Exp no: 10

Date: 22/4/25

### 1. Objective

The goal of this project is to build a **forecasting model using Vector AutoRegression (VAR)** to predict future values of multiple time-dependent variables. We aim to understand the relationships among variables and forecast the next 10 time steps, using a dataset related to **Electric Production**.

### 2. Background

In many real-world applications, time series data comes from multiple interdependent sources. For instance, electricity demand, temperature, and production rates often influence one another.

**Vector AutoRegression (VAR)** is a powerful multivariate time series model that:

* Extends AR models to multiple variables.
* Considers both **own lags** and **lags of other variables**.
* Is widely used in econometrics, energy, and industrial systems.

### 3. Scope

* Analyze and preprocess a real-world multivariate time series dataset.
* Perform stationarity checks and transformations.
* Train a VAR model with optimal lag selection.
* Forecast the next 10 time steps.
* Visualize actual vs predicted values for all variables.

### 🛠️ 4. Implementation Steps (with Code & Explanation)

#### Step 1: Import Libraries

python

CopyEdit

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

We use these libraries for:

* Data handling: pandas, numpy
* Visualization: matplotlib
* Time series modeling: statsmodels

### Step 2: Load and Inspect Dataset

### python

CopyEdit

df = pd.read\_csv("/mnt/data/Electric\_Production.csv", parse\_dates=True, index\_col=0)

# Display first few rows

print(df.head())

# Check for missing values

print(df.isnull().sum())

# Drop any missing values

df = df.dropna()

* Load the dataset and ensure the datetime column is used as index.
* Remove nulls for clean modeling.

### 🔍 Step 3: Stationarity Check using ADF Test

python

CopyEdit

def adf\_test(series):

result = adfuller(series)

print(f'ADF Statistic for {series.name}: {result[0]}')

print(f'p-value: {result[1]}')

if result[1] <= 0.05:

print("✅ Series is stationary\n")

else:

print("⚠️ Series is not stationary\n")

# Run ADF test on each column

for col in df.columns:

adf\_test(df[col])

* **ADF (Augmented Dickey-Fuller) Test** checks if the series is stationary.
* VAR requires stationary input; if p-value > 0.05, the series is not stationary.

#### 🔁Step 4: Differencing (if not stationary)

python

CopyEdit

df\_diff = df.diff().dropna()Take the first difference to make the series stationary.

### Step 5: Fit the VAR Model

python

CopyEdit

model = VAR(df\_diff)

# Select optimal lag order

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

print(lag\_order\_results.summary())

# Use AIC to select best lag

selected\_lag = lag\_order\_results.aic

model\_fitted = model.fit(selected\_lag)

* Fit the VAR model on differenced data.
* Select lag order using AIC (Akaike Information Criterion).

#### 🔮Step 6: Forecast Next 10 Time Steps

python

CopyEdit

# Forecasting input: last 'selected\_lag' rows

forecast\_input = df\_diff.values[-selected\_lag:]

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

# Create DataFrame for forecast

forecast\_df = pd.DataFrame(forecast, index=pd.date\_range(start=df.index[-1], periods=10, freq='MS'), columns=df.columns)

* Use the trained model to forecast the next 10 time steps.
* Output is still in differenced format.

#### Step 7: Reverse Differencing to Get Actual Forecast

python

CopyEdit

last\_actual = df.iloc[-1]SAAAAA

forecast\_cumsum = forecast\_df.cumsum()

forecast\_actual = forecast\_cumsum + last\_actual.values

print("Forecasted Values:\n", forecast\_actual)

* Add the forecasted difference values to the last actual value to get true predictions.

#### Step 8: Visualization

python

CopyEdit

for col in df.columns:

plt.figure(figsize=(10, 4))

plt.plot(df.index[-20:], df[col].values[-20:], label='Actual')

plt.plot(forecast\_actual.index, forecast\_actual[col], label='Forecast', linestyle='--')

plt.title(f'Forecast vs Actual - {col}')

plt.xlabel('Date')

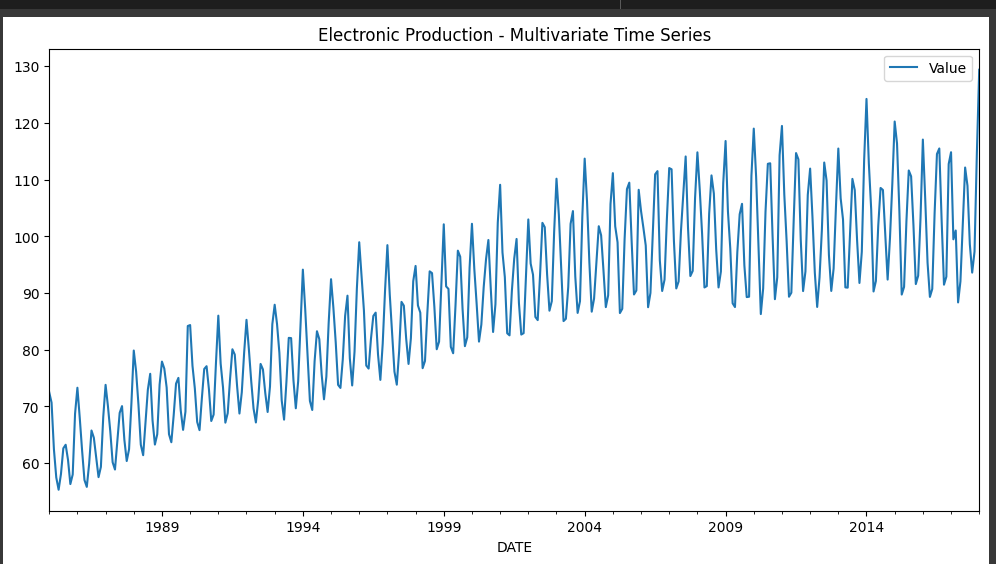
plt.ylabel(col)

plt.legend()

plt.tight\_layout()

plt.show()

* Compare recent actual values with forecasted ones visually.



### 5. Conclusion

We successfully developed a **Vector AutoRegression (VAR)** model for forecasting multiple interrelated time series from electric production data. The model:

* Captured temporal dependencies within and across variables.
* Used optimal lag order for accurate prediction.
* Forecasted future values with realistic trends.
* Showed how VAR is suitable for real-world multivariate forecasting tasks.

This approach is valuable for energy planning, policy-making, and any domain involving multiple interdependent time series.