

# Market Volatility Prediction

☰ Authors Aman Saini

[Problem Statment](#)

[Basic Concepts](#)

[Order Book](#)

[Liquid Order Book](#)

[Statistics from the Order Book](#)

[Spread](#)

[Weighted Average Price](#)

[Log Returns](#)

[Realized Volatility](#)

[Data](#)

[Data files](#)

[Train Data](#)

[Variables/Features Description](#)

[Test Data](#)

[EDA and Feature Engineering](#)

[Basic Information](#)

[Data Shape](#)

[Data Snapshot](#)

[Window Plots](#)

[Stock Plots](#)

[Feature Extraction](#)

[Lag Features](#)

<a href="#">KMean as a Feature Engine</a>
<a href="#">Training and Evaluation</a>
<a href="#">Cross-Validation</a>
<a href="#">Model Selection</a>
<a href="#">Training</a>
<a href="#">Bayesian Optimization</a>
<a href="#">Neptune</a>
<a href="#">Results</a>
<a href="#">RMSPE</a>

## Problem Statment

Volatility is an important factor that can be harnessed to generate better returns from a financial instrument. Accurately predicting volatility is essential for the trading of options whose price is directly related to the volatility.

Build a model that can predict short-term volatility for stocks. Given a 10-minute window with highly granular feature data for stock, predict the volatility for the subsequent 10-minute window.

## Basic Concepts

Financial terminologies and concepts are vital in understanding the given data.

### Order Book

- It is a snapshot of the market situation for a particular stock. It represents the density of buyers and sellers at different prices of a stock.

bid #	price	ask #
	151	196
	150	189
	149	148
	148	221
251	147	
321	146	
300	145	
20	144	

Order Book 1

For example, the order book above shows the market situation for a hypothetical stock, say stock A, for a particular second  $t$ .

- Below is the order book for the same stock A at  $t + 1$

bid #	price	ask #
	151	196
	150	189
	149	148
	148	201
251	147	
321	146	
300	145	
20	144	

This is pretty clear that a trade of 20 stocks for 148 happened. That is, someone bought 20 stocks of stock A for 148 at time  $t + 1$ .

## Liquid Order Book

- When the order book is denser (like the one shown above), it is **liquid**. We can buy and share without worrying too much about the consequences!

For example, look at the order book below:

bid #	price	ask #
	151	20
	150	12
	149	1
	148	
5	147	
2	146	
	145	
16	144	

We to buy this stock, so we put a *bid* at 148. For now, there is no seller for 148, and if we get unlucky and prices go up, someone bid at 149, then we never got a chance to buy the stock.

Suppose we bid at 151, and we got the stocks, but we needed to pay a higher price for that.

Such order books are said to be less *liquid*.

## Statistics from the Order Book

Statistics from the order book can reflect market liquidity and stock valuation.

### Spread

- $Spread = \frac{BestOffer - BestBid}{BestBid}$

- Market makers try to reduce the spread in an order book. Spread is an important measure of liquidity for a stock.
- For example, spread in order book 1 is:

$$spread = \frac{148 - 147}{147} = 6.8e - 3$$

### Weighted Average Price

- The order book is a primary source for stock valuation. WAP (calculated from the order book) is the metric from the order book for this purpose.
- Two main factors in stock valuation are:
  1. The level order (price column in the order book)
  2. The size of the order (number of buyers and sellers)
- The formula for WAP:

$$WAP = \frac{BP_1 \times AS_1 + AP_1 \times BS_1}{BS_1 + AS_1}$$

where,

BP = Bid price

AP = Ask price

BS = Bid size

AS = Ask size

Subscript 1 represents the extreme values of these quantities (values defining the gap in an order book).

- WAP of the order book 1 is:

$$WAP = \frac{147 \times 221 + 148 \times 251}{251 + 221} = 147.532$$

## Log Returns

- To compare stock from one point in time to another, we cannot just look at the difference in stock price. Stock price by itself doesn't contain much information for what's really going on. It is the ratios (or % change) that give a comparatively clearer picture.
- For example, we have two stocks:

Stock A and stock B with current stock prices at \$100 and 10\$. After some time, the prices for stock A jumped to \$102, and stock B jumped to \$11.

