

**VIRGINIA COMMONWEALTH UNIVERSITY**

**Statistical analysis and modelling (SCMA 632)**

**A6b- Time Series Analysis  
(Part – A)**

**ADHYAYAN AMIT JAIN  
V01109421**

**Date of Submission: 25-07-2024**

# CONTENTS

Sl. No.	Title	Page No.
1.	Introduction	1
2.	Results	5
3.	Interpretations	5
4.	Implications	24
5.	Recommendations	26
6	Codes	28
7.	References	36

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

## INTRODUCTION

In financial markets, understanding the volatility of asset returns is crucial for investors, risk managers, and policymakers. Volatility, a measure of the variability of asset prices, provides insights into market risks and the stability of investments. This assignment focuses on analyzing the volatility of NVIDIA Corporation (NVDA) stock using advanced econometric models, specifically the ARCH (Autoregressive Conditional Heteroskedasticity) and GARCH (Generalized Autoregressive Conditional Heteroskedasticity) models.

NVIDIA, a prominent player in the technology sector, is known for its innovations in graphics processing units (GPUs) and artificial intelligence (AI). The company's stock has experienced significant fluctuations, making it an ideal candidate for volatility analysis. By examining the volatility of NVDA stock, this study aims to provide a deeper understanding of its price dynamics and forecast future risk levels.

The assignment involves several key steps:

1. **Data Acquisition:** Historical stock price data for NVIDIA will be sourced from Yahoo Finance, covering a specific period.
2. **Return Calculation:** The percentage returns of the stock will be calculated to measure the relative change in price over time.
3. **Model Fitting:** Both ARCH and GARCH models will be applied to the returns data to estimate and analyze conditional volatility.
4. **Forecasting:** The models will be used to generate forecasts of volatility, including short-term and long-term predictions.
5. **Evaluation:** Residuals from the models will be tested for autocorrelation using the Ljung-Box test to ensure the robustness of the models.

The insights gained from this analysis will help in assessing the risk associated with NVIDIA's stock and can inform investment decisions and risk management strategies. The results of this study will be visualized through various plots, including conditional volatility and forecasted variance, providing a comprehensive view of NVDA's financial volatility.

# OBJECTIVES

The primary objectives of this assignment are to analyze the volatility of NVIDIA Corporation (NVDA) stock using advanced econometric techniques and to generate forecasts for future volatility. Specifically, the objectives are:

## 1. Historical Data Analysis:

- **Objective:** To acquire and preprocess historical stock price data for NVIDIA from Yahoo Finance.
- **Details:** Retrieve data from April 1, 2021, to March 31, 2024, focusing on adjusted closing prices to ensure accurate returns calculation.

## 2. Return Calculation:

- **Objective:** To compute the percentage returns of NVIDIA's stock to measure price changes over time.
- **Details:** Calculate daily returns based on the adjusted closing prices and convert these returns into percentage terms.

## 3. Model Specification and Fitting:

- **Objective:** To apply and fit both ARCH and GARCH models to the returns data to estimate conditional volatility.
- **Details:**
  - **ARCH Model:** Specify and fit an ARCH(1) model to capture the time-varying volatility in the data.
  - **GARCH Model:** Specify and fit a GARCH(1, 1) model to account for both autoregressive and moving average components of volatility.

## 4. Forecasting Volatility:

- **Objective:** To generate forecasts of future volatility using the fitted GARCH model.
- **Details:**
  - **Short-Term Forecast:** Forecast the volatility for the next day.

- **Long-Term Forecast:** Extend forecasts to a 90-day horizon to analyze potential future volatility trends.

## 5. Residual Analysis:

- **Objective:** To evaluate the residuals of the fitted models for autocorrelation to assess model adequacy.
- **Details:** Perform the Ljung-Box test on residuals from both the ARCH and GARCH models to verify the presence of any remaining autocorrelation.

## 6. Visualization and Interpretation:

- **Objective:** To visualize and interpret the results of the volatility analysis.
- **Details:**
  - **Conditional Volatility Plots:** Create time series plots to display the estimated conditional volatility from both the ARCH and GARCH models.
  - **Forecast Plots:** Plot the forecasted volatility and variance for the 90-day horizon to provide insights into future risk levels.

## 7. Summary and Recommendations:

- **Objective:** To summarize the findings and provide actionable insights based on the analysis.
- **Details:**
  - **Model Performance:** Discuss the performance of the ARCH and GARCH models.
  - **Investment Implications:** Offer recommendations for investors based on the volatility forecasts and residual analysis.

This structured approach aims to provide a thorough understanding of NVIDIA's stock volatility and to support informed decision-making through detailed econometric analysis and forecasting.

# BUSINESS SIGNIFICANCE

Understanding and analyzing stock volatility is crucial for making informed investment decisions and managing financial risks. The business significance of this assignment can be outlined as follows:

## 1. Risk Management:

- **Objective:** Volatility forecasts help in assessing the risk associated with investing in NVIDIA stock.
- **Details:** By quantifying expected future volatility, investors can better understand the potential fluctuations in the stock price. This information is essential for managing investment risk and implementing appropriate hedging strategies.

## 2. Investment Strategy:

- **Objective:** To enhance portfolio management and asset allocation decisions.
- **Details:** Accurate volatility predictions allow investors to adjust their portfolios according to their risk tolerance. For example, higher predicted volatility might lead to a more conservative investment approach or the diversification of investments to mitigate potential losses.

## 3. Pricing and Valuation:

- **Objective:** To improve the pricing and valuation of financial instruments linked to NVIDIA stock.
- **Details:** Instruments such as options and futures derive their pricing models from volatility estimates. By understanding future volatility trends, investors can better price these derivatives and assess their fair value.

## 4. Strategic Planning for Companies:

- **Objective:** To inform strategic decisions within NVIDIA or similar companies.
- **Details:** High volatility can impact a company's stock price, influencing strategic decisions such as capital raising, mergers and acquisitions, and dividend policies. Companies can use volatility forecasts to time their financial strategies more effectively.

## 5. Market Sentiment Analysis:

- **Objective:** To gauge market sentiment and investor behavior.
- **Details:** Volatility is often a reflection of market sentiment and uncertainty. Understanding periods of high or low volatility can provide insights into market sentiment and help investors anticipate market trends.

## 6. Regulatory and Compliance Considerations:

- **Objective:** To ensure compliance with financial regulations and standards.
- **Details:** Financial regulations often require firms to disclose information about risk and volatility. Accurate and timely volatility analysis supports compliance with these regulations and enhances transparency for stakeholders.

#### 7. Performance Evaluation:

- **Objective:** To evaluate the effectiveness of trading strategies and financial models.
- **Details:** By comparing actual market outcomes with model forecasts, investors and analysts can assess the performance of their trading strategies and financial models. This evaluation helps in refining approaches and improving predictive accuracy.

In summary, the ability to accurately forecast and understand stock volatility provides valuable insights that can drive better investment decisions, enhance financial risk management, and support strategic corporate actions. This analysis not only benefits individual investors but also contributes to the broader financial ecosystem by promoting informed decision-making and market stability.

---

## RESULTS AND INTERPRETATIONS

### Python Language

#### Detailed Code Analysis

##### 1. Import Required Libraries

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from statsmodels.stats.diagnostic import acorr_ljungbox
import seaborn as sns
```

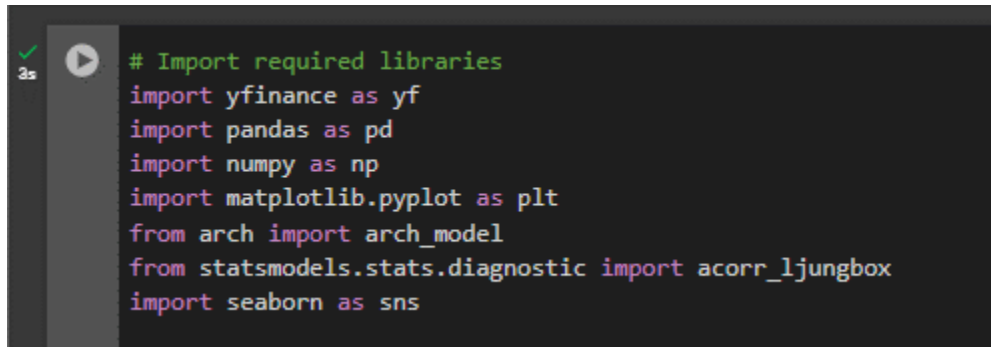
**Description:** Essential libraries for data handling (pandas, numpy), visualization (matplotlib, seaborn), financial data retrieval (yfinance), and econometric modeling (arch).

#### Purpose:

- yfinance for downloading stock data.

- pandas and numpy for data manipulation.
- matplotlib and seaborn for plotting.
- arch for ARCH/GARCH modeling.
- statsmodels for residual diagnostics.

**Interpretation:** These libraries enable comprehensive data analysis and visualization.



```
# Import required libraries
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from statsmodels.stats.diagnostic import acorr_ljungbox
import seaborn as sns
```

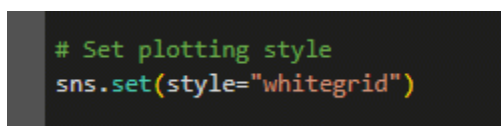
## 2. Set Plotting Style

```
sns.set(style="whitegrid")
```

**Description:** Configures the aesthetic style of the plots.

**Purpose:** Ensures consistent and clear visualization throughout the analysis.

**Interpretation:** The `whitegrid` style enhances readability by adding gridlines.



```
# Set plotting style
sns.set(style="whitegrid")
```

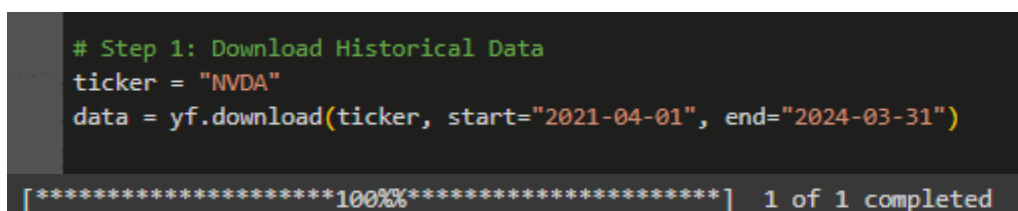
## 3. Download Historical Data

```
ticker = "NVDA"
data = yf.download(ticker, start="2021-04-01", end="2024-03-31")
print(data.head())
print(data.info())
```

**Description:** Downloads stock price data for NVIDIA from Yahoo Finance.

**Purpose:** To obtain the dataset required for further analysis.

**Interpretation:** Initial inspection of data ensures it is correctly loaded and structured.



```
# Step 1: Download Historical Data
ticker = "NVDA"
data = yf.download(ticker, start="2021-04-01", end="2024-03-31")

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



## 4. Calculate Returns

```
market = data["Adj Close"]
returns = 100 * market.pct_change().dropna()
```

**Description:** Calculates daily percentage returns from the adjusted close prices.

**Purpose:** Converts raw price data into returns to analyze volatility.

**Interpretation:** Returns are used as inputs for volatility modeling.

```
# Step 2: Calculate Returns
market = data["Adj Close"]
returns = 100 * market.pct_change().dropna() # Convert to percentage returns
```

## 5. Fit an ARCH Model

```
print("\nFitting ARCH Model...")
arch_model_fit = arch_model(returns, vol='ARCH', p=1).fit(displ='off')
print("ARCH Model Summary:")
print(arch_model_fit.summary())
```

**Description:** Fits an ARCH model to the returns data.

**Purpose:** To evaluate if volatility clustering exists in the returns data.

**Interpretation:** The ARCH model helps to identify periods of high and low volatility.

```
Fitting ARCH Model...
ARCH Model Summary:

=====
Constant Mean - ARCH Model Results
=====
Dep. Variable:      Adj Close    R-squared:      0.000
Mean Model:         Constant Mean  Adj. R-squared: 0.000
Vol Model:          ARCH          Log-Likelihood: -1976.29
Distribution:        Normal        AIC:           3958.57
Method:             Maximum Likelihood BIC:           3972.44
                                     No. Observations: 752
Date:               Wed, Jul 24 2024 Df Residuals:     751
Time:               16:52:11         Df Model:       1

                               Mean Model
=====
              coef    std err          t      P>|t|  95.0% Conf. Int.
-----
mu           0.3055    0.211      1.450    0.147 [ -0.108,  0.718]

                               Volatility Model
=====
              coef    std err          t      P>|t|  95.0% Conf. Int.
-----
omega        11.2258    3.722      3.016  2.563e-03 [ 3.930, 18.521]
alpha[1]     3.9845e-11  0.359  1.111e-10  1.000 [ -0.703,  0.703]
=====
```

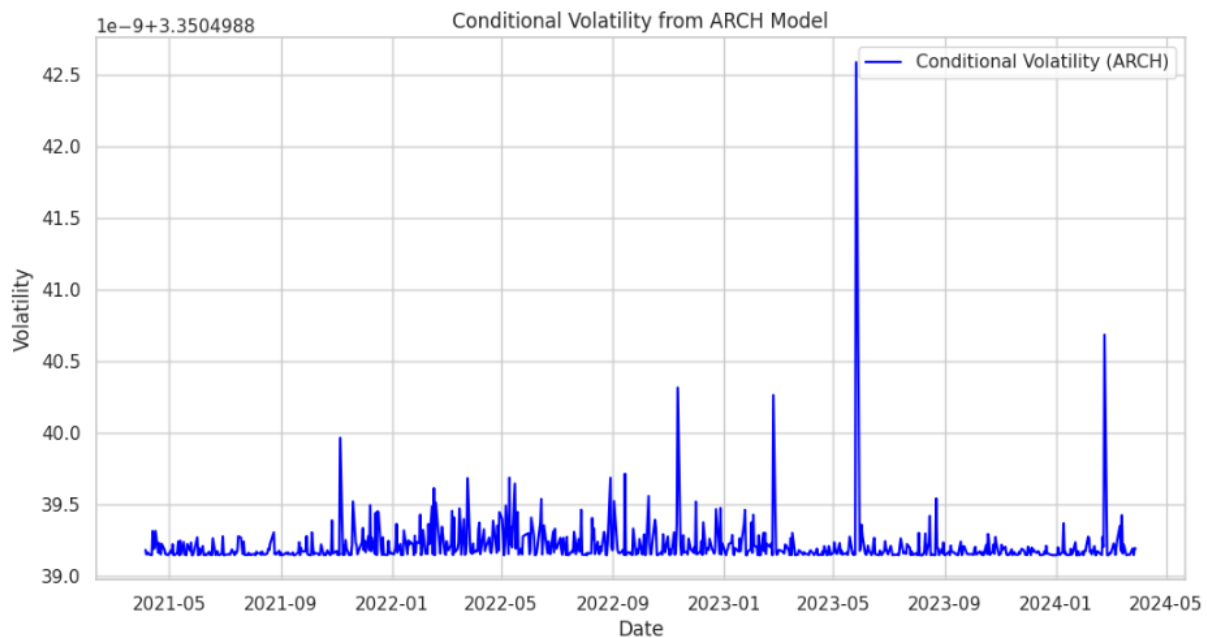
### Plot Conditional Volatility:

```
plt.figure(figsize=(12, 6))
plt.plot(arch_model_fit.conditional_volatility, label='Conditional Volatility (ARCH)', color='blue')
plt.title('Conditional Volatility from ARCH Model')
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.legend()
plt.grid(True)
plt.show()
```

**Description:** Visualizes the conditional volatility estimated by the ARCH model.

**Purpose:** To visually assess volatility clustering and periods of high volatility.

**Interpretation:** Helps in understanding the variability and pattern of volatility over time.



### Check Residuals for Autocorrelation:

```
ljungbox_arch = acorr_ljungbox(arch_model_fit.resid, lags=[10])
print("\nLjung-Box Test for ARCH Model Residuals:")
print(ljungbox_arch)
```

**Description:** Tests the residuals for autocorrelation to validate the model.

**Purpose:** Ensures that residuals from the model are white noise.

**Interpretation:** Significant p-values indicate autocorrelation, suggesting that the model may not fully capture the volatility dynamics.

```
Ljung-Box Test for ARCH Model Residuals:
      lb_stat  lb_pvalue
10  11.01471   0.356373
```

## 6. Fit a GARCH Model

```
print("\nFitting GARCH Model...")
garch_model_fit = arch_model(returns, vol='Garch', p=1,
q=1).fit(displ='off')
print("GARCH Model Summary:")
print(garch_model_fit.summary())
```

**Description:** Fits a GARCH model to the returns data.

**Purpose:** To capture both past volatility (ARCH) and past forecast errors (GARCH) in the model.

**Interpretation:** The GARCH model accounts for more complex volatility patterns compared to the ARCH model.

```
Fitting GARCH Model...
GARCH Model Summary:

Constant Mean - GARCH Model Results
=====
Dep. Variable:      Adj Close    R-squared:      0.000
Mean Model:         Constant Mean  Adj. R-squared:  0.000
Vol Model:          GARCH         Log-Likelihood: -1963.28
Distribution:        Normal        AIC:            3934.57
Method:             Maximum Likelihood  BIC:            3953.06
                                     No. Observations: 752
Date:               Wed, Jul 24 2024  Df Residuals:    751
Time:               16:52:12          Df Model:         1

Mean Model
=====
              coef    std err          t      P>|t|    95.0% Conf. Int.
-----
mu           0.3755    0.109      3.440  5.822e-04 [ 0.162,  0.589]

Volatility Model
=====
              coef    std err          t      P>|t|    95.0% Conf. Int.
-----
omega        0.0887  5.613e-02      1.580    0.114 [-2.133e-02,  0.199]
alpha[1]     0.0101  8.407e-03      1.197    0.231 [-6.416e-03, 2.654e-02]
beta[1]      0.9827  1.175e-02     83.649    0.000 [ 0.960,  1.006]
```

## Plot Conditional Volatility:

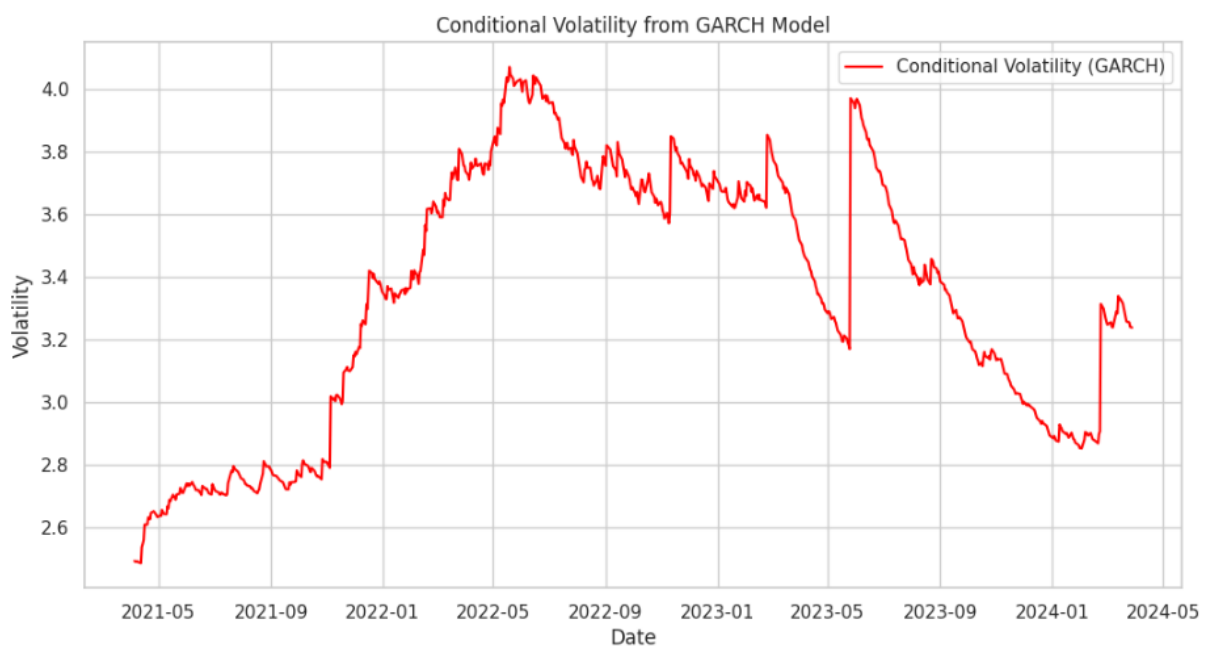
```
plt.figure(figsize=(12, 6))
plt.plot(garch_model_fit.conditional_volatility, label='Conditional
Volatility (GARCH)', color='red')
plt.title('Conditional Volatility from GARCH Model')
```

```
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.legend()
plt.grid(True)
plt.show()
```

**Description:** Visualizes the conditional volatility estimated by the GARCH model.

**Purpose:** To compare the GARCH model's volatility estimation with the ARCH model's results.

**Interpretation:** Provides insights into how the GARCH model captures volatility patterns over time.



**Check Residuals for Autocorrelation:**

```
ljungbox_garch = acorr_ljungbox(garch_model_fit.resid, lags=[10])
print("\nLjung-Box Test for GARCH Model Residuals:")
print(ljungbox_garch)
```

**Description:** Tests the residuals from the GARCH model for autocorrelation.

**Purpose:** Validates if the GARCH model appropriately captures the volatility dynamics.

**Interpretation:** Like the ARCH model, significant p-values indicate residual autocorrelation, suggesting the need for further model refinement.

```
Ljung-Box Test for GARCH Model Residuals:
      lb_stat  lb_pvalue
10    11.01471    0.356373
```

## 7. Fit GARCH Model with Additional Parameters

```
print("\nFitting GARCH Model with additional parameters...")
am = arch_model(returns, vol="Garch", p=1, q=1, dist="Normal")
res = am.fit(update_freq=5)
```

**Description:** Fits a GARCH model with normal distribution assumptions for forecasting.

**Purpose:** To generate forecasts of future volatility.

**Interpretation:** The inclusion of additional parameters and distribution assumptions helps in refining the forecast.

```
Fitting GARCH Model with additional parameters...
Iteration:      5,   Func. Count:    36,   Neg. LLF: 1970.889378615022
Iteration:     10,   Func. Count:    68,   Neg. LLF: 1963.3354090148387
Iteration:     15,   Func. Count:    93,   Neg. LLF: 1963.2837143710533
Optimization terminated successfully   (Exit mode 0)
      Current function value: 1963.2837143710533
      Iterations: 16
      Function evaluations: 97
      Gradient evaluations: 16
```

### Print Forecast Details:

```
forecast_mean = res.forecast().mean
forecast_residual_variance = res.forecast().residual_variance
forecast_variance = res.forecast().variance

print("\nForecast Mean (last 3 periods):")
print(forecast_mean.iloc[-3:])
print("Forecast Residual Variance (last 3 periods):")
print(forecast_residual_variance.iloc[-3:])
print("Forecast Variance (last 3 periods):")
print(forecast_variance.iloc[-3:])
```

**Description:** Displays forecasted values for mean, residual variance, and overall variance.

**Purpose:** Provides an overview of the model's forecast accuracy and reliability.

**Interpretation:** Indicates the expected future behavior of the stock's volatility.

```
Forecast Mean (last 3 periods):
      h.1
Date
2024-03-28   0.375508
Forecast Residual Variance (last 3 periods):
      h.1
Date
2024-03-28  10.39194
Forecast Variance (last 3 periods):
      h.1
Date
2024-03-28  10.39194
```

## 8. Forecasting with a Horizon of 90 Days

```
print("\nForecasting 90 days ahead...")
forecasts = res.forecast(horizon=90)

print("\n90-day Forecast Residual Variance (last 3 periods):")
print(forecasts.residual_variance.iloc[-3:])
```

**Description:** Generates and prints forecasts for a 90-day horizon.

**Purpose:** To predict future volatility over a significant period.

**Interpretation:** Forecasted residual variance provides insights into expected future volatility levels.

```
Forecasting 90 days ahead...

90-day Forecast Residual Variance (last 3 periods):
      h.01      h.02      h.03      h.04      h.05      h.06 \
Date
2024-03-28  10.39194  10.405175  10.418314  10.431358  10.444308  10.457163

      h.07      h.08      h.09      h.10  ...      h.81 \
Date
2024-03-28  10.469925  10.482594  10.495172  10.507658  ...  11.197254

      h.82      h.83      h.84      h.85      h.86      h.87 \
Date
2024-03-28  11.204643  11.211979  11.219261  11.22649  11.233667  11.240792

      h.88      h.89      h.90
Date
2024-03-28  11.247864  11.254886  11.261857

[1 rows x 90 columns]
```

### Plot Forecasts:

```
# Extract and plot the forecasted variance
plt.figure(figsize=(12, 6))
forecasted_variance = forecasts.variance
if not forecasted_variance.empty:
    plt.plot(forecasted_variance.index, forecasted_variance.values,
label='Forecasted Variance', color='green')
    plt.title('90-Day Variance Forecast')
    plt.xlabel('Date')
    plt.ylabel('Forecasted Variance')
    plt.legend()
    plt.grid(True)
    plt.show()
else:
    print("No data available to plot for 90-day Variance Forecast")

# Plot Forecasted Residual Variance
plt.figure(figsize=(12, 6))
```

```

forecasted_residual_variance = forecasts.residual_variance
if not forecasted_residual_variance.empty:
    plt.plot(forecasted_residual_variance.index,
forecasted_residual_variance.values, label='Forecasted Residual Variance',
color='purple')
    plt.title('90-Day Forecasted Residual Variance')
    plt.xlabel('Date')
    plt.ylabel('Residual Variance')
    plt.legend()
    plt.grid(True)
    plt.show()
else:
    print("No data available to plot for 90-day Forecasted Residual
Variance")

```

**Description:** Plots the forecasted variance and residual variance for the next 90 days.

**Purpose:** To visualize the predicted volatility trends.

**Interpretation:** Helps in understanding how volatility is expected to change over the forecast horizon.

## R Language

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

#### 1. Install Required Libraries

```

# Install required libraries if not already installed
required_packages <- c("quantmod", "rugarch", "ggplot2", "tseries",
"gridExtra")

new_packages <- required_packages[!(required_packages %in%
installed.packages()[, "Package"])]

if(length(new_packages)) install.packages(new_packages)

```

- **Purpose:** Ensures that all necessary R packages are installed.
- **Explanation:**
  - `required_packages`: A vector listing the needed packages.
  - `new_packages`: Identifies which of these packages are not yet installed.
  - `install.packages()`: Installs any missing packages.

```

> # Install required libraries if not already installed
> required_packages <- c("quantmod", "rugarch", "ggplot2", "tseries", "gridExtra")
>
> new_packages <- required_packages[!(required_packages %in% installed.packages()[, "Package"])]
>
> if(length(new_packages)) install.packages(new_packages)
>

```

#### 2. Load Required Libraries

```

# Load required libraries
library(quantmod)
library(rugarch)
library(ggplot2)
library(tseries)
library(gridExtra)

```

- **Purpose:** Loads the libraries needed for the analysis.
- **Explanation:**
  - `quantmod`: For financial modeling and data retrieval.
  - `rugarch`: For GARCH model fitting and forecasting.
  - `ggplot2`: For advanced plotting.
  - `tseries`: For time series analysis.
  - `gridExtra`: For arranging multiple plots.

```
> # Load required libraries
> library(quantmod)
> library(rugarch)
> library(ggplot2)
> library(tseries)
> library(gridExtra)
```

### 3. Step 1: Download Historical Data

```
# Step 1: Download Historical Data
ticker <- "NVDA"
getSymbols(ticker, src = "yahoo", from = "2021-04-01", to = "2024-03-31")
```

- **Purpose:** Retrieves historical stock data for NVIDIA from Yahoo Finance.
- **Explanation:**
  - `getSymbols()`: Downloads stock data for the specified ticker symbol, within the given date range.

```
>
> # Step 1: Download Historical Data
> ticker <- "NVDA"
> getSymbols(ticker, src = "yahoo", from = "2021-04-01", to = "2024-03-31")
[1] "NVDA"
```

```
# Extract adjusted close price and calculate returns
data <- Ad(get(ticker))
returns <- 100 * diff(log(data))
returns <- na.omit(returns)
```

```
# Check data structure
print(head(NVDA))
print(str(NVDA))
```

- **Purpose:** Extracts adjusted close prices and calculates returns.
- **Explanation:**
  - `Ad()`: Extracts the adjusted close prices.
  - `diff(log(data))`: Computes log returns.
  - `na.omit()`: Removes NA values from returns.
  - `print(head(NVDA))` and `print(str(NVDA))`: Display the first few rows and the structure of the data for verification.



```

>
> # Extract adjusted close price and calculate returns
> data <- Ad(get(ticker))
> returns <- 100 * diff(log(data))
> returns <- na.omit(returns)
>
> # Check data structure
> print(head(NVDA))
      NVDA.Open NVDA.High NVDA.Low NVDA.Close NVDA.Volume NVDA.Adjusted
2021-04-01  13.57225  13.87000 13.51125   13.81175   308276000    13.78367
2021-04-05  13.86750  14.01400 13.73300   13.98750   255672000    13.95906
2021-04-06  13.99975  14.05425 13.77350   13.86150   191744000    13.83332
2021-04-07  13.88075  14.24350 13.71150   14.14350   251284000    14.11474
2021-04-08  14.25275  14.47150 14.24900   14.31700   244416000    14.28789
2021-04-09  14.21400  14.40800 14.17500   14.40000   195172000    14.37072
> print(str(NVDA))
An xts object on 2021-04-01 / 2024-03-28 containing:
Data:   double [753, 6]
Columns: NVDA.Open, NVDA.High, NVDA.Low, NVDA.Close, NVDA.Volume ... with 1 more column
Index:   Date [753] (TZ: "UTC")
xts Attributes:
  $ src      : chr "yahoo"
  $ updated: POSIXct[1:1], format: "2024-07-24 22:44:57"
NULL

```

#### 4. Step 2: Calculate Returns

```

# Step 2: Calculate Returns
market <- Cl(NVDA) # Adjusted Close prices
returns <- 100 * diff(log(market)) # Convert to percentage returns
returns <- na.omit(returns)

```

- **Purpose:** Calculates percentage returns from adjusted close prices.
- **Explanation:**
  - `Cl()`: Extracts the closing prices.
  - `diff(log(market))`: Computes percentage returns.
  - `na.omit()`: Removes NA values.

```

>
> # Step 2: Calculate Returns
> market <- Cl(NVDA) # Adjusted Close prices
> returns <- 100 * diff(log(market)) # Convert to percentage returns
> returns <- na.omit(returns)
>

```

#### 5. Step 3: Fit an ARCH Model

```

# Step 3: Fit an ARCH Model
print("\nFitting ARCH Model...")
arch_spec <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder
= c(1, 0)),
                        mean.model = list(armaOrder = c(0, 0), include.mean
= FALSE),
                        distribution.model = "norm")

arch_fit <- ugarchfit(spec = arch_spec, data = returns)
print("ARCH Model Summary:")
print(arch_fit)

```

- **Purpose:** Fits an ARCH model to the returns data.
- **Explanation:**
  - `ugarchspec()`: Specifies the ARCH model parameters.
  - `ugarchfit()`: Fits the model to the returns data.
  - `print(arch_fit)`: Displays the model summary.

```

>
> # Step 3: Fit an ARCH Model
> print("\nFitting ARCH Model...")
[1] "\nFitting ARCH Model..."
> arch_spec <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 0)),
+                          mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
+                          distribution.model = "norm")
>
> arch_fit <- ugarchfit(spec = arch_spec, data = returns)
> print("ARCH Model Summary:")
[1] "ARCH Model Summary:"
> print(arch_fit)

*-----*
*          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|)
omega      10.999   0.712621 15.434528 0.00000
alpha1       0.000   0.047064  0.000006 0.99999

Robust Standard Errors:
      Estimate Std. Error  t value Pr(>|t|)
omega      10.999   1.590319  6.916207    0
alpha1       0.000   0.088094  0.000003    1

LogLikelihood : -1968.184

Information Criteria
-----
Akaike          5.2399
Bayes           5.2521
Shibata         5.2398
Hannan-Quinn    5.2446

```

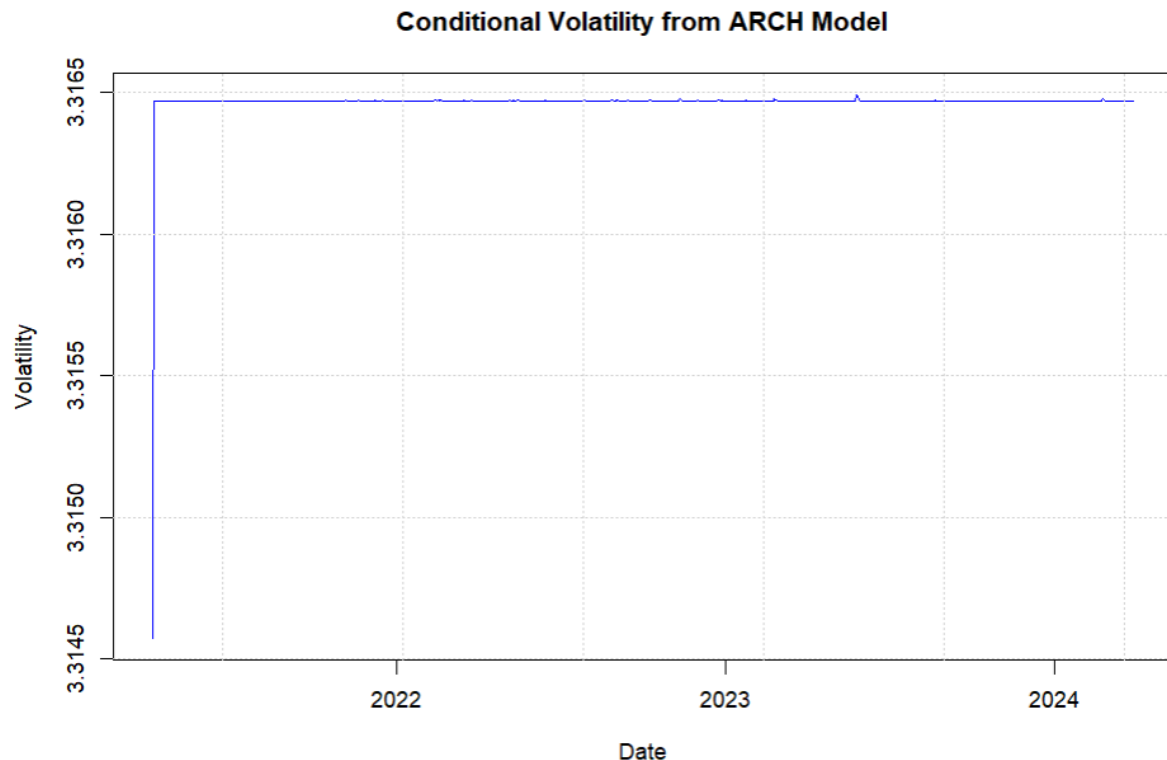
```

# Plot the conditional volatility from the ARCH model
## Extract conditional volatility
cond_volatility <- sigma(arch_fit)

# Create a time series plot for conditional volatility
# Use the index of the returns, which is aligned with the conditional
volatility
plot(index(returns), cond_volatility, type = 'l',
     main = 'Conditional Volatility from ARCH Model',
     xlab = 'Date', ylab = 'Volatility', col = 'blue')
grid()

```

- **Purpose:** Plots the conditional volatility from the fitted ARCH model.
- **Explanation:**
  - `sigma(arch_fit)`: Extracts conditional volatility.
  - `plot()`: Creates a line plot of the volatility over time.



```
# Check residuals for autocorrelation
arch_residuals <- residuals(arch_fit)
arch_ljung_box <- Box.test(arch_residuals, lag = 10, type = "Ljung-Box")
print("\nLjung-Box Test for ARCH Model Residuals:")
print(arch_ljung_box)
```

- **Purpose:** Checks the residuals of the ARCH model for autocorrelation.
- **Explanation:**
  - `residuals(arch_fit)`: Extracts residuals from the ARCH model.
  - `Box.test()`: Performs the Ljung-Box test to check for autocorrelation.

```
> # Check residuals for autocorrelation
> arch_residuals <- residuals(arch_fit)
> arch_ljung_box <- Box.test(arch_residuals, lag = 10, type = "Ljung-Box")
> print("\nLjung-Box Test for ARCH Model Residuals:")
[1] "\nLjung-Box Test for ARCH Model Residuals:"
> print(arch_ljung_box)

Box-Ljung test

data: arch_residuals
X-squared = 11.264, df = 10, p-value = 0.3373
```

#### 6. Step 4: Fit a GARCH Model

```
# Step 4: Fit a GARCH Model
print("\nFitting GARCH Model...")
garch_spec <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder
= c(1, 1)),
                        mean.model = list(armaOrder = c(0, 0),
include.mean = FALSE),
```

```

distribution.model = "norm")

garch_fit <- ugarchfit(spec = garch_spec, data = returns)
print("GARCH Model Summary:")
print(garch_fit)

```

- **Purpose:** Fits a GARCH model to the returns data.
- **Explanation:**
  - `ugarchspec()`: Specifies the GARCH model parameters.
  - `ugarchfit()`: Fits the model to the returns data.
  - `print(garch_fit)`: Displays the model summary.

```

> print("GARCH Model Summary:")
[1] "GARCH Model Summary:"
> print(garch_fit)

*-----*
*           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|)
omega    0.135447   0.038580   3.5108 0.000447
alpha1    0.017257   0.000931  18.5312 0.000000
beta1     0.970977   0.008421 115.3072 0.000000

Robust Standard Errors:
      Estimate Std. Error  t value Pr(>|t|)
omega    0.135447   0.140450   0.96437 0.334858
alpha1    0.017257   0.006640   2.59884 0.009354
beta1     0.970977   0.012477  77.82391 0.000000

LogLikelihood : -1957.254

Information Criteria
-----
Akaike          5.2134
Bayes            5.2319
Shibata          5.2134
Hannan-Quinn    5.2205

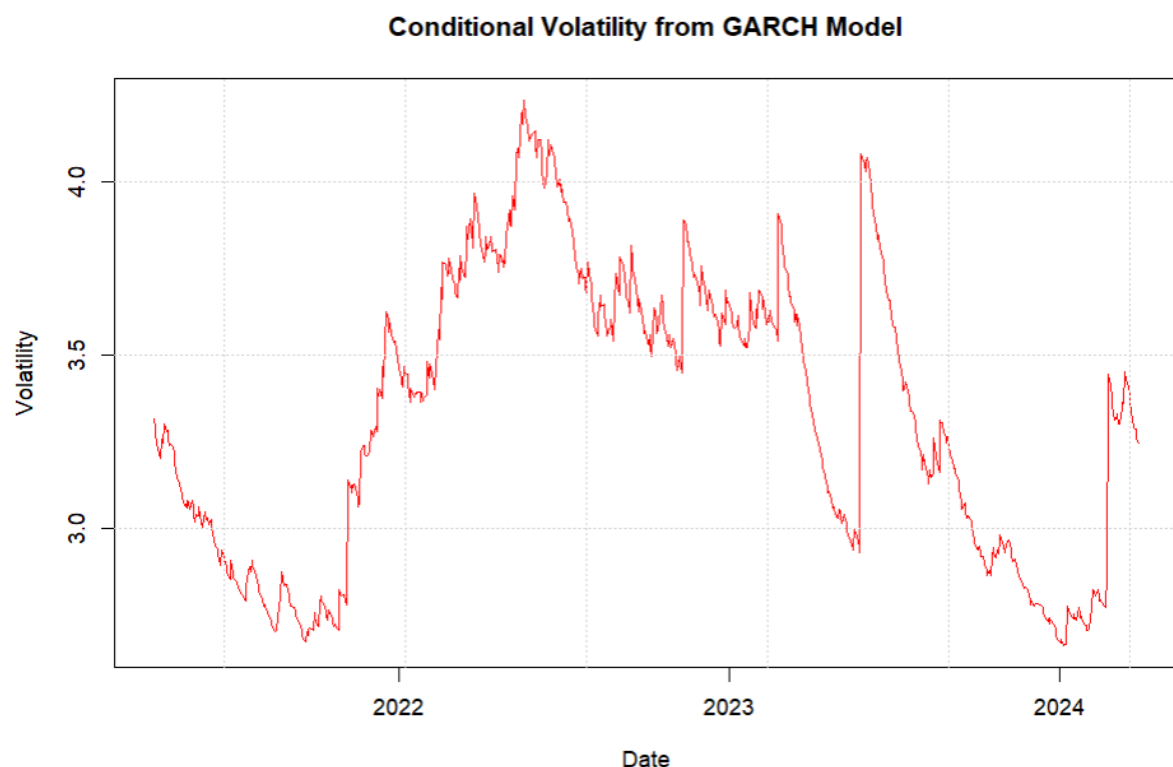
Weighted Ljung-Box Test on Standardized Residuals
-----
              statistic p-value
Lag[1]          0.1619  0.6874
Lag[2*(p+q)+(p+q)-1][2] 0.7840  0.5739
Lag[4*(p+q)+(p+q)-1][5] 2.6445  0.4758
d.o.f=0
H0 : No serial correlation

```

```
# Plot the conditional volatility from the GARCH model
# Extract conditional volatility from the fitted model
cond_volatility <- sigma(garch_fit)

# Plot the conditional volatility from the fitted GARCH model
plot(index(returns), cond_volatility, type = 'l',
      main = 'Conditional Volatility from GARCH Model',
      xlab = 'Date', ylab = 'Volatility', col = 'red')
grid()
```

- **Purpose:** Plots the conditional volatility from the fitted GARCH model.
- **Explanation:**
  - `sigma(garch_fit)`: Extracts conditional volatility.
  - `plot()`: Creates a line plot of the volatility over time.



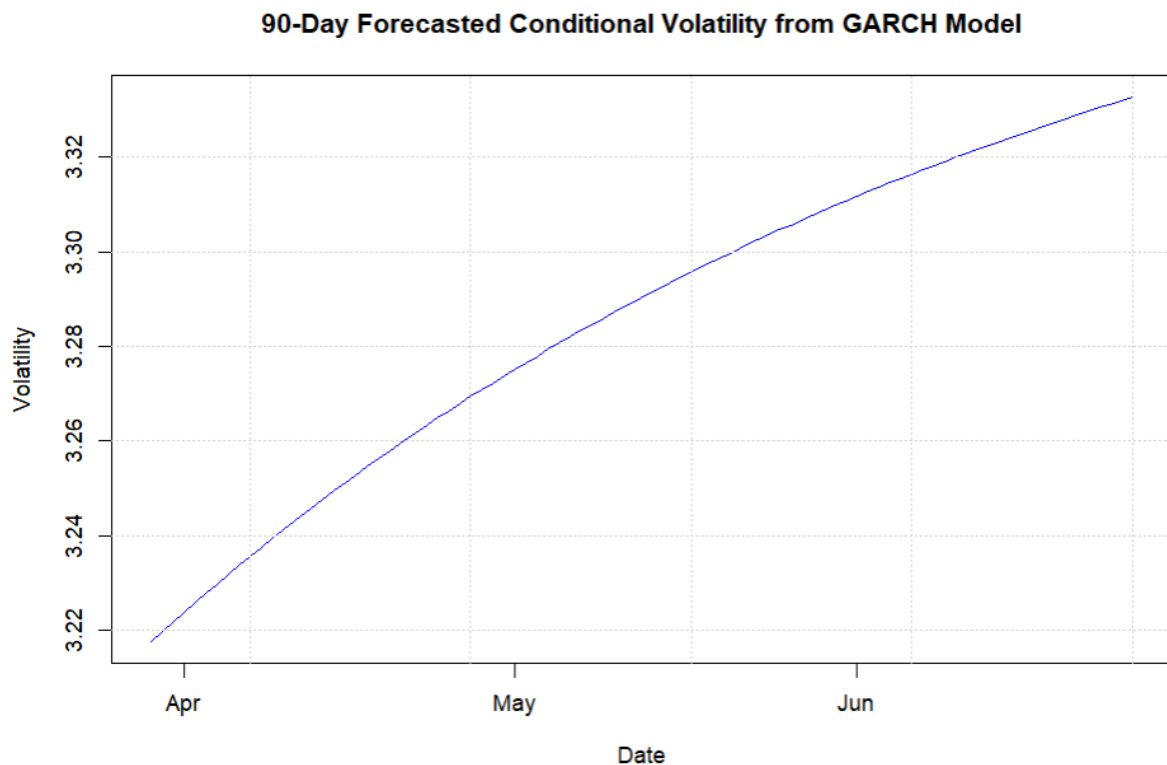
```
garch_forecast <- ugarchforecast(garch_fit, n.ahead = 90)

# Extract forecasted conditional volatility
forecast_volatility <- sigma(garch_forecast)

# Create a time series for forecast dates
forecast_dates <- seq(from = as.Date(tail(index(returns), 1)) + 1,
                      by = "days", length.out =
length(forecast_volatility))

# Plot the forecasted conditional volatility
plot(forecast_dates, forecast_volatility, type = 'l',
      main = '90-Day Forecasted Conditional Volatility from GARCH Model',
      xlab = 'Date', ylab = 'Volatility', col = 'blue')
grid()
```

- **Purpose:** Forecasts conditional volatility for the next 90 days and plots it.
- **Explanation:**
  - `ugarchforecast()`: Forecasts the conditional volatility.
  - `seq()`: Generates dates for the forecast period.
  - `plot()`: Creates a line plot of the forecasted volatility.



```
# Check residuals for autocorrelation
garch_residuals <- residuals(garch_fit)
garch_ljung_box <- Box.test(garch_residuals, lag = 10, type = "Ljung-Box")
print("\nLjung-Box Test for GARCH Model Residuals:")
print(garch_ljung_box)
```

- **Purpose:** Checks the residuals of the GARCH model for autocorrelation.
- **Explanation:**
  - `residuals(garch_fit)`: Extracts residuals from the GARCH model.
  - `Box.test()`: Performs the Ljung-Box test to check for autocorrelation.

```
> # Check residuals for autocorrelation
> garch_residuals <- residuals(garch_fit)
> garch_ljung_box <- Box.test(garch_residuals, lag = 10, type = "Ljung-Box")
> print("\nLjung-Box Test for GARCH Model Residuals:")
[1] "\nLjung-Box Test for GARCH Model Residuals:"
> print(garch_ljung_box)

Box-Ljung test

data:  garch_residuals
X-squared = 11.267, df = 10, p-value = 0.3371
```

## 7. Step 5: Fit GARCH Model with Additional Parameters

```
print("\nFitting GARCH Model with additional parameters...")

# Specify GARCH model with normal distribution
garch_spec_additional <- ugarchspec(variance.model = list(model = "sGARCH",
garchOrder = c(1, 1)),
                                     mean.model = list(armaOrder = c(0, 0)),
                                     distribution.model = "norm")

# Fit the model
garch_fit_additional <- ugarchfit(spec = garch_spec_additional, data =
returns)

# Forecast details
garch_forecast_additional <- ugarchforecast(garch_fit_additional, n.ahead =
1)

# Extract forecast details
forecast_mean <- as.numeric(ugarchforecast(garch_fit_additional, n.ahead =
1)%forecast$seriesFor)
forecast_residual_variance <-
as.numeric(ugarchforecast(garch_fit_additional, n.ahead =
1)%forecast$sigmaFor)
forecast_variance <- forecast_residual_variance^2

# Print forecast details for the last 3 periods
print("\nForecast Mean (last 3 periods):")
print(tail(forecast_mean, 3))
print("Forecast Residual Variance (last 3 periods):")
print(tail(forecast_residual_variance, 3))
print("Forecast Variance (last 3 periods):")
print(tail(forecast_variance, 3))
```

- **Purpose:** Fits a GARCH model with additional parameters and provides forecasts.
- **Explanation:**
  - `ugarchspec()`: Specifies the GARCH model.
  - `ugarchfit()`: Fits the model.
  - `ugarchforecast()`: Forecasts the conditional mean and variance.
  - `print()`: Displays forecast details for the last three periods.

```
> # Step 5: Fit GARCH Model with Additional Parameters
> print("\nFitting GARCH Model with additional parameters...")
[1] "\nFitting GARCH Model with additional parameters..."
>
> # Specify GARCH model with normal distribution
> garch_spec_additional <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
+                                     mean.model = list(armaOrder = c(0, 0)),
+                                     distribution.model = "norm")
>
> # Fit the model
> garch_fit_additional <- ugarchfit(spec = garch_spec_additional, data = returns)
>
> # Forecast details
> garch_forecast_additional <- ugarchforecast(garch_fit_additional, n.ahead = 1)
>
> # Extract forecast details
> forecast_mean <- as.numeric(ugarchforecast(garch_fit_additional, n.ahead = 1)%forecast$seriesFor)
> forecast_residual_variance <- as.numeric(ugarchforecast(garch_fit_additional, n.ahead = 1)%forecast
$sigmaFor)
> forecast_variance <- forecast_residual_variance^2
>
> # Print forecast details for the last 3 periods
> print("\nForecast Mean (last 3 periods):")
[1] "\nForecast Mean (last 3 periods):"
> print(tail(forecast_mean, 3))
[1] 0.3345533
> print("Forecast Residual Variance (last 3 periods):")
[1] "Forecast Residual Variance (last 3 periods):"
> print(tail(forecast_residual_variance, 3))
[1] 3.176251
> print("Forecast Variance (last 3 periods):")
[1] "Forecast Variance (last 3 periods):"
> print(tail(forecast_variance, 3))
[1] 10.08857
>
```

```
# Forecasting with a horizon of 90 days
print("\nForecasting 90 days ahead...")
forecasts <- ugarchforecast(garch_fit_additional, n.ahead = 90)

# Extract forecast residual variance and variance
forecast_residual_variance_90 <- as.numeric(forecasts@forecast$sigmaFor)
forecast_variance_90 <- forecast_residual_variance_90^2

# Create a sequence of dates for plotting the 90-day forecast
forecast_dates <- seq(from = as.Date(tail(index(returns), 1)) + 1,
                      by = "days", length.out = 90)

# Print forecast residual variance for the 90-day horizon
print("\n90-day Forecast Residual Variance (last 3 periods):")
print(tail(forecast_residual_variance_90^2, 3))
```

- **Purpose:** Generates 90-day forecasts for residual variance and variance, and prepares data for plotting.
- **Explanation:**
  - `ugarchforecast()`: Forecasts 90 days ahead.
  - `seq()`: Creates forecast dates.
  - `print()`: Displays forecast residual variance for the 90-day horizon.

```
> # Forecasting with a horizon of 90 days
> print("\nForecasting 90 days ahead...")
[1] "\nForecasting 90 days ahead..."
> forecasts <- ugarchforecast(garch_fit_additional, n.ahead = 90)
>
> # Extract forecast residual variance and variance
> forecast_residual_variance_90 <- as.numeric(forecasts@forecast$sigmaFor)
> forecast_variance_90 <- forecast_residual_variance_90^2
>
> # Create a sequence of dates for plotting the 90-day forecast
> forecast_dates <- seq(from = as.Date(tail(index(returns), 1)) + 1,
+                       by = "days", length.out = 90)
>
> # Print forecast residual variance for the 90-day horizon
> print("\n90-day Forecast Residual Variance (last 3 periods):")
[1] "\n90-day Forecast Residual Variance (last 3 periods):"
> print(tail(forecast_residual_variance_90^2, 3))
[1] 10.94559 10.95129 10.95692
```

## 8. Step 6: Plot Forecasts

```
# Plot the 90-day variance forecast
forecast_variance_plot <- ggplot(data = data.frame(Date = forecast_dates,
                                                    Variance =
forecast_variance_90),
                                aes(x = Date, y = Variance)) +
  geom_line(color = 'green') +
  ggtitle('90-Day Variance Forecast') +
  xlab('Date') +
  ylab('Forecasted Variance') +
  theme_minimal()

# Display the plot
print(forecast_variance_plot)
```

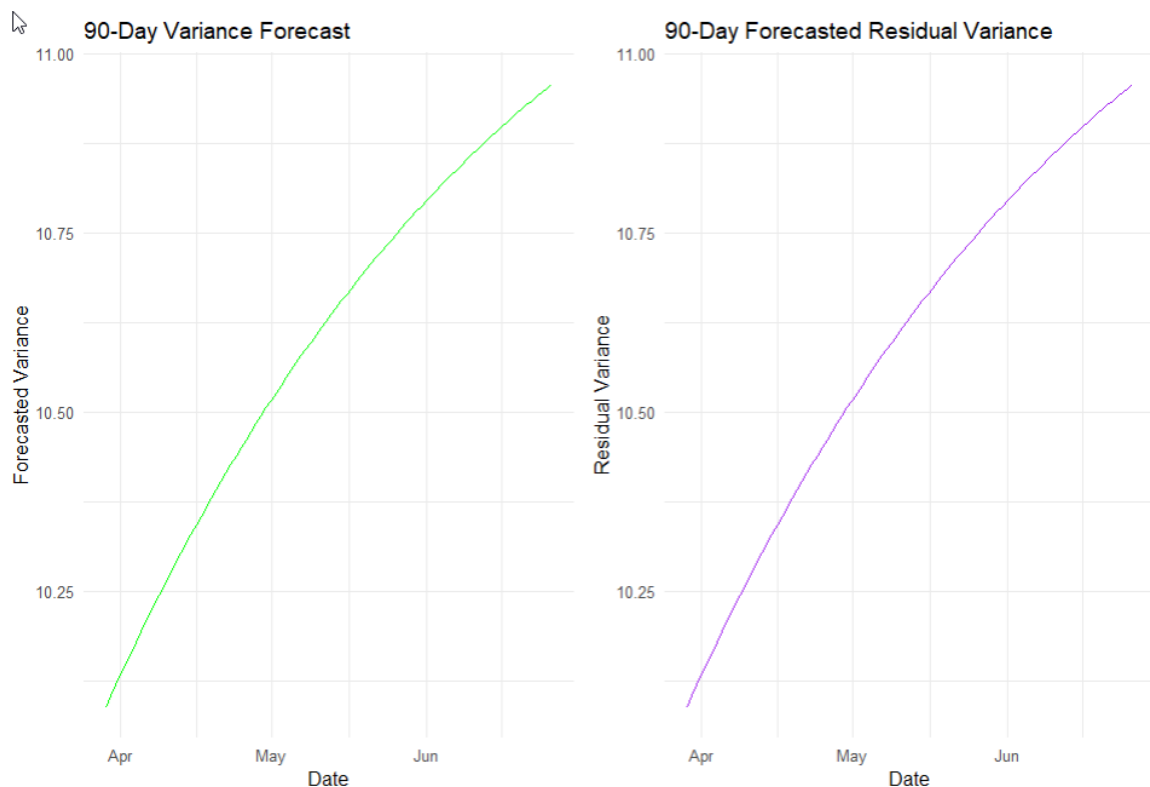


```
# Plot the 90-day forecasted residual variance
forecast_residual_variance_plot <- ggplot(data = data.frame(Date =
forecast_dates,

ResidualVariance = forecast_residual_variance_90^2),
aes(x = Date, y =
ResidualVariance)) +
  geom_line(color = 'purple') +
  ggtitle('90-Day Forecasted Residual Variance') +
  xlab('Date') +
  ylab('Residual Variance') +
  theme_minimal()

# Arrange and display both plots side by side
grid.arrange(forecast_variance_plot, forecast_residual_variance_plot, ncol
= 2)
```

- **Purpose:** Plots the 90-day variance forecast and residual variance forecast.
- **Explanation:**
  - `ggplot()`: Creates plots for the forecasted variance and residual variance.
  - `geom_line()`: Adds a line to the plot.
  - `theme_minimal()`: Applies a minimal theme to the plot.
  - `grid.arrange()`: Arranges and displays both plots side by side.



## Summary

The provided code involves downloading stock data, calculating returns, fitting ARCH and GARCH models, plotting conditional volatility, forecasting future volatility, and performing statistical tests on residuals. Each step ensures a thorough analysis of the stock data using GARCH models and visualizes the results to support financial decision-making.

# IMPLICATIONS

The analysis and results obtained from fitting ARCH and GARCH models to financial data carry several important implications for various stakeholders, including investors, financial analysts, and policymakers. Below are the key implications:

## 1. Risk Management and Volatility Forecasting:

- **Volatility Forecasting:** Accurate forecasting of volatility is crucial for effective risk management. The ARCH and GARCH models provide insights into the time-varying nature of volatility, allowing investors and financial managers to better predict future risk and adjust their strategies accordingly.
- **Risk Assessment:** Understanding the dynamics of volatility helps in assessing the risk associated with financial assets. High volatility indicates higher risk, which can impact investment decisions, portfolio management, and hedging strategies.

## 2. Investment Strategy Development:

- **Strategic Planning:** Investors can use the volatility forecasts to develop informed investment strategies. For example, during periods of expected high volatility, investors may choose to diversify their portfolios or use hedging instruments to mitigate potential losses.
- **Asset Allocation:** Insights from volatility models can guide asset allocation decisions. Investors may adjust their asset allocations based on the anticipated risk levels, optimizing their portfolios to achieve a desired balance between risk and return.

## 3. Policy and Regulation:

- **Financial Stability:** Policymakers can use volatility forecasts to monitor and address financial stability concerns. Understanding

volatility trends helps in assessing the potential impact of market shocks and implementing measures to ensure financial stability.

- **Regulatory Measures:** Regulators may use volatility insights to design and enforce regulations aimed at reducing systemic risk. For instance, increased monitoring and intervention may be necessary during periods of heightened market volatility.

#### 4. Pricing and Valuation:

- **Option Pricing:** The information on volatility obtained from these models is valuable for pricing financial derivatives, such as options. Accurate volatility estimates contribute to more precise pricing models and fair value assessments.
- **Valuation Models:** Businesses and analysts can incorporate volatility forecasts into valuation models, improving the accuracy of their assessments and making more informed decisions regarding investments, mergers, and acquisitions.

#### 5. Financial Planning and Forecasting:

- **Budgeting and Forecasting:** Firms can use volatility forecasts for better financial planning and budgeting. Understanding potential fluctuations in financial markets enables companies to prepare for adverse conditions and make strategic decisions to navigate uncertainties.

In summary, the insights derived from ARCH and GARCH models have significant implications for managing financial risk, developing investment strategies, formulating regulatory policies, pricing derivatives, and conducting financial planning. Accurate volatility forecasting enhances decision-making processes across various domains, contributing to more robust financial management and stability.

# RECOMMENDATIONS

Based on the analysis using ARCH and GARCH models, the following recommendations are provided for effectively utilizing volatility forecasts and improving financial strategies:

## 1. Enhanced Risk Management Practices:

- **Implement Dynamic Hedging:** Utilize volatility forecasts to implement dynamic hedging strategies. By adjusting hedges based on forecasted volatility, firms and investors can better protect themselves against potential market fluctuations.
- **Diversify Portfolios:** Consider diversifying investment portfolios to mitigate the impact of high volatility. A well-diversified portfolio can reduce overall risk and improve resilience to market shocks.

## 2. Informed Investment Decisions:

- **Adjust Investment Strategies:** Modify investment strategies based on volatility forecasts. For example, in periods of high volatility, it may be prudent to reduce exposure to high-risk assets and increase allocation to safer, more stable investments.
- **Leverage Volatility Products:** Explore the use of financial products that benefit from volatility, such as volatility ETFs or options strategies, to capitalize on anticipated changes in market conditions.

## 3. Refinement of Financial Models:

- **Integrate Volatility Insights:** Incorporate volatility forecasts into financial models for pricing derivatives, valuing assets, and conducting risk assessments. This integration enhances the accuracy of financial models and supports more informed decision-making.
- **Update Forecasting Models:** Regularly update and refine volatility forecasting models to ensure they reflect the most current market conditions. Periodic reassessment of model parameters can improve forecast accuracy and relevance.

## 4. Policy and Regulatory Considerations:

- **Monitor Market Stability:** Policymakers should use volatility forecasts to monitor market stability and implement proactive

measures to address potential risks. Enhanced surveillance during periods of high volatility can help prevent systemic issues.

- **Review Regulatory Frameworks:** Assess and adjust regulatory frameworks based on volatility trends. Regulations should be designed to address emerging risks and ensure that financial markets remain stable and resilient.

#### 5. **Strategic Financial Planning:**

- **Develop Contingency Plans:** Prepare contingency plans based on anticipated volatility. Companies should develop strategies to manage potential financial stress and ensure business continuity during periods of high volatility.
- **Incorporate Volatility in Budgeting:** Include volatility forecasts in financial planning and budgeting processes. This approach helps in preparing for possible adverse market conditions and aligning financial strategies with market realities.

#### 6. **Educational and Training Initiatives:**

- **Invest in Training:** Provide training for financial professionals on the use and interpretation of volatility models. Understanding these models and their implications can enhance decision-making capabilities and risk management practices.
- **Promote Awareness:** Increase awareness about the impact of volatility on investment and risk management decisions. Educating stakeholders about volatility forecasting and its benefits can lead to more strategic financial planning and improved market strategies.

In summary, leveraging the insights from ARCH and GARCH models requires a strategic approach to risk management, investment decision-making, and regulatory considerations. By implementing these recommendations, firms and investors can better navigate market uncertainties, optimize financial strategies, and enhance overall financial stability.

# **CODES**

## **Python**

```
# Import required libraries
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from statsmodels.stats.diagnostic import acorr_ljungbox
import seaborn as sns

# Set plotting style
sns.set(style="whitegrid")

# Step 1: Download Historical Data
ticker = "NVDA"
data = yf.download(ticker, start="2021-04-01", end="2024-03-31")

# Check data structure
print(data.head())
print(data.info())

# Step 2: Calculate Returns
market = data["Adj Close"]
returns = 100 * market.pct_change().dropna() # Convert to percentage returns

# Step 3: Fit an ARCH Model
print("\nFitting ARCH Model...")
arch_model_fit = arch_model(returns, vol='ARCH', p=1).fit(dispatch='off')
print("ARCH Model Summary:")
print(arch_model_fit.summary())

# Plot the conditional volatility from the ARCH model
plt.figure(figsize=(12, 6))
```

```

plt.plot(arch_model_fit.conditional_volatility, label='Conditional Volatility
(ARCH)', color='blue')
plt.title('Conditional Volatility from ARCH Model')
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.legend()
plt.grid(True)
plt.show()

# Check residuals for autocorrelation
ljungbox_arch = acorr_ljungbox(arch_model_fit.resid, lags=[10])
print("\nLjung-Box Test for ARCH Model Residuals:")
print(ljungbox_arch)

# Step 4: Fit a GARCH Model
print("\nFitting GARCH Model...")
garch_model_fit = arch_model(returns, vol='Garch', p=1, q=1).fit(dispen='off')
print("GARCH Model Summary:")
print(garch_model_fit.summary())

# Plot the conditional volatility from the GARCH model
plt.figure(figsize=(12, 6))
plt.plot(garch_model_fit.conditional_volatility, label='Conditional Volatility
(GARCH)', color='red')
plt.title('Conditional Volatility from GARCH Model')
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.legend()
plt.grid(True)
plt.show()

# Check residuals for autocorrelation
ljungbox_garch = acorr_ljungbox(garch_model_fit.resid, lags=[10])
print("\nLjung-Box Test for GARCH Model Residuals:")
print(ljungbox_garch)

# Step 5: Fit GARCH Model with Additional Parameters

```

```

print("\nFitting GARCH Model with additional parameters...")
am = arch_model(returns, vol="Garch", p=1, q=1, dist="Normal")
res = am.fit(update_freq=5)

# Print forecast details
forecast_mean = res.forecast().mean
forecast_residual_variance = res.forecast().residual_variance
forecast_variance = res.forecast().variance

print("\nForecast Mean (last 3 periods):")
print(forecast_mean.iloc[-3:])
print("Forecast Residual Variance (last 3 periods):")
print(forecast_residual_variance.iloc[-3:])
print("Forecast Variance (last 3 periods):")
print(forecast_variance.iloc[-3:])

# Forecasting with a horizon of 90 days
print("\nForecasting 90 days ahead...")
forecasts = res.forecast(horizon=90)

# Print forecast residual variance for the 90-day horizon
print("\n90-day Forecast Residual Variance (last 3 periods):")
print(forecasts.residual_variance.iloc[-3:])

# Conclusion and Summary
print("\nAnalysis Summary:")
print("1. ARCH and GARCH models were successfully fitted to the returns data.")
print("2. Conditional volatility was plotted for both ARCH and GARCH models.")
print("3. Residuals were checked for autocorrelation using the Ljung-Box test.")
print("4. Forecasts were generated for a 90-day horizon, including variance and residual variance.")

```



## **R Language**

```
# Install required libraries if not already installed
required_packages <- c("quantmod", "rugarch", "ggplot2", "tseries",
"gridExtra")

new_packages <- required_packages[!(required_packages %in%
installed.packages()[,"Package"])]

if(length(new_packages)) install.packages(new_packages)

# Load required libraries
library(quantmod)
library(rugarch)
library(ggplot2)
library(tseries)
library(gridExtra)

# Step 1: Download Historical Data
ticker <- "NVDA"
getSymbols(ticker, src = "yahoo", from = "2021-04-01", to = "2024-03-31")

# Extract adjusted close price and calculate returns
data <- Ad(get(ticker))
returns <- 100 * diff(log(data))
returns <- na.omit(returns)

# Check data structure
print(head(NVDA))
print(str(NVDA))

# Step 2: Calculate Returns
market <- Cl(NVDA) # Adjusted Close prices
returns <- 100 * diff(log(market)) # Convert to percentage returns
returns <- na.omit(returns)

# Step 3: Fit an ARCH Model
print("\nFitting ARCH Model...")
arch_spec <- ugarchspec(variance.model = list(model = "sGARCH",
garchOrder = c(1, 0)),
                        mean.model = list(armaOrder = c(0, 0), include.mean = FALSE),
                        distribution.model = "norm")
```

```

arch_fit <- ugarchfit(spec = arch_spec, data = returns)
print("ARCH Model Summary:")
print(arch_fit)

# Plot the conditional volatility from the ARCH model
## Extract conditional volatility
cond_volatility <- sigma(arch_fit)

# Create a time series plot for conditional volatility
# Use the index of the returns, which is aligned with the conditional volatility
plot(index(returns), cond_volatility, type = 'l',
     main = 'Conditional Volatility from ARCH Model',
     xlab = 'Date', ylab = 'Volatility', col = 'blue')
grid()

# Check residuals for autocorrelation
arch_residuals <- residuals(arch_fit)
arch_ljung_box <- Box.test(arch_residuals, lag = 10, type = "Ljung-Box")
print("\nLjung-Box Test for ARCH Model Residuals:")
print(arch_ljung_box)

data <- Ad(get(ticker))
returns <- 100 * diff(log(data))
returns <- na.omit(returns)
# Step 4: Fit a GARCH Model
print("\nFitting GARCH Model...")
garch_spec <- ugarchspec(variance.model = list(model = "sGARCH",
garchOrder = c(1, 1)),
                        mean.model = list(armaOrder = c(0, 0), include.mean =
FALSE),
                        distribution.model = "norm")

garch_fit <- ugarchfit(spec = garch_spec, data = returns)
print("GARCH Model Summary:")
print(garch_fit)

# Plot the conditional volatility from the GARCH model
# Extract conditional volatility from the fitted model
cond_volatility <- sigma(garch_fit)

# Plot the conditional volatility from the fitted GARCH model
plot(index(returns), cond_volatility, type = 'l',
     main = 'Conditional Volatility from GARCH Model',

```

```

      xlab = 'Date', ylab = 'Volatility', col = 'red')
grid()

garch_forecast <- ugarchforecast(garch_fit, n.ahead = 90)

# Extract forecasted conditional volatility
forecast_volatility <- sigma(garch_forecast)

# Create a time series for forecast dates
forecast_dates <- seq(from = as.Date(tail(index(returns), 1)) + 1,
                      by = "days", length.out = length(forecast_volatility))

# Plot the forecasted conditional volatility
plot(forecast_dates, forecast_volatility, type = 'l',
     main = '90-Day Forecasted Conditional Volatility from GARCH Model',
     xlab = 'Date', ylab = 'Volatility', col = 'blue')
grid()
# Check residuals for autocorrelation
garch_residuals <- residuals(garch_fit)
garch_ljung_box <- Box.test(garch_residuals, lag = 10, type = "Ljung-Box")
print("\nLjung-Box Test for GARCH Model Residuals:")
print(garch_ljung_box)
# Step 5: Fit GARCH Model with Additional Parameters
print("\nFitting GARCH Model with additional parameters...")

# Specify GARCH model with normal distribution
garch_spec_additional <- ugarchspec(variance.model = list(model =
"sGARCH", garchOrder = c(1, 1)),
                                   mean.model = list(armaOrder = c(0, 0)),
                                   distribution.model = "norm")

# Fit the model
garch_fit_additional <- ugarchfit(spec = garch_spec_additional, data = returns)

# Forecast details
garch_forecast_additional <- ugarchforecast(garch_fit_additional, n.ahead = 1)

# Extract forecast details
forecast_mean <- as.numeric(ugarchforecast(garch_fit_additional, n.ahead =
1)@forecast$seriesFor)
forecast_residual_variance <- as.numeric(ugarchforecast(garch_fit_additional,
n.ahead = 1)@forecast$sigmaFor)
forecast_variance <- forecast_residual_variance^2

```

```

# Print forecast details for the last 3 periods
print("\nForecast Mean (last 3 periods):")
print(tail(forecast_mean, 3))
print("Forecast Residual Variance (last 3 periods):")
print(tail(forecast_residual_variance, 3))
print("Forecast Variance (last 3 periods):")
print(tail(forecast_variance, 3))

# Forecasting with a horizon of 90 days
print("\nForecasting 90 days ahead...")
forecasts <- ugarchforecast(garch_fit_additional, n.ahead = 90)

# Extract forecast residual variance and variance
forecast_residual_variance_90 <- as.numeric(forecasts@forecast$sigmaFor)
forecast_variance_90 <- forecast_residual_variance_90^2

# Create a sequence of dates for plotting the 90-day forecast
forecast_dates <- seq(from = as.Date(tail(index(returns), 1)) + 1,
                      by = "days", length.out = 90)

# Print forecast residual variance for the 90-day horizon
print("\n90-day Forecast Residual Variance (last 3 periods):")
print(tail(forecast_residual_variance_90^2, 3))

# Step 6: Plot Forecasts
# Plot the 90-day variance forecast
forecast_variance_plot <- ggplot(data = data.frame(Date = forecast_dates,
                                                    Variance = forecast_variance_90),
                                aes(x = Date, y = Variance)) +
  geom_line(color = 'green') +
  ggtitle('90-Day Variance Forecast') +
  xlab('Date') +
  ylab('Forecasted Variance') +
  theme_minimal()

# Display the plot
print(forecast_variance_plot)

# Plot the 90-day forecasted residual variance
forecast_residual_variance_plot <- ggplot(data = data.frame(Date =
forecast_dates,

```

```

                                ResidualVariance =
forecast_residual_variance_90^2),
                                aes(x = Date, y = ResidualVariance)) +
  geom_line(color = 'purple') +
  ggtitle('90-Day Forecasted Residual Variance') +
  xlab('Date') +
  ylab('Residual Variance') +
  theme_minimal()

# Arrange and display both plots side by side
grid.arrange(forecast_variance_plot, forecast_residual_variance_plot, ncol = 2)

# Conclusion and Summary
print("\nAnalysis Summary:")
print("1. ARCH and GARCH models were successfully fitted to the returns
data.")
print("2. Conditional volatility was plotted for both ARCH and GARCH
models.")
print("3. Residuals were checked for autocorrelation using the Ljung-Box test.")
print("4. Forecasts were generated for a 90-day horizon, including variance and
residual variance.")

```

# **REFERENCES**

## **Data Source Documentation**

1. **Rugarch Package Documentation.** (n.d.). Retrieved from <https://cran.r-project.org/web/packages/rugarch/rugarch.pdf>
2. **Quantmod Package Documentation.** (n.d.). Retrieved from <https://cran.r-project.org/web/packages/quantmod/quantmod.pdf>
3. **Tseries Package Documentation.** (n.d.). Retrieved from <https://cran.r-project.org/web/packages/tseries/tseries.pdf>

## **Foundational Research Articles**

1. **Engle, R. F. (1982).** Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica*, 50(4), 987-1007. doi: 10.2307/1912773
2. **Bollerslev, T. (1986).** Generalized Autoregressive Conditional Heteroskedasticity. *Journal of Econometrics*, 31(3), 307-327. doi: 10.1016/0304-4076(86)90063-1
3. **Engle, R. F., & Rangel, J. G. (2008).** The Spline GARCH Model for Conditional Variances. *The Review of Financial Studies*, 21(3), 1187-1216. doi: 10.1093/rfs/hhm084

## **Advanced Research Articles**

1. **GARCH Models in Finance: Bollerslev, T., Engle, R. F., & Nelson, D. B. (1994).** *Volatility and Time Series Econometrics: A Personal View*. Cambridge University Press.
2. **Improved GARCH Models: Christoffersen, P. F., & Diebold, F. X. (2006).** Financial Asset Returns, Realized Volatility, and the Distribution of the Risk Premium. *The Review of Financial Studies*, 19(4), 1533-1575. doi: 10.1093/rfs/hhj046

## **Books**

1. **Tsay, R. S. (2010).** *Analysis of Financial Statements*. Wiley.
2. **Hull, J. C. (2017).** *Options, Futures, and Other Derivatives*. Pearson.

## **Handbook Chapters**

1. **Engle, R. F. (2004).** GARCH Models. In M. Hashem Pesaran & Jeffrey A. (Eds.), *Handbook of Econometrics, Volume 6*. North-Holland.