# main\_DecisionMaker.py:

|  |  |
| --- | --- |
| **Step** | **Description** |
| **1. Import Modules** | Imports various trading strategy classes, utility libraries like datetime, pandas, and a custom actionWriter. |
| **2. Define DecisionMaker Class** | Defines the DecisionMaker class to implement the logic for trading decisions, including attributes for stop loss, take profit, and tracking signals. |
| **3. Initialize Attributes** | Initializes attributes in the \_\_init\_\_ method to store previous signal, traded price, stop loss, and take profit values. |
| **4. Define predict Method** | Implements the core logic for making predictions based on market data (history). |
| **5. Convert History** | Converts input history data into a pandas DataFrame for easier manipulation and analysis. |
| **6. Extract Data** | Extracts the last and previous closing prices, high, low, and date values from the historical data. |
| **7. Set TP/SL Levels** | Defines default take\_profit and stop\_loss values for trading decisions. |
| **8. Initialize Strategy** | Creates an instance of the SimpleMAExponentialMA strategy (or others, if uncommented) and runs its run\_sma\_ema() method. |
| Logic   |  |  |  | | --- | --- | --- | | Step | Condition | Action | | 1. Check Stop Loss/Take Profit | If prev\_signal == 1 (buy) and: - curr\_high\_price >= curr\_take\_profit (Take Profit) - curr\_low\_price <= curr\_stop\_loss (Stop Loss) | Reset prev\_signal = 0 | |  | If prev\_signal == -1 (sell) and: - curr\_low\_price <= curr\_take\_profit (Take Profit) - curr\_high\_price >= curr\_stop\_loss (Stop Loss) | Reset prev\_signal = 0 | | 2. Evaluate Signal | If signal == 1 (buy) |  | | 2.1 Close Sell Position | If prev\_signal == -1 (sell) | Close sell position: {"action": "POSITION\_CLOSE\_SYMBOL"} Reset prev\_signal = 0. | | 2.2 Open Buy Position | If prev\_signal == 0 (no position) | Open buy position: {"action": "ORDER\_TYPE\_BUY"} Set stop\_loss = curr\_close\_price + stop\_loss. Set take\_profit = curr\_close\_price + take\_profit. | | 2.3 Modify Existing Buy | If prev\_signal == 1 (already in buy) | - If curr\_close\_price > prev\_close\_price and curr\_close\_price > prev\_traded\_price: {"action": "POSITION\_MODIFY"} with updated SL/TP. - Otherwise: {"action": "skip"}. | | 3. Evaluate Signal | If signal == -1 (sell) |  | | 3.1 Close Buy Position | If prev\_signal == 1 (buy) | Close buy position: {"action": "POSITION\_CLOSE\_SYMBOL"} Reset prev\_signal = 0. | | 3.2 Open Sell Position | If prev\_signal == 0 (no position) | Open sell position: {"action": "ORDER\_TYPE\_SELL"} Set stop\_loss = curr\_close\_price - stop\_loss. Set take\_profit = curr\_close\_price - take\_profit. | | 3.3 Modify Existing Sell | If prev\_signal == -1 (already in sell) | - If curr\_close\_price < prev\_close\_price and curr\_close\_price < prev\_traded\_price: {"action": "POSITION\_MODIFY"} with updated SL/TP. - Otherwise: {"action": "skip"}. | | 4. Hold (No Action) | If signal == 0 (hold) | Do nothing: {"action": "skip"}. | | |
| **14. Return Decision** | Returns the chosen action (buy, sell, modify, skip) along with the signal, updated position status, and data. |
| **15. Execute Actions** | Creates an actionWriter instance and runs the run() method to execute decisions in the trading environment. |

|  |  |  |
| --- | --- | --- |
| **Step** | **Condition** | **Action** |
| **1. Check Stop Loss/Take Profit** | If prev\_signal == 1 (buy) and: - curr\_high\_price >= curr\_take\_profit (Take Profit) - curr\_low\_price <= curr\_stop\_loss (Stop Loss) | Reset prev\_signal = 0 |
|  | If prev\_signal == -1 (sell) and: - curr\_low\_price <= curr\_take\_profit (Take Profit) - curr\_high\_price >= curr\_stop\_loss (Stop Loss) | Reset prev\_signal = 0 |
| **2. Evaluate Signal** | If signal == 1 (buy) |  |
| **2.1 Close Sell Position** | If prev\_signal == -1 (sell) | Close sell position: {"action": "POSITION\_CLOSE\_SYMBOL"} Reset prev\_signal = 0. |
| **2.2 Open Buy Position** | If prev\_signal == 0 (no position) | Open buy position: {"action": "ORDER\_TYPE\_BUY"} Set stop\_loss = curr\_close\_price + stop\_loss. Set take\_profit = curr\_close\_price + take\_profit. |
| **2.3 Modify Existing Buy** | If prev\_signal == 1 (already in buy) | - If curr\_close\_price > prev\_close\_price and curr\_close\_price > prev\_traded\_price: {"action": "POSITION\_MODIFY"} with updated SL/TP. - Otherwise: {"action": "skip"}. |
| **3. Evaluate Signal** | If signal == -1 (sell) |  |
| **3.1 Close Buy Position** | If prev\_signal == 1 (buy) | Close buy position: {"action": "POSITION\_CLOSE\_SYMBOL"} Reset prev\_signal = 0. |
| **3.2 Open Sell Position** | If prev\_signal == 0 (no position) | Open sell position: {"action": "ORDER\_TYPE\_SELL"} Set stop\_loss = curr\_close\_price - stop\_loss. Set take\_profit = curr\_close\_price - take\_profit. |
| **3.3 Modify Existing Sell** | If prev\_signal == -1 (already in sell) | - If curr\_close\_price < prev\_close\_price and curr\_close\_price < prev\_traded\_price: {"action": "POSITION\_MODIFY"} with updated SL/TP. - Otherwise: {"action": "skip"}. |
| **4. Hold (No Action)** | If signal == 0 (hold) | Do nothing: {"action": "skip"}. |

# actionWriter.py

1. where it is used: inside decision maker.py >>from actionWriter import actionWriter
2. save’data ‘ to ‘ time\_close\_csv\_test.csv ‘
3. this is a bridge between mt5 and python.
4. این اطلاعات مرتب میشه به صورت زیر و به استراتژی ها ارسال میشن از طریق

# code from example2.py, send the data to the main\_DecisionMaker.py

predict\_result, signal, prev\_signal, df = self.trading\_algrithm.predict(contents)

which date:

* contents[i][0] = datetime.strptime(contents[i][0], '%Y.%m.%d %H:%M:%S')
* contents[i][1] = float(contents[i][1]) #open
* contents[i][2] = float(contents[i][2]) #high
* contents[i][3] = float(contents[i][3]) #low
* contents[i][4] = float(contents[i][4]) #close
* contents[i][5] = int(contents[i][5]) #tick value
* ·contents[i][6] = int(contents[i][6]) #running or ending.(defined by py\_json\_csv\_0204.mq5)

A screenshot of a spreadsheet

Description automatically generated

اخر کد دسیشن میکر این خط رو داریم.

DecisionMaker()

executor = actionWriter(DecisionMaker())

طبق این عکسه ما به عنوان ورودی decision maker رو دادیم به اکشن رایتر.

class actionWriter():

def \_\_init\_\_(self, trading\_algrithm):

self.trading\_algrithm = trading\_algrithm

ما یک سازنده داریم که تردینگ استراتژی رو میسازه به عنوان یک اتریبیوت یا ویژگی

که DecisionMaker()) میاد به عنوان ورودی اینجا میشینه.

