In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, RocCurveDisplay, PrecisionRecall
```

In [2]:
```
import os
os.getcwd()
```

Out[2]: 'C:\\Users\\LENOVO'

In [3]:
```
data=pd.read_csv("C:\\Users\\LENOVO\\Desktop\\Project\\Credit Card Fraud Detection\
```

In [4]:
```
data.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

In [5]:
```
data.describe()
```

Out[5]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8480
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.6040
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.3802
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.1374
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.9159
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.4335
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.1192
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.4801

8 rows × 31 columns

In [6]:
```
data.rename(columns={'Class':'Case'}, inplace='True')
```

In [7]:
```
data.head()
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

In [8]:
```
f=0                          #to count number of fraud and legitimate cases
l=0
Case=data['Case']
for x in Case:
    if x==1:
        f=f+1
    else:
        l=l+1
print(f)
print(l)
```

```
492
284315
```

As we can see its an imbalanced data, we balance it using SMOTE

In [9]:
```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
```

In [10]:
```
data.head()
```

Out[10]:

	Time	V1	V2	V3	V4	V5	V6	V7	'
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

In [11]:
```
# must have all the data except that of 'Time' and 'Case'
```

In [12]:
```
X=data.drop(['Time','Case'], axis=1)
y=data['Case']
```

Splitting to Train/Test

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_stat
```

Using SMOTE

In [14]:
```
over = SMOTE(sampling_strategy=0.5)
under = RandomUnderSampler(sampling_strategy=1)
steps = [('o',over), ('u', under)] #first over sampling the minority and then under
pipeline=Pipeline(steps=steps)
```

In [15]:
```
X_resample, y_resample = pipeline.fit_resample(X_train, y_train) #performing the pi
y_resample.value_counts(normalize=True) #returning if this was successfull
```

Out[15]:
```
0    0.5
1    0.5
Name: Case, dtype: float64
```

In [16]:
```
#balance in y achieved
```

Scaling the data as it might be required for our Algorithms

In [17]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler() #StandardScaler takes your data and makes it look more li
X_resample_scaled = scaler.fit_transform(X_resample)
X_test_scaled = scaler.transform(X_test)
```

Applying Isolation Forest for Anomaly Detection

In [18]:
```
from sklearn.ensemble import IsolationForest
```

In [19]:
```
iso = IsolationForest(n_estimators=200, random_state=42) #n_estimators: This parame
iso.fit(X_resample_scaled) # This step allows the model to learn the normal pattern
y_pred_iso = iso.predict(X_test_scaled) #The predictions will be either -1 (anomaly
y_pred_iso = [1 if x ==-1 else 0 for x in y_pred_iso] #change the result (-1,1) to
y_score_iso = iso.decision_function(X_test_scaled) #Lower scores typically indicate
```

In [20]:
```
print(len(y_score_iso))
```

```
56962
```

In [21]:
```
#KNOWING HOW ACCURATE WERE WE IN THE MODEL
```

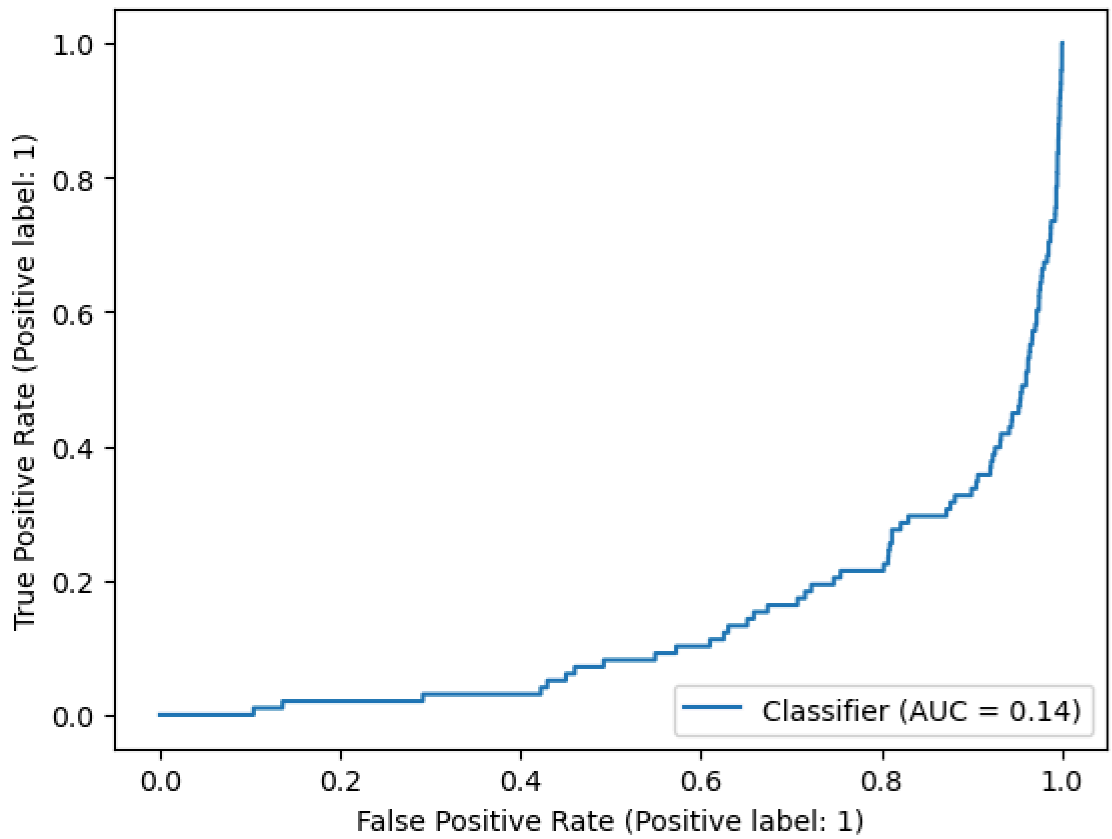
In [22]:
```
print(classification_report(y_test, y_pred_iso))
RocCurveDisplay.from_predictions(y_test, y_score_iso)
PrecisionRecallDisplay.from_predictions(y_test, y_score_iso)
```

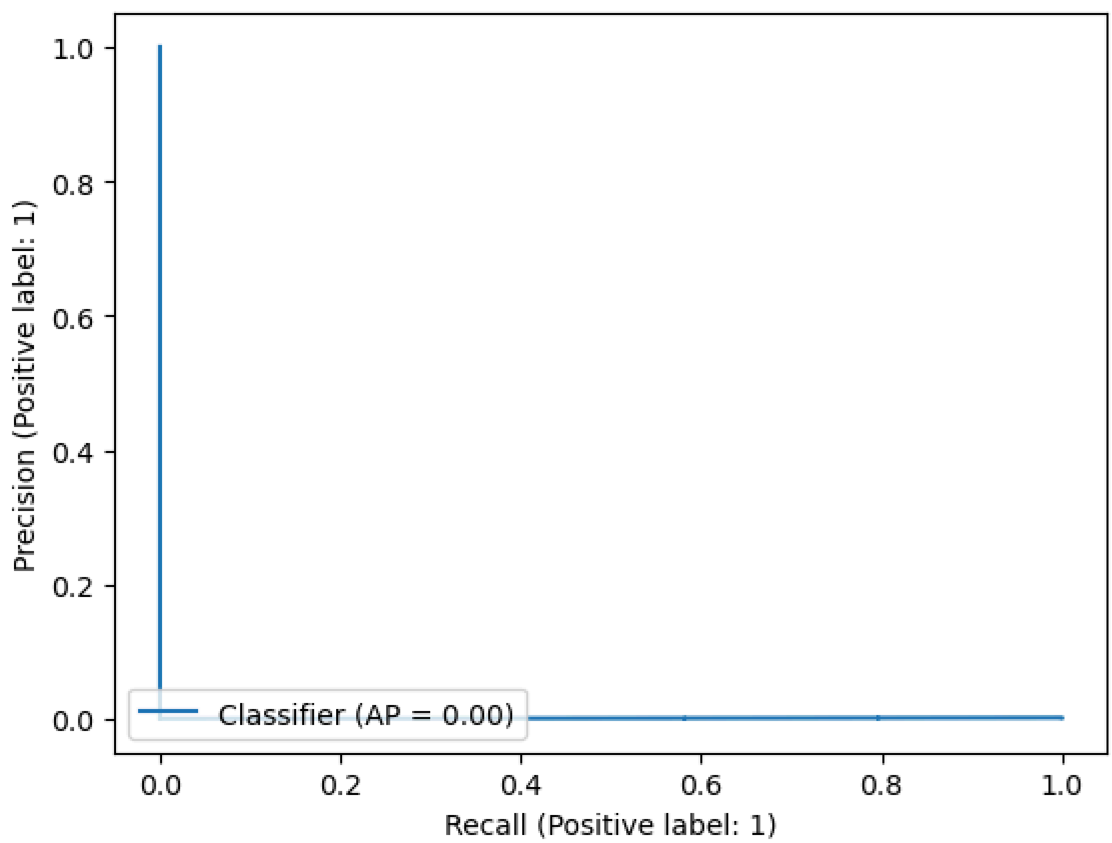
```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56864
           1       0.04      0.27      0.07        98

    accuracy                           0.99     56962
   macro avg       0.52      0.63      0.53     56962
weighted avg       1.00      0.99      0.99     56962
```

Out[22]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x26539d66
380>
```





model is highly accurate at identifying normal transactions but struggles to correctly identify fraudulent transactions

Applying OneClassSVM

In [23]:
```
#from sklearn.svm import OneClassSVM
```

In [24]:
```
#one_class_svm = OneClassSVM(kernel='rbf', nu=0.01) #lower the value of nu, more se
#one_class_svm.fit(X_resample_scaled)
#y_pred_svm = one_class_svm.predict(X_test_scaled)
#y_pred_svm = [1 if x == -1 else 0 for x in y_pred_svm]
#y_score_svm = one_class_svm.decision_function(X_test_scaled)
```

In [25]:
```
#As the f1 and precison is extremely low for the Fraudulent cases
```

Applying DBSCAN to find Anomalies in the data

In [26]:
```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
```

In [27]:
```
# DBSCAN for Anomaly Detection
db = DBSCAN(eps=0.5, min_samples=5)
db.fit(X_resample_scaled)
y_pred_db_test = db.fit_predict(X_test_scaled)
y_pred_db_test = [1 if x == -1 else 0 for x in y_pred_db_test]  # Convert to 0 for
```

In [28]:
```
print(classification_report(y_test, y_pred_db_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.21      0.34     56864
           1       0.00      1.00      0.00        98

    accuracy                           0.21     56962
   macro avg       0.50      0.60      0.17     56962
weighted avg       1.00      0.21      0.34     56962
```

In [29]:
```
#Took a lot of time, also this is not giving us the desired results, although bette
```

Finally using AutoEncoder

In [30]:
```
#When the autoencoder notices something unusual, it's like a signal that something
```

In [31]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input
```

In [32]:
```
#re-scaling and fitting data
```

In [33]:
```
data_normal = data[data['Case'] == 0]
data_fraud = data[data['Case'] == 1]
X_normal = data_normal.drop(['Time','Case'], axis=1)
X_fraud = data_fraud.drop(['Time','Case'], axis=1)

X_normal_scaled = scaler.fit_transform(X_normal)
X_fraud_scaled = scaler.transform(X_fraud)
# split X_normal_scaled to X_train, X_test
X_normal_train, X_normal_test = train_test_split(X_normal_scaled, test_size=0.2, ra
```

In [34]:
```
# create encoder
input_layer = Input(shape=(X_normal_train.shape[1],))
encoder = Dense(100, activation='relu')(input_layer) #layer1
encoder = Dense(50, activation='relu')(encoder) #layer2 for deeper understanding of
encoder = Dense(25, activation='relu')(encoder) #layer3 for yet deeper understandin
# create decoder
decoder = Dense(50, activation='relu')(encoder) #puting back and recreating data 1
decoder = Dense(100, activation='relu')(decoder) #again
output_layer = Dense(X_normal_train.shape[1], activation='relu')(decoder) #puting i

autoencoder = Model(input_layer, output_layer)
autoencoder.compile(optimizer='adam', loss='mse') # tries to minimize something cal

# add early stop
from tensorflow.keras.callbacks import EarlyStopping # It helps prevent the machine
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=3)
```

In [35]:
```
autoencoder.fit(X_normal_train, X_normal_train, epochs=30, batch_size=256, shuffle=
#higher epoch, better at detection
```

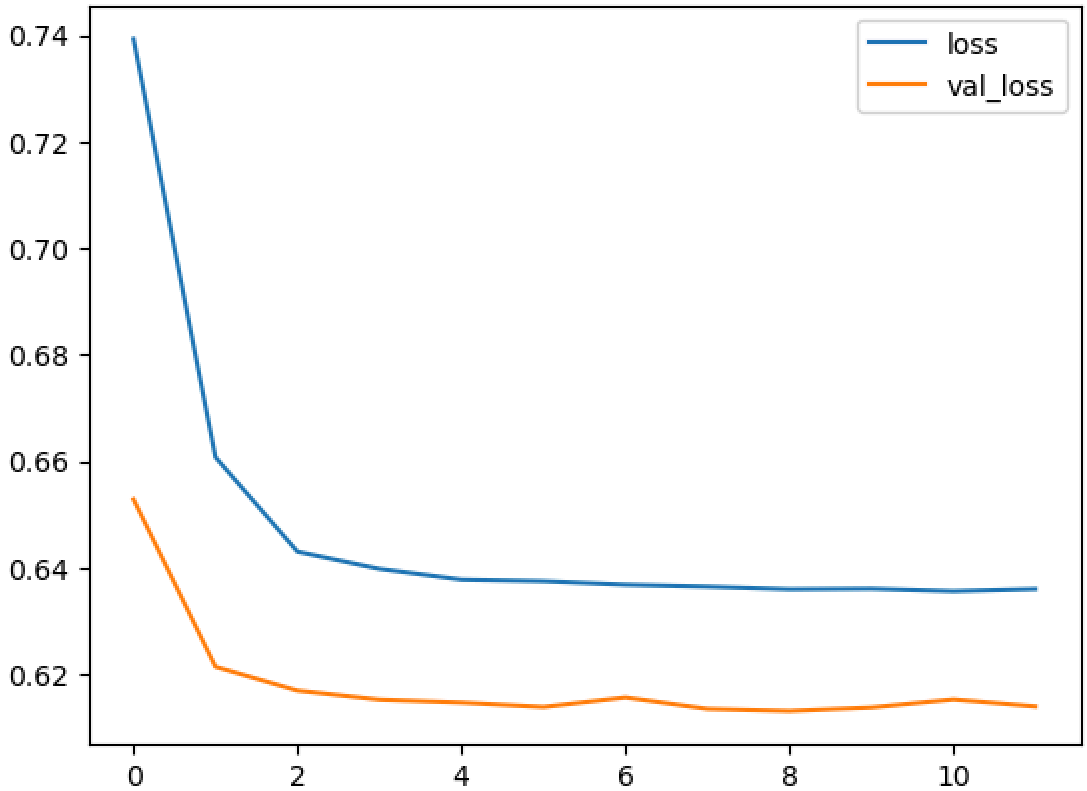
```
Epoch 1/30
711/711 [==============================] - 6s 6ms/step - loss: 0.7393 - val_loss: 0.
6528
Epoch 2/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6608 - val_loss: 0.
6214
Epoch 3/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6430 - val_loss: 0.
6169
Epoch 4/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6398 - val_loss: 0.
6152
Epoch 5/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6378 - val_loss: 0.
6147
Epoch 6/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6375 - val_loss: 0.
6139
Epoch 7/30
711/711 [==============================] - 4s 6ms/step - loss: 0.6368 - val_loss: 0.
6157
Epoch 8/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6365 - val_loss: 0.
6135
Epoch 9/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6360 - val_loss: 0.
6131
Epoch 10/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6361 - val_loss: 0.
6138
Epoch 11/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6356 - val_loss: 0.
6153
Epoch 12/30
711/711 [==============================] - 4s 5ms/step - loss: 0.6360 - val_loss: 0.
6140
```

Out[35]:
```
<keras.src.callbacks.History at 0x26559258b50>
```

In [36]:
```python
# If you see the training loss dropping but the validation loss increasing, it migh
```

In [37]:
```python
pd.DataFrame(autoencoder.history.history).plot()
```

Out[37]:
```
<Axes: >
```



Interpretation:

If both the training loss and validation loss are decreasing and remain close to each other, it suggests that the model is learning effectively and generalizing well. This is a good sign.

If the training loss decreases while the validation loss increases or doesn't decrease, it might be a sign of overfitting. In such cases, you may need to consider techniques like regularization or adjusting the model's complexity.

If both training and validation losses are high and not improving, it might indicate that the model needs further tuning or that the data is too complex for the chosen architecture.

Sometimes, loss curves might have fluctuations, especially for small datasets or complex models. However, the general trend should be downward for effective learning.

In [38]:
```
X_test = np.concatenate([X_normal_test, X_fraud_scaled], axis=0) #concatenating to
y_test = np.concatenate([np.zeros(X_normal_test.shape[0]), np.ones(X_fraud_scaled.s
# predict and count error
y_pred = autoencoder.predict(X_test)
mse = np.mean(np.power(X_test - y_pred, 2), axis=1) #mean square error
```

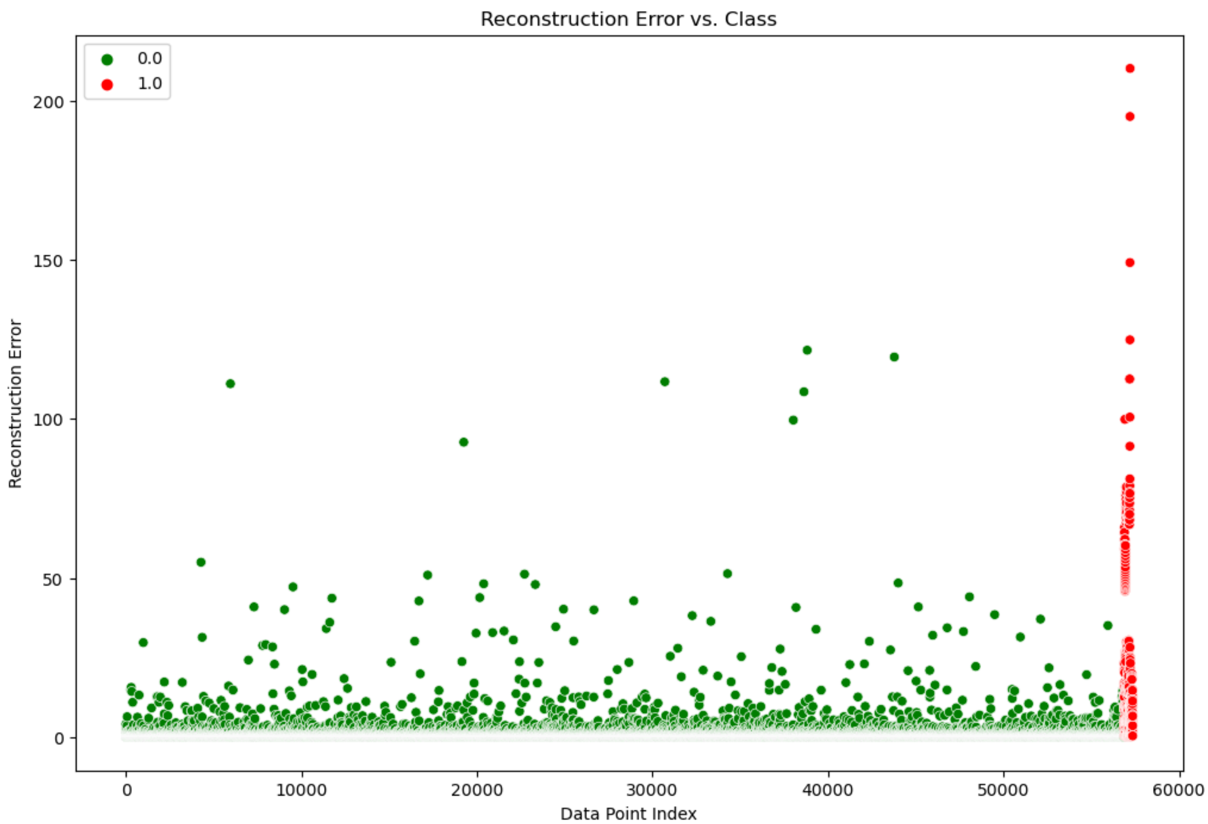
```
1793/1793 [==============================] - 4s 2ms/step
```

In [39]:
```
from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test, mse)
roc_auc
```

Out[39]: 0.9537108088489392

Closer the score is to 1, better the accuracy at prediction

In [40]:
```
# data visualize
import seaborn as sns
plt.figure(figsize=(12, 8))
sns.scatterplot(x=range(len(mse)), y=mse, hue=y_test, palette={0: 'g', 1: 'r'})
plt.title('Reconstruction Error vs. Class')
plt.xlabel('Data Point Index')
plt.ylabel('Reconstruction Error')
plt.show()
```



As the red part is denser near the lower values of y axis (lower reconstruction error) model is more accurate.