CAPSTONE PROJECT REPORT

Submitted in Partial Fulfillment

Of the Requirements for the

Degree of

MASTER OF SCIENCE (Financial Engineering)

At the

TANDON SCHOOL OF NEW YORK UNIVERSITY

Zicheng He

Dec, 6 2020

Contents

IMPLEMENT OF BOLLINGER BANDS TRADING STRATEGY ON U.S STOCK MARKET

By

Zicheng He

Advisor: Song Tang, Ph.D

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science (Financial Engineering)

Dec 6, 2020

ABSTRACT

The purpose of this Capstone project was to implement Bollinger bands trading strategy in a real-time, simulated market. By using the market data from eod historical data api, we are able to select the stocks to trade, also train the parameters used in the strategy. After resist the stocks and parameters into database, we can then do back test on a forward period or implement the strategy in a simulated market, which simulates the real-time trading scenario.

Acknowledgements

I would like to extend my sincere gratitude to Prof. Song Tang for his expert guidance and constructive feedback throughout the course of this project. I appreciate the patience with which he would tackle my doubts and the time he would spare from his busy schedule to meet during the course of his work-day to discuss. His inputs to this project are invaluable. I would also like to thank the Finance and Risk Engineering department at NYU Tandon to give me the opportunity to work on this advanced topic for my capstone project, during the course of which I learned a lot.

List of Abbreviations

BB (BBD) - Bollinger Band

MA - Moving Average

STD - Standard Deviation

PNL - Profit and Loss

1. Introduction

The whole project was taking forward step by step and can be divided into three parts. The first part is to design and develop the trading model, which in our case is the BB trading model. During this part, we designed and implement the trading logic, training method and back testing period.

The second part is to design and develop the front-end web page by Flask, and connect each function in the first part to the web page section. The functions include: Introduction, Stock selection, Training process, Back Test, Probation Test.

The last part is to connect the client side to the server side and implement the auto trading logic, in order to simulate a real-time trading scenario in a simulated market. Then eventually, we also need to put these functions into the front-end web page, in which the sections include: Manual trading, Start trading, and client down.

Next we will introduce each part step by step.

2. Bollinger Band Trading Model

2.1 Stocks Pool Selection:

In order to select our universe to implement our trading model, the first step is to decide our stock pool. For this section, within S&P500 component pool, we select 10 stocks with highest 5-minute volatility over the last month of the historical data part.

Reason for selecting S&P 500:

1. S&P 500 components are with large size, high liquidity.

2. Since using intraday data for generating trading signals, we check the vol among minutes over one month (a short period)

3. Higher volatility may trigger signals more often, more signals can lead better test the efficacy of trading model (i.e. training params)

2.2 Base Trading Model: Bollinger Bands (BB)

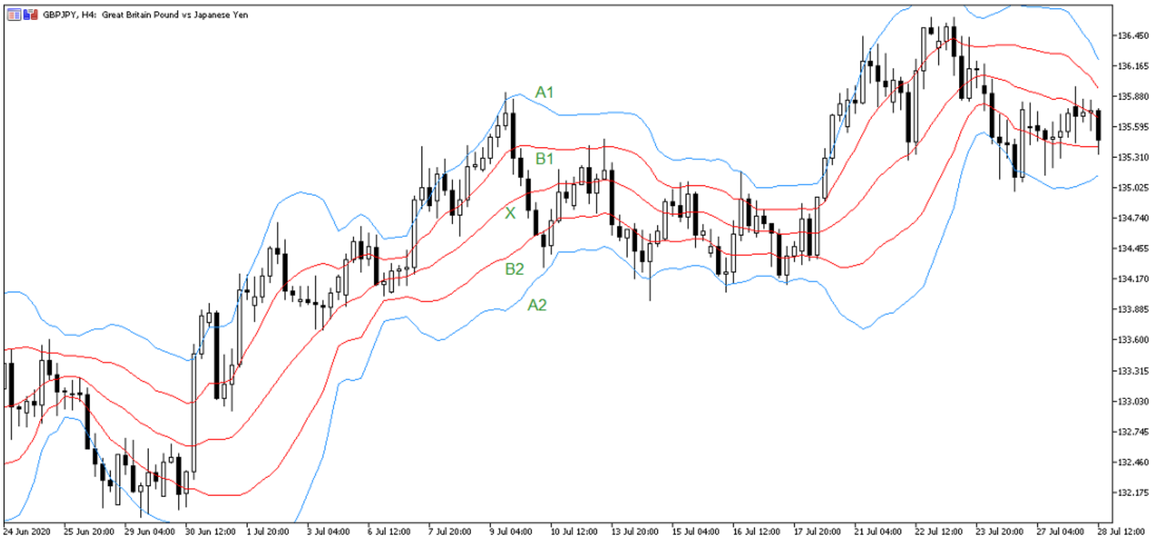


Figure 1 Bollinger Bands

1. Model Description

Bollinger Bands are a type of price envelope developed by John Bollinger. (Price envelopes define upper and lower price range levels.) Bollinger Bands are envelopes plotted at a standard deviation level above and below a simple moving average of the price. Because the distance of the bands is based on standard deviation, they adjust to volatility swings in the underlying price.

Instead using the trend strategy Double BBs that we used before(which is a strategy for trend, not fitting the 1-minute data), we use the single Bollinger Bands, it is a mean-reversion strategy so that it can capture the tiny structure during the market with 1-minute data.

A1: The upper band line that is k1 (default is 2) standard deviations away from line X, which is the H-period simple moving average (SMA of H mins, H is 20 in default)

X: The H-period SMA. This serves as both the center of the BB, and the baseline for determining the location of the two bands

A2: The lower band line that is k1 standard deviations from the H-period SMA

And our strategy will be triggered based on these trading signals:

These bands represent distinct trading zones used by our program to place trades:

1. The Long Zone is below A2 (the lower band) -> Long signal appears when close < A2

2. The Flat Zone is between A2 and A1 -> Flat signal appears when close price falls between A1 and A2.

3. The Short Zone is above A1 (the upper band) -> Short signal appears when close > A1

According to Figure 1, the price can fall into theses zones:

- Long Zone:

When the price is within this bottom zone, it tells us that the stock is oversold, and that there is a higher chance that the price will continue upward according to the thought of "MEAN-REVERSION". As long as next minute price continue to close in the bottom zone, we maintain long positions or open new trades.

- Short Zone:

When the price is in the upper zone, the stock will get higher chance to get down. That tells us that as long as the price close in the upper zone, we should maintain current short positions or open new ones.

- Flat Zone:

When the price gets within the area defined by the two bands (A1 and A2), there is no strong signals to tell it will move up or down, and the price is likely to fluctuate within a trading range. So here we remain flat or close our existing positions.

2. Trading Logic

1) X: The H-period SMA. This serves as both the center of the BBs, and the baseline for determining the location of the two bands

2) Up : The upper band line that is k1 (default is 2) standard deviation from the H-minute SMA

3) Down : The lower band line that is k1 standard deviation from the H-period SMA

4) Open Trades:
- Open short position if Close >= Up, sell the stock
- Open long position if Close <= Down, buy the stock

5) Close Trades:
- Close the open positions when Close return within Up and Down

3. The Parameters To train:

- K1, H

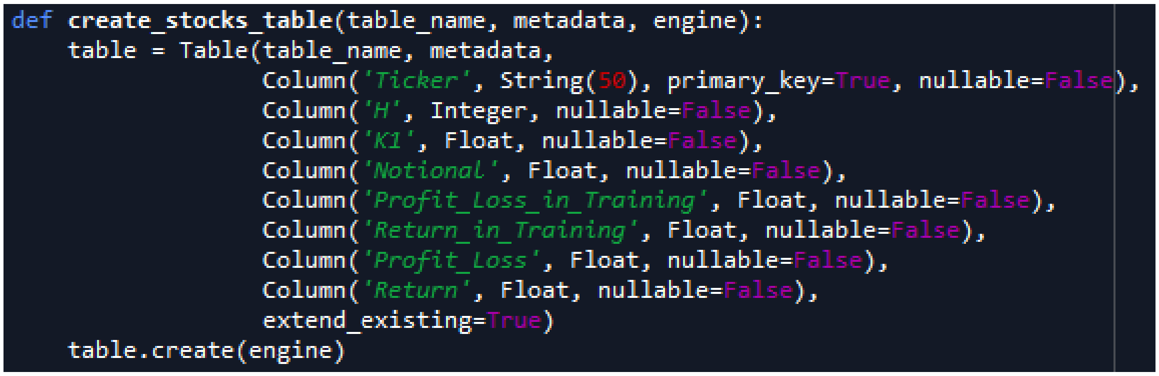
The parameters combination can be trained or tested within the historical data period, once we entered the simulated market. We use the same parameters for the whole simulated period, so the calculation speed and the signal-generating process should be fast.

2.3 Market Data Feed Description

Within my market data feed, it could be divided into steps below:

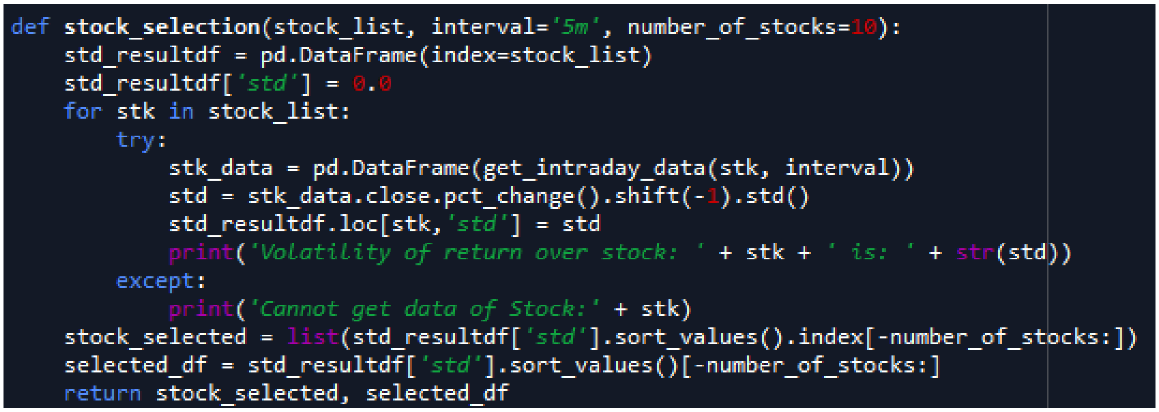
In order to build model:

Step1. Create table “Stocks” by calling function “**create_stocks_table**”, it contains: **Ticker, H, K1,** and **PnL,** where H, K1 are parameters that are not decided (trained) yet.



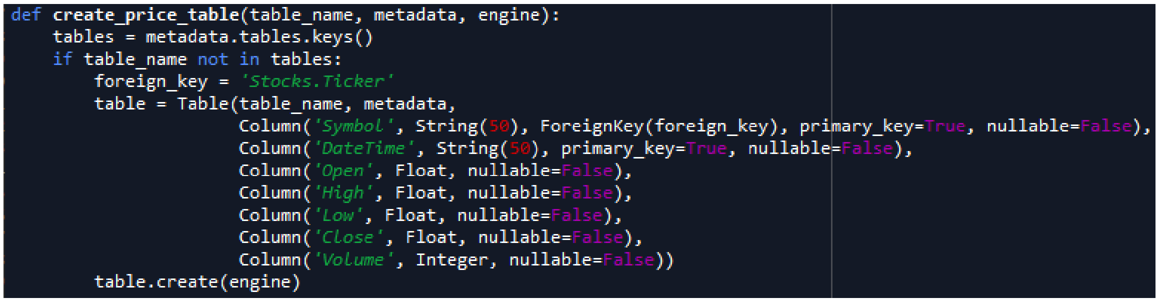
```
def create_stocks_table(table_name, metadata, engine):
    table = Table(table_name, metadata,
                  Column('Ticker', String(50), primary_key=True, nullable=False),
                  Column('H', Integer, nullable=False),
                  Column('K1', Float, nullable=False),
                  Column('Notional', Float, nullable=False),
                  Column('Profit_Loss_in_Training', Float, nullable=False),
                  Column('Return_in_Training', Float, nullable=False),
                  Column('Profit_Loss', Float, nullable=False),
                  Column('Return', Float, nullable=False),
                  extend_existing=True)
    table.create(engine)
```

Step2. Call function “**stock_selection”,** which gets the intraday data of S&P 500 components over the last 30 days (interval as 5 mins), then it calculates std of the return of each stocks and selects **the top 10 stocks**. Populate table “Stocks” with these 10 stocks.



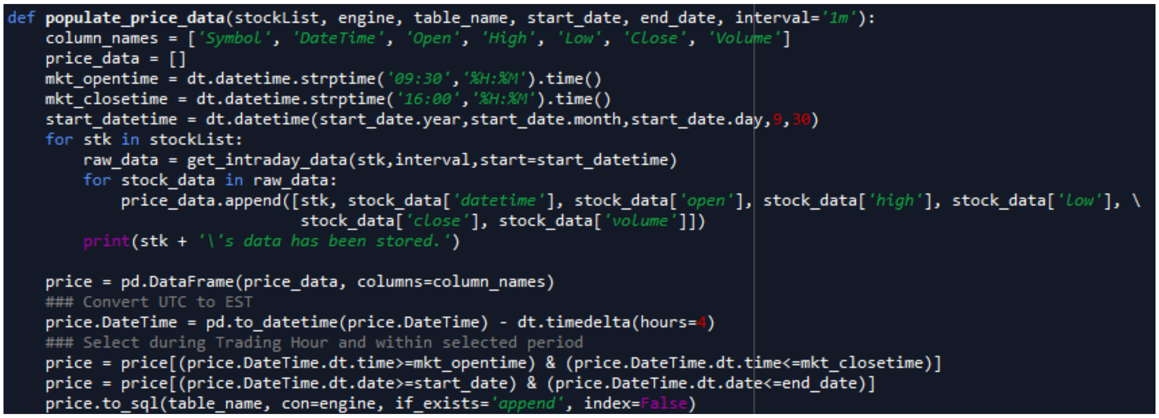
```
def stock_selection(stock_list, interval='5m', number_of_stocks=10):
    std_resultdf = pd.DataFrame(index=stock_list)
    std_resultdf['std'] = 0.0
    for stk in stock_list:
        try:
            stk_data = pd.DataFrame(get_intraday_data(stk, interval))
            std = stk_data.close.pct_change().shift(-1).std()
            std_resultdf.loc[stk,'std'] = std
            print('Volatility of return over stock: ' + stk + ' is: ' + str(std))
        except:
            print('Cannot get data of Stock:' + stk)
    stock_selected = list(std_resultdf['std'].sort_values().index[-number_of_stocks:])
    selected_df = std_resultdf['std'].sort_values()[-number_of_stocks:]
    return stock_selected, selected_df
```

Step3. Create table “Price” by calling function “**create_price_table”,** use Stocks.Ticker as its foreign keys, then clear the whole table.



```
def create_price_table(table_name, metadata, engine):
    tables = metadata.tables.keys()
    if table_name not in tables:
        foreign_key = 'Stocks.Ticker'
        table = Table(table_name, metadata,
                      Column('Symbol', String(50), ForeignKey(foreign_key), primary_key=True, nullable=False),
                      Column('DateTime', String(50), primary_key=True, nullable=False),
                      Column('Open', Float, nullable=False),
                      Column('High', Float, nullable=False),
                      Column('Low', Float, nullable=False),
                      Column('Close', Float, nullable=False),
                      Column('Volume', Integer, nullable=False))
        table.create(engine)
```

Step4. Populate table “Price” with intraday data over the last 30 days (interval as 1 min) by the function “**populate_price_data**”.



```python
def populate_price_data(stockList, engine, table_name, start_date, end_date, interval='1m'):
    column_names = ['Symbol', 'DateTime', 'Open', 'High', 'Low', 'Close', 'Volume']
    price_data = []
    mkt_opentime = dt.datetime.strptime('09:30','%H:%M').time()
    mkt_closetime = dt.datetime.strptime('16:00','%H:%M').time()
    start_datetime = dt.datetime(start_date.year,start_date.month,start_date.day,9,30)
    for stk in stockList:
        raw_data = get_intraday_data(stk,interval,start=start_datetime)
        for stock_data in raw_data:
            price_data.append([stk, stock_data['datetime'], stock_data['open'], stock_data['high'], stock_data['low'], \
                               stock_data['close'], stock_data['volume']])
        print(stk + '\'s data has been stored.')

    price = pd.DataFrame(price_data, columns=column_names)
    ### Convert UTC to EST
    price.DateTime = pd.to_datetime(price.DateTime) - dt.timedelta(hours=4)
    ### Select during Trading Hour and within selected period
    price = price[(price.DateTime.dt.time>=mkt_opentime) & (price.DateTime.dt.time<=mkt_closetime)]
    price = price[(price.DateTime.dt.date>=start_date) & (price.DateTime.dt.date<=end_date)]
    price.to_sql(table_name, con=engine, if_exists='append', index=False)
```

Note: During step4, actually we can set up interval as "1m" using the API provided. However, it returns pre-market data and also post-market data from 2019-7-1. Hence, within the function "**populate_price_data",** we do two things manually:

1. Change the datetime from UTC to EST

2. Select only those within trading hours (i.e. from 9:30 am to 4:00 pm) during the period we want (here I choose one month, but can alter it if we need in the future)

2.4 Parameters Training

Params: H (look back period for computing MA and rolling std)

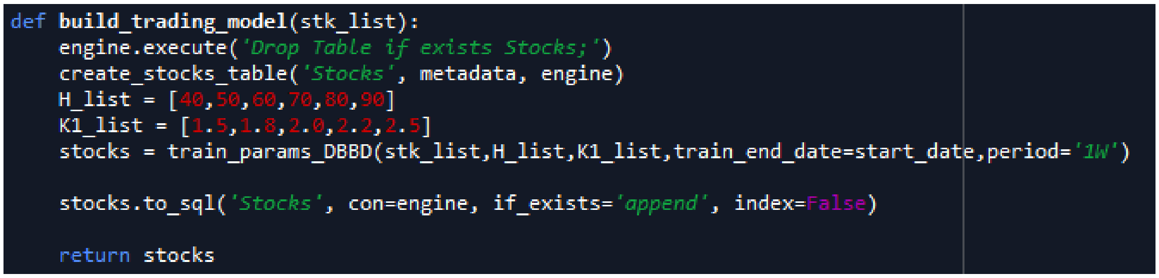
K1: width of the band

Training period: 1 week before the recent 1 week, i.e, 11.21 – 11.28.

That is because for the recent week (11.29 - 12.6), it is used for back testing and we don't want our training period and back testing period to overlap. For the recent week, it is used for back testing. And the reason we use 1 week is because the parameters are short-memory.

Calculation method (Training method): Grid Search.

Since the default parameters of this strategy is H as 60, K1 as 2, so we set those default numbers as centers, and let H range from 40 to 90, let K1 range from [1.5,1.8,2.0,2.2,2.5].



```
def build_trading_model(stk_list):
    engine.execute('Drop Table if exists Stocks;')
    create_stocks_table('Stocks', metadata, engine)
    H_list = [40,50,60,70,80,90]
    K1_list = [1.5,1.8,2.0,2.2,2.5]
    stocks = train_params_DBBD(stk_list,H_list,K1_list,train_end_date=start_date,period='1W')

    stocks.to_sql('Stocks', con=engine, if_exists='append', index=False)

    return stocks
```

At each pair, calculate its information ratio, which is here defined as annualized return over annualized volatility.

Instead of searching for the global maximum, here I choose the right parameters with the max mean when tuning the other parameter. For instance, imagine there is a matrix representing the params grid (H * K1), then I select the right H from the row with highest IR mean, select the right K1 from the column with highest IR.

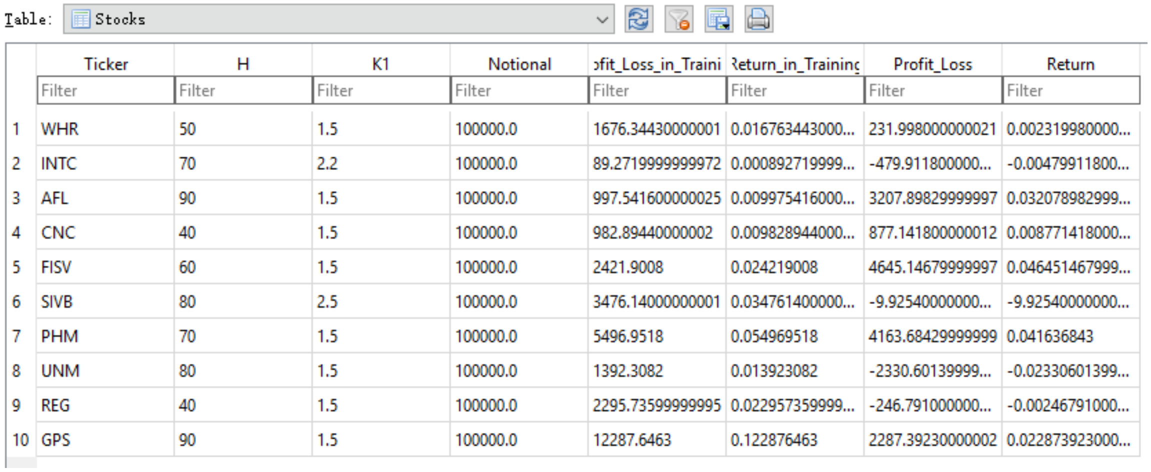
Persist data

After doing grid search, I persist the selected parameters (H, K1) and also the PnL during training to Stocks table.

2.5 Back Test for BB Trading Strategy

Back Test Period: the recent 1 week, i.e, 11.29 - 12.6

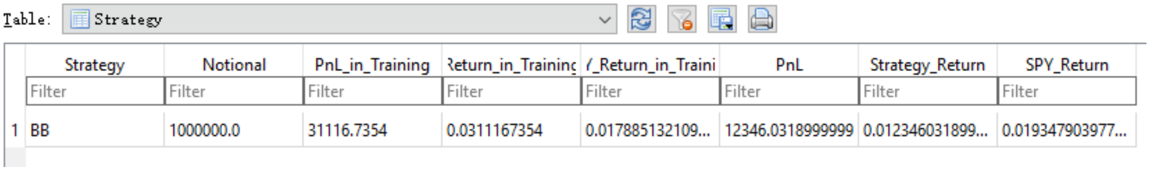
Then we use the right pair (H, K1) from training, doing back test on the recent 1week data. Eventually we store the back test PnL & return into stocks table. As you can see in the screenshot:



	Ticker	H	K1	Notional	ofit_Loss_in_Traini	Return_in_Training	Profit_Loss	Return
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	WHR	50	1.5	100000.0	1676.34430000001	0.016763443000...	231.998000000021	0.002319980000...
2	INTC	70	2.2	100000.0	89.2719999999972	0.000892719999...	-479.911800000...	-0.00479911800...
3	AFL	90	1.5	100000.0	997.541600000025	0.009975416000...	3207.89829999997	0.032078982999...
4	CNC	40	1.5	100000.0	982.89440000002	0.009828944000...	877.141800000012	0.008771418000...
5	FISV	60	1.5	100000.0	2421.9008	0.024219008	4645.14679999997	0.046451467999...
6	SIVB	80	2.5	100000.0	3476.14000000001	0.034761400000...	-9.92540000000...	-9.92540000000...
7	PHM	70	1.5	100000.0	5496.9518	0.054969518	4163.68429999999	0.041636843
8	UNM	80	1.5	100000.0	1392.3082	0.013923082	-2330.60139999...	-0.02330601399...
9	REG	40	1.5	100000.0	2295.73599999995	0.022957359999...	-246.791000000...	-0.00246791000...
10	GPS	90	1.5	100000.0	12287.6463	0.122876463	2287.39230000002	0.022873923000...

Strategy Table:

Here we persist the overall strategy PnL & return (both during training period and from back test period). We also make them compare to the performance of longing SPY, which is a benchmark of return. As you may see in below:



	Strategy	Notional	PnL_in_Training	Return_in_Training	/_Return_in_Traini	PnL	Strategy_Return	SPY_Return
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	BB	1000000.0	31116.7354	0.0311167354	0.017885132109...	12346.0318999999	0.012346031899...	0.019347903977...

Which in total generate a weekly return of 1.5%, so the model’s performance is really time-varying.

3. Front-End Web Page Design and Development

3.1 Flask description

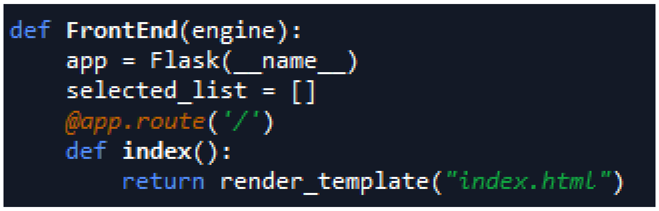
This part is for specifying each piece that we came across related to the Flask part during the capstone project. It will show my understanding over each module and how we fulfill the requirement on the client front end side.

app = Flask(__name__)

Here we create an app object, which is an instance of the Flask class. It'll act as the central configuration object for the entire application for our web page. It will take in the "__name__ "of the script file. This lets Python know how to import from files relative to this one.

3.2 Route in each module

@app.route('/')



```
def FrontEnd(engine):
    app = Flask(__name__)
    selected_list = []
    @app.route('/')
    def index():
        return render_template("index.html")
```

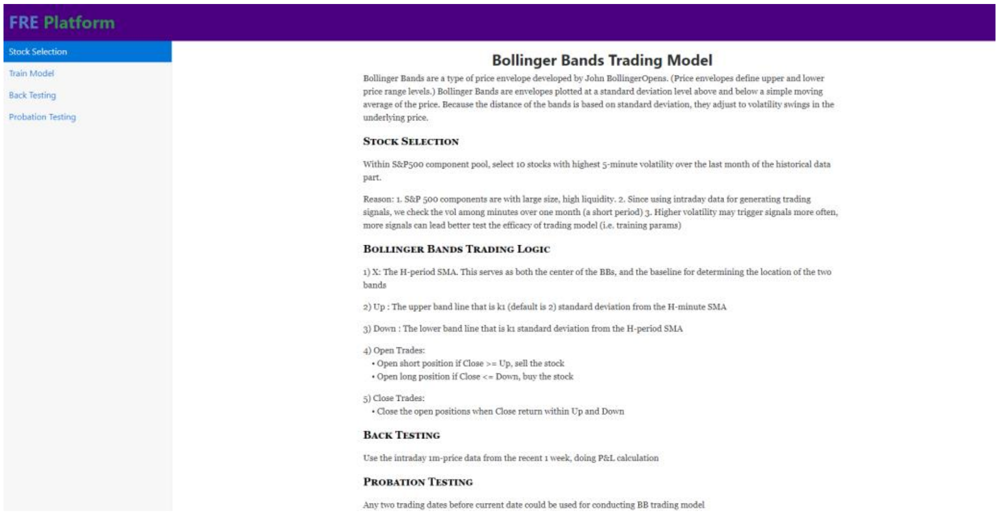
The **app.route** decorator decorates the first view function; it can specify one of the routes used to access the application. Any view we specify must be decorated by this "app.route" to be a functional part of the web page. In this project, we have many templates for us to render our result into, I will illustrate them one by one as below, and also I am going to attach the screen shot for each one from my side:

1. **Base.html**

   This is the baseline template for our project. Every other templates will form in the same format as the base, they will extend it to have their own characteristics.

2. **Index.html**

This is the introduction for the trading model. The code for it is made up of several modules, each one get parameters for the font, so that it can show in a pretty page.



Bollinger Bands Trading Model

Bollinger Bands are a type of price envelope developed by John BollingerOpens. (Price envelopes define upper and lower price range levels.) Bollinger Bands are envelopes plotted at a standard deviation level above and below a simple moving average of the price. Because the distance of the bands is based on standard deviation, they adjust to volatility swings in the underlying price.

Stock Selection

Within S&P500 component pool, select 10 stocks with highest 5-minute volatility over the last month of the historical data part.

Reason: 1. S&P 500 components are with large size, high liquidity. 2. Since using intraday data for generating trading signals, we check the vol among minutes over one month (a short period) 3. Higher volatility may trigger signals more often, more signals can lead better test the efficacy of trading model (i.e. training params)

Bollinger Bands Trading Logic

1) X: The H-period SMA. This serves as both the center of the BBs, and the baseline for determining the location of the two bands

2) Up : The upper band line that is k1 (default is 2) standard deviation from the H-minute SMA

3) Down : The lower band line that is k1 standard deviation from the H-period SMA

4) Open Trades:
- Open short position if Close >= Up, sell the stock
- Open long position if Close <= Down, buy the stock

5) Close Trades:
- Close the open positions when Close return within Up and Down

Back Testing

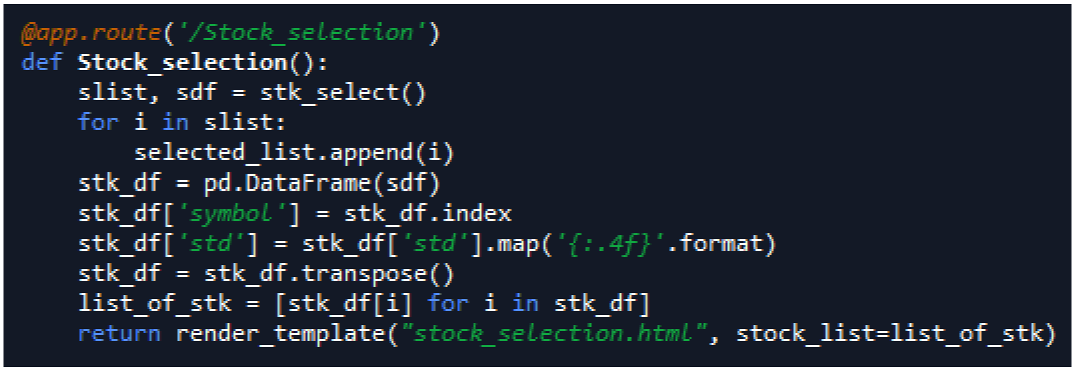
Use the intraday 1m-price data from the recent 1 week, doing P&L calculation

Probation Testing

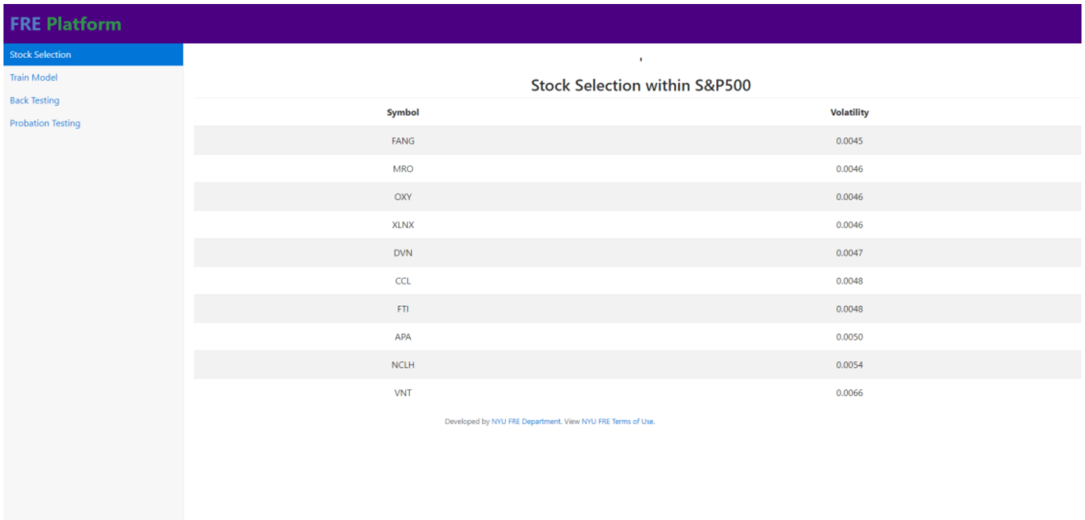
Any two trading dates before current date could be used for conducting BB trading model

3. Stock_selction.html

This template actually is part of the build model part, I separate them since I may want to check and show what stocks are with the highest volatility within the SP500. From this template on, we will need to select the data result from our python program, and then render the result into the formatted input so as to fit the template. For most cases, we use list to store the row for data frame and then send the list to the template.



```python
@app.route('/Stock_selection')
def Stock_selection():
    slist, sdf = stk_select()
    for i in slist:
        selected_list.append(i)
    stk_df = pd.DataFrame(sdf)
    stk_df['symbol'] = stk_df.index
    stk_df['std'] = stk_df['std'].map('{:.4f}'.format)
    stk_df = stk_df.transpose()
    list_of_stk = [stk_df[i] for i in stk_df]
    return render_template("stock_selection.html", stock_list=list_of_stk)
```



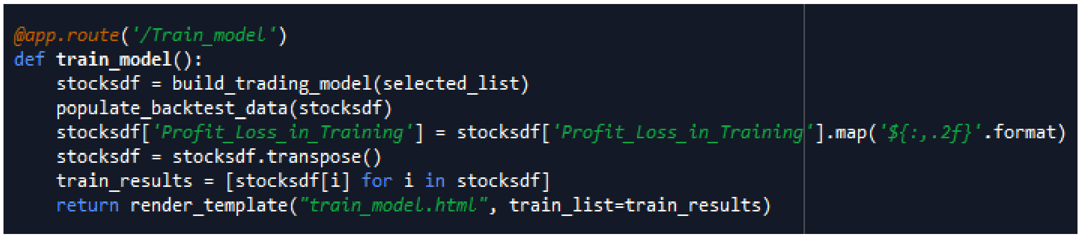
Stock Selection within S&P500

Symbol	Volatility
FANG	0.0045
MRO	0.0046
OXY	0.0046
XLNX	0.0046
DVN	0.0047
CCL	0.0048
FTI	0.0048
APA	0.0050
NCLH	0.0054
VNT	0.0066

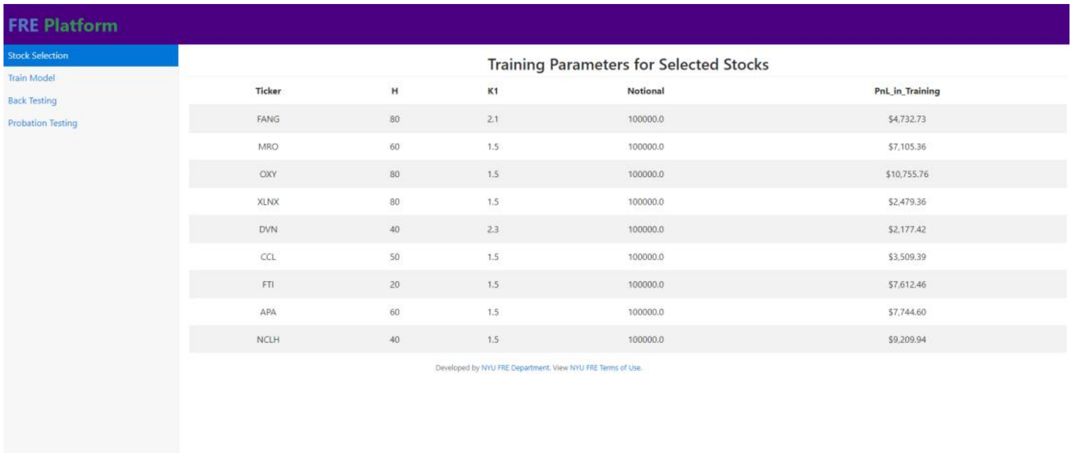
Developed by NYU FRE Department. View NYU FRE Terms of Use.

4. Train_model.html

This is the left part for building our model, basically it will train the parameters by grid search for each stock selected, also do the render template so that fits our requirement.



```
@app.route('/Train_model')
def train_model():
    stocksdf = build_trading_model(selected_list)
    populate_backtest_data(stocksdf)
    stocksdf['Profit_Loss_in_Training'] = stocksdf['Profit_Loss_in_Training'].map('${:,.2f}'.format)
    stocksdf = stocksdf.transpose()
    train_results = [stocksdf[i] for i in stocksdf]
    return render_template("train_model.html", train_list=train_results)
```



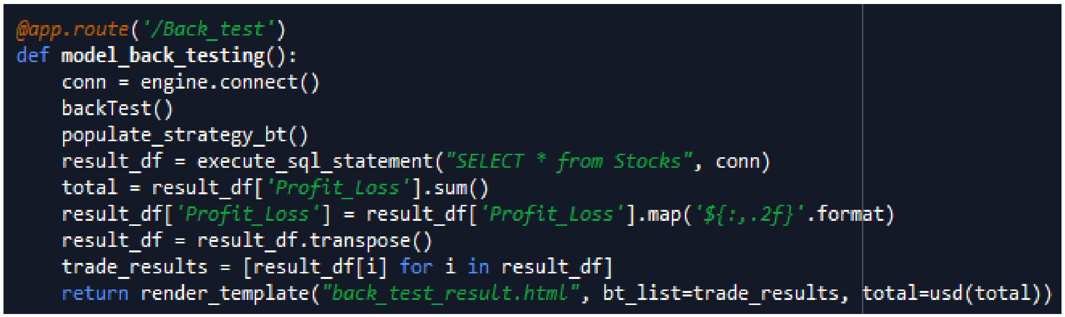
Training Parameters for Selected Stocks

Ticker	H	K1	Notional	PnL_in_Training
FANG	80	2.1	100000.0	$4,732.73
MRO	60	1.5	100000.0	$7,105.36
OXY	80	1.5	100000.0	$10,755.76
XLNX	80	1.5	100000.0	$2,479.36
DVN	40	2.3	100000.0	$2,177.42
CCL	50	1.5	100000.0	$3,509.39
FTI	20	1.5	100000.0	$7,612.46
APA	60	1.5	100000.0	$7,744.60
NCLH	40	1.5	100000.0	$9,209.94

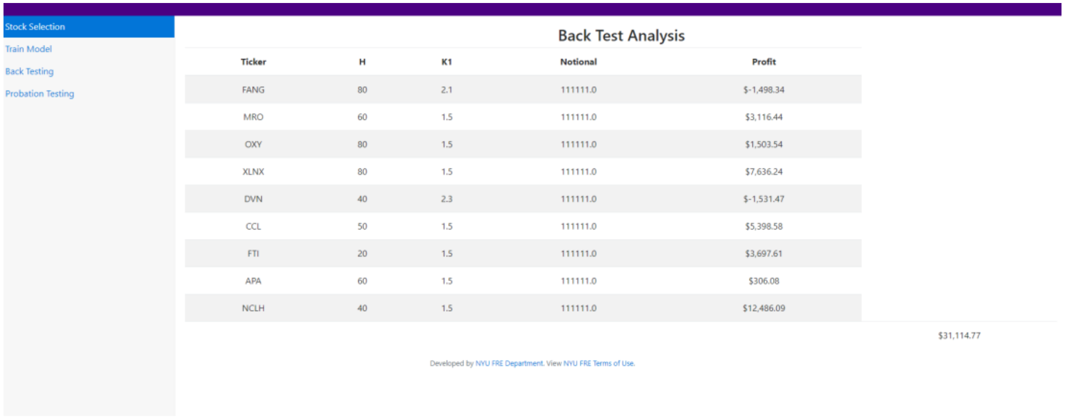
Developed by NYU FRE Department. View NYU FRE Terms of Use.

5. Back_test.html

Back testing the model within the recent week, then render the result into the formatted input for the template. Also, we show the total pnl in the bottom right corner.



```
@app.route('/Back_test')
def model_back_testing():
    conn = engine.connect()
    backTest()
    populate_strategy_bt()
    result_df = execute_sql_statement("SELECT * from Stocks", conn)
    total = result_df['Profit_Loss'].sum()
    result_df['Profit_Loss'] = result_df['Profit_Loss'].map('${:,.2f}'.format)
    result_df = result_df.transpose()
    trade_results = [result_df[i] for i in result_df]
    return render_template("back_test_result.html", bt_list=trade_results, total=usd(total))
```



Back Test Analysis

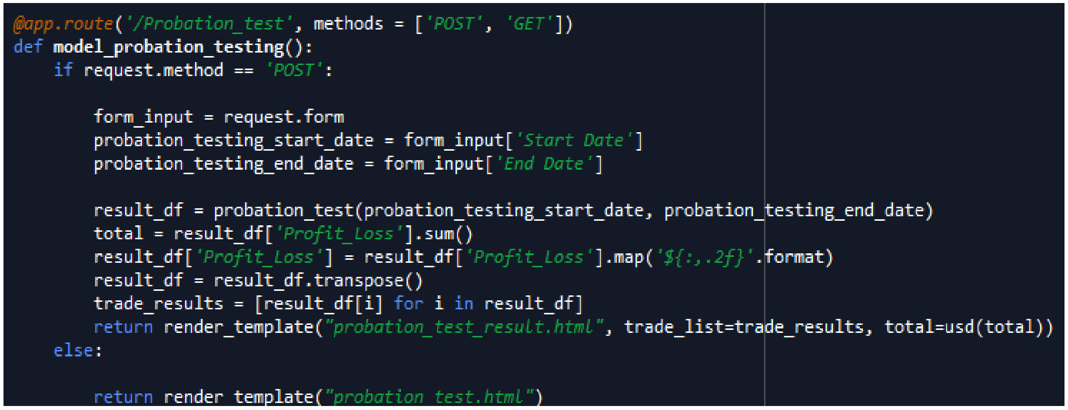
Ticker	H	K1	Notional	Profit
FANG	80	2.1	111111.0	$-1,498.34
MRO	60	1.5	111111.0	$3,116.44
OXY	80	1.5	111111.0	$1,503.54
XLNX	80	1.5	111111.0	$7,636.24
DVN	40	2.3	111111.0	$-1,531.47
CCL	50	1.5	111111.0	$5,398.58
FTI	20	1.5	111111.0	$3,697.61
APA	60	1.5	111111.0	$306.08
NCLH	40	1.5	111111.0	$12,486.09

$31,114.77

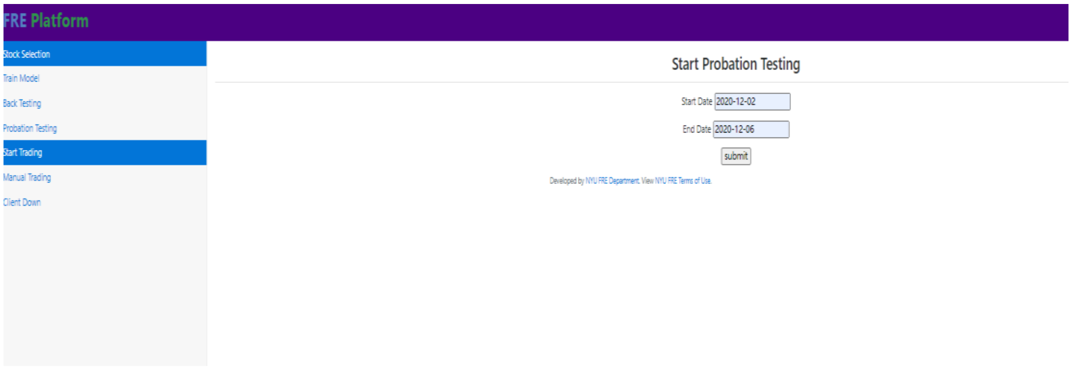
Developed by NYU FRE Department. View NYU FRE Terms of Use.

6. Probation_test.html & Probation_test_result.html

These two templates are different from above, actually there is a for loop for the methods within the application. When it does “GET”, it shows that page that from “Probation_test.html”, which requires us to input two dates. When we input them and click “submit”, the for loop continue so that it reach ”POST”, when the program will get the dates as input for the probation test. In the end, the test result will be rendered into the “Probation_test_result.html” template and show on the page.



```
@app.route('/Probation_test', methods = ['POST', 'GET'])
def model_probation_testing():
    if request.method == 'POST':

        form_input = request.form
        probation_testing_start_date = form_input['Start Date']
        probation_testing_end_date = form_input['End Date']

        result_df = probation_test(probation_testing_start_date, probation_testing_end_date)
        total = result_df['Profit_Loss'].sum()
        result_df['Profit_Loss'] = result_df['Profit_Loss'].map('${:,.2f}'.format)
        result_df = result_df.transpose()
        trade_results = [result_df[i] for i in result_df]
        return render_template("probation_test_result.html", trade_list=trade_results, total=usd(total))
    else:

        return render_template("probation_test.html")
```



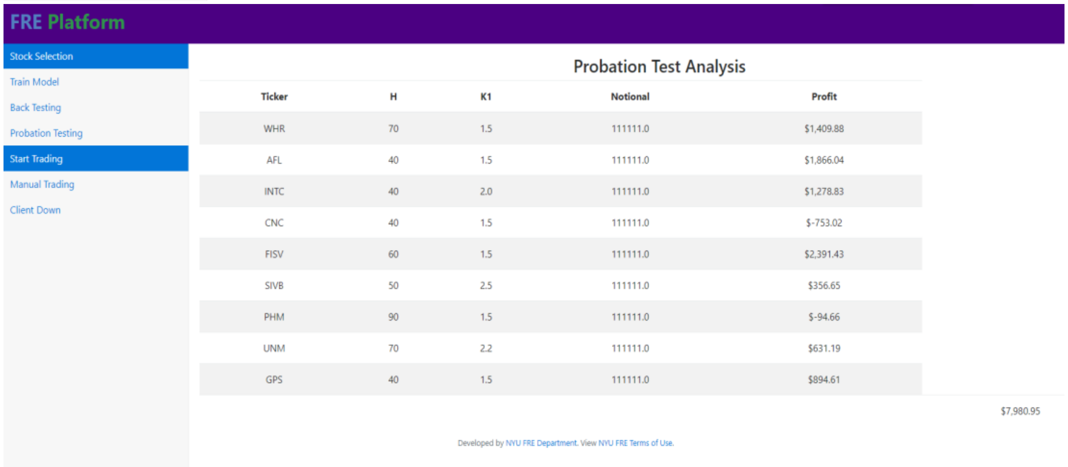
Start Probation Testing

Start Date 2020-12-02

End Date 2020-12-06

submit

Developed by NYU FRE Department. View NYU FRE Terms of Use.



Probation Test Analysis

Ticker	H	K1	Notional	Profit
WHR	70	1.5	111111.0	$1,409.88
AFL	40	1.5	111111.0	$1,866.04
INTC	40	2.0	111111.0	$1,278.83
CNC	40	1.5	111111.0	$-753.02
FISV	60	1.5	111111.0	$2,391.43
SIVB	50	2.5	111111.0	$356.65
PHM	90	1.5	111111.0	$-94.66
UNM	70	2.2	111111.0	$631.19
GPS	40	1.5	111111.0	$894.61

$7,980.95

Developed by NYU FRE Department. View NYU FRE Terms of Use.

3.3 Render Template and Request

1. Render_template

As we discuss above, this is for us transfer our list-lick data into the template so that they can show on the web page. The process is to select each row in the data frame result and store them into a list.

2. Request

This is a tool that we use in **Probation_test.html & Probation_test_result.html,** we use **request.form** to get the input for our python program from the web page. As we click on “submit”, the program continue to the next step: using the input dates to do the probation test and show the result.

4. Trading in Simulated Market

4.1 Simulated Market Description

The simulated market is based on the latest 30 trading days of market data, for which the intraday trading data are available. Therefore, the market simulation will stop after 30 simulated trading days.

1. **Definition of trading day and market status**

1) Each trading day is 60 seconds, followed by 10 seconds of market close, and then a new trading day.

2) Market open time = 50 seconds, status: Open.

3) Market pending close time = 10 seconds, status: Pending Closing.

4) Market close time = 10 seconds, status: Market Closed.

2. **Daily Order Book - Historical Daily Market Data, created before market open**

1) Number of orders = (high_price - low_price)/price scale, price_scale is either 0.05 or 5 based stock price >= 1000 or not.

2) buy_price = open_price - price_scale * rand().

3) sell_price = open_price + price_scale * rand().

3) quantity = randomized, but sum of all the orders = daily volume.

4) The book interested are populated before market open.

3. **Intraday Market Interests - Daily Intraday Market Data, during market open**

1) Buy interests: Evey 5 min low price, 1/2 of the volume * rand().

2) Sell interests: Evey 5 min high price, 1/2 of the volume * rand().

3) Any crossed interests will be traded.

4) The order book is sorted accorinding Side, Symbol, Price and Quantity.

4. **Close Trade - either a buy or sell order is filled**

1) If buy side or sell side book is not empty, a best buy or best sell is filled.

2) If buy side or sell side is empty, the close trade for a buy or sell will be executed at daily market data closing price.

5. Execution logic

1) If market is closed, new orders will be rejected.

2) While market is open or in pending closing.

Market orders - always filled from best price.

Limit orders - will be filled at equal or better price.

A new limit order or a limit order with better price sweep books until it is filled or counter side of the book is empty.

3) Responses to new orders.

Order Fill

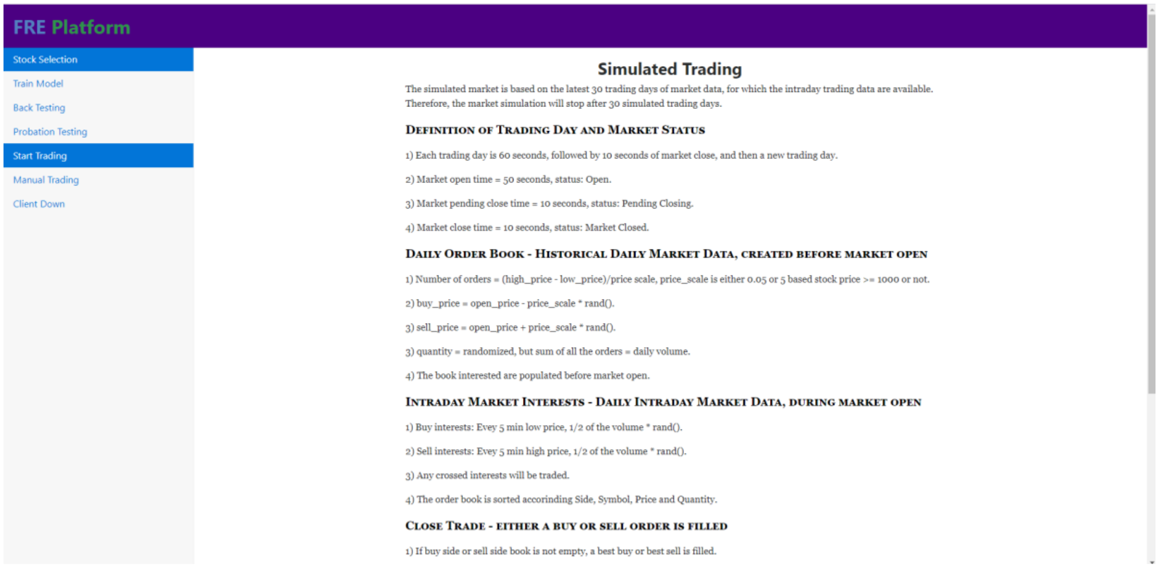
Order Partial Fill

Order Reject

4.2 Manual Trading

Here is the screenshot of old version dashboard, for showing manual trading part, which let us type in command in Client side. So that we can get more familiar to the network connection.

1. Start Trading(Old version):



Simulated Trading

The simulated market is based on the latest 30 trading days of market data, for which the intraday trading data are available. Therefore, the market simulation will stop after 30 simulated trading days.

DEFINITION OF TRADING DAY AND MARKET STATUS

1) Each trading day is 60 seconds, followed by 10 seconds of market close, and then a new trading day.

2) Market open time = 50 seconds, status: Open.

3) Market pending close time = 10 seconds, status: Pending Closing.

4) Market close time = 10 seconds, status: Market Closed.

DAILY ORDER BOOK - HISTORICAL DAILY MARKET DATA, CREATED BEFORE MARKET OPEN

1) Number of orders = (high_price - low_price)/price scale, price_scale is either 0.05 or 5 based stock price >= 1000 or not.

2) buy_price = open_price - price_scale * rand().

3) sell_price = open_price + price_scale * rand().

3) quantity = randomized, but sum of all the orders = daily volume.

4) The book interested are populated before market open.

INTRADAY MARKET INTERESTS - DAILY INTRADAY MARKET DATA, DURING MARKET OPEN

1) Buy interests: Evey 5 min low price, 1/2 of the volume * rand().

2) Sell interests: Evey 5 min high price, 1/2 of the volume * rand().

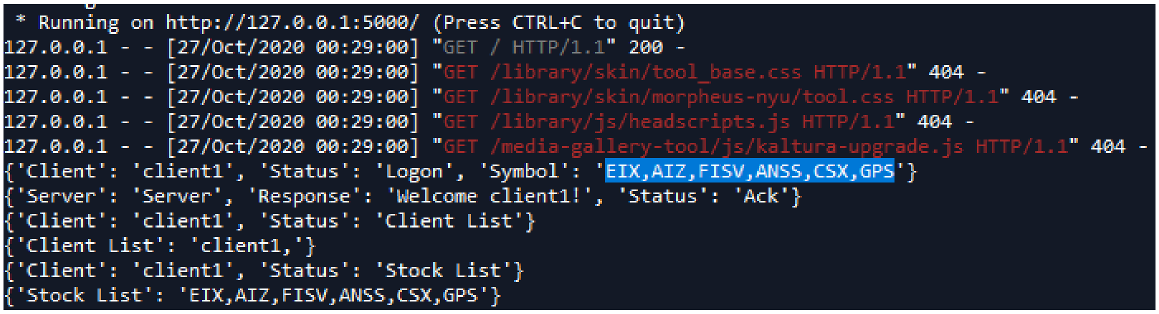
3) Any crossed interests will be traded.

4) The order book is sorted accorinding Side, Symbol, Price and Quantity.

CLOSE TRADE - EITHER A BUY OR SELL ORDER IS FILLED

1) If buy side or sell side book is not empty, a best buy or best sell is filled.

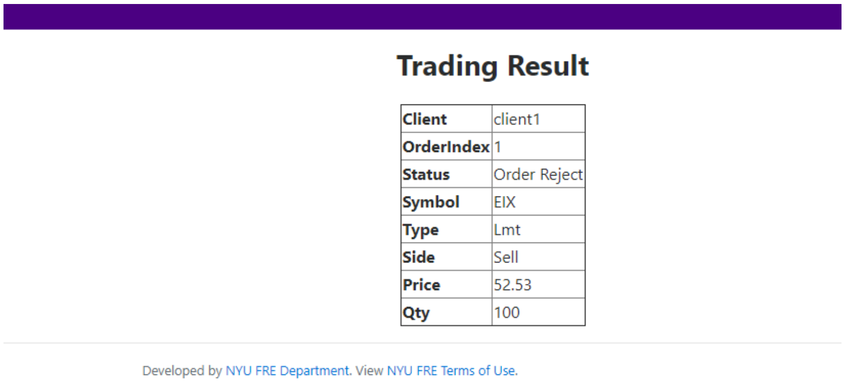
While press it, it will send out Logon(with selected stock), Client List, Stock List to the server side.



```
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [27/Oct/2020 00:29:00] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [27/Oct/2020 00:29:00] "GET /library/skin/tool_base.css HTTP/1.1" 404 -
127.0.0.1 - - [27/Oct/2020 00:29:00] "GET /library/skin/morpheus-nyu/tool.css HTTP/1.1" 404 -
127.0.0.1 - - [27/Oct/2020 00:29:00] "GET /library/js/headscripts.js HTTP/1.1" 404 -
127.0.0.1 - - [27/Oct/2020 00:29:00] "GET /media-gallery-tool/js/kaltura-upgrade.js HTTP/1.1" 404 -
{'Client': 'client1', 'Status': 'Logon', 'Symbol': 'EIX,AIZ,FISV,ANSS,CSX,GPS'}
{'Server': 'Server', 'Response': 'Welcome client1!', 'Status': 'Ack'}
{'Client': 'client1', 'Status': 'Client List'}
{'Client List': 'client1,'}
{'Client': 'client1', 'Status': 'Stock List'}
{'Stock List': 'EIX,AIZ,FISV,ANSS,CSX,GPS'}
```

2. Manual Trading

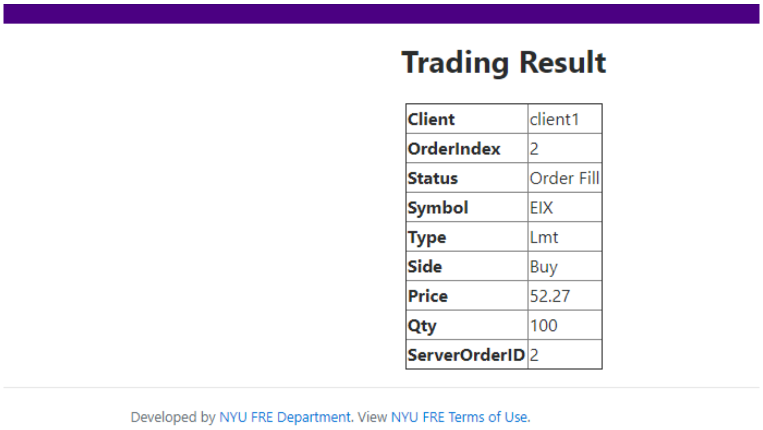
While doing manual trading, we may receive several different responses through the program. Screenshots for different kinds of response:



Trading Result

Client	client1
OrderIndex	1
Status	Order Reject
Symbol	EIX
Type	Lmt
Side	Sell
Price	52.53
Qty	100

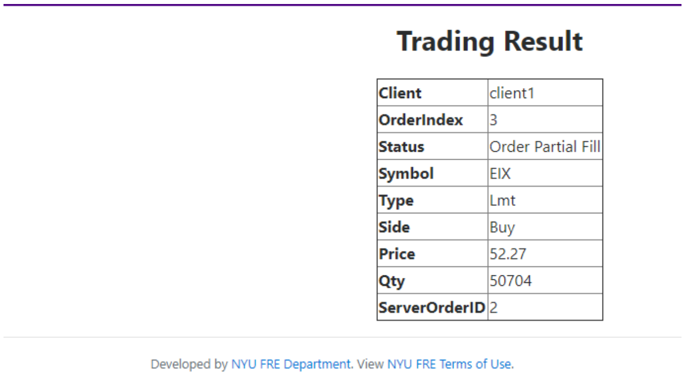
Developed by NYU FRE Department. View NYU FRE Terms of Use.



Trading Result

Client	client1
OrderIndex	2
Status	Order Fill
Symbol	EIX
Type	Lmt
Side	Buy
Price	52.27
Qty	100
ServerOrderID	2

Developed by NYU FRE Department. View NYU FRE Terms of Use.



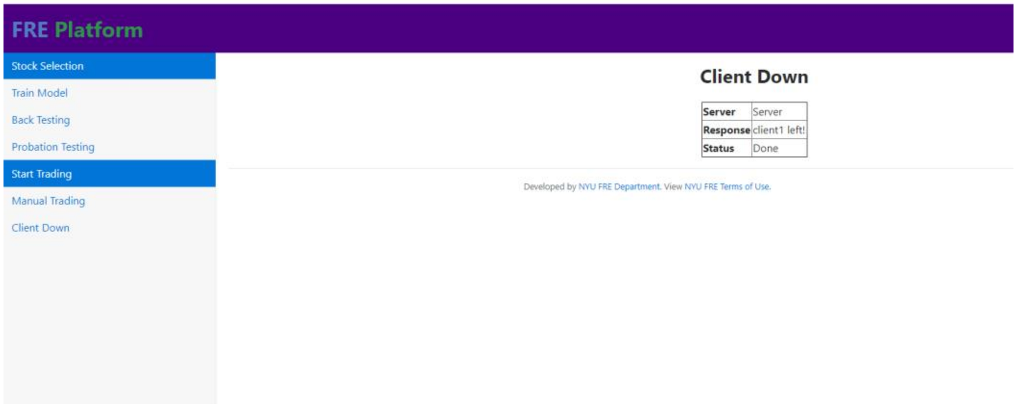
Trading Result

Client	client1
OrderIndex	3
Status	Order Partial Fill
Symbol	EIX
Type	Lmt
Side	Buy
Price	52.27
Qty	50704
ServerOrderID	2

Developed by NYU FRE Department. View NYU FRE Terms of Use.

3. Client Down

This part is the final step towards exiting the trading.



Client Down

Server	Server
Response	client1 left!
Status	Done

Developed by NYU FRE Department. View NYU FRE Terms of Use.

4.3 Auto Trading in Simulated Market

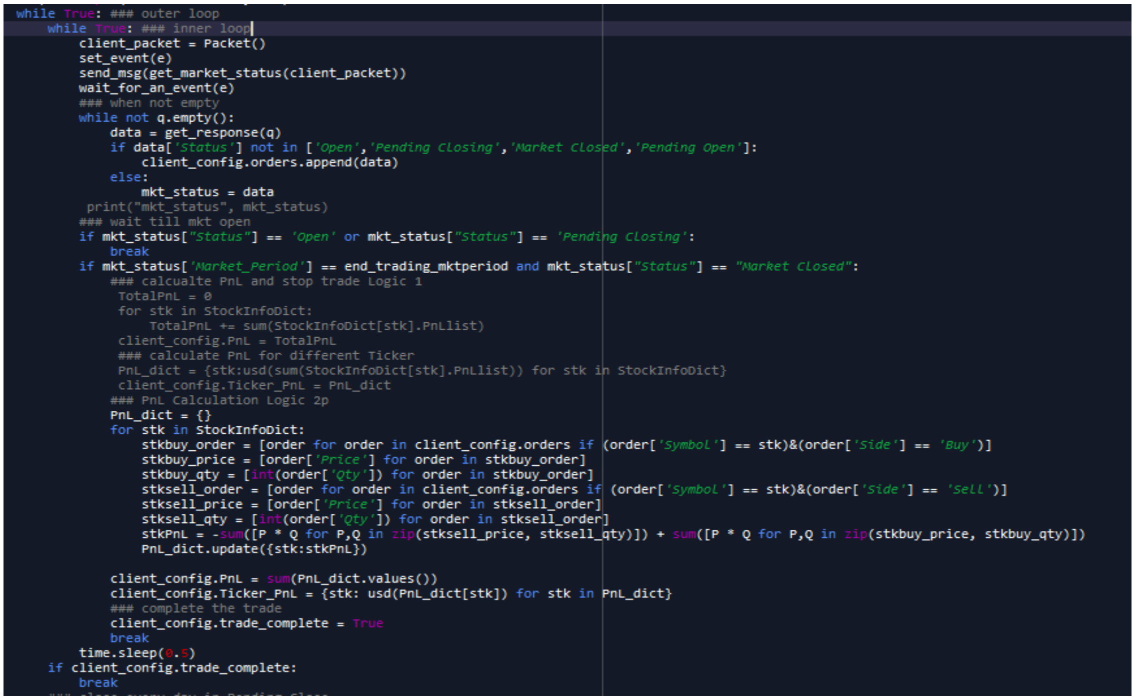
4.3.1 Important Logic to realize the BBD Trading Logic

1. Loops and Timer

There are two loops in the whole trading logic process, we set up a timer(by using time.sleep(1)) in outer loop, and also another timer (time.sleep(0.5)) in inner loop.

The inner loop is for checking the market status, so it will be sending request to check market status every 0.5 second to see whether it is time to trade. If market is open, then the inner loop break so we enter into outer loop again. If market is pending close, then we will close every open position.

The outer loop is the main BBD trading logic, in short, it will pull the order book every second(referring to every 5 sec in the real time) and use the data to calculate the moving average, std for each stock. If the trading signals appear, then we will send market order to the server. We will go through the important details below.

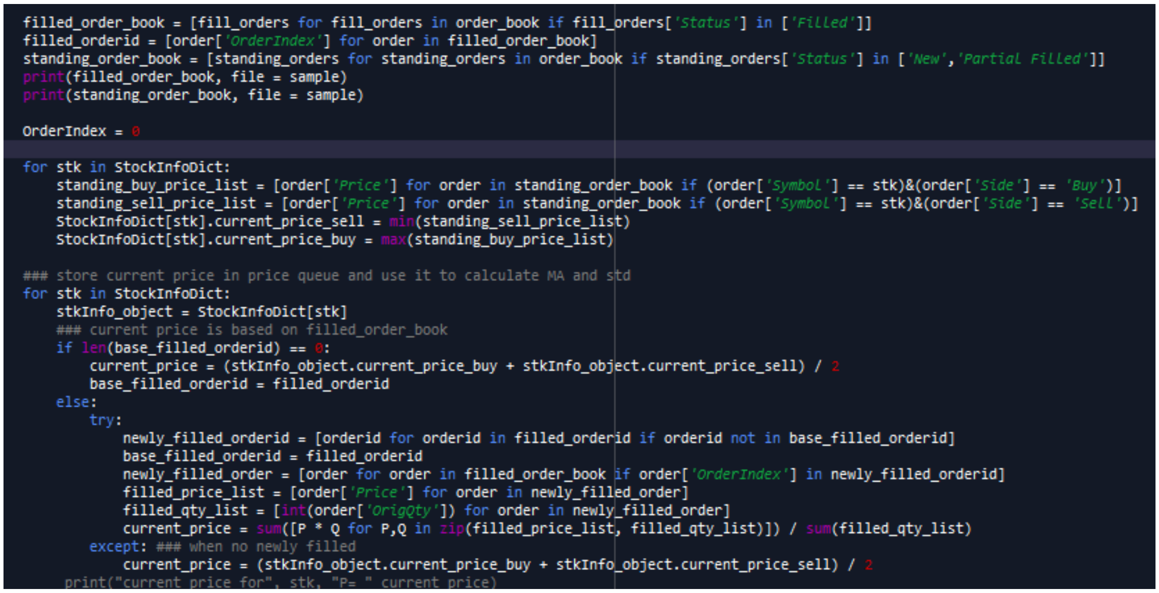


```
while True: ### outer loop
    while True: ### inner loop
        client_packet = Packet()
        set_event(e)
        send_msg(get_market_status(client_packet))
        wait_for_an_event(e)
        ### when not empty
        while not q.empty():
            data = get_response(q)
            if data['Status'] not in ['Open','Pending Closing','Market Closed','Pending Open']:
                client_config.orders.append(data)
            else:
                mkt_status = data
         print("mkt_status", mkt_status)
        ### wait till mkt open
        if mkt_status["Status"] == 'Open' or mkt_status["Status"] == 'Pending Closing':
            break
        if mkt_status['Market_Period'] == end_trading_mktperiod and mkt_status["Status"] == "Market Closed":
            ### calcualte PnL and stop trade Logic 1
             TotalPnL = 0
             for stk in StockInfoDict:
                 TotalPnL += sum(StockInfoDict[stk].PnLlist)
             client_config.PnL = TotalPnL
             ### calculate PnL for different Ticker
             PnL_dict = {stk:usd(sum(StockInfoDict[stk].PnLlist)) for stk in StockInfoDict}
             client_config.Ticker_PnL = PnL_dict
            ### PnL Calculation Logic 2p
            PnL_dict = {}
            for stk in StockInfoDict:
                stkbuy_order = [order for order in client_config.orders if (order['Symbol'] == stk)&(order['Side'] == 'Buy')]
                stkbuy_price = [order['Price'] for order in stkbuy_order]
                stkbuy_qty = [int(order['Qty']) for order in stkbuy_order]
                stksell_order = [order for order in client_config.orders if (order['Symbol'] == stk)&(order['Side'] == 'Sell')]
                stksell_price = [order['Price'] for order in stksell_order]
                stksell_qty = [int(order['Qty']) for order in stksell_order]
                stkPnL = -sum([P * Q for P,Q in zip(stksell_price, stksell_qty)]) + sum([P * Q for P,Q in zip(stkbuy_price, stkbuy_qty)])
                PnL_dict.update({stk:stkPnL})

            client_config.PnL = sum(PnL_dict.values())
            client_config.Ticker_PnL = {stk: usd(PnL_dict[stk]) for stk in PnL_dict}
            ### complete the trade
            client_config.trade_complete = True
            break
        time.sleep(0.5)
    if client_config.trade_complete:
        break
```

2. Standing orders and Newly filled orders

Here we separate the order book data by these two categories. Standing orders are for getting the best buy and sell price, while newly filled orders are used to calculating the **current price**, thus the moving average and std that our strategy need.

Where the newly filled orders are got by comparison of filled orders at different second.

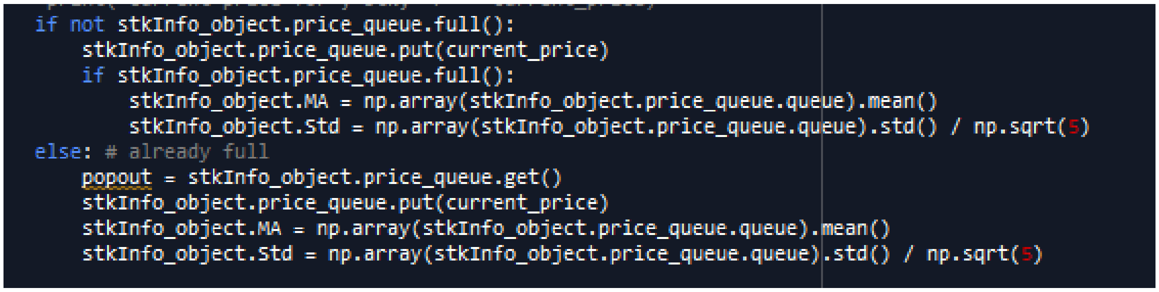


```
filled_order_book = [fill_orders for fill_orders in order_book if fill_orders['Status'] in ['Filled']]
filled_orderid = [order['OrderIndex'] for order in filled_order_book]
standing_order_book = [standing_orders for standing_orders in order_book if standing_orders['Status'] in ['New','Partial Filled']]
print(filled_order_book, file = sample)
print(standing_order_book, file = sample)

OrderIndex = 0

for stk in StockInfoDict:
    standing_buy_price_list = [order['Price'] for order in standing_order_book if (order['Symbol'] == stk)&(order['Side'] == 'Buy')]
    standing_sell_price_list = [order['Price'] for order in standing_order_book if (order['Symbol'] == stk)&(order['Side'] == 'Sell')]
    StockInfoDict[stk].current_price_sell = min(standing_sell_price_list)
    StockInfoDict[stk].current_price_buy = max(standing_buy_price_list)

### store current price in price queue and use it to calculate MA and std
for stk in StockInfoDict:
    stkInfo_object = StockInfoDict[stk]
    ### current price is based on filled_order_book
    if len(base_filled_orderid) == 0:
        current_price = (stkInfo_object.current_price_buy + stkInfo_object.current_price_sell) / 2
        base_filled_orderid = filled_orderid
    else:
        try:
            newly_filled_orderid = [orderid for orderid in filled_orderid if orderid not in base_filled_orderid]
            base_filled_orderid = filled_orderid
            newly_filled_order = [order for order in filled_order_book if order['OrderIndex'] in newly_filled_orderid]
            filled_price_list = [order['Price'] for order in newly_filled_order]
            filled_qty_list = [int(order['OrigQty']) for order in newly_filled_order]
            current_price = sum([P * Q for P,Q in zip(filled_price_list, filled_qty_list)]) / sum(filled_qty_list)
        except: ### when no newly filled
            current_price = (stkInfo_object.current_price_buy + stkInfo_object.current_price_sell) / 2
     print("current price for", stk, "P= " current_price)
```

The current price is thus calculated by the Qty-weighted filled order price.

3. MA and Std calculation

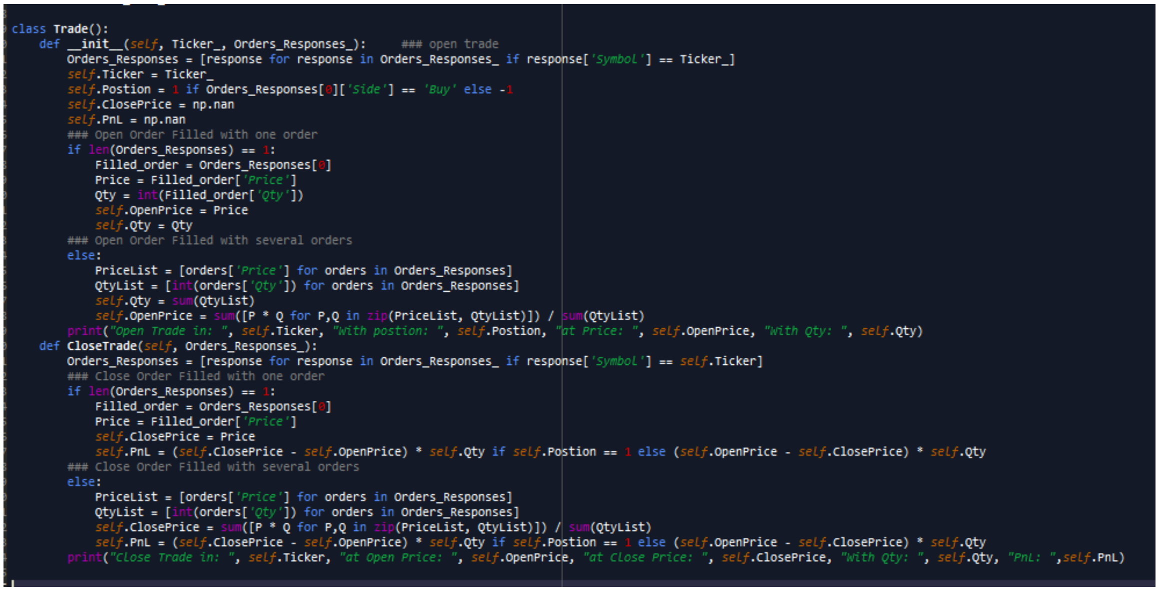
After we get the current price, we need to calculate these two parameters for our BBD strategy. The logic is we put the current price to a queue for each stock. The queue's max size is decided by the window parameter "H" for each stock (we trained and back test it early on). When the queue is full, we calculated its average and std. And when new "current price" need to come in, we pop out the oldest price, and re-do the calculation so that to realize the "rolling" sense.



```
    if not stkInfo_object.price_queue.full():
        stkInfo_object.price_queue.put(current_price)
        if stkInfo_object.price_queue.full():
            stkInfo_object.MA = np.array(stkInfo_object.price_queue.queue).mean()
            stkInfo_object.Std = np.array(stkInfo_object.price_queue.queue).std() / np.sqrt(5)
    else: # already full
        popout = stkInfo_object.price_queue.get()
        stkInfo_object.price_queue.put(current_price)
        stkInfo_object.MA = np.array(stkInfo_object.price_queue.queue).mean()
        stkInfo_object.Std = np.array(stkInfo_object.price_queue.queue).std() / np.sqrt(5)
```

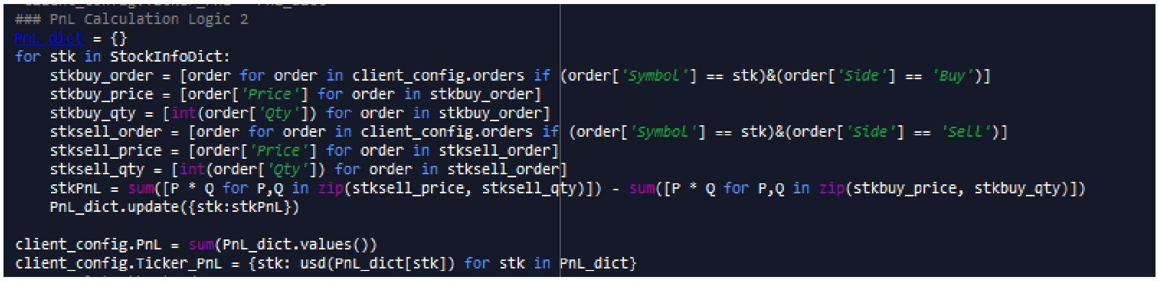
4. PnL calculation

Method1: Actually, in the code side we have construct a class called Trade, which is for record the “open” and “close” in pair so that we can calculate each trade’s PnL. However, due to the fact that although we always get_response from trading queue until it is empty, the response can still be mixed up. Which will cause mis-match of the trading orders, so I decide to use another method.



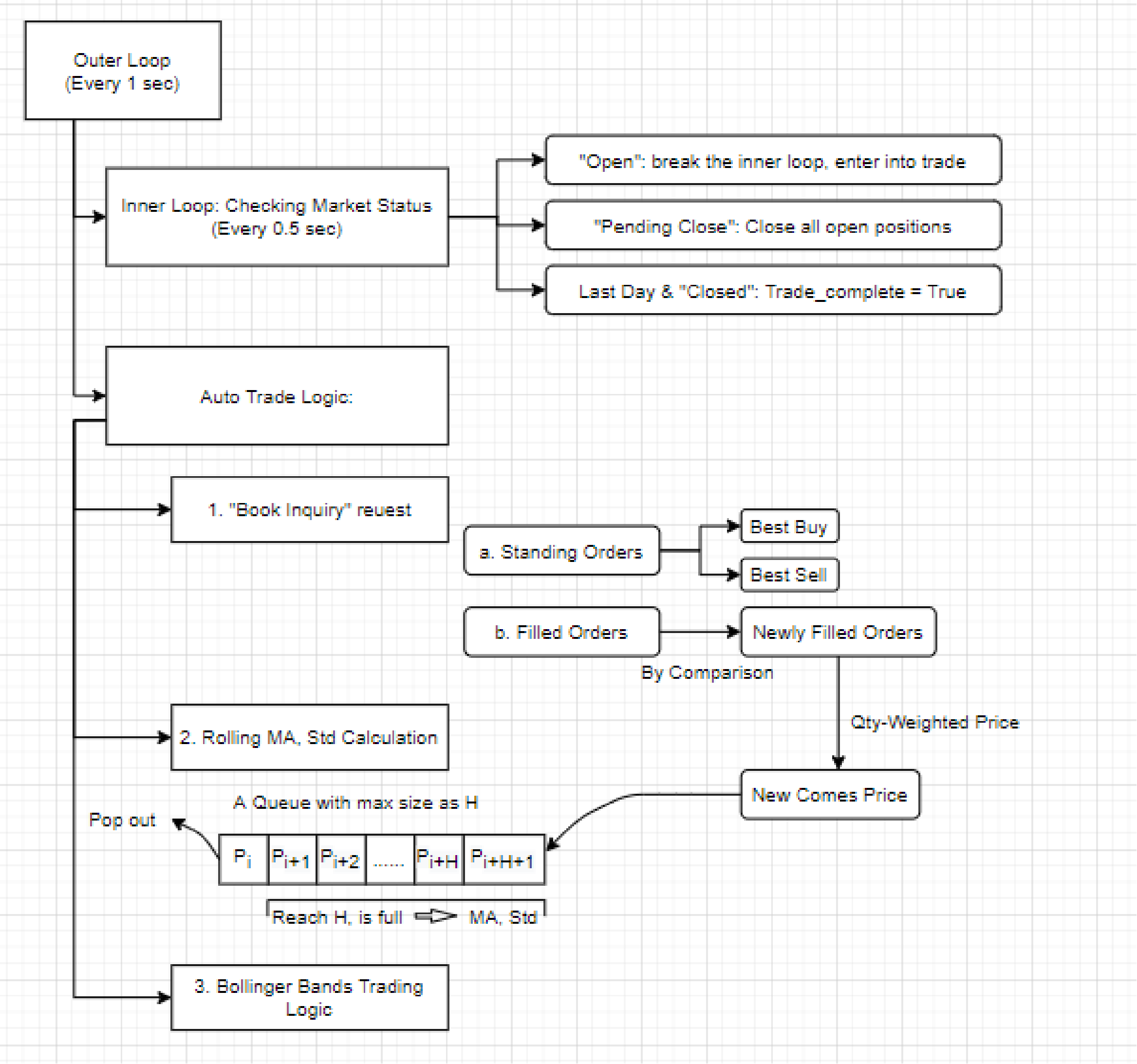
```
class Trade():
    def __init__(self, Ticker_, Orders_Responses_):     ### open trade
        Orders_Responses = [response for response in Orders_Responses_ if response['Symbol'] == Ticker_]
        self.Ticker = Ticker_
        self.Postion = 1 if Orders_Responses[0]['Side'] == 'Buy' else -1
        self.ClosePrice = np.nan
        self.PnL = np.nan
        ### Open Order Filled with one order
        if len(Orders_Responses) == 1:
            Filled_order = Orders_Responses[0]
            Price = Filled_order['Price']
            Qty = int(Filled_order['Qty'])
            self.OpenPrice = Price
            self.Qty = Qty
        ### Open Order Filled with several orders
        else:
            PriceList = [orders['Price'] for orders in Orders_Responses]
            QtyList = [int(orders['Qty']) for orders in Orders_Responses]
            self.Qty = sum(QtyList)
            self.OpenPrice = sum([P * Q for P,Q in zip(PriceList, QtyList)]) / sum(QtyList)
        print("Open Trade in: ", self.Ticker, "With postion: ", self.Postion, "at Price: ", self.OpenPrice, "With Qty: ", self.Qty)
    def CloseTrade(self, Orders_Responses_):
        Orders_Responses = [response for response in Orders_Responses_ if response['Symbol'] == self.Ticker]
        ### Close Order Filled with one order
        if len(Orders_Responses) == 1:
            Filled_order = Orders_Responses[0]
            Price = Filled_order['Price']
            self.ClosePrice = Price
            self.PnL = (self.ClosePrice - self.OpenPrice) * self.Qty if self.Postion == 1 else (self.OpenPrice - self.ClosePrice) * self.Qty
        ### Close Order Filled with several orders
        else:
            PriceList = [orders['Price'] for orders in Orders_Responses]
            QtyList = [int(orders['Qty']) for orders in Orders_Responses]
            self.ClosePrice = sum([P * Q for P,Q in zip(PriceList, QtyList)]) / sum(QtyList)
            self.PnL = (self.ClosePrice - self.OpenPrice) * self.Qty if self.Postion == 1 else (self.OpenPrice - self.ClosePrice) * self.Qty
        print("Close Trade in: ", self.Ticker, "at Open Price: ", self.OpenPrice, "at Close Price: ", self.ClosePrice, "With Qty: ", self.Qty, "PnL: ",self.PnL)
```

Method2 is based on all the orders, use the total money receive SUM(sell qty_i * sell price_i), minus the total payment SUM(buy qty_i * buy price_i). It is much easier than the method1, but can only get the PnL number, cannot analysis further based on each trade level.

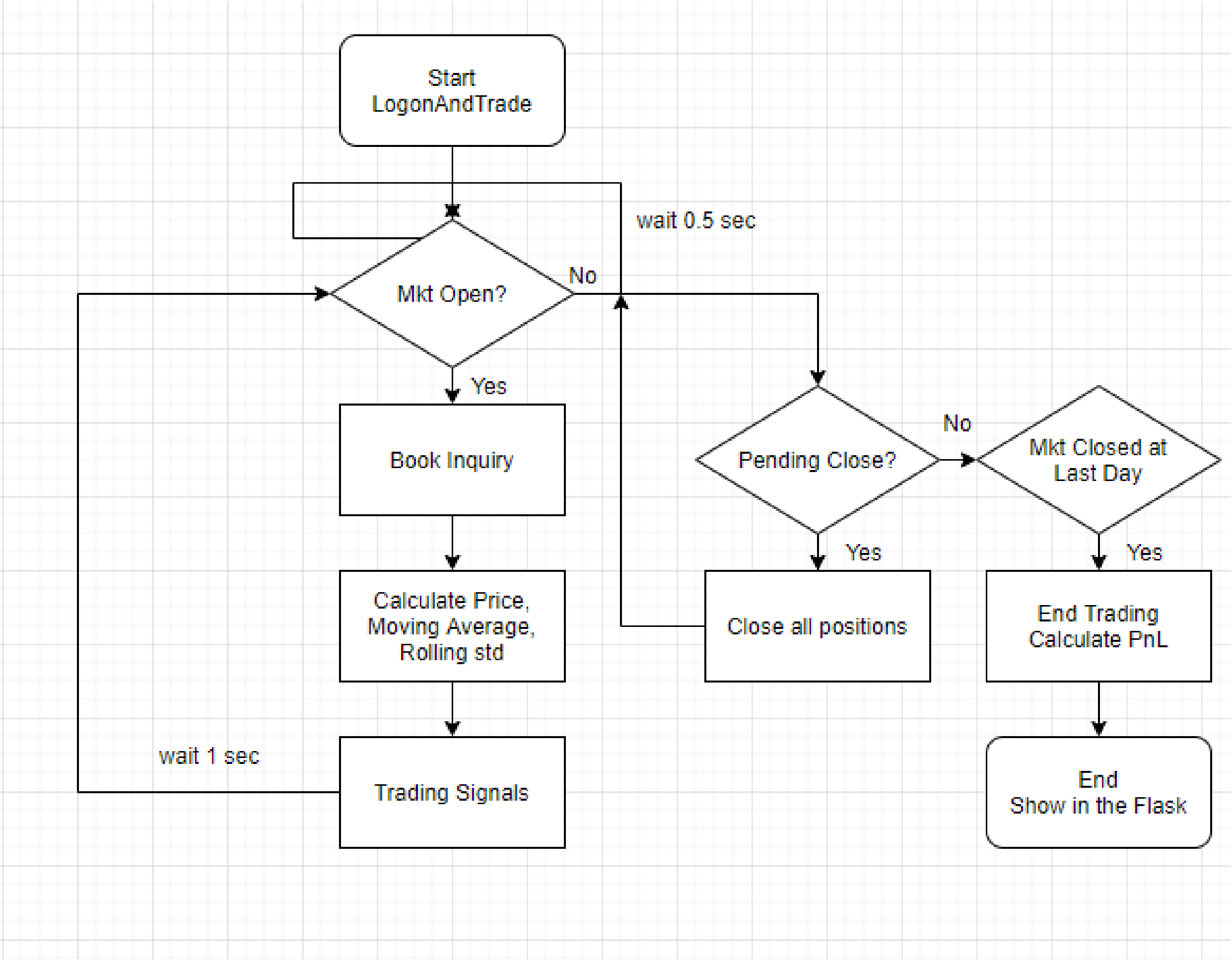


```
### PnL Calculation Logic 2
PnL_dict = {}
for stk in StockInfoDict:
    stkbuy_order = [order for order in client_config.orders if (order['Symbol'] == stk)&(order['Side'] == 'Buy')]
    stkbuy_price = [order['Price'] for order in stkbuy_order]
    stkbuy_qty = [int(order['Qty']) for order in stkbuy_order]
    stksell_order = [order for order in client_config.orders if (order['Symbol'] == stk)&(order['Side'] == 'Sell')]
    stksell_price = [order['Price'] for order in stksell_order]
    stksell_qty = [int(order['Qty']) for order in stksell_order]
    stkPnL = sum([P * Q for P,Q in zip(stksell_price, stksell_qty)]) - sum([P * Q for P,Q in zip(stkbuy_price, stkbuy_qty)])
    PnL_dict.update({stk:stkPnL})

client_config.PnL = sum(PnL_dict.values())
client_config.Ticker_PnL = {stk: usd(PnL_dict[stk]) for stk in PnL_dict}
```

4.3.2 Process description:



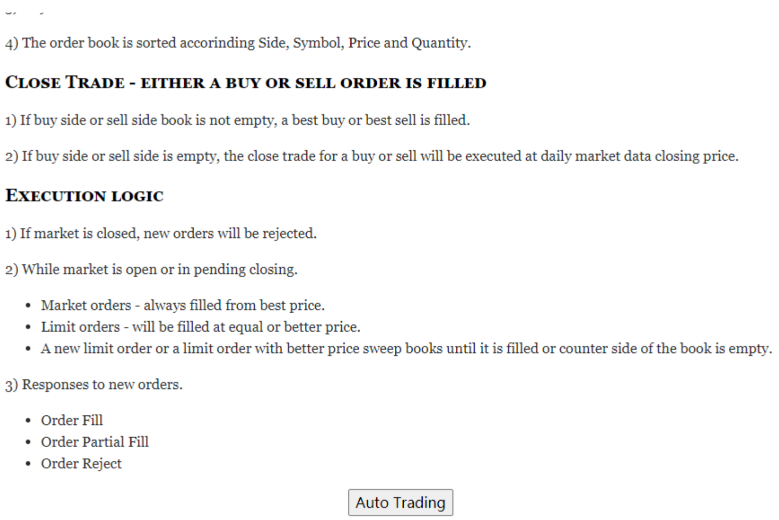
4.3.3 Flow Chart



4.4 Auto Trading Screenshot and Result

Screenshots for auto trading results:

From Stat Trading Page (Where we introduce the simulated market), press the button “Auto Trading”, then the auto trading logic begin. (Make sure that the server side is running.)



4) The order book is sorted accorinding Side, Symbol, Price and Quantity.

Close Trade - either a buy or sell order is filled

1) If buy side or sell side book is not empty, a best buy or best sell is filled.

2) If buy side or sell side is empty, the close trade for a buy or sell will be executed at daily market data closing price.

Execution logic

1) If market is closed, new orders will be rejected.

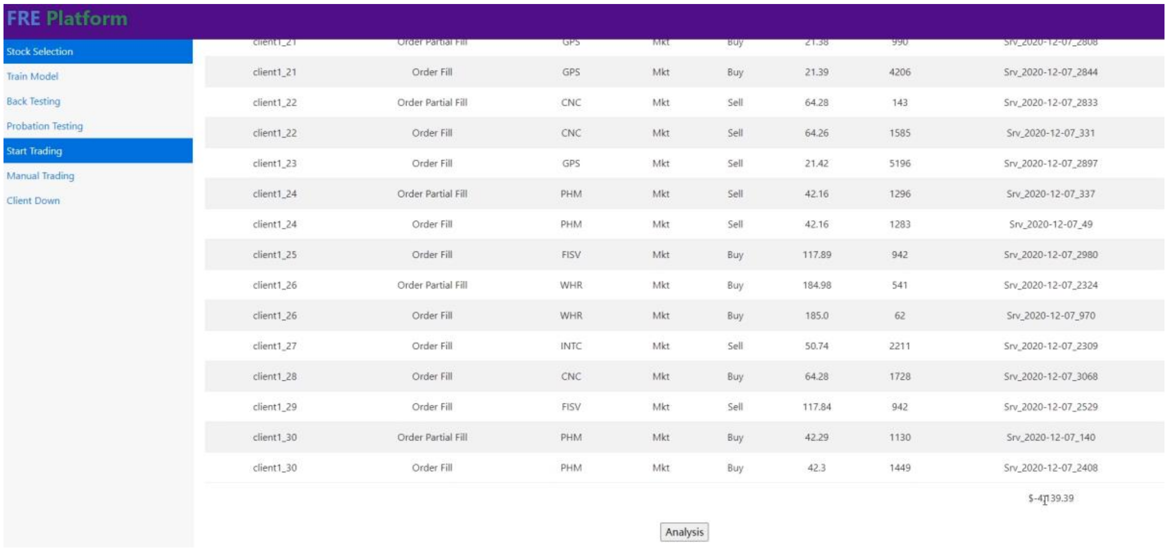
2) While market is open or in pending closing.

- Market orders - always filled from best price.
- Limit orders - will be filled at equal or better price.
- A new limit order or a limit order with better price sweep books until it is filled or counter side of the book is empty.

3) Responses to new orders.

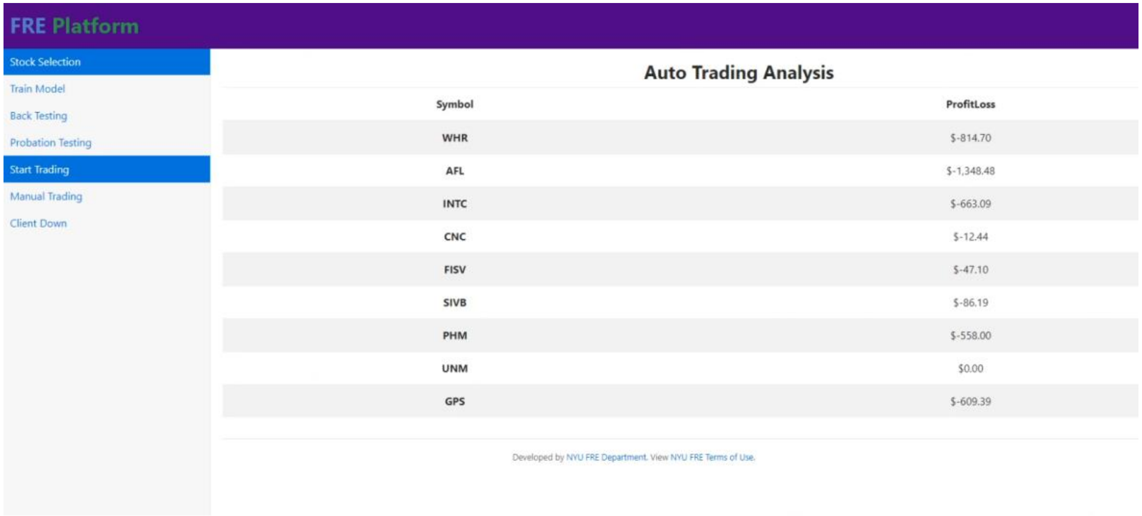
- Order Fill
- Order Partial Fill
- Order Reject

When the simulated market ends, the auto trading process will stop and return the results as below, basically the orders information with Total PnL at the bottom of the Page.



client1_21	Order Partial Fill	GPS	Mkt	Buy	21.38	990	Srv_2020-12-07_2808
client1_21	Order Fill	GPS	Mkt	Buy	21.39	4206	Srv_2020-12-07_2844
client1_22	Order Partial Fill	CNC	Mkt	Sell	64.28	143	Srv_2020-12-07_2833
client1_22	Order Fill	CNC	Mkt	Sell	64.26	1585	Srv_2020-12-07_331
client1_23	Order Fill	GPS	Mkt	Sell	21.42	5196	Srv_2020-12-07_2897
client1_24	Order Partial Fill	PHM	Mkt	Sell	42.16	1296	Srv_2020-12-07_337
client1_24	Order Fill	PHM	Mkt	Sell	42.16	1283	Srv_2020-12-07_49
client1_25	Order Fill	FISV	Mkt	Buy	117.89	942	Srv_2020-12-07_2980
client1_26	Order Partial Fill	WHR	Mkt	Buy	184.98	541	Srv_2020-12-07_2324
client1_26	Order Fill	WHR	Mkt	Buy	185.0	62	Srv_2020-12-07_970
client1_27	Order Fill	INTC	Mkt	Sell	50.74	2211	Srv_2020-12-07_2309
client1_28	Order Fill	CNC	Mkt	Buy	64.28	1728	Srv_2020-12-07_3068
client1_29	Order Fill	FISV	Mkt	Sell	117.84	942	Srv_2020-12-07_2529
client1_30	Order Partial Fill	PHM	Mkt	Buy	42.29	1130	Srv_2020-12-07_140
client1_30	Order Fill	PHM	Mkt	Buy	42.3	1449	Srv_2020-12-07_2408
							$-4139.39

Since we may want to check each stock’s performance, so we add another button named “Analysis”, while we are pressing it may lead us to the page showing that the PnL related to each individual stock.



Auto Trading Analysis

Symbol	ProfitLoss
WHR	$-814.70
AFL	$-1,348.48
INTC	$-663.09
CNC	$-12.44
FISV	$-47.10
SIVB	$-86.19
PHM	$-558.00
UNM	$0.00
GPS	$-609.39

Developed by NYU FRE Department. View NYU FRE Terms of Use.

5. Future Development

After finishing the final Auto trading part, we may re-examine the whole project, and from my perspective, there are several points that I think can be further developed and improve:

1. Parameters Training

The PnL above in the screenshot is good for the recent week, however, if we need to further prove it is profitable, we need to run it on other period. That is because as we mentioned, the profitability of the model is quite time-varying and the parameters we train are quite short-memory.

For instance, we can run in on the recent month, train the params on a rolling basis, and test the model on 4 single weeks following the training period. To see whether it is still profitable on those weeks.

2. Auto Trading Analysis Page

The current Analysis page only separate the PnL to each stock. For further development. Later on, we may calculate some performance statistics and also let them show on this page. Those statistics should be calculated in trade level. For instance, Sharpe ratio, Information ratio, Win ratio, etc. We can even plug in PnL graphs into the page, can also show buy and sell points.

3. Auto Trading Logic

Since we use timer in our program, and often the calculation can take some time since we actually do a lot of stuff in the calculation process, sometimes when we cannot enter the right price that we want to trade. In this sense, we may find way to make our calculation process faster. Or we should make the time interval to be longer, in order to make sure our process finished before the update of order book.

References

1. John Bollinger, *Bollinger On Bollinger Bands*

2. Miguel Grinberg, *Flask Web Development: Developing Web Applications with Python*

3. Richard Stevens, *UNIX networking programming*

4. Anthony Williams, *C++ Concurrency in Action*

Appendix A – Source Code

1. BBDModel.py

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 12 20:11:29 2020

@author: 63438
"""

import json
import datetime as dt
from dateutil import tz
from dateutil.relativedelta import relativedelta
import urllib.request
import pandas as pd
import numpy as np

from sqlalchemy import Column, ForeignKey, Integer, Float, String
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import inspect



engine = create_engine('sqlite:///TradingBBD.db',connect_args={'check_same_thread': False})
conn = engine.connect()
conn.execute("PRAGMA foreign_keys = ON")


metadata = MetaData()
metadata.reflect(bind=engine)


end_date = dt.datetime.today().date()
start_date = end_date - relativedelta(weeks=1)


def EDTtoUnixTime(EDTdatetime):
    utcTime = EDTdatetime.replace(tzinfo = tz.gettz('EDT')).astimezone(tz=dt.timezone.utc)
    unixTime = utcTime.timestamp()
    return str(int(unixTime))

def get_sp500_component():
    url = 'https://eodhistoricaldata.com/api/fundamentals/GSPC.INDX?api_token=5ba84ea974ab42.45160048'
    with urllib.request.urlopen(url) as req:
        data = json.load(req)
        comdf = pd.DataFrame(data['Components']).T
        tickerList = list(comdf['Code'])
    return tickerList[:30] ### for demo, otherwise it is time consuming


def get_daily_data(symbol,
                   start=start_date,
                   end=end_date,
                   requestURL='https://eodhistoricaldata.com/api/eod/',
                   apiKey='5ba84ea974ab42.45160048'):
    symbolURL = str(symbol) + '.US?'
    startURL = 'from=' + str(start)
    endURL = 'to=' + str(end)
    apiKeyURL = 'api_token=' + apiKey
    completeURL = requestURL + symbolURL + startURL + '&' + endURL + '&' + apiKeyURL + '&period=d&fmt=json'
    print('Possessing daily data of stock:' + symbol)
    with urllib.request.urlopen(completeURL) as req:
        data = json.load(req)
        return data

def get_intraday_data(symbol,
                      interval='5m',
                      requestURL='https://eodhistoricaldata.com/api/intraday/',
                      apiKey='5ba84ea974ab42.45160048',
                      start=None): ### start in the format of EDT datetime
    symbolURL = str(symbol) + '.US?'
    apiKeyURL = 'api_token=' + apiKey
    if start != None:
        starttimestamp = EDTtoUnixTime(start)
        completeURL = requestURL + symbolURL + apiKeyURL + '&interval=' + interval + '&fmt=json' + '&from='+starttimestamp
    else:
        completeURL = requestURL + symbolURL + apiKeyURL + '&interval=' + interval + '&fmt=json'
    print('Possessing data of stock:' + symbol)
    with urllib.request.urlopen(completeURL) as req:
        data = json.load(req)
        return data

def create_stocks_table(table_name, metadata, engine):
    table = Table(table_name, metadata,
                  Column('Ticker', String(50), primary_key=True, nullable=False),
                  Column('H', Integer, nullable=False),
                  Column('K1', Float, nullable=False),
                  Column('Notional', Float, nullable=False),
                  Column('Profit_Loss_in_Training', Float, nullable=False),
                  Column('Return_in_Training', Float, nullable=False),
                  Column('Profit_Loss', Float, nullable=False),
                  Column('Return', Float, nullable=False),
                  extend_existing=True)
    table.create(engine)


def create_price_table(table_name, metadata, engine):
    tables = metadata.tables.keys()
    if table_name not in tables:
               foreign_key = 'Stocks.Ticker'
               table = Table(table_name, metadata,
                             Column('Symbol', String(50), ForeignKey(foreign_key), primary_key=True, nullable=False),
                             Column('DateTime', String(50), primary_key=True, nullable=False),
                             Column('Open', Float, nullable=False),
                             Column('High', Float, nullable=False),
                             Column('Low', Float, nullable=False),
                          Column('Close', Float, nullable=False),
                          Column('Volume', Integer, nullable=False))
            table.create(engine)

    def create_strategy_table(table_name, metadata, engine):
        table = Table(table_name, metadata,
                      Column('Strategy', String(50), primary_key=True, nullable=False),
                      Column('Notional', Float, nullable=False),
                      Column('PnL_in_Training', Float, nullable=False),
                      Column('Return_in_Training', Float, nullable=False),
                      Column('SPY_Return_in_Training', Float, nullable=False),
                      Column('PnL', Float, nullable=False),
                      Column('Strategy_Return', Float, nullable=False),
                      Column('SPY_Return', Float, nullable=False),
                      extend_existing=True)
        table.create(engine)

    def clear_a_table(table_name, metadata, engine):
        conn = engine.connect()
        table = metadata.tables[table_name]
        delete_st = table.delete()
        conn.execute(delete_st)


    def execute_sql_statement(sql_stmt, engine, change=False):
        if change:
            engine.execute(sql_stmt)
        else:
            result_set = engine.execute(sql_stmt)
            result_df = pd.DataFrame(result_set.fetchall())
            if not result_df.empty:
                result_df.columns = result_set.keys()
            return result_df

    def stock_selection(stock_list, interval='5m', number_of_stocks=10):
        std_resultdf = pd.DataFrame(index=stock_list)
        std_resultdf['std'] = 0.0
        for stk in stock_list:
            try:
                stk_data = pd.DataFrame(get_intraday_data(stk, interval))
                std = stk_data.close.pct_change().shift(-1).std()
                std_resultdf.loc[stk,'std'] = std
                print('Volatility of return over stock: ' + stk + ' is: ' + str(std))
            except:
                print('Cannot get data of Stock:' + stk)
        stock_selected = list(std_resultdf['std'].sort_values().index[-number_of_stocks:])
        selected_df = std_resultdf['std'].sort_values()[-number_of_stocks:]
        return stock_selected, selected_df


    def populate_price_data(stockList, engine, table_name, start_date, end_date, interval='1m'):
        column_names = ['Symbol', 'DateTime', 'Open', 'High', 'Low', 'Close', 'Volume']
        price_data = []
        mkt_opentime = dt.datetime.strptime('09:30','%H:%M').time()
        mkt_closetime = dt.datetime.strptime('16:00','%H:%M').time()
        start_datetime = dt.datetime(start_date.year,start_date.month,start_date.day,9,30)
        for stk in stockList:
            raw_data = get_intraday_data(stk,interval,start=start_datetime)
            for stock_data in raw_data:
                price_data.append([stk, stock_data['datetime'], stock_data['open'], stock_data['high'], stock_data['low'], \
                                   stock_data['close'], stock_data['volume']])
            print(stk + '\'s data has been stored.')

        price = pd.DataFrame(price_data, columns=column_names)
        ### Convert UTC to EST
        price.DateTime = pd.to_datetime(price.DateTime) - dt.timedelta(hours=4)
        ### Select during Trading Hour and within selected period
        price = price[(price.DateTime.dt.time>=mkt_opentime) & (price.DateTime.dt.time<=mkt_closetime)]
        price = price[(price.DateTime.dt.date>=start_date) & (price.DateTime.dt.date<=end_date)]
        price.to_sql(table_name, con=engine, if_exists='append', index=False)

    def GridSearchinDBBD(stkdata,H,K1):
        data = stkdata.copy()
        Notional = 1000000.00 / 10
        data['SMA'] = data['close'].rolling(H).mean()
        data['rstd'] = data['close'].rolling(H).std()
        data['Up1'] = data['SMA'] + K1 * data['rstd']
        data['Down1'] = data['SMA'] - K1 * data['rstd']

        ### signals
        data['signal'] = 0
        data.loc[data['close'] >= data['Up1'],'signal'] = -1
        data.loc[(data['close'] < data['Up1']) & (data['close'] > data['Down1']),'signal'] = 0
        data.loc[data['close'] <= data['Down1'],'signal'] = 1
        data.signal = data.signal.shift().fillna(0)
        data['trade'] = data.signal.diff()

        ### PnL cal
        data['pre_trade_pos'] = 0
        data['target_pos'] = 0
        data['realized_pnl_d'] = 0
        last_target_pos = 0
        for index, row in data.iterrows():
            data.loc[index,'pre_trade_pos'] = last_target_pos
            if row.trade != 0:
                data.loc[index,'target_pos'] = row.signal * int(Notional / row['close'])
                last_target_pos = row.signal * int(Notional / row['close'])

            if abs(row.signal) < abs(row.trade):
                if row.trade < 0:
                    data.loc[index,'realized_pnl_d'] = data.loc[index,'pre_trade_pos'] * row['close'] - Notional
                else:
```

