Automated Algorithmic Trading

Contents

1. A Brief Overview

1.1 Application in Today's Trading Systems

Currently, millions of trades are being executed on various markets, from the foreign exchange market, the stock market, the commodities market, and even the relatively young cryptocurrency market. Due to the huge profit potential, both corporate and retail traders have taken up trading as a job or even a hobby. Making trade decisions can be a stressful and time-consuming process, as traders have to constantly monitor complicated charts, solve quick calculations, and analyse the market environment, all within an extremely short period of time; depending on the chosen timeframe, many traders have to make decisions in under a minute!

The overall aim of this algorithm is to enforce a more accurate and more consistent way of trading markets. It also offers other important benefits such as preventing emotional factors from influencing trades, preventing human error (for example, mis-clicks or miscalculations), minimising time wasted before executing trades (an algorithm can execute faster than a human), and monitoring markets around the clock (algorithms do not require sleep). These factors are the main causes of missed trading opportunities, but are eliminated when using a computer program to make trading decisions.

1.2 Capabilities

This algorithm offers the following features to its users:

- Automatic market trading of a cryptocurrency of the user's choice;
- viewing the user's available trading account balance;
- viewing the program's trading history;
- configuring the trade settings to suit the user's needs.

On starting the program, the user is required to input the symbol of the cryptocurrency pair they wish the bot to trade. The user also must input their preferred time frame. The time frame determines how often trading decisions are made; for example, if the user sets the time frame to '1 minute', the bot will make judgements about whether to trade or not, every minute. The same applies to the other time frame options ('5 minutes', '1 hour' and '1 day').

These two inputs are the only user interaction required for the bot to start trading. Of course, the user can adjust the bot's trading settings in the 'Configure Trade Settings' tab on the 'Main Menu'. The user may wish to adjust these settings if they want the bot to make safe trading decisions, moderately risky decisions, or risky decisions. The program will state how risky the bot's current trading conditions are. You can see this by navigating to 'Configure Trade Settings', then 'Risk Management Settings', and lastly, 'Risk Level Settings'. The user is also able to change their risk settings by navigating to 'RSI Settings' from 'Configure Trade Settings'. The default trading settings are moderately risky; the RSI Upperband is set to 70, the RSI Lowerband is set to 30, and the RSI Period is set to 14. These figures are explained in further detail here.

Finally, if the user wishes to login to another trading account, they can select 'Logout' on the 'Main Menu' and then select 'Enter Login Details'.

1.3 Construction of the Algorithm

The programming language used to code this algorithm is Python 3. The reason for this choice was that Python offers excellent web-scraping (collecting data from web resources) capabilities using a library named 'Beautiful Soup 4'. Web-scraping was necessary in order to fetch live market data from the internet, such as the current market price of a cryptocurrency pair and previous closing prices.

An essential part of the logistics behind this algorithm is the API key offered by BitMex (the trading platform used by this algorithm). This enables the program to directly communicate with the BitMex servers, located in Europe, to send various request types such as placing an order, returning account balance, viewing trading history and so on. The two types of BitMex API are explained further here.

2. A Detailed Explanation

2.1 Establishing a Connection

This algorithm is used for automatically trading cryptocurrency markets over a platform called BitMex (or BitMex TestNet). BitMex is a cryptocurrency trading platform which provides low-commission, decentralised and flexible trading conditions for users.

In order to request data from the BitMex servers, a form of communication had to be established beforehand. BitMex offers two types of API connections; REST API and WebSocket API. WebSocket API is generally slower than REST API, however, the program takes advantage of both as shown in the code below:

```
apikey = str(input("Enter your API Key: "))
        secretapikey = str(input("Enter your Secret API Key: "))
        ws = BitMEXWebsocket(endpoint="https://testnet.bitmex.com/api/v1", symbol=pair,
api_key=apikey, api_secret=secretapikey)
        client = bitmex(test=True, api_key=apikey, api_secret=secretapikey)
```

Once these connections are established, the bot can make calls to the BitMex server, for example, it can instruct the BitMex server to place a buy order on the BitMex platform.

