EX:No.9 **DATE:12/04/25**	**Develop neural network-based time series forecasting model.**

AIM:

To Develop neural network-based time series forecasting model.

ALGORITHM:

1. **Data Cleaning** – Loaded the dataset, parsed dates, fixed encoding issues, and selected only the Price column.
2. **Normalization** – Scaled price values between 0 and 1 using MinMaxScaler to improve neural network performance.
3. **Sequence Creation** – Created supervised learning format by using the previous 10 timesteps to predict the next one.
4. **Train-Test Split** – Split the dataset into 80% training and 20% testing sets.
5. **Model Building** – Built an LSTM model with one LSTM layer (50 units) and one Dense output layer.
6. **Model Training** – Trained the model using training data for 20 epochs with a batch size of 32.
7. **Prediction & Inverse Scaling** – Predicted future values and converted them back to original scale.
8. **Visualization** – Plotted actual vs predicted price values to evaluate model performance.

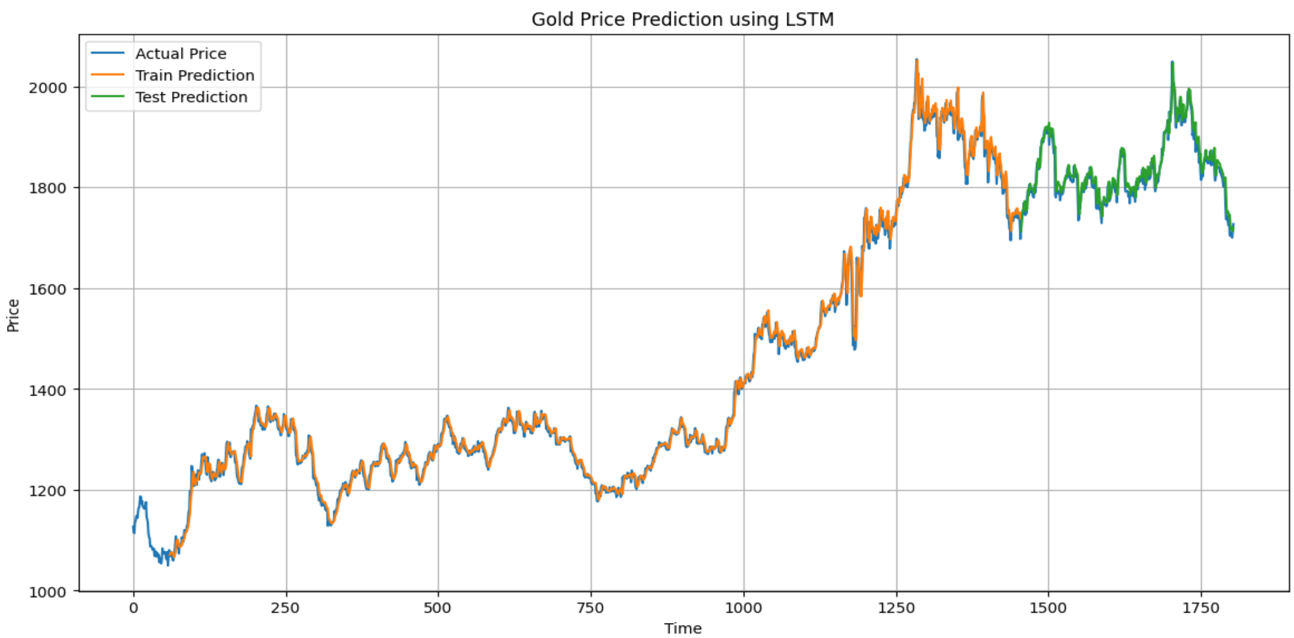
Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Load dataset
df = pd.read_csv('/content/gold_data (1).csv', parse_dates=['Date'])
df.set_index('Date', inplace=True)

# Select the 'Price' column
prices = df['Price'].values.reshape(-1,1)

# Scale the data
scaler = MinMaxScaler()
scaled_prices = scaler.fit_transform(prices)

# Prepare data for LSTM
def create_dataset(data, time_step=60):
```

```
    X, y = [], []
    for i in range(time_step, len(data)):
        X.append(data[i-time_step:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

time_step = 60  # using past 60 days to predict next day
X, y = create_dataset(scaled_prices, time_step)

# Reshape X to [samples, time steps, features]
X = X.reshape(X.shape[0], X.shape[1], 1)

# Split into train/test
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:]

# Build LSTM Model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X.shape[1],1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, batch_size=32, epochs=50)

# Predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)

# Inverse transform to get actual values
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
y_train_actual = scaler.inverse_transform(y_train.reshape(-1,1))
y_test_actual = scaler.inverse_transform(y_test.reshape(-1,1))

# Plotting results
plt.figure(figsize=(14,7))
# shift train predictions for plotting
train_plot = np.empty_like(scaled_prices)
train_plot[:, :] = np.nan
train_plot[time_step:train_size+time_step, :] = train_predict

# shift test predictions for plotting
test_plot = np.empty_like(scaled_prices)
test_plot[:, :] = np.nan
```

```
test_plot[train_size+time_step:, :] = test_predict

# Plot baseline and predictions
plt.plot(prices, label='Actual Price')
plt.plot(train_plot, label='Train Prediction')
plt.plot(test_plot, label='Test Prediction')
plt.title('Gold Price Prediction using LSTM')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.grid()
plt.show()
```

OUTPUT:



RESULT:

Thus, the program using the time series data implementation has been done successfully.