Applied Financial Econometrics Project

B158582

March 2024

Contents

1 B1: Descriptive Statistics

This paper explores the managed-float USD-VND exchange rate from 01.01.2019 to 01.01.2024 (see Figure 1). The first reason for choosing the USD-VND is that the series lends itself well to event analysis in B4, as the VND depreciated 6% against the USD in October 2022 following a Vietnamese financial crisis. Secondly, the author spent six months as an economist in Vietnam, studying this exchange rate and can offer unique insights [1].

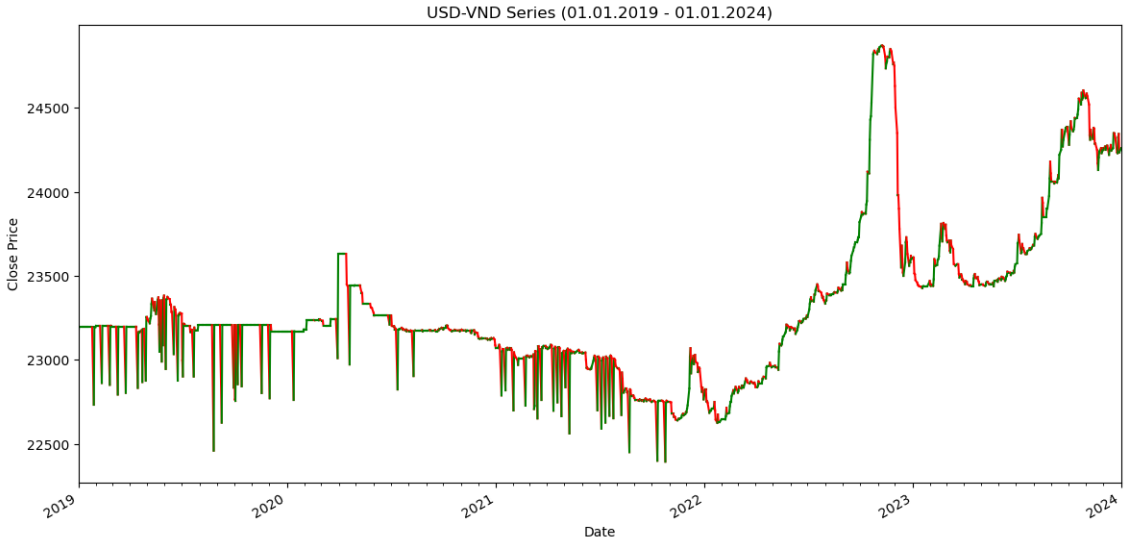


Figure 1: Exchange Rate Series

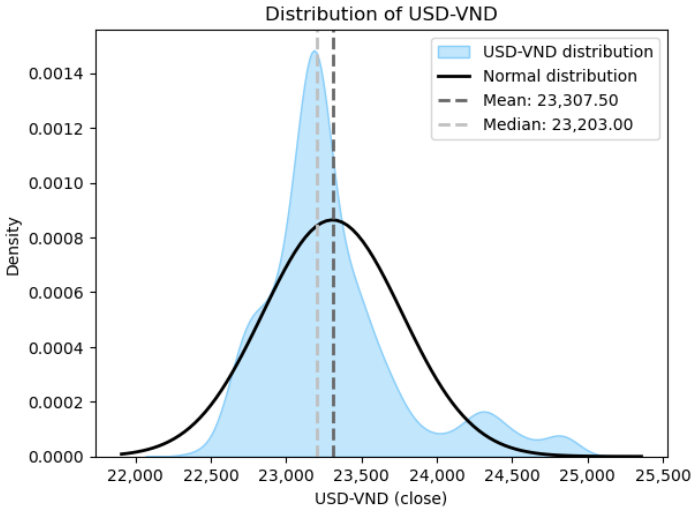


Figure 2: Distribution of the Exchange Rate Series

[1] I consistently cite Kokalari (2022) and Kokalari (2023). I co-authored these reports and produced much of the data analysis. They were part of a bi-weekly series, which we dispatched to media, investors, and the Vietnamese prime minister.

Table 1: Summary Statistics of USD-VND (Close Values)

Statistic	**Value**
Mean Value	23 307.43
Standard Deviation	461.90
Excess Kurtosis	−1.03
Skewness	1.36
Minimum Value	22 394.35
25th Percentile	23 050.00
50th Percentile (Median)	23 203.00
75th Percentile	23 450.00
Maximum Value	24 871.00

Jointly inspecting Figures 1 and 2 reveals five characteristics (see Table 2).

First, the exchange rate series has a non-normal, platykurtic, and multi-modal distribution. It is evidently non-normal from the distribution plot in Figure 2 and it is platykurtic because its excess kurtosis is -1.03 (see table 1). Thus, we should use models that allow for non-normal distributions (e.g. an ARIMA model for B2).

Second, the series is positively skewed, with a skewness of 1.36. From Figure 2, we observe substantial clustering around the mean (23, 308 VND) and the formation of a lower boundary around 22,500 VND. This lower boundary is likely caused by the SBV (i.e. Vietnam's central bank) regularly intervening when the VND strengthens to between 22,000 and 22,500 to ensure that Vietnamese exports remain price-competitive (Kokalari (b), 2022).

The upper range (i.e the relative strength of the USD) is less bounded, with two local modes near 24,250 VND and 24,750 VND. From figure 1, we see the modes come from the appreciation in 2022, which we later show was driven by a surge in demand for USD following a financial crisis. However, the upper bound is limited by the SBV intervening as the VND approaches 25,000 per USD to avoid imports and USD-denominated debt from becoming too expensive (MBS, 2022). Overall, the boundaries reflect Vietnam's managed float between 22,500 and 25,000 VND/USD.

Third, the time-variant mean and variance in Figure 1 indicate the exchange rate series is a unit root, leading us to conduct an Augmented Dickey-Fuller (ADF) test. Using the Python *statsmodels* package, we use the *adfuller()* function to conduct an ADF test and find the un-differenced data has an ADF statistic of -1.659 with a p-value of 0.45. This indicates it is an integrated process, leading us to difference the data and re-run the ADF. The ADF statistic on the differenced data is -8.96 with a p-value of 0, meaning we achieve stationarity and the series was an I(1) process.

Fourth, we observe volatility clustering. Using the *arch* and *statsmodels* packages in Python, we formulate a GARCH(1,1) model for log-returns of the entire series [2]. We choose the (1,1) specification because it has been found to be the best specification

[2]This is slightly skipping ahead, but it is required to fully address the question. We cannot run any GARCH(p,q) on the USD-VND series because it is non-stationary, but we will see in B3 that the log-returns series is stationary. Modelling the GARCH on returns carries evidence of volatility clustering in the exchange rate series.

for exchange rates by Hansen & Lunde (2004).

$$\hat{\sigma}_t^2 = \underset{(7.4\times10^{-9})}{4.5 \times 10^{-7***}} + \underset{(0.018)}{0.10} \times \hat{\epsilon}_{t-1}^{2***} + \underset{(0.009)}{0.88} \times \hat{\sigma}_{t-1}^{2***} \tag{1}$$

Equation (1) shows there is volatility clustering. The statistically significant (1% level) omega value (i.e the intercept) shows there is a non-zero baseline level of volatility. For a given period t, the baseline is augmented through shocks (i.e. the alpha value of 0.1) and volatility from the previous period (i.e. the beta value of 0.88). In particular, at 1% statistical significance, the alpha value shows that past squared returns have a significant influence on current volatility. This means large changes in returns are followed by periods of elevated volatility. Similarly, at 1% statistical significance, the beta value shows there is strong persistence of volatility from the previous period (i.e. 88% of it persists), indicating volatility clustering.

Finally, we observe long-range dependence through serial correlation in the variance of the USD-VND series. We collect the squared residuals of the USD-VND series and use the Ljund-Box test to test for autocorrelation among them. With a Ljund-Box test statistic of 23,444 and p-value of 0, we reject the null of zero autocorrelation in favour of long-range dependence. As such, we will need to select a forecasting model accounting for long-range dependence (e.g. the ARIMA).

Table 2: Comparison to the Relevant Stylized Facts About Financial Data from the Lectures

Facts About Financial Data	Status	Notes
Leptokurtic or platykurtic	✓	Platykurtic (excess kurtosis < 0)
Slight negative skew	×	Positive skew (skewness = 1.36)
Integrated process	✓	I(1) process (ADF test, p = 0)
Volatility clustering	✓	GARCH(1,1) evidence for returns
Long-range dependence	✓	LJ statistic of 23,444 (p = 0)

The features described above and summarized in Table 2 are consistent with other ASEAN emerging market (EM) currency exchange rates with the USD (e.g. USD-THB) (Shen-Li, 2016). Platykurtosis and positive skew can occur due to managed float regimes or capital controls. Volatility clustering may arise from politico-economic uncertainty, stemming from frequent geopolitical tensions in the region or financial crises (see section B4). Long-range dependence is common in exchange rate series, reflecting persistent economic or political trends (e.g. sustained trade surpluses/deficits or systemtic inflation). These factors contribute to creating an integrated process by varying mean and variance across time. Thus, our descriptive conclusions are structurally similar to those reached for peer currencies.

2 B2: Forecasting

We use two models for our ex-ante forecast: the ARIMA and Prophet models. We will evaluate their performance using Root Mean Squared Error, Mean Squared Error, Mean Absolute Percentage Error, and Mean Squared Error.

2.1 ARIMA Forecast

We use the ARIMA for two reasons. First, using its AR component, it can handle the serial correlation identified in B1. To illustrate, for a given AR(1):

$$Y_t = \beta_0 + \beta_1 X_t + \epsilon_t, \text{where } \mathbb{E}(\epsilon_t|\epsilon_1, ..., \epsilon_{t-1}) = \alpha\epsilon_{t-1} \tag{1}$$

We can apply *filtering transformations* below to filter out the serial correlation

$$V_t = Y_t - \alpha Y_{t-1} \tag{2a}$$

$$U_t = X_t - \alpha X_{t-1} \tag{2b}$$

Plugging in (1) into (2a) yields:

$$V_t = \underbrace{(\beta_0 - \alpha\beta_0)}_{\gamma_0} + \beta_1 \underbrace{(X_t - \alpha X_{t-1})}_{U_t} + \underbrace{(\epsilon_t - \alpha\epsilon_{t-1})}_{\epsilon_t^*} \tag{3a}$$

$$V_t = \gamma_0 + \beta_1 U_t + \epsilon_t^* \tag{3b}$$

There is no serial correlation in ϵ_t^*, as $\mathbb{E}(\epsilon_t^*|\epsilon_1^*, ..., \epsilon_{t-1}^*) = 0$ by ϵ_t's properties in (1). The same logic and approach generalizes to p-lags, thus including an AR(p) in the ARIMA controls for the serial correlation observed in B1.

The second reason for the ARIMA is because the MA component can smooth the volatility identified by our GARCH(1,1). In particular, for a given MA(1) process:

$$Y_t = \theta_0 + \epsilon_t + \theta_1\epsilon_{t-1} \tag{4}$$

We can represent the residuals at time $t-1$ as:

$$\epsilon_{t-1} = Y_{t-1} - \theta_0 - \theta_1\epsilon_{t-2} \tag{5}$$

Plugging equation (5) into equation (4) expresses Y_t solely in terms of past Y values and white noise terms:

$$Y_t = \theta_0 + \epsilon_t + \theta_1(Y_{t-1} - \theta_0 - \theta_1\epsilon_{t-2}) \tag{6}$$

Simplifying equation (6) yields:

$$Y_t = \theta_0(1 - \theta_1) + \theta_1 Y_{t-1} + \epsilon_t - \theta_1^2\epsilon_{t-2} \tag{7}$$

Equation (7) shows that current observations are smoothed by incorporating information about past errors, thus mitigating the effect of random fluctuations from one period to the next. Generalizing, a MA(q) process can reduce noise and effectively capture the impact of a shock from previous periods, which can enhance short-term forecast accuracy.

Having justified the model selection, we move to model specification. In B1, we ensured stationarity by differencing the return series once. To identify the amount of lags, we consult the ACF and PACF of the exchange rate series (Figure 3). Determining the lag lengths using Figure 3 was too unreliable, thus we used the *pmdarima* package in Python to find the lag lengths that minimize AIC and BIC. We find that the ARIMA(6,1,5) yields the best fit (i.e. p = 6, d = 1, and q = 5).

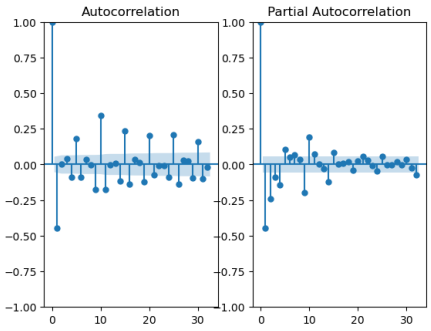


Figure 3: VND-USD ACF and PACF

With this fitted model, we plot the five-day ex-ante forecast for the first five trading days of 2024 (figure 4). The forecast does a poor job at predicting the possible January optimism that often elevates the value of the USD (Ciccone, 2011) and it rather projects the exchange rate series will remain almost unchanged.

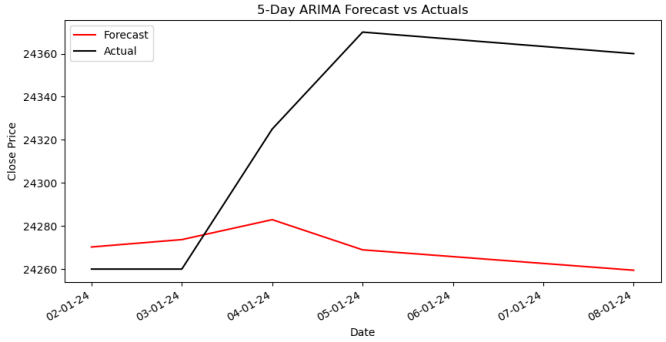


Figure 4: ARIMA (6, 1, 5) Forecast of 2 Jan to 8 Jan 2024 (Jan 6 and Jan 7 are weekend days)

To test if the model is generally poor or it failed on a small sample, we extend it to March 12th 2024. In figure 5, the model still performs poorly, under-predicting the series but capturing it within the 95% confidence interval. We provide a quantitative error assessment in section 2.3, comparing it to the Prophet.

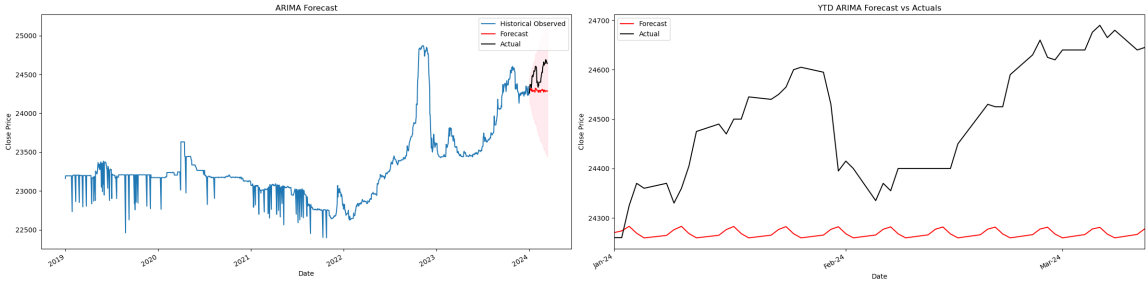


Figure 5: ARIMA(6, 1, 5) Forecast until March 12 2024 (only trading days)

2.2 Prophet Model and Comparison to ARIMA

We also forecast using the Prophet Model, which is Meta's open-source forecasting tool and is widely used by currency traders. We choose it because the model acts as a popular benchmark for model comparison. For simplicity, the model is given by:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{8}$$

$g(t)$ is a piece-wise linear growth trend, $s(t)$ captures seasonality, and $h(t)$ captures the effect of holidays or unlabeled events. Importantly, $g(t)$ exhibits change-point flexibility, which allows it to identify quickly changing trends within a series, like our increasing trend between 2022 and 2024 (figure 1). We also felt that $s(t)$ or $h(t)$ could capture elements like fluctuations from increased demand for Vietnamese products and VND from Western consumers during Christmas (Kokalari, 2023).

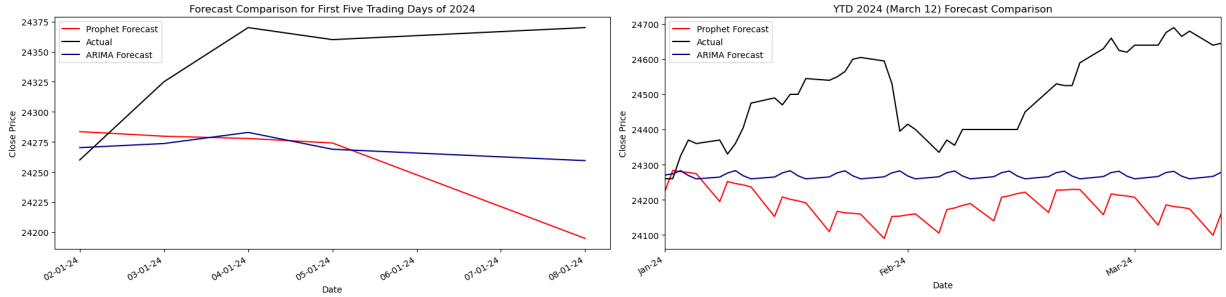


Figure 6: Forecast Comparisons

For the five-day forecast (left panel of Figure 6), the Prophet incorrectly predicts that the exchange rate series will decline, while the ARIMA forecast was relatively flat. This results in the Prophet having larger errors across the forecast horizon, explaining why it performs worse across all error metrics in Table 3.

Table 3: Errors for 5-day Forecast

Model	RMSE	MAE	MAPE	MSE
ARIMA	66.91	53.52	0.22%	4,476.99
Prophet	89.04	70.29	0.29%	7,927.26

For the year-to-date forecast (right panel of Figure 6), the Prophet does much worse and seems to have overfit a weekly oscillation trend (i.e. small ravines for each week). Although the ARIMA also reproduced the ravines, its predictions are consistently closer to the actual values and its oscillations are smoother. Thus, the ARIMA performs better on all metrics even under a ten-times larger forecast horizon (Table 4). Generally, both models replicate the flat trend from 2019 - 2021, failing to account for 2022 - 2024 volatility. Both forecasts are poor and unsatisfactory for a trader, policy-maker or even a company trying to hedge currency risk.

Table 4: Errors for YTD Forecast

Model	RMSE	MAE	MAPE (%)	MSE
ARIMA	252.74	223.09	0.91	63,879.36
Prophet	333.80	302.27	1.23	111,425.10

3 B3: Value at Risk

From the distribution of log-returns, we observe that they are non-normally distributed, strongly leptokurtic (excess kurtosis is 9.2), and roughly symmetric (skewness is 0.09). In particular, there is substantial clustering around the mean of 0% log-returns with two local modes near -1.5% and 1.5%, forming two fat tails.

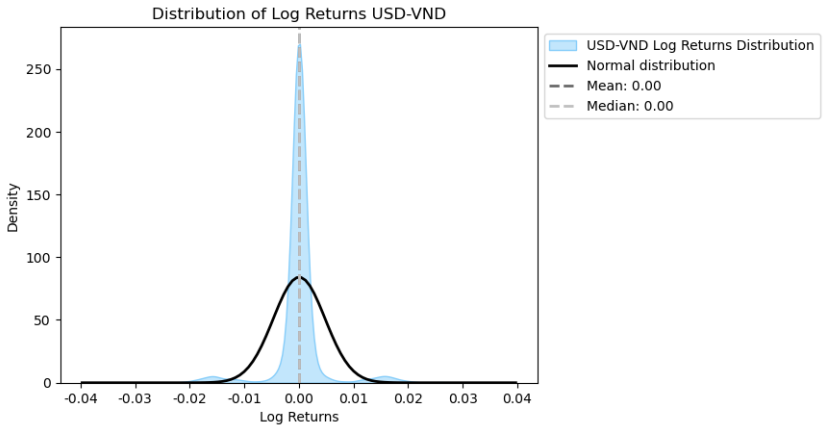


Figure 7: Log Returns Distribution

The distribution's non-normality violates the variance method's assumption of normality. As a result, the variance method may underestimate the tail risk because its assumption of normality does not capture the higher frequency of extreme values that occur in fat-tailed distributions. As the historical approach does not rely on normality, we may consider it more appropraite for our analysis.

Table 5: Summary Statistics of USD-VND (Close Values)

Statistic	**Value**
Mean Value	0.0000
Standard Deviation	0.0047
Excess Kurtosis	9.1957
Skewness	0.0968
Minimum Value	-0.0327
25th Percentile	-0.0002
50th Percentile (Median)	0.0000
75th Percentile	0.0003
Maximum Value	0.0327

In figure 8, we in fact observe the predicted difference between the methods. By historical simulation, we find there is a 1% chance that log-returns will be less than -0.0175 (i.e. -1.735%) on any given day. By the variance method, we estimate a 1% chance of log-retuns being less than -0.0110 (i.e. -1.094%). This results in the series falling below the empirical method estimate about 1% of the time but the variance method 3.61% of the time (see Table 6). This provides further evidence for the variance method being a poor estimation in this case because the distribution is non-normal.

To illustrate, if a hedge fund is considering to hold a long-position of $100 mil. (i.e. betting the USD will appreciate against the VND), on any given day, there is a 1% probability that it would lose $1.1 mil. by the variance approach and $1.7 mil. by historical simulation. For a hedge fund or institutional investor holding a large position, the historical method provides a more conservative and realistic view of the 1% VaR. More generally, the conservative estimate of risk can be preferable if we continue to expect volatility in the future. For instance, as shown in B4, Vietnam's economy is susceptible to political instability, which can trigger large currency movements.

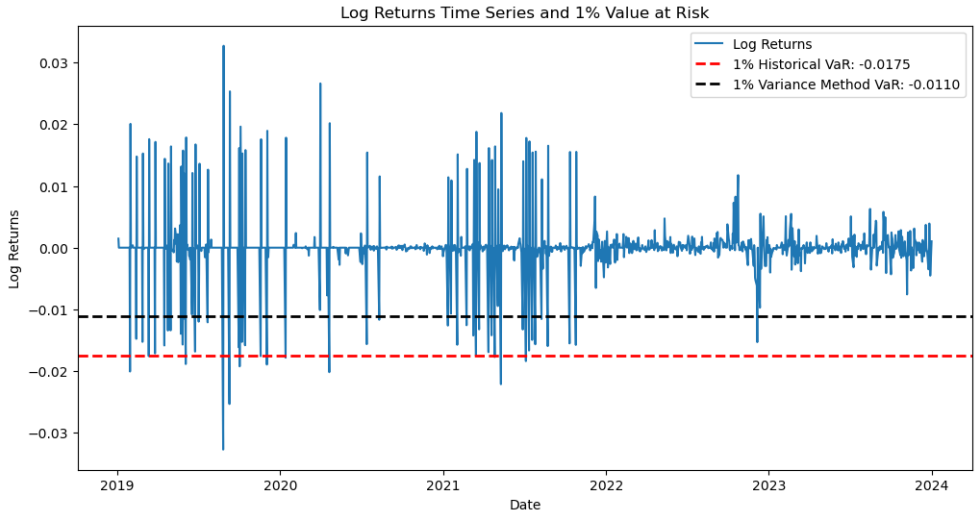


Figure 8: Value at Risk (Log Returns for USD-VND)

Table 6: Violations Count and Percentage

Method	**Violations Count**	**Violations Frequency (%)**
Empirical 1% VaR	14	1.07
Variance Method 1% VaR	47	3.61

4 B4: Event Analysis

Our event occurs on October 10^{th} 2022 - the first trading day after a series of political killings, which triggered a plunge in Vietnamese equity markets and drove domestic demand for USD (Kokalari (a), 2022). This resulted in the largest monthly upward price movement in our series (see Figure 9).

On Saturday of October 8^{th}, Truong My Lan, the chairman of the prominent real estate conglomerate Van Thinh Phat Group, was arrested for "alleged fraud related to the issuance and trading of bonds in 2018-19" (Kokalari (a), 2022). Lan was able to siphon $12.5 bil. through the Saigon Joint Stock Commercial Bank (SCB), in which she held a majority stake (Reuters, 2024). After her arrest, four SCB bank executives were killed, which resulted in viral social media posts and traditional media articles speculating banks will collapse (Phan, 2023). Although unproven, it is believed that Lan killed the executives because they held key evidence of her fraud. Customers of SCB executed a bank run on Monday morning (October 10^{th}) and equity markets responded with fire sales of bank stocks (Phan, 2023).

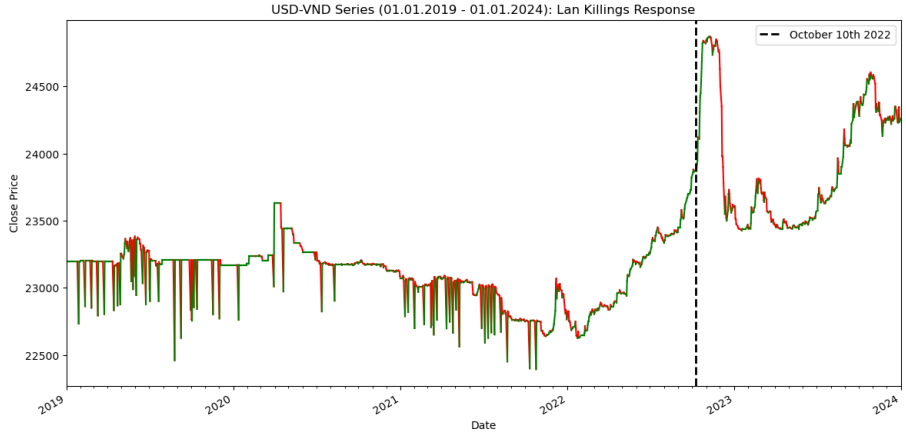


Figure 9: Exchange Rate Series with Lan Killings Impact

The fire sales occurred for two reasons. First, the government released limited information about if the SCB bank run had triggered contagion. As insurance, investors sold a share of their banking stocks shortly after the market opened on Monday, yielding a -3.9% decline in the value of banking stocks within the first hour (Kokalari (c), 2022). Second, the market participants were heavily levered, using banking and real-estate stock as collateral, which lost value as banking stocks plunged. This triggered margin calls, intensifying fire sales. Within two weeks, there was a 40% sell-off of banking stocks and 55% sell-off in real estate stocks (Kokalari (c), 2022). Combined, banking and real-estate stocks represented 50% of the VN Index, which became the worst performing stock market in the world in terms of YTD price development.

The panic spread into the general population in two ways. First, over 90% of market volume is from retail traders (Fortune, 2021), who were also conducting bank-stock fire sales and liquidated also other positions in fear of market collapse (Kokalari (c), 2022). Second, panic spread through sensationalization on social media and elevated speculation of collapse following little information from the government (Phan, 2023).

This could have had two depreciative effects on the VND. First, people influenced by the panic resorted to exchanging VND to USD in the formal and black markets. This resulted in banks and exchanges having to buy a large amount of USD and sell their VND. MBS (2022), a leading broker in Vietnam, argues that this was the leading cause for the depreciation of the VND against USD in October 2022 [3]. Second, freightened foreign investors conducted capital flight, limiting their holdings of VND to protect themselves from ensuing VND depreciation. The currency effects forced the SBV to buy VND with 20 billion USD in November (Kokalari (c), 2023).

Given this, we should expect the USD-VND returns to increase on and after the event. However, there should be little effect prior to the event because the arrests were unexpected and occurred on a weekend. To test this, we measure the cumulative abnormal returns (CAR) 10 trading days before the event, on October 10^{th}, and ten trading days after the event. This period will be referred to as the *event window* and

[3] I worked as an economist in Vietnam at the time and was responsible for producing an event analysis for the Vietnamese prime minister and our investment fund. A very interesting note is that there were immense queues at well-known informal/black market exchanges. The panic was palpable with people screaming and pushing. Unfortunately, I could not find English articles on this.

let everything prior be the *estimation window*. We do not use the dummy variable approach from the textbook because the CAR approach is used more frequently in industry (EventStudy, 2024) and achieves similar results with simpler execution.

Let abnormal returns be given by $A_t = R_t - \bar{R}$, where $\bar{R}$ are average returns in the estimation window. We compute A_t for each day in the event window and find the CAR by summing over the event window, where T is the event day:

$$CAR_{\text{full}} = \sum_{t=T-10}^{T+10} A_t \tag{9a}$$

We collect three CARs: CAR for ten trading days prior to the event (CAR_{prior}), ten trading days after the event (CAR_{after}), and for the entire event window (CAR_{full}) (equation 4a). We formulate the following three sets of two-tailed hypotheses:

$$H_0 : CAR_i = 0 \text{ and } H_1 : CAR_i \neq 0 \text{ where } i \in \{\text{full}, \text{prior}, \text{after}\} \tag{9b}$$

Where n is the amount of sampled days, we test the hypotheses using:

$$t_{CAR_i} = \frac{\text{CAR}_i}{\sqrt{n}\hat{S}_i}, \text{where } n \in \{10, 21\} \text{ and } i \in \{\text{full}, \text{prior}, \text{after}\} \tag{9c}$$

At the five percent level, we find evidence for 3.96% abnormal returns in the 2-week period following the event. For other periods, we fail to reject the null at the 10% level. Interestingly, the CAR for the entire event is statistically insignificant, which could be due to confounding effects (e.g. on October 3, the Federal Reserve signalled a possible interest rate increase). Similarly, it could be due to dilution effects - while there were significant movements after the event, their net effect can appear less significant in the full event window.

Table 7: Cumulative and Abnormal Returns Analysis

Period	CAR (%)	T-Statistic	P-Value
CAR Pre-Event (Sep 26 to Oct 7)	0.68	0.40	0.69
Abnormal Return on Event Day (Oct 10)	0.02	N/A	N/A
CAR Post-Event (Oct 11 to 25)	3.96	2.19	0.03
CAR Entire Event (Sep 23 to Oct 25)	4.68	1.49	0.14

The first conclusion from this is that financial market instability may influence exchange rates if people lose confidence in the domestic currency. This is particularly true in EMs, where the USD is used as a store of value that can become the prevailing currency when the domestic currency fails (Ma and Villar, 2014).

The second conclusion is that intervention from Vietnam's central bank is reflective of both the managed-float regime, but also its monetary prudence. Kokalari (2023) and MBS (2022) find that the depreciation of the VND could have been substantially larger if the SBV had not intervened in the banking system and currency markets.

5 A: Sup Augmented Dickey Fuller Test

An alternate perspective to the explosive behaviour of our event window is that there was a USD bubble in 2022, which resulted in its rapid appreciation against the VND. The first argument in support of this is that the monetary tightening by the Federal Reserve resulted in an appreciation of the USD because it became more attractive to save in the U.S. This argument relies on there being a strong or growing differential between the US and Vietnamese primary interest rates, which was not the case as the SBV closely followed American interest rate hikes (Kokalari (b), 2022). Through its managed float mandate, the SBV pursues this policy to protect the many businesses that have debt denominated in USD and to ensure that essential imports do not become too expensive (MBS, 2023).

Another possible explanation is that investors sought to hold safe-haven assets amidst global economic and political uncertainty. Todorova (2020) finds that the USD, Swiss Frank (CHF), and the Japanese Yen (JPY) exhibit abnormally strong demand during periods of global uncertainty. Given the historical weakening of the JPY in 2022 due to the Bank of Japan refusing to increase interest rates, the USD would be the main safe-haven currency. In studying the capital flows between 2019 and 2023, Thi Thuy (2024) finds that only the USD could be classified as a safe-haven asset. Cheema (2020) argues that if there is only one safe-haven currency, markets may form irrational expectations of continued growth through the crisis, forming a bubble.

The theoretical possibility of a bubble motivates our exploration of the Sup Augmented Dickey Fuller (SADF) test, which we will use to test for the presence of explosive behaviour in 2022. We will first justify our selection of the SADF, discuss the underlying econometrics, and then will apply it to our exchange series. We find evidence of explosive behaviour, which can indicate the presence of a bubble. However, we believe our prior event analysis provides compelling counter-evidence to a bubble and offers a more contextually-relevant explanation of the series' appreciation.

5.1 Justification and Econometrics of the SADF

Standard unit root and cointegration tests are insufficient tools for detecting bubbles because they cannot distinguish between a stationary process and a periodically collapsing bubble series. This is because periodically collapsing bubbles resemble a unit root or stationary auto-regression rather than an explosive series (Phillips et al, 2011). To resolve this, Phillips et al (2011) devise the SADF, which fits the following regression:

$$\Delta y_t = \alpha + \beta y_{t-1} + \sum_{h=1}^{H} \gamma_h \Delta y_{t-h} + \epsilon_t \tag{10}$$

where:

- y_t represents the logarithm of the price exchange series at time t,
- $\Delta y_t = y_t - y_{t-1}$ denotes the first difference of the log price, capturing the asset's return or price change over time,

- α is a constant term, reflecting the drift component of the time series,
- β is the coefficient on the lagged level of the log price y_{t-1}, which is pivotal for testing the presence of a unit root and explosive behavior,
- γ_h are the coefficients of H lagged first differences of the series, Δy_{t-h}, introduced to account for serial correlation in the error term ϵ_t,
- ϵ_t is the error term, assumed to be white noise.

The SADF test focuses on the coefficient β. Whereas the ADF test hypothesizes that $\beta = 0$ (i.e the null is that there is a unit root) or alternately $\beta < 0$ (i.e the series is stationary), the SADF test formulates the following hypotheses:

$$H_0 : \beta \leq 0 \text{ (non-explosive unit root)} \tag{11a}$$

$$H_1 : \beta > 0 \text{ (explosive series)} \tag{11b}$$

Under the SADF framework, the null hypothesis posits a unit root process that is consistent with a random walk and hence no bubble. In contrast, the alternative hypothesis suggests the presence of explosiveness, which may indicate a bubble as the series grows exponentially from its fundamental value.

To conduct the SADF test, a recursive approach is employed. This involves setting a starting point t_0 and an end point t, with t_0 initially set to $t - \tau$, where τ represents the software-determined minimum window size for the test to hold its statistical properties. To run the test (i.e. ensure the sample is large enough), we had to choose the full year 2022 rather than the event window in B4. The end point t is fixed, while the starting point t_0 is expanded backwards, one observation at a time, through the data. This creates a series of overlapping windows that progressively include more data from the past, moving toward the series' beginning. For each time frame, regression (10) is fitted, collecting $\hat{\beta}$ and its t-statistic.

Using this, the SADF test statistic is computed as the maximum t-statistic of $\hat{\beta}$ obtained across all the windows. The selected SADF statistic is compared against the SADF critical values. Rejecting the null hypothesis in favor of the alternative implies that the series exhibits behaviour consistent with the presence of a speculative bubble at some point within the sample period (i.e. 2022 for our analysis).

In particular, the SADF test statistic is given by:

$$SADF_t = \sup_{t_0 \in [1, t-\tau]} \left\{ \frac{\hat{\beta}_{t_0,t}}{\hat{\sigma}_{\hat{\beta}_{t_0,t}}} \right\} \tag{13}$$

Where:

- $\hat{\beta}_{t_0,t}$ is the estimated coefficient from regression (10) over the sample starting at t_0 and ending at t,
- $\hat{\sigma}_{\hat{\beta}_{t_0,t}}$ is the standard error of $\hat{\beta}_{t_0,t}$,
- τ is the minimum window size,

- t_0 represents the starting point of the backward expanding window,
- t is the current endpoint of the window, and
- the supremum (sup) is taken over all possible starting points t_0 within the window $[1, t - \tau]$.

The first major contribution of this method to measuring bubbles is that the recursive approach allows for a dynamic assessment of explosiveness throughout the entire time series. This allows the test to identify periodically collapsing bubble, which can occur in currency series. Secondly, the SADF test does not assume a pre-set number of regime changes or structural breaks, unlike the Bai-Perron Sequential Test (Monschang, 2020). It rather recursively expands the beginning of the sample without presetting the number of potential bubbles, allowing for a more flexible and endogenous detection of speculative bubbles.

In the context of the USD-VND exchange rate, the SADF has two key drawbacks. First, it cannot account for structural breaks or changes in the time series' generating process that are unrelated to bubbles, which can confound the test and result in misleading results. For instance, Vietnam's managed float system means that central bank interventions are frequent in high volatility periods (i.e. potential bubble periods), which the SADF cannot account for. As such, one must be judicious in checking if SADF-reported bubble has proper argumentative grounding or if it could be caused by a government/central bank intervention.

Secondly, it may suffer from the multiple testing problem, which arises when multiple hypothesis tests are conducted simultaneously (e.g. across rolling windows in the SADF case). The more tests are conducted, the higher the chance of encountering at least one false positive (i.e. incorrectly rejecting a true null purely by chance). One could limit this using the Bonferroni correction, which adjusts the significance level by dividing it by the number of tests performed, thereby tightening the criteria for rejecting the null hypothesis and reducing the likelihood of false positives (Armstrong, 2014). However, the correction reduces the test's statistical power (i.e. the likelihood of detecting a true H_1). Thus, while we are less likely to incorrectly identify a bubble, we are also less able to identify a real bubble. We choose to not use the Bonferroni correction because Armstrong (2014) finds that the trade-off is only worthy if there is a near zero tolerance of a type 1 error, which is not our case. This is because we do not solely depend on the SADF test, but evaluate its output by seeing if there exists a credible argument for a bubble and can reason against false positives (see 5.3). With a Bonferroni correction, we would possibly miss some bubbles, which could have catastrophic implications in both trading and policy-making (e.g. for central banks).

5.2 Empirical Investigation: Results

We switch from Python to R in this segment because there was no available package for the SADF in Python. We use the *exuber* package in R to run a SADF test for all the observations in 2022. We chose 2022 as it offered a sufficient sample size and it was the only period for which there seemed to be a credible argument for a bubble (see the chapter 5 introduction and explosive behaviour in Figure 1).

Table 8 reports the SADF statistic and then critical values one would have to exceed

to reject the null. We find, at the 1% level, evidence of explosive behaviour, thus we can reject H_0 in favor of H_1 and we conclude there **might** be a bubble.

Table 8: SADF Test Results

Test	Statistic	90%	95%	99%
SADF	6.02	1.12	1.40	1.95

5.3 Empirical Investigation: Was there a USD bubble?

We argue that the appreciation of the USD-VND was **not** driven by a USD bubble. The first reason is that our event analysis presents a statistically significant and economically reasonable explanation for the largest (i.e. most explosive) monthly increase in the return and exchange rate series. The event analysis is a stronger explanation than a bubble because a bubble fails to explain the precipitous and immediate two-week increase in the exchange rate following the Lan killings in October. The bubble hypothesis would also imply that the bubble burst towards the end of 2022 (see the collpase in Figure 1). But Kokalari (2023) and MBS (2023) find the fall in the exchange rate value in late 2022 and early 2023 was due to substantial central bank purchases of VND using USD. In fact, the Vietnamese foreign reserves fell below the 3-month import threshold [4] at this time (Kokalari (b) , 2022), implying the SBV considered currency stabilization more important than import affordability.

Secondly, beyond the context of Vietnam, little suggests the USD was experiencing a bubble. We run the SADF on the DXY index for 2022, which is a measure of the USD's value relative to its most significant trading partners. If there was a USD bubble, other currencies would depreciate against it and the SADF would signal explosive behavior. We find no evidence for a bubble and fail to reject the null with a SADF statistic of 0.945 (see table 9). This highlights that we must exercise caution in interpreting SADF results and combine them with economic reasoning, as in section B4. Thus, we conclude that our B4 event analysis is a strong explanation of the USD-VND volatility in late 2022.

Table 9: SADF Test Results

Test	Statistic	90%	95%	99%
SADF	0.945	1.11	1.40	1.95

6 Bibliography

[1] 'A Retail Investor Boom Powered Vietnam's Stock Market to All-Time Highs'. Fortune, https://fortune.com/2021/06/11/vietnam-stock-market-boom-retail-investor/.

[2] Armstrong, Richard A. 'When to Use the Bonferroni Correction'. Ophthalmic & Physiological Optics, vol. 34, no. 5, Sept. 2014, pp. 502–08. PubMed

[3] Cheema, Muhammad. 'COVID-19 and Safe Haven Assets'. CEPR, 25 July 2020.

[4] The three-month import threshold is the amount of foreign reserves (namely USD) a country needs to afford 3 months of imports.

[4] Ciccone, Stephen. 'Investor Optimism, False Hopes and the January Effect'. Journal of Behavioral Finance. https://doi.org/10.1080/15427560.2011.602197.

[5] Dollar Edges Lower but Poised for Weekly Gain as Early Rate Cut Hopes Dim — Reuters. https://www.reuters.com/dollar-near-one-month-peak-2024-01-18/.

[6] 'Event Study - AR and CAR Test Statistics'. Event Study, https://eventstudy.de/statistics/ar_car_statistics.html.

[7] Explosive Behaviour in the 1990s NASDAQ: When Did Exuberance Escalate Asset Values? https://www-jstor-org.ezproxy.is.ed.ac.uk/stable/23016628.

[8] Hansen, Peter. 'A Forecast Comparison of Volatility Models: Does Anything Beat a GARCH(1,1)?'.Journal of Applied Econometrics. https://onlinelibrary.wiley.com/doi/full/10.1002/jae.800

[9] Kokalari, Michael [a]. 'Comments on SCB', October 2022. https://vinacapital.com/wp-content//10/Comments-On-Saigon-Commercial-Bank-SCB-and-Van-Thinh-Phat-VTP.pdf

[10] Kokalari, Michael [b]. 'Q3 Update', October, 2022. https://vinacapital.com/wp-content/uploads/2022/10/VinaCapital-SBV-Raised-Interest-Rates-to-Protect-VND.pdf

[11] Kokalari, Michael [c]. 'Credit Crunch', November, 2022. https://vinacapital.com/wp-content/uploads/2022/11/Credit-Crunch-Concerns-Could-Ease-Soon.pdf

[12] Kokalari, Michael. 'Looking ahead into 2023', Jan, 2023. https://vinacapital.com/wp-content/uploads/2023/01/VinaCapital-Insights-Looking-Ahead-at-2023.pdf

[13] Le Thi Thuy, Van, et al. 'The Roles of Gold, US Dollar, and Bitcoin as Safe-Haven Assets in Times of Crisis'. Cogent Economics & Finance, vol. 12, no. 1, Dec. 2024, p. 2322876. Taylor and Francis+NEJM, https://doi.org/10.1080/23322039.2024.2322876.

[14] Li, Wei-Shen, et al. 'General and Specific Statistical Properties of Foreign Exchange Markets during a Financial Crash'. Physica A: Statistical Mechanics and Its Applications, vol. 451, Feb. 2016. ResearchGate, https://doi.org/10.1016/2016.01.077.

[15] Ma, Guonan, and Agustin Villar. Internationalisation of EM Currencies. no. 78.

[16] MBS. 'Báo cáo vietnam outlook oct-2022. October, 2022.

[17] MBS. 'Báo cáo vietnam outlook jan-2023. January, 2023.

[18] Monschang, Verena, and Bernd Wilfling. 'Sup-ADF-Style Bubble-Detection Methods under Test'. Empirical Economics. https://doi.org/10.1007/s00181-020

[19] Phan, Hong Mai, et al. 'Herd Behavior in Vietnam's Stock Market: Impacts of COVID-19'. Cogent Economics & Finance, vol. 11, no. 2, Oct. 2023, p. 2266616. DOI.org (Crossref), https://doi.org/10.1080/23322039.2023.2266616.

[20] Todorova, Vessela. 'Safe Haven Currencies'. Economic Alternatives, no. 4, 2020.

[21] Trial Begins in Vietnam's Largest, Multi-Billion-Dollar Financial Fraud — Reuters. https://www.reuters.com/world/asia-pacific/trial-begins-vietnams-largest-multi-billion-dollar-financial-scam-2024-03-05/.

7 Code Script for Part A

The code script is also available on Github. For an easy-to-navigate file, please refer to Github. I am only appending the code just in case the link does not work.

Question 1 APE

Code

Loading the required pacakages.

Hide

```
library(exuber)
library(readxl)
```

Loading the data and conducting the SADF test for 2022 USD-VND series.

Hide

```
# Loading and formatting the data
data <- read_excel("APEData.xlsx")
data$Date <- as.Date(data$Date)

# Selecting a subset of data
start_date <- as.Date('2022-01-01')
end_date <- as.Date('2023-01-01')
subset_data <- subset(data, Date >= start_date & Date <= end_date)

# Converting close to returns
close_series <- as.numeric(subset_data$Close)
log_close_series <- log(close_series)

# Performing the SADF test using the radf function on the log-transformed series
sadf_result <- radf(log_close_series)

# Viewing the summary of the test results
summary(sadf_result)
```

```
Using `radf_crit` for `cv`.
```

```
── Summary (minw = 31, lag = 0) ──────────────────────────────────────────── Monte Carlo (nrep = 200
0) ──

series1 :
```

stat <fctr>	tstat <dbl>	90 <dbl>	95 <dbl>	99 <dbl>
adf	-0.9070547	-0.4559089	-0.0990145	0.584998
sadf	6.0223487	1.1161360	1.4030080	1.950243
gsadf	8.4853598	1.8788950	2.1130156	2.560539

3 rows

```
NA
```

Conducting theSADF Test for the DXY 2022 series

Hide

```
df <- read_excel("DXY.xlsx")
close_series2 <- as.numeric(df$Close)
log_close_series2 <- log(close_series2)
sadf_result2 <- radf(log_close_series2)
summary(sadf_result2)
```

```
Using `radf_crit` for `cv`.
```

```
── Summary (minw = 31, lag = 0) ──────────────────────────────────────────── Monte Carlo (nrep = 200
0) ──

series1 :
```

stat <fctr>	tstat <dbl>	90 <dbl>	95 <dbl>	99 <dbl>
adf	-0.3394954	-0.4408594	-0.1387046	0.5466869
sadf	0.9453484	1.1107955	1.4003163	1.9502425
gsadf	1.0945471	1.8788950	2.1130156	2.5605388

3 rows

```
NA
```

8 Code Script for Part B

The code script is also available on Github. For an easy-to-navigate file, please refer to Github. I am only appending the code just in case the link does not work.

Question 2 Set-Up

We load in our packages and data.

In [1]:

```
# Packages for numercical manipulation

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import seaborn as sns
from scipy.stats import norm
from scipy import stats

# Packages for forecasting and graphing

import plotly.graph_objects as go
import plotly.offline as po
po.init_notebook_mode(connected=True)
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from prophet import Prophet
from pylab import rcParams
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
import statsmodels
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.dates as mdates
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
from mplfinance.original_flavor import candlestick2_ohlc
import matplotlib.ticker as ticker
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error, mean_absolute_error
import pymc as pm
from pandas.tseries.holiday import USFederalHolidayCalendar
from pandas.tseries.offsets import CustomBusinessDay
import pmdarima as pm
from pmdarima import model_selection
```

```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_excel("APEData.xlsx")
Close = data['Close'].dropna()
Time = data['Date'].dropna()
```

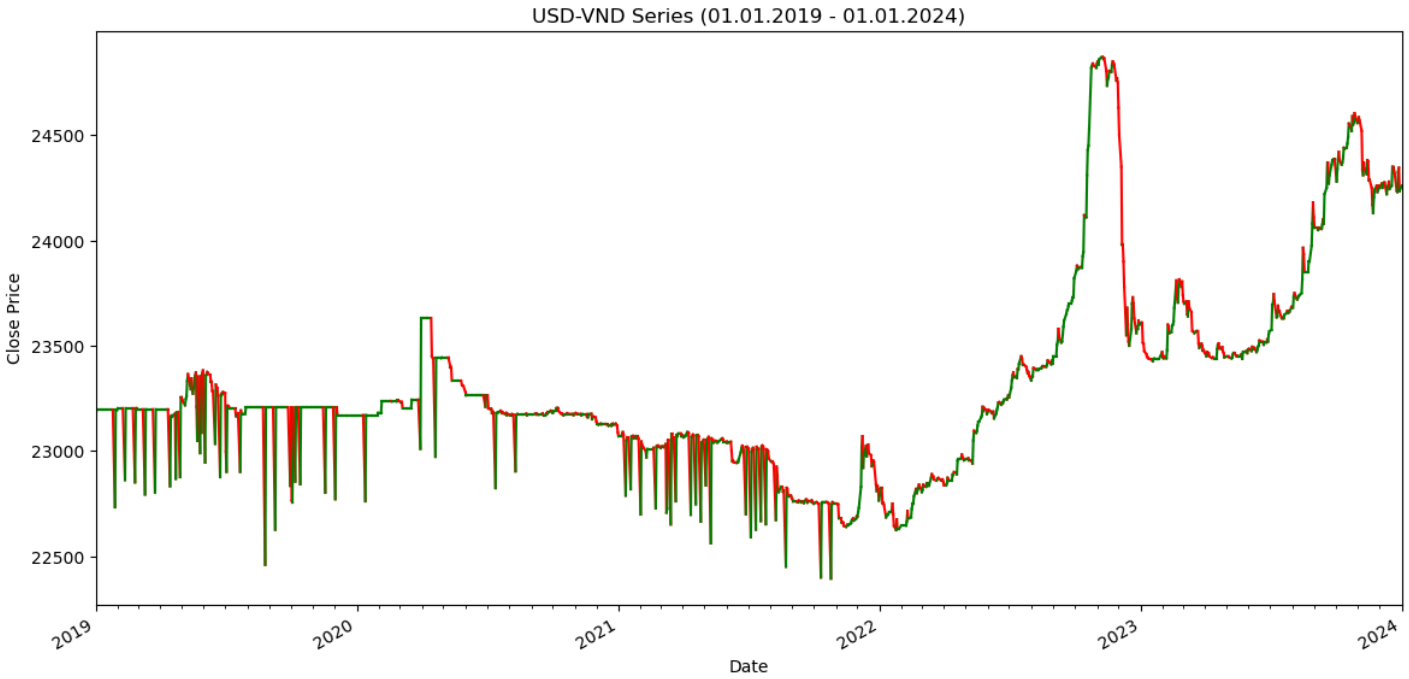
In [3]:
```
pd.options.display.float_format = '{:,.2f}'.format
pd.set_option('display.max_columns', None)  # or 1000
pd.set_option('display.max_rows', None)  # or 1000
pd.set_option('display.max_colwidth', None)  # or 199
```

Question 2a

We plot the exchange rate series below.

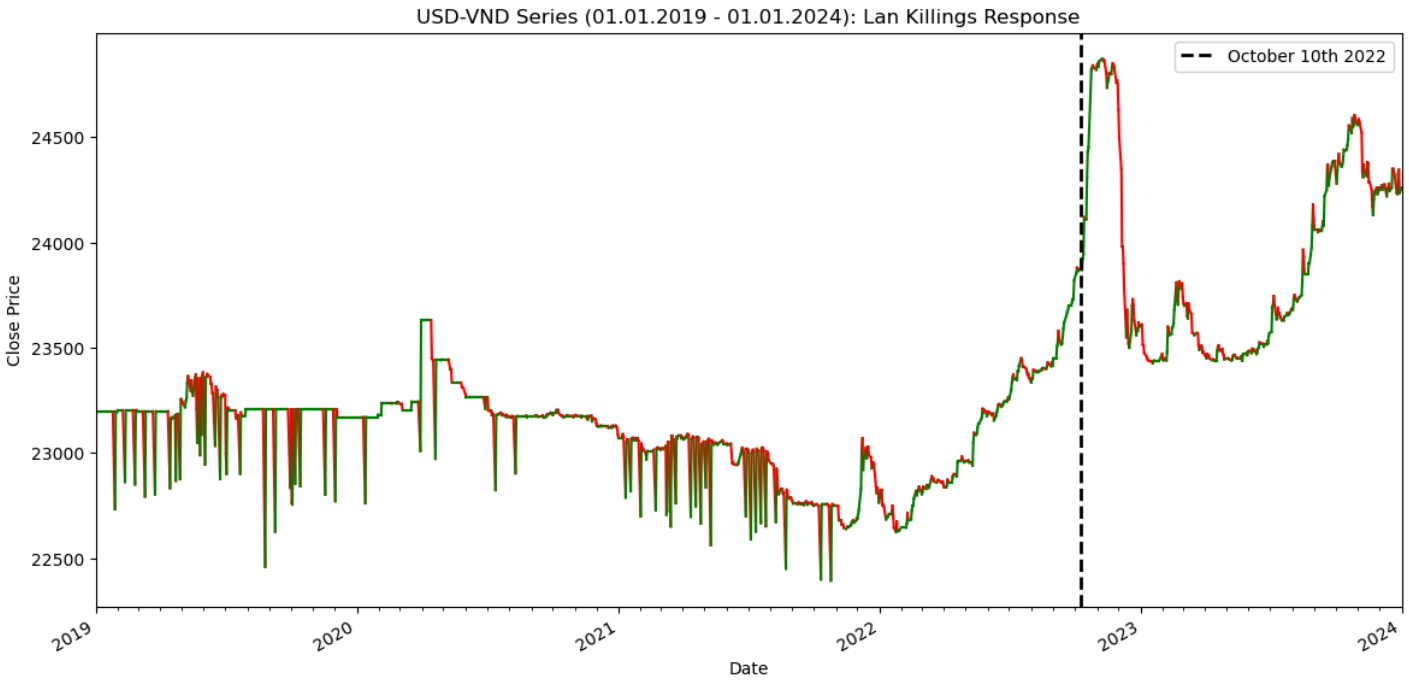
In [4]:
```
# Correcting the datetime conversion for 'Date' column
data['Date'] = pd.to_datetime(data['Date'])

# Assuming the 'Open' is the previous 'Close'
opens = data['Close'].shift(1).fillna(data['Close'][0])

# Since 'High' and 'Low' are not provided, we will use 'Open' and 'Close' for them
highs = lows = opens

data['Close'] = data['Close'].interpolate()

# Creating the figure and axis with a larger size for better visibility
fig, ax = plt.subplots(figsize=(14, 7))

typical_gap = pd.Timedelta(days=1)

# Plotting each day as a line, skipping gaps larger than the typical trading gap
for i in range(1, len(data['Date'])):
    # Calculate the gap in days between subsequent trading days
    gap = (data['Date'].iloc[i] - data['Date'].iloc[i-1]).days
    if gap <= 3:  # assuming 3 to account for weekends
        # Determine the color based on up or down day
        color = 'g' if data['Close'].iloc[i] >= opens.iloc[i] else 'r'
        ax.plot([data['Date'].iloc[i-1], data['Date'].iloc[i]], [data['Close'].iloc[i-1], data['Close'].iloc[i]], color=color)
    else:
```

```
        # Do not connect the line across the gap
        continue


# Setting major ticks format for every year and minor ticks for every month
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_minor_locator(mdates.MonthLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

# Filling gaps by making sure that weekends and holidays are not plotted
# This is done by using only the dates present in the data for the x-axis ticks
ax.set_xlim([data['Date'].min(), data['Date'].max()])

# Rotating dates for better visibility
fig.autofmt_xdate()

# Adding labels and title
ax.set_xlabel('Date')
ax.set_ylabel('Close Price')
ax.set_title('USD-VND Series (01.01.2019 - 01.01.2024)')

# Showing the plot
plt.show()
```



In [5]:
```
# Correcting the datetime conversion for 'Date' column
data['Date'] = pd.to_datetime(data['Date'])

# Assuming the 'Open' is the previous 'Close'
opens = data['Close'].shift(1).fillna(data['Close'].iloc[0])

# Since 'High' and 'Low' are not provided, we will use 'Open' and 'Close' for them
highs = lows = opens

data['Close'] = data['Close'].interpolate()

# Creating the figure and axis with a larger size for better visibility
fig, ax = plt.subplots(figsize=(14, 7))

# Plotting each day as a line, skipping gaps larger than the typical trading gap
```

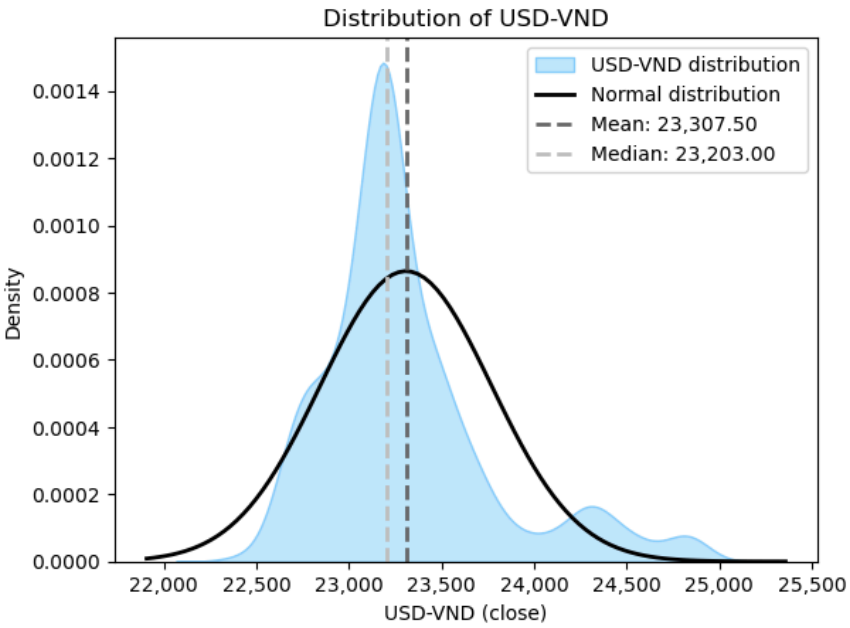
```
for i in range(1, len(data['Date'])):
    # Calculate the gap in days between subsequent trading days
    gap = (data['Date'].iloc[i] - data['Date'].iloc[i-1]).days
    if gap <= 3:  # assuming 3 to account for weekends
        # Determine the color based on up or down day
        color = 'g' if data['Close'].iloc[i] >= opens.iloc[i] else 'r'
        ax.plot([data['Date'].iloc[i-1], data['Date'].iloc[i]], [data['Close'].iloc[i-1], data['Close'].iloc[i]], color=color)

# Highlight October 10th
plt.axvline(x=pd.to_datetime('2022-10-10'), color='black', linestyle='--', linewidth=2, label='October 10th 2022')

# Setting major ticks format for every year and minor ticks for every month
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_minor_locator(mdates.MonthLocator(bymonthday=1))
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

# Adjusting x-axis limits if necessary
ax.set_xlim(data['Date'].min(), data['Date'].max())

# Rotating dates for better visibility
fig.autofmt_xdate()

# Adding labels and title
ax.set_xlabel('Date')
ax.set_ylabel('Close Price')
ax.set_title('USD-VND Series (01.01.2019 - 01.01.2024): Lan Killings Response')

plt.legend()
# Showing the plot
plt.show()



### analysis is then --> we saw this big jump, we investigated
```



We plot the USD-VND distribution

In [6]:
```
# Recalculating the mean, median, and standard deviation
mean_close = np.mean(Close)
median_close = np.median(Close)
std_close = np.std(Close)

# Convert mean and median to string with commas for thousands
mean_close_str = "{:,.2f}".format(mean_close)
median_close_str = "{:,.2f}".format(median_close)

# Initialize a matplotlib figure
fig, ax = plt.subplots()
```

```python
# Create a Kernel Density Estimate plot for the 'Close' column
sns.kdeplot(Close, ax=ax, color="lightskyblue", label="USD-VND distribution", alpha=0.5, fill=True)

# Normal distribution overlay
xmin, xmax = ax.get_xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mean_close, std_close)
ax.plot(x, p, 'k', linewidth=2, label="Normal distribution")

# Vertical line for the mean with label including comma for thousands
ax.axvline(mean_close, color='dimgray', linestyle='dashed', linewidth=2, label=f'Mean: {mean_close_str}')
ax.axvline(median_close, color='silver', linestyle='dashed', linewidth=2, label=f'Median: {median_close_str}')

# Set labels and title
ax.set_ylabel('Density')
ax.set_xlabel('USD-VND (close)')
ax.legend()
ax.set_title("Distribution of USD-VND")

# Format x-axis with commas for thousands
ax.get_xaxis().set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:,}".format(int(x))))

# Show the plot
plt.show()
```



We create a table for the statistical properties to complement the distribution above

In [7]:
```
Percentiles = [0.05, 0.25, 0.5, 0.75, 0.95]
sum_stats = data[['Close']].describe(percentiles=Percentiles)

# Calculating kurtosis and excess kurtosis
kurtosis = data['Close'].kurtosis()
excess_kurtosis = kurtosis - 3
sum_stats.loc['Excess Kurtosis'] = excess_kurtosis

# Calculating skewness
skewness = data['Close'].skew()
sum_stats.loc['Skewness'] = skewness

# Creating a dictionary to map the old index names to the new ones
new_index_labels = {
```

```
    'count': "Amount of Observations",
    'mean': "Mean Value",
    'std': "Standard Deviation",
    'min': "Minimum Value",
    '5%': "5th Percentile",
    '25%': "25th Percentile",
    '50%': "50th Percentile",  # Median
    '75%': "75th Percentile",
    '95%': "95th Percentile",
    'max': "Maximum Value",
    'Excess Kurtosis': "Excess Kurtosis",
    'Skewness': "Skewness"
}

new_column_labels = {
    'Close': "Exchange Rate (Close)"
}

# Renaming the index and columns using the dictionaries
sum_stats_renamed = sum_stats.rename(columns=new_column_labels, index=new_index_labels)

# Displaying the updated summary statistics table
print(sum_stats_renamed)
```

```
                       Exchange Rate (Close)
Amount of Observations              1,305.00
Mean Value                         23,307.43
Standard Deviation                    461.90
Minimum Value                      22,394.35
5th Percentile                     22,730.64
25th Percentile                    23,050.00
50th Percentile                    23,203.00
75th Percentile                    23,450.00
95th Percentile                    24,360.00
Maximum Value                      24,871.00
Excess Kurtosis                        -1.03
Skewness                                1.36
```

Question 2b: Ex Ante Forecasating

We first conduct testing on the Prophet model

Importing the Prophet Model from Meta's opensource Prophet package.

In [8]:
```
from prophet import Prophet
from prophet.plot import plot_plotly
import plotly.offline as py
py.init_notebook_mode()
```

Conducting the Prophet forecast for 5 days:

In [9]:
```
# Converting columns to fit the Prophet model
data.rename(columns={'Date': 'ds', 'Close': 'y'}, inplace=True)

# Sorting values in ascending order for the Prophet model
data = data.sort_values('ds')

# Fitting the model
prophet_model = Prophet(interval_width=0.95)
prophet_model.fit(data)

# Making a DataFrame to forecast future dates, starting after the last date in historical data
last_historical_date = data['ds'].max()
future_dates = prophet_model.make_future_dataframe(periods=5, freq='D', include_history=False)
future_dates = future_dates[future_dates['ds'] > last_historical_date]  # Ensure dates start after the last historical date

# Forecasting the future dates
forecast = prophet_model.predict(future_dates)

# Renaming the forecast columns for clarity
forecast.rename(columns={
    'ds': 'Date',
    'yhat': 'Forecast',
    'yhat_lower': '5th percentile estimate',
    'yhat_upper': '95th percentile estimate'
}, inplace=True)


forecast[['Date', 'Forecast', '5th percentile estimate', '95th percentile estimate']].head()
```

```
17:43:47 - cmdstanpy - INFO - Chain [1] start processing
17:43:47 - cmdstanpy - INFO - Chain [1] done processing
```

Out[9]:

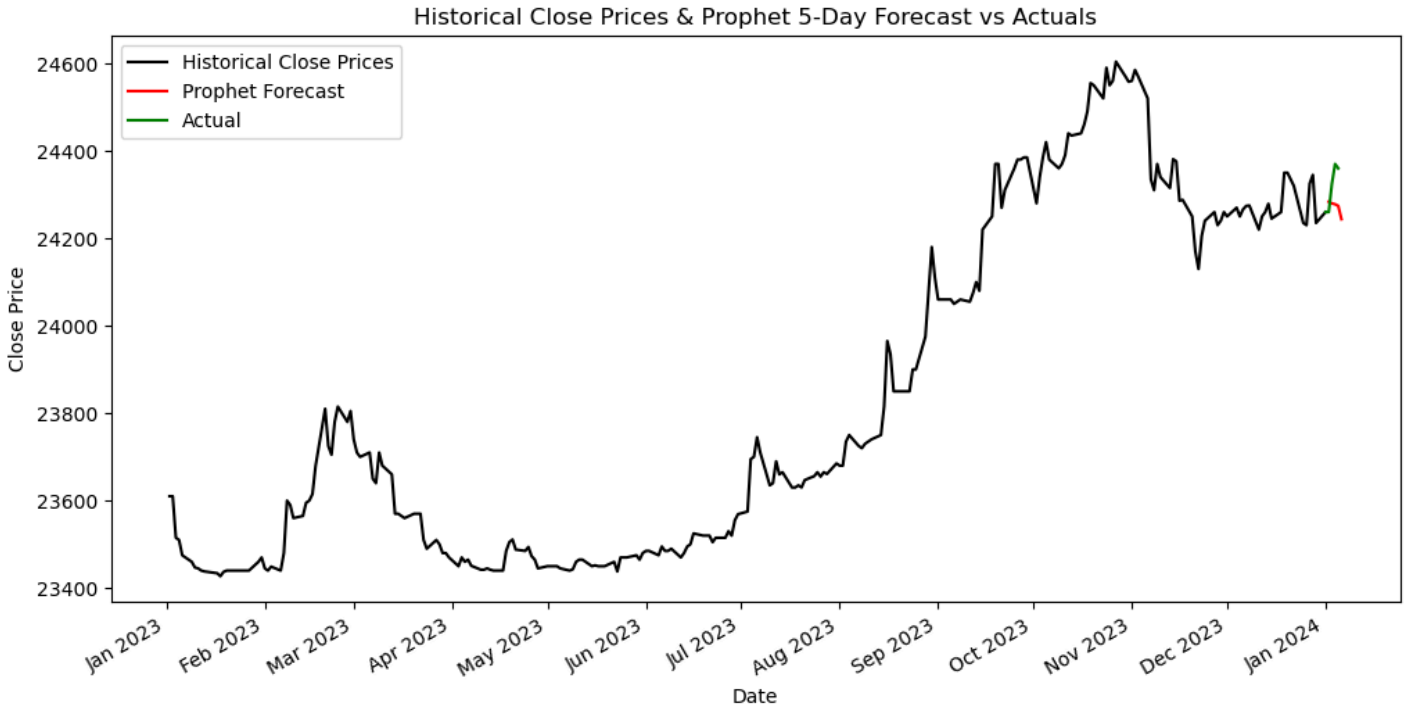
	Date	Forecast	5th percentile estimate	95th percentile estimate
0	2024-01-02	24,283.57	24,058.30	24,489.16
1	2024-01-03	24,279.74	24,050.13	24,496.84
2	2024-01-04	24,277.88	24,056.51	24,515.30
3	2024-01-05	24,274.13	24,062.81	24,477.82
4	2024-01-06	24,243.80	24,035.13	24,467.45

Comparing the forecast to actual values within the context of 2023:

In [10]:

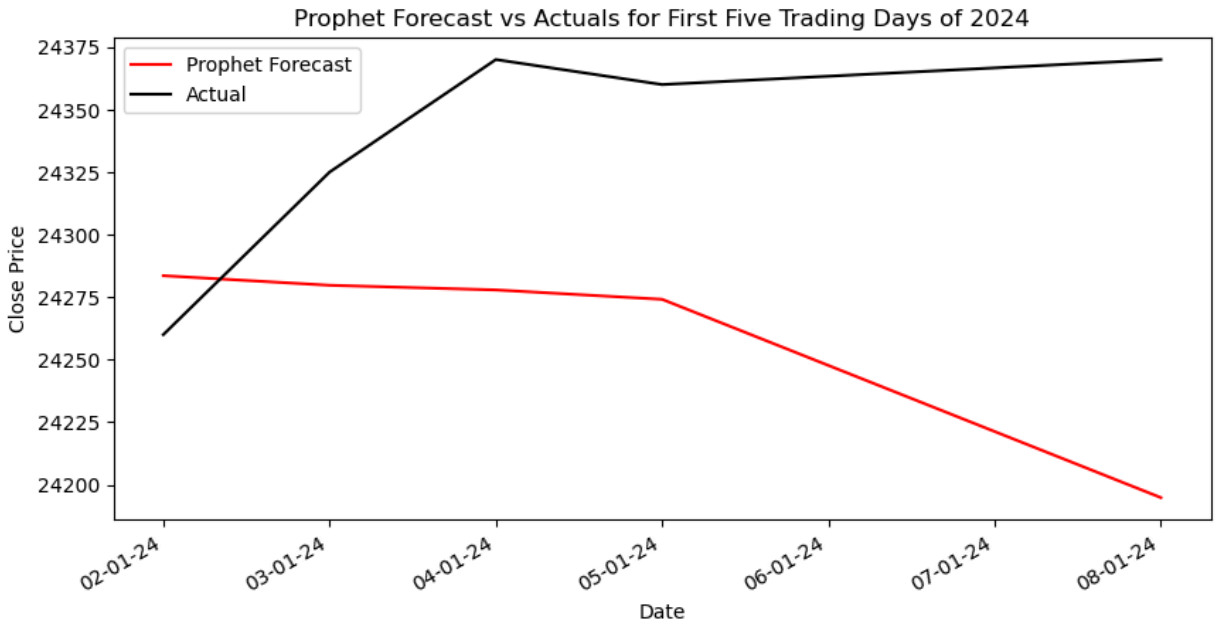
```
# Actual values to compare against
actual_values = [24260.0000, 24260.0000, 24325.0000, 24370.0000, 24360.0000]
date_range_actual = pd.date_range(start='2024-01-01', periods=len(actual_values), freq='D')
actual_series_2024 = pd.Series(actual_values, index=date_range_actual)


# Filter the historical data for plotting (assuming the original dataset is stored in `original_data`)
data['ds'] = pd.to_datetime(data['ds'])
historical_data_2023 = data[(data['ds'] >= '2023-01-01') & (data['ds'] <= '2024-01-01')]

forecast_2024 = forecast[forecast['Date'] >= '2024-01-01']
forecast_2024 = forecast_2024[['Date', 'Forecast']].copy()

# Plotting the results
plt.figure(figsize=(12, 6))

# Plot the historical data from 2023
plt.plot(historical_data_2023['ds'], historical_data_2023['y'], color='black', label='Historical Close Prices')

# Plot the forecasted values from Prophet for January 2024
plt.plot(forecast_2024['Date'], forecast_2024['Forecast'], color='red', label='Prophet Forecast')

# Plot the actual values for January 2024
plt.plot(date_range_actual, actual_series_2024, color='green', label='Actual')

plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Historical Close Prices & Prophet 5-Day Forecast vs Actuals')
plt.legend()

# Format the dates on the x-axis to show only the date part
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
```

```
plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
plt.gcf().autofmt_xdate() # Automatic rotation of date labels to improve fit

plt.show()
```



In [11]:
```
trading_days = ['2024-01-02', '2024-01-03', '2024-01-04', '2024-01-05', '2024-01-08']
future_dates = pd.DataFrame({'ds': pd.to_datetime(trading_days)})

# Forecasting for the specified trading days
forecast_5day_prophet = prophet_model.predict(future_dates)

actual_values = [24260.0000, 24325.0000, 24370.0000, 24360.0000, 24370.0000]
actual_series_2024 = pd.Series(actual_values, index=pd.to_datetime(trading_days))
```

```
# Plotting the results
plt.figure(figsize=(10, 5))

# Plot the forecasted values from Prophet for the specified trading days
plt.plot(forecast_5day_prophet['ds'], forecast_5day_prophet['yhat'], color='red', label='Prophet Forecast')

# Plot the actual values for the corresponding trading days
plt.plot(actual_series_2024.index, actual_series_2024.values, color='black', label='Actual')

plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Prophet Forecast vs Actuals for First Five Trading Days of 2024')
plt.legend()

# Format the dates on the x-axis to show only the date part
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%y'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator())
plt.gcf().autofmt_xdate() # Auto-rotate date labels for better readability

plt.show()
```



We extend the testing samplem to March 12th (YTD at the time of coding). We load in the data below.

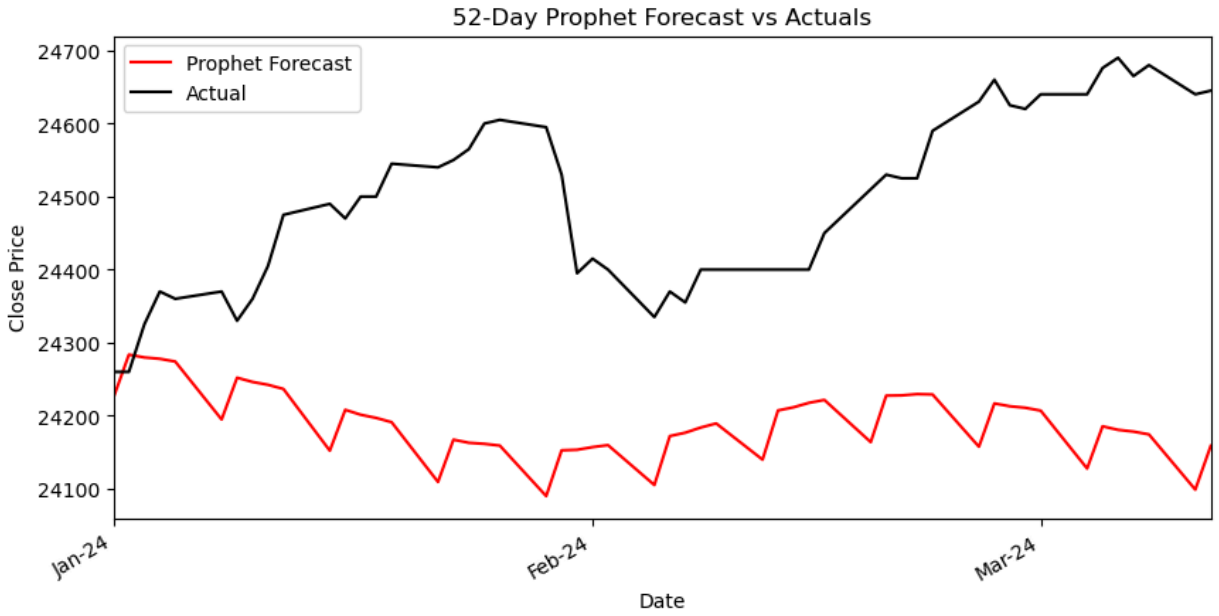
In [12]:
```
## Extending the forecast for YTD values - this will gives us a better sample
data2 = pd.read_excel("YTD VND.xlsx")
Close_YTD_2024 = data2['Close'].dropna()
Time_YTD_2024 = data2['Date'].dropna()
actual_2024_df = pd.DataFrame({'ds': Time_YTD_2024, 'y': Close_YTD_2024})
data['ds'] = pd.to_datetime(data['ds'])
```

Creating the YTD Prophet forecast.

In [13]:
```
# Define the forecasting range
start_forecast_date = '2024-01-01'
date_range_forecast = pd.date_range(start=start_forecast_date, periods=52, freq='B')

# Create a DataFrame for Prophet forecast (business days only)
```

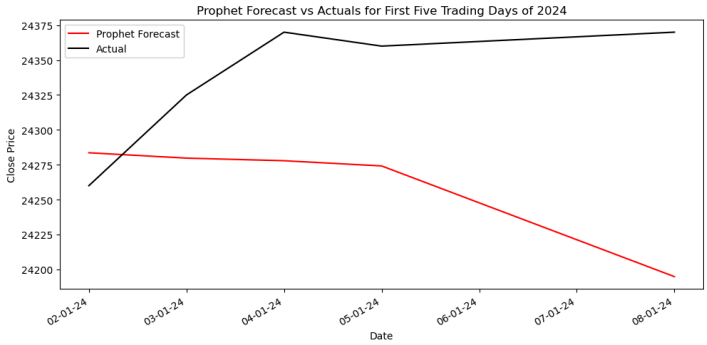
```
future_dates_df = pd.DataFrame(date_range_forecast, columns=['ds'])

# Generate the forecast for the specified dates
forecast_prophet_YTD_2024 = prophet_model.predict(future_dates_df)

# Extract the predicted mean (yhat)
predicted_mean_YTD_2024_prophet = forecast_prophet_YTD_2024['yhat'].values

actual_values = [24260.0000, 24260.0000, 24325.0000, 24370.0000, 24360.0000,
    24370.0000, 24330.0000, 24360.0000, 24405.0000, 24475.0000,
    24490.0000, 24470.0000, 24500.0000, 24500.0000, 24545.0000,
    24540.0000, 24550.0000, 24565.0000, 24600.0000, 24605.0000,
    24595.0000, 24530.0000, 24395.0000, 24415.0000, 24400.0000,
    24335.0000, 24370.0000, 24355.0000, 24400.0000, 24400.0000,
    24400.0000, 24400.0000, 24400.0000, 24400.0000, 24450.0000,
    24510.0000, 24530.0000, 24525.0000, 24525.0000, 24590.0000,
    24630.0000, 24660.0000, 24625.0000, 24620.0000, 24640.0000,
    24640.0000, 24676.0000, 24690.0000, 24665.0000, 24680.0000,
    24640.0000, 24645.0000
]

# Assuming the actual dates align with the business day forecast period
date_range_actual = date_range_forecast
actual_series = pd.Series(actual_values, index=date_range_actual)

# Plotting setup
locator = mdates.MonthLocator()
formatter = mdates.DateFormatter('%b-%y')

plt.figure(figsize=(10, 5))

# Plot the forecasted values from Prophet
plt.plot(date_range_forecast, predicted_mean_YTD_2024_prophet, color='red', label='Prophet Forecast')

# Plot the actual values
plt.plot(date_range_actual, actual_series, color='black', label='Actual')

plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('52-Day Prophet Forecast vs Actuals')
plt.legend()

# Set the locator and formatter for the x-axis
plt.gca().xaxis.set_major_locator(locator)
plt.gca().xaxis.set_major_formatter(formatter)
```

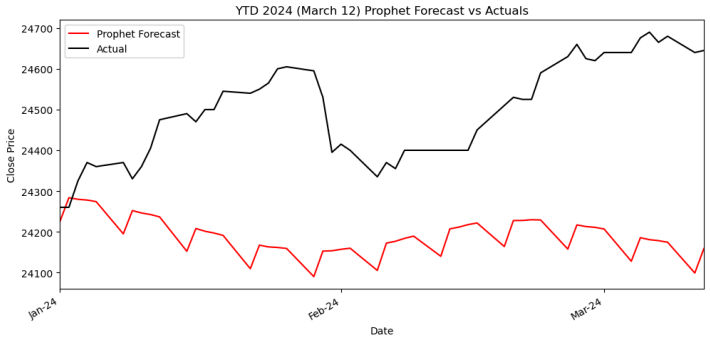
```
# Ensure the plot only shows the dates within our forecast range
plt.xlim([date_range_forecast[0], date_range_forecast[-1]])

plt.gcf().autofmt_xdate() # Auto-rotate date labels

plt.show()
```



In [14]:
```
# Prepare for side-by-side plotting

fig, ax = plt.subplots(1, 2, figsize=(20, 5))

# Plot 1: Prophet Forecast vs Actuals for First Five Trading Days of 2024
ax[0].plot(forecast_5day_prophet['ds'], forecast_5day_prophet['yhat'], color='red', label='Prophet Forecast')
ax[0].plot(actual_series_2024.index, actual_series_2024.values, color='black', label='Actual')
ax[0].set_xlabel('Date')
ax[0].set_ylabel('Close Price')
ax[0].set_title('Prophet Forecast vs Actuals for First Five Trading Days of 2024')
ax[0].legend()
```

```
ax[0].xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%y'))
ax[0].xaxis.set_major_locator(mdates.DayLocator())
fig.autofmt_xdate()  # Auto-rotate date labels for better readability

# Plot 2: 52-Day Prophet Forecast vs Actuals
ax[1].plot(date_range_forecast, predicted_mean_YTD_2024_prophet, color='red', label='Prophet Forecast')
ax[1].plot(date_range_actual, actual_series, color='black', label='Actual')
ax[1].set_xlabel('Date')
ax[1].set_ylabel('Close Price')
ax[1].set_title('YTD 2024 (March 12) Prophet Forecast vs Actuals')
ax[1].legend()
ax[1].xaxis.set_major_locator(locator)
ax[1].xaxis.set_major_formatter(formatter)
ax[1].set_xlim([date_range_forecast[0], date_range_forecast[-1]])

plt.tight_layout()
plt.show()
```





We move to the ARIMA Model

We test for stationarity to identify how many times we should be differencing - we see that we only need to difference once.

In [15]:
```
# Checking for stationarity
data = pd.read_excel("APEData.xlsx")
Close = data['Close'].dropna()
Time = data['Date'].dropna()

result = adfuller(Close)
```

```
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

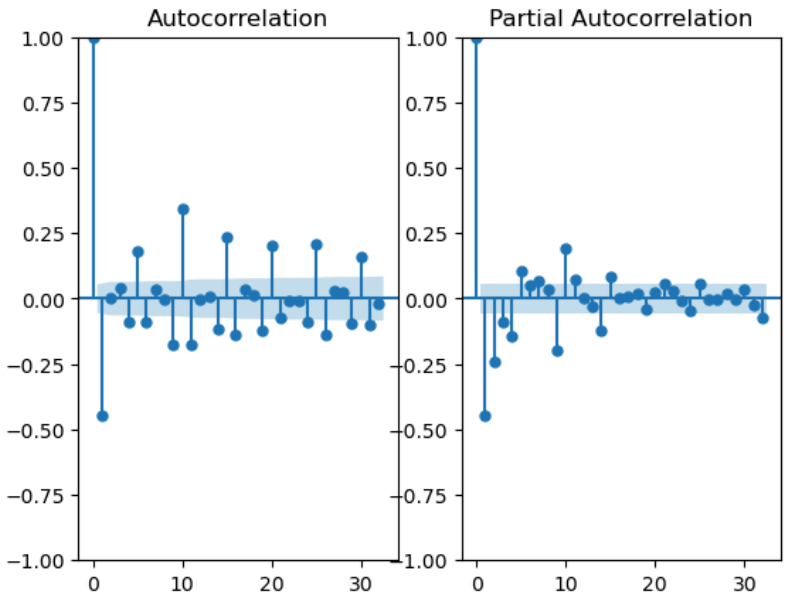
```
ADF Statistic: -1.659199
p-value: 0.452247
```

In [16]:
```
# Difference the data
data_diff = Close.diff().dropna()

# Perform the ADF test again on the differenced data
result_diff = adfuller(data_diff)
print('ADF Statistic: %f' % result_diff[0])
print('p-value: %f' % result_diff[1])
```

```
ADF Statistic: -8.965738
p-value: 0.000000
```

We will use the differenced data that is now stationary. The next step is to identify the number of lags - we use the ACF and PACF.

In [17]:
```
## Running ACF and PACF to identify the amount of p-lags

fig, ax = plt.subplots(1,2)
plot_acf(data_diff, ax=ax[0])
plot_pacf(data_diff, ax=ax[1])

plt.show()
```



The ACF and PACF are quite unclear, so we use the auto_arima function to identify the optimal model specification. We find that the ARIMA(6,1,5) is optimal (p = 6, d = 1, q = 5).

In [18]:

```
model_check = pm.auto_arima(Close,
                      seasonal=False,  # set to True if the time series is seasonal
                      stepwise=True,   # use the stepwise algorithm
                      suppress_warnings=True,
                      error_action="ignore",
                      max_p = 100,  # maximum amount of variables to be used in the ARIMA model
                      max_q = 100,
                      max_d = 1,
                      trace=True)      # print results whilst training

# Summary of the best ARIMA model found
print(model_check.summary())

# Fit the model
```

```
model_check.fit(Close)

# For predictions or model diag
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.48 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=15940.337, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=15654.822, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=15566.386, Time=0.09 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=15938.414, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=15564.459, Time=0.24 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=15563.262, Time=0.13 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=15578.210, Time=0.03 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=15564.874, Time=0.18 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.26 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=15570.002, Time=0.04 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=15542.804, Time=0.49 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=15494.286, Time=0.62 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=15539.172, Time=0.24 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=15467.218, Time=1.06 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=15529.067, Time=0.50 sec
 ARIMA(6,1,2)(0,0,0)[0] intercept   : AIC=15528.124, Time=1.49 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=15465.586, Time=1.13 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=15466.502, Time=0.82 sec
 ARIMA(6,1,3)(0,0,0)[0] intercept   : AIC=15469.572, Time=1.46 sec
 ARIMA(5,1,4)(0,0,0)[0] intercept   : AIC=15468.274, Time=1.10 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=15461.477, Time=1.13 sec
 ARIMA(3,1,4)(0,0,0)[0] intercept   : AIC=15472.655, Time=0.90 sec
 ARIMA(4,1,5)(0,0,0)[0] intercept   : AIC=15390.560, Time=1.28 sec
 ARIMA(3,1,5)(0,0,0)[0] intercept   : AIC=15464.295, Time=1.21 sec
 ARIMA(5,1,5)(0,0,0)[0] intercept   : AIC=15388.314, Time=1.72 sec
 ARIMA(6,1,5)(0,0,0)[0] intercept   : AIC=15387.459, Time=1.70 sec
 ARIMA(6,1,4)(0,0,0)[0] intercept   : AIC=15482.659, Time=1.37 sec
 ARIMA(7,1,5)(0,0,0)[0] intercept   : AIC=15390.800, Time=2.85 sec
 ARIMA(6,1,6)(0,0,0)[0] intercept   : AIC=15394.262, Time=1.61 sec
 ARIMA(5,1,6)(0,0,0)[0] intercept   : AIC=15396.945, Time=1.58 sec
 ARIMA(7,1,4)(0,0,0)[0] intercept   : AIC=15462.243, Time=1.75 sec
 ARIMA(7,1,6)(0,0,0)[0] intercept   : AIC=15413.842, Time=1.77 sec
 ARIMA(6,1,5)(0,0,0)[0]             : AIC=15385.644, Time=1.04 sec
 ARIMA(5,1,5)(0,0,0)[0]             : AIC=15386.998, Time=1.17 sec
 ARIMA(6,1,4)(0,0,0)[0]             : AIC=15480.191, Time=1.11 sec
 ARIMA(7,1,5)(0,0,0)[0]             : AIC=15388.964, Time=1.83 sec
 ARIMA(6,1,6)(0,0,0)[0]             : AIC=15387.726, Time=1.49 sec
 ARIMA(5,1,4)(0,0,0)[0]             : AIC=15468.248, Time=0.82 sec
 ARIMA(5,1,6)(0,0,0)[0]             : AIC=15395.101, Time=1.73 sec
 ARIMA(7,1,4)(0,0,0)[0]             : AIC=15459.122, Time=1.45 sec
 ARIMA(7,1,6)(0,0,0)[0]             : AIC=15412.363, Time=2.85 sec

Best model:  ARIMA(6,1,5)(0,0,0)[0]
Total fit time: 42.768 seconds
```

```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                 1304
Model:               SARIMAX(6, 1, 5)   Log Likelihood               -7680.822
Date:                Sat, 23 Mar 2024   AIC                           15385.644
Time:                        17:44:31   BIC                           15447.713
Sample:                             0   HQIC                          15408.930
                                 - 1304
Covariance Type:                   opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.3595      0.127    -10.688      0.000      -1.609      -1.110
ar.L2         -1.5491      0.190     -8.156      0.000      -1.921      -1.177
ar.L3         -1.5485      0.189     -8.173      0.000      -1.920      -1.177
ar.L4         -1.5256      0.190     -8.027      0.000      -1.898      -1.153
ar.L5         -0.5434      0.187     -2.899      0.004      -0.911      -0.176
ar.L6         -0.1758      0.065     -2.705      0.007      -0.303      -0.048
ma.L1          0.8197      0.126      6.516      0.000       0.573       1.066
ma.L2          0.8463      0.127      6.686      0.000       0.598       1.094
ma.L3          0.8151      0.116      7.010      0.000       0.587       1.043
ma.L4          0.7723      0.118      6.547      0.000       0.541       1.004
ma.L5         -0.1143      0.112     -1.018      0.309      -0.334       0.106
sigma2      8099.7963    179.025     45.244      0.000    7748.913    8450.680
===================================================================================
Ljung-Box (L1) (Q):                   0.18   Jarque-Bera (JB):              4981.79
Prob(Q):                              0.67   Prob(JB):                         0.00
Heteroskedasticity (H):               0.33   Skew:                            -1.42
Prob(H) (two-sided):                  0.00   Kurtosis:                        12.15
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Out[18]:

ARIMA
ARIMA(6,1,5)(0,0,0)[0]

We create the five-day ARIMA forecast using this optimal model.

In [19]:

```
# Load the historical data
data = pd.read_excel("APEData.xlsx")
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
Close = data['Close'].dropna()
```

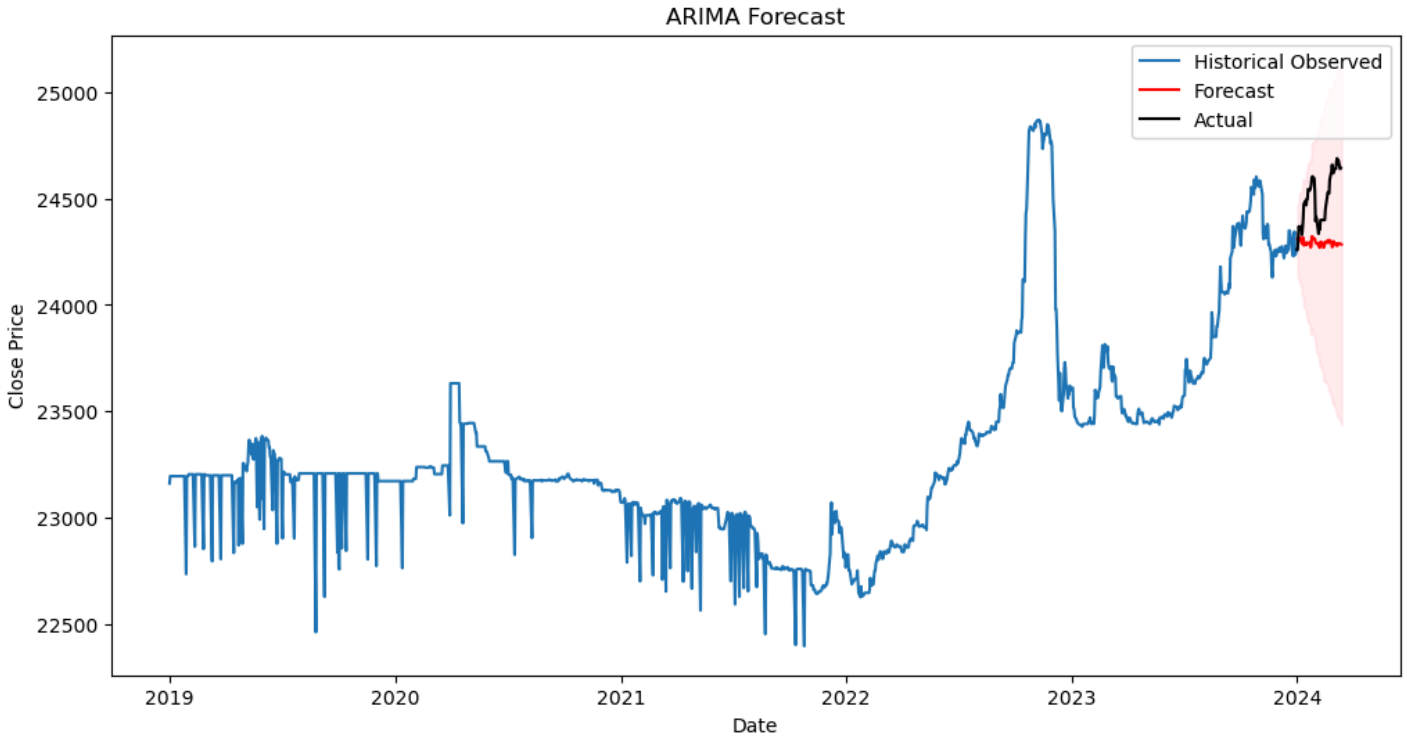
```
# Confirm there are no NaNs in the Close series
Close = Close.interpolate()

arima_model = ARIMA(Close, order = (6,1,5))
model = arima_model.fit()
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                 1304
Model:                 ARIMA(6, 1, 5)   Log Likelihood               -7680.822
Date:                Sat, 23 Mar 2024   AIC                          15385.644
Time:                        17:44:34   BIC                          15447.713
Sample:                             0   HQIC                         15408.930
                                - 1304
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.3595      0.127    -10.688      0.000      -1.609      -1.110
ar.L2         -1.5491      0.190     -8.156      0.000      -1.921      -1.177
ar.L3         -1.5485      0.189     -8.173      0.000      -1.920      -1.177
ar.L4         -1.5256      0.190     -8.027      0.000      -1.898      -1.153
ar.L5         -0.5434      0.187     -2.899      0.004      -0.911      -0.176
ar.L6         -0.1758      0.065     -2.705      0.007      -0.303      -0.048
ma.L1          0.8197      0.126      6.516      0.000       0.573       1.066
ma.L2          0.8463      0.127      6.686      0.000       0.598       1.094
ma.L3          0.8151      0.116      7.010      0.000       0.587       1.043
ma.L4          0.7723      0.118      6.547      0.000       0.541       1.004
ma.L5         -0.1143      0.112     -1.018      0.309      -0.334       0.106
sigma2      8099.7963    179.025     45.244      0.000    7748.913    8450.680
===================================================================================
Ljung-Box (L1) (Q):                   0.18   Jarque-Bera (JB):              4981.79
Prob(Q):                              0.67   Prob(JB):                         0.00
Heteroskedasticity (H):               0.33   Skew:                            -1.42
Prob(H) (two-sided):                  0.00   Kurtosis:                        12.15
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

In [20]:
```
Close_2023_onwards = Close['2023-01-01':].asfreq('B')

# Forecast the next 5 business days
forecast = model.get_forecast(steps=5)
```

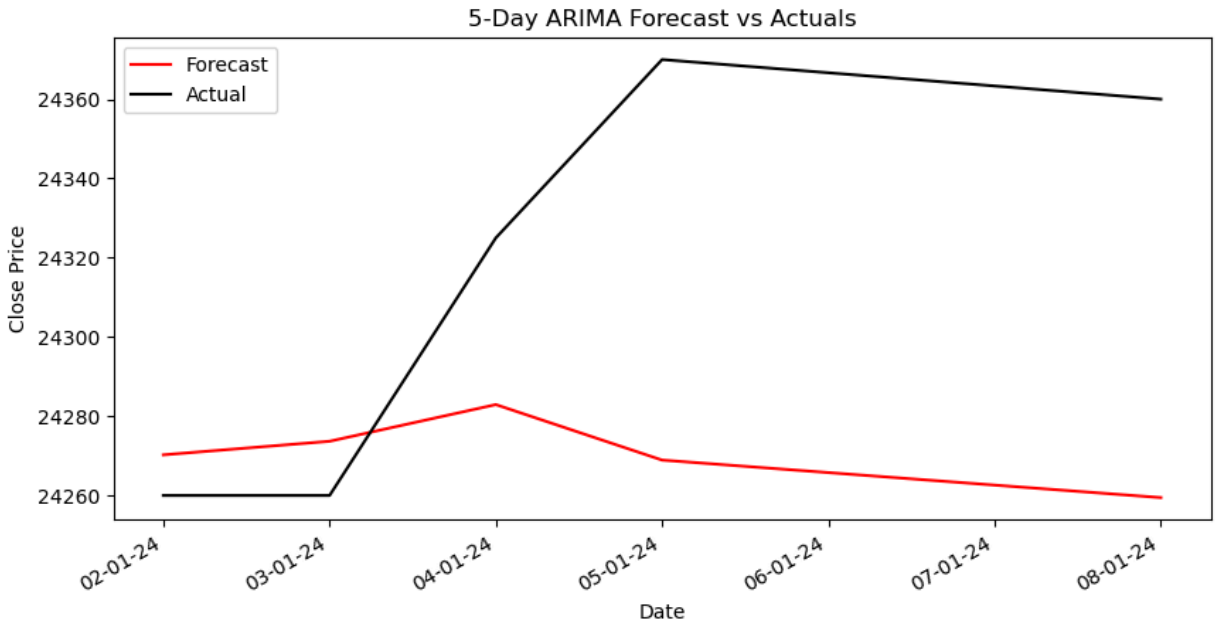
```
predicted_mean = forecast.predicted_mean
confidence_intervals = forecast.conf_int()

actual_values = [
    24260.0000, 24260.0000, 24325.0000, 24370.0000, 24360.0000,
    24370.0000, 24330.0000, 24360.0000, 24405.0000, 24475.0000,
    24490.0000, 24470.0000, 24500.0000, 24500.0000, 24545.0000,
    24540.0000, 24550.0000, 24565.0000, 24600.0000, 24605.0000,
    24595.0000, 24530.0000, 24395.0000, 24415.0000, 24400.0000,
    24335.0000, 24370.0000, 24355.0000, 24400.0000, 24400.0000,
    24400.0000, 24400.0000, 24400.0000, 24400.0000, 24450.0000,
    24510.0000, 24530.0000, 24525.0000, 24525.0000, 24590.0000,
    24630.0000, 24660.0000, 24625.0000, 24620.0000, 24640.0000,
    24640.0000, 24676.0000, 24690.0000, 24665.0000, 24680.0000,
    24640.0000, 24645.0000
]

date_range = pd.date_range(start='2024-01-01', end='2024-03-12', freq='B')


# Create a pandas Series with the actual values
actual_series = pd.Series(actual_values, index=date_range)



arima_model = ARIMA(Close, order=(15, 1, 30))
model_fit = arima_model.fit()

# Forecast the next 52 business days
# The start date for the forecast is the day after the last date in the historical data
forecast_start_date = Close.index[-1] + pd.Timedelta(days=1)
# Ensure we're forecasting only business days
forecast_dates = pd.date_range(start=forecast_start_date, periods=52, freq='B')
forecast = model_fit.get_forecast(steps=52)
predicted_mean = forecast.predicted_mean
confidence_intervals = forecast.conf_int()

# Align the forecast dates with the predicted mean
predicted_mean.index = forecast_dates

# Plot the actual values and forecasted values
plt.figure(figsize=(12, 6))
plt.plot(Close, label='Historical Observed')
plt.plot(predicted_mean.index, predicted_mean, color='red', label='Forecast')
plt.fill_between(predicted_mean.index,
                 confidence_intervals.iloc[:, 0],
```

```
                confidence_intervals.iloc[:, 1], color='pink', alpha=0.3)
plt.plot(actual_series.index, actual_series, color='black', label='Actual')

plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('ARIMA Forecast')
plt.legend()
plt.show()
```



In [21]:
```
start_forecast_date = '2024-01-02'
date_range_forecast = pd.date_range(start=start_forecast_date, periods=5, freq='B')
forecast_5_day_arima = model.get_forecast(steps=5)
```

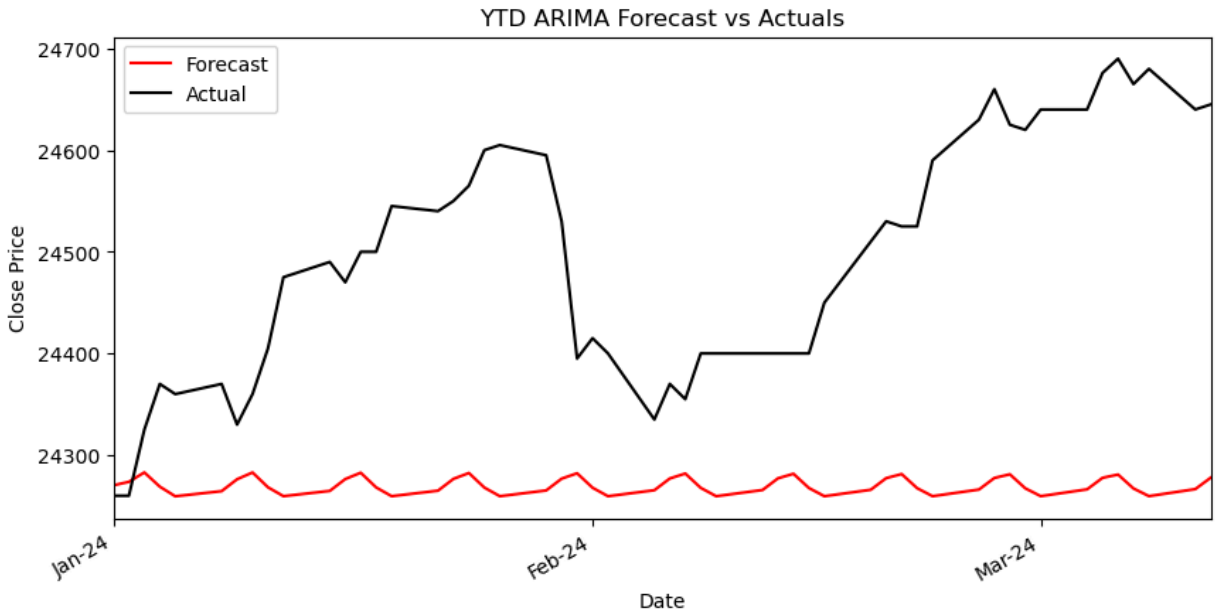
```
predicted_mean_5_day_arima = forecast_5_day_arima.predicted_mean
predicted_mean_5_day_arima.index = date_range_forecast

actual_values = [24260.0000, 24260.0000, 24325.0000, 24370.0000, 24360.0000]
date_range_actual = pd.date_range(start=start_forecast_date, periods=len(actual_values), freq='B')
actual_series = pd.Series(actual_values, index=date_range_actual)

# Plot the results
plt.figure(figsize=(10, 5))

# Plot the forecasted values
plt.plot(date_range_forecast, predicted_mean_5_day_arima, color='red', label='Forecast')

# Plot the actual values
plt.plot(date_range_actual, actual_values, color='black', label='Actual')

plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('5-Day ARIMA Forecast vs Actuals')
plt.legend()

# Format the dates on the x-axis to show only the date part
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%y'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator())
plt.gcf().autofmt_xdate() # Rotation

plt.show()
```



We extend the ARIMA forecast to March 12th 2024.

In [22]:
```
start_forecast_date = '2024-01-01'
date_range_forecast = pd.date_range(start=start_forecast_date, periods=52, freq='B')
forecast_YTD_2024_arima = model.get_forecast(steps=52)
predicted_mean_YTD_2024_arima = forecast_YTD_2024_arima.predicted_mean
predicted_mean_YTD_2024_arima.index = date_range_forecast

actual_values = [
    24260.0000, 24260.0000, 24325.0000, 24370.0000, 24360.0000,
    24370.0000, 24330.0000, 24360.0000, 24405.0000, 24475.0000,
    24490.0000, 24470.0000, 24500.0000, 24500.0000, 24545.0000,
    24540.0000, 24550.0000, 24565.0000, 24600.0000, 24605.0000,
    24595.0000, 24530.0000, 24395.0000, 24415.0000, 24400.0000,
    24335.0000, 24370.0000, 24355.0000, 24400.0000, 24400.0000,
    24400.0000, 24400.0000, 24400.0000, 24400.0000, 24450.0000,
```

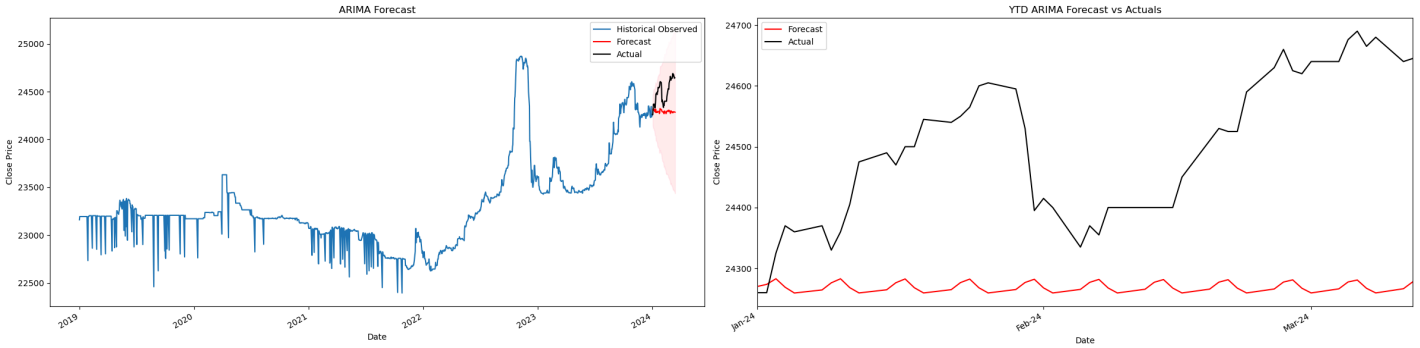
```
    24510.0000, 24530.0000, 24525.0000, 24525.0000, 24590.0000,
    24630.0000, 24660.0000, 24625.0000, 24620.0000, 24640.0000,
    24640.0000, 24676.0000, 24690.0000, 24665.0000, 24680.0000,
    24640.0000, 24645.0000
]
date_range_actual = pd.date_range(start=start_forecast_date, periods=len(actual_values), freq='B')
actual_series = pd.Series(actual_values, index=date_range_actual)

locator = mdates.MonthLocator()
formatter = mdates.DateFormatter('%b-%y')

plt.figure(figsize=(10, 5))

# Plot the forecasted values
plt.plot(date_range_forecast, predicted_mean_YTD_2024_arima, color='red', label='Forecast')

# Plot the actual values
plt.plot(date_range_actual, actual_series, color='black', label='Actual')

plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('YTD ARIMA Forecast vs Actuals')
plt.legend()

# Set the locator
plt.gca().xaxis.set_major_locator(locator)
# Set the formatter
plt.gca().xaxis.set_major_formatter(formatter)

# We set the limit on the x-axis to ensure we only have three ticks as requested
plt.xlim([date_range_forecast[0], date_range_forecast[-1]])

plt.gcf().autofmt_xdate() # Rotation

plt.show()
```



In [23]:

```
# Initialize a figure and two subplots, side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 6))

# First plot
ax1.plot(Close, label='Historical Observed')
ax1.plot(predicted_mean.index, predicted_mean, color='red', label='Forecast')
ax1.fill_between(predicted_mean.index,
                 confidence_intervals.iloc[:, 0],
                 confidence_intervals.iloc[:, 1], color='pink', alpha=0.3)
ax1.plot(actual_series.index, actual_series, color='black', label='Actual')
ax1.set_xlabel('Date')
ax1.set_ylabel('Close Price')
ax1.set_title('ARIMA Forecast')
ax1.legend()

# Second plot
ax2.plot(date_range_forecast, predicted_mean_YTD_2024_arima, color='red', label='Forecast')
```

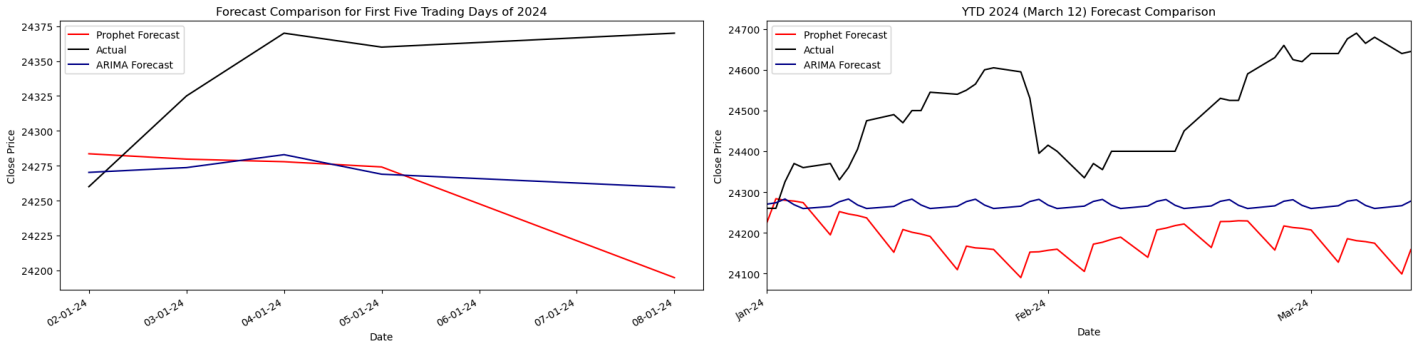
```
ax2.plot(date_range_actual, actual_series, color='black', label='Actual')
ax2.set_xlabel('Date')
ax2.set_ylabel('Close Price')
ax2.set_title('YTD ARIMA Forecast vs Actuals')
ax2.legend()

# Set the locator and formatter for the second plot
locator = mdates.MonthLocator()
formatter = mdates.DateFormatter('%b-%y')
ax2.xaxis.set_major_locator(locator)
ax2.xaxis.set_major_formatter(formatter)
ax2.set_xlim([date_range_forecast[0], date_range_forecast[-1]])
fig.autofmt_xdate() # Rotation for better date display

plt.tight_layout()
plt.show()
```



For clarity, we prepare a side-by-side comparison of the forecasts for the five-day and YTD periods.

In [24]:
```
# Prepare for side-by-side plotting

fig, ax = plt.subplots(1, 2, figsize=(20, 5))

# Plot 1: Prophet Forecast vs Actuals for First Five Trading Days of 2024
ax[0].plot(forecast_5day_prophet['ds'], forecast_5day_prophet['yhat'], color='red', label='Prophet Forecast')
ax[0].plot(actual_series_2024.index, actual_series_2024.values, color='black', label='Actual')
ax[0].plot(forecast_5day_prophet['ds'], predicted_mean_5_day_arima, color='darkblue', label='ARIMA Forecast')
ax[0].set_xlabel('Date')
ax[0].set_ylabel('Close Price')
```

```
ax[0].set_title('Forecast Comparison for First Five Trading Days of 2024')
ax[0].legend()
ax[0].xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%y'))
ax[0].xaxis.set_major_locator(mdates.DayLocator())
fig.autofmt_xdate()  # Auto-rotate date labels for better readability

# Plot 2: 52-Day Prophet Forecast vs Actuals
ax[1].plot(date_range_forecast, predicted_mean_YTD_2024_prophet, color='red', label='Prophet Forecast')
ax[1].plot(date_range_actual, actual_series, color='black', label='Actual')
ax[1].plot(date_range_actual, predicted_mean_YTD_2024_arima, color='darkblue', label='ARIMA Forecast')
ax[1].set_xlabel('Date')
ax[1].set_ylabel('Close Price')
ax[1].set_title('YTD 2024 (March 12) Forecast Comparison')
ax[1].legend()
ax[1].xaxis.set_major_locator(locator)
ax[1].xaxis.set_major_formatter(formatter)
ax[1].set_xlim([date_range_forecast[0], date_range_forecast[-1]])

plt.tight_layout()
plt.show()
```



We compute the errors for both forecast methods across the five day and YTD horizons

In [25]:

```
### Comparing the performance of the two models


# Actual values for YTD
y_true_YTD = np.array([24260.0000, 24260.0000, 24325.0000, 24370.0000, 24360.0000,
    24370.0000, 24330.0000, 24360.0000, 24405.0000, 24475.0000,
    24490.0000, 24470.0000, 24500.0000, 24500.0000, 24545.0000,
```

```
    24540.0000, 24550.0000, 24565.0000, 24600.0000, 24605.0000,
    24595.0000, 24530.0000, 24395.0000, 24415.0000, 24400.0000,
    24335.0000, 24370.0000, 24355.0000, 24400.0000, 24400.0000,
    24400.0000, 24400.0000, 24400.0000, 24400.0000, 24450.0000,
    24510.0000, 24530.0000, 24525.0000, 24525.0000, 24590.0000,
    24630.0000, 24660.0000, 24625.0000, 24620.0000, 24640.0000,
    24640.0000, 24676.0000, 24690.0000, 24665.0000, 24680.0000,
    24640.0000, 24645.0000])

# 5-day forecasted values from ARIMA
y_pred_YTD_arima = predicted_mean_YTD_2024_arima.values

# 5-day forecasted values from Prophet
y_pred_YTD_prophet = forecast_prophet_YTD_2024['yhat'].values


# Calculate metrics
def calculate_mape(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


def calculate_mase(y_true, y_pred):
    n = 52
    d = np.abs(np.diff(y_train)).sum() / (n - 1)
    errors = np.abs(y_true - y_pred)
    return errors.mean() / d

def calculate_mse(y_true, y_pred):
    return mean_squared_error(y_true, y_pred)

# Calculate errors for YTD forecasts
metrics_5d = {
    'RMSE': {
        'ARIMA': np.sqrt(mean_squared_error(y_true_YTD, y_pred_YTD_arima)),
        'Prophet': np.sqrt(mean_squared_error(y_true_YTD, y_pred_YTD_prophet)),
    },
    'MAE': {
        'ARIMA': mean_absolute_error(y_true_YTD, y_pred_YTD_arima),
        'Prophet': mean_absolute_error(y_true_YTD, y_pred_YTD_prophet),
    },
    'MAPE': {
        'ARIMA': calculate_mape(y_true_YTD, y_pred_YTD_arima),
        'Prophet': calculate_mape(y_true_YTD, y_pred_YTD_prophet),
    },
    'MSE':{
        'ARIMA': calculate_mse(y_true_YTD, y_pred_YTD_arima),
```

```
        'Prophet': calculate_mse(y_true_YTD, y_pred_YTD_prophet)

    }
}

metrics_df = pd.DataFrame(metrics_5d)
metrics_df
```

Out[25]:

	RMSE	MAE	MAPE	MSE
ARIMA	252.74	223.09	0.91	63,879.36
Prophet	333.80	302.27	1.23	111,425.10

In [26]:

```
### Comparing the performance of the two models



# Actual values for both 5-day forecasts
y_true_5d = np.array([24260.0000, 24260.0000, 24325.0000, 24370.0000, 24360.0000])

# 5-day forecasted values from ARIMA
y_pred_5d_arima = predicted_mean_5_day_arima.values

# 5-day forecasted values from Prophet
y_pred_5d_prophet = forecast_5day_prophet['yhat'].values

# Calculate metrics
def calculate_mape(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


def calculate_mase(y_true, y_pred):
    n = 5
    d = np.abs(np.diff(y_train)).sum() / (n - 1)
    errors = np.abs(y_true - y_pred)
    return errors.mean() / d

def calculate_mse(y_true, y_pred):
    return mean_squared_error(y_true, y_pred)

# Calculate errors for 5-day forecasts
metrics_5d = {
    'RMSE': {
        'ARIMA': np.sqrt(mean_squared_error(y_true_5d, y_pred_5d_arima)),
        'Prophet': np.sqrt(mean_squared_error(y_true_5d, y_pred_5d_prophet)),
```

```
    },
    'MAE': {
        'ARIMA': mean_absolute_error(y_true_5d, y_pred_5d_arima),
        'Prophet': mean_absolute_error(y_true_5d, y_pred_5d_prophet),
    },
    'MAPE': {
        'ARIMA': calculate_mape(y_true_5d, y_pred_5d_arima),
        'Prophet': calculate_mape(y_true_5d, y_pred_5d_prophet),
    },
    'MSE':{
        'ARIMA': calculate_mse(y_true_5d, y_pred_5d_arima),
        'Prophet': calculate_mse(y_true_5d, y_pred_5d_prophet)

    }
}

metrics_df = pd.DataFrame(metrics_5d)
metrics_df
```

Out[26]:

	RMSE	MAE	MAPE	MSE
ARIMA	66.91	53.52	0.22	4,476.99
Prophet	89.04	70.29	0.29	7,927.26

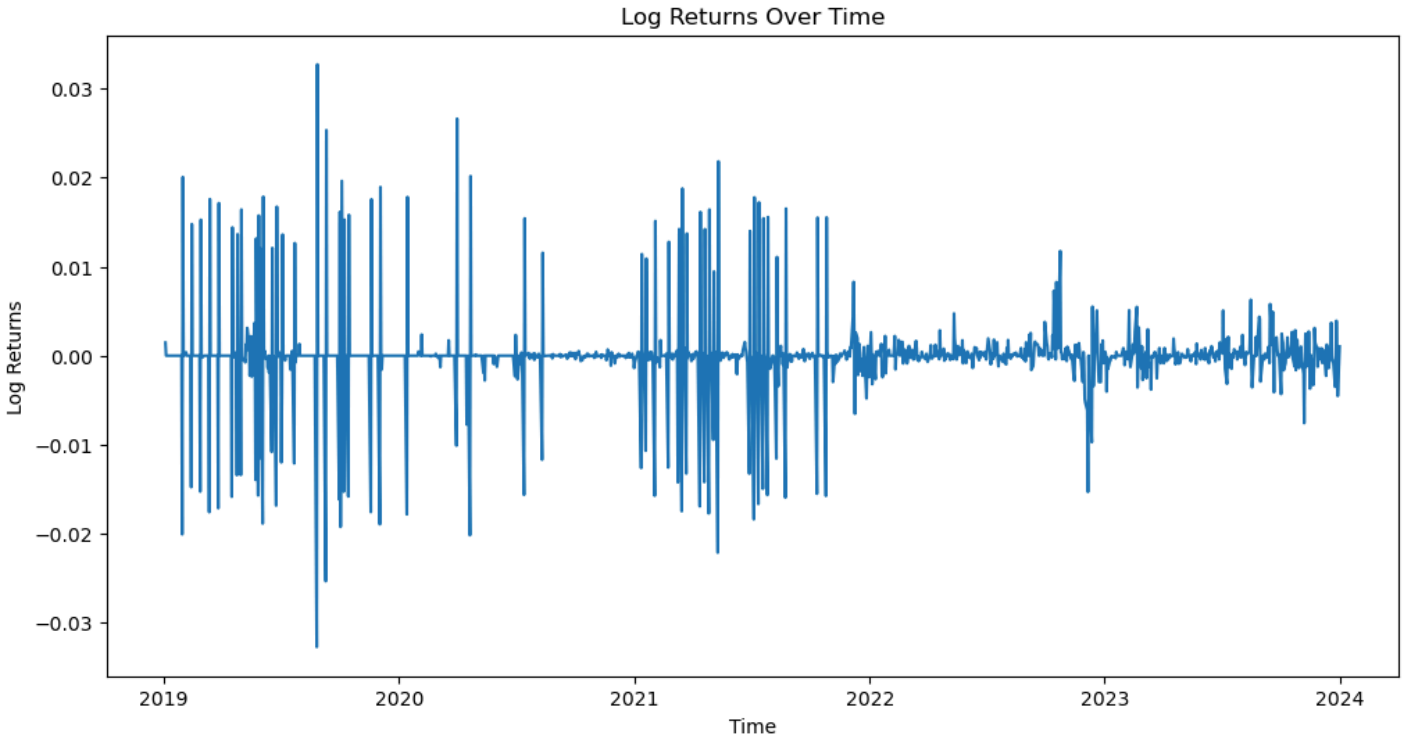
Question 2c: Value at risk

We create and plot the log return series

In [27]:

```
data = pd.read_excel("APEData.xlsx") ## reuploading data after using a different data set for the YTD forecast
Close = data['Close'].dropna()
Time = pd.to_datetime(data['Date'].dropna())

# Calculate log returns
log_returns = np.log(Close).diff()
log_returns = log_returns.dropna()

# Align 'Time' with 'log_returns' by dropping the first entry
Time_adjusted = Time[2:]  # Drop the first date to match 'log_returns' length

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(Time_adjusted, log_returns)
plt.title('Log Returns Over Time')
```

```
plt.xlabel('Time')
plt.ylabel('Log Returns')
plt.show()
```



We test for stationarity to understand what statistical methods we are able to apply.

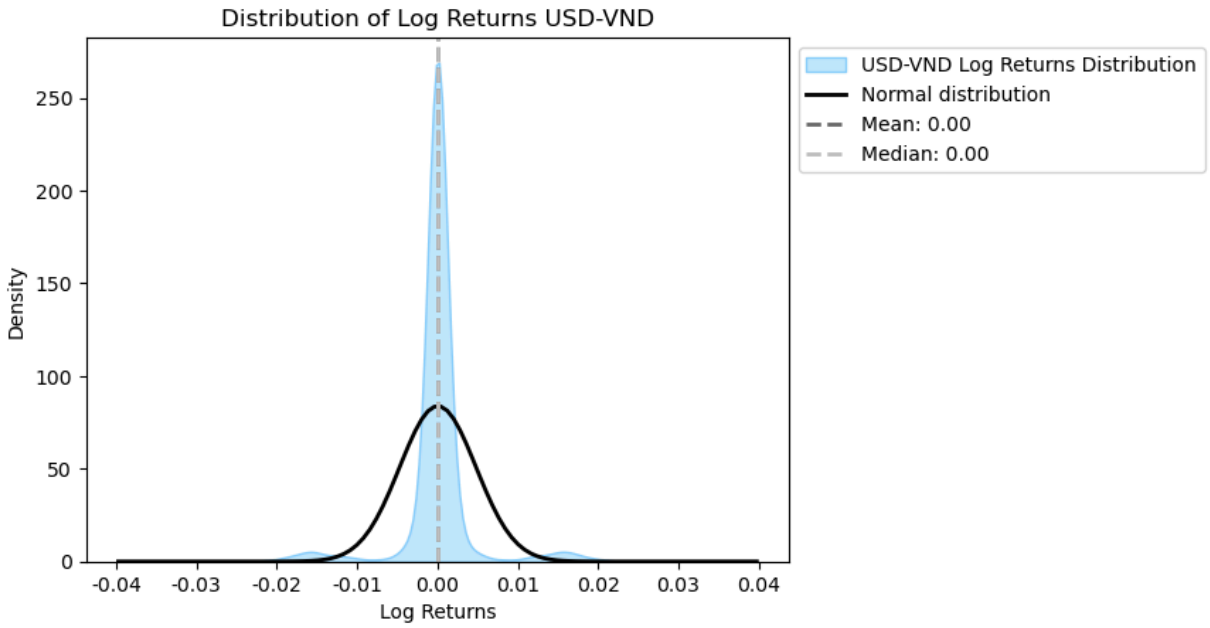
In [28]:
```
### Testing for stationarity

returns_stationary = adfuller(log_returns)
print('ADF Statistic: %f' % returns_stationary[0])
print('p-value: %f' % returns_stationary[1])
```

```
ADF Statistic: -9.020530
p-value: 0.000000
```

Plotting the distribution of the log returns series.

In [29]:

```python
mean_returns = np.mean(log_returns)
median_returns = np.median(log_returns)
std_returns = np.std(log_returns)

# Convert mean and median to string with commas for thousands and two significant figures
mean_returns_str = "{:,.2f}".format(mean_returns)
median_returns_str = "{:,.2f}".format(median_returns)

# Initialize a matplotlib figure
fig, ax = plt.subplots()

# Create a Kernel Density Estimate plot for the log returns
sns.kdeplot(log_returns, ax=ax, color="lightskyblue", label="USD-VND Log Returns Distribution", alpha=0.5, fill=True)

# Normal distribution overlay
xmin, xmax = ax.get_xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mean_returns, std_returns)
ax.plot(x, p, 'k', linewidth=2, label="Normal distribution")

# Vertical line for the mean and median with labels
ax.axvline(mean_returns, color='dimgray', linestyle='dashed', linewidth=2, label=f'Mean: {mean_returns_str}')
ax.axvline(median_returns, color='silver', linestyle='dashed', linewidth=2, label=f'Median: {median_returns_str}')

# Set labels and title
ax.set_ylabel('Density')
ax.set_xlabel('Log Returns')
# Place the legend outside the plot on the right side
ax.legend(loc='upper left', bbox_to_anchor=(1, 1))
ax.set_title("Distribution of Log Returns USD-VND")

# Format x-axis with two significant figures
ax.get_xaxis().set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:,.2f}".format(x)))

# Size

plt.figure(figsize=(12,6))

plt.tight_layout()
plt.show()
```




```
<Figure size 1200x600 with 0 Axes>
```

Identifying key statistical properties of the distribution.

In [30]:

```
Percentiles = [0.05, 0.25, 0.5, 0.75, 0.95]
sum_stats = log_returns.describe(percentiles=Percentiles)

# Calculate kurtosis and excess kurtosis
kurtosis = log_returns.kurtosis()
excess_kurtosis = kurtosis - 3

# Calculate skewness
skewness = log_returns.skew()

# Convert the summary stats to a DataFrame
sum_stats_df = pd.DataFrame(sum_stats)
```

```python
# Add excess kurtosis and skewness
sum_stats_df.loc['excess kurtosis'] = excess_kurtosis
sum_stats_df.loc['skewness'] = skewness

# Renaming the indices to be more descriptive
new_index_labels = {
    'count': "Amount of Observations",
    'mean': "Mean Value",
    'std': "Standard Deviation",
    'min': "Minimum Value",
    '5%': "5th Percentile",
    '25%': "25th Percentile",
    '50%': "50th Percentile",  # Median
    '75%': "75th Percentile",
    '95%': "95th Percentile",
    'max': "Maximum Value",
    'excess kurtosis': "Excess Kurtosis",
    'skewness': "Skewness"  # Added skewness here
}
sum_stats_df_renamed = sum_stats_df.rename(index=new_index_labels)

# Ensure the column name is 'Log Returns'
sum_stats_df_renamed.columns = ['Log Returns']

# Format the DataFrame for better readability
sum_stats_df_formatted = sum_stats_df_renamed.style.format("{:.4f}")

# Display the formatted DataFrame
sum_stats_df_formatted
```

Out[30]:

	Log Returns
Amount of Observations	1303.0000
Mean Value	0.0000
Standard Deviation	0.0048
Minimum Value	-0.0327
5th Percentile	-0.0038
25th Percentile	-0.0002
50th Percentile	0.0000
75th Percentile	0.0003
95th Percentile	0.0043
Maximum Value	0.0327
Excess Kurtosis	9.0646
Skewness	0.0968

Historical approach for computing VaR:

In [31]:
```
VaR_1_percent = np.percentile(log_returns, 1)
```

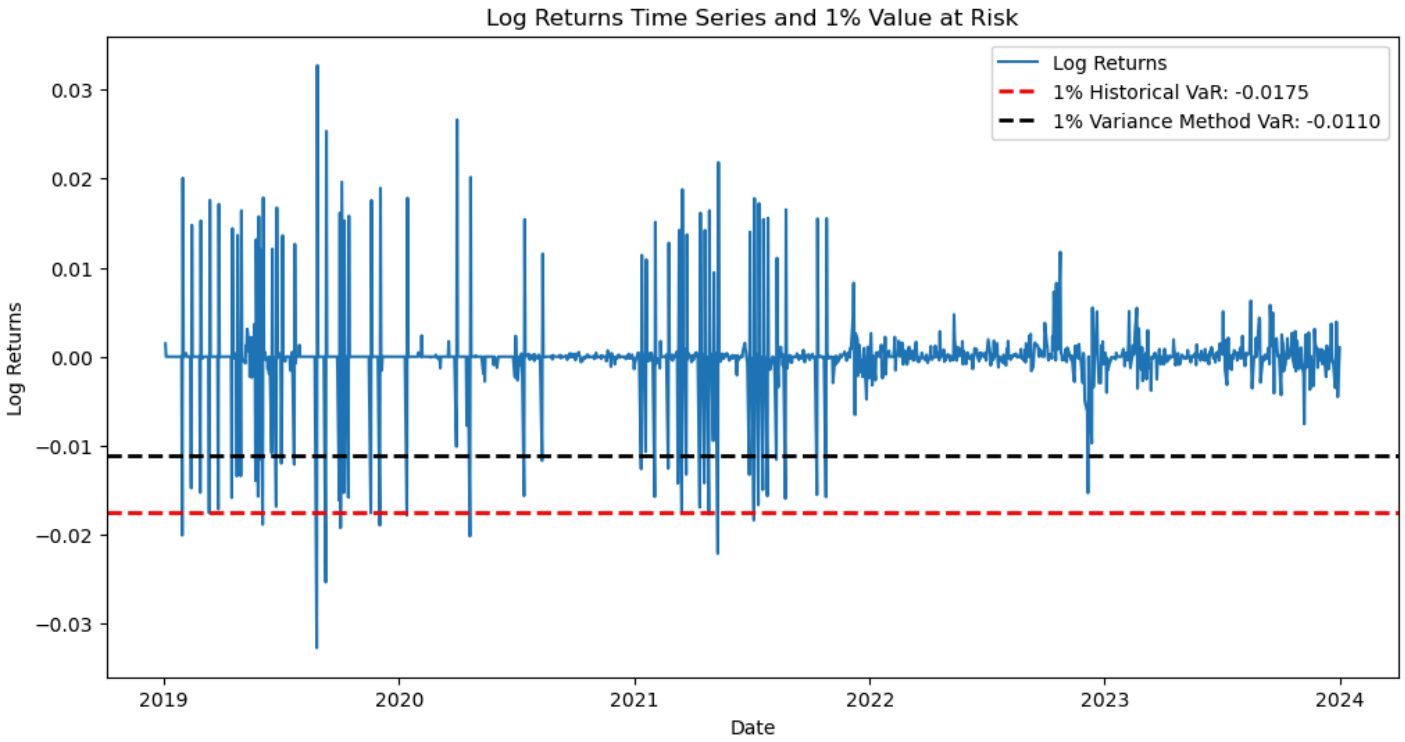
Variance approach for computing VaR:

In [32]:
```
# Find the mean and std deviation
mean_returns = log_returns.mean()
std_returns = log_returns.std()

# For a 1% VaR, we want the z-score that captures 99% of the distribution to the left
z_score = norm.ppf(0.01)

# Calculate VaR using the variance-covariance method
VaR_variance_method = mean_returns + (z_score * std_returns)
```

Plotting log returns with VaRs:

In [33]:
```
# Calculate the 1% empirical Value at Risk (VaR)
VaR_1_percent = np.percentile(log_returns, 1)
```

```python
# Calculate the 1% VaR using the variance-covariance method
mean_returns = np.mean(log_returns)
std_returns = np.std(log_returns)
z_score = norm.ppf(0.01)
VaR_variance_method = mean_returns + (z_score * std_returns)

# Plotting the log returns time series
plt.figure(figsize=(12, 6))
plt.plot(Time[2:], log_returns, label='Log Returns')  # Skip the first element to match the log_returns series

# Overlay the 1% VaR as a horizontal line
plt.axhline(VaR_1_percent, color='red', linestyle='dashed', linewidth=2, label=f'1% Historical VaR: {VaR_1_percent:.4f}')
plt.axhline(VaR_variance_method, color='black', linestyle='dashed', linewidth=2, label=f'1% Variance Method VaR: {VaR_variance_

plt.title('Log Returns Time Series and 1% Value at Risk')
plt.xlabel('Date')
plt.ylabel('Log Returns')
plt.legend()
plt.show()
```



We find the violations count for both methods to identify which is more accurate.

In [34]:
```
# Calculate the percentage of observations below the 1% empirical VaR
empirical_VaR_violations = (log_returns < VaR_1_percent).sum()
percent_empirical_VaR_violations = (empirical_VaR_violations / len(log_returns)) * 100

# Calculate the percentage of observations below the 1% variance method VaR
variance_VaR_violations = (log_returns < VaR_variance_method).sum()
percent_variance_VaR_violations = (variance_VaR_violations / len(log_returns)) * 100

# Create a DataFrame without the default numerical index
```

```
varR_violations_df = pd.DataFrame({
    'Violations Count': [empirical_VaR_violations, variance_VaR_violations],
    'Violations Percentage': [percent_empirical_VaR_violations, percent_variance_VaR_violations]
}, index=['Empirical 1% VaR', 'Variance Method 1% VaR'])

varR_violations_df
```

Out[34]:

	Violations Count	Violations Percentage
Empirical 1% VaR	14	1.07
Variance Method 1% VaR	47	3.61

Question 2d: Event Analysis

We calculate the Abnormal returns for our event window around October 10th 2022

In [35]:

```
# Prepare the DataFrame with 'Date' and 'Log Returns'
data_with_returns = pd.DataFrame({'Date': data['Date'].iloc[1:], 'Log Returns': log_returns})

# Find the index for November 17, 2022
event_date_index = data_with_returns[data_with_returns['Date'] == '2022-10-10'].index[0]

# Define the pre-event and post-event windows
pre_event_end_index = event_date_index - 14
post_event_start_index = event_date_index + 1
post_event_end_index = event_date_index + 14

# Filter log returns for the pre-event window to calculate mean returns
pre_event_returns = data_with_returns.iloc[:pre_event_end_index]['Log Returns']
mean_pre_event_returns = np.mean(pre_event_returns)

# Calculate abnormal returns for the event window
event_window_returns = data_with_returns.iloc[pre_event_end_index + 1:post_event_end_index + 1]['Log Returns']
abnormal_returns = event_window_returns - mean_pre_event_returns

# Add abnormal returns to the DataFrame for visualization
data_with_returns.loc[pre_event_end_index + 1:post_event_end_index, 'Abnormal Returns'] = abnormal_returns

# Filter the DataFrame to only include the event window for visualization or analysis
event_analysis_df = data_with_returns.loc[pre_event_end_index + 1:post_event_end_index]

# Set the 'Date' as the index of the DataFrame
event_analysis_df.set_index('Date', inplace=True)
```

```
# Format the numerical columns to display four significant figures
formatted_df = event_analysis_df.style.format({"Log Returns": "{:.4f}", "Abnormal Returns": "{:.4f}"})

# Display the formatted DataFrame
formatted_df

## https://www.bloomberg.com/news/articles/2022-10-19/vietnam-s-dong-poised-for-longest-losing-streak-since-2008?embedded-check
```

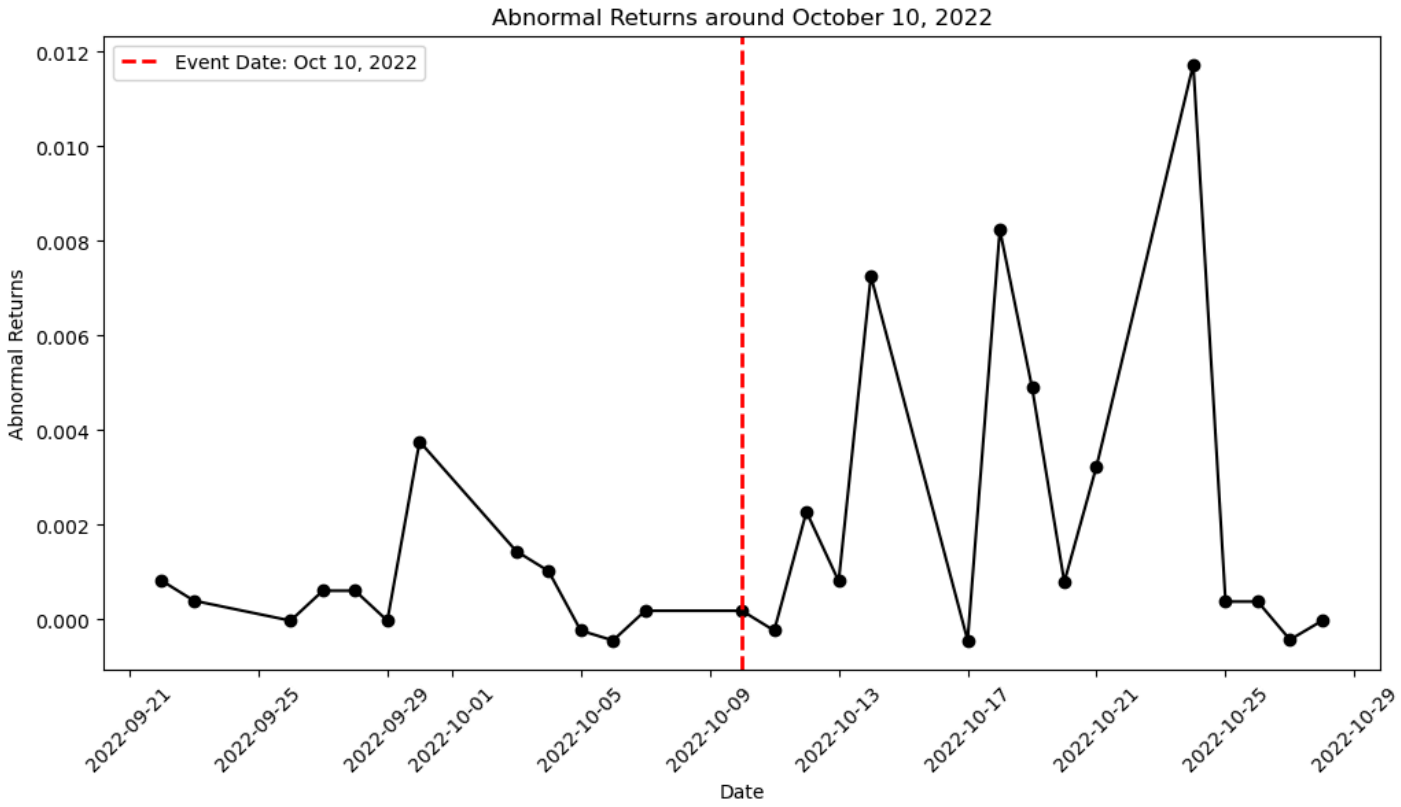
Out[35]:

Date	Log Returns	Abnormal Returns
2022-09-21 00:00:00	0.0000	nan
2022-09-22 00:00:00	0.0008	0.0008
2022-09-23 00:00:00	0.0004	0.0004
2022-09-26 00:00:00	0.0000	-0.0000
2022-09-27 00:00:00	0.0006	0.0006
2022-09-28 00:00:00	0.0006	0.0006
2022-09-29 00:00:00	0.0000	-0.0000
2022-09-30 00:00:00	0.0038	0.0038
2022-10-03 00:00:00	0.0015	0.0014
2022-10-04 00:00:00	0.0010	0.0010
2022-10-05 00:00:00	-0.0002	-0.0002
2022-10-06 00:00:00	-0.0004	-0.0004
2022-10-07 00:00:00	0.0002	0.0002
2022-10-10 00:00:00	0.0002	0.0002
2022-10-11 00:00:00	-0.0002	-0.0002
2022-10-12 00:00:00	0.0023	0.0023
2022-10-13 00:00:00	0.0008	0.0008
2022-10-14 00:00:00	0.0073	0.0073
2022-10-17 00:00:00	-0.0004	-0.0004
2022-10-18 00:00:00	0.0083	0.0082
2022-10-19 00:00:00	0.0049	0.0049
2022-10-20 00:00:00	0.0008	0.0008
2022-10-21 00:00:00	0.0033	0.0032
2022-10-24 00:00:00	0.0118	0.0117
2022-10-25 00:00:00	0.0004	0.0004

Date	Log Returns	Abnormal Returns
2022-10-26 00:00:00	0.0004	0.0004
2022-10-27 00:00:00	-0.0004	-0.0004
2022-10-28 00:00:00	0.0000	-0.0000

We plot abnormal returns for the event window.

In [36]:
```
# Check if 'Date' is not in the columns (which means the DataFrame has not been reset)
if 'Date' not in event_analysis_df.columns:
    event_analysis_df.reset_index(inplace=True)

# Convert 'Date' to datetime if it's not already (useful if the reset was just performed)
event_analysis_df['Date'] = pd.to_datetime(event_analysis_df['Date'])

# Find the position (index) of the event date within the DataFrame for plotting
event_date_position = event_analysis_df[event_analysis_df['Date'] == pd.to_datetime('2022-10-10')].index[0]

plt.figure(figsize=(12, 6))
plt.plot(event_analysis_df['Date'], event_analysis_df['Abnormal Returns'], marker='o', linestyle='-', color='black')
# Use the position for axvline
plt.axvline(x=event_analysis_df['Date'][event_date_position], color='red', linestyle='--', linewidth=2, label='Event Date: Oct
plt.title('Abnormal Returns around October 10, 2022')
plt.xlabel('Date')
plt.ylabel('Abnormal Returns')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



We find the CARs for all the event window periods.

In [37]:
```
# Ensure 'Date' is in datetime format and calculate log returns
data['Date'] = pd.to_datetime(data['Date'])
data['Log Returns'] = np.log(data['Close']).diff().dropna()

# Prepare the DataFrame with 'Date' and 'Log Returns', adjusting for the shift caused by diff()
data_with_returns = pd.DataFrame({
    'Date': data['Date'].iloc[1:],
```

```
    'Log Returns': data['Log Returns'].iloc[1:]
})

# Calculate average log returns from Jan 1, 2019, to September 23, 2022
avg_log_returns = data_with_returns[(data_with_returns['Date'] >= '2019-01-01') & (data_with_returns['Date'] <= '2022-09-23')][

# Define the event and estimation windows
pre_event_end_date = pd.to_datetime('2022-09-23')
post_event_start_date = pd.to_datetime('2022-09-26')
event_date = pd.to_datetime('2022-10-10')
post_event_end_date = pd.to_datetime('2022-10-25')

# Calculate the standard deviation of log returns in the estimation window
std_pre_event_log_returns = data_with_returns[(data_with_returns['Date'] >= '2019-01-01') & (data_with_returns['Date'] <= pre_e

# Pre-event
cumulative_abnormal_returns_pre = data_with_returns[(data_with_returns['Date'] >= post_event_start_date) & (data_with_returns['
t_stat_pre = cumulative_abnormal_returns_pre / (std_pre_event_log_returns * np.sqrt(10))
p_value_pre = stats.t.sf(np.abs(t_stat_pre), df=len(data_with_returns) - 1) * 2  # two-tailed p-value

# Day of event
abnormal_returns_event_day = data_with_returns[data_with_returns['Date'] == event_date]['Log Returns'].iloc[0] - avg_log_return

# Post-event
cumulative_abnormal_returns_post = data_with_returns[(data_with_returns['Date'] >= '2022-10-11') & (data_with_returns['Date'] <
t_stat_post = cumulative_abnormal_returns_post / (std_pre_event_log_returns * np.sqrt(11))
p_value_post = stats.t.sf(np.abs(t_stat_post), df=len(data_with_returns) - 1) * 2  # two-tailed p-value

# Compile the results into a DataFrame
results = pd.DataFrame({
    'Period': ['CAR Pre-Event (Sep 26 to Oct 7)', 'Abnormal Return on Event Day (Oct 10)', 'CAR Post-Event (Oct 11 to 25)'],
    'CAR/Abnormal Return': [cumulative_abnormal_returns_pre, abnormal_returns_event_day, cumulative_abnormal_returns_post],
    'T-Statistic': [t_stat_pre, 'N/A', t_stat_post],
    'P-Value': [p_value_pre, 'N/A', p_value_post]
})

results.set_index('Period', inplace=True)
print(results)
```

```
                                   CAR/Abnormal Return T-Statistic P-Value
Period
CAR Pre-Event (Sep 26 to Oct 7)                   0.01        0.40    0.69
Abnormal Return on Event Day (Oct 10)             0.00         N/A     N/A
CAR Post-Event (Oct 11 to 25)                     0.04        2.19    0.03
```

In [38]:
```
# Calculate CAR for the entire event window (September 23 to October 25)
entire_event_window_start = pd.to_datetime('2022-09-23')
entire_event_window_end = pd.to_datetime('2022-10-25')
days_in_entire_event_window = (entire_event_window_end - entire_event_window_start).days + 1

cumulative_abnormal_returns_entire_event = data_with_returns[(data_with_returns['Date'] >= entire_event_window_start) & (data_w

# Calculate the T-statistic for the entire event window
t_stat_entire_event = cumulative_abnormal_returns_entire_event / (std_pre_event_log_returns * np.sqrt(days_in_entire_event_wind

# Calculate the P-value for the entire event window
p_value_entire_event = stats.t.sf(np.abs(t_stat_entire_event), df=len(data_with_returns) - 1) * 2  # two-tailed p-value

# Add the results for the entire event window to the DataFrame
new_row = pd.DataFrame({
    'Period': ['CAR Entire Event (Sep 23 to Oct 25)'],
    'CAR/Abnormal Return': [cumulative_abnormal_returns_entire_event],
    'T-Statistic': [t_stat_entire_event],
    'P-Value': [p_value_entire_event]
}).set_index('Period')

results = pd.concat([results, new_row])
results['CAR/Abnormal Return'] = results['CAR/Abnormal Return'].apply(lambda x: '{:.4f}'.format(x) if isinstance(x, float) else
results['T-Statistic'] = results['T-Statistic'].apply(lambda x: '{:.4f}'.format(float(x)) if x != 'N/A' else x)
results['P-Value'] = results['P-Value'].apply(lambda x: '{:.4f}'.format(float(x)) if x != 'N/A' else x)

results
```

Out[38]:

Period	CAR/Abnormal Return	T-Statistic	P-Value
CAR Pre-Event (Sep 26 to Oct 7)	0.0068	0.4007	0.6887
Abnormal Return on Event Day (Oct 10)	0.0002	N/A	N/A
CAR Post-Event (Oct 11 to 25)	0.0388	2.1910	0.0286
CAR Entire Event (Sep 23 to Oct 25)	0.0457	1.4913	0.1361

We convert our abnormal log returns into percentage returns.

In [39]:
```
results['CAR/Abnormal Return'] = results['CAR/Abnormal Return'].apply(lambda x: (np.exp(float(x)) - 1) * 100 if x != 'N/A' else

# Re-formatting to 4 significant figures after conversion
```

```
results['CAR/Abnormal Return'] = results['CAR/Abnormal Return'].apply(lambda x: '{:.4f}'.format(x) if isinstance(x, float) else

results
```

Out[39]:

Period	CAR/Abnormal Return	T-Statistic	P-Value
CAR Pre-Event (Sep 26 to Oct 7)	0.6823	0.4007	0.6887
Abnormal Return on Event Day (Oct 10)	0.0200	N/A	N/A
CAR Post-Event (Oct 11 to 25)	3.9563	2.1910	0.0286
CAR Entire Event (Sep 23 to Oct 25)	4.6760	1.4913	0.1361

GARCH and Long Range Dependence testing for 2a

We conduct this testing at the bottom and not in the 2a section because we introduce log returns later on.

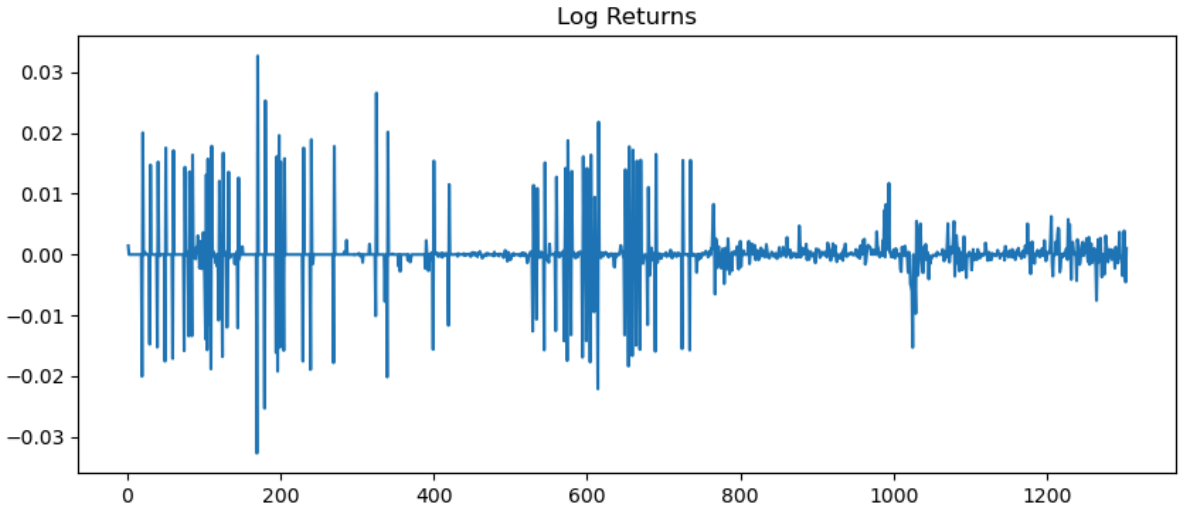
In [40]:
```
# GARCH
from arch import arch_model
from statsmodels.stats.diagnostic import het_arch

test_result = het_arch(log_returns)
print('Test statistic:', test_result[0])
print('P-value:', test_result[1])
```

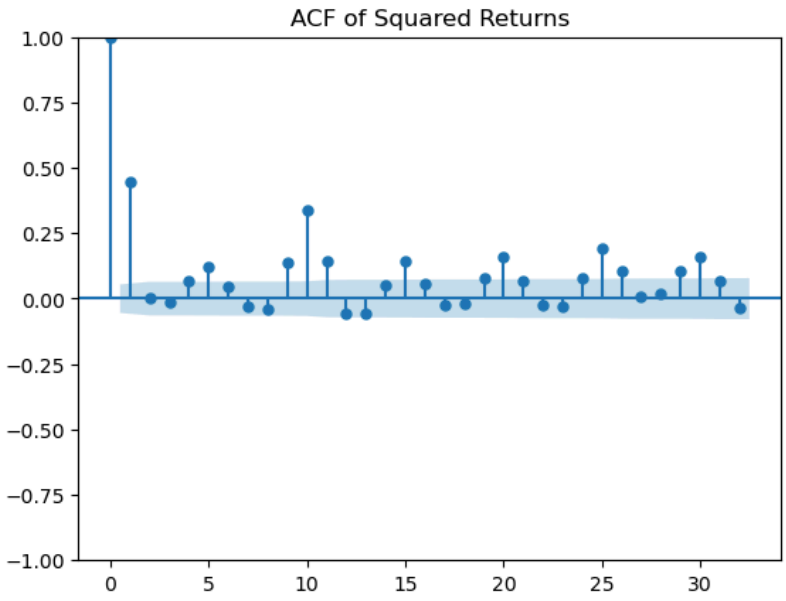
```
Test statistic: 428.911539353894
P-value: 6.551192607382862e-86
```

In [41]:
```
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf

# Check for stationarity
adf_result = adfuller(log_returns)
print(f'ADF Statistic: {adf_result[0]}')
print(f'p-value: {adf_result[1]}')
for key, value in adf_result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')

# Check for signs of volatility clustering
plt.figure(figsize=(10, 4))
plt.plot(log_returns)
```

```
plt.title('Log Returns')
plt.show()

# ACF plot of squared returns to check for autocorrelation
squared_returns = log_returns**2
plot_acf(squared_returns)
plt.title('ACF of Squared Returns')
plt.show()
```

```
ADF Statistic: -9.020530300889867
p-value: 5.7918488744905984e-15
Critial Values:
   1%, -3.435437251933509
Critial Values:
   5%, -2.863786592704128
Critial Values:
   10%, -2.567966103183712
```





In [42]:

```
# GARCH

from arch import arch_model

# Define the range of p and q to explore
p_range = range(1, 10)
q_range = range(1, 10)

results = []

for p in p_range:
    for q in q_range:
        try:
            # Specify the GARCH model with a constant mean, GARCH volatility, and Student's t errors
            model = arch_model(log_returns, mean='Constant', vol='Garch', p=p, q=q, dist='t')
            model_fit = model.fit(disp='off')  # 'disp="off"' turns off the convergence messages
            results.append({'p': p, 'q': q, 'AIC': model_fit.aic, 'BIC': model_fit.bic})
        except:
```

```
            print(f"Model with p={p} and q={q} failed to converge")

# Convert the results to a DataFrame and find the combination of p and q with the lowest AIC
results_df = pd.DataFrame(results)
best_aic_model = results_df.loc[results_df['AIC'].idxmin()]
best_bic_model = results_df.loc[results_df['BIC'].idxmin()]

print("Best model based on AIC:")
print(best_aic_model)
print("\nBest model based on BIC:")
print(best_bic_model)
```

```
/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
```

```
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:
```

```
The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.
```

```
/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 8. The message is:
Positive directional derivative for linesearch
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:
```

```
The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.
```

```
/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
```

```
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.
```

```
/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
```

```
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:
```

```
The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.
```

```
/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:
```

```
The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.
```

```
Best model based on AIC:
p           1.00
q           4.00
AIC   -12,011.35
BIC   -11,969.97
Name: 3, dtype: float64

Best model based on BIC:
p           1.00
q           4.00
AIC   -12,011.35
BIC   -11,969.97
Name: 3, dtype: float64
```

```
/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.


/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.
```

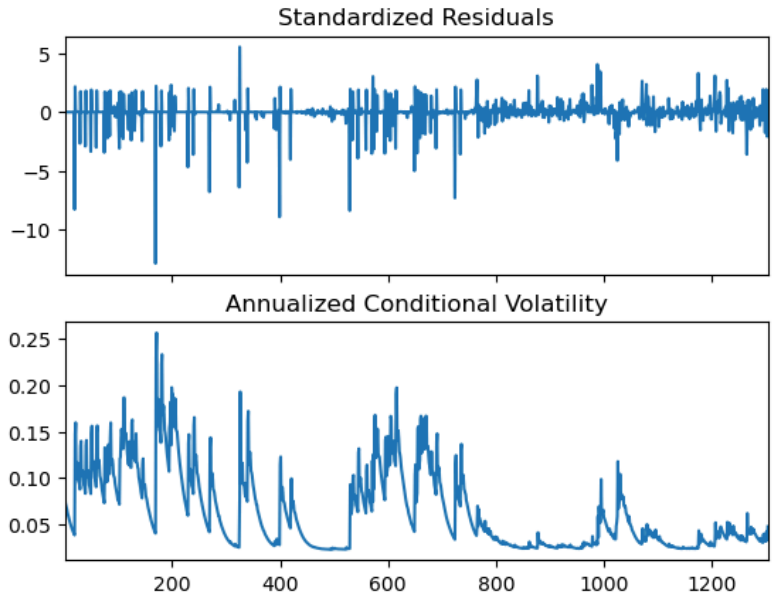
In [43]:

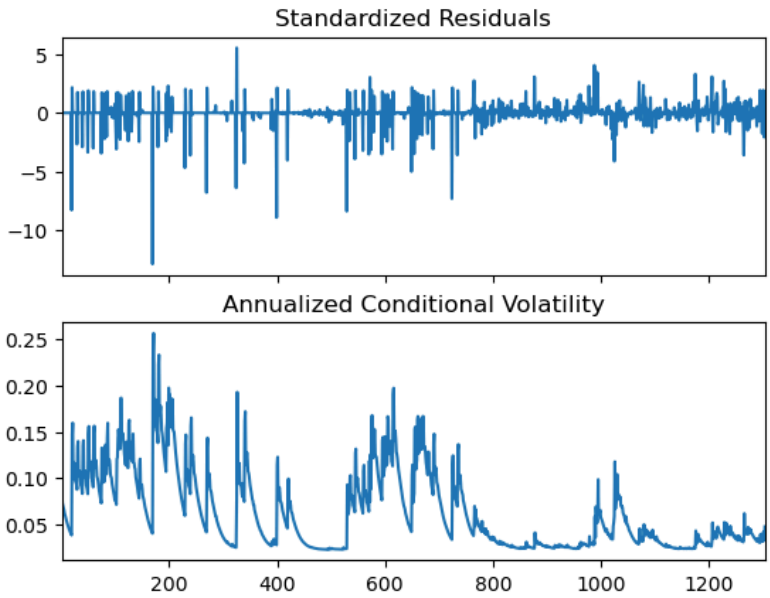
```python
from arch import arch_model

model = arch_model(log_returns, mean='Constant', vol='Garch', p=1, q=4, dist='t')

# Fit the model
model_fit = model.fit(update_freq=5)  # update_freq controls the frequency of output during estimation

# Display the fitting summary
print(model_fit.summary())

model_fit.plot(annualize='D')
```

```
Inequality constraints incompatible    (Exit mode 4)
            Current function value: -6013.674912635219
            Iterations: 1
            Function evaluations: 9
            Gradient evaluations: 1
                      Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                      Close   R-squared:                       0.000
Mean Model:                  Constant Mean   Adj. R-squared:                  0.000
Vol Model:                           GARCH   Log-Likelihood:                6013.67
Distribution:      Standardized Student's t   AIC:                           -12011.3
Method:                 Maximum Likelihood   BIC:                           -11970.0
                                             No. Observations:                 1303
Date:                      Sat, Mar 23 2024   Df Residuals:                     1302
Time:                              17:45:15   Df Model:                            1
                                 Mean Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
mu         3.5612e-05  1.926e-05      1.849  6.440e-02 [-2.130e-06,7.335e-05]
                           Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega      4.5289e-07  9.859e-09     45.936      0.000 [4.336e-07,4.722e-07]
alpha[1]       0.2000  3.822e-02      5.232  1.673e-07     [  0.125,  0.275]
beta[1]        0.1950      0.346      0.564      0.573     [ -0.483,  0.873]
beta[2]        0.1950      0.656      0.297      0.766     [ -1.091,  1.481]
beta[3]        0.1950      0.583      0.335      0.738     [ -0.947,  1.337]
beta[4]        0.1950      0.167      1.169      0.242     [ -0.132,  0.522]
                              Distribution
============================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
----------------------------------------------------------------------------
nu             4.1762  7.977e-02     52.349      0.000 [  4.020,  4.333]
============================================================================

Covariance estimator: robust
WARNING: The optimizer did not indicate successful convergence. The message was Inequality constraints incompatible.
See convergence_flag.
```

```
/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 4. The message is:
Inequality constraints incompatible
See scipy.optimize.fmin_slsqp for code meaning.
```

Out[43]:





Long-range dependence testing

In [44]:
```
from arch import arch_model
from statsmodels.stats.diagnostic import acorr_ljungbox

model = arch_model(Close, mean='Constant', vol='Garch', p=1, q=4, dist='t')
model_fit = model.fit(disp='off')  # Setting disp='off' to silence output

# Get the squared residuals (which represent the variance)
squared_residuals = model_fit.resid**2

# Apply the Ljung-Box test on the squared residuals
ljung_box_results = acorr_ljungbox(squared_residuals, lags=[100], model_df=1, return_df=True)

print(ljung_box_results)
```

```
      lb_stat  lb_pvalue
100 23,522.33       0.00
```

```
/Users/alexandrcarr/anaconda3/lib/python3.11/site-packages/arch/univariate/base.py:766: ConvergenceWarning:

The optimizer returned code 8. The message is:
Positive directional derivative for linesearch
See scipy.optimize.fmin_slsqp for code meaning.
```