2.2 The RSI Indicator

The algorithm makes trade decisions based on a momentum indicator called RSI (Relative Strength Index). RSI is made up of three components; upperband, lowerband and period. It scales from 0 to 100 where 0 indicates a highly overbought market state, and 100 indicates a highly oversold market state. The upperband is a level which must be between 51 and 100 (usually 70). If the current RSI level, which is calculated by the algorithm, matches the upperbound or rises above it, a sell order is executed because an overbought market state indicates that the price will soon fall. Vice versa, if the RSI level reaches or falls below the lowerband, which must be between 0 and 49 (usually 30), a buy order will be executed as an oversold market state indicates that the price will soon rise. The program allows the user to change the bounds manually. A small difference between the bounds is risky because many buy and sell orders will be executed in a market which may not be overbought or oversold. However, the wider apart the bounds are, the safer the trading environment as it is harder for the algorithm to trigger an order since it can only execute once the market is definitely in an overbought or oversold state.

The current RSI level is calculated using the formula:
RSI = 100 - (100 / (1 + RS))
where RS = Average Upwards Gain / Average Downwards Gain

The RSI settings also affect the risk factor. The risk factor is therefore, a variable, and it determines the proportion of capital (in the BitMex account) which is to be used for each trade. The higher the risk factor, the less proportion of capital is allocated per trade, as there is a higher probability of losing the capital due to liquidation. Likewise, the lower the risk factor, a higher proportion of capital is allocated per trade due to a lower probability of the order being liquidated. The value of the risk factor also determines the average frequency of trades. A high risk factor will result in a higher frequency of trades executed, compared to a low risk factor. This is due to the more versatile trade settings which trigger order signals more easily, as fewer conditions have to be met.

2.3 Functions & Their Purposes

The program consists of 18 functions, each of which plays an exclusive role in the operation of the algorithm. The functions are stated below:

- login
- connect
- logout
- mainmenu
- getrsi
- on_press
- determine
- startbot
- printviewbalance
- returnviewbalance
- viewtradinghistory
- configuretradesettings
- rsisettings
- riskmanagementsettings
- risklevelsetting
- capitalproportionmanagementsetting
- marketprice
- placeorder

Each function and its purpose are explained below (or click on a function to navigate to its explanation).

The login function enables the user to enter their public API key and their secret API key, both of which are necessary in order to connect to the BitMex servers and therefore carry out actions such as buying and selling.

```
#login
def login():
    global ws
    global client
    global apikey
    global secretapikey
    global pair

    choice = str(input('''
Login:
1. Enter Login Details
2. Remember Previous Login Details
3. Exit

'''))
    if choice == '1':
        apikey = str(input("Enter your API Key: "))
        secretapikey = str(input("Enter your Secret API Key: "))
        ws = BitMEXWebsocket(endpoint="https://testnet.bitmex.com/api/v1", symbol=pair,
api_key=apikey, api_secret=secretapikey)
        client = bitmex(test=True, api_key=apikey, api_secret=secretapikey)
        print('Login Successful \n')

        mainmenu()

    elif choice == '2':
        apikey = '1Bk8AeOWOjO2DCUjFSOFQK11'
        secretapikey = 'crGr--X10duyKqUKKtfgfS5E0FrgsF2jz439eSrHjlkVMgS3'
        connect()
        mainmenu()

    elif choice == '3':
        exit()
```

The connect function uses the API key inputs from the login function, to connect to the BitMex server. Once the connection is successful, the program will display “Connection Established”.

```
#connect
def connect():
    global ws
    global client
    print("\nConnecting to BitMex servers...")
    ws = BitMEXWebsocket(endpoint="https://testnet.bitmex.com/api/v1", symbol=pair,
api_key=apikey, api_secret=secretapikey)
    client = bitmex(test=True, api_key=apikey, api_secret=secretapikey)
    print("Connection Established \n")
```

The logout function disconnects the user's BitMex trading account from the BitMex servers. Therefore, no actions can take place until the user has logged in again using the login function.

```
#logout
def logout():
    global logout
    logout = requests.get('https://testnet.bitmex.com/api/v1/user/logout')
    print('\nLogout Successful \n')
    login()
```

The mainmenu function simply displays all the possible actions which the user can take in order to setup an optimal trading environment for the bot to run.

```
# main menu
def mainmenu():
    global break_program
    choice  = input('''
Main Menu:
1. Start Bot
2. View Balance
3. View Trading History
4. Configure Trade Settings
5. Logout

''')
    if choice == '1':
       break_program = False
       startbot()
    elif choice == '2':
       printviewbalance()
    elif choice == '3':
       viewtradinghistory()
    elif choice == '4':
       configuretradesettings()
    elif choice == '5':
       logout()
       login()
```

The getrsi function calculates the current RSI level by scraping the market price and previous closing prices from the BitMex API URL with parameters specified earlier on by the user. The RSI level is essential in order to determine whether to execute a trade or not.

```
#calculate rsi
def getrsi():
    global break_program
    def on_press(key):
        global break_program
        if key == Key.shift:
            break_program = True
            mainmenu()

    with Listener(on_press=on_press) as listener:
        while break_program == False:
            global period
            global timeframe
            global pair
            #apikey = '1Bk8AeOWOjO2DCUjFSOFQK11'
            #secretapikey = 'crGr--X10duyKqUKKtfgfS5E0FrgsF2jz439eSrHjlkVMgS3'
            #ws = BitMEXWebsocket(endpoint="https://testnet.bitmex.com/api/v1", symbol=pair,
api_key=apikey, api_secret=secretapikey)
            souparray = []
            closingprices = []
            closingprices.append(float(marketprice()))

            recenttrades = ("https://testnet.bitmex.com/api/v1/trade/bucketed?binSize={}
&partial=true&symbol={}&count={}&reverse=true").format(timeframe, pair, str(period+1))
            r = requests.get(recenttrades)
            soup = str(BeautifulSoup(r.content, 'lxml'))
            soup = soup.replace('<html><body><p>[{', '')
            soup = soup.replace('}]</p></body></html>', '')
            soupstring = ""
            slicedsoup = [char for char in soup]
            while True:
                x = 0
                if slicedsoup[x] == '{':
                    del slicedsoup[x]
                    break
                elif slicedsoup[x] != '{':
                    del slicedsoup[x]
            for ele in slicedsoup:
                soupstring += ele

            souparray = soupstring.split(',')
            for x in range(5, len(souparray), 13):
                souparray[x] = re.sub(r'[a-z]+', '', souparray[x], re.I)
                souparray[x] = (souparray[x])[3:]

                closingprices.append(float(souparray[x]))

            closingprices = closingprices[::-1]
            differences = []
            upwardmovement = []
            downwardmovement = []
            for x in range(period):
                differences.append(closingprices[x+1]-closingprices[x])
            for x in differences:
                if x > 0:
                    upwardmovement.append(x)
                    downwardmovement.append(0)
                elif x < 0:
                    downwardmovement.append(abs(x))
                    upwardmovement.append(0)
                elif x == 0:
                    upwardmovement.append(0)
                    downwardmovement.append(0)
            currentavgupwardsmov = sum(upwardmovement) / len(upwardmovement)
            currentavgdownwardsmov = sum(downwardmovement) / len(downwardmovement)
            rs = currentavgupwardsmov / currentavgdownwardsmov
            rsi = 100 - (100/(rs+1))
            return rsi

        listener.join()
```

The on_press function allows a function (in this case, the getrsi and startbot functions) to carry on executing code until the Shift key is pressed. If the user presses the Shift key, it will stop both of these functions and therefore, prevent the bot from running. This is made known to the user when they start the bot. Once the Shift key is pressed, the user is directed back to the ‘Main Menu’.

```
#listen for Shift key
break_program = False
    def on_press(key):
        global break_program
        if key == Key.shift:
            break_program = True
            mainmenu()


    with Listener(on_press=on_press) as listener:
        while break_program == False:

            ***CODE INSERTED HERE***

            time.sleep(1)
        listener.join()
```

