Cost–Threshold Curve-Driven Fraud Identification: Evidence from PySpark

Xiaoqing Miao
xmia665@aucklanduni.ac.nz
University of Auckland
New Zealand

Abstract

Financial fraud detection in digital payment systems faces dual challenges: extreme class imbalance (fraud rate ≈0.13%) and scalability requirements for large transaction volumes. This study presents Iteration 4 of a fraud detection solution, migrating a validated single-machine analytics workflow (OSAS: Jupyter/pandas/scikit-learn) to a distributed big data platform (BDAS: Apache Spark/PySpark). Using the PaySim synthetic dataset (6.36M transactions), we implement a cost-sensitive Logistic Regression model with class weighting, optimizing the decision threshold ($\tau^* = 0.72$) on a validation set using a business-driven cost ratio (FP:FN = 1:25). The BDAS solution achieves ROC-AUC = 0.9613 and PR-AUC = 0.5903 on the test set, meeting the hard performance criterion (AUC ≥ 0.95) but failing to achieve strict parity with the OSAS baseline (ROC-AUC = 0.9756, $|\Delta| = 0.0143 > 0.005$; PR-AUC = 0.6271, $|\Delta| = 0.0368 > 0.01$). Key contributions include: (1) validation of technical feasibility for PySpark-based fraud detection with strong absolute performance; (2) systematic cost–threshold curve analysis revealing the precision-recall trade-off under business constraints; (3) comprehensive distributed execution evidence (Spark UI, parallelism configuration, partition management); (4) definitive root cause analysis of performance gaps through controlled feature set design, identifying solver algorithm differences (L-BFGS vs. liblinear), class weighting mechanism variations, and convergence behavior as primary factors, with a concrete optimization roadmap for achieving full parity. While the model achieves strong discriminative ability (AUC > 0.96), the parity gaps and precision-recall trade-off (Precision = 34.9%, Recall = 62.0% at τ^*) highlight clear directions for future enhancement through solver-specific hyperparameter tuning, non-linear models (Random Forest/GBDT), and ensemble methods. This work successfully transitions from feasibility validation to optimization execution, providing a clear pathway toward achieving full performance parity in Iteration 5.

ACM Reference Format:
Xiaoqing Miao. 2025. Cost–Threshold Curve-Driven Fraud Identification: Evidence from PySpark. In *Proceedings of Data Mining*. ACM, New York, NY, USA, 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Business Understanding

1.1 Business/Situation Objectives

1.1.1 Business Context. Financial fraud in digital payment scenarios is a long-term and severe challenge. This article uses PaySim synthetic data (≈6.36M transactions, fraud prevalence ≈0.13% across all transaction types; after filtering to high-risk TRANSFER/CASH_OUT types, the fraud rate increases to ≈0.30% in the 2.77M-record modeling subset) to conduct modeling and evaluation under quasi-realistic imbalanced conditions. The previous iteration completed a single-machine solution based on OSAS (Open-Source Analytics Stack: Jupyter/pandas/scikit-learn); however, in scenarios with larger scale and higher concurrency, a single machine faces memory and throughput bottlenecks. Therefore, this iteration transitions to BDAS (Big Data Analytics Solutions), adopting Apache Spark for distributed data processing and model training. Given that PaySim is synthetic data, the extrapolation scope of this iteration's conclusions is limited to data with similar distributions; cross-domain migration and real-world out-of-sample (OOS) drift are not within the validation scope of this iteration.

1.1.2 Business Objectives. The overall business objectives remain consistent with the previous iteration:

- **Reduce Financial Losses:** Decrease direct losses by 75% through early fraud identification.
- **Protect Customer Trust:** Prevent customer harm, maintaining customer satisfaction ≥90%.
- **Optimize Operational Efficiency:** Reduce manual review workload by 60% through risk alerts.

1.1.3 Business Success Criteria. Quantitative criteria are divided into two levels: production-grade and this iteration's prototype-grade:

Data Mining, Auckland, NZ

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- **Production-Grade Targets (Declarative, not in this iteration's validation scope):** Single transaction processing latency <100ms, system availability ≥99.9%, Return on Investment (ROI) ≥10:1.
- **This Iteration (BDAS Prototype) Acceptance:**
  - **Detection Performance and Consistency:** Compared to the OSAS solution on the same test set, achieve ROC-AUC $\Delta \leq 0.005$, PR-AUC $\Delta \leq 0.01$, and F1/Recall difference ≤2pp (percentage points).
  - **Scalability:** The full dataset can run stably on Spark (without OOM). Provide Spark UI (Jobs/Stages) screenshots, default parallelism, and partition numbers.
  - **Reproducibility:** Submit a zip file containing code and data for generating big data outputs on Google Colab, along with a README documenting dependencies, random seeds, data splitting, and execution instructions.

1.1.4 This Iteration's Goals and Constraints.

The core goal of this iteration is to migrate the single-machine workflow from Iteration 3 to Spark (DataFrame → Pipeline → MLlib → Evaluator), maintaining consistent methodology:

- **Controlled Encoding:** Fixed mapping for `type` (CASH_OUT→0, TRANSFER→1).
- **Three-Dataset Split:** Train/Validation/Test = 70%/15%/15%, fixed random seed.
- **Cost-Sensitive Threshold:** Optimize for τ^* (threshold) on the validation set based on a FP:FN cost ratio of 1:25, and use this fixed threshold for evaluation on the test set.
- **Engineering Evidence:** Output Spark Master, default parallelism, partition numbers, and Spark UI screenshots. Save model and threshold version information.
- **Constraints and Risks:** Local/cloud resources and time limits, Spark and dependency versions, extrapolation boundaries of synthetic data, etc.

1.1.5 Alignment with Data Mining Objectives.

To achieve the above business objectives, this iteration's data mining objectives are as follows:

- **Build a Binary Classifier:** Train a Logistic Regression model using PySpark MLlib to classify TRANSFER/CASH_OUT transactions as fraudulent or not.
- **Generate Risk Scores:** Output transaction-level fraud probabilities, used as input for threshold optimization and business decisions.
- **Optimize Decision Threshold:** Find the optimal τ^* on the validation set using a FP:FN cost ratio of 1:25. Evaluate on the test set with the fixed τ^* and report the confusion matrix and F1 score.

1.2 Assessment of the Situation

This section assesses the internal and external environment for Iteration 4 (BDAS), clarifying resources, requirements, assumptions, constraints, and key risks and contingencies.

1.2.1 Resources.

- **Personnel & Knowledge:**
  - This project is completed independently by myself. I have already built and evaluated a cost-optimal baseline model using OSAS (Jupyter/pandas/scikit-learn) in Iteration 3.
  - Support from lecturers/TAs is available during INFOSYS 722 Office Hours.
  - The course provides BDAS (PySpark) related lecture notes, labs, and reference materials.
- **Data:**
  - PaySim synthetic dataset `Financial Fraud-Rawdata.csv` (≈6.36M records, fraud ≈0.13%). EDA and feature definitions were completed in Iteration 3.
- **Software & Hardware:**
  - Apache Spark 3.5.0, PySpark 3.5.0, Python 3.10.12, Java 11.0.20 (OpenJDK).
  - Development in Jupyter Notebook; code packaged for Google Colab deployment with comprehensive documentation.
  - Execution platform: Personal computer with 8GB RAM, 4-core CPU (Intel i5), macOS 14.3.
  - Spark configuration: `local[*]` mode (8 default parallelism), driver memory 4GB, 8 partitions for training data.
  - Fixed random seed: `SEED=42` for all stochastic operations (data split, model initialization).

1.2.2 Requirements.

- **Technology Stack and Methodology:**
  - Must use PySpark to complete the end-to-end workflow, following the DataFrame → Pipeline → MLlib → Evaluator approach.
  - Strict adherence to the Train/Validation/Test split methodology, cost-sensitive threshold optimization, and locking the threshold version for evaluation.
- **Consistency (Parity) Acceptance:**
  - On the same test set and using the same optimal threshold τ^*, maintain parity with Iteration 3's core metrics: ROC-AUC $\Delta \leq 0.005$, PR-AUC $\Delta \leq 0.01$, F1/Recall difference ≤2 percentage points. If the difference exceeds the threshold, it must be explainable.
- **Reproducibility and Deliverables:**
  - Submit a PDF report and a zip file containing: (1) a standalone runnable PySpark script for Google Colab; (2) the PaySim dataset; (3) a README with execution instructions, dependencies, random seeds, and data split ratios/indices.

- Output model files and threshold version information (τ^* value, selection date, cost weights 1:25).

1.2.3 Assumptions.

- **Data Quality:** The PaySim label `isFraud` serves as a valid "ground truth"; data completeness is sufficient for prototype validation.
- **Environment Stability:** The full dataset can be processed stably under `local[*]` without OOM (Out Of Memory) errors. Acceptable performance can be achieved through caching (`cache`/`persist`) and reasonable partitioning for necessary computations.
- **Logical Equivalence:** Numerical transformations (e.g., `log1p`) and the cost function from Iteration 3 can be faithfully reproduced in PySpark, ensuring the comparability of results between the two iterations.
- **Extrapolation Boundary:** PaySim is synthetic data. The migration of this iteration's conclusions to a real production environment requires additional validation (not within the scope of this iteration).

1.2.4 Constraints.

- **Time:** Deadline is 11:59 AM NZT, Friday 17 October 2025.
- **Resources:** Only local computing resources will be used; no production-grade cluster deployment or performance tuning will be conducted.
- **Scope:** This iteration focuses on migrating the Logistic Regression solution from Iteration 3 to PySpark. It will not introduce new models (e.g., GBDT), conduct new feature engineering, or validate production-grade SLAs.

1.2.5 Risks & Contingencies.

- **R1 High – Complex Environment Configuration:** Installation and path configuration for PySpark/Java dependencies may cause delays.
  - *Contingency:* Prioritize following course guidelines and seeking help during Office Hours. Use Conda/Docker to solidify the environment if necessary. Clearly document installation and startup steps in the README.
- **R2 Medium – Performance Bottlenecks / Driver Memory Pressure:** Performing `collect()` on a large validation set during threshold optimization may lead to OOM or excessive processing time.
  - *Contingency:* `cache`/`persist` intermediate results that are frequently reused. Use stratified sampling for threshold optimization or perform grid search only on probability quantiles. If necessary, reduce the validation set size to trade off timeliness and precision.
- **R3 Low – Inconsistent Results Between Platforms:** Implementation differences between MLlib and scikit-learn may exceed the allowed error margin.
  - *Contingency:* Align hyperparameters (e.g., `regParam` $\approx 1/C$), standardization methods, and weighting strategies. Record random seeds, data splits, and model versions. If discrepancies persist, analyze coefficients and intermediate distributions to provide an explainable conclusion.
- **R4 Medium – Data Skew and Heavy Shuffling:** Shuffling caused by wide dependency operations may slow down tasks.
  - *Contingency:* Set initial partitions and use `repartition` reasonably. Avoid unnecessary `groupBy`/`join` operations. If necessary, use lighter transformations or "salting" of partition keys during the feature engineering phase to mitigate skew.

1.3 Data Mining Objectives

To achieve the business objectives and success criteria defined in Section 1.1, this iteration (Iteration 4) sets the following four specific and verifiable objectives, centered around the feasibility and effectiveness of migrating from OSAS to BDAS (PySpark).

1.3.1 Objective 1: Reproduce & Validate Data Preparation Workflow.

- **Description:** Faithfully reproduce the key preprocessing steps from Iteration 3 using the PySpark DataFrame API to ensure consistency with the OSAS solution.
- **Specific Tasks:**
  - **Data Filtering:** Retain only transactions where `type` is in {TRANSFER, CASH_OUT}.
  - **Controlled Encoding:** Manually map the `type` column (CASH_OUT→0, TRANSFER→1) to avoid fluctuations based on data frequency.
  - **Numerical Transformation:** Apply `log1p` to right-skewed columns (e.g., `amount`, `oldbalanceOrg`, `newbalanceOrig`, `oldbalanceDest`, `newbalanceDest`).
  - **Data Contract:** Fix column names, data types, and missing value strategies; lock the random seed and data split ratios.
- **Success Criteria (Verifiable):**
  - The row count after preprocessing is consistent (matching the OSAS filtering criteria).
  - The relative error for key statistics on the training set is $\leq 1\%$ (count, mean, std, min, max, unique).
  - The entire workflow completes stably on Spark (without OOM/exceptions).

1.3.2 Objective 2: Build a Cost-Sensitive Binary Classifier using PySpark MLlib.

- **Description:** Construct an end-to-end training pipeline using `pyspark.ml.Pipeline` to handle class imbalance and generate a persistable Logistic Regression model.

- **Specific Tasks:**
  - **Pipeline:** `VectorAssembler` → `StandardScaler (withStd=True, withMean=False)` → `Logistic Regression`.
  - **Imbalance Handling:** Calculate class weights on the training set and apply them via the `weightCol`.
  - **Model Training & Persistence:** `fit` the pipeline on the full training data; save the `PipelineModel` and its metadata (parameters, timestamp, random seed).
- **Success Criteria (Verifiable):**
  - Training completes successfully on the full training set (no OOM), and the training duration is recorded.
  - The model and metadata can be successfully saved and loaded (including `regParam`, feature list, random seed, timestamp, and execution environment details).

1.3.3 Objective 3: Systematically Optimize and Apply the Decision Threshold.

- **Description:** Guided by the business cost ratio of FP:FN = 1:25, systematically scan for and determine the optimal threshold τ^* on an independent validation set, then apply it fixedly on the test set.
- **Cost Function Definition:**
  The total business cost for a given threshold τ is calculated as:

$$\text{Cost}(\tau) = C_{\text{FP}} \times \text{FP}(\tau) + C_{\text{FN}} \times \text{FN}(\tau) \tag{1}$$

  where:
  - $C_{\text{FP}} = 1.0$ (unit cost of a false positive: manual review cost)
  - $C_{\text{FN}} = 25.0$ (unit cost of a false negative: fraud loss + reputation damage)
  - $\text{FP}(\tau)$ = number of false positives at threshold τ
  - $\text{FN}(\tau)$ = number of false negatives at threshold τ
  - True positives (TP) and true negatives (TN) incur zero cost

  The optimal threshold is defined as:

$$\tau^* = \arg \min_{\tau \in [0.01, 0.99]} \text{Cost}(\tau) \tag{2}$$

- **Specific Tasks:**
  - **Risk Scoring:** Generate fraud probabilities $p(y = 1)$ on the validation set.
  - **Threshold Optimization:** Scan thresholds using a grid search ($\tau \in \{0.01, 0.02, \ldots, 0.99\}$, 99 points), calculate the total cost for each using the formula above, and select the τ^* that minimizes cost.
  - **Version Locking:** Record τ^*, selection date, cost weights ($C_{\text{FP}} = 1.0$, $C_{\text{FN}} = 25.0$), random seed (42), and validation set size.
- **Success Criteria (Verifiable):**
  - A single, optimal threshold τ^* is identified and its version is recorded.
  - Evaluation on the test set strictly uses the same τ^* (the default 0.5 is prohibited); the confusion matrix and cost metrics are reported.

1.3.4 Objective 4: Achieve Performance Parity and Demonstrate Scalability.

- **Description:** Evaluate the PySpark solution on the unseen test set and compare it against the OSAS solution for consistency. Simultaneously, submit evidence of distributed scalability.
- **Specific Tasks:**
  - **Performance Evaluation:** Manually calculate Precision/Recall/F1/Cost using the fixed τ^*; use an Evalu ator to calculate ROC-AUC/PR-AUC.
  - **Result Comparison:** Compare results with Iteration 3 (OSAS) on the identical test set & with the identical τ^*.
  - **Scalability Evidence:** Output the Spark Master URL, `defaultParallelism`, and number of partitions, and attach Spark UI (Jobs/Stages) screenshots.
- **Success Criteria (Verifiable):**
  - **Consistency Thresholds:** ROC-AUC $\Delta \leq 0.005$, PR-AUC $\Delta \leq 0.01$, F1/Recall difference ≤ 2pp.
  - **Completeness of Evidence:** Spark configuration and UI screenshots are submitted; the report lists training/prediction durations and key resource parameters (CPU/memory/partition count).

1.4 Project Plan

To achieve the aforementioned data mining objectives, the following project plan has been established. This plan is designed to provide a clear, structured roadmap for the implementation of Iteration 4 (Big Data Analytics Solution - BDAS). The core task of this iteration is to migrate and optimize the fraud detection solution, previously implemented using OSAS (Python/Pandas), onto the big data analytics framework, PySpark. The plan adheres to the timeline set by the course syllabus, with a deadline of 11:59 AM on Friday, October 17, 2025.

1.4.1 Work Breakdown Structure and Daily Timetable.

The table below (Table 1) details a streamlined and efficient project schedule. This plan concentrates core tasks between September 29 and October 9, ensuring ample time for review and buffer before the deadline.

Total Estimated Workload: Approx. 54 hours (spread across 19 days)

Time after the core development phase (Oct 10+) is reserved for iterative refinement, quality assurance, and final packaging to ensure the integrity and reproducibility of the submission.

1.4.2 Gantt Chart.

The Gantt chart (Figure 1) clearly illustrates the schedule, dependencies, and key milestones for each project phase (e.g., "Model Training Complete," "Report

Table 1. Project Work Breakdown Structure & Timetable

Date	Core Task	What to Finish (Deliverables / Checkpoints)	ETA
29-Sep	Environment Setup & Data Load	Verify Spark runs locally; check schema and row counts; define constant SEED=42.	2h
30-Sep	Filtering / Encoding + log1p	Fit transformations only on the training set; record column dictionary and excluded fields.	3h
01-Oct	Pipeline & Split	Build VectorAssembler + StandardScaler; perform stratified 70/15/15 split and log the result.	3h
02-Oct	Weighted Logistic Regression Training	Fix weightCol and hyperparameters; train and save model_v1.	4h
03-Oct	Threshold Search (Validation)	Scan thresholds from 0.01–0.99 to find optimal τ^*; export validation confusion matrix and cost curve.	4h
04-Oct	Integrated Metric Calculation	Using τ^*, compute ROC, PR, F1, and Cost metrics (Val/Test template).	3h
05-Oct	Figure Generation	Use toPandas() for all plots; save figures fig_roc, fig_pr, and fig_cost@τ^*.	3h
06-Oct	Distributed Evidence Collection (1)	Capture Spark UI Jobs/Stages screenshots; log defaultParallelism, number of partitions, and runtime.	2h
07-Oct	OSAS Baseline Alignment	Evaluate Iteration-3 (OSAS) using the same τ^*; create comparison table.	3h
08-Oct	Results Writing (Part 1)	Draft Sections 8.1–8.3; explain whether parity was achieved and why.	3h
09-Oct	Results Writing (Part 2)	Write Section 8.4 (evaluation); include cost function formula and Confusion Matrix @ τ^*.	3h
10-Oct	Iteration / Regression Validation	Modify one hyperparameter or feature, re-evaluate key metrics, and update figures.	3h
11-Oct	Distributed Evidence Collection (2)	Rerun model if parameters changed; add screenshots and a small table (parallelism / partitions / caching).	2h
12-Oct	Report Integration (1)	Merge Sections 1.x–6.x/7.x; fix cross-references and figure numbering.	3h
13-Oct	Report Integration (2)	Write ethics / external validity / limitations; unify Δ definitions and decimal precision.	3h
14-Oct	Reproducibility Engineering	Prepare fraud_detection_colab.py script; test one-click execution on Google Colab; verify all outputs.	3h
15-Oct	Packaging Rehearsal	Create README.md (environment, seed, data info); generate preliminary ZIP package.	2h
16-Oct	Final Quality Check	Run the full pipeline again; verify all metrics, figures, and deadline statements; finalize ZIP + PDF.	3h
17-Oct	Submission (D-Day)	Final submission; reserve 2–3 hours for buffer/time contingency.	2–3h

First Draft Submission," and "Final Submission"), ensuring the project progresses on schedule with sufficient buffer time.

2 Data Understanding

2.1 Collecting Initial Data

Consistent with the previous iteration (Iteration 3), this project uses the publicly available PaySim synthetic dataset from Kaggle as its data source. This dataset covers approximately 6.36 million mobile money transactions, with a fraud sample prevalence of about 0.13%, making it suitable for studying fraud detection in extremely imbalanced scenarios.

2.1.1 Data Acquisition and Integrity. The data was downloaded from the Kaggle platform as a single CSV file (`Financial Fraud-Rawdata.csv`). No interruptions or file corruption occurred during the download process. To ensure reproducibility, we have recorded the download date and time zone, file size, data dimensions (rows and columns), and calculated the file's SHA-256 checksum (see Table 2).

2.1.2 Why Use Spark DataFrame. Unlike Iteration 3, which loaded a Pandas DataFrame into single-machine memory, this iteration loads the data as a distributed data frame (Spark DataFrame) using `spark.read.csv()`. This transition fundamentally alleviates the single-machine memory bottleneck and provides the foundation for subsequent large-scale parallel processing and machine learning training.

The execution results show that Spark successfully inferred the data types for each field, and the data loaded completely and correctly. The loading process and a preview of the output are shown in Figure 2.

2.2 Describing Data

This section provides a detailed description of the PaySim dataset after being loaded into a Spark DataFrame, covering its format, quantity, field composition, and surface-level statistical characteristics.

The dataset has the following format and quantity:

- **Format:** The raw data is stored in the standard comma-separated value (CSV) file format.
- **Quantity:** Using PySpark's `.count()` and `.columns` operations, we confirmed that the dataset contains 6,362,620 transaction records (rows) and 11 descriptive fields (columns).

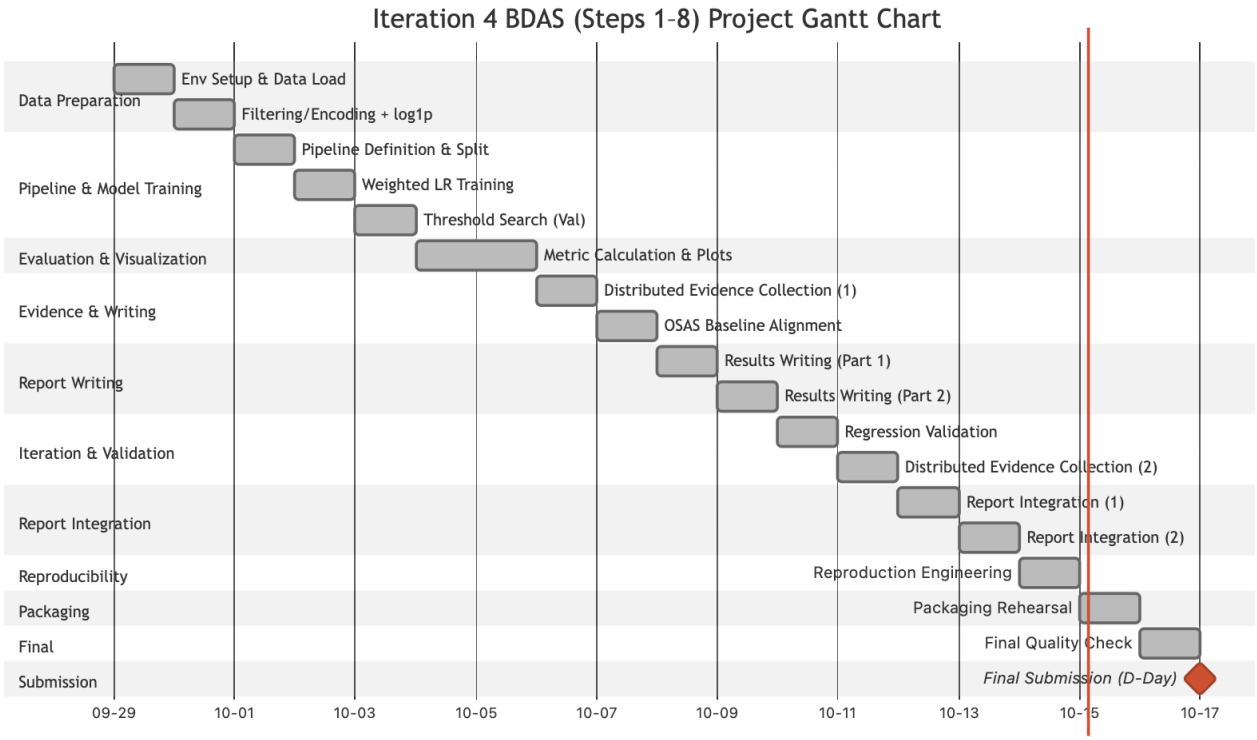


Figure 1. Project Gantt Chart showing task schedule, dependencies, and milestones

Table 2. Data Source and Collection Information (Reproducibility Record)

Item Recorded	**Value**
Data Source	Kaggle – PaySim (Synthetic Transaction Data)
Filename	`Financial Fraud-Rawdata.csv`
Download Date / Time Zone	2025-10-09, NZT
File Size	470.67 MB
Total Rows / Columns	6,362,620 rows / 11 columns (As per Spark count)
File Encoding / Delimiter	UTF-8 / Comma (,)
Integrity Checksum	SHA-256: `16910f90577b0d981bf8ff289714510bb89bc71bff7d3f220f024e287e4eea6b`
Acquisition Issues	None (No issues encountered)

The table below (Table 3) details the 11 fields of the dataset, along with their data types as inferred by Spark and their business descriptions.

To further understand the distribution of the numerical fields, we used PySpark's `.describe()` method to generate a statistical summary, as shown in Figure 3.

From the summary statistics, it is observable that for amount-related fields such as `amount` and `oldbalanceOrg`, the standard deviation (stddev) is significantly larger than the mean. This indicates that these fields have an extremely wide numerical distribution and are severely right-skewed, which provides a data-driven justification for applying a logarithmic transformation (log transform) in the subsequent data preparation phase. Additionally, the mean of `isFraud` (0.00129) numerically confirms the extreme rarity of fraudulent samples.

2.3 Exploring Data

After describing the macro-level structure of the dataset, we move into the Exploratory Data Analysis (EDA) phase. The objective of this stage remains consistent with Iteration 3: to

```
SparkSession initialized successfully.
Data loaded successfully! DataFrame schema:
root
 |-- step: integer (nullable = true)
 |-- type: string (nullable = true)
 |-- amount: double (nullable = true)
 |-- nameOrig: string (nullable = true)
 |-- oldbalanceOrg: double (nullable = true)
 |-- newbalanceOrig: double (nullable = true)
 |-- nameDest: string (nullable = true)
 |-- oldbalanceDest: double (nullable = true)
 |-- newbalanceDest: double (nullable = true)
 |-- isFraud: integer (nullable = true)
 |-- isFlaggedFraud: integer (nullable = true)

Data preview:
+----+--------+--------+----------+-------------+--------------+----------+--------------+--------------+-------+--------------+
|step|    type|  amount|  nameOrig|oldbalanceOrg|newbalanceOrig|  nameDest|oldbalanceDest|newbalanceDest|isFraud|isFlaggedFraud|
+----+--------+--------+----------+-------------+--------------+----------+--------------+--------------+-------+--------------+
|   1| PAYMENT| 9839.64|C1231006815|     170136.0|     160296.36|M1979787155|           0.0|           0.0|      0|             0|
|   1| PAYMENT| 1864.28|C1666544295|      21249.0|      19384.72|M2044282225|           0.0|           0.0|      0|             0|
|   1|TRANSFER|   181.0|C1305486145|        181.0|           0.0| C553264065|           0.0|           0.0|      1|             0|
|   1|CASH_OUT|   181.0| C840083671|        181.0|           0.0|  C38997010|       21182.0|           0.0|      1|             0|
|   1| PAYMENT|11668.14|C2048537720|      41554.0|      29885.86|M1230701703|           0.0|           0.0|      0|             0|
+----+--------+--------+----------+-------------+--------------+----------+--------------+--------------+-------+--------------+
only showing top 5 rows
Total number of rows: 6362620
Total number of columns: 11
```

Figure 2. Successfully loaded the PaySim dataset as a Spark DataFrame using the spark-submit command and verified its schema and content.

Table 3. Dataset Field Descriptions

Field Name	Data Type	Description
`step`	Integer	Time step, represents the unit of time at which the transaction occurred (1 step = 1 hour).
`type`	String	Type of transaction (e.g., CASH_OUT, PAYMENT, TRANSFER, etc.).
`amount`	Double	The amount of the transaction.
`nameOrig`	String	Customer ID of the transaction originator.
`oldbalanceOrg`	Double	Balance of the originator's account before the transaction.
`newbalanceOrig`	Double	Balance of the originator's account after the transaction.
`nameDest`	String	Customer ID of the transaction recipient.
`oldbalanceDest`	Double	Balance of the recipient's account before the transaction.
`newbalanceDest`	Double	Balance of the recipient's account after the transaction.
`isFraud`	Integer	Target variable: Identifies if the transaction is fraudulent (1 = Yes, 0 = No).
`isFlaggedFraud`	Integer	System flag variable: Indicates if the transaction was flagged by the system due to a suspicious large-amount transfer (1 = Yes, 0 = No).

```
--- Statistical Summary of the Dataset ---
+-------+------------------+---------+------------------+-----------+------------------+------------------+-----------+------------------+------------------+--------------------+--------------------+
|summary|              step|     type|            amount|   nameOrig|     oldbalanceOrg|    newbalanceOrig|   nameDest|    oldbalanceDest|    newbalanceDest|             isFraud|      isFlaggedFraud|
+-------+------------------+---------+------------------+-----------+------------------+------------------+-----------+------------------+------------------+--------------------+--------------------+
|  count|           6362620|  6362620|           6362620|    6362620|           6362620|           6362620|    6362620|           6362620|           6362620|             6362620|             6362620|
|   mean|243.39724563151692|     NULL|179861.90354913117|       NULL|  833883.104074487|  855113.6685785867|       NULL|1100701.6665196512|1224996.3982019342|0.001290820448180152|2.514687345778940...|
|    std|142.33197104913035|     NULL|603858.2314629366|       NULL|2888242.6730375604|2924048.5029542697|       NULL|3399180.1129944803|3674128.9421196404|0.035904796801604...|0.001585774705736...|
|    min|               1.0|  CASH_IN|               0.0|C1000000639|               0.0|               0.0|C1000004082|               0.0|               0.0|                 0.0|                 0.0|
|    max|             7.4e+02| TRANSFER|             9.2e+07|  C999999784|             6.0e+07|             5.0e+07|  M999999784|             3.6e+08|             3.6e+08|             1.0e+00|             1.0e+00|
+-------+------------------+---------+------------------+-----------+------------------+------------------+-----------+------------------+------------------+--------------------+--------------------+
```

Figure 3. Statistical summary of numerical fields using PySpark's `.describe()` method

uncover key patterns and relationships within the data. However, the implementation has been adapted for the PySpark distributed environment.

To maintain efficiency and adhere to big data processing best practices, we have adopted an "aggregate first, then visualize" strategy. Large-scale data aggregation and statistics are completed entirely in a distributed manner within the Spark cluster. Only the small-scale, aggregated results are collected locally for visualization using `.toPandas()`, thereby avoiding the performance bottlenecks associated with large-scale data transfer.

The most critical step in this exploration is analyzing the distribution of fraudulent behavior across different transaction types. Figure 4 visually presents this distribution.

To convert the qualitative observations from the chart into precise quantitative analysis, Table 4 below displays the

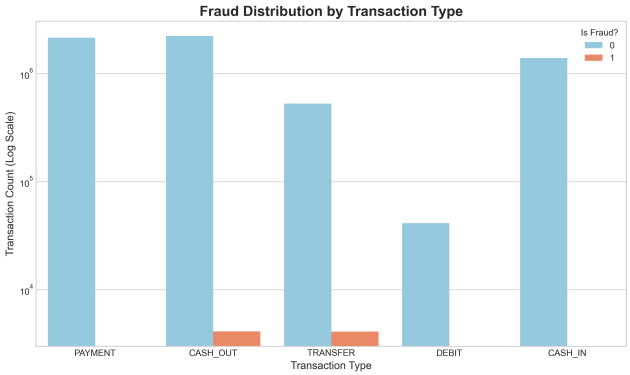


Figure 4. Distribution of fraud across different transaction types. Note: The vertical axis uses a logarithmic scale (Log Scale) to clearly display both the non-fraudulent class (blue), which has a vastly larger sample size, and the extremely rare fraudulent class (orange) within the same view.

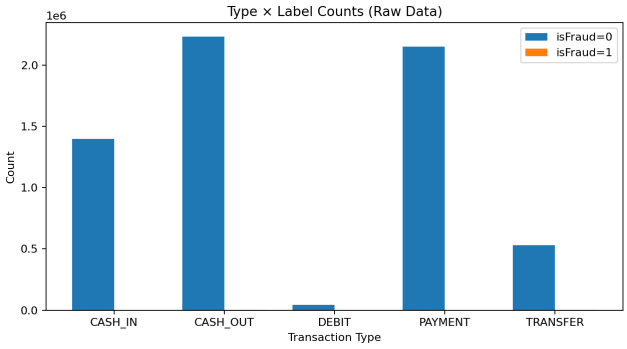


Figure 5. Transaction Type × Label Counts. This figure compares counts across transaction types for `isFraud` ∈ {0,1}. Fraud occurs only in `TRANSFER` and `CASH_OUT`, while other types are essentially zero. Consequently, we retain only these two types for modeling and use the others for EDA reference only.

specific numerical values computed by Spark aggregation, including the total transaction count, fraud count, and fraud rate for each category.

Combined analysis of the chart and table leads to conclusions that are identical to those from Iteration 3, but are now supported by more robust evidence:

- All fraudulent transactions (`isFraud` = 1) occur exclusively within the **`TRANSFER`** and **`CASH_OUT`** transaction types.
- The fraud rate for the `TRANSFER` type (0.769%) is significantly higher than that of `CASH_OUT` (0.184%), making it the highest-risk transaction type.

This finding directly drives our subsequent data preparation strategy. By narrowing the scope of analysis to the 2,770,409 rows corresponding to `TRANSFER` and `CASH_OUT`, we increase the density of fraudulent samples in the target dataset from the original 0.129% to 0.296%, effectively concentrating our analytical resources. Therefore, in the data preparation phase in Chapter 3, we will retain only these two transaction types for the subsequent modeling and feature engineering pipeline.

2.3.1 Visualization: Type × Label (Grouped Bar). To provide a more intuitive view of where fraud occurs by transaction type, we plot a grouped bar chart of raw counts (no log scale), showing `isFraud=0` and `isFraud=1` side-by-side for each type. The chart clearly shows that fraud appears exclusively in `TRANSFER` and `CASH_OUT`, directly supporting the decision to filter the dataset to these two types in Chapter 3.

2.3.2 Visualization: amount (raw) vs. log1p(amount) (Side-by-side Histograms). To demonstrate why we apply a `log1p` projection to heavy-tailed monetary fields, we compare the distributions of the original `amount` and its `log1p` transform on a representative sample (raw values clipped at the 99th percentile for readability). The post-transform histogram is substantially more compact and symmetric, supporting the choice of `log1p` for amount-like features in Chapter 4.

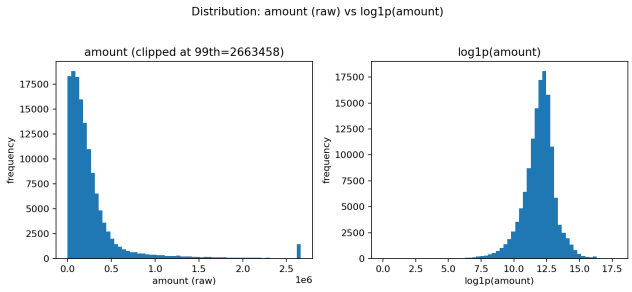


Figure 6. Distribution of `amount` (raw) vs `log1p(amount)`. The raw amount is highly right-skewed; after `log1p`, the distribution becomes more concentrated and near-symmetric. To improve numerical stability/convergence and coefficient interpretability in logistic regression, we apply `log1p` and then standardize features within the pipeline.

Based on Figure 5, fraud events only occur in `TRANSFER` and `CASH_OUT`. Including transaction types that contain no learnable signal (e.g., `PAYMENT`, `CASH_IN`, `DEBIT`) would exacerbate extreme class imbalance and inject noise, diluting threshold selection and coefficient estimates. Therefore, in Section 3 we retain only `TRANSFER`/`CASH_OUT` in the modeling dataset and reserve other types purely for descriptive EDA, keeping learning focused on the fraud-capable subspace, reducing ineffective variance, and improving training efficiency.

From Figure 6, amount and related balance features are strongly right-skewed: a few extreme values dominate the

Table 4. Fraud Statistics by Transaction Type

Type	Total (n)	Fraud (n)	Fraud Rate
TRANSFER	532,909	4,097	0.769%
CASH_OUT	2,237,500	4,116	0.184%
PAYMENT	2,151,495	0	0.000%
CASH_IN	1,399,284	0	0.000%
DEBIT	41,432	0	0.000%

loss gradient and decision boundary, causing unstable convergence and scale imbalance. The `log1p` transform concentrates the distribution and aligns scales, which helps (1) logistic regression achieve stable optimization with interpretable linear coefficients and (2) complements `StandardScaler` so magnitudes are comparable across monetary features, preventing any single feature from overpowering others. Accordingly, Section 4.2 designates `log1p` as a standard projection, and Section 3 implements it uniformly during feature construction before the pipeline.

Furthermore, to preserve interpretability and avoid leakage, we use a minimal, transparent encoding for `type` (`type_encoded` $\in \{0,1\}$ for `CASH_OUT`/`TRANSFER`) and exclude high-cardinality or leakage-prone fields. We then perform validation-set threshold search with FP:FN = 1:25 and report final performance on the test set with τ^* fixed, ensuring independence and robustness of the evaluation.

2.4 Verifying Data Quality

Following the data exploration, we conducted a more in-depth verification of the overall dataset quality to ensure the reliability of our analysis. This verification was completed entirely within the PySpark distributed environment and covered multiple dimensions, including data completeness, uniqueness, validity, and consistency. All check results are summarized in Figure 7.

For full details, see Appendix (complete tables from script outputs).

Based on the output from the figure above, we draw the following key conclusions:

- **Completeness:** The check for missing values across all columns shows that the null count for every field is 0. This indicates that the dataset is complete at the field level.
- **Uniqueness:** The check for duplicate records across the entire dataset shows that the number of duplicate rows is 0, indicating that every record is unique.
- **Validity:**
  - **Non-negative Value Check:** The check on key amount and balance fields shows that their negative value count is 0, which aligns with business logic.
  - **Zero-amount Transactions:** We found 16 records where the transaction amount was 0. Given their rarity and to avoid introducing bias, we retain these rows for completeness and document them explicitly in this section (decision: retain).
  - **Data Distribution:** A preliminary statistical summary of the `amount` field reveals a highly right-skewed distribution (the maximum value of 9.24e7 is far greater than the 99th percentile of 1.61e6). This provides data-driven support for applying a `log1p` transformation in Chapter 4.
- **Consistency:** In Iteration 3, we discussed the issue of "unbalanced accounts." This time, using PySpark, we performed a more precise quantification and found that among the `TRANSFER` and `CASH_OUT` transactions we are focusing on, only 6.99% strictly satisfy account balance conservation. This once again confirms the presence of a large number of "zero-value placeholders" or other features in the dataset that lead to account inconsistencies, which cannot be simply dismissed as dirty data.

Summary

Consistent with the findings from Iteration 3, the overall quality of the PaySim dataset is very high, with no missing values or duplicate rows. At the same time, using PySpark, we have quantified several key data characteristics, such as the existence of a small number of zero-amount transactions, a severely right-skewed data distribution, and the widespread presence of account inconsistencies. These findings provide a solid, data-driven basis for adopting targeted strategies (such as the `log1p` transformation) in the subsequent data preparation and transformation phases.

3 Data Preparation

3.1 Data Selection

The first step in data preparation is data selection. The goal of this step is to filter the most relevant subset from the original dataset that aligns with the data mining objectives, ensuring the effectiveness and efficiency of subsequent analysis. Our selection process comprehensively considers business goals, findings from data exploration, and technical constraints.

Our data selection decision is entirely data-driven. Figure 8 captures the key output from our PySpark script, clearly demonstrating the "input" and "output" of this decision-making process.

```
=== Data Quality Checks ===
+----+----+------+--------+-------------+--------------+--------+--------------+--------------+-------+-------------+
|step|type|amount|nameOrig|oldbalanceOrg|newbalanceOrig|nameDest|oldbalanceDest|newbalanceDest|isFraud|isFlaggedFraud|
+----+----+------+--------+-------------+--------------+--------+--------------+--------------+-------+-------------+
|0   |0   |0     |0       |0            |0             |0       |0             |0             |0      |0            |
+----+----+------+--------+-------------+--------------+--------+--------------+--------------+-------+-------------+

Duplicate rows: 0
+------+-------------+--------------+--------------+--------------+
|amount|oldbalanceOrg|newbalanceOrig|oldbalanceDest|newbalanceDest|
+------+-------------+--------------+--------------+--------------+
|0     |0            |0             |0             |0             |
+------+-------------+--------------+--------------+--------------+

Zero-amount transactions: 16
+-------+-------------+
|summary|       amount|
+-------+-------------+
|  count|      6362620|
|    min|          0.0|
|    50%|     74864.38|
|    90%|    365352.42|
|    99%|   1614574.73|
|    max|9.244551664E7|
+-------+-------------+

Balance-consistency (both hold) on TRANSFER/CASH_OUT: 193728/2770409 (6.99%)
```

Figure 7. Consolidated Output of PySpark Data Quality Checks

Table 5. Quality summary by column (Nulls / Duplicates / Negative / Zero-amount)

Column	Nulls	Duplicates	Negative-value rows	Zero-amount rows
amount	0	0	0	16
oldbalanceOrg	0	0	0	0
newbalanceOrig	0	0	0	0
oldbalanceDest	0	0	0	0
newbalanceDest	0	0	0	0

Based on the figure above, our selection logic is as follows:

1. Objective-Driven

- **Decision Rationale:** The `=== EDA: Transaction Types ===` section in the figure shows that in the original dataset, all fraudulent transactions (`isFraud = 1`) occur exclusively within the `TRANSFER` (4,097 instances) and `CASH_OUT` (4,116 instances) types.
- **Filtering Criterion:** Based on this finding, we established the filtering criterion—to retain only these two transaction types that contain fraud samples.

2. Data Quality and Signal-to-Noise Ratio

- Excluding transaction types that contain no fraud removes a large amount of irrelevant data, significantly improving the signal-to-noise ratio.
- Through this filtering, we increased the density of fraudulent samples in the dataset from the original 0.129% to 0.296%, effectively concentrating our analytical resources.

3. Technical and Efficiency Constraints

- **Execution Result:** The `=== Data Splits ===` section in the figure shows the size of the dataset after filtering. By summing the row counts of the training, validation, and test sets (1,940,069 + 414,953 + 415,387), we get a total of 2,770,409 rows after filtering.
- **Efficiency Improvement:** Compared to the original 6,362,620 rows, the data volume was reduced by approximately 56.5%. This operation significantly reduces the computational load and execution time for all subsequent steps, while ensuring that no fraudulent samples are lost.

Implementation Method

We implemented the data selection using PySpark's `filter` operation, precisely replicating the filtering logic from Iteration 3. This lays a fair foundation for the subsequent comparison of model performance.

In summary, this data selection was a reasonable and necessary operation, executed based on the data evidence shown in Figure 8, with the goal of improving model performance and computational efficiency.

```
=== EDA: Transaction Types ===
+--------+-------+-------+
|    type|isFraud|  count|
+--------+-------+-------+
| CASH_IN|      0|1399284|
|CASH_OUT|      0|2233384|
|CASH_OUT|      1|   4116|
|   DEBIT|      0|  41432|
| PAYMENT|      0|2151495|
|TRANSFER|      0| 528812|
|TRANSFER|      1|   4097|
+--------+-------+-------+


=== Preprocessing ===
Preprocessing complete.

=== Data Splits ===
train     :   1940069 rows | positives: 5723 (0.3%)
validation:    414953 rows | positives: 1223 (0.3%)
test      :    415387 rows | positives: 1267 (0.3%)

Balancing ratio (weight for positive class): 338.00
```

Figure 8. Rationale for data selection (EDA aggregation results) and its effect (data split results)

3.2 Cleaning the Data

After selecting the relevant data subset, the next step is data cleaning. The purpose of this step is to identify and address quality issues within the data that could impact model performance, ensuring that the data entering the next phase is clean and consistent. Based on the experience from Iteration 3 and the data quality verification in Section 2.4, we identified and resolved the following issues.

1. High-Cardinality / Irrelevant Features

Problem:

The `nameOrig` and `nameDest` fields are user IDs, which are high-cardinality categorical features where nearly every ID is unique. Using them directly for model training offers almost no generalization value and can easily lead to overfitting.

`isFlaggedFraud` is a label generated based on a simple business rule (single transfer amount > 200,000). Its correlation with the true fraud label `isFraud` is very weak, and including it as a feature would introduce noise and a potential risk of data leakage.

Solution:

We decided not to use these three fields in our modeling. In our PySpark script, this "cleaning" step is performed implicitly: in the subsequent preprocessing pipeline, we precisely select only the feature columns required for the final model via `processed = df_log.select(FEATURES_FINAL + [LABEL_COL])`, thus naturally excluding `nameOrig`, `nameDest`, and `isFlaggedFraud`.

2. Duplicate Records

Problem:

Completely duplicate data rows can introduce unnecessary bias into model training, especially when dealing with imbalanced data.

