Comparison of different types of volatility models for NIFTY50 index

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Mathematics
by

Tushar Arora
20202022

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2023

Supervisor: Dr. Aniruddha Pant

Certificate

This is to certify that this dissertation entitled **Comparison of different types of volatility models for NIFTY50 index** towards the partial fulfilment of the MS(by Dissertation) degree programme at the Indian Institute of Science Education & Research, Pune represents study/work carried out by Tushar Arora at Indian Institute of Science Education & Research, Pune under the supervision of Dr. Aniruddha Pant, CEO, AlgoAnalytics Pvt. Ltd, Pune during the academic year 2022-2023.

[illegible] Type text here

Supervisor's signature
Dr. Aniruddha Pant

Committee:

Supervisor: Dr. Aniruddha Pant
Local Guide: Dr. Anindya Goswami

To my Family and Friends

Declaration

I hereby declare that the matter embodied in the report entitled **Comparison of different types of volatility models for NIFTY50 index** are the results of the work carried out by me at the Department of Mathematics, Indian Institute of Science Education & Research, Pune, under the supervision of **Dr. Aniruddha Pant** and the same has not been submitted elsewhere for any other degree.

[illegible]

Tushar Arora

Date: 30.04.2023

Acknowledgements

I would like to express my deepest gratitude to my guide, Dr. Aniruddha Pant, whose expertise, understanding, and guidance have made this thesis possible. His encouragement and constant feedback were instrumental in the successful completion of this work.

I would also like to extend my heartfelt appreciation to my local guide, Dr. Anindya Goswami. The courses he taught me and the previous project I undertook under his supervision significantly contributed to my ability to face the challenges in this project. His valuable insights, unwavering support, and dedication to my academic growth have been truly inspiring and played a vital role in the completion of my thesis.

My sincere thanks go to my family and friends, who have always been there for me and supported me unconditionally. Their love and encouragement have been a constant source of strength and motivation.

Abstract

In this thesis, volatility from different models are compared using NIFTY50 index data. In the first half of thesis, we build our understanding of volatility and it's different types. Then, we move to understanding volatility clustering using autocorrelation. To capture the effect of volatility clustering we discuss univariate models like AR, ARMA and GARCH etc. After that, we try to model volatility using Hidden Markov Switching(HMS) model and GARCH model. To know which of the model performs better, we forecast volatility using both the models and compare them using a true volatility indicator, India VIX index.

In the second part of my thesis, we developed a pair trading strategy for NIFTY50 and BANKNIFTY futures using the concepts of stationarity, mean-reversion and correlation. Using HMS model to classify the market into regimes, we try to analyze the performance of our pairs strategy in binary regimes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's finance world, it's really important to have a good knowledge and understanding of volatility. One can't become a good trader, researcher, risk analyst or any other position in the Quantitative finance world without having a clear understanding of volatility and concepts related to it. A proper understanding of this concept can be beneficial to an extent one can't imagine. Volatility, as the name suggests is an indicator of risk and risk now a days is one of the booming topics in the finance world because of COVID, recession etc. These days, there are a lot of regulations on banks (international as well as national) and they can't function properly without having a good risk department. Volatility as a concept finds it's applications mostly that are related to risk.

Volatility can be computed in various ways. And there are different types of volatility like Historical volatility(HV), Implied volatility(IV), beta, drawdown and realized volatility etc. Each one has it's own way of computation and each one indicated volatility in a different manner. Like HV is an indicator of past volatility and is computed using historical data of an asset. IV is computed using the price of an option and is an indicator of future volatility. Beta indicates relative volatility with respect to the market and is computed using linear regression. A lot of concepts arise from volatility such as volatility smile, volatility skew and volatility clustering etc. In this thesis, we will only study volatility clustering in deep.

Computing different types of volatility is fine but modeling it is a totally different task. But before going forward one may ask, what is the need of modeling volatility? The answer is pretty simple. If one have a volatility model, one can make important predictions from it and use them in their strategy. So, forecasting volatility is one of the main reasons of modeling volatility [13]. There has been a lot of research related to volatility modeling [8] [11] [17] [19] . Engle in 1982 described a model called Autoregressive Conditional Heteroskedasticity(ARCH) [11] which is a univariate model that models the variance of the error as an Autoregressive(AR) [18] process. This model was defined to take care of the persistence in volatility. Generalized Autoregressive Conditional Heteroskedasticity(GARCH) [8] is pretty much similar to ARCH with the exception that the variance of the error term is modelled as an Autoregressive Moving Average(ARMA) process. From then a lot of variants of ARCH and GARCH [7] have come up like EGARCH [17], IGARCH, ARCH-M[12], T-GARCH [19] etc. Hidden Markov Switching(HMS) models were first developed in . These models use markov chains to classify returns into different regimes by calculating the probability of being in that volatility state. Heston [15] [20] in 2005 developed a stochastic volatility model with the assumption of volatility being stochastic instead of constant in Black-Scholes Model [6]. It models volatility as a Cox-Ingersoll-Ross(CIR) [9] process.

Although, there is a good literature available on modelling volatility but there has not been much work done in the area of comparing different types of volatility models. Hansen and Lunde in 2005 [14] did some work on comparing different models with GARCH(1,1) and concluded that GARCH(1,1) is over performing every other model. Amskold in 2011 [1] compared implied volatility coming from different models in his thesis. Lim in 2013 [16] compared the performance of GARCH-type models for Malaysian markets. Zhu in 2018 [21] compared 3 types of volatility forecasting models.

In this thesis, we will first model volatility for NIFTY50 index data using models like HMS, GARCH etc. and then compare the volatility forecasted by these models with true volatility indicator. We start by understanding with the basics of volatility, its different types and volatility clustering in Chapter 2. Chapter 3 deals with how ARCH, GARCH explain volatility clustering and working of HMS models. In Chapter 4, we look into how volatility forecasting is done using these models and compare them graphically and statistically with true volatility indicator. Chapter 5 deals with understanding the impact of different regimes on a pairs trading strategy. Finally we present all our findings, results and conclusions in Chapter 6.

Chapter 2

Volatility

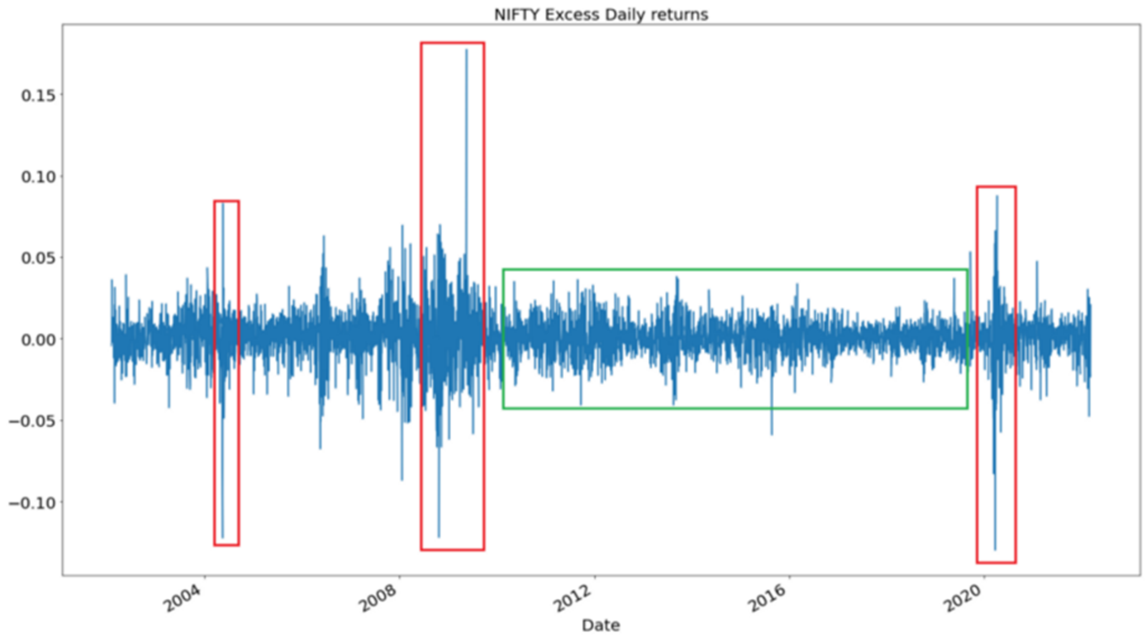


Figure 2.1: NIFTY daily returns from Feb 2002 to March 2022

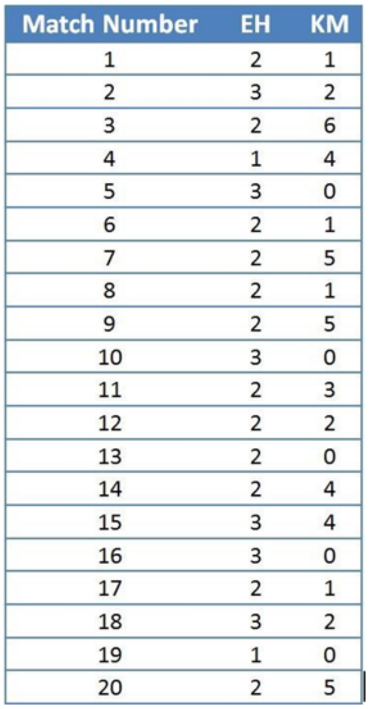
Let us look closely at this graph and try to think why some regions are encircled with red and some with green? Can you think of a reason why these regions are different just by looking at this graph? Does there exist a concept that can explain these different regions? The answers to all these questions are related to Volatility. So, let's begin by understanding Volatility.

2.1 What is Volatility?

Volatility in most simple terms is a measure of risk. It is very much related to what we all know from statistics as variance. High volatility indicates large fluctuations in asset price during short amount of time. Therefore, high volatility implies higher risk. Let's try to understand this term from an example completely unrelated to stock market.

2.1.1 Volatility and Variance

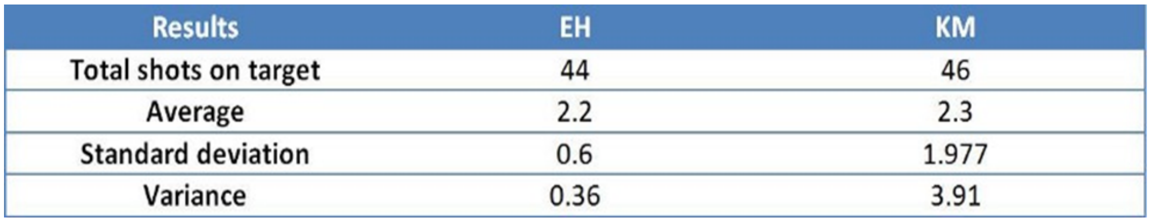
Suppose, there are 2 forward players: EH and KM (Does these initials sound familiar?). We want to choose a forward who can give at least 2 shots on target (more dependable or less risky) in a match. The data available to us is given below:



Match Number	EH	KM
1	2	1
2	3	2
3	2	6
4	1	4
5	3	0
6	2	1
7	2	5
8	2	1
9	2	5
10	3	0
11	2	3
12	2	2
13	2	0
14	2	4
15	3	4
16	3	0
17	2	1
18	3	2
19	1	0
20	2	5

Figure 2.2: Shots on target in last 20 matches

Using the above table we calculate the following results:



Results	EH	KM
Total shots on target	44	46
Average	2.2	2.3
Standard deviation	0.6	1.977
Variance	0.36	3.91

Figure 2.3: Basic stats of the data

If we just look at the first 2 rows of the above table, it feels like it is better to choose KM as our forward. But the whole point of this example is Volatility. We know that standard deviation is defined as the deviation from the mean. So, it is expected (with high probability) that the number of shots on target for EH will lie in the range (2.2–0.6, 2.2 + 0.6) = (1.6,2.8). Similarly, it is expected (with high probability) that the number of shots on target for KM will lie in the range (2.3–1.977,2.3+1.977) = (0.323,4.277). The above calculated intervals shows that if we want a forward who can give at least 2 shots on target, then we should go with EH because less standard deviation (or volatility) implies lesser risk. (For football fans, are you convinced by the above arguments and believe that EH is better forward than KM? I don't think you should because of the sole reason that the above data is made up and doesn't contain actual data for both players. Maybe we can compare EH and KM with actual data another time).

Now, coming back to the questions we asked in the start of this article and keeping in mind the above example of 2 forwards in football, we can deduce that the red marked regions indicate highly volatile (or variable) region because of these large deviations from the mean zero. Similarly, the green regions indicate low volatile region because the deviations from the mean zero are pretty small when compared to red regions. So, one should try to avoid trading in highly volatile regions (Although, there are some traders who secure large amounts of profit in these times) due to the risk and unpredictability present in the market during these times. This section is taken from a blog [5] which I wrote for AlgoAnalytics.

2.2 Different types of Volatility

2.2.1 Historical Volatility

Historical volatility(HV) which is sometimes also known as the statistical volatility, is calculated by computing the standard deviation using the past returns of the asset. The name "historical" comes from the fact that we are using past data to calculate the volatility. One of the main drawbacks of HV is that it gives no indication of what is going to happen in future and only takes care of what the past volatility was.

2.2.2 Implied Volatility

Implied volatility(IV), as the name suggests is forward looking. It gives an estimate of what future volatility is going to be and is calculated using the price of an option. IV is also used in calculating the price of an option in Black-Scholes Model.

2.2.3 Beta

Beta is a measure of relative volatility which measures how volatile an asset is compared to market. By market here we mean some index like NIFTY50, BANKNIFTY etc. Beta value for market is taken to be 1. It is calculated using Linear Regression technique where dependent variable is Asset returns and independent variable is Index returns. Slope given by the Linear Regression is our Beta. A beta value of more than 1 is an indicator of high volatility and a beta value of less than 1 is indicator of low volatility.

2.3 Volatility Clustering

Now that we have a basic understanding of Volatility and what it represents, let's jump to the new concept: Volatility Clustering.

Again we come back to the same graph 3.1 in which NIFTY returns are plotted. Notice that there are periods of high volatility and low volatility. We can't find a period in the graph where we see fluctuations from low volatility to high volatility or high volatility to low volatility in a short amount of time. Why do you think this happens ? The answer lies in Volatility Clustering.

2.3.1 What is Volatility Clustering?

Volatility Clustering in most basic terms is persistence in volatility. What do we mean by persistence in volatility? It means that there will be periods of high volatility and periods of low volatility. Mandelbrot in his paper on "The Variation of Certain Speculative Prices" wrote a very nice observation which can be used to explain volatility clustering. The observation was "large changes are tend to be followed by large changes, of either sign, and small changes are tend to be followed by small changes."

2.3.2 Understanding Volatility clustering through Autocorrelation function

So, we understood volatility clustering and were able to explain it using the above graph. But understanding it pictorially is not enough, we have to find some mathematical evidence to support our observation. Let's plot another graph.

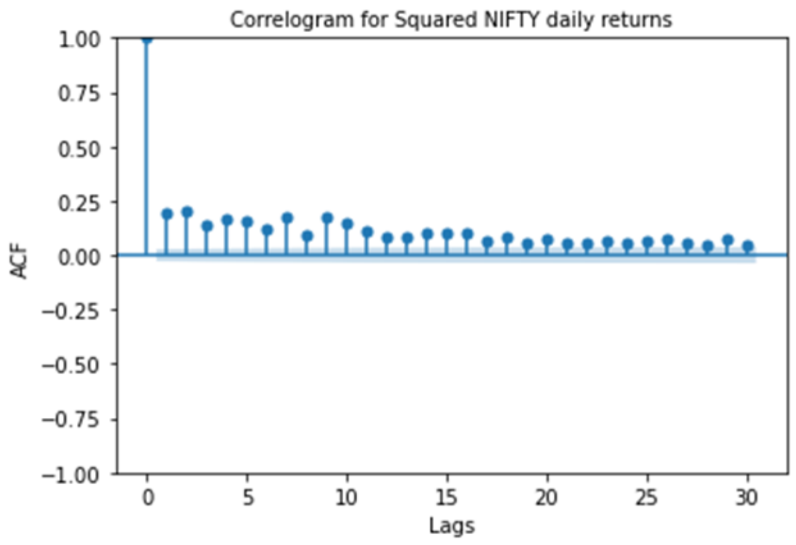


Figure 2.4: Autocorrelation in squared NIFTY daily returns at different lags

What can we understand from this graph and how is it related with volatility clustering ? Let's try to note some observations.

1. It shows that $\text{cor}(r_t^2,\ r_{t+lg}^2) > 0$ for all lg (lags) $= 1,\ 2,\ \ldots\ ,\ 30$ where r_t represents NIFTY daily return and cor represents Pearson correlation coefficient.

2. $\text{cor}(r_t^2,\ r_{t+lg}^2)$ is significant (the value at every lag is above the light blue shaded region) for every lag.

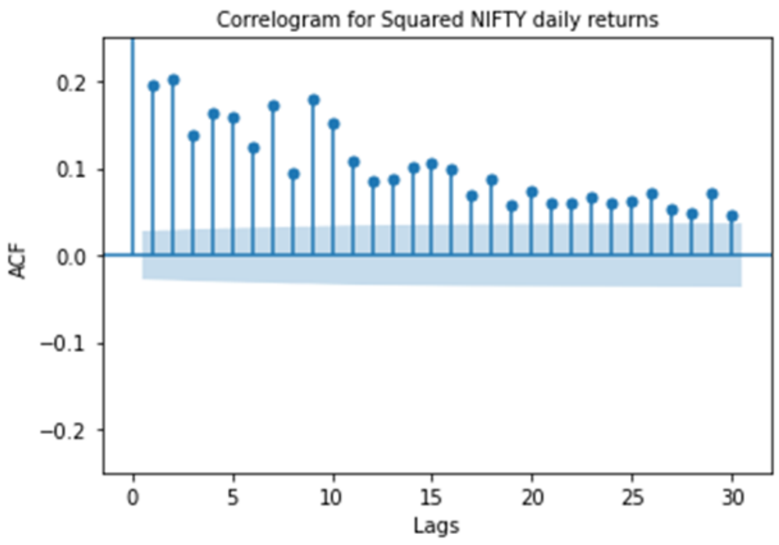


Figure 2.5: Significant autocorrelation in squared NIFTY daily returns at different lags

How does these 2 observations relate to Volatility clustering? Think of what r_t^2 represent. Looking at NIFTY daily return series, it can be assumed that mean of the series is very close to zero. Then, r_t^2 will represent variance. Now, from the previous article we can see volatility coming in the picture. So, positive and significant correlation in r_t^2 & r_{t+lg}^2 indicates that volatility at time t is positively correlated with volatility at time t+lg which provides the explanation for persistence of volatility. This section is taken from a blog [4] which I wrote for AlgoAnalytics.

Chapter 3

Modelling Volatility

In the last articles, we have learned about volatility and volatility clustering. If we just go back to the figure from where we started and take a look it again.

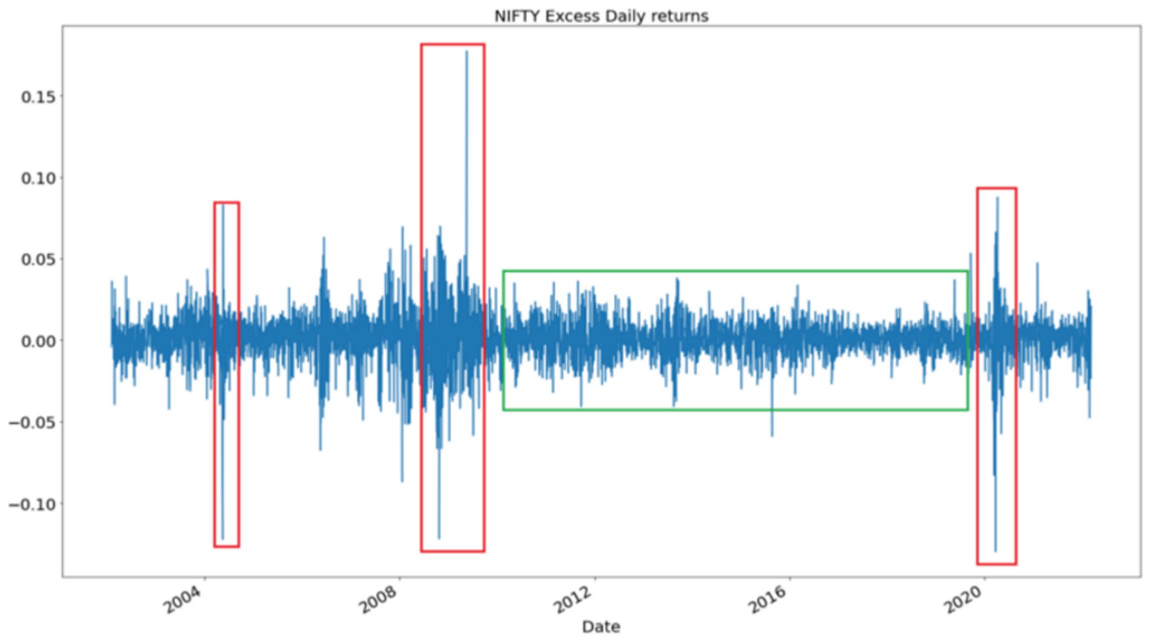


Figure 3.1: NIFTY daily returns from Feb 2002 to March 2022

Remember in the section "What is volatility?", we said that these red encircled regions are indicating high volatility and green regions are indicating low volatility. Keeping this in mind let us first try to understand Hidden Markov Switching Model.

3.1 Hidden Markov Switching Model(HMS)

Keeping the name of the model in mind, let us first try to understand Markov property and Markov Chain.

3.1.1 Markov Property

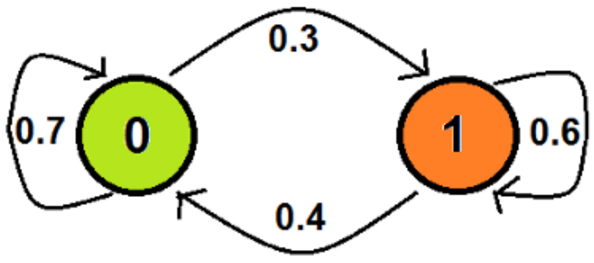
Markov property says that “Future is independent of the Past given the Present” which in time sense would translate to “If we know all the information about today, then whatever is going to happen tomorrow is not dependent on what happened yesterday”.

$$P(Y_t = y|Y_{t-1} = y_1, Y_{t-2} = y_2) = P(Y_t = y|Y_{t-1} = y_1) \tag{3.1}$$

The above equation explains the Markovian property for discrete time markov chains. Here, you saw that a stochastic variable(Y_t) can take these discrete values. But at any particular moment (time t) it can only take one of the available discrete values. So, when a stochastic variable changes its state(value) from one to another, we come to the realm of Markov Chains.

3.1.2 Markov Chain

Markov Chain is a stochastic process describing the transitions of a stochastic variable. Consider this simple Markov Chain :



$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0.7 & 0.3 \\ 1 & 0.4 & 0.6 \end{array}$$

Figure 3.2: 2 State Discrete Time Markov Chain with it's Transition Probability Matrix (TPM)

The above image shows the diagram of the Markov Chain and its Transition Probability Matrix(TPM). Although from the diagram and matrix it is pretty clear what is happening. But for more clarity, let us try to understand by an example what these probabilities represent. Probability that the next state is 1 given the current state is 0 is 0.3 which can also be interpreted as Probability of transitioning to state 1 from state 0. Now that we have a basic understanding of Markov Chain, let us try to answer the question of how Markov Chains are related to volatility?

3.1.3 Markov Chains and Volatility

In the simplest case(see the red, green regions in the picture) we can say that volatility has 2 states or regimes (Low, High). So, considering volatility as a stochastic variable, we can model volatility using Markov chains. Because of switching of volatility between low to high or high to low, these models are sometimes known as "Markov Switching Models". Another thing to note is that these volatility states can't be directly observed and are hidden, what can be observed are the returns. The returns are influenced by these hidden states. That's why these models are also called "Hidden Markov Switching Models" (HMS models).

Let's try to understand it mathematically.

We all know that asset price follows Geometric Brownian motion given by:

$$\frac{dX_t}{X_t^-} = \mu t + \gamma_t dW_t \tag{3.2}$$

where X_t denotes asset price, μ drift, γ_t volatility and W_t represents standard Brownian motion. On discretizing and using the properties of Brownian motion, we get:

$$r_i = \mu\delta + \gamma_t\sqrt{\delta}Z_i \tag{3.3}$$

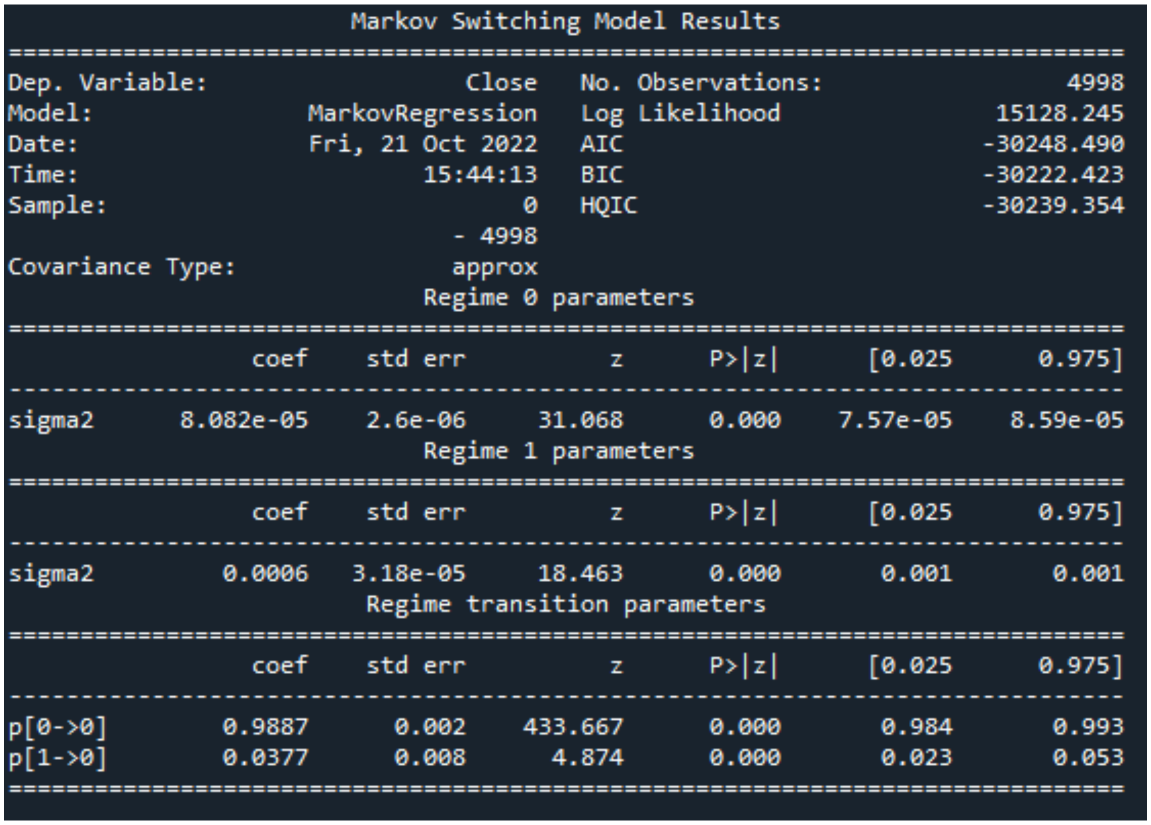
where $r_i = \dfrac{S_i - S_{i-1}}{S_{i-1}}$, $\delta = t_i - t_{i-1}$ and $Z_i \sim \mathcal{N}(0, 1)$. Since, drift μ is an indicator of mean and in our case we know that mean is very close of zero. So, we can assume that μ is zero.

$$r_i = \gamma_t\sqrt{\delta}Z_i \text{ and } Var(r_i) = \delta\gamma_t^2 \tag{3.4}$$

And, γ_t (volatility) here is a Stochastic variable with 2 states. From the above line, you can see that variance of return is coming out to be equal to a stochastic variable having two states. And variance of return is an indicator of volatility, which means that volatility has different states.

3.1.4 Implementing HMS model in Python

Now that we have an understanding of HMS models, let's look at it's implementation in Python. The code for it's implementation is attached in Appendix A. Here, we have used MarkovRegression function from the statsmodels.api package to implement HMS model. Now, let us look at the summary of fitted HMS model and try to make some observations.



```
                         Markov Switching Model Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                 4998
Model:               MarkovRegression   Log Likelihood               15128.245
Date:                Fri, 21 Oct 2022   AIC                         -30248.490
Time:                        15:44:13   BIC                         -30222.423
Sample:                             0   HQIC                        -30239.354
                               - 4998
Covariance Type:               approx
                             Regime 0 parameters
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2      8.082e-05    2.6e-06     31.068      0.000    7.57e-05    8.59e-05
                             Regime 1 parameters
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2         0.0006   3.18e-05     18.463      0.000       0.001       0.001
                         Regime transition parameters
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
p[0->0]        0.9887      0.002    433.667      0.000       0.984       0.993
p[1->0]        0.0377      0.008      4.874      0.000       0.023       0.053
==============================================================================
```

Figure 3.3: Summary of the fitted HMS model

Expectation-Maximation algorithm has been used to find all the parameters. This summary contains a lot of information but some of important things in this summary are the transition probabilities and regime parameters which we will be using in the next chapter to forecast volatility. This section is inspired from a blog [2] which I wrote for AlgoAnalytics.

3.2 Autoregressive model: AR

The coming sections are a part of a blog [3] which I wrote for AlgoAnalytics. We start our journey with some basic models understanding their motivations and drawbacks. First is Autoregressive(AR) model.

Autoregressive(AR) model: The main idea behind this model is that the value of the series at time t is linearly dependent on the past values of the series(that's why the term Autoregressive) plus some noise. These models capture the mean-reversion & momentum trends in stock-price dynamics. Autoregressive model (with zero mean) of order p or AR(p) is given by:

$$y_t = \sum_{i=1}^{p} a_i y_{t-i} + \eta_t \tag{3.5}$$

In the above equation, p represents the lag order, η_t is discrete white noise and $a_i \in \mathbb{R}$ with $a_p \neq 0$. Now, let us try to fit appropriate AR model to NIFTY daily returns and analyze the results. The parameters of fitted AR model are calculated by minimizing Akaike Information Criteria(AIC). For more details on AIC, please check section 4.2.1.

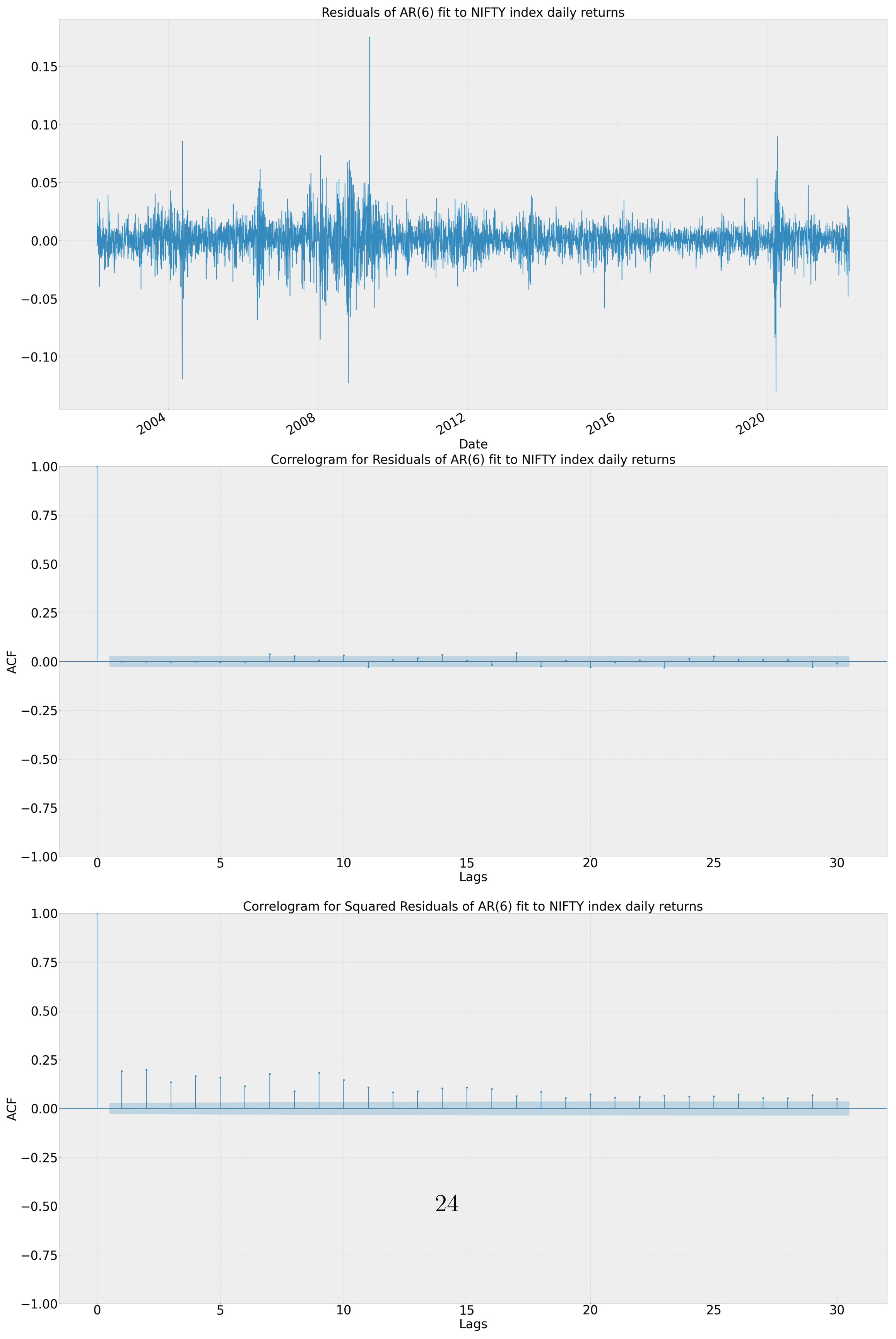


Figure 3.4: Autocorrelation graphs after fitting AR(6) model to NIFTY daily returns

Looking at the above figure, we note some observations:

1. The middle graph shows insignificant autocorrelation at almost all lags except few. Insignificant autocorrelation here indicates that there is no serial correlation (autocorrelation) present in the series.

2. Last graph provides an evidence of AR model being not able to explain volatility clustering. Also, it is not able to explain black swan events. Black swan event is a market event which has very very less probability of occurrence(often six standard deviations away from the mean) and has catastrophic impact.

Next we have is Moving Average(MA) model.

3.3 Moving Average model: MA

Moving Average(MA) model: This model captures the unexpected (which are less likely to happen) events by taking a linear combination of past values of noise. Moving Average model(with zero mean) of order q or MA(q) is given by:

$$y_t = \sum_{i=1}^{p} b_i \eta_{t-i} + \eta_t \tag{3.6}$$

In the above equation, q represents the lag order, η_t is discrete white noise and $b_i \in \mathbb{R}$ with $b_q \neq 0$. Now, let us try to fit appropriate MA model to NIFTY daily returns and analyze the results. The parameters of fitted MA model are calculated by minimizing Akaike Information Criteria(AIC).

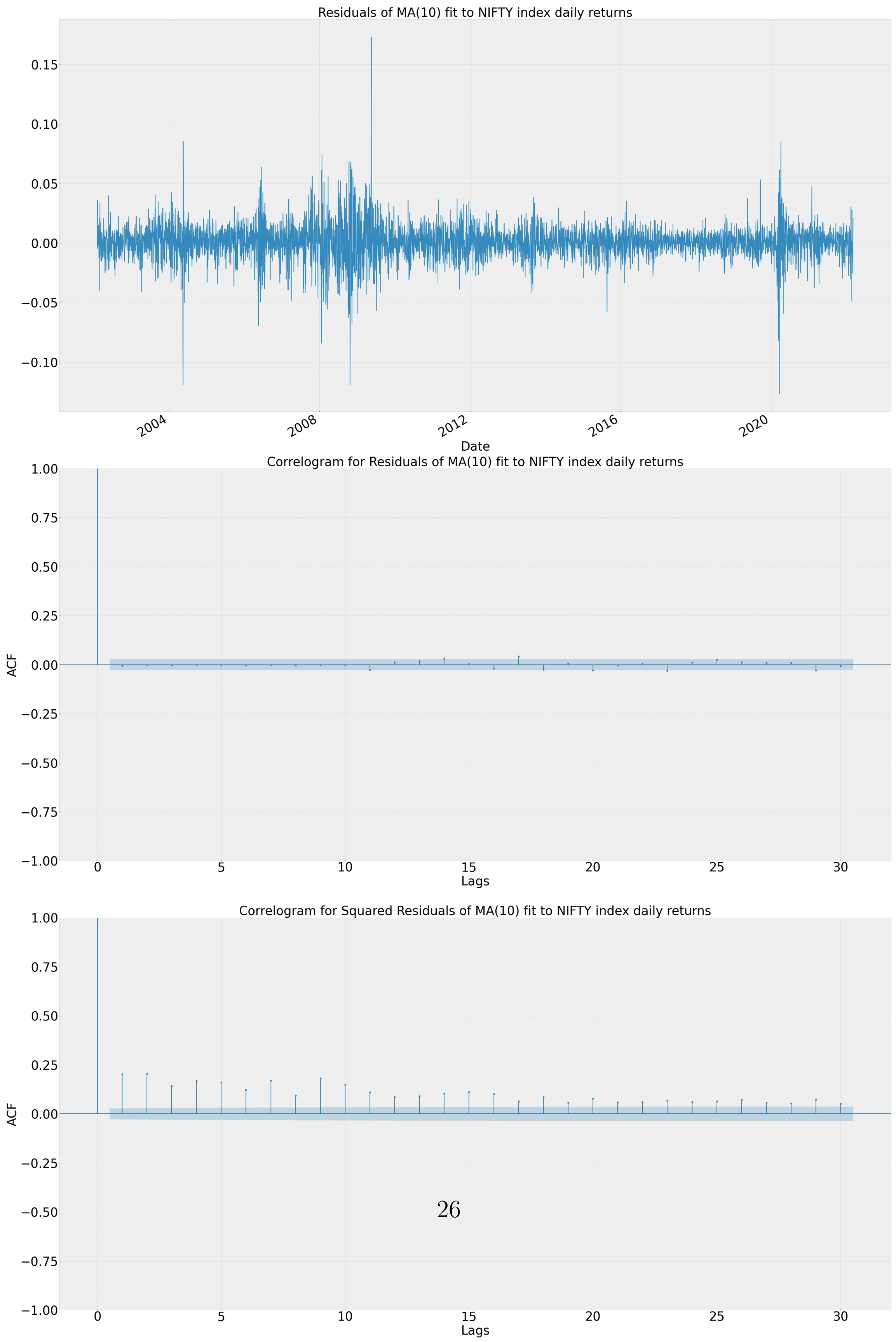


Figure 3.5: Autocorrelation graphs after fitting MA(10) model to NIFTY daily returns

Again, we see insignificant autocorrelation in the middle graph but in the last graph, there is positive and significant autocorrelation in the squared residuals indicating that this model is not able to explain volatility clustering. Next in our store is Autoregressive Moving Average(ARMA) model.

3.4 ARMA model

ARMA model: It combines the above two models (AR(p) and MA(q)) to capture both their properties in a single model. One important use of ARMA model is that it usually requires less parameters than AR or MA model alone. Autoregressive Moving Average model(with zero mean) of order (p, q) or ARMA(p, q) is given by:

$$y_t = \sum_{i=1}^{p} a_i y_{t-i} + \eta_t + \sum_{i=1}^{p} b_i \eta_{t-i} \tag{3.7}$$

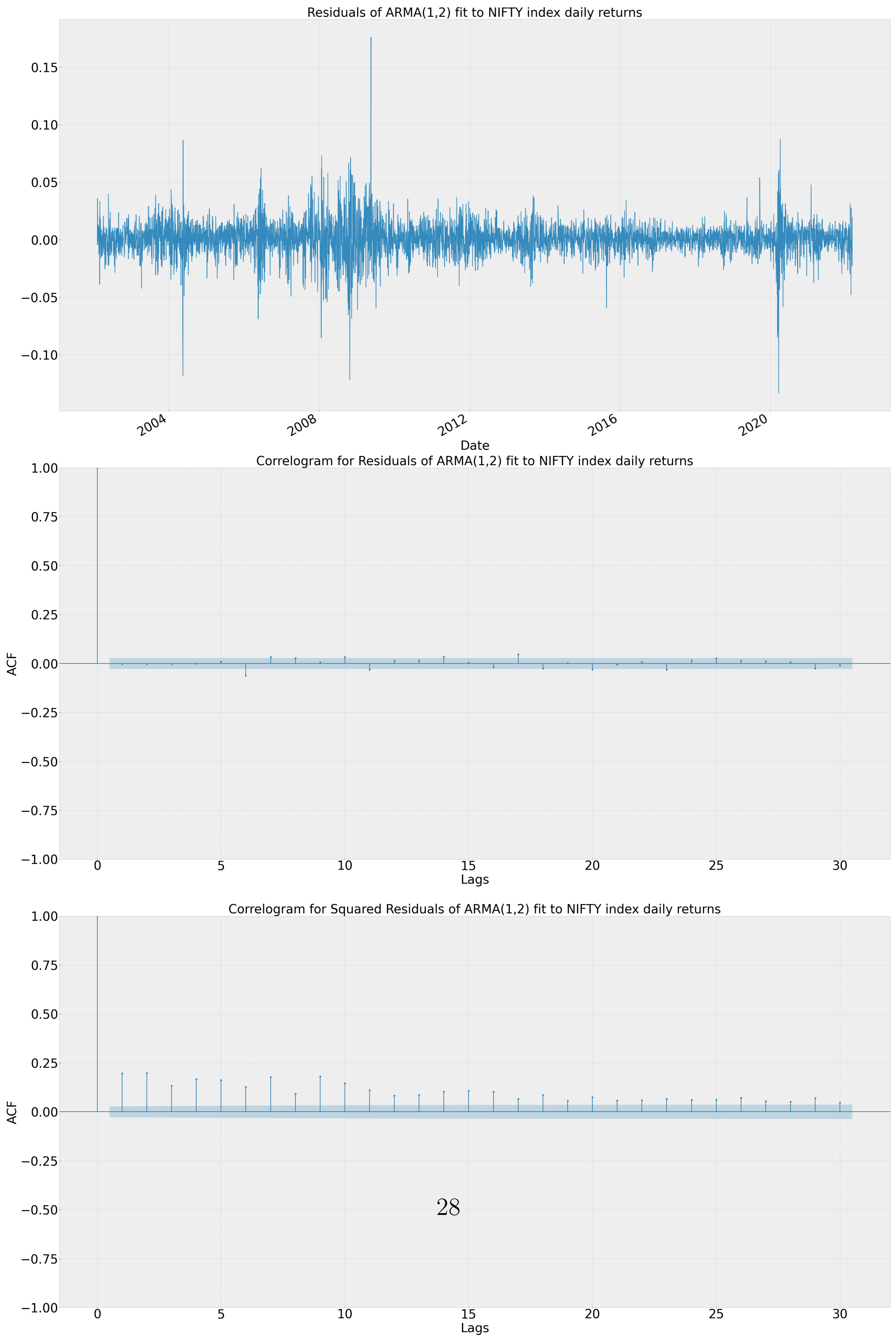


Figure 3.6: Autocorrelation graphs after fitting ARMA(1,2) model to NIFTY daily returns

The parameters of fitted ARMA model are also calculated by minimizing Akaike Information Criteria(AIC). We conclude that although AR(p), MA(q) and ARMA(p, q) are able to explain no autocorrelation in the series but none of them could capture the volatility clustering effect. So, we need another type of models that can explain persistence in volatility. And here comes ARCH and GARCH in our picture.

3.5 ARIMA model

Before going to ARCH and GARCH, let us briefly understand the ARIMA model. ARIMA is Autoregressive Integrated Moving Average model, a generalization of ARMA model which are applied to non stationary data instead of stationary data. The word integrated in ARIMA comes from the fact that this univariate time series model can be applied to stationary as well as non-stationary data. Here, differencing is used to convert a non-stationary series into stationary series. Later, we will see how ARIMA model becomes very handy in converting non-stationary series into stattionary series.

3.6 ARCH model

ARCH is Autoregressive Conditional Heteroskedasticity. We are already familiar with what autoregressive means. Then we move to Heteroskedasticity. Heteroskedasticity refers to the data with different variance (or standard deviation) when monitored over different time periods. Conditional Heteroskedasticity is when the varying variance is dependent on previous time periods. We can go back to the article on volatility clustering once and see the similarities in volatility clustering and conditional heteroskedasticity. Some people call conditional heteroskedasticity, a fancy (and statistical) term for volatility clustering. Now, let's understand ARCH model.

ARCH model: The idea behind this is to model variance of the series as an autoregressive process to explain volatility clustering. Let's try to understand it mathematically.

$$\eta_t = z_t \sqrt{A_0 + \sum_{i=1}^{p} A_i \eta_{t-i}^2} \tag{3.8}$$

In the above equation, p represents the lag order, $\{z_t\}$ is discrete white noise with mean 0 and variance 1, and $A_i \in \mathbb{R}^+ \cup \{0\}$ with $A_p \neq 0$.

3.6.1 How ARCH models Volatility clustering?

To understand how ARCH models volatility clustering, we start by calculating the variance of $\{\eta_t\}$.

$$\mathbb{E}(\eta_t) = \mathbb{E}\left(z_t \sqrt{A_0 + \sum_{i=1}^{p} A_i \eta_{t-i}^2} \right) = \mathbb{E}(z_t)\mathbb{E}\left(\sqrt{A_0 + \sum_{i=1}^{p} A_i \eta_{t-i}^2} \right) = 0$$

$$Var(\eta_t) = \mathbb{E}(\eta_t^2) - (\mathbb{E}(\eta_t))^2 = \mathbb{E}(\eta_t^2) = \mathbb{E}\Big(z_t^2 (A_0 + \sum_{i=1}^{p} A_i \eta_{t-i}^2) \Big)$$

$$Var(\eta_t) = \mathbb{E}(z_t^2)\mathbb{E}\Big(A_0 + \sum_{i=1}^{p} A_i \eta_{t-i}^2 \Big) = A_0 + \sum_{i=1}^{p} A_i Var(\eta_{t-i})$$

Let us try to understand the above calculations and what does it signify. In the first line, we are calculating mean of η_t using the fact that mean of z_t is 0 and z_t & η_{t-i} are independent. Using $\mathbb{E}(\eta_t) = 0$, we calculate Var(η_t) in the second and third line. Look closely at the last line and try to think of what it represents!!! It says that variance at time t is dependent on the previous variance terms and variance is directly related to volatility. This is what persistence of volatility means. So, mathematically ARCH model is able to explain volatility clustering. Now, Let us understand GARCH and check their autocorrelation graphs as well.

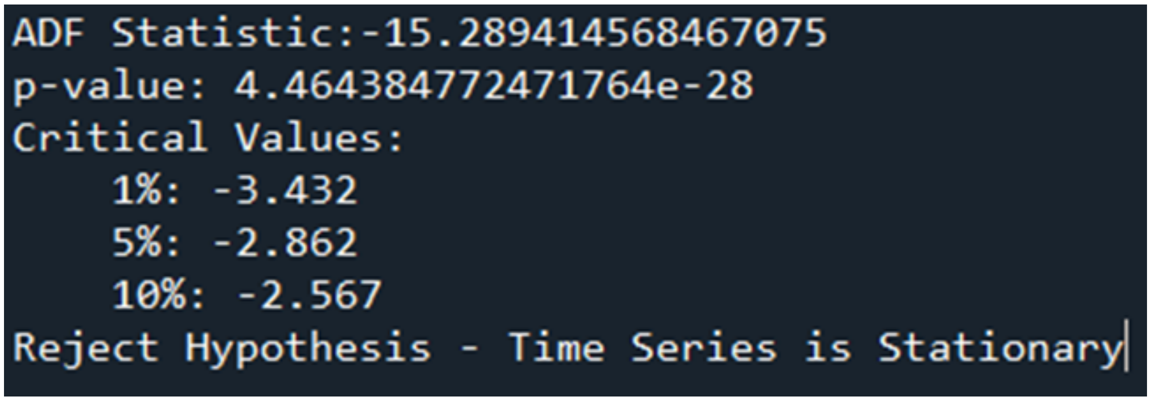
3.7 GARCH

After including autoregressive process in variance, the only thing left to add is the moving average which is the motivation for the GARCH model. GARCH model of order (p, q) is given by:

$$\eta_t = z_t \zeta_t \text{ where } \zeta_t^2 = A_0 + \sum_{i=1}^{p} A_i \eta_{t-i}^2 + \sum_{j=1}^{q} B_i \zeta_{t-j}^2 \tag{3.9}$$

$$\eta_t = z_t \sqrt{A_0 + \sum_{i=1}^{p} A_i \eta_{t-i}^2 + \sum_{j=1}^{q} B_i \zeta_{t-j}^2} \tag{3.10}$$

In the above equation, p, q represents the lag order, $\{z_t\}$ is discrete white noise with mean 0 and variance 1, $A_i \in \mathbb{R}^+ \cup \{0\}$ with $A_p \neq 0$ and $B_j \in \mathbb{R}^+ \cup \{0\}$ with $B_q \neq 0$. One thing should be kept in mind while fitting a GARCH model to a series that the series should be stationary. So, there are two ways in which GARCH model can be fitted to NIFTY daily returns. 1. First step is to check if the given series is stationary or not using Augmented Dickey Fuller (ADF) test. If the series is stationary, then we can directly fit appropriate GARCH model to the given series. Let us check whether our series is stationary or not using the Augmented Dickey Fuller(ADF) test. For more details about ADF test refer to Appendix B and [10].



```
ADF Statistic:-15.289414568467075
p-value: 4.464384772471764e-28
Critical Values:
    1%: -3.432
    5%: -2.862
    10%: -2.567
Reject Hypothesis - Time Series is Stationary
```

Figure 3.7: Output of ADF test for NIFTY daily returns

Now that we have verified that our series is stationary, it is time to check the autocorrelation graphs for the appropriate GARCH fitted model. For more about how appropriate GARCH model is found please refer to section 4 of Chapter 4.

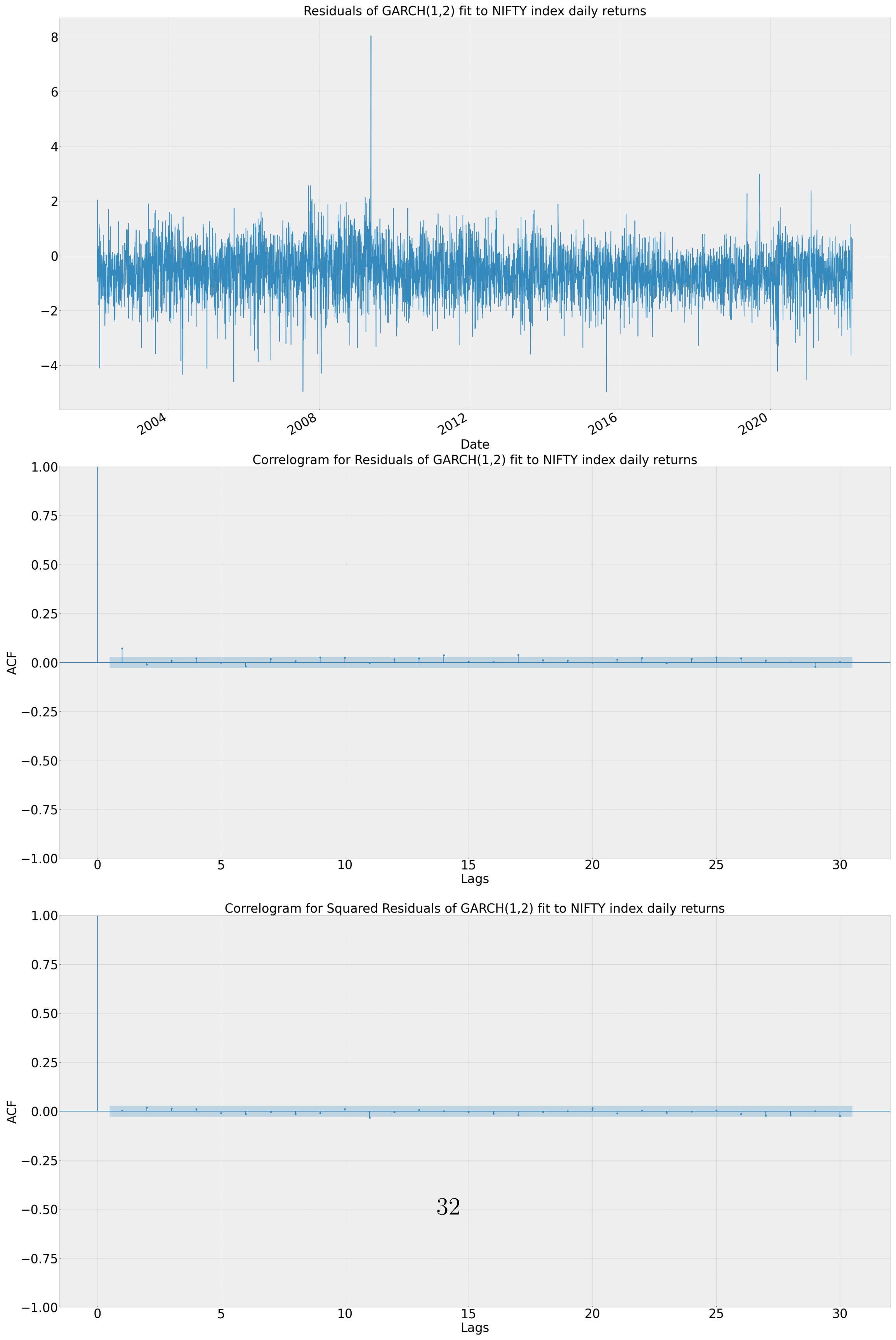


Figure 3.8: Autocorrelation graphs after fitting GARCH(1,2) model to NIFTY daily returns

And finally, we are able to see insignificant autocorrelation in the squared residuals which shows that GARCH model has taken care of volatility clustering.

2. Suppose, In the first method the series turned out to be non-stationary, then what will we do?? This is where another variant of ARMA model known as Autoregressive Integrated Moving Average (ARIMA) model comes for our rescue. ARIMA model converts non-stationary series into stationary series by differencing. And then the residuals we get from ARIMA model are stationary and thus are fitted with appropriate GARCH model.

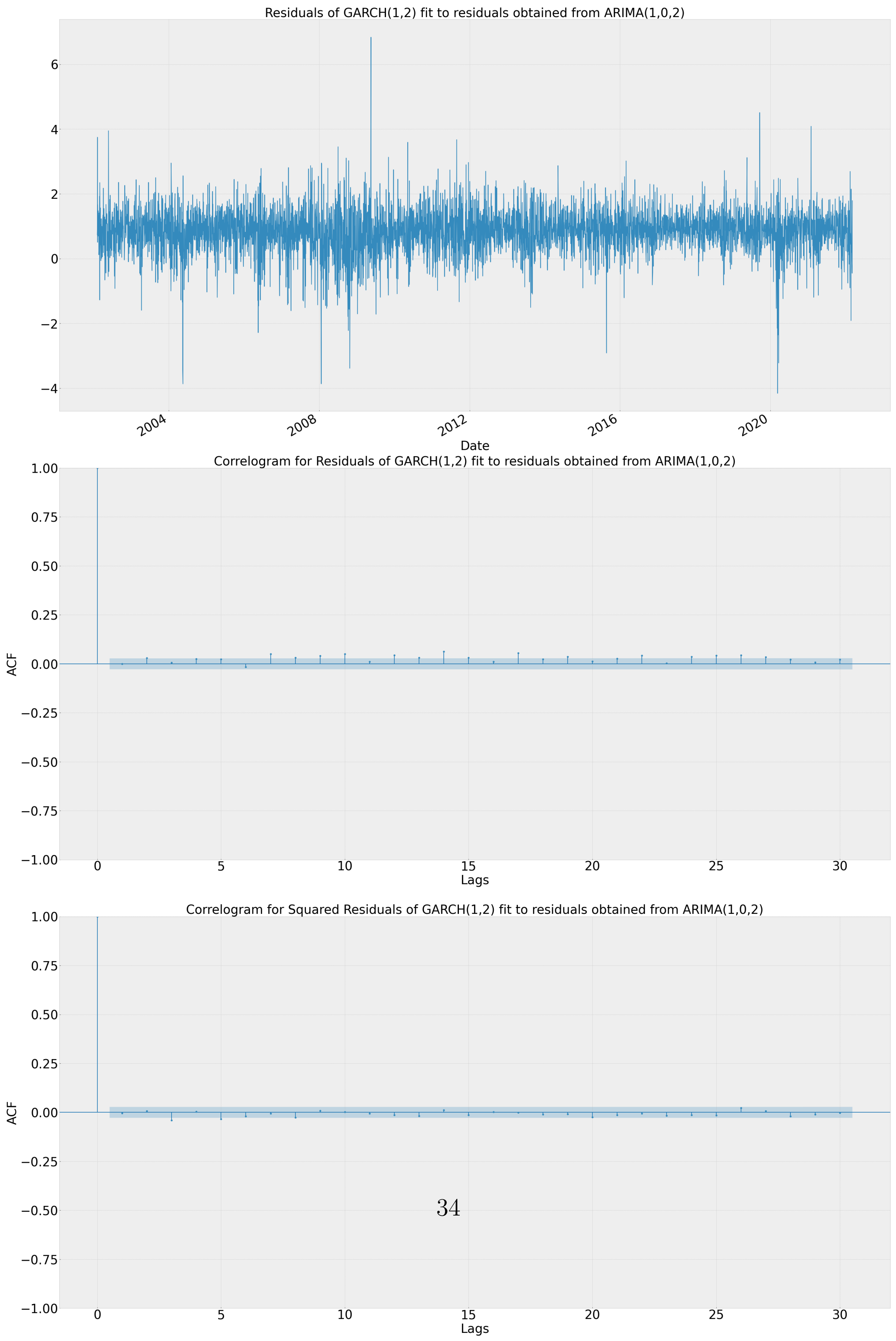


Figure 3.9: Autocorrelation graphs for ARIMA(1,0,2)+GARCH(1,2) model to NIFTY daily returns

Chapter 4

Comparison

Now, that we have understood HMS and GARCH models let us try to compare them. For comparison we will need a true volatility indicator. In this thesis, we are taking NSE India VIX Index as the true volatility indicator. India VIX Index is an implied volatility indicator based on NIFTY option prices. Historical data for India VIX Index from March 2009 to March 2023 has been downloaded from NSE India website. Now we have true volatility indicator in our hand, let us see how to compare the outputs of HMS model and GARCH model with India VIX Index. First let's start with HMS model.

4.1 HMS model v/s India VIX Index

In the last chapter, we learned about the motivation behind HMS models, mathematics behind it and their implementation in Python. In the implementation part, we got a fitted model and now our aim is to make some predictions based on that fitted model. So in this section, our focus shifts to forecasting probabilities using the fitted HMS model. As we know from the previous chapter, the outputs of HMS models are probabilities of being in different regimes. So, let's begin by understanding the intuition behind forecasting probabilities and then we will move to the math behind it.

4.1.1 Motivation behind volatility forecasting using HMS model

For this, let us revisit the example of the Markov chain in section 3.1.2 of Chapter 3. To find out the next state of the stochastic variable, we need two things: Present or Current state of variable and Transition Probabilities. If you look closely at the summary we got 3.3, it clearly gives us the transition probabilities. All we have to worry about is how to find the current volatility state. This is where smoothed and predicted probabilities come into the picture. Smoothed marginal probabilities gives the probability of volatility being in a state at time t given all the observations upto time T. Predicted marginal probabilities gives the probability of volatility being in a state at time t given all the observations upto time (t-1). For forecasting, smoothed marginal probabilities are of no use because it is taking account of future data (at time t, it is using data for time ¿ t as well) to calculate state probabilities. Whereas predicted marginal probabilities are exactly what we needed to solve the problem of figuring out the current volatility state.

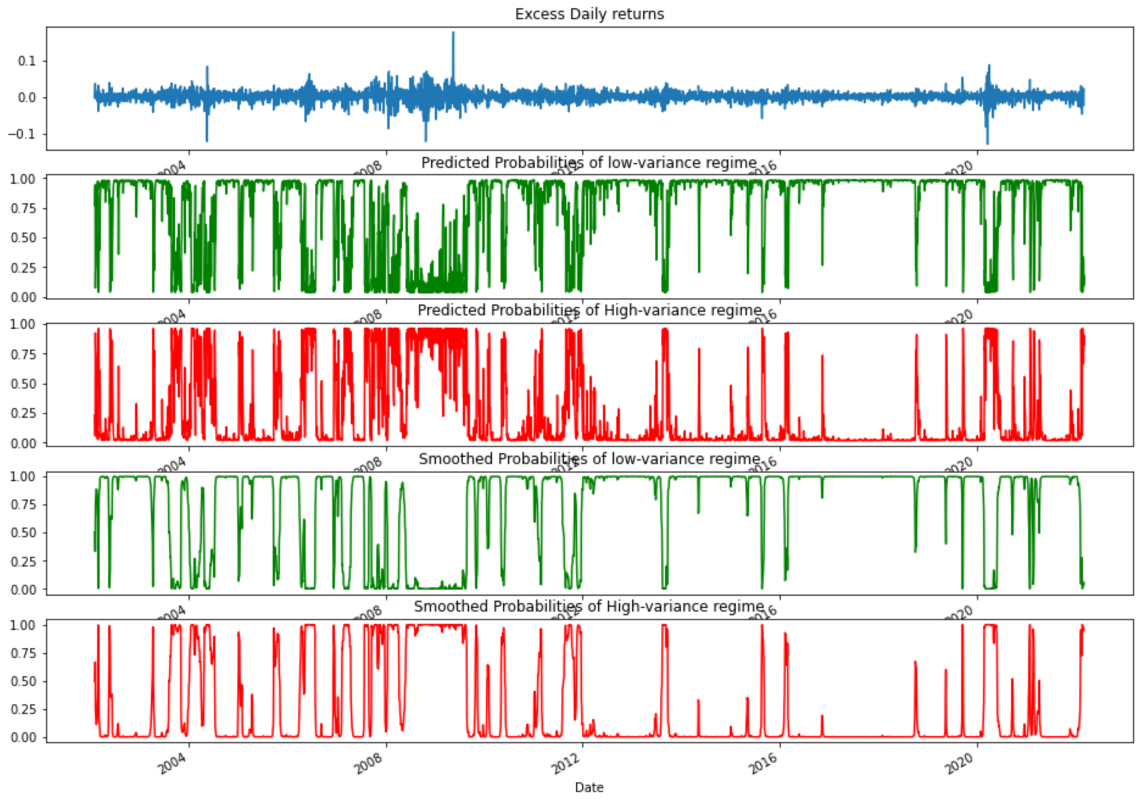


Figure 4.1: Predicted and Smoothed marginal probabilities for NIFTY daily returns from fitted HMS model

4.1.2 Math behind volatility forecasting using HMS model

Now, let's understand the math behind volatility forecasting.
Let X_t be the stochastic variable for volatility at time t, θ_{t-1} denotes all the observations(returns) upto time $t-1$, σ_n is the volatility in state n, r_t is return at time t and p_{mn} are transition probabilities.
We have to calculate $\alpha_{l,t+1} = Pr(X_{t+1} = l|\theta_t)$ for which we start by calculating the filtered probability $\kappa_{nt} = Pr(X_t = n|\theta_t)$ = the probability of volatility (Hidden variable) being in state n at time t given all the returns (observed variable) upto time t are known.

The formula for κ_{nt} and $\alpha_{l,t+1}$ are given below:

$$\kappa_{nt} = \frac{\sum_{m=0}^{1} p_{mn}\kappa_{m,t-1}\zeta_{nt}}{f(r_t|\theta_{t-1})} \text{ for } n = 0, 1 \tag{4.1}$$

$$\alpha_{l,t+1} = \sum_{n=0}^{1} p_{nl}\kappa_{nt} \text{ for } l = 0, 1 \tag{4.2}$$

where

$$\zeta_{nt} = f(r_t|X_t = n, \theta_{t-1}) = \frac{1}{\sqrt{2\pi}\sigma_n} e^{(-\frac{r_t^2}{2\sigma_n^2})} \tag{4.3}$$

$$f(r_t|\theta_{t-1}) = \sum_{m=0}^{1}\sum_{n=0}^{1} p_{mn}\kappa_{m,t-1}\zeta_{nt} \tag{4.4}$$

p_{mn} are transition probabilities which are known from the output of fitted model, ζ_{nt} denotes the density when volatility is in state n that is calculated from 3.4, $f(r_t|\theta_{t-1})$ denotes the conditional density at the tth observation, σ_n is the volatility in state n which is known from the output of the fitted model and $\kappa_{m,t-1}$ is known from filtered marginal probabilities as discussed above. This section is inspired from a blog [2] which I wrote for AlgoAnalytics.

4.1.3 Graphical and Analytical results

Here, we will using a training window if size=900 data points of NIFTY daily return and testing window of size=40 data points. What the above statement means is that the first 900 data values will be used to calculate parameters of the fitted HMS model that will used to forecast probabilities for the next 40 values. This is how a rolling window is used to generate the output. Python code for HMS model and VIX index comparison is given in Appendix C.

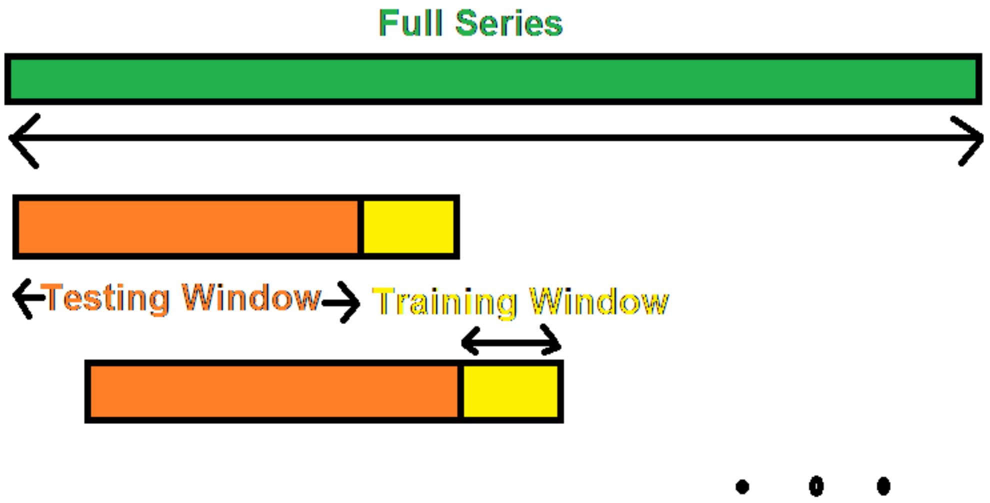


Figure 4.2: How testing, training and rolling is window is used

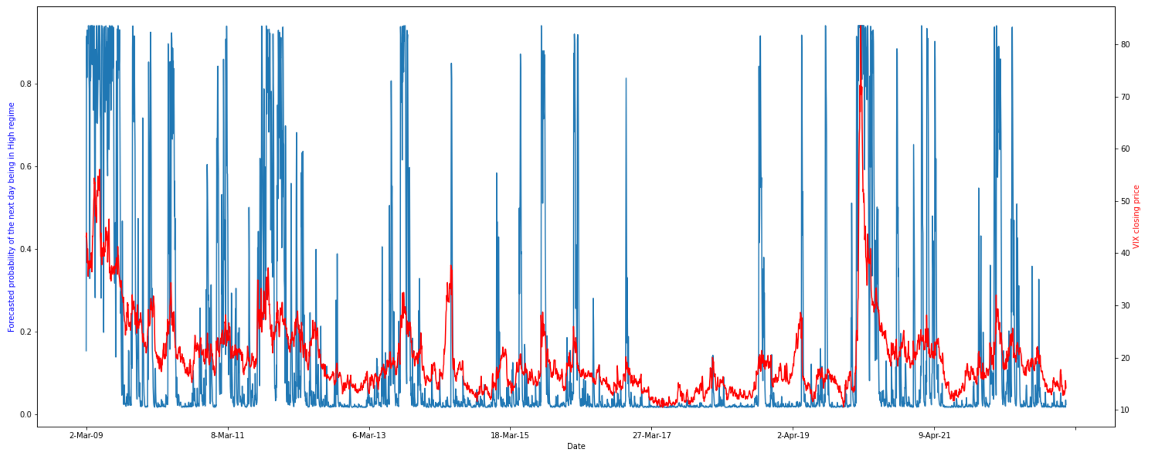


Figure 4.3: Forecasted probabilities of next day being in High regime with VIX closing price

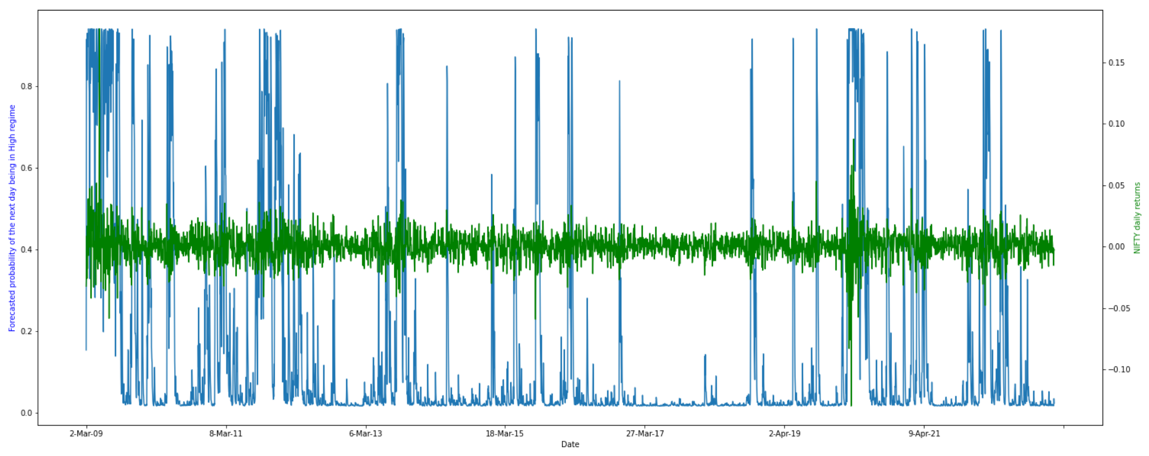
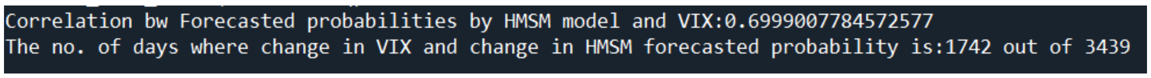


Figure 4.4: Forecasted probabilities of next day being in High regime with daily NIFTY return

Here we see that forecasted probabilities of next day being in High regime looks very similar to closing VIX Index price graphically. Also, forecasted probabilities of next day being in High regime is compared with today's VIX index closing price. Let's check the correlation between them.



```
Correlation bw Forecasted probabilities by HMSM model and VIX:0.6999007784572577
The no. of days where change in VIX and change in HMSM forecasted probability is:1742 out of 3439
```

Figure 4.5: Correlation between Forecasted probabilities of HMS model with VIX closing price

As expected from graphs, the correlation between forecasted probabilities by HMS model and VIX Index closing price is pretty high which is an indicator that our fitted HMS model is able to forecast volatility correctly. Most of the times in finance world, we are not interested in actual volatility value but rather it's change (whether its going to increase or decrease tomorrow). So for this, we have tried to calculate the number of days in which one day change in VIX Index matches with one day change in forecasted HMS probability.

4.2 GARCH v/s VIX

In the last chapter we learned about GARCH model and how it takes of volatility clustering into account. We can get a forecasted variance as an output of the GARCH process which we will use to compare with VIX Index closing price. Here, in this case forecasted variance is an indicator of implied volatility. 1 step ahead forecasted variance will be compared with VIX Index closing price. Before the comparison, we will have come up with a method of finding the order (p and q) of GARCH model.

4.2.1 Finding order p and q

There are many ways in which one can find the order of GARCH model like using ACF/PACF plots, information criteria and GARCH(1,1) [14] etc. But here we will finding the order based on information criteria. We start by finding the best fitted ARIMA model for our series based on Akaike Information Criterion (AIC). AIC is a measure that deals with the amount of information lost by the model. So, from the last line it is pretty mych clear that we need AIC to be lower to get a better fitted model. It takes number of parameters used and how good the fitted model is into account.
The formula for calculating AIC is : $AIC = 2n - 2ln(L_{max})$
where n is the number of parameters estimated and L_{max} is the maximum value of the likelihood function.
Using the auto_arima from pmdarima class in python, we find the best fitted ARIMA model which will give us the order p and q. Then, we can use these p and q to fit GARCH(p, q) on our data using arch_model which finds it's parameters by maximizing the log-likelihood function.

4.2.2 Graphical and Statistical results

Here, we will using a training window if size=900 data points of NIFTY daily return and testing window of size=40 data points. What the above statement means is that the first 900 data values will be used to calculate parameters of the fitted GARCH model that will used to forecast variance for the next 40 values. Size of training window is chosen such that GARCH gives good results. Then a rolling window like in 4.2 is used to generate all the outputs. Now, using forecast in python, we are able to generate 1 step ahead forecast for all the values in testing window. Python code for GARCH model and VIX index comparison is given in Appendix D.

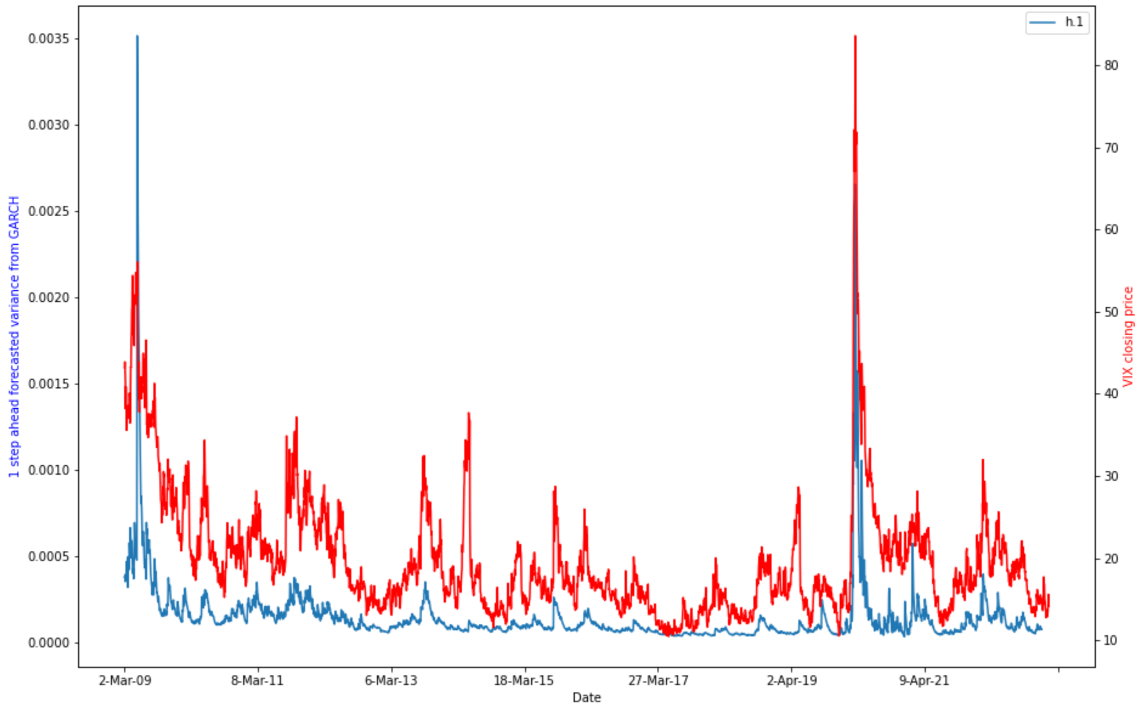


Figure 4.6: 1 step ahead forecasted variance by GARCH with VIX closing price

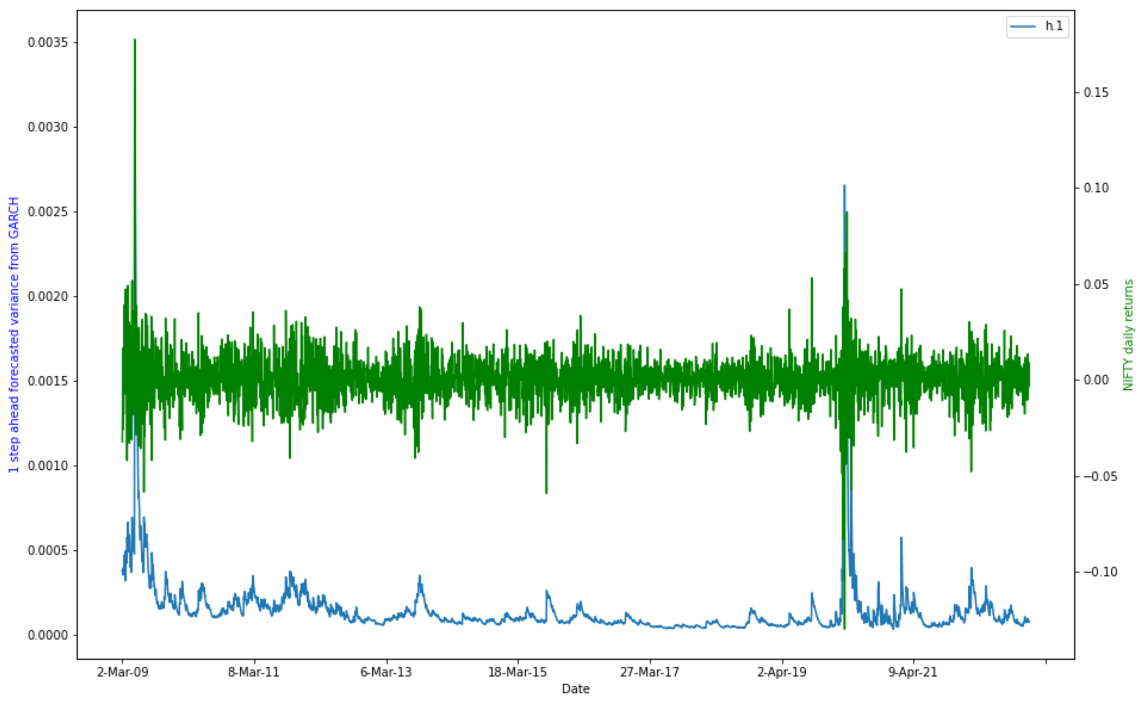
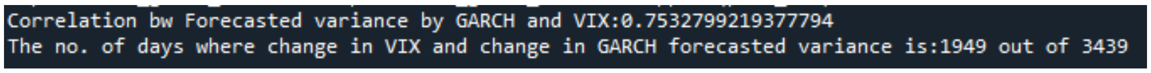


Figure 4.7: 1 step ahead forecasted variance by GARCH with daily nifty return

Here we see that 1 step ahead forecasted variance looks pretty much similar to closing VIX Index price graphically. They are moving up and down around the same and their lows and highs are also occurring at the same time. Alos, it should be noted that 1 step ahead forecasted volatility is compared with today's VIX index closing price. Although, results look convincing graphically but we can't rely on them fully. So, Let's start by calculating some statistical parameters. We first check the correlation between them.



```
Correlation bw Forecasted variance by GARCH and VIX:0.7532799219377794
The no. of days where change in VIX and change in GARCH forecasted variance is:1949 out of 3439
```

Figure 4.8: Correlation between 1 step ahead forecasted variance by GARCH with VIX closing price

As expected from graphs, the correlation between forecasted variance by GARCH model and VIX Index closing price is pretty high which is an indicator that our fitted GARCH model is able to forecast volatility correctly. Most of the times in finance world, we are not interested in actual volatility value but rather it's change (whether its going to increase or decrease tomorrow). So for this, we have tried to calculate the number of days in which one day change in VIX Index matches with one day change in forecasted GARCH variance.

4.3 HMS v/s GARCH

In the previous sections, we have compared HMS and GARCH models with the India VIX index. In this section, we will compare HMS and GARCH models based on their correlation with VIX index and accuracy. Accuracy is calculated by dividing the number of days in which the change in predicted volatility matches with the change in VIX index by total number of days.

$$\text{Accuracy} = \frac{\sum_t \mathbb{1}_{sgn(\Delta y_t)=sgn(\Delta v_t)}}{\text{Total number of days}}$$

where y_t is the forecasted volatility, v_t is VIX index closing price and sgn is the signum function.

Table 4.1: HMS v/s GARCH based on correlation and accuracy for testing window=40

Training Window	Testing Window	GARCH-VIX Correlation	HMS-VIX Correlation	GARCH-VIX Accuracy	HMS-VIX Accuracy
900	40	0.75328	0.69990	0.5667	0.5065
800	40	0.75319	0.70436	0.5708	0.4977
700	40	0.75339	0.68917	0.5673	0.4992
600	40	0.71031	0.69224	0.5681	0.4732
500	40	0.72203	0.66193	0.5655	0.4821

Table 4.2: HMS v/s GARCH based on correlation and accuracy for testing window=20

Training Window	Testing Window	GARCH-VIX Correlation	HMS-VIX Correlation	GARCH-VIX Accuracy	HMS-VIX Accuracy
900	20	0.74918	0.67206	0.5666	0.5057
800	20	0.74977	0.67719	0.5710	0.4971
700	20	0.73419	0.66678	0.5678	0.4981
600	20	0.71081	0.66983	0.5675	0.4719
500	20	0.72850	0.64254	0.5640	0.4792

From the above 2 tables, We can see that there is no clear visible pattern in correlation, accuracy and sizes of training & testing window. But the most important observation from the above tables is that GARCH is over performing HMS model in all the cases. So, according to our hypothesis and comparison basis we can conclude that GARCH is a better model for forecasting volatility than HMS model.

Chapter 5

Effect of Volatility on a trading strategy

In this chapter, we will see the impact of volatility on a trading strategy. In the first section, we start by taking a standard pair trading strategy and explaining the details of that strategy.

5.1 Standard Strategy

This section will explain all the details of the standard strategy which will be used to analyze the effect of volatility.

5.1.1 Mathematical concepts in Pairs Trading

Pairs trading is a type of Mean-Reversion strategy that uses mathematical concepts like correlation, stationarity and cointegration. We will understand the strategy and all it's mathematical concepts one by one. Let us start by looking at the picture below:

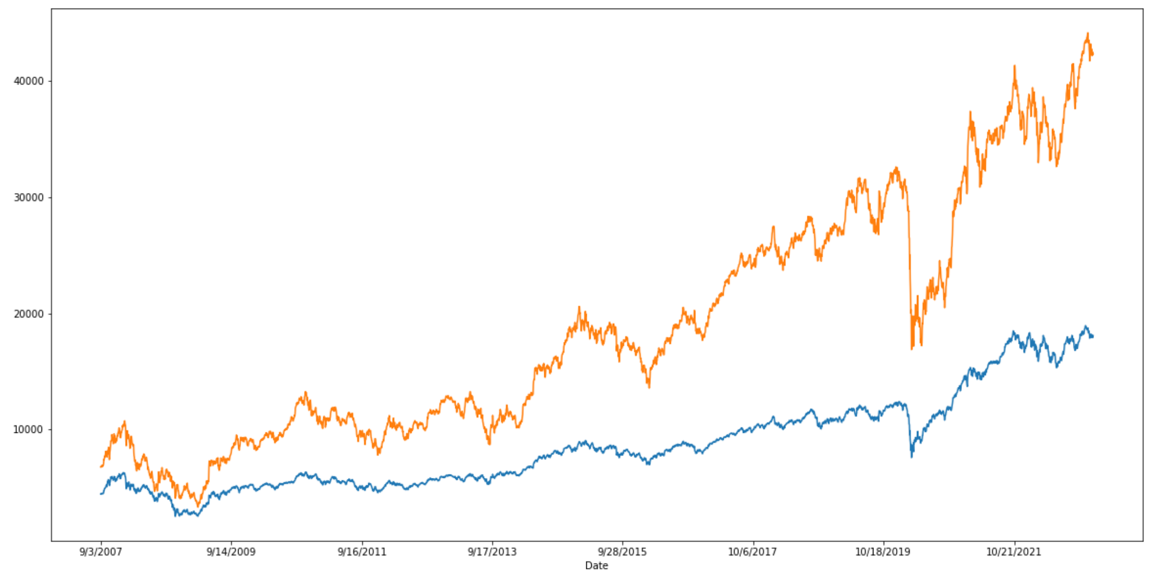
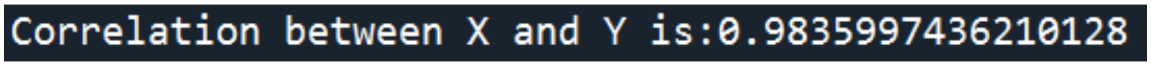


Figure 5.1: Series X and Series Y

What can we observe from it? Price of second stock is very much related to price of first stock. They both are somehow moving in the same direction at almost all times. We can capture the above observations mathematically by saying that the prices of both the stocks are highly correlated. Let's check this by calculating their correlation. Formula for calculating the correlation is given below:

$$\rho = \frac{Cov(X, Y)}{\sigma_X \sigma_Y}$$

where $Cov(X, Y)$ is the Covariance between X and Y, σ_i is the standard deviation of i $\forall i \in \{X, Y\}$ and ρ is correlation coefficient.



```
Correlation between X and Y is:0.9835997436210128
```

Figure 5.2: Correlation between X and Y

It turns out that our observation was correct. High Correlation between two assets/stocks is an indicator that these two assets can be used as a pair for trading.

Now, let's calculate the spread of these 2 stocks and plot it. Formula for calculating spread is given below:

$$\eta_t = \Delta N_t - \Delta B_t$$

where $\Delta N_t = N_t - N_{t-1}$, $\Delta B_t = B_t - B_{t-1}$, N_t is the price of first stock at time t, B_t is the price of second stock at time t and η_t is spread at time t.

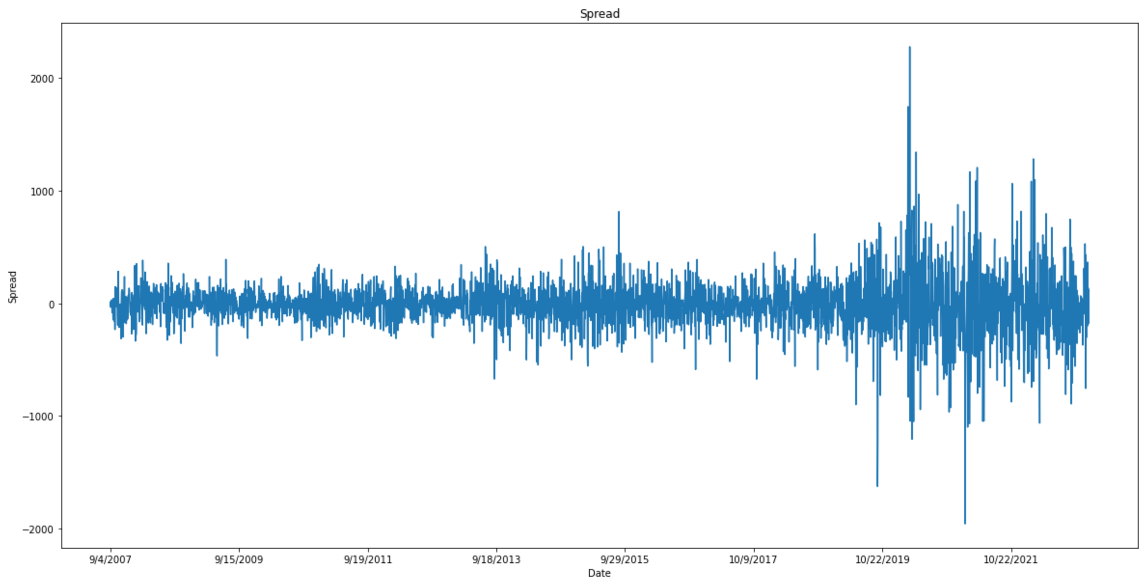


Figure 5.3: Spread series

What can be said after the looking at the spread plot? It is easy to observe from the plot that the spread is always revolving around it's mean or reverting back to it's mean. It is not diverging and is always staying in some kind of bounds. The above observations are described mathematically using the concept of stationarity. Before defining stationarity mathematically, let's try to understand what stationary means. In most simple terms, a series having a constant mean, constant variance and no autocorrelation within itself is called a stationary series. In Statistics, the process satisfying the above properties is called Wide Sense Stationary (WSS) process. Mathematically, we say that a process is stationary when it's unconditional joint pdf remains same during time shift. To test whether a series is stationary or not, Augmented Dickey-Fuller (ADF) test is used. For more details on ADF test check [10]. Now, let's try to test whether our spread series is stationary or not using the ADF test.

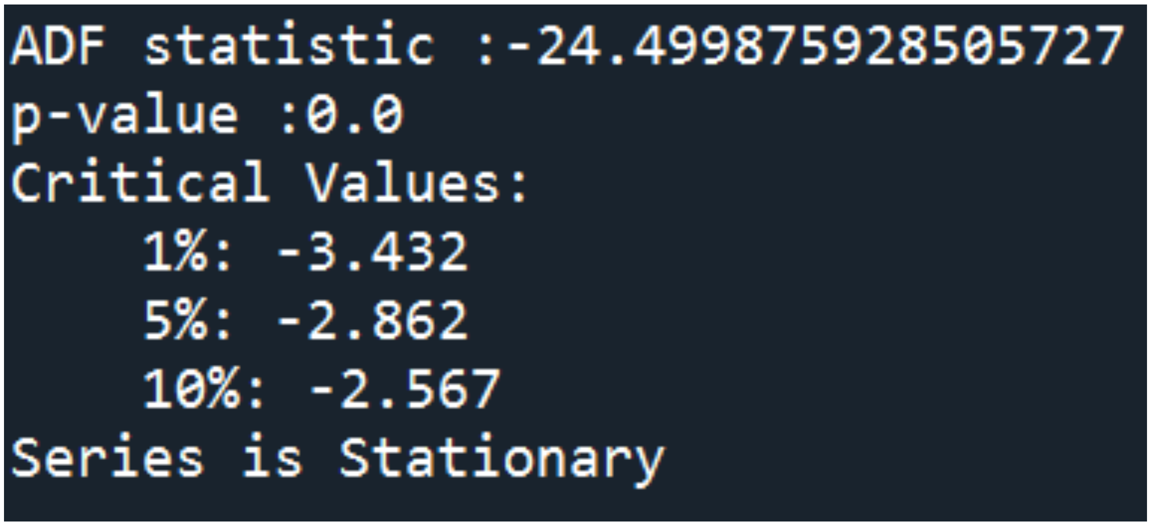

```
ADF statistic :-24.499875928505727
p-value :0.0
Critical Values:
    1%: -3.432
    5%: -2.862
    10%: -2.567
Series is Stationary
```

Figure 5.4: Output of ADF test

The results show that the spread series is stationary.

5.1.2 Intuition behind Pairs trading

Now that we have a basic understanding of mathematical concepts behind pairs trading, let us see how these concepts are used in the strategy.In the previous section, we said that high correlation is needed for a pair to be used in the strategy. But why it so?
Consider 2 companies A and B coming from the same sector (for example Banking sector). Then it will be valid to assume that the external parameters that can impact the price of A and B are almost same; which means that for most of the time these 2 assets move in the same direction. Now, suppose one of the assets deviate from its usual path (maybe because of some internal factor affecting the stock price of a company), then a trading opportunity arises. So, in that case we go long on one and short on the other. Now suppose we had a pair having very less correlation and one

of them deviates from it's usual path, but in this case we can't go short on one and long on other because changes in one won't affect the changes in the other. That is why, high correlation is necessary for a pair to be used in pair trading.
Again we go back to previous section where we said that spread series is coming out to be stationary. We understood that a stationary series around it's mean will always revert back to its mean. This is the most important statistical property which is exploited in pairs trading. We somehow find a series using a pair which is stationary and then use it to generate trading signals. In our case, spread is turning out to be stationary and we will be using it to generate entry-exit signals. The motivation is that whenever spread deviates a lot from its mean, it gives us an opportunity to trade because we know that it (spread) will eventually revert back to its mean. These trading signals are generated by calculating z-score which will be explained in the next section. Suppose we had a pair with high correlation but we were using a non stationary series to generated signals, then there are lot of issues with it. First, the series may not revert back to its mean and can go on deviating from its mean, which is huge problem. Second, the mean or the variance may be varying with time. In that case, it can happen that we have used a mean and a variance to enter the trade but since mean or variance can vary with time, it can create issues during the time of exiting the trade. Hence, a stationary series is essential for a pairs trading. In the next section, we understand the implementation of this strategy.

5.1.3 Implementation of Pairs Trading

Here, we will using historical daily data of 2 most liquid indices in Indian Market: NIFTY50 and BANKNIFTY(BNF) futures. Historical data is downloaded from NSE India website. Near (Current) month contract data is used for our analysis from 2007 to 2023.

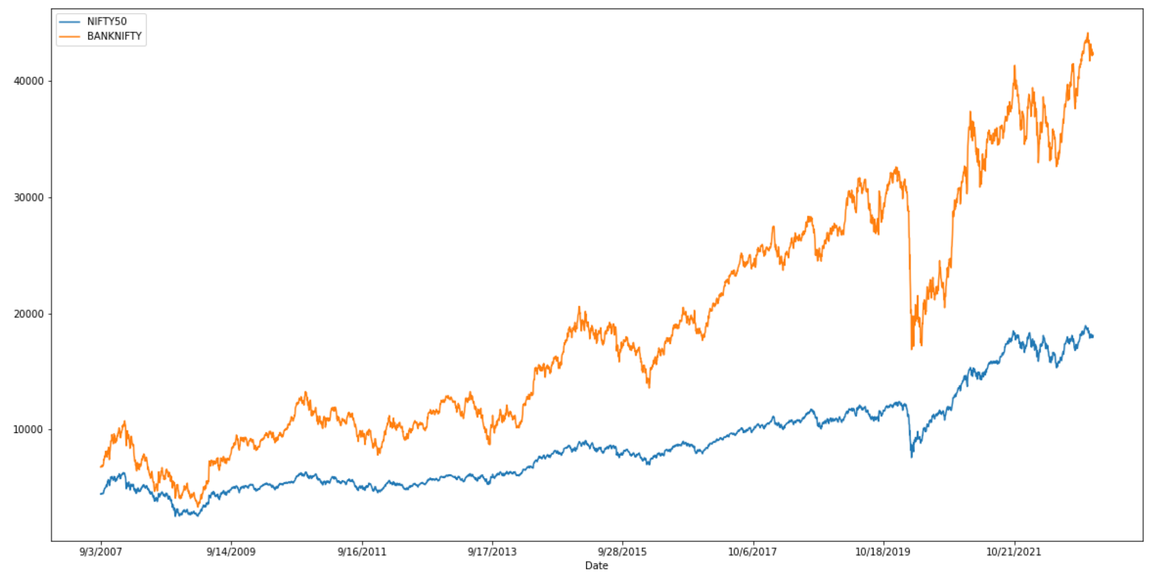
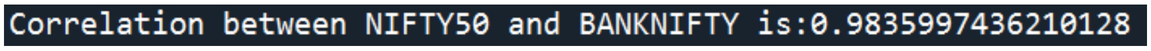


Figure 5.5: NIFTY50 futures and BNF futures

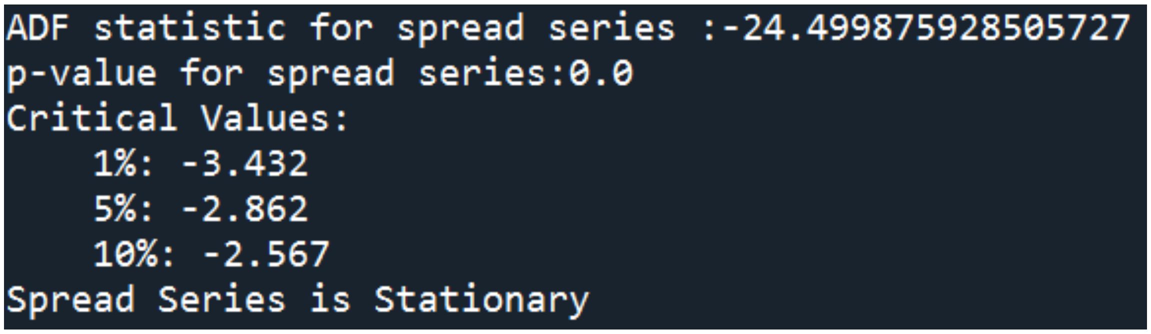
Let's start by calculating the correlation between NIFTY50 and BNF futures.



```
Correlation between NIFTY50 and BANKNIFTY is:0.9835997436210128
```

Figure 5.6: Correlation bewteen NIFTY50 futures and BNF futures

As we can see that the correlation between this pair is really high, so we can use it as a pair. Next thing is stationarity checking. Here we check the statitonarity of spread series. The output of the ADF test for spread is shown below:



```
ADF statistic for spread series :-24.499875928505727
p-value for spread series:0.0
Critical Values:
    1%: -3.432
    5%: -2.862
    10%: -2.567
Spread Series is Stationary
```

Figure 5.7: ADF test output for spread series

It is clear from the output that spread series is stationary. Now, we come to the part of how trading signals are generated. Here, we will using a training window if size=1000 data points and testing window of size=400 data points. What the above statement means is that the first 1000 data values will be used to calculate mean and variance which will be used to generate signals for the next 400 values. Rolling window similar to what is shown in the figure 4.2 is used. Python code for this pairs trading strategy is given in Appendix E.
We should always check the stationarity of spread series for every training window before using it to generate signals. We enter in a trade when the absolute value of z-score for spread series calculated using the mean and variance from the training window goes beyond 2 at any time. When z-score goes beyond 2, it is expected to revert back to its mean and is likely to fall. So, we go short on spread or we sell the spread. By selling the spread it means, we sell NIFTY50 and buy BNF. When z-score becomes less than -2, it is expected to rise in the near future. So, we go long on spread or we buy the spread. By buying the spread it means, we buy NIFTY50 and sell BNF.

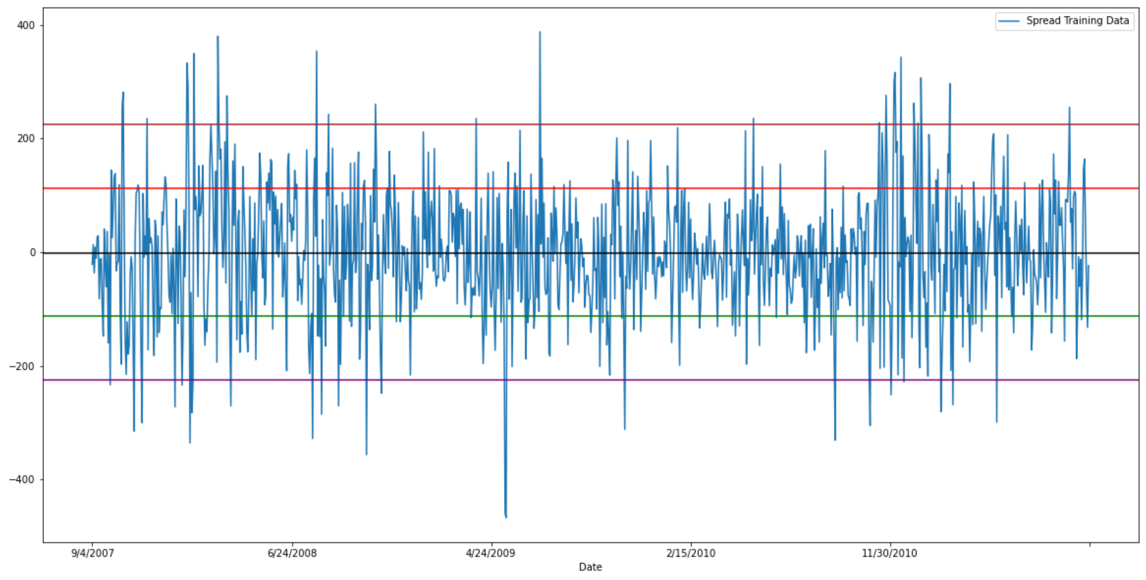


Figure 5.8: Spread training data with mean ans standard deviation intervals

This is how we go in a trade using a pairs trading strategy. Below you see the plot of generated trading signals for spread series for training window of size 1000 and testing window of size 400.

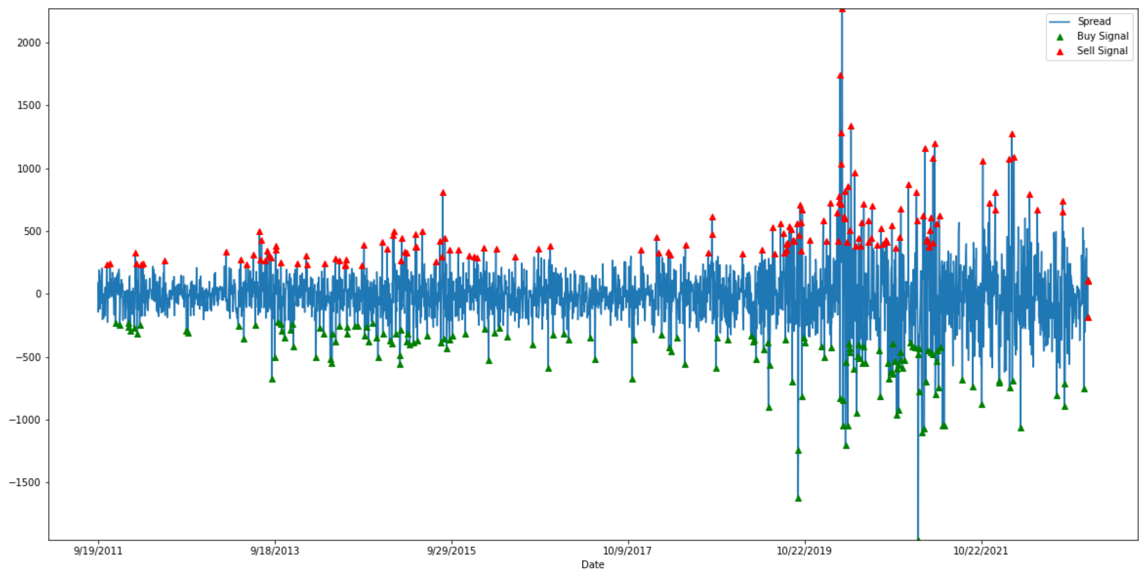


Figure 5.9: Trading Signals generated using z-score

In the next section, we will see how our strategy is performing by calculating different metrics.

5.1.4 Performance of Pairs trading

In the previous section, we learned how to generate signals using z-score. It was mentioned that a signal will be generated when absolute value of z-score goes beyond 2. But nothing was mentioned about the exit signal. We square-off a trade when absolute value of z-score becomes less than 0.75. In sell trade we square-off when z-score falls below 0.75 and during buy trade, we exit the trade when z-score goes beyond -0.75. Here are the performance metrics for the strategy:

Table 5.1: Performance metrics for the Pair Trading strategy

Time Period	2007 to 2022
Stationary series used	Spread
Entry and Exit criteria	$\|z\text{-score}\| > 2$ & $\|z\text{-score}\| < 0.75$
Training Window	1000
Testing Window	400
Accuracy	0.5506607929515418
Total Trades	227
Gain Trades	125
Loss Trades	102
Overall profit	331680.30
Total money gained in gain trades	551818.04
Average gain on a trade	4414.54
Total money lost in loss trades	220137.75
Average loss on a trade	2158.21
Gain/Loss Ratio	2.045462583713991
Average Holding Time	1.5242
Max Holding Time	6
PnL	1.3646
Drawdown	2.78%
Annualized Return	3.7753%
Sharpe Ratio	1.209
Calmer Ratio	1.017

For calculating accuracy, we used one lot each of NIFTY50 (1 lot = 50) and BNF (1 lot = 25) futures and transactions costs including Brokerage, STT/CTT, Transaction charges and bid-ask spread are then subtracted from the money made in the trade. PnL is calculated by starting with an initial capital of 1 and updating initial capital after every trade using the formula:

$$\text{Initial Capital} = \text{Initial Capital}\left(1 + \frac{\text{Money made in trade}}{\text{Money invested in trade}}\right)$$

where Money invested in trade = Notional value of NIFTY50 + Notional value of BNF.

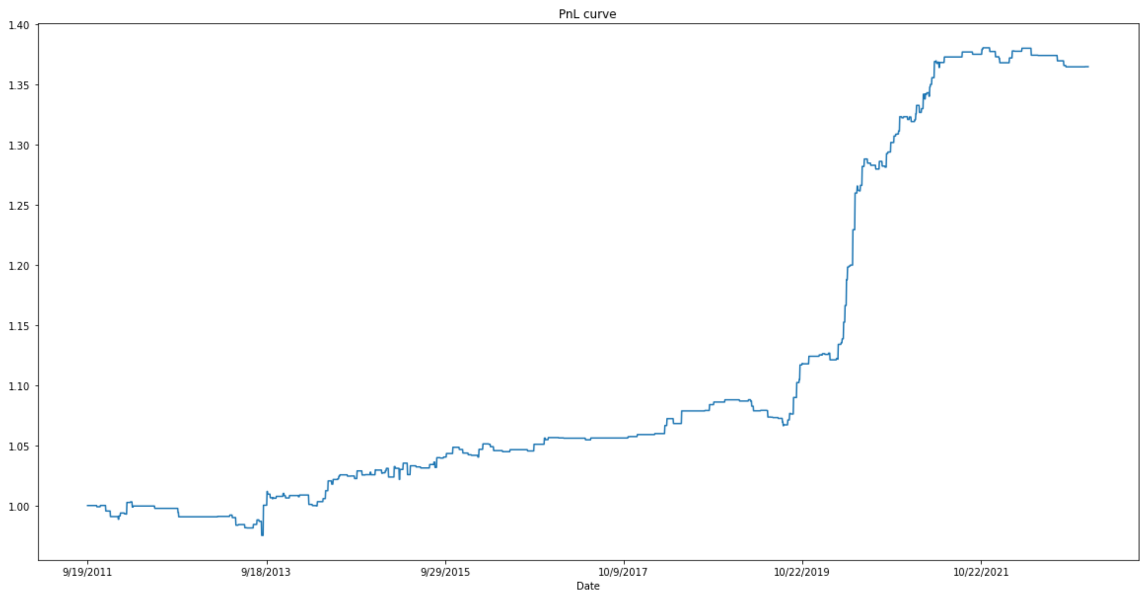


Figure 5.10: PnL curve

It is important to mention that money invested in a futures trade is actually not notional value but some percentage of notional value which are called margins. But margins keep on changing with time, that's why we are using notional values only. This is also the reason why PnL has not increased much in a time span of 12 years. Using the PnL curve metrics like Drawdown, Sharpe ratio and Calmer ratio are calculated.
Now, using the PnL curve we try to see how the pairs trading strategy performs under different regimes. First, let's visualize it using the graph below:

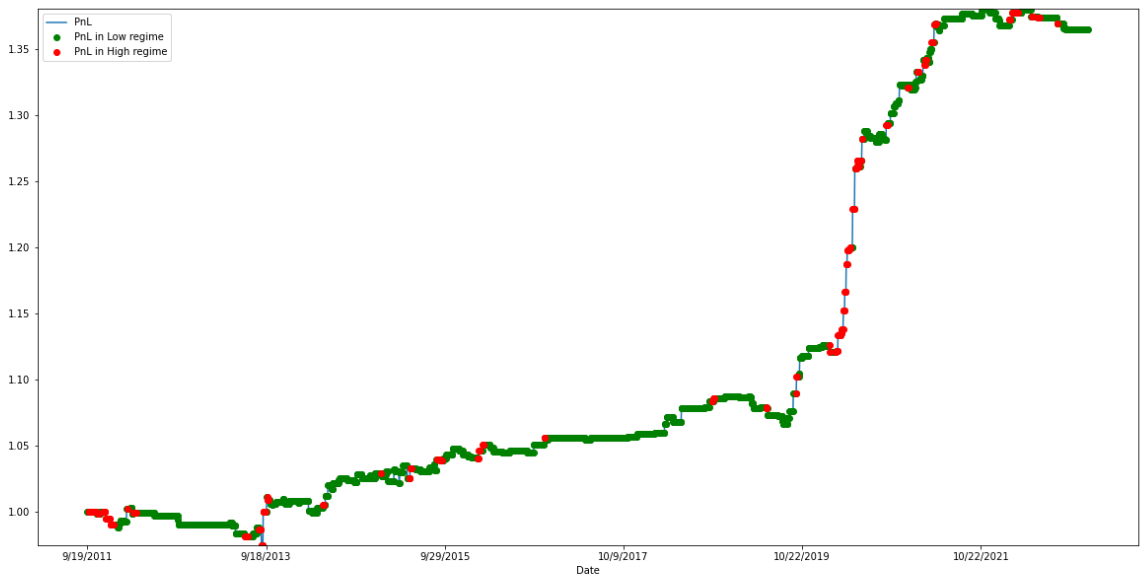


Figure 5.11: PnL curve under different regimes

From this graph, it is pretty much clear that the strategy is performing really well in the High volatility regime. But isn't it surprising that a strategy is making money in highly risky times like COVID? This question remains a topic for further research and will be studied later. Now, let's try to calculate some performance metrics.

Table 5.2: Pair trading strategy performance metrics in High regime

Total days in High regime	232
Initial Capital	1
PnL	1.3695
Annualized Return	40.33%
Sharpe Ratio	4.128
Calmer Ratio	12.688
Drawdown	2.73%

Table 5.3: Pair trading strategy performance metrics in Low regime

Total days in Low regime	2568
Initial Capital	1
PnL	1.3646
Annualized Return	3.072%
Sharpe Ratio	1.0212
Calmer Ratio	1.434
Drawdown	2.16%

It was clearly visible from the PnL graphs that this strategy is performing extremely well in the High volatile region but now from tables 5.2 and 5.3, we can analyze it's performance metrics as well. Having such high returns with huge sharpe ratio is pretty rare but one thing to keep in mind is that we have to wait for High volatile regions to get good returns from this strategy. Although, the performance of our PnL is not so bad in low regime as well like sharpe ratio, calmer ratio and drawdown are good but the annualized returns are pretty low.

Appendix A

Python code for HMS model implementation

```python
import pandas as pd
import statsmodels.api as sm

nifty = pd.read_csv('nifty_cmp.csv',index_col='Date')
closing_price_nifty = nifty['Close'] # Getting NIFTY Index data
daily_ret_nifty = closing_price_nifty.pct_change(1)
daily_ret_nifty = daily_ret_nifty.dropna() # Calculating daily
    returns

# Fit HMS model
model_kns = sm.tsa.MarkovRegression(daily_ret_nifty, k_regimes=2,
    trend='n', switching_variance=True)
result_kns = model_kns.fit()
HMS_summary = result_kns.summary()
print(HMS_summary)
```

Appendix B

Python code for ADF test for stationarity

```python
import pandas as pd
from statsmodels.tsa.stattools import adfuller

nifty = pd.read_csv('nifty_cmp.csv',index_col='Date')
closing_price_nifty = nifty['Close'] # Getting NIFTY Index data
daily_ret_nifty = closing_price_nifty.pct_change(1)
daily_ret_nifty = daily_ret_nifty.dropna() # Calculating daily
    returns

# For checking the stationarity of returns
stationarity_test = adfuller(daily_ret_nifty)
print('ADF Statistic: ' + str(stationarity_test[0]))

print('p-value: '+ str(stationarity_test[1]))

print('Critical Values: ')

for key, value in stationarity_test[4].items():
    print('\t%s: %.3f' % (key,value))
if stationarity_test[0] < stationarity_test[4]["5%"]:
    print("Reject Hypothesis - Time series is Stationary")
else:
    print("Failed to Reject Hypothesis - Time series is Non-
    Stationary")
```

Appendix C

Python code for HMS model and VIX index comparison

```python
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import math as m


nifty = pd.read_csv('nifty_cmp.csv',index_col='Date')
closing_price_nifty = nifty['Close'] # Getting NIFTY Index data
daily_ret_nifty = closing_price_nifty.pct_change(1)
daily_ret_nifty = daily_ret_nifty.dropna() # Calculating daily
    returns
vix = pd.read_csv('VIX_cdata.csv',index_col='Date')
vix_close = vix.iloc[:,3] # Getting VIX closing price data

f = plt.figure(1,figsize=(25,10))
daily_ret_nifty.plot()
f.show()

tn_wndw = 900 # Training window size
tt_wndw = 40 # Training window size
# Defining series for predicted probabilities by HMS model
predicted_hmsm_high_probs = pd.DataFrame(data = nifty['Open'][
    tn_wndw:])
# Setting index for predicted probabilities by HMS model
predicted_hmsm_high_probs = predicted_hmsm_high_probs.set_index
(daily_ret_nifty.index[tn_wndw-1:])
i = 0
while i < len(daily_ret_nifty) - tn_wndw - tt_wndw:
    mod_kns = sm.tsa.MarkovRegression(daily_ret_nifty[i:i+tn_wndw],
    k_regimes=2, trend='n', switching_variance=True)
    res_kns = mod_kns.fit()
    # Filtered marginal probabilities from fitted HMS model
    Low_var_regime_probs = res_kns.filtered_marginal_probabilities
    [0]
     High_var_regime_probs = res_kns.filtered_marginal_probabilities
    [1]
     # Transition Probabilities from fitted HMS model
     p00 = res_kns.regime_transition.T[0, 0, 0]
     p01 = res_kns.regime_transition.T[0, 0, 1]
     p10 = res_kns.regime_transition.T[0, 1, 0]
     p11 = res_kns.regime_transition.T[0, 1, 1]
     var0 = res_kns.params[2] # Variance of Low volatility state
     var1 = res_kns.params[3] # Variance of High volatility state
     ini_low_prob = Low_var_regime_probs.iloc[-1]
     ini_high_prob = High_var_regime_probs.iloc[-1]
     j=0
     # Loop for forecasting probabilities for next state
     while j<tt_wndw:
         # density when volatility is in low state
         nrv_low = (m.e**(-(daily_ret_nifty[j+i+tn_wndw-1])**2/(2*
    var0)))/m.sqrt(2*m.pi*var0)
         # density when volatility is in high state
         nrv_high = (m.e**(-(daily_ret_nifty[j+i+tn_wndw-1])**2/(2*
    var1)))/m.sqrt(2*m.pi*var1)
         next_state_low_prob_temp = (ini_low_prob * p00 +
    ini_high_prob * p10)*nrv_low
         next_state_high_prob_temp = (ini_low_prob * p01 +
    ini_high_prob * p11)*nrv_high
         next_state_low_prob_temp1 = next_state_low_prob_temp/(
    next_state_low_prob_temp+next_state_high_prob_temp)
         next_state_high_prob_temp1 = next_state_high_prob_temp/(
    next_state_low_prob_temp+next_state_high_prob_temp)
         next_state_low_prob =  next_state_low_prob_temp1 * p00 +
    next_state_high_prob_temp1 * p10
         next_state_high_prob = next_state_low_prob_temp1 * p01 +
    next_state_high_prob_temp1 * p11
         predicted_hmsm_high_probs['Open'][j+i] =
    next_state_high_prob
         ini_low_prob = next_state_low_prob
         ini_high_prob = next_state_high_prob
         j = j + 1
     i = i + tt_wndw


# Plotting Graphs
fig,ax1 = plt.subplots(figsize=(25,10))

ax1.yaxis.label.set_color('blue')
predicted_hmsm_high_probs['Open'][:-27].plot(ax = ax1, ylabel = '
```

```python
    Forecasted probability of the next day being in High regime')

ax2 = ax1.twinx()
ax2.yaxis.label.set_color('red')
vix_close.plot(ax = ax2, color = 'red', ylabel = 'VIX closing price'
    )
plt.show()

fig, ax3 = plt.subplots(figsize=(25,10))

ax3.yaxis.label.set_color('blue')
predicted_hmsm_high_probs['Open'][:-27].plot(ax = ax3, ylabel = '
    Forecasted probability of the next day being in High regime')

ax4 = ax3.twinx()
ax4.yaxis.label.set_color('green')
daily_ret_nifty[tn_wndw-1:-27].plot(ax = ax4, color = 'green',
    ylabel = 'NIFTY daily returns')
plt.show()

print('Correlation bw Forecasted probabilities by HMSM model and VIX
    :' + str(predicted_hmsm_high_probs['Open'].corr(vix_close)))
vix_change = vix_close.pct_change(1)
vix_change = vix_change.dropna()
vix_change_ary = vix_change.to_numpy()
pred_prob_change = predicted_hmsm_high_probs['Open'].pct_change(1)
pred_prob_change = pred_prob_change.dropna()
pred_prob_change_ary = pred_prob_change.to_numpy()

iter = 0
cnt = 0
# here 3439 is taken beacuse of available length of NIFTY daily
    return
while iter < 3439:
    if vix_change_ary[iter] < 0 and pred_prob_change_ary[iter] < 0:
        cnt += 1
    elif vix_change_ary[iter] > 0 and pred_prob_change_ary[iter] >
    0:
        cnt += 1
    else:
        cnt = cnt
    iter = iter + 1

print('The no. of days where change in VIX and change in HMSM
    forecasted probability matches is:' + str(cnt) + ' out of ' + str
    (iter))
```

Appendix D

Python code for GARCH model and VIX index comparison

```python
import pandas as pd
import datetime as dt
from arch import arch_model
import matplotlib.pyplot as plt
import pmdarima

#Get nifty prices
nifty = pd.read_csv('nifty_cmp.csv',index_col='Date') # Get NIFTY
   index data
closing_price_nifty = nifty['Close']
daily_ret_nifty = closing_price_nifty.pct_change(1)
daily_ret_nifty = daily_ret_nifty.dropna()
vix = pd.read_csv('VIX_cdata.csv',index_col='Date')
vix_close = vix.iloc[:,3] # Get VIX index closing price

# Defining series for predicted variance by GARCH model
predicted_garch_variance = pd.DataFrame()
tn_wndw = 900 # Training window
tt_wndw = 40 # Testing window
d_r_n = daily_ret_nifty[:]
i = 0
while i<len(d_r_n) - tn_wndw - tt_wndw:
    tn_set = d_r_n[i:i+tn_wndw+tt_wndw-1]
    # Fitting best arima model by minimizing AIC
    arima_model_fitted = pmdarima.auto_arima(tn_set)
    # Finding Lag orders p, q from the fitted ARIMA model
    ar_order = arima_model_fitted.order[0]
    ma_order = arima_model_fitted.order[2]
    # Fitting GARCH(p,q) model
    mdl = arch_model(tn_set, vol='GARCH', p=max(1,ar_order),q=max(1,
   ma_order))
    # Best fitted arima model can have zero value for p or q
    # That's why maximum of 1 and order given by arima is taken in
    GARCH as input
     res = mdl.fit(last_obs=tn_wndw-1)
     model_forecast = res.forecast(horizon=1)
     pred_var = model_forecast.variance[tn_wndw-1:]
     predicted_garch_variance = predicted_garch_variance.append(
    pred_var)
     i=i+tt_wndw

# Plottingg graphs
fig,ax1 = plt.subplots(figsize=(15,10))

ax1.yaxis.label.set_color('blue')
predicted_garch_variance.plot(ax = ax1, ylabel = '1 step ahead
    forecasted variance from GARCH')

ax2 = ax1.twinx()
ax2.yaxis.label.set_color('red')
vix_close.plot(ax = ax2, color = 'red', ylabel = 'VIX closing price'
    )
plt.show()

fig, ax3 = plt.subplots(figsize=(15,10))

ax3.yaxis.label.set_color('blue')
predicted_garch_variance.plot(ax = ax3, ylabel = '1 step ahead
    forecasted variance from GARCH')

ax4 = ax3.twinx()
ax4.yaxis.label.set_color('green')
daily_ret_nifty[tn_wndw-1:-27].plot(ax = ax4, color = 'green',
    ylabel = 'NIFTY daily returns')
plt.show()

print('Correlation bw Forecasted variance by GARCH and VIX:' + str(
    predicted_garch_variance['h.1'].corr(vix_close)))
vix_change = vix_close.pct_change(1)
vix_change = vix_change.dropna()
vix_change_ary = vix_change.to_numpy()
pred_var_change = predicted_garch_variance.pct_change(1)
pred_var_change = pred_var_change.dropna()
pred_var_change_series = pred_var_change.squeeze()
pred_var_change_ary = pred_var_change.to_numpy()

iter = 0
cnt = 0
# here 3439 is taken beacuse of available length of NIFTY daily
```

```
    return
while iter < 3439:
    if vix_change_ary[iter] < 0 and pred_var_change_series[iter] < 0:
        cnt += 1
    elif vix_change_ary[iter] > 0 and pred_var_change_series[iter] > 0:
        cnt += 1
    else:
        cnt = cnt
    iter = iter + 1

print('The no. of days where change in VIX and change in GARCH forecasted variance matches is:' + str(cnt) + ' out of ' + str(iter))
acc = cnt/iter
print(acc)
```

Appendix E

Python code for pairs trading strategy

```python
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
import numpy as np

nifty50 = pd.read_csv('NFT_F1.csv') # Getting NIFTY futures data
nifty50 = nifty50.set_index('Date')
niftybank = pd.read_csv('BNF_F1.csv') # Getting BNF futures data
niftybank = niftybank.set_index('Date')
X = nifty50['Close']
daily_ret_nifty = X.pct_change(1) # Daily NIFTY returns will be
    needed for regime classification
Y = niftybank['Close']


diff_nifty50 = X.diff()
diff_nifty50 = diff_nifty50.dropna()
diff_niftybank = Y.diff()
diff_niftybank = diff_niftybank.dropna()
spread = diff_nifty50 - diff_niftybank # Spread Calculation
f = plt.figure(1, figsize=(20, 10))
spread.plot(ylabel='Spread')
plt.title("Spread")
stationarity_test_spread = adfuller(spread)

# So from now on, we will be using spread as a stock or an asset
spread_mean = spread.mean()
spread_std = spread.std()

# Plotting Spread intervals
f1 = plt.figure(2,figsize=(20, 10))
spread.plot()
```

```python
plt.axhline(spread_mean, color='black')
# Spread mean is very close to zero
plt.axhline(spread_std, color='red')
plt.axhline(-spread_std, color='green')
plt.axhline(2*spread_std, color='brown')
plt.axhline(-2*spread_std, color='purple')
plt.legend(['Spread'])
f1.show()

# Trading Strategy
split = 1000 # Training window
spread_train = spread[:split]
spread_test = spread[split:]
stationarity_test_spread_train = adfuller(spread_train)
delX_train = diff_nifty50[:split]
delX_test = diff_nifty50[split:]
delY_train = diff_niftybank[:split]
delY_test = diff_niftybank[split:]

f2 = plt.figure(3,figsize=(20, 10))
spread_train.plot()
plt.axhline(spread_train.mean(), color='black')
plt.axhline(spread_train.std(), color='red')
plt.axhline(-spread_train.std(), color='green')
plt.axhline(2*spread_train.std(), color='brown')
plt.axhline(-2*spread_train.std(), color='purple')
plt.legend(['Spread Training Data'])
f2.show()

spread_train_mean = spread_train.mean()
spread_train_std = spread_train.std()

def zsc(A,mu,sigma):
    zscr = (A - mu)/sigma
    return zscr

buy = spread_test.copy()
sell = spread_test.copy()
wndw = 400 # Testing window
i=0
while i<len(spread) - split - wndw:
    spread_train_w = spread[i:i+split]
    spread_train_w_mu = spread_train_w.mean()
    spread_train_w_std = spread_train_w.std()
    spread_test_w = spread[i+split:i+wndw+split]
    iter = 0
    while iter < wndw:
        if zsc(spread_test_w[iter],spread_train_w_mu,spread_train_w_std) < -2:
            sell[iter+i] = 3000
        elif zsc(spread_test_w[iter],spread_train_w_mu,spread_train_w_std) > 2:
            buy[iter+i] = -3000
        else:
            buy[iter+i] = -3000
            sell[iter+i] = 3000
        iter = iter + 1
    i=i+wndw

# Plotting generation of trading signals
f3 = plt.figure(4,figsize=(20,10))
spread_test.plot()
buy.plot(color='g', linestyle='None', marker='^')
sell.plot(color='r', linestyle='None', marker='^')
x1, x2, y1, y2 = plt.axis()
plt.axis((x1, x2, spread_test.min(), spread_test.max()))
plt.legend(['Spread', 'Buy Signal', 'Sell Signal'])
f3.show()

# Trade using a simple strategy
def trade(spd,avg,sd,delX,delY,initial_cap):
    # Simulate trading
    # Start with no money and no positions
    money = 0
    countX = 0
    countY = 0
    actual_buy_money = 0
    actual_sell_money = 0
    trade_count = 0
    gain_trade_count = 0
    loss_trade_count = 0
    money_gain = 0
    money_lost = 0
    tr_cost = 0
    hold_time = 0
    trade_start = 0
    max_hold_time = 0
    ini_cap = initial_cap
    pnl_per_iter = np.zeros(wndw)
    i=0
    while i < len(spd):
        # Sell short if the z-score is > 2
        if (zsc(spd[i],avg,sd) > 2):
            if countX != 1:
                money = money + spd[i]
                countX = -1
                countY = +1
                print('Sell spread at index ' + str(spd.index[i]) + ' , since zcore is ' + str(zsc(spd[i],avg,sd))  )
                actual_sell_money = (X[spd.index[i]]*50 - Y[spd.index[i]]*25)
                nv_X = X[spd.index[i]]*50
                nv_Y = Y[spd.index[i]]*25
                tr_cost = min(20,0.0003*nv_X) + min(20,0.0003*nv_Y) + 0.0001*nv_X + 0.00002*(nv_X + nv_Y) + 0.15*50 + 0.15*25
                margin = nv_X + nv_Y
                trade_start = i
            else:
                print('Square off position at index ' + str(spd.index[i]) + ' , since zcore is ' + str(zsc(spd[i],avg,sd))  )
                actual_buy_money += (X[spd.index[i]]*50 - Y[spd.index[i]]*25)
                nv_X = X[spd.index[i]]*50
                nv_Y = Y[spd.index[i]]*25
                tr_cost += min(20,0.0003*nv_X) + min(20,0.0003*nv_Y) + 0.0001*nv_X + 0.00002*(nv_X + nv_Y) + 0.15*50 + 0.15*25
                actual_buy_money = actual_buy_money - tr_cost
                print('Actual Money made in buy trade squared off at ' + str(spd.index[i]) + ' is ' + str(actual_buy_money)  )
                trade_count += 1
                trade_end = i
                hold_time += trade_end - trade_start
                max_hold_time = max(max_hold_time,trade_end-trade_start)
                ini_cap = ini_cap*(1 + actual_buy_money/margin)
                if actual_buy_money > 0:
                    gain_trade_count +=1
                    money_gain += actual_buy_money
                else:
                    loss_trade_count +=1
                    money_lost += actual_buy_money
                actual_buy_money = 0
                tr_cost = 0
                countX = 0
                countY = 0
        # Buy long if the z-score is < -2
        elif (zsc(spd[i],avg,sd) < -2):
            if countX == -1:
                print('Square off position at index ' + str(spd.index[i]) + ' , since zcore is ' + str(zsc(spd[i],avg,sd))  )
                actual_sell_money += -1*(X[spd.index[i]]*50 - Y[spd.index[i]]*25)
                nv_X = X[spd.index[i]]*50
                nv_Y = Y[spd.index[i]]*25
                tr_cost += min(20,0.0003*nv_X) + min(20,0.0003*nv_Y) + 0.0001*nv_Y + 0.00002*(nv_X + nv_Y) + 0.15*50 + 0.15*25
                actual_sell_money = actual_sell_money - tr_cost
                print('Actual Money made in sell trade squared off at ' + str(spd.index[i]) + ' is ' + str(actual_sell_money)  )
                trade_count += 1
                trade_end = i
                hold_time += trade_end - trade_start
                max_hold_time = max(max_hold_time,trade_end-trade_start)
                ini_cap = ini_cap*(1 + actual_sell_money/margin)
                if actual_sell_money > 0:
                    gain_trade_count +=1
                    money_gain += actual_sell_money
                else:
                    loss_trade_count +=1
                    money_lost += actual_sell_money
                actual_sell_money = 0
                tr_cost = 0
                countX = 0
                countY = 0
            else:
                money = money - spd[i]
                countX = 1
                countY = -1
                print('Buy spread at index ' + str(spd.index[i]) + ' , since zcore is ' + str(zsc(spd[i],avg,sd))  )
                actual_buy_money = -1*(X[spd.index[i]]*50 - Y[spd.index[i]]*25)
                nv_X = X[spd.index[i]]*50
                nv_Y = Y[spd.index[i]]*25
                tr_cost = min(20,0.0003*nv_X) + min(20,0.0003*nv_Y) + 0.0001*nv_Y + 0.00002*(nv_X + nv_Y) + 0.15*50 + 0.15*25
                margin = nv_X + nv_Y
                trade_start = i
        # Clear positions if the z-score between -0.75 and 0.75
        elif abs(zsc(spd[i],avg,sd)) < 0.75:
            money += delX[i]*countX + delY[i]*countY
            if countX==-1:
                print('Square off position at index ' + str(spd.
index[i]) + ' , since zcore is ' + str(zsc(spd[i],avg,sd))  )
                actual_sell_money += -1*(X[spd.index[i]]*50 - Y[spd.index[i]]*25)
                nv_X = X[spd.index[i]]*50
                nv_Y = Y[spd.index[i]]*25
                tr_cost += min(20,0.0003*nv_X) + min(20,0.0003*nv_Y) + 0.0001*nv_Y + 0.00002*(nv_X + nv_Y) + 0.15*50 + 0.15*25
                actual_sell_money = actual_sell_money - tr_cost
                print('Actual Money made in sell trade squared off at ' + str(spd.index[i]) + ' is ' + str(actual_sell_money)  )
                trade_count += 1
                trade_end = i
                hold_time += trade_end - trade_start
                max_hold_time = max(max_hold_time,trade_end-trade_start)
                ini_cap = ini_cap*(1 + actual_sell_money/margin)
                if actual_sell_money > 0:
                    gain_trade_count +=1
                    money_gain += actual_sell_money
                else:
                    loss_trade_count +=1
                    money_lost += actual_sell_money
                actual_sell_money = 0
                tr_cost = 0
            elif countX==1:
                print('Square off position at index ' + str(spd.index[i]) + ' , since zcore is ' + str(zsc(spd[i],avg,sd))  )
                actual_buy_money += (X[spd.index[i]]*50 - Y[spd.index[i]]*25)
                nv_X = X[spd.index[i]]*50
                nv_Y = Y[spd.index[i]]*25
                tr_cost += min(20,0.0003*nv_X) + min(20,0.0003*nv_Y) + 0.0001*nv_X + 0.00002*(nv_X + nv_Y) + 0.15*50 + 0.15*25
                actual_buy_money = actual_buy_money - tr_cost
                print('Actual Money made in buy trade squared off at ' + str(spd.index[i]) + ' is ' + str(actual_buy_money))
                trade_count += 1
                trade_end = i
                hold_time += trade_end - trade_start
                max_hold_time = max(max_hold_time,trade_end-trade_start)
                ini_cap = ini_cap*(1 + actual_buy_money/margin)
                if actual_buy_money > 0:
                    gain_trade_count +=1
                    money_gain += actual_buy_money
                else:
                    loss_trade_count +=1
                    money_lost += actual_buy_money
                actual_buy_money = 0
                tr_cost = 0
            countX = 0
            countY = 0
        pnl_per_iter[i] = ini_cap
        i = i+1
    return money,trade_count,gain_trade_count,loss_trade_count,money_gain,money_lost,hold_time,max_hold_time,ini_cap,pnl_per_iter

final_money = 0
Total_trades = 0
Gain_trades = 0
Loss_trades = 0
Gain = 0
Loss = 0
holding_time = 0
max_holding_time = 0
initial_capital = 1
pnl = np.zeros(2800)
j=0
while j<len(spread) - split - wndw:
    spread_train_w = spread[j:j+split]
    spread_train_w_mu = spread_train_w.mean()
    spread_train_w_std = spread_train_w.std()
    spread_test_w = spread[j+split:j+wndw+split]
    delX_test_w = diff_nifty50[j+split:j+wndw+split]
    delY_test_w = diff_niftybank[j+split:j+wndw+split]
    fm,tt,gt,lt,gn,ls,ht,mht,ic,pnlpi = trade(spread_test_w,spread_train_w_mu,spread_train_w_std,delX_test_w,delY_test_w,initial_capital)
    final_money += fm
    Total_trades += tt
    Gain_trades += gt
    Loss_trades += lt
    Gain += gn
    Loss += ls
    holding_time += ht
    max_holding_time = max(mht,max_holding_time)
    initial_capital = ic
    klm = 0
    while klm < wndw:
        pnl[j+klm] = pnlpi[klm]
        klm += 1
    j=j+wndw
print(final_money)
print('Accuracy of Pair Trading Algo is: ' + str(Gain_trades/
    Total_trades))
print('Gain Loss Ratio is: ' + str(abs((Gain*Loss_trades)/(Loss*
    Gain_trades))))
print('Overall Profit:' + str(Gain + Loss))
pnl_s = pd.Series(pnl)
pnl_s.index = diff_nifty50.index[split:-3]
return_pnl = pnl_s.pct_change(1)
return_pnl = return_pnl.dropna()
check_mu = return_pnl.mean()
check_sigma = return_pnl.std()
check_sr = check_mu/check_sigma*(252**0.5)
A_sharpe_ratio = (252**0.5)*return_pnl.mean()/return_pnl.std()
return_pnl_n = return_pnl[return_pnl<0]
check_mu1 = return_pnl_n.mean()
check_sigma1 = return_pnl_n.std()
check_sr1 = check_mu/check_sigma1*(252**0.5)
A_sortino_ratio = (252**0.5)*return_pnl.mean()/return_pnl_n.std()

def max_drawdown(return_series):
    comp_ret = (return_series+1).cumprod()
    # peak = comp_ret.expanding(min_periods=1).max()
    peak = comp_ret.cummax()
    dd = (comp_ret/peak)-1
    return dd.min()

max_dd = max_drawdown(return_pnl)
A_calmer_ratio = 252*return_pnl.mean()/abs(max_dd)

f4 = plt.figure(5,figsize=(20,10))
pnl_s.plot()
plt.title("PnL curve")
f4.show()

f5 = plt.figure(6,figsize=(20,10))
X.plot()
Y.plot()
f5.show()

# For regime classification
High_reg = pnl_s.copy()
Low_reg = pnl_s.copy()
mod_kns = sm.tsa.MarkovRegression(daily_ret_nifty[split:-3],
    k_regimes=2, trend='n', switching_variance=True)
```

```
res_kns = mod_kns.fit()
Low_var_regime_probs = res_kns.predicted_marginal_probabilities[0]
High_var_regime_probs = res_kns.predicted_marginal_probabilities[1]
i=0
while i<2800:
    if Low_var_regime_probs[i] > High_var_regime_probs[i]:
        High_reg[i] = -3
    else:
        Low_reg[i] = -3
    i+=1

# PnL in different regimes
f6 = plt.figure(7,figsize=(20,10))
pnl_s.plot()
Low_reg.plot(color='g',linestyle = 'None', marker='o')
High_reg.plot(color='r',linestyle = 'None', marker='o')
x3, x4, y3, y4 = plt.axis()
plt.axis((x3, x4, pnl_s.min(), pnl_s.max()))
plt.legend(['PnL', 'PnL in Low regime', 'PnL in High regime'])
f6.show()

# Performance metrics in Low regime
return_pnl_low = Low_reg[Low_reg>0].pct_change(1)
A_sharpe_ratio_low_reg = (252**0.5)*return_pnl_low.mean()/
    return_pnl_low.std()
max_dd_low = max_drawdown(return_pnl_low)
A_calmer_ratio_low_reg = 252*return_pnl_low.mean()/abs(max_dd_low)

# Performance metrics in High regime
return_pnl_high = High_reg[High_reg>0].pct_change(1)
A_sharpe_ratio_high_reg = (252**0.5)*return_pnl_high.mean()/
    return_pnl_high.std()
max_dd_high = max_drawdown(return_pnl_high)
A_calmer_ratio_high_reg = 252*return_pnl_high.mean()/abs(max_dd_high
    )
```

Bibliography

[1] Daniel Amsköld. “A comparison between different volatility models”. In: (2011). URL: https://www.math.kth.se/matstat/seminarier/reports/M-exjobb11/110617.pdf.

[2] Tushar Arora. “Forecasting volatility using Markov models”. In: (Nov. 2022). URL: https://blog.algoanalytics.com/2022/11/01/forecasting-volatility-using-markov-models/.

[3] Tushar Arora. “Modeling Volatility Clustering using GARCH model for NIFTY index”. In: (Oct. 2022). URL: https://blog.algoanalytics.com/2022/10/10/modeling-volatility-clustering-using-garch-model-for-nifty-index/.

[4] Tushar Arora. “Volatility Clustering”. In: (Oct. 2022). URL: https://blog.algoanalytics.com/2022/10/03/volatility-clustering/.

[5] Tushar Arora. “What is Volatility?” In: (Sept. 2022). URL: https://blog.algoanalytics.com/2022/09/26/what-is-volatility/.

[6] Fischer Black and Myron Scholes. “The Pricing of Options and Corporate Liabilities”. In: *Journal of Political Economy* 81.3 (1973), pp. 637–654. ISSN: 00223808, 1537534X. URL: http://www.jstor.org/stable/1831029 (visited on 04/19/2023).

[7] Tim Bollerslev. “A Conditionally Heteroskedastic Time Series Model for Speculative Prices and Rates of Return”. In: *The Review of Economics and Statistics* 69.3 (1987), pp. 542–547. ISSN: 00346535, 15309142. URL: http://www.jstor.org/stable/1925546 (visited on 04/19/2023).

[8] Tim Bollerslev. “Generalized autoregressive conditional heteroskedasticity”. In: *Journal of Econometrics* 31.3 (1986), pp. 307–327. ISSN: 0304-4076. DOI: https://doi.org/10.1016/0304-4076(86)90063-1. URL: https://www.sciencedirect.com/science/article/pii/0304407686900631.

[9] John C. Cox, Jonathan E. Ingersoll, and Stephen A. Ross. “A Theory of the Term Structure of Interest Rates”. In: *Econometrica* 53.2 (1985), pp. 385–407. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1911242 (visited on 04/19/2023).

[10] D. Dickey and Wayne Fuller. “Distribution of the Estimators for Autoregressive Time Series With a Unit Root”. In: *JASA. Journal of the American Statistical Association* 74 (June 1979). DOI: 10.2307/2286348.

[11] Robert F. Engle. “Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation”. In: *Econometrica* 50.4 (1982), pp. 987–1007. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1912773 (visited on 04/19/2023).

[12] Robert F. Engle, David M. Lilien, and Russell P. Robins. “Estimating Time Varying Risk Premia in the Term Structure: The Arch-M Model”. In: *Econometrica* 55.2 (1987), pp. 391–407. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1913242 (visited on 04/19/2023).

[13] Robert F. Engle and Andrew J. Patton. “What good is a volatility model?*”. In: *Forecasting Volatility in the Financial Markets (Third Edition)*. Ed. by John Knight and Stephen Satchell. Third Edition. Quantitative Finance. Oxford: Butterworth-Heinemann, 2007, pp. 47–63. ISBN: 978-0-7506-6942-9. DOI: https://doi.org/10.1016/B978-075066942-9.50004-2. URL: https://www.sciencedirect.com/science/article/pii/B9780750669429500042.

[14] Peter R. Hansen and Asger Lunde. “A forecast comparison of volatility models: does anything beat a GARCH(1,1)?” In: *Journal of Applied Econometrics* 20.7 (2005), pp. 873–889. DOI: https://doi.org/10.1002/jae.800. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jae.800. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jae.800.

[15] Steven L. Heston. “A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options”. In: *The Review of Financial Studies* 6.2 (1993), pp. 327–343. ISSN: 08939454, 14657368. URL: http://www.jstor.org/stable/2962057 (visited on 04/19/2023).

[16] Ching Mun Lim and Siok Kun Sek. “Comparing the Performances of GARCH-type Models in Capturing the Stock Market Volatility in Malaysia”. In: *Procedia Economics and Finance* 5 (2013). International Conference On Applied Economics (ICOAE) 2013, pp. 478–487. ISSN: 2212-5671. DOI: https://doi.org/10.1016/S2212-5671(13)00056-7. URL: https://www.sciencedirect.com/science/article/pii/S2212567113000567.

[17] Daniel B. Nelson. “Conditional Heteroskedasticity in Asset Returns: A New Approach”. In: *Econometrica* 59.2 (1991), pp. 347–370. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/2938260 (visited on 04/19/2023).

[18] Sudhakar Madhavrao Pandit, Shien-Ming Wu, and Talivaldis I. Šmits. *Time Series and System Analysis with Applications by Sudhakar Madhavrao Pandit and Shien-Ming Wu*. Vol. 75. 6. 1984, pp. 1924–1925. DOI: 10.1121/1.390924. eprint: https://doi.org/10.1121/1.390924. URL: https://doi.org/10.1121/1.390924.

[19] G. WILLIAM SCHWERT. "Why Does Stock Market Volatility Change Over Time?" In: *The Journal of Finance* 44.5 (1989), pp. 1115–1153. DOI: https://doi.org/10.1111/j.1540-6261.1989.tb02647.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1989.tb02647.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1989.tb02647.x.

[20] "The Heston (1993) Stochastic Volatility Model". In: *Option Pricing Models and Volatility Using Excel®-VBA*. John Wiley Sons, Ltd, 2012. Chap. 5, pp. 136–162. ISBN: 9781119202097. DOI: https://doi.org/10.1002/9781119202097.ch5. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119202097.ch5. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119202097.ch5.

[21] Yunying Zhu. "Comparison of Three Volatility Forecasting Models". In: (2018). URL: https://kb.osu.edu/bitstream/handle/1811/84909/1/Thesis.pdf.