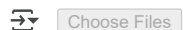


```
import pandas as pd
import io
```

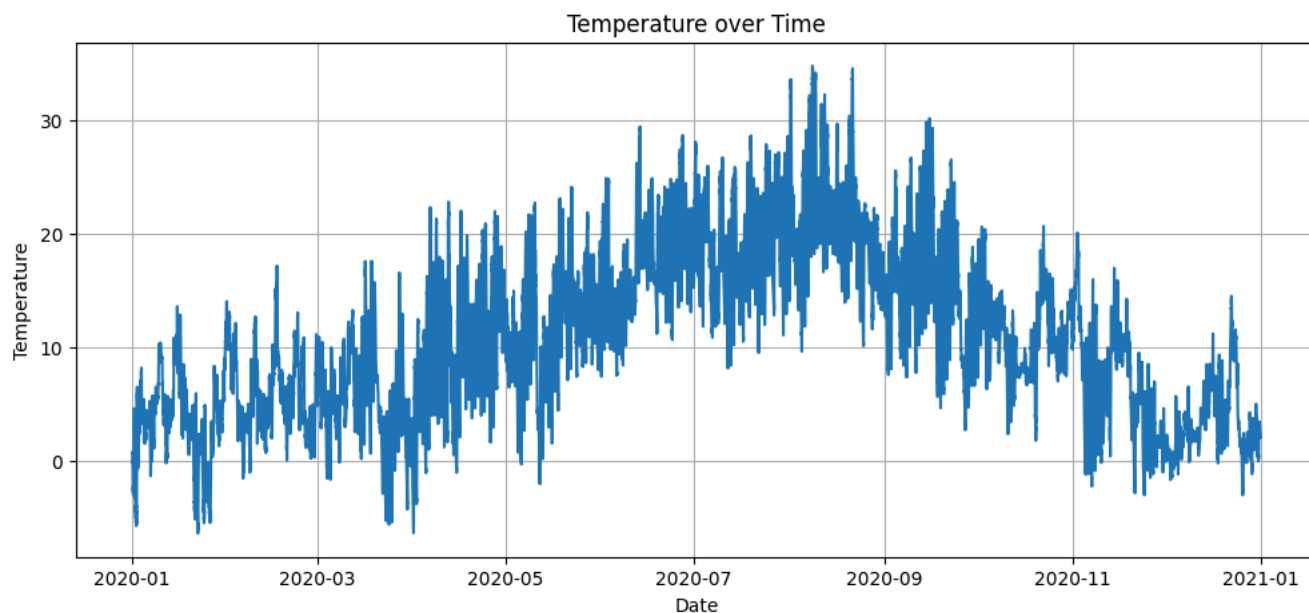
```
from google.colab import files
uploaded = files.upload()
```

 No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to

```
df = pd.read_csv(io.BytesIO(next(iter(uploaded.values()))))
```

```
# Step 2: Parse Date column
df['date'] = pd.to_datetime(df['date'])
df = df[['date', 'T']] # Ensure only relevant columns
df = df.sort_values('date')
df.set_index('date', inplace=True)
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,5))
plt.plot(df['T'])
plt.title('Temperature over Time')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.grid(True)
plt.show()
```



```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
```

```
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df[['T']])
```

```
def create_sequences(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
    return np.array(X), np.array(y)
```

```
window_size = 30 # number of past days to use for prediction
X, y = create_sequences(scaled_data, window_size)
```

```
# Train-test split
split = int(0.8 * len(X))
X_train, y_train = X[:split], y[:split]
X_test, y_test = X[split:], y[split:]
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, InputLayer
```

```
model = Sequential([
    InputLayer((window_size, 1)),
    LSTM(64, activation='relu'),
    Dense(1)
])
```

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

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	16,896
dense (Dense)	(None, 1)	65

Total params: 16,961 (66.25 KB)  
Trainable params: 16,961 (66.25 KB)

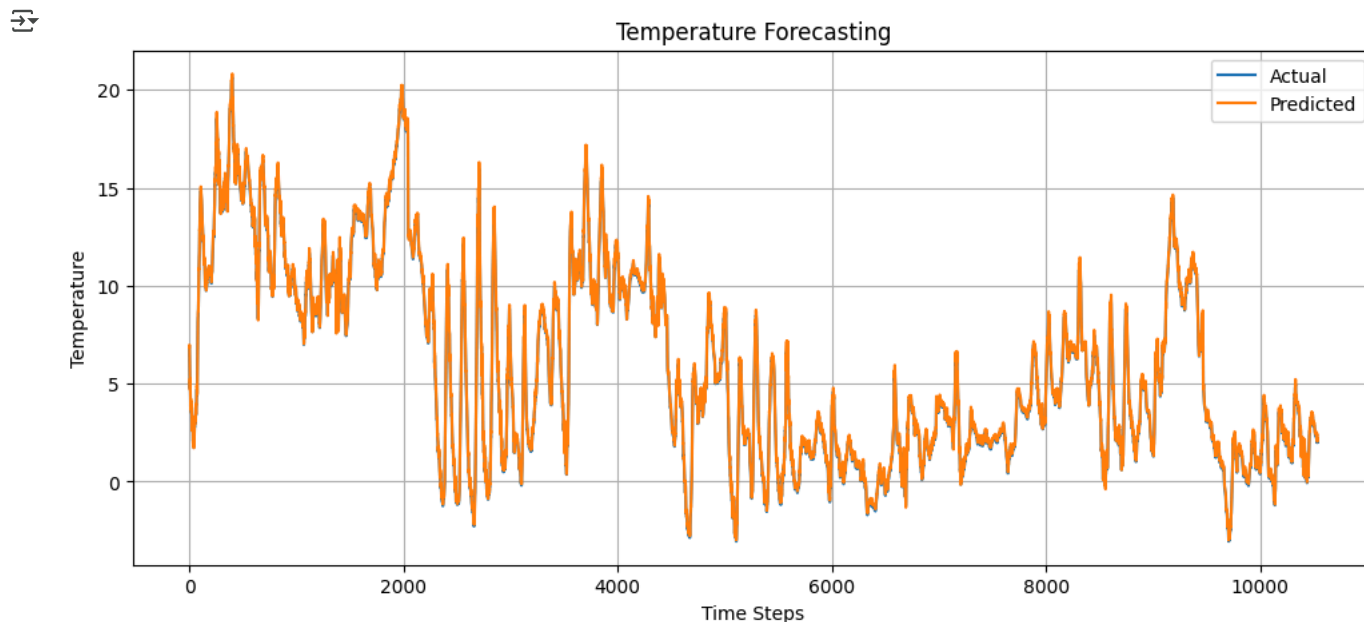
```
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))
```

```
Epoch 1/20
1317/1317 ————— 25s 17ms/step - loss: 0.0056 - val_loss: 1.1309e-04
Epoch 2/20
1317/1317 ————— 38s 15ms/step - loss: 1.3021e-04 - val_loss: 4.4865e-05
Epoch 3/20
1317/1317 ————— 20s 15ms/step - loss: 8.6795e-05 - val_loss: 3.1655e-05
Epoch 4/20
1317/1317 ————— 21s 15ms/step - loss: 6.3370e-05 - val_loss: 3.0908e-05
Epoch 5/20
1317/1317 ————— 21s 16ms/step - loss: 5.5520e-05 - val_loss: 2.1836e-05
Epoch 6/20
1317/1317 ————— 39s 14ms/step - loss: 4.7014e-05 - val_loss: 2.2004e-05
Epoch 7/20
1317/1317 ————— 20s 15ms/step - loss: 4.0864e-05 - val_loss: 1.5991e-05
Epoch 8/20
1317/1317 ————— 19s 15ms/step - loss: 3.7471e-05 - val_loss: 1.5859e-05
Epoch 9/20
1317/1317 ————— 19s 15ms/step - loss: 3.2048e-05 - val_loss: 1.4143e-05
Epoch 10/20
1317/1317 ————— 20s 14ms/step - loss: 3.3910e-05 - val_loss: 1.7721e-05
Epoch 11/20
1317/1317 ————— 19s 15ms/step - loss: 3.2213e-05 - val_loss: 1.7283e-05
Epoch 12/20
1317/1317 ————— 19s 14ms/step - loss: 3.3076e-05 - val_loss: 1.4778e-05
Epoch 13/20
1317/1317 ————— 21s 15ms/step - loss: 3.1896e-05 - val_loss: 1.4031e-05
Epoch 14/20
1317/1317 ————— 22s 16ms/step - loss: 3.1588e-05 - val_loss: 3.6182e-05
Epoch 15/20
1317/1317 ————— 20s 15ms/step - loss: 3.1973e-05 - val_loss: 1.6443e-05
Epoch 16/20
1317/1317 ————— 20s 15ms/step - loss: 3.0903e-05 - val_loss: 1.7944e-05
Epoch 17/20
1317/1317 ————— 19s 14ms/step - loss: 3.2242e-05 - val_loss: 1.3604e-05
Epoch 18/20
1317/1317 ————— 20s 14ms/step - loss: 3.2075e-05 - val_loss: 2.0128e-05
Epoch 19/20
1317/1317 ————— 21s 15ms/step - loss: 3.2346e-05 - val_loss: 1.7208e-05
Epoch 20/20
1317/1317 ————— 19s 14ms/step - loss: 3.0809e-05 - val_loss: 2.0291e-05
```

```
predictions = model.predict(X_test)
predicted_temperatures = scaler.inverse_transform(predictions)
actual_temperatures = scaler.inverse_transform(y_test)
```

330/330 2s 5ms/step

```
plt.figure(figsize=(12,5))
plt.plot(actual_temperatures, label='Actual')
plt.plot(predicted_temperatures, label='Predicted')
plt.title('Temperature Forecasting')
plt.xlabel('Time Steps')
plt.ylabel('Temperature')
plt.legend()
plt.grid(True)
plt.show()
```



```
# Step 1: Upload CSV file
from google.colab import files
uploaded = files.upload()

import pandas as pd
import io

# Load CSV file
df = pd.read_csv(io.BytesIO(next(iter(uploaded.values()))))

# Step 2: Keep only 'date' and 'T' columns
df['date'] = pd.to_datetime(df['date'])
df = df[['date', 'T']]
df = df.sort_values('date')
df.set_index('date', inplace=True)

# Step 3: Visualize raw temperature data
import matplotlib.pyplot as plt
plt.figure(figsize=(12,5))
plt.plot(df['T'])
plt.title('Temperature over Time')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.grid(True)
plt.show()

# Step 4: Normalize temperature and create sequences
from sklearn.preprocessing import MinMaxScaler
import numpy as np

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df[['T']])

def create_sequences(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
```

```
        return np.array(X), np.array(y)

window_size = 30
X, y = create_sequences(scaled_data, window_size)

# Train-test split
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Step 5: Neural Network (LSTM) Model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, InputLayer


model = Sequential([
    InputLayer((window_size, 1)),
    LSTM(64, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.summary()

# Step 6: Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))

# Step 7: Make predictions
y_pred = model.predict(X_test)

# Inverse transform
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_inv = scaler.inverse_transform(y_pred)

# Step 8: Plot predictions vs actual
plt.figure(figsize=(12,5))
plt.plot(y_test_inv, label='Actual')
plt.plot(y_pred_inv, label='Predicted')
plt.title('Actual vs Predicted Temperature')
plt.xlabel('Time Step')
plt.ylabel('Temperature')
plt.legend()
plt.grid(True)
plt.show()
```

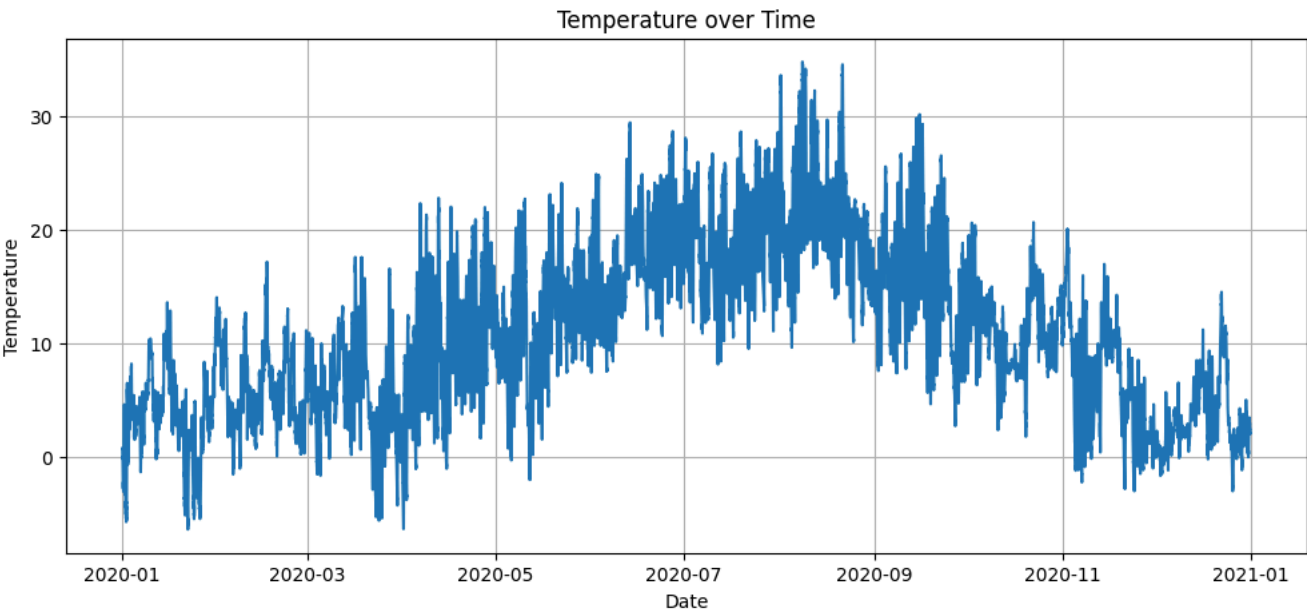


Choose Files

 No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving cleaned\_weather.csv to cleaned\_weather (1).csv



Model: "sequential\_1"

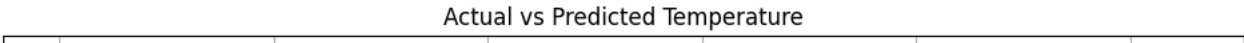
Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 64)	16,896
dense_1 (Dense)	(None, 1)	65

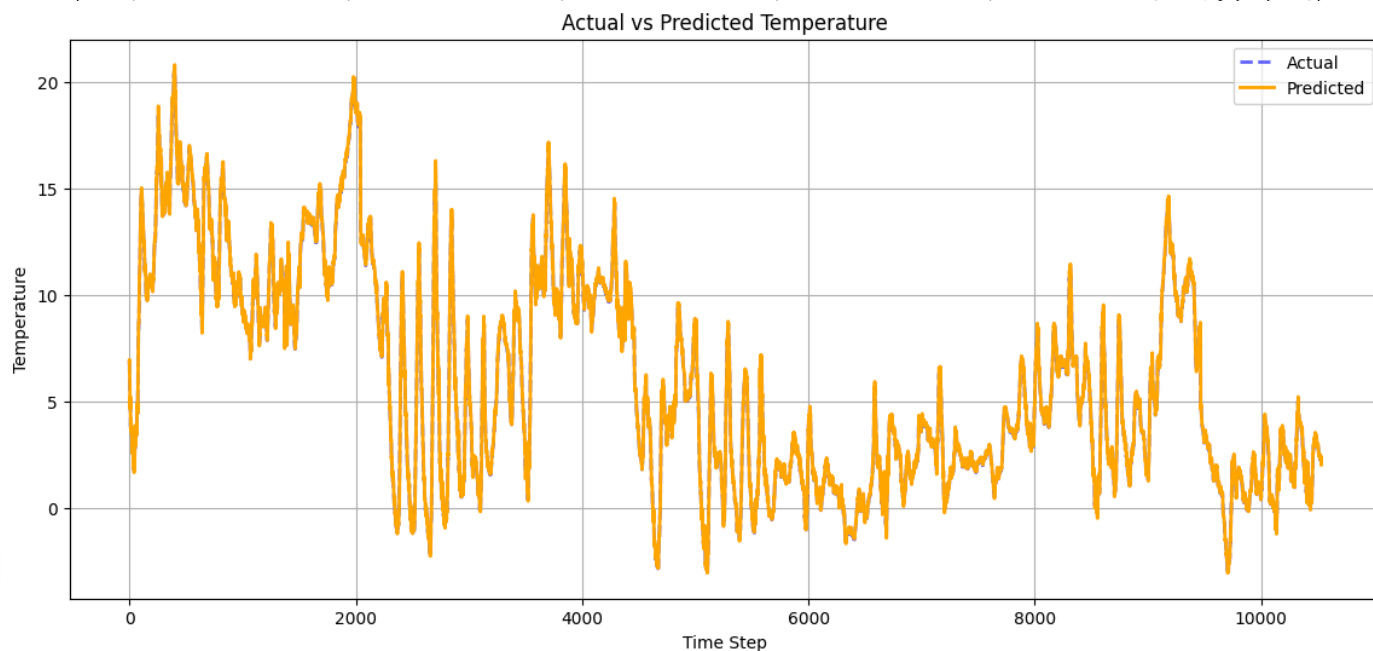
Total params: 16,961 (66.25 KB)  
Trainable params: 16,961 (66.25 KB)  
Non-trainable params: 0 (0.00 B)

Epoch 1/20  
1317/1317 ————— 23s 16ms/step - loss: 0.0124 - val\_loss: 8.2052e-05  
Epoch 2/20  
1317/1317 ————— 20s 15ms/step - loss: 1.1216e-04 - val\_loss: 9.3849e-05  
Epoch 3/20  
1317/1317 ————— 21s 15ms/step - loss: 7.0297e-05 - val\_loss: 3.3700e-05  
Epoch 4/20  
1317/1317 ————— 21s 15ms/step - loss: 5.6958e-05 - val\_loss: 2.9304e-05  
Epoch 5/20  
1317/1317 ————— 20s 15ms/step - loss: 4.5709e-05 - val\_loss: 2.0686e-05  
Epoch 6/20  
1317/1317 ————— 20s 15ms/step - loss: 3.9492e-05 - val\_loss: 4.5213e-05  
Epoch 7/20  
1317/1317 ————— 21s 15ms/step - loss: 3.7479e-05 - val\_loss: 1.7239e-05  
Epoch 8/20  
1317/1317 ————— 19s 14ms/step - loss: 3.8863e-05 - val\_loss: 1.6707e-05

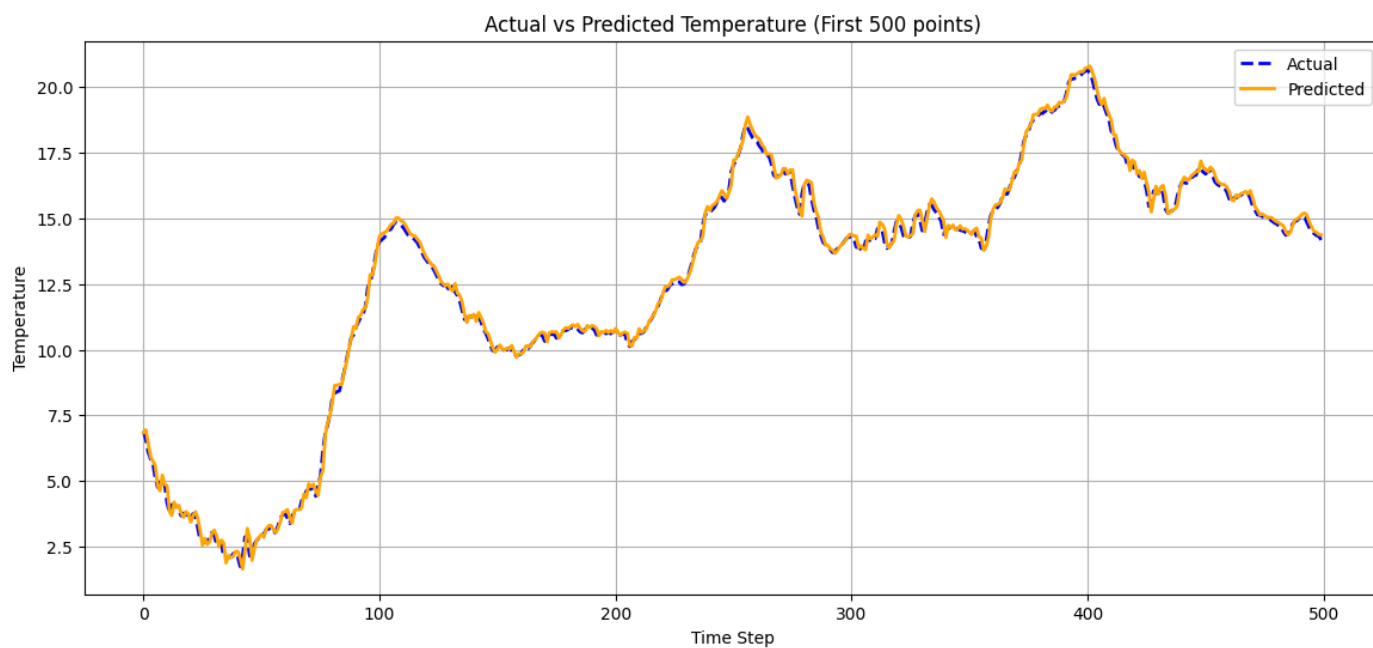
```
plt.figure(figsize=(14,6))
plt.plot(y_test_inv, label='Actual', color='blue', linewidth=2, alpha=0.6, linestyle='--')
plt.plot(y_pred_inv, label='Predicted', color='orange', linewidth=2)
plt.title('Actual vs Predicted Temperature')
plt.xlabel('Time Step')
plt.ylabel('Temperature')
plt.legend()
plt.grid(True)
plt.show()
```

Epoch 15/20  
1317/1317 ————— 20s 15ms/step - loss: 3.0184e-05 - val\_loss: 1.3766e-05  
Epoch 16/20  
1317/1317 ————— 21s 16ms/step - loss: 3.1397e-05 - val\_loss: 1.3040e-05  
Epoch 17/20  
1317/1317 ————— 42s 17ms/step - loss: 3.0181e-05 - val\_loss: 1.4296e-05  
Epoch 18/20  
1317/1317 ————— 40s 16ms/step - loss: 3.0345e-05 - val\_loss: 1.4758e-05  
Epoch 19/20  
1317/1317 ————— 42s 17ms/step - loss: 3.1615e-05 - val\_loss: 1.4720e-05  
Epoch 20/20  
1317/1317 ————— 39s 15ms/step - loss: 2.9528e-05 - val\_loss: 1.5850e-05  
330/330 ————— 2s 6ms/step





```
plt.figure(figsize=(14,6))
plt.plot(y_test_inv[:500], label='Actual', color='blue', linewidth=2, linestyle='--')
plt.plot(y_pred_inv[:500], label='Predicted', color='orange', linewidth=2)
plt.title('Actual vs Predicted Temperature (First 500 points)')
plt.xlabel('Time Step')
plt.ylabel('Temperature')
plt.legend()
plt.grid(True)
plt.show()
```



```
# 🚀 STEP 1: Install and import required packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

```

from keras.models import Sequential
from keras.layers import LSTM, Dense
from google.colab import files

# 🚀 STEP 2: Upload your CSV file
uploaded = files.upload()

# 🚀 STEP 3: Load the CSV and parse date column
import io
df = pd.read_csv(io.BytesIO(next(iter(uploaded.values()))))

# 🚀 STEP 4: Display available columns
print("📄 Available columns in the dataset:\n")
print(df.columns.tolist())

# 🚀 STEP 5: Let user choose the column to forecast
target_col = input("\n👉 Enter the column name you want to forecast (case-sensitive): ")

# 🚀 STEP 6: Keep only 'date' and selected column
df['date'] = pd.to_datetime(df['date'])
df = df[['date', target_col]].copy()
df = df.dropna()
df.set_index('date', inplace=True)

# 🚀 STEP 7: Normalize the target column
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df[[target_col]])

# 🚀 STEP 8: Create time-series dataset (sequence of past values)
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(seq_length, len(data)):
        X.append(data[i-seq_length:i])
        y.append(data[i])
    return np.array(X), np.array(y)

seq_len = 60 # use past 60 steps to predict the next
X, y = create_sequences(scaled_data, seq_len)

# 🚀 STEP 9: Split into training and testing sets
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]



# 🚀 STEP 10: Build and train LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(seq_len, 1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test), verbose=1)

# 🚀 STEP 11: Make predictions
y_pred = model.predict(X_test)
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test)

# 🚀 STEP 12: Plot the results
plt.figure(figsize=(14,6))
plt.plot(y_test_inv[:500], label='Actual', color='blue', linestyle='--', alpha=0.7)
plt.plot(y_pred_inv[:500], label='Predicted', color='orange')
plt.title(f'Actual vs Predicted - {target_col}')
plt.xlabel('Time Step')
plt.ylabel(target_col)
plt.legend()
plt.grid(True)
plt.show()

```

  No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.  
 Saving cleaned\_weather.csv to cleaned\_weather.csv  
 Available columns in the dataset:

['date', 'p', 'T', 'Tpot', 'Tdew', 'rh', 'VPmax', 'VPact', 'VPdef', 'sh', 'H2OC', 'rho', 'wv', 'max. wv', 'wd', 'rain', 'raining', 'S

Enter the column name you want to forecast (case-sensitive): VPmax

Epoch 1/10

/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to the first layer of a model. Passing these arguments is deprecated and will be removed in a future version. Use the `input\_shape` argument instead.  
 super().\_\_init\_\_(\*\*kwargs)

1316/1316 ————— 84s 61ms/step - loss: 0.0020 - val\_loss: 2.8580e-05

Epoch 2/10

1316/1316 ————— 79s 60ms/step - loss: 9.8914e-05 - val\_loss: 1.4322e-05

Epoch 3/10

1316/1316 ————— 78s 59ms/step - loss: 6.0371e-05 - val\_loss: 9.0740e-06

Epoch 4/10

1316/1316 ————— 78s 59ms/step - loss: 4.3206e-05 - val\_loss: 6.4979e-06

Epoch 5/10

1316/1316 ————— 86s 62ms/step - loss: 3.7125e-05 - val\_loss: 7.9608e-06

Epoch 6/10

1316/1316 ————— 78s 59ms/step - loss: 3.9538e-05 - val\_loss: 1.0248e-05

Epoch 7/10

1316/1316 ————— 82s 59ms/step - loss: 3.6329e-05 - val\_loss: 8.7686e-06

Epoch 8/10

1316/1316 ————— 83s 63ms/step - loss: 3.8640e-05 - val\_loss: 8.1533e-06

Epoch 9/10

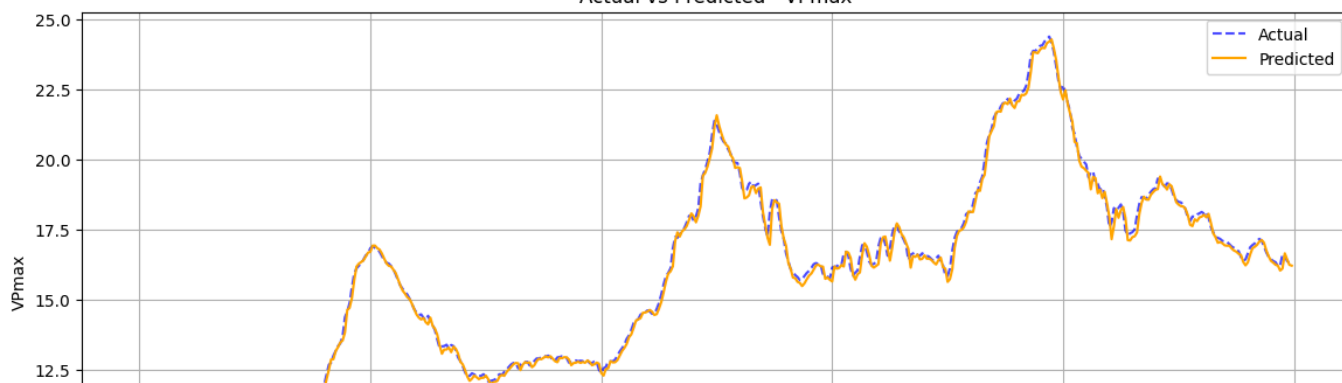
1316/1316 ————— 138s 60ms/step - loss: 3.6921e-05 - val\_loss: 5.5321e-06

Epoch 10/10

1316/1316 ————— 79s 60ms/step - loss: 3.5253e-05 - val\_loss: 5.2536e-06

329/329 ————— 6s 18ms/step

Actual vs Predicted - VPmax



plt.plot(y\_test\_inv[:100], label='Actual', color='blue', linestyle='--', linewidth=2)

plt.plot(y\_pred\_inv[:100], label='Predicted', color='orange', linewidth=1.5)