Libraries and Data Load

Load Libraries

In [1]:

```
#####################################################Initial Packages################
#Basic Operating System Stuff
import os
import gc #garbage collector
import random #random seed generator
import pandas_profiling # requires import and prior install

#Timer
from timeit import default_timer as timer #import a timer

#Basic dataframe, array, and math stuff
import pandas as pd #data frame
from pandas_profiling import ProfileReport
import math #math functions
import numpy as np    #numerical package
from patsy import dmatrix, demo_data, ContrastMatrix, Poly

#Scikit learn
from math import sqrt
import sklearn as sk  #scikit learn
import sklearn.linear_model
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV,Ridge, Lasso,
from sklearn.kernel_ridge import KernelRidge
from sklearn.utils import resample #sampling
from sklearn.model_selection import train_test_split as tts #train test split
from sklearn.decomposition import PCA #principal components
from imblearn.over_sampling import SMOTE #synthetic minority oversampling technique
from sklearn.metrics import confusion_matrix #for 2-class model
from sklearn.metrics import roc_curve #for 2-class model
from sklearn.metrics import plot_confusion_matrix
from scipy import misc #Lots of stuff here
from scipy import stats as st
import itertools
from sklearn.metrics import mean_squared_error, r2_score, plot_confusion_matrix # evalu
from sklearn.preprocessing import StandardScaler # used for variable scaling data
from sklearn.preprocessing import MinMaxScaler as Scaler # used for variable scaling da
from sklearn.preprocessing import PolynomialFeatures as poly #used for interactions
from sklearn.tree import DecisionTreeClassifier as Tree
from sklearn.ensemble import RandomForestClassifier # Random Forest package
from sklearn.ensemble import ExtraTreesClassifier # Extra Trees package
from sklearn.ensemble import GradientBoostingClassifier # Gradient Boosting package
from sklearn.ensemble import AdaBoostClassifier as ADA # Gradient Boosting package
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier as SGD
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report as CR
from sklearn.pipeline import make_pipeline
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
```

```
from sklearn.metrics import average_precision_score

import statsmodels.api as sm
import pyreadstat

#Tensorflow
import tensorflow as tf #backend for keras
from tensorflow.python.client import device_lib #to  see if my GPU is alive!
import tensorflow.keras #keras
from tensorflow.keras.utils import to_categorical #convert categorical to dichotomous
from tensorflow.keras import Sequential,Input,Model #pull in the sequential, input laye
from tensorflow.keras import layers #If I were building a sequential model
from tensorflow.keras.layers import Dense, Dropout, Flatten #pull in the dense, dropout
from tensorflow.keras.layers import Input, Add, Activation, ZeroPadding2D, BatchNormali
from tensorflow.keras.layers import Conv2D, AveragePooling2D, MaxPooling2D, GlobalMaxPo
from tensorflow.keras.layers import BatchNormalization #batch normalization
from tensorflow.keras.layers import LeakyReLU #pull in leakly relu layer
from tensorflow.keras.preprocessing.image import ImageDataGenerator #use for generating
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau #use for early
from tensorflow.keras.models import Model, load_model  #Can't do much without a model
from tensorflow.keras.preprocessing import image  #Just for processing images
from tensorflow.keras import utils #Need utilities for the layers
from tensorflow.keras.utils import get_file #To load certain files
from tensorflow.keras.applications.imagenet_utils import preprocess_input  #Yo'...this
from tensorflow.keras.utils import model_to_dot #Allows plotting of the model
from tensorflow.keras.utils import plot_model #Allows plotting of the model
from tensorflow.keras.initializers import glorot_uniform #to initialize random weights
import tensorflow.keras.backend as K #let's write our own metrics and loss functions
import xgboost
from xgboost import XGBClassifier


#Graphing
import seaborn as sns
import pydot  #For model plotting
import graphviz #python-graphviz package
from IPython.display import SVG #Same here
import matplotlib.pyplot as plt #plotting
import matplotlib #image save
from matplotlib.pyplot import imshow #Show images
from PIL import Image #Another image utility
import cv2 #more image utilities

%matplotlib inline

print(device_lib.list_local_devices()) #Let's see if Python recognizes my GPU, shall we

os.chdir('D:\MI')
#####################################################################################
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 14109763605282888603
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
```

```
incarnation: 8033301467293071779
physical_device_desc: "device: XLA_CPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6920575392
locality {
  bus_id: 1
  links {
  }
}
incarnation: 16221899654740349888
physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 2080 Super, pci bus id: 0000:
01:00.0, compute capability: 7.5"
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 1289657470697671753
physical_device_desc: "device: XLA_GPU device"
]
```

Load Function to Reset GPU

In [2]:

```
##################################################Memory Mgt / Directory Load###########
def reset_keras():
    tensorflow.keras.backend.clear_session
reset_keras()

def store_CPU(x):
    with tf.device('/CPU:0'):
        x=x
        reset_keras()
        gc.collect()
#######################################################################################
```

Load Data

In [3]:

```
#######################################################################################
mydata=pd.read_sas('D:/MI/Y2019.XPT')
#######################################################################################
```

Data Preparation

Determine Shape

In [4]:

```
mydata.shape
```

Out[4]:

```
(418268, 342)
```

Reduce Variable Set and Scope

In [5]:
```
temp1=mydata
temp1=temp1.loc[temp1['_AGE_G']>4]
temp1=temp1[['CVDINFR4',
             '_AGE_G', '_IMPRACE', 'SEXVAR', 'MARITAL', 'VETERAN3',
             'INCOME2','EDUCA', 'EMPLOY1','RENTHOM1',
             'GENHLTH', 'SMOKE100', 'USENOW3','_PA300R3', 'ALCDAY5',
             'TOLDHI2', '_RFHYPE5', '_RFBMI5',
             'DIABETE4', 'PHYSHLTH', 'MENTHLTH',  'ADDEPEV3','CVDSTRK3', 'ASTHMA3',
             'CHCSCNCR','CHCOCNCR', 'CHCCOPD2','CHCKDNY2','HAVARTH4',
             'HLTHPLN1', 'PERSDOC2', 'MEDCOST', 'CHECKUP1',
             '_STATE', '_LLCPWT', '_STSTR']]
temp1.shape
```

Out[5]:
```
(238719, 36)
```

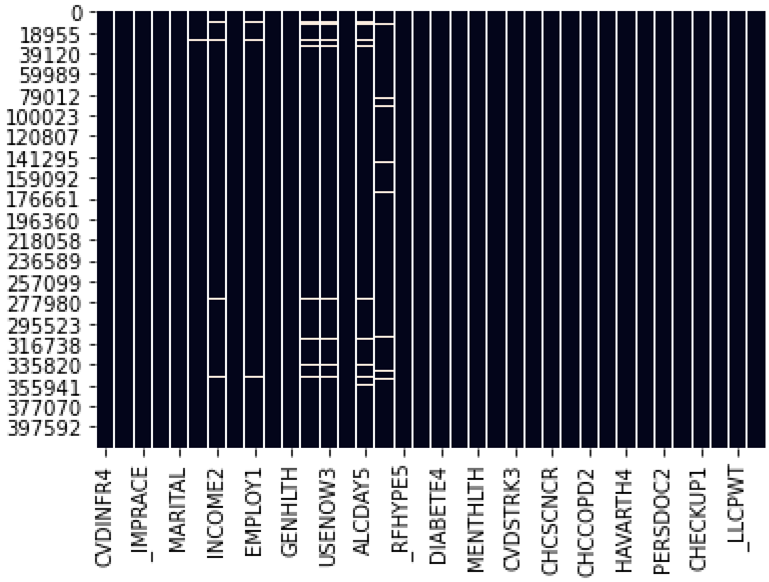
Handle Missing

Rows missing 20% or more are eliminated (8 or more variables, 9 total observations eliminated). Columns missing 20% or more are eliminated (47,744 observations, 0 eliminated).

In [6]:
```
a=temp1.isnull().sum() #count the nulls by column
print(a.sort_values(ascending=False).head(10))

z=[]
ct=[]

###Previously Run, Identified those rows with  8 or more missing variables.  These 9 ob
#for i in range(len(temp1.index)):
#    z.append(temp1.iloc[i].isnull().sum())
#    ct.append(i)
#d={"Index": ct, "Counts": z}
#mydf=pd.DataFrame(d).sort_values(by=['Counts'], ascending=False)
#print(mydf.head(10)) #none are missing 20% or more.

######Identified the following rows for dropping#########
temp1 = temp1.drop([temp1.index[33567] , temp1.index[33159],temp1.index[33557], temp1.i
                    temp1.index[193359], temp1.index[2398], temp1.index[187826], temp1.

sns.heatmap(temp1.isnull(), cbar=False)
```

```
ALCDAY5     9003
USENOW3     8267
SMOKE100    7881
TOLDHI2     4651
INCOME2     3134
EMPLOY1     1374
VETERAN3     732
MARITAL       26
PHYSHLTH      18
GENHLTH       17
dtype: int64
```

Out[6]:
```
<AxesSubplot:>
```



Impute Median

Given the small number of missing values remaining, impute median.

In [7]:

```
num=temp1.isna().sum().sum()
den=temp1.shape[0]*temp1.shape[1]
print('missing:', num/den)
temp1=temp1.fillna(temp1.median())
print(temp1.shape)
```

```
missing: 0.004082126615744814
(238710, 36)
```

Rename Columns

In [8]:

```
mydict={'CVDINFR4':'MI','_AGE_G':'Age65', '_IMPRACE':'Race', 'SEXVAR':'Male', 'MARITAL'
             'INCOME2':'Income','EDUCA':'LowEducation', 'EMPLOY1':'Unemployed', 'RENTHO
             'GENHLTH':'PoorHealth', 'SMOKE100':'Smoker','USENOW3':'ChewSnuff',
             '_PA300R3':'PoorExercise', 'ALCDAY5':'DrinksDaily',
            'TOLDHI2':'HighCholesterol', '_RFHYPE5':'HighBP', '_RFBMI5':'HighBMI',
             'DIABETE4':'Diabetes', 'PHYSHLTH':'PhysicalHealth','MENTHLTH':'MentalHealt
             'ADDEPEV3':'Depression','CVDSTRK3':'Stroke',
            'ASTHMA3':'Asthma','CHCSCNCR':'SkinCancer', 'CHCOCNCR':'Cancer',
            'CHCCOPD2':'COPD','CHCKDNY2':'Kidney','HAVARTH4':'Arthritis',
             'HLTHPLN1':'NoHealthPlan', 'PERSDOC2':'NoDoctor', 'MEDCOST':'Cost', 'CHECK
             '_STATE':'State', '_LLCPWT':'Weights', '_STSTR':'Stratum'}
temp1=temp1.rename(columns=dict(mydict))
```

Recodes

MI, 1=YES

Has a doctor, nurse, or other health professional ever told you that you had any of the following? Heart Attack 1=Yes, 2=No, 7=Don't Know, 9=Refused

In [9]:

```
print(temp1['MI'].value_counts()/len(temp1['MI']))
a_dict = {2:0, 7:0, 9:0}  #Modal response impute
temp1['MI']=temp1['MI'].replace(dict(a_dict))
print(temp1['MI'].value_counts()/len(temp1['MI']))
```

```
2.0    0.903691
1.0    0.089112
7.0    0.006690
9.0    0.000507
Name: MI, dtype: float64
0.0    0.910888
1.0    0.089112
Name: MI, dtype: float64
```

Age, 1=65+

In [10]:

```
print(temp1['Age65'].value_counts()/len(temp1['Age65']))
b_dict={5:0, 6:1}
temp1['Age65']=temp1['Age65'].replace(dict(b_dict))
print(temp1['Age65'].value_counts()/len(temp1['Age65']))
```

```
6.0    0.648042
5.0    0.351958
Name: Age65, dtype: float64
1.0    0.648042
0.0    0.351958
Name: Age65, dtype: float64
```

