In [2]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredLogarithmicError
```

In [10]:
```
# Download the dataset
data = pd.read_csv('ecg.csv', header = None)
data.head()

#data.shape
```

Out[10]:

	0	1	2	3	4	5	6	7	8	
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818286	-1.250522	-(
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258	-0.754680	(
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659	-1.183580	-(
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131	-1.333884	-(
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.594450	-(

5 rows × 141 columns

In [11]:
```
# last column is the target
# 0 = anomaly, 1 = normal
TARGET = 140

features = data.drop(TARGET, axis=1)
target = data[TARGET]

x_train, x_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2, stratify=target
)
```

In [12]:
```
# use case is novelty detection so use only the normal data
# for training
train_index = y_train[y_train == 1].index
train_data = x_train.loc[train_index]
```

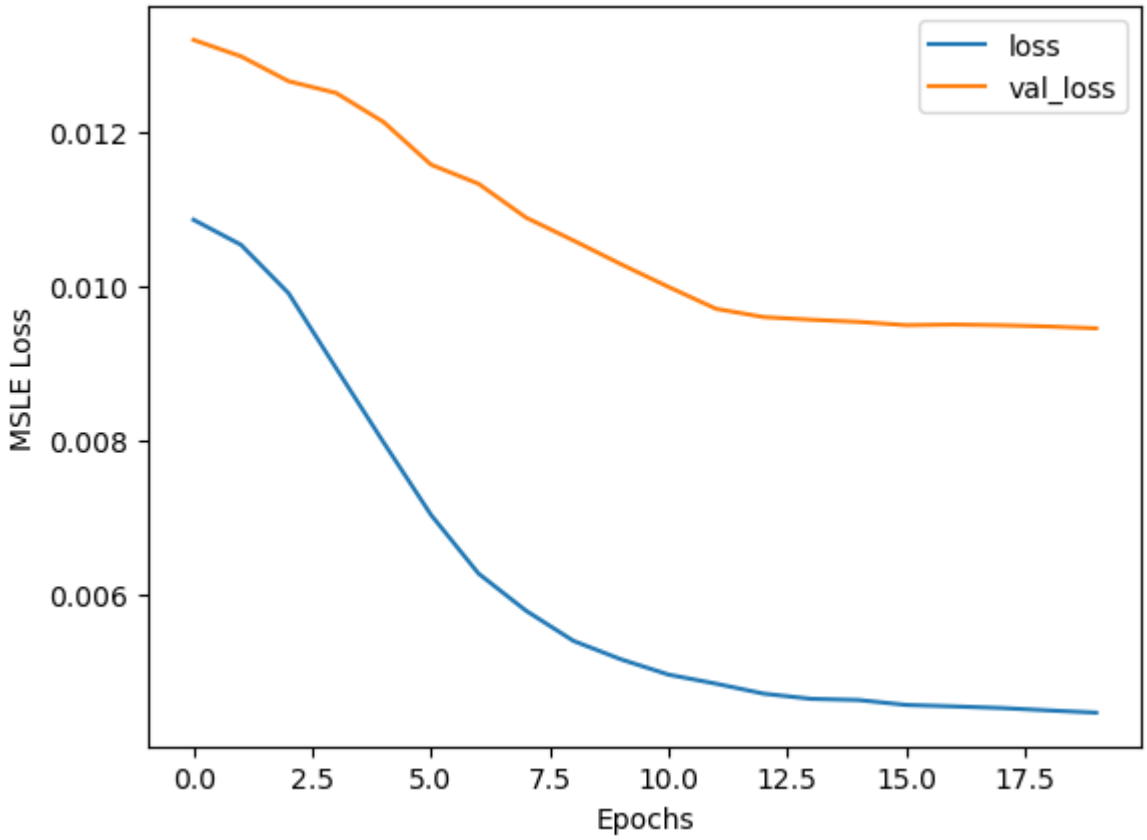
In [13]:
```
# min max scale the input data
min_max_scaler = MinMaxScaler(feature_range=(0, 1))
x_train_scaled = min_max_scaler.fit_transform(train_data.copy())
x_test_scaled = min_max_scaler.transform(x_test.copy())
```

In [14]:
```
# create a model by subclassing Model class in tensorflow
class AutoEncoder(Model):
```

```
  """
  Parameters
  ----------
  output_units: int
    Number of output units

  code_size: int
    Number of units in bottle neck
  """

  def __init__(self, output_units, code_size=8):
    super().__init__()
    self.encoder = Sequential([
      Dense(64, activation='relu'),
      Dropout(0.1),
      Dense(32, activation='relu'),
      Dropout(0.1),
      Dense(16, activation='relu'),
      Dropout(0.1),
      Dense(code_size, activation='relu')
    ])
    self.decoder = Sequential([
      Dense(16, activation='relu'),
      Dropout(0.1),
      Dense(32, activation='relu'),
      Dropout(0.1),
      Dense(64, activation='relu'),
      Dropout(0.1),
      Dense(output_units, activation='sigmoid')
    ])

  def call(self, inputs):
    encoded = self.encoder(inputs)
    decoded = self.decoder(encoded)
    return decoded

model = AutoEncoder(output_units=x_train_scaled.shape[1])
# configurations of model
model.compile(loss='msle', metrics=['mse'], optimizer='adam')

history = model.fit(
    x_train_scaled,
    x_train_scaled,
    epochs=20,
    batch_size=512,
    validation_data=(x_test_scaled, x_test_scaled)
)
```

```
Epoch 1/20
5/5 [==============================] - 1s 76ms/step - loss: 0.0109 - mse: 0.0247 - va
l_loss: 0.0132 - val_mse: 0.0309
Epoch 2/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0105 - mse: 0.0239 - va
l_loss: 0.0130 - val_mse: 0.0304
Epoch 3/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0099 - mse: 0.0225 - va
l_loss: 0.0127 - val_mse: 0.0296
Epoch 4/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0089 - mse: 0.0202 - va
l_loss: 0.0125 - val_mse: 0.0291
Epoch 5/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0080 - mse: 0.0179 - va
l_loss: 0.0121 - val_mse: 0.0282
Epoch 6/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0070 - mse: 0.0157 - va
l_loss: 0.0116 - val_mse: 0.0269
Epoch 7/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0063 - mse: 0.0140 - va
l_loss: 0.0113 - val_mse: 0.0263
Epoch 8/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0058 - mse: 0.0130 - va
l_loss: 0.0109 - val_mse: 0.0253
Epoch 9/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0054 - mse: 0.0121 - va
l_loss: 0.0106 - val_mse: 0.0246
Epoch 10/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0052 - mse: 0.0116 - va
l_loss: 0.0103 - val_mse: 0.0239
Epoch 11/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0050 - mse: 0.0111 - va
l_loss: 0.0100 - val_mse: 0.0233
Epoch 12/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0048 - mse: 0.0109 - va
l_loss: 0.0097 - val_mse: 0.0227
Epoch 13/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0047 - mse: 0.0106 - va
l_loss: 0.0096 - val_mse: 0.0225
Epoch 14/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0046 - mse: 0.0105 - va
l_loss: 0.0096 - val_mse: 0.0224
Epoch 15/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0046 - mse: 0.0104 - va
l_loss: 0.0095 - val_mse: 0.0224
Epoch 16/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0046 - mse: 0.0103 - va
l_loss: 0.0095 - val_mse: 0.0223
Epoch 17/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0046 - mse: 0.0102 - va
l_loss: 0.0095 - val_mse: 0.0223
Epoch 18/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0045 - mse: 0.0102 - va
l_loss: 0.0095 - val_mse: 0.0223
Epoch 19/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0045 - mse: 0.0101 - va
l_loss: 0.0095 - val_mse: 0.0223
Epoch 20/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0045 - mse: 0.0101 - va
l_loss: 0.0095 - val_mse: 0.0222
```

In [15]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('MSLE Loss')
plt.legend(['loss', 'val_loss'])
plt.show()
```



In [16]:
```
def find_threshold(model, x_train_scaled):
  reconstructions = model.predict(x_train_scaled)
  # provides losses of individual instances
  reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_scaled)
  # threshold for anomaly scores
  threshold = np.mean(reconstruction_errors.numpy()) \
      + np.std(reconstruction_errors.numpy())
  return threshold

def get_predictions(model, x_test_scaled, threshold):
  predictions = model.predict(x_test_scaled)
  # provides losses of individual instances
  errors = tf.keras.losses.msle(predictions, x_test_scaled)
  # 0 = anomaly, 1 = normal
  anomaly_mask = pd.Series(errors) > threshold
  preds = anomaly_mask.map(lambda x: 0.0 if x == True else 1.0)
  return preds

threshold = find_threshold(model, x_train_scaled)
print(f"Threshold: {threshold}")
predictions = get_predictions(model, x_test_scaled, threshold)
accuracy_score(predictions, y_test)
```

```
73/73 [==============================] - 0s 1ms/step
Threshold: 0.009750753415600777
32/32 [==============================] - 0s 1ms/step
0.938
```

Out[16]:

In [ ]: