AMC ENGINEERING COLLEGE

Department of Information Science and Engineering

AI TRADING BOT

GROUP 14

18SCP77 – PROJECT WORK PHASE - II

GUIDE	Prof. NAGASHREE K T

USN	NAME
1AM20IS110	VISHARAD VIDYASAGAR (LEADER)
1AM20IS086	SHAIK ARSHIYA
1AM20IS095	SNEHA NAGARAJ SHET
1AM20IS109	VINUTA S SINNUR

INTRODUCTION

AI trading bot is a computer program designed to execute buy or sell orders on financial instruments, such as stocks or cryptocurrencies, without direct human intervention. Unlike traditional trading methods, AI trading bots operate tirelessly, 24/7, analyzing market data, identifying patterns, and making split-second decisions based on predefined rules or learned behaviors.

The significance of AI in trading lies in its ability to process vast amounts of historical and real-time data at speeds beyond human capacity. Machine learning algorithms within these bots can discern intricate patterns and trends, adapting strategies to changing market conditions. By harnessing computational power, AI trading bots aim to optimize trading outcomes, enhance portfolio management, and minimize emotional biases inherent in human decision-making

PROBLEM STATEMENT

Investing in stock market is not an easy task, as it needs learning and practice for years. Even after this, people end up getting losses and lose their money. Seeing the stock market pattern daily and analyzing is also a very big task.

AI trading bots employ sophisticated algorithms and machine learning techniques to analyze the collected data. They identify patterns, trends, correlations and other indicators that may suggest potential trading opportunities beyond regular financial analysis. Based on the analysis of the data, the AI trading bot develops trading strategies or uses predefined strategies programmed into its system. The bot then executes trades on behalf of the user, sending orders to the exchange or trading platform. AI trading bots incorporate risk management techniques to control and minimize potential losses. These techniques may include setting stop-loss orders, position-sizing algorithms or implementing trailing stops to protect profits.

ABSTRACT

The primary objective is to achieve maximum profits with minimal risk, surpassing the results of conventional human trading strategies. To accomplish this, the project will delve into a range of advanced AI techniques and strategies such as Golden Cross and Death Cross, assessing their performance across varying market conditions. The AI Trading Bot holds the promise of reshaping the finance industry, making trading more efficient and accessible for everyone.

The Project utilizes Meta Trader5 and Pine Connector tools for Comprehensive price analysis, access real -time market data, managing trading accounts and making the trading process seamless and efficient. Beyond its technical goals, the project is committed to adhering to all ethical standards and financial trading regulations

LITERATURE SURVEY

1. **Zhang et. al., [1]** "A Deep Reinforcement Learning Approach for Stock Trading (2018)". This paper explores the application of Deep Reinforcement Learning (DRL) for stock trading, specifically focusing on policy gradient methods. The authors propose a DRL agent that interacts with a simulated stock market environment to learn optimal trading strategies.

2. **Jiang et. al., [2]** "An Ensemble Learning Framework for Algorithmic Trading" (2019, IEEE Transactions on Computational Intelligence and Financial Economics). This paper proposes an ensemble learning framework for algorithmic trading, combining multiple machine learning models to improve prediction accuracy and generate profitable trading signals.

3. **Salah Bouktif et. al., [3]** "Augmented Textual Features-Based Stock Market Prediction". This paper explores the use of textual features alongside traditional historical data to improve stock market prediction accuracy. The paper proposes a novel approach that utilizes sentiment analysis, n-grams, and customized textual features extracted from news articles and financial reports, in addition to historical stock prices and technical indicators.

LITERATURE SURVEY

4. **Arya Yudhi Wijaya et. al., [4]** “Stock Price Prediction with Golden Cross and Death Cross on Technical Analysis Indicators Using Long Short-Term Memory (2022)”. This paper explores the use of Long Short-Term Memory (LSTM) networks for stock price prediction, incorporating Golden Cross and Death Cross signals as additional technical indicators. The paper investigates the effectiveness of combining LSTM models with Golden Cross and Death Cross, two popular technical analysis indicators that signal potential trend changes from bearish to bullish (Golden Cross) or vice versa (Death Cross).

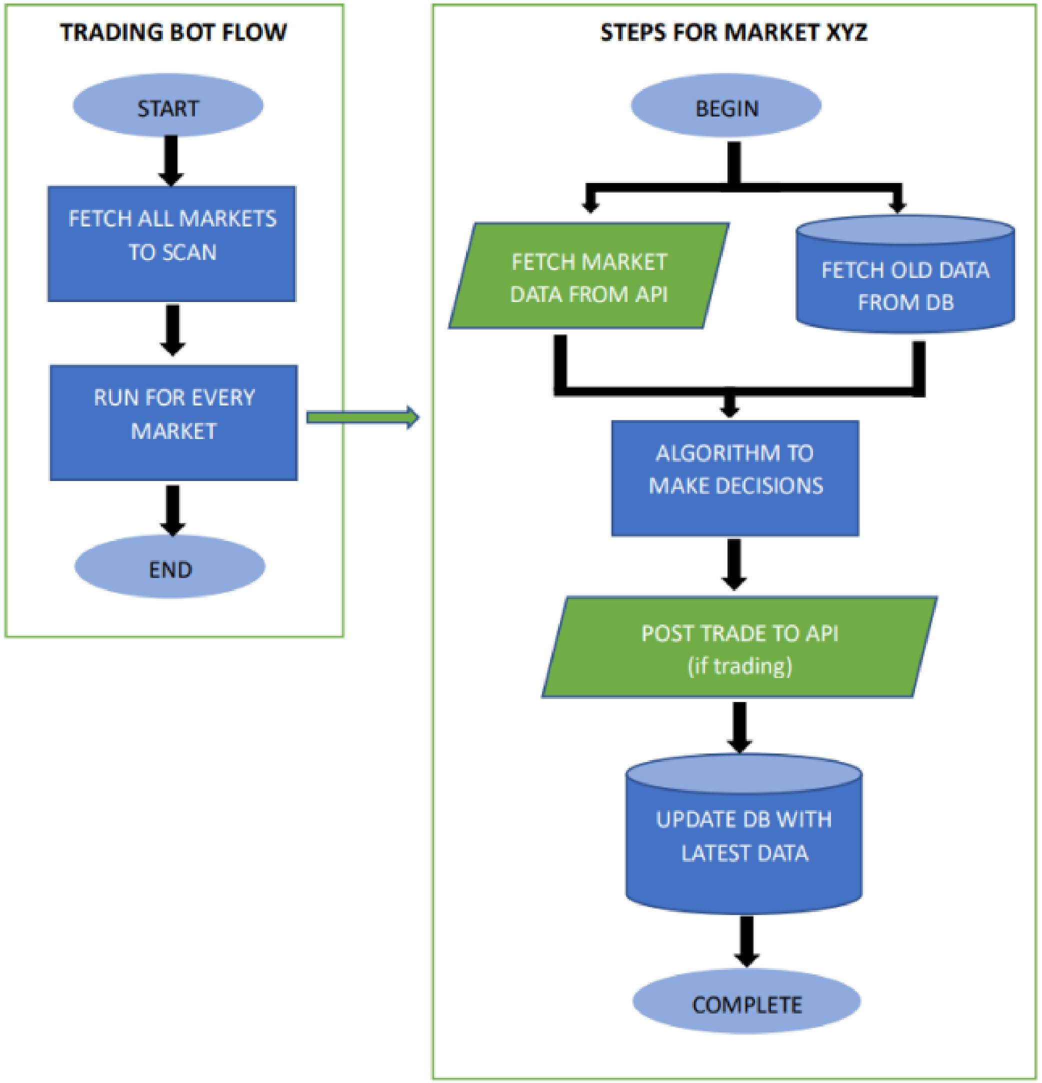
5. **Gourav Bathla et. al., [5]** "Stock Price prediction using LSTM and SVR (2020)". The paper highlights that stock price movement is non-linear and complex, and traditional approaches such as Linear Regression and Support Vector Regression were used but the accuracy was not adequate. The paper introduces a method where LSTM is compared with SVR using various stock index data such as S&P 500, NYSE, NSE, BSE, NASDAQ, and Dow Jones Industrial Average for experiment analysis.

LITERATURE SURVEY

6. **Xiaojian Weng et. al.,[6]** "Stock Price Prediction Based on LSTM and Bert (2022)". This model addresses the limitations of traditional stock forecasting models, which are based on statistical regression models and often struggle to characterize the influential relationships between multiple variables, leading to large prediction errors. The proposed model first calculates the investor sentiment before the stock opening by fine-tuning the BERT model.

7. Sparsh Vohara et. al., [7] "Stock Price Trend Analysis and Prediction of Closing Price Using LSTM (2023)". Accurate prediction of stock market returns is a challenging task due to the volatile and nonlinear nature of those returns. Investment returns depend on many factors including political conditions, local and global economic conditions, company-specific performance, and many others. Recently, the interest in applying Artificial Intelligence in making trading decisions has been growing rapidly with numerous research papers published each year addressing this topic.

ARCHITECTURE DIAGRAM


TRADING BOT FLOW
START
FETCH ALL MARKETS TO SCAN
RUN FOR EVERY MARKET
END
STEPS FOR MARKET XYZ
BEGIN
FETCH MARKET DATA FROM API
FETCH OLD DATA FROM DB
ALGORITHM TO MAKE DECISIONS
POST TRADE TO API (if trading)
UPDATE DB WITH LATEST DATA
COMPLETE

DATA REQUIREMENTS

To build an AI trading bot project using Python, we would typically need the following data and resources:

- **Trading Platform API**: You need access to a trading platform that provides APIs for placing orders. Examples include Alpaca3 and Zerodha4. We are using MetaTrader5 MT5 API and Pine Connector.
- **MetaTrader 5** (MT5) is a trading platform that allows users to trade stocks, CFDs, forex, and futures. It's available on desktop, mobile, and web, and is designed for non-forex and U.S. markets.
- **API Credentials:** You need to create an API key and secret to access your trading account.
- **Trading Account:** You need a trading account with sufficient funds to execute the orders.

DATA REQUIREMENTS

- **Market Data:** You need access to real-time or historical market data for analysis and decision making.

- **Python Libraries:** You would need various Python libraries such as pandas, NumPy, matplotlib for data manipulation and analysis. For interacting with the trading platform, you need specific libraries.

- **Back testing Framework:** To validate your trading strategy, you need a back testing framework that can simulate trading strategies against historical data.

MODULES

- MODULE I – Fetching Data
- MODULE II – Connecting with MT5
- MODULE III – Strategy
- MODULE IV – Automation

MODULE I : Fetching Data & MODULE II : Connection

- The Real time data for a specific Ticker or Symbol (Stock Name), for a period of time is taken. The data is fetched using the MT5 of the IC Market broker.

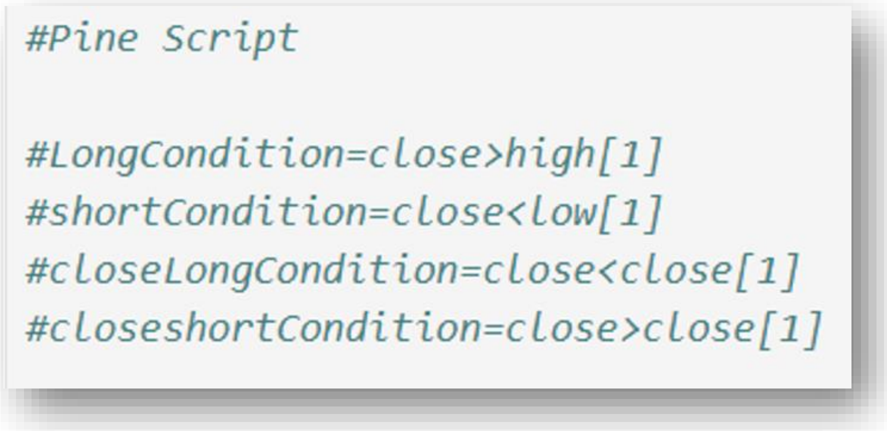
- Pine connector is used to connect our code with the IC Market MT5, so that we can initialize the Market Trader automatically instead of opening it manually.

- The data retrieved can be of specific intervals, depending on the user's need. It can be for 1 hour (H1), 2 hours (H2), 1 day (D1) to months (M1, M2, ..).

- The data is then loaded as per the need with all basic data like, time, open, close, high, low prices of the stocks.

MODULE III : Strategy

- The working strategy for the bot is pretty easy and simple.
- The bot BUYS a stock, and then analyzes the data continuously, when the STOP LOSS or TAKE PROFIT is met, the bot SELLS the stock and completes a Trade.
- The same continues if the bot has already executed a BUY Trade, so it works on the LSTM conditions, to perform the next Trades.



```
#Pine Script

#LongCondition=close>high[1]
#shortCondition=close<low[1]
#closeLongCondition=close<close[1]
#closeshortCondition=close>close[1]
```

MODULE IV : Automation

- The bot starts to work automatically instead of doing everything manually.
- The bot will fetch data, buy, sell stocks on it’s own until the conditions are matched to do so.
- The conditions for the TAKE PROFIT or STOP LOSS is done by using a user defined function, which checks for specific conditions to execute a Trade.
- The bot can work for n number of times, and on a user’s specific needs.
- Which allows the user to follow up their other works, rather than continuously checking the data every time manually
- The bot can be put to sleep as per the user’s need as well.

METHODOLOGY

LSTM (LONG SHORT-TERM MEMORY)

LSTM is a powerful tool which helps to improve the accuracy of the trading bot. It helps in handling data in a sequential order, it incorporates a memory into a model structure which helps to compute multiple functions at every stage.it detects the trading signals, it prepares the data and utilizes the indicators for daily analysis. Overall, its helps in the predictions .and maintains the bot accuracy and efficiency with better performance.

The integration of LSTM aims to improve the ability to predict trading signals accurately. In basic trading strategies, instead of using the current day's closing price to generate technical indicators and make trading decisions, an LSTM model is applied to predict the future closing price.

GOLDEN CROSSOVER

The Golden Cross is a bullish phenomenon when the 50-day moving average crosses above the 200-day moving average. When the market is in a long-term downtrend, the 50-day moving average is below the 200-day moving average. However, no downtrend lasts forever. So, when a new uptrend begins, the 50-day moving average must cross above the 200- day moving average and that's known as the Golden Cross.

Indicators

• 50-Day SMA (Simple Moving Average), represents the short-term trend.

• 200-Day SMA (Simple Moving Average), represents the long-term trend.

Buy Signal & Sell Signal

• Buy Signal, the 50-Day Average Crosses **ABOVE** the 200-Day Moving average, indicating that there may be a bull-market on the horizon.

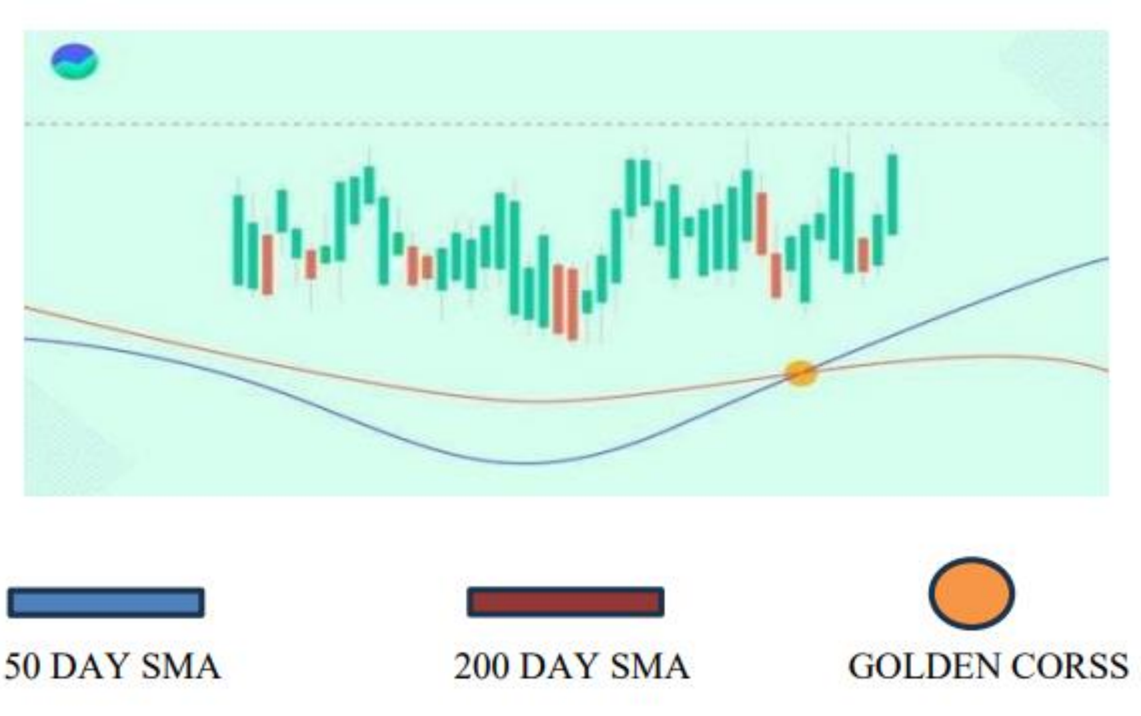
• Sell Signal, the 50-Day Average Crosses **BELOW** the 200-Day Moving average, indicating that there may be a bear-market on the horizon

GOLDEN CROSSOVER

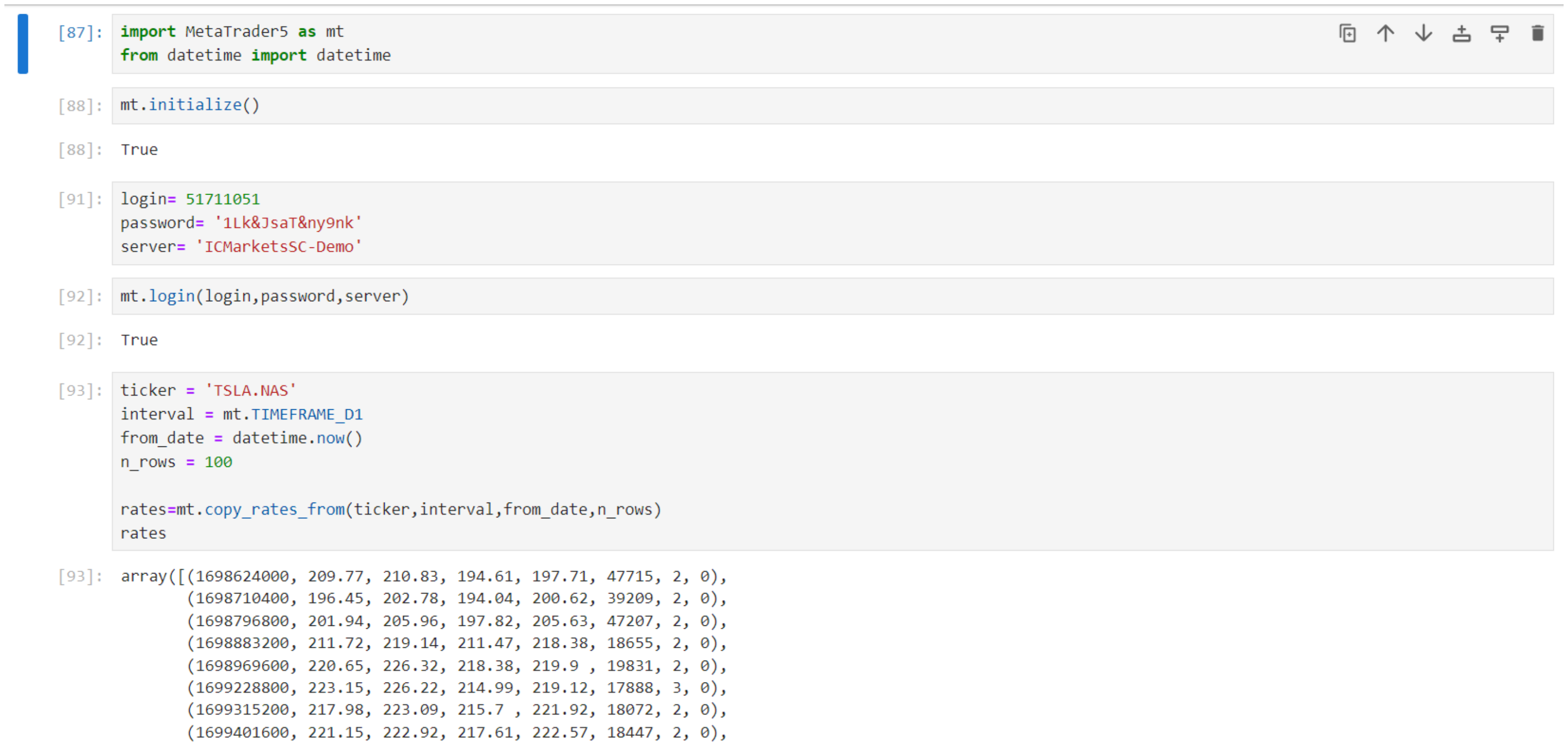
Buy Signal & Sell Signal

• Buy Signal, the 50-Day Average Crosses **ABOVE** the 200-Day Moving average, indicating that there may be a bull-market on the horizon.

• Sell Signal, the 50-Day Average Crosses **BELOW** the 200-Day Moving average, indicating that there may be a bear-market on the horizon

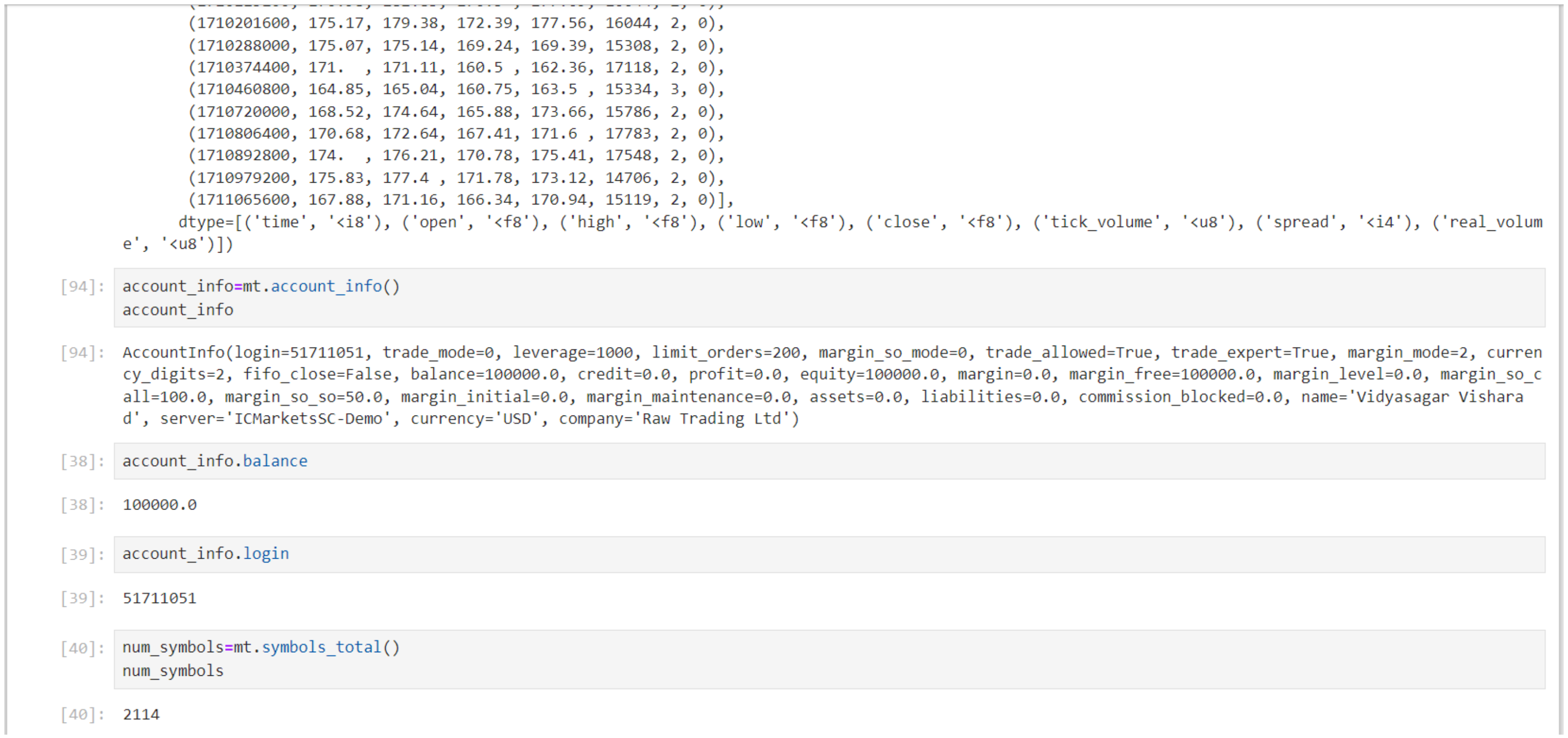


IMPLEMENTATION



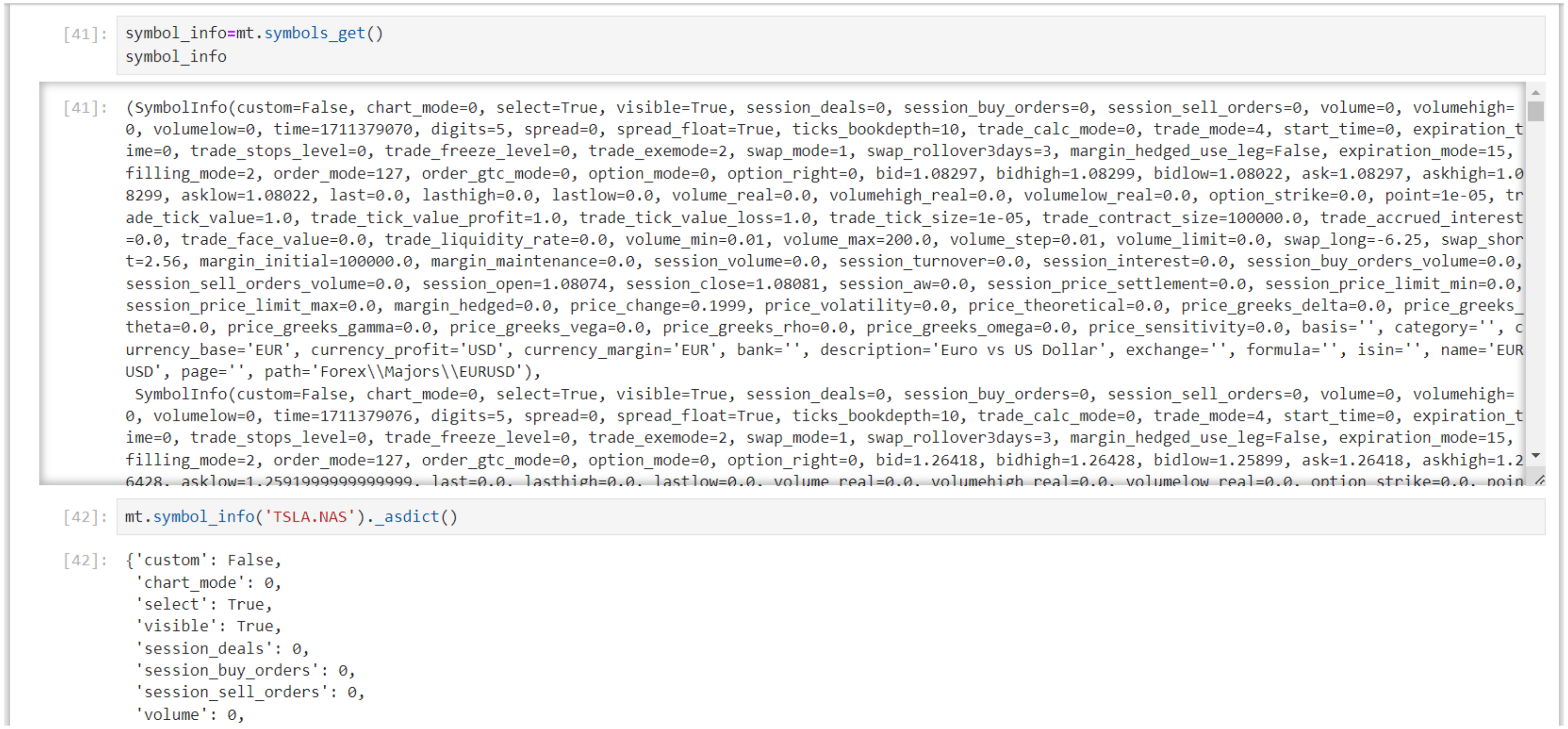
```
[87]: import MetaTrader5 as mt
      from datetime import datetime

[88]: mt.initialize()

[88]: True

[91]: login= 51711051
      password= '1Lk&JsaT&ny9nk'
      server= 'ICMarketsSC-Demo'

[92]: mt.login(login,password,server)

[92]: True

[93]: ticker = 'TSLA.NAS'
      interval = mt.TIMEFRAME_D1
      from_date = datetime.now()
      n_rows = 100

      rates=mt.copy_rates_from(ticker,interval,from_date,n_rows)
      rates

[93]: array([(1698624000, 209.77, 210.83, 194.61, 197.71, 47715, 2, 0),
             (1698710400, 196.45, 202.78, 194.04, 200.62, 39209, 2, 0),
             (1698796800, 201.94, 205.96, 197.82, 205.63, 47207, 2, 0),
             (1698883200, 211.72, 219.14, 211.47, 218.38, 18655, 2, 0),
             (1698969600, 220.65, 226.32, 218.38, 219.9 , 19831, 2, 0),
             (1699228800, 223.15, 226.22, 214.99, 219.12, 17888, 3, 0),
             (1699315200, 217.98, 223.09, 215.7 , 221.92, 18072, 2, 0),
             (1699401600, 221.15, 222.92, 217.61, 222.57, 18447, 2, 0),
```

IMPLEMENTATION



```
        (1710201600, 175.17, 179.38, 172.39, 177.56, 16044, 2, 0),
        (1710288000, 175.07, 175.14, 169.24, 169.39, 15308, 2, 0),
        (1710374400, 171.  , 171.11, 160.5 , 162.36, 17118, 2, 0),
        (1710460800, 164.85, 165.04, 160.75, 163.5 , 15334, 3, 0),
        (1710720000, 168.52, 174.64, 165.88, 173.66, 15786, 2, 0),
        (1710806400, 170.68, 172.64, 167.41, 171.6 , 17783, 2, 0),
        (1710892800, 174.  , 176.21, 170.78, 175.41, 17548, 2, 0),
        (1710979200, 175.83, 177.4 , 171.78, 173.12, 14706, 2, 0),
        (1711065600, 167.88, 171.16, 166.34, 170.94, 15119, 2, 0)],
       dtype=[('time', '<i8'), ('open', '<f8'), ('high', '<f8'), ('low', '<f8'), ('close', '<f8'), ('tick_volume', '<u8'), ('spread', '<i4'), ('real_volum
e', '<u8')])

[94]: account_info=mt.account_info()
      account_info

[94]: AccountInfo(login=51711051, trade_mode=0, leverage=1000, limit_orders=200, margin_so_mode=0, trade_allowed=True, trade_expert=True, margin_mode=2, curren
      cy_digits=2, fifo_close=False, balance=100000.0, credit=0.0, profit=0.0, equity=100000.0, margin=0.0, margin_free=100000.0, margin_level=0.0, margin_so_c
      all=100.0, margin_so_so=50.0, margin_initial=0.0, margin_maintenance=0.0, assets=0.0, liabilities=0.0, commission_blocked=0.0, name='Vidyasagar Vishara
      d', server='ICMarketsSC-Demo', currency='USD', company='Raw Trading Ltd')

[38]: account_info.balance

[38]: 100000.0

[39]: account_info.login

[39]: 51711051

[40]: num_symbols=mt.symbols_total()
      num_symbols

[40]: 2114
```

IMPLEMENTATION



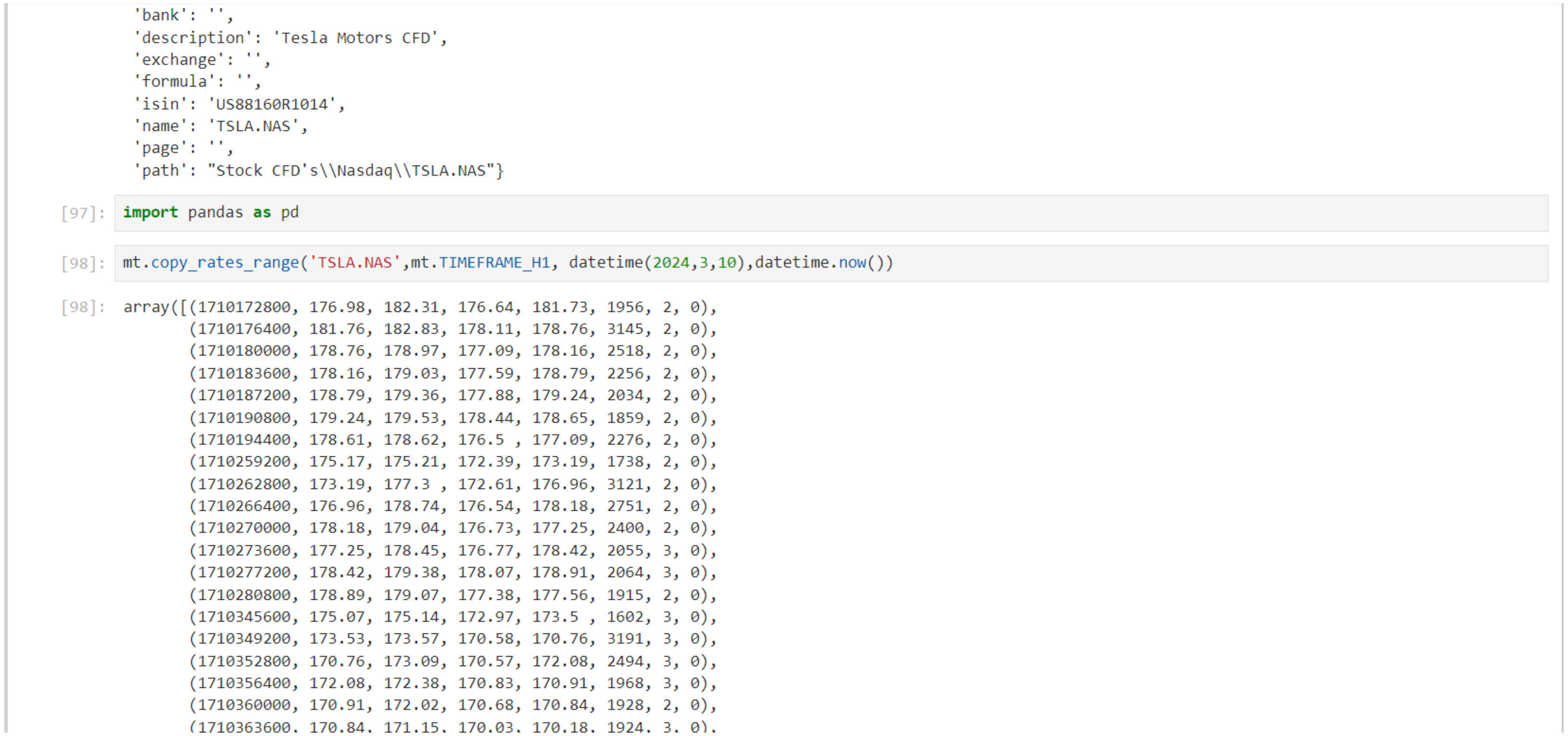
```
[41]: symbol_info=mt.symbols_get()
      symbol_info
```

```
[41]: (SymbolInfo(custom=False, chart_mode=0, select=True, visible=True, session_deals=0, session_buy_orders=0, session_sell_orders=0, volume=0, volumehigh=
      0, volumelow=0, time=1711379070, digits=5, spread=0, spread_float=True, ticks_bookdepth=10, trade_calc_mode=0, trade_mode=4, start_time=0, expiration_t
      ime=0, trade_stops_level=0, trade_freeze_level=0, trade_exemode=2, swap_mode=1, swap_rollover3days=3, margin_hedged_use_leg=False, expiration_mode=15,
      filling_mode=2, order_mode=127, order_gtc_mode=0, option_mode=0, option_right=0, bid=1.08297, bidhigh=1.08299, bidlow=1.08022, ask=1.08297, askhigh=1.0
      8299, asklow=1.08022, last=0.0, lasthigh=0.0, lastlow=0.0, volume_real=0.0, volumehigh_real=0.0, volumelow_real=0.0, option_strike=0.0, point=1e-05, tr
      ade_tick_value=1.0, trade_tick_value_profit=1.0, trade_tick_value_loss=1.0, trade_tick_size=1e-05, trade_contract_size=100000.0, trade_accrued_interest
      =0.0, trade_face_value=0.0, trade_liquidity_rate=0.0, volume_min=0.01, volume_max=200.0, volume_step=0.01, volume_limit=0.0, swap_long=-6.25, swap_shor
      t=2.56, margin_initial=100000.0, margin_maintenance=0.0, session_volume=0.0, session_turnover=0.0, session_interest=0.0, session_buy_orders_volume=0.0,
      session_sell_orders_volume=0.0, session_open=1.08074, session_close=1.08081, session_aw=0.0, session_price_settlement=0.0, session_price_limit_min=0.0,
      session_price_limit_max=0.0, margin_hedged=0.0, price_change=0.1999, price_volatility=0.0, price_theoretical=0.0, price_greeks_delta=0.0, price_greeks_
      theta=0.0, price_greeks_gamma=0.0, price_greeks_vega=0.0, price_greeks_rho=0.0, price_greeks_omega=0.0, price_sensitivity=0.0, basis='', category='', c
      urrency_base='EUR', currency_profit='USD', currency_margin='EUR', bank='', description='Euro vs US Dollar', exchange='', formula='', isin='', name='EUR
      USD', page='', path='Forex\\Majors\\EURUSD'),
       SymbolInfo(custom=False, chart_mode=0, select=True, visible=True, session_deals=0, session_buy_orders=0, session_sell_orders=0, volume=0, volumehigh=
      0, volumelow=0, time=1711379076, digits=5, spread=0, spread_float=True, ticks_bookdepth=10, trade_calc_mode=0, trade_mode=4, start_time=0, expiration_t
      ime=0, trade_stops_level=0, trade_freeze_level=0, trade_exemode=2, swap_mode=1, swap_rollover3days=3, margin_hedged_use_leg=False, expiration_mode=15,
      filling_mode=2, order_mode=127, order_gtc_mode=0, option_mode=0, option_right=0, bid=1.26418, bidhigh=1.26428, bidlow=1.25899, ask=1.26418, askhigh=1.2
      6428, asklow=1.2591999999999999, last=0.0, lasthigh=0.0, lastlow=0.0, volume_real=0.0, volumehigh_real=0.0, volumelow_real=0.0, option_strike=0.0, poin
```

```
[42]: mt.symbol_info('TSLA.NAS')._asdict()
```

```
[42]: {'custom': False,
       'chart_mode': 0,
       'select': True,
       'visible': True,
       'session_deals': 0,
       'session_buy_orders': 0,
       'session_sell_orders': 0,
       'volume': 0,
```

IMPLEMENTATION

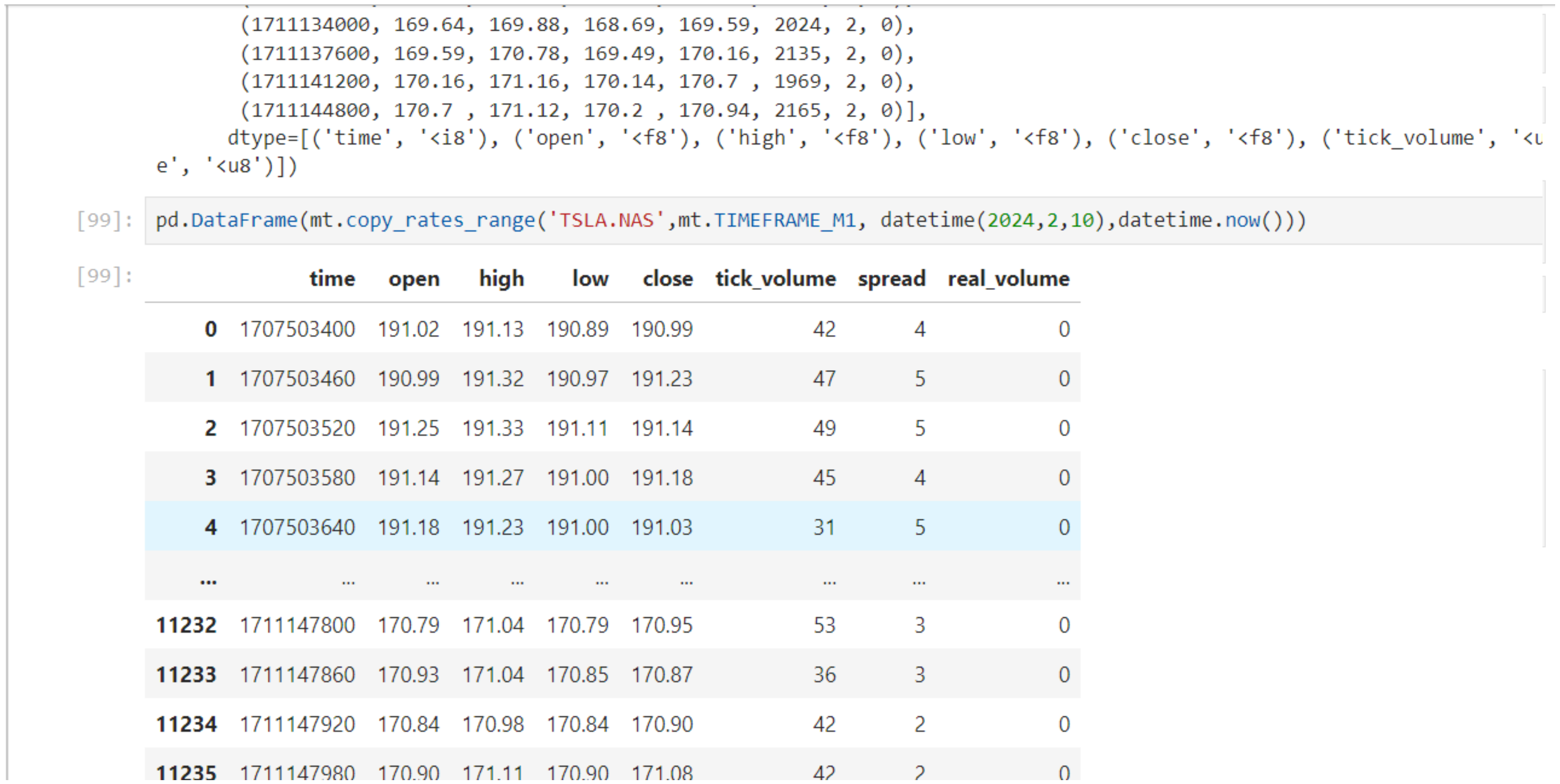


```
 'bank': '',
 'description': 'Tesla Motors CFD',
 'exchange': '',
 'formula': '',
 'isin': 'US88160R1014',
 'name': 'TSLA.NAS',
 'page': '',
 'path': "Stock CFD's\\Nasdaq\\TSLA.NAS"}

[97]: import pandas as pd

[98]: mt.copy_rates_range('TSLA.NAS',mt.TIMEFRAME_H1, datetime(2024,3,10),datetime.now())

[98]: array([(1710172800, 176.98, 182.31, 176.64, 181.73, 1956, 2, 0),
              (1710176400, 181.76, 182.83, 178.11, 178.76, 3145, 2, 0),
              (1710180000, 178.76, 178.97, 177.09, 178.16, 2518, 2, 0),
              (1710183600, 178.16, 179.03, 177.59, 178.79, 2256, 2, 0),
              (1710187200, 178.79, 179.36, 177.88, 179.24, 2034, 2, 0),
              (1710190800, 179.24, 179.53, 178.44, 178.65, 1859, 2, 0),
              (1710194400, 178.61, 178.62, 176.5 , 177.09, 2276, 2, 0),
              (1710259200, 175.17, 175.21, 172.39, 173.19, 1738, 2, 0),
              (1710262800, 173.19, 177.3 , 172.61, 176.96, 3121, 2, 0),
              (1710266400, 176.96, 178.74, 176.54, 178.18, 2751, 2, 0),
              (1710270000, 178.18, 179.04, 176.73, 177.25, 2400, 2, 0),
              (1710273600, 177.25, 178.45, 176.77, 178.42, 2055, 3, 0),
              (1710277200, 178.42, 179.38, 178.07, 178.91, 2064, 3, 0),
              (1710280800, 178.89, 179.07, 177.38, 177.56, 1915, 2, 0),
              (1710345600, 175.07, 175.14, 172.97, 173.5 , 1602, 3, 0),
              (1710349200, 173.53, 173.57, 170.58, 170.76, 3191, 3, 0),
              (1710352800, 170.76, 173.09, 170.57, 172.08, 2494, 3, 0),
              (1710356400, 172.08, 172.38, 170.83, 170.91, 1968, 3, 0),
              (1710360000, 170.91, 172.02, 170.68, 170.84, 1928, 2, 0),
              (1710363600, 170.84, 171.15, 170.03, 170.18, 1924, 3, 0),
```

AMC ENGINEERING COLLEGE

Department of Information Science and Engineering

IMPLEMENTATION



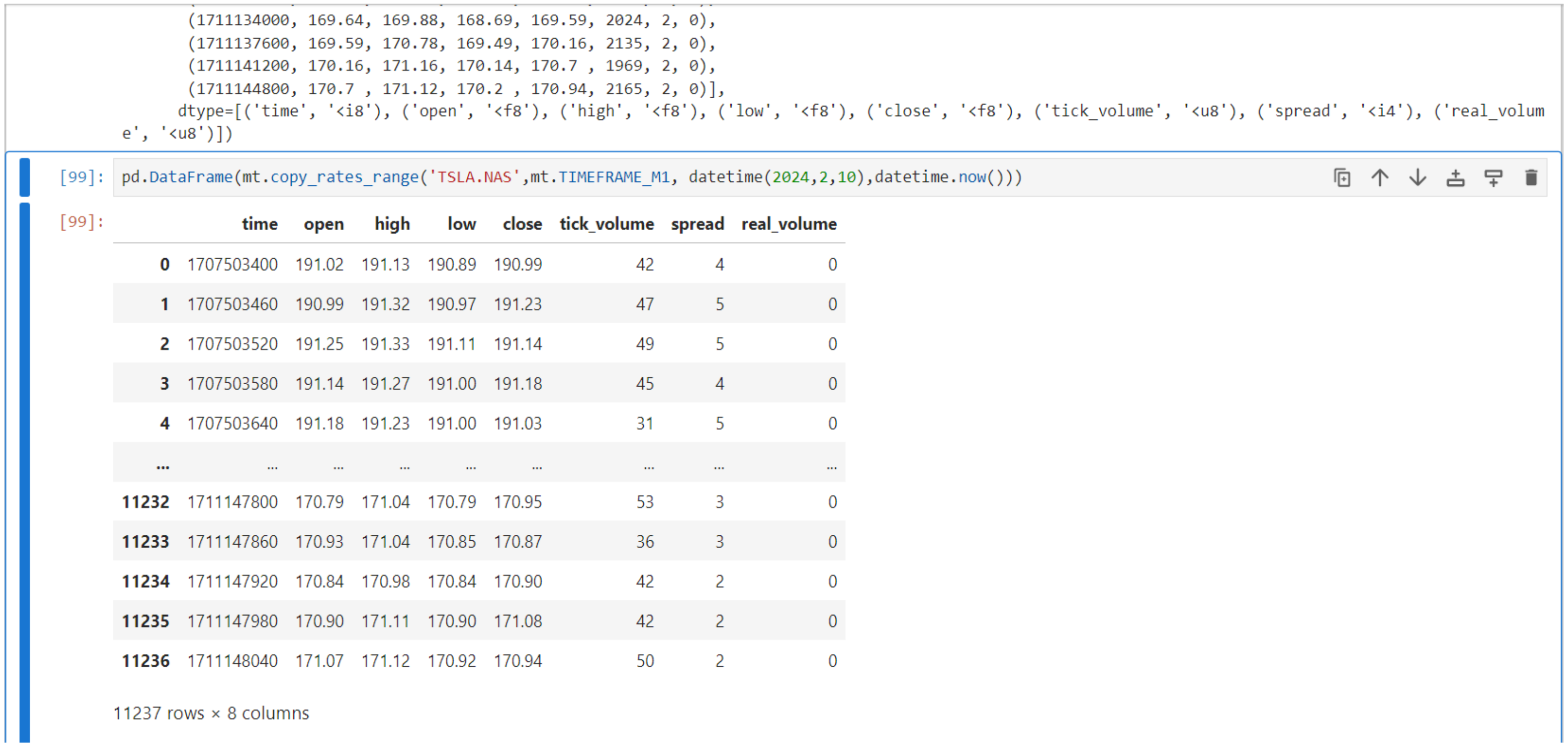
```
        (1711134000, 169.64, 169.88, 168.69, 169.59, 2024, 2, 0),
        (1711137600, 169.59, 170.78, 169.49, 170.16, 2135, 2, 0),
        (1711141200, 170.16, 171.16, 170.14, 170.7 , 1969, 2, 0),
        (1711144800, 170.7 , 171.12, 170.2 , 170.94, 2165, 2, 0)],
       dtype=[('time', '<i8'), ('open', '<f8'), ('high', '<f8'), ('low', '<f8'), ('close', '<f8'), ('tick_volume', '<u
e', '<u8')])
```

```
[99]: pd.DataFrame(mt.copy_rates_range('TSLA.NAS',mt.TIMEFRAME_M1, datetime(2024,2,10),datetime.now()))
```

[99]:

	time	open	high	low	close	tick_volume	spread	real_volume
0	1707503400	191.02	191.13	190.89	190.99	42	4	0
1	1707503460	190.99	191.32	190.97	191.23	47	5	0
2	1707503520	191.25	191.33	191.11	191.14	49	5	0
3	1707503580	191.14	191.27	191.00	191.18	45	4	0
4	1707503640	191.18	191.23	191.00	191.03	31	5	0
...	...	...	...	...	...	...	...	...
11232	1711147800	170.79	171.04	170.79	170.95	53	3	0
11233	1711147860	170.93	171.04	170.85	170.87	36	3	0
11234	1711147920	170.84	170.98	170.84	170.90	42	2	0
11235	1711147980	170.90	171.11	170.90	171.08	42	2	0

IMPLEMENTATION



```
        (1711134000, 169.64, 169.88, 168.69, 169.59, 2024, 2, 0),
        (1711137600, 169.59, 170.78, 169.49, 170.16, 2135, 2, 0),
        (1711141200, 170.16, 171.16, 170.14, 170.7 , 1969, 2, 0),
        (1711144800, 170.7 , 171.12, 170.2 , 170.94, 2165, 2, 0)],
       dtype=[('time', '<i8'), ('open', '<f8'), ('high', '<f8'), ('low', '<f8'), ('close', '<f8'), ('tick_volume', '<u8'), ('spread', '<i4'), ('real_volum
e', '<u8')])
```

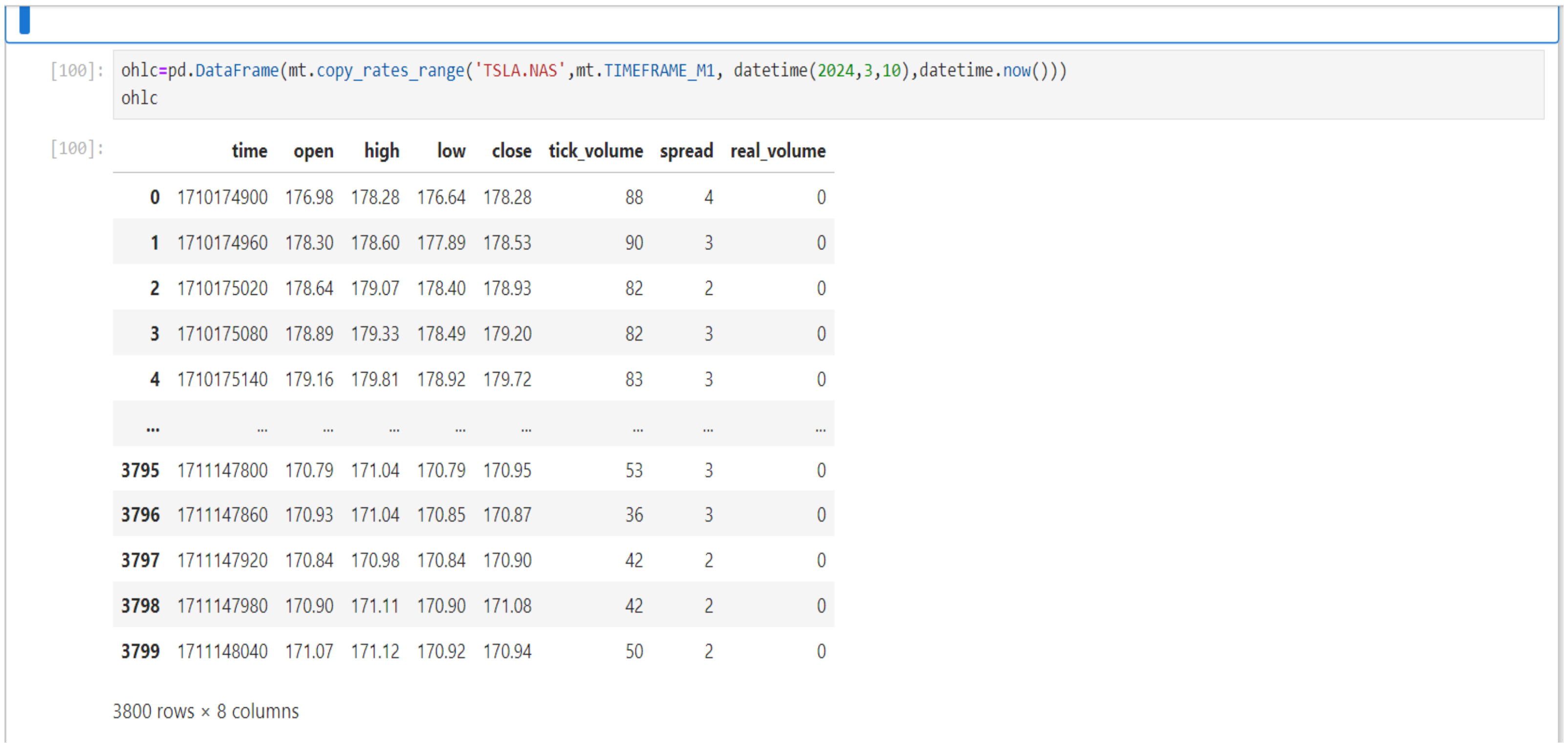
```
[99]: pd.DataFrame(mt.copy_rates_range('TSLA.NAS',mt.TIMEFRAME_M1, datetime(2024,2,10),datetime.now()))
```

[99]:

	time	open	high	low	close	tick_volume	spread	real_volume
0	1707503400	191.02	191.13	190.89	190.99	42	4	0
1	1707503460	190.99	191.32	190.97	191.23	47	5	0
2	1707503520	191.25	191.33	191.11	191.14	49	5	0
3	1707503580	191.14	191.27	191.00	191.18	45	4	0
4	1707503640	191.18	191.23	191.00	191.03	31	5	0
...	...	...	...	...	...	...	...	...
11232	1711147800	170.79	171.04	170.79	170.95	53	3	0
11233	1711147860	170.93	171.04	170.85	170.87	36	3	0
11234	1711147920	170.84	170.98	170.84	170.90	42	2	0
11235	1711147980	170.90	171.11	170.90	171.08	42	2	0
11236	1711148040	171.07	171.12	170.92	170.94	50	2	0

11237 rows × 8 columns

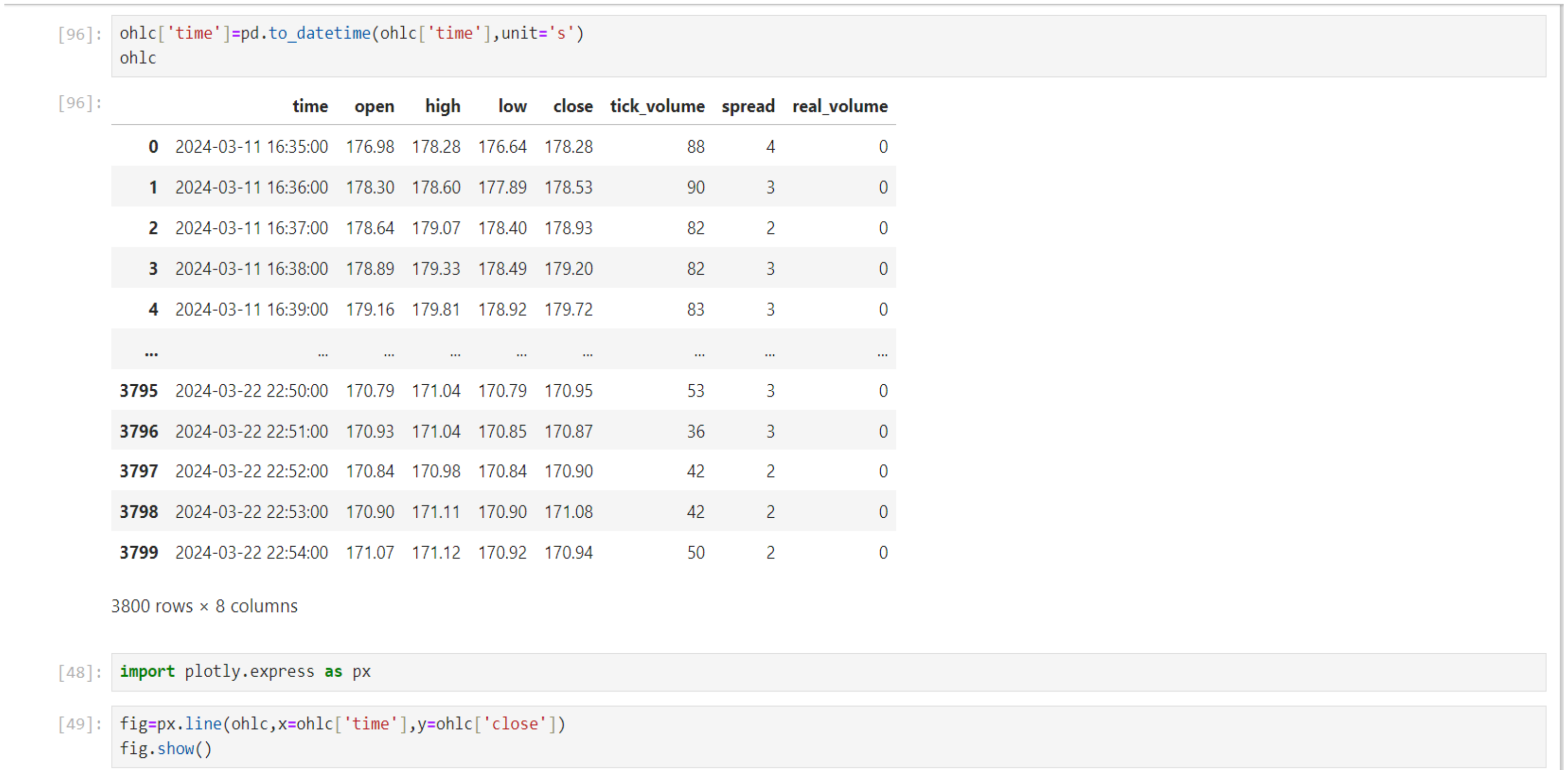
IMPLEMENTATION



```
[100]: ohlc=pd.DataFrame(mt.copy_rates_range('TSLA.NAS',mt.TIMEFRAME_M1, datetime(2024,3,10),datetime.now()))
       ohlc
```

```
[100]:
                time    open    high     low   close  tick_volume  spread  real_volume
          0  1710174900  176.98  178.28  176.64  178.28           88       4            0
          1  1710174960  178.30  178.60  177.89  178.53           90       3            0
          2  1710175020  178.64  179.07  178.40  178.93           82       2            0
          3  1710175080  178.89  179.33  178.49  179.20           82       3            0
          4  1710175140  179.16  179.81  178.92  179.72           83       3            0
        ...         ...     ...     ...     ...     ...          ...     ...          ...
       3795  1711147800  170.79  171.04  170.79  170.95           53       3            0
       3796  1711147860  170.93  171.04  170.85  170.87           36       3            0
       3797  1711147920  170.84  170.98  170.84  170.90           42       2            0
       3798  1711147980  170.90  171.11  170.90  171.08           42       2            0
       3799  1711148040  171.07  171.12  170.92  170.94           50       2            0

       3800 rows × 8 columns
```

IMPLEMENTATION



```
[96]: ohlc['time']=pd.to_datetime(ohlc['time'],unit='s')
      ohlc
```

[96]:

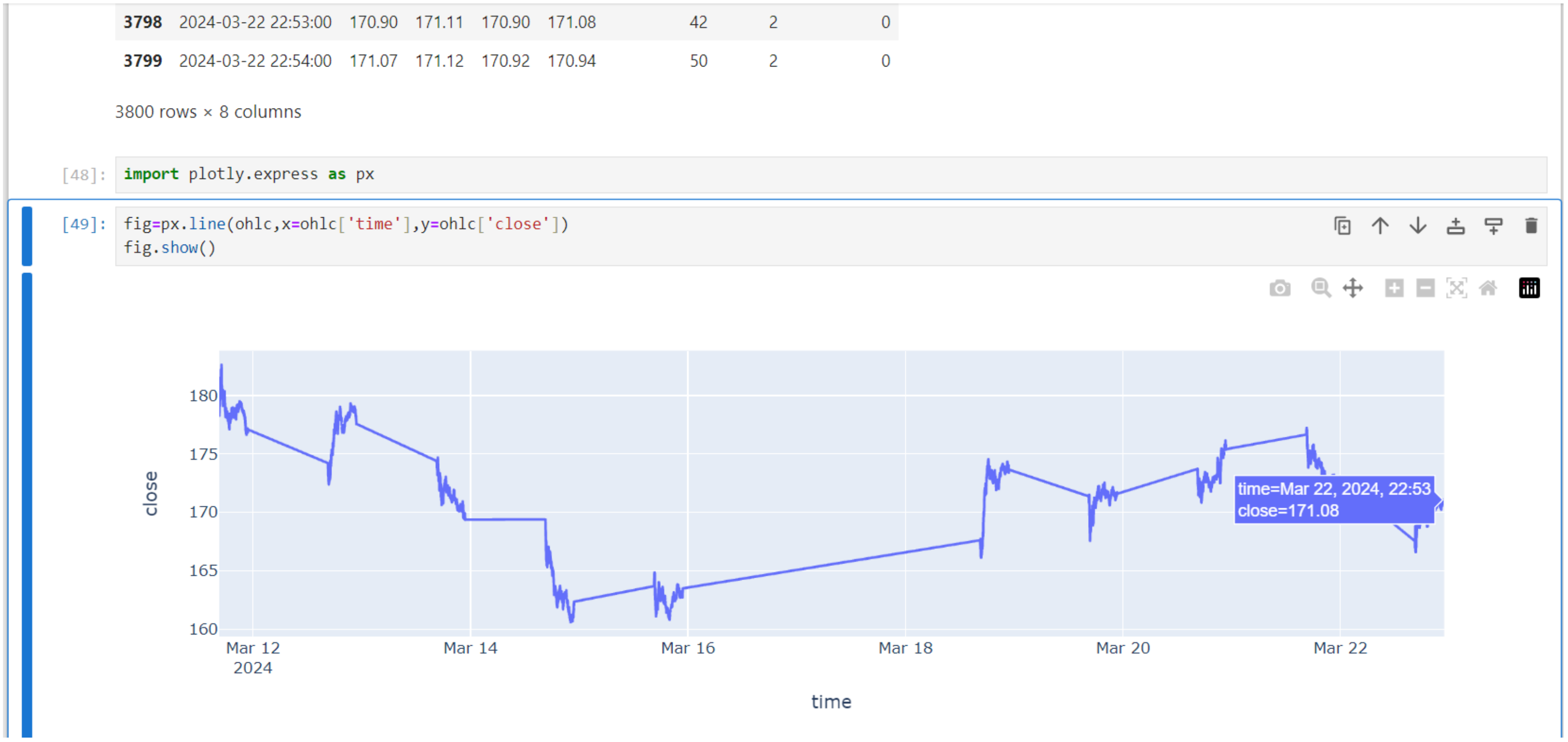
	time	open	high	low	close	tick_volume	spread	real_volume
0	2024-03-11 16:35:00	176.98	178.28	176.64	178.28	88	4	0
1	2024-03-11 16:36:00	178.30	178.60	177.89	178.53	90	3	0
2	2024-03-11 16:37:00	178.64	179.07	178.40	178.93	82	2	0
3	2024-03-11 16:38:00	178.89	179.33	178.49	179.20	82	3	0
4	2024-03-11 16:39:00	179.16	179.81	178.92	179.72	83	3	0
...	...	...	...	...	...	...	...	...
3795	2024-03-22 22:50:00	170.79	171.04	170.79	170.95	53	3	0
3796	2024-03-22 22:51:00	170.93	171.04	170.85	170.87	36	3	0
3797	2024-03-22 22:52:00	170.84	170.98	170.84	170.90	42	2	0
3798	2024-03-22 22:53:00	170.90	171.11	170.90	171.08	42	2	0
3799	2024-03-22 22:54:00	171.07	171.12	170.92	170.94	50	2	0

3800 rows × 8 columns

```
[48]: import plotly.express as px
```

```
[49]: fig=px.line(ohlc,x=ohlc['time'],y=ohlc['close'])
      fig.show()
```

IMPLEMENTATION



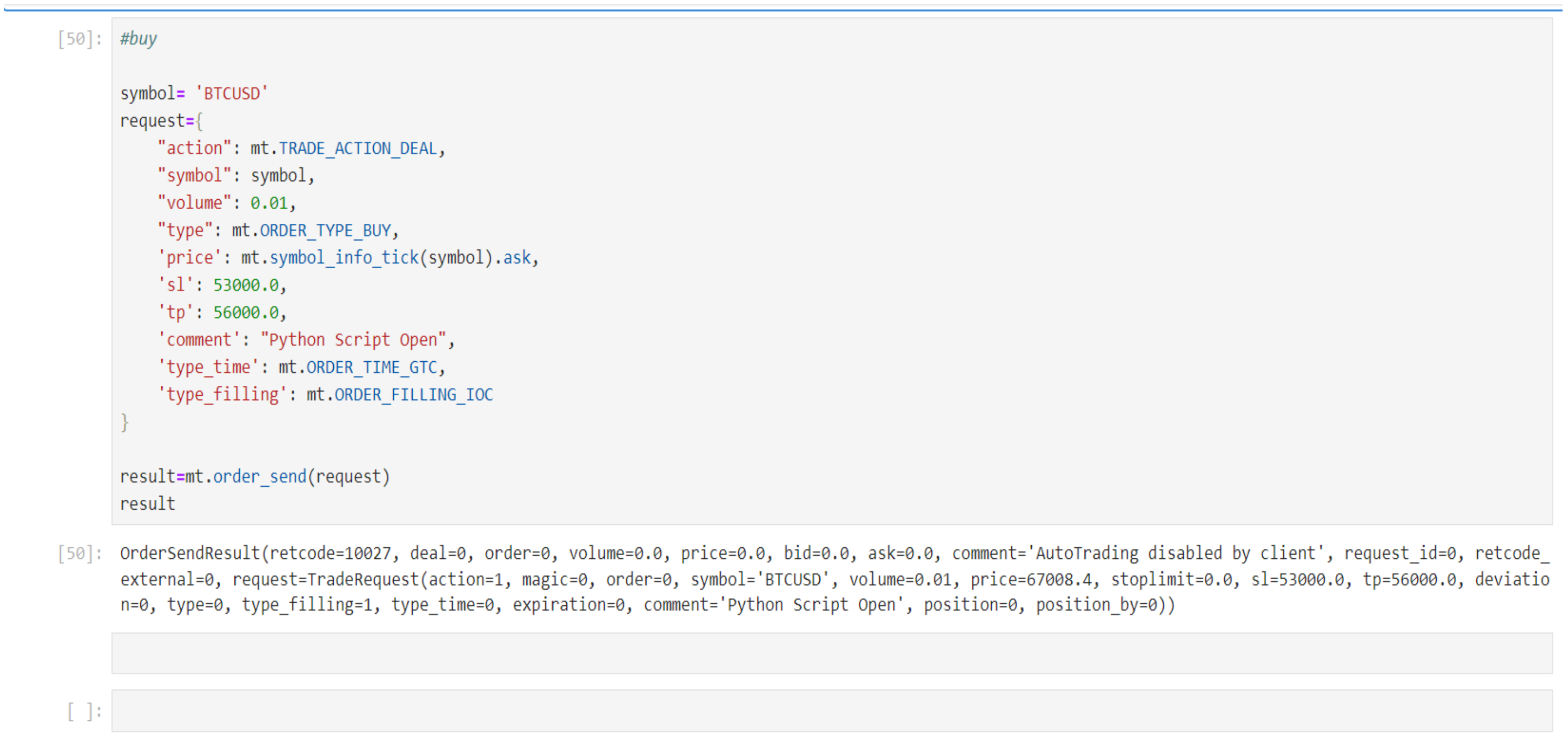
3798	2024-03-22 22:53:00	170.90	171.11	170.90	171.08	42	2	0
3799	2024-03-22 22:54:00	171.07	171.12	170.92	170.94	50	2	0

3800 rows × 8 columns

```
[48]: import plotly.express as px
```

```
[49]: fig=px.line(ohlc,x=ohlc['time'],y=ohlc['close'])
      fig.show()
```

IMPLEMENTATION

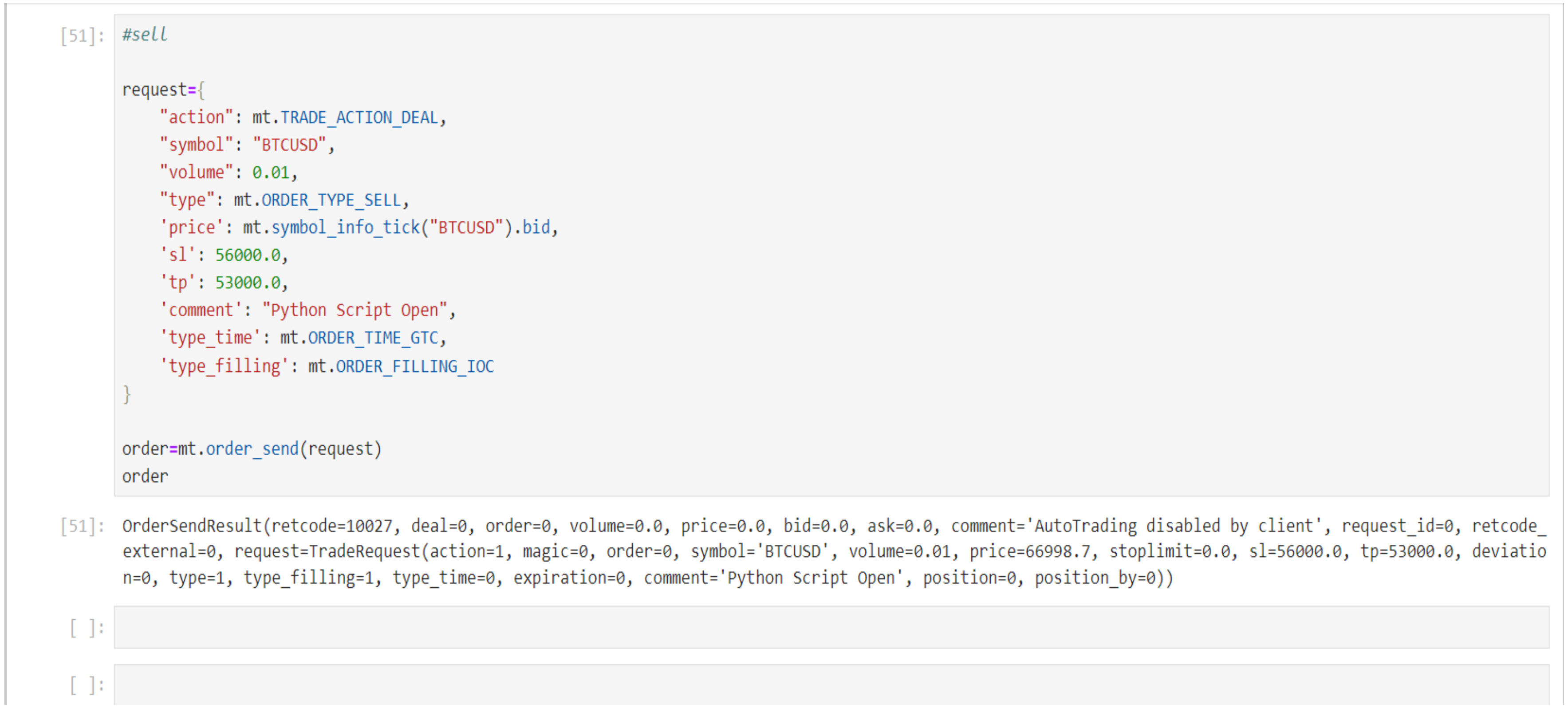


```
[50]: #buy

      symbol= 'BTCUSD'
      request={
          "action": mt.TRADE_ACTION_DEAL,
          "symbol": symbol,
          "volume": 0.01,
          "type": mt.ORDER_TYPE_BUY,
          'price': mt.symbol_info_tick(symbol).ask,
          'sl': 53000.0,
          'tp': 56000.0,
          'comment': "Python Script Open",
          'type_time': mt.ORDER_TIME_GTC,
          'type_filling': mt.ORDER_FILLING_IOC
      }

      result=mt.order_send(request)
      result
```

```
[50]: OrderSendResult(retcode=10027, deal=0, order=0, volume=0.0, price=0.0, bid=0.0, ask=0.0, comment='AutoTrading disabled by client', request_id=0, retcode_
      external=0, request=TradeRequest(action=1, magic=0, order=0, symbol='BTCUSD', volume=0.01, price=67008.4, stoplimit=0.0, sl=53000.0, tp=56000.0, deviatio
      n=0, type=0, type_filling=1, type_time=0, expiration=0, comment='Python Script Open', position=0, position_by=0))
```

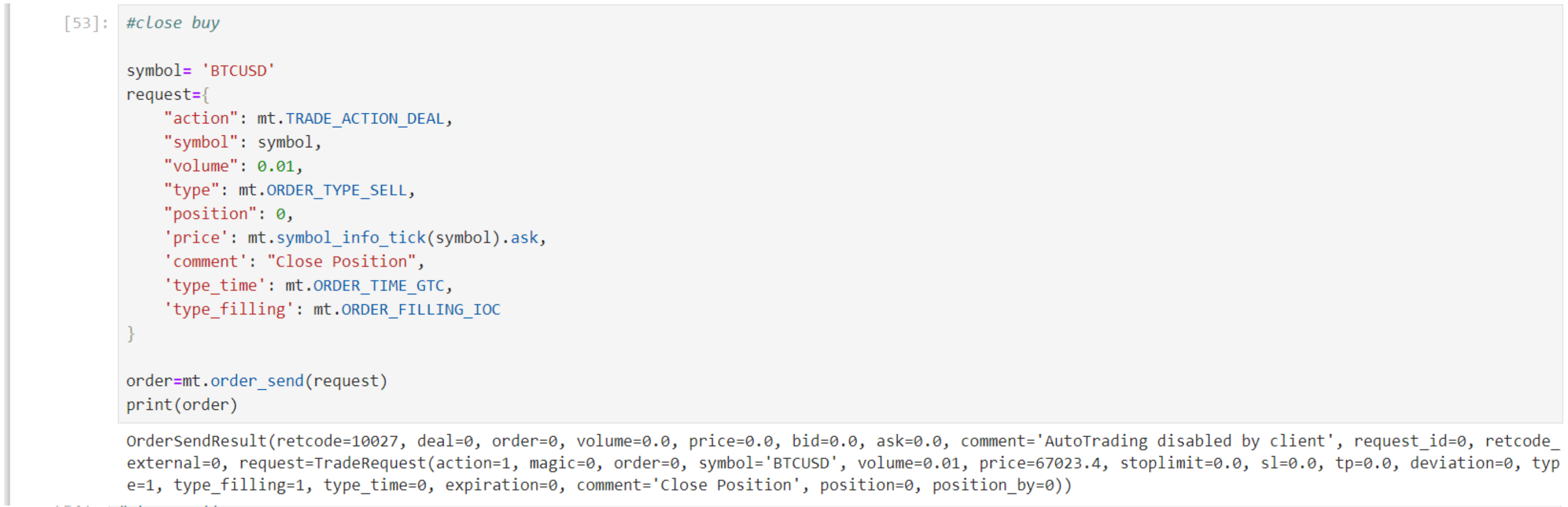
```
[ ]:
```

IMPLEMENTATION



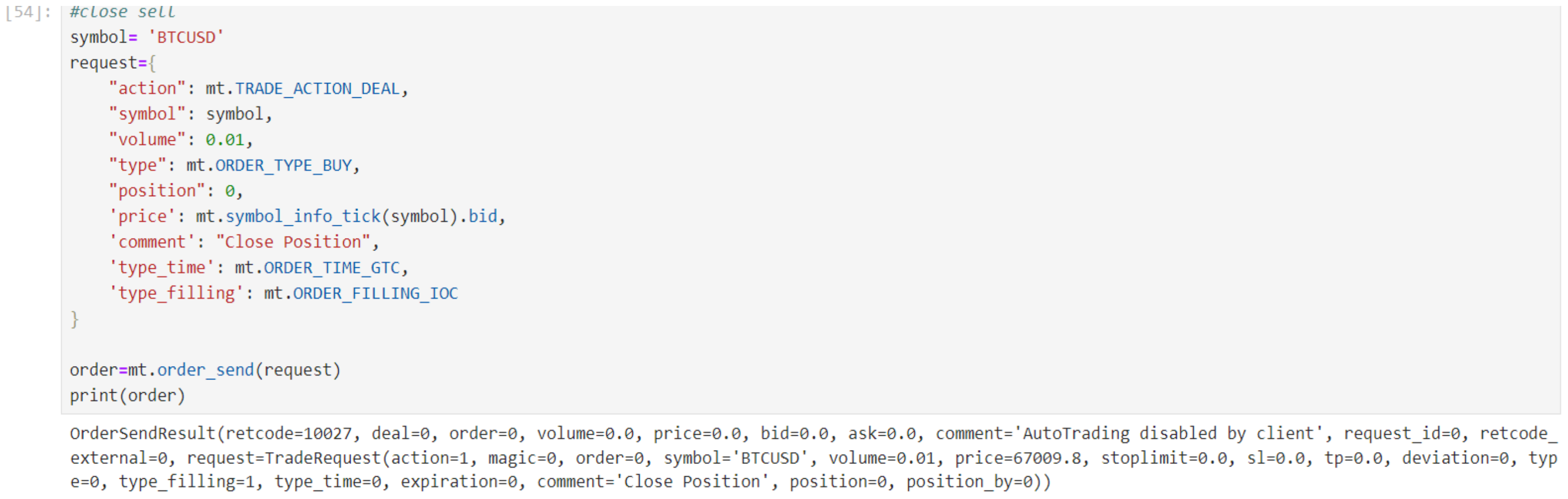
```
[51]: #sell

      request={
          "action": mt.TRADE_ACTION_DEAL,
          "symbol": "BTCUSD",
          "volume": 0.01,
          "type": mt.ORDER_TYPE_SELL,
          'price': mt.symbol_info_tick("BTCUSD").bid,
          'sl': 56000.0,
          'tp': 53000.0,
          'comment': "Python Script Open",
          'type_time': mt.ORDER_TIME_GTC,
          'type_filling': mt.ORDER_FILLING_IOC
      }

      order=mt.order_send(request)
      order

[51]: OrderSendResult(retcode=10027, deal=0, order=0, volume=0.0, price=0.0, bid=0.0, ask=0.0, comment='AutoTrading disabled by client', request_id=0, retcode_
      external=0, request=TradeRequest(action=1, magic=0, order=0, symbol='BTCUSD', volume=0.01, price=66998.7, stoplimit=0.0, sl=56000.0, tp=53000.0, deviatio
      n=0, type=1, type_filling=1, type_time=0, expiration=0, comment='Python Script Open', position=0, position_by=0))

[ ]:

[ ]:
```

IMPLEMENTATION



```
[53]: #close buy

      symbol= 'BTCUSD'
      request={
          "action": mt.TRADE_ACTION_DEAL,
          "symbol": symbol,
          "volume": 0.01,
          "type": mt.ORDER_TYPE_SELL,
          "position": 0,
          'price': mt.symbol_info_tick(symbol).ask,
          'comment': "Close Position",
          'type_time': mt.ORDER_TIME_GTC,
          'type_filling': mt.ORDER_FILLING_IOC
      }

      order=mt.order_send(request)
      print(order)
```

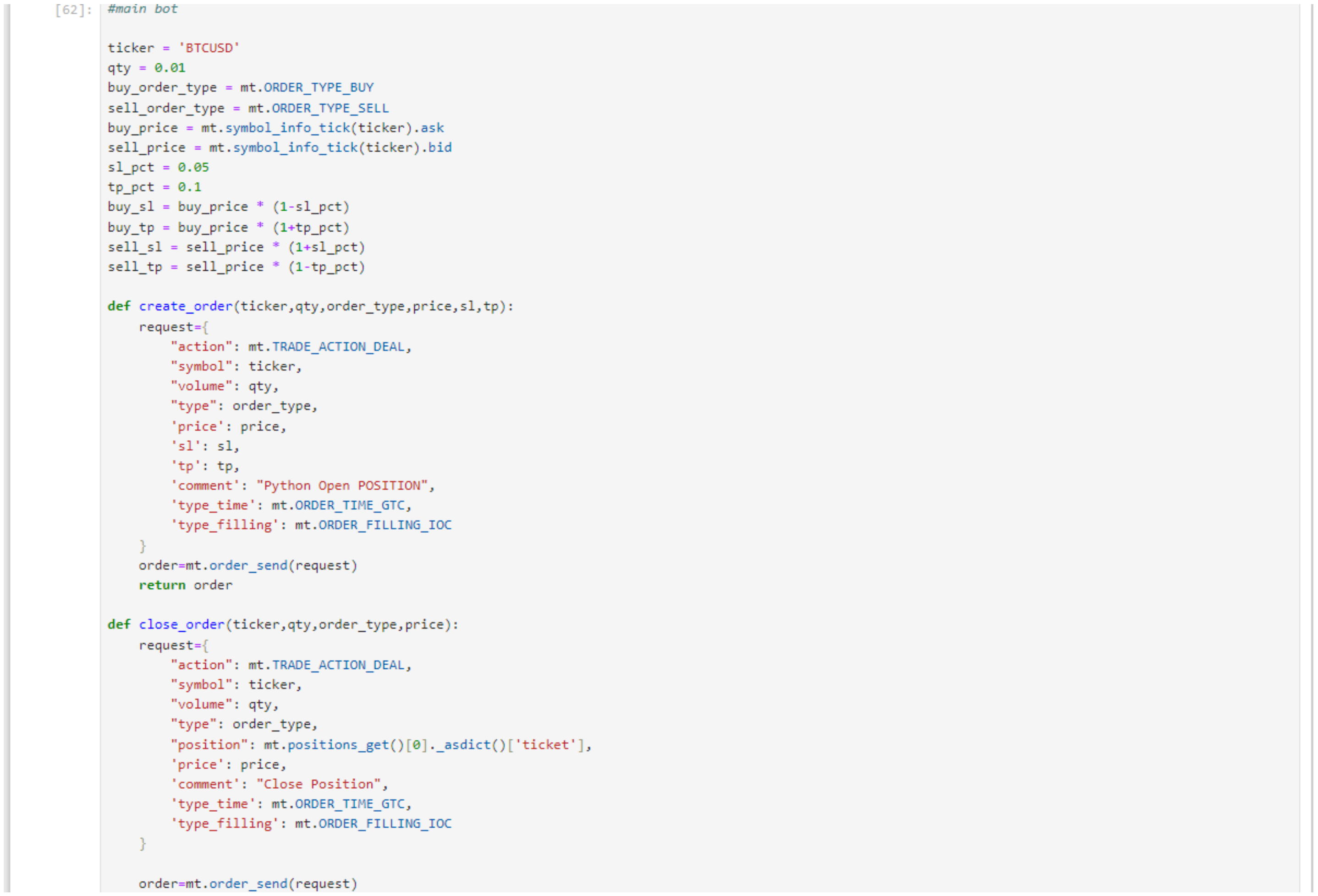
```
OrderSendResult(retcode=10027, deal=0, order=0, volume=0.0, price=0.0, bid=0.0, ask=0.0, comment='AutoTrading disabled by client', request_id=0, retcode_
external=0, request=TradeRequest(action=1, magic=0, order=0, symbol='BTCUSD', volume=0.01, price=67023.4, stoplimit=0.0, sl=0.0, tp=0.0, deviation=0, typ
e=1, type_filling=1, type_time=0, expiration=0, comment='Close Position', position=0, position_by=0))
```



```
[54]: #close sell
      symbol= 'BTCUSD'
      request={
          "action": mt.TRADE_ACTION_DEAL,
          "symbol": symbol,
          "volume": 0.01,
          "type": mt.ORDER_TYPE_BUY,
          "position": 0,
          'price': mt.symbol_info_tick(symbol).bid,
          'comment': "Close Position",
          'type_time': mt.ORDER_TIME_GTC,
          'type_filling': mt.ORDER_FILLING_IOC
      }

      order=mt.order_send(request)
      print(order)
```

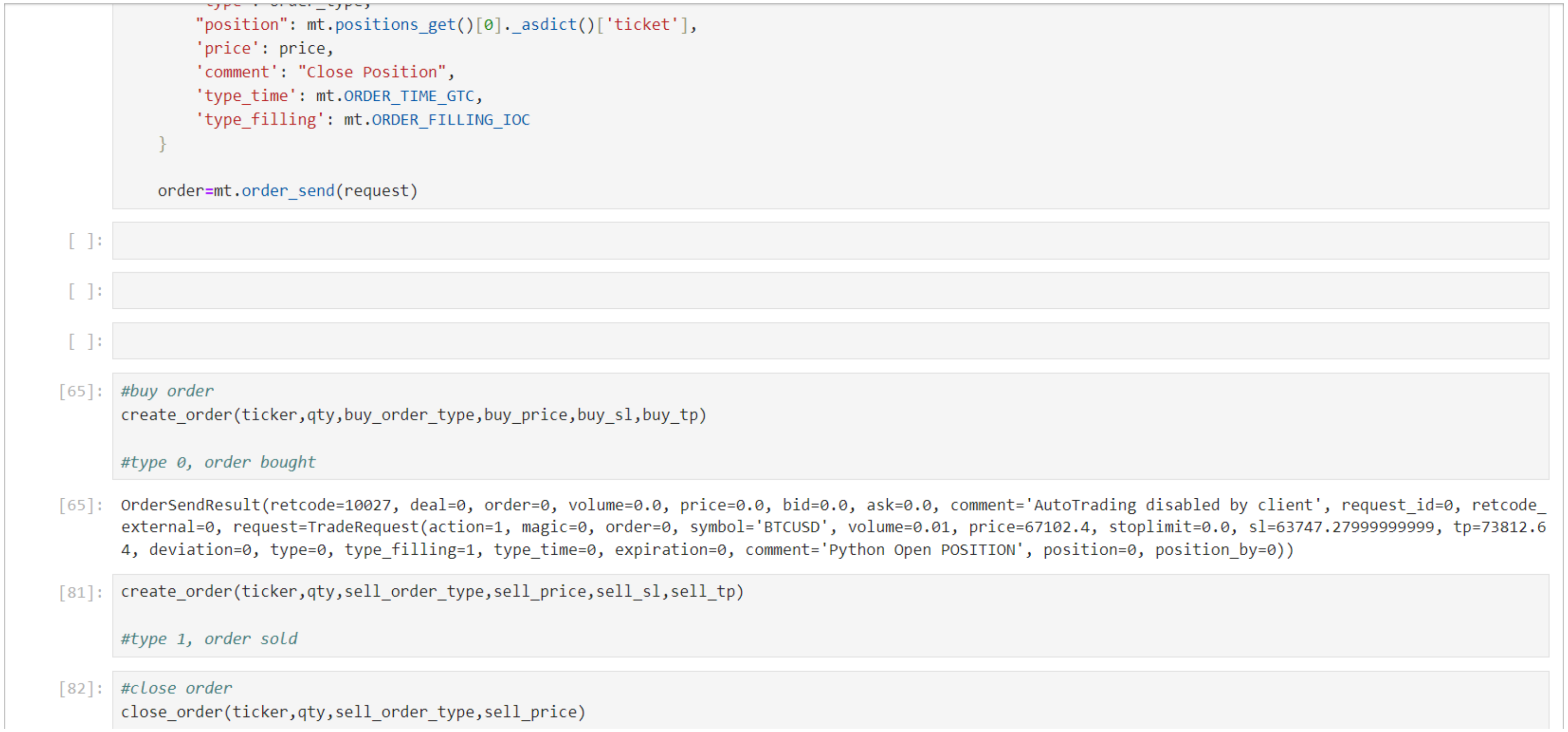
```
OrderSendResult(retcode=10027, deal=0, order=0, volume=0.0, price=0.0, bid=0.0, ask=0.0, comment='AutoTrading disabled by client', request_id=0, retcode_
external=0, request=TradeRequest(action=1, magic=0, order=0, symbol='BTCUSD', volume=0.01, price=67009.8, stoplimit=0.0, sl=0.0, tp=0.0, deviation=0, typ
e=0, type_filling=1, type_time=0, expiration=0, comment='Close Position', position=0, position_by=0))
```

IMPLEMENTATION



```
[62]: #main bot

ticker = 'BTCUSD'
qty = 0.01
buy_order_type = mt.ORDER_TYPE_BUY
sell_order_type = mt.ORDER_TYPE_SELL
buy_price = mt.symbol_info_tick(ticker).ask
sell_price = mt.symbol_info_tick(ticker).bid
sl_pct = 0.05
tp_pct = 0.1
buy_sl = buy_price * (1-sl_pct)
buy_tp = buy_price * (1+tp_pct)
sell_sl = sell_price * (1+sl_pct)
sell_tp = sell_price * (1-tp_pct)

def create_order(ticker,qty,order_type,price,sl,tp):
    request={
        "action": mt.TRADE_ACTION_DEAL,
        "symbol": ticker,
        "volume": qty,
        "type": order_type,
        'price': price,
        'sl': sl,
        'tp': tp,
        'comment': "Python Open POSITION",
        'type_time': mt.ORDER_TIME_GTC,
        'type_filling': mt.ORDER_FILLING_IOC
    }
    order=mt.order_send(request)
    return order

def close_order(ticker,qty,order_type,price):
    request={
        "action": mt.TRADE_ACTION_DEAL,
        "symbol": ticker,
        "volume": qty,
        "type": order_type,
        "position": mt.positions_get()[0]._asdict()['ticket'],
        'price': price,
        'comment': "Close Position",
        'type_time': mt.ORDER_TIME_GTC,
        'type_filling': mt.ORDER_FILLING_IOC
    }

    order=mt.order_send(request)
```

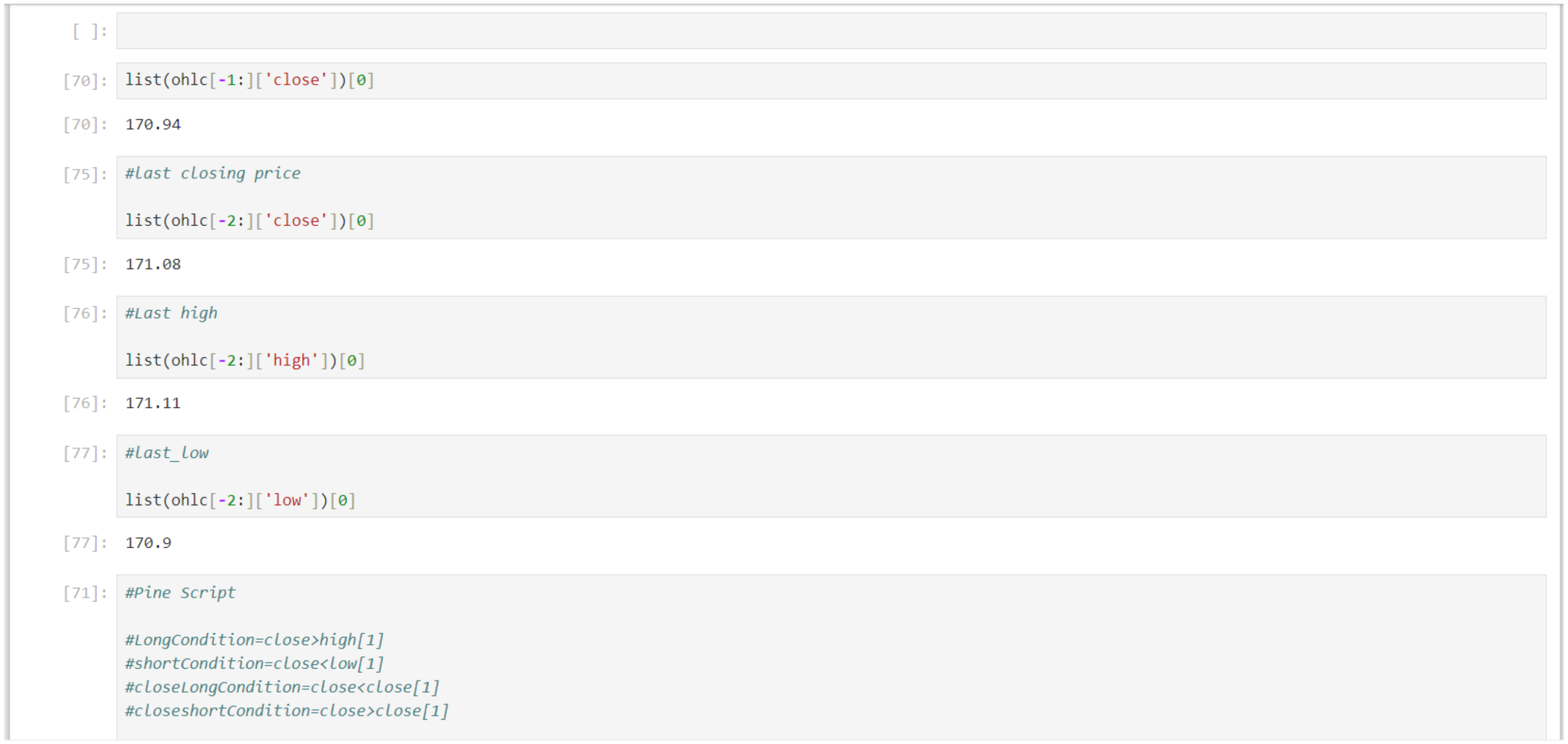
IMPLEMENTATION



```
        "position": mt.positions_get()[0]._asdict()['ticket'],
        'price': price,
        'comment': "Close Position",
        'type_time': mt.ORDER_TIME_GTC,
        'type_filling': mt.ORDER_FILLING_IOC
    }

    order=mt.order_send(request)
```

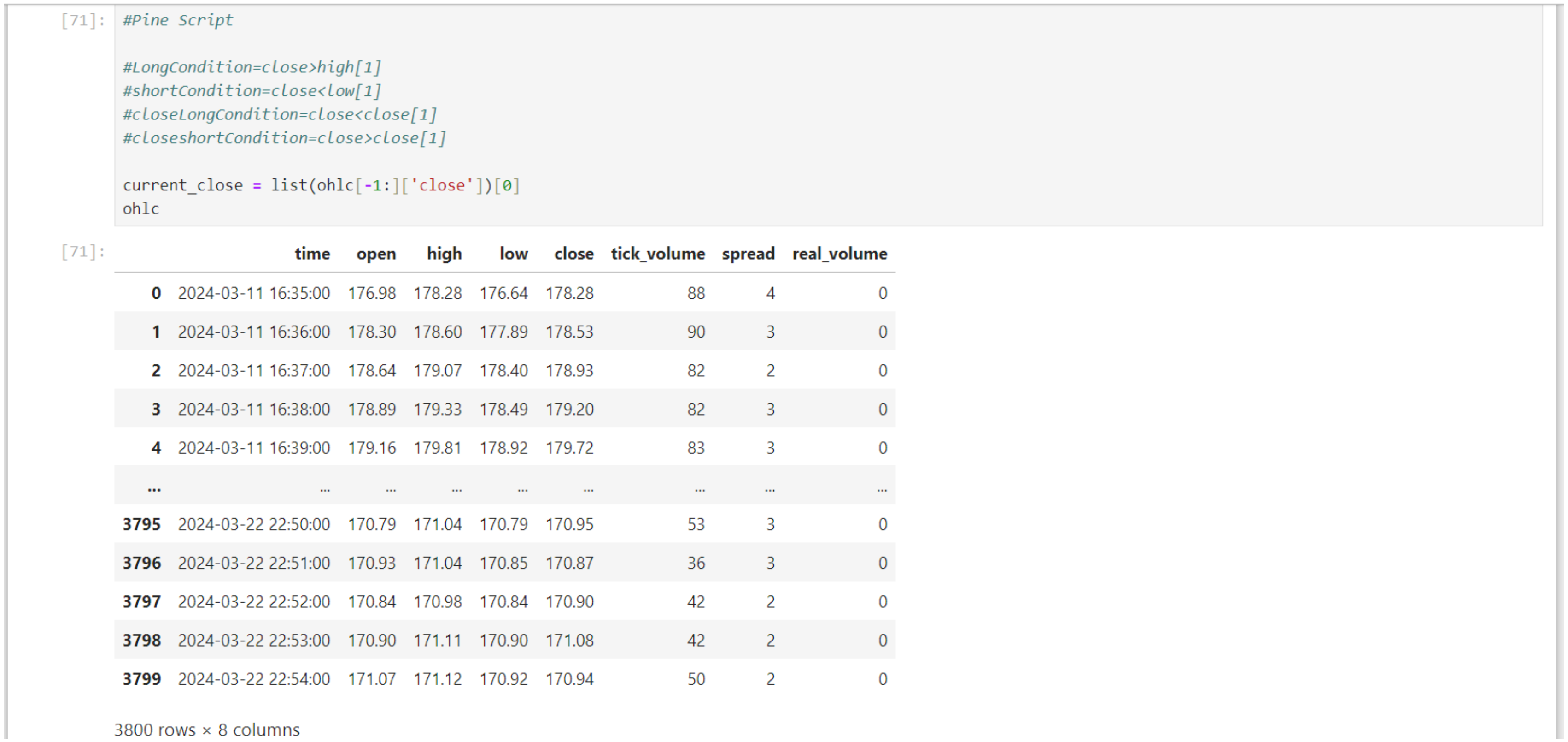
```
[ ]:

[ ]:

[ ]:

[65]: #buy order
      create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)

      #type 0, order bought

[65]: OrderSendResult(retcode=10027, deal=0, order=0, volume=0.0, price=0.0, bid=0.0, ask=0.0, comment='AutoTrading disabled by client', request_id=0, retcode_
      external=0, request=TradeRequest(action=1, magic=0, order=0, symbol='BTCUSD', volume=0.01, price=67102.4, stoplimit=0.0, sl=63747.27999999999, tp=73812.6
      4, deviation=0, type=0, type_filling=1, type_time=0, expiration=0, comment='Python Open POSITION', position=0, position_by=0))

[81]: create_order(ticker,qty,sell_order_type,sell_price,sell_sl,sell_tp)

      #type 1, order sold

[82]: #close order
      close_order(ticker,qty,sell_order_type,sell_price)
```

IMPLEMENTATION



```
[ ]:

[70]: list(ohlc[-1:]['close'])[0]

[70]: 170.94

[75]: #last closing price

      list(ohlc[-2:]['close'])[0]

[75]: 171.08

[76]: #Last high

      list(ohlc[-2:]['high'])[0]

[76]: 171.11

[77]: #last_low

      list(ohlc[-2:]['low'])[0]

[77]: 170.9

[71]: #Pine Script

      #LongCondition=close>high[1]
      #shortCondition=close<low[1]
      #closeLongCondition=close<close[1]
      #closeshortCondition=close>close[1]
```

IMPLEMENTATION



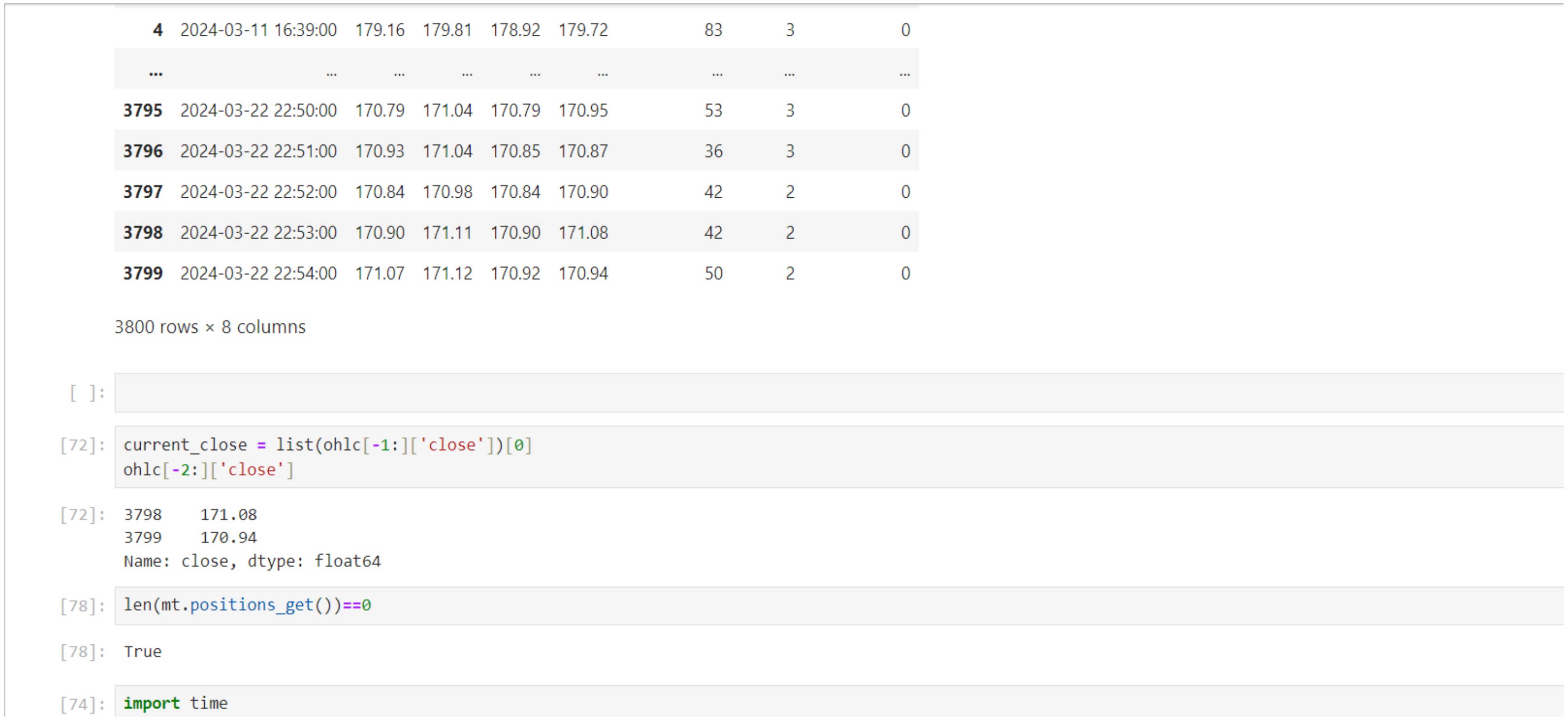
```
[71]: #Pine Script

      #LongCondition=close>high[1]
      #shortCondition=close<low[1]
      #closeLongCondition=close<close[1]
      #closeshortCondition=close>close[1]

      current_close = list(ohlc[-1:]['close'])[0]
      ohlc
```

[71]:

	time	open	high	low	close	tick_volume	spread	real_volume
0	2024-03-11 16:35:00	176.98	178.28	176.64	178.28	88	4	0
1	2024-03-11 16:36:00	178.30	178.60	177.89	178.53	90	3	0
2	2024-03-11 16:37:00	178.64	179.07	178.40	178.93	82	2	0
3	2024-03-11 16:38:00	178.89	179.33	178.49	179.20	82	3	0
4	2024-03-11 16:39:00	179.16	179.81	178.92	179.72	83	3	0
...	...	...	...	...	...	...	...	...
3795	2024-03-22 22:50:00	170.79	171.04	170.79	170.95	53	3	0
3796	2024-03-22 22:51:00	170.93	171.04	170.85	170.87	36	3	0
3797	2024-03-22 22:52:00	170.84	170.98	170.84	170.90	42	2	0
3798	2024-03-22 22:53:00	170.90	171.11	170.90	171.08	42	2	0
3799	2024-03-22 22:54:00	171.07	171.12	170.92	170.94	50	2	0

3800 rows × 8 columns

IMPLEMENTATION

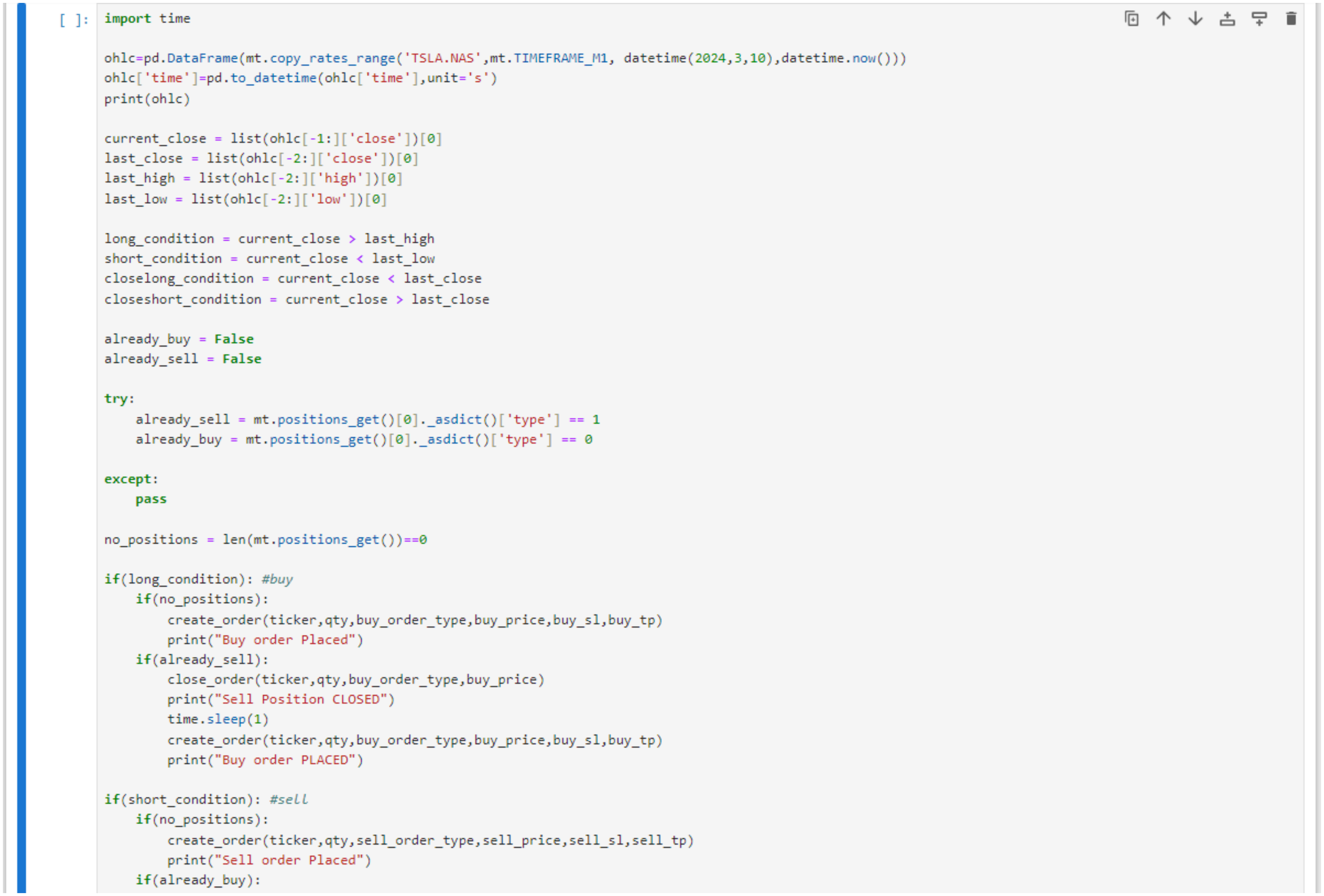


4	2024-03-11 16:39:00	179.16	179.81	178.92	179.72	83	3	0
...	...	...	...	...	...	...	...	...
3795	2024-03-22 22:50:00	170.79	171.04	170.79	170.95	53	3	0
3796	2024-03-22 22:51:00	170.93	171.04	170.85	170.87	36	3	0
3797	2024-03-22 22:52:00	170.84	170.98	170.84	170.90	42	2	0
3798	2024-03-22 22:53:00	170.90	171.11	170.90	171.08	42	2	0
3799	2024-03-22 22:54:00	171.07	171.12	170.92	170.94	50	2	0

3800 rows × 8 columns

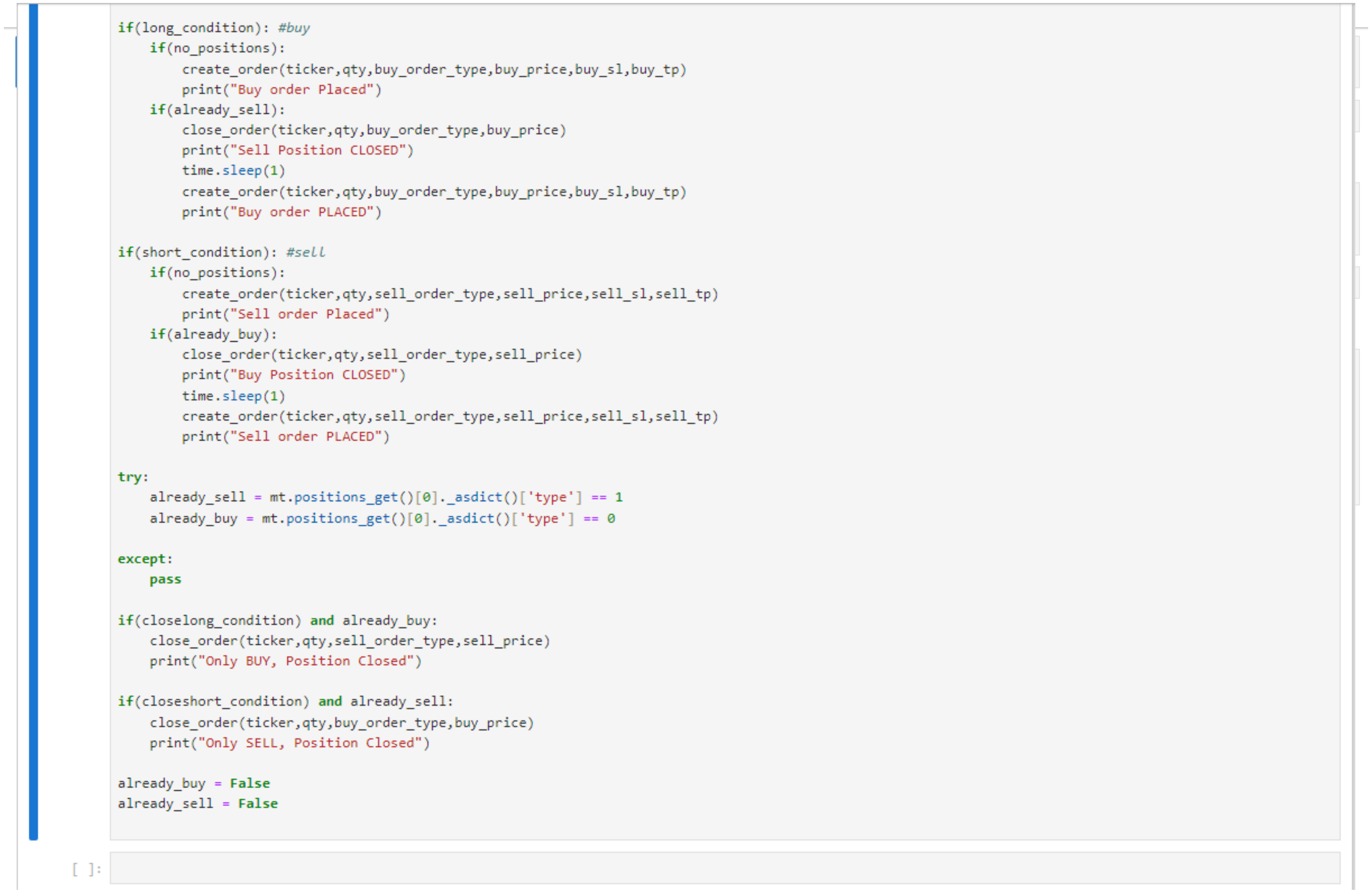
```
[ ]:

[72]: current_close = list(ohlc[-1:]['close'])[0]
      ohlc[-2:]['close']

[72]: 3798    171.08
      3799    170.94
      Name: close, dtype: float64

[78]: len(mt.positions_get())==0

[78]: True

[74]: import time
```

IMPLEMENTATION



```
[ ]: import time

     ohlc=pd.DataFrame(mt.copy_rates_range('TSLA.NAS',mt.TIMEFRAME_M1, datetime(2024,3,10),datetime.now()))
     ohlc['time']=pd.to_datetime(ohlc['time'],unit='s')
     print(ohlc)

     current_close = list(ohlc[-1:]['close'])[0]
     last_close = list(ohlc[-2:]['close'])[0]
     last_high = list(ohlc[-2:]['high'])[0]
     last_low = list(ohlc[-2:]['low'])[0]

     long_condition = current_close > last_high
     short_condition = current_close < last_low
     closelong_condition = current_close < last_close
     closeshort_condition = current_close > last_close

     already_buy = False
     already_sell = False

     try:
         already_sell = mt.positions_get()[0]._asdict()['type'] == 1
         already_buy = mt.positions_get()[0]._asdict()['type'] == 0

     except:
         pass

     no_positions = len(mt.positions_get())==0

     if(long_condition): #buy
         if(no_positions):
             create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)
             print("Buy order Placed")
         if(already_sell):
             close_order(ticker,qty,buy_order_type,buy_price)
             print("Sell Position CLOSED")
             time.sleep(1)
             create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)
             print("Buy order PLACED")

     if(short_condition): #sell
         if(no_positions):
             create_order(ticker,qty,sell_order_type,sell_price,sell_sl,sell_tp)
             print("Sell order Placed")
         if(already_buy):
```

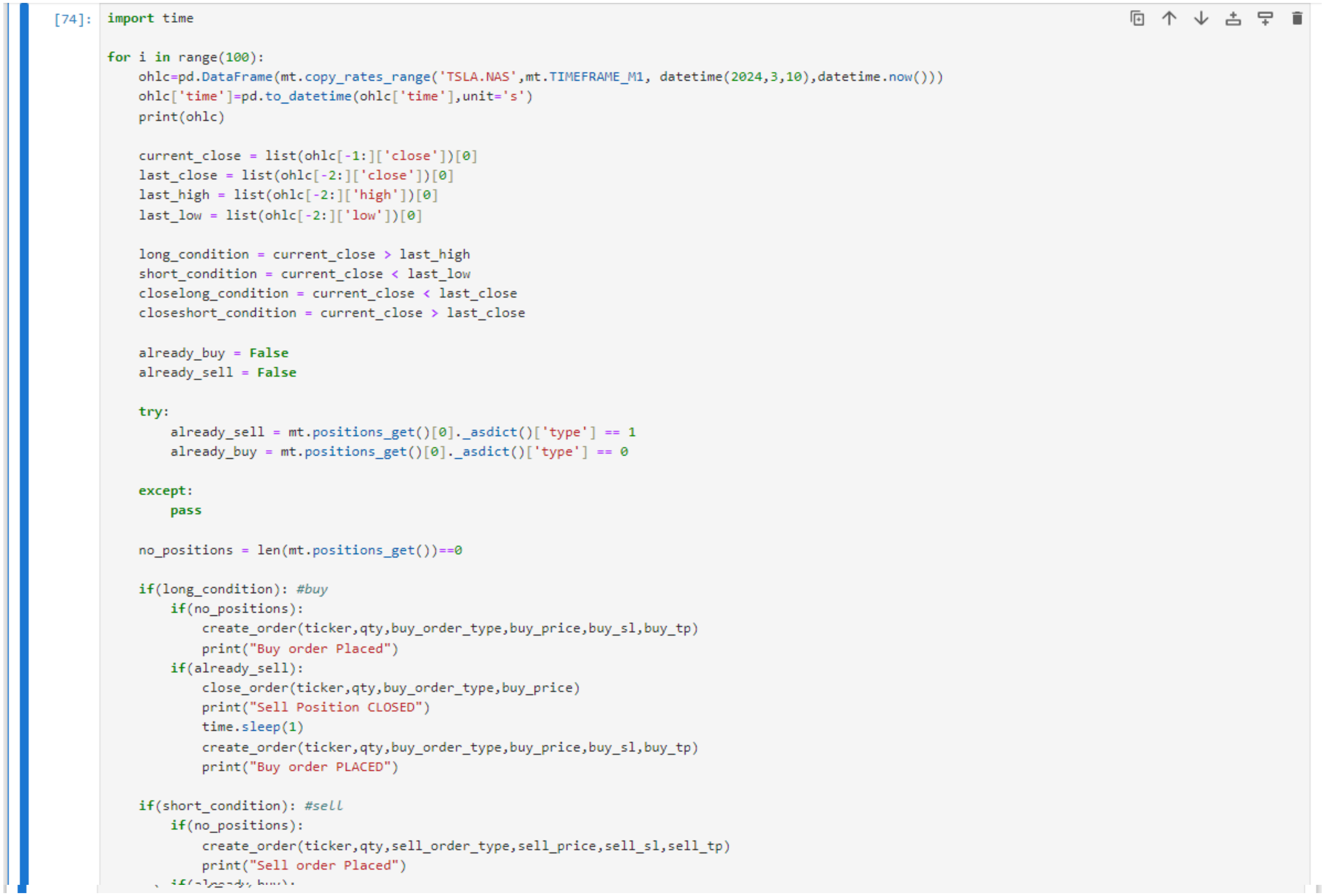
IMPLEMENTATION



```
if(long_condition): #buy
    if(no_positions):
        create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)
        print("Buy order Placed")
    if(already_sell):
        close_order(ticker,qty,buy_order_type,buy_price)
        print("Sell Position CLOSED")
        time.sleep(1)
        create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)
        print("Buy order PLACED")

if(short_condition): #sell
    if(no_positions):
        create_order(ticker,qty,sell_order_type,sell_price,sell_sl,sell_tp)
        print("Sell order Placed")
    if(already_buy):
        close_order(ticker,qty,sell_order_type,sell_price)
        print("Buy Position CLOSED")
        time.sleep(1)
        create_order(ticker,qty,sell_order_type,sell_price,sell_sl,sell_tp)
        print("Sell order PLACED")

try:
    already_sell = mt.positions_get()[0]._asdict()['type'] == 1
    already_buy = mt.positions_get()[0]._asdict()['type'] == 0

except:
    pass

if(closelong_condition) and already_buy:
    close_order(ticker,qty,sell_order_type,sell_price)
    print("Only BUY, Position Closed")

if(closeshort_condition) and already_sell:
    close_order(ticker,qty,buy_order_type,buy_price)
    print("Only SELL, Position Closed")

already_buy = False
already_sell = False
```

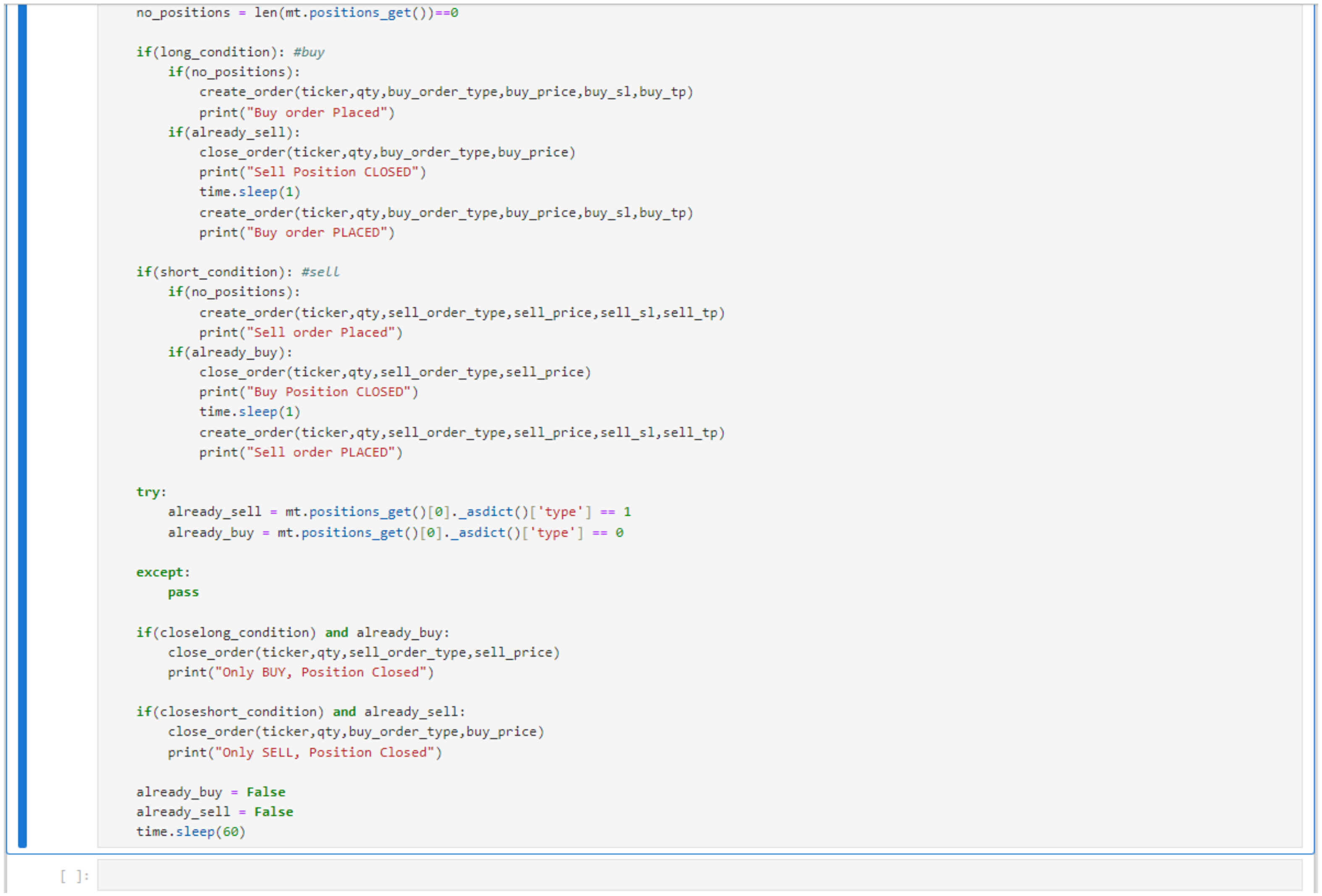
[ ]:

IMPLEMENTATION



```
[74]: import time

for i in range(100):
    ohlc=pd.DataFrame(mt.copy_rates_range('TSLA.NAS',mt.TIMEFRAME_M1, datetime(2024,3,10),datetime.now()))
    ohlc['time']=pd.to_datetime(ohlc['time'],unit='s')
    print(ohlc)

    current_close = list(ohlc[-1:]['close'])[0]
    last_close = list(ohlc[-2:]['close'])[0]
    last_high = list(ohlc[-2:]['high'])[0]
    last_low = list(ohlc[-2:]['low'])[0]

    long_condition = current_close > last_high
    short_condition = current_close < last_low
    closelong_condition = current_close < last_close
    closeshort_condition = current_close > last_close

    already_buy = False
    already_sell = False

    try:
        already_sell = mt.positions_get()[0]._asdict()['type'] == 1
        already_buy = mt.positions_get()[0]._asdict()['type'] == 0

    except:
        pass

    no_positions = len(mt.positions_get())==0

    if(long_condition): #buy
        if(no_positions):
            create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)
            print("Buy order Placed")
        if(already_sell):
            close_order(ticker,qty,buy_order_type,buy_price)
            print("Sell Position CLOSED")
            time.sleep(1)
            create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)
            print("Buy order PLACED")

    if(short_condition): #sell
        if(no_positions):
            create_order(ticker,qty,sell_order_type,sell_price,sell_sl,sell_tp)
            print("Sell order Placed")
        if(already_buy):
```

IMPLEMENTATION



```
    no_positions = len(mt.positions_get())==0

    if(long_condition): #buy
        if(no_positions):
            create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)
            print("Buy order Placed")
        if(already_sell):
            close_order(ticker,qty,buy_order_type,buy_price)
            print("Sell Position CLOSED")
            time.sleep(1)
            create_order(ticker,qty,buy_order_type,buy_price,buy_sl,buy_tp)
            print("Buy order PLACED")

    if(short_condition): #sell
        if(no_positions):
            create_order(ticker,qty,sell_order_type,sell_price,sell_sl,sell_tp)
            print("Sell order Placed")
        if(already_buy):
            close_order(ticker,qty,sell_order_type,sell_price)
            print("Buy Position CLOSED")
            time.sleep(1)
            create_order(ticker,qty,sell_order_type,sell_price,sell_sl,sell_tp)
            print("Sell order PLACED")

    try:
        already_sell = mt.positions_get()[0]._asdict()['type'] == 1
        already_buy = mt.positions_get()[0]._asdict()['type'] == 0

    except:
        pass

    if(closelong_condition) and already_buy:
        close_order(ticker,qty,sell_order_type,sell_price)
        print("Only BUY, Position Closed")

    if(closeshort_condition) and already_sell:
        close_order(ticker,qty,buy_order_type,buy_price)
        print("Only SELL, Position Closed")

    already_buy = False
    already_sell = False
    time.sleep(60)
```

[ ]:

RESULTS

The Bot successfully demonstrated the potential of artificial intelligence and automation in financial trading. By combining Long Short-Term Memory (LSTM) networks for predictive analytics with technical analysis tools like Golden Cross and Death Cross, the bot was able to generate effective trading signals and execute trades in an automated manner. The integration with MetaTrader5 (MT5) allowed for seamless trade execution and real-time market data collection. The Bot can deliver the following outcomes:

• The bot showed promising results in terms of profitability, with a consistent win rate and a favorable risk-adjusted return. The use of LSTM and traditional technical analysis provided a balanced approach to trading.

• Increased efficiency and accuracy in trading activities, potentially leading to higher returns on investment and reduced human error.

• Improved risk management through the implementation of predefined risk parameters and automated stop-loss mechanisms.

REFERENCES

Reference to a Conference publication:

- Zhang et. al., [1] “A Deep Reinforcement Learning Approach for Stock Trading (2018)”.
- Jiang et. al., [2] “An Ensemble Learning Framework for Algorithmic Trading(2019)”.
- Salah Bouktif et. al., [3] “Augmented Textual Features-Based Stock Market Prediction”
- Arya Yudhi Wijaya et. al., [4] “Stock Price Prediction with Golden Cross and Death Cross on Technical Analysis Indicators Using Long Short-Term Memory (2022)”, in Dec. 2022
- Gourav Bathla et. al., [5] “Stock Price prediction using LSTM and SVR”, at Waknaghat, India in Nov.2020
- Xiaojian Weng et. al.,[6] “Stock Price Prediction Based on LSTM And Bert”, at Japan in Sep.2022
- Sparsh Vohara et. al., [7] "Stock Price Trend Analysis and Prediction of Closing Price Using LSTM (2023)“, at Coimbatore India in Jan. 2023

Reference to Web Resources:

- GeeksforGeeks: They have a detailed project idea on an Algorithmic Trading Bot.
- GitHub: There are numerous repositories related to trading bots on GitHub.

Thank You