Utilisation de Big Data avec Apache Spark en Python

Intégration des Données

Projet OpenFoodFacts

Introduction

Mise en place d'une solution ETL distribuée utilisant Apache Spark en Python, avec une base de données Sqlite pour répondre aux préoccupations des consommateurs en matière de programmes alimentaires qui seraient adaptés à leurs besoins. La sources des données massives proviendront d'OpenFoodFacts, et notre solution ETL générera des menus équilibrés en fonction des régimes alimentaires spécifiques pour chaque utilisateur et ses besoins, définis par des programmes alimentaires spécifiques..

Cahier des charges

Spécificités techniques

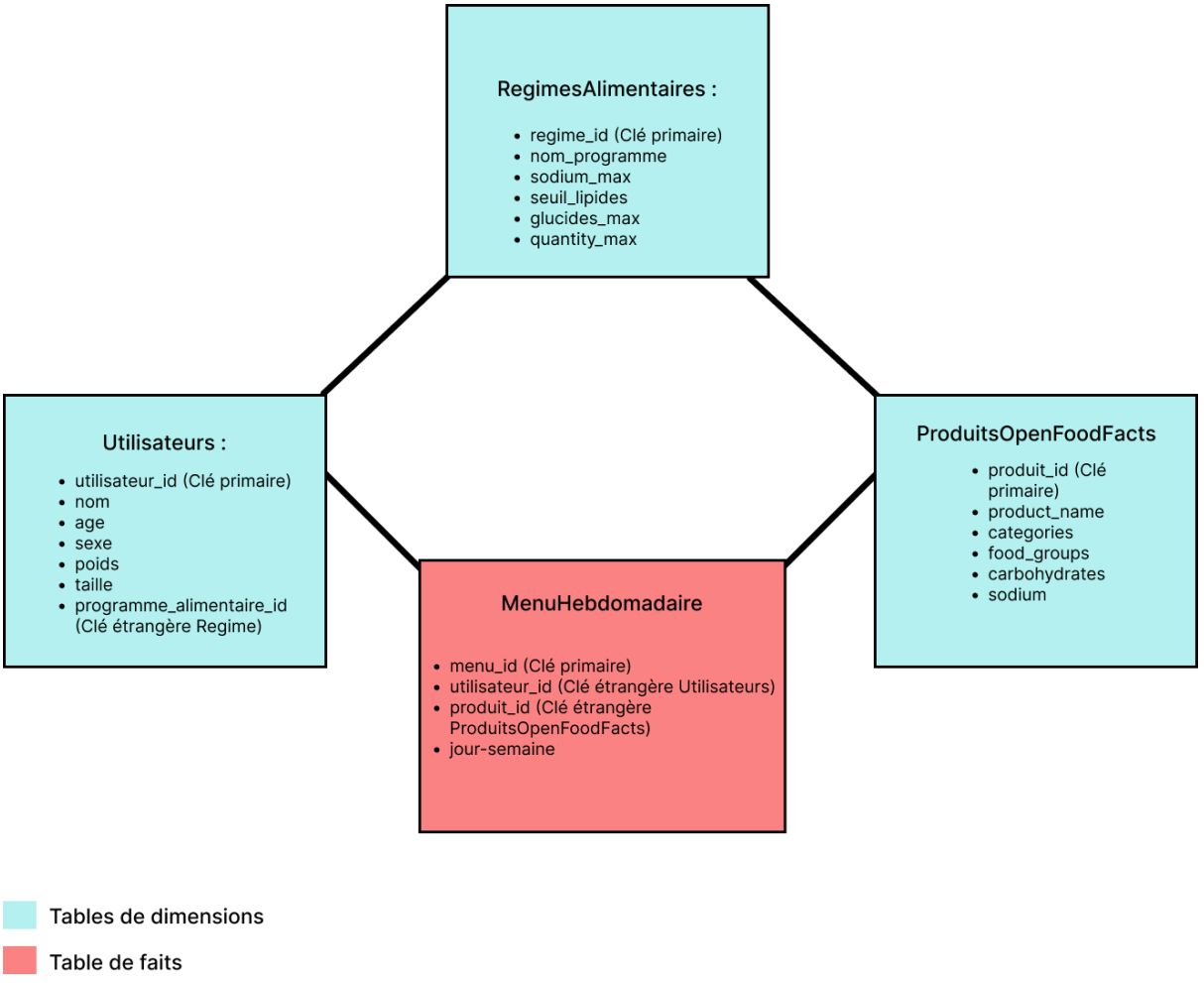
- **Source de données OpenFoodFacts:** https://fr.openfoodfacts.org/data
- **ETL avec Apache Spark en Python:** Utilisation de Spark pour les opérations distribuées.
- **Base de données Sqlite:** Utilisation de Sqlite3 comme Datawarehouse.

Langage de programmation

- **Python:** Langage principal pour la mise en œuvre de la solution ETL avec Apache Spark.
- **Pycharm:** L'environnement de développement (IDE) utilisé pour développer la solution ETL en Python

Modèle de Données

Le modèle de données conçu pour ce projet OpenFoodFacts suit une approche en étoile, offrant une structure claire et optimisée pour stocker les informations nécessaires à la génération de menus hebdomadaires équilibrés en fonction des besoins alimentaires spécifiques des utilisateurs. Voici une description des tables clés dans ce modèle :



Utilisation de l'ORM

L'utilisation d'un Object-Relational Mapping (ORM) dans mon projet de base de données apporte une approche efficace et simplifiée pour interagir avec la base de données relationnelle. Au lieu de rédiger des requêtes SQL directes, j'exploite l'ORM, en l'occurrence

SQLAlchemy, pour représenter mes tables sous forme de classes Python. Cette abstraction facilite la manipulation des données en utilisant des objets et des méthodes plutôt que des requêtes SQL complexes.

L'ORM agit comme une couche intermédiaire entre mon application Python et la base de données, facilitant ainsi les opérations CRUD (Create, Read, Update, Delete). Les classes Python que j'ai définies avec SQLAlchemy reflètent la structure de ma base de données, offrant une manière plus intuitive d'interagir avec les données. De plus, l'utilisation de l'ORM simplifie la gestion des relations entre les tables, grâce à des concepts tels que les clés étrangères et les relations entre objets.

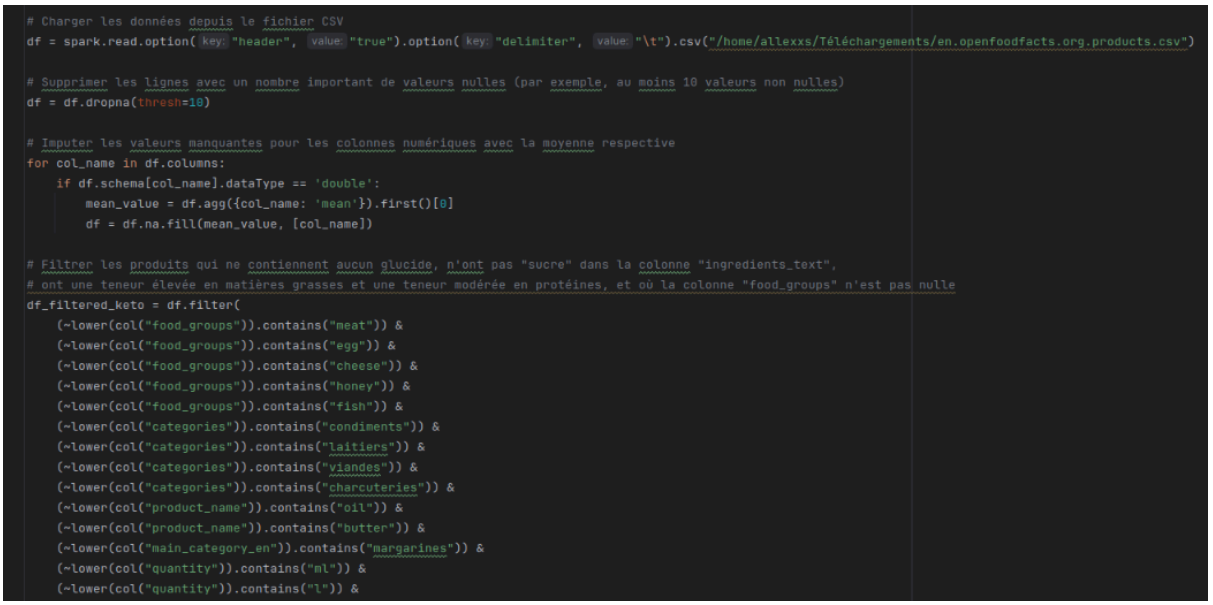
Architecture de la Solution

Elle collecte les données depuis OpenFoodFacts et les sources de données additionnelles, effectue des transformations pour générer les menus hebdomadaires, et charge les résultats dans un Data Warehouse (DWH) central.

Points Clés et explication détaillée dans le Code :

Gestion des Erreurs et Qualité des Données :

- Filtres ajoutés pour exclure les produits avec des données manquantes.
- Validation des données pour garantir la qualité des informations collectées par les régimes alimentaires .



```
# Charger les données depuis le fichier CSV
df = spark.read.option( key: "header",  value: "true").option( key: "delimiter",  value: "\t").csv("/home/allexxs/Téléchargements/en.openfoodfacts.org.products.csv")

# Supprimer les lignes avec un nombre important de valeurs nulles (par exemple, au moins 10 valeurs non nulles)
df = df.dropna(thresh=10)

# Imputer les valeurs manquantes pour les colonnes numériques avec la moyenne respective
for col_name in df.columns:
    if df.schema[col_name].dataType == 'double':
        mean_value = df.agg({col_name: 'mean'}).first()[0]
        df = df.na.fill(mean_value, [col_name])

# Filtrer les produits qui ne contiennent aucun glucide, n'ont pas "sucre" dans la colonne "ingredients_text",
# ont une teneur élevée en matières grasses et une teneur modérée en protéines, et où la colonne "food_groups" n'est pas nulle
df_filtered_keto = df.filter(
    (~lower(col("food_groups")).contains("meat")) &
    (~lower(col("food_groups")).contains("egg")) &
    (~lower(col("food_groups")).contains("cheese")) &
    (~lower(col("food_groups")).contains("honey")) &
    (~lower(col("food_groups")).contains("fish")) &
    (~lower(col("categories")).contains("condiments")) &
    (~lower(col("categories")).contains("laitiers")) &
    (~lower(col("categories")).contains("viandes")) &
    (~lower(col("categories")).contains("charcuteries")) &
    (~lower(col("product_name")).contains("oil")) &
    (~lower(col("product_name")).contains("butter")) &
    (~lower(col("main_category_en")).contains("margarines")) &
    (~lower(col("quantity")).contains("ml")) &
    (~lower(col("quantity")).contains("l")) &
```

Ajouts et transformation des menus de OpenFoodFack dans la Base de donnée:

- Sélection des plats et desserts parmi les produits filtrés.
- Construction de menus hebdomadaires équilibrés en fonction des besoins nutritionnels spécifiques définis par les utilisateurs.

```
ffficher les premières lignes du DataFrame après le nettoyage et le filtrage
nt("\nProduits adaptés au régime Végétarien :")
no_carbs_names.show( n: 20,truncate=False)
nitialiser la liste products_data avant la boucle

nitialiser la liste products_data avant la boucle
ducts_data = []
énérer 2 repas par jour pour 7 jours
 day in range(1, 2):
 print(f"\nJour {day} :")

 # Sélectionner aléatoirement un plat pour chaque repas
 plat1_row = df_plats.orderBy(F.rand()).limit(1).select("product_name", "categories", "quantity
 plat2_row = df_plats.orderBy(F.rand()).limit(1).select("product_name", "categories", "quantity

 # Sélectionner aléatoirement un dessert pour chaque repas
 dessert1_row = df_desserts.orderBy(F.rand()).limit(1).select("product_name", "categories", "qu
 dessert2_row = df_desserts.orderBy(F.rand()).limit(1).select("product_name", "categories", "qu

 plat1 = plat1_row["product_name"]
 plat1_categories = plat1_row["categories"]
 plat1_quantity = plat1_row["quantity"]
 plat1_food_groups = plat1_row["food_groups"]
 plat1_glucides = plat1_row["carbohydrates_100g"]
 plat1_sodium = plat1_row["sodium_100g"]

 plat2 = plat2_row["product_name"]
 plat2_categories = plat2_row["categories"]
 plat2_quantity = plat2_row["quantity"]
```

Collecte des Données :

- Utilisation de la bibliothèque `sqlite3` pour se connecter à la base de données SQLite contenant les données d'OpenFoodFacts.
- Écriture de requêtes SQL dynamiques pour filtrer les produits en fonction des régimes alimentaires définis.

```
# Récupérer les produits correspondant au régime alimentaire de l'utilisateur
cursor.execute( _sql: "SELECT * FROM ProduitsOpenFoodFacts WHERE categories = ?",  _parameters: (regime_alimentaire,))
produits = cursor.fetchall()
```

Proposition de plats et de desserts pour un repas :

```
# Séparer les produits en plats et desserts
plats = [produit for produit in produits if produit[3] == 'plat']
desserts = [produit for produit in produits if produit[3] == 'dessert']
```

Génération de 2 repas par jours pour un menu hebdomadaire (14 repas contenant un plats et dessert) :

```
# Générer les repas pour chaque jour
for jour in range(1, nb_jours + 1):
    print(f"\nJour {jour} :")
    for repas in range(1, nb_repas_par_jour + 1):
        # Sélectionner aléatoirement un plat et un dessert
        plat = random.choice(plats)
        dessert = random.choice(desserts)

        # Insérer le plat et le dessert pour l'utilisateur dans la table ProduitsUsers
        cursor.execute( _sql: "INSERT INTO ProduitsUsers (utilisateur_id, produit_id) VALUES (?, ?)",  _parameters: (utilisateur_id, plat[0]))
        cursor.execute( _sql: "INSERT INTO ProduitsUsers (utilisateur_id, produit_id) VALUES (?, ?)",  _parameters: (utilisateur_id, dessert[0]))

        print(f"  Repas {repas} : Plat - {plat[1]}, Dessert - {dessert[1]}")

# Commit des changements et fermeture de la connexion
conn.commit()
conn.close()
```

Variable à changer pour modifier la génération d'un menu hebdomadaire en fonction du régime alimentaire :

- Changer le régime alimentaire
- Changer le nombre de repas par jour
- Changer le nombre de jours pour un menu

```
# Exemple d'utilisation
utilisateur_id = 2
regime_alimentaire = "vegetarien" # Remplacez le regime alimentaire "vegetarien" ou "vegetalien"
nb_repas_par_jour = 2 # Choisir le nombre de repas par jour
nb_jours = 7 # Choisir le nombre de jour pour le programme

generer_repas(utilisateur_id, regime_alimentaire, nb_repas_par_jour, nb_jours)
```

Perspectives Futures

Améliorations Possibles

Pour enrichir cette solution, plusieurs améliorations pourraient être envisagées :

- **Personnalisation Avancée des Menus:** Offrir une personnalisation plus fine en tenant compte des préférences alimentaires spécifiques, des allergies et des restrictions diététiques.
- **Gestion Dynamique des Régimes:** Permettre aux utilisateurs de modifier dynamiquement leur régime alimentaire en fonction de leurs besoins évolutifs.

Intégration de l'Intelligence Artificielle (IA)

- **Modèles Prédictifs et Algorithmes de Machine Learning:** Utilisation de PyTorch et d'algorithmes de machine learning pour anticiper les préférences des utilisateurs, ajuster automatiquement les menus et optimiser la génération de menus en fonction des retours des utilisateurs.
- **Recommandations Personnalisées:** Développement d'algorithmes de recommandation pour suggérer des aliments et des plats en fonction des préférences historiques de l'utilisateur.

Outils et Technologies

- **Intégration de Solutions de BI:** L'intégration de solutions de Business Intelligence (BI) telles que PowerBI pour la visualisation avancée des données, permettant aux utilisateurs de suivre leur consommation nutritionnelle au fil du temps et d'explorer visuellement les tendances.