EX:No.9 **DATE:13/4/2025**	**Develop neural network-based time series forecasting model.**

AIM:

To Implement program to develop neural network-based time series forecasting model.

ALGORITHM:

 Load and Visualize Data

- Import dataset
- Plot time series to identify patterns or anomalies

 Normalize the Data

- Apply MinMaxScaler or StandardScaler
- Neural networks perform better with scaled inputs

 Convert to Supervised Format

- Create input-output pairs using sliding windows
- For example, use n past values to predict the next one

 Split into Train and Test Sets

- Typically 70–80% training, rest testing
- Maintain temporal order (no shuffling)

 Design Neural Network Architecture

- Use MLP (Dense layers) for basic models
- Use LSTM/GRU/CNN for sequence-aware models
- Input shape: (n_timesteps,)

 Compile the Model

- Define loss function (e.g., MSE)
- Choose optimizer (e.g., Adam)

 Train the Model

- Fit the model using training data
- Use validation split or a separate validation set
- Optionally apply EarlyStopping to avoid overfitting

 Make Predictions

- Predict on test set

- Inverse transform the output (if normalized)

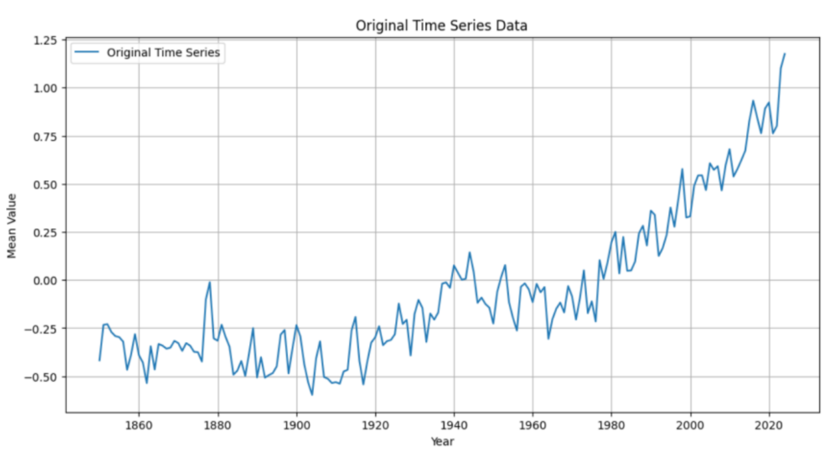
- **Evaluate and Visualize Results**
  - Use metrics like MAE, RMSE
  - Plot actual vs predicted values
  - Optionally plot residuals and forecast horizon

