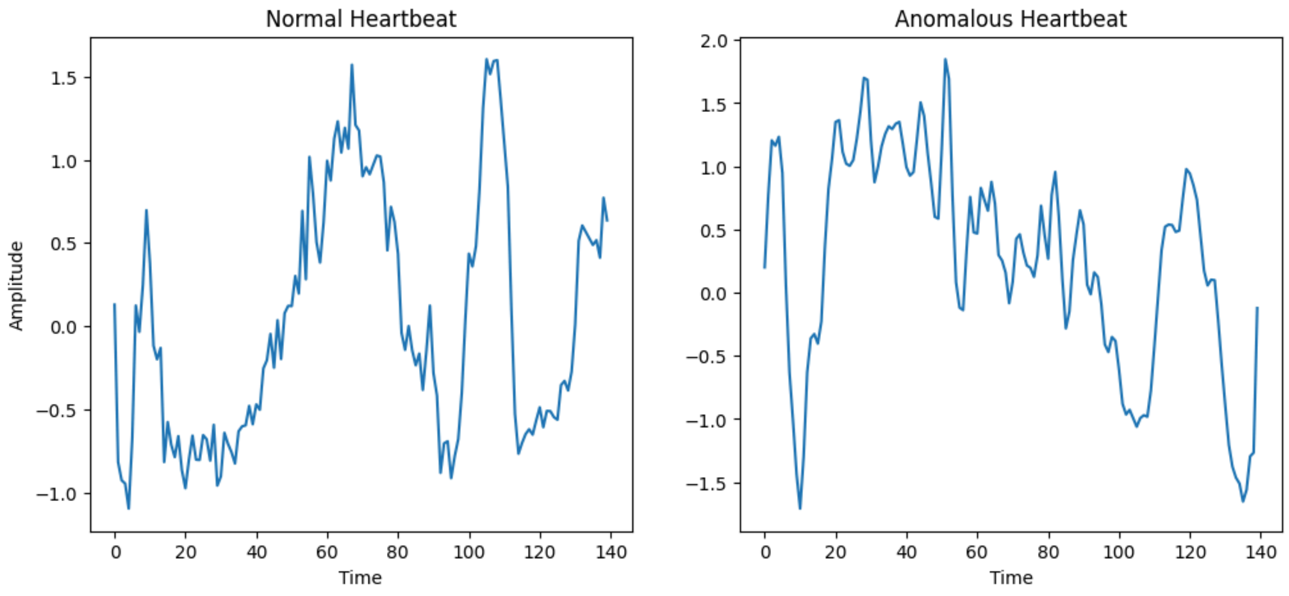
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

print("TensorFlow Version:", tf.__version__)
```

```
TensorFlow Version: 2.19.0
```

```
# Download and load the dataset
dataframe = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv', header=None)
features, labels = dataframe.iloc[:, :-1], dataframe.iloc[:, -1]

# --- Preprocessing ---
# Scale the features to have zero mean and unit variance
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features.values)

# Separate the normal data (label 1) from anomalies (label 0)
normal_features = scaled_features[labels == 1]
anomalous_features = scaled_features[labels == 0]

# We will only train on the normal data
x_train, x_test_normal = train_test_split(normal_features, test_size=0.2, random_state=42)

print(f"Number of normal heartbeats for training: {len(x_train)}")
print(f"Number of normal heartbeats for validation: {len(x_test_normal)}")
print(f"Number of anomalous heartbeats: {len(anomalous_features)}")

# Let's plot a normal vs. an anomalous heartbeat
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(normal_features[0])
plt.title("Normal Heartbeat")
plt.xlabel("Time")
plt.ylabel("Amplitude")

plt.subplot(1, 2, 2)
plt.plot(anomalous_features[0])
plt.title("Anomalous Heartbeat")
plt.xlabel("Time")
plt.show()
```

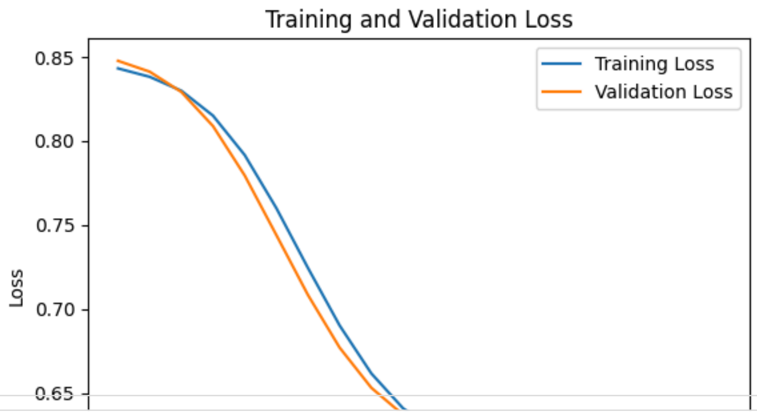
```
Number of normal heartbeats for training: 2335
Number of normal heartbeats for validation: 584
Number of anomalous heartbeats: 2079
```



```
# Define the Autoencoder model architecture
class AnomalyDetector(models.Model):
    def __init__(self):
        super(AnomalyDetector, self).__init__()
        # c. The encoder converts it into a latent representation
        self.encoder = tf.keras.Sequential([
            layers.Dense(32, activation="relu"),
            layers.Dense(16, activation="relu"),
            layers.Dense(8, activation="relu") # The bottleneck (latent representation)
        ])

        # d. Decoder networks convert it back to the original input
        self.decoder = tf.keras.Sequential([
            layers.Dense(16, activation="relu"),
            layers.Dense(32, activation="relu"),
            layers.Dense(140, activation="sigmoid") # Output layer matches input dimensions
        ])

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

autoencoder = AnomalyDetector()
```

```
# e. Compile the models with Optimizer, Loss, and Evaluation Metrics
autoencoder.compile(optimizer='adam', loss='mae')

# Train the model on normal data ONLY
history = autoencoder.fit(x_train, x_train,
                          epochs=20,
                          batch_size=512,
                          validation_data=(x_test_normal, x_test_normal),
                          shuffle=True)

# Plot training and validation loss
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
Epoch 1/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 65ms/step - loss: 0.8458 - val_loss: 0.8475
Epoch 2/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.8398 - val_loss: 0.8411
Epoch 3/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.8324 - val_loss: 0.8290
Epoch 4/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.8178 - val_loss: 0.8087
Epoch 5/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.7998 - val_loss: 0.7794
Epoch 6/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.7672 - val_loss: 0.7440
Epoch 7/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 0.7292 - val_loss: 0.7083
Epoch 8/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.6892 - val_loss: 0.6770
Epoch 9/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.6676 - val_loss: 0.6532
Epoch 10/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.6446 - val_loss: 0.6372
Epoch 11/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.6248 - val_loss: 0.6267
Epoch 12/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.6159 - val_loss: 0.6186
Epoch 13/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.6174 - val_loss: 0.6108
Epoch 14/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.6030 - val_loss: 0.6047
Epoch 15/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.5957 - val_loss: 0.6001
Epoch 16/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.5953 - val_loss: 0.5965
Epoch 17/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.5917 - val_loss: 0.5930
Epoch 18/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.5804 - val_loss: 0.5902
Epoch 19/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.5812 - val_loss: 0.5872
Epoch 20/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.5781 - val_loss: 0.5851
```



```
# 1. Get reconstructions for normal test data
reconstructed_normal = autoencoder.predict(x_test_normal)
# Calculate the Mean Absolute Error for each normal sample
normal_loss = tf.keras.losses.mae(x_test_normal, reconstructed_normal)

# 2. Get reconstructions for anomalous data
reconstructed_anomalous = autoencoder.predict(anomalous_features)
# Calculate the Mean Absolute Error for each anomalous sample
anomalous_loss = tf.keras.losses.mae(anomalous_features, reconstructed_anomalous)

# 3. Plot the loss distributions
plt.figure(figsize=(10, 6))
plt.hist(normal_loss[None, :], bins=50, alpha=0.7, label='Normal')
plt.hist(anomalous_loss[None, :], bins=50, alpha=0.7, label='Anomalous')
plt.xlabel("Reconstruction Loss")
plt.ylabel("Number of Samples")
plt.title("Distribution of Reconstruction Loss")
plt.legend()
plt.show()
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```