DOCUMENTATION POO - Framework de Backtesting d'Indices de Crypto Monnaie

Ce code permet de faire du backtesting sur des indices de crypto-monnaies, à partir d'un ensemble de paramètres renseignés par l'utilisateur.

I / Reference documentation

Le code commence par l'importation des packages nécessaires pour l'utilisation de notre future class. Si vous avez un message d'erreur lors de l'exécution du code sur les packages, vérifiez que vos packages sont importés sur Python. Si l'un des packages n'est pas installé, ouvrir le prompt CMD.exe et taper *pip install <nom du pagkage>*. ex : `(base) C:\Users\flavi>pip install cryptocmd`

Class Backtest :

La classe Backest initialise le backtest, en prenant en input un dataframe avec les valeurs historiques de la ou les crypto(s) sélectionnée(s). La classe contient la fonction __init__ qui initialise, vérifie que tous les paramètres soient corrects et gère les erreurs avec les `raise ValueError('…')`. Une fois les paramètres initialisés, le code enlève les doublons du dataframe, gère les erreurs, reformatte les dates. Les attributs de la classe backtest sont :

- `selection`: le dataframe contenant les valeurs et infos historiques
- `exclusions` : un dataframe composé des colonnes à exclure
- `start_date` : la date de début pour le backtest
- `ref_currency` : la devise de référence
- `log_delistings` : un Boolean qui indique s'il faut log les delistings.
- `wave` : chiffre 1 ou 2 pour désigner la vague à utiliser pour le backtest
- `multiple_days_rebal` : nombre de jours entre les rebalancements
- `cutoff_dt` : séries des dates de cutoff tirée du df selection.
- `effective_dt` : série des effective dates en fonction de la wave
- `wght_dt` : série de poids datés du portfolio
- `frequency_rebal` : fréquence du rebalancement ("Annual", "Semi-Annual", "Quarterly").

On trouve ensuite la fonction `__str__(self)` qui donne l'objet backtest créé sous forme de string. Elle retourne les caractéristiques associées à l'objet.

La fonction `__repr__` retourne un string de l'objet backtest afin de pouvoir répliquer l'objet, avec tous les paramètres de l'objet.

La fonction `performance_metrics` calcule et affiche les principaux indicateurs des indices par rapport au benchmark. Elle prend en paramètre indices_ts (un dataframe comprenant les indices) et benchmark_ts (un dataframe comprenant les valeurs du benchmark). La fonction calcule le rendement réalisé, le rendement moyen annuel, la volatilité annualisée, le ratio de sharpe, la perte maximum, la Value At Risk, la tracking error, le rendement excédentaire annualisé, le ratio d'information et la corrélation des rendements, le tout en pourcentage.

La fonction `generate_review_dates` calcule et retourne les review dates.

Inputs : - *months* : list des mois pour lesquels les review dates doivent être générées.

- *weekday* : jour de la semaine à choisir pour les review dates, exemple : 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'.
- *weekday_position* : numéro qui indique la position du weekday désiré dans le mois. Une valeur positive indique que le comte est fait depuis le début, et une valeur négative indique que l'on comptera depuis la fin. Par exemple., -1 pour la dernier évènement.

Ouput : série de review dates calculé sous contrainte des inputs sous forme d'array.

La fonction `first_wave_effective_dates` génère les dates effectives pour la première vague du backtest, en appelant la fonction generate_review_dates().

Inputs : self de la classe

Outputs : série des dates effectives pour la première vague sous forme d'array

La fonction second_wave_effective_dates génère les dates effectives pour la deuxième vague du backtesting, en appelant la fonction generate_review_dates. La différence avec la fonction juste au-dessus est dans l'input : la fonction prend self mais aussi dates_to_remove qui permet de donner une liste de dates à exclure des dates effectives de backtesting.

La fonction run exécute le backtest, calcule la performance et l'affiche sur un graphique de l'indice crypto. Elle commence par normaliser les les prix historiques du df historique (selection), la moyenne de ces prix normalisés pour créer l'indice crypto.

INPUTS : self

Output : série représentant la performance de l'indice crypto au cours du temps.

La fonction quantitative_strategy_ma applique la stratégie d'investissement Moving Average (MA) aux prix de close historiques de crypto stockés dans le df selection. Elle calcule le short-term MA (50 jours) et le long-terme MA (200 jours). Le buy signal (rep. sell signal) est lorsque le short-term MA est supérieur (rep. inférieur) au long-terme MA.

Output : série représentant les rendements cumulés de la stratégie d'investissement Moving Average.

La fonction quantitative_momentum_rebal applique la stratégie quantitative du Momentum au df selection de prix historiques crypto, de la même manière que la fonction au-dessus.

Input : self

Output : série représentant les rendements cumulés de la stratégie appliquée.

La fonction macd_strategy applique la stratégie quantitative MACD (Moving Average Convergence Divergence) au df selection de prix historiques crypto, de la même manière que la fonction au-dessus.

Input : self

Output : série représentant les rendements cumulés de la stratégie appliquée.

La fonction pairs_trading_strategy applique la stratégie quantitative de pair trading au df selection de prix historiques crypto, de la même manière que la fonction au-dessus.

Input : self

Output : série représentant les rendements cumulés de la stratégie appliquée.

La fonction momentum_crossover_strategy applique la stratégie de Momentum Crossover avec des filtres de volatilité au df selection. Elle calcule les rendements long-terme et court-terme en fonction de la taille des fenêtres données en input (par défaut 50).

Inputs : - self

- short_window : optionnel, taille de la fenêtre pour les rendements court-terme.
- long_window : optionnel, taille de la fenêtre pour les rendements long-terme.
- volatility_threshold : seuil de volatilité pour filtrer les signaux buy et les signaux sell.
- volatility_wondow : taille de la fenêtre pour la volatilité (une fenêtre plus grande peut donner une mesure plus lisse de la volatilité)

Output : dataframe avec les positions (buy, sell, hold) pour chaque crypto-monnaie calculé grâce à la stratégie.

La fonction volume_weighted_strategy implémente la stratégie quantitative de volume-weighted portfolio rééquilibrage.

Inputs : - self

- window : optionnel, taille de la fenêtre pour calculer les rolling averages des volumes traités.

Output : dictionnaire contenant les nouvelles positions de chaque cryptomonnaie après rééquilibrage du portefeuille avec la stratégie.

La fonction plot_performance_intervals affiche sur un graphique les rendements cumulés de l'indice sélectionné et du benchmark, sous contrainte des inputs. Elle définit les intervals en jours de 1 an, 6 mois, 3 mois et 1 mois, et calcule les rendements cumulés sur ces intervalles, avant de les afficher sur le graphique.

Inputs : - self

- benchmark_df : dataframe du benchmark (doit contenir les colonnes 'cutoff' et 'Close').

Output : rien, affiche le graphique des rendements mais ne retourne rien.

Toujours dans la classe Backtest, nous trouvons les fonctions get et set pour contrôler les attributs de l'objet backtest. Ces fonctions sont déclinées pour chaque attribut, par exemple getPrices, setPrices, getSelection, setSelection, getStartDate, ou encore setWave.

II / Tutoriel et lancement de la class basket :

Après avoir défini cette classe et ses fonctions, la suite du code fait appel à CoinMarketCap pour aller chercher les données historiques des cryptos sélectionnées, pour un intervalle de dates données.

La première étape pour l'utilisateur est d'inscrire les cryptomonnaies qu'il veut tester. Il choisit ensuite la date de début (date_from) et de fin (date_to) désirée pour les données. L'utilisateur peut ensuite lancer le script pour récupérer les données historiques.

Exemple :

```
crypto_df = pd.DataFrame({"CRYPTO": ["BTC", "ETH", "XRP"]})

    date_from = "01-01-2017"

    date_to = "01-12-2023"
```

La cellule suivante du code nettoie et réorganise les données récupérées plus haut. Le code enlève notamment les lignes avec données manquantes, celles dont la capitalisation de marché est nulle ou égale à zéro, renomme les colonnes, etc. Cette partie est une réorganisation et nettoyage des données qui vont être utilisées pour le backtest.

Ensuite, le dataframe nettoyé est nommé et assigné selection, puis imprimé pour l'utilisateur.

La class backtest est ensuite appelée, avec comme argument le dataframe selection et la date de début du backtest.

```
test = Backtest(selection, start_date=datetime.datetime(2019, 1, 1), )
```

Le code affiche les paramètres calculés du backtest, comme vu plus haut dans la documentation de la class Backtest. Le code affiche un résultat de cette forme :

```
Backtest parameters:


Start                    2019-01-01 00:00:00
Frequency of rebalancing                 120.0
Currency                                 USD
Wave                                     1
Rebalanced in                        1 day(s)
Log delistings                           False
```

La cellule suivante appelle la fonction run de la classe backtest, expliquée plus haut qui calcule le rendement de l'indice et affiche le graphique de performance de cet indice crypto. Il renvoie aussi le niveau de l'indice à chaque date.

Si vous une colonne weight avec un nom différent, la renommer `Weight` ou la dupliquer avec le nom correct. Ex : `df['Weight'] = df[`FFMC_Weight`]`

Dans la suite du code de l'API, la procédure est répétée mais pour un plus grand nombre de cryptomonnaies. Les données sont récupérées puis triées et organisées, avant de calculer les principaux indicateurs en appelant la fonction performance_metrics de la classe Backtest, avec comme argument le dataframe selection et le dataframe bench (celui que l'on vient de créer). Les résultats sont affichés, comme présenté plus haut dans la description de la fonction.

```
Realized Return (%): -99.99998265962968
Annualized Average Return (%): -79.27952835430376
Annualized Volatility (%): 130.68889083858167
Sharpe Ratio: -0.6066279072811523
Maximum Drawdown (%): 99.99999180716365
Value At Risk (%): -9.267737204716179
Tracking Error (%): 212.55477940195263
Annualized Excess Return (%): -39.24140183813145
Information Ratio: -0.18461782863006732
Return Correlation (%) : 47.791812645573806
```

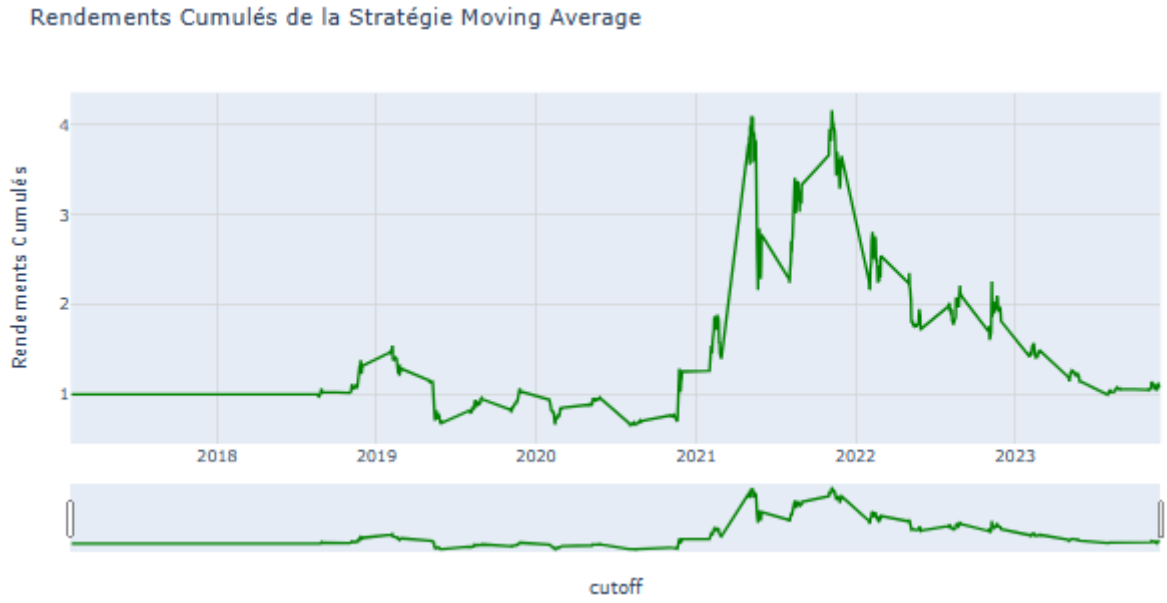
Pour développer l'analyse, le code combine notre premier dataframe et une base de données des composantes de CoinDesk Market récupérée sur un CSV. Le dataframe selection est ainsi enrichi avec une colonne supplémentaire 'Industry Name ', qui est ensuite imprimé pour l'utilisateur.

La class backtest est à nouveau appelé avec cette fois en argument le dataframe selection (enrichi de la colonne Industry Name), affichant les paramètres du backtest.

Le code appel ensuite la fonction getWeightRepartition de la class backtest pour calculer et afficher la répartition des poids des industries dans notre dataframe (bien égale à1)

	Industry Name	Weight
0	Transparent	0.725206
1	Layer 1	0.242666
2	BaaS	0.032128

La suite du code applique les différentes stratégies quantitatives d'investissement (expliquées plus haut) à nos données et en imprime les rendements cumulés sur graphique et en tableau avec chaque date (avec la possibilité de zoomer sur chacun d'entre eux afin d'avoir une meilleure visibilité sur des courtes périodes)



Comme nous pouvons le voir ci-dessous, la barre du dessous permet de sélectionner un range plus petit que le range de base, ce qui peut donc permettre une analyse plus précise sur les stratégies évoquées.