How to use:

1. Use readwrite.py to create a ordered book. Input the file name of the raw data. Restart python after each use because I bind some attributions to the class instead of instance. This can be changed easily. However, I did not bother to fix it since the ordered book is created as I wanted. Besides, I think read one line and throw the previous line to the garbage collector may be a good idea. (It is fun to me at least.)
2. Use analysis class in anlysis.py to read ordered book.

analysis class has mothods:

vwap for calculating the volume weighted average prices at a given time or within a given time window, where the time window is a pair of integers of the form (start time, end time). If vwap takes a time, then it returns a float. if vwap takes a time windows, then it returns a dict, of which keys are time in the time window, values are the corresponding vwap.

twap for calculating the time weighted average prices at a given time or within a given time window. If twap takes a time, then it returns a float. if twap takes a time windows, then it returns a dict, of which keys are time in the time window, values are the corresponding vwap.

arbi for calculating the arbitrage index, which is the number of chances of arbitrage divided by the number of orders within a given time window. arbi takes a time window and return a float, and a dict, of which keys are order id, and values are the corresponding list of order ids that form a chance of arbitrage.

sampling for randomly choose a time window with minimum number of orders. sampling takes an integer for time span of the time window, and an integer for minimum number of orders, returns a time window.

side for splitting orders within a given time window by the side of orders. side takes a time window and returns two dicts, one for each side.

1. In result.py, use data class to time wisely random sample to data.
2. In test.py, I random sampled the data with data class and use the sampled data to train a Lasso model by sklearn. The sampled data is saved as json files.

Other py files:

1. In book.py, the book class will read each line in the file and update the top and bottom 5 prices, respective.
2. In ntime.py, nt function takes a timestamp and returns a nearest timestamp of a order. ntw function takes a time window and return a list of timestamps of orders within the time window. ntw\_order\_count function takes a time window, a dict from timestamp to order id, and a sorted list of timestamp that is the key of the dict.
3. in distri.py, distri function takes either a dict, of which values are list of numbers, or a list. If it takes a list, then it returns an inference of the list. If it takes a dict, then it returns an inference for each key. std function takes a dict, of which values are list of numbers, or a list. If it takes a list, then it returns an standard deviation of the list. If it takes a dict, then it returns an standard deviation for each key. momentum function takes either a dict, of which keys are timestamps, or a list. If it takes a list, then it returns the momentum of the list, which is the logarithm of last entry divided by the first entry. If it takes a dict, then it returns the momentum of the dict.

Predictive features and results:

I looked time span of orders, which the median is more than 3 secondes (3000000). So I decided to sample the the data by 3 seconds. The reason is for sampling is that otherwise, the size of data is too much to analysis. Also I am looking for the chance of arbitrage. I plan to do it in 3 steps:

1. build the model of the arbitrage index, see below, of a time window. Ideally use other features to build a decision tree model.
2. build a time series model of other features so that I could use features in a given time window to predict the arbitrage index in the next time window.
3. for the time window with high arbitrage index, which there are, identify the features of the orders that contributes heavily to the index.

I used following predictive features:

1. arbitrage index. The number of pairs of possible arbitrage divided by the total orders within 3 seconds. This index could be used to predict when a chance of arbitrage appears. To predict this index, I used more predictive features.
2. market index. The number of orders within 3 seconds.
3. Lowest/median/highest twap and twap momentum within a 3 seconds time window.
4. Lowest/median/highest vwap and vwap momentum within a 3 seconds time window.

Unfortunately, I do not have time to complete my 3 steps. I will only present my result of the Lasso model. The dependent variable of the Lasso model is the arbitrage index, and the independent variables are the other features I mentioned above.

The result is

A picture containing text

Description automatically generated

Then I tried to normalized the X and again the model does not look well. The reason of this poor performance of the model is due to:

1. the predictive features X are not suitable to predict the arbitrage index. If I had more time, I would find more features.
2. the arbitrage index of this sample set is sparse. If I had more time, I would form two groups of the same size, one with samples of high arbitrage indices, the other with samples of low arbitrage indices. Then use principal component analysis or clustering methods to find the useful features.