presented in the subsequent sections.

Optuna provides an intuitively understandable interactive dashboard with comprehensive insights into the process of hyperparameter optimization. Such a tool enables researchers and practitioners to present important aspects of their studies intuitively, which may be interpreted more easily and used for model refinement in a far more effective manner.

Hyperparameter Importance: The dashboard visualizes the relative importance of each hyperparameter regarding the optimization process. Thus, it is easy to obtain an idea of which parameter will play a most decisive role in the performance of the model to tune those first.

**Optimization Timeline:** The timeline plot shows how trials develop with time. That provides a good overview of the optimization process' development and, on that basis, lets the user know how efficient the process is, or whether enough trials have been done to get something meaningful from them.

**Best Trial Information:** The dashboard then highlights the best trial that occurred during the study, including the performance metric it achieved (objective value), along with the hyperparameter

configuration used. This makes it easier to pick out the best parameters for use or further experimentation.

**User-Defined Attributes:** The dashboard contains study-specific attributes where users can define and record metadata or, in other words, information related to the studies. This would further help and facilitate the better management and documentation of experiments, usually in complex and collaborative projects.


***Figure 1 - Optuna-dashboard***

## 2.4 High Level Pipeline

First, the pre-processing of raw transactional data is prepared for model training by cleaning, handling missing values, feature engineering, and encoding of categorical variables. In the case of CatBoost, it natively supports categorical features; hence, no special treatment is required. LightGBM uses encoding methods suitable for its structure. Fine-tuning of both models is performed with Optuna for hyperparameter optimization.

The training process encompasses techniques for dealing with imbalanced datasets, such as dynamically computed class weights, to ensure the balance of fraudulent and non-fraudulent transactions in the model. Performance metrics like AUC, recall, precision, and confusion matrices are recorded in PostgreSQL tables: experiment_results_cb for CatBoost and experiment_results_lg for LightGBM. These tables allow tracking the performance of each model through their training.

In production, Kafka facilitates the real-time processing of incoming transaction data. Raw data is ingested by Kafka producers, where preprocessing steps are applied before the data is passed to the models. LightGBM and CatBoost operate simultaneously to classify transactions into three categories.

These models identify suspicious transactions and then send them for further refinement in the labeling phase using a Random Forest algorithm. This adds to the analysis, hence bringing nuanced patterns in suspicious data that might not have been powerfully captured at the time of initial classification. Newly labeled fraudulent transactions are fed back into the training datasets for both models, allowing them to adapt to emerging fraud trends and maintain high detection accuracy.

The last stage of the pipeline is visualization of the results and insights through Apache Superset. This provides an intuitive dashboard for stakeholders to monitor key performance metrics, analyze classification trends, and gain actionable insights from the data. Superset is connected directly to the PostgreSQL database, thus allowing dynamic and real-time visualizations of the outcomes from the pipeline.

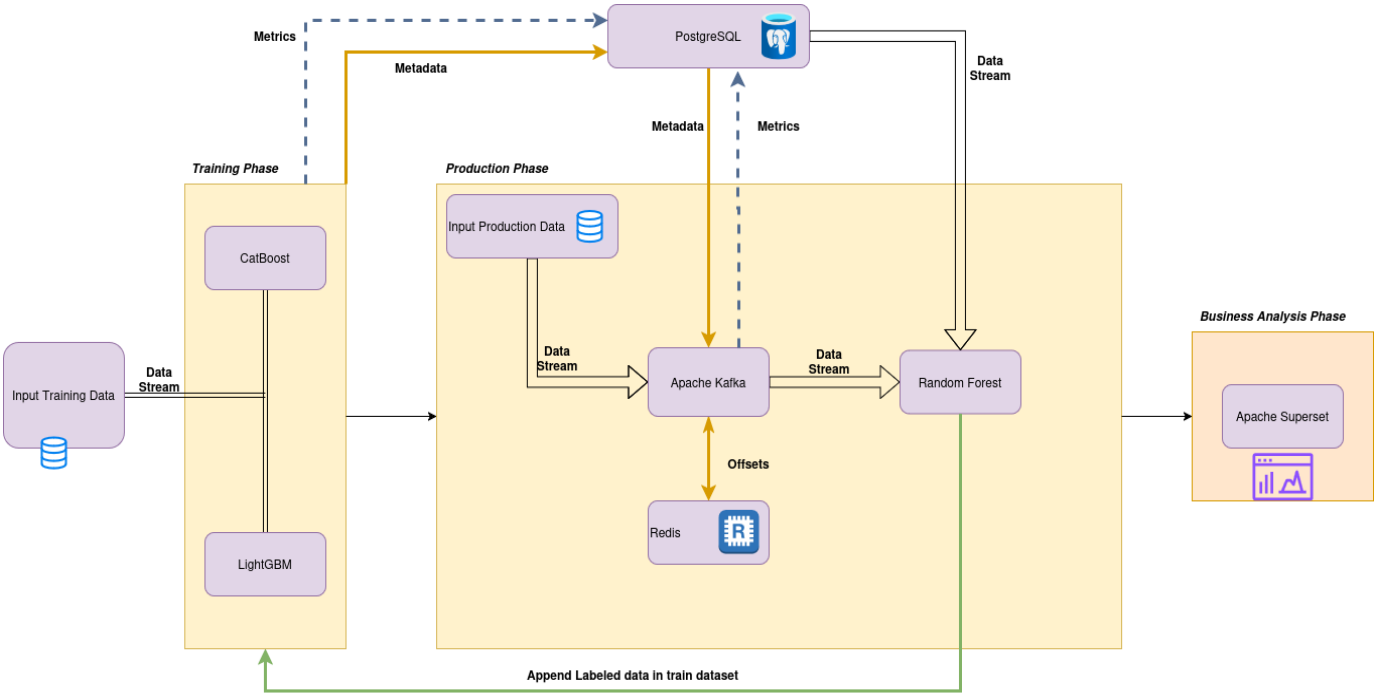



***Figure 2 – High Level Flow***

# 3 Implementation

For a more comprehensive management and support of CatBoost and LightGBM streams, a comprehensive repository established, containing numerous critical folders, each with a distinct role in the development, training, and deployment processes.

## 3.1 Overview of the Repository Structure

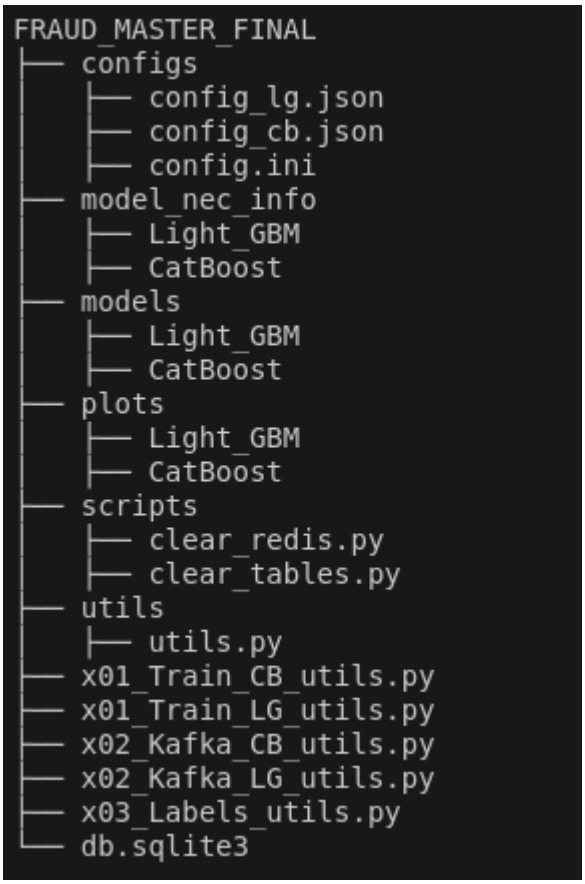



***Figure 3 – Project Structure***

- **Configs** containing configuration files that contain critical parameters such as the number of Optuna trials, data paths for training and production, bulk sizes, and so on, enabling effortless modifications to various configurations.
- **Data** includes the training datasets used to construct and train machine learning models, guaranteeing that all training data is conveniently located and readily accessible.
- **Model_Nec_Info**, storing essential outputs from the training process, including model visualizations, summaries, lists of features used, etc, aiding in model evaluation and documentation.

- **Models** with the best-performing models are saved after training, making future deployment and further analysis easy.
- **Plots**, containing visual outputs, such as precision-recall curves and confusion matrices, help in the assessment of the model's performance and understanding the data.
- **Scripts**, including utility scripts for maintenance tasks, such as cleaning Kafka offsets in Redis and clearing PostgreSQL tables, to keep the database optimized.
- **Source** with the models processing production data from this folder in real-time, ensuring their evaluation of new, unseen data.
- Finally, **Utils**, including reusable functions and utilities that support tasks like data preprocessing, model training, and evaluation, fostering code reuse and enhancing maintainability across the project.

## 3.2 Preprocessing Phase

The first step of the implementation process was importing the paths and parameters from a JSON configuration file (config_cb.json / config_lg.json), which contained key parameters such as the data path (DATA_PATH), the number of optimization trials (N_TRIALS), batch size (BATCH_SIZE) and the number of folds for stratified folding (NFOLDS). These parameters ensured a structured and efficient training process. Python libraries such as NumPy, Pandas, Seaborn, and Matplotlib were utilized for handling and visualizing the data, providing a comprehensive toolkit for the implementation.

The CodeCarbon library was integrated as an efficient way of tracking CO2 emissions during the training process, keeping in line with the project's emphasis on sustainability, and the monitor and mitigation of the environmental impact was deemed essential, after considering the high computational demands of training deep learning and tree-based models, aligning with the broader goal of promoting environmentally responsible AI development.

The training dataset was imported using a csv file, while extracting the isFraud column as the target variable and the remaining columns as features or inputs, both the CatBoost and LightGBM algorithms, aiming to build a comparable methodology regarding data preprocessing, with column names being standardized by converting them to lowercase after the data is loaded, guaranteeing consistency throughout the procedure, while at the same time recording metadata, including column names and data types, into a database for future reference and easy tracking, facilitating the documentation of the features employed during model training and ensures consistency in dataset analysis by both approaches. Several essential preprocessing steps are incorporated by the preprocess_data function, standardizing, cleaning, and enhancing the dataset, ensuring the data fed into the models is robust and reflecting the underlying patterns necessary for effective fraud detection.

### 3.2.1 Data Logging

In this study, an Excel workbook is generated to compile key insights about the dataset at various stages of preprocessing. This report serves as a summary of the dataset's characteristics and transformations, rather than a comprehensive log of all data modifications. It provides a high-level overview to enhance understanding and transparency of the dataset's key attributes. The report includes the following sections:

**Dataset Shapes:** Record the initial and post-processing dimensions (number of rows and columns) of the dataset, for reference. This section captures a picture of the progress of the dataset throughout the course of preprocessing.

**Prevalence of Devices:** Among the set, the distribution of devices is summarized, depicting the counts of each device in their raw state. It gives a great idea of the presence of different devices within the dataset.

**Overview of Missing Data:** Missing values are summarized for each feature in terms of percent. This provides an understanding of which features have a large amount of missing data that might need special handling.

**Removed Features:** Features removed during preprocessing, because they were redundant or irrelevant, are listed. This section identifies columns not included and ensures clarity on changes made to the dataset structure.

**Email Domain Insights:** A split is given of the most frequent domains occurring in the email data set. This part provides an overview of the most common domains linked with transactions, which may give a trend or pattern.

**Transaction Amount Summary:** Descriptive statistics for transaction amount is provided, detailing mean, median, and range. This gives the numerical nature of the actual features of the data set.

The naming convention for this Excel workbook is dynamic to allow immediate identification of the date and time of processing. This summary report provides a quick and accessible overview of the important insights in the dataset, supporting transparency and data-driven decision-making.

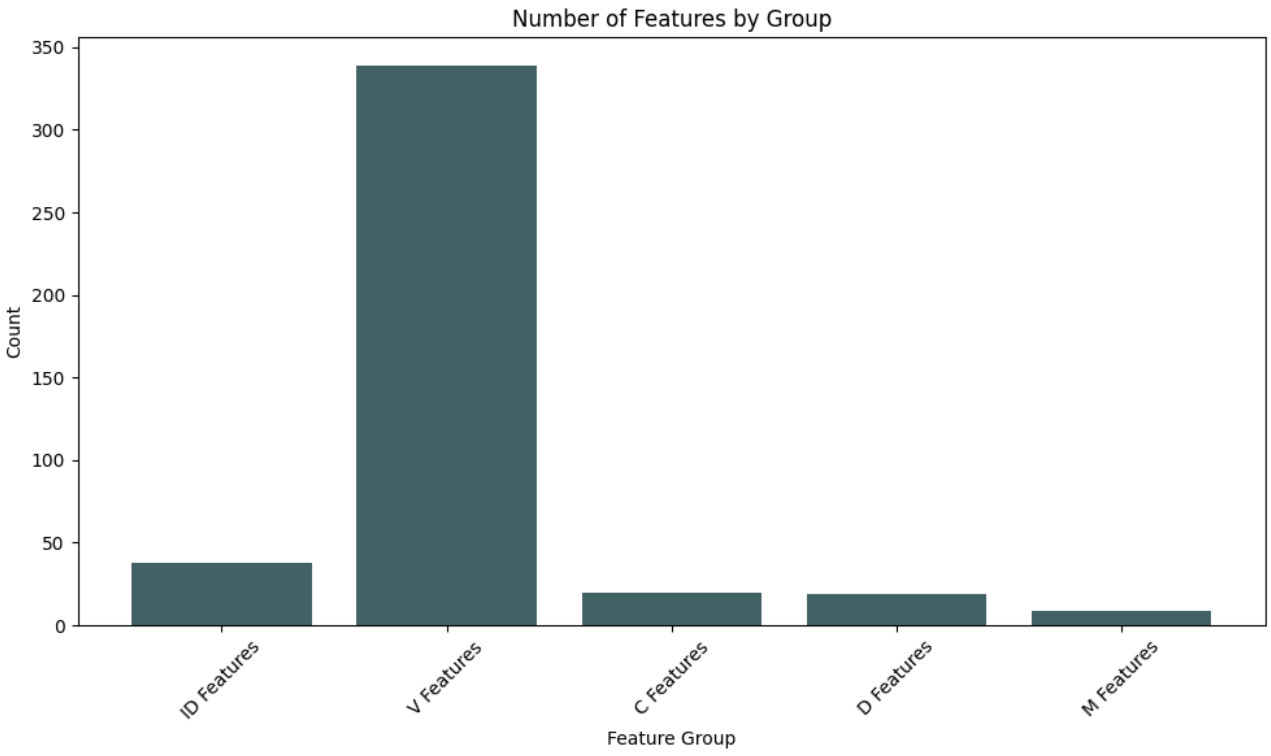



*Figure 4 - Number of Features by Group*

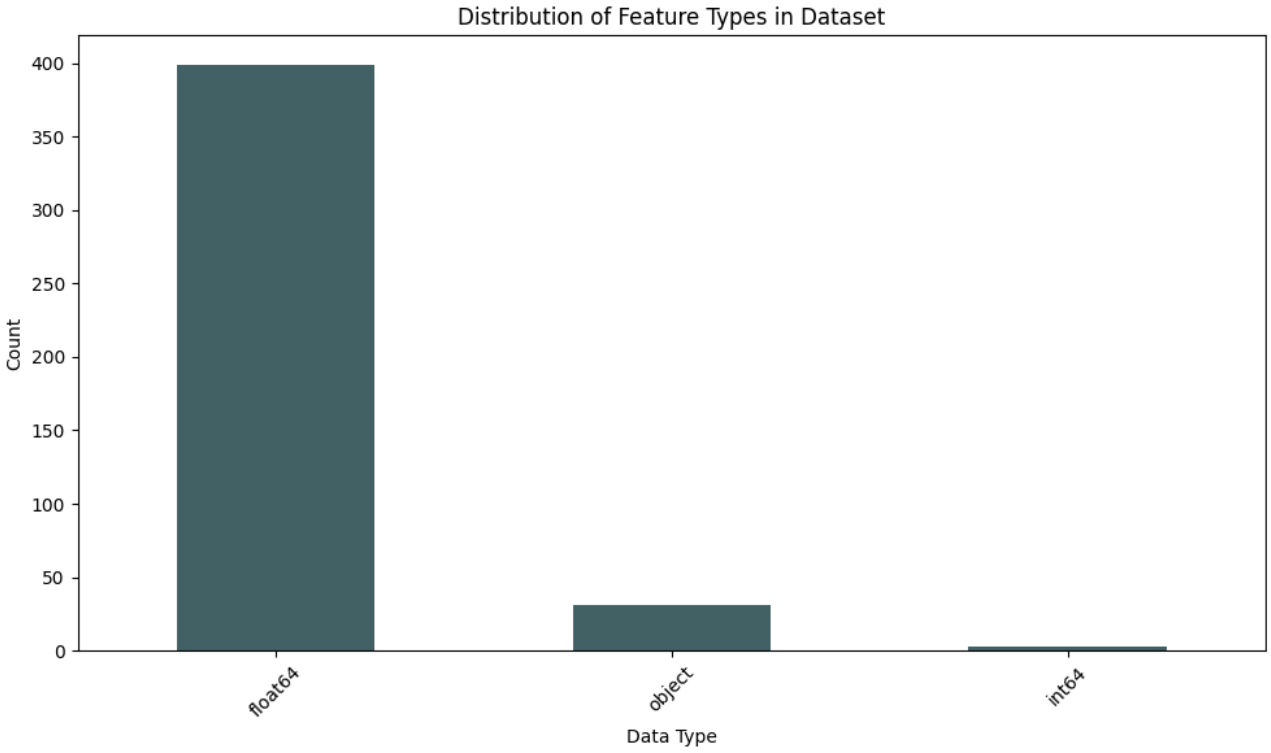



*Figure 5 - Distribution of Feature Types in Dataset*

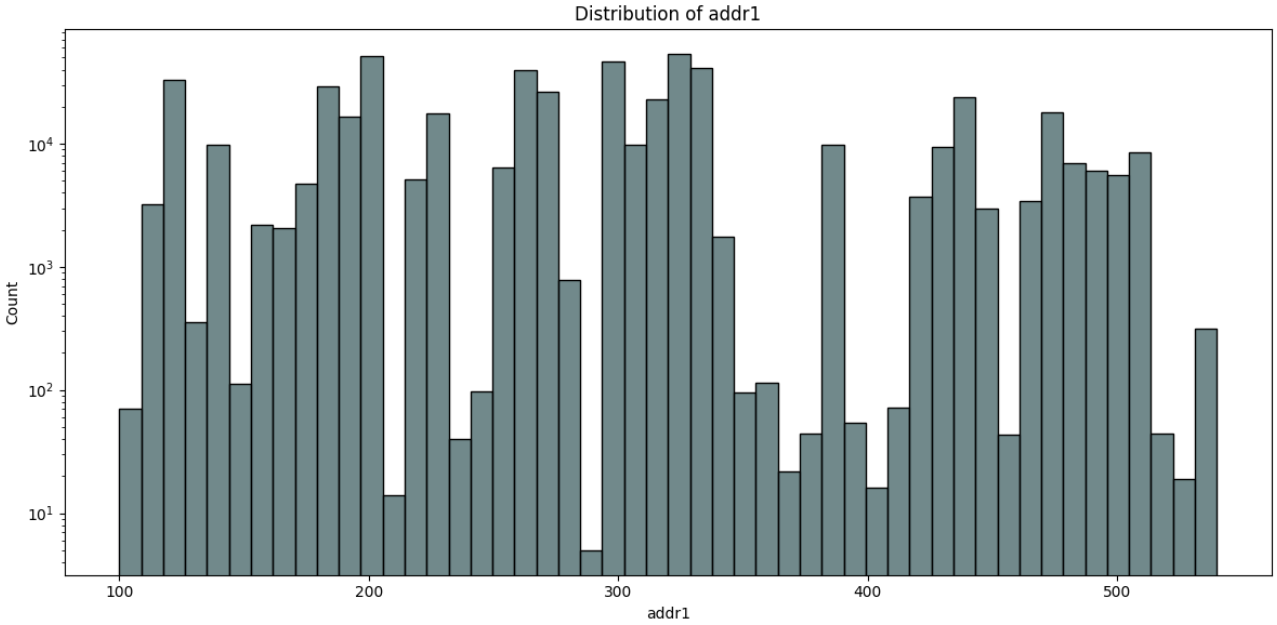



*Figure 6 - Distribution of addr1*

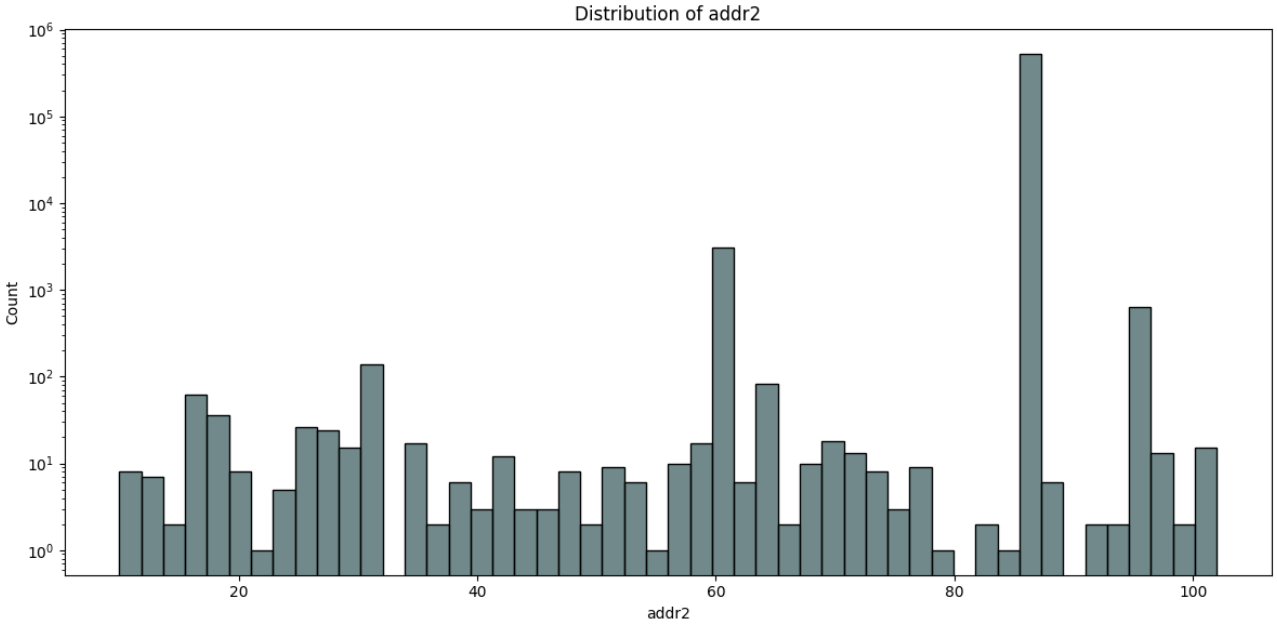



*Figure 7 - Distribution of addr2*

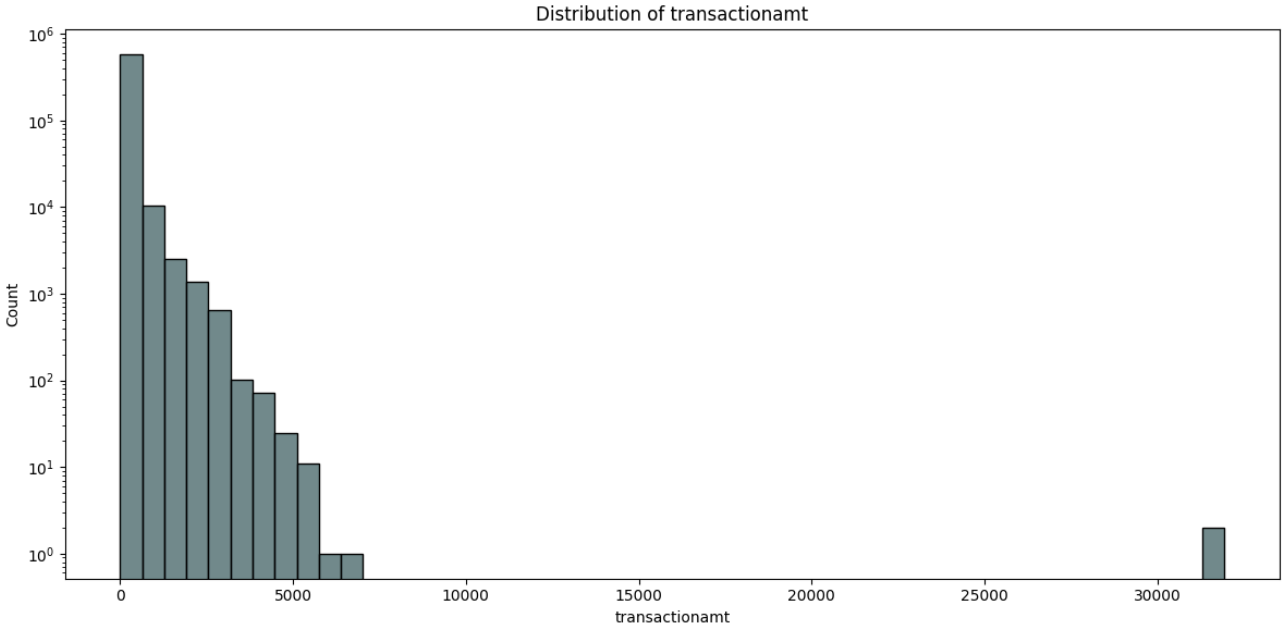



*Figure 8 - Distribution of transactionamt*

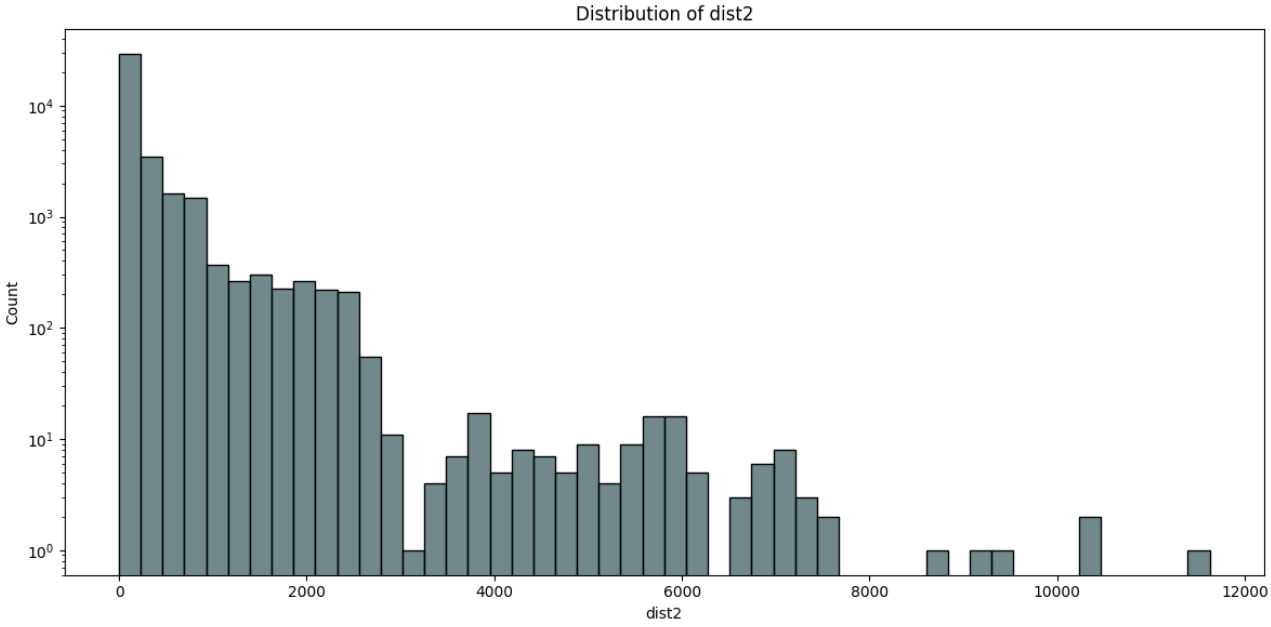



*Figure 9 - Distribution of dist2*

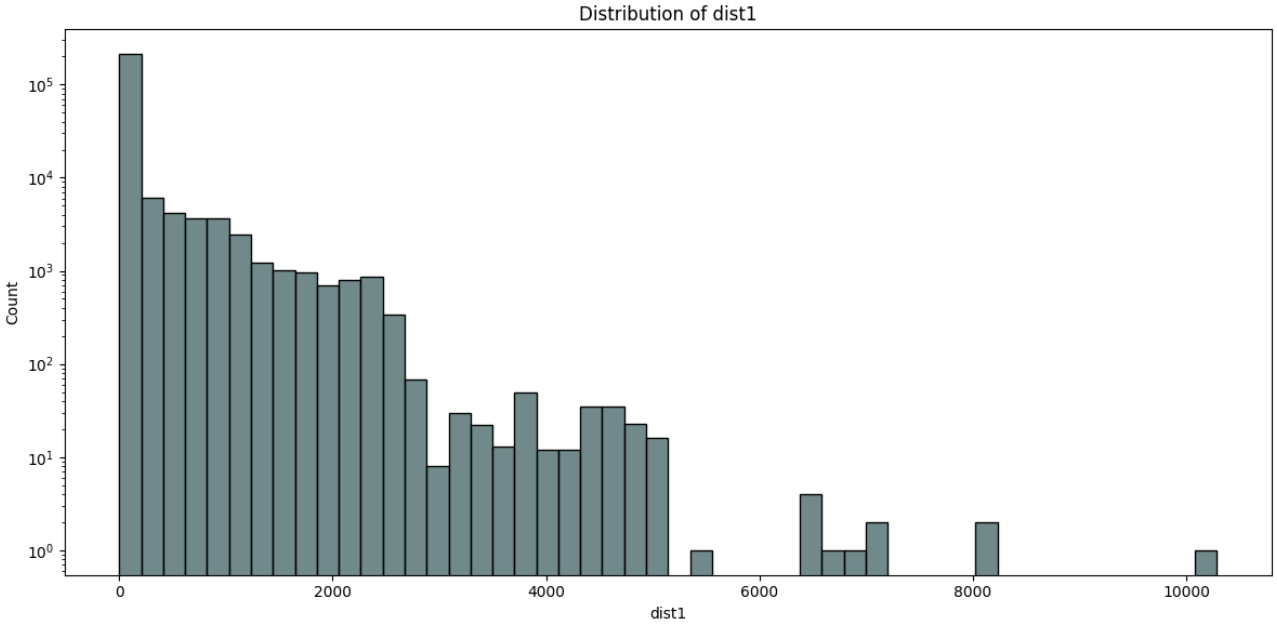



*Figure 10 - Distribution of dist1*

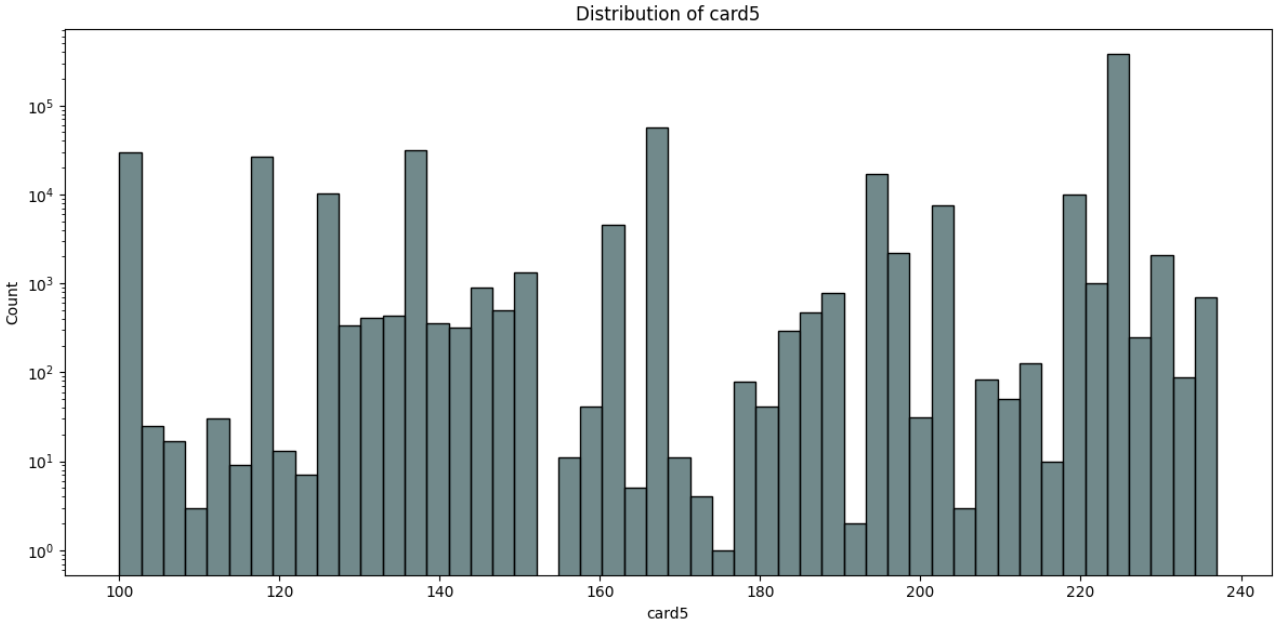



*Figure 11 - Distribution of card5*

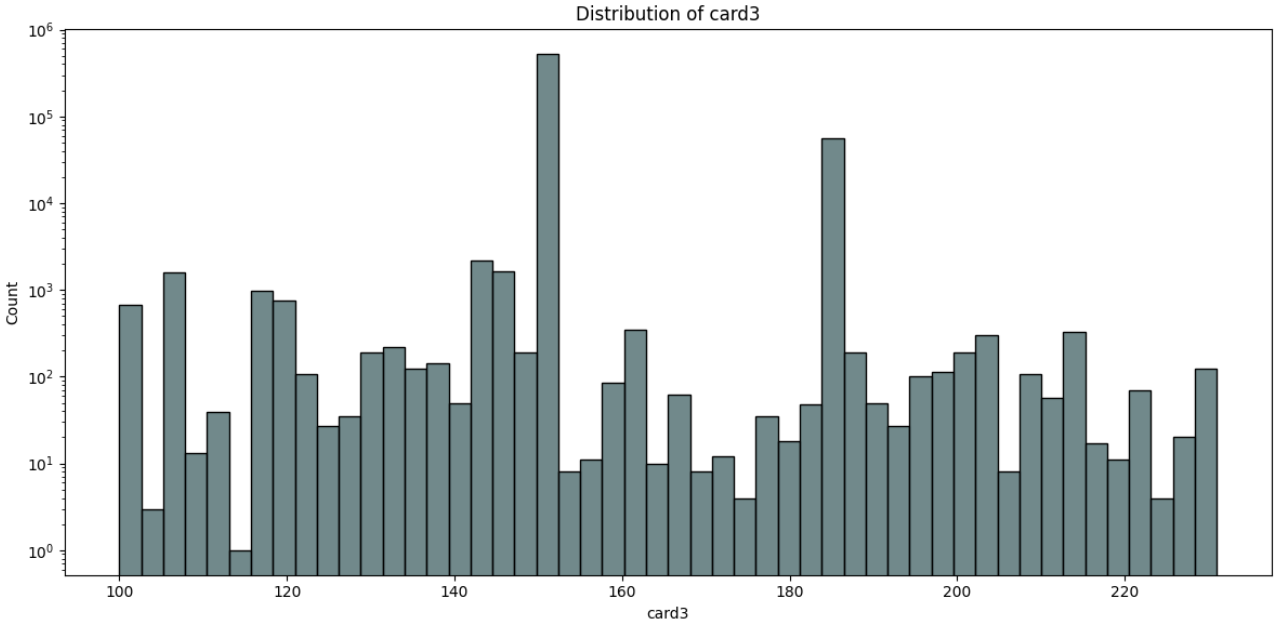



*Figure 12 - Distribution of card3*

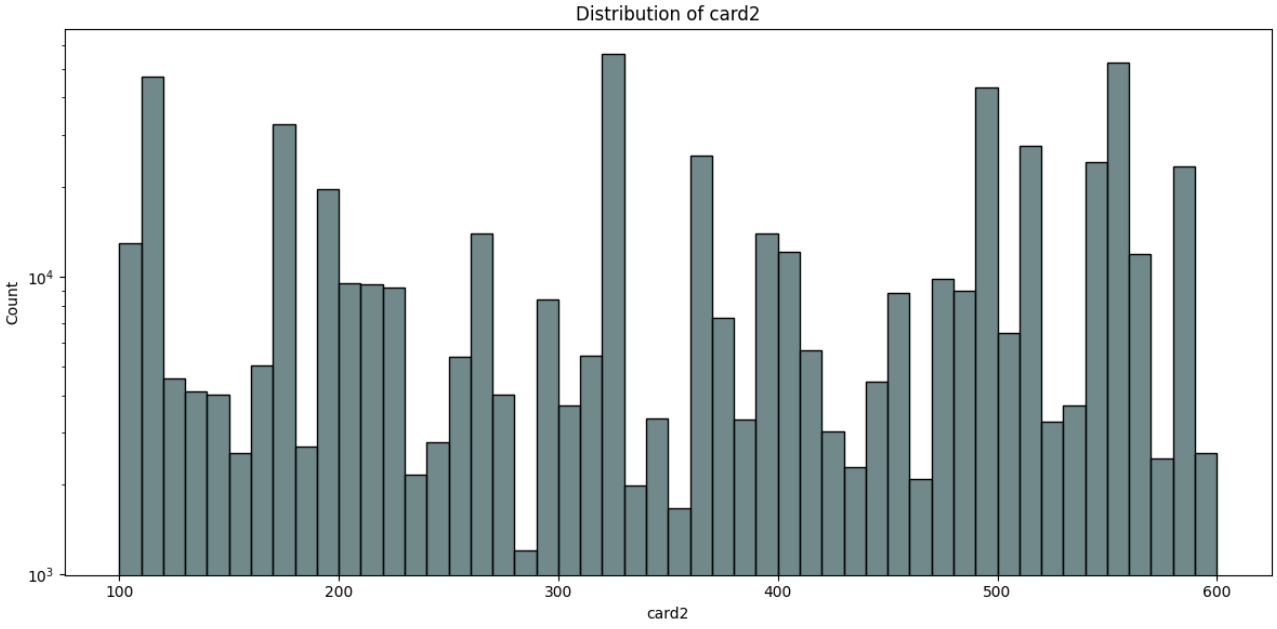



*Figure 13 - Distribution of card2*

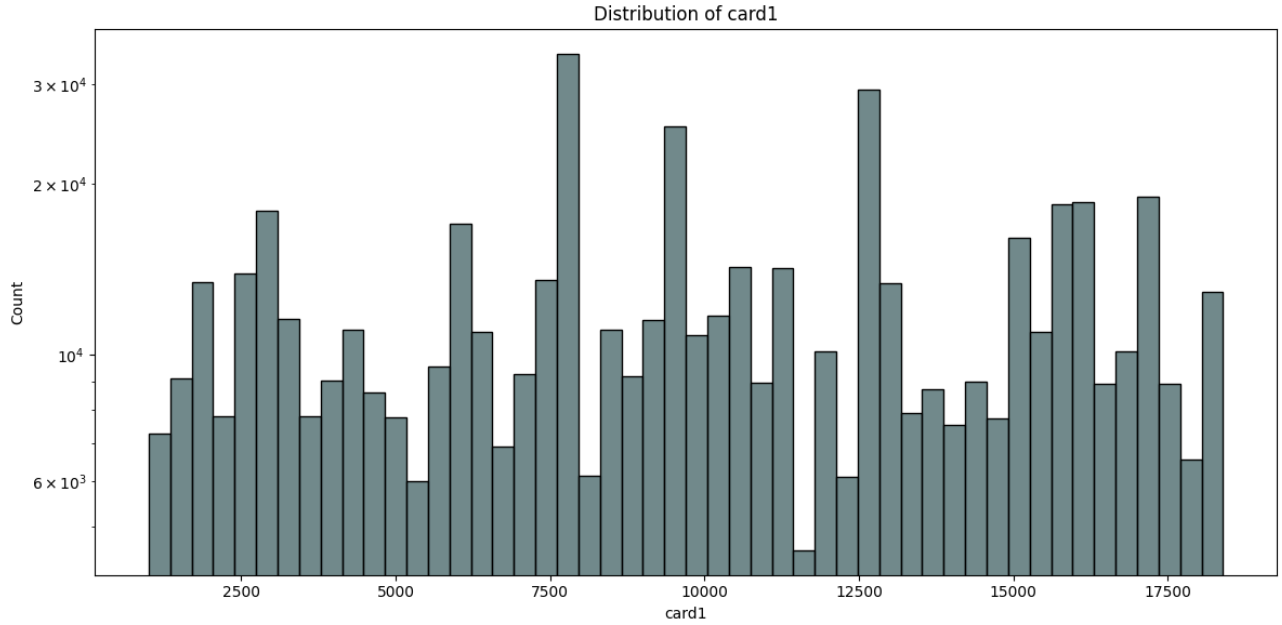



*Figure 14 - Distribution of card1*

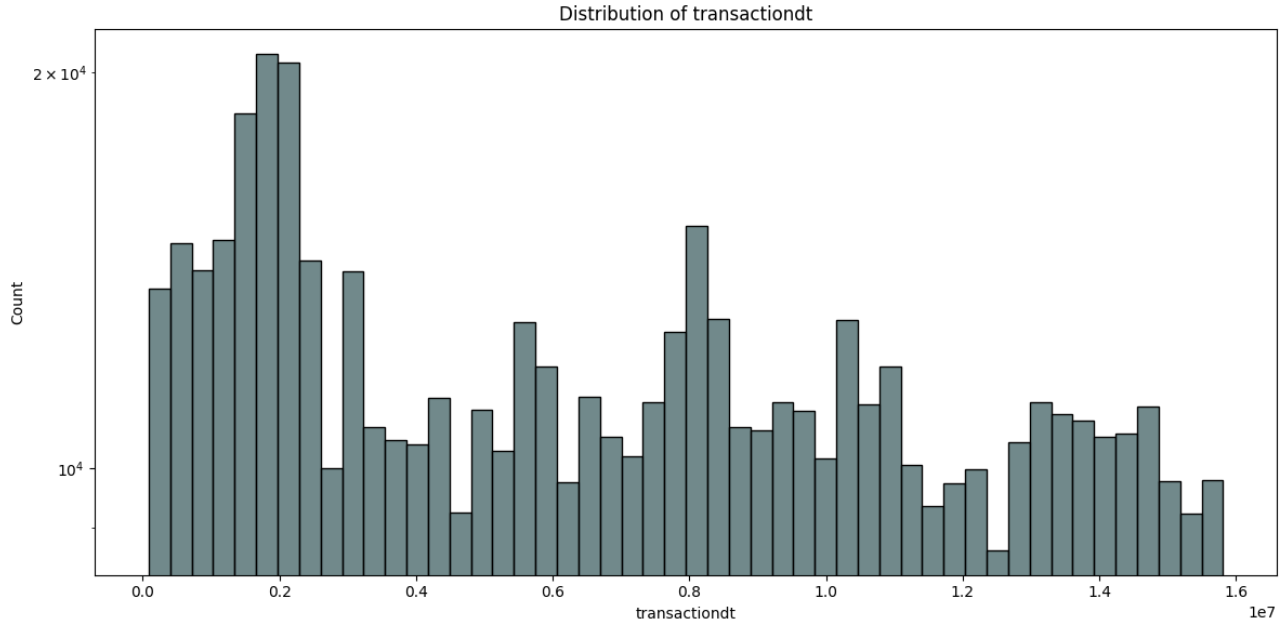



*Figure 15 - Distribution of transactiondt*

### 3.2.2 Handling Missing Values and High Repetitive

This is a very fundamental step in the preprocessing pipeline, since poor handling of missing values might lead to biased or incomplete models. The count and percentage of missing values for each feature are calculated and logged for transparency in the quality of the dataset. Features with a high proportion

of missing values (above 90%) are flagged as candidates for removal. This threshold ensures that features contributing minimal information due to excessive missing values are excluded, hence reducing noise in the dataset.

When working in training mode (MODE='TRN'), a bar plot is created to visualize the percentage of missing values across features. This visualization is saved for easy interpretation and serves as a diagnostic tool for assessing the completeness of the dataset.

It also identifies columns that demonstrate very low variability, usually those dominated by a single value. For example, features where more than 90% of the values consist of the same value—features that are considered here as adding little informational value to the model—are removed. These columns, very commonly known as "redundant features," are excluded to ensure the dataset remains compact, efficient, and focused on informative predictors.

The pipeline then proceeds to handle missing data in the retained features through imputation strategies specific to each feature type. Numerical features will be imputed with their medians, and categorical ones with their most frequent value. These imputation methods preserve the integrity of the dataset and ensure compatibility with downstream machine learning models.

The preprocessing pipeline ensures the quality of the used dataset, optimizes it for training by the systematic handling of missing values and redundant features. This reduces the possibility of bias and inefficiency in the model.

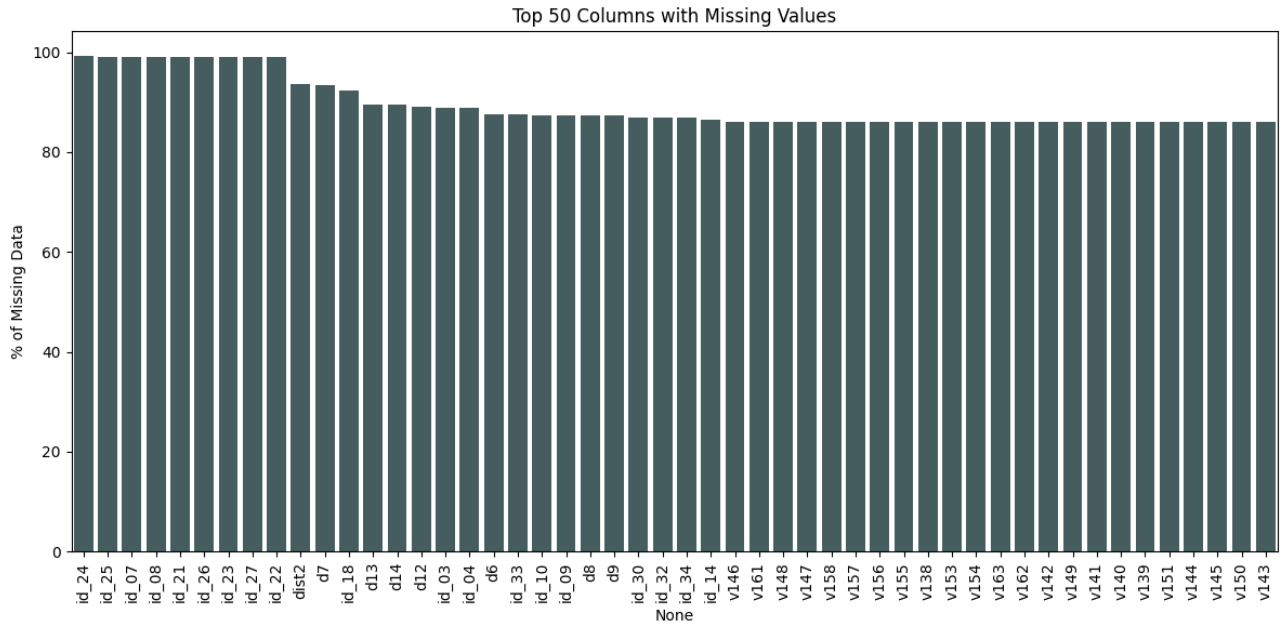



***Figure 16 – Percentage of Missing Values by Feature***

### 3.2.3 Device Information Transformation

The raw device data are usually very noisy, coming from various heterogeneous sources. If not properly handled, this noise will mask the meaningful patterns in the data, which increases the risk of overfitting and reduces the predictive accuracy of the model. Therefore, in this pipeline, a special care has been taken in using the mapping dictionary that helps to normalize the different representations of device manufacturers into consistent categories.

The mapping dictionary is also applied to the deviceinfo column in order to harmonize the representation of manufacturers of devices. For example, "SAMSUNG,","SM,", and "GT-" are mapped to "Samsung." This way, the model may treat these variants as one and retain their predictive value while not adding too much complexity. Devices with fewer than 200 occurrences are replaced by "Others," reducing the impact of rare categories that may otherwise contribute to noise and reduce model generalization. This threshold ensures that the dataset focuses on dominant patterns while managing data sparsity.

Moreover, the pipeline transforms information related to a device into general categories by grouping the devices under major brands—for example, Samsung, Apple, and Motorola. Such transformation captures trends that surround specific manufacturers, generally indicative of user behavior—for example, some device types may have a special spending pattern or relate to transactions with anomalies in them, hence making this an important transformation while detecting fraud.

If the pipeline is in training mode (MODE='TRN'), it creates visualizations of boxplots to analyze the relationship between device brands and transaction amounts. These boxplots give valuable insight into how transaction behavior varies across device categories, adding another layer of interpretability.

Such cleaning and standardization of the device-related data only serve to increase their quality, but the transformation also helps the model learn meaningful patterns. Simplifying input data while retaining its predictive value reduces complexity, ultimately leading to improved model performance and generalization.

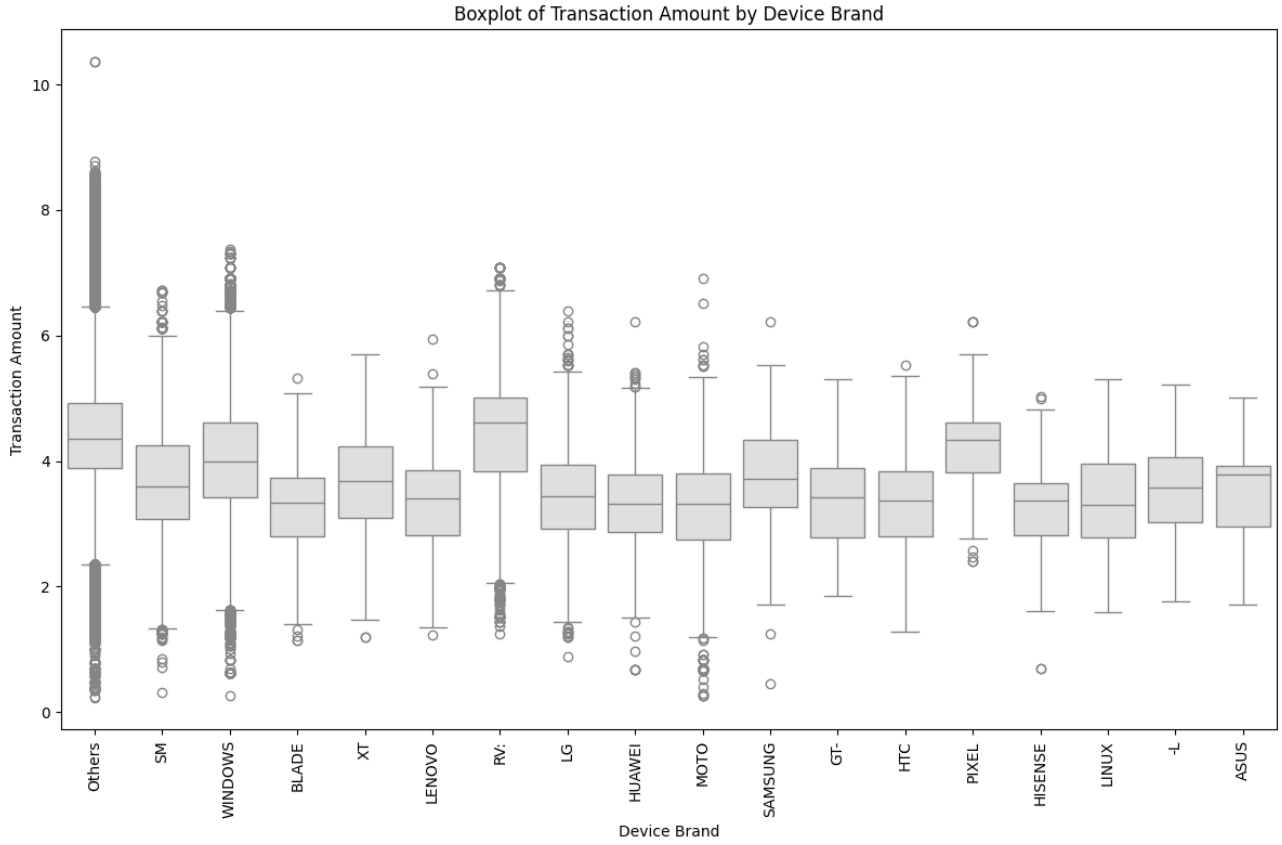



***Figure 17 - Transaction Amount by Device Brand***

### 3.2.4 Transforming Email Domains and Temporal Features

Email domain features, such as p_emaildomain and r_emaildomain, are transformed by mapping them according to a predefined mapping that groups the domains into bigger categories: google, yahoo, and microsoft. Suffixes are also extracted to distinguish U.S.-based from international domains, adding contextual information that is often very indicative of user identity.

Temporal features derived from transaction timestamps capture important time-based behavioral patterns. The pipeline calculates actual transaction dates relative to START_DATE and extracts components such as month, week, and day of the year. Such features allow for the identification of fraudulent patterns that evolve over specific time intervals.

### 3.2.5 Feature Engineering

Feature engineering enriches the dataset by creating new, informative features. Composite features, such as userinfo, derived from card1, card2, and card3, and full_address, derived from addr1 and addr2, are created to capture potential correlations between related features. The engineered features provide

more context for fraud detection algorithms. By combining related attributes, these engineered features help the model uncover complex interactions that might be indicative of fraudulent behavior (Heaton, 2016).

Some of the categorical features are also encoded with frequency encoding, replacing values by their frequency in the dataset. This would help bring out the patterns related to rare or frequent occurrences of features. The transaction amounts in the dataset (transactionamt) are transformed with a natural logarithm (log1p) to reduce skew and normalize their distribution, generally reducing the effect of extremely large outliers.

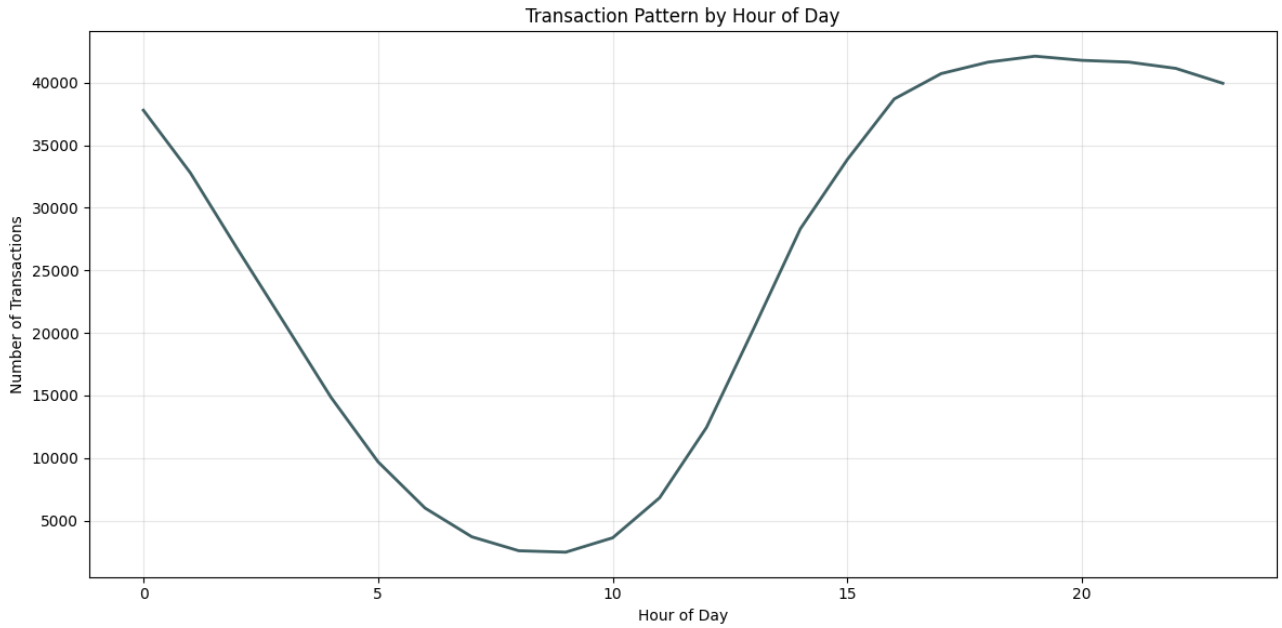



***Figure 18 - Transaction Pattern by Hour of Day***

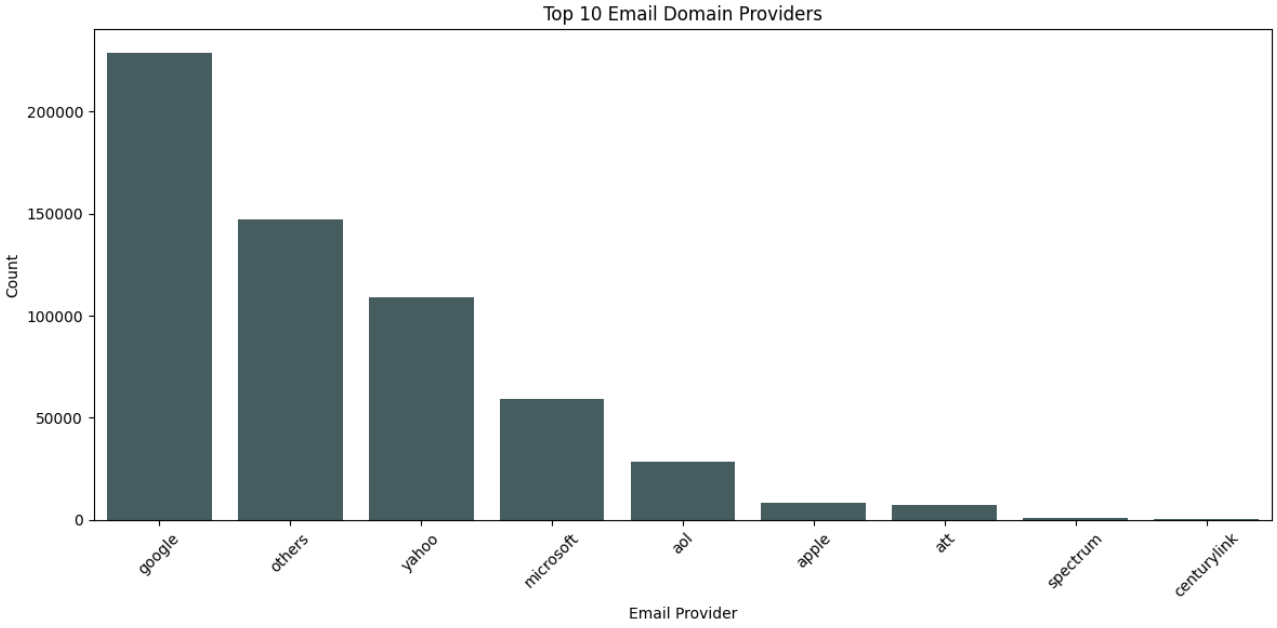



*Figure 19 - Top 10 Email Domain Providers*

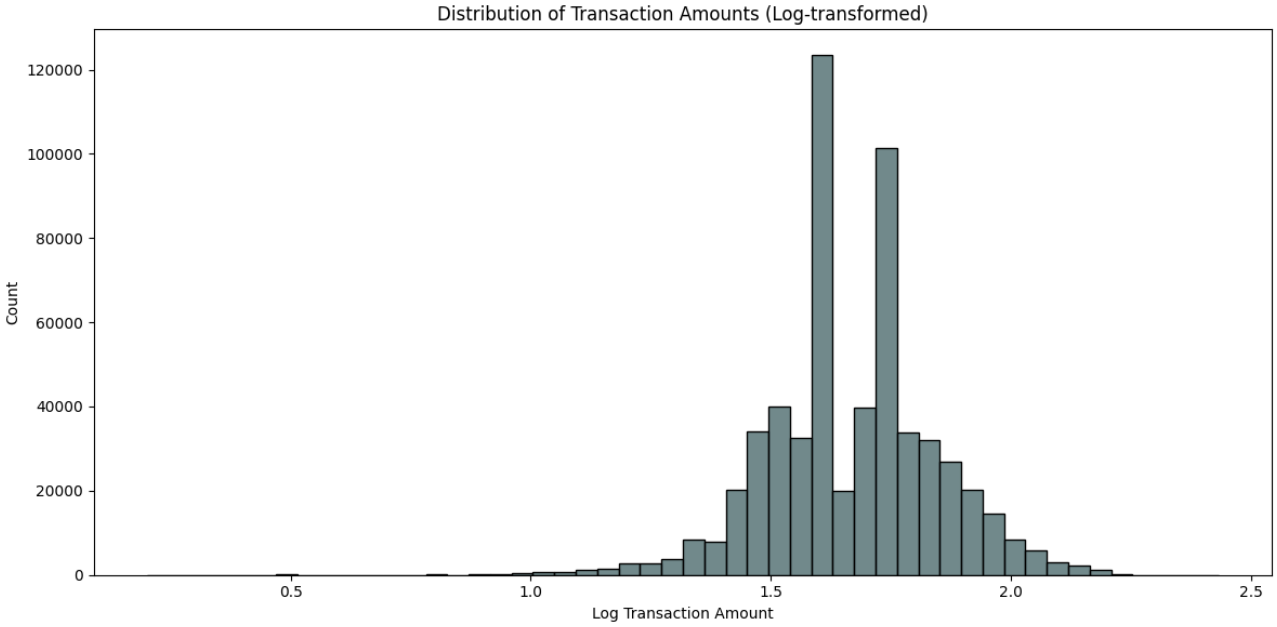



*Figure 20 - Distribution of Transaction Amounts (Log-transformed)*

### 3.2.6 Handling Categorical Features: Label Encoding

Categorical features, such as deviceinfo and others, must be converted into numerical formats for models like LightGBM, which require numerical inputs for training. This conversion process is

commonly achieved through label encoding, where a mapping function transforms unique categories into distinct numerical values. Formally, the label encoding process can be represented as:

$$f(x): C1, C2, \ldots, Cn \longrightarrow \{0, 1, 2, \ldots, n-1\}$$

with C1, C2,…, Cn being the unique categories in the categorical feature xx, and nn is the total number of unique categories with the function f(x)f(x) assigning an integer to each category.

For example, having a column like deviceinfo with categories such as "Samsung," "Motorola," and "Apple," they will be mapped to a unique integer, such as the following example:

$$f(\text{deviceinfo}) = \{\text{"Samsung"}:0, \text{"Motorola"}:1, \text{"Apple"}:2, \ldots\}$$

This step is essential because machine learning algorithms cannot process textual data and rely on numerical representations to learn patterns. Label encoding is particularly advantageous for ordinal features, where categories have a natural order (e.g., "Low," "Medium," "High"), as the numerical mapping aligns with their inherent ranking.

However, when applied to nominal features, which lack an intrinsic order, label encoding can inadvertently introduce arbitrary numerical relationships. While this may pose challenges for certain models, such as linear regression, tree-based models like LightGBM are inherently robust to such numerical ordering. These models split data based on thresholds rather than interpreting numerical relationships, allowing them to handle encoded categorical data effectively without bias.

CatBoost, in contrast, eliminates the need for label encoding entirely by offering built-in support for categorical features. Instead of assigning arbitrary numerical values, CatBoost employs advanced statistical techniques and combinations of categorical values to create meaningful representations. This approach reduces the risk of bias and enhances the model's ability to generalize, particularly when dealing with high-cardinality categorical features (Prokhorenkova et al., 2018). As a result, CatBoost's method not only simplifies preprocessing but also boosts performance on datasets with diverse and complex categorical attributes.

### 3.2.7 Split Dataset - Pipeline

Numerical and categorical features are passed through separate pipelines, each designed for the particular needs of that type of feature:

**Numerical Pipeline:**

The numerical pipeline begins with a FeatureExtractor that identifies and selects the numerical attributes in the dataset. It imputes missing values for these attributes using the median strategy—a robust approach that minimizes the influence of outliers, ensuring the imputed values don't skew the data distribution. After imputation, numerical features are standardized using StandardScaler, which removes the mean and scales features to unit variance. Standardization ensures that all numerical

features are on a comparable scale, which avoids the domination of the learning process by any single feature with a larger range. This step improves the stability of the model and allows it to converge faster during training.

**Categorical Pipeline:**

The categorical pipeline begins with a FeatureExtractor that will select only categorical attributes; it replaces missing values of these features using the strategy of using the most frequent value—imputes missing values with the mode of the column. Together, these steps preserve data integrity and ensure that imputed values make sense in context for the categorical columns.

The transform_features function consolidates these pipelines, which are then applied to the dataset. The pre-processing framework stays adaptive to a variety of datasets and preprocessing requirements while applying transformations consistently by modularizing the pipelines.

To deal with large datasets, the preprocessing pipeline uses a chunked processing approach. Instead of trying to process the whole dataset at once, which might even cause system memory to run out, the dataset is broken down into manageable chunks. Each chunk is then independently passed through the numerical and categorical transformation pipelines using the transform_features function.

Such chunked processing has a few benefits for large datasets. In general, by breaking a dataset into smaller subsets, the pipeline avoids memory bottlenecks, thus the algorithm is compatible with systems having limited memory capacity. These are applied uniformly to the whole dataset, while being processed in chunks, to maintain consistency and integrity of the final recombined data. This ensures that the resultant dataset is cohesive and ready for downstream machine learning tasks.

After transformation of all chunks, they are concatenated together into a single dataset. This last step is performed to make sure that the data is ready for downstream machine learning tasks and that all features are scaled, imputed, and standardized accordingly.

### 3.2.8 Handling Imbalanced Data

Class imbalance was the most crucial step of the entire process, considering the huge amount of variation in the number of instances between the legitimate and fraudulent transactions. This would amount to a severe problem since models could easily get biased towards predicting the majority class, leading to overfitting or a low recall for the minority class. This, in banking, is very risky, especially when a model fails to give a higher recall, as misclassifying a fraudulent transaction as genuine might cause great financial and reputational loss. Therefore, ensuring the model's ability to effectively identify fraudulent transactions was of paramount importance.

The dataset, by default, is imbalanced, with a significantly higher number of legitimate transactions compared to fraudulent ones. This imbalance poses a challenge for machine learning models, as they tend to become biased toward predicting the majority class (legitimate transactions). Such bias can

result in poor performance in detecting the minority class (fraudulent transactions). Addressing this issue is crucial to ensure the model can effectively differentiate between fraudulent and non-fraudulent transactions.

For both models, the imbalance was addressed by calculating weights for the classes based on their relative frequencies in the dataset. Fraudulent transactions, being fewer in number, were given proportionally higher importance. These class weights were incorporated into the model's training process using the scale_pos_weight parameter. This ensured that errors in predicting fraudulent transactions were penalized more heavily than errors for legitimate transactions, improving the model's ability to identify fraud.

Both implementations emphasized the recall metric, which measures the proportion of actual fraudulent transactions that the model correctly identifies. In fraud detection, maximizing recall is crucial because missing fraudulent transactions (false negatives) can lead to significant financial or reputational losses. By prioritizing the minority class, the models were optimized to reduce false negatives and improve recall.

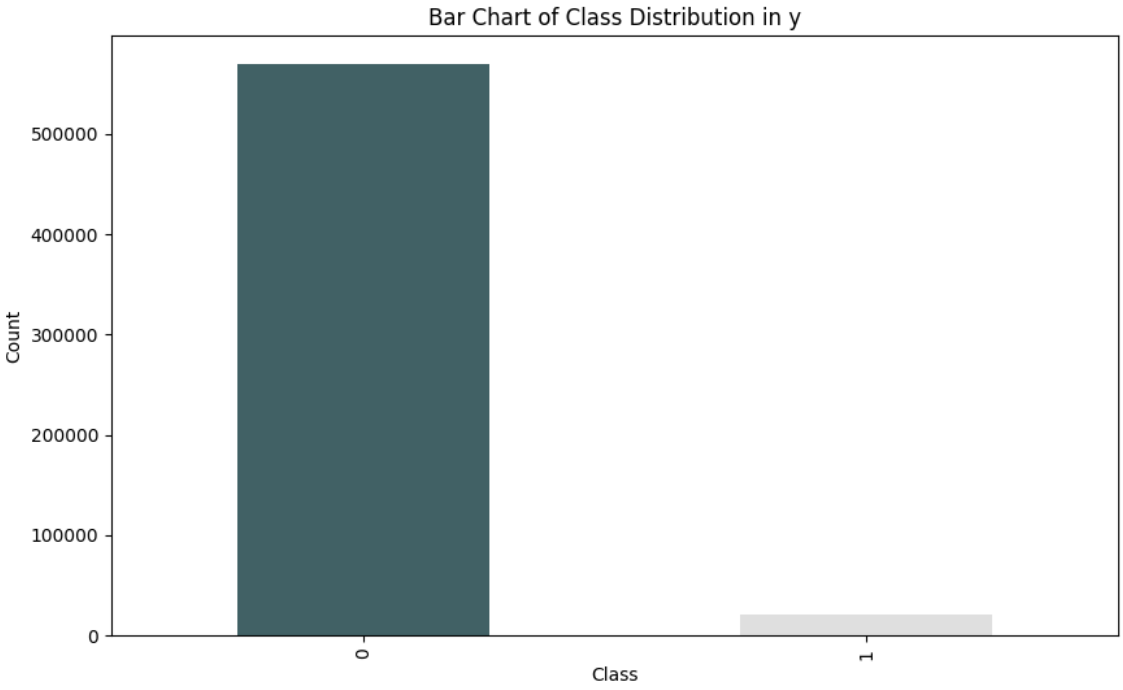



***Figure 21 - Class Distribution (1/2)***

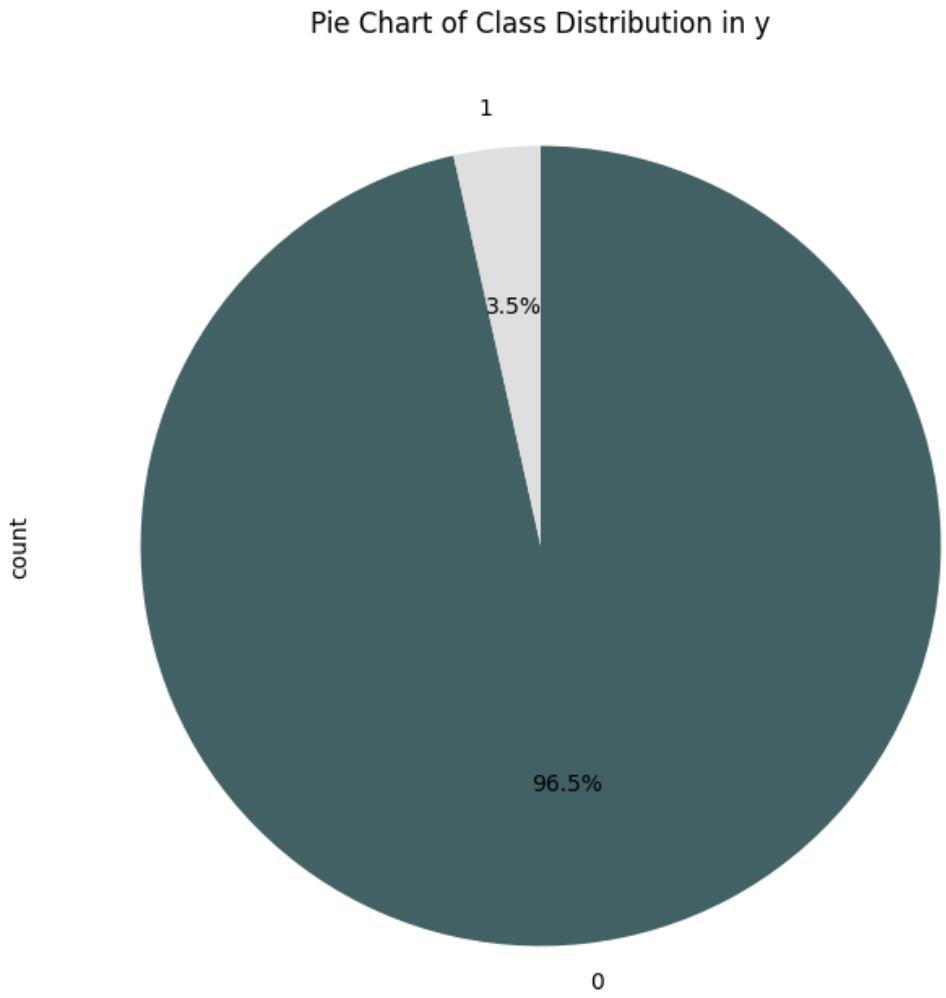



***Figure 22 - Class Distribution (2/2)***

### 3.2.9 Features Scaling

The last step in the chain of the preprocessing pipeline is scaling numerical features using the StandardScaler. This utility removes the mean of each feature and scales those features to a standard range, in order to make all numerical features more equal, preventing overpowering features with large-value ranges from dominating the learning process. This further reduces the possibility of any bias in the model or giving partiality to one of the input features.

Application of StandardScaler standardizes the preprocessing, considering all these models would be blended with other models, and features in large numerical difference may make some impacts to model interpretability. Standardizing features also provides better optimization stability and promises faster convergence when training, especially for numerical features that highly vary in scale.

This is standardized preprocessing that prepares the data for machine learning models quite nicely, as observed in James et al. (2013). It may not be strictly necessary when dealing with tree-based models, such as LightGBM or CatBoost, but this makes the data cleaner and easier to understand, and it also better connects with other preprocessing.

### 3.2.10 Pipeline

LightGBM and CatBoost preprocessing pipeline takes raw data as input, making it ready for training. It takes as an argument the preprocess_data function processing the data and splits the categorical and numerical variables. In CatBoost, this uses native support for categorical variables without any encoding from other libraries, hence maintaining the relationship between such features. For LightGBM, the categorical features are encoded to make them compatible with the model's training process.

Stratified splitting ensures that the class distribution will be well captured across the subsets for a balanced distribution in both fraudulent and no-fraud transactions. The stratified split splits the data into training, validation, and testing subsets.

Because the target variable isfraud is highly imbalanced, class weights are computed dynamically depending on the frequency of each class in the training data. These weights have been added to both models during training so that the predictions are not biased toward the majority class. In fact, this approach gives a great boost in improving recall for the minority fraudulent class, which is so critical when it comes to fraud detection.

Memory control in the above processes involves deleting any intermediate dataset once it is no longer useful and a call to gc.collect() to make sure the garbage collector runs, so big memory allocations can be accommodated. For example, the data objects which could be temporary splits or/and folds are deleted instantly after usage for freeing the occupied resources.

The final models, with the hyperparameters tuned to the best, were trained on combined training and validation sets.

#### 3.2.10.1 CatBoost Hyperparameters

**Iterations**: It defines the number of boosting rounds. Range: 500 to 1000.

**Learning Rate**: Logarithmic scale search from 0.01 to 0.2 to balance training speed and accuracy.

**Depth**: Defines the depth of a tree. Values explored: from 4 to 8 to avoid overfitting but capture the complexity.

**L2 Leaf Regularization**: Controls overfitting, values optimized from 0.1 to 10.

**Bagging Temperature**: Regulates randomness of subsampling, values searched from 0.0 to 1.0.

**Random Strength**: Regularizes noise in split thresholds, optimized between 1e−71e−7 and 10.0.

**Scale Pos Weight**: Scales the loss function to account for class imbalance, calculated using class frequencies dynamically.

**Border Count**: The number of splits for numeric features, fixed at 128 for regular granularity.

The model has been trained on a GPU, which will significantly speed up training and optimization, especially for larger datasets or wide hyperparameter search spaces.

This setup will make sure that the powerful capabilities of CatBoost in handling categorical data and complex feature interactions are unleashed to their full potential, while being optimized for performance by means of a strong hyperparameter tuning process.

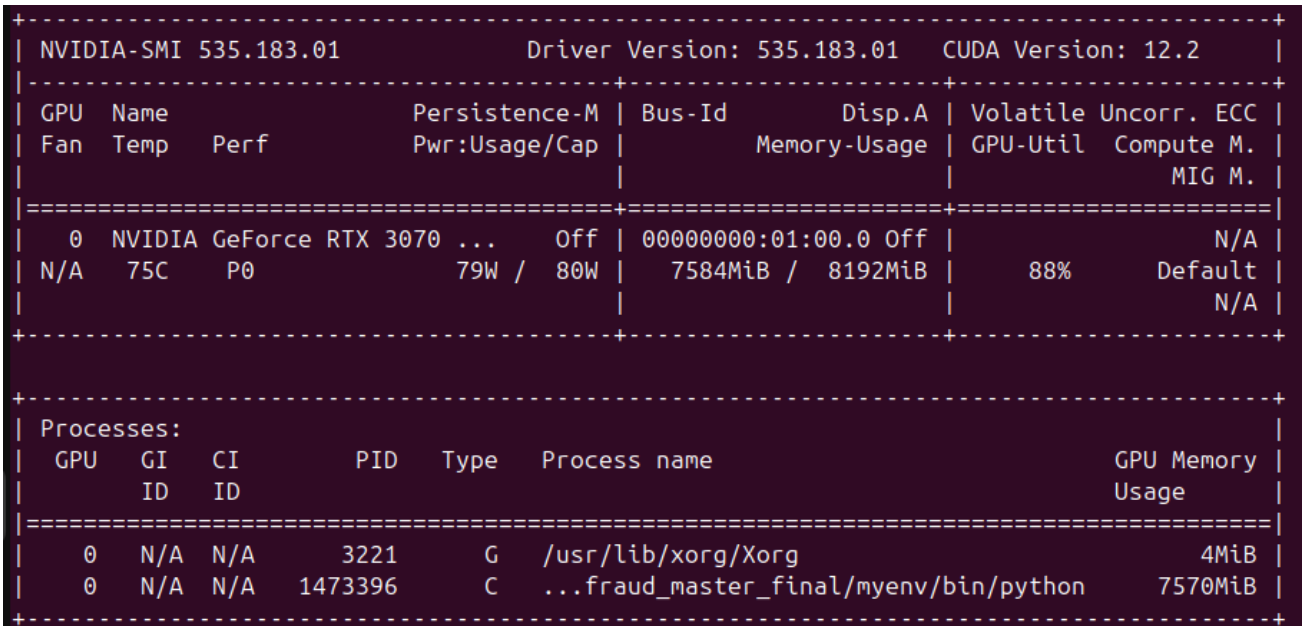



*Figure 23 - CatBoost GPU Utilization*

### 3.2.10.2 LightGBM Hyperparameters

**Learning Rate:** Step size in every optimization.

**Search Space:** Logarithmic scale from 0.03 to 0.1 because it offers a great trade-off between convergence speed and stability.

**Number of Leaves:** To define the complexity of every tree.

**Range:** 100 to 200.

**Feature Fraction:** Fraction of features used for each tree. Range: 0.3 to 0.8.

**Bagging Fraction:** Fraction of data samples used for each tree. Range: 0.3 to 1.0.

**Minimum Data in Leaf:** This ensures that leaves have at least this number of samples, thereby acting as a prevention against overfitting.

**Range:** 10 to 50.

**reg_alpha:** L1 regularization.

**reg_lambda:** L2 regularization.

**Range:** Both optimized between 0.01 and 10 in order to reduce overfitting.

**Scale Pos Weight:** Scales the contribution of the minority class to the loss function.

**Value:** Computed dynamically as a ratio of class frequencies.

**Max Depth:** Maximum depth of trees to prevent overfitting. Range: 8 – 15

**Max Bin:** It computes a number of bins for a continuous feature.

**Range:** 60 - 125.

The model has been trained with n_jobs=5. LightGBM can use full parallel processing on multiple CPU cores. This config greatly speeds up both training and evaluation by dividing the work in an efficient way and is particularly useful when working with big datasets or compute-heavy pipelines. This parallelism is very useful for hyperparameter optimization, where different combinations of parameters can be tested at the same time, saving a lot of time in finding the best model configuration. Besides that, multi-threading ensures that all computational resources of the system are utilized with minimal idle time, which increases overall efficiency.

## 3.3 Plotting Training Results

In the CatBoost and LightGBM scripts, the evaluate_model function is executed to start the evaluation phase following the training phase.

The confusion matrix (conf_matrix), a useful tool for visualizing a classification model's performance, is one of the important performance metrics that the evaluate_model function returns, with the summary of the important metrics being presented below.

1. **The Confusion Matrix** is a 2x2 matrix that provides a detailed breakdown of the classification results by showing the counts of:

| | |
|---|---|
| **True Positives (TP)** | the number of correctly predicted fraud transactions. |
| **True Negatives (TN)** | the number of correctly predicted non-fraud transactions |
| **False Positives (FP)** | the number of non-fraud transactions incorrectly predicted as fraud (Type I error) |
| **False Negatives (FN)** | which is the number of fraud transactions incorrectly predicted as non-fraud (Type II error) |

***Table 4 - The Confusion Matrix***

2. **Precision** is the determining factor of accurate positive predictions to all of the model's positive predictions. i.e., the proportion of predicted fraud cases that actually occurred.
3. **Recall** is the ratio of real fraud cases that the model correctly identified, or the number of real fraud cases that were correctly predicted to be fraud to the total cases.
4. **F1 Score** uses the precision and recall harmonic means, offering an impartial assessment of the model's performance, being helpful in situations where the dataset is unbalanced (like in fraud detection).
5. **Area Under the Receiver Operating Characteristic Curve**, or AUC-ROC, measures how well the model can differentiate between fraud and non-fraud across a range of thresholds with a higher AUC score indicating model performing better.
6. **AUC-PR (Area Under the Precision-Recall Curve)** is another threshold-based metric useful for imbalanced datasets, focusing on the trade-off between precision and recall.

CatBoost and LightGBM generate their predictions with the predict method. Then, probabilities of the fraud class are thresholded at 0.2 to differentiate between classes; that will be done with all models for consistency because this dataset is quite imbalanced. This ensures a standardized evaluation framework across models, aligning with best practices for handling imbalanced datasets (He and Garcia, 2009).

Then, it also presents in an interesting form what are the hyperparameters generated after the optimization using the pipelines and Table from the Rich library. These are both printed to the console for immediate review and outputted to a text file (e.g., lgbm_parameters_<date>.txt) in the model_info directory using PrettyTable. The mentioned practices ensure that both pipelines keep parameters and configurations accessible, thereby facilitating reproducibility, comparison, and documentation.

In both pipelines, the format_elapsed_time function is used to document the total training time in a human-readable format, helping assess computational efficiency and resource usage. The final models

are saved with save_model() function, ensuring they can be reused for future predictions or analysis without retraining. These steps collectively emphasize transparency, reproducibility, and clarity in documenting the training and configuration processes for both machine learning pipelines.

## 3.4 Save Metrics

Both models' training scripts incorporate a dedicated function, insert_results_into_db, that is executed at the conclusion of the training process. This function is accountable for the storage of critical performance metrics in a PostgreSQL database. By enabling the systematic recording of evaluation metrics, this function facilitates future analyses, comparisons, and audits of the models' performance, a practice widely recognized as essential in machine learning workflows (Amershi et al., 2019).

The insert_results_into_db function records numerous critical metrics, including precision, recall, F1 score, ROC AUC, and precision-recall AUC. These metrics are indispensable for assessing the efficacy of the fraud detection algorithms, as they offer valuable insights into their capacity to differentiate between legitimate and fraudulent transactions. Furthermore, the function records the values of the confusion matrix (true negatives, false positives, false negatives, and true positives), which provide a more detailed understanding of the model's classification accuracy. The CodeCarbon library monitors the model's total execution time and the environmental impact in terms of CO2 emissions, supplementing these metrics.

The function initially establishes a connection with the database in accordance with the configuration file (config.ini). It either constructs the table if it does not exist or inserts the results into the appropriate columns after confirming that the table for the respective algorithm exists (e.g., experiment_results_cb for the CatBoost or experiment_results_lgbm for LightGBM). This guarantees that the system is capable of managing numerous experiments and preserving the results for future reference. After inserting the data, the system securely terminates the database connection.

This automated logging process offers a methodical approach to monitoring performance metrics across multiple experiments, thereby simplifying the evaluation and optimization of models over time for data scientists and engineers. Such practices are increasingly crucial in scalable AI systems, as they provide transparency, reproducibility, and opportunities for continual improvement (Mitchell et al., 2019).

## 3.5 Kafka Production Script

Real-time fraud detection is made possible by this production script, which is made to implement CatBoost and LightGBM models in a live setting, aiming to process transaction data in real-time, categorizing transactions as fraudulent, non-fraudulent, or suspicious, and store the results in the relevant databases for additional analysis, the system makes use of TensorFlow for model inference and Kafka for message streaming, with an enhanced and thorough breakdown of the main elements

and features of the script presented below. First, the script loads a configuration file (config_cb.json / config_lg.json) which includes important parameters that control the system's behavior:

| | |
|---|---|
| **DATA_PATH** | Specifies the location of the data to be processed |
| **CHUNK_SIZE** | Defines the batch size for processing data in manageable chunks |
| **OFFSET** | Sets the threshold for the number of suspicious transactions after which the processing halts |
| **DIR** | Specifies the directory where the trained CatBoost / LightGBM models are stored |

***Table 5 – Kafka script configuration parameters***

These settings enable the script to be easily and adaptably adapted to a range of environments and data sources, making it suitable for a variety of use cases, controlling the flow of transaction data using a Kafka producer and a consumer, with preprocessed data being sent to the Kafka topic raw_data_cb / raw_data_lg implemented by KafkaProducer for processing in real time, and KafkaConsumer retrieving batches of messages from the same topic.

The most recent iteration of the CB/LG models are used, then loaded using TensorFlow's load_model function, enabling the system to quickly classify transactions thanks to this Kafka-based configuration, guaranteeing the seamless transfer of real-time data with the model path being dynamically chosen to guarantee that the most accurate and recent model is used, keeping the fraud detection system current with changing fraud patterns.

The script simulates live transactions by reading real-time transaction data from a CSV file and sending each row as a message to Kafka via the producer with the Pandas read_csv function reading the data in chunks, making the handling of large datasets effectively a possibility, with the messages being transmitted to the Kafka topic after each chunk is processed row by row.

The process of loading metadata from a configuration file guarantees that data types and column names are consistent across various data sources with Data Cleaning aiding in ensuring consistency. Column names are also converted to lowercase and hyphens are swapped out for underscores and at the same time using Scaling features ensures that numerical values fall within a steady range, with the scale_features function normalizing the data, allowing the models to process the data efficiently because scaling avoids problems like skewed predictions brought on by large numerical differences.

Following preprocessing and scaling, the data is fed into the trained models to make predictions with the configured CHUNK_SIZE determining the batch size for the script's processing of Kafka batch data. For every batch data is scaled and preprocessed to conform to the model's required format, following the model forecasts each transaction's probability of fraud with each transaction being given a label by the model using a threshold-based classification approach:

| | |
|---|---|
| **0** | Assigned to transactions with a prediction probability below 0.2 |
| **-1** | Assigned to transactions with probabilities between 0.2 and 0.7, indicating uncertainty |
| **1** | Assigned to transactions with probabilities above 0.7 |

*Table 6 – Transactions' labels*

The detection of clear fraud cases flagging of suspicious activities is ensured through the classification mechanism, warranting further investigation. The results are sent to separate Kafka topics and stored in corresponding PostgreSQL tables after the classification of the transactions.

| | |
|---|---|
| **fraud_cb / fraud_lg** | Stores transactions identified as fraudulent |
| **no_fraud_cb / fraud_lg** | Stores legitimate transactions |
| **suspicious_cb / suspicious_lg** | Stores transactions marked as suspicious, requiring further review |

*Table 7 – Transactions' tables*

Effective analysis and monitoring are made possible by this data segregation, with each transaction category being handled correctly with targeted reactions to fraud and suspicious activity being possible by arranging the classified data in separate tables.

This tracking of dubious transactions is one of the script's most crucial functions with the save_offset function being used to update the offset value each time a batch is processed, guaranteeing that the script can continue processing from the most recent transaction without having to reprocess data that has already been handled in the event that it is interrupted.

The quantity of highly dubious transactions is continuously tracked by initiating a stopping condition when the quantity of suspicious transactions surpasses the pre-established OFFSET, enabling the system to restrict the volume of data processed prior to subsequent operations, an action similar to retraining the model.

To guarantee seamless execution and offer insights into its operations, the script incorporates thorough logging throughout the processing pipeline, with important occurrences, like preprocessing, batch processing, and scaling completion. All the above are being recorded to provide operators with insight into the script's development, as in the event of an error, the script records it and offers a thorough stack trace for troubleshooting, guaranteeing that any problems are found and fixed right away, while at the same time, the script is built to withstand errors, with the fraud detection process being minimized by recovering and starting from the most recently saved offset if an error or even a failure happens.

## 3.6 Labeling Suspicious Data

Labeling begins with loading the necessary configurations from a JSON file, config_rf.json, which provides dynamic and reusable settings for the pipeline. It utilizes the RandomForestClassifier, a reliable and interpretable machine learning algorithm, to classify suspicious cases. Training is conducted using stratified cross-validation to ensure balanced evaluation of fraud and non-fraud

classes (Breiman, 2001). To further improve detection performance, class weights address the imbalance in the dataset effectively.

Data preparation involves merging two main datasets, fraud_df and no_fraud_df, into a single dataset while keeping their labels in separate variables to form the target variable. This consolidated dataset serves as the basis for training and validation of the Random Forest model. Meanwhile, suspicious_df undergoes preprocessing to match the processed training dataset, ensuring consistent features and compatibility with the model's input.

Evaluation is performed using Stratified K-Folds Cross-Validation with five folds. Stratification ensures consistent class distribution across splits, which is vital for fraud detection due to the intrinsic class imbalance (Kohavi, 1995). Performance metrics, including precision, recall, F1-score, and AUC, are calculated for each fold, and the results are reviewed. Average metrics, along with their standard deviations, are computed to assess the reliability and robustness of the model.

Once evaluated, the model predicts labels for the suspicious dataset. These predictions are integrated into the dataset under a new column, isfraud, with fraudulent cases extracted for further analysis. Newly labeled fraudulent cases are appended to the training dataset, selectively enriching the model's learning. This process is critical for continuous adaptation, allowing the model to identify emerging fraud patterns effectively.

Efficient data retrieval is managed through the fetch_data function, which interfaces with a PostgreSQL database. This function is designed to handle large datasets by implementing an offset mechanism for tables exceeding a predefined threshold. The mechanism divides the dataset into manageable chunks using SQL's LIMIT and OFFSET clauses, optimizing data retrieval (Rowe & Stonebraker, 1987). This approach minimizes memory overload, reduces server load, and ensures rapid processing.

Fraudulent data is appended to the training dataset in the final step, allowing the model to learn from new fraud cases during subsequent training iterations. By focusing on new fraudulent patterns, the model's predictive accuracy and robustness against evolving schemes are improved. This iterative update process is integral to maintaining the pipeline's effectiveness over time, ensuring reliable detection of sophisticated fraud patterns.

# 4 Challenges and Limitations

## 4.1 Memory Management

One of the biggest limitations faced during the development process was memory management. A system with 16 GB RAM was used, which for most of the operations is more than enough. In some operations, though, such as rebalancing and handling large datasets, complications related to memory appeared. This especially happened when the system required more memory than what was available; this caused performance bottlenecks and increased the possibility of the system crashing.

These challenges were overcome by configuring the system to utilize disk space as additional memory by increasing swap memory to 30 GB. With this, the system can easily handle higher workloads without compromising on performance. In addition, managing the memory effectively also involved the reconfiguration of the models and their respective hyperparameters so that the experiments smoothly ran on the development machine.

For example, the setting of n_jobs = -1 in the hyperparameters of the model and in Optuna trials caused system crashes in the LightGBM pipeline due to excessive consumption of the available resources. This has been tuned to using a limited number of threads as the n_jobs parameter to allow efficient computation without system crashes. Many such adjustments have been done with other parts of the workflow for better utilization of resources without the experiments being cut off.

These changes highlight the importance of aligning model configurations with system constraints, especially for large-scale machine learning pipelines. By carefully managing memory and adapting model settings, the development process maintained stability while achieving the desired experimental outcomes.

| | **Total** | **Used** | **Free** | **Shared** | **Buff/Cache** | **Available** |
|---|---|---|---|---|---|---|
| **Memory** | 30 Gi | 7.8 Gi | 8.0 Gi | 118 Mi | 4.6 Gi | 22 Gi |
| **Swap** | 29 Gi | 0 B | 29 Gi | | | |

***Table 8 - Memory Management***

## 4.2 LGBM GPU Support

Another critical issue encountered was related to using GPU acceleration with LightGBM. While GPU support is intended to speed up training, errors occurred during the process, such as:

*lightgbm.basic.LightGBMError: Check failed: (best_split_info.left_count) > (0)*

This error usually occurs due to problems in the GPU-based split calculation, which often relates to some specific setup of the data or some rare cases present in the dataset. For example, a conversation on GitHub underlines similar problems that users have when training LightGBM models on GPUs; this shows that some data features might cause this error.

# 5 Results

## 5.1 Performance Evaluation Main Training

### 5.1.1 LGBM

#### 5.1.1.1 Main Training

| | |
|---|---|
| **CU_DT** | 20241216 |
| **Precision** | 0.9206 |
| **Recall** | 0.7353 |
| **F1_Score** | 0.8175 |
| **ROC_AUC** | 0.9731 |
| **PR_AUC** | 0.8624 |
| **Execution_Time** | 11 hours, 10 minutes |
| **Emissions** | 0.2229 |

*Table 9 - Main Training Metrics - LightGBM*

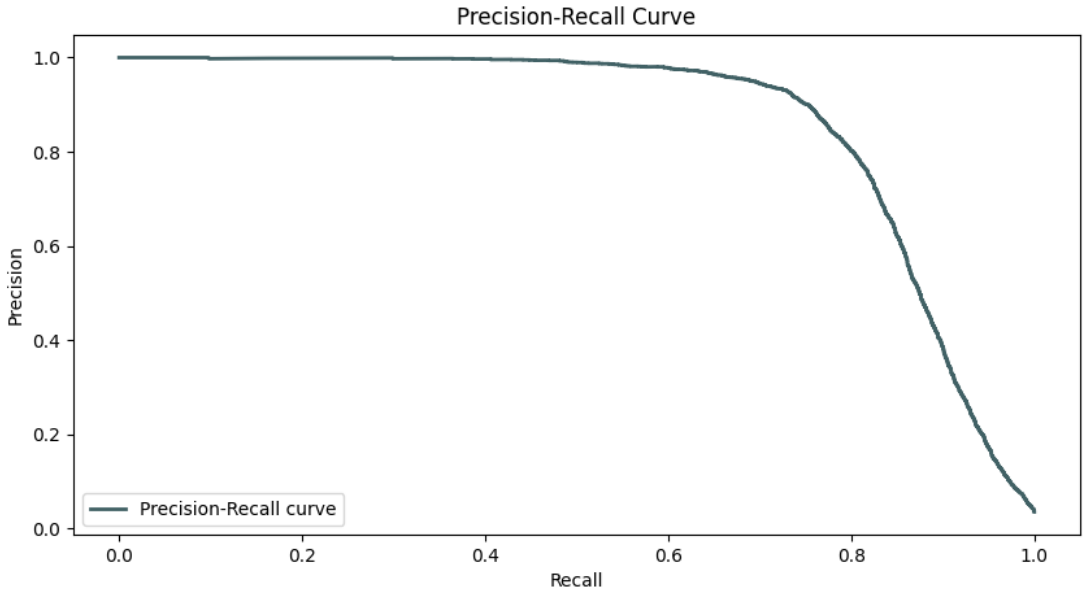



*Figure 24 - Precision-Recall Curve Main training - LightGBM*

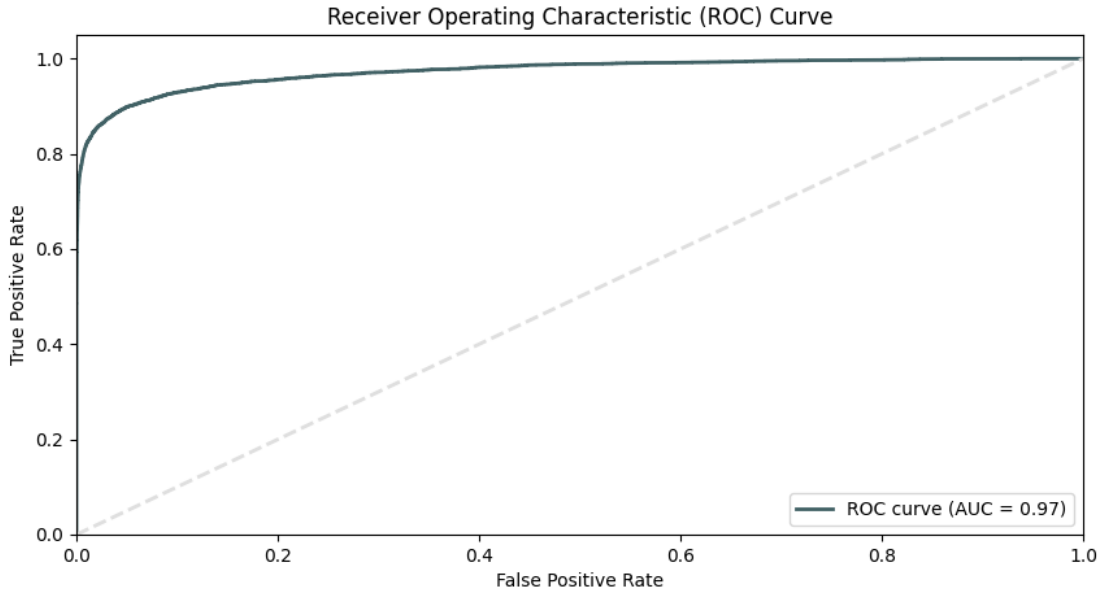



*Figure 25 - Receiver Operating Characteristic (ROC) Curve Main training -LightGBM*

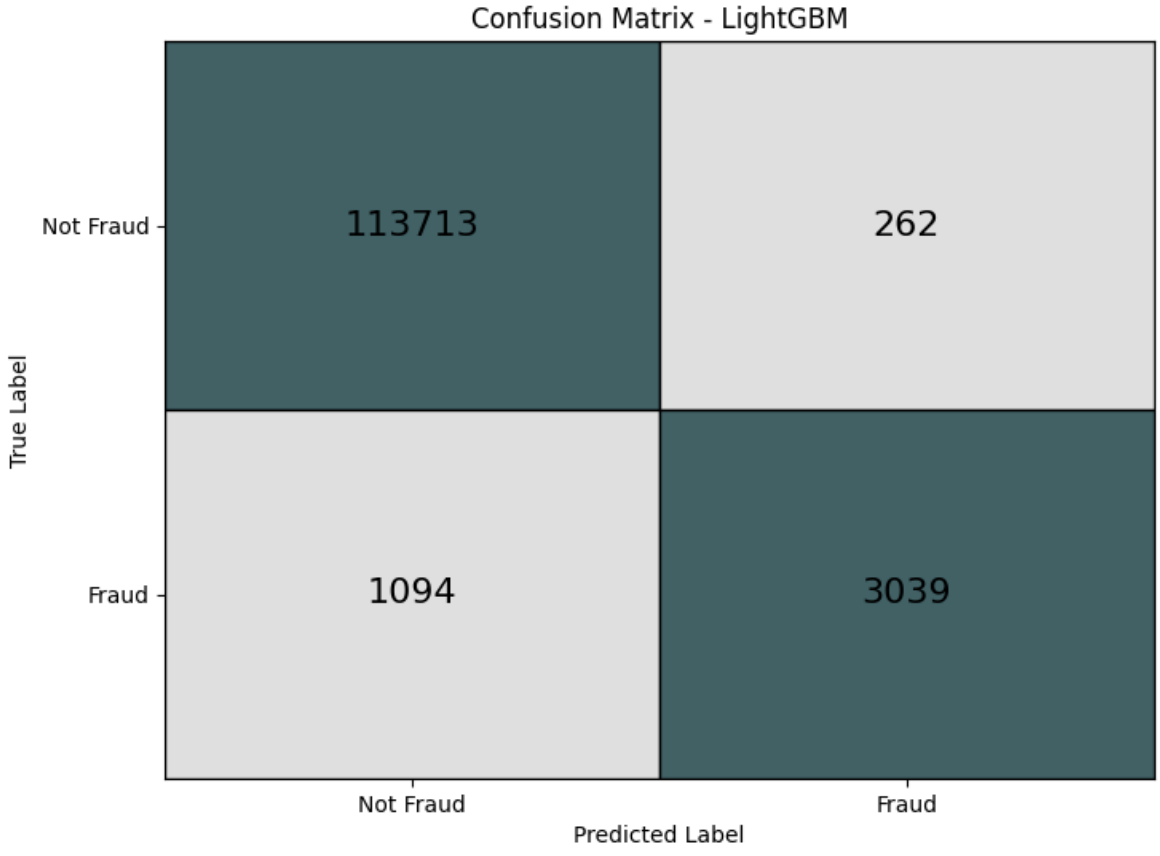



*Figure 26 - Confusion Matrix Main Training– LightGBM*

| Parameter | Value |
|---|---|
| **max_depth** | 15 |
| **num_leaves** | 240 |
| **min_data_in_leaf** | 25 |
| **max_bin** | 187 |

*Table 10 - Tree Structure Parameters (LightGBM Main Training)*

| Parameter | Value |
|---|---|
| **learning_rate** | 0.03572380838577693 |
| **feature_fraction** | 0.44315949154615925 |
| **bagging_fraction** | 0.9244814427726744 |
| **bagging_freq** | 8 |

*Table 11 - Learning Parameters (LightGBM Main Training)*

| Parameter | Value |
|---|---|
| **min_child_weight** | 0.00148097144714855 |
| **reg_alpha** | 0.02464267739550402 |
| **reg_lambda** | 0.01563385157285081 |

*Table 12 - Regularization Parameters (LightGBM Main Training)*

| Parameter | Value |
|---|---|
| **verbosity** | -1 |
| **n_jobs** | 1 |
| **seed** | 42 |

*Table 13 - System settings (LightGBM Main Training)*

#### 5.1.1.2 Labelling Main Training

| Label | Value |
|---|---|
| **no_fraud_lg** | 160209 |
| **fraud_lg** | 2785 |
| **suspicious_lg** | 5006 |

*Table 14 - Production predictions (LightGBM Main Training)*

| Metric | Value |
|---|---|
| **Precision** | 0.963 |
| **Recall** | 0.890 |
| **F1-Score** | 0.925 |

*Table 15 - Random Forest Labeling Metrics (LightGBM Main Training)*

| Label | Value |
|---|---|
| **0** | 5006 |
| **1** | 0 |

*Table 16 - Random Forest Predictions (LightGBM Main Training)*

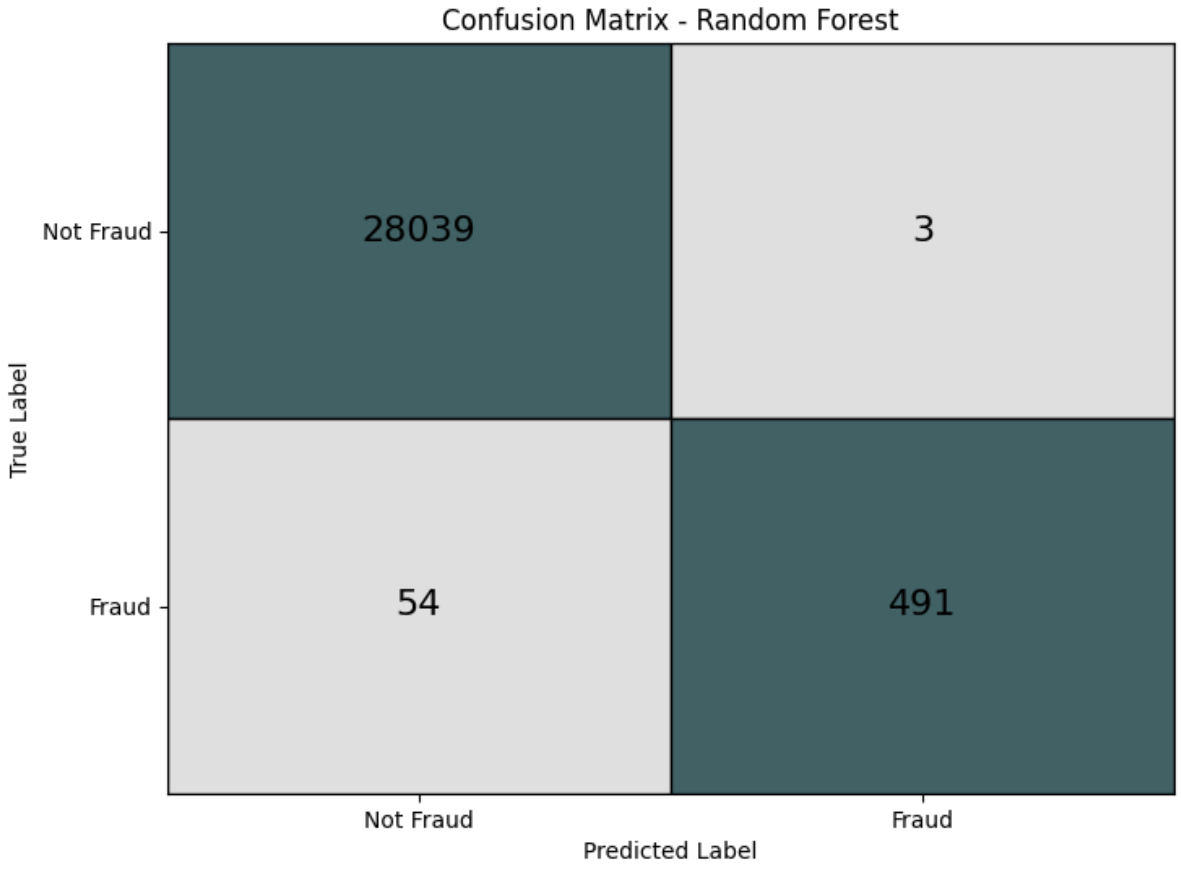



*Figure 27 - Confusion Matrix - Random Forest (LightGBM Main Training)*

### 5.1.2 CatBoost

#### 5.1.2.1 Main Training

| | |
|---|---|
| **CU_DT** | 20241229 |
| **Precision** | 0.8621 |
| **Recall** | 0.8047 |
| **F1_Score** | 0.8324 |
| **ROC_AUC** | 0.9733 |
| **PR_AUC** | 0.8764 |
| **Execution_Time** | 7 hours, 24 minutes |
| **Emissions** | 0.2887 |

*Table 17 - Main Training Metrics - CatBoost*

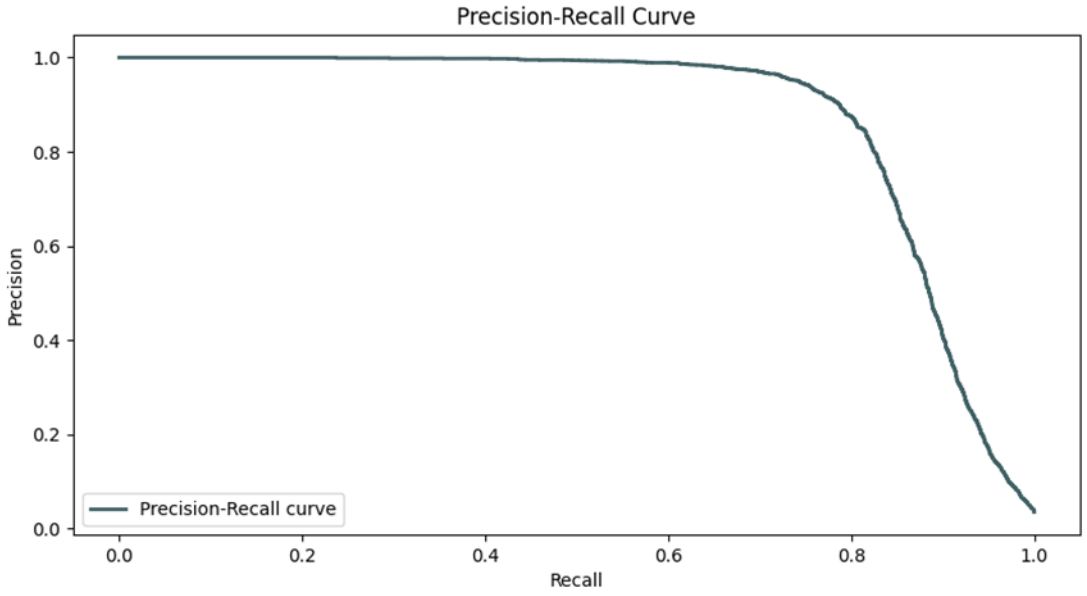



*Figure 28 - Precision-Recall Curve Main training - CatBoost*

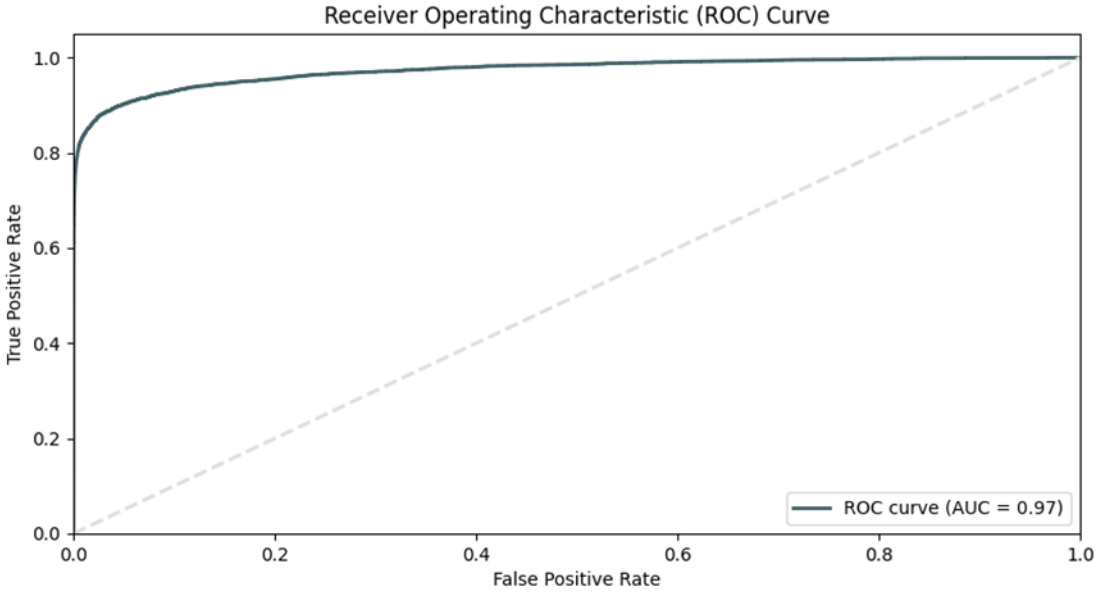



*Figure 29 - Precision-Recall Curve Main training - CatBoost*

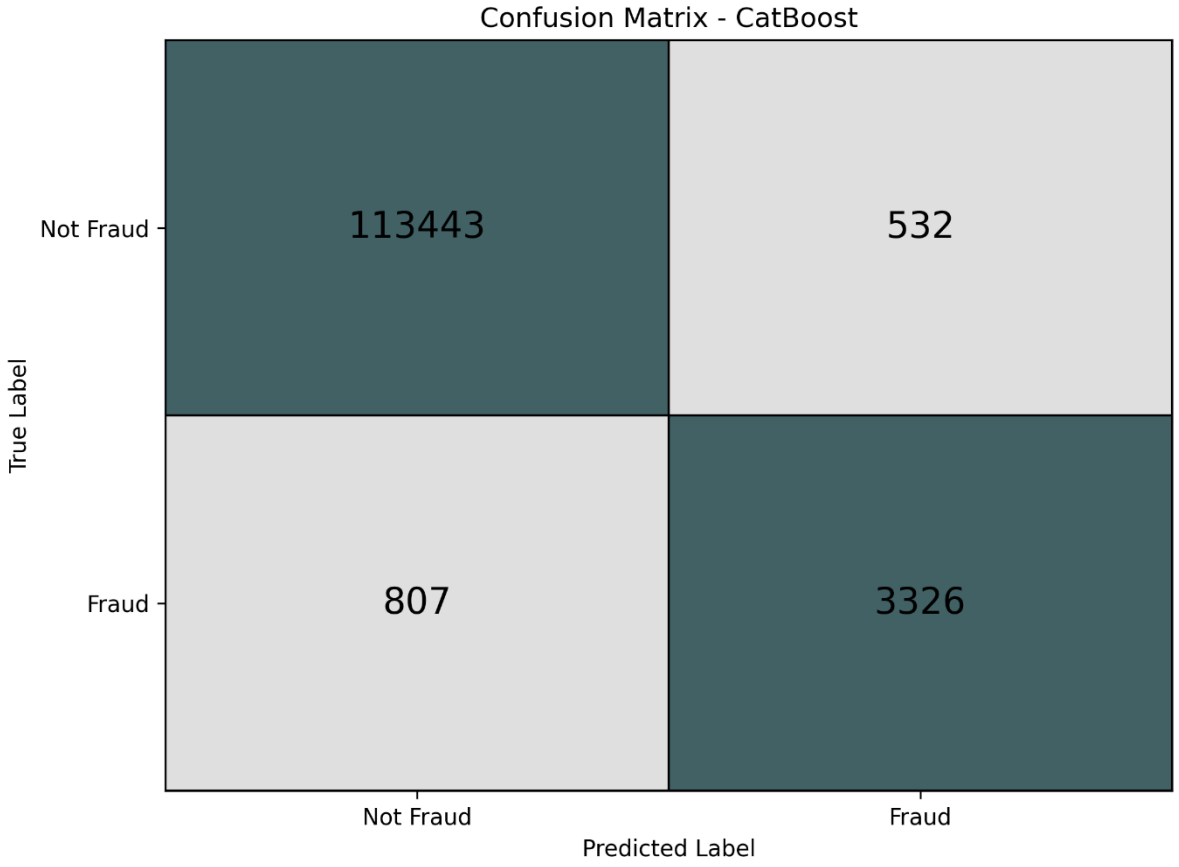



*Figure 30 - Confusion Matrix Main Training– CatBoost*

| Parameter | Value |
|---|---|
| **iterations** | 932 |
| **depth** | 8 |

*Table 18 - Tree Structure Parameters (CatBoost Main Training)*

| Parameter | Value |
|---|---|
| **learning_rate** | 0.19253249019700144 |
| **bagging_temperature** | 0.03119132475308568 |

*Table 19 - Learning Parameters (CatBoost Main Training)*

| Parameter | Value |
|---|---|
| **l2_leaf_reg** | 0.1096079848122838 |
| **random_strength** | 4.0090816333290635e-05 |

*Table 20 - Regularization Parameters (CatBoost Main Training)*

| Parameter | Value |
|---|---|
| **verbose** | 0 |
| **task_type** | GPU |
| **random_seed** | 42 |

*Table 21 - System settings (CatBoost Main Training)*

### 5.1.2.2 Labelling Main Training

| Label | Value |
|---|---|
| **no_fraud_lg** | 3520 |
| **fraud_lg** | 211467 |
| **suspicious_lg** | 5013 |

*Table 22 - Production predictions (CatBoost Main Training)*

| Metric | Value |
|---|---|
| **Precision** | 0.931 |
| **Recall** | 0.695 |
| **F1-Score** | 0.796 |

*Table 23 - Random Forest Labeling Metrics (CatBoost Main Training)*

| Label | Value |
|---|---|
| **0** | 5013 |
| **1** | 0 |

*Table 24 - Random Forest Predictions (CatBoost Main Training)*

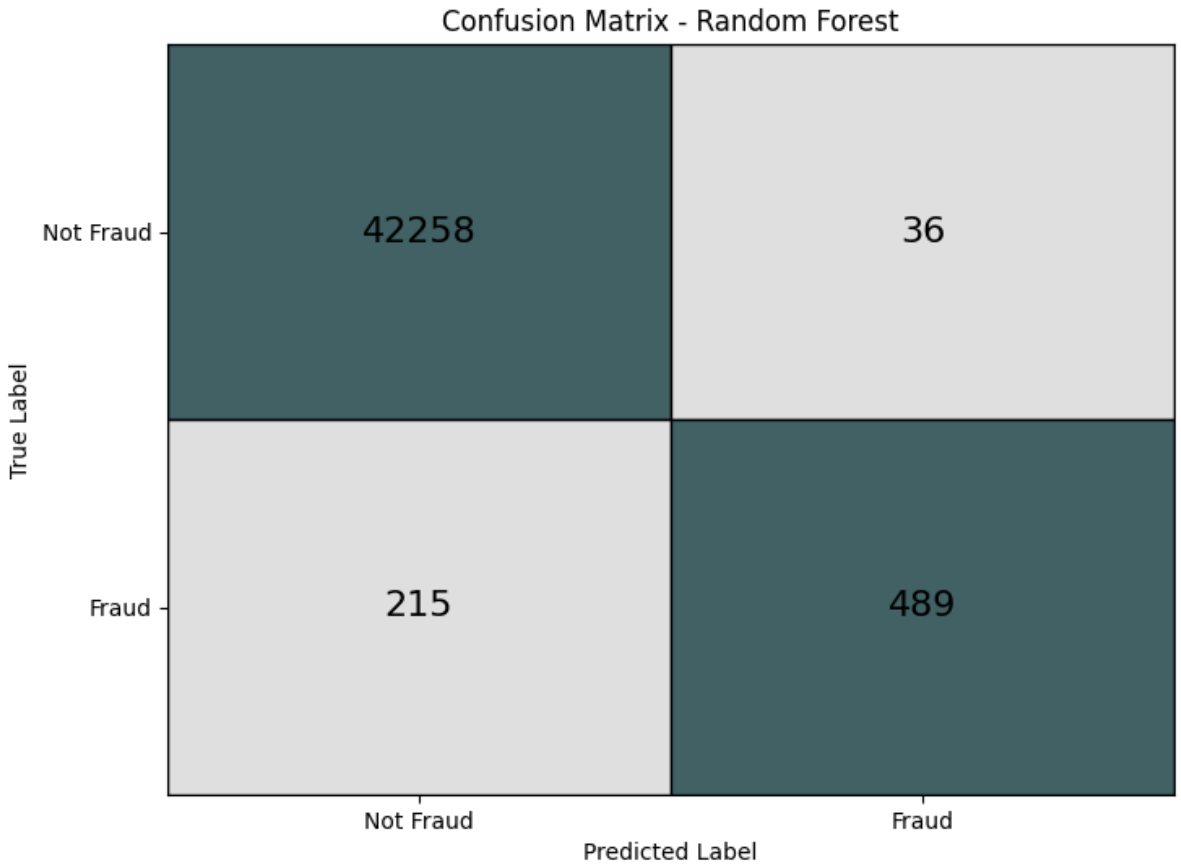



*Figure 31 - Confusion Matrix - Random Forest (CatBoost Main Training)*

## 5.2 Retraining impact on model performance

### 5.2.1 LGBM

#### 5.2.1.1 First Run

##### 5.2.1.1.1 Retraining Phase

| | |
|---|---|
| **CU_DT** | 20241217 |
| **Precision** | 0.9461 |
| **Recall** | 0.7896 |
| **F1_Score** | 0.8608 |
| **ROC_AUC** | 0.9770 |
| **PR_AUC** | 0.9019 |
| **Execution_Time** | 11 hours, 28 minutes |
| **Emissions** | 0.2011 |

*Table 25 - Retraining Metrics (LightGBM First Run)*

| Parameter | Value |
|---|---|
| **max_depth** | 18 |
| **num_leaves** | 209 |
| **min_data_in_leaf** | 67 |
| **max_bin** | 171 |

*Table 26 - Tree Structure Parameters Retraining (LightGBM First Run)*

| Parameter | Value |
|---|---|
| **learning_rate** | 0.0523112513101304 |
| **feature_fraction** | 0.5602917667839747 |
| **bagging_fraction** | 0.991175811839782 |
| **bagging_freq** | 6 |

*Table 27 - Learning Parameters Retraining (LightGBM First Run)*

| Parameter | Value |
|---|---|
| **min_child_weight** | 0.01191905264957574 |
| **reg_alpha** | 0.03124271599931228 |
| **reg_lambda** | 0.12081273980068283 |

*Table 28 - Regularization Parameters Retraining (LightGBM First Run)*

| Parameter | Value |
|---|---|
| **verbosity** | -1 |
| **n_jobs** | 1 |
| **seed** | 42 |

*Table 29 - System settings Retraining (LightGBM First Run)*

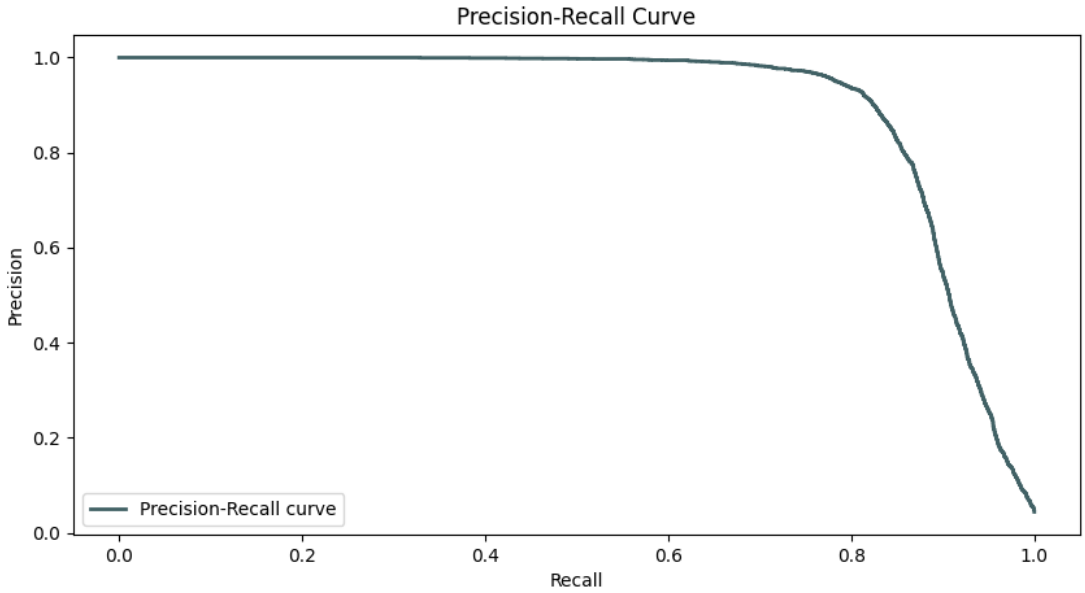



*Figure 32 - Precision-Recall Curve Retraining (LightGBM First Run)*

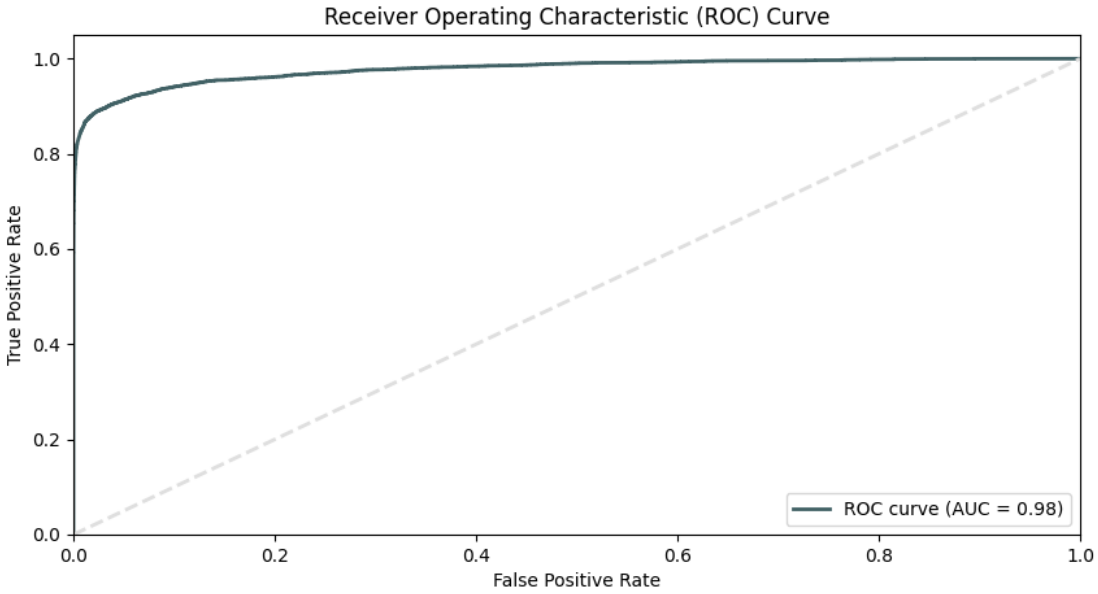



*Figure 33 - Receiver Operating Characteristic (ROC) Curve Retraining (LightGBM First Run)*

