


Question 1 - LSTM Autoencoder Objective: 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]

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense, RepeatVector, TimeDistributed
import numpy as np
```

```
data = pd.read_csv('/content/weather_data.csv', parse_dates=['date'])
data.set_index('date', inplace=True)

print(data.head())
```



date	temperature
2014-01-01	10.248357
2014-01-02	9.950428
2014-01-03	10.362958
2014-01-04	10.820167
2014-01-05	9.961091

### Preprocessing and normalizing the data

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

# Convert data into a supervised learning format
temperature_data = data['temperature'].values

# Split into training and testing
train_size = int(len(temperature_data) * 0.8)
train_data, test_data = temperature_data[:train_size], temperature_data[train_size:]
```

### Creating the sequence

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

sequence_length = 30
train_sequences = create_sequences(train_data, sequence_length)
test_sequences = create_sequences(test_data, sequence_length)
```

```
train_sequences = train_sequences.reshape(-1, sequence_length, 1)
test_sequences = test_sequences.reshape(-1, sequence_length, 1)
```

### Building the LSTM Model

```
input = train_sequences.shape[1] # sequence length
features = train_sequences.shape[2]
```


**\*\*Defining the Autoencoder**

```
# Define the Autoencoder model with additional layers
inputs = Input(shape=(input, features))

# Encoder
encoded = LSTM(128, activation='relu', return_sequences=True)(inputs)
encoded = LSTM(64, activation='relu', return_sequences=False)(encoded)
latent = Dense(32, activation='relu')(encoded) # Dense layer for latent space
latent_repeated = RepeatVector(input)(latent)

# Decoder
decoded = LSTM(64, activation='relu', return_sequences=True)(latent_repeated)
decoded = LSTM(128, activation='relu', return_sequences=True)(decoded)
output = TimeDistributed(Dense(1))(decoded)
```

```
autoencoder = Model(inputs, output)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.summary()
```

 **Model: "functional"**

Layer (type)	Output Shape	Param #
input_layer ( <a href="#">InputLayer</a> )	(None, 30, 1)	0
lstm ( <a href="#">LSTM</a> )	(None, 30, 128)	66,560
lstm_1 ( <a href="#">LSTM</a> )	(None, 64)	49,408
dense ( <a href="#">Dense</a> )	(None, 32)	2,080
repeat_vector ( <a href="#">RepeatVector</a> )	(None, 30, 32)	0
lstm_2 ( <a href="#">LSTM</a> )	(None, 30, 64)	24,832
lstm_3 ( <a href="#">LSTM</a> )	(None, 30, 128)	98,816
time_distributed ( <a href="#">TimeDistributed</a> )	(None, 30, 1)	129

**Total params: 241.825** (944.63 KB)

```
history = autoencoder.fit(train_sequences, train_sequences, epochs=50, batch_size=32, validation_split=0.1,
```



```

57/57 ██████████ 9s 152ms/step - loss: 0.0021 - val_loss: 0.0033
Epoch 32/50
57/57 ██████████ 11s 168ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 33/50
57/57 ██████████ 8s 133ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 34/50
57/57 ██████████ 11s 138ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 35/50
57/57 ██████████ 11s 157ms/step - loss: 0.0021 - val_loss: 0.0033
Epoch 36/50
57/57 ██████████ 11s 176ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 37/50
57/57 ██████████ 7s 117ms/step - loss: 0.0021 - val_loss: 0.0033
Epoch 38/50
57/57 ██████████ 10s 175ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 39/50
57/57 ██████████ 10s 176ms/step - loss: 0.0020 - val_loss: 0.0032
Epoch 40/50
57/57 ██████████ 9s 150ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 41/50
57/57 ██████████ 9s 149ms/step - loss: 0.0020 - val_loss: 0.0032
Epoch 42/50
57/57 ██████████ 9s 134ms/step - loss: 0.0022 - val_loss: 0.0032
Epoch 43/50
57/57 ██████████ 9s 115ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 44/50
57/57 ██████████ 10s 115ms/step - loss: 0.0020 - val_loss: 0.0032
Epoch 45/50
57/57 ██████████ 12s 152ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 46/50
57/57 ██████████ 11s 160ms/step - loss: 0.0020 - val_loss: 0.0032
Epoch 47/50
57/57 ██████████ 8s 118ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 48/50
57/57 ██████████ 11s 135ms/step - loss: 0.0021 - val_loss: 0.0033
Epoch 49/50
57/57 ██████████ 8s 141ms/step - loss: 0.0021 - val_loss: 0.0033
Epoch 50/50
57/57 ██████████ 9s 115ms/step - loss: 0.0020 - val_loss: 0.0032

```

## Evaluating the reconstructed error on the test data

```

reconstructed_sequences = autoencoder.predict(test_sequences)
reconstruction_error = np.mean(np.abs(reconstructed_sequences - test_sequences), axis=(1, 2))

```

```

⇒ 16/16 ██████████ 2s 105ms/step

```

```

print(reconstruction_error)

```

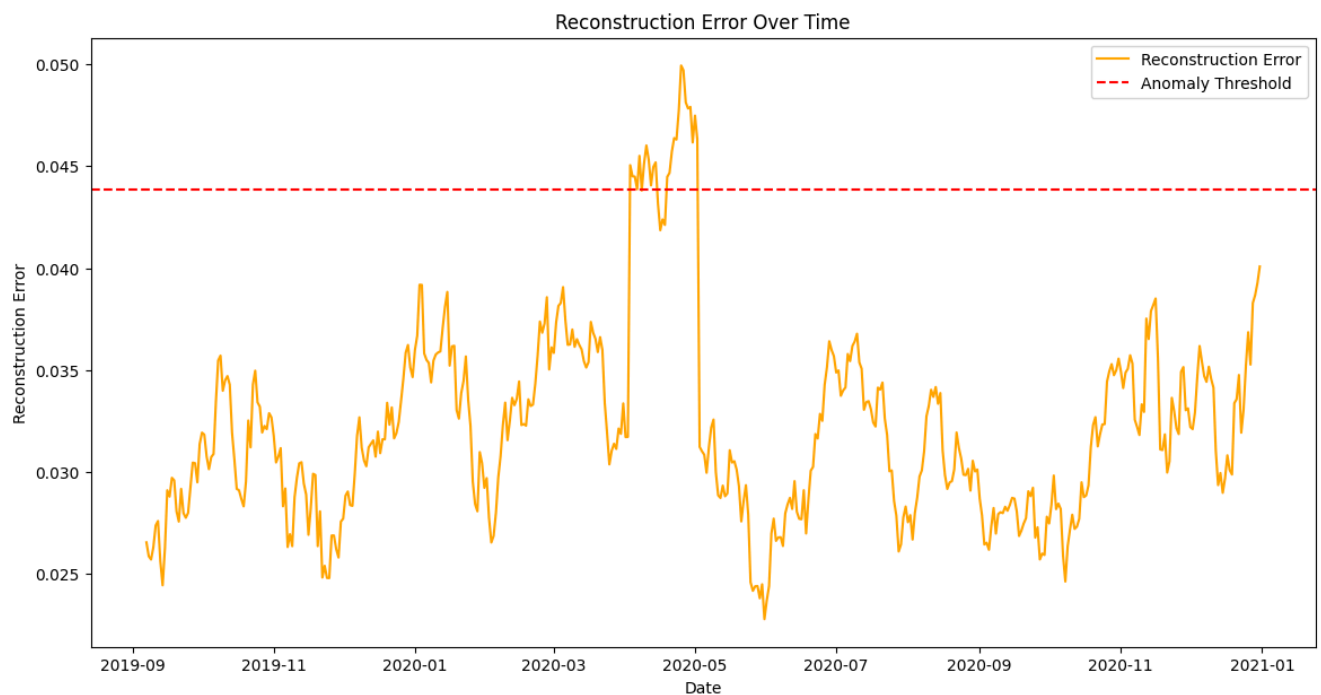
```

⇒

```

0.02857368 0.029355039 0.02790201 0.0245871 0.02416659 0.02458939  
0.02440128 0.02379951 0.02448565 0.02277768 0.02368871 0.02440376  
0.02698077 0.02770839 0.02661778 0.02678649 0.02679724 0.02636816  
0.02797082 0.02840413 0.02872681 0.02818521 0.02954829 0.02804828  
0.02770231 0.02767882 0.02910171 0.02698125 0.02865417 0.03005976  
0.03024631 0.03185913 0.03164115 0.03285141 0.03250482 0.03426595  
0.03513109 0.03641387 0.03598622 0.03568404 0.03487864 0.03497899  
0.03373735 0.03399788 0.03414225 0.03578845 0.03543721 0.03617814  
0.03640196 0.03678376 0.0353737 0.03506329 0.0330481 0.03342571  
0.03347462 0.03307949 0.03242754 0.0322284 0.03413633 0.03403891  
0.03438548 0.03261509 0.03181217 0.03003165 0.03007343 0.02857571  
0.02784397 0.02609654 0.02644606 0.02776655 0.02830365 0.02753163  
0.02788164 0.02668298 0.02799736 0.02873901 0.02977732 0.03007716  
0.03101995 0.03273423 0.03323475 0.0340325 0.03368193 0.0341692  
0.03334407 0.03387023 0.03104715 0.02984039 0.02916631 0.0294828  
0.0295489 0.03012489 0.03193703 0.0311832 0.03065995 0.02986489  
0.02984381 0.03015198 0.02908283 0.0305484 0.03002394 0.03011742  
0.02869221 0.02786463 0.0264399 0.02652712 0.0261804 0.02732668  
0.02823125 0.02697547 0.02793366 0.02802372 0.02797888 0.02828991  
0.02809544 0.02837469 0.02872604 0.02869146 0.02807559 0.02685657  
0.02711559 0.02746921 0.02772862 0.02905359 0.02881187 0.02922308  
0.02678 0.02728398 0.02570515 0.02599337 0.0259262 0.02780557  
0.02746966 0.02840401 0.02982425 0.02817437 0.02844608 0.02818287  
0.02590256 0.02462226 0.02635045 0.02716752 0.02789983 0.02720564  
0.02731731 0.02769811 0.02949184 0.02876363 0.02884772 0.02934087  
0.03120701 0.0323035 0.03268874 0.03125484 0.03185643 0.03232873  
0.03234745 0.03442992 0.03494237 0.03529073 0.03474607 0.03499222  
0.03556191 0.03493811 0.0341155 0.0348742 0.03506122 0.03572457  
0.03531657 0.03256198 0.03221482 0.03181012 0.03331924 0.03294514  
0.03752612 0.0365204 0.03790033 0.03818956 0.03851186 0.03557505  
0.03110378 0.03107529 0.03184307 0.02997039 0.0305227 0.03363525  
0.03299427 0.03214885 0.03185787 0.03491445 0.03514757 0.03302729  
0.03311939 0.03220335 0.03210082 0.03291542 0.03466083 0.03617977  
0.03541436 0.03470181 0.03442113 0.03516349 0.03456202 0.03413546  
0.03104474 0.02935513 0.02994661 0.02897669 0.02969342 0.03081458  
0.03009334 0.02987376 0.03337426 0.03357078 0.03475793 0.03192574  
0.03306044 0.03520394 0.03685619 0.03526705 0.03828815 0.03867658  
0.03927318 0.04008049]

```
# Plotting Reconstruction Error
plt.figure(figsize=(14, 7))
plt.plot(test_dates, reconstruction_error, label='Reconstruction Error', color='orange')
plt.axhline(y=threshold, color='red', linestyle='--', label='Anomaly Threshold')
plt.title('Reconstruction Error Over Time')
plt.xlabel('Date')
plt.ylabel('Reconstruction Error')
plt.legend()
plt.show()
```



The graph illustrates the reconstruction error over time and highlights the points where the error surpasses the defined anomaly threshold, indicating potential anomalies in the temperature data. Significant spikes in the reconstruction error occur at certain dates, signaling that the temperature readings for those days deviated from the expected pattern. Specifically, the threshold is crossed **around March 2020**, where there is a notable increase in the reconstruction error, suggesting an unusual temperature event. Additionally, another significant anomaly is observed towards the **end of 2020**, where the error exceeds the threshold once again.

### Defining the Threshold for anomaly

```
threshold = np.percentile(reconstruction_error, 95)
print(f"Anomaly detection threshold: {threshold}")
```



Anomaly detection threshold: 0.04386938399752767

### Identifying the anomalies

```
anomalies = reconstruction_error > threshold

# Ensuring that test_dates matches the length of reconstruction_error so that it won't be mismatching with
test_dates = test_dates[:len(reconstruction_error)]

# Creating a DataFrame for anomalies
anomalies_df = pd.DataFrame({
    'Date': test_dates,
    'Reconstruction_Error': reconstruction_error,
    'Anomaly': anomalies
})
```

### Visualizing the Anomalies

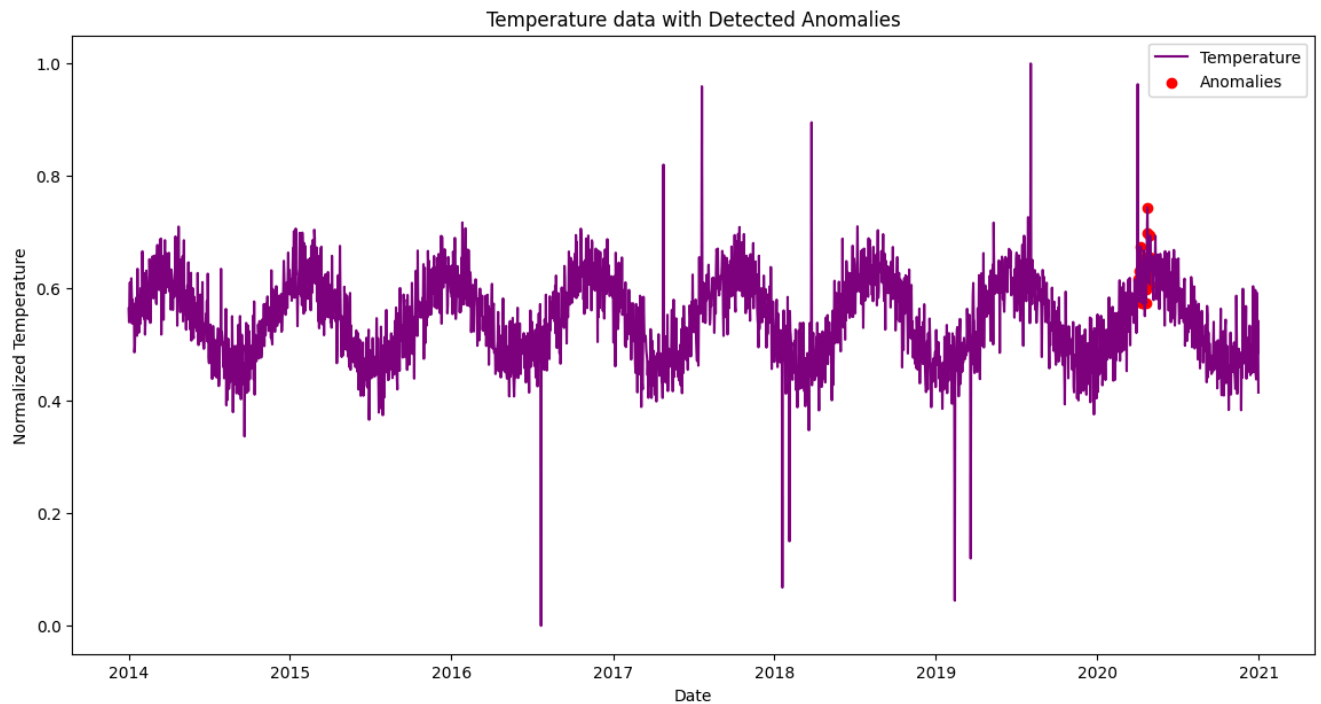
```
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 7))
```

```
plt.plot(data.index, data['temperature'], label='Temperature', color='purple')

# Highlighting anomalies
anomaly_dates = anomalies_df[anomalies_df['Anomaly']]['Date']
plt.scatter(anomaly_dates, data.loc[anomaly_dates, 'temperature'], color='red', label='Anomalies')

plt.title('Temperature data with Detected Anomalies')
plt.xlabel('Date')
plt.ylabel('Normalized Temperature')
plt.legend()
plt.show()
```



## Interpretation

The plot shows temperature changes over the years, with the green line representing the normal seasonal patterns. The red dots highlight unusual days when the temperature was significantly different from what's expected, either much higher or lower. These unusual points could indicate extreme weather events, like heatwaves or cold spells, or possible errors in the data. Most of the temperatures follow the normal trend, but the red dots stand out as anomalies that need further investigation to understand what caused them.