```
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

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

```
            Time         V1         V2        V3        V4
V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321

1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018

2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198

3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309

4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193

...          ...        ...        ...       ...       ...       ...

284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473

284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229

284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515

284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961

284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546


               V6        V7        V8        V9  ...       V21
V22  \
0        0.462388  0.239599  0.098698  0.363787  ... -0.018307
0.277838
1       -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -
0.638672
2        1.800499  0.791461  0.247676 -1.514654  ...  0.247998
0.771679
3        1.247203  0.237609  0.377436 -1.387024  ... -0.108300
0.005274
4        0.095921  0.592941 -0.270533  0.817739  ... -0.009431
0.798278
...           ...       ...       ...       ...  ...       ...       ..
.
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454
```

```
0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205
0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045
0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245
0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057
0.643078

              V23       V24       V25       V26       V27       V28
Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
69.99
...          ...       ...       ...       ...       ...       ...
...
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649
217.00

        Class
0           0
1           0
2           0
3           0
4           0
...       ...
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]
```

```
scaler = StandardScaler()
X = df.drop('Class', axis=1)
y = df['Class']

X_scaled = scaler.fit_transform(X)

X_train = X_scaled[y == 0]
X_test, y_test = X_scaled, y



input_dim = X_train.shape[1]

# Encoder
input_layer = Input(shape=(input_dim,))
encoded = Dense(16, activation='relu')(input_layer)
encoded = Dense(8, activation='relu')(encoded)
bottleneck = Dense(4, activation='relu')(encoded)

# Decoder
decoded = Dense(8, activation='relu')(bottleneck)
decoded = Dense(16, activation='relu')(decoded)
output_layer = Dense(input_dim, activation='sigmoid')(decoded)

autoencoder = Model(inputs=input_layer, outputs=output_layer)
autoencoder.compile(optimizer='adam', loss='mse')

history = autoencoder.fit(
    X_train, X_train,
    epochs=5,
    batch_size=32,
    validation_split=0.2,
    shuffle=True
)
```

```
Epoch 1/5
7108/7108 ━━━━━━━━━━━━━━━━━━━━ 40s 5ms/step - loss: 0.8701 - val_loss:
0.8779
Epoch 2/5
7108/7108 ━━━━━━━━━━━━━━━━━━━━ 36s 5ms/step - loss: 0.8284 - val_loss:
0.8661
Epoch 3/5
7108/7108 ━━━━━━━━━━━━━━━━━━━━ 36s 5ms/step - loss: 0.8190 - val_loss:
0.8589
Epoch 4/5
7108/7108 ━━━━━━━━━━━━━━━━━━━━ 1137s 160ms/step - loss: 0.8149 -
val_loss: 0.8556
Epoch 5/5
7108/7108 ━━━━━━━━━━━━━━━━━━━━ 43s 6ms/step - loss: 0.8124 - val_loss:
0.8532
```

```
reconstructions = autoencoder.predict(X_test)
mse = np.mean(np.power(X_test - reconstructions, 2), axis=1)
```

```
8901/8901 ━━━━━━━━━━━━━━━━━━━━ 22s 2ms/step
```

```
threshold = np.percentile(mse, 95)
print("Reconstruction error threshold:", threshold)
```

```
Reconstruction error threshold: 1.6717150409912815
```

```
y_pred = [1 if e > threshold else 0 for e in mse]

from sklearn.metrics import confusion_matrix, classification_report

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[270505  13810]
 [    61    431]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.98    284315
           1       0.03      0.88      0.06       492

    accuracy                           0.95    284807
   macro avg       0.52      0.91      0.52    284807
weighted avg       1.00      0.95      0.97    284807
```

```
import numpy as np

random_index = np.random.randint(0, X_test.shape[0])
print(f"random",random_index)

random_transaction = X_test[random_index]
true_label = y_test[random_index]

reconstructed = autoencoder.predict(random_transaction.reshape(1, -1))

mse = np.mean(np.power(random_transaction - reconstructed, 2))

if mse > threshold:
    print(f"□ Predicted: FRAUD (Error = {mse:.4f})")
else:
    print(f"□ Predicted: NORMAL (Error = {mse:.4f})")

print(f"□ Actual Label: {'FRAUD' if true_label == 1 else 'NORMAL'}")
```

```
random 211114
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step
 Predicted: NORMAL (Error = 0.3416)
 Actual Label: NORMAL
```