from \_\_future\_\_ import division

from binance.client import Client

from binance.enums import \*

import json

import websocket

import pprint

import numpy

import talib

import math

from math import log10, floor

#

#

#

#

profit = 0.01

loss = 0.015

profit\_calculation = 1+(profit/100)\*100

loss\_calculation = 1-(loss/100)\*100

SOCKET = 'wss://stream.binance.com/ws'

RSI\_PERIOD = 5

RSI\_OVERBOUGHT = 70

RSI\_OVERSOLD = 30

TRADE\_SYMBOL = 'BTCUSDT'

TRADE\_QUANTITY = 0.00027

TAKE\_PROFIT = float(profit\_calculation)

STOP\_LOSS = float(loss\_calculation)

closes = []

in\_position = False

oco = False

API\_KEY = 'bO2sXhybOw7jqCE1ur5rV2fi2KXD57b06jPQa6b3EQS6uOGVVTYBsdgCp9AP67Zv'

API\_SECRET = 'yBfYcDqHYa2nKmuZlSvgUapQVuPahnCtokRwFu2epO29FVyEL3aTA7QowCsZqsFO'

client = Client(API\_KEY, API\_SECRET)

info = client.get\_symbol\_info('TRXUSDT')

print(info)

print(info['filters'][2]['minQty'])

def method\_name():

return print("sending order")

def order(side, quantity, symbol, order\_type=ORDER\_TYPE\_MARKET):

try:

method\_name()

order = client.create\_order(symbol=symbol, side=side, type=order\_type, quantity=quantity)

print(order)

except Exception as e:

print("an exception occured - {}".format(e))

return False

def on\_open(ws):

print("opened")

subscribe\_message = {"method": "SUBSCRIBE", "params": ["btcusdt@kline\_1m"], "id": 2}

ws.send(json.dumps(subscribe\_message))

def on\_close(ws):

print("closed connection")

def on\_message(ws, message):

print("received a message")

print("My Bot!!!")

global closes, in\_position

json\_message = json.loads(message)

pprint.pprint(json\_message)

candle = json\_message['k']

is\_candle\_closed = candle['x']

close = candle['c']

if is\_candle\_closed:

if is\_candle\_closed:

print("candle closed at {}".format(close))

closes.append(float(close))

print("closes")

print(closes)

if len(closes) > RSI\_PERIOD:

np\_closes = numpy.array(closes)

rsi = talib.RSI(np\_closes, RSI\_PERIOD)

print("all rsis calculated so far")

print(rsi)

last\_rsi = rsi[-1]

print("the current rsi is {}".format(last\_rsi))

if last\_rsi < RSI\_OVERSOLD:

if in\_position:

print("It is oversold, but you already own it, nothing to do.")

else:

print("Oversold! Buy! Buy! Buy!")

# put binance buy order logic here

order\_succeeded = order(SIDE\_BUY, TRADE\_QUANTITY, TRADE\_SYMBOL)

oco = True

if order\_succeeded:

in\_position = True

#OCO order parameters

buy\_price = candle['l']

print("The price bought at was {}".format(buy\_price))

new\_buy\_price = float(buy\_price)

significant\_digits = 4

price\_profit = (new\_buy\_price \* TAKE\_PROFIT)

price\_loss = (new\_buy\_price \* STOP\_LOSS)

price\_limit\_loss = (price\_loss \* 0.99)

new\_price\_profit = round(price\_profit, significant\_digits - int(math.floor(math.log10(abs(price\_profit)))) - 1)

new\_price\_loss = round(price\_loss, significant\_digits - int(math.floor(math.log10(abs(price\_loss)))) - 1)

new\_price\_limit\_loss = round(price\_limit\_loss, significant\_digits - int(math.floor(math.log10(abs(price\_limit\_loss)))) - 1)

print(new\_price\_profit)

print(new\_price\_loss)

print(new\_price\_limit\_loss)

if last\_rsi < RSI\_OVERSOLD:

order\_succeeded= client.order\_oco\_sell(

symbol= TRADE\_SYMBOL,

quantity= TRADE\_QUANTITY,

price= new\_price\_profit,

stopPrice= new\_price\_loss,

stopLimitPrice= new\_price\_limit\_loss,

stopLimitTimeInForce= TIME\_IN\_FORCE\_GTC)

if order\_succeeded:

in\_position = True

if in\_position:

print("It is oversold, but you already own it, nothing to do.")

ws = websocket.WebSocketApp(SOCKET, on\_open=on\_open, on\_close=on\_close, on\_message=on\_message)

ws.run\_forever()