Synthèse Python : Risques et performance

1. Extraction de Données avec `yfinance`

La bibliothèque `yfinance` est l'outil principal pour l'acquisition de données de marché en temps réel et historiques.

Récupération des données descriptives

La méthode `yf.Ticker(ticker).info` est employée pour extraire les métadonnées de chaque actif. Ces informations sont essentielles pour la structuration de l'analyse :

Catégorie	Description
Identité	Nom long de l'actif (`longName`).
Classification	Secteur d'activité (`sector`), industrie (`industry`) et pays (`country`).
Données financières	Devise de cotation (`currency`) et capitalisation boursière (`marketCap`).

Téléchargement des prix historiques

La fonction `yf.download(ticker, start="YYYY-MM-DD")` permet de récupérer les séries temporelles de prix.

2. Gestion du Cache et Tests de Présence de Fichiers

Afin d'optimiser les performances et d'éviter des téléchargements redondants, il faut intègrer une logique de mise en cache des données à l'aide de la bibliothèque standard `os`.

Méthodes de persistance

Vérification de l'existence d'un fichier local avant de lancer une requête réseau :

- `os.path.exists(file_path)` : Cette fonction permet de tester si les données (descriptions ou prix) ont déjà été sauvegardées localement.
- **Formats de stockage** : il est conseillé d'utiliser le format **Parquet** (`.parquet`) pour sa rapidité et son efficacité

Une constante de contrôle (`REFRESH_TICKER_DATA`) permet de forcer ou non la mise à jour des données locales en court-circuitant le test de présence.

3. Manipulation et Retraitement avec `pandas`

`pandas` constitue le cœur du moteur de calcul du script. Il est utilisé pour transformer des données brutes en structures matricielles exploitables pour l'analyse financière.

Structuration et Nettoyage

- `pd.pivot_table()` : Transforme les données de prix d'un format long (liste) vers un format matriciel où les dates sont en lignes et les codes d'actifs (*tickers*) en colonnes.

- `ffill()` (*Forward Fill*) : Gère les données manquantes en propageant la dernière valeur connue, une pratique standard pour traiter les jours de fermeture de marchés spécifiques.
- `iloc` et `loc` : Utilisés pour le filtrage temporel et la sélection précise de segments de données.

Analyse et Synthèse

- `resample('M').last()` : Permet de rééchantillonner les données quotidiennes en données mensuelles pour l'analyse des performances périodiques.
- `pct_change()` : Calcule les variations relatives pour obtenir les rendements arithmétiques.
- `merge()` et `concat()` : Unifie les portefeuilles, les descriptions d'actifs et les indicateurs de performance dans des DataFrames consolidés.
- `groupby()` et `agg()` : Permettent des analyses segmentées, par exemple pour calculer le **Profit & Loss (P&L)** par secteur d'activité ou par pays.

4. Calculs Quantitatifs avec `numpy`

La bibliothèque `numpy` intervient pour les opérations mathématiques complexes et l'analyse statistique des risques.

- **Rendements Logarithmiques** : Calculés via `np.log(prix / prix.shift(1))`.
- **Volatilité Annualisée** : Calculée en multipliant l'écart-type des rendements (`std()`) par la racine carrée du nombre de jours de bourse (`np.sqrt(252)`).
- **Analyse de Risque Ex-post** :
  - `np.cov()` : Calcul de la matrice de covariance.
  - `np.corrcoef()` : Corrélation entre un portefeuille et son benchmark.
  - Calcul du **Bêta**.

5. Visualisation et Exportation

Fonctionnalité	Méthode / Bibliothèque	Usage
Graphiques Linéaires	`matplotlib.pyplot.plot()`	Évolution de la Valeur Liquidative (VL) des actifs et portefeuilles.
Histogrammes	`matplotlib.pyplot.bar()`	Comparaison des performances mensuelles et P&L par catégorie.
Exportation CSV	`df.to_csv(sep=";")`	Sauvegarde des synthèses de performance et des détails de portefeuilles.
Exportation JSON	`json.dump()`	Archivage des indicateurs de risque (Bêta, Tracking Error, Corrélation).