Although the price increase in stock A (\$2) is greater than stock B (\$1), the move was proportionally much larger for stock B ( $1/10=10\%$ ) than stock A ( $2/100=2\%$ ). That is, if someone had invested the same amount (say \$1000) in both the stocks, then the profit in stock A ( $10 \times 2 = \$20$ ) is much lower than the profit in stock B ( $100 \times 1 = \$100$ ).

- To compare stock from one timestamp to another, **stock return** is used. The formula is as follow:

$$\text{stock return} = \frac{S_{t_2} - S_{t_1}}{S_{t_1}}$$

where  $S_t$  is the price of the stock  $S$  at time  $t$ .

- However, **log returns** are preferred whenever some mathematical modelling is required. The formula is:

$$r_{t_1, t_2} = \log\left(\frac{S_{t_2}}{S_{t_1}}\right)$$

Where  $t_1 < t_2$

- Advantages of log returns:
  1. They are additive in nature across time

$$r_{t_1, t_2} + r_{t_2, t_3} = r_{t_1, t_3}$$

2. Regular returns cannot go beyond  $\pm 100\%$ , while log-returns are not bounded.

## Realized Volatility

- Volatility is a measure of the variance of the price of an asset over time.
- It represents the extent of returns we get from a stock over time.
- Mathematically, it is the standard deviation of the stock log returns.

Since log returns are computed over a given time period, volatility will differ over longer or shorter intervals. For this reason, it is usually *annualized* over a 1-year period.

- Mostly, the price  $S_t$  in log returns are not the actual price of a stock rather the **WAP**.
- The formula:

$$\sigma = \sqrt{\sum_t r_{t-1, t}^2}$$

The units for time  $t$ , could represent a day, an hour, a minute, etc.

The formula is simpler than a usual standard deviation formula because the mean of log returns is zero (another advantage).

- There could be two types of volatility:

1. **Realized Volatility**

2. **Implied Volatility**

- Realized volatility is a statistical measure (standard deviation) and can be calculated using the asset's past data (order books of the past).
- Implied volatility can be calculated only if we have the actual traded or true price values at which the asset (stock from our examples) was traded. It is usually measured using numerical computation by using the *Black-Scholes* formula.

For example, if we have the past data of a stock such as offer values, bid values, realized volatility, and few other parameters required by the Black-Scholes formula. We can "estimate" the option price or the price at which the trade would happen. But since this is an estimate, it cannot exactly match the true price of the trade. Therefore, we tweak the value of input volatility (realized volatility) so that this estimate would be as close to the true value as possible.

The final volatility we get from this process is called implied volatility.

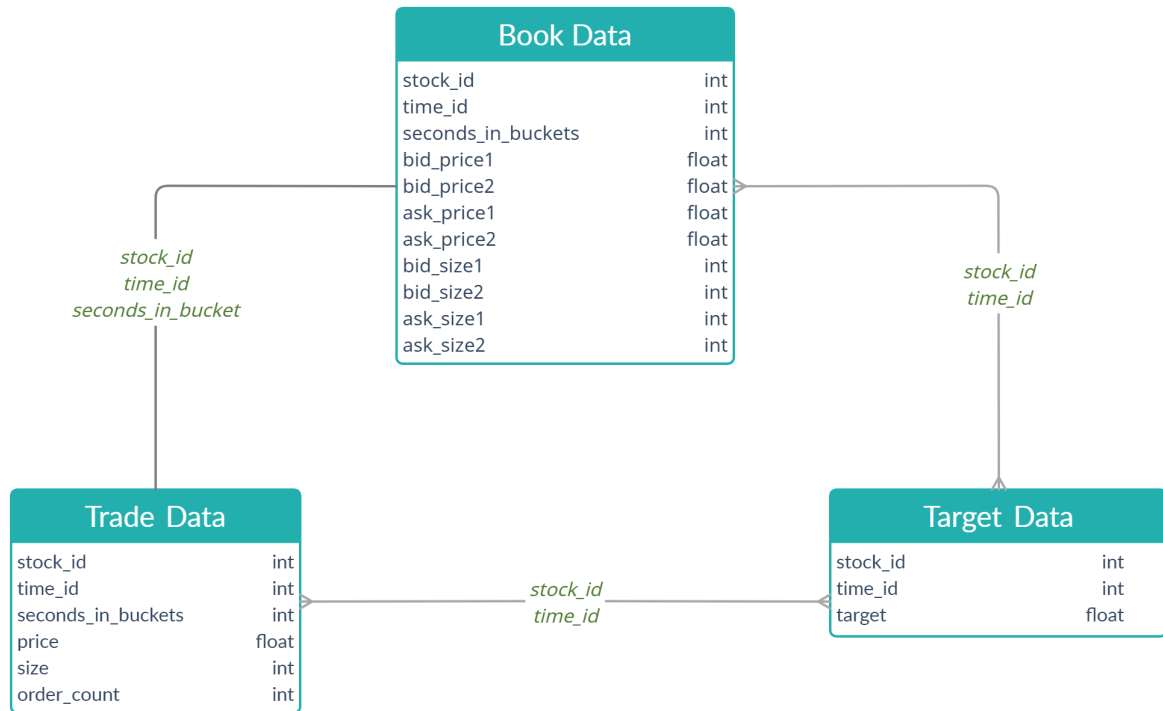
- Realized volatility has a backward-looking approach while the implied volatility is more forward-looking.

## Data

The data can be interpreted as snapshots of an order book and the actual trading due to those order book configurations. The time difference within two consecutive snapshots is one second.

## Data files

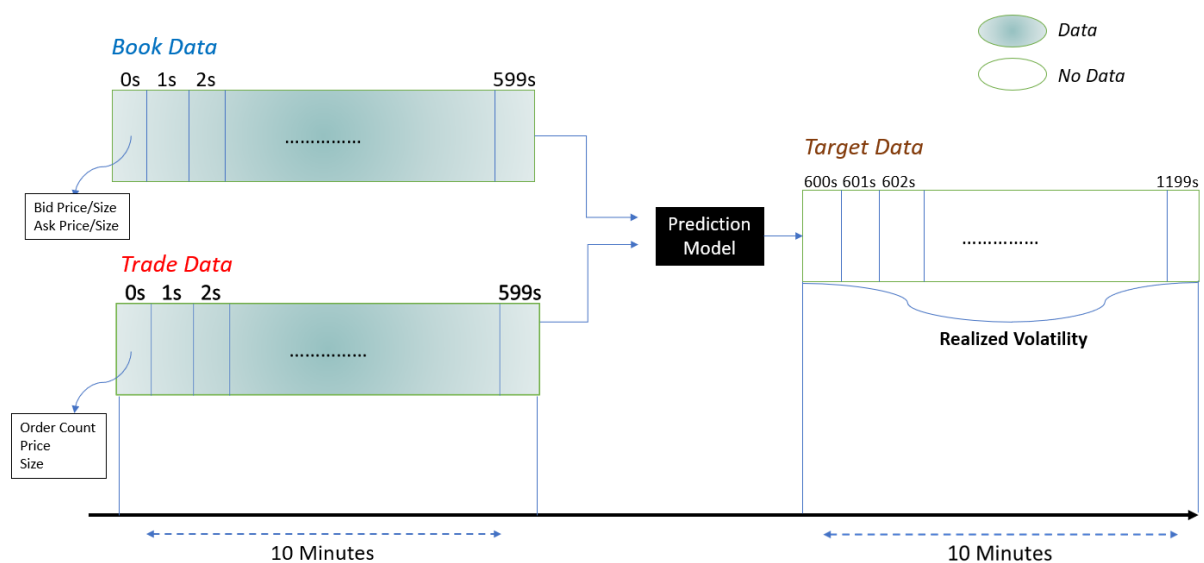
There are three types of data sets connected via few keys given for both training and testing.



## Train Data

As mentioned above, there are three types of data sets. Feature data is present in *book data* and *trade data*. The target variable (volatility) is given in the *target data*.

For each stock, there are several windows of feature data available. For a particular stock (defined by stock\_id), if 10 windows (10 time\_id's) of feature data are given, we would have to predict 10 realized volatility values for each time\_id.



For a particular **stock\_id-time\_id** pair



## Variables/Features Description

**Book Data:** It is the "snapshot" of an order book for a particular stock.

- `stock_id`: ID code for the stock.
- `time_id`: ID code for the time bucket. Time IDs are not necessarily sequential but are consistent across all stocks.
- `seconds_in_bucket`: Number of seconds from the start of the bucket, always starting from 0.
- `bid_price[1/2]`: Normalized prices of the most/second most competitive buy level.
- `ask_price[1/2]`: Normalized prices of the most/second most competitive sell level.
- `bid_size[1/2]`: The number of shares on the most/second most competitive buy level.
- `ask_size[1/2]`: The number of shares on the most/second most competitive sell level.

**Trade Data:** It is the result of actual trades that happened

- `stock_id`: Same as above.
- `time_id`: Same as above.
- `seconds_in_bucket`: Same as above. Trade data is more sparse because there are instances where no trade happened for a particular stock in a particular time window. As a consequence, this field is not necessarily starting from 0.
- `price`: The average price of executed transactions happening in one second. Prices have been normalized, and the average has been weighted by the number of shares traded in each transaction.

$$Price = \frac{w_1 p_1 + w_2 p_2 + \dots + w_n p_n}{w_1 + w_2 + \dots + w_n}$$

Where  $w_i$  is the number of shares traded at a price (normalized price) level  $p_i$

- `size`: The sum number of shares traded.

$$Size = w_1 + w_2 + \dots + w_n$$

- `order_count`: The number of unique trade orders taking place.

$$Order\ Count = n$$

**Target Data:** It is the ground truth values for each `stock_id` - `time_id` pair.

- `stock_id` : Same as above.
- `time_id` : Same as above.
- `target` : The realized volatility computed over the 10-minute window following the same stock/time\_id feature data. There is no overlap between feature and target data.

## Test Data

It is the same as the training data without the target variable in the target data, which we have to predict.

# EDA and Feature Engineering

## Basic Information

### Data Shape

- Total number of different stocks: 112
- Every stock has approximately 3830-time windows.

Summing up the above two points:

Every `stock_id` has 3830 corresponding `time_id` associated with it.

- Therefore, the total number of rows in the data is  $112 \times 3830 = 428960$

### Data Snapshot

- A look at the **book data** (for stock id - 0):

	time_id	seconds_in_bucket	bid_price1	ask_price1	bid_price2	ask_price2	bid_size1	ask_size1	bid_size2	ask_size2
0	5	0	1.001422	1.002301	1.00137	1.002353	3	226	2	100
1	5	1	1.001422	1.002301	1.00137	1.002353	3	100	2	100
2	5	5	1.001422	1.002301	1.00137	1.002405	3	100	2	100
3	5	6	1.001422	1.002301	1.00137	1.002405	3	126	2	100
4	5	7	1.001422	1.002301	1.00137	1.002405	3	126	2	100

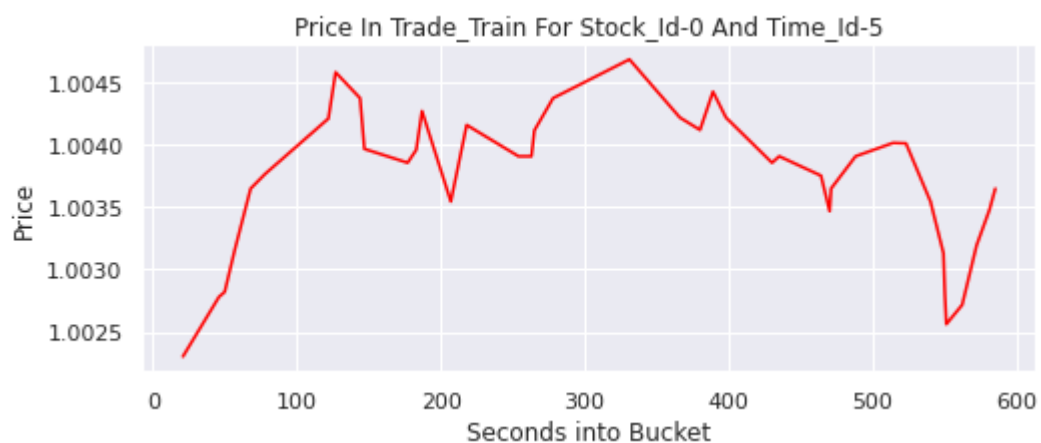
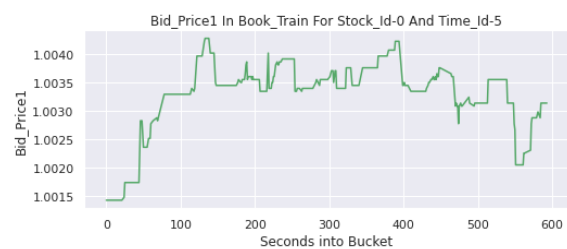
- A look at the **trade data** (for stock id - 0):

	time_id	seconds_in_bucket	price	size	order_count
	0	5	21	1.002301	326
	1	5	46	1.002778	128
	2	5	50	1.002818	55
	3	5	57	1.003155	121
	4	5	68	1.003646	4

## Window Plots

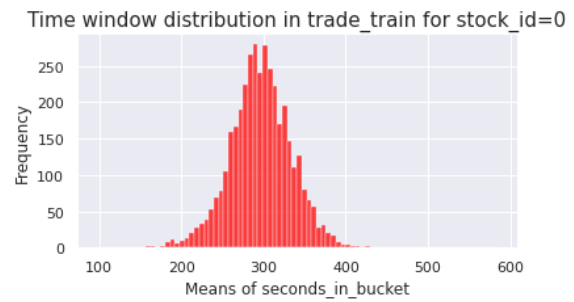
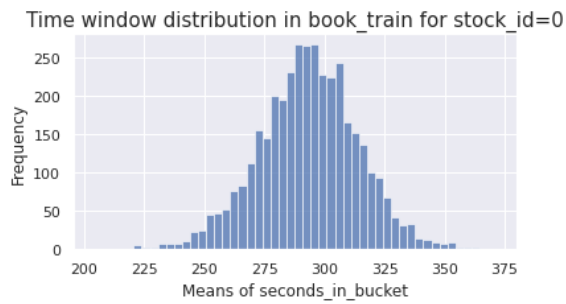
These plots represent the data in its most granular form. The Plots will correspond to a particular time id for a particular stock id.

Few Examples:



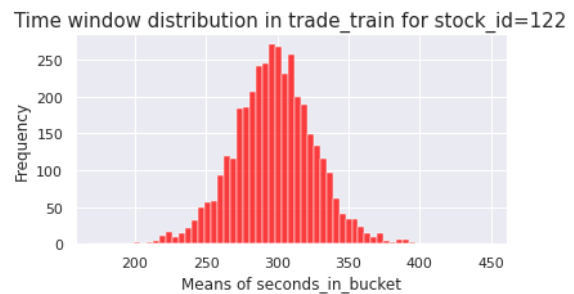
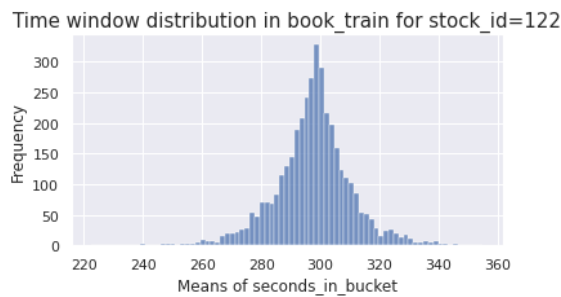
## Stock Plots

Plots that summarize data in a particular stock id.



Clearly, trade data is more sparse (ranging from 200-400) than book data, as stated earlier.

One more example,







The stock IDs 0 and 122 were chosen randomly, mainly to show sparsity in trade data.

## Feature Extraction





Basic features extracted directly from the book and trade data:

### Book Data

Base Features	New Feature	Number of Features	Small Note	Feature Type
<u>Bid and Ask Prices/Sizes</u>	Weighted Average Price (WAP)	2	Representative Price for a Stock	Granular Data
<u>Bid and Ask Prices</u>	Different Spreads	3	Measure of Liquidity	Granular Data
<u>Bid and Ask Prices/Sizes</u>	Log Return	2	Used to calculate Volatility	Granular Data

<u>Aa</u> Base Features	 New Feature	 Number of Features	 Small Note	 Feature Type
<u>Bid and Ask Prices/Sizes</u>	Volume imbalance	2	Measure of Liquidity	Granular Data
<u>WAP + Spreads + Log Returns + Vol Imbalance</u>	Mean and Standard deviation	18		Window Statistic
<u>Log returns</u>	Realized Volatility	2	History Volatility is also an indicator	Window Statistic

## Trading Data

<u>Aa</u> Base Feature	 New Feature	 Number of Features	 Small Note	 Feature Type
<u>Price</u>	Bollinger Bands	3	A volatility indicator	Window Statistic
<u>Price</u>	Donchain channels	3	A volatility indicator	Window Statistic
<u>Price</u>	Ichimoku CClouds	1	A volatility indicator	Window Statistic
<u>Size</u>	Mean, Standard deviation and Sum	3	Represents total trades happened in the window	Window Statistic
<u>Order Count</u>	Mean and Median	2		Window Statistic
<u>Seconds in Bucket</u>	Number of Unique Values	1	Represents total number of seconds in which trade happened	Window Statistic

## Lag Features

- Window statistic features are calculated with different window sizes.

For example, for a 3 window input, the window ranges are as follows

- 0 to 599 (complete window)
- 200 to 599 (2nd window)
- 400 to 599 (3rd window)

This approach is common in forecasting problems and the features generated are usually referred to as *lag features*.

## KMean as a Feature Engine

In practical model fitting, the KMean algorithm can be used to add "local knowledge" in a staged process with other classification techniques. In this project, KMeans is incorporated as follows into the pipeline:

1. KMeans is fitted on the data, and for each record, a classification is derived.
2. That result is added as a new feature to the record and used to aggregate features.

Since stocks can be classified into different sectors, KMeans can try to model these sectors and give an extra hand on the dataset. At first, this might look like a problem of multicollinearity which would hurt the later parts of the pipeline. This is not a problem since the base model in the boosting algorithm is decision trees that are capable of handling multicollinearity efficiently.

# Training and Evaluation

## Cross-Validation

To avoid overfitting and make efficient predictions with the unseen data, a cross-validation strategy is applied. Different cross-validation strategies used in this project are given below:

- **K-Fold CV:** This validation scheme is the simplest of all. We have split our data into K equivalent folds and train a model by taking the 1st fold as the validation set and the rest k-1 folds as the training set. This process is repeated by taking the 2nd fold as the validation set, the 3rd fold, and so on.

We end up with K models trained on different subsets of the data. Since it's a regression problem, we take the mean of all the validation scores which will give us the final *out-of-fold* (oof) score. This oof score is much robust than an evaluation done without cross-validation.

- **Group K-Fold CV:** It is also a K-Fold iterator but without overlapping groups. This group parameter is usually a column.

In our case, we need to make sure that the folds are divided without overlapping `time_id`'s because a `time_id` present in both train and validation sets would result in data leakage between the two sets.

## Model Selection

The two obvious choices of models for such structured data problems are XGBoost and **LightGBM**. They provide efficient results in big data problems. **LightGBM** is a gradient boosting framework that uses tree-based learning algorithms.

LightGBM advantages over other classical machine learning frameworks:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel, distributed, and GPU learning.
- Capable of handling large-scale data.

This project has used **LightGBM** as the base model.

## Training

After a lot of experimentation with different combinations of features and hyperparameters the final configuration is as follows

```
lgb_params = {  
    "learning_rate": 0.135724,  
    "min_data_in_leaf" : 690,  
    "num_leaves" : 769,  
    "bagging_fraction": 0.972676,  
    "num_features": 486,  
    "objective": "rmse"}  
FOLDS = 5  
nWindows = 4 # For lag features
```

## Bayesian Optimization

To select optimal parameters, we need to conduct hyperparameter tuning.

**GridSearchCV** takes too long since this tests all possible combinations of parameters. **RandomSearchCV** takes less time but it chooses a set of parameters randomly(does not test all combinations of parameters), the selected parameter may

not be an optimal parameter. Both algorithms do not contain prior knowledge information.

Bayesian Optimization keeps track of past evaluation results which they use to form a probabilistic model mapping hyperparameters to a probability of a score on the objective function. Also, it's faster than GridSearchCV and more precise than RandomSearchCV.

## Neptune

Like every Machine learning project, a lot of experimentation was done with different approaches and parameters. To organize the experimentation process in one place, we used Neptune, an online platform that helps to manage the experimentation process in machine learning projects. It can be linked to a Jupiter notebook using Neptune API.

# Results

## RMSPE

- Train rmspe score: **0.205303**
- Validation rmspe socre: **0.227225**