In [ ]:

```
pip install tensorflow --user
!pip install keras
!pip install daytime
!pip install torch
```

In [1]:

```
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
```

In [3]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_s
RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

In [26]:

```
dataset = pd.read_csv("E:\Teachning material\Deep learning BE IT 2019 course\creditcard
#dataset.head
print(list(dataset.columns))
dataset.describe()
```

```
['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11'
'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22',
'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class']
```

Out[26]:

	Time	V1	V2	V3	V4	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+0
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+0
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+0
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-0
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-0
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-0
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+0

8 rows × 31 columns

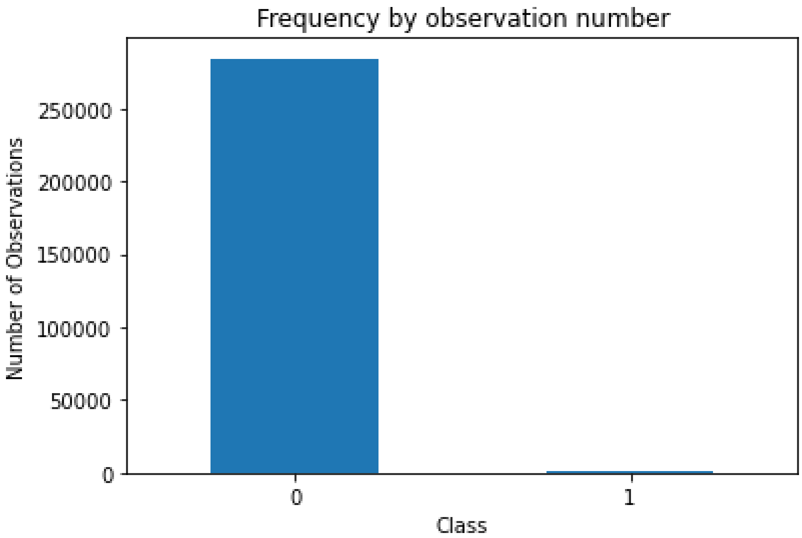
In [6]:

```
#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 card transaction
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
```

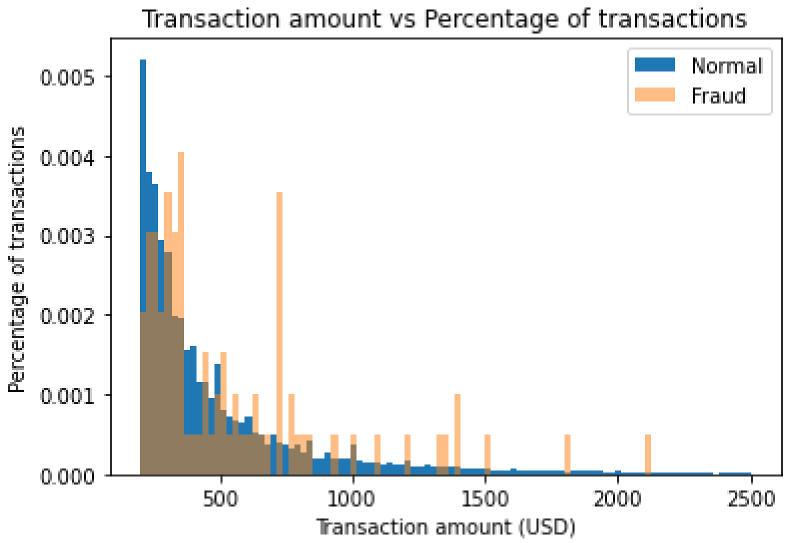
In [7]:

```
#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");
```



In [8]:

```
# 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)
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("Transaction amount vs Percentage of transactions")
plt.xlabel("Transaction amount (USD)")
plt.ylabel("Percentage of transactions");
plt.show()
```



In [ ]:

```
'''Time and Amount are the columns that are not scaled, so applying StandardScaler to c
Normalizing the values between 0 and 1 did not work great for the dataset.'''
```

In [9]:

```
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 [10]:

```
'''The last column in the dataset is our target variable.'''

raw_data = dataset.values
# The last element contains if the transaction is normal which is represented by a 0 ar
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 [12]:

```
'''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 [13]:

```
'''Use only normal transactions to train the Autoencoder.

Normal data has a value of 0 in the target variable. Using the target variable to creat

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 [25]:

```
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 [27]:

```
#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))(i
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"
_________________________________________________________________
 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 [29]:

```
"""Define the callbacks for checkpoints and early stopping"""

cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",
                               mode='min', monitor='val_loss', verbose=2, save_best_on]
# 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 [30]:

```
#Compile the Autoencoder

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

In [38]:

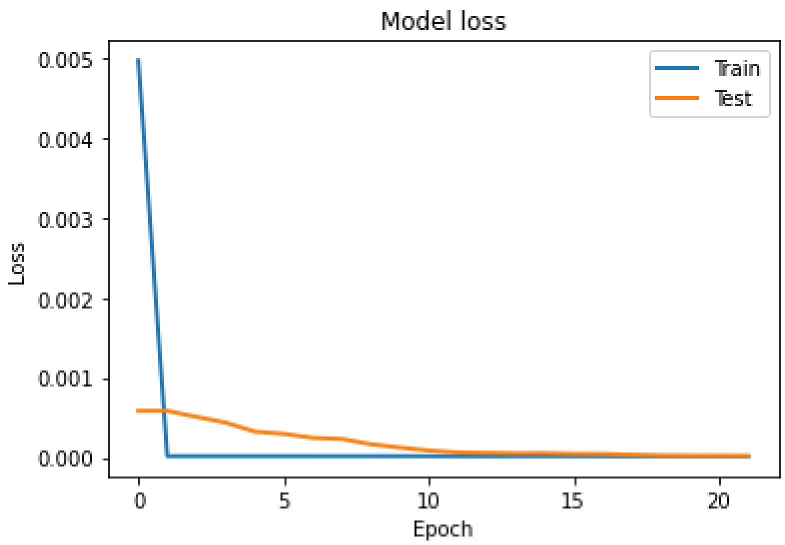
```
#Train the Autoencoder

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 1/50
3551/3554 [============================>.] - ETA: 0s - loss: 1.8010e-05 -
accuracy: 0.1771
Epoch 1: val_loss did not improve from 0.00002
3554/3554 [==============================] - 5s 1ms/step - loss: 1.8009e-0
5 - accuracy: 0.1770 - val_loss: 5.1378e-05 - val_accuracy: 0.0251
Epoch 2/50
3526/3554 [============================>.] - ETA: 0s - loss: 1.7782e-05 -
accuracy: 0.1860
Epoch 2: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7783e-0
5 - accuracy: 0.1863 - val_loss: 3.4659e-05 - val_accuracy: 0.0251
Epoch 3/50
3532/3554 [============================>.] - ETA: 0s - loss: 1.7454e-05 -
accuracy: 0.1958
Epoch 3: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7450e-0
5 - accuracy: 0.1958 - val_loss: 3.4017e-05 - val_accuracy: 0.0251
Epoch 4/50
3547/3554 [============================>.] - ETA: 0s - loss: 1.7239e-05 -
accuracy: 0.2095
Epoch 4: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7239e-0
5 - accuracy: 0.2095 - val_loss: 3.0229e-05 - val_accuracy: 0.0251
Epoch 5/50
3527/3554 [============================>.] - ETA: 0s - loss: 1.6995e-05 -
accuracy: 0.2248
Epoch 5: val_loss did not improve from 0.00002
3554/3554 [==============================] - 5s 1ms/step - loss: 1.6998e-0
5 - accuracy: 0.2248 - val_loss: 2.9758e-05 - val_accuracy: 0.0251
Epoch 6/50
3525/3554 [============================>.] - ETA: 0s - loss: 1.6877e-05 -
accuracy: 0.2382
Epoch 6: val_loss did not improve from 0.00002
3554/3554 [==============================] - 5s 1ms/step - loss: 1.6878e-0
5 - accuracy: 0.2382 - val_loss: 2.8776e-05 - val_accuracy: 0.0251
Epoch 7/50
3516/3554 [============================>.] - ETA: 0s - loss: 1.6769e-05 -
accuracy: 0.2545
Epoch 7: val_loss did not improve from 0.00002
3554/3554 [==============================] - 5s 1ms/step - loss: 1.6757e-0
5 - accuracy: 0.2545 - val_loss: 2.5803e-05 - val_accuracy: 0.0230
Epoch 8/50
3554/3554 [==============================] - ETA: 0s - loss: 1.6662e-05 -
accuracy: 0.2617
Epoch 8: val_loss did not improve from 0.00002
```

```
3554/3554 [==============================] - 4s 1ms/step - loss: 1.6662e-0
5 - accuracy: 0.2617 - val_loss: 2.7398e-05 - val_accuracy: 0.0251
Epoch 9/50
3543/3554 [============================>.] - ETA: 0s - loss: 1.6464e-05 -
accuracy: 0.2691
Epoch 9: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.6460e-0
5 - accuracy: 0.2690 - val_loss: 2.4035e-05 - val_accuracy: 0.0271
Epoch 10/50
3551/3554 [============================>.] - ETA: 0s - loss: 1.6378e-05 -
accuracy: 0.2717
Epoch 10: val_loss did not improve from 0.00002
3554/3554 [==============================] - 4s 1ms/step - loss: 1.6380e-0
5 - accuracy: 0.2717 - val_loss: 2.5148e-05 - val_accuracy: 0.0713
Epoch 11/50
3507/3554 [============================>.] - ETA: 0s - loss: 1.6261e-05 -
accuracy: 0.2759
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: 1.6261e-0
5 - accuracy: 0.2757 - val_loss: 2.4422e-05 - val_accuracy: 0.0692
Epoch 11: early stopping
```

In [32]:

```
#Plot training and test loss

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()
```



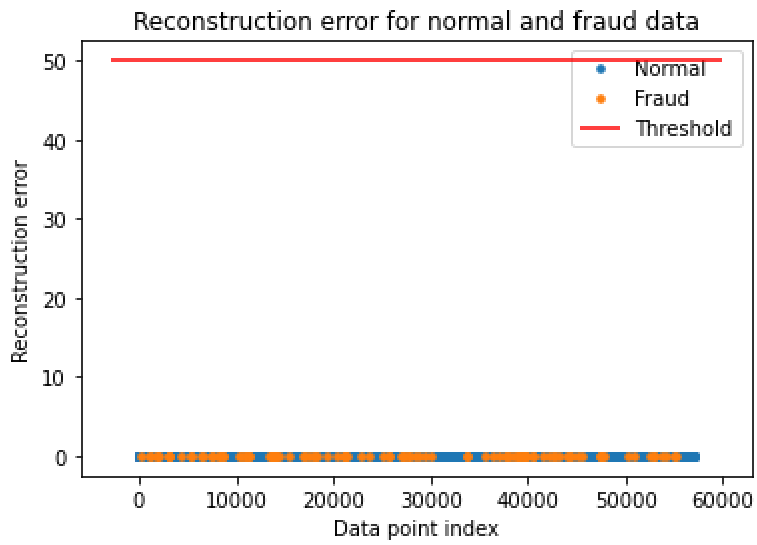
In [33]:

```
"""Detect Anomalies on test data

Anomalies are data points where the reconstruction loss is higher

To calculate the reconstruction loss on test data,
predict the test data and calculate the mean square error between the test data and the

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 680us/step
```

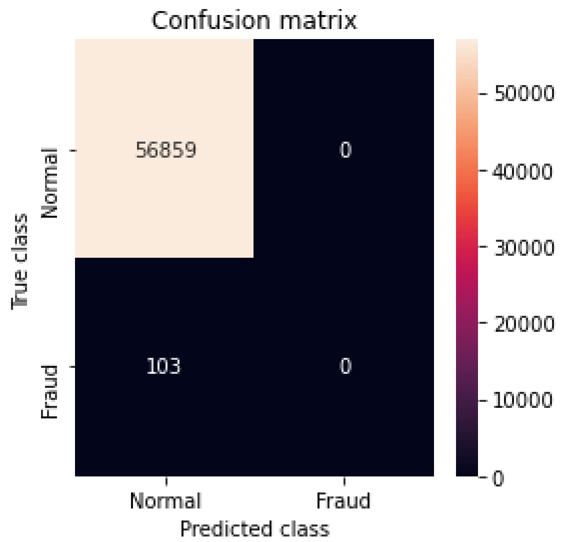
In [34]:

```
#Plotting the test data points and their respective reconstruction error sets a thresho
#if the threshold value needs to be adjusted.

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,
ax.legend()
plt.title("Reconstruction error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();
```



In [35]:

```
'''Detect anomalies as points where the reconstruction loss is greater than a fixed thr
Here we see that a value of 52 for the threshold will be good.

Evaluating the performance of the anomaly detection'''

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:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.
y:1221: 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 [ ]:

```
'''As our dataset is highly imbalanced, we see a high accuracy but a low recall and pre

Things to further improve precision and recall would add more relevant features,
different architecture for autoencoder, different hyperparameters, or a different algor
```

In [ ]:

```
history
```