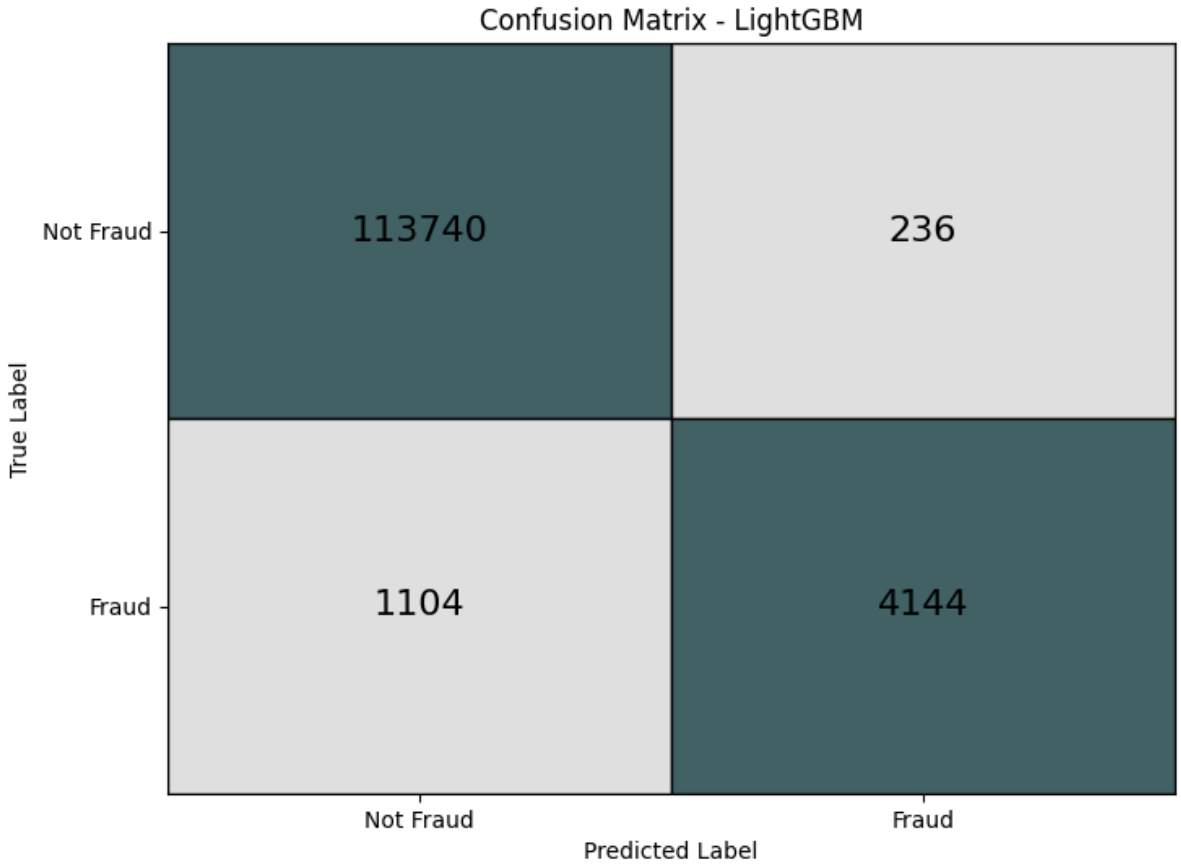



*Figure 34 - Confusion Matrix - Retraining (LightGBM First Run)*

#### 5.2.1.1.2 Labelling Phase

| Label | Value |
|---|---|
| **no_fraud_lg** | 138112 |
| **fraud_lg** | 2842 |
| **suspicious_lg** | 5045 |

*Table 30 - Production predictions (LightGBM First Run)*

| Metric | Value |
|---|---|
| **Precision** | 0.961 |
| **Recall** | 0.782 |
| **F1-Score** | 0.862 |

*Table 31 - Random Forest Labeling Metrics (LightGBM First Run)*

| Label | Value |
|---|---|
| **0** | 4999 |
| **1** | 46 |

*Table 32 - Random Forest Predictions (LightGBM First Run)*

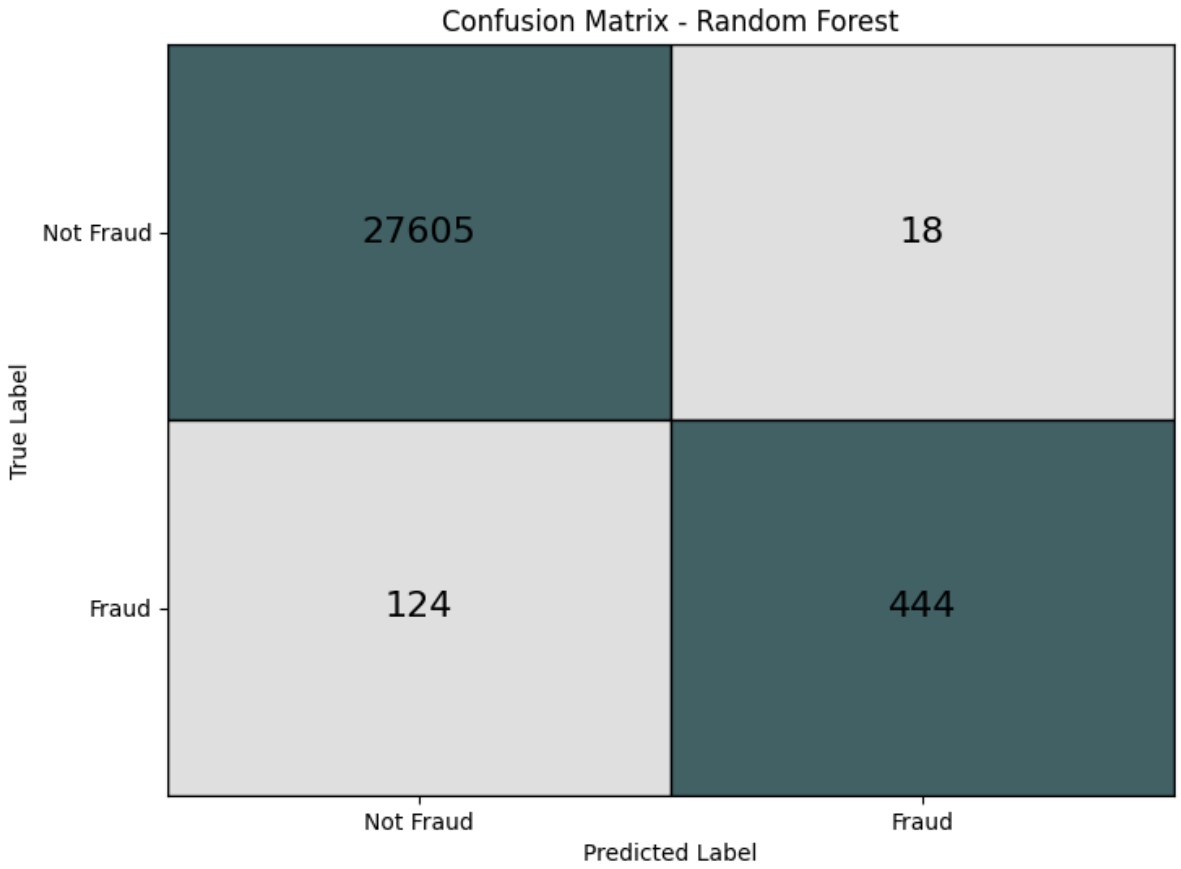



*Figure 35 - Confusion Matrix - Random Forest (LightGBM First Run)*

#### 5.2.1.2 Second Run

##### 5.2.1.2.1 Retraining Phase

| | |
|---|---|
| **CU_DT** | 20241218 |
| **Precision** | 0.9520 |
| **Recall** | 0.8110 |
| **F1_Score** | 0.8758 |
| **ROC_AUC** | 0.9797 |
| **PR_AUC** | 0.9164 |
| **Execution_Time** | 11 hours, 59 minutes |
| **Emissions** | 0.2079 |

*Table 33 - Retraining Metrics (LightGBM Second Run)*

| Parameter | Value |
|---|---|
| **max_depth** | 13 |
| **num_leaves** | 228 |
| **min_data_in_leaf** | 86 |
| **max_bin** | 232 |

*Table 34 - Tree Structure Parameters Retraining (LightGBM Second Run)*

| Parameter | Value |
|---|---|
| **learning_rate** | 0.0549380670305092 |
| **feature_fraction** | 0.6687257546614209 |
| **bagging_fraction** | 0.946378206358222 |
| **bagging_freq** | 4 |

*Table 35 - Learning Parameters Retraining (LightGBM Second Run)*

| Parameter | Value |
|---|---|
| **min_child_weight** | 0.00146217790161635 |
| **reg_alpha** | 0.03958323074861385 |
| **reg_lambda** | 0.01516435533248889 |

*Table 36 - Regularization Parameters Retraining (LightGBM Second Run)*

| Parameter | Value |
|---|---|
| **verbosity** | -1 |
| **n_jobs** | 1 |
| **seed** | 42 |

*Table 37 - System settings Retraining (LightGBM Second Run)*

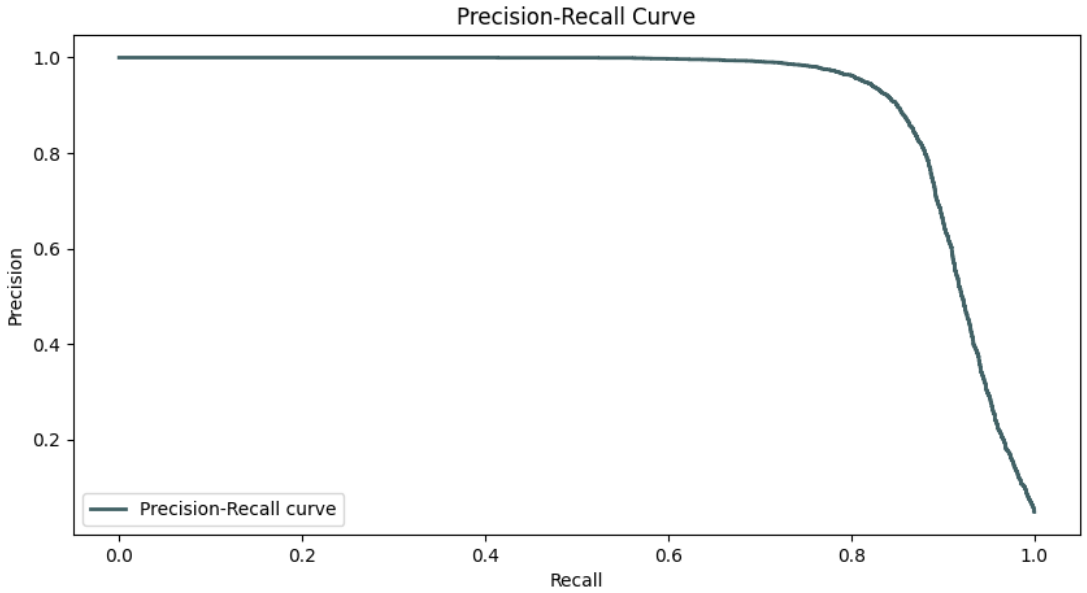



*Figure 36 - Precision-Recall Curve Retraining (LightGBM Second Run)*

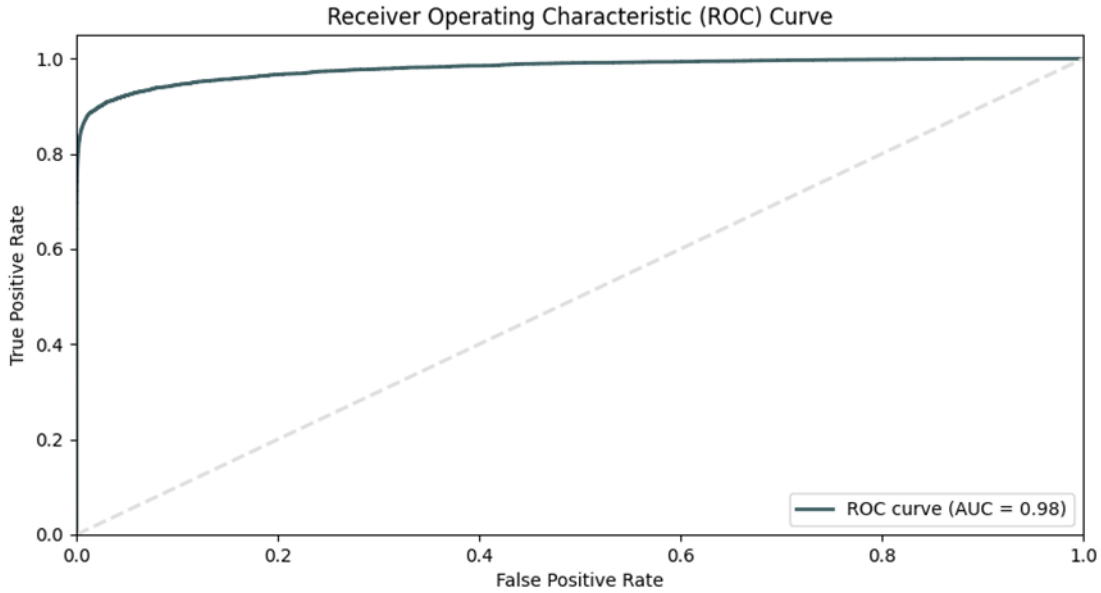



*Figure 37 - Receiver Operating Characteristic (ROC) Curve Retraining (LightGBM Second Run)*

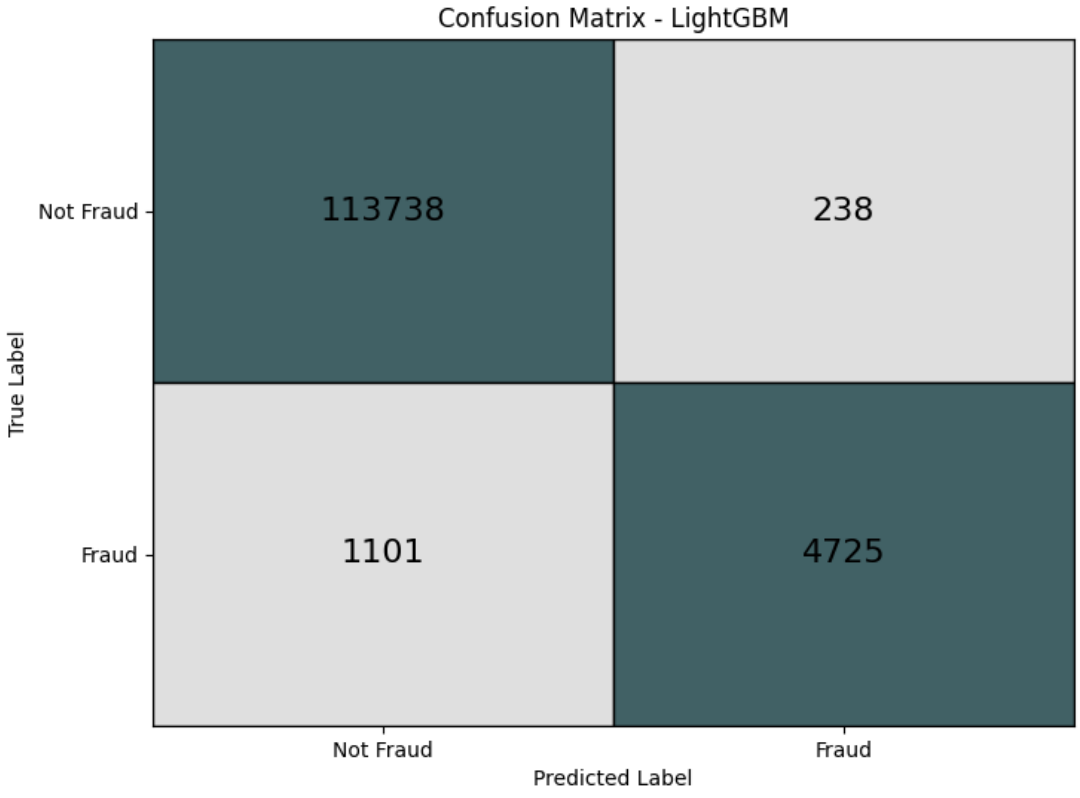



*Figure 38 - Confusion Matrix - Retraining (LightGBM Second Run)*

#### 5.2.1.2.2 Labelling Phase

| Label | Value |
|---|---|
| **no_fraud_lg** | 181155 |
| **fraud_lg** | 2564 |
| **suspicious_lg** | 4281 |

*Table 38 - Production predictions (LightGBM Second Run)*

| Metric | Value |
|---|---|
| **Precision** | 0.981 |
| **Recall** | 0.706 |
| **F1-Score** | 0.821 |

*Table 39 - Random Forest Labeling Metrics (LightGBM Second Run)*

| Label | Value |
|---|---|
| **0** | 4280 |
| **1** | 1 |

*Table 40 - Random Forest Predictions (LightGBM Second Run)*

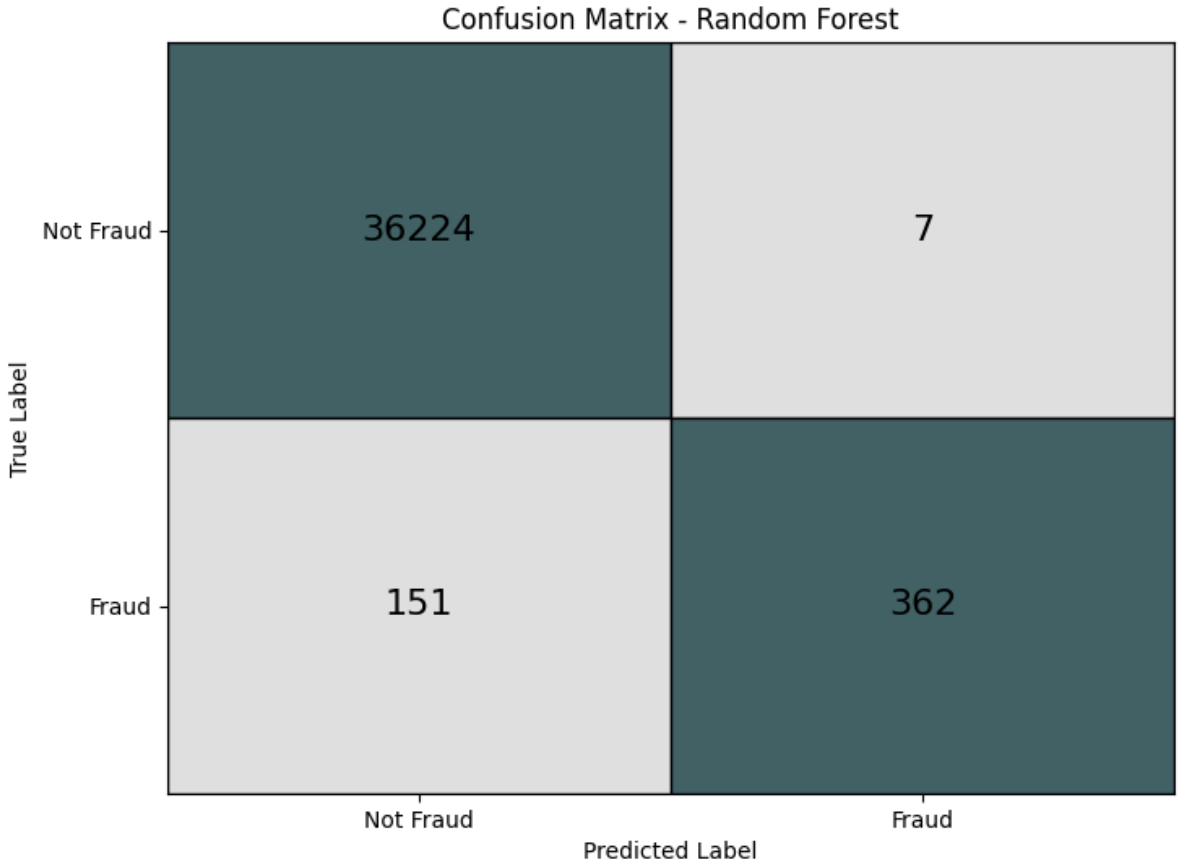



*Figure 39 - Confusion Matrix - Random Forest (LightGBM Second Run)*

### 5.2.2 CatBoost

#### 5.2.2.1 First Run

##### 5.2.2.1.1 Retraining Phase

| | |
|---|---|
| **CU_DT** | 20241230 |
| **Precision** | 0.86791 |
| **Recall** | 0.8327 |
| **F1_Score** | 0.8499 |
| **ROC_AUC** | 0.9772 |
| **PR_AUC** | 0.8998 |
| **Execution_Time** | 8 hours, 9 minutes |
| **Emissions** | 0.3203 |

*Table 41 - Retraining Metrics (CatBoost First Run)*

| Parameter | Value |
|---|---|
| **iterations** | 788 |
| **depth** | 8 |

*Table 42 - Tree Structure Parameters Retraining (CatBoost First Run)*

| Parameter | Value |
|---|---|
| **learning_rate** | 0.19861699134137997 |
| **bagging_temperature** | 0.15063115230908067 |

*Table 43 - Learning Parameters Retraining (CatBoost First Run)*

| Parameter | Value |
|---|---|
| **l2_leaf_reg** | 0.120891453425621 |
| **random_strength** | 0.002759259413445753 |

*Table 44 - Regularization Parameters Retraining (CatBoost First Run)*

| Parameter | Value |
|---|---|
| **verbose** | 0 |
| **task_type** | GPU |
| **random_seed** | 42 |

*Table 45 - System settings Retraining (CatBoost First Run)*

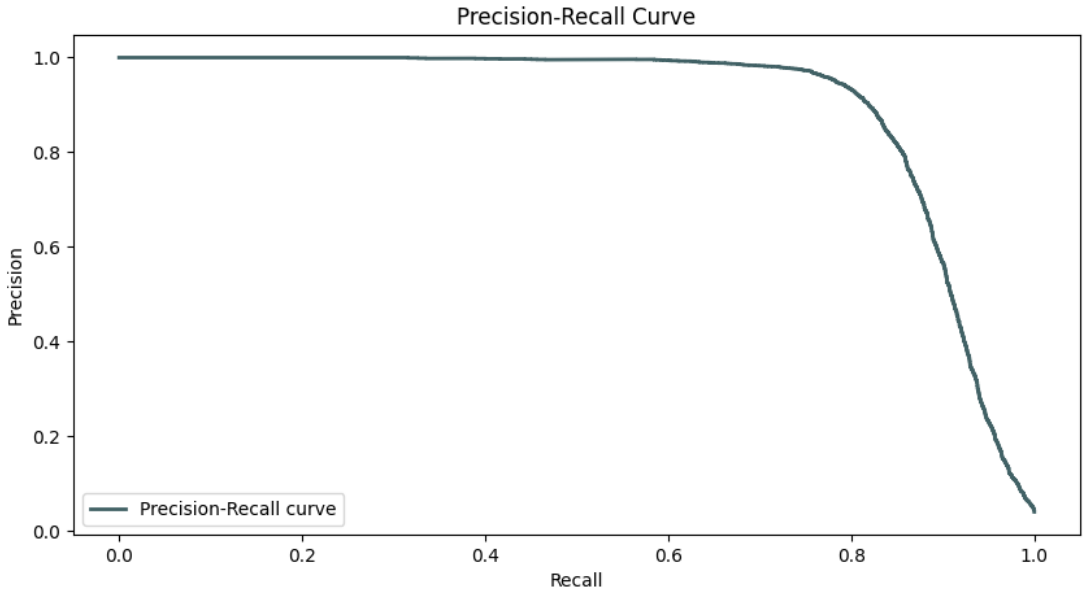



*Figure 40 - Precision-Recall Curve Retraining (CatBoost First Run)*

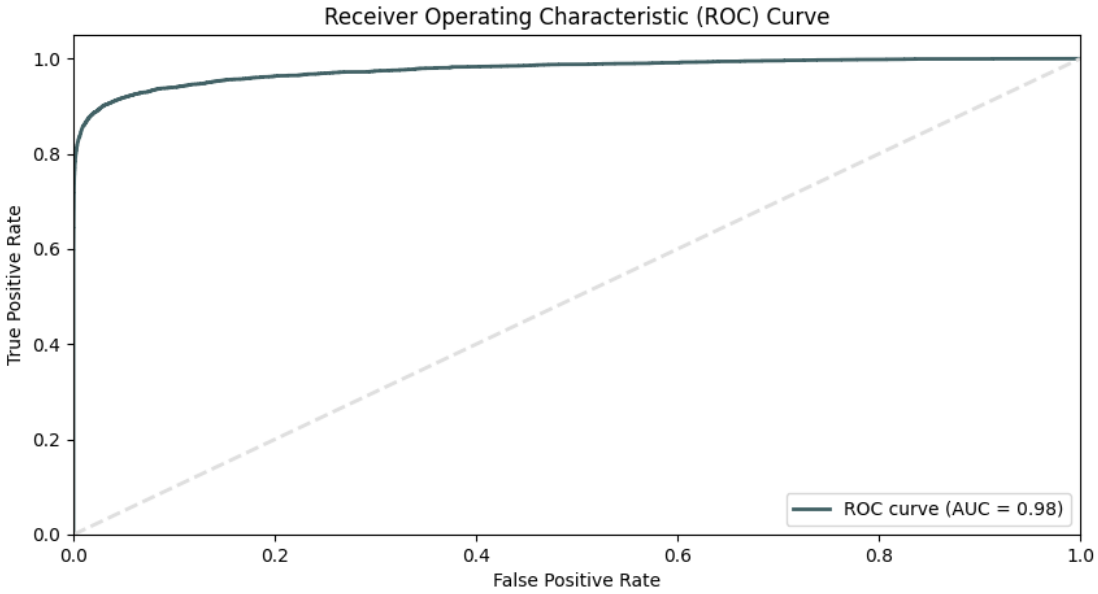



*Figure 41 - Receiver Operating Characteristic (ROC) Curve Retraining (CatBoost First Run)*

