Manuel Youssef Haik 13006600

Part 1: Data Understanding & Exploration

Step 1: Understand the Data Structure

Datasets we worked with:

- **Train_Beneficiarydata.csv** (`bene`): Contains demographic and health information for beneficiaries (patients)
- **Train_Inpatientdata.csv** (`inp`): Contains inpatient claim records
- **Train_Outpatientdata.csv** (`outp`): Contains outpatient claim records
- **Train_labels.csv** (`labels`): Contains provider-level fraud labels

- Figured out how the datasets connect to each other:
  - Beneficiaries link to claims using BeneID
  - Claims link to fraud labels using Provider
- Confirmed all providers in the claims data have matching labels (perfect alignment)

Made sure that:

- Total unique providers in claims data: 5,410
- Total unique providers in labels data: 5,410
- Providers in claims but not in labels: 0
- Providers in labels but not in claims: 0

Step 2: Check Data Quality

- **Found missing values** in many columns, especially:
    - Procedure codes: 99-100% missing (dropped these columns)
    - Some physician columns: 40-88% missing (dropped the worst ones)
    - Diagnosis codes: Many had high missing rates (dropped columns with >85% missing)
- **Fixed inconsistent data:**
    - Changed 'Y' to 1 and '0' to 0 in the RenalDiseaseIndicator column
    - Converted all date columns to proper date format
    - Replaced missing values: 'UNKNOWN' for text, median for numbers
- **Checked for problems:**
    - No duplicate claims found
    - No date errors (all dates make sense)
    - Found some very high claim amounts (kept them as they might indicate fraud)

Step 3: Explore the Data

- **Looked at fraud distribution:** Found 506 fraudulent providers (9.4%) vs 4,904 non-fraudulent (90.6%)
- **Compared fraudulent vs non-fraudulent providers:**
    - **Money:** Fraudulent providers have higher total amounts, higher average claim amounts
    - **Volume:** Fraudulent providers submit more claims
    - **Diversity:** Fraudulent providers use more different diagnosis codes and procedure codes
    - **Networks:** Fraudulent providers work with more different doctors
    - **Geography:** Fraudulent providers serve more states and counties
    - **Patients:** Fraudulent providers serve slightly younger patients and fewer deceased patients
- **Created visualizations:** Made charts showing these differences (bar charts, boxplots, histograms, time series)

Step 4: Create Provider-Level Features

What we did:

- Combined inpatient and outpatient claims into one dataset
- For each provider, calculated:
    - **Counts:** Total claims, unique diagnosis codes, unique procedure codes, unique doctors, unique states/counties
    - **Money metrics:** Total reimbursement, average claim amount, maximum claim amount, inpatient/outpatient ratio
    - **Patient info:** Average age, percentage deceased, average chronic condition rates
    - **Time patterns:** Monthly claim counts and reimbursement amounts
- Merged everything into one file: provider_features.csv with 35 features per provider

Result: One row per provider with all important information aggregated from their claims.

Part 2: Class Imbalance Strategy

Step 1: Analyze the Imbalance

- Only 9.4% of providers are fraudulent (506 out of 5,410)
- This is a 9.7:1 ratio (almost 10 non-fraudulent for every 1 fraudulent)
- **Problem:** Without special handling, a model might just predict "non-fraudulent" for everyone and still get 90% accuracy, but miss all the fraud

---

Step 2: Choose a Strategy

What we chose: Class Weighting

Why:

- **Doesn't lose data:** Unlike undersampling (which throws away data) or oversampling (which creates fake data), class weighting keeps all real data
- **Works with many models:** Most machine learning algorithms support class weighting
- **Makes sense for fraud:** Missing fraud (False Negatives) is much worse than flagging a legitimate provider (False Positives)
- **Simple and effective:** Easy to implement and understand

How it works:

- The model pays more attention to correctly identifying fraudulent providers
- Mistakes on fraudulent providers are penalized more heavily during training
- This helps the model learn to detect fraud even though there are fewer examples

Trade-offs:

- Better at catching fraud (higher recall)
- Might flag more legitimate providers as fraudulent (lower precision)
- This is acceptable because we can manually review flagged cases

Result: Strategy selected and ready to use in model training.

Step 3: Choose Evaluation Metrics

What we chose: Precision, Recall, F1-Score, and PR-AUC

Why not just accuracy:

- Accuracy is misleading with imbalanced data
- A model could get 90% accuracy by always saying "non-fraudulent"
- But it would miss 100% of the fraud (0% recall for fraud)

Why these metrics:

- **Precision:** How many of the providers we flag as fraudulent are actually fraudulent (reduces false alarms)
- **Recall:** How many of the actual fraudulent providers we catch (most important - we can't miss fraud)
- **F1-Score:** Balances precision and recall into one number
- **PR-AUC:** Overall performance on detecting fraud (better than ROC-AUC for imbalanced data)

Priority:

1. **Recall** is most important (must catch fraudulent providers)
2. **PR-AUC** shows overall fraud detection performance
3. **F1-Score** gives balanced view
4. **Precision** is important but secondary (we can review false alarms)

Result: Clear metrics to evaluate model performance.

Dareen Soliman 13007056 & Lama Hany 13002973

PART 02_modeling.ipynb, is dedicated to building and comparing several Machine Learning models for a fraud detection classification task, specifically focusing on identifying 'Potential Fraud' among provider features.

The central goal is to select the model that provides the best balance between catching fraudulent activity (Recall) and minimizing false alarms (Precision), often measured by the F1-score, in a highly imbalanced dataset.

1. Data Loading and Initial Analysis

The first section of the notebook loads the dataset and establishes the challenge:

Metric	Detail
Number of Samples	5,410
Number of Features	35 (mostly numeric, two categorical)
Class Imbalance	Highly imbalanced: 90.6% 'No Fraud' vs. 9.3% 'Fraud'
Model Strategy	Use tree-based models and techniques like scale_pos_weight to handle the nonlinearity and high imbalance of fraud patterns.

2. Preprocessing

This step prepares the data for training, focusing on encoding categorical variables and calculating the class imbalance weight.

Code and Output Summary (Preprocessing)

Action	Code	Detail
Library Imports	Imports pandas, sklearn modules, XGBClassifier, and LGBMClassifier	Sets up the environment for modeling and evaluation.
Encoding	df['PotentialFraud'] = LabelEncoder().fit_transform(df['PotentialFraud'])	Converts the target variable 'PotentialFraud' (Yes/No) and the 'Provider' ID into numerical integers (0/1).
Data Split	train_test_split(X, y, ..., stratify=y)	Splits the features (X) and target (y) into 80% training and 20% test sets, using stratification to ensure the imbalance ratio is preserved in both sets.
Imbalance Weight	fraud_weight = (y_train == 0).sum() / (y_train == 1).sum()	Calculates the ratio of non-fraud to fraud samples, which is used as the scale_pos_weight hyperparameter in boosting models to penalize false negatives more heavily.
Output:	scale_pos_weight = 9.6864...	This value tells the model that a false negative is about 9.7 times more costly than a false positive.

3. Model Comparison (Hyperparameter Tuning)

The notebook compares four models: XGBoost, LightGBM, Random Forest, and Logistic Regression, using GridSearchCV with an F1-score metric, which is crucial for imbalanced classification.

LightGBM Model (LGBMClassifier)

LightGBM is chosen as a primary candidate due to its efficiency and strong performance on tabular, imbalanced data.

- **Code:** `GridSearchCV` **searches over** `n_estimators`, `max_depth`, `learning_rate`, **and** `num_leaves`.
- **Key Parameter:** `scale_pos_weight=9.686` **is used to handle the class imbalance.**
- **Tuning Output:** `Best LightGBM Params: {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 100, 'num_leaves': 31}`.

Random Forest Model (RandomForestClassifier)

Random Forest is chosen as a strong candidate due to its robustness, interpretability, and stable performance on tabular, imbalanced data.

- **Code:** `GridSearchCV` **searches over** `n_estimators`, `max_depth`, `min_samples_split`, **and** `min_samples_leaf`.
- **Key Parameter:** `class_weight='balanced'` **is used to handle the class imbalance.**
- **Tuning Output:** `Best Random Forest Params: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}`

Logistic Regression Model (LogisticRegression)

Logistic Regression is chosen as a baseline candidate due to its simplicity, interpretability, and effectiveness with imbalanced datasets when class weights are applied.

- **Code:** `GridSearchCV` **searches over** `C`, `penalty`, **and** `solver` **within a pipeline with** `median imputation`.
- **Key Parameter:** `class_weight='balanced'` **is used to handle the class imbalance.**
- **Tuning Output:** `Best Logistic Regression Params: {'classifier__C': 1, 'classifier__penalty': 'l2', 'classifier__solver': 'lbfgs'}`

Model Performance Comparison

The following table summarizes the key results for the fraud class (class 1) from the final classification reports for each model on the test set:

Model	Precision (Fraud, Class 1)	Recall (Fraud, Class 1)	F1-score (Fraud, Class 1)	Overall Accuracy	Interpretation
LightGBM	0.55	0.73	0.63	0.92	Best overall balance and highest F1-score. Chosen for deployment.
Random Forest	0.53	0.76	0.63	0.91	High recall, tied for best F1-score, but slightly lower precision than LightGBM.
XGBoost	0.47	0.78	0.59	0.90	Highest recall, but lowest precision among tree-based models (many false positives).
Logistic Regression	0.40	0.87	0.55	0.87	Highest recall, but the lowest F1-score and accuracy due to poor precision (too many false positives).

The final conclusion is that LightGBM provides the best-balanced performance metrics, specifically the highest F1-score of 0.63, and is therefore selected as the model for deployment.

4. Final Prediction

The final steps involve applying the best model (LightGBM) to an unlabeled, holdout dataset for prediction.

Code and Output Summary (Prediction)

- **Load Test Data:** Loads `Test_Provider_Features.csv` and retains the original `Provider` ID for the output file.
- **Preprocessing:** The test data is prepared to match the features used during training: the `Provider` ID is encoded and then dropped from the feature set (`X_test_final`), and the column order is matched.
- **Prediction:** The trained `lgbm_best` model predicts the fraud labels (`y_pred_final`) for the test set.
- **Save Results:** A DataFrame is created combining the original `Provider` IDs with the predicted `PotentialFraud` labels and is saved to a CSV file.
- **Output:** The code prints `Saved: LightGBM_TestProvider_Predictions.csv`.

The notebook concludes that the final output file is clean and ready for reporting or submission.

Osama Loay Osama 13006531

Evaluation Metrics

To compare candidate models, we calculated a set of core performance metrics: Precision, Recall, F1-score, ROC-AUC, and PR-AUC (average precision). Each of these highlights a different aspect of model behavior. Precision helps control how many cases are sent for investigation, which is important from an operational cost perspective. Recall reflects the model's ability to catch truly risky providers. F1 offers a single measure that balances both. ROC-AUC provides a view of the model's overall ability to separate classes, and PR-AUC is particularly useful given the imbalance in the dataset.

Confusion Matrix and Cost Impact

For the held-out test set, we generated confusion matrices (TN, FP, FN, TP) for each model and assessed the cost implications using a simple formula:

Cost = FP × C_FP + FN × C_FN

This connects the model's errors to real-world consequences—false positives translate to unnecessary investigations, while false negatives represent potentially costly missed fraud. The weightings CFPC_{FP}CFP and CFNC_{FN}CFN can be adjusted by the business team, and thresholds can be re-evaluated accordingly.

Error Analysis: False Positives and False Negatives

We reviewed several representative FP and FN cases to understand where the models struggled. For each, we recorded the provider ID, the prediction probability, the top contributing features, and brief contextual notes.

For false positives, we typically saw legitimate providers flagged because of unusual but explainable patterns—for example, a high *average reimbursement* driven by a small patient pool. In false negatives, we found examples where the model failed to recognize more subtle or rare combinations of features.

Potential improvements include adding temporal behavior , incorporating interaction features such as reimbursements per patient, using specialty/risk indicators, and relying more heavily on SHAP explanations during feature engineering.

Measures to Reduce Overfitting

Several steps were taken to limit overfitting during development:

- Using a dedicated validation set and StratifiedKFold splits
- Applying regularization (e.g., `C` for logistic regression, depth and child-sample limits for tree models)
- Enabling early stopping for boosting methods
- Restricting overall tree complexity

Generalization was checked against the test set only after model and threshold choices were finalized. Threshold tuning was based strictly on validation results.

Experiment Tracking

We maintained a concise log of all experiments, documenting the model type, key hyperparameters, cross-validation scores, and the difference between validation and test results.
Example entry:

LightGBM — num_leaves=31, learning_rate=0.05
CV ROC-AUC: 0.82
Validation Recall: 0.68
Notes: Adjusting `min_child_samples` helped reduce variance.

A lightweight CSV or table is sufficient for sorting and comparing trials and supports reproducibility.

Model Selection Criteria

Our selection rule prioritized models that minimized false positives while still achieving a recall of at least `TARGET_RECALL`. When multiple models satisfied this condition, we chose the one with the higher precision. If none met the recall threshold, we selected the model with the lowest number of false positives.

The final model, its decision threshold, confusion matrix, and associated cost calculation were documented in `reports/model_selection.txt` and `reports/model_comparison.csv`.

Example: Detailed FP/FN Case Notes

For each highlighted misclassified case, the report includes:

- Provider ID
- True label and predicted probability
- Predicted label under the chosen threshold
- The top five contributing features (via SHAP or feature importance)
- A short note explaining the likely reason for misclassification and ideas for improvement