EX:No.6 **DATE:5/4/25**	**Implement program to apply moving average smoothing for data preparation and time series forecasting.**

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:

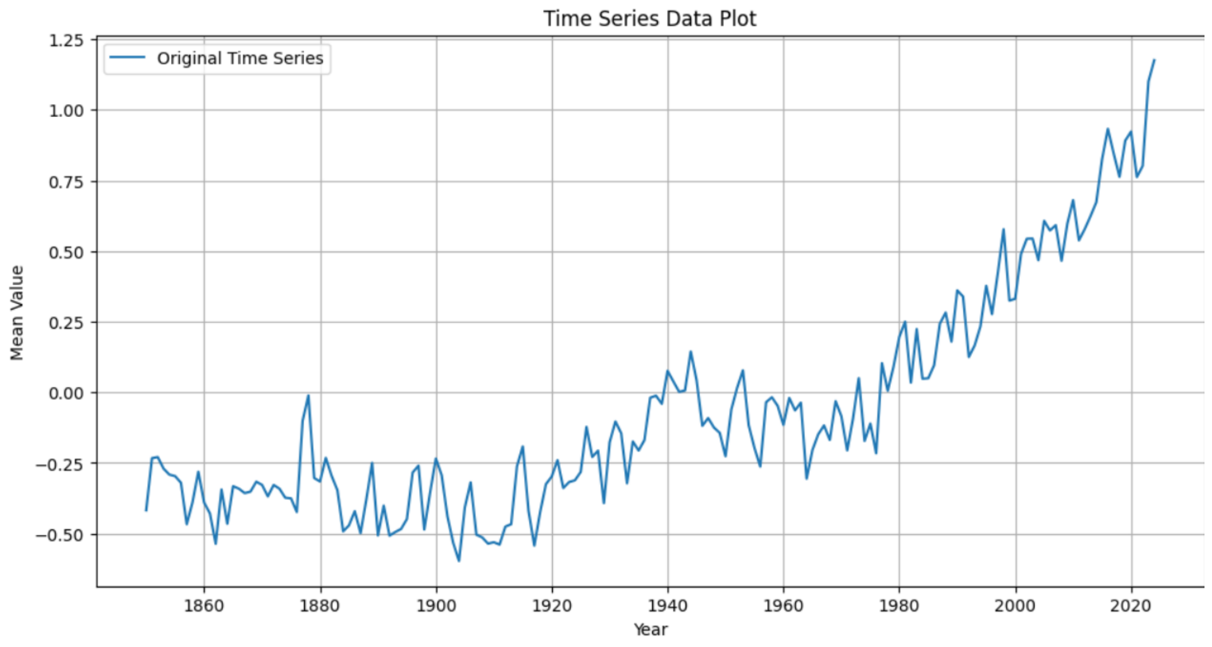
```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.graphics.tsaplots as tsaplots
from sklearn.linear_model import LinearRegression
from sklearn.metrics 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
```

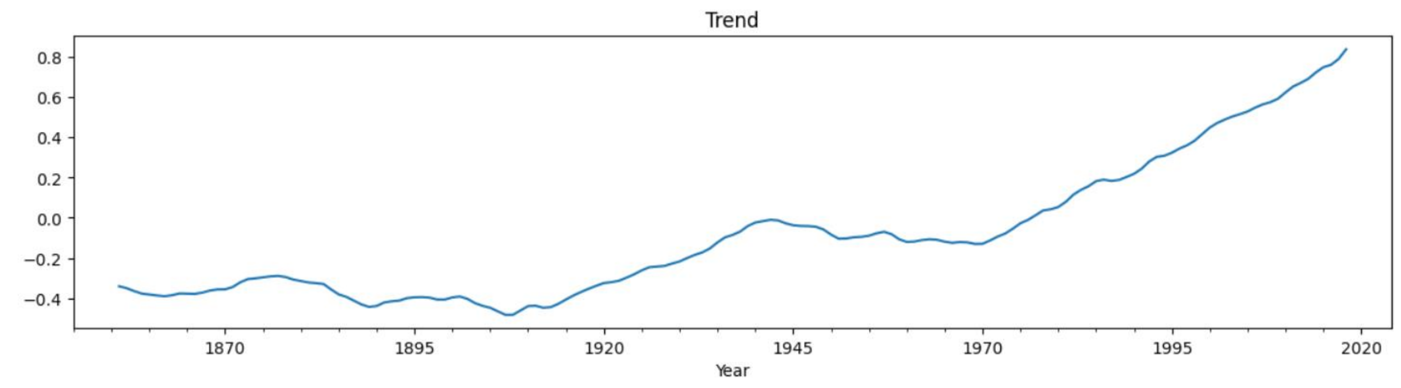
```
df = df[~df.index.duplicated(keep='first')]

# 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:")
```

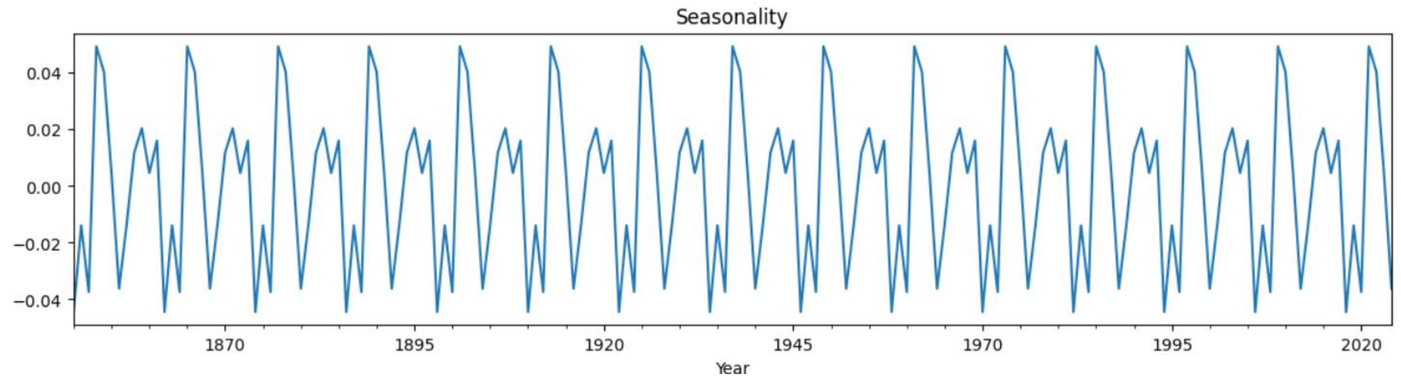
```
for key, value in result[4].items():
    print(f"   {key}: {value}")
if result[1] <= 0.05:
    print("Conclusion: Data is stationary (Reject H0)")
else:
    print("Conclusion: Data is not stationary (Fail to Reject H0)")

# 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')
```

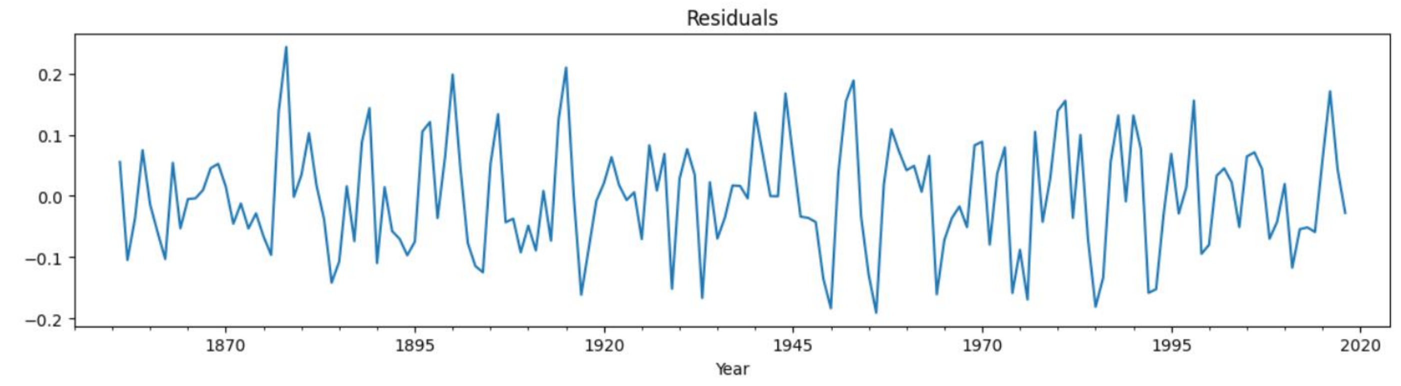
```
plt.xlabel("Year")
plt.ylabel("Differenced Value")
plt.title("Forecast Using Moving Average Smoothing")
plt.legend()
plt.grid()
plt.show()

# Optional: Print forecast values
forecast_df = pd.DataFrame({'Year': future_years, 'Forecasted Smoothed
Mean': y_pred.flatten()})
print(forecast_df)
```

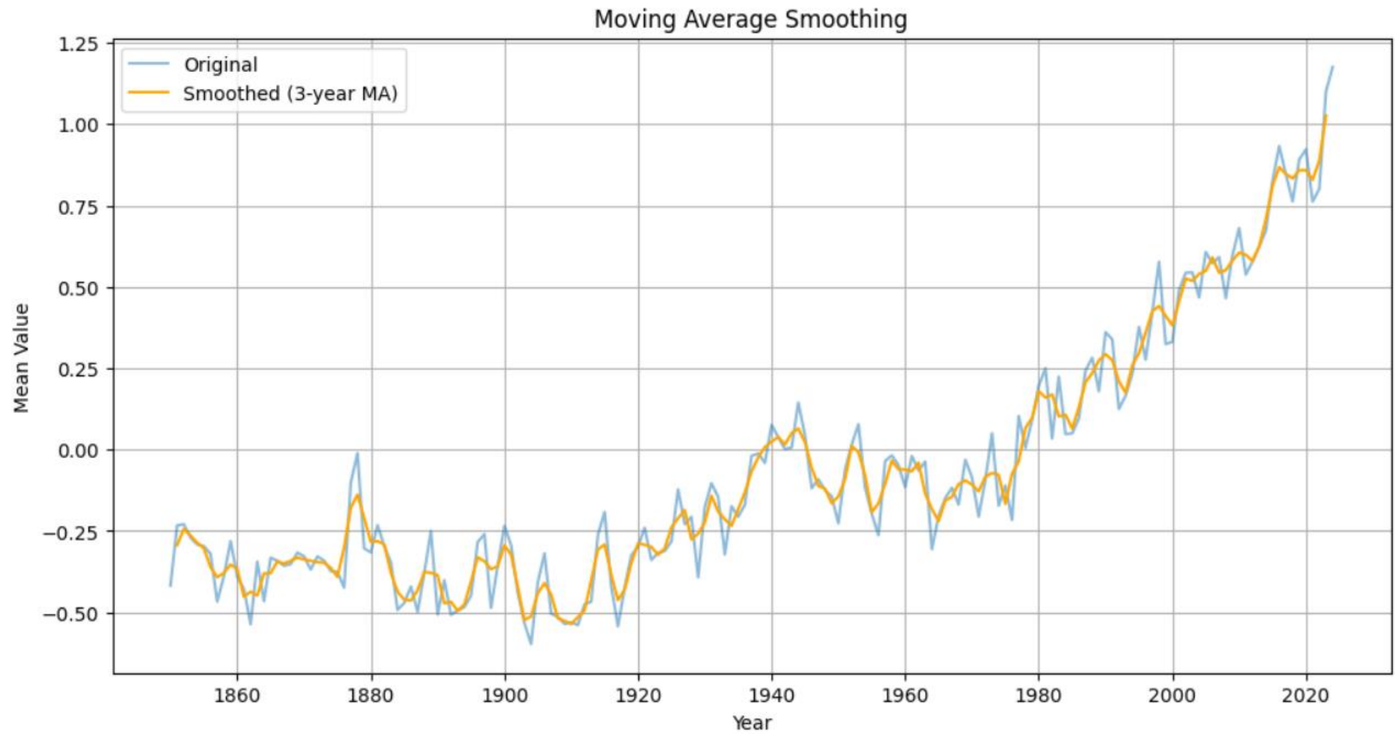
OUTPUT:




Trend
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
1870
1895
1920
1945
1970
1995
2020
Year


Seasonality
0.04
0.02
0.00
−0.02
−0.04
1870
1895
1920
1945
1970
1995
2020
Year


Residuals
0.2
0.1
0.0
−0.1
−0.2
1870
1895
1920
1945
1970
1995
2020
Year



RESULT:

Thus the program has been completed and verified successfully.