Université Paris 1
Master MMMEF
Parcours Finance Quantitative

Sujet : Récupération de données de transactions et optimisation stratégie de trading

Projet de programmation en C++

Réalisé par :

Christopher Clemente Pires

Contents

Chapter 1

Introduction

Nous sommes heureux de vous présenter notre projet innovant en C++, un outil avant-gardiste conçu pour révolutionner l'analyse et la gestion des données financières.

Notre système, développé en tirant parti des connaissances et des compétences acquises en classe, est capable de traiter et d'analyser une variété de transactions de marché - **BUY**, **SELL** et **SELL SHORT** - en les extrayant d'un fichier CSV. Chaque transaction est minutieusement détaillée avec des informations essentielles telles que le *vendeur*, *l'acheteur*, la *quantité* et le *prix unitaire*, et est intégrée dans un système dynamique de gestion de portefeuille.

Pour enrichir notre projet et garantir une expérience complète, nous avons également programmé en C++ la génération du fichier CSV contenant toutes les données. Ce processus automatisé crée un ensemble réaliste de données de transaction, offrant ainsi une base solide pour tester et démontrer les capacités de notre outil de gestion et d'analyse.

Notre système va au-delà de la simple gestion de portefeuille. Il comprend un résolveur de transactions polyvalent qui fournit non seulement une interface pour la manipulation des données financières, mais aussi des statistiques détaillées et en temps réel sur les transactions, telles que le *volume*, le *type* et le *montant total* des transactions. Notre objectif est de fournir une solution intuitive pour les données financières, permettant une analyse statistique approfondie et instantanée des tendances du marché. Nous utilisons également un genetic algorithm pour optimiser une stratégie de trading à partir de notre jeu de données.

Nous vous guiderons à travers l'architecture de notre solution, en mettant en évidence les défis que nous avons rencontrés et surmontés avec détermination.Nous avons utilisé de manière optimale les outils enseignés en cours, comme les *vectors*, *maps* et *sets*, et avons intégré des mesures de performance pour chaque étape critique du programme, de la lecture des fichiers à la transformation et au traitement des données.

Ce projet n'est pas seulement une démonstration de compétences techniques en C++,

mais aussi une exploration passionnante des possibilités offertes par la programmation dans les domaines réels de la finance et de la gestion des données.

Nous sommes impatients de partager avec vous les résultats de notre travail, nos découvertes et notre goût pour l'innovation en programmation.

Chapter 2

Structure du Projet

Cinq classes auront été créées lors de ce projet, et sont liées entre elles dans le but d'extraire des données, les utiliser pour effectuer des calculs, et ainsi afficher les résultats souhaités.

Ces classes opèrent en synergie pour former un système permettant d'afficher des statistiques sur les transactions, mais également pour modifier le gestionnaire de portefeuille.

2.1 Construction du Fichier CSV

La génération du fichier CSV est un processus crucial dans notre projet, car elle fournit les données de base pour toutes les analyses ultérieures. Le fichier est créé par un script en C++ qui simule des transactions financières, illustrant une approche systématique et automatisée pour la création de données réalistes.

Principes de Génération

Le script C++ s'appuie sur des principes de programmation modernes, en utilisant des générateurs de nombres aléatoires et des fonctions spécifiques pour créer des transactions financières variées. L'objectif est de simuler un ensemble de données qui ressemble à de véritables transactions sur le marché, avec des variations dans les vendeurs, les acheteurs, les types de transactions, les quantités et les prix.

Structure et Fonctionnement

Le code commence par inclure des bibliothèques essentielles pour la manipulation de fichiers, la génération aléatoire et la gestion des chaînes de caractères. Deux fonctions principales sont définies : une pour générer les identifiants des vendeurs ou acheteurs, et une autre pour choisir aléatoirement le type de transaction. Le script procède ensuite à l'initialisation des paramètres de simulation, comme le nombre total d'entrées et les plages de valeurs pour les quantités et les prix.

Processus de Création des Données

Dans le corps principal du script, une boucle itère pour créer chaque transaction. Pour chaque itération, le script génère des identifiants uniques pour les vendeurs et les acheteurs, détermine aléatoirement le type de transaction, et assigne des quantités et des prix unitaires basés sur des distributions aléatoires. Ces détails sont ensuite formatés et écrits dans un fichier CSV, construisant progressivement un ensemble de données complet.

Conclusion du Processus

Une fois toutes les transactions générées et enregistrées, le script clôt le fichier CSV et affiche un message de confirmation. Ce fichier CSV généré devient la pierre angulaire pour les étapes suivantes du projet, fournissant une base de données riche pour l'analyse et la manipulation par les autres classes du système.

Ce processus démontre une intégration efficace de diverses techniques de programmation en C++ pour simuler un environnement de marché financier, en préparant le terrain pour une exploration approfondie des données de transaction dans les phases suivantes du projet.

2.2 Classe Transaction

La classe `Transaction` joue un rôle central dans la représentation des opérations financières au sein du projet. Chaque instance de cette classe incarne une transaction individuelle, encapsulant tous ses détails essentiels.

Structure et Fonctionnalités

- **Attributs** : La classe stocke des informations clés comme le vendeur (`seller`), l'acheteur (`buyer`), la quantité échangée (`quantity`), le prix unitaire (`unitPrice`) et le type de transaction (`type`, pouvant être Achat, Vente ou Vente à découvert).
- **Constructeur** : Le constructeur initialise une transaction avec les détails fournis, établissant ainsi les fondements de chaque opération financière.
- **Calcul du Prix Total** : La méthode `getTotalPrice` calcule le prix total de la transaction en multipliant la quantité par le prix unitaire, fournissant une évaluation financière immédiate de la transaction.
- **Accès aux Données** : Des méthodes d'accès (getters) permettent de récupérer les informations de chaque transaction, facilitant l'analyse et la manipulation des données.

Rôle dans le Projet

La classe `Transaction` sert de pierre angulaire pour la modélisation des transactions financières, offrant une structure détaillée et modulaire. Elle permet non seulement une représentation fidèle des opérations mais aussi facilite des calculs financiers complexes et des analyses de tendances du marché. Sa conception orientée objet assure une intégration aisée et efficace dans l'architecture globale du projet.

2.3 Classe TransactionReader

La classe `TransactionReader` joue un rôle crucial dans le projet, étant chargée de lire et d'analyser les données des transactions à partir d'un fichier CSV. Elle sert de pont entre les données brutes et leur représentation structurée dans le système.

Fonctionnement et Méthodes

- **Analyse des Lignes CSV** : La méthode `parseLine` est au cœur de cette classe. Elle prend chaque ligne du fichier CSV et la décompose en champs distincts (vendeur, acheteur, quantité, prix unitaire, type de transaction), en s'assurant de leur validité et cohérence. En cas d'incohérences ou de valeurs invalides, des exceptions sont générées pour signaler des erreurs précises.
- **Lecture Complète du Fichier CSV** : La méthode `readTransactions` ouvre le fichier CSV spécifié et lit chaque ligne, à l'exception de l'en-tête. Chaque ligne est passée à la méthode `parseLine` pour créer une instance de la classe `Transaction`, qui est ensuite stockée dans un conteneur de transactions.
- **Gestion des Exceptions** : Cette classe gère avec soin les erreurs de lecture et d'analyse des données, permettant au système de réagir de manière appropriée en cas d'informations manquantes ou erronées dans le fichier CSV.

Importance dans le Projet

Le rôle de `TransactionReader` est fondamental pour assurer l'intégrité et la précision des données utilisées dans le projet. En transformant les données textuelles brutes en objets `Transaction` bien structurés, cette classe facilite grandement le traitement et l'analyse ultérieurs des transactions financières. Elle sert de base fiable pour la manipulation des données et permet une intégration transparente avec d'autres composants du système.

2.4 Classe PortfolioManager

La classe `PortfolioManager` est au cœur de la gestion des portefeuilles des participants impliqués dans les transactions financières. Elle joue un rôle crucial en assurant que les portefeuilles soient tenus à jour après chaque transaction, et en fournissant une visualisation structurée et organisée des informations.

Fonctionnalités et Opérations

- **Mise à Jour du Portefeuille** : La méthode `updatePortfolio` ajuste les portefeuilles en fonction du type de transaction. Pour les achats (`BUY`), elle augmente la quantité de titres détenus par l'acheteur et la réduit pour le vendeur. Inversement, pour les ventes (`SELL`), elle augmente la quantité pour le vendeur et la diminue pour l'acheteur. Dans le cas des ventes à découvert (`SELL SHORT`), elle ajuste le portefeuille en conséquence.
- **Affichage du Portefeuille** : La méthode `displayPortfolio` trie et affiche l'état actuel des portefeuilles de chaque participant. Cette fonctionnalité est essentielle pour fournir une vue claire et concise de la position de chaque participant sur le marché financier.

Rôle dans le Projet

`PortfolioManager` est indispensable pour le suivi précis et la gestion des portefeuilles dans le cadre des transactions financières. Sa capacité à refléter fidèlement les mouvements du marché en temps réel fait de cette classe un outil puissant pour l'analyse financière et la prise de décision. En intégrant étroitement les données des transactions traitées par `TransactionResolver`, `PortfolioManager` assure une gestion dynamique et réactive des portefeuilles, adaptée aux fluctuations du marché.

2.5 Classe TransactionResolver

La classe `TransactionResolver` est dédiée au traitement et à l'analyse des transactions financières. Elle joue un rôle clé dans le calcul des statistiques relatives aux transactions et interagit directement avec la classe `PortfolioManager` pour la mise à jour des portefeuilles.

Fonctionnement et Méthodes

- **Traitement des Transactions** : Grâce à la méthode `processTransaction`, cette classe met à jour les statistiques pour chaque type de transaction (achat,

vente, vente à découvert), et calcule le montant total impliqué dans toutes les transactions.
- **Interaction avec PortfolioManager** : Lors du traitement de chaque transaction, `TransactionResolver` utilise une référence à `PortfolioManager` pour mettre à jour le portefeuille en fonction du type et des détails de la transaction.
- **Affichage des Statistiques** : La méthode `displayStats` offre une vue d'ensemble des transactions en affichant les comptages et le montant total des transactions par type, facilitant ainsi une compréhension rapide de l'activité du marché.
- **Détails des Transactions** : La méthode `displayTransactionDetails` permet d'examiner chaque transaction individuellement, fournissant des informations détaillées telles que le vendeur, l'acheteur, la quantité, le prix unitaire, et le type de transaction.

Rôle dans le Projet

`TransactionResolver` est essentiel pour la gestion efficace des données de transaction et l'analyse financière dans le projet. En traitant et en consolidant les informations sur les transactions, cette classe permet une analyse financière approfondie et contribue à la gestion stratégique des portefeuilles. Son intégration étroite avec `PortfolioManager` garantit que les mises à jour des portefeuilles sont à la fois précises et opportunes, reflétant fidèlement les dynamiques du marché.

2.6 Classe Timer

La classe `Timer` est essentielle pour mesurer le temps de traitement des différentes phases du projet, comme la lecture des fichiers et le traitement des données.

Caractéristiques Clés

- Utilise `std::chrono::high_resolution_clock` pour une mesure précise du temps.
- Le constructeur initialise le timer, et le destructeur calcule et affiche la durée écoulée.

Avantages

- Permet l'analyse et l'optimisation des performances de diverses sections du code.
- Simple et modulaire, facilitant son intégration et son utilisation dans le projet.

Cette classe offre un outil efficace pour le suivi des performances, aidant à identifier les goulots d'étranglement et à optimiser le code pour une meilleure efficacité.

2.7 Classe TradingStrategy

Cette classe représente une stratégie de trading simplifiée, on y définit un seuil de prix 'priceThreshold' (ici nous avons pris la valeur moyenne des UnitPrice de notre fichier CSV), en dessous ou au-dessus duquel la stratégie déclenchera un achat ou une vente. Pour trouver la meilleure strategie possible, on calcule le fitness, qui détermine à quel point la stratégie est rentable. Ainsi : pour une transaction de type BUY, la stratégie est récompensée lorsqu'elle effectue des achats à des prix inférieurs au seuil (qui est modifié pendant le processus), et le fitness est augmenté du prix de la transaction. Pour un type SELL, la stratégie est récompensée pour les ventes à des prix supérieurs au seuil. Nous avons ici simplifié la notion de SELL SHORT, le fitness est augmentée de la différence entre le seuil de prix et le prix unitaire de la transaction, la stratégie est donc récompensée de la même manière que pour le SELL. On utilise pour cette optimisation de stratégie un genetic algorithm, notre fonction crossover permettant d'effectuer le croisement entre deux stratégies parentes, à partir duquel nous créeons une nouvelle stratégie enfant. Ici, on calcule la moyenne des seuils de prix des deux parents pour déterminer celui de l'enfant.

2.8 Fonction Principale - main

La fonction principale, `main`, orchestre l'utilisation de toutes les classes définies dans le projet pour coordonner le traitement complet des transactions financières. Elle est le point d'entrée du programme et assure l'intégration de toutes les composantes.

Structure et Fonctionnement

- **Gestion des Exceptions** : Un bloc Try-Catch est utilisé pour capturer et gérer les exceptions. Toute erreur survenue dans le bloc Try est interceptée par le bloc Catch, où elle est traitée de manière appropriée, assurant la robustesse du programme.
- **Chargement des Transactions** : La fonction `loadTransactions` est appelée pour lire les transactions depuis un fichier CSV. Le temps de lecture est mesuré grâce à la classe `Timer`, fournissant des informations précieuses sur les performances du processus de lecture.
- **Traitement des Transactions** : Après le chargement, chaque transaction est traitée par le `TransactionResolver`, qui est initialisé avec une référence à `PortfolioManager`. Cette étape comprend la mise à jour des statistiques de transaction et des portefeuilles.

- **Affichage des Résultats** : Le programme affiche ensuite les statistiques accumulées sur les transactions et les détails de chaque transaction traitée. Finalement, l'état des portefeuilles est présenté, offrant une vue complète de l'activité du marché.

Rôle dans le Projet

La fonction `main` est essentielle pour la coordination et l'exécution de l'ensemble du projet. Elle lie toutes les classes et fonctions ensemble, facilitant un flux de travail cohérent et structuré pour le traitement des données financières. Cette organisation méthodique permet une analyse complète et précise des transactions financières, depuis leur chargement initial jusqu'à l'affichage des informations finales.

Chapter 3

Défis et Solutions

Ce projet, investissement de longue haleine, fut pris très au sérieux et s'est avéré être un succès. Cependant, il a présenté des défis, mettant à l'épreuve nos compétences en codage. Nous avons surmonté ces obstacles avec détermination.

3.1 Traitement des erreurs

Le traitement des erreurs a été un axe majeur dans l'amélioration de notre projet, notamment dans la classe PortfolioManager, afin de vérifier qu'un vendeur ait suffisamment d'actions dans une transaction de type SELL, par exemple. Nous avons également ajouté des détails dans nos messages d'erreur pour rendre davantage compréhensibles les potentielles erreurs rencontrées. Dans la même logique, nous avons ajouté des tests pour s'assurer que les composants fonctionnaient correctement, notamment pour le parsing de fichiers.

3.2 Optimisation dans la lecture du fichier CSV

Dans le cas où l'on considérerait des fichiers CSV de taille importante, la performance de notre programme aurait pu être remise en jeu. Nous avons optimisé la méthode 'parseLine' dans notre classe TransactionReader en utilisant une boucle afin d'extraire les champs de la ligne CSV, réduisant ainsi nos appels à 'std::getline'.

3.3 Manque de flexibilité dans la classe Transaction

Notre classe 'Transaction' manquait de flexibilité, les membres de celle-ci étaient initialement sans accesseurs, ce qui limitait l'accès aux informations en dehors de la classe. Ainsi, en utilisant des getters, nous avons protégé les membres privés de modifications non autorisées, tout en permettant au reste du programme d'accéder aux informations nécessaires dans leur logique de traitement.

3.4 Amélioration de la modularité de la fonction 'main'

L'idée ici était de réduire au maximum le nombre de tâches nécessaires, en extrayant le code dans des classes séparées. La création de la classe 'Timer' a notamment permis de rendre cette fonction plus simple à lire. Nous avons également regroupé le traitement des erreurs : plutôt que de lancer une exception pour chaque erreur, on collecte les erreurs afin de les traiter de manière centralisée.

Chapter 4

Résultats et Comparaisons

Ce projet que nous avons conçu permet de gérer et d'analyser des transactions financières, à travers un fichier CSV. Après une lecture de ce programme, nous avons stocké les informations dans un vecteur afin de pouvoir les manipuler.

4.1 Introduction des Résultats

Cette section présente les résultats clés obtenus à partir de l'analyse approfondie des transactions financières de notre base de données. L'objectif était de comprendre les tendances et les patterns dans les transactions, et d'évaluer l'impact de celles-ci sur le portefeuille des participants.

4.2 Méthodologie d'Analyse

L'analyse a été réalisée en plusieurs étapes. Nous avons d'abord segmenté les transactions par type, puis calculé le montant total pour chaque catégorie. Des statistiques descriptives ont été générées pour offrir une vue d'ensemble des activités de transaction. Cette approche nous a permis de déceler des tendances et des anomalies au sein des données.

4.3 Présentation des Résultats

Les résultats obtenus sont présentés ci-dessous :

- Nombre total de transactions : X
- Montant total des transactions : Y euros
- Répartition des transactions par type
- Évolution du portefeuille des participants

Résumé

L'étude des transactions financières de notre base de données a permis de mettre en lumière des tendances clés et des comportements des participants. Ces informations sont cruciales pour la prise de décision stratégique et l'optimisation du portefeuille.

Une importante partie de notre projet est basée sur un algorithme génétique, afin d'étudier différentes stratégies de trading et de sélectionner la meilleure en étudiant leurs performances en fonction des transactions passées, et en itérant sur plusieures générations, en impliquant des mutations et des croisements des stratégies existantes. En résultat, nous obtenons un historique de l'ensemble des transactions réalisées, le nombre de chaque type de transactions, ainsi que le montant total de celle-ci. Pour la partie sur notre genetic algorithm, nous obtenons la meilleure stratégie possible, avec le seuil de prix optimal et son fitness (par exemple pour une exécution : seuil de prix = 455, fitness = 113261).

Chapter 5

Conclusion

Ce projet, à la fois ambitieux et complexe, a constitué un défi passionnant et enrichissant. Grâce à une intégration soignée de concepts avancés de programmation en C++, nous avons pu non seulement appliquer mais aussi approfondir les connaissances acquises en classe. Le cœur de notre réalisation, une application financière concrète, a brillamment mis en lumière la puissance et la polyvalence de la programmation en C++, démontrant notre capacité à transformer des concepts théoriques en solutions pratiques et efficaces.

Notre système a réussi à modéliser et à analyser des transactions financières complexes à travers un fichier CSV, en appliquant des méthodes de traitement des données robustes et des techniques de gestion de portefeuille sophistiquées. Nous avons relevé des défis techniques, notamment dans le traitement des erreurs et l'optimisation de la lecture des fichiers, tout en maintenant une structure de code claire et modulaire.

La capacité de notre application à générer, traiter et analyser des données financières réalistes, tout en gérant dynamiquement les portefeuilles des participants, témoigne de notre compréhension approfondie des principes de la programmation orientée objet et de notre compétence en matière de développement logiciel.