Projets

Université Paris Dauphine

10 octobre 2024

Projet : Framework de Backtesting d'Indices de Crypto-Monnaie

Description du projet

Conception d'un framework de backtesting pour des indices de crypto-monnaies. Ce framework vous permettra d'évaluer les performances de différents indices basés sur des thématiques particulières, en utilisant des données réelles récupérées à partir des API de CoinGecko et Binance. Le framework renverra la performance des indices construits à partir d'un ensemble de paramètres sélectionnés par l'utilisateur.

Consignes

1. **Récupération des données** :
   - Utilisez le package `CoinGeckoAPI` pour récupérer les tags d'une liste d'actif depuis l'API CoinGecko.
   - En utilisant le package `python-binance`, récupérez les prix historiques des actifs qui composeront le backtest à partir de l'API Binance.
2. **Implémentation du backtest** :
   - Concevez un backtester qui accepte un dictionnaire en entrée. Ce dictionnaire doit contenir :
     - Les dates de début et de fin du backtest.
     - La thématique souhaitée (définie à partir des tags).
     - La stratégie à appliquer (equal weight, momentum, etc.).
     - Tout autre élément pertinent.
   - Le backtester devra renvoyer :
     - Un track record.
     - Les poids historiques de la composition de l'indice.
     - Des métriques d'analyse de performance (par exemple : le rendement total, le rendement annualisé, la volatilité, le ratio de Sharpe, etc.).

3. **Structuration du code** :
   — Assurez-vous d'adopter une approche orientée objet pour la structure de votre projet.
   — Divisez votre code en modules et classes distincts pour chaque fonctionnalité (par exemple : récupération de données, analyse, backtesting, etc.).
   — Commentez et documentez votre code pour faciliter sa compréhension.

Critères d'évaluation

1. **Structure du code** :
   — Organisation et modularité du code.
   — Utilisation appropriée des éléments de la programmation orientée objet vus en cours.
   — Clarté et propreté du code (pas de code redondant, noms de variables explicites, etc.).
2. **Fonctionnalité du framework** :
   — Exactitude des données récupérées.
   — Pertinence des métriques de performance proposées.
   — Capacité du framework à gérer différentes thématiques et stratégies.
3. **Documentation** :
   — Présence de commentaires explicatifs.
   — Qualité et clarté de la documentation fournie (par exemple, docstrings pour les classes et méthodes).

Ressources supplémentaires

— Documentation du package `CoinGeckoAPI` : `https://pypi.org/project/pycoingecko/`
— Documentation de l'API CoinGecko : `https://www.coingecko.com/en/api/documentation`
— Documentation du package `python-binance` : `https://python-binance.readthedocs.io/en/latest/overview.html`
— Documentation de l'API Binance : `https://binance-docs.github.io/apidocs/spot/en/#market-data-endpoints`

Conseil

La récupération des données via CoinGecko et Binance peut prendre un certain temps. Si vous souhaitez tester votre code plus facilement, vous pouvez enregistrer les données récupérées dans un fichier Excel ou CSV afin de pouvoir les charger plus rapidement.

Projet : Framework d'Analyse de Portefeuille

Description du projet

Conception d'un framework permettant l'analyse d'un portefeuille pour un sélectionneur de fonds. Ce framework vous aidera à évaluer les performances et les risques associés à un fonds en utilisant des statistiques descriptives et une analyse factorielle basée sur les séries AQR. L'objectif est de fournir un outil complet qui facilite la prise de décision en offrant une vue détaillée des facteurs influençant la performance du fonds.

Consignes

1. **Données** :
   - Utilisez les séries de valeurs liquidatives (NAV) des fonds qui vous seront fournies.
   - Intégrez les séries de facteurs AQR pour effectuer l'analyse factorielle.
2. **Implémentation de l'analyse** :
   - L'analyse devra comporter un ensemble de statistiques descriptives sur la performance et le risque du fonds. Ces statistiques peuvent inclure :
     - Performance totale et annualisée.
     - Volatilité.
     - Ratio de Sharpe.
     - Ratio de Sortino.
     - Downside volatility.
     - Rendement excédentaire (excess return) total et annualisé.
     - Beta.
     - Tracking error.
     - Alpha.
     - Beta up et beta down.
     - Maximum drawdown.
     - Maximum relative drawdown.
   - L'analyse devra également permettre de réaliser une analyse factorielle du fonds en utilisant les séries AQR.
   - Fournissez des visualisations claires des résultats de l'analyse (par exemple, graph, tableaux comparatifs, etc.).
3. **Fonctionnalités du framework** :
   - Permettez à l'utilisateur de sélectionner différents fonds pour l'analyse.
   - Offrez la possibilité de comparer plusieurs fonds entre eux.
4. **Structuration du code** :

— Adoptez une approche orientée objet pour la structure de votre projet.
— Divisez votre code en modules et classes distincts pour chaque fonctionnalité.
— Commentez et documentez votre code pour faciliter sa compréhension.

Critères d'évaluation

1. **Structure du code** :
   — Organisation et modularité du code.
   — Utilisation appropriée des concepts de programmation orientée objet vus en cours.
   — Clarté et propreté du code (pas de redondance, noms de variables explicites, etc.).
2. **Documentation** :
   — Présence de commentaires explicatifs.
   — Qualité et clarté de la documentation fournie (par exemple, docstrings pour les classes et méthodes).

Conseil

De nombreux packages permettent d'effectuer une analyse factorielle. Une solution peut être d'utiliser statsmodels. D'autres packages sont égalemment possible et aucun en particulier n'est imposé.

Pour faciliter le développement et les tests, vous pouvez sauvegarder les données fournies dans un format pratique (par exemple, CSV ou Excel) pour une utilisation rapide lors de l'exécution de votre code.

Projet : FastApiDecoratorBuilder

Description du projet

Le but du projet *FastApiDecoratorBuilder* est de concevoir un décorateur Python qui transforme une fonction Python en une API FastAPI basée sur la fonction et des configurations définies.

Objectifs du projet

— **Création de Décorateur** : Construire un décorateur qui transforme une fonction Python en API FastAPI.

— **Gestion de Configurations** : Implémenter un mécanisme de configuration pour l'API.

Consignes

1. **Développement du Décorateur** :
   — Élaborer un décorateur qui, appliqué à une fonction, génère une API FastAPI correspondante.
2. **Configuration de l'API** :
   — Intégrer une méthode pour configurer les propriétés de l'API générée, telles que les routes et les méthodes HTTP acceptées.
3. **Exemple de test** :
   — Implémenter des cas pratique pour les cas d'usages de votre package.

Critères d'évaluation

— **Fonctionnalité** : Est-ce que le décorateur produit une API fonctionnelle ?

— **Configurabilité** : La configurabilité de l'API est-elle suffisamment flexible ?

— **Documentation** : Est-ce que votre documentation décrit clairement comment utiliser le décorateur et configurer l'API ?

— **Structure du code** : Organisation et modularité du code, Utilisation appropriée des concepts de programmation orientée objet vus en cours, Clarté et propreté du code (pas de redondance, noms de variables explicites, etc.).

Ressources supplémentaires

— `https://fastapi.tiangolo.com/`

— `https://realpython.com/primer-on-python-decorators/`

Note

Assurez-vous d'inclure une documentation détaillée, y compris un fichier `README.md`, qui explique comment utiliser votre décorateur et configurer l'API générée.