```
# Importing Libraries
import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Downloading the dataset
!wget http://www.timeseriesclassification.com/Downloads/ECG5000.zip
```

```
--2022-11-08 02:57:04--  http://www.timeseriesclassification.com/Downloads/ECG5000.zip
Resolving www.timeseriesclassification.com (www.timeseriesclassification.com)... 109.123.71.232
Connecting to www.timeseriesclassification.com (www.timeseriesclassification.com)|109.123.71.232|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10614407 (10M) [application/zip]
Saving to: ‘ECG5000.zip’

ECG5000.zip         100%[===================>]  10.12M  13.0MB/s    in 0.8s

2022-11-08 02:57:05 (13.0 MB/s) - ‘ECG5000.zip’ saved [10614407/10614407]
```

```
# Unzipping the dataset
!unzip ECG5000.zip
```

```
Archive:  ECG5000.zip
  inflating: ECG5000.txt
  inflating: ECG5000_TEST.arff
  inflating: ECG5000_TEST.txt
  inflating: ECG5000_TRAIN.arff
  inflating: ECG5000_TRAIN.txt
  inflating: ECG5000_TEST.ts
  inflating: ECG5000_TRAIN.ts
```

```
# Concatenating the train and test file into a single file named 'ecg_final.txt'
!cat ECG5000_TRAIN.txt ECG5000_TEST.txt > ecg_final.txt
```

```
# Displaying the head of the file
!head ecg_final.txt
```

```
   1.0000000e+00  -1.1252183e-01  -2.8272038e+00  -3.7738969e+00  -4.3497511e+00  -4.3760410e+00  -3.4749863e+00  -2.1814082e+00  -1.8182865e+
   1.0000000e+00  -1.1008778e+00  -3.9968398e+00  -4.2858426e+00  -4.5065789e+00  -4.0223767e+00  -3.2343676e+00  -1.5661258e+00  -9.9225766e-
   1.0000000e+00  -5.6708802e-01  -2.5934502e+00  -3.8742297e+00  -4.5840949e+00  -4.1874487e+00  -3.1514617e+00  -1.7429402e+00  -1.4906585e+
   1.0000000e+00   4.9047253e-01  -1.9144071e+00  -3.6163638e+00  -4.3188235e+00  -4.2680158e+00  -3.8811104e+00  -2.9932802e+00  -1.6711314e+
   1.0000000e+00   8.0023202e-01  -8.7425189e-01  -2.3847613e+00  -3.9732924e+00  -4.3382241e+00  -3.8024222e+00  -2.5345096e+00  -1.7834233e+
   1.0000000e+00  -1.5076736e+00  -3.5745500e+00  -4.4780109e+00  -4.4082752e+00  -3.3212415e+00  -2.1051715e+00  -1.4810482e+00  -1.3013622e+
   1.0000000e+00  -2.9716100e-01  -2.7666349e+00  -4.1021848e+00  -4.5896691e+00  -4.2193569e+00  -3.6504434e+00  -2.3005176e+00  -1.2939171e+
   1.0000000e+00   4.4676853e-01  -1.5073974e+00  -3.1874679e+00  -4.5074621e+00  -4.6042007e+00  -3.6361150e+00  -2.3116038e+00  -1.5977275e+
   1.0000000e+00   8.7630577e-02  -1.7534903e+00  -3.3044731e+00  -4.7046566e+00  -4.6864151e+00  -3.6118167e+00  -2.2672676e+00  -1.5708930e+
   1.0000000e+00  -8.3228111e-01  -1.7003675e+00  -2.2573013e+00  -2.8536712e+00  -2.8533008e+00  -2.7014866e+00  -2.2857261e+00  -1.5555120e+
```

```
# Importing the finla file in pandas dataframe
df = pd.read_csv('ecg_final.txt', sep = '  ', header = None)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/util/_decorators.py:311: ParserWarning: Falling back to the 'python' engine because the 'c' engi
  return func(*args, **kwargs)
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	131	132	133	134
0	1.0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818286	-1.250522	...	0.160348	0.792168	0.933541	0.796958
1	1.0	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258	-0.754680	...	0.560327	0.538356	0.656881	0.787490
2	1.0	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659	-1.183580	...	1.284825	0.886073	0.531452	0.311377
3	1.0	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131	-1.333884	...	0.491173	0.350816	0.499111	0.600345
4	1.0	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.594450	...	0.966606	1.148884	0.958434	1.059025

5 rows × 141 columns

```
df.shape
```

```
(5000, 141)
```

```
df.columns
```

```
Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
            ...
            131, 132, 133, 134, 135, 136, 137, 138, 139, 140],
           dtype='int64', length=141)
```

```
# Adding prefix to column names so that we can easily reference them
# Original file did not contain column names so pandas creates numeric column names automatically that cannot be referenced easily
df = df.add_prefix('c')
```

```
df.columns
```

```
Index(['c0', 'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8', 'c9',
       ...
       'c131', 'c132', 'c133', 'c134', 'c135', 'c136', 'c137', 'c138', 'c139',
       'c140'],
      dtype='object', length=141)
```

```
# Counting the data points of diffrent labels
df['c0'].value_counts()
```

```
1.0    2919
2.0    1767
4.0     194
3.0      96
5.0      24
Name: c0, dtype: int64
```

```
df.describe()
```

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	...	
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	...	5000
mean	1.527400	-0.262476	-1.649511	-2.492211	-3.119443	-3.167438	-2.866308	-2.273126	-1.798127	-1.410124	...	0
std	0.760372	1.152369	1.445493	1.386409	1.302802	1.104382	0.906133	0.731627	0.623100	0.637149	...	1
min	1.000000	-6.729499	-7.090374	-5.132459	-5.363241	-5.375715	-5.330194	-4.782240	-4.311288	-4.071361	...	-3
25%	1.000000	-1.004511	-2.701576	-3.668096	-4.227247	-4.007470	-3.480479	-2.779941	-2.165851	-1.774124	...	-0
50%	1.000000	-0.297541	-1.661892	-2.585677	-3.387934	-3.468718	-2.947061	-2.285578	-1.750157	-1.422570	...	0
75%	2.000000	0.500061	-0.677290	-1.513964	-2.235369	-2.530967	-2.398813	-1.823494	-1.484923	-1.063708	...	1
max	5.000000	4.966414	3.479689	2.660597	1.899798	2.147015	1.614375	1.868728	1.804251	1.683730	...	2

8 rows × 141 columns

```
# splitting into train test data
train_data, test_data, train_labels, test_labels = train_test_split(df.values, df.values[:, 0:1], test_size = 0.2, random_state = 111)
```

```
# Initializing a MinMax Scaler
scaler = MinMaxScaler()

# Fitting the train data to the scaler
data_scaled = scaler.fit(train_data)
```

```
# Scaling dataset according to weights of train data
train_data_scaled = data_scaled.transform(train_data)
test_data_scaled = data_scaled.transform(test_data)
```

```
train_data.shape
```

```
(4000, 141)
```

```
# Making pandas dataframe for the normal and anomaly train data points
normal_train_data = pd.DataFrame(train_data_scaled).add_prefix('c').query('c0 == 0').values[:, 1:]
anomaly_train_data = pd.DataFrame(train_data_scaled).add_prefix('c').query('c0 > 0').values[:, 1:]
```

```
anomaly_train_data
```

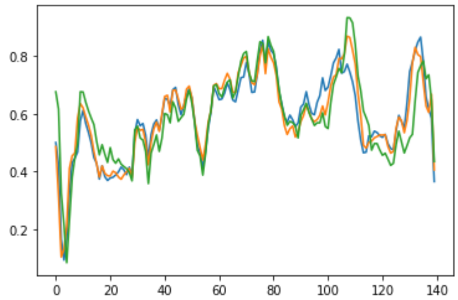
```
array([[0.54603684, 0.52609574, 0.35215565, ..., 0.32938752, 0.41559349,
        0.4550684 ],
       [0.39336652, 0.39486685, 0.27028019, ..., 0.37738131, 0.4863785 ,
        0.45174016],
       [0.66165586, 0.75136705, 0.70959038, ..., 0.15203245, 0.2072104 ,
        0.30963706],
       ...,
```

```
       [0.58122047, 0.57240472, 0.39287094, ..., 0.32309346, 0.41186439,
        0.40845571],
       [0.70698484, 0.7982501 , 0.77487296, ..., 0.23053824, 0.31421167,
        0.37774737],
       [0.69314707, 0.79831145, 0.82004413, ..., 0.68561341, 0.61110713,
        0.53512758]])
```

```
# Making pandas dataframe for the normal and anomaly test data points
normal_test_data = pd.DataFrame(test_data_scaled).add_prefix('c').query('c0 == 0').values[:, 1:]
anomaly_test_data = pd.DataFrame(test_data_scaled).add_prefix('c').query('c0 > 0').values[:, 1:]
```

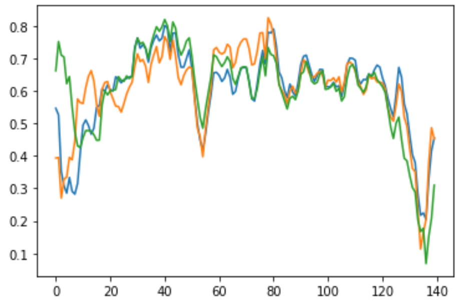
```
# plotting the first three normal data points
plt.plot(normal_train_data[0])
plt.plot(normal_train_data[1])
plt.plot(normal_train_data[2])
```

```
[<matplotlib.lines.Line2D at 0x7f82cb742110>]
```




```
# plotting the first three anomaly data points
plt.plot(anomaly_train_data[0])
plt.plot(anomaly_train_data[1])
plt.plot(anomaly_train_data[2])
```

```
[<matplotlib.lines.Line2D at 0x7f82cb69a710>]
```




```
class Autoencoder(Model):
  def __init__(self):
    super(Autoencoder, self).__init__()
    self.encoder = Sequential([
                            Dense(64, activation='relu'),
                            Dense(32, activation='relu'),
                            Dense(16, activation='relu'),
                            Dense(8, activation='relu')
    ])

    self.decoder = Sequential([
                           Dense(16, activation='relu'),
                           Dense(32, activation='relu'),
                           Dense(64, activation='relu'),
                           Dense(140, activation='sigmoid')
    ])

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

```
# Instantiating the Autoencoder
model = Autoencoder()

# creating an early_stopping
early_stopping = EarlyStopping(monitor='val_loss',
                               patience = 2,
                               mode = 'min')

# Compiling the model
model.compile(optimizer = 'adam',
              loss = 'mae')
```

```
# Training the model
history = model.fit(normal_train_data,normal_train_data,
                    epochs = 50,
                    batch_size = 120,
                    validation_data = (train_data_scaled[:,1:], train_data_scaled[:,1:]),
                    shuffle = True,
                    callbacks = [early_stopping])
```

```
Epoch 1/50
20/20 [==============================] - 5s 29ms/step - loss: 0.1327 - val_loss: 0.1198
Epoch 2/50
20/20 [==============================] - 0s 11ms/step - loss: 0.0866 - val_loss: 0.0820
Epoch 3/50
20/20 [==============================] - 0s 14ms/step - loss: 0.0532 - val_loss: 0.0775
Epoch 4/50
20/20 [==============================] - 0s 12ms/step - loss: 0.0486 - val_loss: 0.0764
Epoch 5/50
20/20 [==============================] - 0s 13ms/step - loss: 0.0481 - val_loss: 0.0759
Epoch 6/50
20/20 [==============================] - 0s 13ms/step - loss: 0.0478 - val_loss: 0.0750
Epoch 7/50
20/20 [==============================] - 0s 15ms/step - loss: 0.0473 - val_loss: 0.0736
Epoch 8/50
20/20 [==============================] - 0s 12ms/step - loss: 0.0459 - val_loss: 0.0708
Epoch 9/50
20/20 [==============================] - 0s 15ms/step - loss: 0.0422 - val_loss: 0.0652
Epoch 10/50
20/20 [==============================] - 0s 13ms/step - loss: 0.0388 - val_loss: 0.0626
Epoch 11/50
20/20 [==============================] - 0s 16ms/step - loss: 0.0372 - val_loss: 0.0619
Epoch 12/50
20/20 [==============================] - 0s 15ms/step - loss: 0.0367 - val_loss: 0.0617
Epoch 13/50
20/20 [==============================] - 0s 14ms/step - loss: 0.0363 - val_loss: 0.0611
Epoch 14/50
20/20 [==============================] - 0s 13ms/step - loss: 0.0361 - val_loss: 0.0612
Epoch 15/50
20/20 [==============================] - 0s 14ms/step - loss: 0.0358 - val_loss: 0.0611
```

```
# predictions for normal test data points
encoder_out = model.encoder(normal_test_data).numpy()
decoder_out = model.decoder(encoder_out).numpy()
```

```
encoder_out.shape
```

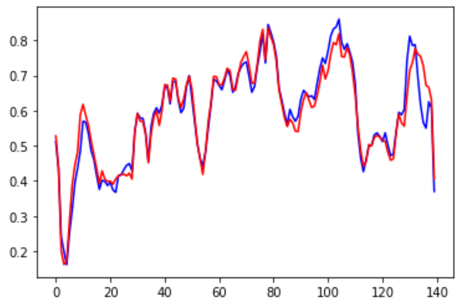
```
(563, 8)
```

```
decoder_out.shape
```

```
(563, 140)
```

```
# plotting normal test data point and its predictiction by the autoencoder
plt.plot(normal_test_data[0], 'b')
plt.plot(decoder_out[0], 'r')
```

```
[<matplotlib.lines.Line2D at 0x7f82cc4ebd50>]
```




```
# predictions for anomaly test data points
encoder_out_a = model.encoder(anomaly_test_data).numpy()
decoder_out_a = model.decoder(encoder_out_a).numpy()
```

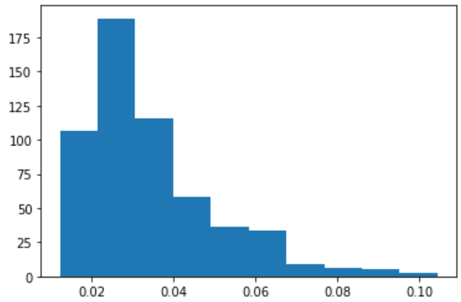
```
# plotting anomaly test data point and its predictiction by the autoencoder
plt.plot(anomaly_test_data[0], 'b')
plt.plot(decoder_out_a[0], 'r')
```

```
[<matplotlib.lines.Line2D at 0x7f82cbb31290>]
```


```
# reconstruction loss for normal test data
reconstructions = model.predict(normal_test_data)
train_loss = tf.keras.losses.mae(reconstructions, normal_test_data)

# Plotting histogram for recontruction loss for normal test data
plt.hist(train_loss, bins = 10)
```

```
18/18 [==============================] - 0s 2ms/step
(array([107., 189., 116.,  58.,  36.,  34.,   9.,   6.,   5.,   3.]),
 array([0.01225547, 0.02146618, 0.03067689, 0.0398876 , 0.04909831,
        0.05830902, 0.06751973, 0.07673044, 0.08594115, 0.09515186,
        0.10436257]),
 <a list of 10 Patch objects>)
```




```
np.mean(train_loss)
```

```
0.034376534790579 69
```

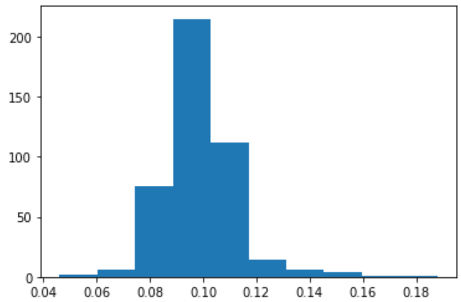
```
np.std(train_loss)
```

```
0.016068337924014516
```

```
# reconstruction loss for anomaly test data
reconstructions_a = model.predict(anomaly_test_data)
train_loss_a = tf.keras.losses.mae(reconstructions_a, anomaly_test_data)

# Plotting histogram for recontruction loss for anomaly test data
plt.hist(train_loss_a, bins = 10)
```

```
14/14 [==============================] - 0s 2ms/step
(array([  2.,   6.,  76., 215., 112.,  14.,   6.,   4.,   1.,   1.]),
 array([0.04616807, 0.06034158, 0.07451508, 0.08868859, 0.1028621 ,
        0.11703561, 0.13120911, 0.14538262, 0.15955613, 0.17372964,
        0.18790314]),
 <a list of 10 Patch objects>)
```




```
np.mean(train_loss_a)
```

```
0.09852357621271686
```

```
np.std(train_loss_a)
```

```
0.013950600324532561
```

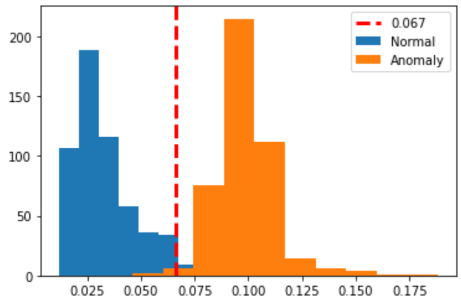
```
# setting threshold
threshold = np.mean(train_loss) + 2*np.std(train_loss)
```

```
threshold
```

```
0.06651321063860872
```

```
# Plotting the normal and anomaly losses with the threshold
```

```
plt.hist(train_loss, bins = 10, label = 'Normal')
plt.hist(train_loss_a, bins = 10, label = 'Anomaly')
plt.axvline(threshold, color='r', linewidth = 3, linestyle = 'dashed', label = '{:0.3f}'.format(threshold))
plt.legend(loc = 'upper right')
plt.show()
```




```
# Number of correct predictions for Normal test data
preds = tf.math.less(train_loss, threshold)
```

```
tf.math.count_nonzero(preds)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=537>
```

```
# Number of correct predictions for Anomaly test data
preds_a = tf.math.greater(train_loss_a, threshold)
```

```
tf.math.count_nonzero(preds_a)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=433>
```

```
preds_a.shape
```

```
TensorShape([437])
```