In [3]:
```
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, classificatio
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

In [4]:
```
dataset=pd.read_csv('creditcard.csv');
dataset.head();
# dataset.shape;
# dataset.describe();
```

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

Out[5]:

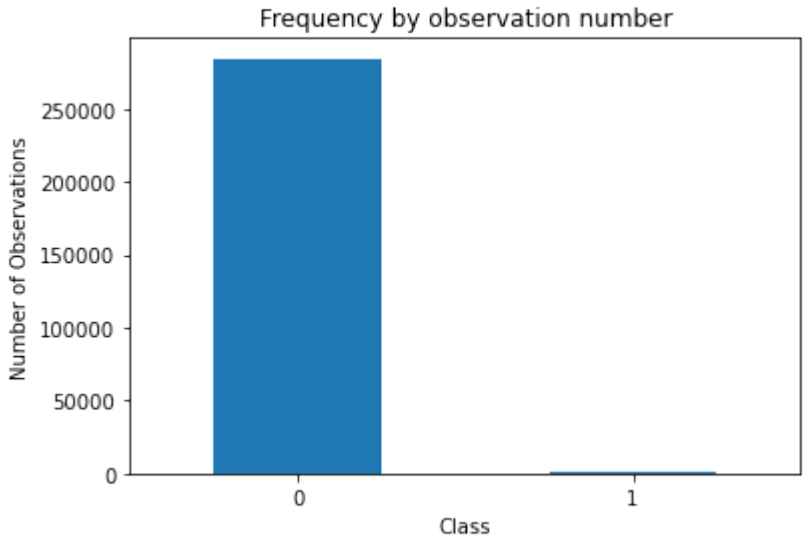
	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [6]:
```
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]
```

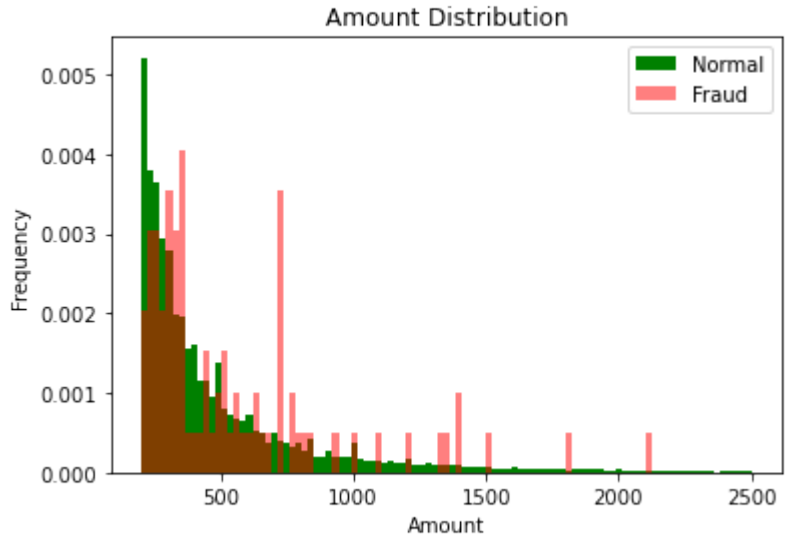
In [7]:
```
count_classes = pd.value_counts(dataset['Class'], sort=True)
count_classes.plot(kind='bar', rot=0)
plt.xticks(range(len(dataset['Class'].unique())), dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations")
plt.show()
```



In [8]:
```
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=Tr
plt.hist(fraud_dataset['Amount'], bins=bins, color='r', alpha=0.5,density=T
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, lab
print(train_data.shape, test_data.shape, train_labels.shape, test_labels.sh
```

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

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

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

In [15]:
```
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]
```

In [16]:
```
input_dim=normal_train_data.shape[1]
print(input_dim)
```

```
30
```

In [17]:
```
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(
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)(


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)
```

In [18]:
```
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: 1,510 (5.90 KB)

Trainable params: 1,510 (5.90 KB)

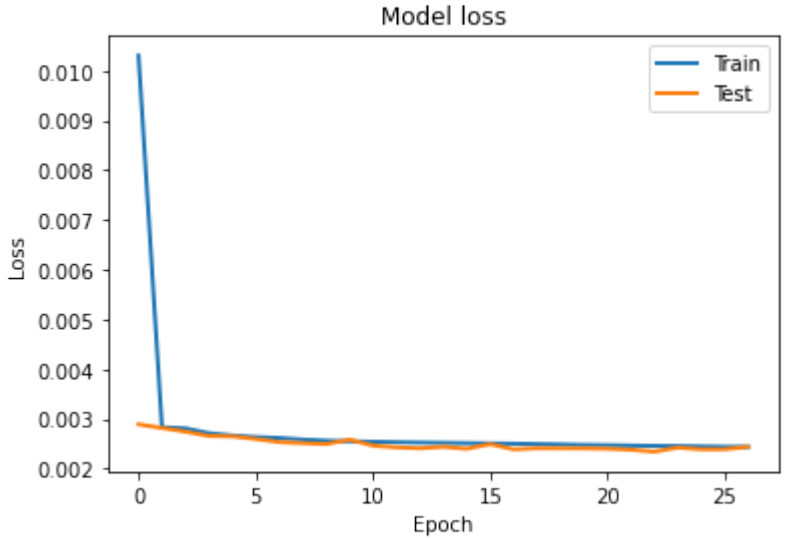
Non-trainable params: 0 (0.00 B)

In [19]:
```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.keras",
# filepath: Specifies where to save the model ("autoencoder_fraud.h5").
# monitor: Monitors the validation loss ('val_loss') during training.
# mode='min': The callback saves the model only when the monitored metric (
# save_best_only=True: It saves the best version of the model.
# verbose=2: Provides verbose logging when the model is saved

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta
# EarlyStopping: This callback stops training early if the validation loss (

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

In [20]:
```
history=autoencoder.fit(normal_train_data, normal_train_data, epochs=50, ba
```

```
Epoch 1/50
3535/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0675 - loss:
0.0387
Epoch 1: val_loss improved from inf to 0.00289, saving model to autoenc
oder_fraud.keras
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 14s 3ms/step - accuracy: 0.0676 - loss:
0.0385 - val_accuracy: 0.1279 - val_loss: 0.0029
Epoch 2/50
3552/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0901 - loss:
0.0028
Epoch 2: val_loss improved from 0.00289 to 0.00282, saving model to aut
oencoder_fraud.keras
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.0901 - loss:
0.0028 - val_accuracy: 0.2170 - val_loss: 0.0028
Epoch 3/50
3550/3554 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0983 - loss:
0.0028
Epoch 3: val_loss improved from 0.00282 to 0.00274, saving model to aut
oencoder_fraud.keras
```

In [21]:
```
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 [22]:
```
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])

# print(test_loss.shape,test_loss.numpy())
error_df = pd.DataFrame({'Reconstruction_error': test_loss, 'True_class': t
# true_class_df = error_df[error_df['True_class'] == True]
# true_class_df
```

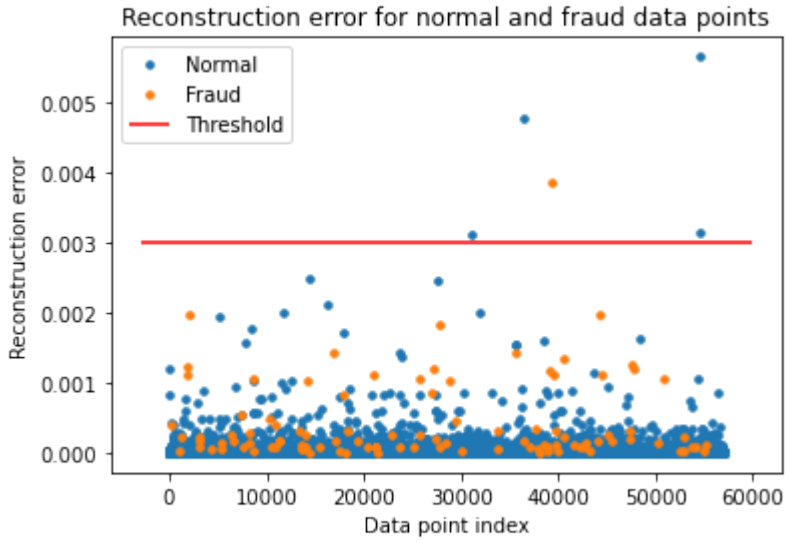
1781/1781 ━━━━━━━━━━━━━━━━━━━━ **2s** 1ms/step

Out[22]:

	Reconstruction_error	True_class
131	0.000395	True
987	0.000018	True
1279	0.000232	True
1787	0.001110	True
1887	0.001223	True
...	...	...
52971	0.000227	True
53694	0.000073	True
54076	0.000075	True
54934	0.000024	True
55126	0.000100	True

103 rows × 2 columns

In [23]:
```
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, li
            label="Fraud" if name==1 else "Normal")
ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[1], colors="r",
ax.legend()
plt.title("Reconstruction error for normal and fraud data points")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```



In [24]:
```
pred_y=[1 if e>threshold_fixed else 0 for e in error_df.Reconstruction_erro
error_df['pred']=pred_y
```

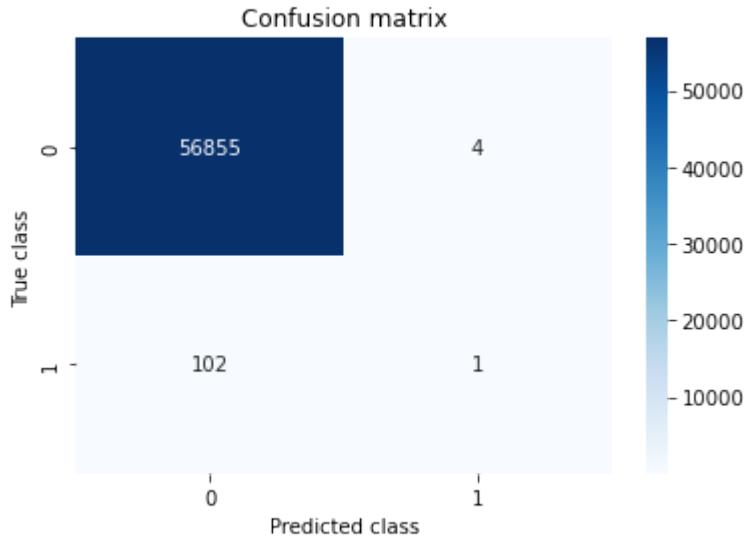
In [25]:
```
true_class_df = error_df[error_df['pred'] == 1]
true_class_df
```

Out[25]:

	Reconstruction_error	True_class	pred
31012	0.003110	False	1
36510	0.004770	False	1
39248	0.003848	True	1
54463	0.005640	False	1
54581	0.003144	False	1

In [26]:
```
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 [27]:
```
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['pre
```

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

In [ ]: