In [5]:
```python
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 [7]:
```python
dataset = pd.read_csv("creditcard.csv")
```

In [9]:
```python
dataset.head()
```

Out[9]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141

5 rows × 31 columns

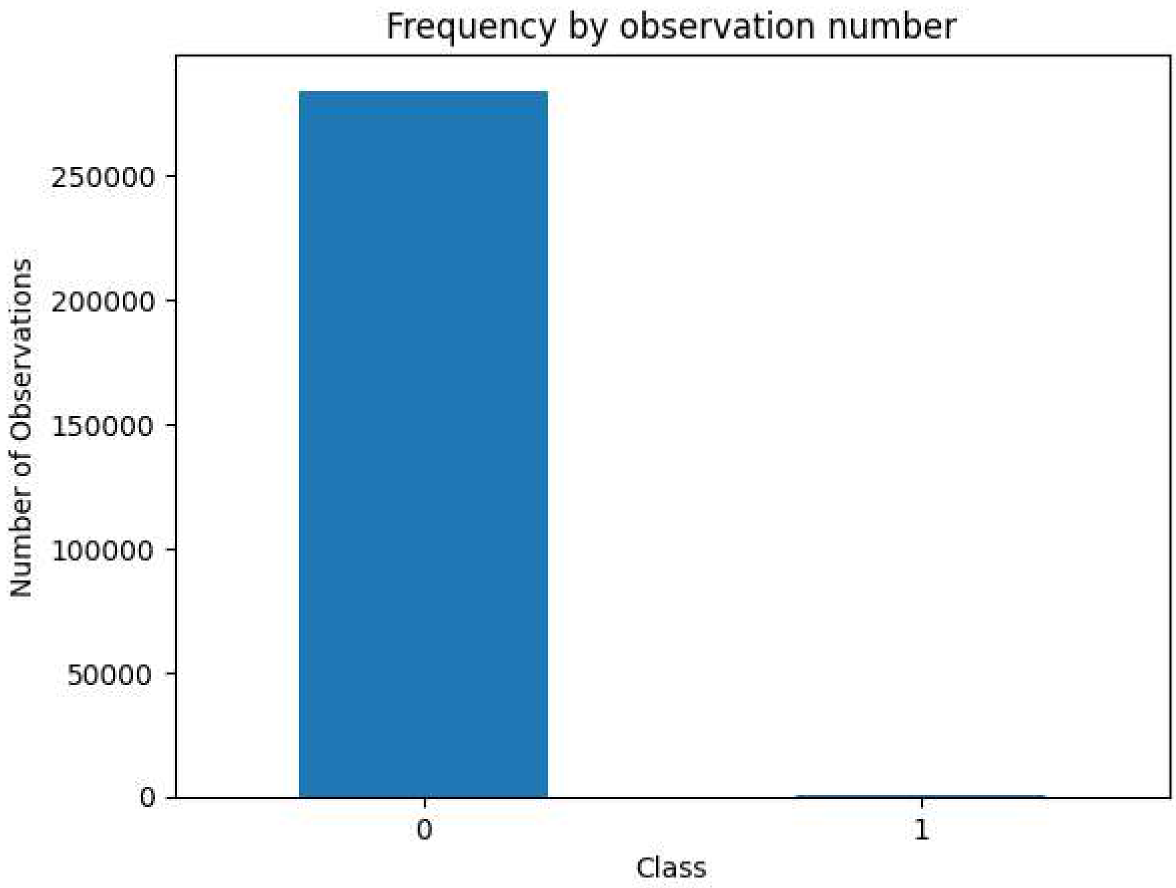
In [10]:
```
#check for any null values
print("Any nulls in the dataset",dataset.isnull().values.any())
print('-------')
print("No. of unique labels",len(dataset['Class'].unique()))
print("Label values",dataset.Class.unique())

#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
```

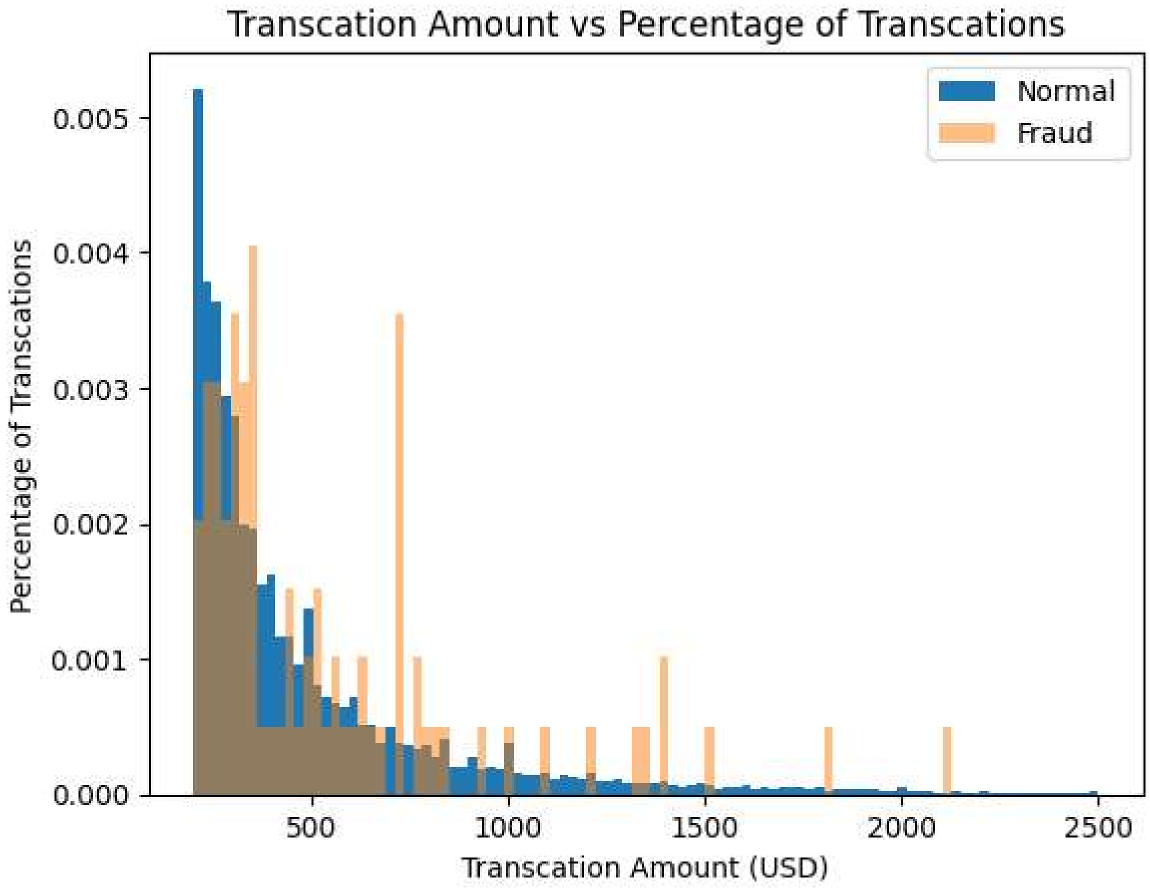
In [11]:
```
#visualizing the imbalanced dataset
count_classes = pd.value_counts(dataset['Class'],sort=True)
count_classes.plot(kind='bar',rot=0)
plt.xticks(range(len(dataset['Class'].unique())),dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations")
```

Out[11]:
```
Text(0, 0.5, 'Number of Observations')
```



In [12]:
```
#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()
```



In [13]:
```
dataset
```

Out[13]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.1104
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.1012
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.9094
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.1903
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.1374
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.0144
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.0124
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.0375
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.1632
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.3767

284807 rows × 31 columns

In [14]:
```
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 [15]:
```
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)
```

In [16]:
```
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 [17]:
```
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 [18]:
```
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 [19]:
```python
#input Layer
input_layer = tf.keras.layers.Input(shape=(input_dim,))

#Encoder
encoder = tf.keras.layers.Dense(encoding_dim,activation="tanh",activity_regularizer = tf.keras.regularizers.l2(learning_r
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 [20]:
```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mode='min',monitor='val_loss',verbose=2,save_bes
#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 [21]:
```
autoencoder.compile(metrics=['accuracy'],loss= 'mean_squared_error',optimizer='adam')
```

In [24]:
```python
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
3520/3554 [============================>.] - ETA: 0s - loss: 1.9600e-05 - accuracy: 0.0634
Epoch 1: val_loss did not improve from 0.00002
3554/3554 [==============================] - 6s 2ms/step - loss: 1.9587e-05 - accuracy: 0.0636 - val_loss: 2.0023e-05 -
val_accuracy: 0.2193
Epoch 2/50
3553/3554 [============================>.] - ETA: 0s - loss: 1.9276e-05 - accuracy: 0.0983
Epoch 2: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9275e-05 - accuracy: 0.0984 - val_loss: 1.9358e-05 -
val_accuracy: 0.0596
Epoch 3/50
3522/3554 [============================>.] - ETA: 0s - loss: 1.9007e-05 - accuracy: 0.1196
Epoch 3: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.8991e-05 - accuracy: 0.1193 - val_loss: 2.0117e-05 -
val_accuracy: 0.0109
Epoch 4/50
3540/3554 [============================>.] - ETA: 0s - loss: 1.9551e-05 - accuracy: 0.0611
Epoch 4: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9543e-05 - accuracy: 0.0610 - val_loss: 2.0157e-05 -
val_accuracy: 0.0661
Epoch 5/50
3536/3554 [============================>.] - ETA: 0s - loss: 1.9626e-05 - accuracy: 0.0612
Epoch 5: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9615e-05 - accuracy: 0.0613 - val_loss: 2.0241e-05 -
val_accuracy: 0.0214
Epoch 6/50
3530/3554 [============================>.] - ETA: 0s - loss: 1.9608e-05 - accuracy: 0.0603
Epoch 6: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9595e-05 - accuracy: 0.0604 - val_loss: 2.0390e-05 -
val_accuracy: 0.2168
Epoch 7/50
3543/3554 [============================>.] - ETA: 0s - loss: 1.9561e-05 - accuracy: 0.0599
Epoch 7: val_loss did not improve from 0.00002
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9569e-05 - accuracy: 0.0599 - val_loss: 2.0156e-05 -
val_accuracy: 0.0269
```

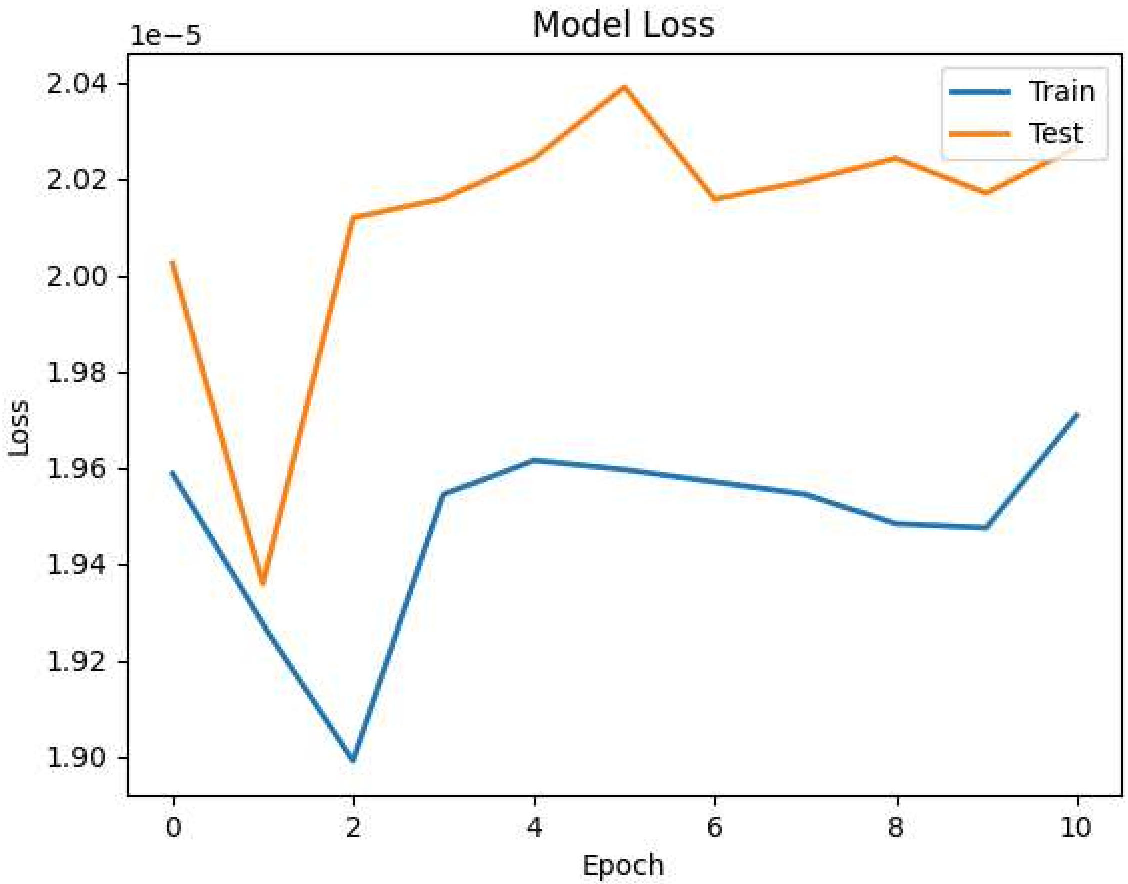
```
Epoch 8/50
3527/3554 [============================>.] - ETA: 0s - loss: 1.9553e-05 - accuracy: 0.0616
Epoch 8: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9544e-05 - accuracy: 0.0617 - val_loss: 2.0194e-05 -
val_accuracy: 0.0078
Epoch 9/50
3536/3554 [============================>.] - ETA: 0s - loss: 1.9472e-05 - accuracy: 0.0615
Epoch 9: val_loss did not improve from 0.00002
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9482e-05 - accuracy: 0.0614 - val_loss: 2.0241e-05 -
val_accuracy: 0.0814
Epoch 10/50
3527/3554 [============================>.] - ETA: 0s - loss: 1.9447e-05 - accuracy: 0.0610
Epoch 10: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9474e-05 - accuracy: 0.0608 - val_loss: 2.0169e-05 -
val_accuracy: 0.0051
Epoch 11/50
3540/3554 [============================>.] - ETA: 0s - loss: 1.9698e-05 - accuracy: 0.0628
Epoch 11: val_loss did not improve from 0.00002
Restoring model weights from the end of the best epoch: 1.
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9708e-05 - accuracy: 0.0627 - val_loss: 2.0264e-05 -
val_accuracy: 0.0111
Epoch 11: early stopping
```

In [35]:

```
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()
```



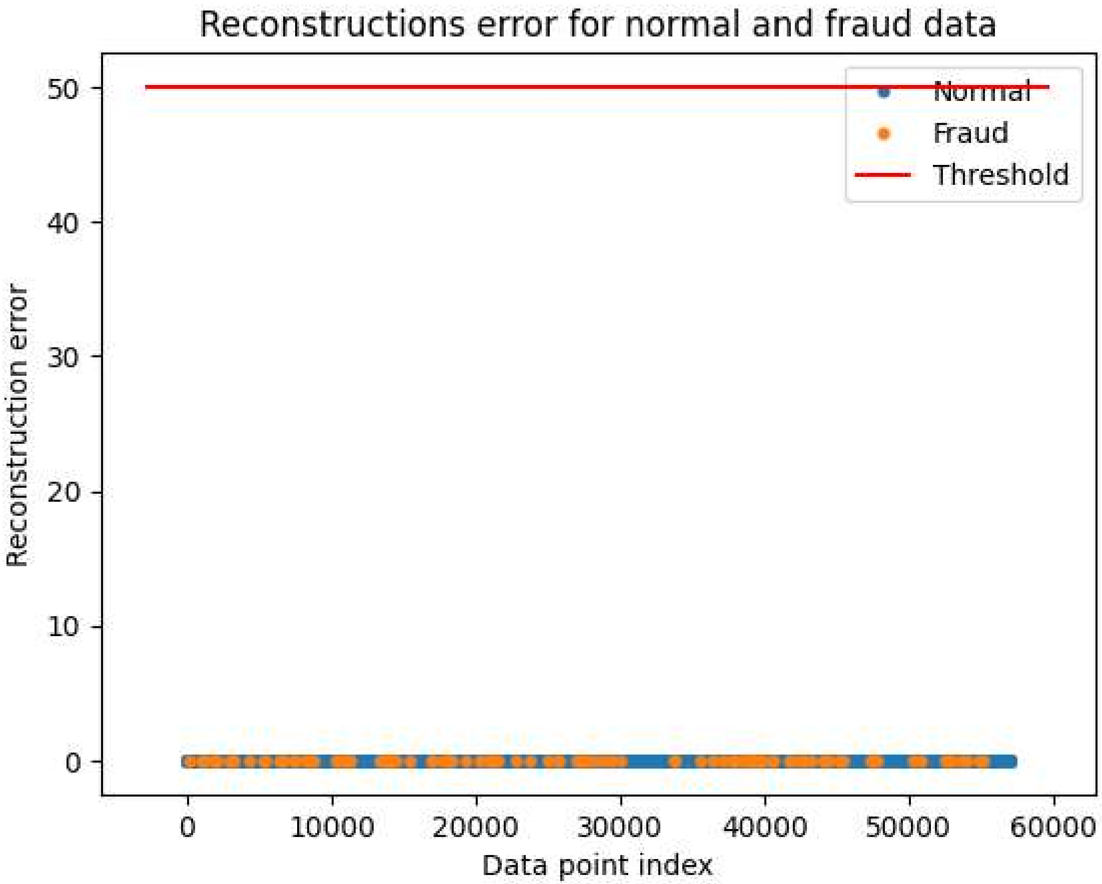
In [28]:

```
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 [==============================] - 2s 1ms/step
```

In [36]:
```
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()
```

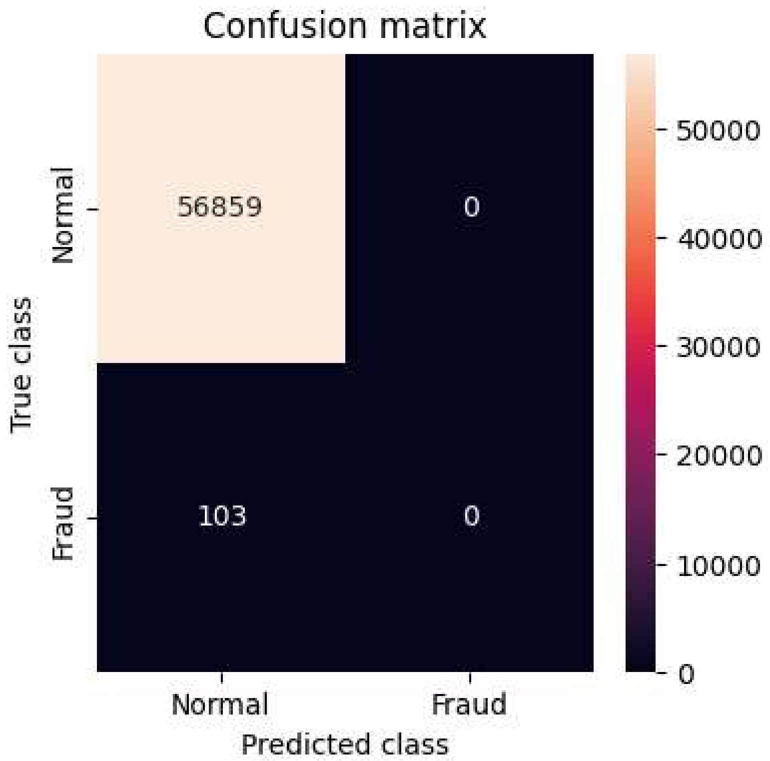


In [37]:

```
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\Admin\.conda\envs\LP4_Practical\lib\site-packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarn
ing: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to contro
l this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]: