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.ensemble import RandomForestClassifier # Random Forest package
from sklearn.ensemble import ExtraTreesClassifier # Extra Trees package
from sklearn.ensemble import GradientBoostingClassifier # Gradient Boosting package
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report as CR
from sklearn.pipeline import make_pipeline
import statsmodels.api as sm

#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

#Graphing
import seaborn as sns
import pydot  #For model plotting
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: 15277966495489453965
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 4897325358030976573
physical_device_desc: "device: XLA_CPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6918604064
locality {
  bus_id: 1
  links {
  }
}
incarnation: 12285729331127543111
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: 13336915221585023638
physical_device_desc: "device: XLA_GPU device"
]
```

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

In [3]:

```
#########################################################################################
mydata=pd.read_csv('y2019_1.csv')
#########################################################################################
```

Describe

In [4]:

```
#########################################################################################
ProfileReport(mydata, title='Pandas Profiling Report')
#########################################################################################
```

Overview

Dataset statistics

Number of variables	90
Number of observations	224765
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	30427
Duplicate rows (%)	13.5%
Total size in memory	154.3 MiB
Average record size in memory	720.0 B

Variable types

Categorical	70
Numeric	20

Warnings

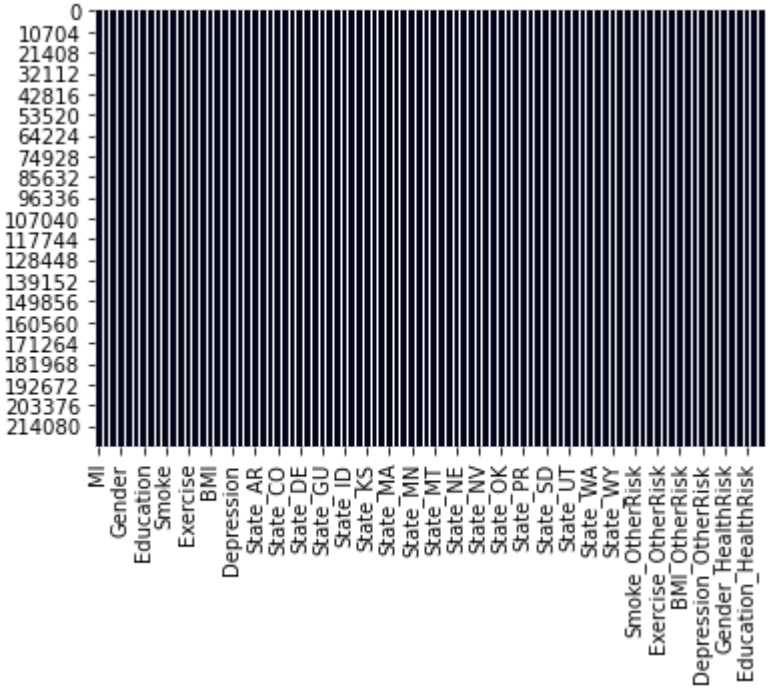
