Healthcare Provider Fraud Detection Analysis

1.Business Problem:

1.1. What is healthcare fraud?

Health insurance companies provide coverage for medical expenses to the policy holder depending on the health insurance plan chosen by the patient which the patient is eligible to claim.These amounts may vary depending upon diagnosis and treatement done on the patient and depending upon doctor and hospital.Many healthcare providers settle huge amounts for patients. But some insured individuals or the provider of health services attempt to make fake claims by giving false claim details which is considered a medical crime

1.2. Problem Statement

Insurance companies are forced to provide benefits for fake claims unknowingly and face many problems in providing benefits to real claims and also those companies are heavily impacted due these bad practices. This has become a serious issue and insurance providers have started finding ways to detect whether the claim is fraud or not. The goal of this case study is to **accurately predict claims into fraudulent or real claims** which will save a very huge amount of money from frauds and help those in real need.

1.3. Datasource Link

https://www.kaggle.com/rohitrox/healthcare-provider-fraud-detection-analysis This dataset is taken from kaggle.

1.4. Business objectives and constraints

1. Rate of misclassification should be very less
2. No low latency requirement

2. Mapping to Machine Learning Problem

2.1. Type of ML problem

This is a binary classification problem. We need to classify datapoint into fraudulent or real claim

2.2. Performance Metric

1. f1 score
2. macro f1 score (as there is slight imbalance of class 64:36 , using macro will be useful. it give equal importance to both class)
3. Accuracy

Deployment of model in heroku

https://healthcare-prediction.herokuapp.com/

Importing important packages

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
from collections import Counter
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn import metrics
from sklearn.metrics import roc_curve, f1_score, confusion_matrix
from prettytable import PrettyTable
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

In [2]:

```
# loading csv file

train_outpatient = pd.read_csv('Train_Outpatientdata.csv')
train_inpatient = pd.read_csv('Train_Inpatientdata.csv')
train_beneficiary = pd.read_csv('Train_Beneficiarydata.csv')
print('train_outpatient shape: ',train_outpatient.shape)
print('train_inpatient shape: ',train_inpatient.shape)
print('train_beneficiary shape: ',train_beneficiary.shape)
```

```
train_outpatient shape:  (517737, 27)
train_inpatient shape:  (40474, 30)
train_beneficiary shape:  (138556, 25)
```

In [3]:

```
# loading target train file

train_target = pd.read_csv('Train.csv')
print('train_target shape: ',train_target.shape)
```

```
train_target shape:  (5410, 2)
```

In [6]:

```
print('train_outpatient columns are : ',train_outpatient.columns.values)
print(' '*100)
print('-'*100)
print(' '*100)
print('train_inpatient columns are : ',train_inpatient.columns.values)
print(' '*100)
print('-'*100)
print(' '*100)
print('train_beneficiary columns are : ',train_beneficiary.columns.values)
```

```
train_outpatient columns are :  ['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']

----------------------------------------------------------------------------------------------------

train_inpatient columns are :  ['BeneID' 'ClaimID' 'ClaimStartDt' 'ClaimEndDt' 'Provider'
 'InscClaimAmtReimbursed' 'AttendingPhysician' 'OperatingPhysician'
 'OtherPhysician' 'AdmissionDt' 'ClmAdmitDiagnosisCode' 'DeductibleAmtPaid'
 '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']

-----------------------------------------------------------------------------------------------------

train_beneficiary columns are :  ['BeneID' 'DOB' 'DOD' 'Gender' 'Race' 'RenalDiseaseIndicator' 'St
ate'
 'County' 'NoOfMonths_PartACov' 'NoOfMonths_PartBCov'
 'ChronicCond_Alzheimer' 'ChronicCond_Heartfailure'
 'ChronicCond_KidneyDisease' 'ChronicCond_Cancer'
 'ChronicCond_ObstrPulmonary' 'ChronicCond_Depression'
 'ChronicCond_Diabetes' 'ChronicCond_IschemicHeart'
 'ChronicCond_Osteoporasis' 'ChronicCond_rheumatoidarthritis'
 'ChronicCond_stroke' 'IPAnnualReimbursementAmt' 'IPAnnualDeductibleAmt'
 'OPAnnualReimbursementAmt' 'OPAnnualDeductibleAmt']
```

merging train_inpatient, train_outpatient data

In [4]:

```
## https://www.kaggle.com/dimagmehanic/mc-termpaper
## merging df without repeting same columns
## left_on and right_on use that columns as keys to merge.
## left_on will take columns from left dataframe and right_on will take columns from right datafra
me as keys.
## so columns with same name will be used as keys ( so repeting of columns will be avoided ) and r
emaining columns will be merged. inpatient - 30 columns, outpatient - 27 columns. merged df shape
will be with 30 columns

final_data = pd.merge(train_inpatient, train_outpatient, left_on = [ col for col in
train_outpatient.columns if col in train_inpatient.columns], \
                    right_on = [ col for col in train_outpatient.columns if col in train_inpatient.
olumns], how = 'outer')
final_data.shape
```

Out[4]:

```
(558211, 30)
```

In [5]:

```
final_data.head()
```

Out[5]:

	BeneID	ClaimID	ClaimStartDt	ClaimEndDt	Provider	InscClaimAmtReimbursed	AttendingPhysician	Operating
0	BENE11001	CLM46614	2009-04-12	2009-04-18	PRV55912	26000.0	PHY390922	NaN
1	BENE11001	CLM66048	2009-08-31	2009-09-02	PRV55907	5000.0	PHY318495	PHY31849
2	BENE11001	CLM68358	2009-09-17	2009-09-20	PRV56046	5000.0	PHY372395	NaN
3	BENE11011	CLM38412	2009-02-14	2009-02-22	PRV52405	5000.0	PHY369659	PHY39296
4	BENE11014	CLM63689	2009-08-13	2009-08-30	PRV56614	10000.0	PHY379376	PHY39825

5 rows × 30 columns

merging final_data, train_beneficiary data

In [6]:

```
final_data = pd.merge(final_data,train_beneficiary,how='inner',on='BeneID' )
```

In [7]:

```
final_data.shape
```

Out[7]:

```
(558211, 54)
```

merging final_data, train_target data

In [8]:

```
final_data = pd.merge(final_data,train_target,how='outer',on='Provider')
```

In [9]:

```
final_data.shape
```

Out[9]:

```
(558211, 55)
```

In [10]:

```
final_data.columns
```

Out[10]:

```
Index(['BeneID', 'ClaimID', 'ClaimStartDt', 'ClaimEndDt', 'Provider',
       'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
       'OtherPhysician', 'AdmissionDt', 'ClmAdmitDiagnosisCode',
       'DeductibleAmtPaid', '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', 'Gender', 'Race',
       'RenalDiseaseIndicator', 'State', 'County', 'NoOfMonths_PartACov',
       'NoOfMonths_PartBCov', 'ChronicCond_Alzheimer',
       'ChronicCond_Heartfailure', 'ChronicCond_KidneyDisease',
       'ChronicCond_Cancer', 'ChronicCond_ObstrPulmonary',
       'ChronicCond_Depression', 'ChronicCond_Diabetes',
       'ChronicCond_IschemicHeart', 'ChronicCond_Osteoporasis',
       'ChronicCond_rheumatoidarthritis', 'ChronicCond_stroke',
       'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
       'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt', 'PotentialFraud'],
      dtype='object')
```

In [10]:

```
final_data.head()
```

Out[10]:

	BeneID	ClaimID	ClaimStartDt	ClaimEndDt	Provider	InscClaimAmtReimbursed	AttendingPhysician	Operatin
0	BENE11001	CLM46614	2009-04-12	2009-04-18	PRV55912	26000.0	PHY390922	NaN
1	BENE16973	CLM565430	2009-09-06	2009-09-06	PRV55912	50.0	PHY365867	PHY3271
2	BENE17521	CLM34721	2009-01-20	2009-02-01	PRV55912	19000.0	PHY349293	PHY3708
3	BENE21718	CLM72336	2009-10-17	2009-11-04	PRV55912	17000.0	PHY334706	PHY3347
4	BENE22934	CLM73394	2009-10-25	2009-10-29	PRV55912	13000.0	PHY390614	PHY3236

5 rows × 55 columns

3. Exploratory Data Analysis

3.1. Percentage of class label

PotentialFraud class distribution in outpatient data using bar plot

In [11]:

```
## merging train_outpatient, train_target

train_outpatient = pd.merge(train_outpatient,train_target,how='outer',on='Provider')


## number of points in fraud ('yes') and non-fraud (no) class

train_outpatient['PotentialFraud'].value_counts()
```

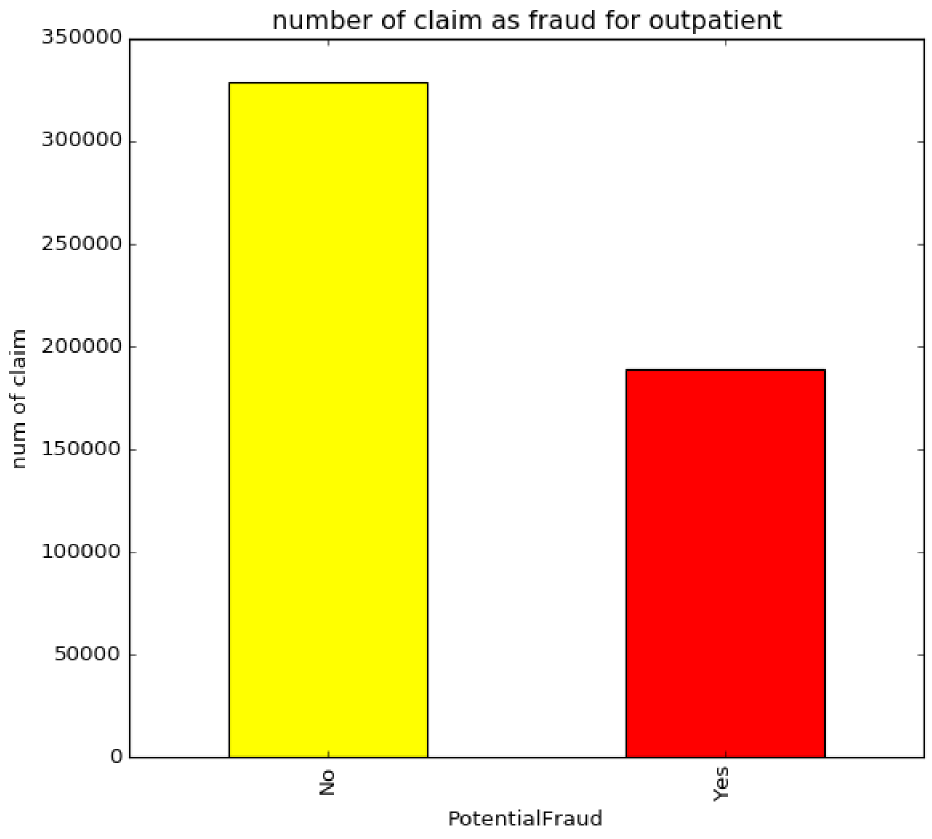
Out[11]:

```
No     328697
Yes    189438
Name: PotentialFraud, dtype: int64
```

In [12]:

```
class_count = train_outpatient['PotentialFraud'].value_counts().to_frame()

""" calculating percentage of yes and no class  """

percentage_yes = (class_count['PotentialFraud']['Yes']/(class_count['PotentialFraud']['Yes']+class_
count['PotentialFraud']['No']))*100
print('percentage of fraud class : ',percentage_yes)
percentage_no = (class_count['PotentialFraud']['No']/(class_count['PotentialFraud']['Yes']+class_co
unt['PotentialFraud']['No']))*100
print('percentage of non fraud class : ',percentage_no)

train_outpatient['PotentialFraud'].value_counts().plot(kind='bar',color=('yellow','red'),title='num
ber of claim as fraud for outpatient',figsize=(8,7))
plt.xlabel('PotentialFraud')
plt.ylabel('num of claim')
plt.show()
```

```
percentage of fraud class :  36.5615138912
percentage of non fraud class :  63.4384861088
```



conclusion

1. In train outpatient file there are 328697 non fraud, 189438 fraud claim
2. percentage of yes class is 36.5615138912, no class is 63.4384861088
3. this show most of outpatient claim is non fraud
4. this can be reason because people are more concentrated on doing fraud claim for inpatient

PotentialFraud class distribution in inpatient data using bar plot

In [13]:

```
## merging train_inpatient, train_target

train_inpatient = pd.merge(train_inpatient,train_target,how='outer',on='Provider')

## number of points in fraud ('yes') and non-fraud (no) class

train_inpatient['PotentialFraud'].value_counts()
```

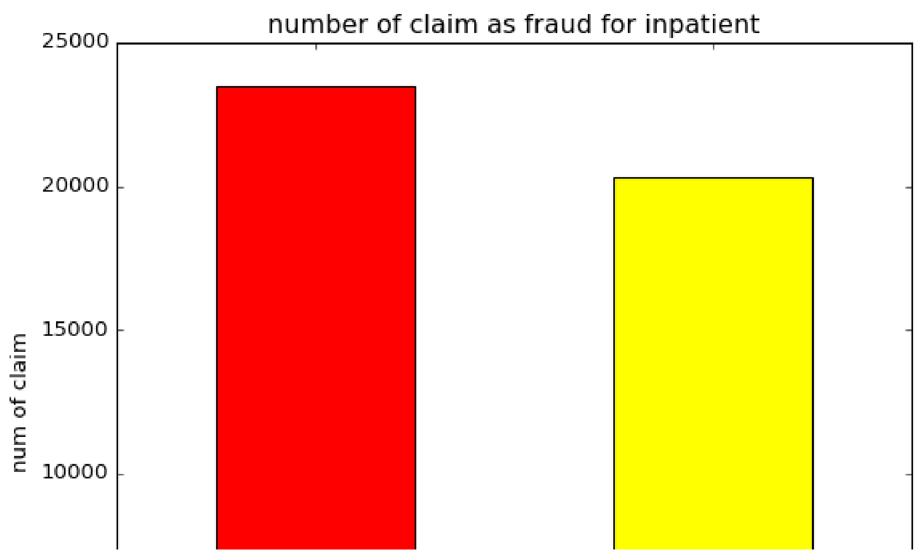
Out[13]:

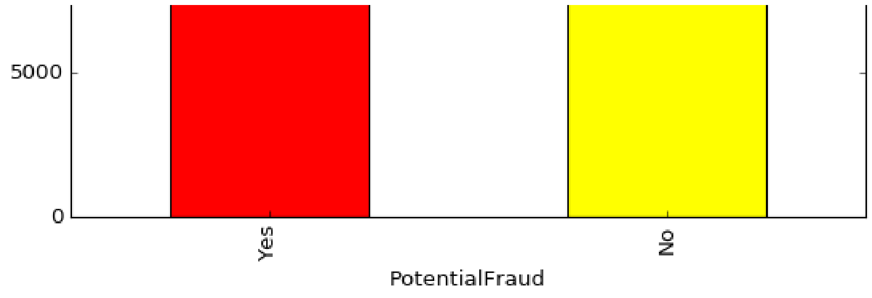
```
Yes    23468
No     20324
Name: PotentialFraud, dtype: int64
```

In [14]:

```
class_count = train_inpatient['PotentialFraud'].value_counts().to_frame()

""" calculating percentage of yes and no class  """

percentage_yes = (class_count['PotentialFraud']['Yes']/(class_count['PotentialFraud']['Yes']+class_
count['PotentialFraud']['No']))*100
print('percentage of fraud class : ',percentage_yes)
percentage_no = (class_count['PotentialFraud']['No']/(class_count['PotentialFraud']['Yes']+class_co
unt['PotentialFraud']['No']))*100
print('percentage of non fraud class : ',percentage_no)

train_inpatient['PotentialFraud'].value_counts().plot(kind='bar',color=('red','yellow'),title='numb
er of claim as fraud for inpatient',figsize=(8,7))
plt.xlabel('PotentialFraud')
plt.ylabel('num of claim')
plt.show()
```

```
percentage of fraud class :  53.5896967483
percentage of non fraud class :  46.4103032517
```





conclusion

1. In train inpatient file there are 20324 non fraud, 23468 fraud claim
2. percentage of yes class is 53.5896967483, no class is 46.4103032517
3. this show people are more concentrated on doing fraud claim for inpatient because more amount can be claimed for inpatient

3.2. AttendingPhysician feature

3.2.1.Univariate analysis - Analysing AttendingPhysician feature with bar plot for outpatient

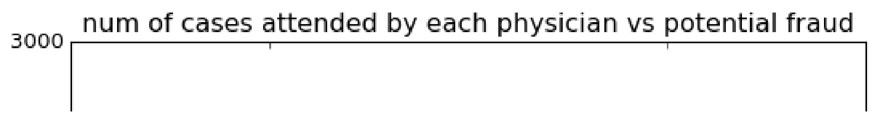
In [15]:

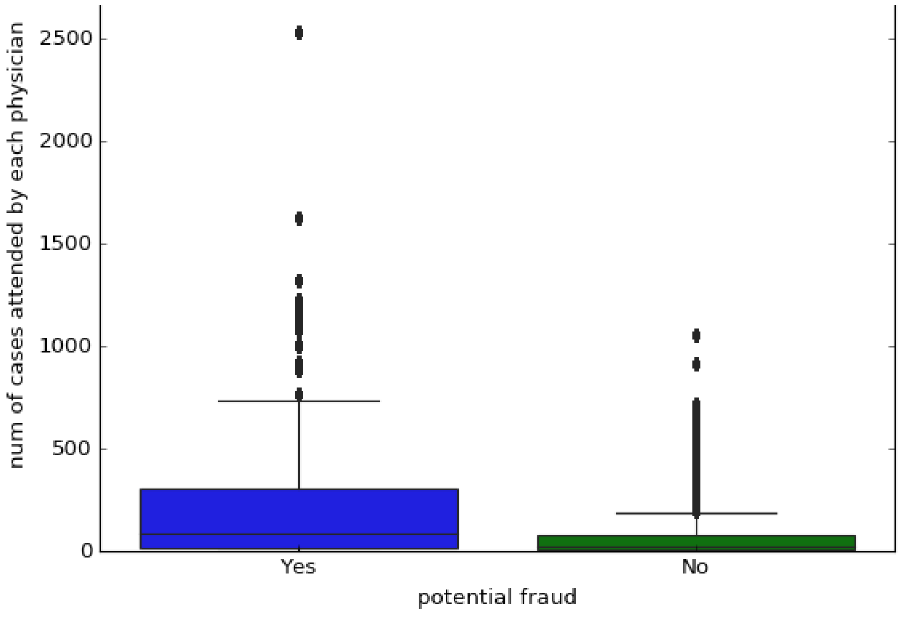
```
def count_(data,column,ylabel,title):
    """ calculating num of cases attended by physician and plotting vs fraud """

    ## counting number of cases for each physician and saving in dict

    count = data[column].value_counts().to_dict()

    col = str(column)+'count'            ## giving column name to be added

    ## making new column for number of cases for each physician as 'AttendingPhysiciancount' by ma
pping count dict to train_outpatient['AttendingPhysician']

    data[col]=data[column].map(count)

    ## top 5 attended physician
    count = data[column].value_counts()[0:5]
    print('top 5',column,'are: ',count)

    ## plot
    sns.boxplot(x='PotentialFraud',y=col, data=data)
    plt.xlabel('potential fraud')
    plt.ylabel(ylabel)
    plt.title(title)
    plt.show()
```

In [16]:

```
count_(train_outpatient,'AttendingPhysician','num of cases attended by each physician','num of cas
es attended by each physician vs potential fraud')
```

```
top 5 AttendingPhysician are:  PHY330576    2534
PHY350277    1628
PHY412132    1321
PHY423534    1223
PHY314027    1200
Name: AttendingPhysician, dtype: int64
```





conclusion

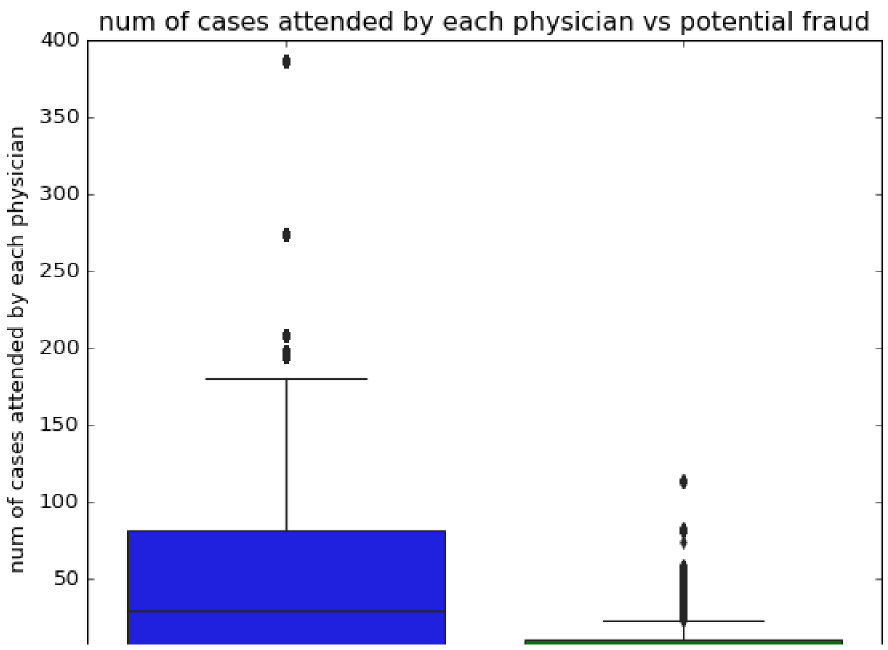
1. The most attended physician is PHY330576 attending 2534 cases.
2. If a physician has attended more number of case, then the physician involved in making fraud claim is more
3. The top 5 attended physician are PHY330576,PHY350277,PHY412132,PHY423534,PHY314027
4. The top attended physician's make more fraud claim
5. The physician who attended only very few cases make non fraud claim

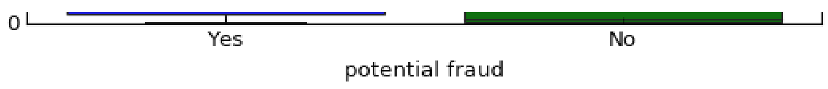
3.2.2. Univariate analysis - Analysing AttendingPhysician feature with bar plot for inpatient

In [17]:

```
count_(train_inpatient,'AttendingPhysician','num of cases attended by each physician','num of case
s attended by each physician vs potential fraud')
```

```
top 5 AttendingPhysician are:  PHY422134    386
PHY341560    274
PHY315112    208
PHY411541    198
PHY362864    195
Name: AttendingPhysician, dtype: int64
```





conclusion

1. The most attended physician is PHY422134 attending 386 cases.
2. If a physician has attended more number of case, then the physician involved in making fraud claim is more
3. The top 5 attended physician are PHY422134,PHY341560,PHY315112,PHY411541,PHY362864
4. The top attended physician's make more fraud claim
5. The physician who attended only very few cases make non fraud claim

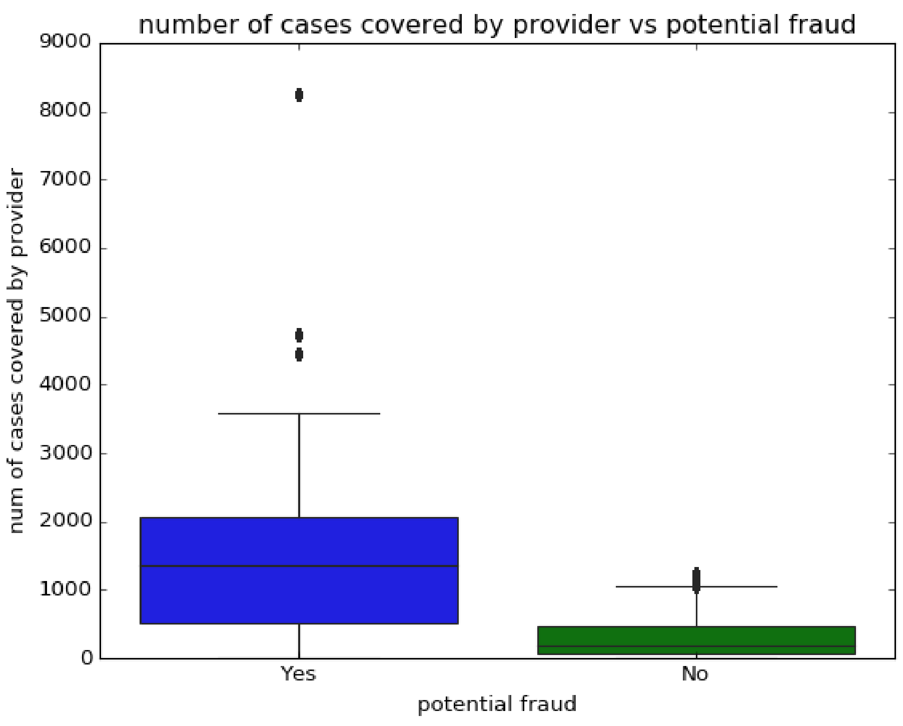
3.3. Provider feature

3.3.1. Univariate analysis - Analysing Provider feature with bar plot for outpatient

In [18]:

```
count_(train_outpatient,'Provider','num of cases covered by provider','number of cases covered by
provider vs potential fraud')
```

```
top 5 Provider are:  PRV51459    8240
PRV53797    4739
PRV51574    4444
PRV53918    3588
PRV54895    3433
Name: Provider, dtype: int64
```



In [104]:

```
""" function to calculate number of yes and no class present for provider """
def count_provider(provider):
    yes=0
    no=0
    for i in range(len(train_inpatient)):  ## use len(train_outpatient) for outpatient provider
        count=''
        if train_inpatient['Provider'][i] == provider:
            count = count+train_inpatient['PotentialFraud'][i]
            if count=='Yes':
                yes=yes+1
            else:
                no=no+1
    return yes,no
```

In [146]:

```
## checking class present for top provider

print('top 1 provider has classes (yes,no) : ',count_provider('PRV51459'))
print('top 2 provider has classes (yes,no) : ',count_provider('PRV53797'))
print('top 93rd provider has classes (yes,no) : ',count_provider('PRV51393'))
print('top 94th provider has classes (yes,no) : ',count_provider('PRV54690'))
```

```
top 1 provider has classes (yes,no) :  (8240, 0)
top 2 provider has classes (yes,no) :  (4739, 0)
top 93rd provider has classes (yes,no) :  (842, 0)
top 94th provider has classes (yes,no) :  (0, 840)
```

conclusion

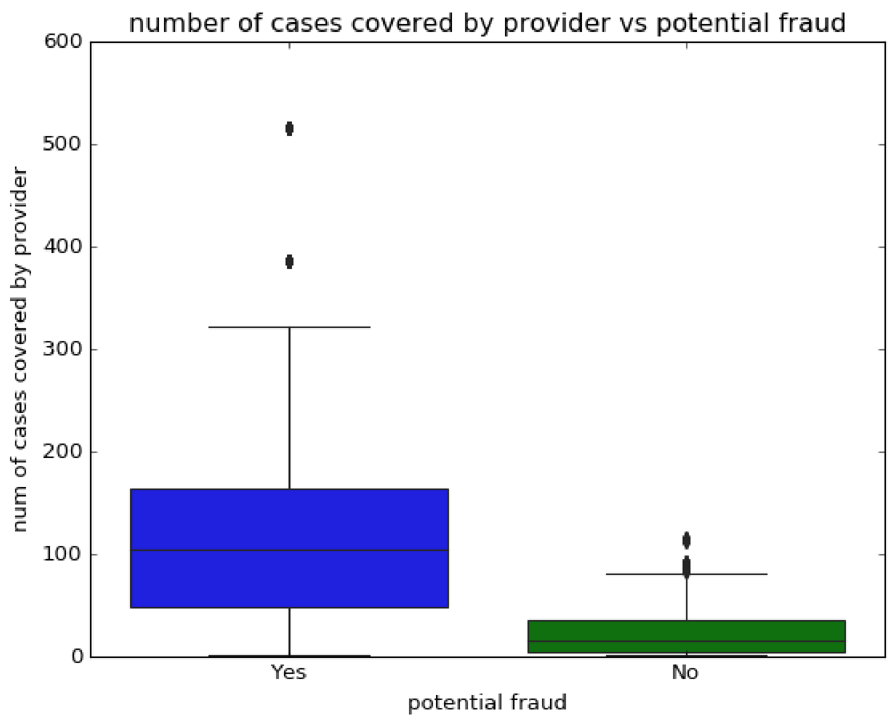
1. The top provider is PRV51459 with 8240 cases
2. here provider those made non fraud claim give coverage for only few cases
3. provider that made coverage for many cases show more fraud claim made
4. The top 5 attended provider are PRV51459,PRV53797,PRV51574,PRV53918,PRV54895
5. top 93 provider has made fraud claim , indicating top provider have more probabilty for making fraud claim

3.3.2. Univariate analysis - Analysing Provider feature with bar plot for inpatient

In [19]:

```
count_(train_inpatient,'Provider','num of cases covered by provider','number of cases covered by p
rovider vs potential fraud')
```

```
top 5 Provider are:  PRV52019    516
PRV55462    386
PRV54367    322
PRV53706    282
PRV55209    275
Name: Provider, dtype: int64
```



In [27]:

```
## checking class present for top provider

print('top 1 provider has classes (yes,no) : ',count_provider('PRV52019'))
print('top 2 provider has classes (yes,no) : ',count_provider('PRV55462'))
print('top 83rd provider has classes (yes,no) : ',count_provider('PRV51542'))
print('top 84th provider has classes (yes,no) : ',count_provider('PRV55978'))
```

```
top 1 provider has classes (yes,no) :  (516, 0)
top 2 provider has classes (yes,no) :  (386, 0)
top 83rd provider has classes (yes,no) :  (93, 0)
top 84th provider has classes (yes,no) :  (0, 92)
```

conclusion

1. The top provider is PRV52019 with 516 cases
2. here provider those made non fraud claim give coverage for only few cases
3. provider that made coverage for many cases show more fraud claim made
4. The top 5 attended provider are PRV52019,PRV55462,PRV54367,PRV53706,PRV55209
5. top 83 provider has made fraud claim , indicating top provider have more probabilty for making fraud claim

3.3.3. Bivariate analysis - pair plot on AttendingPhysician and Provider

In [20]:

```
## checking class label for top 5 Physician and Provider that occur together

## for all row, if fraud take that physician and provider
prophy=[]
for i in range(len(final_data)):
    if final_data['PotentialFraud'][i]=='Yes':
        prophy.append((final_data['AttendingPhysician'][i],final_data['Provider'][i]))

## counting num of time physician and provider occur together
prophy = dict(Counter(prophy))

## sorting descen on values of dict
sortPhyPro = sorted(prophy.items(), key=lambda x: x[1], reverse=True)

## getting top physician and provider and their class label
labelProPhy = []
for i in range(len(sortPhyPro[0:5])):
    df = final_data.loc[(final_data['AttendingPhysician'] == sortPhyPro[i][0][0]) & (final_data['Pr
ovider'] == sortPhyPro[i][0][1])]
    labelProPhy.append(((sortPhyPro[i][0][0],sortPhyPro[i][0][1]),df['PotentialFraud'].values[i]))

print('Top 5 Physician and Provider that occur together has class label: ')
for i in labelProPhy:
    print(i)
```

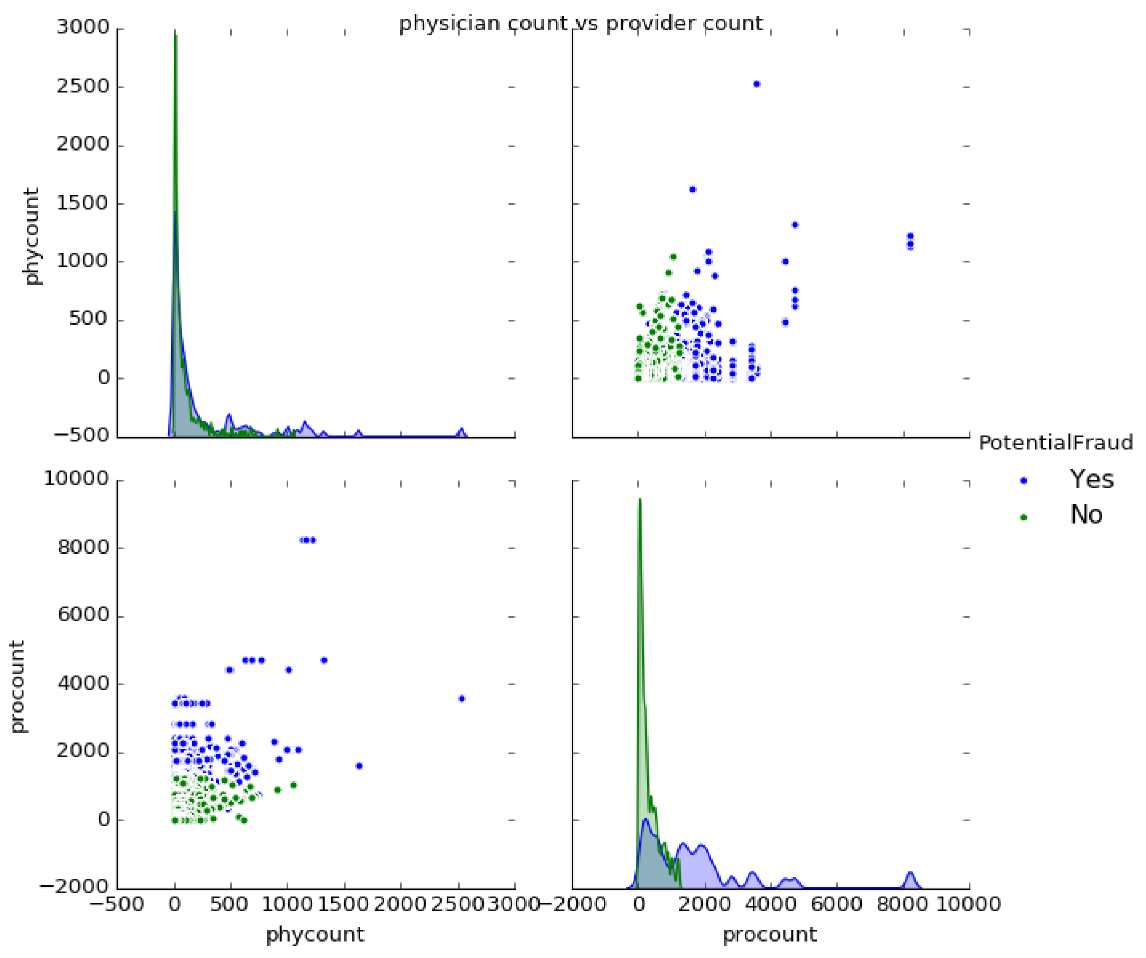
```
Top 5 Physician and Provider that occur together has class label:
(('PHY330576', 'PRV53918'), 'Yes')
(('PHY350277', 'PRV51567'), 'Yes')
(('PHY412132', 'PRV53797'), 'Yes')
(('PHY423534', 'PRV51459'), 'Yes')
(('PHY314027', 'PRV51459'), 'Yes')
```

In [21]:

```
## num of cases attended by phy
attendingPhysician_count = final_data['AttendingPhysician'].value_counts().to_dict()
print('num of unique physician are: ',len(attendingPhysician_count))

## num of cases covered by pro
provider_count = final_data['Provider'].value_counts().to_dict()
print('num of unique provider are :',len(provider_count))
```

```
## adding phycount column by mapping num of cases attended by phy with AttendingPhysician
final_data['phycount']=final_data['AttendingPhysician'].map(attendingPhysician_count)

## adding procount column by mapping num of cases covered by pro with Provider
final_data['procount']=final_data['Provider'].map(provider_count)

sns.pairplot(final_data,hue='PotentialFraud',vars=['phycount','procount'],height=3.8)
plt.suptitle(' physician count vs provider count ')
plt.show()
```

```
num of unique physician are:  82063
num of unique provider are : 5410
```



conclusion

1. Top 5 Physician and Provider that occur together has class label:
   (('PHY330576', 'PRV53918'), 'Yes')
   (('PHY350277', 'PRV51567'), 'Yes')
   (('PHY412132', 'PRV53797'), 'Yes')
   (('PHY423534', 'PRV51459'), 'Yes')
   (('PHY314027', 'PRV51459'), 'Yes')
   provider PRV51459 even occur twice in top 5, so that provider may be highly making fraud claim
2. This show, if top attended physician and top provider occur together, they are supposed to make fraud claim
3. so if famous physician and provider occur together, it is good to check the claim in more depth
4. from pair plot, it show that top attended and physician tend to make fraud claim

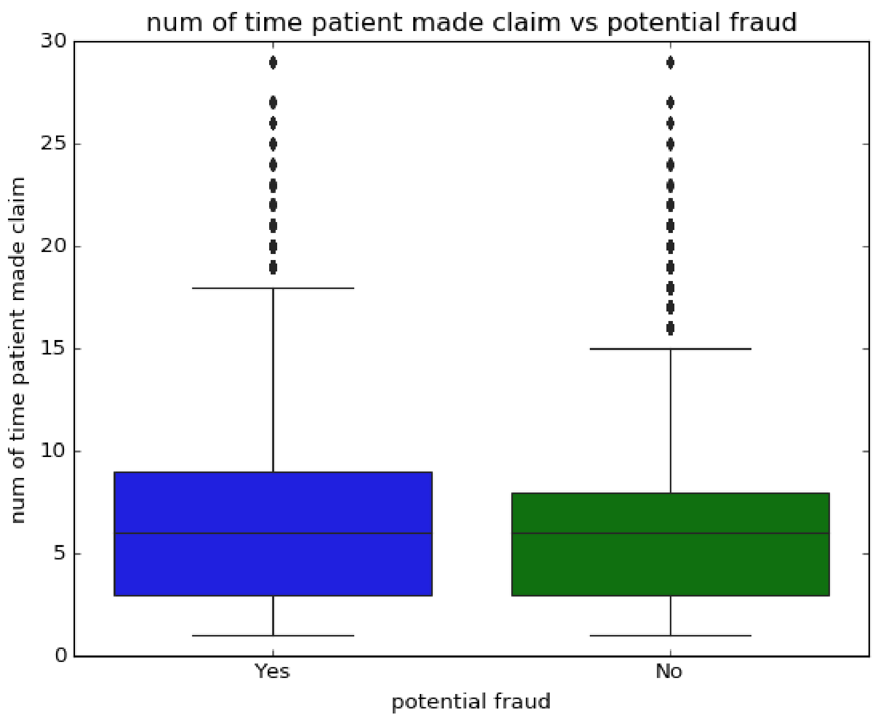
3.4. BeneID feature

3.4.1. Univariate analysis - Analysing BeneID feature with bar plot for outpatient

In [22]:

```
count_(train_outpatient,'BeneID','num of time patient made claim','num of time patient made claim
vs potential fraud')
```

```
top 5 BeneID are:  BENE42721      29
BENE118316     29
BENE143400     27
BENE59303      27
BENE63544      27
Name: BeneID, dtype: int64
```



conclusion

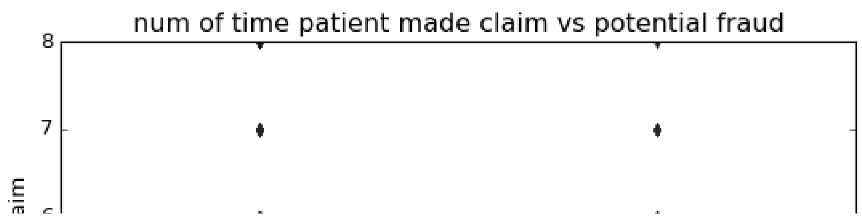
1. The most frequent patient BeneID is BENE42721 with 29 claim made
2. number of claim made by most patient is between 1 to 9
3. The top 5 attended patient are BENE42721,BENE118316,BENE59303,BENE63544,BENE63504
4. patient who already made claim between 2 to 9 time tend to make more fraud claim

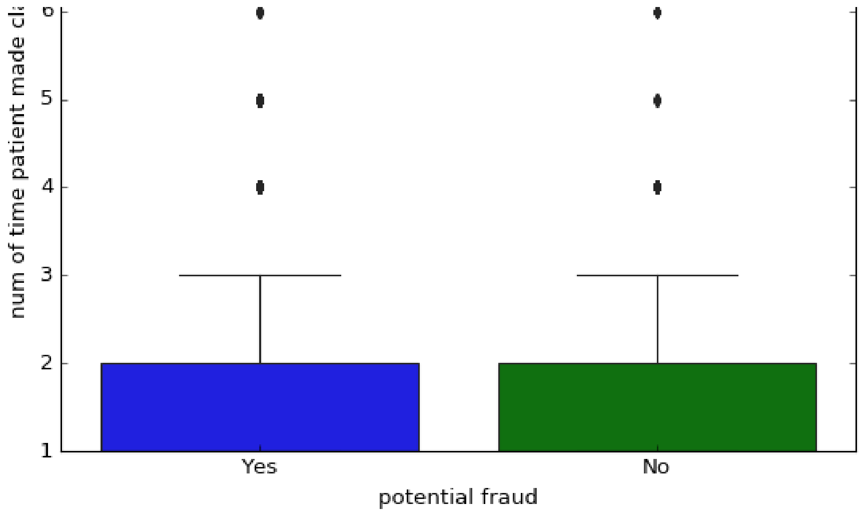
3.4.2. Univariate analysis - Analysing BeneID feature with bar plot for inpatient

In [23]:

```
count_(train_inpatient,'BeneID','num of time patient made claim','num of time patient made claim v
s potential fraud')
```

```
top 5 BeneID are:  BENE134170     8
BENE64791      7
BENE119457     7
BENE121796     7
BENE62091      7
Name: BeneID, dtype: int64
```





conclusion

1. The most frequent patient BeneID is BENE134170 with 8 claim made
2. number of claim made by most patient is only 1 or 2
3. The top 5 attended patient are BENE134170,BENE121796,BENE117116,BENE119457,BENE62091
4. inpatient will be occuring only for particular treatment , so analysing based on num of claim made by patient may not be useful

3.4.3. Bivariate analysis - pair plot on AttendingPhysician and BeneID

In [24]:

```
## checking class label for top 5 Physician and BeneID that occur together

## for all row, if fraud take that physician and BeneID
benephy=[]
for i in range(len(final_data)):
    if final_data['PotentialFraud'][i]=='Yes':
        benephy.append((final_data['AttendingPhysician'][i],final_data['BeneID'][i]))

## counting num of time physician and BeneID occur together
benephy = dict(Counter(benephy))

## sorting descen on values of dict
sortPhyBene = sorted(benephy.items(), key=lambda x: x[1], reverse=True)

## getting top physician and BeneID and their class label
labelPhyBene = []
for i in range(len(sortPhyBene[0:5])):
    df = final_data.loc[(final_data['AttendingPhysician'] == sortPhyBene[i][0][0]) & (final_data['B
eneID'] == sortPhyBene[i][0][1])]
    labelPhyBene.append(((sortPhyBene[i][0][0],sortPhyBene[i][0][1]),df['PotentialFraud'].values[0]
))


print('Top 5 Physician and BeneID that occur together has class label: ')
for i in labelPhyBene:
    print(i)
```

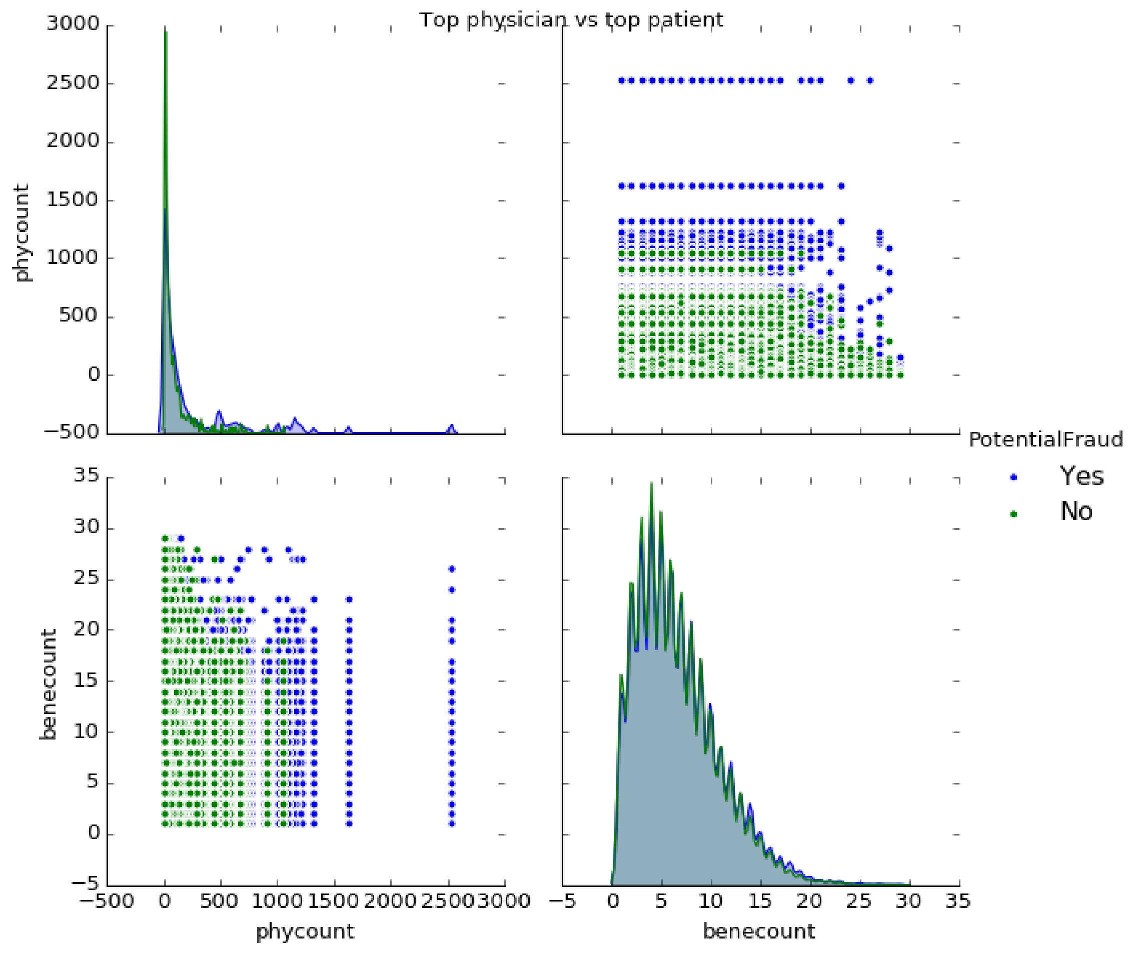
```
Top 5 Physician and BeneID that occur together has class label:
(('PHY339042', 'BENE66093'), 'Yes')
(('PHY313322', 'BENE118316'), 'Yes')
(('PHY385072', 'BENE41087'), 'Yes')
(('PHY385072', 'BENE82133'), 'Yes')
(('PHY344367', 'BENE155227'), 'Yes')
```

In [25]:

```
## num of cases attended by phy

attendingPhysician_count = final_data['AttendingPhysician'].value_counts().to_dict()
```

```
print('num of unique physician are: ',len(attendingPhysician_count))

## num of cases attended by BeneID
bene_count = final_data['BeneID'].value_counts().to_dict()
print('num of unique bene are :',len(bene_count))

## adding phycount column by mapping num of cases attended by phy with AttendingPhysician
final_data['phycount']=final_data['AttendingPhysician'].map(attendingPhysician_count)

## adding phycount column by mapping num of cases attended by patient with BeneID
final_data['benecount']=final_data['BeneID'].map(bene_count)

sns.pairplot(final_data,hue='PotentialFraud',vars=['phycount','benecount'],height=3.8)
plt.suptitle(' Top physician vs top patient \n\n')
plt.show()
```

```
num of unique physician are:  82063
num of unique bene are : 138556
```



conclusion

1. Top 5 Physician and BeneID that occur together has class label:
   (('PHY339042', 'BENE66093'), 'Yes')
   (('PHY313322', 'BENE118316'), 'Yes')
   (('PHY385072', 'BENE41087'), 'Yes')
   (('PHY385072', 'BENE26003'), 'Yes')
   (('PHY344367', 'BENE155227'), 'Yes')
   physician PHY385072 even occur twice in top 5, so that physician may be highly making fraud claim
2. This show, if top attended physician and top patient making claim occur together, they are supposed to make fraud claim
3. so if famous physician and patient with many claim occur together, it is good to check the claim in more depth
4. from pair plot, it show that top attended physician tend to make fraud claim

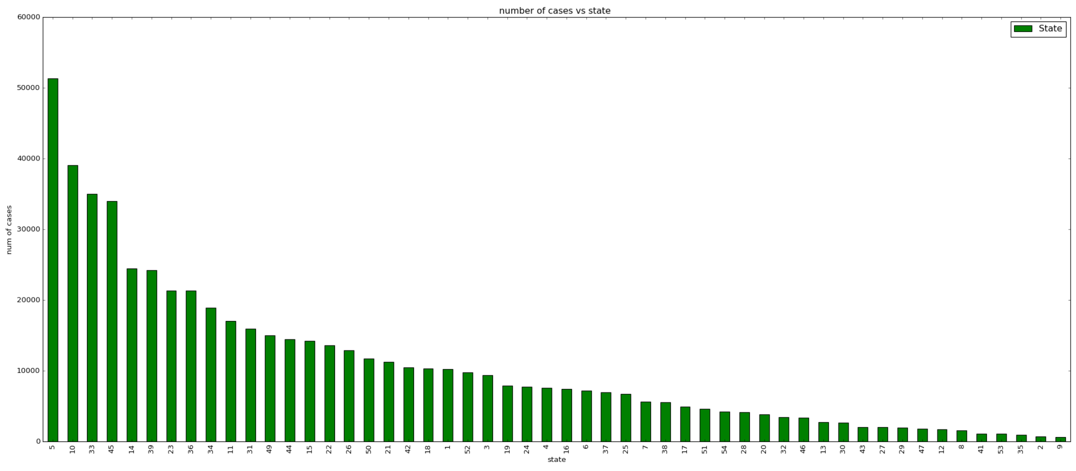
3.5. State feature

3.5.1.Univariate analysis - Analysing State feature with bar plot

In [26]:

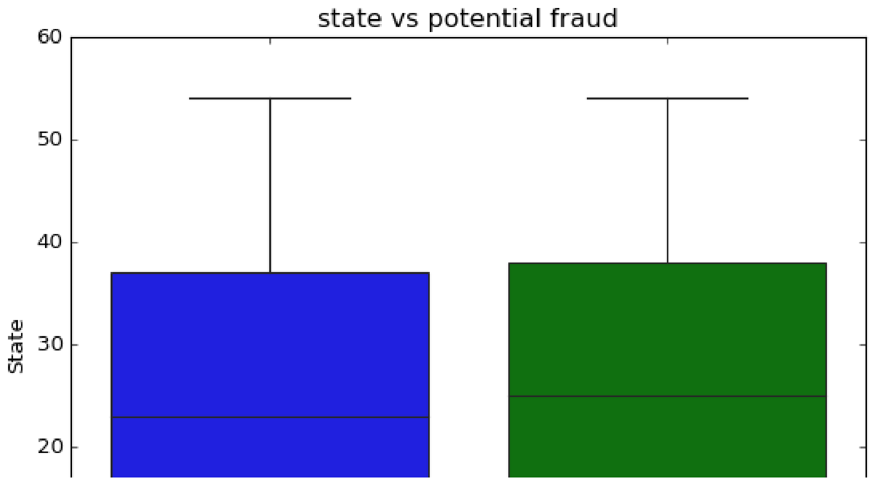
```
## num of cases in state

state_count = final_data['State'].value_counts()
print('num of different state :',len(state_count))
print(state_count[0:5])
state_count.plot(kind='bar', color ='green',figsize=(30,12),legend='reverse',title='number of
cases vs state ')
plt.xlabel('state')
plt.ylabel('num of cases')
plt.show()
```

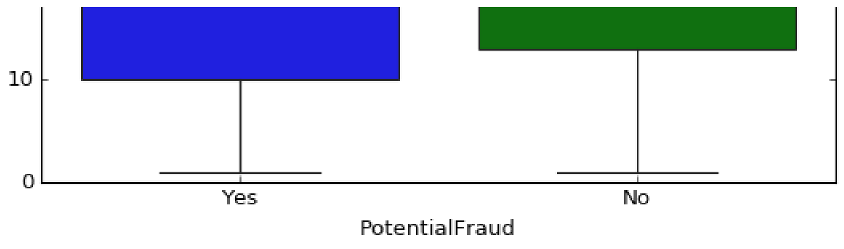
```
num of different state : 52
5      51350
10     39073
33     35024
45     34022
14     24417
Name: State, dtype: int64
```



In [27]:

```
## state vs fraud

sns.boxplot(x='PotentialFraud',y='State',data=final_data)
plt.title('state vs potential fraud')
plt.show()
```

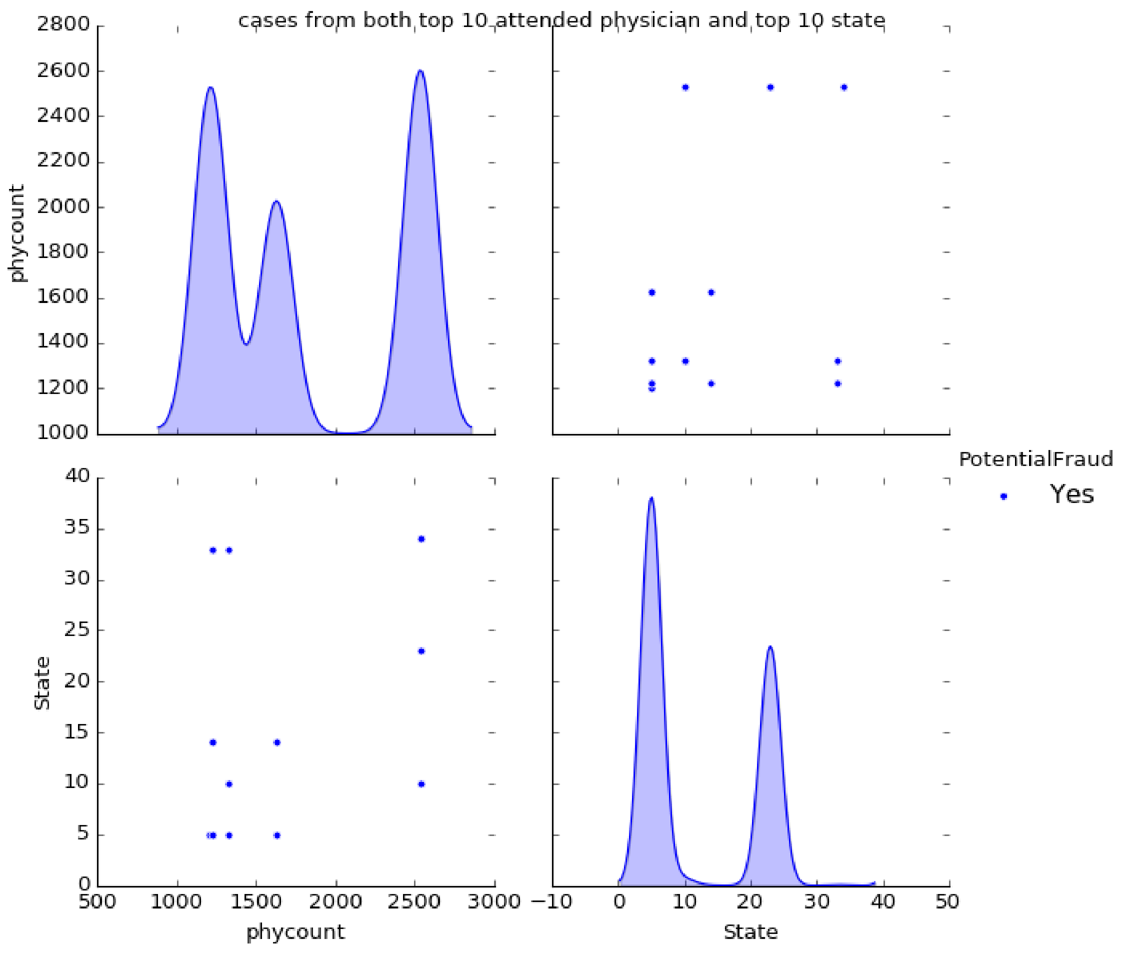




3.5.2. Bivariate analysis - Analysing AttendingPhysician and State feature with pair plot

In [28]:

```
## cases from top 10 attended physician and top 10 state

phy = ['PHY330576','PHY350277','PHY412132','PHY423534','PHY314027','PRV52019','PRV55462','PRV54367'
,'PRV53706','PRV55209']
state = [5,10,33,45,14,39,23,36,34,11]
phystate = final_data.loc[final_data['AttendingPhysician'].isin(phy) & final_data['State'].isin(sta
te)]



## pair plot on top 10 attended physician and top 10 state
sns.pairplot(phystate,hue='PotentialFraud',vars=['phycount','State'],height=3.8)
plt.suptitle('cases from both top 10 attended physician and top 10 state')
plt.show()
```

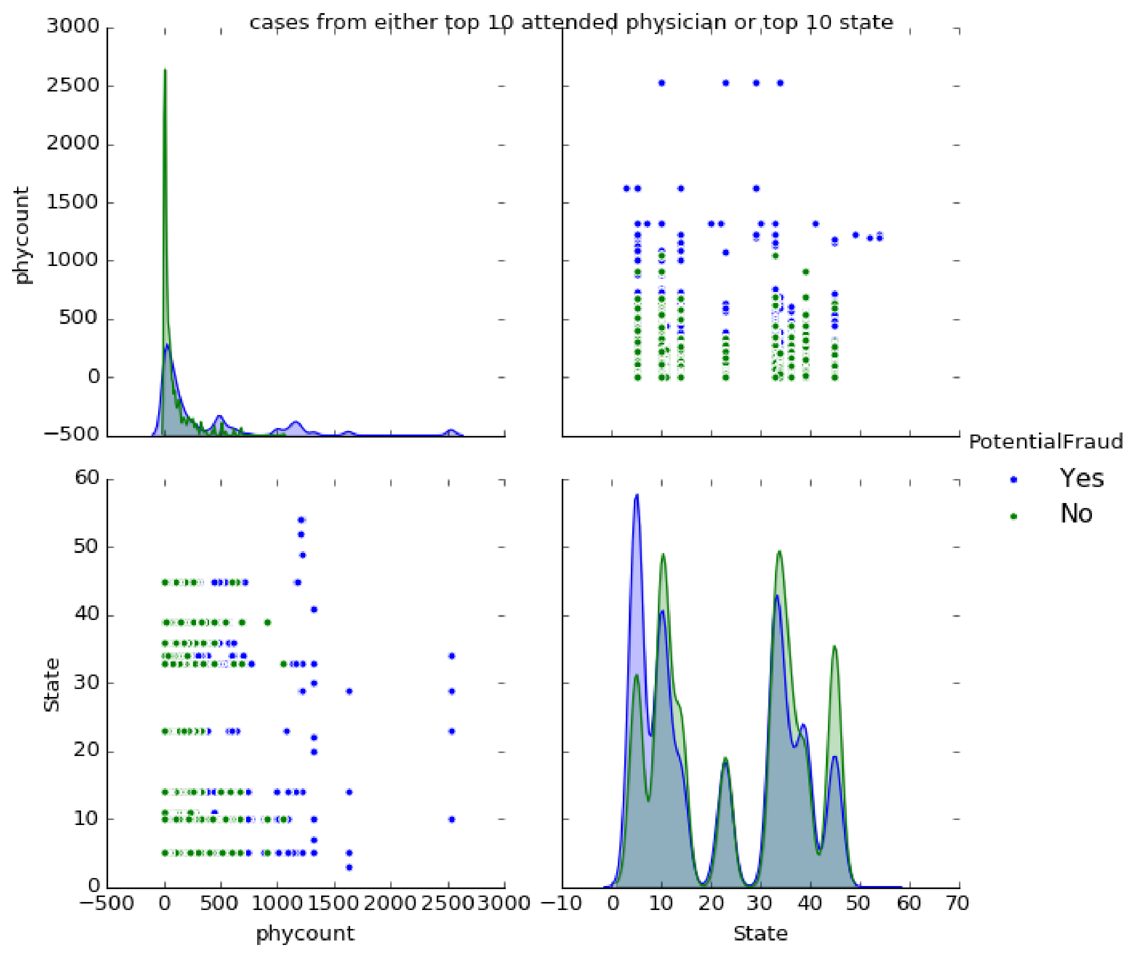


3.5.3. Bivariate analysis - Analysing AttendingPhysician or State feature with pair plot

In [29]:

```
## cases from either top 10 attended physician or top 10 state
```

```
phy = ['PHY330576','PHY350277','PHY412132','PHY423534','PHY314027','PRV52019','PRV55462','PRV54367'
,'PRV53706','PRV55209']
state = [5,10,33,45,14,39,23,36,34,11]
phyOrstate = final_data.loc[final_data['AttendingPhysician'].isin(phy) | final_data['State'].isin(s
tate)]

## pair plot on either top 10 attended physician or top 10 state
sns.pairplot(phyOrstate,hue='PotentialFraud',vars=['phycount','State'],height=3.8)
plt.suptitle('cases from either top 10 attended physician or top 10 state')
plt.show()
```



In [63]:

```
phystateclass = phystate['PotentialFraud'].value_counts().to_dict()
phyOrstateclass = phyOrstate['PotentialFraud'].value_counts().to_dict()

print('percentage of class including both top 10 attended physician and top 10 state  :\n',phystat
eclass)
print('percentage of class either from top 10 attended physician or top 10 state
:\n',phyOrstateclass)
```

```
percentage of class including both top 10 attended physician and top 10 state  :
 {'Yes': 6578}
percentage of class either from top 10 attended physician or top 10 state :
 {'Yes': 120186, 'No': 167821}
```

conclusion

1. number of unique state is 52
2. cases from state category 5.0 have most 52714 case
3. cases from state category 9.0 have least 617 case
4. from bar plot, only top 10 cases have cases around and more than 20000
5. from box plot, yes and no class is almost same distributed
6. from pair plot

