In [1]:
```
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 [2]:
```
path='''http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv'''
data=pd.read_csv(path,header=None)
print(data.shape)
data.head()
```

```
(4998, 141)
```

Out[2]:

	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 [3]:
```
data.tail()
```

Out[3]:

	0	1	2	3	4	5	6	7	8
4993	0.608558	-0.335651	-0.990948	-1.784153	-2.626145	-2.957065	-2.931897	-2.664816	-2.090137
4994	-2.060402	-2.860116	-3.405074	-3.748719	-3.513561	-3.006545	-2.234850	-1.593270	-1.075279
4995	-1.122969	-2.252925	-2.867628	-3.358605	-3.167849	-2.638360	-1.664162	-0.935655	-0.866953
4996	-0.547705	-1.889545	-2.839779	-3.457912	-3.929149	-3.966026	-3.492560	-2.695270	-1.849691
4997	-1.351779	-2.209006	-2.520225	-3.061475	-3.065141	-3.030739	-2.622720	-2.044092	-1.295874

5 rows × 141 columns

In [4]:
```
#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,random_st
```

In [5]:
```
x_train.shape
```

Out[5]:
```
(3998, 140)
```

In [6]:
```
y_test.shape
```

Out[6]:
```
(1000,)
```

In [7]:
```
x_test.shape
```

Out[7]:
```
(1000, 140)
```

In [8]:
```
target.value_counts()
```

Out[8]:
```
1.0    2919
0.0    2079
Name: 140, dtype: int64
```

In [9]:
```
#use case is novelty detection so use only the normal for training
train_index=y_train[y_train==1].index
train_data=x_train.loc[train_index]
```

In [10]:
```
min_max_scaler=MinMaxScaler()
x_train_scaled=min_max_scaler.fit_transform(train_data.copy())
x_test_scaled=min_max_scaler.transform(x_test.copy())
```

In [11]:
```
x_train.describe()
```

Out[11]:

	0	1	2	3	4	5	6	3
count	3998.000000	3998.000000	3998.000000	3998.000000	3998.000000	3998.000000	3998.000000	
mean	-0.243013	-1.627465	-2.479011	-3.111913	-3.169065	-2.872239	-2.275887	
std	1.156946	1.447211	1.385123	1.305425	1.113079	0.913680	0.735427	
min	-6.729499	-7.090374	-5.132459	-5.324706	-5.375715	-5.217053	-4.512132	
25%	-0.988147	-2.679589	-3.650456	-4.222552	-4.017797	-3.494802	-2.786853	
50%	-0.282004	-1.640674	-2.578471	-3.387353	-3.482953	-2.952360	-2.289780	
75%	0.532473	-0.648556	-1.504884	-2.234290	-2.525682	-2.409076	-1.827482	
max	4.966414	3.479689	2.660597	1.899798	2.147015	1.614375	1.868728	

8 rows × 140 columns

In [12]:
```
pd.DataFrame(x_train_scaled).describe()
```

Out[12]:

	0	1	2	3	4	5	6	2
count	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	
mean	0.544746	0.469984	0.274483	0.259759	0.265531	0.347222	0.389287	
std	0.108186	0.136394	0.148195	0.134715	0.099682	0.127931	0.128073	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.470612	0.373531	0.166064	0.173326	0.207592	0.259683	0.298745	
50%	0.534498	0.447919	0.244325	0.222024	0.254606	0.334196	0.405625	
75%	0.616405	0.546008	0.347583	0.310697	0.305755	0.426743	0.478648	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 140 columns

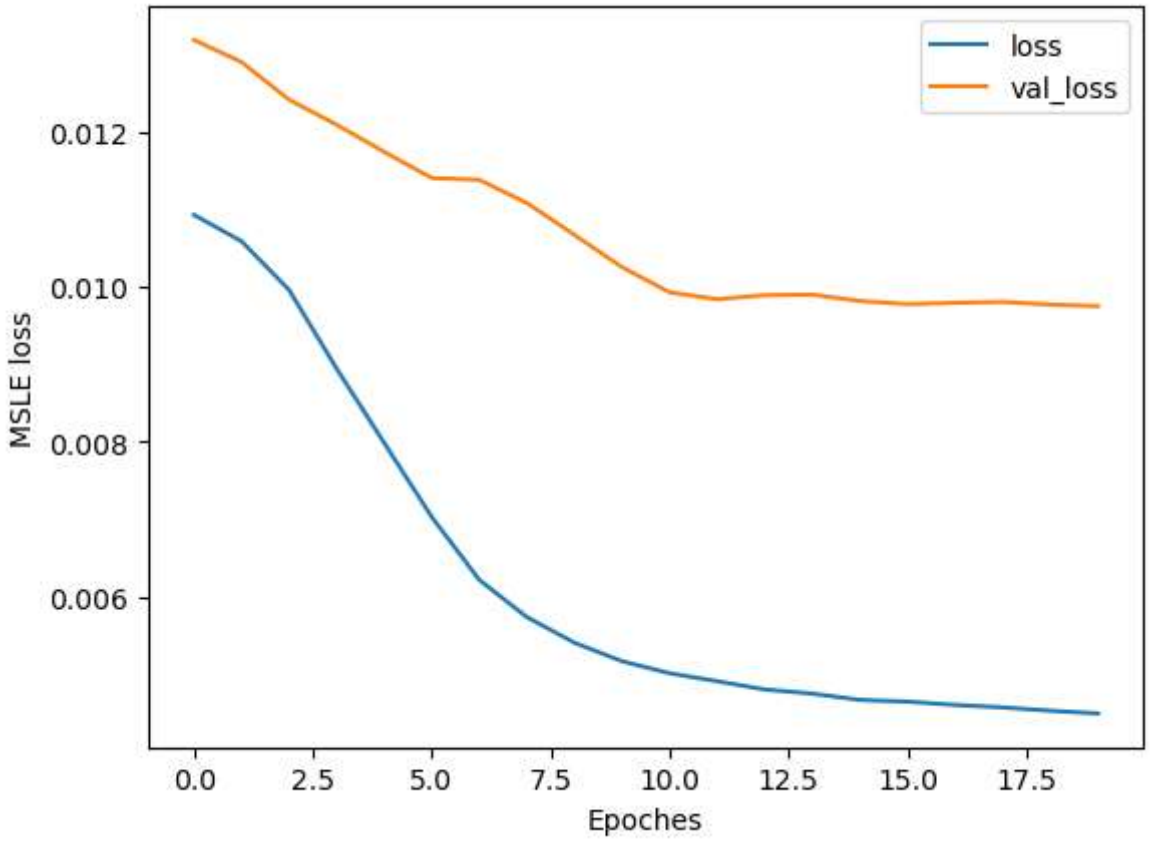
In [13]:
```
#create a model by subclassing Model class in tensorflow
class AutoEncoder(Model):
  """
  Parameters

  _____________
  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,activa
    self.decoder=Sequential ([Dense(16,activation='relu'),Dropout(0.1),Dense(32,activa
  def call(self,inputs):
   encoded=self.encoder(inputs)
   decoded=self.decoder(encoded)
   return decoded
```

In [14]:
```
model=AutoEncoder(output_units=x_train_scaled.shape[1])
```

In [15]:
```
#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_da
```

```
Epoch 1/20
5/5 [==============================] - 3s 132ms/step - loss: 0.0109 - mse: 0.0246 - v
al_loss: 0.0132 - val_mse: 0.0306
Epoch 2/20
5/5 [==============================] - 0s 29ms/step - loss: 0.0106 - mse: 0.0238 - va
l_loss: 0.0129 - val_mse: 0.0299
Epoch 3/20
5/5 [==============================] - 0s 32ms/step - loss: 0.0100 - mse: 0.0224 - va
l_loss: 0.0124 - val_mse: 0.0289
Epoch 4/20
5/5 [==============================] - 0s 27ms/step - loss: 0.0089 - mse: 0.0202 - va
l_loss: 0.0121 - val_mse: 0.0281
Epoch 5/20
5/5 [==============================] - 0s 25ms/step - loss: 0.0080 - mse: 0.0180 - va
l_loss: 0.0117 - val_mse: 0.0272
Epoch 6/20
5/5 [==============================] - 0s 25ms/step - loss: 0.0070 - mse: 0.0158 - va
l_loss: 0.0114 - val_mse: 0.0264
Epoch 7/20
5/5 [==============================] - 0s 87ms/step - loss: 0.0062 - mse: 0.0139 - va
l_loss: 0.0114 - val_mse: 0.0263
Epoch 8/20
5/5 [==============================] - 0s 22ms/step - loss: 0.0057 - mse: 0.0128 - va
l_loss: 0.0111 - val_mse: 0.0256
Epoch 9/20
5/5 [==============================] - 0s 28ms/step - loss: 0.0054 - mse: 0.0120 - va
l_loss: 0.0107 - val_mse: 0.0247
Epoch 10/20
5/5 [==============================] - 0s 30ms/step - loss: 0.0052 - mse: 0.0115 - va
l_loss: 0.0102 - val_mse: 0.0238
Epoch 11/20
5/5 [==============================] - 0s 40ms/step - loss: 0.0050 - mse: 0.0112 - va
l_loss: 0.0099 - val_mse: 0.0231
Epoch 12/20
5/5 [==============================] - 0s 34ms/step - loss: 0.0049 - mse: 0.0109 - va
l_loss: 0.0098 - val_mse: 0.0229
Epoch 13/20
5/5 [==============================] - 0s 29ms/step - loss: 0.0048 - mse: 0.0107 - va
l_loss: 0.0099 - val_mse: 0.0230
Epoch 14/20
5/5 [==============================] - 0s 36ms/step - loss: 0.0048 - mse: 0.0106 - va
l_loss: 0.0099 - val_mse: 0.0230
Epoch 15/20
5/5 [==============================] - 0s 43ms/step - loss: 0.0047 - mse: 0.0104 - va
l_loss: 0.0098 - val_mse: 0.0228
Epoch 16/20
5/5 [==============================] - 0s 28ms/step - loss: 0.0046 - mse: 0.0104 - va
l_loss: 0.0098 - val_mse: 0.0228
Epoch 17/20
5/5 [==============================] - 0s 35ms/step - loss: 0.0046 - mse: 0.0103 - va
l_loss: 0.0098 - val_mse: 0.0228
Epoch 18/20
5/5 [==============================] - 0s 37ms/step - loss: 0.0046 - mse: 0.0102 - va
l_loss: 0.0098 - val_mse: 0.0228
Epoch 19/20
5/5 [==============================] - 0s 28ms/step - loss: 0.0045 - mse: 0.0101 - va
l_loss: 0.0098 - val_mse: 0.0227
Epoch 20/20
5/5 [==============================] - 0s 30ms/step - loss: 0.0045 - mse: 0.0100 - va
l_loss: 0.0097 - val_mse: 0.0227
```

In [16]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epoches')
plt.ylabel('MSLE loss')
plt.legend(['loss','val_loss'])
plt.show()
```



In [ ]:
```

```