Race Recode: Black, Hispanic, Other with 1=Minority

Imputed race/ethnicity value

In [11]:

```
print(temp1['Race'].value_counts()/len(temp1['Race']))
c_dict={2:1, 1:0, 3:0, 4:0, 5:0, 6:0}
d_dict={5:1, 1:0, 2:0, 3:0, 4:0, 6:0}
e_dict={1:0, 2:0, 3:1, 4:1, 5:0, 6:1}
temp1['Black']=temp1['Race'].replace(dict(c_dict))
temp1['Hispanic']=temp1['Race'].replace(dict(d_dict))
temp1['Other']=temp1['Race'].replace(dict(e_dict))
print(temp1['Black'].value_counts())
print(temp1['Hispanic'].value_counts())
print(temp1['Other'].value_counts())
temp1=temp1.drop(columns=['Race'])
```

```
1.0    0.829886
2.0    0.068422
5.0    0.048754
6.0    0.026413
4.0    0.013875
3.0    0.012651
Name: Race, dtype: float64
0.0    222377
1.0     16333
Name: Black, dtype: int64
```

```
0.0    227072
1.0     11638
Name: Hispanic, dtype: int64
0.0    226073
1.0     12637
Name: Other, dtype: int64
```

Gender, 1=Male

Calculated sex variable

In [12]:

```
print(temp1['Male'].value_counts()/len(temp1['Male']))
f_dict={2:0, 1:1}
temp1['Male']=temp1['Male'].replace(dict(f_dict))
print(temp1['Male'].value_counts()/len(temp1['Male']))
```

```
2.0    0.569746
1.0    0.430254
Name: Male, dtype: float64
0.0    0.569746
1.0    0.430254
Name: Male, dtype: float64
```

Unmarried

1=Married, 2=Divorced, 3=Widowed, 4=Separated, 5=Never Married, 6=Member of Unmarried Couple, 9=Refused

In [13]:

```
print(temp1['Unmarried'].value_counts()/len(temp1['Unmarried']))
g_dict={1:0,2:1, 3:1, 4:1, 5:1, 6:1, 9:1}
temp1['Unmarried']=temp1['Unmarried'].replace(dict(g_dict))
print(temp1['Unmarried'].value_counts()/len(temp1['Unmarried']))
```

```
1.0    0.529261
3.0    0.202895
2.0    0.158510
5.0    0.072603
4.0    0.015257
6.0    0.014373
9.0    0.007101
Name: Unmarried, dtype: float64
0.0    0.529261
1.0    0.470739
Name: Unmarried, dtype: float64
```

Veteran, 1=Veteran

Have you ever served on active duty in the United States Armed Forces, either in the regular military or the National Guard or military reserve unit? 1=Yes, 2=No

In [14]:

```
print(temp1['Veteran'].value_counts()/len(temp1['Veteran']))
h_dict={1:1, 2:0, 7:0, 9:0}
temp1['Veteran']=temp1['Veteran'].replace(dict(h_dict))
print(temp1['Veteran'].value_counts()/len(temp1['Veteran']))
```

```
2.0    0.829902
1.0    0.168016
9.0    0.001692
7.0    0.000390
Name: Veteran, dtype: float64
0.0    0.831984
1.0    0.168016
Name: Veteran, dtype: float64
```

Poor Income

1 < 75K, 0 otherwise

In [15]:

```
mysc=Scaler()
print(temp1['Income'].value_counts()/len(temp1['Income']))
i_dict={1:1, 2:1, 3:1, 4:1, 5:1, 6:1, 7:1, 8:0, 77:1,99:1} #impute midpoints
temp1['Income']=temp1['Income'].replace(dict(i_dict))
temp1['Income']=mysc.fit_transform(temp1[['Income']])
print(temp1['Income'].value_counts()/len(temp1['Income']))
```

```
8.0     0.240162
7.0     0.142855
99.0    0.120896
6.0     0.117306
5.0     0.089263
77.0    0.076997
4.0     0.075657
3.0     0.059453
2.0     0.044112
1.0     0.033300
Name: Income, dtype: float64
1.0    0.759838
0.0    0.240162
Name: Income, dtype: float64
```

Education Recode: Below HS, HS, Post-HS but Not College Grad

In [16]:

```
print(temp1['LowEducation'].value_counts())
j_dict={1:1, 2:1, 3:1, 4:0, 5:0, 6:0, 9:1} #add 919 refused to this mix
k_dict={1:0, 2:0, 3:0, 4:1, 5:0, 6:0, 9:0}
l_dict={1:0, 2:0, 3:0, 4:0, 5:1, 6:0, 9:0} #not a college grad but post-HS
temp1['PreHS']=temp1['LowEducation'].replace(dict(j_dict))
temp1['HS']=temp1['LowEducation'].replace(dict(k_dict))
temp1['PostHS']=temp1['LowEducation'].replace(dict(l_dict))
print(temp1['PreHS'].value_counts())
print(temp1['HS'].value_counts())
print(temp1['PostHS'].value_counts())
temp1=temp1.drop(columns=['LowEducation'])
```

```
6.0    88686
4.0    66131
5.0    65861
3.0    10955
2.0     5828
9.0      917
1.0      332
Name: LowEducation, dtype: int64
```

```
0.0    220678
1.0     18032
Name: PreHS, dtype: int64
0.0    172579
1.0     66131
Name: HS, dtype: int64
0.0    172849
1.0     65861
Name: PostHS, dtype: int64
```

Not Employed

In [17]:

```
print(temp1['Unemployed'].value_counts())
m_dict={1:0, 2:0, 3:1, 4:1, 5:1, 6:1, 7:1, 8:1, 9:1}
temp1['Unemployed']=temp1['Unemployed'].replace(dict(m_dict))
print(temp1['Unemployed'].value_counts())
```

```
7.0    127944
1.0     54772
2.0     19278
8.0     19243
5.0      9913
3.0      3428
4.0      2260
9.0      1629
6.0       243
Name: Unemployed, dtype: int64
1.0    164660
0.0     74050
Name: Unemployed, dtype: int64
```

Rent Home

In [18]:

```
print(temp1['RentHome'].value_counts())
m2_dict={2:1, 1:0, 3:0, 7:0, 9:0}
temp1['RentHome']=temp1['RentHome'].replace(dict(m2_dict))
print(temp1['RentHome'].value_counts())
```

```
1.0    195703
2.0     34336
3.0      6903
9.0      1417
7.0       351
Name: RentHome, dtype: int64
0.0    204374
1.0     34336
Name: RentHome, dtype: int64
```

Poor Health

In [19]:

```
print(temp1['PoorHealth'].value_counts())
n_dict={1:0, 2:0, 3:0, 4:1, 5:1, 7:1, 9:1}
temp1['PoorHealth']=temp1['PoorHealth'].replace(dict(n_dict))
print(temp1['PoorHealth'].value_counts())
```

```
3.0    77172
2.0    74341
4.0    39162
```

```
1.0    31142
5.0    16213
7.0      498
9.0      182
Name: PoorHealth, dtype: int64
0.0    182655
1.0     56055
Name: PoorHealth, dtype: int64
```

Smoker

In [20]:

```
print(temp1['Smoker'].value_counts())
temp1['Smoker']=temp1['Smoker'].replace(dict(a_dict))
print(temp1['Smoker'].value_counts())
```

```
2.0    130599
1.0    106470
7.0      1345
9.0       296
Name: Smoker, dtype: int64
0.0    132240
1.0    106470
Name: Smoker, dtype: int64
```

Chew / Snuff

In [21]:

```
print(temp1['ChewSnuff'].value_counts())
n2_dict={1:1, 2:1, 3:0, 7:1, 9:1}
temp1['ChewSnuff']=temp1['ChewSnuff'].replace(dict(n2_dict))
print(temp1['ChewSnuff'].value_counts())
```

```
3.0    233112
1.0      3151
2.0      2060
9.0       291
7.0        96
Name: ChewSnuff, dtype: int64
0.0    233112
1.0      5598
Name: ChewSnuff, dtype: int64
```

Poor Exercise

In [22]:

```
print(temp1['PoorExercise'].value_counts())
o_dict={1:0,2:1,3:1, 9:1}
temp1['PoorExercise']=temp1['PoorExercise'].replace(dict(o_dict))
print(temp1['PoorExercise'].value_counts())
```

```
1.0    82791
3.0    69869
2.0    58892
9.0    27158
Name: PoorExercise, dtype: int64
1.0    155919
0.0     82791
Name: PoorExercise, dtype: int64
```

Drinks Daily

In [23]:

```
#print(temp1['DrinksDaily'].value_counts())
def my_recode(v):
    if v <=107:
        return (v-100)/7
    elif v<=230:
        return (v-200)/30
    else:
        return 0
temp1['DrinksDaily']= temp1['DrinksDaily'].apply(my_recode)
mysc=Scaler()
temp1['DrinksDaily']=mysc.fit_transform(temp1[['DrinksDaily']])
print(temp1['DrinksDaily'].value_counts())
```

```
0.000000    135640
0.033333     16494
1.000000     15567
0.066667     11443
0.142857      8135
0.100000      6236
0.285714      6025
0.133333      5243
0.166667      4713
0.428571      4276
0.666667      3963
0.333333      3820
0.500000      3153
0.714286      2242
0.571429      2077
0.200000      1853
0.833333      1834
0.266667      1372
0.233333      1048
0.857143       849
0.400000       790
0.933333       566
0.466667       238
0.966667       190
0.900000       134
0.800000       118
0.533333       109
0.600000       103
0.300000       101
0.700000        94
0.866667        90
0.733333        71
0.433333        31
0.766667        30
0.566667        28
0.366667        24
0.633333        10
Name: DrinksDaily, dtype: int64
```

High Cholesterol

In [24]:

```
print(temp1['HighCholesterol'].value_counts())
temp1['HighCholesterol']=temp1['HighCholesterol'].replace(dict(a_dict))
print(temp1['HighCholesterol'].value_counts())
```

```
2.0    124769
1.0    110918
7.0      2731
9.0       292
Name: HighCholesterol, dtype: int64
0.0    127792
1.0    110918
Name: HighCholesterol, dtype: int64
```

High BP

In [25]:

```
print(temp1['HighBP'].value_counts())
p_dict={1:0, 2:1, 9:0}  #2 = YES
temp1['HighBP']=temp1['HighBP'].replace(dict(p_dict))
print(temp1['HighBP'].value_counts())
```

```
2.0    130753
1.0    106988
9.0       969
Name: HighBP, dtype: int64
1.0    130753
0.0    107957
Name: HighBP, dtype: int64
```

High BMI

In [26]:

```
print(temp1['HighBMI'].value_counts())
temp1['HighBMI']=temp1['HighBMI'].replace(dict(p_dict))
print(temp1['HighBMI'].value_counts())
```

```
2.0    152414
1.0     67848
9.0     18448
Name: HighBMI, dtype: int64
1.0    152414
0.0     86296
Name: HighBMI, dtype: int64
```

Diabetes

In [27]:

```
print(temp1['Diabetes'].value_counts())
q_dict={1:1, 2:0, 3:0, 4:0, 7:0, 9:0}
temp1['Diabetes']=temp1['Diabetes'].replace(dict(q_dict))
print(temp1['Diabetes'].value_counts())
```

```
3.0    183618
1.0     46813
4.0      6594
2.0      1212
7.0       338
9.0       135
Name: Diabetes, dtype: int64
0.0    191897
1.0     46813
Name: Diabetes, dtype: int64
```

Physical

In [28]:
```
print(temp1['PhysicalHealth'].value_counts())
r_dict={77:0, 88:0, 99:0}
temp1['PhysicalHealth']=temp1['PhysicalHealth'].replace(dict(r_dict))
temp1['PhysicalHealth']=mysc.fit_transform(temp1[['PhysicalHealth']])
print(temp1['PhysicalHealth'].value_counts())
```

```
88.0    140587
30.0     24717
2.0      11792
1.0       8042
3.0       7334
5.0       7248
77.0      6135
10.0      5822
15.0      5467
7.0       4050
4.0       3807
20.0      3472
14.0      2457
25.0      1435
6.0       1209
99.0      1116
8.0        807
21.0       682
12.0       550
28.0       539
29.0       278
9.0        213
18.0       177
27.0       138
16.0       130
17.0        91
24.0        81
11.0        69
22.0        68
13.0        61
26.0        58
23.0        50
19.0        28
Name: PhysicalHealth, dtype: int64
0.000000    147838
1.000000     24717
0.066667     11792
0.033333      8042
0.100000      7334
0.166667      7248
0.333333      5822
0.500000      5467
0.233333      4050
0.133333      3807
0.666667      3472
0.466667      2457
0.833333      1435
0.200000      1209
0.266667       807
0.700000       682
0.400000       550
0.933333       539
0.966667       278
0.300000       213
0.600000       177
0.900000       138
0.533333       130
```

```
0.566667       91
0.800000       81
0.366667       69
0.733333       68
0.433333       61
0.866667       58
0.766667       50
0.633333       28
Name: PhysicalHealth, dtype: int64
```