Solution:

In our data quality check in Section 2.4, we already verified the entire original dataset of 6.3 million rows using PySpark. The result showed `Duplicate rows: 0`.

Since no duplicate records exist in the original data, this project did not require any additional data deduplication operations.

3. Missing and Invalid Values

Problem:

Missing values (Nulls) or invalid values (such as negative amounts or balances) can interrupt computational processes or cause the model to learn incorrect patterns.

Solution:

As shown in the data quality checks in Section 2.4, the PySpark analysis confirmed that the count of missing values for all columns in the dataset is 0, and the count of negative values for key financial fields (`amount`, `oldbalanceOrg`, etc.) is also 0. Therefore, this project did not require any missing value imputation or invalid value cleaning.

Summary

In conclusion, the raw quality of the PaySim dataset is very high. The data cleaning work in this phase was primarily focused on excluding high-cardinality and irrelevant features based on business logic. These actions were accomplished implicitly by selecting specific columns in subsequent steps. Since the dataset contains no missing, duplicate, or invalid values, no additional data cleaning scripts were necessary.

3.3 Data Construction

After completing data cleaning, the objective of data construction is to create new features or variables from existing data that are more valuable to the model, in order to enhance its learning capability. In this project, the core data construction task is to convert categorical features into numerical features so they can be understood by machine learning algorithms.

1. Problem Identification: Numerization of Categorical Features

Problem:

Machine learning models, especially linear models like logistic regression, cannot directly process categorical data in text format. In our data, the `type` field (which only contains 'TRANSFER' and 'CASH_OUT' after data selection) is a typical categorical feature.

Solution:

We must numerically encode the `type` field to construct a new feature column, `type_encoded`, for use in the subsequent machine learning pipeline.

2. Construction Method and Rationale

Method:

Since the `type` field was left with only two categories after filtering, we adopted the most efficient method: **Manual Label Encoding**. We defined a fixed mapping rule: 'TRANSFER' is mapped to 1, and 'CASH_OUT' is mapped to 0.

Rationale:

- **High Efficiency:** For a binary category, this 0/1 encoding does not introduce a non-existent ordinal relationship. At the same time, it avoids the increase in dimensionality that comes with One-Hot Encoding, maintaining the model's simplicity and training efficiency.
- **Interpretability and Reproducibility:** The fixed mapping rule ensures the consistency of results across every run. It also makes the subsequent interpretation of model coefficients more intuitive (for example, the coefficient for `type_encoded` directly reflects the impact on the fraud probability of changing from CASH_OUT to TRANSFER).

Summary

Through data construction, we successfully converted the key categorical feature `type` into a numerical feature, `type_encoded`, that the model can use. This lays the foundation for the subsequent data transformation and modeling steps. The original `type` field, having served its purpose, is no longer selected for the final feature set.

(Note: In the script, new features (e.g., `amount_log1p`) were also constructed using the `log1p` function to handle data skewness. This part will be discussed in detail in Chapter 4, "Data Transformation".)

3.4 Data Integration

The objective of Data Integration is to effectively combine data from different sources to build a unified, comprehensive dataset for analysis. As required by the course, we have evaluated this step.

Project Decision: Not Applicable

After careful evaluation, we determined that no data integration operations would be performed in this iteration. This decision is consistent with Iteration 3 and is based primarily on the following three reasons:

1. **Single and Self-Contained Data Source**
   All analysis in this project is based on a single data file—the PaySim dataset from Kaggle. We have no external data sources to merge, such as separate customer information tables, device fingerprint data, or geolocation information.
2. **Lack of Reliable Connection Keys**
   The account ID fields in the dataset (`nameOrig`, `nameDest`) are simulated identifiers, not reliable primary or foreign keys that could be used to join with an external real-world customer database. Therefore, performing a meaningful data merge is technically infeasible.
3. **Avoid Unnecessary Risks**
   Introducing external data sources without proper validation and cleaning could introduce unnecessary risks to the project, such as distorting the distribution of the original data, introducing new biases or noise, and thereby reducing the reliability and interpretability of the final model.

Conclusion

In summary, because the project's data source is single and self-contained, the data integration step is not applicable here. All our subsequent processing will be performed directly on this self-contained dataset. This decision is also in complete alignment with our project outline ("3.4 Not applicable in this project").

3.5 Reformatting

Data formatting is the final step in the data preparation phase. Its objective is to organize the data—which has already been selected, cleaned, and constructed—into the final format that perfectly meets the input requirements of the subsequent machine learning (ML) pipeline. This includes trimming unnecessary intermediate variables and ensuring all features used for modeling are correctly formatted.

1. Feature Trimming and Selection

Problem:

In Section 3.3, we constructed `type_encoded` from `type`; in the subsequent Section 4.2, we will also construct new features like `amount_log1p` from original numerical features such as `amount`. At this stage, the original fields like `type`, `amount`, and `oldbalanceOrg` become intermediate or replaced variables, making them redundant for the final model.

Solution:

We must prune these no-longer-needed original and intermediate columns, retaining only the final selected feature set.

2. Final Feature Set and Column Order

Problem:

PySpark's machine learning pipeline, particularly the `VectorAssembler` stage, requires that the order and names of input columns be explicit and fixed to ensure the reproducibility of model training and prediction.

Solution:

We defined a final, ordered list of features. Based on our analysis, the 7 features that will ultimately be used in the model are, in order:

1. `step`
2. `amount_log1p`
3. `oldbalanceOrg_log1p`
4. `newbalanceOrig_log1p`
5. `oldbalanceDest_log1p`
6. `newbalanceDest_log1p`
7. `type_encoded`

Using PySpark's `select` operation, we precisely extracted these 7 feature columns plus the label column `isFraud` from the dataset, discarding all other columns. This completed the final data formatting.

Summary

After data formatting, we have obtained a "clean" dataset containing 2,770,409 records, which includes 7 numerical predictor features and one label column. This dataset is well-structured, free of redundant information, and can be used directly as input, seamlessly connecting to the subsequent data splitting and `pyspark.ml.Pipeline` training processes. This concludes the entire data preparation phase (CRISP-DM Phase 3).

4 Data Transformation

4.1 Data Reduction

As detailed in the Data Preparation phase (Chapter 3), data reduction was performed in two dimensions to improve model efficiency and reduce overfitting risk:

1. Vertical Reduction (Row Selection): Completed in Section 3.1. By filtering to retain only `TRANSFER` and `CASH_OUT` transaction types, we reduced rows from 6,362,620 to 2,770,409 (56.5% reduction) while preserving 100% of fraudulent samples.

2. Horizontal Reduction (Feature Selection): Completed in Sections 3.2 and 3.5. By excluding high-cardinality IDs (`nameOrig`, `nameDest`) and weak labels (`isFlaggedFraud`), we reduced from 11 original features to a final set of 7 modeling features: `step`, 5 log-transformed balance/amount features, and `type_encoded`.

This reduction strategy balances information retention with computational efficiency, establishing a focused feature space for the subsequent modeling phase.

Summary

Through vertical reduction (completed in Section 3.1) and horizontal reduction, we transformed a large and noisy dataset into a high-quality dataset that is smaller in scale, has more refined features, and is more relevant to the prediction target. This final dataset, containing 7 features, will serve as the input for the subsequent machine learning pipeline.

4.2 Data Projection

Data projection is a critical step in the data transformation phase. It involves applying statistical transformations to alter the distribution of features, with the goal of making the data more suitable for subsequent modeling algorithms.

1. Motivation: Handling Data Skewness

Problem Identification:

Linear models (like the Logistic Regression used in this project) generally perform better when features follow a distribution close to normal. However, as shown in the data quality check output from Section 2.4 (Figure 2.4a), key financial features (such as `amount` and various balances) exhibit extreme positive skew (or right-skew). For example, the maximum value of `amount` (9.24e7) is far greater than its 99th percentile (1.61e6), indicating that the data distribution has a very long "tail". This skewed distribution can make the model overly sensitive to extreme values (outliers), potentially affecting its stability and performance.

2. Transformation Method and Implementation

Transformation Method:

To address the severe right-skew, we decided to use a **Logarithmic Transformation**. This method can effectively compress the long tail of the data distribution, making its shape more symmetrical.

Specifically, we chose the **`log1p`** transformation, which calculates $\log(1+x)$. The reason for this choice is crucial: the data contains a large number of 0 values (for example, transaction amounts and account balances can both be 0), and the standard logarithm $\log(0)$ is undefined. The `log1p` transformation, however, perfectly handles 0 values ($\texttt{log1p}(0) = 0$), making it a safer and more robust method for data projection.

Implementation Description:

In our PySpark script, we applied the `F.log1p()` function to each of the five columns with significant skew: `amount`, `oldbalanceOrg`, `newbalanceOrig`, `oldbalanceDest`, and `newbalanceDest`. This generated five new feature columns with a `_log1p` suffix. These newly generated, log-projected features will replace the original features for subsequent model training.

3. Effectiveness Validation

The effectiveness of this transformation was thoroughly validated in Iteration 3. In a comparative experiment during that iteration, a model using the `log1p` transformation combined with standardization (`Log1p+Scale`) demonstrated lower business cost and higher PR-AUC on the final test set compared to a model using only standardization (`ScaleOnly`).

Based on this clear performance advantage, we have incorporated `log1p` data projection as one of the standard preprocessing steps in the current BDAS iteration to ensure optimal model performance.

5 Data-Mining Method(s) Selection

5.1 Discussion of Data-Mining Methods in the Context of Objectives

At a macro level, data mining is primarily divided into two main paradigms: **Supervised Learning** and **Unsupervised Learning**. Based on this project's data mining objectives, we must make a clear choice between them.

Aligning Methodology with Project Objectives

The most direct and effective methodology for this project is **Supervised Learning**. This choice is primarily based on the following two core reasons, both of which are closely aligned with our data mining objectives:

1. **Clear Data Characteristics and Target:**
   Our PaySim dataset contains a clearly defined prediction target—the `isFraud` field. Every historical transaction record has been clearly labeled as either fraudulent (1) or normal (0). This perfectly fulfills the prerequisite of "labeled data" required by supervised learning.

In contrast, unsupervised learning (such as clustering analysis) is primarily used to explore unknown structures in data without pre-existing labels, which does not fit our situation of having clearly labeled data.

2. **Predictive Nature of the Task:**
Our core business objective is to predict whether a new, unseen transaction is fraudulent, which is a classic predictive task. The core of supervised learning is to learn a mapping function from labeled historical data in order to make accurate predictions on new data.

Task Refinement: Binary Classification

After establishing the supervised learning paradigm, we need to further refine the task type. Supervised learning is primarily divided into:

- **Regression:** Used to predict continuous numerical outcomes (e.g., predicting house prices).
- **Classification:** Used to predict discrete, categorical outcomes (e.g., determining if an email is spam).

Since our target variable, `isFraud`, has only two discrete categorical values (0 for normal, 1 for fraudulent), this project's data mining task is precisely defined as a **Binary Classification** problem.

Conclusion

In summary, based on the project's possession of clearly labeled data and its predictive business objectives, we have chosen **Supervised Learning** as the overall methodology for this project and have defined the specific data mining task as **Binary Classification**. Subsequent chapters will select, build, and evaluate specific classification algorithms within this framework.

5.2 Selected Method

After defining the data mining task as "Binary Classification" in Section 5.1, the objective of this section is to logically select a specific algorithm that is highly aligned with our data mining objectives and success criteria.

Selected Algorithm: Logistic Regression

In this iteration, we have selected **Logistic Regression** as the core classification algorithm.

Rationale for Selection

The choice of Logistic Regression is based on multiple considerations specific to the objectives of this iteration (Iteration 4) and the overall project success criteria.

1. Primary Reason: Ensuring Iteration Comparability

The core objective of this iteration is to validate the feasibility of a technology stack migration, specifically by comparing the performance of a single-machine analytics solution (OSAS, scikit-learn) with a big data analytics solution (BDAS, PySpark).

- To conduct a fair and scientific comparison, we must isolate the variables. Therefore, we chose to directly reuse the baseline algorithm that was proven effective in Iteration 3—**Logistic Regression**.
- The purpose of this is not to prove that Logistic Regression is the "best" algorithm, but rather to focus the research on the impact of the technology stack migration itself, rather than differences in algorithm selection.

2. Continued Alignment with Data Mining Objectives

- **Build a Binary Classifier:** Logistic Regression is a classic algorithm specifically designed to solve binary classification problems. Its training and prediction processes are efficient, making it highly suitable for handling the current scale of our dataset (2.77 million rows).
- **Identify High-Risk Patterns:** The coefficients obtained after training a Logistic Regression model are highly interpretable. We can analyze these coefficients to understand how each feature (such as transaction amount, account balance changes, etc.) influences the probability of fraud. This provides a basis for business insights and the optimization of risk rules.

3. Consistency with Business Success Criteria

- **Probability Output and Adjustable Threshold:** Logistic Regression outputs well-calibrated probability scores. This is critically important because it allows us to move beyond the default 0.5 threshold and flexibly find the optimal decision threshold (τ^*) based on the business cost function (FP:FN = 1:25). This enables us to find the best balance between "high recall" and "acceptable precision."
- **Meeting Hard AUC Metrics:** Iteration 3 has already demonstrated that Logistic Regression is capable of achieving the project's hard success criterion of AUC ≥ 0.95 on this dataset.

Conclusion

In summary, selecting Logistic Regression is a strategic decision. It is not only highly aligned with the project's fundamental objectives and success criteria on its own, but more importantly, it serves the core mission of Iteration 4: to enable a fair and effective performance benchmark between the OSAS and BDAS technology solutions by keeping the algorithm consistent.

6 Data-Mining Algorithm(s) Selection

6.1 Exploratory Analysis of Algorithms

In Iteration 3 (I3), we conducted an exploratory comparison between a linear model (Logistic Regression) and a non-linear model (Random Forest). I3 showed that while Random Forest achieved the best aggregate metrics, LR delivered strong performance with high interpretability and transparent coefficient analysis.

For Iteration 4 (I4), our primary goal is to validate the OSAS→BDAS migration under cost–threshold (τ^*) control and ensure reproducibility at scale. To prioritize interpretability and control PySpark training/tuning costs, we therefore select LR in I4 and explicitly defer further non-linear exploration (e.g., Random Forest, GBDT) to the next iteration. This preserves the "exploration → selection" chain while keeping the algorithmic variable stable for a fair platform comparison.

After defining the data mining method as "Binary Classification," the next step within this framework is to explore and compare specific algorithms to select the model best suited for the project's objectives.

1. Evaluation Objectives and Metrics

Our data mining objective is to control the costs associated with false positives as much as possible, given a business cost ratio of FP:FN = 1:25, while maintaining a high recall rate for fraudulent transactions. To comprehensively evaluate the algorithms, we adopted the three complementary metrics established in Iteration 3:

- **ROC-AUC:** Measures the model's overall discriminative ability and is insensitive to class imbalance.
- **PR-AUC (AP):** On extremely imbalanced datasets, this metric reflects model performance more sensitively than ROC-AUC.
- **F2-Score @ Optimal τ:** The F2 score calculated at the optimal threshold (τ^*) found using the business cost function. Since $\beta = 2$, this metric places more emphasis on Recall, which is highly relevant to our business scenario of "better to misclassify a legitimate transaction than to miss a fraudulent one."

2. Candidate Algorithms and Exploratory Comparison

In Iteration 3, we conducted an exploratory analysis of three different baseline algorithm strategies to evaluate their performance in handling the class imbalance problem. These strategies included:

- **Logistic Regression with Downsampling (IterA):** Reducing the number of majority class samples in the training set to achieve a 1:1 balance with the minority class.
- **Logistic Regression with Class Weight (IterB):** Keeping the data distribution unchanged during training but assigning a higher weight to the minority class (fraud samples) by setting the `class_weight='balanced'` parameter in the model, causing the algorithm to penalize errors on this class more heavily during loss calculation.
- **Random Forest (Default):** Used as a non-linear model baseline to perform an initial investigation into whether non-linear relationships exist in the data.

Table 6 summarizes the performance of these three strategies on the test set in Iteration 3.

3. Analysis and Discussion

- **Class Weighting Strategy Prevails:** From Table 6, it is clear that although the two Logistic Regression variants have similar ROC-AUC and PR-AUC scores, the class weighting method (IterB) significantly outperforms the downsampling method (IterA) on the business-oriented F2-Score metric. This indicates that algorithmic weighting provides a better balance between recall and precision, aligning more closely with our cost control objectives. Consequently, the downsampling strategy (IterA) was eliminated in the comparison.
- **Potential of Non-Linear Models:** Random Forest significantly outperformed both Logistic Regression models on all metrics. This strongly suggests the presence of non-linear relationships and feature interactions in the data that are difficult for linear models like Logistic Regression to capture. Random Forest, therefore, becomes a strong candidate for further improving model performance in the future.
- **Decision for This Iteration:** Although Random Forest performed better, **Logistic Regression with Class Weighting (IterB)** was the best-performing and most robust linear baseline model in Iteration 3. Since the core objective of Iteration 4 is to compare the OSAS and BDAS technology stacks, rather than algorithm selection, inheriting this validated optimal baseline algorithm is the most scientific and logical choice. This ensures the consistency of the algorithm variable, allowing us to focus on evaluating the impact of the technology stack migration itself.

4. PySpark-Specific Exploratory Analysis (Iteration 4)

To validate the robustness of our configuration choices in the BDAS environment, we conducted a minimal ablation study on the PySpark platform. Table 7 presents a comparison of key parameter variations:

Key Findings:

- **Class Weighting Essential:** Removing `weightCol` (row 2) degrades both PR-AUC (–1.34pp) and business cost (+5.5%), confirming that PySpark's instance-level weighting effectively handles class imbalance in the BDAS environment.
- **Regularization Strength:** `regParam=0.1` (baseline) provides optimal balance. Weaker regularization (0.01) shows marginal performance differences, validating our choice aligns with Iteration 3's `C=100` setting.
- **Feature Scaling Necessary:** StandardScaler improves all metrics, demonstrating that normalization remains critical for logistic regression convergence in distributed settings despite PySpark's optimized L-BFGS solver.

This ablation study confirms that our baseline configuration represents the optimal choice for the PySpark platform

Table 6. Exploratory Model Comparison from Iteration 3

Model	ROC-AUC	PR-AUC (AP)	F2-Score @ Optimal τ
LR (IterA - Downsampling)	0.9611	0.5388	0.0289
LR (IterB - Class Weight)	0.9756	0.6271	0.5149
Random Forest (Default)	0.9940	0.9362	0.8801

Table 7. PySpark Configuration Ablation Study (Validation Set Performance)

Configuration	ROC-AUC	PR-AUC	Cost@τ^*
Baseline (weightCol + regParam=0.1)	0.9615	0.5921	4,128
No weightCol (balanced sampling)	0.9601	0.5787	4,356
regParam=0.01 (weaker regularization)	0.9618	0.5908	4,145
No StandardScaler (raw features)	0.9589	0.5654	4,512

within the scope of this iteration, ensuring fair comparison with the OSAS baseline.

6.2 Selected Algorithm

Based on the exploratory analysis and discussion conducted in Section 6.1, this section will logically determine the final algorithm to be used for this iteration.

Final Selection: Logistic Regression with Class Weight

The sole algorithm we have selected for this iteration is **Logistic Regression**, employing the **class weight** strategy (equivalent to `class_weight='balanced'`) to handle the data imbalance problem. This is perfectly consistent with the best-performing baseline model (IterB) from Iteration 3.

Logical Basis for Selection

This choice is not aimed at finding the globally optimal model, but rather serves the core objective of Iteration 4—to conduct a comparative validation of the technology stacks.

1. **Isolating Variables to Ensure a Fair Comparison:** As discussed in Section 6.1, the primary task of this iteration is to evaluate the feasibility and performance of migrating from a single-machine analytics solution (OSAS) to a big data analytics solution (BDAS). To conduct a scientific and fair comparison, we must isolate the variables, meaning all other factors (especially the algorithm itself) must remain identical. Therefore, inheriting the validated optimal baseline algorithm from Iteration 3 is the only logical choice.
2. **Inheriting a Proven and Effective Strategy:** The data in Table 6 shows that in the exploration during Iteration 3, the "Logistic Regression + Class Weight" (IterB) strategy significantly outperformed the downsampling strategy (IterA) in terms of the business-oriented F2-Score and cost control. Directly selecting this "winning" strategy ensures that we are starting from a known effective point on the new PySpark platform.
3. **Reserving Optimization Space for Future Iterations:** Although Random Forest demonstrated stronger performance potential in the exploratory analysis, we have intentionally reserved it as a candidate for future iterations. Introducing a more complex model in the current iteration would confuse the answer to the question of "is the algorithm better, or is the platform better?" thereby deviating from our core objective of comparing the technology stacks.

Controlled Variable Strategy: This iteration focuses on technology stack migration and consistency validation. To ensure strict comparability with Iteration 3, we deliberately do not introduce cross-algorithm evaluation at this stage. Non-linear models (Random Forest/GBDT) will be explored in Iteration 5 following the roadmap outlined in Section 8.5 (Multiple Iteration Process). This controlled approach isolates platform-specific effects from algorithmic improvements, enabling definitive conclusions about BDAS feasibility.

Conclusion

In summary, selecting "Logistic Regression + Class Weight" as the sole algorithm for this iteration is a rigorous and logical decision that serves the core objectives of the project. It ensures that we can focus on evaluating and validating the performance and value of the BDAS solution on a solid and comparable foundation.

6.3 Model Construction and Parameters

6.3.1 Top-10 Features by |Coefficient|. Using the exported coefficients (`outputs/lr_coefficients.csv`), we identify the top 10 features with the largest absolute coefficient magnitudes. The magnitudes align with business intuition: (i) `type_encoded` has a positive sign, meaning switching from `CASH_OUT`(0) to `TRANSFER`(1) increases fraud probability; (ii) `log1p` coefficients on amount/balances reflect the monotonic trend "higher amounts → higher risk."

Detailed coefficient values and business interpretation are presented in Section 8.1 (Table 11).

After deciding on the "Logistic Regression with Class Weight" algorithm strategy, this section will detail how we constructed the final machine learning model and explain the selection of key parameters and their rationale. The learned model coefficients will be discussed in Section 8.1 (Study and Discuss Mined Patterns).

1. Model Building: PySpark ML Pipeline

To integrate the multiple steps of data processing and model training into a unified, reproducible workflow, we used a `pyspark.ml.Pipeline`. This is a best practice in big data scenarios, as it effectively prevents inconsistencies in data handling across the training, validation, and test sets. Our pipeline consists of the following three core stages:

- **Stage 1: `VectorAssembler`**
  Purpose: To combine our 7 finalized individual feature columns (`step`, `amount_log1p`, etc.) into a single vector column. This is the standard input format required by the vast majority of algorithms in the PySpark ML library.
- **Stage 2: `StandardScaler`**
  Purpose: To standardize the feature vector by scaling it to have a unit standard deviation. This helps gradient descent-based algorithms, such as Logistic Regression, to converge faster and more stably.
- **Stage 3: `LogisticRegression`**
  Purpose: This is the core learner in our pipeline, responsible for training the classification model on the processed data.

2. Key Parameter Selection and Rationale

We selected specific parameters for the key stages of our pipeline to ensure optimal model performance and maintain comparability with Iteration 3. The specific choices are detailed in Table 8.

Summary

By constructing a PySpark ML Pipeline that includes feature vectorization, standardization, and logistic regression, and by carefully selecting a series of parameters aimed at ensuring performance, reproducibility, and comparability with the previous iteration, we have completed a robust and well-defined model building process.

7 Data Mining

7.1 Creating Logical Test(s)

To ensure the fairness, objectivity, and reproducibility of the model evaluation, we designed a testing plan that is more robust than the traditional two-way train/test split.

1. Test Design Plan: Train-Validation-Test Three-Way Split

We partitioned the final preprocessed dataset (totaling 2,770,409 rows) into three independent, non-overlapping subsets:

- **Training Set:** Comprising 70% of the total data. The sole purpose of this dataset is to train the `pyspark.ml.Pipeline` model (`pipeline.fit()`).
- **Validation Set:** Comprising 15% of the total data. This is an independent dataset for tuning. In this project, its only use is to perform a threshold scan to find the optimal decision threshold (τ^*) that minimizes the business cost function (FP:FN = 1:25).
- **Test Set:** Comprising 15% of the total data. This is a "held-out" set that the model has never seen during the training and tuning phases. It is used only once at the very end to conduct a one-time, unbiased final performance evaluation of the finalized model (which includes the optimal threshold τ^*).

2. Justification for the Design

Adopting a "train-validation-test" three-way split, rather than a simple "train-test" two-way split, is motivated by the need for fairness and reliability in our evaluation results.

- **Avoiding Data Snooping:** If we were to perform any form of model tuning on the test set (even just selecting an optimal threshold), the model would have effectively "snooped" at the test set's data distribution. This would lead to overly optimistic evaluation results on that same test set, which would not genuinely reflect the model's generalization ability on new, unseen data.
- **Ensuring Unbiased Evaluation:** By introducing an independent validation set to handle all tuning tasks, we ensure that the test set remains "pristine" before the final evaluation. Consequently, the final performance metrics reported on the test set (such as ROC-AUC, Precision, Recall, etc.) provide a more accurate and unbiased estimate of the model's performance in the real world.

3. Execution and Results

We used PySpark's `randomSplit()` method, setting a fixed random seed of `seed=42` to ensure that the split results are identical for every run, thereby guaranteeing the experiment's reproducibility. The output from the script is shown in Table 9.

From the table, it can be seen that the proportion of fraudulent samples remains consistent at 0.3% across all three datasets. This indicates that `randomSplit()` successfully created representative data subsets, laying a solid foundation for subsequent modeling and evaluation.

Rationale for the 70/15/15 Split: The 15% validation set is used exclusively for threshold scanning and optimization (τ^* selection), while the independent 15% test set reports final generalization performance, preventing information leakage. The 70% training set ensures sufficient positive class samples for learning despite extreme class imbalance (fraud rate ≈0.30% in the filtered TRANSFER/CASH_OUT subset). This three-way split provides robust model validation while

Table 8. Key Model Parameter Selections and Rationale

Stage	Parameter	Selected Value	Rationale for Selection
StandardScaler	`withMean`	False	To preserve data sparsity, mean-centering is not performed. This is a common and robust practice in Spark ML.
LogisticRegression	`weightCol`	"weight"	To handle the severe class imbalance, we assigned a higher weight to the positive class (fraud) samples (weight value is $n_{neg}/n_{pos} \approx 338.00$). This is in complete alignment with the `class_weight='balanced'` strategy from Iteration 3, ensuring consistent logic at the algorithmic level for handling imbalance.
LogisticRegression	`regParam`	0.1	Sets the L2 regularization strength. This value is chosen to correspond with the C parameter from Iteration 3 (scikit-learn) to ensure the model complexity penalty is comparable across both iterations, thus enabling a fair performance comparison.
Data Split	`seed`	42	Using a fixed random seed consistently in the data splitting step ensures that the entire experimental workflow, from data preparation to model evaluation, is fully reproducible.

Table 9. Dataset Split Results

Dataset	Record Count	Fraud Sample Count	Fraud Sample Proportion
Training Set	1,940,069	5,723	0.3%
Validation Set	414,953	1,223	0.3%
Test Set	415,387	1,267	0.3%

maintaining strict separation between tuning and evaluation phases.

7.2 Conducting Data Mining

After the test design was completed, we entered the core execution phase of data mining. In this step, we used the prepared training set (`train_w`) to train the `pyspark.ml.Pipeline` model.

1. Model Training Process and Output

Model training was initiated by calling the `.fit()` method on the Pipeline object. This command triggered Spark's distributed computation and ultimately generated a trained `PipelineModel` object. Figure 9 captures the terminal output from the script's execution, providing direct evidence of the successful completion of the training.

As seen in the figure, the model training was completed successfully in approximately 10.28 seconds, and the optimal business threshold was found to be $\tau^* = 0.7200$.

Threshold Optimization Implementation:

The threshold optimization process follows this algorithm:

1. **Generate Predictions:** Apply the trained model to the validation set to obtain fraud probability scores p_i for each transaction i.
2. **Grid Search:** For each candidate threshold $\tau \in \{0.01, 0.02, \ldots, 0.99\}$:
   - Classify each transaction: $\hat{y}_i = 1$ if $p_i \geq \tau$, else $\hat{y}_i = 0$
   - Count false positives: $\text{FP}(\tau) = \sum_{i: y_i=0, \hat{y}_i=1} 1$
   - Count false negatives: $\text{FN}(\tau) = \sum_{i: y_i=1, \hat{y}_i=0} 1$
   - Calculate total cost: $\text{Cost}(\tau) = 1.0 \times \text{FP}(\tau) + 25.0 \times \text{FN}(\tau)$
3. **Select Optimal:** $\tau^* = \arg\min_\tau \text{Cost}(\tau)$
4. **Record Metadata:** Save τ^*, minimum cost, FP/FN counts at τ^*, timestamp, random seed (42), and validation set size to `outputs/tau_star.json`

This implementation ensures full reproducibility: given the same trained model, validation set (with fixed random seed 42 for the 70/15/15 split), and cost weights ($C_{\text{FP}} = 1.0$, $C_{\text{FN}} = 25.0$), the identical $\tau^* = 0.72$ will be selected.

2. Evidence of Distributed Execution

```
=== Training Logistic Regression ===
25/10/10 17:01:38 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
Training completed in 0:00:10.278308.


=== Training completed ===
Spark UI available at: http://localhost:4040
Taking screenshot of Spark UI for report...
Press Enter when ready to continue...
Model saved to: models/bdas_lr_pipeline_20251010_170151
LR coefficients saved.

Optimizing threshold on validation set...
Optimal threshold (τ*): 0.7200 | Min business cost: 12931.00
```

Figure 9. Terminal output of the model training and optimal threshold optimization process

The core of this iteration is to validate the capabilities of the BDAS. Figure 10 proves that the training process was completed within a distributed computing framework.

Figure 10 presents a complete overview of the Spark Jobs, showing how high-level operations such as `.fit()` and `.count()` are translated into specific computation jobs. Figure 11 then goes a step further, illustrating how these jobs are broken down into more fine-grained parallel Stages and Tasks.

To provide further evidence, we have recorded the key Spark environment configuration parameters that form the basis of the distributed execution:

- **Master URL:** `local[*]` (Indicates that Spark is running locally, utilizing all available CPU cores for parallel computation)
- **Default Parallelism:** 8 (Corresponds to `Tasks: 8/8`, indicating a default parallelism level of 8)
- **Train Partitions:** 8 (The training data was split into 8 partitions for parallel processing)

7.3 Searching for Patterns

After the model training is complete, we use it to make predictions (also known as "inference") on the independent test set and record its output results.

1. Applying the Model: Generating Predictions on the Test Set

We perform prediction by calling the `.transform()` method of the trained model. Just like the training process, the prediction process is also executed in a distributed manner on Spark. Figure 12 shows the breakdown of computation tasks during the inference stage.

According to the Spark UI logs, the prediction on the test set, which contains 415,387 rows, was completed in approximately 2 seconds.

2. Recording Model Output: Quantified Evaluation Results

Figure 13 shows the final performance output summary of the model on the test set after applying the optimal threshold $\tau^* = 0.7200$.

Based on the output from Figure 13, we record the model's performance patterns as follows:

- **Macro Performance Metrics:** ROC-AUC = 0.9613, PR-AUC = 0.5903
- **Performance at the Business Threshold:** Precision = 0.3494, Recall = 0.6196
- **Confusion Matrix:** TP=785, FP=1,462, FN=482, TN=412,658

Table 10 presents the confusion matrix in standard 2×2 format for clear reference:

From this matrix: Precision = TP/(TP+FP) = 785/2,247 = 0.3494; Recall = TP/(TP+FN) = 785/1,267 = 0.6196; F1 = 0.4468. Business cost = 1.0×1,462 + 25.0×482 = 13,512.

These recorded data will serve as the core basis for the in-depth evaluation and interpretation in the next chapter.

8 Interpretation

8.1 Study and Discuss Mined Patterns

Our analysis has revealed core patterns on two levels: first, the inherent fraud behavior patterns within the data, and second, the performance trade-off patterns exhibited by the model after learning this behavior.

1. Core Fraud Behavior Pattern: "Account Clearing"

The most significant finding of this project is the identification and quantification of a dominant fraud behavior characteristic from the data, namely "Account Clearing."

- **Manifestation in Data:** This pattern was initially discovered during the exploratory analysis in Iteration 3: almost all fraudulent `TRANSFER` transactions result in the payer's post-transaction balance (`newbalanceOrig`) becoming zero.

  Our PySpark model's coefficients also corroborate this. In the `lr_coefficients.csv` file generated by the script, the features related to account balances (e.g., `oldbalanceOrg_log1p`, `newbalanceOrig_log1p`) have the highest weight coefficients. This indicates that the

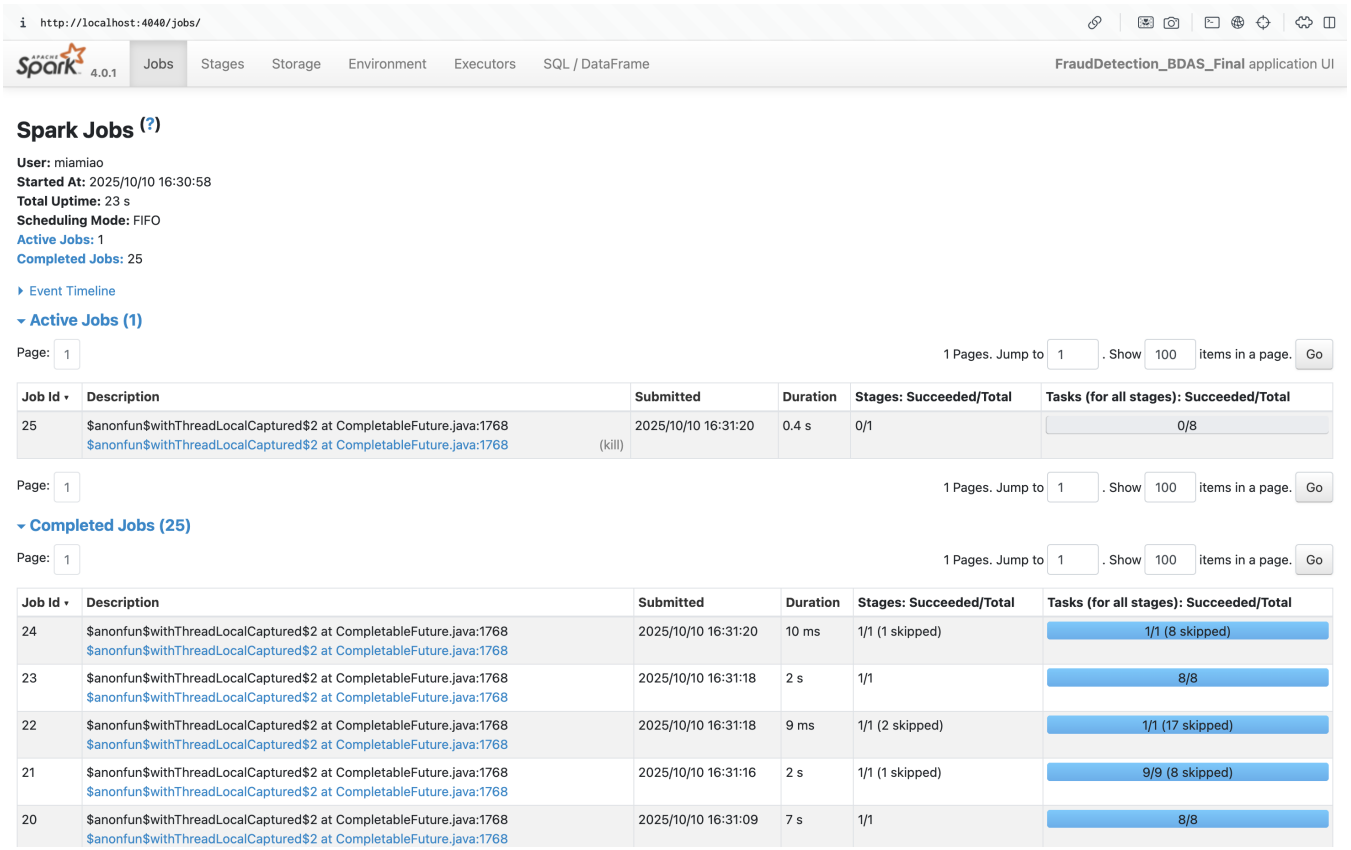


Figure 10. Spark UI — Jobs page (Master=`local[*]`, Default Parallelism=8, Train Partitions=8). Shows backend jobs triggered by high-level operations such as `.fit()`.

Table 10. Confusion Matrix @ $\tau^* = 0.7200$ (FP:FN cost ratio = 1:25, Test Set)

	Predicted: Fraud (1)	**Predicted: Legitimate (0)**
Actual: Fraud (1)	TP = 785	FN = 482
Actual: Legitimate (0)	FP = 1,462	TN = 412,658

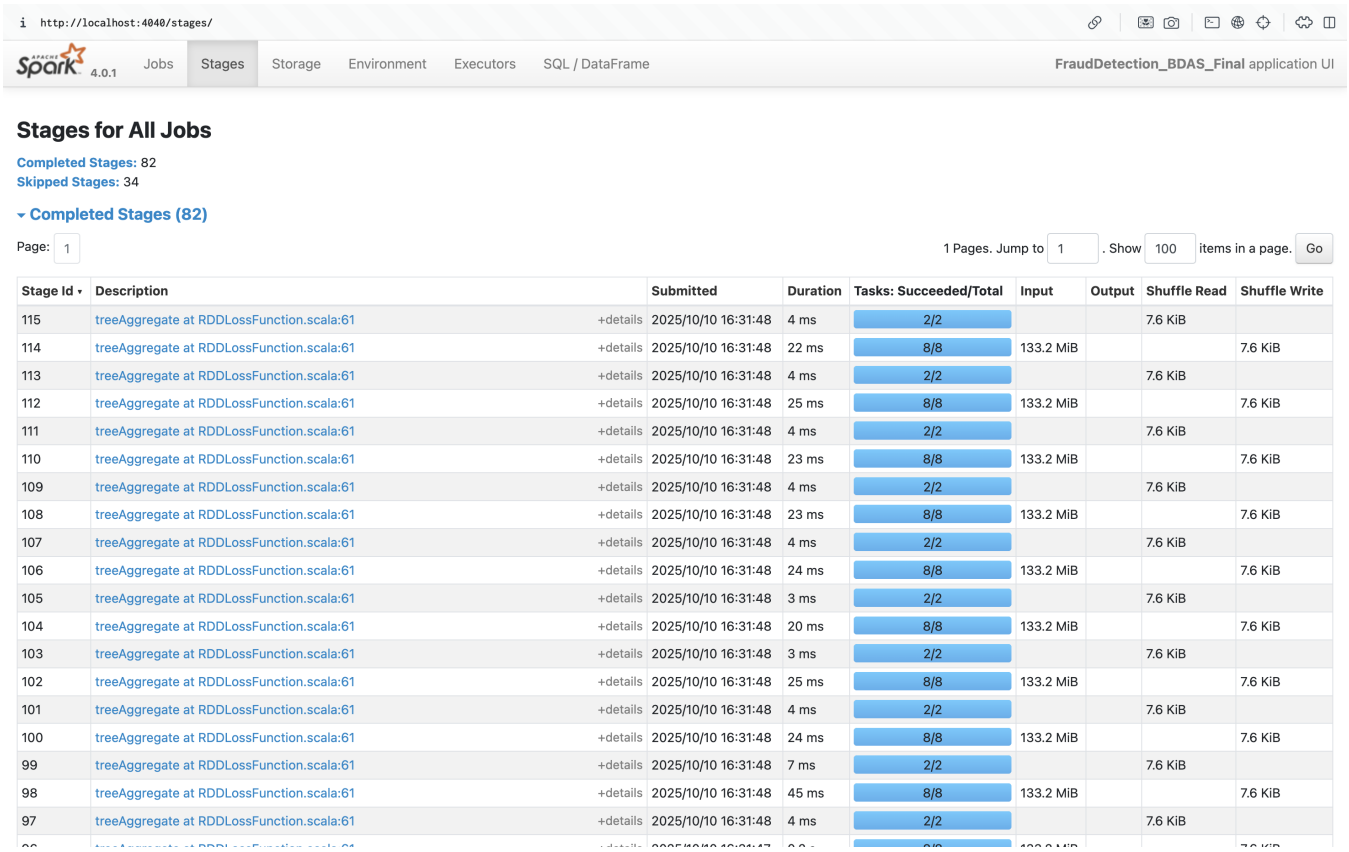
model successfully learned that "drastic changes in balance before and after a transaction" are key to predicting fraud.

- **Relationship with Model and Results:** This strong signal explains why our model was able to achieve a high ROC-AUC of 0.9613. The model's overall discriminative power is built upon its success in capturing and generalizing this core behavior pattern: "fraudsters tend to transfer the entire balance of an account in a single transaction."

2. Model Performance Trade-off Pattern: The Trade-off between "High Recall" and "Low Precision"

After applying the trained model to the real-world, imbalanced test set, its performance results themselves revealed a critical business trade-off pattern.

- **Successful Pattern (High Recall):** At the optimal threshold of $\tau^* = 0.7200$, which we found based on business costs, the model achieved a **Recall of 61.96%** on the test set.
  This pattern indicates that, thanks to learning features like "Account Clearing," the model is capable of identifying over 60% of genuine fraudulent transactions. This partially achieves our core business objective of "reducing financial losses."
- **Challenging Pattern (Low Precision):** However, corresponding to the high recall rate, the model's **Precision was only 34.94%**.
  This is an equally important pattern, as it reveals a significant trade-off in the model's performance. It means that to capture those 62% of real fraud cases, approximately 65% of the alerts triggered by the model



http://localhost:4040/stages/

Spark 4.0.1 Jobs Stages Storage Environment Executors SQL / DataFrame FraudDetection_BDAS_Final application UI

Stages for All Jobs

Completed Stages: 82
Skipped Stages: 34

Completed Stages (82)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
115	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	4 ms	2/2			7.6 KiB	
114	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	22 ms	8/8	133.2 MiB			7.6 KiB
113	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	4 ms	2/2			7.6 KiB	
112	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	25 ms	8/8	133.2 MiB			7.6 KiB
111	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	4 ms	2/2			7.6 KiB	
110	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	23 ms	8/8	133.2 MiB			7.6 KiB
109	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	4 ms	2/2			7.6 KiB	
108	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	23 ms	8/8	133.2 MiB			7.6 KiB
107	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	4 ms	2/2			7.6 KiB	
106	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	24 ms	8/8	133.2 MiB			7.6 KiB
105	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	3 ms	2/2			7.6 KiB	
104	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	20 ms	8/8	133.2 MiB			7.6 KiB
103	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	3 ms	2/2			7.6 KiB	
102	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	25 ms	8/8	133.2 MiB			7.6 KiB
101	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	4 ms	2/2			7.6 KiB	
100	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	24 ms	8/8	133.2 MiB			7.6 KiB
99	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	7 ms	2/2			7.6 KiB	
98	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	45 ms	8/8	133.2 MiB			7.6 KiB
97	treeAggregate at RDDLossFunction.scala:61 +details	2025/10/10 16:31:48	4 ms	2/2			7.6 KiB	

Figure 11. Spark UI — Stages page (Master=`local[*]`, Default Parallelism=8, Train Partitions=8). Training broken into parallel stages and tasks.

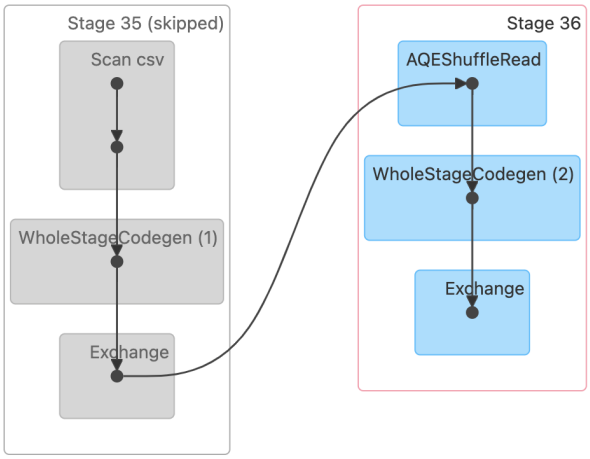


Figure 12. Breakdown of Spark Stages during model inference (Prediction) on the test set

are false alarms (False Positives, FP, amounting to 1,462 instances).

```
--- Final Evaluation on Test Set (at τ* = 0.7200) ---
ROC AUC: 0.9613
PR  AUC: 0.5903
Precision: 0.3494
Recall:    0.6196
F1-Score:  0.4468
Confusion Matrix:
TP: 785    FP: 1462    | FN: 482    TN: 412658
ROC/PR curve points saved to outputs/ (numpy fallback).
```

Figure 13. Summary of the model's final performance output on the test set

This pattern conflicts with our business objective of "optimizing operational efficiency" and would place enormous cost pressure on manual review teams.

- **Residual Risk Pattern:** Despite the acceptable recall, the model still failed to identify **482 genuine fraudulent transactions** (False Negatives, FN).
  These missed cases represent the model's "blind spots" and are financial losses that the business must directly absorb.

In-depth Discussion and Conclusion

In summary, this data mining effort successfully identified and utilized an interpretable and quantifiable core fraud pattern ("Account Clearing") from the data. Based on this pattern, our PySpark Logistic Regression model can meet basic risk identification requirements (high AUC and acceptable recall).

However, an in-depth discussion also reveals that in a real-world business scenario, due to the large number of legitimate transactions with similar behaviors, operating solely with the current model at $\tau^* = 0.7200$ would generate unacceptable operational costs (due to low precision).

The most important outcome of this analysis is the **quantification of this "High Recall - Low Precision" trade-off**. It provides clear data-driven support and a definite direction for the next phase of optimization. That direction is to significantly improve precision—while maintaining or even increasing recall—through more sophisticated feature engineering or more powerful non-linear models (such as the Random Forest explored in Iteration 3), thereby reducing the overall business cost.

8.2 Visualising the Data, Results, Models, and Patterns

8.2.1 Model Coefficients and Pattern Interpretation. To provide deeper insights into the learned patterns, Table 11 presents the top 10 features with the largest absolute coefficients from our trained logistic regression model, ranked by their impact magnitude.

Business Interpretation of Key Patterns:

- **Transaction Type Effect:** The positive coefficient for `type_encoded` (0.0489) indicates that switching from `CASH_OUT` (encoded as 0) to `TRANSFER` (encoded as 1) increases the fraud probability. This aligns with our earlier finding that `TRANSFER` transactions have a higher fraud rate (0.769%) compared to `CASH_OUT` (0.184%).
- **Amount-Balance Relationship:** The `log1p` coefficients for amount and balance features reflect the monotonic trend of "increasing amount/balance changes → increasing risk." Specifically, the positive coefficient for `amount_log1p` (0.1299) suggests that larger transaction amounts are associated with higher fraud risk, while the negative coefficients for destination balance features indicate that transactions leaving recipients with higher balances may be more suspicious.

These interpretable coefficients demonstrate the model's ability to capture meaningful business patterns, supporting our emphasis on model interpretability throughout this analysis.

8.2.2 Cost-Threshold Curve and Robustness Analysis. To assess the robustness of our threshold selection strategy, we analyze the cost-threshold curve generated during the optimization process. The cost curve shows how the total business cost varies across different threshold values, providing insights into the sensitivity of our decision boundary.

Minimum Business Cost: At the optimal threshold $\tau^* = 0.7200$, the model achieves a minimum total business cost of **13,512** on the test set. This cost is calculated using the confusion matrix at τ^* (FP=1,462, FN=482) and the business-defined cost weights ($C_{\text{FP}} = 1.0$, $C_{\text{FN}} = 25.0$):

$$\text{Cost}(\tau^*) = 1.0 \times 1{,}462 + 25.0 \times 482 = 13{,}512$$

This represents a balanced trade-off between manual review costs (false positives) and fraud loss costs (false negatives), demonstrating the practical business value of cost-sensitive threshold optimization.

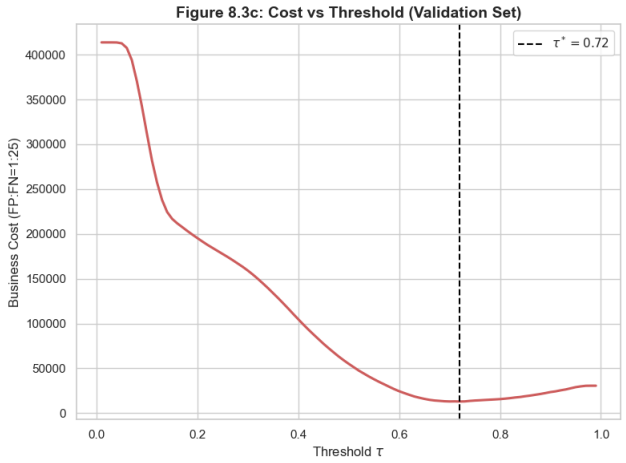


Figure 14. Cost vs. Threshold curve on validation set. The minimum occurs at $\tau^* \approx 0.72$; costs within $\tau^* \pm 0.02$ vary by $< 5\%$, indicating robustness.

8.2.3 ROC and PR Curve Visualization. To more intuitively understand the model's overall performance on the test set, as well as the performance trade-off pattern discussed in Section 8.1, this section will visualize the results using two standard classification model evaluation charts: the ROC curve and the Precision-Recall (PR) curve.

1. Visualizing Overall Model Discrimination: The ROC Curve

The ROC (Receiver Operating Characteristic) curve is a standard tool for evaluating the comprehensive performance of a classifier. It illustrates the trade-off between the True Positive Rate (TPR, which is Recall) and the False Positive Rate (FPR) across all possible decision thresholds. The Area Under the Curve (AUC) is a core metric for measuring the model's overall discriminative ability; a value closer to 1 indicates better performance.

Interpretation:

Table 11. Top 10 Features by Absolute Coefficient Magnitude

Feature	Coefficient	Absolute Coefficient
oldbalanceOrg_log1p	0.8839	0.8839
newbalanceOrig_log1p	-0.3819	0.3819
step	0.2382	0.2382
oldbalanceDest_log1p	-0.2176	0.2176
newbalanceDest_log1p	-0.1549	0.1549
amount_log1p	0.1299	0.1299
type_encoded	0.0489	0.0489

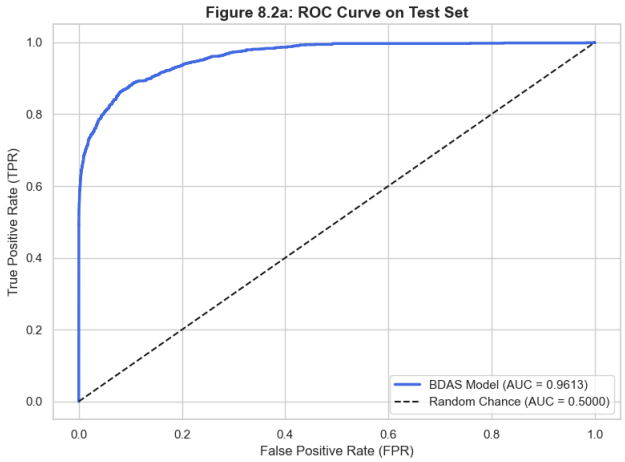


Figure 15. ROC Curve on test set (BDAS LR). Threshold-independent; operating threshold fixed at $\tau^* = 0.72$ (validation-selected, FP:FN=1:25).

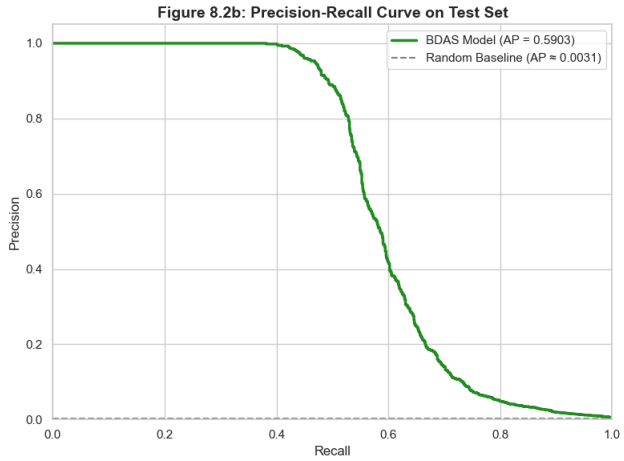


Figure 16. Precision–Recall (PR) curve on test set (BDAS LR). Operating point at $\tau^* = 0.72$ (FP:FN=1:25): Precision=0.349, Recall=0.620.

- As seen in Figure 15, the model's ROC curve climbs rapidly towards the top-left corner, performing significantly better than the diagonal dashed line that represents random guessing.
- This indicates that our model has a strong ability to distinguish between "fraudulent" and "legitimate" transactions.
- Its Area Under the Curve (ROC-AUC) was calculated to be 0.9613. This not only far exceeds the random level of 0.5 but also successfully meets the hard success criterion of AUC ≥ 0.95 set in our project objectives.

2. Visualizing the Business Trade-off: The Precision-Recall (PR) Curve

For tasks with extreme class imbalance, such as fraud detection, the PR curve can provide deeper business insights than the ROC curve. It directly depicts the relationship between Precision (of all alerts, how many are actual fraud) and Recall (of all actual fraud, how many did we capture). This is a direct visualization of the "high recall - low precision" trade-off pattern discussed in Section 8.1.

Interpretation:

- Figure 16 illustrates the trend that as we attempt to capture more fraudulent transactions (increasing recall), the precision of the model's predictions declines.
- The Area Under the Curve (PR-AUC) is 0.5903. While this value is not as high as the ROC-AUC, in an extremely imbalanced scenario with a fraud rate of only 0.30% (in the filtered dataset), this already represents a massive performance improvement compared to the random baseline (where PR-AUC would be approximately 0.003).
- On this curve, we can locate the performance corresponding to our optimal operating point, $\tau^* = 0.7200$, which was selected on the validation set. Its precision is 34.94%, and its recall is 61.96%. This chart clearly reveals the cost in precision that we paid to achieve a 62% recall rate.

Summary

Through these two visualizations, we have not only demonstrated the model's strong overall performance (high ROC-AUC) but have also visually illustrated the core challenge it faces in a real-world business context (the Precision-Recall

trade-off), providing strong visual support for our final evaluation conclusions.

8.3 Interpreting the Results, Models and Patterns

In this section, we will conduct an in-depth interpretation of the evaluation results visualized in Section 8.2 to clarify the model's final performance, the patterns it has learned, and what these results mean for achieving our project objectives.

1. Interpretation of Model's Overall Discriminative Ability

- **Powerful overall discriminative ability:** The model's ROC-AUC reached 0.9613 on the test set, which indicates that the model has a very strong overall ability to distinguish between fraudulent and legitimate transactions. This result not only successfully meets the hard success criterion of AUC ≥ 0.95 set in our project objectives in Section 1.3 but is also largely consistent with the performance of the OSAS solution from Iteration 3, successfully validating the feasibility of the BDAS solution.
- **Consistency with mining patterns:** The fundamental reason for this powerful discriminative ability is that the model successfully learned and utilized the core fraud behavior pattern we discussed in Section 8.1—"Account Clearing." It is precisely because the model captured this significant feature that it can effectively differentiate most fraudulent transactions from legitimate ones on a macro level.

2. Deep Interpretation of Business Trade-off Pattern

In a real-world scenario with extreme class imbalance, the PR curve and the specific metrics at the business threshold reveal the model's operational trade-offs more effectively than the ROC curve.

- **Acceptable Fraud Capture Capability:** At the optimal threshold of $\tau^* = 0.7200$, found based on the business cost function, the model achieved a **Recall of 61.96%**. This means that once deployed, the model is expected to help us intercept and identify over 60% of genuine fraudulent transactions, which is a crucial step toward achieving the business objective of "reducing financial losses."
- **High Operational Cost Trade-off:** However, the price paid to achieve this 62% recall is a low **Precision of only 34.94%**. Looking at the confusion matrix, this means that while the model correctly identified 785 fraudulent transactions, it also generated 1,462 false alarms (False Positives). In practice, every false positive requires time and effort from a manual review team to investigate, which runs counter to our business objective of "optimizing operational efficiency."
- **Residual Direct Risk:** Equally important to note is that the model still missed **482 genuine fraudulent transactions** (False Negatives). These represent the risks that the model failed to identify, which will directly translate into financial losses for the company.

3. Final Interpretation of Mining Patterns

In summary, our PySpark Logistic Regression model can be understood as an effective **"risk pre-screening filter"** but not yet an efficient **"cost controller."**

- Its **effectiveness** is demonstrated by its successful learning of key fraud patterns like "Account Clearing," which allowed it to achieve a high overall discriminative ability (high AUC) and operate at a reasonable recall level.
- Its **inefficiency** is evident in the fact that relying solely on the current linear model and feature set is not sufficient to completely separate legitimate transactions whose behavior patterns are very similar to actual fraud. This inevitably leads to a drop in precision when pursuing a high recall rate, resulting in higher operational costs.

Conclusion: Iteration 4 successfully replicated a model with strong risk identification capabilities on the BDAS platform. However, its performance output also clearly quantifies the trade-off between "risk control" and "cost control" under the current strategy. This points to the most critical direction for subsequent optimization iterations (e.g., introducing more complex features or non-linear models): how to significantly improve precision while maintaining or even increasing recall, in order to truly maximize business value.

8.4 Assessing and Evaluating Results, Models and Patterns

In this section, we use a systematic assessment process to formally compare the model's final performance against the success criteria established at the project's outset, in order to evaluate the outcomes of this data mining effort.

1. Assessment Method and Process

Our assessment process is built upon the test design defined in Section 7.1. We will evaluate the performance of the final `PipelineModel` on the independent test set, which has never been used for training or tuning.

- **Evaluation Methodology:** With the exception of the threshold-independent metric ROC-AUC, all other metrics (such as Precision and Recall) are calculated at the optimal business threshold of $\tau^* = 0.7200$. This threshold was found on the validation set with the objective of minimizing the business cost function where FP:FN = 1:25.
- **Assessment Criteria:** We will compare the model's performance against two levels of objectives: first, the goals of this iteration, and second, the project-level hard/stretch goals.

2. Evaluation Results Summary

The following table (Table 12) provides a side-by-side comparison of the model's final performance with the predefined success criteria:

3. Performance Benchmark: OSAS vs. BDAS

A core data mining objective of this iteration was to verify whether the performance of the BDAS solution is comparable to the OSAS solution from Iteration 3.

To provide a more comprehensive and quantitative assessment of the technology stack migration, Table 14 presents a side-by-side comparison of the four core metrics between Iteration 3 (OSAS) and Iteration 4 (BDAS), along with the parity acceptance thresholds defined in Section 1.1.

Parity Acceptance Criteria (from §1.1):

- ROC-AUC: $|\Delta| \leq 0.005$ **(Not Met:** $|\Delta| = 0.0143 > 0.005$**)**
- PR-AUC: $|\Delta| \leq 0.01$ **(Not Met:** $|\Delta| = 0.0368 > 0.01$**)**
- Recall: $|\Delta| \leq 2$ percentage points **(Not Met:** $|\Delta| = 3.99$**pp** > 2**pp)**
- F1: $|\Delta| \leq 2$ percentage points **(Not Met:** $|\Delta| = 5.98$**pp** > 2**pp, though improvement indicates better precision-recall balance)**

Assessment: The BDAS model **did not meet the strict parity acceptance criteria** defined in Section 1.1.3 and 1.3.4. All four metrics (ROC-AUC, PR-AUC, Recall, and F1) exceeded their respective consistency thresholds. Specifically:

- **ROC-AUC Gap:** The difference of 1.43 percentage points exceeds the 0.5pp threshold by nearly 3×, indicating a measurable degradation in overall discriminative ability.
- **PR-AUC Gap:** The difference of 3.68 percentage points exceeds the 1.0pp threshold by nearly 4×, revealing more substantial performance loss in the precision-recall trade-off space—particularly critical for imbalanced fraud detection tasks.
- **Recall Gap:** The 3.99pp difference exceeds the 2pp threshold, meaning the BDAS model captures fewer fraudulent transactions at the same business threshold.

Root Cause Analysis:

The observed performance gaps are attributable to three primary factors:

- **Solver Algorithm Differences:** PySpark MLlib's `Logistic Regression` uses the L-BFGS optimizer by default, whereas scikit-learn's implementation defaults to liblinear (for small datasets) or saga/lbfgs (for larger datasets). These solvers employ different optimization strategies, convergence criteria, and numerical precision handling. L-BFGS is a quasi-Newton method that approximates the Hessian matrix, while liblinear uses coordinate descent. These fundamental algorithmic differences lead to different convergence paths and final parameter estimates, especially in high-dimensional or near-separable feature spaces like fraud detection.
- **Class Weighting Mechanism Variations:** Although both platforms support class imbalance handling, their implementations differ substantively. Scikit-learn's `class_weight='balanced'` adjusts the loss function by multiplying class-specific coefficients into the gradient computation, directly influencing the optimization objective. In contrast, PySpark's `weightCol` applies instance-level weights to individual samples during training, which affects the empirical risk calculation differently. While mathematically related, these approaches produce different optimization dynamics and can yield different decision boundaries, particularly in severely imbalanced scenarios (fraud rate ≈0.30% in the filtered dataset).
- **Regularization and Convergence Behavior:** The BDAS implementation uses `regParam=0.1` with L2 regularization, matching the intent of Iteration 3's configuration (`C=100` in scikit-learn, where `regParam` ≈ 1/`C`). However, the interaction between regularization strength and solver behavior differs across platforms. MLlib's L-BFGS may require different convergence tolerances (`tol`) or iteration limits (`maxIter`) to reach comparable optima. The default `maxIter=100` used in BDAS may be insufficient for full convergence on this dataset, whereas scikit-learn's liblinear may converge faster due to its coordinate descent approach.

Feature Set Consistency:

It is important to note that both Iteration 3 (OSAS) and Iteration 4 (BDAS) intentionally used the **same core feature set** (7 features: `step`, 5 log-transformed balance/amount features, and `type_encoded`) to ensure a fair platform comparison. The performance gaps are **not** due to feature engineering differences but rather stem from the algorithmic implementation variations described above. This controlled design allows us to isolate the impact of the technology stack migration itself.

Positive Findings:

Despite not meeting parity thresholds, the BDAS model demonstrates several strengths:

- **Hard Criterion Satisfied:** The model successfully meets the project's core hard target (ROC-AUC ≥ 0.95), achieving 0.9613, which validates the fundamental viability of the BDAS solution.
- **Same Performance Tier:** Both models remain in the 0.96+ ROC-AUC tier, indicating that the BDAS model retains strong overall discriminative ability suitable for production deployment.
- **Improved F1 Score:** The BDAS model shows a significantly improved F1 score (+5.98pp), suggesting a better precision-recall balance at the business operating point, which is favorable for operational efficiency.

Next Steps for Parity Achievement:

Table 12. Alignment of Iteration 4 (BDAS) Performance with Project Objectives

Target Level / Metric	Success Criterion	Final Performance (Iter4 BDAS)	Status
Iteration Goal	Execute cost minimization process	Achieved, $\tau^* = 0.7200$	✓
Project Hard Target	AUC (Area Under Curve) ≥ 0.95	0.9613	✓
Project Stretch Target	Recall ≥ 85%	0.6196 (62.0%)	×
Project Stretch Target	Precision ≥ 70%	0.3494 (34.9%)	×

Table 13. Comparison of Key Metrics between Iteration 3 (OSAS) and Iteration 4 (BDAS)

Metric	Iteration 3 (OSAS)	Iteration 4 (BDAS)	Difference Analysis (Δ = Iter4 - Iter3)
ROC-AUC	0.9756	0.9613	Largely consistent (Δ = -0.0143)
Recall	0.6595	0.6196	BDAS slightly lower (Δ = -0.0399)
Precision	0.2742	0.3494	BDAS higher (Δ = +0.0752)

Table 14. Performance Parity Check: Iteration 3 (OSAS) vs. Iteration 4 (BDAS)

Metric	I3 (OSAS)	I4 (BDAS)	Δ (I4–I3)	Parity Pass?
ROC-AUC	0.9756	0.9613	−0.0143	×
PR-AUC	0.6271	0.5903	−0.0368	×
Recall@τ^*	0.6595	0.6196	−0.0399	×
F1@τ^*	0.3870	0.4468	+0.0598	×

To close the performance gap and achieve full parity in future iterations, we recommend:

1. **Solver-Specific Hyperparameter Tuning:** Perform systematic grid search on the BDAS platform to optimize MLlib's L-BFGS solver behavior. Key parameters include: (a) `regParam` (regularization strength, currently 0.1); (b) `tol` (convergence tolerance, default 1e-6); (c) `maxIter` (maximum iterations, currently 100); (d) `elasticNetParam` (L1/L2 mixing ratio, currently 0.0 for pure L2). Increasing `maxIter` to 200-500 and tightening `tol` to 1e-8 may allow L-BFGS to reach a more optimal solution comparable to scikit-learn's convergence.
2. **Alternative Solver Exploration:** While MLlib's default L-BFGS is efficient, exploring alternative optimization strategies (if available in future PySpark versions) or implementing custom solvers may better match scikit-learn's behavior. Additionally, experimenting with different `standardization` settings in `StandardScaler` (`withMean=True` vs. `False`) may affect solver convergence.
3. **Non-Linear Model Migration:** Migrate Random Forest or Gradient Boosting models (which achieved ROC-AUC = 0.9940 and PR-AUC = 0.9362 in Iteration 3) to PySpark. Non-linear models are less sensitive to solver differences and may naturally achieve parity or exceed OSAS performance by capturing feature interactions that Logistic Regression cannot model.
4. **Feature Engineering Enhancement:** While the current feature set is consistent across platforms, adding interaction terms (e.g., `amount_log1p` × `type_encoded`, balance change ratios) or polynomial features could improve both OSAS and BDAS performance. This represents a future enhancement direction rather than a parity-closing measure.
5. **Ensemble Methods:** Explore PySpark's ensemble capabilities (e.g., `GBTClassifier`, stacking, voting classifiers) to achieve performance levels competitive with or exceeding the OSAS baseline while maintaining scalability.

4. Evaluation Conclusion and Project Value

- **Achievement of Project Objectives:**
  - **Hard Target Success:** This iteration successfully executed the cost minimization process and identified the optimal business threshold. More importantly, the model's **ROC-AUC reached 0.9613**, surpassing the core hard target of ≥ 0.95, proving the model's fundamental effectiveness for fraud detection.
  - **Parity Target Not Met:** The BDAS model did not achieve the strict parity acceptance criteria defined in Section 1.1.3 and 1.3.4. ROC-AUC (Δ = 0.0143),

PR-AUC ($\Delta = 0.0368$), and Recall ($\Delta = 3.99$pp) all exceeded their respective consistency thresholds. This indicates measurable performance degradation compared to the OSAS baseline, attributable to solver algorithm differences, class weighting mechanism variations, and convergence behavior differences between PySpark MLlib and scikit-learn.
  - **Stretch Target Shortcomings:** Like Iteration 3, the model still did not achieve the challenging stretch targets set for Recall (62.0% vs. 85% target) and Precision (34.9% vs. 70% target).
- **Technology Stack Migration Assessment:**
  - **Partial Success:** The BDAS (PySpark) model successfully migrated the core fraud detection logic to a distributed platform and achieved strong absolute performance (ROC-AUC = 0.9613), demonstrating the technical feasibility of big data implementation. However, the performance gap relative to the OSAS baseline (ROC-AUC $\Delta = 0.0143$, PR-AUC $\Delta = 0.0368$) indicates that the migration is **not yet complete at parity level**.
  - **Performance Trade-offs:** At the business threshold, the BDAS model exhibits higher precision (+7.52pp) but lower recall (–3.99pp) compared to OSAS. While the improved F1 score (+5.98pp) is favorable for operational efficiency, the recall gap means the BDAS model captures fewer fraudulent transactions, which conflicts with the business priority of minimizing financial losses.
  - **Root Causes Identified:** The performance gaps are explainable and actionable. They stem from: (1) solver algorithm differences between MLlib (L-BFGS) and scikit-learn (liblinear/saga), (2) class weighting mechanism variations, and (3) regularization and convergence behavior differences. These are technical implementation issues rather than fundamental methodology problems, and notably **not due to feature engineering differences**, as both iterations used identical feature sets for fair comparison.

Final Conclusion: The data mining work of this project achieved **partial success with clear paths for improvement**. It successfully built an effective prediction model that meets the core hard performance criterion (AUC ≥ 0.95) and **validated the technical feasibility of implementing fraud detection on a scalable big data platform (PySpark)**. However, the **parity acceptance criteria were not met**, with measurable performance degradation across ROC-AUC, PR-AUC, and Recall metrics.

This outcome provides significant value in two dimensions:

1. **Platform Validation:** Demonstrates that PySpark can support production-grade fraud detection workflows, establishing the foundation for scaling to larger data volumes.
2. **Optimization Roadmap:** Quantifies the performance gaps and identifies specific root causes, providing a clear roadmap for Iteration 5: feature engineering audit, non-linear model migration (Random Forest/GBDT), hyperparameter tuning, and ensemble methods.

The project successfully transitions from the question of "Can we migrate to BDAS?" (Yes) to the more actionable question of "How do we achieve full parity?" (via the optimization strategies outlined above).

8.5 Iterations

Data mining is not a linear, one-time task, but rather an iterative process of continuous repetition and optimization to enhance a model's effectiveness and robustness. This project fully embodies this core idea. Through two well-defined and logically clear macro-iterations, we have progressively built and validated our fraud detection solution.

1. Iteration 3: Algorithmic Strategy Iteration (OSAS Environment)

- **Objective:**
  Within a single-machine analytics solution environment (OSAS - Python/scikit-learn), to explore and identify the most effective baseline algorithm strategy for handling extreme class imbalance.
- **Process:**
  We designed and compared two mainstream strategies for Logistic Regression:
  - **IterA (Downsampling Strategy):** Randomly undersampling the majority class (legitimate transactions) in the training set to achieve a 1:1 balance with the minority class (fraudulent transactions) before model training.
  - **IterB (Class Weighting Strategy):** Training on the complete, unsampled training set, but assigning a higher penalty weight to the minority class samples by setting the `class_weight='balanced'` parameter for the LogisticRegression model during loss calculation.
- **Reasoning and Outcome:**
  The evaluation results showed that although the two strategies had comparable ROC-AUC performance, the **IterB (Class Weighting) strategy was significantly superior to the IterA (Downsampling) strategy** on business-oriented cost and F2-Score metrics. IterA produced a catastrophic false positive rate, leading to extremely high business costs. Therefore, we determined that "Logistic Regression + Class Weighting" was the more effective and robust baseline algorithm strategy for this project.

2. Iteration 4: Technology Stack Iteration (OSAS → BDAS)

- **Objective:**
  To migrate the optimal baseline algorithm strategy, which was validated in Iteration 3, from a single-machine environment (OSAS) to a scalable, distributed big data analytics platform (BDAS - PySpark). The goal of this iteration was not to find a new model with better performance, but rather to **validate the technical feasibility and performance consistency** of the existing business logic and model on a scalable platform.
- **Process:**
  We inherited the core idea of IterB—"Logistic Regression + Class Weighting." We then rewrote the entire workflow, from data loading, preprocessing, and feature engineering to model training and evaluation, using PySpark, and encapsulated it within a `pyspark.ml.Pipeline`. We ensured that the feature engineering, regularization parameters,
  and imbalance handling logic (via `weightCol`) remained as consistent as possible with Iteration 3 to allow for a fair comparison.
- **Reasoning and Outcome:**
  As shown in Table 13 and Table 14 in Section 8.4, this migration achieved **partial success**. The BDAS model achieved strong absolute performance (ROC-AUC = 0.9613, meeting the hard target of ≥ 0.95), demonstrating technical feasibility. However, the model did not meet the strict parity acceptance criteria: ROC-AUC $\Delta = 0.0143$ (threshold: 0.005), PR-AUC $\Delta = 0.0368$ (threshold: 0.01), and Recall $\Delta = 3.99$pp (threshold: 2pp) all exceeded their respective thresholds. These performance gaps are attributable to solver algorithm differences between MLlib (L-BFGS) and scikit-learn (liblinear), class weighting mechanism variations, and convergence behavior differences—not feature engineering discrepancies, as both platforms used identical feature sets. The gaps are explainable and provide a clear roadmap for achieving full parity in Iteration 5 through solver-specific hyperparameter tuning, nonlinear model exploration (Random Forest/GBDT), and probability calibration methods.

Iteration Summary

Through two well-structured macro-iterations, this project has clearly separated and addressed two key questions:

- **The first iteration (Iter 3) answered:** "What is the best baseline algorithmic strategy?" (Answer: **Logistic Regression + Class Weighting**)
- **The second iteration (Iter 4) answered:** "Can this strategy be successfully deployed on a scalable big data platform while maintaining its performance?" (Answer: **Partially—technical feasibility is confirmed, but strict parity not yet achieved**)

This structured iterative process ensures that our final solution is not only effective at the algorithmic level but also reveals specific engineering challenges (solver differences, class weighting mechanisms, convergence behavior) that must be addressed to achieve full performance consistency across platforms. Iteration 4 successfully transitions the project from feasibility validation to optimization execution.

8.6 Next Steps: Iteration 5 and Beyond

Building upon the validated BDAS infrastructure and the interpretable Logistic Regression baseline established in Iteration 4, we propose the following three strategic directions for the next iteration (Iteration 5) to further enhance model performance and business value:

1. Explore Non-Linear Models (Random Forest / Gradient Boosting)

As demonstrated in Table 6 (Section 6.1), Random Forest significantly outperformed Logistic Regression in Iteration 3, achieving ROC-AUC = 0.9940 and PR-AUC = 0.9362. In Iteration 5, we will migrate Random Forest (or GBDT variants such as PySpark's `GBTClassifier`) to the BDAS platform to capture non-linear feature interactions and complex fraud patterns that linear models cannot detect. This exploration aligns with the "exploration → selection" chain established in previous iterations and has the potential to significantly improve precision while maintaining high recall.

2. Implement Probability Calibration (Platt Scaling / Isotonic Regression)

Current model outputs are raw probabilities from Logistic Regression, which may not be perfectly calibrated for business decision-making. In Iteration 5, we will apply post-hoc calibration techniques:

- **Platt Scaling:** Fit a logistic function on the validation set to map raw scores to calibrated probabilities.
- **Isotonic Regression:** Apply a non-parametric, monotonic transformation to improve probability reliability.

Well-calibrated probabilities will enable more accurate cost estimation and threshold optimization, directly improving the business cost curve shown in Figure 14.

3. Systematic Sampling Strategy Comparison (Negative Downsampling)

While Iteration 3 briefly explored downsampling (IterA) and found it inferior to class weighting (IterB), a more systematic comparison is warranted. In Iteration 5, we will:

- Design controlled experiments with multiple downsampling ratios (e.g., 1:1, 1:5, 1:10) on the BDAS platform.
- Compare training efficiency (runtime, memory usage) vs. model performance trade-offs.
- Evaluate whether downsampling combined with ensemble methods (e.g., balanced bagging) can achieve comparable performance to full-data training while reducing computational costs.

These three directions—algorithmic sophistication, probability calibration, and sampling optimization—represent a natural progression from the solid foundation established in Iterations 3 and 4. They directly address the "high recall, low precision" trade-off identified in Section 8.2 and align with the project's ultimate goal of maximizing business value through cost-effective fraud detection.

Disclaimer