In [ ]:
```
#Name:Robinkumar V Singh
#Roll No:76
```

In [ ]:
```
""""
Use Autoencoder to implement anomaly detection. Build the model by using:
a. Import required libraries
b. Upload / access the dataset
c. Encoder converts it into latent representation
d. Decoder networks convert it back to the original input
e. Compile the models with Optimizer, Loss, and Evaluation Metrics
""""
```

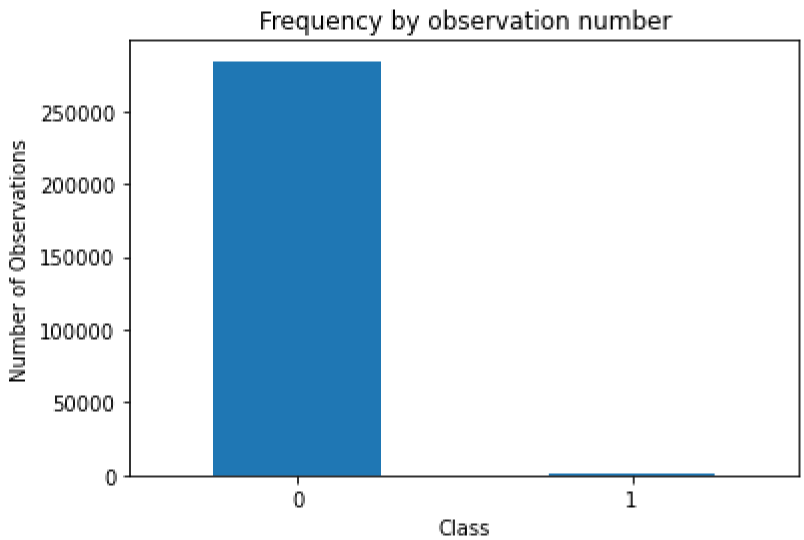
In [6]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib . pyplot as pit
import seaborn as sns
from sklearn.model_selection import train_test_split

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

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

In [7]:
```
dataset = pd.read_csv("C:\\Users\\hp\\Downloads\\creditcard.csv")
```

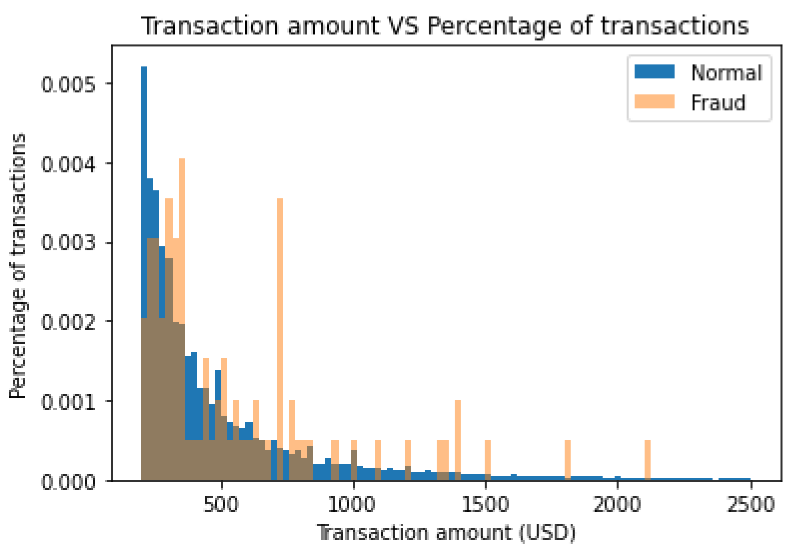
In [8]:
```
#check for any nullvalues
print("Any nulls in the dataset", dataset. isnull() . values . any( ) )
print ( '-------------')
print("No. of unique labels ",len(dataset [ 'Class' ]. unique()))
print("Label values", dataset. Class. unique())
#0 is for normal credit card transaction
#1 is for fraudulent credit cardtransaction
print ( '----------------- ')
print("Break down of the Normal and Fraud Transactions")
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 the Normal and Fraud Transactions
0    284315
1       492
Name: Class, dtype: int64
```

