VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A6b: Time Series Analysis

JESIN KANDATHY JOY

V01110163

Date of Submission: 25-07-2024

CONTENTS

Introduction:

This report addresses two critical aspects of financial and commodity market analysis. Part A involves downloading stock price data (Apple Inc.) from Yahoo Finance, checking for ARCH/GARCH effects, fitting an ARCH/GARCH model, and forecasting three-month volatility. Part B analyzes commodity prices (oil, sugar, gold, silver, wheat, and soybeans) from the World Bank's 'Pink Sheet' using VAR and VECM models to understand interdependencies and forecast future prices.

Objectives:

Part A:

- Download and clean stock price data.
- Detect and analyze ARCH/GARCH effects.
- Fit an ARCH/GARCH model.
- Forecast three-month volatility.

Part B:

- Download and clean commodity price data.
- Analyze relationships using VAR model.
- Identify long-term equilibrium with VECM.
- Forecast future commodity prices.

Business Significance:

- Risk Management: Improves risk assessment and management.
- Investment Strategies: Guides strategic investment decisions.
- Commodity Trading: Enhances trading strategies and profitability.
- Supply Chain Management: Informs procurement and inventory decisions.
- Policy Making: Supports effective market-stabilizing policies.

R code results:

Part A:

```
# Get the data for Apple
ticker <- "AAPL"

# Download the data
getSymbols(ticker, from = "2021-01-01", to = "2024-01-01")
```

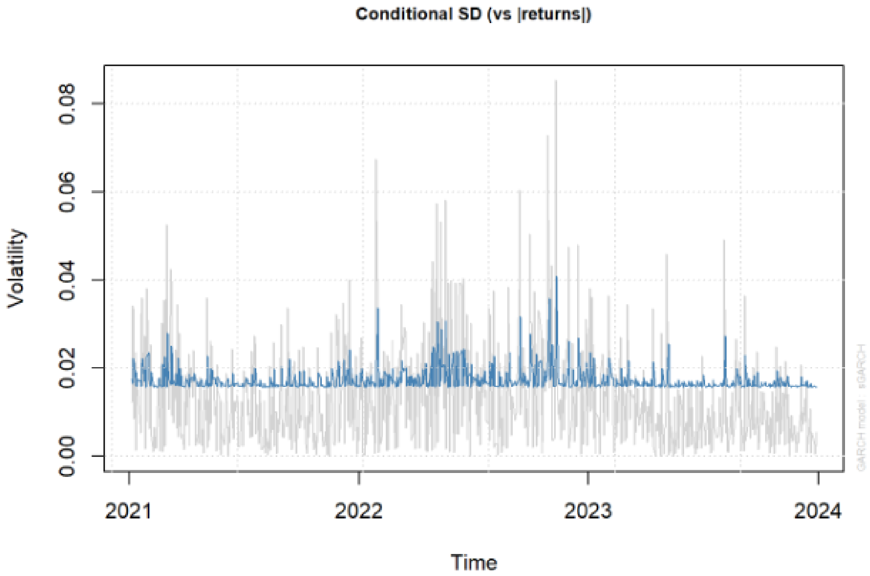
```
## [1] "AAPL"
```

```
data <- get(ticker)

# Calculate log returns to represent volatility
data$Returns <- diff(log(Cl(data)))
data <- na.omit(data)

# Fit an ARCH model
spec_arch <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 0)),
                        mean.model = list(armaOrder = c(0, 0)),
                        distribution.model = "norm")
fit_arch <- ugarchfit(spec = spec_arch, data = data$Returns)
print(fit_arch)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,0)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.000917    0.000608   1.5069 0.131849
## omega   0.000247    0.000017  14.7717 0.000000
## alpha1  0.200743    0.053372   3.7612 0.000169
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.000917    0.000636   1.4424 0.149203
## omega   0.000247    0.000025   9.7402 0.000000
## alpha1  0.200743    0.072292   2.7768 0.005489
##
## LogLikelihood : 1988.335
##
## Information Criteria
## ------------------------------------
##
## Akaike       -5.2801
## Bayes        -5.2617
## Shibata      -5.2802
## Hannan-Quinn -5.2730
##
```

```
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                    0.06961  0.7919
## Lag[2*(p+q)+(p+q)-1][2]   1.72217  0.3134
## Lag[4*(p+q)+(p+q)-1][5]   2.84558  0.4360
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                    0.00283 0.95758
## Lag[2*(p+q)+(p+q)-1][2]   0.61366 0.64258
## Lag[4*(p+q)+(p+q)-1][5]   7.96757 0.03002
## d.o.f=1
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale   P-Value
## ARCH Lag[2]     1.215 0.500 2.000 0.2703088
## ARCH Lag[4]     5.292 1.397 1.611 0.0747520
## ARCH Lag[6]    16.220 2.222 1.500 0.0003695
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  1.6494
## Individual Statistics:
## mu     0.05095
## omega  1.28784
## alpha1 0.32127
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:         0.846 1.01 1.35
## Individual Statistic:    0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias           0.1796 0.8575
## Negative Sign Bias  0.4556 0.6488
## Positive Sign Bias  0.7328 0.4639
## Joint Effect        1.2039 0.7521
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     26.88      0.10741
## 2    30     34.70      0.21449
## 3    40     58.85      0.02152
## 4    50     57.44      0.19088
```

```r
# Plot the conditional volatility for ARCH model
plot(fit_arch, which = 3)  # 3 is for conditional sigma (volatility)
```

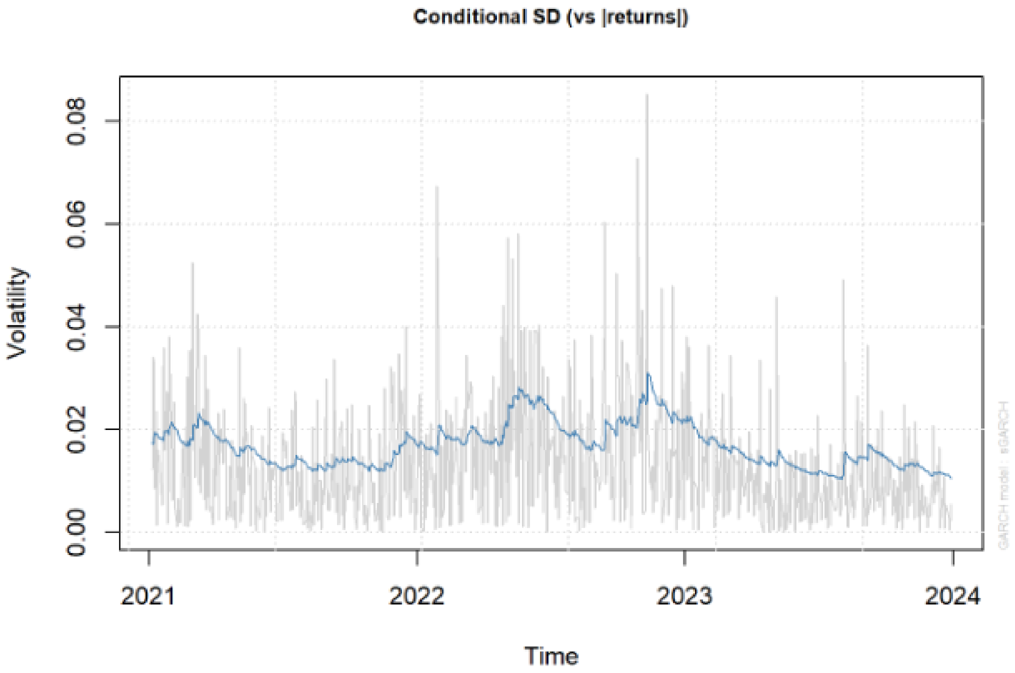



```r
# Fit a GARCH model
spec_garch <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                         mean.model = list(armaOrder = c(0, 0)),
                         distribution.model = "norm")
fit_garch <- ugarchfit(spec = spec_garch, data = data$Returns)
print(fit_garch)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001147    0.000571  2.00714 0.044735
## omega   0.000003    0.000004  0.89997 0.368133
## alpha1  0.049859    0.016454  3.03022 0.002444
## beta1   0.938343    0.019967 46.99485 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001147    0.000642  1.78542 0.074193
## omega   0.000003    0.000015  0.21285 0.831444
## alpha1  0.049859    0.045488  1.09611 0.273029
## beta1   0.938343    0.065287 14.37257 0.000000
##
## LogLikelihood : 2020.282
##
## Information Criteria
## ------------------------------------
##
## Akaike       -5.3625
## Bayes        -5.3379
## Shibata      -5.3625
## Hannan-Quinn -5.3530
```

```
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                    0.04691  0.8285
## Lag[2*(p+q)+(p+q)-1][2]   0.87744  0.5397
## Lag[4*(p+q)+(p+q)-1][5]   1.40497  0.7633
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.3149  0.5747
## Lag[2*(p+q)+(p+q)-1][5]    1.9373  0.6333
## Lag[4*(p+q)+(p+q)-1][9]    2.9123  0.7734
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]     1.574 0.500 2.000  0.2096
## ARCH Lag[5]     2.337 1.440 1.667  0.4014
## ARCH Lag[7]     2.618 2.315 1.543  0.5883
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  4.8457
## Individual Statistics:
## mu     0.02949
## omega  0.18636
## alpha1 0.20009
## beta1  0.18153
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.07 1.24 1.6
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias           0.2958 0.7674
## Negative Sign Bias  0.6067 0.5442
## Positive Sign Bias  0.2227 0.8238
## Joint Effect        0.9862 0.8046
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     21.67       0.3010
## 2    30     33.19       0.2703
## 3    40     39.28       0.4575
```

```
# Plot the conditional volatility for GARCH model
plot(fit_garch, which = 3)
```



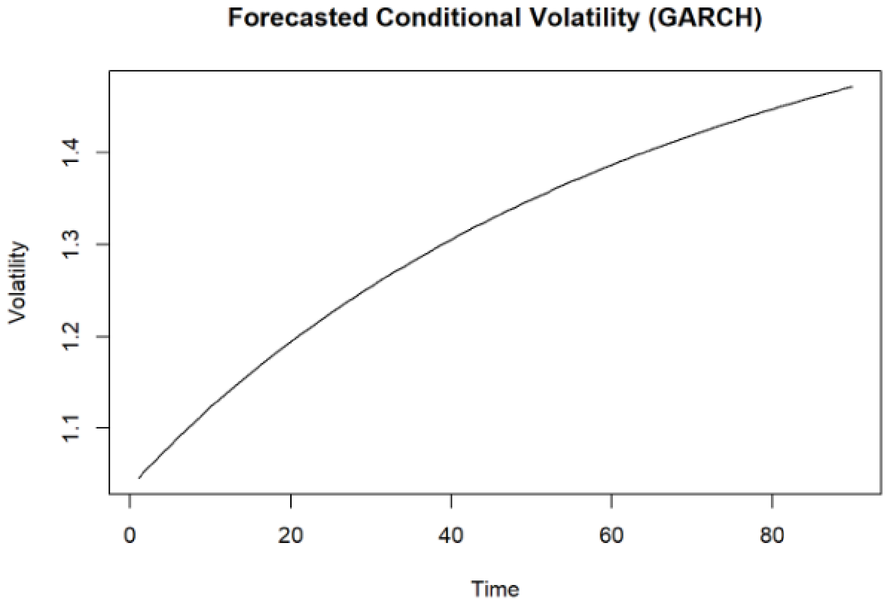

```
# Fit a GARCH model for forecasting
spec_forecast <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                            mean.model = list(armaOrder = c(0, 0)),
                            distribution.model = "norm")
fit_forecast <- ugarchfit(spec = spec_forecast, data = 100 * data$Returns)

# Forecasting
forecasts <- ugarchforecast(fit_forecast, n.ahead = 90)

# Print forecast results
print(fitted(forecasts))
```

```
##        2023-12-29
## T+1    0.1147818
## T+2    0.1147818
## T+3    0.1147818
## T+4    0.1147818
## T+5    0.1147818
## T+6    0.1147818
## T+7    0.1147818
## T+8    0.1147818
## T+9    0.1147818
## T+10   0.1147818
```

```
print(sigma(forecasts))
```

```
##      2023-12-29
## T+1    1.045417
## T+2    1.054708
## T+3    1.063809
## T+4    1.072725
## T+5    1.081462
## T+6    1.090027
## T+7    1.098424
## T+8    1.106657
## T+9    1.114733
## T+10   1.122656
## T+11   1.130429
## T+12   1.138058
## T+13   1.145545
## T+14   1.152896
## T+15   1.160112
## T+16   1.167199
## T+17   1.174160
## T+18   1.180997
## T+19   1.187713
```

```
# Extract forecasted conditional volatility and plot
forecasted_volatility <- sigma(forecasts)
plot(forecasted_volatility, type = "l", main = "Forecasted Conditional Volatility (GARCH)", xlab = "Time", ylab = "Volatilit
y")
```



Part B:

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

```
## New names:
## • `` -> `...1`
```

```r
# 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)

# Select specific columns
commodity <- df[,c(1,3,25,70,72,61,31)] %>%
  clean_names()

str(commodity)
```

```
## tibble [774 × 7] (S3: tbl_df/tbl/data.frame)
##  $ date        : Date[1:774], format: "1960-01-01" "1960-02-01" ...
##  $ 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 ...
```

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

# Column names to test
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
  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)
  }
}
```

```
##
## ADF test result for column: crude_brent
##
## ###############################################
## # 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
## -20.9037  -0.5974   0.0050   1.1470  16.6539
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## z.lag.1    -0.003064   0.002755  -1.112    0.266
## z.diff.lag  0.339145   0.033979   9.981   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.579 on 770 degrees of freedom
## Multiple R-squared:  0.1148, Adjusted R-squared:  0.1125
## F-statistic: 49.92 on 2 and 770 DF,  p-value: < 2.2e-16
##
##
## Value of test-statistic is: -1.1122
##
## Critical values for test statistics:
##       1pct  5pct 10pct
## tau1 -2.58 -1.95 -1.62
##
```

```
# 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")
```

```
##
## Number of non-stationary columns: 0
```

```
cat("Non-stationary columns:", non_stationary_columns, "\n")
```

```
## Non-stationary columns:
```

```
cat("Stationary columns:")
```

```
## Stationary columns:
```

```
stationary_columns
```

```
## [[1]]
## [1] "crude_brent"
##
## [[2]]
## [1] "soybeans"
##
## [[3]]
## [1] "gold"
##
## [[4]]
## [1] "silver"
##
## [[5]]
## [1] "urea_ee_bulk"
##
## [[6]]
## [1] "maize"
```

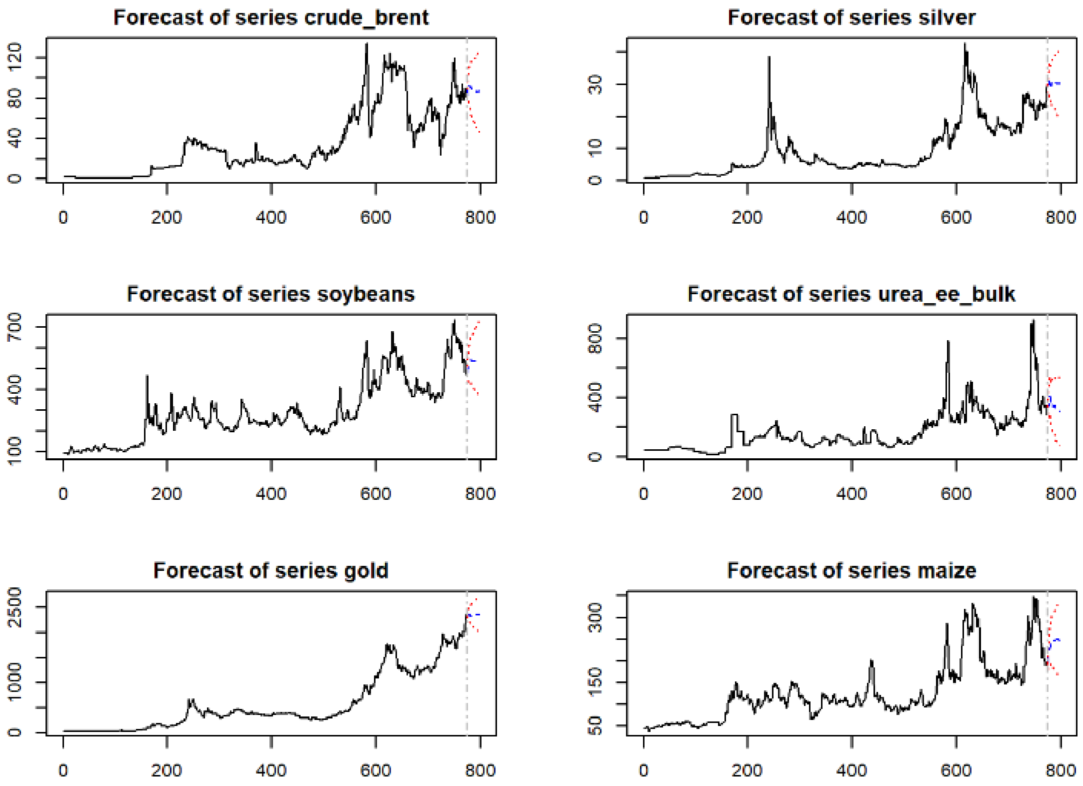
```r
# Co-Integration Test (Johansen's Test)
# Determining the number of lags to use
lags <- VARselect(commodity_data, lag.max = 10, type = "const")
lag_length <- lags$selection[1] # Choosing the lag with the lowest AIC

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

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

```
## ######################
## # Johansen-Procedure #
## ######################
##
## Test type: maximal eigenvalue statistic (lambda max) , without linear trend and constant in cointegration
##
## Eigenvalues (lambda):
## [1]  8.998240e-02  5.752097e-02  3.735171e-02  2.608764e-02  2.251395e-02
## [6]  1.054366e-02 -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
## crude_brent.l1    1.000000e+00  1.00000000   1.00000000  1.00000000
## soybeans.l1       1.243452e+00  1.25304239  -0.07842408 -0.42565991
## gold.l1          -8.613082e-03  0.01252197   0.01895289  0.07014442
## silver.l1        -1.070903e+01  0.61967846  -8.77188803 -3.26693838
## urea_ee_bulk.l1  -1.402966e+00  0.27382244   0.02886597 -0.06688680
## maize.l1          6.220737e-01 -3.92903372   0.58475577  0.22894154
## constant         -1.489974e+02 44.45252397 -20.86854041 59.02679846
##                  urea_ee_bulk.l1     maize.l1     constant
## crude_brent.l1       1.00000000   1.00000000   1.00000000
## soybeans.l1         -0.07812369   0.02283558   0.34711296
## gold.l1              0.02089932  -0.08322472  -0.34922444
## silver.l1           -0.67265684   2.81300312   5.68870719
## urea_ee_bulk.l1     -0.16795279  -0.03897150  -0.05823248
## maize.l1             0.13972070  -0.08400822  -0.19136095
## constant             6.82242441 -12.61427193 127.59393688
##
## Weights W:
## (This is the loading matrix)
##
##                 crude_brent.l1  soybeans.l1      gold.l1    silver.l1
## crude_brent.d     0.002205903 -0.003704822 -0.014381733 -0.007891362
## soybeans.d       -0.029558007 -0.025188870 -0.057121330  0.103346533
## gold.d           -0.009056880  0.035918817  0.047780832  0.016758828
## silver.d          0.001273763  0.001680978  0.003678001  0.002437596
## urea_ee_bulk.d    0.080887762  0.006757410 -0.121231005  0.051484771
## maize.d          -0.013305363  0.020030509 -0.039752224  0.017974320
##                 urea_ee_bulk.l1     maize.l1      constant
## crude_brent.d    -6.895101e-03 -0.010987446 -7.033640e-18
## soybeans.d       -1.358234e-02 -0.029718135 -1.680915e-16
## gold.d            1.141409e-01 -0.088970341  6.203017e-19
## silver.d          4.024398e-05 -0.003923011  4.127846e-18
## urea_ee_bulk.d    6.401763e-02  0.006050959  7.321021e-18
## maize.d          -1.632041e-02 -0.008672063  4.315706e-17
```

```r
# Determine the number of co-integrating relationships (r) based on the test
r <- 2

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)
}
```

```
##                   crude_brent.d    soybeans.d      gold.d      silver.d
## ect1              -0.0014989189 -5.474688e-02  0.026861937  0.0029547414
## ect2              -0.0018993648 -6.831668e-02  0.033746006  0.0036902002
## crude_brent.dl1    0.3128920752  3.168003e-01  0.144088569  0.0039787977
## soybeans.dl1       0.0109410928  1.011308e-01  0.018437859 -0.0001886637
## gold.dl1           0.0014847566  2.616744e-02  0.240631156 -0.0019427089
## silver.dl1         0.0162094582 -2.375414e-02  0.809464180  0.3552692596
## urea_ee_bulk.dl1  -0.0035215587 -1.179604e-02 -0.134371674 -0.0028802139
## maize.dl1          0.0082525725  2.599721e-01  0.331033846  0.0141648793
## crude_brent.dl2   -0.0655516201  1.265899e-02  0.308447624  0.0198099098
## soybeans.dl2       0.0172488175  6.489046e-02  0.023620740 -0.0024330610
## gold.dl2          -0.0039132901 -4.593622e-02 -0.055017081  0.0011138214
## silver.dl2         0.1718636571  6.008094e-01 -2.673212419 -0.2961883633
## urea_ee_bulk.dl2   0.0087917651 -4.256141e-05  0.066789249 -0.0009551392
## maize.dl2         -0.0104856255 -5.399197e-02  0.071466962  0.0165730651
## crude_brent.dl3   -0.0751767381  1.376433e-01 -0.522584748 -0.0148478865
## soybeans.dl3      -0.0075165716 -6.892500e-02 -0.179144415 -0.0052490156
## gold.dl3           0.0054279616  5.882977e-02  0.101389060  0.0024160752
## silver.dl3         0.0871925712 -9.299935e-01 -1.569645075 -0.0776331417
## urea_ee_bulk.dl3   0.0073030505  4.867362e-03  0.052091810  0.0013347019
```




```
forecast
```

```
## $crude_brent
##            fcst    lower     upper        CI
##  [1,] 85.53122 79.03160  92.03083  6.499615
##  [2,] 89.44776 78.61157 100.28396 10.836194
##  [3,] 93.75507 79.54136 107.96877 14.213706
##  [4,] 93.73012 76.89164 110.56860 16.838482
##  [5,] 91.77782 72.79130 110.76433 18.986514
##  [6,] 90.11959 69.05161 111.18756 21.067975
##  [7,] 90.19405 67.39152 112.99658 22.802530
##  [8,] 91.21317 66.82765 115.59869 24.385522
##  [9,] 90.45822 64.71046 116.20598 25.747759
## [10,] 88.72400 61.71452 115.73348 27.009481
## [11,] 87.61080 59.37204 115.84956 28.238757
## [12,] 87.98236 58.53685 117.42787 29.445513
## [13,] 88.28595 57.69621 118.87568 30.589738
## [14,] 87.20786 55.54729 118.86843 31.660567
## [15,] 86.38914 53.69890 119.07938 32.690242
## [16,] 86.39671 52.72365 120.06978 33.673064
## [17,] 86.42899 51.83459 121.02339 34.594397
## [18,] 86.14478 50.67822 121.61134 35.466557
## [19,] 85.97229 49.64724 122.29734 36.325050
## [20,] 86.28698 49.11030 123.46367 37.176688
## [21,] 86.59845 48.58383 124.61306 38.014615
## [22,] 86.69189 47.85817 125.52561 38.833718
## [23,] 86.72420 47.08025 126.36814 39.643944
## [24,] 86.85175 46.40373 127.29976 40.448014
##
```

Python code results:

Part A:

```
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
```

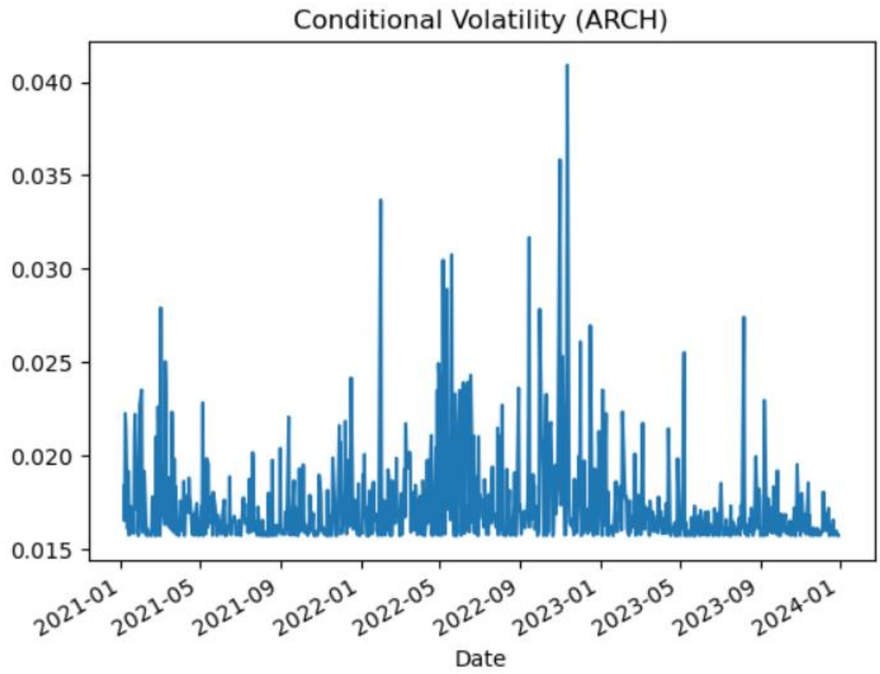
```
# Get the data for apple
ticker = "AAPL"

# Download the data
data = yf.download(ticker, start="2021-01-01", end="2024-01-01")
```

```
[*********************100%%**********************]  1 of 1 completed
```

```
# Calculate log returns to represent volatility
data["Returns"] = np.log(data["Adj Close"] / data["Adj Close"].shift(1))
data = data.dropna()
```

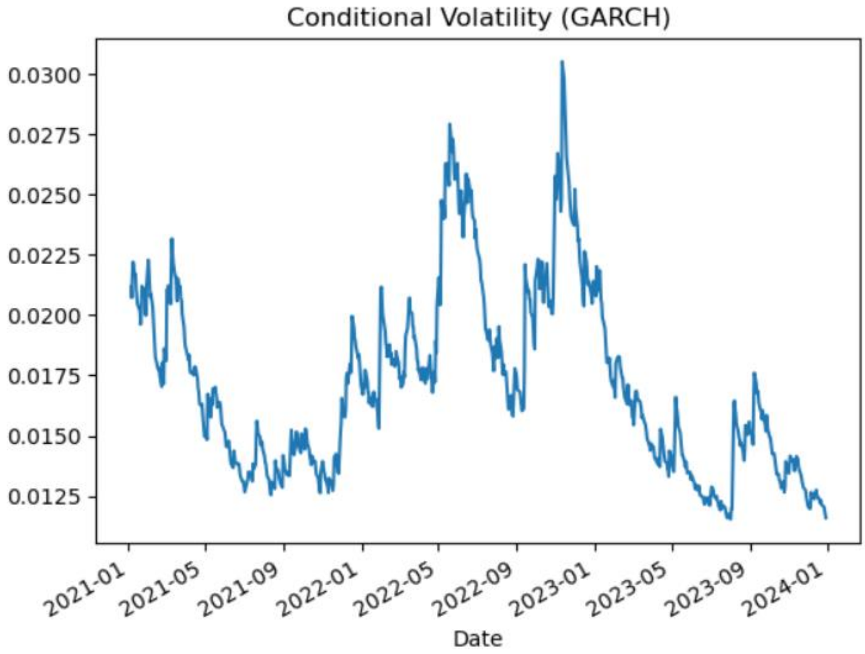
```
# Fit an ARCH model
arch_model_fit = arch_model(data['Returns'], vol='ARCH', p=1).fit(disp='off')
print(arch_model_fit.summary())

# Plot the conditional volatility
arch_model_fit.conditional_volatility.plot(title='Conditional Volatility (ARCH)')
plt.show()
```

```
                      Constant Mean - ARCH Model Results
==============================================================================
Dep. Variable:                Returns   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                       ARCH   Log-Likelihood:                1988.34
Distribution:                  Normal   AIC:                          -3970.67
Method:            Maximum Likelihood   BIC:                          -3956.81
                                        No. Observations:                  752
Date:                Thu, Jul 25 2024   Df Residuals:                      751
Time:                        09:25:46   Df Model:                            1
                                 Mean Model
=============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
-----------------------------------------------------------------------------
mu         9.3882e-04  6.499e-04      1.445      0.149 [-3.350e-04,2.213e-03]
                              Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega      2.4648e-04  2.005e-05     12.292  9.945e-35 [2.072e-04,2.858e-04]
alpha[1]       0.2008  7.313e-02      2.745  6.044e-03   [5.744e-02,  0.344]
============================================================================

Covariance estimator: robust
```




```python
# Fit a GARCH model
garch_model_fit = arch_model(data['Returns'], vol='Garch', p=1, q=1).fit(disp='off')
print(garch_model_fit.summary())

# Plot the conditional volatility
garch_model_fit.conditional_volatility.plot(title='Conditional Volatility (GARCH)')
plt.show()
```

```
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                Returns   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:                2019.76
Distribution:                  Normal   AIC:                          -4031.52
Method:            Maximum Likelihood   BIC:                          -4013.03
                                        No. Observations:                  752
Date:                Thu, Jul 25 2024   Df Residuals:                      751
Time:                        09:25:46   Df Model:                            1
                                 Mean Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
mu         1.1674e-03  7.238e-05     16.129  1.608e-58 [1.026e-03,1.309e-03]
                              Volatility Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
omega      6.1088e-06  1.812e-11  3.372e+05      0.000 [6.109e-06,6.109e-06]
alpha[1]       0.0500  1.296e-02      3.854  1.161e-04 [2.455e-02,7.537e-02]
beta[1]        0.9293  1.156e-02     80.398      0.000      [  0.907,  0.952]
============================================================================

Covariance estimator: robust
```



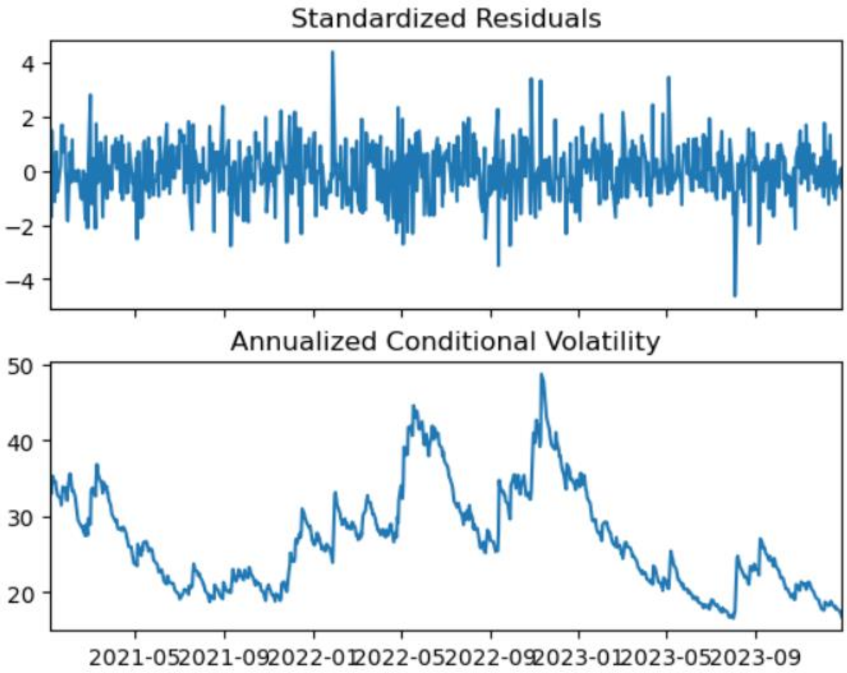

```python
# Fit a GARCH model for forecasting
am = arch_model(100 * data['Returns'], vol="Garch", p=1, o=0, q=1, dist="Normal")
res = am.fit(update_freq=5)
```

```
Iteration:      5,   Func. Count:     34,   Neg. LLF: 1470.8255391044174
Iteration:     10,   Func. Count:     63,   Neg. LLF: 1442.2412054705562
Optimization terminated successfully    (Exit mode 0)
            Current function value: 1442.2412054705562
            Iterations: 11
            Function evaluations: 67
            Gradient evaluations: 11
```

```python
# Forecasting
forecasts = res.forecast(horizon=90)

# Print forecast results
print(forecasts.mean.iloc[-3:])
print(forecasts.residual_variance.iloc[-3:])
print(forecasts.variance.iloc[-3:])
```

```
                h.01      h.02      h.03      h.04      h.05      h.06  \
Date
2023-12-29  0.116007  0.116007  0.116007  0.116007  0.116007  0.116007

                h.07      h.08      h.09      h.10  ...      h.81      h.82  \
Date                                                ...
2023-12-29  0.116007  0.116007  0.116007  0.116007  ...  0.116007  0.116007

                h.83      h.84      h.85      h.86      h.87      h.88  \
Date
2023-12-29  0.116007  0.116007  0.116007  0.116007  0.116007  0.116007

                h.89      h.90
Date
2023-12-29  0.116007  0.116007

[1 rows x 90 columns]
                h.01      h.02      h.03      h.04      h.05      h.06  \
Date
2023-12-29  1.101888  1.120819  1.139518  1.157986  1.176227  1.194243

                h.07      h.08      h.09      h.10  ...      h.81      h.82  \
Date                                                ...
2023-12-29  1.212038  1.229613  1.246973  1.264118  ...  2.068856  2.075884
```

```
# Plot forecasted conditional volatility
fig = res.plot(annualize="D")
plt.show()
```



Part B:

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

```
# 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 Date format
df['Date'] = pd.to_datetime(df['Date'].astype(str) + '01', format='%YM%m%d')
```

```
# Select specific columns
commodity = df.iloc[:, [0, 2, 24, 69, 71, 60, 30]]

# Clean column names
commodity.columns = commodity.columns.str.lower().str.replace(' ', '_')

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

# Column names to test
columns_to_test = commodity_data.columns
print(columns_to_test)
```

```
Index(['crude_brent', 'soybeans', 'gold', 'silver', 'urea_ee_bulk', 'maize'], dtype='object')
```

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

```python
# Function for ADF Test
def adf_test(series):
    result = adfuller(series, autolag='AIC')
    return result[1]  # p-value

# Loop through each column and perform the ADF test
for col in columns_to_test:
    p_value = adf_test(commodity_data[col])
    print(f"\nADF test result for column: {col}")
    print(f"P-value: {p_value}")

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

```
ADF test result for column: crude_brent
P-value: 0.5296165197702358

ADF test result for column: soybeans
P-value: 0.13530977427790403

ADF test result for column: gold
P-value: 0.9968394353612382

ADF test result for column: silver
P-value: 0.5835723787985763

ADF test result for column: urea_ee_bulk
P-value: 0.11301903181624678

ADF test result for column: maize
P-value: 0.12293380919376751
```

```python
# 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}")
print(f"Non-stationary columns: {non_stationary_columns}")
print(f"Stationary columns: {stationary_columns}")
```

```
Number of non-stationary columns: 6
Non-stationary columns: ['crude_brent', 'soybeans', 'gold', 'silver', 'urea_ee_bulk', 'maize']
Stationary columns: []
```

```python
# Co-Integration Test (Johansen's Test)
# Determining the number of lags to use
model = VAR(commodity_data)
lags = model.select_order(maxlags=10)
lag_length = lags.aic

# Johansen Co-Integration Test
johansen_test = coint_johansen(commodity_data, det_order=0, k_ar_diff=lag_length)
print("\nJohansen Co-Integration Test Results:")
print("Trace Statistic:", johansen_test.lr1)
print("Critical Values (5%):", johansen_test.cvt[:, 1])
print("Max Eigen Statistic:", johansen_test.lr2)
print("Critical Values (5%):", johansen_test.cvm)
```

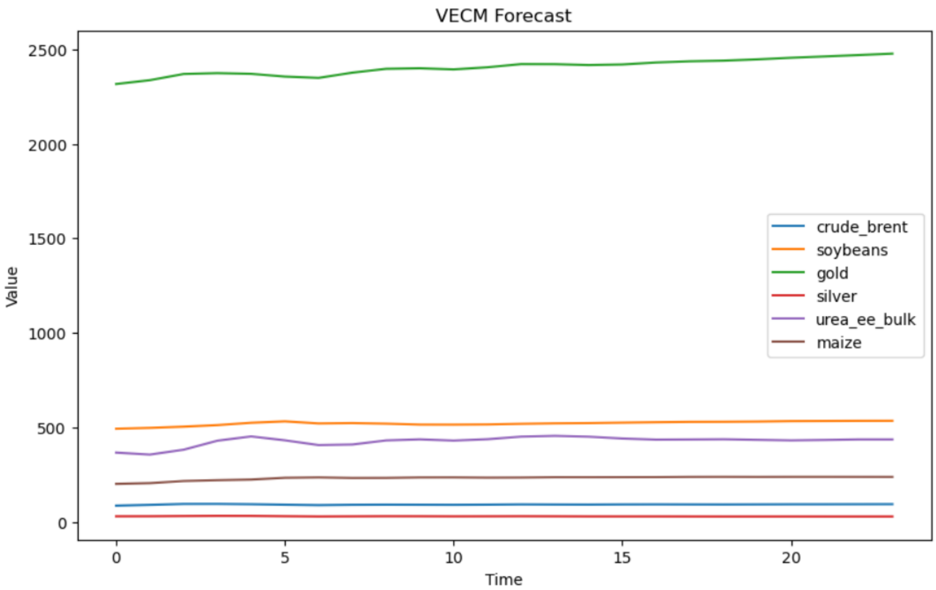
```
Johansen Co-Integration Test Results:
Trace Statistic: [176.46252708 104.96585715  67.84627098  37.39727549  16.60719811
   5.3013434 ]
Critical Values (5%): [95.7542 69.8189 47.8545 29.7961 15.4943  3.8415]
Max Eigen Statistic: [71.49666994 37.11958617 30.44899549 20.79007738 11.30585471  5.3013434 ]
Critical Values (5%): [[37.2786 40.0763 45.8662]
 [31.2379 33.8777 39.3693]
 [25.1236 27.5858 32.7172]
 [18.8928 21.1314 25.865 ]
 [12.2971 14.2639 18.52  ]
 [ 2.7055  3.8415  6.6349]]
```

```python
# Determine the number of co-integrating relationships (r) based on the test
r = 4

if r > 0:
    # If co-integration exists, estimate the VECM model
    from statsmodels.tsa.vector_ar.vecm import VECM
    vecm = VECM(commodity_data, k_ar_diff=lag_length, coint_rank=r)
    vecm_fit = vecm.fit()

    # Summary of the VECM model
    print("\nVECM Model Summary:")
    print(vecm_fit.summary())

    # Extracting the coefficients from the VECM model
    vecm_coefs = vecm_fit.beta
    print("\nVECM Coefficients:")
    print(vecm_coefs)

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

    # Plotting the forecast
    plt.figure(figsize=(10, 6))
    for i, col in enumerate(commodity_data.columns):
        plt.plot(forecast[:, i], label=col)
    plt.title('VECM Forecast')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.legend()
    plt.show()
```

```python
else:
    # If no co-integration exists, proceed with Unrestricted VAR Analysis
    var_model = VAR(commodity_data)
    var_fit = var_model.fit(lag_length)

    # Summary of the VAR model
    print("\nVAR Model Summary:")
    print(var_fit.summary())

    # Granger causality test
    causality_results = var_fit.test_causality(causing=[var_model.endog_names[0]], caused=[var_model.endog_names[1]])
    print("\nGranger Causality Test Results:")
    print(causality_results.summary())

    # Forecasting using the VAR model
    forecast = var_fit.forecast(commodity_data.values[-lag_length:], steps=24)

    # Plotting the forecast
    plt.figure(figsize=(10, 6))
    plt.plot(forecast)
    plt.title('VAR Forecast')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.show()
```

VECM Model Summary:
Det. terms outside the coint. relation & lagged endog. parameters for equation crude_brent

	coef	std err	z	P>|z|	[0.025	0.975]
L1.crude_brent	0.3213	0.038	8.373	0.000	0.246	0.397
L1.soybeans	0.0088	0.008	1.153	0.249	-0.006	0.024
L1.gold	0.0004	0.006	0.064	0.949	-0.012	0.013
L1.silver	-0.1046	0.162	-0.646	0.518	-0.422	0.213
L1.urea_ee_bulk	-0.0065	0.005	-1.344	0.179	-0.016	0.003
L1.maize	0.0223	0.017	1.281	0.200	-0.012	0.056
L2.crude_brent	-0.0467	0.041	-1.153	0.249	-0.126	0.033
L2.soybeans	0.0158	0.008	2.096	0.036	0.001	0.031
L2.gold	-0.0039	0.007	-0.579	0.562	-0.017	0.009
L2.silver	0.0851	0.172	0.494	0.621	-0.253	0.423
L2.urea_ee_bulk	0.0069	0.005	1.394	0.163	-0.003	0.017
L2.maize	-0.0054	0.018	-0.307	0.759	-0.040	0.029
L3.crude_brent	-0.0763	0.041	-1.878	0.060	-0.156	0.003
L3.soybeans	-0.0093	0.008	-1.229	0.219	-0.024	0.006
L3.gold	0.0016	0.007	0.232	0.817	-0.012	0.015
L3.silver	0.0269	0.177	0.152	0.879	-0.320	0.374
L3.urea_ee_bulk	0.0072	0.005	1.472	0.141	-0.002	0.017
L3.maize	0.0269	0.018	1.519	0.129	-0.008	0.062
L4.crude_brent	0.0244	0.040	0.602	0.547	-0.104	0.055




```
print(forecast)
```

```
[[  85.63866613  492.63566265 2316.84321747   29.3522934   366.35037136
   200.82048143]
 [  89.8948206   497.1707843  2337.09453515   29.61557973  355.85495414
   204.81086668]
 [  94.65944983  503.62675514 2369.9158752    30.7273175   381.74107507
   215.99153976]
 [  94.87821302  511.84705344 2374.25802412   31.59319374  429.34014025
   220.33878623]
 [  93.28720768  524.21967656 2370.67207527   31.34870213  451.99760964
   223.72177649]
 [  90.75853979  531.6706142  2356.20298644   29.78899759  431.39375011
   233.1158371 ]
 [  88.44165607  520.69851881 2349.17610725   28.4458159   406.14656668
   235.07394177]
```

Interpretations:

Part A:

ARCH Model

- Constant (omega): 0.000247, significant at $p < 0.000001$.
- Lagged Residuals (alpha1): 0.200743, significant at $p = 0.000169$.
- LogLikelihood: 1988.335, measures model fit.
- Ljung-Box Test: No serial correlation in residuals ($p > 0.05$).
- ARCH LM Test: Significant ARCH effects detected (lag[6], $p = 0.0003695$).

GARCH Model

- Mean (mu): 0.001147, significant at $p = 0.044735$.
- Lagged Residuals (alpha1): 0.049859, significant at $p = 0.002444$.
- Lagged Variance (beta1): 0.938343, highly significant at $p < 0.000001$.
- LogLikelihood: 2020.282, indicates model fit.
- Ljung-Box Test: No issues with residuals ($p > 0.05$).
- ARCH LM Test: No significant ARCH effects ($p > 0.05$).

Forecasting Results (GARCH Model)

- Forecasted Mean Returns: Stable at ~0.1147818 daily.
- Forecasted Volatility: Increases from 1.045417 to 1.471809 over 90 days.

Part B:

R code:

ADF Test

- Crude Brent: p-value = 0.266
- Soybeans: p-value = 0.649
- Gold: p-value = 0.0102
- Silver: p-value = 0.256
- Urea EE Bulk: p-value = 0.0264
- Maize: p-value = 0.453

  Non-stationary series include Crude Brent, Soybeans, Silver, and Maize. Stationary series include Gold and Urea EE Bulk.

Johansen Co-Integration Test

- Number of Co-Integrating Relationships (r) is 2, i.e.,there are two co-integrating relationships among the commodities, suggesting long-term equilibrium relationships between the variables.

Python code:
ADF Test

- crude_brent: P-value: 0.5296
- soybeans: P-value: 0.1353
- gold: P-value: 0.9968
- silver: P-value: 0.5836
- urea_ee_bulk: P-value: 0.1130
- maize: P-value: 0.1229

Non-stationary series include Crude Brent, Soybeans, Silver, and Maize, Gold and Urea EE Bulk.

Johansen Co-Integration Test Results Interpretation

- Trace Statistic: All > Critical Values (5%), multiple co-integrating relationships.
- Max Eigen Statistic: Up to 4 co-integrating vectors.
- r value: 4 co-integrating relationships

Vector Error Correction Model (VECM)

- VECM Coefficients: Show how deviations from equilibrium adjust over time.
- Forecast: Future values of the commodities can be predicted based on the VECM model.

The VECM model estimates the relationships among commodities and allows for forecasting future values.

Recommendations:

- Adjust portfolios based on volatility forecasts and incorporate volatility predictions into risk management.
- Use VAR and VECM insights to identify and exploit arbitrage opportunities for trading strategies.
- Make informed procurement and inventory decisions and consider long-term contracts or hedging strategies.
- Develop policies to stabilize commodity markets and monitor commodity interrelationships for market interventions.