Shira Rozenthal and Alex Olteanu

Professor Maus

CMPS4010

Due November 9th, 2023

**Milestone 3**

Our capstone project focuses on tackling [Kaggle competition](https://www.kaggle.com/competitions/optiver-realized-volatility-prediction/overview) initiated by [Optiver](https://optiver.com/), a prominent electronic market maker. Optiver released extensive high-frequency market data, challenging participants to enhance pricing algorithms by predicting short-term volatility based on order book and trade information. The dataset includes order book details, executed trades, and training data with target realized volatility, while the goal is to accurately predict stock volatility in the 10-minute window following a market order using Root Mean Square Percent Error (RMSPE) as the evaluation metric.

In this Milestone, we will review the progress we’ve made working with high-frequency data, the limitations we’ve run into, and how we plan to move forward into model selection. Access to our notebook can be found [here](https://github.com/aolteanu00/CMPS-4010-Capstone) on our github.

**Progress Made: Data Access & EDA**

Optiver’s data is stored in parquet files, labeled by stockID. Given that we are working with over a year’s worth for stocks in every industry across the market, accessing non-sequential trade data one stock at a time isn’t a viable option.

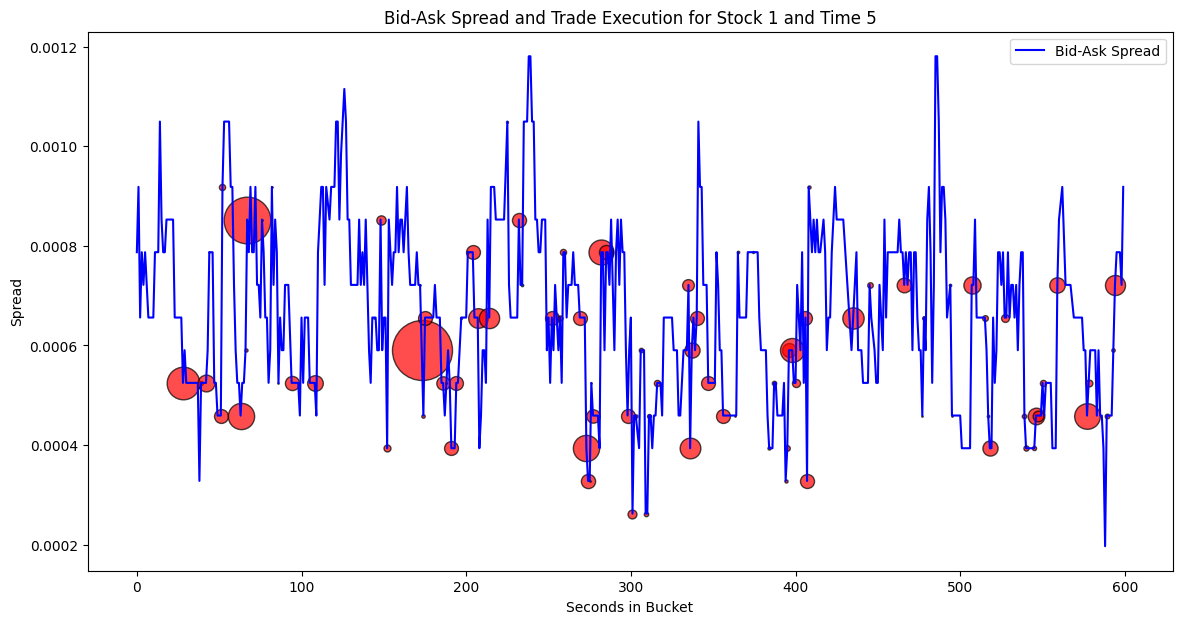
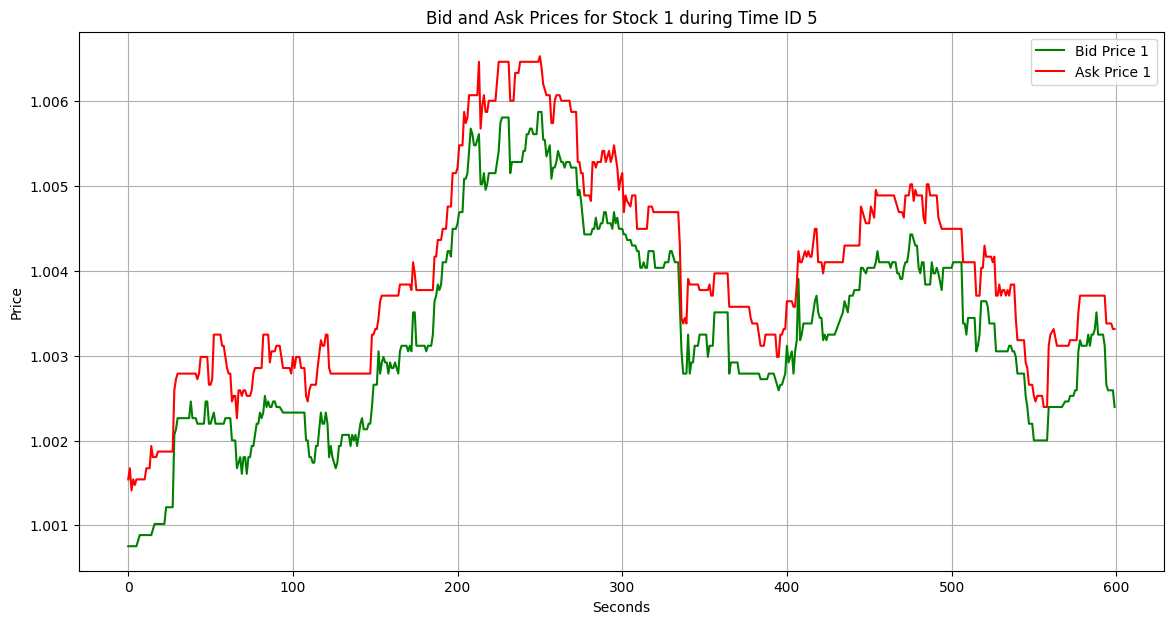
Accordingly, the first thing we did was define a bunch of functions for readily extracting layers of a parquet file. Found under the EDA tab in our notebook, these functions include:

