Optiver Trading at the Close Competition

Predicting closing stock prices for the Nasdaq Stock Exchange

David Jackson

1. Introduction

Predicting the stock market has historically been a hard thing to do because of its complexity; there's an unknown number of variables to account for, which could be hundreds or thousands. Another reason it is challenging to predict is human nature and emotion, which cannot be realistically predicted at all. Some theories of why markets can't be predicted are the efficient market hypothesis and the random walk hypothesis. The efficient market hypothesis states that prices reflect all available information, which essentially means that assets will trade at fair value. The random walk hypothesis states that all price changes are random and, therefore, there is no way to predict them. In this competition, we will analyze the data using multiple different models, like Ordinary linear regression, XGBoost, ARMIA, GARCH, CatBoost, GRU, LSTM, and Torch. The data from Optiver contains 200 NASDAQ stocks in the last 10 minutes of the trading day, over 481 trading days, and each stock is broken up into individual seconds in the last 10 minutes of the day. Seconds in the Bucket refers to how many seconds have passed since the start of the last 10 minutes.

2. Objective

The goal of this competition is to understand time series analysis and gain insight into how different models work and how well they predict the data, and understand which models are better. For the competition, the goal is to have the smallest mean absolute error, which is the average of the absolute differences between predicted and actual values and is given by

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$$

Where n is the number of data points, y_i is the predicted value for data point i, and x_i is the observed value for data point i.

3. Collecting and Cleaning the Data

Collecting the data is very easy. It is provided by Optiver. As long as you have a Kaggle account, you can just download it for free and then import it into whatever application you use for coding. The first thing to do in cleaning the data is to forward fill all of the empty values in the data. These empty values could be caused by holidays or when markets are closed, so forward-filling them with the previous day's price makes the most sense. Then, to remove any other missing values, we will backfill values. The only reason there is a missing value is if it happened at the beginning of the data set, so there wasn't a previous value for it to forward fill. When we did the forward and backward fill, we excluded the target column, which is what we are trying to predict, and missing values should not be filled in, so we just removed the 88 rows with missing values from the target. This is all we need to do to clean the data, and we are ready to move on to the next stage.

4. Engineered features

Creating engineered features is important to capture complexities in the model. The original data set only contains some features like imbalance size, imbalance buy sell flag, reference price, far price, near price, and others, but to fully capture the mode, we want to add more features for better predictions. Here are the engineered features: near minus wap, far minus wap, near minus far, bid ask spread, midprice, wap dev mid, signed imbalance, imbalance ratio, seconds remaining, and lag to capture trends over the last 1 - 6 seconds for wap, near price, far price, signed imbalance, bid ask spread. Wap is the weighted average price in the non-auction book, and most of our features are the differences between prices, with some averages of prices or ratios. The lag features use the last one second, three seconds, and 6 seconds to try to predict future stock prices. We also added a wap moving average of 3 seconds. This uses the previous three seconds and then creates an average, then moves a second forward and repeats, giving us a window of averages. We also made a wap standard deviation window of 6 seconds to help capture risk in this window. Wap RSI is one of the features we made. The Relative Strength Index (RSI) is used to identify momentum to measure the speed and change of price movements. The last engineered feature we made was a log return of wap.

5. Exploratory Data Analysis

The first thing we did was find the skewness and Kurtosis of the data, which were

Skewness of Adj Close: 0.3755400704988964

Kurtosis of Adj Close: 10.886862797291803

This is the skewness and kurtosis of wap, which is the adjusted closing price. As we can see, it is slightly skewed to the right. This is only slightly skewed, so it's probably not a problem. Looking at the kurtosis, this value means it has a sharp peak and very heavy tails. This also means that the data has a lot of outliers. What this means the data will be very hard to predict the closing price movement of the stocks because of these outliers, making a model that can handle all of the values, including the outliers, will be difficult.

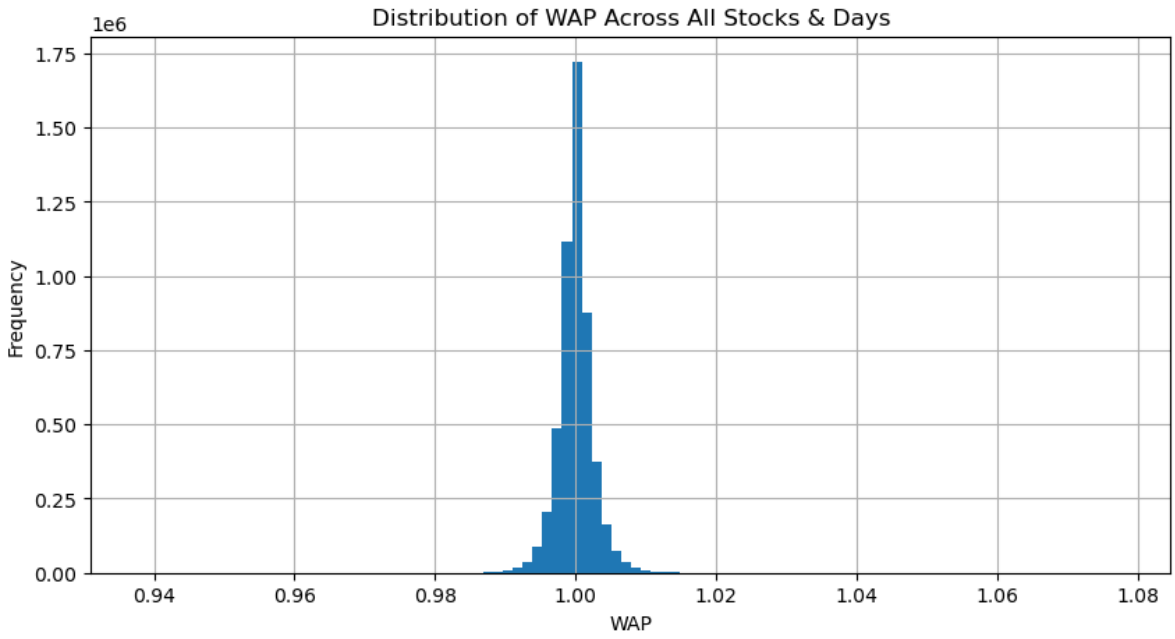

FIGURE 0.0.1.

Next, we made a plot of the imbalance by cell flag distribution, which represents the percent of the imbalance buy sell flag. -1 represents the sell flag, 1 represents the buy flag, and zero represents neither buy nor sell. For example, 40% of the data in the imbalance buy-sell flag is flagged to sell in the following plot.

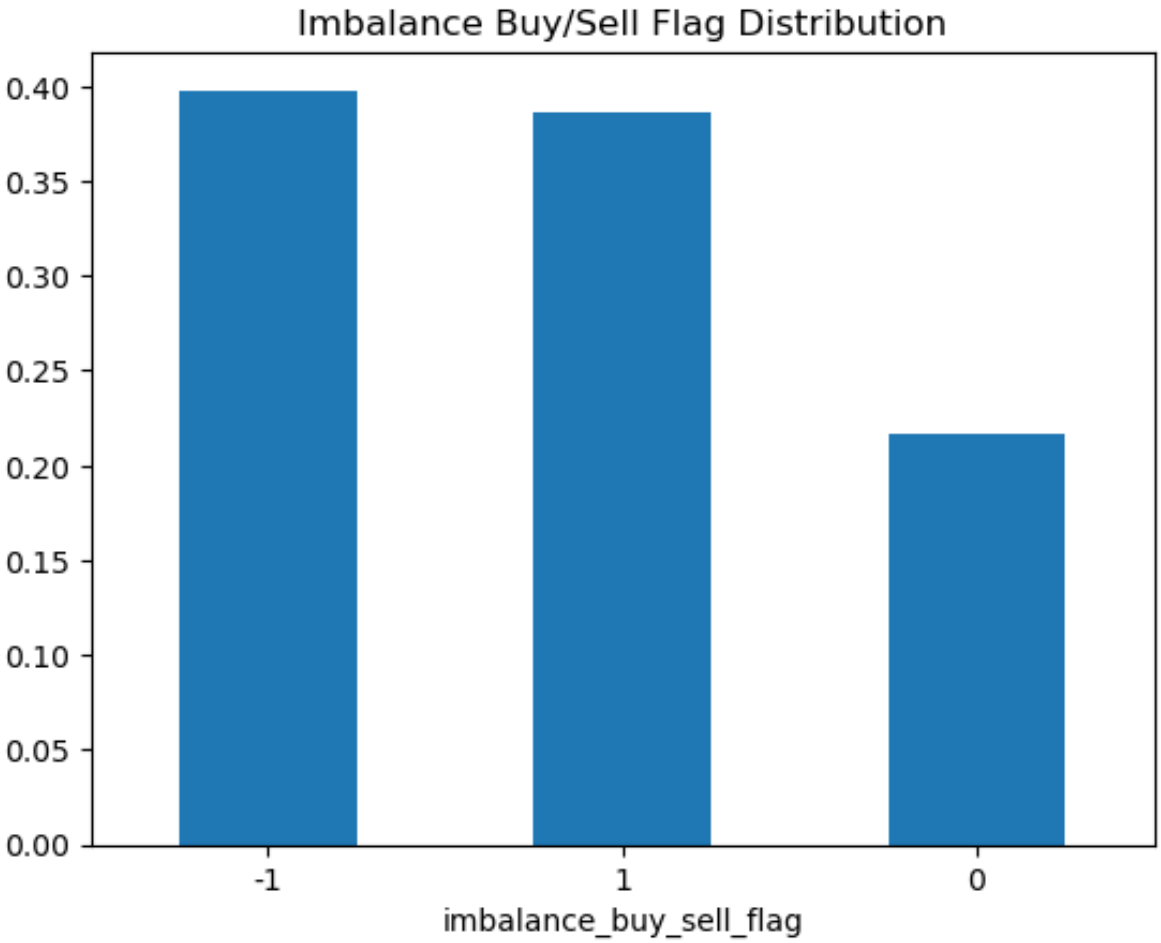

FIGURE 0.0.2.

We also want to know the distribution of the target, which is the price movement that we are trying to predict. The target is "The 60-second future move in the WAP of the stock, less the 60-second future move of the synthetic index."

- "The synthetic index is a custom weighted index of Nasdaq-listed stocks constructed by Optiver for this competition."
- "The unit of the target is basis points, which is a common unit of measurement in financial markets. $A1$ basis point price move is equivalent to a 0.01% price move."
- "Where t is the time at the current observation, we can define the target"

$$Target = \left(\frac{StockWAP_{t+60}}{StockWAP_t} + \frac{IndexWAP_{t+60}}{IndexWAP_t} \right)$$

Looking at the distribution given below, we can see it is very similar to the distribution of wap, which makes sense because the formula for the target is calculated using the wap closing price.

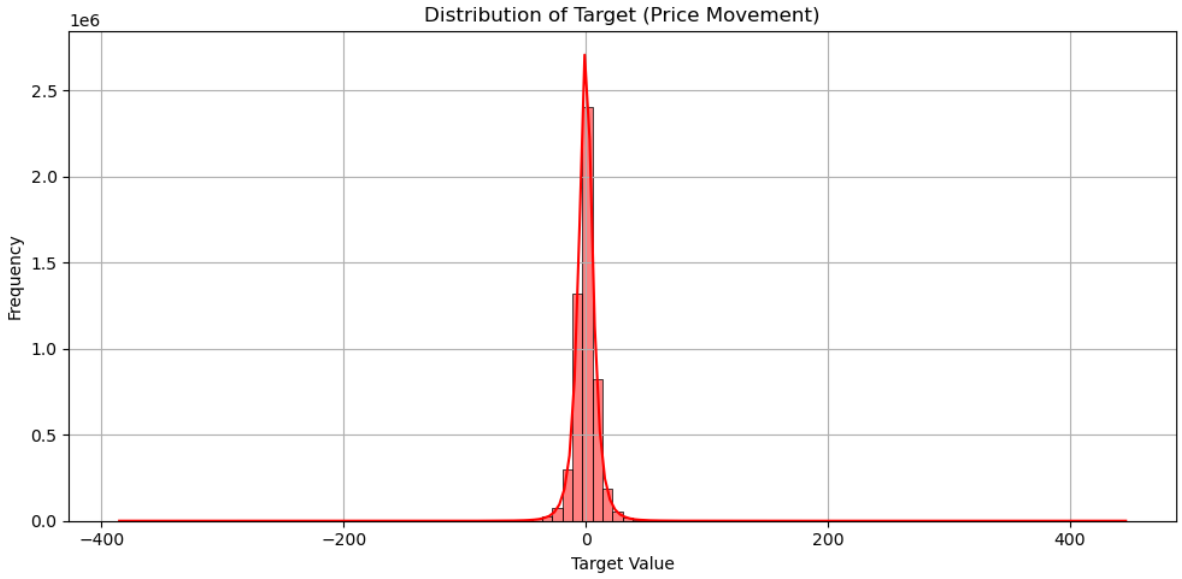

FIGURE 0.0.3.

For the next part of exploratory data analysis, we made an average log return per day plot. This will show us the noise in the model. This will also help us determine if the data is stationary. Looking at the plot given below, we can see that there isn't any clear trend

happening; the data is just bouncing around zero, which is good, as this means that the data is likely stationary. This is needed for time-series analysis.

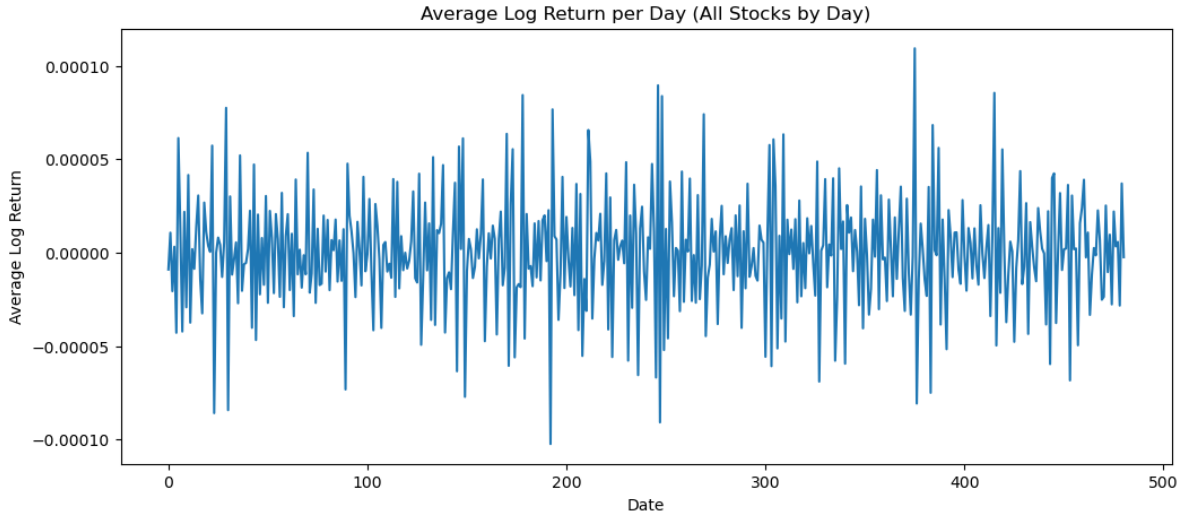

FIGURE 0.0.4.

We also made a pot of the average log return over the auction time. This takes the average of all 200 stocks' log returns for the first second of the auction and then does it for the 2nd second and so on for all 600 seconds in the auction. We can see that there is no clear trend in the plot, and it also bounces around zero. It also doesn't look as dense as the previous plot. This is because the previous plot calculates the log return of the 200 stocks for the entire auction on the first day, then repeats this process for the second day, and so on. The difference between the plots is that the first one is by day and the second one by seconds. Since both of these plots don't shows any clear trends and just bounces around zero, our data is probably stationary, which is good to confirm for our time series analysis later on. In finance, it is often safe to assume, or at least it is common practice to assume, that log returns are stationary, but it is always good to confirm.

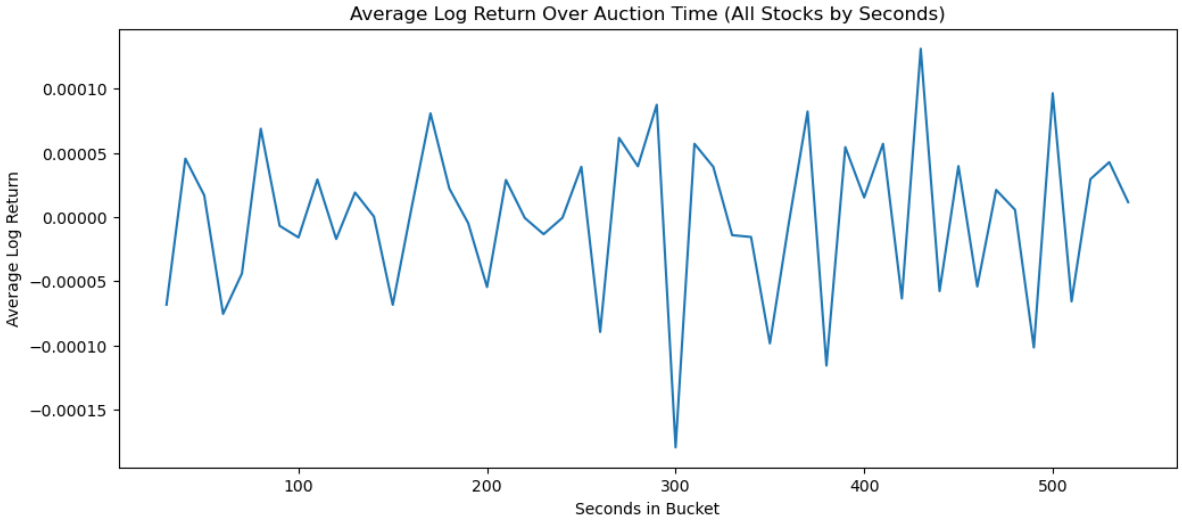

FIGURE 0.0.5.

We used one more plot to check for stationarity in our data set. The only problem with this plot is that it can only be applied to one stock and one day at a time. Still, we had it randomly choose a stock and a day, so each time you run the code, you get a different stock and a different day, so we can rerun it 10 to 20 times and look at the plots to get a sample of our data to see if these individual stocks are stationary and the overall data set. To do this, we made a plot of the log returns of an individual stock with rolling means and rolling standard deviations of the returns alongside it. But what we are looking for on these plots is to see that the rolling mean is a constant value, and we also want the same for the standard deviation. Are plot below has a pretty constant rolling mean. If you drew a line, a horizontal line, through it, that line would fit it well. Our standard deviation isn't as good; if we drew a horizontal line, it would fit, but not as well. To go along with this plot, we made an autocorrelation plot and a partial autocorrelation plot. The autocorrelation plot measures how a time series is related to its own past values. Having high autocorrelation is a good thing for time series analysis. This means we should be able to predict future values based on previous values. Looking at our plot below, we see a spike at zero seconds. This is expected because you're comparing the value to itself. We also have a spike that goes out of the blue shaded area, which is a 95% confidence

interval. This means that the spike is significant and suggests that if the log return goes up in this second, it should go down in the next second. We also have a partial autocorrelation plot, which measures the correlation of a time series with a lag after removing the effects of shorter lags. For example, PACF at 2 seconds shows the pure correlation between the current second and two seconds ago, controlling for 1 second ago. Looking at our plot, the spike at zero is caused by the same reason as before. The spike at 1 second indicates that lagging the data by one second will add the most significance to our model. This is confirmed by the autocorrelation plot. But remember, this is looking at only one stock in the data on one day, although we did rerun this plot multiple times to see what other stocks look like. This is still just a sample of the data, but it is still helpful to see what's happening to individual stocks and can give us insight into the general data set.

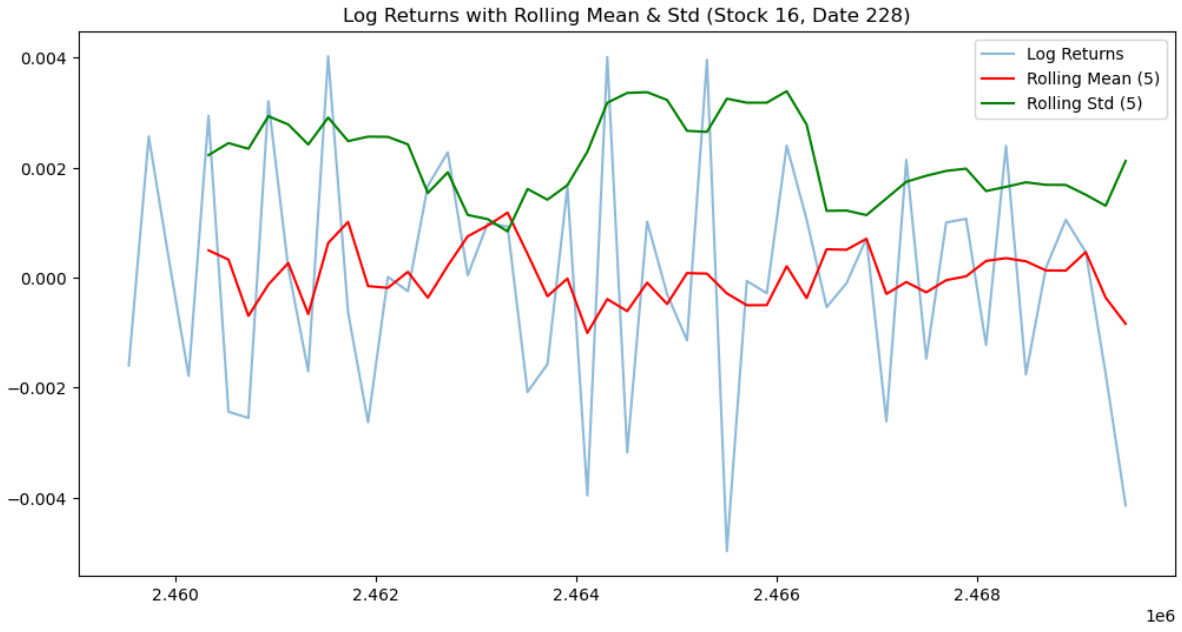

FIGURE 0.0.6.

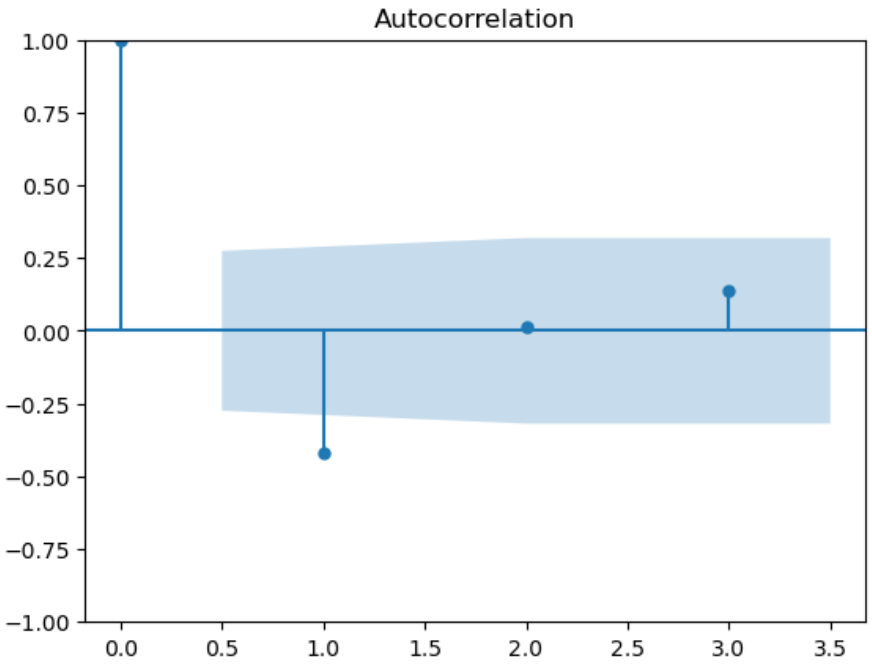

FIGURE 0.0.7.

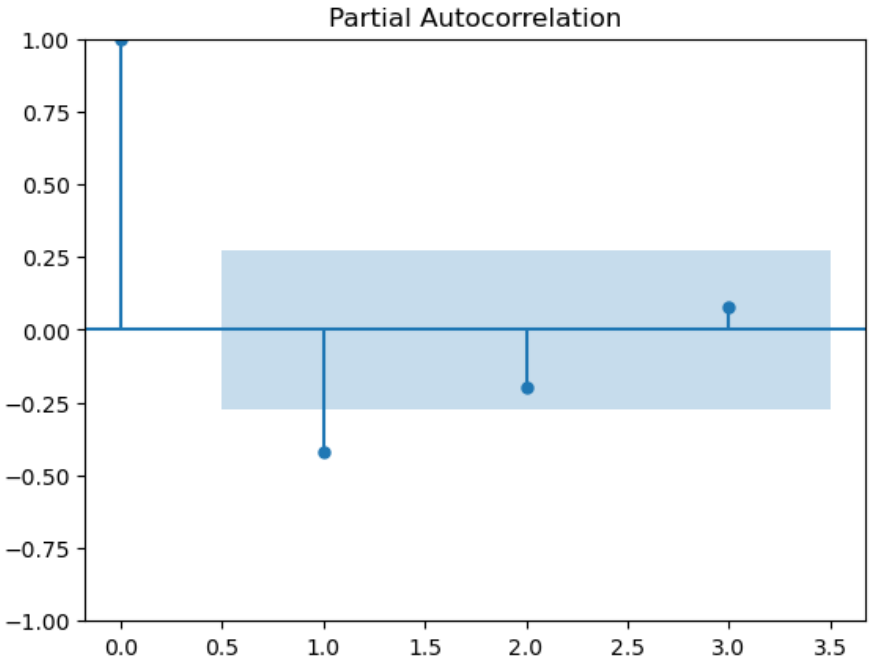

FIGURE 0.0.8.

In the next part of exploratory data analysis, we made a correlation Matrix. It is good to know what indicators have a high correlation with other indicators. If variables have high correlation with each other, they will tend to move in the same direction, and knowing this information can help us predict our Target in Time series analysis. Unlike normal linear regression models, which we will make one later on, multicollinearity can cause a problem for linear regression, especially when trying to interpret slope values for variables, but it does not matter if predicting is all that matters. For this competition, all that matters is predicting. We don't care about slope values, and most of the models we will use are nonlinear models because ordinary linear regression cannot account for all of the complexity and noise in the data set.

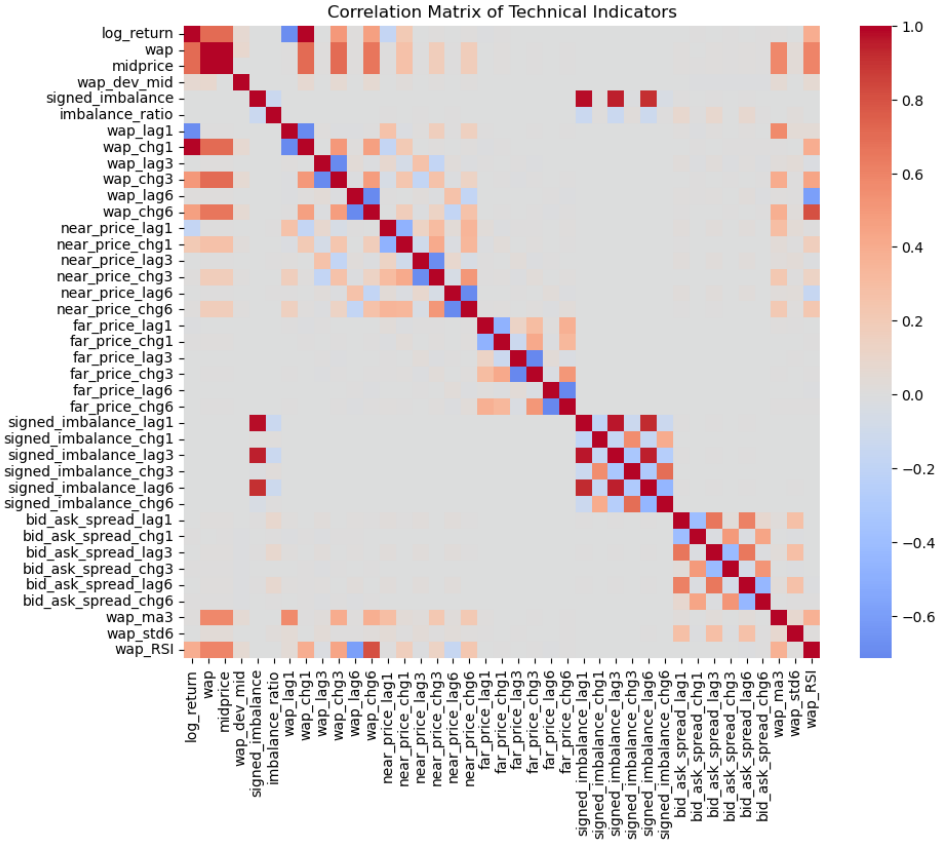

FIGURE 0.0.9.

To see the relationship between each of the indicators, we made a pairwise scatter plot for the technical indicator, and as you can see from looking at the plot below, most of these variables do not have a linear relationship with other variables.

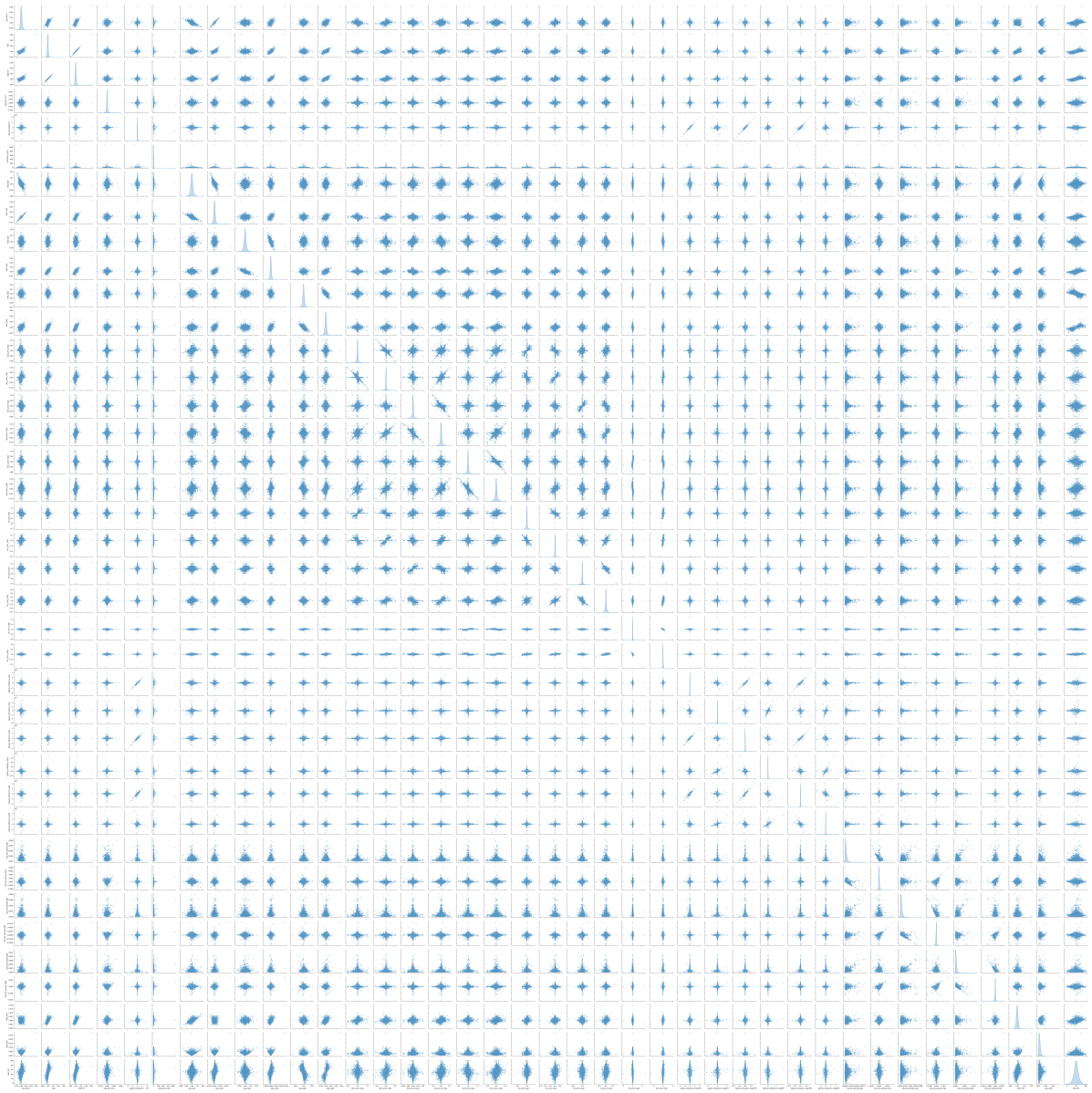

FIGURE 0.0.10.

6. Models

Now we start with models, but before that, we added lag values for log returns of 1 second, 2 seconds, and 3 seconds. Then we made an ordinary linear regression model to establish a baseline for predicting. Here is the output from that model.

OLS Regression Results

Dep. Variable:	target	R-squared:	0.000
Model:	OLS	Adj. R-squared:	0.000
Method:	Least Squares	F-statistic:	172.2
Date:	Wed, 03 Sep 2025	Prob (F-statistic):	0.00
Time:	17:42:18	Log-Likelihood:	-1.4004e+07
No. Observations:	3809371	AIC:	2.801e+07
Df Residuals:	3809361	BIC:	2.801e+07
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|
const	29.0223	4.576	6.342	0.000
log_return_lag1	4.0407	1.967	2.054	0.040
log_return_lag2	9.8490	2.766	3.561	0.000
log_return_lag3	3.6260	1.958	1.852	0.064
wap_ma3	-35.0621	5.817	-6.027	0.000
wap_std6	13.9170	3.975	3.501	0.000
wap_RSI	-0.0002	0.001	-0.381	0.703
wap	5.9516	3.144	1.893	0.058
signed_imbalance	9.812e-09	2.54e-10	38.574	0.000

```
imbalance_ratio      0.0009      0.000      6.761      0.000
==============================================================
Linear Regression MAE: 6.057147891124908
Baseline (predict 0) MAE (bps): 6.369328713330854
Relative improvement vs baseline: 0.04901314349698416
```

Looking at the R-squared and adjusted R-squared values, we see they are both zero. This means that the model explains 0% of the variance in the data, so this means that our model can't predict anything and is not good. This is expected, though a linear model is not what you would normally use for data like this. We will be measuring all of the models using mean absolute error. It measures, on average, how far your predicted values are from the actual values. Looking at the mean absolute error for our model, we see that it is 6.06. This means that our predictions are off by 0.06% from the 60-second actual future move. We also see that we do not have a lot of improvement from the Baseline mean error, which is just predicting the average for every point. Our model has no significant predictive power, though, since the mean absolute error is the average absolute difference of the predicted values minus the actual values, the mean absolute error does not really have a significant meaning; it mostly reflects the natural volatility of the target.

The next model we made was an ARIMA model, or an autoregressive integrated moving average model. Although this is a good model for time series analysis, it only works for one stock at one time period. So it can't be used for the whole data set. We made the model select a random stock and a random day, so each time you rerun it, you get a different result, and you can see how well the model predicts these individual stocks. For some of them, it predicts the stocks very well, and for others, it does poorly. One of the best ones was stock 75 on day 344, and it had an ARIMA MAE of 4.654374795152988. One of the worst ones was stock 82 on day 414 with an ARIMA MAE of 19.139869270010532. We also made a generalized autoregressive conditional heteroskedasticity (GARCH) model, which performed very poorly. This is also expected; GARCH models are meant to identify volatility clustering and aren't

really used for this time series analysis. Now that we have moved through these preliminary models, we are going to move on to machine learning and neural network-type models like XGBoost, CatBoost, GRU, LSTM, and PyTorch.

Moving on to XGBoost, are XGBoost the validation MAE was 6.00510 And the XGBoost MAE was 6.253518456007295 after tuning our parameters. We found that this worked the best.

```
xgb_model = xgb.XGBRegressor(
    n_estimators=3000,
    max_depth=3,
    learning_rate=0.001,
    early_stopping_rounds=10,
    subsample=0.7,
    colsample_bytree=0.7,
    min_child_weight=10,
    gamma=2,
    reg_alpha=1,
    reg_lambda=3,
    objective='reg:squarederror',
    eval_metric='mae',
    random_state=42,
    n_jobs=-1
)
xgb_model.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    verbose=100)
```

N estimators is the number of trees that there will be. For the size of our data, we did 3000. We found that this approach worked well when allowing early stopping rounds to make the decision. We went with a small learning rate, which will make the model slower to calculate, but for models with lots of noise, it will help avoid learning the noise and make smoother learning, and avoid chasing random noise. We also went with a small max depth. This is how deep the trees will go. For a model with this much noise, we went with a small depth, which will prevent memorizing noise. He also added a couple of regulators, like gamma Alpha and Lambda, to help prevent learning random noise. The MAE of 6 is pretty good. This means we are off by about 0.06% on average from the actual value of the 60-second future move. We also made a plot of the most important features that helped in predicting for the XGBoost model.

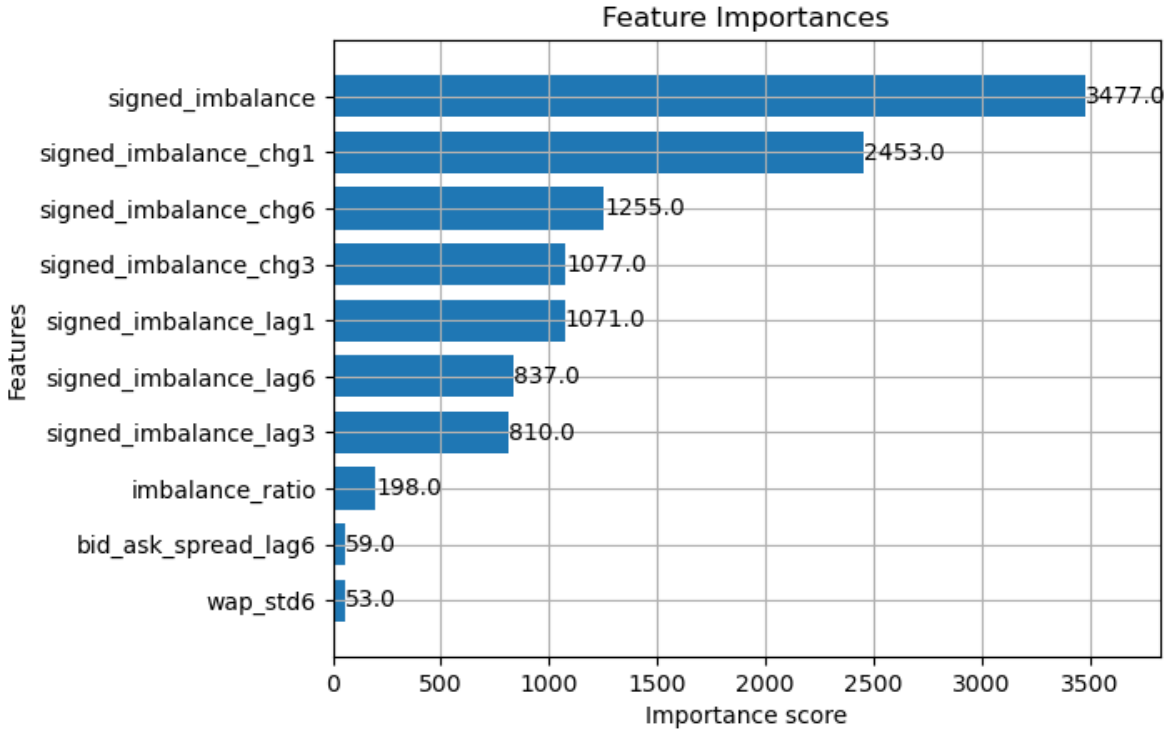

FIGURE 0.0.11.

The next model we made was to combine CatBoost and GRU/LSTM. For the cat boost model, we went with 3000 iterations, as well we also went with the depth of 6 for this model and a learning rate of 0.01, which seemed to work well after trying a bunch of other values for our parameters. For the GRU/LSTM, the parameters we used were epochs of 50, which is essentially the number of iterations of learning for the model. We also used a small batch size for this, so it will use more batches for learning the noisy data. We took a waited average of the mean absolute error for the cat boost model and the GRU/LSTM model. When we did that, we got a MAE of 6.0035049381864525, which is only a little better than the GXBoost model. This is still good, though, and we will move on to using a PyTorch model.

We made a PyTorch model. We thought that PyTorch might do better than XGBoost or CatBoost because PyTorch is a machine learning framework used for building and training deep neural network models, which we thought would be useful for data with this much noise, but it seems that CatBoost worked the best. It also looks like other competitors found CatBoost the best as well. Again, we also used a small batch size for the model to help avoid detecting random noise we also used a small learning rate to avoid noise. We used an Epoch of 50 with early stopping to give the model time to learn. This model did not do as good as the CatBoost model; it did about the same as the XGBoost model, and it had a validation MAE of 6.005850227366836. Although this isn't as good as the last model, this is still good.

We started with an ordinary linear regression model to get a Basic groundwork to start off from, then we went and made an ARIMA model and a GARCH model. The ARIMA model did good, but it can only predict individual stocks which some of which it predicted well and some predicted poorly. The GARCH model didn't do well, and it's not really meant for this type of time series analysis, but we did it anyway just to try out the model type and for our own learning experience. Then we made an XGBoost model, which did well, then we made a CatBoost model, which did the best out of all of our models, which was a little surprising because we thought that our PyTorch model would be the best for this kind of data, which did about the same as the XGBoost.

7. Conclusion

To conclude, for the Optiver trading at the close competition, we gathered our data from the provided data set on kaggle, imported it into Python started to clean it. We filled the missing values using the forward fill function, and for the few missing values that were left, we used backward fill. We did not do this for the Target column because using previous values to fill in other values is not appropriate for the variable we are predicting, so any rows that had missing values in them were removed. While doing this, remember you have to keep your index set because this is the time series data set, so anytime you remove rows, you need to reindex to make sure times still line up with their values. After our data was cleaned up, we started to make engineered features to help predict our target variable better. Then we went into exploratory data analysis, where we made a bunch of graphs to understand our data better and to see what our time series data looks like and how much noise and randomness there is in the model. We also checked the skewness and kurtosis of distributions to see if our data is skewed and it was slightly skewed to the right, and to see if there are a lot of outliers, and there were; we had very heavy tails, indicated by the kurtosis value. Quite a few of our graphs were for exploratory data analysis and to check if our data is stationary, and although we did have some concerns, it looks like our data was most likely stationary, so we continued on. We made our preliminary linear model just to check and give us a baseline for our predictions. The linear model did very poorly and had essentially 0 predictive power, which is expected because this data is not linear, which we checked with are pairwise scatter plot. Then we moved on to an XGBoost model, which did well after tuning all of our parameters to help reduce picking up on random noise. Then we made a CatBoost model combined with GRU/LSTM, which did the best out of all of our models after adjusting the parameters as well. We moved on to make a PyTorch model, which we thought would do the best, but it did not perform as well as we had thought, but still did well after tuning the parameters. Overall, this competition helped us gain insight into time series analysis, high-frequency trading, and making predictive models. It was very interesting making the models, figuring out what parameters worked the best, and giving us the best results. To get better results out of our model in the future, we

could add more engineered features, as we did not have a lot, but our models did pretty good for the amount we had. Adding more engineered features will allow the model to pick up on small details that the current engineered features didn't, and will help predict the target better.

8. Python Code

The complete Python source code for this project is available in this GitHub Repository. (If this link does not work, you may need to download it from GitHub.) You may need to download to see the full code, as GitHub will not display all of it or render the code block sometimes. Note that not all of the code was included in the paper, and some of the outputs were simplified for formatting.

9. Key Takeaways

1. Understand the Target

- The competition predicts the closing auction price.
- The model must capture short-term patterns and market microstructure signals near the close.

2. Feature Engineering is Crucial

- Order book features: Best bid/ask, mid-price, bid-ask spread.
- Time-sensitive features: Rolling averages, changes in depth, and recent trades leading up to close.
- Cross-feature interactions: Ratios like bid/ask volume, price changes $\times$ volume.

3. Model Choice Matters

- Tree-based models (XGBoost, CatBoost, LightGBM): Great for tabular data, fast training, robust to overfitting.
- Sequential models (GRU, LSTM in PyTorch): Best if used in combination with other models.

- Hybrid approaches: Engineered features with time lags, then gradient boosting, may outperform pure deep learning on high-frequency tabular data.

4. Evaluation Strategy

- Use the competition metric, mean absolute error, to evaluate predictions.
- Avoid overfitting in your models
- Cross-validation should respect temporal order (do not shuffle time-series data).

5. Data Challenges

- High-frequency data is massive: Efficient processing and memory management are critical.
- Noise and microstructure effects: Small price movements can be unpredictable; robust features help.

References

[1] Cowles, A. (n.d.). *Efficient-market hypothesis*. Wikipedia, the free encyclopedia. Retrieved September 6, 2025, from https://en.wikipedia.org/wiki/Efficient-market_hypothesis

[2] Optiver. (n.d.). *Trading at the Close*. Kaggle. Retrieved September 6, 2025, from https://www.kaggle.com/competitions/optiver-trading-at-the-close/overview

[3] Random walk hypothesis. (n.d.). *Random walk hypothesis*. Wikipedia, the free encyclopedia. Retrieved September 6, 2025, from https://en.wikipedia.org/wiki/Random_walk_hypothesis

[4] Optiver. (n.d.). *Trading at the Close: Data*. Kaggle. Retrieved September 6, 2025, from https://www.kaggle.com/competitions/optiver-trading-at-the-close/data

[5] Optiver. (n.d.). *Trading at the Close: Code*. Kaggle. Retrieved September 6, 2025, from https://www.kaggle.com/competitions/optiver-trading-at-the-close/code

[6] Optiver. (n.d.). *Trading at the Close: Discussion*. Kaggle. Retrieved September 6, 2025, from https://www.kaggle.com/competitions/optiver-trading-at-the-close/discussion?sort=hotness

[7] Optiver. (n.d.). *Trading at the Close: Leaderboard*. Kaggle. Retrieved September 6, 2025, from https://www.kaggle.com/competitions/optiver-trading-at-the-close/leaderboard

[8] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Penn Arts & Sciences. Retrieved August 19, 2025, from https://www.sas.upenn.edu/~fdiebold/NoHesitations/BookAdvanced.pdf

[9] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. UC Berkeley Statistics department. Retrieved August 19, 2025, from https://www.stat.berkeley.edu/~rabbee/s154/ISLR_First_Printing.pdf