import time

import numpy as np

from binance import Client, ThreadedWebsocketManager, ThreadedDepthCacheManager

# Config

API\_KEY = "YOUR\_API\_KEY"

API\_SECRET = "YOUR\_API\_SECRET"

SYMBOL = "BTCUSDT"

SPREAD\_PERCENT = 0.01 # 0.1% spread

ORDER\_SIZE = 0.001 # 0.001 BTC per order

INVENTORY\_TARGET = 0.1 # Max 0.1 BTC net exposure

# Connect to Binance

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

depth\_cache = ThreadedDepthCacheManager(API\_KEY, API\_SECRET)

class MarketMaker:

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

self.inventory = 0

self.bid\_price = 0

self.ask\_price = 0

self.order\_ids = []

def update\_prices(self):

"""Fetch order book and set bid/ask prices"""

ob = client.futures\_order\_book(symbol=SYMBOL, limit=5)

mid\_price = (float(ob['bids'][0][0]) + float(ob['asks'][0][0])) / 2

self.bid\_price = mid\_price \* (1 - SPREAD\_PERCENT/2)

self.ask\_price = mid\_price \* (1 + SPREAD\_PERCENT/2)

def place\_orders(self):

"""Cancel old orders and place new ones"""

# Cancel existing orders

if self.order\_ids:

for order\_id in self.order\_ids:

client.futures\_cancel\_order(symbol=SYMBOL, orderId=order\_id)

# Place new orders

bid\_order = client.futures\_create\_order(

symbol=SYMBOL,

side='BUY',

type='LIMIT',

quantity=ORDER\_SIZE,

price=self.bid\_price,

timeInForce='GTC'

)

ask\_order = client.futures\_create\_order(

symbol=SYMBOL,

side='SELL',

type='LIMIT',

quantity=ORDER\_SIZE,

price=self.ask\_price,

timeInForce='GTC'

)

self.order\_ids = [bid\_order['orderId'], ask\_order['orderId']]

def check\_inventory(self):

"""Hedge if inventory exceeds target"""

positions = client.futures\_position\_information(symbol=SYMBOL)

self.inventory = float(positions[0]['positionAmt'])

if abs(self.inventory) > INVENTORY\_TARGET:

hedge\_side = 'SELL' if self.inventory > 0 else 'BUY'

client.futures\_create\_order(

symbol=SYMBOL,

side=hedge\_side,

type='MARKET',

quantity=abs(self.inventory)

print(f"Hedged {self.inventory} {SYMBOL}")

# Main loop

mm = MarketMaker()

while True:

try:

mm.update\_prices()

mm.place\_orders()

mm.check\_inventory()

time.sleep(1) # Adjust based on rate limits

except Exception as e:

print(f"Error: {e}")

time.sleep(5)