Healthcare Provider Fraud Detection

Predictive Analytics Project

Submitted to:

Prof. Enayat Rajabi

Table of Contents

Objective:

The Main Objective of the Project is to develop a Predictive model to predict whether Healthcare Provider is a Fraud or Non Fraud Provider

Problem Type:

This is a Binary classification problem where we are trying to predict whether the provider is a Fraud or Non Fraud based on predictor variables.

Dataset:

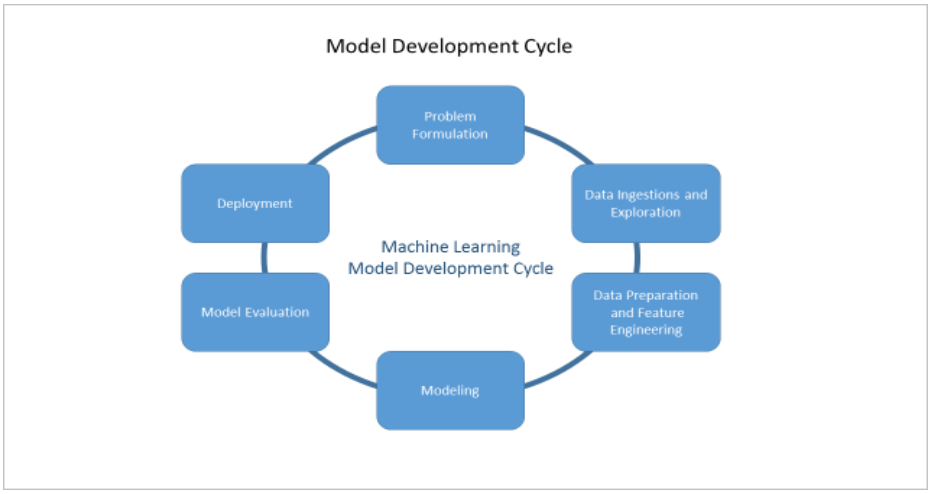
The dataset contains data in 4 different files in .csv format.

1. **Inpatient.csv :** The Inpatient data provides details about the claims filed for those patients who were admitted in the hospitals. It also provides additional details like their admission and discharge dates and admit d diagnosis code.
2. **Outpatient.csv:** This data provides details about the claims filed for those patients who visit hospitals and not admitted in it. The Data in both Inpatients and Outpatients data is contains similar features.
3. **Beneficiary.**csv: This data provides information about the beneficiary details such as health condition, region, state they belongs to etc.
4. **Provider.**csv: The provider data contains details of the Healthcare provider such as the providerId and target variable PotentialFraud which gives information about the Fraud and Non Fraud providers.

The Dataset has been taken from Kaggle:

https://www.kaggle.com/datasets/rohitrox/healthcare-provider-fraud-detection-analysis

Model development Stages:



Exploratory Data Analysis (EDA):

Exploratory Data Analysis is an approach to analyse the dataset to summarize their main characteristics using statistical methods and visualization tools such as charts, plots etc.

1) Data Understanding:

In order to better understand the data and to get insights about the data we are using Data understanding as a basic step which is also the first step in exploratory analysis.

Step:

1. Importing and loading the required python libraries

```
# Import Libraries
import pandas as pd # Pandas for data processing  used to analyse data
import numpy as np #Numpy (Numerical Python consisting of Multi-Dimensional Arrays)
import seaborn as sns # Data visualization and Exploratory data Analysis Library
import matplotlib.pyplot as plt # Matplotlib Library for Data Visualisation
%matplotlib inline
%matplotlib notebook
```

2.Reading the Files and Finding the Total number of records in each Data Frame

```
#Loading Healthcare Fraud Detection Dataset

Provider=pd.read_csv('Provider.csv') # Loading the Provider dataset into the Provider Dataframe
Beneficiary=pd.read_csv("Beneficiarydata.csv")  # Reading and storing the Beneficiary data in Beneficiary Dataframe

#Loading  In-Patient and OutPatient Dataset\

Inpatient=pd.read_csv('Inpatientdata.csv') # Reading the Data from Inpatient Input file into the Inpatient Dataframe
Outpatient=pd.read_csv('Outpatientdata.csv')#Reading the Data from Outpatient Input file into the Inpatient Dataframe
```

```
# Finding the shape of the Datasets
print("********************SHAPE OF DATASET IS*****************\n")
print("Shape of Provider dataset is :",Provider.shape)
print("Shape of  Beneficiary dataset is :",Beneficiary.shape)
print("Shape of  In-Patient dataset is :",Inpatient.shape)
print("Shape of  Out-Patient dataset is :",Outpatient.shape)

```

```
********************SHAPE OF DATASET IS*****************

Shape of Provider dataset is : (5410, 2)
Shape of  Beneficiary dataset is : (138556, 25)
Shape of  In-Patient dataset is : (40474, 30)
Shape of  Out-Patient dataset is : (517737, 27)
```

Our Data set contains four different files with Inpatient, Outpatient, Beneficiary and Provider details.

3. **Data Consolidation:** The data from four different input data frames needs to be merged to create a single data frame with all the features.

4. **Merging Inpatient and OutPatient data:** The Inpatient and outpatient files have similar columns so we merged the data based on columns using outer join

```
#Merging the In-Patient and Out-Patient data based on Similar Columns
InOutPatients=pd.merge(Outpatient,Inpatient,
                           left_on=['BeneID', 'ClaimID', 'ClaimStartDt', 'ClaimEndDt', 'Provider',
       'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
       'OtherPhysician', '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', 'DeductibleAmtPaid',
       'ClmAdmitDiagnosisCode'],
                           right_on=['BeneID', 'ClaimID', 'ClaimStartDt', 'ClaimEndDt', 'Provider',
       'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
       'OtherPhysician', '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', 'DeductibleAmtPaid',
       'ClmAdmitDiagnosisCode']
                           ,how='outer')
```

5. **Merging Beneficiary and Provider Data with Patients:**

Merging the All patient data with beneficiary and Provider Details to create a single Data frame with all the patients and provider data

```
# Merging beneficiary Details Dataset to the InOutPatients Dataframe
AllPatients=pd.merge(InOutPatients,Beneficiary,left_on='BeneID',right_on='BeneID',how='inner')


# Merging the Provider Data from Train into the All Patients data
Healthcare=pd.merge(Provider,AllPatients,on='Provider')

```

```
print("Healthcare",Healthcare.shape) # Print the shape of the Consolidated dataframe
```

```
Healthcare (558211, 55)
```

The consolidated Data frame Healthcare consists of 558211 rows with 55 features.

Statistical Analysis:

	count	mean	std	min	25%	50%	75%	max
InscClaimAmtReimbursed	558211.0	997.012133	3821.534891	0.0	40.00	80.0	300.0	125000.0
ClmProcedureCode_1	23310.0	5896.154612	3050.489933	11.0	3848.00	5363.0	8669.0	9999.0
ClmProcedureCode_2	5490.0	4106.358106	2031.640878	42.0	2724.00	4019.0	4439.0	9999.0
ClmProcedureCode_3	969.0	4221.123839	2281.849885	42.0	2724.00	4019.0	5185.0	9999.0
ClmProcedureCode_4	118.0	4070.262712	2037.626990	42.0	2754.25	4019.0	4439.0	9986.0
ClmProcedureCode_5	9.0	5269.444444	2780.071632	2724.0	4139.00	4139.0	5185.0	9982.0
ClmProcedureCode_6	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DeductibleAmtPaid	557312.0	78.421085	274.016812	0.0	0.00	0.0	0.0	1068.0
Gender	558211.0	1.578838	0.493746	1.0	1.00	2.0	2.0	2.0
Race	558211.0	1.255011	0.717437	1.0	1.00	1.0	1.0	5.0
State	558211.0	25.446969	15.192784	1.0	11.00	24.0	38.0	54.0
County	558211.0	378.588195	265.215531	0.0	150.00	350.0	570.0	999.0
NoOfMonths_PartACov	558211.0	11.931472	0.889712	0.0	12.00	12.0	12.0	12.0
NoOfMonths_PartBCov	558211.0	11.938770	0.785900	0.0	12.00	12.0	12.0	12.0
ChronicCond_Alzheimer	558211.0	1.598132	0.490276	1.0	1.00	2.0	2.0	2.0
ChronicCond_Heartfailure	558211.0	1.409573	0.491755	1.0	1.00	1.0	2.0	2.0
ChronicCond_KidneyDisease	558211.0	1.587998	0.492196	1.0	1.00	2.0	2.0	2.0
ChronicCond_Cancer	558211.0	1.848615	0.358424	1.0	2.00	2.0	2.0	2.0

The Statistical Analysis shows that most of the features with amount values have a maximum value quite high to their standard deviation and some features have a non- numeric values which are erroneous.

Data Dictionary:

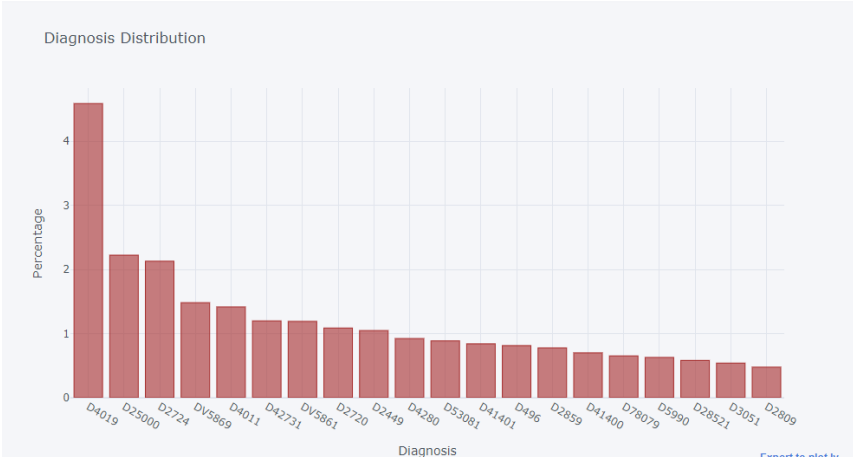
The details about each features such as the data type are

```
1 Healthcare.dtypes
```

```
Provider                          object
PotentialFraud                    object
BeneID                            object
ClaimID                           object
ClaimStartDt                      object
ClaimEndDt                        object
InscClaimAmtReimbursed             int64
AttendingPhysician                object
OperatingPhysician                object
OtherPhysician                    object
ClmDiagnosisCode_1                object
ClmDiagnosisCode_2                object
ClmDiagnosisCode_3                object
ClmDiagnosisCode_4                object
ClmDiagnosisCode_5                object
ClmDiagnosisCode_6                object
ClmDiagnosisCode_7                object
ClmDiagnosisCode_8                object
ClmDiagnosisCode_9                object
ClmDiagnosisCode_10               object
ClmProcedureCode_1               float64
ClmProcedureCode_2               float64
ClmProcedureCode_3               float64
ClmProcedureCode_4               float64
```

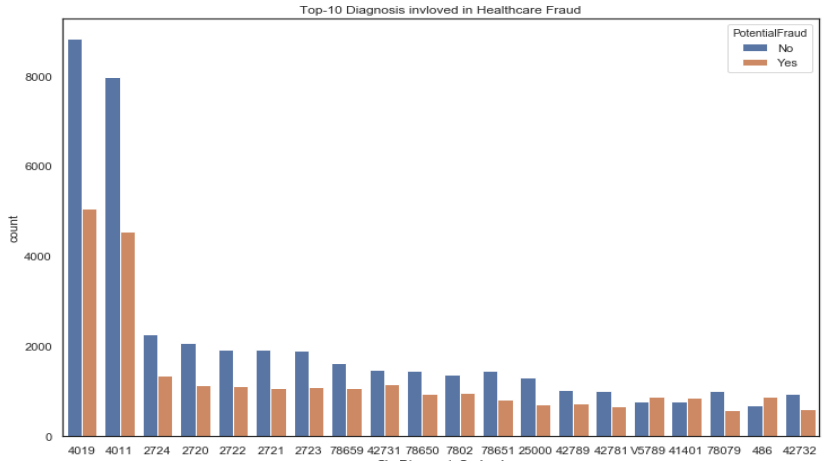
2. Data Analysis:

I. Finding the Most Common Diagnosis applied for Fraud and Non Fraud Claims



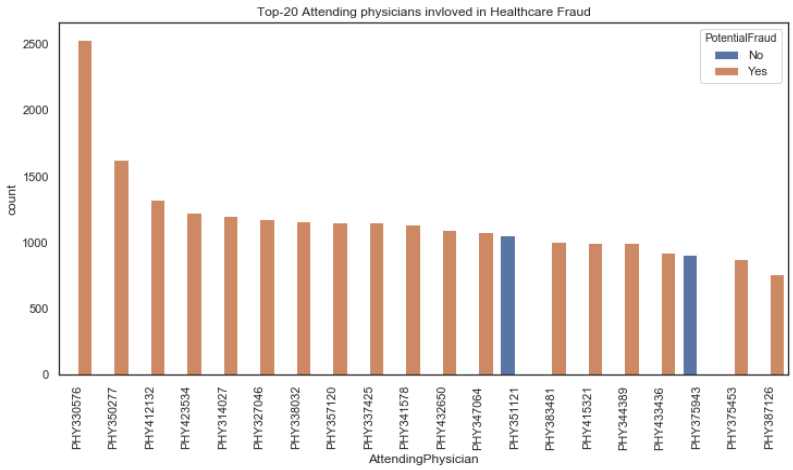
Findings: From the above chart we found that most common Diagnosis codes used in the data set are 4019, 25000, 2724 followed by others.

II. Finding the Top 10 Claim Diagnosis involved in Fraud



Findings: we found that the top 10 Diagnosis codes using for Fraudulent claims are quite different from the most common used diagnosis codes

III. Find the Top 20 Attending Physicians involved in Fraud



Findings: The physician with PHY30576 as Attending Physician code is involved in most number of Fraud claims cases and PHY351121 and PHY375943 physicians are not involved in single Fraud claims

3) Feature Extraction:

Feature Extraction refers to process of extracting new features from the existing features of raw data by preserving the information in the original dataset.

Existing Feature	**New Feature**	**New Feature**
DOB,DOD	Age	Extracted by Subtracting the DOD(date of Death) from DOB column
DOD.isna(),DOD.isnotna	Whether_Dead(0,1)	0 if Dead and 1 if still
Admitted_Dt,Discharge_DT	TotalDaysAdmitted	Calculated the Total number of days admitted from Inpatients data
ClaimStart_Dt,ClaimEnd_Dt	Claims_Days	Given information about how many days it took for reimbursement for
AttendingPhysician, OperatingPhysician, OtherPhysician	Unique_Physician	Find the Number of Unique Physicians from different Physicians
AttendingPhysician, OperatingPhysician, OtherPhysician	Types_of_Physicians	To count the number of types of physicians who attended the Patient

4) Data Cleansing:

Finding the Variables with Missing Data

```
# We find which variables hold missing data
na = Healthcare.isnull().sum()
na[na != 0]
```

```
DeductibleAmtPaid        899
AdmissionDt           517737
DischargeDt           517737
DiagnosisGroupCode    517737
DOD                   554080
Age                   554080
dtype: int64
```

Dropping Columns from the Dataset

```
Drop_columns=['BeneID', 'ClaimID', 'ClaimStartDt','ClaimEndDt','AttendingPhysician',
       'OperatingPhysician', 'OtherPhysician','AdmissionDt',
       'DischargeDt', 'DiagnosisGroupCode','DOB', 'DOD',
        'State', 'County']

Healthcare_cleaned=Healthcare.drop(axis=1,columns=Drop_columns)
```

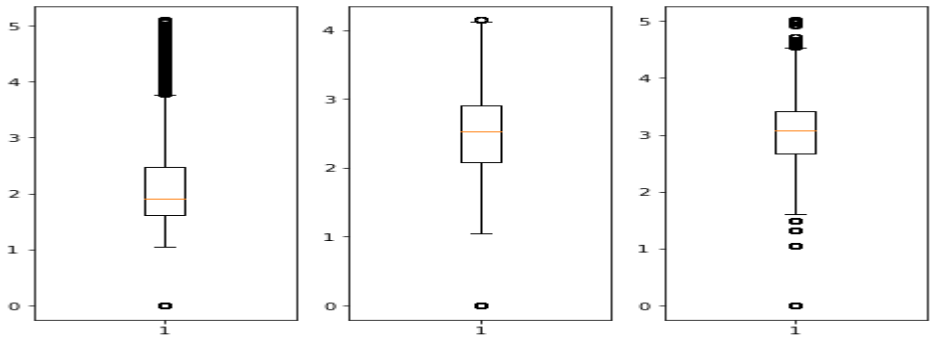
Since we extracted the Age and Total days admitted we dropped the above columns.

Converting the Target variable from Categorical to Numerical

```
Healthcare_cleaned.PotentialFraud.replace(['Yes','No'],['1','0'],inplace=True)
Healthcare_cleaned.PotentialFraud=Healthcare_cleaned.PotentialFraud.astype(str).astype(int)
```

5) Outlier Detection and Handling

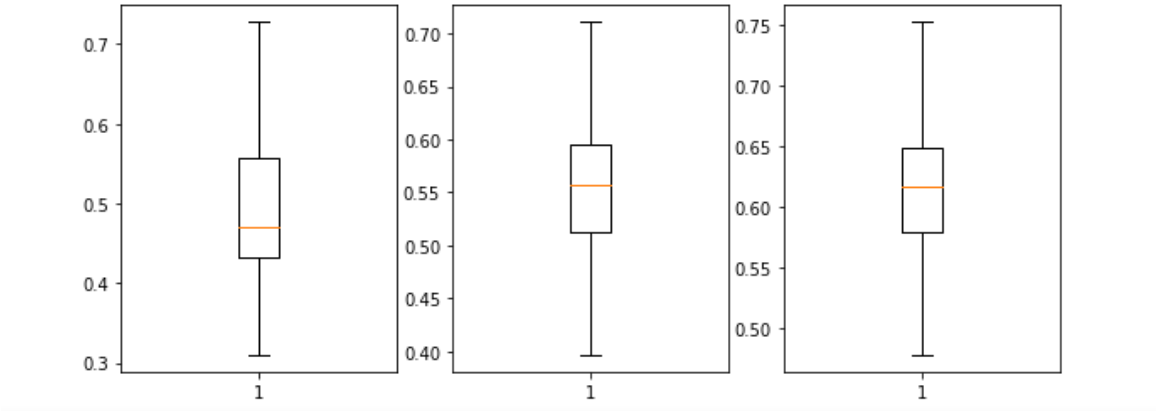
```
plt.figure(figsize = (8,6))
plt.title('Box Plot')

plt.subplot(1,3,1)
plt.boxplot( Healthcare_cleaned['InscClaimAmtReimbursed'] )

plt.subplot(1,3,2)
plt.boxplot(Healthcare_cleaned['OPAnnualDeductibleAmt'])

plt.subplot(1,3,3)
plt.boxplot(Healthcare_cleaned['OPAnnualReimbursementAmt'])
```



Outlier Handling:

```
#for InscClaimAmtReimbursed
Q1 = Healthcare_cleaned['InscClaimAmtReimbursed'].quantile(0.25)
Q3 = Healthcare_cleaned['InscClaimAmtReimbursed'].quantile(0.75)
IQR = Q3-Q1
upper = Q3 + 1.5*IQR
lower = Q1 - 1.5*IQR
Healthcare_cleaned = Healthcare_cleaned[(Healthcare_cleaned['InscClaimAmtReimbursed'] >= lower) & (Healthcare_cleaned['InscC

# for OPAnnualDeductibleAmt

Q1 = Healthcare_cleaned['OPAnnualDeductibleAmt'].quantile(0.25)
Q3 = Healthcare_cleaned['OPAnnualDeductibleAmt'].quantile(0.75)
IQR = Q3-Q1
upper = Q3 + 1.5*IQR
lower = Q1 - 1.5*IQR
Healthcare_cleaned = Healthcare_cleaned[(Healthcare_cleaned['OPAnnualDeductibleAmt'] > lower) & (Healthcare_cleaned['OPAnnua

# for OPAnnualReimbursementAmt
Q1 = Healthcare_cleaned['OPAnnualReimbursementAmt'].quantile(0.25)
Q3 = Healthcare_cleaned['OPAnnualReimbursementAmt'].quantile(0.75)
IQR = Q3-Q1
upper = Q3 + 1.5*IQR
lower = Q1 - 1.5*IQR
Healthcare_cleaned = Healthcare_cleaned[(Healthcare_cleaned['OPAnnualReimbursementAmt'] > lower) & (Healthcare_cleaned['OPAn
```

The Outliers were handled using IQR Method. The values which are more than 1.5 times the Interquartile range from the quartiles which are below are handled using Q1-1.5*IQR and upper values using Q3+1.5*IQR.



The Outliers were handled using the IQR method and we found no outliers in the data.

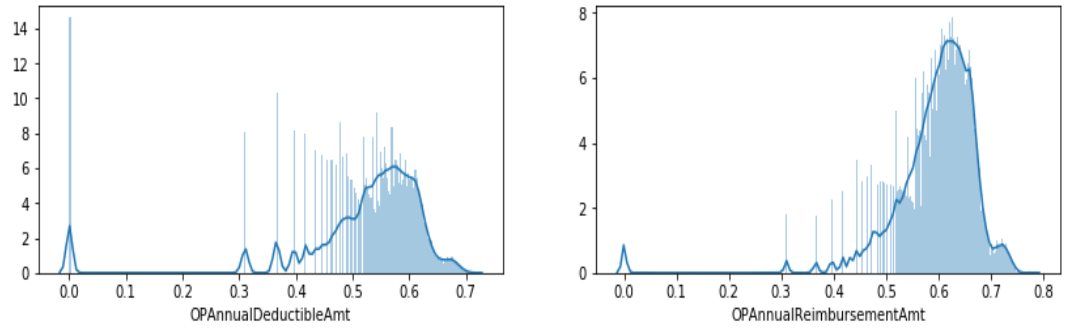
Normalization:

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1

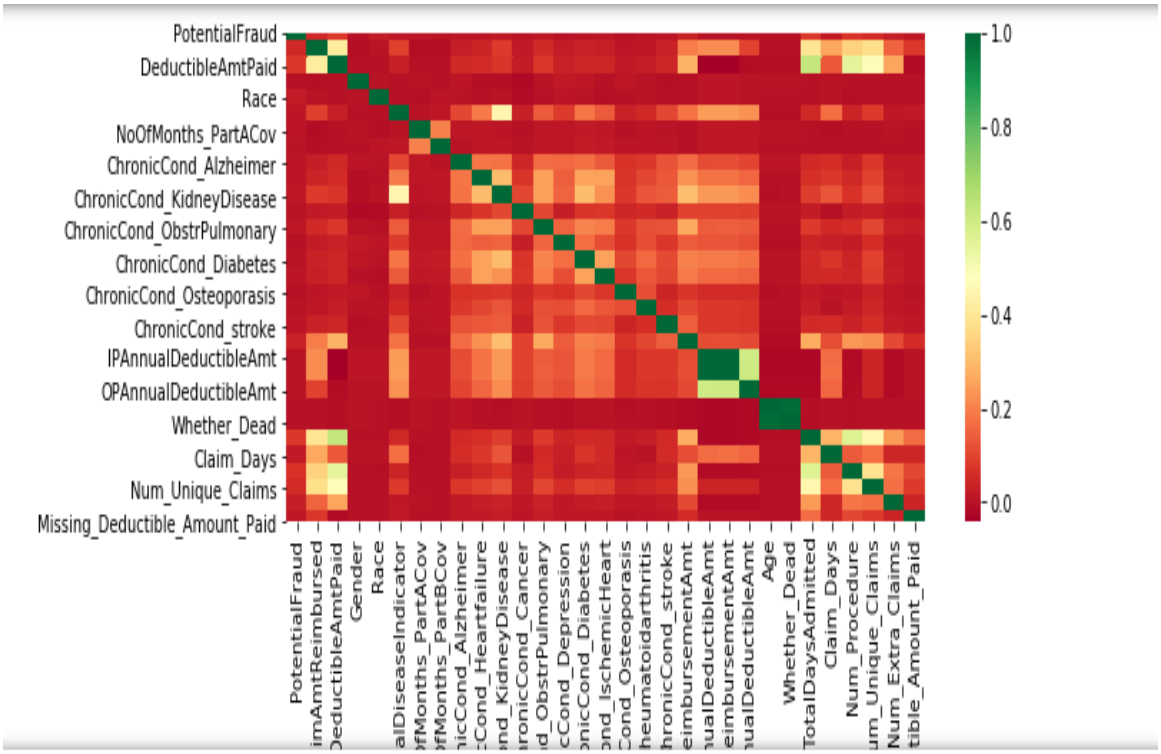
The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values

Some of the columns such as OPAnnualDeductibleAmt, OPAnnualReimbursementAmt etc. were not normalized and were right skewed, therefore we used the Log10 Normalization to Normalized the Features

```
import math
Healthcare_cleaned['OPAnnualDeductibleAmt']=Healthcare_cleaned.apply(lambda x: math.log10(1+ abs(x['OPAnnualDeductibleAmt'])
Healthcare_cleaned['OPAnnualReimbursementAmt']=Healthcare_cleaned.apply(lambda x: math.log10(1+ abs(x['OPAnnualReimbursement
Healthcare_cleaned['IPAnnualDeductibleAmt']=Healthcare_cleaned.apply(lambda x: math.log10(1+ abs(x['OPAnnualReimbursementAmt
Healthcare_cleaned['IPAnnualReimbursementAmt']=Healthcare_cleaned.apply(lambda x: math.log10(1+ abs(x['IPAnnualReimbursement
Healthcare_cleaned['DeductibleAmtPaid']=Healthcare_cleaned.apply(lambda x: math.log10(1+ abs(x['DeductibleAmtPaid'])), axis=
Healthcare_cleaned['InscClaimAmtReimbursed']=Healthcare_cleaned.apply(lambda x: math.log10(1+ abs(x['InscClaimAmtReimbursed'
```



Correlation Matrix with Values:



Findings:

From the Correlation Matrix we depicted that TotalDaysAdmitted, DeductableAmtPaid, Num_Procedure has a strong relationship with Target variable PotentialFraud. Race,Claim_Days,IPAnnualReimbursementAmt has a moderte relationship with the Target variable.

6) Feature Selection:

Feature selection is process of automatically or manually selecting the features that contributes most to the Target Variable

There are various feature extraction techniques in data science which are categorized into 3 parts: Filter methods, Wrapper methods and embedded methods.

Backward Elimination:

In Backward Elimination Method, We feed all the possible features to the model at first and check performance of the model, then iteratively remove the worst performing features one by one till the overall performance of the model comes in acceptable range. The performance metric used here to evaluate feature performance is p value. If the p value is above 0.05 then we remove the feature, else we keep it.

```
#Backward Elimination
cols = list(X.columns)
pmax = 1
while (len(cols)>0):
    pval= []
    X_1 = X[cols]
    X_1 = sm.add_constant(X_1)
    model = sm.OLS(y,X_1).fit()
    pval= pd.Series(model.pvalues.values[1:],index = cols)
    pmax = max(pval)
    feature_with_p_max = pval.idxmax()
    if(pmax>0.05):
        cols.remove(feature_with_p_max)
    else:
        break
selected_features_BE = cols
print(selected_features_BE)
```

Findings:

We used Backward Elimination method to find the most important features by feeding all the features and then iteratively removing features until we get all the best performing features.

We found that Race, ChronicCond_KidneyDisease, ChronicCond_Depression, Age, Whether_Dead, TotalDaysAdmitted, Hospitalized, Unique_Physicians, Types_Of_Physicians, Same_Physician_MR, Num_Procedure as the best performing Features.

RFE(Recursive Feature Elimination)

he Recursive Feature Elimination (RFE) method works by recursively removing attributes and building a model on those attributes that remain. It uses accuracy metric to rank the feature according to their importance. The RFE method takes the model to be used and the number of required features as input. It then gives the ranking of all the variables, 1 being most important. It also gives its support, True being relevant feature and False being irrelevant feature.

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

model = LogisticRegression(solver='lbfgs',max_iter=1000)
#Initializing RFE model
rfe = RFE(model, 7)
#Transforming data using RFE
X_rfe = rfe.fit_transform(X,y)
#Fitting the data to model

model.fit(X_rfe,y)
print(rfe.support_)
print(rfe.ranking_)
```

In the RFE method we created a base model using the Logistic regression and then implement RFE on the base model.

Output:

No. of Features: 7

Selected Features : [False True False False False False False False False False False False False False True False True False True True False True False False True]

Feature Rank: [19 1 8 14 21 3 17 15 4 6 10 13 7 5 11 1 12 1 18 1 1 16 1 2 20 9 1]

Feature	Rank
Gender	19
Race	1
RenalDiseaseIndicator	8
ChronicCond_Alzheimer	14
ChronicCond_Heartfailure	21
ChronicCond_KidneyDisease	3
ChronicCond_Cancer	17
ChronicCond_ObstrPulmonary	15
ChronicCond_Depression	4
ChronicCond_Diabetes	6
ChronicCond_IschemicHeart	10
ChronicCond_Osteoporasis	13
ChronicCond_rheumatoidarthritis	7
ChronicCond_stroke	5
Age	11
Whether_Dead	1
TotalDaysAdmitted	12

Hospitalized	1
Claim_Days	18
Unique_Physicians	1
Types_Of_Physicians	1
One_Physician	16
Same_Physician_MR	1
Num_Procedure	2
Num_Unique_Claims	20
Num_Extra_Claims	9
Missing_Deductible_Amount_Paid	1

Important Features:

Race

Whether_Dead

Hospitalized

Unique Physicians

Types_of_Physicians

Same_Physician_MR

Missing_Deductible_Amount_Paid

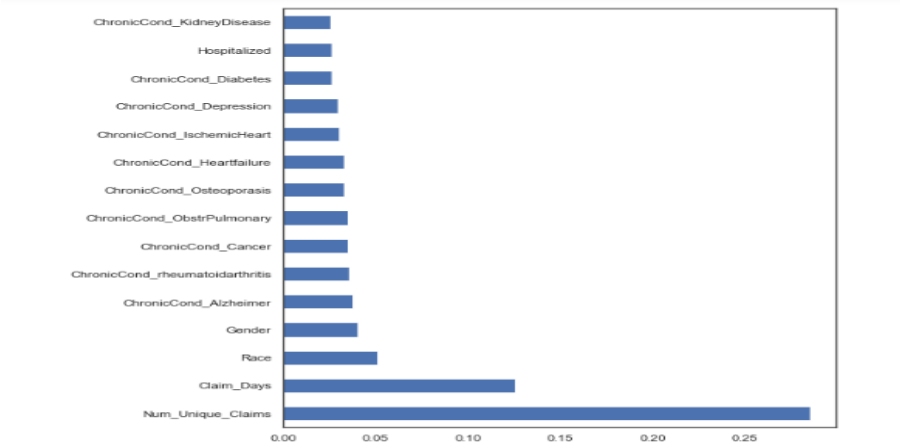
Embedded Method

Create a LassoCV model fit and then we fit the data with all features to the model

```
from sklearn.linear_model import RidgeCV, LassoCV, Ridge, Lasso

reg = LassoCV()
reg.fit(X, y)
print("Best alpha using built-in LassoCV: %f" % reg.alpha_)
print("Best score using built-in LassoCV: %f" %reg.score(X,y))
coef = pd.Series(reg.coef_, index = X.columns)

```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978: FutureWarning:

The default value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to silence this warning.
```

```
Best alpha using built-in LassoCV: 0.000280
Best score using built-in LassoCV: 0.024592
```



Important Features:

Num_Unique_claims

Claim_Days

Race

Gender

ChronicCond_Alzheimer

Selected Features:

We selected the Features that we got from the Backward Elimination method as the important features for model development.

Method Name	**Feature 1**	**Feature 2**	**Feature 3**	**Feature4**	**Feature 5**
Backward Eliminatio	Race	Gender	Whether_Dead	ChronicCond_KidneyDisease	Age
RFE	Race	Gender	Whether_Dea	Hospitalized	Age
EMbedded	Race	Gender	Whether_Dea	ChronicCond_KidneyDisea	Age

Therefore, we selected 15 Features from different Feature selection Methods based in their occurrence in each method.

Model Development:

We have used Logistic Regression model as our Initial Model.

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable.

Method-1: Logistic Regression Model

Splitting the data into Train and Test. We split the data with 70% for Training and 30% as Test with a random state as 42.

Imported the required libraries of the LogisticRegression Model and fit the data to the model.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
X_train,X_test,y_train,y_test = train_test_split(X_standardized,y,test_size=0.3,random_state=42)

#Splitting the data into Train and Test split with 70% data for Training and 30% data as the Test set
```

```
# Finding the shape of Train and Test data
print('X_train :',X_train.shape)
print('y_train :',y_train.shape)

print('X_test :',X_test.shape)
print('y_test :',y_test.shape)
```

```
X_train : (335433, 14)
y_train : (335433,)
X_test : (143758, 14)
y_test : (143758,)
```

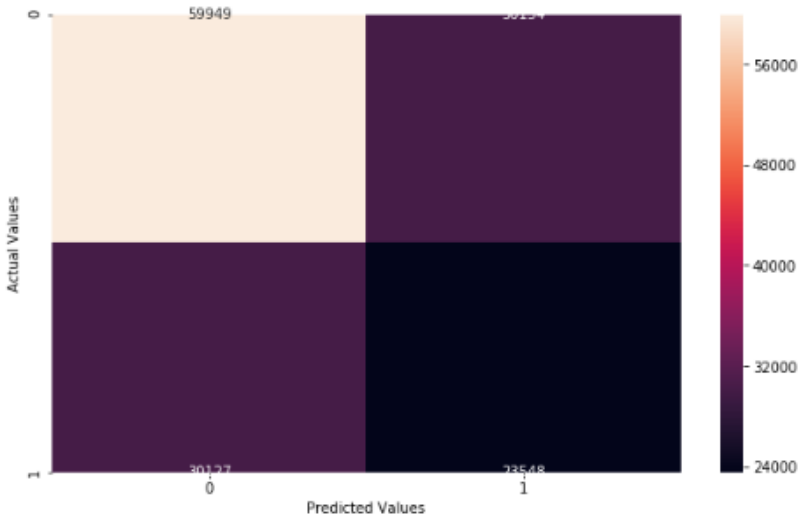
Calculated the Metrices such as Accuracy,Precision, Recall and F1_score of the Model

```
# Calculating the Classification metrices such as Accuracy,Precision,Recall and F1 Score of the Model
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix

# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(y_test, y_pred)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_test, y_pred)
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, y_pred)
print('F1 score: %f' % f1)
# ROC AUC
auc = roc_auc_score(y_test, y_pred)
print('ROC AUC: %f' % auc)
# confusion matrix
matrix = confusion_matrix(y_test, y_pred)
print(matrix)
```

```
Accuracy: 0.580816
Precision: 0.438657
Recall: 0.438714
F1 score: 0.438686
ROC AUC: 0.552100
[[59949 30134]
 [30127 23548]]
```

Accuracy of the Model is used to find the importance of True Positives and True Negatives and F1 Score is used to find importance of False Positives and False Negatives.

Confusion Matrix of Logistic Regression



Method -2: Decision Tree Classifier

The decision tree classifier creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.

In the Decision Tree Classifier we used different parameters like Gini and Entropy for the criterion to select the Tree with Maximum depth of tree between 3,4,5 and minimum split of Tree between the range of 2,3,5.

We got the best model with Gini as a criterion with Maximum Depth of tree as 5 and split size as 2.

```
from sklearn.model_selection import GridSearchCV
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
estimator=  DecisionTreeClassifier()
param_grid= {'criterion':['gini', 'entropy'], # Passsing the Parameters to the Classifier such as Criterion,Max
             'max_depth':[3,4,5],
             'min_samples_split':[2,3,5]
             }
grid_search = GridSearchCV(estimator = estimator, param_grid = param_grid)
grid_search.fit(X_train, y_train) # Fitting the Training data to the classifier/Model
print(grid_search.best_score_)# finding the best score of the model
print(grid_search.best_params_)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978: FutureWarning:

The default value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to silence this warning.
```

```
0.6336943592312027
{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 2}
```

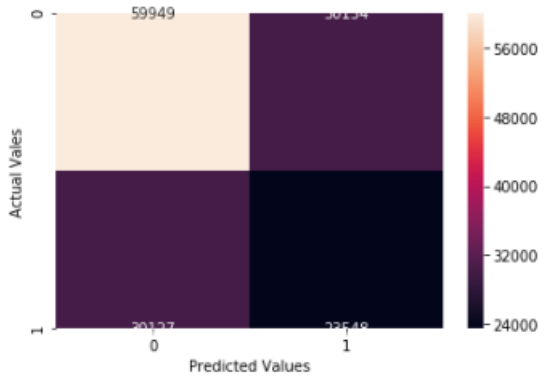
The Accuracy of the model was 63% with decision Tree classifer.

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[59949, 30134],
       [30127, 23548]], dtype=int64)
```

```
cnf_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual Vales'], colnames=['Predicted Values'])
sns.heatmap(cnf_matrix, annot=True,fmt='g')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d982753278>
```



Method -3 : Random Forest Classifier

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest

takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=500,class_weight='balanced',random_state=123,max_depth=6)
rfc.fit(X_train,y_train)  #fit the model
y_pred= rfc.predict(X_test)
cm= confusion_matrix(y_test, y_pred)
print(cm)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %f' % accuracy)
```

```
[[65994 24089]
 [33336 20339]]
Accuracy: 0.600544
```

The random Forest Classifier provided the accuracy of 60% which was better than Logistic Regression, but was low when compared with the Decision Tree Classifier.

Model Assessment:

In the Model Assessment, All the 3 Classification models were compared to find which model perform the best based on the Accuracy and other metrics.

```
names=["Logistic Regression","Decision Tree","Random Forest"]
Classifiers=[LogisticRegression(fit_intercept=False,solver="lbfgs",C=1e9,max_iter=5000),
             DecisionTreeClassifier(),
             RandomForestClassifier(n_estimators=40)]
```

```
model_cols=[]
df=pd.DataFrame(columns=model_cols)
index=0
```

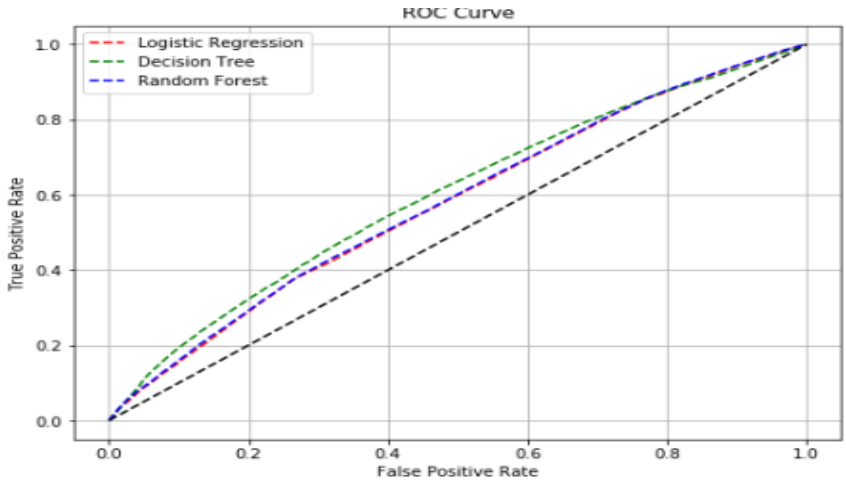
```
for name ,clf in zip (names,Classifiers):
    clf.fit(X_train,y_train)
    df.loc[index,"classifiers"]=name
    df.loc[index,"Accuracy"]=accuracy_score(y_test,clf.predict(X_test))
    df.loc[index,"Precision"]=precision_score(y_test,clf.predict(X_test),average='macro')
    df.loc[index,"Recall"]=recall_score(y_test,clf.predict(X_test),average='macro')
    df.loc[index,"F1"]=f1_score(y_test,clf.predict(X_test),average='macro')
    df.loc[index,"ROC AUC"]=roc_auc_score(y_test,clf.predict(X_test),average='macro')
    index+=1
    df=df.sort_values(by=["Accuracy"],ascending=False)
    df=df.reset_index(drop=True)

df
```

	classifiers	Accuracy	Precision	Recall	F1	ROC AUC
0	Decision Tree	0.632855	0.587303	0.552586	0.535993	0.552586
1	Random Forest	0.629475	0.583356	0.556390	0.545670	0.556390
2	Logistic Regression	0.584343	0.552771	0.551884	0.552195	0.551884

When the Assessment of each of the 3 Models was done, we found that the Decision Tree model performed well with good Accuracy when compared to the Random Forest and Logistic Regression Model.

Model Assessment using the ROC Curve:

The ROC curve was used to find the performance of every model. The graph shows that the Decision Tree Classifier had highest performance when compared to Logistic and Random Forest Classifiers



Conclusion:

- After comparing the models , we found that
- Decision Tree Classifier performance was good with highest accuracy(56%) among all the 3 models
- The logistic Regression Model had an Accuracy of 58% with the similar Precision and Recall score of 55%
- The ROC for all the models was quite similar with 55%.

Interpretations:

1) Adding more fraud data to the training dataset help in predicting unseen fraudulent behaviour time to time.

2) Ensembling methods with parameter tuning can be used to improve performance of the models.

3) Most of the Fraud claims registered were for the patients between the age group of 18-50 Years of age

4) Most of the Fraud claims were found to be with the involvement of Physicians(Attending, Operating and Other Physicians

5) Deep learning techniques such as Activation Function (Relu) can be used to for better performance of the model

References:

- https://www.kaggle.com/code/akshaypaliwal709/payee-provider-fraud-detection-analysis
- https://www.kaggle.com/code/rohitrox/medical-provider-fraud-detection
- https://medium.com/analytics-vidhya/healthcare-provider-fraud-detection-analysis-using-machine-learning-81ebf09ed955
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- https://www.datacamp.com/community/tutorials/random-forests-classifier-python