* read\_parquet\_file(data\_type, stock\_id)
* filter\_data(df, time\_id=None, seconds\_in\_bucket=None)
* extract\_data(data\_type, stock\_id, time\_id=None, seconds\_in\_bucket=None)

Without going too much into depth here, these are cool because now we can access our data.

In addition to data access functions, we conducted a bit of EDA and built out functions that help us visualize market interest. These functions include:

* plot\_trading\_volume(stock\_id, time\_id)
* plot\_bid\_ask\_volume(stock\_id, time\_id)
* plot\_bid\_ask\_price(stock\_id, time\_id)
* visualize\_spread\_and\_trades(stock\_id, time\_id)

These functions gave us a grasp on how our seemingly random and never ending points together become market data. In the two plots below, you can see bids and asks (volume), how the spreads between them move, and the where within that spread trades got executed. 

**Progress Made: Reverse-Engineering TimeID**

The first step of our feature engineering journey is to be able to look at this data chronologically. Not only are the timeIDs ambiguous, they're also randomly labeled. In chronological order, the first 5 timeIDs are: 4294, 24033, 5666, 29740, 22178.

Although chronologically-sequenced data won’t be of help to our model because volatility is a non-stationary series, we need it in order to:

1. **Get an accurate representation of our test error.**

The kaggle competition uses future data as test data, so we should train on older train data and test on newer train data in order to have a better estimate of our model’s performance both on the leaderboard as well as moving forward in time.

1. **Build features around the relationship between “neighboring” timeIDs.**

Understanding the relative positioning of each timeID enables us to train our model on how volatility behaves inter-day. Ex. how does the # of bids/asks at timeID(x) relate to where volatility realizes at timeID(x+1)? (this should be significant, more interest → more trades → more movement in price). We intend to look not only at consecutive timeIDs but to implement a nearest-neighbor approach to how timeIDs relate to each other's targets.

It’s crucial to note that our inspiration (as well as that of many other top submissions) for this approach should be entirely accredited to the [top submission](https://www.kaggle.com/competitions/optiver-realized-volatility-prediction/discussion/274970), whose explanation and code excerpts provided so much clarity for us and will act as the foundation for our model.

We have timeID reverse-engineered (the code for this is under its own folder in our notebook), and have built some additional data-access functions to extract features by timeID as we move into more feature engineering.

**Moving Forward: Basic Model & More Feature Engineering**

Now that we can effectively assess our model, we want to get one running. After speaking with Prof. Hamm, we’re going to start off with a simple K-NN model rather than trying to match some of the deep learning approaches implemented by top submissions off the bat. In accordance with our timeline from last month’s presentation, the goal for Milestone 4 is to have this running, and to improve test error as best we can by feature engineering the relationship between timeIDs.

This leaves us in a good spot for next semester, where we can pick up with the implementation of higher-level models and continue feature engineering to find a combination that works best.



We have been distributing the workload well so far, and much of what we’ve accomplished has been together and in-person. Alex has been specifically on top of programming the data access while Shira has been more focused on research and understanding the volatility space. Moving forward, we plan to independently build features and come together to see which ones work.

**Moving Forward: Meetings with Mentor**

We plan to stay consistent with our Wednesday afternoon meetings with professor Hamm for the rest of the semester. We’ll meet with him next week before the break with a running K-NN model in order to determine how we should go about improving it, and once more after we return from the holidays but before our Milestone 4 deadline.