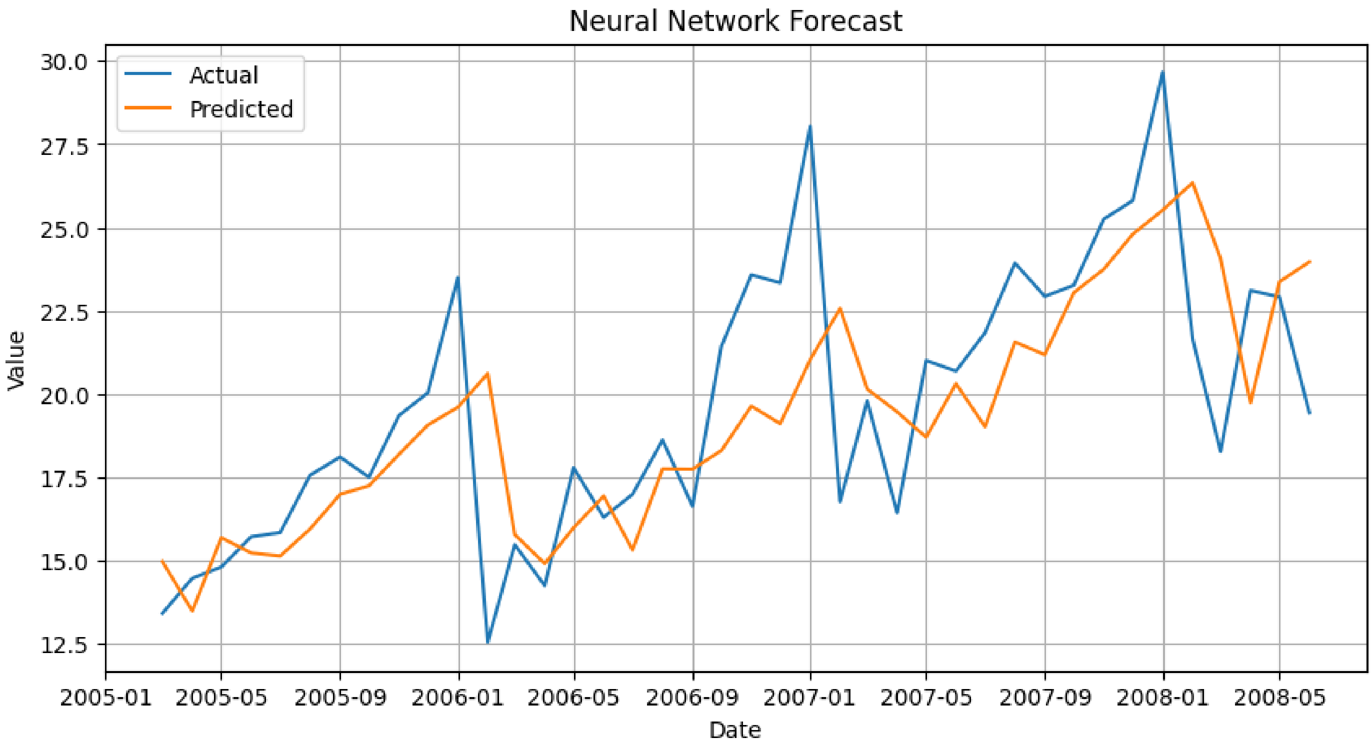
CODE:

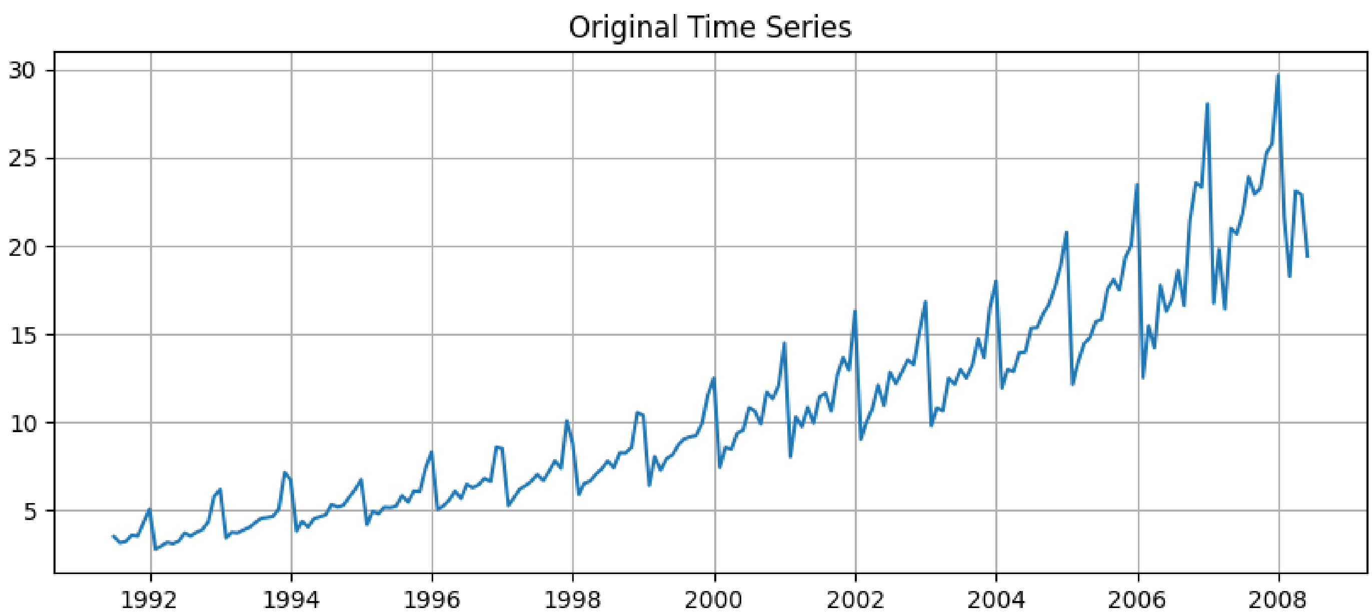
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping

# 1. Load dataset
data = pd.read_csv('sale.csv', parse_dates=['date'], index_col='date')
series = data['value'].values.reshape(-1, 1)

# 2. Normalize data
scaler = MinMaxScaler()
scaled_series = scaler.fit_transform(series)

# 3. Create supervised learning format
def create_sequences(data, n_steps):
    X, y = [], []
    for i in range(len(data) - n_steps):
        X.append(data[i:i+n_steps])
        y.append(data[i+n_steps])
    return np.array(X), np.array(y)

n_steps = 5
X, y = create_sequences(scaled_series, n_steps)

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

# 5. Build model
model = Sequential([
    Dense(64, activation='relu', input_shape=(n_steps,)),
```

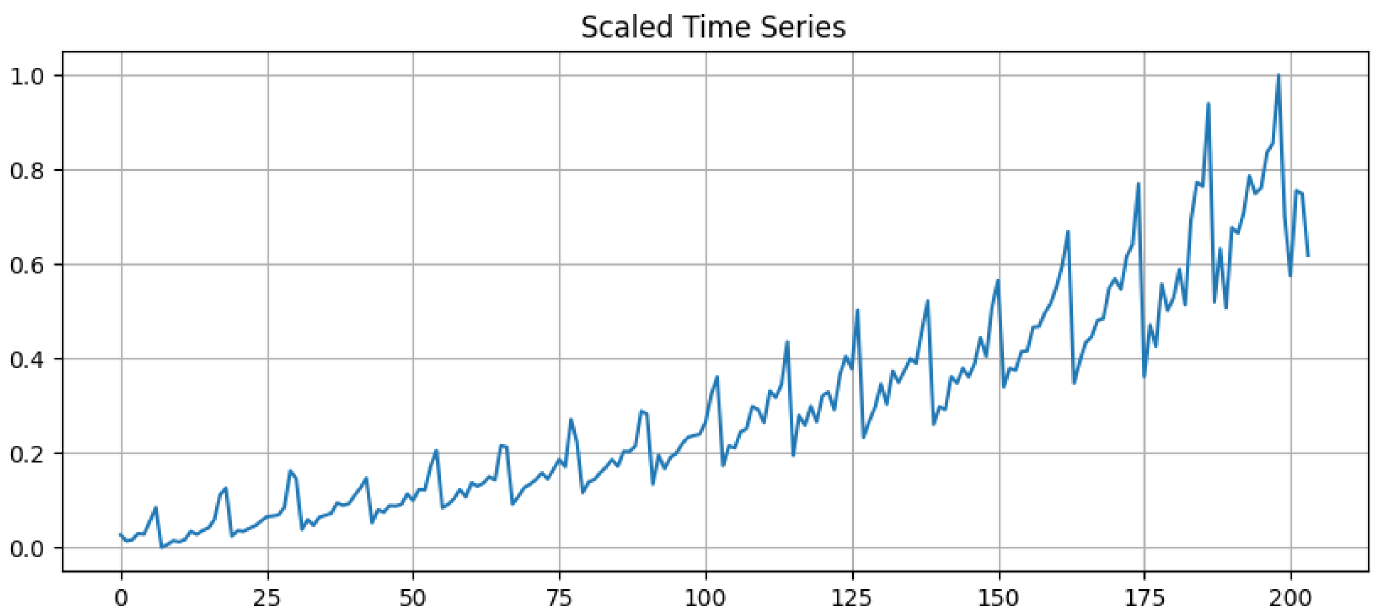
```
    Dense(64, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')

# 6. Train model
early_stop = EarlyStopping(patience=10, restore_best_weights=True)
model.fit(X_train.reshape(X_train.shape[0], -1), y_train,
          validation_data=(X_test.reshape(X_test.shape[0], -1), y_test),
          epochs=100, batch_size=8, callbacks=[early_stop], verbose=0)

# 7. Forecast
y_pred = model.predict(X_test.reshape(X_test.shape[0], -1))
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test)

# 8. Plot original vs predicted
plt.figure(figsize=(10, 5))
plt.plot(data.index[-len(y_test):], y_test_inv, label='Actual')
plt.plot(data.index[-len(y_test):], y_pred_inv, label='Predicted')
plt.title('Neural Network Forecast')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```

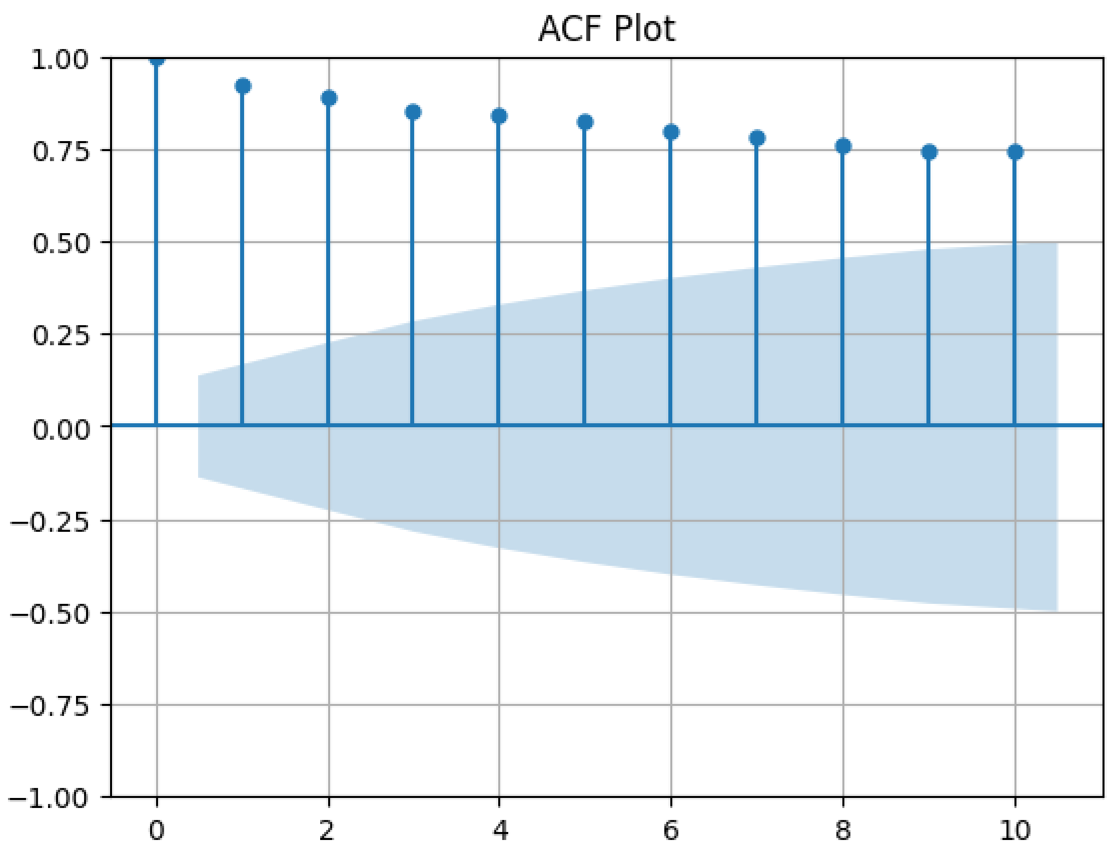
OUTPUT:

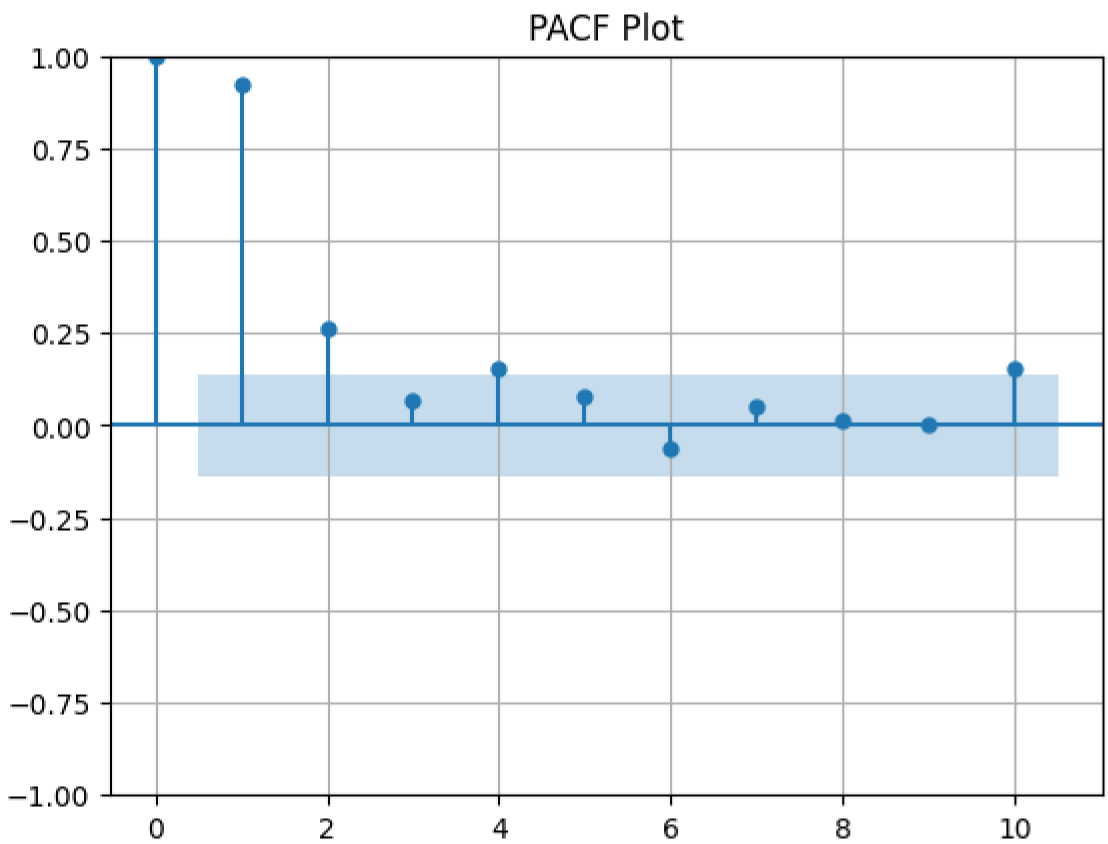



Neural Network Forecast
Actual
Predicted
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
Value
2005-01
2005-05
2005-09
2006-01
2006-05
2006-09
2007-01
2007-05
2007-09
2008-01
2008-05
Date


Original Time Series
30
25
20
15
10
5
1992
1994
1996
1998
2000
2002
2004
2006
2008


Scaled Time Series
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200


ACF Plot
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10



RESULT:

Thus the program has been completed and verified successfully.