EX:No.10 DATE:13/4/2025	**Implement program for decomposing time series data into trend and seasonality.**

AIM:

To Implement program for decomposing time series data into trend and seasonality.

ALGORITHM:

1. **Import Libraries**
   - Use pandas, numpy, matplotlib, statsmodels
2. **Load and Prepare Dataset**
   - Import multivariate time series data (e.g., economic indicators, stock prices)
   - Ensure time is the index and all variables are numeric
3. **Check for Missing Values**
   - Impute or drop missing entries
   - Stationarity assumption requires complete sequences
4. **Make Series Stationary**
   - Apply differencing (data.diff()) if needed
   - Use ADF test for each variable to verify stationarity
5. **Split into Train and Test Sets**
   - Use ~80% for training
   - Reserve the rest for forecasting and validation
6. **Select Optimal Lag Order (p)**
   - Use model.select_order() with AIC, BIC, or FPE criteria
   - Choose optimal lag based on minimum information criteria
7. **Fit the VAR Model**
   - Use VAR(train_data)
   - Fit using model.fit(maxlags=p)
8. **Forecast Future Values**
   - Use model.forecast(y=train.values[-p:], steps=n)
   - Get predictions for multiple time steps and all variables
9. **Inverse Differencing (if applied earlier)**
   - Reconstruct original scale by adding differenced values back to last known actuals

10. **Evaluate Forecast Accuracy**

- Use RMSE or MAE for each variable
- Compare forecasted vs actual values

11. **Plot Results**

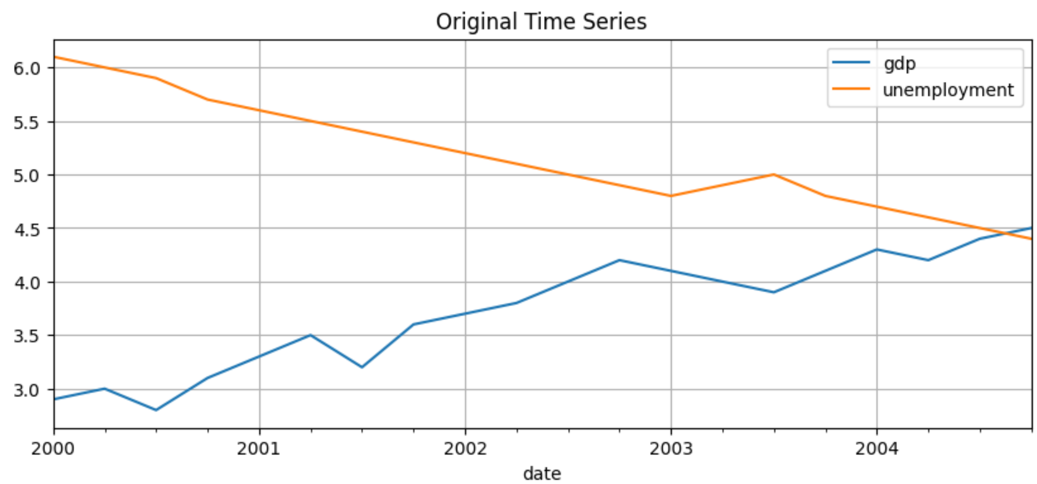
- Time series plots for actual vs predicted for each variable
- Optionally plot residuals and confidence intervals

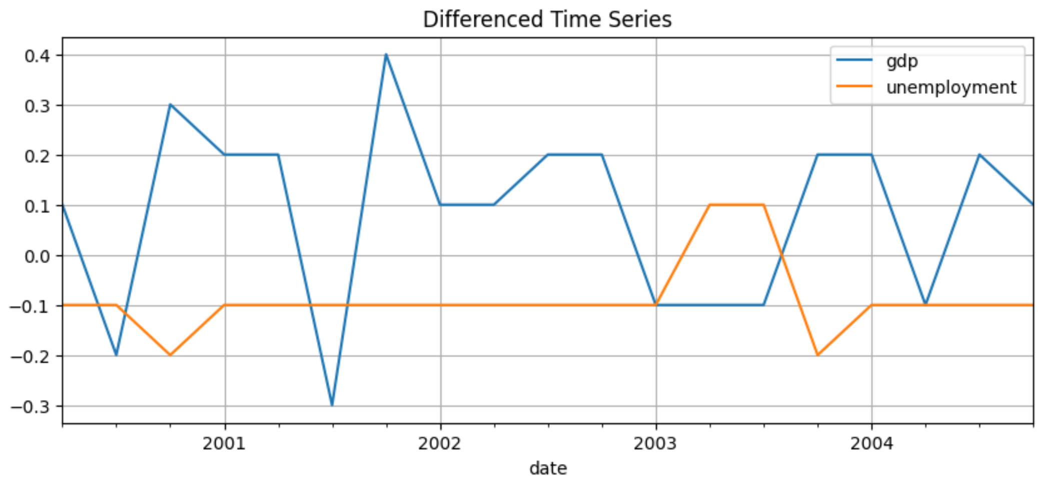
CODE:

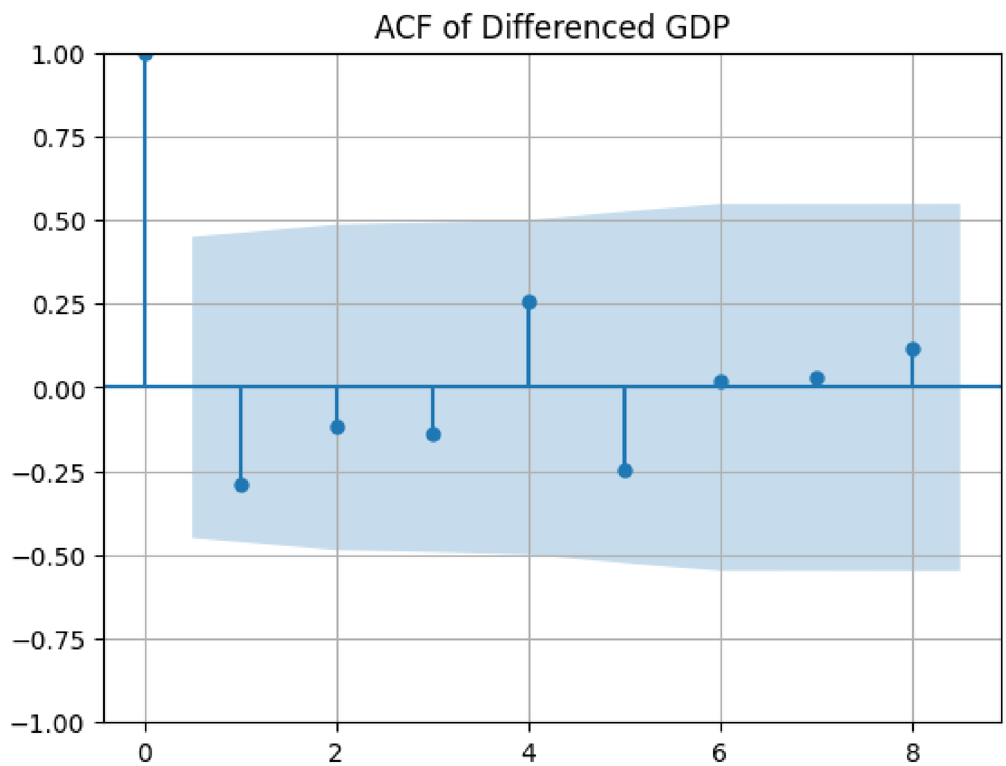
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.api import VAR
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import adfuller

# 1. Sample Dataset (20 quarters of GDP & Unemployment)
data = {
    'date': pd.date_range(start='2000-01', periods=20, freq='Q'),
    'gdp': [2.9, 3.0, 2.8, 3.1, 3.3, 3.5, 3.2, 3.6, 3.7, 3.8,
            4.0, 4.2, 4.1, 4.0, 3.9, 4.1, 4.3, 4.2, 4.4, 4.5],
    'unemployment': [6.1, 6.0, 5.9, 5.7, 5.6, 5.5, 5.4, 5.3, 5.2, 5.1,
                     5.0, 4.9, 4.8, 4.9, 5.0, 4.8, 4.7, 4.6, 4.5, 4.4]
}
df = pd.DataFrame(data).set_index('date')

# 2. Difference to make stationary
df_diff = df.diff().dropna()

# 3. Fit VAR Model with safe lag
model = VAR(df_diff)
results = model.fit(maxlags=2)
k_ar = results.k_ar

# 4. Forecast
forecast_input = df_diff.values[-k_ar:]
forecast = results.forecast(y=forecast_input, steps=4)
forecast_df = pd.DataFrame(forecast, columns=df.columns)

# Inverse transform
last_obs = df.iloc[-1]
forecast_values = forecast_df.cumsum() + last_obs

# 5. PLOTS (5 Total)

# 1. Original Time Series
df.plot(title='Original Time Series', figsize=(10, 4))
```

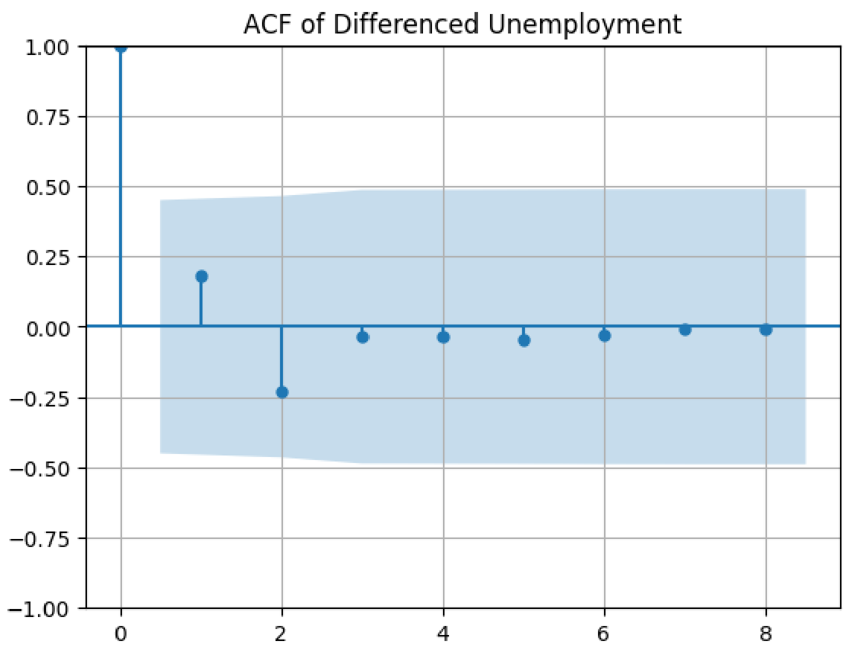
```
plt.grid(True)
plt.show()

# 2. Differenced Time Series
df_diff.plot(title='Differenced Time Series', figsize=(10, 4))
plt.grid(True)
plt.show()

# 3. ACF Plot - GDP
plot_acf(df_diff['gdp'], lags=8)
plt.title('ACF of Differenced GDP')
plt.grid(True)
plt.show()

# 4. ACF Plot - Unemployment
plot_acf(df_diff['unemployment'], lags=8)
plt.title('ACF of Differenced Unemployment')
plt.grid(True)
plt.show()

# 5. Forecast Plot
plt.plot(df['gdp'], label='GDP Actual')
plt.plot(df['unemployment'], label='Unemployment Actual')
forecast_index = pd.date_range(start=df.index[-1], periods=5, freq='Q')[1:]
plt.plot(forecast_index, forecast_values['gdp'], '--', label='GDP Forecast')
plt.plot(forecast_index, forecast_values['unemployment'], '--', label='Unemployment Forecast')
plt.title('Forecast using VAR Model')
plt.legend()
plt.grid(True)
plt.show()
```

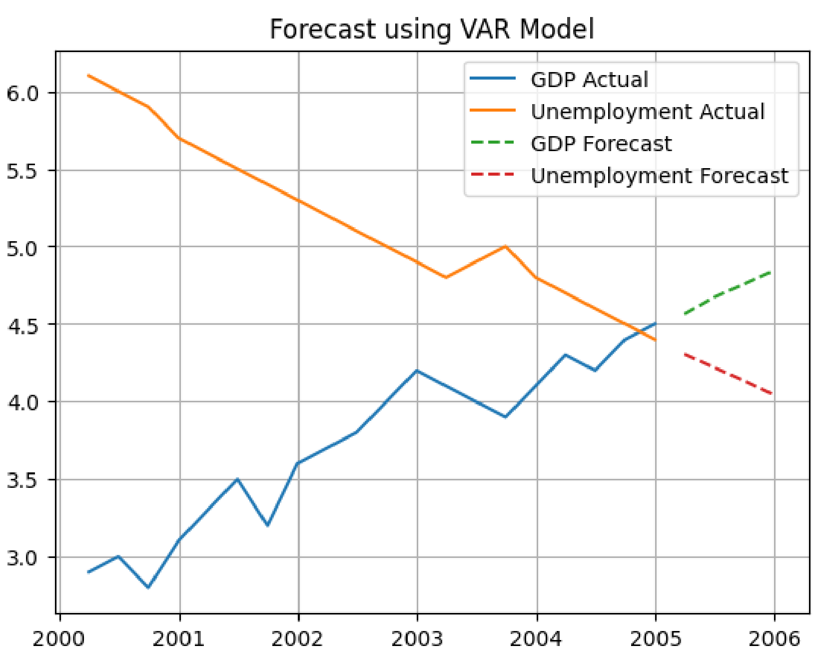
OUTPUT:




Differenced Time Series
gdp
unemployment
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
2001
2002
2003
2004
date


ACF of Differenced GDP
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8





RESULT:

Thus the program has been completed and verified successfully.