Use Autoencoder to implement anomaly detection. Build the model by using the following:

a. Import required libraries b. Upload/access the dataset c. The encoder converts it into a latent representation d. Decoder networks convert it back to the original input e. Compile the models with Optimizer, Loss, and Evaluation Metrics

-Tushar Bhagat Roll_No. 07

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers

# Step 1: Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

# Step 2: Preprocess the data
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Reshape the data for the autoencoder (flatten the images)
X_train = X_train.reshape((X_train.shape[0], -1))
X_test = X_test.reshape((X_test.shape[0], -1))

# Use only '0's as normal class and others as anomalies
normal_class = 0
X_train_normal = X_train[y_train == normal_class]
X_test_normal = X_test[y_test == normal_class]
X_test_anomalous = X_test[y_test != normal_class]

# Step 3: Normalize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_normal)
X_test_scaled = scaler.transform(X_test_normal)
X_test_anomalous_scaled = scaler.transform(X_test_anomalous)
```

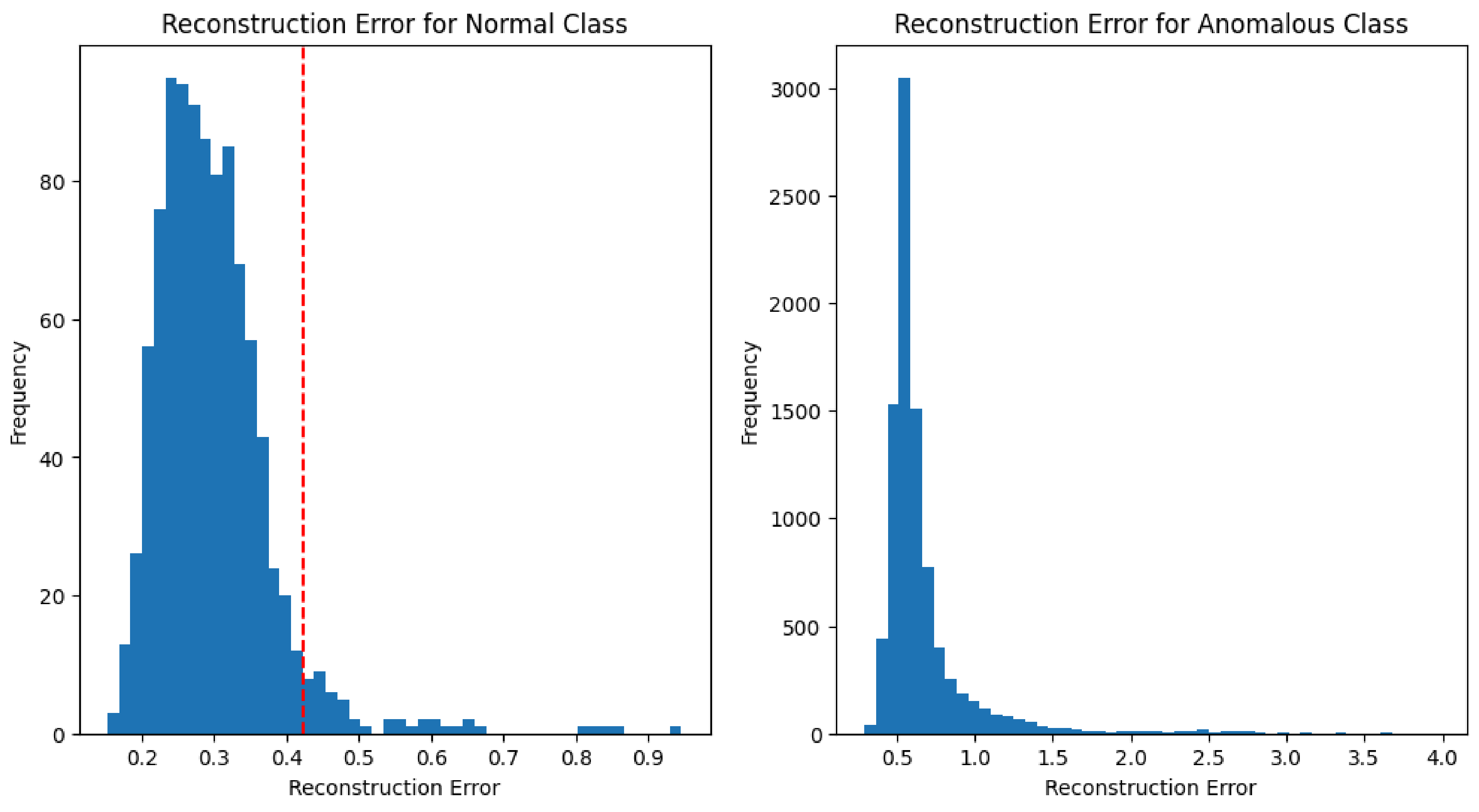
In [2]:
```
# Step 4: Build the Autoencoder Model
# Encoder
input_dim = X_train_scaled.shape[1]  # Number of features
encoder_input = layers.Input(shape=(input_dim,))
encoded = layers.Dense(128, activation='relu')(encoder_input)
encoded = layers.Dense(64, activation='relu')(encoded)  # Latent space

# Decoder
decoded = layers.Dense(128, activation='relu')(encoded)
decoded = layers.Dense(input_dim, activation='sigmoid')(decoded)  # Output layer

# Create the Autoencoder Model
autoencoder = keras.Model(inputs=encoder_input, outputs=decoded)

# Step 5: Compile the Model
autoencoder.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Step 6: Train the Autoencoder
history = autoencoder.fit(X_train_scaled, X_train_scaled,
                          epochs=50,
                          batch_size=256,
                          validation_split=0.2)
```

```
Epoch 1/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.9395 - mae: 0.6917 - val_loss: 0.6312 - val_mae: 0.4255
Epoch 2/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.7001 - mae: 0.4155 - val_loss: 0.5850 - val_mae: 0.3902
Epoch 3/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.6756 - mae: 0.3891 - val_loss: 0.5543 - val_mae: 0.3742
Epoch 4/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.6039 - mae: 0.3750 - val_loss: 0.5301 - val_mae: 0.3584
Epoch 5/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5814 - mae: 0.3573 - val_loss: 0.5112 - val_mae: 0.3441
Epoch 6/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5827 - mae: 0.3444 - val_loss: 0.4965 - val_mae: 0.3358
Epoch 7/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.6291 - mae: 0.3381 - val_loss: 0.4857 - val_mae: 0.3289
Epoch 8/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.5764 - mae: 0.3304 - val_loss: 0.4789 - val_mae: 0.3242
Epoch 9/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5784 - mae: 0.3270 - val_loss: 0.4730 - val_mae: 0.3202
Epoch 10/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5425 - mae: 0.3197 - val_loss: 0.4690 - val_mae: 0.3173
Epoch 11/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5740 - mae: 0.3185 - val_loss: 0.4662 - val_mae: 0.3154
Epoch 12/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5294 - mae: 0.3149 - val_loss: 0.4637 - val_mae: 0.3138
Epoch 13/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5473 - mae: 0.3140 - val_loss: 0.4620 - val_mae: 0.3120
Epoch 14/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.5509 - mae: 0.3133 - val_loss: 0.4602 - val_mae: 0.3110
Epoch 15/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5344 - mae: 0.3108 - val_loss: 0.4587 - val_mae: 0.3100
Epoch 16/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5276 - mae: 0.3094 - val_loss: 0.4577 - val_mae: 0.3092
Epoch 17/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.6071 - mae: 0.3120 - val_loss: 0.4563 - val_mae: 0.3079
Epoch 18/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.5356 - mae: 0.3093 - val_loss: 0.4555 - val_mae: 0.3074
Epoch 19/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5068 - mae: 0.3090 - val_loss: 0.4546 - val_mae: 0.3067
Epoch 20/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.5123 - mae: 0.3052 - val_loss: 0.4539 - val_mae: 0.3062
Epoch 21/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5365 - mae: 0.3064 - val_loss: 0.4533 - val_mae: 0.3058
Epoch 22/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5406 - mae: 0.3073 - val_loss: 0.4528 - val_mae: 0.3053
Epoch 23/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.5485 - mae: 0.3068 - val_loss: 0.4523 - val_mae: 0.3046
Epoch 24/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5271 - mae: 0.3056 - val_loss: 0.4517 - val_mae: 0.3046
Epoch 25/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5464 - mae: 0.3045 - val_loss: 0.4516 - val_mae: 0.3040
Epoch 26/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5225 - mae: 0.3049 - val_loss: 0.4509 - val_mae: 0.3036
Epoch 27/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5012 - mae: 0.3031 - val_loss: 0.4507 - val_mae: 0.3034
Epoch 28/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5101 - mae: 0.3025 - val_loss: 0.4501 - val_mae: 0.3032
Epoch 29/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.4826 - mae: 0.3011 - val_loss: 0.4501 - val_mae: 0.3028
Epoch 30/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5362 - mae: 0.3028 - val_loss: 0.4495 - val_mae: 0.3026
Epoch 31/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5679 - mae: 0.3048 - val_loss: 0.4492 - val_mae: 0.3021
Epoch 32/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.5097 - mae: 0.3011 - val_loss: 0.4489 - val_mae: 0.3021
Epoch 33/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5100 - mae: 0.3016 - val_loss: 0.4486 - val_mae: 0.3019
Epoch 34/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5724 - mae: 0.3043 - val_loss: 0.4482 - val_mae: 0.3015
Epoch 35/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5318 - mae: 0.3014 - val_loss: 0.4479 - val_mae: 0.3014
Epoch 36/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.4921 - mae: 0.3000 - val_loss: 0.4476 - val_mae: 0.3011
Epoch 37/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5234 - mae: 0.3008 - val_loss: 0.4473 - val_mae: 0.3009
Epoch 38/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5441 - mae: 0.3014 - val_loss: 0.4470 - val_mae: 0.3006
Epoch 39/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.4987 - mae: 0.2983 - val_loss: 0.4469 - val_mae: 0.3006
Epoch 40/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5239 - mae: 0.2996 - val_loss: 0.4465 - val_mae: 0.3006
Epoch 41/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5245 - mae: 0.3007 - val_loss: 0.4464 - val_mae: 0.3003
Epoch 42/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5266 - mae: 0.2991 - val_loss: 0.4462 - val_mae: 0.3000
Epoch 43/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5234 - mae: 0.2996 - val_loss: 0.4458 - val_mae: 0.2999
Epoch 44/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5082 - mae: 0.2992 - val_loss: 0.4456 - val_mae: 0.2998
Epoch 45/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5562 - mae: 0.3010 - val_loss: 0.4455 - val_mae: 0.2995
```

```
Epoch 46/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.5287 - mae: 0.2976 - val_loss: 0.4454 - val_mae: 0.2994
Epoch 47/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5339 - mae: 0.2992 - val_loss: 0.4454 - val_mae: 0.2994
Epoch 48/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.5274 - mae: 0.2984 - val_loss: 0.4452 - val_mae: 0.2993
Epoch 49/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.5374 - mae: 0.2995 - val_loss: 0.4449 - val_mae: 0.2990
Epoch 50/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.5845 - mae: 0.2981 - val_loss: 0.4447 - val_mae: 0.2990
```

In [3]:
```python
# Step 7: Evaluate Anomalies
# Predict using the autoencoder
X_test_normal_pred = autoencoder.predict(X_test_scaled)

# Calculate reconstruction error
reconstruction_error_normal = np.mean(np.abs(X_test_scaled - X_test_normal_pred), axis=1)

# Set a threshold for anomaly detection
threshold = np.percentile(reconstruction_error_normal, 95)  # Top 5% as anomalies

# Evaluate on normal and anomalous data
X_test_anomalous_pred = autoencoder.predict(X_test_anomalous_scaled)
reconstruction_error_anomalous = np.mean(np.abs(X_test_anomalous_scaled - X_test_anomalous_pred), axis=1)

# Identify anomalies
normal_anomalies = reconstruction_error_normal > threshold
anomalous_anomalies = reconstruction_error_anomalous > threshold

# Print results
print(f"Number of normal anomalies detected: {np.sum(normal_anomalies)}")
print(f"Number of anomalous data points: {len(X_test_anomalous)}")
print(f"Anomalous points correctly detected: {np.sum(anomalous_anomalies)}")

# Visualize reconstruction errors
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(reconstruction_error_normal, bins=50)
plt.axvline(threshold, color='r', linestyle='--')
plt.title('Reconstruction Error for Normal Class')
plt.xlabel('Reconstruction Error')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(reconstruction_error_anomalous, bins=50)
plt.title('Reconstruction Error for Anomalous Class')
plt.xlabel('Reconstruction Error')
plt.ylabel('Frequency')

plt.show()
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
282/282 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Number of normal anomalies detected: 49
Number of anomalous data points: 9020
Anomalous points correctly detected: 8728
```


Reconstruction Error for Normal Class
Frequency
Reconstruction Error
Reconstruction Error for Anomalous Class
Frequency
Reconstruction Error