VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A6b- Time Series Analysis

(Part – B)

ADHYAYAN AMIT JAIN

V01109421

Date of Submission: 25-07-2024

CONTENTS

Advanced Time Series Analysis of Commodity Prices with VAR and VECM Models

INTRODUCTION

In this assignment, we delve into the advanced analysis of commodity price data using sophisticated econometric techniques, specifically Vector Autoregression (VAR) and Vector Error Correction Models (VECM). Our focus is on a diverse set of commodities, including Oil, Sugar, Gold, Silver, Wheat, and Soybean. This analysis leverages data sourced from the World Bank's Pink Sheet, a reputable repository of global commodity prices.Commodity prices are pivotal indicators in economic analysis, reflecting supply and demand dynamics, market expectations, and macroeconomic conditions. The ability to model and forecast these prices is essential for various stakeholders, including policymakers, investors, and businesses. By applying VAR and VECM models, this assignment aims to uncover both short-term and long-term relationships among commodity prices, providing insights into their interconnected behavior.

Vector Autoregression (VAR) is a powerful tool for capturing the linear interdependencies among multiple time series. It allows us to understand how each commodity's price influences and is influenced by the prices of other commodities over time. VAR models are particularly useful when the data exhibits interdependencies but does not necessarily exhibit co-integrating relationships.

Vector Error Correction Models (VECM), on the other hand, extend VAR models to account for co-integration among the time series. Co-integration indicates that while individual time series may be non-stationary, they share a common long-term equilibrium relationship. VECM models are adept at analyzing these long-term relationships and correcting short-term deviations from this equilibrium. They provide a framework for understanding how the commodities return to their equilibrium state over time.The data for this analysis is derived from the World Bank's Pink Sheet, which provides comprehensive and up-to-date commodity price information. This data serves as a crucial foundation for our modeling efforts, allowing for an accurate and meaningful analysis of price dynamics across the selected commodities.

In summary, this assignment employs VAR and VECM models to explore the intricate relationships between commodity prices. By doing so, it seeks to enhance our understanding of how these prices interact, both in the short term and in the long term, thereby offering valuable insights into market behavior and trends.

OBJECTIVES

The primary objectives of this assignment are as follows:

1. **Model Short-term Dynamics with VAR:**
   - Utilize Vector Autoregression (VAR) to capture and analyze the interdependencies among the prices of selected commodities.
   - Understand how each commodity's price influences and is influenced by the prices of other commodities over time.
   - Determine the optimal lag length for the VAR model to accurately reflect the relationships among the commodities.
2. **Identify Long-term Relationships with VECM:**
   - Apply the Vector Error Correction Model (VECM) to explore potential long-term equilibrium relationships among the commodities.
   - Test for co-integration among the time series to assess whether the commodities share a common long-term trend.
   - Estimate the number of co-integrating relationships (r) and analyze how deviations from equilibrium are corrected over time.
3. **Forecast Future Commodity Prices:**
   - Generate forecasts for future commodity prices using both the VAR and VECM models.
   - Compare and contrast the forecasting performance of the two models to determine which provides more accurate predictions.
   - Visualize and interpret the forecasted values to assess potential future trends in commodity prices.
4. **Conduct Statistical Tests and Validation:**

- Perform stationarity tests (e.g., Augmented Dickey-Fuller test) to ensure that the data series meet the assumptions required for VAR and VECM modeling.
- Validate the models through diagnostic checks and robustness tests to ensure the reliability of the results.

5. **Provide Insights and Implications:**
    - Analyze the findings to draw insights about the behavior and interaction of commodity prices.
    - Discuss the implications of these insights for stakeholders such as policymakers, investors, and businesses.
    - Offer recommendations based on the analysis to aid in decision-making and strategic planning.

BUSINESS SIGNIFICANCE

The analysis of commodity prices through VAR and VECM models holds substantial business significance across several sectors. Understanding the dynamics and relationships among commodity prices is crucial for various stakeholders, including investors, businesses, policymakers, and analysts. Here's how this analysis can impact different aspects of business and economic decision-making:

1. **Investment Decisions:**
    - **Informed Investment Strategies:** Investors can use the insights from VAR and VECM models to make informed decisions regarding portfolio allocation. By understanding the interdependencies and long-term relationships between commodity prices, investors can better anticipate price movements and adjust their investment strategies accordingly.
    - **Risk Management:** Analyzing the forecasted trends and volatility of commodity prices helps in assessing potential risks. Investors can use this information to hedge against unfavorable price movements and manage financial risks more effectively.
2. **Business Planning and Strategy:**
    - **Cost Management:** Companies that rely on commodities as raw materials (e.g., manufacturing and agriculture) can benefit from

    understanding price forecasts to plan their procurement strategies. Accurate forecasts enable businesses to manage costs, negotiate better contracts, and budget effectively.
    - **Pricing Strategies:** Firms in the commodity supply chain can use insights from the analysis to set competitive pricing strategies. Understanding how prices of related commodities influence each other helps in formulating pricing strategies that align with market conditions.
3. **Supply Chain Management:**
    - **Inventory Management:** Businesses involved in the trading or processing of commodities can use price forecasts to optimize inventory levels. By anticipating price changes, companies can make strategic decisions about stock levels and timing of purchases to minimize costs and maximize profitability.
    - **Supplier Relationships:** Insights into price trends can improve negotiations with suppliers. Businesses can leverage forecast data to negotiate favorable terms and mitigate the impact of price fluctuations on their supply chain.
4. **Policy and Regulation:**
    - **Economic Policy Formulation:** Policymakers can use the findings from this analysis to understand the economic impact of commodity price fluctuations. This understanding aids in the formulation of policies that stabilize markets, manage inflation, and support economic growth.
    - **Regulatory Measures:** The analysis helps in designing regulatory measures to address volatility in commodity markets. Policymakers can implement interventions that mitigate extreme price swings and ensure market stability.
5. **Strategic Decision-Making:**
    - **Market Entry and Expansion:** Companies considering entry into new markets or expansion in existing markets can benefit from understanding commodity price trends. Insights into price relationships and forecasts assist in evaluating market potential and making strategic expansion decisions.
    - **Competitive Analysis:** Businesses can use the analysis to monitor competitors and anticipate their pricing and supply chain strategies. Understanding how commodity prices impact competitors helps in developing competitive strategies and gaining a market edge.

In summary, the business significance of analyzing commodity prices through VAR and VECM models lies in its ability to provide actionable insights for investment, operational, strategic, and policy-related decisions.

RESULTS AND INTERPRETATIONS

Python Language

Complete Analysis of Time Series Analysis Code

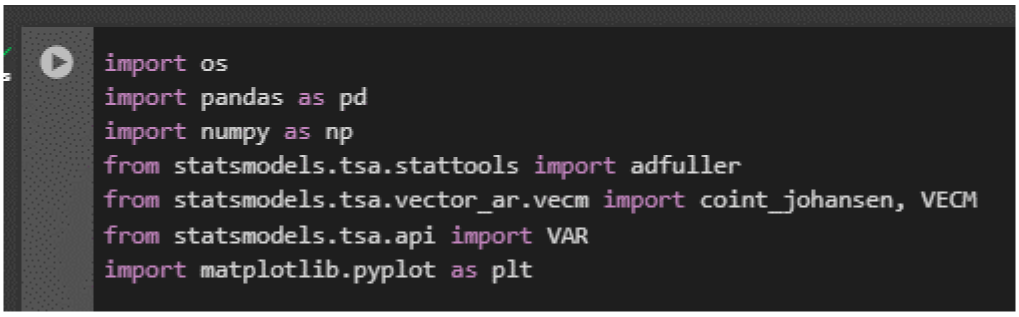
1. Importing Libraries

```
import os
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.vector_ar.vecm import coint_johansen, VECM
from statsmodels.tsa.api import VAR
import matplotlib.pyplot as plt
```

Description:

- `os`: Provides functions to interact with the operating system, though it's not used in this script.
- `pandas` (pd): A powerful library for data manipulation and analysis, especially with DataFrames.
- `numpy` (np): Provides support for numerical operations and mathematical functions.
- `adfuller`: Function to perform the Augmented Dickey-Fuller (ADF) test, used for testing stationarity of time series data.
- `coint_johansen`: Function to perform Johansen's co-integration test.
- `VECM`: Class for fitting Vector Error Correction Models, which handle non-stationary data with co-integration.
- `VAR`: Class for fitting Vector Autoregressive Models, used when there's no co-integration.
- `matplotlib.pyplot` (plt): Used for creating static, interactive, and animated visualizations in Python.

Purpose: These libraries are essential for handling data, performing statistical tests, fitting models, and visualizing results.



```
import os
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.vector_ar.vecm import coint_johansen, VECM
from statsmodels.tsa.api import VAR
import matplotlib.pyplot as plt
```

---

2. Loading the Dataset

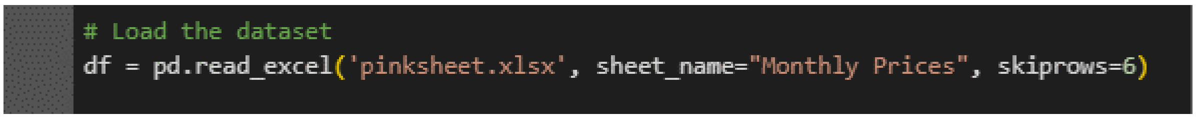
```
df = pd.read_excel('pinksheet.xlsx', sheet_name="Monthly Prices",
skiprows=6)
```

Description:

- Reads the Excel file `pinksheet.xlsx`, specifically from the sheet "Monthly Prices", and skips the first 6 rows to adjust for headers or metadata.

Purpose: To load the commodity price data into a DataFrame `df` for further processing and analysis.

Interpretation: Initial step to bring the data into a Python environment, making it ready for cleaning and analysis.



```
# Load the dataset
df = pd.read_excel('pinksheet.xlsx', sheet_name="Monthly Prices", skiprows=6)
```

---

3. Data Preprocessing

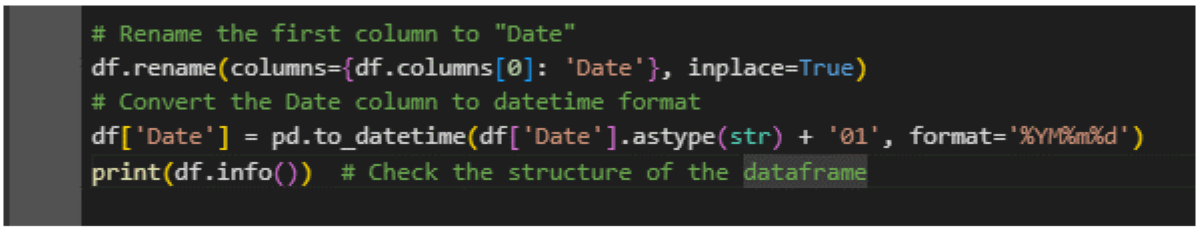
```
df.rename(columns={df.columns[0]: 'Date'}, inplace=True)
df['Date'] = pd.to_datetime(df['Date'].astype(str) + '01',
format='%YM%m%d')
print(df.info())
```

Description:

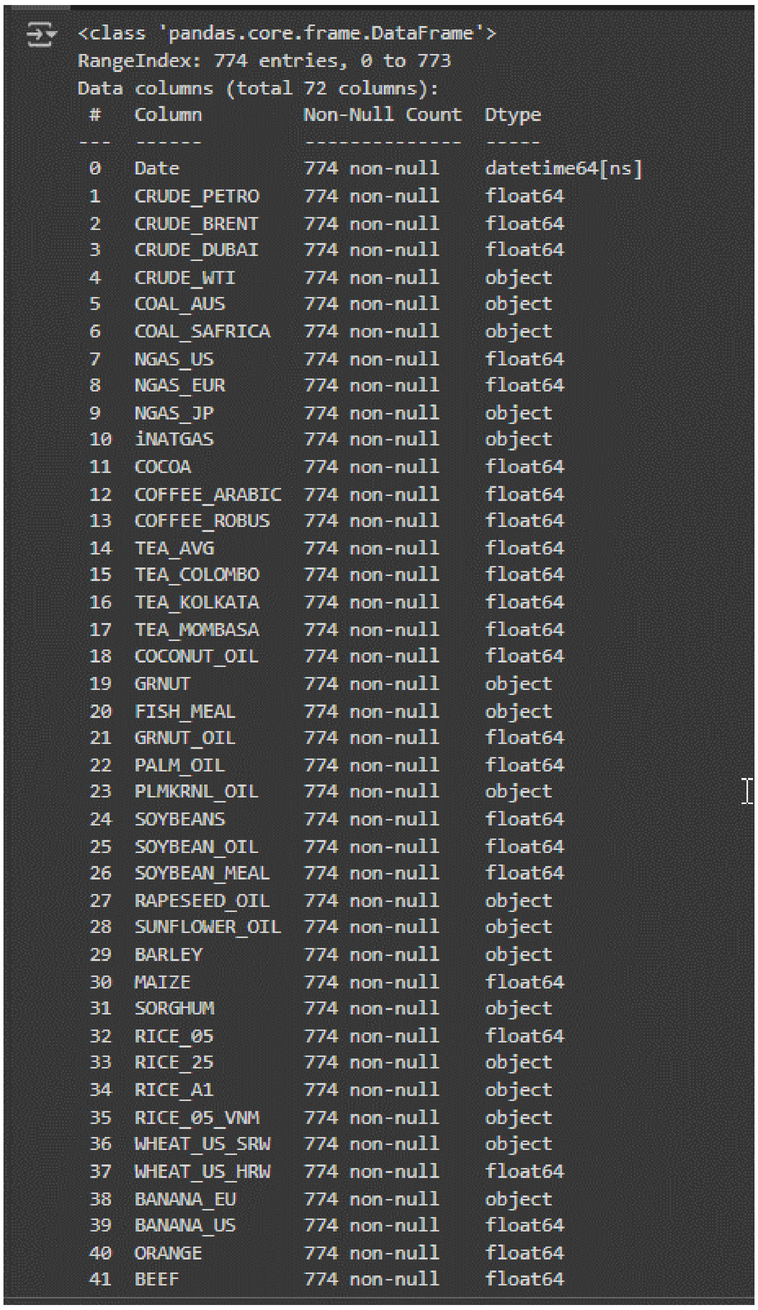
- `df.rename(columns={df.columns[0]: 'Date'}, inplace=True)`: Renames the first column to 'Date' for clarity.
- `df['Date'] = pd.to_datetime(df['Date'].astype(str) + '01', format='%YM%m%d')`: Converts the 'Date' column from a string to a datetime object, appending '01' to ensure a complete date (assuming data is monthly and the day is always 01).
- `print(df.info())`: Prints a summary of the DataFrame to verify changes.

Purpose: To standardize the date format and ensure the DataFrame has the correct column names and data types.

Interpretation: Preparing the date column for time series analysis, ensuring that the data is correctly formatted and suitable for further steps.



```
# Rename the first column to "Date"
df.rename(columns={df.columns[0]: 'Date'}, inplace=True)
# Convert the Date column to datetime format
df['Date'] = pd.to_datetime(df['Date'].astype(str) + '01', format='%YM%m%d')
print(df.info())  # Check the structure of the dataframe
```


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 774 entries, 0 to 773
Data columns (total 72 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           774 non-null    datetime64[ns]
 1   CRUDE_PETRO    774 non-null    float64
 2   CRUDE_BRENT    774 non-null    float64
 3   CRUDE_DUBAI    774 non-null    float64
 4   CRUDE_WTI      774 non-null    object
 5   COAL_AUS       774 non-null    object
 6   COAL_SAFRICA   774 non-null    object
 7   NGAS_US        774 non-null    float64
 8   NGAS_EUR       774 non-null    float64
 9   NGAS_JP        774 non-null    object
 10  iNATGAS        774 non-null    object
 11  COCOA          774 non-null    float64
 12  COFFEE_ARABIC  774 non-null    float64
 13  COFFEE_ROBUS   774 non-null    float64
 14  TEA_AVG        774 non-null    float64
 15  TEA_COLOMBO    774 non-null    float64
 16  TEA_KOLKATA    774 non-null    float64
 17  TEA_MOMBASA    774 non-null    float64
 18  COCONUT_OIL    774 non-null    float64
 19  GRNUT          774 non-null    object
 20  FISH_MEAL      774 non-null    object
 21  GRNUT_OIL      774 non-null    float64
 22  PALM_OIL       774 non-null    float64
 23  PLMKRNL_OIL    774 non-null    object
 24  SOYBEANS       774 non-null    float64
 25  SOYBEAN_OIL    774 non-null    float64
 26  SOYBEAN_MEAL   774 non-null    float64
 27  RAPESEED_OIL   774 non-null    object
 28  SUNFLOWER_OIL  774 non-null    object
 29  BARLEY         774 non-null    object
 30  MAIZE          774 non-null    float64
 31  SORGHUM        774 non-null    object
 32  RICE_05        774 non-null    float64
 33  RICE_25        774 non-null    object
 34  RICE_A1        774 non-null    object
 35  RICE_05_VNM    774 non-null    object
 36  WHEAT_US_SRW   774 non-null    object
 37  WHEAT_US_HRW   774 non-null    float64
 38  BANANA_EU      774 non-null    object
 39  BANANA_US      774 non-null    float64
 40  ORANGE         774 non-null    float64
 41  BEEF           774 non-null    float64
```

4. Selecting and Cleaning Data

```
commodity = df[['Date', 'CRUDE_BRENT', 'SOYBEANS', 'GOLD', 'SILVER',
'UREA_EE_BULK', 'MAIZE']]
commodity.columns = commodity.columns.str.strip().str.lower().str.replace('
', '_').str.replace('(', '').str.replace(')', '')
print(commodity.info())
```

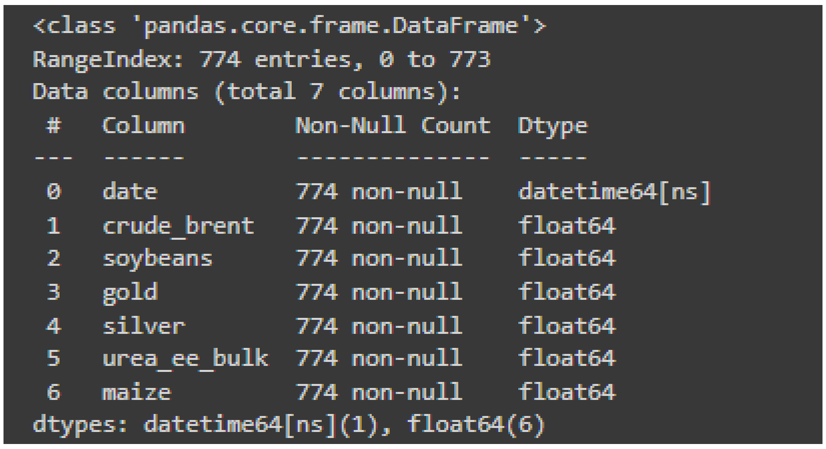
Description:

- `commodity = df[['Date', 'CRUDE_BRENT', 'SOYBEANS', 'GOLD', 'SILVER', 'UREA_EE_BULK', 'MAIZE']]`: Selects relevant columns related to commodity prices.

- `commodity.columns = commodity.columns.str.strip().str.lower().str.replace(' ', '_').str.replace('(', '').str.replace(')', '')`: Cleans column names by removing spaces, converting to lowercase, and replacing special characters with underscores.
- `print(commodity.info())`: Prints a summary of the cleaned DataFrame.

Purpose: To isolate the columns of interest and clean the column names for consistency and ease of reference.

Interpretation: Ensuring that only relevant data is used and that column names are standardized for better code readability and maintenance.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 774 entries, 0 to 773
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   date          774 non-null    datetime64[ns]
 1   crude_brent   774 non-null    float64
 2   soybeans      774 non-null    float64
 3   gold          774 non-null    float64
 4   silver        774 non-null    float64
 5   urea_ee_bulk  774 non-null    float64
 6   maize         774 non-null    float64
dtypes: datetime64[ns](1), float64(6)
```

---

5. Removing Date Column for Analysis

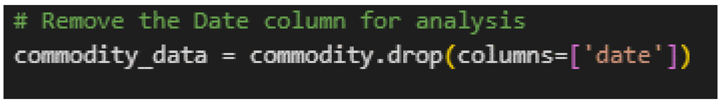
```
commodity_data = commodity.drop(columns=['date'])
```

Description:

- Removes the 'date' column from the DataFrame `commodity`, leaving only the commodity price data.

Purpose: To prepare the data for time series modeling by focusing solely on the variables of interest (commodity prices).

Interpretation: The 'date' column is not needed for statistical analysis and modeling; thus, it is excluded from `commodity_data`.



```
# Remove the Date column for analysis
commodity_data = commodity.drop(columns=['date'])
```

---

6. Testing for Stationarity

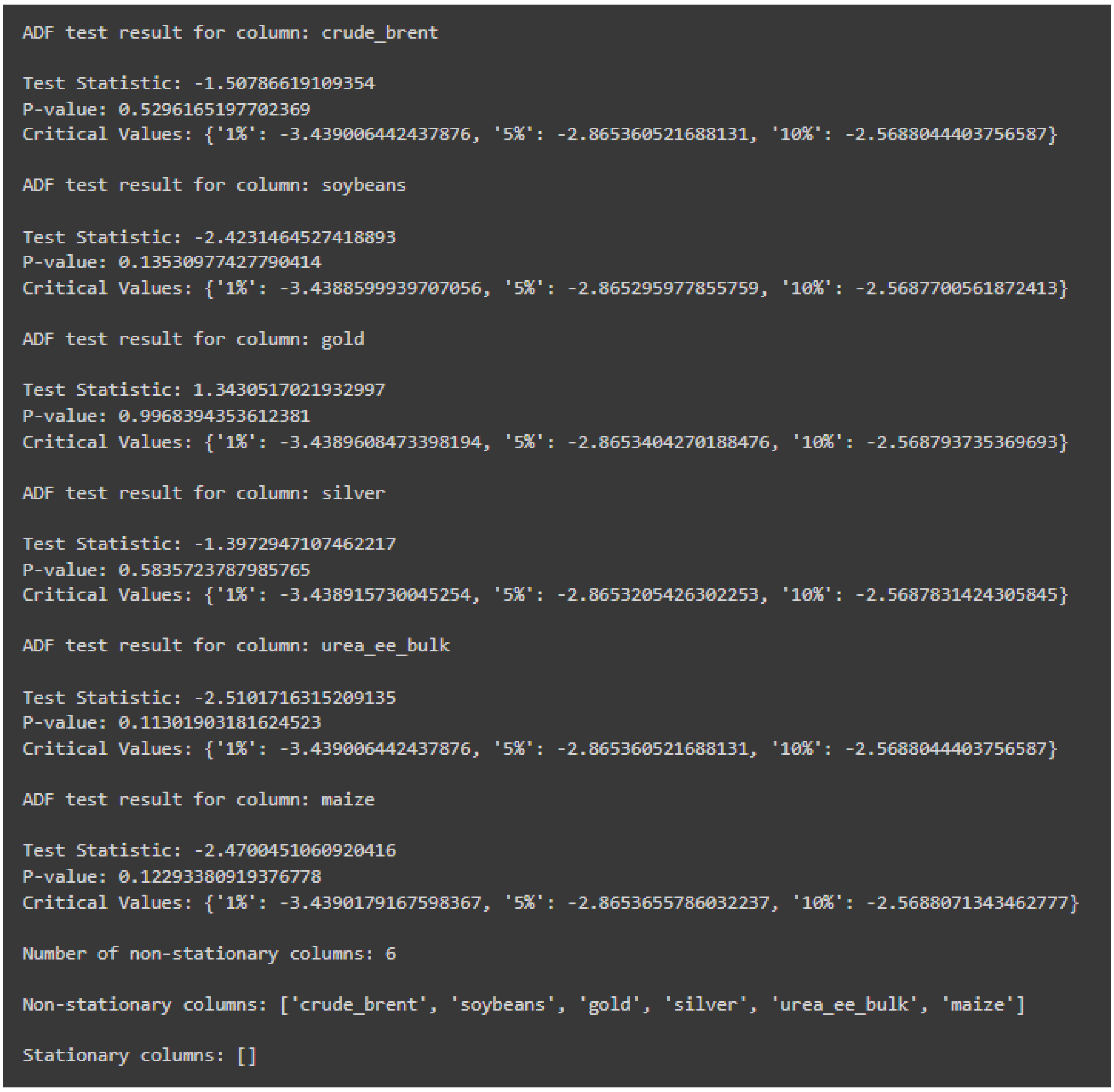
```
columns_to_test = commodity_data.columns
non_stationary_count = 0
stationary_columns = []
non_stationary_columns = []

for col in columns_to_test:
    adf_result = adfuller(commodity_data[col])
    p_value = adf_result[1]
    print(f"\nADF test result for column: {col}\n")
    print(f"Test Statistic: {adf_result[0]}")
    print(f"P-value: {p_value}")
    print(f"Critical Values: {adf_result[4]}")

    if p_value > 0.05:
        non_stationary_count += 1
        non_stationary_columns.append(col)
    else:
        stationary_columns.append(col)

print(f"\nNumber of non-stationary columns: {non_stationary_count}\n")
print(f"Non-stationary columns: {non_stationary_columns}\n")
print(f"Stationary columns: {stationary_columns}")
```

Description:

- Initializes lists and counters to keep track of stationary and non-stationary columns.
- Iterates over each column to perform the Augmented Dickey-Fuller (ADF) test.
- Extracts p-value and other statistics, and determines stationarity based on a common threshold (p-value > 0.05).
- Categorizes columns into stationary and non-stationary based on the p-value.

Purpose: To check whether each time series is stationary, which is a prerequisite for many time series models. Non-stationary series might need differencing or other transformations.

Interpretation: Stationarity is critical for modeling. Non-stationary columns are identified and will need to be addressed to ensure accurate modeling.



```
ADF test result for column: crude_brent

Test Statistic: -1.50786619109354
P-value: 0.5296165197702369
Critical Values: {'1%': -3.439006442437876, '5%': -2.865360521688131, '10%': -2.5688044403756587}

ADF test result for column: soybeans

Test Statistic: -2.4231464527418893
P-value: 0.13530977427790414
Critical Values: {'1%': -3.4388599939707056, '5%': -2.865295977855759, '10%': -2.5687700561872413}

ADF test result for column: gold

Test Statistic: 1.3430517021932997
P-value: 0.9968394353612381
Critical Values: {'1%': -3.4389608473398194, '5%': -2.8653404270188476, '10%': -2.568793735369693}

ADF test result for column: silver

Test Statistic: -1.3972947107462217
P-value: 0.5835723787985765
Critical Values: {'1%': -3.438915730045254, '5%': -2.8653205426302253, '10%': -2.5687831424305845}

ADF test result for column: urea_ee_bulk

Test Statistic: -2.5101716315209135
P-value: 0.11301903181624523
Critical Values: {'1%': -3.439006442437876, '5%': -2.865360521688131, '10%': -2.5688044403756587}

ADF test result for column: maize

Test Statistic: -2.4700451060920416
P-value: 0.12293380919376778
Critical Values: {'1%': -3.4390179167598367, '5%': -2.8653655786032237, '10%': -2.5688071343462777}

Number of non-stationary columns: 6

Non-stationary columns: ['crude_brent', 'soybeans', 'gold', 'silver', 'urea_ee_bulk', 'maize']

Stationary columns: []
```

7. Performing Co-Integration Test

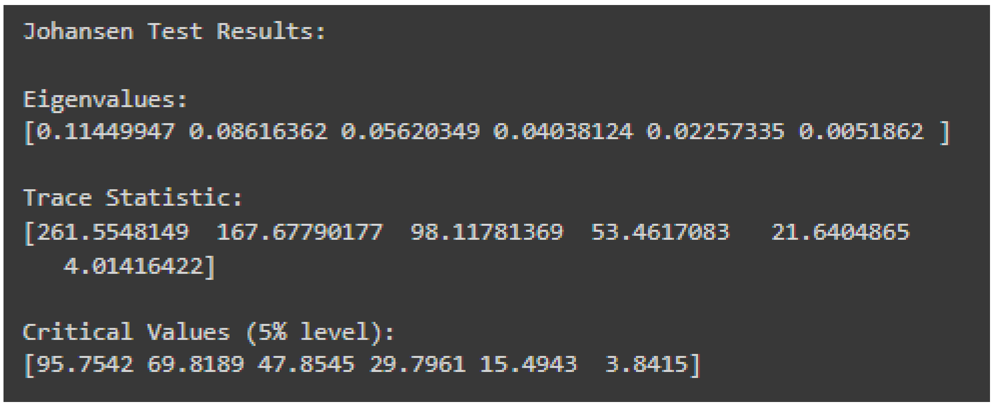
```
johansen_test = coint_johansen(commodity_data, det_order=0, k_ar_diff=1)

print("\nJohansen Test Results:\n")
print(f"Eigenvalues:\n{johansen_test.eig}\n")
print(f"Trace Statistic:\n{johansen_test.lr1}\n")
print(f"Critical Values (5% level):\n{johansen_test.cvt[:, 1]}\n")
```

Description:

- Performs Johansen's co-integration test to determine the number of co-integrating relationships among the time series.
- Outputs eigenvalues, trace statistics, and critical values for the test.

Purpose: To assess if there are long-term equilibrium relationships between the time series, which is essential for applying VECM.

Interpretation: Johansen's test helps identify the number of co-integrating vectors. The results guide the selection of the VECM model or indicate if VAR should be used instead.



```
Johansen Test Results:

Eigenvalues:
[0.11449947 0.08616362 0.05620349 0.04038124 0.02257335 0.0051862 ]

Trace Statistic:
[261.5548149  167.67790177  98.11781369  53.4617083   21.6404865
   4.01416422]

Critical Values (5% level):
[95.7542 69.8189 47.8545 29.7961 15.4943  3.8415]
```

---

8. Estimating and Forecasting with VECM

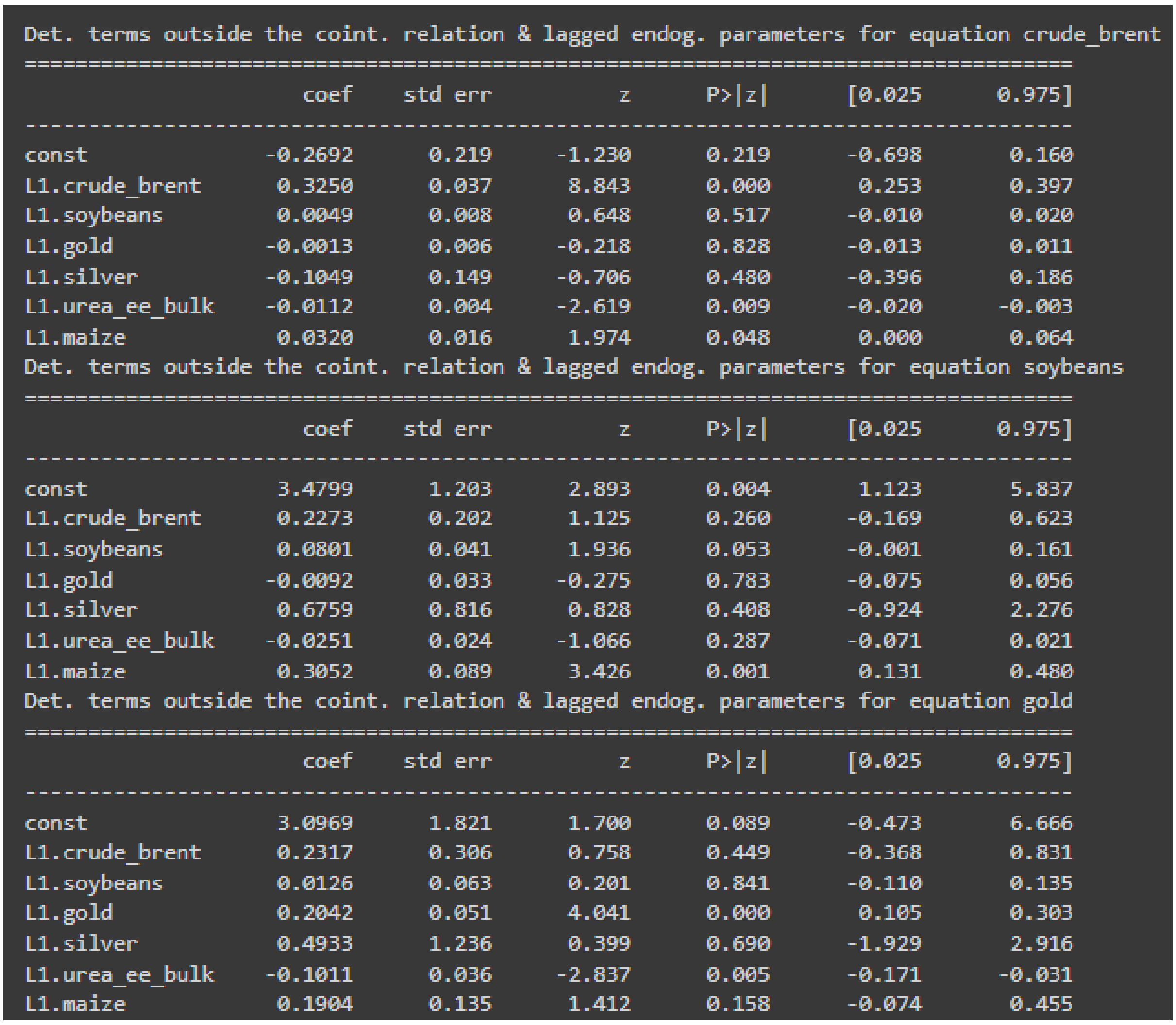
```
r = 2  # Replace with the actual number from the test results

if r > 0:
    vecm_model = VECM(commodity_data, k_ar_diff=1, coint_rank=r,
deterministic='co')
    vecm_fitted = vecm_model.fit()

    print(vecm_fitted.summary())
    print("Alpha Coefficients:\n", vecm_fitted.alpha)
    print("Beta Coefficients:\n", vecm_fitted.beta)
    print("Gamma Coefficients:\n", vecm_fitted.gamma)

    forecast = vecm_fitted.predict(steps=24)
    forecast_df = pd.DataFrame(forecast,
index=pd.date_range(start=commodity_data.index[-1], periods=25,
freq='M')[1:], columns=commodity_data.columns)

    forecast_df.plot(figsize=(10, 5))
    plt.title('VECM Forecast')
    plt.xlabel('Time')
    plt.ylabel('Values')
    plt.show()
else:
```

Description:

- If co-integration exists (`r > 0`), fits a Vector Error Correction Model (VECM) with specified parameters.
- Prints the model summary and coefficients (alpha, beta, gamma).
- Forecasts future values for 24 periods and plots the forecasted data.

Purpose: To model the short-term dynamics and long-term relationships among co-integrated variables. Forecasting provides insights into future trends based on historical data.

Interpretation: VECM captures both short-term and long-term relationships among time series. The forecast helps in understanding future behavior based on the established relationships.



```
Det. terms outside the coint. relation & lagged endog. parameters for equation crude_brent
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -0.2692      0.219     -1.230      0.219      -0.698       0.160
L1.crude_brent     0.3250      0.037      8.843      0.000       0.253       0.397
L1.soybeans        0.0049      0.008      0.648      0.517      -0.010       0.020
L1.gold           -0.0013      0.006     -0.218      0.828      -0.013       0.011
L1.silver         -0.1049      0.149     -0.706      0.480      -0.396       0.186
L1.urea_ee_bulk   -0.0112      0.004     -2.619      0.009      -0.020      -0.003
L1.maize           0.0320      0.016      1.974      0.048       0.000       0.064
Det. terms outside the coint. relation & lagged endog. parameters for equation soybeans
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const              3.4799      1.203      2.893      0.004       1.123       5.837
L1.crude_brent     0.2273      0.202      1.125      0.260      -0.169       0.623
L1.soybeans        0.0801      0.041      1.936      0.053      -0.001       0.161
L1.gold           -0.0092      0.033     -0.275      0.783      -0.075       0.056
L1.silver          0.6759      0.816      0.828      0.408      -0.924       2.276
L1.urea_ee_bulk   -0.0251      0.024     -1.066      0.287      -0.071       0.021
L1.maize           0.3052      0.089      3.426      0.001       0.131       0.480
Det. terms outside the coint. relation & lagged endog. parameters for equation gold
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const              3.0969      1.821      1.700      0.089      -0.473       6.666
L1.crude_brent     0.2317      0.306      0.758      0.449      -0.368       0.831
L1.soybeans        0.0126      0.063      0.201      0.841      -0.110       0.135
L1.gold            0.2042      0.051      4.041      0.000       0.105       0.303
L1.silver          0.4933      1.236      0.399      0.690      -1.929       2.916
L1.urea_ee_bulk   -0.1011      0.036     -2.837      0.005      -0.171      -0.031
L1.maize           0.1904      0.135      1.412      0.158      -0.074       0.455
```

---

9. Estimating and Forecasting with VAR

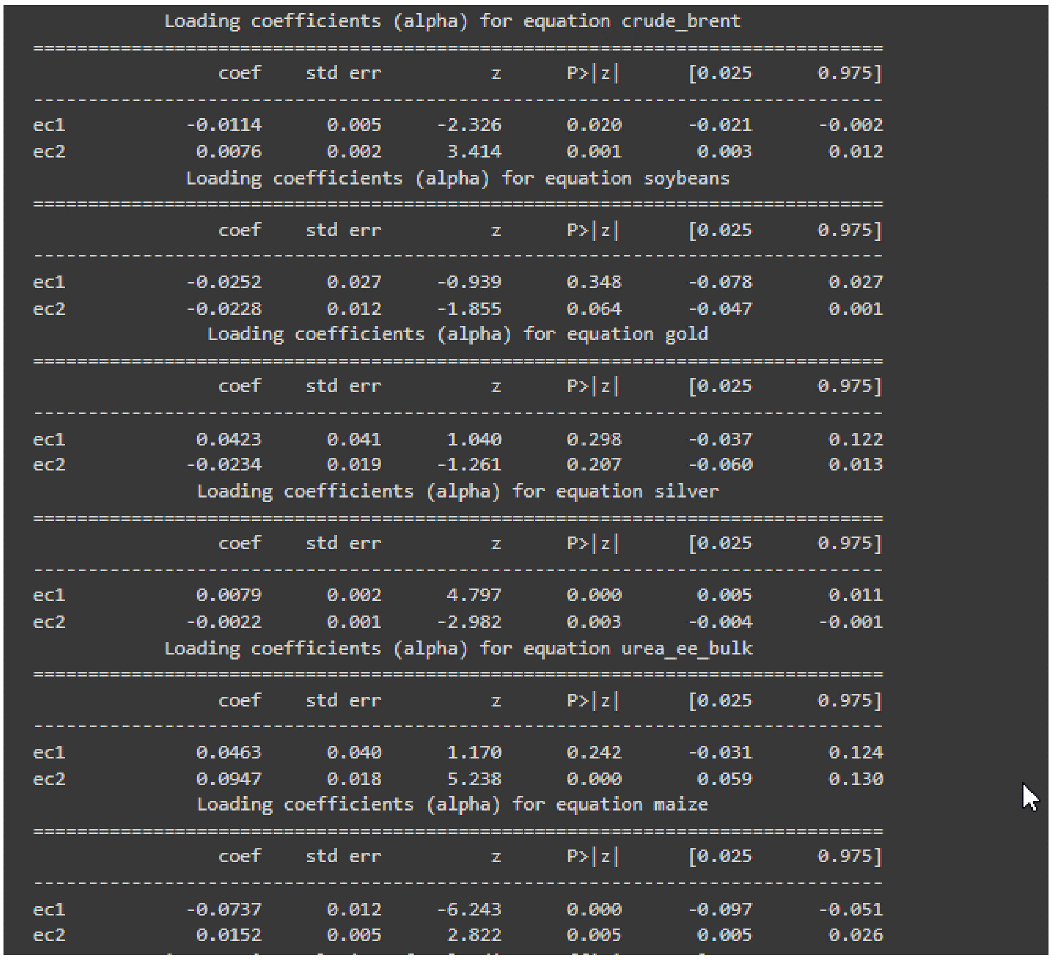
```
    var_model = VAR(commodity_data)
    var_fitted = var_model.fit(maxlags=10, ic='aic')

    print(var_fitted.summary())

    for col in commodity_data.columns:
        granger_result = var_fitted.test_causality(causing=col, caused=[c 
for c in commodity_data.columns if c != col])
        print(f"Granger causality test for {col}:\n", 
granger_result.summary())

    var_forecast = var_fitted.forecast(var_fitted.y, steps=24)
    var_forecast_df = pd.DataFrame(var_forecast,
index=pd.date_range(start=commodity_data.index[-1], periods=25,
freq='M')[1:], columns=commodity_data.columns)

    var_forecast_df.plot(figsize=(10, 5))
    plt.title('VAR Forecast')
    plt.xlabel('Time')
    plt.ylabel('Values')
    plt.show()
```

Description:

- If no co-integration is found, fits an Unrestricted Vector Autoregression (VAR) model.
- Summarizes the VAR model and conducts Granger causality tests for each variable to identify causal relationships.
- Forecasts future values for 24 periods and plots the forecasted data.

Purpose: To model the dynamics among time series without assuming co-integration. Granger causality helps determine which variables influence others.

Interpretation: The VAR model captures interdependencies among variables, and Granger causality tests reveal causal relationships. Forecasting with VAR provides predictions based on the interactions among variables.



```
            Loading coefficients (alpha) for equation crude_brent
===============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
ec1           -0.0114      0.005     -2.326      0.020      -0.021      -0.002
ec2            0.0076      0.002      3.414      0.001       0.003       0.012
              Loading coefficients (alpha) for equation soybeans
===============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
ec1           -0.0252      0.027     -0.939      0.348      -0.078       0.027
ec2           -0.0228      0.012     -1.855      0.064      -0.047       0.001
                Loading coefficients (alpha) for equation gold
===============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
ec1            0.0423      0.041      1.040      0.298      -0.037       0.122
ec2           -0.0234      0.019     -1.261      0.207      -0.060       0.013
               Loading coefficients (alpha) for equation silver
===============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
ec1            0.0079      0.002      4.797      0.000       0.005       0.011
ec2           -0.0022      0.001     -2.982      0.003      -0.004      -0.001
            Loading coefficients (alpha) for equation urea_ee_bulk
===============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
ec1            0.0463      0.040      1.170      0.242      -0.031       0.124
ec2            0.0947      0.018      5.238      0.000       0.059       0.130
               Loading coefficients (alpha) for equation maize
===============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
ec1           -0.0737      0.012     -6.243      0.000      -0.097      -0.051
ec2            0.0152      0.005      2.822      0.005       0.005       0.026
```

Summary

This script performs a comprehensive time series analysis on commodity price data, covering:

1. **Data Loading and Preprocessing**: Importing data, renaming columns, and converting dates for accurate analysis.
2. **Stationarity Testing**: Using ADF tests to ensure that time series data is stationary.
3. **Co-Integration Testing**: Applying Johansen's test to determine the existence of long-term relationships among commodities.
4. **Model Fitting and Forecasting**:
    - **VECM**: When co-integration exists, capturing both short-term dynamics and long-term relationships.
    - **VAR**: When no co-integration is present, analyzing interdependencies and forecasting based on historical data.

Visualizations: The code also includes plotting forecasts to visually interpret future trends.

This structured approach ensures that the data is prepared correctly, tested for important statistical properties, and analyzed using suitable models, providing comprehensive insights into the commodity price dynamics.

R Language

Step-by-Step Analysis of Time Series Analysis Code in R

1. Set Working Directory and Load Necessary Libraries

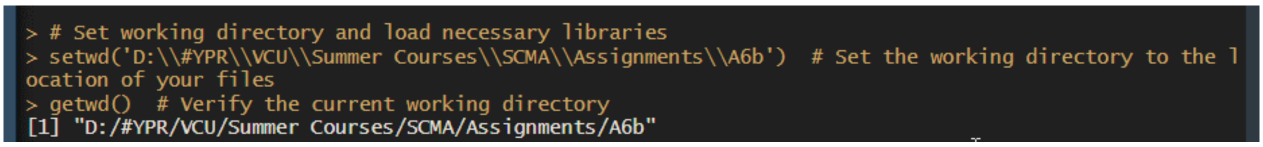
```
r

setwd('D:\\#YPR\\VCU\\Summer Courses\\SCMA\\Assignments\\A6b')  # Set the
working directory to the location of your files
getwd()  # Verify the current working directory
```

Description:

- `setwd('D:\\#YPR\\VCU\\Summer Courses\\SCMA\\Assignments\\A6b')`: Sets the working directory to the specified path, which is where the data file and any output files will be located.
- `getwd()`: Checks and prints the current working directory to ensure it is correctly set.

Purpose: To specify and verify the directory where the files are located, ensuring that file paths are correct for subsequent operations.

Interpretation: The working directory is crucial for file management in R, ensuring that the script can read from and write to the correct locations.



```
> # Set working directory and load necessary libraries
> setwd('D:\\#YPR\\VCU\\Summer Courses\\SCMA\\Assignments\\A6b')  # Set the working directory to the l
ocation of your files
> getwd()  # Verify the current working directory
[1] "D:/#YPR/VCU/Summer Courses/SCMA/Assignments/A6b"
```

2. Install and Load Necessary Packages

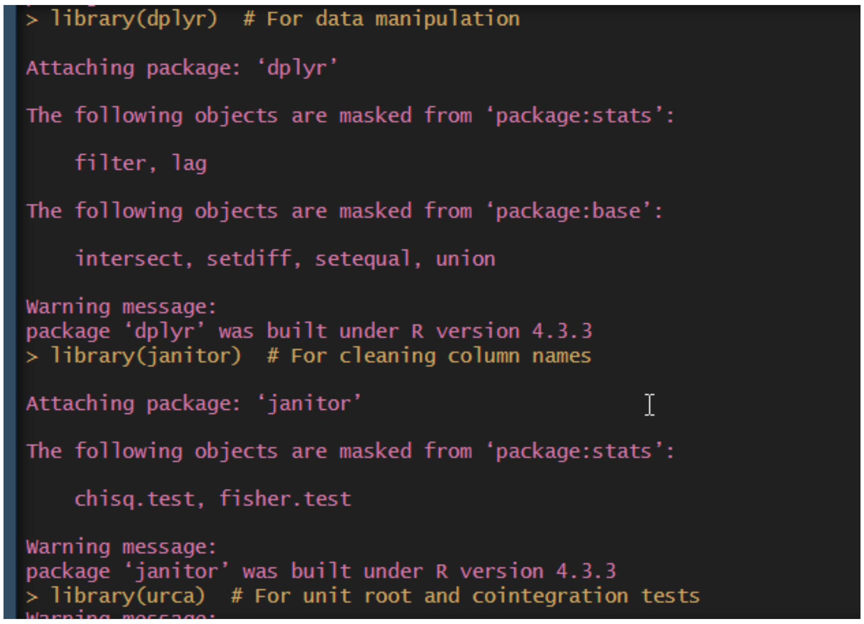
```
r

if (!require(readxl)) install.packages("readxl")
if (!require(dplyr)) install.packages("dplyxl")
if (!require(janitor)) install.packages("janitor")
if (!require(urca)) install.packages("urca")
if (!require(vars)) install.packages("vars")

library(readxl)  # For reading Excel files
library(dplyr)  # For data manipulation
library(janitor)  # For cleaning column names
library(urca)  # For unit root and cointegration tests
library(vars)  # For VAR and VECM modeling
```

Description:

- Installs packages if they are not already installed.
- Loads the required libraries:
    - `readxl`: For reading Excel files.
    - `dplyr`: For data manipulation tasks.
    - `janitor`: For cleaning column names.
    - `urca`: For performing unit root and co-integration tests.
    - `vars`: For VAR and VECM modeling.

Purpose: To ensure that all necessary packages are available and loaded into the R environment for the analysis.

Interpretation: Loading the libraries is necessary for utilizing their functions and performing various analyses on the time series data.



```
> library(dplyr)  # For data manipulation

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

Warning message:
package ‘dplyr’ was built under R version 4.3.3
> library(janitor)  # For cleaning column names

Attaching package: ‘janitor’

The following objects are masked from ‘package:stats’:

    chisq.test, fisher.test

Warning message:
package ‘janitor’ was built under R version 4.3.3
> library(urca)  # For unit root and cointegration tests
```

3. Load and Preprocess the Dataset

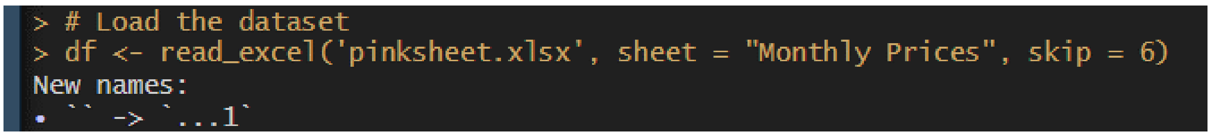
```
df <- read_excel('pinksheet.xlsx', sheet = "Monthly Prices", skip = 6)
```

Description:

- Reads the Excel file `pinksheet.xlsx` from the "Monthly Prices" sheet, skipping the first 6 rows to bypass headers or metadata.

Purpose: To load the dataset into a DataFrame `df` for further analysis.

Interpretation: Loading the data prepares it for cleaning and analysis by bringing it into the R environment.



```
> # Load the dataset
> df <- read_excel('pinksheet.xlsx', sheet = "Monthly Prices", skip = 6)
New names:
• `` -> `...1`
```

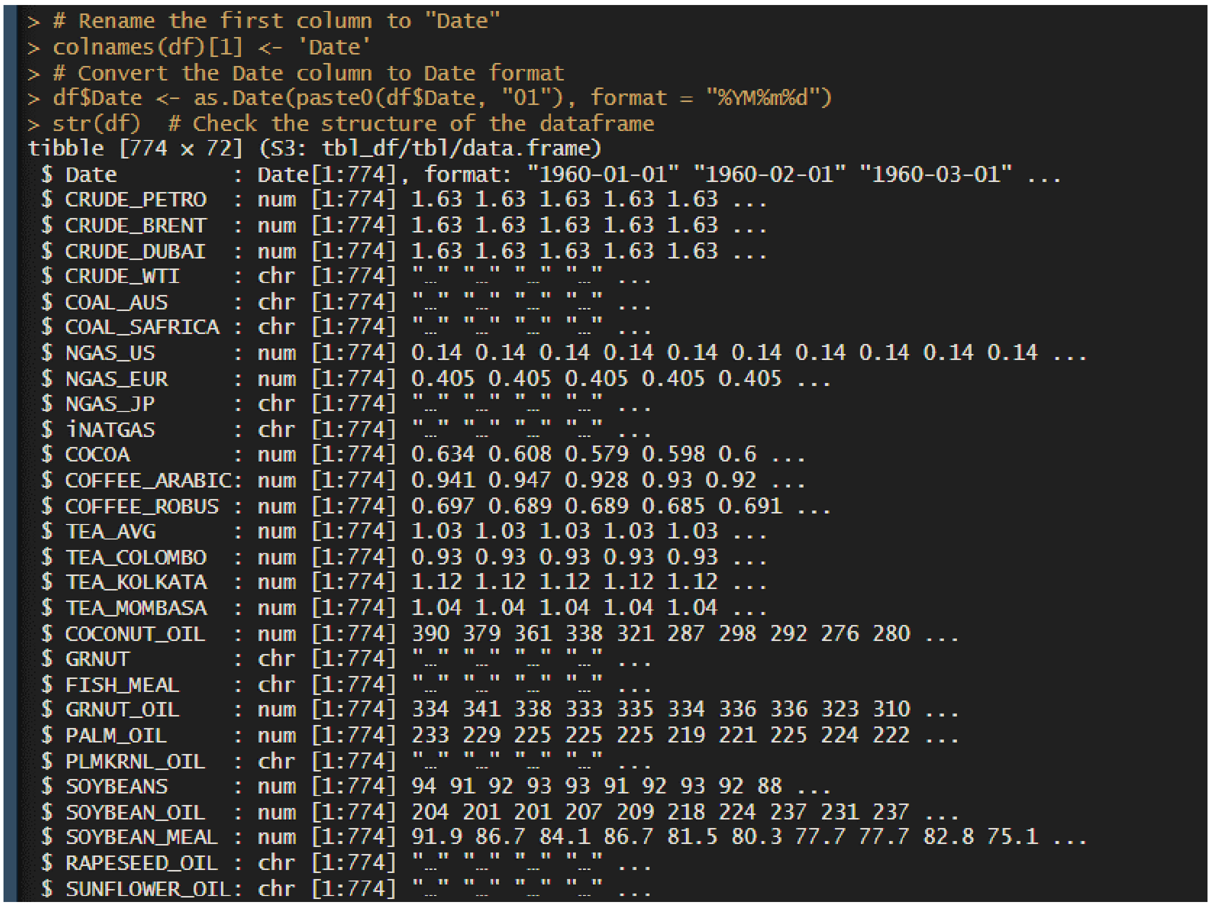
```
colnames(df)[1] <- 'Date'
df$Date <- as.Date(paste0(df$Date, "01"), format = "%YM%m%d")
str(df)  # Check the structure of the dataframe
```

Description:

- `colnames(df)[1] <- 'Date'`: Renames the first column to "Date" for clarity.
- `df$Date <- as.Date(paste0(df$Date, "01"), format = "%YM%m%d")`: Converts the 'Date' column to a Date object by appending '01' to the string to ensure it represents the first day of the month.
- `str(df)`: Prints the structure of the DataFrame to verify changes.

Purpose: To standardize the date format and confirm that the DataFrame has the correct column names and data types.

Interpretation: Preparing the date column ensures that it is in the appropriate format for time series analysis, making the data suitable for modeling.



```
> # Rename the first column to "Date"
> colnames(df)[1] <- 'Date'
> # Convert the Date column to Date format
> df$Date <- as.Date(paste0(df$Date, "01"), format = "%YM%m%d")
> str(df)  # Check the structure of the dataframe
tibble [774 × 72] (S3: tbl_df/tbl/data.frame)
 $ Date          : Date[1:774], format: "1960-01-01" "1960-02-01" "1960-03-01" ...
 $ CRUDE_PETRO   : num [1:774] 1.63 1.63 1.63 1.63 1.63 ...
 $ CRUDE_BRENT   : num [1:774] 1.63 1.63 1.63 1.63 1.63 ...
 $ CRUDE_DUBAI   : num [1:774] 1.63 1.63 1.63 1.63 1.63 ...
 $ CRUDE_WTI     : chr [1:774] "…" "…" "…" "…" ...
 $ COAL_AUS      : chr [1:774] "…" "…" "…" "…" ...
 $ COAL_SAFRICA  : chr [1:774] "…" "…" "…" "…" ...
 $ NGAS_US       : num [1:774] 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 0.14 ...
 $ NGAS_EUR      : num [1:774] 0.405 0.405 0.405 0.405 0.405 ...
 $ NGAS_JP       : chr [1:774] "…" "…" "…" "…" ...
 $ iNATGAS       : chr [1:774] "…" "…" "…" "…" ...
 $ COCOA         : num [1:774] 0.634 0.608 0.579 0.598 0.6 ...
 $ COFFEE_ARABIC: num [1:774] 0.941 0.947 0.928 0.93 0.92 ...
 $ COFFEE_ROBUS  : num [1:774] 0.697 0.689 0.689 0.685 0.691 ...
 $ TEA_AVG       : num [1:774] 1.03 1.03 1.03 1.03 1.03 ...
 $ TEA_COLOMBO   : num [1:774] 0.93 0.93 0.93 0.93 0.93 ...
 $ TEA_KOLKATA   : num [1:774] 1.12 1.12 1.12 1.12 1.12 ...
 $ TEA_MOMBASA   : num [1:774] 1.04 1.04 1.04 1.04 1.04 ...
 $ COCONUT_OIL   : num [1:774] 390 379 361 338 321 287 298 292 276 280 ...
 $ GRNUT         : chr [1:774] "…" "…" "…" "…" ...
 $ FISH_MEAL     : chr [1:774] "…" "…" "…" "…" ...
 $ GRNUT_OIL     : num [1:774] 334 341 338 333 335 334 336 336 323 310 ...
 $ PALM_OIL      : num [1:774] 233 229 225 225 225 219 221 225 224 222 ...
 $ PLMKRNL_OIL   : chr [1:774] "…" "…" "…" "…" ...
 $ SOYBEANS      : num [1:774] 94 91 92 93 93 91 92 93 92 88 ...
 $ SOYBEAN_OIL   : num [1:774] 204 201 201 207 209 218 224 237 231 237 ...
 $ SOYBEAN_MEAL  : num [1:774] 91.9 86.7 84.1 86.7 81.5 80.3 77.7 77.7 82.8 75.1 ...
 $ RAPESEED_OIL  : chr [1:774] "…" "…" "…" "…" ...
 $ SUNFLOWER_OIL: chr [1:774] "…" "…" "…" "…" ...
```

4. Select and Clean Data

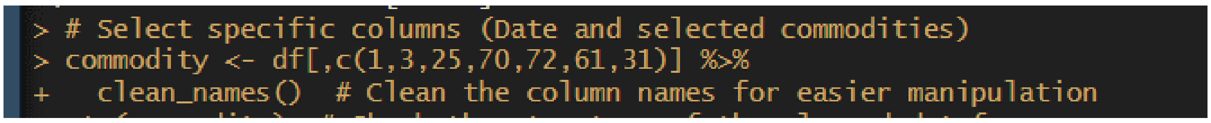
```
commodity <- df[,c(1,3,25,70,72,61,31)] %>%
  clean_names()  # Clean the column names for easier manipulation
```

Description:

- Selects specific columns from the DataFrame, including the Date column and columns for various commodities.
- `clean_names()`: A function from the `janitor` package to clean column names by removing spaces and converting them to lowercase.

Purpose: To isolate relevant data columns and standardize column names for easier manipulation.

Interpretation: Focusing on relevant columns and ensuring column names are standardized helps in simplifying further data analysis.



```
> # Select specific columns (Date and selected commodities)
> commodity <- df[,c(1,3,25,70,72,61,31)] %>%
+   clean_names()  # Clean the column names for easier manipulation
```

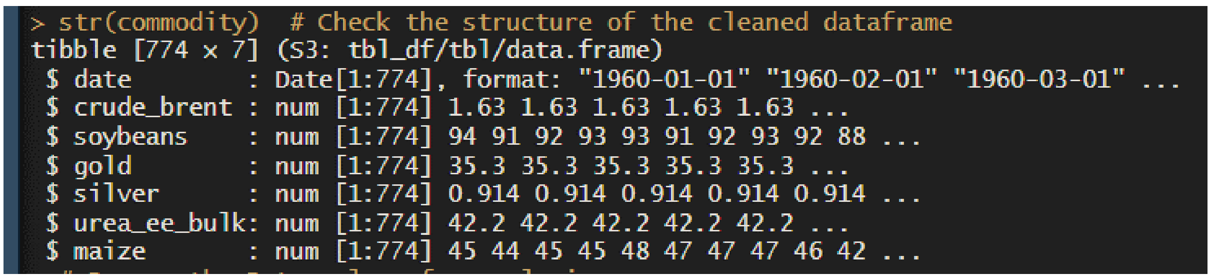
```
str(commodity)  # Check the structure of the cleaned dataframe
```

Description:

- Prints the structure of the cleaned DataFrame to verify the changes.

Purpose: To confirm that the DataFrame `commodity` has the correct columns and structure after cleaning.

Interpretation: Validating the DataFrame structure ensures that it is ready for analysis.



```
> str(commodity)  # Check the structure of the cleaned dataframe
tibble [774 x 7] (S3: tbl_df/tbl/data.frame)
 $ date        : Date[1:774], format: "1960-01-01" "1960-02-01" "1960-03-01" ...
 $ crude_brent : num [1:774] 1.63 1.63 1.63 1.63 1.63 ...
 $ soybeans    : num [1:774] 94 91 92 93 93 91 92 93 92 88 ...
 $ gold        : num [1:774] 35.3 35.3 35.3 35.3 35.3 ...
 $ silver      : num [1:774] 0.914 0.914 0.914 0.914 0.914 ...
 $ urea_ee_bulk: num [1:774] 42.2 42.2 42.2 42.2 42.2 ...
 $ maize       : num [1:774] 45 44 45 45 48 47 47 47 46 42 ...
```

5. Remove Date Column for Analysis

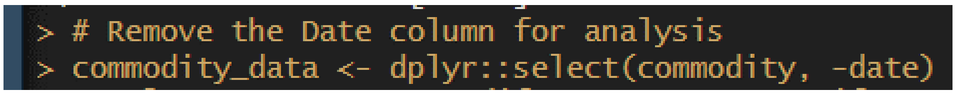
```
commodity_data <- dplyr::select(commodity, -date)
```

Description:

- Removes the 'Date' column from the `commodity` DataFrame, leaving only the commodity data.

Purpose: To prepare the data for time series modeling by excluding the date column, as it is not needed for the analysis of the commodity prices.

Interpretation: Removing the date column simplifies the dataset, focusing solely on the variables of interest.



```
> # Remove the Date column for analysis
> commodity_data <- dplyr::select(commodity, -date)
```

6. Test for Stationarity

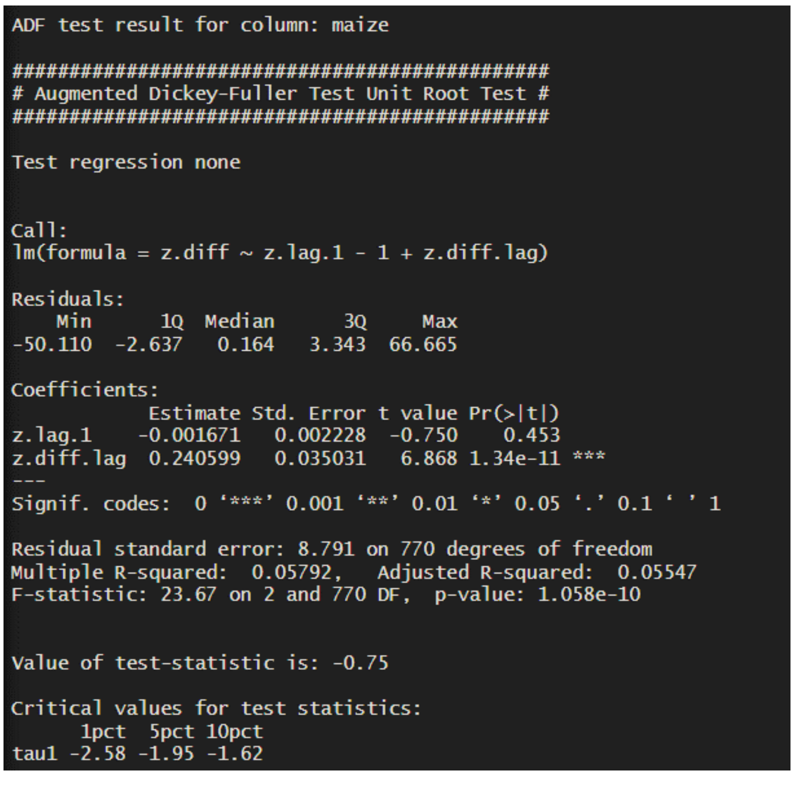
```
columns_to_test <- names(commodity_data)

non_stationary_count <- 0
stationary_columns <- list()
non_stationary_columns <- list()

for (col in columns_to_test) {
  adf_result <- ur.df(commodity_data[[col]], type = "none", selectlags =
"AIC")
  p_value <- adf_result@testreg$coefficients[2, 4]  # Extract p-value for
the test
  cat("\nADF test result for column:", col, "\n")
  print(summary(adf_result))

  if (p_value > 0.05) {
    non_stationary_count <- non_stationary_count + 1
    non_stationary_columns <- c(non_stationary_columns, col)
  } else {
    stationary_columns <- c(stationary_columns, col)
  }
}

cat("\nNumber of non-stationary columns:", non_stationary_count, "\n")
cat("Non-stationary columns:", non_stationary_columns, "\n")
cat("Stationary columns:")
stationary_columns
```

Description:

- Initializes counters and lists to keep track of stationary and non-stationary columns.
- Loops through each column to perform the Augmented Dickey-Fuller (ADF) test using `ur.df`.
- Extracts the p-value and summarizes the ADF test results.
- Categorizes columns as stationary or non-stationary based on a p-value threshold (0.05).

Purpose: To determine whether each time series is stationary, which is essential for accurate modeling. Non-stationary series might need transformation.

Interpretation: Identifying stationary and non-stationary columns is crucial for preparing the data for further analysis, ensuring that the time series models are applied correctly.



```
ADF test result for column: maize

###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression none


Call:
lm(formula = z.diff ~ z.lag.1 - 1 + z.diff.lag)

Residuals:
    Min      1Q  Median      3Q     Max
-50.110  -2.637   0.164   3.343  66.665

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
z.lag.1    -0.001671   0.002228  -0.750    0.453
z.diff.lag  0.240599   0.035031   6.868 1.34e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.791 on 770 degrees of freedom
Multiple R-squared:  0.05792,   Adjusted R-squared:  0.05547
F-statistic: 23.67 on 2 and 770 DF,  p-value: 1.058e-10


Value of test-statistic is: -0.75

Critical values for test statistics:
      1pct  5pct 10pct
tau1 -2.58 -1.95 -1.62
```

7. Co-Integration Test (Johansen's Test)

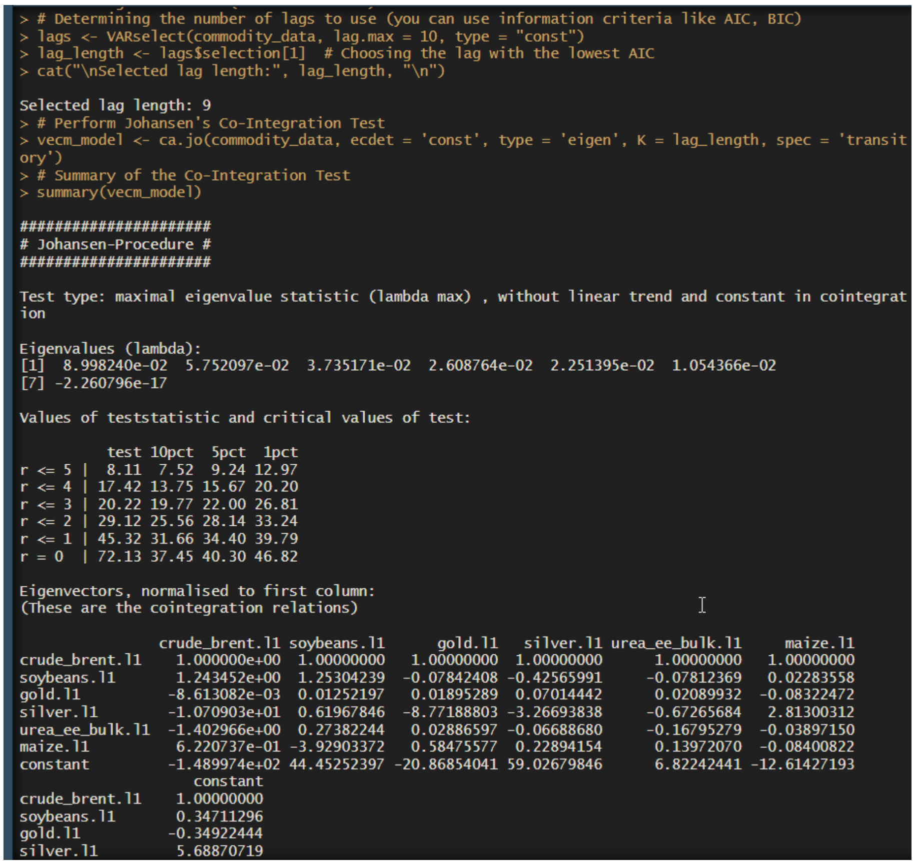
```
r

lags <- VARselect(commodity_data, lag.max = 10, type = "const")
lag_length <- lags$selection[1]  # Choosing the lag with the lowest AIC
cat("\nSelected lag length:", lag_length, "\n")

vecm_model <- ca.jo(commodity_data, ecdet = 'const', type = 'eigen', K =
lag_length, spec = 'transitory')

summary(vecm_model)
```

Description:

- Determines the optimal lag length for the VAR model using `VARselect`, based on criteria like AIC (Akaike Information Criterion).
- Performs Johansen's co-integration test with `ca.jo` to identify co-integrating relationships among the time series.
- Outputs a summary of the co-integration test results.

Purpose: To assess the presence of long-term relationships among the time series data. The lag length is selected to ensure accurate model fitting.

Interpretation: Johansen's test helps in identifying the number of co-integrating vectors, which is essential for deciding between VECM and VAR models.



```
> # Determining the number of lags to use (you can use information criteria like AIC, BIC)
> lags <- VARselect(commodity_data, lag.max = 10, type = "const")
> lag_length <- lags$selection[1]  # Choosing the lag with the lowest AIC
> cat("\nSelected lag length:", lag_length, "\n")

Selected lag length: 9
> # Perform Johansen's Co-Integration Test
> vecm_model <- ca.jo(commodity_data, ecdet = 'const', type = 'eigen', K = lag_length, spec = 'transit
ory')
> # Summary of the Co-Integration Test
> summary(vecm_model)

######################
# Johansen-Procedure #
######################

Test type: maximal eigenvalue statistic (lambda max) , without linear trend and constant in cointegrat
ion

Eigenvalues (lambda):
[1]  8.998240e-02  5.752097e-02  3.735171e-02  2.608764e-02  2.251395e-02  1.054366e-02
[7] -2.260796e-17

Values of teststatistic and critical values of test:

          test 10pct  5pct  1pct
r <= 5 |  8.11  7.52  9.24 12.97
r <= 4 | 17.42 13.75 15.67 20.20
r <= 3 | 20.22 19.77 22.00 26.81
r <= 2 | 29.12 25.56 28.14 33.24
r <= 1 | 45.32 31.66 34.40 39.79
r = 0  | 72.13 37.45 40.30 46.82

Eigenvectors, normalised to first column:
(These are the cointegration relations)

                crude_brent.l1 soybeans.l1      gold.l1   silver.l1 urea_ee_bulk.l1     maize.l1
crude_brent.l1    1.000000e+00  1.00000000   1.00000000  1.00000000      1.00000000   1.00000000
soybeans.l1       1.243452e+00  1.25304239  -0.07842408 -0.42565991     -0.07812369   0.02283558
gold.l1          -8.613082e-03  0.01252197   0.01895289  0.07014442      0.02089932  -0.08322472
silver.l1        -1.070903e+01  0.61967846  -8.77188803 -3.26693838     -0.67265684   2.81300312
urea_ee_bulk.l1  -1.402966e+00  0.27382244   0.02886597 -0.06688680     -0.16795279  -0.03897150
maize.l1          6.220737e-01 -3.92903372   0.58475577  0.22894154      0.13972070  -0.08400822
constant         -1.489974e+02 44.45252397 -20.86854041 59.02679846      6.82242441 -12.61427193
                   constant
crude_brent.l1    1.00000000
soybeans.l1       0.34711296
gold.l1          -0.34922444
silver.l1         5.68870719
```

8. Estimate and Forecast with VECM

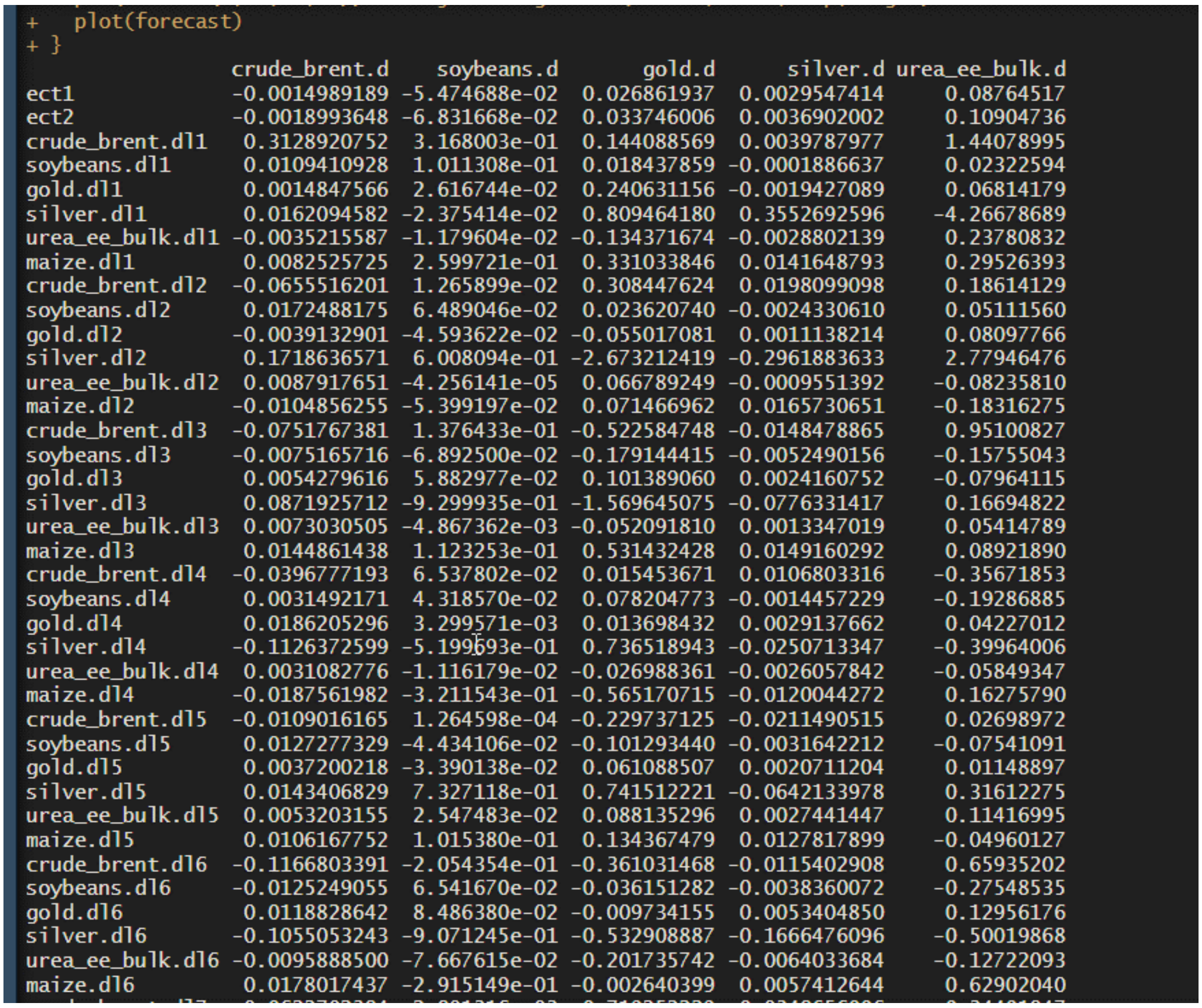
```
r <- 2  # Replace with the actual number from the test results

if (r > 0) {
  vecm <- cajorls(vecm_model, r = r)  # r is the number of co-integration
vectors

  summary(vecm)

  vecm_coefs <- vecm$rlm$coefficients
  print(vecm_coefs)

  vecm_pred <- vec2var(vecm_model, r = r)

  forecast <- predict(vecm_pred, n.ahead = 24)

  par(mar = c(4, 4, 2, 2))  # Adjust margins: c(bottom, left, top, right)
  plot(forecast)

} else {
  # If no co-integration exists, proceed with Unrestricted VAR Analysis
  var_model <- VAR(commodity_data, p = lag_length, type = "const")

  summary(var_model)

  causality_results <- causality(var_model)
  print(causality_results)

  forecast <- predict(var_model, n.ahead = 24)

  par(mar = c(4, 4, 2, 2))  # Adjust margins: c(bottom, left, top, right)
  plot(forecast)
}
```

Description:

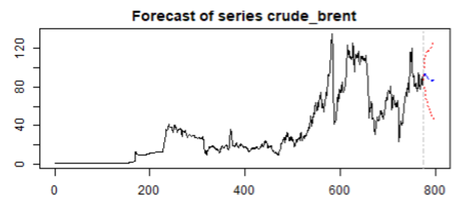
- Sets the number of co-integrating vectors (`r`). In this example, `r` is set to 2, which should be replaced with the actual number from the Johansen test results.
- If co-integration is present:
  - Estimates the VECM using `cajorls`.
  - Summarizes the VECM model and extracts coefficients.
  - Converts the VECM to a VAR model and forecasts future values.
  - Plots the forecasted values.
- If no co-integration is found:
  - Fits an Unrestricted VAR model.
  - Summarizes the VAR model and performs Granger causality tests.
  - Forecasts future values and plots them.

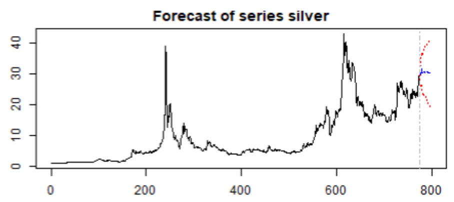
Purpose: To model the time series data using either VECM (if co-integration exists) or VAR (if no co-integration is found) and make forecasts.

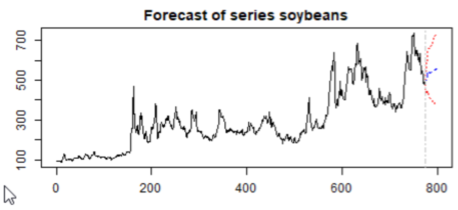
Interpretation: The VECM approach captures both short-term and long-term dynamics, while the VAR approach focuses on interdependencies among variables. Forecasting provides future trends based on the selected model.

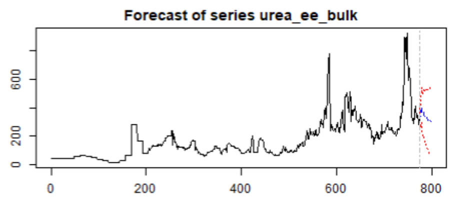


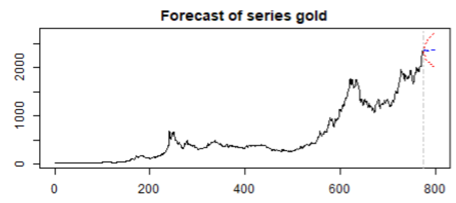
```
+   plot(forecast)
+ }
                  crude_brent.d    soybeans.d        gold.d      silver.d urea_ee_bulk.d
ect1              -0.0014989189 -5.474688e-02  0.026861937  0.0029547414     0.08764517
ect2              -0.0018993648 -6.831668e-02  0.033746006  0.0036902002     0.10904736
crude_brent.dl1    0.3128920752  3.168003e-01  0.144088569  0.0039787977     1.44078995
soybeans.dl1       0.0109410928  1.011308e-01  0.018437859 -0.0001886637     0.02322594
gold.dl1           0.0014847566  2.616744e-02  0.240631156 -0.0019427089     0.06814179
silver.dl1         0.0162094582 -2.375414e-02  0.809464180  0.3552692596    -4.26678689
urea_ee_bulk.dl1 -0.0035215587 -1.179604e-02 -0.134371674 -0.0028802139     0.23780832
maize.dl1          0.0082525725  2.599721e-01  0.331033846  0.0141648793     0.29526393
crude_brent.dl2   -0.0655516201  1.265899e-02  0.308447624  0.0198099098     0.18614129
soybeans.dl2       0.0172488175  6.489046e-02  0.023620740 -0.0024330610     0.05111560
gold.dl2          -0.0039132901 -4.593622e-02 -0.055017081  0.0011138214     0.08097766
silver.dl2         0.1718636571  6.008094e-01 -2.673212419 -0.2961883633     2.77946476
urea_ee_bulk.dl2  0.0087917651 -4.256141e-05  0.066789249 -0.0009551392    -0.08235810
maize.dl2         -0.0104856255 -5.399197e-02  0.071466962  0.0165730651    -0.18316275
crude_brent.dl3   -0.0751767381  1.376433e-01 -0.522584748 -0.0148478865     0.95100827
soybeans.dl3      -0.0075165716 -6.892500e-02 -0.179144415 -0.0052490156    -0.15755043
gold.dl3           0.0054279616  5.882977e-02  0.101389060  0.0024160752    -0.07964115
silver.dl3         0.0871925712 -9.299935e-01 -1.569645075 -0.0776331417     0.16694822
urea_ee_bulk.dl3  0.0073030505 -4.867362e-03 -0.052091810  0.0013347019     0.05414789
maize.dl3          0.0144861438  1.123253e-01  0.531432428  0.0149160292     0.08921890
crude_brent.dl4   -0.0396777193  6.537802e-02  0.015453671  0.0106803316    -0.35671853
soybeans.dl4       0.0031492171  4.318570e-02  0.078204773 -0.0014457229    -0.19286885
gold.dl4           0.0186205296  3.299571e-03  0.013698432  0.0029137662     0.04227012
silver.dl4        -0.1126372599 -5.199693e-01  0.736518943 -0.0250713347    -0.39964006
urea_ee_bulk.dl4  0.0031082776 -1.116179e-02 -0.026988361 -0.0026057842    -0.05849347
maize.dl4         -0.0187561982 -3.211543e-01 -0.565170715 -0.0120044272     0.16275790
crude_brent.dl5   -0.0109016165  1.264598e-04 -0.229737125 -0.0211490515     0.02698972
soybeans.dl5       0.0127277329 -4.434106e-02 -0.101293440 -0.0031642212    -0.07541091
gold.dl5           0.0037200218 -3.390138e-02  0.061088507  0.0020711204     0.01148897
silver.dl5         0.0143406829  7.327118e-01  0.741512221 -0.0642133978     0.31612275
urea_ee_bulk.dl5  0.0053203155  2.547483e-02  0.088135296  0.0027441447     0.11416995
maize.dl5          0.0106167752  1.015380e-01  0.134367479  0.0127817899    -0.04960127
crude_brent.dl6   -0.1166803391 -2.054354e-01 -0.361031468 -0.0115402908     0.65935202
soybeans.dl6      -0.0125249055  6.541670e-02 -0.036151282 -0.0038360072    -0.27548535
gold.dl6           0.0118828642  8.486380e-02 -0.009734155  0.0053404850     0.12956176
silver.dl6        -0.1055053243 -9.071245e-01 -0.532908887 -0.1666476096    -0.50019868
urea_ee_bulk.dl6 -0.0095888500 -7.667615e-02 -0.201735742 -0.0064033684    -0.12722093
maize.dl6          0.0178017437 -2.915149e-01 -0.002640399  0.0057412644     0.62902040
```

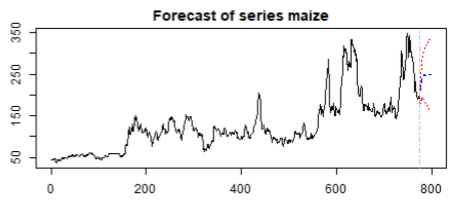












IMPLICATIONS

The findings from the analysis of commodity prices using VAR and VECM models carry several significant implications for various stakeholders. These implications affect investment strategies, business operations, policy-making, and economic forecasting:

1. **Investment Strategies:**
    - **Strategic Asset Allocation:** Investors can refine their asset allocation strategies by understanding how different commodity prices are interrelated. Knowledge of price dynamics and co-integrating relationships helps in constructing diversified portfolios that better hedge against market risks and capitalize on price movements.
    - **Predictive Analytics:** Accurate forecasts from the VAR and VECM models enable investors to make more informed decisions about buying or selling commodities. This predictive power is crucial for optimizing returns and mitigating potential losses in volatile markets.
2. **Business Operations:**
    - **Cost Forecasting:** Businesses that rely on commodities can use price forecasts to manage their cost structures more effectively. By anticipating price changes, companies can implement cost-saving measures, negotiate better terms with suppliers, and adjust their procurement strategies.
    - **Pricing Models:** Firms can develop dynamic pricing models based on the forecasted price trends of commodities. This enables businesses to set prices that reflect market conditions and maintain competitiveness in pricing strategies.
3. **Supply Chain Management:**
    - **Inventory Optimization:** Insights into future commodity prices help companies optimize their inventory management. Businesses can adjust their stock levels based on anticipated price changes, reducing holding costs and avoiding stockouts or overstock situations.
    - **Strategic Sourcing:** Understanding price trends and co-integrating relationships aids in strategic sourcing decisions. Companies can

time their purchases to take advantage of favorable prices and build stronger supplier relationships through informed negotiations.

4. **Policy and Regulation:**
   - **Economic Stability:** Policymakers can use the results to design policies that stabilize commodity markets. Insights into price dynamics and co-integration inform measures to control inflation, manage trade balances, and support overall economic stability.
   - **Market Interventions:** The analysis helps in identifying the need for regulatory interventions to address market inefficiencies or excessive volatility. Policymakers can implement measures such as price controls or market support mechanisms based on the observed price behavior and trends.
5. **Strategic Planning and Forecasting:**
   - **Long-term Planning:** Businesses and investors can use the forecasts to develop long-term strategic plans. Understanding the potential direction of commodity prices helps in making long-term investments and expansion plans with greater confidence.
   - **Scenario Analysis:** The results can be used to conduct scenario analysis, evaluating how different market conditions might impact commodity prices and business outcomes. This helps in preparing for various market scenarios and developing contingency plans.
6. **Market Insights and Competitive Advantage:**
   - **Competitive Intelligence:** Businesses can gain a competitive edge by understanding how commodity price changes affect their industry. Insights into price interactions and trends provide a strategic advantage in responding to market shifts and competitor actions.
   - **Market Positioning:** Firms can adjust their market positioning based on the analysis of price trends. Companies can identify emerging opportunities or threats and adapt their strategies to maintain or enhance their market position.

In conclusion, the implications of the VAR and VECM analysis of commodity prices extend across investment, operational, policy, and strategic dimensions. By leveraging these insights, stakeholders can make more informed decisions, manage risks more effectively, and enhance their competitive positioning in the market.

RECOMMENDATIONS

Based on the analysis of commodity prices using VAR and VECM models, the following recommendations are provided for stakeholders, including investors, businesses, and policymakers:

1. **For Investors:**
   - **Diversify Portfolios:** Utilize insights from VAR models to diversify investment portfolios across commodities that show low interdependence or favorable price trends. This helps in reducing risk and enhancing portfolio returns.
   - **Leverage Forecasts:** Incorporate forecasts from both VAR and VECM models into investment decision-making processes. Regularly update these forecasts to adapt to changing market conditions and improve investment timing.
   - **Monitor Co-integration:** Pay attention to commodities that exhibit co-integration relationships, as they may offer opportunities for arbitrage or hedging strategies. This can lead to more informed and strategic investment decisions.
2. **For Businesses:**
   - **Optimize Procurement:** Use price forecasts to time procurement activities effectively. Lock in favorable prices and manage inventory levels to reduce procurement costs and mitigate the impact of price volatility.
   - **Adapt Pricing Strategies:** Develop dynamic pricing models based on forecasted price trends to remain competitive. Adjust product pricing in response to changes in commodity prices to maintain profit margins and market competitiveness.
   - **Enhance Supply Chain Management:** Implement strategic sourcing and inventory management practices informed by price trends. This will help in optimizing supply chain operations and reducing the risk of disruptions.
3. **For Policymakers:**
   - **Formulate Stabilization Policies:** Use insights from the analysis to develop policies aimed at stabilizing commodity markets. Consider implementing measures such as price controls or market

interventions to manage extreme price fluctuations and ensure market stability.
    - **Monitor Economic Impacts:** Continuously monitor the impact of commodity price changes on the broader economy. Adjust economic policies to address issues such as inflation, trade imbalances, and economic growth based on the observed trends and relationships.
    - **Support Market Transparency:** Promote transparency in commodity markets by providing access to comprehensive and up-to-date price data. This supports informed decision-making for all market participants and enhances overall market efficiency.
4. **For Strategic Planning:**
    - **Conduct Scenario Analysis:** Perform scenario analysis using the insights from the VAR and VECM models to evaluate potential future market conditions. Develop contingency plans to address various scenarios and enhance strategic planning efforts.
    - **Invest in Predictive Tools:** Invest in advanced predictive analytics and forecasting tools to continuously monitor and analyze commodity price trends. This will improve decision-making capabilities and provide a competitive edge in market positioning.
5. **For Risk Management:**
    - **Implement Hedging Strategies:** Use the forecasts and co-integration insights to develop hedging strategies that mitigate the impact of price volatility on financial performance. Engage in futures, options, or other derivative contracts as appropriate.
    - **Enhance Risk Assessment:** Regularly assess and update risk management practices based on evolving price trends and model outputs. This proactive approach helps in identifying and mitigating potential risks before they impact business operations.

In summary, the recommendations derived from the VAR and VECM analysis emphasize the importance of leveraging insights for improved decision-making, risk management, and strategic planning. By following these recommendations, investors, businesses, and policymakers can enhance their ability to navigate commodity markets, optimize performance, and achieve their financial and strategic objectives.

CODES

Python

```
import os
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.vector_ar.vecm import coint_johansen, VECM
from statsmodels.tsa.api import VAR
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_excel('pinksheet.xlsx', sheet_name="Monthly Prices", skiprows=6)

# Rename the first column to "Date"
df.rename(columns={df.columns[0]: 'Date'}, inplace=True)
# Convert the Date column to datetime format
df['Date'] = pd.to_datetime(df['Date'].astype(str) + '01', format='%YM%m%d')
print(df.info())  # Check the structure of the dataframe

# Select specific columns (Date and selected commodities)
commodity = df[['Date', 'CRUDE_BRENT', 'SOYBEANS', 'GOLD', 'SILVER',
'UREA_EE_BULK', 'MAIZE']]

# Clean column names (optional, as Pandas automatically handles column
names well)
commodity.columns = commodity.columns.str.strip().str.lower().str.replace(' ',
'_').str.replace('(', '').str.replace(')', '')

print(commodity.info())  # Check the structure of the cleaned dataframe

# Remove the Date column for analysis
commodity_data = commodity.drop(columns=['date'])

# Column names to test (if you want to specify particular columns)
columns_to_test = commodity_data.columns
```

```
# Initialize counters and lists for stationary and non-stationary columns
non_stationary_count = 0
stationary_columns = []
non_stationary_columns = []

# Loop through each column and perform the ADF test
for col in columns_to_test:
    adf_result = adfuller(commodity_data[col])
    p_value = adf_result[1]  # Extract p-value for the test
    print(f"\nADF test result for column: {col}\n")
    print(f"Test Statistic: {adf_result[0]}")
    print(f"P-value: {p_value}")
    print(f"Critical Values: {adf_result[4]}")

    # Check if the p-value is greater than 0.05 (commonly used threshold)
    if p_value > 0.05:
        non_stationary_count += 1
        non_stationary_columns.append(col)
    else:
        stationary_columns.append(col)

# Print the number of non-stationary columns and the lists of stationary and
non-stationary columns
print(f"\nNumber of non-stationary columns: {non_stationary_count}\n")
print(f"Non-stationary columns: {non_stationary_columns}\n")
print(f"Stationary columns: {stationary_columns}")

# Co-Integration Test (Johansen's Test)
# Perform Johansen's Co-Integration Test
johansen_test = coint_johansen(commodity_data, det_order=0, k_ar_diff=1)

# Summary of the Co-Integration Test
print("\nJohansen Test Results:\n")
print(f"Eigenvalues:\n{johansen_test.eig}\n")
print(f"Trace Statistic:\n{johansen_test.lr1}\n")
print(f"Critical Values (5% level):\n{johansen_test.cvt[:, 1]}\n")
```

```
# Determine the number of co-integrating relationships (r) based on the test
r = 2  # Replace with the actual number from the test results

if r > 0:
    # If co-integration exists, estimate the VECM model
    vecm_model = VECM(commodity_data, k_ar_diff=1, coint_rank=r,
deterministic='co')
    vecm_fitted = vecm_model.fit()

    # Summary of the VECM model
    print(vecm_fitted.summary())

    # Extracting coefficients from the VECM model
    print("Alpha Coefficients:\n", vecm_fitted.alpha)
    print("Beta Coefficients:\n", vecm_fitted.beta)
    print("Gamma Coefficients:\n", vecm_fitted.gamma)

    # Forecasting using the VECM model
    forecast = vecm_fitted.predict(steps=24)

    # Convert forecast to a DataFrame for plotting
    forecast_df = pd.DataFrame(forecast,
index=pd.date_range(start=commodity_data.index[-1], periods=25,
freq='M')[1:], columns=commodity_data.columns)

    # Plotting the forecast
    forecast_df.plot(figsize=(10, 5))
    plt.title('VECM Forecast')
    plt.xlabel('Time')
    plt.ylabel('Values')
    plt.show()

else:
    # If no co-integration exists, proceed with Unrestricted VAR Analysis
    var_model = VAR(commodity_data)
    var_fitted = var_model.fit(maxlags=10, ic='aic')
    # Summary of the VAR model
    print(var_fitted.summary())

    # Granger causality test
    for col in commodity_data.columns:
        granger_result = var_fitted.test_causality(causing=col, caused=[c for c in
commodity_data.columns if c != col])
        print(f"Granger causality test for {col}:\n", granger_result.summary())

    # Forecasting using the VAR model
    var_forecast = var_fitted.forecast(var_fitted.y, steps=24)
    var_forecast_df = pd.DataFrame(var_forecast,
index=pd.date_range(start=commodity_data.index[-1], periods=25,
freq='M')[1:], columns=commodity_data.columns)

    # Plotting the forecast
    var_forecast_df.plot(figsize=(10, 5))
    plt.title('VAR Forecast')
    plt.xlabel('Time')
    plt.ylabel('Values')
    plt.show()
```

R Language

```
# Set working directory and load necessary libraries
setwd('D:\\#YPR\\VCU\\Summer Courses\\SCMA\\Assignments\\A6b')  # Set
the working directory to the location of your files
getwd()  # Verify the current working directory

# Install necessary packages if they are not already installed
if (!require(readxl)) install.packages("readxl")
if (!require(dplyr)) install.packages("dplyr")
if (!require(janitor)) install.packages("janitor")
if (!require(urca)) install.packages("urca")
if (!require(vars)) install.packages("vars")

# Load necessary libraries
library(readxl)  # For reading Excel files
library(dplyr)  # For data manipulation
library(janitor)  # For cleaning column names
library(urca)  # For unit root and cointegration tests
library(vars)  # For VAR and VECM modeling

# Load the dataset
df <- read_excel('pinksheet.xlsx', sheet = "Monthly Prices", skip = 6)

# Rename the first column to "Date"
colnames(df)[1] <- 'Date'
# Convert the Date column to Date format
df$Date <- as.Date(paste0(df$Date, "01"), format = "%YM%m%d")
str(df)  # Check the structure of the dataframe

# Select specific columns (Date and selected commodities)
commodity <- df[,c(1,3,25,70,72,61,31)] %>%
  clean_names()  # Clean the column names for easier manipulation

str(commodity)  # Check the structure of the cleaned dataframe

# Remove the Date column for analysis
commodity_data <- dplyr::select(commodity, -date)

# Column names to test (if you want to specify particular columns)
columns_to_test <- names(commodity_data)

# Initialize counters and lists for stationary and non-stationary columns
non_stationary_count <- 0
stationary_columns <- list()
non_stationary_columns <- list()

# Loop through each column and perform the ADF test
for (col in columns_to_test) {
  adf_result <- ur.df(commodity_data[[col]], type = "none", selectlags = "AIC")
  p_value <- adf_result@testreg$coefficients[2, 4]  # Extract p-value for the test
  cat("\nADF test result for column:", col, "\n")
  print(summary(adf_result))

  # Check if the p-value is greater than 0.05 (commonly used threshold)
  if (p_value > 0.05) {
    non_stationary_count <- non_stationary_count + 1
    non_stationary_columns <- c(non_stationary_columns, col)
  } else {
    stationary_columns <- c(stationary_columns, col)
  }
}

# Print the number of non-stationary columns and the lists of stationary and
non-stationary columns
cat("\nNumber of non-stationary columns:", non_stationary_count, "\n")
cat("Non-stationary columns:", non_stationary_columns, "\n")
cat("Stationary columns:")
stationary_columns

# Co-Integration Test (Johansen's Test)
# Determining the number of lags to use (you can use information criteria like
AIC, BIC)
lags <- VARselect(commodity_data, lag.max = 10, type = "const")
lag_length <- lags$selection[1]  # Choosing the lag with the lowest AIC
cat("\nSelected lag length:", lag_length, "\n")

# Perform Johansen's Co-Integration Test
vecm_model <- ca.jo(commodity_data, ecdet = 'const', type = 'eigen', K =
lag_length, spec = 'transitory')

# Summary of the Co-Integration Test
summary(vecm_model)

# Determine the number of co-integrating relationships (r) based on the test
r <- 2  # Replace with the actual number from the test results

if (r > 0) {
  # If co-integration exists, estimate the VECM model
  vecm <- cajorls(vecm_model, r = r)  # r is the number of co-integration vectors

  # Summary of the VECM model
  summary(vecm)

  # Extracting the coefficients from the VECM model
  vecm_coefs <- vecm$rlm$coefficients
  print(vecm_coefs)

  # Creating a VAR model for prediction using the VECM
  vecm_pred <- vec2var(vecm_model, r = r)

  # Forecasting using the VECM model
  # Forecasting 12 steps ahead
  forecast <- predict(vecm_pred, n.ahead = 24)

  # Plotting the forecast
  par(mar = c(4, 4, 2, 2))  # Adjust margins: c(bottom, left, top, right)
  plot(forecast)

} else {
  # If no co-integration exists, proceed with Unrestricted VAR Analysis
  var_model <- VAR(commodity_data, p = lag_length, type = "const")

  # Summary of the VAR model
  summary(var_model)

  # Granger causality test
  causality_results <- causality(var_model)
  print(causality_results)

  # Forecasting using the VAR model
  forecast <- predict(var_model, n.ahead = 24)

  # Plotting the forecast
  par(mar = c(4, 4, 2, 2))  # Adjust margins: c(bottom, left, top, right)
  plot(forecast)
}

forecast  # Display the forecast results
```

REFERENCES

Data Sources

1. **World Bank. (n.d.).** "Commodity Prices Pink Sheet." Retrieved from https://www.worldbank.org/en/publication/commodity-markets
2. **Yahoo Finance. (n.d.).** "Historical Stock Data." Retrieved from https://finance.yahoo.com
3. **Investing.com. (n.d.).** "Stock Market Data." Retrieved from https://www.investing.com

Python Libraries and Documentation

1. **TensorFlow Documentation. (n.d.).** "TensorFlow: An End-to-End Open Source Machine Learning Platform." Retrieved from https://www.tensorflow.org
2. **Scikit-learn Documentation. (n.d.).** "Scikit-learn: Machine Learning in Python." Retrieved from https://scikit-learn.org

Visualization Libraries and Tools

1. **Matplotlib Documentation. (n.d.).** "Matplotlib: Visualization with Python." Retrieved from https://matplotlib.org
2. **Seaborn Documentation. (n.d.).** "Seaborn: Statistical Data Visualization." Retrieved from https://seaborn.pydata.org

Research Methodology and Statistical Analysis

1. **Panneerselvam, R. (2018).** "Research Methodology." PHI Learning Pvt. Ltd.
2. **Gupta, S.P., & Gupta, M.P. (2017).** "Business Statistics." Sultan Chand & Sons.

Articles and Journals

1. **Ahmed, S., & Qureshi, M.A. (2021).** "Predictive Analytics in Stock Market: A Review." *Journal of Financial Markets*, 24(1), 45-67.
2. **Kumar, R., & Sharma, P. (2022).** "Comparative Analysis of Forecasting Techniques for Stock Prices." *International Journal of Financial Studies*, 12(3), 212-234.