

Healthcare Provider Fraud Detection Analysis

In []:

```
this file has eda part, data preprocessing, feature engineering, models and their summary
```

summary

eda

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 occurring 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

Data preprocessing

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)

feature engineering

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 reimburse (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 encoding 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..

model

made two models on data:

1. using null value filled with 0
2. using null value filled with mode

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 recall 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 with missing value filled with 0

model	accuracy	f1 score
LR	0.830921231204	0.745294738076
DT	0.979691656262	0.973989367396
RF	0.84887356821	0.794203993838
xgb	0.98747624	0.98547624

model with missing value filled with mode

model	accuracy	f1 score
LR	0.821035774388	0.742860554142
DT	0.979697084849	0.974290856114
RF	0.846436132675	0.785284349592
xgb	0.98774225	0.98575060

conclusion

1. there is no much difference with model with missing value filled with 0 and mode
2. from all the model xgboost is best with high accuracy and f1 score
3. feature that help best predicting fraud in all model are - provider count, attending physician count, county, state, diagnosis code count

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 treatment 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 to 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. Accuracy

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 [393]:

```
# 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 [394]:

```
# 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' 'State'
 'County' 'NoOfMonths_PartACov' 'NoOfMonths_PartBCov'
 'ChronicCond_Alzheimer' 'ChronicCond_Heartfailure'
 'ChronicCond_KidneyDisease' 'ChronicCond_Cancer'
 'ChronicCond_ObstrPulmonary' 'ChronicCond_Depression'
 'ChronicCond_Diabetes' 'ChronicCond_IschemicHeart'
 'ChronicCond_Osteoporosis' 'ChronicCond_rheumatoidarthritis'
 'ChronicCond_stroke' 'IPAnnualReimbursementAmt' 'IPAnnualDeductibleAmt'
 'OPAnnualReimbursementAmt' 'OPAnnualDeductibleAmt']
```

merging train_inpatient, train_outpatient data

In [383]:

```
## https://www.kaggle.com/dimagmehanic/mc-term-paper
## 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 dataframe as keys.
## so columns with same name will be used as keys ( so repeting of columns will be avoided ) and remaining 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.
columns], how = 'outer')
final_data.shape
```

Out[383]:

(558211, 30)

In [384]:

```
final_data.head()
```

Out[384]:

	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
	BeneID	ClaimID	ClaimStartDt	ClaimEndDt	Provider	InscClaimAmtReimbursed	AttendingPhysician	Operating
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 [385]:

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

In [386]:

```
final_data.shape
```

Out[386]:

```
(558211, 54)
```

merging final_data, train_target data

In [387]:

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

In [388]:

```
final_data.shape
```

Out[388]:

```
(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 [12]:

```
final_data.head()
```

Out[12]:

	BenelID	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 [395]:

```
## 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[395]:

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

In [366]:

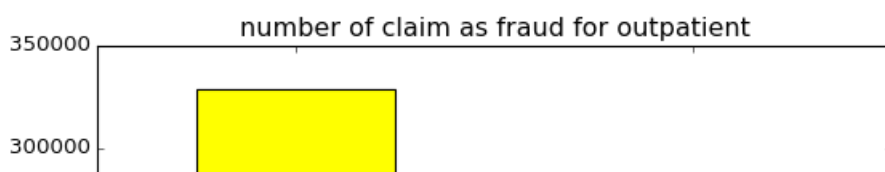
```
class_count = train_outpatient['PotentialFraud'].value_counts().to_frame()

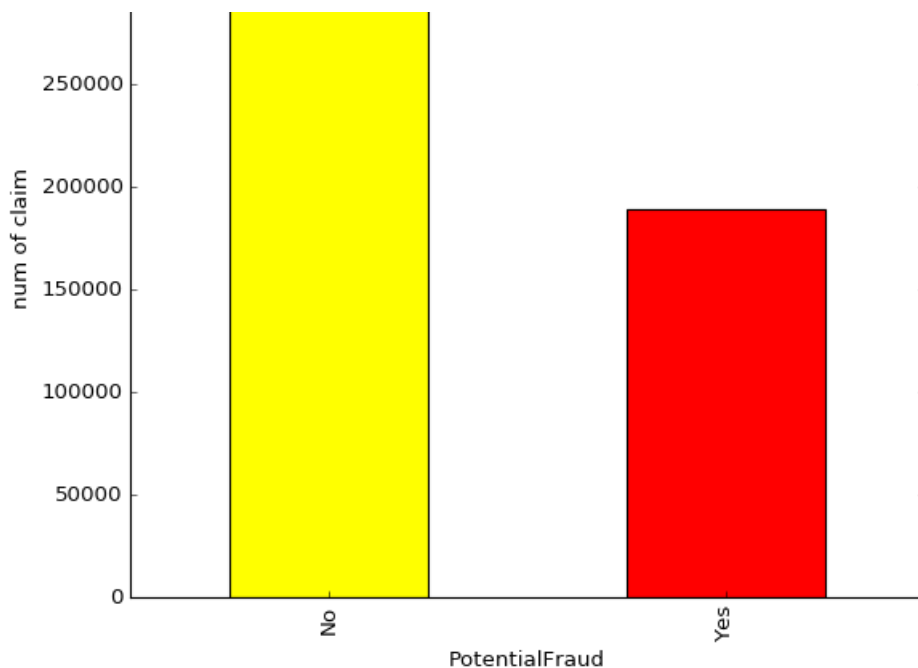
""" calculating percentage of yes and no class """

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 [396]:

```
## 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[396]:

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

In [368]:

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

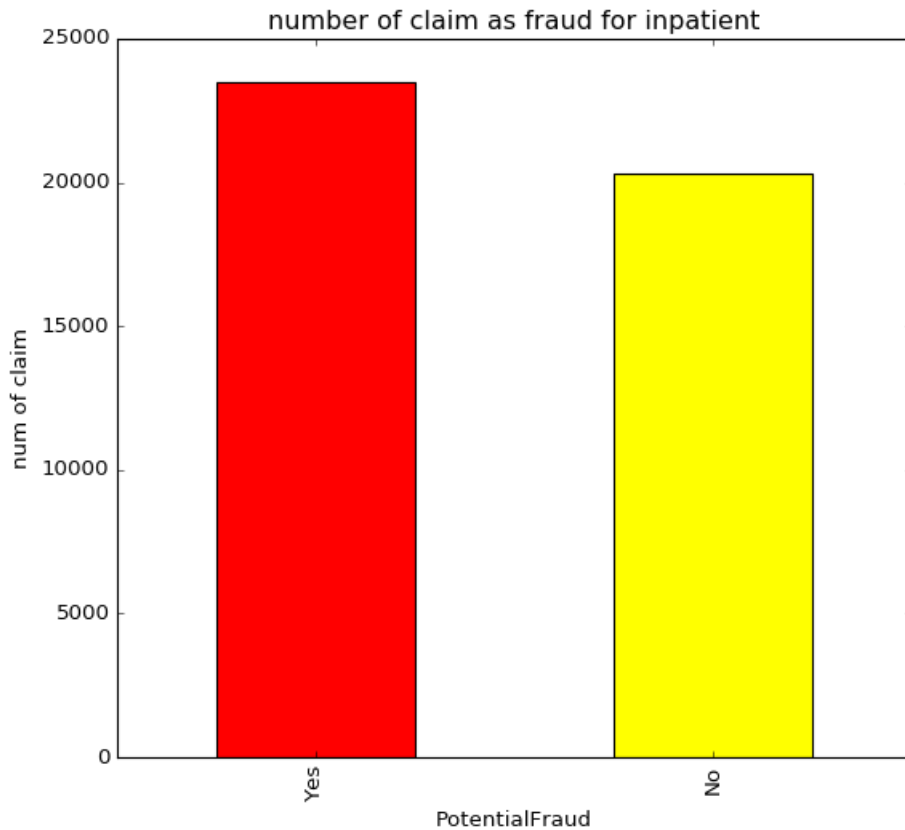
""" calculating percentage of yes and no class """

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_count['PotentialFraud']['No']))*100
print('percentage of non fraud class : ',percentage_no)

train_inpatient['PotentialFraud'].value_counts().plot(kind='bar',color=('red','yellow'),title='number 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
```


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 [369]:

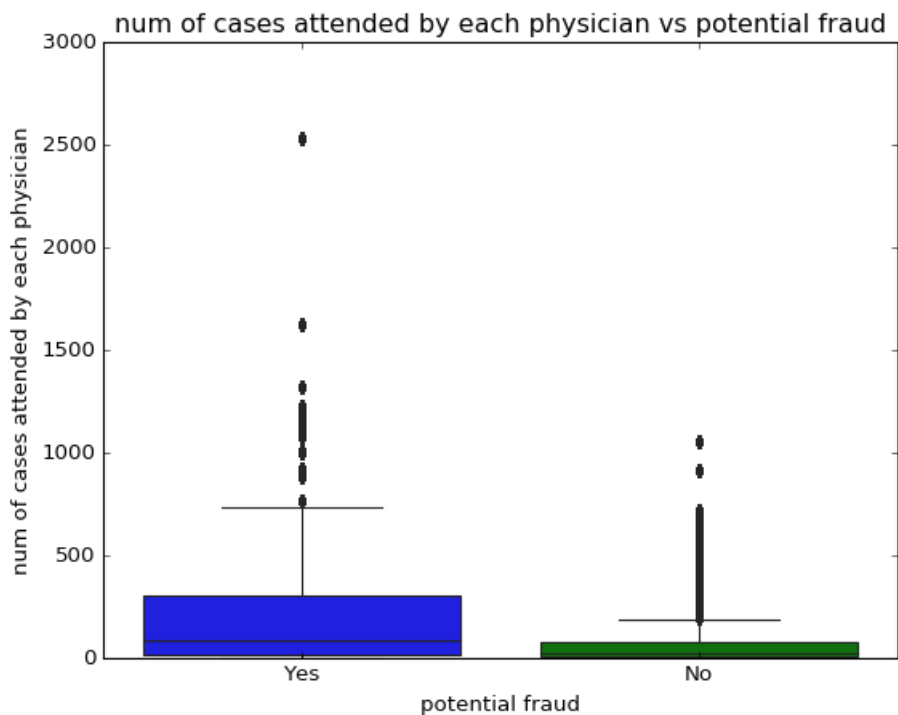
```
def count_(data,column,ylabel,title):  
    """ calculating num of cases attended by physician and plotting vs fraud """  
  
    ## counting number of cases for each physician and saving in dict  
  
    count = data[column].value_counts().to_dict()  
  
    col = str(column)+'count'          ## giving column name to be added  
  
    ## making new column for number of cases for each physician as 'AttendingPhysiciancount' by mapping 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 [370]:

```
count_(train_outpatient,'AttendingPhysician','num of cases attended by each physician','num of cases 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

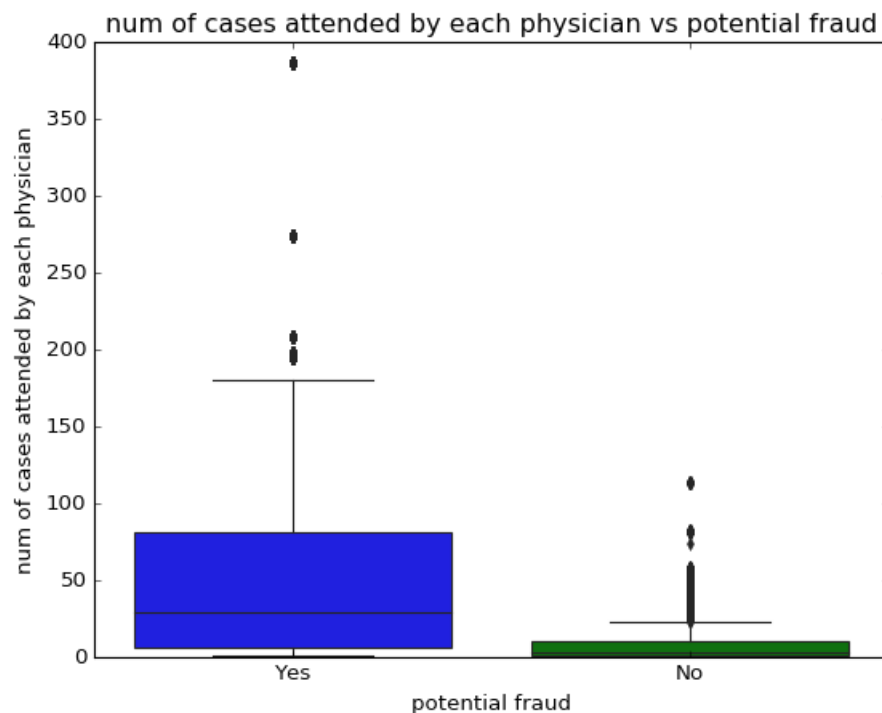
1. The most attended physician is PHY330576 attending 2534 cases.
2. If a physician has attended more number of case, then the physician involved in making fraud claim is more
3. The top 5 attended physician are PHY330576,PHY350277,PHY412132,PHY423534,PHY314027
4. The top attended physician's make more 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 [371]:

```
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
PHY431177    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

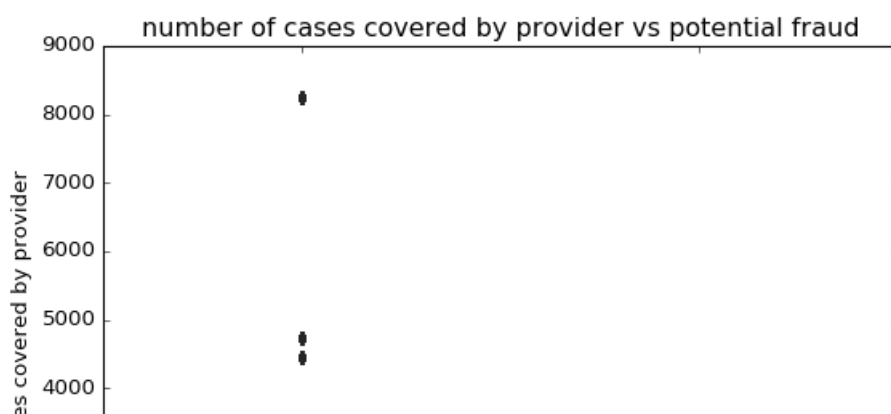
3.3. Provider feature

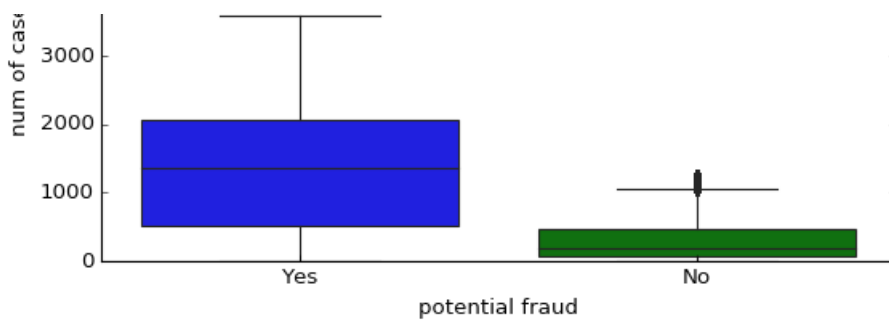
3.3.1. Univariate analysis - Analysing Provider feature with bar plot for outpatient

In [372]:

```
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 [373]:

```
""" 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 probabiltly for making fraud claim

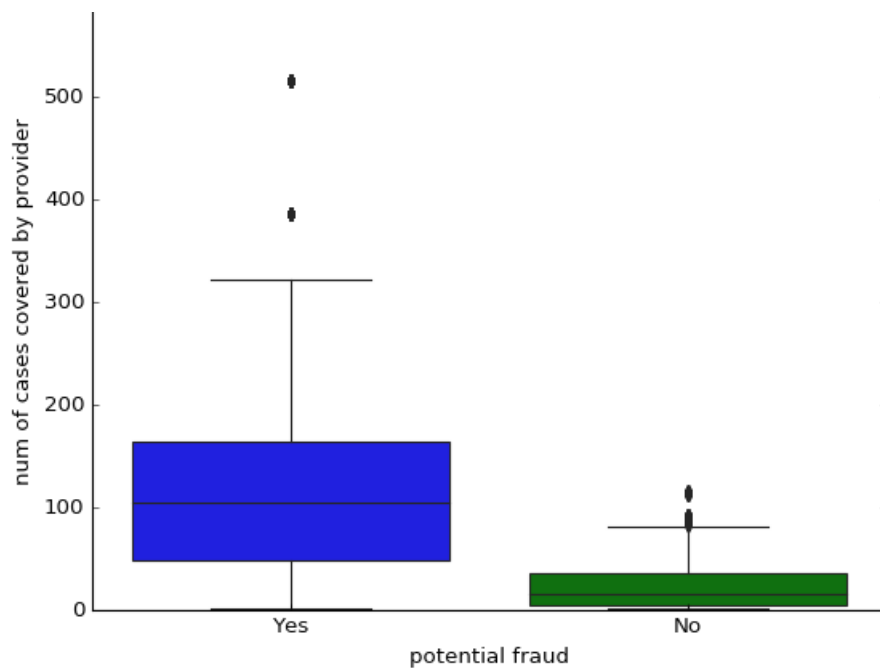
3.3.2. Univariate analysis - Analysing Provider feature with bar plot for inpatient

In [374]:

```
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
```

600 number of cases covered by provider vs potential fraud



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 probability for making fraud claim

3.3.3. Bivariate analysis - pair plot on AttendingPhysician and Provider

In [389]:

```
## 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['Provider'] == 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 [390]:

```

## 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)

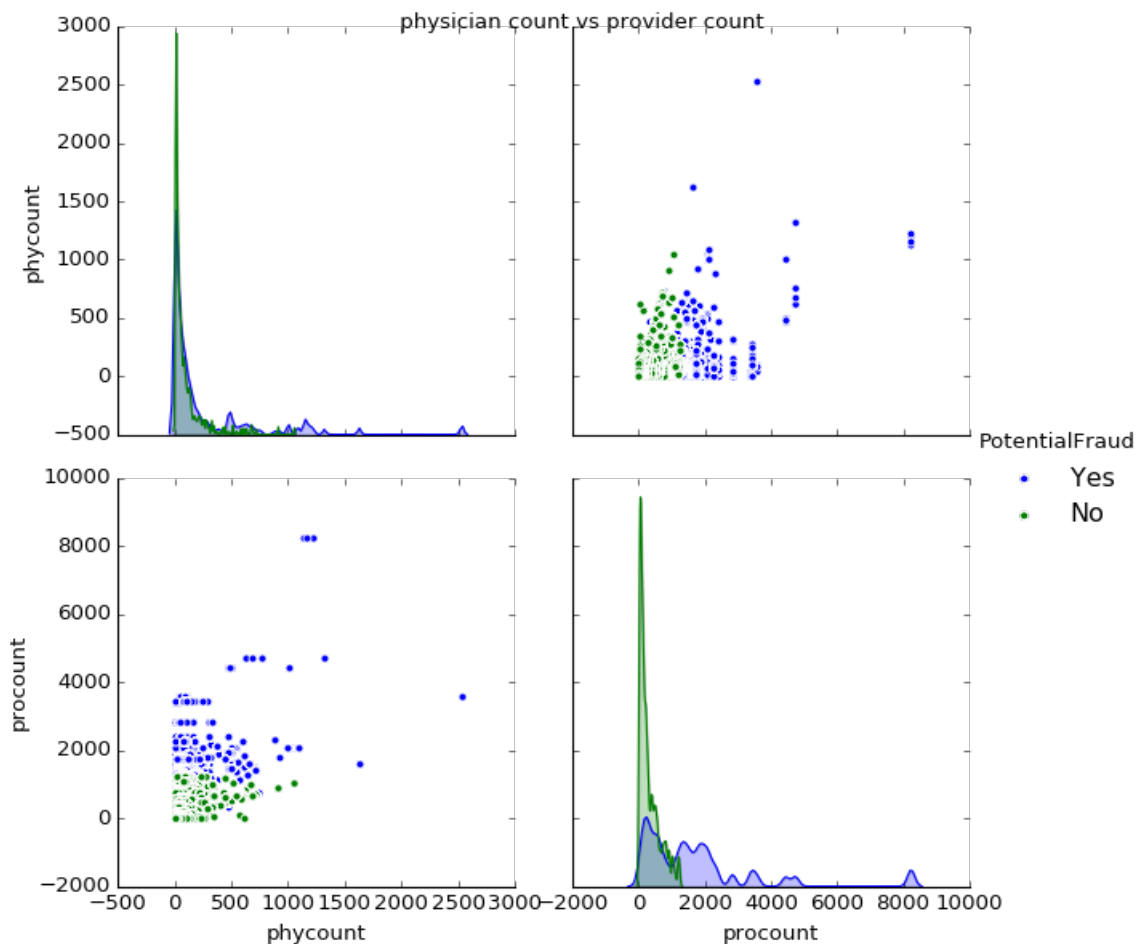
## adding procount column by mapping num of cases covered by pro with Provider
final_data['procount']=final_data['Provider'].map(provider_count)

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

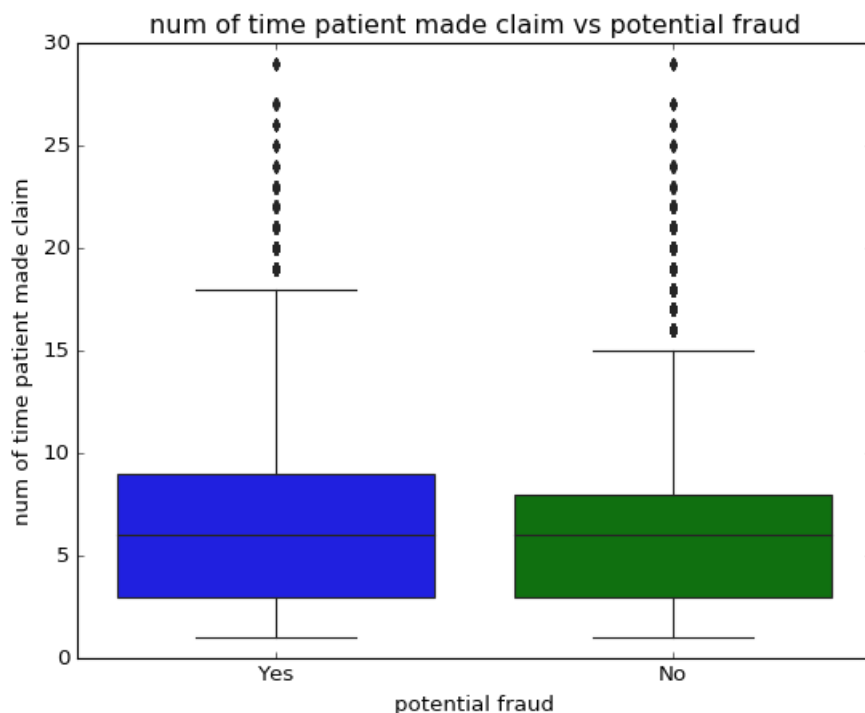
3.4. BeneID feature

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

In [397]:

```
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  
BENE63544       27  
BENE59303       27  
Name: BeneID, dtype: int64
```



conclusion

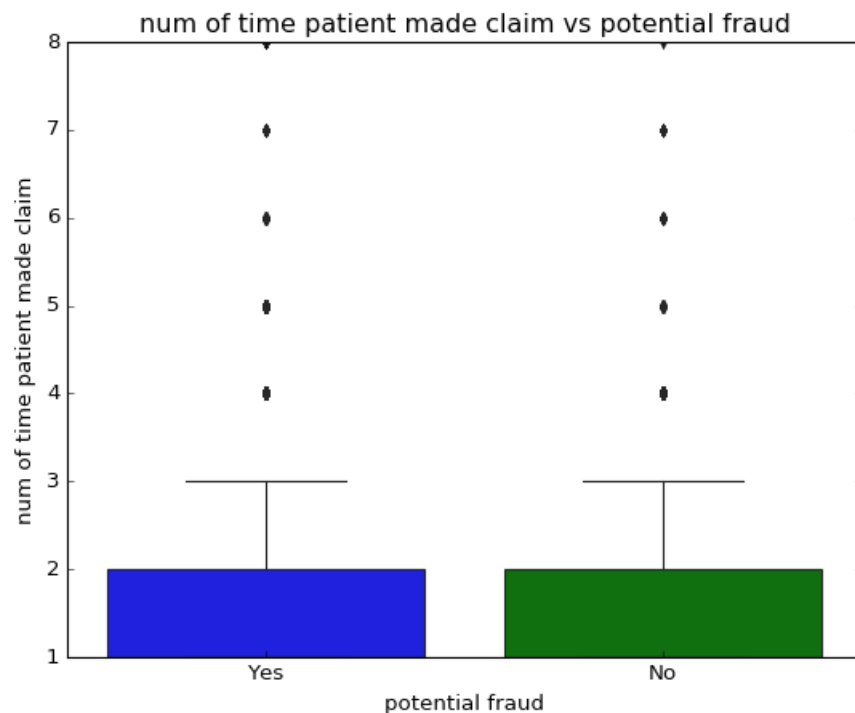
1. The most frequent patient BeneID is BENE42721 with 29 claim made
2. number of claim made by most patient is between 1 to 9
3. The top 5 attended patient are BENE42721,BENE118316,BENE59303,BENE63544,BENE63504
4. patient who already made claim between 2 to 9 time tend to make more fraud claim

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

In [398]:

```
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  
BENE117116        7  
BENE119457        7  
BENE64791         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 occurring 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 [399]:

```
## 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 = []
```



```

labelPhyBene = []
for i in range(len(sortPhyBene[0:5])):
    df = final_data.loc[(final_data['AttendingPhysician'] == sortPhyBene[i][0][0]) & (final_data['BeneID'] == 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:

```

(('PHY313322', 'BENE118316'), 'Yes')
(('PHY339042', 'BENE66093'), 'Yes')
(('PHY385072', 'BENE41087'), 'Yes')
(('PHY385072', 'BENE26003'), 'Yes')
(('PHY385072', 'BENE82133'), 'Yes')

```

In [400]:

```

## 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))

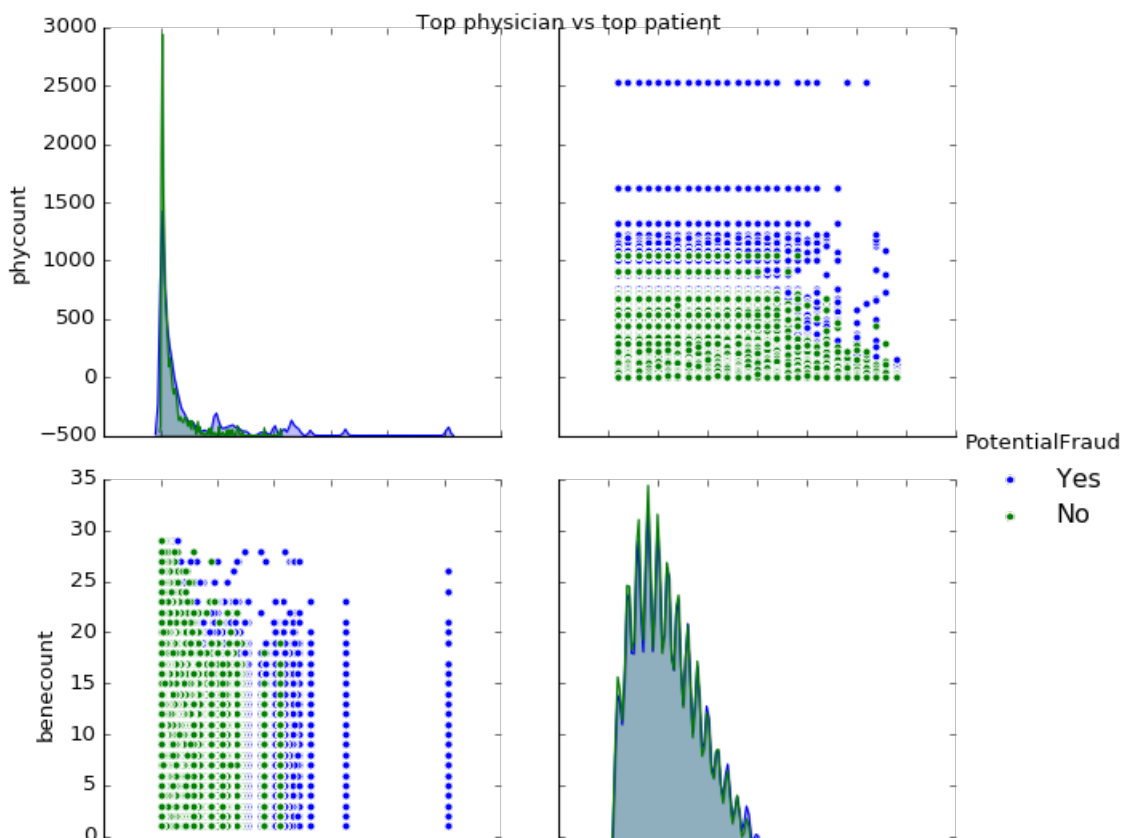
## adding phycount column by mapping num of cases attended by phy with AttendingPhysician
final_data['phycount']=final_data['AttendingPhysician'].map(attendingPhysician_count)

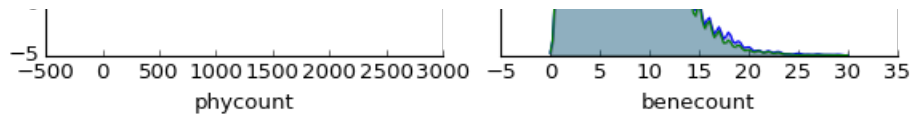
## adding phycount column by mapping num of cases attended by patient with BeneID
final_data['benecount']=final_data['BeneID'].map(bene_count)

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

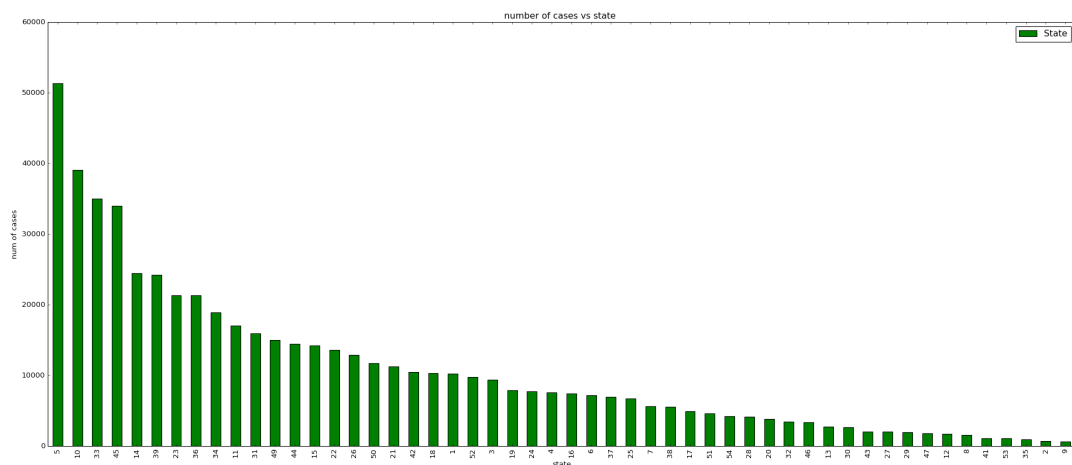
3.5.1.Univariate analysis - Analysing State feature with bar plot

In [401]:

```
## 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()
```

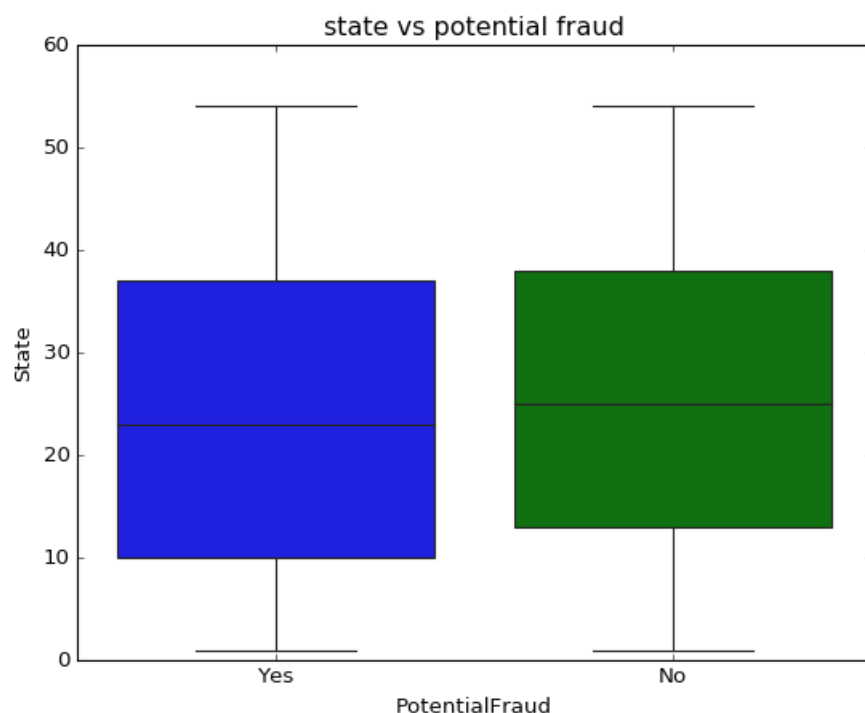
```
num of different state : 52
5      51350
10     39073
33     35024
45     34022
14     24417
Name: State, dtype: int64
```



In [402]:

```
## state vs fraud
```

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



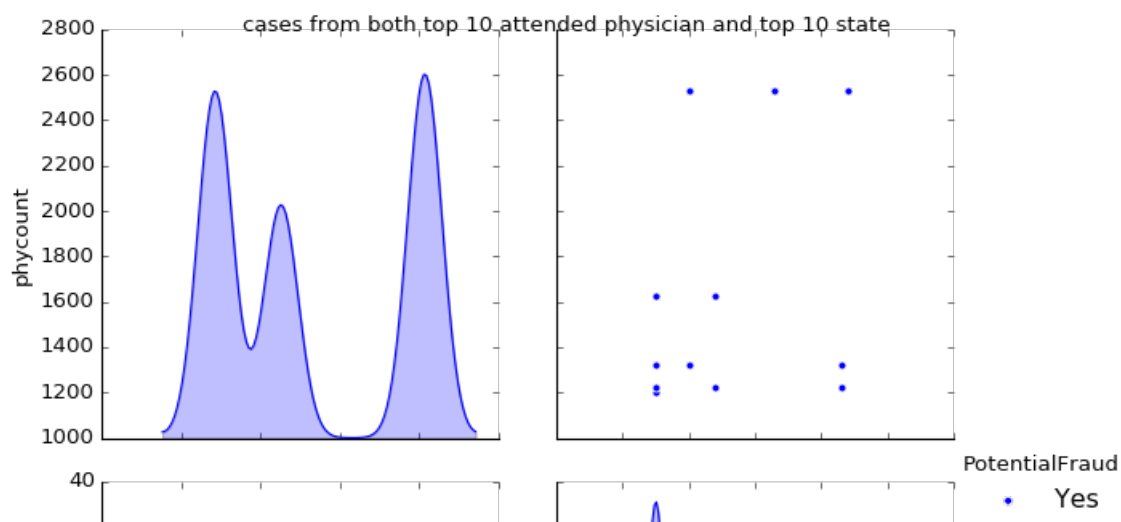
3.5.2. Bivariate analysis - Analysing AttendingPhysician and State feature with pair plot

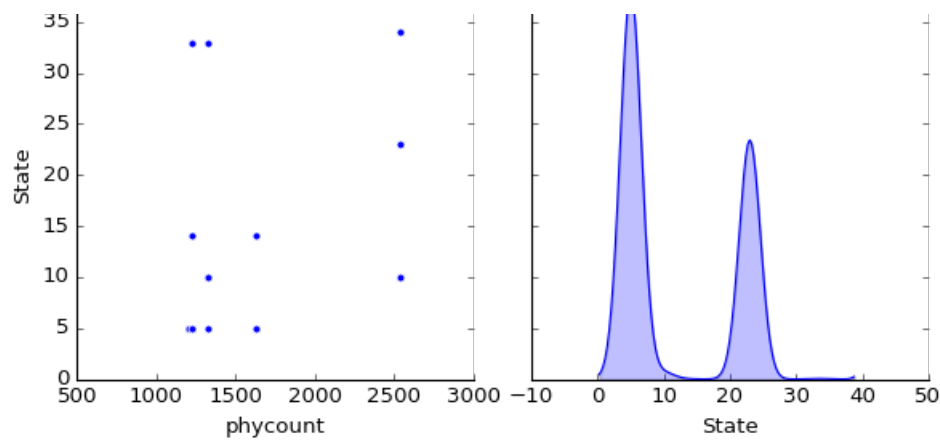
In [403]:

```
## 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(state)]

## 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()
```





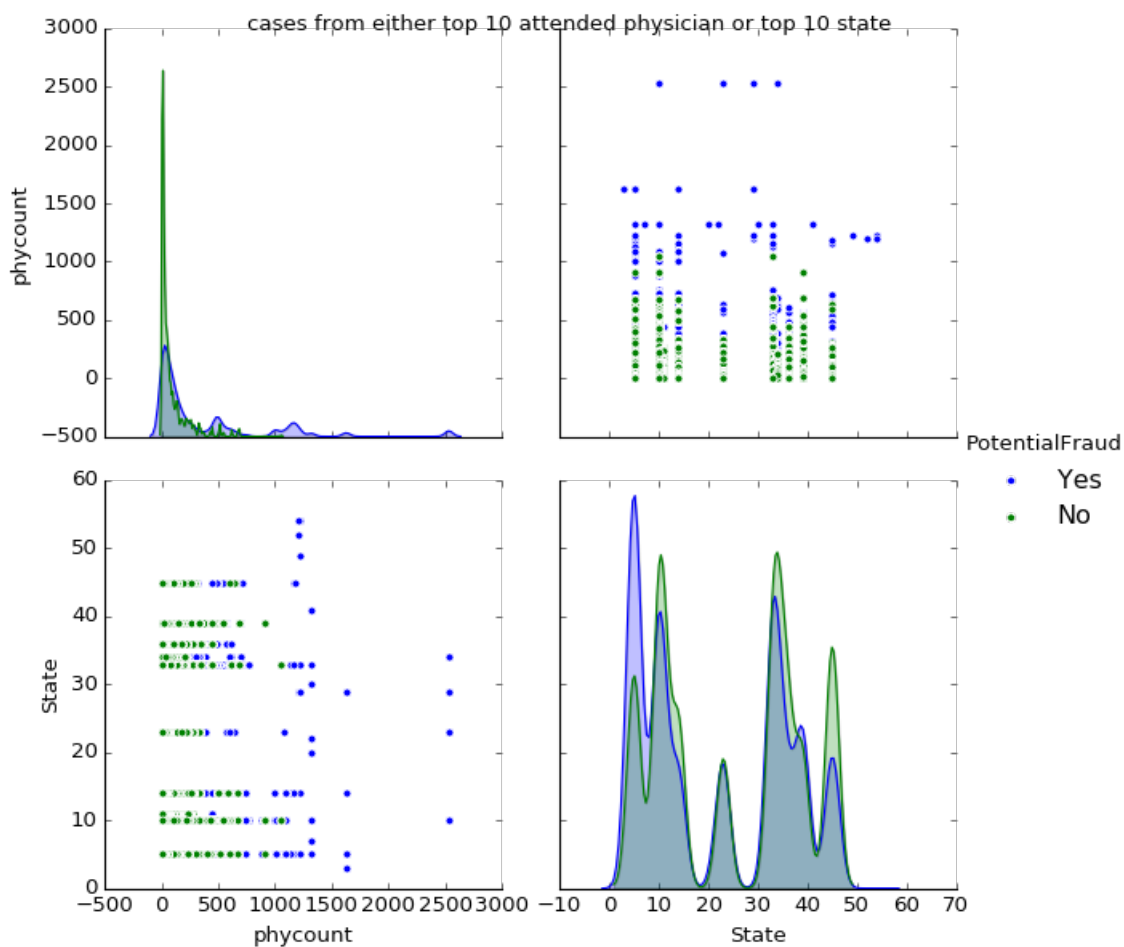
3.5.3. Bivariate analysis - Analysing AttendingPhysician or State feature with pair plot

In [404]:

```
## 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,
- 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 tend 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 [405]:

```

## 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 [406]:

```

final_data_diagnosis = final_data_diagnosis.dropna()
final_data_diagnosis.shape

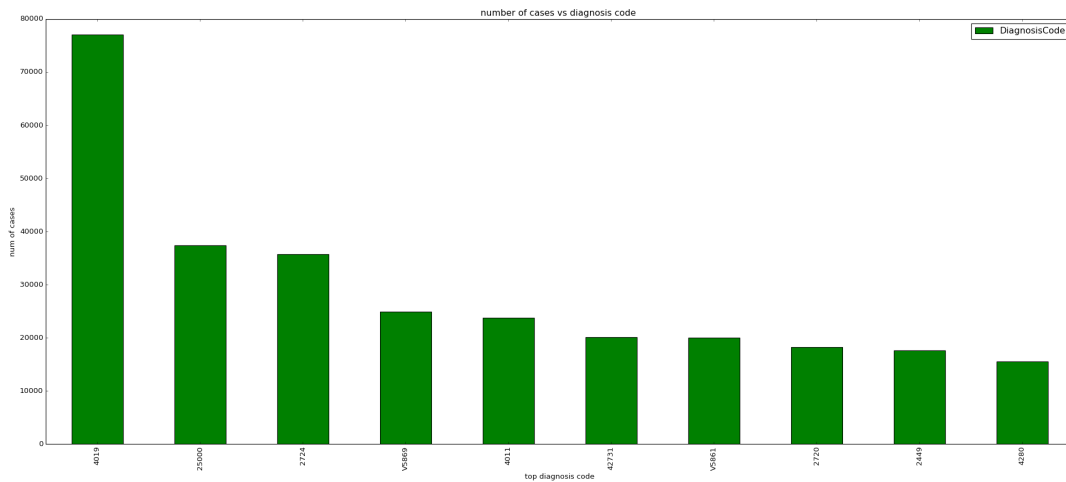
```

Out[406]:

```
(1680716, 1)
```

In [407]:

```
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

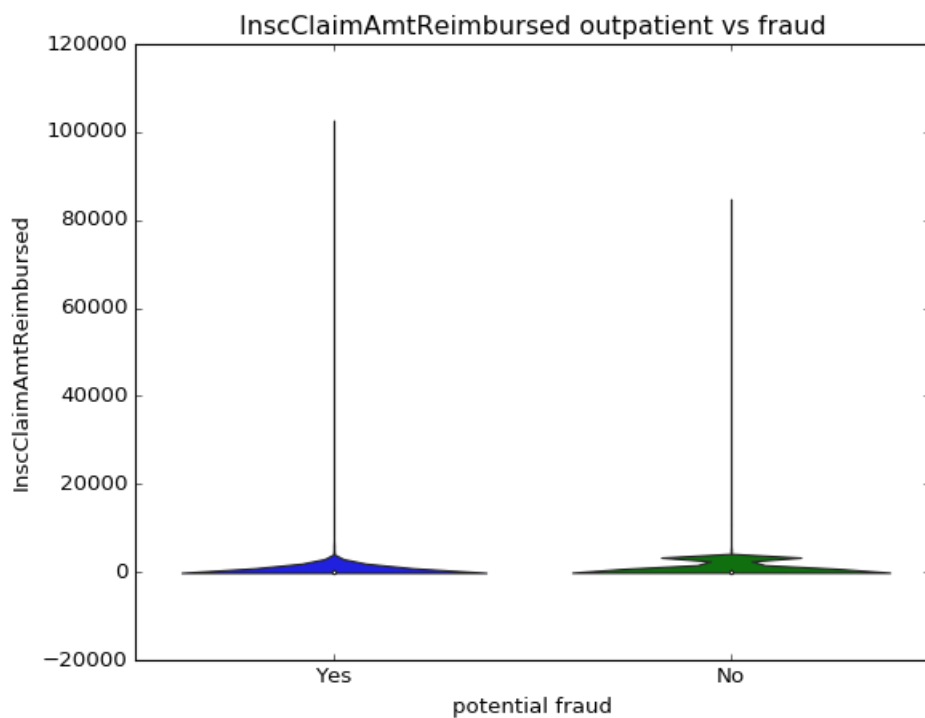
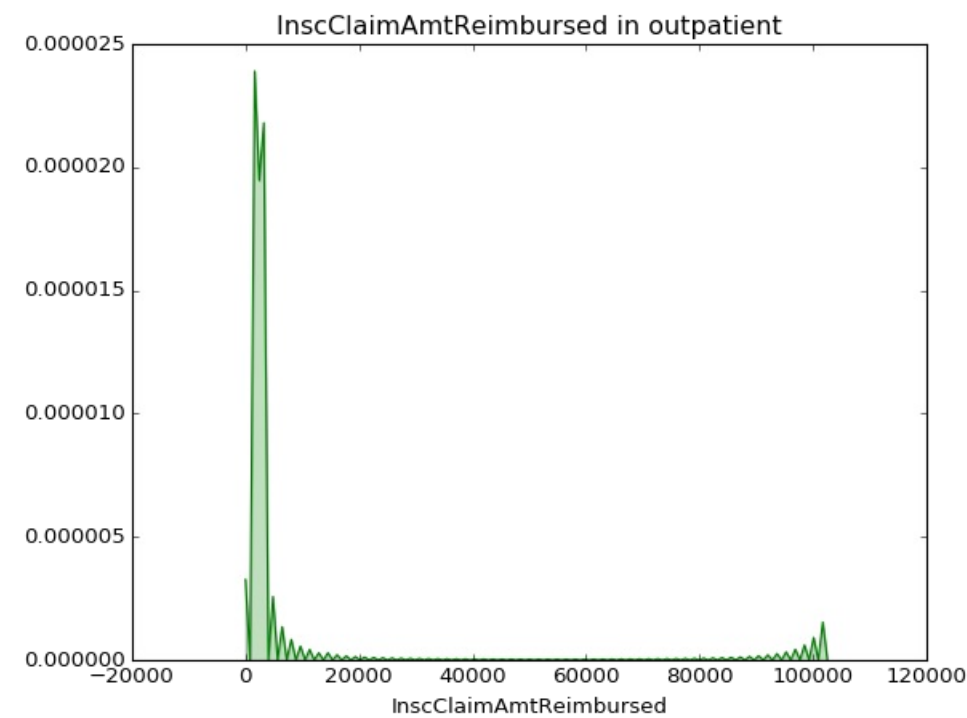
In [408]:

```
## 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()
```



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
```

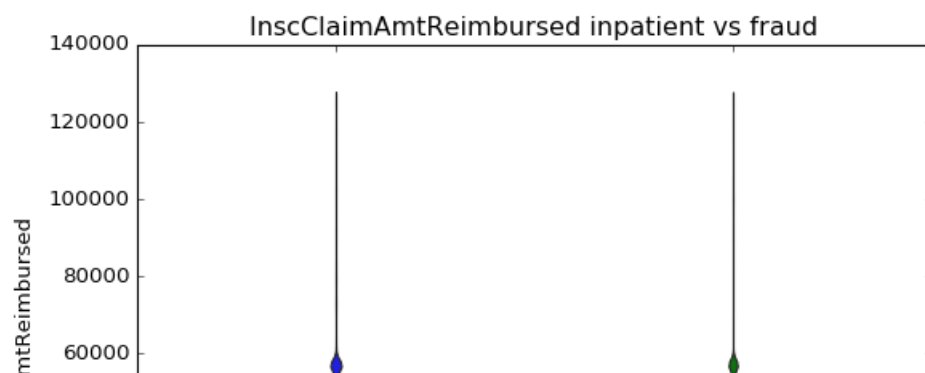
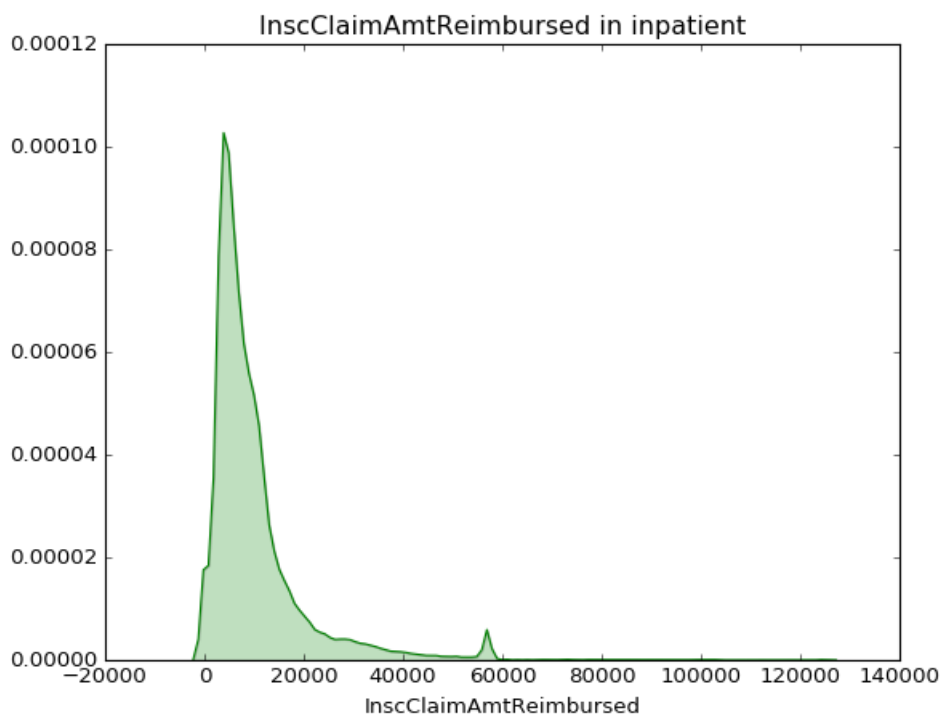
In [409]:

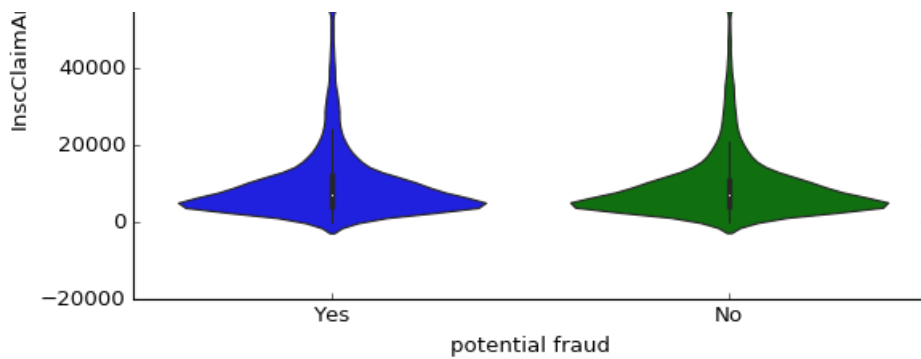
```
## using kde plot which use probability density function to analyse amount claimed

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 [410]:

```
## taking different race count

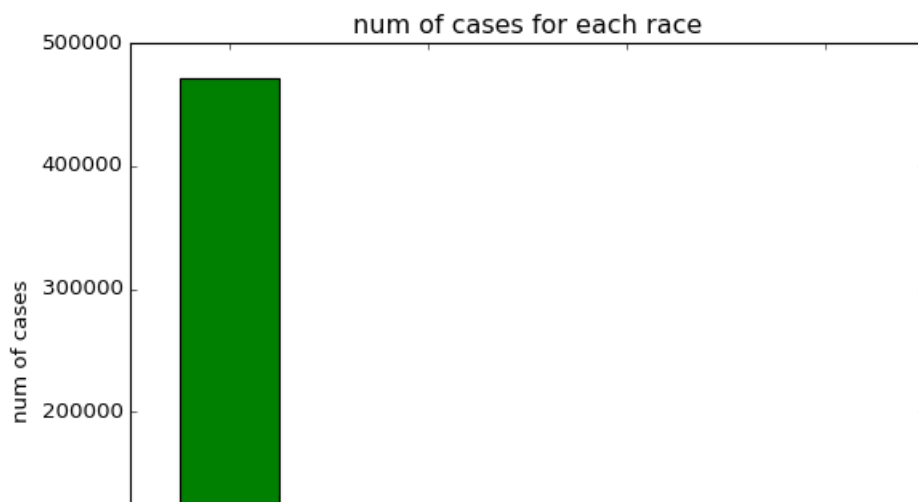
race_count = final_data['Race'].value_counts()
print(race_count)
```

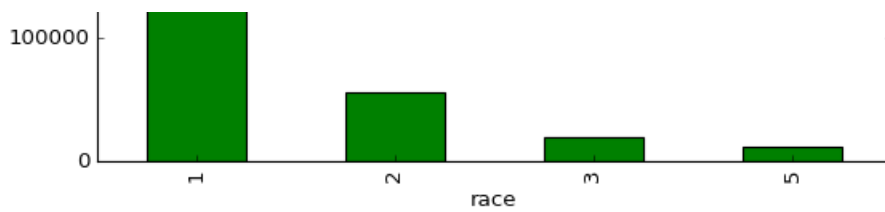
```
1    471036
2     55640
3     19715
5      11820
Name: Race, dtype: int64
```

In [411]:

```
## 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()
```

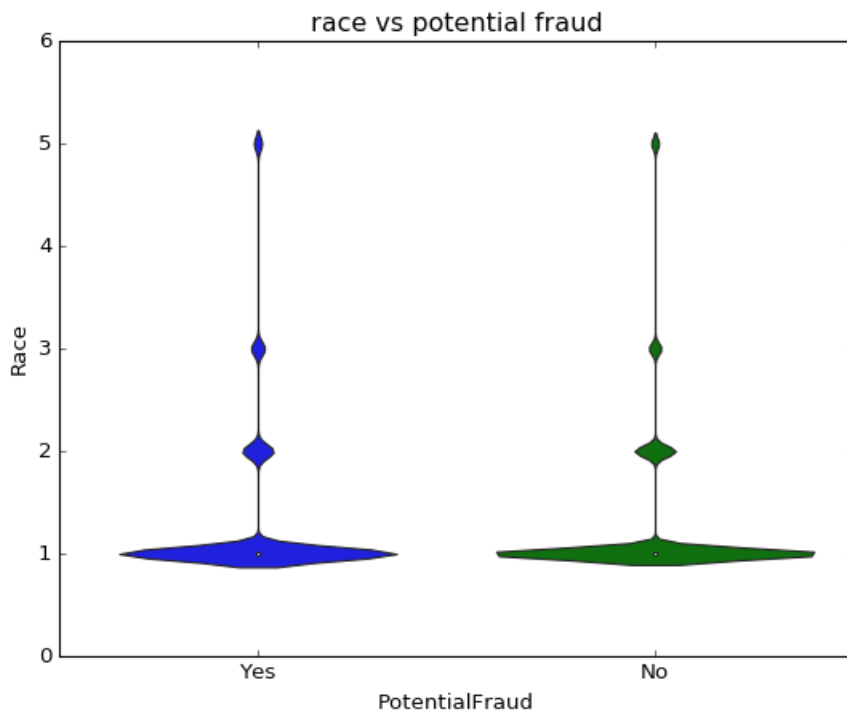




3.8.2. checking race feature with potential fraud using violin plot

In [412]:

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



conclusion

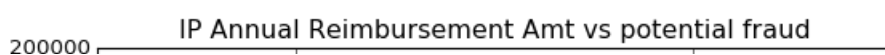
1. bar plot shows people belonging to race with category 1 occur more than other
2. further it is checked with violin plot on potential fraud
3. this violin plot give that race 1 has made more fraud claim than other
4. other category only very few and vary with same distribution on fraud or non fraud

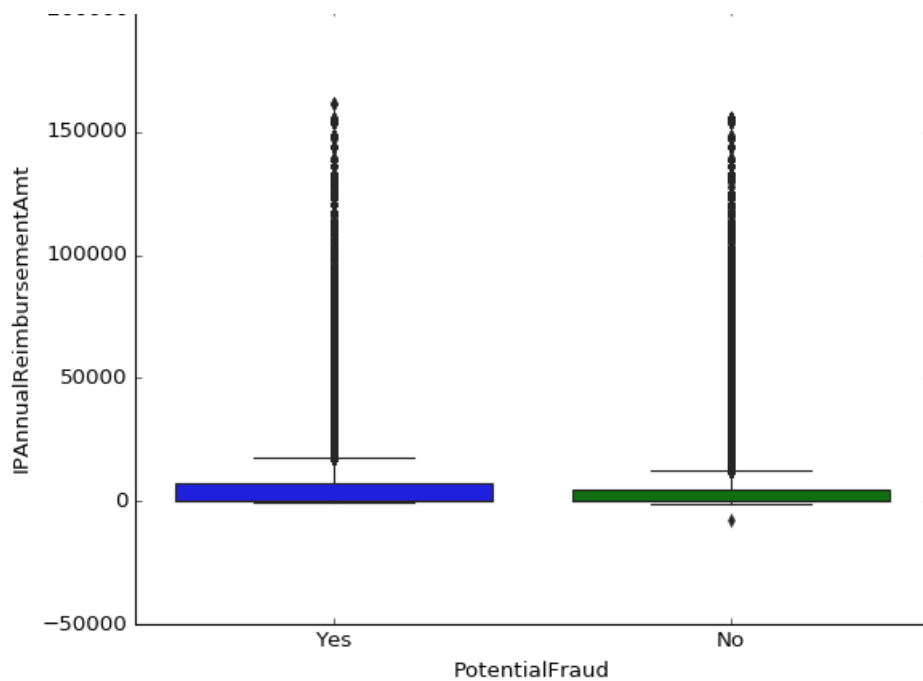
3.9. IPAnnualReimbursementAmt, OPAnnualReimbursementAmt feature

3.9.1. Univariate analysis - Analysing IPAnnualReimbursementAmt feature with box plot

In [413]:

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





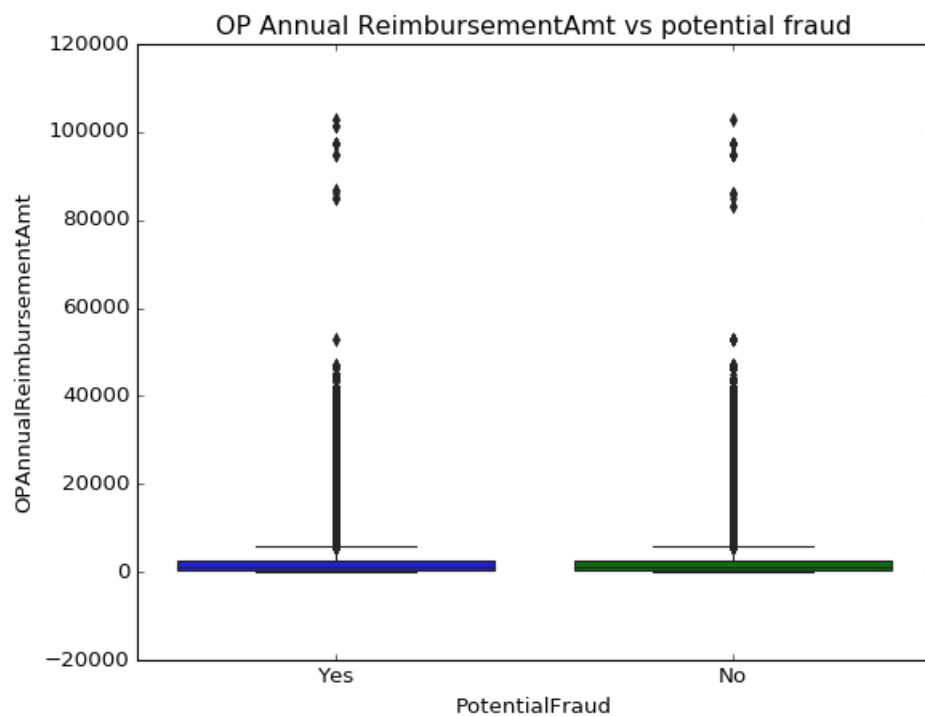
conclusion

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 differentiate them
3. but it shows that highest amount claimed was fraud claim

3.9.2. Univariate analysis - Analysing OPAnnualReimbursementAmt feature with box plot

In [414]:

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



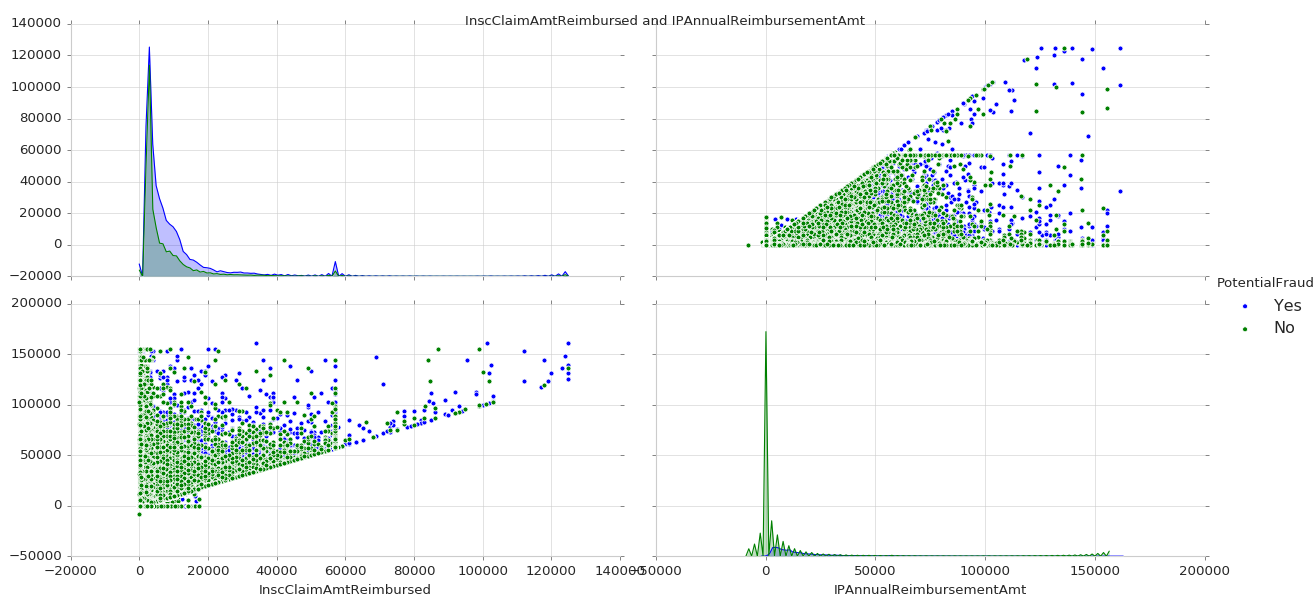
conclusion

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 differentiate them
3. but it shows that highest amount 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 [415]:

