

In []:

```
# Harshal Nahire
# Roll NO-46
# BE-IT
# Assignment No-4
```

In [1]:

```
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"]
```

In [8]:

```
dataset = pd.read_csv("C:\\Users\\nahir\\Desktop\\creditcard.csv")
```

In [9]:

```
#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

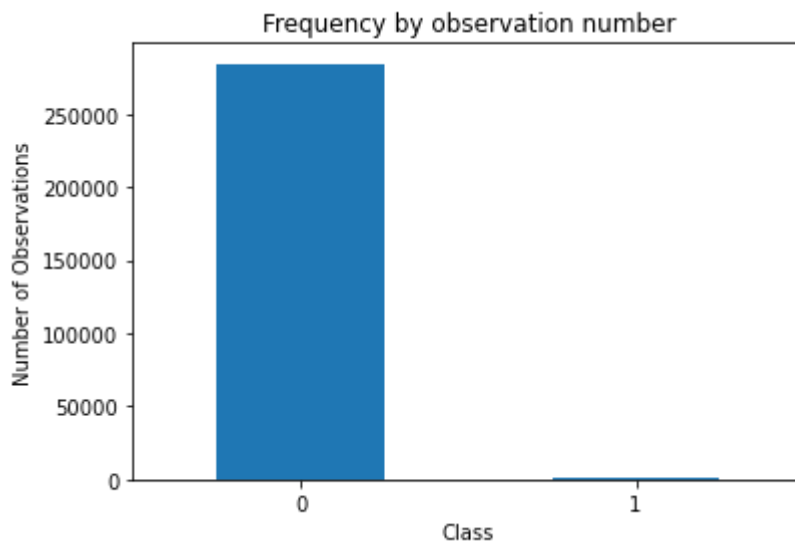
Name: Class, dtype: int64

In [10]:

```
#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")
```

Out[10]:

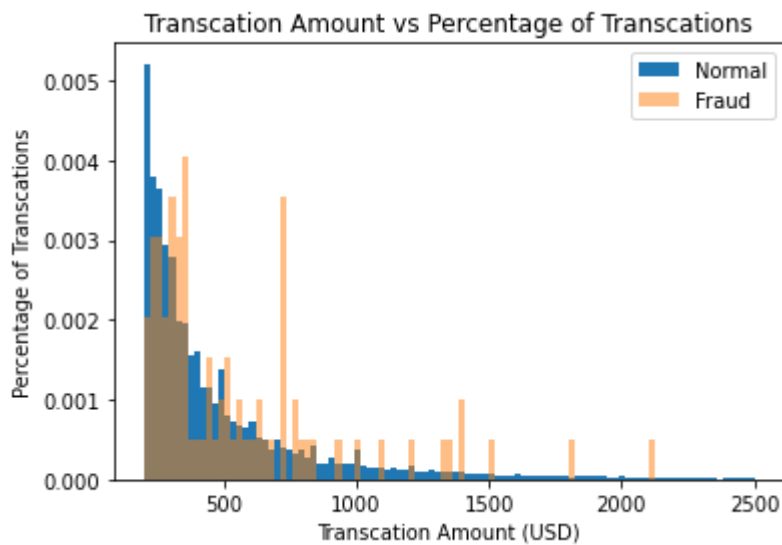
Text(0, 0.5, 'Number of Observations')



In [11]:

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

#Visualize transction amounts for normal and fraudulent transctions
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()
```



In [12]:

dataset

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941
...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

284807 rows × 31 columns

In [13]:

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

In [14]:

```
ues
ains if the transcation is normal which is represented by 0 and if fraud then 1
]

; are the electrocadriogram data
.]

rain_labels,test_labels = train_test_split(data,labels,test_size = 0.2,random_state =2021)
```

In [15]:

```
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)
```

In [16]:

```
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
```

In [17]:

```
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
```

In [18]:

```

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

#Encoder
encoder = tf.keras.layers.Dense(encoding_dim,activation="tanh",activity_regularizer = tf.ke
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: "model"

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		

In [19]:

```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mode='min',monitor=  
#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  
)
```

In [20]:

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

In [21]:

```
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

3542/3554 [=====>.] - ETA: 0s - loss: 0.0040 - accuracy: 0.0669

Epoch 1: val_loss improved from inf to 0.00003, saving model to autoencoder_fraud.h5

3554/3554 [=====] - 13s 3ms/step - loss: 0.0039 - accuracy: 0.0670 - val_loss: 2.7461e-05 - val_accuracy: 0.0051

Epoch 2/50

3554/3554 [=====] - ETA: 0s - loss: 1.9426e-05 - accuracy: 0.0640

Epoch 2: val_loss improved from 0.00003 to 0.00002, saving model to autoencoder_fraud.h5

3554/3554 [=====] - 14s 4ms/step - loss: 1.9426e-05 - accuracy: 0.0640 - val_loss: 2.3022e-05 - val_accuracy: 0.0343

Epoch 3/50

3544/3554 [=====>.] - ETA: 0s - loss: 1.9457e-05 - accuracy: 0.0625

Epoch 3: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5

3554/3554 [=====] - 11s 3ms/step - loss: 1.9463e-05 - accuracy: 0.0624 - val_loss: 2.1226e-05 - val_accuracy: 0.0051

Epoch 4/50

3531/3554 [=====>.] - ETA: 0s - loss: 1.9449e-05 - accuracy: 0.0719

Epoch 4: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5

3554/3554 [=====] - 8s 2ms/step - loss: 1.9451e-05 - accuracy: 0.0723 - val_loss: 1.9882e-05 - val_accuracy: 0.0065

Epoch 5/50

3547/3554 [=====>.] - ETA: 0s - loss: 1.9044e-05 - accuracy: 0.1186

Epoch 5: val_loss did not improve from 0.00002

3554/3554 [=====] - 13s 4ms/step - loss: 1.9044e-05 - accuracy: 0.1184 - val_loss: 2.0042e-05 - val_accuracy: 0.0420

Epoch 6/50

3524/3554 [=====>.] - ETA: 0s - loss: 1.9336e-05 - accuracy: 0.0791

Epoch 6: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5

3554/3554 [=====] - 12s 3ms/step - loss: 1.9327e-05 - accuracy: 0.0798 - val_loss: 1.7985e-05 - val_accuracy: 0.2535

Epoch 7/50

3523/3554 [=====>.] - ETA: 0s - loss: 1.8222e-05 - accuracy: 0.1835

Epoch 7: val_loss did not improve from 0.00002

3554/3554 [=====] - 8s 2ms/step - loss: 1.8213e-05 - accuracy: 0.1839 - val_loss: 1.8441e-05 - val_accuracy: 0.0610

Epoch 8/50

3549/3554 [=====>.] - ETA: 0s - loss: 1.7311e-05 - accuracy: 0.2530

Epoch 8: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5


```

3554/3554 [=====] - 9s 3ms/step - loss: 1.7307e-05
- accuracy: 0.2530 - val_loss: 1.7317e-05 - val_accuracy: 0.2699
Epoch 9/50
3543/3554 [=====>.] - ETA: 0s - loss: 1.7062e-05 - ac
curacy: 0.2604
Epoch 9: val_loss did not improve from 0.00002
3554/3554 [=====] - 9s 3ms/step - loss: 1.7062e-05
- accuracy: 0.2604 - val_loss: 1.7320e-05 - val_accuracy: 0.2685
Epoch 10/50
3528/3554 [=====>.] - ETA: 0s - loss: 1.6936e-05 - ac
curacy: 0.2665
Epoch 10: val_loss improved from 0.00002 to 0.00002, saving model to autoenc
oder_fraud.h5
3554/3554 [=====] - 10s 3ms/step - loss: 1.6937e-05
- accuracy: 0.2664 - val_loss: 1.7288e-05 - val_accuracy: 0.2736
Epoch 11/50
3542/3554 [=====>.] - ETA: 0s - loss: 1.6828e-05 - ac
curacy: 0.2695
Epoch 11: val_loss improved from 0.00002 to 0.00002, saving model to autoenc
oder_fraud.h5
Restoring model weights from the end of the best epoch: 1.
3554/3554 [=====] - 9s 2ms/step - loss: 1.6820e-05
- accuracy: 0.2697 - val_loss: 1.6838e-05 - val_accuracy: 0.2946
Epoch 11: early stopping

```

In [23]:

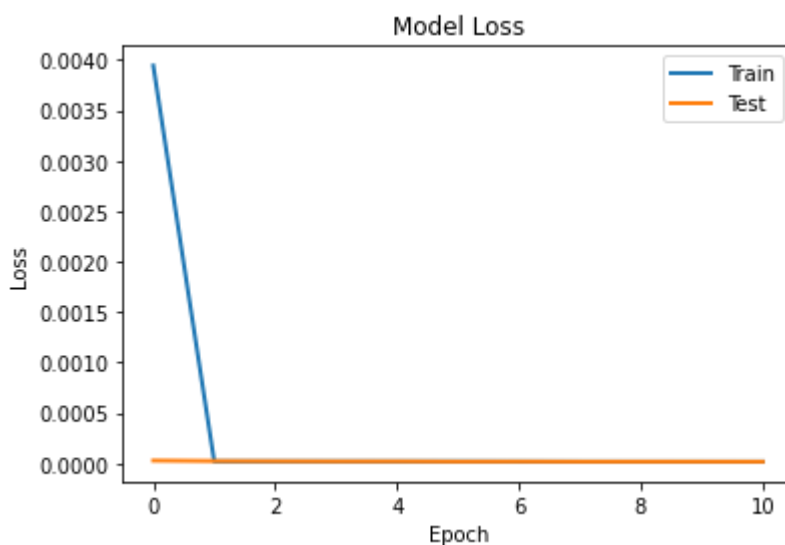
```

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()

```



In [24]:

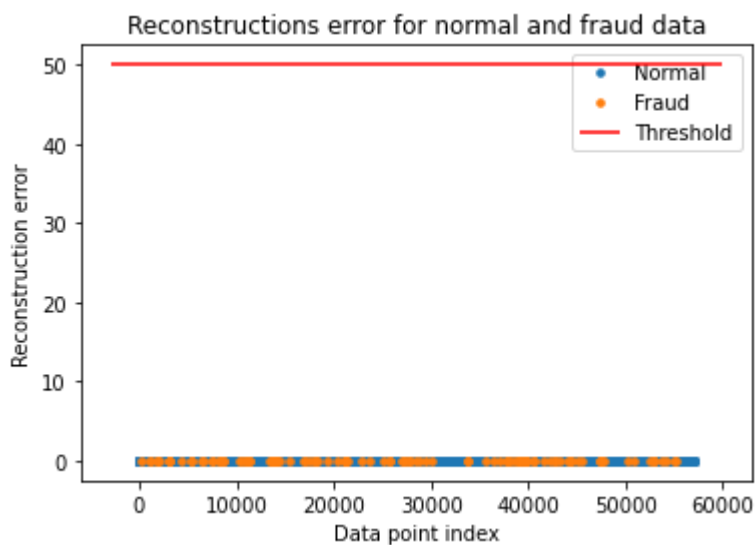
```
test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2),axis = 1)
error_df = pd.DataFrame({'Reconstruction_error':mse,
                        'True_class':test_labels})
```

1781/1781 [=====] - 5s 2ms/step

In [25]:

```
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="Th")
ax.legend()
plt.title("Reconstructions error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```



In [26]:

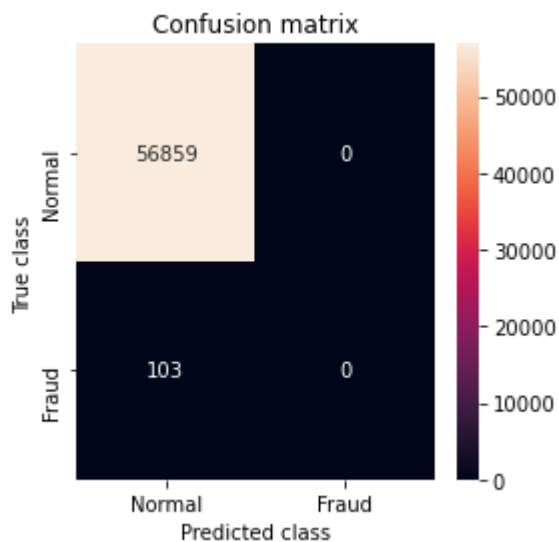
```

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['pred']))
print("Recall :",recall_score(error_df['True_class'],error_df['pred']))
print("Precision :",precision_score(error_df['True_class'],error_df['pred']))

```



Accuracy : 0.9981917769741231

Recall : 0.0

Precision : 0.0

C:\Users\nahir\anaconda3\lib\site-packages\sklearn\metrics_classification.p
y:1245: 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))
```

In []: