In [1]:
```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from keras.models import Model,load_model, Sequential
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint,TensorBoard
```

In [33]:
```
%matplotlib inline
sns.set(style='whitegrid')
```

In [34]:
```
df=pd.read_csv('C:/sameer/creditcard1.csv')
```

In [35]:
```
df = df.drop(['Time'], axis=1)
```

In [36]:
```
df['Amount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1, 1))
```

In [37]:
```
df_fraud = df[df['Class']==1]
df_normal =df[df['Class']==0]
df_normal = df_normal.sample(frac =1.0).reset_index(drop = True)
df_normal_1 = df_normal.iloc[:int(df_normal.shape[0]*0.8),:]
df_normal_2 = df_normal.iloc[:int(df_normal.shape[0]*0.8):,:]
```

In [38]:
```
x_test =pd.concat([df_fraud,df_normal_2], axis = 0)
x_test = x_test.sample(frac = 1.0).reset_index(drop = True)

#Separate in input and target variables
x_train = df_normal_1[df_normal_1['Class']== 0]
x_train = x_train.drop(['Class'], axis=1)

y_test = x_test['Class']
x_test = x_test.drop(['Class'],axis=1)
```

In [39]:
```
#Build the Neural Network
input_dim = x_train.shape[1]
encoding_dim = 14

model = Sequential()
model.add(Dense(29,input_dim = input_dim, activation="relu"))
model.add(Dense(14, activation="relu"))
model.add(Dense(7, activation="relu"))
model.add(Dense(14, activation="relu"))
model.add(Dense(input_dim, activation="sigmoid"))
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
```

In [40]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 29)                870

 dense_10 (Dense)            (None, 14)                420

 dense_11 (Dense)            (None, 7)                 105

 dense_12 (Dense)            (None, 14)                112

 dense_13 (Dense)            (None, 29)                435

=================================================================
Total params: 1942 (7.59 KB)
Trainable params: 1942 (7.59 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [41]:
```
#Fit the autoencoder and check loss for train and test
checkpointer = ModelCheckpoint(filepath="nae.h5", verbose=0, save_best_only=True)
```

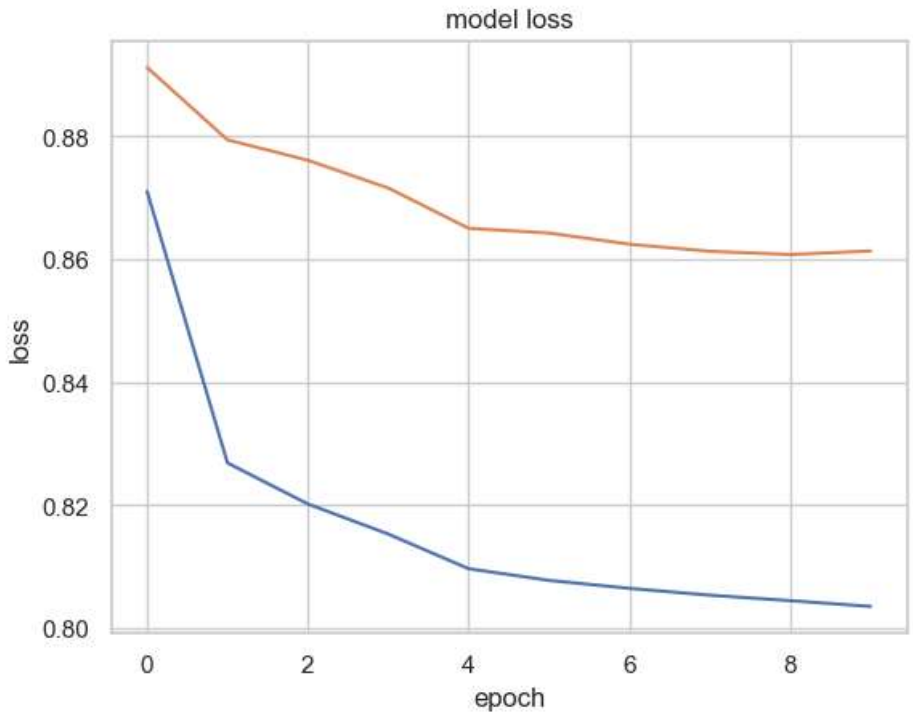
In [42]:
```
#Save history to plot learning curves
history = model.fit(x_train, x_train,
epochs=10,
batch_size=32,
shuffle=True,
validation_data=(x_test, x_test),
verbose=1,
callbacks=[checkpointer]).history

autoencoder = load_model('nae.h5')
```

```
Epoch 1/10
7108/7108 [==============================] - 62s 8ms/step - loss: 0.8710 - accuracy: 0.5202 - val_loss: 0.8912
- val_accuracy: 0.5530
Epoch 2/10
  21/7108 [..............................] - ETA: 37s - loss: 0.9002 - accuracy: 0.5729
C:\Users\samir\anaconda3\Lib\site-packages\keras\src\engine\training.py:3079: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the
native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
7108/7108 [==============================] - 52s 7ms/step - loss: 0.8269 - accuracy: 0.5632 - val_loss: 0.8794
- val_accuracy: 0.5690
Epoch 3/10
7108/7108 [==============================] - 52s 7ms/step - loss: 0.8202 - accuracy: 0.5681 - val_loss: 0.8761
- val_accuracy: 0.5617
Epoch 4/10
7108/7108 [==============================] - 53s 7ms/step - loss: 0.8153 - accuracy: 0.5779 - val_loss: 0.8716
- val_accuracy: 0.5863
Epoch 5/10
7108/7108 [==============================] - 55s 8ms/step - loss: 0.8097 - accuracy: 0.5748 - val_loss: 0.8650
- val_accuracy: 0.5739
Epoch 6/10
7108/7108 [==============================] - 52s 7ms/step - loss: 0.8078 - accuracy: 0.5772 - val_loss: 0.8643
- val_accuracy: 0.5822
Epoch 7/10
7108/7108 [==============================] - 49s 7ms/step - loss: 0.8065 - accuracy: 0.5771 - val_loss: 0.8625
- val_accuracy: 0.5845
Epoch 8/10
7108/7108 [==============================] - 50s 7ms/step - loss: 0.8054 - accuracy: 0.5778 - val_loss: 0.8613
- val_accuracy: 0.5803
Epoch 9/10
7108/7108 [==============================] - 50s 7ms/step - loss: 0.8045 - accuracy: 0.5765 - val_loss: 0.8608
- val_accuracy: 0.5747
Epoch 10/10
7108/7108 [==============================] - 52s 7ms/step - loss: 0.8035 - accuracy: 0.5711 - val_loss: 0.8614
- val_accuracy: 0.5717
```

In [43]:
```
#Plot losses
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
```

Out[43]:
```
Text(0.5, 0, 'epoch')
```



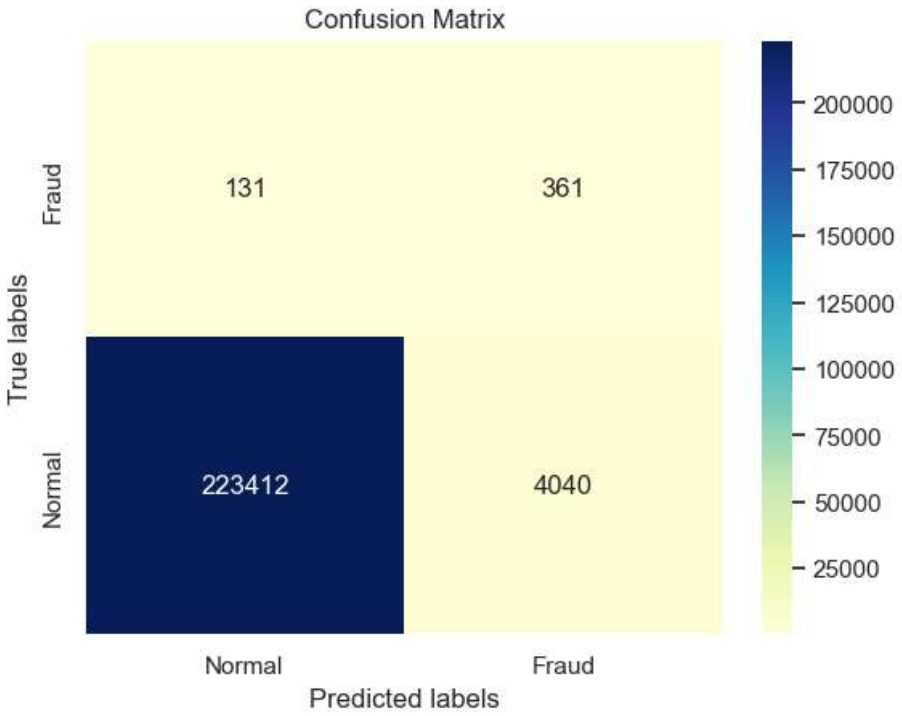
In [44]:
```
#Predict on test set
predictions = model.predict(x_test)

mse = np.mean(np.power(x_test - predictions, 2), axis=1)
error_df = pd.DataFrame({'mse': mse,'fraud':y_test})
```

```
7124/7124 [==============================] - 22s 3ms/step
```

In [45]:
```
#Set an error threshold above which a transaction is considered fraud
threshold = 4.5
error_df['pred_01'] = [1 if e> threshold else 0 for e in error_df['mse'].values]
conf_mat = confusion_matrix(error_df['fraud'], error_df['pred_01'])
```

In [46]:
```
#Print confusion matrix for the given threshold
ax=plt.subplot()
sns.heatmap(conf_mat, annot=True, fmt="g", cmap="YlGnBu")
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.set_ylim([0,2])
ax.xaxis.set_ticklabels(["Normal", "Fraud"]); ax.yaxis.set_ticklabels(["Normal","Fraud"])
```

Out[46]:
```
[Text(0, 0.5, 'Normal'), Text(0, 1.5, 'Fraud')]
```


Confusion Matrix
131
361
223412
4040
Fraud
Normal
True labels
Normal
Fraud
Predicted labels
200000
175000
150000
125000
100000
75000
50000
25000

In [ ]:

In [ ]: