In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input,Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import MeanAbsoluteError
```

In [2]:
```
data = pd.read_csv('creditcard.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3637
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2554
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5146
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3870
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8177

In [4]:
```
data.shape
```

Out[4]:
```
(284807, 31)
```

In [5]:
```
x = data.drop(['Time','Class'],axis=1)
y = data['Class']
```

In [6]:
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

In [7]:
```
x_train,x_test = train_test_split(x_scaled,test_size=0.2,random_state=35)
```

In [14]:
```
input_dim = x_train.shape[1]
encoding_dim = 2
input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim,activation='relu')(input_layer)
decoder = Dense(input_dim,activation='sigmoid')(encoder)
autoencoder = Model(inputs=input_layer,outputs=decoder)
autoencoder.compile(optimizer=Adam(learning_rate=0.001),loss=MeanSquaredError(),metric
```

In [17]:
```
history = autoencoder.fit(x_train,x_train,epochs=10,validation_data=(x_test,x_test))
```

```
Epoch 1/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 14s 2ms/step - loss: 0.9352 - mean_absolute_error: 0.6
281 - val_loss: 0.9705 - val_mean_absolute_error: 0.6280
Epoch 2/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 11s 2ms/step - loss: 0.9218 - mean_absolute_error: 0.6
254 - val_loss: 0.9693 - val_mean_absolute_error: 0.6273
Epoch 3/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step - loss: 0.9084 - mean_absolute_error: 0.6
247 - val_loss: 0.9686 - val_mean_absolute_error: 0.6268
Epoch 4/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step - loss: 0.9187 - mean_absolute_error: 0.6
250 - val_loss: 0.9681 - val_mean_absolute_error: 0.6265
Epoch 5/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step - loss: 0.9215 - mean_absolute_error: 0.6
258 - val_loss: 0.9679 - val_mean_absolute_error: 0.6264
Epoch 6/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 14s 2ms/step - loss: 0.9243 - mean_absolute_error: 0.6
249 - val_loss: 0.9677 - val_mean_absolute_error: 0.6263
Epoch 7/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step - loss: 0.9144 - mean_absolute_error: 0.6
246 - val_loss: 0.9676 - val_mean_absolute_error: 0.6262
Epoch 8/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step - loss: 0.9103 - mean_absolute_error: 0.6
240 - val_loss: 0.9674 - val_mean_absolute_error: 0.6261
Epoch 9/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 22s 3ms/step - loss: 0.9219 - mean_absolute_error: 0.6
255 - val_loss: 0.9674 - val_mean_absolute_error: 0.6261
Epoch 10/10
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 25s 4ms/step - loss: 0.9152 - mean_absolute_error: 0.6
246 - val_loss: 0.9672 - val_mean_absolute_error: 0.6261
```

In [18]:
```
x_train_pred = autoencoder.predict(x_train)
x_test_pred = autoencoder.predict(x_test)
train_r_error = np.mean(np.abs(x_train_pred - x_train),axis = 1)
test_r_error = np.mean(np.abs(x_test_pred - x_test),axis = 1)
```

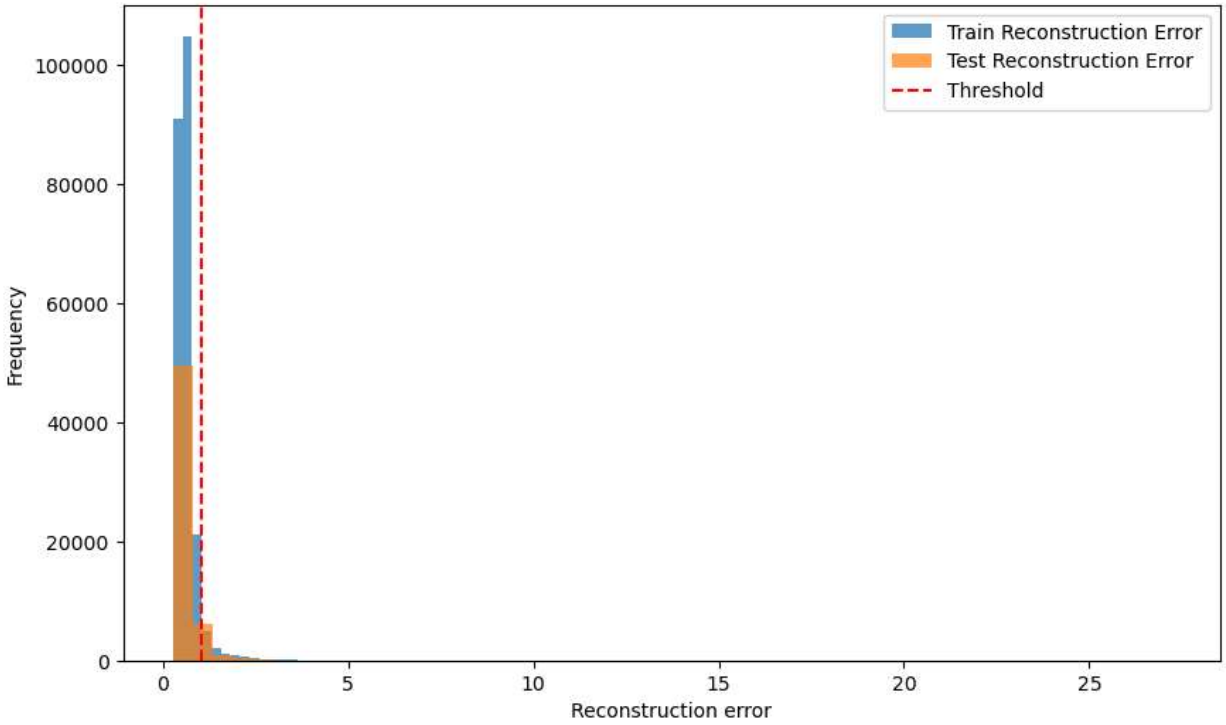
```
7121/7121 ━━━━━━━━━━━━━━━━━━━━ 6s 809us/step
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 2s 960us/step
```

In [19]:
```
threshold = np.percentile(train_r_error,95)
anamoly = test_r_error > threshold

print("Threshold:",threshold)
print("Number of anamoly:",np.sum(anamoly))
```

```
Threshold: 1.0247256017529378
Number of anamoly: 2874
```

In [20]:
```
plt.figure(figsize=(10, 6))
plt.hist(train_r_error, bins=50, alpha=0.7, label='Train Reconstruction Error')
plt.hist(test_r_error, bins=50, alpha=0.7, label='Test Reconstruction Error')
plt.axvline(threshold, color='red', linestyle='--', label='Threshold')
plt.xlabel("Reconstruction error")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```


Train Reconstruction Error
Test Reconstruction Error
Threshold
Frequency
Reconstruction error
0
20000
40000
60000
80000
100000
0
5
10
15
20
25

In [ ]: