2024 LeekHarvester

Algorithm Trading System

Group 1：Zewei Yu，Qiang Hu，Jiarui Sun，Dexin Sun

2024. 04. 27

Contents

01

Overview

This system includes two main modules.

02

Backtest Module

Do backtest and moniter strategy.

03

Live Trading Module

Do live trading and tca.

Overview

A

Backtest Module

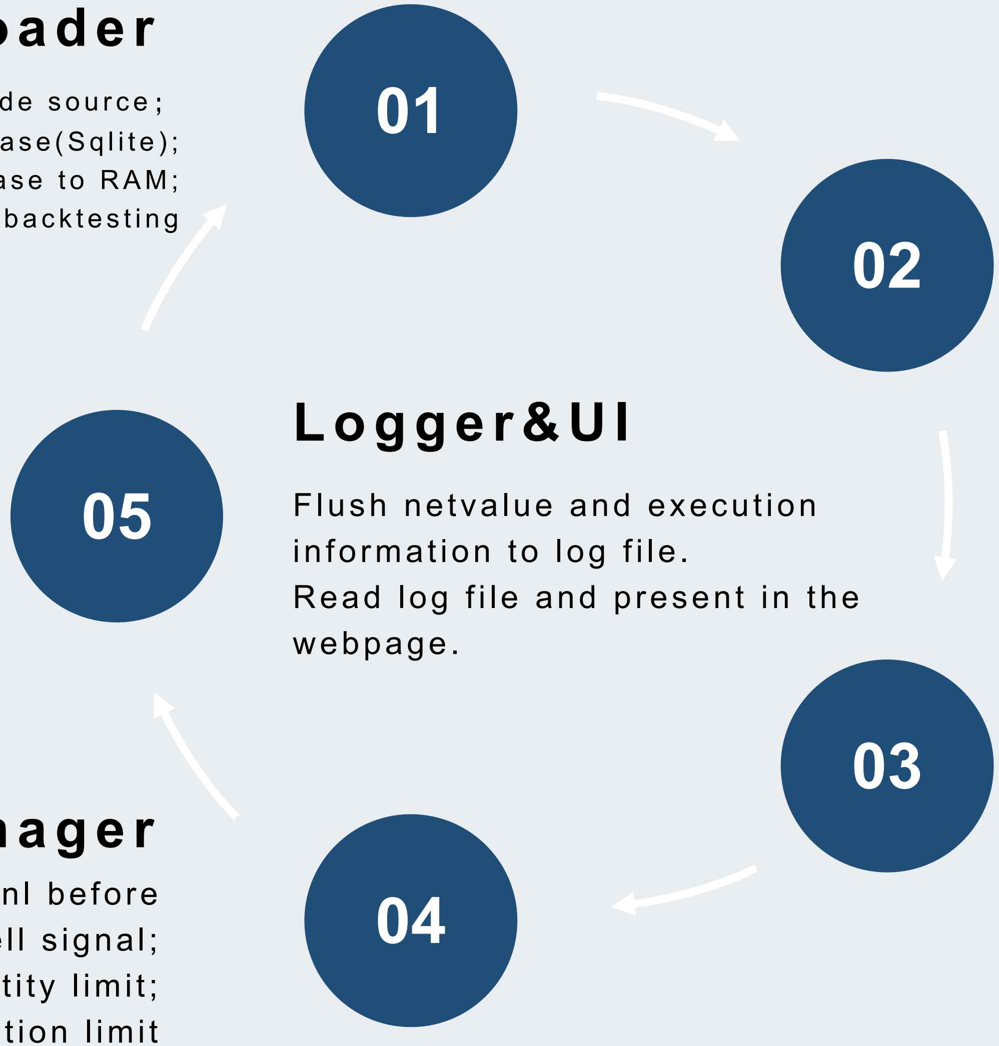
Data_Loader, Account, Order_Excection, Strategy, Backtest

B

Live Trading Module

TradingBot

Backtest



DataLoader

Get data from outside source;
Store data in local database(Sqlite);
Aquire data from database to RAM;
Process data requirement during backtesting

Account & Evaluation

Manage position and netvalue;
Get buy and sell signal and adjust the position;
Calculate the return metrics such as Sharpe Ratio, MaxDrawdown

OrderExecution

Process order signal from strategy;
Send buy or sell information (quantity,price)to Account

RiskManager

Check position and pnl before sending buy or sell signal;
Check order quantity limit;
Check execution limit

Strategy

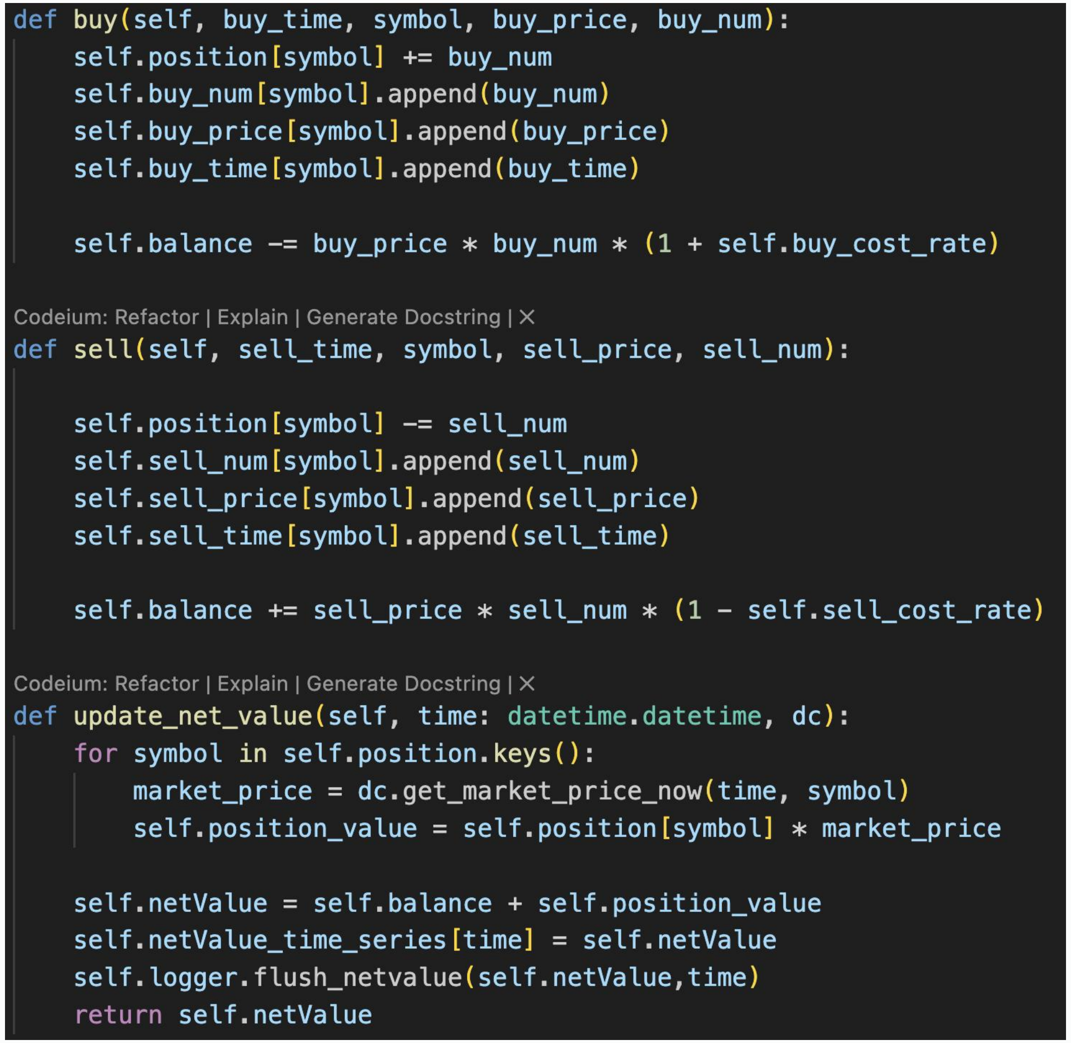
Generate signal according to the some logic and send the signal to OrderExecution

Logger&UI

Flush netvalue and execution information to log file.
Read log file and present in the webpage.

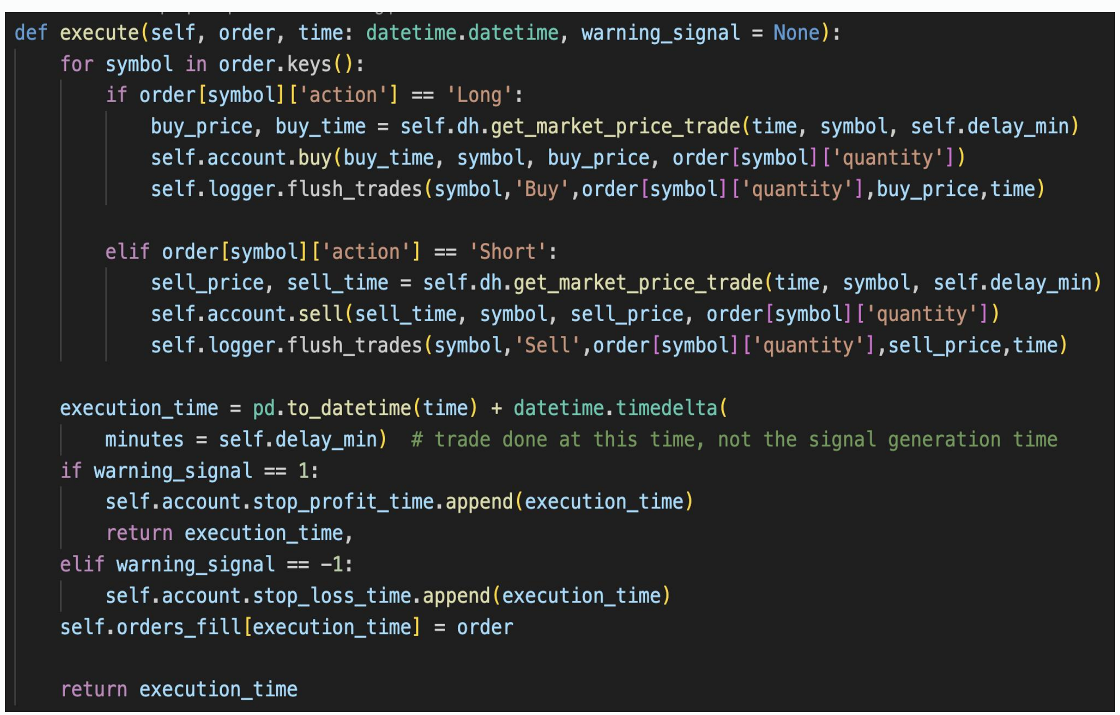
Backtest

Account



```
def buy(self, buy_time, symbol, buy_price, buy_num):
    self.position[symbol] += buy_num
    self.buy_num[symbol].append(buy_num)
    self.buy_price[symbol].append(buy_price)
    self.buy_time[symbol].append(buy_time)

    self.balance -= buy_price * buy_num * (1 + self.buy_cost_rate)

Codeium: Refactor | Explain | Generate Docstring | X
def sell(self, sell_time, symbol, sell_price, sell_num):

    self.position[symbol] -= sell_num
    self.sell_num[symbol].append(sell_num)
    self.sell_price[symbol].append(sell_price)
    self.sell_time[symbol].append(sell_time)

    self.balance += sell_price * sell_num * (1 - self.sell_cost_rate)

Codeium: Refactor | Explain | Generate Docstring | X
def update_net_value(self, time: datetime.datetime, dc):
    for symbol in self.position.keys():
        market_price = dc.get_market_price_now(time, symbol)
        self.position_value = self.position[symbol] * market_price

    self.netValue = self.balance + self.position_value
    self.netValue_time_series[time] = self.netValue
    self.logger.flush_netvalue(self.netValue,time)
    return self.netValue
```

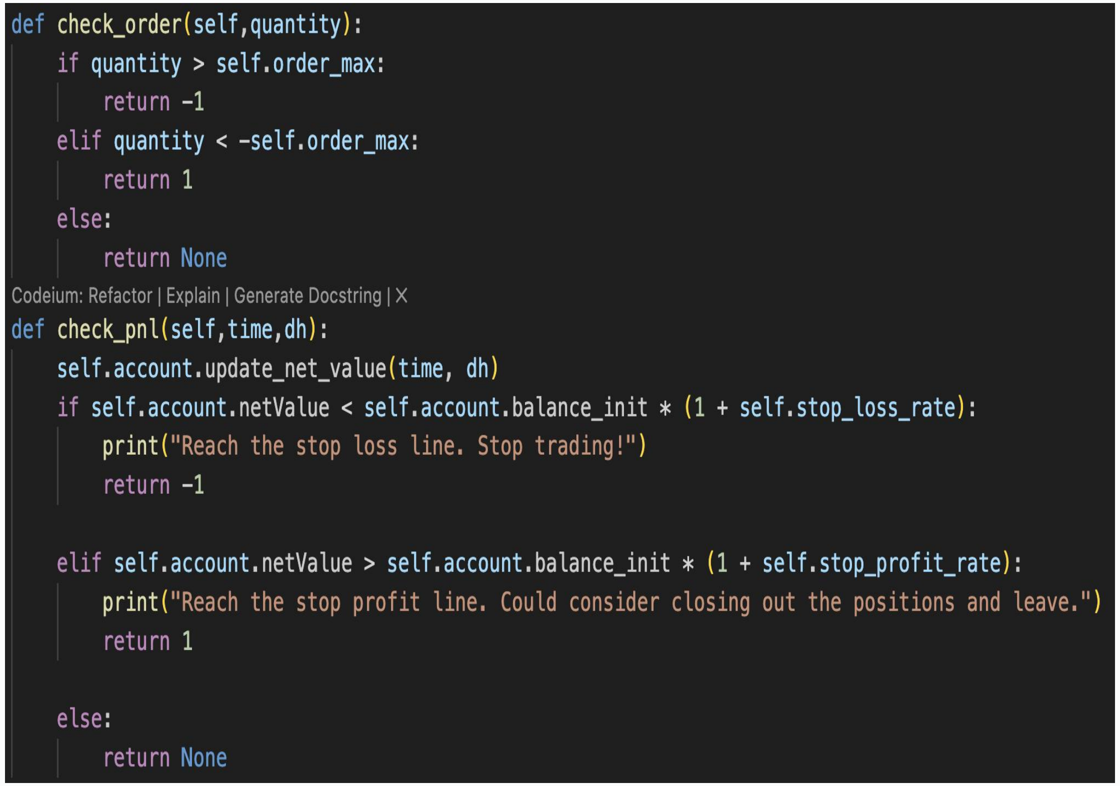
OrderExecution



```
def execute(self, order, time: datetime.datetime, warning_signal = None):
    for symbol in order.keys():
        if order[symbol]['action'] == 'Long':
            buy_price, buy_time = self.dh.get_market_price_trade(time, symbol, self.delay_min)
            self.account.buy(buy_time, symbol, buy_price, order[symbol]['quantity'])
            self.logger.flush_trades(symbol,'Buy',order[symbol]['quantity'],buy_price,time)

        elif order[symbol]['action'] == 'Short':
            sell_price, sell_time = self.dh.get_market_price_trade(time, symbol, self.delay_min)
            self.account.sell(sell_time, symbol, sell_price, order[symbol]['quantity'])
            self.logger.flush_trades(symbol,'Sell',order[symbol]['quantity'],sell_price,time)

    execution_time = pd.to_datetime(time) + datetime.timedelta(
        minutes = self.delay_min)  # trade done at this time, not the signal generation time
    if warning_signal == 1:
        self.account.stop_profit_time.append(execution_time)
        return execution_time,
    elif warning_signal == -1:
        self.account.stop_loss_time.append(execution_time)
    self.orders_fill[execution_time] = order

    return execution_time
```

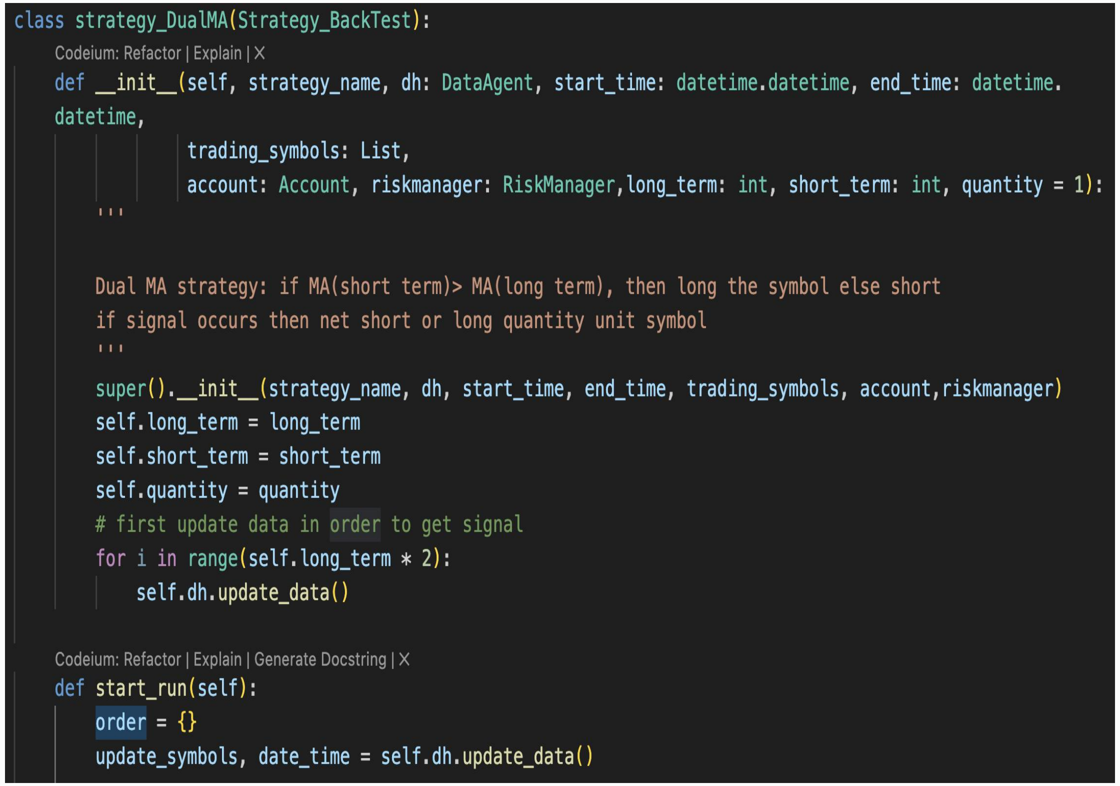
Backtest

RiskManager



```
def check_order(self,quantity):
    if quantity > self.order_max:
        return -1
    elif quantity < -self.order_max:
        return 1
    else:
        return None
Codeium: Refactor | Explain | Generate Docstring | X
def check_pnl(self,time,dh):
    self.account.update_net_value(time, dh)
    if self.account.netValue < self.account.balance_init * (1 + self.stop_loss_rate):
        print("Reach the stop loss line. Stop trading!")
        return -1

    elif self.account.netValue > self.account.balance_init * (1 + self.stop_profit_rate):
        print("Reach the stop profit line. Could consider closing out the positions and leave.")
        return 1

    else:
        return None
```

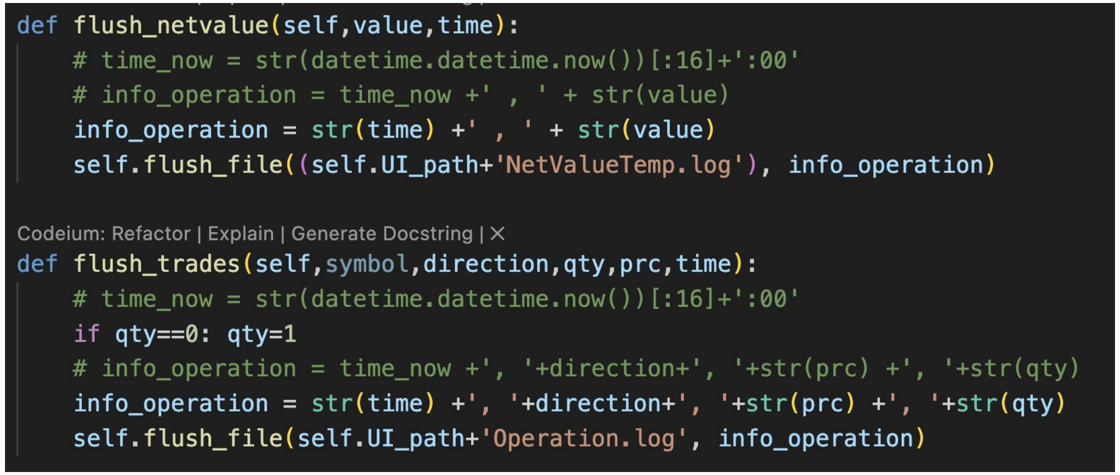
Strategy



```
class strategy_DualMA(Strategy_BackTest):
    Codeium: Refactor | Explain | X
    def __init__(self, strategy_name, dh: DataAgent, start_time: datetime.datetime, end_time: datetime.
    datetime,
                trading_symbols: List,
                account: Account, riskmanager: RiskManager,long_term: int, short_term: int, quantity = 1):
        '''

        Dual MA strategy: if MA(short term)> MA(long term), then long the symbol else short
        if signal occurs then net short or long quantity unit symbol
        '''
        super().__init__(strategy_name, dh, start_time, end_time, trading_symbols, account,riskmanager)
        self.long_term = long_term
        self.short_term = short_term
        self.quantity = quantity
        # first update data in order to get signal
        for i in range(self.long_term * 2):
            self.dh.update_data()

    Codeium: Refactor | Explain | Generate Docstring | X
    def start_run(self):
        order = {}
        update_symbols, date_time = self.dh.update_data()
```

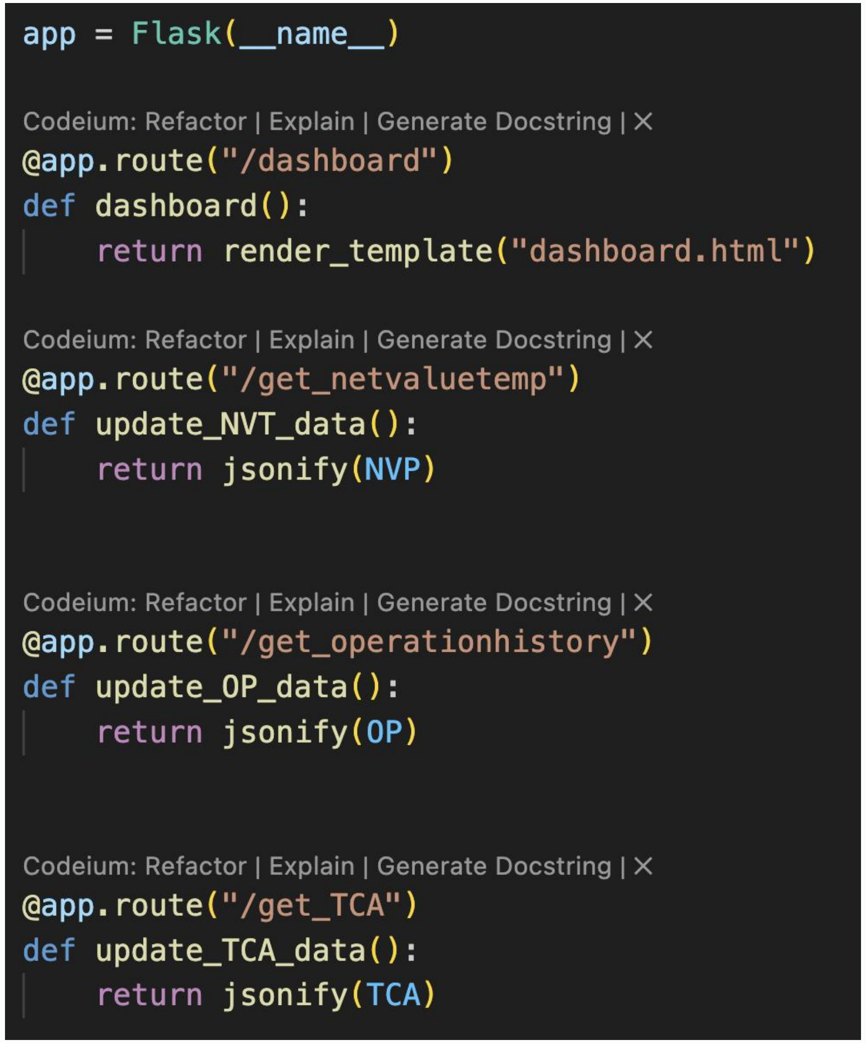
Backtest

Logger



```
def flush_netvalue(self,value,time):
    # time_now = str(datetime.datetime.now())[:16]+':00'
    # info_operation = time_now +' , ' + str(value)
    info_operation = str(time) +' , ' + str(value)
    self.flush_file((self.UI_path+'NetValueTemp.log'), info_operation)

Codeium: Refactor | Explain | Generate Docstring | ×
def flush_trades(self,symbol,direction,qty,prc,time):
    # time_now = str(datetime.datetime.now())[:16]+':00'
    if qty==0: qty=1
    # info_operation = time_now +', '+direction+', '+str(prc) +', '+str(qty)
    info_operation = str(time) +', '+direction+', '+str(prc) +', '+str(qty)
    self.flush_file(self.UI_path+'Operation.log', info_operation)
```

UI



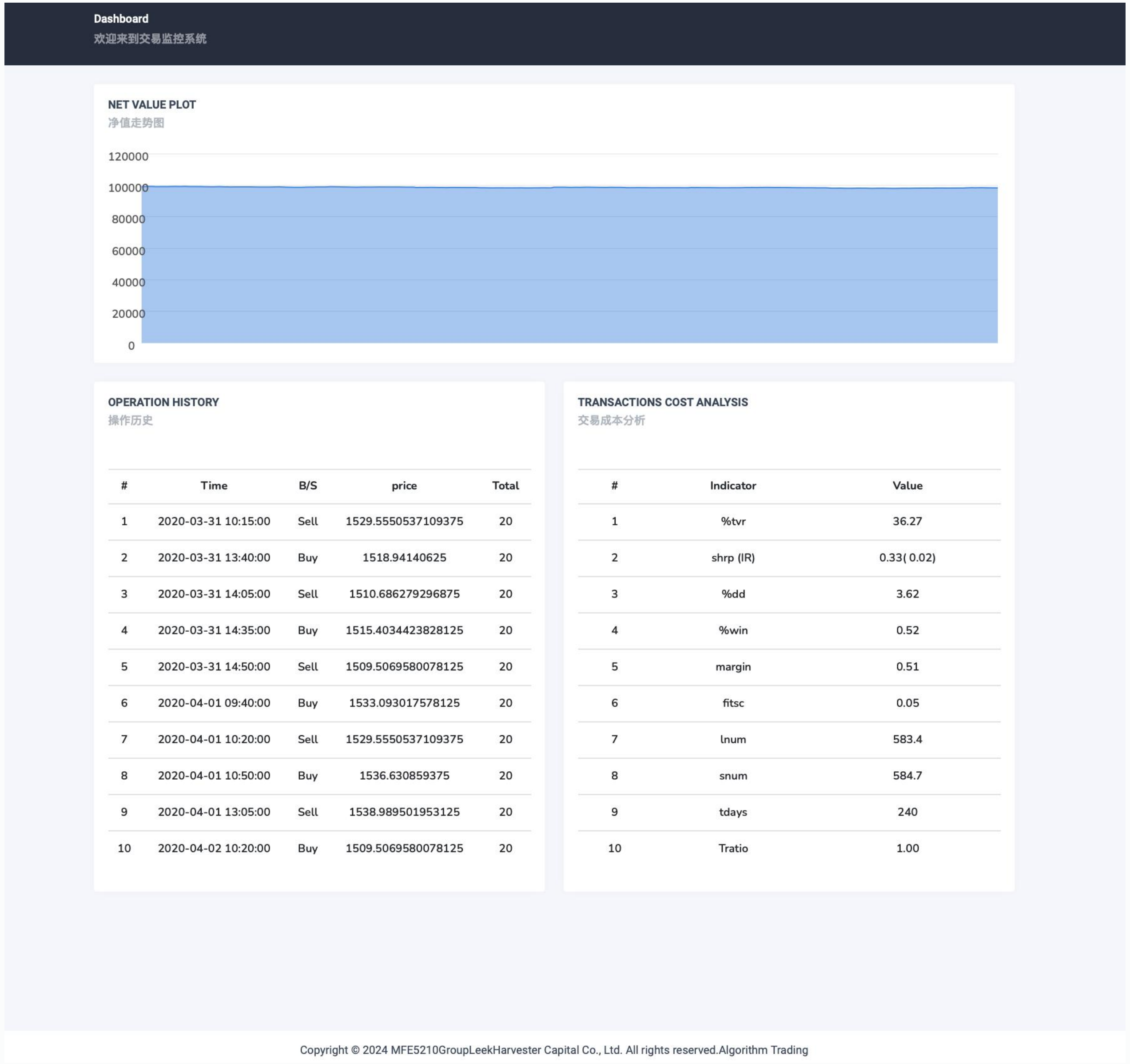
```
app = Flask(__name__)

Codeium: Refactor | Explain | Generate Docstring | ×
@app.route("/dashboard")
def dashboard():
    return render_template("dashboard.html")

Codeium: Refactor | Explain | Generate Docstring | ×
@app.route("/get_netvaluetemp")
def update_NVT_data():
    return jsonify(NVP)


Codeium: Refactor | Explain | Generate Docstring | ×
@app.route("/get_operationhistory")
def update_OP_data():
    return jsonify(OP)


Codeium: Refactor | Explain | Generate Docstring | ×
@app.route("/get_TCA")
def update_TCA_data():
    return jsonify(TCA)
```

