**DECISION 546Q Team Project Final Report**

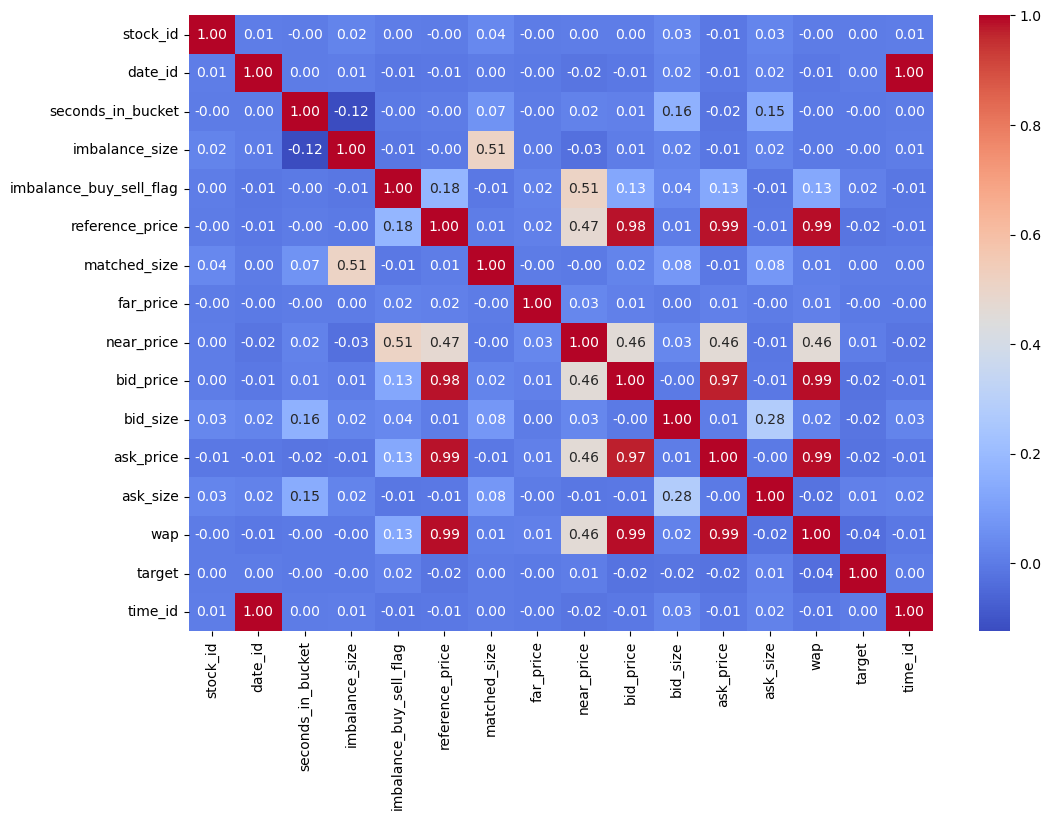
Section A Team 9

Yan Guan(yg238), Erik Henig(eh307), Jingjing Hu(jh892), Chuangfa Liang (cl670), Nupur Shah(ns417)

**I. Motivation and Data Understanding**

In our project, we're addressing the challenge of predicting closing price movements for Nasdaq-listed stocks in the critical final ten minutes of trading. This period is known for its heightened volatility and rapid price shifts. We aim to develop a model that effectively combines order book and auction data to provide accurate price forecasts. The model will aid market makers like Optiver enhance market efficiency and contribute to more reliable trading strategies for various market participants. This project also provides valuable exposure to solving real-world data science challenges, particularly in the dynamic and high-stakes realm of stock trading.

This dataset contains historical data for the daily ten-minute closing auction on the NASDAQ stock exchange. The data structure presents itself as a time series issue, progressing in 10-second intervals, yet it deviates from conventional time series characteristics. This dataset encompasses 200 stocks simultaneously fluctuating over 481 days, with each day comprising 540 seconds of activity. Our inspection of the columns yielded that the far\_price and near\_price columns contain more than 50% of NA values. We should carefully deal with this situation by diving deeper into the dataset.

 The above heatmap, also called Correlation Matrix, is like a scorecard showing which information pieces are connected. When two things are connected, they either go up or one goes up when the other goes down. Bright red spots show things that move together a lot, and deep blue areas show things that don't. By looking at this, we can determine which pieces of information tend to change under similar conditions, which helps make decisions where these connections matter to avoid multicollinearity.

Finally, in terms of the data source, we obtained all of the data from Kaggle (Optiver, 2023).

**II. Data Preparation**

The training dataset is vast, with 5.2 million rows and 17 columns (including row\_id). The data's format presents a time series challenge involving 200 stocks, spanning 481 days, and segmented into 55 unique-second intervals within each bucket.

The first step of our data preparation was to reduce the memory usage of this cumbersome dataset. We created a function designed to optimize the memory usage of a DataFrame by downcasting numeric columns to more memory-efficient data types without losing information. It works by checking each numeric column to find the smallest data type that can fit the range of values in the column. If a smaller data type can represent a column's values, the function will convert it to that type. It is helpful, especially when dealing with a massive dataset like this. After using this function, our memory was decreased by roughly 51.20%.

We then handled missing values in the target, imbalance\_size, reference\_price, matched\_size, far\_price, near\_price, bid\_price, ask\_price, and wap columns using imputation techniques. We have broken down the techniques used below.

Target: Since this is a time series problem, we want to keep all the information. Thus, we filled the missed “target” variable (count==88) with the mean of this specific stock for each missing value.

Price variables: The variables are “reference\_price,” “far\_price,” “near\_price,” “bid\_price,” “ask\_price,” “wap.” We filled them with an integer 1 to those variables defined as a price because it does not make sense that the price goes down to 0.

Volume variables: “matched\_size” and “imbalance\_size.” We filled them with the median of this specific stock for each missing value. We used median, not mean, because, based on their distribution, it does not seem like a normal distribution. Thus, the median is a better imputation strategy.

After the imputation, we sorted the data using chronological order (date\_id, seconds\_in\_bucket). The variable stock\_id is meaningless here since it does not provide any insights on the movement of the stock price along with the time series.

To prepare our data for deep learning modeling, we added features to capture specific aspects of the trading dynamics that might not be obvious from the raw data alone. We spent more than a week repeatedly testing the model with new features to determine the value of the four features below, ensuring they provided meaningful information rather than just adding noise to the model.

imbalance\_ratio: This feature represents the ratio of the size of imbalances to the extent of matches. In trading, an "imbalance" refers to the difference between buy and sell orders at a particular price level. A high imbalance ratio could indicate a robust directional interest (buying or selling pressure), potentially moving the market price.

imbalance: This feature is the product of the size of the imbalance and a flag indicating whether the imbalance is on the buy or sell sides. The imbalance feature could help determine the magnitude of the imbalance and its direction - whether the pressure is to buy or sell.

ordersize\_imbalance: This feature calculates the net order size imbalance as the difference between bid size and ask size relative to the total size (sum of bid and ask size). It measures which side of the market is more aggressive. For example, a positive value indicates more pressure to buy, while a negative value suggests more pressure to sell.

matching\_imbalance: This feature calculates the net imbalance between the size of imbalances and the size of matches relative to their sum. The matching\_imbalance could reflect the ease or difficulty of order execution - a high value might indicate that many orders need to be matched (which could suggest a lack of liquidity or a fast-moving market). In contrast, a low value might suggest that most orders find matches easily.

**III. Modeling**

Since this is a time series problem with 200 stock\_id, 481 date\_id, and 55 seconds in the bucket, we decided to use what we learned in the class, the RNN, LSTM, AND GRU models. We ultimately use the LGBM model for the active competition code, but it is beyond the main topics of this course. Thus, we will emphasize the modeling part mainly on the neural network models.

|  |  |  |
| --- | --- | --- |
| Selected Model | Pros | Cons |
| Linear Regression | Simplicity and can serve as a baseline model | Limited to a linear relationship |
| RNN | Good with sequence data like time series or text. | Struggles with long-range dependencies due to vanishing gradients. |
| LSTM | Can capture long-term dependencies in sequences effectively | More complex and computationally expensive than RNNs and GRUs |
| GRU | Efficient and faster to train compared to LSTM | It might not capture the long dependencies as the LSTM did |
| LGBM | Highly efficient and fast, especially for large datasets and high-dimensional data | Not eligible for this project |

In addition, in our project, choosing the optimal hyperparameters was a critical step to ensure the effectiveness of our model. We employed a systematic approach to tune hyperparameters by exploring various potential combinations. Our process involved defining a range of values for each hyperparameter, including input size, hidden layer size, number of layers, output size, and learning rate. For each combination of hyperparameters, we trained the model using our training data and then evaluated its performance on a validation set. The parimary metric for comparison was the validation loss, which reflects the model's accuracy in making predictions on unseen data.

By iterating through all possible combinations, we identified the set of hyperparameters (listed later in the Implementation section) that resulted in the lowest validation loss, indicating the most accurate model performance. This exhaustive exploration, although computationally intensive, ensured that we did not overlook any combination that could potentially generate the best results. This rigorous approach to hyperparameter tuning has allowed us to refine our model to the point where it achieves the best balance between complexity and predictive power, as evidenced by its performance on the validation dataset.

**IV. Implementation**

In our deep learning model implementation, a critical aspect was monitoring performance metrics, mainly focusing on Mean Absolute Error (MAE). We utilized this metric as a primary indicator of model accuracy, guiding our iterative refinement process. Throughout the model training, we continuously logged the MAE at each epoch for each variant – from basic RNNs to advanced Transformer models. This ongoing monitoring allowed us to identify when a model began to overfit, as indicated by a decrease in training error but an increase or plateau in validation error. Additionally, tracking the MAE facilitated the comparison of different model architectures and configurations. It was instrumental in determining the effectiveness of adding features and hyper-tuning parameters. For instance, when we compared the MAE of a standard RNN to that of an LSTM with added features, we could quantitatively assess the impact of these features on the model's predictive capability. This focus on MAE also played a crucial role in hyperparameter tuning. By evaluating how different settings – such as learning rates, number of layers, and dropout rates – influenced the MAE, we could fine-tune our models for optimal performance. The process involved adjusting hyperparameters based on their performance across training and validation datasets, balancing model complexity and practical efficacy.

In summary, by centering our implementation strategy on the careful monitoring of MAE, we achieved a data-driven approach to model development. The MAE monitoring enhanced the accuracy of our deep learning models and ensured their robustness and generalizability to new data.

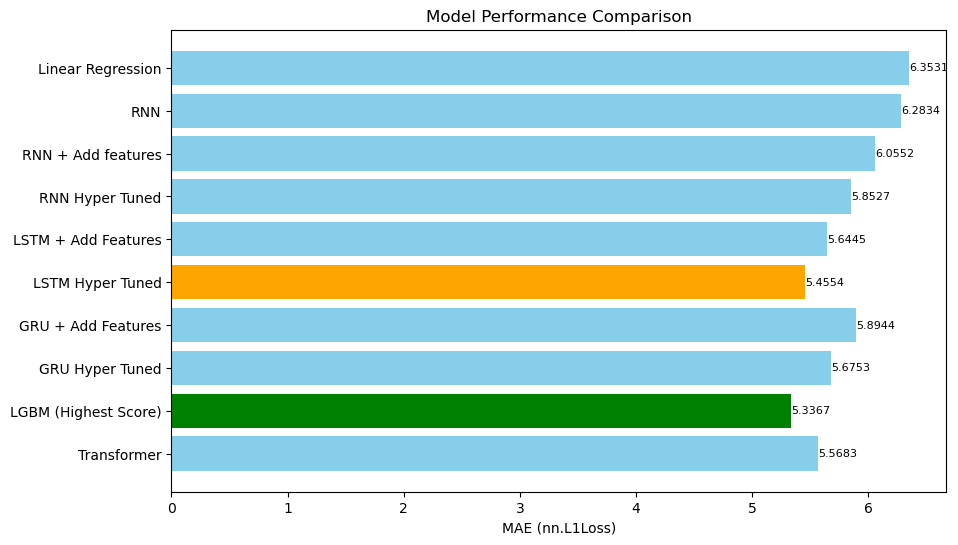
To conclude, the optimal parameters for our best Long-Short Term Memory model are:

|  |  |  |  |
| --- | --- | --- | --- |
| Learning Rate = 0.0001 | Input Dim = 17 | Hidden Size = 32 | Number of Layers = 3 |
| Activation Function = Relu | Epochs = 582 | Batch Size = 512 | Optimizer = Adam |
| L2 Regularization = 0.00001 | Dropout = 0.01 | Output Size = 1 |  |

The challenges we met in our implementation and solutions begin with aggregating code from different people and places. Some prefer to use local cloud-based VSCode, and others prefer Google Colab. A significant challenge in this project is the extensive dataset, which demands considerable training and feature engineering time. Since some of us work on a MacBook, PyTorch limits our ability to leverage Nvidia CUDA GPU processing. Unfortunately, our MacBook's AMD GPU isn't compatible with PyTorch, resulting in prolonged run times. While Google Colab is an option, its free version restricts GPU usage. Therefore, as a workaround, we develop the code on our local computers and then transfer it to Kaggle Notebooks, where we can utilize their T4 GPU. This approach significantly improves our code's execution speed.

**V. Results and Evaluation**

To understand how a business should evaluate the result, we want to demonstrate the segmentation of our data frame. Our DataFrame was segmented into sequences, each spanning 20 rows, equivalent to 200 seconds of historical data. For every 20-row sequence, the first 19 rows are utilized as a feature, while the final row serves as the label or target. Thus, each sequence aims to predict the outcome of its last row. After segmenting sequences, we further implicate the hold-out strategy for test/validation purposes. We have split the data into 8:2 for training and tests.



We can see from the above picture that the best performance model is the LGBM model with the lowest loss of 5.3367 (Currently positioned at 490 out of 4044 teams/ 5097 competitors on Kaggle's public leaderboard)

A business case for our model is grounded in its potential to revolutionize decision-making in the stock market. Accurate stock closing price predictions can significantly benefit stock exchanges, hedge funds, and market makers. Exchanges can leverage this model to enhance market stability, while trading firms can use it for risk management and identifying trading opportunities. Regulatory bodies can also utilize it for monitoring market dynamics, ensuring a fair and orderly market environment. Thus, our model offers substantial value across different financial trading and regulation facets.

**VI. Deployment**

Our data mining results are a sophisticated analytical tool within trading systems. This tool will analyze order book and auction data in real-time to forecast closing prices of Nasdaq stocks in the last ten minutes of trading. It will be integrated into the existing infrastructure of financial institutions, providing them with real-time predictive insights. We would also like to develop an experimental testing lab to tweak the model without dismantling the entire data pipeline.

When deploying our predictive model for Nasdaq stock closing prices, the firm must know several vital issues. First and foremost, ensuring seamless technical integration with existing systems while maintaining high data security standards is essential. The model must comply with all relevant financial regulations, and its performance needs to be scalable and reliable under high-frequency trading conditions. Additionally, accuracy in predictions is critical, so regular model updates and maintenance are necessary to keep up with market changes.

In addition to these considerations, we would also like to track any shifts in trading psychology as these prices improve. This would serve to mitigate any arbitrage opportunities we are not aware of and monitor any herd behavior of traders. Unlike the development's stable conditions, the model would need adjustments under a significant market shock. Finally, managing risks associated with potential technical failures and cultivating user trust through transparency and support are essential to ensuring the model is used ethically and responsibly in the market.

Fairness and transparency are essential ethical considerations in deploying our predictive model for Nasdaq stock closing prices. The model must avoid creating or exacerbating market imbalances or manipulations through its design. It's essential to ensure that the model's predictions use objective data analysis and do not include extreme biases or external interests. Transparency about how the model works, its limitations, and the data it uses is crucial to maintain trust among all market participants. Additionally, safeguarding the privacy and security of the model's sensitive data is a critical ethical obligation to prevent misuse or unauthorized access to financial information. In summary, the ethical deployment of this model requires a commitment to fairness, transparency, data privacy, and security.

In deploying our predictive model, we must navigate several risks. Market volatility challenges prediction accuracy, necessitating ongoing model updates and validation to adapt to changing conditions. Data security is paramount, as handling sensitive financial information requires robust cybersecurity measures and regular audits. Ensuring regulatory compliance is crucial to avoid legal repercussions, demanding continuous monitoring and adaptation to economic laws. Over-reliance on the model by users is another risk mitigated through comprehensive user education about the model's capabilities and limitations. Our group would also focus on creating a partner model that isn’t a neural network, allowing us to unwind the processes inside the model. Additionally, the risk of technical failures, such as system outages, requires the implementation of reliable backup systems and contingency plans. Addressing these risks with proactive strategies is vital to successfully deploying and operating our model in the stock trading arena.

**Appendix**

**I. Group Contributions**

|  |  |
| --- | --- |
| Motivation and Data Understanding | Erik, Yan, Nupur |
| Data Preparation | Jinjing, Nupur |
| Modeling | Erik, Chuangfa |
| Implementation | Chuangfa, Jingjing |
| Results and Evaluation | Chuangfa, Yan |
| Deployment | Erik, Jingjing |

**II. Bibliography**

Optiver. (2023, September 20). *Optiver - trading at the close*. Kaggle. https://www.kaggle.com/competitions/optiver-trading-at-the-close