A Work Project, presented as part of the requirements for the Award of a Master's degree in Finance from the Nova School of Business and Economics.

FORECASTING STOCK INDEX VOLATILITY – A COMPARISON OF MODELS

MARION VAN GEMST - 31342

Work project carried out under the supervision of:

André Castro Silva

03-01-2020

Forecasting Stock Index Volatility – A comparison of Models

Abstract

This thesis explores the use of popular machine learning algorithms (K-Nearest Neighbor and Random Forest) and compares them to traditional techniques (Random Walk, ARIMA and GARCH) for forecasting one-day, one-week, one-month and one-quarter volatility using The Oslo Stock Exchange All Share Index. A number of error metrics are applied (RMSE, MAE, MAPE and R-squared) in order to compare their results. Machine learning methods are shown to forecast the changes in volatility to some extent, however, evidence is found favouring the ARIMA model when forecasting volatility time series.

Keywords: Volatility, Forecasting, GARCH, Machine Learning

This work used infrastructure and resources funded by Fundação para a Ciência e a Tecnologia (UID/ECO/00124/2013, UID/ECO/00124/2019 and Social Sciences DataLab, Project 22209), POR Lisboa (LISBOA-01-0145-FEDER-007722 and Social Sciences DataLab, Project 22209) and POR Norte (Social Sciences DataLab, Project 22209).

Introduction

Forecasting volatility is important in many areas of finance and is therefore the subject of a vast amount of research. In this thesis, a comparison between different techniques for forecasting stock volatility is conducted. Volatility in financial markets refers to how variable the price of a security or index is over a period of time. It is measured by the standard deviation or variance of the underlying asset's returns, or logarithmic returns. There are many reasons for why volatility and volatility forecasting are of great interest. Firstly, volatility is a common measure of risk and an important variable in finance, for example for VaR-calculation in risk management, and asset allocation in portfolio management. Additionally, option pricing techniques rely on a volatility parameter which is, unlike other variables in theoretical pricing formulas, not directly observed in the market. Hence, volatility is the most difficult and uncertain part of the valuation and affects the option prices significantly. Furthermore, we are often concerned about future volatility levels, which makes forecasting a necessity. However, due to the variable's specific characteristics and complex external influences, it is challenging to accurately forecast volatility from historical values without large errors. Thus, minimising the error in the forecast proves to be crucial.

Popular techniques for data analysis, forecasting, classification and regression are provided by machine learning. This is because of their ability to learn from an input and predict an output given a correlation between the two variables. It is therefore interesting to explore the predictive abilities of machine learning algorithms, when used on volatility time series.
The techniques can be divided into supervised and unsupervised learning. Supervised learning is where the algorithm learns about the relationships of variables from a training set in order to produce an output. Unsupervised learning refers to when there are no output variables, and the algorithm is used to find previously unknown patters in the data in order to learn more

about it. Since supervised learning fits better with the forecasting tasks in the present research, unsupervised learning will be excluded. Furthermore, forecasting is considered a regression task, hence regression algorithms will be the focus of this study.

The aim of this thesis is to compare the accuracy of predictions made by traditional volatility forecasting models found in literature, to some of the most popular machine learning methods. Since there is little to no research concerning the volatility of the Norwegian stock market, 10 years of daily closing prices from The Oslo Stock Exchange All Share Index (OSEAX) are used. The models implemented are Random Walk, ARIMA, GARCH, K-Nearest Neighbor and Random Forests algorithms. In order to find evidence for whether the same model provides the most accurate forecasts over different timeframes, the one-day, one-week and one-month volatilities are forecasted. All forecasts are evaluated by their out-of-sample predictive power, which is measured by the differences between the actual observed values in the test dataset and the predicted values. The error metrics used are Mean Absolute Percentage Error (MAPE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R squared.

The remainder of this thesis consist of a literature review, an explanation of methodology and a presentation of results. Subsequently, a discussion about the results is conducted, before arriving at a conclusion with proposed further research. All code used to create, fit and test the models is written in Python, and can be found in Appendix A.

Literature Review

Due to the importance of volatility in finance and the challenges of forecasting it, volatility has been the subject of a large body of research for the past three decades. Research shows that volatility time series have a number of characteristics differentiating them from other time series, for example that the distribution has fat tails and that stock shocks have a strong impact on volatility. There are two characteristics particularly relevant for the research conducted in this thesis. Firstly, there is a great amount of evidence of volatility clustering, which mean that a high volatility period tends to be followed by another period of high volatility and similarly a low volatility period tends to be followed by one of low volatility. Mandelbrot (1963) and Fama (1965) are among many studies providing empirical evidence for this. Furthermore, Bollerslev, Chou and Kroner (1992), Bollerslev, Engle and Nelson (1994) and Shephard (1996) support the existence of volatility persistence in financial time series meaning that the volatility in many periods in the future are affected by the stock return today.

Attempting to take different stylized facts into consideration, a large number of models have been developed and implemented to solve the task of forecasting volatility. Moving on from simple historical volatility models and linear regression, some of the biggest contributions to the field are The Autoregressive Moving Average Model (Whittle, 1951), its extension ARIMA, as well as the introduction of ARCH models by Robert. F. Engle (1982) leading to the development of the generalised ARCH model, GARCH (Bollerslev, 1986). In the following years, several extensions to the ARCH models have been added to the literature in an attempt to include more of the volatility characteristics. An example of such a model is EGARCH proposed by Nelson (1991) which argued that the nonnegativity constraints in the linear GARCH model are too restrictive. Moreover, TGARCH (Zakoian, 1994) was

introduced to take leverage effects into account, which refers to the tendency of an asset's volatility to be negatively correlated with the asset's return. There are many papers comparing the forecasting accuracy of historical volatility models, time series techniques, stochastic volatility models and autoregressive models with conflicting conclusions. In the majority of cases, the conflicting evidence arises from different evaluation metrics, forecasting time-frames, measures of volatility and asset types. See for instance the comparison by Poon and Granger (2003).

Machine learning is not a new phenomenon. The first algorithms can be dated back to as early as the 1950s. Despite this, it was not until the 21st century that the use of such algorithms exploded and the techniques started to become common in most industries. Economics and finance were slower to implement the models (Athey and Imbens, 2019). Econometric methods are, however, currently being challenged by machine learning which is being increasingly included in the research area.

There is evidence in the existing literature demonstrating the ability of machine learning to perform well in financial prediction tasks. Most of the research concerns asset pricing and stock predicting. Gu, Kelly and Xiu (2019), for example, find evidence suggesting that machine learning methods, especially neural networks and regression trees, can help improve empirical understanding of asset prices, and is most valuable in forecasting larger and more liquid stock returns and portfolios. Furthermore, Alkhatib, Najadat, Hmeidi, Shatnawi (2013) applied the K-Nearest Neighbor (k-NN) algorithm and non-linear regression to predict stock prices. They found evidence suggesting that the algorithm produced reasonable forecasts with small errors. There is, however, existing research concerning machine learning in volatility forecasting as well. For example, Luong and Dokuchaev (2018) combines the Heterogenous

Autoregressive Model (HAR) and the Random Forest algorithm, in order to forecast the direction and magnitude of the realised volatility. They conclude that the HAR model framework was improved by the algorithm. In addition to this, Zhang and Li applies a modified version of the k-Nearest Neighbor algorithm to forecast volatility of the Tsingtao Brewery Co Ltd stock prices and find that the method predicts better than traditional methods.

Taking the difficulty of volatility forecasting, machine learning theory and existing literature into consideration, it is reasonable to believe that with a comparison of traditional forecasting techniques to some of the most popular machine learning regression algorithms, the inclusion of machine learning models have the potential to outperform the other techniques.

Methodology

Dataset

The dataset used is 10 years of daily prices from The Oslo Stock Exchange All Share Index starting on 01.10.2009 and ending on 01.10.2019, resulting in 2510 values, retrieved from Bloomberg. Volatility is measured by standard deviation and calculated by

$$\sigma_i = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(r_t - \bar{r})^2}, \quad (1)$$

where $\bar{r}$ is the average of the daily logarithmic return which is on day t calculated by

$$r_t = \ln(P_t) - \ln(P_{t-1}), \quad (2)$$

with P_t being the security closing price on day t and P_{t-1} the security opening price on day t. After calculating returns and standard deviations, the data consists of 2509 and 2508 dates and their respective values. In order to make forecasting possible for the supervised machine learning models, a sliding window procedure is performed on the dataset creating an independent and a dependent value from the time series. Furthermore, the dataset is divided into training and testing sets, where the models are fitted using the training set and their out-of-sample predictive power evaluated with the testing set. The splitting between training and testing sets is done with respect to different forecasting horizons, with the aim of determining whether the same models perform well on short and semi-long forecasting horizons. The time frames used are one-day, one-week and one-month, given by series of 1, 5, 21 and 63 values respectively, which represent the number of trading days in the period. The data is plotted for a visual illustration.

Figure 1: Daily closing price OSEAX

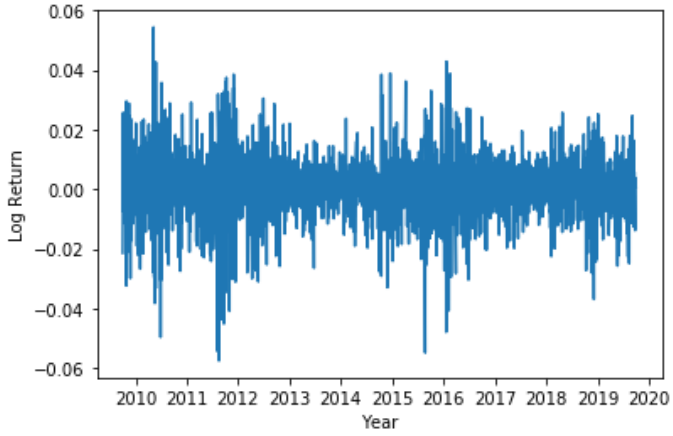

Figure 2: Daily logarithmic returns of OSEAX

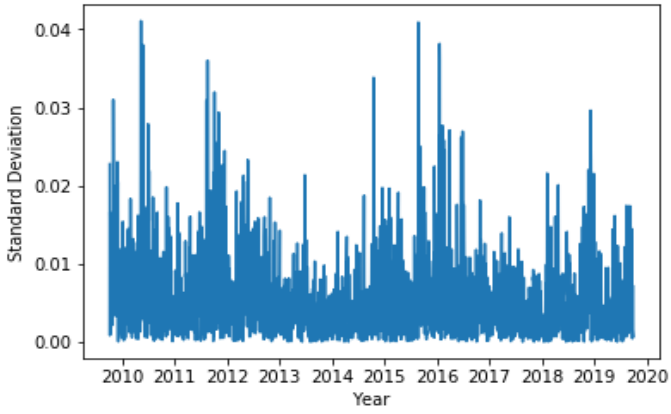

Figure 3: Daily standard deviation of OSEAX return

The visual illustration of daily standard deviation (Figure 3) indicates that volatility clustering is present in the dataset. Furthermore, despite periods of extreme volatility, the series appears to be mean reverting meaning that it tends to return to its average levels after fluctuating.

Stationarity

Stationarity refers to the situation where the statistical properties, such as mean and variance, do not change over time, and is a common requirement for time series modelling. The first step in order to determine if the time series is stationary, is to inspect the plotted data for time dependent structures such as trend or seasonality. The daily closing prices (Figure 1) show a clear increasing trend. However, the plotted daily returns (Figure 2) and standard deviation of daily returns (Figure 3), display no such pattern. An Augmented Dickey-Fuller test is conducted in order to formally confirm the following hypothesis:

H0: The time series is non-stationary.

H1: The time series is stationary.

The result from the test contains a test statistic, a critical value for different confidence levels and a p-value. The p-value needs to be smaller than the significance level of 0.05 for the null hypothesis to be rejected and the time series to be assumed stationary. When conducting the test on the return and standard deviation datasets, H0 is rejected since the p-value is smaller than 0.05, and the test statistics less than the values for each confidence level. Hence, both time series are assumed to be stationary, with small probability of the result being false.

Traditional Models

Random Walk (RW)

A random walk is a mathematical process which describes a path where the directionto each next step is a random step away from the current observation, within a set of predefined directions. In the present research a two-dimensional random walk is conducted, where the possible directions to move at each step are up and down, with equal probability to move in

either direction. Furthermore, the number of steps in the walk is equal to the forecast horizon. The random walk is mathematically given by the sequence

$$(W_t) = (S_t)_{t=1}^{T}, \quad (3)$$

where $S_t = \sum_{k=1}^{t} X_k$ is the value at time step t in the walk, and X_k the random variable at each time step. Hence, T is the forecasting horizon. Let the initial value S_0 be set to the current level of volatility.

The reason for including such a simple model is that there is a large amount of randomness in volatility time series, and a random walk can provide understanding of whether the time series is predictable. Furthermore, it is interesting to see whether the far more complex techniques provide more accurate predictions to a large degree. The magnitude of the movement at each time step is chosen to be 0.0058 for scaling purposes, which is the average daily volatility in the dataset.

Autoregressive Integrated Moving Average (ARIMA)

The ARIMA model creates a linear equation to forecast a future series based on past values, where lags and the lagged forecast errors are taken into account. It can be considered as a combination of simpler models. The Autoregressive term (AR) refers to how the values at different time steps are autocorrelated. Autocorrelation, or serial correlation, is a term used to describe the situation where the value in a timeseries is correlated with the values of previous steps, meaning the model can use the current values to forecast future ones. Moving Average (MA) refers to the fact that the time series does not have a constant average, and the model therefore sets different averages along the dataset. Further, the ARIMA equation requires the time series to be stationary. If the time series is not stationary, the series must be differenced

by replacing the values with the change from the preceding period, which is represented by the "Integrated" part of ARIMA. This is given by

No difference (d = 0): $$y_t = Y_t, \quad (4)$$

First difference (d=1): $$y_t = Y_t - Y_{t-1}. \quad (5)$$

Y is the original series and y the differentiated (stationary) series defined by

$$Y = (Y_1, \dots, Y_T), \qquad y = (y_1, \dots, y_T), \quad (6)$$

where

$$Y_t = ß_0 + ß_1 Y_{t-1} + \cdots + ß_p Y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_q \varepsilon_{t-q}, \quad (7)$$

with $ß_0$ a constant, $ß_i Y_{t-i} + \varepsilon_t$ representing the autoregressive terms and $\theta_i \varepsilon_{t-i}$ representing the moving average terms. A generally accepted notation of the model is ARIMA(p,d,q) where parameters p (AR terms), d (order of differencing) and q (MA terms) areto be determined. The values of p and q are identified by analysing the Autocorrelation Function (ACF) plot and Partial Autocorrelation function (PACF) plot combined with significance levels. The ACF plot illustrates the number of lags where values in the series are autocorrelated.

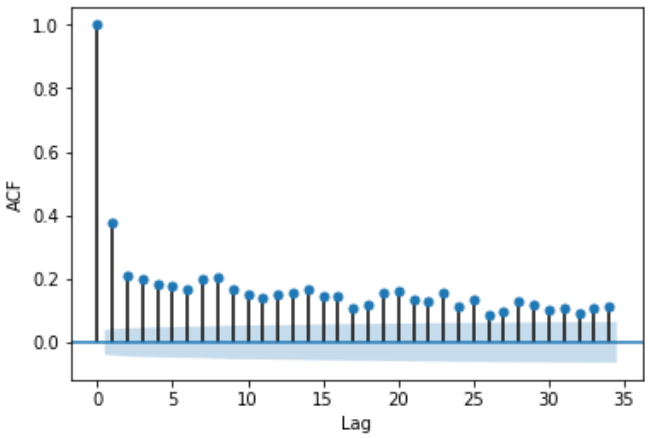

Figure 4: ACF plot for standard deviation

Figure 4 reveals that the series has significant positive autocorrelations up to a high number of lags. Despite the fact that the stationarity test concludes with a stationary time series, it appears here to be under-differenced, which is further confirmed by analysing the residuals of ARIMA(0,0,0) and ARIMA(0,1,0). Taking the first difference of the series makes the models' residuals closer to normally distributed and fluctuating around a constant mean. Thus, differencing the time series once results in the following ACF plot (Figure 5), which shows the correlation between the series and lag after contributions from previous lags are excluded.

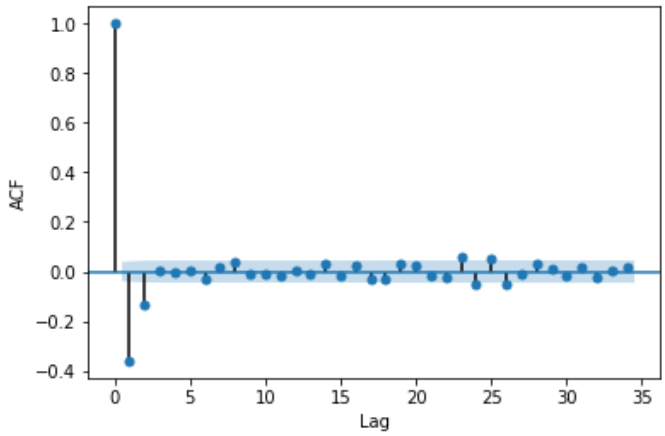

Figure 5: ACF plot of differenced series

The first difference, d = 1, does not result in negative autocorrelation on the first lag which indicates that the time series is not over-differenced. In order to identify the AR and MA terms, the PACF plot (Figure 6) is also analysed.

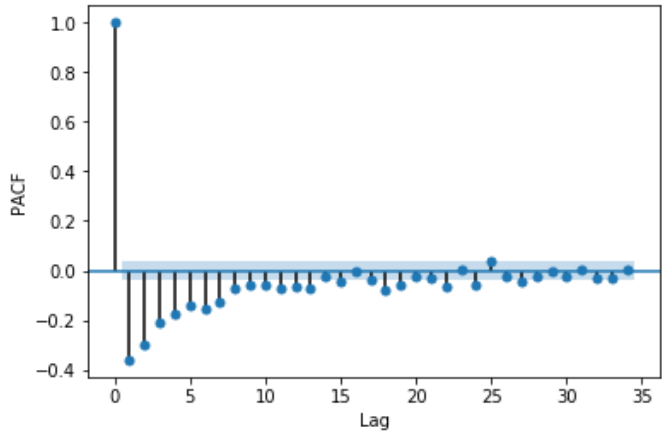

Figure 6: PACF plot of differenced series

Figure 6 shows a sharp decline of partial autocorrelation after the first lag. This, combined with the positive first lag of autocorrelation, indicates that the model is slightly under-differenced. An AR term is therefore added to the model. The order of AR terms is determined by the lag where PACF is within the accepted values for the first time. Hence, the model used is ARIMA(1,1,0).

Generalised Autoregressive Conditional Heteroskedasticity (GARCH)

One problem with autoregressive models such as ARIMA, is that they do not take changes in variance over time into account, which is usually a characteristic of stock market volatility. GARCH is a generalisation of The Autoregressive Conditional Heteroskedasticity Model (ARCH), which is a non-linear model attempting to model the error of the change of variance in the time series based on previous lags and their errors. Furthermore, the model recognises the difference between conditional and unconditional variance. Unconditional variance is time varying, whereas the conditional is not. It creates a function with weighted averages of squared past forecast errors and uses this to allow the conditional variance to change over time. ARCH is therefore creating a "weighted variance" meaning that the recent values are given a greater weight than the ones further in the past. In order to describe the model mathematically, let ε_t denote the unexpected returns of the model. The error terms are split into a stochastic part z_t, which is a white-noise process, and a time-dependent standard deviation σ_t. Thus, the error term is defined by

$$\varepsilon_t = \sigma_t z_t. \tag{8}$$

Since the current value of variance of errors in the model depends on the previous squared error terms, the ARCH(p) model can be described as variance of the series, σ_t^2, and is modelled by

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2, \quad (9)$$

where σ_t^2 is the current variance of errors, α_0 a positive constant, $a_i \geq 0$ and ε_{t-i}^2 represents the squared errors for the period t-i. It is common practice to write the model as ARCH(p) where p denotes the number of included. An LM test for ARCH effect was conducted with the indication that ARCH effects are present in the time series.

GARCH includes a moving average element which makes it possible to model both the conditional change in variance as well as changes in the time-dependent variance, with the aim of capturing more of the variance in volatility. A common notation for the model is GARCH(p,q). Following the notation from the ARCH section, let p be the order of ARCH terms (ε^2) and q the order of GARCH terms (σ^2). The GARCH(p,q) model is then defined by

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{i=1}^{q} \beta_i \sigma_{t-i}^2, \quad (10)$$

where $\beta_i \geq 0$. This makes GARCH (0,q) equivalent to an ARCH model, while GARCH (0,0) is simply white noise. GARCH assumes, like ARCH, that the time series is stationary apart from the change in variance. In order to determine the ARCH and GARCH term, ACF and PACF plots of the squared returns are inspected.

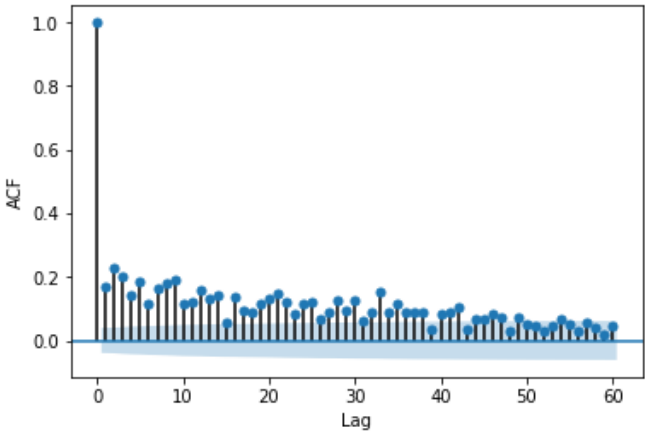

Figure 7: ACF plot of squared returns

The ACF plot shows significant positive autocorrelation for multiple lags and crosses the upper confidence level at a value of 49 resulting in choosing this as the order of ARCH terms. The GARCH effect can be found by inspecting the PACF plot of the squared returns.

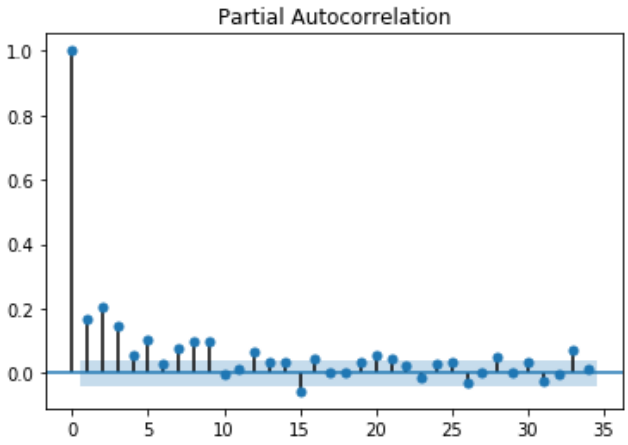

Figure 8: PACF plot of squared returns

The plot (Figure 8) displays positive significant partial autocorrelation for up to 9 lags, resulting in setting q = 9 in the model leading to a GARCH(49,9) model.

Machine Learning Methods

K-Nearest Neighbor (k-NN)

k-NN is a simple algorithm to use and is commonly used in different regression tasks. The model is non-parametric meaning it does not make any assumptions about the underlying data. The rationale behind the algorithm is that it assumes that similar things are close to each other. It works by calculating the distances between the values in the dataset with the chosen distance function, considering what the closest value is to the one it is trying to predict and takes the average of the closest points in order to do so. A value of K, which is the number of near data points to include in the average, must be chosen. Increasing K makes the model more stable but will at some point result in bigger errors. Decreasing K below this point will result in a more unstable model, making the predictions less accurate when tested out-of-

sample. The parameter is chosen by fitting the model with different values of K and selecting the one that minimises the errors on the training set. In this research, up to K=300 is tested, where K=39 is the value for which the model produces the smallest errors.

Random Forest (RF)

The Random Forest algorithm is built up by many decision trees. Thus, in order to understand The Random Forest, it is critical to first understand how a decision tree works. A decision tree uses a tree-like structure to make decisions. It starts with a root and divides the dataset following certain criteria suited for the dataset. At each node a new rule is introduced, and the values will either follow this rule or not. This continues until there are no more conditions to be met and the leaf nodes are reached, which are the target variables. The depth of a tree refers to how many criteria are included and will vary depending on the dataset. Finally, the decision tree can use the same rules created on a training set to predict out-of-sample values.

Decision trees are easy to understand and use, and are fast even when implemented on large datasets. However, there is a possibility of overfitting, especially if the tree is very deep. One way of tackling this is to set a maximum depth of the tree, but this will make the model a worse fit for the data which creates bigger errors. Basing the decision on several trees (creating a "forest") will reduce the possibility of overfitting, and not give less accurate forecasts. Each of the trees in a random forest is individually a worse predictor than when a single decision tree is used. However, if enough trees are included, combined they will be more robust and produce better predictions than a simple decision tree. The individual trees are trained on different samples with random features in order to make them less correlated. More trees will lead to a better model, but the algorithm is very slow when a large number of trees are included. Furthermore, the added value of each tree will decrease and at some point be close to zero. The number of trees included in the model, N, is found by running the

algorithm with different values of N and evaluating when the added value of one additional tree declines. Here, the process results in N=3.

Evaluation

In order to evaluate and compare the models' predictive power, several error metrics have been implemented. These are evaluation techniques that measure the difference between the forecasted series and test dataset. Different evaluation techniques are used to get a clearer picture of the model's performance. The following error functions are included in the evaluation, where y represents the observed values, $\hat{y}$ the forecasted values and n the number of values.

Mean Absolute Percentage Error (MAPE)

MAPE presents the accuracy of the forecast as a percentage. It can be calculated as the averaged absolute value of the difference between the observed and forecasted values, divided by the observed value. The percentage errors are summed without the consideration of positive/negative percentages, eliminating the problem of the errors cancelling each other out. The metric works best when there are no extreme outliers or zeros. It is easier to interpret when different measures of volatility are used because it provides the error in percentage form, unlike the other error functions. Naturally, the smaller the percentage the better. Mathematically it can be calculated by

$$MAPE = \left(\frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i-\hat{y}_i}{y_i}\right|\right). \quad (11)$$

Mean Squared Error (MSE)

MSE is the average of squared differences between the predicted values and the test dataset. It is always positive, and the smaller the errors, the better. It measures both how widely spread

the forecasted values are and how close they are to the observed values. The error function is mathematically described as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \quad (12)$$

Root Mean Squared Error (RMSE)

RMSE is the square root of the MSE. The metric gives relatively large weight to large errors because it squares the errors before calculating the average. The measurement is negatively oriented meaning that lower scores are preferred. It can be defined as

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}. \quad (13)$$

R-Squared (R^2)

The problem with the so far described error functions is that it can be challenging to know what values are acceptable for a model. The coefficient of determination, R^2, is therefore included as an error metric. It evaluates how well the regression line created by the model fits the data and can be considered a ratio. It measures how good the fitted model is compared to the simplest model possible. The R-Squared formula is given by

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2}, \quad (14)$$

where $\bar{y}_i$ is the predicted value of a non-fitted model. This means that the error compares the sum of error squares for the model (regression line) to the total sum of squares for a non-fitted model. Thus, R^2 is a measure of how the best fitted line from the model follows the forecast, or how much of the forecast can be explained by the model. Its advantage is that it will always have a value ranging between negative infinity and one, and is therefore easy to interpret. Negative values reveal that the model fits the data worse than a simple horizontal line.

Results

In order to visually analyse the model's predictive power, their one-month forecast is plotted against the actual observed volatility in the period.

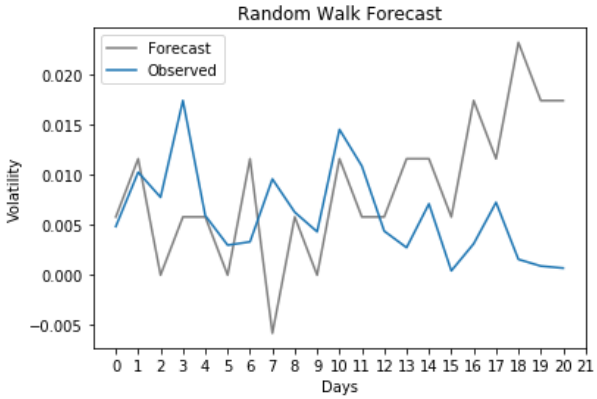

Figure 9: Random walk forecast

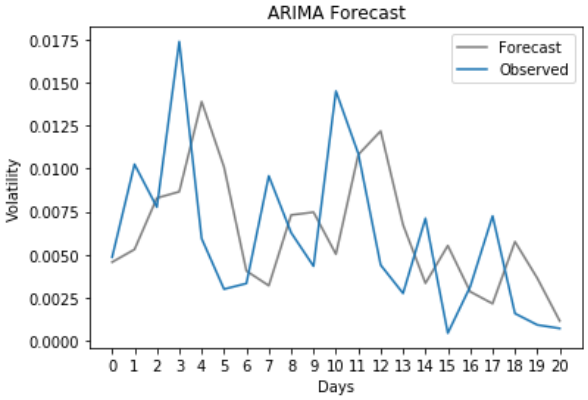

Figure 10: ARIMA Forecast

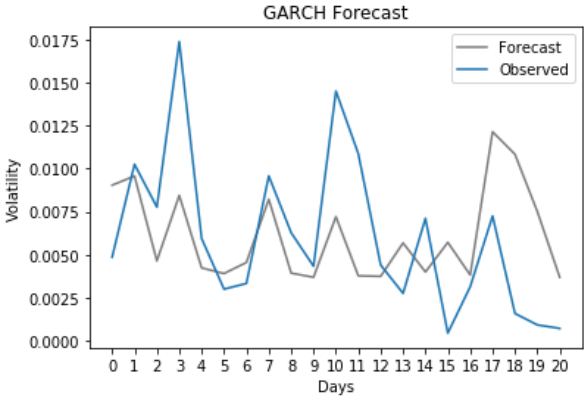

Figure 11: GARCH Forecast

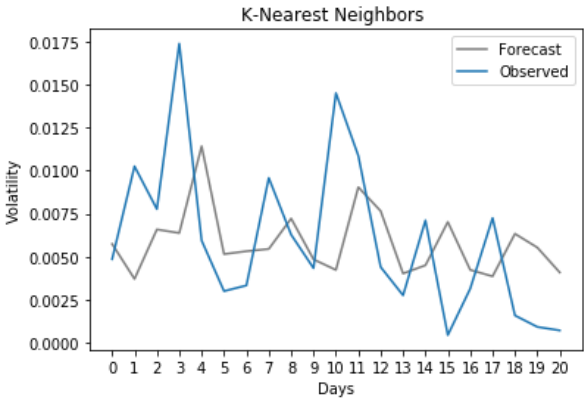

Figure 12: k-NN forecast

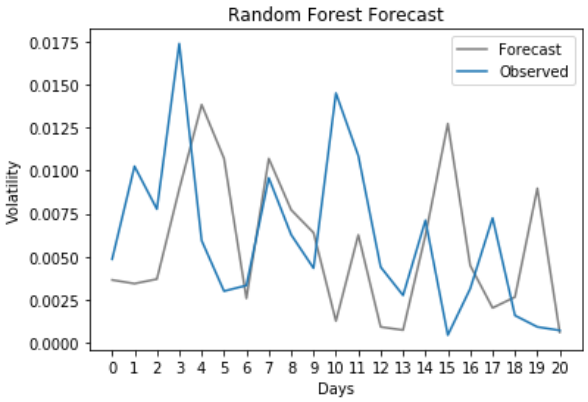

Figure 13: RF Forecast

The following table shows the results of the error metrics. Note that the function for R-squared fails when only one prediction is made by the models. This is shown as nan in Table 1.

	Horizon	RMSE	MAE	MAPE	R^2
RW	Day	0.00510	0.00510	729.07300	nan
	Week	0.00396	0.00372	303.48195	-1.67467
	Month	0.00960	0.00737	413.04350	-3.72282
	Quarter	0.02081	0.01582	973.27802	-26.7539
ARIMA	Day	0.00043	0.00043	62.28832	nan
	Week	0.00320	0.00254	158.46472	-0.74335
	Month	0.00501	0.00399	199.64139	-0.28388
	Quarter	0.00442	0.00347	352.14641	-0.25431
GARCH	Day	0.00832	0.00832	1189.6266	nan
	Week	0.00487	0.00446	350.78178	-3.03820
	Month	0.00454	0.00361	177.57392	-0.05891
	Quarter	0.00449	0.00350	268.31458	-0.29240
k-NN	Day	0.00337	0.00338	482.60138	nan
	Week	0.00368	0.00344	275.78895	-1.31527
	Month	0.00470	0.00371	172.59254	-0.13223
	Quarter	0.00416	0.00325	340.18410	-0.05669
RF	Day	0.00010	0.00012	15.26155	nan
	Week	0.00437	0.00316	218.63676	-2.25156
	Month	0.00589	0.00447	236.55045	-0.77757
	Quarter	0.00565	0.00457	467.05198	-1.05172

Table 1: Error metrics represented for each model and the respective time frame.

The errors in Table 1 reveal large differences in the forecasting accuracy across the models, and between different forecast horizons with the same model. One of the most noticeable trends in Table 1 is that all models have poor R-squared values. This means that they do not explain the changes in the dataset well and can be considered an illustration of the challenging nature of volatility forecasting. However, when compared to the graphs of the fitted models, the plotted Random Walk (Figure 9) illustrates their predictive power. While the Random Walk forecast can end up far from actual volatility levels for long time horizons, the other

models are able to capture the mean-reverting tendency of volatility. Despite a low R-squared indicating an inability among the trained models to explain a large amount of the changing values in the time series, the graphs display that certain models are able to model the series fairly well.

The models are fitted with daily volatility. One could therefore expect that the shorter horizons would have the lowest error rate. The error metrics reveal that this is true for ARIMA and RF who both produce one-day forecasts with small errors. However, k-NN and GARCH forecast one-month volatility most accurately. It is reasonable to believe that the mean-reverting characteristics of volatility and changes in the underlying provide difficulties when forecasting longer horizons with daily volatility, which to some degree can be seen in Table 1.

With regards to the most accurate forecasts, Figures 10, 12 and 13 show that ARIMA, k-NN and RF are able to capture the direction of changes in volatility. ARIMA is, on average, the model that best predicts the levels at all points in time. However, the model predicts the changes to occur at a slightly later time compared to the real data, which appears to be a trend among the models in general. When taking the error metrics into consideration, it is also the aforementioned models which produce, on average, the forecasts with the smallest errors. Furthermore, both Random Forest and ARIMA have small one-day forecast errors. This is a surprising result given the large differences in complexity between the two models.

It is interesting to inspect the differences in the forecasts produced by the machine learning methods. They are both able to predict the general structure of the series, however k-NN tends to underestimate the magnitude of the changes in volatility. Taking the construction of the model into consideration, this is not a surprising result. Random Forest, on the other hand,

switches between overestimating and underestimating the variability of volatility, and produces larger individual errors.

The plotted GARCH forecast (Figure 11) shows some similarities to that of ARIMA in the sense that it underestimates the level of changes in volatility. However, GARCH greatly overestimates the volatility towards the end of the forecast horizon. Given the development of the forecast, this could be due to the model putting more weight on recent observations. Furthermore, when evaluating the errors of the model, it is necessary to consider metrics other than MAPE, given its emphasis on large individual errors which becomes evident in the plotted forecast. It is seen that the errors are on the average level for long forecast horizons, but unreasonably large short-term. Based on the model's popularity in the field, one would expect more accurate predictions. However, it must be mentioned that with longer a longer forecasting horizon, GARCH could outperform the other models given its attention to conditional and unconditional changes in variance. Nevertheless, the models' poor performance indicates that different parameters, a combination with other models or another version of ARCH-family or would have been more appropriate to use.

Conclusion

Forecasting volatility is a challenging task. Due to the specific characteristics of volatility and its large implications in financial markets, a vast number of models have been created with the aim of producing more accurate forecasts. Some of them are compared in the present research.

Machine Learning was included due to its applicability to complex regression tasks and successful implementations in the field's existing literature, when combined with other models. This is also evident in the present research, even when the models are used alone. Although the algorithms have low errors in most forecast horizons and are able to capture the

general structure of the series, it is shown that they do not outperform the traditional, and much simpler, model ARIMA. This could be due to the fact that the algorithms are not specialised for volatility. If one were to take characteristics of volatility time series into consideration, they would be likely to produce improved forecasts. A limitation with machine learning algorithms is that they are trained to find relationships between values in a series and will do so even in cases where there are none, potentially resulting in wrongful forecasts.

The accuracy of the ARIMA(1,1,0) model is somewhat surprising. Despite being a relatively non-complex model compared to the other techniques, it produces the most accurate forecasts in the majority of the time horizons. This illustrates the fact that the accuracy of a forecast does not necessarily increase with the complexity of a model. One of the advantages to ARIMA is that it is flexible. It can be fitted to the data to determine the orders of parameters which lower the errors. However, a disadvantage with time series models compared to machine learning algorithms, is that they assume more about the input data.

If machine learning is specialised to volatility and combined with time series type models, the resulting forecasts would likely be of greater accuracy than their individual predictions included in the present thesis. This is suggested as further research, in addition to exploring other models in the ARCH family.

References

Alkhatib, Khalid; Najadat, Hassan; Hmeidi, Ismail; Shatnawi, Mohamme K. Ali. 2013. "Stock Price Prediction Using K-Nearest Neighbor (kNN) Algorithm." *International Journal of Business, Humanities and Technology*, 3(3): 32-44

Athey, Susan; Imbens, Guido W. 2019. "Machine Learning Methods that Economists Should Know About." *Annual Review of Economics*, 11: 685-725.

Bollerslev, Tim. 1986. "Generalized autoregressive conditional heteroskedasticity." *Journal of Econometrics*, 31(3): 307-327.

Bollerslev, Tim; Engle, Robert F. Nelson, Daniel B. 1994. "Arch models." In *Handbook of conometrics*, ed. Robert F Engle and Daniel L. McFadden, 2959-3038. Amsterdam: North Holland.

Bollerslev, Tim; Chou, Ray Y; Kroner, Kenneth F. 1992. «ARCH modeling in finance: A review of the theory and empirical evidence." *Journal of Econometrics*, 52(1-2): 5-59.

Engle, Robert F. 1982. "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation." *Econometrica*, 50(4): 987-1007.

Fama, Eugene F. 1965. "The Behavior of Stock-Market Prices." *The Journal of Business*, 38(1): 34-105.

Gu, Shihao; Kelly, Bryan T; Xiu; Dacheng. 2018. "Empirical Asset Pricing via Machine Learning." *Chicago Booth Research Paper*, 18(04).

Luong, Chuong; Dokuchaev, Nikolai. 2018. "Forecasting of Realised Volatility with the Random Forests Algorithm." *Journal of Risk and Financial Management*, 11(4): 1-15

Mandelbrot, Benoit. 1963. “The Variation of Certain Speculative Prices.” *The Journal of Business*, 36(4): 394-419.

Nelson, Daniel B. 1991. “Conditional Heteroskedasticity in Asset Returns: A new approach.”, *Econometrica*, 59(2): 347-370.

Poon, Ser-Huang; Granger, Clive W.J. 2003. “Forecasting Volatility in Financial Markets: A Review.” *Journal of Economic Literature*, 41(2): 478-539.

Samsudin, Ruhaidah; Shabri, Ani; Saad, Phil. 2010. “A Comparison of Time Series Forecasting using Support Vector Machine and Artificial Neural Network Model.” *Journal of Applied Sciences*, 10(11): 950-958.

Whittle, Peter. 1951. *Hypothesis Testing in Time Series Analysis.* Uppsala: Almqvist & Wiksells Boktryckeri.

Zakoian, Jean-Michel. 1994. “Threshold heterskedastic models.” *Journal of Economic Dynamics and Control*, 18(5): 931-955.

Zhang, Hong; Shu Fang Li. “Forecasting Volatility in Financial Markets.” *Key Engineering Materials*, 439(440): 679-82

Appendix

APPENDIX A: Code used to create and test the models

```
# Plotting prices, returns and standard deviation

import pandas as pd
from matplotlib import pyplot
from statsmodels.tsa.stattools import adfuller

df = pd.read_excel("/Users/mariongemst/Desktop/DATASETS OSEAX /OSEAX
MAIN.xlsx")
df = df.dropna()
returns = df["RETURN"]
stdev = df["STDEV"]
prices = df["PX_LAST"]

pyplot.plot(prices)
pyplot.xlabel("Year")
pyplot.ylabel('Closing Price')
pyplot.show()

pyplot.xlabel("Year")
pyplot.ylabel('Log Return')
pyplot.plot(returns)
pyplot.show()

pyplot.plot(stdev)
pyplot.xlabel("Year")
pyplot.ylabel('Standard Deviation')
pyplot.show()

#Testing stationarity
print("Restults of Stationarity Test:")
test = adfuller(returns, autolag="AIC")
print("ADF Statistic: %f" % test[0])
print("p-value: %f" %test[1])
print("Critical Values:")
for key, value in test[4].items():
    print('\t%s: %.3f' % (key, value))

print("Restults of Stationarity Test:")
test = adfuller(stdev)
print("ADF Statistic: %f" % test[0])
print("p-value: %f" %test[1])
print("Critical Values:")
for key, value in test[4].items():
    print('\t%s: %.3f' % (key, value))
```

```
#RANDOM WALK MODEL

from random import seed
from random import random
from matplotlib import pyplot
import pandas as pd
from sklearn import metrics
import numpy as np

df = pd.read_excel('/Users/mariongemst/Desktop/DATASETS OSEAX /OSEAX
MAIN.xlsx')
stdev = df["STDEV"]
stdev = stdev.values

n_test = 5                    #Number changes with forecasting horizon
test= stdev[-n_test:]         #Making the test set of observed values


#Creating the forecast
seed(0)
forecast = list()
forecast.append(-0.0058 if random() < 0.5 else 0.0058)
for i in range(n_test-1):
    step = -0.0058 if random() < 0.5 else 0.0058
    value = forecast[i-1] + step
    forecast.append(value)


#plot the 30 day forecast
pyplot.plot(forecast, label="Forecast", color='grey')
pyplot.plot(test, label = "Observed" )
pyplot.legend(loc="upper left")
pyplot.xlabel("Days")
pyplot.ylabel("Volatility")
pyplot.title("Random Walk Forecast")
pyplot.xticks(range(0,22))
pyplot.show()

#Evaluating the forecast
print("Root Mean Squared Error", np.sqrt(metrics.mean_squared_error(test,
forecast)))
print("Mean Absolute Error:", metrics.mean_absolute_error(test, forecast))

def mean_absolute_percentage_error(test, forecast):
    test, forecast = np.array(test), np.array(forecast)
    return np.mean(np.abs((test - forecast) / test)) * 100
print("Mean Absolute Error",mean_absolute_percentage_error(test,forecast))
```

```
print("confidence", metrics.r2_score(test, forecast))
```

```
#ARIMA MODEL

import pandas as pd
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from matplotlib import pyplot
from sklearn import metrics
from statsmodels.tsa.arima_model import ARIMA
import numpy as np

df = pd.read_excel('/Users/mariongemst/Desktop/DATASETS OSEAX /OSEAX
MAIN.xlsx')
df = df.dropna()
returns = df["RETURN"]
stdev = df["STDEV"]


n_test = 22
train = stdev[:-n_test]
test = stdev[-n_test:]

#Plotting ACF and PACF

plot_acf(stdev, title="")
pyplot.xlabel("Lag")
pyplot.ylabel("ACF")
pyplot.show()

stdev_diff = stdev.diff()
stdev_diff = stdev_diff.dropna()
stdev_diff = stdev_diff.values

plot_acf(stdev_diff, title="")
pyplot.xlabel("Lag")
pyplot.ylabel("ACF")
pyplot.show()

plot_pacf(stdev_diff, title="")
pyplot.xlabel("Lag")
pyplot.ylabel("PACF")
pyplot.show()

#Confirming if we need first order differencing

stdev = stdev.values
```

```
model1 = ARIMA(test, order = (0,1,0))
model_fit_check = model1.fit(disp=0)

residuals = DataFrame(model_fit_check.resid)
residuals.plot()
pyplot.show()
residuals.plot(kind="kde")
pyplot.show()
print(residuals.describe())
print(model_fit_check.summary())

residuals = DataFrame(model_fit_check.resid)
residuals.plot()
pyplot.show()
residuals.plot(kind="kde")
pyplot.show()
print(residuals.describe())
print(model_fit_check.summary())

#Creating and fitting the model

previous = [x for x in train]
predictions = list()
for t in range(len(test)):
        regressor = ARIMA(previous, order=(1,1,0))
        model_fit = regressor.fit(disp=0)
        output = model_fit.forecast()
        prediction = output[0]
        predictions.append(prediction)
        obs = test[t]
        previous.append(obs)

print(model_fit.summary().tables[1])

#Plotting the Forecast and observed values

#Plotting the one month forecast
pyplot.plot(predictions, label="Forecast", color='grey')
pyplot.plot(test.values, label = "Observed" )
pyplot.legend(loc="upper right")
pyplot.xlabel("Days")
pyplot.ylabel("Volatility")
pyplot.title("ARIMA Forecast")
pyplot.xticks(range(0,21))
pyplot.show()

#Evaluating the model
```

```
print("Root Mean Squared Error", np.sqrt(metrics.mean_squared_error(test,
predictions)))
print("Mean Absolute Error:", metrics.mean_absolute_error(test, predictions))

def mean_absolute_percentage_error(test, forecast):
    test, forecast = np.array(test), np.array(forecast)
    return np.mean(np.abs((test - forecast) / test)) * 100
print(mean_absolute_percentage_error(test,predictions))

print("confidence", metrics.r2_score(test, predictions))
```

```
#GARCH MODEL

import pandas as pd
from arch import arch_model
from matplotlib import pyplot
import numpy as np
from sklearn import metrics
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.stats.diagnostic import het_arch
import pandas as pd

df = pd.read_excel("/Users/mariongemst/Desktop/DATASETS OSEAX /OSEAX
MAIN.xlsx")
df = df.dropna()
returns = df["RETURN"]
stdev = df["STDEV"] # this doesnt remove date
stdev_diff = stdev.diff()
stdev_diff = stdev_diff.dropna()
stdev_diff = stdev_diff.values
stdev = stdev.values
returns = returns

#Testing for ARCH effects
het_arch(returns)

#Plotting ACF and PACF of squared returns
squared_returns = returns**2

plot_acf(squared_returns, lags = 60, title="")
pyplot.xlabel("Lag")
pyplot.ylabel("ACF")
pyplot.show()

plot_pacf(squared_returns, title="")
pyplot.xlabel("Lag")
pyplot.ylabel("PACF")
```

```
pyplot.show()

#Create train and test
n_test = 5
train = returns[:-n_test]
test = stdev[-n_test:]

#Scaling returns
returns *= 100

#Creating and fitting model
model = arch_model(train, vol='GARCH', p=49, q=9)  #include mean?
res = model.fit()

df["forecast_stdev"] = 0.01*np.sqrt(res.params['omega']
+ res.params['alpha[1]'] * res.resid**2
 + res.conditional_volatility**2 * res.params['beta[1]'])

forecasted = df["forecast_stdev"]
forecast = forecasted[:n_test]

#Plotting one month forecast
pyplot.plot(forecast.values, label="Forecast", color='grey')
pyplot.plot(test, label = "Observed" )
pyplot.legend(loc="upper right")
pyplot.xlabel("Days")
pyplot.ylabel("Volatility")
pyplot.title("GARCH Forecast")
pyplot.show()

#Evaluating the forecast
print("Root Mean Squared Error", np.sqrt(metrics.mean_squared_error(test,
forecast.values)))
print("Mean Absolute Error:", metrics.mean_absolute_error(test, forecast.values))

def mean_absolute_percentage_error(test, forecast):
    test, forecast = np.array(test), np.array(forecast)
    return np.mean(np.abs((test - forecast) / test)) * 100
print("Mean Absolute Percentage Error",
mean_absolute_percentage_error(test,forecast.values))

print("confidence", metrics.r2_score(test, forecast.values))
```

```
#KNN

import pandas as pd
import matplotlib.pyplot as pyplot
```

```
from sklearn import metrics
import numpy as np
from sklearn.neighbors import KNeighborsRegressor

df = pd.read_excel('/Users/mariongemst/Desktop/DATASETS OSEAX /Sliding
Window stddev.xlsx')

#Create train and test

n_test = 22
train = df[:-n_test]
test = df[-n_test:]

#Seperate the independent and the target varibale on training data
train_x = train.drop(columns=["y Dependent"],axis=1)
train_y = train["y Dependent"]

#seperate the independent and target variable on testing data
test_x = test.drop(columns=["y Dependent"],axis=1)  #
test_y = test["y Dependent"]

#Determing the value of K

error = []
for K in range(300):
    K = K+1
    test = KNeighborsRegressor(n_neighbors = K)
    model_test.fit(train_x, train_y)  l
    pred_test = model.predict(test_x)
    rmse= np.sqrt(metrics.mean_squared_error(test_y,pred))
    error.append(error) #store rmse values
    print('RMSE value for k= ' , K , 'is:', error)

#plotting the rmse values against k values
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
pyplot.show()

#Creating, fitting and making predictions
regressor = KNeighborsRegressor(n_neighbors=39)
regressor.fit(train_x,train_y)
y_pred = regressor.predict(test_x)

#Plotting one month forecast
pyplot.plot(y_pred, label="Forecast", color='grey')
pyplot.plot(test_y.values, label = "Observed" )
pyplot.legend(loc="upper right")
pyplot.xlabel("Days")
```

```
pyplot.ylabel("Volatility")
pyplot.title("K-Nearest Neighbors ")
pyplot.xticks(range(0,21))
pyplot.show()

#Evaluating the model
print("Root Mean Squared Error", np.sqrt(metrics.mean_squared_error(test_y,
y_pred)))
print("Mean Absolute Error:", metrics.mean_absolute_error(test_y, y_pred))

def mean_absolute_percentage_error(test, forecast):
    test, forecast = np.array(test), np.array(forecast)
    return np.mean(np.abs((test - forecast) / test)) * 100
print(mean_absolute_percentage_error(test_y, y_pred))

print("confidence", metrics.r2_score(test_y, y_pred))
```

```
#RANDOM FOREST

import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from matplotlib import pyplot
from sklearn import metrics
import numpy as np

df = pd.read_excel('/Users/mariongemst/Desktop/DATASETS OSEAX /Sliding
Window stddev.xlsx')

n_test= 22
train = df[:-n_test]
test = df[-n_test:]

#Seperate the independent and the target varibale on  training data
train_x = train.drop(columns=["y Dependent"],axis=1)
train_y = train["y Dependent"]

#seperate the independent and target variable on testing data
test_x = test.drop(columns=["y Dependent"],axis=1)
test_y = test["y Dependent"]

#Creating, fitting and predicting
regressor = RandomForestRegressor(n_estimators=3, random_state=0)
regressor.fit(train_x,train_y)
y_pred = regressor.predict(test_x)
test_y = test_y.values
```

```
#Plotting the forecast and observed values
pyplot.plot(y_pred, label="Forecast", color='grey')
pyplot.plot(test_y, label = "Observed" )
pyplot.legend(loc="upper right")
pyplot.xlabel("Days")
pyplot.ylabel("Volatility")
pyplot.title("Random Forest Forecast")
pyplot.show()

print("Root Mean Squared Error", np.sqrt(metrics.mean_squared_error(test_y,
y_pred)))
print("Mean Absolute Error:", metrics.mean_absolute_error(test_y, y_pred))

def mean_absolute_percentage_error(test, forecast):
    test, forecast = np.array(test), np.array(forecast)
    return np.mean(np.abs((test - forecast) / test)) * 100
print(mean_absolute_percentage_error(test_y, y_pred))

print("confidence", metrics.r2_score(test_y, y_pred))
```