EX: 9 Develop neural network-based time series forecasting model.

Aim:

To develop an **neural network model** for forecasting the **rank trends** in a Google Trends dataset using time series analysis.

Procedure And Code :

◆ Step 1: Install and Import Required Libraries

```
# Install TensorFlow (if needed)
!pip install tensorflow --quiet

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

🔹 Step 2: Upload and Load the Dataset

```
from google.colab import files
uploaded = files.upload()  # Upload cleaned_weather.csv

# Load the CSV file
df = pd.read_csv("cleaned_weather.csv")
```

🔹 Step 3: Preprocess the Data

1. Convert date column to datetime
2. Sort by date
3. Set date as index
4. Select the target column (e.g. temperature 'T')
5. Normalize the values

```
# Convert date column to datetime
df['date'] = pd.to_datetime(df['date'])

# Sort and set date as index
df = df.sort_values('date')
df.set_index('date', inplace=True)
```

```
# Select the target column (e.g., 'T' for temperature)
target_column = 'T'


# Normalize using MinMaxScaler
scaler = MinMaxScaler()
df['T_scaled'] = scaler.fit_transform(df[[target_column]])
```

🔹 Step 4: Create Time Series Sequences

Use previous values (time steps) to predict the next one.

```
# Function to create input-output sequences
def create_dataset(data, time_steps=10):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i + time_steps])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)


# Create dataset
time_steps = 10
X, y = create_dataset(df['T_scaled'].values, time_steps)
```

```
# Reshape to 3D for LSTM: [samples, time_steps, features]

X = X.reshape((X.shape[0], X.shape[1], 1))
```

◆ Step 5: Train-Test Split

```
# Split data into training and testing sets (no shuffle for time series)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

◆ Step 6: Build and Train LSTM Model

```
# Build LSTM Model

model = Sequential([

    LSTM(50, activation='relu', return_sequences=True, input_shape=(time_steps, 1)),

    LSTM(50, activation='relu'),

    Dense(1)

])


# Compile model

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

```
# Train model

model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_data=(X_test, y_test), verbose=1)
```

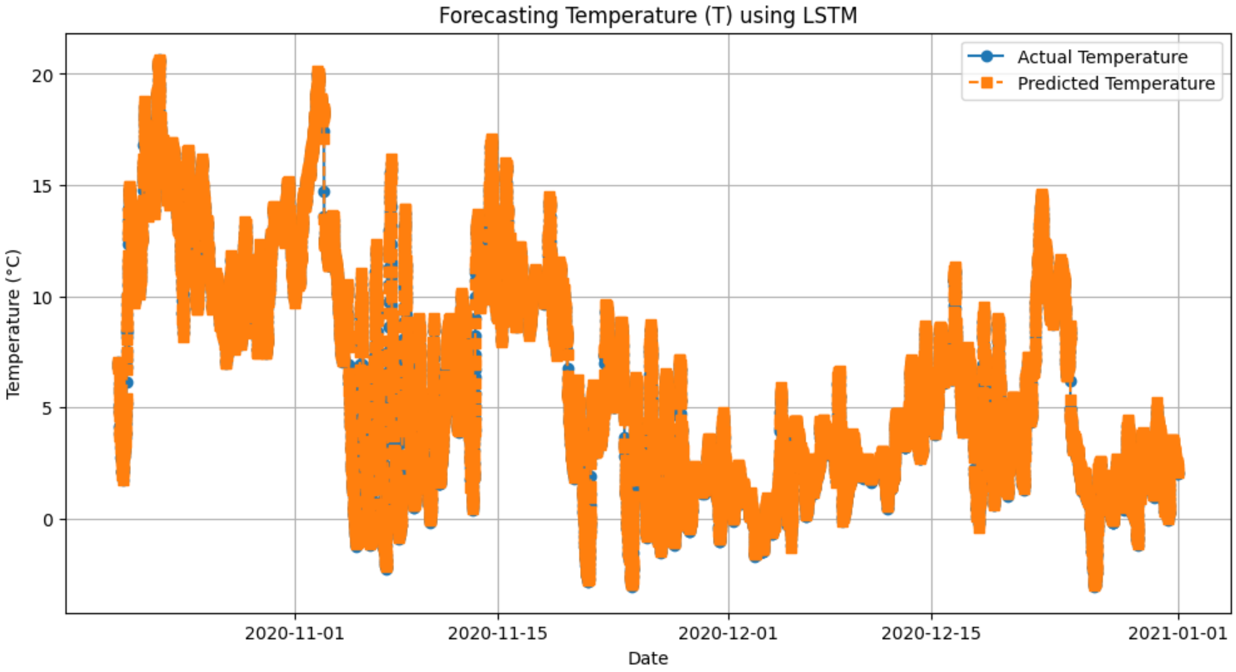
◆ Step 7: Make Predictions and Inverse Transform

```
# Predict on test data

y_pred = model.predict(X_test)


# Inverse transform to get actual values

y_pred_actual = scaler.inverse_transform(y_pred)

y_test_actual = scaler.inverse_transform(y_test.reshape(-1, 1))
```

◆ Step 8: Plot Actual vs Predicted Values

```
# Use last test index range for plotting

date_range = df.index[-len(y_test):]


plt.figure(figsize=(12, 6))

plt.plot(date_range, y_test_actual, label="Actual Temperature", marker='o')

plt.plot(date_range, y_pred_actual, label="Predicted Temperature",
linestyle='dashed', marker='s')

plt.title("Forecasting Temperature (T) using LSTM")
plt.xlabel("Date")

plt.ylabel("Temperature (°C)")

plt.legend()

plt.grid(True)

plt.show()
```

Output:



Result:

The program to develop a neural network based time series forecasting model has been successfully implemented