```
sns.set_style("whitegrid")
sns.pairplot(final_data,hue='PotentialFraud',vars=['InscClaimAmtReimbursed','IPAnnualReimbursementAmt'],height=3.8,aspect=2)
plt.suptitle('InscClaimAmtReimbursed and IPAnnualReimbursementAmt')
plt.show()
```



conclusion

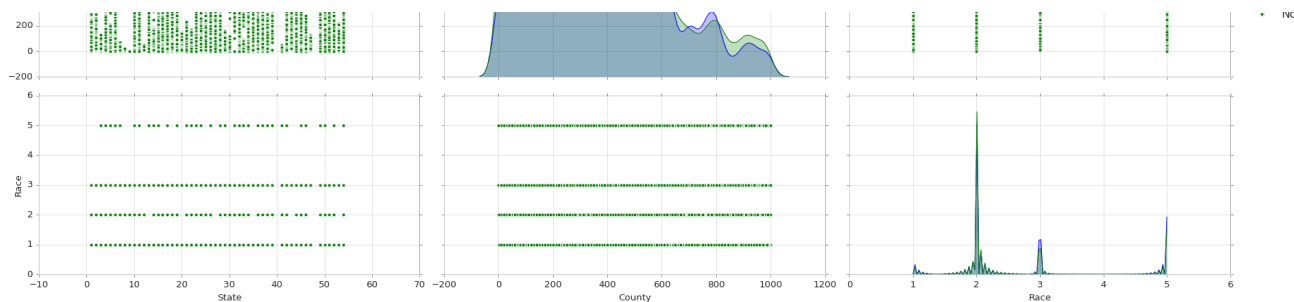
1. InscClaimAmtReimbursed and IPAnnualReimbursementAmt overlap more, but show that some fraud claim are made with higher amount than non fraud claim
2. this show these two feature can help in classifying fraud claim that are made with higher amount

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

In [416]:

```
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 [419]:

```
## 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 32 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
BeneIDcount           40474 non-null float64
dtypes: datetime64[ns](2), float64(9), object(21)
memory usage: 11.0+ MB
```

number of days admitted

In [420]:

```
train_inpatient['numOfDaysAdmitted'] = train_inpatient['DischargeDt'] - train_inpatient['AdmissionD
```

```
t']
```

```
In [421]:
```

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

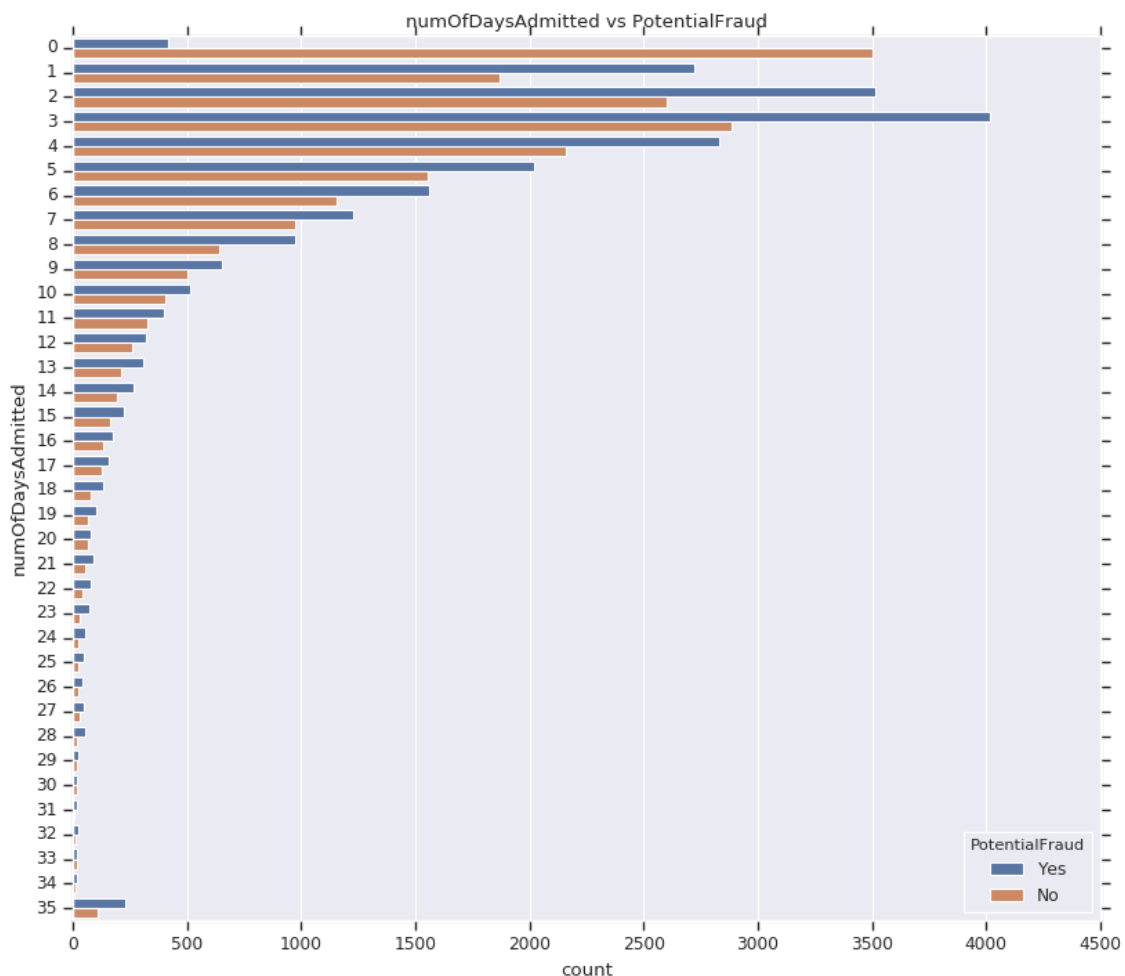
```
In [422]:
```

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

3.12.1. numOfDaysAdmitted vs PotentialFraud using countplot

```
In [423]:
```

```
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 [424]:
```

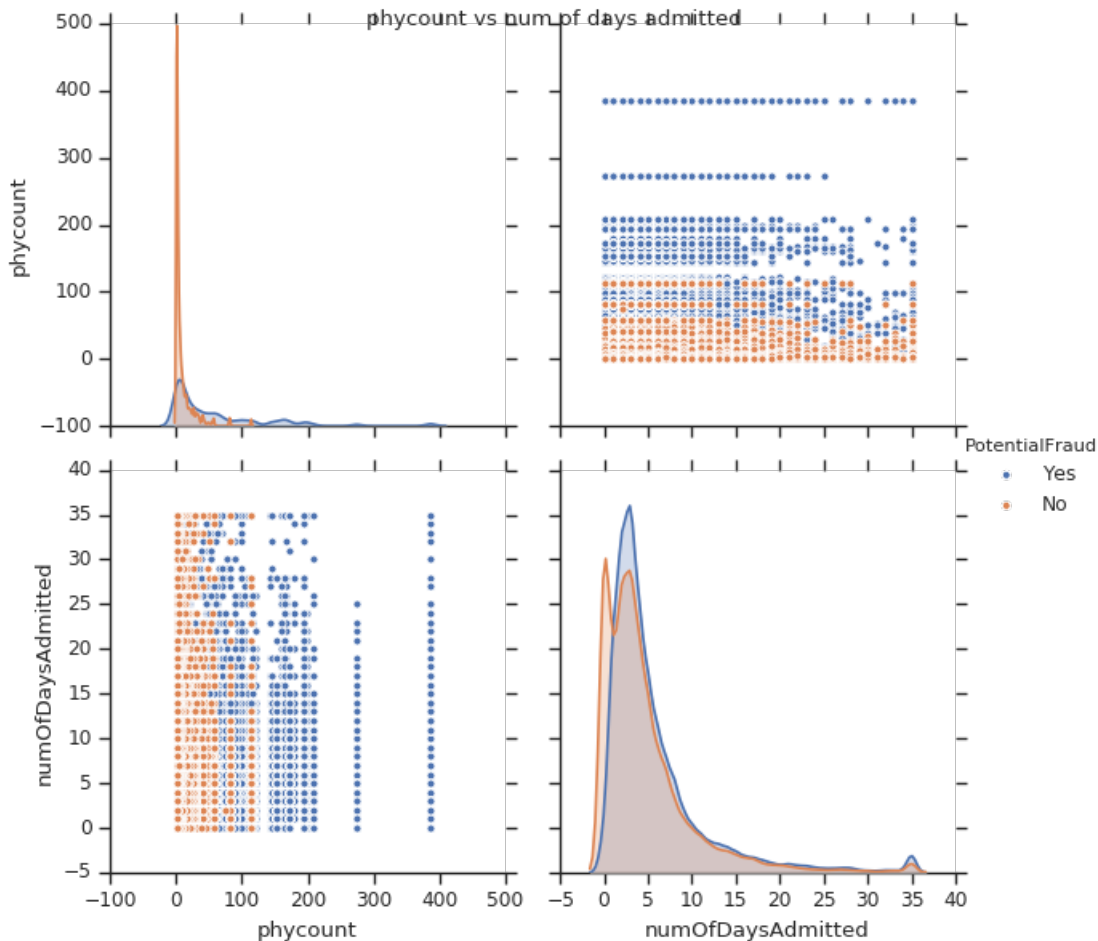
```
## num of cases attended by phy  
attendingPhysician_count = train_inpatient['AttendingPhysician'].value_counts().to_dict()  
print('num of cases attended by phy: ', attendingPhysician_count)
```

```
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

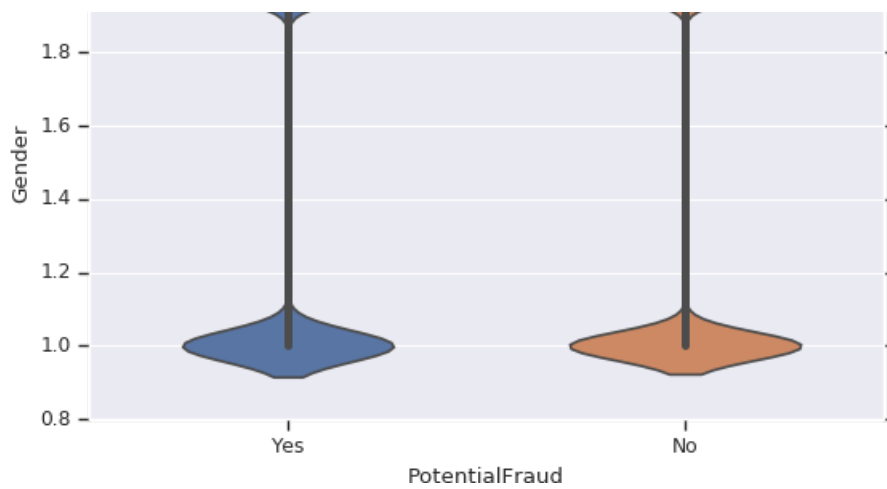
1. num of day patient admitted can be got from, discharge date - admitted
2. from counterplot, patient who are admitted between 2 to 7 tend to show more fraud claim
3. from pair plot, top attended physician tend to make more fraud claim in all case of number of day admitted
4. from all analysis it is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

3.13. Gender feature

In [425]:

```
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

4. Data preprocessing

In [162]:

```
final_data.columns
```

Out[162]:

```
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_Osteoporosis',
       'ChronicCond_rheumatoidarthritis', 'ChronicCond_stroke',
       'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
       'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt', 'PotentialFraud'],
      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_Osteoporosis	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
DischargedDt	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_Osteoporosis	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,'O':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_Osteoporosis'] = final_data['ChronicCond_Osteoporosis'].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})
```

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)
```

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 dischargeddt)

(getting information from admission,discharge)

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
```

```
0      0.0
1      0.4
2     12.0
3     18.0
4      4.0
Name: numOfDayAdmitted, 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['numOfDayForClaim'] = final_data['ClaimEndDt'] - final_data['ClaimStartDt']

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

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

Out[259]:

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

In [260]:

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

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

mean : 1.7

Out[260]:

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

5.6. calculating total ip,op amount reimburse

In [261]:

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

Out[261]:

	IPAnnualReimbursementAmt	OPAnnualReimbursementAmt	IPAnnualDeductibleAmt	OPAnnualDeductibleAmt
0	36000	60	3204	70
1	24000	450	2136	200

	IPAnnualReimbursementAmt	OPAnnualReimbursementAmt	IPAnnualDeductibleAmt	OPAnnualDeductibleAmt
2	19000	1000	1068	540
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_Osteoporosis'] + \
                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
```

```
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
```

HOL

```
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 occurred, 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()

```

```

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(Cl mAdmitDiagnosisCode_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_Osteoporosis', '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'
```

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 × 9 columns

In [151]:

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

In [304]:

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

In [152]:

```
## 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 [316]:

```
## 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 [153]:

```
final_data.head()
```

Out[153]:

	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
	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCond_Alzhe
3	17000.0	1068.0	1	1	0	39	600	0
4	13000.0	1068.0	0	1	0	39	280	0

5 rows × 9 columns

◀		▶
---	--	---

In [317]:

```
final_data.head()
```

Out[317]:

	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 × 9 columns

◀		▶
---	--	---

In [318]:

```
## dividing data and class label

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

In [319]:

```
X.shape
```

Out[319]:

```
(558211, 58)
```

In [320]:

```
Y.shape
```

Out[320]:

```
(558211,)
```

train test split

In [321]:

```
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 [322]:

```
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 [159]:

```
X_train.head()
```

Out[159]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCond
314387	12000.0	1068.0	0	1	0	3	60	1
394115	20.0	0.0	0	2	0	45	570	1
138537	30.0	30.0	0	1	0	14	640	1
186434	20.0	0.0	0	1	0	6	20	1
285742	16000.0	1068.0	1	1	0	45	480	0

5 rows × 58 columns



normalizing using min max scalar

In [323]:

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

In [324]:

```
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 [325]:

```
## 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',  
              'ClnAdmitDiagnosisCode_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 [163]:

```
X_train.head()
```

Out[163]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicC
314387	0.09600	1.00000	0	0.00	0	0.037736	0.060060	1
394115	0.00016	0.00000	0	0.25	0	0.830189	0.570571	1

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCo
138537	0.00024	0.02809	0	0.00	0	0.245283	0.640641	1
186434	0.00016	0.00000	0	0.00	0	0.094340	0.020020	1
285742	0.12800	1.00000	1	0.00	0	0.830189	0.480480	0

5 rows × 58 columns

◀		▶
---	--	---

In [326]:

```
X_train.head()
```

Out[326]:

	InscClaimAmtReimbursed	DeductibleAmtPaid	Gender	Race	RenalDiseaseIndicator	State	County	ChronicCo
183911	0.00072	0.0	0	0.0	0	0.773585	0.25025	0
441745	0.00072	0.0	1	0.0	0	0.075472	0.06006	1
161352	0.00040	0.0	0	0.0	1	0.075472	0.46046	1
370226	0.00000	0.0	1	0.0	0	0.603774	0.07007	1
348570	0.00000	0.0	1	0.0	0	1.000000	0.11011	1

5 rows × 58 columns

◀		▶
---	--	---

6. model

In [327]:

```
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):
    """ plotting 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)
```

```

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 [328]:

```

def f1(model,x_train,x_test,y_train,y_test):
    """ calculate 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))
    return f1_train,f1_test

```

In [329]:

```

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 [330]:

```

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: 3.4min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 7.5min finished
```

Out[330]:

```
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={'penalty': ['l1', 'l2'], 'C': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]},
                   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 [166]:

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

```
best parameter : {'penalty': 'l2', 'C': 0.01}
accuracy on train : 0.829396178085
```

In [228]:

```
## on best parameter

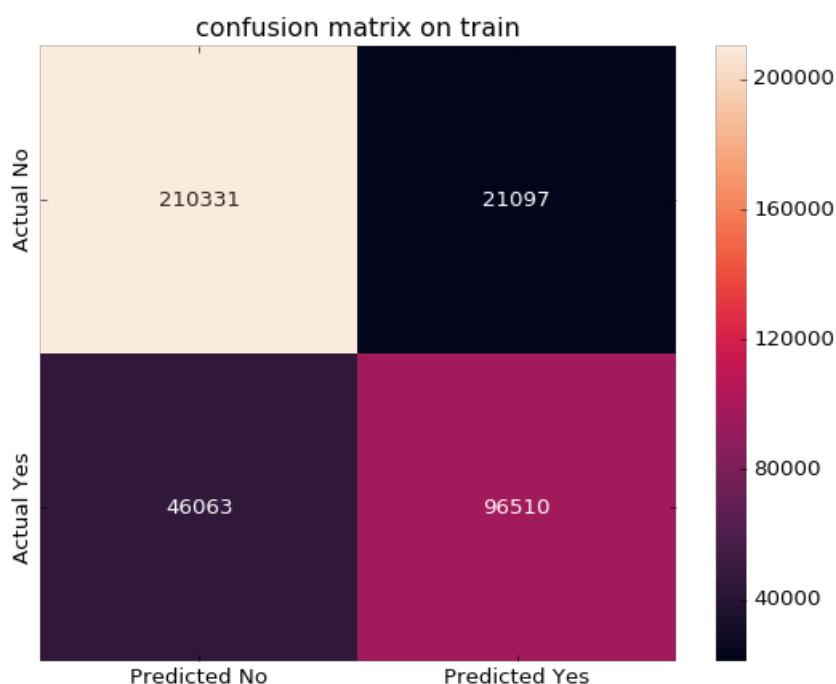
lr_tuned = LogisticRegression(C=0.01,penalty='l2')
lr_tuned.fit(X_train,Y_train)
print('accuracy on test :',lr_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.830921231204
```

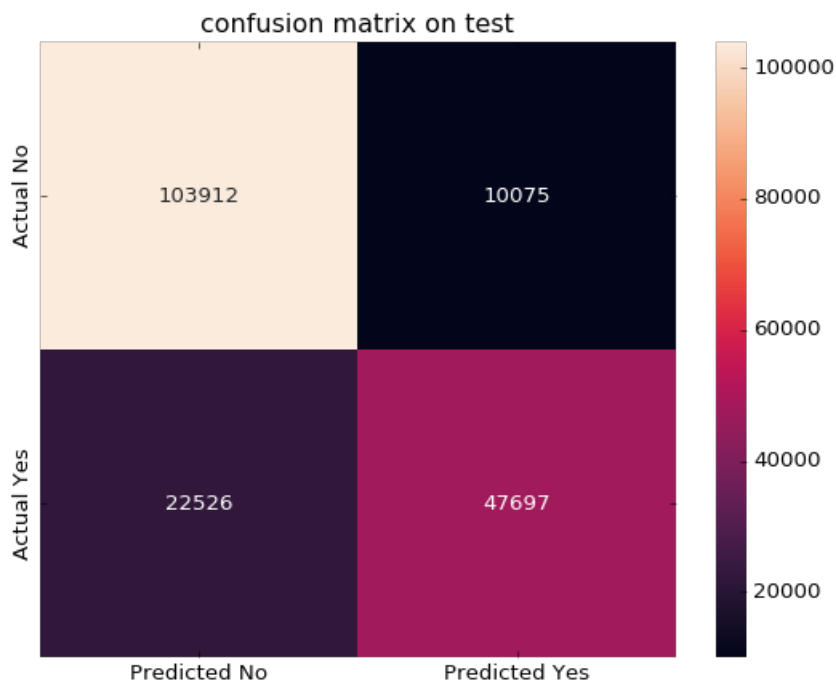
In [229]:

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

```
best train threshold : 0.458854891555
```



best test threshold : 0.461646093072



None

In [178]:

```
## checking confusion matrix value with threshold 0.5 and with best threshold with 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 - 91892
## true positive in train with best threshold - 96510

## false negative in train with threshold 0.5 - 50681
## false negative in train with best threshold - 46063

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:
[[218163  13265]
 [ 50681  91892]]
test confusion matrix without best threshold:
[[107469   6518]
 [ 24628  45595]]
best train threshold : 0.458854891555
best test threshold : 0.461646093072
train confusion matrix with best threshold:
[[210331  21097]
 [ 46063  96510]]
train confusion matrix with best threshold:
[[103912  10075]
 [ 22526  47697]]
```

In [206]:

```
## f1 score

f1_train,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)
```

```
Train F1 score is : 0.741871012376
Test F1 score is : 0.745294738076
```

In [174]:

```
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
ip_op_total_amount	-0.16683
OPAnnualDeductibleAmt	-0.13983
OPAnnualReimbursementAmt	-0.09119
State	-0.06582
diagnosis_V5869	-0.05721
DiagnosisCode_10_count	-0.04
num_of_phy	-0.03277
diagnosis_4011	-0.03257
ChronicCond_Diabetes	-0.03132
diagnosis_2720	-0.02711
ChronicCond_Depression	-0.0269
DiagnosisCode_3_count	-0.01811
ChronicCond_IschemicHeart	-0.01699
diagnosis_4019	-0.0166
diagnosis_25000	-0.01617
ChronicCond_Alzheimer	-0.0157
diagnosis_2449	-0.01505
ChronicCond_Heartfailure	-0.01466
ChronicCond_ObstrPulmonary	-0.01335
ChronicCond_Osteoporasis	-0.01286
diagnosis_V5861	-0.01144

diagnosis_v3001	-0.01144
procedure_9904.0	-0.01143
Gender	-0.00987
DiagnosisCode_2_count	-0.00835
ChronicCond_Cancer	-0.00673
num_of_chronic	-0.00663
ChronicCond_rheumatoidarthritis	0.00114
ChronicCond_KidneyDisease	0.0017
numOfDaysForClaim	0.00196
diagnosis_2724	0.00328
DiagnosisCode_4_count	0.00355
ClmAdmitDiagnosisCode_count	0.00376
procedure_66.0	0.00779
DiagnosisCode_1_count	0.009
IPAnnualDeductibleAmt	0.01055
diagnosis_4280	0.01337
RenalDiseaseIndicator	0.0136
DiagnosisCode_8_count	0.01499
DiagnosisCode_6_count	0.02047
DiagnosisCode_5_count	0.02195
diagnosis_42731	0.02405
DiagnosisCode_7_count	0.02773
procedure_4019.0	0.0287
ChronicCond_stroke	0.04248
DiagnosisCode_9_count	0.05252
Race	0.07241
IPAnnualReimbursementAmt	0.07746
procedure_2724.0	0.08702
procedure_8154.0	0.10247
num_of_diag_proc	0.12135
County	0.29504
numOfDaysAdmitted	0.2974
AttendingPhysicianCount	0.31342
DiagnosisGroupCode_count	0.33544
InscClaimAmtReimbursed	0.35479
BeneCount	0.37454
DeductibleAmtPaid	1.4126
ProviderCount	15.80232

6.1.2. model with null filled with mode

In [331]:

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

```
best parameter : {'penalty': 'l1', 'C': 0.01}
accuracy on train : 0.820922938709
```

In [332]:

```
## on best parameter

lr_tuned = LogisticRegression(C=0.01,penalty='l1')
lr_tuned.fit(X_train,Y_train)
print('accuracy on test :',lr_tuned.score(X_test,Y_test))
```

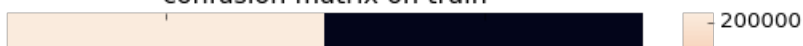
```
accuracy on test : 0.821035774388
```

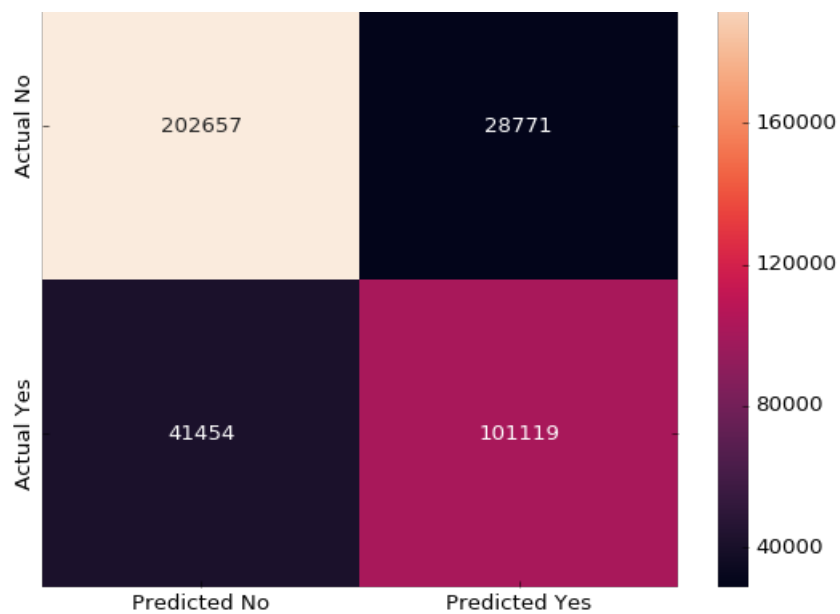
In [333]:

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

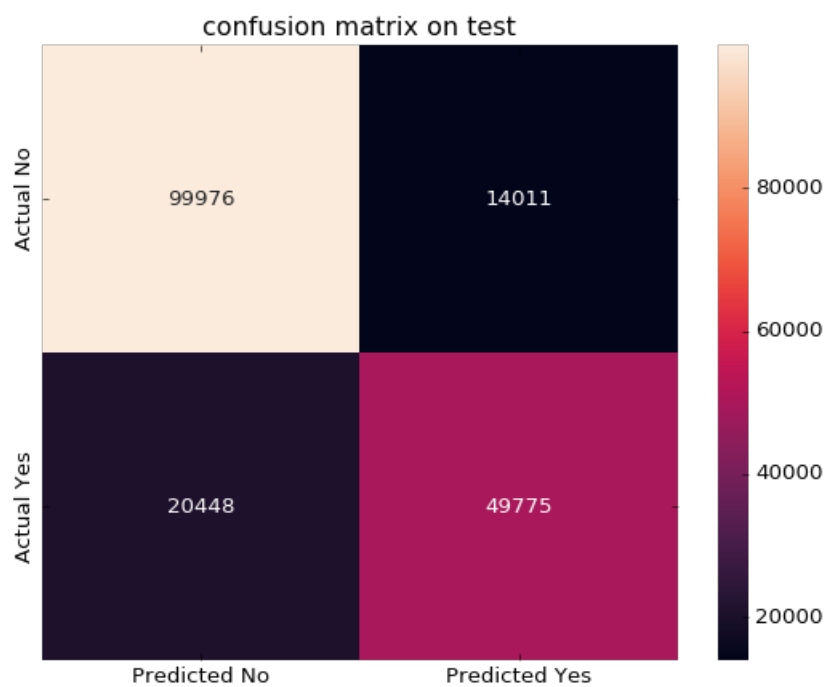
```
best train threshold : 0.441047680382
```

confusion matrix on train





best test threshold : 0.438223427577



None

In [334]:

```
## f1 score

f1_train,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)
```

Train F1 score is : 0.742258581899
Test F1 score is : 0.742860554142

In [335]:

```
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	-2.26087
DiagnosisGroupCode_count	-2.08078
procedure_4019.0	-0.17994
State	-0.11536
procedure_9904.0	-0.05619
DiagnosisCode_10_count	-0.04801
DiagnosisCode_9_count	-0.04038
ChronicCond_Alzheimer	-0.01812
ChronicCond_Diabetes	-0.01744
num_of_phy	-0.01428
ChronicCond_Depression	-0.00957
ChronicCond_IschemicHeart	-0.00474
ChronicCond_rheumatoidarthritis	-0.00209
diagnosis_4019	-6e-05
DiagnosisCode_3_count	0.0
DiagnosisCode_8_count	0.0
diagnosis_V5869	0.0
diagnosis_4011	0.0
diagnosis_42731	0.0
diagnosis_V5861	0.0
diagnosis_2720	0.0
DiagnosisCode_7_count	0.0
diagnosis_2449	0.0
diagnosis_4280	0.0
DiagnosisCode_6_count	0.0
DiagnosisCode_5_count	0.0
procedure_2724.0	0.0
procedure_8154.0	0.0
procedure_66.0	0.0
DiagnosisCode_1_count	0.0
DiagnosisCode_4_count	0.0
DiagnosisCode_2_count	0.0
diagnosis_2724	0.0
InscClaimAmtReimbursed	0.0
num_of_diag_proc	0.0
DeductibleAmtPaid	0.0
Gender	0.0
RenalDiseaseIndicator	0.0
ChronicCond_Heartfailure	0.0
ChronicCond_KidneyDisease	0.0
ChronicCond_Cancer	0.0
ChronicCond_ObstrPulmonary	0.0
diagnosis_25000	0.0
IPAnnualReimbursementAmt	0.0
num_of_chronic	0.0
OPAnnualReimbursementAmt	0.0
OPAnnualDeductibleAmt	0.0
numOfDaysAdmitted	0.0
numOfDaysForClaim	0.0
ip_op_total_amount	0.0
ClmAdmitDiagnosisCode_count	0.0
IPAnnualDeductibleAmt	0.0
ChronicCond_Osteoporasis	0.0001
ChronicCond_stroke	0.00548
Race	0.04917
BeneCount	0.23721
County	0.38501
ProviderCount	25.25654

6.2. DecisionTreeClassifier

In [336]:

```
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: 49.8s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 2.2min finished
```

Out[336]:

```
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={'max_depth': [5, 10, 50, 100, 250, 500, 1000], 'min_samples_split':
[5, 10, 50, 100, 200, 300, 400], 'criterion': ['gini', 'entropy']},
                   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 [183]:

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

```
best parameter : {'max_depth': 50, 'min_samples_split': 5, 'criterion': 'entropy'}
accuracy on train : 0.977328937623
```

In [184]:

```
## 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))
```

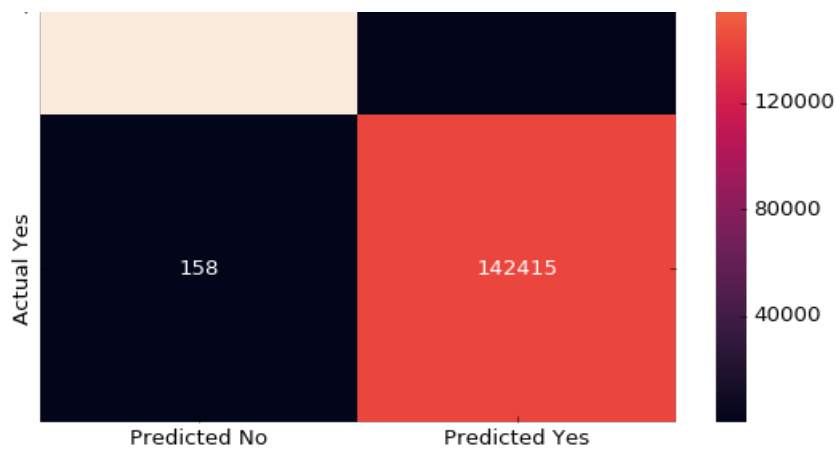
accuracy on test : 0.979691656262

In [225]:

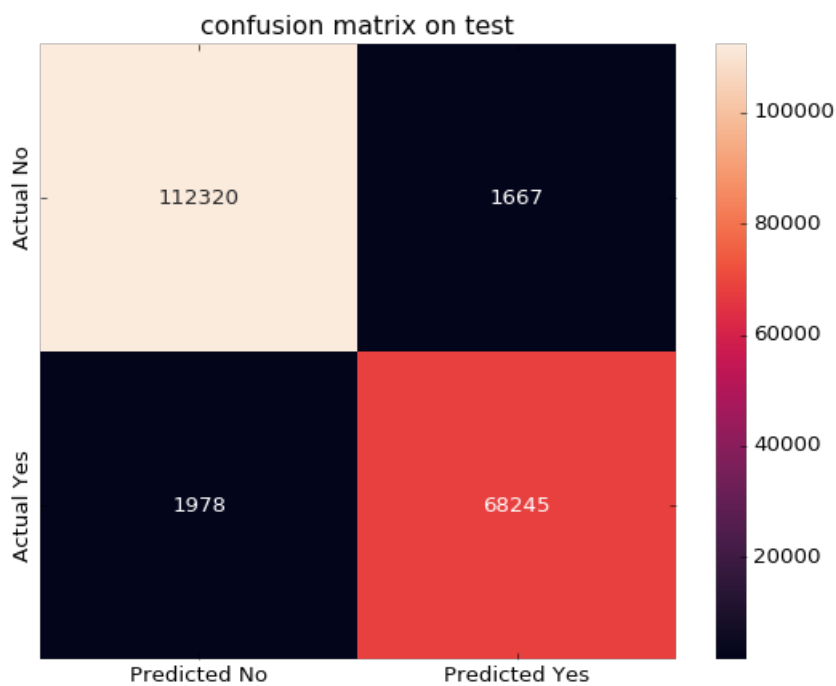
```
print(plot_confusion_matrix(dt_tuned,X_train,X_test,Y_train,Y_test))
```

best train threshold : 0.5





best test threshold : 0.75



None

In [187]:

```
## f1 score

f1_train,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)
```

Train F1 score is : 0.997754588032

Test F1 score is : 0.973989367396

In [205]:

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

important features :

feature	weight
ProviderCount	0.68459

ProviderCount	0.00199
State	0.11772
AttendingPhysicianCount	0.078
County	0.04629
DiagnosisGroupCode_count	0.03782
DiagnosisCode_1_count	0.00374
OPAnnualDeductibleAmt	0.00313
OPAnnualReimbursementAmt	0.00283
ip_op_total_amount	0.00239
InscClaimAmtReimbursed	0.00216
BeneCount	0.00191
DiagnosisCode_2_count	0.00182
num_of_chronic	0.00153
ClmAdmitDiagnosisCode_count	0.00124
DiagnosisCode_3_count	0.00124
IPAnnualReimbursementAmt	0.00117
DiagnosisCode_4_count	0.00089
num_of_diag_proc	0.00085
num_of_phy	0.00084
DiagnosisCode_5_count	0.0008

6.2.2. model with null filled with mode

In [337]:

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

```
best parameter : {'max_depth': 1000, 'min_samples_split': 5, 'criterion': 'entropy'}
accuracy on train : 0.977853000393
```

In [340]:

```
## on best parameter

dt_tuned = DecisionTreeClassifier(max_depth= 1000,min_samples_split= 5,criterion= 'entropy')
dt_tuned.fit(X_train,Y_train)
print('accuracy on test :',dt_tuned.score(X_test,Y_test))
```

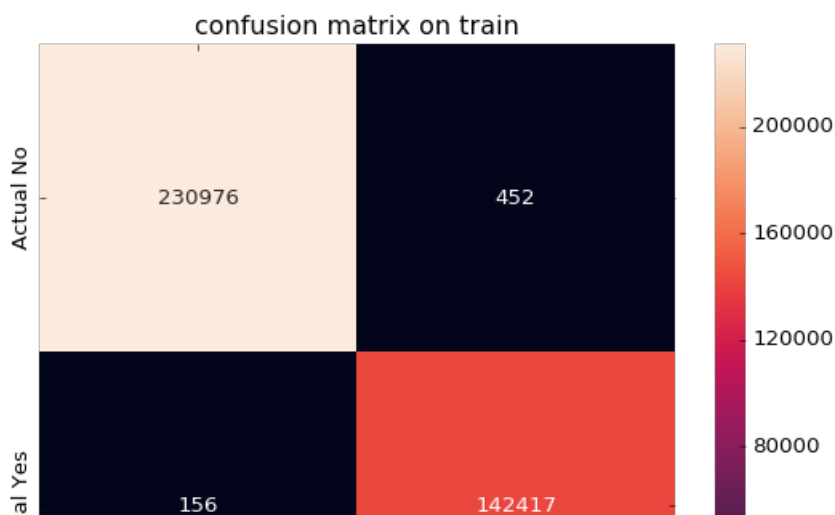
```
accuracy on test : 0.979697084849
```

In [341]:

```
## confusion matrix

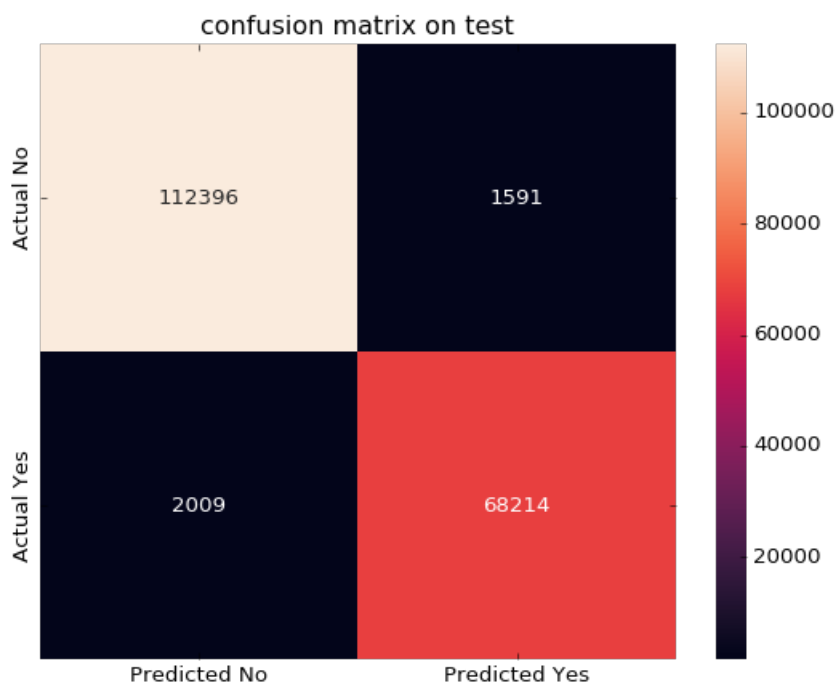
print(plot_confusion_matrix(dt_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.5
```





best test threshold : 1.0



None

In [343]:

```
## f1 score
f1_train,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)

## important features
print(important_feature(dt_tuned,X))
```

Train F1 score is : 0.997869970081

Test F1 score is : 0.974290856114

important features :

feature	weight
ProviderCount	0.68323
State	0.12273
AttendingPhysicianCount	0.08279
County	0.0389
DiagnosisGroupCode_count	0.03829
DiagnosisCode_1_count	0.00315
OPAnnualDeductibleAmt	0.00306
OPAnnualReimbursementAmt	0.00269
ip_op_total_amount	0.00248
InscClaimAmtReimbursed	0.00217
DiagnosisCode_2_count	0.00203
BeneCount	0.00184
num_of_chronic	0.00151

DiagnosisCode_3_count	0.00129
DiagnosisCode_4_count	0.00108
ClmAdmitDiagnosisCode_count	0.00086
IPAnnualReimbursementAmt	0.00085
DiagnosisCode_5_count	0.00082
DiagnosisCode_6_count	0.00066
numOfDaysForClaim	0.00065

+-----+-----+

6.3. RandomForestClassifier

In [344]:

```
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: 6.1min
[Parallel(n_jobs=-1)]: Done 50 out of 50 | elapsed: 8.5min finished
```

Out[344]:

```
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={'max_depth': [2, 4, 6, 8, 10, 12], 'min_samples_split': [5, 10, 50,
    100, 200, 250], 'criterion': ['gini', 'entropy'], 'n_estimators': [10, 50, 100, 500, 1000]},
    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 [192]:

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

```
best parameter : {'max_depth': 10, 'min_samples_split': 5, 'criterion': 'gini', 'n_estimators': 10
0}
accuracy on train : 0.852655474183
```

In [193]:

```
## on best parameter

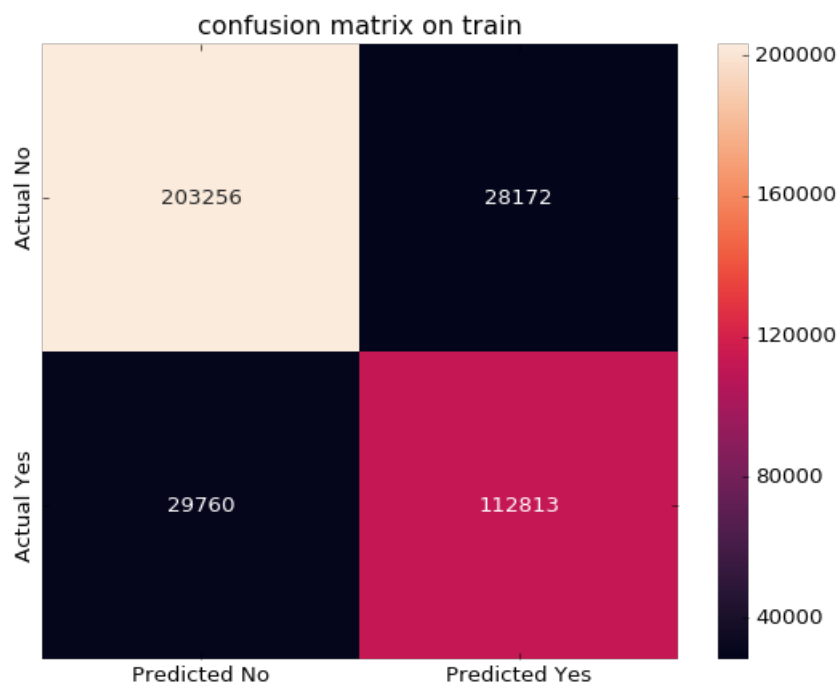
rf_tuned = RandomForestClassifier(max_depth=10,min_samples_split=5,criterion='gini',n_estimators=1
00,n_jobs=-1)
rf_tuned.fit(X_train,Y_train)
print('accuracy on test :',rf_tuned.score(X_test,Y_test))
```

```
accuracy on test : 0.84887356821
```

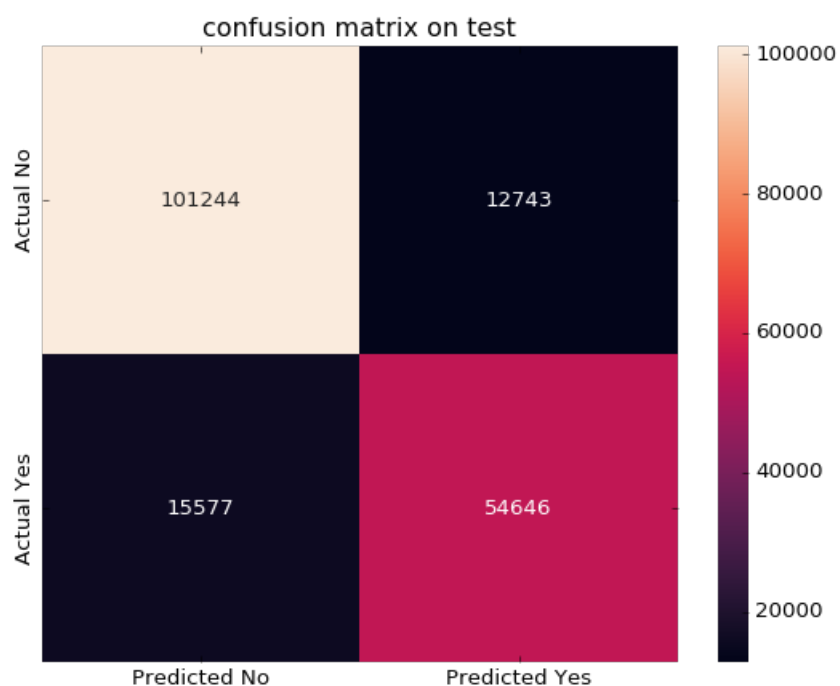
In [196]:

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

best train threshold : 0.274097610379



best test threshold : 0.278399342874



None

In [201]:

```
f1_train,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)
```

Train F1 score is : 0.795696118607

```
Test F1 score is : 0.794203993838
```

In [207]:

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

important features :

feature	weight
ProviderCount	0.72361
AttendingPhysicianCount	0.1591
State	0.03035
County	0.02055
numOfDaysAdmitted	0.01073
DiagnosisGroupCode_count	0.01043
DeductibleAmtPaid	0.00656
InscClaimAmtReimbursed	0.00593
num_of_diag_proc	0.00467
ClmAdmitDiagnosisCode_count	0.00253
DiagnosisCode_9_count	0.00219
BeneCount	0.00186
numOfDaysForClaim	0.0018
IPAnnualReimbursementAmt	0.00166
DiagnosisCode_8_count	0.00154
ip_op_total_amount	0.00151
DiagnosisCode_7_count	0.00146
OPAnnualReimbursementAmt	0.00138
OPAnnualDeductibleAmt	0.00135
DiagnosisCode_6_count	0.00108

6.3.2. model with null filled with mode

In [345]:

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

```
best parameter : {'max_depth': 10, 'min_samples_split': 10, 'criterion': 'gini', 'n_estimators': 100}
accuracy on train : 0.852294512582
```

In [346]:

```
## on best parameter

rf_tuned = RandomForestClassifier(max_depth=10,min_samples_split=10,criterion='gini',n_estimators=100,n_jobs=-1)
rf_tuned.fit(X_train,Y_train)
print('accuracy on test :',rf_tuned.score(X_test,Y_test))
```

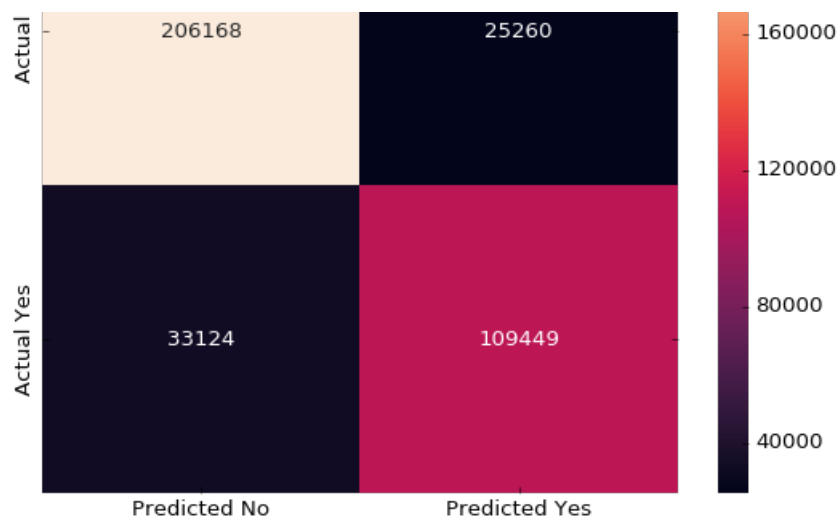
```
accuracy on test : 0.846436132675
```

In [347]:

