In [78]:
```
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 [79]:
```
# !pip install tensorflow --user
# !pip install keras
# !pip install daytime
# !pip install torch
```

In [80]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score,
RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

In [81]:
```
dataset = pd.read_csv("creditcard.csv")
#dataset.head
print(list(dataset.columns))
dataset.describe()
```

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

Out[81]:

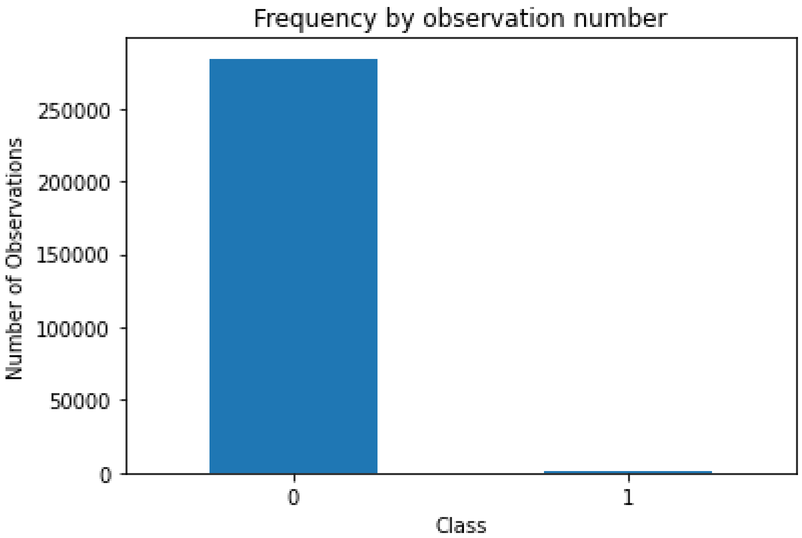
	Time	V1	V2	V3	V4	V5	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.84
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.48
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.33
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.61
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.68
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.74
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.98
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.33

8 rows × 31 columns

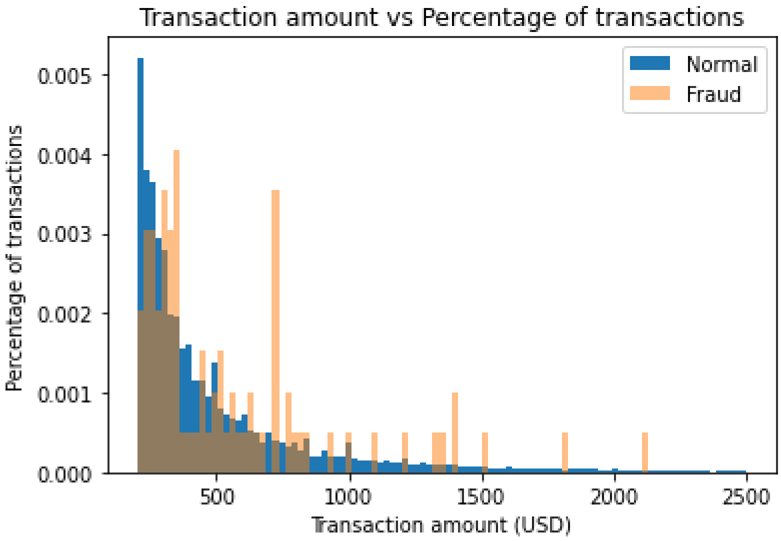
In [82]:
```
#check for any nullvalues
print("Any nulls in the dataset ",dataset.isnull().values.any() )
print('-------')
print("No. of unique labels ", len(dataset['Class'].unique()))
print("Label values ",dataset.Class.unique())
#0 is for normal credit card transaction
#1 is for fraudulent credit card 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
Class
0    284315
1       492
Name: count, dtype: int64
```

In [83]:
```
#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 [84]:
```
# 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='Nor
plt.hist(fraud_dataset.Amount, bins=bins, alpha=0.5, density=True, label='Fr
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 [85]:
```
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 [86]:
```
raw_data = dataset.values
# The last element contains if the transaction is normal which is represente
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, la
```

In [87]:
```
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 [88]:

```
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 [89]:
```
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

#input Layer
input_layer = tf.keras.layers.Input(shape=(input_dim, ))
#Encoder
encoder = tf.keras.layers.Dense(encoding_dim, activation="tanh",activity_reg
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)(e
# 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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 30)]              0

 dense_12 (Dense)            (None, 14)                434

 dropout_4 (Dropout)         (None, 14)                0

 dense_13 (Dense)            (None, 7)                 105

 dense_14 (Dense)            (None, 4)                 32

 dense_15 (Dense)            (None, 7)                 35

 dropout_5 (Dropout)         (None, 7)                 0

 dense_16 (Dense)            (None, 14)                112

 dense_17 (Dense)            (None, 30)                450

=================================================================
Total params: 1168 (4.56 KB)
Trainable params: 1168 (4.56 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [90]:
```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mode
# 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 [91]:
```
#Compile the Autoencoder

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

#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
```

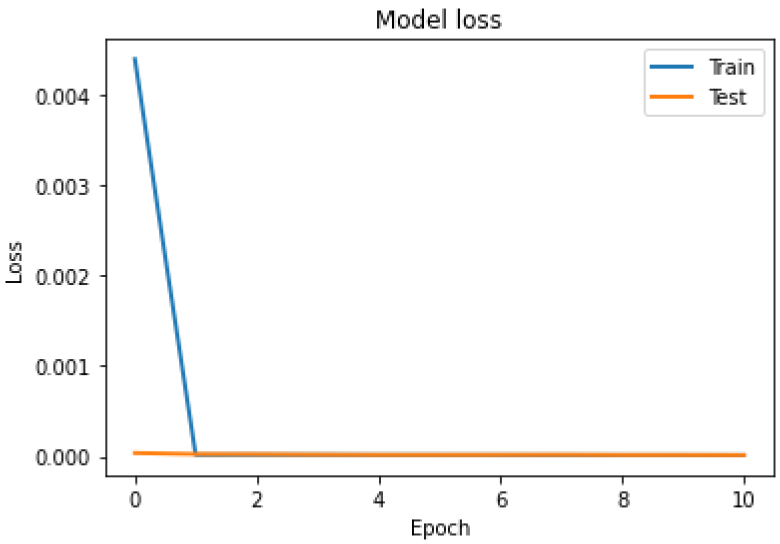
```
Epoch 1/50
3526/3554 [============================>.] - ETA: 0s - loss: 0.0044 - accur
acy: 0.0377
Epoch 1: val_loss improved from inf to 0.00004, saving model to autoencoder
_fraud.h5
3554/3554 [==============================] - 4s 838us/step - loss: 0.0044 -
accuracy: 0.0377 - val_loss: 3.9357e-05 - val_accuracy: 0.1279
Epoch 2/50
 227/3554 [>.............................] - ETA: 2s - loss: 1.9374e-05 - a
ccuracy: 0.0516

/home/student/.local/lib/python3.10/site-packages/keras/src/engine/trainin
g.py:3000: UserWarning: You are saving your model as an HDF5 file via `mode
l.save()`. This file format is considered legacy. We recommend using instea
d the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
3507/3554 [============================>.] - ETA: 0s - loss: 1.9476e-05 - a
ccuracy: 0.0674
Epoch 2: val_loss improved from 0.00004 to 0.00003, saving model to autoenc
oder_fraud.h5
3554/3554 [==============================] - 3s 803us/step - loss: 1.9478e-
05 - accuracy: 0.0676 - val_loss: 2.7531e-05 - val_accuracy: 0.1279
Epoch 3/50
3543/3554 [============================>.] - ETA: 0s - loss: 1.9434e-05 - a
ccuracy: 0.0702
Epoch 3: val_loss improved from 0.00003 to 0.00003, saving model to autoenc
oder_fraud.h5
3554/3554 [==============================] - 3s 805us/step - loss: 1.9424e-
05 - accuracy: 0.0702 - val_loss: 2.6025e-05 - val_accuracy: 0.1279
Epoch 4/50
3517/3554 [============================>.] - ETA: 0s - loss: 1.9119e-05 - a
ccuracy: 0.0867
Epoch 4: val_loss improved from 0.00003 to 0.00002, saving model to autoenc
oder_fraud.h5
3554/3554 [==============================] - 3s 796us/step - loss: 1.9116e-
05 - accuracy: 0.0873 - val_loss: 2.1979e-05 - val_accuracy: 0.1304
Epoch 5/50
3518/3554 [============================>.] - ETA: 0s - loss: 1.8351e-05 - a
ccuracy: 0.1363
Epoch 5: val_loss improved from 0.00002 to 0.00002, saving model to autoenc
oder_fraud.h5
3554/3554 [==============================] - 3s 798us/step - loss: 1.8336e-
05 - accuracy: 0.1366 - val_loss: 1.9372e-05 - val_accuracy: 0.1435
Epoch 6/50
3485/3554 [============================>.] - ETA: 0s - loss: 1.8130e-05 - a
ccuracy: 0.1362
Epoch 6: val_loss improved from 0.00002 to 0.00002, saving model to autoenc
oder_fraud.h5
3554/3554 [==============================] - 3s 804us/step - loss: 1.8119e-
05 - accuracy: 0.1365 - val_loss: 1.9140e-05 - val_accuracy: 0.1871
Epoch 7/50
3549/3554 [============================>.] - ETA: 0s - loss: 1.7495e-05 - a
ccuracy: 0.1854
Epoch 7: val_loss did not improve from 0.00002
3554/3554 [==============================] - 3s 799us/step - loss: 1.7493e-
05 - accuracy: 0.1855 - val_loss: 1.9924e-05 - val_accuracy: 0.2464
Epoch 8/50
3518/3554 [============================>.] - ETA: 0s - loss: 1.6879e-05 - a
ccuracy: 0.2491
Epoch 8: val_loss did not improve from 0.00002
3554/3554 [==============================] - 3s 791us/step - loss: 1.6873e-
05 - accuracy: 0.2495 - val_loss: 2.0869e-05 - val_accuracy: 0.2480
Epoch 9/50
3517/3554 [============================>.] - ETA: 0s - loss: 1.6562e-05 - a
ccuracy: 0.2736
Epoch 9: val_loss improved from 0.00002 to 0.00002, saving model to autoenc
oder_fraud.h5
3554/3554 [==============================] - 3s 798us/step - loss: 1.6560e-
05 - accuracy: 0.2736 - val_loss: 1.8248e-05 - val_accuracy: 0.2607
Epoch 10/50
3527/3554 [============================>.] - ETA: 0s - loss: 1.6383e-05 - a
ccuracy: 0.2862
Epoch 10: val_loss improved from 0.00002 to 0.00002, saving model to autoen
coder_fraud.h5
3554/3554 [==============================] - 3s 809us/step - loss: 1.6379e-
05 - accuracy: 0.2863 - val_loss: 1.8081e-05 - val_accuracy: 0.2658
Epoch 11/50
3515/3554 [============================>.] - ETA: 0s - loss: 1.6024e-05 - a
ccuracy: 0.3019
Epoch 11: val_loss improved from 0.00002 to 0.00002, saving model to autoen
coder_fraud.h5
```

```
Restoring model weights from the end of the best epoch: 1.
3554/3554 [==============================] - 3s 800us/step - loss: 1.6030e-05 - accuracy: 0.3019 - val_loss: 1.6610e-05 - val_accuracy: 0.3025
Epoch 11: early stopping
```

