In [ ]:

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Dec  5 11:50:28 2023

@author: Majd Rabbaj
"""

import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller, coint
from statsmodels.tsa.api import VAR
from pandas_datareader import data as pdr
import datetime
from fredapi import Fred
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.vector_ar.vecm import coint_johansen
from statsmodels.tsa.api import VAR
import numpy as np
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.api import SVAR

#%% Data
api_key = '230970faf44ea208229d77dff9f995f3'
fred = Fred(api_key=api_key)

# Define the series IDs
series_ids = {


    'USGDP': 'GDPC1',
    'ALLUKSHARES': 'SPASTT01GBM661N',
    'USD/GBP': 'DEXUSUK',
    'UKCPI': 'GBRCPIALLMINMEI',
    'UKGDP':'UKNGDP'
}

# Define the observation period
start_date = '1990-01-01'
end_date = '2023-11-30'

# Download the data
```

```python
data = {}
for series_name, series_id in series_ids.items():
    data[series_name] = fred.get_series(series_id, start_date, end_date, frequency='q')

# Transform the series (log)
for series_name, series_data in data.items():
    data[series_name] = pd.Series(data[series_name])

# Create a DataFrame from the log-transformed data
df = pd.DataFrame(data)
df=np.log(df.dropna())

csv_file_path = 'C:/Users/acer/Desktop/file.csv'
df.to_csv(csv_file_path, index=True)

#%% ADF
df.dtypes

for column in df.columns:
    plt.figure(figsize=(10, 4))
    plt.plot(df.index, df[column], label=column)
    plt.title(f"Time Series Plot for {column}")
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

# Performing Augmented Dickey-Fuller test on each column
adf_results = {}
for column in df.columns:
    result = adfuller(df[column])
    adf_results[column] = {
        'ADF Statistic': result[0],
        'p-value': result[1],
        'Critical Values': result[4]
    }

    print(f"ADF Test Result for {column}:")
    print(f"ADF Statistic: {result[0]}")
    print(f"p-value: {result[1]}")
    for key, value in result[4].items():
        print(f'Critical Value ({key}): {value}')
    print("\n")
```

```python
# Applying first difference to all columns except the first index column
df_diff = df.iloc[:,:].diff().dropna()

adf_results_diff = {}
for column in df_diff.columns:
    result = adfuller(df_diff[column])
    adf_results_diff[column] = {
        'ADF Statistic': result[0],
        'p-value': result[1],
        'Critical Values': result[4]
    }

    print(f"ADF Test Result for {column}:")
    print(f"ADF Statistic: {result[0]}")
    print(f"p-value: {result[1]}")
    for key, value in result[4].items():
        print(f'Critical Value ({key}): {value}')
    print("\n")


#%% Cointegration ?

# Create a VAR model
model = VAR(df_diff)  # Assuming df_diff is your differenced DataFrame

# Select the optimal lag order with various information criteria
results_aic = model.select_order(maxlags=15)
print('AIC Selection:', results_aic.aic)
print('BIC Selection:', results_aic.bic)
print('HQIC Selection:', results_aic.hqic)

johansen_test = coint_johansen(df, det_order=0, k_ar_diff=5)

# Trace statistic
print('Eigenvalues:', johansen_test.eig)
print('Trace statistic:', johansen_test.lr1)
print('Critical values (90%, 95%, 99%):', johansen_test.cvt)

# Max Eigenvalues statistic
print('Max Eigenvalue statistic:', johansen_test.lr2)
print('Critical values (90%, 95%, 99%):', johansen_test.cvm)

# Display the estimated cointegration equations (eigenvectors)
```

```python
print('Eigenvectors (Cointegration equations):')
print(johansen_test.evec)



#%% As no cointegration / VAR + Estimation + Causality tests
model = VAR(df_diff)
results = model.select_order(maxlags=15)

print('AIC:', results.aic)
print('BIC:', results.bic)
print('HQIC:', results.hqic)

#Estimation
var_model = model.fit(5)

print(var_model.summary())

if var_model.is_stable():
    print("VAR stable.")
else:
    print("VAR not stable.")

#Ljung-Box Residuals
for col in df_diff.columns:
    lb_test = acorr_ljungbox(var_model.resid[col], lags=[10])
    print(f"Résultats du test de Ljung-Box pour {col}:")
    print(lb_test)
    print()

#Forecast Error Variance Decomposition
fevd = var_model.fevd(10)
fevd.plot()
plt.show()
plt.savefig('fevd_plot.png')

# Granger causality
variables = df_diff.columns

for var1 in variables:
    for var2 in variables:
        if var1 != var2:
            print(f'Granger Causality test from {var2} to {var1}:')
            test_result = var_model.test_causality(var1, [var2], kind='f')
```

```python
            print(test_result.summary())
            print()

#Forecast
lagged_values = df_diff.values[-5:]

forecast = var_model.forecast(lagged_values, steps=10)

# Forecast index
num_periods = 10
last_date = df_diff.index[-1]
forecast_index = pd.date_range(start=last_date, periods=num_periods + 1, freq='Q')[1:]

# Forecast dataframe
forecast_df = pd.DataFrame(forecast, index=forecast_index, columns=df_diff.columns)
print(forecast_df)

forecast_df.plot()
plt.title("VAR Forecast")
plt.show()

#Forecast orignial values
last_values = df.iloc[-1]

reintegrated_forecast_df = pd.DataFrame(index=forecast_df.index, columns=forecast_df.columns)

for col in forecast_df.columns:
    reintegrated_forecast_df[col] = last_values[col] + forecast_df[col].cumsum()

print(reintegrated_forecast_df)

for col in reintegrated_forecast_df.columns:
    plt.figure(figsize=(10, 4))
    plt.plot(df[col], label=f'Original {col}')
    plt.plot(reintegrated_forecast_df[col], label=f'Forecast {col}', linestyle='--')
    plt.title(f"VAR forecast for {col}")
    plt.legend()
    plt.show()

#%%
#Cholesky decomposition
cholesky_matrix = np.linalg.cholesky(var_model.sigma_u).T

# Display the Cholesky matrix
```

```python
print(cholesky_matrix)

#Impulse Response
irf = var_model.irf(5)
irf.plot(orth=True)
plt.show()

irf = var_model.irf(10)
n_vars = len(irf.model.names)
for i in range(n_vars):
    irf.plot(orth=True, response=irf.model.names[i])
    plt.show()

#VAR in levels
model_2 = VAR(df)
var_model_2 = model_2.fit(5)

print(var_model_2.summary())

irf = var_model_2.irf(5)
irf.plot(orth=True)
plt.show()

irf = var_model_2.irf(10)
n_vars = len(irf.model.names)
for i in range(n_vars):
    irf.plot(orth=True, response=irf.model.names[i])
    plt.show()

#SVAR
# Ensure the A matrix dimensions match the number of variables in df_diff
n_vars = df_diff.shape[1]  # Number of variables

# Define the A matrix (identity matrix as an example)
A = np.eye(n_vars)

# Create and fit the SVAR model
svar_model = SVAR(df_diff, svar_type='A', A=A)
svar_results = svar_model.fit(maxlags=5)

# Print the model summary
print(svar_results.summary())

# Plot Impulse Response Functions
```

```
irf = svar_results.irf(10)
irf.plot(orth=True)
plt.show()
```