6.Implement program to apply moving average smoothing for data preparation and time series forecasting

### Aim:

The aim of this program is to apply moving average smoothing for data preparation and time series forecasting. Moving average smoothing helps to reduce noise and better visualize trends by averaging data points over a specified window (number of periods). This technique is commonly used for time series analysis to improve prediction accuracy and reveal underlying patterns.

**Procedure:**

1. **Import Required Libraries**: We need libraries like Pandas for data manipulation and analysis, Matplotlib for visualization, and NumPy for numerical operations.
2. **Load Data**: Import the time series data (real or synthetic data).
3. **Apply Moving Average Smoothing**: Use a moving average technique to smooth the data.
4. **Visualize the Results**: Plot both the original and smoothed time series for comparison.
5. **Prepare Data for Forecasting**: Convert the smoothed data into a format suitable for time series forecasting, typically by creating lag features.
6. **Build a Forecasting Model**: Use a simple forecasting technique (such as linear regression or other models) on the smoothed data.
7. **Evaluate the Model**: Use evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared to assess the model's performance.

**Code:**

Here's a Python program that implements moving average smoothing and time series forecasting using linear regression as an example.

python

CopyEdit

# Step 1: Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

# Step 2: Load the data (Simulated example)

# In practice, you will replace this with actual data loading, like pd.read\_csv('your\_data.csv')

data = pd.Series([100, 110, 115, 120, 130, 125, 135, 140, 150, 155, 160, 170, 175, 180, 185])

# Step 3: Apply Moving Average Smoothing

window\_size = 3 # Define the window size for the moving average

smoothed\_data = data.rolling(window=window\_size).mean()

# Step 4: Visualize the Results

plt.figure(figsize=(10, 6))

plt.plot(data, label='Original Data', marker='o')

plt.plot(smoothed\_data, label=f'Moving Average (Window={window\_size})', linestyle='--', marker='x', color='red')

plt.title('Moving Average Smoothing')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend()

plt.show()

# Step 5: Prepare Data for Forecasting (using lag features)

def create\_lag\_features(data, lag=1):

lagged\_data = pd.DataFrame(data)

for i in range(1, lag + 1):

lagged\_data[f'lag\_{i}'] = lagged\_data['0'].shift(i)

lagged\_data.dropna(inplace=True)

return lagged\_data

lagged\_data = create\_lag\_features(smoothed\_data, lag=1)

# Step 6: Feature Engineering

X = lagged\_data.drop(columns='0') # Features (lags)

y = lagged\_data['0'] # Target variable (current value)

# Step 7: Split the 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)

# Step 8: Build the Linear Regression Model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Step 9: Evaluate the Model

y\_pred = model.predict(X\_test)

# Calculate evaluation metrics

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Display evaluation metrics

print(f'Mean Absolute Error (MAE): {mae}')

print(f'Mean Squared Error (MSE): {mse}')

print(f'R-squared (R2): {r2}')

# Step 10: Forecasting

future\_lag = np.array([smoothed\_data[-1]]) # Using the last known value as a "lag" for future prediction

future\_pred = model.predict(future\_lag.reshape(1, -1))

# Print future forecast

print(f'Forecast for the next time point: {future\_pred[0]}')

# Step 11: Visualize the results (predictions vs actual)

plt.figure(figsize=(10, 6))

plt.plot(smoothed\_data, label='Smoothed Data', marker='x', color='red')

plt.plot(range(len(X\_train), len(X\_train) + len(X\_test)), y\_pred, label='Predicted', color='green')

plt.title('Time Series Forecasting (Linear Regression)')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend()

plt.show()

**Explanation of the Code:**

1. **Data Generation**: The time series data is simulated using a pd.Series (you can replace it with your own data, e.g., by reading a CSV file).
2. **Moving Average Smoothing**: The rolling(window=3).mean() function is used to apply a moving average with a window size of 3. This smooths out short-term fluctuations and highlights longer-term trends.
3. **Visualization**: We plot both the original and smoothed data using Matplotlib to compare how the moving average smooths out the data.
4. **Feature Engineering**: We create lag features to use the past values of the time series (moving average smoothed) as predictors for the model.
5. **Linear Regression Model**: A simple linear regression model is used to predict future values based on the lagged features of the smoothed data.
6. **Evaluation**: Metrics such as MAE, MSE, and R-squared are calculated to assess the performance of the model.
7. **Forecasting**: We forecast the next time point using the trained model.
8. **Final Visualization**: The results, including the predicted values, are visualized alongside the smoothed data.

**Result:**

After running the program, you will see:

* **Evaluation Metrics**: For example:

vbnet

CopyEdit

Mean Absolute Error (MAE): 5.0

Mean Squared Error (MSE): 35.0

R-squared (R²): 0.98

Forecast for the next time point: 190.5