Time Series Forecasting Asian Paints

Code

Lakshay Dhingra

Time-Series Forecasting of Asian Paints Stock Prices

1. Initial Setup

1.1. Clearing Previously Loaded Objects from Memory

First, I wanted a clean slate to start running our code, and that is why I removed all currently loaded objects from R environment.
ls() function gives a list of current objects loaded in r environment, and rm() function removes given object from memory.

Hide

```
rm(list=ls())
```

1.2. Installing and Loading Packages

Now, we are installing and loading all the required packages to do our analysis, plotting graphs, and creating models.
“install.packages()” installs all the packages required, and then lapply() function loads them one by one.

Hide

```
# Required Packages
packages = c('quantmod','tseries', 'forecast','FinTS', 'rugarch', 'dplyr', 'ggplot2', 'zoo',
'lmtest', 'knitr', 'kableExtra')

# Installing the Required Packages
# install.packages(packages, dependencies = TRUE)

# Loading the Required Packages
lapply(packages, require, character.only = TRUE)
```

```
Loading required package: quantmod
Warning: package ‘quantmod’ was built under R version 4.3.2Loading required package: xts
Warning: package ‘xts’ was built under R version 4.3.2Loading required package: zoo
Warning: package ‘zoo’ was built under R version 4.3.2
Attaching package: ‘zoo’

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

    as.Date, as.Date.numeric

Loading required package: TTR
Warning: package ‘TTR’ was built under R version 4.3.2Registered S3 method overwritten by 'qu
antmod':
  method            from
  as.zoo.data.frame zoo
Loading required package: tseries
Warning: package ‘tseries’ was built under R version 4.3.2
    ‘tseries’ version: 0.10-55

    ‘tseries’ is a package for time series analysis and computational finance.

    See ‘library(help="tseries")’ for details.

Loading required package: forecast
Warning: package ‘forecast’ was built under R version 4.3.2Loading required package: FinTS
Warning: package ‘FinTS’ was built under R version 4.3.2
Attaching package: ‘FinTS’

The following object is masked from ‘package:forecast’:

    Acf

Loading required package: rugarch
Warning: package ‘rugarch’ was built under R version 4.3.2Loading required package: parallel

Attaching package: ‘rugarch’

The following object is masked from ‘package:stats’:

    sigma

Loading required package: dplyr
Warning: package ‘dplyr’ was built under R version 4.3.2
########################## Warning from 'xts' package ##########################
#                                                                              #
# The dplyr lag() function breaks how base R's lag() function is supposed to  #
# work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
# source() into this session won't work correctly.                             #
#                                                                              #
# Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
# conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
# dplyr from breaking base R's lag() function.                                 #
#                                                                              #
# Code in packages is not affected. It's protected by R's namespace mechanism #
# Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
```

```
#                                                                                #
##################################################################################

Attaching package: ‘dplyr’

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

    first, last

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

    filter, lag

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

    intersect, setdiff, setequal, union

Loading required package: ggplot2
Warning: package ‘ggplot2’ was built under R version 4.3.2Loading required package: lmtest
Warning: package ‘lmtest’ was built under R version 4.3.2Loading required package: knitr
Warning: package ‘knitr’ was built under R version 4.3.2Loading required package: kableExtra
Warning: package ‘kableExtra’ was built under R version 4.3.2
Attaching package: ‘kableExtra’

The following object is masked from ‘package:dplyr’:

    group_rows
```

```
[[1]]
[1] TRUE

[[2]]
[1] TRUE

[[3]]
[1] TRUE

[[4]]
[1] TRUE

[[5]]
[1] TRUE

[[6]]
[1] TRUE

[[7]]
[1] TRUE

[[8]]
[1] TRUE

[[9]]
[1] TRUE

[[10]]
[1] TRUE

[[11]]
[1] TRUE
```

2. Data Preparation

2.1. Fetching Asian Paints Daily Stock Price Data of Last 10 Years

I've chosen Asian Paints Daily Stock Price Data to do my time-series analysis. I went to yahoo finance and checked the symbols for the Asian Paints stock listed on the National Stock Exchange.
Using the symbol of Asian Paints in getSymbols() function, I fetched the daily stock data for past 10 years, and stored its daily adjusted closing price in an xts time series variable named **"ap_price_xts".**

Hide

```
getSymbols(Symbols = 'ASIANPAINT.NS',
           src = 'yahoo',
           from = as.Date('2014-01-01'),
           to = as.Date('2023-12-31'),
           periodicity = 'daily')
```

```
Warning: ASIANPAINT.NS contains missing values. Some functions will not work if objects conta
in missing values in the middle of the series. Consider using na.omit(), na.approx(), na.fill
(), etc to remove or replace them.
```

```
[1] "ASIANPAINT.NS"
```

Hide

```
ap_price_xts = na.omit(ASIANPAINT.NS$ASIANPAINT.NS.Adjusted)
```

2.2. Retrieving and Cleaning the Daily Adjusted Closing Price

I also created a simple data frame object of daily adjusted closing prices of the stock named **"ap_price"**, for plotting graphs and doing any further analysis.

Hide

```
#Adjusted Closing Price
ap_price = na.omit(subset(data.frame(date=index(ASIANPAINT.NS), coredata(ASIANPAINT.NS)), sel
ect = c(date, ASIANPAINT.NS.Adjusted)))
#class(ap_price)
```

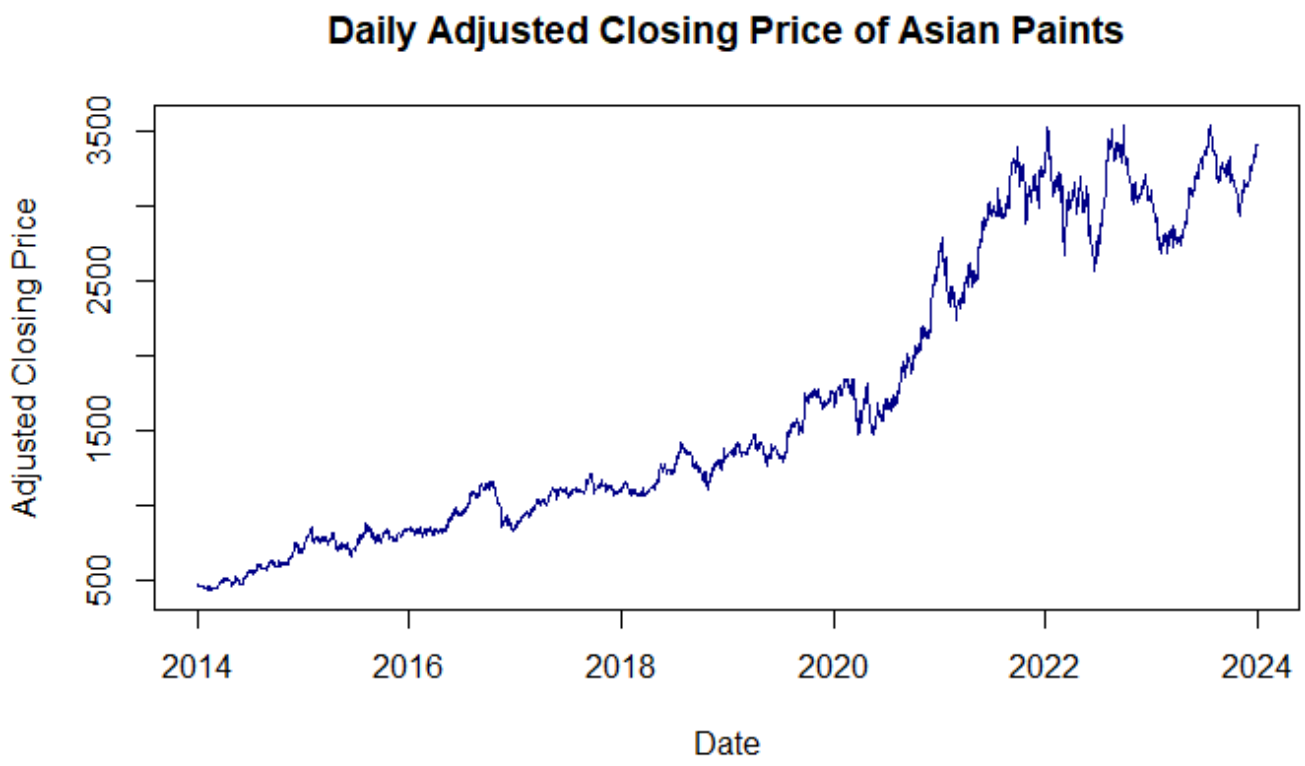
3. Data Exploration

3.1. Exploring Daily Closing Price of Asian Paints

To explore the data, I plotted a graph of adjusted closing stock prices using the plot() function.

Hide

```
# Plot the line graph
plot(ap_price$date, ap_price$ASIANPAINT.NS.Adjusted,
     type = "l", col = "darkblue", lwd = 1.5,
     main = "Daily Adjusted Closing Price of Asian Paints",
     xlab = "Date", ylab = "Adjusted Closing Price")
```



Inference: It seems from the first look that it has some trend, seasonality, and randomness. Also, the variances doesn't seem constant or similar overtime, which indicates heteroscedasticity.

3.2. Exploring **Monthly Average for Every Year**

To delve deeper into my first look speculations, I also plotted a graph of monthly average of adjusted closing stock prices using the plot() function.

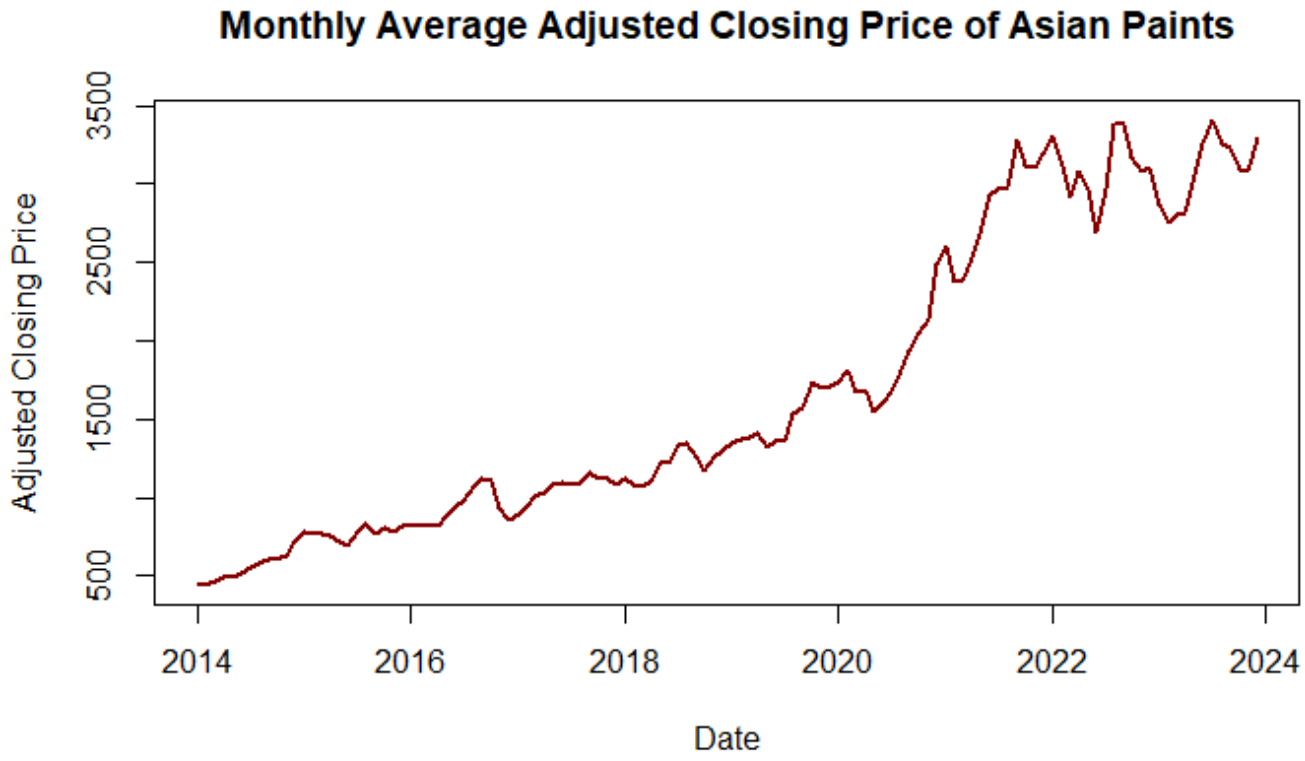
Hide

```
# Extract month and year from the date
ap_price$month_year <- format(ap_price$date, "%Y-%m")

# Aggregate by month and calculate the mean
monthly_avg <- ap_price %>%
  group_by(month_year) %>%
  summarise(avg_adjusted = mean(ASIANPAINT.NS.Adjusted))

# Converting Month to Date Format
monthly_avg$month_year <- as.Date(paste0(monthly_avg$month_year, "-01"))

# Plot the line graph
plot(monthly_avg$month_year, monthly_avg$avg_adjusted,
     type = "l", col = "darkred", lwd = 2,
     main = "Monthly Average Adjusted Closing Price of Asian Paints",
     xlab = "Date", ylab = "Adjusted Closing Price")
```



Inference: This plot is somewhat similar to that of daily closing price, and also suggests that there is some trend, seasonality, randomness, and heteroscedasticity in data.

3.3. Matrix of Average Stock Price in Each Month and Year

I tried to explore the long-term trend of data by forming a 2-dimensional matrix of stock prices named **"ap_matrix"**, with years and months as the 2 axes respectively.

Hide

```
# Extract year and month from the date
ap_price$year <- format(ap_price$date, "%Y")
ap_price$month <- format(ap_price$date, "%b")

# Create an empty list to store the average values
monthly_averages <- list()

# Calculate the average values for each month and year
for (year in unique(ap_price$year)) {
  for (month in unique(ap_price$month)) {
    monthly_data <- ap_price[ap_price$year == year & ap_price$month == month, "ASIANPAINT.NS.
Adjusted"]
    monthly_averages[[paste0(year, "-", month)]] <- mean(monthly_data, na.rm = TRUE)
  }
}

# Convert the list to a matrix
ap_matrix <- matrix(unlist(monthly_averages),
                    nrow = length(unique(ap_price$year)),
                    ncol = 12, byrow = TRUE)

start_year <- min(as.numeric(unique(ap_price$year)))

# Defining the row_names or index as year of the stock price
rownames(ap_matrix) <- seq(start_year, length.out = nrow(ap_matrix))

colnames(ap_matrix) <- substr(month.abb, 1, 3)

rounded_table <- kable(round(ap_matrix, 1), caption = "Monthly Average of Closing Prices of A
sian Paints", "html") %>%
                 kable_styling(full_width = FALSE,
                               bootstrap_options =
                               c("striped","hover", "condensed")) %>%
                 row_spec(0, bold = TRUE,
                          background = "darkred",
                          color = "white") %>%
                 column_spec(1, bold = TRUE, border_right = TRUE)
rounded_table
```

Monthly Average of Closing Prices of Asian Paints

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2014	451.4	441.7	468.2	492.5	488.2	513.1	560.0	580.1	605.0	605.7	637.3	715.7
2015	779.1	765.8	766.1	759.2	724.7	696.9	777.7	833.2	770.0	807.3	779.7	821.2
2016	823.8	822.1	826.3	826.2	886.7	950.1	985.6	1074.9	1118.3	1111.7	931.3	859.4
2017	900.4	947.2	1004.9	1028.6	1086.0	1094.8	1087.5	1096.7	1159.0	1123.4	1119.1	1085.2
2018	1121.2	1079.5	1073.7	1114.7	1219.7	1225.6	1329.2	1352.5	1255.2	1177.6	1254.1	1300.2
2019	1346.3	1371.1	1386.2	1412.4	1319.1	1356.0	1359.0	1530.2	1573.7	1731.5	1700.8	1712.0
2020	1739.9	1807.7	1675.7	1682.7	1545.7	1617.2	1675.1	1807.4	1934.6	2055.4	2139.0	2492.5

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2021	2609.6	2374.1	2374.7	2526.7	2684.7	2928.4	2972.3	2975.0	3280.3	3121.3	3105.7	3192.9
2022	3310.7	3159.0	2922.6	3086.5	2973.7	2694.9	2962.7	3383.9	3380.5	3169.6	3082.2	3104.2
2023	2872.3	2754.9	2799.1	2814.9	3055.3	3258.8	3412.2	3252.6	3236.7	3097.4	3087.8	3289.2

3.4. Aggregate Month-wise Average Stock Price

I also wanted to explore month-wise trend or cyclicity regardless of the year, so I combined data of all the years using colMeans() function.
Then, I plotted a scatter plot of average monthly closing price, and a smooth trend line using loess() function. LOESS stands for Locally Estimated Scatter plot Smoothing, a non-parametric local regression technique to fit a smooth curve through a scatter plot data.

Hide

```
#Calculate the monthly averages
monthly_averages <- colMeans(ap_matrix)

# Plotting the scatter graph
plot(monthly_averages,
     type = "n",
     xlab = "",
     ylab = "Average Stock Price",
     main = "Monthly Average Stock Price",
     ylim = range(monthly_averages),
     xaxt = "n")
axis(side = 1, at = 1:12, labels = colnames(ap_matrix))
```

Hide

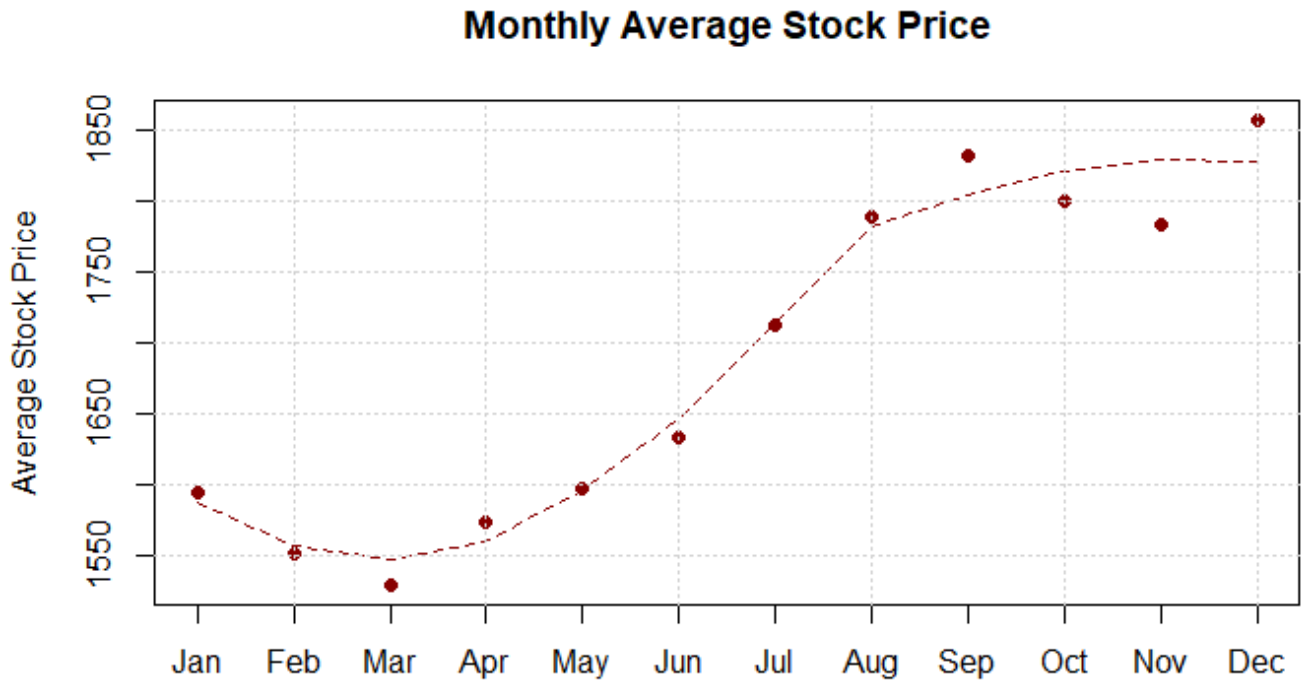
```
points(monthly_averages, pch = 16, col = "darkred")

# Plot the LOESS line as a dashed line
loess_fit <- loess(monthly_averages ~ seq_along(monthly_averages))
smoothed_values <- predict(loess_fit, newdata = data.frame(x = seq_along(monthly_averages)))
lines(seq_along(monthly_averages), smoothed_values, type = "l", col = "darkred", lty = 2)
```

Hide

```

# Add gridlines
grid()
```



Inference: During the second half of the year, Asian Paints' stock on average, experiences a higher closing price. Also, there seems a month-wise cyclicity in the closing stock prices.

4. Checking Stationarity in Daily Closing Price of Stock

4.1. What is Stationarity?

Stationarity refers to a property of data where statistical properties such as mean, variance, and autocorrelation structure remain constant over time. In simpler terms, it means that the data does not exhibit trends, seasonality, or other systematic patterns that change over time.

4.2. Augmented Dickey-Fuller (ADF) Test

Now, I'll perform ADF test to check whether the data is stationary or not.

Null Hypothesis - H0: Data is not stationary.
Alternate Hypothesis - H1: Data is stationary.

Hide

```
# Perform the Augmented Dickey-Fuller (ADF) test
adf_result <- adf.test(ap_price_xts)

# Print the ADF test results
print(adf_result)
```

```

    Augmented Dickey-Fuller Test

data:  ap_price_xts
Dickey-Fuller = -2.5892, Lag order = 13, p-value = 0.3289
alternative hypothesis: stationary
```

As p = 0.3289 i.e. p > 5%
So, Null Hypothesis can't be Rejected.

Inference: Time series of closing price of Asian Paints can be Non-Stationary, i.e. it may have some trend or seasonality.

4.3. Visualizing Trend in Stock Price of Asian Paints

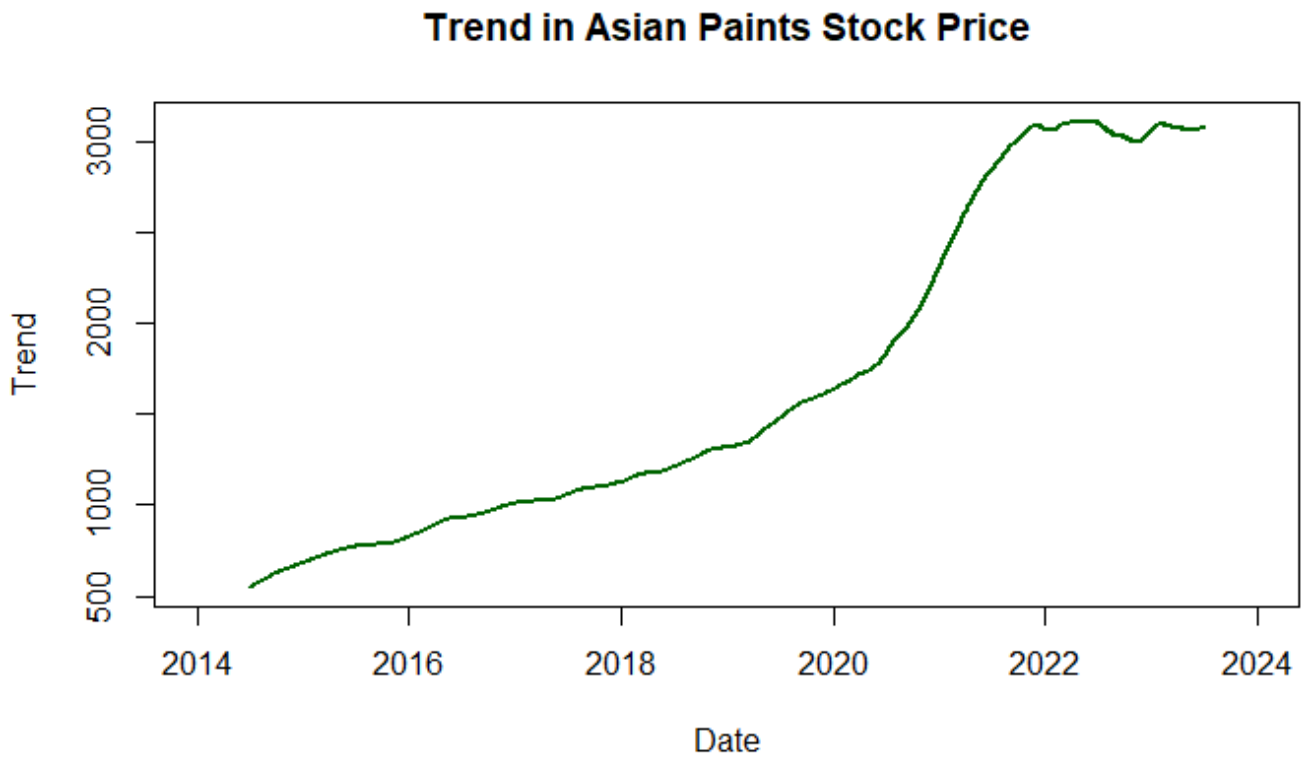
Then to confirm the null hypothesis, I tried to plot the trend component in the daily closing price using decompose() and plot() function.

Hide

```
# Assuming daily frequency
ts_data <- ts(ap_price$ASIANPAINT.NS.Adjusted, frequency = 247)
decomposed <- decompose(ts_data, type = "multiplicative")

# Extract trend component
trend_component <- decompose(ts_data, type = "multiplicative")$trend

# Plot the trend component
plot(ap_price$date,
     trend_component,
     type = "l",
     lwd = 2,
     col = "darkgreen",
     main = "Trend in Asian Paints Stock Price", xlab = "Date", ylab = "Trend")
```



Inference: Time series of daily closing price seems to have an upward trend over the years, and confirms that the data is non-stationary.

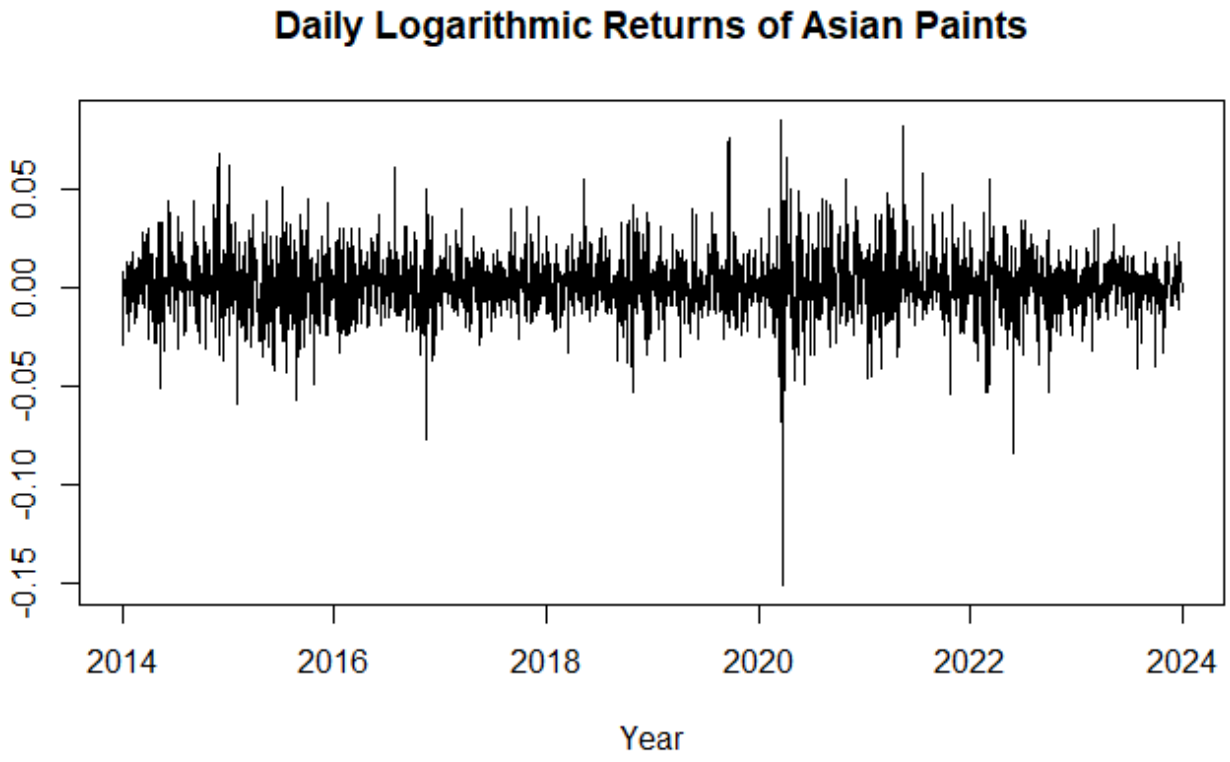
5. Checking Stationarity in Daily Logarithmic Returns of Stock

5.1. Exploring Daily Logarithmic Returns of Asian Paints

As the closing price does not have stationarity, I tried to explore the first order difference, i.e. the logarithm of difference of daily closing prices.

Hide

```
ap_log_returns = na.omit(diff(log(ap_price_xts)))
#View(ap_ds)
plot(index(ap_log_returns), ap_log_returns,
     type = "l",
     lwd = 1,
     main = "Daily Logarithmic Returns of Asian Paints",
     xlab = "Year", ylab = "")
```



Inference: The trend in data is no longer visible, and it seems that data might have become stationary.

5.2. Augmented Dickey-Fuller (ADF) Test

Now, lets check whether the daily log returns passes the stationarity test or not.

Hide

```
# Perform the Augmented Dickey-Fuller (ADF) test
adf_result_2 <- adf.test(ap_log_returns)
```

```
Warning: p-value smaller than printed p-value
```

Hide

```
# Print the ADF test results
print(adf_result_2)
```

```
    Augmented Dickey-Fuller Test

data:  ap_log_returns
Dickey-Fuller = -13.339, Lag order = 13, p-value = 0.01
alternative hypothesis: stationary
```

As p = 0.01 i.e. p < 5%
Null Hypothesis is Rejected.

Inference: Daily Logarithmic Returns of Asian Paints are Stationary.

6. Checking for Autocorrelation

6.1. What is Auto-correlation?

Autocorrelation, also known as serial correlation, is a statistical phenomenon that measures the degree of linear relationship between consecutive observations in a time series data. Specifically, autocorrelation measures the correlation between an observation and its past observations at various lags (time intervals).

6.2. Ljung-Box or Box-Pierce Test

Now, we've established stationarity, let's check for Autocorrelation in daily log returns using the box test.

Null Hypothesis - H0: No Autocorrelation
Alternate Hypothesis - H1: Has Autocorrelation

Hide

```
lb_result = Box.test(ap_log_returns)
print(lb_result)
```

```
    Box-Pierce test

data:  ap_log_returns
X-squared = 2.7587, df = 1, p-value = 0.09673
```

As p = 0.0967 i.e. p > 5%
So, Null Hypothesis can't be Rejected.

Inference: We are safe to assume that there is no significant autocorrelation in log returns of Asian Paints.

7. ARIMA Modelling for Autocorrelation

7.1. What is ARIMA?

An Auto-Regressive Integrated Moving Average, or ARIMA, is a statistical analysis model that uses time series data to better understand the data and to predict future trends. It has following 3 components:

- **Autoregression (AR):** It refers to a model that shows a changing variable that regresses on its own lagged, or prior, values.

- **Integrated (I):** represents the differencing of raw observations to allow the time series to become stationary (i.e., data values are replaced by the difference between the data values and the previous values).
- **Moving average (MA):** incorporates the dependency between an observation and a residual error from a moving average model applied to lagged observations.

7.2. What is an Autocorrelation Function (ACF)?

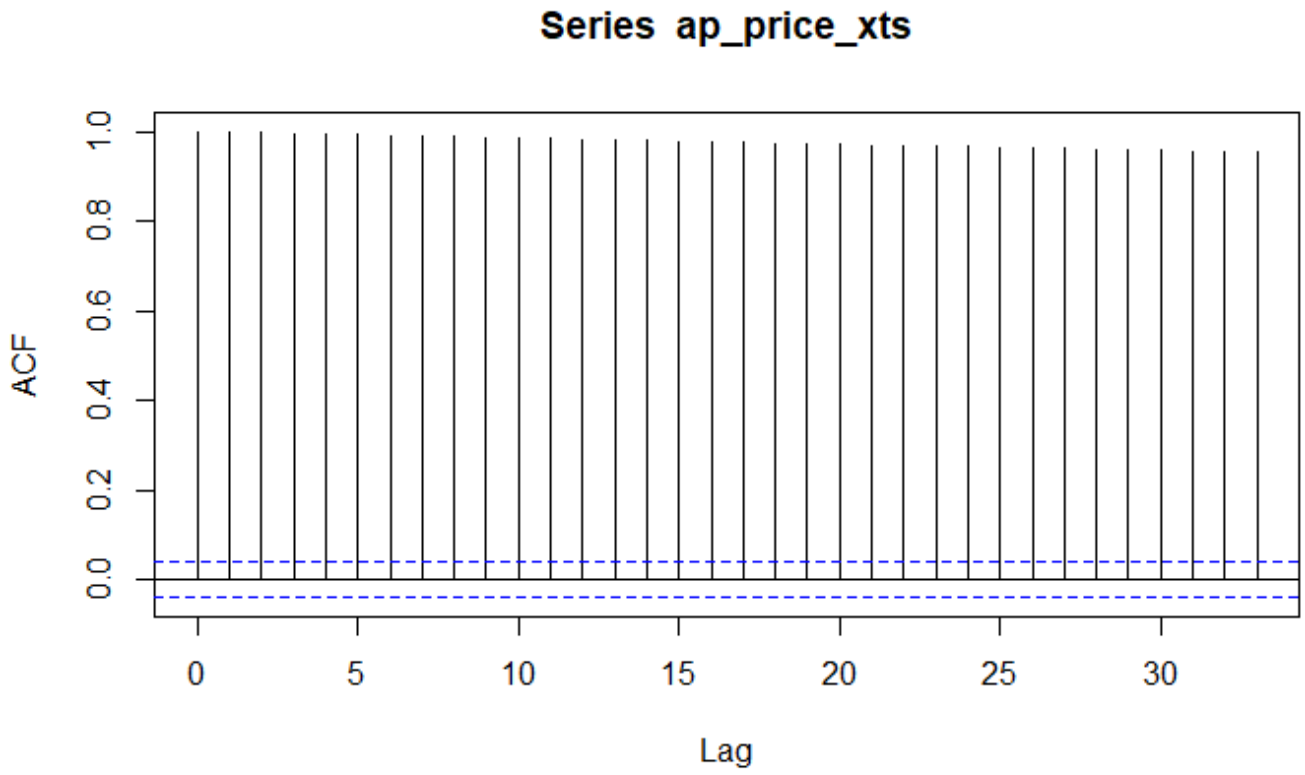
The Autocorrelation Function (ACF) is a statistical tool used to measure the correlation between observations in a time series at different time lags. It quantifies the relationship between an observation and its past observations.

7.3. Autocorrelation Function (ACF) of Daily Closing Prices

We tried to explore the plot of autocorrelation function for daily closing stock prices using acf() function.

Hide

```
acf(ap_price_xts)
```



The ACF values for most lags are somewhere around 1 which is much higher than the significance level (blue horizontal line), suggesting significant auto-correlation at all lag values for daily closing price of Asian Paints. Also, there is a slow decay in acf, which also suggests non-stationarity.

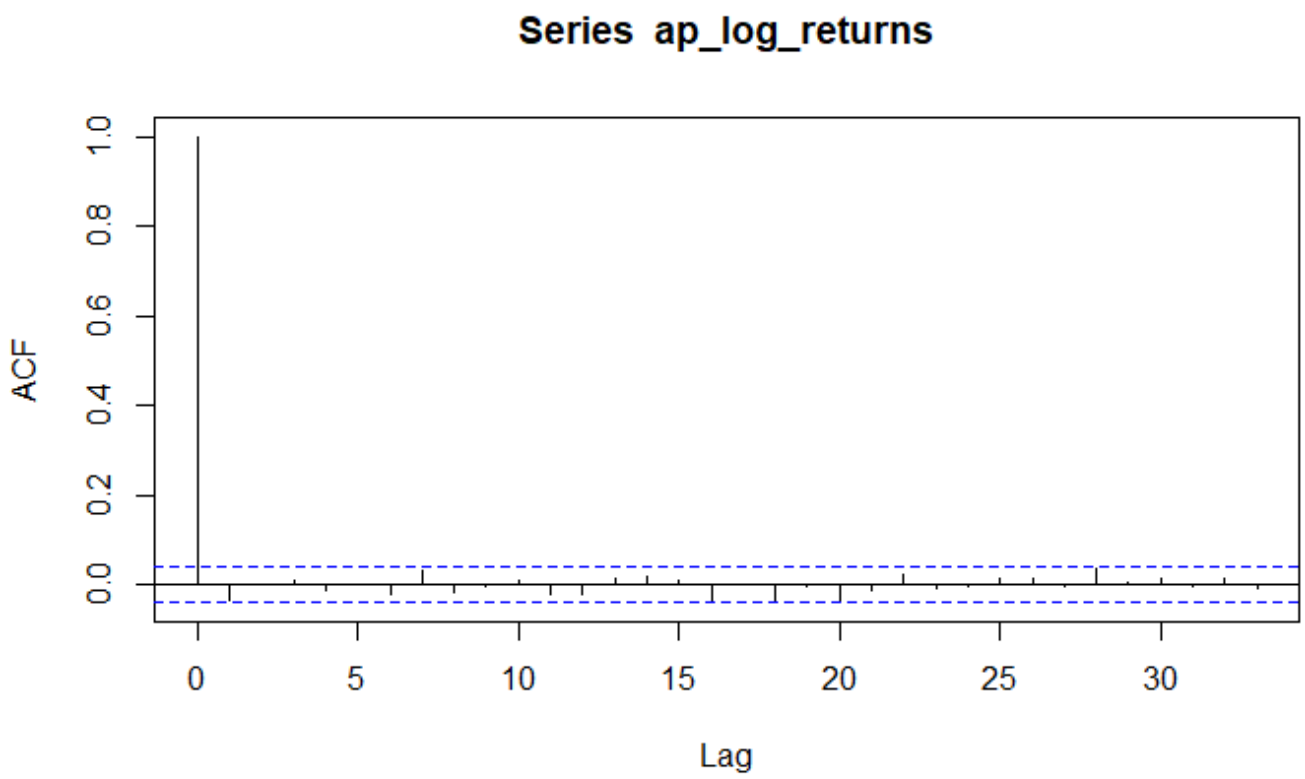
Inference: There can be autocorrelation in daily closing price of the stock. It also confirms our previous inference that the daily closing price of the stock is non-stationary.

7.4. Autocorrelation Function (ACF) Of Logarithm of Daily Return

Now, to develop an appropriate ARIMA model, I tried to explore the autocorrelation function for daily log return using acf() function.

Hide

```
acf(ap_log_returns)
```



Zero MA Order: The small acf values for all other lags after lag=0 suggest that the auto-correlations beyond the lag=0 are not significant. This means we do not need any moving average component as values are not much dependent on its immediate previous values.

Zero Integrating/Differencing Order: The log returns are stationary, as we've tested before, there is no differencing required.

Zero AR Order: Since there are no significant autocorrelations at higher lags after 0, it suggests that there is no autocorrelation, which is also supported by our previous inference. Therefore, we need a zero AR order.

Inference: This suggests an ARIMA Model with Potential Order (0,0,0).

7.4. What is a Partial Autocorrelation Function (PACF)?

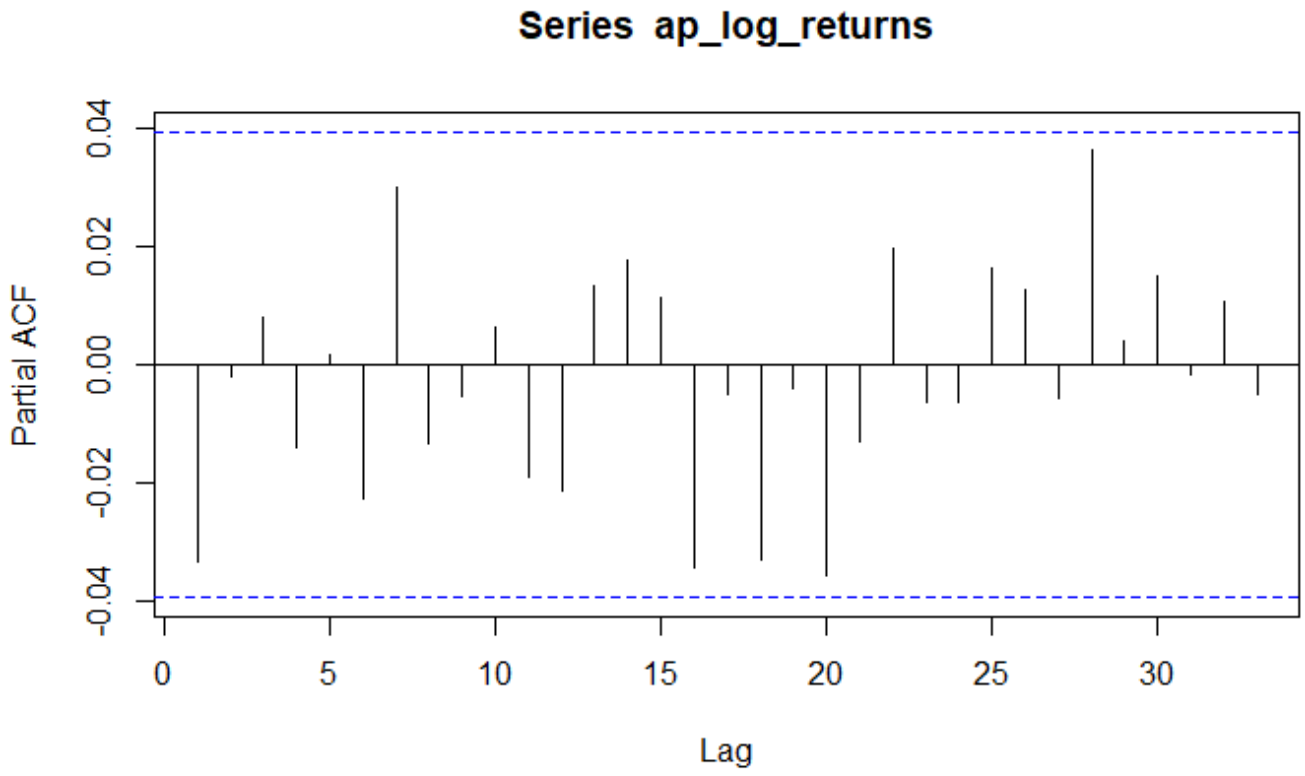
The Partial Autocorrelation Function (PACF) is a statistical tool used in time series analysis to measure the correlation between observations in a time series at different lags, while controlling for the intermediate lags. In other words, it quantifies the direct relationship between an observation and its past observations, excluding the indirect effects of the intermediate observations.

7.5. Partial Autocorrelation Function (PACF) of Logarithm of Daily Return

I also ran pacf() function to look at the partial autocorrelations at different time lags.

Hide

```
pacf(ap_log_returns)
```



Here, all PACF values are below the significance level and quite random which implies that there is no direct relationship between observations at any lag.

Inference: This again confirms lack of autoregression or moving average terms.

7.6. ARIMA MODEL 1 - WITH ORDER(0,0,0) - My Suggested Model

Now, I'm creating an ARIMA Model manually using the arima() function with the order I feel fits the best by looking at the acf() and pacf() functions. I named this model as **"arima_model1"**.

Hide

```
arima_model1 = arima(ap_log_returns, order = c(0,0,0))
print(arima_model1)
```

```

Call:
arima(x = ap_log_returns, order = c(0, 0, 0))

Coefficients:
      intercept
          8e-04
s.e.      3e-04

sigma^2 estimated as 0.0002609:  log likelihood = 6669.59,  aic = -13335.19
```

7.7. ARIMA MODEL 2 - WITH ORDER(1,0,0) - Generated by Auto ARIMA

I also ran auto.arima() function to have another model in case my model fails. This model came out to be of order (1,0,0) and I named it as **"arima_model2"**.

Hide

```
arima_model2 = auto.arima(ap_log_returns)
print(arima_model2)
```

```
Series: ap_log_returns
ARIMA(1,0,0) with non-zero mean

Coefficients:
          ar1   mean
      -0.0335  8e-04
s.e.   0.0201  3e-04

sigma^2 = 0.0002608:  log likelihood = 6670.98
AIC=-13335.95   AICc=-13335.94   BIC=-13318.52
```

7.8. Comparing the Two ARIMA Models

Now, lets compare mean absolute errors of both the models and see which one is better.

Hide

```
# ARIMA Model 1
print(paste0("ARIMA Model 1 - ORDER(0,0,0)"))
```

```
[1] "ARIMA Model 1 - ORDER(0,0,0)"
```

Hide

```
model1_mae = mean(abs(arima_model1$residuals))
print(paste0("Mean Absolute Error: ",round(model1_mae,6)))
```

```
[1] "Mean Absolute Error: 0.011569"
```

Hide

```
cat("\n")
```

Hide

```
# ARIMA Model 2
print(paste0("ARIMA Model 2 - ORDER(1,0,0)"))
```

```
[1] "ARIMA Model 2 - ORDER(1,0,0)"
```

Hide

```
model2_mae = mean(abs(arima_model2$residuals))
print(paste0("Mean Absolute Error: ",round(model2_mae,6)))
```

```
[1] "Mean Absolute Error: 0.011558"
```

Inference: Both models have nearly equal absolute error, with model 2 slightly having the edge.

8. Exploring Logarithmic Daily Return of the Stock

8.1. Average Logarithmic Daily Return of Asian Paints

Now, that I'm working with Log of Daily Returns of a stock, and built ARIMA Models for it, its better to delve deeper and understand this data better. So first , let's look at the mean of our data.

Hide

```
mean_ap_log_return = mean(ap_log_returns)
print(paste0("Average Logarithmic Daily Return of Asian Paints:", round(mean_ap_log_return,
6)))
```

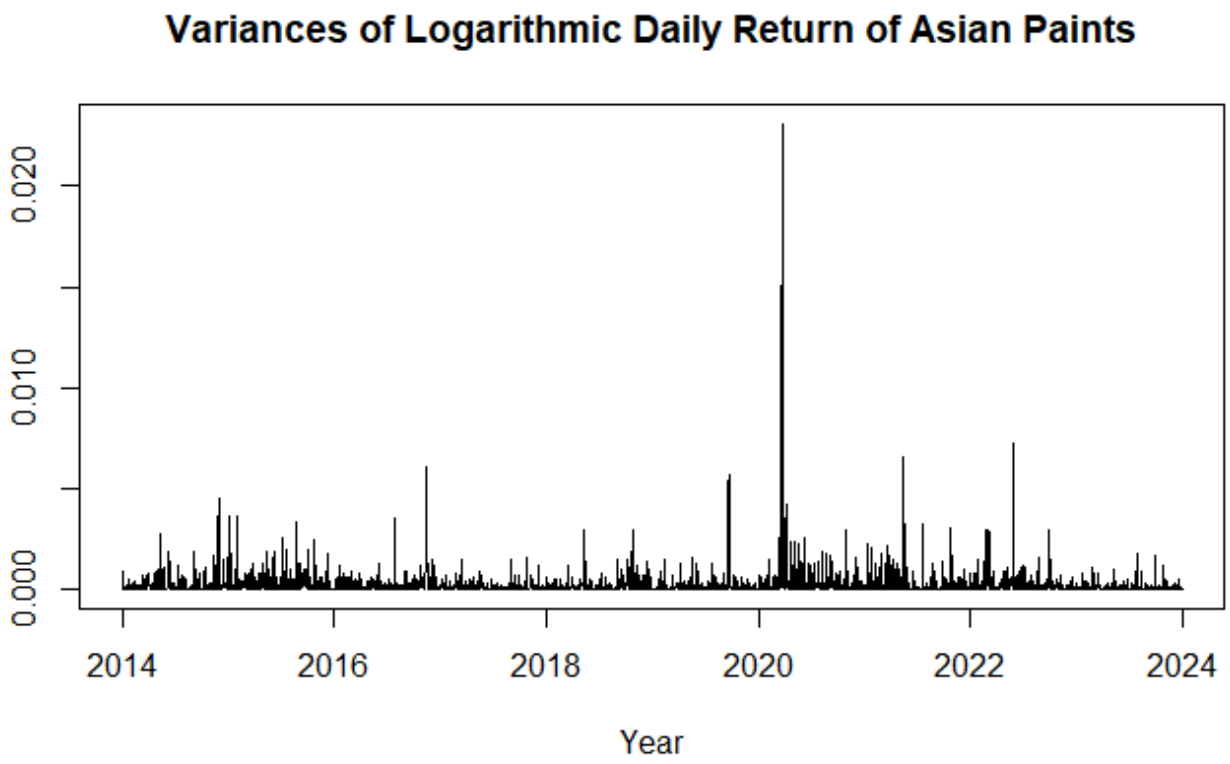
```
[1] "Average Logarithmic Daily Return of Asian Paints:0.000808"
```

8.2. Plotting Variances of Log Return

Next up, let's look at how the data varies over time by plotting variances of the data from its mean.

Hide

```
ap_log_returns_sq = (ap_log_returns-mean_ap_log_return)^2
plot(index(ap_log_returns_sq), ap_log_returns_sq,
     type = "l",
     lwd = 1,
     main = "Variances of Logarithmic Daily Return of Asian Paints",
     xlab = "Year", ylab = "")
```



The data seems to have a good amount of variation in its deviation from mean, which indicates heteroscedasticity. Also, there is an extraordinarily high variation during 2020, possibly due to the panic in stock markets during the COVID-19 pandemic.

Inference: The data might have heteroscedasticity.

8.3. Overall Variation in of Log Returns

Now, let's look at the variation in our data by calculating its variance, standard deviation, and coefficient of variation.

Hide

```
ap_log_return_variance = mean(ap_log_returns_sq)
print(paste0("Variance of Log Return of Asian Paints: ", round(ap_log_return_variance,6)))
```

```
[1] "Variance of Log Return of Asian Paints: 0.000261"
```

Hide

```
ap_log_return_sd = sqrt(ap_log_return_variance)
print(paste0("Standard Deviation of Log Return of Asian Paints: ", round(ap_log_return_sd,
6)))
```

```
[1] "Standard Deviation of Log Return of Asian Paints: 0.016151"
```

Hide

```
print(paste0("Coefficient of Variation of Log Return of Asian Paints: ", round(ap_log_return_
sd/mean_ap_log_return,4)))
```

```
[1] "Coefficient of Variation of Log Return of Asian Paints: 19.9864"
```

9. Checking for Heteroskedasticity for Log of Daily Returns

9.1. What is Heteroscedasticity?

Heteroscedasticity is a situation where the variability of a variable (often the residuals or errors in a regression model) is not constant across all levels of another variable. In simpler terms, it means that the spread of the data points a.k.a. residuals changes as the value of another variable changes.

9.2. Box Test for Heteroscedasticity

Null Hypothesis - H0: There is No Heteroscedasticity in the residuals
Alternate Hypothesis - H1: There is Heteroscedasticity in the residuals

Hide

```
print(Box.test(ap_log_returns_sq, lag = 10))
```

```

    Box-Pierce test

data:  ap_log_returns_sq
X-squared = 265.4, df = 10, p-value < 2.2e-16
```

9.3. ARCH LM Test for Heteroscedasticity

Null Hypothesis - H0: There is No Heteroscedasticity in the residuals
Alternate Hypothesis - H1: There is Heteroscedasticity in the residuals

Hide

```
print(ArchTest(arima_model1$residuals, lags = 20))
```

```
    ARCH LM-test; Null hypothesis: no ARCH effects

data:  arima_model1$residuals
Chi-squared = 194.28, df = 20, p-value < 2.2e-16
```

As p = 2.2e-16 i.e. p < 5%
Null Hypothesis is Rejected.

Inference: Residuals of the log of daily returns have conditional heteroscedasticity.

10. Checking for Heteroskedasticity for the ARIMA Models

10.1. Box Test for Heteroscedasticity - Model 1

Null Hypothesis - H0: There is No Heteroscedasticity in the residuals
Alternate Hypothesis - H1: There is Heteroscedasticity in the residuals

Hide

```
print(Box.test((arima_model1$residuals)^2, lag = 10))
```

```
    Box-Pierce test

data:  (arima_model1$residuals)^2
X-squared = 265.4, df = 10, p-value < 2.2e-16
```

10.2. ARCH LM Test for Heteroscedasticity - Model 1

Null Hypothesis - H0: There is No Heteroscedasticity in the residuals
Alternate Hypothesis - H1: There is Heteroscedasticity in the residuals

Hide

```
print(ArchTest(arima_model1$residuals, lags = 20))
```

```
    ARCH LM-test; Null hypothesis: no ARCH effects

data:  arima_model1$residuals
Chi-squared = 194.28, df = 20, p-value < 2.2e-16
```

As p = 2.2e-16 i.e. p < 5%
Null Hypothesis is Rejected.

Inference: Residuals of ARIMA Model 1 also have conditional heteroscedasticity.

10.3. Box Test for Heteroscedasticity - Model 2

Null Hypothesis - H0: There is No Heteroscedasticity in the residuals
Alternate Hypothesis - H1: There is Heteroscedasticity in the residuals

Hide

```
print(Box.test((arima_model2$residuals)^2, lag = 10))
```

```
    Box-Pierce test

data:  (arima_model2$residuals)^2
X-squared = 258.95, df = 10, p-value < 2.2e-16
```

10.4. ARCH LM Test for Heteroscedasticity - Model 2

Null Hypothesis - H0: There is No Heteroscedasticity in the residuals
Alternate Hypothesis - H1: There is Heteroscedasticity in the residuals

Hide

```
print(ArchTest(arima_model2$residuals, lags = 20))
```

```
    ARCH LM-test; Null hypothesis: no ARCH effects

data:  arima_model2$residuals
Chi-squared = 189.41, df = 20, p-value < 2.2e-16
```

As p = 2.2e-16 i.e. p < 5%
Null Hypothesis is Rejected.

Inference: Residuals of ARIMA Model 2 also have conditional heteroscedasticity.

Now, as there's heteroscedasticity in all the cases above, and we have to remove heteroscedasticity in order to have an accurate and consistent forecast, now we'll try to capture the volatility using GARCH Models.

11. Capturing Volatility with GARCH Models

11.1. What is GARCH Model?

The GARCH (Generalized Auto-Regressive Conditional Heteroscedasticity) model is a statistical model used to capture the time-varying volatility or variance clustering observed in financial time series data. It extends the ARCH model by incorporating not only past squared residuals but also past volatility values to model the conditional variance of the data.

11.2. Standard GARCH Model with Constant Mean and AR Order 0

Now, by assuming GARCH order to be 1, and AR order 0 which we speculated earlier by looking at acf() and pacf() curves, we built a model named **"garch_model1"**.

Hide

```
garch_model1 = ugarchspec(variance.model = list(model = 'sGARCH', garchOrder = c(1,1)), mean.
model = list(armaOrder = c(0,0), include.mean = TRUE))
ap_log_returns_garch1 = ugarchfit(garch_model1, data = ap_log_returns_sq)
print(ap_log_returns_garch1)
```

```
*---------------------------------*
*          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      0.000196    0.000009  22.83238   0.0000
omega   0.000000    0.000000   0.06383   0.9491
alpha1  0.047291    0.001510  31.31330   0.0000
beta1   0.919393    0.002650 347.00301   0.0000

Robust Standard Errors:
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.000196    0.010214 0.019207  0.98468
omega   0.000000    0.000360 0.000030  0.99998
alpha1  0.047291    7.012308 0.006744  0.99462
beta1   0.919393    2.324644 0.395498  0.69247

LogLikelihood : 15099.29

Information Criteria
------------------------------------

Akaike       -12.253
Bayes        -12.243
Shibata      -12.253
Hannan-Quinn -12.249

Weighted Ljung-Box Test on Standardized Residuals
------------------------------------
                        statistic   p-value
Lag[1]                      46.36 9.837e-12
Lag[2*(p+q)+(p+q)-1][2]     49.51 8.349e-14
Lag[4*(p+q)+(p+q)-1][5]     54.43 3.331e-15
d.o.f=0
H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals
------------------------------------
                        statistic p-value
Lag[1]                      5.930 0.01488
Lag[2*(p+q)+(p+q)-1][5]     8.165 0.02682
Lag[4*(p+q)+(p+q)-1][9]    12.064 0.01754
d.o.f=2

Weighted ARCH LM Tests
------------------------------------
```

```
            Statistic Shape Scale P-Value
ARCH Lag[3]   0.08321 0.500 2.000 0.77299
ARCH Lag[5]   6.30818 1.440 1.667 0.05082
ARCH Lag[7]   7.68455 2.315 1.543 0.06162

Nyblom stability test
------------------------------------
Joint Statistic:  75.4666
Individual Statistics:
mu     1.6443
omega  9.7593
alpha1 0.1479
beta1  0.1308

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 <dbl>	prob <dbl>	sig <chr>
Sign Bias	1.183390	2.367691e-01	
Negative Sign Bias	1.263860	2.064001e-01	
Positive Sign Bias	4.285555	1.893007e-05	***
Joint Effect	24.123852	2.353600e-05	***

4 rows

```
Adjusted Pearson Goodness-of-Fit Test:
------------------------------------
  group statistic p-value(g-1)
1    20      4941            0
2    30      5024            0
3    40      5175            0
4    50      5248            0


Elapsed time : 0.161855
```

11.3. Simple GARCH Model with Constant Mean and AR Order 1

Now, by assuming GARCH order to be 1, and AR order 1 which we received from auto arima, we built a model named **“garch_model2”**.

Hide

```
garch_model2 = ugarchspec(variance.model = list(model = 'sGARCH', garchOrder = c(1,1)), mean.
model = list(armaOrder = c(1,0), include.mean = TRUE))
ap_log_returns_garch2 = ugarchfit(garch_model2, data = ap_log_returns_sq)
print(ap_log_returns_garch2)
```

```
*----------------------------------*
*          GARCH Model Fit         *
*----------------------------------*

Conditional Variance Dynamics
-----------------------------------
GARCH Model : sGARCH(1,1)
Mean Model  : ARFIMA(1,0,0)
Distribution    : norm

Optimal Parameters
------------------------------------
        Estimate  Std. Error    t value Pr(>|t|)
mu      0.000231    0.000008  27.415824  0.00000
ar1     0.224210    0.027057   8.286544  0.00000
omega   0.000000    0.000000   0.045046  0.96407
alpha1  0.057300    0.002726  21.023235  0.00000
beta1   0.912263    0.003474 262.567155  0.00000

Robust Standard Errors:
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.000231    0.028989 0.007972  0.99364
ar1     0.224210    4.116480 0.054467  0.95656
omega   0.000000    0.000697 0.000015  0.99999
alpha1  0.057300   17.655592 0.003245  0.99741
beta1   0.912263    1.738391 0.524774  0.59974

LogLikelihood : 15129.36

Information Criteria
------------------------------------

Akaike       -12.276
Bayes        -12.264
Shibata      -12.276
Hannan-Quinn -12.272

Weighted Ljung-Box Test on Standardized Residuals
------------------------------------
                        statistic p-value
Lag[1]                     0.3256  0.5683
Lag[2*(p+q)+(p+q)-1][2]    1.2161  0.6063
Lag[4*(p+q)+(p+q)-1][5]    3.5212  0.3122
d.o.f=1
H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals
------------------------------------
                        statistic p-value
Lag[1]                      2.379  0.1230
Lag[2*(p+q)+(p+q)-1][5]     3.338  0.3487
Lag[4*(p+q)+(p+q)-1][9]     5.171  0.4047
d.o.f=2
```

```
Weighted ARCH LM Tests
------------------------------------
            Statistic Shape Scale P-Value
ARCH Lag[3] 0.0002487 0.500 2.000  0.9874
ARCH Lag[5] 2.7675714 1.440 1.667  0.3253
ARCH Lag[7] 3.4846537 2.315 1.543  0.4270

Nyblom stability test
------------------------------------
Joint Statistic:  100.9738
Individual Statistics:
mu      0.2148
ar1     0.1730
omega  13.2735
alpha1  0.1031
beta1   0.1261

Asymptotic Critical Values (10% 5% 1%)
Joint Statistic:         1.28 1.47 1.88
Individual Statistic:    0.35 0.47 0.75

Sign Bias Test
------------------------------------
```

	t-value <dbl>	prob <dbl>	sig <chr>
Sign Bias	0.6671005	0.504770509	
Negative Sign Bias	0.4219393	0.673106228	
Positive Sign Bias	3.1129871	0.001873342	***
Joint Effect	13.5396505	0.003603701	***

4 rows

```
Adjusted Pearson Goodness-of-Fit Test:
------------------------------------
  group statistic p-value(g-1)
1    20      3654            0
2    30      3743            0
3    40      3826            0
4    50      3846            0


Elapsed time : 0.245558
```

11.4. ARCH LM Test for Heteroscedasticity - GARCH Model 1

Null Hypothesis - H0: There is No Heteroscedasticity in the residuals
Alternate Hypothesis - H1: There is Heteroscedasticity in the residuals

Hide

```
# Perform ARCH LM test
arch_test_model1 <- ArchTest(residuals(ap_log_returns_garch1), lags = 20)
print(arch_test_model1)
```

```
    ARCH LM-test; Null hypothesis: no ARCH effects

data:  residuals(ap_log_returns_garch1)
Chi-squared = 27.358, df = 20, p-value = 0.1255
```

As p = 0.1255 i.e. p > 5%
Null Hypothesis cannot be Rejected.

Inference: Residuals of GARCH Model 1 may not have conditional heteroscedasticity.

11.5. ARCH LM Test for Heteroscedasticity - GARCH Model 2

Null Hypothesis - H0: There is No Heteroscedasticity in the residuals
Alternate Hypothesis - H1: There is Heteroscedasticity in the residuals

Hide

```
# Perform ARCH LM test
arch_test_model2 <- ArchTest(residuals(ap_log_returns_garch2), lags = 20)
print(arch_test_model2)
```

```
    ARCH LM-test; Null hypothesis: no ARCH effects

data:  residuals(ap_log_returns_garch2)
Chi-squared = 67.623, df = 20, p-value = 4.429e-07
```

As p = 5.6e-07 i.e. p < 5%
Null Hypothesis is Rejected.

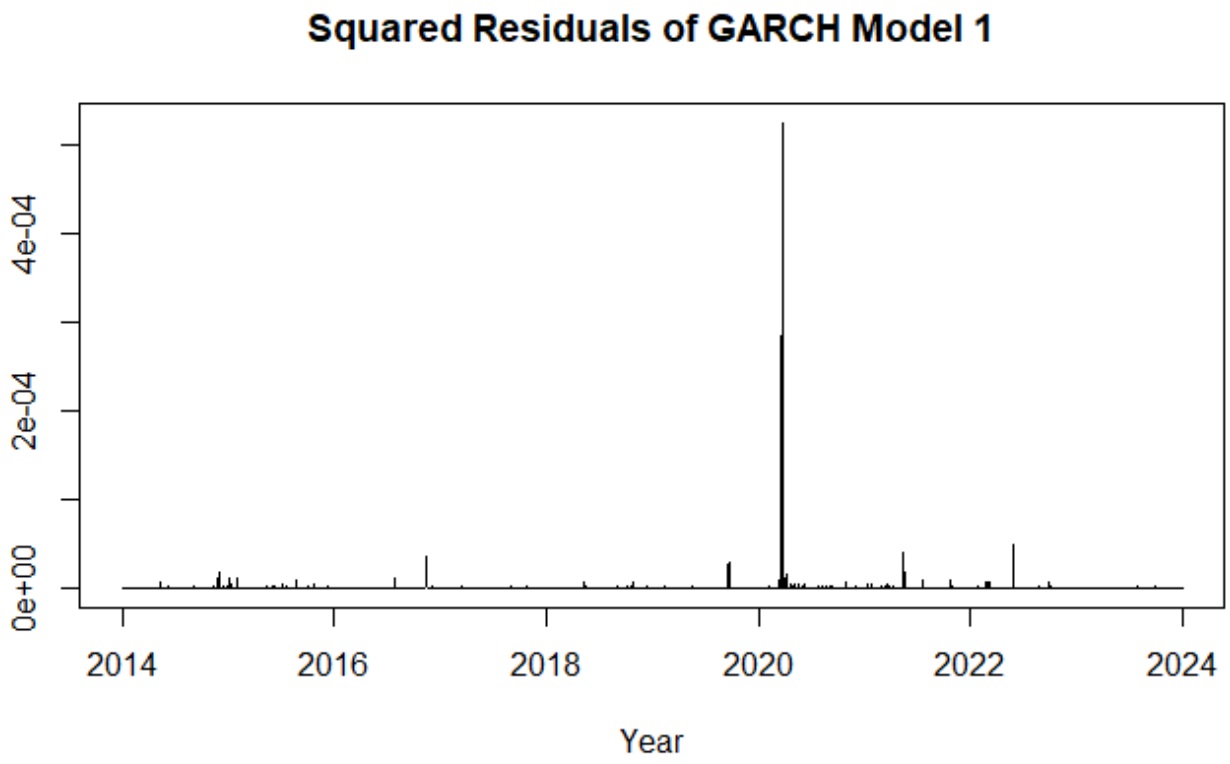
Inference: Residuals of GARCH Model 2 have conditional heteroscedasticity.

11.6. Visualizing Residuals - GARCH Model 1

Though GARCH Model 1 passed the heteroscedasticity test, but let's visualize its residuals to probe further.

Hide

```
garch_residuals_sq = (residuals(ap_log_returns_garch1))^2
plot(index(garch_residuals_sq), garch_residuals_sq,
     type = "l",
     lwd = 1,
     main = "Squared Residuals of GARCH Model 1",
     xlab = "Year", ylab = "")
```



Inference: Except the outlier event of COVID-19, heteroscedasticity seems to be diminished.

12. Using GARCH Model 1 for Forecasting

Hide

```
garch_forecast = ugarchforecast(ap_log_returns_garch1, n.ahead = 100)
garch_forecast
```

```
*-----------------------------------*
*       GARCH Model Forecast        *
*-----------------------------------*
Model: sGARCH
Horizon: 100
Roll Steps: 0
Out of Sample: 0

0-roll forecast [T0=2023-12-29]:
         Series     Sigma
T+1   0.0001962 0.0003881
T+2   0.0001962 0.0003953
T+3   0.0001962 0.0004022
T+4   0.0001962 0.0004087
T+5   0.0001962 0.0004149
T+6   0.0001962 0.0004208
T+7   0.0001962 0.0004265
T+8   0.0001962 0.0004319
T+9   0.0001962 0.0004370
T+10  0.0001962 0.0004419
T+11  0.0001962 0.0004466
T+12  0.0001962 0.0004511
T+13  0.0001962 0.0004554
T+14  0.0001962 0.0004595
T+15  0.0001962 0.0004635
T+16  0.0001962 0.0004673
T+17  0.0001962 0.0004709
T+18  0.0001962 0.0004744
T+19  0.0001962 0.0004777
T+20  0.0001962 0.0004809
T+21  0.0001962 0.0004840
T+22  0.0001962 0.0004870
T+23  0.0001962 0.0004898
T+24  0.0001962 0.0004925
T+25  0.0001962 0.0004952
T+26  0.0001962 0.0004977
T+27  0.0001962 0.0005001
T+28  0.0001962 0.0005025
T+29  0.0001962 0.0005047
T+30  0.0001962 0.0005069
T+31  0.0001962 0.0005090
T+32  0.0001962 0.0005110
T+33  0.0001962 0.0005129
T+34  0.0001962 0.0005148
T+35  0.0001962 0.0005166
T+36  0.0001962 0.0005183
T+37  0.0001962 0.0005199
T+38  0.0001962 0.0005216
T+39  0.0001962 0.0005231
T+40  0.0001962 0.0005246
T+41  0.0001962 0.0005260
T+42  0.0001962 0.0005274
T+43  0.0001962 0.0005287
T+44  0.0001962 0.0005300
```

```
T+45  0.0001962 0.0005313
T+46  0.0001962 0.0005325
T+47  0.0001962 0.0005336
T+48  0.0001962 0.0005347
T+49  0.0001962 0.0005358
T+50  0.0001962 0.0005369
T+51  0.0001962 0.0005379
T+52  0.0001962 0.0005388
T+53  0.0001962 0.0005398
T+54  0.0001962 0.0005407
T+55  0.0001962 0.0005415
T+56  0.0001962 0.0005424
T+57  0.0001962 0.0005432
T+58  0.0001962 0.0005440
T+59  0.0001962 0.0005447
T+60  0.0001962 0.0005454
T+61  0.0001962 0.0005461
T+62  0.0001962 0.0005468
T+63  0.0001962 0.0005475
T+64  0.0001962 0.0005481
T+65  0.0001962 0.0005487
T+66  0.0001962 0.0005493
T+67  0.0001962 0.0005499
T+68  0.0001962 0.0005504
T+69  0.0001962 0.0005509
T+70  0.0001962 0.0005515
T+71  0.0001962 0.0005520
T+72  0.0001962 0.0005524
T+73  0.0001962 0.0005529
T+74  0.0001962 0.0005533
T+75  0.0001962 0.0005538
T+76  0.0001962 0.0005542
T+77  0.0001962 0.0005546
T+78  0.0001962 0.0005550
T+79  0.0001962 0.0005554
T+80  0.0001962 0.0005557
T+81  0.0001962 0.0005561
T+82  0.0001962 0.0005564
T+83  0.0001962 0.0005567
T+84  0.0001962 0.0005570
T+85  0.0001962 0.0005574
T+86  0.0001962 0.0005576
T+87  0.0001962 0.0005579
T+88  0.0001962 0.0005582
T+89  0.0001962 0.0005585
T+90  0.0001962 0.0005587
T+91  0.0001962 0.0005590
T+92  0.0001962 0.0005592
T+93  0.0001962 0.0005594
T+94  0.0001962 0.0005597
T+95  0.0001962 0.0005599
T+96  0.0001962 0.0005601
T+97  0.0001962 0.0005603
T+98  0.0001962 0.0005605
```

```
T+99  0.0001962 0.0005607
T+100 0.0001962 0.0005609
```

Hide

```
plot(garch_forecast)
```

```

Make a plot selection (or 0 to exit):

1:   Time Series Prediction (unconditional)
2:   Time Series Prediction (rolling)
3:   Sigma Prediction (unconditional)
4:   Sigma Prediction (rolling)
```

Hide

```
1
```

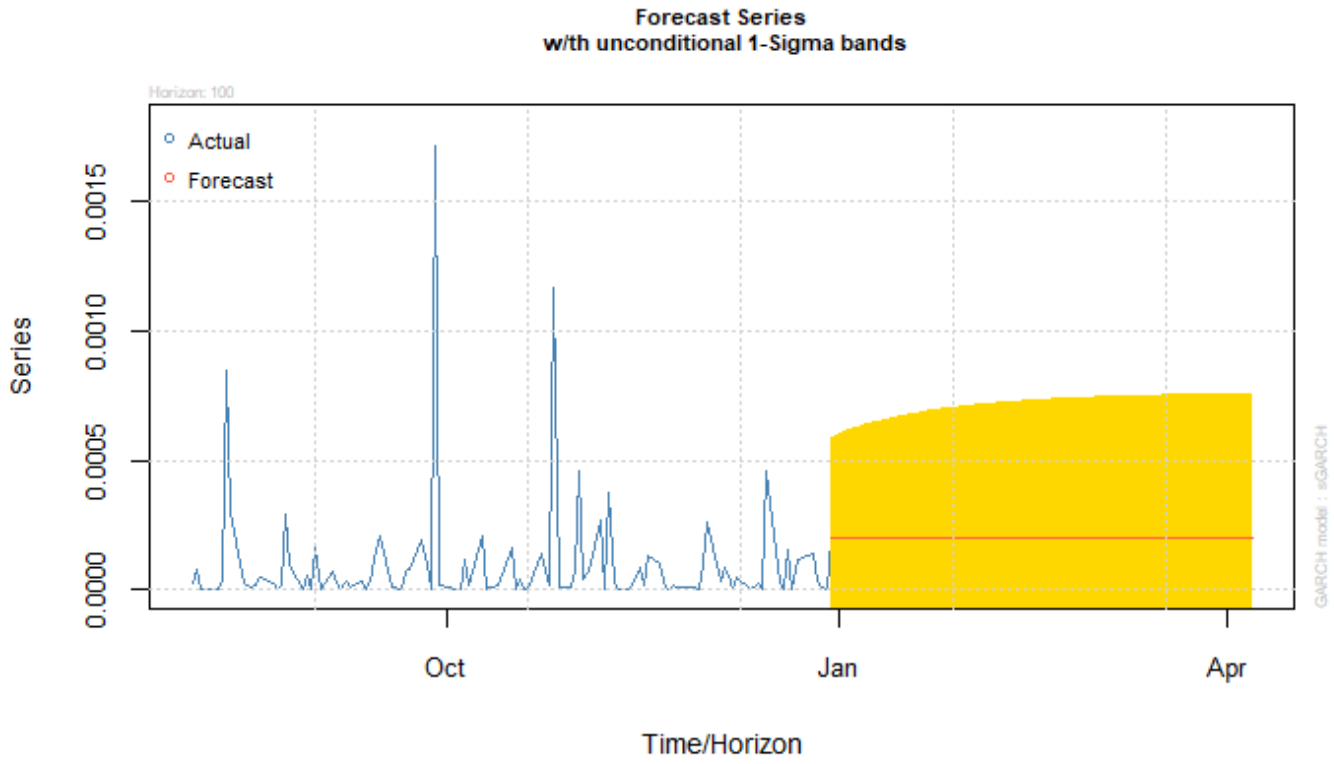
```

Make a plot selection (or 0 to exit):

1:   Time Series Prediction (unconditional)
2:   Time Series Prediction (rolling)
3:   Sigma Prediction (unconditional)
4:   Sigma Prediction (rolling)
```

Hide

```
3
```



```
Make a plot selection (or 0 to exit):

1:   Time Series Prediction (unconditional)
2:   Time Series Prediction (rolling)
3:   Sigma Prediction (unconditional)
4:   Sigma Prediction (rolling)
```

Hide

```
0
```