```
            data.loc[index,'realized_pnl_d'] = Notional + data.loc[index,'pre_trade_pos'] * row['close']

      data['realized_pnl_p'] = data['realized_pnl_d'] / Notional
      data['cum_pnl_p'] = data['realized_pnl_p'].cumsum() + 1
      data['cum_pnl_p_diff'] = data['cum_pnl_p'].diff()

      ### IR calculation
      data.index = data.datetime
      daily_return = pd.DataFrame(data.realized_pnl_p).copy()
      daily_return['date'] = daily_return.index
      daily_return['date'] = daily_return['date'].apply(lambda x:x.date())
      df_whole = daily_return.groupby(['date']).sum()
      # performance strat
      cumulative_return = df_whole.cumsum()
      Annualized_return = cumulative_return.iloc[-1,:]* 252 / len(df_whole)
      Annualized_vol = df_whole.std() * np.sqrt(252)
      Information_ratio = (Annualized_return / Annualized_vol)[0]
      CumPnL = cumulative_return.iloc[-1,:][0] * Notional
      return Information_ratio, CumPnL



    def train_params_DBBD(stk_list,H_list,K1_list,train_end_date,period='1M'):
      if period[1] == 'M':
        train_start_date = train_end_date - relativedelta(months=int(period[0]))
      elif period[1] == 'W':
        train_start_date = train_end_date - relativedelta(weeks=int(period[0]))
      train_start = dt.datetime(train_start_date.year,train_start_date.month,train_start_date.day,9,30)
      mkt_opentime = dt.datetime.strptime('09:30','%H:%M').time()
      mkt_closetime = dt.datetime.strptime('16:00','%H:%M').time()
      stocks = pd.DataFrame(stk_list,columns=['Ticker'])
      stocks["H"] = 0
      stocks["K1"] = 0.0
      stocks['Notional'] = 1000000.00 / 10
      stocks["Profit_Loss_in_Training"] = 0.0
      stocks['Return_in_Training'] = 0.0
      stocks["Profit_Loss"] = 0.0
      stocks['Return'] = 0.0
      for stk in stk_list:
        print("Training params for: " + stk +' ...')
        train_data = pd.DataFrame(get_intraday_data(stk, interval='1m',start=train_start))
        ### Convert UTC to EST
        train_data.datetime = pd.to_datetime(train_data.datetime) - dt.timedelta(hours=4)
        ### Select during Trading Hour and within selected period
        train_data = train_data[(train_data.datetime.dt.time>=mkt_opentime) & (train_data.datetime.dt.time<=mkt_closetime)]
        train_data = train_data[(train_data.datetime.dt.date>=train_start_date) & (train_data.datetime.dt.date<train_end_date)]
        IR_df = pd.DataFrame(index=H_list,columns=K1_list)
        CumPnLdf = pd.DataFrame(index=H_list,columns=K1_list)
        try:
          for H in H_list:
```

```
                    for K1 in K1_list:
                        IR, CumPnL = GridSearchinDBBD(train_data,H,K1)
                        IR_df.loc[H,K1] = IR
                        CumPnLdf.loc[H,K1] = CumPnL
                        print(stk + ':H,K pair:(' + str(H) + ',' + str(K1) + ')done, with CumPnL:' + str(CumPnL))
                ### select the pair from IR
                H0 = CumPnLdf.mean(axis=1).idxmax()
                K10 = CumPnLdf.mean().idxmax()
                ### delete those with negative PnL within training period
                if CumPnLdf.loc[H0,K10] <= 0:
                    print('Training performance bad, delete stk:{}.'.format(stk))
                    stocks = stocks.drop(stocks[stocks.Ticker==stk].index)
                else:
                    stocks.loc[stocks[stocks.Ticker==stk].index,'H'] = H0
                    stocks.loc[stocks[stocks.Ticker==stk].index,'K1'] = K10
                    stocks.loc[stocks[stocks.Ticker==stk].index,'Profit_Loss_in_Training'] = CumPnLdf.loc[H0,K10]
                    stocks.loc[stocks[stocks.Ticker==stk].index,'Return_in_Training'] = CumPnLdf.loc[H0,K10] * 10 / 1000000.00
            except:
                print("Deleted. Missing data for stk: " + stk)
                stocks = stocks.drop(stocks[stocks.Ticker==stk].index)
        return stocks

    def stk_select():
        sp500_symbol_list = get_sp500_component()
        selected_stk, stk_df = stock_selection(sp500_symbol_list)
        return selected_stk, stk_df

    def build_trading_model(stk_list):
        engine.execute('Drop Table if exists Stocks;')
        create_stocks_table('Stocks', metadata, engine)
        H_list = [40,50,60,70,80,90]
        K1_list = [1.5,1.8,2.0,2.2,2.5]
        stocks = train_params_DBBD(stk_list,H_list,K1_list,train_end_date=start_date,period='1W')

        stocks.to_sql('Stocks', con=engine, if_exists='append', index=False)

        return stocks

    def populate_backtest_data(stocks):
        ### CREATE price table
        create_price_table('Price', metadata, engine)
        inspector = inspect(engine)
        print(inspector.get_table_names())
        clear_a_table('Price', metadata, engine)
        ### populate price table
        populate_price_data(stocks['Ticker'].unique(), engine, 'Price', start_date, end_date)

        ### create strategy table
        engine.execute('Drop Table if exists Strategy;')
        create_strategy_table('Strategy', metadata, engine)

    def backTest():
        ### read data through data base
        stockdf = execute_sql_statement("SELECT * from Stocks",conn)
        pricedf = execute_sql_statement("SELECT * from Price", conn)
        ### divide the notional
        Notional = 1000000 // len(stockdf['Ticker'].unique())
        for stk in stockdf['Ticker'].unique():
            print("Back Testing:" + stk)
            data = pricedf[pricedf.Symbol==stk].copy()
            ### select the right parameters according to the training result

            H = stockdf.loc[stockdf[stockdf.Ticker==stk].index,'H'].iloc[0]

            K1 = stockdf.loc[stockdf[stockdf.Ticker==stk].index,'K1'].iloc[0]
            ## calculating rolling H-period Moving average and rolling std
            ## the number of the std to form the band is K1
            data['SMA'] = data['Close'].rolling(H).mean()
            data['rstd'] = data['Close'].rolling(H).std()
            data['Up1'] = data['SMA'] + K1 * data['rstd']
            data['Down1'] = data['SMA'] - K1 * data['rstd']

            ### signals generation
            data['signal'] = 0
            ## when close larger than ma + k1 * std ->銆€short, when close smaller than ma - k1 * std -> long
            data.loc[data['Close'] >= data['Up1'],'signal'] = -1
            data.loc[(data['Close'] < data['Up1']) & (data['Close'] > data['Down1']),'signal'] = 0
            data.loc[data['Close'] <= data['Down1'],'signal'] = 1
            ##銆€shift signal since we trade at next min
            data.signal = data.signal.shift().fillna(0)
            data['trade'] = data.signal.diff()

            ### PnL calculation
            data['pre_trade_pos'] = 0
            data['target_pos'] = 0
            data['realized_pnl_d'] = 0
            last_target_pos = 0 # store the target position we trade at last entry, in order to cal the realized pnl
            for index, row in data.iterrows():
                data.loc[index,'pre_trade_pos'] = last_target_pos
                if row.trade != 0:
                    ## cal the position according to Notional and close, can not do fraction so always make it int
                    data.loc[index,'target_pos'] = row.signal * int(Notional / row['Close'])
                    last_target_pos = row.signal * int(Notional / row['Close'])

                if abs(row.signal) < abs(row.trade): ## when it is an exit, we calculate the realized pnl of this trade
                    if row.trade < 0:
                        data.loc[index,'realized_pnl_d'] = data.loc[index,'pre_trade_pos'] * row['Close'] - Notional
                    else:
                        data.loc[index,'realized_pnl_d'] = Notional + data.loc[index,'pre_trade_pos'] * row['Close']
            ## cal pnl in percentage, also cumulative pnl
            data['realized_pnl_p'] = data['realized_pnl_d'] / Notional
            data['cum_pnl_p'] = data['realized_pnl_p'].cumsum()

            CumPnL = data['realized_pnl_d'].cumsum().iloc[-1]
            Return = data['cum_pnl_p'].iloc[-1]
            engine.execute("UPDATE Stocks SET Profit_Loss = " + str(CumPnL) + " WHERE Ticker = '" + stk +"'""'")
            engine.execute("UPDATE Stocks SET Notional = " + str(Notional) + " WHERE Ticker = '" + stk +"'")
            engine.execute("UPDATE Stocks SET Return = " + str(Return) + " WHERE Ticker = '" + stk +"'")

    def probation_test(probation_testing_start_date, probation_testing_end_date):
        ### read data through data base
        stockdf = pd.read_sql_query("SELECT * from Stocks", conn)
        pricedf = pd.read_sql_query("SELECT * from Price", conn)
        #engine.execute('Drop Table if exists Probation;')
        #create_stocks_table('Probation', metadata, engine)
        stocks = stockdf.copy(deep=True)
        stocks["Profit_Loss_in_Training"] = 0.0
        stocks['Return_in_Training'] = 0.0
        stocks["Profit_Loss"] = 0.0
        stocks['Return'] = 0.0
        ### divide the notional
        Notional = 1000000 // len(stockdf['Ticker'].unique())
        for stk in stockdf['Ticker'].unique():
            print("Back Testing:" + stk)
            data = pricedf[pricedf.Symbol==stk].copy()
            data = data[(data.DateTime>=probation_testing_start_date)&(data.DateTime<=probation_testing_end_date)]
            ### select the right parameters according to the training result

            H = stockdf.loc[stockdf[stockdf.Ticker==stk].index,'H'].iloc[0]

            K1 = stockdf.loc[stockdf[stockdf.Ticker==stk].index,'K1'].iloc[0]
            ## calculating rolling H-period Moving average and rolling std
            ## the number of the std to form the band is K1
            data['SMA'] = data['Close'].rolling(H).mean()
            data['rstd'] = data['Close'].rolling(H).std()
            data['Up1'] = data['SMA'] + K1 * data['rstd']
            data['Down1'] = data['SMA'] - K1 * data['rstd']

            ### signals generation
            data['signal'] = 0
            ## when close larger than ma + k1 * std ->鍗€short, when close smaller than ma - k1 * std -> long
            data.loc[data['Close'] >= data['Up1'],'signal'] = -1
            data.loc[(data['Close'] < data['Up1']) & (data['Close'] > data['Down1']),'signal'] = 0
            data.loc[data['Close'] <= data['Down1'],'signal'] = 1
            ##鍗€shift signal since we trade at next min
            data.signal = data.signal.shift().fillna(0)
            data['trade'] = data.signal.diff()

            ### PnL calculation
            data['pre_trade_pos'] = 0
            data['target_pos'] = 0
            data['realized_pnl_d'] = 0
            last_target_pos = 0 # store the target position we trade at last entry, in order to cal the realized pnl
            for index, row in data.iterrows():
                data.loc[index,'pre_trade_pos'] = last_target_pos
                if row.trade != 0:
                    ## cal the position according to Notional and close, cannot do fraction so always make it int
                    data.loc[index,'target_pos'] = row.signal * int(Notional / row['Close'])
                    last_target_pos = row.signal * int(Notional / row['Close'])

                if abs(row.signal) < abs(row.trade): ## when it is an exit, we calculate the realized pnl of this trade
                    if row.trade < 0:
                        data.loc[index,'realized_pnl_d'] = data.loc[index,'pre_trade_pos'] * row['Close'] - Notional
                    else:
                        data.loc[index,'realized_pnl_d'] = Notional + data.loc[index,'pre_trade_pos'] * row['Close']
            ## cal pnl in percentage, also cumulative pnl
            data['realized_pnl_p'] = data['realized_pnl_d'] / Notional
            data['cum_pnl_p'] = data['realized_pnl_p'].cumsum()

            CumPnL = data['realized_pnl_d'].cumsum().iloc[-1]
            stocks.loc[stocks[stocks.Ticker==stk].index,'Profit_Loss'] = CumPnL
            #stocks.to_sql('Probation', con=engine, if_exists='append', index=False)
        return stocks

    def populate_strategy_bt():

        stockdf = pd.read_sql_query("SELECT * from Stocks", conn)
        Notional = 1000000.00
        train_Pnl = stockdf.Profit_Loss_in_Training.sum()
        train_Ret = train_Pnl / Notional

        bt_Pnl =  stockdf.Profit_Loss.sum()
        bt_Ret = bt_Pnl / Notional

        ### divide time into training period and bt period
        bt_end_date = dt.datetime.today().date()
        bt_start_date = end_date - relativedelta(weeks=1)
        train_end_date = bt_start_date - relativedelta(days=1)
        train_start_date = train_end_date - relativedelta(weeks=1)
        ### get SPY data
        spy_df = pd.DataFrame(get_daily_data('SPY',train_start_date,bt_end_date))
        train_start_open = spy_df.open.iloc[0]
        train_end_close = spy_df.loc[spy_df.date<=str(train_end_date),'close'].iloc[-1]

        ### calculate return of SPY, dividing into training and back testing

        bt_start_open = spy_df.loc[spy_df.date>=str(bt_start_date),'open'].iloc[0]
        bt_end_close = spy_df.close.iloc[-1]
        train_ret = (train_end_close - train_start_open) / train_start_open

        bt_ret = (bt_end_close - bt_start_open) / bt_start_open

        ### populate result into strategy table
        strategy = pd.DataFrame(index=[0],columns = ['Strategy'])
        strategy['Strategy'] = 'BB'
        strategy['Notional'] = Notional
```

```
            strategy['PnL_in_Training'] = train_Pnl
            strategy['Return_in_Training'] = train_Ret
            strategy['SPY_Return_in_Training'] = train_ret
            strategy['PnL'] = bt_Pnl
            strategy['Strategy_Return'] = bt_Ret
            strategy['SPY_Return'] = bt_ret

            strategy.to_sql('Strategy', con=engine, if_exists='append', index=False)
```

2. BBD_client.py

```
# -*- coding: utf-8 -*
#!/usr/bin/env python3
#@ Copyright -

import json
import sys

from network import PacketTypes, Packet
import queue
import threading
import pandas as pd
import numpy as np
from queue import Queue
from socket import AF_INET, socket, SOCK_STREAM, IPPROTO_TCP, TCP_NODELAY

import datetime

from sqlalchemy import create_engine
from sqlalchemy import MetaData
import time

def usd(value):
    """Format value as USD."""
    return f"${value:,.2f}"

def get_stock_selection_list():
    enginestk = create_engine('sqlite:///TradingBBD.db',connect_args={'check_same_thread': False})
    connstk = enginestk.connect()
    stockdf = pd.read_sql_query("SELECT * from Stocks", connstk)
    stklist = list(stockdf.Ticker)
    return stklist

class ClientConfig:
    def __init__(self):
        self.client_id = "client1"
        ### server side below
        self.HOST = "127.0.0.1" ### local host, local ip
        self.PORT = 6510
        self.BUF_SIZE = 4096 ### maximum message 4k
        self.ADDR = (self.HOST, self.PORT)

        self.client_socket = socket(AF_INET, SOCK_STREAM)
        self.client_socket.setsockopt(IPPROTO_TCP, TCP_NODELAY, True)
        self.client_thread = threading.Thread()
        self.client_receiver = threading.Thread()

        self.orders = []

        self.client_thread_started = False
        self.trade_complete = False
        self.client_up = False
        self.client_symbols = ''
        self.PnL = 0
        self.Ticker_PnL = {}


trading_queue = queue.Queue()
trading_event = threading.Event()
client_config = ClientConfig()

def receive(q=None, e=None):
    total_server_response = b'' ##in bytes
    while True:
        try:
            server_response = client_config.client_socket.recv(client_config.BUF_SIZE)
            total_server_response += server_response
            msgSize = len(total_server_response)
            while msgSize > 0: ## get sth
                if msgSize > 12: ## more than 3 int
                    server_packet = Packet()
                    server_packet.deserialize(total_server_response)
                if msgSize > 12 and server_packet.m_data_size <= msgSize: ## i have complete message
                    data = json.loads(server_packet.m_data)
                    q.put([server_packet.m_type, data]) ## pass from receiver to sender, via queue
                    total_server_response = total_server_response[server_packet.m_data_size:] ## cut the message and move on
                    msgSize = len(total_server_response)
                else: ## not enough, keep receiving more
                    server_response = client_config.client_socket.recv(client_config.BUF_SIZE)
                    total_server_response += server_response
                    msgSize = len(total_server_response)

            if not q.empty() and e.isSet():
                e.clear()

        except (OSError,Exception):
            print("Exception in receive\n")
            sys.exit(0)

def send_msg(client_packet):
    client_config.client_socket.send(client_packet.serialize())
    data = json.loads(client_packet.m_data)
#    print(data) ## take a look
    return data

def get_response(q):
    msg_type, msg_data = q.get()
#    print(msg_data)
    if msg_data is not None:
        if msg_type == PacketTypes.END_RSP.value or msg_type == PacketTypes.SERVER_DOWN_RSP.value or \
                       (msg_type == PacketTypes.CONNECTION_RSP.value and msg_data["Status"] == "Rejected"):
```

```
                client_config.client_socket.close() ## client get out
                sys.exit(0)
        return msg_data

    def set_event(e):
        e.set()

    def wait_for_an_event(e):
        while e.isSet():
            continue

    def send(q=None, e=None):
        try:
            while True:
                client_packet = Packet()
                user_input = input("Action:")
                input_list = user_input.strip().split(" ")
                if len(input_list) < 2:
                    print("Incorrect Input.\n")
                    continue
                ### case sensitive, won't use next week
                if "Logon" in user_input:
                    client_packet.m_type = PacketTypes.CONNECTION_REQ.value ### enum here
                    client_packet.m_data = json.dumps({'Client':client_config.client_id, 'Status':input_list[0], 'Symbol':input_list[1]})

                elif "Client List" in user_input:
                    client_packet.m_type = PacketTypes.CLIENT_LIST_REQ.value
                    client_packet.m_data  = json.dumps({'Client':client_config.client_id, 'Status':input_list[0] + ' ' + input_list[1]})

                elif "Stock List" in user_input:
                    client_packet.m_type = PacketTypes.STOCK_LIST_REQ.value
                    client_packet.m_data  = json.dumps({'Client':client_config.client_id, 'Status':input_list[0] + ' ' + input_list[1]})

                elif "Book Inquiry" in user_input:
                    if len(input_list) < 3:
                        print("Missing input item(s).\n")
                        continue
                    client_packet.m_type = PacketTypes.BOOK_INQUIRY_REQ.value
                    client_packet.m_data  = json.dumps({'Client':client_config.client_id, 'Status':input_list[0] + ' ' + input_list[1], 'Symbol':input_list[2]})

                elif "New Order" in user_input:
                    if len(input_list) < 6:
                        print("Missing input item(s).\n")
                        continue
                    client_packet.m_type = PacketTypes.NEW_ORDER_REQ.value
                    client_packet.m_data  = json.dumps({'Client':client_config.client_id, 'Status':input_list[0] + ' ' + input_list[1], 'Symbol':input_list[2], 'Type':'Lmt', 'Side':input_list[3], 'Price':input_list[4], 'Qty':input_list[5]})

                elif "Client Quit" in user_input:
                    client_packet.m_type = PacketTypes.END_REQ.value
                    client_packet.m_data = json.dumps({'Client':client_config.client_id, 'Status':input_list[0]})

                else:
                    print("Invalid message\n")
                    continue

                set_event(e)
                send_msg(client_packet)
                wait_for_an_event(e)
                msg_type, msg_data = q.get()
                q.task_done()
                print(msg_data)
                if msg_data is not None:
                    if msg_type == PacketTypes.END_RSP.value or msg_type == PacketTypes.SERVER_DOWN_RSP.value or \
                        (msg_type == PacketTypes.CONNECTION_RSP.value and msg_data["Status"] == "Rejected"):
                        client_config.client_socket.close()
                        sys.exit(0)

        except(OSError, Exception):
            q.put(PacketTypes.NONE.value, Exception('send'))
            client_socket.close()
            sys.exit(0)




    def Client_Manual():
        try:

                client_config.client_socket = socket(AF_INET, SOCK_STREAM)
                client_config.client_socket.setsockopt(IPPROTO_TCP, TCP_NODELAY, True)
                status = client_config.client_socket.connect_ex(client_config.ADDR)
                if status != 0:
                    print("Fail in connecting to server")
                    sys.exit(0)

                ## send thread and receive thread, thread is also object
                client_config.client_receiver = threading.Thread(target=receive, args=(trading_queue, trading_event))
                client_config.client_sender = threading.Thread(target=send, args=(trading_queue, trading_event))

                client_config.client_receiver.start()
                client_config.client_sender.start()

                if client_config.client_receiver.is_alive() is True:
                    client_config.client_receiver.join()

                if client_config.client_sender.is_alive() is True:
                    client_config.client_receiver.join()

        except (KeyError, KeyboardInterrupt, SystemExit, Exception):
                client_config.client_socket.close()
                sys.exit(0)


    def logon(client_packet):
        client_config.client_symbols = ','.join([str(elem) for elem in get_stock_selection_list()])
        client_packet.m_type= PacketTypes.CONNECTION_REQ.value
        client_packet.m_data = json.dumps({'Client':client_config.client_id, 'Status':'Logon', 'Symbol':client_config.client_symbols})
        return client_packet

    def get_client_list(client_packet):
        client_packet.m_type = PacketTypes.CLIENT_LIST_REQ.value
        client_packet.m_data  = json.dumps({'Client':client_config.client_id, 'Status':'Client List'})
        return client_packet

    def get_stock_list(client_packet):
        client_packet.m_type = PacketTypes.STOCK_LIST_REQ.value
        client_packet.m_data  = json.dumps({'Client':client_config.client_id, 'Status':'Stock List'})
        return client_packet

    def get_market_status(client_packet):
        client_packet.m_type = PacketTypes.MARKET_STATUS_REQ.value
        client_packet.m_data = json.dumps({'Client':client_config.client_id, 'Status':'Market Status'})
        return client_packet

    def get_order_book(client_packet, symbol):
        client_packet.m_type = PacketTypes.BOOK_INQUIRY_REQ.value
        client_packet.m_data = json.dumps({'Client':client_config.client_id, 'Status':'Book Inquiry', 'Symbol':symbol})
        return client_packet

    def new_order(client_packet, order_id, symbol, order_type, side, price, qty):
        if order_type == "Mkt":
            price = 0
        client_packet.m_type = PacketTypes.NEW_ORDER_REQ.value
        client_packet.m_data = json.dumps({'Client':client_config.client_id, 'OrderIndex':order_id, 'Status':'New Order', 'Symbol':symbol, 'Type':order_type, 'Side':side, 'Price':price, 'Qty':qty})
        return client_packet

    def client_quit(client_packet):
        client_packet.m_type = PacketTypes.END_REQ.value
        client_packet.m_data = json.dumps({'Client':client_config.client_id, 'Status':'Client Quit'})
        return client_packet

    class StocksInfo():
        def __init__(self, Ticker_, H_, K1_, Notional_, price_queue_):
            self.Ticker = Ticker_
            self.H = H_
            self.K1 = K1_
            self.Notional = Notional_
            self.price_queue = price_queue_
            self.Std = "null"
            self.MA = "null"
            self.position = 0
            self.Qty = 0
            self.current_price_buy = 0
            self.current_price_sell = 1e6
            self.Tradelist = []
            self.PnLlist = []
            self.PnL = 0

    def StkInfo_init():
        enginestk = create_engine('sqlite:///TradingBBD.db',connect_args={'check_same_thread': False})
        connstk = enginestk.connect()
        stockdf = pd.read_sql_query("SELECT * from Stocks", connstk) ###
        stockdf.index = stockdf['Ticker']
        stock_info_dict = {stk:StocksInfo(stk,stockdf.loc[stk,'H'], stockdf.loc[stk,'K1'], stockdf.loc[stk,'Notional'],
                                          Queue(int(stockdf.loc[stk,'H'] / 5))) for stk in stockdf['Ticker']}
        return stock_info_dict

    class Trade():
        def __init__(self, Ticker_, Orders_Responses_):     ### open trade
            Orders_Responses = [response for response in Orders_Responses_ if response['Symbol'] == Ticker_]
            self.Ticker = Ticker_
            self.Postion = 1 if Orders_Responses[0]['Side'] == 'Buy' else -1
            self.ClosePrice = np.nan
            self.PnL = np.nan
            ### Open Order Filled with one order
            if len(Orders_Responses) == 1:
                Filled_order = Orders_Responses[0]
                Price = Filled_order['Price']
                Qty = int(Filled_order['Qty'])
                self.OpenPrice = Price
                self.Qty = Qty
            ### Open Order Filled with several orders
            else:
                PriceList = [orders['Price'] for orders in Orders_Responses]
                QtyList = [int(orders['Qty']) for orders in Orders_Responses]
                self.Qty = sum(QtyList)
                self.OpenPrice = sum([P * Q for P,Q in zip(PriceList, QtyList)]) / sum(QtyList)
            print("Open Trade in: ", self.Ticker, "With postion: ", self.Postion, "at Price: ", self.OpenPrice, "With Qty: ", self.Qty)
        def CloseTrade(self, Orders_Responses_):
            Orders_Responses = [response for response in Orders_Responses_ if response['Symbol'] == self.Ticker]
            ### Close Order Filled with one order
            if len(Orders_Responses) == 1:
                Filled_order = Orders_Responses[0]
                Price = Filled_order['Price']
                self.ClosePrice = Price
                self.PnL = (self.ClosePrice - self.OpenPrice) * self.Qty if self.Postion == 1 else (self.OpenPrice - self.ClosePrice) * self.Qty
            ### Close Order Filled with several orders
            else:
                PriceList = [orders['Price'] for orders in Orders_Responses]
                QtyList = [int(orders['Qty']) for orders in Orders_Responses]
                self.ClosePrice = sum([P * Q for P,Q in zip(PriceList, QtyList)]) / sum(QtyList)
                self.PnL = (self.ClosePrice - self.OpenPrice) * self.Qty if self.Postion == 1 else (self.OpenPrice - self.ClosePrice) * self.Qty
            print("Close Trade in: ", self.Ticker, "at Open Price: ", self.OpenPrice, "at Close Price: ", self.ClosePrice, "With Qty: ", self.Qty, "PnL: ",self.PnL)


    def LogonAndTrade(q=None, e=None):

        client_packet = Packet()
        set_event(e)
        send_msg(logon(client_packet))
        wait_for_an_event(e)
        get_response(q)

        set_event(e)
        send_msg(get_client_list(client_packet))
        wait_for_an_event(e)
        get_response(q)

        set_event(e)
        send_msg(get_stock_list(client_packet))
        wait_for_an_event(e)
        get_response(q)

        ### find end date
        lastBusDay = datetime.datetime.today()
        if datetime.date.weekday(lastBusDay) == 5:      #if it's Saturday
            lastBusDay = lastBusDay - datetime.timedelta(days = 1) #then make it Friday
        elif datetime.date.weekday(lastBusDay) == 6:      #if it's Sunday
            lastBusDay = lastBusDay - datetime.timedelta(days = 2)
        ##TO DO
        end_date = lastBusDay - datetime.timedelta(days = 1)  ###
        end_trading_mktperiod = end_date.strftime("%Y-%m-%d")

        ### initialize StkInfo Dict
        StockInfoDict = StkInfo_init()
        base_filled_orderid = []
    #    sample = open('orderbook.txt', 'w')
        OrderIndex = 0
        while True: ### outer loop
            while True: ### inner loop
                client_packet = Packet()
                set_event(e)
                send_msg(get_market_status(client_packet))
                wait_for_an_event(e)
                ### when not empty
                while not q.empty():
                    data = get_response(q)
                    if data['Status'] not in ['Open','Pending Closing','Market Closed','Pending Open']:
                        client_config.orders.append(data)
                    else:
                        mkt_status = data
    #            print("mkt_status", mkt_status)
                ### wait till mkt open
                if mkt_status["Status"] == 'Open' or mkt_status["Status"] == 'Pending Closing':
                    break
                if mkt_status['Market_Period'] == end_trading_mktperiod and mkt_status["Status"] == "Market Closed":
                    ### calcualte PnL and stop trade Logic 1
    #                TotalPnL = 0
    #                for stk in StockInfoDict:
    #                    TotalPnL += sum(StockInfoDict[stk].PnLlist)
    #                client_config.PnL = TotalPnL
    #                ### calculate PnL for different Ticker
    #                PnL_dict = {stk:usd(sum(StockInfoDict[stk].PnLlist)) for stk in StockInfoDict}
    #                client_config.Ticker_PnL = PnL_dict
                    ### PnL Calculation Logic 2
                    PnL_dict = {}
                    for stk in StockInfoDict:
                        stkbuy_order = [order for order in client_config.orders if (order['Symbol'] == stk)&(order['Side'] == 'Buy')]
                        stkbuy_price = [order['Price'] for order in stkbuy_order]
                        stkbuy_qty = [int(order['Qty']) for order in stkbuy_order]
                        stksell_order = [order for order in client_config.orders if (order['Symbol'] == stk)&(order['Side'] == 'Sell')]
                        stksell_price = [order['Price'] for order in stksell_order]
                        stksell_qty = [int(order['Qty']) for order in stksell_order]
                        stkPnL = sum([P * Q for P,Q in zip(stksell_price, stksell_qty)]) - sum([P * Q for P,Q in zip(stkbuy_price, stkbuy_qty)])
                        PnL_dict.update({stk:stkPnL})

                    client_config.PnL = sum(PnL_dict.values())
                    client_config.Ticker_PnL = {stk: usd(PnL_dict[stk]) for stk in PnL_dict}
                    ### complete the trade
                    client_config.trade_complete = True
                    break
                time.sleep(0.5)
            if client_config.trade_complete:
                break
            ### close every day in Pending Close
            if mkt_status["Status"] == "Pending Closing":
    #            OrderIndex = 0
                for stk in StockInfoDict:
                    stkInfo_object = StockInfoDict[stk]
                    stkInfo_object.MA = 'null'
                    stkInfo_object.Std = 'null'
                    stkInfo_object.price_queue = Queue(int(stkInfo_object.H / 5))  # reset the members
                    if stkInfo_object.position != 0:
                        client_packet = Packet()
                        OrderIndex += 1
                        client_order_id = client_config.client_id + '_' + str(OrderIndex)
                        ### if longing
                        if stkInfo_object.position > 0:
                            new_order(client_packet, client_order_id, stk, 'Mkt', 'Sell', 100, stkInfo_object.Qty)
                            print("Close Trade in: ", stk, "With postion: Sell", "With Qty: ", stkInfo_object.Qty)
                            print("Because: Close at Pending Close.")
                            set_event(e)
                            send_msg(client_packet)
                            wait_for_an_event(e)
                            ### close trade logic
                            response_list = []
                            while not q.empty():
                                response_data = get_response(q)
                                response_list.append(response_data)
                                client_config.orders.append(response_data)
    #                         Trade_object = stkInfo_object.Tradelist[-1]
    #                         Trade_object.CloseTrade(response_list)
    #                         stkInfo_object.PnLlist.append(Trade_object.PnL)
                            stkInfo_object.Qty = 0
                            stkInfo_object.position = 0

                        ### if shorting
                        else:
                            new_order(client_packet, client_order_id, stk, 'Mkt', 'Buy', 100, stkInfo_object.Qty)
                            print("Close Trade in: ", stk, "With postion: Buy", "With Qty: ", stkInfo_object.Qty)
                            print("Because: Close at Pending Close.")
                            set_event(e)
                            send_msg(client_packet)
                            wait_for_an_event(e)
                            ### close trade logic
                            response_list = []
                            while not q.empty():
                                response_data = get_response(q)
                                response_list.append(response_data)
                                client_config.orders.append(response_data)
    #                         Trade_object = stkInfo_object.Tradelist[-1]
    #                         Trade_object.CloseTrade(response_list)
    #                         stkInfo_object.PnLlist.append(Trade_object.PnL)
                            stkInfo_object.Qty = 0
                            stkInfo_object.position = 0
                continue ### re-enter into checking "Open" while-loop

            ### BBD Trading Logic
            client_packet = Packet()
            set_event(e)
            client_msg = get_order_book(client_packet, client_config.client_symbols)
            send_msg(client_msg)
            wait_for_an_event(e)

            ### when not empty
            while True:
                data = get_response(q)
                if type(data) == dict:
                    if data['Status']!= 'Done':
                        client_config.orders.append(data)
                else:
                    break

            book_data = json.loads(data)
            order_book = book_data["data"]

    #        print(order_book, file = sample)
```

```

        filled_order_book = [fill_orders for fill_orders in order_book if fill_orders['Status'] in ['Filled']]
        filled_orderid = [order['OrderIndex'] for order in filled_order_book]
        standing_order_book = [standing_orders for standing_orders in order_book if standing_orders['Status'] in ['New','Partial Filled']]
#        print(filled_order_book, file = sample)
#        print(standing_order_book, file = sample)

#        OrderIndex = 0

        for stk in StockInfoDict:
            standing_buy_price_list = [order['Price'] for order in standing_order_book if (order['Symbol'] == stk)&(order['Side'] == 'Buy')]
            standing_sell_price_list = [order['Price'] for order in standing_order_book if (order['Symbol'] == stk)&(order['Side'] == 'Sell')]
            StockInfoDict[stk].current_price_sell = min(standing_sell_price_list)
            StockInfoDict[stk].current_price_buy = max(standing_buy_price_list)

        ### store current price in price queue and use it to calculate MA and std
        for stk in StockInfoDict:
            stkInfo_object = StockInfoDict[stk]
            ### current price is based on filled_order_book
            if len(base_filled_orderid) == 0:
                current_price = (stkInfo_object.current_price_buy + stkInfo_object.current_price_sell) / 2
                base_filled_orderid = filled_orderid
            else:
                try:
                    newly_filled_orderid = [orderid for orderid in filled_orderid if orderid not in base_filled_orderid]
                    base_filled_orderid = filled_orderid
                    newly_filled_order = [order for order in filled_order_book if order['OrderIndex'] in newly_filled_orderid]
                    filled_price_list = [order['Price'] for order in newly_filled_order]
                    filled_qty_list = [int(order['OrigQty']) for order in newly_filled_order]
                    current_price = sum([P * Q for P,Q in zip(filled_price_list, filled_qty_list)]) / sum(filled_qty_list)
                except: ### when no newly filled
                    current_price = (stkInfo_object.current_price_buy + stkInfo_object.current_price_sell) / 2
#            print("current price for", stk, "P= " current_price)
            if not stkInfo_object.price_queue.full():
                stkInfo_object.price_queue.put(current_price)
                if stkInfo_object.price_queue.full():
                    stkInfo_object.MA = np.array(stkInfo_object.price_queue.queue).mean()
                    stkInfo_object.Std = np.array(stkInfo_object.price_queue.queue).std() / np.sqrt(5)
            else: # already full
                popout = stkInfo_object.price_queue.get()
                stkInfo_object.price_queue.put(current_price)
                stkInfo_object.MA = np.array(stkInfo_object.price_queue.queue).mean()
```

```
                        stkInfo_object.Std = np.array(stkInfo_object.price_queue.queue).std() / np.sqrt(5)


                for stk in StockInfoDict:
                    stkInfo_object = StockInfoDict[stk]
                    K1 = stkInfo_object.K1
                    MA = stkInfo_object.MA
                    Std = stkInfo_object.Std
                    Notional = stkInfo_object.Notional
                    if MA == 'null':
                        continue
                    current_buy = stkInfo_object.current_price_buy
                    current_sell = stkInfo_object.current_price_sell
        #            current_p = (current_buy + current_sell) / 2
        #            print("K1: ",K1)
        #            print("MA: ",MA)
        #            print("Std: ",Std)
        #            print("sell p:",current_sell)
        #            print("buy p:",current_buy)
                    if stkInfo_object.position == 0: # not yet open position, could open
                        if current_sell <= MA - K1 * Std: # below lower band, go long
                            stkInfo_object.position = 1
                            client_packet = Packet()
                            OrderIndex += 1
                            client_order_id = client_config.client_id + '_' + str(OrderIndex)
                            stkInfo_object.Qty = int(Notional / current_sell)
                            new_order(client_packet, client_order_id, stk, 'Mkt', 'Buy', 100, stkInfo_object.Qty)
                            print("Open Trade in: ", stk, "With postion: Buy", " at Price:", current_sell, "With Qty:", stkInfo_object.Qty)
                            print("Because: Price below lower band:", usd(MA - K1 * Std))
                            set_event(e)
                            send_msg(client_packet)
                            wait_for_an_event(e)
                            ### open logic
                            response_list = []
                            while not q.empty():
                                response_data = get_response(q)
                                response_list.append(response_data)
                                client_config.orders.append(response_data)
        #                    Trade_object = Trade(stk, response_list)
        #                    stkInfo_object.Tradelist.append(Trade_object)



                        elif current_buy >= MA + K1 * Std: # above upper band, go short
                            stkInfo_object.position = -1
                            client_packet = Packet()
                            OrderIndex += 1
                            client_order_id = client_config.client_id + '_' + str(OrderIndex)
                            stkInfo_object.Qty = int(Notional / current_buy)
                            new_order(client_packet, client_order_id, stk, 'Mkt', 'Sell', 100, stkInfo_object.Qty)
                            print("Open Trade in: ", stk, "With postion: Sell", "at Price:", current_buy, "With Qty: ", stkInfo_object.Qty)
                            print("Because: Price above upper band:", usd(MA + K1 * Std))
                            set_event(e)
                            send_msg(client_packet)
                            wait_for_an_event(e)
                            ### open logic
                            response_list = []
                            while not q.empty():
                                response_data = get_response(q)
                                response_list.append(response_data)
                                client_config.orders.append(response_data)
        #                    Trade_object = Trade(stk, response_list)
        #                    stkInfo_object.Tradelist.append(Trade_object)


                    elif stkInfo_object.position == 1: # longing now
                        if current_buy >= MA: # above lower bound, sell to close postion
                            client_packet = Packet()
                            OrderIndex += 1
                            client_order_id = client_config.client_id + '_' + str(OrderIndex)
                            new_order(client_packet, client_order_id, stk, 'Mkt', 'Sell', 100, stkInfo_object.Qty)
                            print("Close Trade in: ", stk, "With postion: Sell", "at Price:", current_buy, "With Qty: ", stkInfo_object.Qty)
                            print("Because: Price above lower band:", usd(MA))
                            set_event(e)
                            send_msg(client_packet)
                            wait_for_an_event(e)
                            ### close trade logic
                            response_list = []
                            while not q.empty():
                                response_data = get_response(q)
                                response_list.append(response_data)
                                client_config.orders.append(response_data)
        #                    Trade_object = stkInfo_object.Tradelist[-1]
        #                    Trade_object.CloseTrade(response_list)
        #                    stkInfo_object.PnLlist.append(Trade_object.PnL)
                            stkInfo_object.Qty = 0
                            stkInfo_object.position = 0


                    else: # shorting now
                        if current_sell <= MA: # below upper bound, buy to close postion
                            client_packet = Packet()
                            OrderIndex += 1
                            client_order_id = client_config.client_id + '_' + str(OrderIndex)
                            new_order(client_packet, client_order_id, stk, 'Mkt', 'Buy', 100, stkInfo_object.Qty)
                            print("Close Trade in: ", stk, "With postion: Buy", "at Price:", current_sell, "With Qty: ", stkInfo_object.Qty)
                            print("Because: Price below upper band:", usd(MA))
                            set_event(e)
                            send_msg(client_packet)
                            wait_for_an_event(e)
                            ### close trade logic
```

```
                          response_list = []
                          while not q.empty():
                              response_data = get_response(q)
                              response_list.append(response_data)
                              client_config.orders.append(response_data)
      #                     Trade_object = stkInfo_object.Tradelist[-1]
      #                     Trade_object.CloseTrade(response_list)
      #                     stkInfo_object.PnLlist.append(Trade_object.PnL)
                          stkInfo_object.Qty = 0
                          stkInfo_object.position = 0

              time.sleep(1) ### request order book every sec
```

3. ClientFrontEnd.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 19 22:39:06 2020

@author: 63438
"""
import pandas as pd
from flask import Flask, render_template, request

from BBDModel import stk_select, build_trading_model, populate_backtest_data, backTest, probation_test, execute_sql_statement, populate_strategy_bt
from BBD_client import *

from network import PacketTypes, Packet
import queue
import threading
from socket import AF_INET, socket, SOCK_STREAM, IPPROTO_TCP, TCP_NODELAY


def usd(value):
    """Format value as USD."""
    return f"${value:,.2f}"

def FrontEnd(engine):
    app = Flask(__name__)
    selected_list = []
    @app.route('/')
    def index():
        return render_template("index.html")


    @app.route('/Stock_selection')
    def Stock_selection():
        slist, sdf = stk_select()
        for i in slist:
            selected_list.append(i)
        stk_df = pd.DataFrame(sdf)
        stk_df['symbol'] = stk_df.index
        stk_df['std'] = stk_df['std'].map('{:.4f}'.format)
        stk_df = stk_df.transpose()
        list_of_stk = [stk_df[i] for i in stk_df]
        return render_template("stock_selection.html", stock_list=list_of_stk)


    @app.route('/Train_model')
    def train_model():
        stocksdf = build_trading_model(selected_list)
        populate_backtest_data(stocksdf)
        stocksdf['Profit_Loss_in_Training'] = stocksdf['Profit_Loss_in_Training'].map('${:,.2f}'.format)
        stocksdf = stocksdf.transpose()
        train_results = [stocksdf[i] for i in stocksdf]
        return render_template("train_model.html", train_list=train_results)


    @app.route('/Back_test')
    def model_back_testing():
        conn = engine.connect()
        backTest()
        populate_strategy_bt()
        result_df = execute_sql_statement("SELECT * from Stocks", conn)
        total = result_df['Profit_Loss'].sum()
        result_df['Profit_Loss'] = result_df['Profit_Loss'].map('${:,.2f}'.format)
        result_df = result_df.transpose()
        trade_results = [result_df[i] for i in result_df]
        return render_template("back_test_result.html", bt_list=trade_results, total=usd(total))


    @app.route('/Probation_test', methods = ['POST', 'GET'])
    def model_probation_testing():
        if request.method == 'POST':

            form_input = request.form
            probation_testing_start_date = form_input['Start Date']
            probation_testing_end_date = form_input['End Date']

            result_df = probation_test(probation_testing_start_date, probation_testing_end_date)
            total = result_df['Profit_Loss'].sum()
            result_df['Profit_Loss'] = result_df['Profit_Loss'].map('${:,.2f}'.format)
            result_df = result_df.transpose()
            trade_results = [result_df[i] for i in result_df]
            return render_template("probation_test_result.html", trade_list=trade_results, total=usd(total))
        else:

            return render_template("probation_test.html")

    @app.route('/start_trading',methods = ['POST', 'GET'])
    def start_trading():
        if request.method == 'POST':
            if request.form.get("Auto Trading"):
                client_config.client_socket = socket(AF_INET, SOCK_STREAM)
                client_config.client_socket.setsockopt(IPPROTO_TCP, TCP_NODELAY, True)
                status = client_config.client_socket.connect_ex(client_config.ADDR)
                if status != 0:
                    print("Fail in connecting to server")
                    sys.exit(0)

                ## send thread and receive thread, thread is also object
```

```
     client_config.client_receiver = threading.Thread(target=receive, args=(trading_queue, trading_event))
     client_config.client_sender = threading.Thread(target=LogonAndTrade, args=(trading_queue, trading_event))

            client_config.client_receiver.start()
            client_config.client_sender.start()

            while not client_config.trade_complete:
                continue
#            client_packet = Packet()
#            send_msg(get_stock_list(client_packet))
#            data1 = get_response(trading_queue)
            return render_template("auto_trading.html",trading_results=client_config.orders, total=usd(client_config.PnL)) ##client_config.orders
        elif request.form.get("Analysis"):
            return render_template("auto_trading_analysis.html",trading_results=client_config.Ticker_PnL)
    else:
        return render_template("start_trading.html")


@app.route('/manual_trading',methods = ['POST', 'GET'])
def manual_trading():
    if request.method == 'POST':
        form_input = request.form
        print(form_input)
        trading_id = form_input['OrderId']
        trading_Ticker = form_input['Symbol']
        trading_Side = form_input['Side']
        trading_Price = form_input['Price']
        trading_Quantity = form_input['Quantity']
        ### edit the packet
        client_packet = Packet()
        client_msg = new_order(client_packet, trading_id, trading_Ticker, 'Lmt', trading_Side, trading_Price, trading_Quantity)
        send_msg(client_msg)
        data = get_response(trading_queue)
        while not trading_queue.empty():
            client_config.orders.append(get_response(trading_queue))

        return render_template("trading_results.html", trading_results=data)
    else:

        return render_template("manual_trading.html")

@app.route('/client_down')
def client_down():
    client_packet = Packet()
    msg_data = {}
    try:
        send_msg(client_quit(client_packet))
        msg_type, msg_data = trading_queue.get()
        trading_queue.task_done()
        print(msg_data)
        return render_template("client_down.html", server_response=msg_data)
    except(OSError, Exception):
        print(msg_data)
```

```
                    return render_template("client_down.html", server_response=msg_data)

            return app

```

4. FRE_server5.py

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python3
#@ Copyright - Song Tang

import json
import urllib.request
import sys
import pandas as pd
import random
import numpy as np

from socket import AF_INET, socket, SOCK_STREAM, IPPROTO_TCP, TCP_NODELAY, gethostname, gethostbyname
import threading
import queue

from sqlalchemy import create_engine
from sqlalchemy import MetaData

import sched, time
import datetime

from pandas.tseries.holiday import USFederalHolidayCalendar, GoodFriday
from pandas.tseries.offsets import CustomBusinessDay

from network import PacketTypes, Packet

import pandas_market_calendars as mcal

serverID = "Server"
engine = create_engine('sqlite:///BBDTradingServer.db')
conn = engine.connect()
conn.execute("PRAGMA foreign_keys = ON")

metadata = MetaData()
metadata.reflect(bind=engine)

mutex = threading.Lock()

requestURL = "https://eodhistoricaldata.com/api/eod/"
myEodKey = "5ba84ea974ab42.45160048"
defaultStartDate = "2020-06-01"
defaultEndDate = "2020-09-30"
def get_daily_data(symbol, startDate=defaultStartDate, endDate=defaultEndDate, apiKey=myEodKey):
    symbolURL = str(symbol) + ".US?"
    startDateURL = "from=" + str(startDate)
    endDateURL = "to=" + str(endDate)
    apiKeyURL = "api_token=" + apiKey
    completeURL = requestURL + symbolURL + startDateURL + '&' + endDateURL + '&' + apiKeyURL + '&period=d&fmt=json'
    print(completeURL)
    with urllib.request.urlopen(completeURL) as req:
        data = json.load(req)
        return data

def populate_stock_data(tickers, engine, table_name, stock_market_periods):
    column_names = ['symbol', 'date', 'open', 'high', 'low', 'close', 'adjusted_close', 'volume']
    price_data = []
    for ticker in tickers:
        stock = get_daily_data(ticker, stock_market_periods[ticker][0], stock_market_periods[ticker][len(stock_market_periods[ticker])-1])
        for stock_data in stock:
            price_data.append([ticker, stock_data['date'], stock_data['open'], stock_data['high'], stock_data['low'], \
                               stock_data['close'], stock_data['adjusted_close'], stock_data['volume']])
        print(price_data)
    stocks = pd.DataFrame(price_data, columns=column_names)
    stocks.to_sql(table_name, con=engine, if_exists='replace', index=False)


intradayRequestURL = "https://eodhistoricaldata.com/api/intraday/"
myEodKey = "5ba84ea974ab42.45160048"
defaultStartSeconds = "1585800000"
defaultEndSeconds   = "1585886400"
def get_intraday_data(symbol, startTime=defaultStartSeconds, endTime=defaultEndSeconds, apiKey=myEodKey):
    symbolURL = str(symbol) + ".US?"
    startDateURL = "from=" + str(startTime)
    endDateURL = "to=" + str(endTime)
    apiKeyURL = "api_token=" + apiKey
    completeURL = intradayRequestURL + symbolURL + startDateURL + '&' + endDateURL + '&' + apiKeyURL + '&period=d&fmt=json'
    with urllib.request.urlopen(completeURL) as req:
        data = json.load(req)
        return data

def populate_intraday_stock_data(tickers, engine, days_in_seconds):
    column_names = ['datetime', 'symbol', 'open', 'high', 'low', 'close', 'volume']
    for ticker in tickers:
        stock = get_intraday_data(ticker, days_in_seconds[0], days_in_seconds[len(days_in_seconds)-1])
        print(stock)
        price_data = []
        for stock_data in stock:
            if ((stock_data['open'] is not None and stock_data['open'] > 0) and
                (stock_data['high'] is not None and stock_data['high'] > 0) and
                (stock_data['low']  is not None and stock_data['low'] > 0) and
                (stock_data['close'] is not None and stock_data['close'] > 0 )and
                (stock_data['volume'] is not None and stock_data['volume'] > 0)):
                price_data.append([stock_data['datetime'], ticker, stock_data['open'], stock_data['high'], stock_data['low'], \
                               stock_data['close'], stock_data['volume']])

    print(price_data)
    stocks = pd.DataFrame(price_data, columns=column_names)
    stocks = stocks.dropna()
    stocks.to_sql(ticker, con=engine, if_exists='replace', index=False)

def populate_intraday_order_map(symbols, engine, market_periods):
    for i in range(len(market_periods)):
        intraday_order_map[market_periods[i]] = []

    stock_market_periods = {}
    for symbol in symbols:
        stock_market_periods[symbol] = []
        select_st = "SELECT * FROM " + symbol + ";"
        result_set = engine.execute(select_st)
        result_df = pd.DataFrame(result_set.fetchall())
        result_df.columns = result_set.keys()

        for i in range(len(market_periods)):
            if (result_df['datetime'].str.contains(market_periods[i])).any():
                mask = (result_df['datetime'].str.contains(market_periods[i])) & (result_df['symbol'] == symbol)
                result = result_df.loc[(mask.values)]
                intraday_order_map[market_periods[i]].append(result[['symbol', 'open', 'high', 'low', 'close', 'volume']])
                stock_market_periods[symbol].append(market_periods[i])

    #print(intraday_order_map, file = intrday_order_file)

    return intraday_order_map, stock_market_periods


def accept_incoming_connections(q=None):
    while True:
        try:
            client, client_address = fre_server.accept()
            print("%s:%s has connected." % client_address, file=server_output)
            client_thread = threading.Thread(target=handle_client, args=(client,q))
            client_thread.setDaemon(True)
            client_thread.start()
        except (KeyError, KeyboardInterrupt, SystemExit, Exception):
            print("Exception in accept_incoming_connections\n", file=server_output)
            q.put(Exception("accept_incoming_connections"))
            break


def receive(client_socket):
    total_client_request = b''
    msgSize = 0
    while True:
        try:
            client_request = client_socket.recv(buf_size)
            list_client_requests = []
            if len(client_request) > 0:
                total_client_request += client_request
                msgSize = len(total_client_request)
                while msgSize > 0:
                    if msgSize > 12:
                        client_packet = Packet()
                        client_request = client_packet.deserialize(total_client_request)
                        #print(client_packet.m_msg_size, msgSize, len(client_request), file=server_output)
                    if msgSize > 12 and client_packet.m_data_size <= msgSize:
                        #data = json.loads(client_packet.m_data)
                        #print(type(data), data, file=server_output)
                        total_client_request = total_client_request[client_packet.m_data_size:]
                        msgSize = len(total_client_request)
                        client_request = b''
                        list_client_requests.append(client_packet)
                    else:
                        client_request = client_socket.recv(buf_size)
                        total_client_request += client_request
                        msgSize = len(total_client_request)

            return list_client_requests

        except (OSError, Exception):
            del clients[client_socket]
            print("Exception in receive\n", file=server_output)
            q.put(Exception("receive"))
            #raise Exception('receive')
            break


def handle_client(client, q=None):
    global symbols
    global market_period
    while True:
        try:
            list_client_requests = receive(client)
            for client_request in list_client_requests:
                msg_data = json.loads(client_request.m_data)
                msg_type = client_request.m_type
                print(msg_data, file=server_output)
                clientID = msg_data["Client"]

                server_packet = Packet()

                if msg_type == PacketTypes.CONNECTION_REQ.value:
                    server_packet.m_type = PacketTypes.CONNECTION_RSP.value
                    if (clientID in clients.values()):
                        text = "%s duplicated connection request!" % clientID
                        server_msg = json.dumps({'Server': serverID, 'Response': text, 'Status': 'Rejected'})
                    else:
                        client_symbols = list(msg_data["Symbol"].split(','))
                        if all(symbol in symbols for symbol in client_symbols):
                            text = "Welcome %s!" % clientID
                            server_msg = json.dumps({'Server': serverID, 'Response': text, 'Status': 'Ack'})
                            clients[client] = clientID
                        else:
                             text = "%s Not all your symbols are eligible!" % clientID
                             server_msg = json.dumps({'Server': serverID, 'Response': text, 'Status': 'Rejected'})
                    server_packet.m_data = server_msg
                    client.send(server_packet.serialize())
                    data = json.loads(server_packet.m_data)
                    print(data, file=server_output)

                elif msg_type == PacketTypes.END_REQ.value:
                    text = "%s left!" % clientID
                    server_msg = json.dumps({'Server':serverID, 'Response':text, 'Status':'Done'})
                    server_packet.m_type = PacketTypes.END_RSP.value
                    server_packet.m_data = server_msg
                    client.send(server_packet.serialize())
                    data = json.loads(server_packet.m_data)
                    print(data, file=server_output)

                elif msg_type == PacketTypes.CLIENT_LIST_REQ.value:
                    user_list = str('')
                    for clientKey in clients:
                        user_list += clients[clientKey] + str(',')
                        print(clients[clientKey], file=server_output)
                    server_msg = json.dumps({'Client List':user_list})
                    server_packet.m_type = PacketTypes.CLIENT_LIST_RSP.value
                    server_packet.m_data = server_msg
                    client.send(server_packet.serialize())
                    data = json.loads(server_packet.m_data)
                    print(data, file=server_output)

                elif msg_type == PacketTypes.STOCK_LIST_REQ.value:
                    stock_list = ','.join(symbols)
                    server_msg = json.dumps({"Stock List":stock_list})
                    server_packet.m_type = PacketTypes.STOCK_LIST_RSP.value
                    server_packet.m_data = server_msg
                    client.send(server_packet.serialize())
                    data = json.loads(server_packet.m_data)
                    print(data, file=server_output)

                elif msg_type == PacketTypes.SERVER_DOWN_REQ.value:
                    server_msg = json.dumps({'Server':serverID, 'Status':'Server Down Confirmed'})
                    server_packet.m_type = PacketTypes.SERVER_DOWN_RSP.value
                    server_packet.m_data = server_msg
                    client.send(server_packet.serialize())
                    data = json.loads(server_packet.m_data)
                    print(data, file=server_output)

                elif msg_type == PacketTypes.BOOK_INQUIRY_REQ.value:
                    server_packet.m_type = PacketTypes.BOOK_INQUIRY_RSP.value
                    if "Symbol" in msg_data and msg_data["Symbol"] != "":
                        if order_table.empty:
                            print("Server order book is empty\n", file=server_output)
                            text = "Server order book is empty"
```

```
                                    server_msg = json.dumps({'Server':serverID, 'Response':text, 'Status':'Done'})
                                else:
                                    server_msg = json.dumps(order_table.loc[order_table['Symbol'].isin(list(msg_data["Symbol"].split(',')))].to_json(orient='table'))
                            else:
                                print("Bad message, missing symbol\n", file=server_output)
                                text = "Bad message, missing symbol"
                                server_msg = json.dumps({'Server':serverID, 'Response':text, 'Status':'Done'})
                            server_packet.m_data = server_msg
                            client.send(server_packet.serialize())
                            data = json.loads(server_packet.m_data)
                            print(data, file=server_output)

                        elif msg_type == PacketTypes.MARKET_STATUS_REQ.value:
                            server_packet.m_type = PacketTypes.MARKET_STATUS_RSP.value
                            server_msg = json.dumps({'Server':serverID, 'Status':market_status, 'Market_Period':market_period})
                            server_packet.m_data = server_msg
                            client.send(server_packet.serialize())
                            data = json.loads(server_packet.m_data)
                            print(data, file=server_output)

                        elif msg_type == PacketTypes.NEW_ORDER_REQ.value:
                            server_packet.m_type = PacketTypes.NEW_ORDER_RSP.value

                            if market_status == "Market Closed":
                                msg_data["Status"] = "Order Reject"
                                server_msg = json.dumps(msg_data)
                                server_packet.m_data = server_msg
                                client.send(server_packet.serialize())
                                data = json.loads(server_packet.m_data)
                                print(data, file=server_output)

                            mutex.acquire()

                            #TODO Possible issue with price comparison
                            '''''
                            Actually it is due to floating comparison:

                                row["Price"] <= float(msg_data["Price"]

                                On the order book:

                                'OrderIndex': 226, 'Symbol': 'XOM', 'Side': 'Sell', 'Price': 39.45, 'Qty': 2632036, 'Status': 'New'

                                When an order at 39.45 was sent,

                                {'Client': 'client1', 'OrderIndex': '1', 'Status': 'New Order', 'Symbol': 'XOM', 'Type': 'Lmt', 'Side': 'Buy', 'Price': '39.45', 'Qty': '100'}

                                {'Client': 'client1', 'OrderIndex': '1', 'Status': 'Order Reject', 'Symbol': 'XOM', 'Type': 'Lmt', 'Side': 'Buy', 'Price': 39.45, 'Qty': '100'}

                                As the price is floating number, actually it could be a little less than 39.45, so the order was rejected.

                                When an order at 39.46 was sent, we got a fill:

                                {Client': 'client1', 'OrderIndex': '2', 'Status': 'New Order', 'Symbol': 'XOM', 'Type': 'Lmt', 'Side': 'Buy', 'Price': '39.46', 'Qty': '100'}
                                {'Client': 'client1', 'OrderIndex': '2', 'Status': 'Order Fill', 'Symbol': 'XOM', 'Type': 'Lmt', 'Side': 'Buy', 'Price': 39.45, 'Qty': '100', 'ServerOrderID': 226}

                                So the best way is to convert the price to integer for comparison
                            '''

                            if (("Symbol" not in msg_data) or (msg_data["Symbol"] == "")) or \
                                (("Side" not in msg_data) or (msg_data["Side"] == "")) or \
                                (("Type" not in msg_data) or (msg_data["Type"] == "")) or \
                                (("Price" not in msg_data) or (msg_data["Type"] == "Lmt" and msg_data["Price"] == "")) or \
                                (("Qty" not in msg_data) or (msg_data["Qty"] == "") or int(msg_data["Qty"]) < 1):
                                print("Bad message, missing critical data item\n", file=server_output)
                                text = "Bad message, missing critial item"
                                server_msg = json.dumps({'Server':serverID, 'Response':text, 'Status':'Done'})
                                server_packet.m_type = PacketTypes.NEW_ORDER_RSP.value
                                server_packet.m_data = server_msg
                                client.send(server_packet.serialize())
                                data = json.loads(server_packet.m_data)
                                print(data, file=server_output)

                            if msg_data["Type"] == "Lmt":

                                msg_order_qty = int(msg_data["Qty"])

                                for (index, row) in order_table.iterrows():
                                    if msg_data["Status"] == "Order Fill":
                                        break

                                    if ((row["Symbol"] == msg_data["Symbol"]) and
                                        (row["Side"] != msg_data["Side"]) and
                                        (row["Price"] <= float(msg_data["Price"]) if msg_data["Side"] == "Buy" else row["Price"] >= float(msg_data["Price"])) and
                                       (row["Status"] != "Filled") & (row["Status"] != "Open Trade") & (int(row["Qty"]) > 0)):
                                        order_qty = int(row['Qty'])

                                        if (order_qty == msg_order_qty):
                                            order_table.loc[index, 'Qty'] = 0
                                            order_table.loc[index, 'Status'] = 'Filled'
                                            msg_data["Price"] = round(order_table.loc[index, 'Price'], 2)
                                            msg_data['Qty'] = str(msg_order_qty)
                                            msg_data["Status"] = "Order Fill"

                                        elif (order_qty< msg_order_qty):
                                            order_table.loc[index, 'Qty'] = 0
                                            order_table.loc[index, 'Status'] = 'Filled'
                                            msg_data["Price"] = round(order_table.loc[index, 'Price'], 2)
                                            msg_data['Qty'] = str(order_qty)
                                            msg_data["Status"] = "Order Partial Fill"
                                            msg_order_qty -= order_qty
                                        else:
                                            order_table.loc[index, 'Qty'] -= msg_order_qty
                                            order_table.loc[index, 'Status'] = 'Partial Filled'
                                            msg_data["Price"] = round(order_table.loc[index, 'Price'], 2)
                                            msg_data['Qty'] = str(msg_order_qty)
                                            msg_data["Status"] = "Order Fill"

                                        msg_data["ServerOrderID"] = order_table.loc[index, 'OrderIndex']
                                        server_msg = json.dumps(msg_data)
                                        server_packet.m_type = PacketTypes.NEW_ORDER_RSP.value
                                        server_packet.m_data = server_msg
                                        client.send(server_packet.serialize())
                                        data = json.loads(server_packet.m_data)
                                        print(data, file=server_output)

                                if msg_data["Status"] == "New Order":
                                    msg_data["Status"] = "Order Reject"
                                    msg_data["Price"] = round(float(msg_data["Price"]), 2)
                                    server_msg = json.dumps(msg_data)
                                    server_packet.m_type = PacketTypes.NEW_ORDER_RSP.value
                                    server_packet.m_data = server_msg
                                    client.send(server_packet.serialize())
                                    data = json.loads(server_packet.m_data)
                                    print(data, file=server_output)

                            elif msg_data["Type"] == "Mkt":

                                msg_order_qty = int(msg_data["Qty"])

                                while ((order_table["Symbol"] == msg_data["Symbol"]) &
                                    (order_table["Side"] != msg_data["Side"]) &
                                    (order_table["Status"] != "Filled") & (order_table["Status"] != "Open Trade" ) & \
                                    (order_table['Qty'] != 0)).any():
```

```

                                    if msg_data["Status"] == "Order Fill":
                                        break

                                    mask = (order_table["Symbol"] == msg_data["Symbol"]) &  \
                                            (order_table["Side"] != msg_data["Side"]) &  \
                                            (order_table["Status"] != "Filled") & (order_table["Status"] != "Open Trade" ) & \
                                            (order_table['Qty'] != 0)

                                    index = -1
                                    order_qty = 0
                                    if msg_data["Side"] == "Sell":
                                        index = order_table.loc[(mask.values), 'Price'].idxmax()
                                        order_qty = int(order_table.loc[index, 'Qty'])
                                    else:
                                        index = order_table.loc[(mask.values), 'Price'].idxmin()
                                        order_qty = int(order_table.loc[index, 'Qty'])

                                    if (order_qty == msg_order_qty):
                                        order_table.loc[index, 'Qty'] = 0
                                        order_table.loc[index, 'Status'] = 'Filled'
                                        msg_data["Price"] = round(order_table.loc[index, 'Price'], 2)
                                        msg_data['Qty'] = str(msg_order_qty)
                                        msg_data["Status"] = "Order Fill"

                                    elif (order_qty < msg_order_qty):
                                        order_table.loc[index, 'Qty'] = 0
                                        order_table.loc[index, 'Status'] = 'Filled'
                                        msg_data['Qty'] = str(order_qty)
                                        msg_data["Price"] = round(order_table.loc[index, 'Price'], 2)
                                        msg_data["Status"] = "Order Partial Fill"
                                        msg_order_qty -= order_qty
                                    else:
                                        order_table.loc[index, 'Qty'] -= msg_order_qty
                                        order_table.loc[index, 'Status'] = 'Partial Filled'
                                        msg_data["Price"] = round(order_table.loc[index, 'Price'], 2)
                                        msg_data['Qty'] = str(msg_order_qty)
                                        msg_data["Status"] = "Order Fill"

                                    msg_data["ServerOrderID"] = order_table.loc[index, 'OrderIndex']
                                    server_msg = json.dumps(msg_data)
                                    server_packet.m_type = PacketTypes.NEW_ORDER_RSP.value
                                    server_packet.m_data = server_msg
                                    client.send(server_packet.serialize())
                                    data = json.loads(server_packet.m_data)
                                    print(data, file=server_output)

                                if msg_data["Status"] == "New Order":
                                    msg_data["Status"] = "Order Reject"
                                    server_msg = json.dumps(msg_data)
                                    server_packet.m_type = PacketTypes.NEW_ORDER_RSP.value
                                    server_packet.m_data = server_msg
                                    client.send(server_packet.serialize())
                                    data = json.loads(server_packet.m_data)
                                    print(data, file=server_output)

                            else:
                                msg_data["Status"] = "Order Reject"
                                server_msg = json.dumps(msg_data)
                                server_packet.m_type = PacketTypes.NEW_ORDER_RSP.value
                                server_packet.m_data = server_msg
                                client.send(server_packet.serialize())
                                data = json.loads(server_packet.m_data)
                                print(data, file=server_output)

                            mutex.release()

                        else:
                            print("Unknown Message from Client\n", file=server_output)
                            text = "Unknown Message from Client"
                            server_msg = json.dumps({"Server":serverID, "Response":text, "Status":"Done"})
                            print(server_msg, file=server_output)
                            server_packet.m_type = PacketTypes.END_RSP.value

                            server_packet.m_data = server_msg
                            client.send(server_packet.serialize())
                            data = json.loads(server_packet.m_data)
                            print(data, file=server_output)

                        if (server_packet.m_type == PacketTypes.END_RSP.value or (server_packet.m_type == PacketTypes.CONNECTION_RSP.value and \
                            data['Status'] == "Rejected")):
                            client.close()
                            if server_packet.m_type == PacketTypes.END_RSP.value:
                                del clients[client]
                            users = ''
                            for clientKey in clients:
                                users += clients[clientKey] + ' '
                                print(users, file=server_output)
                            return
                        elif server_packet.m_type == PacketTypes.SERVER_DOWN_RSP.value:
                                Exception("Server Down")
                except (KeyboardInterrupt, KeyError, Exception):
                    print("Exception in handle client", file=server_output)
                    q.put(Exception("handle_client"))
                    client.close()
                    sys.exit(0)
                except json.decoder.JSONDecodeError:
                    q.put(Exception("handle_client"))
                    client.close()
                    sys.exit(0)

        def get_stock_list():
            enginestk = create_engine('sqlite:///TradingBBD.db',connect_args={'check_same_thread': False})
            connstk = enginestk.connect()
            stockdf = pd.read_sql_query("SELECT * from Stocks", connstk)
            stklist = list(stockdf.Ticker)
            return stklist

        def generate_qty(number_of_qty):
            total_qty = 0
            list_of_qty = []
            for index in range(number_of_qty):
                qty = random.randint(1,101)
                list_of_qty.append(qty)
                total_qty += qty
            return np.array(list_of_qty)/total_qty

        def populate_order_table(symbols, start, end):
            #price_scale = 0.05
            #price_unit = 100
            global order_index
            global order_table
            global market_status

            if (market_status == "Open" or market_status == "Pending Closing"):

                return

            symbol_list = ','.join('"''"' + symbol + '"' for symbol in symbols)

            select_st = "SELECT * FROM FRE_Stocks WHERE date >= " + "\"" + start + "\"" + " AND date <= " + "\"" + end + "\"" + " AND symbol in (" + symbol_list + ");"
            result_set = engine.execute(select_st)
            result_df = pd.DataFrame(result_set.fetchall())
            result_df.columns = result_set.keys()

            order_table.drop(order_table.index, inplace=True)
            list_of_qty_map = {}
            max_number_orders = 0
            for index, stock_data in result_df.iterrows():

                if stock_data['open'] > high_price_min:
                    price_scale = high_price_scale
                else:
                    price_scale = low_price_scale

                list_of_qty = generate_qty(int((float(stock_data['high'])-float(stock_data['low']))/price_scale))
                list_of_qty_map[stock_data['symbol']] = list_of_qty
                if max_number_orders < len(list_of_qty):
                    max_number_orders = len(list_of_qty)

            if mutex.locked() is True:
                print("Is locked!")

            mutex.acquire()

```

```
            order_table.fillna(0)
            order_index = 0

            #print(list_of_qty_map)
            #print(max_number_orders)

            for index in range(0, max_number_orders-1, 2):
                for i, stock_data in result_df.iterrows():

                    if index >= len(list_of_qty_map[stock_data['symbol']]):
                        #print(i, index, list_of_qty_map[stock_data['symbol']])
                        continue

                    buy_price = float(stock_data['low']);
                    sell_price = float(stock_data['high'])
                    daily_volume = float(stock_data['volume'])
                    open_price = float(stock_data['open'])
                    close_price = float(stock_data['close'])

                    if stock_data['open'] > high_price_min:
                        price_scale = high_price_scale
                    else:
                        price_scale = low_price_scale

                    buy_price = open_price - (index+1) * price_scale * (random.randint(1, price_unit)/price_unit)
                    buy_price = float("{:.2f}".format(buy_price))
                    qty = float(list_of_qty_map[stock_data['symbol']][index])

                    order_index += 1
                    order_qty = int(qty*daily_volume*(random.randint(1, price_unit)/price_unit))
                    if index == 0:
                        order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), stock_data['symbol'], open_price, close_price,
                                       'Buy' if random.randint(1,11) % 2 == 0 else 'Sell', open_price, 0, order_qty,'Open Trade']
                    else:
                        order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), stock_data['symbol'], open_price, close_price,
                                       'Buy', buy_price, order_qty, order_qty,'New']

                    sell_price = open_price + (index+1) * price_scale * (random.randint(1, price_unit)/price_unit)
                    sell_price = float("{:.2f}".format(sell_price))
                    qty = float(list_of_qty_map[stock_data['symbol']][index])

                    order_index += 1
                    order_qty = int(qty*daily_volume*(random.randint(1, price_unit)/price_unit))
                    order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), stock_data['symbol'], open_price, close_price,
                                   'Sell', sell_price, order_qty, order_qty, 'New']

            order_table = order_table.sort_values(['Side', 'Symbol', 'Price', 'Qty'])

            mutex.release()
```

```

        print(order_table)


    def create_market_interest(symbols):

        global market_period
        global order_table
        global order_index
        #print(market_status, market_period, "No new market interest")

        while True:
            time.sleep(order_interval_time)
            if len(order_table) != 0 and market_status == 'Open':

                for i in range(len(symbols)):

                    # Some stocks may have fewer intraday data than others,

                    # it could be empty while other stocks still create intrday orders
                    if intraday_order_map[market_period][i].empty == True:
                        continue

                    symbol = symbols[i]

                    try:

                        mutex.acquire()

                        ### BUY logic
                        if ((order_table['Symbol'] == symbol) & (order_table['Side'] == 'Buy')).any():
                            mask = (order_table['Symbol'] == symbol) & (order_table['Side'] == 'Buy')

                            best_buy_index = order_table.loc[(mask.values), 'Price'].idxmax()
                            close_price = intraday_order_map[market_period][i].iloc[0]['close']
                            open_price = intraday_order_map[market_period][i].iloc[0]['open']

                            new_buy_price = intraday_order_map[market_period][i].iloc[0]['low']
                            new_buy_price = float("{:.2f}".format(new_buy_price))
                            new_buy_qty = intraday_order_map[market_period][i].iloc[0]['volume']/2
                            new_buy_qty = int(new_buy_qty * random.uniform(0,1))
                            order_index += 1

                            order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), symbol, open_price, close_price,
                                                   'Buy', new_buy_price, new_buy_qty, new_buy_qty, 'New']

                            print(order_table.loc[order_index])

                        #### Sell
```

```
                            if ((order_table['Symbol'] == symbol) & (order_table['Side'] == 'Sell')).any():

                                mask = (order_table['Symbol'] == symbol) & (order_table['Side'] == 'Sell')

                                best_sell_index = order_table.loc[(mask.values), 'Price'].idxmin()
                                close_price = intraday_order_map[market_period][i].iloc[0]['close']
                                open_price = intraday_order_map[market_period][i].iloc[0]['open']

                                new_sell_price = intraday_order_map[market_period][i].iloc[0]['high']
                                new_sell_price = float("{:.2f}".format(new_sell_price))
                                new_sell_qty = intraday_order_map[market_period][i].iloc[0]['volume']/2
                                new_sell_qty = int(new_sell_qty * random.uniform(0,1))
                                order_index += 1
                                order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), symbol, open_price, close_price,
                                                       'Sell', new_sell_price, new_sell_qty, new_sell_qty, 'New']

                                print(order_table.loc[order_index])

                            intraday_order_map[market_period][i].drop(intraday_order_map[market_period][i].index[0] ,inplace=True)

                            while ((order_table['Symbol'] == symbol) & (order_table['Qty'] != 0)).any():
                                buy_mask = (order_table['Symbol'] == symbol) & (order_table['Qty'] != 0) & (order_table['Side'] == 'Buy')
                                sell_mask = (order_table['Symbol'] == symbol) & (order_table['Qty'] != 0) & (order_table['Side'] == 'Sell')
                                buy_prices = order_table.loc[(buy_mask.values), 'Price']
                                sell_prices = order_table.loc[(sell_mask.values), 'Price']

                                if buy_prices.empty == False and sell_prices.empty == False:
                                    best_buy_index = buy_prices.idxmax()
                                    best_sell_index = sell_prices.idxmin()
                                    best_buy_price = order_table.loc[best_buy_index, 'Price']
                                    best_sell_price = order_table.loc[best_sell_index, 'Price']
                                    #TODO Avoid floating point issue
                                    if best_buy_price >= best_sell_price:

                                        if order_table.loc[best_buy_index, 'Qty'] == order_table.loc[best_sell_index, 'Qty']:
                                            order_table.loc[best_buy_index, 'Qty'] = 0
                                            order_table.loc[best_buy_index, 'Status'] = 'Filled'
                                            order_table.loc[best_sell_index, 'Qty'] = 0
                                            order_table.loc[best_sell_index, 'Status'] = 'Filled'

                                        elif order_table.loc[best_buy_index, 'Qty'] > order_table.loc[best_sell_index, 'Qty']:
                                            order_table.loc[best_buy_index, 'Qty'] -= order_table.loc[best_sell_index, 'Qty']
                                            order_table.loc[best_buy_index, 'Status'] = 'Partial Filled'
                                            order_table.loc[best_sell_index, 'Qty'] = 0
                                            order_table.loc[best_sell_index, 'Status'] = 'Filled'

                                        else:
                                            order_table.loc[best_sell_index, 'Qty'] -= order_table.loc[best_buy_index, 'Qty']
                                            order_table.loc[best_sell_index, 'Status'] = 'Partial Filled'
                                            order_table.loc[best_buy_index, 'Qty'] = 0
                                            order_table.loc[best_buy_index, 'Status'] = 'Filled'

                                    else:
                                        order_table = order_table.sort_values(['Side', 'Symbol', 'Price', 'Qty'])
                                        print(order_table)
                                        break

                                else:
                                    order_table = order_table.sort_values(['Side', 'Symbol', 'Price', 'Qty'])
                                    print(order_table)
                                    break

                            mutex.release()

                        #except (KeyboardInterrupt):
                        except Exception as e:
                            print("Except in create market interest")
                            print(e)
                            if mutex.locked() == True:
                                print("Still locked")
                                mutex.release()
                            #sys.exit(0)


        #TODO! The logic need to be optimized
        def close_trades(symbols):
            global order_index
            global order_table
            for symbol in symbols:
                side = 'Buy' if random.randint(1,11) % 2 == 0 else 'Sell'
                if ((order_table['Symbol'] == symbol) & (order_table['Side'] == side)).any():
                    mask = (order_table['Symbol'] == symbol) & (order_table['Side'] == side) & (order_table['Qty'] != 0)
                    if side == 'Buy':
                        buy_prices = order_table.loc[(mask.values), 'Price']
                        if buy_prices.empty == False:
                            best_buy_index = buy_prices.idxmax()
                            order_table.loc[best_buy_index, 'Qty'] = 0
                            order_table.loc[best_buy_index, 'Status'] = 'Close Trade'
                        else:
                            open_price = order_table.loc[order_index-1, 'Open']
                            close_price = order_table.loc[order_index-1, 'Close']
                            qty = order_table.loc[order_index, 'OrigQty']
                            order_index += 1
                            order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), symbol, open_price, close_price,
                                                   'Buy', close_price, 0, qty, 'Close Trade']
                    else:
                        sell_prices = order_table.loc[(mask.values), 'Price']
                        if sell_prices.empty == False:
                            best_sell_index = sell_prices.idxmin()
                            order_table.loc[best_sell_index, 'Qty'] = 0
                            order_table.loc[best_sell_index, 'Status'] = 'Close Trade'
                        else:
                            open_price = order_table.loc[order_index-1, 'Open']
                            close_price = order_table.loc[order_index-1, 'Close']
                            qty = order_table.loc[order_index, 'OrigQty']
                            order_index += 1
                            order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), symbol, open_price, close_price,
                                                   'Sell', close_price, 0, qty, 'Close Trade']

                order_table = order_table.sort_values(['Side', 'Symbol', 'Price', 'Qty'])
                print(order_table)

                #print(order_table, file = order_table_file)


        def update_market_status(status, day):
            global market_status
            global order_index
            global order_table
            market_status = status

            global symbols
            global market_periods
            global market_period

            print("day=", day)
            print(market_status, market_periods[day])
            market_period = market_periods[day]

            populate_order_table(symbols, market_periods[day], market_periods[day])
            market_status = 'Open'
            print(market_status)
            time.sleep(market_open_time)
            market_status = 'Pending Closing'
            print(market_status)
            time.sleep(market_pending_close_time)
            market_status = 'Market Closed'
            print(market_status)
            close_trades(symbols)
            time.sleep(market_close_time)

        def set_market_status(scheduler, time_in_seconds):
            value = datetime.datetime.fromtimestamp(time_in_seconds)
            print(value.strftime('%Y-%m-%d %H:%M:%S'))
            for day in range(total_market_days):
                scheduler.enter((market_close_time+market_open_time+market_pending_close_time)*day+1,1, update_market_status, argument=('Pending Open',day))
            scheduler.run()

        port = 6510
        buf_size = 4096
        fre_server = socket(AF_INET, SOCK_STREAM)
        fre_server.setsockopt(IPPROTO_TCP, TCP_NODELAY, True)
        print(gethostname())
        fre_server.bind((gethostbyname(""), port))

        location_of_pairs = 'csv/PairTrading.csv'
        stock_table_name = "FRE_Stocks"
        market_open_time = 75
        market_pending_close_time = 3
        market_close_time = 10
        order_interval_time = 1
        low_price_scale = 0.01
        high_price_scale = 1
        high_price_min = 1000
        price_unit = 100
        intraday_order_map = {}

        clients = {}

        if __name__ == "__main__":

            #server_output = open("server_output.txt", "w")
            server_output = sys.stderr

            q = queue.Queue()

            total_market_days = 4  # intrady data range is 30 days away from today
            order_index = 0

            symbols = get_stock_list()
            order_table_columns = ['OrderIndex', 'Symbol', 'Open', 'Close', 'Side', 'Price', 'Qty', 'OrigQty', 'Status']
            order_table = pd.DataFrame(columns=order_table_columns)
            order_table = order_table.fillna(0)
            order_table = pd.DataFrame(columns=order_table_columns)
            order_table = order_table.fillna(0)
            order_table['Price'] = order_table['Price'].astype(float)
            order_table['Open'] = order_table['Open'].astype(float)
            order_table['Close'] = order_table['Close'].astype(float)
            order_table['Qty'] = order_table['Qty'].astype(int)
            order_table['OrigQty'] = order_table['OrigQty'].astype(int)
```

```

        # USFederalHolidayCalendar has a bug, GoodFriday is not excluded
        us_bd = CustomBusinessDay(holidays=['2020-04-10'], calendar=USFederalHolidayCalendar())

        lastBusDay = datetime.datetime.today()
        if datetime.date.weekday(lastBusDay) == 5:      #if it's Saturday
            lastBusDay = lastBusDay - datetime.timedelta(days = 1) #then make it Friday
        elif datetime.date.weekday(lastBusDay) == 6:      #if it's Sunday
            lastBusDay = lastBusDay - datetime.timedelta(days = 2); #then make it Friday

        end_date = lastBusDay - datetime.timedelta(days = 1) # day before last trading day

        #end_date = datetime.datetime.today() - datetime.timedelta(days = 1) # yesterday
        start_date = end_date + datetime.timedelta(-total_market_days)

        #market_periods = pd.DatetimeIndex(pd.date_range(start=start_date.strftime("%Y-%m-%d"), end=end_date.strftime("%Y-%m-%d"), freq=us_bd)).strftime("%Y-%m-%d").tolist()
        trading_calendar = mcal.get_calendar('NYSE')
        market_periods = trading_calendar.schedule(start_date=start_date.strftime("%Y-%m-%d"), \
                                                   end_date=end_date.strftime("%Y-%m-%d")).index.strftime("%Y-%m-%d").tolist()

        print(market_periods)
        total_market_days = len(market_periods)  # Update for remove non-trading days

        #market_period_objects = pd.DatetimeIndex(pd.date_range(start=start_date.strftime("%Y-%m-%d"), end=end_date.replace(hour=23, minute=30).strftime("%Y-%m-%d %H:%M:%S"), freq=us_bd)).tolist()
        market_period_objects = trading_calendar.schedule(start_date=start_date.strftime("%Y-%m-%d"), end_date=end_date.strftime("%Y-%m-%d")).index.tolist()


        market_period_seconds = []
        for i in range(len(market_period_objects)):
            market_period_seconds.append(int(time.mktime(market_period_objects[i].timetuple())))   # As timestamp is 12am of each day
        market_period_seconds.append(int(time.mktime(market_period_objects[len(market_period_objects)-1].timetuple()))+24*3600)  # For last day intraday data
        #print(market_period_objects)
        #print(market_period_seconds)
        populate_intraday_stock_data(symbols, engine, market_period_seconds)

        intraday_order_map, stock_market_periods = populate_intraday_order_map(symbols, engine, market_periods)
        print(intraday_order_map)

        print(stock_market_periods)
        for value in stock_market_periods.values():
            if total_market_days > len(value):
                total_market_days = len(value)
```

```
                    market_periods = value

            print(market_periods)

            populate_stock_data(symbols, engine, stock_table_name, stock_market_periods)

            fre_server.listen(1)
            print("Waiting for client requests")
            try:
                scheduler = sched.scheduler(time.time, time.sleep)
                current_time_in_seconds = time.time()
                scheduler_thread = threading.Thread(target=set_market_status, args=(scheduler, current_time_in_seconds))
                #scheduler_thread.setDaemon(True)

                server_thread = threading.Thread(target=accept_incoming_connections, args=(q,))
                create_market_thread = threading.Thread(target=create_market_interest, args=(symbols,))
                #server_thread.setDaemon(True)

                scheduler_thread.start()
                server_thread.start()
                create_market_thread.start()

                error = q.get()
                q.task_done()
                if error is not None:
                    raise error

                scheduler_thread.join()
                server_thread.join()

                fre_server.close()
                sys.exit(0)

            except (KeyError, KeyboardInterrupt, SystemExit, Exception):
                print("Exception in main\n")
                fre_server.close()
                sys.exit(0)
```

5. network.py

```
from enum import Enum
import struct


class PacketTypes(Enum):
    CONNECTION_NONE = 0
    CONNECTION_REQ = 1  # request
    CONNECTION_RSP = 2  # response
    CLIENT_LIST_REQ = 3
    CLIENT_LIST_RSP = 4
    STOCK_LIST_REQ = 5
    STOCK_LIST_RSP = 6
    STOCK_REQ = 7
    STOCK_RSP = 8
    BOOK_INQUIRY_REQ = 9
    BOOK_INQUIRY_RSP = 10
    NEW_ORDER_REQ = 11
    NEW_ORDER_RSP = 12
    MARKET_STATUS_REQ = 13
    MARKET_STATUS_RSP = 14
    END_REQ = 15
    END_RSP = 16
    SERVER_DOWN_REQ = 17
    SERVER_DOWN_RSP = 18


class Packet:
    def __init__(self): ### three member
        self.m_type = 0
        self.m_msg_size = 0
        self.m_data_size = 0
        self.m_data = ""

    def __str__(self):
        return str(self.__class__) + ": " + str(self.__dict__) + "\n"

    def __repr__(self):
        return str(self.__class__) + ": " + str(self.__dict__) + "\n"

    def serialize(self):
        self.m_data_size = 12 + len(self.m_data)
        self.m_msg_size = self.m_data_size
        return self.m_type.to_bytes(4, byteorder='little') + \
               self.m_msg_size.to_bytes(4, byteorder='little') + \
               self.m_data_size.to_bytes(4, byteorder='little') + \
               bytes(self.m_data, 'utf-8')

    def deserialize(self, message):
        msg_len = len(message)
        msg_unpack_string = '<iii' + str(msg_len - 12) + 's' ### first three as int and following the string
        self.m_type, self.m_msg_size, self.m_data_size, msg_data = struct.unpack(msg_unpack_string, message)
        self.m_data = msg_data[0:self.m_data_size - 12].decode('utf-8')
        return message[self.m_data_size:]
```

6. main.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 19 23:16:42 2020

@author: 63438
"""

import json
import datetime as dt
from dateutil import tz
from dateutil.relativedelta import relativedelta
import urllib.request
import pandas as pd
import numpy as np
from sqlalchemy import Column, ForeignKey, Integer, Float, String
from sqlalchemy import create_engine
```

```python
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import inspect
import ClientFrontEnd
from BBD_client import Client_Manual



engine = create_engine('sqlite:///TradingBBD.db',connect_args={'check_same_thread': False})
conn = engine.connect()
conn.execute("PRAGMA foreign_keys = ON")

metadata = MetaData()
metadata.reflect(bind=engine)


end_date = dt.datetime.today().date()
start_date = end_date - relativedelta(weeks=1)

### This week we dont do FrontEnd but Manual connect
FrontEnd = True
Manual_Client_Connect = False

if __name__ == "__main__":
    if FrontEnd:
        app = ClientFrontEnd.FrontEnd(engine)
        app.run()
    if Manual_Client_Connect:
        Client_Manual()
```

7. base.html

```html
<!DOCTYPE html>
<html lang="en">
    <head>

        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

        <title>FRE Platform: {% block title %}{% endblock %}</title>
        <link href="https://v4-alpha.getbootstrap.com/dist/css/bootstrap.min.css" rel="stylesheet">
        <link href="https://v4-alpha.getbootstrap.com/examples/dashboard/dashboard.css" rel="stylesheet">

        <link href="/static/favicon.ico" rel="icon">
        <link href="/static/styles.css" rel="stylesheet">

        <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"></script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"></script>

    </head>

    <body>
```

```
        <nav class="navbar navbar-toggleable-md navbar-custom fixed-top navbar-dark border">
            <a class="navbar-brand" href="/"><span class="blue"><b>FRE </b></span><span class="green"><b>Platform</b></span></a>
            <button class="navbar-toggler navbar-toggler-right hidden-lg-up" type="button" data-toggle="collapse" data-target="#navbarsExampleDefault" aria-controls="navbarsExampleDefault" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbar">
            </div>

        </nav>

        {% if get_flashed_messages() %}
            <header>
                <div class="alert alert-primary border text-center" role="alert">
                    {{ get_flashed_messages() | join(" ") }}
                </div>
            </header>
        {% endif %}

        <div class="container-fluid">
            <div class="row">
                {% block content %} {% endblock %}
            </div>
        </div>

        <footer class="small text-center text-muted">
            Developed by <a href="https://engineering.nyu.edu/academics/departments/finance-and-risk-engineering">NYU FRE Department</a>. View <a href="https://engineering.nyu.edu/academics/departments/finance-and-risk-engineering">NYU FRE Terms of Use</a>.
        </footer>

    </body>

</html>
```

8. index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>index.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
```

```
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3><strong> '</strong></h3>
    <h3><strong>Bollinger Bands Trading Model</strong></h3>
    <div align="center">
        <table cellspacing="0" cellpadding="0" border="0" style="font-family:Georgia, Garamond, Serif;">
            <tbody>
                <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                <tr>
                    <td width="881" valign="top">
                        <p>
                            Bollinger Bands are a type of price envelope developed by John BollingerOpens.
                            (Price envelopes define upper and lower price range levels.) Bollinger Bands are envelopes plotted at a standard deviation level above and below a simple moving average of the price.
                            Because the distance of the bands is based on standard deviation, they adjust to volatility swings in the underlying price.
                        </p>
                    </td>
                </tr>
                <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                    <th width="881" valign="top">
                        <p>
                            <strong>Stock Selection</strong>
                        </p>
                    </th>
                </tr>
                <tr>
                    <td width="881" valign="top">
```

```
                        <p>
                            Within S&P500 component pool, select 10 stocks with highest 5-minute volatility over the last month of the historical data part.
                        </p>
                    </td>
                </tr>
                <tr>
                    <td width="881" valign="top">
                        <p>
                            Reason:
                            1.  S&P 500 components are with large size, high liquidity.
                            2.  Since using intraday data for generating trading signals, we check the vol among minutes over one month (a short period)
                            3.  Higher volatility may trigger signals more often, more signals can lead better test the efficacy of trading model (i.e. training params)

                        </p>
                    </td>
                </tr>
                <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                    <th width="881" valign="top">
                        <p>
                            <strong>Bollinger Bands Trading Logic</strong>
                        </p>
                    </th>
                </tr>
                <tr>
                    <td width="881" valign="top">
                        <p>
                            1) X: The H-period SMA. This serves as both the center of the BBs, and the baseline for determining the location of the two bands
                        </p>
                    </td>
                </tr>
                <tr>
                    <td width="881" valign="top">
                        <p>
                            2) Up : The upper band line that is k1 (default is 2) standard deviation from the H-minute SMA
                        </p>
                    </td>
                </tr>
                <tr>
                    <td width="881" valign="top">
                        <p>
                            3) Down : The lower band line that is k1 standard deviation from the H-period SMA
                        </p>
                    </td>
                </tr>
                <tr>
                    <td width="881" valign="top">
                        <p>
                            4) Open Trades:
                                <br>    • Open short position if Close >= Up, sell the stock
```

```
                                        <br>     • Open long position if Close <= Down, buy the stock
                                </p>
                            </td>
                        </tr>
                        <td width="881" valign="top">
                                <p>
                                    5) Close Trades:
                                        <br>     • Close the open positions when Close return within Up and Down </br>
                                </p>
                            </td>
                        </tr>
                        <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                            <th width="881" valign="top">
                                <p>
                                    <strong>Back Testing</strong>
                                </p>
                            </th>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    Use the intraday 1m-price data from the recent 1 week, doing P&L calculation
                                </p>
                            </td>
                        </tr>
                        <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                            <th width="881" valign="top">
                                <p>
                                    <strong>Probation Testing</strong>
                                </p>
                            </th>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    Any two trading dates before current date could be used for conducting BB trading model
                                </p>
                            </td>
                        </tr>
                    </tbody>
                </table>
            </div>

        </main>
        <br></br>
        <br></br>

        {% endblock %}
          </body>
        </html>
```

9. stock_selection.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>stock_selection.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3>Stock Selection within S&P500</h3>
    <div class="table-responsive">
        <table class="table table-striped">
            <thead>
            <tr>
                <th style="text-align:center">Symbol</th>
                <th style="text-align:center">Volatility</th>
            </tr>
            </thead>
            <tbody>
            {% for stock in stock_list %}
            <tr>
```

```
                <td> {{stock.symbol}}</td>
                <td> {{stock['std']}}</td>
            </tr>
            {% endfor %}
            </tbody>
        </table>
    </div>
</main>
{% endblock %}
  </body>
</html>
```

10. train_model.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>stock_selection.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
```

```
    <h3>'</h3>
    <h3>Training Parameters for Selected Stocks</h3>
    <div class="table-responsive">
        <table class="table table-striped">
            <thead>
            <tr>
                <th style="text-align:center">Ticker</th>
                <th style="text-align:center">H</th>
                <th style="text-align:center">K1</th>
                <th style="text-align:center">Notional</th>
                <th style="text-align:center">PnL_in_Training</th>
            </tr>
            </thead>
            <tbody>
            {% for stock in train_list %}
            <tr>
                <td> {{stock.Ticker}}</td>
                <td> {{stock.H}}</td>
                <td> {{stock.K1}}</td>
                <td> {{stock.Notional}}</td>
                <td> {{stock.Profit_Loss_in_Training}}</td>

            </tr>
            {% endfor %}
            </tbody>
        </table>
    </div>
</main>
{% endblock %}
  </body>
</html>
```

11. back_test_result.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>pair_trade_back_test_result.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3>Back Test Analysis</h3>
    <div class="table-responsive">
        <table class="table table-striped">
            <thead>
            <tr>
                <th style="text-align:center">Ticker</th>
                <th style="text-align:center">H</th>
                <th style="text-align:center">K1</th>
                <th style="text-align:center">Notional</th>
                <th style="text-align:center">Profit</th>
            </tr>
            </thead>
            <tbody>
            {% for trade in bt_list %}
            <tr>
                <td> {{trade.Ticker}}</td>
                <td> {{trade.H}}</td>
                <td> {{trade.K1}}</td>
                <td> {{trade.Notional}}</td>
                <td> {{trade.Profit_Loss}}</td>
            </tr>
            {% endfor %}
            </tbody>
            <tfoot>
                <tr>
                    <td colspan="5"></td>
                    <td>{{ total }}</td>
                </tr>
            </tfoot>
        </table>
    </div>
</main>
{% endblock %}
  </body>
</html>
```

12. probation_test.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>pair_trade_probation_test.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3>Start Probation Testing</h3>
    <hr>
    <div class="form-group">
        <form action="/Probation_test" id="buy" method="POST">
            <p>Start Date <input type = "text" name = "Start Date" placeholder="yyyy-mm-dd"></p>
            <p>End Date <input type = "text" name = "End Date" placeholder="yyyy-mm-dd"></p>
            <p><input type = "submit" value = "submit" /></p>
        </form>
    </div>
</main>
```

```
{% endblock %}
  </body>
</html>
```

13. probation_test_result.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>pair_trade_probation_test_result.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3>Probation Test Analysis</h3>
    <div class="table-responsive">
        <table class="table table-striped">
            <thead>
            <tr>
                <th style="text-align:center">Ticker</th>
                <th style="text-align:center">H</th>
        <th style="text-align:center">K1</th>
        <th style="text-align:center">Notional</th>
        <th style="text-align:center">Profit</th>
      </tr>
      </thead>
      <tbody>
      {% for trade in trade_list %}
      <tr>
        <td> {{trade.Ticker}}</td>
        <td> {{trade.H}}</td>
        <td> {{trade.K1}}</td>
        <td> {{trade.Notional}}</td>
        <td> {{trade.Profit_Loss}}</td>
      </tr>
      {% endfor %}
      </tbody>
      <tfoot>
        <tr>
          <td colspan="5"></td>
          <td>{{ total }}</td>
        </tr>
      </tfoot>
    </table>
  </div>
</main>
{% endblock %}
 </body>
</html>
```

14. start_trading.html

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta name="viewport" content="width=device-width">
  <title>start_trading.html</title>
  <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
  <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
  <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>  <style>body { padding: 5px !important; }</style>
 </head>
 <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
  <ul class="nav nav-pills flex-column">
    <li class="nav-item">
      <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="/Train_model">Train Model</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="/Back_test">Back Testing</a>
```

```
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3><strong>Simulated Trading</strong></h3>
    <div align="center">
        <table cellspacing="0" cellpadding="0" border="0" style="font-family:Georgia, Garamond, Serif;">
            <tbody>
                <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                <tr>
                    <td width="881" valign="top">
                        <p>
                            The simulated market is based on the latest 30 trading days of market data, for which the intraday trading data are available.
                            Therefore, the market simulation will stop after 30 simulated trading days.
                        </p>
                    </td>
                </tr>
                <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                    <th width="881" valign="top">
                        <p>
                            <strong>Definition of Trading Day and Market Status</strong>
                        </p>
                    </th>
                </tr>
                <tr>
                    <td width="881" valign="top">
                        <p>
                            1) Each trading day is 60 seconds, followed by 10 seconds of market close, and then a new trading day.
                        </p>
                    </td>
                </tr>
                <tr>
                    <td width="881" valign="top">
                        <p>
                            2) Market open time = 50 seconds, status: Open.
                        </p>
                    </td>
                </tr>
                <tr>
                    <td width="881" valign="top">
```

```
                        <p>
                            3) Market pending close time = 10 seconds, status: Pending Closing.
                        </p>
                    </td>
                </tr>
                <tr>
                    <td width="881" valign="top">
                        <p>
                            4) Market close time = 10 seconds, status: Market Closed.
                        </p>
                    </td>
                </tr>
                <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                    <th width="881" valign="top">
                        <p>
                            <strong>Daily Order Book - Historical Daily Market Data, created before market open</strong>
                        </p>
                    </th>
                </tr>
                <tr>
                    <td width="881" valign="top">
                                <p>
                                    1) Number of orders = (high_price - low_price)/price scale,
                                    price_scale is either 0.05 or 5 based stock price >= 1000 or not.
                                </p>
                            </td>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    2) buy_price = open_price - price_scale * rand().
                                </p>
                            </td>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    3) sell_price = open_price + price_scale * rand().
                                </p>
                            </td>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    3) quantity = randomized, but sum of all the orders = daily volume.
                                </p>
                            </td>
                        </tr>
                        <td width="881" valign="top">
                                <p>
                                    4) The book interested are populated before market open.
                                </p>
                            </td>
                        </tr>
                        <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                            <th width="881" valign="top">
                                <p>
                                    <strong>Intraday Market Interests - Daily Intraday Market Data, during market open</strong>
                                </p>
                            </th>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    1) Buy interests: Evey 5 min low price, 1/2 of the volume * rand().
                                </p>
                            </td>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    2) Sell interests: Evey 5 min high price, 1/2 of the volume * rand().
                                </p>
                            </td>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    3) Any crossed interests will be traded.
                                </p>
                            </td>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    4) The order book is sorted accorinding Side, Symbol, Price and Quantity.
                                </p>
                            </td>
                        </tr>
                        <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                            <th width="881" valign="top">
                                <p>
                                    <strong>Close Trade - either a buy or sell order is filled</strong>
                                </p>
                            </th>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    1) If buy side or sell side book is not empty, a best buy or best sell is filled.
                                </p>
                            </td>
                        </tr>
                        <tr>
```

```
                            <td width="881" valign="top">
                                <p>
                                    2) If buy side or sell side is empty, the close trade for a buy or sell will be executed at daily market data closing price.
                                </p>
                            </td>
                        </tr>
                        <tr style="font-variant:small-caps;font-style:normal;color:black;font-size:20px;">
                            <th width="881" valign="top">
                                <p>
                                    <strong>Execution logic</strong>
                                </p>
                            </th>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    1) If market is closed, new orders will be rejected.
                                </p>
                            </td>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    2) While market is open or in pending closing.
                                </p>
                                <ul>
                                    <li>Market orders - always filled from best price.</li>
                                    <li>Limit orders - will be filled at equal or better price.</li>
                                    <li>A new limit order or a limit order with better price sweep books until it is filled or counter side of the book is empty.</li>
                                </ul>
                            </td>
                        </tr>
                        <tr>
                            <td width="881" valign="top">
                                <p>
                                    3) Responses to new orders.
                                </p>
                                <ul>
                                    <li>Order Fill</li>
                                    <li>Order Partial Fill</li>
                                    <li>Order Reject</li>
                                </ul>
                            </td>
                        </tr>
                    </tbody>
                </table>
            </div>
            <div class="form-group">
                <form action="/start_trading" id="buy" method="POST">
                    <dl>
                    <dd><input type = "submit" name = "Auto Trading" value = "Auto Trading" /></dd>
                    </dl>
                </form>
            </div>
        </main>
        <br></br>
        <br></br>

        {% endblock %}
          </body>
        </html>
```

15. auto_trading.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>auto_trading.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
  <h3><strong>Auto Trading Result</strong></h3>
  <div class="table-responsive">
    <table class="table table-striped">
      <thead>
      <tr>
        <th style="text-align:center">OrderIndex</th>
        <th style="text-align:center">Status</th>
        <th style="text-align:center">Symbol</th>
        <th style="text-align:center">Type</th>
        <th style="text-align:center">Side</th>
        <th style="text-align:center">Price</th>
        <th style="text-align:center">Qty</th>
        <th style="text-align:center">ServerOrderID</th>
      </tr>
      </thead>
      <tbody>
      {% for trade in trading_results %}
        <tr>
          <td> {{trade.OrderIndex}}</td>
          <td> {{trade.Status}}</td>
          <td> {{trade.Symbol}}</td>
          <td> {{trade.Type}}</td>
          <td> {{trade.Side}}</td>
          <td> {{trade.Price}}</td>
          <td> {{trade.Qty}}</td>
          <td> {{trade.ServerOrderID}}</td>
        </tr>
      {% endfor %}
      </tbody>
      <tfoot>
        <tr>
          <td colspan="7"></td>
          <td>{{ total }}</td>
        </tr>
      </tfoot>
    </table>
  </div>
  <div class="form-group">
    <form action="/start_trading" id="buy2" method="POST">
      <dl>
      <dd><input type = "submit" name = "Analysis" value = "Analysis" /></dd>
      </dl>
    </form>
  </div>
  <hr>
</main>
{% endblock %}
 </body>
</html>
```

16. auto_trading_analysis.html

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta name="viewport" content="width=device-width">
  <title>auto_trading_analysis.html</title>
```

```
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3><strong>Auto Trading Analysis</strong></h3>
    <div  class="table-responsive">
        <table class="table table-striped">
        <thead>
            <tr>
                <th style="text-align:center">Symbol</th>
                <th style="text-align:center">ProfitLoss</th>
            </tr>
            </thead>
            <tbody>
            {% for key, value in trading_results.items() %}
                <tr>
                    <th style="text-align:center">{{ key }}</th>
                    <td style="text-align:center">{{ value }}</td>
                </tr>
            {% endfor %}
            </tbody>
        </table>
    </div>
    <hr>
</main>
```

```
{% endblock %}
  </body>
</html>
```

17. manual_trading.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>manual_trading.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3><strong>Manual Trading</strong></h3>
    <hr>
    <div class="form-group">
        <form action="/manual_trading" id="buy" method="POST">
            <dl>
            <dt>OrderId</dt>
            <dd><input type = "number" name = "OrderId" min="1", step="1"></dd>

            <dt>Symbol</dt>
            <dd><input type = "text" name = "Symbol"></dd>
            <dt>Side</dt>
            <dd><input type = "text" name = "Side"></dd>
            <dt>Price</dt>
            <dd><input type = "number" name = "Price" min="0", step="0.01"></dd>

            <dt>Quantity</dt>
            <dd><input type = "number" name = "Quantity" min="10", step="1"></dd>
            <dd><input type = "submit" value = "submit" /></dd>
            </dl>
        </form>
    </div>
</main>

{% endblock %}
  </body>
</html>
```

18. trading_results.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>trading_results.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3><strong>Trading Result</strong></h3>
    <div align="center" class="table-responsive">
        <table class="table table-striped">
        <table border = 1>
            {% for key, value in trading_results.items() %}
                <tr>
                    <th>{{ key }}</th>
                    <td>{{ value }}</td>
                </tr>
            {% endfor %}
            </tbody>
        </table>
    </div>
    <hr>
</main>
{% endblock %}
  </body>
</html>
```

19. client_down.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Style-Type" content="text/css" />
    <meta name="viewport" content="width=device-width">
    <title>client_down.html</title>
    <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
    <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
    <script type="text/javascript" src="/library/js/headscripts.js"></script>
<script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
  </head>
  <body>
{% extends "base.html" %}
{% block content %}
<nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
    <ul class="nav nav-pills flex-column">
        <li class="nav-item">
            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Train_model">Train Model</a>
        </li>
        <li class="nav-item">
```

```
            <a class="nav-link" href="/Back_test">Back Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/Probation_test">Probation Testing</a>
        </li>
        <li class="nav-item">
            <a class="nav-link active" href="/start_trading">Start Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/manual_trading">Manual Trading</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/client_down">Client Down</a>
        </li>
    </ul>
</nav>

<main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
    <h3>'</h3>
    <h3><strong>Client Down</strong></h3>
    <div align="center" class="table-responsive">
        <table class="table table-striped">
        <table border = 1>
            <tbody>
            {% for key, value in server_response.items() %}
                <tr>
                    <th>{{ key }}</th>
                    <td>{{ value }}</td>
                </tr>
            {% endfor %}
            </tbody>
        </table>
    </div>
    <hr>
</main>
{% endblock %}

  </body>
</html>
```