Dataset has 30427 (13.5%) duplicate rows	**Duplicates**
`Age` is highly correlated with `Age_HealthRisk`	**High correlation**
`Race` is highly correlated with `Race_HealthRisk`	**High correlation**
`Gender` is highly correlated with `Gender_HealthRisk`	**High correlation**

Out[4]:

Evaluate Missing

In [5]:
```
###############################################################################
sns.heatmap(mydata.isnull(), cbar=False)
###############################################################################
```

Out[5]: <AxesSubplot:>

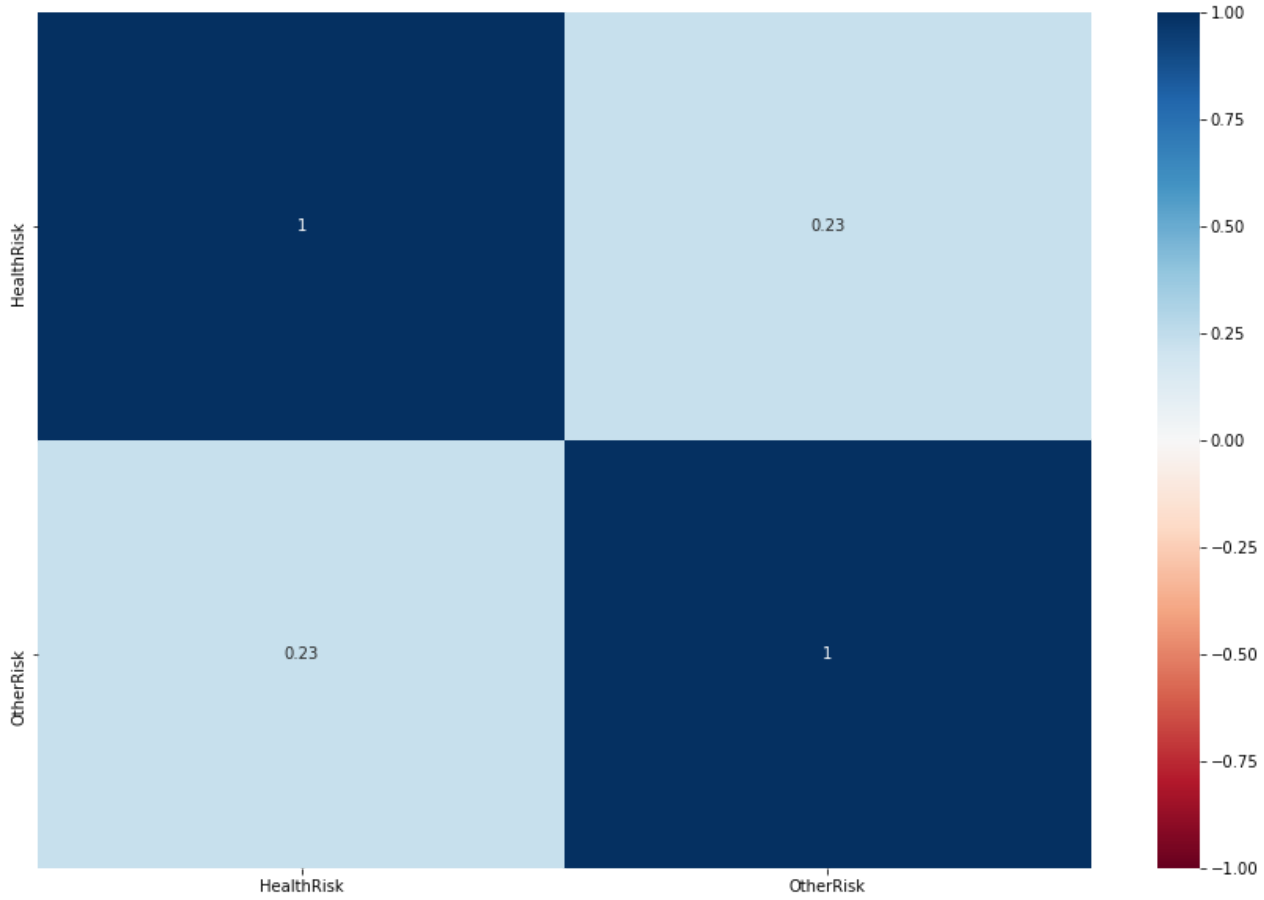


In [6]:
```
mydata[['HealthRisk','OtherRisk']].describe()
```

Out[6]:

	HealthRisk	OtherRisk
count	224765.000000	224765.000000
mean	0.338951	0.501380
std	0.165625	0.204582
min	0.000000	0.000000
25%	0.200000	0.375000
50%	0.300000	0.500000
75%	0.400000	0.625000
max	1.000000	1.000000

In [7]:
```
plt.figure(figsize=(15,10))
correlations = mydata[['HealthRisk','OtherRisk']].corr()
sns.heatmap(round(correlations,2), cmap='RdBu', annot=True,
            annot_kws={"size": 10}, vmin=-1, vmax=1);
```



Build Training and Test Set

In [8]:

```
##############################################################################################
# Seed value for random number generators to obtain reproducible results
temp=mydata.values
tempy=temp[:,0]
tempx=temp[:,1:72]

X_train, X_test, y_train, y_test = tts(tempx, tempy, test_size=.2, random_state=1234)

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

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

Tensorflow

In [9]:

```
tf.random.set_seed(64)

model = tf.keras.Sequential()
model.add(Flatten())
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
```

```
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10,activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1,activation='sigmoid'))
```

In [10]:

```
#compile
mybatch=32
myopt='sgd'
myepoch=10
mycalls=tf.keras.callbacks.ModelCheckpoint('D:/', monitor='val_loss', verbose=0, save_b
mymod=model.compile(optimizer=myopt, loss='binary_crossentropy',metrics=['accuracy', 'R

# define the layers
model.fit(X_train, y_train, epochs=myepoch, batch_size=mybatch, validation_split=.2,cal
```

```
Epoch 1/10
8158/8173 [============================>.] - ETA: 0s - loss: 0.5544 - accuracy: 0.7102 -
recall: 0.5184 - precision: 0.6402WARNING:tensorflow:From C:\Users\tf\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\tf\lib\site-packages\tensorflow\python\training\trackin
g\tracking.py:111: Layer.updates (from tensorflow.python.keras.engine.base_layer) is dep
recated 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
8173/8173 [==============================] - 22s 3ms/step - loss: 0.5544 - accuracy: 0.7
102 - recall: 0.5186 - precision: 0.6403 - val_loss: 0.5961 - val_accuracy: 0.7285 - val
_recall: 0.7285 - val_precision: 1.0000
Epoch 2/10
8173/8173 [==============================] - 20s 2ms/step - loss: 0.5347 - accuracy: 0.7
245 - recall: 0.5654 - precision: 0.6534 - val_loss: 0.7990 - val_accuracy: 0.5673 - val
_recall: 0.5673 - val_precision: 1.0000
Epoch 3/10
8173/8173 [==============================] - 20s 3ms/step - loss: 0.5309 - accuracy: 0.7
261 - recall: 0.5804 - precision: 0.6514 - val_loss: 0.6491 - val_accuracy: 0.6851 - val
_recall: 0.6851 - val_precision: 1.0000
Epoch 4/10
8173/8173 [==============================] - 20s 2ms/step - loss: 0.5278 - accuracy: 0.7
286 - recall: 0.5876 - precision: 0.6536 - val_loss: 0.7948 - val_accuracy: 0.5636 - val
_recall: 0.5636 - val_precision: 1.0000
Epoch 5/10
8173/8173 [==============================] - 21s 3ms/step - loss: 0.5245 - accuracy: 0.7
308 - recall: 0.5907 - precision: 0.6569 - val_loss: 0.7277 - val_accuracy: 0.5872 - val
_recall: 0.5872 - val_precision: 1.0000
Epoch 6/10
8173/8173 [==============================] - 21s 3ms/step - loss: 0.5213 - accuracy: 0.7
324 - recall: 0.5931 - precision: 0.6590 - val_loss: 0.6513 - val_accuracy: 0.6770 - val
_recall: 0.6770 - val_precision: 1.0000
Epoch 7/10
8173/8173 [==============================] - 20s 2ms/step - loss: 0.5185 - accuracy: 0.7
351 - recall: 0.5925 - precision: 0.6646 - val_loss: 0.6086 - val_accuracy: 0.6971 - val
_recall: 0.6971 - val_precision: 1.0000
Epoch 8/10
```

```
8173/8173 [==============================] - 29s 4ms/step - loss: 0.5159 - accuracy: 0.7
373 - recall: 0.5949 - precision: 0.6682 - val_loss: 0.9363 - val_accuracy: 0.5043 - val
_recall: 0.5043 - val_precision: 1.0000
Epoch 9/10
8170/8173 [==============================>.] - ETA: 0s - loss: 0.5133 - accuracy: 0.7393 -
recall: 0.5957 - precision: 0.6720INFO:tensorflow:Assets written to: D:/assets
8173/8173 [==============================] - 25s 3ms/step - loss: 0.5133 - accuracy: 0.7
393 - recall: 0.5956 - precision: 0.6719 - val_loss: 0.5829 - val_accuracy: 0.7176 - val
_recall: 0.7176 - val_precision: 1.0000
Epoch 10/10
8173/8173 [==============================] - 20s 3ms/step - loss: 0.5104 - accuracy: 0.7
426 - recall: 0.5975 - precision: 0.6780 - val_loss: 0.6853 - val_accuracy: 0.6304 - val
_recall: 0.6304 - val_precision: 1.0000
```

Out[10]:
```
<tensorflow.python.keras.callbacks.History at 0x244271a3240>
```

In [11]:
```
yhat = np.round(model.predict(X_test))
pd.DataFrame(CR(y_test, yhat, output_dict=True))
```

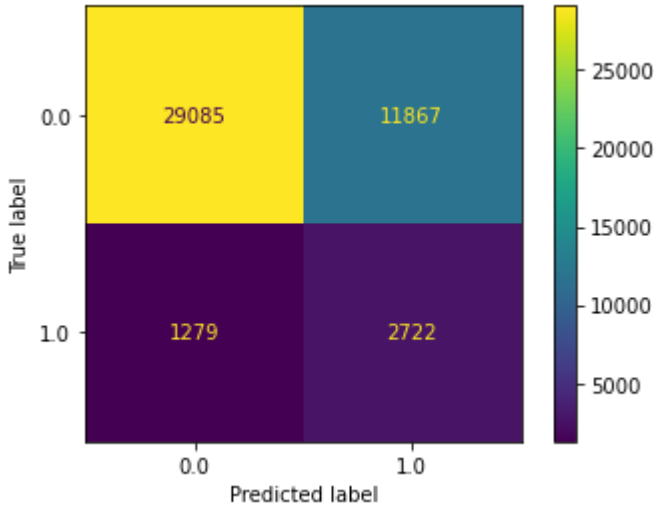
Out[11]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.945025	0.215372	0.790225	0.580198	0.880083
recall	0.817274	0.513372	0.790225	0.665323	0.790225
f1-score	0.876519	0.303442	0.790225	0.589981	0.825513
support	40952.000000	4001.000000	0.790225	44953.000000	44953.000000

ElasticNet Logistic Regression

In [12]:
```
##############################################################################################
mylm=LogisticRegression(fit_intercept = True, penalty='elasticnet', solver='saga', l1_r
mylm.fit(X_train, y_train)  #Fit on training data
plot_confusion_matrix(mylm,X_test,y_test)
mypred=mylm.predict(X_test)
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
##############################################################################################
```

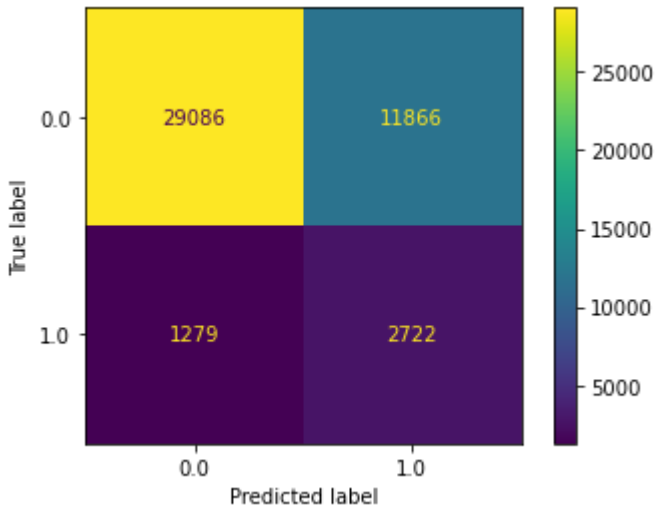
```
                   0.0          1.0  accuracy     macro avg  weighted avg
precision     0.957878     0.186579  0.707561      0.572228      0.889229
recall        0.710222     0.680330  0.707561      0.695276      0.707561
f1-score      0.815665     0.292846  0.707561      0.554256      0.769132
support   40952.000000  4001.000000  0.707561  44953.000000  44953.000000
```



L2 Regularized Logistic Regression

In [13]:
```
##############################################################################
myrr=LogisticRegression(fit_intercept = True, solver='liblinear', penalty='l2') #logist
myrr.fit(X_train, y_train)  #Fit on training data
plot_confusion_matrix(myrr,X_test,y_test)
mypred=myrr.predict(X_test)
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
##############################################################################
```

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957879	0.186592	0.707583	0.572235	0.889231
recall	0.710246	0.680330	0.707583	0.695288	0.707583
f1-score	0.815682	0.292861	0.707583	0.554272	0.769149
support	40952.000000	4001.000000	0.707583	44953.000000	44953.000000

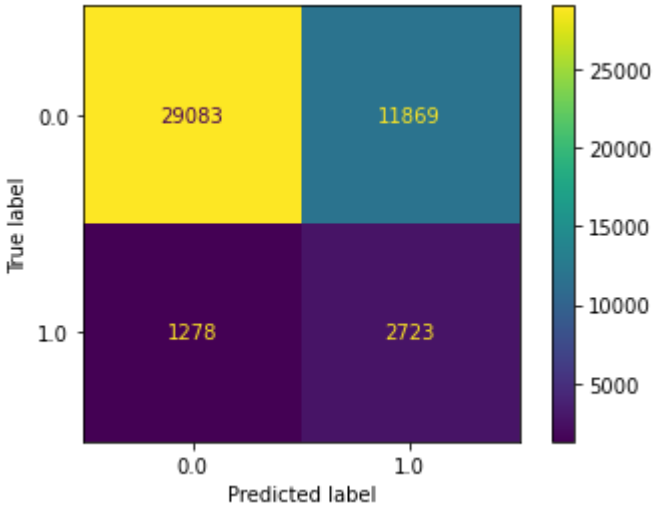


L1 Regularized Logistic Regression

In [14]:
```
##############################################################################
mylasso=LogisticRegression(fit_intercept = True, solver='liblinear', penalty='l1') #Log
mylasso.fit(X_train, y_train)  #Fit on training data
plot_confusion_matrix(mylasso,X_test,y_test)
mypred=mylasso.predict(X_test)
```

```
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
#################################################################################################
```

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.957907	0.186609	0.707539	0.572258	0.889258
recall	0.710173	0.680580	0.707539	0.695376	0.707539
f1-score	0.815644	0.292906	0.707539	0.554275	0.769118
support	40952.000000	4001.000000	0.707539	44953.000000	44953.000000

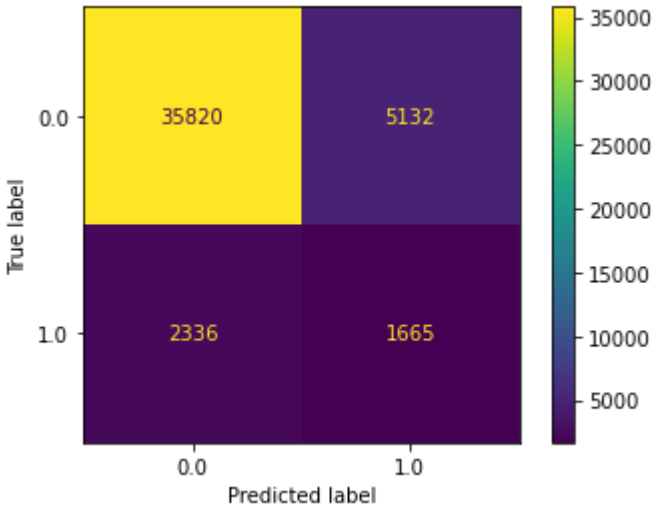


Random Forest Classifier

In [15]:

```
#################################################################################################
myrf=RandomForestClassifier(n_estimators = 50,
                            max_depth=20,min_samples_split=2, criterion='entropy',
                            bootstrap=True, n_jobs = -1, random_state = 64) #RF Model
myrf.fit(X_train, y_train)  # Fit on the training set
plot_confusion_matrix(myrf,X_test,y_test)
mypred=myrf.predict(X_test)
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
#################################################################################################
```

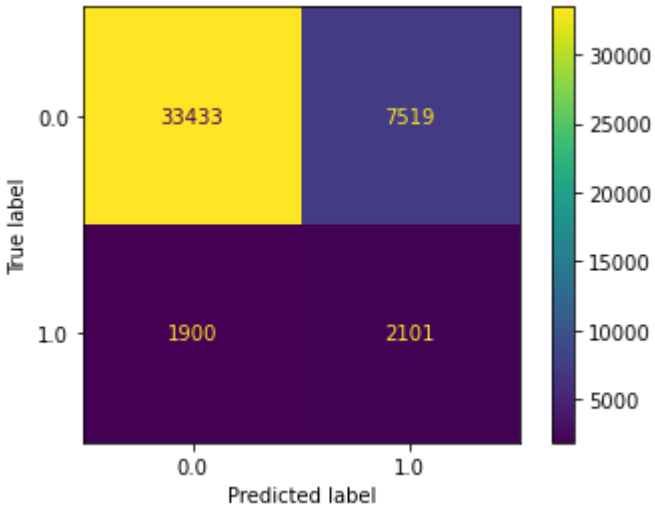
	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.938778	0.244961	0.833871	0.591869	0.877025
recall	0.874683	0.416146	0.833871	0.645414	0.833871
f1-score	0.905597	0.308390	0.833871	0.606994	0.852444
support	40952.000000	4001.000000	0.833871	44953.000000	44953.000000



Extra Trees Classifier

In [16]:
```
##############################################################################################
myextra=ExtraTreesClassifier(n_estimators = 50, max_depth=22, min_samples_split=2,crite
                             bootstrap=True, n_jobs = -1, random_state = 64)
myextra.fit(X_train, y_train)  #Fit on training set
plot_confusion_matrix(myextra,X_test,y_test)
mypred=myextra.predict(X_test)
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
##############################################################################################
```

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.946226	0.218399	0.79047	0.582313	0.881446
recall	0.816395	0.525119	0.79047	0.670757	0.790470
f1-score	0.876529	0.308494	0.79047	0.592512	0.825971
support	40952.000000	4001.000000	0.79047	44953.000000	44953.000000

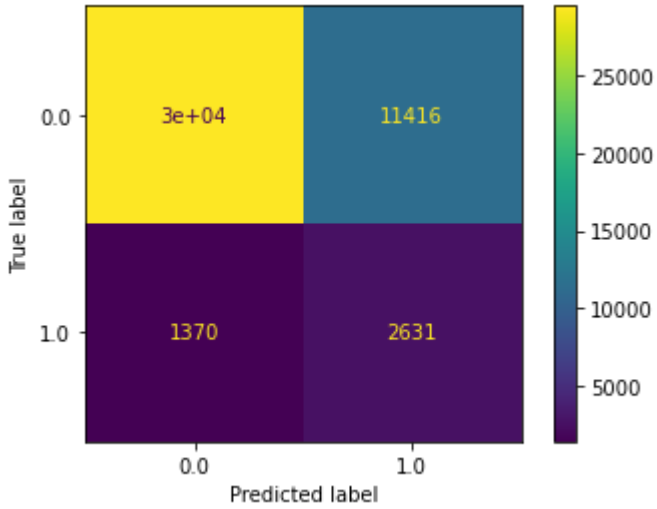


Extreme Gradient Boosting

In [17]:
```
##############################################################################################
from xgboost import XGBClassifier
myxgb=XGBClassifier(booster='gbtree', gamma=0.1, eta=0.05,
       n_estimators=50, objective='binary:logistic',
       max_depth=2,
       reg_alpha=0, scale_pos_weight=1, verbosity=1,
       subsample=1, random_state = 66)
myxgb.fit(X_train, y_train)  #Fit on training set
plot_confusion_matrix(myxgb,X_test,y_test)
mypred=myxgb.predict(X_test)
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
##############################################################################################
```

```
C:\Users\tf\lib\site-packages\xgboost\sklearn.py:888: UserWarning: The use of label enco
der in XGBClassifier is deprecated and will be removed in a future release. To remove th
is warning, do the following: 1) Pass option use_label_encoder=False when constructing X
GBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0,
1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[14:42:42] WARNING: ..\src\learner.cc:1061: Starting in XGBoost 1.3.0, the default evalu
```

```
ation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
                    0.0          1.0  accuracy     macro avg  weighted avg
precision      0.955672     0.187300   0.71557      0.571486      0.887284
recall         0.721235     0.657586   0.71557      0.689410      0.715570
f1-score       0.822066     0.291556   0.71557      0.556811      0.774848
support    40952.000000  4001.000000   0.71557  44953.000000  44953.000000
```

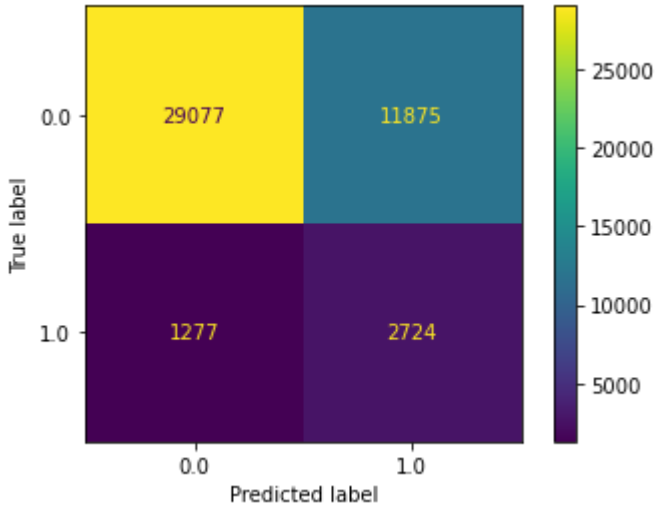


LDA

In [18]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
mylda=LDA()
mylda.fit(X_train, y_train)  # Fit on the training set
plot_confusion_matrix(mylda,X_test,y_test)
mypred=mylda.predict(X_test)
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
```

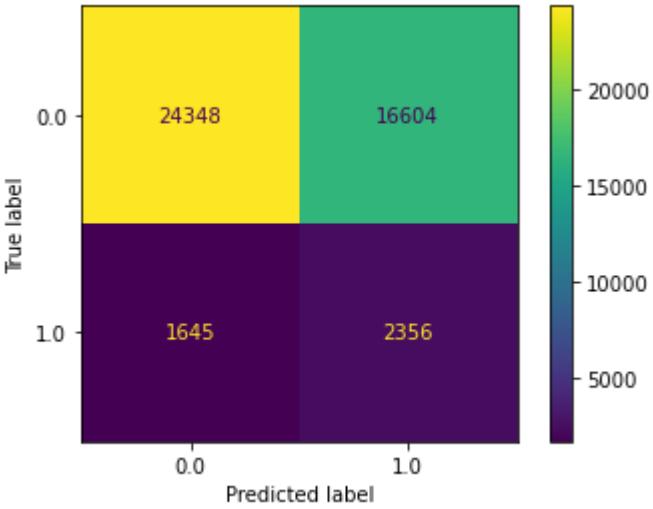
```
                    0.0          1.0  accuracy     macro avg  weighted avg
precision      0.957930     0.186588  0.707428      0.572259      0.889277
recall         0.710026     0.680830  0.707428      0.695428      0.707428
f1-score       0.815555     0.292903  0.707428      0.554229      0.769037
support    40952.000000  4001.000000  0.707428  44953.000000  44953.000000
```



QDA

In [19]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
myqda=QDA()
myqda.fit(X_train, y_train)  #Fit on training set
plot_confusion_matrix(myqda, X_test,y_test)
mypred=myqda.predict(X_test)
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
```

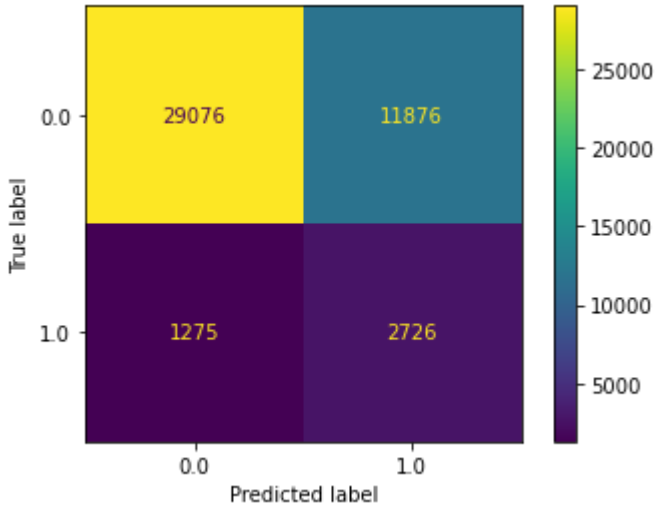
```
C:\Users\tf\lib\site-packages\sklearn\discriminant_analysis.py:715: UserWarning: Variabl
es are collinear
  warnings.warn("Variables are collinear")
                   0.0          1.0  accuracy     macro avg  weighted avg
precision     0.936714     0.124262  0.594043      0.530488      0.864402
recall        0.594550     0.588853  0.594043      0.591701      0.594043
f1-score      0.727403     0.205218  0.594043      0.466310      0.680926
support    40952.000000  4001.000000  0.594043  44953.000000  44953.000000
```



Linear Support Vector Machine

In [20]:
```
mysvm=LinearSVC(random_state=0, tol=1e-5)
mysvm.fit(X_train, y_train)  #Fit on training set
plot_confusion_matrix(mysvm, X_test,y_test)
mypred=mysvm.predict(X_test)
print(pd.DataFrame(CR(y_test, mypred, output_dict=True)))
```

```
                   0.0          1.0  accuracy     macro avg  weighted avg
precision     0.957991     0.186687   0.70745      0.572339      0.889342
recall        0.710002     0.681330   0.70745      0.695666      0.707450
f1-score      0.815562     0.293071   0.70745      0.554316      0.769058
support    40952.000000  4001.000000   0.70745  44953.000000  44953.000000
```


29076
11876
1275
2726
0.0
1.0
True label
0.0
1.0
Predicted label
25000
20000
15000
10000
5000