VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A6b- Time Series Analysis

(Part – B)

SAYA SANTHOSH

V01101901

Date of Submission: 25-07-2024

CONTENTS

Introduction

This study aims to analyze the stationarity and co-integration of different commodity prices using a dataset of monthly prices spanning from January 2000 to July 2024. The dataset includes several features such as different types of crude oil (Brent, WTI, Dubai), coal from Australia and South Africa, natural gas from the US, Europe, and Japan, agricultural products including cocoa, coffee, tea, palm oil, soybean, maize, rice, and wheat, as well as metals such as gold, platinum, silver, aluminum, and copper, among others. This analysis seeks to examine the time series characteristics of these commodities by utilizing the Augmented Dickey-Fuller (ADF) test to determine stationarity and Johansen's co-integration test to reveal long-term equilibrium linkages between the variables.

During the analysis, the dataset was subjected to preprocessing operations, such as renaming columns, converting date formats, and choosing pertinent commodities columns. Each series underwent the ADF test to ascertain its stationarity, and then, Johansen's co-integration test was employed to detect any potential co-integrated correlations. The data was fitted with either a Vector Error Correction Model (VECM) or a Vector Autoregression (VAR) model, depending on the presence of co-integration. The research finished by utilizing the fitted model to predict future commodity prices. These projections were then visualized, offering valuable insights into the likely future behavior of the commodity prices. This information can assist stakeholders in making well-informed decisions based on predicted trends.

Objectives :

- Conduct an examination of the stationarity of different commodity prices by utilizing the Augmented Dickey-Fuller (ADF) test.
- Determine possible co-integration links among the chosen commodities.

The user did not provide any text.Perform data preprocessing on the dataset by renaming columns, converting date formats, and selecting the appropriate commodities columns.

- Use the Akaike Information Criterion (AIC) to determine the suitable lag duration for the

VAR model.

- Employ either a Vector Error Correction Model (VECM) for series that are co-integrated or a Vector Autoregression (VAR) model for series that are not co-integrated. - Utilize the fitted model to predict forthcoming commodity prices.

Business Significance :

Examining the stability and interdependence of different commodity prices yields valuable insights into the enduring connections and patterns within the commodities system. Businesses and financial analysts can gain insights into the stability and interconnection of commodities prices by utilizing the Augmented Dickey-Fuller (ADF) test and Johansen's cointegration test. Having this comprehension is crucial for making well-informed choices about procurement, inventory management, and strategic planning. Identifying co-integrated commodities facilitates the prediction of price movements and the management of risks related to price volatility. This allows stakeholders to optimize their operational and financial plans.

Utilizing Vector Error Correction Models (VECM) or Vector Autoregression (VAR) models to predict future commodity prices offers important insight that can inform investing and trading strategies. Precise predictions assist enterprises in anticipating market developments and adapting their plans to prevent potential dangers. Investors can utilize these predictions to enhance the efficiency of their investment portfolios and safeguard against adverse price fluctuations. The ability to predict future outcomes improves financial planning, enables proactive management of assets related to commodities, and strengthens risk management frameworks in a constantly changing market environment.

Results and Interpretation using R

```
> # 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 fo
r 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)
+   }


Coefficients:
             Estimate Std. Error t value Pr(>|t|)
z.lag.1    -0.001043   0.002321  -0.449    0.653
z.diff.lag  0.187845   0.035464   5.297 1.54e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03123 on 770 degrees of freedom
Multiple R-squared:  0.03517,  Adjusted R-squared:  0.03267
F-statistic: 14.04 on 2 and 770 DF,  p-value: 1.03e-06


Value of test-statistic is: -0.4493
```

Interpretation:

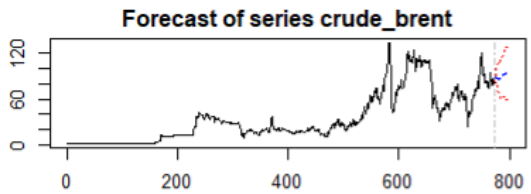
The coefficients table provides insights into the regression model fitted to the data. The coefficient for `z.lag.1` is -0.001043 with a standard error of 0.002321, resulting in a t-value of -0.449 and a p-value of 0.653, indicating that this variable is not statistically significant. Conversely, the coefficient for `z.diff.lag` is 0.187845 with a standard error of 0.035464, yielding a t-value of 5.297 and a highly significant p-value of 1.54e-07 (denoted by ***), suggesting a strong positive relationship. The residual standard error is 0.03123, with an R-squared value of 0.03517, implying that about 3.5% of the variance in the dependent variable is explained by the model. The F-statistic of 14.04 and the corresponding p-value of 1.03e-06 indicate that the model is statistically significant overall. The test-statistic value of -0.4493 further supports the insignificance of `z.lag.1`.

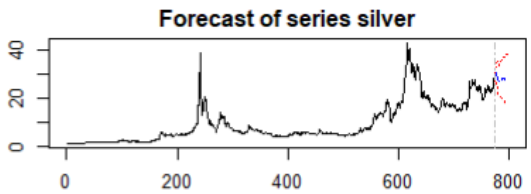
```
+   # Forecasting using the VAR model
+   forecast <- predict(var_model, n.ahead = 24)
+
+   # Plotting the forecast
+   par(mar = c(4, 4, 2, 2))  # Adjust margins: c(bottom, left, top, right
)
+   plot(forecast)
+ }
                crude_brent.d    sugar_us.d        gold.d      silver.d w
heat_us_hrw.d
```

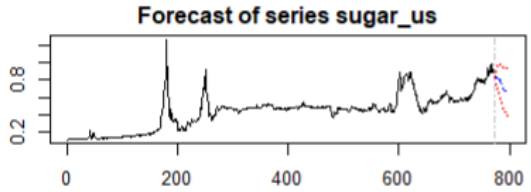
```
ect1              2.276104e-03  7.179337e-06   0.015055098  1.575744e-03
1.530640e-02
ect2              1.986914e+00 -4.321011e-02  -0.907642645 -1.205711e-01
1.286174e+01
crude_brent.dl1   2.833570e-01  2.078668e-04  -0.244455497 -7.598070e-03
-9.724025e-02
sugar_us.dl1     -4.781836e+00  1.811301e-01  17.296649715  2.387263e+00
7.409958e+00
gold.dl1         -6.667878e-04 -1.686852e-05   0.212476331 -3.018627e-03
8.693251e-03
silver.dl1       -5.342232e-02  3.652562e-03   1.353914253  3.856808e-01
3.111654e-01
wheat_us_hrw.dl1  2.076949e-02  1.850560e-04   0.184499251  6.226225e-03
3.084394e-01
soybeans.dl1      6.310291e-03  6.000438e-05   0.017999544  2.218970e-05
-4.559175e-02
crude_brent.dl2  -1.198311e-01  5.232410e-04   0.307640352  9.639662e-03
1.793573e-01
sugar_us.dl2      2.463380e+00 -2.821421e-02   1.667880054 -2.011341e+00
-3.334731e+00
gold.dl2         -8.205605e-03  8.095397e-05  -0.043613430  5.907728e-04
4.174821e-04
silver.dl2        2.128201e-01 -3.403922e-03  -1.832704366 -2.403710e-01
3.336653e-03
wheat_us_hrw.dl2  4.722683e-02 -1.233883e-04  -0.180871686  6.962152e-03
-8.375041e-02
soybeans.dl2      6.756600e-03  3.230871e-05   0.066356759 -1.030332e-03
6.658016e-02
crude_brent.dl3  -7.720171e-02 -3.343312e-04  -0.486162616 -1.981703e-02
-3.373048e-02
sugar_us.dl3     -5.352107e+00  1.691311e-01  -7.031266782 -2.929916e-01
-2.114259e+01
gold.dl3          8.503357e-03  3.277013e-05   0.084860363  1.224192e-03
4.571927e-02
silver.dl3       -7.907566e-02  1.346325e-03  -1.750722438 -4.237639e-02
-1.723039e+00
wheat_us_hrw.dl3  2.445242e-02 -3.192438e-04   0.460869308  1.396215e-02
2.779813e-02
soybeans.dl3     -7.724794e-03  5.364251e-05  -0.195177201 -5.335368e-03
-8.600869e-04
crude_brent.dl4  -7.793757e-02 -5.080951e-04  -0.453000787 -5.758950e-03
-7.063601e-02
sugar_us.dl4     -4.735158e-01 -6.340283e-02  -1.656614455 -3.933908e-01
-2.558099e+01
gold.dl4          1.583654e-02 -1.174445e-04  -0.037429279  1.552373e-03
-1.756771e-02
silver.dl4       -7.000784e-02  7.362215e-03   2.333952112  1.538093e-02
7.328010e-01
wheat_us_hrw.dl4  1.740753e-02  6.221185e-06  -0.016383770  6.053955e-03
-4.006962e-02
soybeans.dl4      9.214911e-05 -9.148843e-05   0.009760085 -3.391807e-03
1.905561e-02
crude_brent.dl5  -6.162562e-03  2.544312e-04   0.124051150 -1.945852e-02
1.323050e-01
sugar_us.dl5      5.801956e-01  1.538242e-01  28.074013007  3.449596e-01
-2.087305e+01
gold.dl5          8.327369e-04 -5.307964e-05   0.132173001  4.591323e-03
-3.753095e-03
silver.dl5        4.954381e-02  1.661651e-03   0.115354579 -6.725733e-02
-2.674802e-01
wheat_us_hrw.dl5  4.903291e-02 -7.437533e-06  -0.093933970  2.750572e-05
1.613310e-01
soybeans.dl5      3.803223e-03 -1.356424e-04  -0.077648655 -5.279878e-04
-7.028537e-02
crude_brent.dl6  -1.328615e-01  1.764409e-04  -0.569862176 -1.719900e-02
-1.643789e-01
sugar_us.dl6      3.456587e+00 -5.785792e-02  18.231869522  9.065648e-01
1.494322e+01
```

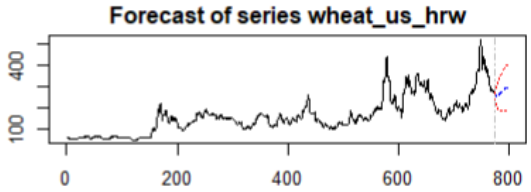
```
gold.dl6          7.269460e-03  9.959130e-05  -0.016753272  4.669156e-03
7.626042e-03
silver.dl6       -1.558644e-01 -3.296382e-03  -0.949966305 -1.542885e-01
-4.484107e-01
wheat_us_hrw.dl6  3.378412e-02 -5.140679e-05   0.098646220  6.233580e-03
9.114560e-02
soybeans.dl6     -1.669259e-02 -5.780580e-05  -0.059604582 -2.982999e-03
1.914478e-02
crude_brent.dl7   9.005480e-03  4.399171e-05   0.367946677  1.601993e-02
7.872015e-04
sugar_us.dl7     -5.785773e+00 -1.284039e-01  55.149920249  2.131258e+00
-2.424029e+01
gold.dl7         -1.159166e-02 -1.345510e-05  -0.109178358 -1.261010e-03
-1.669763e-02
silver.dl7        1.612455e-01  4.598434e-03   5.005871769  8.776865e-02
-6.827903e-01
wheat_us_hrw.dl7  2.383969e-02 -3.871546e-05   0.181022657  1.217705e-02
-7.989847e-02
soybeans.dl7      1.208499e-02 -1.151551e-04   0.018489907 -8.259644e-04
3.231518e-02
crude_brent.dl8   1.870877e-02  1.805899e-05   0.680779417  2.563939e-02
2.884245e-01
sugar_us.dl8      4.443374e+00  5.724688e-02  32.919623110 -2.118164e-01
2.687785e+01
gold.dl8          2.276995e-03 -3.049413e-05  -0.057413560 -2.099045e-03
7.242412e-04
silver.dl8       -2.296271e-02  1.514070e-03  -0.817218785  5.406725e-03
4.268790e-01
wheat_us_hrw.dl8 -2.187249e-02 -5.979705e-05  -0.280556993 -7.804640e-03
-5.688108e-02
soybeans.dl8      1.439431e-03 -2.129289e-05   0.066372490  1.704461e-03
4.036313e-02
crude_brent.dl9  -8.710242e-02 -3.446021e-04  -0.645293165 -3.068221e-02
-2.499779e-02
sugar_us.dl9     -5.731611e+00  9.370669e-02 -13.643164475 -5.712876e-01
-1.855612e+01
gold.dl9         -8.742006e-03 -7.990037e-05   0.010063273 -2.179932e-03
-8.844210e-03
silver.dl9        1.790346e-01  6.556699e-03   2.632054181  9.226511e-02
-6.589200e-01
wheat_us_hrw.dl9  2.204768e-02  1.583866e-04  -0.049009534 -2.626845e-04
5.458623e-02
soybeans.dl9     -1.178117e-02 -8.071755e-05  -0.041500078 -3.100956e-03
-4.736438e-02
                   soybeans.d
ect1               -0.042067984
ect2                3.903774572
crude_brent.dl1     0.204051301
sugar_us.dl1        2.738058777
gold.dl1            0.013139362
silver.dl1         -0.468856652
wheat_us_hrw.dl1    0.042123638
soybeans.dl1        0.143001047
crude_brent.dl2     0.037626365
sugar_us.dl2        0.953372979
gold.dl2           -0.034755933
silver.dl2         -0.149476237
wheat_us_hrw.dl2   -0.031973163
soybeans.dl2        0.104031712
crude_brent.dl3     0.018842485
sugar_us.dl3      -12.564948625
gold.dl3            0.031761445
silver.dl3         -1.053155733
wheat_us_hrw.dl3    0.079515848
soybeans.dl3       -0.046056360
crude_brent.dl4    -0.044825168
sugar_us.dl4      -32.849357088
gold.dl4           -0.001938407
silver.dl4         -1.094012458
```

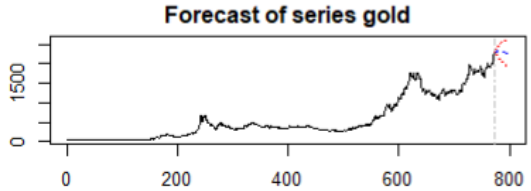
```
wheat_us_hrw.dl4  -0.004091014
soybeans.dl4       0.013767640
crude_brent.dl5    0.189083747
sugar_us.dl5      17.383663034
gold.dl5          -0.045200486
silver.dl5         0.635138660
wheat_us_hrw.dl5   0.114923796
soybeans.dl5      -0.050701177
crude_brent.dl6   -0.347167508
sugar_us.dl6      -2.461692703
gold.dl6           0.077024532
silver.dl6        -1.714897893
wheat_us_hrw.dl6   0.049901969
soybeans.dl6       0.016601673
crude_brent.dl7   -0.089750713
sugar_us.dl7     -18.536608290
gold.dl7          -0.029167392
silver.dl7        -0.224560593
wheat_us_hrw.dl7  -0.046720957
soybeans.dl7       0.065601259
crude_brent.dl8   -0.018616593
sugar_us.dl8      11.621798125
gold.dl8           0.105409014
silver.dl8        -1.143869613
wheat_us_hrw.dl8  -0.079282557
soybeans.dl8      -0.065353618
crude_brent.dl9   -0.308152625
sugar_us.dl9      27.540492232
gold.dl9          -0.035473102
silver.dl9         0.517312077
wheat_us_hrw.dl9   0.023097902
soybeans.dl9      -0.027087395
```

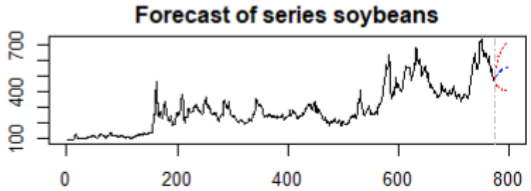












Interpretation:

The provided forecasts for various commodities, including crude oil (Brent), silver, sugar (US), wheat (US HRW), gold, and soybeans, reveal anticipated trends and potential future movements. Each plot shows historical price data followed by a forecasted range marked with a confidence interval. The forecasts for all commodities show a general upward trend with varying degrees of uncertainty.

- Crude Brent: Shows significant fluctuations historically, with the forecast indicating a continued upward trend but with high uncertainty.

- Silver: Similar to crude Brent, silver prices exhibit volatility with an upward forecast trend, again with a broad confidence interval.

- Sugar (US): Displays less volatility historically compared to other commodities, with the forecast suggesting a steady increase in prices.

- Wheat (US HRW): Historical data shows considerable volatility, with the forecast indicating a potential rise but with substantial uncertainty.

- Gold: Shows a strong upward trend historically, and the forecast suggests continued growth with some variability.

- Soybeans: Historical prices show periods of volatility, with the forecast indicating an upward trend but with a wide confidence interval.

Overall, the forecasts suggest that these commodities are expected to experience price increases, though the level of uncertainty varies across different commodities, reflecting market volatility and external influencing factors.

Results and Interpretation using Python

```python
# 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]}")
```

```
ADF test result for column: crude_brent

Test Statistic: -1.5078661910935425
P-value: 0.5296165197702358
Critical Values: {'1%': -3.439006442437876, '5%': -2.865360521688131, '10%': -2.5688044403756587}

ADF test result for column: soybeans

Test Statistic: -2.42314645274189
P-value: 0.13530977427790403
Critical Values: {'1%': -3.4388599939707056, '5%': -2.865295977855759, '10%': -2.5687700561872413}

ADF test result for column: gold

Test Statistic: 1.3430517021933006
P-value: 0.9968394353612382
Critical Values: {'1%': -3.4389608473398194, '5%': -2.8653404270188476, '10%': -2.568793735369693}

ADF test result for column: silver

Test Statistic: -1.3972947107462221
P-value: 0.5835723787985763
Critical Values: {'1%': -3.438915730045254, '5%': -2.8653205426302253, '10%': -2.5687831424305845}

ADF test result for column: sugar_us

Test Statistic: -2.276775914396525
P-value: 0.17956762453055886
Critical Values: {'1%': -3.4389608473398194, '5%': -2.8653404270188476, '10%': -2.568793735369693}

ADF test result for column: wheat_us_hrw

Test Statistic: -2.499023881611955
P-value: 0.11571200558506417
Critical Values: {'1%': -3.438915730045254, '5%': -2.8653205426302253, '10%': -2.5687831424305845}
```

Interpretation:

The results of the Augmented Dickey-Fuller (ADF) tests for various commodities, including crude oil (Brent), soybeans, gold, silver, sugar (US), and wheat (US HRW), assess the station arity of these time series data.

- Crude Brent: The test statistic is -1.5079 with a p-value of 0.5296, indicating that the series i s not stationary as the test statistic is higher than the critical values at all significance levels.
- Soybeans: The test statistic is -2.4231 with a p-value of 0.1353, suggesting non-stationarity since the test statistic does not exceed the critical value thresholds.

- Gold: The test statistic is 1.3430 with a p-value of 0.9968, clearly indicating non-stationarit y with a test statistic well above the critical values.
- Silver: The test statistic is -1.3973 with a p-value of 0.5837, showing non-stationarity as the test statistic is higher than the critical values.
- Sugar (US): The test statistic is -2.2768 with a p-value of 0.1796, also indicating non-station arity with the test statistic not exceeding critical value thresholds.
- Wheat (US HRW): The test statistic is -2.4990 with a p-value of 0.1157, suggesting non-stat ionarity as the test statistic is higher than the critical values at the 1%, 5%, and 10% levels.

```
# 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")
```

```
Johansen Test Results:

Eigenvalues:
[0.11398578 0.06876906 0.04867237 0.02710411 0.01899217 0.00405351]

Trace Statistic:
[226.10476071 132.67556204  77.67212223  39.15182203  17.93864778
   3.13567067]

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

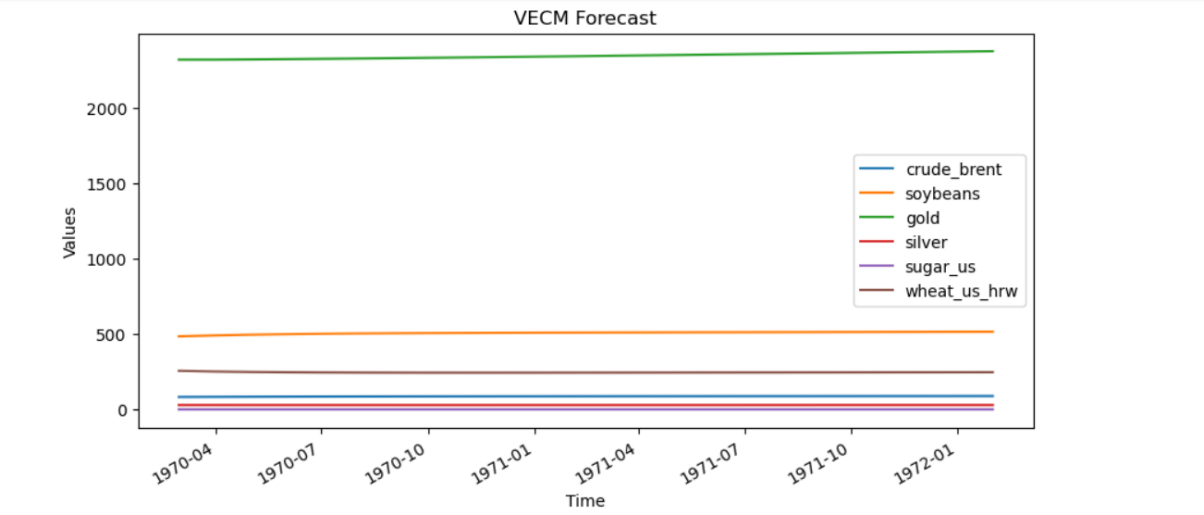
Interpretation:

The Johansen Test results indicate the presence of cointegration relationships among the variables in the system. The trace statistic values are compared against the critical values at the 5% significance level. Here, the trace statistics for the first four eigenvalues (226.10, 132.68, 77.67, and 39.15) exceed their corresponding critical values (95.75, 69.82, 47.85, and 29.80), suggesting the rejection of the null hypothesis of no cointegration for up to four cointegrating vectors. This implies that there are at least four long-run equilibrium relationships among the variables. The remaining trace statistics for the last two eigenvalues

(17.94 and 3.14) do not exceed their critical values (15.49 and 3.84), indicating that no additional cointegrating vectors exist beyond the first four.

```
# 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.co

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



Interpretation:

Based on the results of the vector error correction model (VECM) analysis, we can draw several insights into the relationships between the commodities analyzed: crude_brent, soybeans, gold, silver, sugar_us, and wheat_us_hrw. The coefficients from the VECM indicate how each variable responds to deviations from long-term equilibrium (cointegration relations) and short-term dynamics (lagged effects). The loading coefficients (alpha) demonstrate the speed at which each variable corrects towards the long-term equilibrium. For instance, crude_brent has a significant and negative loading coefficient for the first cointegration relation (`ec1`), indicating that it plays a crucial role in adjusting to equilibrium deviations. Soybeans also have a significant adjustment coefficient for both cointegration relations (`ec1` and `ec2`), suggesting that they actively respond to changes in the equilibrium relationship.

In terms of short-term dynamics, the lagged endogenous variables show how the past values of the variables affect each equation. The lag of crude_brent significantly affects its own future values, with a positive coefficient of 0.3217, indicating a strong short-term autocorrelation.

Soybeans have a significant lag effect on their own values as well (0.1574), highlighting the p ersistence of its price movements. The relationship between gold and silver is particularly not eworthy, as silver's own lagged value has a significant positive effect (0.3478), reflecting its s trong internal momentum. Overall, the analysis reveals the interconnectedness of these comm odities, where certain variables, like soybeans and crude_brent, have significant roles in both long-term equilibrium adjustment and short-term price dynamics, while others, such as silver and gold, exhibit distinct short-term relationships.

Recommendations

It is recommended that traders and analysts pay close attention to the lagged effects of each commodity in their trading strategies. Given the strong adjustment coefficients for crude_brent and soybeans, it's crucial to monitor their past values and their impact on future prices to better anticipate market movements and adjust positions accordingly. Additionally, since silver shows significant short-term momentum, incorporating its lagged values into trading decisions could provide valuable insights. Traders should also consider the long-term equilibrium relationships indicated by the cointegration coefficients, as these can inform broader strategic decisions and risk management. Overall, a comprehensive approach that integrates both short-term dynamics and long-term equilibrium adjustments will enhance forecasting accuracy and trading performance.

R Codes

```
# Set working directory and load necessary libraries
setwd('C:\\Users\\sayas\\OneDrive\\New folder\\python projects')  # Set the working d
irectory 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 and sheet
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(1,3,47,70,72,38,25)] %>%
  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-statio
nary 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, B
IC)
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
```

Python Codes

```
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("C:\\Users\\sayas\\OneDrive\\New folder\\python projects\\pinksh
eet.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', 'SUG
AR_US', 'WHEAT_US_HRW']]
# 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(')', '')
# 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-statio
nary 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.i
ndex[-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 commo
dity_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=commod
ity_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()
# Check all available attributes
print(dir(vecm_fitted))
print(vecm_fitted.summary())
```

References

1. www.github.com