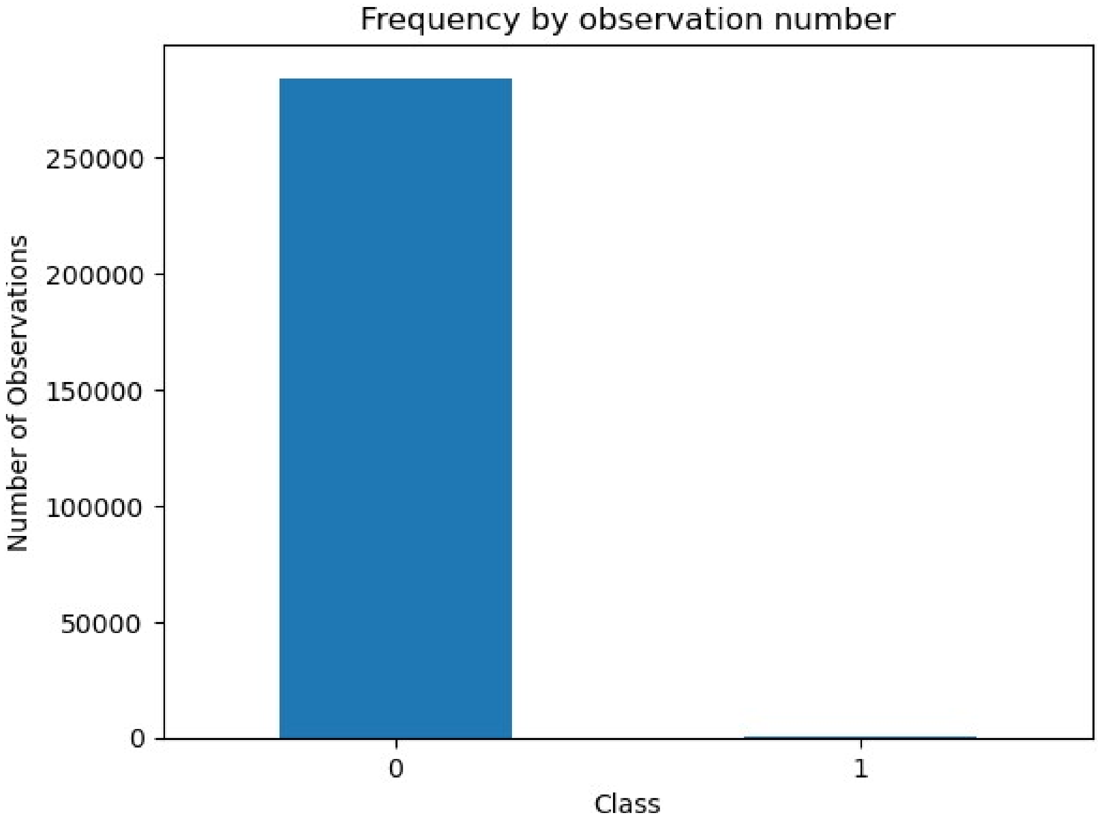
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score,
accuracy_score, precision_score

RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]

dataset = pd.read_csv("creditcard.csv")

#check for any null values
print("Any nulls in the dataset",dataset.isnull().values.any())
print('-------')
print("No. of unique labels",len(dataset['Class'].unique()))
print("Label values",dataset.Class.unique())

#0 is for normal credit card transcation
#1 is for fraudulent credit card transcation
print('-------')
print("Break down of Normal and Fraud Transcations")
print(pd.value_counts(dataset['Class'],sort=True))
```

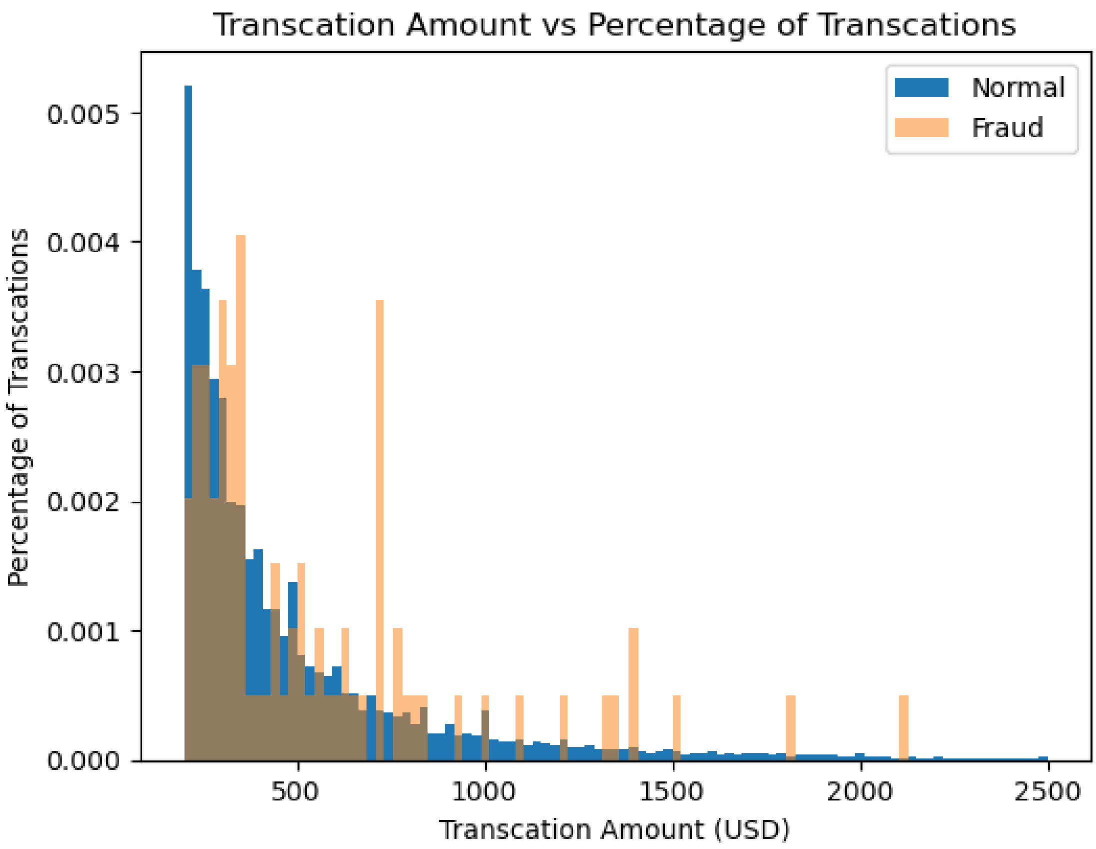
```
Any nulls in the dataset False
-------
No. of unique labels 2
Label values [0 1]
-------
Break down of Normal and Fraud Transcations
0    284315
1       492
Name: Class, dtype: int64
```

```
#visualizing the imbalanced dataset
count_classes = pd.value_counts(dataset['Class'],sort=True)
count_classes.plot(kind='bar',rot=0)
plt.xticks(range(len(dataset['Class'].unique())),dataset.Class.unique(
))
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations")
```

```
Text(0, 0.5, 'Number of Observations')
```



```
#Save the normal and fradulent transcations in seperate dataframe
normal_dataset = dataset[dataset.Class == 0]
fraud_dataset = dataset[dataset.Class == 1]

#Visualize transcation amounts for normal and fraudulent transcations
bins = np.linspace(200,2500,100)
plt.hist(normal_dataset.Amount,bins=bins,alpha=1,density=True,label='Normal')
plt.hist(fraud_dataset.Amount,bins=bins,alpha=0.5,density=True,label='Fraud')
plt.legend(loc='upper right')
plt.title("Transcation Amount vs Percentage of Transcations")
plt.xlabel("Transcation Amount (USD)")
plt.ylabel("Percentage of Transcations")
plt.show()
```



```
dataset
```

```
            Time         V1         V2        V3        V4
V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321

1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018

2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198

3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309

4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193

...          ...        ...        ...       ...       ...       ...

284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473

284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229

284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515

284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
```

```
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546


              V6        V7        V8        V9  ...       V21
V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307
0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -
0.638672
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998
0.771679
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300
0.005274
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431
0.798278
...          ...       ...       ...       ...  ...       ...       ..
.
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454
0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205
0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045
0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245
0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057
0.643078

             V23       V24       V25       V26       V27       V28
Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
69.99
...          ...       ...       ...       ...       ...       ...
...
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
```

```
10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649
217.00

        Class
0           0
1           0
2           0
3           0
4           0
...       ...
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]
```

```
sc = StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-
1,1))
dataset['Amount'] =
sc.fit_transform(dataset['Amount'].values.reshape(-1,1))

raw_data = dataset.values
#The last element contains if the transcation is normal which is
represented by 0 and if fraud then 1
labels = raw_data[:,-1]

#The other data points are the electrocadriogram data
data = raw_data[:,0:-1]

train_data,test_data,train_labels,test_labels =
train_test_split(data,labels,test_size = 0.2,random_state =2021)

min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)

train_data = tf.cast(train_data,tf.float32)
test_data = tf.cast(test_data,tf.float32)

train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

#Creating normal and fraud datasets
normal_train_data = train_data[~train_labels]
```

```
normal_test_data = test_data[~test_labels]

fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print("No. of records in Fraud Train Data=",len(fraud_train_data))
print("No. of records in Normal Train Data=",len(normal_train_data))
print("No. of records in Fraud Test Data=",len(fraud_test_data))
print("No. of records in Normal Test Data=",len(normal_test_data))
```

```
No. of records in Fraud Train Data= 389
No. of records in Normal Train Data= 227456
No. of records in Fraud Test Data= 103
No. of records in Normal Test Data= 56859
```

```
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1]
#num of columns,30
encoding_dim = 14
hidden_dim1 = int(encoding_dim / 2)
hidden_dim2 = 4
learning_rate = 1e-7

#input layer
input_layer = tf.keras.layers.Input(shape=(input_dim,))

#Encoder
encoder =
tf.keras.layers.Dense(encoding_dim,activation="tanh",activity_regulari
zer = tf.keras.regularizers.l2(learning_rate))(input_layer)
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')
(encoder)
encoder =
tf.keras.layers.Dense(hidden_dim2,activation=tf.nn.leaky_relu)
(encoder)

#Decoder
decoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')
(encoder)
decoder = tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim,activation='relu')
(decoder)
decoder = tf.keras.layers.Dense(input_dim,activation='tanh')(decoder)

#Autoencoder
autoencoder = tf.keras.Model(inputs = input_layer,outputs = decoder)
autoencoder.summary()
```

```
Model: "functional_1"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 30)]              0
_________________________________________________________________
dense (Dense)                (None, 14)                434
_________________________________________________________________
dropout (Dropout)            (None, 14)                0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 105
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 32
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 35
_________________________________________________________________
dropout_1 (Dropout)          (None, 7)                 0
_________________________________________________________________
dense_4 (Dense)              (None, 14)                112
_________________________________________________________________
dense_5 (Dense)              (None, 30)                450
=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
_________________________________________________________________
```

```
cp =
tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mod
e='min',monitor='val_loss',verbose=2,save_best_only=True)
#Define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
                monitor='val_loss',
                min_delta=0.0001,
                patience=10,
                verbose=11,
                mode='min',
                restore_best_weights=True
)

autoencoder.compile(metrics=['accuracy'],loss=
'mean_squared_error',optimizer='adam')

history = autoencoder.fit(normal_train_data,normal_train_data,epochs =
nb_epoch,
                         batch_size = batch_size,shuffle = True,
                         validation_data = (test_data,test_data),
                         verbose=1,
                         callbacks = [cp,early_stop]).history
```

```
Epoch 1/50
   1/3554 [..............................] - ETA: 0s - loss: 0.2476 -
accuracy: 0.0312WARNING:tensorflow:Callbacks method
```

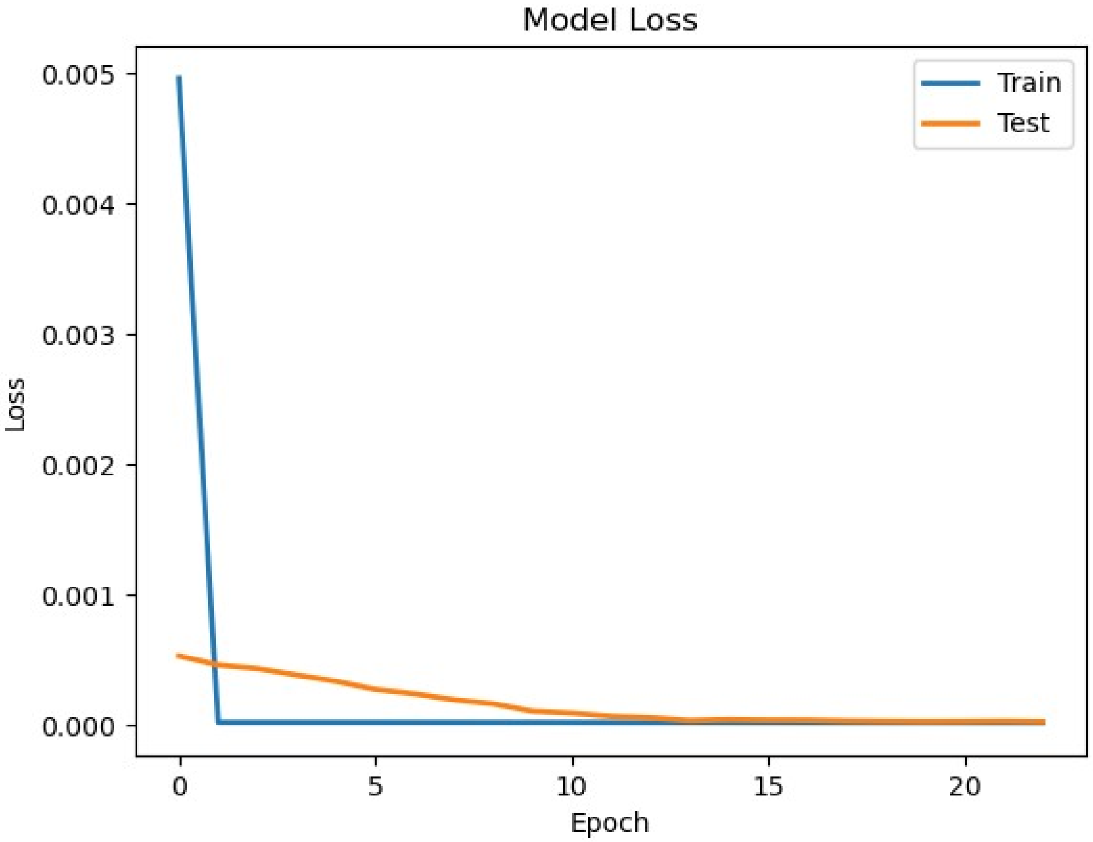
```
`on_train_batch_end` is slow compared to the batch time (batch time:
0.0000s vs `on_train_batch_end` time: 0.0010s). Check your callbacks.
3535/3554 [============================>.] - ETA: 0s - loss: 0.0050 -
accuracy: 0.0561
Epoch 00001: val_loss improved from inf to 0.00053, saving model to
autoencoder_fraud.h5
3554/3554 [==============================] - 2s 642us/step - loss:
0.0050 - accuracy: 0.0562 - val_loss: 5.2896e-04 - val_accuracy:
0.0236
Epoch 2/50
3469/3554 [============================>.] - ETA: 0s - loss: 1.9377e-
05 - accuracy: 0.0736
Epoch 00002: val_loss improved from 0.00053 to 0.00046, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 566us/step - loss:
1.9396e-05 - accuracy: 0.0737 - val_loss: 4.5975e-04 - val_accuracy:
0.0236
Epoch 3/50
3495/3554 [============================>.] - ETA: 0s - loss: 1.9433e-
05 - accuracy: 0.0639
Epoch 00003: val_loss improved from 0.00046 to 0.00043, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 592us/step - loss:
1.9440e-05 - accuracy: 0.0636 - val_loss: 4.3223e-04 - val_accuracy:
0.0236
Epoch 4/50
3508/3554 [============================>.] - ETA: 0s - loss: 1.9483e-
05 - accuracy: 0.0640
Epoch 00004: val_loss improved from 0.00043 to 0.00038, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 693us/step - loss:
1.9526e-05 - accuracy: 0.0641 - val_loss: 3.8281e-04 - val_accuracy:
0.0236
Epoch 5/50
3516/3554 [============================>.] - ETA: 0s - loss: 1.9479e-
05 - accuracy: 0.0620
Epoch 00005: val_loss improved from 0.00038 to 0.00033, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 576us/step - loss:
1.9505e-05 - accuracy: 0.0621 - val_loss: 3.3468e-04 - val_accuracy:
0.1279
Epoch 6/50
3501/3554 [============================>.] - ETA: 0s - loss: 1.9459e-
05 - accuracy: 0.0663
Epoch 00006: val_loss improved from 0.00033 to 0.00027, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 579us/step - loss:
1.9461e-05 - accuracy: 0.0664 - val_loss: 2.7270e-04 - val_accuracy:
0.1279
Epoch 7/50
```

```
3451/3554 [============================>.] - ETA: 0s - loss: 1.9352e-
05 - accuracy: 0.0638
Epoch 00007: val_loss improved from 0.00027 to 0.00024, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 600us/step - loss:
1.9421e-05 - accuracy: 0.0643 - val_loss: 2.3810e-04 - val_accuracy:
0.0251
Epoch 8/50
3503/3554 [============================>.] - ETA: 0s - loss: 1.9365e-
05 - accuracy: 0.0710
Epoch 00008: val_loss improved from 0.00024 to 0.00019, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 566us/step - loss:
1.9355e-05 - accuracy: 0.0708 - val_loss: 1.9277e-04 - val_accuracy:
0.0251
Epoch 9/50
3536/3554 [============================>.] - ETA: 0s - loss: 1.9274e-
05 - accuracy: 0.0710
Epoch 00009: val_loss improved from 0.00019 to 0.00016, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 575us/step - loss:
1.9277e-05 - accuracy: 0.0711 - val_loss: 1.6159e-04 - val_accuracy:
0.0251
Epoch 10/50
3470/3554 [============================>.] - ETA: 0s - loss: 1.9170e-
05 - accuracy: 0.0754
Epoch 00010: val_loss improved from 0.00016 to 0.00011, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 598us/step - loss:
1.9138e-05 - accuracy: 0.0758 - val_loss: 1.0516e-04 - val_accuracy:
0.0251
Epoch 11/50
3511/3554 [============================>.] - ETA: 0s - loss: 1.8960e-
05 - accuracy: 0.0844
Epoch 00011: val_loss improved from 0.00011 to 0.00009, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 689us/step - loss:
1.8965e-05 - accuracy: 0.0843 - val_loss: 9.0067e-05 - val_accuracy:
0.0252
Epoch 12/50
3502/3554 [============================>.] - ETA: 0s - loss: 1.8767e-
05 - accuracy: 0.0838
Epoch 00012: val_loss improved from 0.00009 to 0.00007, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 606us/step - loss:
1.8748e-05 - accuracy: 0.0845 - val_loss: 6.6048e-05 - val_accuracy:
0.0252
Epoch 13/50
3449/3554 [============================>.] - ETA: 0s - loss: 1.8550e-
05 - accuracy: 0.0849
```

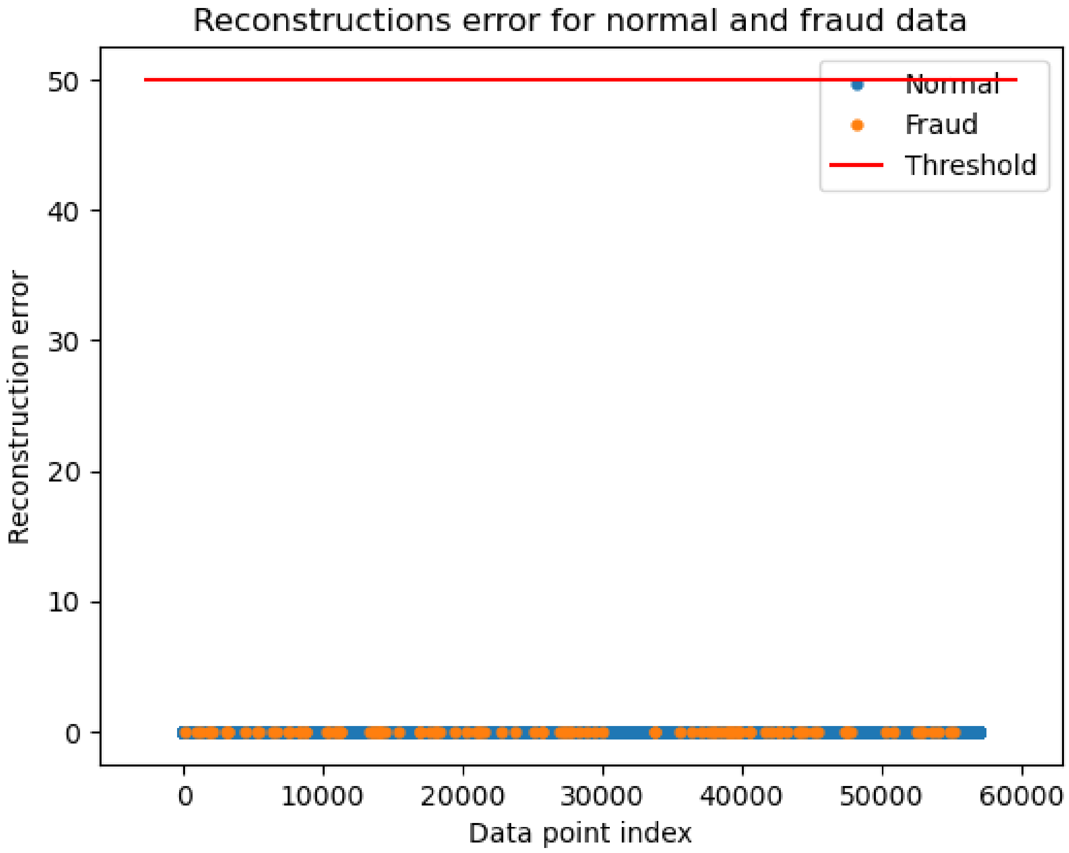
```
Epoch 00013: val_loss improved from 0.00007 to 0.00005, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 599us/step - loss:
1.8576e-05 - accuracy: 0.0855 - val_loss: 5.4927e-05 - val_accuracy:
0.0253
Epoch 14/50
3500/3554 [============================>.] - ETA: 0s - loss: 1.8386e-
05 - accuracy: 0.0892
Epoch 00014: val_loss improved from 0.00005 to 0.00003, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 578us/step - loss:
1.8389e-05 - accuracy: 0.0892 - val_loss: 3.3855e-05 - val_accuracy:
0.0253
Epoch 15/50
3517/3554 [============================>.] - ETA: 0s - loss: 1.8035e-
05 - accuracy: 0.0925
Epoch 00015: val_loss did not improve from 0.00003
3554/3554 [==============================] - 2s 560us/step - loss:
1.8029e-05 - accuracy: 0.0924 - val_loss: 4.0921e-05 - val_accuracy:
0.0253
Epoch 16/50
3535/3554 [============================>.] - ETA: 0s - loss: 1.7700e-
05 - accuracy: 0.1055
Epoch 00016: val_loss did not improve from 0.00003
3554/3554 [==============================] - 2s 572us/step - loss:
1.7701e-05 - accuracy: 0.1056 - val_loss: 3.6986e-05 - val_accuracy:
0.0252
Epoch 17/50
3488/3554 [============================>.] - ETA: 0s - loss: 1.7497e-
05 - accuracy: 0.1210
Epoch 00017: val_loss did not improve from 0.00003
3554/3554 [==============================] - 2s 562us/step - loss:
1.7491e-05 - accuracy: 0.1214 - val_loss: 3.6741e-05 - val_accuracy:
0.0252
Epoch 18/50
3494/3554 [============================>.] - ETA: 0s - loss: 1.7332e-
05 - accuracy: 0.1364
Epoch 00018: val_loss improved from 0.00003 to 0.00003, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 585us/step - loss:
1.7327e-05 - accuracy: 0.1367 - val_loss: 3.0728e-05 - val_accuracy:
0.0253
Epoch 19/50
3523/3554 [============================>.] - ETA: 0s - loss: 1.7232e-
05 - accuracy: 0.1513
Epoch 00019: val_loss improved from 0.00003 to 0.00003, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 691us/step - loss:
1.7230e-05 - accuracy: 0.1514 - val_loss: 2.7787e-05 - val_accuracy:
0.0253
```

```
Epoch 20/50
3499/3554 [============================>.] - ETA: 0s - loss: 1.7147e-
05 - accuracy: 0.1674
Epoch 00020: val_loss improved from 0.00003 to 0.00003, saving model
to autoencoder_fraud.h5
3554/3554 [==============================] - 2s 594us/step - loss:
1.7145e-05 - accuracy: 0.1674 - val_loss: 2.5756e-05 - val_accuracy:
0.0253
Epoch 21/50
3524/3554 [============================>.] - ETA: 0s - loss: 1.7070e-
05 - accuracy: 0.1872
Epoch 00021: val_loss did not improve from 0.00003
3554/3554 [==============================] - 2s 559us/step - loss:
1.7076e-05 - accuracy: 0.1871 - val_loss: 2.6697e-05 - val_accuracy:
0.0252
Epoch 22/50
3489/3554 [============================>.] - ETA: 0s - loss: 1.7009e-
05 - accuracy: 0.2023
Epoch 00022: val_loss did not improve from 0.00003
3554/3554 [==============================] - 2s 622us/step - loss:
1.7025e-05 - accuracy: 0.2026 - val_loss: 2.8282e-05 - val_accuracy:
0.0251
Epoch 23/50
3539/3554 [============================>.] - ETA: 0s - loss: 1.6937e-
05 - accuracy: 0.2238
Epoch 00023: val_loss improved from 0.00003 to 0.00002, saving model
to autoencoder_fraud.h5
Restoring model weights from the end of the best epoch.
3554/3554 [==============================] - 2s 625us/step - loss:
1.6938e-05 - accuracy: 0.2240 - val_loss: 2.4854e-05 - val_accuracy:
0.0251
Epoch 00023: early stopping
```

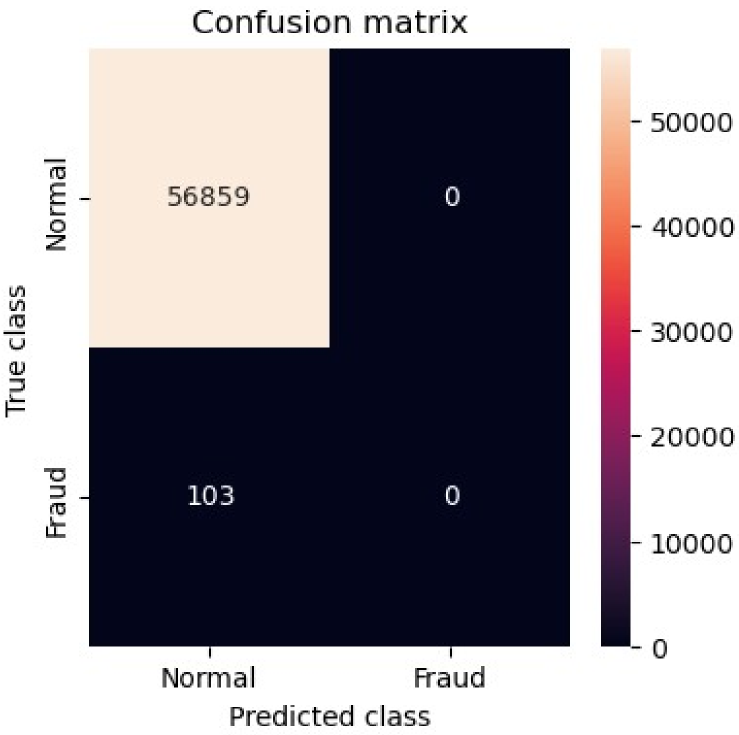
```
plt.plot(history['loss'],linewidth = 2,label = 'Train')
plt.plot(history['val_loss'],linewidth = 2,label = 'Test')
plt.legend(loc='upper right')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')

#plt.ylim(ymin=0.70,ymax=1)

plt.show()
```



```
test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2),axis = 1)
error_df = pd.DataFrame({'Reconstruction_error':mse,
                         'True_class':test_labels})

threshold_fixed = 50
groups = error_df.groupby('True_class')
fig,ax = plt.subplots()

for name,group in groups:
        ax.plot(group.index,group.Reconstruction_error,marker='o',ms =
3.5,linestyle='',
                label = "Fraud" if  name==1 else "Normal")
ax.hlines(threshold_fixed,ax.get_xlim()[0],ax.get_xlim()
[1],colors="r",zorder=100,label="Threshold")
ax.legend()
plt.title("Reconstructions error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```



```
threshold_fixed = 52
pred_y = [1 if e > threshold_fixed else 0
          for e in
        error_df.Reconstruction_error.values]
error_df['pred'] = pred_y
conf_matrix = confusion_matrix(error_df.True_class,pred_y)

plt.figure(figsize = (4,4))
sns.heatmap(conf_matrix,xticklabels = LABELS,yticklabels =
LABELS,annot = True,fmt="d")
plt.title("Confusion matrix")
plt.ylabel("True class")
plt.xlabel("Predicted class")
plt.show()

#Print Accuracy,Precision and Recall
print("Accuracy :",accuracy_score(error_df['True_class'],error_df['pre
d']))
print("Recall :",recall_score(error_df['True_class'],error_df['pred'])
)
print("Precision :",precision_score(error_df['True_class'],error_df['p
red']))
```



```
Accuracy : 0.9981917769741231
Recall : 0.0
Precision : 0.0
```

```
C:\Users\Manish\.conda\envs\tensorflow\lib\site-packages\sklearn\
metrics\_classification.py:1318: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```