Backtest

Paper Trading Demo

Parameters:

- Symble: SZ.000001
- Frequency: 5 min
- Period: 2020-01-02 - 2020-04-02 10:30:00
- Strategy: Dual Moving Average(10,5)



Dashboard

欢迎来到交易监控系统

NET VALUE PLOT

净值走势图

OPERATION HISTORY

操作历史

#	Time	B/S	price	Total
1	2020-03-31 10:15:00	Sell	1529.5550537109375	20
2	2020-03-31 13:40:00	Buy	1518.94140625	20
3	2020-03-31 14:05:00	Sell	1510.686279296875	20
4	2020-03-31 14:35:00	Buy	1515.4034423828125	20
5	2020-03-31 14:50:00	Sell	1509.5069580078125	20
6	2020-04-01 09:40:00	Buy	1533.093017578125	20
7	2020-04-01 10:20:00	Sell	1529.5550537109375	20
8	2020-04-01 10:50:00	Buy	1536.630859375	20
9	2020-04-01 13:05:00	Sell	1538.989501953125	20
10	2020-04-02 10:20:00	Buy	1509.5069580078125	20

TRANSACTIONS COST ANALYSIS

交易成本分析

#	Indicator	Value
1	%tvr	36.27
2	shrp (IR)	0.33(0.02)
3	%dd	3.62
4	%win	0.52
5	margin	0.51
6	fitsc	0.05
7	lnum	583.4
8	snum	584.7
9	tdays	240
10	Tratio	1.00

Live Trading

TradingBot

01

Connet to live data source and fetch data;
Manage strategy and account

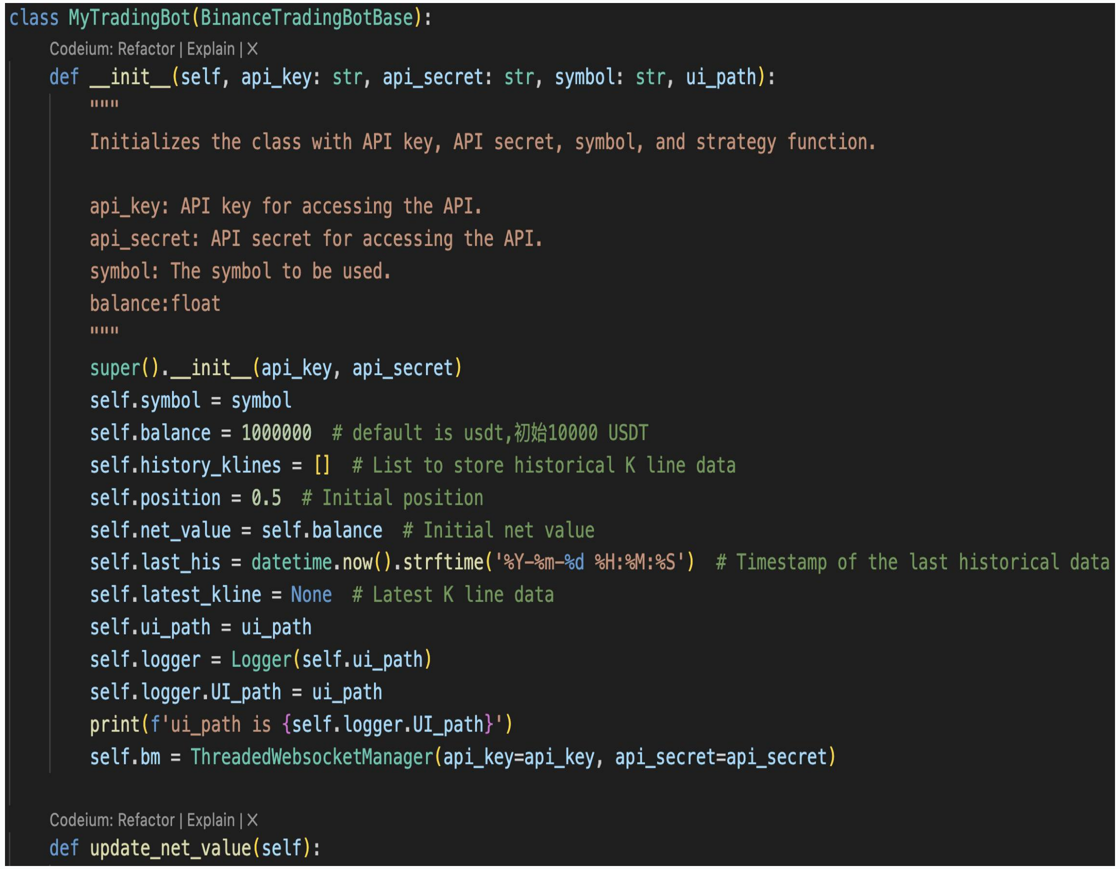
TCA

02

Calculate the transaction cost metrics, such as average execution price,inplementation shortfall,RPM

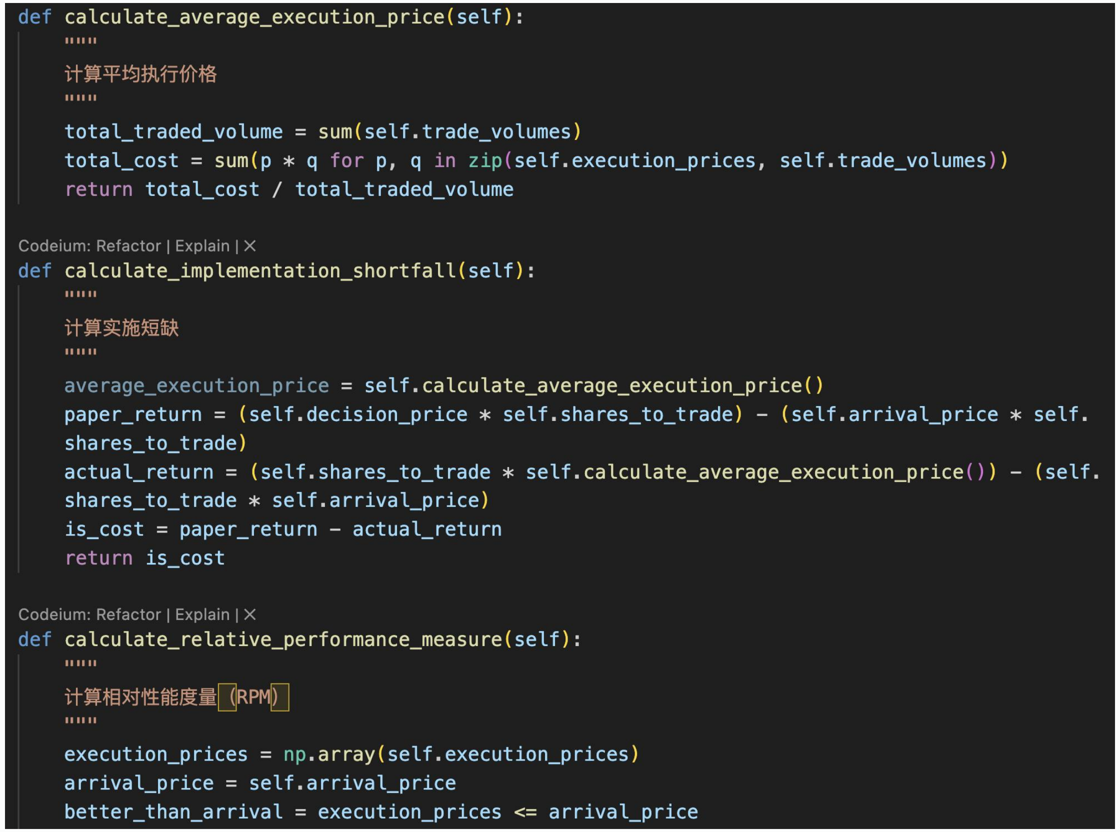
Live Trading

TradingBot



```
class MyTradingBot(BinanceTradingBotBase):
    Codeium: Refactor | Explain | X
    def __init__(self, api_key: str, api_secret: str, symbol: str, ui_path):
        """
        Initializes the class with API key, API secret, symbol, and strategy function.

        api_key: API key for accessing the API.
        api_secret: API secret for accessing the API.
        symbol: The symbol to be used.
        balance:float
        """
        super().__init__(api_key, api_secret)
        self.symbol = symbol
        self.balance = 1000000  # default is usdt,初始10000 USDT
        self.history_klines = []  # List to store historical K line data
        self.position = 0.5  # Initial position
        self.net_value = self.balance  # Initial net value
        self.last_his = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  # Timestamp of the last historical data
        self.latest_kline = None  # Latest K line data
        self.ui_path = ui_path
        self.logger = Logger(self.ui_path)
        self.logger.UI_path = ui_path
        print(f'ui_path is {self.logger.UI_path}')
        self.bm = ThreadedWebsocketManager(api_key=api_key, api_secret=api_secret)

    Codeium: Refactor | Explain | X
    def update_net_value(self):
```