The determine function is a vital part of the program as it consists of the formulas used to make the trade decisions. It determines whether the current RSI level is within the buy or sell thresholds, and if not, it does nothing. It also prevents an overload of buy or sell orders by introducing a ‘buycounter’ and ‘sellcounter’. This means that once a buy order is executed, it cannot be followed by another buy order unless a sell order is executed.

```
#determine whether to Buy, Sell or wait
def determine():
    global sellcounter
    global buycounter
    global mult
    while True:
        time.sleep(10) #to prevent request overload
        if getrsi() <= lowerband and (buycounter == 1): # oversold - buy signal
            mult = 1
            sellcounter = 1
            buycounter = 0
            placeorder()

        if getrsi() >= upperband and (sellcounter == 1): # overbought - sell signal
            mult = -1
            buycounter = 1
            sellcounter = 0
            placeorder()
```

The startbot function is called when the user starts the bot from the ‘Main Menu’. It calls the determine function, which then calls the getrsi function, which in turn, calls the placeorder function. These functions operate in a loop as they call each other to make trade decisions. Therefore, the startbot function initiates this loop and cannot be called again once the loop starts (unless the Shift key is pressed).

```
#start bot
def startbot():
    global capitalproportion
    global percsafe
    global percmodrisky
    global percrisky
    global percentage
    global risklevel
    global index
    global lowerboundindex
    global upperboundindex
    global period
    global timeframe
    global pair

    break_program = False
    def on_press(key):
        global break_program
        if key == Key.shift:
            break_program = True
            mainmenu()

    with Listener(on_press=on_press) as listener:
        while break_program == False:
            global capitalproportion
            global percsafe
            global percmodrisky
            global percrisky
            global percentage
            global risklevel
            global index
            global lowerboundindex
            global upperboundindex
            global period
            global timeframe
            global pair
            print("\nBot Running\n")
            print("Press Shift to Stop Bot\n")
            rsi_risk_level = (lowerband / upperband) / period
            risklevel = rsi_risk_level
            index = 0.03061225 #(30/70)/14
            lowerboundindex = index - 0.01785714
            upperboundindex = index + 0.04761905
            if risklevel > lowerboundindex and risklevel < upperboundindex:
                percentage = float(percmodrisky)
            elif risklevel <= lowerboundindex:
                percentage = float(percsafe)
            elif risklevel >= upperboundindex:
                percentage = float(percrisky)

            capitalproportion = int(math.ceil((percentage/100) *
(returnviewbalance()*marketprice())))

            determine()

            time.sleep(1)
        listener.join()
```

The printviewbalance and returnviewbalance functions both carry out the same operations, however the former displays the balance on-screen, and the latter returns the balance when needed in other functions during calculations. As forward referencing may be required, both functions had to be made instead of using ‘print(returnviewbalance)’.

```
#displays balance on-screen
def printviewbalance():
    global balance
    #ws = BitMEXWebsocket(endpoint="https://testnet.bitmex.com/api/v1", symbol=pair,
api_key=apikey, api_secret=secretapikey)
    funds = dict(ws.funds())
    balance = float(funds['amount'] / 100000000)
    print('\nAvailable Balance: ', balance/100000000, 'BTC')
    choice = str(input("\n3. Back\n"))
    if choice == '3':
        mainmenu()

#returns balance value to other functions when required
def returnviewbalance():
    global returnbalance
    #ws = BitMEXWebsocket(endpoint="https://testnet.bitmex.com/api/v1", symbol=pair,
api_key=apikey, api_secret=secretapikey)
    returnfunds = dict(ws.funds())
    returnbalance = float(returnfunds['amount'] / 100000000)
    return returnbalance
```

The viewtradinghistory function enables the user to see all of the trades executed by the bot. The user can choose how to display the orders; they can view the most recent orders first, or the oldest orders first.

```
#view bot’s executed orders
def viewtradinghistory():
    if len(history) == 0 or None:
        print("No Order History Available")
        goback = str(input("\n3. Back\n"))
        if goback == '3':
            mainmenu()
    else:
        choice = str(input('''
Display Orders By:
1. Newest Trades First
2. Oldest Trades First
3. Back

'''))
        if choice == '1':
            for x in history[::-1]:
                print(x)
            newback = str(input("\n3. Back\n"))
            if newback == '3':
                mainmenu()
        elif choice == '2':
            for x in history:
                print(x)
            oldback = str(input("\n3. Back\n"))
            if oldback == '3':
                mainmenu()
        elif choice == '3':
            mainmenu()
```

The configuretradesettings function can be called from the ‘Main Menu’. It allows the user to navigate to change their RSI settings or Risk Management settings.

```
#menu option - configure trading settings
def configuretradesettings():
    choice = input('''
Trade Settings:
1. RSI Settings
2. Risk Management Settings
3. Back

''')

    if choice == '1':
        rsisettings()
    elif choice == '2':
        riskmanagementsettings()
    elif choice == '3':
        mainmenu()
```

The configuretradesettings leads onto the rsisettings function and the riskmanagementsettings function. The rsisettings function enables the user to adjust the bot to make safe trade decisions, moderately risky trade decisions, or risky trade decisions. This is done by changing the upperbound, lowerbound and period of the RSI indicator. The riskmanagementsettings function allows the user to access the risklevelsetting function and the capitalproportionmanagement function. These indicate which of the three risk modes the bot is currently trading according to.

```
#change rsi settings
def rsisettings():
    choice = str(input('''
RSI Settings:
1. Upper Band
2. Lower Band
3. Period
4. Back

'''))
    if choice == '1':
        global upperband
        print(("\nUpperband currently set to {}.").format(upperband))
        op1 = str(input('''
Would you like to change the upperband?
1. Yes
2. No
3. Back

'''))
        if op1 == '1':
            upperband = int(input("\nEnter upperband level: "))
            rsisettings()
        elif op1 == '2' or '3':
            rsisettings()

    elif choice == '2':
```

```
        global lowerband
        print(("\nLowerband currently set to {}.").format(lowerband))
        op2 = str(input('''
Would you like to change the lowerband?
1. Yes
2. No
3. Back

'''))
        if op2 == '1':
            lowerband = int(input("\nEnter lowerband level: "))
            rsisettings()
        elif op2 == '2' or '3':
            rsisettings()

    elif choice == '3':
        global period
        print(("\nPeriod currently set to {}.").format(period))
        op3 = str(input('''
Would you like to change the period?
1. Yes
2. No
3. Back

'''))
        if op3 == '1':
            period = int(input("\nEnter period level: "))
            rsisettings()
        elif op3 == '2' or '3':
            rsisettings()

    elif choice == '4':
        configuretradesettings()



#view/change risk management settings

def riskmanagementsettings():
    choice = input('''
Risk Management Settings:
1. Risk Level Settings
2. Capital Proportion Management Settings
3. Back

''')
    if choice == '1':
        risklevelsetting()
    elif choice == '2':
        capitalproportionmanagementsetting()
    elif choice == '3':
        configuretradesettings()
```

The risklevelsetting function undergoes a calculation which works out how risky the bot's current settings are. It uses the default values of 70 for the upperband, 30 for the lowerband and 14 for the period to set an index value. This index value represents a moderately risky trading environment. A lowerbandindex and upperbandindex are also calculated using the default values. They indicate a safe trading environment and a risky trading environment, respectively. Once the risk level is calculated, the function displays the bot's current trading environment so the user can understand the potential benefits and consequences of running the bot, and change them is desired.

```
# manually set risk level
def risklevelsetting():
    global risklevel
    global index
    global lowerboundindex
    global upperboundindex
    global percentage
    global capitalproportion
    global percsafe
    global percmodrisky
    global percrisky
    global period
    global timeframe
    global pair

    rsi_risk_level = (lowerband / upperband) / period
    risklevel = rsi_risk_level
    index = 0.03061225 #(30/70)/14
    lowerboundindex = index - 0.01785714
    upperboundindex = index + 0.04761905

    if risklevel <= lowerboundindex:
        percentage = float(percsafe)
        print('''
Current trading settings are relatively safe.
(Disclaimer: Your capital is always at risk.)
''')
        #capitalproportionmanagementsetting(percsafe)
    elif risklevel > lowerboundindex and risklevel < upperboundindex:
        percentage = float(percmodrisky)
        print('''
Current trading settings are moderately risky.
(Disclaimer: Your capital is always at risk.)
''')
        #capitalproportionmanagementsetting(percmodrisky)
    elif risklevel >= upperboundindex:
        percentage = float(percrisky)
        print('''
Current trading settings are risky.
(Disclaimer: Your capital is always at risk.)
''')
        #capitalproportionmanagementsetting(percrisky)

    choice = str(input('''
Would you like to change trading settings?
1. Yes
2. No
3. Back

'''))
    if choice == '1':
        configuretradesettings()
    elif choice == '2' or '3':
        riskmanagementsettings()
```

The capitalproportionmanagementsetting function calculates how much capital to allocate towards each trade depending on the bot's trading environment; for example, if the environment is risky, the bot will allocate 2% of the user's account balance towards each trade (this value can be changed manually by the user but is defaulted to 2).

```
# manually or automatically set proportional of capital per trade
def capitalproportionmanagementsetting():
    global capitalproportion
    global percsafe
    global percmodrisky
    global percrisky
    global percentage
    global risklevel
    global index
    global lowerboundindex
    global upperboundindex
    global period
    global timeframe
    global pair

    rsi_risk_level = (lowerband / upperband) / period

    risklevel = rsi_risk_level
    index = 0.03061225 #(30/70)/14
    lowerboundindex = index - 0.01785714
    upperboundindex = index + 0.04761905

    if risklevel > lowerboundindex and risklevel < upperboundindex:
        percentage = float(percmodrisky)
    elif risklevel <= lowerboundindex:
        percentage = float(percsafe)
    elif risklevel >= upperboundindex:
        percentage = float(percrisky)

    print(("\nCurrent proportion of capital allocated per trade is: {}
%").format(percentage))
    capitalproportion = (percentage/100) * returnviewbalance()

    choice = str(input('''
Capital Proportion Management Settings:
1. Capital Proportion for Relatively Safe Trades
2. Capital Proportion for Moderately Risky Trades
3. Capital Proportion for Risky Trades
4. Back
'''))
    if choice == '1':
        print(("Current capital proportion for relatively safe trades is: {}
%").format(percsafe))
        op1 = str(input('''
Would you like to change this proportion?
1. Yes
2. No
3. Back

'''))
        if op1 == '1':
            percsafe = float(input("Enter capital proportion for relatively safe trades
(%): "))
            capitalproportionmanagementsetting()
        elif op1 == '2':
            capitalproportionmanagementsetting()
        elif op1 == '3':
            capitalproportionmanagementsetting()
```

```
    elif choice == '2':
        print(("Current capital proportion for moderately risky trades is: {}
%").format(percmodrisky))
        op2 = str(input('''
Would you like to change this proportion?
1. Yes
2. No
3. Back

'''))
        if op2 == '1':
            percmodrisky = float(input("Enter capital proportion for moderately risky
trades (%): "))
            capitalproportionmanagementsetting()
        elif op2 == '2':
            capitalproportionmanagementsetting()
        elif op2 == '3':
            capitalproportionmanagementsetting()


    elif choice == '3':
        print(("Current capital proportion for risky trades is: {}%").format(percrisky))
        op3 = str(input('''
Would you like to change this proportion?
1. Yes
2. No
3. Back

'''))
        if op3 == '1':
            percrisky = float(input("Enter capital proportion for risky trades (%): "))
            capitalproportionmanagementsetting()
        elif op3 == '2':
            capitalproportionmanagementsetting()
        elif op3 == '3':
            capitalproportionmanagementsetting()


    elif choice == '4':
        riskmanagementsettings()
```

