

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

Acknowledgements.....	4
List of Abbreviations	5
1. Introduction.....	6
2. Bollinger Band Trading Model.....	6
2.1 Stocks Pool Selection:	6
2.2 Base Trading Model: Bollinger Bands (BB)	7
2.3 Market Data Feed Description.....	9
2.4 Parameters Training.....	11
2.5 Back Test for BB Trading Strategy	12
3. Front-End Web Page Design and Development.....	14
3.1 Flask description	14
3.2 Route in each module.....	14
3.3 Render Template and Request	18
4. Trading in Simulated Market.....	20
4.1 Simulated Market Description.....	20
4.2 Manual Trading.....	21
4.3 Auto Trading in Simulated Market	24
4.3.1 Important Logic to realize the BBD Trading Logic	24
4.3.2 Process description:	27
4.3.3 Flow Chart	28
4.4 Auto Trading Screenshot and Result	28
5. Future Development	30
References.....	32
Appendix A – Source Code	33
1. BBDModel.py.....	33
2. BBD_client.py	42
3. ClientFrontEnd.py.....	54
4. FRE_server5.py	57

5. network.py	75
6. main.py.....	76
7. base.html	77
8. index.html	78
9. stock_selection.html	82
10. train_model.html.....	83
11. back_test_result.html	84
12. probation_test.html	86
13. probation_test_result.html	87
14. start_trading.html.....	88
15. auto_trading.html.....	93
16. auto_trading_analysis.html.....	94
17. manual_trading.html.....	96
18. trading_results.html	97
19. client_down.html	98

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)

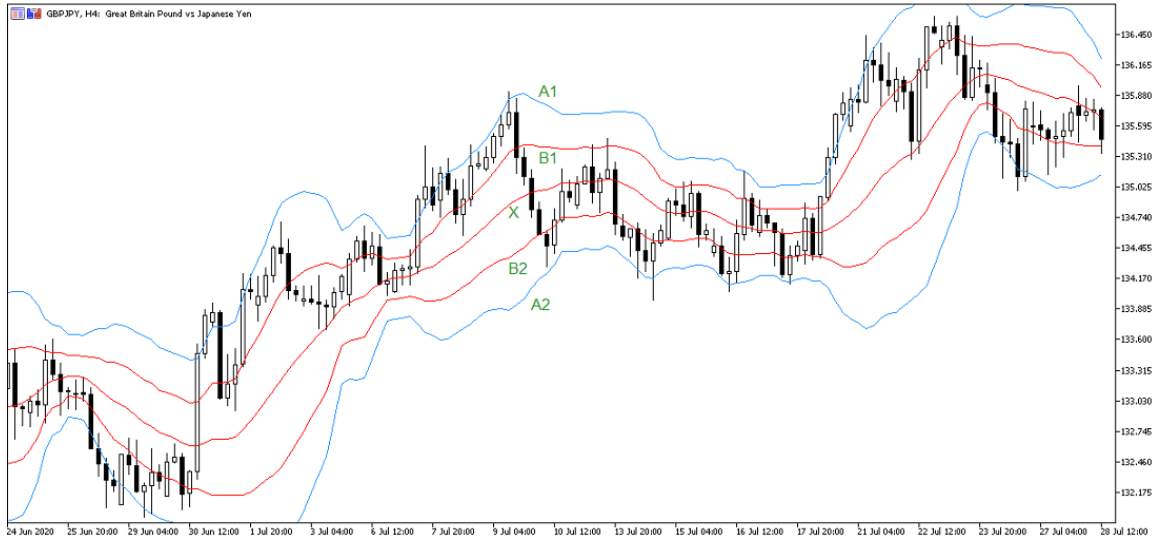


Figure 1 Bollinger Bands

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 k_1 (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 k_1 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 $\text{close} < A2$
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 $\text{close} > A1$

According to Figure 1, the price can fall into these 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 $\text{Close} \geq \text{Up}$, sell the stock
 - Open long position if $\text{Close} \leq \text{Down}$, 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**”.

```
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 + '\n'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_D8BD(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 1 week data. Eventually we store the back test PnL & return into stocks table. As you can see in the screenshot:

Table: Stocks

	Ticker	H	K1	Notional	Profit_Loss_in_Training	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	982.894400000002	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:

Table: Strategy

	Strategy	Notional	PnL_in_Training	Return_in_Training	/Return_in_Training	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")
```

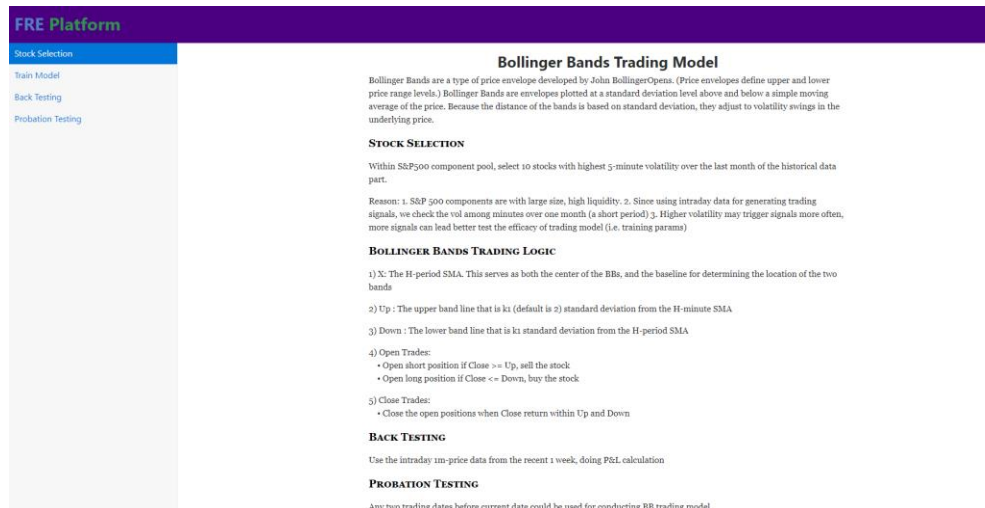
The **app.route** decorator decorates the first view function; it can specify one of the routes used to access the application. Any view we specify must be decorated by this “app.route” to be a functional part of the web page. In this project, we have many templates for us to render our result into, 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.



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.

```
@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)
```


FRE Platform

Stock Selection

Train Model

Back Testing

Probation Testing

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)
```

FRE Platform

Stock Selection

Train Model

Back Testing

Probation Testing

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))
```

Stock Selection			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 F&E Department. View NYU F&E 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")
```

The screenshot displays the FRE Platform web application. The left sidebar contains navigation links: Stock Selection, Train Model, Back Testing, Probation Testing, Start Trading, Manual Trading, and Client Down. The main content area is titled 'Start Probation Testing' and includes input fields for 'Start Date' (2020-12-02) and 'End Date' (2020-12-06), along with a 'submit' button. Below the form, a table titled 'Probation Test Analysis' shows the results of the test. The table has columns for Ticker, H, K1, Notional, and Profit. The data is as follows:

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

The total profit is \$7,980.95. The footer of the page states: '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-like 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 = $(\text{high_price} - \text{low_price}) / \text{price_scale}$, price_scale is either 0.05 or 5 based stock price ≥ 1000 or not.
- 2) $\text{buy_price} = \text{open_price} - \text{price_scale} * \text{rand}()$.
- 3) $\text{sell_price} = \text{open_price} + \text{price_scale} * \text{rand}()$.
- 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: Every 5 min low price, $1/2$ of the volume $* \text{rand}()$.
- 2) Sell interests: Every 5 min high price, $1/2$ of the volume $* \text{rand}()$.
- 3) Any crossed interests will be traded.
- 4) The order book is sorted according 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):

FRE Platform

Stock Selection
Train Model
Back Testing
Probation Testing
Start Trading
Manual Trading
Client Down

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 = $(\text{high_price} - \text{low_price}) / \text{price_scale}$, price_scale is either 0.05 or 5 based stock price >= 1000 or not.
- 2) $\text{buy_price} = \text{open_price} - \text{price_scale} * \text{rand}()$.
- 3) $\text{sell_price} = \text{open_price} + \text{price_scale} * \text{rand}()$.
- 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: Every 5 min low price, $1/2$ of the volume * $\text{rand}()$.
- 2) Sell interests: Every 5 min high price, $1/2$ of the volume * $\text{rand}()$.
- 3) Any crossed interests will be traded.
- 4) The order book is sorted according 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.

FRE Platform

Stock Selection

Train Model

Back Testing

Probation Testing

Start Trading

Manual Trading

Client Down

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':
                ## calculate 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: 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: sum(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(S)
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(S)
```

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.

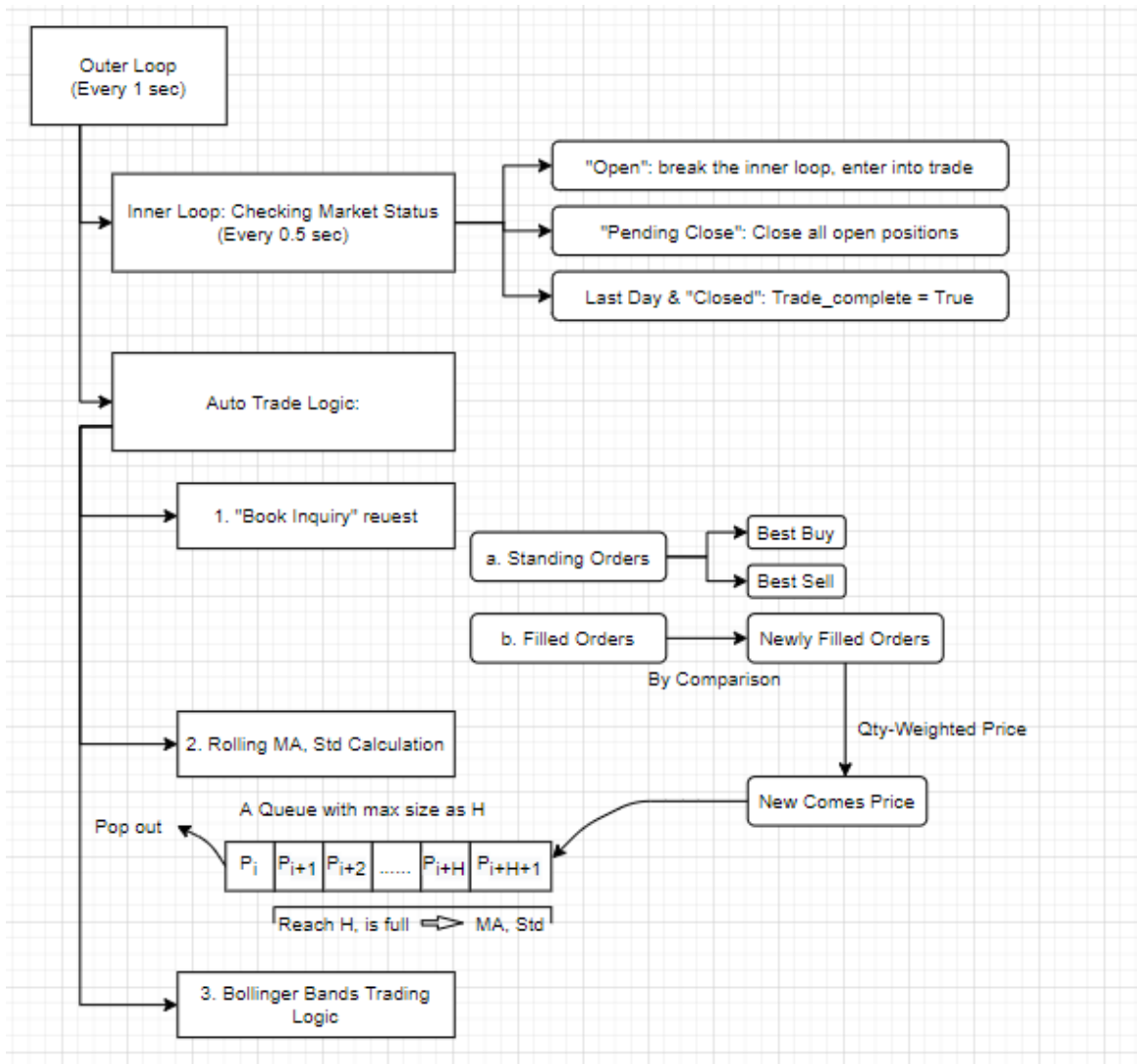
```
class Trade():
    def __init__(self, Ticker_, Orders_Responses_):    ## open trade
        Orders_Responses_ = [response for response in Orders_Responses_ if response['Symbol'] == Ticker_]
        self.Ticker = Ticker_
        self.Position = 1 if Orders_Responses[0]['Side'] == 'Buy' else -1
        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 position: ", self.Position, "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.Position == 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.Position == 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.

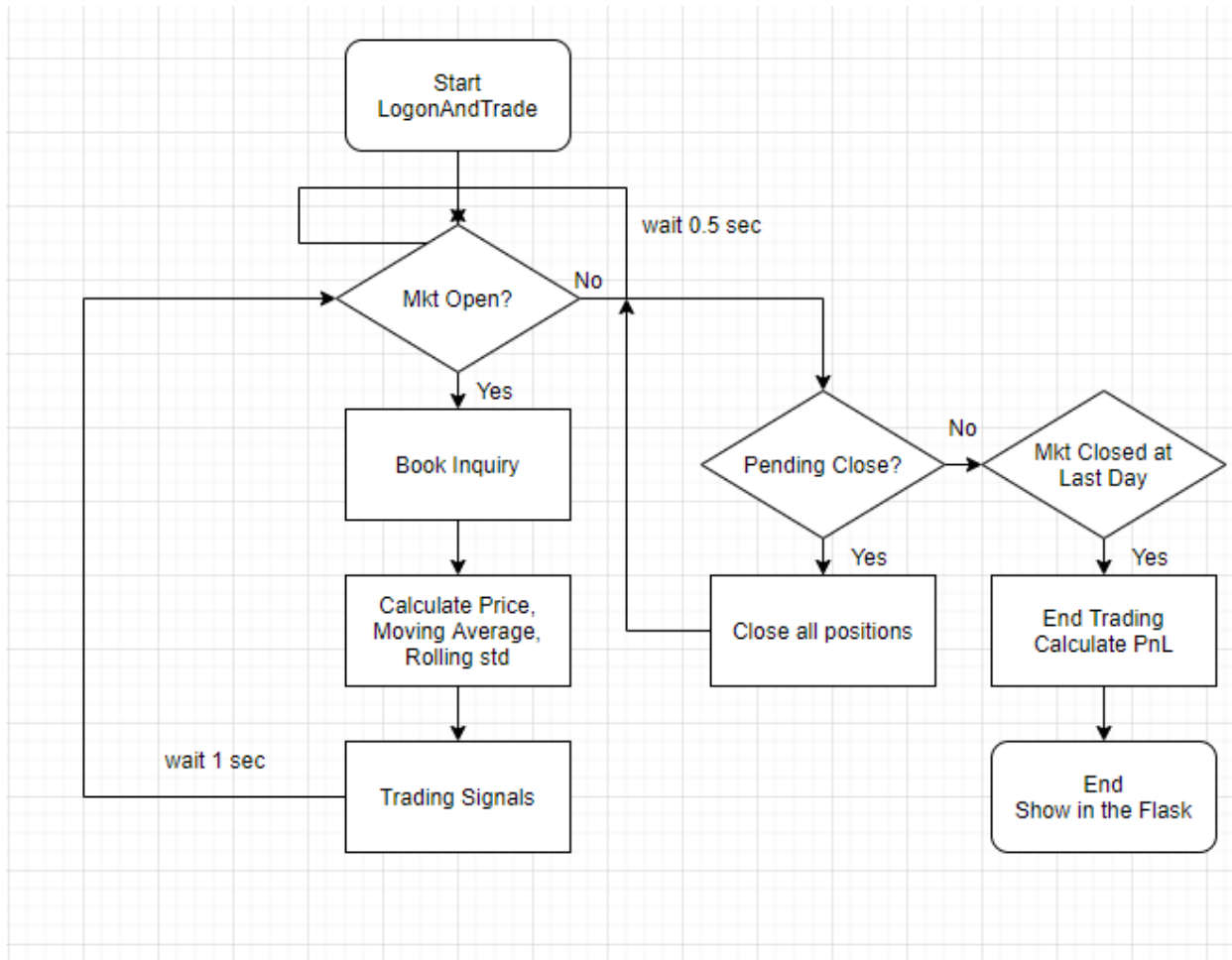
```
## PnL Calculation Logic 2
PnL_dict = {}
for stk in StockInfoDict:
    stkbuy_order = [order for order in client_config.orders if (order['Symbol'] == stk)&(order['Side'] == 'Buy')]
    stkbuy_price = [order['Price'] for order in stkbuy_order]
    stkbuy_qty = [int(order['Qty']) for order in stkbuy_order]
    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 according 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

Auto Trading

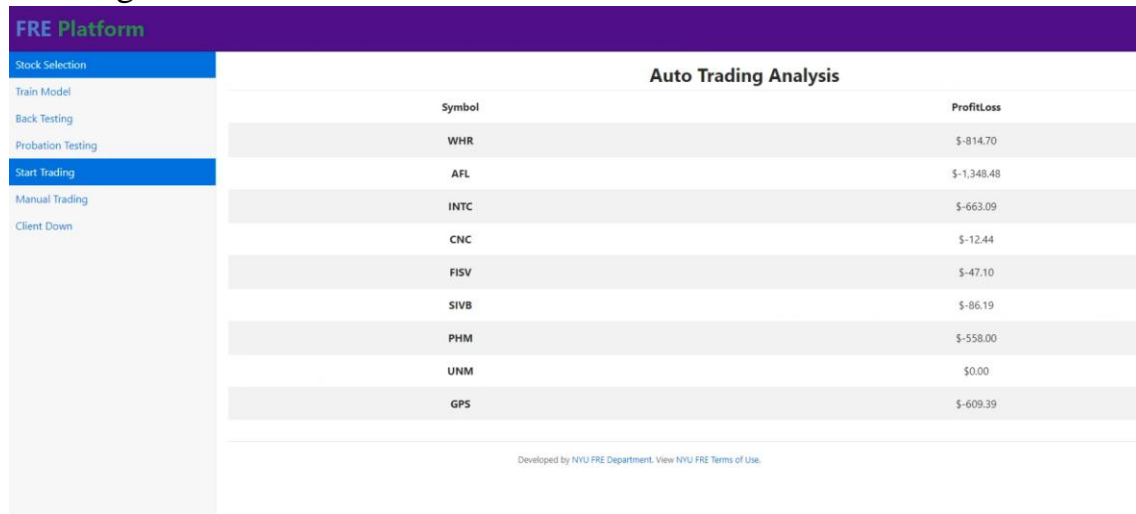
When the simulated market ends, the auto trading process will stop and return the results as below, basically the orders information with Total PnL at the bottom of the Page.

