Technical Report

1. Introduction

Healthcare fraud costs the U.S. healthcare system more than $68 billion annually, draining resources from legitimate patients. Since CMS (Centers for Medicare & Medicaid Services) can manually investigate only a small fraction of suspicious cases, there is a strong need for an intelligent automated detection system.

This project develops an **end-to-end machine learning pipeline** capable of identifying high-risk healthcare providers using a real Medicare fraud dataset from Kaggle.

The goals are:

- Detect fraudulent providers at the provider level
- Handle severe class imbalance (~9% fraud)
- Build interpretable and practical models
- Justify all steps: data cleaning, feature engineering, model selection, tuning, and error analysis

All implementation steps were originally completed in a **single notebook (mlproj2)** before being divided into the required three-notebook structure. Because of this workflow, some steps may depend on earlier transformations.

---

2. Dataset Description

The dataset contains four CSV files at different granularities:

1. **Train_Beneficiarydata.csv**
    - Beneficiary demographics (DOB, DOD, Gender, Race, State)
    - Chronic condition indicators (e.g., CHF, cancer, diabetes)
    - **Granularity:** BeneID
2. **Train_Inpatientdata.csv**
    - Inpatient hospital claims
    - Claim dates, reimbursement amounts, deductibles
    - Diagnosis codes, physician IDs
    - **Granularity:** Claim-level (BeneID → Provider)
3. **Train_Outpatientdata.csv**
    - Outpatient visits, tests, procedures
    - Similar structure to inpatient
    - **Granularity:** Claim-level
4. **Train_Labels.csv**

- Provider fraud labels ("Yes" / "No")
- **Granularity:** Provider

Key Relationships

- **BeneID** links beneficiary → claims
- **Provider** links claims → fraud label

Thus, modeling must be done at the provider level, requiring extensive aggregation across all tables.

---

3. Data Understanding & Exploration (1.5.1)

3.1 Initial Inspection

We performed:

- `.info()` for data types
- `.isnull().sum()` for missing values
- `.shape + .nunique()`
- Validated date columns
- Checked Beneficiary–Claim–Provider coverage

Findings:

- Some missing dates
- Missing chronic conditions
- Strongly skewed reimbursement amounts
- Providers appear across datasets inconsistently

3.2 Beneficiary Analysis

We converted DOB/DOD to datetime and computed:

- Age distribution (mainly 70–80+)
- Gender distribution
- Race distribution
- Renal disease prevalence
- Chronic condition prevalence
- State distribution

Beneficiaries exhibit many chronic conditions, typical for Medicare.

3.3 Claims Analysis

Performed on both inpatient & outpatient claims:

- Monthly claim counts

- Claim duration
- Reimbursement and deductible distributions
- Temporal trends
- Outliers
- Geographic patterns

Findings:

- Monthly claim volume fluctuates
- Reimbursement amounts highly skewed
- State-level concentration of claims

---

4. Provider-Level Aggregation Strategy

Since labels are provider-level, we aggregated all claim-level features.

4.1 Inpatient Aggregations

For each provider:

- Sum / mean / std of **InscClaimAmtReimbursed**
- Sum / mean of **DeductibleAmtPaid**
- Count of inpatient claims
- Unique attending / operating / other physicians

4.2 Outpatient Aggregations

Identical aggregator set applied.

4.3 Combined Provider Features

Created:

- `total_claims`
- `inpatient_ratio`
- `avg_claim_amount`
- `physician_variety`
- Chronic condition percentages per provider

This produced the provider-level feature table for modeling with **51 engineered features**.

---

5. Advanced Feature Engineering

5.1 High-Cost Claim Percentages

Using 90th-percentile thresholds:

- `pct_high_cost_inpatient`
- `pct_high_cost_outpatient`
- `pct_high_cost_total`

5.2 Chronic Condition Intensity

Converted chronic condition indicators to binary and computed:

- Mean prevalence per provider
- Chronic condition ratios for inpatient/outpatient
- `pct_chronic_patients`

5.3 Operational Features

- 30-day readmission rate
- Physician utilization
- Cost-per-physician

---

6. Class Imbalance Analysis (1.5.2)

Fraud distribution:

- **No fraud:** ~91.13% (3,816 providers)
- **Fraud:** ~8.87% (399 providers)

Approximately **9.5:1 imbalance ratio**.

Visualized:

- Fraud vs. non-fraud pie chart
- Class counts

Insight:
A model predicting “No fraud” always would still achieve 91.13% accuracy.

---

7. Imbalance Strategy: Class Weighting

Class weights using sklearn’s `compute_class_weight('balanced')`:

- **Class 0 weight (No fraud):** 0.567
- **Class 1 weight (Fraud):** 4.258

Why class weighting?

1. No data loss
2. Avoids synthetic samples
3. Penalizes false negatives
4. Works with many algorithms

For XGBoost: `scale_pos_weight = 7.52`.

---

8. Algorithm Selection (1.5.3)

Models implemented with tuning:

1. Decision Tree
2. Random Forest
3. Gradient Boosting / XGBoost
4. Logistic Regression

SVM Considered but Not Implemented

Due to:

- High computational cost
- Poor performance on imbalanced tabular data
- Low interpretability
- Outperformed by tree ensembles + logistic regression

---

9. Validation Strategy

We used a robust two-part strategy:

✔ **5-fold cross-validation** (GridSearchCV)
✔ **20% hold-out test set**

Data splits:

- Total providers: 4,215
- Training: 3,372
- Test: 843

---

10. Comparative Model Performance

Baseline Models (class weighting):

Model	Precision	Recall	F1	Accuracy
Decision Tree	0.444	0.449	0.447	0.859
Random Forest	0.667	0.430	0.523	0.900
XGBoost	0.553	0.533	0.543	0.886
Logistic Regression	0.413	0.794	0.543	0.830

Tuned Models:

Model	Precision	Recall	F1	Accuracy
Decision Tree (Tuned)	0.397	**0.832**	0.538	0.819
Random Forest (Tuned)	0.593	0.654	0.622	0.899
XGBoost (Tuned)	0.464	**0.841**	0.598	0.856
Logistic Regression (Tuned)	0.449	**0.860**	**0.590**	0.848

Key Findings

- Recall most important
- Logistic Regression balances precision/recall best
- Trees yield high precision but miss more fraud
- Tuning dramatically boosts recall

---

11. Experiment Log — Detailed Trial Documentation

11.1–11.8 Model Summaries

- **Baseline Decision Tree:** Recall 0.449, unstable
- **Tuned Decision Tree:** Recall 0.832
- **Baseline Random Forest:** High precision, low recall
- **Tuned Random Forest:** Recall 0.654
- **Baseline XGBoost:** Moderate
- **Tuned XGBoost:** Recall 0.841
- **Baseline Logistic Regression:** Strong recall 0.794
- **Tuned Logistic Regression:** Best overall recall 0.860

---

12. Final Model Selection: Logistic Regression

Primary Reason

- **Highest recall (0.860)**

- Minimizes false negatives
- Aligns with CMS mission

Secondary Advantages

- Interpretability
- Stability
- Efficiency
- Probabilistic outputs

Top Fraud Indicators

- High % expensive claims
- Many unique physicians
- High chronic condition intensity
- Balanced claim mix correlated negatively with fraud

---

13. Evaluation & Error Analysis

Test Set Results (Tuned Logistic Regression):

- Accuracy: 0.848
- Precision: 0.449
- Recall: 0.860
- F1: 0.590

Confusion Matrix

```
               Predicted
              No      Yes
Actual No   [TN=625]  [FP=111]
Actual Yes  [FN=15]   [TP=92]
```

Error Patterns

False Positives (111):

- High claim amounts
- Multiple physicians
- Explosive costs

False Negatives (15):

- Low claim volume
- Subtle fraud

- Hard to distinguish

---

14. Business Impact Analysis

With Logistic Regression

- Fraud detection: **86%**
- FP rate: **15.1%**
- Investigation workload: **24%**
- Missed fraud: **15 providers**

Compared with Other Models

- Random Forest: misses 37 fraud cases
- XGBoost: high recall but less interpretable
- Decision Tree: unstable performance

---

15. Limitations & Future Work

Improvements

- Threshold tuning
- Cost-sensitive learning
- Ensemble combinations

Advanced Features

- Temporal patterns
- Network analysis
- Anomaly detection

Operational

- Model monitoring
- Feedback loop retraining
- Explainability dashboard

---

16. Conclusion

This project delivered a **complete Medicare fraud detection pipeline**, including:

1. Comprehensive data exploration

2. 51 engineered provider-level features
3. Class-weighting for imbalance
4. Full model comparison
5. Logistic Regression with **0.860 recall** selected
6. Detailed error analysis

The final system is **transparent, practical, and aligned with CMS objectives**, maximizing fraud detection while supporting investigators.

Deployment Recommendation:
Use Logistic Regression with a **0.3–0.4 probability threshold** and integrate an explainability dashboard.