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

EX.N0 : 10	Develop vector auto regression model for multivariate time series data forecasting.
DATE : 12/04/2025	

AIM:

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

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller, grangercausalitytests
from statsmodels.tools.eval_measures import rmse
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

# Set modern plotting style
plt.style.use('ggplot')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = [12, 6]
plt.rcParams['figure.dpi'] = 100

# ====================== DATA LOADING & PREPROCESSING
======================
def load_data(earthquake.csv):
from io import StringIO
    df = pd.read_csv(StringIO(data))

    # Convert month names to numbers
    month_map = {m:i+1 for i,m in enumerate(['January','February','March','April','May','June',
                                    'July','August','September','October','November','December'])}
    df['month'] = df['month'].map(month_map)

    # Create datetime index
    df['date'] = pd.to_datetime(df[['year','month','day']])
```

```
    df.set_index('date', inplace=True)
    df.sort_index(inplace=True)

    # Feature engineering
    df['log_deaths'] = np.log1p(df['deaths'])  # Log transform for skewed death counts
    df['major_quake'] = (df['richter'] >= 7.0).astype(int)  # Binary flag for major quakes

    # Select final features
    return df[['richter', 'log_deaths', 'major_quake']]

df = load_data()

# Resample to monthly frequency (max values)
df_monthly = df.resample('M').max().ffill()

# ====================== EXPLORATORY VISUALIZATIONS ======================
fig, axes = plt.subplots(3, 1, figsize=(12, 12))

# Time series of earthquake magnitudes
axes[0].plot(df_monthly.index, df_monthly['richter'], color='darkorange')
axes[0].set_title('Earthquake Magnitudes Over Time', fontsize=14)
axes[0].set_ylabel('Richter Scale')
axes[0].grid(alpha=0.4)

# Time series of log deaths
axes[1].plot(df_monthly.index, df_monthly['log_deaths'], color='royalblue')
axes[1].set_title('Log-Transformed Death Toll Over Time', fontsize=14)
axes[1].set_ylabel('log(1 + Deaths)')
axes[1].grid(alpha=0.4)

# Major earthquakes
axes[2].stem(df_monthly.index, df_monthly['major_quake'], linefmt='crimson', markerfmt=' ')
axes[2].set_title('Major Earthquakes (≥7.0 Richter)', fontsize=14)
axes[2].set_ylabel('Occurrence')
axes[2].grid(alpha=0.4)

plt.tight_layout()
plt.show()

# Correlation heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df_monthly.corr(), annot=True, cmap='coolwarm', center=0)
plt.title('Feature Correlation Matrix', fontsize=14)
plt.show()

# ====================== STATIONARITY CHECK ======================
def check_stationarity(series):
```

```
    result = adfuller(series.dropna())
    print(f'Variable: {series.name}')
    print(f'ADF Statistic: {result[0]:.4f}')
    print(f'p-value: {result[1]:.4f}')
    print('Critical Values:')
    for key, value in result[4].items():
        print(f'\t{key}: {value:.4f}')
    print('-'*50)

print("\nStationarity Tests:")
for col in df_monthly.columns:
    check_stationarity(df_monthly[col])

# Differencing to achieve stationarity
df_stationary = df_monthly.diff().dropna()

# ====================== VAR MODEL IMPLEMENTATION ======================
# Scale the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_stationary)
df_scaled = pd.DataFrame(scaled_data, index=df_stationary.index,
                         columns=df_stationary.columns)

# Train-test split (80-20)
n_obs = int(len(df_scaled)*0.8)
train, test = df_scaled[:n_obs], df_scaled[n_obs:]

# Lag order selection
model = VAR(train)
lag_results = model.select_order(maxlags=12)
print(lag_results.summary())

# Fit model with optimal lags (using AIC)
optimal_lags = lag_results.aic
var_model = VAR(train)
fitted_model = var_model.fit(optimal_lags)
print(fitted_model.summary())

# ====================== FORECASTING & EVALUATION ======================
# Forecast on test set
lag_order = fitted_model.k_ar
forecast = fitted_model.forecast(train.values[-lag_order:], steps=len(test))

# Inverse transform
def inverse_transform(forecast, original_df, scaler):
    dummy = np.zeros((len(forecast), original_df.shape[1]))
    for i in range(original_df.shape[1]):
```

```
        dummy[:,i] = forecast[:,i]
    return scaler.inverse_transform(dummy)

forecast_inv = inverse_transform(forecast, df_stationary, scaler)
test_inv = inverse_transform(test.values, df_stationary, scaler)

# Calculate RMSE
for i, col in enumerate(df_stationary.columns):
    actual = test_inv[:,i]
    pred = forecast_inv[:,i]
    rmse_val = rmse(actual, pred)
    print(f'RMSE for {col}: {rmse_val:.4f}')

# ====================== VISUALIZATION OF RESULTS ======================
# Plot forecasts vs actuals
fig, axes = plt.subplots(3, 1, figsize=(12, 12))

# Richter scale
axes[0].plot(df_monthly.index[n_obs:], df_monthly['richter'].iloc[n_obs:],
            label='Actual', color='navy')
axes[0].plot(df_monthly.index[n_obs:], df_monthly['richter'].iloc[n_obs-1] +
np.cumsum(forecast_inv[:,0]),
            label='Forecast', linestyle='--', color='orange')
axes[0].set_title('Earthquake Magnitude: Actual vs Forecast', fontsize=14)
axes[0].set_ylabel('Richter Scale')
axes[0].legend()
axes[0].grid(alpha=0.3)

# Death toll (original scale)
axes[1].plot(df_monthly.index[n_obs:], np.expm1(df_monthly['log_deaths'].iloc[n_obs:]),
            label='Actual', color='navy')
axes[1].plot(df_monthly.index[n_obs:], np.expm1(df_monthly['log_deaths'].iloc[n_obs-1] +
np.cumsum(forecast_inv[:,1])),
            label='Forecast', linestyle='--', color='orange')
axes[1].set_title('Death Toll: Actual vs Forecast', fontsize=14)
axes[1].set_ylabel('Number of Deaths')
axes[1].legend()
axes[1].grid(alpha=0.3)

# Major quakes probability
axes[2].plot(df_monthly.index[n_obs:], df_monthly['major_quake'].iloc[n_obs:],
            label='Actual', color='navy')
axes[2].plot(df_monthly.index[n_obs:], (df_monthly['major_quake'].iloc[n_obs-1] +
np.cumsum(forecast_inv[:,2]) > 0.5).astype(int),
            label='Forecast', linestyle='--', color='orange')
axes[2].set_title('Major Earthquake Occurrence: Actual vs Forecast', fontsize=14)
axes[2].set_ylabel('Probability')
```

```
axes[2].legend()
axes[2].grid(alpha=0.3)

plt.tight_layout()
plt.show()

# ===================== FORECAST INTO FUTURE ======================
# 12-month forecast
n_forecast = 12
future_forecast = fitted_model.forecast(df_scaled.values[-lag_order:], steps=n_forecast)
future_forecast_inv = inverse_transform(future_forecast, df_stationary, scaler)

# Create future dates
last_date = df_monthly.index[-1]
future_dates = pd.date_range(start=last_date + pd.DateOffset(months=1), periods=n_forecast, freq='M')

# Plot future forecast
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Magnitude forecast
axes[0].plot(df_monthly.index[-24:], df_monthly['richter'].iloc[-24:], label='Historical', color='navy')
axes[0].plot(future_dates, df_monthly['richter'].iloc[-1] + np.cumsum(future_forecast_inv[:,0]),
        label='Forecast', marker='o', color='darkorange')
axes[0].set_title('12-Month Magnitude Forecast')
axes[0].set_ylabel('Richter Scale')
axes[0].legend()
axes[0].grid(alpha=0.3)

# Death toll forecast
axes[1].plot(df_monthly.index[-24:], np.expm1(df_monthly['log_deaths'].iloc[-24:]), label='Historical',
color='navy')
axes[1].plot(future_dates, np.expm1(df_monthly['log_deaths'].iloc[-1] +
np.cumsum(future_forecast_inv[:,1])),
        label='Forecast', marker='o', color='darkorange')
axes[1].set_title('12-Month Death Toll Forecast')
axes[1].set_ylabel('Number of Deaths')
axes[1].legend()
axes[1].grid(alpha=0.3)

# Major quake probability
axes[2].plot(df_monthly.index[-24:], df_monthly['major_quake'].iloc[-24:], label='Historical',
color='navy')
axes[2].plot(future_dates, (df_monthly['major_quake'].iloc[-1] + np.cumsum(future_forecast_inv[:,2])) >
0.5,
        label='Forecast', marker='o', color='darkorange')
axes[2].set_title('Major Earthquake Probability')
axes[2].set_ylabel('Probability')
```

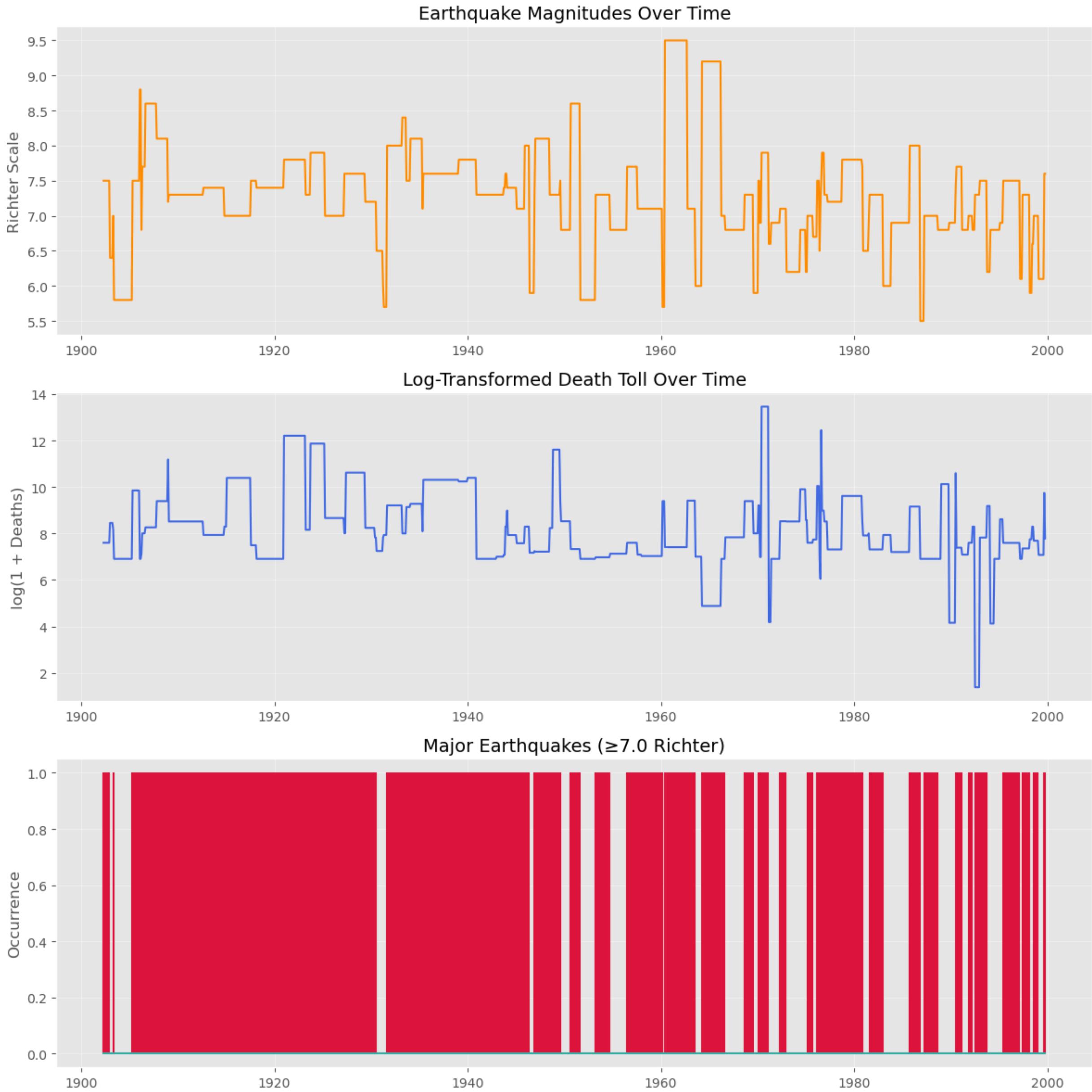
```
axes[2].legend()
axes[2].grid(alpha=0.3)

plt.tight_layout()
plt.show()

# Print numerical forecasts
print("\n12-Month Forecast Summary:")
forecast_df = pd.DataFrame({
    'Date': future_dates,
    'Magnitude': df_monthly['richter'].iloc[-1] + np.cumsum(future_forecast_inv[:,0]),
    'Death_Estimate': np.expm1(df_monthly['log_deaths'].iloc[-1] +
np.cumsum(future_forecast_inv[:,1])),
    'Major_Quake_Probability': 1/(1+np.exp(-(df_monthly['major_quake'].iloc[-1] +
np.cumsum(future_forecast_inv[:,2]))))
})

print(forecast_df.round(2))
```

OUTPUT:



Earthquake Magnitudes Over Time
Richter Scale
9.5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
5.5
1900
1920
1940
1960
1980
2000
Log-Transformed Death Toll Over Time
log(1 + Deaths)
14
12
10
8
6
4
2
1900
1920
1940
1960
1980
2000
Major Earthquakes (≥7.0 Richter)
Occurrence
1.0
0.8
0.6
0.4
0.2
0.0
1900
1920
1940
1960
1980
2000

RESULT:

Thus, Development vector auto regression model for multivariate time series data forecasting. is executed successfully.