In [92]:
```
#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()
```



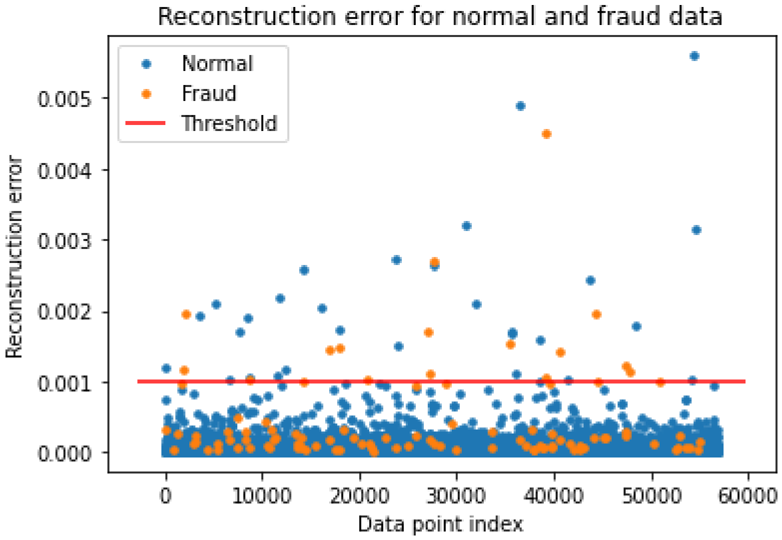
In [93]:
```
test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2), axis=1)
error_df = pd.DataFrame({'Reconstruction_error': mse,
    'True_class': test_labels})
```

```
1781/1781 [==============================] - 1s 462us/step
```

In [94]:

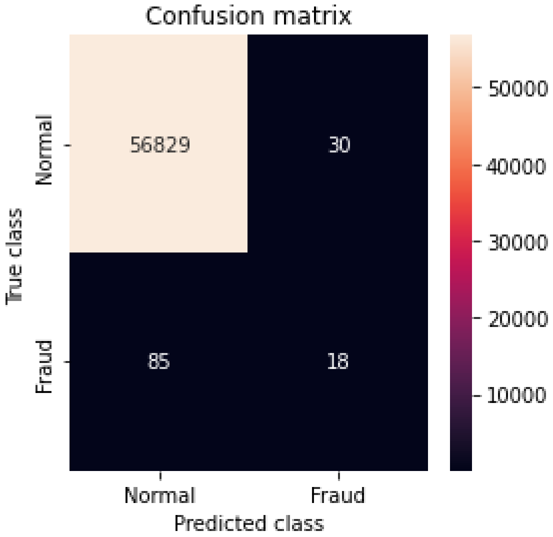
```
threshold_fixed = 0.001
groups = error_df.groupby('True_class')
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.index, group.Reconstruction_error, marker='o', ms=3.5, lin

ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", z
ax.legend()
plt.title("Reconstruction error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();
```



In [103]:

```
threshold_fixed =0.001
pred_y = [1 if e > threshold_fixed else 0 for e in error_df.Reconstruction_e
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,
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:  0.9979811102138267
 Recall:  0.17475728155339806
 Precision:  0.375
```

In [ ]: