### **Step 1: Understand the Problem Statement**

* **Objective**: Detect fraudulent medical claims using machine learning.
* **Goal**: Build a model that can accurately classify claims as **fraudulent** or **genuine**.
* **Business Impact**: Reduce financial losses and improve the efficiency of insurance claim processing.

### **Step 2: Explore and Understand the Dataset**

First check the contents of your uploaded dataset to understand its structure (columns, data types, size, etc.).

### **Dataset Summary**

The dataset contains **10,000 records** and **23 columns** related to medical insurance claims. Key fields include:

1. **Claim Information**:
   * Procedure codes/CPT Code: Medical procedure code.
   * Total amount: Total claim amount.
   * Claim Limits: Maximum allowed claim.
   * Treatment Expenses: Costs of medical treatment.
2. **Provider and Patient Details**:
   * Provider ID, Patient ID, Doctor Name, Doctor Specialization, Hospital Name.
3. **Time-Based Attributes**:
   * Date of Admission, Date of Discharge, Start Date, End Date.
4. **Insurance Details**:
   * Insurance Company Name, Policy Type, Premium Amount.
5. **Fraud Label**:
   * **Fraud History** – This is the target variable (Approved, Rejected, Under Investigation).

### **Step 3: Data Preprocessing Plan**

1. **Handle Missing Values**: Check for null entries and fill/remove them.
2. **Data Cleaning**: Convert dates to datetime format, extract useful features (e.g., length of hospitalization).
3. **Encode Categorical Variables**: Convert non-numeric columns using encoding (e.g., one-hot encoding).
4. **Label Target Variable**: Map the Fraud History to binary values (e.g., Fraud = 1, Non-Fraud = 0).
5. **Feature Scaling**: Standardize numerical columns for better model performance.

### **Step 4: Model Building Plan**

1. **Split Data**: Train-test split (e.g., 70% train, 30% test).
2. **Model Selection**: Use a combination of models for comparison:
   * Logistic Regression (Baseline)
   * Random Forest (Robust for structured data)
   * XGBoost (High performance on tabular data)
3. **Hyperparameter Tuning**: Use GridSearchCV or RandomizedSearch for best parameters.

### **Step 5: Deployment & Final Report 🡪 this has to be done**

1. **Create a Flask or FastAPI App** to serve the model.
2. **Document** the entire pipeline for submission.

## **Step 1: Load and Explore the Dataset**

### **1. Import Required Libraries**

python

CopyEdit

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder, StandardScaler, label\_binarize

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score, roc\_curve, auc, roc\_auc\_score

* **pandas**: Data manipulation and analysis.
* **matplotlib & seaborn**: Data visualization.
* **sklearn**: Machine learning functions for preprocessing, modeling, and evaluation.

### **2. Load the Dataset**

python

CopyEdit

file\_path = 'health\_insurance\_fraud\_large\_dataset\_v1.csv'

df = pd.read\_csv(file\_path)

Loads the CSV file into a **DataFrame** (df), which is a table-like structure to store and manipulate data.

### **3. Explore the Dataset**

python

CopyEdit

print(df.info())

print(df.head())

print(df.describe())

print(df.isnull().sum())

* **df.info()**: Shows column names, data types, and missing values.
* **df.head()**: Displays the first 5 rows of the dataset.
* **df.describe()**: Provides statistical summaries (mean, std, min, max, etc.) for numeric columns.
* **df.isnull().sum()**: Counts missing (NaN) values in each column.

## **Step 2: Visualize the Target Variable**

python

CopyEdit

plt.figure(figsize=(8, 6))

sns.countplot(x='Fraud History', data=df, palette='Set2')

plt.title('Distribution of Fraud History')

plt.xlabel('Fraud Status')

plt.ylabel('Count')

plt.show()

* **countplot**: Displays the frequency of each class in the **Fraud History** column.
* This helps you understand class balance (e.g., how many fraud vs. non-fraud cases).

## **Step 3: Explore Numerical Features**

python

CopyEdit

plt.figure(figsize=(12, 8))

df[['Total amount', 'Claim Limits', 'Premium Amount', 'Treatment Expenses']].hist(bins=30, figsize=(12, 8), layout=(2, 2), color='skyblue')

plt.suptitle('Distribution of Numerical Features')

plt.show()

* **hist()**: Plots histograms for four numeric columns.
* **layout=(2, 2)**: Arranges 4 plots in a 2x2 grid.
* This helps identify skewness, outliers, and value ranges.

## **Step 4: Correlation Heatmap**

python

CopyEdit

plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(numeric\_only=True), annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Correlation Heatmap')

plt.show()

* **corr()**: Computes pairwise correlations between numeric columns.
* **heatmap()**: Visualizes the correlation matrix.
* Helps identify relationships between variables (e.g., whether some features are strongly correlated).

## **Step 5: Data Preprocessing**

### **1. Handle Missing Values**

python

CopyEdit

df.dropna(inplace=True)

* **dropna()**: Removes rows with missing values.
* **inplace=True**: Changes the DataFrame directly (no copy).

**2. Convert Dates & Calculate Length of Stay**

python

CopyEdit

date\_cols = ['Date of Admission', 'Date of Discharge', 'Start Date', 'End Date', 'Hospitalized Date']

for col in date\_cols:

df[col] = pd.to\_datetime(df[col], errors='coerce')

df['Length of Stay'] = (df['Date of Discharge'] - df['Date of Admission']).dt.days

df.drop(date\_cols, axis=1, inplace=True)

* **pd.to\_datetime()**: Converts date columns to datetime format.
* **dt.days**: Computes the length of stay in days.
* Drops the original date columns after feature extraction.

### **3. Encode Categorical Variables**

python

CopyEdit

label\_encoder = LabelEncoder()

df['Fraud History'] = label\_encoder.fit\_transform(df['Fraud History'])

cat\_cols = df.select\_dtypes(include='object').columns

for col in cat\_cols:

df[col] = label\_encoder.fit\_transform(df[col])

* **LabelEncoder**: Converts categorical variables to numbers (e.g., "Yes" → 1, "No" → 0).
* Encodes **Fraud History** and all other object-type columns.

### **4. Scale Numerical Features**

python

CopyEdit

scaler = StandardScaler()

numeric\_cols = ['Total amount', 'Claim Limits', 'Premium Amount', 'Treatment Expenses', 'Length of Stay']

df[numeric\_cols] = scaler.fit\_transform(df[numeric\_cols])

* **StandardScaler**: Scales numerical data to **mean = 0** and **standard deviation = 1**.
* This improves model performance by standardizing different feature scales.

## **Step 6: Model Building**

### **1. Define Features and Target**

python

CopyEdit

X = df.drop(['Fraud History', 'Provider ID', 'Patient ID', 'Doctor Name', 'Hospital Name', 'Policy Number'], axis=1)

y = df['Fraud History']

* **X**: Input features (excludes non-predictive columns).
* **y**: Target variable (**Fraud History**).

### **2. Split Data**

python

CopyEdit

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

* **train\_test\_split()**: Splits the data into 70% training and 30% testing.
* **random\_state**: Ensures reproducibility.

### **3. Train the Model**

python

CopyEdit

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

* **RandomForestClassifier**: An ensemble of decision trees for classification.
* **n\_estimators=100**: Uses 100 trees in the forest.

### **4. Make Predictions**

python

CopyEdit

y\_pred = model.predict(X\_test)

* **predict()**: Generates predictions on the test set.

## **Step 7: Model Evaluation**

### **1. Accuracy**

python

CopyEdit

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

* **accuracy\_score()**: Measures the proportion of correct predictions.

### **2. Confusion Matrix & Classification Report**

python

CopyEdit

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

* **confusion\_matrix()**: Displays true/false positives and negatives.
* **classification\_report()**: Outputs precision, recall, and F1-score.

### **3. ROC Curve & AUC (Multiclass Classification)**

python

CopyEdit

y\_prob = model.predict\_proba(X\_test)

y\_test\_binarized = label\_binarize(y\_test, classes=range(y\_prob.shape[1]))

plt.figure(figsize=(8, 6))

for i in range(y\_prob.shape[1]):

fpr, tpr, \_ = roc\_curve(y\_test\_binarized[:, i], y\_prob[:, i])

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, lw=2, label=f'Class {i} (AUC = {roc\_auc:.2f})')

* **predict\_proba()**: Returns class probabilities.
* **roc\_curve()**: Computes the ROC curve.
* **auc()**: Calculates the Area Under the Curve.