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
PATH_TO_DATA = 'http://storage.googleapis.com/download.tensorflow.org/data/
data = pd.read_csv(PATH_TO_DATA, header=None)
data.head()
```

Out[1]:

	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	-1.2
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258	-0.7
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659	-1.1
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131	-1.3
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.5

5 rows × 141 columns

In [2]:
```
data.shape
```

Out[2]:
```
(4998, 141)
```

In [3]:
```
features = data.drop(140, axis=1)
target = data[140]
x_train, x_test, y_train, y_test = train_test_split(
features, target, test_size=0.2, stratify=target
)
train_index = y_train[y_train == 1].index
train_data = x_train.loc[train_index]
```

In [4]:
```
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 [7]:
```
class AutoEncoder(Model):
    def init (self, output_units, ldim=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(ldim, 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
```

In [13]:
```
model = AutoEncoder(output_units = x_train_scaled.shape[1])
model.compile(loss='msle', metrics=['mse'], optimizer='adam')
epochs = 20
history = model.fit(
    x_train_scaled,
    x_train_scaled,
    epochs=epochs,
    batch_size=512,
    validation_data=(x_test_scaled, x_test_scaled)
)
```

```
-------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
Cell In[13], line 1
----> 1 model = AutoEncoder(output_units = x_train_scaled.shape[1])
      2 model.compile(loss='msle', metrics=['mse'], optimizer='adam')
      3 epochs = 20

File ~\anaconda3\lib\site-packages\tensorflow\python\trackable\base.py:20
4, in no_automatic_dependency_tracking.<locals>._method_wrapper(self, *ar
gs, **kwargs)
    202 self._self_setattr_tracking = False  # pylint: disable=protected-
access
    203 try:
--> 204   result = method(self, *args, **kwargs)
    205 finally:
    206   self._self_setattr_tracking = previous_value  # pylint: disable
=protected-access

File ~\anaconda3\lib\site-packages\keras\src\utils\traceback_utils.py:70,
in filter_traceback.<locals>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File ~\anaconda3\lib\site-packages\keras\src\utils\generic_utils.py:514,
in validate_kwargs(kwargs, allowed_kwargs, error_message)
    512 for kwarg in kwargs:
    513     if kwarg not in allowed_kwargs:
--> 514         raise TypeError(error_message, kwarg)

TypeError: ('Keyword argument not understood:', 'output_units')
```

In [14]:
```
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()
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
Cell In[14], line 1
----> 1 plt.plot(history.history['loss'])
      2 plt.plot(history.history['val_loss'])
      3 plt.xlabel('Epochs')

NameError: name 'history' is not defined
```

In [16]:
```
def find_threshold(model, x_train_scaled):
    reconstructions = model.predict(x_train_scaled)
    reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_s
    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)
    errors = tf.keras.losses.msle(predictions, x_test_scaled)
    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}")
```

In [17]:
```
predictions = get_predictions(model, x_test_scaled, threshold)
accuracy_score(predictions, y_test)
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
Cell In[17], line 1
----> 1 predictions = get_predictions(model, x_test_scaled, threshold)
      2 accuracy_score(predictions, y_test)

NameError: name 'model' is not defined
```

In [ ]: