Vector auto-regression model for multivariate time series data forecasting

Aim:
To write a python program for implementing a vector auto regression model for multivariate time series data forecasting.

Algorithm:

1. **Import Libraries and Load Data:** Begin by importing necessary Python libraries such as pandas, numpy, matplotlib, and statsmodels. Load the multivariate time series dataset, for instance, the U.S. macroeconomic data from statsmodels.
2. **Preprocess Data**: Convert the 'year' and 'quarter' columns into a datetime format to create a proper time index. Set this datetime column as the index of the DataFrame. Select relevant variables (e.g., real GDP, real consumption, real investment) for analysis.
3. **Visualize Time Series**: Plot the selected variables to understand their trends and seasonal patterns. This helps in assessing the stationarity and identifying any anomalies in the data.
4. **Split Data into Training and Testing Sets**: Divide the dataset into training and testing subsets, typically using an 80/20 split. The training set is used to fit the model, while the testing set evaluates its forecasting performance.
5. **Determine Optimal Lag Order**: Use criteria like the Akaike Information Criterion (AIC) to select the optimal lag length for the VAR model. This involves fitting the model with different lag orders and choosing the one with the lowest AIC value.
6. **Fit the VAR Model**: Using the selected lag order, fit the VAR model on the training data. This step estimates the coefficients that capture the relationships between the variables over time.
7. **Forecast Future Values**: Utilize the fitted VAR model to forecast future values of the variables over the testing period. This involves generating predictions based on the model's equations and the most recent observations.
8. **Visualize Forecasts vs. Actual Data**: Plot the forecasted values alongside the actual observed values from the testing set. This comparison helps in evaluating the accuracy and effectiveness of the VAR model in capturing the dynamics of the multivariate time series.

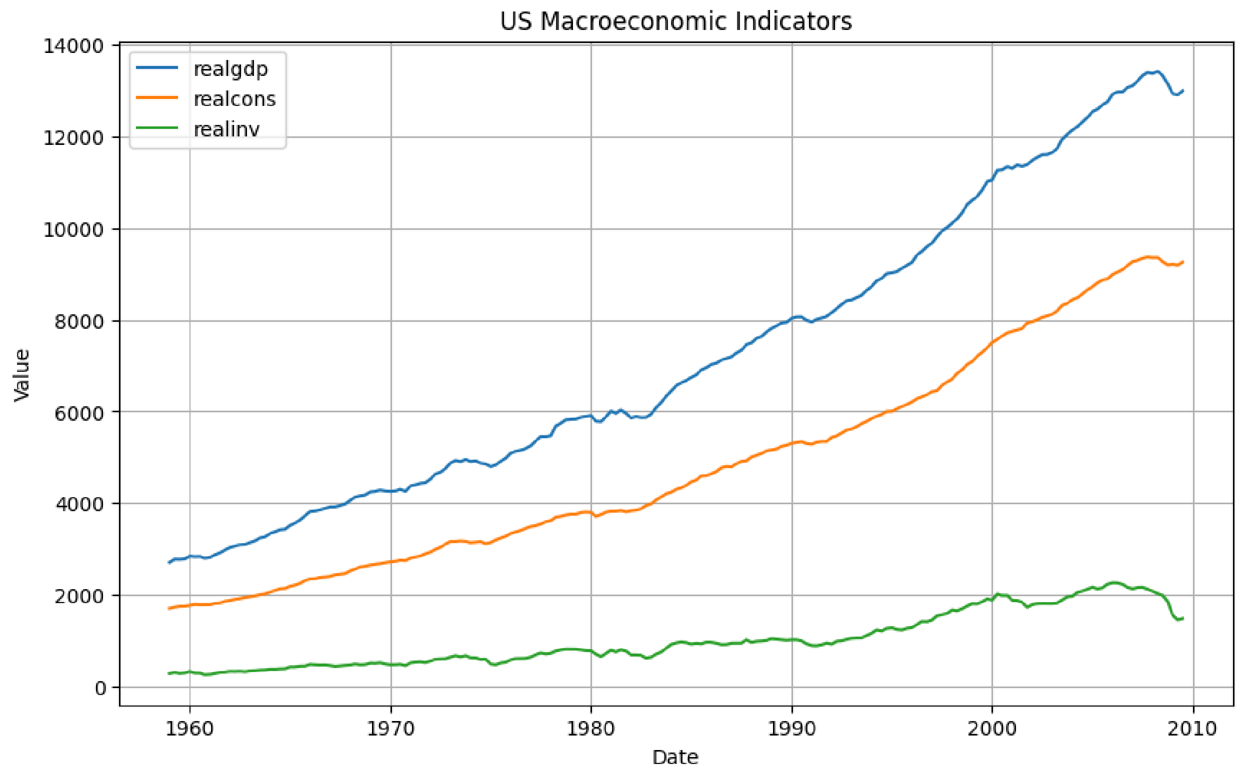
Program Code:

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.api import VAR
import statsmodels.api as sm

# Load the U.S. macrodata dataset (quarterly data)
data = sm.datasets.macrodata.load_pandas().data

# Display the first few rows
print("Raw Macrodata:")
print(data.head())
```

```
Raw Macrodata:
     year  quarter   realgdp  realcons  realinv  realgovt  realdpi    cpi  \
0  1959.0      1.0  2710.349    1707.4  286.898   470.045   1886.9  28.98
1  1959.0      2.0  2778.801    1733.7  310.859   481.301   1919.7  29.15
2  1959.0      3.0  2775.488    1751.8  289.226   491.260   1916.4  29.35
3  1959.0      4.0  2785.204    1753.7  299.356   484.052   1931.3  29.37
4  1960.0      1.0  2847.699    1770.5  331.722   462.199   1955.5  29.54

      m1  tbilrate  unemp      pop  infl  realint
0  139.7      2.82    5.8  177.146  0.00     0.00
1  141.7      3.08    5.1  177.830  2.34     0.74
2  140.5      3.82    5.3  178.657  2.74     1.09
3  140.0      4.33    5.6  179.386  0.27     4.06
4  139.6      3.50    5.2  180.007  2.31     1.19
```

```
# Define a helper function to convert year and quarter into a date (first day of the quarter)
def convert_to_date(year, quarter):
    year = int(year)
```

```
    quarter = int(quarter)
    month = (quarter - 1) * 3 + 1
    return pd.Timestamp(year=year, month=month, day=1)

# Apply the conversion and set the new datetime index
data['date'] = data[['year', 'quarter']].apply(lambda x: convert_to_date(x['year'], x['quarter']),
axis=1)
data.set_index('date', inplace=True)

# Select a subset of variables for VAR modeling (e.g., real GDP, real consumption, and real
investment)
data = data[['realgdp', 'realcons', 'realinv']]

# Plot the selected variables to inspect the multivariate time series
plt.figure(figsize=(10, 6))
plt.plot(data)
plt.legend(data.columns)
plt.title("US Macroeconomic Indicators")
plt.xlabel("Date")
plt.ylabel("Value")
plt.grid(True)
plt.show()
```




```
# Use an 80/20 split for training and testing
n_obs = int(len(data) * 0.8)
train = data.iloc[:n_obs]
test = data.iloc[n_obs:]

print("Training data shape:", train.shape)
print("Testing data shape:", test.shape)
```

```
Training data shape: (162, 3)
Testing data shape: (41, 3)
```

```
# Initialize the VAR model with the training data
model = VAR(train)

# Select the optimal lag order using AIC (you can set maxlags as needed)
lag_order_results = model.select_order(maxlags=8)
```

```
print("Lag Order Selection (AIC):")
print(lag_order_results.summary())

# Use the optimal lag order determined by AIC
optimal_lag = lag_order_results.selected_orders['aic']
print("Optimal lag order according to AIC:", optimal_lag)

# Fit the VAR model using the selected lag order
var_model = model.fit(optimal_lag)
print(var_model.summary())
```

```
Lag Order Selection (AIC):
 VAR Order Selection (* highlights the minimums)
==================================================
     AIC        BIC        FPE        HQIC
-----------------------------------------------
0      31.61      31.67   5.349e+13      31.63
1      19.32      19.56   2.455e+08      19.42
2     19.10*     19.51*  1.966e+08*     19.26*
3      19.15      19.74   2.073e+08      19.39
4      19.17      19.94   2.124e+08      19.49
5      19.21      20.16   2.210e+08      19.60
6      19.21      20.34   2.218e+08      19.67
7      19.27      20.57   2.350e+08      19.80
8      19.30      20.78   2.442e+08      19.91
-----------------------------------------------
Optimal lag order according to AIC: 2
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Tue, 15, Apr, 2025
Time:                     08:57:32
--------------------------------------------------------------------
No. of Equations:         3.00000    BIC:                   19.4596
Nobs:                     160.000    HQIC:                  19.2199
```

Log likelihood: -2184.57 FPE: 1.88794e+08

AIC: 19.0560 Det(Omega_mle): 1.66034e+08

Results for equation realgdp

	coefficient	std. error	t-stat	prob
const	91.110405	25.761716	3.537	0.000
L1.realgdp	0.573321	0.175263	3.271	0.001
L1.realcons	1.075266	0.203010	5.297	0.000
L1.realinv	0.440488	0.203149	2.168	0.030
L2.realgdp	0.183206	0.177101	1.034	0.301
L2.realcons	-0.730723	0.222235	-3.288	0.001
L2.realinv	-0.379408	0.207671	-1.827	0.068

Results for equation realcons

	coefficient	std. error	t-stat	prob
const	43.252961	15.770705	2.743	0.006
L1.realgdp	-0.198788	0.107292	-1.853	0.064
L1.realcons	1.330738	0.124278	10.708	0.000
L1.realinv	0.287303	0.124363	2.310	0.021
L2.realgdp	0.100948	0.108417	0.931	0.352
L2.realcons	-0.199575	0.136047	-1.467	0.142
L2.realinv	-0.210766	0.127131	-1.658	0.097

Results for equation realinv

	coefficient	std. error	t-stat	prob
const	14.794737	16.672884	0.887	0.375

```
L1.realgdp       -0.211777       0.113429        -1.867         0.062
L1.realcons       0.849495       0.131387         6.466         0.000
L1.realinv        1.184897       0.131477         9.012         0.000
L2.realgdp        0.128031       0.114619         1.117         0.264
L2.realcons      -0.720398       0.143830        -5.009         0.000
L2.realinv       -0.223338       0.134404        -1.662         0.097
============================================================================
======

Correlation matrix of residuals
            realgdp  realcons   realinv
realgdp     1.000000  0.580128  0.704673
realcons    0.580128  1.000000  0.053055
realinv     0.704673  0.053055  1.000000
```

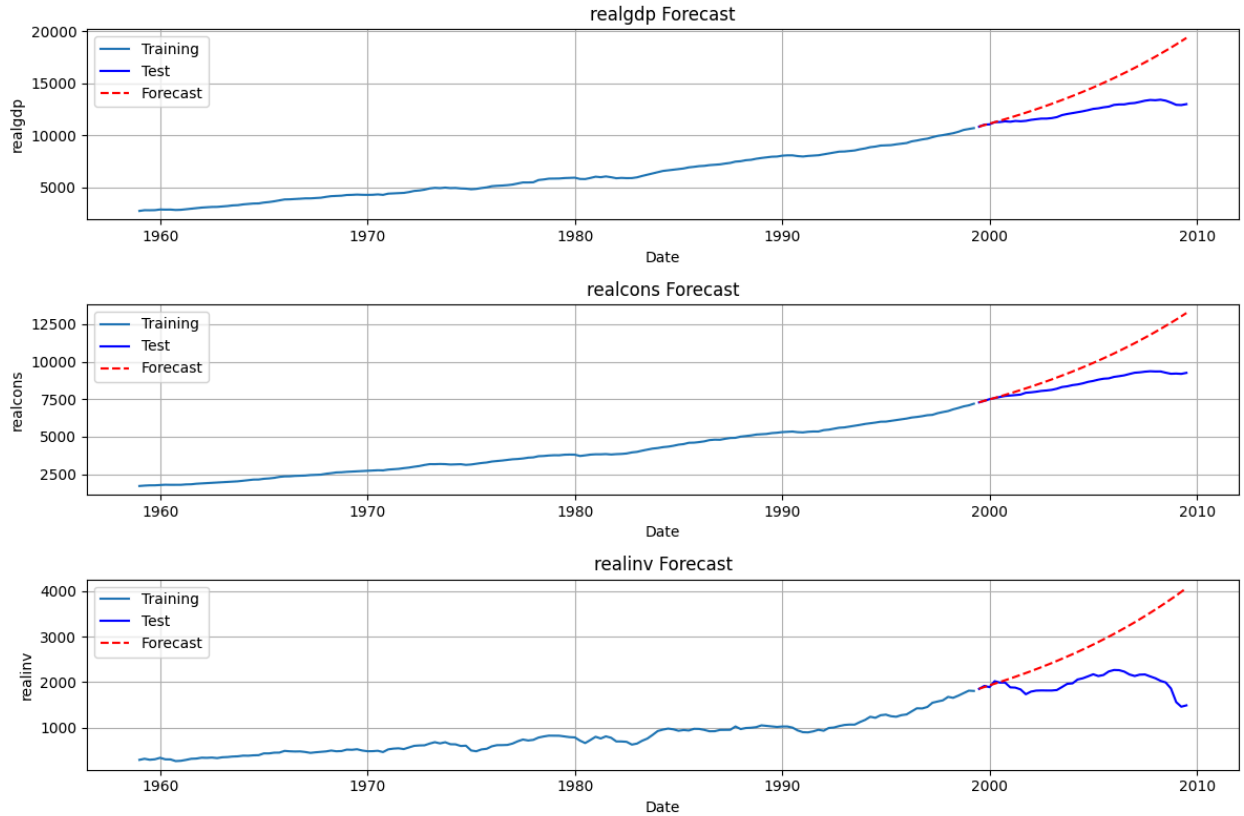
```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency QS-OCT will be used.
  self._init_dates(dates, freq)
```

```python
# Determine the number of lags used in the model
lag_order = var_model.k_ar

# Prepare the forecast input using the last 'lag_order' observations from the training set
forecast_input = train.values[-lag_order:]

# Forecast the future values for the entire test period
steps = len(test)
forecast = var_model.forecast(y=forecast_input, steps=steps)

# Convert the forecast array into a DataFrame with the same columns as the original dataset
forecast_df = pd.DataFrame(forecast, index=test.index, columns=test.columns)

print("Forecasted values:")
```

```
print(forecast_df.head())
```

```
Forecasted values:
                realgdp     realcons      realinv
date
1999-07-01  10828.699243  7292.802045  1852.238589
1999-10-01  10968.947922  7389.053080  1889.693539
2000-01-01  11109.538751  7485.844508  1926.883935
2000-04-01  11251.751790  7584.439872  1963.652534
2000-07-01  11396.416968  7684.974591  2000.823824
```

```
# Plot the forecasted values along with the actual test data for each variable
plt.figure(figsize=(12, 8))
for i, col in enumerate(test.columns, 1):
    plt.subplot(len(test.columns), 1, i)
    plt.plot(train.index, train[col], label="Training")
    plt.plot(test.index, test[col], label="Test", color='blue')
    plt.plot(forecast_df.index, forecast_df[col], label="Forecast", color='red', linestyle='--')
    plt.title(f"{col} Forecast")
    plt.xlabel("Date")
    plt.ylabel(col)
    plt.legend(loc='upper left')
    plt.grid(True)
plt.tight_layout()
plt.show()
```



RESULTS:

The program has been created and implemented successfully for implementing a vector auto regression model for multivariate time series data forecasting.