TCA



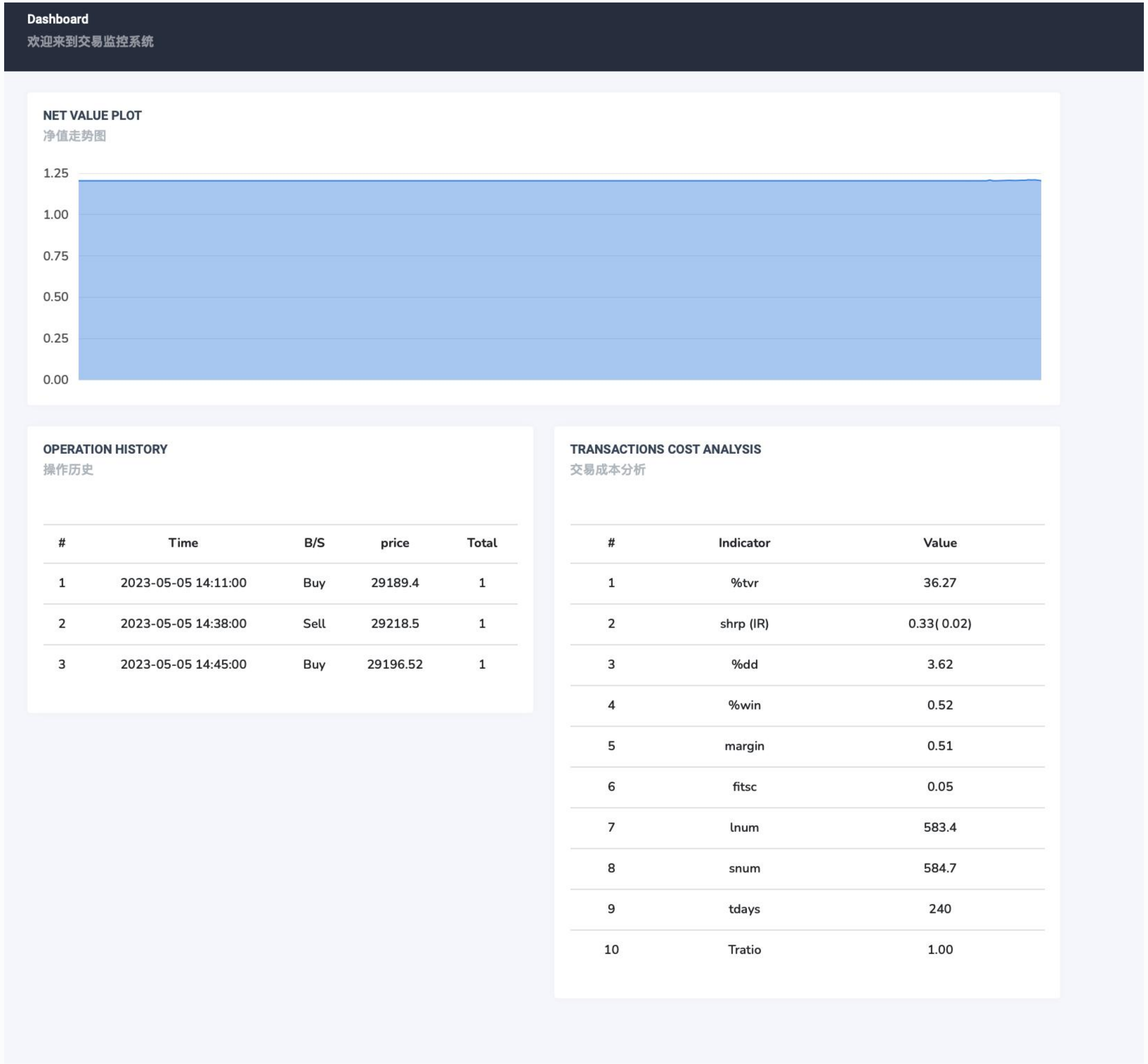
```
def calculate_average_execution_price(self):
    """
    计算平均执行价格
    """
    total_traded_volume = sum(self.trade_volumes)
    total_cost = sum(p * q for p, q in zip(self.execution_prices, self.trade_volumes))
    return total_cost / total_traded_volume

Codeium: Refactor | Explain | X
def calculate_implementation_shortfall(self):
    """
    计算实施短缺
    """
    average_execution_price = self.calculate_average_execution_price()
    paper_return = (self.decision_price * self.shares_to_trade) - (self.arrival_price * self.
    shares_to_trade)
    actual_return = (self.shares_to_trade * self.calculate_average_execution_price()) - (self.
    shares_to_trade * self.arrival_price)
    is_cost = paper_return - actual_return
    return is_cost

Codeium: Refactor | Explain | X
def calculate_relative_performance_measure(self):
    """
    计算相对性能度量（RPM）
    """
    execution_prices = np.array(self.execution_prices)
    arrival_price = self.arrival_price
    better_than_arrival = execution_prices <= arrival_price
```

Live Trading

LiveTrading Demo

Parameters:

- Symble: BTCUSDT
- Frequency: 1 min
- Period: Live
- Strategy: Dual Moving Average(10,5)



Dashboard
欢迎来到交易监控系统

NET VALUE PLOT
净值走势图

OPERATION HISTORY
操作历史

#	Time	B/S	price	Total
1	2023-05-05 14:11:00	Buy	29189.4	1
2	2023-05-05 14:38:00	Sell	29218.5	1
3	2023-05-05 14:45:00	Buy	29196.52	1

TRANSACTIONS COST ANALYSIS
交易成本分析

#	Indicator	Value
1	%tvr	36.27
2	shrp (IR)	0.33(0.02)
3	%dd	3.62
4	%win	0.52
5	margin	0.51
6	fitsc	0.05
7	lnum	583.4
8	snum	584.7
9	tdays	240
10	Tratio	1.00

Future Improvements

- Better pattern design.
- Better strategy.
- Multiple accounts management.
- ...

Thanks!