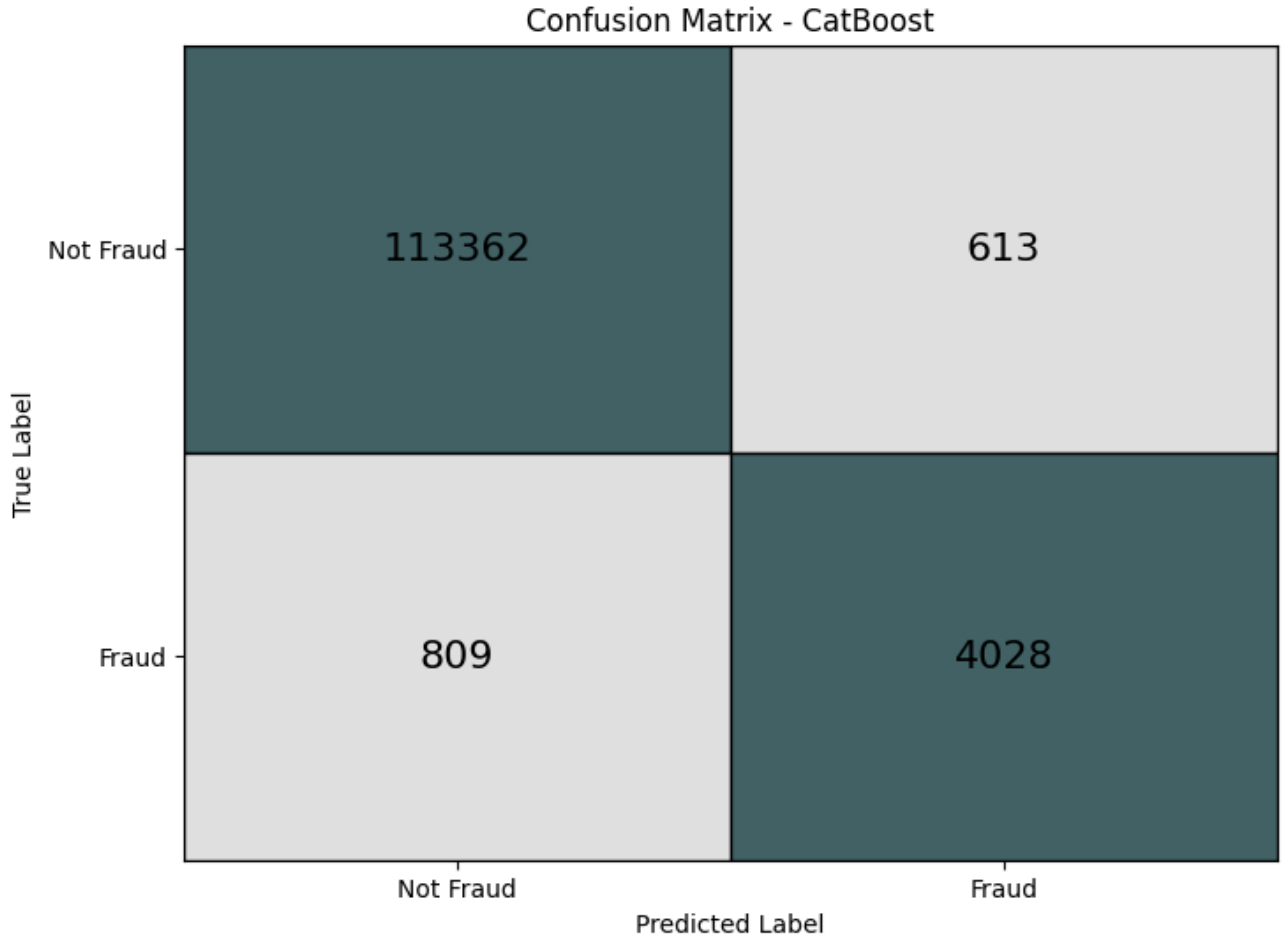



*Figure 42 - Confusion Matrix - Retraining (CatBoost First Run)*

#### 5.2.2.1.2 Labelling Phase

| Label | Value |
|---|---|
| **no_fraud_lg** | 84701 |
| **fraud_lg** | 2269 |
| **suspicious_lg** | 5030 |

*Table 46 - Production predictions (CatBoost First Run)*

| Metric | Value |
|---|---|
| **Precision** | 0.974 |
| **Recall** | 0.496 |
| **F1-Score** | 0.657 |

*Table 47 - Random Forest Labeling Metrics (CatBoost First Run)*

| Label | Value |
|---|---|
| **0** | 5003 |
| **1** | 27 |

*Table 48 - Random Forest Predictions (CatBoost First Run)*

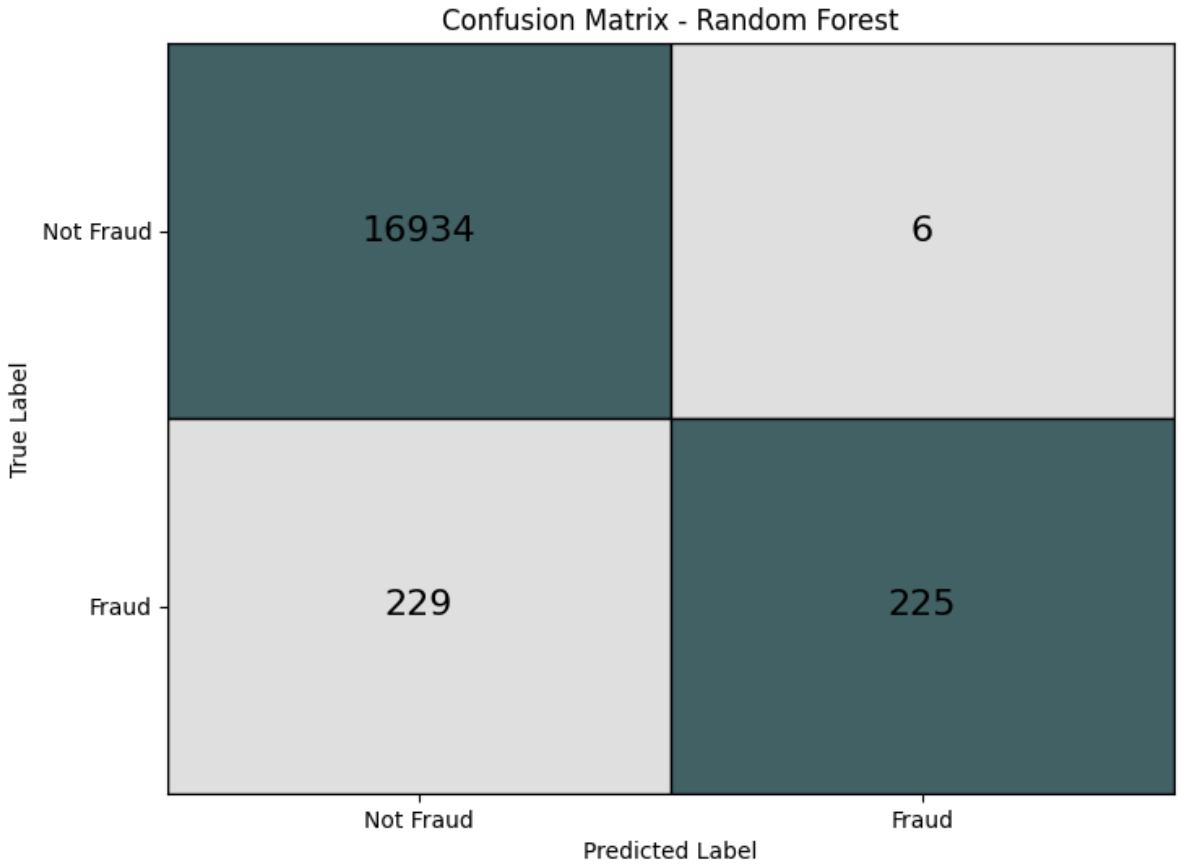



*Figure 43 - Confusion Matrix - Random Forest (CatBoost First Run)*

### 5.2.2.2 Second Run

#### 5.2.2.2.1 Retraining Phase

| | |
|---|---|
| **CU_DT** | 20241231 |
| **Precision** | 0.8828 |
| **Recall** | 0.8440 |
| **F1_Score** | 0.8630 |
| **ROC_AUC** | 0.9789 |
| **PR_AUC** | 0.9091 |
| **Execution_Time** | 8 hours, 57 minutes |
| **Emissions** | 0.3531 |

*Table 49 - Retraining Metrics (CatBoost Second Run)*

| Parameter | Value |
|---|---|
| **iterations** | 980 |
| **depth** | 8 |

*Table 50 - Tree Structure Parameters Retraining (CatBoost Second Run)*

| Parameter | Value |
|---|---|
| **learning_rate** | 0.19940029466202946 |
| **bagging_temperature** | 0.5547321050988824 |

*Table 51 - Learning Parameters Retraining (CatBoost Second Run)*

| Parameter | Value |
|---|---|
| **l2_leaf_reg** | 0.1274298734640127 |
| **random_strength** | 0.05009820908437887 |

*Table 52 - Regularization Parameters Retraining (CatBoost Second Run)*

| Parameter | Value |
|---|---|
| **verbose** | 0 |
| **task_type** | GPU |
| **random_seed** | 42 |

*Table 53 - System settings Retraining (CatBoost Second Run)*

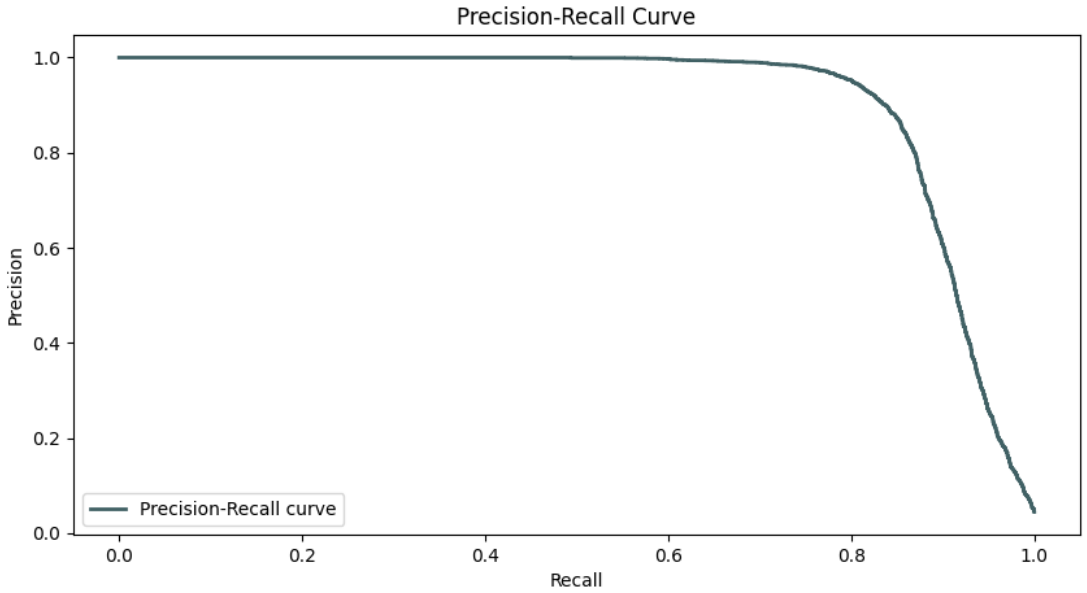



*Figure 44 - Precision-Recall Curve Retraining (CatBoost Second Run)*

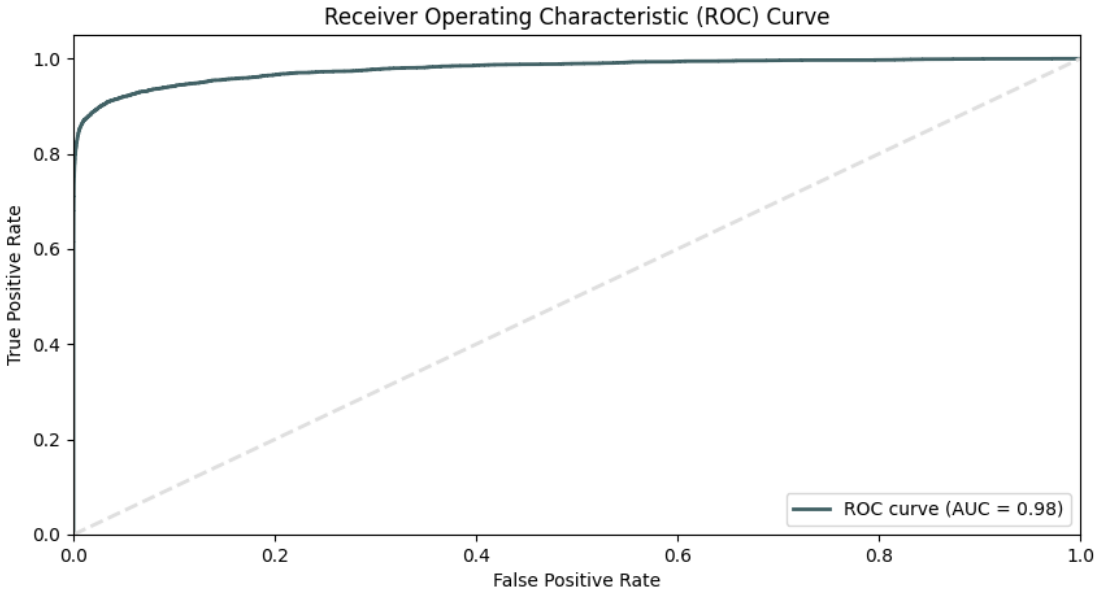



*Figure 45 - Receiver Operating Characteristic (ROC) Curve Retraining (CatBoost Second Run)*

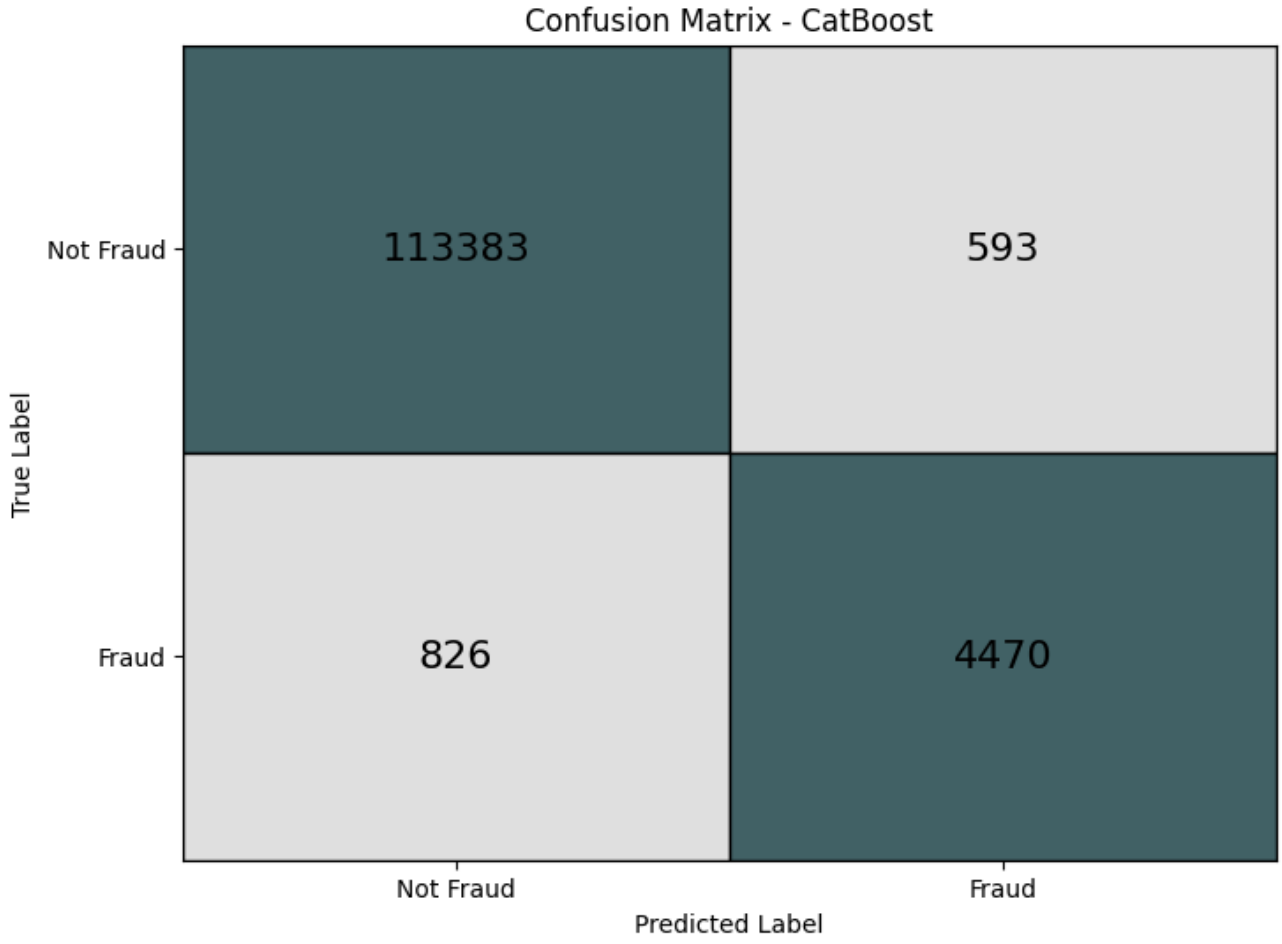



*Figure 46 - Confusion Matrix - Retraining (CatBoost Second Run)*

#### 5.2.2.2.2 Labelling Phase

| Label | Value |
|---|---|
| **no_fraud_lg** | 35018 |
| **fraud_lg** | 1975 |
| **suspicious_lg** | 5007 |

*Table 54 - Production predictions (CatBoost Second Run)*

| Metric | Value |
|---|---|
| **Precision** | 0.919 |
| **Recall** | 0.289 |
| **F1-Score** | 0.439 |

*Table 55 - Random Forest Labeling Metrics (CatBoost Second Run)*

| Label | Value |
|---|---|
| **0** | 4998 |
| **1** | 9 |

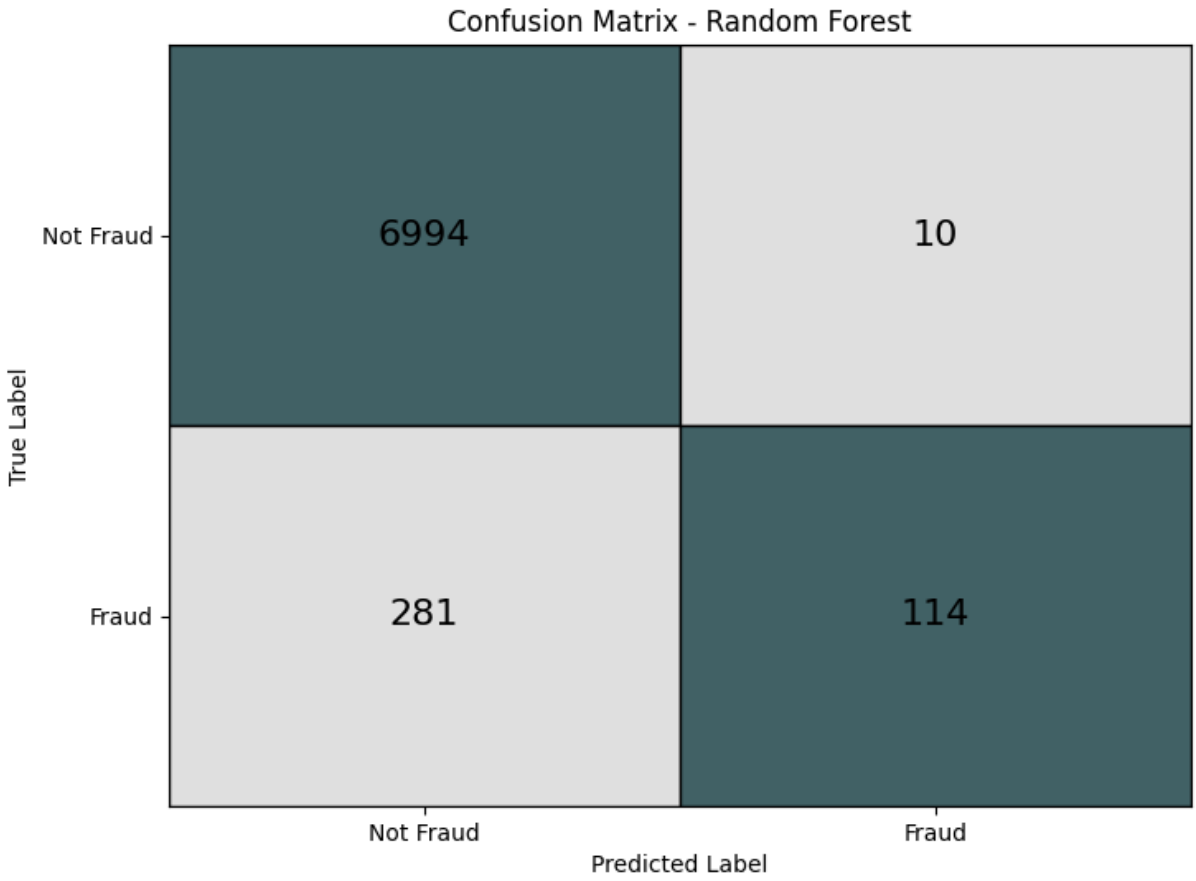



*Table 56 - Random Forest Predictions (CatBoost Second Run)*

*Figure 47 - Confusion Matrix - Random Forest (CatBoost Second Run)*

## 5.3 Co2 emissions and Electricity usage

### 5.3.1 LGBM

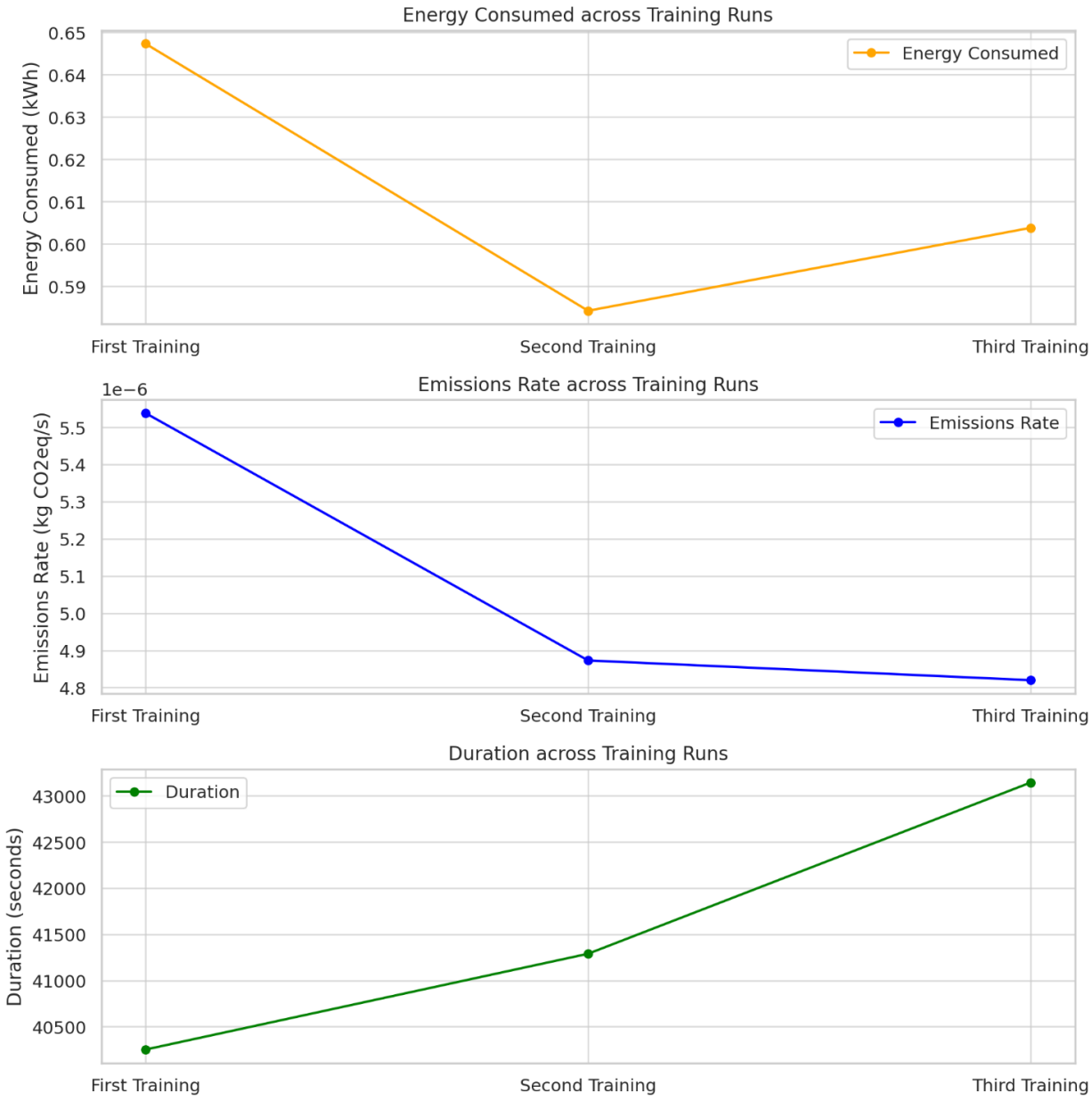



*Figure 48 – Energy Consumption/ Emissions Rate/ Duration (LightGBM)*

| Training | Energy(kWh) | Emissions Rate (kg Co2eq/s) |
|---|---|---|
| **First** | 0.6473 | 5.538e-06 |
| **Second** | 0.5842 | 4.873e-06 |
| **Third** | 0.6038 | 4.82e-06 |

*Table 33 – LGBM Energy / Emissions*

### 5.3.2 CatBoost

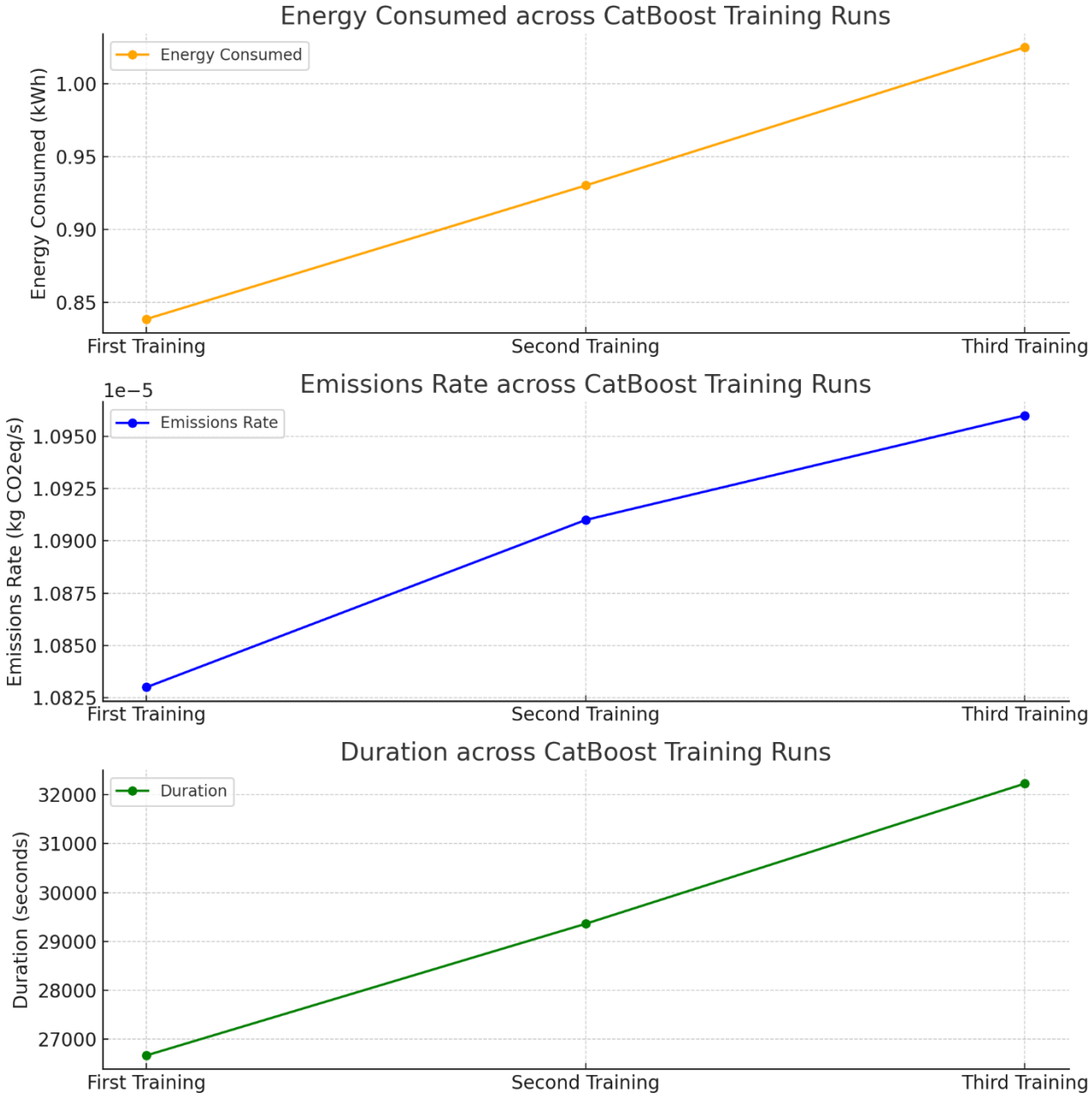



*Figure 49 – Energy Consumption/ Emissions Rate/ Duration (CatBoost)*

| Training | Energy(kWh) | Emissions Rate (kg Co2eq/s) |
|---|---|---|
| **First** | 0.83836 | 1.083e-05 |
| **Second** | 0.93022 | 1.091e-05 |
| **Third** | 1.02529 | 1.096e-05 |

*Table 33 – CatBoost Energy / Emissions*

## 5.4 Evaluation of Pipelines

The evaluation of the LightGBM and CatBoost pipelines by different measures of performance, computational efficiency, and environmental impact points toward different strengths and limitations for each model. Similar to both, the models would be trained, followed by categorizing data into fraud, no_fraud, and suspicious tables. The suspicious category was labeled with the use of Random Forest. That newly labeled fraud data appended into the training set could also be used for model retraining. Therefore, LightGBM is better to go with for the overall framework based on the results. The reasons with a comparison of the results are described in detail below.

LightGBM has always outperformed for all main metrics and improved iteration after iteration. It reached 0.9206 in precision for the first run and 0.952 for the third. Recall increased from 0.7353 to 0.811, showing that the model is capable of detecting fraudulent cases while keeping false negatives low. The F1-score, providing a balance between precision and recall, increased from 0.8175 to 0.8758, hence very good general performance. The ROC AUC score has also risen from 0.9731 to 0.9797, showing a high capability of distinguishing between fraud and non-fraud transactions.

By comparison, the CatBoost was competitive but slightly lagged in precision and F1-score. Precision for CatBoost increased from 0.8621 in the first run to 0.8829 in the third run, while recall increased from 0.8047 to 0.844. Its F1-score only peaked at 0.8630 in the third run, remaining far below LightGBM's through iterations. Even though the ROC AUC score was close, at 0.9789 for CatBoost in the third run, the model will turn out to be less effective for tasks with high predictive accuracy at minimal false positives due to the lower precision and F1-score.

The superiority of LightGBM is further indicated by the labeling phase, which is based on Random Forest. While both pipelines employed Random Forest to label suspicious data, LightGBM always outperformed CatBoost in precision and F1-scores during the labeling phase. In the case of LightGBM, the Random Forest classifier achieved a precision of 0.981 in the third run, while in the same iteration, the precision dropped to 0.919 for CatBoost's labeling phase. Similar differences are demonstrated by LightGBM with higher F1-scores in all runs, meaning a better balance between the identification of fraud and reduction of false positives during labeling.

CatBoost shows more variance in the outcomes of its labeling, such as with its Random Forest recall decreasing significantly from 0.695 on the first run to 0.289 on the third. The decline suggests difficulties in the handling of ambiguous cases from the suspicious category in the course of pipeline iterations.

Execution time and emission are two important parameters that describe how efficiently and eco-friendly the performance of the pipelines is. For execution time, CatBoost showed higher speeds, since it executed its first run in 7 hours and 24 minutes and its third run in 8 hours and 57 minutes, while LightGBM took 11 hours and 10 minutes and 11 hours and 59 minutes for the same runs, respectively. This makes CatBoost more time-efficient.

However, LightGBM was very environmentally efficient. Its level of emissions went down from 0.2229 kg $CO_2$ in the first run to 0.2079 kg $CO_2$ in the third run, while that of the CatBoost model started at 0.2887 kg $CO_2$ and increased to 0.3531 kg $CO_2$, reflecting how much more it consumed resources though with the shortest execution time. This trade-off against time and environmental impact makes LightGBM more sustainable.

The results indicated that LightGBM fits best in the framework for more than one reason, since it showed higher accuracy. Indeed, LightGBM gave consistently high precision, recall, and F1-scores across all iterations. These metrics are really critical in fraud detection, as false positives result in unnecessary disruptions while false negatives lead to undetected fraudulent activity.

**Improved Labeling Performance**: The Random Forest classifier in LightGBM's pipeline outperformed CatBoost in labeling the suspicious category with higher precision and F1-scores. This consistency ensures that high-quality data is integrated into the iterative retraining process, further improving the model.

**Environmental Efficiency**: Although LightGBM took more execution time, its less emission makes it more eco-friendly, tuned to the latest awareness about computational efficiency and environmental care.

**Scalability and Generalization**: Robustness with big datasets combined with strong LightGBM's ROC AUC and precision-recall AUC score guarantees good generalization over unseen data. This makes it more applicable to real-world situations, as scalability and adaptability are featured here.

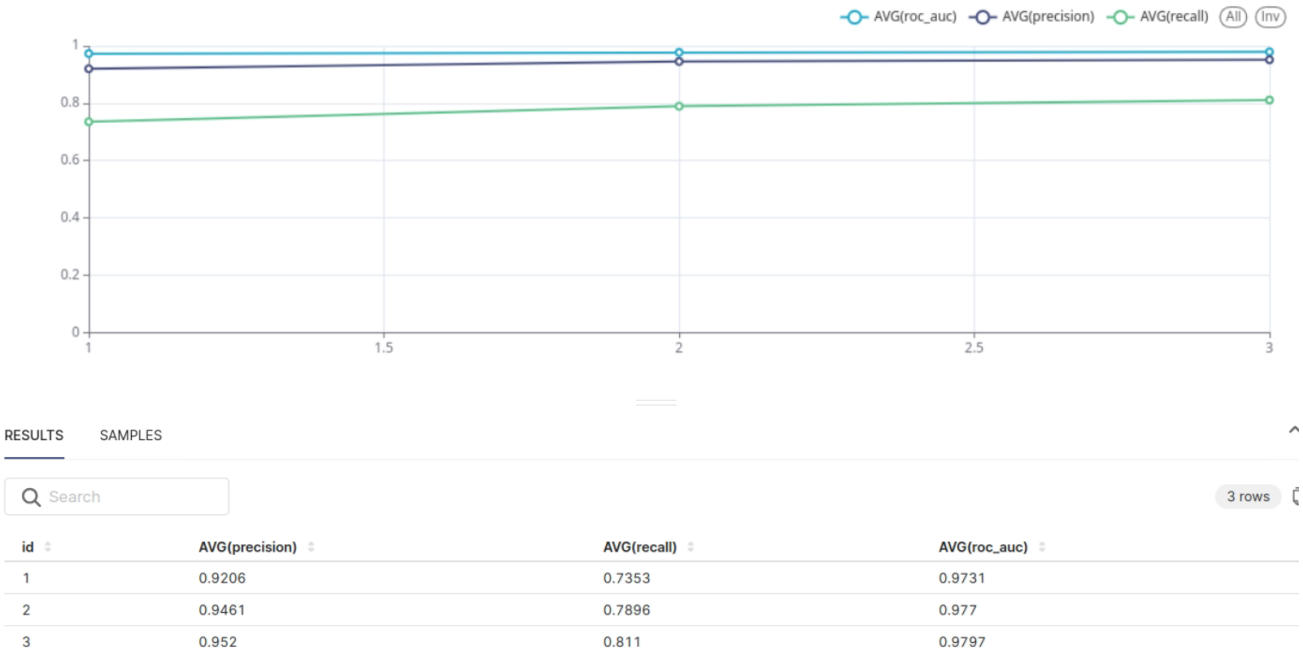
# 6 Visualizing Insights with Apache Superset

The last step of the fraud detection framework is to provide actionable insights to its end users via a strong Business Intelligence tool. This is an important phase that bridges the gap between the technical implementation of the framework and its practical utility by ensuring that the stakeholders are able to use the processed data effectively.

Key metrics will be developed to visualize and interpret an opensource BI platform such as Apache Superset. Superset allows defining and tracking KPIs on raw or processed data, enabling the teams to monitor several aspects of the system performance. However, in this particular implementation, Superset was used to build dashboards that would visualize the model evaluation metrics comprising Model Precision, Model Recall, and Model AUC (ROC_AUC).

This intuitive, user-facing interface will give a nontechnical end user insight into the performance of the system and evaluation of its outputs in near real time. Apache Superset serves a fundamental purpose in practical fraud detection framework deployment by offering the capability to make informed decisions with well-defined, easily accessible metrics.

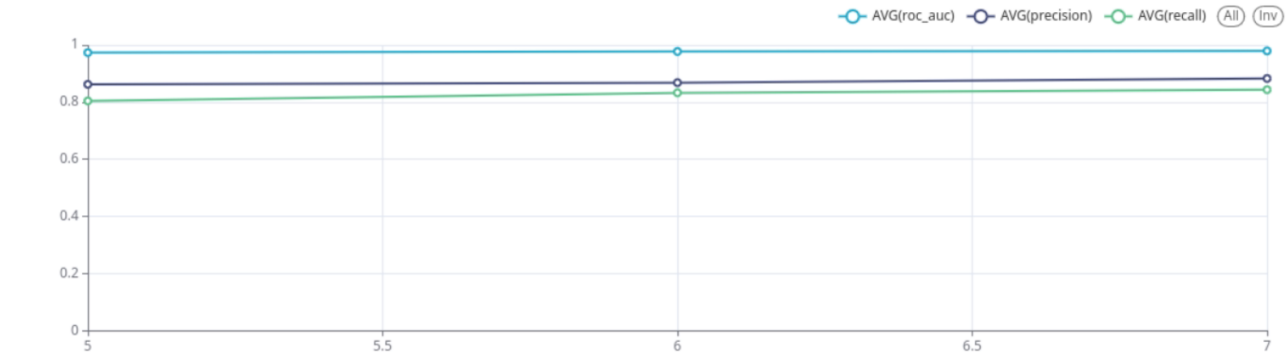
## 6.1 LGBM




| id | AVG(precision) | AVG(recall) | AVG(roc_auc) |
|---|---|---|---|
| 1 | 0.9206 | 0.7353 | 0.9731 |
| 2 | 0.9461 | 0.7896 | 0.977 |
| 3 | 0.952 | 0.811 | 0.9797 |

*Figure 50 - LightGBM Superset Metrics*

## 6.2 CatBoost




RESULTS SAMPLES

Search

3 rows

| id | AVG(precision) | AVG(recall) | AVG(roc_auc) |
|---|---|---|---|
| 5 | 0.8621047174701918 | 0.8047423179288652 | 0.9733120656359847 |
| 6 | 0.8679163973281621 | 0.8327475708083523 | 0.9772862299332896 |
| 7 | 0.882875765356508 | 0.8440332326283988 | 0.9789638675057897 |

*Figure 51 - CatBoost Superset Metrics*

# 7 Conclusion

This thesis has explored and compared LightGBM and CatBoost algorithms of advanced machine learning pipelines used in fraud detection within a financial transaction. In view of this, a framework is implemented which is built upon iterative training, classification, and labeling suspicious data dynamically to construct a pretty robust system that could recognize fraudulent activities effectively. Upon drawing detailed performance analysis, it emerged that LightGBM presents the best model for the proposed framework due to its good precision, F1-score, and environmental efficiency. In the real-world application, even though CatBoost executes faster, the overall balance between accuracy, sustainability, and scalability will lead to choosing LightGBM for deployment.

These results emphasize how effective an iterative retraining approach could be, with the model learning progressively from the data labeled by Random Forest. This strategy improved the results in both pipelines across successive iterations and demonstrated how the framework is able to adapt to changing fraud patterns.

In the future, the scope of data analysis would be extended to deeper insights and refinement of feature engineering. Extensive explorative data analysis shall be carried out to bring forth many hidden patterns and trends that may help in enhancing model performance even further. Another critical task will involve deploying this fraud detection framework as a web application. The application is designed in such a way that it is data-agnostic and optimized for banking datasets with fraud-related problems, adaptable to challenges in other sectors.

A web-based user interface allows stakeholders to input data easily and obtain real-time predictions, thus further enhancing the system's accessibility and ease of use. Furthermore, scalability and adaptability are designed into the system for increasingly complex and larger data, supported by robust infrastructure and computing power. It will, therefore, be developed further into a practical, efficient, and complete fraud case detection solution that will meet the demands of modern financial systems and related industries.

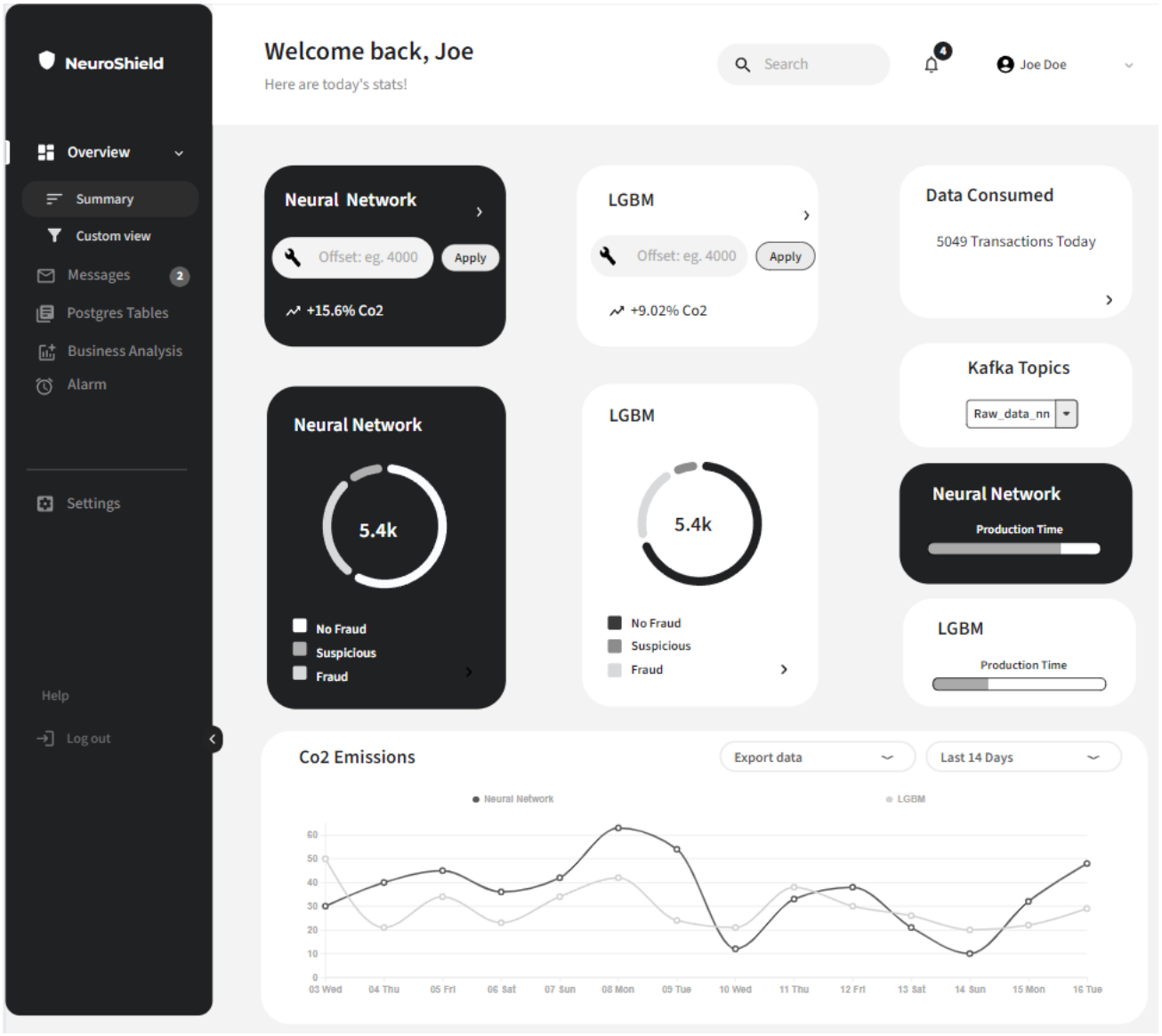



*Figure 52 - Web user interface*

This research laid the ground for a scalable and adaptive fraud detection system. The results emphasized the potential of machine learning and iterative feedback loops in solving complex fraud detection problems, while future developments will mold this framework into a versatile tool for the financial industry and beyond.

# 8 References


| | |
|---|---|
| **1** | Ngai, E. W. T., Hu, Y., Wong, Y. H., Chen, Y., & Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic literature review. Decision Support Systems, 50(3), 559-569. |
| **2** | Phua, C., Lee, V., Smith, K., & Gayler, R. (2010). A comprehensive survey of data mining-based fraud detection research. Artificial Intelligence Review, 34(1), 1–14. |
| **3** | Bolton, R. J., & Hand, D. J. (2002). Statistical fraud detection: A review. Statistical Science, 17(3), 235–249. |
| **4** | Zhang, Y., Yang, J., & Wu, B. (2017). Fraud detection via unsupervised learning. Computers & Security, pp. 75, 64–74. |
| **5** | Bahnsen, A. C., Aouada, D., & Ottersten, B. (2016). Example-dependent cost-sensitive decision trees. Expert Systems with Applications, 42(19), 6609-6619. |
| **6** | Hawkins, D. M., He, H., & Williams, G. J. (2002). Outlier detection using replicator neural networks. Machine Learning, 58(1), 143–172. |
| **7** | Heaton, J. B. (2016). Creating Features for Machine Learning. *Communications of the ACM, 59*(11), 64–72. |
| **8** | Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T. Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Proceedings of the 31st International Conference on Neural Information Processing Systems. |
| **9** | Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2016). Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing. |
| **10** | Kaggle. (2019). IEEE-CIS Fraud Detection Dataset. Available at: https://www.kaggle.com/competitions/ieee-fraud-detection |
| **11** | Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. |
| **12** | Prokhorenkova, L., Gusev, G., Vorobev, A., Veronika Dorogush, A., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. arXiv preprint arXiv:1810.11363. |
| **13** | Dorogush, A. V., Ershov, V., & Gulin, A. (2018). CatBoost: gradient boosting with categorical features support. Proceedings of the IEEE International Conference on Machine Learning. |
| **14** | Amershi, S., et al. (2019). Software Engineering for Machine Learning: A Case Study. Proceedings of the 41st International Conference on Software Engineering. |
| **15** | Mitchell, M., et al. (2019). Model Cards for Model Reporting. Proceedings of the Conference on Fairness, Accountability, and Transparency. |
| **16** | He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263–1284. |

| | |
|---|---|
| **17** | Zheng, Z., Zhang, H., & Han, J. (2020). A hybrid approach for fraud detection using machine learning techniques and expert knowledge. IEEE Transactions on Neural Networks and Learning Systems, 31(1), 1-12. |
| **18** | Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A Distributed Messaging System for Log Processing. Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB). |
| **19** | Narkhede, N., Shapira, G., & Palino, T. (2017). Kafka: The Definitive Guide. O'Reilly Media. |
| **20** | Aggarwal, C. C. (2016). Outlier Analysis (2nd ed.). Springer. |
| **21** | James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: With Applications in R. Springer. |
| **22** | Rowe, L. A., & Stonebraker, M. R. (1987). The POSTGRES Data Model. *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, 83–96. |
| **23** | Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1137–1145. |
| **24** | Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324 |