Rapport de projet : Algorithme de Trading automatique

ALINEJAD Kévin
kevin.alinejad@telecom-sudparis.eu

BITAN Paul
paul.bitan@telecom-sudparis.eu

FULOP Aymeric
aymeric.fulop@telecoms-sudparis.eu

VOL Sébastien
sebastien.vol@telecom-sudparis.eu

KHEDIMI Bilal
bilal.khedimi@telecom-sudparis.eu

Enseignant résponsable : Daniel Ranc

February 2022

Contents

1 Introduction **3**
1.1 Les moyennes mobiles . . . 3
1.2 Les bandes de Bollinger . . . 5
1.3 Les Volumes . . . 6
1.4 Le RSI . . . 6
1.5 MACD . . . 7
1.6 Les bougies japonaises . . . 7
1.7 Liste types d'ordres . . . 9
1.8 Analyse des données On-chain . . . 10

2 Cahier des charges **10**

3 Développement **11**
3.1 Analyse du problème et spécification fonctionnelle . . . 11
3.2 Conception préliminaire . . . 14
3.3 Conception détaillée . . . 17
3.3.1 Indicateurs.py . . . 17
3.3.2 bot.py . . . 21
3.3.3 main.py . . . 25
3.4 Codage . . . 28
3.5 Tests unitaires . . . 28
3.6 Tests d'intégration . . . 28
3.7 Tests de validation . . . 28

4 Intérpretation des résultats **28**

5 Manuel utilisateur **28**

6 Conclusion **28**

7 Annexes **28**
7.1 Gestion de projet . . . 28
7.1.1 Plan de charge . . . 28
7.1.2 Planning prévisionnel . . . 29
7.1.3 Suivi d'activités . . . 29
7.2 Stratégies . . . 29
7.3 Configuration de bougies japonaise . . . 29
7.3.1 Le parabolique SAR . . . 30
7.3.2 Stratégies basées sur les données On-chain . . . 31

1 Introduction

Cette introduction a pour but d'introduire rapidement des concepts d'analyse technique des marchés. Nous nous sommes principalement basés sur l'ouvrage de *John Murphy, Analyse technique des marchés financiers.* Pour plus d'informations, nous vous invitons à vous référer à cet ouvrage car nous ne ferons que survoler quelques indicateurs importants sans rentrer dans les détails.
Il est important de comprendre cette notion qui est fondamentale dans l'analyse technique : Le Trader (l'analyste) ne cherche - grâce à des techniques (indicateurs) - qu'à déterminer la tendance du marché. En effet, si nous supposons la tendance du marché connu, il suffit d'ouvrir une position dans le sens du marché, rien de plus.

1.1 Les moyennes mobiles

La moyenne mobile donne le cours moyen d'une valeur sur une période donnée.Elle est appelée mobile car son calcul est effectué jour après jour. Il existe trois types de moyennes mobiles :

- Moyenne mobile arithmétique : sur X jours, on fait la somme des cours de clôture des X dernières séances, somme divisée ensuite par X. Elle est recalculée chaque jour. Donc :

$$MM(n,t) = \frac{1}{n}\sum_{i=0}^{n-1} X_{t-i} \tag{1}$$

  Avec n : la durée de la période choisie et les X_i les valeurs prises par notre série financière. Et t l'instant choisit.

- Moyenne mobile pondérée : les données les plus récentes sont affectées d'un poids plus important que les données plus anciennes, ceci afin de mettre plus facilement en évidence les variations récentes du cours.

- Moyenne mobile exponentielle : encore plus de poids est donné au cours le plus récent. En effet, Pour essayer de pallier ce manque de réactivité et donner plus d'importance aux données les plus récentes, on peut utiliser la moyenne pondérée avec notamment celle exponentielle qui comme son nom l'indique utilise une pondération à décroissance exponentielle. Dans celle-ci, une constante de lissage contrôle le degré de décroissance des poids applicables à chaque observation. Cette constante que l'on notera α (comprise entre 0 et 1). On la définit ensuite de cette façon:

$$EMA(n,t) = \sum_{i=0}^{n-1} \alpha(1-n)^i X_{t-i} \tag{2}$$

  En utilisant les mêmes notations que pour (1)

Cette indicateur est très populaire auprès des analystes. Des stratégies gagnantes, bien que peu efficace, peuvent être misent au point en utilisant uniquement cette indicateur. On peut par exemple s'en servir pour identifier les tendances :

Lorsque la pente de la moyenne mobile est positif, on représente la moyenne en bleu et en rouge si elle est négatif. Une idée toute simple consiste se positionner dans ces changements de direction qui sont censés être des retournements de tendance.
On peut aussi utiliser plusieurs moyennes mobiles :

Lorsque la MA_X (Moving Average sur X jours ou Moyenne Mobile) croise la MA_Y vers le haut (la MA_X se retrouve au dessus de la MA_Y) avec $X < Y$ un signal d'achat est donné et inversement si le croisement est vers le bas.

1.2 Les bandes de Bollinger

Les moyennes mobiles sont également utilisées pour tracer les bandes de Bollinger. Enfait, il s'agit observer la position relative du cours par rapport à la moyenne mobile. Par exemple si les cours viennent à dépasser la moyenne à la hausse, il s'agit d'une bonne indication d'achat. Les moyennes mobiles sont donc couramment accompagnée des bandes de Bollinger qui enserrent la série financière dans une sorte de tube. Pour cela on définit d'abord la distance moyenne entre la série et sa moyenne mobile suivant la formule suivante :

$$dist(n,t) = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(X_{t-i} - MM(n,t))^2} \tag{3}$$

Les bandes de Bollingers est l'intervalle de prix :

$$[MM(n,t) - \alpha dist(n,t), MM(n,t) + \alpha dist(n,t)]$$

Où α est un paramètre arbitraire qui détermine la longueur de l'intervalle (en général +/- deux écarts types à la moyenne). Ces bandes permettent de savoir quand les cours s'éloignent trop d'une moyenne en prenant en compte la volatilité (qui est la base de la gestion du risque). Les bandes supérieures et inférieures définissent une zone de volatilité «normal» de l'actif. Plus la volatilité augmente, plus les bandes s'élargissent et si l'actif est à tendance neutre la dispersion des valeurs va être de plus en plus faibles comme le montre la premier tiers du graphique ci-dessous :

Graphique issu du logiciel d'analyse technique de Waldata sur l'indicateur de volatilité

De même sur la zone centrale le prix décale fortement (vers la baisse) et dépasse sa volatilité habituelle et sort de sa bande centrale. Les bandes supérieures et

inférieures s'éloignent. Dans le cas pratique, on remarque que les changements de prix majeurs ont tendance à se produire après un resserrement des bandes (on ne sait ceci dit pas quand exactement puisqu'elles peuvent continuer . De même lorsque les prix sortent nettement d'une bande, une poursuite de la tendance s'ensuit.

Ses avantages :

- Identifie les phases de latéralisation
- Identifie le démarrage de nouvelle tendance avec une augmentation de la volatilité
- Permet de suivre facilement la volatilité d'un actif

Ses inconvénients :

- Pris tel quel, l'indicateur fournie beaucoup de faux signaux. Il faut nécessairement filtrer chaque signal. En fait, cet inconvénient est généralisable à tous les indicateurs de cette introduction.

1.3 Les Volumes

Le volume mesure la quantité de titre échangé au cours d'une période donnée, de ce fait ils nous indiquent le niveau d'activité sur les marchés en nous permettant de confirmer la pertinence d'un support ou d'une résistance ou bien d'une tendance, en somme il nous permet de renforcer la qualité d'une analyse chartiste par leur fort aspect psychologique. C'est un véritable signal de confirmation.

1.4 Le RSI

Le RSI est un indicateur de suivi de vitesse des prix qui, borné entre 0 et 100 %, permet de déterminer la force interne d'une valeur dans le temps. Le RSI met en évidence des zones de sur achat et de survente. Le calcul du RSI crée un rapport entre les moyennes des hausses et les moyennes des baisses. Le RSI est calculé de la manière suivante :

$$RSI = 100 - \frac{100}{1 + \frac{H}{B}}$$

H : moyenne mobile exponentielle des hausses au cours des X derniers jours ;

B : valeur absolue de la moyenne mobile exponentielle des baisses au cours des X derniers jours. (Bien que l'on parle de baisses, B est positif)

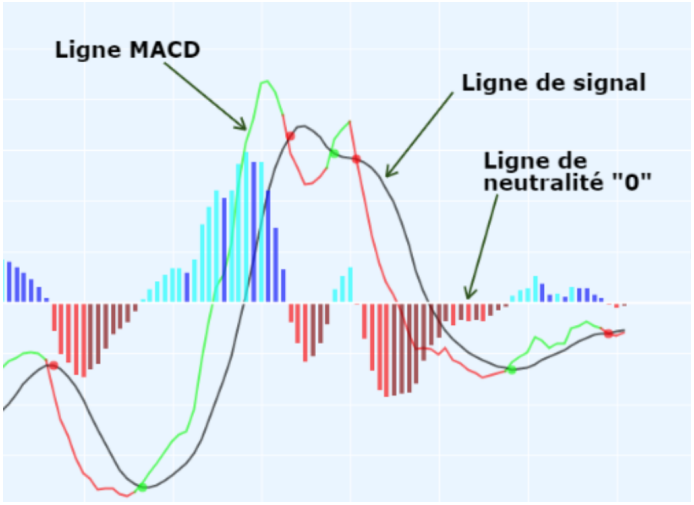
Comment le lire ? Il apparaît deux bornes importantes :

- Si le RSI chute en dessous de 30%, cela signifie que l'action est en niveau de survente.
- Si au contraire il passe au-dessus de 70%, l'action est en zone de sur-achat.

Le RSI peut également communiquer une autre information : dans une situation de tendance haussière, si le prix de titre atteint un nouveau sommet mais que le RSI ne crée pas de nouveau sommet, il y a une divergence baissière. C'est un fort signal de renversement de tendance. En bref, le RSI permet de repérer la puissance d'un mouvement (indiquer si le mouvement s'essouffle).

1.5 MACD

La MACD représente l'écart des moyennes mobiles aux cours, et sa courbe se trace sur le graphique de l'évolution du cours - en se fixant une ligne zéro de la MACD. Développée par Gerald Appel2, elle se calcule instantanément par la différence entre une moyenne mobile exponentielle de long terme et une moyenne mobile exponentielle de court terme (communément 26 et 12 périodes). Cette première courbe, qui est donc un simple oscillateur de moyennes mobiles exponentielles, est appelée ligne de MACD rapide, puis on ajoute sur le même graphique une seconde courbe appelée ligne de signal représentant une moyenne mobile exponentielle de période 9 de la première courbe.

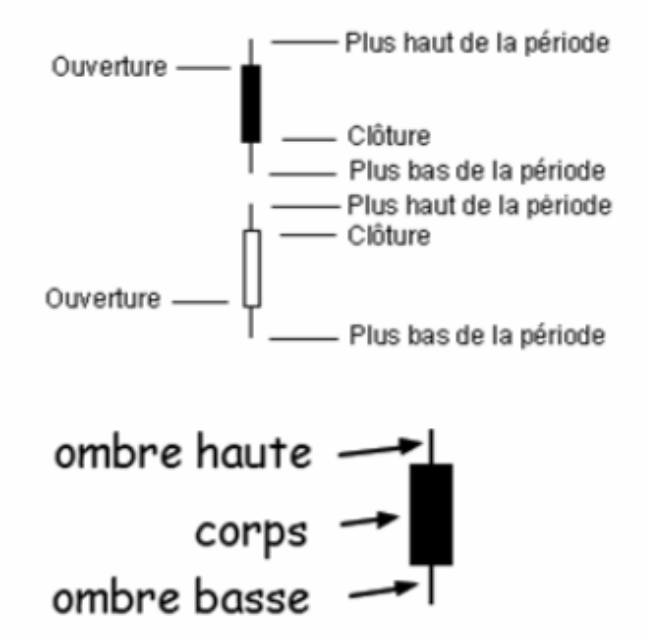


L'analyse de la MACD permet d'anticiper techniquement l'évolution des marchés. Ainsi, il est préconisé d'acheter lorsque la courbe de MACD rapide coupe à la hausse la ligne de signal lente, en effet, les croisements identifient les changements dans l'équilibre des pouvoirs entre haussiers et baissiers. À l'inverse, le franchissement à la baisse de la courbe de signal lente par celle de la MACD rapide est un signal de vente.

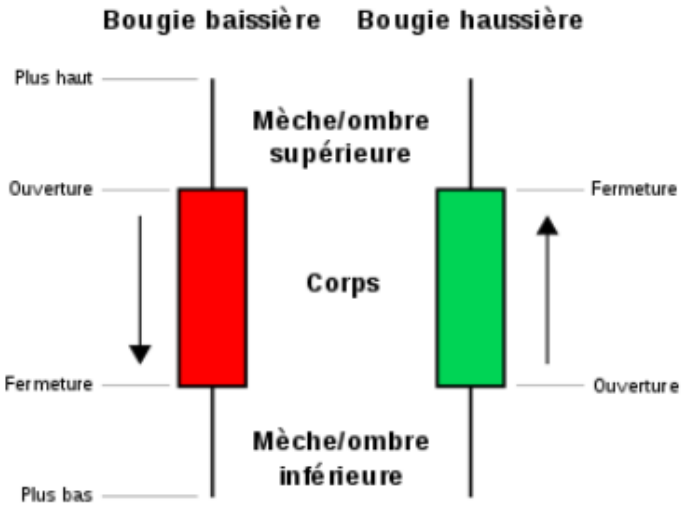
1.6 Les bougies japonaises

Les chandeliers japonais, ou bougies japonaises, sont un type de représentation des cours de bourse. Typiquement, une bougie représente une séance de cotation d'un actif financier. Par extension, un chandelier, autre nom de la bougie,

représente aussi une barre correspondant à une unité de temps. Cette unité de temps peut varier, cela peut être 1an, 1trimestre, 1mois, 1jour,4heures,1heure, 15mn, 3mn, 1mn...



Le rectangle est le corps de la bougie et les deux traits aux extrémités sont les mèches ou les ombres. Un chandelier est défini comme un ensemble de bougies, et ce mot peut désigner aussi une seule bougie.



Une bougie peut montrer comment les vendeurs et les acheteurs ont évolué sur le marché au cours de la dernière bougie. En effet, plus la mèche et le corps seront grands, plus les vendeurs et les acheteurs se seront livrés à un combat intense. Certaines formes de bougies permettent de mieux comprendre le marché et d'annoncer un certain retournement haussier ou baissier, ou encore la continuation d'une tendance.

1.7 Liste types d'ordres

Voici les ordres que l'on peut donner dans un marché :

- Ordre « au marché » : acheter/vendre une quantité de titres sans limite de prix privilégié dans l'ordre d'exécution mais aucune maitrise dans le risque de modification du prix.
- Ordre « à la meilleur limite » : transmis sur le marché sans indication de prix, l'investisseur délègue au marché de placer l'ordre de manière à avoir la meilleure contrepartie.
- Ordre « à cours limité » : acheter/vendre au niveau d'une certaine limite de prix.
- Ordre « à plage de déclenchement » : donne un intervalle dans lequel l'opération peut se produire.
- Ordre « stop on quote » : donne une limite, une fois qu'elle est dépassée, envoi un ordre « au marché ».
- Ordre « stop on quote limité » : donne une limite, une fois qu'elle est dépassée, envoi un ordre « à cours limité ».

- Ordre TAL : permet de bénéficier d'une période de négociation supplémentaire après les fixings du jour. Pendant cette plage horaire, le titre se négocie uniquement au dernier cours coté.

1.8 Analyse des données On-chain

L'analyse on-chain est une branche du domaine de l'analyse de données dédiée à l'étude d'une blockchains. Pour pouvoir en expliquer de la meilleure des manières son fonctionnement, il nous faudra tout d'abord la définition et l'explication des termes principaux. Une blockchain est "une technologie moderne de stockage et de transmission d'informations. Elle fonctionne sans organe central de contrôle, mais apporte transparence et sécurité grâce à la validation des transactions par les nœuds du réseau" (définition d'après futura-tech). C'est une sorte d'immense livre des comptes dans lequel chaque transaction est écrite, ineffaçable et non modifiable. La multiplicité des acteurs participant au réseau est à l'origine de sa décentralisation. Personne ne peut contrôler la blockchain car il faut l'approbation de l'ensemble des acteurs pour valider une transaction.

L'analyse des données on-chain consiste à regrouper de nombreuses données que l'on possède sur les acteurs d'une blockchain. Comme c'est un immense livre des comptes, on peut regarder les volumes d'échanges, les pertes et les gains de chaque acteur (que l'on nommera aussi "adresse" par la suite) par exemple. L'analyse des données on-chain observe et synthétise le comportement global des participants d'un réseau pour en extraire des signaux de bonne santé et d'investissement.

Ces informations sont d'une grande utilité pour les investisseurs et pour notre algorithme de trading car ils nous donnent des signaux à l'achat ou à la vente grâce à ces informations. Cela donne des indications sur les personnes détenant certaines cryptomonnaies, le nombre de tokens qui sont sur des exchanges ou alors sur un wallet, les personnes qui en profitent avec une certaine cryptomonnaie ...

2 Cahier des charges

L'objectif est de mettre en place une application permettant à l'utilisateur d'acheter et vendre des actifs financiers, en d'autres termes, l'objectif est de fournir une application permettant de trader. Ainsi les fonctionnalités de l'application sont les suivantes :

- **S'identifier à la plateforme Metatrader5**
- **Proposer une liste de stratégies**
- **Pouvoir passer des ordres d'achat et de vente de manière automatique en fonction des stratégies sélectionnées.**

- **Faire les comptes.**

Ces fonctions seront proposées à travers une interface graphique où l'utilisateur aura le choix entre différentes méthodes de trader et chaque méthode sera expliqué (chiffres clés, quel type de trading etc.. . .). Il pourra répartir son capital selon différentes méthodes. Pour cela, nous utiliserons le langage Python. L'interface devra être assez intuitif afin de faciliter l'utilisation.

D'autres extensions seraient envisageables, par exemple une consultation en mode web à travers l'utilisation d'un navigateur (une application complète en ligne avec ajout et suppression demanderait la gestion de l'accès concurrent aux données de l'application côté serveur)

Contraintes techniques :

- Manque de connaissance en analyse technique. Il faut apprendre ces notions au cours du projet.
- L'utilisation de multiple bibliothèques python : metatrader5, pandas, yfinance, numpy, matplotlib, TA-Lib, TKtinter.

3 Développement

3.1 Analyse du problème et spécification fonctionnelle

Cette partie vise à lever toutes les ambiguïtés et répondre à de nombreuses questions. Nous donnons ci-dessous les principales questions et les réponses qui y ont été apportées :

Avec le capital de qui allons nous proposer notre application ?
Il faut bien comprendre que nous n'allons pas réellement mettre en jeu de l'argent en jeu. Seul de l'argent "virtuel" est mis en jeu. Beaucoup de plateforme d'échange propose de faire du Paper Trading : Le Paper Trading consiste à simuler des achats et des ventes d'instruments financiers (actions, warrants. . .). On fait comme si, on essaye de se mettre en situation.
Ainsi, l'utilisateur n'a que le résultat de l'application. Nous avons fait ce choix pour limiter le risque, aucun membre du projet est un expert en investissement financier.

Où acheter un actif ?
Les Actifs seront achetés en passant par la plateforme MetaTrader5 principalement, qui sera notre courtier

Pourquoi utiliser cette plateforme ?
MetaTrader5 est une plateforme largement utilisé dans l'univers du trading, mais surtout une bibliothèque python très détaillée est fournie : des fonctions permettant de fournir des ordres, d'obtenir des trames de

données de l'actif, d'avoir l'historique des ordres éffectué sur un actif spécifique etc ... En clair cette plateforme simplifie notre travail en se chargeant d'aspect non relatif à l'analyse technique.

Est ce que l'utilisateur a accès à toutes les fonctionnalites ?
On considère ici que tous les utilisateurs ont le même privilège (c'est-à-dire qu'il n'y a pas de notion d'administrateur)

Comment l'utilisateur s'authentifie-t-il ?
L'utilisateur doit avoir un compte MetaTrader5 déjà crée, ainsi que les paramètres pour le Paper Trading initialisés. Une fonction de Metatrader5 permet de s'authentifier à la plateforme.

Doit-on gérer les accès simultanés ?
On considère qu'à un moment donné, il n'y a qu'un seul utilisateur de l'application.

Quels sont les actifs mis en jeu ?
Des actifs de type actions, des cryptomonnaies, et éventuellement des futures. Il n'y a pas de produits financier complexe de type options, Warrants ...

Comment récuperer les données On-chain ?
Pour récupérer ces outils et les utiliser dans notre projet, nous allons utiliser différentes API disponibles depuis des sites internet spécialisés dans l'agglomération des données tels que : Glassnode, Cryptoquant ou encore Santiment. Une fois que nous avons récupéré ces données nous mettrons en place des signaux d'achat ou de ventes corrélés à ce que nous apprennent les données.

Quels sont les résultats envisageable ?
Il ne faut pas s'attendre à devenir millionaire. L'analyse technique est un métier, donc au mieux nous pouvons espérer être en positif ou nul.

Pour chacune des fonctionnalités offertes par l'application, il convient de préciser les conditions d'utilisation si nécessaire :

1. **S'identifier à la plateforme Metatrader5 :** Doit pouvoir s'identifier à la plateforme.
   Test de validation :
   - Donnée : un identifiant invalide (utilisateur et mot de passe aléatoire)
     Résultat : un message d'erreur.
   - Donnée : un identifiant valide
     Résultat : accès a l'application.
2. **Proposer une liste de stratégies :** L'utilisateur doit pouvoir observer une liste de stratégies, avec une description spécifique pour chacune.
   Test de validation :

- Donnée : plusieurs textes (éventuellement des graphiques ou tableaux)
  Résultat : Les textes sont bien pris en compte par l'interface graphique avec un rendu correcte. Une présentation conviviale et une bonne coordination entre ces textes est désirée.

3. **Pouvoir passer des ordres d'achat et de vente :** L'objectif étant de proposer une application qui ouvre et ferme des positions de manière automatique piloté par une interface graphique sans ambiguité, l'utilisateur ne doit donc pas se soucier de ce qui se trouve "derrière" l'interface.
   Test de validation :

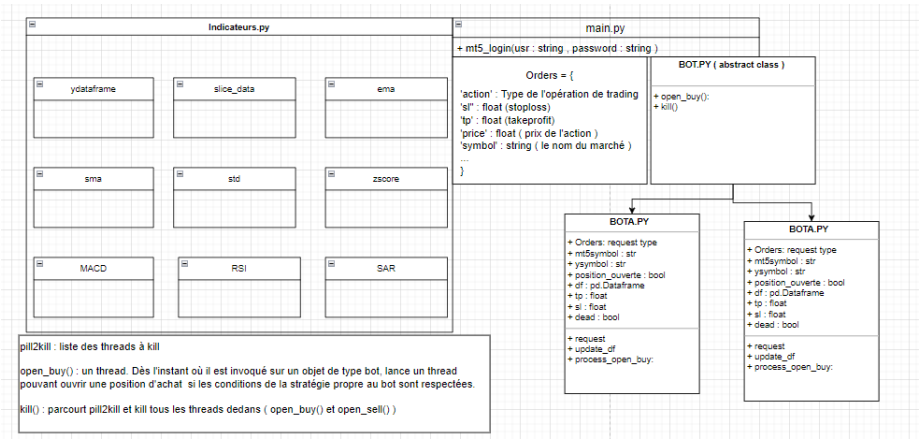
   - Donnée : un ordre (d'abord d'achat puis de vente à découverte)
     Résultat : L'ordre a été pris en compte par Metatrader5.

4. **Faire les comptes :** L'utilisateur doit à tout moment pouvoir faire ses comptes : L'interface doit présenter le montant disponible, ainsi que d'autres informations : pertes ou gains par rapport au montant initial, listes des opérations, gains ou pertes pour chaque opérations etc ... le tout sur une interface conviviale !
   Test de validation :

   - Donnée : Un ordre
     Résultat : L'interface graphique indique que l'odre a été pris en compte, le gain (perte) de l'opération pour le moment, le montant disponible etc ...

3.2 Conception préliminaire

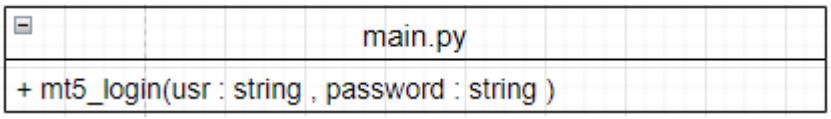
Pour ce qui est de l'architecture globale du système, nous avons choisi de décomposer l'application en plusieurs modules. Voici le schéma de l'architecture:



Cette séction vise donc à expliquer les modules et les intéractions entre eux. Comment on peut le constater sur le schéma, il y a :

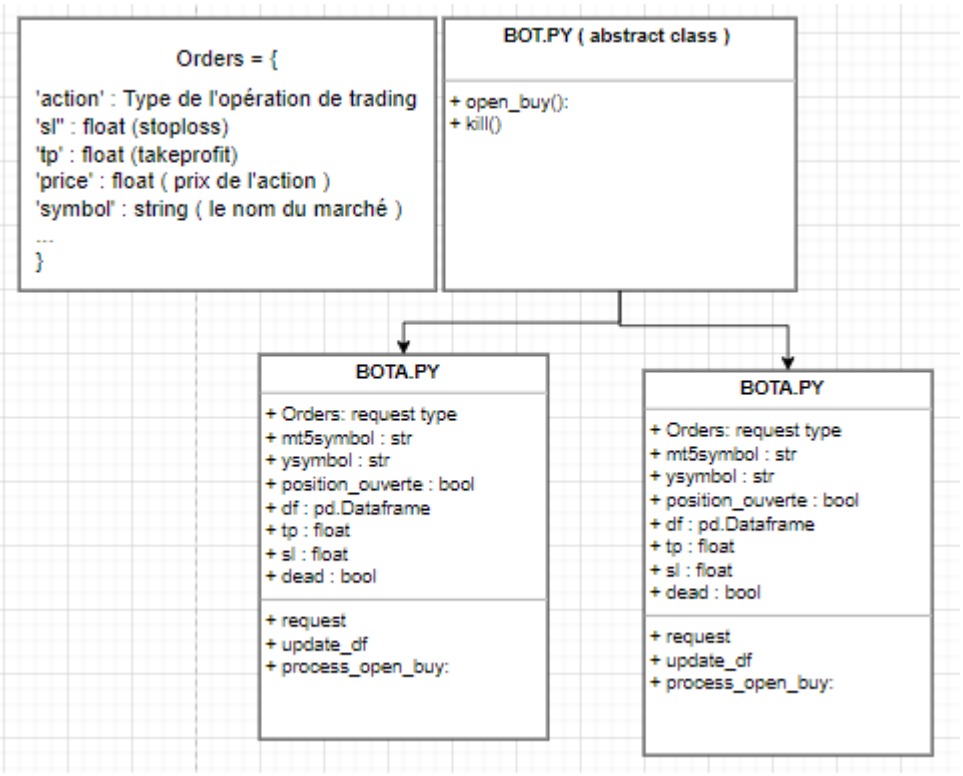
Indicateurs : Il s'agit d'un fichier python contenant les fonctions de type indicateurs. Ces fonctions sont indépendant des stratégies implementées dans l'application. Par exemple, il est tout à fait possible qu'une stratégie A n'utilise que deux ou trois indicateurs, mais qu'une stratégie B en utilise deux autres. Cependant, l'implémentation de la stratégie ne se fait pas à ce niveau. Pour l'instant, ces fonctions permettent de donner l'état de santé du marché, des indications et rien de plus. Les différents indicateurs retenus seront ainsi que les descriptions des fonctions à implémenter seront détaillés dans la prochaine partie.

main.py correspond à la fonction principale, avec le point d'entrée de l'application (initialisation, affichage d'un menu). Il y a une petite spécificité : une fonction d'identification est à définir en plus des fonctionnalités liées à l'interface graphique :



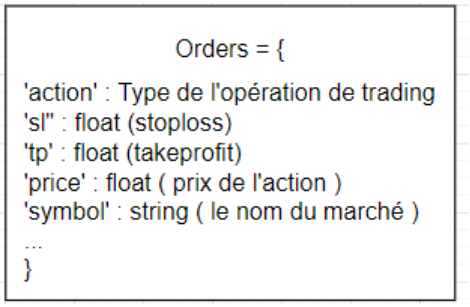
Comme précisé précedemment, nous allons passer par le broker MetaTrader5. Il faut donc un identifiant et un mot de passe pour accéder à l'application, qui sera à préciser via l'interface graphique par l'utilisateur. Toutes les fonctionnalités de l'interface graphique ne sont pas explicitées ici.

BOT.PY et ses enfants : Il s'agit de la partie qui automatise les stratégies de trading. Pour ce projet nous avons choisi d'opter pour une approche en programmation orientée objet. La classe abstraite BOT est le parent des autres classes de type BOT :



Pour chaque stratégie, la structure du programme est la même. Seule certaines méthodes changent, mais leurs attributs restent identiques, d'où cette architecture.

- Orders est l'ordre de l'opération que nous souhaitons éffectuer. Il s'agit d'un dictionnaire python construit de manière spécifique. En effet, comme nous l'avons précisé, nous allons utiliser la bibliothèque metatrader5, et une opération (une requête) s'éffectue en construisant un dictionnaire contenant certaine informations.Plus simplement, il s'agit du "ticket d'entrée" sur le marché, où le ticket contient un bon nombre d'informations sur l'opération voulue. Orders ressemblerai donc à ceci :



- pill2kill est une pill de tâche à terminer. En effet, à chaque fois qu'un bot est instancié et qu'on lui applique la méthode open_buy() ou open_sell(), un thread est lancé. pill2kill est donc l'attribut de l'objet que l'on veut kill.
- open_buy est une méthode lançant en parallèle le programme qui suit.
- process_open_buy() est un thread. Dès l'instant où open_buy est utilisé sur un objet de type bot, lance un thread pouvant ouvrir une position d'achat si les conditions de la stratégie propre au bot sont respectées. Par exemple, la méthode process_open_buy() d'un objet de type StratégieA (enfant de Bot) est différente de celle d'un objet de type StratégieB (enfant de Bot aussi), mais open_buy reste identique pour les deux stratégies. C'est sur cette méthode que la spécificité des différentes statégies est décrite.
- kill() : parcourt pill2kill et kill tous les threads dedans (open_buy() et open_sell()).
- request permet de mettre à niveau le ticket orders.
- update_df permet de mettre à niveau la dataframe (pour faire du trading en direct) .

Ainsi, lorsque l'utilisateur s'identifie et choisie la répartition de son portefeuille en fonction des différentes stratégies proposées, un objet de type BOT est instancié pour chaque stratégie séléctionnée (les constructeurs prennent en paramètre le nom du marché, ainsi qu'un réel représentant le montant alloué à cette stratégie).

Plan des tests d'intégrations :
Le découpage en modules est l'occasion de spécifier les tests d'intégration. Ces tests permettent de vérifier la bonne coordination entre les fonctions lors des différentes interactions. On peut choisir une méthode Top-down, où l'on s'interesse au bon fonctionnement du module main avant de descendre aux indicateurs. Voici une liste non exhaustives des tests d'intégration que l'on effectuera :

1. on vérifie la coordination entre l'interface graphique et le module main : on doit pouvoir se connecter à un compte metatrader5 existant.

2. Ensuite, il faut vérifier que l'on puisse choisir l'allocation du portefeuille via le logiciel. C'est donc la coordination entre le module main et les modules BOT. Pour ce, nous pouvons créer quelques objet BOTS ayant des stratégies toute simple et vérifier que ces objets ont bien été instancié une fois que l'utilisateur a sélectionné ces stratégies par l'interface graphique.

3. Enfin, on vérifie que ces objets peuvent effectuer des opérations (coordination indicateurs/BOT). Pour ce, on peut créer un module BOT n'ayant aucune stratégie et qui ouvre une position d'achat ou de court. Dans le cadre de ce projet nous ne manipulons que de l'argent fictif, donc pour ce test cela nous importe peu de perdre de "l'argent". On vérifie manuellement sur notre compte qu'une opération a bien eu lieu. Pour affiner le test, on peut essayer une statégie toute simple comme le croisement de moyennes mobiles, ce qui nous permet aussi de tester la fonctionnalité de handle_buy().

3.3 Conception détaillée

On peut diviser la conception du projet en 4 grande partie :

- Le backtest (conception et optimisation de la stratégie)
- Indicateurs.py
- bot.py
- main.py (interface graphique)

3.3.1 Indicateurs.py

Indicateurs.py est le fichier le plus simple à comprendre. Il s'agit d'une boîte à outils qui sera appelée dans la conception des stratégies dans bot.py . Nous avons donc codé un fichier indicateurs.py, qui va contenir tous les indicateurs qui seront utilisés dans les stratégies par notre bot. De manière analogue aux autres fichiers, il est nécessaire d'importer les bibliothèques usuelles de Python ainsi que les informations provenant du marché fournis par YahooFinance pour pouvoir construire les différents indicateurs. :

```python
from pyparsing import col
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import MetaTrader5 as mt5
```

Le fichier sera ensuite composé d'un ensemble de fonctions. La première est la ydataframe, qui permet de récupérer une dataframe adapté au trading journalier facilement.

```python
def ydataframe(stock : str, start : str , interval : str ) -> pd.DataFrame :
    # ""
    # permet d'obtenir une dataframe issue de yfinance rapidement sans prise de tête. Le problème est que
    # yfinance fournie des dataframes limitées en terme de données ( pour des interval de 5m, on ne peut
    # pas dépasser 60 jours par exemple). Pour du backtest, il faut importer la data manuellement, mais
    # pour du live trading, cela suffit.
    # ""

    df = yf.download(stock,start = start, interval=interval )
    df.dropna(inplace=True)
    return df
```

Un DataFrame est une structure de données bidimensionnelle, c'est-à-dire que les données sont alignées de façon tabulaire en lignes et en colonnes. Voici la dataframe pour Microsoft, avec un interval journalier avec $start =' 2022-03-14'$

```
[*********************100%***********************]  1 of 1 completed
                 Open        High         Low       Close   Adj Close    Volume
Date
2022-03-14  280.339996  285.399994  275.820007  276.440002  276.440002  30660700
2022-03-15  280.350006  287.820007  278.730011  287.149994  287.149994  34245100
2022-03-16  289.109985  294.570007  283.200012  294.390015  294.390015  37826300
2022-03-17  293.290009  295.609985  289.369995  295.220001  295.220001  30816600
2022-03-18  295.369995  301.000000  292.730011  300.429993  300.429993  43317000
2022-03-21  298.890015  300.140015  294.899994  299.160004  299.160004  28351200
2022-03-22  299.799988  305.000000  298.769989  304.059998  304.059998  27599700
2022-03-23  300.510010  303.230011  297.720001  299.489990  299.489990  25715400
2022-03-24  299.140015  304.200012  298.320007  304.100006  304.100006  24484500
2022-03-25  305.230011  305.500000  299.290009  303.679993  303.679993  22566500
2022-03-28  304.329987  310.799988  304.329987  310.700012  310.700012  29578200
2022-03-29  313.910004  315.820007  309.049988  315.410004  315.410004  30393400
2022-03-30  313.760010  315.950012  311.579987  313.859985  313.859985  28163600
2022-03-31  313.899994  315.140015  307.890015  308.309998  308.309998  33422100
2022-04-01  309.369995  310.130005  305.540009  309.420013  309.420013  27085100
PS C:\Users\aline\OneDrive\Bureau\Pro info>
```

Les autres fonctions de ce fichiers vont tout simplement modifier la dataframe. Par exemple, rajouter une colonne, en supprimer une, ne prendre en compte que certaines lignes etc . . . On peut prendre l'exemple de la fonction $sma(data, length : str, column : str)$ qui rajoute une colonne à la dataframe fournie en argument :

```python
def sma(data,length : str, column : str) -> pd.DataFrame:
    #""
    #rajoute la colonne des sma d'une colonne. Le sma se calcul sur length unités. Renvoie une dataframe.
    #""
    data[str(length) + "SMA_" + column] = data[column].rolling(window=length).mean()
```

Date	Open	High	Low	Close	Adj Close	Volume	20SMA_Close
2022-03-14	280.339996	285.399994	275.820007	276.440002	276.440002	30660700	289.929501
2022-03-15	280.350006	287.820007	278.730011	287.149994	287.149994	34245100	289.537001
2022-03-16	289.109985	294.570007	283.200012	294.390015	294.390015	37826300	289.233002
2022-03-17	293.290009	295.609985	289.369995	295.220001	295.220001	30816600	289.019002
2022-03-18	295.369995	301.000000	292.730011	300.429993	300.429993	43317000	289.504001
2022-03-21	298.890015	300.140015	294.899994	299.160004	299.160004	28351200	290.065501
2022-03-22	299.799988	305.000000	298.769989	304.059998	304.059998	27599700	290.882501
2022-03-23	300.510010	303.230011	297.720001	299.489990	299.489990	25715400	291.843501
2022-03-24	299.140015	304.200012	298.320007	304.100006	304.100006	24484500	292.319002
2022-03-25	305.230011	305.500000	299.290009	303.679993	303.679993	22566500	292.637502
2022-03-28	304.329987	310.799988	304.329987	310.700012	310.700012	29578200	293.233002
2022-03-29	313.910004	315.820007	309.049988	315.410004	315.410004	30393400	294.256001
2022-03-30	313.760010	315.950012	311.579987	313.859985	313.859985	28163600	294.939500
2022-03-31	313.899994	315.140015	307.890015	308.300998	308.300998	33422100	295.559000
2022-04-01	309.369995	310.130005	305.540009	309.420013	309.420013	27085100	296.537001

Le fichier indicateurs.py comporte également deux fonctions permettant de gérer les ordres de vente/achat. On utilise pour cela un dictionnaire orders : Pour ouvrir un ordre, il faut fournir un dictionnaire orders qui ne contient pas de key ≪position≫. S'il en contient une, on utilisera la fonction removekey pour la supprimer.

```python
def removekey(orders) :
    #""
    # Permet de supprimer la key "position" de orders. Pour ouvrir un ordre, il faut fournir un dictionnaire orders (qui ne contient pas
    # de key "position"). Or si orders contient une key 'position', et bien il faut l'enlever pour pouvoir passer la transaction.
    #""
    orders.pop("position")
    return orders
```

Au contraire si l'on souhaite fermer une position, il faut une key ≪position≫. Si l'order ne la contient pas on l'ajoute manuellement avec addkey :

```python
def addkey(orders, position : int) :
    #""
    # Permet d'ajouter la key "position" de orders. Pour fermer une position, il faut fournir un dictionnaire orders (qui contient
    # une key "position"). Or si orders ne contient pas une key 'position', et bien il faut l'ajouter pour pouvoir passer la transaction.
    #""
    orders['position'] = position
    return position
```

Dans une optique d'amélioration et de pluralité des stratégies de trading proposés, d'autres indicateurs vont ou ont déjà été programmé.

Finalement, une dernière fonction money_to_volume permet de palier le problème de convertibilité des actifs en volume. Mt5 ne comprend que le ≪volume≫ pour connaître le montant à placer dans une opération, or le ≪volume≫ dépend du prix de l'actif mis en jeu. Dans un souci d'efficacité, si l'on souhaite placer une certaine somme d'argent sans avoir à vérifier combien cela représente en volume de l'actif, on utilisera cette fonction.

```python
def money_to_volume(market: str, money : float) -> float :
    #""
    # Permet de convertir un nombre float représentant l'argent que l'on souhaite placer dans un actif en volume. Dans orders, Mt5 ne comprend
    # que 'volume' pour connaître le montant à placer dans une opération, or volume dépend du prix de l'actif mis en jeu. Si on veut placer
    # 2000 euros sans se soucier de combien ça représente en volume de l'actif, on utilise cette fonction.
    #""
    prix_1market = mt5.symbol_info_tick(market).ask
    return round(money/prix_1market,2)
```

On a de même les fonctions ema,std, quantile,MACD,SAR, RSI qui sont des fonctions directement implémentées sur Python. On se contentera donc simplement de rappeler leur définition :

- Ema (Exponential Moving Average) : Celle-ci ne fait que calculer une moyenne mobile en pondérant d'avantage les derniers termes afin de leur donner d'avantage d'importance.
- Quantile : Comme son nom l'indique, cette fonction rajoute la colonne des quantiles d'une colonne. Le quantile se calcul sur une length unité.
- MACD(Moving Average convergence Divergence) : Celui-ci représente l'écart des moyennes mobiles aux cours. Sa courbe se trace sur le graphique de l'évolution du cours.
- RSI (Relative Strength Index) : C'est un indicateur de suivi de vitesse des prix qui, borné entre 0 et 100%, permet de déterminer la force interne d'une valeur dans le temps.
- SAR (Stop And Reversal Indicateur) : Indicateur matérialisé par une série de points sur un graphique. Ceux-ci indiquent des retournements de tendance : lorsque le cours rencontre un point SAR, on doit s'attendre à un arrêt puis à un renversement de la tendance antérieure.

Voici les codes pour les trois derniers points par exemple :

```
125  def RSI(data,length) -> pd.DataFrame :
126      #""
127      # Ajoute une colonne RSI.
128      #""
129      data['RSI'] = talib.RSI(data['Close'], timeperiod=length)
130      return data
131
132  def SAR(data)->pd.DataFrame :
133      #""
134      # Ajoute une colonne SAR.
135      #""
136      data['SAR'] = talib.SAR(data['High'], data['Low'], acceleration=0.02, maximum=0.2)
137      return data
138
139  def MACD(df) -> pd.DataFrame :
140      #""
141      # Ajoute une colonne MACD, Signal, et l'historigramme.
142      #""
143      df['MACD'],df['Signal'],df['Hist'] = talib.MACD(df['Close'], fastperiod=12, slowperiod=26, signalperiod=9)
144      return df
```

La fonction suivante, qui nous fournie le Z-score, un indicateur qui sera primordial pour une des stratégies fournies portant le même nom :

```
55  def zscore(data, length : int, column : str) -> pd.DataFrame:
56      #""
57      #rajoute la colonne des zscore d'une colonne. Le zscore se calcul sur le length unité. Renvoie une datafram
58      #Le zscore se calcul en trois étapes
59      #""
60      sma(data,length, column)
61      displacement = data[column] - data[str(length) + "SMA_" + column]
62      std(data, length, column)
63      data[str(length) + "Zscore_" + column] = displacement.divide(data[str(length) + "STD_" + column])
64      return data
65
```

Cet indicateur exprime le nombre d'écarts-types qui se trouvent au-dessus ou en dessous de lamoyenne. Pour cela on a besoin de faire appel à la fonction «sma» qui nous calcul la moyenne mobile de nos données ainsi qu'à la fonction «std» (standard deviation) qui nous fournit les écarts- types.

3.3.2 bot.py

bot.py est le fichier regroupant les différentes stratégies sous formes de classes. nous allons importer les librairies ainsi que les fichiers suivants :

```
import threading
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import Indicateurs as ind
import MetaTrader5 as mt5
from datetime import datetime
from threading import Thread
import time
```

Par rapport à Indicateurs.py, on dénote l'importation de la librairie «threading» et du module «time» (nous permettant d'avoir accès à la date et heure à tout instant) ainsi que du fichier Indicateurs.py lui-même puisqu'il sera utilisé dans les différentes stratégies. Le module «threading» quant à lui, élabore des interfaces haut-niveau de fils d'exécutions multiples (threading) conçues en s'appuyant sur le module _thread ce qui va permettre d'interagir avec le programme même lorsque celui-ci est en train d'exécuter une autre tâche.

Comme nous l'avons dit dans le premier livrable, l'objectif était dans un premier temps de mettre en place une stratégie (donc une classe) simple. Nous avons choisi la stratégie des croisements des 3 emas. La stratégie est aussi expliquée dans le premier livrable dans l'introduction.
Le concept est le suivant : lorsque l'utilisateur choisit une stratégie, il instancie un objet de la classe associé à la stratégie. Dans l'instanciation, il doit préciser : Le montant qu'il souhaite alloué, le marché en précisant le symbole MetaTrader5 et celui de yahoo finance (par exemple BTCUSD et BTC-USD). Voici le constructeur pour la classe 3 ema :

```python
def __init__(self, mt5symbol : str, volume : float, ysymbol :str) -> None:
    #""
    #Constructeur de la classe. On initialise le champ orders qui est le ticket d'entrée. On ne
    #spécifie que le symbol et le volume. Attention pour le symbol, il faut spécifier celui de
    # MetaTrader5 et celui de yfinance !
    #""
    self.mt5symbol = mt5symbol
    self.ysymbol = ysymbol
    self.orders = {
    "action" : "a"  ,
    "symbol" : mt5symbol ,
    "volume" : volume ,
    "type" : "a" ,
    "price" : "a",
    "sl" : 0.0 ,
    "tp" : 0.0 ,
    "deviation" : 20 ,
    "magic" : 234000 ,
    "comment" : "test",
    "type_time" : mt5.ORDER_TIME_GTC,
    "type filling" : mt5.ORDER_FILLING_IOC,
    }
    self.position_ouverte = False
    # determine la position dans laquelle où nous sommes : a-t-on une
    # position ouverte ? False signifie que nous n'avons pas de position ouverte : on peut procéder
    # à une opération.
    self.df = ind.ydataframe(stock = ysymbol, start= '2022-03-14', interval='1m')
    self.df = ind.ema(self.df,length=20,column='Close')
    self.df = ind.ema(self.df,length=5,column='Close')
    self.df = ind.ema(self.df,length=60,column='Close')
    # cette série d'opération sur self.df permet d'actualiser la dataframe (base de donnée) self.df en ajoutant
    # les colonnes 20ema, 5ema, 60ema
    self.pill2kill = []
    # self.pill2kill contient la liste des threads à terminer. Pour cette exemple, il n'y aura que
    # proccess_open_buy à terminer.
    self.dead = True
    # self.dead sera utiliser plus tard pour terminer proccess_open_buy à terminer grâce à la méthode kill.
```

On ne renseigne pour l'instant que les symboles en faisant bien attention à renseigner à la fois le symbole de yfinance et celui de MetaTrader5 pour s'assurer de les faire coïncider ainsi que le volume. Il est important de s'assurer qu'aucune position n'est déjà ouverte sur un même ordre pour une même stratégie pour ne pas en lancer un autre ce qui explique la dernière ligne du code ci- dessus (variable booléenne initialisée à False). Deux autres méthodes permettent de mettre à jour le ticket $self.orders$ et $self.df$ car en effet, ces deux variables ne cessent d'être modifiées tant que la stratégie est en place :

```python
def request(self,action,type,price,sl,tp,comment,position_ouverte : bool) -> None :
    #""
    #méthode pour update orders de l'objet. On update aussi position_ouverte
    #""
    self.orders['action'] = action
    self.orders['type'] = type
    self.orders['price'] = price
    self.orders['sl'] = sl
    self.orders['tp'] = tp
    self.orders['comment'] = comment
    self.position_ouverte = position_ouverte

def update_df(self) :
    #""
    # Cette méthode permet de mettre à jour self.df, ce qui est nécessaire pour faire du trading en direct.
    #""
    self.df = ind.ydataframe(stock = self.ysymbol, start= '2022-03-14', interval='1m')
    self.df = ind.ema(self.df,length=20,column='Close')
    self.df = ind.ema(self.df,length=5,column='Close')
    self.df = ind.ema(self.df,length=60,column='Close')
```

Enfin, voici la méthode principal : $process\_open\_buy(self)$. Comme indiqué dans le livrable 1, notre projet fait de la programmation en parallèle : ceci

permet de faire tourner plusieurs stratégie à la fois. $process\_open\_buy(self)$ est le programme lancé en parallèle (le "thread") qui cherche sans arrêt une opportunité d'achat. le thread $process\_open\_buy(self)$ ne s'arrête qu'avec la méthode $kill(self)$, qui change $self.dead$ en $False$ ce qui a pour conséquence d'arrêter la boucle de $process\_open\_buy(self)$.

```
def process_open_buy(self) :
    #""
    # Il s'agit de la méthode qui permet l'achat et la vente de l'actif selon la stratégie. La stratégie est simple :
    # Si la courbe de la moyenne mobile exponentielle de 5 jours (5ema) est au dessus de celle de 20ema, et que cette
    # dernière est au dessus de 60ema, alors On achète. Si on a 5ema<20ema<60ema, on vend. Pour procéder à une
    # opération d'achat il faut d'abord qu'il y ait eu un achat avant (c'est le rôle de self.position_ouverte)
    #""
    while self.dead :
        n=len(self.df) - 1
        if self.df['5EMA_Close'][n] > self.df['20EMA_Close'][n] and self.df['20EMA_Close'][n] > self.df['60EMA_Close'][n] and  self.position_ouverte == False :
            # Les conditions d'achat sont respectées, et la dernière opération étaint une vente. Il n'y a donc pas de position ouverte.
            prix = mt5.symbol_info_tick(self.mt5symbol).ask
            self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_BUY, price = prix, sl = 0.0,tp=0.0, comment = "call",  position_ouverte= True )
            if 'position' in self.orders.keys() :
                # pour pouvoir acheter, il faut un ticket orders (dictionnaire) ne contenant pas de clé 'position'.
                # En effet, pour vendre, il faut une clé en plus que pour le ticket d'achat, position, qui permet
                # d'indiquer l'opération que l'on souhaite modifier. Ici, vu que l'on souhaite acheter, on vérifie
                # que self.orders a le bon format.
                ind.removekey(self.orders)
                mt5.order_send(self.orders)
            else :
                # self.orders a le bon format.
                mt5.order_send(self.orders)
                print(self.orders)
        elif self.df['5EMA_Close'][n] < self.df['20EMA_Close'][n] and self.df['20EMA_Close'][n] < self.df['60EMA_Close'][n] and  self.position_ouverte == True :
            # Les conditions de ventes sont réunies.
            prix = mt5.symbol_info_tick(self.mt5symbol).bid
            self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_SELL, price = prix, sl = 0.0,tp=0.0, comment = "sell",  position_ouverte = False )
            position = mt5.positions_get()[-1].ticket
            # On détermine la position de l'opération que l'on souhaite clôturer.
            ind.addkey(self.orders, position=position)
            mt5.order_send(self.orders)
        self.update_df()
        time.sleep(10)
```

Les deux prochaines méthodes sont expliquées de manière explicite avec les commentaires du code :

```
def open_buy(self) :
    #""
    # Cette méthode permet de lancer en parallèle (faire un thread) la méthode process_open_buy(). Dès
    # qu'un objet de classe Trois Ema est instancié, on lui applique cette méthode pour lancer l'algorithme de
    # trading automatique.
    # ""
    thread = threading.Thread(target= self.process_open_buy) #lancement de la programation en parallèle.
    self.pill2kill.append(thread)
    thread.start()

def kill(self) :
    #""
    # Cette méthode permet d'ârreter les opérations d'achat et de vente, en arrêtant proccess_open_buy()
    #""
    self.dead = False
    self.pill2kill[0].join()
```

Voici un exemple d'instanciation et de lancement de la stratégie :

```
bot1 = bot.TroisMA(mt5symbol="BTCUSD",volume = 0.03,ysymbol="BTC-USD")
bot1.open_buy()
```

Et un exemple de son arrêt :

```
bot.TroisMA.kill(bot1)
```

Ce qui précède ce généralise pour les autres stratégies. En effet, la conception des autres classes (stratégies) est très similaires à celle de la première, avec de légères différences.

Par exemple, prenons la classe Zscore. Le Zscore à un instant T est simplement la distance entre la moyenne et le cours en nombre d'écart type.

```
98      def process_open_buy(self) :
99          c=1
100         while self.dead :
101             print(c)
102             c+=1
103             n=len(self.df) - 1
104             print(self.df['5EMA_Close'][n] > self.df['200EMA_Close'][n] and self.df['20EMA_Close'][n] > self.df['60EMA_Close'][n] and  self.position_ouverte == False )
105             print(self.df['5EMA_Close'][n] < self.df['200EMA_Close'][n] and self.df['20EMA_Close'][n]<self.df['60EMA_Close'][n] and  self.position_ouverte == True)
106             if self.df['5EMA_Close'][n] > self.df['200EMA_Close'][n] and self.df['20EMA_Close'][n] > self.df['60EMA_Close'][n] and  self.position_ouverte == False :
107                 prix = mt5.symbol_info_tick(self.mt5symbol).ask
108                 print("ok")
109                 self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_BUY, price = prix, sl = 0.0,tp=0.0, comment = "call",  position_ouverte= True )
110                 if 'position' in self.orders.keys() :
111                     ind.removekey(self.orders)
112                     mt5.order_send(self.orders)
113                 else :
114                     mt5.order_send(self.orders)
115                     print(self.orders)
116                 self.position = mt5.positions_get()[-1].ticket # On détermine la position de l'opération que l'on souhaite clôturer plus tard.
117             elif self.df['5EMA_Close'][n] < self.df['200EMA_Close'][n] and self.df['20EMA_Close'][n] < self.df['60EMA_Close'][n] and  self.position_ouverte == True :
118                 prix = mt5.symbol_info_tick(self.mt5symbol).bid
119                 self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_SELL, price = prix, sl = 0.0,tp=0.0, comment = "sell",  position_ouverte = False )
120                 ind.addkey(self.orders, position=self.position)
121                 mt5.order_send(self.orders)
122             print(self.df['5EMA_Close'][n])
123             print(self.df['200EMA_Close'][n])
124             print(self.df['60EMA_Close'][n])
125             self.update_df()
126             time.sleep(10)
127
```

La différence provient du fait que nous considérons maintenant un stop loss (limite de perte) et un take profit (limite de gain).

La dernière class présente une autre différence par rapport aux autres : Pour la classe PSAR_MCAD, Stratégie du PSAR + MACD (cf livrable section annexe) + 200Ema, il s'agit d'acheter uniquement quand les conditions décrites dans le livrable (PSAR + MACD) sont réalisées et que les prix sont au-dessus de la courbe des 200Ema. On parie sur la baisse pour le reste. Donc pour la méthode _init_ de la classe PSAR_MCAD, il y a une légère différence avec les classes précédentes. On a deux "self.position_ouverte", car avant on ne faisait que des paries à la hausse (achat) uniquement ou que des paries à la baisse (short) uniquement. Mais la logique reste la même : on ne fait aucune opération tant qu'une opération est en cours cad qu'on engage une nouvelle opération (long ou short) que si self.position_ouverte_bull et self.position_ouverte_bear sont égaux à False.

```
    def process_open_buy(self) :
        while self.dead :
            ask = mt5.symbol_info_tick(self.mt5symbol).ask
            bid = mt5.symbol_info_tick(self.mt5symbol).bid
            n=len(self.df) - 1

            if  self.df['Close'][n] > self.df['200EMA_Close'][n] and self.position_ouverte_bull == False and self.position_ouverte_bear == False and self.df['Hist'][n] >
                print("Signal d'ouverture de position")
                prix = ask
                self.sl = (self.df['SAR'][n] + self.df['Close'][n])/2
                self.tp = self.df['Close'][n] + (self.df['Close'][n] - self.df['SAR'][n])
                self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_BUY, price = prix, sl = 0.0,tp=0.0, comment = "long")
                self.position_ouverte_bull = True
                if 'position' in self.orders.keys() :
                    ind.removekey(self.orders)
                    mt5.order_send(self.orders)
                else :
                    mt5.order_send(self.orders)
                    print(self.orders)
                time.sleep(2)
                self.position = mt5.positions_get()[-1].ticket # On détermine la position de l'opération que l'on souhaite clôturer plus tard.

            elif bid < self.sl and  self.position_ouverte_bull == True :
                print("Clôture par SL")
                prix = bid
                self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_SELL, price = prix, sl = 0.0,tp=0.0, comment = "Clôture par SL")
                self.position_ouverte_bull = False
                ind.addkey(self.orders, position=self.position)
                mt5.order_send(self.orders)

            elif bid > self.tp and  self.position_ouverte_bull == True :
                print("Clôture par TP")
                prix = bid
                self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_SELL, price = prix, sl = 0.0,tp=0.0, comment = "Clôture par TP")
                self.position_ouverte_bull = False
                ind.addkey(self.orders, position=self.position)
                mt5.order_send(self.orders)

            # partie short
        # partie short
        if  self.df['Close'][n] < self.df['200EMA_Close'][n]  and self.df['Hist'][n] < 0 and self.df['SAR'][n] > self.df['Close'][n] and self.df['SAR'][n-1] < self.df
            print("Signal d'ouverture de position short")
            prix = bid
            self.sl = (self.df['SAR'][n] + self.df['Close'][n])/2
            self.tp = self.df['Close'][n] + (self.df['Close'][n] - self.df['SAR'][n])
            self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_SELL, price = prix, sl = 0.0,tp=0.0, comment = "short")
            self.position_ouverte_bear = True
            if 'position' in self.orders.keys() :
                ind.removekey(self.orders)
                mt5.order_send(self.orders)
            else :
                mt5.order_send(self.orders)
                print(self.orders)
            time.sleep(2)
            self.position = mt5.positions_get()[-1].ticket # On détermine la position de l'opération que l'on souhaite clôturer plus tard.

        elif self.position_ouverte_bear == True and ask < self.tp :
            print("Clôture par TP")
            prix = ask
            self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_BUY, price = prix, sl = 0.0,tp=0.0, comment = "Clôture par TP en short")
            self.position_ouverte_bear = False
            ind.addkey(self.orders, position=self.position)
            mt5.order_send(self.orders)

        elif ask >= self.sl and  self.position_ouverte_bull == True :
            print("Clôture par SL")
            prix = ask
            self.request(action = mt5.TRADE_ACTION_DEAL, type = mt5.ORDER_TYPE_BUY, price = prix, sl = 0.0,tp=0.0, comment = "Clôture par SL")
            self.position_ouverte_bear = False
            ind.addkey(self.orders, position=self.position)
            mt5.order_send(self.orders)

        self.update_df()
        time.sleep(10)
```

3.3.3 main.py

Notre bot de trading utilise MetaTrader5 afin de pouvoir fonctionner (acheter/vendre des actions). Cependant afin de rendre notre bot plus pratique à l'utilisation, nous avons programmé une interface graphique ayant plusieurs utilités :

- Créer un nouveau bot en sélectionnant certains critères ainsi qu'une stratégie.
- Avoir une interface de gestion des bots déjà créés
- Obtenir des informations sur les différentes stratégies utilisées

Pour cela, nous avons utilisé le module Tkinter de Python.

```
from tkinter import *
```

