Diploma Thesis, University of Hyderabad

Project Title: Predict the potentially Fraudulent Providers based on claims filed by them

Name: Praful Batra

Email: praful.batra.pb@gmail.com

Enrollment No:40AIML386-21/1

Content

1 Literature survey & Data Acquisition

1.1 Problem Definition

a) Overview of Healthcare Provider Fraud

Healthcare fraud involves filling of false claims to earn profit. It can be committed by Providers, patients and other individuals who intentionally deceive healthcare system to receive unlawful benefits. The task of this project is to predict potentially fraudulent providers based on claims filled by them. Common types of Healthcare fraud committed by Providers are:

- Double Billing- Submitting multiple claims for same service.
- Phantom Billing-Billing for a service visit or supplies the patient never received.
- Unbundling: Submitting multiple bills for the same service.
- Upcoding: Billing for a more expensive service than the patient actually received.

b) Importance of Solving this Problem

Healthcare fraud is very widespread and statistics now show that 10 cents of every dollar spent on health care goes toward paying for fraudulent health care claims. It is estimated that nearly 60 billion dollars are lost annually due to health care fraud and abuse. It affects both individuals and businesses and it can raise health insurance premiums, expose people to unnecessary medical procedures, and increase taxes.

Insurance companies are most affected by these fraud practices. The main problem lies in the fact that there is huge amount of claims that needs to be processed and companies have very little time to complete the request. As per U.S. legislation, an insurance company should pay a legitimate healthcare claim within 30 days. So, there is very little time to perform an adequate investigation before an insurer has to pay.

If an efficient Provider Fraud detection system is built using advanced Statistical methods and Machine Learning, it will really help insurance companies scrutinize claims thoroughly which in turn will boost economy.

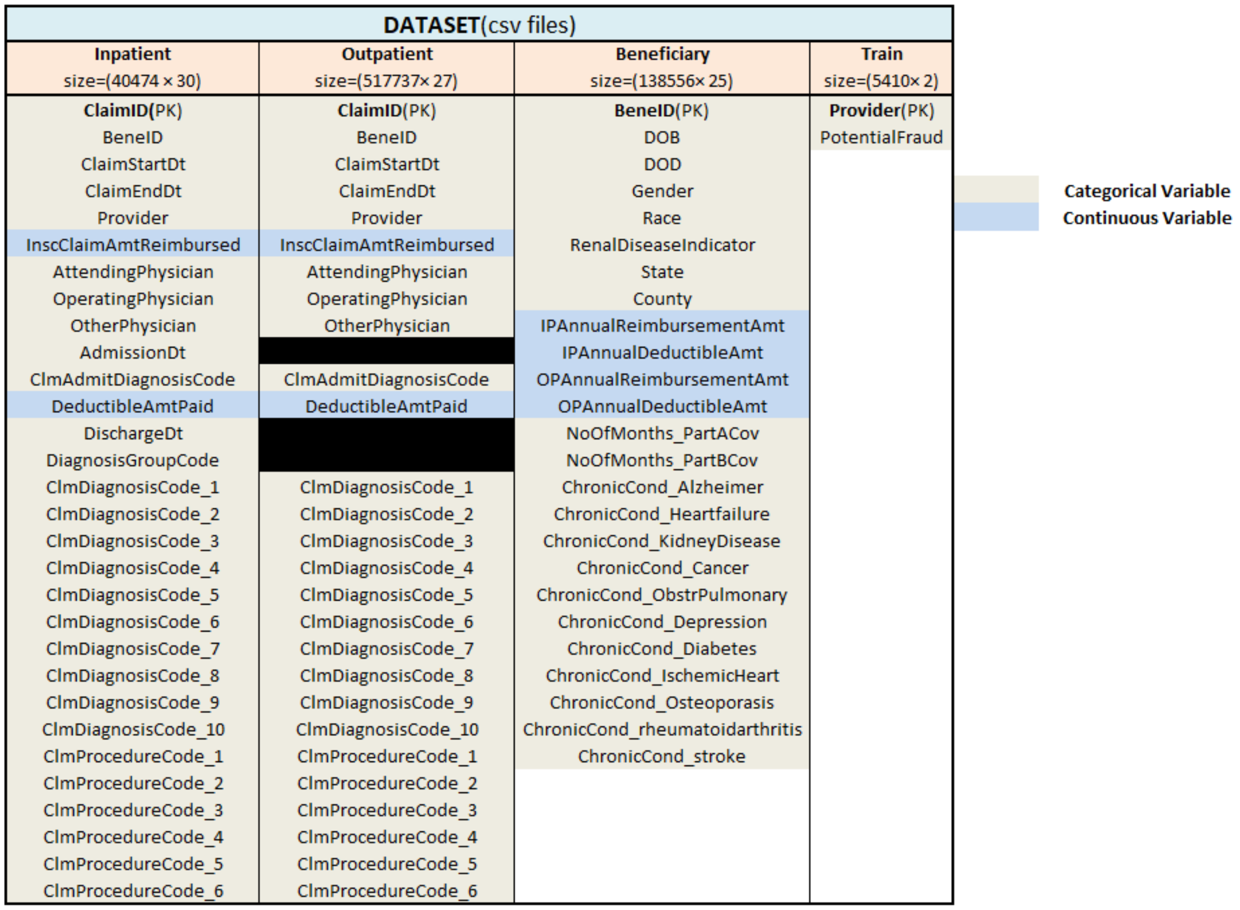
1.2 Dataset

a) Source of the dataset

The dataset open sourced and is available on Kaggle. The exact source of data is not mentioned on Kaggle also due to confidential reasons.

b) Data description

The Dataset is in tabular format and consists of eight csv files which include four files for train and test data each. It consists of Inpatient data, outpatient data Beneficiary details and Train file.



DATASET(csv files)

Inpatient size=(40474 × 30)	**Outpatient** size=(517737× 27)	**Beneficiary** size=(138556× 25)	**Train** size=(5410× 2)
ClaimID(PK)	**ClaimID**(PK)	**BeneID**(PK)	**Provider**(PK)
BeneID	BeneID	DOB	PotentialFraud
ClaimStartDt	ClaimStartDt	DOD	
ClaimEndDt	ClaimEndDt	Gender	
Provider	Provider	Race	
InscClaimAmtReimbursed	InscClaimAmtReimbursed	RenalDiseaseIndicator	
AttendingPhysician	AttendingPhysician	State	
OperatingPhysician	OperatingPhysician	County	
OtherPhysician	OtherPhysician	IPAnnualReimbursementAmt	
AdmissionDt		IPAnnualDeductibleAmt	
ClmAdmitDiagnosisCode	ClmAdmitDiagnosisCode	OPAnnualReimbursementAmt	
DeductibleAmtPaid	DeductibleAmtPaid	OPAnnualDeductibleAmt	
DischargeDt		NoOfMonths_PartACov	
DiagnosisGroupCode		NoOfMonths_PartBCov	
ClmDiagnosisCode_1	ClmDiagnosisCode_1	ChronicCond_Alzheimer	
ClmDiagnosisCode_2	ClmDiagnosisCode_2	ChronicCond_Heartfailure	
ClmDiagnosisCode_3	ClmDiagnosisCode_3	ChronicCond_KidneyDisease	
ClmDiagnosisCode_4	ClmDiagnosisCode_4	ChronicCond_Cancer	
ClmDiagnosisCode_5	ClmDiagnosisCode_5	ChronicCond_ObstrPulmonary	
ClmDiagnosisCode_6	ClmDiagnosisCode_6	ChronicCond_Depression	
ClmDiagnosisCode_7	ClmDiagnosisCode_7	ChronicCond_Diabetes	
ClmDiagnosisCode_8	ClmDiagnosisCode_8	ChronicCond_IschemicHeart	
ClmDiagnosisCode_9	ClmDiagnosisCode_9	ChronicCond_Osteoporasis	
ClmDiagnosisCode_10	ClmDiagnosisCode_10	ChronicCond_rheumatoidarthritis	
ClmProcedureCode_1	ClmProcedureCode_1	ChronicCond_stroke	
ClmProcedureCode_2	ClmProcedureCode_2		
ClmProcedureCode_3	ClmProcedureCode_3		
ClmProcedureCode_4	ClmProcedureCode_4		
ClmProcedureCode_5	ClmProcedureCode_5		
ClmProcedureCode_6	ClmProcedureCode_6		

▨	**Categorical Variable**
🟦	**Continuous Variable**

Inpatient and **Outpatient** contains the claim data of patients. Outpatients are the patients who only visited the hospital for treatment and didn't get admitted while Inpatient are the patients who got admitted in the hospital for treatment and they have 3 extra columns other than the outpatient table (Admission Date, Discharge Date and Diagnosis Group code)

Most of the columns are self explanatory by their names in above image (Claim Id, Beneficiary Id, Provider, Attending Physician, Claim Start Date, Claim End Date, Other Physician etc.)
Other features are Claim Diagnosis Codes and Claim Procedure codes.

Providers that bill Medicare use codes for patient diagnoses and codes for care, equipment, and medications provided. Procedure code is used to identify what was done to or given to a patient (surgeries, durable medical equipment, medications, etc.) while Diagnosis codes are used for the disease that patient was diagnosed.

Train dataset contains unique Providers and their corresponding label which tells whether provider is Fraud or not Fraud.

Beneficiary table contains details about the patient (Beneficiary Id, Date Of Birth, Date of Death, Gender, State, Country, Various Disease indicators, Reimbursement and Deductible details).

c) Data Size and any challenges
The data size is not very large and each file contains few thousands rows only and maximum of 20-30 columns. For the test data, there are no labels present for providers and train dataset will be used for all the analysis and validations. Also, it is an imbalanced data only 9.35% of positive labels out of total labeled providers and when we use these for all training, cross validation and test, the positive points will get further reduced for each stage. We can't use under sampling techniques as size of labeled points is very small. We will need to use oversampling techniques to handle imbalanced data.

There is one problem with this data, it doesn't fully replicate real world scenario as labels are given at provider level and not at the claim level. In reality, it may not happen that all the claims belonging to one provider are either fraud or non-fraud. But only a fraction of them can be fraudulent for each provider.

d) Tools to process data
Since dataset is already present in csv format. Pandas will be used to read and process the data. It can be easily processed using numpy and pandas.

1.3 Key Performance Indicators.

a) Business Constraint

In our problem, both False Positives and False Negatives are important and both values should be low. If we take Fraudulent as Negative class and Not Fraudulent as Positive class then False Positive is when fraud provider is predicted as Not Fraud. False Positive should be less as we want our model to catch all Fraud providers as they cause huge loss to insurance company. On the other hand, False Negative is when a Non-Fraud provider is predicted as Fraud. If False Negative is high, it will lead to increase in money spent on manual investigation and also, it can put company's reputation at risk if genuine providers are blacklisted. So we need some metric that considers both FP and FN.

b) Performance metric

We will be using MCC (Mathews Correlation Coefficient) as the performance metric. It measures the correlation b/w actual and the predicted values.
and ranges in the interval [−1,+1], with extreme values –1 and +1 reached in case of perfect misclassification and perfect classification, respectively, while MCC=0 is the expected value for the coin random classifier.
Its formula is

$$\mathrm{MCC} = \frac{\mathrm{TP}.\mathrm{TN} - \mathrm{FP}.FN}{\sqrt{(\mathrm{TP}+\mathrm{FP}).(\mathrm{TP}+\mathrm{FN}).(\mathrm{TN}+\mathrm{FP}).(\mathrm{TN}+\mathrm{FN})}}$$

Apart from MCC, we will be checking False Positives and False Negatives separately also.

c) Pros and Cons of preferred Metric

Pros

MCC is perfectly symmetric and it gives equal weight to all the classes. Also, if we switch the positive and negative labels we still get the same value.
MCC takes into account all four values in the confusion matrix. It is highly interpretable and its results and improvements can be easily communicated to Business Stakeholders.

Cons

Since MCC gives equal weight to all 4 values, it can't be used in situation where we need to prioritize one value over other. If we are asked to give more importance to False Positives in this case, we can't implement this in MCC.

Also, we can get division by zero error if any one row or column values are zero and we should be careful to use it when a dumb model is being tested.

d) Applications in Data Science

MCC is used as performance both in binary and multi-classification problems with some change in the formula. it works well on imbalanced datasets as well. Also, It's implementation is available in scikit-learn.

e) Alternative Metrics

- F beta-score is the weighted harmonic mean of precision and recall. It can be used as it considers both False Positives and False Negatives in its calculation. But it doesn't work well with imbalanced data especially when majority class is taken as positive class label. It gives high value even if minority class is predicted with very less accuracy.

$$\text{F-beta} = \frac{(1+\beta^2)*\text{precision}*\text{recall}}{\beta^2*\text{precision}+\text{recall}} \qquad \text{For F1-score, beta=1}$$

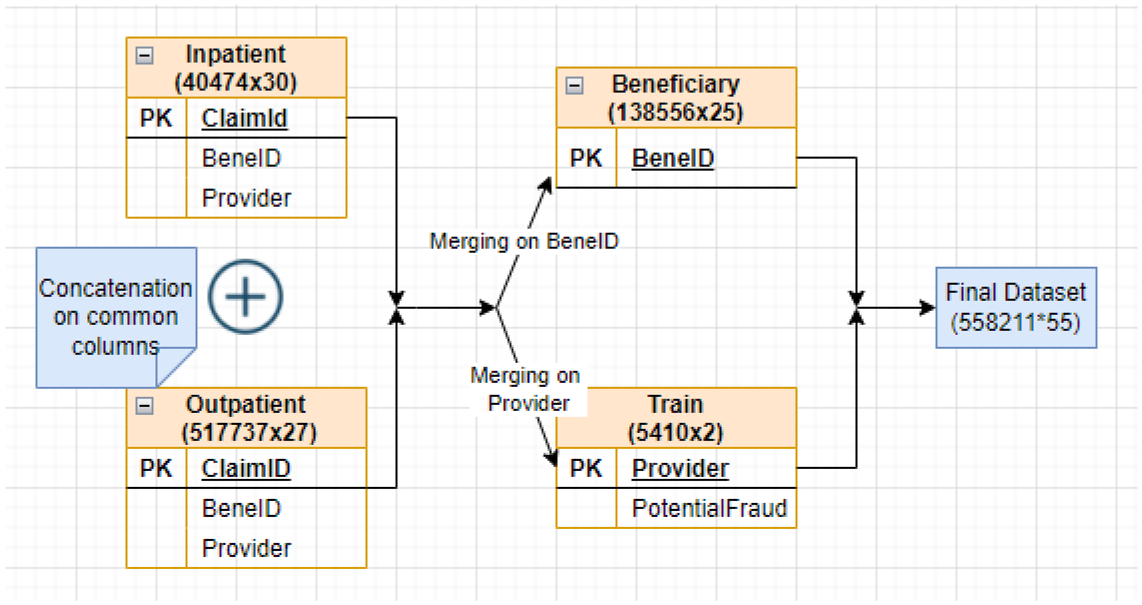
- Accuracy – It can't be used as we have imbalanced data and even a no-skill or dumb model can give high accuracy also when data is imbalanced.

1.4 Mapping to ML problem

Given Business problem can be mapped to a binary classification task in Machine Learning wherein we need to predict whether the provider is fraud or not given all the claims data of inpatients, outpatients and beneficiary details.

a) Aggregating all Tables

First, we create a common table using all four files and use it for further analysis. Inpatient and outpatient will be concatenated on common columns and a new column will be added having binary values for inpatient and outpatients. Then table will be merged with Provider and Beneficiary table on provider and BeneID respectively.



In the final Dataframe, we will remove columns which have too many NaN values and also modify other columns before doing the exploratory data analysis.

b) Solution Approaches

From the feature engineering perspective, we can do univariate and bivariate analysis with the output variable and along with that we can do feature groupings of numerical features to arrive at the important features that can help us find insights about fraudulent claims.

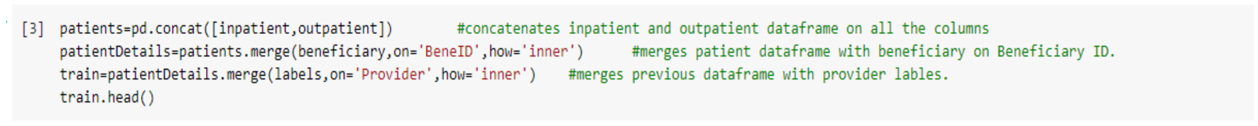
Since this a binary classification problem, various classification algorithms can be used such as Logistics Regression, Random Forests and other Ensembles. Along with that unsupervised machine learning and deep learning techniques such as K-means and Auto Encoders respectively can be used as anamoly detection mechanisms.

2.Exploratory Data Analysis (EDA)

2.1 Merging the datasets

We have four datasets in our training set namely; beneficiary, inpatients, outpatients and labels for Provider. First, we have merged all the datasets to form one training dataset and we will do further analysis on it.

Ref: Cell 3 in notebook

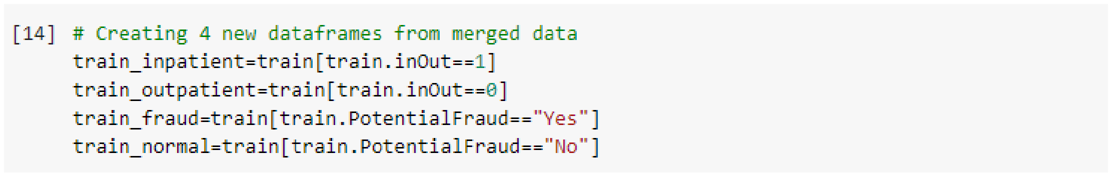


```
[3]  patients=pd.concat([inpatient,outpatient])        #concatenates inpatient and outpatient dataframe on all the columns
     patientDetails=patients.merge(beneficiary,on='BeneID',how='inner')      #merges patient dataframe with beneficiary on Beneficiary ID.
     train=patientDetails.merge(labels,on='Provider',how='inner')    #merges previous dataframe with provider lables.
     train.head()
```

- First, we concatenate inpatients and outpatients on all columns.
- Then, we merge above dataset with beneficiary on Beneficiary ID using inner join and we get same rows as first step indicating all the beneficiary Ids in concatenated dataset are present in Beneficiary dataset.
- After that, we merge on Provider with labels dataset to get final shape as (558211, 55) of our Merged dataset.

2.2 Data Preprocessing

- All the Date columns were converted from String data type to Datetime data type. (ClaimStartDt, ClaimEndDt, AdmissionDt, DischargeDt, DOB, DOD) – *Cell 6*
- Added two new columns 'noDaysAdmit' and 'noDaysClaim' which represents the duration of claim and hospitalization in days. For outpatients, noDaysAdmit is taken as zero. *–Cell 7*
- Added new column 'age' which is calculated by subtracting Date of Birth from Claim Start Date. *–Cell 8*
- Added new column 'whetherAlive' whose value is 0 if patient is dead and 1 if he is alive. *–Cell 9*
- Added new column ClaimMinusAdmitDays which is difference b/w claim and hospitalization duration. *–Cell 12*
- Replaced 2 with 0 in chronic disease indicator columns and Gender.
  Zero indicates the person doesn't have that disease and 1 means patient has that disease *–Cell 11*
- Creating 4 new Dataframes for inpatients, outpatients, fraudulent and non-fraudulent claims.

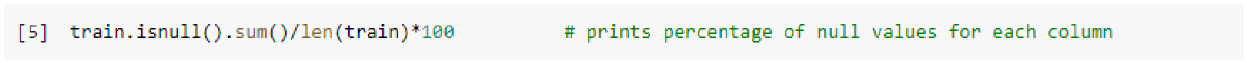


```
[14] # Creating 4 new dataframes from merged data
     train_inpatient=train[train.inOut==1]
     train_outpatient=train[train.inOut==0]
     train_fraud=train[train.PotentialFraud=="Yes"]
     train_normal=train[train.PotentialFraud=="No"]
```

2.3 Basic Exploratory Data Analysis

- For univariate analysis, we are plotting the distribution of different variables with and without response variables. This we can understand feature importance individually. If the proportion if variable is different in both cases, then it can be considered as an important feature.
- If the no. of categories is few, we can understand by Grouped Bar chart and if the categories are large, we can plot Stacked Bar chart for percentages of Fraud and Non-Fraud for each variable.
- In addition to this, analysis of features will be done on overall, inpatients and outpatients to see the trend across all three.

a) Percentage of Null Values

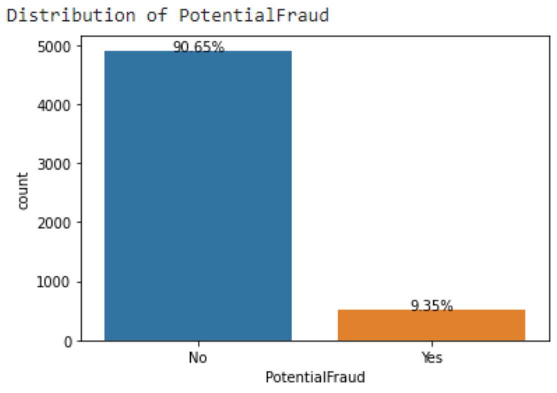


```
[5]  train.isnull().sum()/len(train)*100          # prints percentage of null values for each column
```

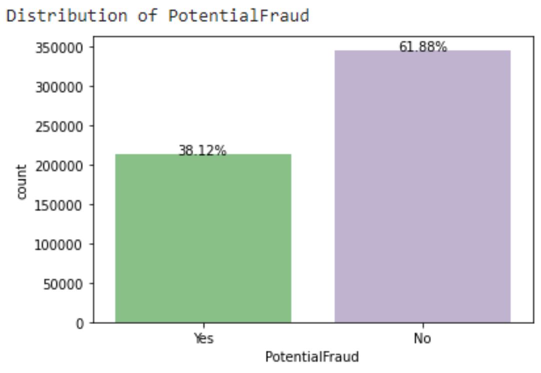
Cell 5:
13 columns out of 55 have more than 80% of Null Values which mostly includes Claim Procedure and Claim Diagnosis codes and almost 30 columns have 0% null values.

b) Distribution of Response Variable

Refer Cell: 21

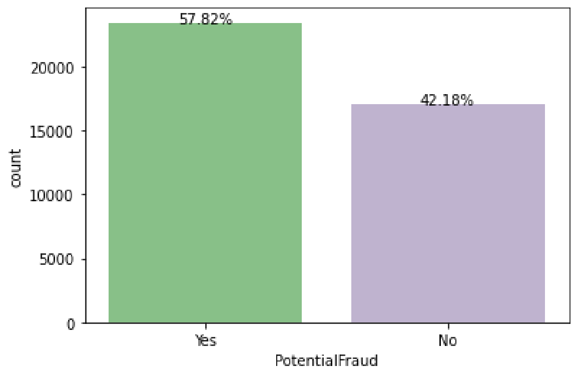


-Percentages in Labels data

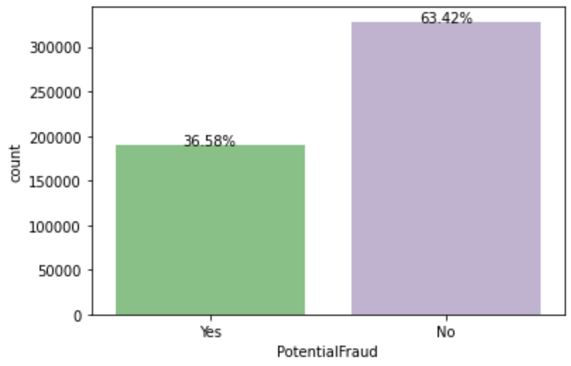


-Percentages in Merged dataset

Observation:
In the original lables data only 9.35% of providers were fraud but after merging data at the claim level, 38.12% of claims are fraud. That means, even though fraudulent providers are less, they have huge number of claims associated with them.



Distribution of PotentialFraud in inpatient data.



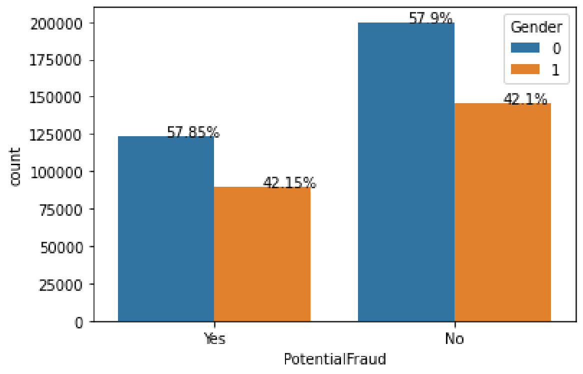
Distribution of PotentialFraud in outpatient data.

Observation: Inpatients have more fraudulent cases as compared to outpatients Inpatients have more options to make false claims in medicines and procedures. Thus , inOut is a very important feature in classification.

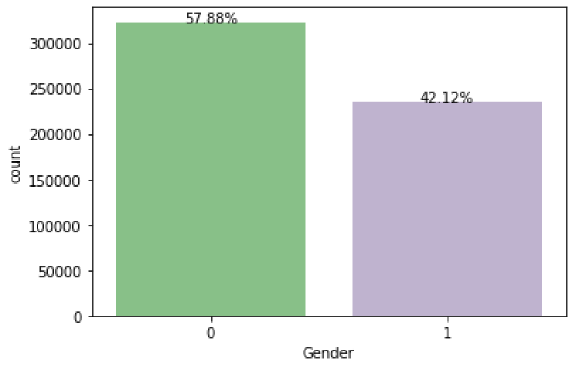
c) Distribution of Gender across Response variable

Cell 19: The proportion of Gender was calculated with and without response variable. This was done to understand Gender has any impact on the response variable.

Distribution of Gender across PotentialFraud



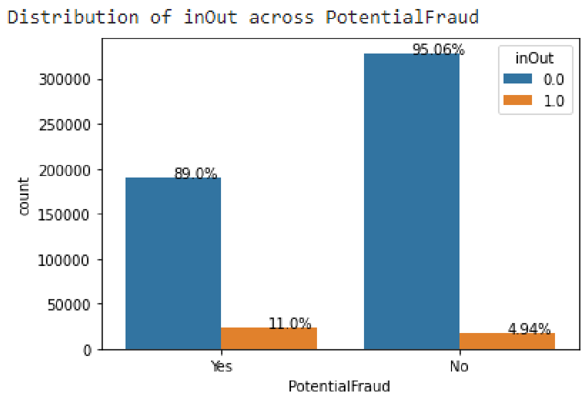
Distribution of Gender

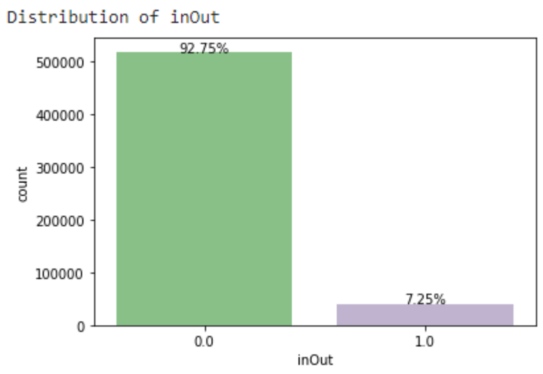


Observation: The Gender ratio is almost similar in both Fraud and Non-Fraud cases. That means, both males and females patient claims are equally probable to be fraudulent, so Gender is not a useful feature in fraud classification.

d) Distribution of inOut feature which indicates whether patient is inpatient is outpatient across response variable.

0- outpatient , 1-inpatient



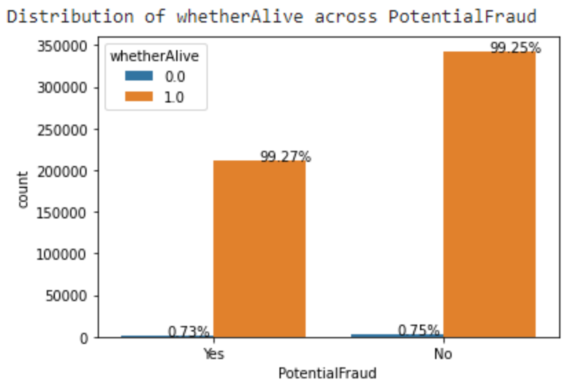


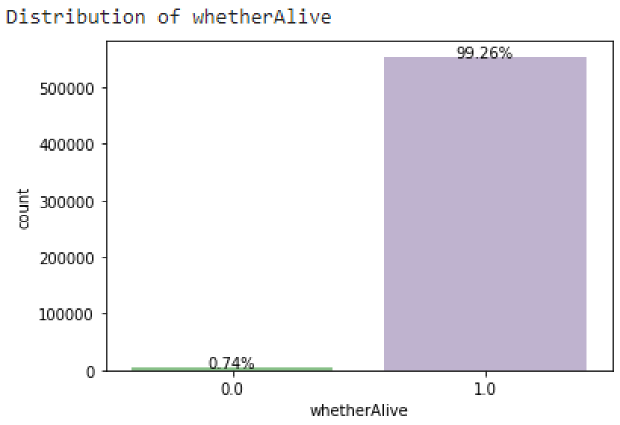
Observation:

Inpatients are only 7.25% out of total patients but in the fraudulent activities, this percent age goes upto 11% if we look only at the fraudulent cases. Hence, inOut can be conside red as one of the features in modelling.

e) Distribution of whetherAlive across Response variable.

Ref- Cell 22

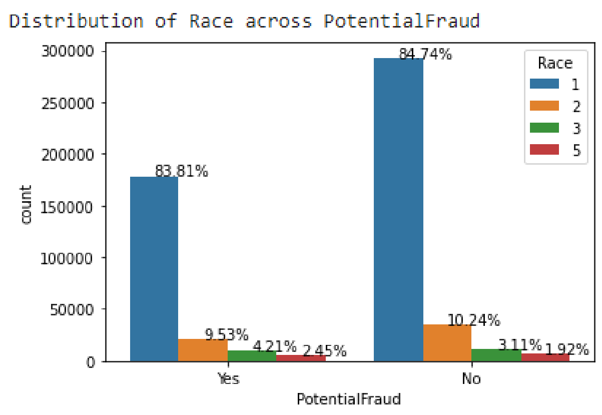


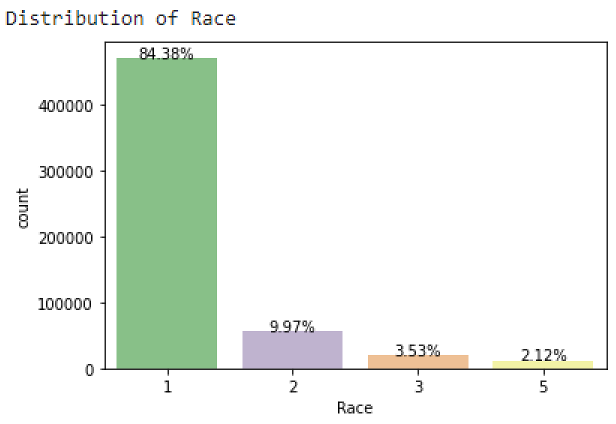


Observation:

There is 0.01% reduction in death patient's percentage for fraudulent claims,
that means there is slightly more chance for alive patients to have fraud claim than dead ones. But it is not that significant to be considered as an important feature.

f) Distribution of Race across Response Variable



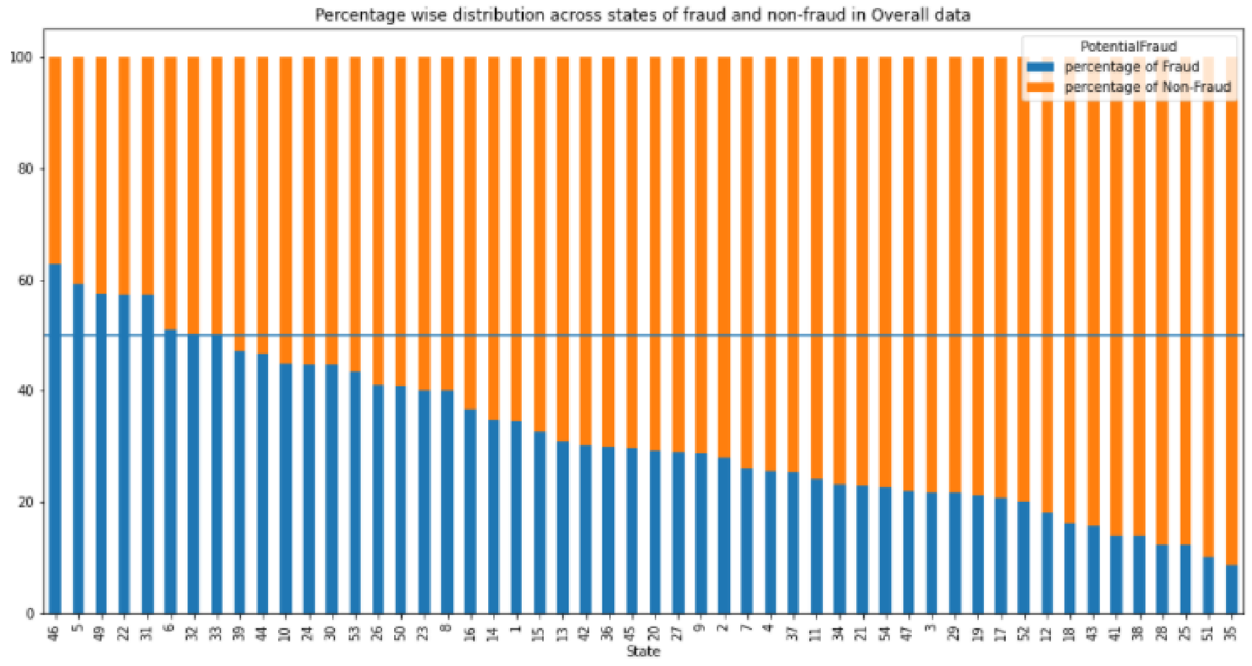


Observation: There is a difference b/w proportion of races in Fraudulent and non-fraud claims especially Race 3 & Race 5 as their percentages increased in fraud claims. So, Race can be considered as a good feature in classification.

g) Proportion of Response variable across states in overall, inpatient and outpatient data.

Ref- Cell 26,27,28.

y-axis represents percentage of fraudulent and non-fraudulent claims.

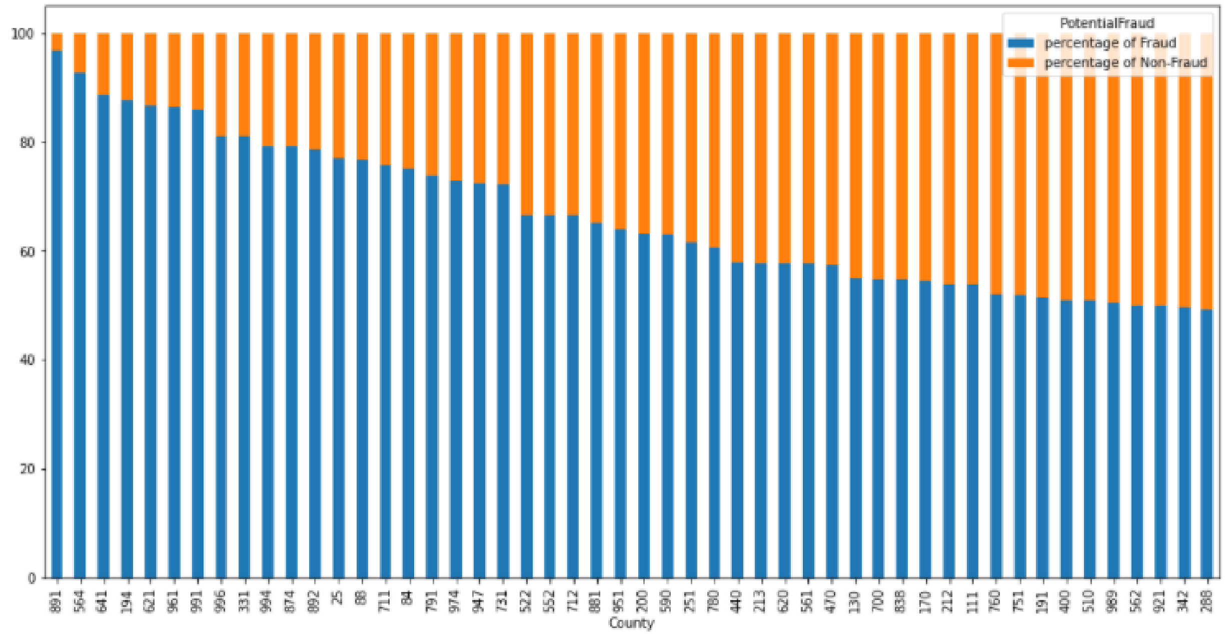


Same plot was done on inpatient and outpatient data also.

Observation: Many states have higher percentage of Fraudulent cases which include state 46,5,49,22,31 in overall data. States 6,7,30,21,36, for inpatient data and state 46,5,49,31,22 for outpatient data.

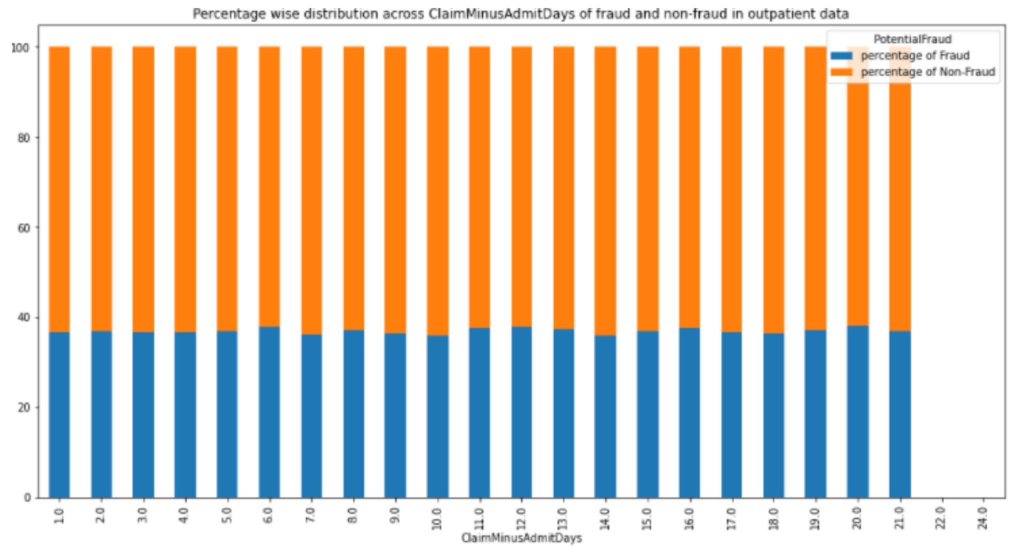
h) Proportion of Response variable across states in overall , inpatient and outpatient data.

Ref –Cell 30



Observation:
USA has total 3200 county but in our dataset, we have around 314 unique county in our dataset and out of 314, we have plotted only 50 of them. From the plot, we can see that there are 9 county where more than 80% of their claims as fraud. Thus, if a claim is co ming from these geograhical regions, there is more chance of being it a fraudulent one.

i) **ClaimMinusAdmitDays-** It is the difference b/w claim and hospitalization duration and it was observed that it is not an important feature in classification as for outpatients it has similar behaviour for all the values and for inpatients, it is highly imbalanced on 1 value and no conclusion can be made from that.



j) ‘NoOfMonths_PartACov’ and ‘NoOfMonths_PartBCov’ with response variable.

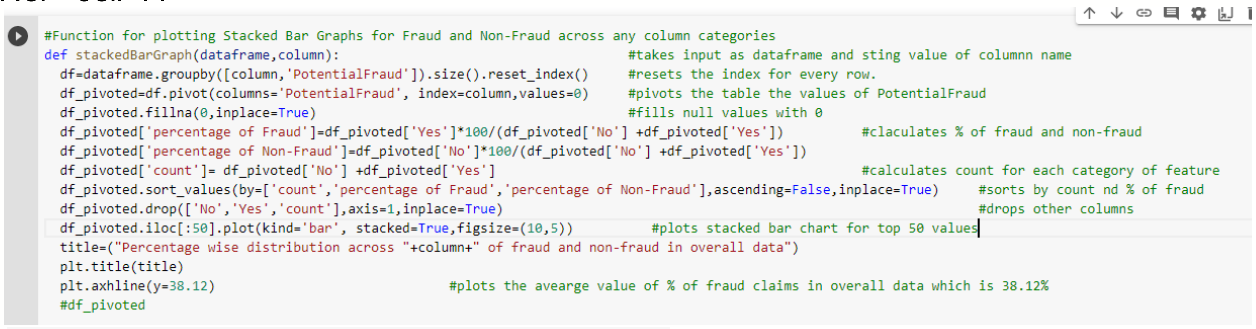
Ref Cell – 38-43.

It was observed that values 4,8,0,12,7 have higher percentage of fraudulent claims in inpatient data. Thus, these two can be considered as good features for classification.

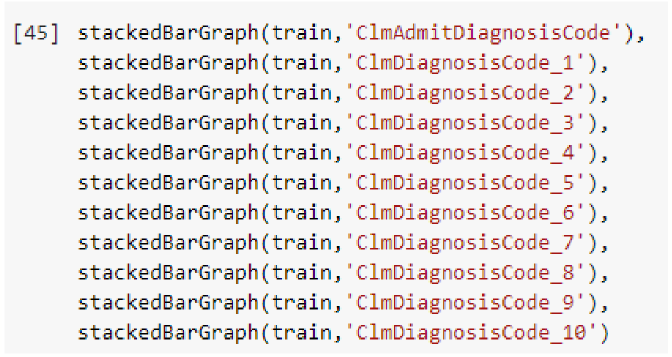
k) Claim Diagnosis Codes Analysis

A function was written which takes input as dataframe and column name and returns stacked bar plot with top 50 values having maximum count having percentage of fraud and non-fraud.

Ref –cell 44



```
#Function for plotting Stacked Bar Graphs for Fraud and Non-Fraud across any column categories
def stackedBarGraph(dataframe,column):                                         #takes input as dataframe and sting value of columnn name
  df=dataframe.groupby([column,'PotentialFraud']).size().reset_index()      #resets the index for every row.
  df_pivoted=df.pivot(columns='PotentialFraud', index=column,values=0)      #pivots the table the values of PotentialFraud
  df_pivoted.fillna(0,inplace=True)                                         #fills null values with 0
  df_pivoted['percentage of Fraud']=df_pivoted['Yes']*100/(df_pivoted['No'] +df_pivoted['Yes'])           #claculates % of fraud and non-fraud
  df_pivoted['percentage of Non-Fraud']=df_pivoted['No']*100/(df_pivoted['No'] +df_pivoted['Yes'])
  df_pivoted['count']= df_pivoted['No'] +df_pivoted['Yes']                                                #calculates count for each category of feature
  df_pivoted.sort_values(by=['count','percentage of Fraud','percentage of Non-Fraud'],ascending=False,inplace=True)      #sorts by count nd % of fraud
  df_pivoted.drop(['No','Yes','count'],axis=1,inplace=True)                                                             #drops other columns
  df_pivoted.iloc[:50].plot(kind='bar', stacked=True,figsize=(10,5))           #plots stacked bar chart for top 50 values
  title=("Percentage wise distribution across "+column+" of fraud and non-fraud in overall data")
  plt.title(title)
  plt.axhline(y=38.12)                                  #plots the avearge value of % of fraud claims in overall data which is 38.12%
  #df_pivoted
```

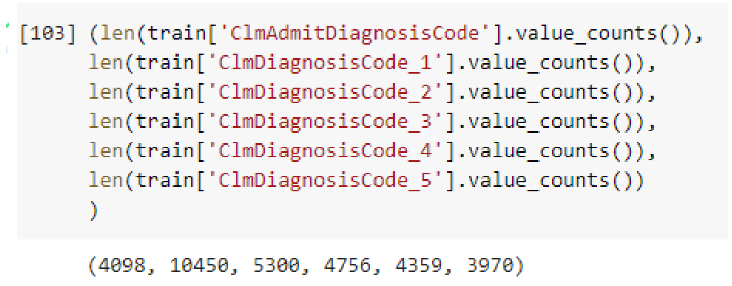


```
[45] stackedBarGraph(train,'ClmAdmitDiagnosisCode'),
     stackedBarGraph(train,'ClmDiagnosisCode_1'),
     stackedBarGraph(train,'ClmDiagnosisCode_2'),
     stackedBarGraph(train,'ClmDiagnosisCode_3'),
     stackedBarGraph(train,'ClmDiagnosisCode_4'),
     stackedBarGraph(train,'ClmDiagnosisCode_5'),
     stackedBarGraph(train,'ClmDiagnosisCode_6'),
     stackedBarGraph(train,'ClmDiagnosisCode_7'),
     stackedBarGraph(train,'ClmDiagnosisCode_8'),
     stackedBarGraph(train,'ClmDiagnosisCode_9'),
     stackedBarGraph(train,'ClmDiagnosisCode_10')
```

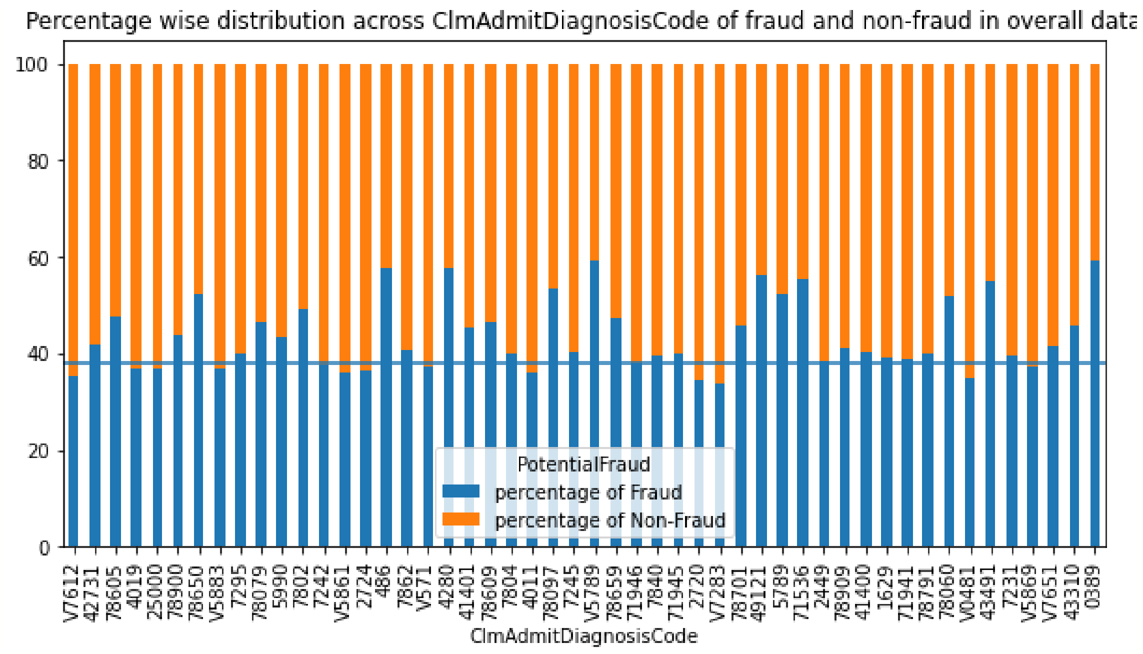
Ref –Cell 45

Observation: The stacked Bar charts were plotted for all the Claim Diagnosis codes and it was observed that there is huge variance in percentage of Fraudulent and Non-Fraudulent claims across various categories of codes and this is an important to consider in further stages. But, there are huge number of categories across each diagnosis codes and these many will be difficult to encode while modelling.

- ClmDiagnosisCode_1 has around 10000 categories
- ClmAdmitDiagnosisCode has around 4000 categories.



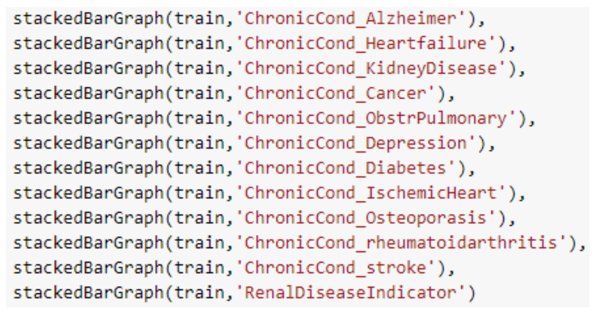
```
[103] (len(train['ClmAdmitDiagnosisCode'].value_counts()),
      len(train['ClmDiagnosisCode_1'].value_counts()),
      len(train['ClmDiagnosisCode_2'].value_counts()),
      len(train['ClmDiagnosisCode_3'].value_counts()),
      len(train['ClmDiagnosisCode_4'].value_counts()),
      len(train['ClmDiagnosisCode_5'].value_counts())
      )

      (4098, 10450, 5300, 4756, 4359, 3970)
```



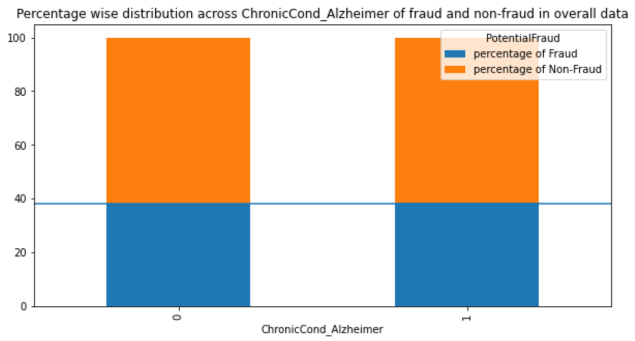
Note: Similar operation was done on Claim Procedure codes and same observations were made.

l) Chronic Disease Indicators

Ref Cell - 47

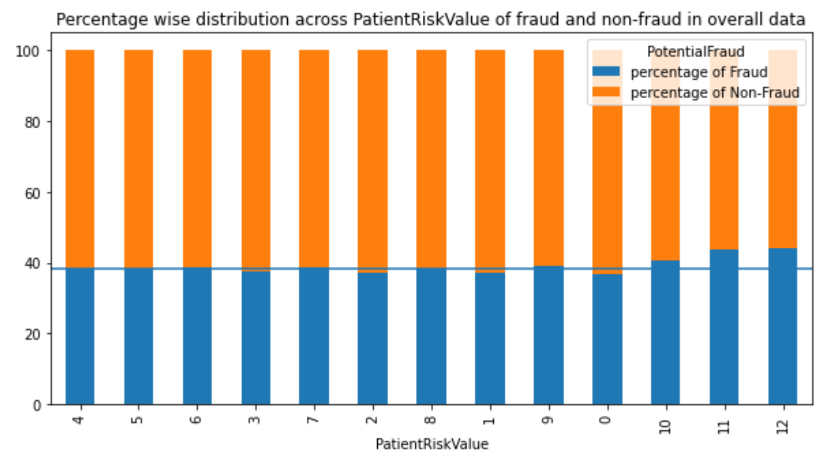


```
stackedBarGraph(train,'ChronicCond_Alzheimer'),
stackedBarGraph(train,'ChronicCond_Heartfailure'),
stackedBarGraph(train,'ChronicCond_KidneyDisease'),
stackedBarGraph(train,'ChronicCond_Cancer'),
stackedBarGraph(train,'ChronicCond_ObstrPulmonary'),
stackedBarGraph(train,'ChronicCond_Depression'),
stackedBarGraph(train,'ChronicCond_Diabetes'),
stackedBarGraph(train,'ChronicCond_IschemicHeart'),
stackedBarGraph(train,'ChronicCond_Osteoporasis'),
stackedBarGraph(train,'ChronicCond_rheumatoidarthritis'),
stackedBarGraph(train,'ChronicCond_stroke'),
stackedBarGraph(train,'RenalDiseaseIndicator')
```



Observation: In the Distribution for all these disease indicators, the percentage of fraudulent claims was almost equal to overall average of 38.12% and these don't add any value to the classification task. We will combine them into a single number for representation. We will add all these scores and create a new column 'PatientRiskValue'.

```
stackedBarGraph(train,'PatientRiskValue')
```



Observation:
We can see that patients with risk value 11 & 12 have more chance to be classified as F raud as compared to other values.

k) Analysis of Numerical features

Ref: Cell 51-65

- KDE plots of IPAnnualReimbursementAmt , IPAnnualDeductibleAmt , OPAnnualReimbursementAmt , OPAnnualDeductibleAmt , InscClaimAmtReibusement , DeductibleAmtPaid were drawn for inpatient , outpatient and overall data. It was observed that density of Fraudulent values were more in Inpatient data as compared to Overall and outpatient data for all these features.

- The deductible paid by all inpatients is 1068.0 excluding the Nan values. It may be the minimum amount to be paid when a patient is required to be admitted in a hospital in USA.

```
[67] #sns.displot(data=train_inpatient, x="DeductibleAmtPaid", hue="PotentialFraud")
     train_inpatient.DeductibleAmtPaid.value_counts()

     1068.0    39575
     Name: DeductibleAmtPaid, dtype: int64
```

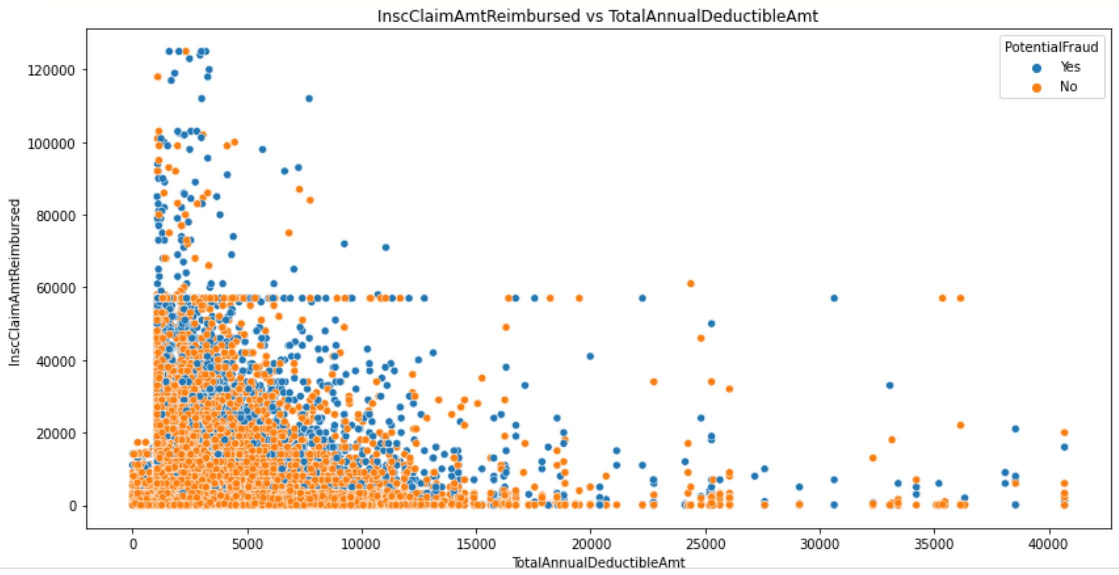
- IPAnnualReimbursementAmt , OPAnnualReimbursementAmt , InscClaimAmtReibusement have right skewed distributions.
- IPAnnualDeductibleAmt has distribution with multiple peaks and that may depend on no of days patient is admit while OPAnnualDeductibleAmt has more bell shaped distribution.

2.3.2 Bivariate analysis

Scatter plots were plotted among various numerical features (IPAnnualReimbursementAmt, IPAnnualDeductibleAmt, OPAnnualReimbursementAmt ,OPAnnualDeductibleAmt , InscClaimAmtReibusement, DeductibleAmtPaid) and their trends were observed. (Cell-69 to 75).

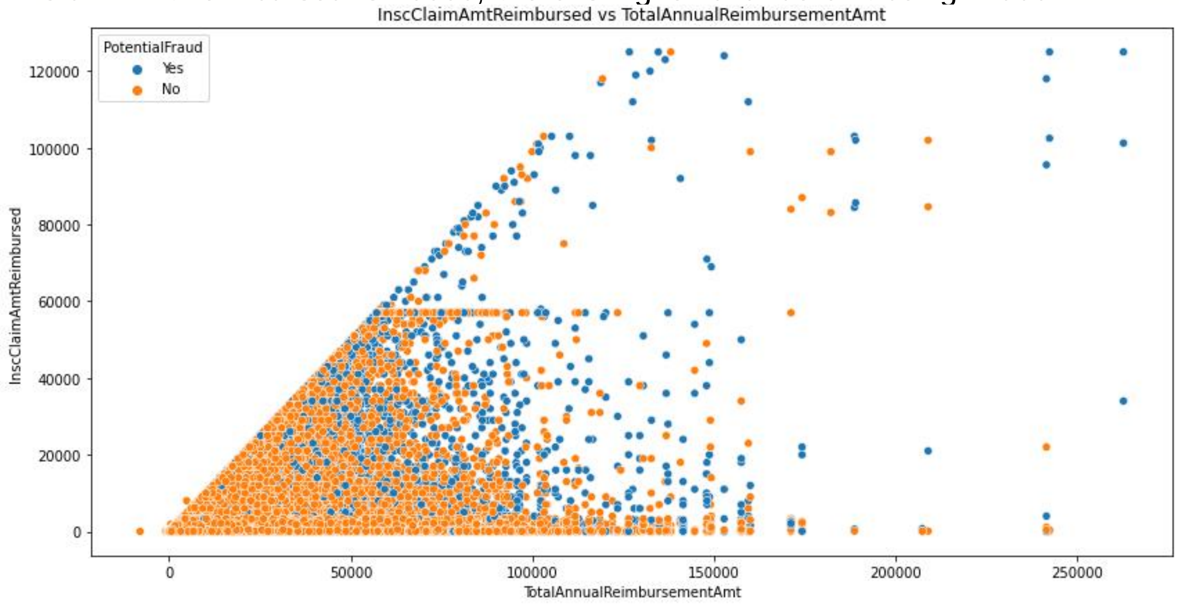
- **Observation**: **TotalAnnualDeductible vs InscClaimAmtReimbursed**

  If the InscClaimAmtReimbursed is more than 60000, there is more chance of Fraudulent transaction. Also, if InscClaimAmtReimbursed is b/w 20000 to 60000 and TotalAnnualDeductibleAmt is greater than 100000, then also, more chance of Fraudulent is there.



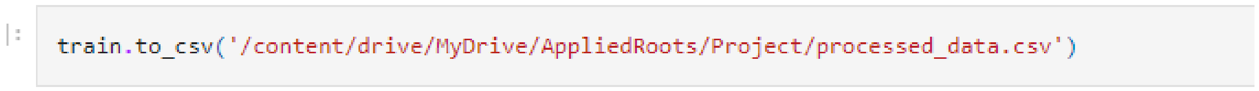
- **Observation: TotalAnnualReimbursedAmt vs InscClaimAmtReimbursed**

  If the TotalAnnualReimbursementAmt is greater than 100000 for a patient and ClaimAmtReimbursed is 20000, there is higher chance of it being Fraud.



- There were no significant patterns other plots such as (IPAnnualReimbursementAmt vs IPAnnualDeductibleAmt , OPAnnualReimbursementAmt vs OPAnnualDeductibleAmt, InscClaimAmtReibusement vs DeductibleAmtPaid) which could help us identify between fraud and non-fraud cases.

- After all the processing, the processed data was saved as a csv file for Phase-3 and Phase-4.



```
train.to_csv('/content/drive/MyDrive/AppliedRoots/Project/processed_data.csv')
```

- For complete code, refer the below Notebook on Github. https://github.com/prafulbatra/Fraudulent-Provider-Prediction/blob/dc8409957cf41fd42a7af918ce02ac6100214e31/Notebook/Phase_2_EDA.ipynb

3. Modeling and Error Analysis

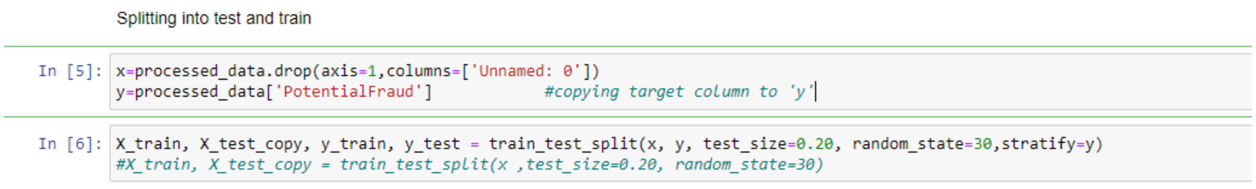
In this phase, we will first split our processed data from phase 2 and split it into train, test datasets randomly. Then, we will create some new features in train dataset and using the information learnt from those, we will create new for test set also. After, that we will train various models using PyCaret and will pick the best model based on various performance metrics that suits our business problem. Post modelling and selection of best model, we will use SHAP for error analysis. In error analysis, we will consider different categories of points from correctly and wrongly predicted points and check the important features which contributed towards prediction. Using this information, we will understand if model performance can be enhanced or not.

Note: PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. We can install it using ‘pip install pycaret’ in our system.

3.1 Data splitting and Featurization

- Test data given on Kaggle doesn't have labels. We will split the above processed data into train & test and use those datasets for training and validations.
- Before we do feature engineering, we will split data into train and test. Whatever transformations we will do, first we will apply in train set. Then we will concatenate train & test and apply same transformations on that. Later we will separate test data for validation.
- The above process is followed so that train set is transformed independently, no information from test set is present and there is no data leakage. Also, we are concatenating train & test to transform test set, so that information derived from train set is utilized to transform test set also.

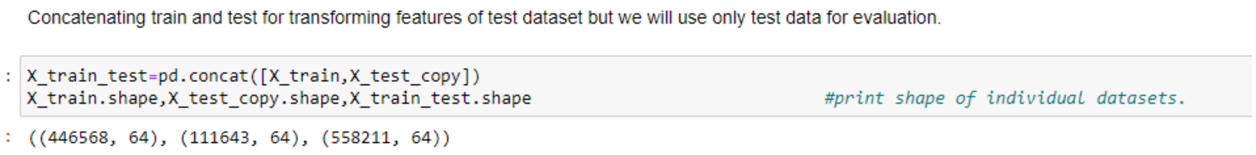
Code Snippet:



Splitting into test and train

```
In [5]: x=processed_data.drop(axis=1,columns=['Unnamed: 0'])
        y=processed_data['PotentialFraud']          #copying target column to 'y'
```

```
In [6]: X_train, X_test_copy, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=30,stratify=y)
        #X_train, X_test_copy = train_test_split(x ,test_size=0.20, random_state=30)
```



Concatenating train and test for transforming features of test dataset but we will use only test data for evaluation.

```
: X_train_test=pd.concat([X_train,X_test_copy])
  X_train.shape,X_test_copy.shape,X_train_test.shape                        #print shape of individual datasets.

: ((446568, 64), (111643, 64), (558211, 64))
```

Now, we will add new features on both these datasets and later separate test dataset from concatenated dataset for validating the trained model.

In exploratory data analysis, we observed that there were below 6 different amounts present in our dataset for each claim.

Table 1

InscClaimAmtReimbursed	OPAnnualReimbursementAmt	IPAnnualReimbursementAmt
DeductibleAmtPaid	OPAnnualDeductibleAmt	IPAnnualDeductibleAmt

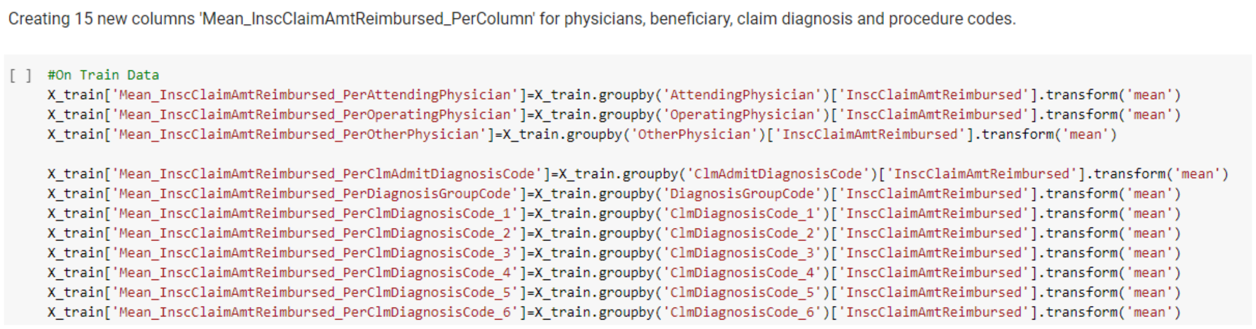
Also, we know from preliminary research that medicare fraud is an organized crime which involves physicians , beneficiaries and they use different diagnosis, procedure codes to forge data of the above money amounts. We will group by all our features with values of below features and calculate mean of each of these 6 amounts for each group.

Table 2

AttendingPhysician	OperatingPhysician	OtherPhysician
ClmAdmitDiagnosisCode	DiagnosisGroupCode	ClmDiagnosisCode_1
ClmDiagnosisCode_2	ClmDiagnosisCode_3	ClmDiagnosisCode_4
ClmDiagnosisCode_5	ClmDiagnosisCode_6	ClmProcedureCode_1
ClmProcedureCode_2	ClmProcedureCode_3	BeneID

For each feature in Table 1, we will use pandas group by for each feature in Table 2. In total, we will have 90 new features.

Code Snippet:



Creating 15 new columns 'Mean_InscClaimAmtReimbursed_PerColumn' for physicians, beneficiary, claim diagnosis and procedure codes.

```
[ ]  #On Train Data
     X_train['Mean_InscClaimAmtReimbursed_PerAttendingPhysician']=X_train.groupby('AttendingPhysician')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerOperatingPhysician']=X_train.groupby('OperatingPhysician')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerOtherPhysician']=X_train.groupby('OtherPhysician')['InscClaimAmtReimbursed'].transform('mean')

     X_train['Mean_InscClaimAmtReimbursed_PerClmAdmitDiagnosisCode']=X_train.groupby('ClmAdmitDiagnosisCode')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerDiagnosisGroupCode']=X_train.groupby('DiagnosisGroupCode')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerClmDiagnosisCode_1']=X_train.groupby('ClmDiagnosisCode_1')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerClmDiagnosisCode_2']=X_train.groupby('ClmDiagnosisCode_2')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerClmDiagnosisCode_3']=X_train.groupby('ClmDiagnosisCode_3')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerClmDiagnosisCode_4']=X_train.groupby('ClmDiagnosisCode_4')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerClmDiagnosisCode_5']=X_train.groupby('ClmDiagnosisCode_5')['InscClaimAmtReimbursed'].transform('mean')
     X_train['Mean_InscClaimAmtReimbursed_PerClmDiagnosisCode_6']=X_train.groupby('ClmDiagnosisCode_6')['InscClaimAmtReimbursed'].transform('mean')
```

After this we removed all the unnecessary columns as their information has already been converted into new features. And, we did one-hot encoding for Gender and Race variables.

```
#Dropping from train data
X_train.drop(['BeneID', 'ClaimID', 'ClaimStartDt', 'ClaimEndDt','AttendingPhysician', 'OperatingPhysician',
       'OtherPhysician', 'AdmissionDt', 'ClmAdmitDiagnosisCode', 'DischargeDt', 'DiagnosisGroupCode',
       'ClmDiagnosisCode_1', 'ClmDiagnosisCode_2', 'ClmDiagnosisCode_3',
       'ClmDiagnosisCode_4', 'ClmDiagnosisCode_5', 'ClmDiagnosisCode_6',
       'ClmDiagnosisCode_7', 'ClmDiagnosisCode_8', 'ClmDiagnosisCode_9',
       'ClmDiagnosisCode_10', 'ClmProcedureCode_1', 'ClmProcedureCode_2',
       'ClmProcedureCode_3', 'ClmProcedureCode_4', 'ClmProcedureCode_5',
       'ClmProcedureCode_6', 'DOB', 'DOD', 'RenalDiseaseIndicator',
       'State', 'County','ChronicCond_Alzheimer', 'ChronicCond_Heartfailure',
       'ChronicCond_KidneyDisease', 'ChronicCond_Cancer',
       'ChronicCond_ObstrPulmonary', 'ChronicCond_Depression',
       'ChronicCond_Diabetes', 'ChronicCond_IschemicHeart',
       'ChronicCond_Osteoporasis', 'ChronicCond_rheumatoidarthritis',
       'ChronicCond_stroke','ClaimMinusAdmitDays'],axis=1,inplace=True)
```

Also, we separated test data from concatenated train and test data. And final shape of data was as below.

```
[ ]  X_train.shape,X_test.shape

     ((446568, 112), (111643, 112))
```

3.2 Modelling

We will be using PyCaret for our model building and analysis. PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. It is an end-to-end machine learning and model management tool that exponentially speeds up the experiment cycle. We will use the classification module of this library.

To build a model in PyCaret, first we call setup() function which takes input as train data, test data and target column. This function needs to be executed before calling any other function, it does automatic pre-processing before modelling. Then, we can use create_model() to train new models and predict_model() for predictions. We will be training below models for our problem.

- Logistics Regression
- Decision Tree
- Random Forests
- Quadratic Discriminant Analysis
- Gradient Boosted Decision TreeClassifier
- LightGBM

3.2.1 Logistics Regression

It is a binary classification model which works well only when our classes are linearly separable by a hyperplane. It tries to find a hyperplane which minimizes log-loss in case classification and uses Stochastics Gradient Descent for finding minimum loss. By default , PyCaret uses default parameters like L2 regularization but that can be changed also. The advantage of this model is that it doesn't need to store all data at runtime and we can just need weights of the trained model for evaluation.

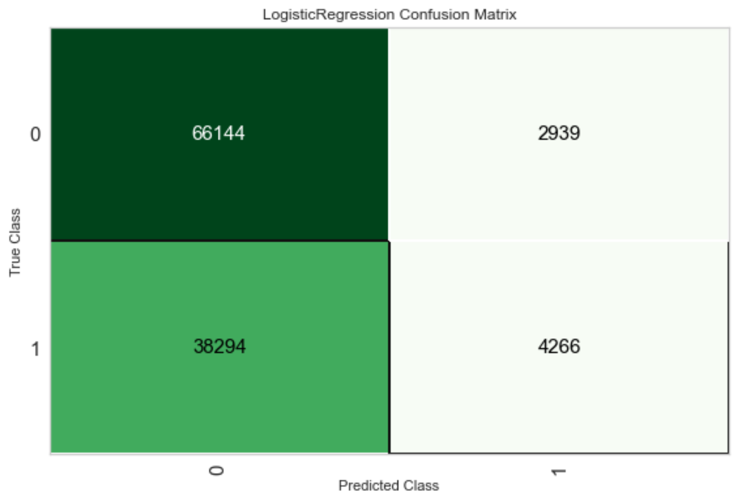
Scores on Test Data:

```
predict_model(lr);        #the scores on unseen data is also approximately same.
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.6307	0.5544	0.1002	0.5921	0.1714	0.0686	0.1140

Confusion Matrix on test data:

```
plot_model(lr, plot = 'confusion_matrix')
```



From the above performance, we can see that logistics regression has a very poor recall and approximately 10% of fraud claims are predicted correctly. 90% of fraud claims are predicted as non-fraud. It has precision of around 0.59, which is also not a very good score.

By default, it uses the below parameters for training.

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=100, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

We observed during training that even after hyperparameter tuning also, the recall increased from 0.10 to 0.15 but it was not very significant improvement.

3.2.2 Decision Trees

A decision tree is basically nested if-else conditions and it creates axis parallel hyperplanes as a model which can be used both for classification. The if-else conditions are written on nodes and usually 'Gini Impurity' is used as a splitting measure. Decision trees are unaffected by scale of individual data and are easy to interpret also. But , they tend to overfit and result in poor generalization performance.

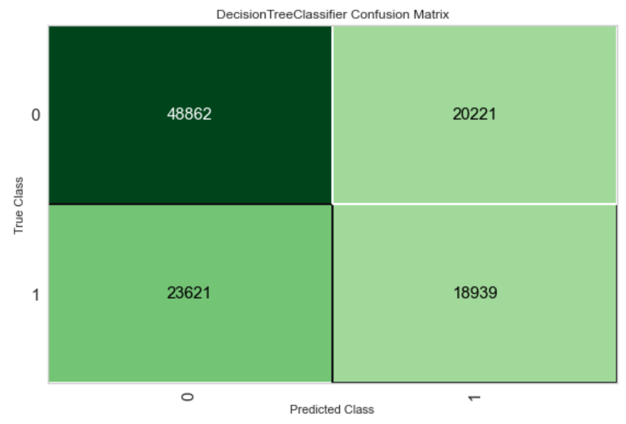
Scores on test data:

```
predict_model(dt);
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Decision Tree Classifier	0.6073	0.5761	0.4450	0.4836	0.4635	0.1547	0.1550

Confusion matrix on test data:

```
plot_model(dt, plot = 'confusion_matrix')
```



From the above performances we see that, recall has increased from the previous Logistics Regression model and it around 60% overall accuracy.

We also did tuning of the model but didn't get much improvement in the performance of the model and we have put parameter 'Chose_better=True', which return original model if tuning doesn't improvement performance.

```
tuned_dt
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=100, splitter='best')
```

3.2.3 Random Forest

This is a tree based ensemble model which combines low bias and high variance Decision trees to form one resultant low bias and low variance model. These models are simple to understand and don't require extensive hyperparameter tuning to get good performance and we can also do parallel computing on large datasets. But , they loose visual interpretability of single decision tree model and are slower to train than linear models.

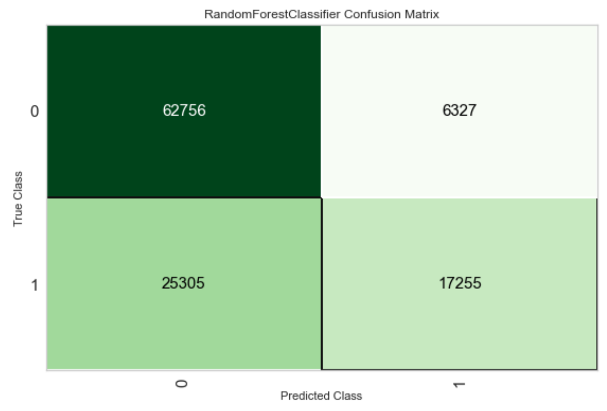
Scores on test:

```
predict_model(rf)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.7167	0.7318	0.4054	0.7317	0.5218	0.3432	0.3734

Confusion matrix on test:

```
plot_model(rf, plot = 'confusion_matrix')
```



In this model also we did tuning, but there was not significant improvement in the scores of the model. From performance, we observe that it has it has overall accuracy of 70% but the recall is still low and is only 40%.

3.2.4 LightGBM

It is an gradient boosting framework based on decision trees and it uses two techniques called Gradient-Based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) which makes it run faster with good accuracy. It uses less memory and is also compatible with large datasets. But, it is sensitive to overfitting and we should be careful to check this.

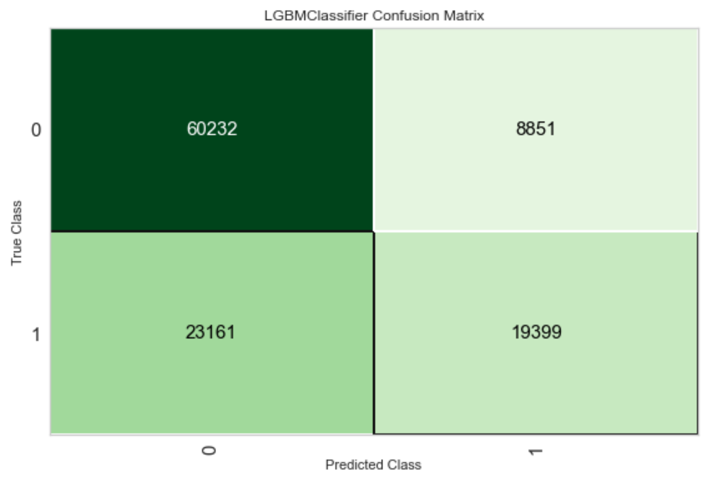
Scores on test:

```
predict_model(lightgbm)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Light Gradient Boosting Machine	0.7133	0.7172	0.4558	0.6867	0.5479	0.3503	0.3661

Confusion matrix on test:

```
plot_model(lightgbm,plot='confusion_matrix')
```

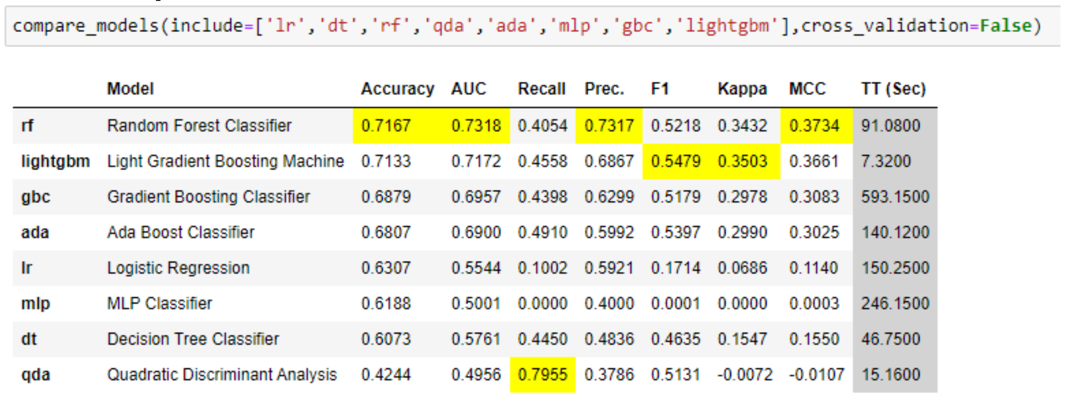


We also trained a tuned model, it was performing better on train dataset but its performance decreased on test dataset. So, we picked the initial lightgbm model.

Model parameters:

```
                        LGBMClassifier(boosting_type='gbdt', class_weight=None,
                                       colsample_bytree=1.0, importance_type='split',
                                       learning_rate=0.1, max_depth=-1,
                                       min_child_samples=20, min_child_weight=0.001,
                                       min_split_gain=0.0, n_estimators=100, n_jobs=-1,
                                       num_leaves=31, objective=None, random_state=100,
                                       reg_alpha=0.0, reg_lambda=0.0, silent='warn',
                                       subsample=1.0, subsample_for_bin=200000,
                                       subsample_freq=0)]],
         verbose=False),
 'lightgbm.pkl')
```

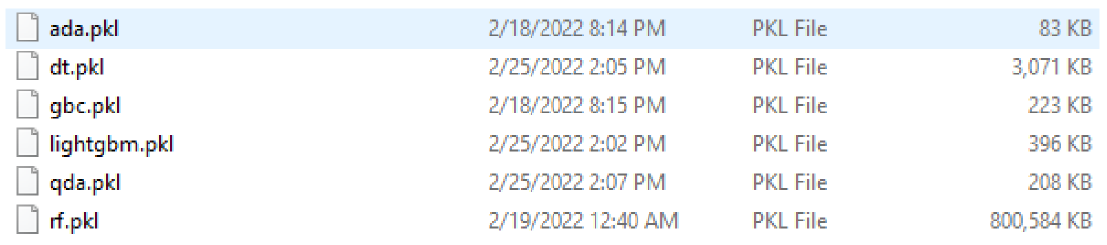
3.2.5 Comparison of all models



```
compare_models(include=['lr','dt','rf','qda','ada','mlp','gbc','lightgbm'],cross_validation=False)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.7167	0.7318	0.4054	0.7317	0.5218	0.3432	0.3734	91.0800
lightgbm	Light Gradient Boosting Machine	0.7133	0.7172	0.4558	0.6867	0.5479	0.3503	0.3661	7.3200
gbc	Gradient Boosting Classifier	0.6879	0.6957	0.4398	0.6299	0.5179	0.2978	0.3083	593.1500
ada	Ada Boost Classifier	0.6807	0.6900	0.4910	0.5992	0.5397	0.2990	0.3025	140.1200
lr	Logistic Regression	0.6307	0.5544	0.1002	0.5921	0.1714	0.0686	0.1140	150.2500
mlp	MLP Classifier	0.6188	0.5001	0.0000	0.4000	0.0001	0.0000	0.0003	246.1500
dt	Decision Tree Classifier	0.6073	0.5761	0.4450	0.4836	0.4635	0.1547	0.1550	46.7500
qda	Quadratic Discriminant Analysis	0.4244	0.4956	0.7955	0.3786	0.5131	-0.0072	-0.0107	15.1600

Observation: From the model performance scores, we find that we are getting better precision in Random Forest but better recall in Boosting models like AdaBoost & Light GBM. There is one advantage of using lightgbm that its pickle file size is very less as compared to Random Forest.

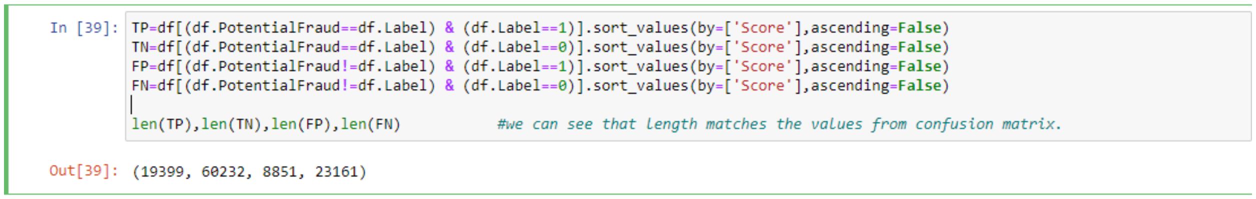


ada.pkl	2/18/2022 8:14 PM	PKL File	83 KB
dt.pkl	2/25/2022 2:05 PM	PKL File	3,071 KB
gbc.pkl	2/18/2022 8:15 PM	PKL File	223 KB
lightgbm.pkl	2/25/2022 2:02 PM	PKL File	396 KB
qda.pkl	2/25/2022 2:07 PM	PKL File	208 KB
rf.pkl	2/19/2022 12:40 AM	PKL File	800,584 KB

Random Forest has around 800 MB model pipeline size but for LightGBM, its only 400 KB. So, we will be choosing LightGBM as it will be easy to download and use in free tier of cloud services. We will also try some blend and stack models by using different model combinations to check if model performance improves.

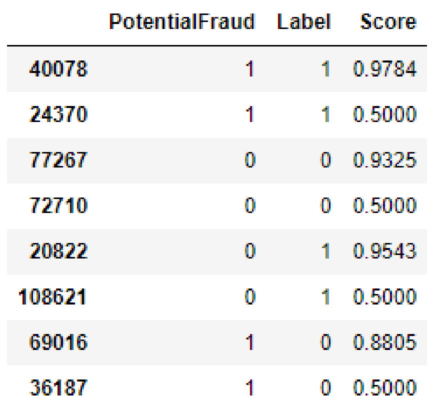
3.3) Error Analysis

For doing error analysis, we will group our points into 4 groups (TP,TN,FP,FN) and create 4 different datasets.



```
In [39]: TP=df[(df.PotentialFraud==df.Label) & (df.Label==1)].sort_values(by=['Score'],ascending=False)
         TN=df[(df.PotentialFraud==df.Label) & (df.Label==0)].sort_values(by=['Score'],ascending=False)
         FP=df[(df.PotentialFraud!=df.Label) & (df.Label==1)].sort_values(by=['Score'],ascending=False)
         FN=df[(df.PotentialFraud!=df.Label) & (df.Label==0)].sort_values(by=['Score'],ascending=False)

         len(TP),len(TN),len(FP),len(FN)          #we can see that length matches the values from confusion matrix.

Out[39]: (19399, 60232, 8851, 23161)
```

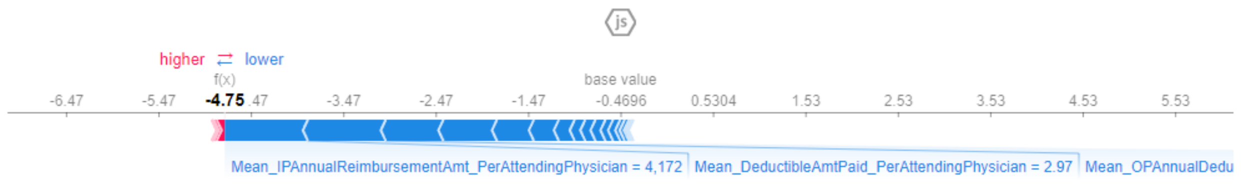
We will find the index in each of these dataframe for points with highest and lowest score and analyze each point using SHAP values to see which features are contributing in the prediction of points. In total, we will use below 8 points for error analysis.



	PotentialFraud	Label	Score
40078	1	1	0.9784
24370	1	1	0.5000
77267	0	0	0.9325
72710	0	0	0.5000
20822	0	1	0.9543
108621	0	1	0.5000
69016	1	0	0.8805
36187	1	0	0.5000

We used reason plot available in PyCaret library which internally calls SHAP library for plotting. The below datapoint is True Positive(provided and predicted label is Fraud) with high probability of 0.9784. In this plot , blue arrows shows the features which contribute to Fraud label and red arrows shows features contributing to Non-Fraud label.

```
interpret_model(lightgbm,plot='reason',observation=indexes[0],save=True)
```



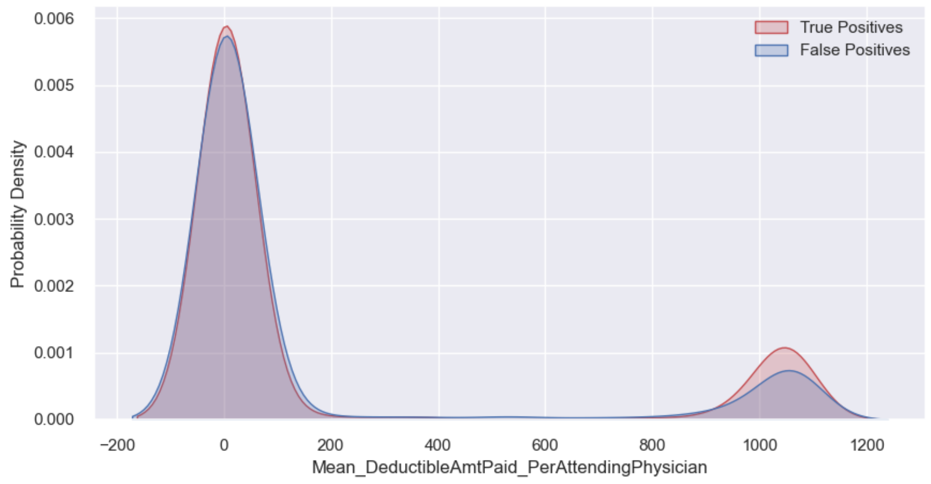
For this point, we got below features contributing for this prediction.

Mean_IPAnnualReimbursementAmt_PerAttendingPhysician	Mean_DeductibleAmtPaid_PerAttendingPhysician
Mean_OPAnnualDeductibleAmt_PerAttendingPhysician	Mean_OPAnnualReimbursementAmt_PerAttendingPhysician
Mean_InscClaimAmtReimbursed_PerAttendingPhysician	Mean_InscClaimAmtReimbursed_PerOtherPhysician

Features correlated with Non-Fraud.:

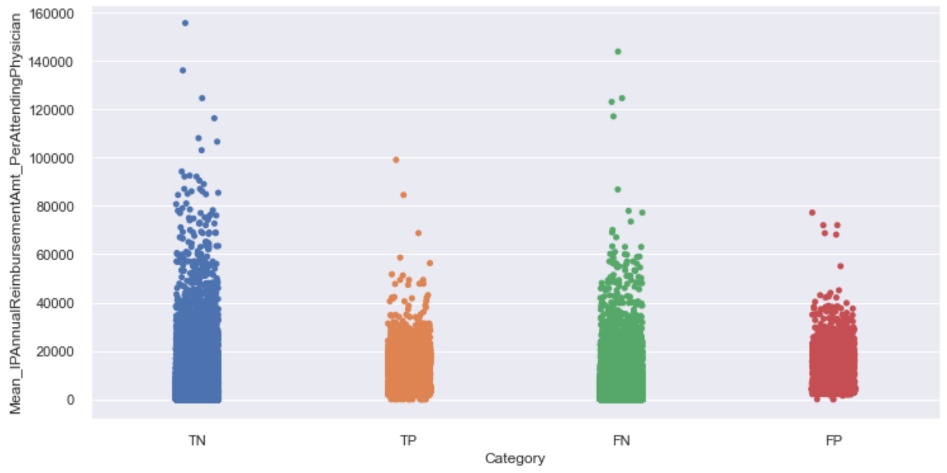
Mean_OPAnnualReimbursementAmt_PerOperatingPhysician
Mean_IPAnnualDeductibleAmt_PerOperatingPhysician

We found out the common features correlated with response variable for all the 8 points and plotted the distributions to understand more about the predictions.



Observation:

We saw from SHAP plots that 'Mean_DeductibleAmtPaid_PerAttendingPhysician' feature is contributing in both True Positives and False positives for the prediction. And from the above plots, we can see that both the distribution are exactly same which implies that model is considering this as an important factor in prediction and NonFraud points which have this behaviour, it is predicting those points also as Fraud. So , this is one of the drawbacks of our model.



We plotted multiple such plots and observed that True Positives & False Positives have similar kind of distribution and our model is looking at the distribution of these features predicting it to be Fraud. Similarly, the distribution of True Negatives and False Negatives is found to be similar. That means, based on some feature values, model is classifying some other points as Non-Fraud. In most features, the fraud claims have lesser value of these mean amounts.

This may be due to the reason that Physicians don't intentionally keep charge higher amounts for fraud claims so that it doesn't come to notice.

Please click on below link for Phase 3 python notebook.

Phase 3 Modelling

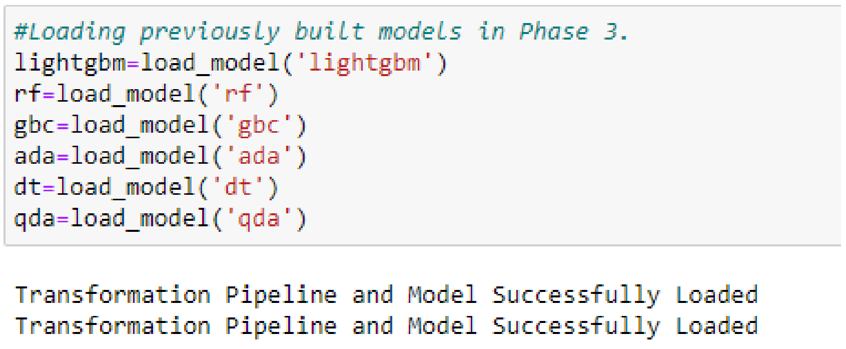
4. Advanced Modeling and Feature Engineering

In this phase, we will try some advance machine learning models like stacking and blend models which are another class of ensemble models and check if the performance improves from our best model from phase 2. We will also create new features based on errors in predictions produced by our best model to see if performance improves.

4.1 Advance Modelling

In Phase 3, we saved the data train and test data after feature engineering and also saved all the models along their transformation pipelines as pickle files. We will load all those models and build different ensemble models. We need to call the setup function again as it is must needed before executing any other command. We can also save the configuration of this function and load but its file size is very large around 2 GB, so we will run the function again.

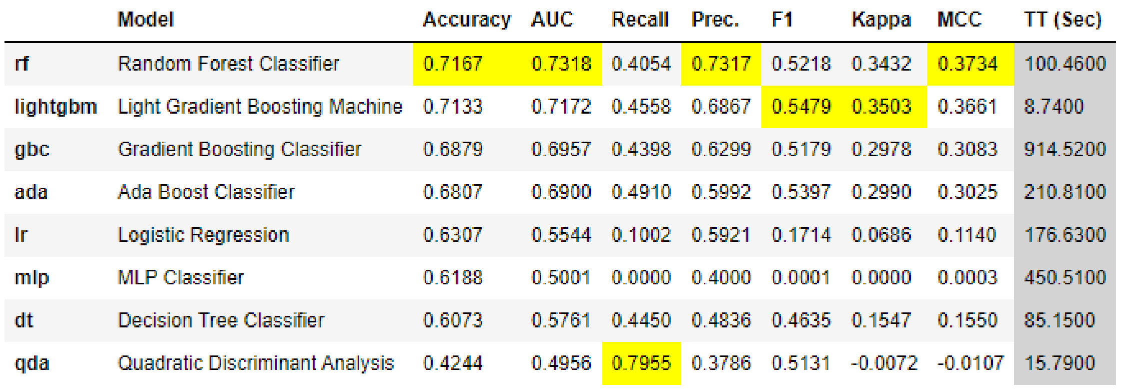
Loading all saved models



```
#Loading previously built models in Phase 3.
lightgbm=load_model('lightgbm')
rf=load_model('rf')
gbc=load_model('gbc')
ada=load_model('ada')
dt=load_model('dt')
qda=load_model('qda')
```

```
Transformation Pipeline and Model Successfully Loaded
Transformation Pipeline and Model Successfully Loaded
```

Scores from previous phase:



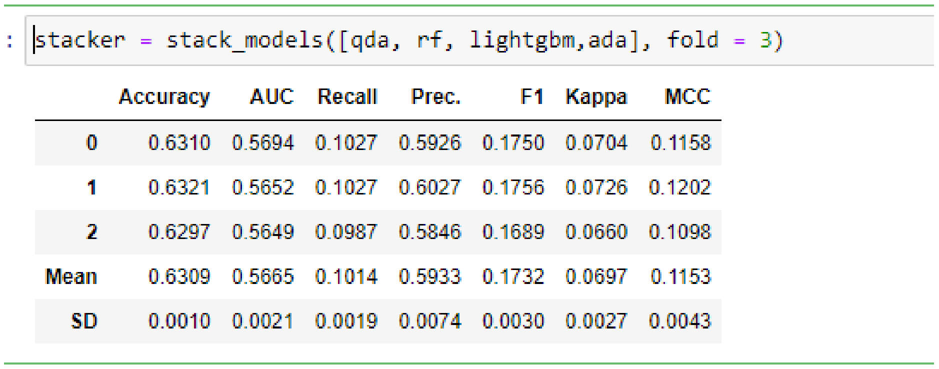
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.7167	0.7318	0.4054	0.7317	0.5218	0.3432	0.3734	100.4600
lightgbm	Light Gradient Boosting Machine	0.7133	0.7172	0.4558	0.6867	0.5479	0.3503	0.3661	8.7400
gbc	Gradient Boosting Classifier	0.6879	0.6957	0.4398	0.6299	0.5179	0.2978	0.3083	914.5200
ada	Ada Boost Classifier	0.6807	0.6900	0.4910	0.5992	0.5397	0.2990	0.3025	210.8100
lr	Logistic Regression	0.6307	0.5544	0.1002	0.5921	0.1714	0.0686	0.1140	176.6300
mlp	MLP Classifier	0.6188	0.5001	0.0000	0.4000	0.0001	0.0000	0.0003	450.5100
dt	Decision Tree Classifier	0.6073	0.5761	0.4450	0.4836	0.4635	0.1547	0.1550	85.1500
qda	Quadratic Discriminant Analysis	0.4244	0.4956	0.7955	0.3786	0.5131	-0.0072	-0.0107	15.7900

Now we will create a stacking model. In stacking model, the different the models are the better the resulting classifier is. In above models, we will also include qda as it has high recall. We will also include adaboost, random forest, and lightgbm.

4.1.1 Stack Classifiers

In stacking, we have two types of models, Level-0 models and Level-1 model. Level-0 or base models fits on the training data and whose predictions are combined while Level-1 or Meta model learns how to best combine the predictions of base models. The meta model is trained on the predictions made by base models or it can be trained both on train data and predictions together.

Stack Model 1: It uses Quadratic Discriminant Analysis, Random Forest, LightGBM and AdaBoost as base models and by default Logistics Regression as meta model

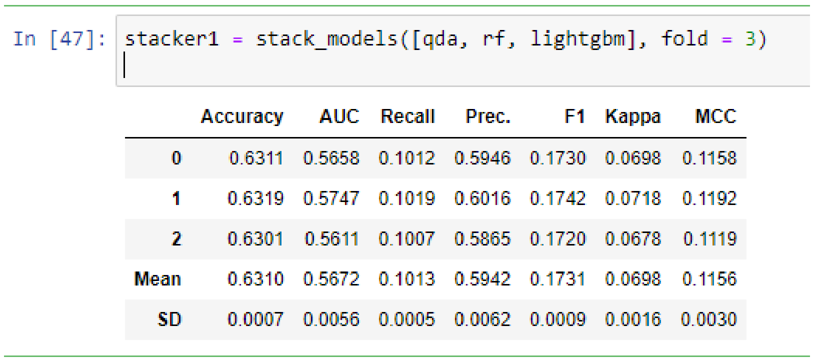


```
: stacker = stack_models([qda, rf, lightgbm,ada], fold = 3)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.6310	0.5694	0.1027	0.5926	0.1750	0.0704	0.1158
1	0.6321	0.5652	0.1027	0.6027	0.1756	0.0726	0.1202
2	0.6297	0.5649	0.0987	0.5846	0.1689	0.0660	0.1098
Mean	0.6309	0.5665	0.1014	0.5933	0.1732	0.0697	0.1153
SD	0.0010	0.0021	0.0019	0.0074	0.0030	0.0027	0.0043

Observation: We see that we are not getting good results with the stacking model. We will try some other combination. It’s similar to our baseline logistics regression model which is worse than LightGBM.

Stack Model 2: We will remove AdaBoost and use rest three models as base models.

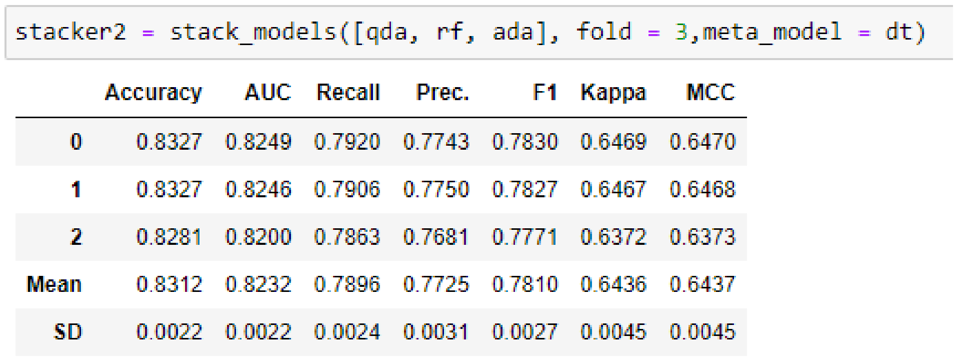


```
In [47]: stacker1 = stack_models([qda, rf, lightgbm], fold = 3)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.6311	0.5658	0.1012	0.5946	0.1730	0.0698	0.1158
1	0.6319	0.5747	0.1019	0.6016	0.1742	0.0718	0.1192
2	0.6301	0.5611	0.1007	0.5865	0.1720	0.0678	0.1119
Mean	0.6310	0.5672	0.1013	0.5942	0.1731	0.0698	0.1156
SD	0.0007	0.0056	0.0005	0.0062	0.0009	0.0016	0.0030

Observation: The scores are still low. PyCaret by default uses the meta model as logistics regression. We will try using some other model as meta model to see if performance improves as our dataset is not linearly separable and logistics regression is not able to separate well.

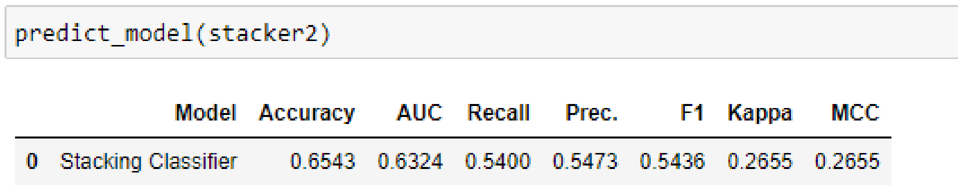
Stack Model 3:



```
stacker2 = stack_models([qda, rf, ada], fold = 3,meta_model = dt)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8327	0.8249	0.7920	0.7743	0.7830	0.6469	0.6470
1	0.8327	0.8246	0.7906	0.7750	0.7827	0.6467	0.6468
2	0.8281	0.8200	0.7863	0.7681	0.7771	0.6372	0.6373
Mean	0.8312	0.8232	0.7896	0.7725	0.7810	0.6436	0.6437
SD	0.0022	0.0022	0.0024	0.0031	0.0027	0.0045	0.0045

Now we see that, the model is performing very good on train data. Let's check performance on test dataset.



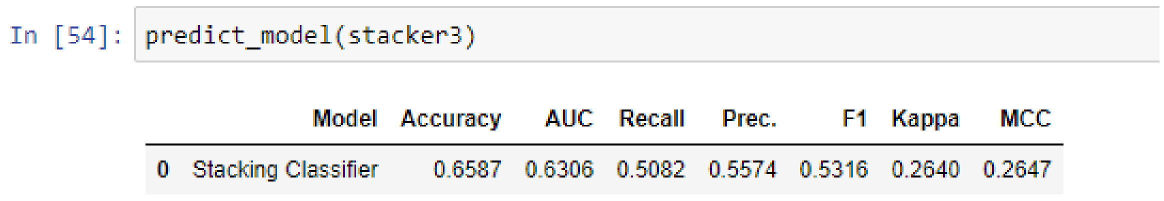
```
predict_model(stacker2)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Stacking Classifier	0.6543	0.6324	0.5400	0.5473	0.5436	0.2655	0.2655

The scores on test set is not good means this model was overfitting on train dataset.

Stack Model 4: In PyCaret, there is one parameter 'restack' which when set to False, the meta model uses both train data and predictions made by base models for its training.

We trained a model based on this and the performance decreased from the initial Stack Model 3



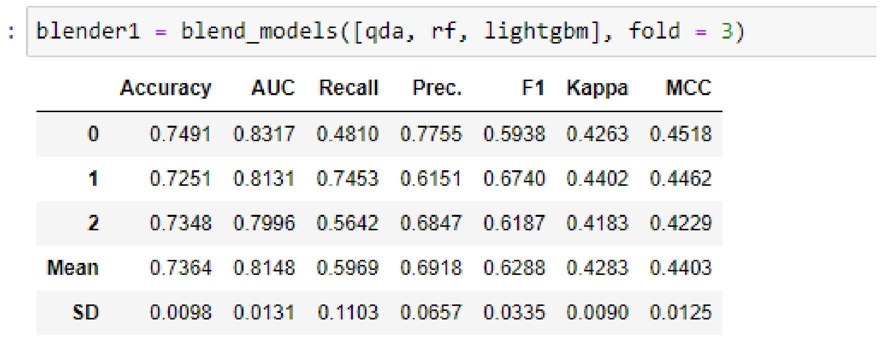
```
In [54]: predict_model(stacker3)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Stacking Classifier	0.6587	0.6306	0.5082	0.5574	0.5316	0.2640	0.2647

Observation: Thus, after trying various types of stack models, we come to a conclusion that our initial LightGBM model is best one till now.

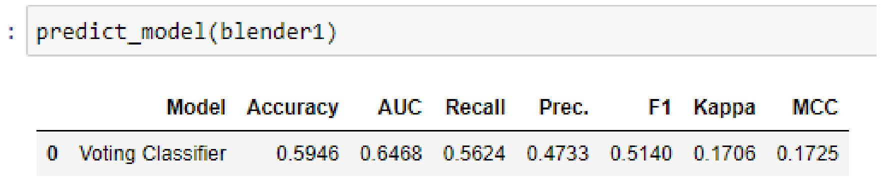
4.1.2 Blend Model

We will also try one blend model. Blend model in PyCaret trains a majority rule classifier for selected models that we pass in the list. We will use QDA, Random Forest and LightGBM as the participant models and check the performance.



```
: blender1 = blend_models([qda, rf, lightgbm], fold = 3)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7491	0.8317	0.4810	0.7755	0.5938	0.4263	0.4518
1	0.7251	0.8131	0.7453	0.6151	0.6740	0.4402	0.4462
2	0.7348	0.7996	0.5642	0.6847	0.6187	0.4183	0.4229
Mean	0.7364	0.8148	0.5969	0.6918	0.6288	0.4283	0.4403
SD	0.0098	0.0131	0.1103	0.0657	0.0335	0.0090	0.0125



```
: predict_model(blender1)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Voting Classifier	0.5946	0.6468	0.5624	0.4733	0.5140	0.1706	0.1725

Observation: From the performance on the test dataset, we find that, although Recall increased from 0.45 to 0.56 but Precision decreased from 0.62 to 0.47 and overall performance also decreased.

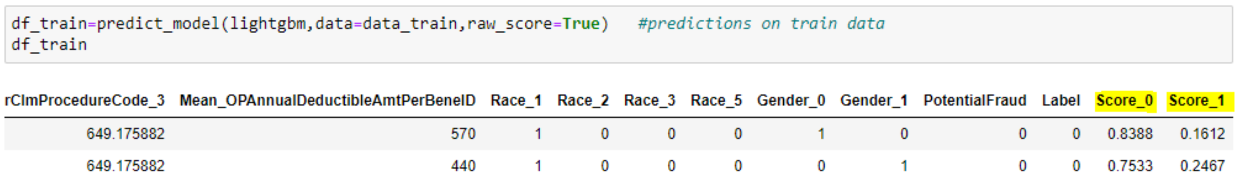
Therefore, we chose our LightGBM model from phase 3 as our final model. We will further so advance feature engineering on errors predicted by our model.

Note: We also tried Multi Layered Perceptron in compare_model function of PyCaret in Phase-3 but it had very poor performance.

4.2 Advanced Feature Engineering

In this, we will first segregate all the points into different categories whether they are predicted correctly or incorrectly based on their labels and probability scores. After providing labels to each point, we will predict each label using multi-class classification and we will use prediction probability scores of each class as new features for our original train dataset and see if there is any improvement in performance using these features.

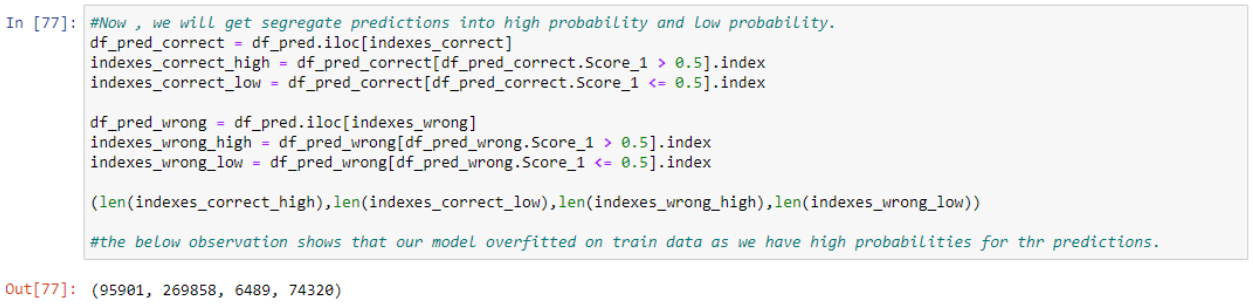
- First, we will get to know the errors predicted by out LightGBM model on both labels. We can use 'raw_score=True' argument to get both the scores.



```
df_train=predict_model(lightgbm,data=data_train,raw_score=True)   #predictions on train data
df_train
```

rClmProcedureCode_3	Mean_OPAnnualDeductibleAmtPerBeneID	Race_1	Race_2	Race_3	Race_5	Gender_0	Gender_1	PotentialFraud	Label	Score_0	Score_1
649.175882	570	1	0	0	0	1	0	0	0	0.8388	0.1612
649.175882	440	1	0	0	0	0	1	0	0	0.7533	0.2467

- After this, we used Score_1 probability values to bin points into 4 categories.

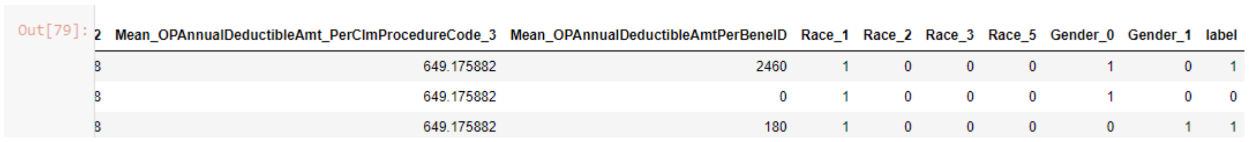


```
In [77]: #Now , we will get segregate predictions into high probability and low probability.
         df_pred_correct = df_pred.iloc[indexes_correct]
         indexes_correct_high = df_pred_correct[df_pred_correct.Score_1 > 0.5].index
         indexes_correct_low = df_pred_correct[df_pred_correct.Score_1 <= 0.5].index

         df_pred_wrong = df_pred.iloc[indexes_wrong]
         indexes_wrong_high = df_pred_wrong[df_pred_wrong.Score_1 > 0.5].index
         indexes_wrong_low = df_pred_wrong[df_pred_wrong.Score_1 <= 0.5].index

         (len(indexes_correct_high),len(indexes_correct_low),len(indexes_wrong_high),len(indexes_wrong_low))

         #the below observation shows that our model overfitted on train data as we have high probabilities for thr predictions.

Out[77]: (95901, 269858, 6489, 74320)
```

- We segregate all the points below:

index_correct_high	contains Fraud points predicted correctly with high probability
index_correct_low	contains Non-Fraud points predicted correctly with high probability
index_wrong_high	conatains Non-Fraud points predicted incorrectly with high probability
index_wrong_low	conatains Fraud points predicted incorrectly with high probability

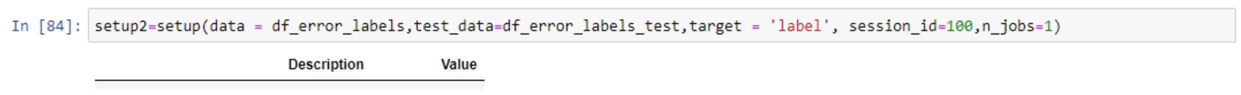
- Then, we created four DataFrames for all these points and merged them to create labels(0,1,2,3) as below which are used as a multiclass classification problem.



Out[79]:

2	Mean_OPAnnualDeductibleAmt_PerClmProcedureCode_3	Mean_OPAnnualDeductibleAmtPerBeneID	Race_1	Race_2	Race_3	Race_5	Gender_0	Gender_1	label
8	649.175882	2460	1	0	0	0	1	0	1
8	649.175882	0	1	0	0	0	1	0	0
8	649.175882	180	1	0	0	0	0	1	1

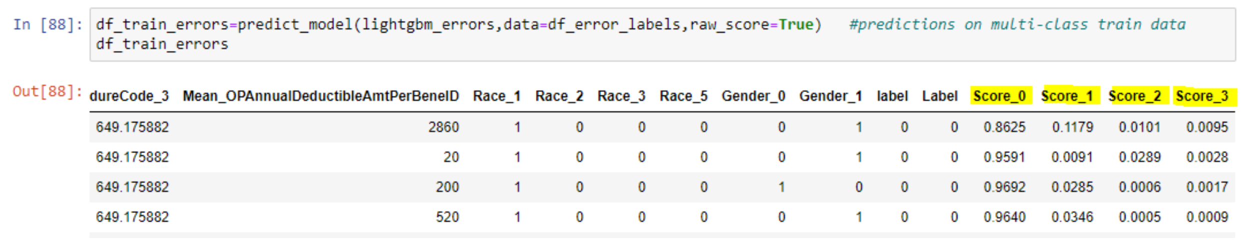
- We did all the same preprocessing on test set also and called the setup function again as below for the multiclass classification problem.



```
In [84]: setup2=setup(data = df_error_labels,test_data=df_error_labels_test,target = 'label', session_id=100,n_jobs=1)
```

Description	Value

- After training and testing, we got 4 new probability scores for each label which will be used as new features in our initial train set. Similarly, 4 new features will be added in our test dataset also.

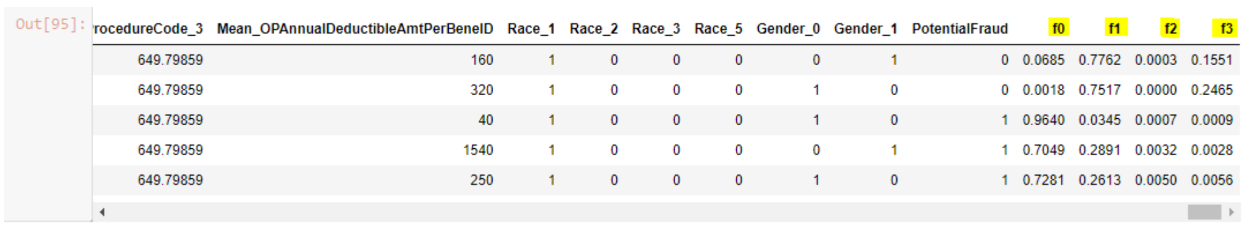


```
In [88]: df_train_errors=predict_model(lightgbm_errors,data=df_error_labels,raw_score=True)    #predictions on multi-class train data
         df_train_errors
```

Out[88]:

dureCode_3	Mean_OPAnnualDeductibleAmtPerBeneID	Race_1	Race_2	Race_3	Race_5	Gender_0	Gender_1	label	Label	Score_0	Score_1	Score_2	Score_3
649.175882	2860	1	0	0	0	0	1	0	0	0.8625	0.1179	0.0101	0.0095
649.175882	20	1	0	0	0	0	1	0	0	0.9591	0.0091	0.0289	0.0028
649.175882	200	1	0	0	0	1	0	0	0	0.9692	0.0285	0.0006	0.0017
649.175882	520	1	0	0	0	0	1	0	0	0.9640	0.0346	0.0005	0.0009

- Below are new features for our initial datasets on which we will train our best performing model LightGBM.

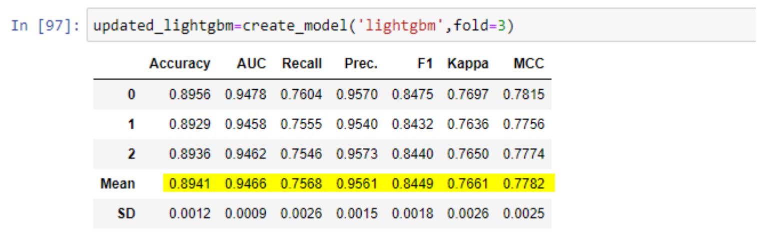


Out[95]:

rocedureCode_3	Mean_OPAnnualDeductibleAmtPerBeneID	Race_1	Race_2	Race_3	Race_5	Gender_0	Gender_1	PotentialFraud	f0	f1	f2	f3
649.79859	160	1	0	0	0	0	1	0	0.0685	0.7762	0.0003	0.1551
649.79859	320	1	0	0	0	1	0	0	0.0018	0.7517	0.0000	0.2465
649.79859	40	1	0	0	0	1	0	1	0.9640	0.0345	0.0007	0.0009
649.79859	1540	1	0	0	0	0	1	1	0.7049	0.2891	0.0032	0.0028
649.79859	250	1	0	0	0	1	0	1	0.7281	0.2613	0.0050	0.0056

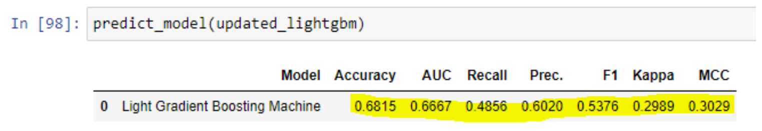
- Now, we will use these 4 new features to predict our original Fraudulent claim detect to see if there is any improvement in performance.

- Below was the performance on our train and test datasets using new four features.



```
In [97]: updated_lightgbm=create_model('lightgbm',fold=3)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8956	0.9478	0.7604	0.9570	0.8475	0.7697	0.7815
1	0.8929	0.9458	0.7555	0.9540	0.8432	0.7636	0.7756
2	0.8936	0.9462	0.7546	0.9573	0.8440	0.7650	0.7774
Mean	0.8941	0.9466	0.7568	0.9561	0.8449	0.7661	0.7782
SD	0.0012	0.0009	0.0026	0.0015	0.0018	0.0026	0.0025

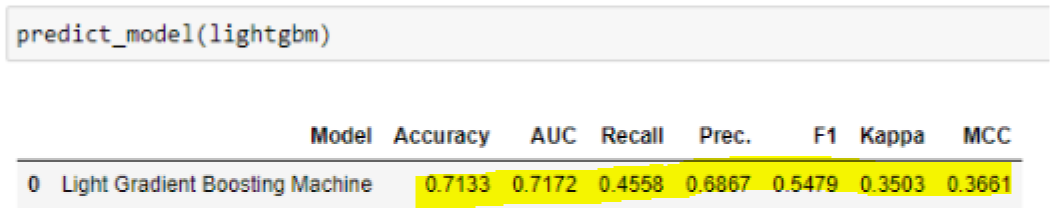


```
In [98]: predict_model(updated_lightgbm)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Light Gradient Boosting Machine	0.6815	0.6667	0.4856	0.6020	0.5376	0.2989	0.3029

Observation: From the performance of above model, we find that recall has increased from 0.45 to 0.48 from our original dataset but precision decreased from 0.68 to 0.60 and overall performance decreased if we compare by F1 score and MCC.

Performance of our model on initial data without feature engineering.



```
predict_model(lightgbm)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Light Gradient Boosting Machine	0.7133	0.7172	0.4558	0.6867	0.5479	0.3503	0.3661

Conclusion:

From the above performances of all stack models, blend models, and also updated LightGBM model built on errors of previous model, we find that initial LightGBM model has the best performance and we will be using that for deployment.

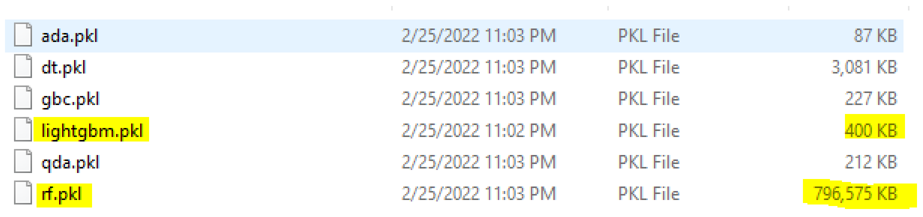
Please refer to this notebook for complete code. Phase-4-Advance-Modelling

5. Model Deployment

5.1 Introduction and Execution Steps

In this phase, we will deploy our model on Streamlit Cloud Platform. Streamlit is a python library which let us build interactive Data apps using simple commands. It also provides PaaS(Platform as a Service) to host our web application using free-tier. We will place our code on Github and from there Streamlit will connect to the repository to install the libraries and run the application.

Before deploying, we will see the size of different pickle files of various models and reaffirm the reason for choosing LightGBM as our preferred model for deployment.



ada.pkl	2/25/2022 11:03 PM	PKL File	87 KB
dt.pkl	2/25/2022 11:03 PM	PKL File	3,081 KB
gbc.pkl	2/25/2022 11:03 PM	PKL File	227 KB
lightgbm.pkl	2/25/2022 11:02 PM	PKL File	400 KB
qda.pkl	2/25/2022 11:03 PM	PKL File	212 KB
rf.pkl	2/25/2022 11:03 PM	PKL File	796,575 KB

Although, Random Forest and LightGBM have almost similar performance but the pickle file size of Random Forest is too large. That's why we will LightGBM for deployment.

- Link for Github Repo: https://github.com/prafulbatra/Fraudulent-Provider-Prediction
- Link for Web Application: https://share.streamlit.io/prafulbatra/fraudulent-provider-prediction/main

Steps for Running the web application from Browser

1. Open the web application link mentioned above in any internet browser.
2. Download the test datasets present in Data folder present on Github repository.
3. Upload the datasets on the web app and click Submit button to see the predictions.

Steps for Running the app in Local

1. Clone the repo using git clone:
   git clone https://github.com/prafulbatra/Fraudulent-Provider-Prediction

2. Install dependencies using pip:
   pip install -r requirements.txt

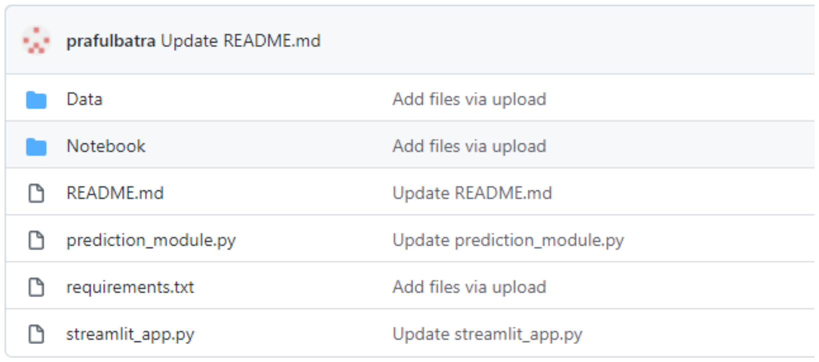
3. Go to CMD and run the following command:
   streamlit run streamlit_app.py
   (Run it from the root directory of the project)

After running the above command, it will start the streamlit server in local machine and the application will open automatically in local host on port 8501. After application opens in local machine, we need to follow the same steps mentioned to get predictions.

Now, we will see in detail the structure of Github Repository, the code files and also the UI of web application.

Github Repository

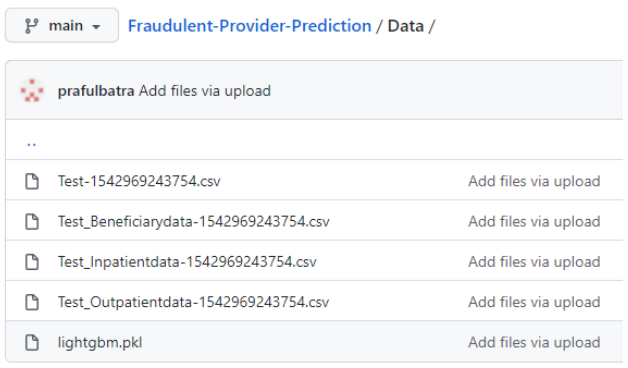
Below is the directory of Github repository. We have updated all the information in the README.md file as to what the individual file does and also what each folder contains.



prafulbatra Update README.md

Data	Add files via upload
Notebook	Add files via upload
README.md	Update README.md
prediction_module.py	Update prediction_module.py
requirements.txt	Add files via upload
streamlit_app.py	Update streamlit_app.py

- streamlit_app.py is the main file to run the Streamlit app.
- prediction_module.py contains functions for checking file format, preprocessing, feature engineering and for predictions.
- requirements.txt contains Dependencies which gets installed on Streamlit platform automatically on runtime.
- Data folder contains the test datasets that needs to be uploaded on the web application. It also contains our saved model pickle file, lightgbm.pkl.



main Fraudulent-Provider-Prediction / Data /

prafulbatra Add files via upload

..	
Test-1542969243754.csv	Add files via upload
Test_Beneficiarydata-1542969243754.csv	Add files via upload
Test_Inpatientdata-1542969243754.csv	Add files via upload
Test_Outpatientdata-1542969243754.csv	Add files via upload
lightgbm.pkl	Add files via upload

- Notebook folder contains the iPython Notebooks for all the phases.

5.2 Code Walkthrough

Code For streamlit_app.py

- For doing all necessary imports.

```
import streamlit as st
import pandas as pd
from pycaret.classification import *
import os
import requests
from prediction_module import file_check,preprocessing,feature_engineering,predictions
```

- For setting Page title, header and uploading all four files.

```
st.set_page_config(page_title="Claim Prediction")                                     # webpage title
st.title("This page predicts if a Healthcare claim is fraudulent or not")             #Prints page Header.

#uploading files
uploaded_file1=st.file_uploader(label='Upload Beneficiary excel file',key=1)
uploaded_file2=st.file_uploader(label='Upload Inpatient excel file',key=2)
uploaded_file3=st.file_uploader(label='Upload Outpatient excel file',key=3)
uploaded_file4=st.file_uploader(label='Upload Provider excel file',key=4)
```

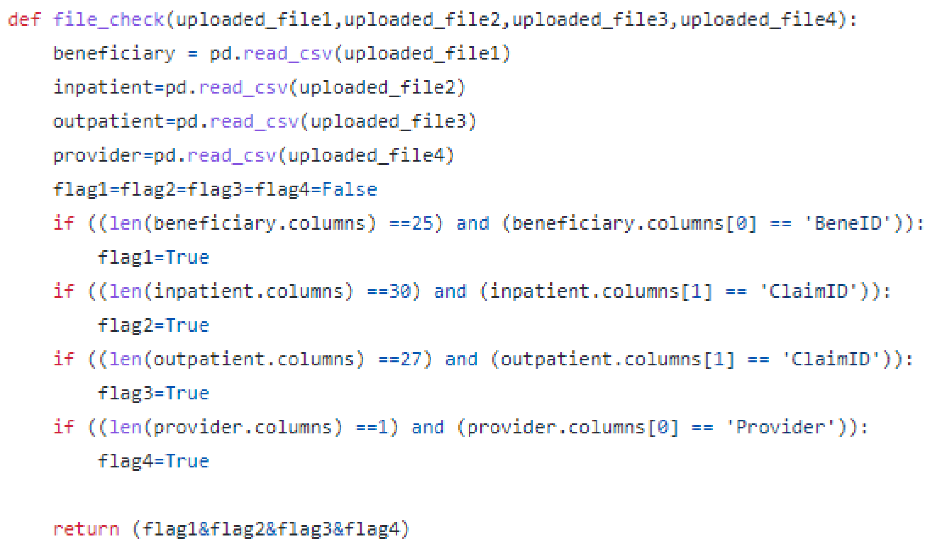
- Below code checks if all the files are uploaded and also it calls various functions of prediction_module for checking format of all 4 csv files and afterwards it calls the predictions function which internally calls preprocessing and feature_engineering functions.

```
if ((uploaded_file1 is not None)&(uploaded_file2 is not None)&(uploaded_file3 is not None)&(uploaded_file4 is not None)):     #when all files are uploaded, it goes inside
    if(file_check(uploaded_file1,uploaded_file2,uploaded_file3,uploaded_file4)):
        st.write("Now , you are all set for prediction. Click Submit Button to get the results")
        uploaded_file1.seek(0)
        beneficiary = pd.read_csv(uploaded_file1)                  #creates a dataframe for beneficiary data
        uploaded_file2.seek(0)
        inpatient=pd.read_csv(uploaded_file2)                      #creates a dataframe for inpatient data
        uploaded_file3.seek(0)
        outpatient=pd.read_csv(uploaded_file3)                     #creates a dataframe for outpatient data
        uploaded_file4.seek(0)
        provider=pd.read_csv(uploaded_file4)                       #creates dataframe for providers
        with st.form("Fraud Prediction",clear_on_submit=True):     #it creates a Submit form section with Submit button inside.
            if st.form_submit_button("Submit"):                    #when Submit button is clicked, it calls the predictions() function from prediction_module.
                results=predictions(beneficiary,inpatient,outpatient,provider)
                st.write(results)
    else:
        st.write('Upload correct file')   #when the file format is not correct, it shows below message.

else:                                     #when all files are not uploaded, it throws below message.
    st.write('Upload all files')
```

Code for prediction_module

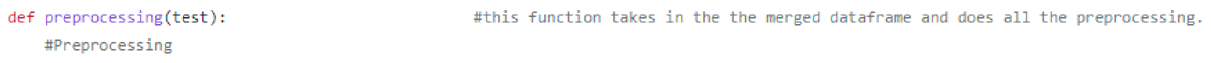
This has 4 functions -file_check, preprocessing, feature_engineering and predictions.

- The below functions validates the no. of columns and name of primary key column for each of the four uploaded files and returns True if it matches all conditions.



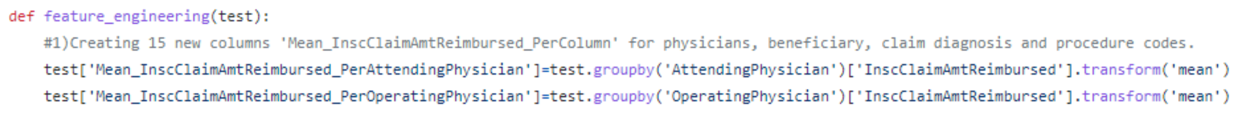
```
def file_check(uploaded_file1,uploaded_file2,uploaded_file3,uploaded_file4):
    beneficiary = pd.read_csv(uploaded_file1)
    inpatient=pd.read_csv(uploaded_file2)
    outpatient=pd.read_csv(uploaded_file3)
    provider=pd.read_csv(uploaded_file4)
    flag1=flag2=flag3=flag4=False
    if ((len(beneficiary.columns) ==25) and (beneficiary.columns[0] == 'BeneID')):
        flag1=True
    if ((len(inpatient.columns) ==30) and (inpatient.columns[1] == 'ClaimID')):
        flag2=True
    if ((len(outpatient.columns) ==27) and (outpatient.columns[1] == 'ClaimID')):
        flag3=True
    if ((len(provider.columns) ==1) and (provider.columns[0] == 'Provider')):
        flag4=True

    return (flag1&flag2&flag3&flag4)
```

- The below function takes input as dataframe does all the pre-processing as done in Phase-2



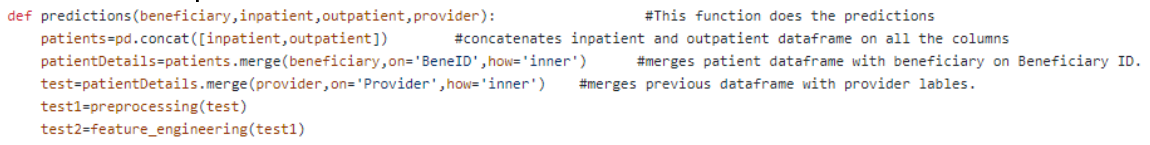
```
def preprocessing(test):                          #this function takes in the the merged dataframe and does all the preprocessing.
    #Preprocessing
```

- The below function adds all the new features as done in Phase-3



```
def feature_engineering(test):
    #1)Creating 15 new columns 'Mean_InscClaimAmtReimbursed_PerColumn' for physicians, beneficiary, claim diagnosis and procedure codes.
    test['Mean_InscClaimAmtReimbursed_PerAttendingPhysician']=test.groupby('AttendingPhysician')['InscClaimAmtReimbursed'].transform('mean')
    test['Mean_InscClaimAmtReimbursed_PerOperatingPhysician']=test.groupby('OperatingPhysician')['InscClaimAmtReimbursed'].transform('mean')
```

- The below function downloads the lightgbm pickle file in current working directory and does the predictions on test dataset and returns in into a dataframe.



```
def predictions(beneficiary,inpatient,outpatient,provider):                 #This function does the predictions
    patients=pd.concat([inpatient,outpatient])        #concatenates inpatient and outpatient dataframe on all the columns
    patientDetails=patients.merge(beneficiary,on='BeneID',how='inner')      #merges patient dataframe with beneficiary on Beneficiary ID.
    test=patientDetails.merge(provider,on='Provider',how='inner')    #merges previous dataframe with provider lables.
    test1=preprocessing(test)
    test2=feature_engineering(test1)
```

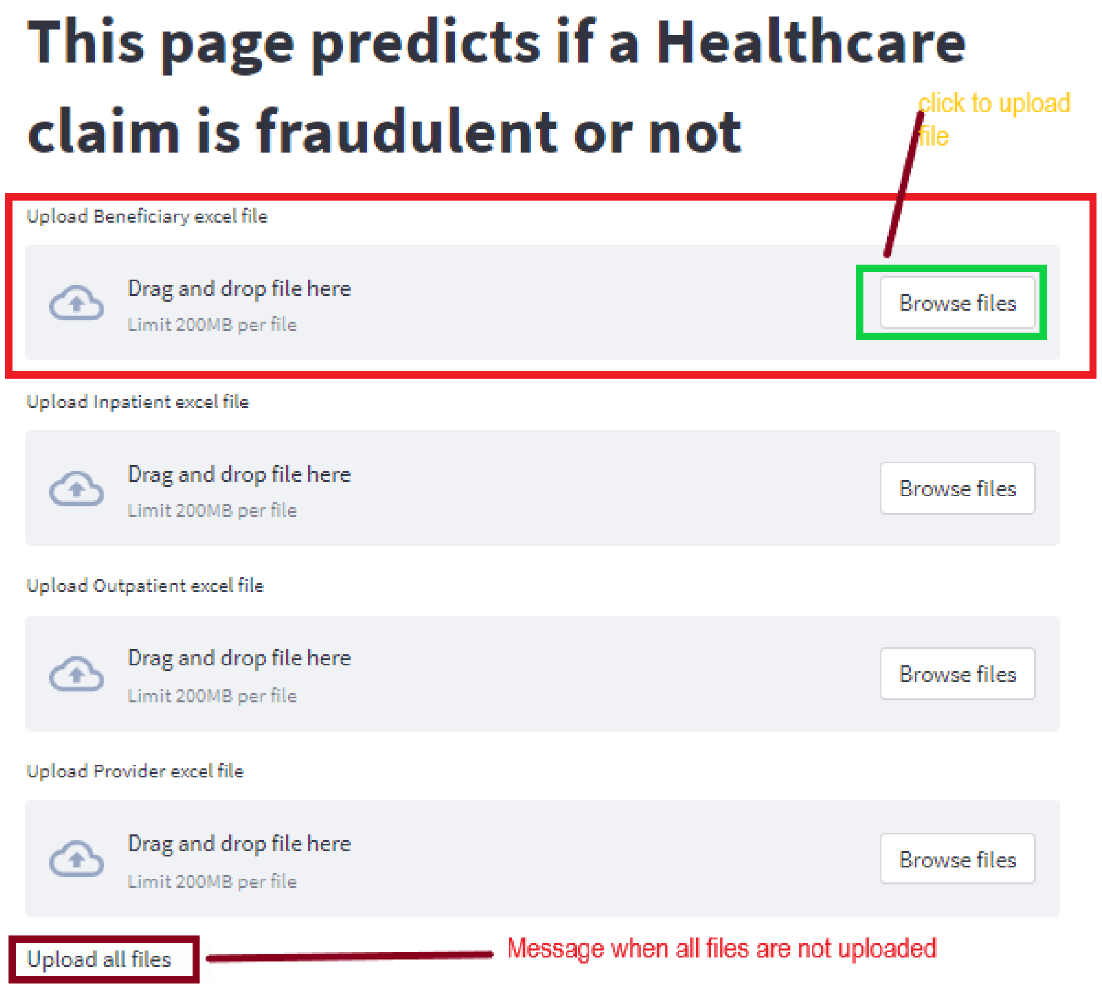
For full code, refer to the files:https://github.com/prafulbatra/Fraudulent-Provider-Prediction/edit/main/prediction_module.py

https://github.com/prafulbatra/Fraudulent-Provider-Prediction/blob/main/streamlit_app.py

5.3 Application Walkthrough

Link: https://share.streamlit.io/prafulbatra/fraudulent-provider-prediction/main

Before uploading the files:



This page predicts if a Healthcare claim is fraudulent or not

Upload Beneficiary excel file

Drag and drop file here
Limit 200MB per file

Upload Inpatient excel file

Drag and drop file here
Limit 200MB per file

Upload Outpatient excel file

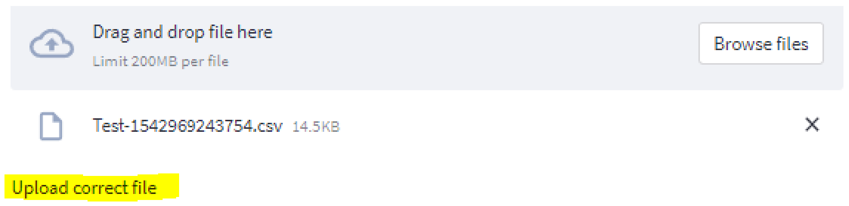
Drag and drop file here
Limit 200MB per file

Upload Provider excel file

Drag and drop file here
Limit 200MB per file

Upload all files

As we see above, we need to upload respective files in each uploader. When all the files are not uploaded, we get Message below 'Upload all files'
When any incorrect file is uploaded in any of the four uploaders, we get below message. in place of previous message.

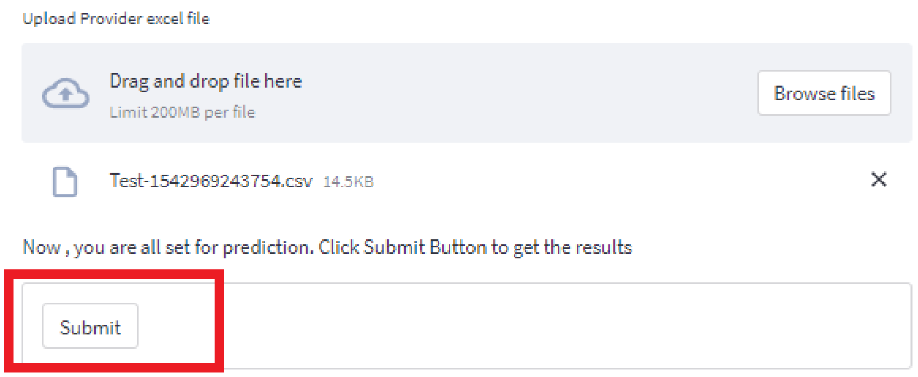


Drag and drop file here
Limit 200MB per file

Test-1542969243754.csv 14.5KB

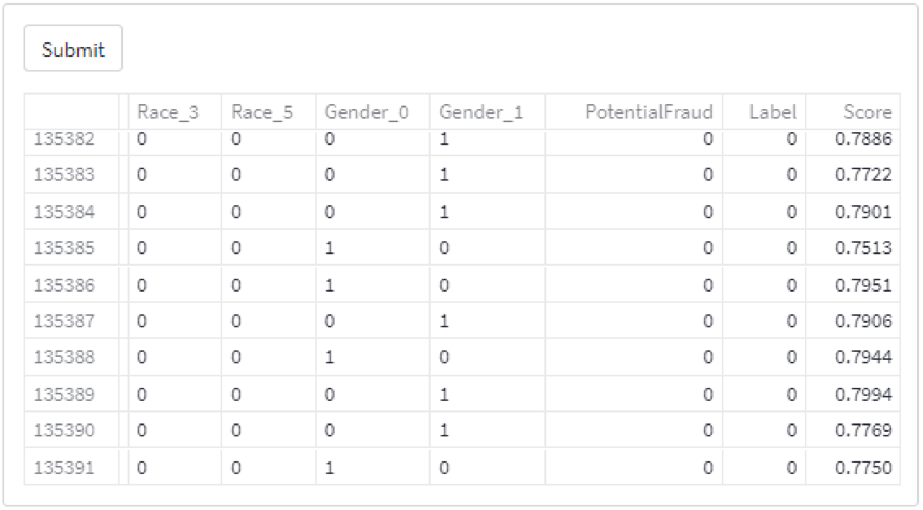
Upload correct file

After uploading all the files correctly, we see 'Submit' button gets displayed.



Now , you are all set for prediction. Click Submit Button to get the results

Post submitting, we get the details of all 135391 claims, and their probability scores of prediction.



	Race_3	Race_5	Gender_0	Gender_1	PotentialFraud	Label	Score
135382	0	0	0	1	0	0	0.7886
135383	0	0	0	1	0	0	0.7722
135384	0	0	0	1	0	0	0.7901
135385	0	0	1	0	0	0	0.7513
135386	0	0	1	0	0	0	0.7951
135387	0	0	0	1	0	0	0.7906
135388	0	0	1	0	0	0	0.7944
135389	0	0	0	1	0	0	0.7994
135390	0	0	0	1	0	0	0.7769
135391	0	0	1	0	0	0	0.7750

Note: This is Kaggle test dataset, and we didn't have the response labels. But, in PyCaret, it is mandatory to have the response column for the setup function to run. Therefore, we added the response column 'PotentialFraud' with all the values equal to zero. This column isn't used anywhere in the predictions as we are using our trained model for predictions.

5.4 Limitations and Improvements

Limitations of current system

- We have used pycaret for whole model pipeline and one mandatory thing to do in pycaret is to run the setup command which takes more time to execute and because of that there is more latency observed.

- We can save the configuration of setup function which was used in training but its file size is almost 2GB and we can’t use that in Streamlit as it has only 1GB of storage in free tier plus it will take huge time to download and load that file system’s RAM.
- While feature engineering, we have not used the train data as the size of processed train data was more than 100 MB which can’t be uploaded in free account of Github. If we use train data, we could get better results due to better averaging of new values.

Possible Improvements

- Instead of Pycaret, we can use Sklearn for model deployment as now we know the best performing model and we don’t need to do other activities which takes more time in Scikit Learn. We can perform all the pre-processing ourselves without pycaret and save the Standard Scaler as pickle file which can be used for standardization during runtime.

- We could use some other cloud provider like AWS where we can place our processed train data in S3 bucket and can be used during feature engineering at runtime.

- We can improve the UI further and results display further. Instead of uploading all files separately, we can just take all files at once and do the predictions. While displaying the results of predictions, we can further display the feature importance.

- Currently, we are not checking names of all the columns before passing it further for predictions. We can validate names of all the columns in all four files to make it more robust to incorrect file upload.

References

1) https://www.kaggle.com/rohitrox/healthcare-provider-fraud-detection-analysis
2) https://pycaret.gitbook.io/docs/get-started/functions
3) https://docs.streamlit.io/library/api-reference