Nishk Patel Tuesday, February 20, 2023
Research Team Application

Quant Options Analysis

Overview:

The scope of the project included log parsing and data analysis. With the goal of analyzing the file generated from a simulated equity options strategy to increase the fill rate and reduce latency, I focused on those two conditions in connection to other pertinent values. Ultimately I was able to find a way for the Order Manager to reduce latency on outgoing trades and

Implementation:

The research I completed was implemented in both R and Python. Firstly in R, I made visual representations of the relationship between many of the variables and found evidence for specific correlations between columns of the imported data frame.

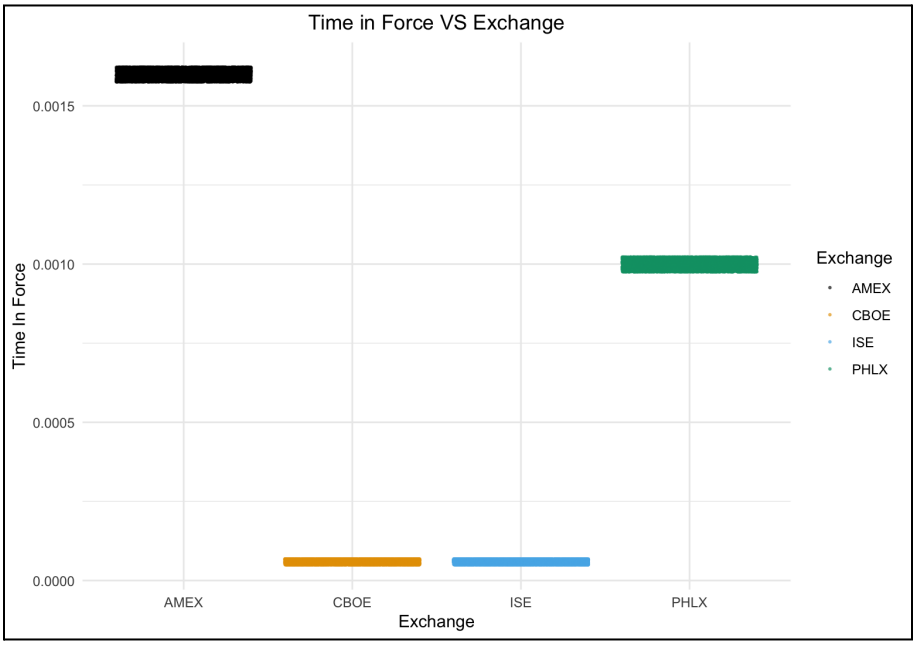
R: In R I performed exploratory data analysis (EDA) to identify patterns and insights. The majority of my analysis was done by utilizing the tidyverse library to plot the relationships between variables. After importing the dataset, I cleaned the data and reorganized specific variables.

- ➢ Date and Time variables were converted to numeric values
  - TotalTime variable created for Time in Force representation
- ➢ AdjOptionData had removed OrderEventType rejected values
  - This was vital to visualizing data without the orders rejected by the order manager from interfering with the analysis

➢ OrderCancel/OrderFill is a data frame of all canceled orders and filled orders

Analysis 1: Time In Force VS Exchange

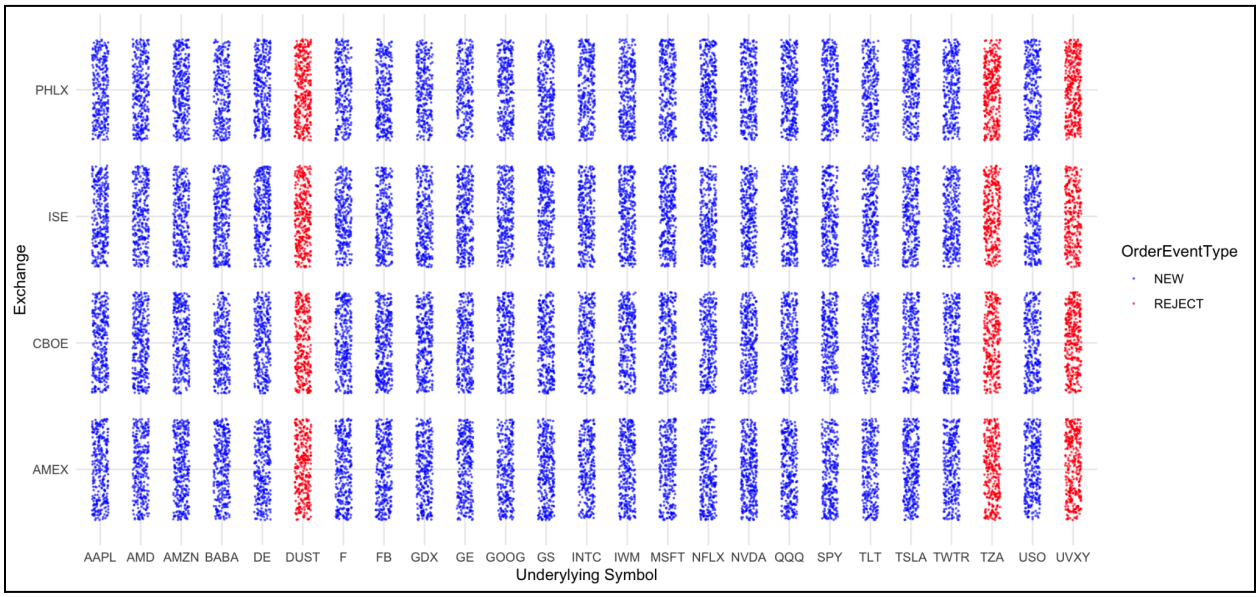
One major aspect of reducing latency was looking at the exchange that the option was being traded. After taking a look at the different exchanges and the latency between the order being placed and its execution or cancellation, it became abundantly clear that specific exchanges seemed to take longer to trade on. This means that the moment from when the Order Manager sends out the order to the moment it receives confirmation about what occurred, there is a difference in the delay that is dependent on the exchange. If the Strategy Engine were to only provide orders that could be placed on the CBOE and ISE exchanges rather than including the AMEX and PHLX exchanges, it would reduce the time in force latency. This is a significant factor to take into consideration given the fact of information that making faster trades with more efficient systems means that on the extremely marginal scales, the trades will be more precise and therefore, occur at a better rate.



Analysis 2: Rejected VS Underlying Symbol

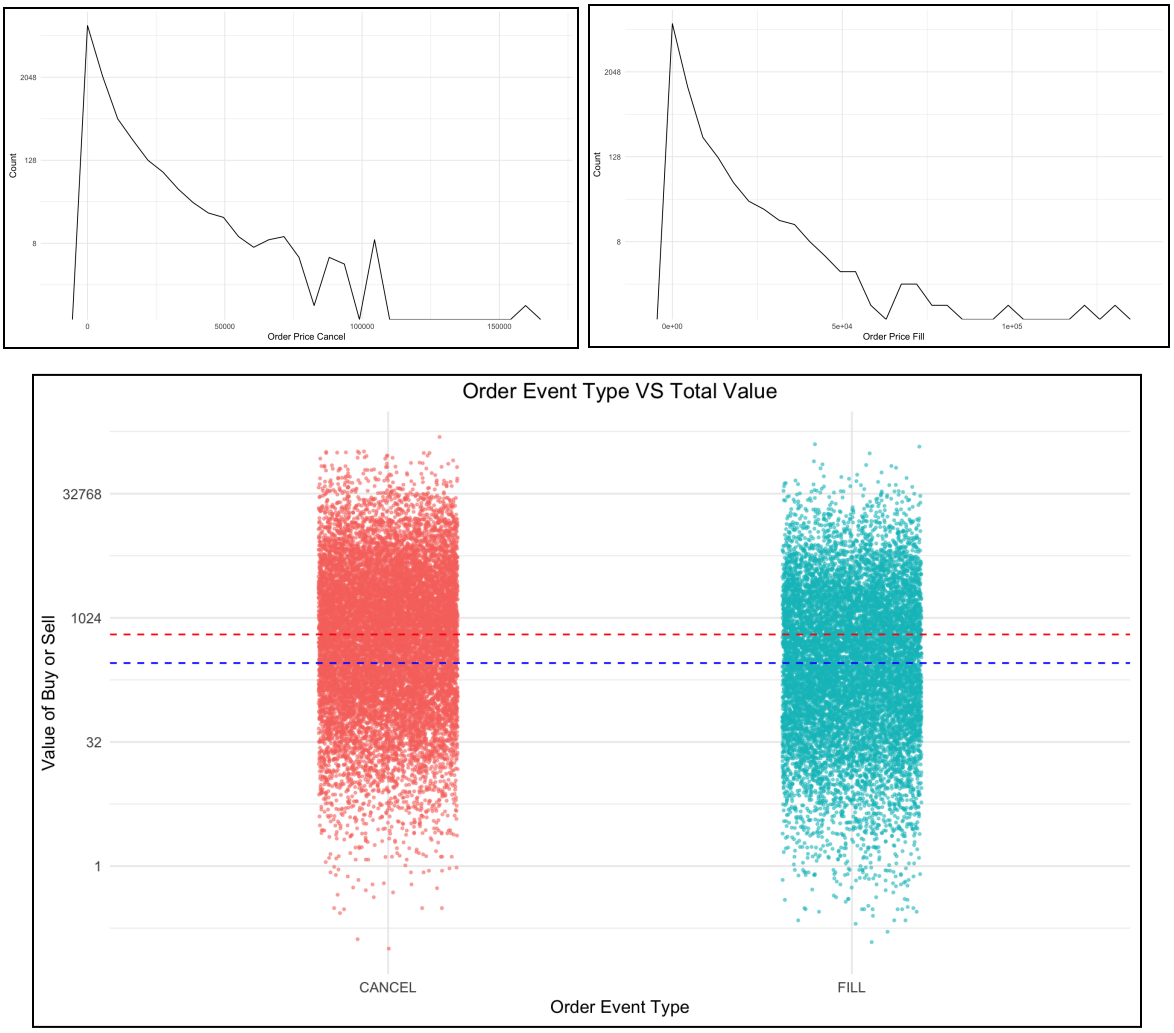
The Order Manager chose to reject certain orders and instead of being given a ‘New’ tag, they were ‘Rejected’ and the order was never sent to market. In the case of specific

stocks, the order manager immediately chose to reject them as demonstrated by the visualization below. With this information, it is possible to reduce the amount of time the Order Manager takes on the orders by having the Strategy Engine never suggest options from the options with the tickers 'DUST', 'TZA', and 'UVXY' regardless of the exchange they are being traded on.



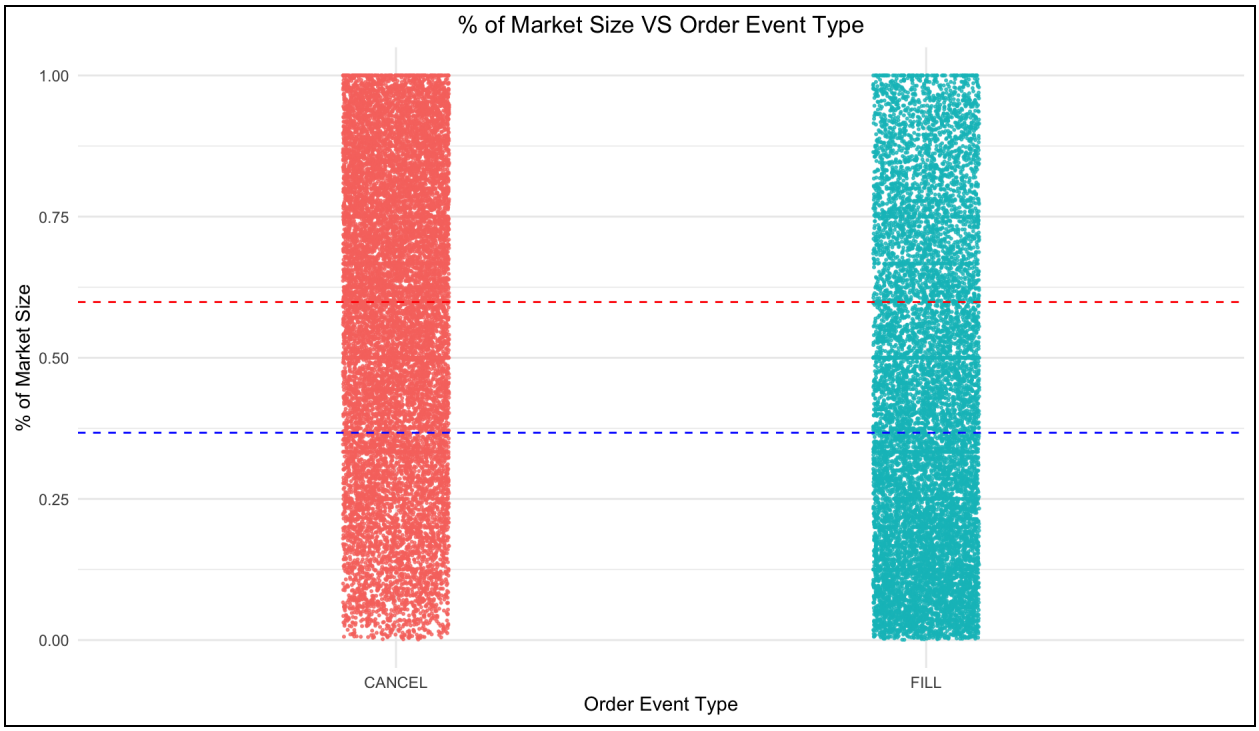
Analysis 3: Price VS Order Type and Analysis

The first relationship I noticed was between the order event type and the total value of the order. All orders that were canceled were found were graphed on the right in red and the orders that were filled were graphed on the right in blue. Each dashed line represents the average price of each option traded within the respective category. Whereas the orders that were canceled had a median of 2797.863 orders that were filled had a median of 1519.511. The median was chosen as the distribution of filled and canceled orders was heavily skewed due to the non-normal distribution as shown below. Furthermore, I ran a t-test on the difference of means of the two order event types and the total price of the order and got a p-value of $< 2.2e\text{-}16$. This highlights that the true difference in means of the two is not equal to 0, therefore, showing that there is a correlation between the value of the orders and whether or not they were canceled and filled.



Analysis 4: Portion of Market Size VS Order Event Type

Another correlating factor between whether orders were canceled or filled was the percentage of the market that the specific order held. Again, the canceled and filled orders were plotted in red and blue respectively. However, the response variable, in this case, is illustrated to be the portion of the market size that is derived from the order size over the market size. In this case, the density of each can be seen against the plotted points as more of the orders that had been canceled were closer to the market size whereas the ones that were filled were a smaller portion of the market. The means of these two were significantly different; orders deemed canceled were 59.87% of the market on average while orders filled were only 36.72%. Running a statistical t-test on this data also gave a p-value of <2.2e-16 further supporting the evidence that an order with a smaller market size would be more likely to have a higher fill rate.



Metrics

The following represent specific before and after the suggested changes have been made:

Average Time in Force Before: 0.0006831878 sec

Average Time in Force After: 0.00005903926 sec

Total Daily Latency Before: 18.03411 sec

Total Latency After: 0.775894 sec

Percentage Rejected After: 12.01%

Percentage Rejected After: 0%

Python: In python, I used the pandas and TensorFlow libraries to create a basic neural network model to predict whether an order was going to be canceled or filled. This process allowed me to analyze the data in a machine-learning model with specific parameters in an attempt to get accurate predictions.

Neural Network:

The first step in creating the model was to import and clean my data from the optionsData.csv. I did this primarily using the pandas library. In order to obtain trainable data, I kept what I believed to be the most relevant variables that would affect the output

the largest. I also converted certain non-numeric and categorical variables into numeric inputs and created the training and validation data.

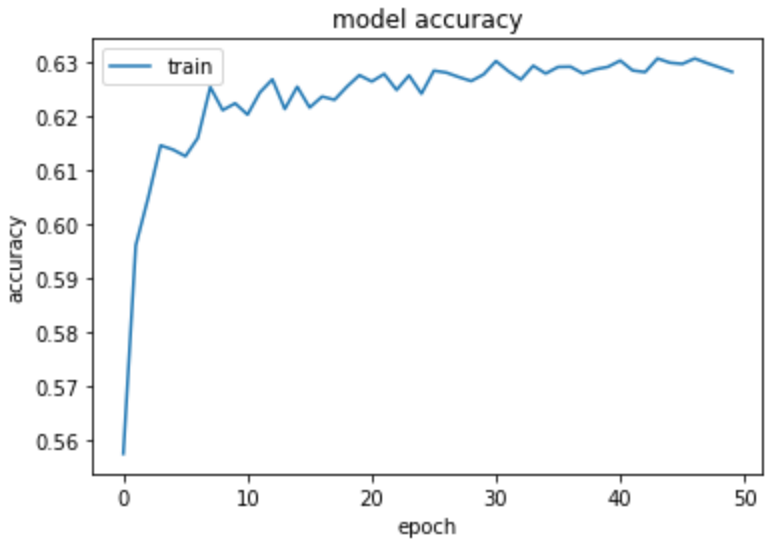
The model was a binary classification system in which the model predicted whether an option order was going to be filled or canceled. I implemented a sequential model with 1 input layer, 4 hidden layers, and 1 output layer.

The following was a summary of the model:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 5)                 0

 dense (Dense)               (None, 64)                384

 dense_1 (Dense)             (None, 248)               16120

 dense_2 (Dense)             (None, 512)               127488

 dense_3 (Dense)             (None, 128)               65664

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 217,977
Trainable params: 217,977
Non-trainable params: 0
_________________________________________________________________
```

While training the model, I began with a fewer number of hidden layers and fewer nodes, however, I realized that an initial increase in layers and nodes helped improve the accuracy of the data without it causing overfitting. After compiling and running the model, I received an accuracy of around 65% on the validation data for some models while an average had an accuracy of around 63% as visualized below. Although this accuracy was not as high as I had hoped, it indicated that some relationship existed between my 4 input variables and the fill rate as the increased accuracy occurred over the higher cancellation rate over the fill rate.

Accuracy per epoch:



Conclusion:

Ultimately, I was able to find many relationships between specifically correlating variables in the given options log data set. Given the visual analysis in R, I was able to find a relationship between the Exchange and the time in force, allowing me to suggest a way to decrease latency among the possible orders sent out by the Strategy Engine. Also, to decrease the number of 'Rejected' orders given by the Strategy Engine, I highlighted the relation between specific stocks and the rejection rate by the Order Manager. To optimize the fill rate, I found correlating variables such as the percent of the market and total value of the order which can be utilized to increase the fill rate by the Strategy Engine. In the neural network model in python, although the accuracy was only around 63% it was still able to show a path to more accurately predicting whether the order was going to be filled or canceled.

Future Plans:

With more time and hardware resources, I may be able to perfect the python neural network model with a better classification system and more optimized trainable parameters. Things I would do initially would be to try adjusting the learning rate, hidden layers, and tensors to different values to try and optimize the accuracy. The most significant and likely most beneficial change I would be able to make would be to use the strike price in junction with the API data on specific options underlying symbol to use the intrinsic value to better find relationships between the variables in the given data and those found outside of the limited data.