Après importation, on implémente une classe ainsi que la racine de notre application, sur laquelle on construit les différentes frames que l'on va utiliser.

```
class Myapp :
    def __init__(self):
        self.root = Tk()
        self.root.title("TEST")
        self.root.minsize(480,380)
        self.root.geometry("1080x690")
        self.root.config(background="#4A4A4A")
        self.bot_frame = Frame(self.root,bg="#4A4A4A")
        self.gestion_bot_frame = Frame(self.root,bg="#4A4A4A")
        self.menu_bar()
        self.pill2kill = []
```

Dans la suite, cette unique classe contiendra toutes les fonctions qui nous seront utiles. Afin de pouvoir circuler sur notre application, une fonction menue permettra d'accéder aux onglets bot, stratégie et aide.

```
def menu_bar(self):
    nav_bar = Menu(self.root)
    nav_bar.add_command(label = "Acceuil")
    file_menu = Menu(nav_bar,tearoff = 0)
    file_menu.add_command(label="Stratégie A", command= self.openWindowStrategyA)
    file_menu.add_command(label = "Stratégie B", command= self.openWindowStrategyB)
    file_menu.add_command(label = "Stratégie C", command= self.openWindowStrategyC)
    nav_bar.add_cascade(label = "Stratégies", menu = file_menu)
    menu_bot = Menu(nav_bar, tearoff = 0)
    menu_bot.add_command(label = "Créer un nouveau bot", command = self.bot_frame)
    menu_bot.add_command(label = "Gestion des bots", command = self.gestion_bot_frame)
    nav_bar.add_cascade(label = "Bots", menu= menu_bot)
    menu_help = Menu(nav_bar, tearoff = 0)
    menu_help.add_command(label = "Comment se servir du logiciel ?", command = self.openWindowHelp)
    nav_bar.add_cascade(label = "Help", menu = menu_help)
    nav_bar.add_command(label = "Déconnexion")
    self.root.config(menu = nav_bar)
```

Les plus importantes seront les fonctions frame, qui afficheront une nouvelle page permettant l'intéraction avec le back.

La fonction bot_frame permet de sélectionner notre actif, le montant ainsi que la stratégie utilisée, elle permet dans le même temps de supprimer un bot déjà créé.

```
def bot_frame(self) :
    # Création du texte et des boutons
    label_title1 = Label(self.bot_frame,text="Veuillez choisir une ou plusieurs stratégies. \n Pour obtenir plus d'informations sur les stratégies mis en place, allez dans l'onglet Stratégies ",
    label_strat = Label(self.bot_frame,text="Stratégie", font=("Courrier", 10), bg = "#252424", fg = 'white')
    liste_strat= ["Choisir une stratégie", "Stratégie A", "Stratégie B", "Stratégie C"]
    listeCombo = ttk.Combobox(self.bot_frame, value=liste_strat)
    listeCombo.current(0)
    label_vol = Label(self.bot_frame,text="Montant", font=("Courrier", 10), bg = "#252424", fg = 'white')
    volEntry = Entry(self.bot_frame,width= 25)
    label_actif = Label(self.bot_frame,text="L'actif", font=("Courrier", 10), bg = "#252424", fg = 'white')
    liste_actifs ["Choisir un actif", "BTC-EUR", "ETH-EUR"]
    listeCombo2 = ttk.Combobox(self.bot_frame, value=liste_actif)
    listeCombo2.current(0)
    runButton = Button(self.bot_frame,text = "RUN")
    label_title2 = Label(self.bot_frame,text="Veuillez choisir un bot actif que vous voulez arrêter (STOP ALL termine tous les bots) :", font =("Courrier",15),bg= "#4A4A4A",fgs'white' )
    label_tokill = Label(self.bot_frame, text="Indiquez la stratégie que vous voulez arrêter",font="Courrier", 10), bg = "#252424", fg = 'white')
    listeComboToKill = ttk.Combobox(self.bot_frame, value=liste_strat)
    listeComboToKill.current(0)
    kill_allButton = Button(self.bot_frame,text = "KILL ALL")
    killButton = Button(self.bot_frame,text = "KILL")
    # Affichage de la frame (empaquettage)
    label_title1.pack(side = TOP)
    label_strat.pack(pady=5)
    listeCombo.pack(pady=5)
    label_vol.pack(pady=5)
    volEntry.pack(pady=5)
    label_actif.pack(pady=5)
    listeCombo2.pack(pady=5)
    runButton.pack(pady=10)
    label_title2.pack(pady=10)
    label_tokill.pack(pady=5)
    listeComboToKill.pack(pady=5)
    killButton.pack(pady=5)
    kill_allButton.pack(pady=5)
    self.bot_frame.pack(expand = YES)
```

Cette fonction contient l'ensemble des éléments qui seront affichés : blocs de texte, menu de sélection ou bouton d'activation. Dans la suite, l'ensemble des fonctions contiendront le même type d'élément, nous ne les détaillerons pas dans ce livrable mais l'ensemble des commentaires expliquant leur utilité seront présent sur le code final.

Une autre fonction permet d'afficher la frame de gestion des bots, un tableau indiquant les bots activés.

```
def gestion_bot_frame(self):
```

La fonction connexion_frame permet de connecter le logiciel avec MetaTrader5, afin de pouvoir démarrer une session de trading.

```
def connexion_frame(self) :
```

Un ensemble de fonctions sont également implémentées afin de rendre fonctionnel l'interface :

- Hide_frame pour changer de frame
- openWindowStrategy pour ouvrir les informations liées aux différentes stratégies et openWindowHelp pour ouvrir un onglet d'aide
- usr_login pour assurer la connexion avec MetaTrader5
- run_strat pour créer un nouveau bot
- kill pour supprimer un bot et kill_all pour tous les supprimer

```
def openWindowHelp(self):
    #Fonction qui va nous permettre d'ouvrir une page avec les différentes instructions pour se servir de notre logiciel
    newWindow= Toplevel(self.root) #création d'une nouvelle page par dessus celle désignée root
    newWindow.title("Menu d'aide")
    newWindow.geometry("1080x720")
    newWindow.config(background="#4A4A4A") #configuration de la nouvelle page
    newWindow.minsize(720,480)
    help_title= Label(newWindow,text="Tutoriel pour se servir de ce logiciel de trading", font =("Courrier",15),bg= "#4A4A4A",fg='white')
    help_title.pack(expand=YES)
    help_txt= Label(newWindow, text="Votre compte MetaTrader5 est directement connecté à ce logiciel, \n pour démarrer votre session il vous suffit
    help_txt.pack(expand=YES) #les informations liées seront ainsi ajoutées à la page

#les stratégies suivantes permettent d'afficher les informations liées aux stratégies. Elles sont assez répétitives mais il apparaissait
#plus simple pour la compréhension du code de faire plusieurs fois la même fonction similaire plutôt qu'une seule fonction

def openWindowStrategyA(self):
    newWindowA= Toplevel(self.root)
    newWindowA.title("Stratégie A")
    newWindowA.geometry("1080x720")
    newWindowA.config(background="#4A4A4A")
    newWindowA.minsize(720,480)
    strat_A= Label(newWindowA,text="Comment fonctionne la stratégie A ?", font =("Courrier",15),bg= "#4A4A4A",fg='white')
    strat_A.pack(expand=YES)

def openWindowStrategyB(self):
    newWindowB= Toplevel(self.root)
    newWindowB.title("Stratégie B")
    newWindowB.geometry("1080x720")
    newWindowB.config(background="#4A4A4A")
    newWindowB.minsize(720,480)
    strat_A= Label(newWindowB,text="Comment fonctionne la stratégie B ?", font =("Courrier",15),bg= "#4A4A4A",fg='white')
    strat_A.pack(expand=YES)

def openWindowStrategyC(self):
    newWindowC= Toplevel(self.root)
    newWindowC.title("Stratégie C")
    newWindowC.geometry("1080x720")
    newWindowC.config(background="#4A4A4A")
    newWindowC.minsize(720,480)
    strat_A= Label(newWindowC,text="Comment fonctionne la stratégie C ?", font =("Courrier",15),bg= "#4A4A4A",fg='white')
    strat_A.pack(expand=YES)
```

L'ensemble de cette classe rend ergonomique l'utilisation du bot pour l'utilisateur.

3.4 Codage

3.5 Tests unitaires

3.6 Tests d'intégration

3.7 Tests de validation

4 Intérpretation des résultats

5 Manuel utilisateur

6 Conclusion

7 Annexes

7.1 Gestion de projet

7.1.1 Plan de charge

Exemple à adapter à chaque projet

PLAN DE CHARGES PRÉVISIONNEL

Description de l'activité	Charge en %	Charge en H	Charge en H / Participant				
			Bitan	Alinejad	Fülöp	Vol	Khedimi
Total	**100%**	**281**	**56**	**57**	**56**	**56**	**56**
Gestion de projets	**24%**	**67**	**15**	**15**	**13**	**12**	**12**
Réunion de lancement	2%	6,25	1,25	1,25	1,25	1,25	1,25
Planning prévisionnel et Suivi d'activités	1%	3,75	0,75	0,75	0,75	0,75	0,75
Réunions de suivi	11%	30	6	6	6	6	6
Rédaction	7%	20	4	4	4	4	4
Outils collaboratifs (svn, etc.)	2%	7	3	3	1	0	0
Spécification	**5%**	**14**	**4**	**2**	**2**	**3**	**3**
Définition des fonctionnalités	5%	14	4	2	2	3	3
Conception préliminaire	**12%**	**35**	**6**	**9**	**6**	**7**	**7**
Définition d'un modèle de données	3%	8	1	2	1	2	2
Définition d'un format de fichiers associé au modèle de données	2%	5	1	1	1	1	1
Définition des fonctionnalités	6%	17	3	5	3	3	3
Définition des modules	2%	5	1	1	1	1	1
Conception détaillée	**27%**	**76**	**14**	**14**	**16**	**16**	**16**
Définition des classes	2%	7	2	1	1	1	2
Définiton des méthodes	7%	19	3	4	5	4	3
Définition des tests unitaires	7%	20	4	4	4	4	4
Auto-formation	7%	20	3	3	4	5	5
Maquettage des interfaces	4%	10	2	2	2	2	2
Codage	**17%**	**48**	**9**	**9**	**11**	**10**	**9**
Codage des classes	2%	5	1	1	1	1	1
Codage des méthodes	11%	31	6	6	7	6	6
Codage des tests unitaires	4%	12	2	2	3	3	2
Intégration	**6%**	**16**	**3**	**3**	**3**	**3**	**4**
Intégration des modules	4%	11	2	2	2	2	3
Tests d'intégration	2%	5	1	1	1	1	1

Soutenance	9%	25	5	5	5	5	5
Préparation de la soutenance	7%	20	4	4	4	4	4
Soutenance	2%	5	1	1	1	1	1

7.1.2 Planning prévisionnel

Nous avons décider d'utiliser trello pour le planning prévisionnel, voici le lien : https://trello.com/b/J8NwstmW/projet-info

7.1.3 Suivi d'activités

7.2 Stratégies

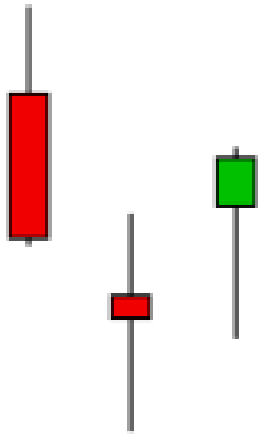
Dans cet Annexe nous fournissons une liste des stratégies que nous allons utiliser.

7.3 Configuration de bougies japonaise

les bougies japonaises (ou chandeliers japonais ou candlesticks) se sont imposés comme la représentation la plus claire et riche en informations. Ces figures reflètent la psychologie des investisseurs, acheteurs comme vendeurs, ainsi que les rapports de force qui les opposent. Associées à d'autres indicateurs, les bougies japonaises permettent d'identifier des tendances de marché et d'anticiper des retournements au sein de celles-ci. Donnons l'exemple de certaines de ces configuartions :

L'étoile du matin :

L'étoile du matin est située en bas d'une tendance baissière et annonce un renversement de tendance à la hausse. Cette figure est constituée de trois bougies : une grande bougie rouge, qui exprime la force des vendeurs, une deuxième bougie rouge de petite taille, qui annonce l'épuisement de la force vendeuse et une troisième et dernière bougie, verte celle-ci, qui confirme le retour d'une force acheteuse sur le titre.



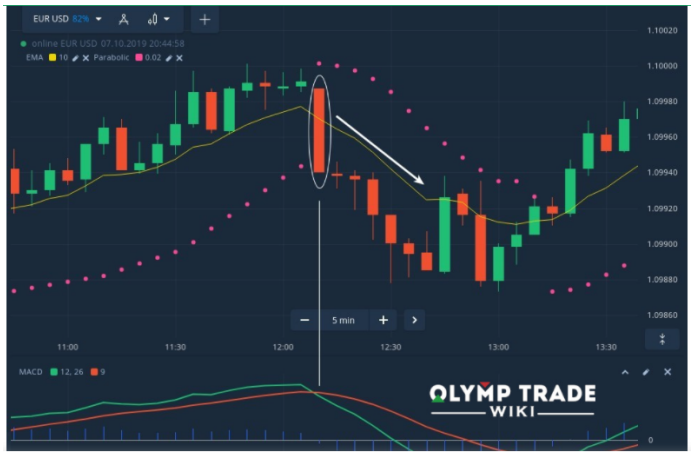
Voici un exemple de cette configuration. La première bougie rouge, à grand corps, signale la force omniprésente des vendeurs. La bougie verte qui suit, à petit corps, annonce l'épuisement de la force vendeuse précédente. La troisième bougie verte, à petit corps, annonce le retour des acheteurs sur la valeur. La hauteur du corps correspond à la force acheteuse. Il est à noter qu'une configuration homologue pour un retournement à la baisse : c'est l'étoile du soir.

Il en existe bien d'autre (une trentaine si ce n'est plus). L'idée est donc de pouvoir détecter ces configurations et agir de manière adéquate. Cependant, Il y a beaucoupe de faux signaux si on s'arrête ici, rendant le rendement vraiment médiocre. **On a besoin de filtrer ces signaux**. Pour ce, nous pouvons utiliser le RSI: Si on détecte un signal, nous regardons immédiatement le RSI pour savoir si celui-ci est en zone de sur-achat ou de survente. Si ce n'est pas le cas, nous ignorons le signal. Sinon, on le considère.

7.3.1 Le parabolique SAR

Tout d'abord, voyons comment lire l'indicateur SAR parabolique. Si l'indicateur se déplace en dessous des prix, la tendance est à la hausse et vous devez entrer une position d'achat. Si elle dépasse les prix, la tendance est à la baisse et il faut entrer dans une position de vente.Notre objectif devrait être de négocier en utilisant le SAR parabolique lors des inversions de tendance. De cette façon, vous entrerez tôt sur le marché avant que la tendance ne s'épuise et finisse par s'inverser.
Maintenant, lorsque la tendance est à la hausse et que l'indicateur se déplace en dessous, attendez que l'indicateur croise le prix et casse. Il commencera alors à se déplacer au-dessus des prix. Lorsque la tendance est à la baisse et que l'indicateur se déplace au-dessus des prix, attendez que l'indicateur croise et casse le prix. Il est temps d'entrer dans une position de vente longue.

Maintenant, cet indicateur pris comme tel fourni lui aussi beaucoup de faux signaux. On utilise la $MACD$ cette fois en guise de filtre : Par exemple si vous envisagez de prendre une position à la baisse on regarde si la ligne verte de la $MACD$ croise la ligne rouge et se déplace en dessous. Ensuite, il faut que le parabolique SAR s'arrête, puis casse et se déplace sur le prix :



7.3.2 Stratégies basées sur les données On-chain

Le Bitcoin pourcent supply in profit : Cette donnée indique quel pourcentage de bitcoin dont le prix d'achat est supérieur au prix du cours actuel. Lorsque l'on superpose cette donnée au prix, on remarque que les points de retournement du marché à la hausse se trouvent lorsque le "Bitcoin pourcent supply in profit" est inférieur à 50% ceci engagerait alors un signal d'achat. A l'inverse lorsque le "Bitcoin pourcent supply in profit" est supérieur à 90% on peut estimer qu'on se trouve sur un top long terme du marché et que l'on peut alors avoir un retournement baissier. Ci joint deux exemples de retournement de la tendance :

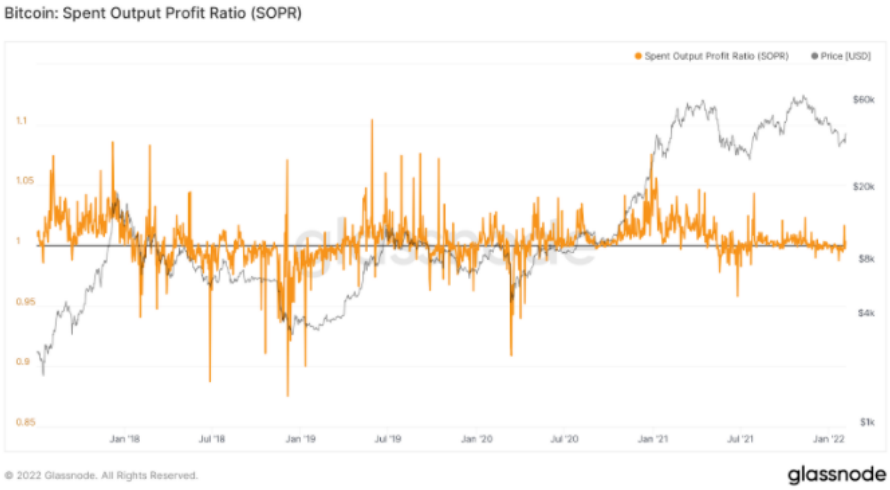
Au niveau du code, l'idée est de créer une fonction qui permet de chercher la donnée "Bitcoin pourcent supply in profit" qui renverrait la donnée de la journée actuelle. Comme c'est une stratégie long terme, on appelle cette fonction une seule fois par jour. On peut imaginer une stratégie centrée uniquement sur le Bitcoin pourcent supply in profit qui en fonction des niveaux détaillés précédemment donnerait un signal d'achat, de vente ou un signal qui indique qu'il n'y a rien à faire.

Net realise profit/loss : Le Net Realized Profit/Loss est le profit ou les pertes net de l'ensemble de coin qui ont été vendu et acheté par jour. Il est défini par la différence du Profit réalisé - Les pertes réalisées.

On remarque que sous la bar rouge qui correspond à des pertes nets de 600 millions de dollars sur la journée, on se situe à un bas du marché moyen terme. A l'inverse au-dessus de la barre verte on trouve des hauts

du cours. Néanmoins ces hauts ne sont pas assez précis pour envisager un achat (trop nombreux et signaux peu clairs). Une stratégie alors possible est alors de faire un achat lorsque les pertes nets réalisées sont de 600 millions ou plus et de clôturer le trade lorsque que les gains nets réalisés dans la journée sont supérieurs à 1 milliard de dollars. On peut aussi envisager une stratégie en corrélant les Net realise profit/loss avec le SOPR qui est indicateur que nous détaillerons dans la suite.

Le Spent Output Profit Ratio (SOPR) : est calculé en divisant le prix de vente du coin / le prix payé pour le coin.



On remarque qu'un SOPR inférieur à 0.9 est caractéristique d'un bas du marché et constitue alors une bonne zone d'achat puisque le marché est alors propice à un rebond. Le SOPR est une donnée qui se calcule tous les jours. Dans le code on mettra en place une fonction permettant de récupérer la valeur actuelle du SOPR. Celle-ci pourra être utilisée dans différentes stratégies pour valider le fait que l'on se situe dans un creux du marché.