## Provider Fraud Case Study

### Files in Zip

1. CSV with unseen data claim ID matched to potential fraud predictions (1 = “Yes”, 2 = “No”)
2. Code as an IPYNB (Jupyter Notebooks) file (primary code)
3. PDF of IPYNB
4. Code as a PY file
5. Word document on brief reasoning

**Note:** To run the code, please save all datasets in the root folder with no subfolders.

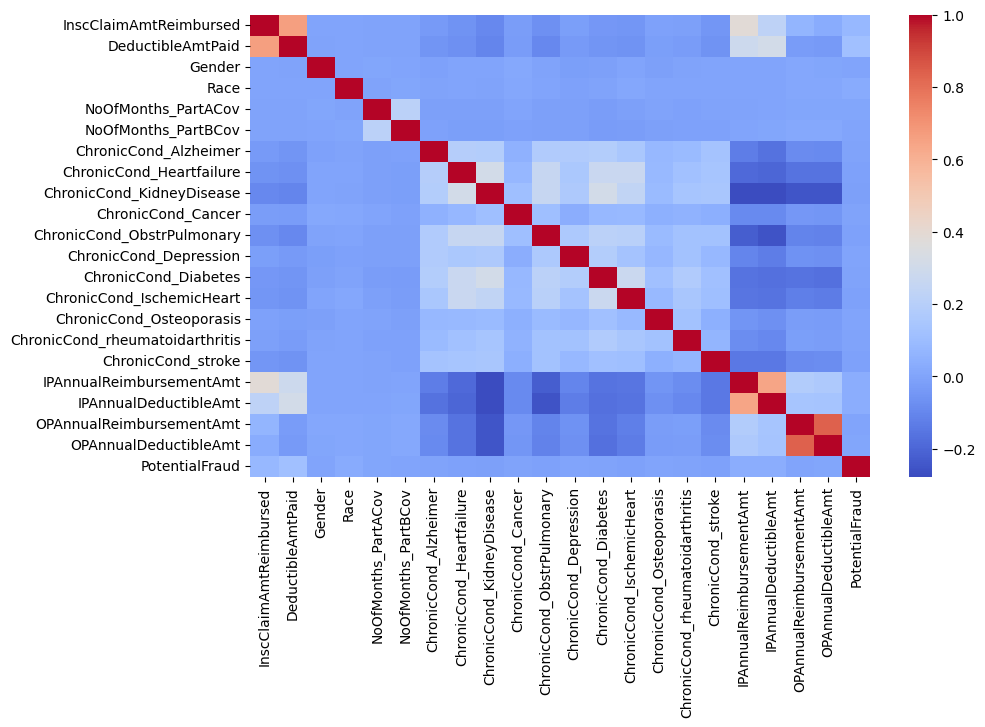
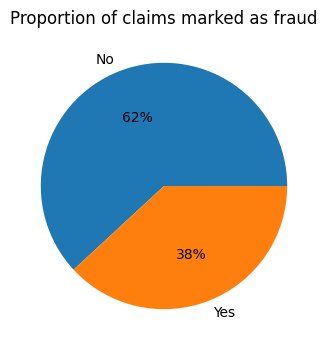
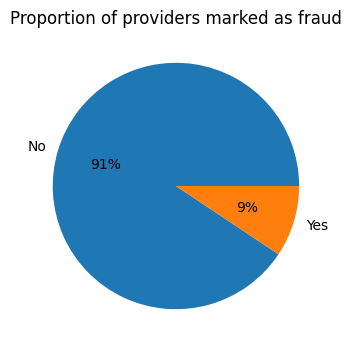
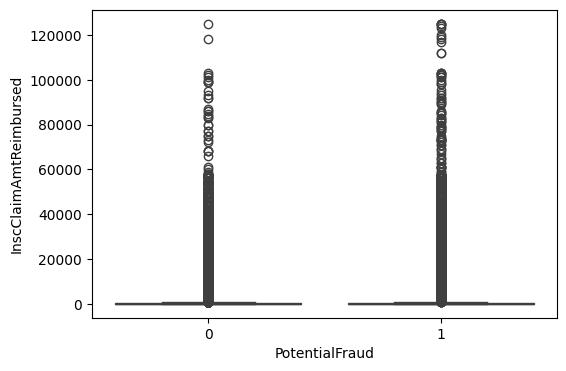
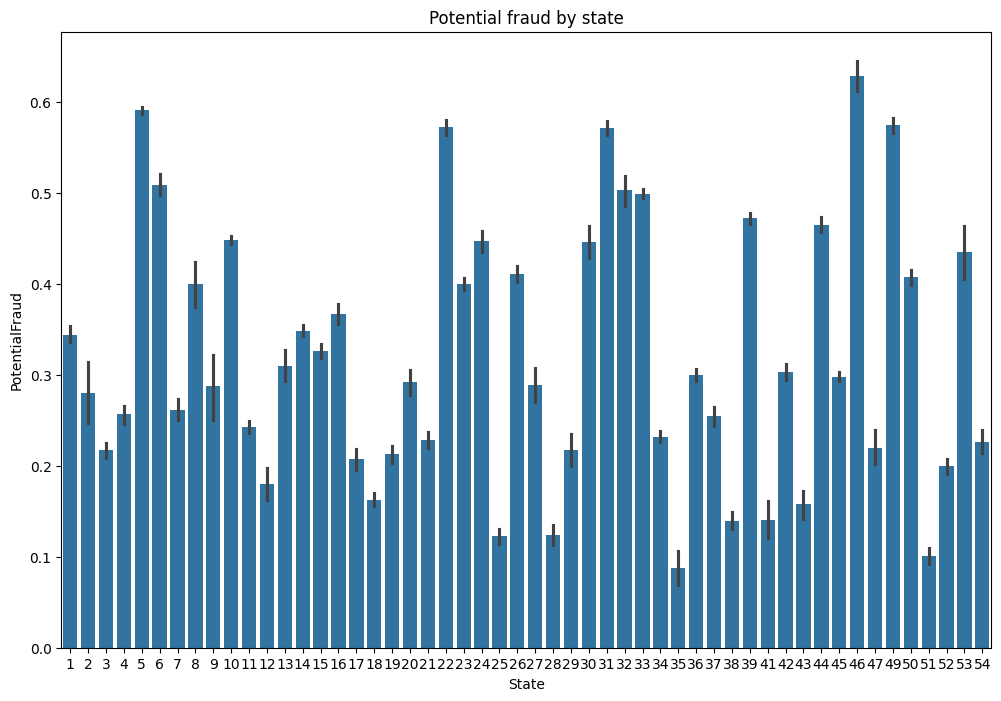
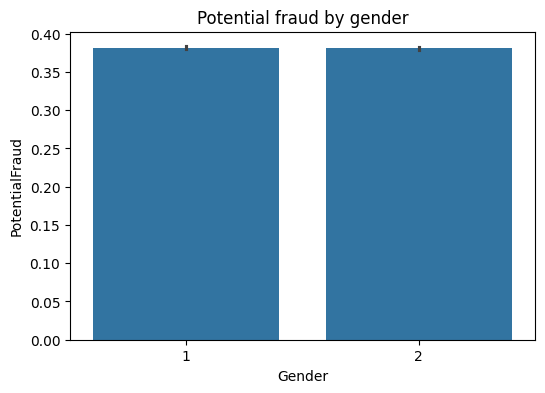
### Introduction

We have four datasets each, for training and testing. The testing data is unlabelled. The label is “PotentialFraud”, and this may have one of two values. “PotentialFraud” is on a per-provider level, not a per-claim level. The rest of the data is on a per-claim level.

We combined all the data, first by stacking inpatient and outpatient data (removing extra inpatient columns, as there are too few inpatients compared to outpatients (around 7% of the data), we cannot extract too much information from these columns), and then by merging with the benefactor and provider information using “BeneID” and “Provider” as keys respectively.

We noticed that all the code columns are essentially categorical variables. There are a few other misleading numeric columns as well, such as the county. We classified all these as strings to mark them as categorical. We will preserve state information as an indicator for geographical location.

### Exploratory Data Analysis

1. There seems to be no significant correlation between any of the numeric columns and the likelihood of fraud. This poses great difficulties to our task, as most of the information will have to be retrieved from categorical columns. Some columns which correlate with each other are the reimbursement amounts and deductible amounts, which makes sense.  
     
     
   
2. In the dataset, the proportion of claims marked as fraud are around 38%, while the number of providers marked as fraud are around 9%. This tells us that the dataset is biased to feature more claims by likely fraudster providers. The goal must then be to try to extract other information such as benefactor, physician, and code information from these fraud claims.  
     
    
3. Most claim amounts are tiny, but there are a lot of outliers using the interquartile range method. So we need a technique that is robust to outliers.  
     
   
4. Likelihood of fraud by state reveals that the state numbered 46 has the highest likelihood of fraud and/or that this state is highly represented in the database.  
     
   
5. There seems to be no correlation between potential fraud and gender.  
     
   

### Data Cleaning and Feature Engineering

1. Using the code and physician columns, which have many categorical values, we need to extract the most relevant information. We decide to one-hot encode the “n”most popular in each of these values, as well as all the values which were flagged as potential frauds. The columns we devise are:

* Count of number of diagnoses per claim
* Count of number of procedures per claim
* Check to see if any admit diagnoses and diagnoses match
* One-hot-encoding of most common admit diagnoses; most common admit diagnoses where fraud was detected
* One-hot-encoding of most common diagnoses; most common diagnoses where fraud was detected
* One-hot-encoding of most common procedures; most common procedures where fraud was detected.
* Count of number of physicians per claim
* Check to see if any of the physicians (attending, operating, other) match
* One-hot-encoding of most common physicians
* One-hot-encoding of most common physicians where fraud was detected, most common physicians where fraud was detected

1. For the date columns, we obtain the following.

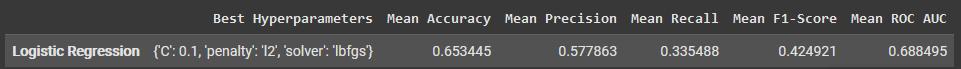
* Age at time claim starts
* Length of claim
* Living or dead

1. All the other columns are dropped if there are too many nulls, or one-hot-encoded if they are categoricals.

### Model Training

Feature scaling is then done using standardisation. PCA is applied to reduce the final number of columns to 100.

Model selection was conducted using grid search for hyperparameter selection, and k-fold (k=5) cross-validation for validation. However, in the end, only logistic regression was picked for performance reasons. If you have a good system, you can attempt to run the entire hyperparameter grid (please uncomment the code).  
  
A lot of models were tried out, but owing to performance limitations only the results of logistic regression are given below.



This model achieves 65.3% accuracy. Other metrics are provided in the code file. Other models like XGBoost can be attempted with a better system and would likely lead to superior results.

The results are attached as a CSV file with claim number as an index.

### Recommendations

With further industry knowledge, it would be possible to potentially engineer more relevant features. It is recommended that better models are explored, such as XGBoost or a neural network. The framework required to implement such code is provided, it just would require a good system to run optimally. The number of features governed by the **n\_common** parameter can also be optimised to run for each block of code.