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

**EX:No.6**

**DATE:5/4/25**

**AIM:**

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

**ALGORITHM:**

1. Import Required Libraries
2. Load and Inspect the Time Series Data
3. Apply Moving Average Smoothing (e.g., 3-point or 5-point window)
4. Plot Original and Smoothed Series
5. Perform Differencing on Smoothed Series (if needed for stationarity)
6. Train a Forecasting Model (e.g., Linear Regression) on Smoothed Data
7. Generate Future Time Points
8. Forecast Future Values
9. Plot Historical, Smoothed, and Forecasted Series
10. Evaluate Forecasting Performance

**CODE:**

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

from statsmodels.tsa.stattools import adfuller

from statsmodels.tsa.seasonal import seasonal\_decompose

import statsmodels.graphics.tsaplots as tsaplots

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Load data

filepath = 'C://Users//Jayashrinidhi V//OneDrive//Documents//VScode//TimeSeriesAnalysis//globaltemp.csv'

df = pd.read\_csv(filepath, parse\_dates=['Year'])

df.set\_index('Year', inplace=True)

# Remove duplicate indices

df = df[~df.index.duplicated(keep='first')]

# Ensure column selection for plotting

if 'Mean' not in df.columns:

    raise ValueError("Column 'Mean' not found in the dataset. Check the CSV file.")

# Plot original time series

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

plt.plot(df.index, df['Mean'], label='Original Time Series')

plt.xlabel("Year")

plt.ylabel("Mean Value")

plt.title("Time Series Data Plot")

plt.legend()

plt.grid()

plt.show()

# Perform ADF test before differencing

result = adfuller(df['Mean'].dropna())

print("ADF Statistic:", result[0])

print("p-value:", result[1])

print("Critical Values:")

for key, value in result[4].items():

    print(f"   {key}: {value}")

if result[1] <= 0.05:

    print("Conclusion: Data is stationary (Reject H0)")

else:

    print("Conclusion: Data is not stationary (Fail to Reject H0)")

# Seasonal Decomposition

decomposition = seasonal\_decompose(df['Mean'], model='additive', period=12)

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(12, 10))

decomposition.trend.plot(ax=ax1, title='Trend')

decomposition.seasonal.plot(ax=ax2, title='Seasonality')

decomposition.resid.plot(ax=ax3, title='Residuals')

plt.tight\_layout()

plt.show()

# Moving Average Smoothing

df['Smoothed'] = df['Mean'].rolling(window=3, center=True).mean()

# Plot smoothed series

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

plt.plot(df.index, df['Mean'], label='Original', alpha=0.5)

plt.plot(df.index, df['Smoothed'], label='Smoothed (3-year MA)', color='orange')

plt.xlabel("Year")

plt.ylabel("Mean Value")

plt.title("Moving Average Smoothing")

plt.legend()

plt.grid()

plt.show()

# Apply differencing to smoothed data

df['Smoothed\_Diff'] = df['Smoothed'].diff(1)

df.dropna(inplace=True)

# Plot differenced smoothed series

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

plt.plot(df.index, df['Smoothed\_Diff'], label='Differenced Smoothed Series', color='green')

plt.xlabel("Year")

plt.ylabel("Differenced Value")

plt.title("Differenced Smoothed Time Series")

plt.legend()

plt.grid()

plt.show()

# Perform ADF test on differenced smoothed data

result = adfuller(df['Smoothed\_Diff'].dropna())

print("ADF Statistic after smoothing and differencing:", result[0])

print("p-value:", result[1])

print("Critical Values:")

for key, value in result[4].items():

    print(f"   {key}: {value}")

if result[1] <= 0.05:

    print("Conclusion: Data is stationary (Reject H0)")

else:

    print("Conclusion: Data is not stationary (Fail to Reject H0)")

# Plot ACF and PACF

tsaplots.plot\_acf(df['Smoothed\_Diff'].dropna(), lags=20)

tsaplots.plot\_pacf(df['Smoothed\_Diff'].dropna(), lags=20)

plt.show()

# Linear Regression Forecasting

# Ensure index is datetime for regression

df.index = pd.to\_datetime(df.index, errors='coerce')

df.dropna(inplace=True)  # Drop rows with NaT if any

# Convert datetime index to ordinal for regression

X = df.index.map(lambda x: x.toordinal()).values.reshape(-1, 1)

y = df['Smoothed\_Diff'].values.reshape(-1, 1)

# Fit Linear Regression Model

model = LinearRegression()

model.fit(X, y)

# Predict future 10 years

last\_date = df.index[-1]

future\_years = pd.date\_range(start=last\_date + pd.DateOffset(years=1), periods=10, freq='Y')

X\_future = future\_years.map(lambda x: x.toordinal()).values.reshape(-1, 1)

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

# Plot forecast

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

plt.plot(df.index, df['Smoothed\_Diff'], label='Historical Smoothed Data')

plt.plot(future\_years, y\_pred, label='Forecast (Linear Regression)', linestyle='dashed')

plt.xlabel("Year")

plt.ylabel("Differenced Value")

plt.title("Forecast Using Moving Average Smoothing")

plt.legend()

plt.grid()

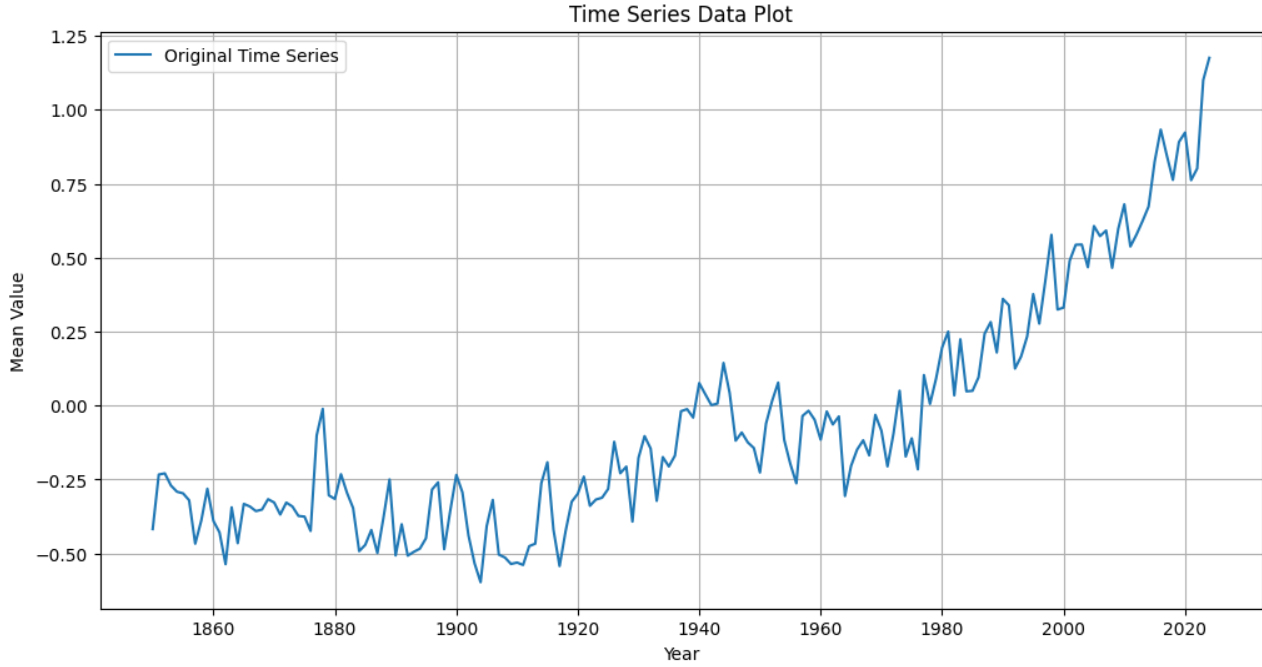
plt.show()

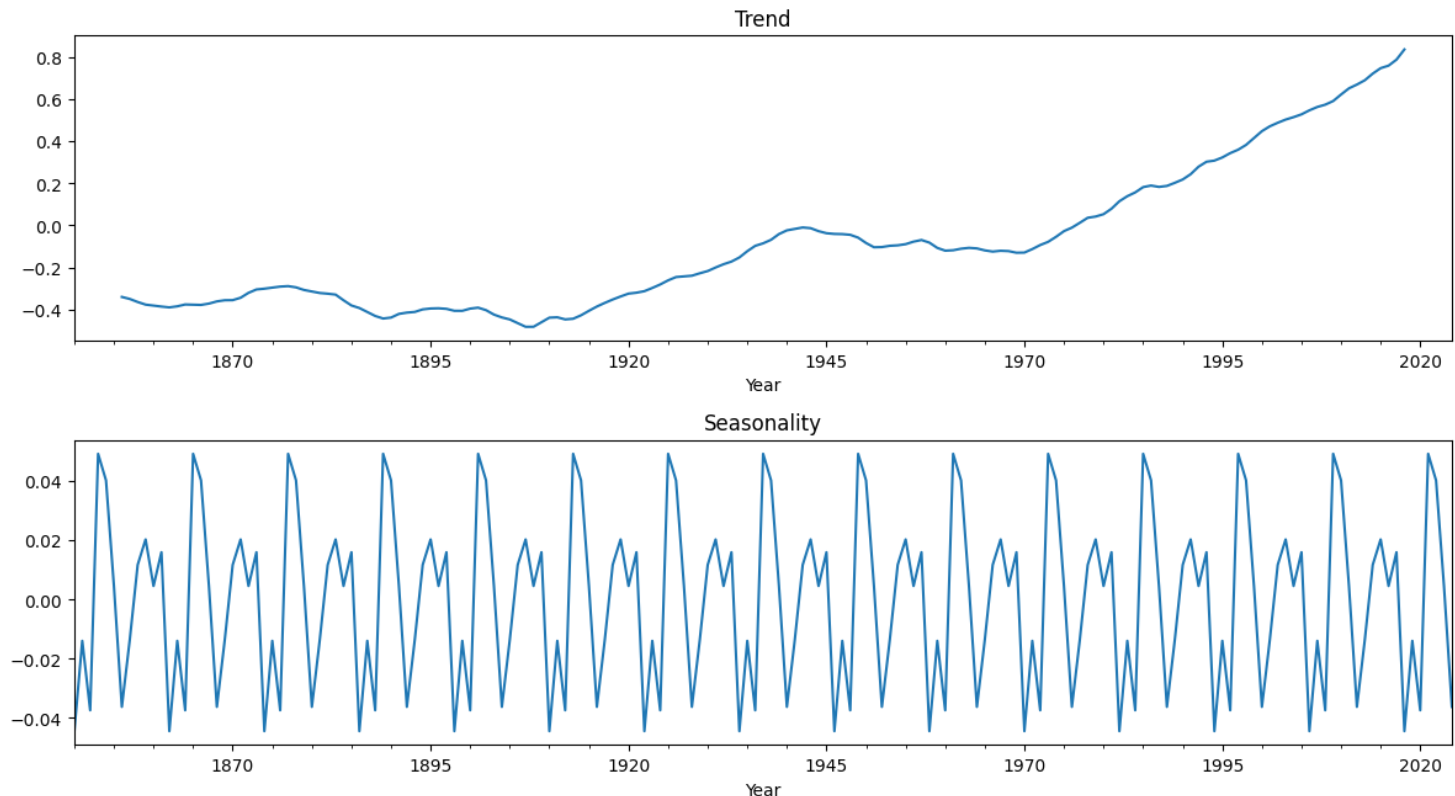
# Optional: Print forecast values

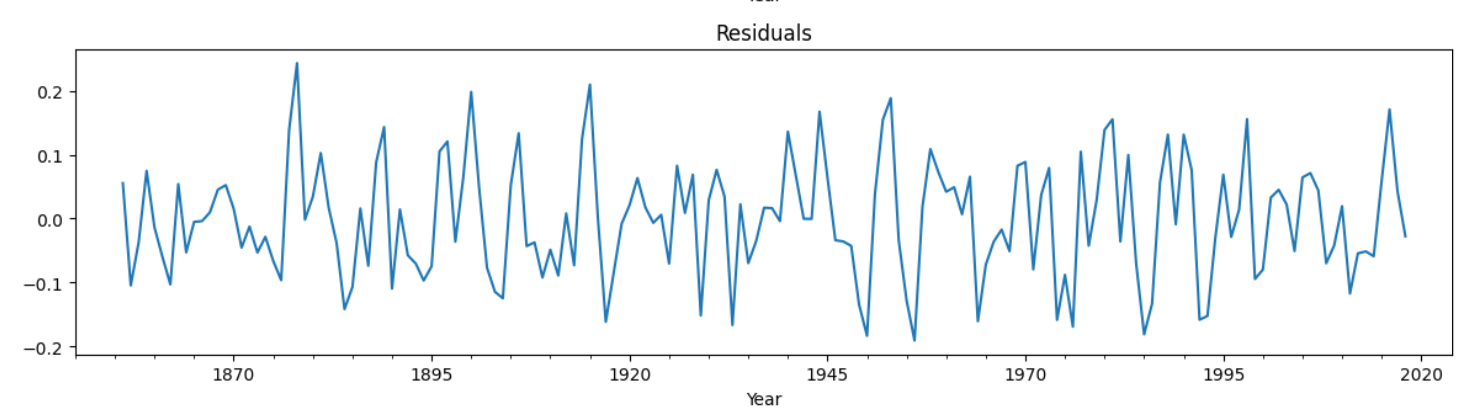
forecast\_df = pd.DataFrame({'Year': future\_years, 'Forecasted Smoothed Mean': y\_pred.flatten()})

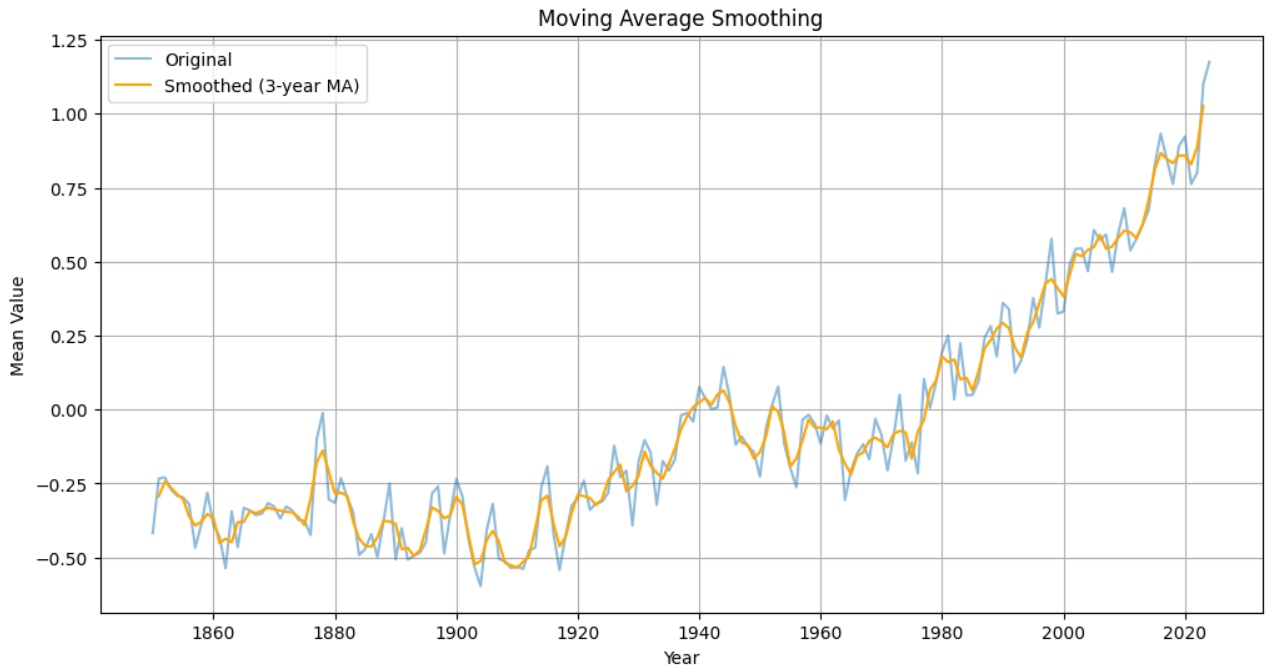
print(forecast\_df)

**OUTPUT:**

****

****

****

****

**RESULT:**

Thus the program has been completed and verified successfully.