In [9]:
```
#Visualizing the imbalanced dataset
count_classes =pd. value_counts (dataset [ 'Class' ], sort =True)
count_classes . plot (kind = 'bar' , rot=0)

pit. xticks(range(len(dataset [ 'Class' ]. unique())), dataset. Class . unique())
pit . title("Frequency by observation number")
pit. xlabel("Class")
pit. ylabel("Number of Observations") ;
```



In [10]:
```
# Save the normal and fradulent transactions in separate dataframe
normal_dataset = dataset [dataset.Class ==0]
fraud_dataset = dataset [dataset.Class ==1]

#Visualize transactionamounts for normal and fraudulent transactions
bins = np.linspace(200, 2500, 100)

pit.hist(normal_dataset.Amount,  bins=bins,  alpha=1,  density=True,  label='Normal' )
pit.hist(fraud_dataset.Amount,  bins=bins,alpha=0.5,  density=True,  label='Fraud' )
pit.legend(loc='upper right' )
pit.title("Transaction amount VS Percentage of transactions")
pit.xlabel("Transaction amount (USD)")
pit.ylabel( "Percentage of transactions");
pit.show( )
```



In [11]:
```
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 [12]:
```
raw_data = dataset.values
# The last element contains if the transaction is normal which is represented by a 0 and
labels = raw_data[: , -1]

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

est_data, train_labels,test_labels = train_test_split(data, labels, test_si
```

In [13]:
```
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 [14]:
```
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 TrainData=", len(fraud_train_data))
print(" No. of records in Normal Traindata=", len(normal_train_data) )
print(" No. of records in Fraud TestData=", len(fraud_test_data) )
print(" No. of records in Normal Testdata=", len(normal_test_data) )
```

```
 No. of records in Fraud TrainData= 389
 No. of records in Normal Traindata= 227456
 No. of records in Fraud TestData= 103
 No. of records in Normal Testdata= 56859
```

In [15]:
```
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data . shape [1]
#num of columns, 30

encoding_dim = 14
hidden_dim_1 = int(encoding_dim / 2) #
hidden_dim_2=4
learning_rate = 1e-7
```

In [35]:
```
#input Layer
input_layer  =  tf.keras.layers.Input(shape=(input_dim,  ))
#Encoder
encoder  =  tf.keras.layers.Dense(encoding_dim,  activation="tanh",
encoder=tf.keras.layers.Dropout(0.2)(encoder)
encoder  =  tf.keras.layers.Dense(hidden_dim_1,  activation='relu')(encoder)
encoder  =  tf.keras.layers.Dense(hidden_dim_2,  activation=tf.nn.leaky_relu)(encoder)
# Decoder
decoder  =  tf.keras.layers.Dense(hidden_dim_1,  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_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_20 (InputLayer)       [(None, 30)]              0

 dense_101 (Dense)           (None, 14)                434

 dropout_35 (Dropout)        (None, 14)                0

 dense_102 (Dense)           (None, 7)                 105

 dense_103 (Dense)           (None, 4)                 32

 dense_104 (Dense)           (None, 7)                 35

 dropout_36 (Dropout)        (None, 7)                 0

 dense_105 (Dense)           (None, 14)                112

 dense_106 (Dense)           (None, 30)                450

=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
_________________________________________________________________
```

In [37]:

```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",
                               mode='min', monitor='val_loss', verbose=2, save_best_only
# define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True)
```

In [38]:

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

In [39]:

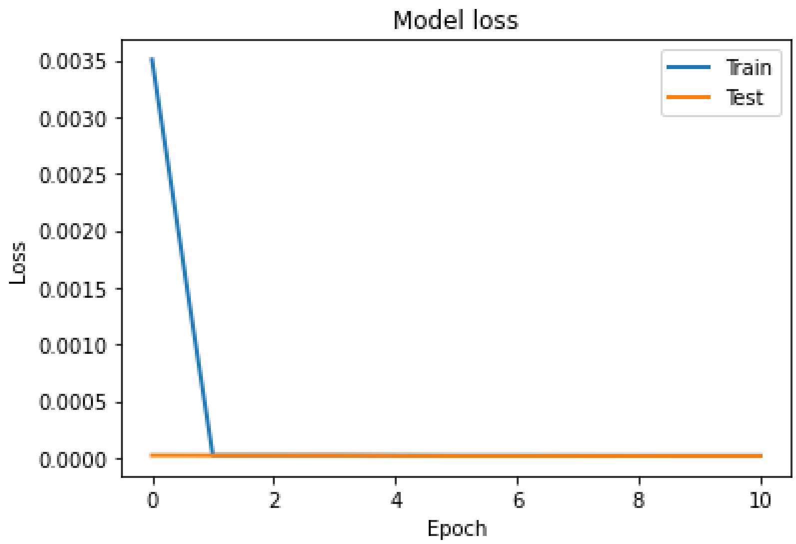
```
history = autoencoder.fit(normal_train_data, normal_train_data,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(test_data, test_data),
                    verbose=1,
                    callbacks=[cp,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

```
Epoch 1/50
3546/3554 [============================>.] - ETA: 0s - loss: 0.0035 - accuracy: 0.0428
Epoch 1: val_loss improved from inf to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 0.0035 - accuracy: 0.04
29 - val_loss: 2.1010e-05 - val_accuracy: 0.0010
Epoch 2/50
3498/3554 [============================>.] - ETA: 0s - loss: 1.9511e-05 - accuracy: 0.06
54
Epoch 2: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9497e-05 - accuracy:
0.0653 - val_loss: 2.0159e-05 - val_accuracy: 0.1279
Epoch 3/50
3531/3554 [============================>.] - ETA: 0s - loss: 1.9491e-05 - accuracy: 0.06
04
Epoch 3: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9496e-05 - accuracy:
0.0605 - val_loss: 2.0055e-05 - val_accuracy: 0.0363
Epoch 4/50
3515/3554 [============================>.] - ETA: 0s - loss: 1.9514e-05 - accuracy: 0.06
16
Epoch 4: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9509e-05 - accuracy:
0.0614 - val_loss: 2.0365e-05 - val_accuracy: 0.1282
Epoch 5/50
3510/3554 [============================>.] - ETA: 0s - loss: 1.8691e-05 - accuracy: 0.12
29
Epoch 5: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.8683e-05 - accuracy:
0.1240 - val_loss: 1.8075e-05 - val_accuracy: 0.2459
Epoch 6/50
3534/3554 [============================>.] - ETA: 0s - loss: 1.7261e-05 - accuracy: 0.24
89
Epoch 6: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7249e-05 - accuracy:
0.2491 - val_loss: 1.6787e-05 - val_accuracy: 0.3458
Epoch 7/50
3529/3554 [============================>.] - ETA: 0s - loss: 1.6975e-05 - accuracy: 0.25
92
Epoch 7: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.6975e-05 - accuracy:
0.2593 - val_loss: 1.6922e-05 - val_accuracy: 0.3552
Epoch 8/50
3527/3554 [============================>.] - ETA: 0s - loss: 1.6565e-05 - accuracy: 0.28
15
Epoch 8: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.6566e-05 - accuracy:
0.2813 - val_loss: 1.6613e-05 - val_accuracy: 0.3128
Epoch 9/50
3497/3554 [============================>.] - ETA: 0s - loss: 1.6354e-05 - accuracy: 0.29
62
Epoch 9: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.6329e-05 - accuracy:
0.2966 - val_loss: 1.6252e-05 - val_accuracy: 0.2860
Epoch 10/50
3514/3554 [============================>.] - ETA: 0s - loss: 1.6045e-05 - accuracy: 0.30
35
Epoch 10: val_loss improved from 0.00002 to 0.00002, saving model to autoencoder_fraud.h
5
3554/3554 [==============================] - 4s 1ms/step - loss: 1.6034e-05 - accuracy:
0.3036 - val_loss: 1.6071e-05 - val_accuracy: 0.2991
Epoch 11/50
3550/3554 [============================>.] - ETA: 0s - loss: 1.5913e-05 - accuracy: 0.30
65
Epoch 11: val_loss did not improve from 0.00002
```

```
Restoring model weights from the end of the best epoch: 1.
3554/3554 [==============================] - 4s 1ms/step - loss: 1.5911e-05 - accuracy:
0.3065 - val_loss: 1.6081e-05 - val_accuracy: 0.3095
Epoch 11: early stopping
```

In [41]:

```
pit.plot(history['loss'], linewidth=2, label='Train')
pit.plot(history['val_loss'],  linewidth=2,  label='Test')
pit.legend(loc='upper right')
pit.title('Model loss')
pit.ylabel('Loss')
pit.xlabel('Epoch')
#plt.ylim(ymin=0.70,ymax=1)
pit.show()
```



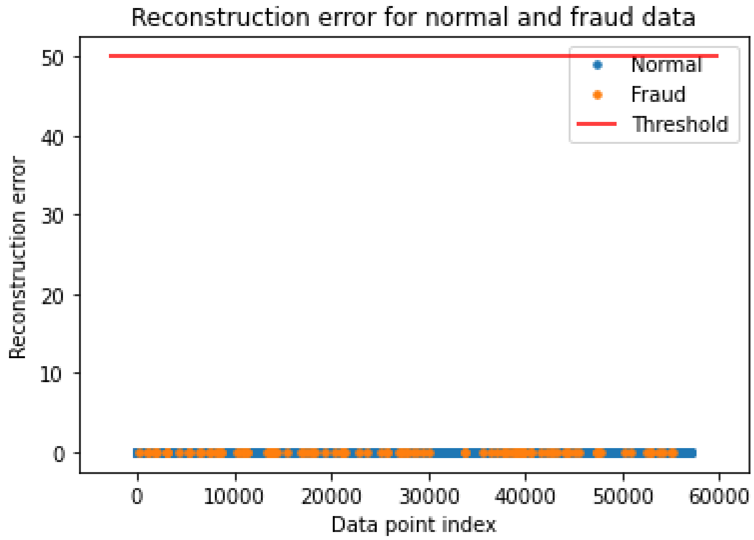
In [42]:

```
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  [==============================]  -  1s  516us/step
```

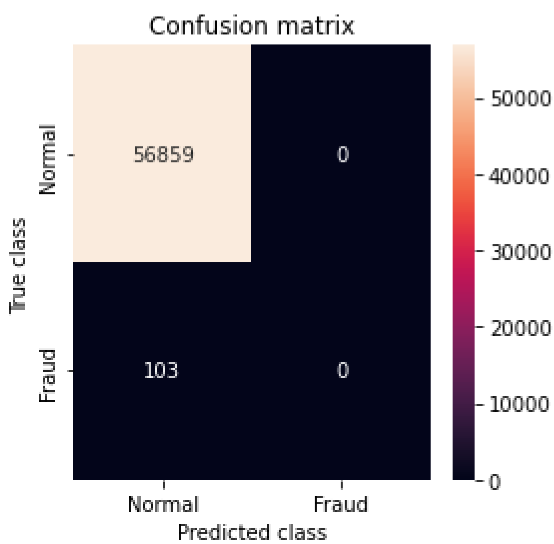
In [44]:

```
threshold_fixed = 50
groups  =  error_df.groupby('True_class')
fig,  ax = pit.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,  l
ax.legend()
pit.title("Reconstruction error for normal and fraud data")
pit.ylabel("Reconstruction  error")
pit.xlabel("Data point index")
pit.show();
```



In [45]:

```python
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)
pit.figure(figsize=(4, 4))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
pit.title("Confusion matrix")
pit.ylabel('True class')
pit.xlabel('Predicted class')
pit.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\hp\anaconda3\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))
```

In [ ]:

Loading [MathJax]/extensions/Safe.js