📦 Ecommerce Analytics avec Spark & Scala

Projet académique de Master 1 Intelligence Artificielle (DIT)
Étudiant : Moussa Mallé
Email : mallemoussa091@gmail.com

📚 Sujet

Développer un système d'analyse de données e-commerce complet en utilisant Apache Spark avec Scala.

Objectifs :

- Ingestion de données multi-format (CSV, JSON, Parquet)
- Nettoyage et validation
- Enrichissement avancé via UDF et fonctions de fenêtrage
- Analyse business (KPI, cohortes)

- Optimisations Spark (cache, persist, broadcast)
- Export des résultats en CSV et Parquet

📁 Structure du projet

```
EcommerceAnalytics/
├── data/ # Fichiers d'entrée
├── output/ # Résultats finaux
├── src/
| └── main/
| └── scala/com/ecommerce/analytics/
| ├── MainApp.scala
| ├── DataIngestion.scala
| ├── DataTransformation.scala
| ├── Analytics.scala
| └── models.scala
├── application.conf # Configuration externe (
├── build.sbt # Fichier de build SBT
└── README.md # Ce document
```

🛠️ Prérequis

- Java 11
- Scala 2.11.12
- Spark 2.2.1
- SBT
- Spark (via dépendances `spark-core`, `spark-sql`)

⚙️ Configuration

Le fichier `application.conf` centralise tous les chemins d'entrée/sortie :

```
app {
  name = "EcommerceAnalytics"
  env = "dev"
  spark {
    master = "local[*]"
    appName = "EcommerceAnalyticsApp"
  }
  data {
    input {
      transactions = "data/transactions.csv"
      merchants = "data/merchants.csv"
      users = "data/users.json"
      products = "data/products.parquet"
    }
```

```
    output {
      path = "output/results"
    }
  }
}
```

🚀 Exécution

1. Cloner le projet :

```
git clone https://github.com/codeangel223/Data
cd EcommerceAnalytics
```

2. Placer les données dans le dossier `data/` .

3. Lancer :

```
sbt run
```

✨ Fonctionnalités Implémentées en Détail

Le projet couvre l'ensemble d'un pipeline analytique Spark, structuré en 7 parties, avec implémentation complète et optimisations professionnelles.

🔧 Partie 1 – Configuration & Structure du Projet

- Mise en place d'une **architecture modulaire** : `models`, `analytics`, `transformation`, `ingestion`, `app`
- Configuration de `build.sbt` :
  - Scala 2.11.12 compatible avec Spark 2.2.1
  - Ajout de `spark-core`, `spark-sql`, et `typesafe-config`
- Chargement des paramètres via un fichier `application.conf` propre

📥 Partie 2 – Ingestion & Validation des Données

- Lecture multi-format :

    - `CSV` : transactions & merchants
    - `JSON` : users
    - `Parquet` : products
- Création de `case class` typées pour chaque entité
- Validation métier :
    - Transactions : `amount > 0`, timestamp de 14 caractères
    - Users : `age ∈ [16, 100]`, `income > 0`
    - Products : `price > 0`, `rating ∈ [1, 5]`
    - Merchants : `commission_rate ∈ [0, 1]`
- Gestion d'erreurs :
    - Try/catch des erreurs de lecture
    - Affichage du nombre de lignes lues & valides

🧠 Partie 3 – Transformations Avancées

- **UDF extractTimeFeatures** : à partir d'un timestamp (`yyyyMMddHHmmss`), calcul des dimensions :
    - Heure, jour, mois, `is_weekend` , `is_working_hour` , `day_period`
- Enrichissement des transactions :
    - Jointure avec users, merchants, products
    - Ajout du `rang` de transaction par utilisateur (Window)
    - Comptage total de transactions
    - Classification par **tranche d'âge** : Jeune, Adulte, Âge Moyen, Senior
- Détection de comportements par **fenêtre glissante 7 jours** :
    - `active_days_7d` : nb de jours où un utilisateur a transigé
    - `is_active_user_7d` : flag si ≥ 5 jours actifs sur 7

📊 Partie 4 – Analytique Business

✅ Rapport KPI par Marchand

- Chiffre d'affaires total, nombre de clients uniques
- Montant moyen par transaction
- Commission totale perçue
- Classement par **CA dans sa catégorie et région** (avec `Window`)
- Répartition des ventes par tranche d'âge

📈 Analyse de Cohortes

- Identification du mois de **première transaction** (`cohort_month`)
- Calcul du taux de **rétention mensuelle**
- Construction d'un tableau croisé cohort_month × transaction_month

---

🚀 Partie 5 – Optimisations Spark

- **Gestion mémoire** :
  - `cache()` sur les DataFrame réutilisés
  - `persist(StorageLevel.MEMORY_AND_DISK_SER)` pour les grands volumes
  - `unpersist()` après utilisation
- **Optimisation des jointures** :
  - `broadcast()` sur `merchants` pour limiter le shuffle

🧩 Partie 6 – Application Principale

- `MainApp.scala` orchestre :
  - Ingestion, transformation, analyse
  - Affichage console des résultats
  - Sauvegarde en **CSV** & **Parquet**
- Configuration centralisée (`application.conf`)
- `try/catch/finally` pour robustesse

⚙️ Partie 7 – Configuration Externalisée

- Chemins des datasets & sortie définis dans `application.conf`
- Variables comme le `app.name` et `spark.master` modifiables facilement

Souhaites-tu que je mette à jour ton `README.md` avec cette nouvelle section enrichie ?

💾 Résultats

Les résultats sont enregistrés en double format :

```
output/results/
├── merchant_report/
|   ├── csv/
|   └── parquet/
└── cohort_report/
    ├── csv/
```

```
└── parquet/
```

📊 Exemple de sorties

KPI par marchand :

- Chiffre d'affaires total
- Nombre de clients uniques
- Montant moyen par transaction
- Commission totale perçue
- Répartition par tranche d'âge
- Rang dans sa catégorie et sa région

Analyse de cohortes :

- Premier mois d'achat de chaque utilisateur
- Retention mensuelle post-cohorte

👨‍🎓 Étudiant

- **Nom** : Moussa Mallé
- **Email** : mallemoussa091@gmail.com
- **Formation** : Master 1 Intelligence Artificielle – DIT

📄 Licence