EX:No.6 DATE:29/03/25	**Implement program to apply moving average smoothing for data preparation and time series forecasting**

AIM:

To implement a time series forecasting system for electric production data by applying moving average smoothing for data preparation and using the ARIMA model to predict future values. The goal is to enhance forecasting accuracy by reducing noise in the data and modeling the temporal patterns effectively.

ALGORITHM:

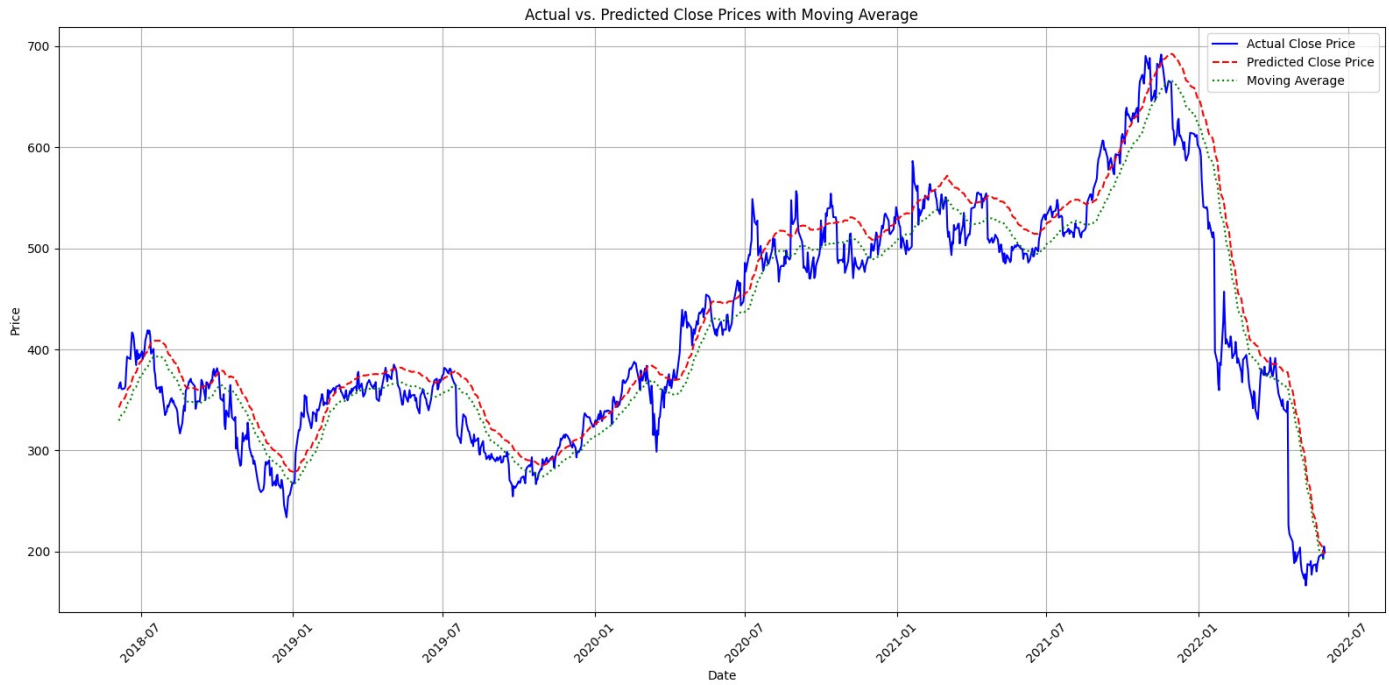
1. Import required libraries (pandas, numpy, matplotlib, statsmodels) for data handling, visualization, and modeling.
2. Load the electric production dataset, convert the date column to datetime format, and set it as the index.
3. Handle missing or duplicate data by filling missing values and removing duplicates if any.
4. Apply moving average smoothing on the target column using a 12-month rolling window to reduce short-term noise.
5. Check for stationarity using the Augmented Dickey-Fuller (ADF) test and apply differencing if the series is non-stationary.
6. Fit an ARIMA model to the processed data and forecast future values, then visualize the original, smoothed, and forecasted results using plots.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Load the data
df_nflx = pd.read_csv('NFLX (1).csv', parse_dates=['Date'])

# Calculate the 30-day moving average
df_nflx['MovingAverage'] = df_nflx['Close'].rolling(window=30).mean()

# Fill NaN values with the first valid observation
df_nflx['MovingAverage'] = df_nflx['MovingAverage'].fillna(method='bfill')

# Prepare features and target variable
X = df_nflx[['MovingAverage']]
y = df_nflx['Close']
```

```
    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False,
random_state=42)

    # Train the Linear Regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate the model
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)  # Calculate RMSE manually
    r2 = r2_score(y_test, y_pred)

    # Print the metrics
    print(f"Mean Absolute Error (MAE): {mae}")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"Root Mean Squared Error (RMSE): {rmse}")
    print(f"R-squared (R2): {r2}")

    # Visualize the results
    plt.figure(figsize=(16, 8))
    plt.plot(df_nflx['Date'][X_test.index], y_test, label='Actual Close Price', color='blue')
    plt.plot(df_nflx['Date'][X_test.index], y_pred, label='Predicted Close Price', color='red', linestyle='--
')
    plt.plot(df_nflx['Date'][X_test.index], X_test['MovingAverage'], label='Moving Average',
color='green', linestyle=':')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.title('Actual vs. Predicted Close Prices with Moving Average')
    plt.legend()
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
plt.show()
```

OUTPUT:



RESULT:

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