**Rapport Final sur les Jobs Spark**

**1. Résumé des résultats**

|  |  |  |
| --- | --- | --- |
| Méthode | Tina | Yann |
| Sans UDF | 6.2000 | 2.844 |
| UDF Python | 5.4187 | 2.819 |
| UDF Scala | 0.6247 | 0.054 |

**2. Analyse des performances**

**Spark est lazy (évaluation différée)**

Spark applique le principe d’évaluation différée (lazy evaluation). Cela signifie que les transformations sur les DataFrames/Datasets (comme withColumn, select, etc.) ne sont pas exécutées immédiatement. Au lieu de cela, Spark crée un plan d'exécution logique, mais les calculs réels ne sont déclenchés que lorsqu'une action est appelée, comme :  
- show()  
- collect()  
- write()  
  
Ce comportement permet à Spark :  
1. D'optimiser le plan d'exécution avant de le lancer.  
2. D'éliminer les étapes inutiles, comme des calculs intermédiaires non nécessaires.

**3. Comparaison des méthodes**

Sans UDF  
- Utilise uniquement les fonctions natives de Spark (when, sum, etc.).  
- Les calculs sont optimisés par Catalyst, mais les opérations sur des fenêtres (window functions) augmentent la complexité.  
- Temps : 6.2000 secondes.  
- Conclusion : Malgré les optimisations natives, les calculs complexes (comme les totaux sur 30 jours) rendent cette méthode plus lente.

UDF Python  
- Implémente une logique en Python via une User Defined Function (UDF).  
- Les UDF Python introduisent une surcharge due à la sérialisation/désérialisation entre Python et la JVM.  
- Temps : 5.4187 secondes.  
- Conclusion : Plus rapide que la méthode sans UDF, mais limitée par la communication entre Python et JVM.

UDF Scala  
- La logique est codée en Scala, fonctionnant directement dans la JVM.  
- Évite la surcharge de communication entre Python et JVM.  
- Profite pleinement des optimisations Catalyst.  
- Temps : 0.6247 secondes.  
- Conclusion : La méthode la plus performante, idéale pour des calculs intensifs et des volumes de données importants.

**4. Recommandations**

1. Priorité à la performance :  
- Utilisez UDF Scala pour des performances optimales sur des données volumineuses et des transformations complexes.  
2. Priorité à la simplicité :  
- Pour un développement rapide et une flexibilité, optez pour UDF Python si la performance n'est pas critique.  
3. Cas spécifiques :  
- Si les transformations peuvent être exprimées avec des fonctions Spark natives, utilisez la méthode Sans UDF pour maintenir la simplicité et minimiser la complexité.

**5. Visualisation des résultats**

Un graphique comparatif des temps d'exécution a été généré pour visualiser clairement les performances des trois méthodes. Le graphique montre que :  
- UDF Scala est nettement plus rapide.  
- UDF Python est un bon compromis entre simplicité et performance.  
- Sans UDF est la méthode la plus lente en raison de la complexité des calculs.



**6. Conclusion**

Spark, grâce à son évaluation différée (lazy evaluation), optimise les transformations pour réduire les temps d'exécution. Cependant, le choix de la méthode (Sans UDF, UDF Python, UDF Scala) dépend des priorités entre performance, flexibilité et simplicité.