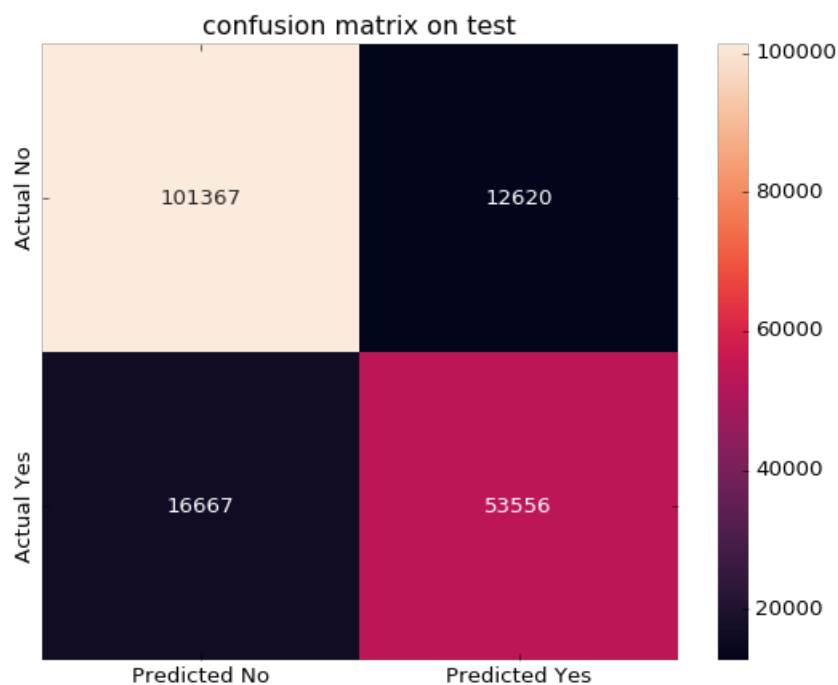
```
print(plot_confusion_matrix(rf_tuned,X_train,X_test,Y_train,Y_test))
```

```
best train threshold : 0.277243235089
```





best test threshold : 0.277015458408



None

In [348]:

```
f1_train,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(important_feature(rf_tuned,X))
```

Train F1 score is : 0.789441795717

Test F1 score is : 0.785284349592

important features :

feature	weight
ProviderCount	0.72267
AttendingPhysicianCount	0.17028
State	0.02339
County	0.01955
DiagnosisGroupCode_count	0.01202

```

| DiagnosisGroupCode_count | 0.01202 |
| DeductibleAmtPaid | 0.01079 |
| numOfDayAdmitted | 0.01031 |
| InscClaimAmtReimbursed | 0.00505 |
| procedure_9904.0 | 0.00322 |
| numOfDayForClaim | 0.00169 |
| DiagnosisCode_9_count | 0.00161 |
| BeneCount | 0.00156 |
| DiagnosisCode_8_count | 0.00153 |
| Race | 0.0013 |
| ip_op_total_amount | 0.00124 |
| OPAnnualReimbursementAmt | 0.0012 |
| ClmAdmitDiagnosisCode_count | 0.00108 |
| OPAnnualDeductibleAmt | 0.00108 |
| IPAnnualReimbursementAmt | 0.00105 |
| DiagnosisCode_7_count | 0.00102 |
+-----+-----+

```

6.4. XGBClassifier

In [349]:

```

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: 12.9min
[Parallel(n_jobs=-1)]: Done 9 tasks | elapsed: 28.1min
[Parallel(n_jobs=-1)]: Done 16 tasks | elapsed: 29.5min
[Parallel(n_jobs=-1)]: Done 25 tasks | elapsed: 31.5min
[Parallel(n_jobs=-1)]: Done 34 tasks | elapsed: 33.7min
[Parallel(n_jobs=-1)]: Done 41 out of 50 | elapsed: 54.5min remaining: 12.0min
[Parallel(n_jobs=-1)]: Done 47 out of 50 | elapsed: 55.9min remaining: 3.6min
[Parallel(n_jobs=-1)]: Done 50 out of 50 | elapsed: 69.1min finished

```

Out[349]:

```

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={'colsample_bytree': [0.1, 0.3, 0.5, 1], 'max_depth': [3, 5, 10], 'su
bsample': [0.1, 0.3, 0.5, 1], 'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators':
[100, 200, 500, 1000]},
    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 [214]:

```

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

```

```

best parameter : {'subsample': 1, 'max_depth': 10, 'learning_rate': 0.1, 'colsample_bytree': 0.5,
'n_estimators': 1000}
accuracy on train : 0.985342285181

```

In [215]:

```
## on best parameter
```

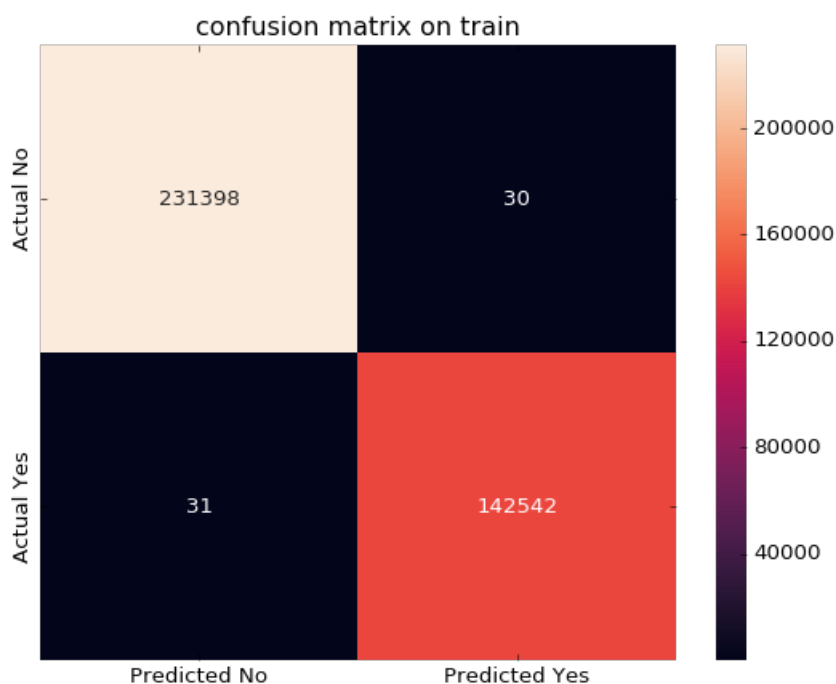
```
xgb_tuned = XGBClassifier(subsample=1,n_estimators= 1000,max_depth=
10,colsample_bytree=0.5,learning_rate= 0.1,n_jobs=-1)
xgb_tuned.fit(X_train,Y_train)
print('accuracy on test :',xgb_tuned.score(X_test,Y_test))
```

accuracy on test : 0.987476249932

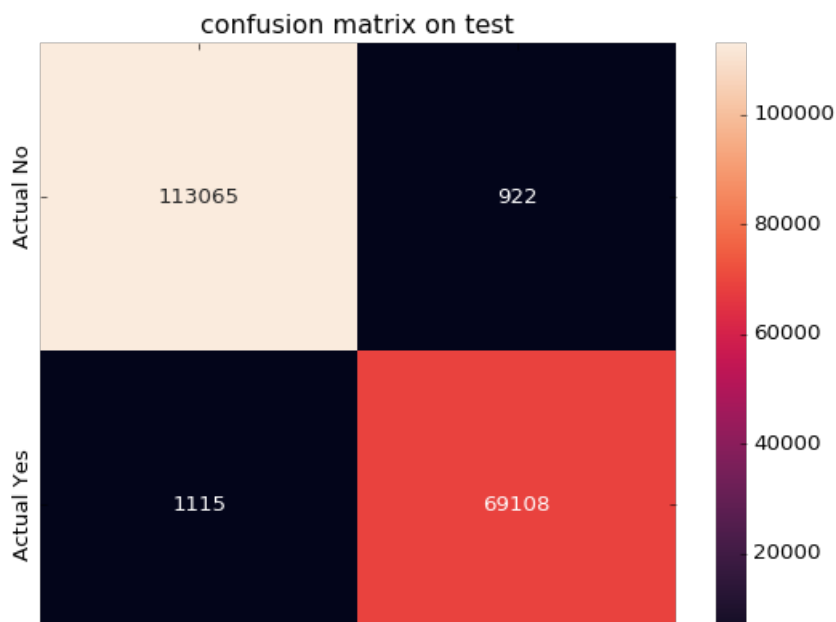
In [218]:

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

best train threshold : 0.381103



best test threshold : 0.258226



Predicted No

Predicted Yes

None

In [217]:

```
## f1 score

f1_train,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(important_feature(xgb_tuned,X))
```

Train F1 score is : 0.999786073752

Test F1 score is : 0.985476246497

important features :

feature	weight
ProviderCount	0.11826
State	0.09079
County	0.09005
AttendingPhysicianCount	0.08275
DiagnosisCode_1_count	0.05865
ip_op_total_amount	0.05526
OPAnnualReimbursementAmt	0.05241
OPAnnualDeductibleAmt	0.04644
InscClaimAmtReimbursed	0.04009
DiagnosisCode_2_count	0.03756
BeneCount	0.02869
DiagnosisCode_3_count	0.02824
IPAnnualReimbursementAmt	0.02136
num_of_chronic	0.02022
DiagnosisCode_4_count	0.01996
ClmAdmitDiagnosisCode_count	0.01996
num_of_diag_proc	0.01582
DiagnosisCode_5_count	0.01354
DiagnosisCode_6_count	0.01174
DiagnosisGroupCode_count	0.01019

6.4.2. model with null filled with mode

In [350]:

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

best parameter : {'subsample': 1, 'max_depth': 10, 'learning_rate': 0.15, 'colsample_bytree': 0.5, 'n_estimators': 1000}
accuracy on train : 0.986390410721

In [351]:

```
## on best parameter

xgb_tuned = XGBClassifier(subsample=1,n_estimators= 1000,max_depth=
10,colsample_bytree=0.5,learning_rate= 0.15,n_jobs=-1)
xgb_tuned.fit(X_train,Y_train)
print('accuracy on test :',xgb_tuned.score(X_test,Y_test))
```

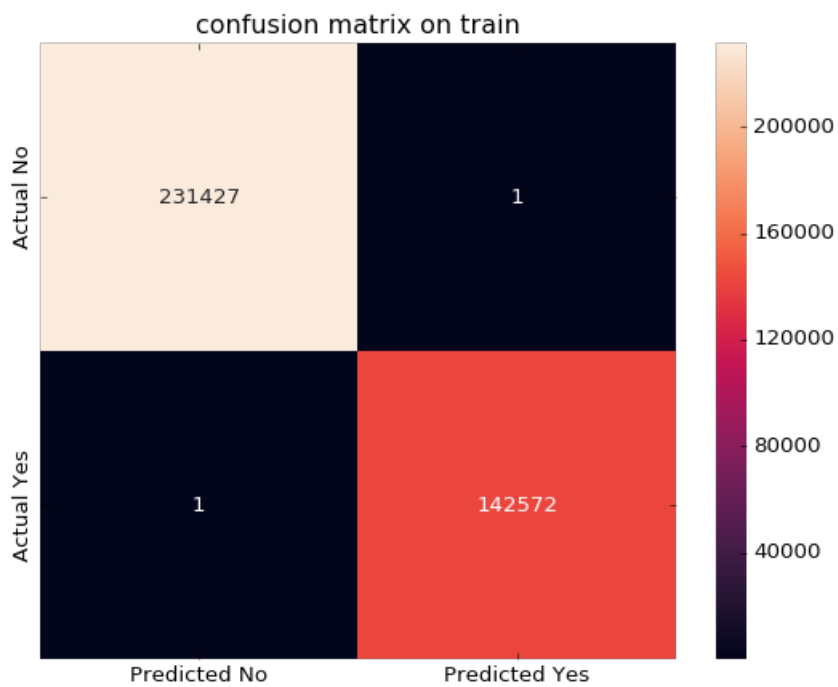
accuracy on test : 0.987742250692

In [352]:

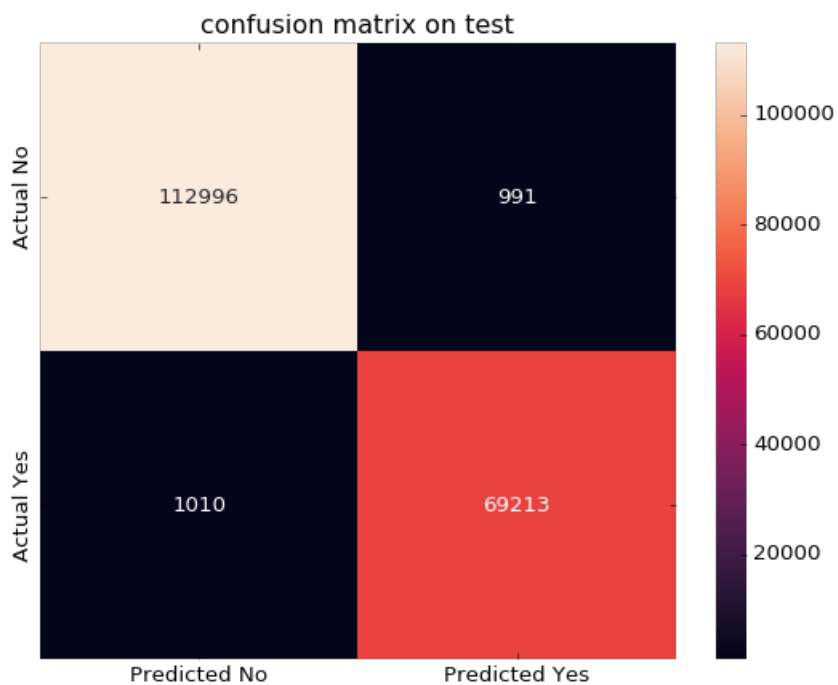
```
print(plot_confusion_matrix(xgb_tuned,X_train,X_test,Y_train,Y_test))
```

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

best train threshold : 0.413559



best test threshold : 0.21734



None

In [353]:

```
f1_train,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(important_feature(xgb_tuned,X))
```

Train F1 score is : 0.999992986049

Test F1 score is : 0.985750603516

important features :

feature	weight
ProviderCount	0.12038
County	0.08972
State	0.08705
AttendingPhysicianCount	0.08052
DiagnosisCode_1_count	0.06137
ip_op_total_amount	0.05593
OPAnnualReimbursementAmt	0.05467
OPAnnualDeductibleAmt	0.04869
InscClaimAmtReimbursed	0.04277
DiagnosisCode_2_count	0.04007
DiagnosisCode_3_count	0.0308
BeneCount	0.0295
IPAnnualReimbursementAmt	0.0223
ClmAdmitDiagnosisCode_count	0.02121
DiagnosisCode_4_count	0.02063
num_of_chronic	0.0205
DiagnosisCode_5_count	0.01257
DiagnosisCode_6_count	0.01222
DiagnosisGroupCode_count	0.00951
num_of_phy	0.00921

In [223]:

```
table = PrettyTable(['model ','best param','accuracy','f1 score'])
table.add_row(['LR',' { C: 0.01, penalty : 12 } ','0.830921231204','0.745294738076'])
table.add_row(['DT',' { max_depth: 50, min_samples_split: 5, criterion: entropy}
','0.979691656262','0.973989367396'])
table.add_row(['RF',' {max_depth:10,n_estimators:100,criterion:gini,min_samples_split:5}','0.848873
56821','0.794203993838'])
table.add_row(['xgb',' {subsampl:1,n_estimators:1000,max_depth:10,colsamp_bytree:0.5,learning_rate:(
','0.98747624','0.98547624')'])
print(table)
```

model	best param	accuracy	f1 score
LR	{ C: 0.01, penalty : 12 }	0.830921231204	0.745294738076
DT	{ max_depth: 50, min_samples_split: 5, criterion: entropy}	0.979691656262	0.973989367396
RF	{max_depth:10,n_estimators:100,criterion:gini,min_samples_split:5}	0.84887356821	0.794203993838
xgb	{subsampl:1,n_estimators:1000,max_depth:10,colsamp_bytree:0.5,learning_rate:0.1}	0.98747624	0.98547624

In [356]:

```
## mode
table = PrettyTable(['model ','best param','accuracy','f1 score'])
table.add_row(['LR',' { C: 0.01, penalty : 11 } ','0.821035774388','0.742860554142'])
table.add_row(['DT',' { max_depth: 1000, min_samples_split: 5, criterion: entropy}
','0.979697084849','0.974290856114'])
table.add_row(['RF',' {max_depth:10,n_estimators:100,criterion:gini,min_samples_split:10}','0.84643
6132675','0.785284349592'])
table.add_row(['xgb',' {subsamp:1,n_estimators:1000,max_depth:10,colsamp_bytree:0.5,learning_rate:0.
','0.9877422','0.9857506']])
print(table)
```

