ML project 2
Data cleaning and exploration (done to all files):

---

1. **Date parsing**
   All date columns (DOB, DOD, etc.) were converted to datetime format using errors="coerce" to safely handle invalid or missing values.
2. **All duplicate data was dropped**
3. **Handled missing numeric/payment values:**
   Filled missing payment fields (DeductibleAmtPaid, InscClaimAmtReimbursed) with 0.
4. **Handled missing date values:**
   Example: Missing DOD interpreted as patient alive (not data is not present), so a new attribute is created; is_alive which is better indicator than actual value.
5. **Missing durations safely handled using minimum-day logic**
   Minimum number of patient being present is 1.
6. **Cleaned diagnosis & procedure codes:**
   Converted to string and stripped whitespace. Replaced placeholder strings ('nan','none','unknown','na','','nan.') with NaN. Created <col>_present binary indicators. Summarized into num_diagnoses and num_procedures in relevant dates.
7. **Cleaned physician ID fields (Outpatient):**
   Normalized physician IDs. Created frequency-encoded <phys>_freq and presence indicators.

Feature engineering (divided based on done in which table individually, and commonly done in all tables)

Beneficiary Table (Feature Engineering)

---

1. **Alive status (is_alive)**
   The Medicare dataset often leaves DOD blank for living patients.
   We created:

- is_alive = 1 if DOD is null
- is_alive = 0 otherwise

2. **Age feature (age)**
   Age was computed as the number of years between:

- DOB and DOD for deceased beneficiaries, or
- DOB and the current pd.Timestamp.today() for living beneficiaries.

The result is stored as an integer (`age`).

3. **Coverage summary (`has_any_coverage`)**
   All columns whose name contains `"NoOfMonths_Part"` (e.g. `NoOfMonths_PartACovered`, `NoOfMonths_PartBCovered`, …) were summed row-wise to determine whether a beneficiary ever had Medicare coverage:

- `has_any_coverage = True` if the total covered months > 0
- `has_any_coverage = False` otherwise

This binary coverage flag was later merged into the inpatient and outpatient tables to flag claims filed during no coverage via `claim_during_no_coverage`.

4. **Chronic condition flags used downstream**
   The original chronic condition indicators were retained and later aggregated by provider:

- `ChronicCond_Alzheimer`
- `ChronicCond_Heartfailure`
- `ChronicCond_Cancer`

These describe patient clinical risk profiles and are later averaged per provider, along with a count of beneficiary-claim rows (`num_patients`) as a proxy for patient volume.

Inpatient Table (Feature Engineering)

---

1. **Physical length of stay:**
   `los_days = DischargeDt - AdmissionDt`
2. **`Admission_after_Discharge`**
   `Admission_after_Discharge = 1` if `AdmissionDt > DischargeDt`
3. **`ClaimStart_before_Admission`**
   `ClaimStart_before_Admission = 1` if `ClaimStartDt < AdmissionDt`
4. **`ClaimEnd_after_Discharge`**
   `ClaimEnd_after_Discharge = 1` if `ClaimEndDt > DischargeDt`
5. **`ClaimEnd_after_Discharge_no_patient_payment`**
   `ClaimEnd_after_Discharge_no_patient_payment = 1` if `ClaimEnd_after_Discharge == 1` and `patient_paid == 0`
6. **`ClaimStart_before_Admission_no_payment`**
   `ClaimStart_before_Admission_no_payment = 1` if `ClaimStart_before_Admission == 1` and `any_payment == 0`
7. **Length-based anomaly:**

- `claim_length_vs_los_ratio = claim_length_days / los_days` (with safeguards for 0 / missing)
- `claim_length_much_greater_than_los = 1` if ratio > 2

Outpatient Table (Feature Engineering)

Most feature engineering logic applied to inpatient is also applied to outpatient; outpatient-specific differences (such as physician ID cleaning and frequency encoding) are handled as described in the common and provider-level sections below.

Inpatient & Outpatient Table (common Feature Engineering)

1. **Date anomalies and risk flags**

Using `today = pd.Timestamp.today()`:

- `ClaimStart_after_ClaimEnd = 1` if `ClaimStartDt > ClaimEndDt`
- `ClaimStart_before_ClaimEnd = 1` if `ClaimStartDt < ClaimEndDt` (expected normal case)
- `claim_duration_days = ClaimEndDt − ClaimStartDt`
- `Future_dates = 1` if `ClaimStartDt` or `ClaimEndDt > today`
- `ClaimStart_before_Now_but_End_missing = 1` if `ClaimStartDt` exists but `ClaimEndDt` is missing

Conditional versions:

- `ClaimStart_after_ClaimEnd_no_payment = 1` if `ClaimStart_after_ClaimEnd == 1` and `any_payment == 0`
- `Future_dates_no_payment = 1` if `Future_dates == 1` and `any_payment == 0`

2. **Encoding for diagnosis code and procedure cost**

For each:

- Cleaned code values (stripping and treating placeholders such as `"nan"`, `"none"`, `"unknown"`, `"na"`, `""` as missing).
- Created `<col>_present = 1` if a valid code is present, `0` otherwise.

Then:

- `num_diagnoses` = sum of diagnosis `_present` columns
- `num_procedures` = sum of procedure `_present` columns

3. **Utilization / unbundling proxies**

- `procedures_per_day = num_procedures / (|claim_duration_days| + 1)`

- `same_day_multiple_claims_flag`: same logic as inpatient (multiple claims for the same `(BeneID, Provider, ClaimStartDt)` on a given day).
- `visits_per_bene_provider`: total visits per `(BeneID, Provider)` pair computed and merged back.

4. **Death/coverage flags (via merged beneficiary data):**

- `claim_after_death` and `days_after_death` as in inpatient
- `claim_during_no_coverage` based on `has_any_coverage`
- `claim_after_death` and `days_after_death` as in inpatient

5. **Payment features**

- Missing `DeductibleAmtPaid` and `InscClaimAmtReimbursed` were filled with 0 and cast to float.
- `total_payment = DeductibleAmtPaid + InscClaimAmtReimbursed`
- `patient_paid = 1` if `DeductibleAmtPaid > 0`
- `insurer_paid = 1` if `InscClaimAmtReimbursed > 0`
- `any_payment = 1` if either payment component is positive

6. **Expected vs actual payment (Find out what would be a normal payment per day given diagnoses on claim):**

- Step 1: compute total payments = `DeductibleAmtPaid + InscClaimAmtReimbursed` to produce numeric total payment for each claim
- Step 2: calculate claim length: `ClaimEndDt - ClaimStartDt`
- Step 3: calculate payment per day: `Total Payment ÷ Claim Length` (nonzero)
- Step 4: extract diagnosis code, create table with `ClaimID`, `Diagnosis` and `payment per day`. Outcome: dataset where each diagnosis is linked to the claim's Payment per day
- Step 5: clean missing values
- Step 6: compute diagnosis level stats; for each of `diag_code` compute: `count`, `mean_ppd`, `median_ppd`
- Step 7: handle diagnosis that appear a few times (producing unstable averages). Solution: use weighted smoothing (Bayesian shrinkage: Average payment per day across all diagnoses, Average payment per day for a specific diagnosis, Based on how many claims contain that diagnosis, More claims → higher weight, Rare diagnoses → lower weight (more shrinkage))
- Step 8: map expected cost back to each claim. The claim now has expected cost values derived from diagnoses
- Step 9: compare actual vs expected
- Step 10: flag rare diagnosis as in medical sector rare diagnosis often correlate with: higher fraud risk

7. **Create a weighted "date anomaly score"**

A scoring system was built to rank claims based on how suspicious their date patterns are. Each anomaly type is grouped by severity and assigned a weight.

Weight 3 — Most Severe Date Issues
Flags:

- `ClaimStart_after_ClaimEnd`
- `Admission_after_Discharge` (inpatient only)
- `Future_dates`

Meaning:
These represent impossible timelines such as:

- A claim starting after it ends
- Admission occurring after discharge
- Claims dated in the future

Such patterns strongly suggest fabricated or incorrectly submitted claims, so they receive the highest weight.

Weight 2 — Major Suspicious Issues
Flags:

- `ClaimEnd_after_Discharge_no_patient_payment`
- `ClaimStart_after_ClaimEnd_no_payment`
- `Future_dates_no_payment`
- `claim_length_much_greater_than_los` (inpatient)

Meaning:
These indicate serious inconsistencies where the timeline is abnormal and no payment occurred, increasing suspicion.

Examples:

- Claim continues past discharge and the patient paid nothing
- A future-dated claim with no payment
- Stay duration far longer than the medical length of stay

These are strong fraud indicators, so they receive a weight of 2.

Weight 1 — Minor Suspicious Issues
Flags:

- `ClaimStart_before_Admission_no_payment`
- `ClaimStart_before_Now_but_End_missing`

Meaning:
These are softer anomalies—unexpected but not impossible. For example:

- Claim starts before admission yet no payment is made
- `ClaimStart` exists but `ClaimEnd` is missing

These inconsistencies may still indicate unusual billing but are less severe, receiving a weight of 1.

Final Score
The `date_anomaly_score =` weighted sum of all triggered anomalies.
A secondary unweighted count (`date_issue_count`) was also created to measure how many issues occurred, independent of severity.

Fraud Types & Corresponding Engineered Features

1. **Billing for services never rendered**

- `claim_after_death`, `days_after_death`
- `claim_during_no_coverage`
- Date anomaly flags and `date_anomaly_score` (impossible / inconsistent timelines)
- `same_day_multiple_claims_flag` (multiple claims same day for same bene–provider)
- Outpatient diagnosis–payment mismatches:
  - `claim_to_diag_expected_mean`, `claim_to_diag_expected_max`
  - `expected_vs_actual_gap / expected_vs_actual_pct`

2. **Upcoding (billing for higher-cost procedures than performed)**

- **Inpatient:**
  - `claim_payment_ppd` vs `diag_expected_mean_ppd / diag_expected_max_ppd`
  - `expected_vs_actual_gap_ppd`, `expected_vs_actual_pct_ppd`
  - `claim_has_rare_diag`
    (engineered at claim level; not yet aggregated in provider table)
- **Outpatient:**
  - `claim_to_diag_expected_mean` (provider-level mean)
  - `expected_vs_actual_gap`, `expected_vs_actual_pct` (claim level)
  - `claim_has_rare_diag` (provider-level mean)

3. **Unbundling (splitting services that should be billed together)**

- `procedures_per_day` (inpatient & outpatient; provider-level mean)
- `same_day_multiple_claims_flag` (provider-level mean)
- `visits_per_bene_provider` (claim-level intensity measure)
- High `num_procedures / num_diagnoses` (provider-level mean and max)

4. **Submitting claims for deceased patients**

- `claim_after_death` (claim-level, with provider-level sum and mean)
- `days_after_death` (magnitude of post-death billing)

5. **Prescribing unnecessary treatments for financial gain**

- `procedures_per_day` (over-treatment intensity)
- `visits_per_bene_provider` (frequent repeat visits)
- Long stays / durations:
    - `claim_length_days`, `los_days`, and `claim_length_much_greater_than_los` (inpatient)
- `num_physicians` (multiple physicians involved per claim)
- `num_diagnoses` (complexity inflation)
- Expected vs actual payment gaps (inpatient ppd and outpatient total payment-based ratios)

6. **Kickback or referral schemes**

- `visits_per_bene_provider` (strong repeated relationships between specific patients and providers)
- `same_day_multiple_claims_flag` (clustered billing patterns)
- Beneficiary mix & volumes:
    - `ChronicCond_Alzheimer`, `ChronicCond_Heartfailure`, `ChronicCond_Cancer` (homogeneous or unusual risk profiles)
    - `num_patients` (extreme patient volume per provider)

Visualizations & Exploratory Analysis

1. **Missing data structure – Inpatient & Outpatient**

- **Missingness heatmaps:** For both inpatient and outpatient, I plotted binary heatmaps (rows = claims, columns = only variables with missing values) to visually inspect which features suffer from missingness and whether there are row-wise patterns (e.g., entire blocks of dates missing).
- **Missingness barplots:** For each of inpatient and outpatient, I computed the percentage of missing values per column and plotted sorted bar charts. This helped identify which specific variables (e.g., certain dates or diagnosis/procedure codes) have high missing rates, guiding later cleaning/encoding decisions.

2. **Target class distribution – Provider fraud labels**

- I plotted a bar chart of `PotentialFraud` from `train_labels` (Yes/No) to visualize the class imbalance at provider level.
- This confirmed that the dataset is imbalanced (fewer fraudulent providers) and justified the later use of imbalance-aware metrics and resampling/weighting strategies.

3. **Payment distributions – Inpatient vs Outpatient**

- I created a `total_payment` variable for both inpatient and outpatient as: `InscClaimAmtReimbursed + DeductibleAmtPaid` (with missing values filled as 0).
- Using a reusable function, I plotted histograms with KDE for `total_payment` separately for inpatient and outpatient claims.

- These plots showed the overall payment distribution, presence of heavy right tails / outliers, and differences between inpatient and outpatient cost profiles, motivating later use of robust statistics and careful scaling.

4. **Provider-level behavior vs fraud label (post-aggregation)**

4.1 Main payment / intensity feature vs fraud

- After building the provider-level table (`train_final`), I plotted a boxplot + jittered stripplot of a key provider metric (`out_total_payment_mean` if available, otherwise `out_claim_procedures_per_day_mean`) against `PotentialFraud`.
- I overlaid group means as text on the plot to highlight differences.
- This visualization shows how average outpatient payment (or procedures per day) differs between fraudulent and non-fraudulent providers, and that fraud-labeled providers often have higher central tendency or more extreme values.

4.2 Same-day multiple claims vs fraud

- I plotted a boxplot of `out_same_day_multiple_claims_flag_mean_y` vs `PotentialFraud`, where the feature represents the average rate of same-day multiple claims per provider.
- This captures whether fraudulent providers tend to split claims into multiple same-day submissions more frequently than non-fraudulent ones.

4.3 Number of patients vs fraud

- I plotted a boxplot of `num_patients_y` vs `PotentialFraud` as a proxy for provider size / patient volume.
- This helps interpret whether fraudulent providers tend to have unusually high or low patient counts, which can signal either over-servicing or suspiciously concentrated activity.

5. **Correlation heatmap – Provider-level features**

- I selected a small set of important provider-level features:
  - `out_same_day_multiple_claims_flag_mean_y`
  - `ChronicCond_Alzheimer_y`, `ChronicCond_Heartfailure_y`, `ChronicCond_Cancer_y`
  - `num_patients_y`
- I computed the correlation matrix between these variables and plotted an annotated heatmap using `sns.heatmap`.
- This allowed me to check for multicollinearity and to see that behavioral fraud signals (same-day multiple claims, patient counts) are largely independent from chronic condition prevalence, supporting their use as distinct predictive features.

6. **Temporal patterns – Claims volume and payments over time**

6.1 Yearly claim counts

- For both inpatient and outpatient, I extracted `ClaimYear` from `ClaimStartDt` and plotted countplots of claims per year.
- Along with printed counts, these visualizations revealed strong growth in claim volume over time (e.g., surge from 2008 → 2009), providing context for temporal drift and possible anomalies.

6.2 Monthly inpatient payment trend

- For inpatient claims, I extracted `ClaimMonth` (year–month period) and computed the average `total_payment` per month.
- I plotted this as a time series line plot to visualize the trend of mean inpatient payment over time.
- The plot shows that after early fluctuations, monthly averages stabilize, which gives a baseline for spotting unusual spikes or drops in later analysis and supports the interpretation of temporal stability vs. abnormal periods.

Aggregation to Provider

Inpatient Provider-Level Aggregation

Aggregated inpatient claim features per provider to capture payment behavior, clinical complexity, temporal anomalies, and physician patterns:

- **Payments & intensity**
  - `total_payment` → mean, sum, max, median, std
  - `claim_payment_ppd` (payment per day) → mean, max, median
- **Clinical complexity & volume**
  - `num_diagnoses` → mean, max
  - `num_procedures` → mean, max
- **Payment behavior**
  - `patient_paid` → mean (share of claims where patient contributes)
  - `insurer_paid` → mean (share of claims reimbursed by insurer)
  - `any_payment` → mean (overall paid-claim rate)
- **Coverage & death-related fraud flags**
  - `claim_after_death` → sum, mean
    (how often provider bills after patient DOD)
  - `days_after_death` → mean, max
    (average and worst delay between DOD and claim)
  - `claim_during_no_coverage` → mean
    (share of claims submitted when beneficiary has no Medicare coverage)
- **Utilization & unbundling / over-treatment proxies**
  - `procedures_per_day` → mean, max
  - `visits_per_bene_provider` → mean, max
    (how often each BeneID–Provider pair appears)

    - `same_day_multiple_claims_flag` → mean
      (rate of splitting a beneficiary's care into multiple same-day claims)
- **Date anomaly score & temporal issue count**
    - `date_anomaly_score` → mean, max
      (weighted severity of all date-related anomalies per provider)
    - `date_issue_count` → mean, max
      (average and worst number of date issues per claim)
- **Granular date anomaly rates (per provider)**
    - `ClaimStart_after_ClaimEnd` → mean
    - `Admission_after_Discharge` → mean
    - `Future_dates` → mean
    - `ClaimEnd_after_Discharge` → mean
    - `ClaimEnd_after_Discharge_no_patient_payment` → mean
    - `ClaimStart_before_Admission` → mean
    - `ClaimStart_before_Admission_no_payment` → mean
    - `ClaimStart_before_Now_but_End_missing` → mean
    - `claim_length_much_greater_than_los` → mean
- **Diagnosis-based payment expectations (upcoding / overbilling)**
    - `diag_expected_mean_ppd` → mean, max
      (typical expected payment-per-day given diagnoses)
    - `diag_expected_max_ppd` → mean
    - `diag_present_count` → mean, max
      (how many diagnosis slots are actively "explaining" payment)
    - Ratios & gaps:
        - `claim_ppd_to_diag_expected_mean` → mean, max, median
        - `claim_ppd_to_diag_expected_max` → mean, max
        - `expected_vs_actual_gap_ppd` → mean, max
        - `expected_vs_actual_pct_ppd` → mean, max
- **Physician network behavior**
    - `num_physicians` → mean, max (physician involvement per claim)
    - `AttendingPhysician_freq` → mean, max
    - `OperatingPhysician_freq` → mean, max
    - `OtherPhysician_freq` → mean, max
      (concentration of work around a small set of physicians)
- **Rare diagnosis usage**
    - `claim_has_rare_diag` → mean
      (how often the provider uses rare diagnosis codes)

All inpatient provider-level features are prefixed with `inp_` (e.g., `inp_total_payment_mean`, `inp_date_anomaly_score_max`).
Diagnosis–payment expectation features are now explicitly aggregated per provider, not just left at claim level.

Outpatient Provider-Level Aggregation

Aggregated outpatient claim features per provider to capture ambulatory behavior, cost vs diagnosis expectations, and date irregularities:

- **Payments, duration & volume**
  - `total_payment` → mean, sum, max, median, std
  - `claim_duration_days` → mean, max
  - `claim_duration_days_nonzero` → mean
  - `num_diagnoses` → mean, max
  - `num_procedures` → mean, max
- **Payment behavior**
  - `patient_paid` → mean
  - `insurer_paid` → mean
  - `any_payment` → mean
- **Coverage & death-related fraud flags**
  - `claim_after_death` → sum, mean
  - `days_after_death` → mean, max
  - `claim_during_no_coverage` → mean
- **Utilization / unbundling & visit patterns**
  - `procedures_per_day` → mean, max
  - `same_day_multiple_claims_flag` → mean
  - `visits_per_bene_provider` → mean, max
- **Date anomaly score & issue count**
  - `date_anomaly_score` → mean, max
  - `date_issue_count` → mean, max
- **Granular outpatient date anomaly rates**
  - `ClaimStart_after_ClaimEnd` → mean
  - `Future_dates` → mean
  - `ClaimStart_before_Now_but_End_missing` → mean
  - `ClaimStart_after_ClaimEnd_no_payment` → mean
  - `Future_dates_no_payment` → mean
- **Diagnosis-based expected payment (upcoding / unnecessary visits)**
  - `diag_expected_mean_payment` → mean, max
  - `diag_expected_max_payment` → mean
  - `diag_present_count` → mean, max
  - Ratios & gaps:
    - `claim_to_diag_expected_mean` → mean, max, median
    - `claim_to_diag_expected_max` → mean, max
    - `expected_vs_actual_gap` → mean, max
    - `expected_vs_actual_pct` → mean, max
- **Rare diagnosis usage**
  - `claim_has_rare_diag` → mean
- **Physician network behavior**
  - `num_physicians` → mean, max
  - `AttendingPhysician_freq` → mean, max
  - `OperatingPhysician_freq` → mean, max
  - `OtherPhysician_freq` → mean, max

All outpatient features are prefixed with `out_` (e.g., `out_total_payment_mean`, `out_claim_to_diag_expected_mean_max`).

Beneficiary Provider-Level Aggregation

Using merged inpatient–beneficiary and outpatient–beneficiary tables, we summarize each provider's patient mix, age profile, chronic conditions, and coverage pattern:

- **Patient volume (unique)**
  - `BeneID → nunique`
    → renamed to `bene_num_unique_patients`: number of distinct beneficiaries seen by the provider.
- **Demographics**
  - `age` → mean, min, max
    (overall age profile of patients seen)
  - `is_alive` → mean
    (fraction of currently alive beneficiaries; indirect proxy for very old/high-risk populations)
- **Chronic disease burden (per provider)**
  - `ChronicCond_Alzheimer` → mean
  - `ChronicCond_Heartfailure` → mean
  - `ChronicCond_Cancer` → mean
    (share of beneficiaries with each major chronic condition)
- **Coverage profile**
  - `has_any_coverage` → mean
    (fraction of patients that ever had Medicare coverage across Parts)

All beneficiary-based features are prefixed with `bene_`, e.g.:

- `bene_num_unique_patients`
- `bene_age_mean`, `bene_age_max`
- `bene_ChronicCond_Heartfailure_mean`, `bene_has_any_coverage_mean`

These features describe how complex and high-risk the provider's patient population is, and how often they are actually covered.

Final Provider-Level Dataset Construction

1. **Prefixing strategy**

- Inpatient provider-level features → prefixed with `inp_`
- Outpatient provider-level features → prefixed with `out_`
- Beneficiary mix features → prefixed with `bene_`

This avoids name collisions and encodes the data source directly into the feature name.

2. **Merging all components (Train)**

For the training provider-level matrix:

- Start from `train_labels` (one row per Provider, with `PotentialFraud` label).

- Merge on `Provider` with:
  - `inpatient_provider`
  - `outpatient_provider`
  - `beneficiary_provider`

Result: one row per provider, aggregating:

- Inpatient behavior (`inp_*`)
- Outpatient behavior (`out_*`)
- Beneficiary mix & chronic conditions (`bene_*`)
- Target label (`PotentialFraud` → later cast to 0/1)

3. **Merging all components (Test)**

For the test set:

- Start from `test_id` (unique Provider identifiers).
- Merge on `Provider` with:
  - `inpatient_test_provider`
  - `outpatient_test_provider`
  - `beneficiary_test_provider`

Result: same schema as training (minus `PotentialFraud`), used for final fraud-risk scoring.

4. **Handling missing values**

- After merges, providers that appear only in one source (e.g., only outpatient) will have NaN in the other blocks.
- All missing aggregated features are filled with 0 to indicate "no evidence / no activity in that channel" rather than dropping the provider

1.5 Modeling

This section describes the full modeling workflow used to develop a provider-level fraud detection system. Multiple supervised learning algorithms were implemented, compared, and optimized using techniques appropriate for highly imbalanced classification tasks.

1.5.1 Algorithms Evaluated

We implemented and compared **five machine-learning models** commonly used in fraud detection:

- **Logistic Regression**
  Interpretable linear baseline; limited for complex fraud patterns.

- **Decision Tree Classifier**
  Captures non-linear interactions but prone to overfitting.
- **Random Forest Classifier**
  Ensemble of diverse trees; robust to noise and imbalance.
- **Gradient Boosting Classifier**
  Sequential boosting algorithm that captures subtle, high-variance fraudulent behaviors.
  Provided the strongest predictive performance.
- **Support Vector Machine (RBF kernel)**
  Strong boundary separation model; requires feature scaling; heavier computationally.

All applicable models were trained with **class_weight = 'balanced'** to address class imbalance.

1.5.2 Data Splitting Strategy

A **stratified train/validation/test split** was applied:

- **60% training**
- **20% validation** → used for threshold tuning
- **20% test** → untouched until final evaluation

Stratification ensures the minority fraud class remains proportionally represented.

1.5.3 Handling Class Imbalance

Imbalance mitigation is critical in fraud detection. We applied:

- **class_weight='balanced'** for all models
- **PR-AUC & F1** prioritized over accuracy
- **Threshold tuning** to optimize detection of rare fraud instances

These help ensure the model does not default to predicting all providers as “Not Fraud.”

1.5.4 Model Training Details

- Logistic Regression and SVM used **scaled features** via StandardScaler.
- Tree-based models trained on unscaled numeric features.
- Hyperparameter tuning was performed via **RandomizedSearchCV** for:
  - Random Forest: depth, estimators, min samples

  - Gradient Boosting: learning rate, depth, estimators

1.5.5 Model Comparison

Models were evaluated on the validation set using:

- Precision
- Recall
- F1 Score
- ROC-AUC
- PR-AUC

Gradient Boosting achieved the strongest performance, especially on PR-AUC and recall — the two most important metrics for fraud detection.

Therefore, **Gradient Boosting was selected as the final model**.

1.5.6 Threshold Optimization

Instead of using a default 0.5 cutoff, we optimized the classification threshold by:

- sweeping thresholds ∈ [0.1, 0.9],
- computing F1 scores at each threshold,
- selecting the threshold that maximized F1.

This improved recall of fraud cases without excessively increasing false positives.

1.5.7 Final Model Selection Rationale

Gradient Boosting was chosen due to:

- **Highest PR-AUC**
- **Best F1/Recall trade-off**
- **Ability to capture non-linear interactions**
- **Consistent CV performance**
- **Interpretability via feature importance**

It aligns with the dataset properties and fraud-detection priorities.

1.6 Evaluation

This section evaluates the final model using rigorous metrics, validation procedures, threshold tuning, cost-based analysis, and detailed error inspection to understand the model's strengths and limitations.

1.6.1 Validation Procedure

The evaluation used:

1) Stratified 60/20/20 split

- Training: model fitting
- Validation: threshold tuning & model selection
- Test: final unseen performance check

2) 5-Fold Stratified Cross-Validation

Used on training+validation combined to:

- verify stability,
- reduce variance,
- detect potential overfitting.

1.6.2 Evaluation Metrics

Metrics suitable for imbalanced fraud detection:

Metric	Purpose
Precision	Avoid accusing legitimate providers.
Recall	Catch as many fraud providers as possible.
F1-Score	Combines precision & recall.
ROC-AUC	Measures ranking ability across thresholds.
PR-AUC	Best measure when fraud cases are rare.
Confusion Matrix	Shows real-world error composition.

Accuracy was intentionally **not** used.

1.6.3 Threshold Tuning

We tuned the threshold using the **validation PR curve**:

- Compute precision, recall across probability thresholds
- Compute F1 for each threshold
- Select threshold maximizing F1

This threshold was then applied to:

- validation predictions
- refitted model (train+validation)
- final test predictions

1.6.4 Cost-Based Evaluation

Fraud detection has asymmetric costs:

- **False Negative (FN)** → missing a fraudulent provider → *very high cost*
- **False Positive (FP)** → wrongly flagging a good provider → *moderate cost*

We computed:

Expected Cost = FP × FP_COST + FN × FN_COST

With FP_COST=1000, FN_COST=10000 (example values):

- thresholds around 0.25–0.35 minimized total expected cost.

This supports using a **lower threshold** than the standard 0.5.

1.6.5 Confusion Matrix, ROC, and PR Curves

Diagnostics showed:

- ROC curve with high AUC → strong ranking ability
- PR curve → significantly better than baseline
- Confusion matrices → acceptable false positive rate and improved recall

1.6.6 Error Analysis (Required Case Studies)

We analyzed misclassified cases from the **test set**:

False Positives (legitimate flagged as fraud)

Drivers included:

- unusually high total payments,
- high volume of same-day claims,
- diagnosis patterns deviating from typical providers.

These cases are *anomalous but not fraudulent.*

False Negatives (fraud missed by the model)

Drivers included:

- moderate payments that don't look suspicious,
- anomalies diluted by generally normal claim patterns,
- missing temporal/behavioral features.

These suggest needed future improvements.

1.6.7 Insights & Future Refinements

To reduce FP/FN rates:

- Add temporal features (claim velocity, trend shifts)
- Add peer-group normalization
- Replace binary rules with anomaly-score features
- Consider cost-sensitive boosting or monotonic GBM variants

1.6.8 Overfitting Prevention

We prevented overfitting using:

- proper train/val/test isolation
- stratified cross-validation
- Gradient Boosting regularization (tree depth, learning rate)
- threshold tuning on validation only
- test set untouched until final evaluation

The consistent performance across splits confirms good generalization.