```
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
from tensorflow.keras.losses import MeanSquaredError


# load the dataset
data = pd.read_csv('/content/MY_Final_Dataset_csv.csv')
data.head()
```

	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	Bwd Pkt Len Max	Bwd Pkt Len Min	
0	3268	1	3	34	690	34	34	34.000000	0.000000	334	34	2
1	3203	1	3	30	422	30	30	30.000000	0.000000	248	30	1
2	3737838	15	15	1067	32637	517	0	71.133333	171.958411	5200	0	21
3	283085	16	18	1067	39137	517	0	66.687500	167.076716	6500	0	21
4	109993357	27	1	862	38	76	28	31.925926	12.848005	38	38	

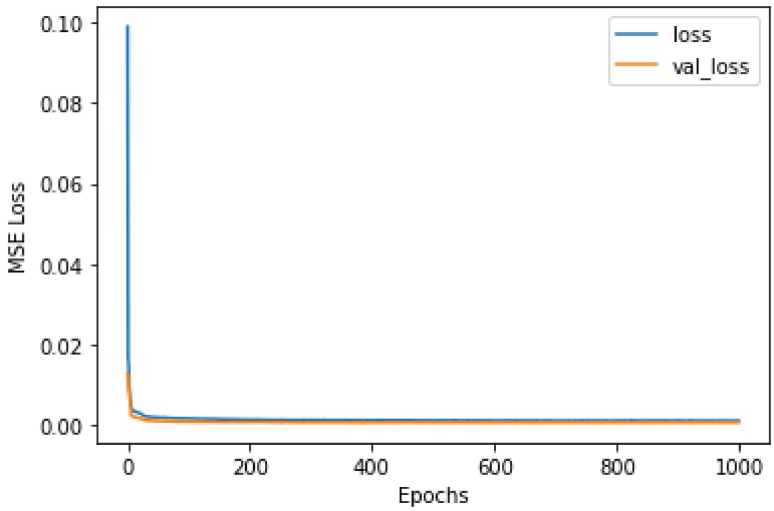
```
features = data.drop('Label', axis=1)
target = data['Label']

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

# use case is novelty detection so use only the normal data
# for training
train_index = y_train[y_train == 0].index
train_data = x_train.loc[train_index]

# 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())


class AutoEncoder(Model):
  def __init__(self, output_units, code_size=16):
    super().__init__()
    self.encoder = Sequential([
      Dense(66, 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(66, 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='mse', metrics=['accuracy'], optimizer='adam')

history = model.fit(
    x_train_scaled,
    x_train_scaled,
    epochs=1000,
    batch_size=64,
    validation_data=(x_test_scaled, x_test_scaled)
)
```

```
Epoch 251/1000
94/94 [==============================] - 1s 6ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 252/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.88
Epoch 253/1000
94/94 [==============================] - 1s 6ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 254/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 255/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 256/1000
```

```
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.894
Epoch 257/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.88
Epoch 258/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 259/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.894
Epoch 260/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 261/1000
94/94 [==============================] - 1s 6ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 262/1000
94/94 [==============================] - 1s 6ms/step - loss: 0.0014 - accuracy: 0.88
Epoch 263/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 264/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 265/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 266/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 267/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.894
Epoch 268/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0013 - accuracy: 0.89
Epoch 269/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 270/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 271/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.894
Epoch 272/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 273/1000
94/94 [==============================] - 1s 6ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 274/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 275/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 276/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 277/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0014 - accuracy: 0.89
Epoch 278/1000
94/94 [==============================] - 1s 7ms/step - loss: 0.0013 - accuracy: 0.894
Epoch 279/1000
```

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




```
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)

  anomaly_mask = pd.Series(errors) > threshold
  preds = anomaly_mask.map(lambda x: 1.0 if x == True else 0.0)
  return preds

threshold = find_threshold(model, x_train_scaled)
print(f"Threshold: {threshold}")
```

```
Threshold: 0.0025255849341829207
```

```
predictions = get_predictions(model, x_test_scaled, threshold)
accuracy_score(y_test,predictions)
```

```
0.5355632449058054
```

7m 40s completed at 2:46 PM