The marketprice function uses the BitMex API to obtain details about the most recent candlestick (time interval) for a cryptocurrency pair. Since this returns many values, the data has been converted into a dictionary data type which can then easily distinguish ‘lastPrice’ from the other values. The last price is the market price.

```
# get market price
def marketprice():
    data = dict(ws.get_instrument())
    marketprice = data['lastPrice']
    return marketprice
```

The placeorder function is called when the determine function triggers a buy or sell order. It sends an API order call which executes the trade, with the specified parameters, in the BitMex trading platform. This function also displays the details of each trade once they're executed. The displayed messages are also appended to an array which is used in the viewtradinghistory function. The function waits for one minute before calling the determine function and continuing the loop. This is to prevent an overload of API requests which could crash the program as BitMex have API call limits.

```
#place order
def placeorder():
    global mult
    global order
    global buydatamsg
    global selldatamsg
    global history
    price = marketprice()
    logtime = time.ctime(time.time())
    order = client.Order.Order_new(symbol=pair, orderQty=capitalproportion*mult,
price=price).result()
    if capitalproportion*mult > 0:
        buymsg = 'Buy Order Executed'
        print(buymsg)
        buydatamsg = ('{}: Bought {} contract(s) of {} at {}').format(str(logtime),
str(capitalproportion), pair, str(price))
        history.append(buydatamsg)
        print(buydatamsg)
    elif capitalproportion*mult < 0:
        sellmsg = 'Sell Order Executed'
        print(sellmsg)
        selldatamsg = ('{}: Sold {} contract(s) of {} at {}').format(str(logtime),
str(capitalproportion), pair, str(price))
        history.append(selldatamsg)
        print(selldatamsg)

    time.sleep(60) #waits 1 minute to prevent overload of requests
    determine()
```

3. Program Setup & Configuration

3.1 Installing Python

This program is written using Python 3.8.

If you do not already have Python 3.8 installed, then please install it from here: python.org/downloads/

3.2 Installing an Interpreter

If you do not already have a Python interpreter, then please install one.

If you installed Python 3 from the link above, or you already have Python installed on your computer, then you should already have the IDLE interpreter installed.

However if not, PyCharm Edu is recommended.

You can download it here:
https://www.jetbrains.com/pycharm-edu/download/index.html

3.3 Creating a BitMex Account

You can use the API Keys already specified in the program by entering ‘2’ on the Login page. However, if you would like to use your own for future use please navigate to https://testnet.bitmex.com/ and register an account.

From there, click the ‘API’ tab in the header and navigate to ‘API Key Management’ on the left-hand side of the page. From here, you are able to name your API Key and set API key permissions to ‘Order’ (not ‘Order Cancel’). Click ‘Create API Key’ once you have filled in the required fields. Once you have received your BitMex API Keys, press ‘1’ on the Login page and enter your keys into the fields when prompted.

3.4 Opening & Running the File

Download the Python file from the email attachment.

Once it has downloaded, double click the file to run the program in your interpreter.

4. References

Platform & Execution Testing:
https://testnet.bitmex.com/

API Endpoints:
https://github.com/BitMEX/api-connectors/tree/master/official-ws/python
https://github.com/BitMEX/api-connectors/blob/master/swagger.json
https://testnet.bitmex.com/api/explorer/

RSI Indicator Calculations:
https://www.youtube.com/watch?v=Dt0KQg52c6c
https://www.marketvolume.com/quotes/calculatersi.asp

Indicator Testing:
https://www.tradingview.com

Programming Help & Support:
https://stackoverflow.com/