In [2]:

```python
import pandas as pd
import numpy as np
from scipy import stats
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
from sklearn.model_selection import train_test_split
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precisi

sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
```

In [4]:

```python
df = pd.read_csv("creditcard.csv")
```

In [5]:

```python
df.shape
```

Out[5]:

```
(284807, 31)
```

In [6]:

```python
df.isnull().values.any()
```

Out[6]:

```
False
```

In [7]:

```python
print(list(df.columns))
df.describe()
```

```
['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V1
2', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23',
'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class']
```

Out[7]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

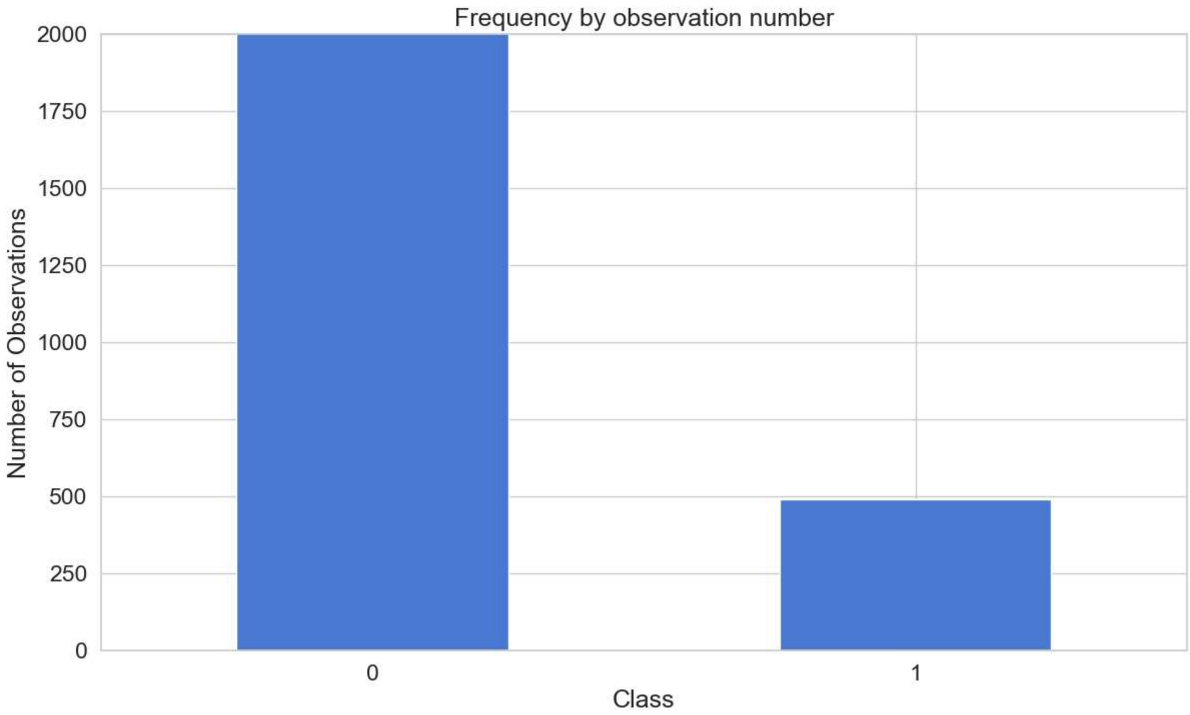
8 rows × 31 columns

In [8]:

```python
#Visualizing the imbalanced dataset
count_classes = pd.value_counts(df['Class'], sort = True)
```

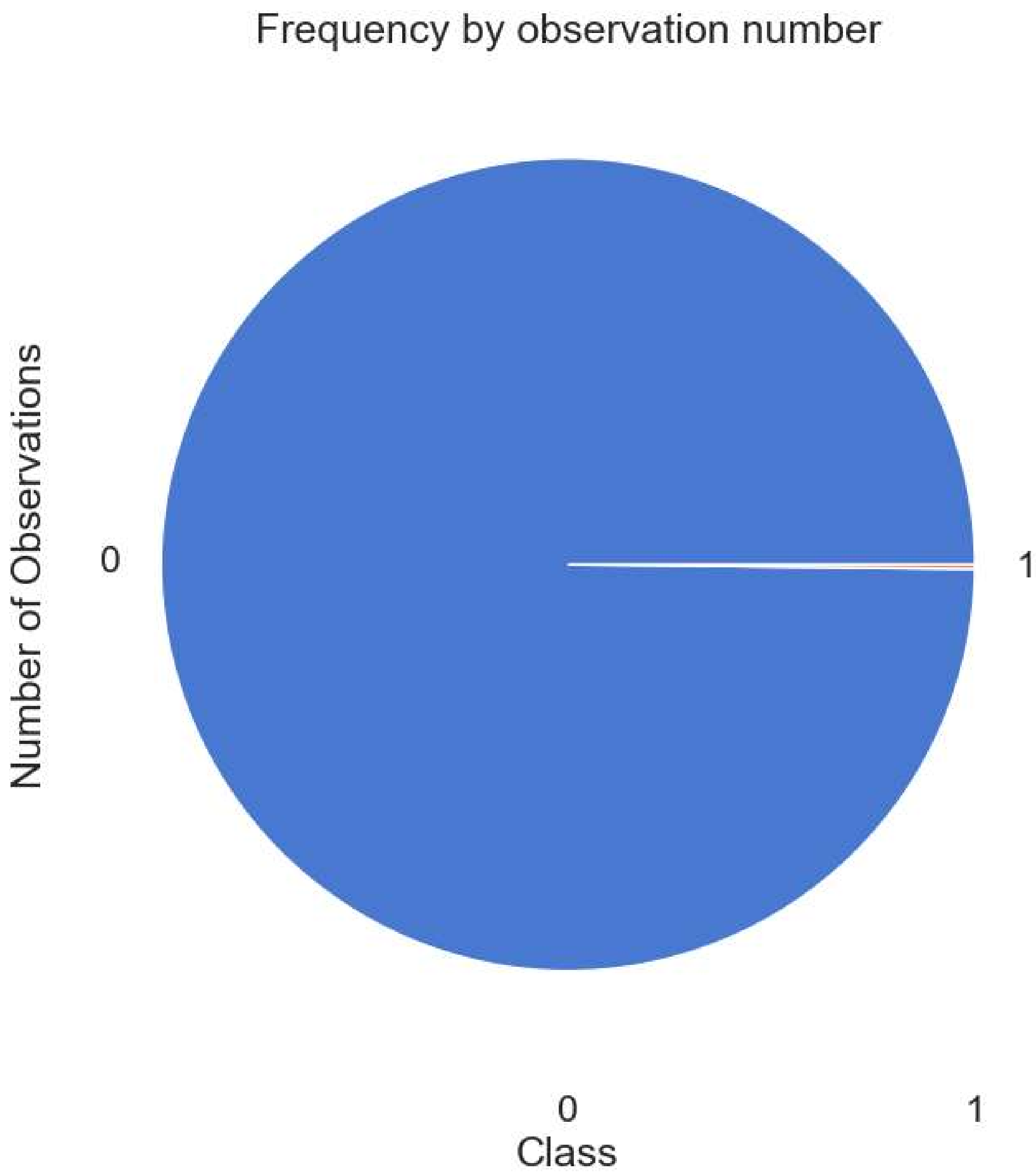
```python
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(len(df['Class'].unique())), df.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");
plt.ylim(0,2000)
```

Out[8]: (0.0, 2000.0)



In [11]:

```python
#Visualizing the imbalanced dataset
count_classes = pd.value_counts(df['Class'], sort = True)
count_classes.plot(kind = 'pie', rot=0)
plt.xticks(range(len(df['Class'].unique())), df.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");
```




```
In [12]: sc=StandardScaler()
         df['Time'] = sc.fit_transform(df['Time'].values.reshape(-1, 1))
         df['Amount'] = sc.fit_transform(df['Amount'].values.reshape(-1, 1))
```

```
In [13]: raw_data = df.values
         # The last element contains if the transaction is normal which is represented by a
         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, t
```

```
In [14]: #'''Normalize the data to have a value between 0 and 1'''
         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 [15]: train_labels = train_labels.astype(bool)
         test_labels = test_labels.astype(bool)
         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 [16]:
```
raw_data = df.values
# The last element contains if the transaction is normal which is represented by a
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, t
```

In [17]:
```
# Normalize the data to have a value between 0 and 1
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 [18]:
```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
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))
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
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
 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 [19]:
```
nb_epoch = 100
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 [20]:
```
#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_rate))(input_layer)
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)
```

In [21]:
```
# 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"
_________________________________________________________________
 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: 1168 (4.56 KB)
Trainable params: 1168 (4.56 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [22]:
```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mode='min',
```

In [23]:
```
# 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)
```

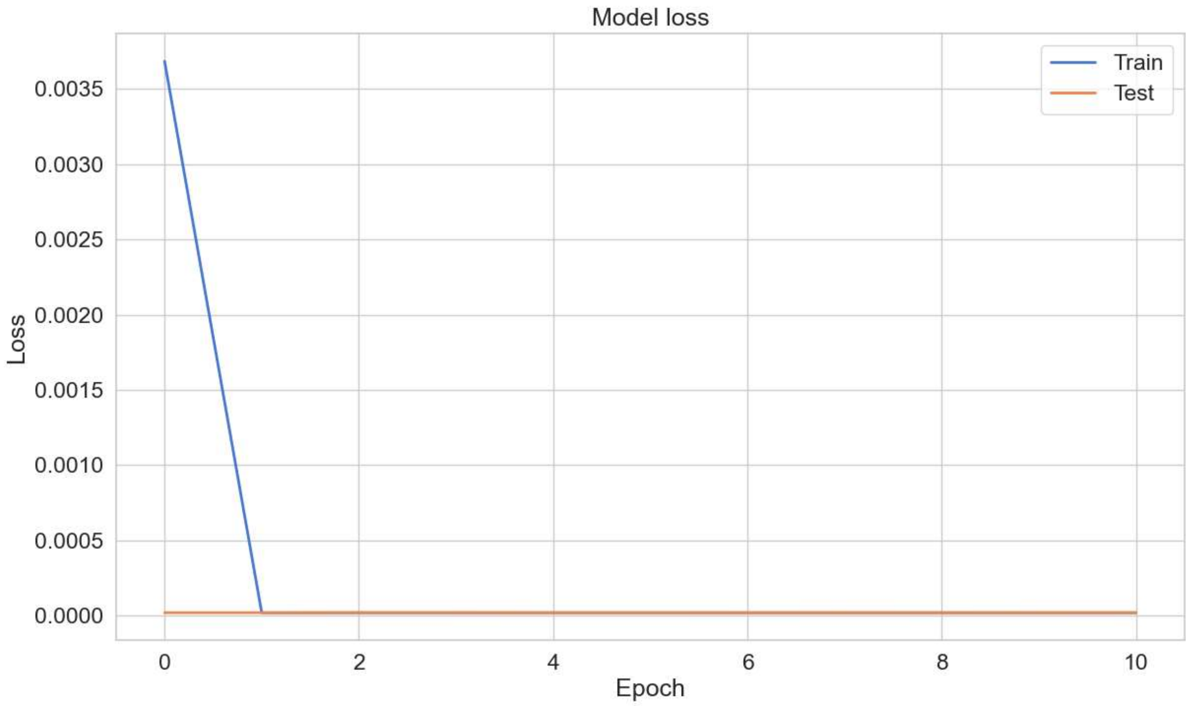
In [24]:
```
#Compile the Autoencoder

autoencoder.compile(metrics=['accuracy'],
```

```python
loss='mean_squared_error',

optimizer='adam')

#Train the Autoencoder
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 1/100
3540/3554 [============================>.] - ETA: 0s - loss: 0.0037 - accuracy: 0.
0538
Epoch 1: val_loss improved from inf to 0.00002, saving model to autoencoder_fraud.
h5
3554/3554 [==============================] - 17s 4ms/step - loss: 0.0037 - accurac
y: 0.0539 - val_loss: 1.9879e-05 - val_accuracy: 0.0189
Epoch 2/100
  16/3554 [..............................] - ETA: 12s - loss: 2.3124e-05 - accurac
y: 0.0205
```

```
C:\Users\ANANYAPRANAV\AppData\Roaming\Python\Python311\site-packages\keras\src\eng
ine\training.py:3000: UserWarning: You are saving your model as an HDF5 file via `
model.save()`. This file format is considered legacy. We recommend using instead t
he native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

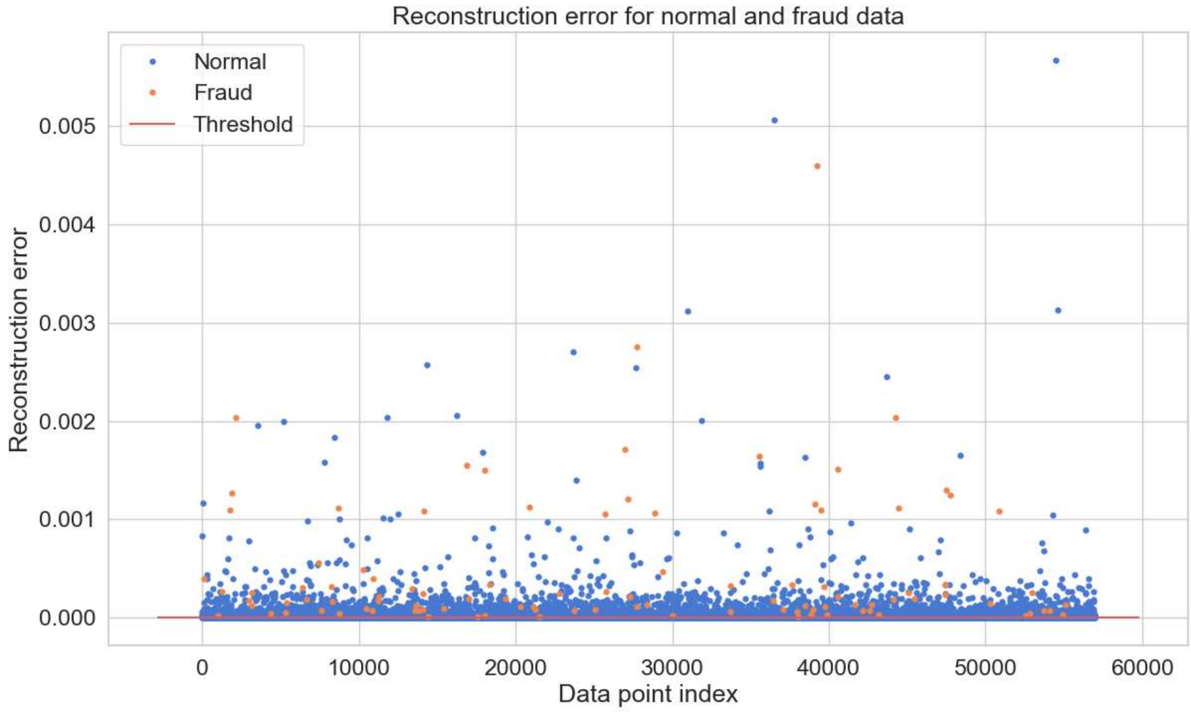
```
plt.plot(history['val_loss'], linewidth=2, label='Test')
plt.legend(loc='upper right')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
#plt.ylim(ymin=0.70,ymax=1)
plt.show()
```



In [26]:
```
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 [==============================] - 4s 2ms/step
```

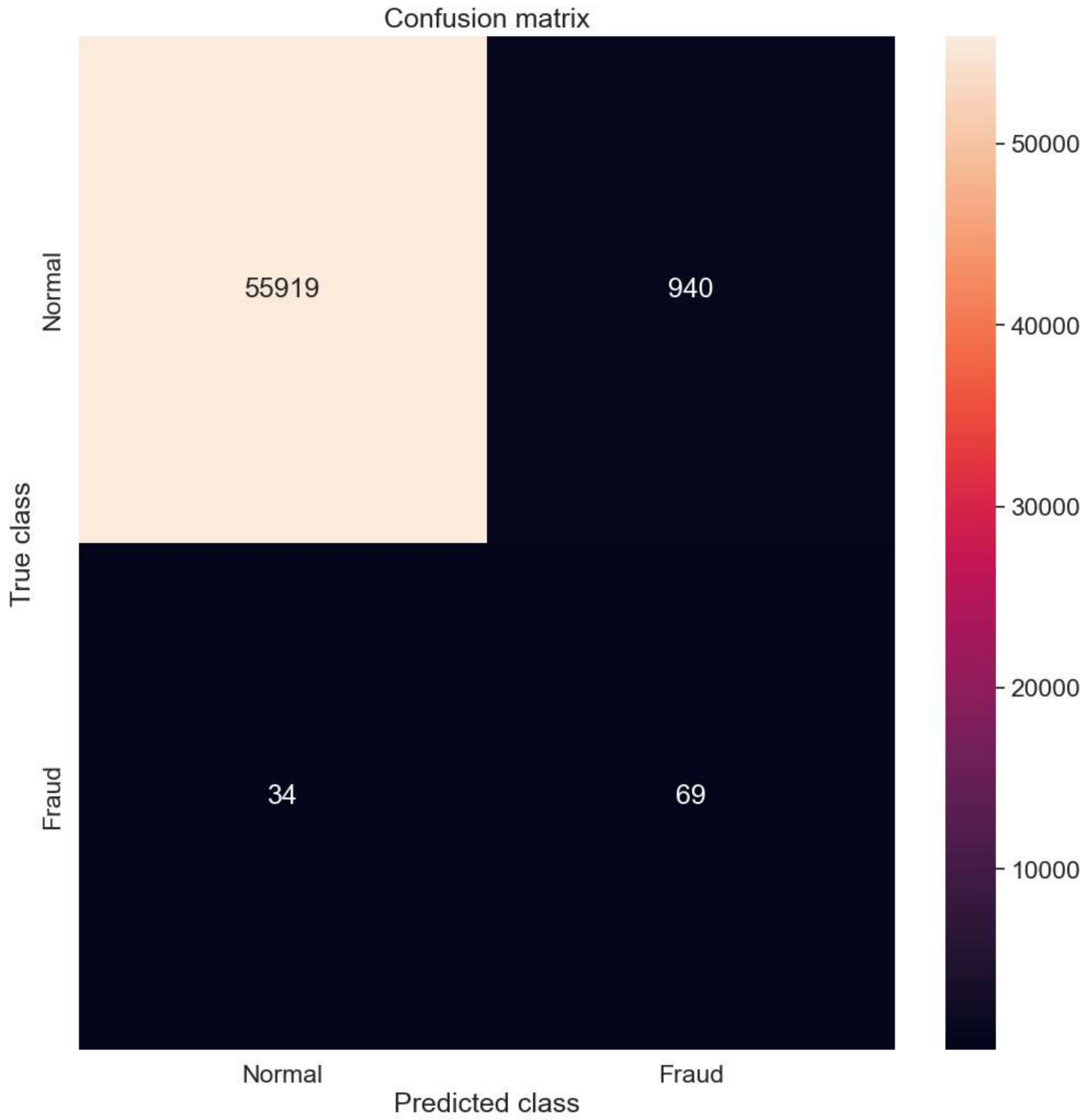
In [27]:
```
threshold_fixed = 0
groups = error_df.groupby('True_class')
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.index, group.Reconstruction_error, marker='o', ms=3.5, linestyle=
ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", zorder=1
ax.legend()
plt.title("Reconstruction error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```



In [37]:
```
df.isnull().values.any()
```

Out[37]:
```
False
```

In [38]:
```
threshold_fixed = 0.0001

y_pred = [1 if e > threshold_fixed else 0 for e in error_df.Reconstruction_error.va
conf_matrix = confusion_matrix(error_df.True_class, y_pred)

plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="c
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```


Confusion matrix
55919
940
34
69
Normal
Fraud
True class
Normal
Fraud
Predicted class
50000
40000
30000
20000
10000