Norwegian School of Economics
Bergen, Autumn 2023

Fear and Forecasting in Scandinavia

Implied Volatility Indices for the Scandinavian Equity Markets

Vebjørn Kydland and Martin Vagle

Supervisor: Jørgen Haug

Master thesis, Economics and Business Administration

Major: Economic Analysis

NORWEGIAN SCHOOL OF ECONOMICS

Acknowledgements

This thesis was written as part of the Master of Science in Economics and Business Administration program at the Norwegian School of Economics (NHH), with a major in Economic Analysis. Throughout the writing process, we faced challenges, gained interesting insights, and learned a great deal. We hope the reader finds the topic as intriguing as we found it rewarding to work on.

We extend our gratitude to the Nasdaq European Data Team for supplying us with the necessary data to conduct our research and for their patience during the data access process.

Special thanks to our families, classmates and friends (including the four-legged ones) for their constant support and reassurance. Our appreciation goes to it.gruppen for allowing us the use of their facilities during the semester, and for the ’liquid motivation’ readily available on location.

Finally, we give special thanks to our supervisor, Jørgen Haug. He was always ready to answer questions, both simple and complex, and his advice was invaluable to the quality of this thesis.

Norwegian School of Economics

Bergen, December 2023

Vebjørn Kydland

Martin Vagle

Abstract

This thesis investigates the use of the model-based and model-free implied volatility index methodologies in Scandinavia from 2018 to 2023, leading to the creation of a composite index for the region: SCANDI-VIX. It confirms a significant negative contemporaneouss relationship between the Scandinavian implied volatility indices (NORVIX, DANVIX, SWEVIX, SCANDI-VIX) and their underlying index returns, validating their role as a "fear gauge." The study reveals an asymmetric response of these implied volatility indices to market returns, aligning with the leverage effect theory. We investigate the structural changes in the underlying market time-series as a consequence of the COVID-19 pandemic, and it's implications for EGARCH forecasting. With a final key finding being the increased forecasting quality observed when utilising implied volatility as an input feature in the Extreme Gradient Boosting (XGBoost) machine learning model.

Keywords – VIX, Model-Free, Model-Based, Scandinavia, Forecasting, XGBoost

Contents

List of Figures

List of Tables

1 Introduction

In 1993, the Chicago Board Options Exchange (CBOE) launched the VIX index which was developed from the work done by Whaley (1993). It was launched to act as the benchmark for estimating the expected volatility of the S&P100 index, earning it the nickname “Investor Fear Gauge” (Whaley, 2000). Over the past 30 years, the VIX has gained significant popularity for providing transparency in the derivatives market by showing the amount of volatility that is incorporated in derivative prices. It functions like a barometer of the broader market, rising in times of increased uncertainty. Moreover, it is used for speculating or hedging strictly on volatility as there are listed tradable futures and options on the VIX. Additionally, there have been ongoing discussions about the predictive capability of implied volatility in relation to actual market volatility. Further, Slim et al. (2020) has shown that implied volatility can be used as an input variable for a Value-at-Risk calculation, helping investors manage risk better.

Global economies are currently facing the challenge of controlling inflation. The situation arised partly due to increased public spending and generous policies during the recent coronavirus pandemic. In response, central banks are raising interest rates, a move that significantly raises the risk of triggering a widespread recession. These economic conditions often remind investors of the potential for major stock market downturns. In such uncertain times, having a reliable gauge on the uncertainty in asset returns would be very beneficial. In asset pricing, the uncertainty of asset returns is termed risk and is often measured by an asset’s volatility. As opposed to realised volatility, which is backward-looking, the VIX is based on implied volatility and offers a forward-looking perspective.

Since its launch, the VIX has been present during significant market downturns, including the dot-com bubble, the 2008 financial crisis, and the recent COVID-19 pandemic, each accompanied by notable spikes in the VIX index. This paper takes a closer look at the years leading up to and following the COVID-19 pandemic, with a special focus on the Scandinavian markets. While much of the existing research, as Fassas and Siriopoulos (2020) notes, centres on US-based equities, our study aims to fill a gap in the literature by focusing on implied volatility in Scandinavia.

We construct and compare the implied volatility for Scandinavian benchmark indices: For Norway, the NORVIX is derived from the OMXO20 index, for Sweden, the SWEVIX from the OMXS30 index, and for Denmark, the DANVIX from the OMXC25 index. We calculate each implied volatility index based on the VIX's former model-based methodology and the current model-free methodology. Then assess which approach is most suitable to the conditions of the Scandinavian markets. Initially, our intention was to utilize the Nasdaq index for the Nordic market (OMXN40). However, due to the lack of tradable derivatives on OMXN40 and its primary role as a reference index, we opted to create a weighted composite implied volatility index based on the semi-annual rebalancing of the three Scandinavian indices. We have named this the SCANDI-VIX.

Our study delves into the empirical properties of the implied volatility indices, the relationship between returns and volatility, and the informational content of implied volatility regarding forward-realised volatility. Notably, we draw on Bugge et al. (2015) who constructed a Norwegian implied volatility index, NOVIX, explored its empirical properties, and assessed its forecasting abilities. Lastly, we base a lot of our empirical properties on Fassas and Siriopoulos (2020) review, where they compare 68 implied volatility indices.

We compare the SCANDI-VIX, with the OMXN40 to see if it can act as an implied volatility proxy index. Our objective is twofold: Firstly, to analyse how well the underlying assets of the SCANDI-VIX correlate with the OMXN40, and secondly, to investigate whether the relationship between the SCANDI-VIX and its own underlying assets diverge from its relationship with the OMXN40, as one comprising 40 constituents and the other 75. The outcome of this comparison will shed light on the potential of the SCANDI-VIX as a predictive and descriptive tool for OMXN40 movements.

To concisely outline the hypotheses addressed in this paper:
H1: There exists a negative contemporaneous relationship between the Scandinavian implied volatility indices returns and underlying indices returns, thus they will act as a fear gauge.
H2: The return-volatility relationship is asymmetric, meaning the Scandinavian implied volatility indices react differently to negative and positive returns.
H3: Using implied volatility for forecasting surpasses GARCH models, especially when it's integrated into the machine learning algorithm XGBoost.

H4: The composite proxy of the three Scandinavian implied volatility indices will absorb information well enough for it to serve as a useful predictive and descriptive tool of the OMXN40.

If the first hypothesis confirmed, it will demonstrate that our implied volatility indices act in accordance with typical market behaviour. Where investors perceive more risk when the market is performing poorly, and it can thus act as a fear gauge for the markets.

Our second hypothesis suggests non-uniformity in the Scandinavian implied volatility indices' reactions to their underlying indices' returns. Also known as the "leverage effect", where markets tend to exhibit greater volatility during underlying index declines, than during ascents. From a behavioural perspective, it can be argued that return asymmetry reflects investor risk aversion. It is also crucial to investigate the asymmetry prior to selecting forecasting models, as certain models might not handle the non-uniformity.

The third hypothesis explores the possibility that implied volatility in forecasting could exceed the performance of conventional GARCH models. We delve deeper into the efficacy of implied volatility by integrating it into an XGBoost machine learning framework. We will train these models using data spanning from 2018 to 2022 and then conduct an empirical evaluation on a monthly basis, from January 2023 through November 2023, to determine the accuracy and reliability of their forecasts.

The final hypothesis lays the groundwork for applying proxy indices to various techniques when trading similar underlying assets. This approach could potentially enable hedging strategies against volatility in ETFs or other traded instruments tracking the OMXN40.

Our paper is organised to initially provide a background on the two common methods used for calculating an implied volatility index. Followed by a literature review covering topics relevant to our research. Subsequently, we detail the process of data collection and modification, which involves acquiring bid and ask prices for options that serve as input for our implied volatility indices. Onwards we outline the methodologies employed for constructing the implied volatility indices and for conducting tests on these indices. We present our findings and discussions, including any limitations identified. The paper concludes with a summary of our results in relation to the initial hypothesis.

2 Background

This chapter presents the CBOE VIX index, including the concepts and methodologies behind the calculations. We begin by outlining the model-based and model-free approaches to calculating the implied volatility indices, followed by a review of relevant empirical literature. Finally we review relevant literature on the characteristics of the implied volatility indices such as negative return relationships, asymmetry and volatility forecasting.

2.1 The VIX Index

As of today, the CBOE VIX index estimates the 30-day expected volatility of the S&P500 index (SPX). The estimation is derived from the midpoint prices between bid and ask for a broad set of SPX options, encompassing both puts and calls as outlined in the CBOE white paper (Cboe Global Indices, LLC, 2023). The resulting index is then expressed in percentage terms as an annualized move of return on the S&P500 index. For instance, a VIX index value of 20 suggests that the market anticipates the SPX to fluctuate by 20% annually, or $20/\sqrt{12} = 5.77\%$ monthly.

The creation of implied volatility indices typically follows one of two methodologies: model-based or model-free. Carr and Wu (2005) provide a great comparison of these approaches, but we will offer an overview.

2.1.1 The model-based VXO

Until 2003, the VIX index used the S&P100 as its underlying index. This VIX index would later be known as VXO and was calculated using the Black-Scholes at-the-money implied volatilities (Carr & Wu, 2005). This methodology is often referred to as a model-based methodology because the Black-Scholes option pricing model is needed to compute the implied volatility index (Gonzalez-Perez & Novales, 2011).

To find the implied volatility we can observe that the price of an option is a function of five variables (Hull, 2022, p. 352): *price=f(S,K,T,r,σ)*. The spot price (S), strike price (K), time to maturity (T), and risk-free rate (r) are all directly observable in the market or from specific option specifications, but this is not the case for the volatility of the underlying asset (σ). Since option prices are observable in the market, we can numerically solve for what the implied volatility per option would be.

The computation of a value for the VXO index relies on the implied volatility of eight options (Carr & Wu, 2005). From each of the two nearest maturities four near-the-money options are used. This would be the put-call pair for the two strikes on either side of the spot price. The implied volatility for the put-call pair would then be averaged for each strike price. According to the put-call parity, the implied volatility for the put and call in each pair should theoretically be identical (Hull, 2022, p. 255). However, in practice there might be a slight discrepancy due to market inefficiency. Averaging out these differences might give a more holistic view than only using one of the options. Once the implied volatilities for the put-call pairs at each strike price have been averaged, the next step involves using linear interpolation to get the implied volatility at the spot price. This interpolated figure represents the at-the-money implied volatility (ATMV) for the maturity.

Instead of using the ATMV directly, the CBOE introduced a trading day conversion to better represent the traded days (Carr & Wu, 2005). This adjustment is achieved by multiplying the ATMV with a factor that considers the number of calendar days and trading days until option maturity. We denote this adjusted measure as the trading day implied volatility (TV), calculated as follows:

$$TV = ATMV \times \frac{\sqrt{D^{\text{Calendar}}}}{\sqrt{D^{\text{Trading}}}} \tag{2.1}$$

Where D^{Calendar} and D^{Trading} represent the number of calendar days and trading days, respectively, until the option's maturity. To construct the VXO value, the CBOE then interpolates between the at-the-money implied volatilities of two maturities to obtain a 22-trading day volatility:

$$VXO = TV_1 \times \frac{D_2^{\text{Trading}} - 22}{D_2^{\text{Trading}} - D_1^{\text{Trading}}} + TV_2 \times \frac{22 - D_1^{\text{Trading}}}{D_2^{\text{Trading}} - D_1^{\text{Trading}}} \tag{2.2}$$

In this formula, subscript 1 refers to the nearest maturity, while subscript 2 indicates the next closest maturity. Given that there are approximately 22 trading days in a month, the VXO effectively represents a one-month at-the-money implied volatility.

However, the method of calculations has received criticism from academia and the industry (Carr & Wu, 2005). As there are more calendar days than trading days in a month, this method introduces an upward bias in the volatility estimation. Consequently, it is no longer comparable to the annual realised volatilities computed from index returns on a 365-day calendar.

The CBOE sustained the VXO until it was discontinued during August 2021. This was due to the index using a "legacy methodology to measure the implied volatility of an underlying asset" as stated by the Cboe Global Markets (2021). Additionally, the absence of a straightforward method to replicate the VXO's returns for the purpose of creating derivative products meant that no derivative products were ever developed based on this index (Carr & Wu, 2005).

2.1.2 The model-free VIX

Currently, the CBOE employs a different methodology from the Black-Scholes framework for calculating the VIX index. This newer approach of the VIX was introduced on September 22, 2003, when the CBOE transitioned the VIX's underlying index to the S&P500, a more popular index. The calculation is now relying on the fair value of variance swaps, a concept originating from the model-free implied variance as proposed by Britten-Jones and Neuberger (2000).

(Demeterfi et al., 1999) demonstrated that variance swaps can be replicated by a hedged portfolio of standard options with appropriate strikes. Under the assumption of no arbitrage, the fair value of a variance swap equates to the cost of this replicated portfolio. Consequently, the revised CBOE VIX methodology, detailed in the Cboe Global Indices, LLC (2023), calculates variance directly from option prices for each strike. The general formula for the variance for a single maturity date is defined as:

$$\sigma^2 = \frac{2}{T}\sum_i \frac{\Delta K_i}{K_i^2} e^{rT} Q(K_i) - \frac{1}{T}\left(\frac{F}{K_0} - 1\right)^2 \tag{2.3}$$

T represents the time to expiration, r is the risk-free rate, K_i is the strike price of the i-th out-of-the-money (OTM) option, $Q(K_i)$ is the midpoint price of the bid-ask spread for the option at strike K_i, F stands for the put-call parity implied forward price where the put and call prices are closest, K_0 is the strike price immediately below the forward price. ΔK_i is the interval between strike prices and are defined as:

$$\Delta K_i = \frac{K_{i+1} - K_{i-1}}{2} \tag{2.4}$$

For the edge cases ΔK_i is the difference between K_i and the adjacent strike price. The rationale behind the formula is that the price of each OTM option contributes to the overall variance, with this contribution being weighted by the intervals between strike prices. Essentially, options at various strike levels influence the variance calculation to different extents, based on their relative positioning. $2/T$ represents the process of annualizing the variance over the period from the present to the expiration (time T). Regarding the final term in the formula, it serves as an adjustment linked to the forward rate used in the variance swap replication. The term corrects for the small discrepancy when the forward price F is not exactly equal to the strike price immediately below it K_0 (Hull, 2022, p. 629).

VIX calculation is based on two maturities, the "near-term" and "next-term". These terms refer to the option expiration dates that fall immediately before and after a fixed 30-day period, known as the constant maturity term. To determine the variance at this constant maturity date, the CBOE employs a linear interpolation between the near-term and next-term variances. The formula for the implied volatility (IV) index value is as follows:

$$\text{IV Index} = 100 \times \sqrt{\left\{ T_1\sigma_1^2 \left[\frac{M_{T_2} - M_{CM}}{M_{T_2} - M_{T_1}}\right] + T_2\sigma_2^2 \left[\frac{M_{CM} - M_{T_1}}{M_{T_2} - M_{T_1}}\right] \right\} \times \frac{M_{365}}{M_{CM}}} \tag{2.5}$$

In this formula M_{T_1} and M_{T_2} denote the number of minutes until expiration of the near-term and next-term options respectively. M_{CM} are the number of minutes in the given constant maturity term and M_{365} is the number of minutes in a 365-day year. T_i is defined as M_{T_i}/M_{365} The variance terms, σ_1^2 and σ_2^2, are the calculated variances for the near-term and next-term maturities, respectively.

Most implied volatility indices available today adopt a model-free method with a 30-day constant maturity date (Fassas & Siriopoulos, 2020). Carr and Wu (2005) outline three benefits of this model-free index. First, the VIX now represents the price of a portfolio of options, providing it with a more tangible economic interpretation than the VXO, which only represented a monotonic nonlinear transformation of the at-the-money option prices. Second, this method avoids the upward bias associated with the VXO, as it is based on a 365-day year. Third, the approach's replicability has enabled the CBOE to efficiently launch options and futures based on the VIX.

Jiang and Tian (2007) have highlighted certain issues with the current methodology used in the index calculation, particularly focusing on truncation and discretization errors. Truncation errors emerge from excluding strike prices outside the range of listed strikes, while discretization errors result from the gaps between listed strike prices. Their findings suggest that these errors can lead to underestimation or overestimation of the actual volatility by as much as 198 and 79 index basis points, respectively. Considering that each basis point is worth USD10 per VIX futures contract this can be substantial. To mitigate these issues, Jiang and Tian propose a smoothing technique involving the use of natural cubic spline interpolation to create an implied volatility function. Subsequently, the Black-Scholes-Merton model is applied to determine the prices from these implied volatilities, which are then used as input for the CBOE method. This approach effectively reduces the error margin to approximately ± 8 index basis points.

Although Jiang and Tian (2007) identified certain limitations with the CBOE's methodology, we will continue to employ it in our analysis. This comes from the fact that most of the strike price intervals in our study fall below 1% of the strike price. In their research, Jiang and Tian analysed scenarios with strike price intervals ranging from 1% to 2.5%, finding that discretization errors were significantly higher for intervals at the upper end of this spectrum. Similarly, truncation errors are minimal if the edge strike prices of the listed options are more than 10% away from the spot price. In our situation, this is often the case. In addition, adopting the CBOE's methodology provides a consistent and comparative framework for evaluating our indices against an established benchmark, the VIX.

2.2 Literature

There is comprehensive research on implied volatility indices, we will provide some of the literature relevant to our theses divided into sections on different topics.

2.2.1 On return relationships and asymmetry

Empirical literature on the contemporaneous relationship between implied volatility and underlying asset returns suggests a negative relationship. According to Whaley (2000) an implied volatility index can be seen as an investor fear gauge if changes in the index maintain a significant and negative relationship with stock market returns. We anticipate an asymmetric response in the return series, where negative return shocks increase volatility more than positive returns decrease it. Black (1976) and Campbell and Hentschel (1992) propose different explanations for return asymmetry, respectively: The leverage hypothesis and the volatility feedback hypothesis.

The leverage hypothesis, as proposed by Black (1976) explains the asymmetry as a relationship between companies' leverage and volatility in equity returns. Companies taking up debt-financing, are committing to fixed interest payments, which then hikes risk during low earning periods for said company. This translates to an increased sensitivity to negative equity returns. The leverage effect hypothesis remains inconclusive however, as reported by Figlewski and Wang (2000), who found no link between volatility and leverage changes due to change in debt or numbers of shares. They conclude that the cause of implied volatility index changes to be solely based on the change in underlying stock price.

Campbell and Hentschel (1992) proposed the volatility feedback hypothesis to explain the asymmetry as a feedback loop between stock market volatility and stock returns. When investors demand higher risk premia for holding stocks during high market volatility, the present value of said stock lowers, leading to a decrease in prices. The negative returns then contribute to volatility in the market, which constitutes the feedback loop.

Fassas and Siriopoulos (2020) contains a large overview of empirical contemporaneous return relationships and asymmetry statistics from 68 implied volatility indices. They linearly regressed log-returns of the implied volatility indices on the underlying index log-returns to assess the negative contemporaneous return relationship. The same regression was used by Bugge et al. (2016). Both Fassas and Siriopoulos (2020) and Bugge et al. (2016) find that the contemporaneous return relationship between the implied volatility indices investigated and their underlying indices, is negative. Bugge et al. (2016) confirms asymmetry in the returns by means of a multiple linear regression. Where they regress implied volatility index log-returns on the positive return series and negative return series of the underlying index. Fassas and Siriopoulos (2020) compliments this exercise with an identical regression over each non-overlapping 10th percentile in the return series, confirming asymmetry. They also describe the monotonically increasing effects in the returns series for each quantile investigated. Meaning that negative underlying market movements cause a greater positive reaction in the implied volatility index, dependent on the magnitude of said market movement.

2.2.2 On stationarity and forecasting

Several of the papers mentioned already attempt to predict future realised volatility by means of regressions or the ARCH models. Bugge et al. (2016) compares the GARCH models with the HAR model, using both implied volatility and realised volatility. Concluding that HAR models outperform GARCH models by introduction of implied volatility. To assess predictive qualities of the implied volatility indices, Fassas and Siriopoulos (2020) utilises a linear regression model, an autoregressive model, and a multiple regression model with lagged variables. Teller et al. (2022), use the extreme gradient boosting framework (XGBoost) to predict future realised volatility. Comparing it with a HAR model and a long-term short-term memory (LSTM) deep-learning model, they report that including implied volatility in their forecasting increases accuracy. XGBoost is based on a decision tree model and was made open source by the creators Chen and Guestrin (2016). Teller et al. (2022) utilises grid searching to their extreme gradient boosting model's hyperparameters. Grid searching is training the model for each possible combination of hyperparameters and saving the best fit. Fassas and Siriopoulos (2020) and Bugge et al. (2016) report the robustness of their forecasts by mean squared errors, with Teller et al. (2022) including mean absolute errors in addition.

Due to the non-stationarity of index levels, it is widely accepted that it is better to use log-returns for prediction. Non-stationarity in data, implies that the data's statistical properties like mean, variance and autocorrelation are not constant. Levels exhibit changing mean over time, something log-returns often avoids. Stationarity is critical in volatility forecasting, particularly in GARCH models. GARCH models always assume weak stationarity in the input data, that being a constant mean and variance. Should the input data be non-stationary, it will lead to parameter instability and poor out of sample predictions. To check whether our log-return series are stationary, we employ the ADF test (Dickey & Fuller, 1979) and KPSS test (Shin & Schmidt, 1992). The ADF test's null hypothesis is the existence of a unit root. A unit roots existence in the series implies non-stationarity. The KPSS test's null hypothesis compliments this, being the existence of stationarity around a mean or deterministic trend. By cross-referencing the tests' null hypotheses, we can identify inconclusive results if both tests' null hypotheses are either rejected or both holds.

2.2.3 On national volatility indices

Numerous studies have developed implied volatility indices for major benchmark indices in various countries. Our primary interest lies in the research concerning Scandinavian countries, supplemented by a comprehensive review by Fassas and Siriopoulos (2020).

"Implied volatility index for the Norwegian equity market" (Bugge et al., 2016) is the basis of our thesis. They created the NOVIX for Norway based on the OBX index options where they used daily close data on all available call and put options for the period January 3, 2000, to February 22, 2016. Due to low volumes in the early years, they adjusted the start date of their data set from 2000 to January 3, 2006. The Norwegian Interbank rate (NIBOR) was used as the risk-free rate proxy, and they employed the model-free method.

In a master's thesis by Öström (2015), the SVIX for Sweden was developed using the model-free method and based on the OMXS30 option. The index was constructed daily from January 3, 2005, to June 11, 2015, The Stockholm Interbank Offered Rate (STIBOR) was used as the risk-free rate proxy. He showcases non-normality, stationarity, and autocorrelation in the SVIX return series. In the paper he assesses the forecast quality of SVIX by means of linear and multiple regression, using exclusively realised volatility and the returns on the SVIX.

To the best of our knowledge, there hasn't been a dedicated implied volatility index developed for Denmark. However, a study on implied volatility was conducted by Hansen (1999), regressing realised volatility on fitted implied volatility. Hansen's study focused on the implied volatility of options based on the Danish KFX index, the predecessor to the OMXC25. In contrast to the model-free approaches used in the other Scandinavian studies, Hansen employed a variation of Black-Scholes method to extract implied volatility. Her data consisted of 33 observations monthly, from September 1995 to April 1998 and used Copenhagen Interbank Offered Rate (CIBOR) for a risk-free rate proxy. The study finds that the volatility implied by the KFX index options is an efficient predictor of future realised return volatility, outperforming historical volatility measures.

The GVIX implied volatility index, introduced by Skiadopoulos (2004), is derives from the options on FTSE/ASE-20. He analyses the impact of using either options' bid-ask or settlement prices for calculating this index. Throughout most of the observed period, both pricing methods show similar trends in GVIX movements. However, Skiadopoulos points out a stronger correlation between implied volatility calculated from settlement prices and the underlying index. In addition, he argues that settlement prices are less susceptible to manipulation. He further finds that the GVIX can be used as an investor fear gauge and in volatility forecasting. He uses a time-series consisting of the period from September 2000 to December 2002.

The Spanish volatility indices, VIBEX and VIBEX-NEW (Gonzalez-Perez & Novales, 2011), employ both the model-based and the model-free methodology. This approach provides insight into the changes brought about by the model-free methodology as compared to the traditional model-based methodology. The relative simplicity of the model-free methodology is highlighted, allowing for the creation of implied volatility indices in less liquid markets. The model-based methodology, requiring a minimum of eight near-the-money options each day, proved inadequate in markets where days without the required options are frequent. In the same study, the predictive strengths of these implied volatility indices are explored. Gonzales-Perez and Novales report that these indices produce forecasts with comparable strength to those generated by GARCH and historical volatility models.

2.3 Adding to the Literature

In this thesis we adapt the CBOE model-based and model-free methodology to the Scandinavian countries, Norway, Sweden, and Denmark, and introduce a composite index for the whole region. The research focuses on the dynamics of the implied volatility indices during the COVID-19 pandemic. Building upon the groundwork laid by Bugge et al. (2016)) in Norway, this investigation examines if a newly developed Norwegian implied volatility index mirrors the behaviour of the previously established NOVIX. Notably, this research pioneers the creation of the first Danish implied volatility index, offering insights into the implied volatility characteristics of a market that has not been extensively studied before. For Sweden, an implied volatility index is formulated using more contemporary data. These newly constructed implied volatility indices are then benchmarked against the VIX to determine if the typical attributes of an implied volatility index are consistent within the Scandinavian context.

Moreover, the study delves into the predictive capabilities of these indices. By adapting the XGBoost algorithm to Scandinavian data, the research evaluates its effectiveness against the GARCH family of models in forecasting realised volatility. This approach also tests the impact of incorporating implied volatility into predictive modelling, assessing whether such integration markedly enhances the accuracy of future market risk forecasts.

3 Data

Our data comes from various sources, with the majority originating from Nasdaq. In this section, we'll clarify the methods and locations of data collection, preparation, and modification. Additionally, we will discuss the rationale behind our decisions.

3.1 Nasdaq data

3.1.1 Underlying indices options

As we make one implied volatility index for each of the Scandinavian countries, we use options from three underlying indices. The OMXS30, OMXC25, and OMXO20 indices represent the Swedish, Danish, and Norwegian stock markets respectively, featuring the most traded shares on their respective exchanges (Nasdaq, 2023). OMXC25 and OMXO20 options have lifetimes of 3, 12 and 24 months at issue, with contract sizes of DKK 100 and NOK 100 respectively. OMXS30 options have 3, 18, and 60-month lifetimes at issue, with a contract size of SEK 100. All are European style options expiring on the third Friday of their expiration month. Additionally, OMXS30 offers weekly options expiring each Friday, except on the third Friday of the month when monthly options expire.

We opted for the OMXO20 index from Nasdaq instead of the OBX index from Oslo Stock Exchange (Euronext). Both indices share many of the same constituent companies, although this varies over time. The OBX includes 25 companies compared to the OMXO20's 20. During our sample period, the OBX and OMXO20 prices had a very high correlation of 0.996, assuring us that the implied volatility index based on the OMXO20 would closely resemble one based on the OBX. This choice streamlined data collection, allowing us to source all information from Nasdaq.

3.1.2 Bid-ask and transaction prices

In deciding the input for constructing the implied volatility indices, we had the choice of either using transaction prices or prices based on the bid-ask quotes of the options. The CBOE prefers using the bid-ask midpoint, a method also endorsed by Fleming et al. (1995) as being superior to actual transaction prices. They contend that reliance on transaction prices could lead to negative first-order autocorrelation in changes to implied volatility, as option prices fluctuate between bid and ask levels. On the other hand, Skiadopoulos (2004) discovered that indices built using settlement prices yield returns that are more correlated with those of the underlying compared to those using the bid-ask prices. He also notes that indices based on bid-ask quotes tend to be more susceptible to noise.

Given the relative illiquidity of the Scandinavian option market compared to options on the S&P500, we realised that relying solely on transaction prices often resulted in too few data points to calculate an index value. A minimum requirement for the model-free methodology is having both a call and put price for the same strike, applicable to both next- and near-term maturities. The model-based methodology requires even more. For instance, while attempting to construct the DANVIX using transaction prices, we only managed to compute five values over a period of almost six years. Additionally, the sparse trading volume raises concerns about the relevance of transaction prices by day's end. Prices executed in the morning or mid-day might become outdated. This is because option prices fluctuate alongside their underlying assets and should ideally reflect the closing price of these assets. Consequently, we decided to use the midpoint price of the bid-ask spread.

3.1.3 ITCH messages

These bid and ask prices were only accessible through ITCH messages. With help from NASDAQ Stockholm and the NASDAQ European DataOps Team we got access to historical ITCH messages covering the period January 1, 2018, until November 10, 2023, excluding weekends and holidays. These messages enabled us to construct order books for the traded options, from which we could then extract the bid-ask prices. The ITCH messages are high frequency (nanoseconds) messages recording every interaction with Nordic Nasdaq stock exchange regarding options available on the exchange. The messages detail all added, deleted, and modified orders by market participants, including market makers. Following a system update by Nasdaq in March 2022, two different ITCH formats were provided. For data prior to April 2022, a format now referred to as GITS was used, while data from April 2022 onwards is in a format called NITCH. Summary tables outlining key message types used are available in Appendix A. For more detailed information on these specific ITCH formats, please refer to the *Genium INET®ITCH*

Protocol Specification (2017) for GITS, and *ITCH Specification for Nasdaq Derivatives Markets: Nordic Equity Derivatives* (2022) for NITCH.

The dataset we received included information on every underlying asset with derivatives traded on the Nordic Nasdaq exchanges. To tailor it to our needs, we filtered the data to include only messages related to derivatives with the underlying indices OMXO20, OMXC25, and OMXS30. The combined size of the data came to be 1.2TB in compressed material (.zip for GITS, .gz for NITCH) accumulating to 120,024,344,639 ITCH messages. After filtering out the irrelevant messages we were left with 24,768,088,402 messages. This refined subset represents approximately 21% of the initially retrieved data and constitutes 235.94 GB of compressed data.

After the data where downloaded it was necessary to process it to obtain relevant metrics for calculating the implied volatility indices. Guided by Cboe Global Indices, LLC (2023), we required the best ask and bid prices for each strike price per maturity. As the data comprised daily traded messages, we created a historical representation of each instrument's orderbook, simulating its state at any point during the day. The construction of the orderbook was done based on the specifications outlined in Appendix A of *Genium INET®ITCH Protocol Specification* (2017) for GITS and Appendix A of *ITCH Specification for Nasdaq Derivatives Markets: Nordic Equity Derivatives* (2022) for NITCH.

The general way of constructing the orderbooks began by saving all instruments traded for the day implied by the derivative directory message. We then formed separate lists for ask orders and bid orders for each instrument, sorting the list by the rank of the orders. As we iterate further through the messages, we built up the orderbooks. All incoming orders includes a rank. Rank 1 will be the best bid/ask order and the order should be sorted first in the respective list. When new orders are added, all other orders in the bid/ask list with the same or worse rank have their rank increased (i.e., they move down in the order of attractiveness). When orders are deleted, all orders placed after the deleted order in the list has their rank decreased (up in the order of attractiveness).

At the beginning of each hour, from 9:00 AM to 4:00 PM, we captured the current best bid and ask prices for each instrument and stored them in a CSV file dated for that day. This method allowed us to develop indices with a frequency of up to one hour. The Python scripts used to construct the orderbook for NITCH messages are located in Appendix B.1. The script for creating the orderbook for GITS messages is not provided, as it closely resembles the one for NITCH but is specifically adapted for GITS messages.

Although there is an option to use hourly data, we have chosen to maintain a daily frequency. Given the substantial volume of data spanning almost six years, high-frequency data could potentially obscure the genuine signal with noise and anomalies. Lyócsa et al. (2021) observed that for forecasts extending up to a month, the precision is statistically comparable whether employing intraday frequencies or daily data as the basis for the forecast. Considering our forecast horizon of 11 months, we have decided to exclude hourly frequencies.

3.2 Eikon data

In addition to the option data, the underlying indices closing prices (levels), volume, and logarithmic returns are gathered via Refinitiv Eikon's Python API (Refinitiv, 2023). The same goes for the market caps, though the procedure differs somewhat. The daily closing prices and volume are fetched via the native functions of the API, however for the market cap we needed to create a custom function, nesting two different data fetching functions from Eikon. To summarize, the total market cap at the rebalancing dates of each underlying, is calculated by taking the constituent companies of the indices at each date, taking the sum to get the total market cap of the index, then we weight them across one another for the total SCANDI-VIX cap. Occasionally there are holes in the desired data due to limitations on our access or unavailability of the data. This was filled by approximation. The script for fetching the data is given in Appendix B.4.

In detail, we extract the Refinitiv Identification Codes (RICs); .OBX, .OMXC25CAP and .OMXS30, where the OBX is used to calculate our approximation of the OMXO20. This a measure taken due to restrictions on our data subscription. When constructing the weights, we make sure to convert the underlying constituent market capitalisation into the common currency Euro. The SEK, NOK, and DKK exchange rates to Euros are also fetched via the same script as is used to gather market capitalisations for the constituent companies.

Obtaining the levels and trading volume enables us to compute our 30-day logarithmic return. We use 30-day log differences as they are equivalent to logarithmic returns, while avoiding the potentiality of trend introduction, also known as non-stationarity. For the benchmark indices, S&P500 and VIX, we simply fetched them via the Eikon API and ensured the time horizon matches that of our own implied volatility indices.

3.3 Risk-free rates

The construction of an implied volatility index necessitates a risk-free rate. For the CBOE VIX, this is achieved by employing the U.S. Treasury yield curve rates, to which a Cubic Spline interpolation method is applied for estimating yields between the maturities. In contrast, our European references, as indicated in Bugge et al. (2016), Hansen (1999), Öström (2015), and Skiadopoulos (2004), utilize interbank offered rates. Following this precedent, our methodology incorporates the closing interbank rates as the risk-free rate proxy.

For NORVIX we were provided with NIBOR from Norske Finansielle Referanser AS. For the SWEVIX we were provided with the STIBOR from Swedish Financial Benchmark Facility. Lastly for the DANVIX we were provided with the CIBOR from Danish Financial Benchmarks Facility. These rates were delivered with a daily frequency and spanned the same period as the options data, covering from January 2018 to the end of November 2023. The rates acquired have maturity for 1-week, 1-month, 2-months, 3-months, and 6-months for NIBOR and STIBOR, while CIBOR got 1-week, 1-month, 3-month, 6-month and 1-year rates.

4 The Scandinavian Implied Volatility Indices

4.1 Building the implied volatility indices

When developing implied volatility indices, we create two versions for the same underlying, the model-based and the model-free. Both methodologies consist of four stages to calculate an index value. The first two stages are quite similar, and we'll begin by discussing these shared steps. From the third stage onwards, the methodologies diverge, and we will discuss these stages separately.

Initially, the expiration date, option type (call or put) and strike prices are identified from the instrument name, facilitating the organisation of the option specifications. The first step of both methodologies is to determine the near-term and next-term maturity dates. In the model-based approach, all options maturing within eight calendar days are excluded. Of the remaining options, the closest maturity is designated as the near-term and the next closest designated as the next-term maturity. For the model-free methodology, we pick the maturities immediately before and after the 30-day constant maturity date. This will be noted as the near-term and next-term maturity dates, respectively.

The next stage involves determining the risk-free interest rates for both term maturities. As these maturities often don't align precisely with the maturity dates of Interbank Rates, we employ Natural Cubic Spline interpolation to fill in the gaps between days. To do this, we utilise the CubicSpline module from SciPy (2023) and make sure that the interpolation follows the boundary condition outlined in the white paper (Cboe Global Indices, LLC, 2023). This is then transformed from an annual nominal rate to a continuously compounded rate by:

$$r = \ln(1 + R) \tag{4.1}$$

At the third stage, the paths of the two methodologies diverge. For the model-based method, this step involves calculating the implied volatilities for options at two near-the-money strikes. Since there is no closed-form solution for implied volatility, we use iterative search to find the correct volatility through the Black-Scholes pricing formula. These calculated implied volatilities are subsequently interpolated to identify the at-the-money implied volatility for both the near- and next-term maturities. The fourth step involves interpolating the implied volatility across these maturities. Ultimately, we adjust the volatility value by multiplying it by 100 to present it as a percentage. The Python code for implementing the model-based methodology is provided in Appendix B.2.

The model-free methodology involves a more complex process. In its third step, we calculate the near- and next-term variances, which include some sub-steps. Initially, we identify an at-the-money strike, defined as the strike price with the smallest difference between the call and put price. Then we calculate the option implied forward rate F. By using F we find the strike price that is immediately below F, which will be our K_0.

With K_0 established, the options are filtered to only contain OTM options. This means selecting puts where the strike price is less than K_0 and calls where the strike price is more than K_0 . Furthermore, all bids must be strictly positive for inclusion of a strike price. To eliminate outliers, the process excludes all strikes where there are non-strictly positive values for two consecutive instances. One in-the-money strike price, the K_0, is included. Here, the price of the option will be calculated as the average of the put and call prices for this strike. Using these filtered options, we then calculate the variance for each term. The fourth step is to determine the variance by using linear interpolation between the two term dates, thereby establishing the 30-day variance. The final step is to take the square root of this variance and multiply it by 100 to convert it into a percentage.

The Python code for implementing the model-free methodology is provided in Appendix B.3. The calculation has been successfully tested by using the example data provided in the "Sample Calculation for the VIX index" chapter in the white paper (Cboe Global Indices, LLC, 2023).

4.2 Output of the implied volatility indices

The option observations used for calculating implied volatility indices occasionally presented issues. In some situations, options were only available for one maturity. Other instruments were not considered because their bid price was higher than the ask price. In the case of model-based indices, there were times when there were too few options to straddle the spot price. For the model-free indices, some options lacked both a traded call and put at the same strike price, which is crucial for determining the minimum difference. There were also cases where no strike prices traded below the calculated forward price, preventing us from getting an at-the-money strike.

Consequently, there were days when it was not feasible to calculate a value. Between January 3, 2018, and November 10, 2023, the data recorded was as follows: For the model-based methodology, NORVIX-MB had 1370 daily observations, DANVIX-MB had 1374, and SWEVIX-MB had 1456. In contrast, the model-free methodology recorded 1419 daily observations for NORVIX, 1422 for DANVIX, and 1466 for SWEVIX. Given that there were around 1480 trading days during this period, the number of days when observations are missing is relatively small. Evidently, there are more observations for the model-free indices.

Figure 4.1 presents the time series for the implied volatility indices, where a clear correlation is immediately noticeable between the model-based and model-free indices for each underlying asset. The correlation coefficient for DANVIX and DANVIX-MB stands at 0.880, while it's 0.960 for NORVIX and NORVIX-MB, and 0.941 for SWEVIX and SWEVIX-MB. Upon closer examination, it's apparent that the model-based indices generally show higher implied volatility compared to their model-free counterparts, a tendency likely linked to the date conversion method previously discussed. This trend is more evident when examining the mean and median values in Table 4.1. Furthermore, the data on skewness and kurtosis in Table 4.1 indicate that, except for SWEVIX, the model-based approach usually results in more pronounced outliers than the model-free approach. A visual inspection of the SWEVIX-MB also suggests the presence of some atypical outliers.

Gonzalez-Perez and Novales (2011)) argue that the sole use of out-of-the-money (OTM) options in model-based indices more effectively reflects investor fear by capturing the sentiment of hedgers (OTM put options) and market optimism (OTM call options). Considering this, alongside the fewer observations and common occurrence of extreme outliers in the model-based method, we have resolved to exclusively concentrate on the model-free approach henceforth. This decision is also logical for forecasting purposes, where the focus is on realised volatility, and as mentioned, the model-based approach doesn't align well with this given a 365-day calendar year.

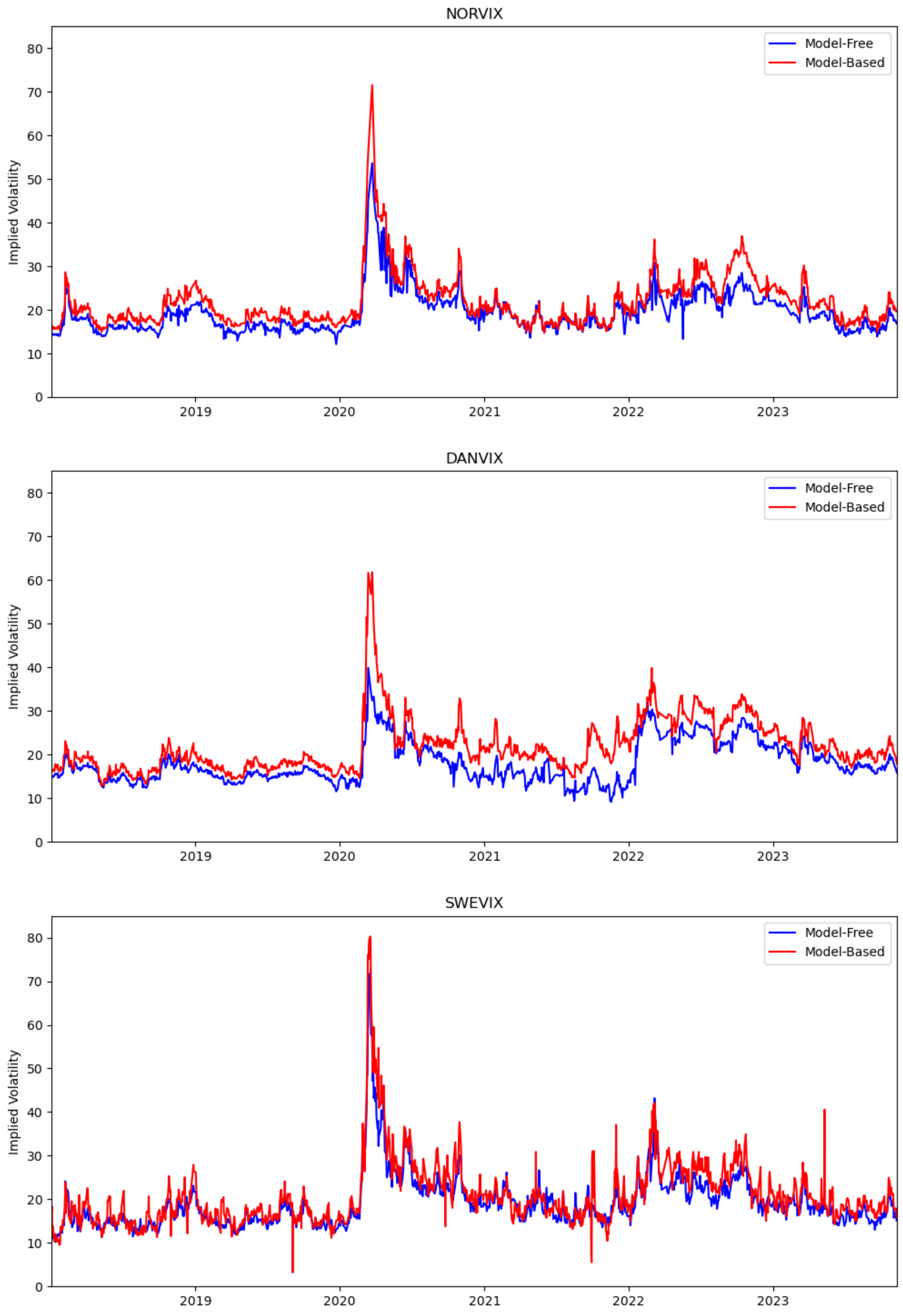

Figure 4.1: The Three Scandinavian Volatility Indices
Using both model-based and model-free methodologies.

Table 4.1: Descriptive Statistics for the Scandinavian Implied Volatility Indices

	Mean	Median	Std.	Skew	Ex. Kurtosis
NORVIX					
Model-Free	19.32	18.20	4.48	2.04	7.29
Model-Based	21.72	19.97	5.62	2.28	9.51
DANVIX					
Model-Free	17.91	16.63	4.54	0.91	0.34
Model-Based	21.66	20.27	5.68	1.76	6.19
SWEVIX					
Model-Free	19.32	18.00	6.33	3.33	18.52
Model-Based	20.89	19.10	7.56	3.03	15.87

Looking at Figure 4.1, the indices exhibit a typical pattern of implied volatility indices, with occasional spikes reflecting periods of market stress or uncertainty. This is especially evident during March 2020 with the start of the COVID-19 pandemic. The mean values of these indices are relatively close, with NORVIX and SWEVIX having a slightly higher mean at 19.32, compared to DANVIX's average of 17.91. This suggests a slightly higher long-term volatility level for Norway and Sweden's markets as captured by these indices.

The standard deviation for DANVIX and NORVIX is comparatively lower, suggesting more stable volatility levels, whereas SWEVIX exhibits a higher standard deviation, indicating more significant fluctuations in volatility. This pattern is clearly visible in Figure 4.1, where the volatility spike during the COVID-19 period was noticeably more pronounced for SWEVIX than for NORVIX and DANVIX.

4.3 The composite Scandinavian implied volatility index: SCANDI-VIX

To measure the implied volatility in Scandinavia as a whole, we need an underlying asset with traded options. As the OMXN40 has no options, we decided to create a weighted index, following the semi-annual rebalancing of the three Scandinavian OMX indices. By using the same weighting scheme, we can approximate a Scandinavian Composite Index (SCI) to base our implied volatility index (IV) on.

We first calculate the total Scandinavian market cap:

$$IDX_{\text{mktcap}}^{\text{Composite}} = \sum_{n=1}^{N} \sum_{m=1}^{M} c_{n,m} \tag{4.2}$$

Where N is the set of indices we wish to incorporate, M is the set of constituent companies and $c_{n,m}$ a constituent company's market capitalisation. We then determine the weights of the country-specific indices based on their market capitalisation:

$$\omega_n = \frac{IDX^n}{IDX_{\text{mktcap}}^{\text{Composite}}} \tag{4.3}$$

Where ω_n is the weight parameter of the n-th index. Table 4.2 show the weights used in our composite index. Finally, we calculate the SCI and SCANDI-VIX:

$$\text{SCI} = \sum_{n=1}^{N} \omega_n IDX_n, \qquad \text{SCANDI-VIX} = \sum_{n=1}^{N} \omega_n IV_n \tag{4.4}$$

Table 4.2: Scandinavian Composite Weights Over Time

Date	ω_{OMXO20}	ω_{OMXC25}	ω_{OMXS30}
Jan 2018	0.183	0.329	0.488
Jun 2018	0.207	0.274	0.519
Jan 2019	0.210	0.278	0.512
Jun 2019	0.192	0.278	0.530
Jan 2020	0.165	0.286	0.545
Jun 2020	0.145	0.308	0.547
Jan 2021	0.147	0.334	0.520
Jun 2021	0.156	0.330	0.514
Jan 2022	0.148	0.342	0.510
Jun 2022	0.178	0.344	0.478
Jan 2023	0.155	0.359	0.486
Jun 2023	0.138	0.350	0.512
Mean	0.168	0.318	0.514

Given that all OMX indices undergo semi-annual rebalancing (Nasdaq, 2023), we have chosen to align our rebalancing schedule with that of the indices. This weighting approach aims to mirror the composition of the underlying indices. Rebalancing the SCANDI-VIX more frequently could potentially offer a more precise reflection of dynamic market

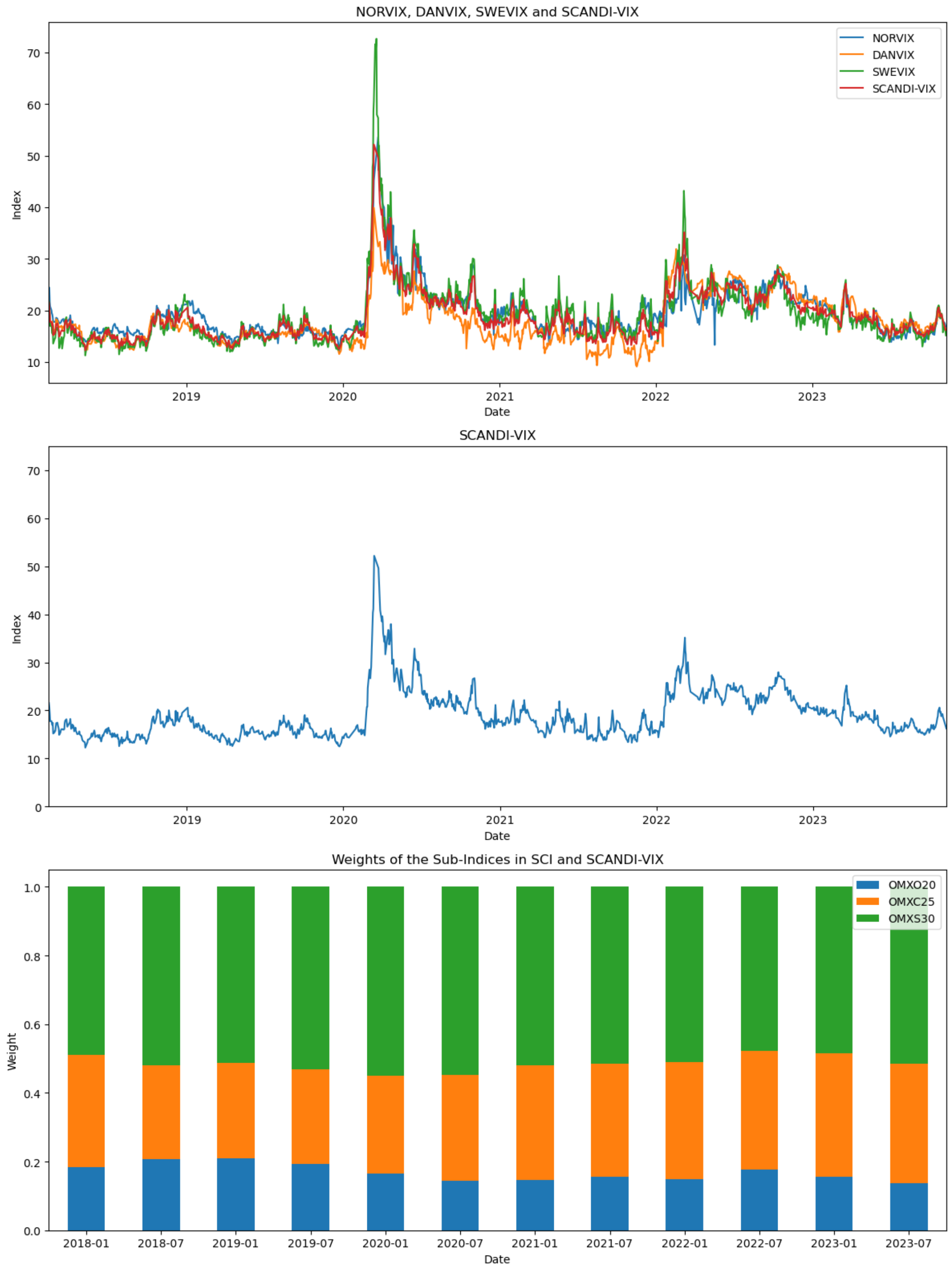

Figure 4.2: The Scandinavian Implied Volatility Index: SCANDI-VIX.

conditions and short-term volatility trends. A potential downside of higher frequency is the introduction of noise into the SCANDI-VIX. Moreover, increasing the frequency of rebalancing is likely to have a minimal effect on long-term forecasting horizons. Therefore, we view semi-annual rebalancing more as a practicality than a strict law.

Figure 4.2 displays the implied volatility indices for the Nordic countries, with each country's index stacked in the first pane and SCANDIVIX showcased separately in the middle. To facilitate a direct comparison with the CBOE VIX, Table 4.3 includes VIX statistics for the same time-frame.

Table 4.3: Summary Statistics Table

	Mean(%)	Std(%)	Min(%)	Max(%)	Skew	Ex. Kurtosis
Panel 1: Implied Volatility Index Log-Returns						
NORVIX	0.13	20.48	-59.04	118.28	1.17	4.67
DANVIX	0.19	21.85	-51.61	104.40	1.48	3.89
SWEVIX	0.52	25.65	-105.16	149.42	1.18	6.07
SCANDI-VIX	0.37	21.04	-68.38	126.39	1.49	5.37
VIX	0.48	31.43	-90.15	156.09	1.31	4.62
Panel 2: Underlying Index Log-Returns						
OMXO20	1.18	5.06	-32.71	19.22	-1.33	6.98
OMXC25	0.97	6.03	-29.72	22.44	-0.54	1.75
OMXS30	0.62	6.14	-36.31	18.60	-1.40	5.28
OMXN40	0.89	5.16	-35.22	21.75	-1.21	5.21
SCI	0.85	5.71	-30.35	20.64	-0.89	2.39
S&P500	0.92	6.27	-39.70	24.85	-1.61	6.57
Panel 3: Implied Volatility Index Levels						
NORVIX	19.32	4.48	12.10	53.65	2.04	7.29
DANVIX	17.92	4.54	9.12	39.91	0.91	0.34
SWEVIX	19.31	6.33	11.27	72.67	3.33	18.52
SCANDI-VIX	18.76	4.67	12.26	52.17	1.84	5.81
VIX	20.86	7.97	10.85	82.69	2.59	12.02
Panel 4: Underlying Index Levels						
OMXO20	926.53	153.35	591.03	1213.18	0.17	-1.46
OMXC25	1471.82	305.94	980.75	2020.69	-0.02	-1.49
OMXS30	1899.73	295.53	1292.27	2456.17	0.08	-1.41
OMXN40	1886.93	326.78	1261.57	2462.51	0.14	-1.50
SCI	1599.72	272.85	1090.38	2080.84	0.07	-1.49
S&P500	3581.51	678.14	2237.40	4796.56	0.01	-1.48

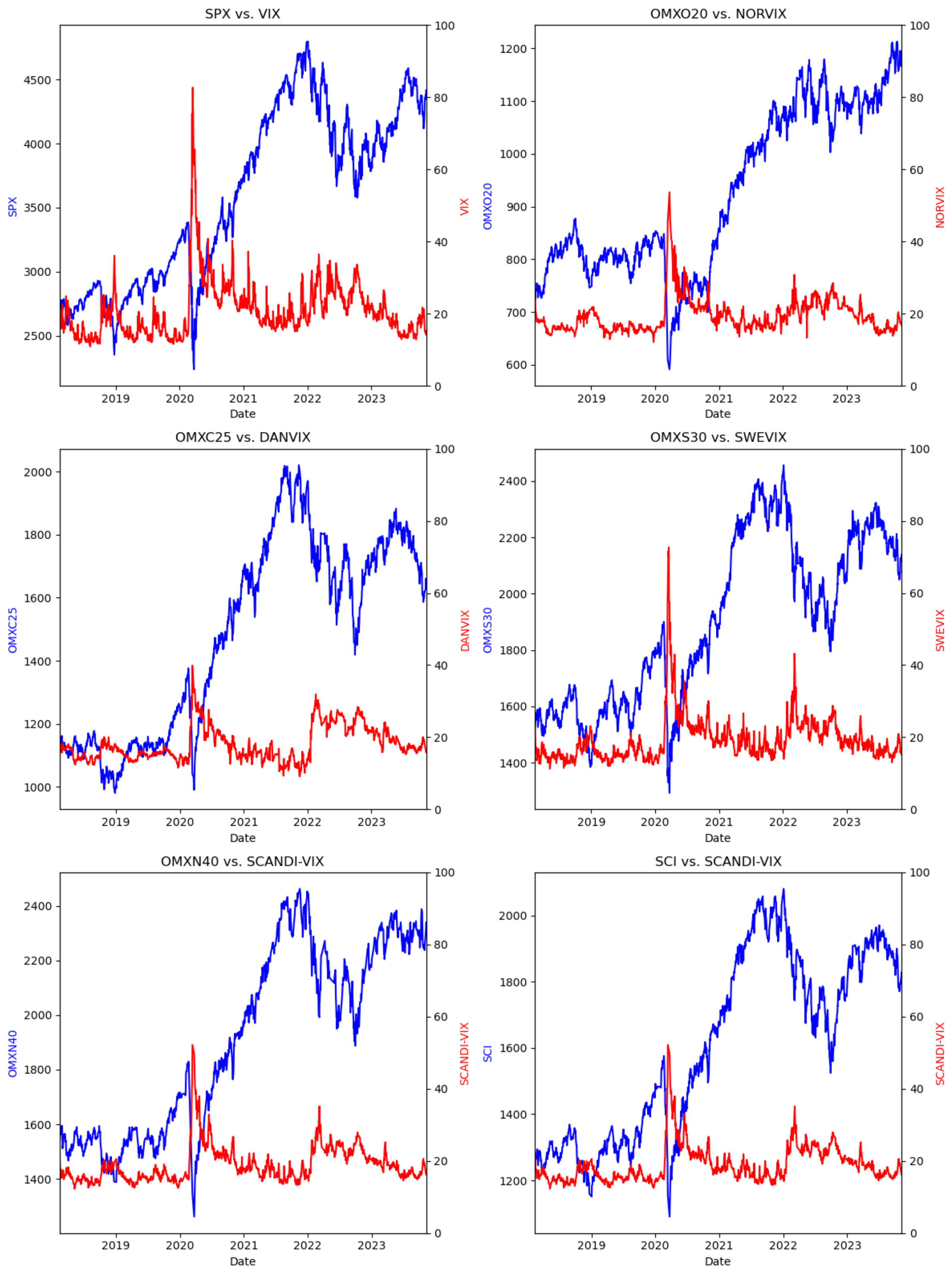

Figure 4.3: Implied Volatility Indices vs. Underlying Index

In Figure 4.3, the implied volatility indices are overlaid on their respective underlying assets, clearly showing each Nordic index's reaction to the COVID-19 pandemic. Table 5.1 offers a detailed look at SCANDI-VIX's performance relative to NORVIX, DANVIX, and SWEVIX. SCANDI-VIX's mean return mirrors the average volatilities of its components, aligning with its weighted nature. Its standard deviation is similar to NORVIX and DANVIX, indicating it reflects their volatility but is less volatile than SWEVIX, the most volatile component. SCANDI-VIX also exhibits moderate skewness and kurtosis compared to SWEVIX, suggesting a slightly more right-skewed and peakier distribution than NORVIX and DANVIX, yet not as extreme as SWEVIX. Overall, SCANDI-VIX offers a diversified view of the Nordic volatility landscape, presenting a balanced yet less volatile option for investors and risk managers.

Comparing SCANDI-VIX and its constituents with the VIX reveals interesting points. The average implied volatility levels suggest that in the Nordic markets, the SWEVIX mirrors the US market's volatility the most. Potentially due to Sweden's market liquidity. Figure 4.4 shows the 252-day rolling correlation between the underlying indices and their returns. Throughout the period, DANVIX and NORVIX consistently display weaker correlations, with SWEVIX slightly higher, yet still significantly lower than the VIX. Intriguingly, the composite SCANDI-VIX shows a higher overall correlation with the underlying indices than its individual components. Remarkably, the correlation over the entire period is higher between SCANDI-VIX and OMXN40 (-0.795) than between SCANDIVIX and SCI (-0.765), approaching the correlation level of -0.798 seen between the S&P500 and VIX.

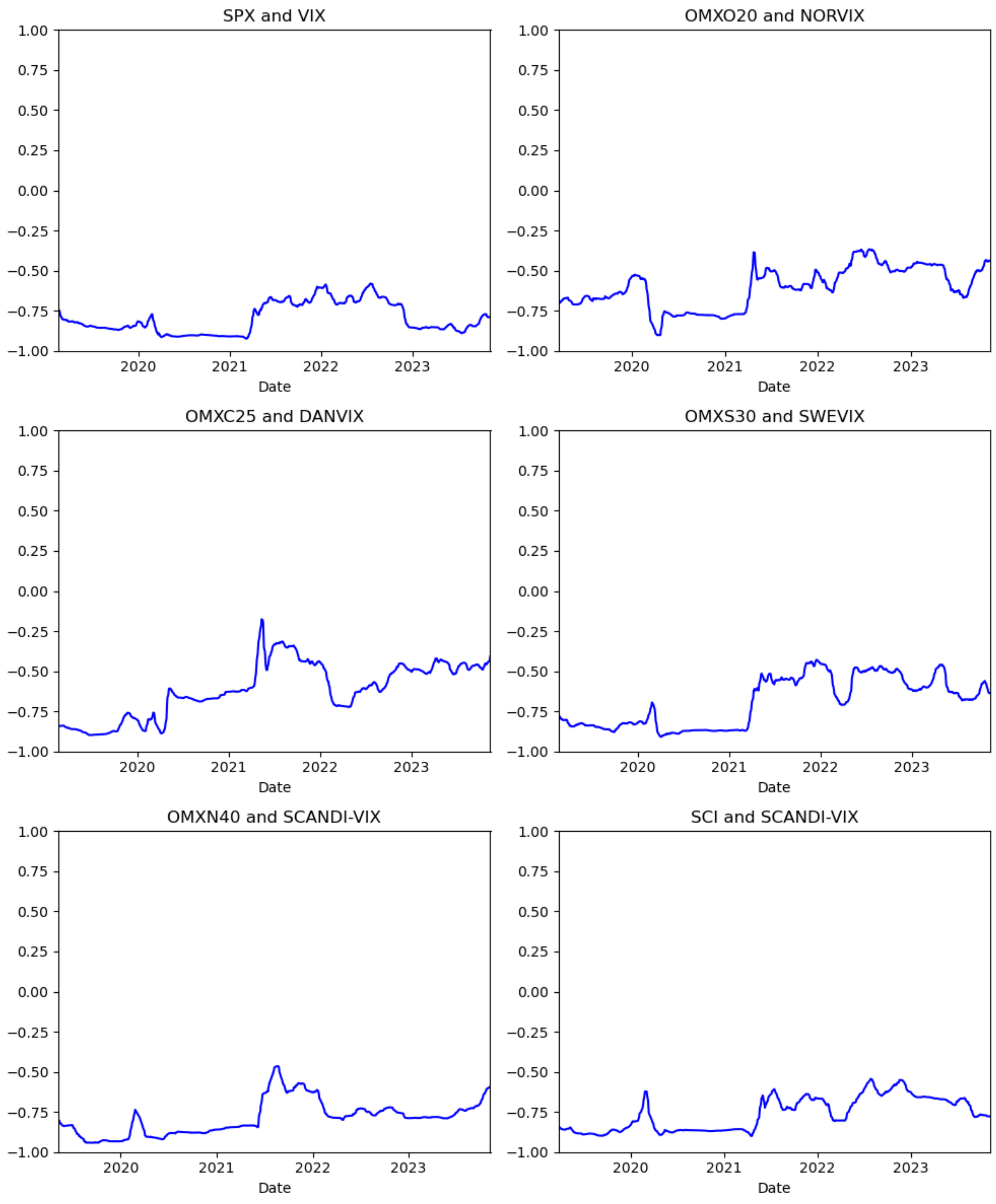

Figure 4.4: 252-Day Rolling Return Correlation

5 Contemporaneous Return Relationship and Return Asymmetry

With the methods from empirical literature on the relationship between implied volatility indices and underlying index returns, we can test the first two of our hypotheses.
H1: There exists a negative contemporaneous relationship between the Scandinavian implied volatility indices returns and underlying indices returns, thus they will act as a fear gauge.
H2: The return-volatility relationship is asymmetric, meaning the Scandinavian implied volatility indices react differently to negative and positive returns. We employ the following empirical model to test the first hypothesis.

$$IV_i^R = \alpha_i + \beta_i IDX_{i,t}^R + e_{i,t} \tag{5.1}$$

Where, IV_i^R is the daily logarithmic return of the implied volatility index i. And $IDX_{i,t}^R$ is the daily logarithmic return of the underlying index$i$. The second hypothesis is tested with the following models.

$$IV_{i,t}^R = \alpha_i + \beta_i IDX_{i,t}^+ + \gamma_i IDX_{i,t}^- + u_{i,t} \tag{5.2}$$

$$Q(IV_{i,t}^R) = \alpha_i^q + \beta_i^q IDX_{i,t}^+ + \gamma_i^q IDX_{i,t}^- + u_{i,t} \tag{5.3}$$

Where, $IV_{i,t}^R$ is the logarithmic returns on the implied volatility index. $IDX_{i,t}^+ = \max(IDX_{i,t}^R, 0)$ is the positive logarithmic return series. $IDX_{i,t}^- = \min(IDX_{i,t}^R, 0)$ is the negative logarithmic return series. The intercept α and coefficients β and γ are assumed to be independent. The third regression is over each quantile $q \in Q$, for $Q = (0.1, 0.2, \ldots, 1)$. Without the quantile element of the regression, it is assumed that the effect of positive and negative returns is static (5.2). In Fassas and Siriopoulos (2020) they report that past the 0.5 quantile, the regular OLS under-estimates the effect of negative returns. While overestimating the effect of positive underlying index returns on implied volatility index returns. They note that for all the equity indices in their review, the absolute values of

$IDX^{-}_{i,t}$ increases in the upper quantile. Whereas $IDX^{+}_{i,t}$ decreases in absolute values when moving from a lower quantile to a higher quantile.

5.1 Negative return relationship

The results of Eq. (5.1), shown in Table 5.1, indicate a statistically significant negative relationship between the returns on our implied volatility indices and the associated underlying indices. This is indicated by the negative results in the $IDX^{R}_{i,t}$ column. The adjusted R^2 shows to what extent the daily movement in the implied volatility index return can be explained by the underlying index return. We combine the Shapiro-Wilk (Shapiro & Wilk, 1965) and Durbin-Watson (Durbin & Watson, 1971) tests to the regression model results. Shapiro-Wilk tests have the null hypothesis that a sample $x_1, ..., x_n$ comes from a normally distributed population. Mota (2012) notes that though it is widely assumed log-returns are normally distributed, in majority of cases this is rejected. The Durbin-Watson test detects autocorrelations, with $d = 2$ meaning no autocorrelation. The slight deviation from 2 in the test, suggests minor negative auto-correlation. This could be an indication of serial dependence, where past returns impact future returns. We have therefore confirmed our hypothesis of negatively contemporaneous related returns between the implied volatility indices and the respective implied volatility index. The implication of non-normality and autocorrelation in the data will however have implications for our forecasting model choice.

Table 5.1: Emprical results Eq.(5.1), showing significant evidence for a negative contemporaneous return relationship between the volatility indices and their underlying indices.

	Intercept	$IDX^{R}_{i,t}$	Adj. R-sq	F-Stat	Durbin-Watson
NORVIX	0.0006	-2.49	0.21	363.17***	2.61***
DANVOX	0.0003	-1.53	0.10	153.49***	2.41***
SWEVIX	0.0005	-2.97	0.28	541.06***	2.38***
ScVIX/N40	0.0008	-2.70	0.44	813.14***	2.36***
ScVIX/SCI	0.0006	-2.64	0.36	694.79***	2.44***
VIX	0.0012	-4.28	0.52	1578.88***	2.25***

*Where *** denotes* $p < 0.001$, **** $p < 0.01$, *and ** $p < 0.05$

5.2 Asymmetric return effect

The result of Eq.(5.2) is given in Table 5.2 as the OLS regression. It reveals an asymmetric return relationship across the various implied volatility indices. The negative return coefficients R^- are uniformly negative and significantly different from zero across all indices. This indicate that negative returns on the underlying indices are associated with a significant increase in the corresponding implied volatility indices, under the assumption that the effect of the return series are static. As shown by Fassas and Siriopoulos (2020), this assumption does not necessarily hold. We therefore utilise the alternative quantile regression framework Eq.(5.3), given in Table 5.2.

The findings from the quantile regression, aligns with the similar findings in Fassas and Siriopoulos (2020). There is a clear difference in the magnitude of coefficients for positive and negative return series. The table suggests that negative underlying market movements illicit a stronger response from investors, as indicated by the larger coefficients for negative returns. Showcasing investor aversion to loss, which may drive investors to react more sharply to negative news. The non-monotonic curves observed in the positive return series for the SCANDI-VIX relationships, hints at more nuanced interaction between the positive return series and the volatility in the markets. It could be due to diversification, as the SCANDI-VIX is a linear combination of the other Scandinavian implied volatility indices. There might also be idiosyncratic risks in the underlying Scandinavian indices or cross-market dynamics affecting the results. It could also be interpreted as diminishing sensitivity to positive news as underlying index returns go up. Implying good news has less incremental impact during bull markets. The monotonic increase in the negative return series, on the other hand, is consistent with literature. It suggests that investors' reactions and risk perceptions differ fundamentally in the face of gains versus losses. The Eq.(5.2) results does over-estimate the effect of negative returns in the higher quantiles, when compared to the results of Eq.(5.3). It also under-estimate the effect of positive returns in the higher quantiles, which the opposite being true in the lower quantiles. The same under-estimation occurs for negative returns in the lower quantiles, as shown in Figure 5.1.

Table 5.2: Return-Volatility Asymmetric Relationship

Quantile	NORVIX	DANVIX	SWEVIX	ScVIX/N40	ScVIX/SCI	VIX
Panel 1: Positive Return Coefficient (R_t^+)						
0.1	-4.113***	-2.732***	-3.166***	-2.120***	-2.260***	-5.896***
0.2	-3.289***	-2.628***	-3.502***	-2.376***	-2.359***	-5.234***
0.3	-2.755***	-2.191***	-3.160***	-2.413***	-2.302***	-4.477***
0.4	-2.244***	-1.874***	-3.176***	-2.484***	-1.984***	-3.912***
0.5	-1.952***	-1.382***	-2.792***	-2.336***	-2.027***	-3.657***
0.6	-1.449***	-0.906***	-2.716***	-2.207***	-1.955***	-2.979***
0.7	-1.389***	-0.257	-2.259***	-1.992***	-1.594***	-3.012***
0.8	-1.117***	0.210	-1.698***	-1.437***	-1.348***	-2.599***
0.9	-0.444	0.677**	-1.186***	-1.198***	-0.771**	-1.962***
OLS	-2.166***	-0.997***	-2.614***	-2.072***	-1.778***	-3.332***
Panel 2: Negative Return Coefficient (R_t^-)						
0.1	-1.078***	0.603*	-2.257***	-1.936***	-1.810***	-3.168***
0.2	-1.346***	-0.161	-2.307***	-2.356***	-2.076***	-3.709***
0.3	-1.746***	-1.010***	-2.678***	-2.816***	-2.385***	-4.209***
0.4	-2.003***	-1.535***	-2.846***	-2.915***	-2.942***	-4.916***
0.5	-2.700***	-2.087***	-3.474***	-3.163***	-3.017***	-5.890***
0.6	-2.953***	-2.369***	-4.010***	-3.249***	-3.269***	-6.475***
0.7	-3.244***	-2.721***	-4.284***	-3.529***	-3.741***	-7.113***
0.8	-3.583***	-3.130***	-4.869***	-4.466***	-3.700***	-7.977***
0.9	-4.505***	-3.874***	-5.116***	-4.567***	-4.287***	-9.515***
OLS	-2.765***	-2.028***	-3.271***	-3.298***	-3.161***	-5.077***

*Where *** denotes* $p < 0.01$, **** $p < 0.05$, *and ** $p < 0.1$

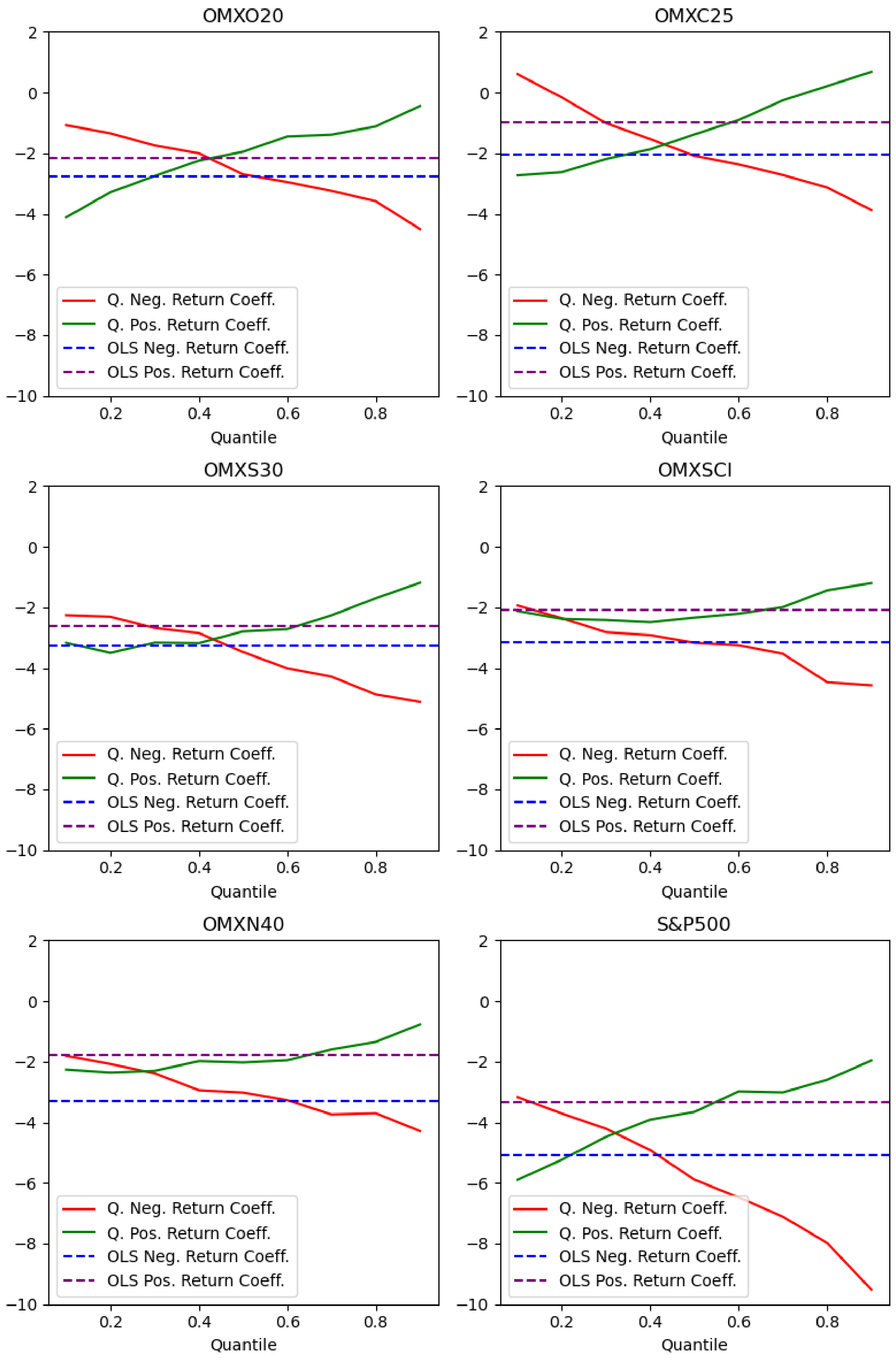

Figure 5.1: Asymmetric Return Effect

6 Stationarity and Forecasts

Drawing upon the existing body of literature on forecasting future realised volatility, we attempt to evaluate the following hypotheses.

H3: Using implied volatility for forecasting surpasses GARCH models, especially when it's integrated into the machine learning algorithm XGBoost.

H4: The composite proxy of the three Scandinavian implied volatility indices will absorb information well enough for it to serve as a useful predictive and descriptive tool of the OMXN40.

To assess H3, our approach involves comparing the forecasting capabilities of lagged implied volatility indices on their own. Additionally, we will predict realised volatility by using two methods. GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model and an XGBoost (Extreme Gradient Boosting) machine learning algorithm, thereby determining the efficacy of utilising implied volatility in these models.

6.1 Cross-validation of stationarity

To validate the reliability of our out-of-sample forecasts of realized volatility, we assess whether the input log-returns exhibit stationarity. Stationarity refers to a process whose statistical properties, like mean and variance, are constant over time. This is crucial, as non-stationary data can lead to misleading results in models like the GARCH model, which assumes stationary inputs. XGBoost is meant to better handle non-stationarity, though significant levels of non-stationarity might still affect the forecasts negatively. We apply the Augmented Dickey-Fuller (ADF) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. The ADF test, focusing on a unit root presence, has a null hypothesis of non-stationarity. While the KPSS test, examines trend stationarity, and assumes stationarity under the null hypothesis. Their complementary nature provides a robust cross-validation of stationarity in our log-returns series.

Our results, as detailed in Table 6.1, show both affirming and concerning trends. The ADF test results for the returns of the implied volatility indices are as expected, rejecting the null hypothesis of a unit root, suggesting no presence of non-stationarity. The KPSS test compliments this, indicating stationarity in the IV returns with statistical significance.

This cross-validation suggests that using implied volatility log-returns series for XGBoost forecasting is appropriate. However, the results for the levels of the implied volatility indices remain inconclusive. Most of the underlying index returns show signs of non-stationarity, potentially undermining the reliability of our GARCH model's forecasts. This inconclusiveness, somewhat uncharacteristic compared to results in Bugge et al. (2016), might be attributed to the COVID-19 pandemic. Which has potentially introduced structural changes and mean-reversion trends post-volatility spikes. This could explain the mixed results we observe compared to standard behaviors, as indicated in studies like those by Bugge et al.

Table 6.1: ADF-KPSS Stationarity Cross-validation

	Levels		Returns	
	ADF	KPSS	ADF	KPSS
NORVIX	-4.54**	2.41**	-6.88**	0.22
DANVIX	-3.51**	5.93**	-6.05**	0.17
SWEVIX	-5.02**	2.69**	-7.04**	0.22
SCANDI-VIX	-4.24**	3.79**	-6.04**	0.22
VIX	-4.49**	2.53**	-6.76**	0.21
OMXO20	-0.92	27.98**	-6.1**	0.42*
OMXC25	-1.34	27.61**	-5.52**	1.04**
OMXS30	-1.53	26.74**	-5.39**	0.4*
OMXN40	-1.04	23.16**	-7.2**	0.63**
SCI	-1.39	25.19**	-5.41**	0.55**
S&P500	-1.18	29.86**	-5.54**	0.41*

Completed with lags equal to $\sqrt[4]{12\frac{n}{100}}$ *(Bugge et al., 2016).*
*Where ** p < 0.05, and * p < 0.1*

Considering these findings, the practical implications for our predictive modeling are significant. Non-stationarity in data can lead to poor predictive performance and invalid assumptions in GARCH models. Which will have negative consquences for our ability to verify *H3*, as the GARCH model is unable to perform due to the data. While this highlights a practical weakness of the GARCH family of models, it does not rule out the models as effective forecasting tools. Utilising the lagged implied volatility as a lower benchmark will thus be vital, to verify whether our GARCH model is uncalibrated due to the non-stationarity of the underlying index returns.

6.2 Forecasting models

In the subsequent forecasting tasks, we utilise log-returns of the underlying indices, monthly realised volatility, and our set of implied volatility indices. The formula to calculate realised volatility is as follows.

$$RV_{i,m} = \sqrt{\frac{365}{n_m} \sum_{t=1}^{N_m} (R_{i,t})^2} \tag{6.1}$$

Where $R_{i,t}$ is the log-returns for the underlying index at time t, m is the months in our dataset, and $RV_{i,m}$ is the realised volatilities for said months. For the forecasting exercises, we have split the dataset into an in-sample and out-of-sample subset. Where the in-sample is 69 monthly observations from 2018 to the end of 2022, and the out-of-sample set contain observations from Jan. 2023 to Nov. 2023. We utilise three approaches to the forecasting, the simple baseline of lagged implied volatility, a GARCH model and an XGBoost model. Where the GARCH model is selected via hyperparameter grid-searching, and XGBoost via hyperparameter Bayesian optimisation.

6.2.1 Lagged implied volatility

As the lower benchmark, we employ lagged implied volatility as a simple method to forecast realised volatility. This technique assumes that the previous period's implied volatility is the most accurate predictor of the current period's realised volatility. This method is quite straightforward and serves as a benchmark for the other forecasting models. The estimated future realised volatility as lagged implied volatility is given as follows.

$$\hat{RV}_t = IV_{t-1} \tag{6.2}$$

6.2.2 GARCH models

The Generalized Autoregressive Conditional Heteroskedasticity model is given as follows.

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 \tag{6.3}$$

Where σ_t^2 is the conditional variance of the historical index time-series,ϵ_{t-i} is the residual term, α_0 the constant term, α_i and β_j are the short-term ARCH component and long-term GARCH component respectively. Lastly, p is the order of the ARCH term and q the order of the GARCH term. The moving average is captured by $\sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2$ and the autoregressive component by $\sum_{j=1}^{q} \beta_j \sigma_{t-j}^2$. Our analysis involved implementing a variety of GARCH models through a grid-search algorithm, testing every possible combination of GARCH volatility model, residual distribution, and mean structure. We then save only the models with the smallest Akaike Information Criterion (Akaike, 1973). Which is given by: $\text{AIC} = 2k - 2\ln(L)$ where k is the number of parameters in the model and L is the maximised value of the likelihood function for the estimated model. In short, it describes the explanatory power of the model subject to a parameter count penalty. Upon applying the grid-search algorithm to the log-return series of all underlying indices, only the EGARCH (Exponential GARCH) model was selected for each index. These models consistently featured a constant mean structure and a skew-t distribution of residuals – a typical expectation for financial log-return time-series, which EGARCH is theorized to handle more adeptly than the standard GARCH model. Each index's model displayed unique values for the short-term p and long-term q variance orders. The EGARCH model, formulated as.

$$\ln(\sigma_t^2) = \alpha_0 + \sum_{i=1}^{p} \alpha_i g(\epsilon_{t-i}) + \sum_{j=1}^{q} \beta_j \ln(\sigma_{t-j}^2) \tag{6.4}$$

Where $g(\epsilon) = \theta\epsilon + \gamma(|\epsilon| - \mathbb{E}[|\epsilon|])$ allows for differential responses to positive and negative shocks. The terms θ and γ capture the asymmetric effects of these shocks. The finalized model configurations for each index are detailed in Table 6.2, indicating the specific p and q values for each underlying index.

Table 6.2: Orders of ARCH and GARCH Terms

	p	q
OMXO20	3	2
OMXC25	8	8
OMXS30	3	5
OMXN40	6	1
SCI	10	3

6.2.3 Extreme gradient boosting regressor

In their research, Teller et al. (2022) introduced the application of the XGBoost machine learning model for volatility forecasting, building on the foundational work of Chen and Guestrin (2016). XGBoost is grounded in the Classification and Regression Trees (CART) methodology, initially proposed by Breiman et al. (1984). CART divides input features into tree nodes of a decision tree, with predictive variable space division forming the branches. The predicted values of the algorithm are then used in its corresponding terminal nodes. The algorithm then assigns predicted values to these terminal nodes through a series of conditional statements, evaluating the input features against them. By defining the predicted value $\hat{y}_i$ of any input element x_i are defined in the regression tree model, dividing the predictive variable space into regions $R_1, \ldots, R_K$.

$$\hat{y}_i = \sum_{k=1}^{K} \hat{w}_k I(x_i \in R_k) \tag{6.5}$$

Where, K denotes the number of terminal nodes, $\hat{w}_k$ are estimated predictions or "leaf weights", and I is the indicator function. The estimated predictions are given as follows.

$$\hat{w}_k = \underset{w_k}{\operatorname{argmin}} \sum_{i=1}^{n} l(y_i, w_k) I(x_i \in R_k) \tag{6.6}$$

Where, l typically is a squared error loss function, $\hat{w}_k$ acts as the mean response in its region. CART incorporates recursive binary splitting alongside least squares, which checks every potential input and variable value combination before creating a new tree branch. Figure 6.1 depicts a complete tree structure, tracing from the terminal nodes at the top, down through the branches, until reaching a final leaf node at the bottom. Unlike non-boosting scenarios where a predefined model $F(x)$ relates inputs to outputs, tree

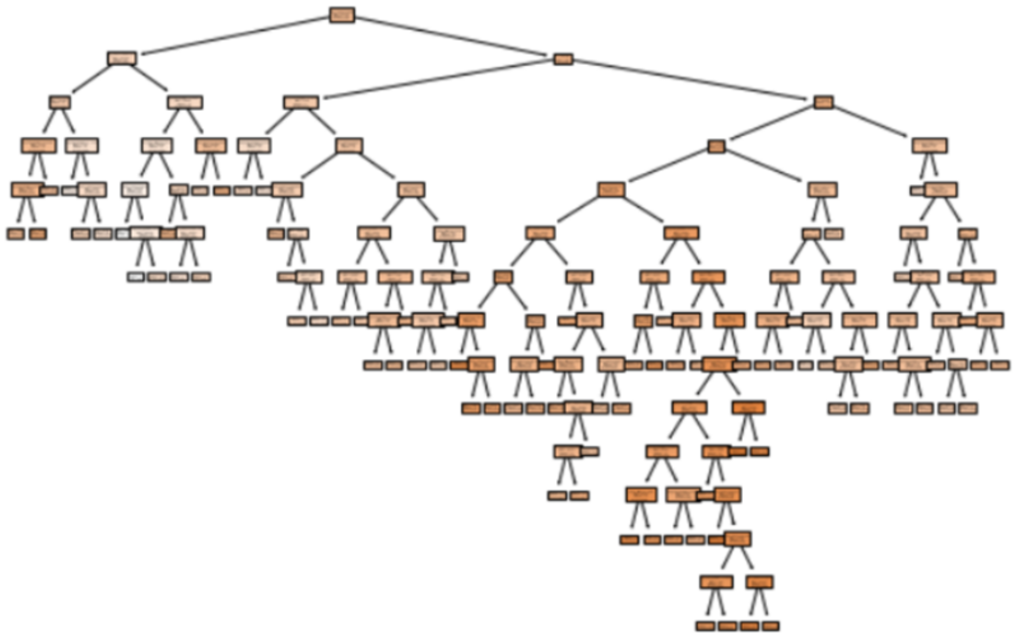

Figure 6.1: Example of a Decision Tree
Features Terminal Nodes, Branches and Leaf Weights. Input features are passed down the branches in a route determined by conditional "if" statements, until reaching a leaf weight (estimated prediction).

boosting considers every possible function F. The generalised optimisation problem can be formally presented as such:

$$F = \underset{F}{\operatorname{argmin}} \sum_{i=1}^{n} l(y_i, F(x)) \tag{6.7}$$

This process minimises an aggregated, twice-differentiable loss function l with $F(x_i)$. Boosting involves sequential updates to the model estimates from tree learners $f_m(x_i)$, also known as estimators, leading to the predicted value after M iterations:

$$\hat{y}_i = F_{m-1}(x_i) + \eta f_m(x_i) = \sum_{m=1}^{M} \eta f_m(x_i) \tag{6.8}$$

Where η is a set learning rate of the model. The tree learners are then estimated in a step-wise optimisation algorithm, which adds new tree learners to the decision tree ensemble based on their short-term reduction of errors. The following objective function is minimised in every m iterations:

$$O_m = \sum_{i=1}^{n} [l(y_i, F_{m-1}(x_i)) + \eta f_m(x_i)] \tag{6.9}$$

Chen and Guestrin (2016) introduces extreme gradient boosting by adding regularization to Eq.(6.9) to penalize predicted values and total terminal nodes K in an attempt to avoid overfitting, giving us the following equation:

$$\Omega(f_m) = \gamma K + \frac{1}{2}\lambda \sum_{k=1}^{K} w_k^2 + \alpha \sum_{k=1}^{K} |w_k| \tag{6.10}$$

Where, α is the L1 regularisation term that adds a penalty proportional to the absolute value of the feature weights, reducing the number of features with non-zero weights. λ is the L2 regularization term, shrinking the weights to avoid over-fitting. γ the parameter that controls the complexity of the tree by penalizing the number of terminal nodes. Giving us the final objective function:

$$O_m = \sum_{i=1}^{n} [l(y_i, F_{m-1}(x_i)) + \eta f_m(x_i)] + \Omega(f_m) \tag{6.11}$$

Furthermore, they minimise the objective function via a second-order Taylor expansion approximation, referred to as Newton Boosting. Extreme gradient boosting models also have further regularization techniques such as maximum depth restrictions, sub-sampling, minimum and max restriction on tree structures, depth of branches and iterations. Of the parameters, we utilise a Bayesian optimisation to find the number of tree learners $f_m(x_i)$ we add together in each boosting round, the learning rate η as a multiplier of $f_m(x_i)$ when updating $F_{m-1}(x_i)$ to $F_m(x)$, and a maximum depth of each tree $f_m(x)$. We also apply the optimisation to Ω, to affect the regularisation terms regard the minimum loss reduction required to make a split in a node γ and the penalty term with proportionality to the square of the magnitude of weights λ. Bayesian optimisation is a black-box function. However, it is intuitively a strategy to finding the minimum of a function that is expensive to evaluate. Utilising a defined search space and prior probabilities, typically represented by a probabilistic model, such as a Gaussian Process. Given an XGBoost model with hyperparameters θ, we aim to find the optimal set of hyperparameters θ^* that minimizes the function $f(\theta)$:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} f(\theta) \tag{6.12}$$

Where the procedure can be broken down into five steps, first you initialize the Gaussian Process with a prior distribution, secondly you select a set of hyperparameters θ to evaluate by optimising the acquisition function. Thirdly you evaluate the objective function $f(\theta)$ with the selected hyperparameters. Then, the Gaussian Process is updated with the new observations $(\theta, f(\theta))$. This process is repeated until a stopping criterion is met, be it maximum number of iterations or after a set amount of time as been spent. The result is a sequence of hyperparameter evaluations which has the aim of converging to the optimal hyperparameters θ^*. We utilize a similar search-space as Teller et al. (2022): Total count of $f_m(x)$ being $M \in \mathbb{Z} \cap [1, 100]$, regularisation parameters $\alpha, \gamma, \lambda \in \mathbb{R} \cap [0, 1]$, learning rate $\eta \in \mathbb{R} \cap [0, 1]$ and maximum depth of trees $Z \cap [1, 20]$. Despite the uncertainties of Bayesian optimization, it's preferred over grid-search due to the latter's impracticality in handling a large number of combinations and extended training times.

In our analysis, we focus on predicting the logarithmic returns of realised volatility to maintain consistency in our forecasting approach. XGBoost, in contrast to the EGARCH model, outputs predictions in the same data format as its input. This means when we input logarithmic returns of the underlying index into XGBoost, it returns predictions in the form of logarithmic returns. Log-returns of realised volatility is given as follows:

$$RV^R_{i,m} = \ln RV_{i,m} - \ln RV_{i,m} \tag{6.13}$$

Where RV^R_i is the log-returns of realised volatility. We also calculate lagged realised volatility returns, lagged implied volatility index returns and lagged underlying index returns. Three XGBoost regression models are then constructed:

$$\hat{RV}_{i,m} = F(IDX^R_{i,m}, IDX^R_{i,m-1}, RV^R_{i,m-1}) \tag{6.14}$$

$$\hat{RV}_{i,m} = F(IDX^R_{i,m}, IDX^R_{i,m-1}, IV^R_{i,m}, IV^R_{i,m-1}) \tag{6.15}$$

$$\hat{RV}_{i,m} = F(IDX^R_{i,m}, IDX^R_{i,m-1}, IV^R_{i,m}, IV^R_{i,m-1}, RV^R_{i,m-1}) \tag{6.16}$$

Where $IDX^R_{i,m}$ is the logarithmic return on the underlying index, and $IV^R_{i,m}$ is the log-returns on the corresponding implied volatility index. The purpose of the models is to validate *H3* and *H4*, as given at the beginning of the chapter. We will conclude this by measuring the errors of the levels, akin to what EGARCH outputs. To do so, we will calculate the levels by taking the last date levels for the realised volatility on the OMXs and SCI in the training set. Then adjusting for the predicted and actual realised volatility returns $\hat{RV}_{i,m}$ in the test set.

$$VL_m = VL_0 \times \prod_{i=1}^{m} e^{RV_i^R} \tag{6.17}$$

Where VL_0 is the initial realised volatility at the beginning of the testing period. We also ensure that both our EGARCH and XGBoost outputs are annualized and comparable.

6.3 Results

Our study's forecasting outcomes, detailed in Table 6.3, show noteworthy findings for the prediction of 2023's realised volatility. A key observation is the enhancement in forecast accuracy when including implied volatility in the forecasts. We experimented with several XGBoost configurations, selecting the best performers based on in-sample testing. The decision to use implied volatility alone or alongside lagged realised volatility in XGBoost showed different levels of effectiveness from market to market. Notably, the OMXO20 model, struggled significantly with only implied volatility. In regard to *H3*, our findings affirm that incorporating implied volatility, either in isolation or in combination with lagged realised volatility, in the XGBoost framework does indeed improve forecasting accuracy. However, assessing the performance of the EGARCH model is more challenging due to the inconclusiveness of non-stationarity in the underlying index time-series. Such non-stationarity adversely affect out-of-sample forecasts of GARCH family models. Despite these challenges, the selected EGARCH model represents the best option among various parameter combinations tested. We cannot therefore fully conclude that utilising implied volatility in XGBoost is strictly better than the EGARCH. However, it is apparent that the EGARCH is more susceptible to intricacies in the data, which XGBoost is designed to handle. Regarding *H4*, our analysis reveals that the SCANDI-VIX effectively predicts future realised volatility on the OMXN40 when utilized in XGBoost regression analysis with implied volatility. This is inline with the observed higher rate of absorption of information, as underpinned by the high correlation between OMXN40 and the SCANDI-VIX when compared to SCI. This finding conclusively supports our hypothesis, underscoring the significant utility of the SCANDI-VIX both as a descriptive and predictive instrument for the Nasdaq reference index.

Table 6.3: Forecast Errors

	OMXO20	OMXC25	OMXS30	OMXN40	SCI
Lagged Implied Volatility IV_{t-1}					
MAE	3.77	3.69	2.81	3.30	4.38
MSE	17.88	22.50	10.79	15.80	22.61
RMSE	4.23	4.74	3.28	3.98	4.76
EGARCH					
MAE	2.85	5.06	3.36	5.63	3.73
MSE	10.69	32.53	18.90	56.58	17.95
RMSE	3.27	5.70	4.35	7.52	4.24
XGBoost $F(IDX^R_{i,m}, IDX^R_{i,m-1}, RV^R_{i,m-1})$					
MAE	4.08	10.09	2.35	5.63	4.14
MSE	26.03	174.08	17.48	56.59	24.84
RMSE	5.10	13.19	4.18	7.52	3.28
XGBoost $F(IDX^R_{i,m}, IDX^R_{i,m-1}, IV^R_{i,m}, IV^R_{i,m-1})$					
MAE	13.69	1.76	1.92	2.94	3.86
MSE	244.33	4.84	8.73	13.01	18.60
RMSE	15.63	2.20	2.83	3.61	4.31
XGBoost $F(IDX^R_{i,m}, IDX^R_{i,m-1}, IV^R_{i,m}, IV^R_{i,m-1}, RV^R_{i,m-1})$					
MAE	1.90	2.70	2.32	3.15	2.81
MSE	9.38	11.57	12.31	15.15	10.77
RMSE	3.06	3.40	3.50	3.89	3.28

7 Limitations and further research questions

In the following section of our paper, we will briefly address some limitations of our thesis and offer recommendations for future research.

7.1 Limitations

For our analysis we have used a time period ranging from January, 2018 to November 2023. Even if this is a period of six years, the VIX index has been present for 30 years. Option trading on the OMXS30, OMXO20 and OMXC25 has also existed for much of that period. It could be that this limited time window negatively affects our results. However, since the window contains the recent COVID-19 stock market crash, we still believe it holds valuable information.

The research period can also adversely impact GARCH modelling. If the period is too short, it may not supply sufficient data points for the GARCH model to effectively capture the volatility dynamics of the series. Leading to parameter instability, due to reduced evidence of data stationarity. In addition, the hyperparameter space we chose to optimise the XGBoost model on, is a smaller subset of the total parameter choice. Meaning there might be possible configurations which provide better forecast quality.

Furthermore, we have continuously compared our results to other papers looking at different indices over different time periods, the financial time-series might experience structural changes or regime shifts that makes this a bad comparison. However, certain characteristics of implied volatility indices persists, despite the changing underlying dynamics. Leading us to believe, there might be interesting observations anyways.

7.2 Further research

There are multiple alternative hypotheses to test regarding our implied volatility indices, such as the effect of higher frequency data, potentially larger scale composite indices and more variations of machine learning algorithms. Whereas the SCANDI-VIX performed well in relation to our SCI index and the OMXN40, it would be interesting to see how larger regional or continental composite proxy indices would behave. It would also be interesting to see whether the framework could be applied to commodities. Such as a composite implied volatility index for energy, consisting of gas, oil, electricity, etc.

In terms of predictive modelling, our research was limited to evaluating the forecast quality of just three models. There may be other models better suited for this data, representing an avenue for further research. Although we decided against using intraday implied volatility indices for predicting future realised volatility, investigating the validity of this decision could yield interesting insights.

We observed that the movements in all the implied volatility indices we constructed tend to mirror each other, hinting at potential volatility spillover effects. It would be intriguing to investigate the extent to which volatility spikes or trends in one Scandinavian country influence the others. Furthermore, it would be equally interesting to explore whether the volatility from the US market has a spillover effect on the Scandinavian markets.

Finally, the correlation observed between NORVIX and DANVIX and their respective underlying markets was not as strong as the correlation between the S&P500 and VIX. It would be worthwhile to explore if the return movements in indices like NORVIX could be more comprehensively explained by factors such as fluctuations in oil prices.

8 Conclusion

In this thesis we have replicated the model-based and model-free implied volatility index methodology to Norway, Sweden and Denmark, for the period of 2018 to 2023. Then constructed a composite implied volatility index for the Scandinavian region. Adding to existing literature further evidence for the predictive power of the implied volatility indices and the application of Extreme Gradient Boosting to predict future realised volatility. For our four hypothesis, we conclude with the following results.

H1: There exists a negative contemporaneous relationship between the Scandinavian implied volatility indices returns and underlying indices returns, thus they will act as a fear gauge. In regards to the first hypothesis, all of the Scandinavian implied volatility indices holds. We found significant evidence for the negative contemporaneous relationship between each implied volatility index return and it's associated underlying index returns This is the same conclusion which Bugge et al. (2016) and Fassas and Siriopoulos (2020) arrived at.

H2: The return-volatility relationship is asymmetric, meaning the Scandinavian implied volatility indices react differently to negative and positive returns. For our second hypothesis our country-specific implied volatility indices, NORVIX, DANVIX and SWEVIX, holds. We showcase both the asymmetry under the assumption of the asymmetric effect being static across the log-return distribution, but also the different levels of assymetrics return responses to the underlying returns in each of the 10th quantiles. We showcase the same for the SCANDI-VIX with both OMXN40 and the SCI index as underlying, though the SCANDI-VIX asymmetric return effect cannot be said to be monotonically increasing in absolute values. The effects observed are consisten with the leverage effect theory, which is what Bugge et al. (2016) and Fassas and Siriopoulos (2020) concluded with.

H3: Using implied volatility for forecasting surpasses GARCH models, especially when it's integrated into the machine learning algorithm XGBoost. Akin to the results in Teller et al. (2022), we find that including implied volatility in our XGBoost regressor significantly improves the forecasting quality. However, whether to utilise implied volatility in isolation or as a compliment to lagged realised volatility, varies across the Scandinavian markets. We conclude with this part of the hypothesis holding. However, our EGARCH model is not a fair model to run XGBoost against in a "horse race". Due to the non-stationarity in the underlying index returns, the EGARCH model's capabilities to do out-of-sample forecasts becomes severely limited. This does highlight a weakness of the GARCH family of models, the sensitivity of the data's structure, which XGBoost avoids. We can in the end conclude that including implied volatility in predictive models improves the forecasting quality significantly. However, we cannot conclude that XGBoost outright out-competes EGARCH in this application, due to issues with non-stationarity in the input data.

H4: The composite proxy of the three Scandinavian implied volatility indices will absorb information well enough for it to serve as a useful predictive and descriptive tool of the OMXN40. For our final hypothesis, we find that SCANDI-VIX has a high rate of information absorption with the OMXN40, surpassing the weighted combination of the underlying Scandinavian indices (SCI index). Additionally, we find that the SCANDI-VIX has useful forecasting abilities for the OMXN40. We therefore conclude that H4 holds.

References

Akaike, H. (1973). Maximum likelihood identification of Gaussian autoregressive moving average models. *Biometrika*, *60*(2), 255–265. https://doi.org/10.1093/biomet/60.2.255

Black, F. (1976). The pricing of commodity contracts. *Journal of Financial Economics*, *3*(1-2), 167–179. https://doi.org/10.1016/0304-405X(76)90024-6

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Cart. *Classification and regression trees.*

Britten-Jones, M., & Neuberger, A. (2000). Option prices, implied price processes, and stochastic volatility. *The Journal of Finance*, *55*(2), 839–866. https://doi.org/https://doi.org/10.1111/0022-1082.00228

Bugge, S. A., Guttormsen, H. J., Molnár, P., & Ringdal, M. (2016). Implied volatility index for the norwegian equity market. *International Review of Financial Analysis*, *47*, 133–141. https://doi.org/https://doi.org/10.1016/j.irfa.2016.07.007

Campbell, J., & Hentschel, L. (1992). No news is good news *1: An asymmetric model of changing volatility in stock returns. *Journal of Financial Economics*, *31*(3), 281–318. https://EconPapers.repec.org/RePEc:eee:jfinec:v:31:y:1992:i:3:p:281-318

Carr, P., & Wu, L. (2005). A tale of two indices [Accessed on December 14, 2023]. https://ssrn.com/abstract=871729

Cboe Global Indices, LLC. (2023). *Volatility index®methodology: Cboe volatility index®* [Last accessed on December 14, 2023]. https://cdn.cboe.com/api/global/us_indices/governance/Volatility_Index_Methodology_Cboe_Volatility_Index.pdf

Cboe Global Markets. (2021). Vxo and vxhyg indices consultation summary [Retrieved December 11, 2023]. https://cdn.cboe.com/resources/release_notes/2021/VXO-and-VXHYG-Indices-Consultation-Summary.pdf

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system, 785–794. https://doi.org/10.1145/2939672.2939785

Demeterfi, K., Derman, E., Kamal, M., & Zou, J. (1999). *More than you ever wanted to know about volatility swaps* (Research Note No. 41). Goldman Sachs Quantitative Strategies Research Notes.

Dickey, D. A., & Fuller, W. A. (1979). Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, *74*(366), 427–431. Retrieved December 15, 2023, from http://www.jstor.org/stable/2286348

Durbin, J., & Watson, G. S. (1971). Testing for serial correlation in least squares regression.III. *Biometrika*, *58*(1), 1–19. https://doi.org/10.1093/biomet/58.1.1

Fassas, P. A., & Siriopoulos, C. (2020). Implied volatility indices – a review [July 8]. *The Quarterly Review of Economics and Finance.* https://www.sciencedirect.com/science/article/pii/S1062976920300855

Figlewski, S., & Wang, X. (2000, November). Is the 'leverage effect' a leverage effect? https://doi.org/10.2139/ssrn.256109

Fleming, J., Ostdiek, B., & Whaley, R. E. (1995). Predicting stock market volatility: A new measure. *Journal of Futures Markets*, *15*, 265–302. https://doi.org/10.1002/fut.3990150303

Genium inet®itch protocol specification (tech. rep.). (2017, August) (© 2017 Nasdaq, Inc. All Rights Reserved. Commercial in Confidence). Nasdaq.

Gonzalez-Perez, M. T., & Novales, A. (2011). The information content in a volatility index for spain. *SERIEs*, *2*, 185–216. https://doi.org/10.1007/s13209-010-0031-6

Hansen, C. S. (1999, September). The relationship between implied and realized volatility in the danish option and equity markets. https://doi.org/10.2139/ssrn.197608

Hull, J. (2022). *Options, futures, and other derivatives* (11th). Pearson.

Itch specification for nasdaq derivatives markets: Nordic equity derivatives (tech. rep.). (2022, November) (Effective from November 10, 2022). Nasdaq.

Jiang, G., & Tian, Y. S. (2007). Extracting model-free volatility from option prices: An examination of the vix index [January 8]. *Journal of Derivatives*, *14*(3). https://ssrn.com/abstract=880459

Lyócsa, Š., Molnár, P., & Výrost, T. (2021). Stock market volatility forecasting: Do we need high-frequency data? *International Journal of Forecasting*, *37*(3), 1092–1110. https://doi.org/https://doi.org/10.1016/j.ijforecast.2020.12.001

Mota, P. P. (2012). Normality assumption for the log-return of the stock prices. *Discussiones Mathematicae Probability and Statistics*, *32*, 47–58. https://doi.org/10.7151/dmps.1143

Nasdaq. (2023). Nasdaq omx indexes [Retrieved December 11, 2023]. https://indexes.nasdaqomx.com/

Öström, E. (2015, December). *A swedish model-free implied volatility index constructed from omxs30 options* [Master's thesis, University of Gothenburg]. https://gupea.ub.gu.se/bitstream/handle/2077/44287/gupea_2077_44287_1.pdf?sequence=1

Refinitiv. (2023). Eikon python api [Python 3.10.9]. https://developers.lseg.com/en/api-catalog/eikon/eikon-data-api

SciPy. (2023, November). Cubicspline – scipy v1.11.4 reference guide [Retrieved from SciPy Documentation].

Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, *52*(3/4), 591–611. Retrieved December 15, 2023, from http://www.jstor.org/stable/2333709

Shin, Y., & Schmidt, P. (1992). The kpss stationarity test as a unit root test. *Economics Letters*, *38*(4), 387–392. https://doi.org/https://doi.org/10.1016/0165-1765(92)90023-R

Skiadopoulos, G. (2004). The greek implied volatility index: Construction and properties. *Applied Financial Economics*, *14*(16), 1187–1196. https://doi.org/10.1080/0960310042000280438

Slim, S., Dahmene, M., & Boughrara, A. (2020). How informative are variance risk premium and implied volatility for value-at-risk prediction? international evidence. *The Quarterly Review of Economics and Finance*, *76*, 22–37.

Teller, A., Pigorsch, U., & Pigorsch, C. (2022, November). Short- to long-term realized volatility forecasting using extreme gradient boosting. https://doi.org/10.2139/ssrn.4267541

Whaley, R. E. (1993). Derivatives on market volatility: Hedging tools long overdue. *Journal of Derivatives*, *1*, 71–84. https://doi.org/10.3905/jod.1993.407868

Whaley, R. E. (2000). The investor fear gauge [Spring]. *Journal of Portfolio Management*. https://www.proquest.com/docview/195578390?pq-origsite=gscholar&fromopenview=true

Appendices

A ITCH message types

A.1 Key ITCH messages in the GITS format

Message Type	**Message Example**	**Comment**
S: System Event Message	S, 2021-02-26 T00:45:44.000000000 (1614300344000000000), O	The system event message type is used to signal a market or data feed handler event. This is usually start and end of messages for the day
R: Order Book Directory	R, 2021-02-26 T00:45:44.621179800 (1614300344621179800), 853268, OMXO201Q955, , SE0015578356, 1, NOK, 2, 0, 0, 1, 0, 0	At the start of each trading day, Order book directory messages are disseminated for all active securities
A: Add order Events	A, 2021-02-26 T09:07:33.904618800 (1614330453904618800), 7395587891604886683, OMXC251Q1590(37044257), S, 1, 50, 5550, 0, 2	An Add Order Message indicates that a new order has been accepted by the Trading system and was added to the displayable book. The message includes an Order ID that is unique per order book and side used by the system to track the order.
C: Order Executed Message	E, 2021-02-26 T09:07:34.512617500 (1614330454512617500), 7395587891604702620, OMXS301C(208929722), S, 1, 73955630280361439:0, ,	This message is sent whenever an order on the book is executed in whole or in part. If the incoming order causing the match cannot be fully filled, the remainder will be placed in the book after the match has occurred.
D: Single Delete	D, 2021-02-26 T09:07:33.912039800 (1614330453912039800), 7395587891604736430, OMXS301X1800(69731258), B	This message is sent whenever an order on the book is being deleted. There will be no remaining quantity, so the order should be removed from the book.

Table A.1: Key ITCH messages in the GITS format

A.2 Key ITCH message in the NITCH format

Message Type	Message Example	Comment
S: System Event	msgType=Stimestamp=9091018225459trackingNumber=0eventCode=O(START_OF_MESSAGES)	The system event message type is used to signal a market or data feed handler event. This is usually start and end of messages for the day
R: Derivative Directory	msgType=Rtimestamp=6026825447966trackingNumber=0instrumentId=6516underlyingSymbol="OMXS30"underlyingId=188underlyingType=02(INDEX)underlyingIsin="SE0000337842"isin="SE0020162915"tickSizeTableId=5expYear=18expMonth=06expDate=14expType=M(MONTHLY)strikePrice=170000000000optionType=C(CALL)productSymbol="OXS30"instrumentType=01(OPTION)tradingCurrency="SEK"mic="SEED"instrumentSymbol="OMXS304F1700"settlementMethod=C(CASH)exerciseStyle=E(EUROPEAN)returnType=P(PRICE_RETURN)contractSize=100version=0flexible=N(NOT_FLEXIBLE)traded=Y(TRADED)lastTradingYear=18lastTradingMonth=06lastTradingDate=14referenceInstrumentId=0tarIndicator=N(NONE)reserved=""	Derviative Directory messages are distributed at the beginning of the day right after system starts and lists all the tradeable instruments for this day. Here it has been noticed that the expiration dates and last trading days are not updated and we have solely used the instrument symbol to defer expiration dates according to the specification given by Nasdaq

Table A.2: Key ITCH message in the NITCH format part1

Message Type	Message Example	Comment
A: Add order	msgType=Atimestamp=22200224952837trackingNumber=0instrumentId=19689orderReference=2mktSide=S(SELL)price=5900000volume=1rank=1	Add Order Message indicates that a new order has been accepted by the system and added to the displayable book. The message includes a day-unique Order Reference Number used to track the order.
D: Order Delete Message	msgType=Dtimestamp=25260093059307trackingNumber=0orderReference=100	This message is sent whenever an order on the book or a side of a quote is being cancelled. All remaining volume is no longer accessible so the order should be removed from the book.
U: Order Replace Message	timestamp=27000056008550trackingNumber=1origOrderReference=234orderReference=236volume=6price=2131000000rank=1	This message is sent whenever an order on the book or a side of a quote has been cancel-replaced. All remaining volume from the original order is no longer accessible and must be removed. The new order details are provided for the replacement, along with a new order reference number which will be used henceforth.
E: Order Executed Message	msgType=Etimestamp=27000060286634trackingNumber=0reference=231volume=1comboId=0matchId=1	This message is sent whenever an order or quote on the book is executed in whole or in part. It is possible to receive several Order Executed Messages for the same order if that order is executed in several parts. Multiple Order Executed Messages on the same order are cumulative.
P: Trade Non Cross	msgType=Ptimestamp=27931904491684trackingNumber=1instrumentId=32832volume=2comboId=0matchId=63price=2122500000	A trade that is flagged to the system that is done outside of the normal bidding. Non cross means that there are two (or more) institutions doing the trade. (Cross trades are ignored as they are trades where the same institution holds both sides.)

Table A.3: Key ITCH message in the NITCH format part2

Message Type	**Message Example**	**Comment**
J: Add Quote	msgType=Jtimestamp=28805845091036trackingNumber=0instrumentId=31672bidReference=2778askReference=2779bidPrice=2042500000bidSize=1askPrice=2046500000askSize=1bidRank=1askRank=1	The Add Quote message contains information for a quote. A quote is typically double-sided, i.e., both a BidReferenceNumber and an AskReferenceNumber (with the corresponding bid and ask fields) are present. If the bid or ask reference number is zero, the quote message does not contain any information for that side and all corresponding bid or ask fields are zero.
K: Replace Quote	msgType=Ktimestamp=28806050091877trackingNumber=0instrumentId=31672origBidReference=2778bidReference=2796origAskReference=2779askReference=2797bidPrice=2042750000bidSize=1askPrice=2046750000askSize=1bidRank=1askRank=1	This message is sent whenever a quote on the book is replaced. The replaced quote has a new reference number. The new reference number replaces the prior reference number on the quote. Note: Original Reference Number is “zero” for quote items where a prior bid or ask quote does not exist. If New Reference Number is “zero,” the original quote item is deleted.
Y: Quote Delete	msgType=Ytimestamp=28809584311649trackingNumber=0bidReference=3211askReference=3212	This message is sent whenever an order on the book is being cancelled. All remaining volume is no longer accessible so the order should be removed from the book.

Table A.4: Key ITCH message in the NITCH format part3

B Code Listings

Listings

B.1 Building orderbook

```python
from datetime import datetime, timedelta
from sortedcontainers import SortedList
import pandas as pd

class Instrument:
    """
    A financial instrument, this would be a option.
    """
    def __init__(
        self,
        instrumentId: int,
        isin: str,
        underlyingId: str,
        underlyingSymbol: str,
        strikePrice: str,
        optionType: str,
        currency: str,
        instrumentSymbol: str,
    ):
        self.instrumentId = instrumentId
        self.isin = isin
        self.underlyingId = underlyingId
        self.underlyingSymbol = underlyingSymbol
        self.strikePrice = int(strikePrice)//10**8
        self.optionType = optionType
        self.currency = currency
        self.instrumentSymbol = instrumentSymbol

        self.failed = 0
        self.failed_orders = []

        self.last_exec = (None, None, None)
        #Tuple of (rank, timestamp, price, size side)
        self.best_bid = (None, None, None, None, None)
        self.best_ask = (None, None, None, None, None)

        self.orders = {} #Tracks the orders and the values
        self.askList = SortedList() #List of asks
        self.bidList = SortedList(key=lambda x: -x[0]) #List of bids


    def print_details(self):
        print(f"Instrument ID: {self.instrumentId}")
        print(f"Instrument Symbol: {self.instrumentSymbol}")
        print(f"Underlying Symbol: {self.underlyingSymbol}")
        print(f"Strike Price: {self.strikePrice}")
        print(f"Option Type: {self.optionType}")
        print(f"Currency: {self.currency}")

    def save_order_book(self, date):
        """Save the order book to a df."""
        df = None
        self.find_best_ask()
        self.find_best_bid()
        if (self.best_bid != (None, None, None, None, None)) and (self.best_ask != (None
    , None, None, None, None)):
            df = pd.DataFrame({
            'Date': [date],
            'Instrument': [self.instrumentSymbol],
            'Bid_Price': [self.best_bid[1]/10**6],
            'Bid_Volume': [self.best_bid[2]],
            'Bid_Time': [ns_to_hours(self.best_bid[0])],
            'Ask_Price': [self.best_ask[1]/10**6],
            'Ask_Volume': [self.best_ask[2]],
            'Ask_Time': [ns_to_hours(self.best_ask[0])],
            'Last_Price': [None if self.last_exec[1] is None else self.last_exec[1] /
    10**6],
            'Last_Volume': [self.last_exec[2]],
            'Last_Time': [ns_to_hours(self.last_exec[0])]
            })

            return df

    def add_order(self, orderTuple):
        """Add an order to the order book."""
        orderreference, timestamp, mktSide, price, volume, rank = orderTuple
        if int(orderreference) == 0:
            return
        if volume <= 0: #If volume is 0 or less do not bother adding to order book
            return
        order = Order(orderreference, self.instrumentId, timestamp, mktSide, price,
    volume, rank)
        self.orders[orderreference] = order
        if mktSide == 'S' or mktSide == 'N': #If ask
            asktuple = (rank, price, timestamp, volume, orderreference)
            self.askList.add(asktuple) #Increasing order
            #Change the rank of all other orders if rank is same as others
            for i in range(len(self.askList)):
                irank, iprice, itimestamp, ivolume, iorderref = self.askList[i]
                #if rank is same or higher and not the new order
                if irank >= rank and iorderref != orderreference:
                    self.askList.pop(i)
                    self.askList.add((irank+1, iprice, itimestamp, ivolume, iorderref))
                    self.orders[iorderref].rankup()
        elif mktSide == 'B' or mktSide == 'M': #If bid
```

```python
            bidtuple = (-rank, price, timestamp, volume, orderreference)
            self.bidList.add(bidtuple) #Decreasing order
            #Change the rank of all other orders if rank is same as others
            for i in range(len(self.bidList)):
                irank, iprice, itimestamp, ivolume, iorderref = self.bidList[i]
                irank = -irank
                if irank >= rank and iorderref != orderreference:
                    self.bidList.pop(i)
                    self.bidList.add((-(irank+1), iprice, itimestamp, ivolume, iorderref))
                    self.orders[iorderref].rankup()

    def execute_order(self, orderTuple):
        """Execute an order from the order book."""
        orderreference, timestamp, price, volume =  orderTuple
        if orderreference in self.orders:
            oldOrder = self.orders[orderreference]
            n_price = oldOrder.price
            if price is not None:
                n_price = price
            n_vol = oldOrder.vol - volume
            newOrder = Order(orderreference, self.instrumentId, timestamp, oldOrder.side, n_price, n_vol, oldOrder.rank)
            self.last_exec = (ns_to_hours(timestamp), n_price, volume)
            if n_vol <= 0:
                self.delete_order(orderreference)
            else:
                self.orders[orderreference] = newOrder
                if newOrder.side == 'S' or newOrder.side == 'N':
                    self.askList.remove((oldOrder.rank, oldOrder.price, oldOrder.timestamp, oldOrder.vol, orderreference))
                    self.askList.add((newOrder.rank, newOrder.price, newOrder.timestamp, newOrder.vol, orderreference))
                elif newOrder.side == 'B' or newOrder.side == 'M':
                    self.bidList.remove((-oldOrder.rank, oldOrder.price, oldOrder.timestamp, oldOrder.vol, orderreference))
                    self.bidList.add((-newOrder.rank, newOrder.price, newOrder.timestamp, newOrder.vol, orderreference))

    def delete_order(self, reference):
        """Delete an order from the order book."""
        order = self.orders.pop(reference, None)
        #askside
        if order.side == 'S' or order.side == 'N':
            askTuple = (order.rank, order.price, order.timestamp, order.vol, reference)
            pos = self.askList.index(askTuple)
            self.askList.remove(askTuple)
            #Change the rank of all other orders if rank is same or above as others
            for i in range(pos, len(self.askList)):
                irank, iprice, itimestamp, ivolume, iorderref = self.askList.pop(i)
                self.askList.add((irank-1, iprice, itimestamp, ivolume, iorderref))
                self.orders[iorderref].rankdown()
        #bidside
        elif order.side == 'B' or order.side == 'M':
            bidTuple = (-order.rank, order.price, order.timestamp, order.vol, reference)
            pos = self.bidList.index(bidTuple)
            self.bidList.remove(bidTuple)
            #change the rank of all other orders if rank is same or above as others
            for i in range(pos, len(self.bidList)):
                irank, iprice, itimestamp, ivolume, iorderref = self.bidList.pop(i)
                irank = -irank
                self.bidList.add((-(irank-1), iprice, itimestamp, ivolume, iorderref))
                self.orders[iorderref].rankdown()
        else:
            print('Error: Market side not recognized.')

    def trade(self, timestamp, price, volume):
        """Register a trade"""
        self.last_exec = (ns_to_hours(timestamp), price, volume)

    def flush(self, timestamp):
        if self.last_exec != (None, None, None):
            """Flush the order book."""
            self.orders = {}
            self.askList = SortedList()
            self.bidList = SortedList(key=lambda x: -x[0])

    def find_best_bid(self):
        """Print the best bid."""
        bestorderref = self.bidList[0][-1]
        bestorder = self.orders[bestorderref]
        self.best_bid = (bestorder.timestamp, bestorder.price, bestorder.vol,
    bestorderref)
        return self.best_bid

    def find_best_ask(self):
        """Print the best ask."""
        bestorderref = self.askList[0][-1]
        bestorder = self.orders[bestorderref]
        self.best_ask = (bestorder.timestamp, bestorder.price, bestorder.vol,
    bestorderref)
        return self.best_ask

class Order:
    def __init__(
        self,
        orderId: str,
        instrumentId: str,
        timestamp: int,
        side: str,
        price: int,
        vol: int,
        rank: int
    ):
        self.orderId = orderId
        self.instrumentId = instrumentId
        self.timestamp = timestamp
        self.side = side
        self.price = price
        self.vol = vol
        self.rank = rank

    def print_details(self):
        print(f'Order_{self.orderId} at {ns_to_hours(self.timestamp)}, {self.side} at {
    self.price} for {self.vol} units.')

    def rankup(self):
        self.rank += 1

    def rankdown(self):
        self.rank -= 1

def ns_to_hours(ns):
    """Convert nanoseconds to hours."""
    if ns is None:
        return None
    midnight = datetime(2013, 1, 1)
    hour = midnight + timedelta(seconds=ns//1e9)
    return hour.strftime('%H:%M:%S')
```

Listing 1: Classes used for building orderbook

```python
import gzip
from datetime import datetime, timedelta
from models import *
import os
import pandas as pd

#### Configurations ####
whitelist = ["OMXO20", "OMXS30", "OMXC25"]
input_path = 'ITCH/input/'
output_path = 'ITCH/output/'
output_path_unique = 'ITCH/unique_output/'

#List all files in the input path that are filtered
files = [f'{input_path}{f}' for f in os.listdir(input_path) if os.path.isfile(os.path.
    join(input_path, f)) and 'NDX_Filt' in f and f.endswith('.gz')]

#### Parsing Functions ####
def SystemEvent(line):
     #Handles system event messages like start end
    event_code = line[-2]
    if event_code.startswith("eventCode=O"):
        timestamp = line[2]
        timestamp = timestamp.split('=')[1]
        print(f'System started at {ns_to_hours(int(timestamp))}')
    elif event_code.startswith("eventCode=C"):
        timestamp = line[2]
        timestamp = timestamp.split('=')[1]
        print(f'System ended at {ns_to_hours(int(timestamp))}')


def DerivateDirectory(line, whitelist, instruments): #Handles derivative directory
    messages
    """
    This function parse all the instruments available for trade and stores them in a
    dictionary
    The whitelist is a list of instruments that we want to keep track of
    Derivative directory messages are the first messages sendt by the system
    """
    instrumentType = line[18].split('=')[-1]
    if instrumentType == '01(OPTION)': #Only care about options not futures
        underlyingSymbol = line[5].split('=')[-1]
        underlyingSymbol = underlyingSymbol.strip('"')
        if underlyingSymbol in whitelist:
            instrumentId = int(line[4].split('=')[-1])
            underlyingId = line[6].split('=')[-1]
            isin = line[9].split('=')[-1].strip('"')
            strikePrice = line[15].split('=')[-1]
            optionType = line[16].split('=')[-1]
            currency = line[19].split('=')[-1].strip('"')
            instrumentSymbol = line[21].split('=')[-1].strip('"')
            instrument = Instrument(instrumentId, isin, underlyingId, underlyingSymbol,
                                    strikePrice, optionType, currency, instrumentSymbol)
     #Create instrument object
            instruments[instrumentId] = instrument #Add instrument to dictionary

def AddOrder(line, instruments, order_map): #Handles add order messages
    """
    This function parses all the add order messages and stores them in a dictionary
    Orders will be added to a dictunary to keep track of the referencecode and the
    values
    We only care about orders that are for instruments that we are interested in
    """
    instrumentId = int(line[4].split('=')[-1])
    if instrumentId in instruments:
        instrument = instruments[instrumentId]
        orderreference = int(line[5].split('=')[-1])
        timestamp = int(line[2].split('=')[-1])
        mktSide = line[6].split('=')[-1][0]
        price = int(line[7].split('=')[-1])
        volume = int(line[8].split('=')[-1])
        rank = int(line[9].split('=')[-1])
        orderTuple = (orderreference, timestamp, mktSide, price, volume, rank)
        order_map[orderreference] = instrumentId
        instrument.add_order(orderTuple)
        return instrument
    return None


def OrderExecuted(line, instruments, order_map, msgtype): #Handles order executed
    messages
    """
    This function parses all the order executed messages and stores them in a list
    """
    orderreference = int(line[4].split('=')[-1])
    if orderreference in order_map:
        instrumentId = order_map[orderreference]
        instrument = instruments[instrumentId]
        timestamp = int(line[2].split('=')[-1])
        volume = int(line[5].split('=')[-1])
        price = None
        if msgtype == 'C':
            price = int(line[8].split('=')[-1])
        execTuple = (orderreference, timestamp, price, volume)
        instrument.execute_order(execTuple)
        return instrument
    return None

def DeleteOrder(line, instruments, order_map): #Handles delete order messages
```

```python
    orderreference = int(line[4].split('=')[-1])
    if orderreference in order_map:
        instrumentId = order_map[orderreference]
        instrument = instruments[instrumentId]
        instrument.delete_order(orderreference)
        del order_map[orderreference]
        return instrument
    return None

def OrderBookFlush(line, instruments): #Handles order book flush messages
    instrumentId = int(line[4].split('=')[-1])
    if instrumentId in instruments:
        instrument = instruments[instrumentId]
        timestamp = int(line[2].split('=')[-1])
        instrument.flush(timestamp)
    return None

def OrderReplace(line, instruments, order_map, msg_type): #Handles order replace messages
    old_reference = int(line[4].split('=')[-1])
    if old_reference in order_map:
        instrument_id = order_map[old_reference]
        instrument = instruments[instrument_id]
        timestamp = int(line[2].split('=')[-1])
        oldOrder = instrument.orders[old_reference]
        if msg_type == 'G':
            new_reference = old_reference
            new_volume = int(line[6].split('=')[-1])
            new_price = int(line[5].split('=')[-1])
            new_rank = oldOrder.rank
        elif msg_type == 'U':
            new_reference = int(line[5].split('=')[-1])
            new_volume = int(line[6].split('=')[-1])
            new_price = int(line[7].split('=')[-1])
            new_rank = int(line[8].split('=')[-1])
            del order_map[old_reference]
            order_map[new_reference] = instrument_id
        newOrder = (new_reference, timestamp, oldOrder.side, new_price, new_volume, new_rank)
        instrument.delete_order(old_reference)
        instrument.add_order(newOrder)
        return instrument
    return None

def TradeMessage(line, instruments): #Handles trade messages
    instrumentId = int(line[4].split('=')[-1])
    if instrumentId in instruments:
        instrument = instruments[instrumentId]
        timestamp = int(line[2].split('=')[-1])
        price = int(line[8].split('=')[-1])
        volume = int(line[5].split('=')[-1])
        instrument.trade(timestamp, price, volume)

def AddQuote(line, instruments, order_map):
    instrumentId = int(line[4].split('=')[-1])
    if instrumentId in instruments:
        instrument = instruments[instrumentId]
        timestamp = int(line[2].split('=')[-1])
        bidreference = int(line[5].split('=')[-1])
        bidPrice = int(line[7].split('=')[-1])
        bidVolume = int(line[8].split('=')[-1])
        bidRank = int(line[11].split('=')[-1])
        askreference = int(line[6].split('=')[-1])
        askPrice = int(line[9].split('=')[-1])
        askVolume = int(line[10].split('=')[-1])
        askRank = int(line[12].split('=')[-1])
        if bidreference != 0:
            bidTuple = (bidreference, timestamp, 'B', bidPrice, bidVolume, bidRank)
            instrument.add_order(bidTuple)
            order_map[bidreference] = instrumentId
        if askreference != 0:
            askTuple = (askreference, timestamp, 'S', askPrice, askVolume, askRank)
            instrument.add_order(askTuple)
            order_map[askreference] = instrumentId
        return instrument
    return None

def ReplaceQuote(line, instruments, order_map):
    instrumentId = int(line[4].split('=')[-1])
    if instrumentId in instruments:
        instrument = instruments[instrumentId]
        timestamp = int(line[2].split('=')[-1])
        oldBidReference = int(line[5].split('=')[-1])
        oldAskReference = int(line[7].split('=')[-1])
        if oldBidReference in order_map:
            instrument.delete_order(oldBidReference)
            del order_map[oldBidReference]
        if oldAskReference in order_map:
            instrument.delete_order(oldAskReference)
            del order_map[oldAskReference]
        newBidRef = int(line[6].split('=')[-1])
        bidPrice = int(line[9].split('=')[-1])
        bidVolume = int(line[10].split('=')[-1])
        bidRank = int(line[13].split('=')[-1])
        newAskRef = int(line[8].split('=')[-1])
        askPrice = int(line[11].split('=')[-1])
        askVolume = int(line[12].split('=')[-1])
        askRank = int(line[14].split('=')[-1])
        if newBidRef != 0:
            instrument.add_order((newBidRef, timestamp, 'B', bidPrice, bidVolume,
    bidRank))
            order_map[newBidRef] = instrumentId
        if newAskRef != 0:
            instrument.add_order((newAskRef, timestamp, 'S', askPrice, askVolume,
    askRank))
            order_map[newAskRef] = instrumentId
        return instrument
    return None

def DeleteQuote(line, instruments, order_map):
    bidReference = int(line[4].split('=')[-1])
    askReference = int(line[5].split('=')[-1])
    if bidReference in order_map:
        instrumentId = order_map[bidReference]
        instrument = instruments[instrumentId]
        instrument.delete_order(bidReference)
        del order_map[bidReference]
        return instrument
    if askReference in order_map:
        instrumentId = order_map[askReference]
        instrument = instruments[instrumentId]
        instrument.delete_order(askReference)
        del order_map[askReference]
        return instrument
    return None

def ns_to_date(date, ns):
    """
    This function converts nanoseconds to a date
    """
    date = date + timedelta(seconds=ns/1e9)
    return date

def time_interval(timestamp):
    """
    This function returns the time interval that the timestamp belongs to according to a
     1-hour interval.
    """
    total_minutes = timestamp.hour * 60 + timestamp.minute
    # Calculate how many minutes to subtract to get to the nearest 4-hour interval start
    minutes_to_subtract = total_minutes % (1*60)
    interval = timestamp - timedelta(minutes=minutes_to_subtract, seconds=timestamp.
    second, microseconds=timestamp.microsecond)
    return interval

###### Main loop #####
#Main
```

```python
for filepath in files:
    date = datetime.strptime(filepath.split('_')[-1].split('.')[0], '%Y%m%d')
    date_str = date.strftime('%Y%m%d')
    instruments = {}
    order_map = {}
    dayframe = pd.DataFrame(columns=['Date', 'Instrument', 'Bid_Price', 'Bid_Volume', '
    Bid_Time', 'Ask_Price', 'Ask_Volume', 'Ask_Time', 'Last_Price', 'Last_Volume', '
    Last_Time'])
    current_interval = None
    with gzip.open(filepath, 'rt') as file:
        for linenumber, line in enumerate(file):
            parts = line.split()
            msg_type = parts[1][-1]
            inst = None

            if msg_type == 'S':
                SystemEvent(parts)

            elif msg_type == 'R':
                DerivateDirectory(parts, whitelist, instruments)

            elif msg_type == 'A':
                inst = AddOrder(parts, instruments, order_map)

            elif msg_type == 'E' or msg_type == 'C':
                inst = OrderExecuted(parts, instruments, order_map, msg_type)

            elif msg_type == 'D':
                inst = DeleteOrder(parts, instruments, order_map)

            elif msg_type == 'y':
                try:
                    inst = OrderBookFlush(parts, instruments, order_map)
                except KeyError as e:
                    inst = None
                    print(e)
                    print(f'KeyError for line {line.strip()}')

            elif msg_type == 'U' or msg_type == 'G':
                inst = OrderReplace(parts, instruments, order_map, msg_type)

            elif msg_type == 'P':
                inst = TradeMessage(parts, instruments)

            elif msg_type == 'J':
                inst = AddQuote(parts, instruments, order_map)

            elif msg_type == 'K':
                inst = ReplaceQuote(parts, instruments, order_map)

            elif msg_type == 'Y':
                inst = DeleteQuote(parts, instruments, order_map)

            # Save to csv if we have a new hour interval
            if inst is not None:
                timestamp = ns_to_date(date, int(parts[2].split('=')[-1]))
                interval = time_interval(timestamp)
                if interval != current_interval:
                    for ins in instruments.values():
                        dataline = ins.save_order_book(interval)
                        dayframe = pd.concat([dayframe, dataline], ignore_index=True)
                    current_interval = interval

    # Remove instruments with bid>ask
    dayframe = dayframe[dayframe['Bid_Price'] <= dayframe['Ask_Price']]

    dayframe.to_csv(output_path+date_str+'.csv', index=False)

    dayframe = dayframe.drop_duplicates(subset=['Instrument'], keep='last')
    dayframe.to_csv(output_path_unique+date_str+'_unique.csv', index=False)

print('Done!')
```

Listing 2: Building orderbook

B.2 Model-based index code

```python
import numpy as np
import pandas as pd
import datetime
import time
import warnings
import os
import math
from scipy.interpolate import CubicSpline
from scipy.stats import norm
from scipy.optimize import brentq

##### Settings #####
#Paths
input_path = 'ITCH/unique_output/'
omxo20 = 'Data/VIX/Final/BSM_norvixV2.csv'
omxc25 = 'Data/VIX/Final/BSM_danvixV2.csv'
omxs30 = 'Data/VIX/Final/BSM_swevixV2.csv'

O20 = 'Data/VIX/Raw/OMXO20.csv'
C25 = 'Data/VIX/Raw/OMXC25.csv'
S30 = 'Data/VIX/Raw/OMXS30.csv'

#List all files in the input path
files = [f'{input_path}{f}' for f in os.listdir(input_path) if os.path.isfile(os.path.
    join(input_path, f))]
names = ['OMXO20', 'OMXC25', 'OMXS30']

#List of paths to cleaned rate files
rate_paths = [
    'Data/VIX/Processed/NIBOR_cleaned.csv',
    'Data/VIX/Processed/CIBOR_cleaned.csv',
    'Data/VIX/Processed/STIBOR_cleaned.csv'
]

###### Functions ######
def clean_file(df):
    """Extracts the relevant information from the
    option name and returns a cleaned dataframe."""
    def process_maturity(row):
        CALLS = {
        'A': 1, 'B': 2, 'C': 3, 'D': 4,
        'E': 5, 'F': 6, 'G': 7, 'H': 8,
        'I': 9, 'J': 10, 'K': 11, 'L': 12
        }
        PUTS =  {
            'M': 1, 'N': 2, 'O': 3, 'P': 4,
            'Q': 5, 'R': 6, 'S': 7, 'T': 8,
            'U': 9, 'V': 10, 'W': 11, 'X': 12
        }
        mat = row['Maturity']

        #find year
        y = int(mat[0])
        if y > 5:
            exp_year = 2010 + y
        else:
            exp_year = 2020 + y

        #find month
        m = mat[1]
        if m in CALLS:
            exp_month = CALLS[m]
            exp_type = 'Call'
        elif m in PUTS:
            exp_month = PUTS[m]
            exp_type = 'Put'
        else:
            raise ValueError(f'Unknown month {m} in {mat}')

        #find day
        if mat[-1] == 'Y':
            exp_day = int(mat[-3:-1]) if mat[-2].isdigit() else int(mat[-2])
        else:
            exp_day = third_friday(datetime.date(exp_year, exp_month, 1)).day
        return pd.Series([exp_year, exp_month, exp_day, exp_type])

    def third_friday(date):
        # Find out what day of the week the first day of the month is
        first_day = date.weekday()
        # Calculate how many days are needed to get to the first Friday
        days_to_first_friday = (4 - first_day) % 7
        # Return the third Friday
        return date + pd.Timedelta(days=days_to_first_friday + 14)

    # Regular Expression to extract the components
    # \w{6} captures the first 6 characters for Underlying
    # (.*?) captures everything in a non-greedy way up to the final digits for Maturity
    # (\d+)$ captures the final digits for Strike Price
    pattern = r'(\w{6})(.*?)(\d+)$'

    # Apply the regex pattern
    df[['Underlying', 'Maturity', 'Strike']] = df['Instrument'].str.extract(pattern)
    df['Strike'] = df['Strike'].astype(float)
    df[['Exp year', 'Exp month', 'Exp day', 'Option Type']] = df.apply(process_maturity,
     axis=1, result_type='expand')
    df['Exp date'] = pd.to_datetime(df['Exp year'].astype(str) +
                                    df['Exp month'].astype(str).str.zfill(2) +
                                    df['Exp day'].astype(str).str.zfill(2),
                                    format='%Y%m%d',  # Adjusting format to include time
                                    errors='coerce')
    df.drop(columns=['Exp year', 'Exp month', 'Exp day', 'Maturity', 'Bid_Time', '
    Ask_Time', 'Last_Time'], inplace=True)
    df['Date'] = pd.to_datetime(df['Date'])
    return df

def pivot_df(df):
    """Pivots the dataframe such that the
    call ans put options are in the same row."""
    df.reset_index(drop=True)
    #remove columns we don't
    df.drop(columns=['Date'], inplace=True)

    #Pivot table by strike and type
    df = df.pivot_table(index=['Underlying', 'Exp date', 'Strike'], columns='Option Type
    ', values=['Bid_Price', 'Ask_Price', 'Bid_Volume', 'Ask_Volume'])
    # Flatten the MultiIndex columns
    df = df.reset_index()

    # Flatten and rename the columns
    df.columns = ['_'.join(col).strip() if isinstance(col, tuple) else col for col in df
    .columns.values]
    df = df.rename(columns={
        'Ask_Price_Call': 'Call_Ask',
        'Ask_Price_Put': 'Put_Ask',
        'Bid_Price_Call': 'Call_Bid',
        'Bid_Price_Put': 'Put_Bid',
        'Bid_Volume_Call': 'Call_Volume',
        'Bid_Volume_Put': 'Put_Volume',
        'Strike_': 'Strike',
        'Exp date_': 'Exp date',
        'Underlying_': 'Underlying'
    })

    df = df[['Underlying', 'Exp date', 'Strike', 'Call_Ask', 'Call_Bid', 'Put_Ask', '
    Put_Bid', 'Call_Volume', 'Put_Volume']]

    #Calculate mid price
    df['Call'] = (df['Call_Ask'] + df['Call_Bid']) / 2
    df['Put'] = (df['Put_Ask'] + df['Put_Bid']) / 2
    df = df.drop(columns=['Call_Ask', 'Call_Bid', 'Put_Ask', 'Put_Bid'])

    df['Diff'] = (df['Call'] - df['Put']).abs()
    df = df.sort_values(by=['Exp date', 'Strike'])

    return df

#Validatting bounds for interpolation
def validate_interpolation(interpolated_y, interpolated_x, x, y):
    """Validates the interpolation by forcing the interpolated curve
    to be within the upper and lower bound of the original curve."""
    def find_upperlower(x,y):
        #finds upper and lower bound for the interpolated curve
        #x and y are the interpolated curve
        #returns mlower, mupper, blower, bupper
        t1 = x[0]
        CMT1 = y[0]
        tx = math.inf
        tz = math.inf
        for i in range(len(y)):
            if x[i] < tx and x[i] > t1 and y[i] >= CMT1:
                tx = x[i]
                CMTx = y[i]
            if x[i] < tz and x[i] > t1 and y[i] <= CMT1:
                tz = x[i]
                CMTz = y[i]
        if tx == math.inf:
            mlower = 0
        else:
            mlower = (CMTx - CMT1)/(tx - t1)
        if tz == math.inf:
            mupper = 0
        else:
            mupper = (CMTz - CMT1)/(tz - t1)
        blower = CMT1 - mlower*t1
        bupper = CMT1 - mupper*t1
        return mlower, mupper, blower, bupper

    interpolated_y = np.array(interpolated_y)
    interpolated_x = np.array(interpolated_x)
    x = np.array(x)
    y = np.array(y)
    upper_bound = np.zeros(len(interpolated_x))
    lower_bound = np.zeros(len(interpolated_x))
    #find upper and lower bound for the extrapolated curve
    if interpolated_x[0] < x[0]:
        mlower, mupper, blower, bupper = find_upperlower(x,y)
        upper_bound[:x[0]] = mupper*interpolated_x[:x[0]] + bupper
        lower_bound[:x[0]] = mlower*interpolated_x[:x[0]] + blower
    #find upper and lower bound for the interpolated curve
    for i in range(len(x)-1):
        minvalue=min(y[i], y[i+1])
        maxvalue=max(y[i], y[i+1])
        upper_bound[x[i]:x[i+1]] = maxvalue
        lower_bound[x[i]:x[i+1]] = minvalue

    #force interpolation to be within upper and lower bound
    interpolated_y = np.minimum(interpolated_y, upper_bound)
    interpolated_y = np.maximum(interpolated_y, lower_bound)

    return interpolated_y

# Maturity days mapping
maturity_days = {
    '1 Day': 1,
    '1 Wk': 7,
    '2 Wk': 14,
    '1 Mo': 30,
    '2 Mo': 60,
    '3 Mo': 91,
    '6 Mo': 182,
    '9 Mo': 273,
    '1 Yr': 365,
    '2 Yr': 730,
    '3 Yr': 1095,
    '5 Yr': 1825,
    '7 Yr': 2555,
    '10 Yr': 3650,
    '20 Yr': 7300,
    '30 Yr': 10950
}

def compute_rates(path, current_date, days_diffs):
    """Computes the rates for the terms"""
    yields = pd.read_csv(path)
    yields['Date'] = pd.to_datetime(yields['Date'])
    #Find the rates for the terms
    c = current_date
    rate = yields[yields['Date'] == c]
    #check if rate is found, if not, find the closest date or skip
    first_date = yields['Date'].iloc[0].date()
    while len(rate) == 0:
        if c < first_date:
            print(f'No rates found for this date {current_date}')
            break
        c = c - pd.Timedelta(days=1)
        rate = yields[yields['Date'] == pd.Timestamp(c)]
    if len(rate) == 0:
        return None
    #Extracting x (days to maturity) if in the yield column names
    x = [maturity_days[key] for key in yields.columns if key in maturity_days.keys()]
    interpolated_x = np.arange(0, max(x))
    row = yields.loc[yields['Date'] == current_date.strftime('%Y-%m-%d')]
```

```python
    row = rate
    y = [row[key] for key in yields.columns if key in maturity_days.keys()]
    ncs = CubicSpline(x, y, bc_type='natural')
    interpolated_y = ncs(interpolated_x, extrapolate=True)
    interpolated_y = validate_interpolation(interpolated_y, interpolated_x, x, y)
    interpolated_y = np.log(1 + interpolated_y) #From apy to r
    r = [interpolated_y[0][d] for d in days_diffs] #get the interpolated rates for each
    term date
    return r

def findspot(name, current_date):
    """Finds the spot price for the index on the current date"""
    if name == 'OMXO20':
        spot_df = pd.read_csv(O20)
    elif name == 'OMXC25':
        spot_df = pd.read_csv(C25)
    elif name == 'OMXS30':
        spot_df = pd.read_csv(S30)
    spot_df['Date'] = pd.to_datetime(spot_df['Date'])
    first_date = spot_df['Date'].iloc[0].date()

    #Find the spot price for the date or the former date
    spot = spot_df[spot_df['Date'] == pd.Timestamp(current_date)]
    c = current_date
    while len(spot) == 0:
        if c < first_date:
            break
        c = c - pd.Timedelta(days=1)
        spot = spot_df[spot_df['Date'] == c]
    if len(spot) == 0:
        return None
    else:
        spot = spot['Price'].iloc[0]
        return spot

def black_scholes_call(S, K, T, r, sigma):
    """
    Price of a European call option using Black-Scholes formula.
    """
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)

def black_scholes_put(S, K, T, r, sigma):
    """
    Price of a European put option using Black-Scholes formula.
    """
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)

def implied_volatility(option_price, S, K, T, r, option_type):
    """ Calculate the implied volatility using
    Black-Scholes formula and iterative searches """
    def bs_price(sigma):
        return (black_scholes_call(S, K, T, r, sigma) if option_type == 'call'
                else black_scholes_put(S, K, T, r, sigma)) - option_price

    return brentq(bs_price, 0.0001, 5)

###### Main ######

# Dataframes for the indices
norvix_df = pd.DataFrame(columns=['Date', 'Index'])
danvix_df = pd.DataFrame(columns=['Date', 'Index'])
swevix_df = pd.DataFrame(columns=['Date', 'Index'])

M365 = 365*1440 #Minutes in a year

#Start building the volatility index
for i, file in enumerate(files):
    date = datetime.datetime.strptime(file.split('/')[-1].split('_')[0], '%Y%m%d')

    #Import option chain
    total_df = pd.read_csv(file)
    total_df = clean_file(total_df)

    #Split into OMXO20, OMXC25 and OMXS30
    omxo20_df = total_df[total_df['Underlying'] == 'OMXO20']
    omxc25_df = total_df[total_df['Underlying'] == 'OMXC25']
    omxs30_df = total_df[total_df['Underlying'] == 'OMXS30']

    df_list = [omxo20_df, omxc25_df, omxs30_df]

    # iterate over the three indices
    for j, df in enumerate(df_list):
        name = names[j]
        current_date = df['Date'].iloc[0].date()
        df = pivot_df(df)

        exp_dates = df['Exp date'].dt.date.unique()
        exp_dates = np.sort(exp_dates)

        #filter out dates that are inside a 8 day window
        exp_dates = exp_dates[exp_dates > (current_date + datetime.timedelta(days=8))]
        #Find near term and next term as the two first dates
        near_term_date =  exp_dates[0]
        next_term_date = exp_dates[1]
        term_dates = [near_term_date, next_term_date]

        #days between current date and term dates
        days_diffs = [int((term - current_date).days) for term in term_dates]

        #Compute rates
        rates = compute_rates(rate_paths[j], current_date, days_diffs)

        minutes = [(term_date - current_date).total_seconds() // 60 for term_date in
    term_dates]
        times = [round(min / M365, 7) for min in minutes]

        ### Calculate the variance for the terms
        implied_volatilities = []
        trading_days = []

        for j, term_date in enumerate(term_dates):
            term_df = df[df['Exp date'] == pd.Timestamp(term_date)]

            ##ATM##
            spotprice = findspot(name, current_date)

            #stradle the spot price
            term_df['Diff'] = (term_df['Strike'] - spotprice)
            above = term_df[term_df['Diff'] > 0].sort_values(by=['Diff'])
            below = term_df[term_df['Diff'] <= 0].sort_values(by=['Diff'], ascending=
    False)
            above = above.iloc[0]
            below = below.iloc[0]

            above = above.to_frame().T
            below = below.to_frame().T

            #Filter to only contain near the money options
            atm = pd.concat([above, below], axis=0)
            atm = atm.sort_values(by=['Strike'])

            #calculate the implied volatility for the call and put
            atm['IV_Call'] = atm.apply(lambda row: implied_volatility(row['Call'],
    spotprice, row['Strike'], times[j], rates[j], 'call'), axis=1)
            atm['IV_Put'] = atm.apply(lambda row: implied_volatility(row['Put'],
    spotprice, row['Strike'], times[j], rates[j], 'put'), axis=1)
            atm['IV'] = (atm['IV_Call'] + atm['IV_Put']) / 2

            #Linear interpolation of the implied volatility
            atm_iv = np.interp(spotprice, atm['Strike'], atm['IV'])

            nc = days_diffs[j] #actual calenderdays
            nt = nc - 2 * nc//7 #trading days
```

```python

            trading_days.append(nt)
            trading_IV = atm_iv * (np.sqrt(nc)/np.sqrt(nt)) #trading day implied volatility

            implied_volatilities.append(trading_IV)

        ###### VXO ######
        vxo = implied_volatilities[0]*((trading_days[1]-22)/(trading_days[1]-trading_days[0])) + implied_volatilities[1]*((22-trading_days[0])/(trading_days[1]-trading_days[0]))
        vxo = round(vxo*100, 2)
        if name == 'OMXO20':
            print(f'{name} got an index of {vxo} for {current_date}')
            norvix_df = pd.concat([norvix_df, pd.DataFrame([[current_date, vxo]], columns=['Date', 'Index'])])
        elif name == 'OMXC25':
            print(f'{name} got an index of {vxo} for {current_date}')
            danvix_df = pd.concat([danvix_df, pd.DataFrame([[current_date, vxo]], columns=['Date', 'Index'])])
        elif name == 'OMXS30':
            print(f'{name} got an index of {vxo} for {current_date}')
            swevix_df = pd.concat([swevix_df, pd.DataFrame([[current_date, vxo]], columns=['Date', 'Index'])])

#Save to csv
norvix_df.sort_values(by=['Date'], inplace=True)
danvix_df.sort_values(by=['Date'], inplace=True)
swevix_df.sort_values(by=['Date'], inplace=True)
norvix_df.to_csv(omxo20, index=False)
danvix_df.to_csv(omxc25, index=False)
swevix_df.to_csv(omxs30, index=False)
```

Listing 3: Code for making the model-based volatility indices

B.3 Model-free index code

```python
import numpy as np
import pandas as pd
import datetime
import os
import math
from scipy.interpolate import CubicSpline


##### Settings #####
#Paths
input_path = 'ITCH/unique_output/'
omxo20 = 'Data/VIX/Final/ITCH_norvix.csv'
omxc25 = 'Data/VIX/Final/ITCH_danvix.csv'
omxs30 = 'Data/VIX/Final/ITCH_swevix.csv'

#List all files in the input path
files = [f'{input_path}{f}' for f in os.listdir(input_path) if os.path.isfile(os.path.
    join(input_path, f))]
names = ['OMXO20', 'OMXC25', 'OMXS30']

#List of paths to cleaned rate files
rate_paths = [
    'Data/VIX/Processed/NIBOR_cleaned.csv',
    'Data/VIX/Processed/CIBOR_cleaned.csv',
    'Data/VIX/Processed/STIBOR_cleaned.csv'
]

MCM = pd.Timedelta(days=30).total_seconds() / 60 #Minutes in the contant maturity period
M365 = 365*1440 #Minutes in a year

###### Functions ######
def clean_file(df):
    """Extracts the relevant information from the
    option name and returns a cleaned dataframe."""
    def process_maturity(row):
        # Function to process the maturity date
        CALLS = {
        'A': 1, 'B': 2, 'C': 3, 'D': 4,
        'E': 5, 'F': 6, 'G': 7, 'H': 8,
        'I': 9, 'J': 10, 'K': 11, 'L': 12
        }
        PUTS =  {
            'M': 1, 'N': 2, 'O': 3, 'P': 4,
            'Q': 5, 'R': 6, 'S': 7, 'T': 8,
            'U': 9, 'V': 10, 'W': 11, 'X': 12
        }
        mat = row['Maturity']

        #find year
        y = int(mat[0])
        if y > 5:
            exp_year = 2010 + y
        else:
            exp_year = 2020 + y

        #find month
        m = mat[1]
        if m in CALLS:
            exp_month = CALLS[m]
            exp_type = 'Call'
        elif m in PUTS:
            exp_month = PUTS[m]
            exp_type = 'Put'
        else:
            raise ValueError(f'Unknown month {m} in {mat}')

        #find day
        if mat[-1] == 'Y':
            exp_day = int(mat[-3:-1]) if mat[-2].isdigit() else int(mat[-2])
        else:
            exp_day = third_friday(datetime.date(exp_year, exp_month, 1)).day

        return pd.Series([exp_year, exp_month, exp_day, exp_type])

    def third_friday(date):
        # Find out what day of the week the first day of the month is
        first_day = date.weekday()
        # Calculate how many days are needed to get to the first Friday
        days_to_first_friday = (4 - first_day) % 7
        # Return the third Friday
        return date + pd.Timedelta(days=days_to_first_friday + 14)

    # Regular Expression to extract the components
    # \w{6} captures the first 6 characters for Underlying
    # (.*?) captures everything in a non-greedy way up to the final digits for Maturity
    # (\d+)$ captures the final digits for Strike Price
    pattern = r'(\w{6})(.*?)(\d+)$'

    # Apply the regex pattern
    df[['Underlying', 'Maturity', 'Strike']] = df['Instrument'].str.extract(pattern)

    df['Strike'] = df['Strike'].astype(float)
    df[['Exp year', 'Exp month', 'Exp day', 'Option Type']] = df.apply(process_maturity,
     axis=1, result_type='expand')
    df['Exp date'] = pd.to_datetime(df['Exp year'].astype(str) +
                                df['Exp month'].astype(str).str.zfill(2) +
                                df['Exp day'].astype(str).str.zfill(2),
                                format='%Y%m%d',  # Adjusting format to include time
                                errors='coerce')

    # Remove the unnecessary columns
    df.drop(columns=['Exp year', 'Exp month', 'Exp day', 'Maturity', 'Bid_Time', '
    Ask_Time', 'Last_Time'], inplace=True)
    df['Date'] = pd.to_datetime(df['Date'])
    return df

def pivot_df(df):
    """Pivots the dataframe such that the
    call ans put options are in the same row."""
    df.reset_index(drop=True)
    df.drop(columns=['Date'], inplace=True)

    #Pivot table by strike and type
    df = df.pivot_table(index=['Underlying', 'Exp date', 'Strike'], columns='Option Type
    ', values=['Bid_Price', 'Ask_Price', 'Bid_Volume', 'Ask_Volume'])
    df = df.reset_index()

    # Flatten and rename the columns
    df.columns = ['_'.join(col).strip() if isinstance(col, tuple) else col for col in df
    .columns.values]
    df = df.rename(columns={
        'Ask_Price_Call': 'Call_Ask',
        'Ask_Price_Put': 'Put_Ask',
        'Bid_Price_Call': 'Call_Bid',
        'Bid_Price_Put': 'Put_Bid',
        'Bid_Volume_Call': 'Call_Volume',
        'Bid_Volume_Put': 'Put_Volume',
        'Strike_': 'Strike',
        'Exp date_': 'Exp date',
        'Underlying_': 'Underlying'
    })

    #Only keep the columns we need
    df = df[['Underlying', 'Exp date', 'Strike', 'Call_Ask', 'Call_Bid', 'Put_Ask', '
    Put_Bid', 'Call_Volume', 'Put_Volume']]

    #Calculate mid price
    df['Call'] = (df['Call_Ask'] + df['Call_Bid']) / 2
    df['Put'] = (df['Put_Ask'] + df['Put_Bid']) / 2
    df = df.drop(columns=['Call_Ask', 'Call_Bid', 'Put_Ask', 'Put_Bid'])

    #Calculate diff
    df['Diff'] = (df['Call'] - df['Put']).abs()
    df = df.sort_values(by=['Exp date', 'Strike'])
```

```python
    return df

def validate_interpolation(interpolated_y, interpolated_x, x, y):
    """Validates the interpolation by forcing
    it to be within the upper and lower bound"""
    def find_upperlower(x,y):
        #finds upper and lower bound for the interpolated curve
        t1 = x[0]
        CMT1 = y[0]
        tx = math.inf
        tz = math.inf
        for i in range(len(y)):
            if x[i] < tx and x[i] > t1 and y[i] >= CMT1:
                tx = x[i]
                CMTx = y[i]
            if x[i] < tz and x[i] > t1 and y[i] <= CMT1:
                tz = x[i]
                CMTz = y[i]
        if tx == math.inf:
            mlower = 0
        else:
            mlower = (CMTx - CMT1)/(tx - t1)
        if tz == math.inf:
            mupper = 0
        else:
            mupper = (CMTz - CMT1)/(tz - t1)
        blower = CMT1 - mlower*t1
        bupper = CMT1 - mupper*t1
        return mlower, mupper, blower, bupper

    #convert all to np arrays
    interpolated_y = np.array(interpolated_y)
    interpolated_x = np.array(interpolated_x)
    x = np.array(x)
    y = np.array(y)
    upper_bound = np.zeros(len(interpolated_x))
    lower_bound = np.zeros(len(interpolated_x))

    #find upper and lower bound for the extrapolated curve
    if interpolated_x[0] < x[0]:
        mlower, mupper, blower, bupper = find_upperlower(x,y)
        upper_bound[:x[0]] = mupper*interpolated_x[:x[0]] + bupper
        lower_bound[:x[0]] = mlower*interpolated_x[:x[0]] + blower

    #find upper and lower bound for the interpolated curve
    for i in range(len(x)-1):
        minvalue=min(y[i], y[i+1])
        maxvalue=max(y[i], y[i+1])
        upper_bound[x[i]:x[i+1]] = maxvalue
        lower_bound[x[i]:x[i+1]] = minvalue

    #Edit the interpolated curve to be within the bounds
    interpolated_y = np.minimum(interpolated_y, upper_bound)
    interpolated_y = np.maximum(interpolated_y, lower_bound)
    return interpolated_y

# Maturity days mapping
maturity_days = {
    '1 Day': 1,
    '1 Wk': 7,
    '2 Wk': 14,
    '1 Mo': 30,
    '2 Mo': 60,
    '3 Mo': 91,
    '6 Mo': 182,
    '9 Mo': 273,
    '1 Yr': 365,
    '2 Yr': 730,
    '3 Yr': 1095,
    '5 Yr': 1825,
    '7 Yr': 2555,
    '10 Yr': 3650,
    '20 Yr': 7300,
    '30 Yr': 10950
}

def compute_rates(path, current_date, days_diffs):
    """Computes the riskless rates for the given terms."""
    yields = pd.read_csv(path)
    yields['Date'] = pd.to_datetime(yields['Date'])
    c = current_date
    rate = yields[yields['Date'] == c]
    first_date = yields['Date'].iloc[0].date()
    while len(rate) == 0:
        if c < first_date:
            print(f'No rates found for this date {current_date}')
            break
        c = c - pd.Timedelta(days=1)
        rate = yields[yields['Date'] == pd.Timestamp(c)]
    if len(rate) == 0:
        return None

    #Extracting x (days to maturity) if in the yield column names
    x = [maturity_days[key] for key in yields.columns if key in maturity_days.keys()]
    interpolated_x = np.arange(0, max(x))

    row = yields.loc[yields['Date'] == current_date.strftime('%Y-%m-%d')]
    row = rate

    y = [row[key] for key in yields.columns if key in maturity_days.keys()]
    ncs = CubicSpline(x, y, bc_type='natural')
    interpolated_y = ncs(interpolated_x, extrapolate=True)
    interpolated_y = validate_interpolation(interpolated_y, interpolated_x, x, y)

    #From apy to continuous compounding
    interpolated_y = np.log(1 + interpolated_y)
    #get the interpolated rates for each term date
    r = [interpolated_y[0][d] for d in days_diffs]
    return r


###### Main ######
# Dataframes for the indices
norvix_df = pd.DataFrame(columns=['Date', 'Index'])
danvix_df = pd.DataFrame(columns=['Date', 'Index'])
swevix_df = pd.DataFrame(columns=['Date', 'Index'])

#Start building the volatility index
for i, file in enumerate(files):
    date = datetime.datetime.strptime(file.split('/')[-1].split('_')[0], '%Y%m%d')

    #Import option chain
    total_df = pd.read_csv(file)
    total_df = clean_file(total_df)

    #Split into OMXO20, OMXC25 and OMXS30
    omxo20_df = total_df[total_df['Underlying'] == 'OMXO20']
    omxc25_df = total_df[total_df['Underlying'] == 'OMXC25']
    omxs30_df = total_df[total_df['Underlying'] == 'OMXS30']

    df_list = [omxo20_df, omxc25_df, omxs30_df]

    for j, df in enumerate(df_list):
        name = names[j]
        current_date = df['Date'].iloc[0].date()
        df = pivot_df(df)

        constant_maturity_date = current_date + pd.Timedelta(days=30)

        exp_dates = df['Exp date'].dt.date.unique()
        exp_dates = np.sort(exp_dates)

        #Find near term dates less than or equal constant maturity date
        near_term_dates = exp_dates[exp_dates <= constant_maturity_date]
        if len(near_term_dates) == 0:
            #No near term expiry dates found, using closest date instead
            near_term_date = exp_dates[0]
        else:
            near_term_date = near_term_dates[-1]
        #Find next term dates greater than constant maturity date
        next_term_dates = exp_dates[exp_dates > near_term_date]
        #find the first value greater than near term date
        next_term_date = next_term_dates[0]

        term_dates = [near_term_date, next_term_date]

        #days between current date and term dates
        days_diffs = [int((term - current_date).days) for term in term_dates]

        #Compute rates
        rates = compute_rates(rate_paths[j], current_date, days_diffs)

        minutes = [(term_date - current_date).total_seconds() // 60 for term_date in
    term_dates]
        times = [round(min / M365, 7) for min in minutes]

        ### Calculate the variance for the terms
        variance = []
        for j, term_date in enumerate(term_dates):
            term_df = df[df['Exp date'] == pd.Timestamp(term_date)]

            atm = term_df[term_df['Diff'] == term_df['Diff'].min()]
            atm = atm.sort_values(by=['Strike'])
            atm = atm.iloc[0]

            atm_strike = atm['Strike']
            atm_call = atm['Call']
            atm_put = atm['Put']

            #Calculate forward price
            f = atm_strike + np.exp(rates[j]*times[j])*(atm_call - atm_put)

            #find strike right below forward price
            strikes = term_df['Strike'].values
            k = max([s for s in strikes if s <= f], default=None)

            ####### Strike selection #######
            #sort by strike price
            term_df = term_df.sort_values(by=['Strike'])

            #select puts
            #start at K_0 and iterate to lower strikes for puts
            put = term_df[term_df['Strike'] < k]
            #sort descending such that we start at K_0 and move downwards in strikes
            put.sort_values(by=['Strike'], inplace=True, ascending=False)

            #set all to true for inclution
            put['Include'] = True

            previous_include = True #used to check if we've encountered two rows with Bid = 0
            for l, row in put.iterrows(): #iterate over rows in put
                if  pd.isna(row['Put']) or pd.isna(row['Put_Volume']): #if bid is 0, set inclution to false
                    put.at[l, 'Include'] = False

                # If we've encountered two rows with Bid = 0, break the loop
                if not previous_include and not put.at[l, 'Include']:
                    # all less than i are false
                    put.loc[l:, 'Include'] = False
                    break
                previous_include = put.at[l, 'Include'] #set previous_include to current include for next iteration

            #select calls, as with puts
            call = term_df[term_df['Strike'] > k]
            call.sort_values(by=['Strike'], inplace=True) #sort ascending such that we start at K_0 and move upwards in strikes

            call['Include'] = True
            previous_include = True
            for l, row in call.iterrows():
                if  pd.isna(row['Call']) or pd.isna(row['Call_Volume']):
                    call.at[l, 'Include'] = False

                # If we've encountered two rows with Bid = 0, break the loop
                if not previous_include and not call.at[l, 'Include']:
                    # all less than i are false, meaning all larger strikes are false
                    call.loc[l:, 'Include'] = False
                    #print(f'Breaking at {i}')
                    break
                previous_include = call.at[l, 'Include']

            # Combine the two dataframes in a new one
            put.sort_values(by=['Strike'], inplace=True) #sort ascending again for concat
            call = call[call['Include'] == True]
            put = put[put['Include'] == True]

            put['Option Type'] = 'Put'
            call['Option Type'] = 'Call'
            put = put.rename(columns={'Put': 'Midpoint Price'})
            call = call.rename(columns={'Call': 'Midpoint Price'})

            #Add mid where K_0 is, average of call and put
```

```python
        mid_row = term_df[term_df['Strike'] == k]
        mid_row['Option Type'] = 'Put/Call Average'
        mid_row['Midpoint Price'] = term_df[['Call', 'Put']].mean(axis=1)

        #Combine the dataframes to a final term dataframe
        term_df = pd.concat([put, mid_row, call])
        term_df.drop(columns=['Diff','Call_Volume', 'Call', 'Put', 'Put_Volume', 'Include'], inplace=True)

        ######### Delta K #########
        term_df['next'] = term_df['Strike'].shift(-1)
        term_df['prev'] = term_df['Strike'].shift(1)
        #delta_k = (K_i+1 - K_i-1) / 2
        term_df['delta_k'] = (term_df['next'] - term_df['prev']) / 2

        # Assign values for the first and last rows
        term_df.iloc[0, term_df.columns.get_loc('delta_k')] = term_df['next'].iloc[0] - term_df['Strike'].iloc[0]
        term_df.iloc[-1, term_df.columns.get_loc('delta_k')] = term_df['Strike'].iloc[-1] - term_df['prev'].iloc[-1]
        term_df.drop(columns=['next', 'prev'], inplace=True)

        ### Calculate variance by contribution ####
        term_df['contribution'] = (term_df['delta_k'] / (term_df['Strike']**2)) * term_df['Midpoint Price'] * np.exp(rates[j] * times[j])
        contribution_sum = term_df['contribution'].sum()

        var = round(contribution_sum*(2/times[j]), 10) - round((1/times[j]) * (f/k - 1)**2, 10)
        variance.append(var)

    ###### VIX ######
    denominator = minutes[1]-minutes[0]
    volatility_index = round(100 * np.sqrt((times[0]*variance[0]*((minutes[1]-MCM)/(denominator)) + times[1]*variance[1]*((MCM-minutes[0])/(denominator)))*(M365/MCM)), 2)

    #Save to df
    if name == 'OMXO20':
        print(f'{name} got an index of {volatility_index} for {current_date}')
        norvix_df = pd.concat([norvix_df, pd.DataFrame([[current_date, volatility_index]], columns=['Date', 'Index'])])
    elif name == 'OMXC25':
        print(f'{name} got an index of {volatility_index} for {current_date}')
        danvix_df = pd.concat([danvix_df, pd.DataFrame([[current_date, volatility_index]], columns=['Date', 'Index'])])
    elif name == 'OMXS30':
        print(f'{name} got an index of {volatility_index} for {current_date}')
        swevix_df = pd.concat([swevix_df, pd.DataFrame([[current_date,
    volatility_index]], columns=['Date', 'Index'])])

#Save to csv
norvix_df.sort_values(by=['Date'], inplace=True)
danvix_df.sort_values(by=['Date'], inplace=True)
swevix_df.sort_values(by=['Date'], inplace=True)
norvix_df.to_csv(omxo20, index=False)
danvix_df.to_csv(omxc25, index=False)
swevix_df.to_csv(omxs30, index=False)
```

Listing 4: Code for making the model-free volatility indices

B.4 EIKON data

```
# Importing necessary libraries
# installed via pip
import datetime as dt
import eikon as ek
import refinitiv.dataplatform as rdp
import pandas as pd
import time
import holidays

# Setting up Eikon API
# Eikon API keys are generated from
# the Eikon Desktop Application.
# Ensure that you have the Eikon Desktop
# App running before running this script.
your_data_folder = ".../Data/"
key = '6d1xxxxxxxxxxxxxxxxxxxxxxxxxxxxfed'
ek.set_app_key(key)
rdp.open_desktop_session(key)

##################
### INDEX DATA ###
##################
# Fetching index data from Eikon.
start_date = "2017-01-01"
end_date = "2023-11-11"
interval = "daily"

omxs30 = ek.get_timeseries([".OMXS30"],
                           start_date=start_date,
                           end_date=end_date,
                           interval=interval)
omxc25 = ek.get_timeseries([".OMXC25CAP"],
                           start_date=start_date,
                           end_date=end_date,
                           interval=interval)
omxo20 = ek.get_timeseries([".OBX"],
                           start_date=start_date,
                           end_date=end_date,
                           interval=interval)
omxn40 = ek.get_timeseries([".OMXN40"],
                           start_date=start_date,
                           end_date=end_date,
                           interval=interval)

# Save to CSV
omxs30.to_csv('OMXS30.csv')
```

```
omxc25.to_csv('OMXC25.csv')
omxo20.to_csv('OMXO20.csv')

# Processing data to get correct dates.
# OMXS30, OMXC25 and OMXO20 are rebalanced
# semi-annially.
dates = [dt.date(i, j, 1) for i in range(2018, 2024) for j in [1, 7]]

# Checking for weekends.
def is_weekend(date):
    return date.weekday() > 4  # 5 and 6 correspond to Saturday and Sunday

# Checking for holidays.
def is_public_holiday(date, country_code):
    if country_code not in ['SE', 'NO', 'DK']:
        raise ValueError(f"Country code {country_code} is not supported.")
    return date in holidays.CountryHoliday(country_code)

# Rolling forward to next trading day.
def next_trading_day(date, country_code=['SE', 'NO', 'DK']):
    while is_weekend(date) or is_public_holiday(date, country_code):
        date += dt.timedelta(days=1)
    return date

# Adjusting to nearest trading day
adjusted_dates_se = [next_trading_day(date, 'SE') for date in dates]
adjusted_dates_no = [next_trading_day(date, 'NO') for date in dates]
adjusted_dates_dk = [next_trading_day(date, 'DK') for date in dates]
# Controlling that the dates are the same for all countries
adjusted_dates_se == adjusted_dates_no == adjusted_dates_dk # True
# Saving one of them to general variable 'dates'
dates = adjusted_dates_se

#####################
## MARKET CAP DATA ##
#####################
# Function to constitute market cap for a given ric
# dates: list of dates
# ric: refinitiv idenitification code of the index
# (same you would use in Eikon)
def constituentMktCaps(dates, ric):
    all_data = pd.DataFrame()
    for date in dates:
        # Convert date to string
        date_str = date.strftime("%Y-%m-%d")

        # Get the constituents of the ric
        # TR. = Time Series Request
        df, err = ek.get_data(ric,
                            ['TR.IndexConstituentRIC',
                             'TR.IndexConstituentName'],
                            {'SDate': date_str})

        # Saving date to DataFrame
        df['Date'] = date_str

        # Check if df is not empty and 'Constituent RIC' column exists
        if not df.empty and 'Constituent RIC' in df.columns:
            # Fetch market cap for each company in 'Constituent RIC'
            market_cap_df, err = ek.get_data(
                instruments=df['Constituent RIC'].tolist(),
                fields=['TR.CompanyMarketCap',
                        'TR.CompanyMarketCap.Currency'],
                parameters={'SDate': date_str}
            )

            # Merge the market cap data with df
            df = df.merge(market_cap_df,
                          left_on='Constituent RIC',
                          right_on='Instrument',
                          how='left')

        # Append the results to the all_data DataFrame
        all_data = pd.concat([all_data, df],
                             ignore_index=True)

        # Delay for 1 second to avoid time-outs
        time.sleep(1)

    # Final formatting
    all_data = all_data[['Date',
                         'Instrument_x',
                         'Constituent RIC',
                         'Constituent Name',
                         'Company Market Cap',
                         'Currency']]
    all_data.columns = [ 'Date', 'Index', 'RIC',
                        'Name', 'Market Cap', 'Currency']

    # Returning data
    return all_data

# Utilising Function
# Fetching for Stockholm
dates = adjusted_dates_se
ric = '.OMXS30'
swe = constituentMktCaps(dates, ric)
# Fetching for Oslo
```

```python
dates = adjusted_dates_no
ric = '.OBX'
nor = constituentMktCaps(dates, ric)
# Fetching for Copenhagen
dates = adjusted_dates_dk
ric = '.OMXC25CAP'
den = constituentMktCaps(dates, ric)

# save all three to csv
swe.to_csv('OMXS30mktcap.csv')
nor.to_csv('OMXO20mktcap.csv')
den.to_csv('OMXC25mktcap.csv')

####################
## EXCHANGE RATES ##
####################
def quartRates(start_date, end_date, currency_pairs):

    # Fetch the historical exchange rates
    exchange_rates_ts = ek.get_timeseries(
        currency_pairs,
        fields='CLOSE',
        start_date=start_date,
        end_date=end_date,
        interval='quarterly'
    )

    # Reshape the DataFrame for easier processing
    exchange_rates_ts = exchange_rates_ts.reset_index()
    exchange_rates_ts = exchange_rates_ts.melt(id_vars='Date',
                                               var_name='Currency_Pair',
                                               value_name='Rate')
    # rename columns
    exchange_rates_ts.columns = ['Date', 'Currency', 'Rate']
    # inverse the Rates
    exchange_rates_ts['Rate'] = 1 / exchange_rates_ts['Rate'] # NOK/DKK/SEK to EUR
    # replace EURSEK= to SEKtoEUR, etc.
    exchange_rates_ts['Currency'] = exchange_rates_ts['Currency'].str.replace('DKK', '-
    DKK')
    exchange_rates_ts['Currency'] = exchange_rates_ts['Currency'].str.replace('SEK', '-
    SEK')
    exchange_rates_ts['Currency'] = exchange_rates_ts['Currency'].str.replace('NOK', '-
    NOK')
    exchange_rates_ts['Currency'] = exchange_rates_ts['Currency'].str.replace('=', '')

    # Return the DataFrame
    return exchange_rates_ts

# Utilising Function
```

```python
currency_pairs = ['EURSEK=', 'EURNOK=', 'EURDKK=']
start_date = "2017-06-01"
end_date = "2023-12-31"
exchange_rates_ts = quartRates(start_date, end_date, currency_pairs)

# save to csv
exchange_rates_ts.to_csv('ExchangeRates.csv')
```

Listing 5: Fetching data from Eikon Refinitiv