In [ ]:
```
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, TimeDistri
import numpy as np
```

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

print(data.head())
```

```
            temperature
date
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

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

temperature_data = data['temperature'].values

# Split into training and testing here train is 80% and test is 20%
train_size = int(len(temperature_data) * 0.8)
train_data, test_data = temperature_data[:train_size], temperature_data[train_si
```

Creating the sequence

In [ ]:
```
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)
```

In [ ]:
```
train_sequences = train_sequences.reshape(-1, sequence_length, 1)
test_sequences = test_sequences.reshape(-1, sequence_length, 1)
```

Building the LSTM Model

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

Defining the Autoencoder model

In [ ]:
```
# Define the Autoencoder model
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)
```

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

Model: "functional"

Layer (type)	Output Shape
input_layer (InputLayer)	(None, 30, 1)
lstm (LSTM)	(None, 30, 128)
lstm_1 (LSTM)	(None, 64)
dense (Dense)	(None, 32)
repeat_vector (RepeatVector)	(None, 30, 32)
lstm_2 (LSTM)	(None, 30, 64)
lstm_3 (LSTM)	(None, 30, 128)
time_distributed (TimeDistributed)	(None, 30, 1)

Total params: 241,825 (944.63 KB)

Trainable params: 241,825 (944.63 KB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
history = autoencoder.fit(train_sequences, train_sequences, epochs=50, batch_siz
```

```
Epoch 1/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 16s 134ms/step - loss: 0.1476 - val_loss: 0.0215
Epoch 2/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 11s 158ms/step - loss: 0.0179 - val_loss: 0.0082
Epoch 3/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 9s 144ms/step - loss: 0.0050 - val_loss: 0.0037
Epoch 4/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 9s 115ms/step - loss: 0.0025 - val_loss: 0.0041
Epoch 5/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 12s 151ms/step - loss: 0.0025 - val_loss: 0.0034
Epoch 6/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 8s 147ms/step - loss: 0.0023 - val_loss: 0.0034
Epoch 7/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 7s 116ms/step - loss: 0.0023 - val_loss: 0.0034
Epoch 8/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 9s 154ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 9/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 7s 131ms/step - loss: 0.0023 - val_loss: 0.0034
Epoch 10/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 9s 114ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 11/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 11s 129ms/step - loss: 0.0022 - val_loss: 0.0035
Epoch 12/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 8s 142ms/step - loss: 0.0024 - val_loss: 0.0035
Epoch 13/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 7s 117ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 14/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 12s 146ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 15/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 7s 119ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 16/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 10s 117ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 17/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 10s 113ms/step - loss: 0.0022 - val_loss: 0.0035
Epoch 18/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 9s 152ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 19/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 8s 115ms/step - loss: 0.0023 - val_loss: 0.0033
Epoch 20/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 10s 119ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 21/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 11s 134ms/step - loss: 0.0022 - val_loss: 0.0035
Epoch 22/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 11s 150ms/step - loss: 0.0022 - val_loss: 0.0033
Epoch 23/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 11s 163ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 24/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 8s 126ms/step - loss: 0.0023 - val_loss: 0.0035
Epoch 25/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 11s 145ms/step - loss: 0.0022 - val_loss: 0.0034
Epoch 26/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 10s 176ms/step - loss: 0.0023 - val_loss: 0.0036
Epoch 27/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 8s 139ms/step - loss: 0.0022 - val_loss: 0.0033
Epoch 28/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 11s 151ms/step - loss: 0.0021 - val_loss: 0.0035
Epoch 29/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 10s 151ms/step - loss: 0.0021 - val_loss: 0.0032
Epoch 30/50
57/57 ━━━━━━━━━━━━━━━━━━━━ 10s 139ms/step - loss: 0.0022 - val_loss: 0.0033
```

```
Epoch 31/50
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

In [ ]:

```
reconstructed_sequences = autoencoder.predict(test_sequences)
reconstruction_error = np.mean(np.abs(reconstructed_sequences - test_sequences),
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 105ms/step
```

In [ ]:

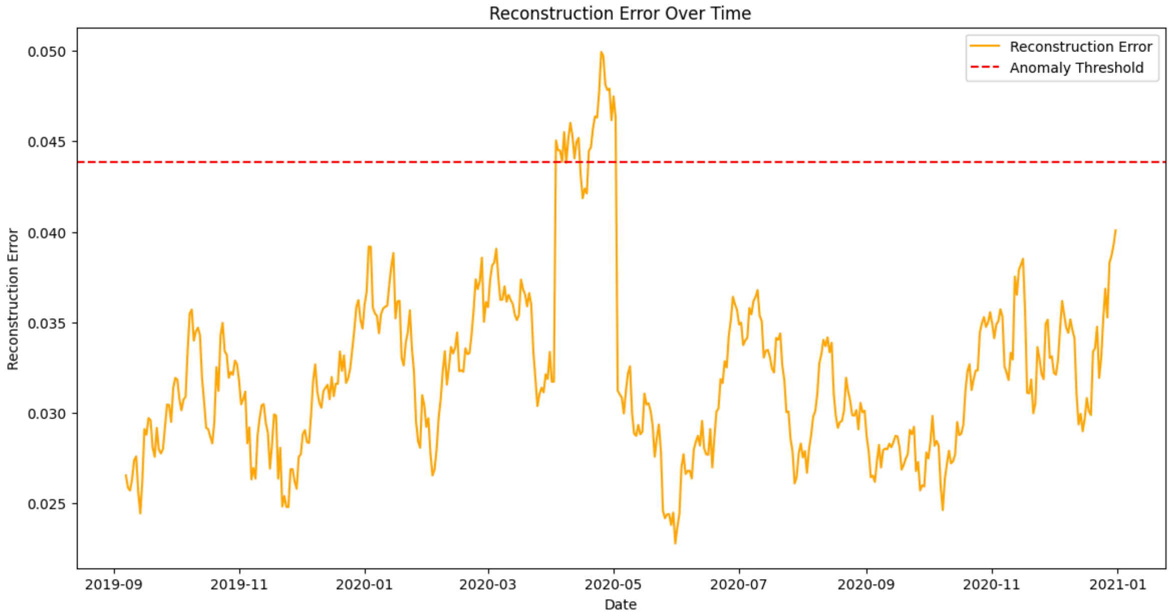
```
print(reconstruction_error)
```

```
[0.02654336 0.02586378 0.02570156 0.0263128  0.02737298 0.0275918
 0.02562043 0.0244387  0.02627327 0.02910139 0.02878802 0.02971195
 0.0295864  0.02806762 0.02756502 0.02916734 0.02798005 0.0277499
 0.02799226 0.0292581  0.03045627 0.03042937 0.02949074 0.03136913
 0.0319277  0.03183334 0.03076586 0.03013434 0.03073007 0.03088105
 0.03341803 0.03547095 0.03570749 0.0339844  0.03449505 0.03470341
 0.03428251 0.03191402 0.0306152  0.02916922 0.0290991  0.02867593
 0.02831038 0.02952028 0.03252427 0.03120742 0.03428234 0.03496773
 0.03341628 0.03320396 0.03193354 0.03225524 0.03210103 0.03288249
 0.0326788  0.03177464 0.03046389 0.03075379 0.03116748 0.02831598
 0.02918788 0.02631635 0.02694649 0.0263589  0.02875356 0.02970265
 0.03041847 0.03047745 0.02942998 0.02889165 0.02691572 0.02822017
 0.02990897 0.0298489  0.02636058 0.02806514 0.02481824 0.02539696
 0.0247987  0.02479107 0.02688083 0.0268815  0.02620983 0.02580223
 0.02756385 0.02770718 0.02883419 0.02904134 0.02837334 0.02833468
 0.02989393 0.03176818 0.03267954 0.0311951  0.03056417 0.03027849
 0.03120494 0.0313825  0.03154805 0.03074861 0.03198005 0.03091977
 0.03160488 0.03159999 0.03339173 0.03231817 0.03316526 0.03164898
 0.0319019  0.03244975 0.03348529 0.03457806 0.03584694 0.03623006
 0.0351235  0.03465509 0.03598382 0.03668959 0.03918505 0.03917643
 0.03581393 0.03549958 0.03534539 0.03439362 0.03545868 0.03576036
 0.03586183 0.03591478 0.03707687 0.03810749 0.03883241 0.03521689
 0.03616176 0.03619076 0.03303691 0.0326137  0.03386454 0.0344613
 0.03566637 0.03355186 0.03226199 0.02950803 0.02841898 0.02806674
 0.03097346 0.03042771 0.02922008 0.02969531 0.02775521 0.02653694
 0.02686071 0.02799531 0.02972301 0.03077673 0.0323043  0.03340023
 0.03155809 0.03250915 0.03363989 0.03327568 0.03358943 0.03443845
 0.0323     0.03235097 0.03226919 0.0335641  0.03323457 0.03330258
 0.03432403 0.03569718 0.03737895 0.03683084 0.03728195 0.03857271
 0.03502733 0.03611407 0.0358394  0.03733024 0.0381485  0.0382916
 0.03906982 0.03745168 0.03623868 0.03625457 0.03698829 0.03613772
 0.0365146  0.03623324 0.03601744 0.035435   0.03512188 0.03538906
 0.03736153 0.03683177 0.03653231 0.03588189 0.03661505 0.03598252
 0.03339652 0.03193173 0.03037013 0.03103588 0.03138352 0.03112924
 0.03213021 0.03186877 0.0333627  0.03170905 0.03172098 0.04505282
 0.04450559 0.04449171 0.04387247 0.04551043 0.04381066 0.04514442
 0.04601866 0.04531985 0.04405978 0.04496478 0.04518928 0.043159
 0.04186105 0.0423924  0.04211949 0.04447844 0.04467598 0.04572269
 0.0463824  0.04630958 0.04773516 0.04994293 0.04971113 0.04813613
 0.04783441 0.0479123  0.04616582 0.04748443 0.04632007 0.03122838
 0.03101682 0.03084349 0.0299596  0.03120415 0.03218197 0.03257026
 0.02999219 0.0288557  0.02872221 0.02933065 0.02880775 0.02895108
 0.03106506 0.03046218 0.03052536 0.03010881 0.02928688 0.02757141
 0.02857368 0.02935039 0.02790201 0.0245871  0.02416659 0.02438939
 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]
```

In [ ]:

```python
# Plotting Reconstruction Error
plt.figure(figsize=(14, 7))
plt.plot(test_dates, reconstruction_error, label='Reconstruction Error', color='
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

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

```
Anomaly detection threshold: 0.04386938399752767
```

Identifing the anomalies

In [ ]:
```
anomalies = reconstruction_error > threshold

# Ensuring that  test_dates matches the length of reconstruction_error so that i
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
})
```

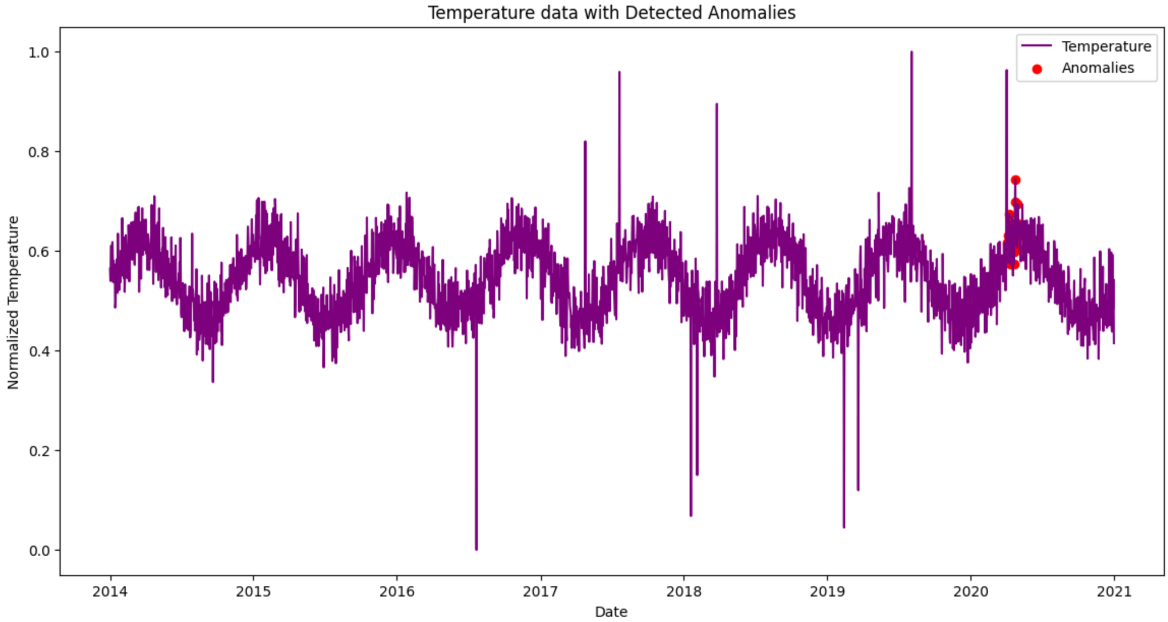
Visualizing the Anomalies

In [ ]:
```
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',

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.