Data Management

Céline Nevo, Tristan Voirin, Axel Sottile

Mars 2025

Contents

1 Introduction

Ce document explique le code utilisé dans notre projet. Le projet vise à construire une base de données pour un fonds d'investissement, à développer plusieurs stratégies en fonction des profils clients, et à évaluer la performance des stratégies établies entre le début de l'année 2023 et la fin de l'année 2024.

2 Structure du code

2.1 Création de la database

Le fichier `creation_db.py` regroupe l'ensemble des fonctions nécessaires à la création de la base de données. Dans un premier temps, six tables vides sont initialisées : `clients`, `portfolios`, `deals`, `returns`, `products` et `manager`.

Ensuite, différentes fonctions permettent d'insérer des données dans ces tables :

- trois portefeuilles fictifs sont générés,
- dix clients sont créés avec des profils de risque attribués aléatoirement,
- des gestionnaires (`managers`) sont associés chacun à un portefeuille tiré au hasard parmi les trois existants,
- les *tickers* issus du fichier `dico.py` sont importés dans la table `products`, avec leur type respectif (`equity`, `bond`, `commodity`).

2.2 Importation des données

Le fichier d'importation des données dans la base de données est nommé `import_data.py`. À l'intérieur, on trouve une classe `DataImporter` qui contient la fonction `fill_returns`. Cette fonction s'appuie sur trois paramètres : le chemin de la base de données, une date de début et une date de fin.

Dans un premier temps, elle récupère tous les tickers des actifs depuis la table `Products`, avant de télécharger leurs cours sur la période délimitée par les deux dates sur Yahoo Finance. Elle calcule ensuite les rendements associés et les insère dans la table `Returns` via une requête SQL. Chaque rendement est caractérisé par un `product_id`, une date et une valeur.

Pour calibrer le modèle linéaire à partir de données historiques, nous avons intégré les valeurs comprises entre le début de l'année 2019 et la fin de l'année 2022.

2.3 Création du modèle linéaire

Le modèle de régression linéaire est implémenté dans le fichier `model.py`. Il est utilisé dans la stratégie 2 (`low_turnover`) pour prédire les rendements à une date donnée.

La fonction `fit_model`, chargée de construire et d'entraîner ce modèle, prend cinq paramètres : une date de début, une date de fin, le chemin vers la base de données, une taille de fenêtre (`window_size`) et un chemin de sauvegarde pour enregistrer le modèle entraîné.

La fonction extrait les rendements depuis la table `returns` de la base de données, pour la période définie entre les deux dates. Le modèle s'appuie alors sur une fenêtre glissante de `window_size` observations comme variables explicatives, afin de prédire le rendement du jour suivant. L'apprentissage repose sur une régression linéaire classique de la librairie : sklearn.linear_model.

2.4 Statégies

Le fichier `strategies.py` regroupe trois stratégies d'investissement sous forme de fonctions, chacune adaptée à un type d'investisseur :

- **low_risk** : pour ceux qui souhaitent limiter le risque
- **linear_strategy** : pour les investisseurs souhaitant un faible turnover
- **high_yield** : pour les investisseurs prêts à accepter plus de risque pour viser un meilleur rendement

Chaque fonction renvoie un DataFrame indexé par le `product_id` (identifiant de l'actif) présent dans la table `Products`. Ce DataFrame comporte une colonne "weight", qui correspond à la pondération retenue pour chaque actif parmi les 129 possibles. Au final, c'est donc la répartition optimale des actifs selon la stratégie sélectionnée qui est retournée.

2.4.1 low_risk

Cette stratégie requiert deux paramètres : la table `returns` de la base de données et une volatilité cible (`target_volatility`). Elle commence par récupérer les rendements quotidiens (dans la table `returns`) de chaque actif sur la dernière année (les 252 derniers jours), via une requête SQL, avant de lancer l'optimisation. L'objectif est de réduire au minimum l'écart entre la volatilité observée sur cette période et la volatilité visée (par défaut, 10%). L'optimisation se fait en tenant compte des contraintes suivantes :

- la somme des poids doit être égale à 1
- la somme des poids attribués aux actifs de type `bond` doit représenter au moins 60% du portefeuille, car ces actifs sont moins risqués
- la vente à découvert est interdite (les poids sont compris entre 0 et 1)

2.4.2 low_turnover

Cette stratégie repose sur un modèle de régression linéaire présenté dans la section **??**. La fonction associée prend quatre arguments en entrée :

- le chemin vers la base de données contenant la table `returns` ;
- une date cible ;
- une taille de fenêtre (`window_size`) ;
- le chemin vers le modèle préalablement entraîné.

La fonction permet d'estimer les rendements de chaque actif à la date cible en utilisant les `window_size` derniers rendements observés. Les poids du portefeuille sont ensuite calculés en divisant le rendement prédit de l'actif i par la somme des rendements (en valeur absolue) de l'ensemble des actifs.

Le portefeuille est considéré comme pertinent pour un investissement uniquement si la moyenne des rendements (en valeur absolue) dépasse un seuil de 0,0025. Ce critère permet de filtrer les signaux faibles et de ne pas sélectionner tous les portefeuilles. Seuls ceux présentant un potentiel suffisamment significatif sont considérés.

2.4.3 high_yield

Cette stratégie fait appel à deux paramètres essentiels : la table `returns` ainsi qu'un nombre de jours (`days`). À partir de ces informations, elle extrait les rendements des actifs sur la période correspondant aux `days` derniers jours, puis exécute un programme d'optimisation. L'objectif est de maximiser le rendement moyen du portefeuille sur cet intervalle, ce qui revient, en pratique, à minimiser la valeur négative de ce rendement. Cette optimisation se fait selon trois contraintes :

- la somme des poids doit être égale à 1
- l'intégralité de l'investissement doit être réalisée sur des actifs de type `equity`
- la vente à découvert est autorisée

2.5 Mise à jour de la base de données

Le fichier `base_update.py` contient trois fonctions destinées à mettre à jour la base de données chaque fois qu'un nouveau portefeuille est créé.

La fonction **update_portfolio** prend en entrée un DataFrame contenant une colonne `weight`, un profil de risque, une date ainsi que le chemin vers la base de données. Elle insère ensuite ces informations dans la table `Portfolios`.

La fonction **update_deals** utilise les mêmes paramètres que la fonction **update_portfolio**. Toutefois, son rôle est différent : elle identifie le dernier portefeuille associé au profil de risque donné dans la table `Portfolios`, puis calcule la différence de pondération entre ce portefeuille et le nouveau. La table `deals` n'enregistre donc pas les quantités exactes d'actions achetées ou vendues, mais uniquement les variations de pondération entre deux portefeuilles successifs. En effet, le volume réel de titres échangés dépend du montant total investi, qui peut varier d'une date à l'autre. Pour rendre le code plus flexible, seule la différence de pondération est stockée ; elle pourra ensuite être multipliée par le montant investi pour obtenir le nombre de titres effectivement échangés. Par ailleurs, la fonction permet d'insérer une unique ligne par date, contenant les pondérations des trois profils de portefeuille. Si aucun investissement n'est effectué à cette date pour un profil donné, la valeur associée est renseignée par `NULL`.

Les portefeuilles sont enregistrés dans la base de données sous le format JSON.

2.6 Performances

Le fichier `metrics.py` calcule l'ensemble des métriques relatives aux performances des fonds. Il est composé d'une classe, `PortfolioMetrics`, qui a 3 fonctions :

- Il extrait les données de rendements afin de calculer les performances de chaque portefeuille
- Il calcule les statistiques associées (rendement moyen, volatilité, ratio de sharpe ...)
- Il met en place des graphiques permettant de visualiser les performances

2.7 Dashboard

Le dashboard permet de présenter convenablement les résultats à l'utilisateur via la bibliothèque *streamlit*. Il se décompose en 4 parties :

- **Indicateurs de performance** : l'utilisateur choisit des métriques de performance pour une période d'évaluation d'une des trois stratégies
- **Graphiques** : présentation graphique des rendements cumulés ainsi que des drawdown de la stratégie
- **Comparaison** : possibilité pour l'utilisateur de comparer deux stratégies
- **Détails et transactions récentes** : récapitulatif des règles d'investissement de la stratégie choisit et présentation des transactions effectuées sur les cinq dernières semaines

2.8 Fichier d'exécution principal

Le fichier `main.ipynb` permet d'exécuter l'ensemble des fonctions nécessaires pour évaluer la performance du fonds entre le début de l'année 2023 et la fin de l'année 2024. La procédure suivie est la suivante :

1. Importation des bibliothèques Python ainsi que des modules développés pour le projet
2. Création de la base de données
3. Remplissage de la table `returns` à l'aide du module `DataImporter`
4. Entraînement du modèle de prédiction
5. Mise en place d'une boucle d'investissement simulée chaque lundi sur la période définie
6. Calcul des performances du portefeuille
7. Visualisation des résultats à l'aide de l'interface développée sous `Streamlit`