INFO
INFORMATIQUE

IUT ANNECY
UNIVERSITÉ SAVOIE MONT BLANC

SAE-C-4.01

Développement avec une base de données et visualisation

09 AVRIL 2024

BACHAR Bilal | DEDDOUCHE Yanisse | MIHOUBI Marouane | RABEHI Milhane

INFO
INFORMATIQUE

IUT ANNECY UNIVERSITÉ SAVOIE MONT BLANC

Table des matières

I) Amélioration de la base de données :

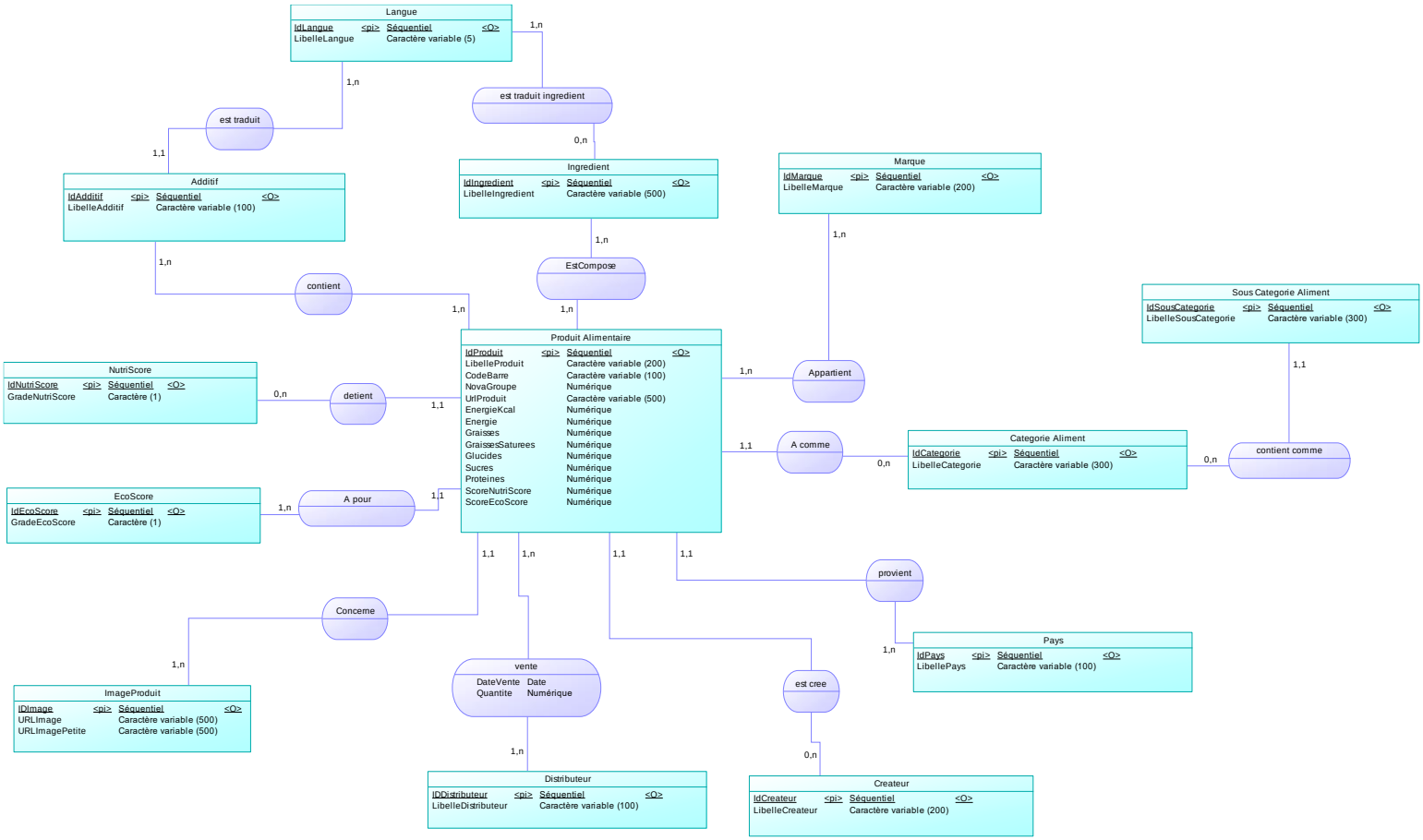
Pour commencer, le fichier CSV fourni avec le sujet a été traité pour assurer sa qualité. Les changements effectués comprenaient le nettoyage du fichier, impliquant l'élimination des lignes vides ou mal écrites. Ces actions ont été réalisées dans le but de garantir la cohérence et la fiabilité des données. Après ce processus de nettoyage, le fichier csv contenait 62 413 lignes, prêtes à être utilisées pour être analyser et inséré dans la nouvelle base de données.

I.I) Nouvelle base de données OLTP :

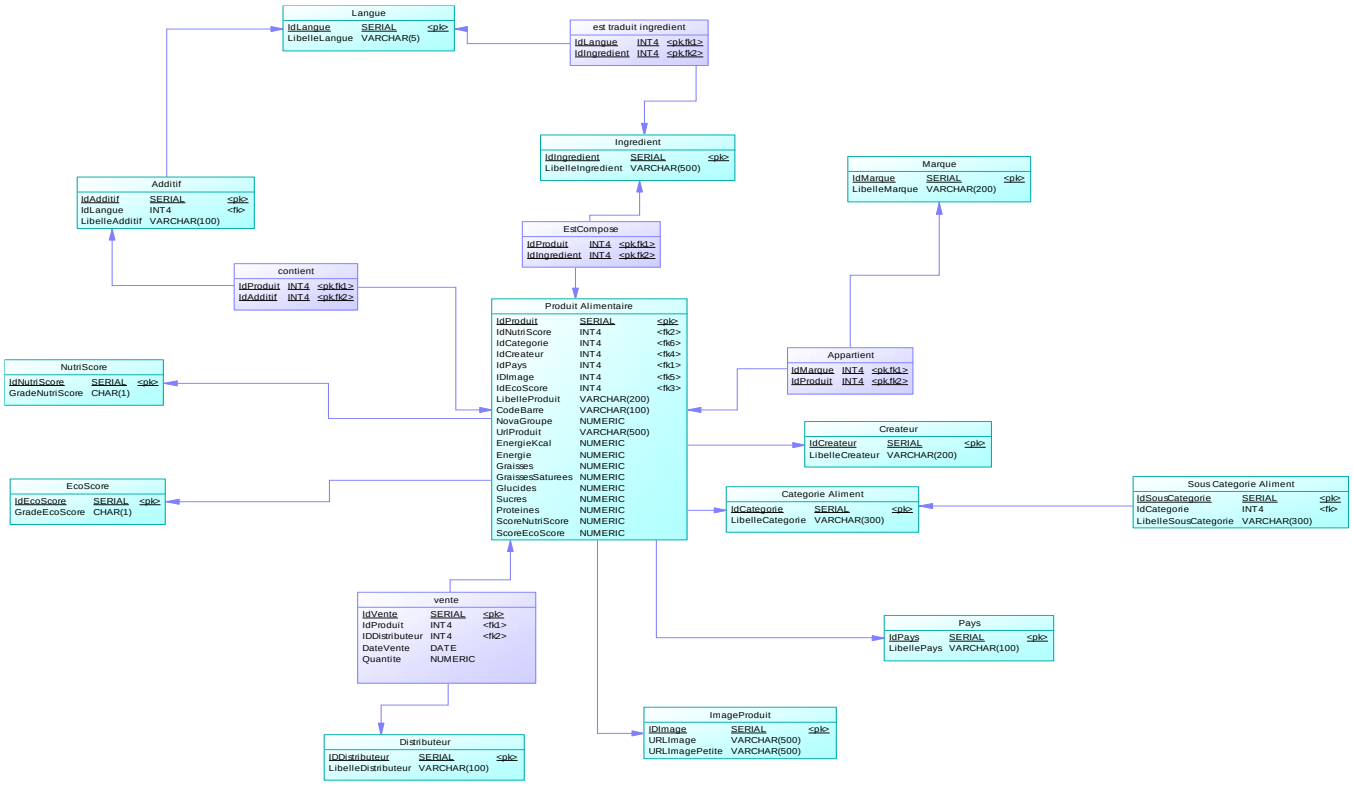
Pour débuter la conception du modèle conceptuel de données (MCD), la création des tables se basait sur les informations fournies dans le fichier CSV. Cette démarche s'est appuyée sur les intitulés des colonnes ainsi que sur les données disponibles dans le fichier. En conséquence, les tables suivantes ont été identifiées :

- Marque
- ProduitAlimentaire
- Additif
- Fournisseur
- SousCategorieAliment
- CategorieAliment
- Pays
- Createur
- Ingredient
- Langue
- EcoScore
- NutriScore

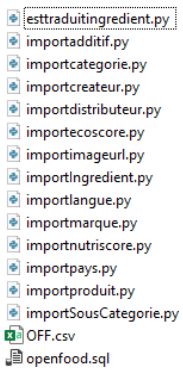
Par la suite, conformément aux directives du sujet, qui exprimait le désir d'OFF d'étendre sa base de données PostgreSQL afin de stocker, pour chaque produit, les volumes de ventes observés à différentes périodes et pour différents distributeurs, une nouvelle table nommée "Distributeur" a été créée. Cette table est en relation avec la table "ProduitAlimentaire". Des attributs tels qu'une date et une quantité ont été ajoutés à cette association pour permettre le stockage des volumes de ventes observés pour chaque produit, à différentes périodes et pour différents distributeurs.



Cela a conduit à la création du modèle logique de données suivant, où un ID vente a été ajouté à la table "Vente".



Ensuite, les données ont été insérées dans notre base pgAdmin de manière à ce que chaque table dispose de son propre script Python pour extraire les données du fichier CSV :



Exemple pour la table additif :

```python
import csv
import psycopg2

def clean_additive(additive_str):
    # Retire les prefixes des additifs
    additives_list = additive_str.split(',')
    cleaned_additives = [additive.split(':')[-1] for additive in additives_list if ':' in additive]
    return ','.join(cleaned_additives)

def insert_additives(cursor, csv_file):
    with open(csv_file, newline='', encoding='utf-8') as csvfile:
        reader = csv.DictReader(csvfile, delimiter=';')
        for row in reader:
            libelle_langue = row['Langue']  # Recuperer le libelle de la langue a partir du CSV
            libelle_additif = row['LibelleAdditif']  # Recuperer le libelle de l'additif a partir du CSV

            # Nettoyage des additifs (retrait des prefixes)
            clean_additives_str = clean_additive(libelle_additif)

            # Recherche de l'ID de la langue correspondant au libelle dans la table LANGUE
            cursor.execute("SELECT IDLANGUE FROM LANGUE WHERE LIBELLELANGUE = %s", (libelle_langue,))
            langue_result = cursor.fetchone()
            if langue_result:
                id_langue = langue_result[0]

                # Inserer les additifs avec l'ID de langue correspondant
                cursor.execute("INSERT INTO ADDITIF (IDLANGUE, LIBELLEADDITIF) VALUES (%s, %s) ON CONFLICT DO NOTHING", (id_langue, clean_additives_str))
            else:
                print(f"La langue '{libelle_langue}' n'existe pas dans la table LANGUE.")
```

```python
# Connexion a la base de donnees PostgreSQL
def connect_to_database():
    try:
        connection = psycopg2.connect(
            dbname="OFF",
            user="postgres",
            password="postgres",
            host="localhost"
        )
        cursor = connection.cursor()
        return connection, cursor
    except psycopg2.Error as e:
        print("Erreur lors de la connexion a la base de donnees PostgreSQL:", e)

# Fermeture de la connexion a la base de donnees PostgreSQL
def close_connection(connection, cursor):
    if cursor:
        cursor.close()
    if connection:
        connection.close()

# Exemple d'utilisation
if __name__ == "__main__":
    connection, cursor = connect_to_database()
    if connection and cursor:
        try:
            insert_additives(cursor, "P:/Parcours C/sae/Insertions/Insertions/test/OFF.csv")
            connection.commit()
            print("Les additifs ont ete inseres avec succes.")
        except psycopg2.Error as e:
            print("Erreur lors de l'insertion des additifs dans PostgreSQL:", e)
        finally:
            close_connection(connection, cursor)
```

Pour la table Distributeur, des données « brutes » ont été insérées, étant donné qu'il n'y avait pas de données relatives aux distributeurs dans le fichier CSV :

```python
import psycopg2

def insert_distributeurs(cursor):
    distributeurs = [
        "Auchan",
        "Carrefour",
        "Costco",
        "Cora",
        "Match",
        "Leclerc",
        "Geant Casino",
        "Hyper Casino",
        "Hyper U",
        "Intermarche Hyper"
    ]

    for distributeur in distributeurs:
        try:
            cursor.execute("INSERT INTO DISTRIBUTEUR (LIBELLEDISTRIBUTEUR) VALUES (%s)", (distributeur,))
            print(f"Le distributeur '{distributeur}' a été inséré avec succès.")
        except psycopg2.Error as e:
            print("Erreur lors de l'insertion du distributeur:", e)

# Connexion à la base de données PostgreSQL
def connect_to_database():
    try:
        connection = psycopg2.connect(
            dbname="OFF",
            user="postgres",
            password="postgres",
            host="localhost"
        )
        cursor = connection.cursor()
        return connection, cursor
    except psycopg2.Error as e:
        print("Erreur lors de la connexion à la base de données PostgreSQL:", e)

```

```python
# Fermeture de la connexion à la base de données PostgreSQL
def close_connection(connection, cursor):
    if cursor:
        cursor.close()
    if connection:
        connection.close()

# Exemple d'utilisation
if __name__ == "__main__":
    connection, cursor = connect_to_database()
    if connection and cursor:
        try:
            insert_distributeurs(cursor)
            connection.commit()
            print("Les données ont été insérées avec succès.")
        except psycopg2.Error as e:
            print("Erreur lors de l'insertion des données dans PostgreSQL:", e)
        finally:
            close_connection(connection, cursor)
```

Les ventes, qui ne sont pas présentes dans le fichier CSV et doivent être simulées, il faut donc un script Python. Ce script génère des données de vente simulées pour des produits alimentaires en utilisant les informations préexistantes sur les produits et les distributeurs stockées dans la base de données.

