```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model


datafirst = pd.read_csv('flight.csv', header=None)
raw=datafirst.values
dataframe = pd.read_csv('flight.csv', header=None)
raw_data = dataframe.values
dataframe.head()
```

	0	1	2	3	4	5	6	7	8
0	0	00M	Thigpen	Bay Springs	MS	USA	31.953765	-89.234505	1
1	1	00R	Livingston Municipal	Livingston	TX	USA	30.685861	-95.017928	1
2	2	00V	Meadow Lake	Colorado Springs	CO	USA	38.945749	-104.569893	1
3	3	01G	Perry-Warsaw	Perry	NY	USA	42.741347	-78.052081	0
4	4	01J	Hilliard Airpark	Hilliard	FL	USA	30.688012	-81.905944	1

```
dataframe.drop(0, axis=1, inplace=True)

raw_data = dataframe.values
dataframe.head()
```

	1	2	3	4	5	6	7	8
0	00M	Thigpen	Bay Springs	MS	USA	31.953765	-89.234505	1
1	00R	Livingston Municipal	Livingston	TX	USA	30.685861	-95.017928	1
2	00V	Meadow Lake	Colorado Springs	CO	USA	38.945749	-104.569893	1
3	01G	Perry-Warsaw	Perry	NY	USA	42.741347	-78.052081	0
4	01J	Hilliard Airpark	Hilliard	FL	USA	30.688012	-81.905944	1

```
dataframe.drop(1, axis=1, inplace=True)

dataframe.drop(2, axis=1, inplace=True)
dataframe.drop(3, axis=1, inplace=True)

raw_data = dataframe.values
dataframe.head()
```

	4	5	6	7	8
0	MS	USA	31.953765	-89.234505	1
1	TX	USA	30.685861	-95.017928	1
2	CO	USA	38.945749	-104.569893	1
3	NY	USA	42.741347	-78.052081	0
4	FL	USA	30.688012	-81.905944	1

```
def dummies(x,df):
   temp = pd.get_dummies(df[x], drop_first=True)
   df = pd.concat([df, temp], axis=1)
   df.drop([x], axis=1, inplace=True)
   return df
```

```
dataframe=dummies(4, dataframe)
```

```
dataframe
```

	5	6	7	8	AL	AR	AS	AZ	CA	CO	...	TN	TX	UT	VA	VI	VT	WA	WI	WV	WY
0	USA	31.953765	-89.234505	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	USA	30.685861	-95.017928	1	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	0
2	USA	38.945749	-104.569893	1	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
3	USA	42.741347	-78.052081	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	USA	30.688012	-81.905944	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3371	USA	36.280024	-80.786069	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3372	USA	40.706449	-76.373147	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3373	USA	28.228065	-82.155916	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3374	USA	35.083227	-108.791777	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3375	USA	39.944458	-81.892105	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

3376 rows × 59 columns

```
dataframe=dummies(5, dataframe)
```

```
dataframe
```

	6	7	8	AL	AR	AS	AZ	CA	CO	CQ	...	VI	VT	WA	WI	WV	WY	N Mariana Islands	Palau	Thailand	USA
0	31.953765	-89.234505	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
1	30.685861	-95.017928	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2	38.945749	-104.569893	1	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	1
3	42.741347	-78.052081	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
4	30.688012	-81.905944	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3371	36.280024	-80.786069	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
3372	40.706449	-76.373147	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
3373	28.228065	-82.155916	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
3374	35.083227	-108.791777	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
3375	39.944458	-81.892105	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

3376 rows × 62 columns

```
dataframe.drop(8, axis=1, inplace=True)
```

```
dataframe
```

	6	7	AL	AR	AS	AZ	CA	CO	CQ	CT	...	VI	VT	WA	WI	WV	WY	N Mariana Islands	Palau	Thailand	USA
0	31.953765	-89.234505	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
1	30.685861	-95.017928	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2	38.945749	-104.569893	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	1
3	42.741347	-78.052081	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
4	30.688012	-81.905944	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3371	36.280024	-80.786069	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
3372	40.706449	-76.373147	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

```
raw_data = dataframe.values
dataframe.head()
```

	6	7	AL	AR	AS	AZ	CA	CO	CQ	CT	...	VI	VT	WA	WI	WV	WY	N Mariana Islands	Palau	Thailand	USA
0	31.953765	-89.234505	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
1	30.685861	-95.017928	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2	38.945749	-104.569893	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	1
3	42.741347	-78.052081	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
4	30.688012	-81.905944	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

5 rows × 61 columns

```
raw_data
```

```
array([[  31.95376472,  -89.23450472,    0.        ,    0.        ,
             0.        ,    1.        ],
       [  30.68586111,  -95.01792778,    0.        ,    0.        ,
             0.        ,    1.        ],
       [  38.94574889, -104.5698933 ,    0.        ,    0.        ,
             0.        ,    1.        ],
       ...,
       [  28.22806472,  -82.15591639,    0.        ,    0.        ,
             0.        ,    1.        ],
       [  35.08322694, -108.7917769 ,    0.        ,    0.        ,
             0.        ,    1.        ],
       [  39.94445833,  -81.89210528,    0.        ,    0.        ,
             0.        ,    1.        ]])
```

```
cat_col=[col for col in dataframe.columns if dataframe[col].dtype =='object']
print('Categorical columns :', cat_col)
num_col=[col for col in dataframe.columns if dataframe[col].dtype !='object']
print('Numerical columns :', num_col)
```

```
Categorical columns : []
Numerical columns : [6, 7, 'AL', 'AR', 'AS', 'AZ', 'CA', 'CO', 'CQ', 'CT', 'DC', 'DE', 'FL', 'GA', 'GU', 'HI', 'IA', 'ID', 'IL', 'IN',
```

```
# The last element contains the labels
labels = raw[:, -1]
```

```
# The other data points are the electrocadriogram data
data = raw_data

train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=21
)
```

```
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)
min_val
max_val
```

```
<tf.Tensor: shape=(), dtype=float64, numpy=145.621384>
```

Normalize the data to `[0,1]`.

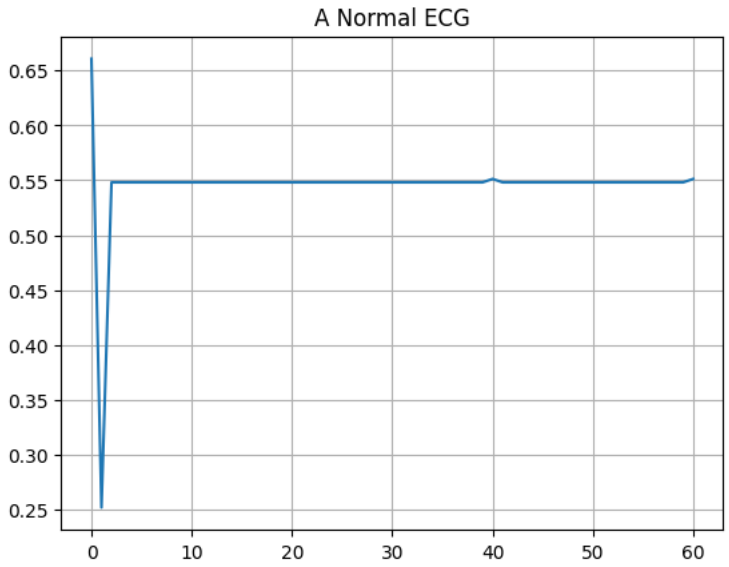
```
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)

train_data = tf.cast(train_data, tf.float32)
test_data = tf.cast(test_data, tf.float32)
```

You will train the autoencoder using only the normal rhythms, which are labeled in this dataset as `1`. Separate the normal rhythms from the abnormal rhythms.

```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

normal_train_data = train_data[train_labels]
normal_test_data = test_data[test_labels]

anomalous_train_data = train_data[~train_labels]
anomalous_test_data = test_data[~test_labels]
```

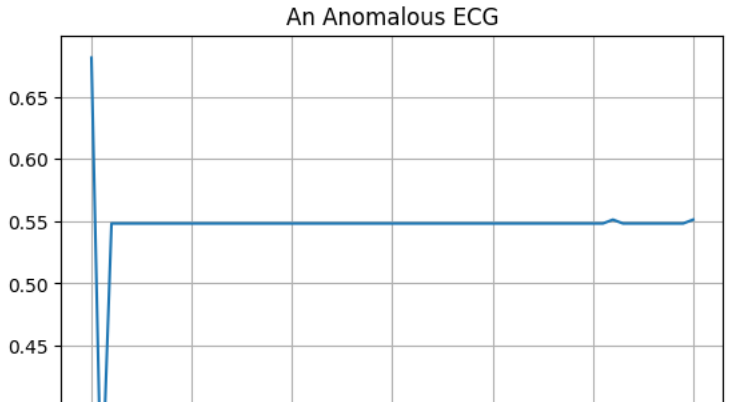
Plot a normal ECG.

```
plt.grid()
plt.plot(np.arange(61), normal_train_data[0])
plt.title("A Normal ECG")
plt.show()
```



Plot an anomalous ECG.

```
plt.grid()
plt.plot(np.arange(61), anomalous_train_data[0])
plt.title("An Anomalous ECG")
plt.show()
```



Build the model

```
class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Dense(32, activation="relu"),
      layers.Dense(16, activation="relu"),
      layers.Dense(8, activation="relu")])

    self.decoder = tf.keras.Sequential([
      layers.Dense(16, activation="relu"),
      layers.Dense(32, activation="relu"),
      layers.Dense(61, activation="sigmoid")])

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

autoencoder = AnomalyDetector()


autoencoder.compile(optimizer='adam', loss='mae')
```

Notice that the autoencoder is trained using only the normal ECGs, but is evaluated using the full test set.

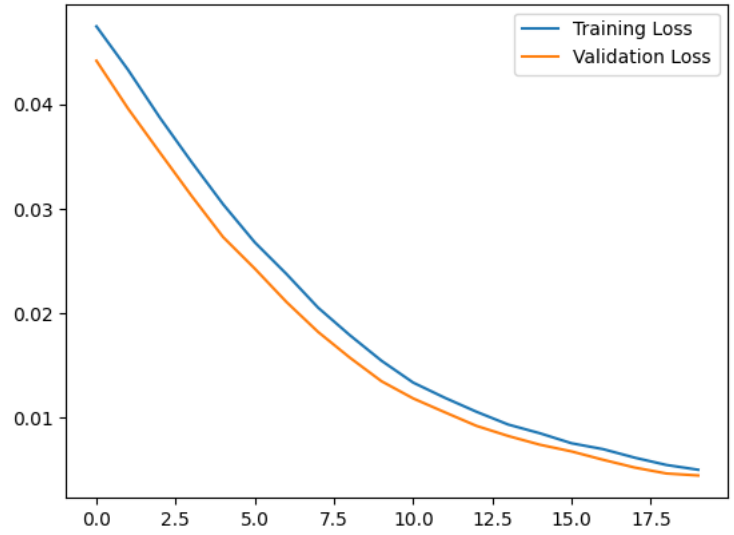
```
history = autoencoder.fit(normal_train_data, normal_train_data,
          epochs=20,
          batch_size=512,
          validation_data=(test_data, test_data),
          shuffle=True)
```

```
Epoch 1/20
4/4 [==============================] - 2s 207ms/step - loss: 0.0474 - val_loss: 0.0441
Epoch 2/20
4/4 [==============================] - 0s 20ms/step - loss: 0.0433 - val_loss: 0.0396
Epoch 3/20
4/4 [==============================] - 0s 23ms/step - loss: 0.0387 - val_loss: 0.0354
Epoch 4/20
4/4 [==============================] - 0s 23ms/step - loss: 0.0345 - val_loss: 0.0312
Epoch 5/20
4/4 [==============================] - 0s 21ms/step - loss: 0.0304 - val_loss: 0.0273
Epoch 6/20
4/4 [==============================] - 0s 26ms/step - loss: 0.0268 - val_loss: 0.0243
Epoch 7/20
4/4 [==============================] - 0s 23ms/step - loss: 0.0237 - val_loss: 0.0211
Epoch 8/20
4/4 [==============================] - 0s 18ms/step - loss: 0.0205 - val_loss: 0.0182
Epoch 9/20
4/4 [==============================] - 0s 18ms/step - loss: 0.0179 - val_loss: 0.0157
Epoch 10/20
4/4 [==============================] - 0s 24ms/step - loss: 0.0155 - val_loss: 0.0135
Epoch 11/20
4/4 [==============================] - 0s 19ms/step - loss: 0.0134 - val_loss: 0.0118
Epoch 12/20
4/4 [==============================] - 0s 23ms/step - loss: 0.0119 - val_loss: 0.0105
Epoch 13/20
4/4 [==============================] - 0s 25ms/step - loss: 0.0106 - val_loss: 0.0092
```

```
Epoch 14/20
4/4 [==============================] - 0s 24ms/step - loss: 0.0094 - val_loss: 0.0083
Epoch 15/20
4/4 [==============================] - 0s 24ms/step - loss: 0.0085 - val_loss: 0.0074
Epoch 16/20
4/4 [==============================] - 0s 19ms/step - loss: 0.0076 - val_loss: 0.0068
Epoch 17/20
4/4 [==============================] - 0s 24ms/step - loss: 0.0070 - val_loss: 0.0060
Epoch 18/20
4/4 [==============================] - 0s 21ms/step - loss: 0.0062 - val_loss: 0.0052
Epoch 19/20
4/4 [==============================] - 0s 27ms/step - loss: 0.0055 - val_loss: 0.0047
Epoch 20/20
4/4 [==============================] - 0s 18ms/step - loss: 0.0050 - val_loss: 0.0045
```

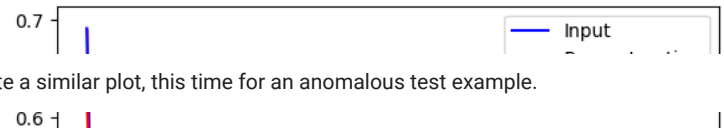
```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c98046253f0>
```



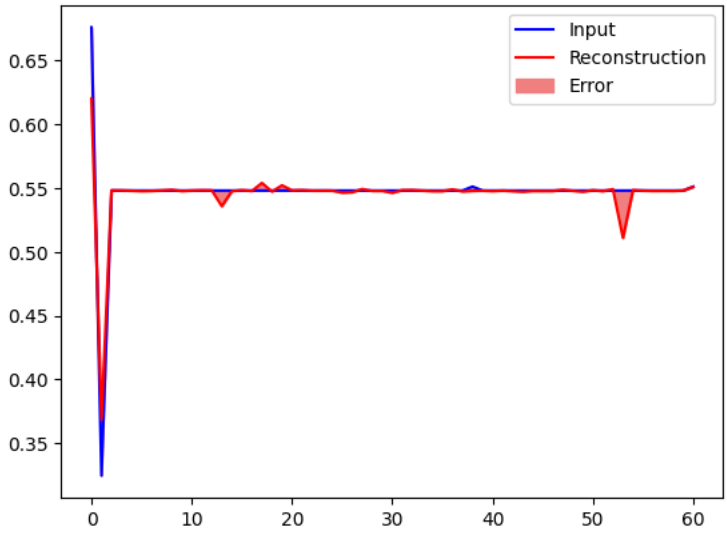
You will soon classify an ECG as anomalous if the reconstruction error is greater than one standard deviation from the normal training examples. First, let's plot a normal ECG from the training set, the reconstruction after it's encoded and decoded by the autoencoder, and the reconstruction error.

```
encoded_data = autoencoder.encoder(normal_test_data).numpy()
decoded_data = autoencoder.decoder(encoded_data).numpy()

plt.plot(normal_test_data[0], 'b')
plt.plot(decoded_data[0], 'r')
plt.fill_between(np.arange(61), decoded_data[0], normal_test_data[0], color='lightcoral')
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```



Create a similar plot, this time for an anomalous test example.


```
encoded_data = autoencoder.encoder(anomalous_test_data).numpy()
decoded_data = autoencoder.decoder(encoded_data).numpy()

plt.plot(anomalous_test_data[0], 'b')
plt.plot(decoded_data[0], 'r')
plt.fill_between(np.arange(61), decoded_data[0], anomalous_test_data[0], color='lightcoral')
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```



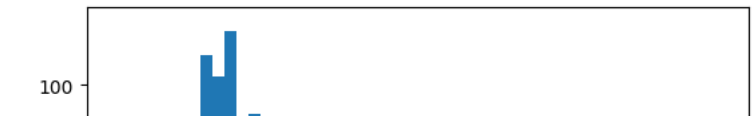
Detect anomalies

Detect anomalies by calculating whether the reconstruction loss is greater than a fixed threshold. In this tutorial, you will calculate the mean average error for normal examples from the training set, then classify future examples as anomalous if the reconstruction error is higher than one standard deviation from the training set.

Plot the reconstruction error on normal ECGs from the training set

```
reconstructions = autoencoder.predict(normal_train_data)
train_loss = tf.keras.losses.mae(reconstructions, normal_train_data)

plt.hist(train_loss[None,:], bins=50)
plt.xlabel("Train loss")
plt.ylabel("No of examples")
plt.show()
```

```
55/55 [==============================] - 0s 2ms/step
```



Choose a threshold value that is one standard deviations above the mean.

```
threshold = np.mean(train_loss) + np.std(train_loss)
print("Threshold: ", threshold)
```

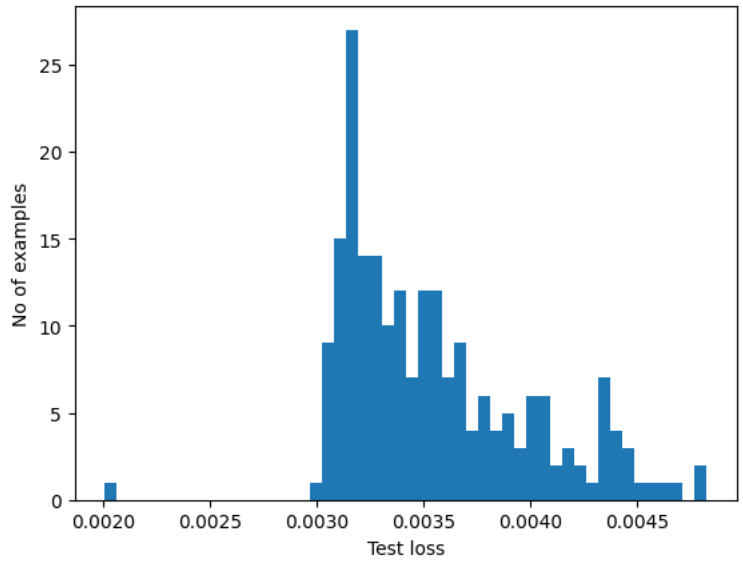
```
Threshold:  0.0066462317
```

Note: There are other strategies you could use to select a threshold value above which test examples should be classified as anomalous, the correct approach will depend on your dataset. You can learn more with the links at the end of this tutorial.

If you examine the reconstruction error for the anomalous examples in the test set, you'll notice most have greater reconstruction error than the threshold. By varing the threshold, you can adjust the precision and recall of your classifier.

```
reconstructions = autoencoder.predict(anomalous_test_data)
test_loss = tf.keras.losses.mae(reconstructions, anomalous_test_data)

plt.hist(test_loss[None, :], bins=50)
plt.xlabel("Test loss")
plt.ylabel("No of examples")
plt.show()
```

```
7/7 [==============================] - 0s 2ms/step
```



Classify an ECG as an anomaly if the reconstruction error is greater than the threshold.

```
def predict(model, data, threshold):
  reconstructions = model(data)
  loss = tf.keras.losses.mae(reconstructions, data)
  return tf.math.less(loss, threshold)

def print_stats(predictions, labels):
  print("Accuracy = {}".format(accuracy_score(labels, predictions)))
  print("Precision = {}".format(precision_score(labels, predictions)))
  print("Recall = {}".format(recall_score(labels, predictions)))


preds = predict(autoencoder, test_data, threshold)
print_stats(preds, test_labels)
```

```
Accuracy = 0.6094674556213018
Precision = 0.6602564102564102
Recall = 0.8879310344827587
```