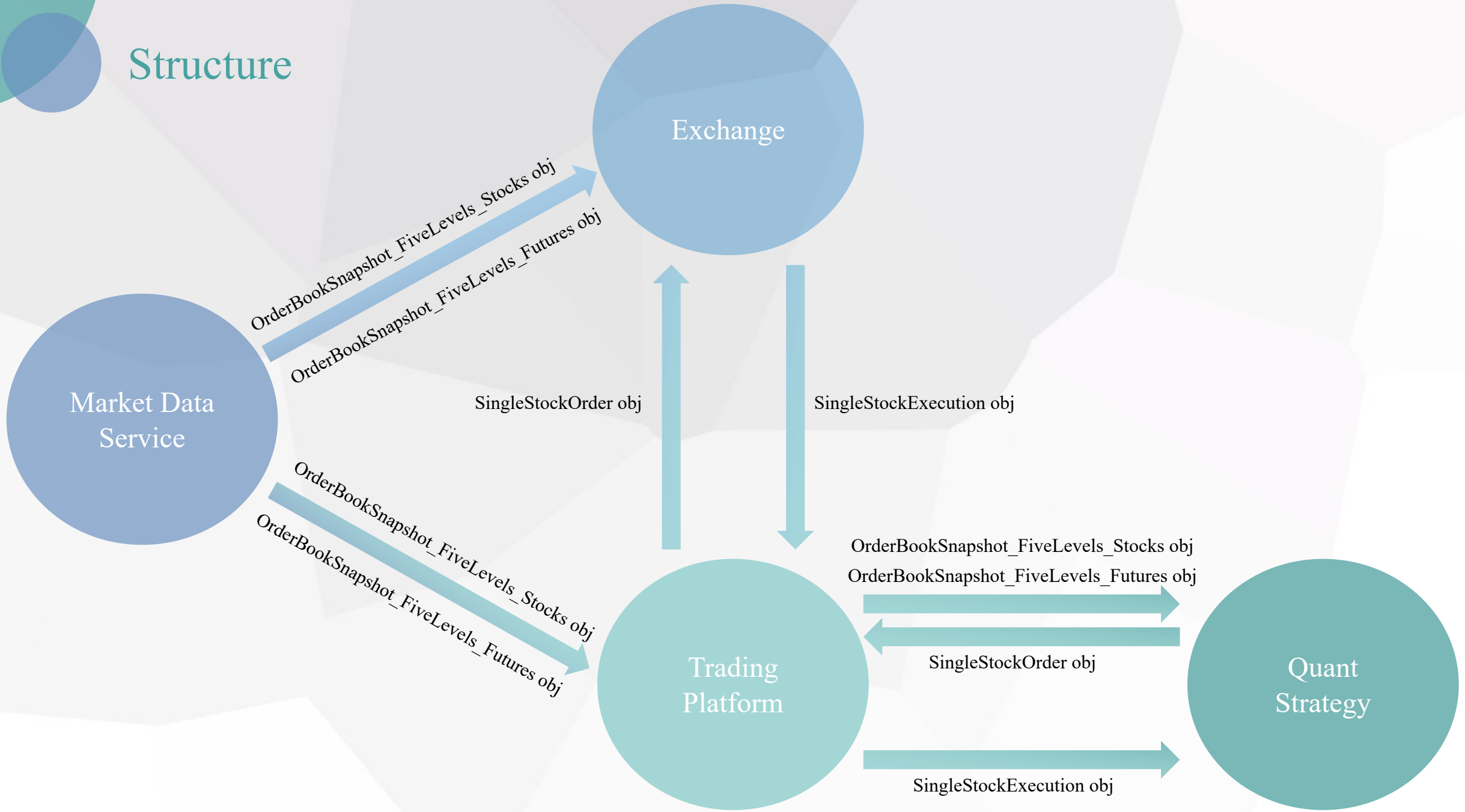
2023
Group Project

CONTENT

Structure


Exchange
Market Data Service
Trading Platform
Quant Strategy
OrderBookSnapshot_FiveLevels_Stocks obj
OrderBookSnapshot_FiveLevels_Futures obj
SingleStockOrder obj
SingleStockExecution obj
OrderBookSnapshot_FiveLevels_Stocks obj
OrderBookSnapshot_FiveLevels_Futures obj
OrderBookSnapshot_FiveLevels_Stocks obj
OrderBookSnapshot_FiveLevels_Futures obj
SingleStockOrder obj
SingleStockExecution obj

PART 1

Trade Data Processing and Re-publishing

LI Yangming
HE Xinyi
WANG Yishang
ZHENG Yadan

Trade Data Processing and Re-publishing

Stocks

1. Publish stock market data at the same speed as history
   Start time: now
   Time intervals: same as data used

```
self.next_time = data.at[0, 'time']
self.timegap = (self.next_time - self.this_time)/1000

for i in range(10): # data.shape[0]
    time.sleep(self.timegap)
    self.produce_quote(marketData_2_exchSim_q, marketData_2_platform_q, i, stock, data)
```

2. Simulate real market to show ten stocks together

3. Delay 15 min to print stocks to match with time of futures

Trade Data Processing and Re-publishing

Futures

1. Publish future market data at the same speed as history

```
self.timeStamps = np.unique(np.array(df.timeStamp))
self.timeIntervals = np.diff(self.timeStamps)
self.allTickerData = [group.values[:, 1:] for _, group in df.groupby('ticker')]

self.produce_quote(marketData_2_exchSim_q, marketData_2_platform_q)
```

```
time.sleep(self.timeIntervals[i]/1000000000)
```

*The timestamp is nanosecond: 1 ns=10^{-9}s

2. Simulate real market to show futures trading data

Trade Data Processing and Re-publishing

Stock Data Sample

```
[79301] <<<<< call MarketDataService.init
[79296] <<<<< call MarketDataService.init
[79304] <<<<< call MarketDataService.init
[79298] <<<<< call MarketDataService.init
[79302] <<<<< call MarketDataService.init
[79299] <<<<< call MarketDataService.init
[79297] <<<<< call MarketDataService.init
[79303] <<<<< call MarketDataService.init
[79300] <<<<< call MarketDataService.init
[79305] <<<<< call MarketDataService.init
[79305]MarketDataService >>> produce_quote
  ticker        date      time  volume  size  lastPx  askPrice5  askPrice4  ...  askSize3  askSize2  askSize1  bidSize1  bidSize2  bidSize3  bidSize4  bidSize5
0   5425  2023-06-30  90001584      45  45.0  9750.0     9790.0     9780.0  ...         7         6        12         2         5         9        11         4

[1 rows x 26 columns]
[79302]MarketDataService >>> produce_quote
  ticker        date      time  volume   size   lastPx  askPrice5  askPrice4  ...  askSize3  askSize2  askSize1  bidSize1  bidSize2  bidSize3  bidSize4  bidSiz
e5
0   3035  2023-06-30  90001682     338  338.0  21100.0    21350.0    21300.0  ...        62        82        44        37        81        54        22
64

[1 rows x 26 columns]
[79302]MarketDataService >>> produce_quote
  ticker        date      time  volume  size   lastPx  askPrice5  askPrice4  ...  askSize3  askSize2  askSize1  bidSize1  bidSize2  bidSize3  bidSize4  bidSize
5
0   3035  2023-06-30  90001719     346   8.0  21100.0    21350.0    21300.0  ...        62        82        44        29        81        54        22        6
4

[1 rows x 26 columns]
[79302]MarketDataService >>> produce_quote
  ticker        date      time  volume  size   lastPx  askPrice5  askPrice4  ...  askSize3  askSize2  askSize1  bidSize1  bidSize2  bidSize3  bidSize4  bidSize
5
0   3035  2023-06-30  90001719     346   0.0  21100.0    21350.0    21300.0  ...        62        82        44        29        81        54        22        6
4

[1 rows x 26 columns]
[79302]MarketDataService >>> produce_quote
  ticker        date      time  volume  size   lastPx  askPrice5  askPrice4  ...  askSize3  askSize2  askSize1  bidSize1  bidSize2  bidSize3  bidSize4  bidSize
5
0   3035  2023-06-30  90001729     346   0.0  21100.0    21350.0    21300.0  ...        62        82        44        29        81        54        22        6
6

[1 rows x 26 columns]
[79302]MarketDataService >>> produce_quote
  ticker        date      time  volume  size   lastPx  askPrice5  askPrice4  ...  askSize3  askSize2  askSize1  bidSize1  bidSize2  bidSize3  bidSize4  bidSize
5
0   3035  2023-06-30  90001745     375  29.0  21100.0    21350.0    21300.0  ...        62        82        44        81        54        22        66        7
4
```

PART 2

Exchange Simulator

LU Jiaqian
BIAN Yalan
FANG Hao
ZHENG Yuchang
GUO Chupeng

Exchange Simulator

Save Data

Receive data from the first part and store it in the dictionary, update the data according to the stock name

```
md = marketData_2_exchSim_q.get()
self.current_orderbook_dict[md.ticker] = md

print('[%d]ExchSim.consume_md' % (os.getpid()))
print(md.outputAsDataFrame())
```

Save Order

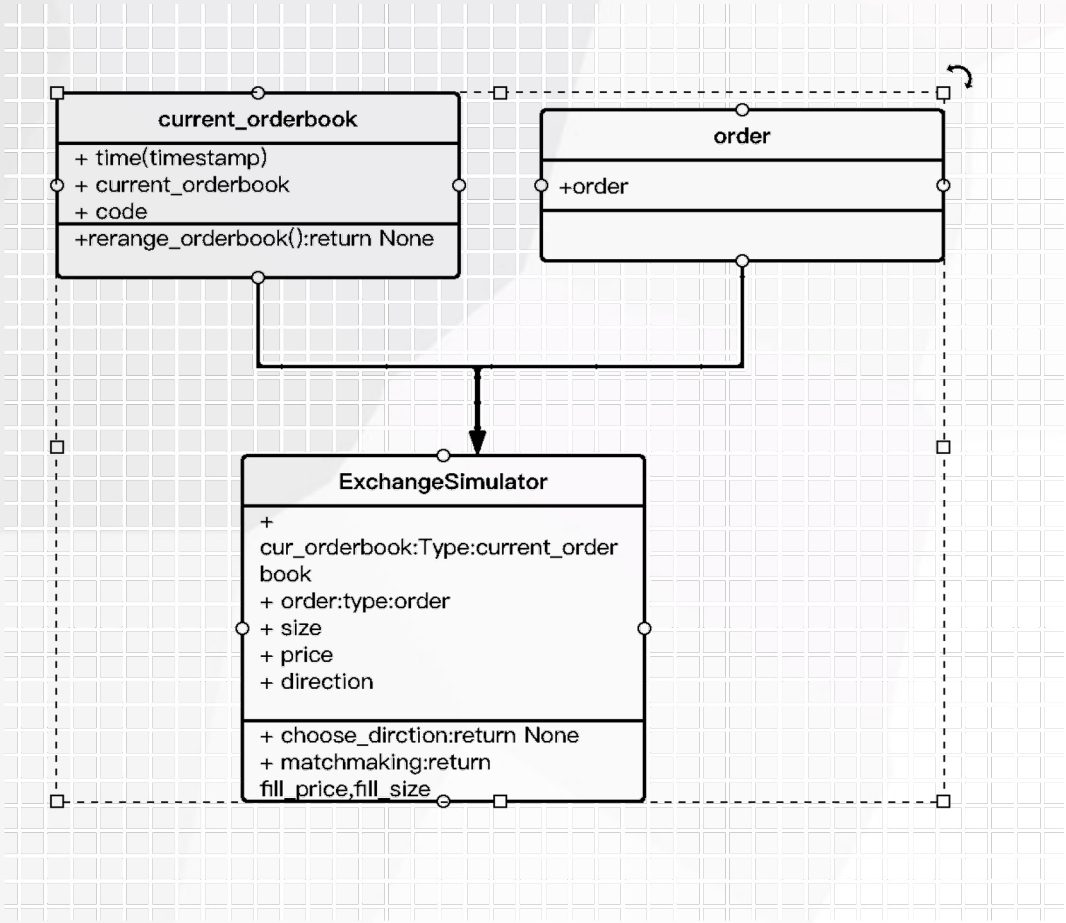
Take stock orders from the part three

Exchange Simulator

Transaction Judgment

1.Match the order sent by the third part with the saved stock data according to the stock code, and judge whether to trade

2.According to the direction of the order, and the corresponding price and transaction volume, judge the successful transaction volume and transaction price



Exchange Simulator

Send Information

Send the calculated transaction price and transaction volume to the third group.
If there is no transaction, a null value will be sent

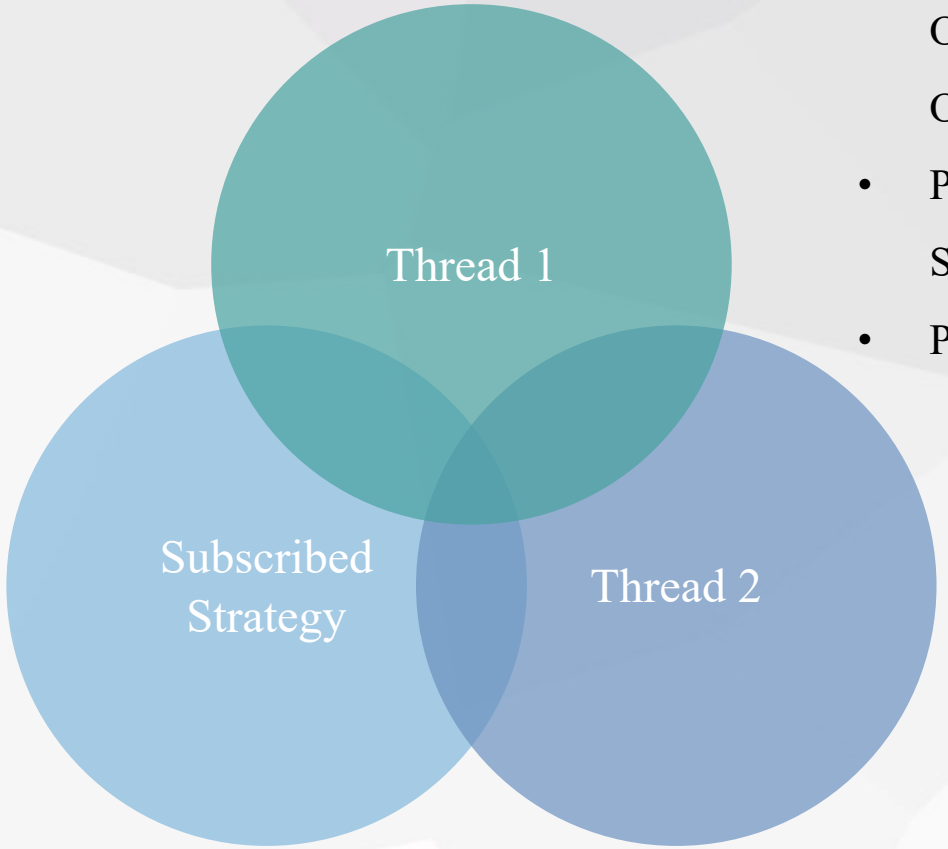
```
cur_orderbook is
   askPrice  bidPrice  askSize  bidSize                  timestamp  code
0      8005      8000       17      316 2023-07-23 13:33:42.215426  0050
1      8010      7995       30      490 2023-07-23 13:33:42.215426  0050
2      8015      7990       27      477 2023-07-23 13:33:42.215426  0050
3      8020      7985       41      236 2023-07-23 13:33:42.215426  0050
4      8025      7980      250      241 2023-07-23 13:33:42.215426  0050
order_submit_now is
    direction  price  size
01          1   8010    34
fill_price:8007.5
fill_size:34
```

PART 3

Trading Platform

YAO Shuochen
TANG Sijia
NI Jianle
KANG Zhenqing

Structure



- Receive market data from Market Data Service as OrderBookSnapshot_FiveLevels_Stocks objects and OrderBookSnapshot_FiveLevels_Futures objects
- Pass orderbook snapshots to Quant Strategy and receive order back as SingleStockOrder object
- Pass order to Exchange

- Receive execution from Exchange as SingleStockExecution object
- Pass execution to Quant Strategy to calculate PnL

Details

Pass orderbook snapshot to Quant Strategy

- Dictionary of tickers as keys and OrderBookSnapshot_FiveLevels_Stocks objects and OrderBookSnapshot_FiveLevels_Futures objects as values
- Ignore empty queue of orderbook snapshots in specific ticker to avoid blocking queues

```
data_dic = {}
# fTicker_lis =  ['EHF1', 'QWF1']
# sTicker_lis =  ['1319','2388']
fTicker_lis = ['JBF1', 'QWF1', 'HCF1', 'DBF1', 'EHF1', 'IPF1', 'IIF1', 'QXF1', 'PEF1', 'NAF1']
sTicker_lis =  ['3443','2388', '2498', '2610', '1319', '3035', '3006', '2615', '5425', '3105']
for i in range(len(fTicker_lis)):
    string = '''
    if marketData_2_platform_q_{}.empty() == False:
        data_dic[sTicker_lis[{}]] = marketData_2_platform_q_{}.get()
    else:
        pass
    if marketData_2_platform_q_{}.empty() == False:
        data_dic[fTicker_lis[{}]] = marketData_2_platform_q_{}.get()
    else:
        pass
    '''.format(sTicker_lis[i], i, sTicker_lis[i], fTicker_lis[i], i, fTicker_lis[i])
    exec(string)
```

Details

Receive orders back from strategy and pass them to Exchange separately for each stock and futures

```
if len(data_dic) > 0:
    new_order = self.quantStrat.run(data_dic, None)
else:
    new_order = None
if new_order is None:
    pass
else:
    # do something with the new order
    for key in new_order.keys():
        platform_2_exchSim_order_q.put(new_order[key])
```

PART 4

Single Stock and Single Stock Future HFT Strategies

XIAO Li
ZHANG Guangyu
ZHENG Zixin
WONG Chun Ho

How to build and back-testing a forecasting model

Steps to build our forecasting model

01
Part one
Model Assumption

02
Part one
Model Assumption

03
Part three
Model Selection

04
Part four
Model building

Steps to build our forecasting model

01

Part one

Model Assumption

- History can predict future
- More recent data play more important part in predicting the future.
- There are leading lagging effects between selected pairs of single futures and single stocks.
- In this project, we specifically look into how futures′ lagging return leads the stocks′ forward return.

Steps to build our forecasting model

02

Part two

Data Preparation

- Sample dataset choosing:
  - Loading stocks and futures data using most recent 6 days
    - [″2023-06-29″, ″2023-06-28″, ″2023-06-27″, ″2023-06-26″, ″2023-06-21″, ″2023-06-20″]

Steps to build our forecasting model

02

Part two

Data Preparation

- Data cleaning:
  - Handling NaN INF …
  - Synchronize time series of stock and future pairs (1)

```
#Handle NA
stockData['lastPx'] = stockData.groupby('date')['lastPx'].fillna(method='ffill')
stockData['size'] = stockData.groupby('date')['size'].fillna(0)
futuresData['midQ'] = (futuresData['bidPrice1'] + futuresData['askPrice1']) / 2
```

```
# SYNCHRONIZE the two time series
# First, we want to get the common trading days from the two time series
##We first get the trading days from the stock data
stockData_dates = np.unique(stockData.date)
stoD = pd.to_datetime(stockData_dates, format="%Y-%m-%d")
qqqq = stoD.year * 10000 + stoD.month * 100 + stoD.day

##Next, we get the trading days from the futures data
indexData_dates = np.unique(futuresData.date)
indD = pd.to_datetime(indexData_dates, format="%Y-%m-%d")
pppp = indD.year * 10000 + indD.month * 100 + indD.day

##Now, we can get the common days of the two time series
commonDays = pd.to_datetime(pppp.intersection(qqqq), format="%Y%m%d")
```

```
##In order to synchronize trade time, too, we create indexes for both dataframes that have the same format

stockData_DateTime = pd.to_datetime(stockData.date.astype(str) + ' ' + stockData.time.astype(str),
                                    format="%Y-%m-%d %H%M%S%f")
futuresData_DateTime = pd.to_datetime(futuresData.date.astype(str) + ' ' + futuresData.time.astype(str),
                                      format="%Y-%m-%d %H%M%S%f")
```

```
##Now, we create indexes of the same format for both the stock data and the index futures data
stockData.index = stockData_DateTime
stockData = stockData[~stockData.index.duplicated(keep='last')]

futuresData.index = futuresData_DateTime
futuresData = futuresData[~futuresData.index.duplicated(keep='last')]
```

Steps to build our forecasting model

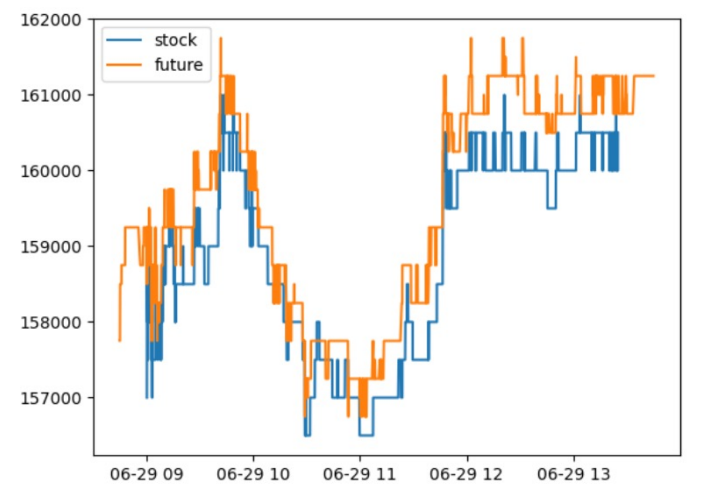
02

Part two

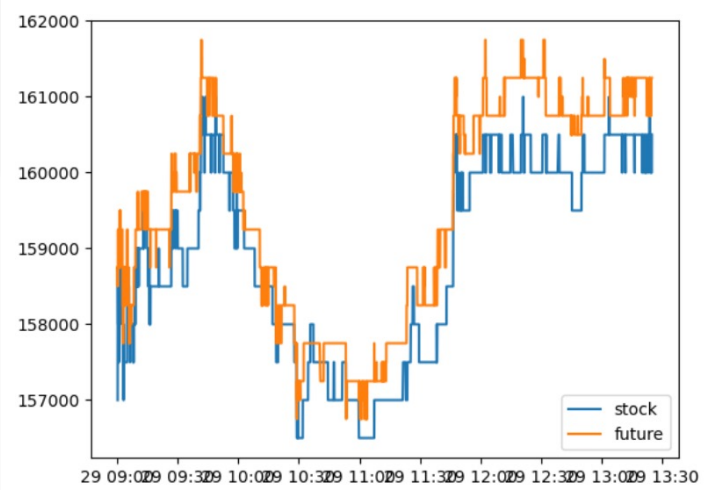
Data Preparation

- Data cleaning:
  - Synchronize time series of stock and future pairs (2)
  - Down-sample the time series with more time stamps

Before down-sample – 29/6 intraday



After down-sample – 29/6 intraday



Steps to build our forecasting model

02

Part two

Data Preparation

- Sample dataset choosing:
- Feature and Label Preparation:
  - Features: calculate x ticks lagging return on futures
    - Formula: return = (MidQ[t]-MidQ[t-x])/MidQ[t-x]
    - [fLaggingRtn_10′, 'fLaggingRtn_20′, …, 'fLaggingRtn_90′, 'fLaggingRtn_100′]
  - Labels: calculate x ticks forward return on stocks
    - Formula: return = (MidQ[t+x]-MidQ[t])/MidQ[t]
    - [sForwardRtn_10', 'sForwardRtn_20’, …, 'sForwardRtn_90', 'sForwardRtn_100']

Steps to build our forecasting model

02

Part two

Data Preparation

- Data splitting:
  - Training data: one day's whole tick data
    - [“2023-06-28”, “2023-06-27”, “2023-06-26”, “2023-06-21”, “2023-06-20”]
  - Testing data: the next day's whole day tick data
    - [“2023-06-29”, “2023-06-28”, “2023-06-27”, “2023-06-26”, “2023-06-21”]

Steps to build our forecasting model

03

Part three

Model Selection

- Linear regression
  - Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent or predictor variables.
- Gradient Boosted Regression Trees (GBRT)
  - GBRT is a machine learning algorithm that combines multiple decision trees to make predictions.
  - GBRT is particularly effective for regression problems where the goal is to predict a continuous numerical value, such as predicting housing prices or stock prices.

Steps to build our forecasting model

04

Part four

Model building

- Training process:
  - Applying GBRT on features above to predict different labels
  - Feature selection: select 5 most important features
  - Model evaluation: In sample R^2
- Testing process:
  - Evaluation: calculate the mean out of sample R^2 for each label in each day, and select the label with the highest mean R2 as the label for the best model.
  - Backtesting: to be finished

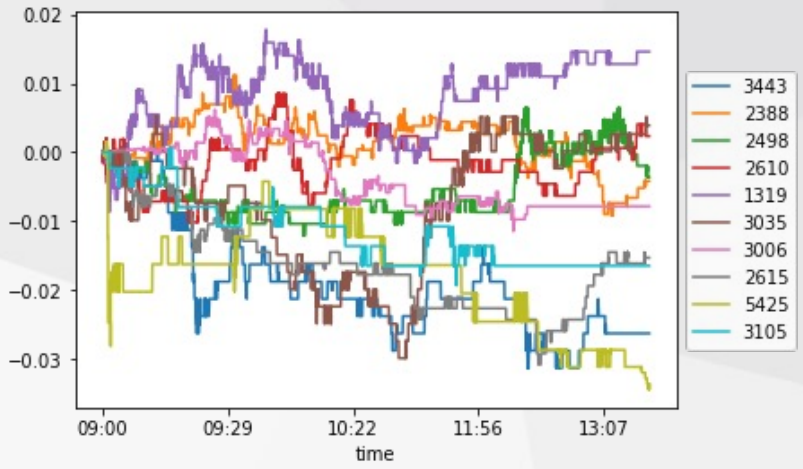
Back testing

- Back testing for the last 5 days (assume 06-21, 06-26, 06-27, 06-28 and 06-29 as the last trading day respectively and do the processes in previous part with corresponding days pushed back.)
- Consider single stock as a portfolio and see the performance to determine which stocks we will choose in the final simulated trading with our model.

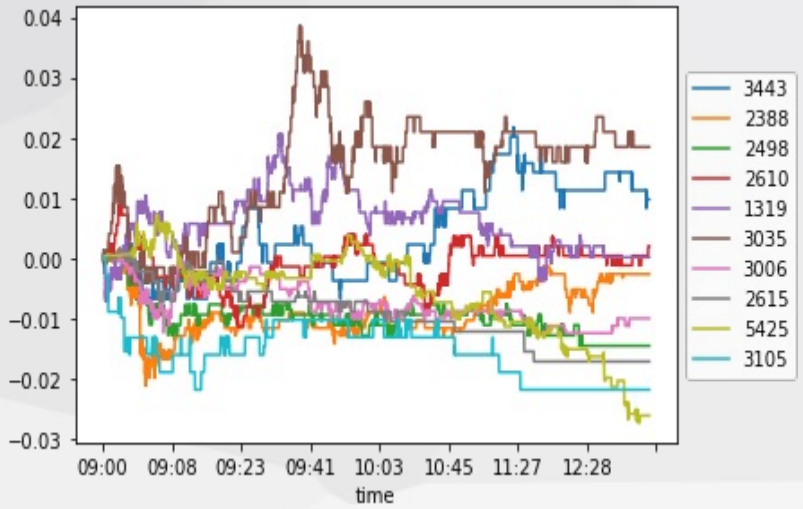
Back testing

Figure and data from different “‘GBRT_’ + date” folders

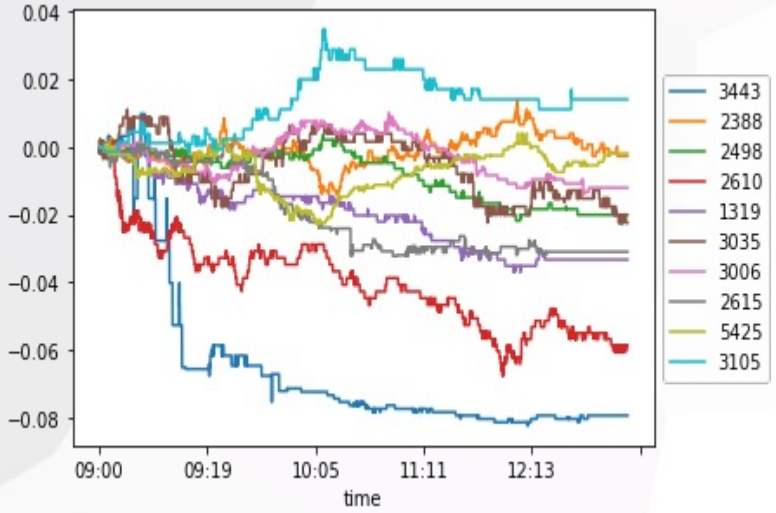
2023-06-21



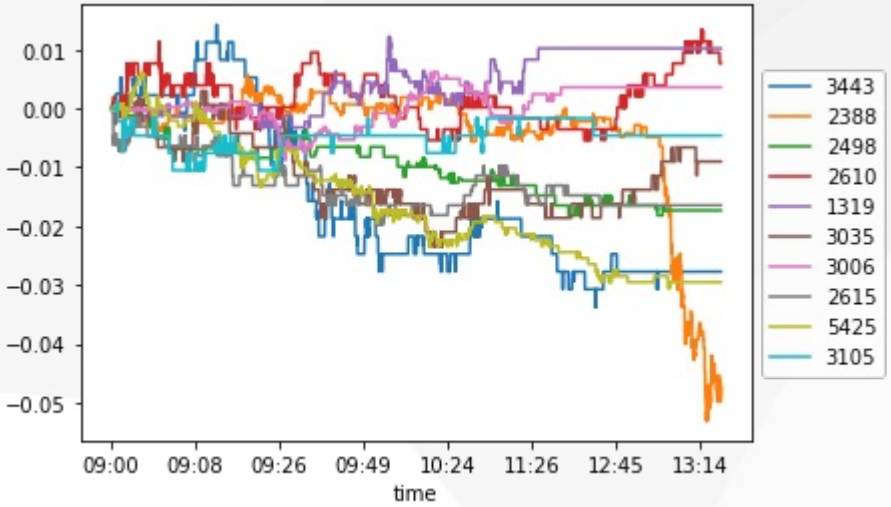
2023-06-26



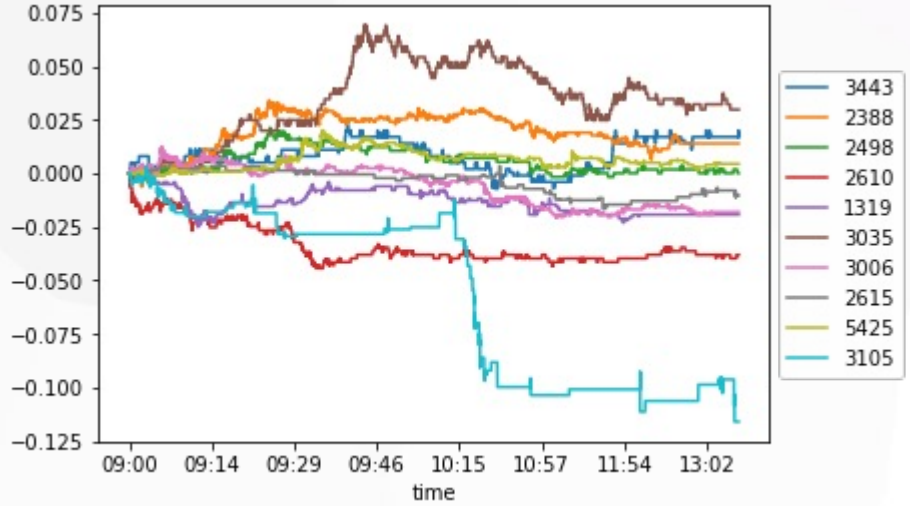
2023-06-27



2023-06-28



2023-06-29



Back testing

Information from ‘Mean and std for single stock.ipynb’

In [19]:
```
df
```

Out[19]:

	3443	2388	2498	2610	1319	3035	3006	2615	5425	3105
13:24:56.566000	-0.02625	-0.00417	-0.00364	0.002345	0.014585	0.0039	-0.00784	-0.01529	-0.03357	-0.01645
13:23:03.336000	0.00975	-0.00266	-0.01456	0.00198	0.000135	0.01845	-0.010015	-0.017175	-0.02616	-0.02185
13:24:01.890000	-0.0795	-0.002655	-0.020105	-0.058705	-0.03335	-0.02245	-0.012065	-0.03103	-0.0019	0.0141
13:21:10.018000	-0.02775	-0.047005	-0.017345	0.007642	0.01021	-0.009025	0.00358	-0.016465	-0.02954	-0.004625
13:24:18.715000	0.017	0.013965	0.000075	-0.037997	-0.01924	0.02975	-0.01802	-0.010285	0.004435	-0.115825

In [20]:
```
df.mean().sort_values(ascending = False)
```

Out[20]:
```
3035    0.004125
1319   -0.005532
2388   -0.008505
3006   -0.008872
2498   -0.011115
2610   -0.016947
5425   -0.017347
2615   -0.018049
3443   -0.021350
3105   -0.028930
dtype: float64
```

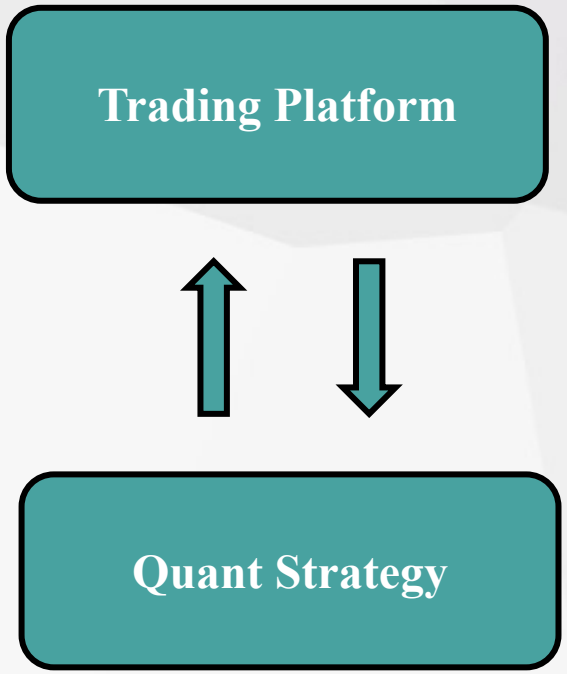
In [21]:
```
df.std().sort_values()
```

Out[21]:
```
2615    0.007741
3006    0.007926
2498    0.008840
5425    0.017339
1319    0.020288
3035    0.020863
2388    0.022772
2610    0.029673
3443    0.038355
3105    0.050495
dtype: float64
```

Back testing

- From the above information, we can find that our model does not perform well in this market, only one stock (3035) has got positive average return. So, in simulated trading we would trade that one.
- However, we can also find that 3035 has relatively high standard deviation. As a result, we would like to add several stocks to reduce the risk, then it comes to the stocks got negative return but not go too far from 0 (here we consider stocks have return higher than -0.01). Based on our condition, 1319, 2388 and 3006 could be take into consideration.
- Then we combine the information with the figure, we would find that 2388 once generated relatively big loss (on 0628) and has relatively high standard deviation. For this reason, we would not invest on it.
- Above all, our final invest pool would be 3035, 1319 and 3006.

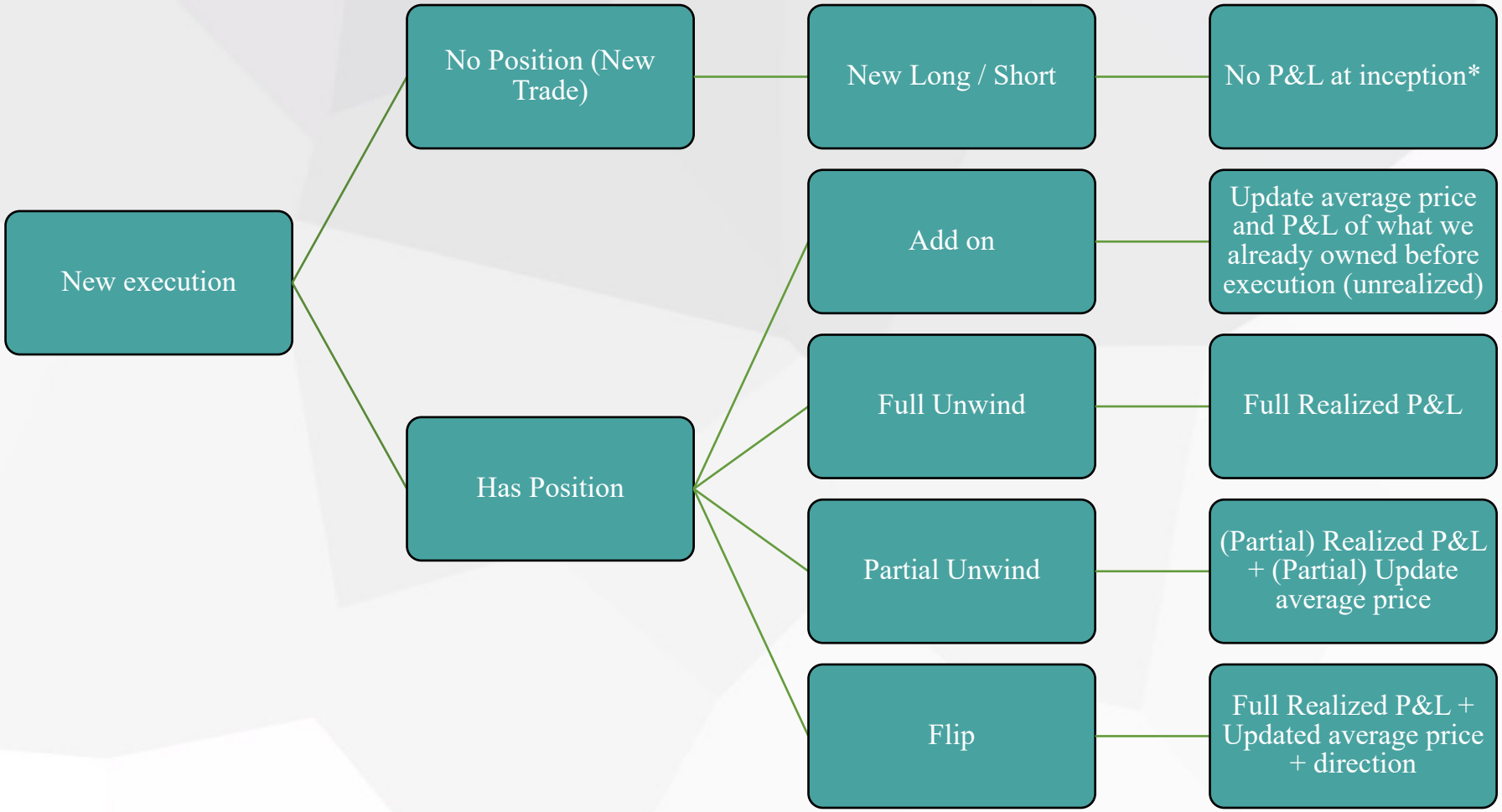
How to communicate with trading platform



- Get snapshot data of stocks and futures from Trading platform:
- Generating order to Trading platform
- Receive execution information from Trading platform

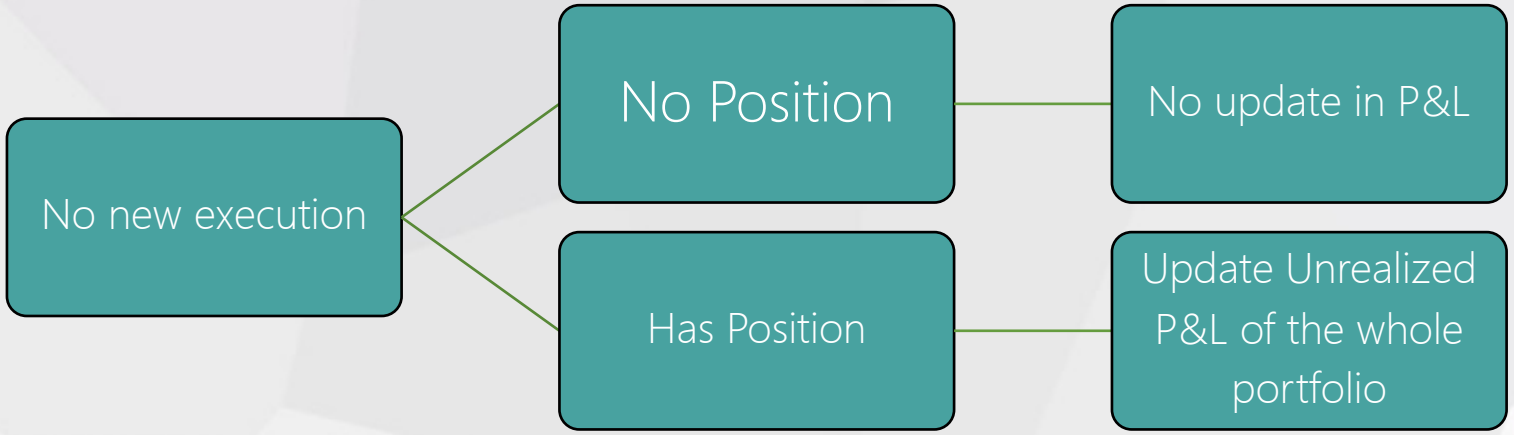
How to calculate and update P&L and other strategy metrics

Position P&L for individual stocks:



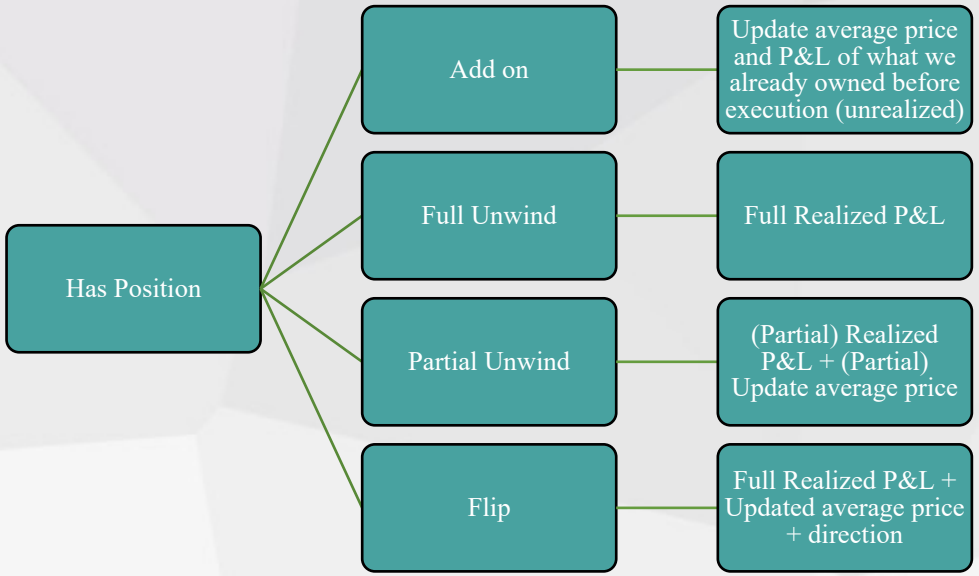
* We assume no trading cost (no levy, stamp, exchange fee and commission)

How to calculate and update P&L and other strategy metrics



- Unrealized P&L
  - Long position:
    - P&L = Position x (latest Mid price – average traded price)
    - Position is positive
  - Short position:
    - P&L = Position x (average traded price – latest Mid price)
    - Position is negative

How to calculate and update P&L and other strategy metrics



Position P&L for individual stocks

- Add on (when execution direction = position direction)
  - New Average price = (filled size x filled price + old average price x abs(position))/(filled size +abs(position))
  - Update P&L of what we already owned before execution (unrealized): position before execution x (average traded price – filled price)
- Full Realized P&L (when filled size = position and opposite direction)
  - P&L = Position x (filled price – average traded price)
  - Position = +ve when long and –ve when short
- Partial Realized P&L (when executed size < position and opposition direction)
  - P&L = Filled Size x (filled price – average traded price)

How to calculate and update P&L and other strategy metrics

Other potential strategy metrics:

- Sharpe Ratio
  - measure the risk-adjusted return
  - to avoid overfocus on absolute P&L without consideration of risk
  - Sharpe Ratio = (Portfolio return – risk free rate) / portfolio volatility

- Max Drawdown
  - Measure the drop in the value of a portfolio from peak to bottom
  - Max drawdown = (Trough value – peak value) / Peak Value

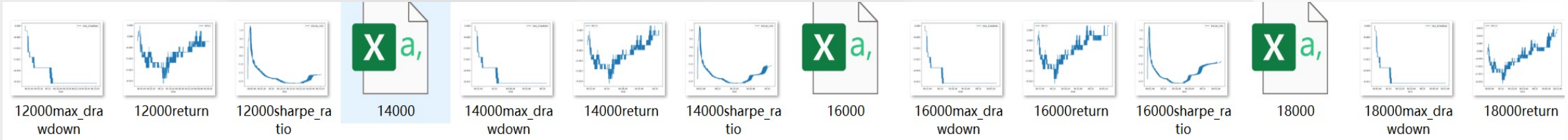
How to calculate and update P&L and other strategy metrics

```
[24384] Strategy.marketdata: position of 3035: 47

[24384] Strategy.marketdata: current cash amount is 730.0
[24384] Strategy.marketdata: current asset amount is 1001655.0
[24384] Strategy.marketdata: current invest status of the portfolio:
 return                0.002385
sharpe_ratio          -1.07376
max_drawdown          -0.02583
total_pnl               2385.0
std                   0.006102
time           09:34:58.729000
```

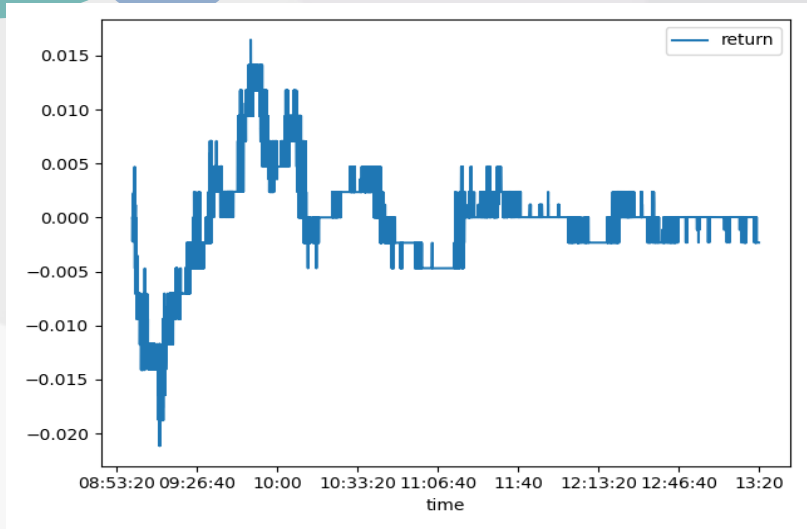
- Whenever we receive a new market data of stocks, we will print out our current ‘position’ (stands for the number of stocks we have here) of that stock, our current cash amount and asset amount, our current investment status (include return, sharpe ratio, max drawdown, total P&L, standard deviation of return and the time of that line of market data).

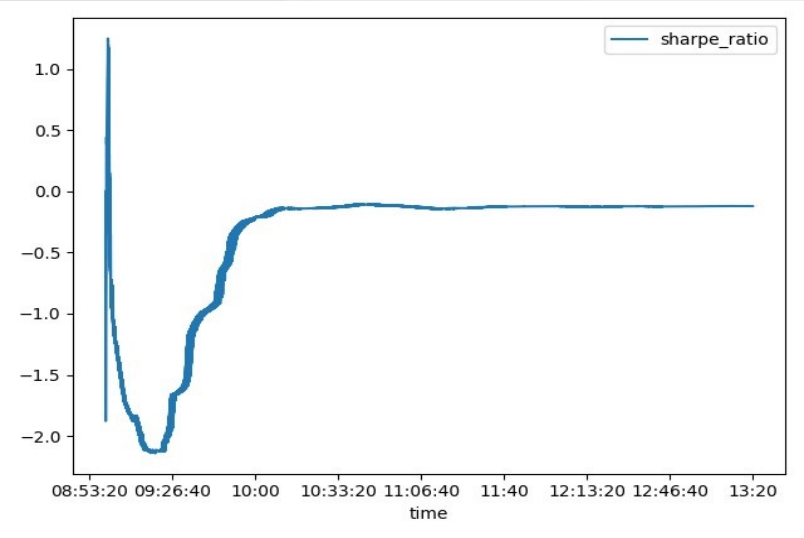
How to calculate and update P&L and other strategy metrics

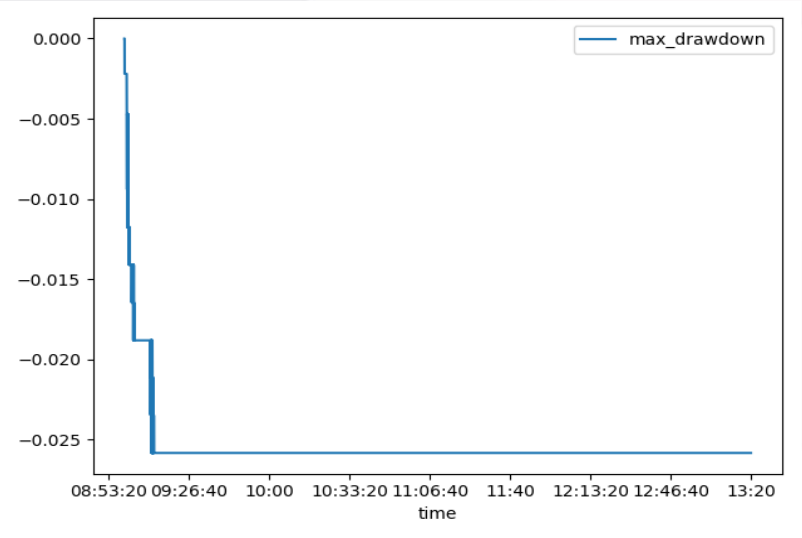


- For every 2000 lines of strategy metrics we calculated, we will create a csv file of them and create three figures (return, sharpe ratio and max drawdown (may have some fluctuations which might be due to multi-thread issue at the beginning)).

How to calculate and update P&L and other strategy metrics







- Plot of return, sharpe ratio and max drawdown, after the last plot has generated (40000 lines of metrics here).

How to calculate and update P&L and other strategy metrics

```
[27008] Strategy.marketdata: position of 3035: 47

[27008] Strategy.marketdata: current cash amount is 730.0
[27008] Strategy.marketdata: current asset amount is 999305.0
[27008] Strategy.marketdata: current invest status of the portfolio:
 return                0.000035
sharpe_ratio          -0.127908
max_drawdown          -0.025714
total_pnl                  35.0
std                    0.006649
time           13:24:58.440000
Name: 0, dtype: object
```

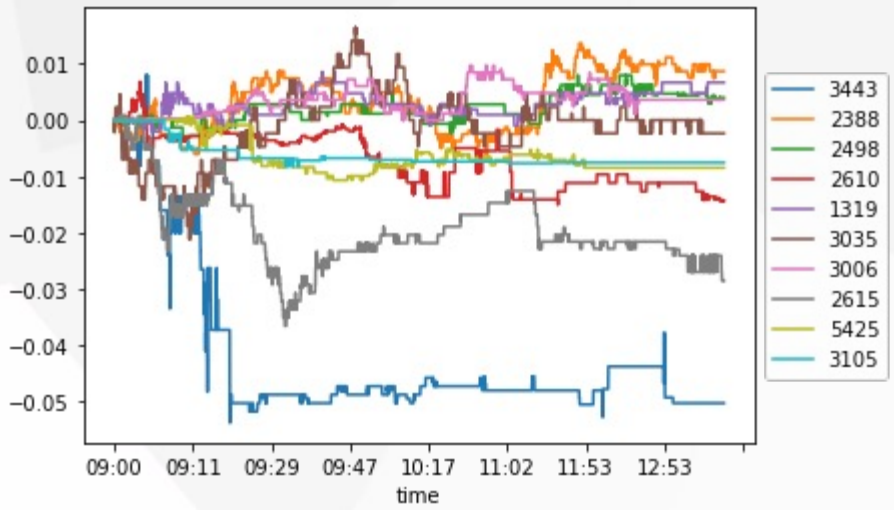
- Final status of our portfolio after receiving the last market data.

More information

Trading stocks individually with our strategy (see it after getting the performance of our portfolio to see the relative performance of the stocks we selected to invest).

Figure and data from 'GBRT_0630' folder, this is the information in simulated trading.

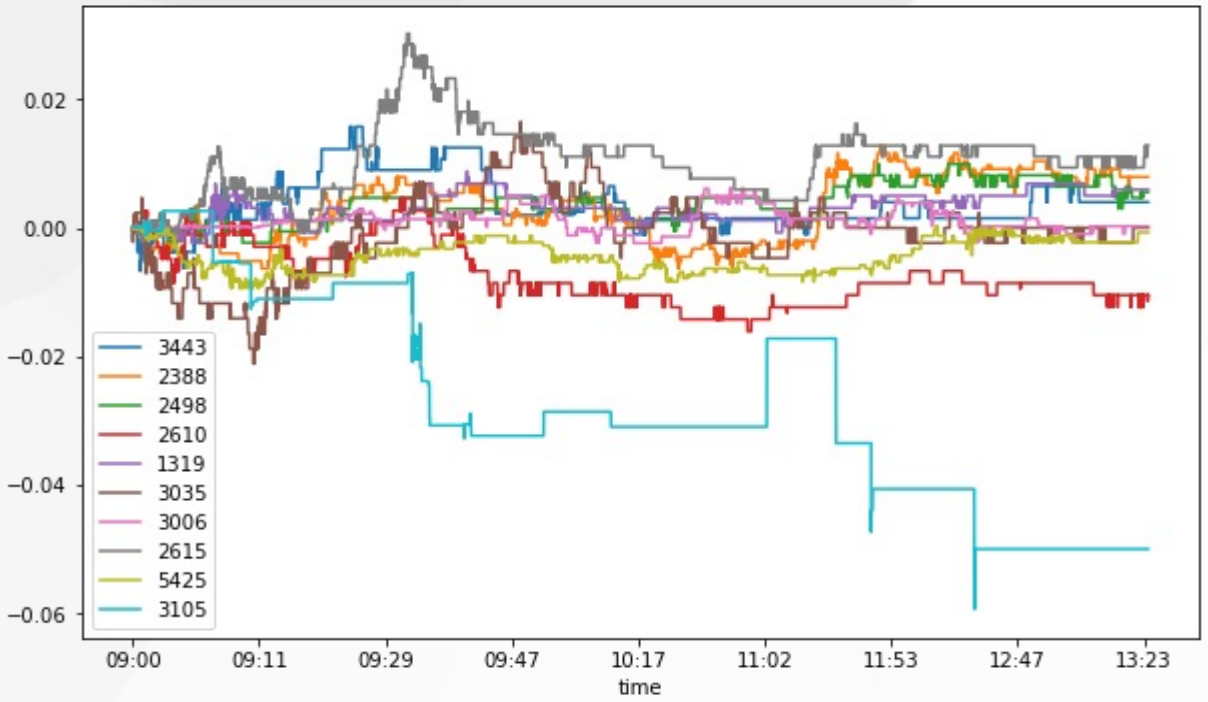
- As we can see, we have managed to select 3 of 5 stocks to trade that have the top 5 return when they are traded individually with our model. And have generated positive return while most of 10 stocks here have negative return (and some of them are much less than 0). As a result, we consider we have selected stocks that are suitable for our model.



More information

Performance of linear model for individually traded stocks Figure and data from ‘linear’ folder

- As we can see, linear model has got more stocks with return close or higher than 0, it may have better performance than GBRT strategy. In future work, we can try to do the similar process in previous part with linear regression.



Future improvement of Quant strategy

- Find more features to train the model, and take feature importance into consideration, using important features to predict.
- Add model selection process, try different models and find suitable models or strategies for different stocks.
- Consider more about when and how to submit order (here just simply submit order when predicted return larger than or less than and the submitted price is always askPrice1 or bidPrice1).
- Reasonably allocate investment amount on each stocks.

END

THANK YOU