**Develop vector auto regression model for multivariate time series data forecasting.**

**EX:No.9**

**DATE:12/04/25**

# AIM:

To Develop vector auto regression model for multivariate time series data forecasting.

# ALGORITHM:

1. Import Libraries.
2. Create or Load Multivariate Time Series Data.
3. Visualize the Data.
4. Check Stationarity
5. Forecasting – Predicts future PM2.5 values for the next 30 days using the trained model.
6. Plot Results – Plots actual vs forecasted PM2.5 levels to visualize model performance.

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.api import VAR

from statsmodels.tsa.stattools import adfuller

# Step 1: Generate Synthetic Data

np.random.seed(42)

dates = pd.date\_range(start="2020-01-01", periods=200, freq='D')

# Create synthetic multivariate time series

price = np.cumsum(np.random.normal(loc=0.2, scale=1.5, size=200)) + 60

demand = np.cumsum(np.random.normal(loc=0.3, scale=2, size=200)) + 100

production = np.cumsum(np.random.normal(loc=0.25, scale=1, size=200)) + 90

data = pd.DataFrame({

'Date': dates,

'Price': price,

'Demand': demand,

'Production': production

}).set\_index('Date')

# Step 2: Plot the data

data.plot(title='Synthetic Air Pollution Time Series Data', figsize=(10, 5))

plt.grid()

plt.show()

# Step 3: Check stationarity and difference if needed

def adf\_test(series, name):

result = adfuller(series)

print(f'{name}: ADF Statistic = {result[0]}, p-value = {result[1]}')

for column in data.columns:

adf\_test(data[column], column)

# If p > 0.05, apply differencing

data\_diff = data.diff().dropna()

# Step 4: Fit VAR Model

model = VAR(data\_diff)

lag\_order = model.select\_order(maxlags=15)

print("Selected Lags:\n", lag\_order.summary())

model\_fitted = model.fit(lag\_order.aic)

print(model\_fitted.summary())

# Step 5: Forecasting

forecast\_input = data\_diff.values[-model\_fitted.k\_ar:]

forecast = model\_fitted.forecast(y=forecast\_input, steps=10)

# Step 6: Convert forecast back to original scale

forecast\_df = pd.DataFrame(forecast, columns=['Price', 'Demand', 'Production'])

forecast\_df.index = pd.date\_range(start=data.index[-1] + pd.Timedelta(days=1), periods=10)

# Reverse differencing by adding last known values

last\_values = data.iloc[-1]

forecast\_df = forecast\_df.cumsum() + last\_values

# Step 7: Plot the forecast

plt.figure(figsize=(10, 5))

plt.plot(data['Price'], label='Historical Price')

plt.plot(forecast\_df['Price'], label='Forecast Price', color='red')

plt.title('Crude Oil Price Forecast (VAR Model)')

plt.legend()

plt.grid()

plt.show()

model\_fitted = model.fit(lag\_order.aic)

print(model\_fitted.summary())

# Step 5: Forecasting

forecast\_input = data\_diff.values[-model\_fitted.k\_ar:]

forecast = model\_fitted.forecast(y=forecast\_input, steps=10)

# Step 6: Convert forecast back to original scale

forecast\_df = pd.DataFrame(forecast, columns=['Price', 'Demand', 'Production'])

forecast\_df.index = pd.date\_range(start=data.index[-1] + pd.Timedelta(days=1), periods=10)

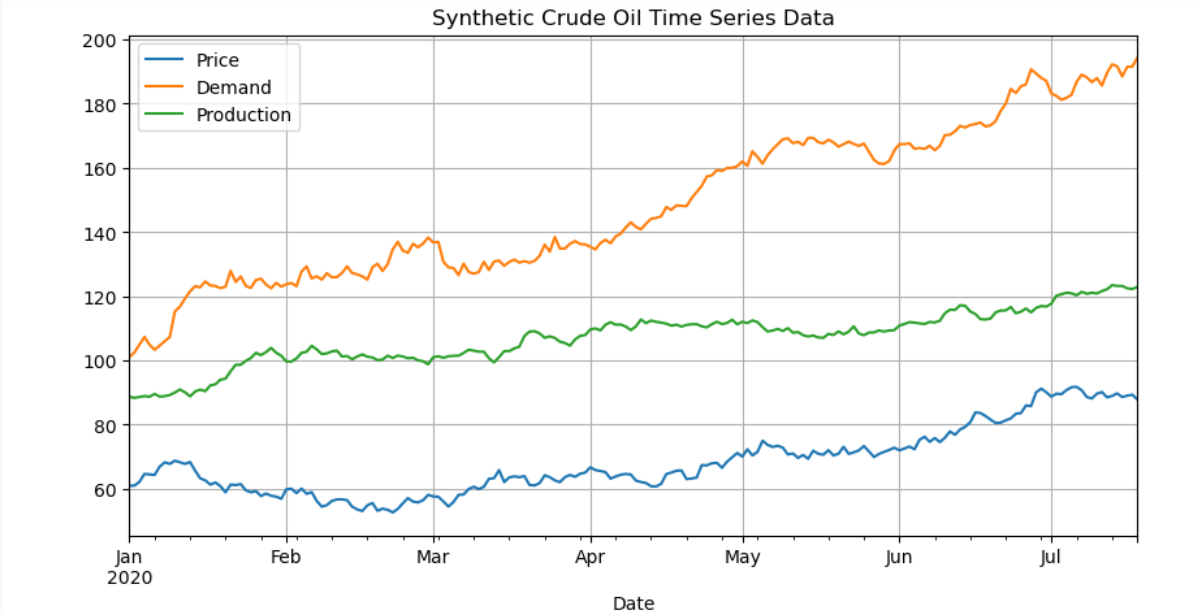
plt.grid()

plt.show()

model\_fitted = model.fit(lag\_order.aic)

print(model\_fitted.summary())

**OUTPUT:**



**RESULT:**

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

# 