

## ✓ Question 1 - LSTM Autoencoder

You are required to build an LSTM Autoencoder to detect anomalies in a time series dataset. The dataset contains daily temperature readings from a weather station over the course of a few years. Parameters in the dataset [Date and Temperature]

<https://drive.google.com/drive/folders/1rD4HcUNmh7fLz-bt68VxQMfGzPzV-oeF> [You can also prepare Synthetic dataset with 2500 instances with the features Date and Temperature]

Your task is to:

1. Load the dataset: The dataset will contain a single column temperature and a date column.
2. Preprocess the data: Normalize the temperature data and split it into training and testing sets.
3. Build an LSTM Autoencoder:
  - o The encoder should reduce the input dimensions to a latent representation.
  - o The decoder should reconstruct the input from the latent representation.
4. Train the model: Train the autoencoder on the training data and evaluate the reconstruction error on the test set.
5. Anomaly Detection: Use the reconstruction error to detect anomalies. Define a threshold for the reconstruction error, and identify days where the temperature is considered anomalous.
6. Visualize the results: Plot the original temperature data and highlight the detected anomalies.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, RepeatVector, TimeDistributed
from tensorflow.keras.optimizers import Adam

data = pd.read_csv("/content/weather_data.csv", parse_dates=["date"], index_col="date")
```

## ✓ Normalization

```
scaler = MinMaxScaler()
data["temperature"] = scaler.fit_transform(data[["temperature"]])
```

## ✓ Putting it into a sequence

```
def create_sequences(data, seq_length):
    sequences = []
    for i in range(len(data) - seq_length):
        seq = data[i:i + seq_length]
        sequences.append(seq)
    return np.array(sequences)
```

## ✓ Giving Length to the sequence

```
sequence_length = 30
temperature_values = data["temperature"].values
sequences = create_sequences(temperature_values, sequence_length)
```

## ✓ Split the data into training and testing sets

```
X_train, X_test = train_test_split(sequences, test_size=0.2, random_state=42)
```

## ✓ Reshape data for LSTM input


```
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

## ✓ Build the LSTM Autoencoder

here autoencoder is used to reduces the input sequence into a lower-dimensional latent representation. and Repeatvector rearrange the data so that the decoder can reconstruct it to the original form.

```
model = Sequential([
    LSTM(64, activation="relu", input_shape=(sequence_length, 1), return_sequences=False),
    RepeatVector(sequence_length),
    LSTM(64, activation="relu", return_sequences=True),
    TimeDistributed(Dense(1))
])
```

```
model.compile(optimizer=Adam(learning_rate=0.001), loss="mse")
model.summary()
```

 /usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to the `LSTM` layer. The input shape is inferred from the first batch of data.  
 super().\_\_init\_\_(\*\*kwargs)  
 Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	16,896
repeat_vector (RepeatVector)	(None, 30, 64)	0
lstm_1 (LSTM)	(None, 30, 64)	33,024
time_distributed (TimeDistributed)	(None, 30, 1)	65

Total params: 49,985 (195.25 KB)  
 Trainable params: 49,985 (195.25 KB)  
 Non-trainable params: 0 (0.00 B)

## ✓ Training the model

```
epochs = 50
batch_size = 32
history = model.fit(
    X_train, X_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_split=0.2,
    shuffle=True
)
```



```

51/51 ————— 2s 34ms/step - loss: 0.0021 - val_loss: 0.0021
Epoch 37/50
51/51 ————— 2s 34ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 38/50
51/51 ————— 2s 33ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 39/50
51/51 ————— 2s 33ms/step - loss: 0.0021 - val_loss: 0.0021
Epoch 40/50
51/51 ————— 2s 35ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 41/50
51/51 ————— 3s 51ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 42/50
51/51 ————— 2s 34ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 43/50
51/51 ————— 2s 34ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 44/50
51/51 ————— 3s 33ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 45/50
51/51 ————— 3s 33ms/step - loss: 0.0021 - val_loss: 0.0021
Epoch 46/50
51/51 ————— 2s 42ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 47/50
51/51 ————— 3s 55ms/step - loss: 0.0023 - val_loss: 0.0022
Epoch 48/50
51/51 ————— 2s 37ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 49/50
51/51 ————— 2s 34ms/step - loss: 0.0021 - val_loss: 0.0021
Epoch 50/50
51/51 ————— 2s 35ms/step - loss: 0.0022 - val_loss: 0.0021

```

## ✓ Evaluate on test data

```

reconstruction = model.predict(X_test)
reconstruction_error = np.mean(np.abs(reconstruction - X_test), axis=(1, 2))

```

```

↔ 16/16 ————— 1s 44ms/step

```

## ✓ Anomaly Detection

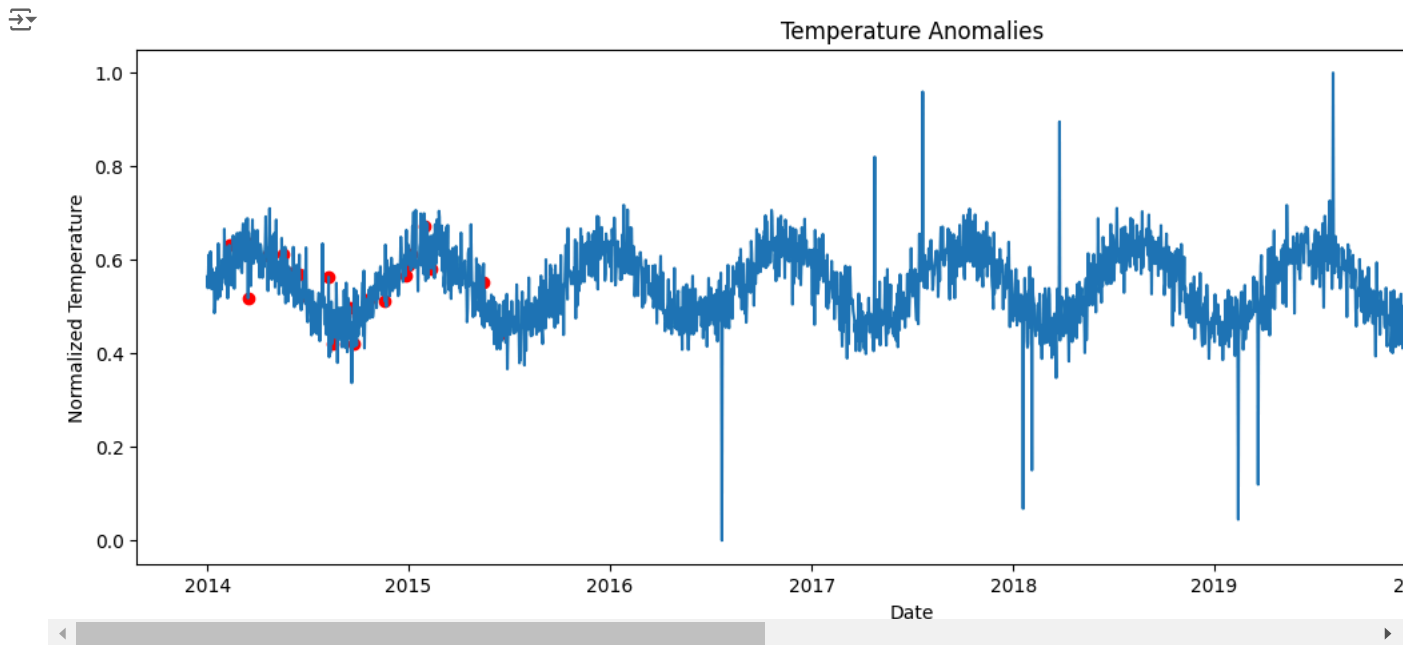
Thresholding is given for detecting anomaly in the series

```

threshold = np.percentile(reconstruction_error, 95)
anomalies = reconstruction_error > threshold
anomaly_dates = data.index[sequence_length:][np.where(anomalies)[0]]

plt.figure(figsize=(15, 5))
plt.plot(data.index, data["temperature"], label="Temperature")
plt.scatter(anomaly_dates, data.loc[anomaly_dates, "temperature"], color="red", label="Anomalies")
plt.title("Temperature Anomalies")
plt.xlabel("Date")
plt.ylabel("Normalized Temperature")
plt.legend()
plt.show()

```



Interpretation The temperature data shows seasonal patterns, likely corresponding to yearly climatic changes with periodic rises and falls.

The anomalies marked as red points are detected where the reconstruction error exceeds the predefined threshold.

These points represent instances where the LSTM was not able to reconstruct the input sequence.