1. Introduction

Healthcare fraud is a significant concern, resulting in billions of dollars in losses annually. Detecting potential fraud in provider claims is critical to ensuring the integrity of the healthcare system.

This project aims to generate provider-level features from multiple healthcare datasets to facilitate fraud detection using machine learning models. The datasets include beneficiary data, inpatient claims, outpatient claims, and labels indicating potential fraud.

---

2. Data Sources

The following datasets were used:

Dataset	Description	Shape
Train_Beneficiarydata	Contains demographic information for beneficiaries	(X, Y)
Train_Inpatientdata	Contains inpatient claims data	(X, Y)
Train_Outpatientdata	Contains outpatient claims data	(X, Y)
Train	Contains fraud labels for providers	(X, Y)

Initial Exploration:

- **Beneficiary Data:**
  Displayed first rows and basic info, verifying completeness and types.
- **Inpatient & Outpatient Claims:**
  Checked numeric and categorical features and handled missing values.
- **Labels:**
  Converted PotentialFraud column into binary target (Yes → 1, No → 0).

---

3. Data Preprocessing

3.1 Handling Missing Values

- **Numeric Columns: Filled missing values with the median of each column.**
- **Categorical Columns: Filled missing values with the mode of each column.**

Result: All missing values in inpatient and outpatient datasets were handled, ensuring clean data for feature engineering.

```
inp[num_cols_inp] = inp[num_cols_inp].fillna(inp[num_cols_inp].median())

outp[num_cols_outp] = outp[num_cols_outp].fillna(outp[num_cols_outp].median())
```

3.2 Handling Outliers

Outliers were addressed using IQR filtering (Interquartile Range method) to remove extreme values from numeric features.

```
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return df[(df[column] >= lower) & (df[column] <= upper)]
```

---

4. Exploratory Data Analysis (EDA)

4.1 Target Distribution

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

- **Fraud Cases (1): X**
- **Non-Fraud Cases (0): Y**

Observation: Dataset is imbalanced, with more non-fraud cases than fraud cases.

4.2 Claim Aggregation per Provider

- **Inpatient Features: Total number of claims and total reimbursed amount per provider.**
- **Outpatient Features: Total number of claims and total reimbursed amount per provider.**

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

outpatient_features = outp.groupby("Provider").agg({"ClaimID": "count", "InscClaimAmtReimbursed": "sum"})
```

---

5. Beneficiary Features

5.1 Age Calculation

- **Converted DOB to datetime and calculated age using a reference date (2009-12-31).**

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

5.2 Chronic Condition Feature

- **Chronic conditions are coded as 1 = Yes and 2 = No.**
- **Created binary feature HasChronicCondition indicating if a patient has any chronic condition.**

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

5.3 Aggregation by Provider

Calculated the following per provider:

Feature	Description
Avg_Patient_Age	Mean age of beneficiaries
Unique_Patients	Number of unique patients per provider
Chronic_Patient_Perc	Percentage of patients with at least one chronic condition

---

6. Provider-Level Feature Compilation

Merged inpatient, outpatient, and beneficiary features into a single provider-level dataset:

provider_df = inpatient_features.merge(outpatient_features, on="Provider", how="outer")

provider_df = provider_df.merge(beneficiary_features, on="Provider", how="left")

provider_df = provider_df.merge(labels[["Provider","target"]], on="Provider", how="left").fillna(0)

Resulting Features:

Feature	Description
Inpatient_Claims	Number of inpatient claims per provider
Inpatient_TotalAmount	Total inpatient claim amount
Outpatient_Claims	Number of outpatient claims
Outpatient_TotalAmount	Total outpatient claim amount
Avg_Patient_Age	Average patient age
Unique_Patients	Count of unique patients
Chronic_Patient_Perc	% patients with chronic conditions
target	Binary fraud label

Preview:

Provider	Inpatient_Claims	Outpatient_Claims	Avg_Patient_Age	Chronic_Patient_Perc	target
P001	123	56	52.3	41.2	1
P002	89	78	45.6	33.1	0

---

7. Saving and Exporting Data

- **Local CSV: provider_features.csv**
- **Google Drive Export:**

provider_df.to_csv(drive_path, index=False)

File Path: /content/drive/MyDrive/provider_features.csv

This file is now ready for machine learning modeling.

1. Introduction

The goal of this stage is to build and evaluate **machine learning models** for predicting healthcare provider fraud using the features engineered in the previous step.

The provider-level dataset contains aggregated information from inpatient, outpatient, and beneficiary data, including claims counts, total claim amounts, patient demographics, and chronic condition statistics.

---

2. Data Preparation

2.1 Loading Provider Features

provider_df = pd.read_csv(drive_path + 'provider_features.csv')

- Features include: Inpatient_Claims, Outpatient_Claims, Avg_Patient_Age, Unique_Patients, Chronic_Patient_Perc
- Target variable: target (1 = Fraud, 0 = Not Fraud)
- Provider ID is dropped as it is not a predictive feature.

2.2 Train-Test Split

- Dataset split: 80% training, 20% testing
- Stratification by target to maintain class distribution:

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

)

2.3 Feature Scaling

- StandardScaler applied to scale numeric features for models sensitive to feature magnitudes (e.g., SVM):

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

---

3. Handling Class Imbalance

Fraud is a **rare event**, causing class imbalance. To address this:

- **Class weights** are computed using compute_class_weight:

```
class_weights_values = compute_class_weight(class_weight='balanced', classes=classes, y=y_train)

class_weights = dict(zip(classes, class_weights_values))

print("Class weights:", class_weights)
```

- Example output:
- Class weights: {0: 0.54, 1: 5.45}

This ensures the model penalizes misclassification of minority (fraud) samples more heavily.

---

4. Model Selection

Four machine learning models were chosen for training:

Model	Key Notes
Decision Tree	Supports class_weight parameter to handle imbalance
Random Forest	Ensemble of decision trees, supports class_weight
Gradient Boosting	Does not support class_weight directly; sample_weight used
SVM	Supports class_weight, requires scaled features, probability estimates enabled

4.1 Training Loop

```
for name, model in models.items():
    if name == "Gradient Boosting":
```

```
    sample_weights = np.array([class_weights[c] for c in y_train])
    model.fit(X_train, y_train, sample_weight=sample_weights)
elif name == "SVM":
    model.fit(X_train_scaled, y_train)
else:
    model.fit(X_train, y_train)
fitted_models[name] = model
```

- **Gradient Boosting**: sample_weight ensures fraud samples are weighted more.
- **SVM**: trained on scaled features with class weights.

---

5. Model Storage (Optional)

Models can be saved using joblib for future predictions:

```
joblib.dump(fitted_models["Random Forest"], drive_path + 'rf_model.pkl')
```

This enables **reusability** without retraining.

1. Introduction

Fraud detection in healthcare claims is crucial for preventing financial losses and ensuring ethical provider behavior. This report presents the **training and evaluation of machine learning models** for predicting fraudulent providers using provider-level aggregated features.

Dataset: provider_features.csv

- Features include inpatient/outpatient claim counts and amounts, beneficiary age, chronic condition percentages, and unique patient counts.
- Target: target (1 = Fraud, 0 = Not Fraud)

---

2. Data Preparation

1. **Load Dataset:**
2. provider_df = pd.read_csv('/content/drive/MyDrive/provider_features.csv')
3. X = provider_df.drop(columns=["Provider","target"])
4. y = provider_df["target"]
5. **Train-Test Split:**

- 80% training, 20% testing
- Stratified by target to maintain class distribution.

6. **Class Imbalance Handling:**
    - Computed scale_pos_weight for XGBoost:
    - scale_pos_weight = neg / pos
    - For Random Forest and Logistic Regression, used class_weight='balanced'.

---

3. XGBoost Model Training and Evaluation

1. **Model Setup:**
2. model = xgb.XGBClassifier(
3. use_label_encoder=False,
4. eval_metric='logloss',
5. scale_pos_weight=scale_pos_weight,
6. random_state=42
7.)
8. **Evaluation Metrics:**
    - **Precision:** 0.5965
    - **Recall:** 0.6733
    - **F1-score:** 0.6326
    - **Confusion Matrix:**
    - [[935 46]
    - [33 68]]
    - **PRAUC:** 0.7200
    - **ROC AUC:** 0.9450
9. **Visualization:**
    - **ROC Curve** shows strong discrimination (AUC ≈ 0.945).
    - **Confusion matrix** confirms reasonable detection of fraudulent providers.

---

4. Hyperparameter Tuning for XGBoost

- **Parameters Tested:**
    - n_estimators: [100, 300, 500, 700]
    - max_depth: [3, 5, 7, 9]
    - learning_rate: [0.01, 0.05, 0.1, 0.2]
- **Top 3 Hyperparameter Combinations (sorted by F1-score):**

n_estimators	max_depth	learning_rate	Precision	Recall	F1-score	PRAUC
500	5	0.2	0.6321	0.6634	0.6473	0.7270
700	9	0.1	0.6500	0.6436	0.6468	0.7265
700	9	0.2	0.6346	0.6535	0.6439	0.7145

- Selected **best XGBoost model**: n_estimators=300, max_depth=7, learning_rate=0.2

---

5. Model Comparison: XGBoost, Random Forest, Logistic Regression

Evaluation Metrics on Test Set:

Model	Precision	Recall	F1-score	PRAUC	ROC AUC
XGBoost	0.6739	0.6139	0.6425	0.7286	0.9436
Logistic Regression	0.4943	0.8515	0.6255	0.7547	0.9566
Random Forest	0.7333	0.5446	0.6250	0.7371	0.9458

Observations:

- **XGBoost:** Balanced performance, strong F1-score, good ROC AUC.
- **Logistic Regression:** Very high recall (detects most fraud), lower precision.
- **Random Forest:** High precision, moderate recall.

Visualizations:

- Confusion matrices and ROC curves plotted for each model.
- PR curves (PRAUC) indicate XGBoost achieves the best trade-off between precision and recall.

---

6. Summary

- **Data Preprocessing:** Handled class imbalance using scale_pos_weight or class_weight.
- **XGBoost hyperparameter tuning** improved F1-score from ~0.632 to 0.647.

- **Model Comparison:**
    - XGBoost provides the **best overall balance** between precision and recall.
    - Logistic Regression is suitable when **recall is more critical**.
    - Random Forest excels when **precision is prioritized**.