Mental Health

In [29]:

```
print(temp1['MentalHealth'].value_counts())
temp1['MentalHealth']=temp1['MentalHealth'].replace(dict(r_dict))
temp1['MentalHealth']=mysc.fit_transform(temp1[['MentalHealth']])
print(temp1['MentalHealth'].value_counts())
```

```
88.0    171096
30.0     11568
2.0       9739
1.0       6565
5.0       6544
3.0       5416
10.0      5086
15.0      4737
77.0      4443
4.0       2764
20.0      2758
7.0       2017
99.0      1206
25.0      1044
14.0       863
6.0        825
8.0        506
12.0       296
28.0       287
21.0       197
29.0       190
9.0         91
18.0        77
16.0        71
27.0        66
22.0        48
17.0        46
24.0        36
26.0        33
13.0        30
23.0        28
11.0        26
19.0        11
Name: MentalHealth, dtype: int64
0.000000    176745
1.000000     11568
0.066667      9739
0.033333      6565
0.166667      6544
0.100000      5416
0.333333      5086
0.500000      4737
0.133333      2764
0.666667      2758
0.233333      2017
```

```
0.833333      1044
0.466667       863
0.200000       825
0.266667       506
0.400000       296
0.933333       287
0.700000       197
0.966667       190
0.300000        91
0.600000        77
0.533333        71
0.900000        66
0.733333        48
0.566667        46
0.800000        36
0.866667        33
0.433333        30
0.766667        28
0.366667        26
0.633333        11
Name: MentalHealth, dtype: int64
```

Depression

In [30]:

```
print(temp1['Depression'].value_counts())
temp1['Depression']=temp1['Depression'].replace(dict(a_dict))
print(temp1['Depression'].value_counts())
```

```
2.0    196764
1.0     40724
7.0       947
9.0       275
Name: Depression, dtype: int64
0.0    197986
1.0     40724
Name: Depression, dtype: int64
```

Stroke

In [31]:

```
print(temp1['Stroke'].value_counts())
temp1['Stroke']=temp1['Stroke'].replace(dict(a_dict))
print(temp1['Stroke'].value_counts())
```

```
2.0    221960
1.0     15855
7.0       781
9.0       114
Name: Stroke, dtype: int64
0.0    222855
1.0     15855
Name: Stroke, dtype: int64
```

Asthma

In [32]:

```
print(temp1['Asthma'].value_counts())
temp1['Asthma']=temp1['Asthma'].replace(dict(a_dict))
print(temp1['Asthma'].value_counts())
```

```
2.0    207304
1.0     30496
7.0       807
9.0       103
Name: Asthma, dtype: int64
0.0    208214
1.0     30496
Name: Asthma, dtype: int64
```

Skin Cancer

In [33]:

```
print(temp1['SkinCancer'].value_counts())
temp1['SkinCancer']=temp1['SkinCancer'].replace(dict(a_dict))
print(temp1['SkinCancer'].value_counts())
```

```
2.0    200073
1.0     37665
7.0       863
9.0       109
Name: SkinCancer, dtype: int64
0.0    201045
1.0     37665
Name: SkinCancer, dtype: int64
```

Cancer (Other than Skin)

In [34]:

```
print(temp1['Cancer'].value_counts())
temp1['Cancer']=temp1['Cancer'].replace(dict(a_dict))
print(temp1['Cancer'].value_counts())
```

```
2.0    201374
1.0     36567
7.0       583
9.0       186
Name: Cancer, dtype: int64
0.0    202143
1.0     36567
Name: Cancer, dtype: int64
```

COPD

In [35]:

```
print(temp1['COPD'].value_counts())
temp1['COPD']=temp1['COPD'].replace(dict(a_dict))
print(temp1['COPD'].value_counts())
```

```
2.0    208877
1.0     28267
7.0      1437
9.0       129
Name: COPD, dtype: int64
0.0    210443
1.0     28267
Name: COPD, dtype: int64
```

Kidney Disease

In [36]:

```
print(temp1['Kidney'].value_counts())
temp1['Kidney']=temp1['Kidney'].replace(dict(a_dict))
print(temp1['Kidney'].value_counts())
```

```
2.0    224218
1.0     13307
7.0      1059
9.0       126
Name: Kidney, dtype: int64
0.0    225403
1.0     13307
Name: Kidney, dtype: int64
```

Arthritis

In [37]:

```
print(temp1['Arthritis'].value_counts())
temp1['Arthritis']=temp1['Arthritis'].replace(dict(a_dict))
print(temp1['Arthritis'].value_counts())
```

```
2.0    125725
1.0    111392
7.0      1438
9.0       155
Name: Arthritis, dtype: int64
0.0    127318
1.0    111392
Name: Arthritis, dtype: int64
```

No Health Plan

In [38]:

```
print(temp1['NoHealthPlan'].value_counts())
s_dict={1:0, 2:1, 7:0, 9:0}
temp1['NoHealthPlan']=temp1['NoHealthPlan'].replace(dict(s_dict))
print(temp1['NoHealthPlan'].value_counts())
```

```
1.0    228154
2.0      9687
9.0       540
7.0       329
Name: NoHealthPlan, dtype: int64
0.0    229023
1.0      9687
Name: NoHealthPlan, dtype: int64
```

No Personal Doctor

In [39]:

```
print(temp1['NoDoctor'].value_counts())
t_dict={1:0, 2:0, 3:1, 7:1, 9:1}
temp1['NoDoctor']=temp1['NoDoctor'].replace(dict(t_dict))
print(temp1['NoDoctor'].value_counts())
```

```
1.0    196433
2.0     21390
3.0     19940
7.0       643
9.0       304
Name: NoDoctor, dtype: int64
```

```
0.0    217823
1.0     20887
Name: NoDoctor, dtype: int64
```

Medical Cost Prevented Care

In [40]:

```
print(temp1['Cost'].value_counts())
temp1['Cost']=temp1['Cost'].replace(dict(a_dict))
print(temp1['Cost'].value_counts())
```

```
2.0    221743
1.0     16261
7.0       557
9.0       149
Name: Cost, dtype: int64
0.0    222449
1.0     16261
Name: Cost, dtype: int64
```

No Checkup within Year

In [41]:

```
print(temp1['NoCheckup'].value_counts())
u_dict={1:0, 2:1, 3:1, 4:1, 7:1, 8:1, 9:1}
temp1['NoCheckup']=temp1['NoCheckup'].replace(dict(u_dict))
print(temp1['NoCheckup'].value_counts())
```

```
1.0    209464
2.0     14598
4.0      6118
3.0      5762
7.0      1790
8.0       764
9.0       214
Name: NoCheckup, dtype: int64
0.0    209464
1.0     29246
Name: NoCheckup, dtype: int64
```

State Recodes

In [42]:

```
v_dict={1:'AL', 2:'AK', 4:'AZ',   5:'AR', 6:'CA',    8:'CO',  9:'CT', 10:'DE', 11:'DC',
       12:'FL', 13:'GA', 15:'HI', 16:'ID', 17:'IL', 18:'IN', 19:'IA', 20:'KS', 21:'KY',
       23:'ME', 24:'MD', 25:'MA', 26:'MI', 27:'MN', 28:'MS', 29:'MO', 30:'MT', 31:'NE',
       33:'NH', 35:'NM', 36:'NY', 37:'NC', 38:'ND', 39:'OH', 40:'OK', 41:'OR', 42:'PA',
       46:'SD', 47:'TN', 48:'TX', 49:'UT', 50:'VT', 51:'VA', 53:'WA', 54:'WV', 55:'WI',
       72:'PR'}
temp1['State']=temp1['State'].replace(dict(v_dict))
print(temp1['State'].value_counts())
temp1['State']=temp1['State'].astype('str')
temp1=pd.get_dummies(temp1,columns=['State'])
temp1=temp1.drop(columns=['State_MD']) #modal response...drop for collinearity
```

```
MD    11268
FL    10218
NE     9191
OH     8735
NY     8013
```

```
ME      7945
MN      7806
WA      7340
TX      6751
KS      6330
MI      5990
CT      5762
VA      5603
AZ      5566
IN      5315
IA      5305
UT      5204
CO      5059
CA      5014
KY      4492
MO      4398
SC      4365
HI      4203
VT      4183
AL      4181
MA      4147
SD      3960
GA      3929
NH      3884
MT      3847
RI      3844
AR      3699
ND      3596
NM      3575
OK      3556
TN      3388
PA      3346
WV      3325
WY      3135
WI      3050
ID      2966
OR      2960
MS      2889
PR      2853
IL      2665
LA      2379
DE      2112
NC      1991
AK      1627
DC      1397
NV      1371
GU       982
Name: State, dtype: int64
```

Write CSV

In [43]:

```
temp1.to_csv('D:/MI/MI.csv', index=False)
```

Describe

Pandas Profiling Report is too large for GitHub.

In [44]:

```
##################################################################################
mydata=temp1
```

```
mydata.drop(columns=['Weights','Stratum'])
pd.set_option('display.max_columns', 500)
mydata.describe()
#pandas_profiling.ProfileReport(mydata)
#########################################################################################
```

Out[44]:

	MI	Age65	Male	Unmarried	Veteran	Income	Un
count	238710.000000	238710.000000	238710.000000	238710.000000	238710.000000	238710.000000	2387
mean	0.089112	0.648042	0.430254	0.470739	0.168016	0.759838	
std	0.284906	0.477582	0.495113	0.499144	0.373881	0.427182	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	
50%	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	
75%	0.000000	1.000000	1.000000	1.000000	0.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

Build Training and Test Set

In [45]:

```
#########################################################################################
# Seed value for random number generators to obtain reproducible results
temp=mydata
temp=temp.drop(columns=['Weights', 'Stratum'])
temp=temp.values
tempy=temp[:,0]
tempx=temp[:,1:len(temp)]

X_train, X_test, y_train, y_test = tts(tempx, tempy, test_size=.2, random_state=1234)
X_train1, X_test1, y_train1,y_test1=X_train, X_test, y_train, y_test #Backup, NOT overs

oversample = SMOTE()
X_train, y_train = oversample.fit_resample(X_train, y_train)

#########################################################################################
```

Tensorflow

Autoencoder-Not Used

In [46]:

```
tf.random.set_seed(64)

autoencoder = tf.keras.Sequential()
autoencoder.add(Flatten())
autoencoder.add(Dense(89,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(80,activation='relu', kernel_initializer='he_normal')) #H1
```

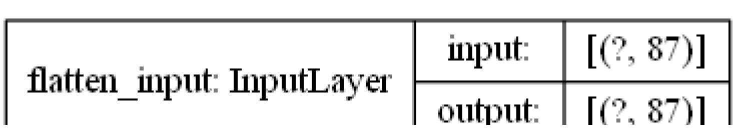
```
autoencoder.add(Dense(70,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(60,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(50,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(40,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(30,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(20,activation='relu', kernel_initializer='he_normal', name='bottl
autoencoder.add(Dense(20,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(30,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(40,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(50,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(60,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(70,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(80,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.add(Dense(87,activation='relu', kernel_initializer='he_normal')) #H1
autoencoder.compile(loss='mean_squared_error', optimizer = 'adam')
trained_model = autoencoder.fit(X_train, X_train, batch_size=1024, epochs=10, verbose=1
encoder = Model(autoencoder.input, autoencoder.get_layer('bottleneck').output)
encoded_train = encoder.predict(X_train)  # bottleneck representation
encoded_test = encoder.predict(X_test)  # bottleneck representation
decoded_train = autoencoder.predict(X_train)        # reconstructed training set
decoded_test=autoencoder.predict(X_test)            # applied to test set
encoding_dim = 20
```

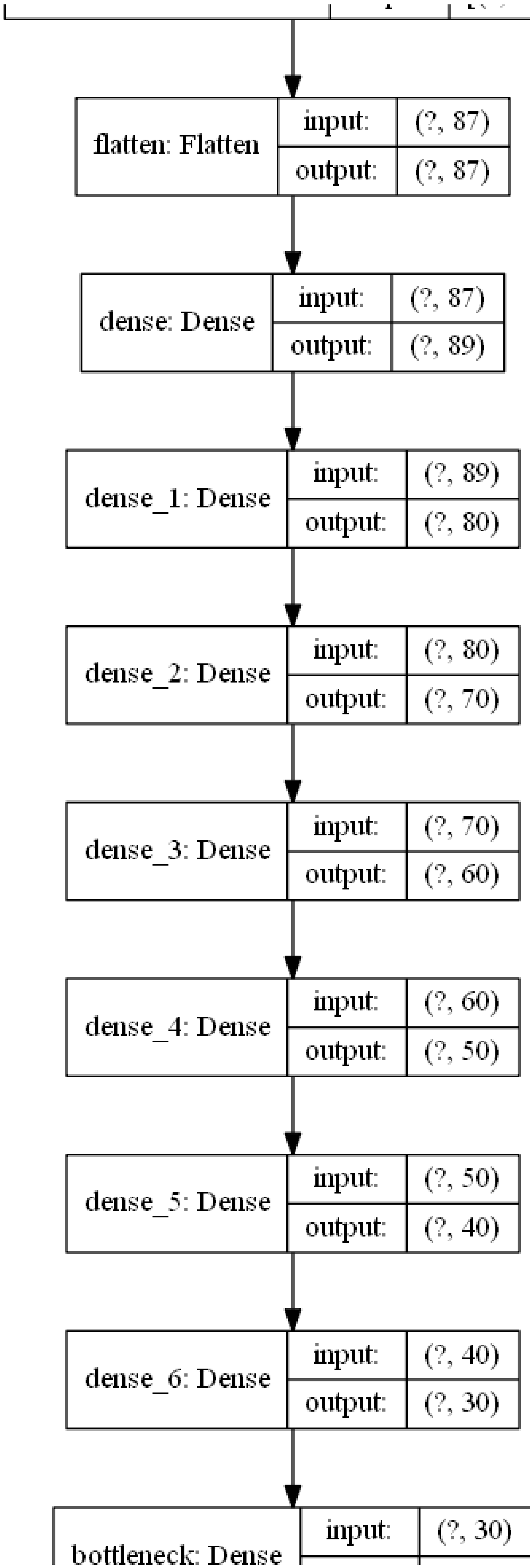
```
Epoch 1/10
272/272 [==============================] - 1s 3ms/step - loss: 0.0552 - val_loss: 0.0377
Epoch 2/10
272/272 [==============================] - 1s 2ms/step - loss: 0.0390 - val_loss: 0.0327
Epoch 3/10
272/272 [==============================] - 1s 2ms/step - loss: 0.0355 - val_loss: 0.0319
Epoch 4/10
272/272 [==============================] - 1s 3ms/step - loss: 0.0330 - val_loss: 0.0282
Epoch 5/10
272/272 [==============================] - 1s 3ms/step - loss: 0.0297 - val_loss: 0.0273
Epoch 6/10
272/272 [==============================] - 1s 2ms/step - loss: 0.0280 - val_loss: 0.0251
Epoch 7/10
272/272 [==============================] - 1s 2ms/step - loss: 0.0276 - val_loss: 0.0278
Epoch 8/10
272/272 [==============================] - 1s 2ms/step - loss: 0.0287 - val_loss: 0.0278
Epoch 9/10
272/272 [==============================] - 1s 2ms/step - loss: 0.0298 - val_loss: 0.0348
Epoch 10/10
272/272 [==============================] - 1s 2ms/step - loss: 0.0346 - val_loss: 0.0289
```

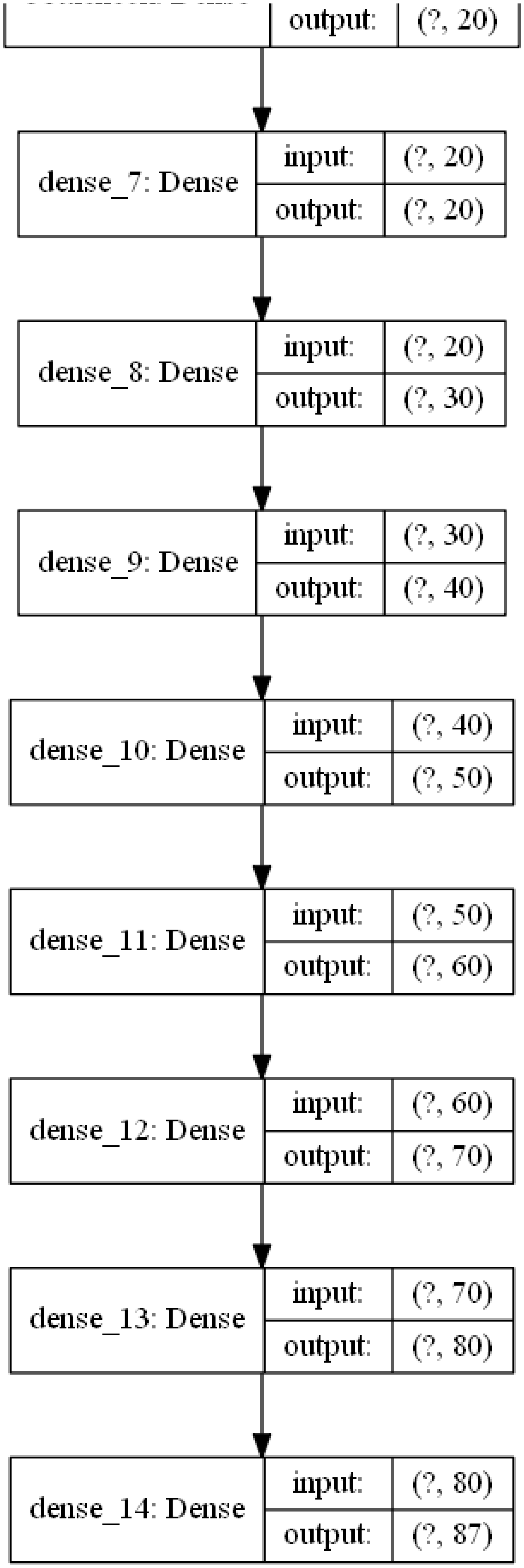
In [47]:

```
encoded_input = Input(shape=(encoding_dim,))
decoder = autoencoder.layers[-8](encoded_input) #subtract out layers
decoder = autoencoder.layers[-7](decoder)
decoder = autoencoder.layers[-6](decoder)
decoder = autoencoder.layers[-5](decoder)
decoder = autoencoder.layers[-4](decoder)
decoder = autoencoder.layers[-3](decoder)
decoder = autoencoder.layers[-2](decoder)
decoder = autoencoder.layers[-1](decoder)
decoder = Model(encoded_input, decoder)

dot_img_file = 'autoencoder.png'
tf.keras.utils.plot_model(autoencoder, to_file=dot_img_file, show_shapes=True)
```

Out[47]:




flatten: Flatten
input: (?, 87)
output: (?, 87)
dense: Dense
input: (?, 87)
output: (?, 89)
dense_1: Dense
input: (?, 89)
output: (?, 80)
dense_2: Dense
input: (?, 80)
output: (?, 70)
dense_3: Dense
input: (?, 70)
output: (?, 60)
dense_4: Dense
input: (?, 60)
output: (?, 50)
dense_5: Dense
input: (?, 50)
output: (?, 40)
dense_6: Dense
input: (?, 40)
output: (?, 30)
bottleneck: Dense
input: (?, 30)



Custom Loss Function and Metric

In [48]:
```
def f1(y_true, y_pred):
    y_pred = K.round(y_pred)
    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)

def f1_loss(y_true, y_pred):

    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 2*p*r / (p+r+K.epsilon())
    f1 = tf.where(tf.math.is_nan(f1), tf.zeros_like(f1), f1)
    return 1 - K.mean(f1)
```

Functions for Confusion Matrix and PR Plot

In [49]:
```
def myf(mod):
    y_hat=mod.predict(X_test).astype(int) #can use either encoded or decoded data..does
    results=pd.DataFrame(CR(y_test, y_hat, output_dict=True))
    try:
        plot_confusion_matrix(mod,X_test,y_test)
    except:
        print('No plot.')
    return(results)

def prplot(mod):
    average_precision = average_precision_score(y_test, mod.predict(X_test))
    disp = plot_precision_recall_curve(mod, X_test, y_test)
    disp.ax_.set_title('2-class Precision-Recall curve: '
                       'AP={0:0.2f}'.format(average_precision))
```

Fit

In [50]:
```
tf.random.set_seed(84)

model = tf.keras.Sequential()
model.add(Flatten())
for i in range(20):
    model.add(Dense(100,activation='relu', kernel_initializer='he_normal')) #H1
```

```
    model.add(Dropout(.2))
model.add(Dense(1,activation='sigmoid', kernel_initializer='he_normal')) #H1
```

Compile and Fit

In [51]:

```
#compile
mybatch=256
myopt='sgd'
myepoch=5
mycalls=tf.keras.callbacks.ModelCheckpoint('D:/', monitor='val_loss', verbose=0, save_b
mymod=model.compile(optimizer=myopt, loss=f1_loss,metrics=['accuracy', 'Recall','Precis

# define the layers
NNET=model.fit(X_train, y_train, epochs=myepoch, batch_size=mybatch, validation_split=.
```

```
Epoch 1/5
1082/1087 [============================>.] - ETA: 0s - loss: 0.4575 - accuracy: 0.3758 -
recall: 0.9962 - precision: 0.3749WARNING:tensorflow:From C:\Users\tf2\lib\site-packages
\tensorflow\python\training\tracking\tracking.py:111: Model.state_updates (from tensorfl
ow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automaticall
y.
WARNING:tensorflow:From C:\Users\tf2\lib\site-packages\tensorflow\python\training\tracki
ng\tracking.py:111: Layer.updates (from tensorflow.python.keras.engine.base_layer) is de
precated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automaticall
y.
INFO:tensorflow:Assets written to: D:/assets
1087/1087 [==============================] - 10s 9ms/step - loss: 0.4574 - accuracy: 0.3
758 - recall: 0.9962 - precision: 0.3750 - val_loss: 0.0369 - val_accuracy: 1.0000 - val
_recall: 1.0000 - val_precision: 1.0000
Epoch 2/5
1077/1087 [============================>.] - ETA: 0s - loss: 0.4553 - accuracy: 0.3749 -
recall: 1.0000 - precision: 0.3749INFO:tensorflow:Assets written to: D:/assets
1087/1087 [==============================] - 9s 9ms/step - loss: 0.4552 - accuracy: 0.37
50 - recall: 1.0000 - precision: 0.3750 - val_loss: 0.0244 - val_accuracy: 1.0000 - val_
recall: 1.0000 - val_precision: 1.0000
Epoch 3/5
1086/1087 [============================>.] - ETA: 0s - loss: 0.4552 - accuracy: 0.3750 -
recall: 1.0000 - precision: 0.3750INFO:tensorflow:Assets written to: D:/assets
1087/1087 [==============================] - 10s 9ms/step - loss: 0.4552 - accuracy: 0.3
750 - recall: 1.0000 - precision: 0.3750 - val_loss: 0.0198 - val_accuracy: 1.0000 - val
_recall: 1.0000 - val_precision: 1.0000
Epoch 4/5
1087/1087 [==============================] - 5s 4ms/step - loss: 0.4550 - accuracy: 0.37
50 - recall: 1.0000 - precision: 0.3750 - val_loss: 0.0210 - val_accuracy: 1.0000 - val_
recall: 1.0000 - val_precision: 1.0000
Epoch 5/5
1087/1087 [==============================] - 4s 4ms/step - loss: 0.4551 - accuracy: 0.37
50 - recall: 1.0000 - precision: 0.3750 - val_loss: 0.0228 - val_accuracy: 1.0000 - val_
recall: 1.0000 - val_precision: 1.0000
```

Results

In [52]:

```
myf(model)
```

```
No plot.
```

```
C:\Users\tf2\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\tf2\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\tf2\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[52]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.912844	0.0	0.912844	0.456422	0.833284
recall	1.000000	0.0	0.912844	0.500000	0.912844
f1-score	0.954436	0.0	0.912844	0.477218	0.871252
support	43581.000000	4161.0	0.912844	47742.000000	47742.000000

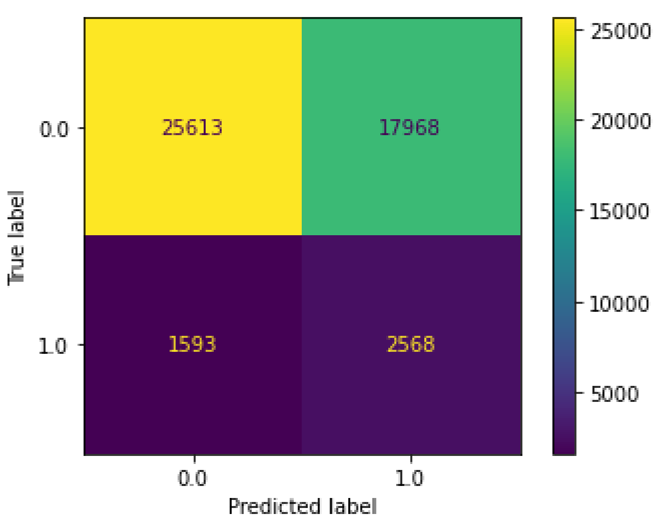
Naive Bayes

In [53]:

```
mynb=GaussianNB()
mynb.fit(X_train, y_train)
myf(mynb)
```

Out[53]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.941447	0.125049	0.590277	0.533248	0.870293
recall	0.587710	0.617159	0.590277	0.602435	0.590277
f1-score	0.723664	0.207960	0.590277	0.465812	0.678717
support	43581.000000	4161.000000	0.590277	47742.000000	47742.000000



In [ ]:

K Nearest Neighbors

In [54]:

```
#myknn=KNN(2)
#myknn.fit(X_train, y_train)
#myf(myknn)
##Not Run.
```

Logistic Regression Models

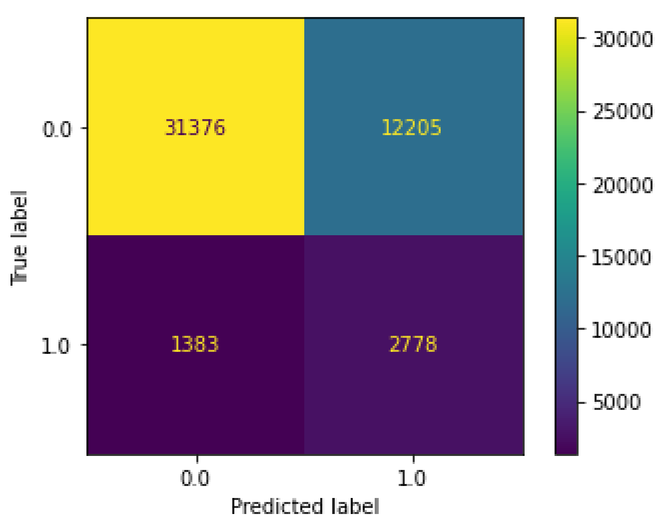
Elasticnet Regularization

In [55]:

```
##################################################################################
mylm=LogisticRegression(fit_intercept = True, penalty='elasticnet', solver='saga', l1_r
myfit2=mylm.fit(X_train, y_train)  #Fit on training data
myf(mylm)
##################################################################################
```

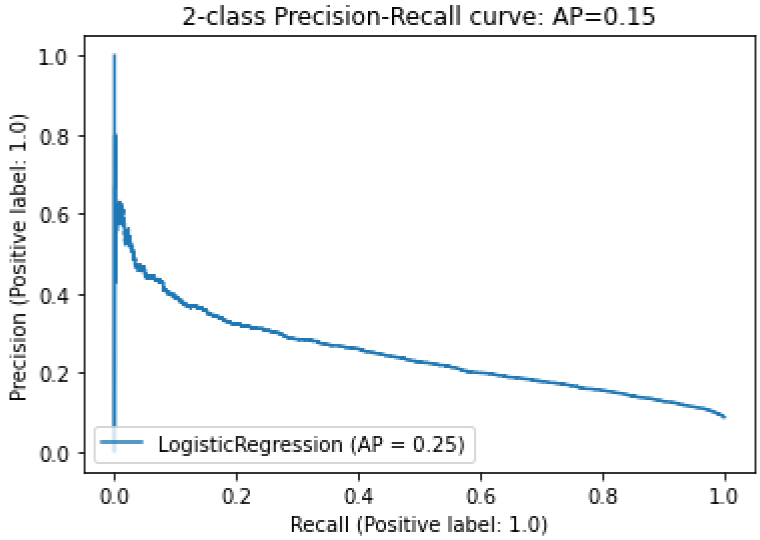
Out[55]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957783	0.185410	0.715387	0.571596	0.890466
recall	0.719947	0.667628	0.715387	0.693787	0.715387
f1-score	0.822007	0.290221	0.715387	0.556114	0.775659
support	43581.000000	4161.000000	0.715387	47742.000000	47742.000000



In [56]:

```
prplot(mylm)
```



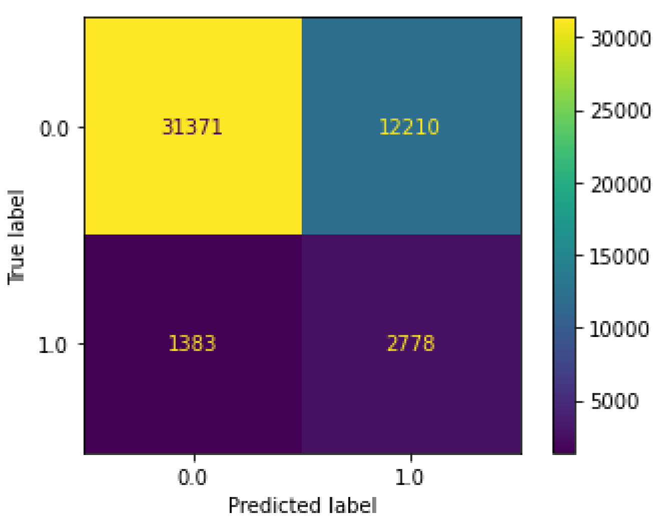
L2 Regularized Logistic Regression

In [57]:

```
###################################################################################
myrr=LogisticRegression(fit_intercept = True, solver='liblinear', penalty='l2') #logist
myrr.fit(X_train, y_train)  #Fit on training data
myf(myrr) #predict on test set and plot
###################################################################################
```

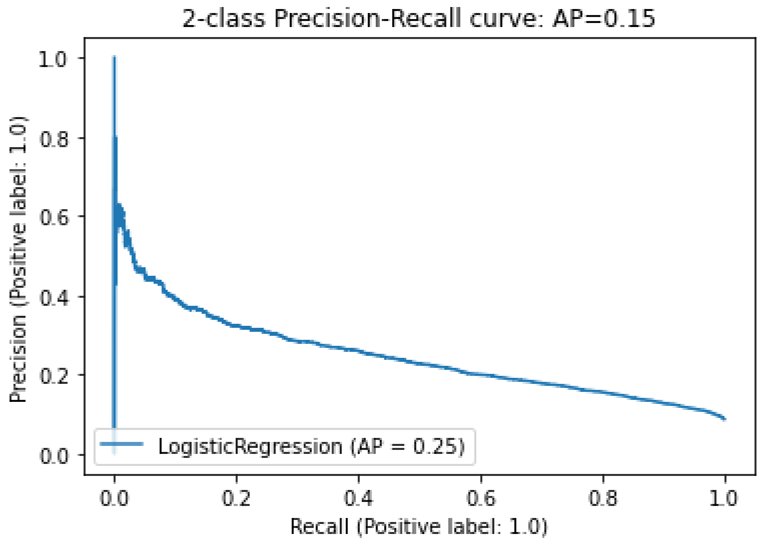
Out[57]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957776	0.185348	0.715282	0.571562	0.890454
recall	0.719832	0.667628	0.715282	0.693730	0.715282
f1-score	0.821930	0.290146	0.715282	0.556038	0.775582
support	43581.000000	4161.000000	0.715282	47742.000000	47742.000000



In [58]:

```
prplot(myrr)
```



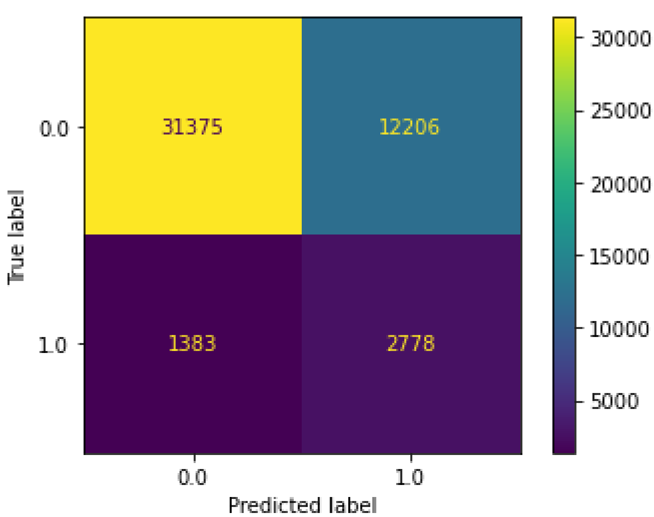
L1 Regularized Logistic Regression

In [59]:

```
#########################################################################################
mylasso=LogisticRegression(fit_intercept = True, solver='liblinear', penalty='l1') #log
mylasso.fit(X_train, y_train)  #Fit on training data
myf(mylasso) #predict on test set and plot
#########################################################################################
```

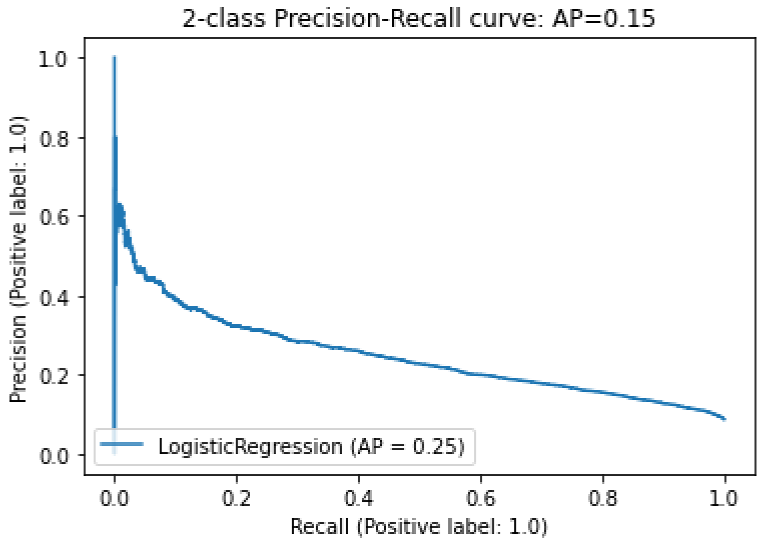
Out[59]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957781	0.185398	0.715366	0.571590	0.890463
recall	0.719924	0.667628	0.715366	0.693776	0.715366
f1-score	0.821991	0.290206	0.715366	0.556099	0.775643
support	43581.000000	4161.000000	0.715366	47742.000000	47742.000000



In [60]:

```
prplot(mylasso)
```

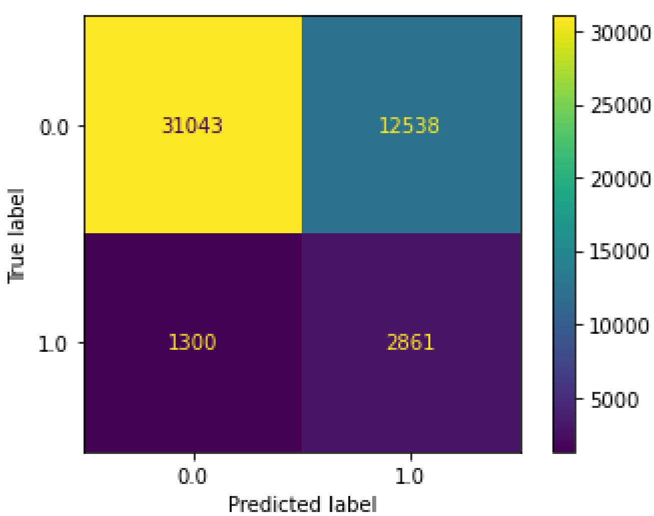


SGD Classifier

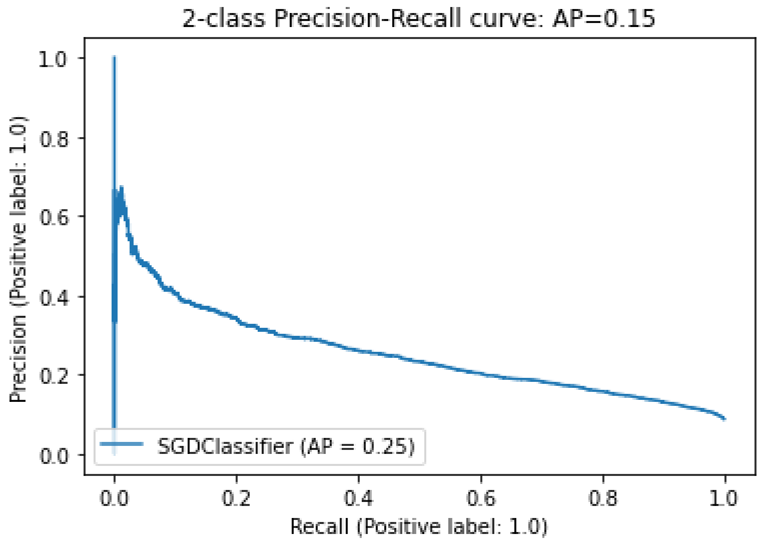
In [61]:
```
mysgd=SGD(loss='log', penalty='l1', alpha=0.001,fit_intercept=True, random_state=43)
mysgd.fit(X_train,y_train)
myf(mysgd)
```

Out[61]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.959806	0.185791	0.71015	0.572799	0.892346
recall	0.712306	0.687575	0.71015	0.699940	0.710150
f1-score	0.817739	0.292536	0.71015	0.555137	0.771964
support	43581.000000	4161.000000	0.71015	47742.000000	47742.000000



In [62]:
```
prplot(mysgd)
```



Comparison of Coefficients

In [63]:

```
def flat(x):
    return(np.reshape(x,87))
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)
p1, p2, p3, p4, p5=temp1.columns.values[1:88],flat(np.exp(mylm.coef_)),flat(np.exp(myrr
pd.DataFrame(list(zip(p1, p2, p3, p4, p5)), columns=['Name','Elasticnet', 'L2', 'L1', '
#print(mylm.intercept_)
```

Out[63]:

	Name	Elasticnet	L2	L1	SGD
0	Age65	1.661347	1.661113	1.660677	1.654502
1	Male	2.646639	2.646401	2.645631	2.534301
2	Unmarried	1.201797	1.201793	1.201718	1.184809
3	Veteran	1.084321	1.084387	1.084407	1.068595
4	Income	1.258690	1.258612	1.258179	1.233259
5	Unemployed	1.529611	1.529493	1.529174	1.515400
6	RentHome	1.069201	1.069254	1.069172	1.000000
7	PoorHealth	2.029135	2.029113	2.029282	1.983233
8	Smoker	1.412261	1.412242	1.412099	1.415895
9	ChewSnuff	0.623653	0.623310	0.623744	0.827535
10	PoorExercise	1.081895	1.081824	1.081522	1.077592
11	DrinksDaily	0.533456	0.533375	0.533294	0.559502
12	HighCholesterol	1.633729	1.633642	1.633392	1.617978
13	HighBP	1.829978	1.829878	1.829659	1.820029
14	HighBMI	1.035593	1.035600	1.035305	1.040787
15	Diabetes	1.419641	1.419547	1.419567	1.419995

	Name	Elasticnet	L2	L1	SGD
16	PhysicalHealth	1.139879	1.140101	1.139921	1.075607
17	MentalHealth	0.814558	0.814326	0.814565	0.925929
18	Depression	0.884984	0.884912	0.884928	0.923029
19	Stroke	2.613279	2.613333	2.613261	2.390141
20	Asthma	0.902248	0.902089	0.902148	0.937267
21	SkinCancer	0.863745	0.863646	0.863713	0.905511
22	Cancer	0.856420	0.856295	0.856358	0.913641
23	COPD	1.507652	1.507703	1.507746	1.448758
24	Kidney	1.252416	1.252750	1.252499	1.151125
25	Arthritis	1.195929	1.195954	1.195785	1.189481
26	NoHealthPlan	0.884715	0.884264	0.884678	1.000000
27	NoDoctor	0.718501	0.718454	0.718494	0.751770
28	Cost	1.049369	1.049705	1.049247	1.000000
29	NoCheckup	0.668632	0.668566	0.668508	0.696244
30	Weights	0.463114	0.462900	0.463011	0.563087
31	Stratum	0.545778	0.545786	0.545779	0.638079
32	Black	0.778853	0.778458	0.778791	0.853295
33	Hispanic	1.394247	1.394650	1.394115	1.215130
34	Other	1.160166	1.160301	1.159960	1.099733
35	PreHS	1.100885	1.100975	1.100625	1.014166
36	HS	0.341462	0.340766	0.340756	1.000000
37	PostHS	0.404955	0.403828	0.404209	1.000000
38	State_AK	0.439328	0.438071	0.438540	1.000000
39	State_AL	0.431559	0.430353	0.430728	1.000000
40	State_AR	0.424763	0.423547	0.423908	1.000000
41	State_AZ	0.396126	0.395010	0.395340	1.000000
42	State_CA	0.320210	0.319362	0.319602	0.916362
43	State_CO	0.459979	0.458491	0.459161	1.000000
44	State_CT	0.393962	0.392889	0.393213	1.000000
45	State_DC	0.468722	0.467432	0.467856	1.000000
46	State_DE	0.396078	0.394973	0.395338	1.000000
47	State_FL	0.325336	0.324938	0.324667	1.000000
48	State_GA	0.388987	0.387946	0.388182	1.000000

	Name	Elasticnet	L2	L1	SGD
49	State_GU	0.480011	0.478605	0.479095	1.000000
50	State_HI	0.383566	0.382586	0.382820	1.000000
51	State_IA	0.444556	0.443172	0.443688	1.000000
52	State_ID	0.444591	0.443317	0.443743	1.000000
53	State_IL	0.398233	0.397132	0.397466	1.000000
54	State_IN	0.513380	0.511892	0.512477	1.000000
55	State_KS	0.338848	0.338026	0.338203	1.000000
56	State_KY	0.393881	0.392788	0.393135	1.000000
57	State_LA	0.539401	0.537855	0.538376	1.000000
58	State_MA	0.483949	0.482572	0.483054	1.000000
59	State_ME	0.537136	0.535506	0.536099	1.000000
60	State_MI	0.525653	0.524099	0.524662	1.000000
61	State_MN	0.415428	0.414274	0.414687	1.000000
62	State_MO	0.478520	0.477091	0.477648	1.000000
63	State_MS	0.366624	0.365738	0.365932	1.000000
64	State_MT	0.506115	0.504541	0.505144	1.000000
65	State_NC	0.516492	0.515009	0.515520	1.000000
66	State_ND	0.362318	0.361357	0.361635	1.000000
67	State_NE	0.444792	0.443439	0.443949	1.000000
68	State_NH	0.383895	0.382776	0.383153	1.000000
69	State_NM	0.439813	0.438550	0.438970	1.000000
70	State_NV	0.510958	0.509499	0.510029	1.000000
71	State_NY	0.418092	0.416890	0.417319	1.000000
72	State_OH	0.378321	0.377408	0.377599	1.000000
73	State_OK	0.433137	0.431846	0.432286	1.000000
74	State_OR	0.459953	0.458419	0.459128	1.000000
75	State_PA	0.311076	0.310353	0.310487	1.000000
76	State_PR	0.442829	0.441597	0.442043	1.000000
77	State_RI	0.437879	0.436615	0.437063	1.000000
78	State_SC	0.431429	0.430181	0.430636	1.000000
79	State_SD	0.440130	0.438914	0.439301	1.000000
80	State_TN	0.513209	0.511751	0.512179	1.000000
81	State_TX	0.430721	0.429546	0.429893	1.000000

	Name	Elasticnet	L2	L1	SGD
82	State_UT	0.476065	0.474649	0.475105	1.000000
83	State_VA	0.409581	0.408472	0.408775	1.000000
84	State_VT	0.383927	0.382987	0.383220	1.000000
85	State_WA	0.582292	0.580415	0.581216	1.000000
86	State_WI	0.444506	0.443152	0.443681	1.000000

Tree Models

Tree Plots

In [64]:

```
def mytree(mod):
    imp, std=mod.feature_importances_, np.std([mod.feature_importances_ for tree in mod
    importances = pd.Series(imp, index=temp1.columns[1:88]).sort_values(ascending=False
    fig, ax = plt.subplots()
    importances.plot.bar(yerr=std[0:20], ax=ax)
    ax.set_title("Feature importances using MDI")
    ax.set_ylabel("Mean decrease in impurity")
    fig.tight_layout()
```

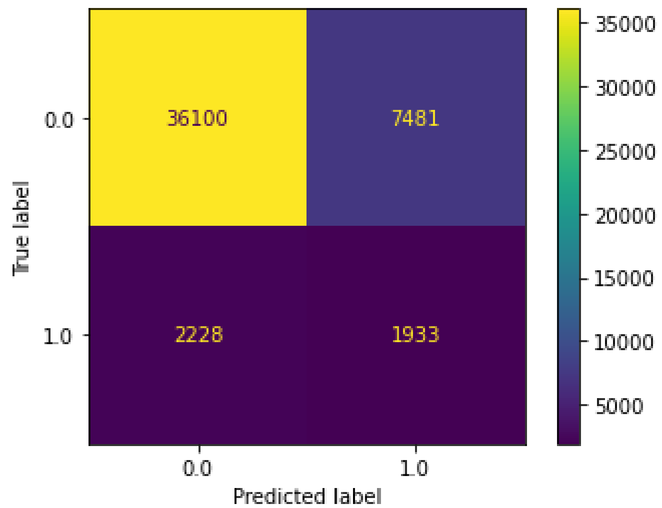
Decision Tree Classifier

In [65]:

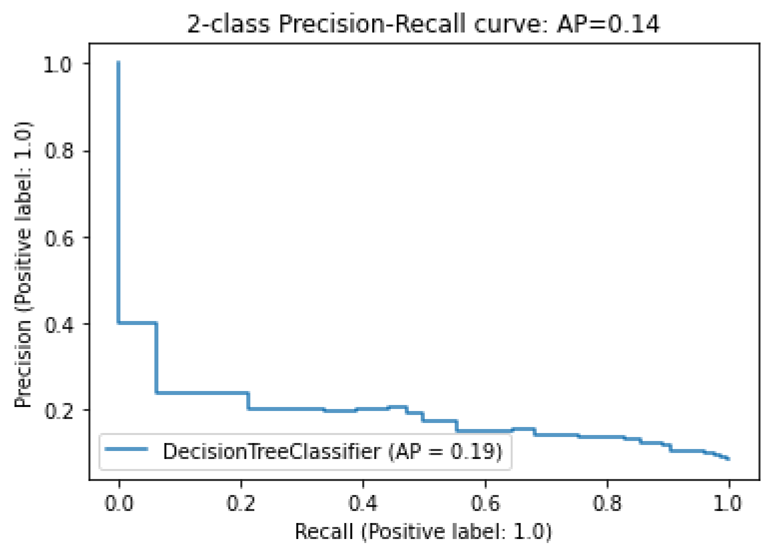
```
myTree=Tree(criterion='entropy', splitter='best', max_depth=8,min_samples_split=2, min_
myTree.fit(X_train,y_train)
myf(myTree)
```

Out[65]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.941870	0.205332	0.796636	0.573601	0.877677
recall	0.828343	0.464552	0.796636	0.646447	0.796636
f1-score	0.881466	0.284788	0.796636	0.583127	0.829462
support	43581.000000	4161.000000	0.796636	47742.000000	47742.000000



In [66]:
```
prplot(myTree)
```

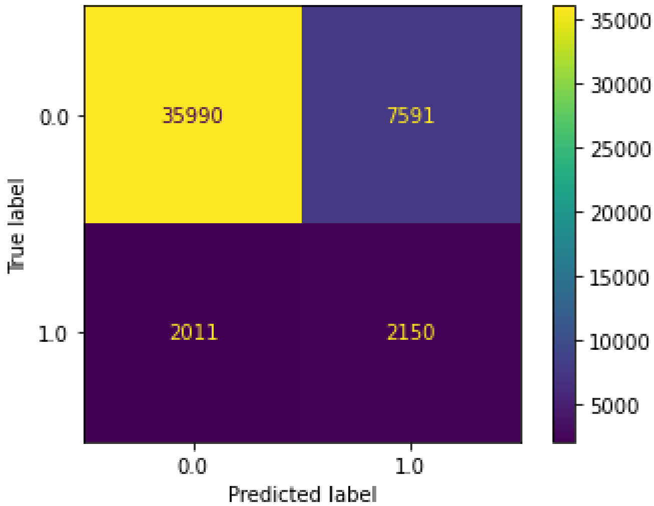


Random Forest Classifier

In [67]:
```
#########################################################################################
myrf=RandomForestClassifier(n_estimators = 100,max_depth=10,min_samples_split=2, criter
                            n_jobs = -1, random_state = 64) #RF Model
myrf.fit(X_train, y_train)  # Fit on the training set
myf(myrf)
#########################################################################################
```

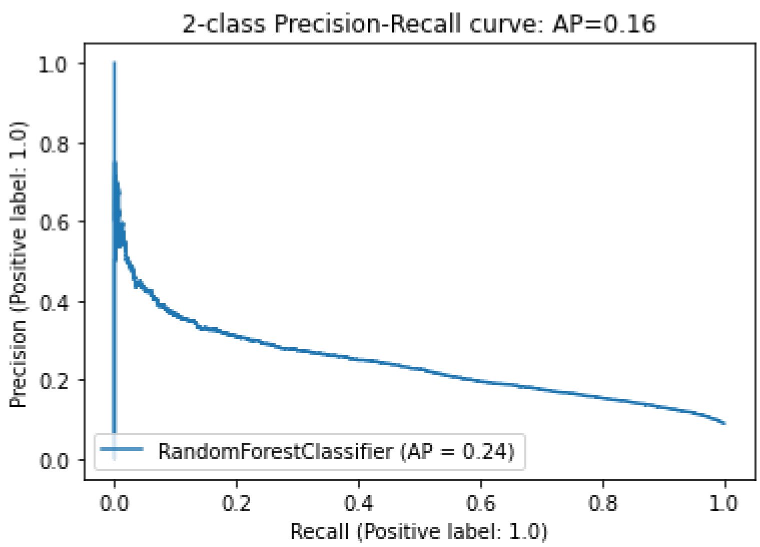
Out[67]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.947080	0.220717	0.798877	0.583898	0.883773
recall	0.825819	0.516703	0.798877	0.671261	0.798877
f1-score	0.882302	0.309308	0.798877	0.595805	0.832363
support	43581.000000	4161.000000	0.798877	47742.000000	47742.000000



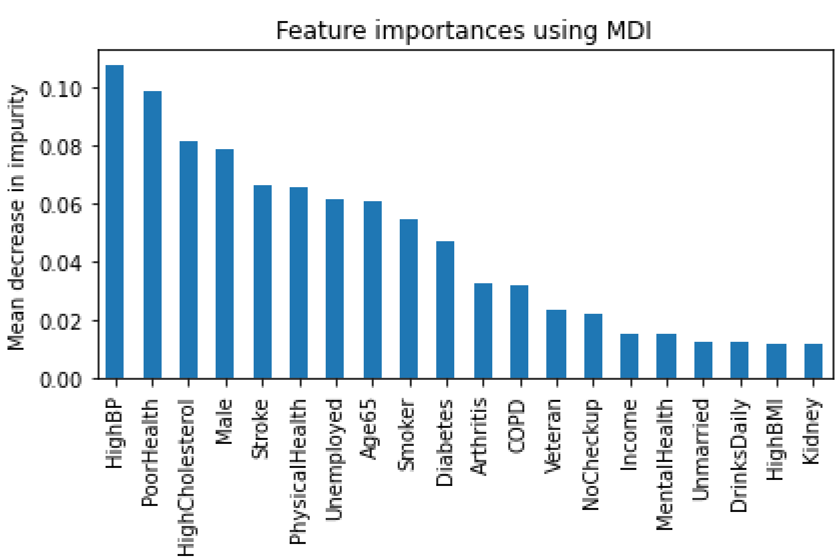
In [68]:

```
prplot(myrf)
```



In [69]:

```
mytree(myrf)
```



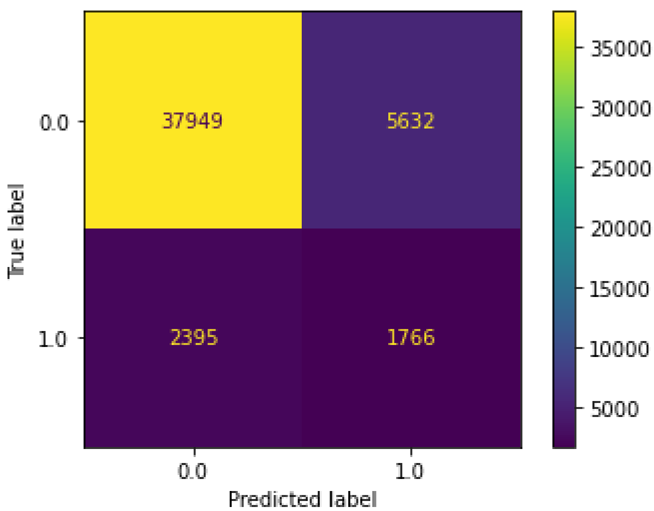
Extra Trees Classifier

In [70]:

```
##########################################################################################
myextra=ExtraTreesClassifier(n_estimators = 50, max_depth=22, min_samples_split=2,crite
                            bootstrap=True, n_jobs = -1, random_state = 64) #previously
myextra.fit(X_train, y_train)  #Fit on training set
myf(myextra)
########################################################
```

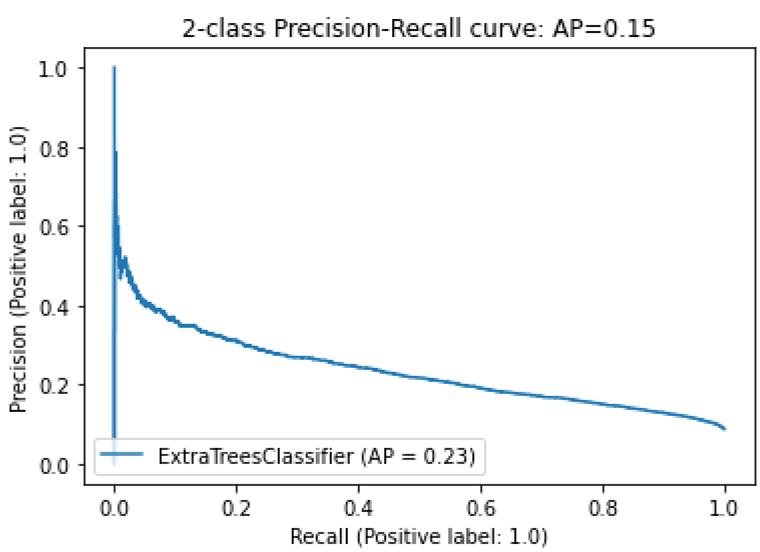
Out[70]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.940636	0.238713	0.831867	0.589674	0.879459
recall	0.870769	0.424417	0.831867	0.647593	0.831867
f1-score	0.904355	0.305563	0.831867	0.604959	0.852167
support	43581.000000	4161.000000	0.831867	47742.000000	47742.000000



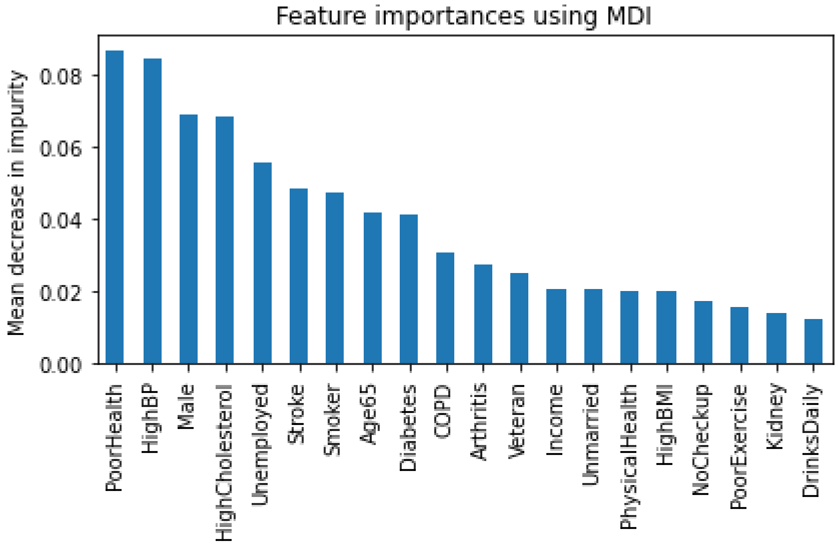
In [71]:

```
prplot(myextra)
```



In [72]:

```
mytree(myextra)
```



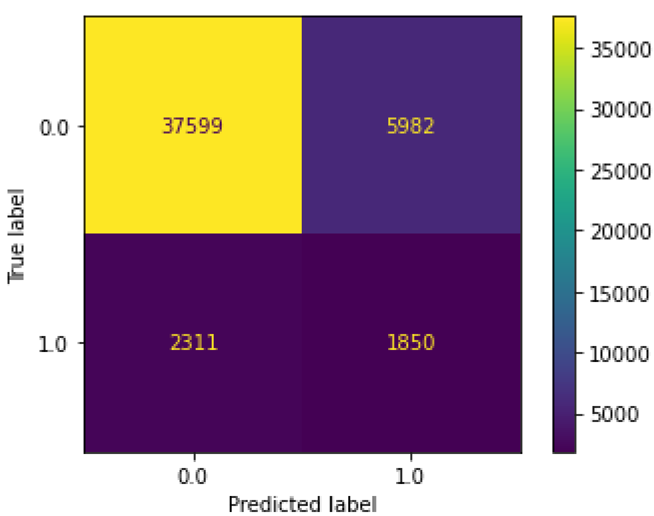
Gradient Boosting

In [73]:

```
####################################################################################
myGBC=GradientBoostingClassifier(n_estimators = 50, random_state = 64)
myGBC.fit(X_train, y_train)  #Fit on training set
myf(myGBC)
####################################################################################
```

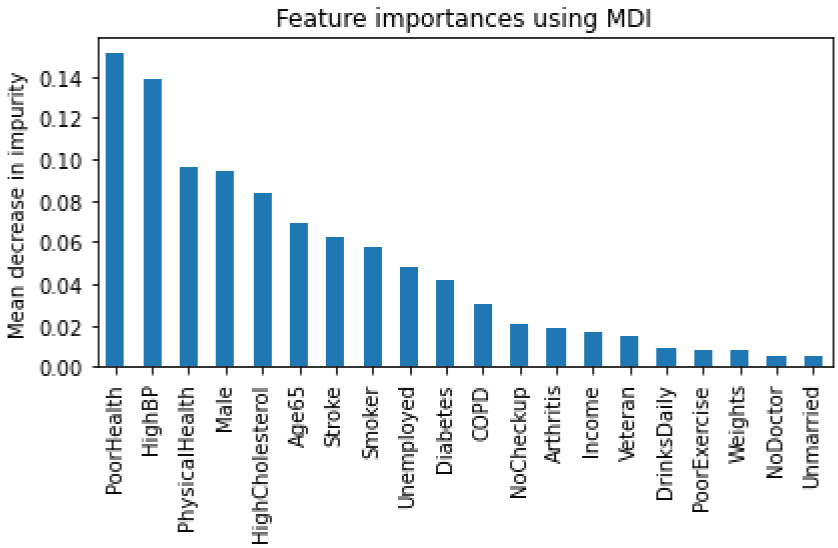
Out[73]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.942095	0.236210	0.826296	0.589153	0.880573
recall	0.862738	0.444605	0.826296	0.653672	0.826296
f1-score	0.900672	0.308513	0.826296	0.604593	0.849062
support	43581.000000	4161.000000	0.826296	47742.000000	47742.000000



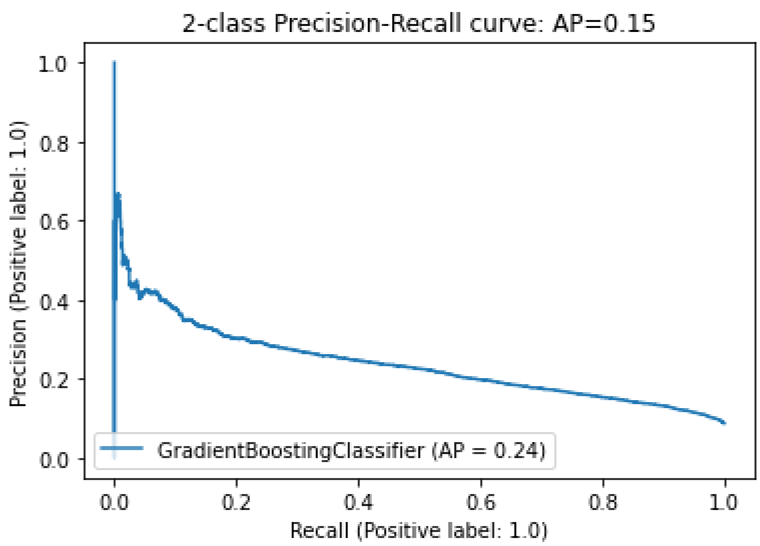
In [74]:

```
mytree(myGBC)
```



In [75]:

```
prplot(myGBC)
```



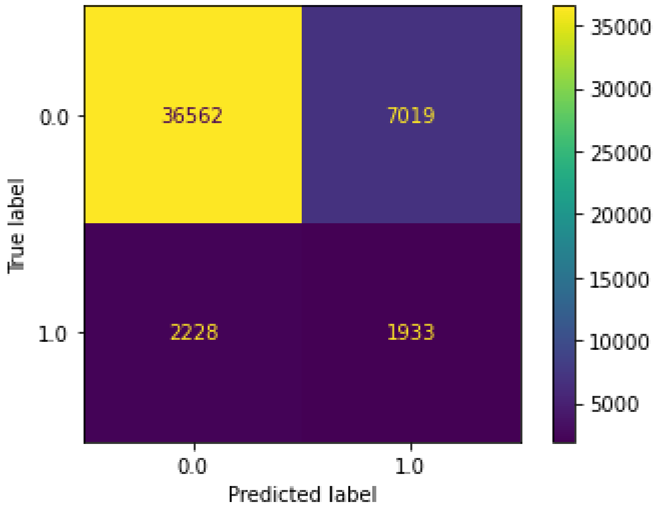
ADA Boost Classifier

In [76]:

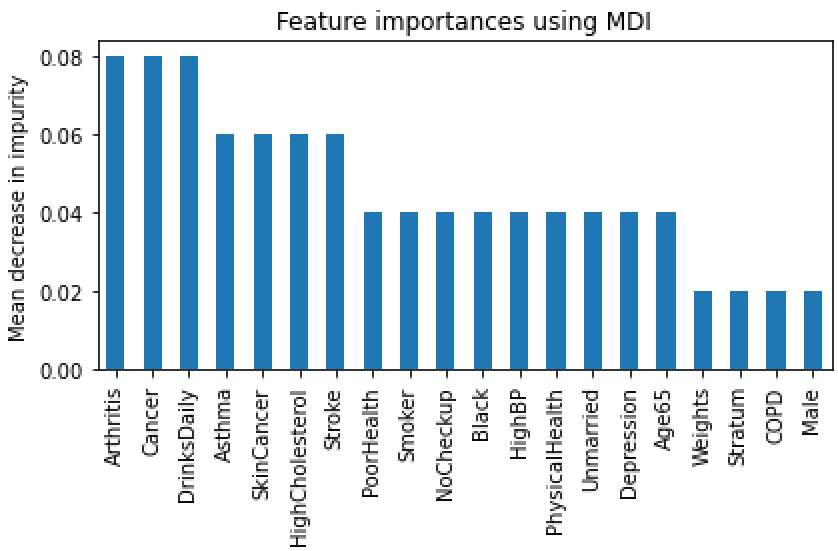
```
myADA=ADA(n_estimators=50, random_state=0)
myADA.fit(X_train,y_train)
myf(myADA)
```

Out[76]:

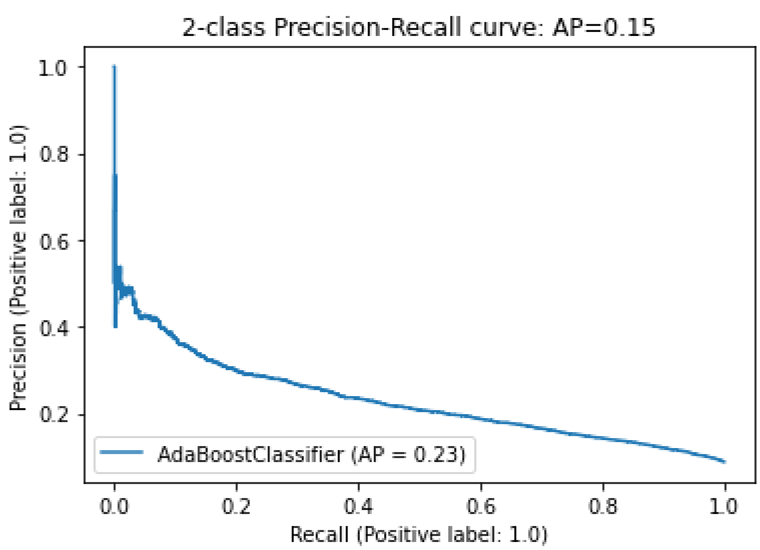
	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.942563	0.215929	0.806313	0.579246	0.879232
recall	0.838944	0.464552	0.806313	0.651748	0.806313
f1-score	0.887740	0.294822	0.806313	0.591281	0.836063
support	43581.000000	4161.000000	0.806313	47742.000000	47742.000000



In [77]:
```
mytree(myADA)
```



In [78]:
```
prplot(myADA)
```



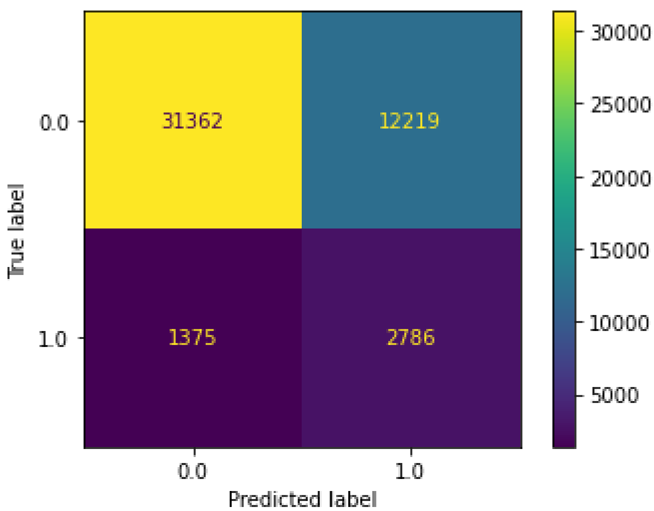
Discriminant Analysis

LDA

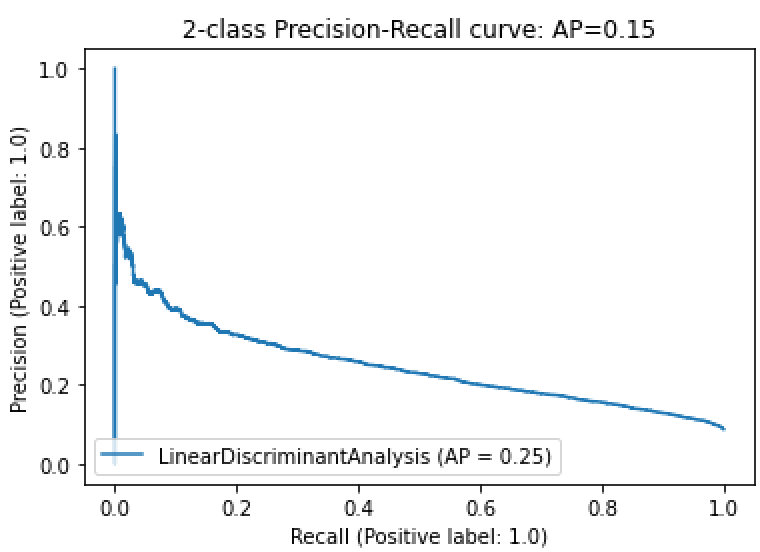
In [79]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
mylda=LDA()
mylda.fit(X_train, y_train)  # Fit on the training set
myf(mylda)
```

Out[79]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957999	0.185671	0.715261	0.571835	0.890686
recall	0.719626	0.669551	0.715261	0.694588	0.715261
f1-score	0.821877	0.290723	0.715261	0.556300	0.775584
support	43581.000000	4161.000000	0.715261	47742.000000	47742.000000



In [80]:
```
prplot(mylda)
```

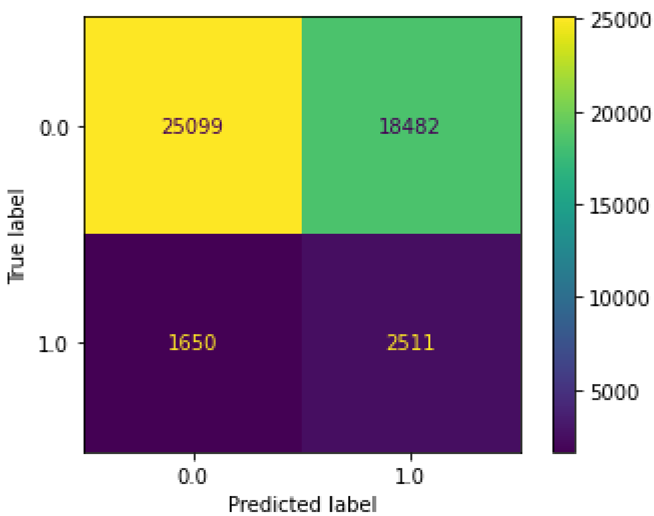


QDA

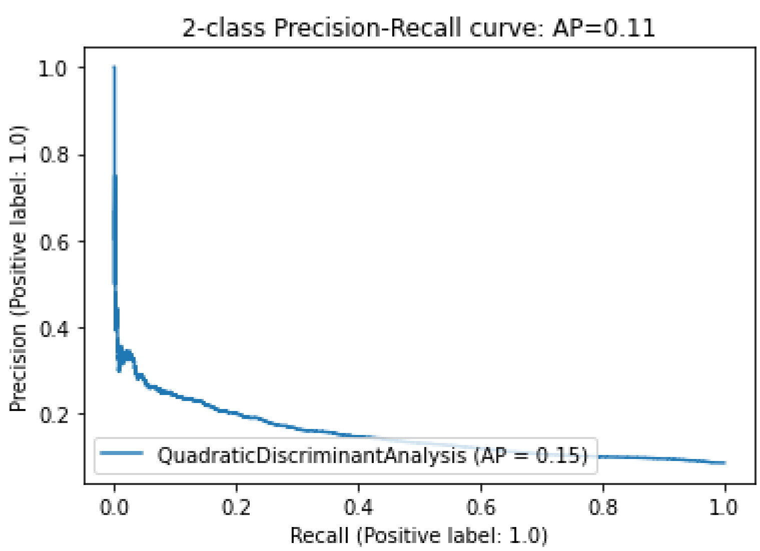
In [81]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
myqda=QDA()
myqda.fit(X_train, y_train)  #Fit on training set
myf(myqda)
```

Out[81]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.938315	0.119611	0.578317	0.528963	0.866961
recall	0.575916	0.603461	0.578317	0.589688	0.578317
f1-score	0.713749	0.199650	0.578317	0.456700	0.668943
support	43581.000000	4161.000000	0.578317	47742.000000	47742.000000



In [82]:
```
prplot(myqda)
```

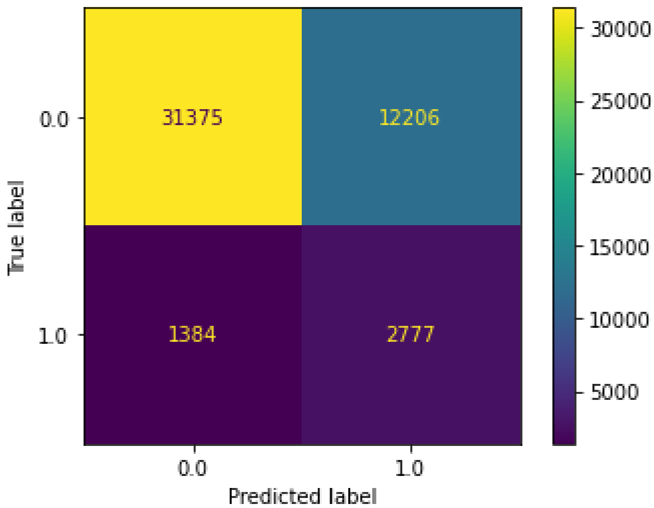


Linear Support Vector Machine

In [83]:
```
mysvm=LinearSVC(random_state=0, tol=1e-5)
mysvm.fit(X_train, y_train)  #Fit on training set
myf(mysvm)
```

Out[83]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957752	0.185343	0.715345	0.571548	0.890432
recall	0.719924	0.667388	0.715345	0.693656	0.715345
f1-score	0.821981	0.290117	0.715345	0.556049	0.775626
support	43581.000000	4161.000000	0.715345	47742.000000	47742.000000



In [84]:
```
prplot(mysvm)
```