3. from pair plot,

- cases attended including both top 10 attended physician and top 10 state (pair plot 1)

```
1. there is no non fraud claim made from top 10 physician and top state occuring together
2. this show when top physician and top state(containing more cases) occur together, they t
end to make fraud claim
```

- cases attended either from top 10 attended physician or top 10 state (pair plot 2)

```
1. this show when either top physician or state(containing more cases) occur, they tend to
make fraud claim equal to non fraud claim. In this around 40% claim are made fraud.
```

i.e., top physician and state(containing more cases) try to make fraud claim

3.6. DiagnosisCode

3.6.1. Univariate analysis - Analysing DiagnosisCode feature with bar plot

In [30]:

```
## adding all code and checking top code in diagnosis

final_data_diagnosis = pd.DataFrame(columns = ['DiagnosisCode'])
final_data_diagnosis['DiagnosisCode'] = pd.concat([final_data["ClmDiagnosisCode_1"],
                                           final_data["ClmDiagnosisCode_2"],
                                           final_data["ClmDiagnosisCode_3"],
                                           final_data["ClmDiagnosisCode_4"],
                                           final_data["ClmDiagnosisCode_5"],
                                           final_data["ClmDiagnosisCode_6"],
                                           final_data["ClmDiagnosisCode_7"],
                                           final_data["ClmDiagnosisCode_8"],
                                           final_data["ClmDiagnosisCode_9"],
                                           final_data["ClmDiagnosisCode_10"]], axis=0)
```

In [31]:

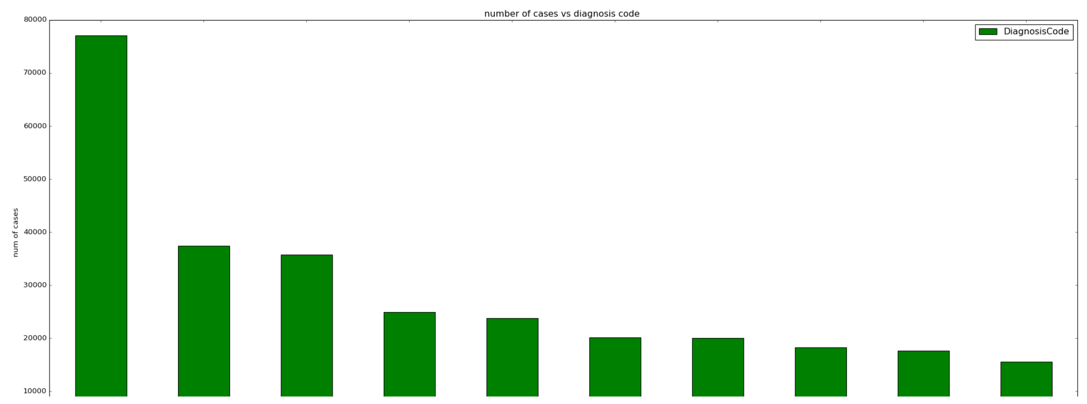
```
final_data_diagnosis = final_data_diagnosis.dropna()
final_data_diagnosis.shape
```

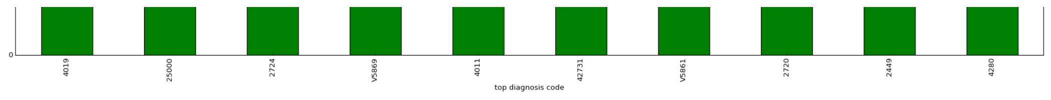
Out[31]:

```
(1680716, 1)
```

In [32]:

```
final_data_diagnosiscount = final_data_diagnosis['DiagnosisCode'].value_counts()[0:10]
final_data_diagnosiscount.plot(kind='bar', color ='green',figsize=(30,12),legend='reverse',title='
number of cases vs diagnosis code ')
plt.xlabel('top diagnosis code')
plt.ylabel('num of cases')
plt.show()
```





conclusion

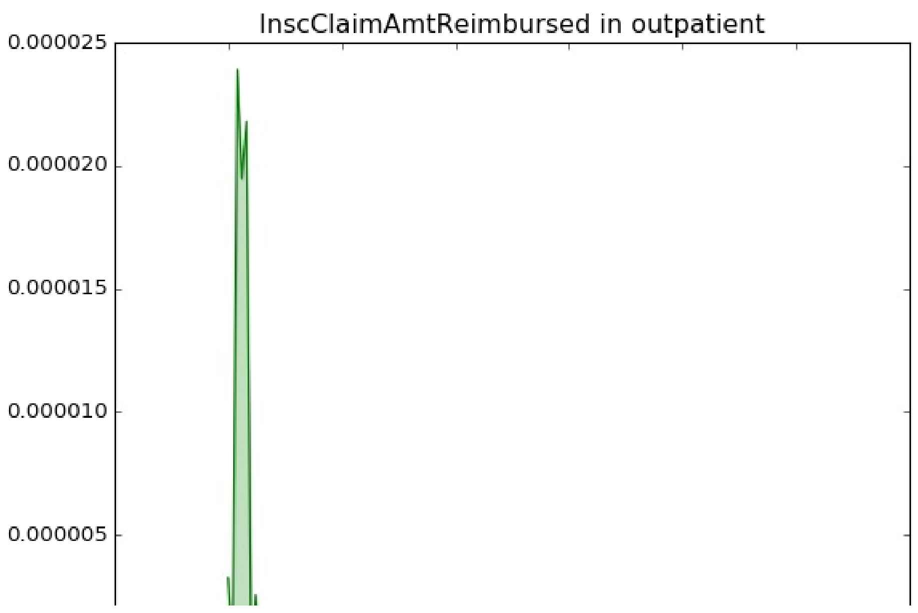
1. top 10 diagnosis code and their num of occur are
   '4019' -- 77056
   '25000' -- 37356
   '2724' -- 35763
   'V5869' -- 24904
   '4011' -- 23773
   '42731' -- 20138
   'V5861' -- 20001
   '2720' -- 18268
   '2449' -- 17600
   '4280' -- 15507
2. diagnosis code '4019' has 77056 highest num of occur
3. 4.4 % of patient undergone diagnosis code '4019'
4. these top 10 code are frequent so these code may be important in making fraud claim

3.7. InscClaimAmtReimbursed feature

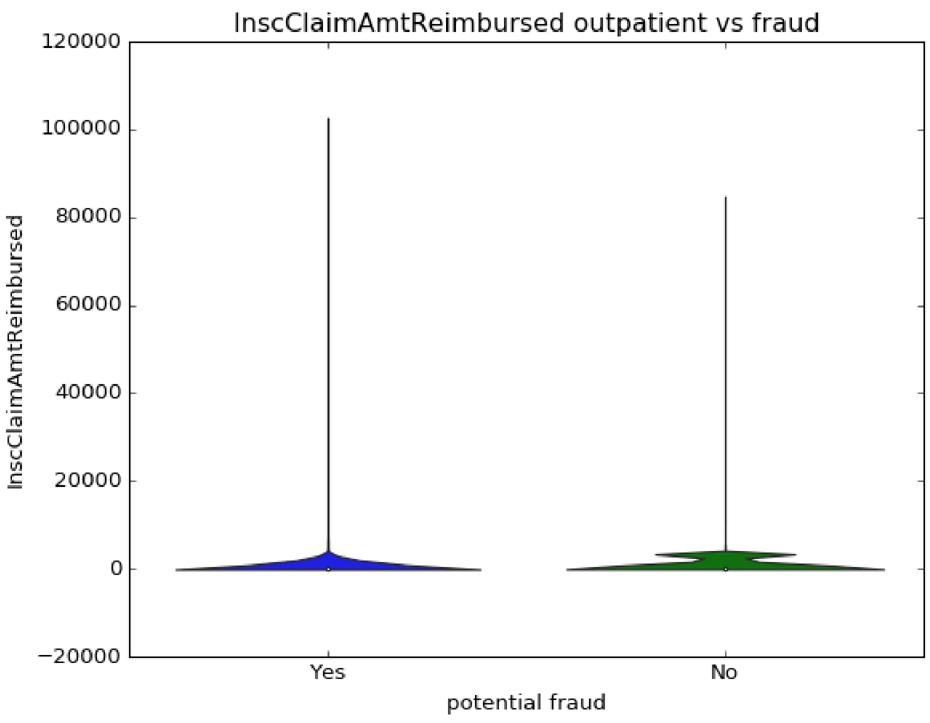
3.7.1. Univariate analysis - Analysing InscClaimAmtReimbursed feature with kde plot for outpatient

In [33]:

```
## using kde plot which use probability density function to analyse amount claimed

sns.kdeplot(train_outpatient['InscClaimAmtReimbursed'],shade=True,color='green',legend=False)
plt.title('InscClaimAmtReimbursed in outpatient')
plt.xlabel('InscClaimAmtReimbursed')
plt.show()


## InscClaimAmtReimbursed outpatient vs fraud

sns.violinplot(x='PotentialFraud',y='InscClaimAmtReimbursed', data=train_outpatient)
plt.title('InscClaimAmtReimbursed outpatient vs fraud')
plt.xlabel('potential fraud')
plt.ylabel('InscClaimAmtReimbursed')
plt.show()
```



0.000000
−20000
0
20000
40000
60000
80000
100000
120000
InscClaimAmtReimbursed



conclusion

1. this follows log normal distribution
2. this shows only few claim amount is greater than 20000
3. most of claims have amount ranging within 20000
4. few claim even reach 60000 showing that may be fraud claim

3.7.2. Univariate analysis - Analysing InscClaimAmtReimbursed feature with kde plot for inpatient

In [63]:

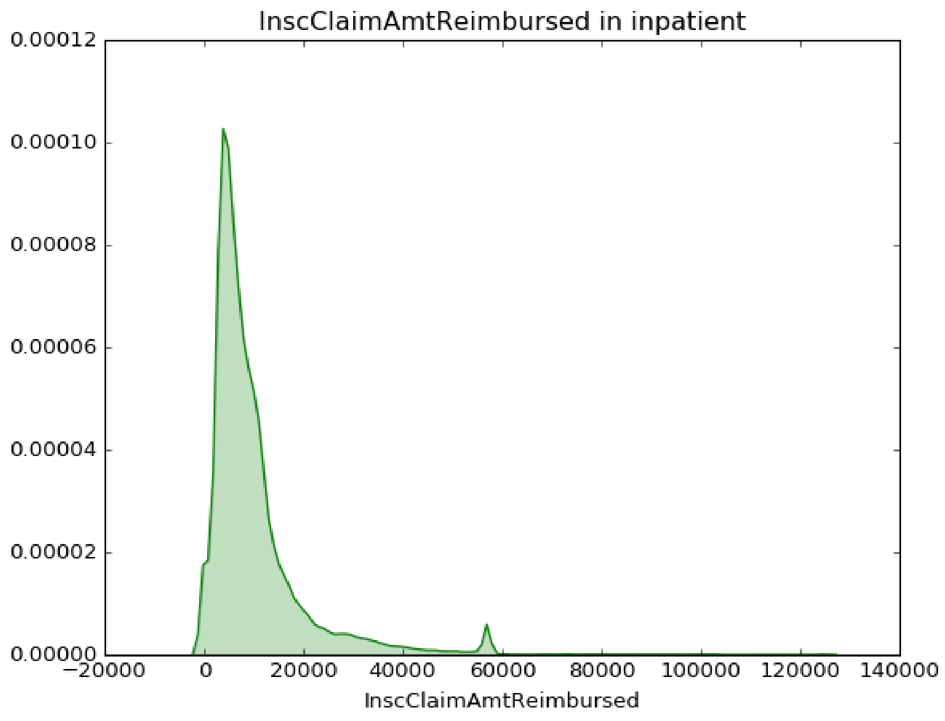
```
## percentile to check amount below 99 and 100 percentile

q = [0.97,0.98,0.99,1]
for i in q:
    print('percentile',i,'is',final_data.InscClaimAmtReimbursed_y.dropna().quantile(i))
```

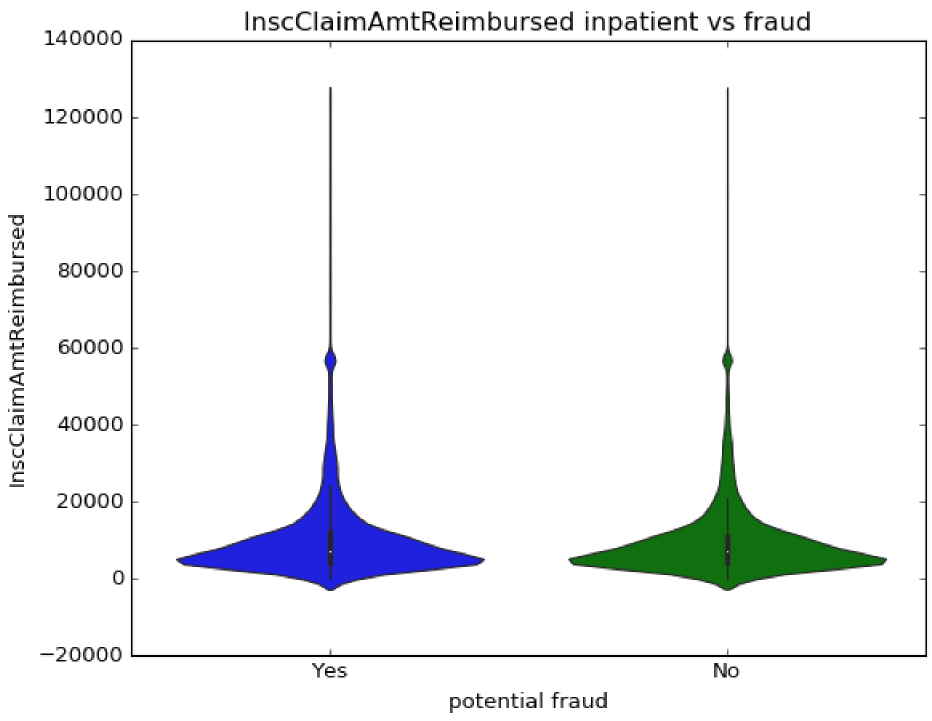
```
percentile 0.97 is 2200.0
percentile 0.98 is 2600.0
percentile 0.99 is 3300.0
percentile 1 is 102500.0
```

In [34]:

```
## using kde plot which use probability density function to analyse amount claimed

sns.kdeplot(train_inpatient['InscClaimAmtReimbursed'],shade=True,color='green',legend=False)
plt.title('InscClaimAmtReimbursed in inpatient')
plt.xlabel('InscClaimAmtReimbursed')
plt.show()


## InscClaimAmtReimbursed inpatient vs fraud
```

```
sns.violinplot(x='PotentialFraud',y='InscClaimAmtReimbursed', data=train_inpatient)
plt.title('InscClaimAmtReimbursed inpatient vs fraud')
plt.xlabel('potential fraud')
plt.ylabel('InscClaimAmtReimbursed')
plt.show()
```





conclusion

1. this follows log normal distribution
2. this shows only few claim amount is around 100000
3. most of claims have amount ranging within 3000
4. there is a huge difference between 99 and 100 percentile, showing it may be fraud claim
5. few claim even reach 100000 showing that may be fraud claim

3.8. Race feature

3.8.1. Univariate analysis - Analysing Race feature with bar plot

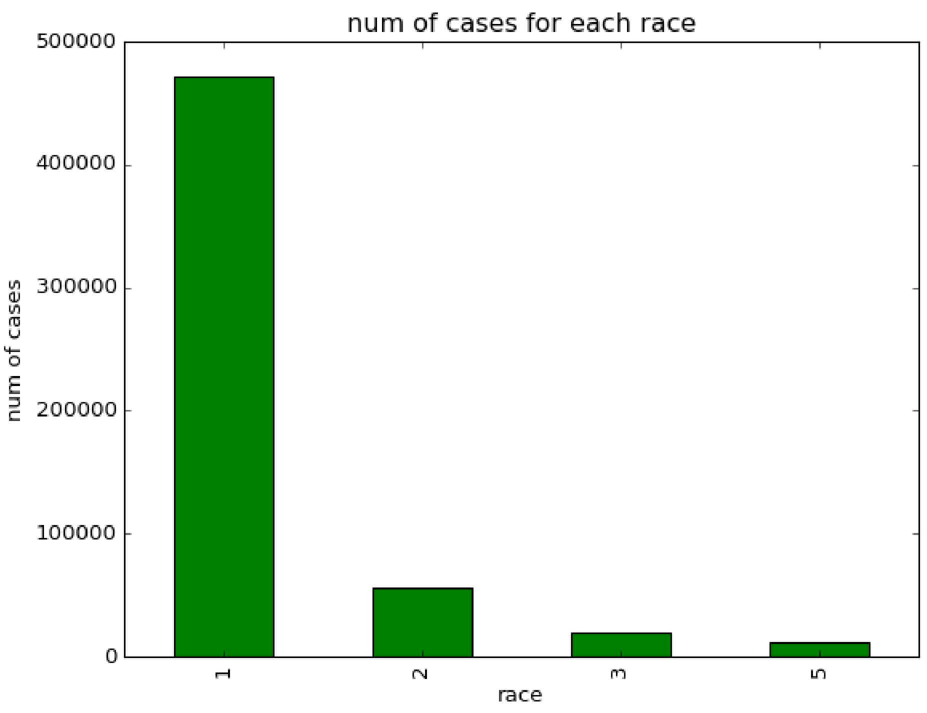
In [36]:

```
## taking different race count

race_count = final_data['Race'].value_counts()
print(race_count)
```

```
1    471036
2     55640
3     19715
5     11820
Name: Race, dtype: int64
```

In [37]:

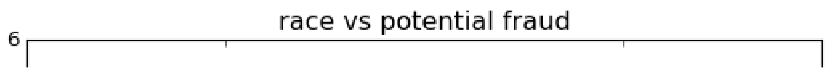
```
## num of cases for each race

race_count.plot(kind='bar', color ='green',title='num of cases for each race')
plt.xlabel('race')
plt.ylabel('num of cases')
plt.show()
```

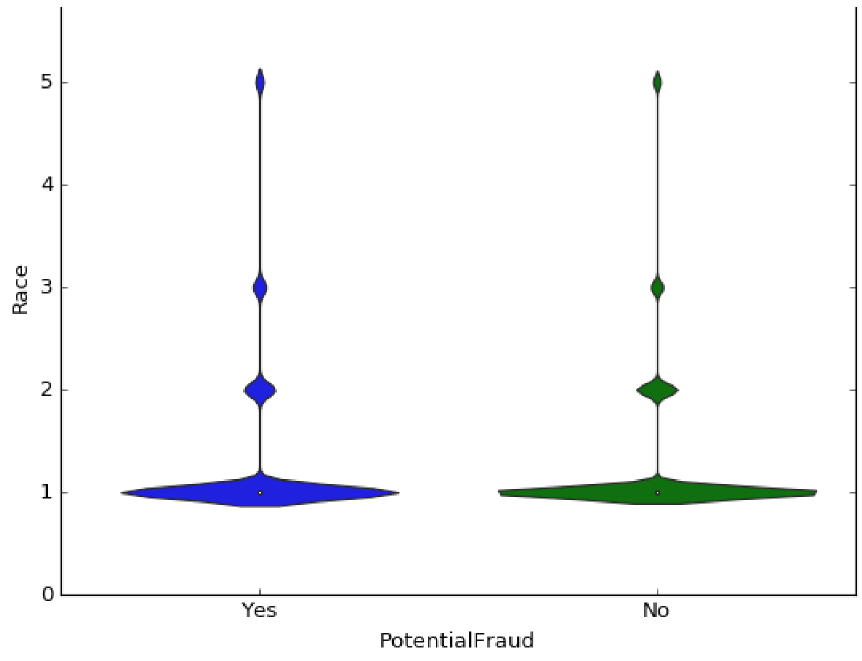


3.8.2. checking race feature with potential fraud using violin plot

In [38]:

```
sns.violinplot(x='PotentialFraud',y='Race',data=final_data)
plt.title('race vs potential fraud')
plt.show()
```





conclusion

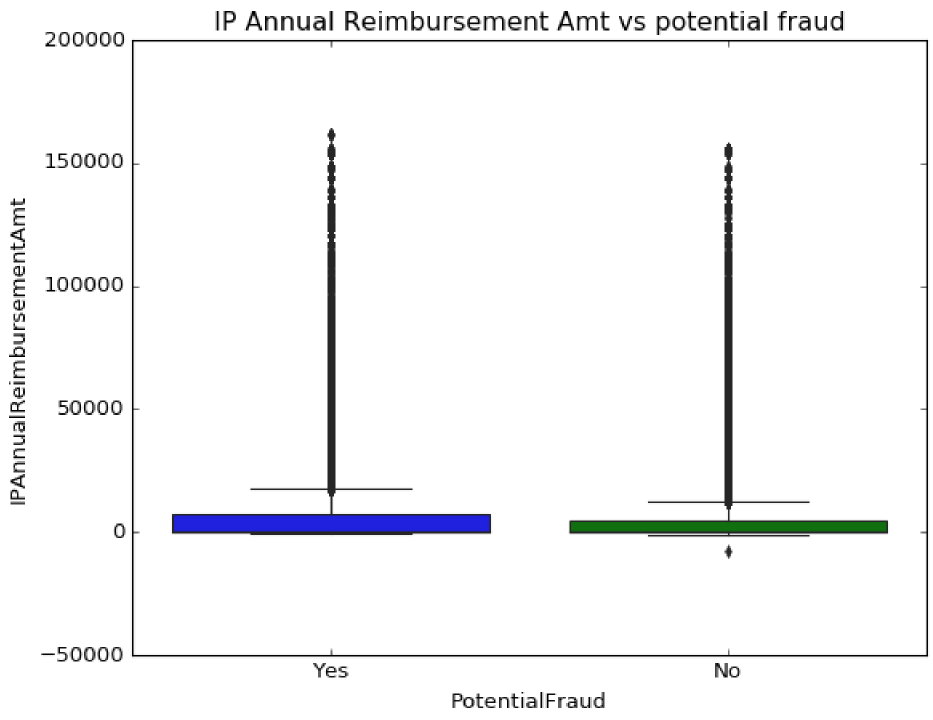
1. bar plot shows people belonging to race with category 1 occur more than other
2. further it is checked with violin plot on potential fraud
3. this violin plot give that race 1 has made more fraud claim than other
4. other category only very few and vary with same distribution on fraud or non fraud

3.9. IPAnnualReimbursementAmt, OPAnnualReimbursementAmt feature

3.9.1. Univariate analysis - Analysing IPAnnualReimbursementAmt feature with box plot

In [39]:

```
sns.boxplot(x='PotentialFraud',y='IPAnnualReimbursementAmt', data=final_data)
plt.title('IP Annual Reimbursement Amt vs potential fraud')
plt.show()
```



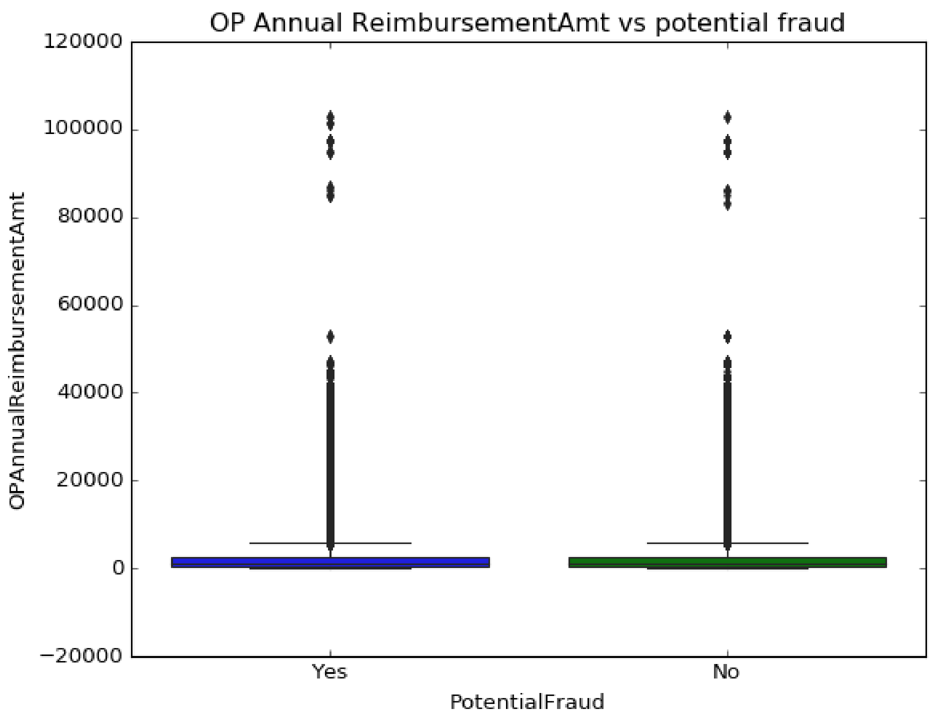
conclusion

1. number of fraud claim have more range than non fraud claim
2. but fraud and non fraud cases overlap fully which do not help to differntiate them
3. but it shows that highest amound claimed was fraud claim

3.9.2. Univariate analysis - Analysing OPAnnualReimbursementAmt feature with box plot

In [40]:

```
sns.boxplot(x='PotentialFraud',y='OPAnnualReimbursementAmt', data=final_data)
plt.title('OP Annual ReimbursementAmt vs potential fraud')
plt.show()
```



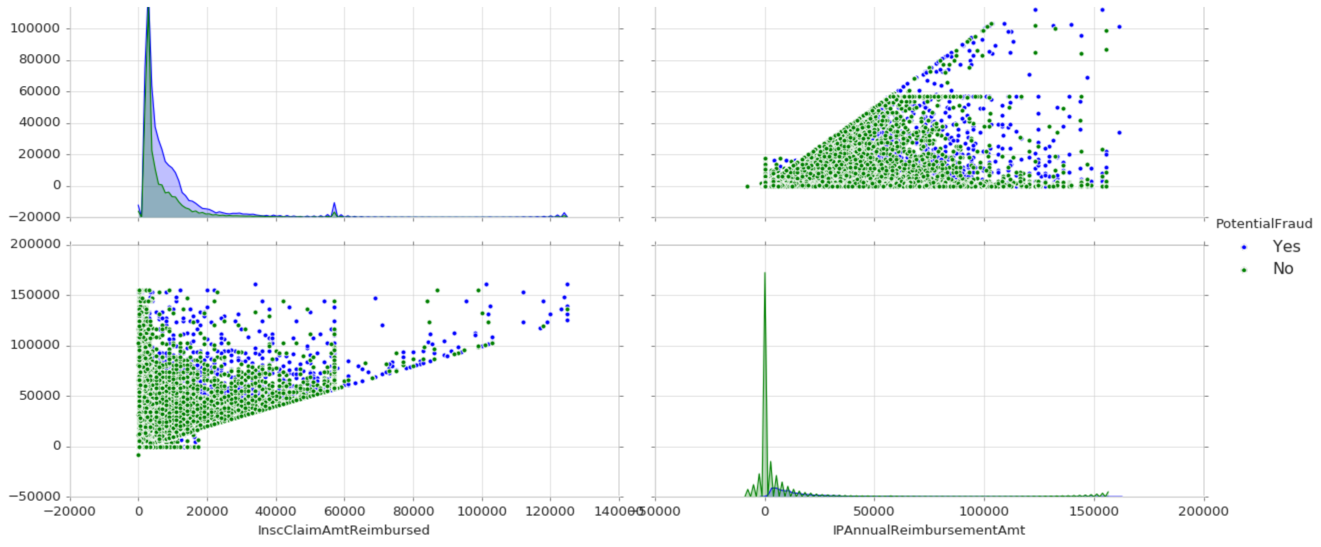
conclusion

1. number of fraud claim have same range as non fraud claim
2. but fraud and non fraud cases overlap fully which do not help to differntiate them
3. but it shows that highest amound claimed was fraud claim
4. this analysis on IP and OP amount show fraud claims are paid highest than non fraud, which show that if amount for claim is high more analysis should be done on that claim to check if it is fraud or not

3.10. Bivariate analysis - pair plot on InscClaimAmtReimbursed and IPAnnualReimbursementAmt

In [41]:

```
sns.set_style("whitegrid")
sns.pairplot(final_data,hue='PotentialFraud',vars=['InscClaimAmtReimbursed','IPAnnualReimbursementA
t'],height=3.8,aspect=2)
plt.suptitle('InscClaimAmtReimbursed and IPAnnualReimbursementAmt')
plt.show()
```



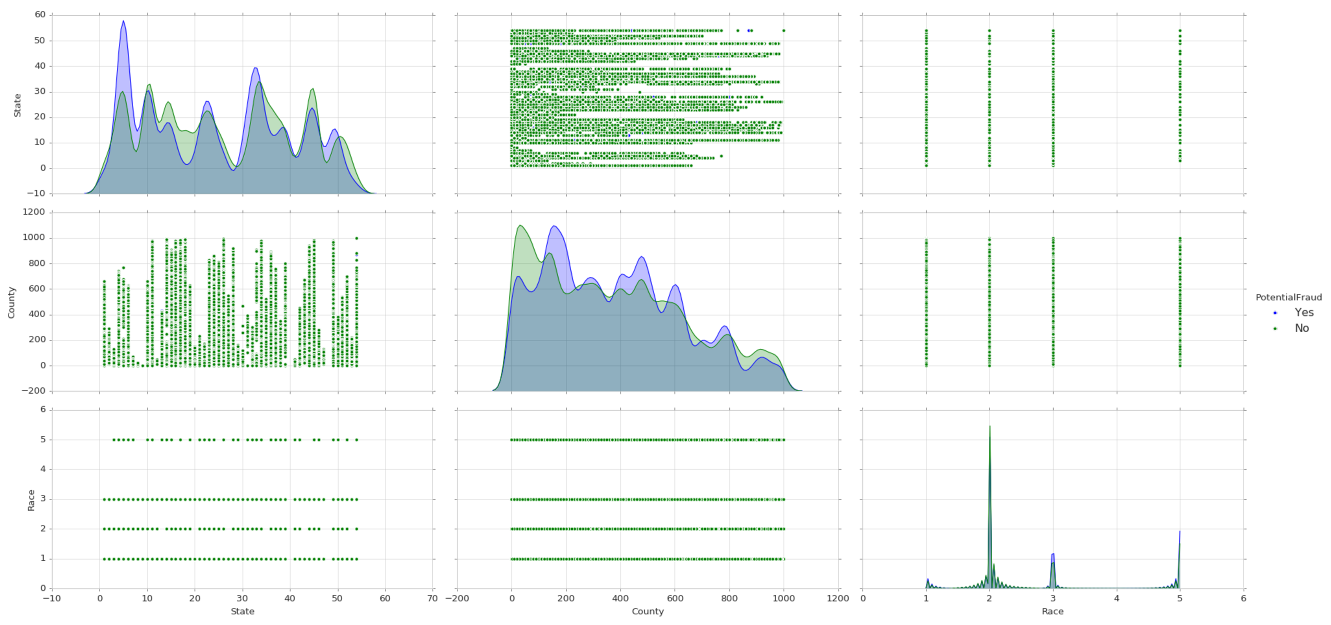
conclusion

1. InscClaimAmtReimbursed and IPAnnualReimbursementAmt overlap more, but show that some fraud claim are made with higher amount tha non fraud claim
2. this show these two feature can help in classifying fraud claim that are made with higher amount

3.11. Bivariate analysis - pair plot on State,County,Race

In [43]:

```
sns.set_style("whitegrid")
sns.pairplot(final_data,hue='PotentialFraud',vars=['State','County','Race'],height=3.8,aspect=2)
plt.show()
```



conclusion

1. state feature tend to show some fraud claim in some state, but when combined with country and race it do not differ any fraud and non fraud claim
2. fraud and non fraud claim overlap in all case

3.12. AdmissionDt and DischargeDt in inpatient

In [44]:

```
## converting non-null object type col to datetime64
train_inpatient['AdmissionDt']= pd.to_datetime(train_inpatient['AdmissionDt'])
train_inpatient['DischargeDt']= pd.to_datetime(train_inpatient['DischargeDt'])
train_inpatient.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43792 entries, 0 to 43791
Data columns (total 34 columns):
BeneID                     40474 non-null object
ClaimID                    40474 non-null object
ClaimStartDt               40474 non-null object
ClaimEndDt                 40474 non-null object
Provider                   43792 non-null object
InscClaimAmtReimbursed     40474 non-null float64
AttendingPhysician         40362 non-null object
OperatingPhysician         23830 non-null object
OtherPhysician             4690 non-null object
AdmissionDt                40474 non-null datetime64[ns]
ClmAdmitDiagnosisCode      40474 non-null object
DeductibleAmtPaid          39575 non-null float64
DischargeDt                40474 non-null datetime64[ns]
DiagnosisGroupCode         40474 non-null object
ClmDiagnosisCode_1         40474 non-null object
ClmDiagnosisCode_2         40248 non-null object
ClmDiagnosisCode_3         39798 non-null object
ClmDiagnosisCode_4         38940 non-null object
ClmDiagnosisCode_5         37580 non-null object
ClmDiagnosisCode_6         35636 non-null object
ClmDiagnosisCode_7         33216 non-null object
ClmDiagnosisCode_8         30532 non-null object
ClmDiagnosisCode_9         26977 non-null object
ClmDiagnosisCode_10        3927 non-null object
ClmProcedureCode_1         23148 non-null float64
ClmProcedureCode_2         5454 non-null float64
ClmProcedureCode_3         965 non-null float64
ClmProcedureCode_4         116 non-null float64
ClmProcedureCode_5         9 non-null float64
ClmProcedureCode_6         0 non-null float64
PotentialFraud             43792 non-null object
AttendingPhysiciancount    40362 non-null float64
Providercount              43792 non-null int64
BeneIDcount                40474 non-null float64
dtypes: datetime64[ns](2), float64(10), int64(1), object(21)
memory usage: 11.7+ MB
```

number of days admitted

In [45]:

```
train_inpatient['numOfDaysAdmitted'] = train_inpatient['DischargeDt'] - train_inpatient['AdmissionD
t']
```

In [46]:

```
train_inpatient['numOfDaysAdmitted'] = train_inpatient['numOfDaysAdmitted'].fillna(0)
```

In [47]:

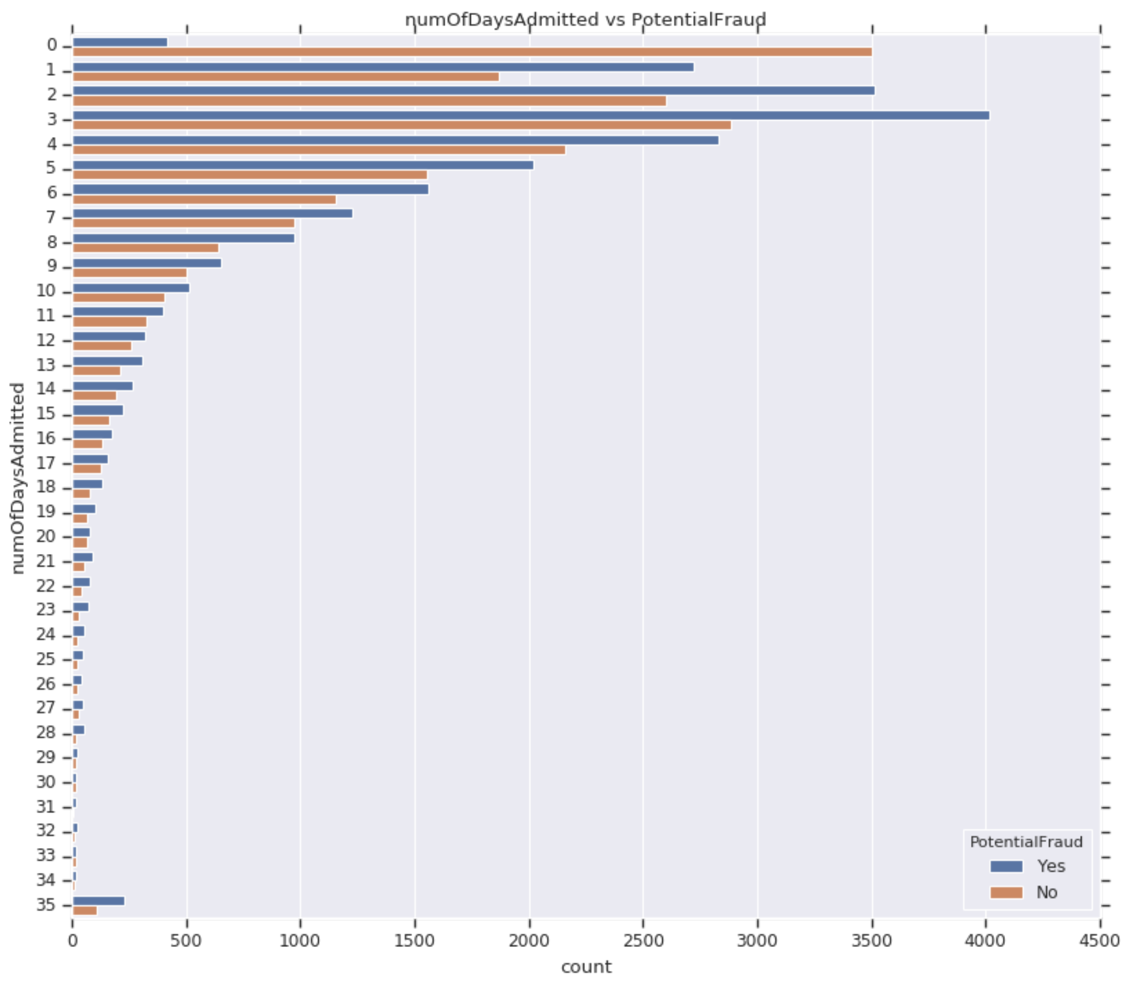
```
## converting days to int
train_inpatient['numOfDaysAdmitted'] = train_inpatient['numOfDaysAdmitted'].dt.days.astype('int64')
```

3.12.1. numOfDaysAdmitted vs PotentialFraud using countplot

In [48]:

```
plt.close()
sns.set(rc={'figure.figsize':(12,10)})
ax = sns.countplot(y='numOfDaysAdmitted',data=train_inpatient,hue='PotentialFraud')
```

```
ax.set_title('numOfDaysAdmitted vs PotentialFraud')
plt.show()
```

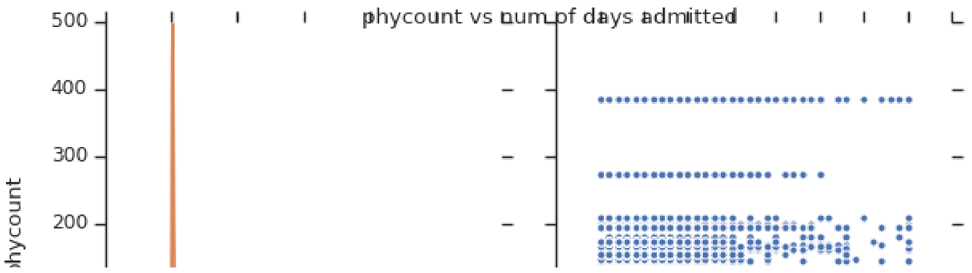


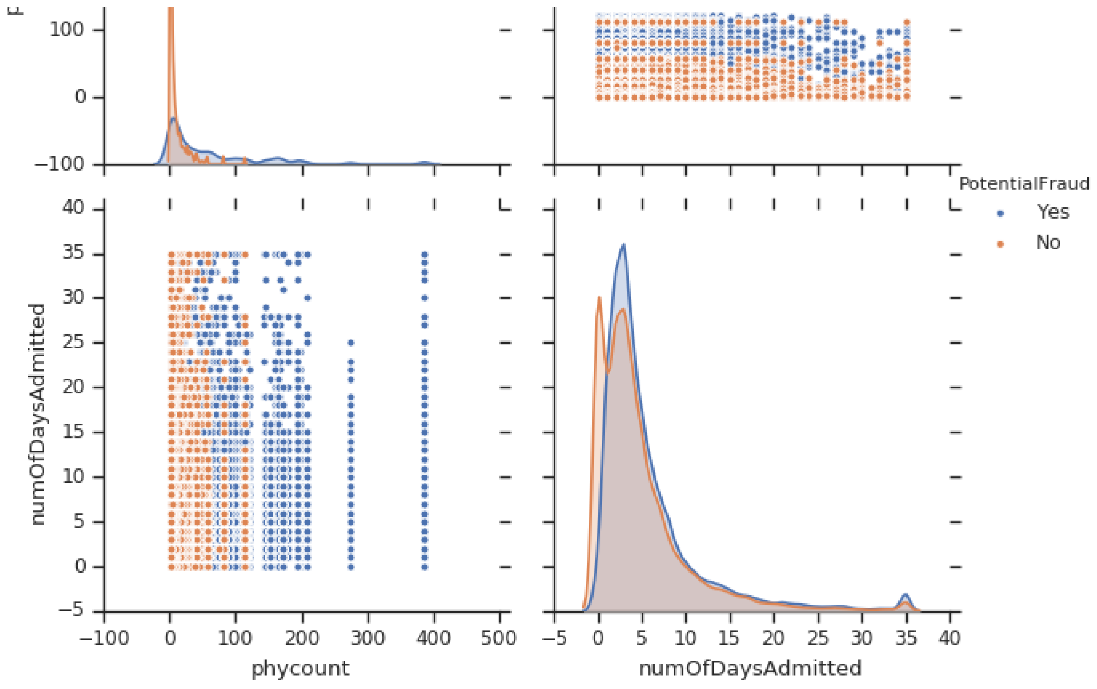
3.12.2. Bivariate analysis - numOfDaysAdmitted and AttendingPhysician using pair plot

In [49]:

```
## num of cases attended by phy
attendingPhysician_count = train_inpatient['AttendingPhysician'].value_counts().to_dict()
print('num of unique physician are: ',len(attendingPhysician_count))

train_inpatient['phycount']=train_inpatient['AttendingPhysician'].map(attendingPhysician_count)

sns.set_style("white")
sns.pairplot(train_inpatient,hue='PotentialFraud',vars=['phycount','numOfDaysAdmitted'],height=3.8)
plt.suptitle('phycount vs num of days admitted')
plt.show()
```

```
num of unique physician are:  11604
```





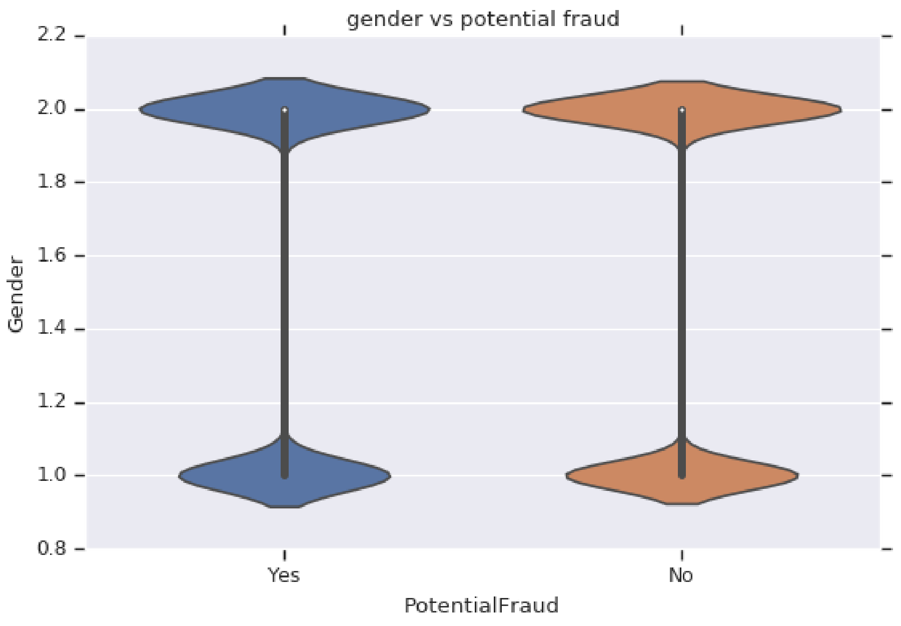
conclusion

1. num of day patient admitted can be got from, discharge date - admitted
2. from counterplot, patient who are admitted between 2 to 7 tend to show more fraud claim
3. from pair plot, top attended physician tend to make more fraud claim in all case of number of day admitted
4. from all analysis it is shown that top attended physician tend to make fraud claim

3.13. Gender feature

In [50]:

```
sns.set(rc={'figure.figsize':(8,5)})
sns.violinplot(x='PotentialFraud',y='Gender',data=final_data)
plt.title('gender vs potential fraud')
plt.show()
```



conclusion

1. gender 2 occur more num of times than 1
2. num of fraud occur more time in gender 2 than gender 1

summary

1. class distribution -

```
    percentage of fraud class :  36.5615138912,
    percentage of non fraud class :  63.4384861088
```

2. If a physician has attended many number of cases previously, there is more chance of making fraud claim next time
3. If a provider has high num of cases covered by a provider previously, there is more chance to do fraud claim
4. If top attended physician and top covered provider occur together they tend to make fraud claim
5. If top attended physician and top patient who made more claim occur together, they are supposed to make fraud claim. outpatient made 1 - 9 claim previously tend to make fraud claim
6. few state show more fraud claim
7. If top attended physician and top ten state(containing more cases) occur either together or separate, it more possibility of fraud claim
8. top occuring diagnosis code that most patient undergone are '4019','25000','2724','V5869','4011','42731','V5861','2720','2449','4280'
9. range of fraud claim made by outpatient is around 5000 and for inpatient 20000
10. race 1 occur more number of time, and has most fraud claim
11. reimbursed amount distribution in fraud and non fraud claim almost overlap
12. state, country, race has distribution almost overlap
13. If patient is admitted 2 to 7 days previously, they tend to make fraud claim next time

Main feature that help in classify fraud and non fraud

1. provider count (has high num of cases covered) is main feature
2. physician count (has attended many number of cases) is next main feature

4. Data preprocessing

In [48]:

```
final_data.columns
```

Out[48]:

```
Index(['BeneID', 'ClaimID', 'ClaimStartDt', 'ClaimEndDt', 'Provider',
       'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
       'OtherPhysician', 'AdmissionDt', 'ClmAdmitDiagnosisCode',
       'DeductibleAmtPaid', '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', 'Gender', 'Race',
       'RenalDiseaseIndicator', 'State', 'County', 'NoOfMonths_PartACov',
       'NoOfMonths_PartBCov', 'ChronicCond_Alzheimer',
       'ChronicCond_Heartfailure', 'ChronicCond_KidneyDisease',
       'ChronicCond_Cancer', 'ChronicCond_ObstrPulmonary',
       'ChronicCond_Depression', 'ChronicCond_Diabetes',
       'ChronicCond_IschemicHeart', 'ChronicCond_Osteoporasis',
       'ChronicCond_rheumatoidarthritis', 'ChronicCond_stroke',
       'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
       'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt', 'PotentialFraud',
       'phycount', 'procount', 'benecount'],
      dtype='object')
```

In [163]:

```
final_data.shape
```

Out[163]:

```
(558211, 55)
```

4.1. checking null value

In [304]:

```
## checking null value

final_data.isnull().sum()
```

Out[304]:

```
BeneID                                 0
ClaimID                                0
ClaimStartDt                           0
ClaimEndDt                             0
Provider                               0
InscClaimAmtReimbursed                 0
AttendingPhysician                  1508
OperatingPhysician                443764
OtherPhysician                    358475
AdmissionDt                       517737
ClmAdmitDiagnosisCode             412312
DeductibleAmtPaid                    899
DischargeDt                       517737
DiagnosisGroupCode                517737
ClmDiagnosisCode_1                 10453
ClmDiagnosisCode_2                195606
ClmDiagnosisCode_3                315156
ClmDiagnosisCode_4                393675
ClmDiagnosisCode_5                446287
ClmDiagnosisCode_6                473819
ClmDiagnosisCode_7                492034
ClmDiagnosisCode_8                504767
ClmDiagnosisCode_9                516396
ClmDiagnosisCode_10               553201
ClmProcedureCode_1                534901
ClmProcedureCode_2                552721
ClmProcedureCode_3                557242
ClmProcedureCode_4                558093
ClmProcedureCode_5                558202
ClmProcedureCode_6                558211
DOB                                    0
DOD                               554080
Gender                                 0
Race                                   0
RenalDiseaseIndicator                  0
State                                  0
County                                 0
NoOfMonths_PartACov                    0
NoOfMonths_PartBCov                    0
ChronicCond_Alzheimer                  0
ChronicCond_Heartfailure               0
ChronicCond_KidneyDisease              0
ChronicCond_Cancer                     0
ChronicCond_ObstrPulmonary             0
ChronicCond_Depression                 0
ChronicCond_Diabetes                   0
ChronicCond_IschemicHeart              0
ChronicCond_Osteoporasis               0
ChronicCond_rheumatoidarthritis        0
ChronicCond_stroke                     0
IPAnnualReimbursementAmt               0
IPAnnualDeductibleAmt                  0
OPAnnualReimbursementAmt               0
OPAnnualDeductibleAmt                  0
PotentialFraud                         0
dtype: int64
```

In [305]:

```
print('num of null value in ClmProcedureCode_5 :',final_data['ClmProcedureCode_5'].isnull().sum())
print('num of null value in ClmProcedureCode_6 :',final_data['ClmProcedureCode_6'].isnull().sum())
```

```
num of null value in ClmProcedureCode_5 : 558202
num of null value in ClmProcedureCode_6 : 558211
```

In [314]:

```
## removing ClmProcedureCode_5, ClmProcedureCode_6 as it has all null

final_data.drop('ClmProcedureCode_5',axis=1,inplace=True)
final_data.drop('ClmProcedureCode_6',axis=1,inplace=True)

## remove as it has all value as same

final_data.drop('NoOfMonths_PartACov',axis=1,inplace=True)
final_data.drop('NoOfMonths_PartBCov',axis=1,inplace=True)
```

In [87]:

```
## replacing null with 0

colFillna = ['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',
             'DiagnosisGroupCode','ClmAdmitDiagnosisCode']

final_data[colFillna]= final_data[colFillna].replace({np.nan:0})
```

In [315]:

```
## replacing null with mode

colFillmode = ['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',
             'DiagnosisGroupCode','ClmAdmitDiagnosisCode']

for i in colFillna:
    mode = final_data[i].mode()[0]
    print(i,'mode :',mode)
    final_data[i]=final_data[i].fillna(mode)
```

```
ClmDiagnosisCode_1 mode : 4019
ClmDiagnosisCode_2 mode : 4019
ClmDiagnosisCode_3 mode : 4019
ClmDiagnosisCode_4 mode : 4019
ClmDiagnosisCode_5 mode : 4019
ClmDiagnosisCode_6 mode : 4019
ClmDiagnosisCode_7 mode : 4019
ClmDiagnosisCode_8 mode : 4019
ClmDiagnosisCode_9 mode : 4019
ClmDiagnosisCode_10 mode : 4019
ClmProcedureCode_1 mode : 9904.0
ClmProcedureCode_2 mode : 4019.0
ClmProcedureCode_3 mode : 4019.0
ClmProcedureCode_4 mode : 4019.0
DiagnosisGroupCode mode : 882
ClmAdmitDiagnosisCode mode : V7612
```

In [238]:

```
final_data.isnull().sum()
```

Out[238]:

```
BeneID                                        0
```

```
ClaimID                                0
ClaimStartDt                           0
ClaimEndDt                             0
Provider                               0
InscClaimAmtReimbursed                 0
AttendingPhysician                  1508
OperatingPhysician                443764
OtherPhysician                    358475
AdmissionDt                       517737
ClmAdmitDiagnosisCode                  0
DeductibleAmtPaid                    899
DischargeDt                       517737
DiagnosisGroupCode                     0
ClmDiagnosisCode_1                     0
ClmDiagnosisCode_2                     0
ClmDiagnosisCode_3                     0
ClmDiagnosisCode_4                     0
ClmDiagnosisCode_5                     0
ClmDiagnosisCode_6                     0
ClmDiagnosisCode_7                     0
ClmDiagnosisCode_8                     0
ClmDiagnosisCode_9                     0
ClmDiagnosisCode_10                    0
ClmProcedureCode_1                     0
ClmProcedureCode_2                     0
ClmProcedureCode_3                     0
ClmProcedureCode_4                     0
DOB                                    0
DOD                               554080
Gender                                 0
Race                                   0
RenalDiseaseIndicator                  0
State                                  0
County                                 0
ChronicCond_Alzheimer                  0
ChronicCond_Heartfailure               0
ChronicCond_KidneyDisease              0
ChronicCond_Cancer                     0
ChronicCond_ObstrPulmonary             0
ChronicCond_Depression                 0
ChronicCond_Diabetes                   0
ChronicCond_IschemicHeart              0
ChronicCond_Osteoporasis               0
ChronicCond_rheumatoidarthritis        0
ChronicCond_stroke                     0
IPAnnualReimbursementAmt               0
IPAnnualDeductibleAmt                  0
OPAnnualReimbursementAmt               0
OPAnnualDeductibleAmt                  0
PotentialFraud                         0
dtype: int64
```

4.2. checking for duplicate row

In [309]:

```
print('num of claim :',final_data.shape[0])
print('num of unique claim :',len(final_data['ClaimID'].value_counts()))
```

```
num of claim : 558211
num of unique claim : 558211
```

4.3. PotentialFraud has yes and no - replacing with 1 and 0

In [239]:

```
final_data['PotentialFraud'] = final_data['PotentialFraud'].map({'Yes':1,'No':0})
```

In [240]:

```
final_data['PotentialFraud'].value_counts()
```

Out[240]:

```
0    345415
1    212796
Name: PotentialFraud, dtype: int64
```

4.4. RenalDiseaseIndicator has y and 0 - replacing with 1 and 0

In [241]:

```
final_data['RenalDiseaseIndicator'] = final_data['RenalDiseaseIndicator'].map({'Y':1,'0':0})
```

In [242]:

```
final_data['RenalDiseaseIndicator'].value_counts()
```

Out[242]:

```
0    448363
1    109848
Name: RenalDiseaseIndicator, dtype: int64
```

4.5. ChronicCond has 2 and 1 - replacing with 0 and 1

In [243]:

```
final_data['ChronicCond_Alzheimer'] = final_data['ChronicCond_Alzheimer'].map({2:0,1:1})
final_data['ChronicCond_Heartfailure'] = final_data['ChronicCond_Heartfailure'].map({2:0,1:1})
final_data['ChronicCond_KidneyDisease'] = final_data['ChronicCond_KidneyDisease'].map({2:0,1:1})
final_data['ChronicCond_Cancer'] = final_data['ChronicCond_Cancer'].map({2:0,1:1})
final_data['ChronicCond_ObstrPulmonary'] = final_data['ChronicCond_ObstrPulmonary'].map({2:0,1:1})
final_data['ChronicCond_Depression'] = final_data['ChronicCond_Depression'].map({2:0,1:1})
final_data['ChronicCond_Diabetes'] = final_data['ChronicCond_Diabetes'].map({2:0,1:1})
final_data['ChronicCond_IschemicHeart'] = final_data['ChronicCond_IschemicHeart'].map({2:0,1:1})
final_data['ChronicCond_Osteoporasis'] = final_data['ChronicCond_Osteoporasis'].map({2:0,1:1})
final_data['ChronicCond_rheumatoidarthritis'] = final_data['ChronicCond_rheumatoidarthritis'].map(
{2:0,1:1})
final_data['ChronicCond_stroke'] = final_data['ChronicCond_stroke'].map({2:0,1:1})
final_data['Gender'] = final_data['Gender'].map({2:0,1:1})
```

summary

1. no duplicate rows
2. PotentialFraud has yes and no - replacing with 1 and 0
3. RenalDiseaseIndicator has y and 0 - replacing with 1 and 0
4. ChronicCond has 2 and 1 - replacing with 0 and 1
5. diagnosis and procedure code has more null value (procedure code 5, 6 has all value as null - removed those two column)
6. removed 'NoOfMonths_PartACov', 'NoOfMonths_PartBCov' has same value for all row (it may be due to error)

5. Feature engineering

5.1. counting num of times patient made claim

(if a patient made more claim before, chnce of making fraud claim is more)

In [244]:

```
bene_count = final_data['BeneID'].value_counts().to_dict()
```

In [245]:

```
final_data['BeneCount'] = final_data['BeneID'].map(bene_count)
```

In [246]:

```
final_data['BeneCount'][0:5]
```

Out[246]:

```
0    3
1    3
2    2
3    4
4    7
Name: BeneCount, dtype: int64
```

5.2. counting num of times provider covered claim

(if a provider is more famous, chnce of making fraud claim is more)

In [247]:

```
pro_count = final_data['Provider'].value_counts().to_dict()
```

In [248]:

```
final_data['ProviderCount'] = final_data['Provider'].map(pro_count)
```

In [249]:

```
final_data['ProviderCount'][0:5]
```

Out[249]:

```
0    107
1    107
2    107
3    107
4    107
Name: ProviderCount, dtype: int64
```

5.3. counting num of times physician attended all patient

(if a physician has attended many patient, chnce of making fraud claim is more)

In [250]:

```
attphy_count = final_data['AttendingPhysician'].value_counts().to_dict()
```

In [251]:

```
final_data['AttendingPhysicianCount'] = final_data['AttendingPhysician'].map(attphy_count)
```

In [102]:

```
## replacing null with 0

final_data['AttendingPhysicianCount'] = final_data['AttendingPhysicianCount'].fillna(0)
```

In [252]:

```
## replacing null with mode

mode = final_data['AttendingPhysicianCount'].mode()[0]
print('mode :',mode)
final_data['AttendingPhysicianCount']=final_data['AttendingPhysicianCount'].fillna(mode)
```

```
final_data['AttendingPhysicianCount']=final_data['AttendingPhysicianCount'].fillna(mode)
```

```
mode : 1.0
```

In [253]:

```
final_data['AttendingPhysicianCount'][0:5]
```

Out[253]:

```
0    1.0
1    1.0
2    1.0
3    2.0
4    3.0
Name: AttendingPhysicianCount, dtype: float64
```

5.4. num of days patient admitted

(getting information from admissiondt,dischargedt)

In [254]:

```
df = pd.DataFrame(final_data,columns = ['ClaimStartDt','ClaimEndDt','AdmissionDt','DischargeDt'])
df.dtypes
```

Out[254]:

```
ClaimStartDt    object
ClaimEndDt      object
AdmissionDt     object
DischargeDt     object
dtype: object
```

In [255]:

```
## converting non-null object type of dt column to datetime

final_data['AdmissionDt']= pd.to_datetime(final_data['AdmissionDt'])
final_data['DischargeDt']= pd.to_datetime(final_data['DischargeDt'])
final_data['ClaimStartDt']= pd.to_datetime(final_data['ClaimStartDt'])
final_data['ClaimEndDt']= pd.to_datetime(final_data['ClaimEndDt'])

df = pd.DataFrame(final_data,columns = ['ClaimStartDt','ClaimEndDt','AdmissionDt','DischargeDt'])
df.dtypes
```

Out[255]:

```
ClaimStartDt    datetime64[ns]
ClaimEndDt      datetime64[ns]
AdmissionDt     datetime64[ns]
DischargeDt     datetime64[ns]
dtype: object
```

In [257]:

```
## filling na with 0

## calculating num of days admitted

final_data['numOfDaysAdmitted'] = final_data['DischargeDt'] - final_data['AdmissionDt']

## filling na with 0
final_data['numOfDaysAdmitted'] = final_data['numOfDaysAdmitted'].fillna(0)

## converting days type to int
final_data['numOfDaysAdmitted'] = final_data['numOfDaysAdmitted'].dt.days.astype('int64')
final_data['numOfDaysAdmitted'][0:5]
```

Out[257]:

```
0      6
1      0
2     12
3     18
4      4
Name: numOfDaysAdmitted, dtype: int64
```

In [258]:

```
## filling 0 with mean ( mode is 0 so mean is used )

mean = final_data['numOfDaysAdmitted'].mean()
mean = round(mean,1)
print('mean :',mean)
final_data['numOfDaysAdmitted']=final_data['numOfDaysAdmitted'].replace({0:mean})
final_data['numOfDaysAdmitted'][0:5]
```

```
mean : 0.4
```

Out[258]:

```
0      6.0
1      0.4
2     12.0
3     18.0
4      4.0
Name: numOfDaysAdmitted, dtype: float64
```

5.5. num of days for claim took to reimbursed

(getting information from ClaimStartDt,ClaimEndDt)

In [259]:

```
## filling na with 0

## calculating num of days for claim

final_data['numOfDaysForClaim'] = final_data['ClaimEndDt'] - final_data['ClaimStartDt']

## filling na with 0
final_data['numOfDaysForClaim'] = final_data['numOfDaysForClaim'].fillna(0)

## converting days type to int
final_data['numOfDaysForClaim'] = final_data['numOfDaysForClaim'].dt.days.astype('int64')
final_data['numOfDaysForClaim'][0:5]
```

Out[259]:

```
0      6
1      0
2     12
3     18
4      4
Name: numOfDaysForClaim, dtype: int64
```

In [260]:

```
## filling 0 with mean ( mode is 0 so mean is used )

mean = final_data['numOfDaysForClaim'].mean()
mean = round(mean,1)
print('mean :',mean)
final_data['numOfDaysForClaim']=final_data['numOfDaysForClaim'].replace({0:mean})
final_data['numOfDaysForClaim'][0:5]
```

```
mean : 1.7
```

Out[260]:

```
0      6.0
1      1.7
2     12.0
3     18.0
4      4.0
Name: numOfDaysForClaim, dtype: float64
```

5.6. calculating total ip,op amount reimburse

In [261]:

```
final_data[['IPAnnualReimbursementAmt','OPAnnualReimbursementAmt','IPAnnualDeductibleAmt','OPAnnual
DeductibleAmt']][0:5]
```

Out[261]:

	IPAnnualReimbursementAmt	OPAnnualReimbursementAmt	IPAnnualDeductibleAmt	OPAnnualDeductibleAmt
0	36000	60	3204	70
1	24000	450	2136	200
2	19000	100	1068	20
3	17000	1050	1068	540
4	27000	450	2136	160

In [262]:

```
## adding ip op amount

ip_op_total_amount = final_data['IPAnnualReimbursementAmt'] +
final_data['OPAnnualReimbursementAmt']
```

In [263]:

```
## adding deductible amount

ip_op_ded_amount = final_data['IPAnnualDeductibleAmt'] + final_data['OPAnnualDeductibleAmt']
```

In [264]:

```
## total amount - deductible amount

ip_op_total_amount = ip_op_total_amount - ip_op_ded_amount
```

In [265]:

```
final_data['ip_op_total_amount'] = ip_op_total_amount
```

In [266]:

```
final_data['ip_op_total_amount'][0:5]
```

Out[266]:

```
0     32786
1     22114
2     18012
3     16442
4     25154
Name: ip_op_total_amount, dtype: int64
```

5.7. calculating total disease patient was diagnosed before

In [267]:

```
## adding disease diagnosed on patient before

num_of_chronic = final_data['RenalDiseaseIndicator'] + final_data['ChronicCond_Alzheimer'] + \
                    final_data['ChronicCond_Heartfailure'] + final_data['ChronicCond_KidneyDisease'
] + \
                    final_data['ChronicCond_Cancer'] + final_data['ChronicCond_ObstrPulmonary'] + \
                    final_data['ChronicCond_Depression'] + final_data['ChronicCond_Diabetes'] + \
                    final_data['ChronicCond_IschemicHeart'] + final_data['ChronicCond_Osteoporasis'
] + \
                    final_data['ChronicCond_rheumatoidarthritis'] + final_data['ChronicCond_stroke'
]
```

In [268]:

```
final_data['num_of_chronic'] = num_of_chronic
final_data['num_of_chronic'][0:5]
```

Out[268]:

```
0    7
1    4
2    5
3    2
4    6
Name: num_of_chronic, dtype: int64
```

5.8. calculating num of diagnosis procedure undergonw by patient

In [269]:

```
## converting each row into array

num_of_diag_proc = final_data[['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',
       'DiagnosisGroupCode','ClmAdmitDiagnosisCode']].values
```

In [270]:

```
num_of_diag_proc
```

Out[270]:

```
array([['1970', '4019', '5853', ..., 4019.0, '201', '7866'],
       ['V7183', '53081', '78959', ..., 4019.0, '882', 'V7612'],
       ['4240', '2639', '2948', ..., 4019.0, '987', '45340'],
       ...,
       ['5854', '7907', '4019', ..., 4019.0, '882', 'V7612'],
       ['42789', '4019', '4019', ..., 4019.0, '882', 'V7612'],
       ['V5861', 'V454', '8404', ..., 4019.0, '882', 'V7612']], dtype=object)
```

In [118]:

```
len(num_of_diag_proc)
```

Out[118]:

```
558211
```

In [271]:

```
## counting non zero value in each row

countnum_of_diag_proc = []
for i in range(len(num_of_diag_proc)):
    countnum_of_diag_proc.append(np.count_nonzero(num_of_diag_proc[i]))
```

In [272]:

```
len(countnum_of_diag_proc)
```

Out[272]:

```
558211
```

In [273]:

```
final_data['num_of_diag_proc'] = countnum_of_diag_proc
```

In [274]:

```
final_data['num_of_diag_proc'][0:5]
```

Out[274]:

```
0    16
1    16
2    16
3    16
4    16
Name: num_of_diag_proc, dtype: int64
```

5.9. calculating num of physician treating patient

In [275]:

```
## fill na with 0

num_of_phy =
final_data[['AttendingPhysician','OperatingPhysician','OtherPhysician']].fillna(0).values
```

In [276]:

```
num_of_phy
```

Out[276]:

```
array([['PHY390922', 0, 0],
       ['PHY365867', 'PHY327147', 0],
       ['PHY349293', 'PHY370861', 'PHY363291'],
       ...,
       ['PHY338096', 0, 0],
       ['PHY416646', 0, 0],
       ['PHY392440', 0, 'PHY392440']], dtype=object)
```

In [277]:

```
## counting non zero value in each row

countnum_of_phy = []
for i in range(len(num_of_phy)):
    countnum_of_phy.append(np.count_nonzero(num_of_phy[i]))
```

In [278]:

```
final_data['num_of_phy'] = countnum_of_phy
```

In [279]:

```
final_data['num_of_phy'][0:5]
```

Out[279]:

```
0    1
```

```
1    2
2    3
3    2
4    3
Name: num_of_phy, dtype: int64
```

In [280]:

```
np.count_nonzero(final_data['num_of_phy'])
```

Out[280]:

```
556728
```

In [281]:

```
## replacing 0 with mode

mode = final_data['num_of_phy'].mode()[0]
print(mode)
final_data['num_of_phy']=final_data['num_of_phy'].replace({0:mode})
```

```
1
```

In [282]:

```
np.count_nonzero(final_data['num_of_phy'])
```

Out[282]:

```
558211
```

5.10. one hot encoding diagnosis code

number of category in each column is more than 1000

so picking top 10 category

encode column with value from top 10 category as 1 else 0

In [283]:

```
diagnosis_code = final_data[['ClmDiagnosisCode_1', 'ClmDiagnosisCode_2', 'ClmDiagnosisCode_3',
       'ClmDiagnosisCode_4', 'ClmDiagnosisCode_5', 'ClmDiagnosisCode_6',
       'ClmDiagnosisCode_7', 'ClmDiagnosisCode_8', 'ClmDiagnosisCode_9',
       'ClmDiagnosisCode_10', 'DiagnosisGroupCode','ClmAdmitDiagnosisCode']]
```

In [284]:

```
## total category in each column

for i in diagnosis_code.columns:
    print('num of category in',i,'is',len(diagnosis_code[i].unique()))
```

```
num of category in ClmDiagnosisCode_1 is 10450
num of category in ClmDiagnosisCode_2 is 5300
num of category in ClmDiagnosisCode_3 is 4756
num of category in ClmDiagnosisCode_4 is 4359
num of category in ClmDiagnosisCode_5 is 3970
num of category in ClmDiagnosisCode_6 is 3607
num of category in ClmDiagnosisCode_7 is 3388
num of category in ClmDiagnosisCode_8 is 3070
num of category in ClmDiagnosisCode_9 is 2774
num of category in ClmDiagnosisCode_10 is 1158
num of category in DiagnosisGroupCode is 736
num of category in ClmAdmitDiagnosisCode is 4098
```

In [285]:

```
## getting top 10 code

top10 = ['4019','25000','2724','V5869','4011','42731','V5861','2720','2449','4280']
```

In [286]:

```
## 1. pick one category 4019
## 2. pick one column (eg ClmDiagnosisCode_1)
## 3. add new column with name diagnosis 4019
## 4. using np.where check for category 4019 presnt int that column or not, if present replace wit
h 1 else 0, put in column diagnosis4019
## 5. again same 4019 is encoded for ClmDiagnosisCode_2
## 6. here it will take ClmDiagnosisCode_2 and check for 4019 category and replace with 1 if prese
nt,
      ## else it will check whether it is already encoded for ClmDiagnosisCode_1 ( if already
encoded means it will have value 1)
     ## here if 1 it will keep same value as 1, else 0
## 7. step 5 6 is repeated for all diagnosis column, after doing for all, next category is picked
and for all category
     ## here ten new column will be added, each row will represent whether that code is present or
not

for col in top10:
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_1']==col,1,0)
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_2']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_3']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_4']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_5']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_6']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_7']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_8']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_9']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmDiagnosisCode_10']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['DiagnosisGroupCode']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
    final_data['diagnosis_'+str(col)] = np.where(final_data['ClmAdmitDiagnosisCode']==col,1,\
                                   np.where(final_data['diagnosis_'+str(col)]==1,1,0 ))
```

5.11. one hot encoding procedure code

number of category very less and most of value is na. non na is around 0.8 %

so picking top 5 category

encode column with value from top 5 category as 1 else 0

In [287]:

```
procedure_code = final_data[['ClmProcedureCode_1', 'ClmProcedureCode_2',
       'ClmProcedureCode_3', 'ClmProcedureCode_4']]
```

In [288]:

```
## total category in each column

for i in procedure_code.columns:
    print('num of category in',i,'is',len(procedure_code[i].unique()))
```

```
num of category in ClmProcedureCode_1 is 1117
num of category in ClmProcedureCode_2 is 300
num of category in ClmProcedureCode_3 is 154
num of category in ClmProcedureCode_4 is 48
```

In [135]:

```
## procedure with na

final_data_proc = pd.DataFrame(columns = ['ProCode'])
final_data_proc['ProCode'] = pd.concat([final_data["ClmProcedureCode_1"],
                                        final_data["ClmProcedureCode_2"],
                                        final_data["ClmProcedureCode_3"],
                                        final_data["ClmProcedureCode_4"]], axis=0).dropna()

print(final_data_proc.shape)
```

```
(2232844, 1)
```

In [136]:

```
np.count_nonzero(final_data_proc)
```

Out[136]:

```
29887
```

In [137]:

```
print('percentage of non zero value in procedure :',(29887/2232844)*100)
```

```
percentage of non zero value in procedure : 1.338517155699189
```

In [289]:

```
## procedure without na

final_data_proc = pd.DataFrame(columns = ['ProCode'])
final_data_proc['ProCode'] = pd.concat([final_data["ClmProcedureCode_1"],
                                        final_data["ClmProcedureCode_2"],
                                        final_data["ClmProcedureCode_3"],
                                        final_data["ClmProcedureCode_4"]], axis=0).dropna()

print(final_data_proc.shape)
```

```
(2232844, 1)
```

In [290]:

```
np.count_nonzero(final_data_proc)
```

Out[290]:

```
2232844
```

In [291]:

```
## getting top 5 code

topproc = final_data_proc['ProCode'].value_counts()
topproc.index[0:10]
```

Out[291]:

```
Float64Index([4019.0, 9904.0, 2724.0, 8154.0, 66.0, 3893.0, 3995.0, 4516.0,
              3722.0, 8151.0],
             dtype='float64')
```

In [292]:

```
top5 = [4019.0, 9904.0, 2724.0, 8154.0, 66.0]
```

In [293]:

```
## 1. pick one category 4019
## 2. pick one column (eg ClmProcedureCode_1)
## 3. add new column with name diagnosis 4019
## 4. using np.where check for category 4019 presnt int that column or not, if present replace wit
h 1 else 0, put in column procedure4019
## 5. again same 4019 is encoded for ClmProcedureCode_2
## 6. here it will take ClmProcedureCode_2 and check for 4019 category and replace with 1 if prese
nt,
      ## else it will check whether it is already encoded for ClmProcedureCode_2 ( if already
encoded means it will have value 1)
     ## here if 1 it will keep same value as 1, else 0
## 7. step 5 6 is repeated for all procedure column, after doing for all, next category is picked
and for all category
     ## here new column will be added, each row will represent whether that code is present or not

for col in top5:
    final_data['procedure_'+str(col)] = np.where(final_data['ClmProcedureCode_1']==col,1,0)
    final_data['procedure_'+str(col)] = np.where(final_data['ClmProcedureCode_2']==col,1,\
                                  np.where(final_data['procedure_'+str(col)]==1,1,0 ))
    final_data['procedure_'+str(col)] = np.where(final_data['ClmProcedureCode_3']==col,1,\
                                  np.where(final_data['procedure_'+str(col)]==1,1,0 ))
    final_data['procedure_'+str(col)] = np.where(final_data['ClmProcedureCode_4']==col,1,\
                                  np.where(final_data['procedure_'+str(col)]==1,1,0 ))
```

5.12. frequency encoding diagnosis, procedure code

```
counting num of times code occured, replacing with count
```

In [141]:

```
## col with 0 value present

DiagnosisCode_1_count = final_data['ClmDiagnosisCode_1'].value_counts().to_dict()
DiagnosisCode_1_count[0]=0

DiagnosisCode_2_count = final_data['ClmDiagnosisCode_2'].value_counts().to_dict()
DiagnosisCode_2_count[0]=0

DiagnosisCode_3_count = final_data['ClmDiagnosisCode_3'].value_counts().to_dict()
DiagnosisCode_3_count[0]=0

DiagnosisCode_4_count = final_data['ClmDiagnosisCode_4'].value_counts().to_dict()
DiagnosisCode_4_count[0]=0

DiagnosisCode_5_count = final_data['ClmDiagnosisCode_5'].value_counts().to_dict()
DiagnosisCode_5_count[0]=0

DiagnosisCode_6_count = final_data['ClmDiagnosisCode_6'].value_counts().to_dict()
DiagnosisCode_6_count[0]=0

DiagnosisCode_7_count = final_data['ClmDiagnosisCode_7'].value_counts().to_dict()
DiagnosisCode_7_count[0]=0

DiagnosisCode_8_count = final_data['ClmDiagnosisCode_8'].value_counts().to_dict()
DiagnosisCode_8_count[0]=0

DiagnosisCode_9_count = final_data['ClmDiagnosisCode_9'].value_counts().to_dict()
DiagnosisCode_9_count[0]=0

DiagnosisCode_10_count = final_data['ClmDiagnosisCode_10'].value_counts().to_dict()
DiagnosisCode_10_count[0]=0

ClmAdmitDiagnosisCode_count = final_data['ClmAdmitDiagnosisCode'].value_counts().to_dict()
ClmAdmitDiagnosisCode_count[0]=0

DiagnosisGroupCode_count = final_data['DiagnosisGroupCode'].value_counts().to_dict()
DiagnosisGroupCode_count[0]=0
```

In [294]:

```
## col with mode value present

DiagnosisCode_1_count = final_data['ClmDiagnosisCode_1'].value_counts().to_dict()

DiagnosisCode_2_count = final_data['ClmDiagnosisCode_2'].value_counts().to_dict()

DiagnosisCode_3_count = final_data['ClmDiagnosisCode_3'].value_counts().to_dict()

DiagnosisCode_4_count = final_data['ClmDiagnosisCode_4'].value_counts().to_dict()

DiagnosisCode_5_count = final_data['ClmDiagnosisCode_5'].value_counts().to_dict()

DiagnosisCode_6_count = final_data['ClmDiagnosisCode_6'].value_counts().to_dict()

DiagnosisCode_7_count = final_data['ClmDiagnosisCode_7'].value_counts().to_dict()

DiagnosisCode_8_count = final_data['ClmDiagnosisCode_8'].value_counts().to_dict()

DiagnosisCode_9_count = final_data['ClmDiagnosisCode_9'].value_counts().to_dict()

DiagnosisCode_10_count = final_data['ClmDiagnosisCode_10'].value_counts().to_dict()

ClmAdmitDiagnosisCode_count = final_data['ClmAdmitDiagnosisCode'].value_counts().to_dict()

DiagnosisGroupCode_count = final_data['DiagnosisGroupCode'].value_counts().to_dict()
```

In [295]:

```
final_data['DiagnosisCode_1_count'] = final_data['ClmDiagnosisCode_1'].map(DiagnosisCode_1_count)
final_data['DiagnosisCode_2_count'] = final_data['ClmDiagnosisCode_2'].map(DiagnosisCode_2_count)
final_data['DiagnosisCode_3_count'] = final_data['ClmDiagnosisCode_3'].map(DiagnosisCode_3_count)
final_data['DiagnosisCode_4_count'] = final_data['ClmDiagnosisCode_4'].map(DiagnosisCode_4_count)
final_data['DiagnosisCode_5_count'] = final_data['ClmDiagnosisCode_5'].map(DiagnosisCode_5_count)
final_data['DiagnosisCode_6_count'] = final_data['ClmDiagnosisCode_6'].map(DiagnosisCode_6_count)
final_data['DiagnosisCode_7_count'] = final_data['ClmDiagnosisCode_7'].map(DiagnosisCode_7_count)
final_data['DiagnosisCode_8_count'] = final_data['ClmDiagnosisCode_8'].map(DiagnosisCode_8_count)
final_data['DiagnosisCode_9_count'] = final_data['ClmDiagnosisCode_9'].map(DiagnosisCode_9_count)
final_data['DiagnosisCode_10_count'] =
final_data['ClmDiagnosisCode_10'].map(DiagnosisCode_10_count)
final_data['ClmAdmitDiagnosisCode_count'] =
final_data['ClmAdmitDiagnosisCode'].map(ClmAdmitDiagnosisCode_count)
final_data['DiagnosisGroupCode_count'] =
final_data['DiagnosisGroupCode'].map(DiagnosisGroupCode_count)
```

In [296]:

```
final_data.columns
```

Out[296]:

```
Index(['BeneID', 'ClaimID', 'ClaimStartDt', 'ClaimEndDt', 'Provider',
       'InscClaimAmtReimbursed', 'AttendingPhysician', 'OperatingPhysician',
       'OtherPhysician', 'AdmissionDt', 'ClmAdmitDiagnosisCode',
       'DeductibleAmtPaid', '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', 'DOB', 'DOD', 'Gender',
       'Race', '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', 'IPAnnualReimbursementAmt',
       'IPAnnualDeductibleAmt', 'OPAnnualReimbursementAmt',
       'OPAnnualDeductibleAmt', 'PotentialFraud', 'BeneCount', 'ProviderCount',
       'AttendingPhysicianCount', 'numOfDaysAdmitted', 'numOfDaysForClaim',
       'ip_op_total_amount', 'num_of_chronic', 'num_of_diag_proc',
       'num_of_phy', 'diagnosis_4019', 'diagnosis_25000', 'diagnosis_2724',
       'diagnosis_V5869', 'diagnosis_4011', 'diagnosis_42731',
       'diagnosis_V5861', 'diagnosis_2720', 'diagnosis_2449', 'diagnosis_4280',
       'procedure_4019.0', 'procedure_9904.0', 'procedure_2724.0',
       'procedure_8154.0', 'procedure_66.0', 'DiagnosisCode_1_count',
       'DiagnosisCode_2_count', 'DiagnosisCode_3_count',
       'DiagnosisCode_4_count', 'DiagnosisCode_5_count',
       'DiagnosisCode_6_count', 'DiagnosisCode_7_count',
       'DiagnosisCode_8_count', 'DiagnosisCode_9_count',
       'DiagnosisCode_10_count', 'ClmAdmitDiagnosisCode_count',
       'DiagnosisGroupCode_count'],
      dtype='object')
```

In [297]:

```
final_data.isnull().sum()
```

Out[297]:

```
BeneID                             0
ClaimID                            0
ClaimStartDt                       0
ClaimEndDt                         0
Provider                           0
InscClaimAmtReimbursed             0
AttendingPhysician              1508
OperatingPhysician            443764
OtherPhysician                358475
AdmissionDt                   517737
ClmAdmitDiagnosisCode              0
DeductibleAmtPaid                899
DischargeDt                   517737
DiagnosisGroupCode                 0
ClmDiagnosisCode_1                 0
ClmDiagnosisCode_2                 0
ClmDiagnosisCode_3                 0
ClmDiagnosisCode_4                 0
ClmDiagnosisCode_5                 0
ClmDiagnosisCode_6                 0
ClmDiagnosisCode_7                 0
ClmDiagnosisCode_8                 0
ClmDiagnosisCode_9                 0
ClmDiagnosisCode_10                0
ClmProcedureCode_1                 0
ClmProcedureCode_2                 0
ClmProcedureCode_3                 0
ClmProcedureCode_4                 0
DOB                                0
DOD                           554080
                               ...
num_of_chronic                     0
num_of_diag_proc                   0
num_of_phy                         0
diagnosis_4019                     0
diagnosis_25000                    0
diagnosis_2724                     0
diagnosis_V5869                    0
diagnosis_4011                     0
diagnosis_42731                    0
diagnosis_V5861                    0
diagnosis_2720                     0
diagnosis_2449                     0
diagnosis_4280                     0
procedure_4019.0                   0
procedure_9904.0                   0
procedure_2724.0                   0
procedure_8154.0                   0
procedure_66.0                     0
DiagnosisCode_1_count              0
DiagnosisCode_2_count              0
DiagnosisCode_3_count              0
DiagnosisCode_4_count              0
DiagnosisCode_5_count              0
DiagnosisCode_6_count              0
DiagnosisCode_7_count              0
DiagnosisCode_8_count              0
DiagnosisCode_9_count              0
```

```
DiagnosisCode_10_count              0
ClmAdmitDiagnosisCode_count         0
DiagnosisGroupCode_count            0
dtype: int64
```

In [ ]:

```
## DeductibleAmtPaid missing value
```

In [298]:

```
np.count_nonzero(final_data['DeductibleAmtPaid'])
```

Out[298]:

```
61510
```

In [146]:

```
## replacing null with 0

final_data['DeductibleAmtPaid']= final_data['DeductibleAmtPaid'].replace({np.nan:0})
```

In [299]:

```
## replacing null with mean

mean = final_data['DeductibleAmtPaid'].mean()
mean = round(mean,0)
print('mean :',mean)
final_data['DeductibleAmtPaid']=final_data['DeductibleAmtPaid'].replace({np.nan:mean})
```

```
mean : 78.0
```

In [300]:

```
final_data['DeductibleAmtPaid'].isnull().sum()
```

Out[300]:

```
0
```

In [ ]:

```
## dropping col after featurization
```

In [301]:

```
final_data = final_data.drop(['BeneID', 'ClaimID', 'ClaimStartDt', 'ClaimEndDt',
'Provider','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','DOB', 'DOD'],axis=1)
```

In [302]:

```
final_data.columns
```

Out[302]:

```
Index(['InscClaimAmtReimbursed', 'DeductibleAmtPaid', 'Gender', 'Race',
       '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',
       'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
       'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt', 'PotentialFraud',
       'BeneCount', 'ProviderCount', 'AttendingPhysicianCount',
       'numOfDaysAdmitted', 'numOfDaysForClaim', 'ip_op_total_amount',
       'num_of_chronic', 'num_of_diag_proc', 'num_of_phy', 'diagnosis_4019',
       'diagnosis_25000', 'diagnosis_2724', 'diagnosis_V5869',
       'diagnosis_4011', 'diagnosis_42731', 'diagnosis_V5861',
       'diagnosis_2720', 'diagnosis_2449', 'diagnosis_4280',
       'procedure_4019.0', 'procedure_9904.0', 'procedure_2724.0',
       'procedure_8154.0', 'procedure_66.0', 'DiagnosisCode_1_count',
       'DiagnosisCode_2_count', 'DiagnosisCode_3_count',
       'DiagnosisCode_4_count', 'DiagnosisCode_5_count',
       'DiagnosisCode_6_count', 'DiagnosisCode_7_count',
       'DiagnosisCode_8_count', 'DiagnosisCode_9_count',
       'DiagnosisCode_10_count', 'ClmAdmitDiagnosisCode_count',
       'DiagnosisGroupCode_count'],
      dtype='object')
```

new col added

```
'BeneCount', 'ProviderCount', 'AttendingPhysicianCount',
'numOfDaysAdmitted', 'numOfDaysForClaim', 'ip_op_total_amount',
   'num_of_chronic', 'num_of_diag_proc', 'num_of_phy', 'diagnosis_4019',
   'diagnosis_25000', 'diagnosis_2724', 'diagnosis_V5869',
   'diagnosis_4011', 'diagnosis_42731', 'diagnosis_V5861',
   'diagnosis_2720', 'diagnosis_2449', 'diagnosis_4280',
   'procedure_4019.0', 'procedure_9904.0', 'procedure_2724.0',
   'procedure_8154.0', 'procedure_66.0', 'DiagnosisCode_1_count',
   'DiagnosisCode_2_count', 'DiagnosisCode_3_count',
   'DiagnosisCode_4_count', 'DiagnosisCode_5_count',
   'DiagnosisCode_6_count', 'DiagnosisCode_7_count',
   'DiagnosisCode_8_count', 'DiagnosisCode_9_count',
   'DiagnosisCode_10_count', 'ClmAdmitDiagnosisCode_count',
   'DiagnosisGroupCode_count'
```

summary

frequency encoding

```
1. BeneCount - counting num of times patient made claim
2. ProviderCount - counting num of times provider covered claim
3. AttendingPhysician - counting num of times physician attended all patient
4. num of days patient admitted ( DischargeDt - AdmissionDt )
5. num of days for claim took to reimbursed ( ClaimEndDt - ClaimStartDt )
6. ip_op_total_amount - calculating total ip,op amount reimburse (
(IPAnnualReimbursementAmt + OPAnnualReimbursementAmt) - (IPAnnualDeductibleAmt +
OPAnnualDeductibleAmt) )
7. num_of_chronic - calculating total disease patient was diagnosed before ( adding all chr
onic disease )
8. num_of_diag_proc - calculating num of diagnosis procedure undergonw by patient
9. num_of_phy - calculating num of physician treating patient ( AttendingPhysician +
OperatingPhysician + OtherPhysician )
10. [ 'DiagnosisCode_1_count',
   'DiagnosisCode_2_count', 'DiagnosisCode_3_count',
   'DiagnosisCode_4_count', 'DiagnosisCode_5_count',
   'DiagnosisCode_6_count', 'DiagnosisCode_7_count',
   'DiagnosisCode_8_count', 'DiagnosisCode_9_count',
   'DiagnosisCode_10_count', 'ClmAdmitDiagnosisCode_count',
   'DiagnosisGroupCode_count' ] - counting num of times code occured, replacing with count
```

one hot encoding

```
    1. diagnosis, procedure code - each diagnosis column has more than 1000 category, so
took top ten code in diagnosis and top  five code in procedure and performed one hot encodi
```

```
ng on that code only
     [ 'diagnosis_4019',
    'diagnosis_25000', 'diagnosis_2724', 'diagnosis_V5869',
    'diagnosis_4011', 'diagnosis_42731', 'diagnosis_V5861',
    'diagnosis_2720', 'diagnosis_2449', 'diagnosis_4280',
    'procedure_4019.0', 'procedure_9904.0', 'procedure_2724.0',
    'procedure_8154.0', 'procedure_66.0' ]

after performing feature engineering all column with string value are removed like
BeneID,Provider..
```

In [303]:

```
final_data.head()
```

Out[303]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCond_Alzhe
0	26000.0	1068.0	1	1	0	39	230	1
1	50.0	0.0	1	1	0	39	310	1
2	19000.0	1068.0	0	1	0	39	230	1
3	17000.0	1068.0	1	1	0	39	600	0
4	13000.0	1068.0	0	1	0	39	280	0

5 rows × 59 columns

In [151]:

```
final_data.to_csv('final_data_feature.csv')
```

In [304]:

```
final_data.to_csv('final_data_feature_mode.csv')
```

In [60]:

```
## file with null filled with 0

final_data = pd.read_csv('final_data_feature.csv')
final_data.drop('Unnamed: 0',axis=1,inplace=True)
```

In [80]:

```
## file with null filled with mode

final_data = pd.read_csv('final_data_feature_mode.csv')
final_data.drop('Unnamed: 0',axis=1,inplace=True)
```

In [61]:

```
## file with null filled with 0

final_data.head()
```

Out[61]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCond_Alzhe
0	26000.0	1068.0	1	1	0	39	230	1
1	50.0	0.0	1	1	0	39	310	1
2	19000.0	1068.0	0	1	0	39	230	1
3	17000.0	1068.0	1	1	0	39	600	0

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCond_Alzhe
4	13000.0	1068.0	0	1	0	39	280	0

5 rows × 59 columns

In [81]:

```
## file with null filled with mode

final_data.head()
```

Out[81]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCond_Alzhe
0	26000.0	1068.0	1	1	0	39	230	1
1	50.0	0.0	1	1	0	39	310	1
2	19000.0	1068.0	0	1	0	39	230	1
3	17000.0	1068.0	1	1	0	39	600	0
4	13000.0	1068.0	0	1	0	39	280	0

5 rows × 59 columns

In [82]:

```
## dividing data and class label

Y = final_data['PotentialFraud']
X = final_data.drop('PotentialFraud',axis=1)
```

In [83]:

```
X.shape
```

Out[83]:

```
(558211, 58)
```

In [84]:

```
Y.shape
```

Out[84]:

```
(558211,)
```

train test split

In [85]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, stratify = Y)
```

In [66]:

```
print('X_train size :',X_train.shape)
print('Y_train size :',Y_train.shape)
print('X_test size :',X_test.shape)
print('Y_test size :',Y_test.shape)
```

```
X_train size : (374001, 58)
Y_train size : (374001,)
X_test size : (184210, 58)
Y_test size : (184210,)
```

In [67]:

```
X_train.head()
```

Out[67]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCond_
440173	100.0	0.0	0	1	0	11	260	1
208392	40.0	0.0	1	1	0	10	400	1
302778	400.0	0.0	0	2	0	49	891	0
436736	3000.0	1068.0	0	1	0	5	530	1
250221	40.0	0.0	1	1	0	13	0	0

5 rows × 58 columns

normalizing using min max scalar

In [86]:

```
min_max_scaler = preprocessing.MinMaxScaler()
```

In [87]:

```
def min_max(train,test,column):
    """ scaling column value using min max scalar, fitting on train and transforming """

    min_max_scaler.fit(train[column].values.reshape(-1,1))

    train_scale = min_max_scaler.transform(train[column].values.reshape(-1,1))
    test_scale = min_max_scaler.transform(test[column].values.reshape(-1,1))

    return train_scale,test_scale
```

In [88]:

```
## col to normalize

col_to_nor = ['InscClaimAmtReimbursed', 'DeductibleAmtPaid', 'Race',
       'State', 'County', 'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
       'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt', 'BeneCount',
       'ProviderCount', 'AttendingPhysicianCount', 'numOfDaysAdmitted',
       'numOfDaysForClaim', 'ip_op_total_amount', 'num_of_chronic',
       'num_of_diag_proc', 'num_of_phy', 'DiagnosisCode_1_count', 'DiagnosisCode_2_count',
       'DiagnosisCode_3_count', 'DiagnosisCode_4_count',
       'DiagnosisCode_5_count', 'DiagnosisCode_6_count',
       'DiagnosisCode_7_count', 'DiagnosisCode_8_count',
       'DiagnosisCode_9_count', 'DiagnosisCode_10_count',
       'ClmAdmitDiagnosisCode_count', 'DiagnosisGroupCode_count']

for i in col_to_nor:
    train_scale,test_scale = min_max(X_train,X_test,i)

    X_train[i] = train_scale
    X_test[i] = test_scale
```

In [71]:

```
X_train.head()
```

Out[71]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicC
440173	0.00080	0.0	0	0.00	0	0.188679	0.260260	1
208392	0.00032	0.0	1	0.00	0	0.169811	0.400400	1

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicC
302778	0.00320	0.0	0	0.25	0	0.905660	0.891892	0
436736	0.02400	1.0	0	0.00	0	0.075472	0.530531	1
250221	0.00032	0.0	1	0.00	0	0.226415	0.000000	0

5 rows × 58 columns

In [89]:

```
## with mode

X_train.head()
```

Out[89]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicC
109382	0.00080	0.0	0	0.00	0	0.169811	0.12012	0
243180	0.00008	0.0	1	0.25	0	0.415094	0.31031	0
99226	0.00008	0.0	1	0.00	0	0.641509	0.08008	1
431832	0.00040	0.0	1	0.00	0	0.962264	0.04004	0
369738	0.00016	0.0	0	0.00	0	0.603774	0.37037	0

5 rows × 58 columns

6. model

In [90]:

```
def find_best_threshold(threshold, fpr, tpr):
    """ finding best threshold value """

    best_t = threshold[np.argmax(tpr*(1-fpr))]
    return best_t


def predict_using_best_threshold(probability, threshold):
    """ convert proba value using best threshold """

    predictions = []
    for i in probability:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

def plot_confusion_matrix(clf,x_train,x_test,y_train,y_test):
    """ ploting confusion matrix using best threshold """

    ## fpr,tpr,threshold
    y_train_pred = clf.predict_proba(x_train)
    y_train_pred = y_train_pred[:, 1]

    y_test_pred = clf.predict_proba(x_test)
    y_test_pred = y_test_pred[:, 1]


    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
    test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)


    #tn, fp, fn, tp


    ## confusion matrix

    ## train
    best_tr_threshold = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
    print('best train threshold :', best_tr_threshold)
    print('best train threshold :',best_tr_threshold)
    train_conmat = confusion_matrix(y_train, predict_using_best_threshold(y_train_pred, best_tr_thr
eshold))

    sns.heatmap(train_conmat,xticklabels=['Predicted No','Predicted Yes'],yticklabels=['Actual No'
,'Actual Yes'],annot=True,fmt='d')
    plt.title('confusion matrix on train')
    plt.show()


    ## test
    best_te_threshold = find_best_threshold(te_thresholds, test_fpr, test_tpr)
    print('best test threshold :',best_te_threshold)
    test_conmat = confusion_matrix(y_test, predict_using_best_threshold(y_test_pred, best_te_thresh
old))

    sns.heatmap(test_conmat,xticklabels=['Predicted No','Predicted Yes'],yticklabels=['Actual No','
Actual Yes'],annot=True,fmt='d')
    plt.title('confusion matrix on test')
    plt.show()
```

In [91]:

```
def f1(model,x_train,x_test,y_train,y_test):
    """ calculate f1, macro f1 score """
    y_train_pred = model.predict_proba(x_train)
    y_train_pred = y_train_pred[:, 1]

    y_test_pred = model.predict_proba(x_test)
    y_test_pred = y_test_pred[:, 1]

    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
    test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

    best_tr_threshold = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
    best_te_threshold = find_best_threshold(te_thresholds, test_fpr, test_tpr)

    f1_train = f1_score(y_train, predict_using_best_threshold(y_train_pred, best_tr_threshold))

    f1_test = f1_score(y_test, predict_using_best_threshold(y_test_pred, best_te_threshold))

    macro_f1_train = f1_score(y_train, predict_using_best_threshold(y_train_pred, best_tr_threshold
),average='macro')

    macro_f1_test = f1_score(y_test, predict_using_best_threshold(y_test_pred, best_te_threshold),a
verage='macro')

    return f1_train,f1_test,macro_f1_train,macro_f1_test
```

In [92]:

```
def important_feature(model,data):
    """ print important feature """
    table = PrettyTable(['feature','weight'])

    feature = data.columns   ## getting column name
    feature_importances = model.feature_importances_  ## getting feature weight
    indices = (np.argsort(feature_importances))[-20:]  ## getting impor feature weight
    indices = list(indices)[::-1]  ## getting in descending
    print('important features :')
    for i in indices:
        table.add_row([feature[i],np.round(feature_importances[i],5)])
    return table
```

6.1. LogisticRegression

hyperparameter tuning using random search

In [123]:

```
param = { 'C': [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000], 'penalty' : ['l1','l2'] }
lr = LogisticRegression()
```

```
lr_tune = RandomizedSearchCV(lr,param,cv=10,n_jobs=-1,verbose=1)
lr_tune.fit(X_train,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 12.2min finished
```

Out[123]:

```
RandomizedSearchCV(cv=10, error_score='raise-deprecating',
          estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'C': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000], 'penalty':
['l1', 'l2']},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=1)
```

6.1.1. model with null filled with 0

In [65]:

```
print('best parameter :',lr_tune.best_params_)
print("accuracy on train :",lr_tune.best_score_)
```

```
best parameter : {'C': 0.1, 'penalty': 'l2'}
accuracy on train : 0.820219732033
```

In [66]:

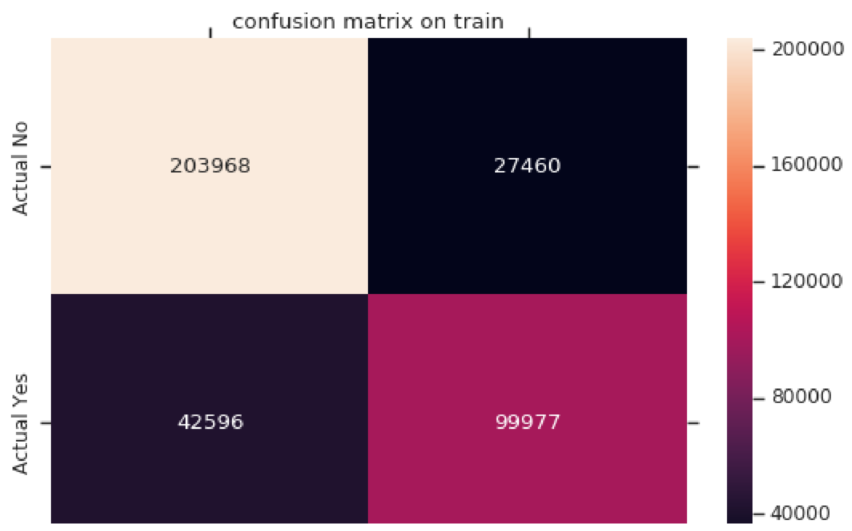
```
## on best parameter

lr_tuned = LogisticRegression(C=0.1,penalty='l2')
lr_tuned.fit(X_train,Y_train)
print('accuracy on test :',lr_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.822420064057
```

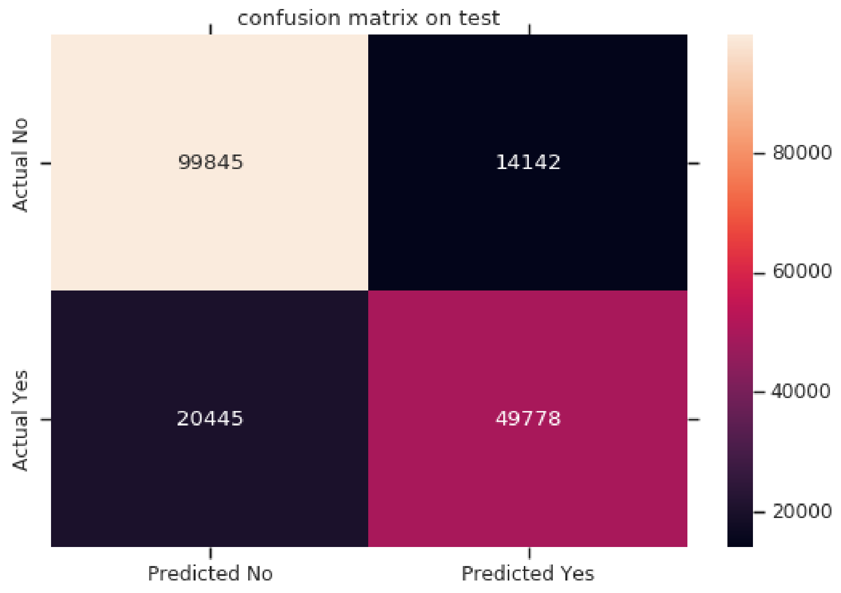
In [72]:

```
print(plot_confusion_matrix(lr_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.441949667403
```



```
best test threshold : 0.434832439881
```



```
None
```

In [73]:

```
## checking confusion matrix value with threshold 0.5 and with best threshold wuth roc_curve
## we need high recall, so false negative ( actual fraud but predicted as non fraud ) is less
## so here true positive increase and false negative decrease

## true positive in train with threshold 0.5 - 94705
## true positive in train with best threshold -  99988

## false negative in train with threshold 0.5 - 47868
## false negative in train with best threshold -  42585


y_train_pred = lr_tuned.predict(X_train)
y_test_pred = lr_tuned.predict(X_test)

train_conf_matrix = confusion_matrix(Y_train, y_train_pred)
test_conf_matrix = confusion_matrix(Y_test, y_test_pred)
print('train confusion matrix without best threshold:\n',train_conf_matrix)
print('test confusion matrix without best threshold:\n',test_conf_matrix)

## fpr,tpr,threshold
y_train_pred = lr_tuned.predict_proba(X_train)
y_train_pred = y_train_pred[:, 1]

y_test_pred = lr_tuned.predict_proba(X_test)
y_test_pred = y_test_pred[:, 1]


train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

#tn, fp, fn, tp

## train
best_tr_threshold = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('best train threshold :',best_tr_threshold)
train_conmat = confusion_matrix(Y_train, predict_using_best_threshold(y_train_pred,
best_tr_threshold))

## test
```

```
best_te_threshold = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print('best test threshold :',best_te_threshold)
test_conmat = confusion_matrix(Y_test, predict_using_best_threshold(y_test_pred, best_te_threshold)
)


print('train confusion matrix with best threshold:\n',train_conmat)
print('train confusion matrix with best threshold:\n',test_conmat)
```

```
train confusion matrix without best threshold:
 [[212273  19155]
 [ 47949  94624]]
test confusion matrix without best threshold:
 [[104615   9372]
 [ 23429  46794]]
best train threshold : 0.441949667403
best test threshold : 0.434832439881
train confusion matrix with best threshold:
 [[203968  27460]
 [ 42596  99977]]
train confusion matrix with best threshold:
 [[99845 14142]
 [20445 49778]]
```

In [81]:

```
## f1 score

f1_train,f1_test,macro_f1_train,macro_f1_test = f1(lr_tuned,X_train,X_test,Y_train,Y_test)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
```

```
Train F1 score is : 0.740054326507
Test F1 score is : 0.743772804213
Train macro F1 score is : 0.796519050801
Test macro F1 score is : 0.799572764874
```

In [71]:

```
import joblib
joblib.dump(lr_tuned, 'lr_tuned_on_0.pkl')
```

Out[71]:

```
['lr_tuned_on_0.pkl']
```

In [58]:

```
lr_tuned = joblib.load('lr_tuned_on_0.pkl')
```

In [83]:

```
table = PrettyTable(['feature','weight'])

feature = X.columns
feature_importances = lr_tuned.coef_
indices = (np.argsort(feature_importances)).tolist()
print('important features :')
for i in indices[0]:
    table.add_row([feature[i],np.round(feature_importances[0][i],5)])
print(table)
```

```
important features :
+----------------------------------+----------+
|             feature              |  weight  |
+----------------------------------+----------+
|     AttendingPhysicianCount      | -2.24543 |
|     OPAnnualReimbursementAmt     | -0.34694 |
|        ip_op_total_amount        | -0.26395 |
|       DiagnosisCode_10_count       | -0.24388 |
|       IPAnnualDeductibleAmt        | -0.13677 |
|               State                | -0.1361  |
|       OPAnnualDeductibleAmt        | -0.11795 |
|          diagnosis_V5869           | -0.06226 |
|           diagnosis_4011           | -0.04728 |
|       DiagnosisCode_6_count        | -0.04519 |
|       DiagnosisCode_3_count        | -0.03324 |
|           diagnosis_2720           | -0.03248 |
|       ChronicCond_Alzheimer        | -0.03038 |
|           diagnosis_4019           | -0.02581 |
|          diagnosis_V5861           | -0.02543 |
|        ChronicCond_Diabetes        | -0.01913 |
|             num_of_phy             | -0.01899 |
|       ChronicCond_Depression       | -0.0171  |
|           diagnosis_2449           | -0.01468 |
|          procedure_4019.0          | -0.01389 |
|       DiagnosisCode_2_count        |  -0.01   |
|      ChronicCond_Heartfailure      | -0.00886 |
|     ChronicCond_KidneyDisease      | -0.0081  |
|     ChronicCond_ObstrPulmonary     | -0.00803 |
| ChronicCond_rheumatoidarthritis    | -0.00777 |
|         ChronicCond_Cancer         | -0.00523 |
|           num_of_chronic           | -0.00363 |
|          procedure_9904.0          | -0.00309 |
|               Gender               | -0.00178 |
|          diagnosis_25000           | 0.00036  |
|      ChronicCond_Osteoporasis      |  0.0004  |
|     ChronicCond_IschemicHeart      | 0.00046  |
|         numOfDaysForClaim          | 0.00652  |
|       DiagnosisCode_1_count        | 0.00677  |
|    ClmAdmitDiagnosisCode_count     | 0.00898  |
|           diagnosis_2724           | 0.01007  |
|           diagnosis_4280           | 0.01037  |
|       RenalDiseaseIndicator        | 0.02156  |
|          diagnosis_42731           |  0.0287  |
|       DiagnosisCode_4_count        | 0.03473  |
|       DiagnosisCode_9_count        | 0.03585  |
|         ChronicCond_stroke         | 0.03868  |
|       DiagnosisCode_5_count        | 0.04004  |
|       DiagnosisCode_7_count        | 0.04617  |
|       DiagnosisCode_8_count        | 0.04668  |
|           procedure_66.0           | 0.04684  |
|                Race                | 0.08937  |
|      IPAnnualReimbursementAmt      | 0.12158  |
|          procedure_2724.0          | 0.13576  |
|          num_of_diag_proc          | 0.13675  |
|          procedure_8154.0          | 0.22731  |
|      DiagnosisGroupCode_count      | 0.28692  |
|               County               |  0.3942  |
|             BeneCount              | 0.44499  |
|         numOfDaysAdmitted          | 0.44936  |
|       InscClaimAmtReimbursed       | 0.79983  |
|         DeductibleAmtPaid          | 1.74061  |
|           ProviderCount            | 24.24234 |
+------------------------------------+----------+
```

6.1.2. model with null filled with mode

In [124]:

```
print('best parameter :',lr_tune.best_params_)
print("accuracy on train :",lr_tune.best_score_)
```

```
best parameter : {'C': 0.01, 'penalty': 'l2'}
accuracy on train : 0.830225052874
```

In [125]:

```
## on best parameter

lr_tuned = LogisticRegression(C=0.01,penalty='l2')
lr_tuned.fit(X_train,Y_train)
```

```
print('accuracy on test :',lr_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.83012865751
```

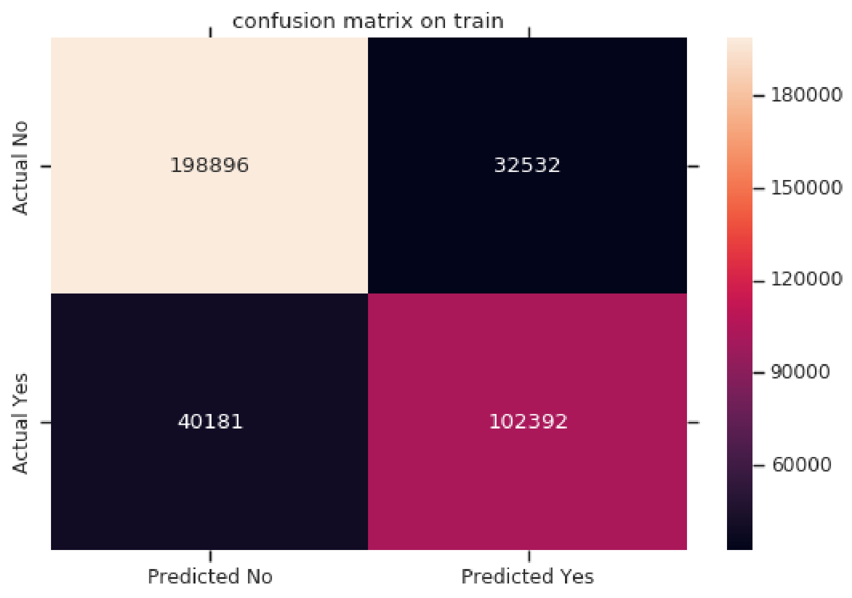
In [93]:

```
lr_tuned = joblib.load('lr_tuned_on_mode.pkl')
```

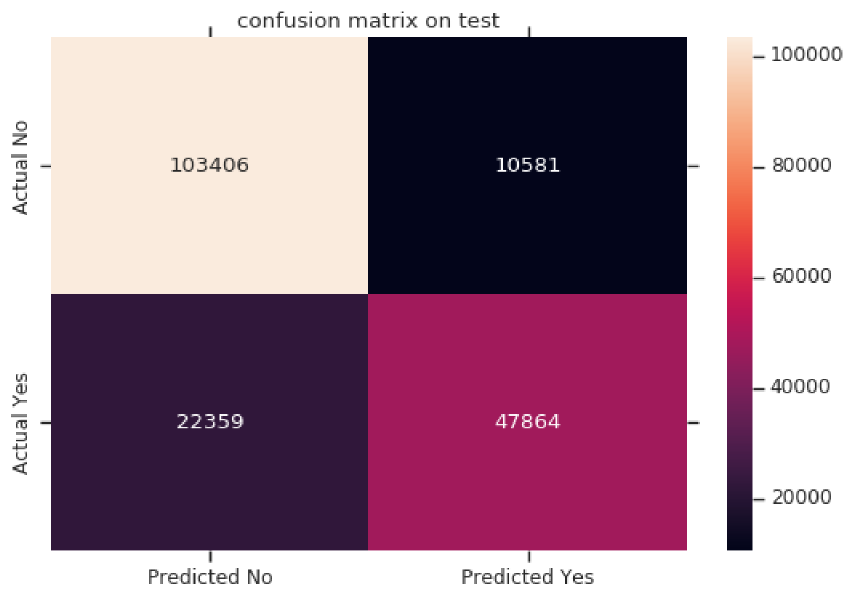
In [94]:

```
print(plot_confusion_matrix(lr_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.395049450959
```



```
best test threshold : 0.461054714066
```



```
None
```

In [127]:

```
## f1 score
```

```
f1_train,f1_test,macro_f1_train,macro_f1_test = f1(lr_tuned,X_train,X_test,Y_train,Y_test)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
```

```
Train F1 score is : 0.742916034512
Test F1 score is : 0.739406238366
Train macro F1 score is : 0.802563378762
Test macro F1 score is : 0.792575792876
```

In [128]:

```
joblib.dump(lr_tuned, 'lr_tuned_on_mode.pkl')
```

Out[128]:

```
['lr_tuned_on_mode.pkl']
```

In [129]:

```
table = PrettyTable(['feature','weight'])

feature = X.columns
feature_importances = lr_tuned.coef_
indices = (np.argsort(feature_importances)).tolist()
print('important features :')
for i in indices[0]:
    table.add_row([feature[i],np.round(feature_importances[0][i],5)])
print(table)
```

```
important features :
+----------------------------------+----------+
|             feature              |  weight  |
+----------------------------------+----------+
|     DiagnosisGroupCode_count     | -1.11594 |
|         procedure_4019.0         | -0.35876 |
|     OPAnnualReimbursementAmt     | -0.11609 |
|      OPAnnualDeductibleAmt       | -0.09376 |
|          diagnosis_4019          |  -0.085  |
|         procedure_9904.0         | -0.06758 |
|      DiagnosisCode_10_count      | -0.06651 |
|        ip_op_total_amount        | -0.06201 |
|      IPAnnualDeductibleAmt       | -0.05437 |
|      DiagnosisCode_8_count       | -0.03432 |
|              State               | -0.03392 |
|         diagnosis_V5869          | -0.03382 |
|          diagnosis_2720          | -0.0244  |
|       ChronicCond_Cancer         | -0.02425 |
|      ChronicCond_Diabetes        | -0.02308 |
|      ChronicCond_Alzheimer       | -0.02191 |
|      ChronicCond_Depression      | -0.02171 |
|          diagnosis_2449          | -0.0216  |
|    ChronicCond_IschemicHeart     | -0.0215  |
|          diagnosis_4011          | -0.01759 |
|   ClmAdmitDiagnosisCode_count    | -0.01654 |
|      DiagnosisCode_9_count       | -0.01107 |
|      DiagnosisCode_6_count       | -0.01034 |
|      DiagnosisCode_2_count       | -0.00858 |
|          num_of_chronic          |  -0.007  |
|      DiagnosisCode_1_count       | -0.00591 |
|      RenalDiseaseIndicator       | -0.00437 |
|         diagnosis_25000          | -0.00213 |
|         diagnosis_42731          | -0.00144 |
|     ChronicCond_Osteoporasis     | -0.00129 |
|    ChronicCond_ObstrPulmonary    | -0.00081 |
|              Gender              | -0.00026 |
|      DiagnosisCode_3_count       |  -7e-05  |
|         num_of_diag_proc         |   0.0    |
| ChronicCond_rheumatoidarthritis  | 0.00054  |
|         diagnosis_V5861          | 0.00089  |
|            num_of_phy            | 0.00165  |
|     IPAnnualReimbursementAmt     | 0.00185  |
|      IPAnnualReimbursementAmt     | 0.00185  |
|      ChronicCond_Heartfailure     | 0.00299  |
|       DiagnosisCode_4_count       | 0.00301  |
|     ChronicCond_KidneyDisease     | 0.00613  |
|       DiagnosisCode_5_count       | 0.01454  |
|           diagnosis_4280          |  0.0155  |
|       DiagnosisCode_7_count       | 0.01963  |
|         ChronicCond_stroke        | 0.02528  |
|           diagnosis_2724          | 0.02655  |
|         numOfDaysForClaim         | 0.03819  |
|           procedure_66.0          | 0.04979  |
|                Race               | 0.06021  |
|          procedure_2724.0         | 0.06814  |
|          procedure_8154.0         | 0.08536  |
|         numOfDaysAdmitted         |  0.1611  |
|       InscClaimAmtReimbursed      | 0.22066  |
|               County              | 0.31805  |
|      AttendingPhysicianCount      | 0.38372  |
|             BeneCount             | 0.39401  |
|         DeductibleAmtPaid         | 0.51946  |
|           ProviderCount           | 15.8925  |
+-----------------------------------+----------+
```

6.2. DecisionTreeClassifier

In [130]:

```
dt = DecisionTreeClassifier(class_weight='balanced')
param = {'max_depth': [5,10,50,100,250,500,1000],
         'min_samples_split': [5,10,50,100,200,300,400],
         'criterion': ['gini','entropy']}

dt_tune = RandomizedSearchCV(dt,param,cv=10,n_jobs=-1,verbose=1)
dt_tune.fit(X_train,Y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   46.2s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  2.3min finished
```

Out[130]:

```
RandomizedSearchCV(cv=10, error_score='raise-deprecating',
          estimator=DecisionTreeClassifier(class_weight='balanced', 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=False, random_state=None,
            splitter='best'),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'criterion': ['gini', 'entropy'], 'max_depth': [5, 10, 50, 100,
250, 500, 1000], 'min_samples_split': [5, 10, 50, 100, 200, 300, 400]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=1)
```

6.2.1. model with null filled with 0

In [85]:

```
print('best parameter :',dt_tune.best_params_)
print("accuracy on train :",dt_tune.best_score_)
```

```
best parameter : {'criterion': 'gini', 'max_depth': 50, 'min_samples_split': 10}
accuracy on train : 0.975278140968
```

In [86]:

```
## on best parameter
```

```
dt_tuned = DecisionTreeClassifier(max_depth=50,min_samples_split=10,criterion='gini')
dt_tuned.fit(X_train,Y_train)
print('accuracy on test :',dt_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.976792790837
```

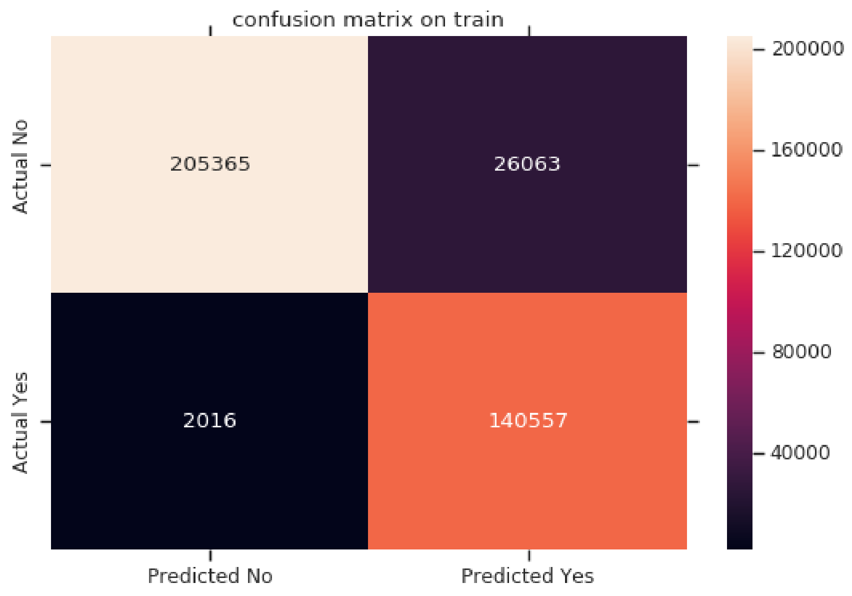
In [74]:

```
dt_tuned = joblib.load('dt_tuned_on_0.pkl')
```

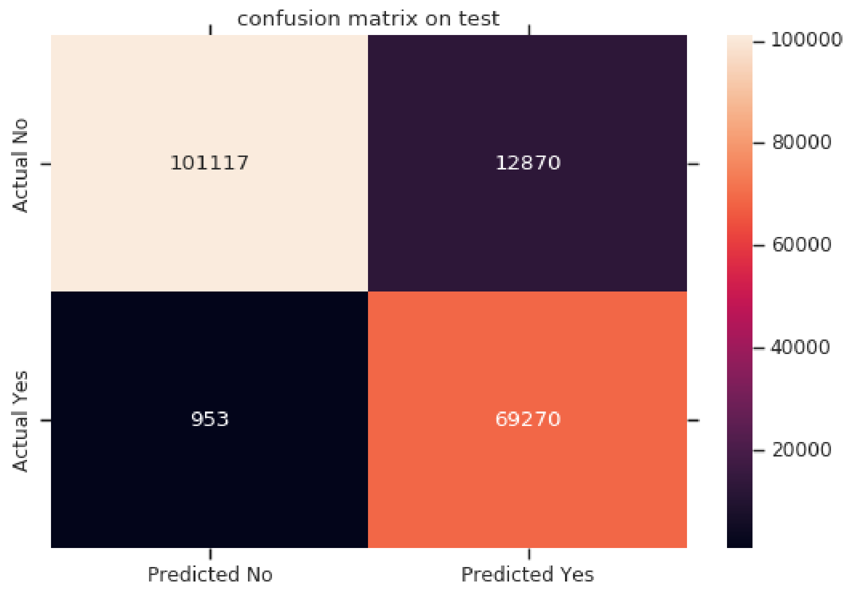
In [75]:

```
print(plot_confusion_matrix(dt_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.5
```



```
best test threshold : 0.428571428571
```



```
None
```

In [89]:

```
## f1 score

f1_train,f1_test,macro_f1_train,macro_f1_test = f1(dt_tuned,X_train,X_test,Y_train,Y_test)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
```

```
Train F1 score is : 0.991839362826
Test F1 score is : 0.969521107079
Train macro F1 score is : 0.993397182279
Test macro F1 score is : 0.975392086812
```

In [91]:

```
joblib.dump(dt_tuned, 'dt_tuned_on_0.pkl')
```

Out[91]:

```
['dt_tuned_on_0.pkl']
```

In [90]:

```
print(important_feature(dt_tuned,X))
```

```
important features :
+-----------------------------+---------+
|           feature           |  weight |
+-----------------------------+---------+
|        ProviderCount        |  0.7119 |
|            State            | 0.09898 |
|   AttendingPhysicianCount   | 0.06816 |
|            County           | 0.05745 |
|   DiagnosisGroupCode_count  | 0.03973 |
|    DiagnosisCode_1_count    | 0.00194 |
|    OPAnnualDeductibleAmt    | 0.00189 |
|      ip_op_total_amount     | 0.00185 |
|   OPAnnualReimbursementAmt  | 0.00168 |
|    InscClaimAmtReimbursed   | 0.00158 |
|          BeneCount          | 0.00125 |
|        num_of_chronic       | 0.00106 |
|    DiagnosisCode_2_count    | 0.00101 |
| ClmAdmitDiagnosisCode_count | 0.00099 |
|    DiagnosisCode_3_count    | 0.00092 |
|   IPAnnualReimbursementAmt  | 0.00086 |
|    DiagnosisCode_4_count    | 0.00079 |
|             Race            | 0.00058 |
|          num_of_phy         | 0.00057 |
|    DiagnosisCode_5_count    | 0.00051 |
+-----------------------------+---------+
```

6.2.2. model with null filled with mode

In [131]:

```
print('best parameter :',dt_tune.best_params_)
print("accuracy on train :",dt_tune.best_score_)
```

```
best parameter : {'criterion': 'entropy', 'max_depth': 50, 'min_samples_split': 5}
accuracy on train : 0.977716637121
```

In [132]:

```
## on best parameter

dt_tuned = DecisionTreeClassifier(max_depth= 50,min_samples_split= 5,criterion= 'entropy')
dt_tuned.fit(X_train,Y_train)
print('accuracy on test :',dt_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.980533087237
```

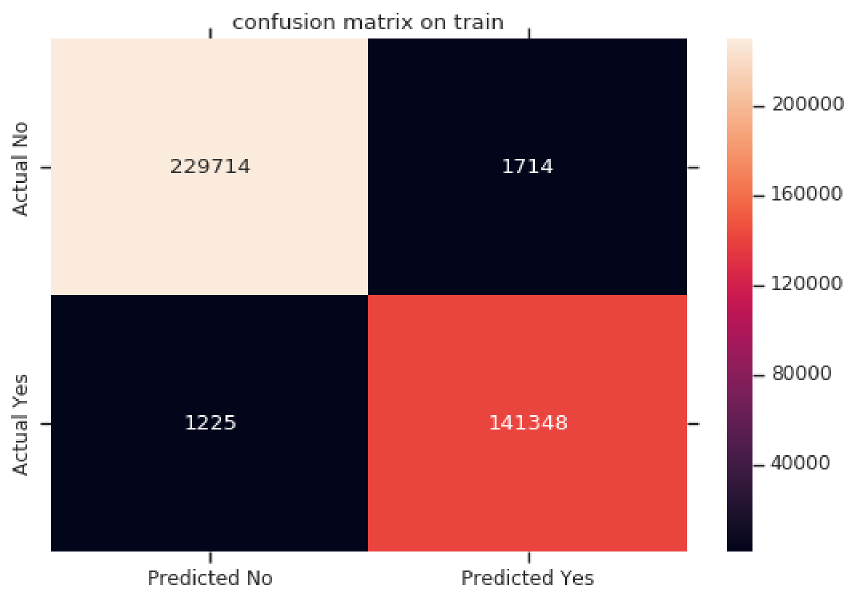
In [97]:

```
dt_tuned = joblib.load('dt_tuned_on_mode.pkl')
```

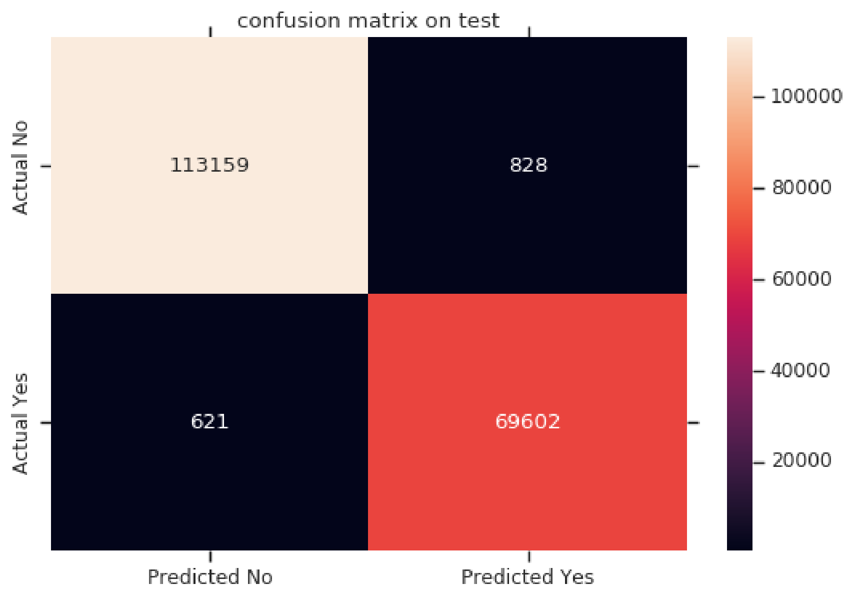
In [98]:

```
## confusion matrix

print(plot_confusion_matrix(dt_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.5
```



```
best test threshold : 0.5
```



```
None
```

In [134]:

```
joblib.dump(dt_tuned, 'dt_tuned_on_mode.pkl')
```

Out[134]:

```
['dt_tuned_on_mode.pkl']
```

In [135]:

```
## f1 score

f1_train,f1_test,macro_f1_train,macro_f1_test = f1(dt_tuned,X_train,X_test,Y_train,Y_test)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)

## important features

print(important_feature(dt_tuned,X))
```

```
Train F1 score is : 0.997838447606
Test F1 score is : 0.974476505004
Train macro F1 score is : 0.998252281856
Test macro F1 score is : 0.979371526166
important features :
+------------------------------+---------+
|           feature            |  weight |
+------------------------------+---------+
|        ProviderCount         | 0.68205 |
|            State             | 0.12157 |
|   AttendingPhysicianCount    | 0.08067 |
|            County            | 0.04307 |
|   DiagnosisGroupCode_count   | 0.03786 |
|   OPAnnualReimbursementAmt   | 0.00323 |
|    OPAnnualDeductibleAmt     | 0.00305 |
|    DiagnosisCode_1_count     | 0.00304 |
|      ip_op_total_amount      | 0.00268 |
|    InscClaimAmtReimbursed    | 0.00229 |
|          BeneCount           | 0.00197 |
|    DiagnosisCode_2_count     | 0.00173 |
|        num_of_chronic        |  0.0016 |
|    DiagnosisCode_3_count     | 0.00132 |
|    DiagnosisCode_4_count     | 0.00117 |
| ClmAdmitDiagnosisCode_count  | 0.00115 |
|   IPAnnualReimbursementAmt   | 0.00078 |
|    DiagnosisCode_5_count     | 0.00072 |
|    DiagnosisCode_6_count     | 0.00064 |
|      numOfDaysForClaim       |  0.0006 |
+------------------------------+---------+
```

6.3. RandomForestClassifier

In [136]:

```
rf = RandomForestClassifier(class_weight='balanced')
param = { 'n_estimators': [10,50,100,500,1000], 'max_depth' : [2,4,6,8,10,12],'min_samples_split':
[5,10,50,100,200,250],'criterion' :['gini', 'entropy']}

rf_tune = RandomizedSearchCV(rf,param,cv=5,n_jobs=-1,verbose=1)
rf_tune.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 20.7min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 28.3min finished
```

Out[136]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
          estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            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,
            n_estimators='warn', n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'criterion': ['gini', 'entropy'], 'n_estimators': [10, 50, 100, 500,
1000], 'max_depth': [2, 4, 6, 8, 10, 12], 'min_samples_split': [5, 10, 50, 100, 200, 250]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=1)
```

6.3.1. model with null filled with 0

In [93]:

```
print('best parameter :',rf_tune.best_params_)
print("accuracy on train :",rf_tune.best_score_)
```

```
best parameter : {'criterion': 'entropy', 'n_estimators': 50, 'max_depth': 12,
'min_samples_split': 10}
accuracy on train : 0.854636752308
```

In [94]:

```
## on best parameter

rf_tuned =
RandomForestClassifier(max_depth=12,min_samples_split=10,criterion='entropy',n_estimators=50,n_job
s=-1)
rf_tuned.fit(X_train,Y_train)
print('accuracy on test :',rf_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.847576135932
```

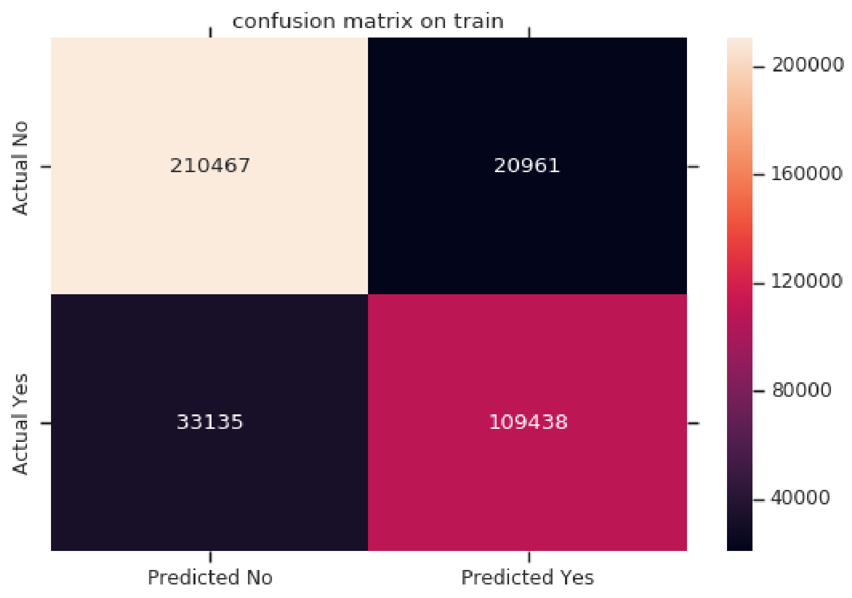
In [76]:

```
rf_tuned = joblib.load('rf_tuned_on_0.pkl')
```

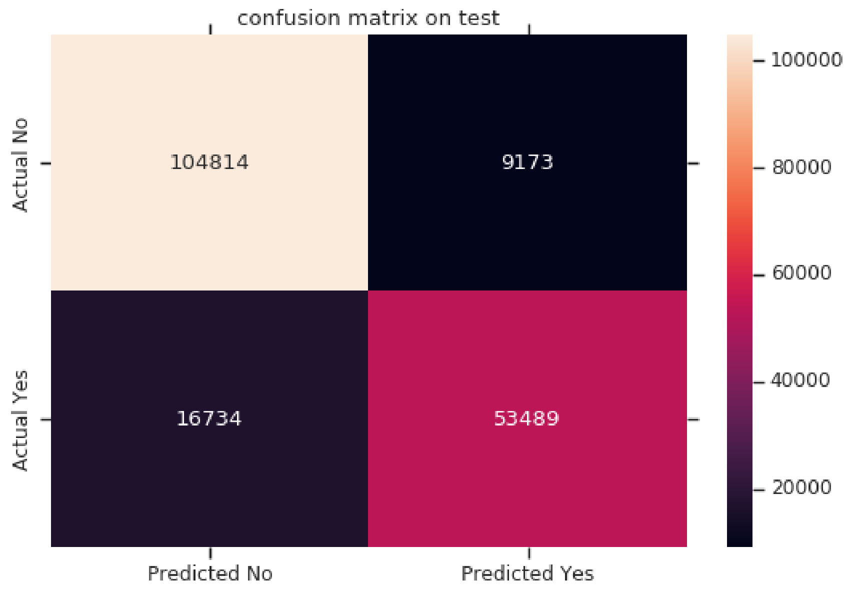
In [77]:

```
print(plot_confusion_matrix(rf_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.318406527274
```



```
best test threshold : 0.321365268658
```



```
None
```

In [96]:

```
f1_train,f1_test,macro_f1_train,macro_f1_test = f1(rf_tuned,X_train,X_test,Y_train,Y_test)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
```

```
Train F1 score is : 0.807312387061
Test F1 score is : 0.799113218315
Train macro F1 score is : 0.846008923538
Test macro F1 score is : 0.840319340769
```

In [97]:

```
joblib.dump(rf_tuned, 'rf_tuned_on_0.pkl')
```

Out[97]:

```
['rf_tuned_on_0.pkl']
```

In [98]:

```
print(important_feature(rf_tuned,X))
```

```
important features :
+------------------------------+---------+
|           feature            |  weight |
+------------------------------+---------+
|        ProviderCount         | 0.75777 |
|   AttendingPhysicianCount    | 0.13083 |
|            State             | 0.02579 |
|            County            | 0.01998 |
|   DiagnosisGroupCode_count   |  0.0084 |
|    InscClaimAmtReimbursed    | 0.00703 |
|       numOfDaysAdmitted      | 0.00624 |
|       DeductibleAmtPaid      | 0.00548 |
|        num_of_diag_proc      | 0.00398 |
|   OPAnnualReimbursementAmt   |  0.0023 |
|      ip_op_total_amount      | 0.00228 |
|          BeneCount           | 0.00218 |
|    OPAnnualDeductibleAmt     | 0.00191 |
|    DiagnosisCode_9_count     | 0.00189 |
|   IPAnnualReimbursementAmt   | 0.00175 |
|    DiagnosisCode_8_count     | 0.00174 |
|    IPAnnualDeductibleAmt     | 0.00171 |
|      numOfDaysForClaim       | 0.00156 |
| ClmAdmitDiagnosisCode_count | 0.00153 |
|    DiagnosisCode_1_count     | 0.00149 |
+-----------------------------+---------+
```

6.3.2. model with null filled with mode

In [137]:

```
print('best parameter :',rf_tune.best_params_)
print("accuracy on train :",rf_tune.best_score_)
```

```
best parameter : {'criterion': 'gini', 'n_estimators': 500, 'max_depth': 10, 'min_samples_split':
5}
accuracy on train : 0.852487025436
```

In [138]:

```
## on best parameter

rf_tuned = RandomForestClassifier(max_depth=10,min_samples_split=5,criterion='gini',n_estimators=5
00,n_jobs=-1)
rf_tuned.fit(X_train,Y_train)
print('accuracy on test :',rf_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.847994137126
```

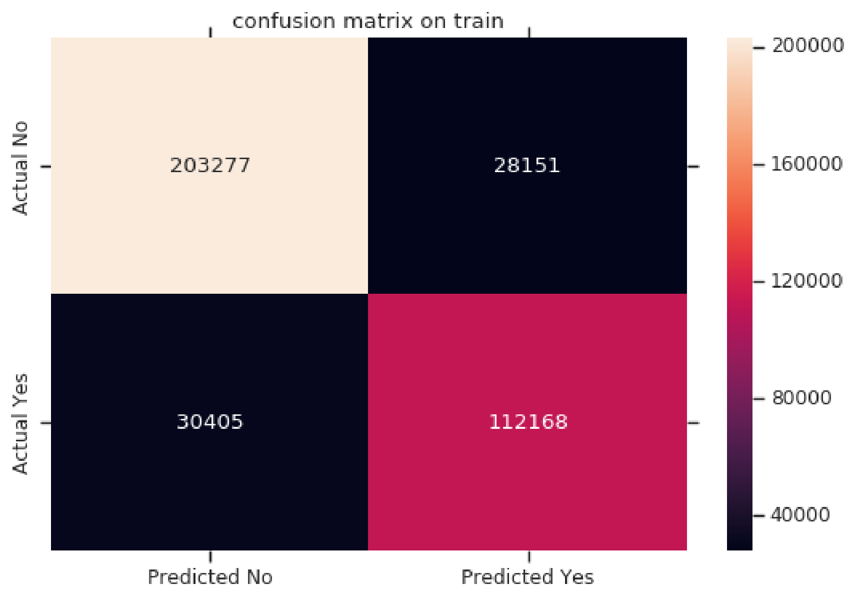
In [99]:

```
rf_tuned = joblib.load('rf_tuned_on_mode.pkl')
```

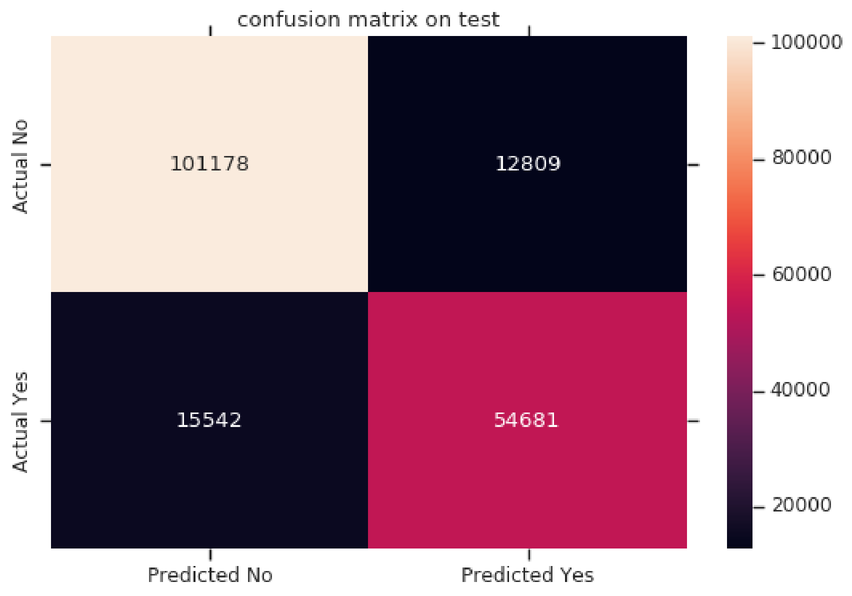
In [100]:

```
print(plot_confusion_matrix(rf_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.279080256525
```



```
best test threshold : 0.281760510223
```



```
None
```

In [140]:

```
joblib.dump(rf_tuned, 'rf_tuned_on_mode.pkl')
```

Out[140]:

```
['rf_tuned_on_mode.pkl']
```

In [141]:

```
f1_train,f1_test,macro_f1_train,macro_f1_test = f1(rf_tuned,X_train,X_test,Y_train,Y_test)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)

print(important_feature(rf_tuned,X))
```

```
Train F1 score is : 0.795408517328
Test F1 score is : 0.788209669561
Train macro F1 score is : 0.835646462347
Test macro F1 score is : 0.829520481848
important features :
+-----------------------------+---------+
|           feature           |  weight |
+-----------------------------+---------+
|        ProviderCount        | 0.72332 |
|    AttendingPhysicianCount  | 0.15823 |
|            State            | 0.02878 |
|            County           | 0.02106 |
|    DiagnosisGroupCode_count | 0.01212 |
|       DeductibleAmtPaid     | 0.01002 |
|       numOfDaysAdmitted     | 0.00921 |
|     InscClaimAmtReimbursed  |  0.0061 |
|        procedure_9904.0     |  0.0034 |
|     DiagnosisCode_9_count   |  0.002  |
|          BeneCount          | 0.00179 |
|     DiagnosisCode_8_count   | 0.00175 |
|     DiagnosisCode_7_count   | 0.00169 |
| ClmAdmitDiagnosisCode_count | 0.00166 |
|       ip_op_total_amount    |  0.0016 |
|    IPAnnualReimbursementAmt |  0.0015 |
|       numOfDaysForClaim     | 0.00146 |
|    OPAnnualReimbursementAmt | 0.00135 |
|     OPAnnualDeductibleAmt   | 0.00133 |
|     DiagnosisCode_6_count   | 0.00132 |
+------------------------------+---------+
```

6.4. XGBClassifier

In [142]:

```
xgb = XGBClassifier()
param = {'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],'n_estimators':[100,200,500,1000],'max_depth
':[3,5,10],'colsample_bytree':[0.1,0.3,0.5,1],'subsample':[0.1,0.3,0.5,1] }

xgb_tune = RandomizedSearchCV(xgb,param,cv=5,n_jobs=-1,verbose=10)

xgb_tune.fit(X_train,Y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  6.1min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  9.4min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 16.9min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 33.4min remaining:  7.3min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 42.6min remaining:  2.7min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 52.7min finished
```

Out[142]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators':
[100, 200, 500, 1000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1],
'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

6.4.1. model with null filled with 0

In [106]:

```
print('best parameter :',xgb_tune.best_params_)
print("accuracy on train :",xgb_tune.best_score_)
```

```
best parameter : {'learning_rate': 0.2, 'n_estimators': 1000, 'max_depth': 10, 'colsample_bytree':
0.5, 'subsample': 0.5}
accuracy on train : 0.978615030441
```

In [101]:

```
## on best parameter

xgb_tuned = XGBClassifier(subsample=0.5,n_estimators= 1000,max_depth=
10,colsample_bytree=0.5,learning_rate= 0.2,n_jobs=-1)
xgb_tuned.fit(X_train,Y_train)
print('accuracy on test :',xgb_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.981797947994
```

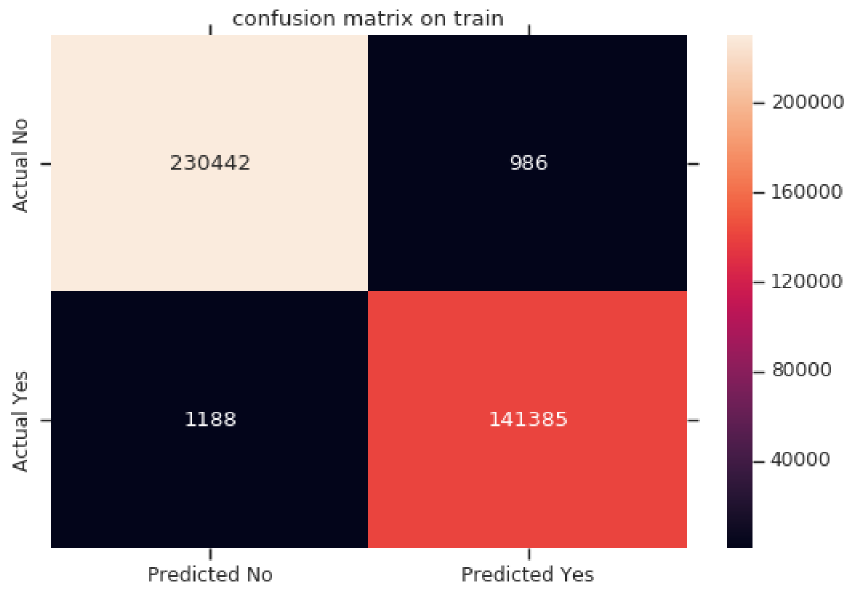
In [78]:

```
xgb_tuned = joblib.load('xgb_tuned_on_0.pkl')
```

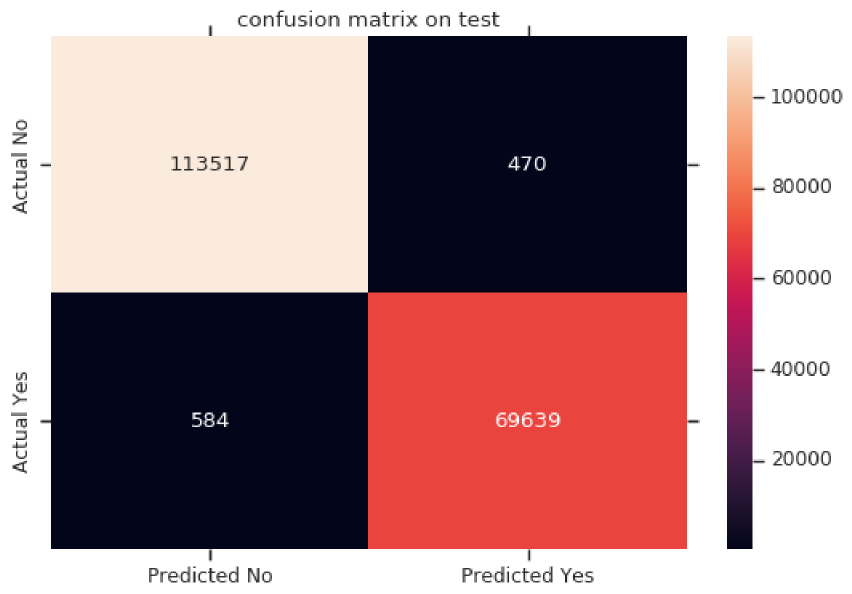
In [79]:

```
print(plot_confusion_matrix(xgb_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.211525
```



```
best test threshold : 0.223045
```



```
None
```

In [104]:

```
joblib.dump(xgb_tuned, 'xgb_tuned_on_0.pkl')
```

Out[104]:

```
['xgb_tuned_on_0.pkl']
```

In [105]:

```
## f1 score

f1_train,f1_test,macro_f1_train,macro_f1_test = f1(xgb_tuned,X_train,X_test,Y_train,Y_test)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)

print(important_feature(xgb_tuned,X))
```

```
Train F1 score is : 1.0
Test F1 score is : 0.977490353008
Train macro F1 score is : 1.0
Test macro F1 score is : 0.981851794151
important features :
+-----------------------------+---------+
|           feature           |  weight |
+-----------------------------+---------+
|        ProviderCount        | 0.11823 |
|            County           | 0.07264 |
|   OPAnnualReimbursementAmt  | 0.06902 |
|   AttendingPhysicianCount   | 0.06714 |
|     DiagnosisCode_1_count   | 0.06436 |
|            State            |  0.0629 |
|     OPAnnualDeductibleAmt   | 0.06165 |
|      ip_op_total_amount     | 0.06164 |
|    InscClaimAmtReimbursed   | 0.04316 |
|     DiagnosisCode_2_count   | 0.04158 |
|     DiagnosisCode_3_count   | 0.03246 |
|          BeneCount          | 0.02967 |
|        num_of_chronic       | 0.02331 |
|   IPAnnualReimbursementAmt  | 0.02187 |
| ClmAdmitDiagnosisCode_count | 0.02084 |
|     DiagnosisCode_4_count   | 0.02018 |
|       num_of_diag_proc      |  0.0171 |
|     DiagnosisCode_5_count   | 0.01254 |
|     DiagnosisCode_6_count   |  0.0108 |
|          num_of_phy         | 0.01079 |
+-----------------------------+---------+
```

6.4.2. model with null filled with mode

In [143]:

```
print('best parameter :',xgb_tune.best_params_)
print("accuracy on train :",xgb_tune.best_score_)
```

```
best parameter : {'learning_rate': 0.2, 'n_estimators': 1000, 'max_depth': 10, 'colsample_bytree':
0.5, 'subsample': 0.3}
accuracy on train : 0.972853013762
```

In [144]:

```
## on best parameter

xgb_tuned = XGBClassifier(subsample=0.3,n_estimators= 1000,max_depth=
10,colsample_bytree=0.5,learning_rate= 0.2,n_jobs=-1)
xgb_tuned.fit(X_train,Y_train)
print('accuracy on test :',xgb_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.97632593236
```

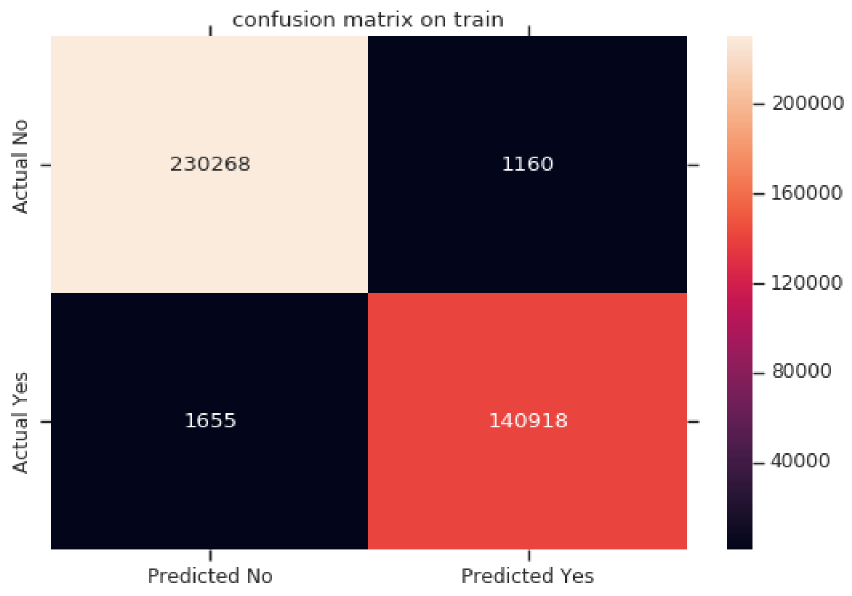
In [101]:

```
xgb_tuned = joblib.load('xgb_tuned_on_mode.pkl')
```

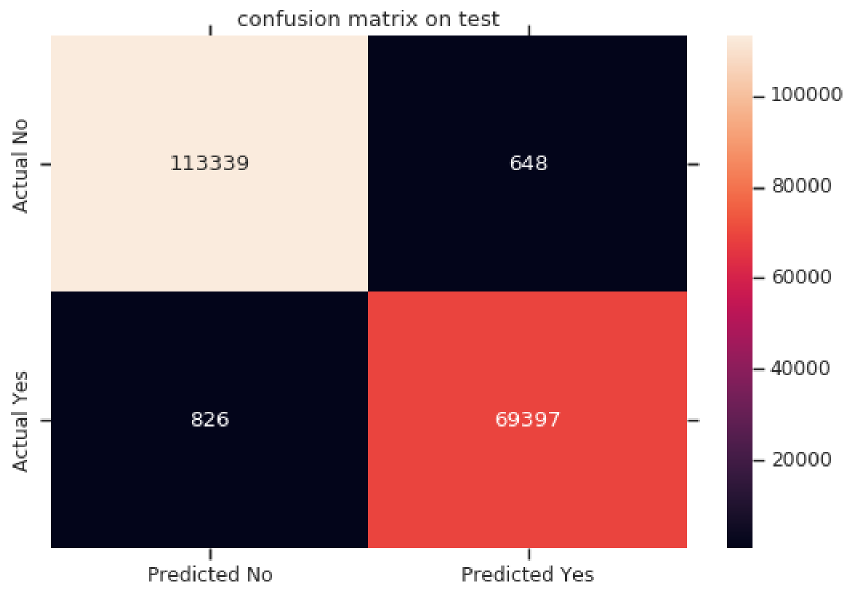
In [102]:

```
print(plot_confusion_matrix(xgb_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.263377
```



```
best test threshold : 0.247369
```



```
None
```

In [146]:

```
joblib.dump(xgb_tuned, 'xgb_tuned_on_mode.pkl')
```

Out[146]:

```
['xgb_tuned_on_mode.pkl']
```

In [147]:

```
f1_train,f1_test,macro_f1_train,macro_f1_test = f1(xgb_tuned,X_train,X_test,Y_train,Y_test)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
print(important_feature(xgb_tuned,X))
```

```
Train F1 score is : 1.0
Test F1 score is : 0.969402872909
Train macro F1 score is : 1.0
Test macro F1 score is : 0.975355172118
important features :
+------------------------------+---------+
|           feature            |  weight |
+------------------------------+---------+
|        ProviderCount         | 0.11673 |
|   OPAnnualReimbursementAmt   | 0.07042 |
|            County            | 0.06854 |
|    DiagnosisCode_1_count     | 0.06611 |
|   AttendingPhysicianCount    |  0.0645 |
|    OPAnnualDeductibleAmt     | 0.06373 |
|      ip_op_total_amount      | 0.06189 |
|            State             | 0.05939 |
|    InscClaimAmtReimbursed    | 0.04685 |
|    DiagnosisCode_2_count     | 0.04484 |
|    DiagnosisCode_3_count     |  0.0333 |
|          BeneCount           | 0.03149 |
|        num_of_chronic        | 0.02558 |
|   IPAnnualReimbursementAmt   | 0.02258 |
| ClmAdmitDiagnosisCode_count  | 0.02245 |
|    DiagnosisCode_4_count     |  0.0221 |
|    DiagnosisCode_5_count     | 0.01384 |
|    DiagnosisCode_6_count     | 0.01109 |
|          num_of_phy          | 0.01066 |
|    DiagnosisCode_7_count     | 0.00863 |
+------------------------------+---------+
```

Custom Ensemble model

procedure followed

```
1. split data into 80:20 train,test
2. split train into D1, D2 50:50
3. do sampling with replacement on D1 and take k samples
4. train k model on this k sample ( DT as base model )
5. use D2 and pass it to this k model and get k prediction
6. here we have actual y of D2 (from step 2) and target of D2 (from step 5)
7. train meta model with data of step 6
8. for evaluation take test data (from step 1)
9. take test and predict on k model (step 4) and get k prediction
10. here we have actual y of test (from step 1) and target of test (from step 9)
11. as performance evaluation, train on (pred_for_D2,D2Y) and fit on (pred_for_test,test_Y)

num of base model trained is 30,45,40. 40 seem to have good accuracy
```

different meta model trained:

```
1.LR
2.DT
3.RF
4.xgb
```

In [159]:

```
## file with null filled with 0

final_data = pd.read_csv('final_data_feature.csv')
final_data.drop('Unnamed: 0',axis=1,inplace=True)
```

In [160]:

```
## split data into 80:20

from sklearn.model_selection import train_test_split
train, test = train_test_split(final_data, test_size=0.20,random_state=0)
```

In [146]:

```
min_max_scaler = preprocessing.MinMaxScaler()

def min_max(D1,D2,test,column):
    """ scaling column value using min max scalar, fitting on train and transforming """

    min_max_scaler.fit(D1[column].values.reshape(-1,1))

    d1_scale = min_max_scaler.transform(D1[column].values.reshape(-1,1))
    d2_scale = min_max_scaler.transform(D2[column].values.reshape(-1,1))
    test_scale = min_max_scaler.transform(test[column].values.reshape(-1,1))

    return d1_scale,d2_scale,test_scale
```

In [147]:

```
import random
def generating_samples(input_data, target_data):
    """ row and col sampling with replacement """

    ## selecting 60 % row from input data
    selecting_rows = np.sort(np.random.choice(input_data.shape[0],size=int((60/100)*input_data.shap
e[0]),replace=False))

    ## selecting 3 - 58 col from input data
    selecting_colums = np.sort(np.random.choice(input_data.shape[1],size=random.randint(3,input_dat
a.shape[1]),replace =False))

    ## sampling row from input
    sample_data = input_data[selecting_rows[:,None],selecting_colums]

    ## sampling target row from input
    target_of_sample_data = target_data[selecting_rows]

    ## replicating from sample data
    replacing_rows =np.sort(np.random.choice(selecting_rows,size=int(input_data.shape[0]*0.4),repla
ce=False))

    ## re sampling row from sample data
    replicated_sample_data = input_data[replacing_rows[:, None], selecting_colums]

    ## re sampling target row from sample data
    target_of_replicated_sample_data = target_data[replacing_rows]

    final_sample_data = np.vstack((sample_data, replicated_sample_data))
    final_target_data =
np.vstack((target_of_sample_data.reshape(-1,1),target_of_replicated_sample_data.reshape(-1,1)))

    return final_sample_data,final_target_data,selecting_rows,selecting_colums
```

In [148]:

```
def custom_ensemble(train, test, n_estimators):
    """ take train, test, num of base model and divide train into D1,D2 ,
    train base model with D1, predict on D2, take test and predict using base model"""

    ## split train into D1,D2
    D1,D2 = train_test_split(train, test_size=0.50,random_state=0)


    ## col to normalize
    col_to_nor = ['InscClaimAmtReimbursed', 'DeductibleAmtPaid', 'Race',
       'State', 'County', 'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
       'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt', 'BeneCount',
       'ProviderCount', 'AttendingPhysicianCount', 'numOfDaysAdmitted',
       'numOfDaysForClaim', 'ip_op_total_amount', 'num_of_chronic',
       'num_of_diag_proc', 'num_of_phy', 'DiagnosisCode_1_count', 'DiagnosisCode_2_count',
       'DiagnosisCode_3_count', 'DiagnosisCode_4_count',
        'DiagnosisCode_5_count', 'DiagnosisCode_6_count',
        'DiagnosisCode_7_count', 'DiagnosisCode_8_count',
        'DiagnosisCode_9_count', 'DiagnosisCode_10_count',
        'ClmAdmitDiagnosisCode_count', 'DiagnosisGroupCode_count']

    for i in col_to_nor:
        d1_scale,d2_scale,test_scale = min_max(D1,D2,test,i)

        D1[i] = d1_scale
        D2[i] = d2_scale
        test[i] = test_scale
    print('nomalizing done...')


    ## split D1 to x,y
    Y = D1['PotentialFraud'].values
    X = D1.drop('PotentialFraud',axis=1).values


    ## generating n_estimators sample
    list_input_data =[]
    list_output_data =[]
    list_selected_row= []
    list_selected_columns=[]

    for i in range(0,n_estimators):
        data,target,selecting_rows,selecting_columns = generating_samples(X,Y)
        list_input_data.append(data)
        list_output_data.append(target)
        list_selected_row.append(selecting_rows)
        list_selected_columns.append(selecting_columns)


    ## train n_estimators model on this n_estimators sample
    ## using best param from Dt trained with null filled with 0
    list_of_all_models = []
    for i in range(0, n_estimators):
        input_data = list_input_data[i]
        target_data = list_output_data[i]
        classifier = DecisionTreeClassifier(class_weight='balanced',max_depth=50,min_samples_split=
10,criterion='gini',random_state=0)
        classifier.fit(input_data,target_data)
        list_of_all_models.append(classifier)
    print(n_estimators,'base model trained...')


    ## split D1 to x,y
    D2Y = D2['PotentialFraud'].values
    D2X=  D2.drop('PotentialFraud', axis=1).values


    ## passing the D2 set to each of these n_estimators models, and getting n_estimators set of pr
edictions for D2, from each of these models
    pred_for_D2 = []
    for i in range(0,n_estimators):
        pred_y = list_of_all_models[i].predict(D2X[:,list_selected_columns[i]])
        pred_for_D2.append(pred_y)


    ## shape of prediction on D2 is (30, ), so transpose to get ( ,30)
    pred_for_D2 = np.array(pred_for_D2).transpose()


    ## fitting LR on : prediction of D2 as X, and actual D2Y as Y

    param = { 'C': [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000], 'penalty' : ['l1','l2'] }
    lr = LogisticRegression()
    lr_tune = RandomizedSearchCV(lr,param,cv=10,n_jobs=-1)
    lr_tune.fit(pred_for_D2,D2Y)

    print('best parameter :',lr_tune.best_params_)
    print("accuracy on train :",lr_tune.best_score_)


    ## getting prediction on test using base models

    test_Y = test['PotentialFraud'].values
    test_X=  test.drop('PotentialFraud', axis=1).values

    pred_for_test = []
    for i in range(0,n_estimators):
        pred_y = list_of_all_models[i].predict(test_X[:,list_selected_columns[i]])
        pred_for_test.append(pred_y)

    pred_for_test = np.array(pred_for_test).transpose()
    print('prediction on test done...')

    return pred_for_test,pred_for_D2,D2Y,test_Y
```

with 30 base model

In [149]:

```
## return D2 actual y, pred on D2, actual y test, pred on y test

pred_for_test,pred_for_D2,D2Y,test_Y = custom_ensemble(train, test, 30)
```

```
nomalizing done...
30 base model trained...
best parameter : {'penalty': 'l1', 'C': 100}
accuracy on train : 0.982116945236
prediction on test done...
```

In [150]:

```
## fitting LR on : prediction of D2 as X, and actual D2Y as Y
## testing on : prediction on test using base models as X, and actual test_Y as Y

lr_tuned = LogisticRegression(C=100,penalty='l1')
lr_tuned.fit(pred_for_D2,D2Y)
print('accuracy on test :',lr_tuned.score(pred_for_test,test_Y))
```

```
accuracy on test : 0.982739625413
```

with 45 base model

In [153]:

```
pred_for_test,pred_for_D2,D2Y,test_Y = custom_ensemble(train, test, 45)
```

```
nomalizing done...
45 base model trained...
best parameter : {'penalty': 'l1', 'C': 1000}
accuracy on train : 0.983617276652
prediction on test done...
```

In [154]:

```
## fitting LR on : prediction of D2 as X, and actual D2Y as Y
## testing on : prediction on test using base models as X, and actual test_Y as Y

lr_tuned = LogisticRegression(C=1000,penalty='l1')
lr_tuned.fit(pred_for_D2,D2Y)
print('accuracy on test :',lr_tuned.score(pred_for_test,test_Y))
```

```
accuracy on test : 0.984342950297
```

with 40 base model

In [161]:

```
pred_for_test,pred_for_D2,D2Y,test_Y = custom_ensemble(train, test, 40)
```

```
nomalizing done...
40 base model trained...
best parameter : {'penalty': 'l2', 'C': 0.1}
accuracy on train : 0.983312731768
prediction on test done...
```

In [162]:

```
## fitting LR on : prediction of D2 as X, and actual D2Y as Y
## testing on : prediction on test using base models as X, and actual test_Y as Y

lr_tuned = LogisticRegression(C=0.1,penalty='l2')
lr_tuned.fit(pred_for_D2,D2Y)
print('accuracy on test :',lr_tuned.score(pred_for_test,test_Y))
```

```
accuracy on test : 0.983393495338
```

accuracy on base model

```
accuracy on base model 30 : 0.982739625413
accuracy on base model 45 : 0.984342950297
accuracy on base model 40 : 0.983393495338
```

conclusion

```
accuracy doesnt increase much, so number of base model selected 40
```

LR as metamodel with 40 base model

In [125]:

```
param = { 'C': [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000], 'penalty' : ['l1','l2'] }
lr = LogisticRegression()
lr_tune = RandomizedSearchCV(lr,param,cv=10,n_jobs=-1)
lr_tune.fit(pred_for_D2,D2Y)

print('best parameter :',lr_tune.best_params_)
print("accuracy on train :",lr_tune.best_score_)
```

```
best parameter : {'penalty': 'l2', 'C': 1}
accuracy on train : 0.982107988033
```

In [126]:

```
lr_tuned = LogisticRegression(C=1,penalty='l2')
lr_tuned.fit(pred_for_D2,D2Y)
print('accuracy on test :',lr_tuned.score(pred_for_test,test_Y))
```

```
accuracy on test : 0.982238026567
```

In [127]:

```
joblib.dump(lr_tuned, 'ensem_lr_tuned_on_0.pkl')
```

Out[127]:

```
['ensem_lr_tuned_on_0.pkl']
```

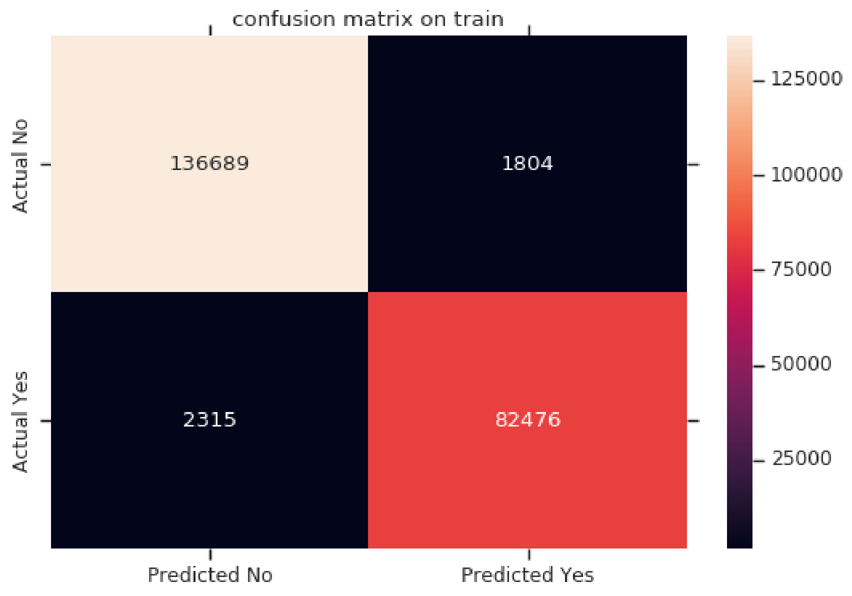
In [116]:

```
lr_tuned = joblib.load('ensem_lr_tuned_on_0.pkl')
```

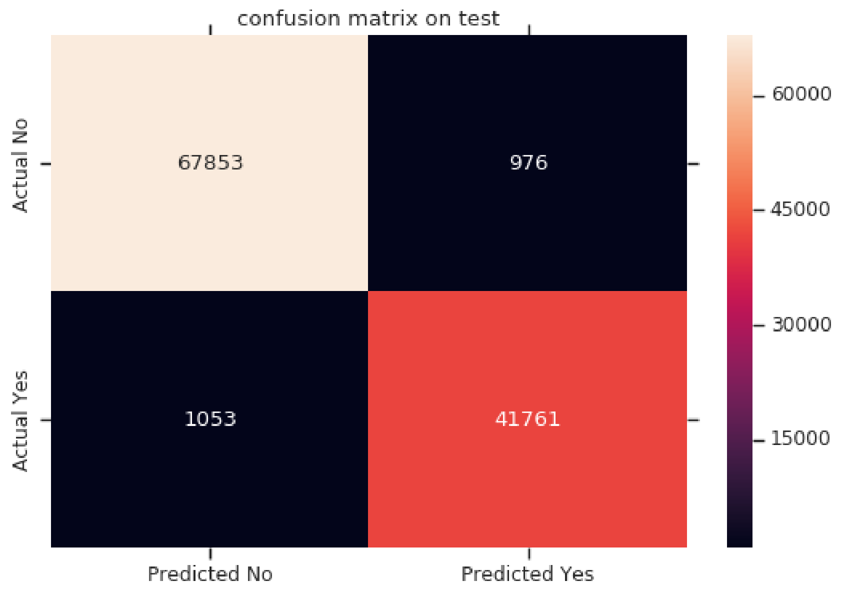
In [128]:

```
print(plot_confusion_matrix(lr_tuned,pred_for_D2,pred_for_test,D2Y,test_Y))

## f1 score

f1_train,f1_test,macro_f1_train,macro_f1_test = f1(lr_tuned,pred_for_D2,pred_for_test,D2Y,test_Y)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
```

```
best train threshold : 0.361948358428
```



```
best test threshold : 0.318580649356
```



```
None
Train F1 score is : 0.975637454087
Test F1 score is : 0.976283152739
Train macro F1 score is : 0.980397025187
Test macro F1 score is : 0.98077598302
```

Decision Tree metamodel

In [129]:

```
dt = DecisionTreeClassifier(class_weight='balanced')
param = {'max_depth': [5,10,50,100,250,500,1000],
         'min_samples_split': [5,10,50,100,200,300,400],
         'criterion': ['gini','entropy']}

dt_tune = RandomizedSearchCV(dt,param,cv=10,n_jobs=-1,verbose=1)
dt_tune.fit(pred_for_D2,D2Y)

print('best parameter :',dt_tune.best_params_)
print("accuracy on train :",dt_tune.best_score_)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   25.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   59.0s finished
```

```
best parameter : {'min_samples_split': 100, 'max_depth': 250, 'criterion': 'gini'}
accuracy on train : 0.980092617474
```

In [130]:

```
dt_tuned = DecisionTreeClassifier(max_depth= 250,criterion= 'gini',min_samples_split= 100)
dt_tuned.fit(pred_for_D2,D2Y)
print('accuracy on test :',dt_tuned.score(pred_for_test,test_Y))
```

```
accuracy on test : 0.980804887006
```

In [131]:

```
joblib.dump(dt_tuned, 'ensem_dt_tuned_on_0.pkl')
```

Out[131]:

```
['ensem_dt_tuned_on_0.pkl']
```

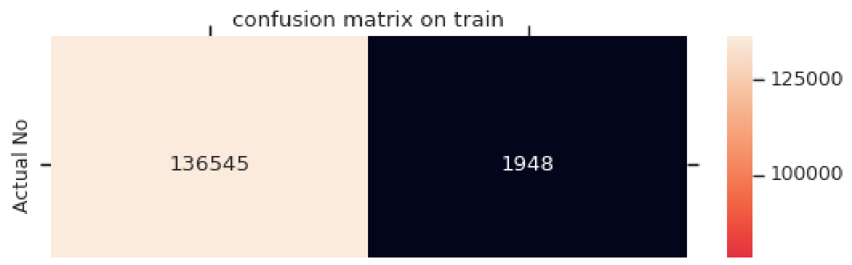
In [112]:

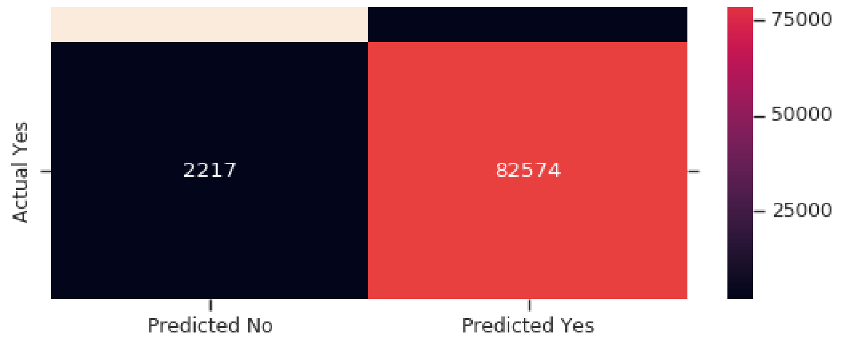
```
dt_tuned = joblib.load('ensem_dt_tuned_on_0.pkl')
```

In [132]:

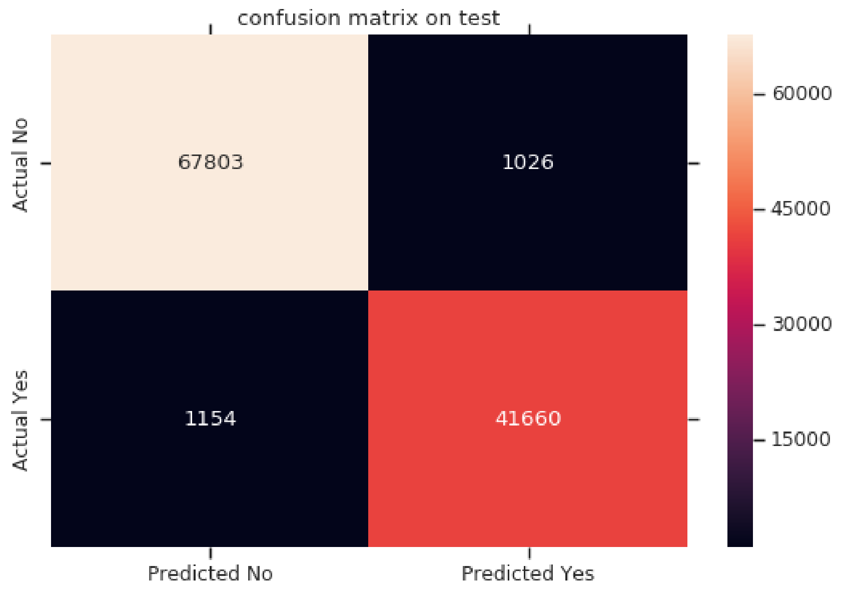
```
print(plot_confusion_matrix(dt_tuned,pred_for_D2,pred_for_test,D2Y,test_Y))

## f1 score

f1_train,f1_test,macro_f1_train,macro_f1_test = f1(dt_tuned,pred_for_D2,pred_for_test,D2Y,test_Y)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
```

```
best train threshold : 0.384615384615
```





```
best test threshold : 0.384615384615
```



```
None
Train F1 score is : 0.975400589441
Test F1 score is : 0.974502923977
Train macro F1 score is : 0.980189158763
Test macro F1 score is : 0.979340643763
```

Random Forest metamodel

In [133]:

```
rf = RandomForestClassifier(class_weight='balanced')
param = { 'n_estimators': [10,50,100,500,1000], 'max_depth' : [2,4,6,8,10,12],'min_samples_split':
[5,10,50,100,200,250],'criterion' :['gini', 'entropy']}

rf_tune = RandomizedSearchCV(rf,param,cv=5,n_jobs=-1,verbose=1)
rf_tune.fit(pred_for_D2,D2Y)

print('best parameter :',rf_tune.best_params_)
print("accuracy on train :",rf_tune.best_score_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  6.1min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  6.6min finished
```

```
best parameter : {'n_estimators': 1000, 'min_samples_split': 5, 'max_depth': 6, 'criterion':
'entropy'}
accuracy on train : 0.981888536572
```

In [134]:

```
rf_tuned = RandomForestClassifier(max_depth= 6,criterion= 'entropy',min_samples_split=
5,n_estimators= 1000)
rf_tuned.fit(pred_for_D2,D2Y)
print('accuracy on test :',rf_tuned.score(pred_for_test,test_Y))
```

```
accuracy on test : 0.981834956065
```

In [135]:

```
joblib.dump(rf_tuned, 'ensem_rf_tuned_on_0.pkl')
```

Out[135]:

```
['ensem_rf_tuned_on_0.pkl']
```

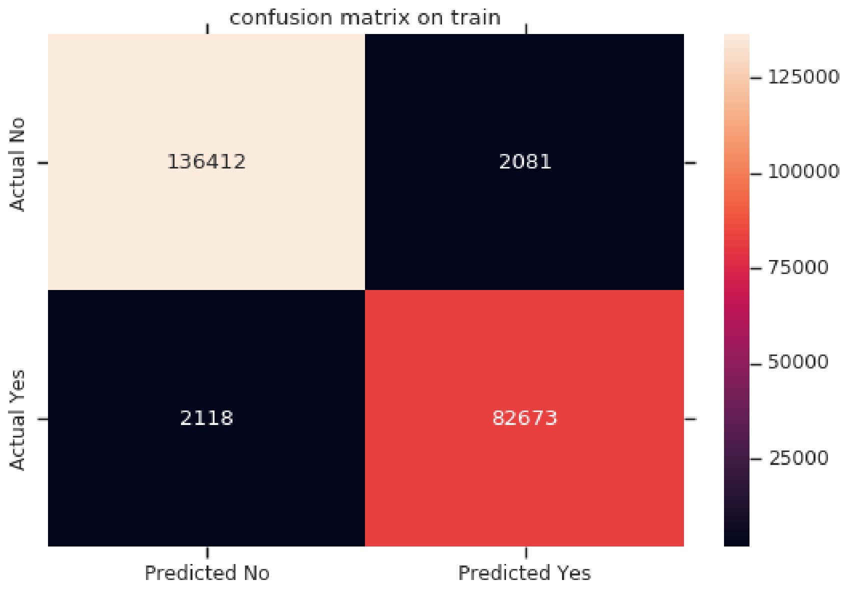
In [114]:

```
rf_tuned = joblib.load('ensem_rf_tuned_on_0.pkl')
```

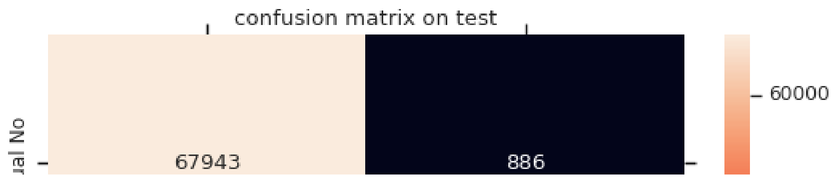
In [136]:

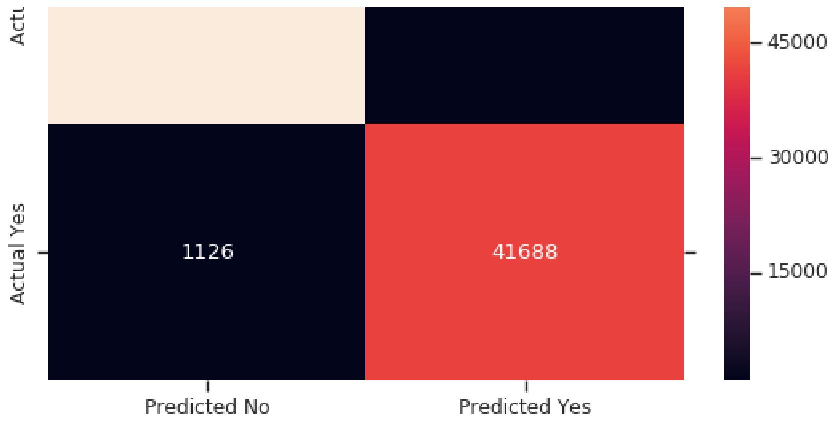
```
print(plot_confusion_matrix(rf_tuned,pred_for_D2,pred_for_test,D2Y,test_Y))

## f1 score

f1_train,f1_test,macro_f1_train,macro_f1_test = f1(rf_tuned,pred_for_D2,pred_for_test,D2Y,test_Y)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
```

```
best train threshold : 0.320728092789
```



```
best test threshold : 0.35351924474
```





```
None
Train F1 score is : 0.975233713763
Test F1 score is : 0.976436970066
Train macro F1 score is : 0.980038063785
Test macro F1 score is : 0.980923237821
```

XGB metamodel

In [137]:

```
xgb = XGBClassifier()
param = {'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],'n_estimators':[100,200,500,1000],'max_depth
':[3,5,10],'colsample_bytree':[0.1,0.3,0.5,1],'subsample':[0.1,0.3,0.5,1] }

xgb_tune = RandomizedSearchCV(xgb,param,cv=5,n_jobs=-1,verbose=10)

xgb_tune.fit(pred_for_D2,D2Y)

print('best parameter :',xgb_tune.best_params_)
print("accuracy on train :",xgb_tune.best_score_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   25.2s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   46.8s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed:  6.0min remaining:  1.3min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed:  6.7min remaining:   25.6s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  7.8min finished
```

```
best parameter : {'learning_rate': 0.05, 'n_estimators': 500, 'colsample_bytree': 1, 'max_depth':
3, 'subsample': 0.5}
accuracy on train : 0.982161731248
```

In [138]:

```
xgb_tuned = XGBClassifier(learning_rate=0.05,n_estimators=500,max_depth=3,colsample_bytree=1,subsam
ple=0.5)
xgb_tuned.fit(pred_for_D2,D2Y)
print('accuracy on test :',xgb_tuned.score(pred_for_test,test_Y))
```

```
accuracy on test : 0.982202198078
```

In [139]:

```
joblib.dump(xgb_tuned, 'ensem_xgb_tuned_on_0.pkl')
```

Out[139]:

```
['ensem_xgb_tuned_on_0.pkl']
```

```
['ensem_xgb_tuned_on_0.pkl']
```

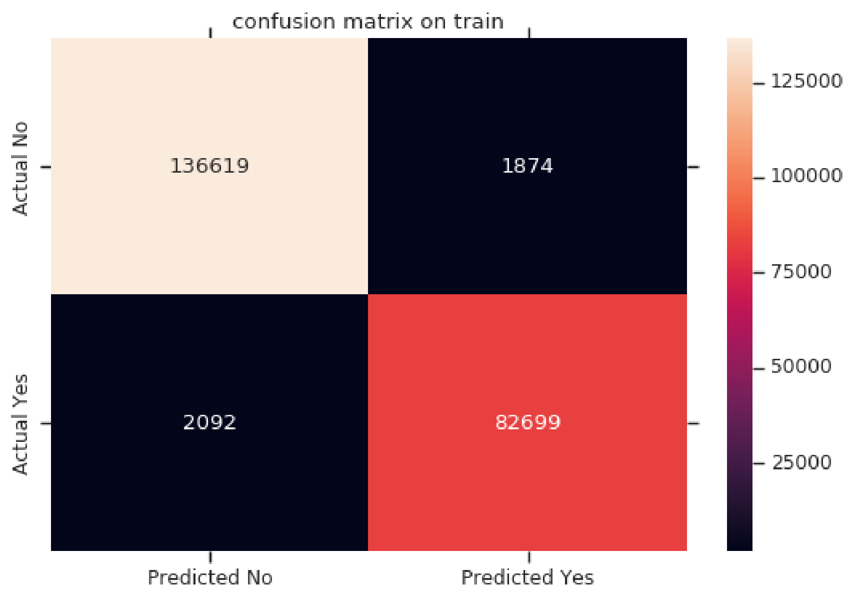
In [76]:

```
xgb_tuned = joblib.load('ensem_xgb_tuned_on_0.pkl')
```

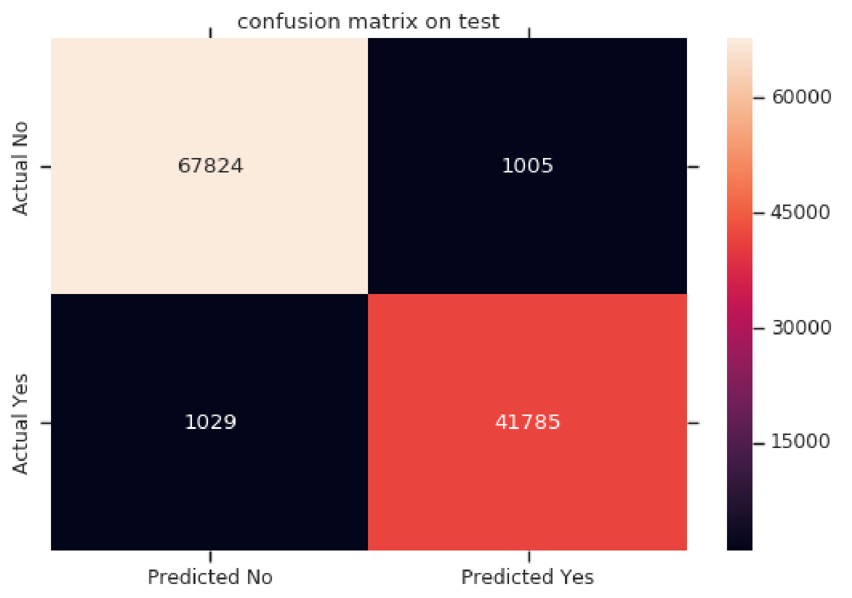
In [140]:

```
print(plot_confusion_matrix(xgb_tuned,pred_for_D2,pred_for_test,D2Y,test_Y))

## f1 score

f1_train,f1_test,macro_f1_train,macro_f1_test = f1(xgb_tuned,pred_for_D2,pred_for_test,D2Y,test_Y)
print('Train F1 score is :',f1_train)
print('Test F1 score is :',f1_test)
print('Train macro F1 score is :',macro_f1_train)
print('Test macro F1 score is :',macro_f1_test)
```

```
best train threshold : 0.347485
```



```
best test threshold : 0.328514
```



```
None
```

```
Train F1 score is : 0.976582981035
Test F1 score is : 0.976239428064
Train macro F1 score is : 0.98113791409
Test macro F1 score is : 0.98073312755
```

conclusion

1. model with null filled with 0

   ```
   xgb, DT have better performance
   ```

2. model with null filled with mode

   ```
   no much increase in performance in all model
   ```

3. custom ensemble model with null filled with 0

   ```
   LR,DT,RF,xgb all metamodel perform good and same.
   ```

   #### overall performance of xgb is better than all other model

summary

made two models on data:

```
1. using null value filled with 0
2. using null value filled with mode
3. custom ensemble model with null value filled with 0
```

column that are performed with imputation are ----- ['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', 'DiagnosisGroupCode','ClmAdmitDiagnosisCode', AttendingPhysicianCount, num_of_phy, DeductibleAmtPaid]

mean (mode is 0) - [numOfDaysAdmitted, numOfDaysForClaim]

1. train test split test size = 0.33
2. feature scaling done with min max scalar on ----- ['InscClaimAmtReimbursed', 'DeductibleAmtPaid', 'Race', 'State', 'County', 'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt', 'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt', 'BeneCount', 'ProviderCount', 'AttendingPhysicianCount', 'numOfDaysAdmitted', 'numOfDaysForClaim', 'ip_op_total_amount', 'num_of_chronic', 'num_of_diag_proc', 'num_of_phy', 'DiagnosisCode_1_count', 'DiagnosisCode_2_count', 'DiagnosisCode_3_count', 'DiagnosisCode_4_count', 'DiagnosisCode_5_count', 'DiagnosisCode_6_count', 'DiagnosisCode_7_count', 'DiagnosisCode_8_count', 'DiagnosisCode_9_count', 'DiagnosisCode_10_count', 'ClmAdmitDiagnosisCode_count', 'DiagnosisGroupCode_count'] done separately on train and test to avoid data leak
3. model trained using missing value filled with 0
   A. LogisticRegression
   B. DecisionTreeClassifier
   C. RandomForestClassifier
   D. XGBClassifier
4. model trained using missing value filled with mode
   A. LogisticRegression
   B. DecisionTreeClassifier
   C. RandomForestClassifier
   D. XGBClassifier
5. hyperparameter tuning using random search
6. high presicion needed, so number of point which is actually fraud being predicted as non fraud decreases. best threshold is found using roc_curve and f1 score is build on this.

model performance with null filled with 0

model	accuracy	f1 score	macro f1 score

LR	0.822420064057	0.743772804213	0.799572764874
DT	0.976792790837	0.969521107079	0.975392086812
RF	0.847576135932	0.799113218315	0.840319340769
xgb	0.981797947994	0.977490353008	0.981851794151

model performance with null filled with mode

model	accuracy	f1 score	macro f1 score
LR	0.83012865751	0.739406238366	0.792575792876
DT	0.980533087237	0.974476505004	0.979371526166
RF	0.847994137126	0.788209669561	0.829520481848
xgb	0.97632593236	0.969402872909	0.975355172118

conclusion

```
1. model with null filled with 0
    xgb, DT have better performance
2. model with null filled with mode
    no much increase in performance in all model
3. ensemble model with null filled with 0
    LR,DT,RF,xgb all metamodel perform good and same.
```

overall performance of xgb is better than all other model

```
    4. feature that help best predicting fraud in all model are - provider count, attending
physician count, county, state, diagnosis code count
```

custom ensemble model with null filled with 0

model	accuracy	f1 score	macro f1 score
LR	0.982238026567	0.976283152739	0.98077598302
DT	0.980804887006	0.974502923977	0.979340643763
RF	0.981834956065	0.976436970066	0.980923237821
xgb	0.982202198078	0.976239428064	0.98073312755

```
In [ ]:
```