FRE Platform								
Stock Selection	client1_21	Order Partial Fill	GPS	Mkt	Buy	21.38	990	Srv_2020-12-07_2808
Train Model	client1_21	Order Fill	GPS	Mkt	Buy	21.39	4206	Srv_2020-12-07_2844
Back Testing	client1_22	Order Partial Fill	CNC	Mkt	Sell	64.28	143	Srv_2020-12-07_2833
Probation Testing	client1_22	Order Fill	CNC	Mkt	Sell	64.26	1585	Srv_2020-12-07_331
Start Trading	client1_23	Order Fill	GPS	Mkt	Sell	21.42	5196	Srv_2020-12-07_2897
Manual Trading	client1_24	Order Partial Fill	PHM	Mkt	Sell	42.16	1296	Srv_2020-12-07_337
Client Down	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
								\$-47139.39

Analysis

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

```
1. # -*- coding: utf-8 -*-
2. """
3. Created on Mon Oct 12 20:11:29 2020
4.
5. @author: 63438
6. """
7.
8. import json
9. import datetime as dt
10. from dateutil import tz
11. from dateutil.relativedelta import relativedelta
12. import urllib.request
13. import pandas as pd
14. import numpy as np
15.
16. from sqlalchemy import Column, ForeignKey, Integer, Float, String
17. from sqlalchemy import create_engine
18. from sqlalchemy import MetaData
19. from sqlalchemy import Table
20. from sqlalchemy import inspect
21.
22.
23.
24. engine = create_engine('sqlite:///TradingBBD.db', connect_args={'check_same_thread': False})
25. conn = engine.connect()
26. conn.execute("PRAGMA foreign_keys = ON")
27.
28.
29. metadata = MetaData()
30. metadata.reflect(bind=engine)
31.
32.
33. end_date = dt.datetime.today().date()
34. start_date = end_date - relativedelta(weeks=1)
35.
36.
37. def EDTtoUnixTime(EDTdatetime):
38.     utcTime = EDTdatetime.replace(tzinfo = tz.gettz('EDT')).astimezone(tz=dt.timezone.utc)
39.     unixTime = utcTime.timestamp()
40.     return str(int(unixTime))
41.
42. def get_sp500_component():
43.     url = 'https://eodhistoricaldata.com/api/fundamentals/GSPC.INDX?api_token=5ba84ea974ab42.45160048'
44.     with urllib.request.urlopen(url) as req:
45.         data = json.load(req)
46.         comdf = pd.DataFrame(data['Components']).T
47.         tickerList = list(comdf['Code'])
48.     return tickerList[:30] ### for demo, otherwise it is time consuming
49.
50.
51. def get_daily_data(symbol,
52.                     start=start_date,
```

```
53.             end=end_date,
54.             requestURL='https://eodhistoricaldata.com/api/eod/',
55.             apiKey='5ba84ea974ab42.45160048'):
56.     symbolURL = str(symbol) + '.US?'
57.     startURL = 'from=' + str(start)
58.     endURL = 'to=' + str(end)
59.     apiKeyURL = 'api_token=' + apiKey
60.     completeURL = requestURL + symbolURL + startURL + '&' + endURL + '&' + apiKeyURL + '&period=d&fmt=json'
61.     print('Possessing daily data of stock:' + symbol)
62.     with urllib.request.urlopen(completeURL) as req:
63.         data = json.load(req)
64.         return data
65.
66. def get_intraday_data(symbol,
67.                       interval='5m',
68.                       requestURL='https://eodhistoricaldata.com/api/intraday/',
69.                       apiKey='5ba84ea974ab42.45160048',
70.                       start=None): ### start in the format of EDT datetime
71.     symbolURL = str(symbol) + '.US?'
72.     apiKeyURL = 'api_token=' + apiKey
73.     if start != None:
74.         starttimestamp = EDTtoUnixTime(start)
75.         completeURL = requestURL + symbolURL + apiKeyURL + '&interval=' + interval + '&fmt=json' + '&from=' + starttimestamp
76.     else:
77.         completeURL = requestURL + symbolURL + apiKeyURL + '&interval=' + interval + '&fmt=json'
78.     print('Possessing data of stock:' + symbol)
79.     with urllib.request.urlopen(completeURL) as req:
80.         data = json.load(req)
81.         return data
82.
83. def create_stocks_table(table_name, metadata, engine):
84.     table = Table(table_name, metadata,
85.                   Column('Ticker', String(50), primary_key=True, nullable=False),
86.                   Column('H', Integer, nullable=False),
87.                   Column('K1', Float, nullable=False),
88.                   Column('Notional', Float, nullable=False),
89.                   Column('Profit_Loss_in_Training', Float, nullable=False),
90.                   Column('Return_in_Training', Float, nullable=False),
91.                   Column('Profit_Loss', Float, nullable=False),
92.                   Column('Return', Float, nullable=False),
93.                   extend_existing=True)
94.     table.create(engine)
95.
96.
97. def create_price_table(table_name, metadata, engine):
98.     tables = metadata.tables.keys()
99.     if table_name not in tables:
100.         foreign_key = 'Stocks.Ticker'
101.         table = Table(table_name, metadata,
102.                       Column('Symbol', String(50), ForeignKey(foreign_key), primary_key=True, nullable=False),
103.                       Column('DateTime', String(50), primary_key=True, nullable=False),
104.                       Column('Open', Float, nullable=False),
105.                       Column('High', Float, nullable=False),
106.                       Column('Low', Float, nullable=False),
```

```
107.         Column('Close', Float, nullable=False),
108.         Column('Volume', Integer, nullable=False))
109.     table.create(engine)
110.
111.     def create_strategy_table(table_name, metadata, engine):
112.         table = Table(table_name, metadata,
113.             Column('Strategy', String(50), primary_key=True, nullable=False),
114.             Column('Notional', Float, nullable=False),
115.             Column('PnL_in_Training', Float, nullable=False),
116.             Column('Return_in_Training', Float, nullable=False),
117.             Column('SPY_Return_in_Training', Float, nullable=False),
118.             Column('PnL', Float, nullable=False),
119.             Column('Strategy_Return', Float, nullable=False),
120.             Column('SPY_Return', Float, nullable=False),
121.             extend_existing=True)
122.         table.create(engine)
123.
124.     def clear_a_table(table_name, metadata, engine):
125.         conn = engine.connect()
126.         table = metadata.tables[table_name]
127.         delete_st = table.delete()
128.         conn.execute(delete_st)
129.
130.
131.     def execute_sql_statement(sql_stmt, engine, change=False):
132.         if change:
133.             engine.execute(sql_stmt)
134.         else:
135.             result_set = engine.execute(sql_stmt)
136.             result_df = pd.DataFrame(result_set.fetchall())
137.             if not result_df.empty:
138.                 result_df.columns = result_set.keys()
139.             return result_df
140.
141.     def stock_selection(stock_list, interval='5m', number_of_stocks=10):
142.         std_resultdf = pd.DataFrame(index=stock_list)
143.         std_resultdf['std'] = 0.0
144.         for stk in stock_list:
145.             try:
146.                 stk_data = pd.DataFrame(get_intraday_data(stk, interval))
147.                 std = stk_data.close.pct_change().shift(-1).std()
148.                 std_resultdf.loc[stk, 'std'] = std
149.                 print('Volatility of return over stock: ' + stk + ' is: ' +
150.                     str(std))
151.             except:
152.                 print('Cannot get data of Stock: ' + stk)
153.                 stock_selected = list(std_resultdf['std'].sort_values().index[-
154.                     number_of_stocks:])
155.                 selected_df = std_resultdf['std'].sort_values()[-
156.                     number_of_stocks:]
157.                 return stock_selected, selected_df
158.
159.     def populate_price_data(stockList, engine, table_name, start_date, end_date, interval='1m'):
160.         column_names = ['Symbol', 'DateTime', 'Open', 'High', 'Low', 'Close', 'Volume']
161.         price_data = []
162.         mkt_opentime = dt.datetime.strptime('09:30', '%H:%M').time()
```

```

161.         mkt_closetime = dt.datetime.strptime('16:00','%H:%M').time()
162.         start_datetime = dt.datetime(start_date.year,start_date.month,start_
            date.day,9,30)
163.         for stk in stockList:
164.             raw_data = get_intraday_data(stk,interval,start=start_datetime)
165.             for stock_data in raw_data:
166.                 price_data.append([stk, stock_data['datetime'], stock_data['
                    open'], stock_data['high'], stock_data['low'], \
167.                                     stock_data['close'], stock_data['volume']
                ])
168.                 print(stk + '\s data has been stored.')
169.
170.             price = pd.DataFrame(price_data, columns=column_names)
171.             ### Convert UTC to EST
172.             price.DateTime = pd.to_datetime(price.DateTime) - dt.timedelta(hours
                =4)
173.             ### Select during Trading Hour and within selected period
174.             price = price[(price.DateTime.dt.time>=mkt_opentime) & (price.DateTi
                me.dt.time<=mkt_closetime)]
175.             price = price[(price.DateTime.dt.date>=start_date) & (price.DateTime
                .dt.date<=end_date)]
176.             price.to_sql(table_name, con=engine, if_exists='append', index=False
                )
177.
178.         def GridSearchinDBBD(stkdata,H,K1):
179.             data = stkdata.copy()
180.             Notional = 1000000.00 / 10
181.             data['SMA'] = data['close'].rolling(H).mean()
182.             data['rstd'] = data['close'].rolling(H).std()
183.             data['Up1'] = data['SMA'] + K1 * data['rstd']
184.             data['Down1'] = data['SMA'] - K1 * data['rstd']
185.
186.             ### signals
187.             data['signal'] = 0
188.             data.loc[data['close'] >= data['Up1'],'signal'] = -1
189.             data.loc[(data['close'] < data['Up1']) & (data['close'] > data['Down
                1']), 'signal'] = 0
190.             data.loc[data['close'] <= data['Down1'],'signal'] = 1
191.             data.signal = data.signal.shift().fillna(0)
192.             data['trade'] = data.signal.diff()
193.
194.             ### PnL cal
195.             data['pre_trade_pos'] = 0
196.             data['target_pos'] = 0
197.             data['realized_pnl_d'] = 0
198.             last_target_pos = 0
199.             for index, row in data.iterrows():
200.                 data.loc[index,'pre_trade_pos'] = last_target_pos
201.                 if row.trade != 0:
202.                     data.loc[index,'target_pos'] = row.signal * int(Notional / r
                        ow['close'])
203.                     last_target_pos = row.signal * int(Notional / row['close'])
204.
205.                 if abs(row.signal) < abs(row.trade):
206.                     if row.trade < 0:
207.                         data.loc[index,'realized_pnl_d'] = data.loc[index,'pre_t
                            rade_pos'] * row['close'] - Notional
208.                     else:

```

```

209.             data.loc[index,'realized_pnl_d'] = Notional + data.loc[i
            ndex,'pre_trade_pos'] * row['close']
210.
211.             data['realized_pnl_p'] = data['realized_pnl_d'] / Notional
212.             data['cum_pnl_p'] = data['realized_pnl_p'].cumsum() + 1
213.             data['cum_pnl_p_diff'] = data['cum_pnl_p'].diff()
214.
215.             ### IR calculation
216.             data.index = data.datetime
217.             daily_return = pd.DataFrame(data.realized_pnl_p).copy()
218.             daily_return['date'] = daily_return.index
219.             daily_return['date'] = daily_return['date'].apply(lambda x:x.date())

220.             df_whole = daily_return.groupby(['date']).sum()
221.             # performance strat
222.             cumulative_return = df_whole.cumsum()
223.             Annualized_return = cumulative_return.iloc[-
            1,:]* 252 / len(df_whole)
224.             Annualized_vol = df_whole.std() * np.sqrt(252)
225.             Information_ratio = (Annualized_return / Annualized_vol)[0]
226.             CumPnL = cumulative_return.iloc[-1,:][0] * Notional
227.             return Information_ratio, CumPnL
228.
229.
230.
231.         def train_params_DBBD(stk_list,H_list,K1_list,train_end_date,period='1M'
            ):
232.             if period[1] == 'M':
233.                 train_start_date = train_end_date - relativedelta(months=int(per
                    iod[0]))
234.             elif period[1] == 'W':
235.                 train_start_date = train_end_date - relativedelta(weeks=int(peri
                    od[0]))
236.             train_start = dt.datetime(train_start_date.year,train_start_date.mon
                    th,train_start_date.day,9,30)
237.             mkt_opentime = dt.datetime.strptime('09:30','%H:%M').time()
238.             mkt_closetime = dt.datetime.strptime('16:00','%H:%M').time()
239.             stocks = pd.DataFrame(stk_list,columns=['Ticker'])
240.             stocks["H"] = 0
241.             stocks["K1"] = 0.0
242.             stocks['Notional'] = 1000000.00 / 10
243.             stocks["Profit_Loss_in_Training"] = 0.0
244.             stocks['Return_in_Training'] = 0.0
245.             stocks["Profit_Loss"] = 0.0
246.             stocks['Return'] = 0.0
247.             for stk in stk_list:
248.                 print("Training params for: " + stk + '...')
249.                 train_data = pd.DataFrame(get_intraday_data(stk, interval='1m',s
                    tart=train_start))
250.                 ### Convert UTC to EST
251.                 train_data.datetime = pd.to_datetime(train_data.datetime) - dt.t
                    imedelta(hours=4)
252.                 ### Select during Trading Hour and within selected period
253.                 train_data = train_data[(train_data.datetime.dt.time>=mkt_openti
                    me) & (train_data.datetime.dt.time<=mkt_closetime)]
254.                 train_data = train_data[(train_data.datetime.dt.date>=train_star
                    t_date) & (train_data.datetime.dt.date<train_end_date)]
255.                 IR_df = pd.DataFrame(index=H_list,columns=K1_list)
256.                 CumPnLdf = pd.DataFrame(index=H_list,columns=K1_list)
257.                 try:
258.                     for H in H_list:

```

```
259.         for K1 in K1_list:
260.             IR, CumPnL = GridSearchinDBBD(train_data,H,K1)
261.             IR_df.loc[H,K1] = IR
262.             CumPnLdf.loc[H,K1] = CumPnL
263.             print(stk + ':H,K pair:(' + str(H) + ',' + str(K1) +
                ')done, with CumPnL:' + str(CumPnL))
264.             ### select the pair from IR
265.             H0 = CumPnLdf.mean(axis=1).idxmax()
266.             K10 = CumPnLdf.mean().idxmax()
267.             ### delete those with negative PnL within training period
268.             if CumPnLdf.loc[H0,K10] <= 0:
269.                 print('Training performance bad, delete stk:{}'.format(
stk))
270.                 stocks = stocks.drop(stocks[stocks.Ticker==stk].index)
271.             else:
272.                 stocks.loc[stocks[stocks.Ticker==stk].index, 'H'] = H0
273.                 stocks.loc[stocks[stocks.Ticker==stk].index, 'K1'] = K10
274.                 stocks.loc[stocks[stocks.Ticker==stk].index, 'Profit_Loss
_in_Training'] = CumPnLdf.loc[H0,K10]
275.                 stocks.loc[stocks[stocks.Ticker==stk].index, 'Return_in_T
raining'] = CumPnLdf.loc[H0,K10] * 10 / 1000000.00
276.             except:
277.                 print("Deleted. Missing data for stk: " + stk)
278.                 stocks = stocks.drop(stocks[stocks.Ticker==stk].index)
279.         return stocks
280.
281.     def stk_select():
282.         sp500_symbol_list = get_sp500_component()
283.         selected_stk, stk_df = stock_selection(sp500_symbol_list)
284.         return selected_stk, stk_df
285.
286.     def build_trading_model(stk_list):
287.         engine.execute('Drop Table if exists Stocks;')
288.         create_stocks_table('Stocks', metadata, engine)
289.         H_list = [40,50,60,70,80,90]
290.         K1_list = [1.5,1.8,2.0,2.2,2.5]
291.         stocks = train_params_DBBD(stk_list,H_list,K1_list,train_end_date=st
art_date,period='1W')
292.
293.         stocks.to_sql('Stocks', con=engine, if_exists='append', index=False)
294.
295.         return stocks
296.
297.     def populate_backtest_data(stocks):
298.         ### CREATE price table
299.         create_price_table('Price', metadata, engine)
300.         inspector = inspect(engine)
301.         print(inspector.get_table_names())
302.         clear_a_table('Price', metadata, engine)
303.         ### populate price table
304.         populate_price_data(stocks['Ticker'].unique(), engine, 'Price', star
t_date, end_date)
305.
306.         ### create strategy table
307.         engine.execute('Drop Table if exists Strategy;')
308.         create_strategy_table('Strategy', metadata, engine)
309.
310.     def backTest():
311.         ### read data through data base
```

```
312.         stockdf = execute_sql_statement("SELECT * from Stocks",conn)
313.         pricedef = execute_sql_statement("SELECT * from Price", conn)
314.         ### divide the notional
315.         Notional = 1000000 // len(stockdf['Ticker'].unique())
316.         for stk in stockdf['Ticker'].unique():
317.             print("Back Testing:" + stk)
318.             data = pricedef[pricedef.Symbol==stk].copy()
319.             ### select the right parameters according to the training result
320.
321.             H = stockdf.loc[stockdf[stockdf.Ticker==stk].index, 'H'].iloc[0]
322.             K1 = stockdf.loc[stockdf[stockdf.Ticker==stk].index, 'K1'].iloc[0]
323.             ]
324.             ## calculating rolling H-period Moving average and rolling std
325.             ## the number of the std to form the band is K1
326.             data['SMA'] = data['Close'].rolling(H).mean()
327.             data['rstd'] = data['Close'].rolling(H).std()
328.             data['Up1'] = data['SMA'] + K1 * data['rstd']
329.             data['Down1'] = data['SMA'] - K1 * data['rstd']
330.
331.             ### signals generation
332.             data['signal'] = 0
333.             ## when close larger than ma + k1 * std -> 短
334.             €short, when close smaller than ma - k1 * std -> long
335.             data.loc[data['Close'] >= data['Up1'], 'signal'] = -1
336.             data.loc[(data['Close'] < data['Up1']) & (data['Close'] > data['Down1']), 'signal'] = 0
337.             data.loc[data['Close'] <= data['Down1'], 'signal'] = 1
338.             ## 短€shift signal since we trade at next min
339.             data.signal = data.signal.shift().fillna(0)
340.             data['trade'] = data.signal.diff()
341.
342.             ### PnL calculation
343.             data['pre_trade_pos'] = 0
344.             data['target_pos'] = 0
345.             data['realized_pnl_d'] = 0
346.             last_target_pos = 0 # store the target position we trade at last
347.             entry, in order to cal the realized pnl
348.             for index, row in data.iterrows():
349.                 data.loc[index, 'pre_trade_pos'] = last_target_pos
350.                 if row.trade != 0:
351.                     ## cal the position according to Notional and close, can
352.                     not do fraction so always make it int
353.                     data.loc[index, 'target_pos'] = row.signal * int(Notional
354.                     / row['Close'])
355.                     last_target_pos = row.signal * int(Notional / row['Close
356.                     '])
357.                 if abs(row.signal) < abs(row.trade): ## when it is an exit,
358.                 we calculate the realized pnl of this trade
359.                 if row.trade < 0:
360.                     data.loc[index, 'realized_pnl_d'] = data.loc[index, 'pre_trade_pos'] * row['Close'] - Notional
361.                 else:
362.                     data.loc[index, 'realized_pnl_d'] = Notional + data.loc[index, 'pre_trade_pos'] * row['Close']
363.                 ## cal pnl in percentage, also cumulative pnl
364.                 data['realized_pnl_p'] = data['realized_pnl_d'] / Notional
365.                 data['cum_pnl_p'] = data['realized_pnl_p'].cumsum()
366.
367.             CumPnL = data['realized_pnl_d'].cumsum().iloc[-1]
```



```
361.         Return = data['cum_pnl_p'].iloc[-1]
362.         engine.execute("UPDATE Stocks SET Profit_Loss = " + str(CumPnL)
+ " WHERE Ticker = '" + stk + "'"")
363.         engine.execute("UPDATE Stocks SET Notional = " + str(Notional) +
" WHERE Ticker = '" + stk + "'"")
364.         engine.execute("UPDATE Stocks SET Return = " + str(Return) + " W
HERE Ticker = '" + stk + "'"")
365.
366.     def probation_test(probation_testing_start_date, probation_testing_end_d
ate):
367.         ### read data through data base
368.         stockdf = pd.read_sql_query("SELECT * from Stocks", conn)
369.         pricdf = pd.read_sql_query("SELECT * from Price", conn)
370.         #engine.execute('Drop Table if exists Probation;')
371.         #create_stocks_table('Probation', metadata, engine)
372.         stocks = stockdf.copy(deep=True)
373.         stocks["Profit_Loss_in_Training"] = 0.0
374.         stocks['Return_in_Training'] = 0.0
375.         stocks["Profit_Loss"] = 0.0
376.         stocks['Return'] = 0.0
377.         ### divide the notional
378.         Notional = 1000000 // len(stockdf['Ticker'].unique())
379.         for stk in stockdf['Ticker'].unique():
380.             print("Back Testing:" + stk)
381.             data = pricdf[pricdf.Symbol==stk].copy()
382.             data = data[(data.DateTime>=probation_testing_start_date)&(data.
DateTime<=probation_testing_end_date)]
383.             ### select the right parameters according to the training result
384.             H = stockdf.loc[stockdf[stockdf.Ticker==stk].index, 'H'].iloc[0]
385.             K1 = stockdf.loc[stockdf[stockdf.Ticker==stk].index, 'K1'].iloc[0
]
386.             ## calculating rolling H-period Moving average and rolling std
387.             ## the number of the std to form the band is K1
388.             data['SMA'] = data['Close'].rolling(H).mean()
389.             data['rstd'] = data['Close'].rolling(H).std()
390.             data['Up1'] = data['SMA'] + K1 * data['rstd']
391.             data['Down1'] = data['SMA'] - K1 * data['rstd']
392.
393.             ### signals generation
394.             data['signal'] = 0
395.             ## when close larger than ma + k1 * std -> 鎏
€short, when close smaller than ma - k1 * std -> long
396.             data.loc[data['Close'] >= data['Up1'], 'signal'] = -1
397.             data.loc[(data['Close'] < data['Up1']) & (data['Close'] > data['
Down1']), 'signal'] = 0
398.             data.loc[data['Close'] <= data['Down1'], 'signal'] = 1
399.             ##鎏€shift signal since we trade at next min
400.             data.signal = data.signal.shift().fillna(0)
401.             data['trade'] = data.signal.diff()
402.
403.             ### PnL calculation
404.             data['pre_trade_pos'] = 0
405.             data['target_pos'] = 0
406.             data['realized_pnl_d'] = 0
407.             last_target_pos = 0 # store the target position we trade at last
entry, in order to cal the realized pnl
408.             for index, row in data.iterrows():
409.                 data.loc[index, 'pre_trade_pos'] = last_target_pos
410.                 if row.trade != 0:
```

```
411.         ## cal the position according to Notional and close, can
not do fraction so always make it int
412.         data.loc[index, 'target_pos'] = row.signal * int(Notional
/ row['Close'])
413.         last_target_pos = row.signal * int(Notional / row['Close
'])
414.
415.         if abs(row.signal) < abs(row.trade): ## when it is an exit,
we calculate the realized pnl of this trade
416.             if row.trade < 0:
417.                 data.loc[index, 'realized_pnl_d'] = data.loc[index, 'p
re_trade_pos'] * row['Close'] - Notional
418.             else:
419.                 data.loc[index, 'realized_pnl_d'] = Notional + data.l
oc[index, 'pre_trade_pos'] * row['Close']
420.         ## cal pnl in percentage, also cumulative pnl
421.         data['realized_pnl_p'] = data['realized_pnl_d'] / Notional
422.         data['cum_pnl_p'] = data['realized_pnl_p'].cumsum()
423.
424.         CumPnL = data['realized_pnl_d'].cumsum().iloc[-1]
425.         stocks.loc[stocks[stocks.Ticker==stk].index, 'Profit_Loss'] = Cum
PnL
426.         #stocks.to_sql('Probation', con=engine, if_exists='append', inde
x=False)
427.         return stocks
428.
429.     def populate_strategy_bt():
430.
431.         stockdf = pd.read_sql_query("SELECT * from Stocks", conn)
432.         Notional = 1000000.00
433.         train_PnL = stockdf.Profit_Loss_in_Training.sum()
434.         train_Ret = train_PnL / Notional
435.
436.         bt_PnL = stockdf.Profit_Loss.sum()
437.         bt_Ret = bt_PnL / Notional
438.
439.         ### divide time into training period and bt period
440.         bt_end_date = dt.datetime.today().date()
441.         bt_start_date = end_date - relativedelta(weeks=1)
442.         train_end_date = bt_start_date - relativedelta(days=1)
443.         train_start_date = train_end_date - relativedelta(weeks=1)
444.         ### get SPY data
445.         spy_df = pd.DataFrame(get_daily_data('SPY', train_start_date, bt_end_d
ate))
446.         train_start_open = spy_df.open.iloc[0]
447.         train_end_close = spy_df.loc[spy_df.date<=str(train_end_date), 'close
'].iloc[-1]
448.
449.         ### calculate return of SPY, dividing into training and back testing
450.         bt_start_open = spy_df.loc[spy_df.date>=str(bt_start_date), 'open'].i
loc[0]
451.         bt_end_close = spy_df.close.iloc[-1]
452.         train_ret = (train_end_close - train_start_open) / train_start_open
453.
454.         bt_ret = (bt_end_close - bt_start_open) / bt_start_open
455.
456.         ### populate result into strategy table
457.         strategy = pd.DataFrame(index=[0], columns = ['Strategy'])
458.         strategy['Strategy'] = 'BB'
459.         strategy['Notional'] = Notional
```

```
459.         strategy['PnL_in_Training'] = train_PnL
460.         strategy['Return_in_Training'] = train_Ret
461.         strategy['SPY_Return_in_Training'] = train_ret
462.         strategy['PnL'] = bt_PnL
463.         strategy['Strategy_Return'] = bt_Ret
464.         strategy['SPY_Return'] = bt_ret
465.
466.         strategy.to_sql('Strategy', con=engine, if_exists='append', index=False)
```

2. BBD_client.py

```
1.  # -*- coding: utf-8 -*-
2.  #!/usr/bin/env python3
3.  #@ Copyright -
4.
5.  import json
6.  import sys
7.
8.  from network import PacketTypes, Packet
9.  import queue
10. import threading
11. import pandas as pd
12. import numpy as np
13. from queue import Queue
14. from socket import AF_INET, socket, SOCK_STREAM, IPPROTO_TCP, TCP_NODELAY
15.
16. import datetime
17.
18. from sqlalchemy import create_engine
19. from sqlalchemy import MetaData
20. import time
21.
22. def usd(value):
23.     """Format value as USD."""
24.     return f"${value:,.2f}"
25.
26. def get_stock_selection_list():
27.     enginestk = create_engine('sqlite:///TradingBBD.db', connect_args={'check_same_thread': False})
28.     connstk = enginestk.connect()
29.     stockdf = pd.read_sql_query("SELECT * from Stocks", connstk)
30.     stklist = list(stockdf.Ticker)
31.     return stklist
32.
33. class ClientConfig:
34.     def __init__(self):
35.         self.client_id = "client1"
36.         ### server side below
37.         self.HOST = "127.0.0.1" ### local host, local ip
38.         self.PORT = 6510
39.         self.BUF_SIZE = 4096 ### maximum message 4k
40.         self.ADDR = (self.HOST, self.PORT)
41.
42.         self.client_socket = socket(AF_INET, SOCK_STREAM)
43.         self.client_socket.setsockopt(IPPROTO_TCP, TCP_NODELAY, True)
44.         self.client_thread = threading.Thread()
45.         self.client_receiver = threading.Thread()
46.
```

```
47.         self.orders = []
48.
49.         self.client_thread_started = False
50.         self.trade_complete = False
51.         self.client_up = False
52.         self.client_symbols = ''
53.         self.PnL = 0
54.         self.Ticker_PnL = {}
55.
56.
57. trading_queue = queue.Queue()
58. trading_event = threading.Event()
59. client_config = ClientConfig()
60.
61. def receive(q=None, e=None):
62.     total_server_response = b'' ##in bytes
63.     while True:
64.         try:
65.             server_response = client_config.client_socket.recv(client_config.BUF
        _SIZE)
66.             total_server_response += server_response
67.             msgSize = len(total_server_response)
68.             while msgSize > 0: ## get sth
69.                 if msgSize > 12: ## more than 3 int
70.                     server_packet = Packet()
71.                     server_packet.deserialize(total_server_response)
72.                     if msgSize > 12 and server_packet.m_data_size <= msgSize: ## i h
        ave complete message
73.                         data = json.loads(server_packet.m_data)
74.                         q.put([server_packet.m_type, data]) ## pass from receiver to
        sender, via queue
75.                         total_server_response = total_server_response[server_packet.
        m_data_size:] ## cut the message and move on
76.                         msgSize = len(total_server_response)
77.                     else: ## not enough, keep receiving more
78.                         server_response = client_config.client_socket.recv(client_co
        nfig.BUF_SIZE)
79.                         total_server_response += server_response
80.                         msgSize = len(total_server_response)
81.
82.                 if not q.empty() and e.isSet():
83.                     e.clear()
84.
85.             except (OSError,Exception):
86.                 print("Exception in receive\n")
87.                 sys.exit(0)
88.
89. def send_msg(client_packet):
90.     client_config.client_socket.send(client_packet.serialize())
91.     data = json.loads(client_packet.m_data)
92.     # print(data) ## take a look
93.     return data
94.
95. def get_response(q):
96.     msg_type, msg_data = q.get()
97.     # print(msg_data)
98.     if msg_data is not None:
99.         if msg_type == PacketTypes.END_RSP.value or msg_type == PacketTypes.SERV
        ER_DOWN_RSP.value or \
100.            (msg_type == PacketTypes.CONNECTION_RSP.value and msg_da
        ta["Status"] == "Rejected"):
```

```
101.         client_config.client_socket.close() ## client get out
102.         sys.exit(0)
103.         return msg_data
104.
105.     def set_event(e):
106.         e.set()
107.
108.     def wait_for_an_event(e):
109.         while e.isSet():
110.             continue
111.
112.     def send(q=None, e=None):
113.         try:
114.             while True:
115.                 client_packet = Packet()
116.                 user_input = input("Action:")
117.                 input_list = user_input.strip().split(" ")
118.                 if len(input_list) < 2:
119.                     print("Incorrect Input.\n")
120.                     continue
121.                 ### case sensitive, won't use next week
122.                 if "Logon" in user_input:
123.                     client_packet.m_type = PacketTypes.CONNECTION_REQ.value
124.                     ### enum here
125.                     client_packet.m_data = json.dumps({'Client':client_conf
126. g.client_id, 'Status':input_list[0], 'Symbol':input_list[1]})
127.
128.                 elif "Client List" in user_input:
129.                     client_packet.m_type = PacketTypes.CLIENT_LIST_REQ.value
130.
131.                     client_packet.m_data = json.dumps({'Client':client_conf
132. ig.client_id, 'Status':input_list[0] + ' ' + input_list[1]})
133.
134.                 elif "Stock List" in user_input:
135.                     client_packet.m_type = PacketTypes.STOCK_LIST_REQ.value
136.
137.                     client_packet.m_data = json.dumps({'Client':client_conf
138. ig.client_id, 'Status':input_list[0] + ' ' + input_list[1]})
139.
140.                 elif "Book Inquiry" in user_input:
141.                     if len(input_list) < 3:
142.                         print("Missing input item(s).\n")
143.                         continue
144.                     client_packet.m_type = PacketTypes.BOOK_INQUIRY_REQ.valu
145. e
146.                     client_packet.m_data = json.dumps({'Client':client_conf
147. ig.client_id, 'Status':input_list[0] + ' ' + input_list[1], 'Symbol':input_list[
148. 2]})
149.
150.                 elif "New Order" in user_input:
151.                     if len(input_list) < 6:
152.                         print("Missing input item(s).\n")
153.                         continue
154.                     client_packet.m_type = PacketTypes.NEW_ORDER_REQ.value
155.                     client_packet.m_data = json.dumps({'Client':client_conf
156. ig.client_id, 'Status':input_list[0] + ' ' + input_list[1], 'Symbol':input_list[
157. 2], 'Type':'Lmt', 'Side':input_list[3], 'Price':input_list[4], 'Qty':input_list[
158. 5]})
159.
160.                 elif "Client Quit" in user_input:
161.                     client_packet.m_type = PacketTypes.END_REQ.value
```

```
150.         client_packet.m_data = json.dumps({'Client':client_conf
g.client_id, 'Status':input_list[0]})
151.
152.         else:
153.             print("Invalid message\n")
154.             continue
155.
156.             set_event(e)
157.             send_msg(client_packet)
158.             wait_for_an_event(e)
159.             msg_type, msg_data = q.get()
160.             q.task_done()
161.             print(msg_data)
162.             if msg_data is not None:
163.                 if msg_type == PacketTypes.END_RSP.value or msg_type ==
PacketTypes.SERVER_DOWN_RSP.value or \
164.                 (msg_type == PacketTypes.CONNECTION_RSP.value and ms
g_data["Status"] == "Rejected"):
165.                     client_config.client_socket.close()
166.                     sys.exit(0)
167.
168.             except(OSError, Exception):
169.                 q.put(PacketTypes.NONE.value, Exception('send'))
170.                 client_socket.close()
171.                 sys.exit(0)
172.
173.
174.
175.
176.
177.     def Client_Manual():
178.         try:
179.
180.             client_config.client_socket = socket(AF_INET, SOCK_STREAM)
181.             client_config.client_socket.setsockopt(IPPROTO_TCP, TCP_NODE
LAY, True)
182.             status = client_config.client_socket.connect_ex(client_conf
g.ADDR)
183.             if status != 0:
184.                 print("Fail in connecting to server")
185.                 sys.exit(0)
186.
187.             ## send thread and receive thread, thread is also object
188.             client_config.client_receiver = threading.Thread(target=rece
ive, args=(trading_queue, trading_event))
189.             client_config.client_sender = threading.Thread(target=send,
args=(trading_queue, trading_event))
190.
191.             client_config.client_receiver.start()
192.             client_config.client_sender.start()
193.
194.             if client_config.client_receiver.is_alive() is True:
195.                 client_config.client_receiver.join()
196.
197.             if client_config.client_sender.is_alive() is True:
198.                 client_config.client_sender.join()
199.
200.         except (KeyError, KeyboardInterrupt, SystemExit, Exception):
201.             client_config.client_socket.close()
202.             sys.exit(0)
203.
```

```
204.
205.     def logon(client_packet):
206.         client_config.client_symbols = ','.join([str(elem) for elem in get_s
tock_selection_list()])
207.         client_packet.m_type= PacketTypes.CONNECTION_REQ.value
208.         client_packet.m_data = json.dumps({'Client':client_config.client_id,
'Status':'Logon', 'Symbol':client_config.client_symbols})
209.         return client_packet
210.
211.     def get_client_list(client_packet):
212.         client_packet.m_type = PacketTypes.CLIENT_LIST_REQ.value
213.         client_packet.m_data = json.dumps({'Client':client_config.client_id
, 'Status':'Client List'})
214.         return client_packet
215.
216.     def get_stock_list(client_packet):
217.         client_packet.m_type = PacketTypes.STOCK_LIST_REQ.value
218.         client_packet.m_data = json.dumps({'Client':client_config.client_id
, 'Status':'Stock List'})
219.         return client_packet
220.
221.     def get_market_status(client_packet):
222.         client_packet.m_type = PacketTypes.MARKET_STATUS_REQ.value
223.         client_packet.m_data = json.dumps({'Client':client_config.client_id,
'Status':'Market Status'})
224.         return client_packet
225.
226.     def get_order_book(client_packet, symbol):
227.         client_packet.m_type = PacketTypes.BOOK_INQUIRY_REQ.value
228.         client_packet.m_data = json.dumps({'Client':client_config.client_id,
'Status':'Book Inquiry', 'Symbol':symbol})
229.         return client_packet
230.
231.     def new_order(client_packet, order_id, symbol, order_type, side, price,
qty):
232.         if order_type == "Mkt":
233.             price = 0
234.             client_packet.m_type = PacketTypes.NEW_ORDER_REQ.value
235.             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})
236.             return client_packet
237.
238.     def client_quit(client_packet):
239.         client_packet.m_type = PacketTypes.END_REQ.value
240.         client_packet.m_data = json.dumps({'Client':client_config.client_id,
'Status':'Client Quit'})
241.         return client_packet
242.
243.     class StocksInfo():
244.         def __init__(self, Ticker_, H_, K1_, Notional_, price_queue_):
245.             self.Ticker = Ticker_
246.             self.H = H_
247.             self.K1 = K1_
248.             self.Notional = Notional_
249.             self.price_queue = price_queue_
250.             self.Std = "null"
251.             self.MA = "null"
252.             self.position = 0
253.             self.Qty = 0
254.             self.current_price_buy = 0
```

```
255.         self.current_price_sell = 1e6
256.         self.Tradelist = []
257.         self.PnLlist = []
258.         self.PnL = 0
259.
260.     def StkInfo_init():
261.         enginestk = create_engine('sqlite:///TradingBBD.db', connect_args={'c
heck_same_thread': False})
262.         connstk = enginestk.connect()
263.         stockdf = pd.read_sql_query("SELECT * from Stocks", connstk) ###
264.         stockdf.index = stockdf['Ticker']
265.         stock_info_dict = {stk: StocksInfo(stk, stockdf.loc[stk, 'H'], stockdf.
loc[stk, 'K1'], stockdf.loc[stk, 'Notional'],
266.                                     Queue(int(stockdf.loc[stk, 'H'] / 5
))) for stk in stockdf['Ticker']}
267.         return stock_info_dict
268.
269.     class Trade():
270.         def __init__(self, Ticker_, Orders_Responses_):    ### open trade
271.             Orders_Responses = [response for response in Orders_Responses_ i
f response['Symbol'] == Ticker_]
272.             self.Ticker = Ticker_
273.             self.Postion = 1 if Orders_Responses[0]['Side'] == 'Buy' else -
1
274.             self.ClosePrice = np.nan
275.             self.PnL = np.nan
276.             ### Open Order Filled with one order
277.             if len(Orders_Responses) == 1:
278.                 Filled_order = Orders_Responses[0]
279.                 Price = Filled_order['Price']
280.                 Qty = int(Filled_order['Qty'])
281.                 self.OpenPrice = Price
282.                 self.Qty = Qty
283.             ### Open Order Filled with several orders
284.             else:
285.                 PriceList = [orders['Price'] for orders in Orders_Responses]
286.                 QtyList = [int(orders['Qty']) for orders in Orders_Responses
]
287.                 self.Qty = sum(QtyList)
288.                 self.OpenPrice = sum([P * Q for P, Q in zip(PriceList, QtyLis
t)]) / sum(QtyList)
289.                 print("Open Trade in: ", self.Ticker, "With postion: ", self.Pos
tion, "at Price: ", self.OpenPrice, "With Qty: ", self.Qty)
290.             def CloseTrade(self, Orders_Responses_):
291.                 Orders_Responses = [response for response in Orders_Responses_ i
f response['Symbol'] == self.Ticker]
292.                 ### Close Order Filled with one order
293.                 if len(Orders_Responses) == 1:
294.                     Filled_order = Orders_Responses[0]
295.                     Price = Filled_order['Price']
296.                     self.ClosePrice = Price
297.                     self.PnL = (self.ClosePrice - self.OpenPrice) * self.Qty if
self.Postion == 1 else (self.OpenPrice - self.ClosePrice) * self.Qty
298.                 ### Close Order Filled with several orders
299.                 else:
300.                     PriceList = [orders['Price'] for orders in Orders_Responses]
301.                     QtyList = [int(orders['Qty']) for orders in Orders_Responses
]
```



```
302.         self.ClosePrice = sum([P * Q for P,Q in zip(PriceList, QtyList
303.         st)]) / sum(QtyList)
304.         self.PnL = (self.ClosePrice - self.OpenPrice) * self.Qty if
305.         self.Postion == 1 else (self.OpenPrice - self.ClosePrice) * self.Qty
306.         print("Close Trade in: ", self.Ticker, "at Open Price: ", self.O
307.         penPrice, "at Close Price: ", self.ClosePrice, "With Qty: ", self.Qty, "PnL: ",s
308.         elf.PnL)
309.
310.
311.     def LogonAndTrade(q=None, e=None):
312.
313.         client_packet = Packet()
314.         set_event(e)
315.         send_msg(logon(client_packet))
316.         wait_for_an_event(e)
317.         get_response(q)
318.
319.         set_event(e)
320.         send_msg(get_client_list(client_packet))
321.         wait_for_an_event(e)
322.         get_response(q)
323.
324.         set_event(e)
325.         send_msg(get_stock_list(client_packet))
326.         wait_for_an_event(e)
327.         get_response(q)
328.
329.         ### find end date
330.         lastBusDay = datetime.datetime.today()
331.         if datetime.date.weekday(lastBusDay) == 5:      #if it's Saturday
332.             lastBusDay = lastBusDay - datetime.timedelta(days = 1) #then mak
333.         e it Friday
334.         elif datetime.date.weekday(lastBusDay) == 6:      #if it's Sunday
335.             lastBusDay = lastBusDay - datetime.timedelta(days = 2)
336.         ##TO DO
337.         end_date = lastBusDay - datetime.timedelta(days = 1) ###
338.         end_trading_mktperiod = end_date.strftime("%Y-%m-%d")
339.
340.         ### initialize StkInfo Dict
341.         StockInfoDict = StkInfo_init()
342.         base_filled_orderid = []
343.         # sample = open('orderbook.txt', 'w')
344.         OrderIndex = 0
345.         while True: ### outer loop
346.             while True: ### inner loop
347.                 client_packet = Packet()
348.                 set_event(e)
349.                 send_msg(get_market_status(client_packet))
350.                 wait_for_an_event(e)
351.                 ### when not empty
352.                 while not q.empty():
353.                     data = get_response(q)
354.                     if data['Status'] not in ['Open', 'Pending Closing', 'Mark
355.                     et Closed', 'Pending Open']:
356.                         client_config.orders.append(data)
357.                     else:
358.                         mkt_status = data
359.                         # print("mkt_status", mkt_status)
360.                         ### wait till mkt open
361.                         if mkt_status["Status"] == 'Open' or mkt_status["Status"] ==
362.                         'Pending Closing':
```

```
356.         break
357.         if mkt_status['Market_Period'] == end_trading_mktperiod and
           mkt_status["Status"] == "Market Closed":
358.             ### calculate PnL and stop trade Logic 1
359.             # TotalPnL = 0
360.             # for stk in StockInfoDict:
361.             #     TotalPnL += sum(StockInfoDict[stk].PnLlist)
362.             # client_config.PnL = TotalPnL
363.             # ### calculate PnL for different Ticker
364.             # PnL_dict = {stk:usd(sum(StockInfoDict[stk].PnLlist)) fo
           r stk in StockInfoDict}
365.             # client_config.Ticker_PnL = PnL_dict
366.             ### PnL Calculation Logic 2
367.             PnL_dict = {}
368.             for stk in StockInfoDict:
369.                 stkbuy_order = [order for order in client_config.ord
           ers if (order['Symbol'] == stk)&(order['Side'] == 'Buy')]
370.                 stkbuy_price = [order['Price'] for order in stkbuy_o
           rder]
371.                 stkbuy_qty = [int(order['Qty']) for order in stkbuy_
           order]
372.                 stksell_order = [order for order in client_config.or
           ders if (order['Symbol'] == stk)&(order['Side'] == 'Sell')]
373.                 stksell_price = [order['Price'] for order in stksell
           _order]
374.                 stksell_qty = [int(order['Qty']) for order in stksel
           l_order]
375.                 stkPnL = sum([P * Q for P,Q in zip(stksell_price, st
           ksell_qty)]) - sum([P * Q for P,Q in zip(stkbuy_price, stkbuy_qty)])
376.                 PnL_dict.update({stk:stkPnL})
377.
378.                 client_config.PnL = sum(PnL_dict.values())
379.                 client_config.Ticker_PnL = {stk: usd(PnL_dict[stk]) for
           stk in PnL_dict}
380.             ### complete the trade
381.             client_config.trade_complete = True
382.             break
383.             time.sleep(0.5)
384.             if client_config.trade_complete:
385.                 break
386.             ### close every day in Pending Close
387.             if mkt_status["Status"] == "Pending Closing":
388.                 # OrderIndex = 0
389.                 for stk in StockInfoDict:
390.                     stkInfo_object = StockInfoDict[stk]
391.                     stkInfo_object.MA = 'null'
392.                     stkInfo_object.Std = 'null'
393.                     stkInfo_object.price_queue = Queue(int(stkInfo_object.H
           / 5)) # reset the members
394.                     if stkInfo_object.position != 0:
395.                         client_packet = Packet()
396.                         OrderIndex += 1
397.                         client_order_id = client_config.client_id + '_' + st
           r(OrderIndex)
398.                     ### if longing
399.                     if stkInfo_object.position > 0:
400.                         new_order(client_packet, client_order_id, stk, '
           Mkt', 'Sell', 100, stkInfo_object.Qty)
401.                         print("Close Trade in: ", stk, "With postion: Se
           ll", "With Qty: ", stkInfo_object.Qty)
402.                         print("Because: Close at Pending Close.")
```

```
403.                set_event(e)
404.                send_msg(client_packet)
405.                wait_for_an_event(e)
406.                ### close trade logic
407.                response_list = []
408.                while not q.empty():
409.                    response_data = get_response(q)
410.                    response_list.append(response_data)
411.                    client_config.orders.append(response_data)
412.                #                Trade_object = stkInfo_object.Tradelist[-1]
413.                #                Trade_object.CloseTrade(response_list)
414.                #                stkInfo_object.PnLlist.append(Trade_object.PnL)

415.                stkInfo_object.Qty = 0
416.                stkInfo_object.position = 0
417.
418.                ### if shorting
419.                else:
420.                    new_order(client_packet, client_order_id, stk, '
Mkt', 'Buy', 100, stkInfo_object.Qty)
421.                    print("Close Trade in: ", stk, "With postion: Bu
y", "With Qty: ", stkInfo_object.Qty)
422.                    print("Because: Close at Pending Close.")
423.                    set_event(e)
424.                    send_msg(client_packet)
425.                    wait_for_an_event(e)
426.                    ### close trade logic
427.                    response_list = []
428.                    while not q.empty():
429.                        response_data = get_response(q)
430.                        response_list.append(response_data)
431.                        client_config.orders.append(response_data)
432.                    #                Trade_object = stkInfo_object.Tradelist[-1]
433.                    #                Trade_object.CloseTrade(response_list)
434.                    #                stkInfo_object.PnLlist.append(Trade_object.PnL)

435.                    stkInfo_object.Qty = 0
436.                    stkInfo_object.position = 0
437.                    continue ### re-enter into checking "Open" while-loop
438.
439.                ### BBD Trading Logic
440.                client_packet = Packet()
441.                set_event(e)
442.                client_msg = get_order_book(client_packet, client_config.client_
symbols)
443.                send_msg(client_msg)
444.                wait_for_an_event(e)
445.
446.                ### when not empty
447.                while True:
448.                    data = get_response(q)
449.                    if type(data) == dict:
450.                        if data['Status']!= 'Done':
451.                            client_config.orders.append(data)
452.                        else:
453.                            break
454.
455.                book_data = json.loads(data)
456.                order_book = book_data["data"]
457.
458.                #                print(order_book, file = sample)
```

```
459.
460.         filled_order_book = [fill_orders for fill_orders in order_book i
461. f fill_orders['Status'] in ['Filled']]
461.         filled_orderid = [order['OrderIndex'] for order in filled_order_
book]
462.         standing_order_book = [standing_orders for standing_orders in or
der_book if standing_orders['Status'] in ['New', 'Partial Filled']]
463.         #         print(filled_order_book, file = sample)
464.         #         print(standing_order_book, file = sample)
465.
466.         #         OrderIndex = 0
467.
468.         for stk in StockInfoDict:
469.             standing_buy_price_list = [order['Price'] for order in stand
ing_order_book if (order['Symbol'] == stk)&(order['Side'] == 'Buy')]
470.             standing_sell_price_list = [order['Price'] for order in stan
ding_order_book if (order['Symbol'] == stk)&(order['Side'] == 'Sell')]
471.             StockInfoDict[stk].current_price_sell = min(standing_sell_pr
ice_list)
472.             StockInfoDict[stk].current_price_buy = max(standing_buy_pric
e_list)
473.
474.             ### store current price in price queue and use it to calculate M
A and std
475.             for stk in StockInfoDict:
476.                 stkInfo_object = StockInfoDict[stk]
477.                 ### current price is based on filled_order_book
478.                 if len(base_filled_orderid) == 0:
479.                     current_price = (stkInfo_object.current_price_buy + stkI
nfo_object.current_price_sell) / 2
480.                     base_filled_orderid = filled_orderid
481.                 else:
482.                     try:
483.                         newly_filled_orderid = [orderid for orderid in fille
d_orderid if orderid not in base_filled_orderid]
484.                         base_filled_orderid = filled_orderid
485.                         newly_filled_order = [order for order in filled_orde
r_book if order['OrderIndex'] in newly_filled_orderid]
486.                         filled_price_list = [order['Price'] for order in new
ly_filled_order]
487.                         filled_qty_list = [int(order['OrigQty']) for order i
n newly_filled_order]
488.                         current_price = sum([P * Q for P,Q in zip(filled_pri
ce_list, filled_qty_list)]) / sum(filled_qty_list)
489.                     except: ### when no newly filled
490.                         current_price = (stkInfo_object.current_price_buy +
stkInfo_object.current_price_sell) / 2
491.                         #         print("current price for", stk, "P= " current_price)
492.                         if not stkInfo_object.price_queue.full():
493.                             stkInfo_object.price_queue.put(current_price)
494.                         if stkInfo_object.price_queue.full():
495.                             stkInfo_object.MA = np.array(stkInfo_object.price_qu
eue.queue).mean()
496.                             stkInfo_object.Std = np.array(stkInfo_object.price_q
ueue.queue).std() / np.sqrt(5)
497.                         else: # already full
498.                             popout = stkInfo_object.price_queue.get()
499.                             stkInfo_object.price_queue.put(current_price)
500.                             stkInfo_object.MA = np.array(stkInfo_object.price_queue.
queue).mean()
```

```
501.         stkInfo_object.Std = np.array(stkInfo_object.price_queue
.queue).std() / np.sqrt(5)
502.
503.
504.         for stk in StockInfoDict:
505.             stkInfo_object = StockInfoDict[stk]
506.             K1 = stkInfo_object.K1
507.             MA = stkInfo_object.MA
508.             Std = stkInfo_object.Std
509.             Notional = stkInfo_object.Notional
510.             if MA == 'null':
511.                 continue
512.             current_buy = stkInfo_object.current_price_buy
513.             current_sell = stkInfo_object.current_price_sell
514.             #         current_p = (current_buy + current_sell) / 2
515.             #         print("K1: ",K1)
516.             #         print("MA: ",MA)
517.             #         print("Std: ",Std)
518.             #         print("sell p:",current_sell)
519.             #         print("buy p:",current_buy)
520.             if stkInfo_object.position == 0: # not yet open position, co
uld open
521.                 if current_sell <= MA - K1 * Std: # below lower band, go
long
522.                     stkInfo_object.position = 1
523.                     client_packet = Packet()
524.                     OrderIndex += 1
525.                     client_order_id = client_config.client_id + '_' + st
r(OrderIndex)
526.                     stkInfo_object.Qty = int(Notional / current_sell)
527.                     new_order(client_packet, client_order_id, stk, 'Mkt'
, 'Buy', 100, stkInfo_object.Qty)
528.                     print("Open Trade in: ", stk, "With postion: Buy", "
at Price:", current_sell, "With Qty:", stkInfo_object.Qty)
529.                     print("Because: Price below lower band:", usd(MA - K
1 * Std))
530.                     set_event(e)
531.                     send_msg(client_packet)
532.                     wait_for_an_event(e)
533.                     ### open logic
534.                     response_list = []
535.                     while not q.empty():
536.                         response_data = get_response(q)
537.                         response_list.append(response_data)
538.                         client_config.orders.append(response_data)
539.                     #         Trade_object = Trade(stk, response_list)
540.                     #         stkInfo_object.Tradelist.append(Trade_object)
541.
542.
543.
544.                 elif current_buy >= MA + K1 * Std: # above upper band, g
o short
545.                     stkInfo_object.position = -1
546.                     client_packet = Packet()
547.                     OrderIndex += 1
548.                     client_order_id = client_config.client_id + '_' + st
r(OrderIndex)
549.                     stkInfo_object.Qty = int(Notional / current_buy)
550.                     new_order(client_packet, client_order_id, stk, 'Mkt'
, 'Sell', 100, stkInfo_object.Qty)
```

```
551.         print("Open Trade in: ", stk, "With postion: Sell",
    "at Price:", current_buy, "With Qty: ", stkInfo_object.Qty)
552.         print("Because: Price above upper band:", usd(MA + K
    1 * Std))
553.         set_event(e)
554.         send_msg(client_packet)
555.         wait_for_an_event(e)
556.         ### open logic
557.         response_list = []
558.         while not q.empty():
559.             response_data = get_response(q)
560.             response_list.append(response_data)
561.             client_config.orders.append(response_data)
562.             # Trade_object = Trade(stk, response_list)
563.             # stkInfo_object.Tradelist.append(Trade_object)
564.
565.
566.         elif stkInfo_object.position == 1: # longing now
567.             if current_buy >= MA: # above lower bound, sell to close
    postion
568.                 client_packet = Packet()
569.                 OrderIndex += 1
570.                 client_order_id = client_config.client_id + '_' + st
    r(OrderIndex)
571.                 new_order(client_packet, client_order_id, stk, 'Mkt'
    , 'Sell', 100, stkInfo_object.Qty)
572.                 print("Close Trade in: ", stk, "With postion: Sell",
    "at Price:", current_buy, "With Qty: ", stkInfo_object.Qty)
573.                 print("Because: Price above lower band:", usd(MA))
574.                 set_event(e)
575.                 send_msg(client_packet)
576.                 wait_for_an_event(e)
577.                 ### close trade logic
578.                 response_list = []
579.                 while not q.empty():
580.                     response_data = get_response(q)
581.                     response_list.append(response_data)
582.                     client_config.orders.append(response_data)
583.                     # Trade_object = stkInfo_object.Tradelist[-1]
584.                     # Trade_object.CloseTrade(response_list)
585.                     # stkInfo_object.PnLlist.append(Trade_object.PnL)
586.                     stkInfo_object.Qty = 0
587.                     stkInfo_object.position = 0
588.
589.
590.             else: # shorting now
591.                 if current_sell <= MA: # below upper bound, buy to close
    postion
592.                     client_packet = Packet()
593.                     OrderIndex += 1
594.                     client_order_id = client_config.client_id + '_' + st
    r(OrderIndex)
595.                     new_order(client_packet, client_order_id, stk, 'Mkt'
    , 'Buy', 100, stkInfo_object.Qty)
596.                     print("Close Trade in: ", stk, "With postion: Buy",
    "at Price:", current_sell, "With Qty: ", stkInfo_object.Qty)
597.                     print("Because: Price below upper band:", usd(MA))
598.                     set_event(e)
599.                     send_msg(client_packet)
600.                     wait_for_an_event(e)
601.                     ### close trade logic
```

```
602.             response_list = []
603.             while not q.empty():
604.                 response_data = get_response(q)
605.                 response_list.append(response_data)
606.                 client_config.orders.append(response_data)
607.             #             Trade_object = stkInfo_object.Tradelist[-1]
608.             #             Trade_object.CloseTrade(response_list)
609.             #             stkInfo_object.PnLlist.append(Trade_object.PnL)
610.             stkInfo_object.Qty = 0
611.             stkInfo_object.position = 0
612.
613.             time.sleep(1) ### request order book every sec
```

3. ClientFrontEnd.py

```
1. # -*- coding: utf-8 -*-
2. """
3. Created on Mon Oct 19 22:39:06 2020
4.
5. @author: 63438
6. """
7. import pandas as pd
8. from flask import Flask, render_template, request
9.
10. from BBModel import stk_select, build_trading_model, populate_backtest_data, backTest, probation_test, execute_sql_statement, populate_strategy_bt
11. from BBD_client import *
12.
13. from network import PacketTypes, Packet
14. import queue
15. import threading
16. from socket import AF_INET, socket, SOCK_STREAM, IPPROTO_TCP, TCP_NODELAY
17.
18.
19. def usd(value):
20.     """Format value as USD."""
21.     return f"${value:,.2f}"
22.
23. def FrontEnd(engine):
24.     app = Flask(__name__)
25.     selected_list = []
26.     @app.route('/')
27.     def index():
28.         return render_template("index.html")
29.
30.
31.     @app.route('/Stock_selection')
32.     def Stock_selection():
33.         slist, sdf = stk_select()
34.         for i in slist:
35.             selected_list.append(i)
36.         stk_df = pd.DataFrame(sdf)
37.         stk_df['symbol'] = stk_df.index
38.         stk_df['std'] = stk_df['std'].map('{:.4f}'.format)
39.         stk_df = stk_df.transpose()
40.         list_of_stk = [stk_df[i] for i in stk_df]
41.         return render_template("stock_selection.html", stock_list=list_of_stk)
42.
43.
```

```
44.     @app.route('/Train_model')
45.     def train_model():
46.         stocksdf = build_trading_model(selected_list)
47.         populate_backtest_data(stocksdf)
48.         stocksdf['Profit_Loss_in_Training'] = stocksdf['Profit_Loss_in_Training']
49.     ].map('${:,.2f}'.format)
50.     stocksdf = stocksdf.transpose()
51.     train_results = [stocksdf[i] for i in stocksdf]
52.     return render_template("train_model.html", train_list=train_results)
53.
54.     @app.route('/Back_test')
55.     def model_back_testing():
56.         conn = engine.connect()
57.         backTest()
58.         populate_strategy_bt()
59.         result_df = execute_sql_statement("SELECT * from Stocks", conn)
60.         total = result_df['Profit_Loss'].sum()
61.         result_df['Profit_Loss'] = result_df['Profit_Loss'].map('${:,.2f}'.format)
62.     t)
63.         result_df = result_df.transpose()
64.         trade_results = [result_df[i] for i in result_df]
65.         return render_template("back_test_result.html", bt_list=trade_results, total=total)
66.
67.     @app.route('/Probation_test', methods = ['POST', 'GET'])
68.     def model_probation_testing():
69.         if request.method == 'POST':
70.
71.             form_input = request.form
72.             probation_testing_start_date = form_input['Start Date']
73.             probation_testing_end_date = form_input['End Date']
74.
75.             result_df = probation_test(probation_testing_start_date, probation_testing_end_date)
76.             total = result_df['Profit_Loss'].sum()
77.             result_df['Profit_Loss'] = result_df['Profit_Loss'].map('${:,.2f}'.format)
78.             result_df = result_df.transpose()
79.             trade_results = [result_df[i] for i in result_df]
80.             return render_template("probation_test_result.html", trade_list=trade_results, total=total)
81.         else:
82.
83.             return render_template("probation_test.html")
84.
85.     @app.route('/start_trading', methods = ['POST', 'GET'])
86.     def start_trading():
87.         if request.method == 'POST':
88.             if request.form.get("Auto Trading"):
89.                 client_config.client_socket = socket(AF_INET, SOCK_STREAM)
90.                 client_config.client_socket.setsockopt(IPPROTO_TCP, TCP_NODELAY,
91.                 True)
92.                 status = client_config.client_socket.connect_ex(client_config.ADDR)
93.                 if status != 0:
94.                     print("Fail in connecting to server")
95.                     sys.exit(0)
96.
97.             ## send thread and receive thread, thread is also object
```



```
97.         client_config.client_receiver = threading.Thread(target=receive,
98.             args=(trading_queue, trading_event))
99.         client_config.client_sender = threading.Thread(target=LogonAndTrade, args=(trading_queue, trading_event))
100.         client_config.client_receiver.start()
101.         client_config.client_sender.start()
102.
103.         while not client_config.trade_complete:
104.             continue
105.             # client_packet = Packet()
106.             # send_msg(get_stock_list(client_packet))
107.             # data1 = get_response(trading_queue)
108.             return render_template("auto_trading.html",trading_results=client_config.orders, total=usd(client_config.PnL)) ##client_config.orders
109.             elif request.form.get("Analysis"):
110.                 return render_template("auto_trading_analysis.html",trading_results=client_config.Ticker_PnL)
111.             else:
112.                 return render_template("start_trading.html")
113.
114.
115. @app.route('/manual_trading',methods = ['POST', 'GET'])
116. def manual_trading():
117.     if request.method == 'POST':
118.         form_input = request.form
119.         print(form_input)
120.         trading_id = form_input['OrderId']
121.         trading_Ticker = form_input['Symbol']
122.         trading_Side = form_input['Side']
123.         trading_Price = form_input['Price']
124.         trading_Quantity = form_input['Quantity']
125.         ### edit the packet
126.         client_packet = Packet()
127.         client_msg = new_order(client_packet, trading_id, trading_Ticker, 'Lmt', trading_Side, trading_Price, trading_Quantity)
128.         send_msg(client_msg)
129.         data = get_response(trading_queue)
130.         while not trading_queue.empty():
131.             client_config.orders.append(get_response(trading_queue))
132.         return render_template("trading_results.html", trading_results=data)
133.     else:
134.
135.         return render_template("manual_trading.html")
136.
137. @app.route('/client_down')
138. def client_down():
139.     client_packet = Packet()
140.     msg_data = {}
141.     try:
142.         send_msg(client_quit(client_packet))
143.         msg_type, msg_data = trading_queue.get()
144.         trading_queue.task_done()
145.         print(msg_data)
146.         return render_template("client_down.html", server_response=msg_data)
147.     except (OSError, Exception):
148.         print(msg_data)
```

```
149.                 return render_template("client_down.html", server_response=m
    sg_data)
150.
151.                 return app
152.
```

4. FRE_server5.py

```
1.  # -*- coding: utf-8 -*-
2.  #!/usr/bin/env python3
3.  #@ Copyright - Song Tang
4.
5.  import json
6.  import urllib.request
7.  import sys
8.  import pandas as pd
9.  import random
10. import numpy as np
11.
12. from socket import AF_INET, socket, SOCK_STREAM, IPPROTO_TCP, TCP_NODELAY, getho
    stname, gethostbyname
13. import threading
14. import queue
15.
16. from sqlalchemy import create_engine
17. from sqlalchemy import MetaData
18.
19. import sched, time
20. import datetime
21.
22. from pandas.tseries.holiday import USFederalHolidayCalendar, GoodFriday
23. from pandas.tseries.offsets import CustomBusinessDay
24.
25. from network import PacketTypes, Packet
26.
27. import pandas_market_calendars as mcal
28.
29. serverID = "Server"
30. engine = create_engine('sqlite:///BBDTradingServer.db')
31. conn = engine.connect()
32. conn.execute("PRAGMA foreign_keys = ON")
33.
34. metadata = MetaData()
35. metadata.reflect(bind=engine)
36.
37. mutex = threading.Lock()
38.
39. requestURL = "https://eodhistoricaldata.com/api/eod/"
40. myEodKey = "5ba84ea974ab42.45160048"
41. defaultStartDate = "2020-06-01"
42. defaultEndDate = "2020-09-30"
43. def get_daily_data(symbol, startDate=defaultStartDate, endDate=defaultEndDate, a
    piKey=myEodKey):
44.     symbolURL = str(symbol) + ".US?"
45.     startDateURL = "from=" + str(startDate)
46.     endDateURL = "to=" + str(endDate)
47.     apiKeyURL = "api_token=" + apiKey
48.     completeURL = requestURL + symbolURL + '&' + endDateURL + '&'
        + apiKeyURL + '&period=d&fmt=json'
```

```
49.     print(completeURL)
50.     with urllib.request.urlopen(completeURL) as req:
51.         data = json.load(req)
52.         return data
53.
54. def populate_stock_data(tickers, engine, table_name, stock_market_periods):
55.     column_names = ['symbol', 'date', 'open', 'high', 'low', 'close', 'adjusted_
close', 'volume']
56.     price_data = []
57.     for ticker in tickers:
58.         stock = get_daily_data(ticker, stock_market_periods[ticker][0], stock_ma
rket_periods[ticker][len(stock_market_periods[ticker])-1])
59.         for stock_data in stock:
60.             price_data.append([ticker, stock_data['date'], stock_data['open'], s
tock_data['high'], stock_data['low'], \
61.                               stock_data['close'], stock_data['adjusted_close']
, stock_data['volume']])
62.         print(price_data)
63.     stocks = pd.DataFrame(price_data, columns=column_names)
64.     stocks.to_sql(table_name, con=engine, if_exists='replace', index=False)
65.
66.
67. intradayRequestURL = "https://eodhistoricaldata.com/api/intraday/"
68. myEodKey = "5ba84ea974ab42.45160048"
69. defaultStartSeconds = "1585800000"
70. defaultEndSeconds = "1585886400"
71. def get_intraday_data(symbol, startTime=defaultStartSeconds, endTime=defaultEndS
econds, apiKey=myEodKey):
72.     symbolURL = str(symbol) + ".US?"
73.     startDateURL = "from=" + str(startTime)
74.     endDateURL = "to=" + str(endTime)
75.     apiKeyURL = "api_token=" + apiKey
76.     completeURL = intradayRequestURL + symbolURL + startDateURL + "&" + endDateU
RL + "&" + apiKeyURL + "&period=d&fmt=json"
77.     with urllib.request.urlopen(completeURL) as req:
78.         data = json.load(req)
79.         return data
80.
81. def populate_intraday_stock_data(tickers, engine, days_in_seconds):
82.     column_names = ['datetime', 'symbol', 'open', 'high', 'low', 'close', 'volum
e']
83.     for ticker in tickers:
84.         stock = get_intraday_data(ticker, days_in_seconds[0], days_in_seconds[le
n(days_in_seconds)-1])
85.         print(stock)
86.         price_data = []
87.         for stock_data in stock:
88.             if ((stock_data['open'] is not None and stock_data['open'] > 0) and
89.                 (stock_data['high'] is not None and stock_data['high'] > 0) and
90.                 (stock_data['low'] is not None and stock_data['low'] > 0) and
91.                 (stock_data['close'] is not None and stock_data['close'] > 0) )an
d
92.                 (stock_data['volume'] is not None and stock_data['volume'] > 0))
:
93.             price_data.append([stock_data['datetime'], ticker, stock_data['o
pen'], stock_data['high'], stock_data['low'], \
94.                               stock_data['close'], stock_data['volume']])
95.
```

```
96.         print(price_data)
97.         stocks = pd.DataFrame(price_data, columns=column_names)
98.         stocks = stocks.dropna()
99.         stocks.to_sql(ticker, con=engine, if_exists='replace', index=False)
100.
101.     def populate_intraday_order_map(symbols, engine, market_periods):
102.         for i in range(len(market_periods)):
103.             intraday_order_map[market_periods[i]] = []
104.
105.             stock_market_periods = {}
106.             for symbol in symbols:
107.                 stock_market_periods[symbol] = []
108.                 select_st = "SELECT * FROM " + symbol + ";"
109.                 result_set = engine.execute(select_st)
110.                 result_df = pd.DataFrame(result_set.fetchall())
111.                 result_df.columns = result_set.keys()
112.
113.                 for i in range(len(market_periods)):
114.                     if (result_df['datetime'].str.contains(market_periods[i])).a
ny():
115.                         mask = (result_df['datetime'].str.contains(market_peri
s[i])) & (result_df['symbol'] == symbol)
116.                         result = result_df.loc[(mask.values)]
117.                         intraday_order_map[market_periods[i]].append(result[['sy
mbol', 'open', 'high', 'low', 'close', 'volume']])
118.                         stock_market_periods[symbol].append(market_periods[i])
119.
120.             #print(intraday_order_map, file = intraday_order_file)
121.
122.             return intraday_order_map, stock_market_periods
123.
124.
125.     def accept_incoming_connections(q=None):
126.         while True:
127.             try:
128.                 client, client_address = fre_server.accept()
129.                 print("%s:%s has connected." % client_address, file=server_o
utput)
130.                 client_thread = threading.Thread(target=handle_client, args=
(client,q))
131.                 client_thread.setDaemon(True)
132.                 client_thread.start()
133.             except (KeyError, KeyboardInterrupt, SystemExit, Exception):
134.                 print("Exception in accept_incoming_connections\n", file=ser
ver_output)
135.                 q.put(Exception("accept_incoming_connections"))
136.                 break
137.
138.
139.     def receive(client_socket):
140.         total_client_request = b''
141.         msgSize = 0
142.         while True:
143.             try:
144.                 client_request = client_socket.recv(buf_size)
145.                 list_client_requests = []
146.                 if len(client_request) > 0:
147.                     total_client_request += client_request
148.                     msgSize = len(total_client_request)
149.                     while msgSize > 0:
150.                         if msgSize > 12:
```

```
151.         client_packet = Packet()
152.         client_request = client_packet.deserialize(total
    _client_request)
153.         #print(client_packet.m_msg_size, msgSize, len(c
    ient_request), file=server_output)
154.         if msgSize > 12 and client_packet.m_data_size <= msg
    Size:
155.             #data = json.loads(client_packet.m_data)
156.             #print(type(data), data, file=server_output)
157.             total_client_request = total_client_request[clie
    nt_packet.m_data_size:]
158.             msgSize = len(total_client_request)
159.             client_request = b''
160.             list_client_requests.append(client_packet)
161.         else:
162.             client_request = client_socket.recv(buf_size)
163.             total_client_request += client_request
164.             msgSize = len(total_client_request)
165.
166.         return list_client_requests
167.
168.     except (OSError, Exception):
169.         del clients[client_socket]
170.         print("Exception in receive\n", file=server_output)
171.         q.put(Exception("receive"))
172.         #raise Exception('receive')
173.         break
174.
175.
176.     def handle_client(client, q=None):
177.         global symbols
178.         global market_period
179.         while True:
180.             try:
181.                 list_client_requests = receive(client)
182.                 for client_request in list_client_requests:
183.                     msg_data = json.loads(client_request.m_data)
184.                     msg_type = client_request.m_type
185.                     print(msg_data, file=server_output)
186.                     clientID = msg_data["Client"]
187.
188.                     server_packet = Packet()
189.
190.                     if msg_type == PacketTypes.CONNECTION_REQ.value:
191.                         server_packet.m_type = PacketTypes.CONNECTION_RSP.va
    lue
192.                         if (clientID in clients.values()):
193.                             text = "%s duplicated connection request!" % cli
    entID
194.                             server_msg = json.dumps({'Server': serverID, 'Re
    sponse': text, 'Status': 'Rejected'})
195.                         else:
196.                             client_symbols = list(msg_data["Symbol"].split('
    ,'))
197.                             if all(symbol in symbols for symbol in client_sy
    mbols):
198.                                 text = "Welcome %s!" % clientID
199.                                 server_msg = json.dumps({'Server': serverID,
    'Response': text, 'Status': 'Ack'})
200.                                 clients[client] = clientID
201.                             else:
```

```
202.                text = "%s Not all your symbols are eligibl
    e!" % clientID
203.                server_msg = json.dumps({'Server': serverID
    , 'Response': text, 'Status': 'Rejected'})
204.                server_packet.m_data = server_msg
205.                client.send(server_packet.serialize())
206.                data = json.loads(server_packet.m_data)
207.                print(data, file=server_output)
208.
209.                elif msg_type == PacketTypes.END_REQ.value:
210.                    text = "%s left!" % clientID
211.                    server_msg = json.dumps({'Server':serverID, 'Respons
    e':text, 'Status':'Done'})
212.                    server_packet.m_type = PacketTypes.END_RSP.value
213.                    server_packet.m_data = server_msg
214.                    client.send(server_packet.serialize())
215.                    data = json.loads(server_packet.m_data)
216.                    print(data, file=server_output)
217.
218.                elif msg_type == PacketTypes.CLIENT_LIST_REQ.value:
219.                    user_list = str('')
220.                    for clientKey in clients:
221.                        user_list += clients[clientKey] + str(',')
222.                        print(clients[clientKey], file=server_output)
223.                    server_msg = json.dumps({'Client List':user_list})
224.                    server_packet.m_type = PacketTypes.CLIENT_LIST_RSP.v
    alue
225.                    server_packet.m_data = server_msg
226.                    client.send(server_packet.serialize())
227.                    data = json.loads(server_packet.m_data)
228.                    print(data, file=server_output)
229.
230.                elif msg_type == PacketTypes.STOCK_LIST_REQ.value:
231.                    stock_list = ','.join(symbols)
232.                    server_msg = json.dumps({"Stock List":stock_list})
233.                    server_packet.m_type = PacketTypes.STOCK_LIST_RSP.va
    lue
234.                    server_packet.m_data = server_msg
235.                    client.send(server_packet.serialize())
236.                    data = json.loads(server_packet.m_data)
237.                    print(data, file=server_output)
238.
239.                elif msg_type == PacketTypes.SERVER_DOWN_REQ.value:
240.                    server_msg = json.dumps({'Server':serverID, 'Status'
    : 'Server Down Confirmed'})
241.                    server_packet.m_type = PacketTypes.SERVER_DOWN_RSP.v
    alue
242.                    server_packet.m_data = server_msg
243.                    client.send(server_packet.serialize())
244.                    data = json.loads(server_packet.m_data)
245.                    print(data, file=server_output)
246.
247.                elif msg_type == PacketTypes.BOOK_INQUIRY_REQ.value:
248.                    server_packet.m_type = PacketTypes.BOOK_INQUIRY_RSP.
    value
249.                    if "Symbol" in msg_data and msg_data["Symbol"] != ""
    :
250.                        if order_table.empty:
251.                            print("Server order book is empty\n", file=s
    erver_output)
252.                            text = "Server order book is empty"
```

```
253.         server_msg = json.dumps({'Server':serverID,
254.         'Response':text, 'Status':'Done'})
255.         else:
256.             server_msg = json.dumps(order_table.loc[order_
257.             r_table['Symbol'].isin(list(msg_data["Symbol"].split(',')
258.             table'))
259.             else:
260.                 print("Bad message, missing symbol\n", file=server_
261.                 er_output)
262.                 text = "Bad message, missing symbol"
263.                 server_msg = json.dumps({'Server':serverID, 'Res
264.                 ponse':text, 'Status':'Done'})
265.                 server_packet.m_data = server_msg
266.                 client.send(server_packet.serialize())
267.                 data = json.loads(server_packet.m_data)
268.                 print(data, file=server_output)
269.
270.             elif msg_type == PacketTypes.MARKET_STATUS_REQ.value:
271.                 server_packet.m_type = PacketTypes.MARKET_STATUS_RSP
272.                 .value
273.                 server_msg = json.dumps({'Server':serverID, 'Status'
274.                 :market_status, 'Market_Period':market_period})
275.                 server_packet.m_data = server_msg
276.                 client.send(server_packet.serialize())
277.                 data = json.loads(server_packet.m_data)
278.                 print(data, file=server_output)
279.
280.             elif msg_type == PacketTypes.NEW_ORDER_REQ.value:
281.                 server_packet.m_type = PacketTypes.NEW_ORDER_RSP.val
282.                 ue
283.
284.                 if market_status == "Market Closed":
285.                     msg_data["Status"] = "Order Reject"
286.                     server_msg = json.dumps(msg_data)
287.                     server_packet.m_data = server_msg
288.                     client.send(server_packet.serialize())
289.                     data = json.loads(server_packet.m_data)
290.                     print(data, file=server_output)
291.
292.                 mutex.acquire()
293.
294.                 #TODO Possible issue with price comparison
295.                 #....
296.                 Actually it is due to floating comparison:
297.
298.                 row["Price"] <= float(msg_data["Price"])
299.
300.                 On the order book:
301.
302.                 'OrderIndex': 226, 'Symbol': 'XOM', 'Side': 'Sel
303.                 l', 'Price': 39.45, 'Qty': 2632036, 'Status': 'New'
304.
305.                 When an order at 39.45 was sent,
306.
307.                 {'Client': 'client1', 'OrderIndex': '1', 'Status
308.                 ': 'New Order', 'Symbol': 'XOM', 'Type': 'Lmt', 'Side': 'Buy', 'Price': '39.45',
309.                 'Qty': '100'}
310.
311.                 {'Client': 'client1', 'OrderIndex': '1', 'Status
312.                 ': 'Order Reject', 'Symbol': 'XOM', 'Type': 'Lmt', 'Side': 'Buy', 'Price': 39.45
313.                 , 'Qty': '100'}
```

```
301.
302.                                     As the price is floating number, actually it cou
ld be a little less than 39.45, so the order was rejected.
303.
304.                                     When an order at 39.46 was sent, we got a fill:
305.
306.                                     {'Client': 'client1', 'OrderIndex': '2', 'Status'
: 'New Order', 'Symbol': 'XOM', 'Type': 'Lmt', 'Side': 'Buy', 'Price': '39.46',
'Qty': '100'}
307.                                     {'Client': 'client1', 'OrderIndex': '2', 'Status
': 'Order Fill', 'Symbol': 'XOM', 'Type': 'Lmt', 'Side': 'Buy', 'Price': 39.45,
'Qty': '100', 'ServerOrderID': 226}
308.
309.                                     So the best way is to convert the price to integ
er for comparison
310.                                     ...
311.
312.                                     if (("Symbol" not in msg_data) or (msg_data["Symbol"
] == "")) or \
313.                                     (("Side" not in msg_data) or (msg_data["Side"] =
= "")) or \
314.                                     (("Type" not in msg_data) or (msg_data["Type"] =
= "")) or \
315.                                     (("Price" not in msg_data) or (msg_data["Type"]
== "Lmt" and msg_data["Price"] == "")) or \
316.                                     (("Qty" not in msg_data) or (msg_data["Qty"] ==
"")) or int(msg_data["Qty"]) < 1):
317.                                     print("Bad message, missing critical data item\n
", file=server_output)
318.                                     text = "Bad message, missing critial item"
319.                                     server_msg = json.dumps({'Server':serverID, 'Res
ponse':text, 'Status':'Done'})
320.                                     server_packet.m_type = PacketTypes.NEW_ORDER_RSP
.value
321.                                     server_packet.m_data = server_msg
322.                                     client.send(server_packet.serialize())
323.                                     data = json.loads(server_packet.m_data)
324.                                     print(data, file=server_output)
325.
326.                                     if msg_data["Type"] == "Lmt":
327.
328.                                     msg_order_qty = int(msg_data["Qty"])
329.
330.                                     for (index, row) in order_table.iterrows():
331.                                         if msg_data["Status"] == "Order Fill":
332.                                             break
333.
334.                                     if ((row["Symbol"] == msg_data["Symbol"]) an
d
335.                                     (row["Side"] != msg_data["Side"]) and
336.                                     (row["Price"] <= float(msg_data["Price"]
) if msg_data["Side"] == "Buy" else row["Price"] >= float(msg_data["Price"]))) an
d
337.                                     (row["Status"] != "Filled") & (row["Statu
s"] != "Open Trade") & (int(row["Qty"]) > 0)):
338.                                         order_qty = int(row['Qty'])
339.
340.                                     if (order_qty == msg_order_qty):
341.                                         order_table.loc[index, 'Qty'] = 0
342.                                         order_table.loc[index, 'Status'] = '
Filled'
```



```
343.                                     msg_data["Price"] = round(order_tabl
    e.loc[index, 'Price'], 2)
344.                                     msg_data['Qty'] = str(msg_order_qty)
345.                                     msg_data["Status"] = "Order Fill"
346.
347.                                     elif (order_qty< msg_order_qty):
348.                                         order_table.loc[index, 'Qty'] = 0
349.                                         order_table.loc[index, 'Status'] = '
        Filled'
350.                                     msg_data["Price"] = round(order_tabl
    e.loc[index, 'Price'], 2)
351.                                     msg_data['Qty'] = str(order_qty)
352.                                     msg_data["Status"] = "Order Partial
        Fill"
353.                                     msg_order_qty -= order_qty
354.                                     else:
355.                                         order_table.loc[index, 'Qty'] -
        = msg_order_qty
356.                                         order_table.loc[index, 'Status'] = '
        Partial Filled'
357.                                     msg_data["Price"] = round(order_tabl
    e.loc[index, 'Price'], 2)
358.                                     msg_data['Qty'] = str(msg_order_qty)
359.                                     msg_data["Status"] = "Order Fill"
360.
361.                                     msg_data["ServerOrderID"] = order_table.
    loc[index, 'OrderIndex']
362.                                     server_msg = json.dumps(msg_data)
363.                                     server_packet.m_type = PacketTypes.NEW_O
    RDER_RSP.value
364.                                     server_packet.m_data = server_msg
365.                                     client.send(server_packet.serialize())
366.                                     data = json.loads(server_packet.m_data)
367.                                     print(data, file=server_output)
368.
369.                                     if msg_data["Status"] == "New Order":
370.                                         msg_data["Status"] = "Order Reject"
371.                                         msg_data["Price"] = round(float(msg_data["Pr
        ice"]), 2)
372.                                         server_msg = json.dumps(msg_data)
373.                                         server_packet.m_type = PacketTypes.NEW_ORDER
        _RSP.value
374.                                         server_packet.m_data = server_msg
375.                                         client.send(server_packet.serialize())
376.                                         data = json.loads(server_packet.m_data)
377.                                         print(data, file=server_output)
378.
379.                                     elif msg_data["Type"] == "Mkt":
380.
381.                                         msg_order_qty = int(msg_data["Qty"])
382.
383.                                         while ((order_table["Symbol"] == msg_data["Symbo
        l"]) &
384.                                             (order_table["Side"] != msg_data["Side"]) &
385.                                             (order_table["Status"] != "Filled") & (order
        _table["Status"] != "Open Trade" ) & \
386.                                             (order_table['Qty'] != 0)).any():
```

```
387.
388.             if msg_data["Status"] == "Order Fill":
389.                 break
390.
391.             mask = (order_table["Symbol"] == msg_data["S
392. ymbol"]) & \
393.                 (order_table["Side"] != msg_data["Si
394. de"]) & \
395.                 (order_table["Status"] != "Filled")
396.             & (order_table["Status"] != "Open Trade" ) & \
397.                 (order_table['Qty'] != 0)
398.
399.             index = -1
400.             order_qty = 0
401.             if msg_data["Side"] == "Sell":
402.                 index = order_table.loc[(mask.values), '
403. Price'].idxmax()
404.                 order_qty = int(order_table.loc[index, '
405. Qty'])
406.             else:
407.                 index = order_table.loc[(mask.values), '
408. Price'].idxmin()
409.                 order_qty = int(order_table.loc[index, '
410. Qty'])
411.
412.             if (order_qty == msg_order_qty):
413.                 order_table.loc[index, 'Qty'] = 0
414.                 order_table.loc[index, 'Status'] = 'Fill
415. ed'
416.                 msg_data["Price"] = round(order_table.lo
417. c[index, 'Price'], 2)
418.                 msg_data['Qty'] = str(msg_order_qty)
419.                 msg_data["Status"] = "Order Fill"
420.
421.             elif (order_qty < msg_order_qty):
422.                 order_table.loc[index, 'Qty'] = 0
423.                 order_table.loc[index, 'Status'] = 'Fill
424. ed'
425.                 msg_data['Qty'] = str(order_qty)
426.                 msg_data["Price"] = round(order_table.lo
427. c[index, 'Price'], 2)
428.                 msg_data["Status"] = "Order Partial Fill
429. "
430.                 msg_order_qty -= order_qty
431.             else:
432.                 order_table.loc[index, 'Qty'] -
433. = msg_order_qty
434.                 order_table.loc[index, 'Status'] = 'Part
435. ial Filled'
436.                 msg_data["Price"] = round(order_table.lo
437. c[index, 'Price'], 2)
438.                 msg_data['Qty'] = str(msg_order_qty)
439.                 msg_data["Status"] = "Order Fill"
440.
441.             msg_data["ServerOrderID"] = order_table.loc[
442. index, 'OrderIndex']
443.             server_msg = json.dumps(msg_data)
444.             server_packet.m_type = PacketTypes.NEW_ORDER
445.
446.             _RSP.value
447.             server_packet.m_data = server_msg
448.             client.send(server_packet.serialize())
```

```
431.         data = json.loads(server_packet.m_data)
432.         print(data, file=server_output)
433.
434.         if msg_data["Status"] == "New Order":
435.             msg_data["Status"] = "Order Reject"
436.             server_msg = json.dumps(msg_data)
437.             server_packet.m_type = PacketTypes.NEW_ORDER
438.             _RSP.value
439.             server_packet.m_data = server_msg
440.             client.send(server_packet.serialize())
441.             data = json.loads(server_packet.m_data)
442.             print(data, file=server_output)
443.         else:
444.             msg_data["Status"] = "Order Reject"
445.             server_msg = json.dumps(msg_data)
446.             server_packet.m_type = PacketTypes.NEW_ORDER_RSP
447.             .value
448.             server_packet.m_data = server_msg
449.             client.send(server_packet.serialize())
450.             data = json.loads(server_packet.m_data)
451.             print(data, file=server_output)
452.             mutex.release()
453.
454.         else:
455.             print("Unknown Message from Client\n", file=server_o
456.             utput)
457.             text = "Unknown Message from Client"
458.             server_msg = json.dumps({"Server":serverID, "Respons
459.             e":text, "Status":"Done"})
460.             print(server_msg, file=server_output)
461.             server_packet.m_type = PacketTypes.END_RSP.value
462.
463.             server_packet.m_data = server_msg
464.             client.send(server_packet.serialize())
465.             data = json.loads(server_packet.m_data)
466.             print(data, file=server_output)
467.
468.             if (server_packet.m_type == PacketTypes.END_RSP.value or
469.             (server_packet.m_type == PacketTypes.CONNECTION_RSP.value and \
470.             data['Status'] == "Rejected")):
471.                 client.close()
472.                 if server_packet.m_type == PacketTypes.END_RSP.value
473.                 :
474.                     del clients[client]
475.                     users = ''
476.                     for clientKey in clients:
477.                         users += clients[clientKey] + ' '
478.                     print(users, file=server_output)
479.                     return
480.             elif server_packet.m_type == PacketTypes.SERVER_DOWN_RSP
481.             .value:
482.                 Exception("Server Down")
483.             except (KeyboardInterrupt, KeyError, Exception):
484.                 print("Exception in handle client", file=server_output)
485.                 q.put(Exception("handle_client"))
486.                 client.close()
487.                 sys.exit(0)
488.             except json.decoder.JSONDecodeError:
489.                 q.put(Exception("handle_client"))
```

```
485.         client.close()
486.         sys.exit(0)
487.
488.     def get_stock_list():
489.         enginestk = create_engine('sqlite:///TradingBBD.db',connect_args={'c
heck_same_thread': False})
490.         connstk = enginestk.connect()
491.         stockdf = pd.read_sql_query("SELECT * from Stocks", connstk)
492.         stklist = list(stockdf.Ticker)
493.         return stklist
494.
495.     def generate_qty(number_of_qty):
496.         total_qty = 0
497.         list_of_qty = []
498.         for index in range(number_of_qty):
499.             qty = random.randint(1,101)
500.             list_of_qty.append(qty)
501.             total_qty += qty
502.         return np.array(list_of_qty)/total_qty
503.
504.     def populate_order_table(symbols, start, end):
505.         #price_scale = 0.05
506.         #price_unit = 100
507.         global order_index
508.         global order_table
509.         global market_status
510.
511.         if (market_status == "Open" or market_status == "Pending Closing"):
512.
513.             return
514.
515.             symbol_list = ','.join('"' + symbol + '"' for symbol in symbols)
516.
517.             select_st = "SELECT * FROM FRE_Stocks WHERE date >= " + "\"" + start
+ "\"" + " AND date <= " + "\"" + end + "\"" + " AND symbol in (" + symbol_list
+ ");"
518.             result_set = engine.execute(select_st)
519.             result_df = pd.DataFrame(result_set.fetchall())
520.             result_df.columns = result_set.keys()
521.
522.             order_table.drop(order_table.index, inplace=True)
523.             list_of_qty_map = {}
524.             max_number_orders = 0
525.             for index, stock_data in result_df.iterrows():
526.
527.                 if stock_data['open'] > high_price_min:
528.                     price_scale = high_price_scale
529.                 else:
530.                     price_scale = low_price_scale
531.
532.                 list_of_qty = generate_qty(int((float(stock_data['high'])-
float(stock_data['low']))/price_scale))
533.                 list_of_qty_map[stock_data['symbol']] = list_of_qty
534.                 if max_number_orders < len(list_of_qty):
535.                     max_number_orders = len(list_of_qty)
536.
537.             if mutex.locked() is True:
538.                 print("Is locked!")
539.
540.             mutex.acquire()
```

```
540.         order_table.fillna(0)
541.         order_index = 0
542.
543.         #print(list_of_qty_map)
544.         #print(max_number_orders)
545.
546.         for index in range(0, max_number_orders-1, 2):
547.             for i, stock_data in result_df.iterrows():
548.
549.                 if index >= len(list_of_qty_map[stock_data['symbol']]):
550.                     #print(i, index, list_of_qty_map[stock_data['symbol']])
551.
552.                     continue
553.
554.                     buy_price = float(stock_data['low']);
555.                     sell_price = float(stock_data['high'])
556.                     daily_volume = float(stock_data['volume'])
557.                     open_price = float(stock_data['open'])
558.                     close_price = float(stock_data['close'])
559.
560.                     if stock_data['open'] > high_price_min:
561.                         price_scale = high_price_scale
562.                     else:
563.                         price_scale = low_price_scale
564.
565.                     buy_price = open_price - (index+1) * price_scale * (random.r
566. andint(1, price_unit)/price_unit)
567.                     buy_price = float("{:.2f}".format(buy_price))
568.                     qty = float(list_of_qty_map[stock_data['symbol']][index])
569.
570.                     order_index += 1
571.                     order_qty = int(qty*daily_volume*(random.randint(1, price_un
572. it)/price_unit))
573.                     if index == 0:
574.                         order_table.loc[order_index] = ['Srv_' + market_period +
575. '_' + str(order_index), stock_data['symbol'], open_price, close_price,
576. 'Buy' if random.randint(1,11) % 2 == 0 el
577. se 'Sell', open_price, 0, order_qty, 'Open Trade']
578.                     else:
579.                         order_table.loc[order_index] = ['Srv_' + market_period +
580. '_' + str(order_index), stock_data['symbol'], open_price, close_price,
581. 'Buy', buy_price, order_qty, order_qty, 'N
582. ew']
583.
584.                     sell_price = open_price + (index+1) * price_scale * (random.
585. randint(1, price_unit)/price_unit)
586.                     sell_price = float("{:.2f}".format(sell_price))
587.                     qty = float(list_of_qty_map[stock_data['symbol']][index])
588.
589.                     order_index += 1
590.                     order_qty = int(qty*daily_volume*(random.randint(1, price_un
591. it)/price_unit))
592.                     order_table.loc[order_index] = ['Srv_' + market_period + '_'
593. + str(order_index), stock_data['symbol'], open_price, close_price,
594. 'Sell', sell_price, order_qty, order_qty, 'Ne
595. w']
596.
597.                     order_table = order_table.sort_values(['Side', 'Symbol', 'Price', 'Q
598. ty'])
599.
600.                     mutex.release()
```

```
589.  
590.         print(order_table)  
591.  
592.  
593.         def create_market_interest(symbols):  
594.  
595.             global market_period  
596.             global order_table  
597.             global order_index  
598.             #print(market_status, market_period, "No new market interest")  
599.  
600.             while True:  
601.                 time.sleep(order_interval_time)  
602.                 if len(order_table) != 0 and market_status == 'Open':  
603.  
604.                     for i in range(len(symbols)):  
605.  
606.                         # Some stocks may have fewer intraday data than others,  
607.                         # it could be empty while other stocks still create intr  
608.                         day orders  
609.                         if intraday_order_map[market_period][i].empty == True:  
610.                             continue  
611.                             symbol = symbols[i]  
612.  
613.                             try:  
614.  
615.                                 mutex.acquire()  
616.  
617.                                 ### BUY logic  
618.                                 if ((order_table['Symbol'] == symbol) & (order_table  
619.                                     ['Side'] == 'Buy')).any():  
620.                                     mask = (order_table['Symbol'] == symbol) & (orde  
621.                                     r_table['Side'] == 'Buy')  
622.                                     best_buy_index = order_table.loc[(mask.values),  
623.                                     'Price'].idxmax()  
624.                                     close_price = intraday_order_map[market_period][  
625.                                     i].iloc[0]['close']  
626.                                     open_price = intraday_order_map[market_period][i  
627.                                     ].iloc[0]['open']  
628.                                     new_buy_price = intraday_order_map[market_period  
629.                                     ][i].iloc[0]['low']  
630.                                     new_buy_price = float("{:.2f}".format(new_buy_pr  
631.                                     ice))  
632.                                     new_buy_qty = intraday_order_map[market_period][  
633.                                     i].iloc[0]['volume']/2  
634.                                     new_buy_qty = int(new_buy_qty * random.uniform(0  
635.                                     ,1))  
636.                                     order_index += 1  
637.                                     order_table.loc[order_index] = ['Srv_' + market_  
638.                                     period + '_' + str(order_index), symbol, open_price, close_price,  
639.                                     'Buy', new_buy_price, new  
640.                                     _buy_qty, new_buy_qty, 'New']  
641.  
642.                                     print(order_table.loc[order_index])  
643.  
644.                                     #### Sell
```

```

637.                                     if ((order_table['Symbol'] == symbol) & (order_table
638. [ 'Side' ] == 'Sell')).any():
639.                                     mask = (order_table['Symbol'] == symbol) & (orde
640. r_table['Side'] == 'Sell')
641.                                     best_sell_index = order_table.loc[(mask.values),
642. 'Price'].idxmin()
643.                                     close_price = intraday_order_map[market_period][i
644. i].iloc[0]['close']
645.                                     open_price = intraday_order_map[market_period][i
646. ].iloc[0]['open']
647.                                     new_sell_price = intraday_order_map[market_perio
648. d][i].iloc[0]['high']
649.                                     new_sell_price = float("{:.2f}".format(new_sell_
650. price))
651.                                     new_sell_qty = intraday_order_map[market_period]
652. [i].iloc[0]['volume']/2
653.                                     new_sell_qty = int(new_sell_qty * random.uniform
654. (0,1))
655.                                     order_index += 1
656.                                     order_table.loc[order_index] = ['Srv_' + market_
657. period + '_' + str(order_index), symbol, open_price, close_price,
658. 'Sell', new_sell_price, n
659. ew_sell_qty, new_sell_qty, 'New']
660.
661.                                     print(order_table.loc[order_index])
662.
663.                                     intraday_order_map[market_period][i].drop(intraday_o
664. rder_map[market_period][i].index[0] ,inplace=True)
665.
666.                                     while ((order_table['Symbol'] == symbol) & (order_ta
667. ble['Qty'] != 0)).any():
668.                                     buy_mask = (order_table['Symbol'] == symbol) & (
669. order_table['Qty'] != 0) & (order_table['Side'] == 'Buy')
670.                                     sell_mask = (order_table['Symbol'] == symbol) &
671. (order_table['Qty'] != 0) & (order_table['Side'] == 'Sell')
672.                                     buy_prices = order_table.loc[(buy_mask.values),
673. 'Price']
674.                                     sell_prices = order_table.loc[(sell_mask.values)
675. , 'Price']
676.
677.                                     if buy_prices.empty == False and sell_prices.emp
678. ty == False:
679.                                     best_buy_index = buy_prices.idxmax()
680.                                     best_sell_index = sell_prices.idxmin()
681.                                     best_buy_price = order_table.loc[best_buy_in
682. dex, 'Price']
683.                                     best_sell_price = order_table.loc[best_sell_
684. index, 'Price']
685.
686.                                     #TODO Avoid floating point issue
687.                                     if best_buy_price >= best_sell_price:
688.
689.                                     if order_table.loc[best_buy_index, 'Qty'
690. ] == order_table.loc[best_sell_index, 'Qty']:
691.                                     order_table.loc[best_buy_index, 'Qty
692. ' ] = 0
693.                                     order_table.loc[best_buy_index, 'Sta
694. tus'] = 'Filled'

```

```
674.                                     order_table.loc[best_sell_index, 'Qty'] = 0
675.                                     order_table.loc[best_sell_index, 'Status'] = 'Filled'
676.
677.                                     elif order_table.loc[best_buy_index, 'Qty'] > order_table.loc[best_sell_index, 'Qty']:
678.                                         order_table.loc[best_buy_index, 'Qty'] -= order_table.loc[best_sell_index, 'Qty']
679.                                         order_table.loc[best_buy_index, 'Status'] = 'Partial Filled'
680.                                     order_table.loc[best_sell_index, 'Qty'] = 0
681.                                     order_table.loc[best_sell_index, 'Status'] = 'Filled'
682.
683.                                     else:
684.                                         order_table.loc[best_sell_index, 'Qty'] -= order_table.loc[best_buy_index, 'Qty']
685.                                         order_table.loc[best_sell_index, 'Status'] = 'Partial Filled'
686.                                         order_table.loc[best_buy_index, 'Qty'] = 0
687.                                         order_table.loc[best_buy_index, 'Status'] = 'Filled'
688.
689.                                     else:
690.                                         order_table = order_table.sort_values(['Side', 'Symbol', 'Price', 'Qty'])
691.                                         print(order_table)
692.                                         break
693.
694.                                     else:
695.                                         order_table = order_table.sort_values(['Side', 'Symbol', 'Price', 'Qty'])
696.                                         print(order_table)
697.                                         break
698.
699.                                     mutex.release()
700.
701.                                     #except (KeyboardInterrupt):
702.                                     except Exception as e:
703.                                         print("Except in create market interest")
704.                                         print(e)
705.                                         if mutex.locked() == True:
706.                                             print("Still locked")
707.                                             mutex.release()
708.                                         #sys.exit(0)
709.
710.
711.                                     #TODO! The logic need to be optimized
712.                                     def close_trades(symbols):
713.                                         global order_index
714.                                         global order_table
715.                                         for symbol in symbols:
716.                                             side = 'Buy' if random.randint(1,11) % 2 == 0 else 'Sell'
717.                                             if ((order_table['Symbol'] == symbol) & (order_table['Side'] == side)).any():
718.                                                 mask = (order_table['Symbol'] == symbol) & (order_table['Side'] == side) & (order_table['Qty'] != 0)
719.                                                 if side == 'Buy':
```



```
720.         buy_prices = order_table.loc[(mask.values), 'Price']
721.         if buy_prices.empty == False:
722.             best_buy_index = buy_prices.idxmax()
723.             order_table.loc[best_buy_index, 'Qty'] = 0
724.             order_table.loc[best_buy_index, 'Status'] = 'Close Trade'
725.         else:
726.             open_price = order_table.loc[order_index-
727.             1, 'Open']
728.             close_price = order_table.loc[order_index-
729.             1, 'Close']
730.             qty = order_table.loc[order_index, 'OrigQty']
731.             order_index += 1
732.             order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), symbol, open_price, close_price,
733.             'Buy', close_price, 0, qty, 'Close Trade']
734.         else:
735.             sell_prices = order_table.loc[(mask.values), 'Price']
736.             if sell_prices.empty == False:
737.                 best_sell_index = sell_prices.idxmin()
738.                 order_table.loc[best_sell_index, 'Qty'] = 0
739.                 order_table.loc[best_sell_index, 'Status'] = 'Close Trade'
740.             else:
741.                 open_price = order_table.loc[order_index-
742.                 1, 'Open']
743.                 close_price = order_table.loc[order_index-
744.                 1, 'Close']
745.                 qty = order_table.loc[order_index, 'OrigQty']
746.                 order_index += 1
747.                 order_table.loc[order_index] = ['Srv_' + market_period + '_' + str(order_index), symbol, open_price, close_price,
748.                 'Sell', close_price, 0, qty, 'Close Trade']
749.
750.         order_table = order_table.sort_values(['Side', 'Symbol', 'Price', 'Qty'])
751.         print(order_table)
752.         #print(order_table, file = order_table_file)
753.
754.     def update_market_status(status, day):
755.         global market_status
756.         global order_index
757.         global order_table
758.         market_status = status
759.
760.         global symbols
761.         global market_periods
762.         global market_period
763.
764.         print("day=", day)
765.         print(market_status, market_periods[day])
766.         market_period = market_periods[day]
767.
768.         populate_order_table(symbols, market_periods[day], market_periods[day])
769.         market_status = 'Open'
770.         print(market_status)
```

```
769.         time.sleep(market_open_time)
770.         market_status = 'Pending Closing'
771.         print(market_status)
772.         time.sleep(market_pending_close_time)
773.         market_status = 'Market Closed'
774.         print(market_status)
775.         close_trades(symbols)
776.         time.sleep(market_close_time)
777.
778.     def set_market_status(scheduler, time_in_seconds):
779.         value = datetime.datetime.fromtimestamp(time_in_seconds)
780.         print(value.strftime('%Y-%m-%d %H:%M:%S'))
781.         for day in range(total_market_days):
782.             scheduler.enter((market_close_time+market_open_time+market_pending_close_time)*day+1,1, update_market_status, argument=('Pending Open',day))
783.             scheduler.run()
784.
785.     port = 6510
786.     buf_size = 4096
787.     fre_server = socket(AF_INET, SOCK_STREAM)
788.     fre_server.setsockopt(IPPROTO_TCP, TCP_NODELAY, True)
789.     print(gethostname())
790.     fre_server.bind((gethostbyname(""), port))
791.
792.     location_of_pairs = 'csv/PairTrading.csv'
793.     stock_table_name = "FRE_Stocks"
794.     market_open_time = 75
795.     market_pending_close_time = 3
796.     market_close_time = 10
797.     order_interval_time = 1
798.     low_price_scale = 0.01
799.     high_price_scale = 1
800.     high_price_min = 1000
801.     price_unit = 100
802.     intraday_order_map = {}
803.
804.     clients = {}
805.
806.     if __name__ == "__main__":
807.
808.         #server_output = open("server_output.txt", "w")
809.         server_output = sys.stderr
810.
811.         q = queue.Queue()
812.
813.         total_market_days = 4 # intraday data range is 30 days away from today
814.         order_index = 0
815.
816.         symbols = get_stock_list()
817.         order_table_columns = ['OrderIndex', 'Symbol', 'Open', 'Close', 'Side', 'Price', 'Qty', 'OrigQty', 'Status']
818.         order_table = pd.DataFrame(columns=order_table_columns)
819.         order_table = order_table.fillna(0)
820.         order_table = pd.DataFrame(columns=order_table_columns)
821.         order_table = order_table.fillna(0)
822.         order_table['Price'] = order_table['Price'].astype(float)
823.         order_table['Open'] = order_table['Open'].astype(float)
824.         order_table['Close'] = order_table['Close'].astype(float)
825.         order_table['Qty'] = order_table['Qty'].astype(int)
826.         order_table['OrigQty'] = order_table['OrigQty'].astype(int)
```

```
827.
828.         # USFederalHolidayCalendar has a bug, GoodFriday is not excluded
829.         us_bd = CustomBusinessDay(holidays=['2020-04-
10'], calendar=USFederalHolidayCalendar())
830.
831.         lastBusDay = datetime.datetime.today()
832.         if datetime.date.weekday(lastBusDay) == 5:         #if it's Saturday
833.             lastBusDay = lastBusDay - datetime.timedelta(days = 1) #then mak
e it Friday
834.         elif datetime.date.weekday(lastBusDay) == 6:         #if it's Sunday
835.             lastBusDay = lastBusDay - datetime.timedelta(days = 2); #then ma
ke it Friday
836.
837.         end_date = lastBusDay - datetime.timedelta(days = 1) # day before la
st trading day
838.
839.         #end_date = datetime.datetime.today() - datetime.timedelta(days = 1)
# yesterday
840.         start_date = end_date + datetime.timedelta(-total_market_days)
841.
842.         #market_periods = pd.DatetimeIndex(pd.date_range(start=start_date.st
rftime("%Y-%m-%d"), end=end_date.strftime("%Y-%m-%d"), freq=us_bd)).strftime("%Y
-%m-%d").tolist()
843.         trading_calendar = mcal.get_calendar('NYSE')
844.         market_periods = trading_calendar.schedule(start_date=start_date.str
ftime("%Y-%m-%d"), \
845.                                                     end_date=end_date.strftim
e("%Y-%m-%d")).index.strftime("%Y-%m-%d").tolist()
846.
847.         print(market_periods)
848.         total_market_days = len(market_periods) # Update for remove non-
trading days
849.
850.         #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()
851.         market_period_objects = trading_calendar.schedule(start_date=start_d
ate.strftime("%Y-%m-%d"), end_date=end_date.strftime("%Y-%m-%d")).index.tolist()

852.
853.
854.         market_period_seconds = []
855.         for i in range(len(market_period_objects)):
856.             market_period_seconds.append(int(time.mktime(market_period_objec
ts[i].timetuple())) # As timestamp is 12am of each day
857.             market_period_seconds.append(int(time.mktime(market_period_objects[1
en(market_period_objects)-
1].timetuple()))+24*3600) # For last day intraday data
858.             #print(market_period_objects)
859.             #print(market_period_seconds)
860.             populate_intraday_stock_data(symbols, engine, market_period_seconds)

861.
862.         intraday_order_map, stock_market_periods = populate_intraday_order_m
ap(symbols, engine, market_periods)
863.         print(intraday_order_map)
864.
865.         print(stock_market_periods)
866.         for value in stock_market_periods.values():
867.             if total_market_days > len(value):
868.                 total_market_days = len(value)
```

```
869.             market_periods = value
870.
871.             print(market_periods)
872.
873.             populate_stock_data(symbols, engine, stock_table_name, stock_market_
periods)
874.
875.             fre_server.listen(1)
876.             print("Waiting for client requests")
877.             try:
878.                 scheduler = sched.scheduler(time.time, time.sleep)
879.                 current_time_in_seconds = time.time()
880.                 scheduler_thread = threading.Thread(target=set_market_status, ar
gs=(scheduler, current_time_in_seconds))
881.                 #scheduler_thread.setDaemon(True)
882.
883.                 server_thread = threading.Thread(target=accept_incoming_connecti
ons, args=(q,))
884.                 create_market_thread = threading.Thread(target=create_market_int
erest, args=(symbols,))
885.                 #server_thread.setDaemon(True)
886.
887.                 scheduler_thread.start()
888.                 server_thread.start()
889.                 create_market_thread.start()
890.
891.                 error = q.get()
892.                 q.task_done()
893.                 if error is not None:
894.                     raise error
895.
896.                 scheduler_thread.join()
897.                 server_thread.join()
898.
899.                 fre_server.close()
900.                 sys.exit(0)
901.
902.             except (KeyError, KeyboardInterrupt, SystemExit, Exception):
903.                 print("Exception in main\n")
904.                 fre_server.close()
905.                 sys.exit(0)
```

5. network.py

```
1. from enum import Enum
2. import struct
3.
4.
5. class PacketTypes(Enum):
6.     CONNECTION_NONE = 0
7.     CONNECTION_REQ = 1 # request
8.     CONNECTION_RSP = 2 # response
9.     CLIENT_LIST_REQ = 3
10.    CLIENT_LIST_RSP = 4
11.    STOCK_LIST_REQ = 5
12.    STOCK_LIST_RSP = 6
13.    STOCK_REQ = 7
14.    STOCK_RSP = 8
15.    BOOK_INQUIRY_REQ = 9
```

```
16.     BOOK_INQUIRY_RSP = 10
17.     NEW_ORDER_REQ = 11
18.     NEW_ORDER_RSP = 12
19.     MARKET_STATUS_REQ = 13
20.     MARKET_STATUS_RSP = 14
21.     END_REQ = 15
22.     END_RSP = 16
23.     SERVER_DOWN_REQ = 17
24.     SERVER_DOWN_RSP = 18
25.
26.
27. class Packet:
28.     def __init__(self): ### three member
29.         self.m_type = 0
30.         self.m_msg_size = 0
31.         self.m_data_size = 0
32.         self.m_data = ""
33.
34.     def __str__(self):
35.         return str(self.__class__) + ": " + str(self.__dict__) + "\n"
36.
37.     def __repr__(self):
38.         return str(self.__class__) + ": " + str(self.__dict__) + "\n"
39.
40.     def serialize(self):
41.         self.m_data_size = 12 + len(self.m_data)
42.         self.m_msg_size = self.m_data_size
43.         return self.m_type.to_bytes(4, byteorder='little') + \
44.             self.m_msg_size.to_bytes(4, byteorder='little') + \
45.             self.m_data_size.to_bytes(4, byteorder='little') + \
46.             bytes(self.m_data, 'utf-8')
47.
48.     def deserialize(self, message):
49.         msg_len = len(message)
50.         msg_unpack_string = '<iii' + str(msg_len - 12) + 's' ### first three as
int and following the string
51.         self.m_type, self.m_msg_size, self.m_data_size, msg_data = struct.unpack
(msg_unpack_string, message)
52.         self.m_data = msg_data[0:self.m_data_size - 12].decode('utf-8')
53.         return message[self.m_data_size:]
```

6. main.py

```
1. # -*- coding: utf-8 -*-
2. """
3. Created on Mon Oct 19 23:16:42 2020
4.
5. @author: 63438
6. """
7.
8. import json
9. import datetime as dt
10. from dateutil import tz
11. from dateutil.relativedelta import relativedelta
12. import urllib.request
13. import pandas as pd
14. import numpy as np
15. from sqlalchemy import Column, ForeignKey, Integer, Float, String
16. from sqlalchemy import create_engine
```

```
17. from sqlalchemy import MetaData
18. from sqlalchemy import Table
19. from sqlalchemy import inspect
20. import ClientFrontEnd
21. from BBD_client import Client_Manual
22.
23.
24.
25. engine = create_engine('sqlite:///TradingBBD.db',connect_args={'check_same_thread': False})
26. conn = engine.connect()
27. conn.execute("PRAGMA foreign_keys = ON")
28.
29. metadata = MetaData()
30. metadata.reflect(bind=engine)
31.
32.
33. end_date = dt.datetime.today().date()
34. start_date = end_date - relativedelta(weeks=1)
35.
36. ### This week we dont do FrontEnd but Manual connect
37. FrontEnd = True
38. Manual_Client_Connect = False
39.
40. if __name__ == "__main__":
41.     if FrontEnd:
42.         app = ClientFrontEnd.FrontEnd(engine)
43.         app.run()
44.     if Manual_Client_Connect:
45.         Client_Manual()
```

7. base.html

```
1. <!DOCTYPE html>
2. <html lang="en">
3.     <head>
4.
5.         <meta charset="UTF-8">
6.         <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no">
7.
8.         <title>FRE Platform: {% block title %}{% endblock %}</title>
9.         <link href="https://v4-
alpha.getbootstrap.com/dist/css/bootstrap.min.css" rel="stylesheet">
10.        <link href="https://v4-
alpha.getbootstrap.com/examples/dashboard/dashboard.css" rel="stylesheet">
11.
12.        <link href="/static/favicon.ico" rel="icon">
13.        <link href="/static/styles.css" rel="stylesheet">
14.
15.        <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
16.        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd
/popper.min.js"></script>
17.        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstra
p.min.js"></script>
18.
19.    </head>
20.
21.    <body>
```

```
22.     <nav class="navbar navbar-toggleable-md navbar-custom fixed-top navbar-  
dark border">  
23.         <a class="navbar-  
brand" href="/"><span class="blue"><b>FRE </b></span><span class="green"><b>Plat  
form</b></span></a>  
24.         <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">  
25.             <span class="navbar-toggler-icon"></span>  
26.         </button>  
27.         <div class="collapse navbar-collapse" id="navbar">  
28.             </div>  
29.  
30.     </nav>  
31.  
32.     {% if get_flashed_messages() %}  
33.         <header>  
34.             <div class="alert alert-primary border text-  
center" role="alert">  
35.                 {{ get_flashed_messages() | join(" ") }}  
36.             </div>  
37.         </header>  
38.     {% endif %}  
39.  
40.     <div class="container-fluid">  
41.         <div class="row">  
42.             {% block content %} {% endblock %}  
43.         </div>  
44.     </div>  
45.  
46.     <footer class="small text-center text-muted">  
47.         Developed by <a href="https://engineering.nyu.edu/academics/departme  
nts/finance-and-risk-  
engineering">NYU FRE Department</a>. View <a href="https://engineering.nyu.edu/a  
cademics/departments/finance-and-risk-engineering">NYU FRE Terms of Use</a>.  
48.     </footer>  
49.  
50. </body>  
51.  
52. </html>
```

8. index.html

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

```

13. {% extends "base.html" %}
14. {% block content %}
15. <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
16.     <ul class="nav nav-pills flex-column">
17.         <li class="nav-item">
18.             <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
19.         </li>
20.         <li class="nav-item">
21.             <a class="nav-link" href="/Train_model">Train Model</a>
22.         </li>
23.         <li class="nav-item">
24.             <a class="nav-link" href="/Back_test">Back Testing</a>
25.         </li>
26.         <li class="nav-item">
27.             <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.         </li>
29.         <li class="nav-item">
30.             <a class="nav-link active" href="/start_trading">Start Trading</a>
31.         </li>
32.         <li class="nav-item">
33.             <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.         </li>
35.         <li class="nav-item">
36.             <a class="nav-link" href="/client_down">Client Down</a>
37.         </li>
38.     </ul>
39. </nav>
40.
41. <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.     <h3><strong> ' </strong></h3>
43.     <h3><strong>Bollinger Bands Trading Model</strong></h3>
44.     <div align="center">
45.         <table cellpadding="0" cellspacing="0" border="0" style="font-
46.             family:Georgia, Garamond, Serif;">
47.             <tbody>
48.                 <tr style="font-variant:small-caps;font-
49.                     style:normal;color:black;font-size:20px;">
50.                     <tr>
51.                         <td width="881" valign="top">
52.                             <p>
53.                                 Bollinger Bands are a type of price envelope develop
54.                                 ed by John BollingerOpens.
55.                                 (Price envelopes define upper and lower price range
56.                                 levels.) Bollinger Bands are envelopes plotted at a standard deviation level abo
57.                                 ve and below a simple moving average of the price.
58.                                 Because the distance of the bands is based on standa
59.                                 rd deviation, they adjust to volatility swings in the underlying price.
60.                             </p>
61.                         </td>
62.                         <td>
63.                             <tr>
64.                                 <tr>
65.                                     <td width="881" valign="top">

```



```

66.         <p>
67.             Within S&P500 component pool, select 10 stocks with
highest 5-minute volatility over the last month of the historical data part.
68.         </p>
69.     </td>
70. </tr>
71. <tr>
72.     <td width="881" valign="top">
73.         <p>
74.             Reason:
75.             1. S&P 500 components are with large size, high liq
uidity.
76.             2. Since using intraday data for generating trading
signals, we check the vol among minutes over one month (a short period)
77.             3. Higher volatility may trigger signals more often
, more signals can lead better test the efficacy of trading model (i.e. training
params)
78.         <p>
79.     </td>
80. </tr>
81. <tr style="font-variant:small-caps;font-
style:normal;color:black;font-size:20px;">
82.     <th width="881" valign="top">
83.         <p>
84.             <strong>Bollinger Bands Trading Logic</strong>
85.         <p>
86.     </th>
87. </tr>
88. <tr>
89.     <td width="881" valign="top">
90.         <p>
91.             1) X: The H-
period SMA. This serves as both the center of the BBs, and the baseline for dete
rmining the location of the two bands
92.         <p>
93.     </td>
94. </tr>
95. <tr>
96.     <td width="881" valign="top">
97.         <p>
98.             2) Up : The upper band line that is k1 (default is 2
) standard deviation from the H-minute SMA
99.         <p>
100.     </td>
101. </tr>
102. <tr>
103.     <td width="881" valign="top">
104.         <p>
105.             3) Down : The lower band line that is k1 sta
ndard deviation from the H-period SMA
106.         <p>
107.     </td>
108. </tr>
109. <tr>
110.     <td width="881" valign="top">
111.         <p>
112.             4) Open Trades:
113.             <br> • Open short position if Close >
114.             = Up, sell the stock

```

```

115.                                     <br>      • Open long position if Close <=
      Down, buy the stock
116.                                     </p>
117.                                 </td>
118.                             </tr>
119.                             <td width="881" valign="top">
120.                                 <p>
121.                                     5) Close Trades:
122.                                     <br>      • Close the open positions when
      Close return within Up and Down <br>
123.                                 </p>
124.                             </td>
125.                         </tr>
126.                         <tr style="font-variant:small-caps;font-
style:normal;color:black;font-size:20px;">
127.                             <th width="881" valign="top">
128.                                 <p>
129.                                     <strong>Back Testing</strong>
130.                                 </p>
131.                             </th>
132.                         </tr>
133.                         <tr>
134.                             <td width="881" valign="top">
135.                                 <p>
136.                                     Use the intraday 1m-
      price data from the recent 1 week, doing P&L calculation
137.                                 </p>
138.                             </td>
139.                         </tr>
140.                         <tr style="font-variant:small-caps;font-
style:normal;color:black;font-size:20px;">
141.                             <th width="881" valign="top">
142.                                 <p>
143.                                     <strong>Probation Testing</strong>
144.                                 </p>
145.                             </th>
146.                         </tr>
147.                         <tr>
148.                             <td width="881" valign="top">
149.                                 <p>
150.                                     Any two trading dates before current date co
      uld be used for conducting BB trading model
151.                                 </p>
152.                             </td>
153.                         </tr>
154.                     </tbody>
155.                 </table>
156.             </div>
157.
158.         </main>
159.         <br></br>
160.         <br></br>
161.
162.         {% endblock %}
163.     </body>
164. </html>

```

9. stock_selection.html

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <meta http-equiv="Content-Style-Type" content="text/css" />
5.     <meta name="viewport" content="width=device-width">
6.     <title>stock_selection.html</title>
7.     <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
8.     <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
9.     <script type="text/javascript" src="/library/js/headscripts.js"></script>
10.    <script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
11.  </head>
12.  <body>
13.    {% extends "base.html" %}
14.    {% block content %}
15.      <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
16.        <ul class="nav nav-pills flex-column">
17.          <li class="nav-item">
18.            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
19.          </li>
20.          <li class="nav-item">
21.            <a class="nav-link" href="/Train_model">Train Model</a>
22.          </li>
23.          <li class="nav-item">
24.            <a class="nav-link" href="/Back_test">Back Testing</a>
25.          </li>
26.          <li class="nav-item">
27.            <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.          </li>
29.          <li class="nav-item">
30.            <a class="nav-link active" href="/start_trading">Start Trading</a>
31.          </li>
32.          <li class="nav-item">
33.            <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.          </li>
35.          <li class="nav-item">
36.            <a class="nav-link" href="/client_down">Client Down</a>
37.          </li>
38.        </ul>
39.      </nav>
40.
41.      <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.        <h3>'</h3>
43.        <h3>Stock Selection within S&P500</h3>
44.        <div class="table-responsive">
45.          <table class="table table-striped">
46.            <thead>
47.              <tr>
48.                <th style="text-align:center">Symbol</th>
49.                <th style="text-align:center">Volatility</th>
50.              </tr>
51.            </thead>
52.            <tbody>
53.              {% for stock in stock_list %}
54.                <tr>
```

```
55.         <td> {{stock.symbol}}</td>
56.         <td> {{stock['std']}}</td>
57.     </tr>
58.     {% endfor %}
59. </tbody>
60. </table>
61. </div>
62. </main>
63. {% endblock %}
64. </body>
65. </html>
```

10. train_model.html

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <meta http-equiv="Content-Style-Type" content="text/css" />
5.     <meta name="viewport" content="width=device-width">
6.     <title>stock_selection.html</title>
7.     <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
8.     <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
9.     <script type="text/javascript" src="/library/js/headscripts.js"></script>
10.    <script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
11.  </head>
12.  <body>
13.    {% extends "base.html" %}
14.    {% block content %}
15.      <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
16.        <ul class="nav nav-pills flex-column">
17.          <li class="nav-item">
18.            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
19.          </li>
20.          <li class="nav-item">
21.            <a class="nav-link" href="/Train_model">Train Model</a>
22.          </li>
23.          <li class="nav-item">
24.            <a class="nav-link" href="/Back_test">Back Testing</a>
25.          </li>
26.          <li class="nav-item">
27.            <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.          </li>
29.          <li class="nav-item">
30.            <a class="nav-link active" href="/start_trading">Start Trading</a>
31.          </li>
32.          <li class="nav-item">
33.            <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.          </li>
35.          <li class="nav-item">
36.            <a class="nav-link" href="/client_down">Client Down</a>
37.          </li>
38.        </ul>
39.      </nav>
40.
41.      <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
```

```
42.     <h3>'</h3>
43.     <h3>Training Parameters for Selected Stocks</h3>
44.     <div class="table-responsive">
45.         <table class="table table-striped">
46.             <thead>
47.                 <tr>
48.                     <th style="text-align:center">Ticker</th>
49.                     <th style="text-align:center">H</th>
50.                     <th style="text-align:center">K1</th>
51.                     <th style="text-align:center">Notional</th>
52.                     <th style="text-align:center">PnL_in_Training</th>
53.                 </tr>
54.             </thead>
55.             <tbody>
56.                 {% for stock in train_list %}
57.                 <tr>
58.                     <td> {{stock.Ticker}}</td>
59.                     <td> {{stock.H}}</td>
60.                     <td> {{stock.K1}}</td>
61.                     <td> {{stock.Notional}}</td>
62.                     <td> {{stock.Profit_Loss_in_Training}}</td>
63.                 </tr>
64.                 {% endfor %}
65.             </tbody>
66.         </table>
67.     </div>
68. </main>
69. {% endblock %}
70. </body>
71. </html>
```

11. back_test_result.html

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

```
22.         </li>
23.         <li class="nav-item">
24.             <a class="nav-link" href="/Back_test">Back Testing</a>
25.         </li>
26.         <li class="nav-item">
27.             <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.         </li>
29.         <li class="nav-item">
30.             <a class="nav-link active" href="/start_trading">Start Trading</a>
31.         </li>
32.         <li class="nav-item">
33.             <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.         </li>
35.         <li class="nav-item">
36.             <a class="nav-link" href="/client_down">Client Down</a>
37.         </li>
38.     </ul>
39. </nav>
40.
41. <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.     <h3>'</h3>
43.     <h3>Back Test Analysis</h3>
44.     <div class="table-responsive">
45.         <table class="table table-striped">
46.             <thead>
47.                 <tr>
48.                     <th style="text-align:center">Ticker</th>
49.                     <th style="text-align:center">H</th>
50.                     <th style="text-align:center">K1</th>
51.                     <th style="text-align:center">Notional</th>
52.                     <th style="text-align:center">Profit</th>
53.                 </tr>
54.             </thead>
55.             <tbody>
56.                 {% for trade in bt_list %}
57.                 <tr>
58.                     <td> {{trade.Ticker}}</td>
59.                     <td> {{trade.H}}</td>
60.                     <td> {{trade.K1}}</td>
61.                     <td> {{trade.Notional}}</td>
62.                     <td> {{trade.Profit_Loss}}</td>
63.                 </tr>
64.                 {% endfor %}
65.             </tbody>
66.             <tfoot>
67.                 <tr>
68.                     <td colspan="5"></td>
69.                     <td>{{ total }}</td>
70.                 </tr>
71.             </tfoot>
72.         </table>
73.     </div>
74. </main>
75. {% endblock %}
76. </body>
77. </html>
```

12. probation_test.html

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <meta http-equiv="Content-Style-Type" content="text/css" />
5.     <meta name="viewport" content="width=device-width">
6.     <title>pair_trade_probation_test.html</title>
7.     <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
8.     <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
9.     <script type="text/javascript" src="/library/js/headscripts.js"></script>
10.    <script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
11.  </head>
12.  <body>
13.    {% extends "base.html" %}
14.    {% block content %}
15.      <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
16.        <ul class="nav nav-pills flex-column">
17.          <li class="nav-item">
18.            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
19.          </li>
20.          <li class="nav-item">
21.            <a class="nav-link" href="/Train_model">Train Model</a>
22.          </li>
23.          <li class="nav-item">
24.            <a class="nav-link" href="/Back_test">Back Testing</a>
25.          </li>
26.          <li class="nav-item">
27.            <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.          </li>
29.          <li class="nav-item">
30.            <a class="nav-link active" href="/start_trading">Start Trading</a>
31.          </li>
32.          <li class="nav-item">
33.            <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.          </li>
35.          <li class="nav-item">
36.            <a class="nav-link" href="/client_down">Client Down</a>
37.          </li>
38.        </ul>
39.      </nav>
40.
41.      <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.        <h3>'</h3>
43.        <h3>Start Probation Testing</h3>
44.        <hr>
45.        <div class="form-group">
46.          <form action="/Probation_test" id="buy" method="POST">
47.            <p>Start Date <input type = "text" name = "Start Date" placeholder="yyyy-mm-dd"></p>
48.            <p>End Date <input type = "text" name = "End Date" placeholder="yyyy-mm-dd"></p>
49.            <p><input type = "submit" value = "submit" /></p>
50.          </form>
51.        </div>
52.      </main>
```

```
53. {% endblock %}  
54. </body>  
55. </html>
```

13. probation_test_result.html

```
1. <!DOCTYPE html>  
2. <html>  
3.   <head>  
4.     <meta http-equiv="Content-Style-Type" content="text/css" />  
5.     <meta name="viewport" content="width=device-width">  
6.     <title>pair_trade_probation_test_result.html</title>  
7.     <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />  
8.     <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />  
9.     <script type="text/javascript" src="/library/js/headscripts.js"></script>  
10.    <script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>  
11.   </head>  
12.   <body>  
13.   {% extends "base.html" %}  
14.   {% block content %}  
15.     <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">  
16.       <ul class="nav nav-pills flex-column">  
17.         <li class="nav-item">  
18.           <a class="nav-link active" href="/Stock_selection">Stock Selection</a>  
19.         </li>  
20.         <li class="nav-item">  
21.           <a class="nav-link" href="/Train_model">Train Model</a>  
22.         </li>  
23.         <li class="nav-item">  
24.           <a class="nav-link" href="/Back_test">Back Testing</a>  
25.         </li>  
26.         <li class="nav-item">  
27.           <a class="nav-link" href="/Probation_test">Probation Testing</a>  
28.         </li>  
29.         <li class="nav-item">  
30.           <a class="nav-link active" href="/start_trading">Start Trading</a>  
31.         </li>  
32.         <li class="nav-item">  
33.           <a class="nav-link" href="/manual_trading">Manual Trading</a>  
34.         </li>  
35.         <li class="nav-item">  
36.           <a class="nav-link" href="/client_down">Client Down</a>  
37.         </li>  
38.       </ul>  
39.     </nav>  
40.     <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">  
41.       <h3></h3>  
42.       <h3>Probation Test Analysis</h3>  
43.       <div class="table-responsive">  
44.         <table class="table table-striped">  
45.           <thead>  
46.             <tr>  
47.               <th style="text-align:center">Ticker</th>  
48.               <th style="text-align:center">H</th>  
49.
```



```
50.         <th style="text-align:center">K1</th>
51.         <th style="text-align:center">Notional</th>
52.         <th style="text-align:center">Profit</th>
53.     </tr>
54. </thead>
55. <tbody>
56.     {% for trade in trade_list %}
57.     <tr>
58.         <td> {{trade.Ticker}}</td>
59.         <td> {{trade.H}}</td>
60.         <td> {{trade.K1}}</td>
61.         <td> {{trade.Notional}}</td>
62.         <td> {{trade.Profit_Loss}}</td>
63.     </tr>
64.     {% endfor %}
65. </tbody>
66. <tfoot>
67.     <tr>
68.         <td colspan="5"></td>
69.         <td>{{ total }}</td>
70.     </tr>
71. </tfoot>
72. </table>
73. </div>
74. </main>
75. {% endblock %}
76. </body>
77. </html>
```

14. start_trading.html

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

```
25.         </li>
26.         <li class="nav-item">
27.             <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.         </li>
29.         <li class="nav-item">
30.             <a class="nav-link active" href="/start_trading">Start Trading</a>
31.         </li>
32.         <li class="nav-item">
33.             <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.         </li>
35.         <li class="nav-item">
36.             <a class="nav-link" href="/client_down">Client Down</a>
37.         </li>
38.     </ul>
39. </nav>
40.
41. <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.     <h3>'</h3>
43.     <h3><strong>Simulated Trading</strong></h3>
44.     <div align="center">
45.         <table cellpadding="0" cellspacing="0" border="0" style="font-
family:Georgia, Garamond, Serif;">
46.             <tbody>
47.                 <tr style="font-variant:small-caps;font-
style:normal;color:black;font-size:20px;">
48.                     <tr>
49.                         <td width="881" valign="top">
50.                             <p>
51.                                 The simulated market is based on the latest 30 tradi
ng days of market data, for which the intraday trading data are available.
52.                                 Therefore, the market simulation will stop after 30
simulated trading days.
53.                             </p>
54.                         </td>
55.                     </tr>
56.                     <tr style="font-variant:small-caps;font-
style:normal;color:black;font-size:20px;">
57.                         <th width="881" valign="top">
58.                             <p>
59.                                 <strong>Definition of Trading Day and Market Status<
/strong>
60.                             </p>
61.                         </th>
62.                     </tr>
63.                     <tr>
64.                         <td width="881" valign="top">
65.                             <p>
66.                                 1) Each trading day is 60 seconds, followed by 10 se
conds of market close, and then a new trading day.
67.                             </p>
68.                         </td>
69.                     </tr>
70.                     <tr>
71.                         <td width="881" valign="top">
72.                             <p>
73.                                 2) Market open time = 50 seconds, status: Open.
74.                             </p>
75.                         </td>
76.                     </tr>
77.                     <tr>
78.                         <td width="881" valign="top">
```

```

79.                <p>
80.                3) Market pending close time = 10 seconds, status: P
    ending Closing.
81.                </p>
82.            </td>
83.        </tr>
84.        <tr>
85.            <td width="881" valign="top">
86.                <p>
87.                4) Market close time = 10 seconds, status: Market Cl
    osed.
88.                </p>
89.            </td>
90.        </tr>
91.        <tr style="font-variant:small-caps;font-
    style:normal;color:black;font-size:20px;">
92.            <th width="881" valign="top">
93.                <p>
94.                <strong>Daily Order Book - Historical Daily Market D
    ata, created before market open</strong>
95.                </p>
96.            </th>
97.        </tr>
98.        <tr>
99.            <td width="881" valign="top">
100.                <p>
101.                1) Number of orders = (high_price - low_pric
    e)/price scale,
102.                price_scale is either 0.05 or 5 based stock
    price >= 1000 or not.
103.                </p>
104.            </td>
105.        </tr>
106.        <tr>
107.            <td width="881" valign="top">
108.                <p>
109.                2) buy_price = open_price - price_scale * ra
    nd().
110.                </p>
111.            </td>
112.        </tr>
113.        <tr>
114.            <td width="881" valign="top">
115.                <p>
116.                3) sell_price = open_price + price_scale * r
    and().
117.                </p>
118.            </td>
119.        </tr>
120.        <tr>
121.            <td width="881" valign="top">
122.                <p>
123.                3) quantity = randomized, but sum of all the
    orders = daily volume.
124.                </p>
125.            </td>
126.        </tr>
127.        <td width="881" valign="top">
128.            <p>
129.            4) The book interested are populated before
    market open.

```

```

130.                </p>
131.            </td>
132.        </tr>
133.        <tr style="font-variant:small-caps;font-
style:normal;color:black;font-size:20px;">
134.            <th width="881" valign="top">
135.                <p>
136.                    <strong>Intraday Market Interests - Daily In
traday Market Data, during market open</strong>
137.                </p>
138.            </th>
139.        </tr>
140.        <tr>
141.            <td width="881" valign="top">
142.                <p>
143.                    1) Buy interests: Evey 5 min low price, 1/2
of the volume * rand().
144.                </p>
145.            </td>
146.        </tr>
147.        <tr>
148.            <td width="881" valign="top">
149.                <p>
150.                    2) Sell interests: Evey 5 min high price, 1/
2 of the volume * rand().
151.                </p>
152.            </td>
153.        </tr>
154.        <tr>
155.            <td width="881" valign="top">
156.                <p>
157.                    3) Any crossed interests will be traded.
158.                </p>
159.            </td>
160.        </tr>
161.        <tr>
162.            <td width="881" valign="top">
163.                <p>
164.                    4) The order book is sorted accorinding Side
, Symbol, Price and Quantity.
165.                </p>
166.            </td>
167.        </tr>
168.        <tr style="font-variant:small-caps;font-
style:normal;color:black;font-size:20px;">
169.            <th width="881" valign="top">
170.                <p>
171.                    <strong>Close Trade - either a buy or sell o
rder is filled</strong>
172.                </p>
173.            </th>
174.        </tr>
175.        <tr>
176.            <td width="881" valign="top">
177.                <p>
178.                    1) If buy side or sell side book is not empt
y, a best buy or best sell is filled.
179.                </p>
180.            </td>
181.        </tr>
182.        <tr>

```

```

183.         <td width="881" valign="top">
184.             <p>
185.                 2) If buy side or sell side is empty, the cl
                   ose trade for a buy or sell will be executed at daily market data closing price.
186.             </p>
187.         </td>
188.     </tr>
189.     <tr style="font-variant:small-caps;font-
style:normal;color:black;font-size:20px;">
190.         <th width="881" valign="top">
191.             <p>
192.                 <strong>Execution logic</strong>
193.             </p>
194.         </th>
195.     </tr>
196.     <tr>
197.         <td width="881" valign="top">
198.             <p>
199.                 1) If market is closed, new orders will be r
                   ejected.
200.             </p>
201.         </td>
202.     </tr>
203.     <tr>
204.         <td width="881" valign="top">
205.             <p>
206.                 2) While market is open or in pending closin
                   g.
207.             </p>
208.             <ul>
209.                 <li>Market orders - always filled from best
                   price.</li>
210.                 <li>Limit orders - will be filled at equal o
                   r better price.</li>
211.                 <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>
212.             </ul>
213.         </td>
214.     </tr>
215.     <tr>
216.         <td width="881" valign="top">
217.             <p>
218.                 3) Responses to new orders.
219.             </p>
220.             <ul>
221.                 <li>Order Fill</li>
222.                 <li>Order Partial Fill</li>
223.                 <li>Order Reject</li>
224.             </ul>
225.         </td>
226.     </tr>
227. </tbody>
228. </table>
229. </div>
230. <div class="form-group">
231.     <form action="/start_trading" id="buy" method="POST">
232.         <dl>
233.             <dd><input type = "submit" name = "Auto Trading" value = "Au
                   to Trading" /></dd>

```

```
234.         </dl>
235.     </form>
236. </div>
237. </main>
238. <br></br>
239. <br></br>
240.
241. {% endblock %}
242. </body>
243. </html>
```

15. auto_trading.html

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <meta http-equiv="Content-Style-Type" content="text/css" />
5.     <meta name="viewport" content="width=device-width">
6.     <title>auto_trading.html</title>
7.     <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
8.     <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
9.     <script type="text/javascript" src="/library/js/headscripts.js"></script>
10.    <script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script> <style>body { padding: 5px !important; }</style>
11.  </head>
12.  <body>
13.    {% extends "base.html" %}
14.    {% block content %}
15.      <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
16.        <ul class="nav nav-pills flex-column">
17.          <li class="nav-item">
18.            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
19.          </li>
20.          <li class="nav-item">
21.            <a class="nav-link" href="/Train_model">Train Model</a>
22.          </li>
23.          <li class="nav-item">
24.            <a class="nav-link" href="/Back_test">Back Testing</a>
25.          </li>
26.          <li class="nav-item">
27.            <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.          </li>
29.          <li class="nav-item">
30.            <a class="nav-link active" href="/start_trading">Start Trading</a>
31.          </li>
32.          <li class="nav-item">
33.            <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.          </li>
35.          <li class="nav-item">
36.            <a class="nav-link" href="/client_down">Client Down</a>
37.          </li>
38.        </ul>
39.      </nav>
40.
41.      <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.        <h3>'</h3>
```

```
43.     <h3><strong>Auto Trading Result</strong></h3>
44.     <div class="table-responsive">
45.         <table class="table table-striped">
46.             <thead>
47.                 <tr>
48.                     <th style="text-align:center">OrderIndex</th>
49.                     <th style="text-align:center">Status</th>
50.                     <th style="text-align:center">Symbol</th>
51.                     <th style="text-align:center">Type</th>
52.                     <th style="text-align:center">Side</th>
53.                     <th style="text-align:center">Price</th>
54.                     <th style="text-align:center">Qty</th>
55.                     <th style="text-align:center">ServerOrderID</th>
56.                 </tr>
57.             </thead>
58.             <tbody>
59.                 {% for trade in trading_results %}
60.                     <tr>
61.                         <td> {{trade.OrderIndex}} </td>
62.                         <td> {{trade.Status}} </td>
63.                         <td> {{trade.Symbol}} </td>
64.                         <td> {{trade.Type}} </td>
65.                         <td> {{trade.Side}} </td>
66.                         <td> {{trade.Price}} </td>
67.                         <td> {{trade.Qty}} </td>
68.                         <td> {{trade.ServerOrderID}} </td>
69.                     </tr>
70.                 {% endfor %}
71.             </tbody>
72.             <tfoot>
73.                 <tr>
74.                     <td colspan="7"></td>
75.                     <td>{{ total }}</td>
76.                 </tr>
77.             </tfoot>
78.         </table>
79.     </div>
80.     <div class="form-group">
81.         <form action="/start_trading" id="buy2" method="POST">
82.             <dl>
83.                 <dd><input type = "submit" name = "Analysis" value = "Analysis" /></dd>
84.             </dl>
85.         </form>
86.     </div>
87.     <hr>
88. </main>
89. {% endblock %}
90. </body>
91. </html>
```

16. auto_trading_analysis.html

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

```
7.     <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" me
    dia="all" />
8.     <link href="/library/skin/morpheus-
    nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
9.     <script type="text/javascript" src="/library/js/headscripts.js"></script>
10.    <script type="text/javascript" src="/media-gallery-tool/js/kaltura-
    upgrade.js"></script>    <style>body { padding: 5px !important; }</style>
11.    </head>
12.    <body>
13.    {% extends "base.html" %}
14.    {% block content %}
15.    <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
16.        <ul class="nav nav-pills flex-column">
17.            <li class="nav-item">
18.                <a class="nav-
    link active" href="/Stock_selection">Stock Selection</a>
19.            </li>
20.            <li class="nav-item">
21.                <a class="nav-link" href="/Train_model">Train Model</a>
22.            </li>
23.            <li class="nav-item">
24.                <a class="nav-link" href="/Back_test">Back Testing</a>
25.            </li>
26.            <li class="nav-item">
27.                <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.            </li>
29.            <li class="nav-item">
30.                <a class="nav-link active" href="/start_trading">Start Trading</a>
31.            </li>
32.            <li class="nav-item">
33.                <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.            </li>
35.            <li class="nav-item">
36.                <a class="nav-link" href="/client_down">Client Down</a>
37.            </li>
38.        </ul>
39.    </nav>
40.
41.    <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.        <h3></h3>
43.        <h3><strong>Auto Trading Analysis</strong></h3>
44.        <div class="table-responsive">
45.            <table class="table table-striped">
46.                <thead>
47.                    <tr>
48.                        <th style="text-align:center">Symbol</th>
49.                        <th style="text-align:center">ProfitLoss</th>
50.                    </tr>
51.                </thead>
52.                <tbody>
53.                    {% for key, value in trading_results.items() %}
54.                    <tr>
55.                        <th style="text-align:center">{{ key }}</th>
56.                        <td style="text-align:center">{{ value }}</td>
57.                    </tr>
58.                    {% endfor %}
59.                </tbody>
60.            </table>
61.        </div>
62.        <hr>
63.    </main>
```



```
64. {% endblock %}  
65. </body>  
66. </html>
```

17. manual_trading.html

```
1. <!DOCTYPE html>  
2. <html>  
3.   <head>  
4.     <meta http-equiv="Content-Style-Type" content="text/css" />  
5.     <meta name="viewport" content="width=device-width">  
6.     <title>manual_trading.html</title>  
7.     <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />  
8.     <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />  
9.     <script type="text/javascript" src="/library/js/headscripts.js"></script>  
10.    <script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>    <style>body { padding: 5px !important; }</style>  
11.   </head>  
12.   <body>  
13.     {% extends "base.html" %}  
14.     {% block content %}  
15.       <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">  
16.         <ul class="nav nav-pills flex-column">  
17.           <li class="nav-item">  
18.             <a class="nav-link active" href="/Stock_selection">Stock Selection</a>  
19.           </li>  
20.           <li class="nav-item">  
21.             <a class="nav-link" href="/Train_model">Train Model</a>  
22.           </li>  
23.           <li class="nav-item">  
24.             <a class="nav-link" href="/Back_test">Back Testing</a>  
25.           </li>  
26.           <li class="nav-item">  
27.             <a class="nav-link" href="/Probation_test">Probation Testing</a>  
28.           </li>  
29.           <li class="nav-item">  
30.             <a class="nav-link active" href="/start_trading">Start Trading</a>  
31.           </li>  
32.           <li class="nav-item">  
33.             <a class="nav-link" href="/manual_trading">Manual Trading</a>  
34.           </li>  
35.           <li class="nav-item">  
36.             <a class="nav-link" href="/client_down">Client Down</a>  
37.           </li>  
38.         </ul>  
39.       </nav>  
40.       <div class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">  
41.         <h3></h3>  
42.         <h3><strong>Manual Trading</strong></h3>  
43.         <hr>  
44.         <div class="form-group">  
45.           <form action="/manual_trading" id="buy" method="POST">  
46.             <dl>  
47.               <dt>OrderId</dt>
```

```
49.         <dd><input type = "number" name = "OrderId" min="1", step="1"></dd>
50.         <dt>Symbol</dt>
51.         <dd><input type = "text" name = "Symbol"></dd>
52.         <dt>Side</dt>
53.         <dd><input type = "text" name = "Side"></dd>
54.         <dt>Price</dt>
55.         <dd><input type = "number" name = "Price" min="0", step="0.01"></dd>
56.         <dt>Quantity</dt>
57.         <dd><input type = "number" name = "Quantity" min="10", step="1"></dd>
58.     >
59.         <dd><input type = "submit" value = "submit" /></dd>
60.     </dl>
61. </form>
62. </div>
63. </main>
64. {% endblock %}
65. </body>
66. </html>
```

18. trading_results.html

```
1. <!DOCTYPE html>
2. <html>
3.   <head>
4.     <meta http-equiv="Content-Style-Type" content="text/css" />
5.     <meta name="viewport" content="width=device-width">
6.     <title>trading_results.html</title>
7.     <link href="/library/skin/tool_base.css" type="text/css" rel="stylesheet" media="all" />
8.     <link href="/library/skin/morpheus-nyu/tool.css" type="text/css" rel="stylesheet" media="all" />
9.     <script type="text/javascript" src="/library/js/headscripts.js"></script>
10.    <script type="text/javascript" src="/media-gallery-tool/js/kaltura-upgrade.js"></script>
11.    <style>body { padding: 5px !important; }</style>
12.  </head>
13.  <body>
14.    {% extends "base.html" %}
15.    {% block content %}
16.      <nav class="col-sm-3 col-md-2 hidden-xs-down bg-faded sidebar">
17.        <ul class="nav nav-pills flex-column">
18.          <li class="nav-item">
19.            <a class="nav-link active" href="/Stock_selection">Stock Selection</a>
20.          </li>
21.          <li class="nav-item">
22.            <a class="nav-link" href="/Train_model">Train Model</a>
23.          </li>
24.          <li class="nav-item">
25.            <a class="nav-link" href="/Back_test">Back Testing</a>
26.          </li>
27.          <li class="nav-item">
28.            <a class="nav-link" href="/Probation_test">Probation Testing</a>
29.          </li>
30.          <li class="nav-item">
31.            <a class="nav-link active" href="/start_trading">Start Trading</a>
```

```
32.         <li class="nav-item">
33.             <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.         </li>
35.         <li class="nav-item">
36.             <a class="nav-link" href="/client_down">Client Down</a>
37.         </li>
38.     </ul>
39. </nav>
40.
41. <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.     <h3></h3>
43.     <h3><strong>Trading Result</strong></h3>
44.     <div align="center" class="table-responsive">
45.         <table class="table table-striped">
46.             <table border = 1>
47.                 {% for key, value in trading_results.items() %}
48.                     <tr>
49.                         <th>{{ key }}</th>
50.                         <td>{{ value }}</td>
51.                     </tr>
52.                 {% endfor %}
53.             </tbody>
54.         </table>
55.     </div>
56.     <hr>
57. </main>
58. {% endblock %}
59. </body>
60. </html>
```

19. client_down.html

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

```
24.         <a class="nav-link" href="/Back_test">Back Testing</a>
25.     </li>
26.     <li class="nav-item">
27.         <a class="nav-link" href="/Probation_test">Probation Testing</a>
28.     </li>
29.     <li class="nav-item">
30.         <a class="nav-link active" href="/start_trading">Start Trading</a>
31.     </li>
32.     <li class="nav-item">
33.         <a class="nav-link" href="/manual_trading">Manual Trading</a>
34.     </li>
35.     <li class="nav-item">
36.         <a class="nav-link" href="/client_down">Client Down</a>
37.     </li>
38. </ul>
39. </nav>
40.
41. <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-5">
42.     <h3>'</h3>
43.     <h3><strong>Client Down</strong></h3>
44.     <div align="center" class="table-responsive">
45.         <table class="table table-striped">
46.             <table border = 1>
47.                 <tbody>
48.                     {% for key, value in server_response.items() %}
49.                         <tr>
50.                             <th>{{ key }}</th>
51.                             <td>{{ value }}</td>
52.                         </tr>
53.                     {% endfor %}
54.                 </tbody>
55.             </table>
56.         </div>
57.         <hr>
58. </main>
59. {% endblock %}
60.
61. </body>
62. </html>
```