```
# Exp 9: Anomaly detection using Autoencoder


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# Download the Dataset
PATH_TO_DATA = 'http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv'
data = pd.read_csv(PATH_TO_DATA, header=None)


print(data.shape)
```

```
(4998, 141)
```

```
data.head()
```

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

5 rows × 141 columns

```
Target = 140


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


from sklearn.model_selection import train_test_split

xTrain, xTest, yTrain, yTest = train_test_split(features, target, test_size=0.2, st


xTrain.shape
```

```
(3998, 140)
```

```
yTrain.shape
```

```
(3998,)
```

```
# Consider only those samples where the index is 1, i.e. normal samples
train_index = yTrain[yTrain == 1].index
trainData = xTrain.loc[train_index]
trainData.shape
```

```
(2335, 140)
```

```
# Preprocess the Data
from sklearn.preprocessing import MinMaxScaler
min_MaxScaler = MinMaxScaler(feature_range=(0, 1))
xTrainScaled = min_MaxScaler.fit_transform(trainData)
xTestScaled = min_MaxScaler.fit_transform(xTest)
```

```
xTrainScaled.shape
```

```
(2335, 140)
```

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
import keras
```

```
# Input shape, Required for first layer of deep neural network
inputShape = keras.Input(shape=(xTrainScaled.shape[1],)) # Shape of 140, Input neur
```

```
# Building the Encoder
x = Dense(64, activation='relu')(inputShape)
x = Dropout(0.1)(x)
x = Dense(32, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(16, activation='relu')(x)
x = Dropout(0.1)(x)
encoderLayer = Dense(xTrainScaled.shape[1], activation='sigmoid')(x)
```

```
# Building the Decoder
x = Dense(16, activation='relu')(encoderLayer)
x = Dropout(0.1)(x)
x = Dense(32, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.1)(x)
decoderLayer = Dense(xTrainScaled.shape[1], activation='sigmoid')(x)
```

```
# Defining the Autoencoder
autoencoder = keras.Model(inputShape, decoderLayer)
```

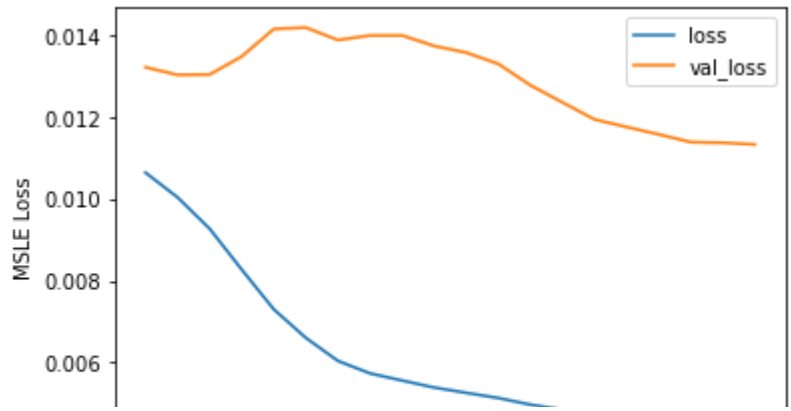
```
# Compiling the Model
autoencoder.compile(loss='msle', metrics=['mse'], optimizer='adam')

# Fit the Model
history = autoencoder.fit(xTrainScaled, xTrainScaled, epochs=20,
                          batch_size=512,
                          validation_data=(xTestScaled, xTestScaled))
```

```
Epoch 1/20
5/5 [==============================] - 1s 50ms/step - loss: 0.0106 - mse: 0.02
Epoch 2/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0100 - mse: 0.02
Epoch 3/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0093 - mse: 0.02
Epoch 4/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0083 - mse: 0.01
Epoch 5/20
5/5 [==============================] - 0s 18ms/step - loss: 0.0073 - mse: 0.01
Epoch 6/20
5/5 [==============================] - 0s 18ms/step - loss: 0.0066 - mse: 0.01
Epoch 7/20
5/5 [==============================] - 0s 14ms/step - loss: 0.0060 - mse: 0.01
Epoch 8/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0057 - mse: 0.01
Epoch 9/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0056 - mse: 0.01
Epoch 10/20
5/5 [==============================] - 0s 16ms/step - loss: 0.0054 - mse: 0.01
Epoch 11/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0053 - mse: 0.01
Epoch 12/20
5/5 [==============================] - 0s 17ms/step - loss: 0.0051 - mse: 0.01
Epoch 13/20
5/5 [==============================] - 0s 16ms/step - loss: 0.0050 - mse: 0.01
Epoch 14/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0048 - mse: 0.01
Epoch 15/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0047 - mse: 0.01
Epoch 16/20
5/5 [==============================] - 0s 16ms/step - loss: 0.0047 - mse: 0.01
Epoch 17/20
5/5 [==============================] - 0s 17ms/step - loss: 0.0046 - mse: 0.01
Epoch 18/20
5/5 [==============================] - 0s 19ms/step - loss: 0.0046 - mse: 0.01
Epoch 19/20
5/5 [==============================] - 0s 18ms/step - loss: 0.0045 - mse: 0.01
Epoch 20/20
5/5 [==============================] - 0s 16ms/step - loss: 0.0045 - mse: 0.01
```

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

```
<matplotlib.legend.Legend at 0x2ceec367d60>
```




```
def findThreshold(model, xTrainScaled):
    reconstructions = model.predict(xTrainScaled)
    reconstructionErrors = keras.losses.msle(reconstructions, xTrainScaled)

    threshold = np.mean(reconstructionErrors.numpy()) + np.std(reconstructionErrors
    return threshold


def getPredictions(model, xTestScaled, threshold):
    predictions = model.predict(xTestScaled)
    errors = keras.losses.msle(predictions, xTestScaled)

    anomalyMask = pd.Series(errors) > threshold

    preds = anomalyMask.map(lambda x: 0.0 if x == True else 1.0)
    return preds


threshold = findThreshold(autoencoder, xTrainScaled)
print(threshold)
```

```
0.009861804304962064
```

```
from sklearn.metrics import accuracy_score
preds = getPredictions(autoencoder, xTestScaled, threshold)
accuracy_score(preds, yTest)
```

```
0.855
```