In [1]:
```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, accuracy_score, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

In [3]:
```python
dataset=pd.read_csv('creditcard.csv');
dataset.head();
# dataset.shape;
# dataset.describe();

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

```
Any nulls in the dataset  False
No. of unique labels  2
Label values  [0 1]
```

In [4]:
```python
dataset.head(5)
```

Out[4]:

	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	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

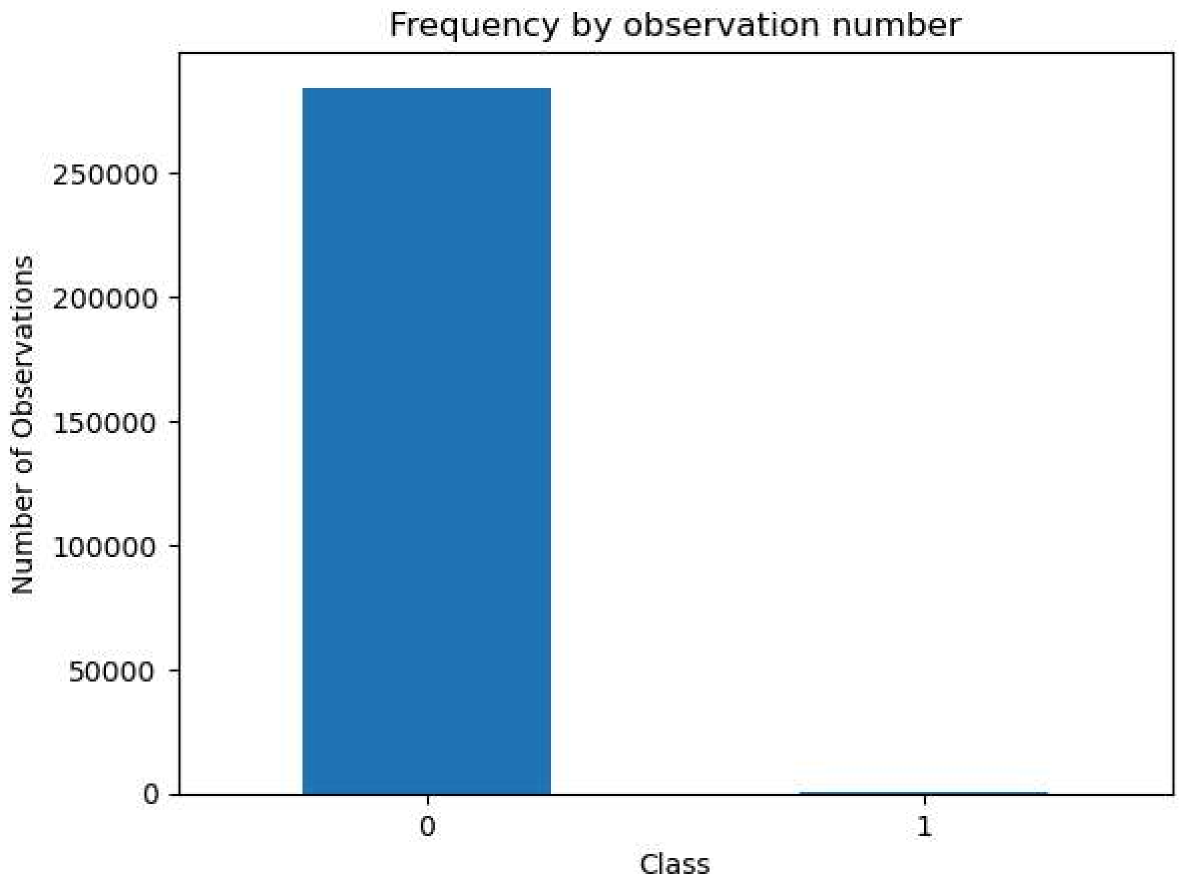
In [5]:
```python
print("Any nulls in the dataset ", dataset.isnull().values.any())
print("No. of unique labels ", len(dataset['Class'].unique()))
print("Label values ", dataset.Class.unique())
```

```
Any nulls in the dataset  False
No. of unique labels  2
Label values  [0 1]
```

In [6]:
```python
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")
plt.show()
```

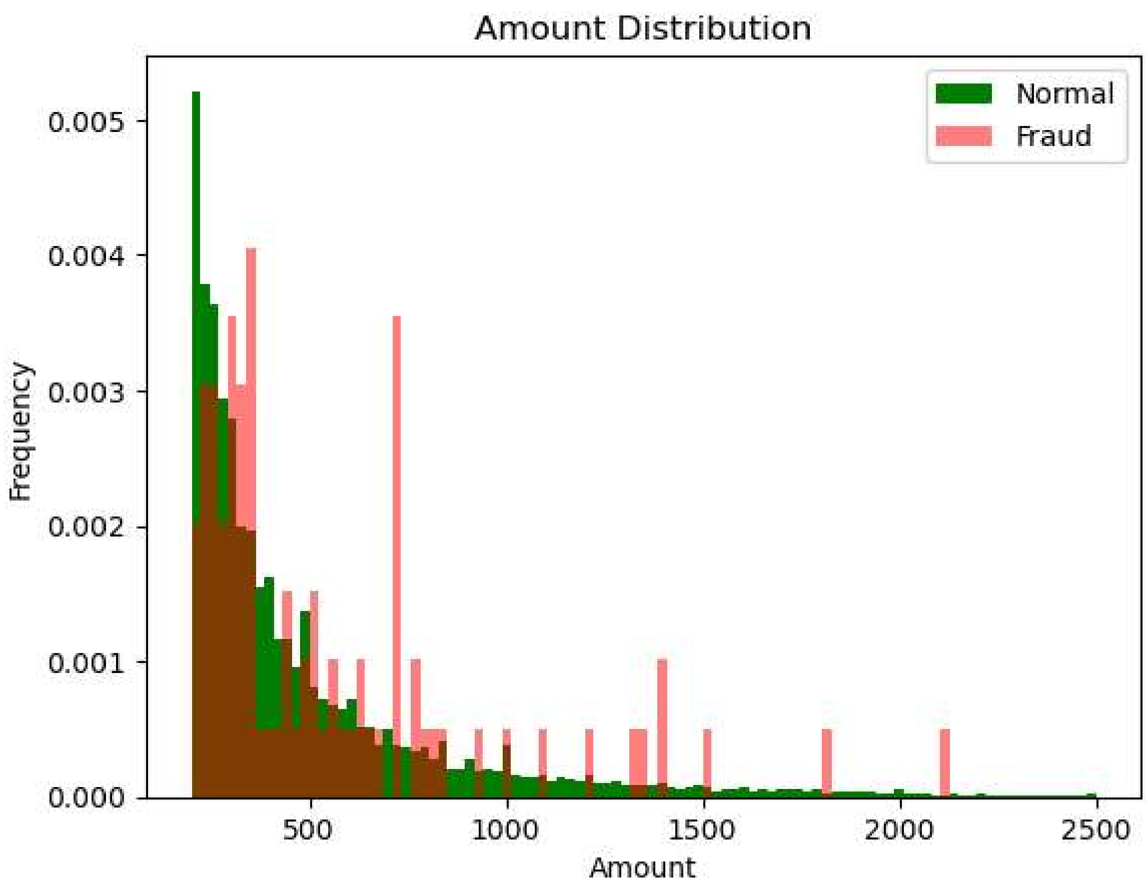
```
C:\Users\bonde\AppData\Local\Temp\ipykernel_5380\459402839.py:1: FutureWarning: pand
as.value_counts is deprecated and will be removed in a future version. Use pd.Series
(obj).value_counts() instead.
  count_classes = pd.value_counts(dataset['Class'], sort=True)
```



In [7]:
```
normal_dataset=dataset[dataset['Class']==0]
fraud_dataset=dataset[dataset['Class']==1]
print("Normal dataset shape ", normal_dataset.shape)
print("Fraud dataset shape ", fraud_dataset.shape)
```

```
Normal dataset shape  (284315, 31)
Fraud dataset shape  (492, 31)
```

In [9]:
```
bins=np.linspace(200,2500,100)
plt.hist(normal_dataset['Amount'], bins=bins, color='g', alpha=1,density=True, labe
plt.hist(fraud_dataset['Amount'], bins=bins, color='r', alpha=0.5,density=True, lab
plt.legend(loc='upper right')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.title('Amount Distribution')
plt.show()
```



In [10]:
```
sc=StandardScaler()
amount=dataset['Amount'].values
time=dataset['Time'].values
# dataset.drop(['Time','Amount'], axis=1, inplace=True)
dataset['Amount']=sc.fit_transform(amount.reshape(-1,1))
dataset['Time']=sc.fit_transform(time.reshape(-1,1))
```

In [11]:
```
raw_data=dataset.values
labels=raw_data[:,-1]
data=raw_data[:,0:-1]
```

In [12]:
```
train_data, test_data, train_labels, test_labels=train_test_split(data, labels, tes
print(train_data.shape, test_data.shape, train_labels.shape, test_labels.shape)

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)
print(train_labels.shape)
print(test_labels.shape)

normal_train_data=train_data[~train_labels]
normal_train_labels=train_labels[~train_labels]
```

```
fraud_train_data=train_data[train_labels]
fraud_train_labels=train_labels[train_labels]
```

```
(227845, 30) (56962, 30) (227845,) (56962,)
(227845,)
(56962,)
```

In [13]:
```
input_dim=normal_train_data.shape[1]
print(input_dim)

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

input_layer=tf.keras.layers.Input(shape=(input_dim,))
encoder=tf.keras.layers.Dense(units=hidden_dim_1, activation='tanh',
                              activity_regularizer=tf.keras.regularizers.l1(learning
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 = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
decoder = tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(input_dim, activation='relu')(decoder)
decoder = tf.keras.layers.Dense(input_dim, activation='tanh')(decoder)
autoencoder = tf.keras.models.Model(inputs=input_layer, outputs=decoder)
```

```
30
```

In [17]:
```
autoencoder.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 30)	0
dense (Dense)	(None, 7)	217
dropout (Dropout)	(None, 7)	0
dense_1 (Dense)	(None, 7)	56
dense_2 (Dense)	(None, 4)	32
dense_3 (Dense)	(None, 7)	35
dropout_1 (Dropout)	(None, 7)	0
dense_4 (Dense)	(None, 30)	240
dense_5 (Dense)	(None, 30)	930

Total params: 4,532 (17.71 KB)

Trainable params: 1,510 (5.90 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 3,022 (11.81 KB)

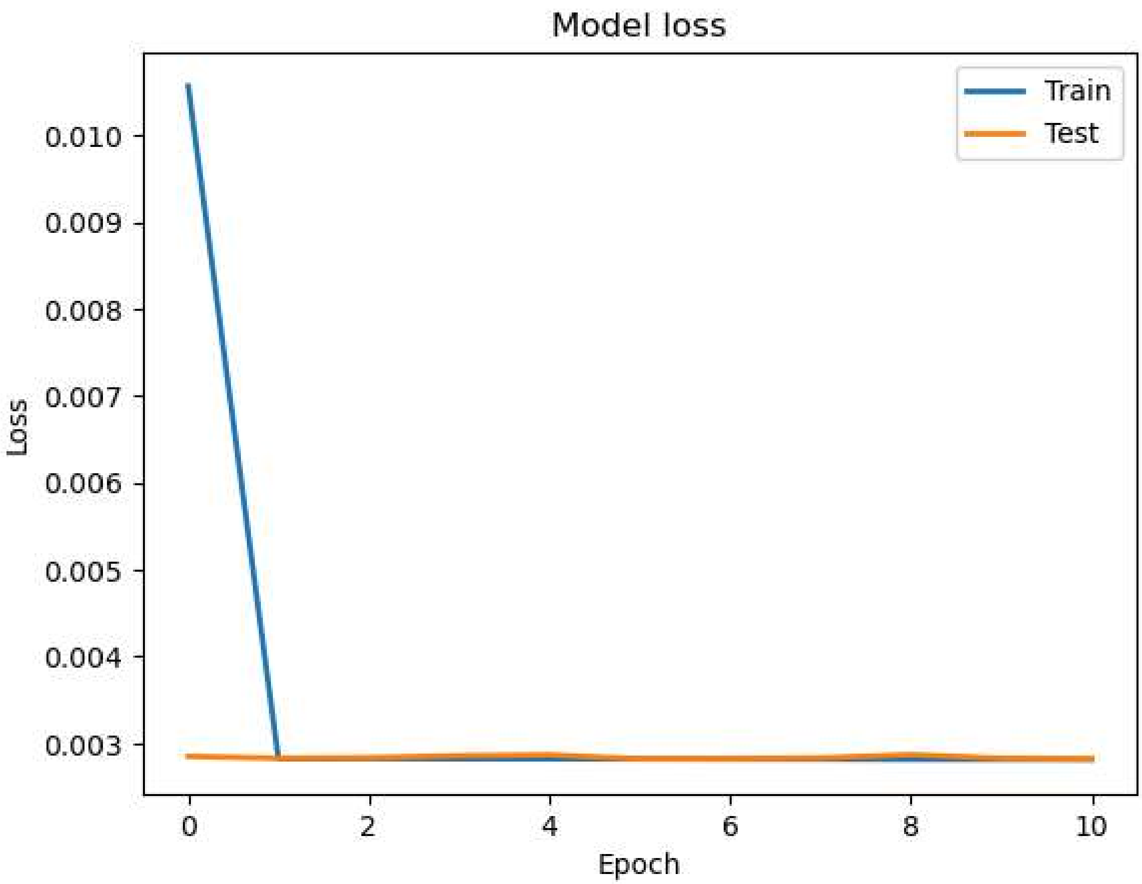
In [14]:
```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.keras", mode='m
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0.0001,
# EarlyStopping: This callback stops training early if the validation loss doesn't
autoencoder.compile(metrics=['accuracy'], loss='mae', optimizer='adam')
```

In [15]:
```
history=autoencoder.fit(normal_train_data, normal_train_data, epochs=50, batch_size
```

```
Epoch 1/50
3513/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 957us/step - accuracy: 0.0760 - loss: 0.0409
Epoch 1: val_loss improved from inf to 0.00285, saving model to autoencoder_fraud.ke
ras
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step - accuracy: 0.0762 - loss: 0.0405 - val_a
ccuracy: 0.2168 - val_loss: 0.0028
Epoch 2/50
3507/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 929us/step - accuracy: 0.0999 - loss: 0.0028
Epoch 2: val_loss improved from 0.00285 to 0.00283, saving model to autoencoder_frau
d.keras
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - accuracy: 0.0998 - loss: 0.0028 - val_a
ccuracy: 0.0341 - val_loss: 0.0028
Epoch 3/50
3519/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 915us/step - accuracy: 0.0942 - loss: 0.0028
Epoch 3: val_loss did not improve from 0.00283
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - accuracy: 0.0942 - loss: 0.0028 - val_a
ccuracy: 0.0596 - val_loss: 0.0028
Epoch 4/50
3549/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 982us/step - accuracy: 0.0928 - loss: 0.0028
Epoch 4: val_loss did not improve from 0.00283
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - accuracy: 0.0928 - loss: 0.0028 - val_a
ccuracy: 0.1279 - val_loss: 0.0029
Epoch 5/50
3533/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 928us/step - accuracy: 0.0913 - loss: 0.0028
Epoch 5: val_loss did not improve from 0.00283
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - accuracy: 0.0913 - loss: 0.0028 - val_a
ccuracy: 0.0363 - val_loss: 0.0029
Epoch 6/50
3536/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 939us/step - accuracy: 0.0944 - loss: 0.0028
Epoch 6: val_loss improved from 0.00283 to 0.00282, saving model to autoencoder_frau
d.keras
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - accuracy: 0.0944 - loss: 0.0028 - val_a
ccuracy: 0.0351 - val_loss: 0.0028
Epoch 7/50
3552/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0923 - loss: 0.0028
Epoch 7: val_loss did not improve from 0.00282
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.0923 - loss: 0.0028 - val_a
ccuracy: 0.0351 - val_loss: 0.0028
Epoch 8/50
3553/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0903 - loss: 0.0028
Epoch 8: val_loss did not improve from 0.00282
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - accuracy: 0.0903 - loss: 0.0028 - val_a
ccuracy: 0.2168 - val_loss: 0.0028
Epoch 9/50
3523/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0912 - loss: 0.0028
Epoch 9: val_loss did not improve from 0.00282
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.0912 - loss: 0.0028 - val_a
ccuracy: 0.2168 - val_loss: 0.0029
Epoch 10/50
3530/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0899 - loss: 0.0028
Epoch 10: val_loss did not improve from 0.00282
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.0899 - loss: 0.0028 - val_a
ccuracy: 0.0269 - val_loss: 0.0028
Epoch 11/50
3530/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0837 - loss: 0.0028
Epoch 11: val_loss improved from 0.00282 to 0.00282, saving model to autoencoder_fra
```

```
ud.keras
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.0837 - loss: 0.0028 - val_a
ccuracy: 0.0420 - val_loss: 0.0028
Epoch 11: early stopping
Restoring model weights from the end of the best epoch: 1.
```

In [16]:
```python
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.show()
```

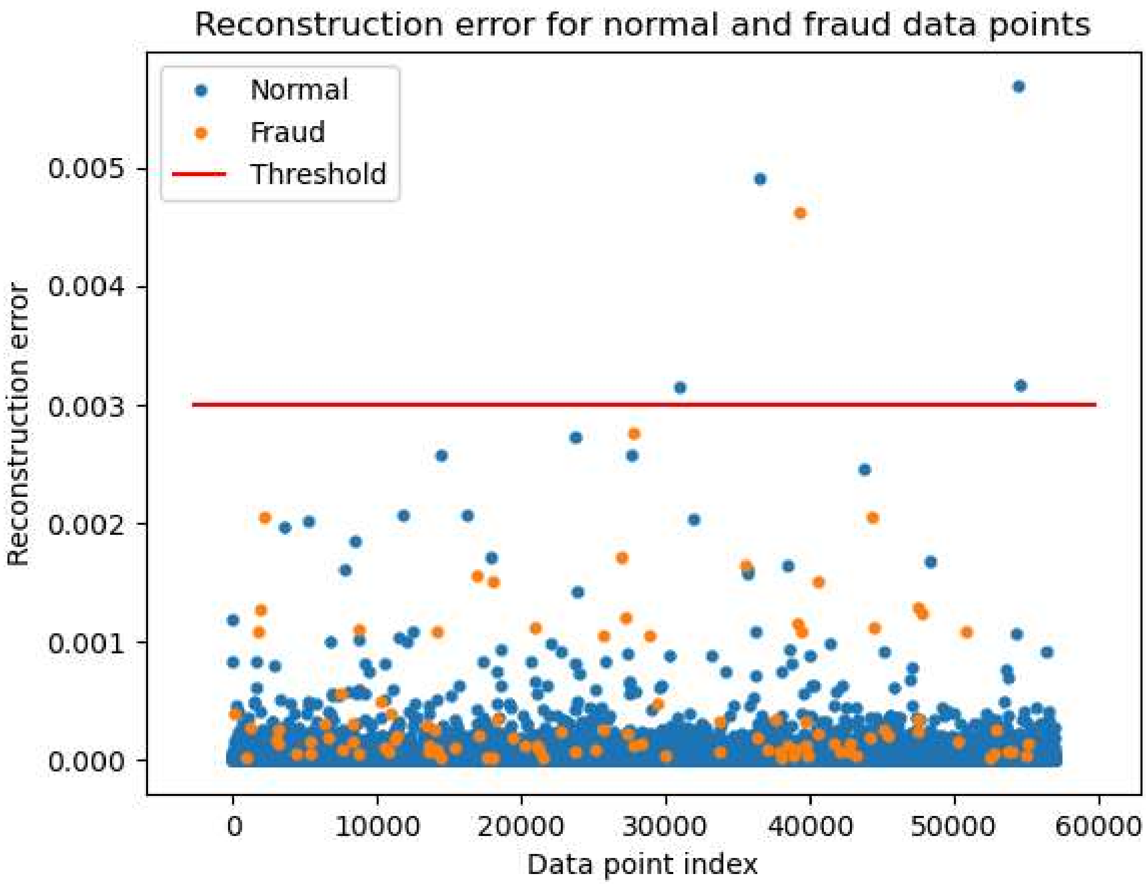


In [18]:
```python
test_x_prediction=autoencoder.predict(test_data)
test_loss=tf.keras.losses.mse(test_data, test_x_prediction)
test_loss=tf.reshape(test_loss, [-1])
```

```
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 1s 650us/step
```

In [20]:
```python
# print(test_loss.shape,test_loss.numpy())
error_df = pd.DataFrame({'Reconstruction_error': test_loss, 'True_class': test_labe
# true_class_df = error_df[error_df['True_class'] == True]
threshold_fixed=0.003
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=1
ax.legend()
plt.title("Reconstruction error for normal and fraud data points")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```



In [21]:
```
pred_y=[1 if e>threshold_fixed else 0 for e in error_df.Reconstruction_error.values
error_df['pred']=pred_y
true_class_df = error_df[error_df['pred'] == 1]
true_class_df
```

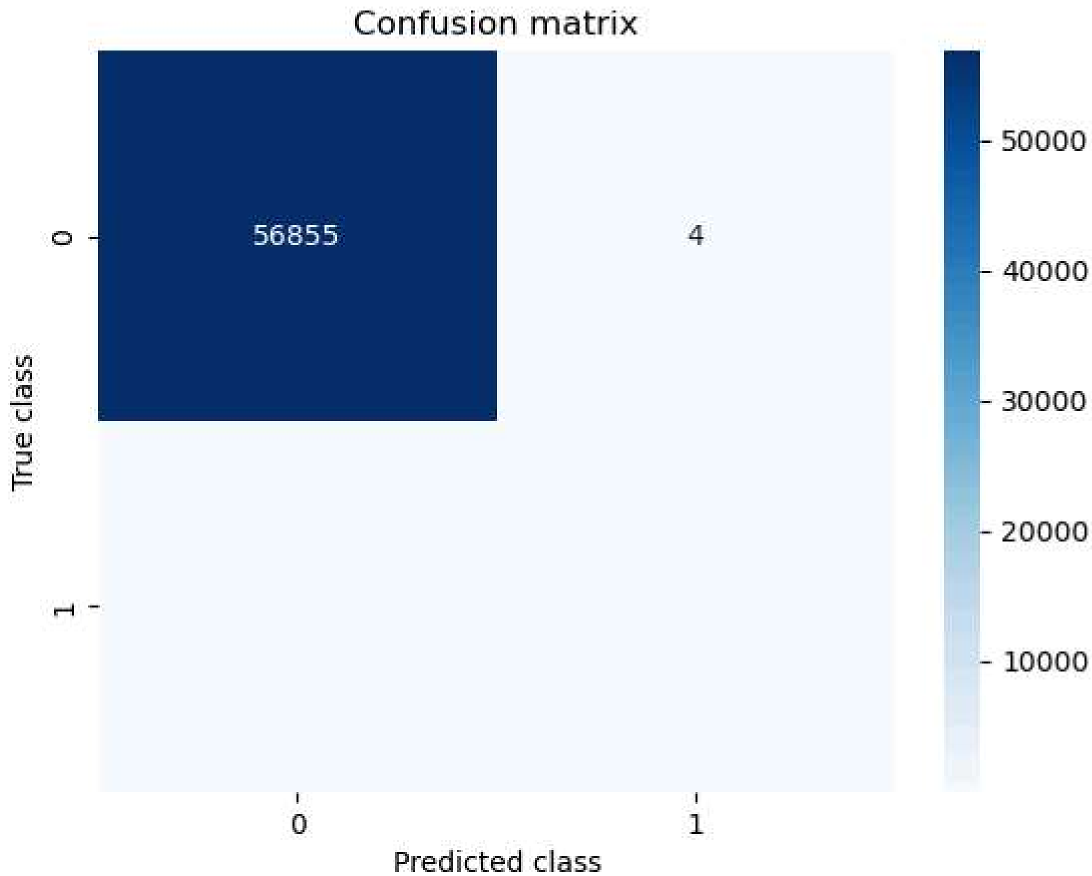
Out[21]:

	Reconstruction_error	True_class	pred
31012	0.003149	False	1
36510	0.004909	False	1
39248	0.004619	True	1
54463	0.005690	False	1
54581	0.003165	False	1

In [22]:
```
conf_matrix=confusion_matrix(error_df.True_class, pred_y)
print(conf_matrix)
sns.heatmap(conf_matrix,cmap='Blues', annot=True, fmt='d')
plt.title('Confusion matrix')
plt.ylabel('True class')
```

```
plt.xlabel('Predicted class')
plt.show()
```

```
[[56855     4]
 [  102     1]]
```



In [23]:
```
from sklearn.metrics import recall_score, precision_score

print("Accuracy:", accuracy_score(error_df.True_class, pred_y))
print(" Recall: ", recall_score(error_df['True_class'], error_df['pred']))
print(" Precision: ", precision_score(error_df['True_class'], error_df['pred']))
```

```
Accuracy: 0.998139110284049
 Recall:  0.009708737864077669
 Precision:  0.2
```

In [ ]: