ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MACHINE LEARNING IN FINANCE
FINAL PROJECT

# Optiver Realized Volatility Prediction

*Authors:*
Marco GIULIANO
368707
Luca SALVADOR
369196
Riccardo CROCI
373478
Alessandra DI GIACOMO
368706

*Instructor:*
Elise Marie GOURIER
*Teaching Assistants:*
Giuseppe MATERA

# Contents

# Introduction

Volatility is a key concept in the financial world as it measures the degree of variation or fluctuations in the prices of financial instruments over time. The Optiver Realized Volatility Prediction challenge, hosted on the website Kaggle from June 28, 2021, to January 10, 2022, was all about predicting these fast changes in stock prices. As the title suggests, the competition was organised by Optiver, a leading global electronic market maker.

In the Optiver challenge, the dataset contains order book and trades data to predict short-term stock volatility. This high granularity data is fundamental for modeling precise price movements over 10-minute intervals. The aim is to develop, evaluate and compare different econometrics and machine learning methods to predict the realized volatility in high frequency traded stocks. Additionally, we aim to identify the key variables that significantly contribute to accurate predictions and give them an economic interpretation.
The contributions of this report are:

- In section 1, we describe the dataset structure.
- In section 2, we analyse the literature on volatility prediction and its significance for market participants.
- In section 3, we cover the exploratory data analysis and present the feature engineering techniques applied to augment the dataset's dimensions for forecasting purposes.
- In section 4 we explain how we calculated the variables that have been used for data augmentation.
- In section 5, we detail the implementation of various statistical and machine learning models, including GARCH, Lasso, XGBoost, LightGBM, LSTM and Multi-Layer Perceptron.
- In section 6, we discuss the methodologies used and interpret the results to identify the most effective methods and variables for predicting volatility in high-frequency trading contexts.
- In section 7, we summarise the results and provide an economic interpretation of the variables used, explaining their significance and reasons for their varying levels of importance.

# 1 Data Structure

This dataset contains stock market data, specifically designed for the practical aspects of trade executions in financial markets. It contains snapshots of the order book and executed trades with a one-second resolution, offering a granular view of the micro-structure of modern financial markets. The dataset is part of a code competition where the majority of the test set is hidden and only accessible during code submission. For this reason we will split the data we have for the training in order to test our result.

## Example Book Data

In table 1 there is a snippet of the book table:

## Book Variables Descriptions

- **Time Id:** Identifier for the time bucket.
- **Seconds in bucket:** Seconds elapsed since the start of the bucket.
- **Bid Price 1** and **Bid Price 2:** Normalized prices of the most competitive and second most competitive buy orders, respectively.
- **Ask Price 1** and **Ask Price 2:** Normalized prices of the most competitive and second most competitive sell orders, respectively.

Table 1: Example of Order Book Data

| Time Id | Seconds in bucket | Bid Price 1 | Ask price 1 | Bid Price 2 | Ask Price 2 |
|---|---|---|---|---|---|
| 5 | 0 | 1.001422 | 1.002301 | 1.00137 | 1.002353 |
| 5 | 1 | 1.001422 | 1.002301 | 1.00137 | 1.002353 |
| 5 | 5 | 1.001422 | 1.002301 | 1.00137 | 1.002405 |
| 5 | 6 | 1.001422 | 1.002301 | 1.00137 | 1.002405 |
| 5 | 7 | 1.001422 | 1.002301 | 1.00137 | 1.002405 |

| Bid Size 1 | Ask Size 1 | Bid Size 2 | Ask Size 2 | Stock Id |
|---|---|---|---|---|
| 3 | 226 | 2 | 100 | 0 |
| 3 | 100 | 2 | 100 | 0 |
| 3 | 100 | 2 | 100 | 0 |
| 3 | 126 | 2 | 100 | 0 |
| 3 | 126 | 2 | 100 | 0 |

- **Bid Size 1** and **Bid Size 2:** Number of shares available at the first and second most competitive buy levels.
- **Ask Size 1** and **Ask Size 2:** Number of shares available at the first and second most competitive sell levels.
- **Stock Id:** Stock identifier, used to track which stock the data pertains to.

The dataset is uniquely keyed by a combination of *stock id*, *time id*, and *seconds in bucket*. For effective feature engineering, missing seconds within each *time id* segment are forward-filled using the preceding data. This is essential as both bid and ask prices are expected to be available at every second in the market. For instance, if data for *seconds in bucket* = 2 is missing, it is filled with the data from *seconds in bucket* = 1.

Each pair of *stock id* and *time id* encapsulates data for 600 seconds, representing 10 minutes of market activity. The *time id* is not sequential for the stocks but serves as a unique identifier for these 10-minute intervals, while *seconds in bucket* are sequential within each interval.

In the challenge hosted on Kaggle, participants who have won, managed to reverse-engineer the sequential nature of *time id* and correlate it to specific stocks in the market. This gave them a competitive advantage because now they had the time series of the bid and ask, and also more data on the stock. However, this was not the intended goal set by the challenge's creators. Instead, participants were encouraged to focus on predicting stock volatility over the 10-minute periods using the provided data which is the approach used in this research project.

The dataset contains 112 different *stock ids* and 3,830 different *time ids*, yielding 428,960 unique pairs of *stock id* and *time id*. Ideally, this would result in 257,376,000 rows in *book trade*.

Due to missing data mentioned before, however, the actual number of rows currently stands at 167,253,289. This necessitates the forward-filling of missing data to complete the dataset for analysis.

However, some stock ids have a different number of distinct time ids compared to the others (see table 2).

Table 2: Number of distinct time ids for each stock

| Stock Id | Number of distinct time ids |
|---|---|
| 100 | 3829 |
| 13 | 3829 |
| 38 | 3815 |
| 75 | 3829 |
| 80 | 3820 |

This results in a total of 257,359,200 rows for the forward-filled dataset.

## Example Trade Data

In table 3 we can see an example of the trade data.

Table 3: Example of Stock Order Data

| Time Id | Seconds in bucket | Price | Size | Order Count | Stock Id |
|---|---|---|---|---|---|
| 5 | 21 | 1.002301 | 326 | 12 | 0 |
| 5 | 46 | 1.002778 | 128 | 4 | 0 |
| 5 | 50 | 1.002818 | 55 | 1 | 0 |
| 5 | 57 | 1.003155 | 121 | 5 | 0 |
| 5 | 68 | 1.003646 | 4 | 1 | 0 |

## Trade Variables Description

The trade dataset contains the following key variables:

- **Stock Id:** Identical to the stock identifier used in the order book.
- **Time Id:** Same as in the order book, this identifier marks the specific 10-minute trading window to which the data pertains.
- **Seconds in bucket:** Records the specific second within the *Time Id* when the trade occurred. Unlike in the order book, this field is not necessarily starting from 0 due to the sparser nature of trade occurrences.
- **Price:** Represents the normalized, volume-weighted average price of trades executed within that specific second.
- **Size:** Indicates the total number of shares traded in those transactions occurring within the specified second. This measure helps assess the volume and liquidity of the stock at any given time.
- **Order count:** Denotes the number of distinct trade orders that were executed during the second. This count is indicative of the market activity and the number of transactions being processed.

## Example Target Data

In table 4 we can see an example of the target data.

Table 4: Example Target Data

| Stock Id | Time Id | Target |
|---|---|---|
| 0 | 5 | 0.004136 |
| 0 | 11 | 0.001445 |
| 0 | 16 | 0.002168 |
| 0 | 31 | 0.002195 |
| 0 | 62 | 0.001747 |

### Target Variables Description

- **Stock Id:** This identifier is consistent with the one used in the order book and trade datasets.
- **Time Id:** Identical to those in the previously described datasets, this ID marks specific 10-minute trading windows.
- **Target:** Represents the realized volatility computed over the 10-minute window immediately following the period covered by the feature data. This metric serves as the prediction target and is the y variable to the forecasting tasks.

## 2 Literature Review

Realized volatility measures the degree of volatility of the return on investment over a specified period. Since the return is a random process, the actual volatility cannot be precisely computed in advance. Predicting volatility is important for economic and financial reasons. Firstly, it helps investors and portfolio managers assess risk levels, enabling them to adjust portfolios, hedge against losses, and make informed investment decisions. Volatility is also key in pricing options and derivatives, where accurate predictions lead to better pricing strategies. Additionally, volatility reflects market sentiment; increased volatility may indicate higher market uncertainty or credit risk, providing insights into market trends.[1].

Optiver operates as market maker: it quotes both buy and sell prices for a financial instrument with the aim of earning profit from the spread between these prices. The objective is to buy and sell in equal amounts to minimize large net positions. The primary function of market makers is to provide liquidity, ensuring the instrument can always be traded at quoted prices. Market makers are prevalent in foreign exchange trading and stock exchanges, where they facilitate transactions by acting as official market makers for specific equities. [2].

The classic market-making literature views the bid–ask spread, a significant part of investors' transaction costs, as compensation for the expenses a market maker incurs. This spread consists of three primary components: order-handling costs (such as the fees an exchange charges to process an order), the cost of being adversely selected on a bid or ask quote, and the premium that risk-averse market makers require to manage price risk on nonzero positions. Market makers prefer to operate in environments where these costs are minimized, such as systems with low fees or fast access [3]. Since these costs are influenced by volatility, being able to forecast this quantity is one of the major challenges for market makers. A wide range of methods have been developed over the years to provide an accurate prediction of this fundamental risk measure.
For example, past studies have employed time series techniques to model and predict the stock volatility. ARCH model was first proposed by Engle in 1982 and used for volatility forecasting [4].By looking at empirical evidences, it was found that the volatility was strongly affected by macroeconomics and management decisions. For this reason, the model was subsequently extended by Bollerslev in 1986 who developed a generalized autoregressive conditional heteroskedasticity (GARCH) model [5]. In 2005, Awartani and Corradi proposed that symmetric and asymmetric GARCH is applicable to symmetric and asymmetric stock volatility forecasting [6].
Even though these methods provided decent forecasting results, they have been developed and evaluated based on low frequency data.

Instead, nowadays, with significant improvements in computing performance, data can be collected at very high frequencies, at intervals of a few seconds, and with increasingly higher precision. Thus, computational progress has led the way for a new challenge: predicting high-frequency volatility, which has different characteristics that past models are unable to capture.
Machine learning techniques could provide powerful forecasting performances. The gradient boosting decision trees (GBDT) implemented with the distributed computing framework XGBoost by Basak, S. in 2019[7], outperformed the first model proposed by Khaidem in 2016 [8] by which was based on random forests.
Artificial neural networks (ANN) have also been applied to the problem of the volatility forecasting. However, because of the strong randomness and non linearity of financial data, ordinary neural networks might have poor performances. On the other hand, the long short-term memory neural network (LSTM), developed by Hochreiter in 1997[9], tends to perform better as it learns time dependencies, which is a unique advantage working with time series data.

There are different studies which display the power of the LSTM in volatility forecasting. In 2012 Maknickiene and Maknickas [10], improved the prediction performance of feed-forward NN using LSTM models and demonstrated that recurrent neural network models outperformed CNN models for the prediction of financial indicators. Besides, in 2017, Nelson used an LSTM model to predict the volatility of the stock market[11].

In summary, models like ARCH and GARCH have laid the foundation for volatility forecasting, but advancements in technology have enabled the use of high-frequency data and machine learning models. Methods such as XGBoost and LSTM, as well as other neural network models, have demonstrated superior performance in predicting volatility leading to a deeper understanding of market dynamics.

# 3 Exploratory Data Analysis

The target variable is characterized by the summary statistics in table 5

Table 5: Summary Statistics of the Target Variable

| **Statistic** | **Value** |
|---|---|
| Mean | 0.0039 |
| Median | 0.0030 |
| Standard Deviation | 0.0029 |
| Minimum | 0.0001 |
| 25th Percentile | 0.0020 |
| Median | 0.0030 |
| 75th Percentile | 0.0047 |
| Maximum | 0.0703 |
| Skewness | 2.8226 |
| Kurtosis | 14.9611 |

The distribution of the target is heavily right-skewed and there are extreme outliers since some of the stocks are very volatile. Those extreme outliers can be spotted in figure 1 and 2 .

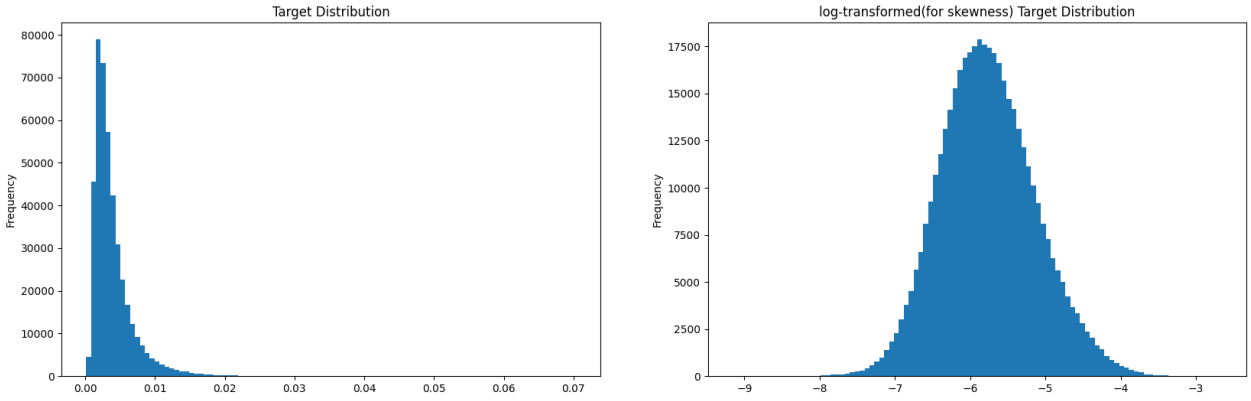


Figure 1: Target distribution

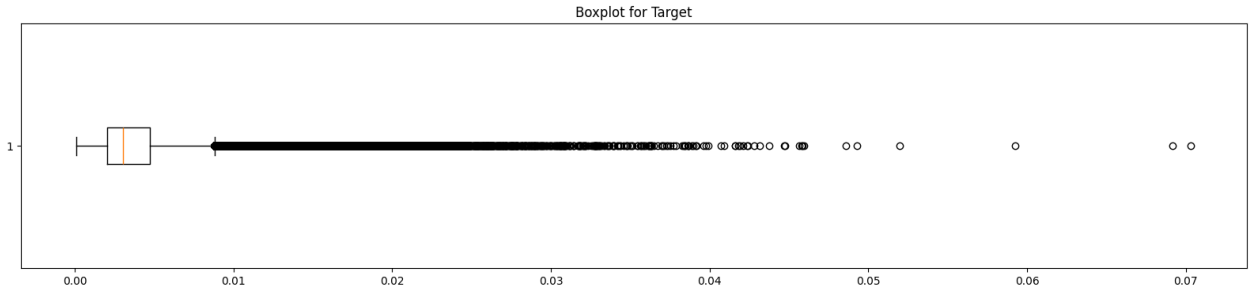


Figure 2

One stock can be extremely volatile in one time bucket and less volatile in another time bucket. This phenomenon can be explained by time ids not being sequential, and there are not temporal dependencies between them.
To understand differences in magnitude of realized volatility it is useful to analyse the least and the most volatile time buckets in figures 3 and 4

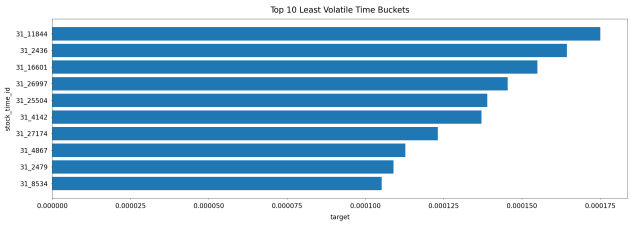


Figure 3: Least volatile buckets

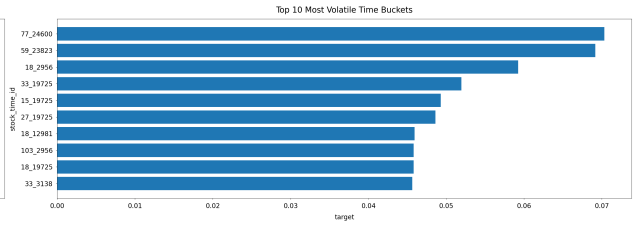


Figure 4: Most volatile buckets

The most volatile time bucket belongs to stock 77 and its time id is 24600. The most volatile stock was stock 18 and it has 3 time buckets in this list.

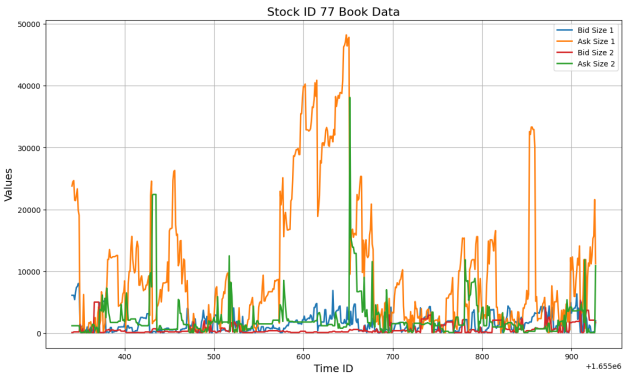


Figure 5: Ask and Bid size

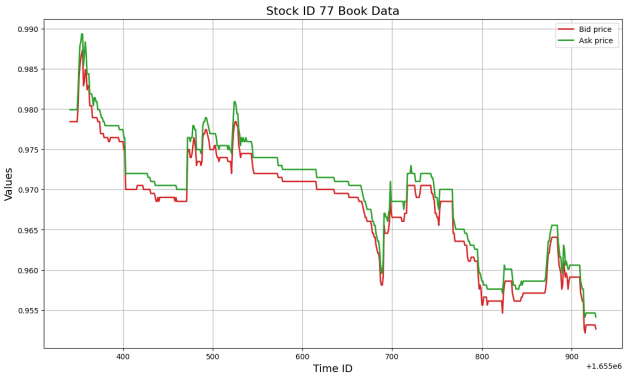


Figure 6: Ask and Bid price

Examining the features of the most volatile time buckets reveals a significant and consistent variation in ask and bid sizes, along with an extremely volatile bid-ask spread. This combined effect contributes to increased realized volatility. Additionally, a substantial imbalance between buy and sell orders can signal potential stock movement, with buying pressure possibly driving the stock price higher, and selling pressure potentially causing it to drop.

Furthermore, in the most volatile time buckets, numerous trades are recorded. For example, in the most volatile bucket, trading occurs 228 times, exhibiting significant variability in order size, order count, and price. This pattern is consistent across all the buckets shown in figure 4.

To illustrate, in table 6 are the summary statistics for the most volatile bucket.

Table 6: Summary Statistics for most volatile bucket

| | **Bid Price 1** | **Ask Price 1** | **Ask Size 1** | **Bid Size 1** | **Price** |
|---|---|---|---|---|---|
| count | 587 | 587 | 587 | 587 | 228 |
| mean | 0.9685 | 0.9691 | 10074.3 | 1539.8 | 0.9680 |
| std | 0.0071 | 0.0072 | 10935.8 | 1434.4 | 0.0080 |
| min | 0.9527 | 0.9537 | 16 | 30 | 0.9530 |
| 25% | 0.9636 | 0.9641 | 2100 | 500 | 0.9605 |
| 50% | 0.9700 | 0.9705 | 5950 | 1100 | 0.9693 |
| 75% | 0.9730 | 0.9735 | 14039.5 | 2186 | 0.9741 |
| max | 0.9879 | 0.9888 | 48222 | 8021 | 0.9880 |

The least volatile 10 time buckets are shown in figure 3. All of the least volatile 10 time buckets belong to stock 31, even though it has an average volatility overall. It can be notice that bid and ask prices are extremely stable over the period. Regarding order size, event though there is a difference in terms of magnitude, in markets with a constant bid-ask spread, the impact of large trades can be mitigated. If large trades do not significantly widen the spread, it suggests that the market can absorb these trades without drastic price changes, leading to lower volatility.

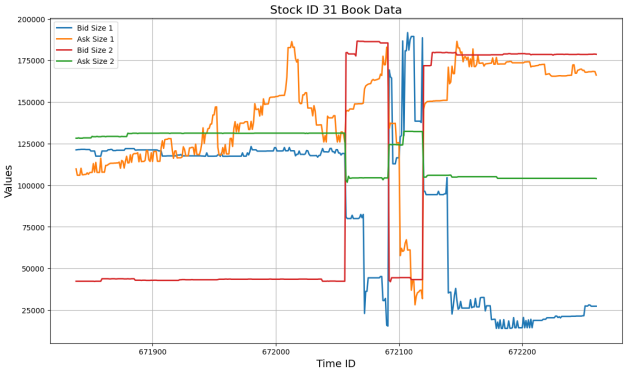


Figure 7: Ask and Bid size

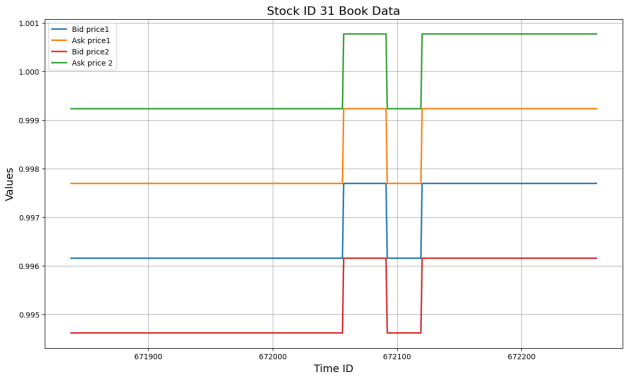


Figure 8: Ask and Bid price

Moreover, a stable and tight bid ask spread indicates higher liquidity which is typically associated with lower volatility. In contrast to the pattern related to most volatile buckets, here the number of trade occurrences is significantly lower : only 10 in the least volatile bucket, and even less in the two subsequent buckets, namely 3 and 1 for bucket 2436 and 16601 , respectively. Table 7 illustrates, as example the statistics for the least volatile bucket.

Table 7: Summary Statistics for least volatile bucket

| | **Bid Price 1** | **Ask Price 1** | **Ask Size 1** | **Bid Size 1** | **Price** |
|---|---|---|---|---|---|
| count | 423 | 423 | 423 | 423 | 11 |
| mean | 0.9968 | 0.9983 | 87543 | 141486.7 | 0.9977 |
| std | 0.0008 | 0.0008 | 47182.2 | 30536.5 | 0 |
| min | 0.9962 | 0.9977 | 13979 | 28170 | 0.9977 |
| 25% | 0.9962 | 0.9977 | 27579 | 120438 | 0.9977 |
| 50% | 0.9962 | 0.9977 | 117473 | 146385 | 0.9977 |
| 75% | 0.9977 | 0.9992 | 120588 | 167577 | 0.9977 |
| max | 0.9977 | 0.9992 | 191692 | 186430 | 0.9977 |

Finally, it can be noticed that the trade price has no volatility for this bucket. More generally, it tends to be much more stable when the realized volatility is lower.
In order to take into account all these features combined together, the weighted average price and the bid-ask spread are computed within each time bucket.
From this data analysis also emerges that the number of times in which a trade is realized within the bucket can play an important role in predicting volatility : larger trader activity is generally linked with more volatile buckets and stocks. Therefore, a dummy variable indicating whether a trade occurs or not within the time bucket is included in the dataset.

The table 8 shows that the bid ask spread and the weighted average price are significantly correlated with volatility, potentially indicating that these 3 will be significant variables while forecasting. A larger spread between the first and the second most competitive ask and bid price it is also likely to result in higher realized volatility, but this effect it is expected to be less relevant.

Table 8: Correlation of various features with volatility

| Feature | Correlation with volatility |
|---|---|
| Bid-ask spread | 0.690 |
| Wap1 | 0.561 |
| Wap2 | 0.578 |
| Ask spread | 0.301 |
| Bid spread | 0.298 |

# 4 Feature engineering

## 4.1 Variables created

In this section we explain the process of feature engineering implemented. First we calculated the following variables from the data we already had.

- **Weighted average price**:

$$WAP = \frac{BidPrice \times AskSize + AskPrice \times BidSize}{BidSize + AskSize}. \tag{1}$$

  A fair book-based valuation must consider two key factors: order level and order size. Therefore, the Weighted Average Price (WAP) in equation 1, is used to calculate the instantaneous stock valuation, incorporating both price and volume information. If two order books have the same bid and ask prices, the one with more offers will result in a lower stock valuation. This in because a higher number of sell orders indicates a greater supply in the market, leading to lower stock valuation.

- **Bid-Ask Spread**: This is the difference between the lowest price a seller is willing to accept (the ask) and the highest price a buyer is willing to pay for an asset (the bid), divided by the bid price as in equation 2. It serves as a measure of market liquidity. A more liquid market, with more active buyers and sellers, results in a tighter bid-ask spread. Conversely, wider spreads generally lead to higher volatility.

$$BidAskSpread = \frac{AskPrice_1 - BidPrice_1}{BidPrice_1} \tag{2}$$

- **Bid-spread**: computed as:

$$BidSpread = \frac{BidPrice_1 - BidPrice_2}{BidPrice_1 + BidPrice_2} \tag{3}$$

  A higher bid spread implied a more illiquid market. A larger gap between the highest and the second highest bid prices indicates fewer buyers willing to purchase the asset at prices close to the highest bid. In a highly competitive market, many buyers compete for the asset, leading to bid prices that are closer to each other.

- **Ask-spread**:

$$AskSpread = \frac{AskPrice_2 - AskPrice_1}{AskPrice_1 + AskPrice_2} \tag{4}$$

  Similar to the bid spread, the ask spread captures market illiquidity on the ask side.

## 4.2 Our approach

Then, the 600-second interval was divided into **60 windows of 10 seconds each**, and **six variables were created** to summarise the relevant features **of each interval**.

- **Wap1-log-high-low**: Computed as

$$Wap1LogHighLow = \log(\max(\text{WAP}_1)) - \log(\min(\text{WAP}_1)) \tag{5}$$

  It represents the difference in logarithms of the highest and lowest WAPs within each 10-second interval. This metric captures the price volatility within the interval, indicating significant price movements or stability. By using logarithms, it normalizes the data, making it easier to compare price fluctuations across different intervals.

- **Wap2-log-high-low**: Same as above, but for the second most competitive WAPs.

- **Ask1-bid1-spread-avg**: Average bid-ask spread value for the most competitive ask and bid in the interval.

- **Bid-spread-avg**: Average bid spread value in the interval.

- **Ask-spread-avg**: Average ask spread value in the interval.

- **Trade**: Dummy variable which takes the value 0 or 1. It indicates whether there has been a trade or not in the interval.

# 5 Models Defintion

## K-Fold Cross validation

To train our models, we split the dataset using the k-fold cross-validation technique [12, Chapter 7.10]. In this approach, the dataset is divided into $k$ different subsets called folds. The model is then trained $k$ times, each time using $k-1$ folds for training and the remaining fold for testing.

During the k-fold cross-validation process, we used 90% of the data, dividing it into k equal parts each time to determine the best hyperparameters. The remaining 10% of the data was reserved as the test set for the final evaluation.

## 5.1 GARCH

The GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model, introduced by Bollerslev (1986) [13], extends the ARCH model by Engle (1982) [14] to model financial time series with volatility clustering. It incorporates past variances and past errors to model changes in variance over time.

The GARCH model generalizes the ARCH by incorporating past conditional variances:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2 \tag{6}$$

where $\sigma_t^2$ is the conditional variance at time $t$, $\alpha_0$ is a constant term, $\alpha_i$ are the coefficients associated with the lagged squared errors, $\beta_j$ are the coefficients associated with past conditional variances, $q$ is the number of lagged squared errors, $p$ is the number of lagged conditional variances, $\epsilon_{t-i}$ are the error terms form previous time periods and $\sigma_{t-j}^2$ are the conditional variances from previous time periods. The error terms formula $\epsilon_t = \sigma_t z_t$ remains the same as the ARCH model.

## 5.2 Lasso

The Least Absolute Shrinkage and Selection Operator (Lasso) Regression, represented a significant contribution in the field of statistics. Introduced by Robert Tibshirani in his seminal 1996 paper, "Regression Shrinkage and Selection via the Lasso" [15], this method has significantly influenced modelling techniques.

As per Hastie [12], equation 7 is the one for the estimation of the parameters in lasso regression.

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}. \tag{7}$$

Unlike the Ridge Regression, no closed form solution exists for Lasso.

## 5.3 XGBoost (eXtreme Gradient Boosting)

XGBoost (eXtreme Gradient Boosting) is a highly efficient and scalable machine learning model, as described by Tianqi Chen and Carlos Guestrin[16]. XGBoost has become popular for its performance in machine learning competitions and its capability in handling large-scale data. It implements the gradient boosting algorithm for decision trees, building the model in a stage-wise manner and allowing optimization of arbitrary differentiable loss functions. The model includes an algorithm for approximate tree learning, which selects the best split based on a subset of the data, making it suitable for large datasets. To manage instance weights effectively in tree learning, XGBoost uses a weighted quantile sketch algorithm. This helps in finding the best split points under weighted data scenarios.

An essential formula in XGBoost includes the loss function optimization, where the model aims to minimize a given differentiable loss function $L$ with respect to the predictions $y_i$ made at each step. The update rule for the model's predictions can be generally expressed as in equation 8.

$$y_i^{(t+1)} = y_i^{(t)} + \eta \cdot f_t(x_i) \tag{8}$$

Here, $y_i^{(t)}$ represents the prediction at step $t$, $\eta$ is the learning rate, and $f_t(x_i)$ is the output of the new tree added at step $t$. The trees are built to minimize the objective function in equation 9 at each iteration:

$$\text{Obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k) \tag{9}$$

where $l$ is the loss function comparing the predicted $\hat{y}_i^{(t)}$ and true $y_i$ labels, and $\Omega$ represents the regularization term for the trees.[17]

These features collectively make XGBoost a powerful tool for many predictive modeling tasks and a favorite in the machine learning community.[17]

## 5.4 LightGBM

LightGBM follows the gradient boosting framework and it works by constructing an ensemble of decision trees, but it differs from other gradient boosting implementations in how it grows these trees. Indeed, instead of growing trees level-wise, LightGBM grows trees leaf-wise. This means it selects the leaf that is expected to result in the largest decrease in loss for further splitting[18]. Additionally, LightGBM does not use the commonly employed sorted-based decision tree learning algorithm, which finds the best split point on sorted feature values[19], as seen in XGBoost and similar frameworks. Instead, LightGBM utilizes an optimized histogram-based decision tree learning algorithm, which improves efficiency and reduces memory consumption. To prevent over.fitting, it uses some regularization parameters :

- **Number of Leaves**: controls the maximum number of leaves for each tree. Larger numbers are associated with better fit but also with higher risk of overfitting.
- **Learning rate**: scales the contribution of each tree added to the ensemble. A lower learning rate makes the model more robust to overfitting but it is more computationally expensive.
- **Feature fraction and Bagging fraction**: introduce randomness in the training process, preventing the tree from relying too heavily on the same subset of features and data.
- **Bagging frequency**: controls how often bagging is applied.

## 5.5 LSTM Neural Network

Long Short-Term Memory (LSTM) [20] networks are a type of recurrent neural network (RNN) architecture address the vanishing gradient problem faced by standard RNNs by incorporating memory cells that can maintain information over long periods. The key components of LSTM are:

- **Cell State:** The cell state acts as a memory that can carry information across many time steps.
- **Gates:** Gates are used to control the flow of information into and out of the cell state.
- **Forget Gate:** Decides what information to discard from the cell state.
- **Input Gate and Output Gate:** Determine what new information to store in the cell state and what to output from the same cell .
- **Hidden state:** is a vector that captures the information from previous time steps and is passed along to the subsequent time steps.
- **Activation Function** introduces non linearity in the model.

The activation functions employed in this work are the Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

and the Hyperbolic Tangent :

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

.

We used an LSTM network which begins with an input layer followed by an LSTM layer, dropout layer, and two fully connected layers with an activation function in between. The LSTM layer captures temporal dependencies in the input sequences, while the fully connected layers process the hidden states to produce the final output. Dropout is used for regularization to prevent overfitting during training. We fit different models using Sigmoid and Tanh as activation functions and we compare the results with those obtained without account for non linearity. Additionally, input features have been standardized and we used batch size of 64, lower than the training set size.

## 5.6 Multi Layer Percepton

A Multi-Layer Perceptron (MLP) consists of multiple layers of interconnected neurons. Each neuron in a layer is connected to every neuron in the subsequent layer, making the network fully connected. Activation functions introduce non-linearity into the network, enabling it to learn and model complex patterns. The most common activation function and the one used here in this report is: `relu(x)=max(0,x)` The training of MLPs involves adjusting the weights and biases of neurons to minimise the prediction error, which in our case is the RMSPE to align with the final metric (instead of the classical choice of MSE Loss). This process employs back-propagation and optimization algorithms like gradient descent. According to Goodfellow et al. (2016) [21], the key steps in training an MLP include initialisation, forward propagation, loss calculation, backpropagation, and weight updates, iteratively refining the network until it achieves satisfactory performance.

Our neural network architecture is as follows:

- **Input Layer:** 302 features
- **Hidden Layers:** 3 hidden layers with dimensions:
  - 256 units
  - 128 units
  - 64 units

- **Output Layer:** 64 to 1 unit

Between each linear layer, we applied ReLU activation.

We also experimented with applying batch normalization and dropout between each layer which seem to help toghether the training of the MLP. Batch normalization normalizes the output of a layer by scaling and shifting, stabilizing, and accelerating training [21]. Dropout, on the other hand, randomly sets a fraction of input units to zero during training to prevent overfitting.

# 6 Approaches used and Interpretation of Results

Feature importance in LASSO and in the neural network models can be assessed using the permutation importance function. This method evaluates the significance of each feature by measuring how much the model's performance metric, in this case, the Root Mean Square Percentage Error (RMSPE), changes when the values of that feature are randomly shuffled. If shuffling a feature's values results in a significant increase in the RMSPE, the feature is considered important for the model's predictions. On the other hand, if the RMSPE does not change much, the feature is considered less important. This technique for assessing feature importance was introduced by Breiman [22].

In the context of LightGBM and XGBoost, gain-based feature importance is theoretically preferred over permutation importance because it directly measures the contribution of each feature to the model's decision-making process. Gain importance assesses how much the improvement in the model's objective function is attributable to each feature's split across all trees. This method captures the intrinsic value of features in creating effective splits that improve model accuracy, reflecting their true predictive power within the model's structure.

## 6.1 GARCH

In this project, the GARCH model yielded an RMSPE of approximately 0.9, indicating a substantial difference between predicted and actual values. This result highlights the model's struggle to capture asymmetric responses in volatility to past innovations and certain volatility patterns, as it assumes symmetric behavior. GARCH treats positive and negative shocks of the same magnitude similarly, often missing patterns where negative shocks cause more significant changes—known as the leverage effect[23].

We did not test models like AP-ARCH and E-GARCH because we believed they would not provide significant improvement. The primary reason for GARCH's suboptimal performance lies in the nature of this task. Specifically, with GARCH approach, we were predicting 600 observations (and then averaging them) with 600 data points or fewer, without the benefit of a rolling window approach.

## 6.2 Lasso

Initially, we attempted to scale the data using the `MinMaxScaler`. However, the results were unsatisfactory. Therefore, we switched to using the `StandardScaler`, which standardizes the features by removing the mean and scaling to unit variance. To evaluate the model's performance, we implemented the RMSPE scoring function.

To fine-tune the learning rate of the Lasso regression model, we used `GridSearchCV` to perform a search over a specified parameter grid. The grid included different values for the regularization parameter $\alpha$. We configured the grid search to use 5-fold cross-validation, which involves splitting the training data into five subsets. After performing the grid search on values ranging between $10^{-7}$ to $10^{-4}$, the best $\alpha$ parameter, $10^{-7}$, was identified based on the lowest RMSPE score. The best Lasso model was then selected and evaluated on the test set, yielding the following results:

- Validation RMSPE: 0.318
- Test RMSPE: 0.308
- Validation $R^2$: 0.793

- Test $R^2$: 0.794

The majority of the significant variables were in the last 15 ten-second buckets ( of the 60 total).

Among the top 50 features, the average bid-ask spread appears 31 times. Additionally, the difference in the logarithm of the highest and lowest weighted average appears 11 times for the second class and 10 times for the first class. The ranking of feature importance remains almost unchanged when using the absolute values of the coefficients as a possible metric for evaluating feature importance (since variables are standardised in our case, the importance of each feature could also be interpreted by examining the magnitude of its coefficient in absolute terms).

## 6.3 XGBoost (eXtreme Gradient Boosting)

To find the optimal hyperparameters we performed a grid search search on the 10 seconds bucket dataset. This Grid search is done with 5-fold cross-validation, and selecting the combination of hyper-parameters that averages the lowest RMSPE. The optimal hyper-parameters are:

- Number of estimators: The maximum number of boosting trees used in building the model. Optimal value found of 600.
- Max depth: The maximum depth of the trees in the model. Optimal value of 7.
- Learning rate: the step size of the weights update. Optimal value of 0.1.

Using these hyper parameters and training the model, the model reaches on the out of sample test set an RMSPE of: 0.258 and an $R^2$ of: 0.801.

The evolution of bid-ask spread throughout the interval plays a significant role in the decisions taken by the model, this can be seen in the feature importance chart in Figure 10.

It can also be observed that the weighted average price is fairly important in the decision. The dummy variable for the trade has moderate importance. Its relevance suggests that the occurrence of trades within a time frame is a notable factor for the model, potentially impacting liquidity and price movements.

## 6.4 LightGBM

We used 3-fold cross validation on the validation set to find the optimal hyperparameters as

- Number of leaves : 742
- Learning rate : 0.098
- Feature fraction: 0.3
- Bagging fraction: 0.9
- Bagging frequency: 1

With respect to XGBoost, we do not achieve significant improvements. The only remarkable difference in terms of features importance is that the variable indicating the presence of trades in the interval is not relevant now. On the other hand, as for XGBoost, the most relevant variables in forecasting process is the bid ask spread between the most competitive values and the logarithmic difference between both weighted average prices. These results suggest that realized volatility is significantly dependent of the market imbalances in bid and ask prices. Also the order size seems to be a key factor in predicting volatility as showed by the importance of the weighted average prices. The low persistence of the trade dummy suggests that in this context the occurrence of trades is much less relevant in terms of volatility impact.
After training the model, the performance metrics are :

- Validation RMSPE: 0.288

- Test RMSPE: 0.278
- Validation $R^2$: 0.806
- Test $R^2$: 0.814

A comparison with the results achieved with XGBoost, reveals that there is no improvement in terms of out of sample prediction performance.

## 6.5 LSTM Neural Network

We used a network with only one LSTM layer since the minor improvements achieved with more layers did not justify the high computation cost. We tried different activation functions keeping the dimension of the hidden state constant. The best results on the validation set have been achieved with Hyperbolic Tangent activation which outperforms both the model with Sigmoid activation and the one without activation function which does not take into account non linearity. Since Sigmoid activation function smooths gradient preventing jumps in output values it is not adequate to model our target variables since it presents significant fluctuations.

The results achieved with a learning rate equal to 0.001 consistently outperform the one obtained with learning rate= 0.005 for a given hidden size . Furthermore, the 96-dim hidden state does better than the other sizes across the different models, suggesting that larger networks are more prone to overfitting in this context. Therefore, the most accurate out of sample predictions have been achieved with learning rate = 0.001 and and hidden state size = 96 with a model trained for 30 epochs. The performance metrics on validation and set are :

- Validation RMSPE: 0.247
- Test RMSPE: 0.24
- Validation $R^2$: 0.738
- Test $R^2$: 0.743

Despite the fact that it has been proved that this method is particularly powerful in volatility forecasting[11], the LSTM does not perform significantly better than the other algorithms in this context. It can be related to the fact that there is not a precise time relationship between all the observations in the dataset. The values available are randomly selected for each stock in different periods and even though the time ID is consistent with the real order, they are not strictly consecutive. From a financial point of view, in such situation, incorporating and maintaining past observation into the the neurons of the network, to generate the successive output, could introduce large mistakes. The issue is particularly severe if observations far apart in reality are interpreted as consecutive in time. Volatility tends to be persistent within short periods if liquidity and risk conditions do not change suddenly. However, out data cannot exploit past information in such way since the recording period is not specified.

## 6.6 Multi Layer Percepton

During training, we experimented with different neural network architectures. Convolutional Neural Networks (CNNs) did not yield satisfactory results, likely due to their unsuitability for this regression task. Simpler Multi-Layer Perceptrons (MLPs) were also tested but resulted in an RMSPE of around 0.35.

Initially, using batch normalisation alone worsened our training performance, likely causing convergence towards a local minimum. Similarly, dropout alone did not yield significant improvements. However, combining batch normalisation and dropout proved beneficial, leading to more stable and effective training. We implemented dropout with a probability of 0.1, which significantly improved performance.

Additionally, we used a learning rate scheduler to dynamically adjust the learning rate, helping to prevent over-fitting and improve convergence.

The best performance was achieved with the chosen architecture trained for 30 epochs. We explored various configurations, including the number of hidden layers and the number of nodes in each layer, and ultimately found the most effective architecture. The achieved performance metrics are :

- Validation RMSPE: 0.240
- Test RMSPE: 0.237
- Validation $R^2$: 0.752
- Test $R^2$: 0.742

## 6.7 Feature Importance Plots

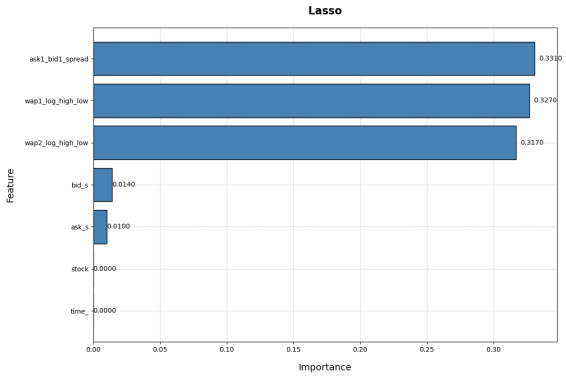


Figure 9: Feature importance with feature permutation for Lasso

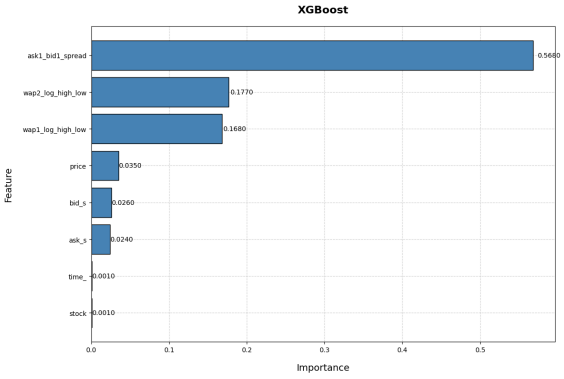


Figure 10: Feature importance with XGBoost

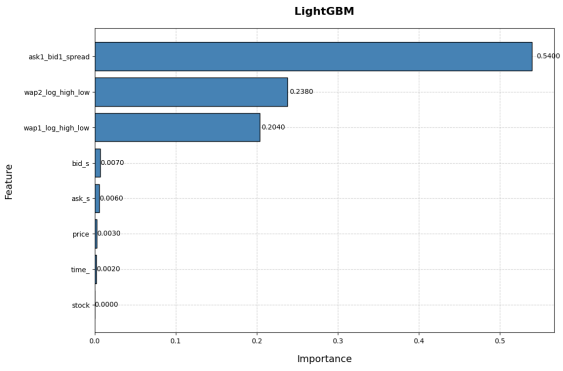


Figure 11: Feature importance with LightGBM

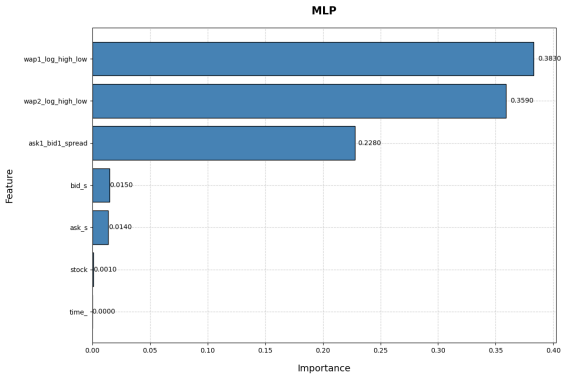


Figure 12: Feature importance with feature permutation for MLP

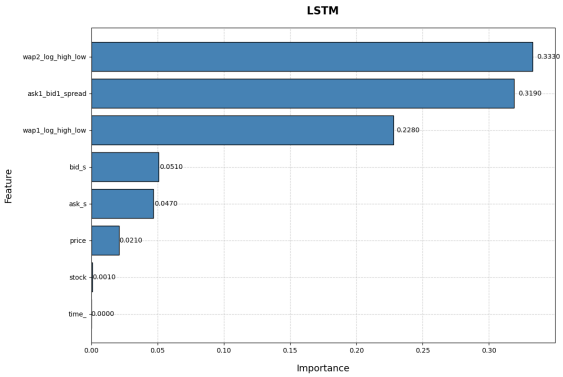


Figure 13: Feature importance with feature permutation for LSTM

As it can be seen for MLP there is a prevalence of a certain type of variable, while for Lasso, in general the more important features which contribute are the ones towards the end of the 600 seconds bucket.

# 7 Results Summary and Economic Interpretation

Summary results can be seen in table 9. In this analysis, we found that the Bid-Ask Spread is a highly relevant feature for predicting volatility. This metric, which reflects market depth and liquidity, shows a strong correlation with volatility. A narrower spread corresponds to higher liquidity and lower transaction costs, resulting in a more stable and less volatile market. Conversely, during periods of high volatility, the spread tends to widen due to increased uncertainty among market participants, leading to a larger gap between bid and ask prices. This correlation underscores the importance of the Bid-Ask Spread in our predictive models [24].

Similarly, the spread of log WAP is an important feature for volatility prediction in our analysis. This spread, economically might indicate market reactions to information and events, where a larger spread reflects higher uncertainty [25]
Large spreads can be associated with high volatility as indeed when the log(WAP) spread is wide, it signifies that there have been fluctuations in the WAP during the observed period.

On the other hand, bid spread and ask spread have shown not to have high relevance in volatility prediction. It is reasonable to think that the first and the second most competitive ask and bid prices tend to be relatively closer within the buckets likely due to high liquidity in the markets. However, it is not possible to identify the specific market on which stocks are traded to verify the real liquidity level. Furthermore, the presence of trades within the interval does not appear to significantly impact realized volatility. The associated dummy variable shows some relevance only within the XGBoost model, and even there, its importance is considerably lower compared to other variables. The dummy for trade was removed from the analysis in Lasso and MLP since it was very close to zero, and the models were re-trained not including that variable.

In our analysis, we observed that features became more significant in the last 600-second period bucket, indicating that recent data holds higher predictive power for future volatility. This trend suggests that short-term market movements are heavily influenced by the most recent information. Additionally, this finding aligns with the broader economic theory of market efficiency, particularly the semi-strong form, which asserts that markets quickly incorporate all publicly available information. Investors and policymakers can leverage this insight to optimise their strategies, mitigate risks, and enhance portfolio management practices. Economically, this implies that traders and investors can benefit from focusing on the latest data for decision-making, improving the accuracy of their strategies. The increased significance of recent variables also suggests that integrating real-time data can lead to more robust financial models, contributing to a more stable and efficient financial system. This real-time responsiveness is important for improving overall market liquidity and further enhancing market stability [26].

Table 9: Final Results on Test Set

| | **RMSPE** | $R^2$ |
|---|---|---|
| GARCH | 0.9 | x |
| LASSO | 0.308 | 0.794 |
| XGBoost | 0.258 | 0.801 |
| LightGBM | 0.276 | 0.814 |
| LSTM | 0.240 | 0.743 |
| MLP | 0.237 | 0.742 |

# Conclusion

This study explored the prediction of realized volatility in high-frequency trading environments using both econometric and machine learning models on the Optiver Realized Volatility Prediction dataset. The models developed, evaluated, and compared include GARCH, Lasso, XGBoost, LightGBM, LSTM, and MLP.

Among the models, the GARCH model showed the highest RMSPE, indicating its limitations in this context. Lasso Regression provided better performance, with an RMSPE of 0.308 and an $R^2$ of 0.794. Gradient boosting models, XGBoost and LightGBM, performed robustly, with RMSPEs of 0.258 and 0.276, respectively, emphasising the significance of the bid-ask spread and recent data in predicting volatility. Neural network models, LSTM and MLP, achieved RMSPEs of 0.240 and 0.237, respectively, and gave more importance to the variable using the WAP.

The variables related to WAP and bid-ask spread were identified as the most significant predictors of volatility, showing their strong correlation with market movements. Recent data within the last seconds of the 10-minute intervals proved highly predictive, aligning with the market efficiency hypothesis that the most recent information is crucial in financial markets.

Overall, the findings demonstrate the potential of machine learning techniques in financial volatility prediction, providing valuable insights into the important variables even when using deep learning techniques.

# References


[1] Maik Schmeling Charlotte Christiansen and Andreas Schrimpf. "A Comprehensive Look at Financial Volatility Prediction by Economic Variables". In: *Bank for International Settlements* (March 2012).

[2] Tanmoy Chakraborty and Michael Kearns. "Market Making and mean reversion". In: *UPenn CIS* (2011).

[3] Albert J. Menkveld. "High frequency trading and the new market makers". In: Volume 16 (4 2013).

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

[5] Tim Bollerslev. "Generalized Autoregressive Conditional Heteroskedasticity". In: *Journal of Econometrics* 31 (1986), pp. 307–327.

[6] B.M.A. Wartani and Valentina Corradi. "Predicting the Volatility of the S&P-500 Stock Index via GARCH Models: The Role of Asymmetries". In: *International Journal of Forecasting* 21.1 (2005), pp. 167–183.

[7] Sourav Basak et al. "Predicting the Direction of Stock Market Prices Using Tree-Based Classifiers". In: *North American Journal of Economics and Finance* 47 (2019), pp. 552–567.

[8] Luckyson Khaidem, Snehanshu Saha, and Sudeepa Roy Dey. *Predicting the direction of stock market prices using random forest*. 2016. arXiv: 1605.00003 [cs.LG].

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[10] Nijole Maknickiene and Arturas Maknickas. "Application of Neural Network for Forecasting of Exchange Rates and Forex Trading". In: (Nov. 2012), pp. 122–127.

[11] D. M. Q. Nelson, A. C. M. Pereira, and R. A. de Oliveira. "Stock Market's Price Movement Prediction with LSTM Neural Networks". In: (May 2017), pp. 1419–1426.

[12] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN: 9780387848846. URL: https://books.google.ch/books?id=eBSgoAEACAAJ.

[13] Tim Bollerslev. "Generalized autoregressive conditional heteroskedasticity". In: *Journal of Econometrics* 31.3 (Apr. 1986), pp. 307–327. URL: https://ideas.repec.org/a/eee/econom/v31y1986i3p307-327.html.

[14] Robert F. Engle. "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation". In: *Econometrica* 50.4 (1982), pp. 987–1007. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1912773 (visited on 05/01/2024).

[15] Robert Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996), pp. 267–288. ISSN: 00359246. URL: http://www.jstor.org/stable/2346178 (visited on 05/10/2024).

[16] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: KDD '16 (Aug. 2016). DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[17] Atallah Amor. "Machine learning for market volatility prediction". In: *Journal of Economics and Sustainable Development* 06 (2023), pp. 680–701.

[18] Andre Ye. "XGBoost, LightGBM, and Other Kaggle Competition Favorites". In: *Towards Data Science* (Sept. 2020). https://towardsdatascience.com/xgboost-lightgbm-and-other-kaggle-competition-favorites-5d07ca2907ef.

[19] Manish Mehta Rakesh Agrawal Jorma Rissanen. "SLIQ: A fast scalable classifier for data mining". In: *International Conference on Extending Database Technology* (2020).

[20] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[21] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[22] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[23] Robert W. Faff Timothy J. Brailsford. “An evaluation of volatility forecasting techniques”. In: *Journal of Banking Finance* 20 (3 1996), pp. 419–438.

[24] Chunchi Wu Jinliang Li. “Daily Return Volatility, Bid-Ask Spreads, and Information Flow: Analyzing the Information Content of Volume”. In: *The Journal of Business* Vol. 79 (September 2006), pp. 2697–2739.

[25] Yifan He and al. “Beyond the Bid–Ask: Strategic Insights into Spread Prediction and the Global Mid-Price Phenomenon”. In: (May 5, 2024).

[26] Kim Christensen and al. “A machine learning approach to volatility forecasting”. In: *CREATES Research Paper* (2021).