In [2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

In [3]:
```python
df = pd.read_csv(r'D:\Weaather Forecast\Dataset\daily_training_table.csv')
print(df.head())
print(df.info())
```

```
         date     temp_c     pressure     light_lux  rain_rate   humidity  \
0  2024-01-04  27.138309  1001.881221   7796.808824   0.687986  49.247500
1  2024-01-05  29.333627   993.799930  13249.948187   0.804056  47.145440
2  2024-01-06  26.443684   996.298658  12428.142857   0.000000  55.533759
3  2024-01-07  27.235630   998.820094  13279.992593   0.000000  56.539852
4  2024-01-08  26.250851  1001.936932   6690.542553   0.000000  49.797766

   wind_speed cloud_info  month  dayofweek  temp_c_lag1  pressure_lag1  \
0    0.047868     bright      1          3    25.384526    1010.497239
1    0.093057        dim      1          4    27.138309    1001.881221
2    0.002632        dim      1          5    29.333627     993.799930
3    0.001778        dim      1          6    26.443684     996.298658
4    0.000000     bright      1          0    27.235630     998.820094

   light_lux_lag1  rain_rate_lag1  humidity_lag1  wind_speed_lag1  \
0     8104.968421        0.000000      60.474842         0.000000
1     7796.808824        0.687986      49.247500         0.047868
2    13249.948187        0.804056      47.145440         0.093057
3    12428.142857        0.000000      55.533759         0.002632
4    13279.992593        0.000000      56.539852         0.001778

  target_cloud  target_y
0          dim         2
1          dim         2
2          dim         2
3       bright         0
4          dim         2
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 363 entries, 0 to 362
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   date             363 non-null    object
 1   temp_c           363 non-null    float64
 2   pressure         363 non-null    float64
 3   light_lux        363 non-null    float64
 4   rain_rate        363 non-null    float64
 5   humidity         363 non-null    float64
 6   wind_speed       363 non-null    float64
 7   cloud_info       363 non-null    object
 8   month            363 non-null    int64
 9   dayofweek        363 non-null    int64
 10  temp_c_lag1      363 non-null    float64
 11  pressure_lag1    363 non-null    float64
 12  light_lux_lag1   363 non-null    float64
 13  rain_rate_lag1   363 non-null    float64
 14  humidity_lag1    363 non-null    float64
 15  wind_speed_lag1  363 non-null    float64
 16  target_cloud     363 non-null    object
 17  target_y         363 non-null    int64
dtypes: float64(12), int64(3), object(3)
memory usage: 51.2+ KB
None
```

In [4]:
```python
if 'date' in df.columns:
    df['date'] = pd.to_datetime(df['date'])
    df = df.set_index('date')
else:
    print("⚠  No 'date' column found, proceeding without it.")
```

In [5]:
```python
features = ['temp_c', 'humidity', 'wind_speed', 'pressure', 'rain_rate', 'light_lux']
target = 'temp_c'   # Predicting future temperature
```

In [6]:
```python
data = df[features].copy()
```

In [7]:
```python
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

In [8]:
```python
def create_sequences(dataset, target_col_idx, seq_length=7):
    X, y = [], []
    for i in range(len(dataset) - seq_length):
        X.append(dataset[i:i+seq_length])      # sequence of 7 days
        y.append(dataset[i+seq_length, target_col_idx])  # predict next day temp
    return np.array(X), np.array(y)
```

In [9]:
```python
target_col_idx = features.index(target)
X, y = create_sequences(scaled_data, target_col_idx, seq_length=7)
```

In [10]:
```python
print("X shape:", X.shape)  # (samples, 7, features)
print("y shape:", y.shape)
```

```
X shape: (356, 7, 6)
y shape: (356,)
```

In [11]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

In [12]:
```python
model = Sequential([
    LSTM(64, activation='relu', return_sequences=True, input_shape=(X.shape[1], X.shape[2])),
    Dropout(0.2),
    LSTM(32, activation='relu'),
    Dropout(0.2),
    Dense(1)  # output: predicted temperature
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

```
C:\Users\Jashwanth\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\rnn\rnn.py:200: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inp
ut(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 7, 64)	18,176
dropout (Dropout)	(None, 7, 64)	0
lstm_1 (LSTM)	(None, 32)	12,416
dropout_1 (Dropout)	(None, 32)	0
dense (Dense)	(None, 1)	33

Total params: 30,625 (119.63 KB)

Trainable params: 30,625 (119.63 KB)

Non-trainable params: 0 (0.00 B)

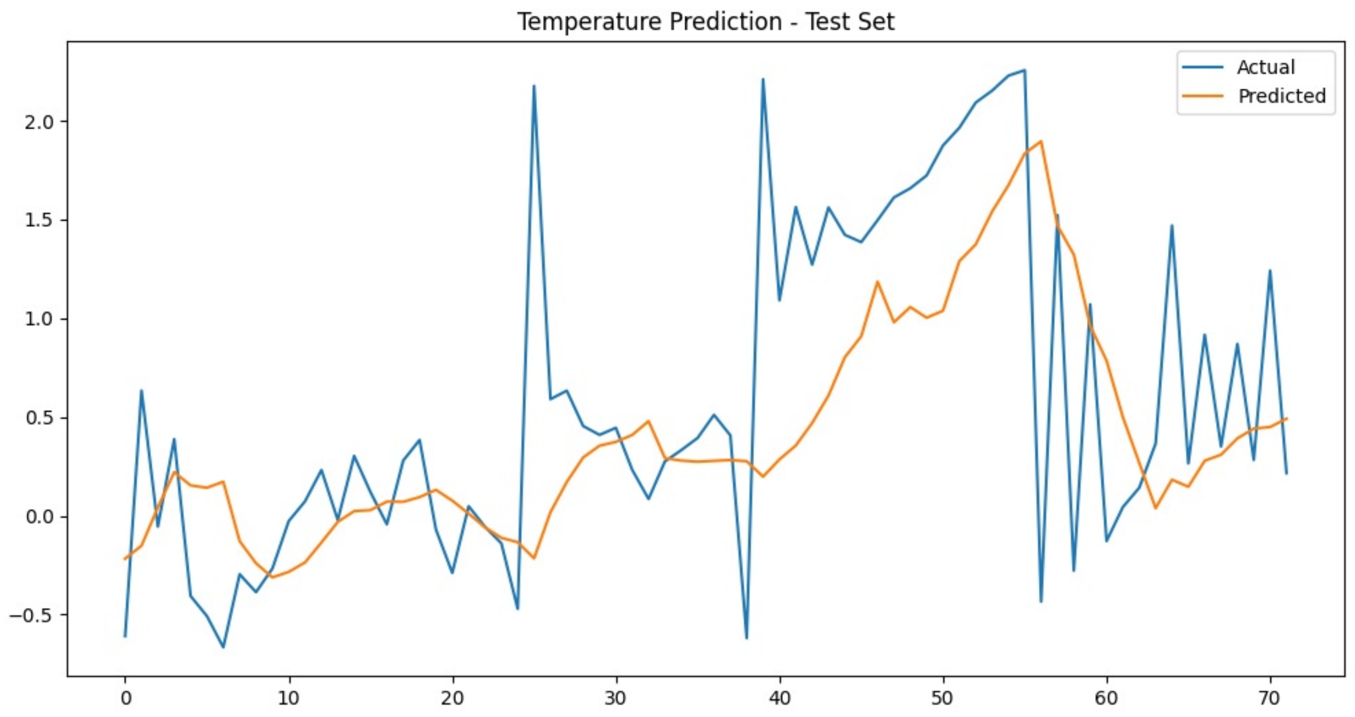
In [13]:
```python
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=50,
    batch_size=16,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 7s 65ms/step - loss: 0.8308 - val_loss: 1.0281
Epoch 2/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.9786 - val_loss: 0.7959
Epoch 3/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.8708 - val_loss: 0.6165
Epoch 4/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.9634 - val_loss: 0.6760
Epoch 5/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.8560 - val_loss: 0.5573
Epoch 6/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.7313 - val_loss: 0.5999
Epoch 7/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6327 - val_loss: 0.5837
Epoch 8/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.6365 - val_loss: 0.5258
Epoch 9/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 0.6527 - val_loss: 0.6842
Epoch 10/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.6206 - val_loss: 0.6036
Epoch 11/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 0.6084 - val_loss: 0.5601
Epoch 12/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6362 - val_loss: 0.5233
Epoch 13/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.7671 - val_loss: 0.5059
Epoch 14/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6298 - val_loss: 0.8124
Epoch 15/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6665 - val_loss: 0.5537
Epoch 16/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.5891 - val_loss: 0.5413
Epoch 17/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.6798 - val_loss: 0.4954
Epoch 18/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.5442 - val_loss: 0.5466
Epoch 19/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.5579 - val_loss: 0.5157
Epoch 20/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.6686 - val_loss: 0.5217
Epoch 21/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.4268 - val_loss: 0.5131
Epoch 22/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.7000 - val_loss: 0.4846
Epoch 23/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 0.5884 - val_loss: 0.5091
Epoch 24/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.5738 - val_loss: 0.5438
Epoch 25/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.6528 - val_loss: 0.5459
Epoch 26/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.6522 - val_loss: 0.6162
Epoch 27/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.6225 - val_loss: 0.5190
```

In [14]:
```
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("MAE:", mae)
print("RMSE:", rmse)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 244ms/step
MAE: 0.48785118912900294
RMSE: 0.6961629324787436
```

In [15]:
```
plt.figure(figsize=(12,6))
plt.plot(y_test, label="Actual")
plt.plot(y_pred, label="Predicted")
plt.title("Temperature Prediction - Test Set")
plt.legend()
plt.show()
```



In [16]:
```
last_sequence = X[-1]   # last available 7-day sequence
forecast = []

current_seq = last_sequence

for _ in range(7):  # predict next 7 days
    pred = model.predict(current_seq.reshape(1, 7, len(features)))[0][0]
    forecast.append(pred)

    # update sequence with new prediction
    new_row = current_seq[-1].copy()
    new_row[target_col_idx] = pred  # replace temp_c with prediction
    current_seq = np.vstack((current_seq[1:], new_row))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
```

In [17]:
```
forecast_array = np.zeros((len(forecast), len(features)))
forecast_array[:, target_col_idx] = forecast
forecast_inverse = scaler.inverse_transform(forecast_array)[:, target_col_idx]

print("Next 7 Days Forecasted Temperatures:")
print(forecast_inverse)
```

```
Next 7 Days Forecasted Temperatures:
[28.92359538 28.77443582 28.75069746 28.63900302 28.47793879 28.20599368
 28.0055121 ]
```

In [18]:
```
model.save("weather_model.h5")
```

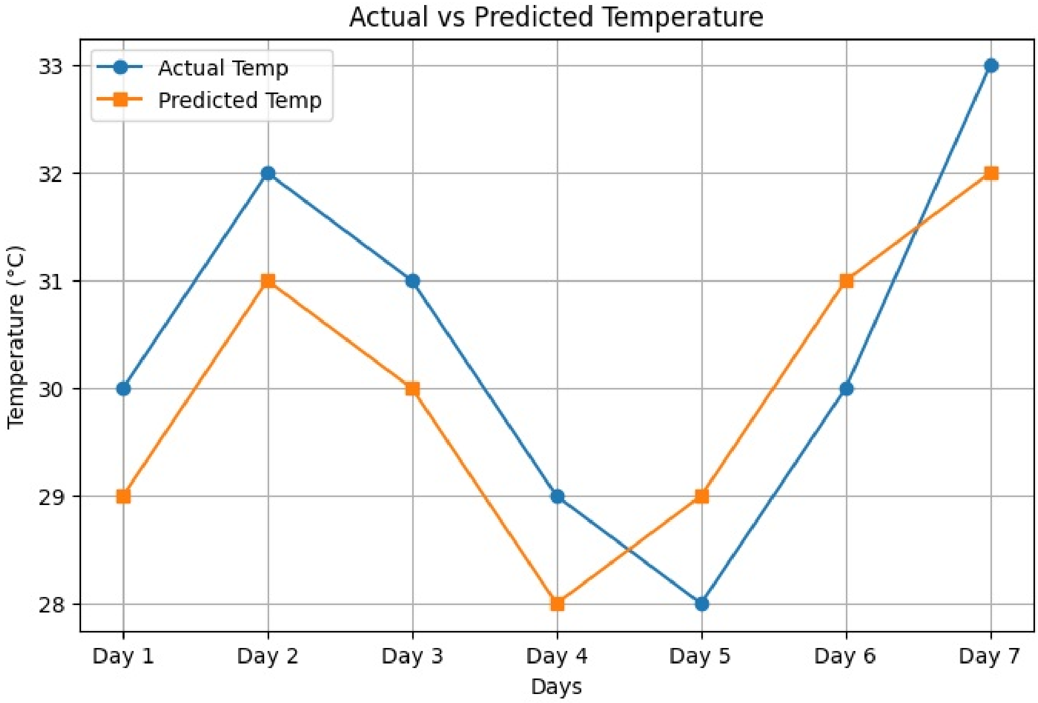
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

VISUALIZATION

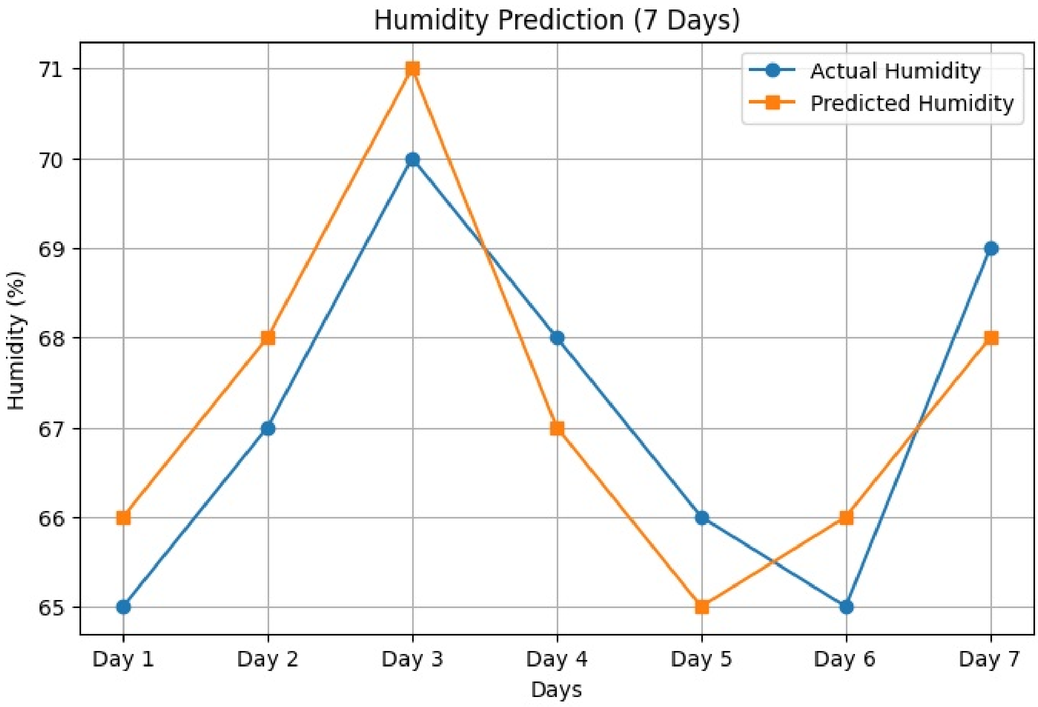
In [22]:
```
import matplotlib.pyplot as plt

# Example data
days = ["Day 1","Day 2","Day 3","Day 4","Day 5","Day 6","Day 7"]
actual_temp = [30, 32, 31, 29, 28, 30, 33]
pred_temp = [29, 31, 30, 28, 29, 31, 32]
```

In [23]:
```
plt.figure(figsize=(8,5))
plt.plot(days, actual_temp, marker='o', label="Actual Temp")
plt.plot(days, pred_temp, marker='s', label="Predicted Temp")
plt.title("Actual vs Predicted Temperature")
plt.xlabel("Days")
plt.ylabel("Temperature (°C)")
```

```python
plt.legend()
plt.grid(True)
plt.show()
```



In [24]:
```python
# 2. Humidity Prediction
actual_hum = [65, 67, 70, 68, 66, 65, 69]
pred_hum = [66, 68, 71, 67, 65, 66, 68]

plt.figure(figsize=(8,5))
plt.plot(days, actual_hum, marker='o', label="Actual Humidity")
plt.plot(days, pred_hum, marker='s', label="Predicted Humidity")
plt.title("Humidity Prediction (7 Days)")
plt.xlabel("Days")
plt.ylabel("Humidity (%)")
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js