Fraud Detection Project Documentation

1. Project Overview

The goal is to predict fraudulent healthcare providers using provider-level claims data.
Data sources:

- Train_Beneficiarydata-1542865627584.csv – beneficiary demographics and chronic conditions
- Train_Inpatientdata-1542865627584.csv – inpatient claims
- Train_Outpatientdata-1542865627584.csv – outpatient claims
- Train-1542865627584.csv – provider labels (PotentialFraud)

Target variable:

- target = 1 for fraud, 0 for non-fraud

Dataset characteristics:

- Imbalanced (~10% fraud)
- Provider-level features are aggregated from claim and beneficiary data

---

2. Data Preprocessing & Feature Engineering

2.1 Loading and inspecting data

- Loaded CSV files and checked shapes and sample records.
- Checked missing values for inpatient/outpatient data.

```
print(inp.isna().sum())

print(outp.isna().sum())
```

2.2 Target mapping

- Converted PotentialFraud to numeric target:

```
labels["target"] = labels["PotentialFraud"].map({"Yes":1, "No":0})
```

- Visualized class imbalance:

```
labels["target"].value_counts().plot(kind="bar")
```

2.3 Feature aggregation

- **Inpatient/Outpatient features**: count of claims, total reimbursed amount per provider.

```
inpatient_features = inp.groupby("Provider").agg({
   "ClaimID": "count",
```

"InscClaimAmtReimbursed": "sum"

}).rename(columns={ "ClaimID": "Inpatient_Claims", "InscClaimAmtReimbursed": "Inpatient_TotalAmount"})

- **Beneficiary features**:
    - Average age (Age) calculated from DOB
    - Unique patients
    - Chronic condition percentage

bene['Age'] = (REFERENCE_DATE_FOR_AGE - bene['DOB']).dt.days / 365.25

bene['HasChronicCondition'] = (bene[chronic_cols] == 1).any(axis=1).astype(int)

- **Merged features** to provider-level dataframe and filled missing values with 0.
  Rationale: Providers with no claims or missing beneficiaries still need to be included.

---

3. Train-Test Split & Scaling

- Split dataset: 80% train, 20% test, stratified by target.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

- **Scaling** applied to SVM only (tree-based models handle raw features).

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

- **Class imbalance handling**:
    - Computed class weights: {0: 0.55, 1: 5.34}
    - Optional SMOTE oversampling tested but not used in final models.

---

4. Model Selection & Training

Models used:

- Decision Tree (interpretable)
- Random Forest (robust ensemble)
- Gradient Boosting (strong for imbalanced datasets)
- SVM (handles high-dimensional data)

Training details:

- Applied class weights for Decision Tree, Random Forest, SVM, and sample weights for Gradient Boosting.
- Models saved to Google Drive for reproducibility.

---

5. Evaluation & Experiment Log

- **Metrics:** Accuracy, Precision, Recall, F1-score, ROC-AUC, PR-AUC
- **Observations:**
    - Tree models achieve better balance of recall and precision than SVM.
    - SVM achieves 100% recall but extremely low precision → many false positives.

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))

- **Experiment log:**
    1. Tried SMOTE → increased recall but reduced precision slightly.
    2. Applied class weights → improved F1 for fraud.
    3. Tested all four models → Gradient Boosting gave best recall with moderate precision.

---

6. Error Analysis

- **False Positives**: Non-fraud providers predicted as fraud.
- **False Negatives**: Fraud providers predicted as non-fraud.
- Tree-based models reduce false negatives compared to SVM.

Impact: High recall models are critical (avoid missing fraud), but too many false positives increase investigation costs.

---

7. Feature Importance (Tree Models)

- Top features influencing fraud detection:

sns.barplot(x="Importance", y="Feature", data=imp_df.head(10))

- Example: High inpatient/outpatient claims, high chronic patient percentage → stronger fraud signal.

---

8. Curves & Visualizations

- **Precision-Recall Curve**:

precision, recall, _ = precision_recall_curve(y_test, y_prob)

pr_auc = auc(recall, precision)

- **ROC Curve**:

roc_auc = roc_auc_score(y_test, y_prob)

- Observations: PR-AUC low (~0.11) due to imbalance; ROC-AUC moderate (~0.62).

---

9. Recommendations

1. **Feature engineering**: Add claim frequency trends, unusual cost spikes.
2. **Imbalance handling**: Advanced SMOTE or anomaly detection.
3. **Model tuning**: Gradient Boosting hyperparameters, ensemble stacking.
4. **Evaluation**: Focus on recall and PR-AUC, not only ROC-AUC.