In [50]:

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#sklearn
import sklearn

#Data preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#metrics
from sklearn.metrics import confusion_matrix, roc_curve, accuracy_score
```

Read the data

In [52]:

```
#Read the data
HEPMASS = pd.read_csv('train_new_data.csv')
print(HEPMASS.shape)
```

```
(500000, 30)
```

Split the data : 80/20 for training and testing.

In [54]:

```

column = HEPMASS.columns

#Training data
Xtrain = HEPMASS.iloc[1:400000,:].drop(column[0:2], axis=1)
ytrain = HEPMASS[column[1]].iloc[1:400000]

#Testing data
Xtest = HEPMASS.iloc[400000:,:].drop(column[0:2], axis=1)
ytest = HEPMASS[column[1]].iloc[400000:]


```

In [55]:

```
#Preprocess the data
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
Xtrain = scaler.fit_transform(Xtrain)
Xtest  = scaler.transform(Xtest)
```

In [56]:

```
print(Xtrain.shape)
```

```
(399999, 28)
```

Neural Network

In [57]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

```

In [58]:

```
#-------Construct model-------------------#
model = Sequential()
model.add(Dense(units=512,
                activation='relu', input_dim=28))
model.add(Dropout(0.4))
model.add(Dense(units=384,
                activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=256,
                activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(units=1,
                activation='sigmoid'))
```

In [59]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 512)               14848     
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 384)               196992    
_________________________________________________________________
dropout_7 (Dropout)          (None, 384)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 256)               98560     
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 257       
=================================================================
Total params: 310,657
Trainable params: 310,657
Non-trainable params: 0
_________________________________________________________________
```

In [60]:

```
#Compile the data
model.compile(optimizer='adam',
              loss='mse',
              metrics=['accuracy'])
```

In [61]:

```python
#Train on the data
history = model.fit(
    Xtrain,
    ytrain,
    epochs=50,
    batch_size=400,
    shuffle=True,
    validation_split=0.1,
    verbose=2
)
```

```
Train on 359999 samples, validate on 40000 samples
Epoch 1/50
359999/359999 - 16s - loss: 0.1037 - accuracy: 0.8486 - val_loss: 0.0931 -
val_accuracy: 0.8663
Epoch 2/50
359999/359999 - 16s - loss: 0.0951 - accuracy: 0.8619 - val_loss: 0.0915 -
val_accuracy: 0.8688
Epoch 3/50
359999/359999 - 22s - loss: 0.0930 - accuracy: 0.8657 - val_loss: 0.0899 -
val_accuracy: 0.8705
Epoch 4/50
359999/359999 - 17s - loss: 0.0913 - accuracy: 0.8691 - val_loss: 0.0887 -
val_accuracy: 0.8739
Epoch 5/50
359999/359999 - 17s - loss: 0.0903 - accuracy: 0.8707 - val_loss: 0.0880 -
val_accuracy: 0.8745
Epoch 6/50
359999/359999 - 18s - loss: 0.0894 - accuracy: 0.8722 - val_loss: 0.0873 -
val_accuracy: 0.8757
Epoch 7/50
359999/359999 - 20s - loss: 0.0887 - accuracy: 0.8734 - val_loss: 0.0873 -
val_accuracy: 0.8756
Epoch 8/50
359999/359999 - 18s - loss: 0.0881 - accuracy: 0.8745 - val_loss: 0.0867 -
val_accuracy: 0.8774
Epoch 9/50
359999/359999 - 17s - loss: 0.0876 - accuracy: 0.8751 - val_loss: 0.0863 -
val_accuracy: 0.8777
Epoch 10/50
359999/359999 - 16s - loss: 0.0873 - accuracy: 0.8757 - val_loss: 0.0863 -
val_accuracy: 0.8776
Epoch 11/50
359999/359999 - 17s - loss: 0.0869 - accuracy: 0.8764 - val_loss: 0.0858 -
val_accuracy: 0.8785
Epoch 12/50
359999/359999 - 20s - loss: 0.0864 - accuracy: 0.8774 - val_loss: 0.0859 -
val_accuracy: 0.8788
Epoch 13/50
359999/359999 - 17s - loss: 0.0862 - accuracy: 0.8778 - val_loss: 0.0858 -
val_accuracy: 0.8777
Epoch 14/50
359999/359999 - 17s - loss: 0.0858 - accuracy: 0.8784 - val_loss: 0.0857 -
val_accuracy: 0.8787
Epoch 15/50
```

```
359999/359999 - 17s - loss: 0.0855 - accuracy: 0.8785 - val_loss: 0.0859 -
val_accuracy: 0.8780
Epoch 16/50
359999/359999 - 20s - loss: 0.0852 - accuracy: 0.8791 - val_loss: 0.0859 -
val_accuracy: 0.8774
Epoch 17/50
359999/359999 - 21s - loss: 0.0849 - accuracy: 0.8798 - val_loss: 0.0860 -
val_accuracy: 0.8787
Epoch 18/50
359999/359999 - 18s - loss: 0.0847 - accuracy: 0.8803 - val_loss: 0.0856 -
val_accuracy: 0.8793
Epoch 19/50
359999/359999 - 17s - loss: 0.0844 - accuracy: 0.8805 - val_loss: 0.0859 -
val_accuracy: 0.8781
Epoch 20/50
359999/359999 - 16s - loss: 0.0842 - accuracy: 0.8811 - val_loss: 0.0854 -
val_accuracy: 0.8784
Epoch 21/50
359999/359999 - 22s - loss: 0.0839 - accuracy: 0.8816 - val_loss: 0.0855 -
val_accuracy: 0.8779
Epoch 22/50
359999/359999 - 16s - loss: 0.0838 - accuracy: 0.8814 - val_loss: 0.0855 -
val_accuracy: 0.8788
Epoch 23/50
359999/359999 - 15s - loss: 0.0836 - accuracy: 0.8820 - val_loss: 0.0855 -
val_accuracy: 0.8785
Epoch 24/50
359999/359999 - 15s - loss: 0.0834 - accuracy: 0.8824 - val_loss: 0.0855 -
val_accuracy: 0.8786
Epoch 25/50
359999/359999 - 16s - loss: 0.0831 - accuracy: 0.8827 - val_loss: 0.0852 -
val_accuracy: 0.8792
Epoch 26/50
359999/359999 - 18s - loss: 0.0828 - accuracy: 0.8831 - val_loss: 0.0854 -
val_accuracy: 0.8795
Epoch 27/50
359999/359999 - 21s - loss: 0.0827 - accuracy: 0.8839 - val_loss: 0.0860 -
val_accuracy: 0.8779
Epoch 28/50
359999/359999 - 16s - loss: 0.0826 - accuracy: 0.8838 - val_loss: 0.0856 -
val_accuracy: 0.8774
Epoch 29/50
359999/359999 - 16s - loss: 0.0823 - accuracy: 0.8842 - val_loss: 0.0855 -
val_accuracy: 0.8783
Epoch 30/50
359999/359999 - 21s - loss: 0.0820 - accuracy: 0.8844 - val_loss: 0.0858 -
val_accuracy: 0.8778
Epoch 31/50
359999/359999 - 17s - loss: 0.0819 - accuracy: 0.8852 - val_loss: 0.0853 -
val_accuracy: 0.8798
Epoch 32/50
359999/359999 - 16s - loss: 0.0818 - accuracy: 0.8852 - val_loss: 0.0856 -
val_accuracy: 0.8777
Epoch 33/50
359999/359999 - 17s - loss: 0.0817 - accuracy: 0.8854 - val_loss: 0.0852 -
val_accuracy: 0.8782
Epoch 34/50
```

```
359999/359999 - 15s - loss: 0.0813 - accuracy: 0.8858 - val_loss: 0.0859 -
val_accuracy: 0.8787
Epoch 35/50
359999/359999 - 16s - loss: 0.0811 - accuracy: 0.8868 - val_loss: 0.0860 -
val_accuracy: 0.8786
Epoch 36/50
359999/359999 - 18s - loss: 0.0809 - accuracy: 0.8866 - val_loss: 0.0858 -
val_accuracy: 0.8790
Epoch 37/50
359999/359999 - 19s - loss: 0.0808 - accuracy: 0.8867 - val_loss: 0.0858 -
val_accuracy: 0.8789
Epoch 38/50
359999/359999 - 17s - loss: 0.0806 - accuracy: 0.8872 - val_loss: 0.0857 -
val_accuracy: 0.8781
Epoch 39/50
359999/359999 - 17s - loss: 0.0805 - accuracy: 0.8875 - val_loss: 0.0857 -
val_accuracy: 0.8781
Epoch 40/50
359999/359999 - 20s - loss: 0.0803 - accuracy: 0.8876 - val_loss: 0.0857 -
val_accuracy: 0.8787
Epoch 41/50
359999/359999 - 17s - loss: 0.0802 - accuracy: 0.8878 - val_loss: 0.0857 -
val_accuracy: 0.8795
Epoch 42/50
359999/359999 - 16s - loss: 0.0800 - accuracy: 0.8881 - val_loss: 0.0859 -
val_accuracy: 0.8777
Epoch 43/50
359999/359999 - 16s - loss: 0.0798 - accuracy: 0.8883 - val_loss: 0.0858 -
val_accuracy: 0.8785
Epoch 44/50
359999/359999 - 16s - loss: 0.0799 - accuracy: 0.8881 - val_loss: 0.0860 -
val_accuracy: 0.8779
Epoch 45/50
359999/359999 - 16s - loss: 0.0796 - accuracy: 0.8887 - val_loss: 0.0858 -
val_accuracy: 0.8789
Epoch 46/50
359999/359999 - 17s - loss: 0.0794 - accuracy: 0.8891 - val_loss: 0.0864 -
val_accuracy: 0.8776
Epoch 47/50
359999/359999 - 19s - loss: 0.0791 - accuracy: 0.8896 - val_loss: 0.0864 -
val_accuracy: 0.8778
Epoch 48/50
359999/359999 - 16s - loss: 0.0792 - accuracy: 0.8894 - val_loss: 0.0863 -
val_accuracy: 0.8781
Epoch 49/50
359999/359999 - 15s - loss: 0.0789 - accuracy: 0.8899 - val_loss: 0.0862 -
val_accuracy: 0.8781
Epoch 50/50
359999/359999 - 14s - loss: 0.0785 - accuracy: 0.8908 - val_loss: 0.0863 -
val_accuracy: 0.8779
```

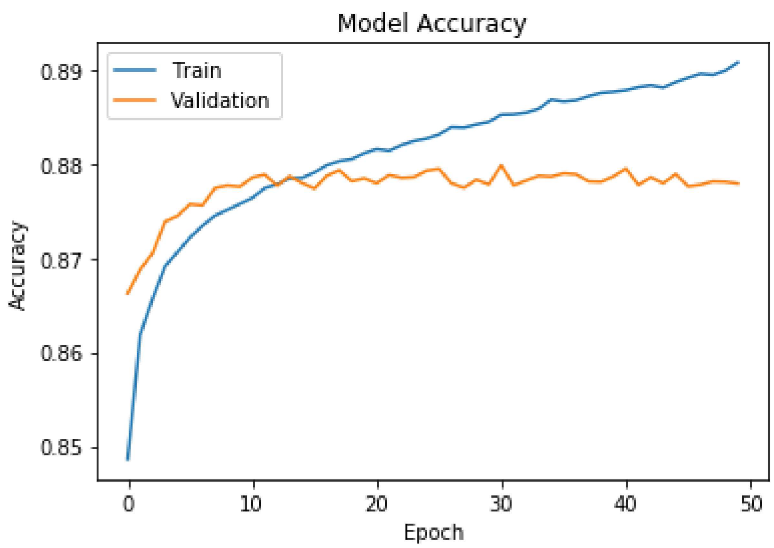
In [62]:

```
model_loss, model_accuracy = model.evaluate(
    Xtest, ytest, verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
100000/100000 - 4s - loss: 0.0858 - accuracy: 0.8778
Loss: 0.08581883184373379, Accuracy: 0.8778300285339355
```

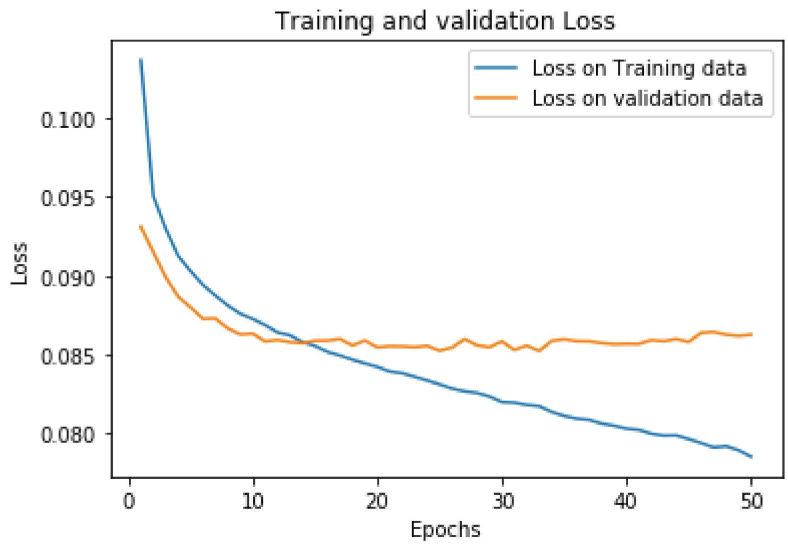
In [63]:

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc = 'upper left')
plt.show()
```



In [64]:

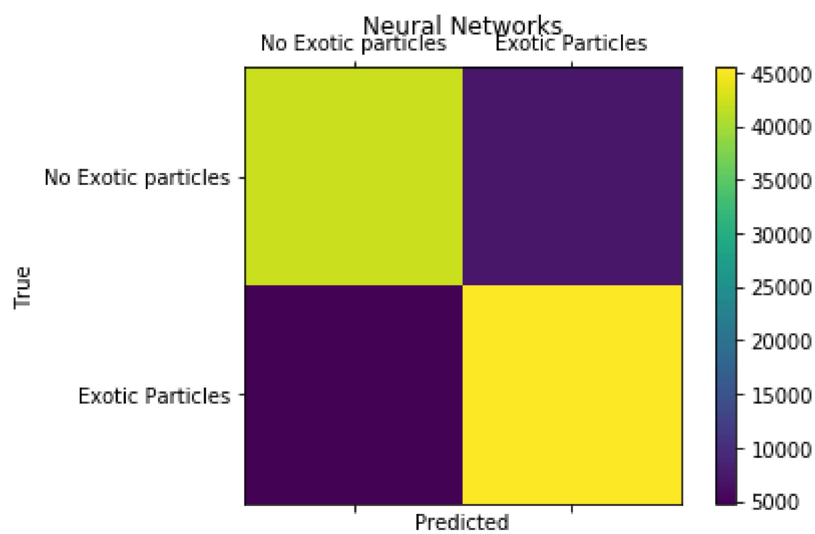
```
loss = history.history['loss']
validation_loss = history.history['val_loss']
epochs = range(1,len(loss)+1)

plt.plot(epochs, loss,label = 'Loss on Training data')
plt.plot(epochs, validation_loss,label = 'Loss on validation data')
plt.title('Training and validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [65]:

```
prediction = model.predict(Xtest)
labels = ['No Exotic particles','Exotic Particles']
conf_mat = confusion_matrix(ytest,prediction>0.5)
print(conf_mat)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat)
plt.title('Neural Networks')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
[[42291  7447]
 [ 4770 45492]]
```



Precision = TP/(TP+FP) : Proportion of correctly classified reactions where exotic particles were not generated as compared to the predicted number of positives

In [66]:

```
Precision = conf_mat[0][0]/(conf_mat[0][0]+conf_mat[0][1])
print(Precision)
```

```
0.8502754433230126
```

Recall = TP/(TP+FN) : Proportion of correctly classified reactions where exotic particles were not generated as compared to the actual number of positives

In [67]:
```
Recall = conf_mat[0][0]/(conf_mat[0][0]+conf_mat[1][0])
print(Recall)
```

```
0.8986421877988143
```

Decision Tree

In [68]:
```
#Decsion tree
from sklearn import tree

#Train Decision tree
dec_tree = sklearn.tree.DecisionTreeClassifier(max_leaf_nodes=7000, ma
dec_tree.fit(Xtrain,ytrain)

#Prediction using decision tree
dec_tree_predict = dec_tree.predict(Xtest)
dec_tree_predict_train = dec_tree.predict(Xtrain)
```

Accuracy for decision tree

In [69]:
```
#Results for Decision tree

#Training data
print('Train Accuracy: ', sklearn.metrics.accuracy_score(ytrain, dec_tr
print("Confusion Matrix")
print(confusion_matrix(ytrain, dec_tree_predict_train))
print('\n')

#Testing data
dec_conf_mat = confusion_matrix(ytest, dec_tree_predict)
print('Test Accuraccy: ', sklearn.metrics.accuracy_score(ytest, dec_tre
print("Confusion Matrix")
print(dec_conf_mat)
print('\n')
```

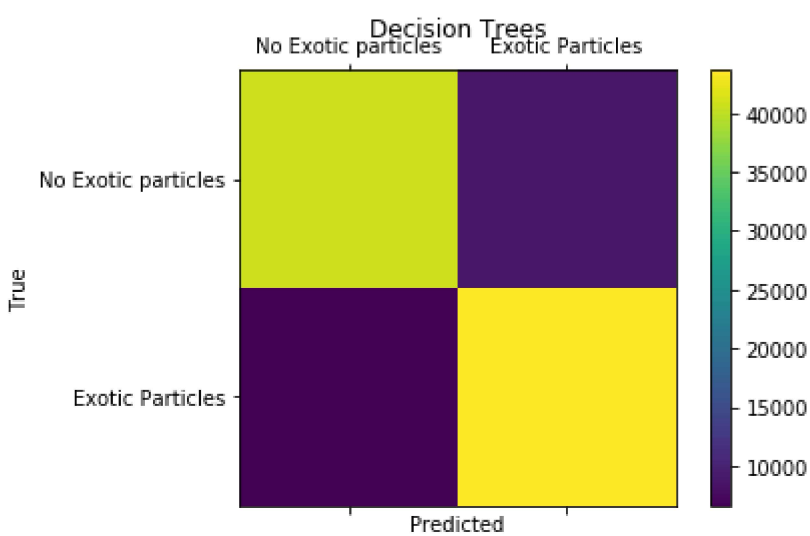
```
Train Accuracy:  0.9074022685056713
Confusion Matrix
[[177134  22842]
 [ 14197 185826]]


Test Accuraccy:  0.84571
Confusion Matrix
[[40919  8819]
 [ 6610 43652]]
```

Confusion matrix Analysis for Decision trees

In [71]:

```
print(dec_conf_mat)

labels = ['No Exotic particles','Exotic Particles']

fig = plt.figure()
ax = fig.add_subplot(111)
dec_cax = ax.matshow(dec_conf_mat)
plt.title('Confusion matrix')
fig.colorbar(dec_cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.title('Decision Trees')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

### Precision = TP/(TP+FP) : Proportion of correctly classified reactic

dec_Precision = dec_conf_mat[0][0]/(dec_conf_mat[0][0]+dec_conf_mat[0][
print("Precision "+str(dec_Precision))

### Recall  = TP/(TP+FN) : Proportion of correctly classified  reactior

dec_Recall = dec_conf_mat[0][0]/(dec_conf_mat[0][0]+dec_conf_mat[1][0])
print("Recall "+str(dec_Recall))
```

```
[[40919  8819]
 [ 6610 43652]]
```




```
Precision 0.8226909003176646
Recall 0.8609270129815481
```

Random Forest

In [45]:

```
#Random Forest
from sklearn import ensemble
rfclassifier = sklearn.ensemble.RandomForestClassifier(max_depth=15, n_

#Train Random Forest
rfclassifier.fit(Xtrain,ytrain)

#Predict on Random Forest
rfclassifier_predict = rfclassifier.predict(Xtest)
rfclassifier_predict_train = rfclassifier.predict(Xtrain)
```

Accuracy for Random Forest

In [46]:

```
#Results for Random Forest

#Training data
print('Train Accuracy: ', sklearn.metrics.accuracy_score(ytrain, rfclas
print("Confusion Matrix for training data")
print(confusion_matrix(ytrain, rfclassifier_predict_train))
print('\n')

#Testing data
rfclassifier_conf_mat = confusion_matrix(ytest, rfclassifier_predict)
print('Test Accuraccy: ', sklearn.metrics.accuracy_score(ytest, rfclass
print("Confusion Matrix for testing data")
print(rfclassifier_conf_mat)
print('\n')

```

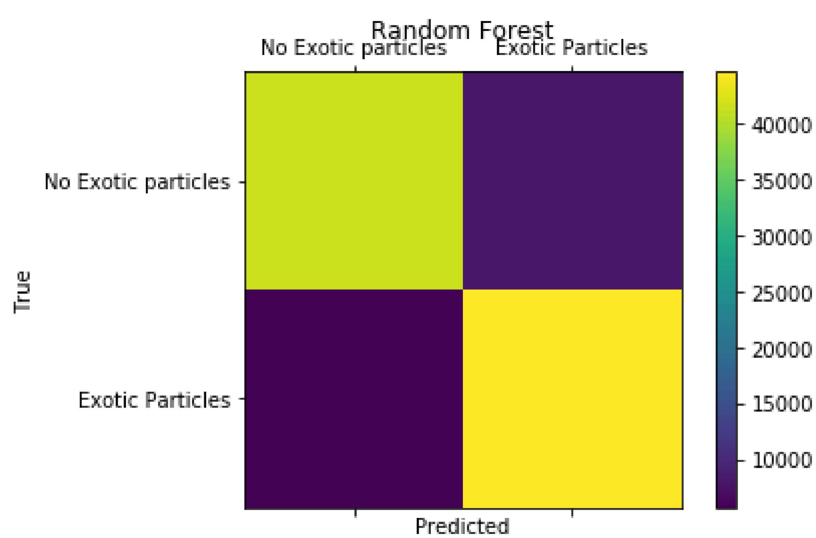
```
Train Accuracy:  0.9126872817182043
Confusion Matrix for training data
[[179619  20357]
 [ 14568 185455]]


Test Accuraccy:  0.8614
Confusion Matrix for testing data
[[41578  8160]
 [ 5700 44562]]
```

Confusion Matrix for Random Forest

In [47]:

```python
print(rfclassifier_conf_mat)

labels = ['No Exotic particles','Exotic Particles']

fig = plt.figure()
ax = fig.add_subplot(111)
rfclassifier_cax = ax.matshow(rfclassifier_conf_mat)
plt.title('Random Forest')
fig.colorbar(rfclassifier_cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

### Precision = TP/(TP+FP) : Proportion of correctly classified reactic

rfclassifier_Precision = rfclassifier_conf_mat[0][0]/(rfclassifier_conf
print("Precision"+ str(rfclassifier_Precision))

### Recall  = TP/(TP+FN) : Proportion of correctly classified  reactior

rfclassifier_Recall = rfclassifier_conf_mat[0][0]/(rfclassifier_conf_ma
print("Recall"+str(rfclassifier_Recall))
```

```
[[41578  8160]
 [ 5700 44562]]
```




```
Precision0.8359403273151312
Recall0.8794365243876644
```

In [ ]:

```

```