Willène MANARI
Nichart SOTHINATHAN
ING4 FINANCE

Projet Intelligence Artificielle n°2 :

Trading Algorithmique sur Cryptomonnaie

Sommaire

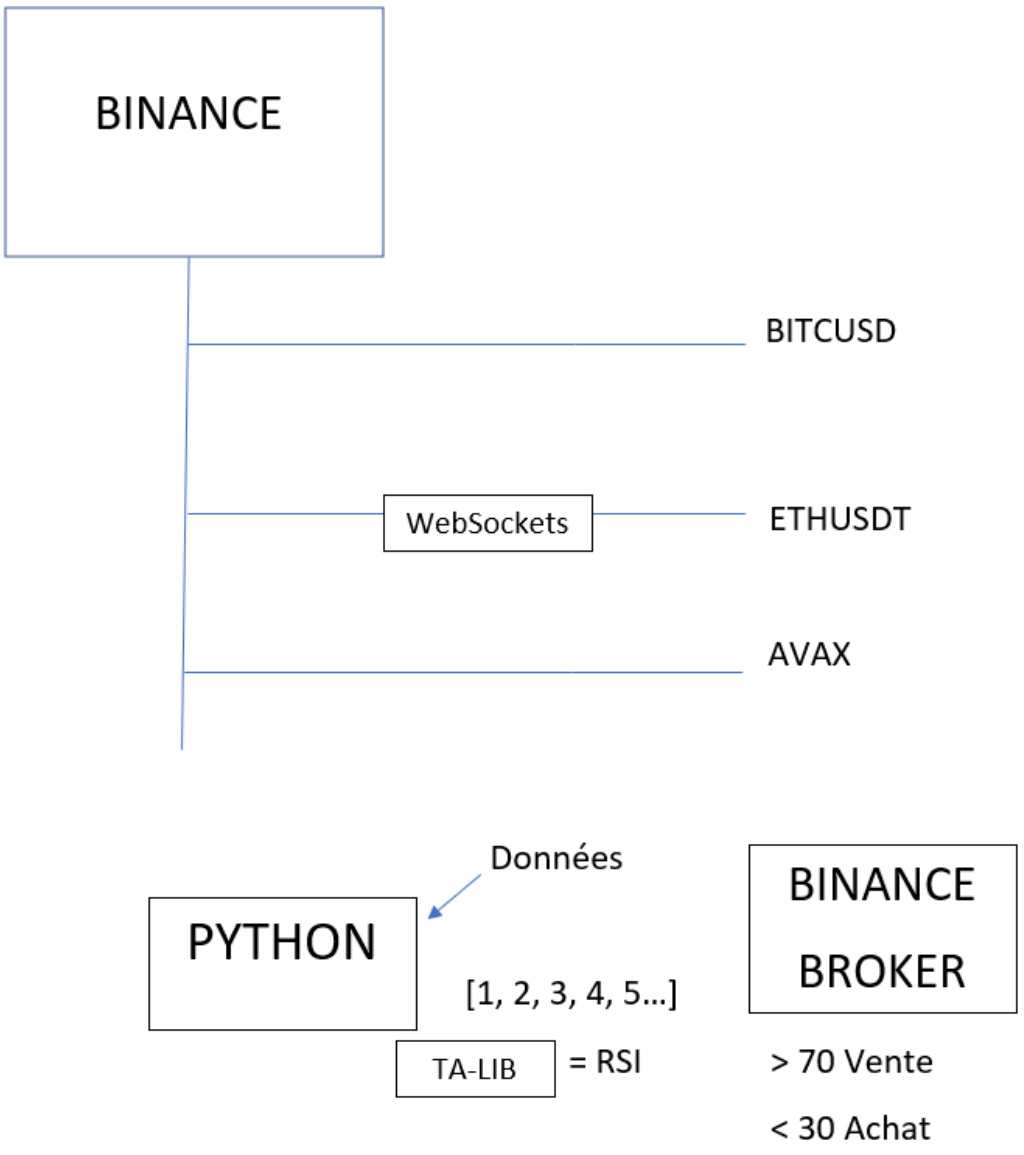
I.Introduction

De nos jours, l'informatisation de la plupart des services financiers, nous pousse à nous intéresser à la notion de cryptomonnaie. Cette devise numérique utilise des protocoles de blockchain pour assurer la traçabilité des transactions effectuées par des utilisateurs. En d'autres termes il s'agit d'une monnaie dématérialisée, dont la popularité et l'utilisation accroît de manière de manière exponentielle. Phénomène s'expliquant par la mondialisation et le partage constant de données informatiques.
Le trading consiste en l'achat de titres financiers, dans le but de revendre ces derniers au meilleur prix. En appliquant les principes du trading à l'échange de crypto-monnaies, cette dernière devient alors un "titre" financier soumis à toutes les lois du marché.

Ce projet nous offre l'opportunité de mettre en pratique, la théorie vu en cours d'Intelligence Artificielle, en appliquant la cryptomonnaie aux principes du trading. Avec ce projet, nous souhaitons automatiser le travail d'un trader et d'un broker en codant un algorithme Python, capable d'acheter et de vendre cette crypto monnaie en fonction des flux financiers. On donnera alors accès au bot a un portefeuille et on lui fournira des ordres à éxecuter.

Pour être plus précis , nous avons choisis de concevoir un bot qui se connectera à la plateforme d'exchange binance et assurera le trading de la paire de crypto-monnaies Ethereum/USDT.

II.Stratégie



Binance va être notre Broker (un intermédiaire habilité a effectuer des opérations), il va donc exécuter les ordres qu'on va lui transmettre via son API. Il va aussi nous être utile pour récupérer des données via WebSockets. Comme le schéma ci-dessus, Binance propose plusieurs flux de données continus, avec les prix des différentes monnaies crypto, avec des noms différents. WebSockets va nous permettre de nous connecter avec un flux en particulier.

