DESCRIPTION DÉTAILLÉE DE NOTRE SOLUTION

Le projet est décomposé en plusieurs classes et classes héritées. Nous présentons dans ce document la structure du code et les différentes méthodes.

Program

C'est dans ce document que se trouve la méthode principale. La méthode principale permet de :

- Lancer les tests unitaires en faisant appel à la méthode `TestsUnitaires` (cette ligne est mise en commentaire et se lance donc lorsque le développeur le souhaite).
- Lancer le formulaire `Form_Strat`.

Form_Strat

Ce formulaire est l'interface principal qui permet d'interagir avec l'utilisateur. Celui-ci a la possibilité de choisir une stratégie de trading parmi celles proposées, ainsi que les paramètres qui s'y rapportent. Il lance ensuite le calcul du modèle d'évaluation de performance de sa stratégie à l'aide d'un bouton (Run).

Cette classe est composée de :

- Méthodes d'initialisation : Lors du lancement du formulaire, il faut remplir les listes qui permettent à l'utilisateur de choisir parmi des stratégies. On fait appel à `FillCombos`.

- Méthodes évènementielles :
  `Combo_StratType_SelectedIndexChanged` : Comme les différentes stratégies requièrent des paramètres différents, nous avons codé des évènements qui permettent de masquer les champs non nécessaires afin de rendre le formulaire plus *user-friendly*.
  `btn_Run_Click` : C'est le bouton qui lance la méthode `RunModel` et donc le calcul du modèle. Il calcule la performance, la volatilité, le ratio rendement/volatilité de la stratégie, ainsi que la performance de l'indice sur lequel se base la stratégie. La méthode reporte également dans un graphique dans le formulaire les performances historiques de la stratégie comparées à celles de son indice de référence.
  `btn_Optimize_Click` : Ce bouton permet de calculer, pour une stratégie donnée, pour des paramètres choisis par l'utilisateur, la période glissante n optimale sur laquelle la stratégie réalise le meilleur ratio rendement/volatilité. Pour le rechercher, on calcule successivement le ratio sur plusieurs périodes glissantes dans l'intervalle $[10; 300]$, avec un pas de 10, et on retient le n optimal. Il est ensuite reporté directement dans la zone de texte dédiée.

- Méthodes annexes :
  `RunModel` : Méthode qui lance le calcul du modèle. Il récupère les données financières depuis le fichier Data STOXX.csv et instancie un objet `MatriceFinanciere` contenant les données et dont nous expliciterons les attributs et méthodes plus bas. Ensuite, en fonction des paramètres choisis, est instancié un objet stratégie qui dépend de ces derniers. L'instanciation de cet objet stratégie lance automatiquement le calcul de toutes les statistiques liées à celle-ci et nous permet de reporter directement les

résultats à l'utilisateur dans les zones de texte dédiées. Nous présentons plus bas ces objets stratégie.

Matrice

Cette classe permet de traiter des données massives. Elle possède un constructeur simple, un constructeur à partir d'un nombre de lignes et de colonnes, et enfin un constructeur à partir d'un array à deux dimensions. Elle possède comme attributs NbLignes et NbColonnes et un array de données à deux dimensions.

Cette classe contient la première `DataBruteToMatrice` méthode qui nous permet de traiter les données brutes extraites du fichier csv : celles-ci sont ainsi stockées dans l'array à deux dimensions. La fonction `SubMatrix` est un outil qui nous permet d'extraire une matrice d'une autre, en spécifiant les indices de début et de fin en ligne et en colonne.

MatriceFinanciere

Cette classe est héritée de la classe `Matrice` et contient des méthodes et fonctions propres à des données financières. Dans un premier temps, il nous a fallu convertir les données extraites au format string en double.

- La fonction `ConvertDataToDouble` permet ainsi de convertir les données contenues dans l'array à deux dimensions en valeurs décimales.
- La méthode `SeparateData` permet de séparer les données extraites en 3 tableaux : un pour le cours des titres du STOXX 50, un autre pour récupérer les en-têtes, et un dernier pour garder en mémoire les dates.
- La méthode `ComputeReturns` permet de calculer la performance titre par titre, la performance de l'indice équipondéré, ainsi que la performance cumulée, et ce, selon le modèle suivant :
  Performance du titre i à la date t :

$$r_{i,t} = \frac{p_t}{p_{t-1}} - 1$$

  Performance de la poche composée des k titres à la date t :

$$r_{p,t} = \frac{1}{n}\sum_{i=1}^{k} r_{i,t}$$

  Performance cumulée de la poche à la date t :

$$r_{p,t}^{cum} = \left(1 + r_{p,t-1}^{cum}\right)\left(1 + r_{p,t}\right) - 1$$

- Les fonctions `Avg` et `Volatility` nous permettent de calculer aisément des moyennes et des volatilités, des métriques dont nous avons au besoin à de nombreuses reprises. Plus précisément, ces fonctions prennent en argument un array de données, ainsi que des indices de début et de fin en ligne et en colonne, qui délimitent la sous-plage de données sur laquelle on calcule la métrique. La délimitation des indices sur lesquels calculer ces métriques nous est utile par la suite lorsque nous calculons des moyennes et volatilités sur des périodes glissantes.

Strategy

Cette classe représente le cœur du projet. Nous avons implémenté le calcul de trois types de stratégies : moyenne mobile, bandes de Bollinger, et RSI. Pour ce faire, nous avons donc dans un premier temps crée une classe `Strategy` qui réunit les attributs et méthodes communs aux trois stratégies.

Il s'agit notamment des différents tableaux de performances, de la volatilité totale, et des méthodes pour lesquelles le calcul de ces derniers est similaire. Il s'agit notamment des méthodes :

- `ComputeReturn` : calcul des rendements de la stratégie de chaque actif ;
- `ComputeCumulativeReturn` : calcul des rendements cumulés de la stratégie de chaque actif ;
- `ComputeTotalReturn` : calcul de la performance du portefeuille équipondéré à partir de la performance de la stratégie de chaque actif ;
- `ComputeTotalCumulativeReturn` : calcul de la performance cumulée du portefeuille à partir de la performance de celui-ci ;
- `ComputeTotalVol` : calcul de la volatilité du portefeuille sur toute la période.

Nous avons ensuite créé des classes héritées qui permettent d'intégrer les particularités des différentes stratégies que nous avons implémentées. De manière globale, l'implémentation d'une stratégie de trading consiste en le calcul d'une métrique, puis d'un signal défini par la stratégie. Par exemple pour une stratégie Moyenne Mobile Momentum (MMM), le signal d'achat intervient lorsque le cours de la veille dépasse la moyenne mobile. Nous expliquons plus en détails chaque stratégie.

La classe `MovingAvg`

Cette stratégie a pour premier paramètre une période n qui définit la durée en jours sur laquelle on calcule une moyenne du cours, et que l'on fait glisser dans le temps. Le second paramètre est un booléen dans lequel on indique si la stratégie est de type Momentum ou Contrariante, ce qui influe sur le signal d'achat. Ci-dessous se trouvent les différentes étapes du calcul.

La méthode `ComputeMovAvg` permet le calcul de la moyenne mobile sur n jours à la date t, avec P_t le prix de l'actif à la date t, comme suit :

$$MM_t^n = \frac{1}{n}\sum_{i=1}^{n} P_{t-i}$$

La méthode `ComputeSignal` permet le calcul du signal d'achat, qui se matérialise par un tableau de nombres binaires, avec pour chaque actif, un 1 lorsque la stratégie suggère l'achat, et 0 pour la vente. Le calcul du signal d'une stratégie MMM à la date t est le suivant :

$$S_t^{MMM} = \begin{cases} 1 \; si \; P_{t-1} > MM_{t-1}^n \; et \; t > n+1 \\ 0 \; sinon \end{cases}$$

Pour une stratégie contrariante, c'est l'inverse : l'achat se produit lorsque le cours de la veille est en-dessous de sa moyenne mobile :

$$S_t^{MMC} = \begin{cases} 1 \; si \; P_{t-1} < MM_{t-1}^n \; et \; t > n+1 \\ 0 \; sinon \end{cases}$$

La classe `Bollinger`

Cette stratégie est une extension de la stratégie `MovingAvg` : on intègre au calcul de la moyenne mobile sur n périodes le calcul d'une volatilité, elle aussi glissante. En ajoutant et soustrayant donc cette volatilité à la moyenne mobile, on obtient alors les bandes de Bollinger. Tout comme la stratégie `MovingAvg`, celle-ci prend en paramètres un n et un booléen qui indique s'il s'agit d'une stratégie

Momentum ou Contrariante. Un paramètre vient s'ajouter : celui du nombre d'écarts-types q que l'on ajoute afin de créer les bandes de Bollinger.

La méthode `ComputeMovAvg` calcule la moyenne mobile glissante comme le fait la stratégie `MovingAvg`. `ComputeBands` permet ensuite le calcul des bandes supérieure $BB_t^{H,n}$ et inférieure $BB_t^{B,n}$ en intégrant l'écart-type glissant comme suit :

$$BB_t^{H,n} = MM_t^n + q\sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_{t-i} - MM_t^n)^2} \quad \text{et} \quad B_t^{B,n} = MM_t^n - q\sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_{t-i} - MM_t^n)^2}$$

Le calcul du signal `ComputeSignal` se fait comme suit :

$$S_t^{BBM} = \begin{cases} 1 \; si \; P_{t-1} > BB_{t-1}^{H,n} \; et \; t > n+1 \\ S_{t-1}^{BBM} \; si \; P_{t-1} \in \left[BB_{t-1}^{B,n}; BB_{t-1}^{H,n}\right] \; et \; t > n+1 \\ 0 \; sinon \end{cases}$$

$$S_t^{BBC} = \begin{cases} 1 \; si \; P_{t-1} < BB_{t-1}^{B,n} \; et \; t > n+1 \\ S_{t-1}^{BBC} \; si \; P_{t-1} \in \left[BB_{t-1}^{B,n}; BB_{t-1}^{H,n}\right] \; et \; t > n+1 \\ 0 \; sinon \end{cases}$$

La classe `RSI`

Cette stratégie a pour paramètres un n représentant la période glissante sur laquelle nous calculons l'indicateur, ainsi que des valeurs comprises entre 0 et 100 qui représentent les seuils pour lesquels les signaux d'achat et de vente ont lieu. Le calcul de l'indicateur RSI se décompose en plusieurs étapes dans les méthodes `ComputePnL` et `ComputeRSI`. Ces étapes sont les suivantes :

Calcul de la variation arithmétique du prix :

$$\Delta P_t = P_t - P_{t-1}$$

Ensuite, on extrait des séries de gains et de pertes définies comme suit :

$$G_t = \begin{cases} \Delta P_t \; si \; \Delta P_t > 0 \\ 0 \; sinon \end{cases} \quad \text{et} \quad L_t = \begin{cases} \Delta P_t \; si \; \Delta P_t < 0 \\ 0 \; sinon \end{cases}$$

On calcule ensuite une moyenne mobile sur n périodes des séries G et L, nous obtenons alors les séries $\bar{G} \; et \; \bar{L}$.

Le calcul du RSI se fait comme suit :

$$RSI_t = 100 - \frac{100}{\left(1 + \frac{\bar{G}_t}{\bar{L}_t}\right)}$$

Le calcul du signal (méthode `ComputeSignal`) se fait donc à partir des seuils h et l choisis par l'utilisateur (par défaut, les usuels 70 et 30), et selon le calcul suivant :

$$S_t^{RSI} = \begin{cases} 0 \; si \; RSI_{t-1} < h \; et \; RSI_t > h \; ou \; t > n+1 \\ 1 \; si \; RSI_{t-1} < l \; et \; RSI_t > l \; et \; t > n+1 \\ S_{t-1}^{RSI} \; sinon \end{cases}$$