In [3]:

```
pip install --user tensorflow
```

```
     ---------------------------------------- 440.8/440.8 kB 3.1 MB/s eta
0:00:00
Collecting termcolor>=1.1.0
  Using cached termcolor-2.3.0-py3-none-any.whl (6.9 kB)
Requirement already satisfied: numpy<=1.24.3,>=1.22 in c:\users\user\an
aconda3\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow)
(1.23.5)
Collecting absl-py>=1.0.0
  Downloading absl_py-1.4.0-py3-none-any.whl (126 kB)
     ---------------------------------------- 126.5/126.5 kB 7.3 MB/s eta
0:00:00
Collecting gast<=0.4.0,>=0.2.1
  Using cached gast-0.4.0-py3-none-any.whl (9.8 kB)
Requirement already satisfied: six>=1.12.0 in c:\users\user\appdata\roa
ming\python\python310\site-packages (from tensorflow-intel==2.13.0->ten
sorflow) (1.16.0)
Collecting opt-einsum>=2.3.2
  Using cached opt_einsum-3.3.0-py3-none-any.whl (65 kB)
Collecting google-pasta>=0.1.1
```

In [4]:

```
# Importing libraries

import numpy as np
import zipfile
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dropout
from tensorflow.keras import callbacks
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.callbacks import EarlyStopping

# Clustering
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn import mixture

# 3D Visualization
import plotly.express as px
import plotly as py
import plotly.graph_objs as go
```

In [5]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import ShuffleSplit, train_test_split
from sklearn.metrics import confusion_matrix, precision_recall_curve
from sklearn.metrics import recall_score, classification_report, auc, roc_curve
from sklearn.metrics import precision_recall_fscore_support, f1_score

from numpy.random import seed
seed(7)
from tensorflow.random import set_seed
set_seed(11)
from sklearn.model_selection import train_test_split

SEED = 123 #used to help randomly select the data points
DATA_SPLIT_PCT = 0.2
```

In [8]:

```
df = pd.read_csv('sensor.csv')
```

In [9]:

```
# consider that sensor was having an issue and its giving the same reading for these tim
# I used forward fill propagation as a method of dealing with missing values

df.fillna(method = 'ffill' , inplace = True)
```

In [10]:

```
df.head()
```

Out[10]:

	Unnamed: 0	timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_0
0	0	2018-04-01 00:00:00	2.465394	47.09201	53.2118	46.310760	634.3750	76.4597
1	1	2018-04-01 00:01:00	2.465394	47.09201	53.2118	46.310760	634.3750	76.4597
2	2	2018-04-01 00:02:00	2.444734	47.35243	53.2118	46.397570	638.8889	73.5459
3	3	2018-04-01 00:03:00	2.460474	47.09201	53.1684	46.397568	628.1250	76.9889
4	4	2018-04-01 00:04:00	2.445718	47.13541	53.2118	46.397568	636.4583	76.5889

5 rows × 55 columns

In [11]:

```
df.index = df.timestamp
df.drop(columns=['Unnamed: 0', 'timestamp','sensor_51','sensor_15'], inplace = True)
```

In [12]:

```
input_X = df.loc[:, df.columns != 'machine_status'].values  # converts the df to a numpy
input_y = df['machine_status'].values

n_features = input_X.shape[1]  # number of features
```

In [13]:

```
input_y
```

Out[13]:

```
array(['NORMAL', 'NORMAL', 'NORMAL', ..., 'NORMAL', 'NORMAL', 'NORMAL'],
      dtype=object)
```

In [14]:

```
input_y = pd.DataFrame(input_y, columns=['State'])
input_y['Label'] = input_y['State'] != 'NORMAL'
```

In [15]:

```
input_y['Label']
```

Out[15]:

```
0         False
1         False
2         False
3         False
4         False
          ...
220315    False
220316    False
220317    False
220318    False
220319    False
Name: Label, Length: 220320, dtype: bool
```

In [16]:

```
input_y.mean()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_10812\336824573.py:1: FutureWar
ning: The default value of numeric_only in DataFrame.mean is deprecated. I
n a future version, it will default to False. In addition, specifying 'num
eric_only=None' is deprecated. Select only valid columns or specify the va
lue of numeric_only to silence this warning.
  input_y.mean()
```

Out[16]:

```
Label    0.065741
dtype: float64
```

In [17]:

```
#labelencoder = LabelEncoder()
#input_y = pd.DataFrame(input_y, columns=['State'])
#input_y['Label'] = labelencoder.fit_transform(input_y['State'])
#input_y
```

In [18]:

```
input_y =input_y.Label.values
```

In [19]:

```
def temporalize(X, y, lookback):
    '''
    Inputs
    X         A 2D numpy array ordered by time of shape:
              (n_observations x n_features)
    y         A 1D numpy array with indexes aligned with
              X, i.e. y[i] should correspond to X[i].
              Shape: n_observations.
    lookback  The window size to look back in the past
              records. Shape: a scalar.

    Output
    output_X  A 3D numpy array of shape:
              ((n_observations-lookback-1) x lookback x
              n_features)
    output_y  A 1D array of shape:
              (n_observations-lookback-1), aligned with X.
    '''
    output_X = []
    output_y = []
    for i in range(len(X) - lookback - 1):
        t = []
        for j in range(1, lookback + 1):
            # Gather the past records upto the lookback period
            t.append(X[[(i + j + 1)], :])
        output_X.append(t)
        output_y.append(y[i + lookback + 1])
    return np.squeeze(np.array(output_X)), np.array(output_y)
```

In [20]:

```
lookback = 5  # Equivalent to 5 min of past data.
# Temporalize the data
X, y = temporalize(X = input_X, y = input_y, lookback = lookback)
```

In [21]:

```
y
```

Out[21]:

```
array([False, False, False, ..., False, False, False])
```

Train - Val - Test Split

In [22]:

```
X_train, X_test, y_train, y_test = train_test_split(np.array(X), np.array(y), test_size=
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=DATA_S
```

In [23]:

```
# When y = 1, the Status is Normal.  It means right functioning patterns
X_train_y0 = X_train[y_train!=1]
X_train_y1 = X_train[y_train==1]
X_valid_y0 = X_valid[y_valid!=1]
X_valid_y1 = X_valid[y_valid==1]
```

In [24]:

```
X_train = X_train.reshape(X_train.shape[0], lookback, n_features)
X_train_y0 = X_train_y0.reshape(X_train_y0.shape[0], lookback, n_features)
X_train_y1 = X_train_y1.reshape(X_train_y1.shape[0], lookback, n_features)
X_valid = X_valid.reshape(X_valid.shape[0], lookback, n_features)
X_valid_y0 = X_valid_y0.reshape(X_valid_y0.shape[0], lookback, n_features)
X_valid_y1 = X_valid_y1.reshape(X_valid_y1.shape[0], lookback, n_features)
X_test = X_test.reshape(X_test.shape[0], lookback, n_features)
```

In [ ]:

```

```

Data standardization

In [25]:

```
def flatten(X):
    '''
    Flatten a 3D array.

    Input
    X            A 3D array for lstm, where the array is sample x timesteps x features.

    Output
    flattened_X  A 2D array, sample x features.
    '''
    flattened_X = np.empty((X.shape[0], X.shape[2]))  # sample x features array.
    for i in range(X.shape[0]):
        flattened_X[i] = X[i, (X.shape[1]-1), :]
    return(flattened_X)
```

In [26]:

```
def scale(X, scaler):
    '''
    Scale 3D array.

    Inputs
    X            A 3D array for lstm, where the array is sample x timesteps x features.
    scaler       A scaler object, e.g., sklearn.preprocessing.StandardScaler, sklearn.pr

    Output
    X            Scaled 3D array.
    '''
    for i in range(X.shape[0]):
        X[i, :, :] = scaler.transform(X[i, :, :])

    return X
```

In [27]:

```
# Initialize a scaler using the training data.
scaler = StandardScaler().fit(flatten(X_train_y0))
```

In [28]:

```
X_train_y0_scaled = scale(X_train_y0, scaler)
```

In [29]:

```
a = flatten(X_train_y0_scaled)
print('colwise mean', np.mean(a, axis=0).round(6))
print('colwise variance', np.var(a, axis=0))
```

```
colwise mean [-0.  0.  0.  0. -0.  0.  0.  0. -0.  0.  0. -0.  0. -0.  0.
0. -0.  0.
 -0.  0.  0. -0. -0. -0.  0.  0.  0. -0. -0.  0.  0.  0.  0.  0. -0. -0.
 -0.  0.  0.  0.  0.  0.  0. -0. -0. -0. -0.  0.  0.  0.]
colwise variance [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1.]
```

In [30]:

```
X_valid_scaled = scale(X_valid, scaler)
X_valid_y0_scaled = scale(X_valid_y0, scaler)
X_test_scaled = scale(X_test, scaler)
```

Building architecture

In [31]:

```
timesteps =  X_train_y0_scaled.shape[1] # equal to the lookback
n_features =  X_train_y0_scaled.shape[2] # equal to the columns

epochs = 30
batch = 64
lr = 0.0001
```

In [32]:

```
# Recurrent autoencoder

#Encoder

input_layer = keras.Input(shape=(timesteps,n_features))
encoding_layer = layers.LSTM(32, activation='tanh', return_sequences=True)(input_layer)
hidden_layer = layers.LSTM(16,activation='tanh', return_sequences=False)(encoding_layer)

compact_features = layers.RepeatVector(timesteps)(hidden_layer)

#encoder_2d = keras.Model(inputs=input_layer, outputs = compact_features)

# Decoder

decoding_1layer = layers.LSTM(32, activation='tanh', return_sequences=True)(compact_feat
decoding_2layer = layers.LSTM(40, activation='tanh', return_sequences=True)(decoding_1la

decoding_3layer = layers.Dense(n_features)
output_layer = layers.TimeDistributed(decoding_3layer)(decoding_2layer)

autoencoder = keras.Model(inputs=input_layer, outputs=output_layer)
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 5, 50)]           0

 lstm (LSTM)                 (None, 5, 32)             10624

 lstm_1 (LSTM)               (None, 16)                3136

 repeat_vector (RepeatVecto  (None, 5, 16)             0
 r)

 lstm_2 (LSTM)               (None, 5, 32)             6272

 lstm_3 (LSTM)               (None, 5, 40)             11680

 time_distributed (TimeDist  (None, 5, 50)             2050
 ributed)

=================================================================
Total params: 33762 (131.88 KB)
Trainable params: 33762 (131.88 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

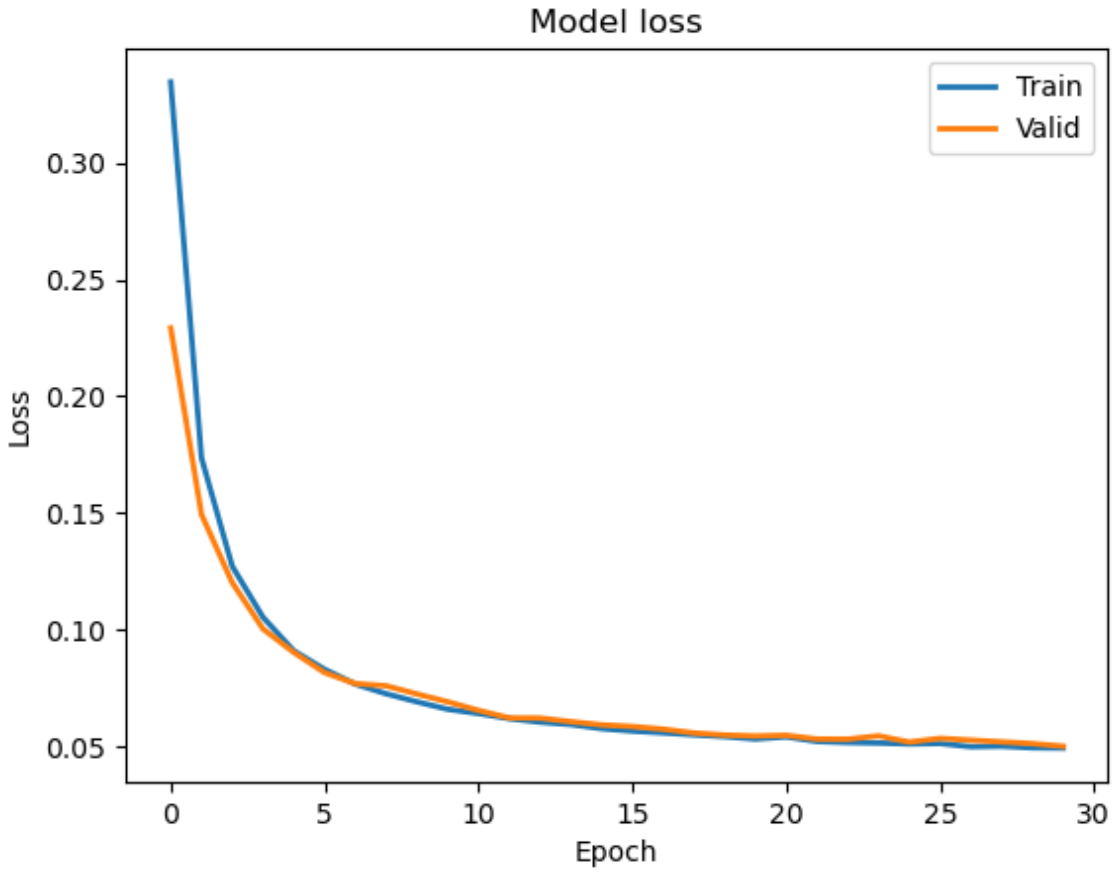
In [33]:

```
autoencoder.compile(loss='mse', optimizer='adam')

cp = ModelCheckpoint(filepath="lstm_autoencoder_classifier.h5",
                                save_best_only=True,
                                verbose=0)

tb = TensorBoard(log_dir='./logs',
                 histogram_freq=0,
                 write_graph=True,
                 write_images=True)



lstm_autoencoder = autoencoder.fit(X_train_y0_scaled, X_train_y0_scaled,
                                #validation_split = 0.33,
                                 validation_data=(X_valid_y0_scaled, X_valid_y0_scaled),
                                 epochs = epochs,
                                 #callbacks = [E_Stop],
                                 batch_size = batch,
                                 shuffle = True,
                                 verbose=2
                                 )
```

```
Epoch 1/30
2060/2060 - 63s - loss: 0.3345 - val_loss: 0.2292 - 63s/epoch - 30ms/st
ep
Epoch 2/30
2060/2060 - 44s - loss: 0.1737 - val_loss: 0.1495 - 44s/epoch - 21ms/st
ep
Epoch 3/30
2060/2060 - 44s - loss: 0.1273 - val_loss: 0.1203 - 44s/epoch - 21ms/st
ep
Epoch 4/30
2060/2060 - 45s - loss: 0.1055 - val_loss: 0.1005 - 45s/epoch - 22ms/st
ep
Epoch 5/30
2060/2060 - 42s - loss: 0.0911 - val_loss: 0.0904 - 42s/epoch - 20ms/st
ep
Epoch 6/30
2060/2060 - 44s - loss: 0.0832 - val_loss: 0.0818 - 44s/epoch - 22ms/st
ep
Epoch 7/30
```

In [34]:

```
plt.plot(lstm_autoencoder.history['loss'], linewidth=2, label='Train')
plt.plot(lstm_autoencoder.history['val_loss'], linewidth=2, label='Valid')
plt.legend(loc='upper right')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```



Threshold definition

In [35]:

```
from sklearn.metrics import confusion_matrix, precision_recall_curve
from sklearn.metrics import recall_score, classification_report, auc, roc_curve
from sklearn.metrics import precision_recall_fscore_support, f1_score
```

In [36]:

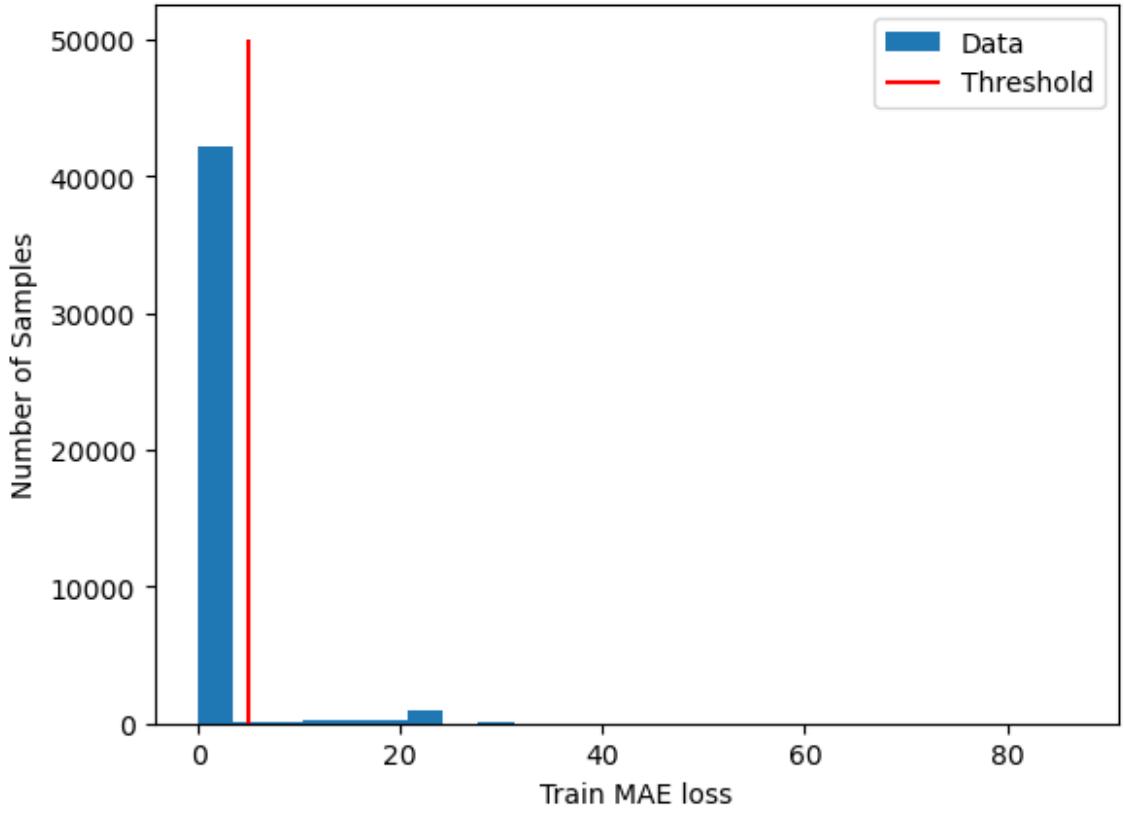
```
valid_x_predictions = autoencoder.predict(X_test_scaled)
mse = np.mean(np.power(flatten(X_test_scaled) - flatten(valid_x_predictions), 2), axis=1

error_df = pd.DataFrame({'Reconstruction_error': mse,
                         'True_class': y_test.tolist()})
```

```
1377/1377 [==============================] - 10s 5ms/step
```

In [37]:

```
plt.figure()
plt.hist(mse, bins=25, label='Data')
plt.xlabel('Train MAE loss')
plt.ylabel('Number of Samples')

threshold = np.mean(mse) + np.std(mse)
plt.vlines(threshold, ymin=0, ymax=50000, colors='r', label='Threshold')

plt.legend()
print(round(threshold,3))
plt.show()
```

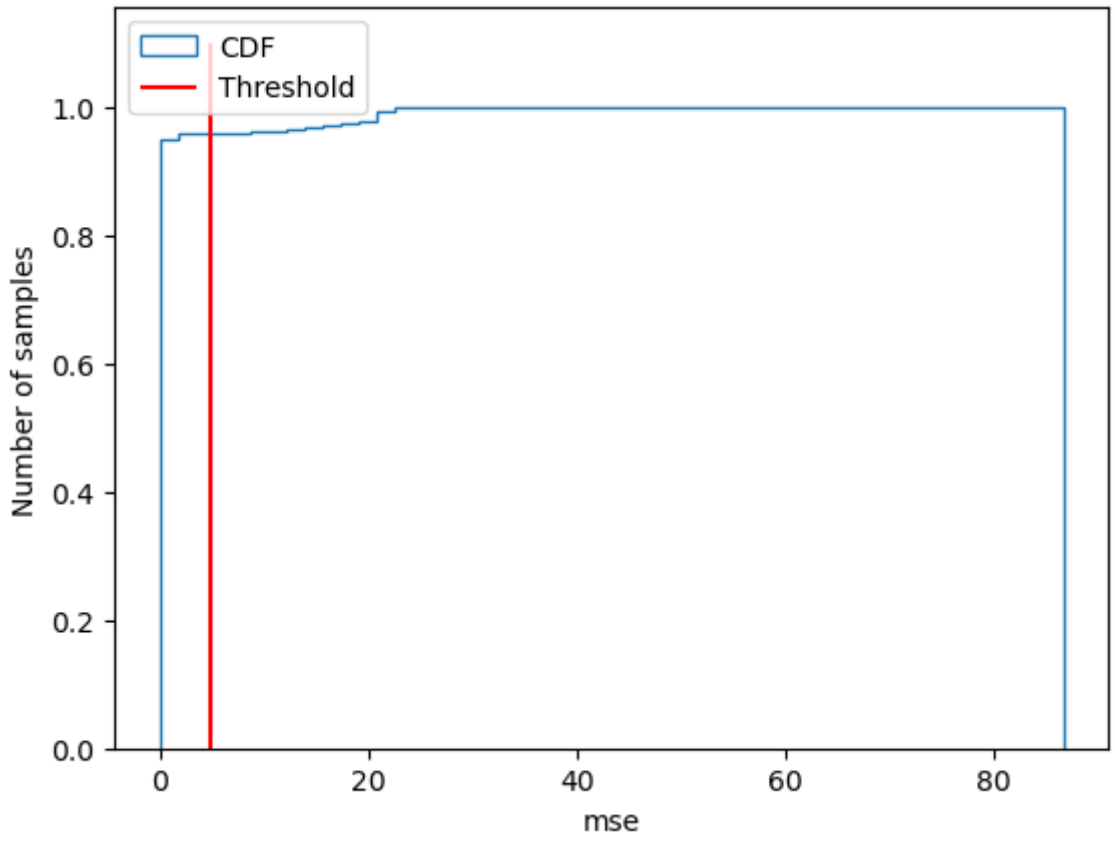
```
4.882
```



In [38]:

```
plt.figure()
plt.hist(mse, bins=50,
         cumulative=True,
         label='CDF',
         histtype='step',
         density=True
        )
plt.xlabel('mse')
plt.ylabel('Number of samples');

threshold = np.mean(mse) + np.std(mse)
plt.vlines(threshold, ymin=0, ymax=1.1, colors='r', label='Threshold')

plt.legend()
print(threshold)
plt.show()
```

```
4.882224503420016
```

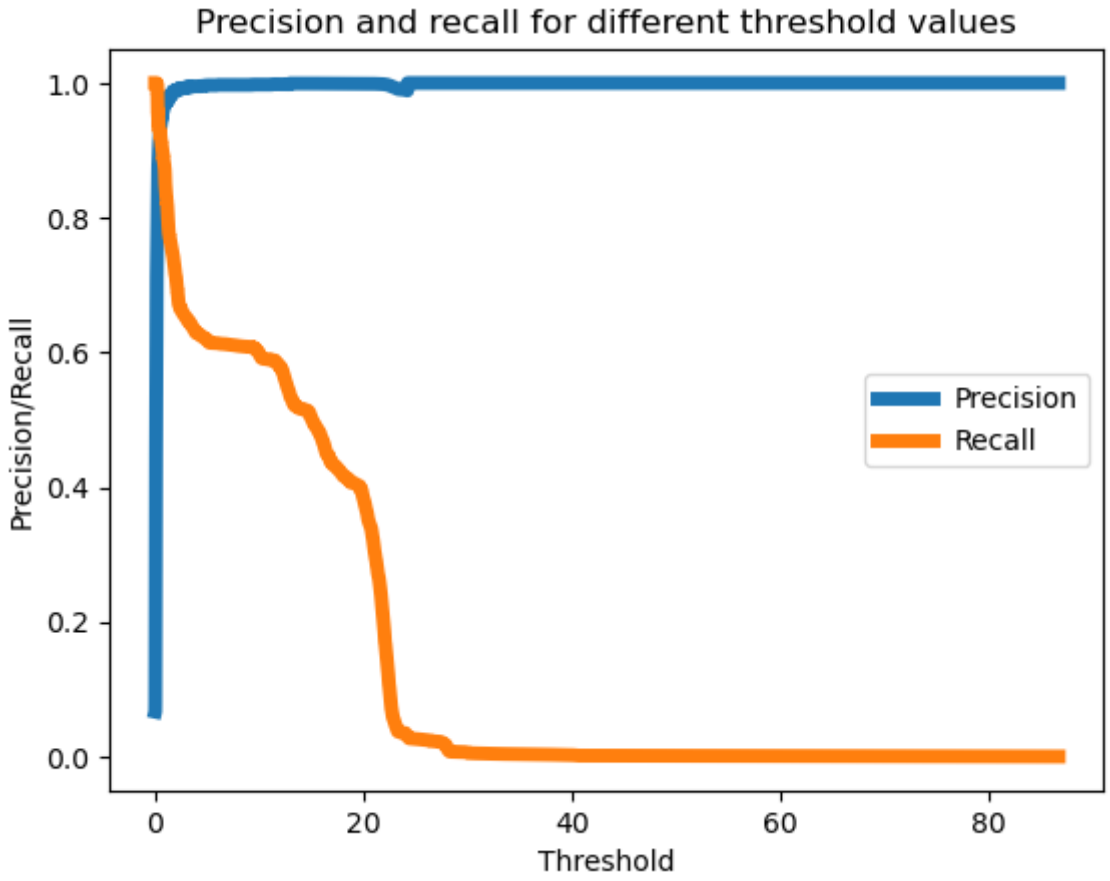


In [39]:

```
precision_rt, recall_rt, threshold_rt = precision_recall_curve(error_df.True_class, erro
```

In [40]:

```
plt.plot(threshold_rt, precision_rt[1:], label="Precision",linewidth=5)
plt.plot(threshold_rt, recall_rt[1:], label="Recall",linewidth=5)
plt.title('Precision and recall for different threshold values')
plt.xlabel('Threshold')
plt.ylabel('Precision/Recall')
plt.legend()
plt.show()
```



In [41]:

```
test_x_predictions = autoencoder.predict(X_test_scaled)
mse = np.mean(np.power(flatten(X_test_scaled) - flatten(test_x_predictions), 2), axis=1)

error_df = pd.DataFrame({'Reconstruction_error': mse,
                         'True_class': y_test.tolist()})

threshold_fixed = 2.3
groups = error_df.groupby('True_class')
```

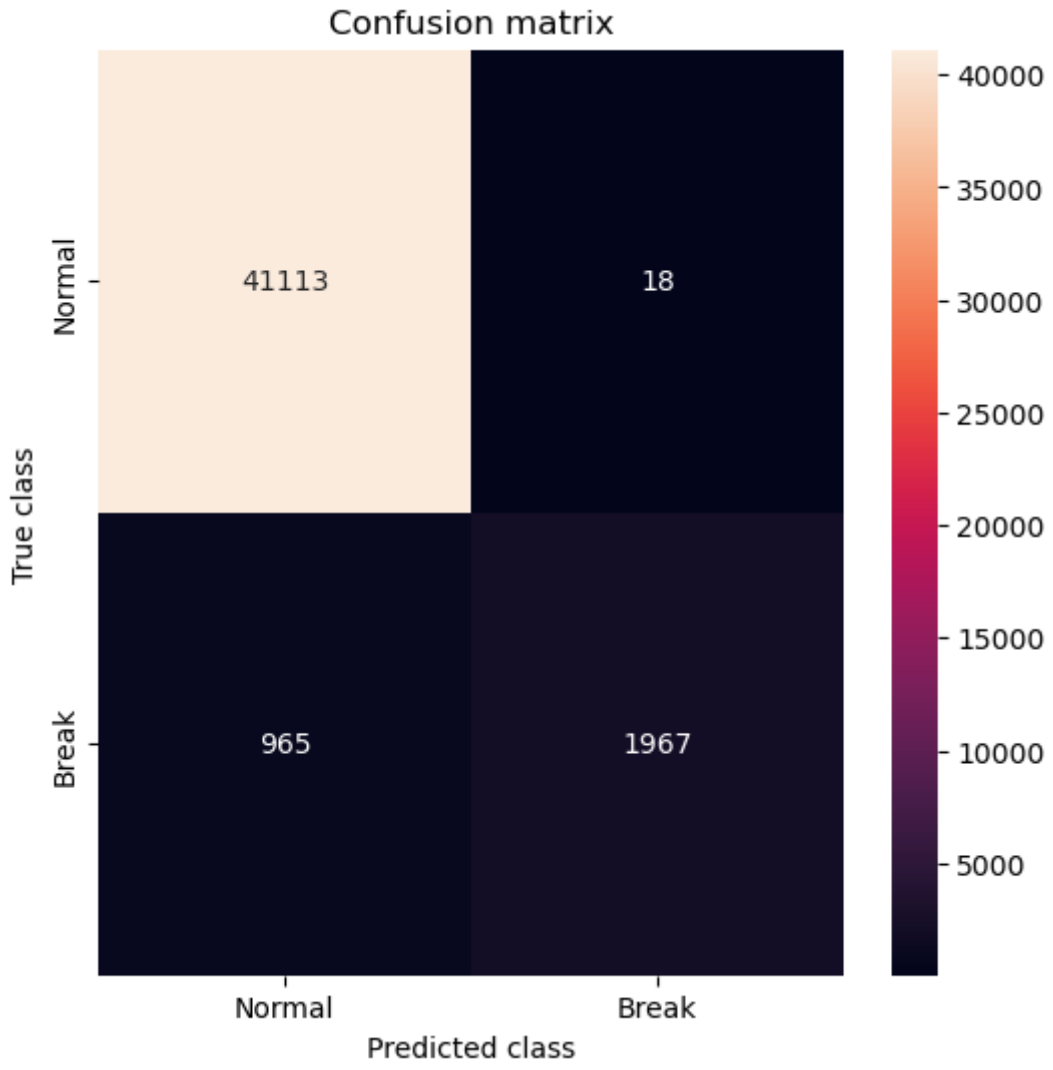
```
1377/1377 [==============================] - 8s 6ms/step
```

In [51]:

```
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index, group.Reconstruction_error, marker='o', ms=3.5, linestyle='',
            label= "Break" if name == False else "Normal",  alpha=0.2)
ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", zorder=100, l
ax.legend()
plt.title("Reconstruction error for different classes")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();
```

In [43]:

```
pred_y = [1 if e > threshold_fixed else 0 for e in error_df.Reconstruction_error.values]
conf_matrix = confusion_matrix(error_df.True_class, pred_y)
LABELS = ['Normal', 'Break']
plt.figure(figsize=(6, 6))
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()
```



In [44]:

```
false_pos_rate, true_pos_rate, thresholds = roc_curve(error_df.True_class, error_df.Reco
roc_auc = auc(false_pos_rate, true_pos_rate,)

plt.plot(false_pos_rate, true_pos_rate, linewidth=5, label='AUC = %0.3f'% roc_auc)
plt.plot([0,1],[0,1], linewidth=5)

plt.xlim([-0.01, 1])
plt.ylim([0, 1.01])
plt.legend(loc='lower right')
plt.title('Receiver operating characteristic curve (ROC)')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```