Grâce à ce client on va pouvoir accumuler ces données dans Python et lire ces dernières. On va s'intéresser au prix les plus proche (1, 2, 3, etc..) et on va utiliser la librairie TA-LIB pour dresser un RSI.

Un RSI nous permettra de savoir si on est dans une situation de vente ou d'achat.

L'indice de force relative consiste en la mesure de la force relative interne d'un stock ou d'un marché avec lui-même. Grâce à ses valeurs attribuées à ses pics on pourra alors savoir si nous sommes dans de bonnes conditions pour acheter ou vendre.

A partir de là cet indicateur RSI nous donnera des valeurs que l'on va comparer par rapport à 70 et 30. En fonction de cela on demandera à Binance en tant que Broker de vendre ou d'acheter (l'utilité d'un bot). Si le RSI est supérieur à 70 on sera en situation d'overbought, il sera donc approprié de vendre. Et si le RSI est inférieur a 30 on sera en situation d'oversold, il sera donc approprié d'acheter.

Grâce à ce graphique on pourra voir les pics de surcoût et de survente.

Le problème que l'on pourrait rencontrer est que si par exemple le RSI est à 29 et ensuite 28.5 et donc que le mouvement est pendant longtemps en mouvement d'oversold, cela résulterait par de multiples ordres d'achat par le bot et nous devrions donc de la même façon à l'inverse vendre en plusieurs fois si on est en situation continue d'overbought. Ce n'est pas ce que l'on souhaite ici, nous voulons juste acheter une fois puis vendre. Nous blinderons donc le bot pour qu'il ne puisse pas passer d'ordre d'achat s'il est déjà en position. Ainsi nous ne nous retrouverons pas avec des situations compliquées.

III.Infrastructure

Nous avons commencé par créer un fichier bot.py

Pour la réalisation de ce projet nous allons utiliser un certain nombre de librairies. Nous allons utiliser websocket, nous allons donc avoir un client websocket. On utilise aussi json, numpy, talib et également un fichier local.

On va aussi utiliser le package python de Binance comme on peut le voir ci-dessous

```
1   import websocket, json, pprint, talib, numpy
2   import config
3   from binance.client import Client
4   from binance.enums import *
```

Nous voulons tout d'abord avoir un accès au cours ETH/USDT depuis notre bot, pour cela nous avons utiliser le client websocket, grâce à la documentation fourni par binance nous avons pu avoir accès au cours ETH/USDT sous forme de candle par tranche de 1 minute, nous avons choisis 1 minutes, pour pouvoir procéder à du trading sur très court terme et ainsi voir si le bot fonctionne sans attendre plusieurs jours. De plus, le trading à court terme nous semblait plus populaire que le trading à long terme.

kline représente ici le cours sous forme de candlesticks et 1m la période d'une minute

```
SOCKET = "wss://stream.binance.com:9443/ws/ethusdt@kline_1m"
```

Une fois connecté au flux, le client websocket demande de définir quelques fonctions, la fonction de l'événement connexion, la fonction de l'événement fermeture et ainsi que la fonction de l'événement "on_message". Ces fonctions se lancent donc en conséquence à chacun de ces événements, à l'ouverture, la fermeture, etc...

```
ws = websocket.WebSocketApp(SOCKET, on_open=on_open, on_close=on_close, on_message=on_message)
ws.run_forever()
```

Nous avons donc ensuite définit chacune de ces fonctions dans le code :

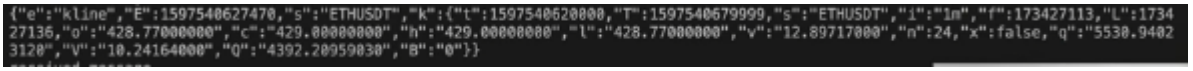
```python
def on_open(ws):
    print('connexion etablie')

def on_close(ws):
    print('connexion fermee')

def on_message(ws, message):

    print('message recu')
```

A partir d'ici, si nous lançons le programme, nous recevons donc des données du cours ETH/USDT de binance. Nous allons donc nous intéresser à comment traiter ces données.



Tout d'abord pour rendre ces données plus lisibles nous utiliserons la librairie json la librairie pprint. Ainsi nous obtiendrons les données dans un format beaucoup plus lisible, comme celui ci :

Comme on va utiliser l'indicateur RSI, la principale donnée que l'on va utiliser est la closing value de chaque candlestick

Comme on sait (depuis l'image ci-dessus) que quand x est true cela correspond à la fermeture du candlestick.

Il nous suffit donc de récupérer la valeur de fermeture des candlestick lorsque x est true après avoir initialiser la liste "closes" :

```
candle = json_message['k']

is_candle_closed = candle['x']
close = candle['c']

if is_candle_closed:
    print("candle closed at {}".format(close))
    closes.append(float(close))
    print("closes")
    print(closes)
```

Nous voulons donc obtenir un grand nombre de valeurs de fermeture. Ensuite nous convertirons la liste en tableau numpy et on utilisera la librairie ta lib pour appliquer un indicateur RSI à ces valeurs et ainsi obtenir une valeur RSI.

On va donc commencer par initialiser une constante pour le RSI que l'on fera varier, on mettra donc des valeurs par défaut fournies par les utilisateurs de bot de trading algorithmique.
On ajoute également les constantes RSI d'achat et de vente, c'est donc les valeurs RSI à partir desquelles on va acheter ou vendre.

Et finalement les constantes représentant les cryptos que l'on va acheter/vendre et leur quantité à trade à chaque ordre.

```
RSI_PERIOD = 14
RSI_OVERBOUGHT = 70
RSI_OVERSOLD = 30
TRADE_SYMBOL = 'ETHUSD'
TRADE_QUANTITY = 0.05
```

On veut calculer le RSI sur une periode de 14 fermeture, nous devons donc avoir une liste de minimum 15 closing value pour pouvoir calculer le RSI.

Nous allons donc ajouter une condition et ainsi convertir la liste de valeurs en tableau numpy et calculer les valeurs du RSI en applicant l'indicateur RSI depuis talib si bien sur la condition est remplie.

```
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))
```

On va maintenant connecter le bot au client binance grâce aux librairies importées initialement. Nous avons trouvé la ligne de code nécessaire grâce à la documentation fournie par binance :

```
client = Client(config.API_KEY, config.API_SECRET, tld='us')
```

Nous devrons donc renseigner la clé API et le mot de passe API pour connecter le bot au compte binance.

Nous allons maintenant coder la fonction order (grâce à la documentation binance) pour pouvoir placer des ordres d'achat et de vente sur binance :

```
def order(side, quantity, symbol,order_type=ORDER_TYPE_MARKET):
    try:
        print("sending order")
        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

    return True
```

Ensuite, selon la valeur du RSI que l'on aura on va placer des ordres d'achat ou de vente de la cryptomonnaie. SI le RSI est supérieur au RSI_Overbought on sera donc en situation ou il y a un gros mouvement d'achat, il sera donc moment de vendre. Et inversement si le RSI est inferieur au RSI_oversold nous serons donc en situation où il y a un gros mouvement de vente, il sera donc moment d'acheter.

```python
if last_rsi > RSI_OVERBOUGHT:
    if in_position:
        print("Overbought! Sell! Sell! Sell!")
        # put binance sell logic here
        order_succeeded = order(SIDE_SELL, TRADE_QUANTITY, TRADE_SYMBOL)
        if order_succeeded:
            in_position = False
    else:
        print("It is overbought, but we don't own any. Nothing to do.")

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)
        if order_succeeded:
            in_position = True
```

Voici maintenant le code au complet :

```python
import websocket, json, pprint, talib, numpy
import config
from binance.client import Client
from binance.enums import *

SOCKET = "wss://stream.binance.com:9443/ws/ethusdt@kline_1m"

RSI_PERIOD = 14
RSI_OVERBOUGHT = 70
RSI_OVERSOLD = 30
TRADE_SYMBOL = 'ETHUSD'
TRADE_QUANTITY = 0.05

closes = []
in_position = False

client = Client(config.API_KEY, config.API_SECRET, tld='us')

def order(side, quantity, symbol,order_type=ORDER_TYPE_MARKET):
    try:
        print("sending order")
        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

    return True


def on_open(ws):
    print('opened connection')

def on_close(ws):
    print('closed connection')

def on_message(ws, message):
    global closes, in_position

    print('received message')
    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:
        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_OVERBOUGHT:
                if in_position:
                    print("Overbought! Sell! Sell! Sell!")
                    # put binance sell logic here
                    order_succeeded = order(SIDE_SELL, TRADE_QUANTITY, TRADE_SYMBOL)
                    if order_succeeded:
                        in_position = False
                else:
                    print("It is overbought, but we don't own any. Nothing to do.")

            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)
                    if order_succeeded:
                        in_position = True


ws = websocket.WebSocketApp(SOCKET, on_open=on_open, on_close=on_close, on_message=on_message)
ws.run_forever()
```

IV.Résultat

Pair	Type	Side	Average	Price	Executed	Amount	Total	Trigger Conditions	Status
ETH/USD	Market	Sell	422.49	Market	0.05000	0	21.13	–	Filled
ETH/USD	Market	Buy	426.68	Market	0.05000	0	21.34	–	Filled
ETH/USD	Market	Sell	428.74	Market	0.05000	0	21.44	–	Filled
ETH/USD	Market	Buy	428.73	Market	0.05000	0	21.44	–	Filled
ETH/USD	Market	Buy	428.51	Market	0.05000	0	21.43	–	Filled
ETH/USD	Market	Sell	428.12	Market	0.05000	0	21.41	–	Filled
ETH/USD	Market	Sell	433.09	Market	0.05000	0	21.66	–	Filled
ETH/USD	Market	Buy	433.33	Market	0.05000	0	21.67	–	Filled

En faisant tourner le bot une nuit, on obtient plusieurs échanges (achat et vente) qui ont été réalisés.