Conestoga College

School of Applied Computer Science & Information Technology

SENG8081 - Case Studies Big Data

Credit Card Fraud Detection Using Big Data Tools

Team Members

Phani Mallampati

Vikas Manchala

Roshan Bartaula

Satyam Patel

**Abstract:**

Credit card fraud is a growing problem, and catching it quickly is a challenge for banks and businesses. In this project, we set out to build a working example of a fraud detection system using only free, open-source big data tools. We use two real-world datasets from Kaggle, each with thousands of credit card transactions, to test our approach. Our setup includes tools like Kafka for streaming data, HDFS for storage, and Spark for analysis and machine learning. Everything runs locally or on college computers—no paid services or cloud platforms. Along the way, we look at how to collect, clean, and analyze the data, and we try out some machine learning methods to spot unusual transactions. The goal is to show what’s possible with open tools and to learn what works when tackling fraud detection with big data.

# Contents

[1. Contents 4](#_Toc201774359)

[2. Introduction: 5](#_Toc201774360)

[3. Data Research and Integration 5](#_Toc201774361)

[4. Data Collection 6](#_Toc201774362)

[5. Data Storage and Maintenance 7](#_Toc201774363)

[6. Data Quality 8](#_Toc201774364)

[7. Data Analysis and Visualization 8](#_Toc201774365)

[8. Future Plans 9](#_Toc201774366)

[9. Project Team Roles 10](#_Toc201774367)

[10. Project Timeline 10](#_Toc201774368)

# Introduction:

Credit card fraud is a constant concern for both consumers and financial institutions. With the sheer volume of transactions happening every second, it’s not practical to check each one by hand. That’s where big data tools come in—they can help process and analyze huge amounts of information quickly, making it possible to spot suspicious activity as it happens.

For this project, we’re building a hands-on example of a fraud detection system using only open-source software. We’re working with two large datasets of real credit card transactions, both available for free on Kaggle. Our goal is to set up a pipeline that can handle everything from collecting and storing the data to analyzing it and flagging possible fraud.

We’re using tools like Kafka to simulate real-time data streams, HDFS to store the data, and Spark for analysis and machine learning. Everything is designed to run on local machines or college resources, so there’s no need for paid cloud services. By the end of the project, we hope to have a working system that shows how these tools can be used together to tackle real-world problems like fraud detection.

# Project Team Roles

To ensure a smooth workflow and clear accountability, we’ve divided the project responsibilities as follows:

* Vikas Manchala - Visualization and Reporting
* Roshan Bartaula - Data Engineer
* Phani Mallampati - Team Lead & ETL Pipeline Engineer
* Satyam Patel - ML Engineer

Each team member takes primary responsibility for their assigned area, ensuring that all aspects of the project are covered. Regular communication and collaboration are maintained through weekly meetings, shared documents, and GitHub for code management and issue tracking.

# Project Timeline

To keep the project on track, we’ve set out a schedule with key milestones and who’s responsible for each part. This helps us stay organized and makes sure nothing gets missed as the deadline approaches.

|  |  |  |
| --- | --- | --- |
| **Date** | **Deliverable** | **Responsible** |
| 08-Jun | GitHub repository created | Phani |
| 08-Jun | First dataset selected and collected | Roshan |
| 22-Jun | Second dataset collected | Roshan |
| 22-Jun | Initial project report draft | All |
| 30-Jun | Data download scripts finalized | Roshan |
| 05-Jul | Data cleaning and quality checks | Roshan, Satyam |
| 12-Jul | ETL pipeline and storage setup | Phani |
| 19-Jul | Initial data analysis and EDA | Satyam, Vikas |
| 26-Jul | Machine learning model development | Satyam |
| 02-Aug | Visualization and reporting draft | Vikas |
| 05-Aug | Final review and adjustments | All |
| 10-Aug | Project report and presentation due | All |

We’ll check in each week to review progress and make changes to the plan if needed. If any task takes longer than expected, we’ll adjust our schedule to make sure everything is finished on time.

# Data Research and Integration

For this project, we’re using two datasets that are both available to the public on Kaggle. The first one is a well-known credit card fraud dataset with nearly 285,000 transactions, each labeled as either fraudulent or legitimate. The second dataset is more recent and even larger, with over half a million records. Both datasets include details like transaction time, amount, and a set of anonymized features.

We chose these two because they offer a good mix of size and variety. They come from different sources and cover different time periods, which helps us test our approach on more than one set of data. Since both datasets use similar formats, it’s possible to combine them for some parts of the analysis, though we’ll also look at them separately to see if patterns change over time or between sources.

We’re focusing on the features that both datasets share. If we need to merge them, we’ll line up the columns that match and make sure the data types are consistent. Any differences in structure or missing information will be handled during the data cleaning step.

Dataset 1: [Credit Card Fraud Detection (Kaggle)](https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud)

Contains 284,807 transactions with features such as time, amount, anonymized variables, and a fraud label.

Dataset 2: [Credit Card Fraud Detection Dataset 2023 (Kaggle)](https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023)

Contains over 550,000 transactions with similar features, updated for 2023.

# Data Collection

Getting the data into our system is a key first step. Since we're using Kaggle datasets, we've opted to use the Kaggle API to download the data directly. This lets us automate the process and makes it easy to update the datasets later if needed.

We've created a Python script that uses the Kaggle API to download both datasets. The script takes care of authenticating with Kaggle, specifying which datasets to download, and extracting the data into the appropriate folders. This script is included in our project repository, so anyone can easily grab the data themselves.

One challenge we faced was the size of the second dataset, which is around 300MB. GitHub has limits on file sizes, so we can't store the dataset directly in the repository. Instead, we store the download script and instructions, so users can download the data on their own.

In a real-world scenario, we'd likely be pulling data from multiple sources in real time. To simulate this, we could modify our script to download new data at regular intervals, or even use a tool like Apache Flume to ingest data from various sources.

* Tools Used: Python scripts, Kaggle API
* Process:
  + Download datasets using scripts (download\_dataset2.py).
  + Store raw data in the data/ directory.
* Challenges:
  + Managing large file sizes (over 300MB for Dataset 2)
  + Ensuring data is up-to-date and not duplicated

# Data Storage and Maintenance

Once the data is collected, it needs to be stored somewhere that’s both reliable and easy to access. For our project, we’re starting with local storage, keeping the raw files on our own machines. This works well for development and testing, especially since we’re working with a manageable amount of data.

If we were dealing with much larger datasets or needed to support more users, we’d look at distributed storage. Our plan is to use HDFS (Hadoop Distributed File System) for this purpose. HDFS is designed to handle big data and makes it possible to store and process files that are much too large for a single computer.

We’re also thinking about how to keep the data organized and safe. That means setting up folders for raw data, cleaned data, and any processed results. Regular backups are important, even for a student project, so we’re making sure to keep copies of the original files in case something goes wrong during processing.

As for the future, if the project were to grow, we’d need to consider things like data retention policies and access controls to make sure only authorized people can see sensitive information.

# Data Quality

Before we start analyzing the data, it’s important to make sure it’s accurate and consistent. Data quality can have a big impact on the results, so we need to check for any issues early on.

One of the first things we’ll do is look for missing values. If some transactions are missing key information, we need to decide how to handle them. We might choose to ignore those transactions, fill in the missing values with estimates, or use more advanced techniques to predict the missing data.

We’ll also check for duplicate transactions. If the same transaction appears multiple times in the dataset, it could skew our analysis. We’ll need to identify and remove any duplicates to make sure our results are accurate.

Another important step is to look for outliers—transactions that are very different from the norm. These could be legitimate but unusual transactions, or they could be signs of fraud. We’ll use statistical methods and visualizations to identify outliers and decide whether to include or exclude them from our analysis.

Finally, we’ll validate the data types and ranges. For example, transaction amounts should be positive , and timestamps should fall within a reasonable range. If we find any values that don’t make sense, we’ll need to investigate and correct them.

# Data Analysis and Visualization

Once the data is cleaned up, we can start digging into it to see what stories it tells. Our first step is to get a sense of the overall patterns—how many transactions are in each dataset, how often fraud actually happens, and what the typical transaction looks like.

We’ll use simple charts and graphs to show things like the distribution of transaction amounts, the times of day when fraud is most common, and how the number of fraudulent transactions compares to legitimate ones. These visualizations help us spot trends and outliers that might not be obvious from just looking at the raw numbers.

For the machine learning part, we’re planning to use Spark’s MLlib library. We’ll start with unsupervised learning methods, like Isolation Forest, to try to pick out transactions that look unusual or suspicious. Since fraud is rare compared to normal transactions, these methods can help us flag potential problems without needing a huge number of labeled examples.

We’ll also look at how well our models perform by checking metrics like precision and recall. If we have time, we might try out other algorithms or tweak our approach based on what we learn from the data.

All of our analysis and visualizations will be included in the final report, with clear explanations of what we found and what it might mean for real-world fraud detection.

# Future Plans

If we wanted to take this project further, there are a few clear ways to scale it up. Right now, everything runs on our own computers and uses a limited amount of data. In a real-world setting, we’d need to handle much more information, possibly coming in from different banks or payment systems all at once.

The first step would be to move our storage to something like HDFS, which is built for handling huge datasets across many machines. We’d also want to set up Kafka and Flume to bring in data from multiple sources in real time, instead of just downloading files every so often.

Processing would need to be distributed as well. Using a Spark cluster, we could analyze data as it arrives, flagging suspicious transactions right away. If the project kept growing, we’d look at cloud-based solutions, but for now, everything is designed to work with free, open-source tools.

We’d also have to think about how much storage we’d need over time. For example, if we started getting a million transactions a day, we’d need to plan for terabytes of data each year. Setting up regular backups and deciding how long to keep old data would become more important as the system grows.