Présentation de la structure du projet

Présentation de *data_collector.py*

Le module *data_collector.py* est conçu pour récupérer les données financières nécessaires à l'analyse des portefeuilles. Il utilise des bibliothèques comme *yfinance*, *pandas*, et *numpy* pour collecter et traiter les données. Voici une présentation détaillée de ce module :

1. **Fonctionnalités Principales**

1.1. Imports des Librairies :

- *yfinance* : Utilisé pour récupérer les données financières depuis Yahoo Finance.

- *pandas* : Permet de manipuler les données sous forme de DataFrame.

- *numpy* : Utilisé pour les calculs numériques, notamment pour remplacer les valeurs infinies.

1.2. Initialisation des Paramètres :

- Dates de Début et de Fin : La période du projet est définie entre le 1er janvier 2023 et le 31 décembre 2024.

- Dictionnaires de Données :

  - *dict_products* : Associe des tickers boursiers à des noms de produits financiers.

  - *dict_risk_type* : Associe chaque ticker à un profil de risque spécifique (par exemple, "low_risk", "high_yield_equity_only").

1.3. Fonction *main* :

- Récupère les prix de clôture des produits financiers définis dans *dict_products* pour la période spécifiée.

- Calcule les rendements journaliers avec .pct_change().

- Traite les données manquantes avec .ffill() et remplace les rendements infinis par 0.

- Renomme les colonnes pour correspondre aux noms des produits financiers.

- Supprime les lignes contenant des valeurs NaN.

- Retourne un DataFrame contenant les rendements journaliers prêts pour l'analyse.

2. Chronologie d'Exécution

1. Importation des Librairies : Chargement des bibliothèques nécessaires pour la récupération, la manipulation des données, et les calculs.

2. Initialisation des Paramètres : Définition des dates et des dictionnaires contenant les produits financiers et leurs profils de risque.

3. Récupération des Données Financières : Utilisation de yfinance.download() pour obtenir les prix de clôture des tickers spécifiés.

4. Calcul des Rendements : Calcul et nettoyage des rendements journaliers.

5. Retour des Résultats : Renvoi du DataFrame final contenant les rendements journaliers.

Présentation de *base_builder.py*

Le module *base_builder.py* est responsable de la création et du peuplement de la base de données pour la gestion des portefeuilles financiers. Il définit la structure des tables et utilise plusieurs modules pour insérer les données.

1. Structure des Tables et Logique de Peuplement

1.1. Clients :

- Description : Stocke les informations des clients, notamment leur identité, email, et profil de risque.

- Table Clients :

  - id (clé primaire : auto-incrémenté)

  - nom, prenom, birth_date, address, phone_number, email, entry_date, risk_profile

- Logique de Peuplement :

  - Utilisation du module *faker* pour générer des caractéristiques clients fictifs.

  - Attribution d’un profil de risque différent à chaque client.

- Code Associé :

  - Classe Clients : Désigne les caractéristiques des clients.

  - Module *clients_to_base* : Insère les clients créés dans la base.

  - Module *pop_clients_base* : Peuple la base selon la logique décrite.

1.2. Products :

- Description : Répertorie les actifs disponibles du fonds et leur profil de risque.

- Table Products :

  - product_id (clé primaire : auto-incrémenté)

  - ticker, product_risk_profile, name

- Logique de Peuplement :

  - Récupération des dictionnaires de data_collector pour les tickers et les profils de risque.

  - Insertion des produits dans la base.

- Code Associé :

  - Classe Products : Désigne les caractéristiques des produits.

  - Module products_to_base : Insère les produits créés dans la base.

  - Module pop_products_base : Peuple la base selon la logique décrite.

1.3. Portefeuilles :

- Description : Représente les portefeuilles d'investissement, chacun étant associé à un profil de risque et contenant une sélection d'actifs financiers.

- Table Portfolios :

  - portfolio_id (clé primaire : auto-incrémenté)

  - wallet_name, risk_profile, products (JSON contenant les produits du portefeuille)

- Logique de Peuplement :

  - Récupération des tickers et profils de risque via get_tickers_by_risk_profile().

  - Création de portefeuilles en regroupant les produits par profil de risque.

  - Insertion des portefeuilles dans la base.

- Code Associé :

  - Classe Wallet : Modélise un portefeuille.

  - Module wallet_to_base : Insère les portefeuilles dans la base.

  - Module populate_wallets : Crée et insère les portefeuilles.

1.4. Gestionnaires de Portefeuilles :

- Description : Représente les gestionnaires responsables de la supervision des portefeuilles.

- Table Managers :

  - manager_id (clé primaire : auto-incrémenté)

  - manager_name, email, wallets_managed_id

- Logique de Peuplement :

  - Récupération des identifiants des portefeuilles via get_wallet_id().

  - Création de gestionnaires associés à chaque portefeuille.

  - Insertion des gestionnaires dans la base.

- Code Associé :

  - Classe Manager : Modélise un gestionnaire.

  - Module manager_to_base : Insère les gestionnaires dans la base.

  - Module pop_manager_base : Crée et insère les gestionnaires.

1.5. Transactions :

- Description : Représente les transactions réalisées sur les portefeuilles.

- Table Deals :

  - deal_id (clé primaire : auto-incrémenté)

  - date, wallet_id, manager_id, product_id, qty

- Logique de Peuplement :

  - Récupération des IDs des produits via fetch_product_ids() et des noms via fetch_product_name().

  - Enregistrement des transactions effectuées par les gestionnaires.

- Insertion des transactions dans la base.

- Code Associé :

- Classe Deal : Modélise une transaction.

- Module *deal_to_base* : Insère les transactions dans la base.

- Module *populate_returns_table* : Insère les données des rendements.

2. Fonction *main*

2.1. Récupération des Données de Rendements :

- Appel de dc.main() pour obtenir les données de rendements et les dictionnaires nécessaires.

2.2. Création des Tables :

- Exécution des requêtes SQL pour créer les tables Clients, Products, Portfolios, Managers, Deals, et Returns.

2.3. Peuplement de la Base de Données :

- Exécution des fonctions de peuplement pour insérer des données fictives dans les tables, en respectant un ordre précis pour garantir l'intégrité des données.

Présentation de *strategy.py*

Le module *strategy.py* joue un rôle central dans la gestion des stratégies d'investissement et la mise à jour des portefeuilles au sein du projet de gestion de fonds. Il intègre diverses stratégies adaptées à différents profils de risque et assure l'enregistrement des transactions dans la base de données. Voici une présentation détaillée de ce module.

1. **Fonctionnalités Principales**

Récupération des Rendements Historiques :

La fonction *fetch_returns_from_db* est responsable de l'extraction des données de rendements historiques depuis la base de données SQLite. Elle utilise pandas pour structurer ces données sous forme de DataFrame, facilitant ainsi leur manipulation et leur analyse. Cette étape est cruciale car elle fournit les données nécessaires pour évaluer la performance des produits financiers et prendre des décisions éclairées.

- Stratégies d'Investissement :

- Stratégie à Low Volatility:

La fonction *low_risk_strategy* utilise la volatilité et le momentum pour prendre des décisions d'investissement. Les produits à acheter ou vendre sont déterminés selon qu'ils sont en dessous ou pas du seuil de 10%. La quantité à acheter ou vendre est ensuite déterminée à partir du momentum.

- Stratégie Low Turnover :

La fonction *low_turnover_strategy* limite le nombre de transactions mensuelles à deux. Les produits à acheter ou vendre sont ceux dont la quantité, déterminée par le momentum, est le plus élevée, afin de profiter uniquement des mouvements les plus importants.

- Stratégie High Yield Equity :

La fonction *high_yield_equity_strategy* se concentre sur les actions présentant un momentum élevé, ajustant les positions en fonction des rendements récents. Sans limite de transactions mensuelles, on peut mettre en œuvre une stratégie plus « court-terme » et utiliser un momentum 10 jours plutôt que 30. Pour équilibrer, les quantités sont alors déterminées en multipliant par 5 le momentum et plus par 10.

- Enregistrement des Transactions :

La fonction *record_deals* insère les décisions d'investissement dans la table Deals de la base de données. Elle prend en compte les limites de transactions, si nécessaire, pour s'assurer que les opérations respectent les contraintes définies par chaque stratégie.

- Mise à Jour Hebdomadaire des Portefeuilles :

La fonction *update_portfolios* met à jour les portefeuilles chaque semaine en appliquant la stratégie appropriée en fonction du profil de risque. La fonction *run_weekly_updates* orchestre ces mises à jour sur une période définie, assurant ainsi que les portefeuilles restent alignés avec les objectifs d'investissement et les conditions du marché.

2. Chronologie d'Exécution

- Connexion à la Base de Données :

Le module commence par se connecter à la base de données SQLite pour récupérer les données nécessaires et enregistrer les transactions. Cette étape est essentielle pour garantir l'intégrité et la cohérence des données utilisées dans les analyses et les décisions d'investissement.

- Calcul des Indicateurs :

Les stratégies calculent des indicateurs financiers tels que la volatilité et le momentum. Ces indicateurs sont utilisés pour évaluer la performance des produits financiers et prendre des décisions d'investissement éclairées.

- Application des Stratégies :

En fonction du profil de risque de chaque portefeuille, la stratégie appropriée est appliquée pour déterminer les achats et les ventes. Cette approche permet de personnaliser la gestion des portefeuilles en fonction des préférences et des tolérances au risque des investisseurs.

- Enregistrement des Décisions :

Les décisions sont enregistrées dans la base de données, et les transactions sont effectuées en conséquence. Cette étape assure la traçabilité des opérations et permet de suivre l'évolution des portefeuilles au fil du temps.

- Mise à Jour Hebdomadaire :

Les portefeuilles sont mis à jour chaque semaine pour refléter les nouvelles données de marché et les décisions d'investissement. Cette mise à jour régulière permet de maintenir les portefeuilles optimisés et alignés avec les objectifs d'investissement.

En résumé, *strategy.py* est un module clé qui assure la gestion dynamique et efficace des portefeuilles, en s'adaptant aux conditions du marché et aux profils de risque des clients.

Présentation de *performances.py*

Le module *performances.py* réalise une analyse des portefeuilles du fond en se connectant à la base de données *project_database.db*, en récupérant des données de marché, et en calculant divers indicateurs de performance. Voici une présentation détaillée de ce module :

1. **Bibliothèques Utilisées et Paramètres**

- Bibliothèques :

- *sqlite3* : Pour la gestion de la base de données.

- *pandas* et *numpy* : Pour le traitement des données.

- *matplotlib.pyplot* : Pour la visualisation graphique.

- *yfinance* : Pour télécharger les données du SP500.

- *ast* : Pour convertir des chaînes de caractères en listes.

- Paramètres Globaux :

- Définition de constantes comme le chemin de la base de données, le taux sans risque annuel, le nombre de jours de bourse, et la période d'analyse.

2. **Fonctions de Gestion de la Base de Données et Récupération des Données**

- Connexion et Extraction :

- *connect_db(db_path)* : Se connecte à la base SQLite et renvoie la connexion.

- *get_products_for_wallet(conn, wallet_id)* : Récupère la liste des produits associés à un portefeuille.

- Récupération des Retours et des Deals :

- *get_portfolio_returns(conn, wallet_id)* : Récupère les retours journaliers moyens des produits associés à un portefeuille.

- *get_recent_deals(conn, wallet_id, limit=50)* : Récupère les derniers deals d'un portefeuille.

- Récupération des Données de Marché et des Correspondances :

- *get_sp500_returns()* : Télécharge et prépare les données du SP500.

- *get_portfolio_ids(conn)* et *get_portfolio_manager_mapping(conn)* : Récupèrent la liste des portefeuilles et la correspondance entre portefeuilles et managers.

3. Fonctions de Calcul des Indicateurs Financiers

- Indicateurs de Performance :

- *compute_sharpe_ratio(returns_series)* : Calcule le ratio de Sharpe annualisé.

- *compute_cumulative_returns(df)* : Calcule le rendement cumulé.

- *compute_beta(portfolio_df, benchmark_df)* : Calcule le bêta du portefeuille par rapport au SP500.

- *compute_volatility(returns_series)* : Calcule la volatilité annualisée.

- *compute_max_drawdown(df)* : Calcule le maximum drawdown du portefeuille.

- Affichage du Contenu des Portefeuilles :

- *display_portfolio_content(conn, wallet_name)* : Affiche les produits contenus dans un portefeuille.

4. Fonction *main* et Chronologie d'Exécution

1. Connexion à la Base de Données :

- Appel de *connect_db(DB_PATH)* pour établir la connexion.

2. Affichage des Portefeuilles et de leur Contenu :

- Récupération des portefeuilles avec *get_portfolio_ids(conn)*.

- Affichage des produits pour chaque portefeuille via *display_portfolio_content*.

3. Affichage des Derniers Deals :

- Récupération et affichage des 10 derniers deals pour chaque portefeuille avec *get_recent_deals*.

4. Récupération des Données du SP500 :

- Téléchargement et préparation des données du SP500 avec *get_sp500_returns*.

5. Calcul des Indicateurs Financiers :

- Pour chaque portefeuille, calcul du ratio de Sharpe, de la volatilité, du rendement cumulé, du max drawdown, et du bêta.

6. Affichage des Résultats :

- Affichage des indicateurs calculés dans la console.

- Identification du meilleur manager en fonction du rendement cumulé moyen.

7. Visualisation Graphique :

- Traçage d'un graphique représentant l'évolution du rendement cumulé de chaque portefeuille.

8. Fermeture de la Connexion :

- Fermeture de la connexion à la base de données à la fin de l'exécution.

5. Application Web Interactive pour la Visualisation des Performances

Une application web interactive a été créée via Streamlit pour mieux visualiser les performances des portefeuilles. Elle est accessible via le lien suivant :

- https://data-management-project-performances.streamlit.app.

Le code source de cette application se trouve dans le fichier *app.py*.