```
In [1]: import os
   import numpy as np
   import pandas as pd
   import matplotlib.pyplot as plt
   import seaborn as sns
   import ipywidgets as widgets
   from IPython.display import display
   from sklearn.preprocessing import MinMaxScaler
   import tensorflow as tf
```

```
# Function to load and preprocess data
In [2]:
        def load and preprocess data(folder path):
            csv_files = [file for file in os.listdir(folder_path) if file.endswith('.c
            combined_data = pd.concat([pd.read_csv(os.path.join(folder_path, file)) fd
            combined_data['ts'] = pd.to_datetime(combined_data['ts'])
            combined_data.set_index('ts', inplace=True)
            return combined_data
        # Function to train LSTM model
        def train_lstm_model(combined_data, seq_length=30):
            data = combined_data[['humidity1', 'temperature1', 'humidity2', 'temperatu'
            scaler = MinMaxScaler()
            scaled_data = scaler.fit_transform(data)
            X, y = [], []
            for i in range(len(scaled_data) - seq_length):
                X.append(scaled_data[i:i + seq_length])
                y.append(scaled_data[i + seq_length])
            X, y = np.array(X), np.array(y)
            train_size = int(len(X) * 0.8)
            X_train, X_test = X[:train_size], X[train_size:]
            y_train, y_test = y[:train_size], y[train_size:]
            model = tf.keras.models.Sequential([
                tf.keras.layers.LSTM(units=50, return_sequences=True, input_shape=(X_t
                tf.keras.layers.Dropout(0.2),
                tf.keras.layers.LSTM(units=50, return_sequences=False),
                tf.keras.layers.Dropout(0.2),
                tf.keras.layers.Dense(units=16),
                tf.keras.layers.Dense(units=len(data.columns))
            model.compile(optimizer='adam', loss='mse')
            history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation
            mse = model.evaluate(X_test, y_test)
            predictions = model.predict(X test)
            predictions = scaler.inverse transform(predictions)
            y_test = scaler.inverse_transform(y_test)
            return predictions, y_test, history
        # Function to update plots based on user input
        def update plots(folder path):
            combined data = load and preprocess data(folder path)
            predictions, y_test, history = train_lstm_model(combined_data)
            plt.figure(figsize=(12, 6))
            plt.plot(y_test, label='Actual')
            plt.plot(predictions, label='Predicted')
            plt.xlabel('Time')
            plt.ylabel('Temperature')
            plt.title('Actual vs Predicted Temperature')
            plt.legend()
            plt.show()
        # Widget for folder path input
        folder path widget = widgets.Text(value=r'D:\Training Dataset\Training Dataset
        # Button to trigger update
```

```
update_button = widgets.Button(description='Update Plots')

# Event handler for button click
def on_button_click(b):
    folder_path = folder_path_widget.value
        update_plots(folder_path)

update_button.on_click(on_button_click)

# Display widgets
display(folder_path_widget)
display(update_button)
```

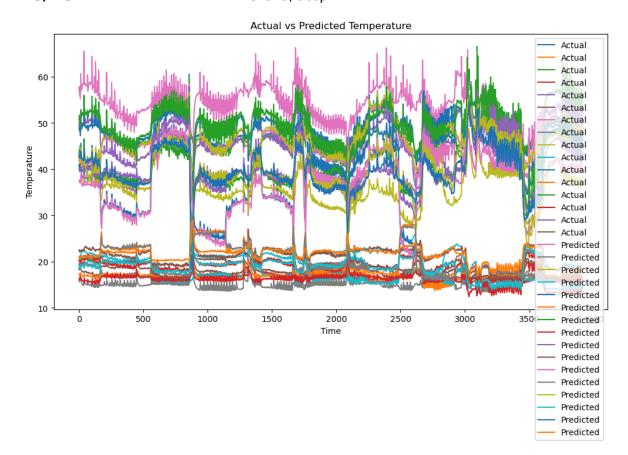
Folder Path:

D:\Training Dataset\Training Datas

**Update Plots** 

C:\Users\lenovo\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. Wh en using Sequential models, prefer using an `Input(shape)` object as the firs t layer in the model instead.

```
super().__init__(**kwargs)
```



C:\Users\lenovo\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. Wh en using Sequential models, prefer using an `Input(shape)` object as the firs t layer in the model instead.

super().\_\_init\_\_(\*\*kwargs)

134/134 — 1s 4ms/step - loss: 0.0011 134/134 — 1s 5ms/step