A screen shot of a computer program

Description automatically generated A screenshot of a black screen

Description automatically generated

class DecisionMaker:

def \_\_init\_\_(self):

def predict(self, history):

اینشکی هم دسیشن میکر این اتربیوت رو داره و ما ازش استفاده کردیم جلو تر.

predict\_result, signal, prev\_signal, df = self.trading\_algrithm.predict(contents)

|  |  |
| --- | --- |
| Step | Description |
| 1. Import Libraries | Imports required libraries like datetime, time, os, copy, pandas, numpy, talib, and json. |
| 2. Define actionWriter Class | Implements a class to handle writing trading decisions to files and interacting with the trading algorithm. |
| 3. Initialize Class (\_\_init\_\_) | Accepts a trading\_algorithm instance (e.g., DecisionMaker) and stores it as an attribute. |
| 4. Write Strategies | Defines the write\_strategies method to save predictions as JSON in **action\_test.txt**. |
| 5. Save to CSV | Implements save2csv to save predictions and other details into a CSV file using an external output module. |
| 6. Clean File | Provides the cleanFile method to overwrite the contents of a file, effectively clearing it. |
| 7. Define run Method | Core logic for running the action writer continuously: |
| 8. Check File Existence | Verifies if the time\_close\_csv\_test.csv file exists and is not empty. If not, prints a message and exits. |
| 9. Read File Contents | Opens and reads the CSV file with utf-16 encoding, splitting lines and converting to the appropriate data types. |
| 10. Parse Data | Processes each row of the file to extract datetime, open, high, low, close, and tick volume. |
| 11. Detect New Timebar | Checks if the latest time bar (newTimebar) is different from the previous time bar (pre\_Timebar). |
| 12. Process Signals | If a new time bar is detected: |
|  | - Calls the predict method of the trading algorithm. |
|  | - Ensures the result is a dictionary, then prints and saves it. |
|  | - Writes the result to a JSON file (action\_test.txt) and saves details to a CSV.  # write the result to txt or csv  self.write\_strategies(predict\_result) |
|  | - Outputs intermediate results every 50 steps. |
| 13. Handle No Change | If the time bar hasn’t changed, waits briefly (time.sleep(0.003) to minimize resource usage. |
| 14. Stop Condition | Checks if the current position is "Ending". If so: |
|  | - Calls the output\_csv method from output\_save. |
|  | - Prints a message indicating the server has stopped and exits the loop. |
| 15. Handle Errors | Catches exceptions during file reading or processing, skips the iteration, and waits briefly. |
| 16. Loop Control | Continuously checks the file and processes new data until the program is stopped or encounters an "Ending" position. |

filename = "time\_close\_csv\_test.csv"

self.save2csv(output\_save, predict\_result, contents, signal, prev\_signal, df)

check\_point += 1

if check\_point % 50 == 0:

output\_save.output\_csv()  
  
هر50تا یبار یه خروجی میده و در

output.csv  
همچین چیزی رو ذخیره میکنه

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| date | close\_price | signal | prev\_signal | action |  |
| 01/10/2024 05:30 | 2637.91 | -1 | -1 | ORDER\_TYPE\_SELL | |
| 01/10/2024 05:45 | 2636.37 | -1 | -1 | ORDER\_TYPE\_SELL | |
| 01/10/2024 06:00 | 2635.43 | -1 | -1 | ORDER\_TYPE\_SELL | |
| 01/10/2024 06:15 | 2636.08 | -1 | -1 | ORDER\_TYPE\_SELL | |
| 01/10/2024 06:45 | 2637.95 | 0 | 0 | skip |  |
| 01/10/2024 07:00 | 2638.39 | 0 | 0 | skip |  |
| 01/10/2024 07:15 | 2638.39 | 0 | 0 | skip |  |
| 01/10/2024 07:30 | 2639.99 | 0 | 0 | skip |  |
| 01/10/2024 07:45 | 2639.61 | 0 | 0 | skip |  |
| 01/10/2024 08:00 | 2642.43 | 0 | 0 | skip |  |
| 01/10/2024 08:15 | 2642.81 | 0 | 0 | skip |  |
| 01/10/2024 08:30 | 2643.82 | 0 | 0 | skip |  |
| 01/10/2024 08:45 | 2643.06 | 0 | 0 | skip |  |
| 01/10/2024 09:00 | 2643.72 | 0 | 0 | skip |  |
| 01/10/2024 09:15 | 2644.59 | 0 | 0 | skip |  |
| 01/10/2024 09:30 | 2644.4 | 0 | 0 | skip |  |
| 01/10/2024 09:45 | 2645.29 | 0 | 0 | skip |  |
| 01/10/2024 10:00 | 2643.17 | -1 | -1 | ORDER\_TYPE\_SELL | |
| 01/10/2024 10:15 | 2642.63 | -1 | -1 | ORDER\_TYPE\_SELL | |
| 01/10/2024 10:30 | 2644.02 | 0 | 0 | skip |  |
| 01/10/2024 10:45 | 2645.58 | 0 | 0 | skip |  |
| 01/10/2024 11:00 | 2644.35 | 0 | 0 | skip |  |

if check\_point % 50 == 0:

output\_save.output\_csv()

print("output is saved","\t",check\_point)

# output.py

* input: date','close\_price', 'signal','prev\_signal','action’
* output: output.csv in form of panda data frame work
* where it is used : inside the actionwriter: from output import output

|  |  |
| --- | --- |
| Step | Description |
| 1. Define output Class | Implements a class to store and manage data for trading actions and save the results to a CSV file. |
| 2.Initialize Attributes | The \_\_init\_\_ method initializes lists to store date, close price, signal, previous signal, and action data. |
| 3.save\_csv Method | Saves trading-related data for each new signal into the initialized lists: |
|  | - Parameters: contents (raw CSV data), dframe (dataframe of historical data), signal, prev\_signal, predict\_result. |
|  | - Extracts the most recent date and close price from contents and dframe. |
|  | - Appends the extracted data and predictions to the respective lists. |
| 4.output\_csv Method | Outputs the stored data to a CSV file (output.csv): |
|  | - Combines all lists into a single list (output\_lst). |
|  | - Creates a pandas DataFrame (df\_output) from the combined list and transposes it to align columns correctly. |
|  | - Assigns column names: date, close\_price, signal, prev\_signal, action. |
|  | - Writes the DataFrame to a CSV file (output.csv) without the index. |

py\_json\_csv\_0204.mq5 |

این فایل میاد میشینه میخونه چیزایی که پایتون فرستاده رو بعد براساس اونها سفارش میفرسته

void readFile()

{

// open txt file

int fileHandleTxt = FileOpen("action\_test.txt",FILE\_READ|FILE\_TXT|FILE\_ANSI);

if(fileHandleTxt==INVALID\_HANDLE) Alert("could not open txt file, error: "+(string)GetLastError());

if(fileHandleTxt!=INVALID\_HANDLE)

{

string s;

while(!FileIsEnding(fileHandleTxt)) StringAdd(s,FileReadString(fileHandleTxt));

Print(s);

FileClose(fileHandleTxt);

srce.Deserialize(s);

string actionType = srce["action"].ToStr();

double TP = srce["takeprofit"].ToDbl();

double SL = srce["stoploss"].ToDbl();

Print("action: ",actionType);

Print("takeprofit: ",TP);

Print("stoploss: ",SL);

int idNimber = NULL;

double price = NULL;

double volume = 1;

string comment = NULL;

//action\_int[0] = (int) FileReadNumber(fileHandleTxt);

//int pre\_signal = action\_int[0];

//Print("action Received: ",pre\_signal);

// Market orders

if(actionType=="ORDER\_TYPE\_BUY" || actionType=="ORDER\_TYPE\_SELL"){

ENUM\_ORDER\_TYPE orderType=ORDER\_TYPE\_BUY;

price = SymbolInfoDouble(my\_symbol,SYMBOL\_ASK);

if(actionType=="ORDER\_TYPE\_SELL") {

orderType=ORDER\_TYPE\_SELL;

price=SymbolInfoDouble(my\_symbol,SYMBOL\_BID);

}

if(m\_Trade.PositionOpen(my\_symbol,orderType,volume,price,SL,TP,comment)){

OrderDoneOrError(false, \_\_FUNCTION\_\_, m\_Trade);

FileDelete("action\_test.txt");

Print("Delete file");

return;

}

}

// Position modify

else if(actionType=="POSITION\_MODIFY"){

if(m\_Trade.PositionModify(idNimber,SL,TP)){

OrderDoneOrError(false, \_\_FUNCTION\_\_, m\_Trade);

FileDelete("action\_test.txt");

Print("Delete file");

return;

}

}

// Position close by symbol

else if(actionType=="POSITION\_CLOSE\_SYMBOL"){

if(m\_Trade.PositionClose(my\_symbol)){

OrderDoneOrError(false, \_\_FUNCTION\_\_, m\_Trade);

FileDelete("action\_test.txt");

Print("Delete file");

return;

}

}

// old one

/\*

if(pre\_signal == 1){

double ask = SymbolInfoDouble(my\_symbol,SYMBOL\_ASK);

double sl = ask - 250 \* SymbolInfoDouble(my\_symbol, SYMBOL\_POINT);

double tp = ask + 200 \* SymbolInfoDouble(my\_symbol, SYMBOL\_POINT);

//Print("ask", ask);

//Print("sl",sl);

if(m\_Position.Select(my\_symbol))

{

if(m\_Position.PositionType()==POSITION\_TYPE\_SELL) m\_Trade.PositionClose(my\_symbol); //and this is a Sell position, then close it

if(m\_Position.PositionType()==POSITION\_TYPE\_BUY){

FileClose(fileHandleTxt);

FileDelete("action\_test.txt");

Print("Delete file");

return;

}

}

m\_Trade.Buy(1,my\_symbol,ask,sl,tp,"This is a buy");

Print("Buy");

}

else if(pre\_signal == -1){

double bid = SymbolInfoDouble(my\_symbol,SYMBOL\_BID);

double sl = bid + 250 \* SymbolInfoDouble(my\_symbol, SYMBOL\_POINT);

double tp = bid - 200 \* SymbolInfoDouble(my\_symbol, SYMBOL\_POINT);

if(m\_Position.Select(my\_symbol)) //if the position for this symbol already exists

{

if(m\_Position.PositionType()==POSITION\_TYPE\_BUY) m\_Trade.PositionClose(my\_symbol); //and this is a Buy position, then close it

if(m\_Position.PositionType()==POSITION\_TYPE\_SELL) {

FileClose(fileHandleTxt);

FileDelete("action\_test.txt");

Print("Delete file");

return;

} //or else, if this is a Sell position, then exit

}

m\_Trade.Sell(1,my\_symbol,bid,sl,tp,"This is a sell");

Print("Sell");

}

\*/

else Print("Skip");

//FileClose(fileHandleTxt);

FileDelete("action\_test.txt");

Print("Delete file");

}

}

# Address:

از فایل سیمولاتور اپن فایل رو میزنی باز میکنه پوشه رو

C:\Users\Nima\AppData\Roaming\MetaQuotes\Tester\D0E8209F77C8CF37AD8BF550E51FF075\Agent-127.0.0.1-3000\MQL5\Files

# Bug:

POSITION\_MODIFY:

مثلکه به جای اینکه همون قبلی رو مدیفای کنه میاد یدونه جدید ایجاد میکنه که تر میزنه

# محدود کردن به معامله در زمان و ساعت خاص

Actionwriter:  
                        hour = newTimebar.hour

                        minute = newTimebar.minute

pre\_Timebar != newTimebar     and (hour >15 or (hour == 15 and minute >0)):

# تعریف استاپلاس وتیک پرافیت

decision maker,

  # adjust TP/SL values here, remember to x100 if testing on JPY currency

        #take\_profit = 0.0200

        take\_profit = 9.00

        #stop\_loss = -0.0250

        stop\_loss = -3.00

سیگنال خرید و فروش به صورت تکی انجام میشه

ولی مدیریت اون سیگنالی که بازه مشخص نیست توسط پایتوون بشه انجام داد یا نه

همچنین بحث پلاگین متاتردیر هم مطرح است

# تشخیص پرایس اکشنی

اومده یه کلس تشکیل داده به اسم متاتردیر 5 بعد اردرهارو توش تعریف کرده. اون یکی که ویدیوهاشو دانلود کردم دوره یودمی داشت.  
Chapter\_08\_MT5

|  |
| --- |
| رو 15 دقیقه تشخیص میده اسپایک رو  وقتی لو کندل بعدی از لو کندل قبل پایین تر باشه ترندلاین برک شده  بعد اگه هایش بیشتر باشه از های کندل قبلی سیگنال ورود داده  صبر میکنه  وقتی قیمت رسید به محدوده 0.5ایکس و استاپ اولیه شروع میکنه به خرید و فروش  هدف باید اول تولید سیگنال و گرفتن بک تست باشه |

|  |
| --- |
| history\_dataframe = pd.DataFrame(history, columns=("time", "open", "high", "low", "close", "tick\_volume","pos"))  decision making  strategis |

# py\_json\_csv\_0204.mq5 runing and ending is defined here in this file for current position from line 970 is important the rest is about discriting json.

سه تا تابع داره

## void WriteFile(string position)

خودم تعریف کردم که بهم خروجی بده

الان یه خروجی جدید تعریف کردم به اسم full account detail

که خروجیش به شکل زیر است

|  |
| --- |
| Category Key Value  Account Balance 10416.81  Account Equity 10416.81  Account Free Margin 10416.81  Account Margin Level 0.0  Account Leverage 100  Account Currency USD  Position No open positions  History Ticket: 4, Symbol: XAUUSD, Volume: 1.00, Price: 2658.02000, Profit: -194.00, Time: 2024.10.02 03:35  History Ticket: 5, Symbol: XAUUSD, Volume: 1.00, Price: 2657.95000, Profit: -207.00, Time: 2024.10.02 03:40  History Ticket: 7, Symbol: XAUUSD, Volume: 1.00, Price: 2661.94000, Profit: 407.00, Time: 2024.10.02 04:54  History Ticket: 9, Symbol: XAUUSD, Volume: 1.00, Price: 2658.29000, Profit: -251.00, Time: 2024.10.02 06:28 |

و یک کد نوشتم رو پایتون که این رو دیکد میکنه و میریزه داخل دیکشنری

|  |
| --- |
| import pandas as pd    file\_path = "full\_account\_details.csv"  try:  # باز کردن فایل با انکودینگ utf-16  with open(file\_path, encoding='utf-16') as f:  contents = f.read()  print("File read successfully!")  print(contents) # برای نمایش محتوا در کنسول  except UnicodeDecodeError as e:  print("Failed to decode file with utf-16 encoding:")  print(e)  # ادامه پردازش محتوا  else:  print("Processing complete!")  # دیکشنری برای ذخیره داده‌ها  data\_dict = {  "Account": [],  "Position": [],  "History": []  }  # پردازش هر خط  lines = contents.split("\n")  for line in lines:  line = line.strip()  if line.startswith("Account"):  \_, key, value = line.split("\t")  data\_dict["Account"].append({"Key": key, "Value": value})  elif line.startswith("Position"):  \_, position\_data = line.split("\t", 1)  data\_dict["Position"].append(position\_data)  elif line.startswith("History"):  \_, history\_data = line.split("\t", 1)  details\_parts = history\_data.split(", ")  history\_entry = {}  for part in details\_parts:  key, value = part.split(": ")  history\_entry[key.strip()] = value.strip()  data\_dict["History"].append(history\_entry)  # نمایش داده‌ها  print("Account Data:")  print(data\_dict["Account"])  print("\nPosition Data:")  print(data\_dict["Position"])  print("\nHistory Data:")  print(data\_dict["History"]) |

## on denit

وقتی که کد تموم شد

میاد همه توابع رو میبنده و به خروجی time\_close\_csv\_test.csv دستور اندینگ رو میفرسته

## on init

## on tick

# تو هر استراتژی چخبره

class SimpleMAExponentialMA:

def \_\_init\_\_(self, file\_path): کانستراکتر دیتابیس داره هر استراتژی

#self.df = pd.read\_csv(file\_path)

self.df = pd.DataFrame(file\_path, columns=("time", "open", "high", "low", "close", "tick\_volume","pos"))

self.close = self.df['close'] # retrieves the most recent closing price

یه اتریبیوت ران هم داره که میاد

def run\_sma\_ema(self):

self.calculate\_144ema()

self.calculate\_169ema()

self.calculate\_5sma()

signal = self.determine\_signal(self.df)

return signal, self.df

# وقتی سیگنال مشخص میشه چه اتفاقی میفته:

0

1

-1

اون بالا نوشتم که چخبره