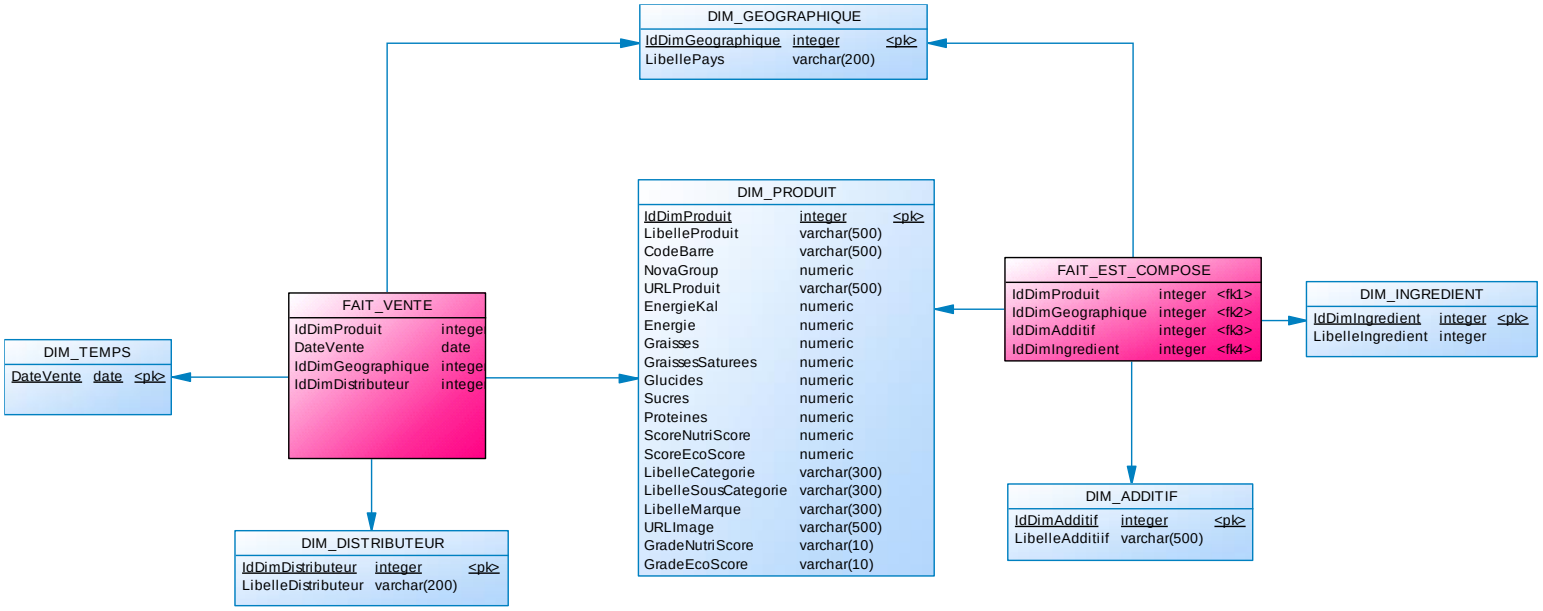
Pour chaque produit, le script simule un certain nombre de ventes en se basant sur une moyenne et un écart type préétablis. Il choisit ensuite aléatoirement un distributeur et une date de vente pour chaque transaction simulée. Ces sélections sont déterminées en fonction de probabilités prédéfinies pour les mois et les distributeurs. Après la simulation de chaque vente, le script insère les données correspondantes dans la base de données PostgreSQL. Voici le code :

```python
def InsertSales(cursor):
    cursor.execute("SELECT IDPRODUIT FROM PRODUIT_ALIMENTAIRE")
    product_ids = [row[0] for row in cursor.fetchall()]

    cursor.execute("SELECT IDDISTRIBUTEUR FROM DISTRIBUTEUR")
    distributer_ids = [row[0] for row in cursor.fetchall()]

    month_proba = [1, 0.9, 1, 1, 1, 1.3, 1.4, 1.3, 1, 1, 1, 1.8]
    month_proba /= np.sum(month_proba)

    distributer_proba = [1.2, 0.8, 1.5, 1.1, 0.7, 0.9, 1.5, 0.8, 0.9, 1.3]
    distributer_proba /= np.sum(distributer_proba)

    nb = 0
    for product_id in product_ids:
        total_quantity = 0  # Initialise la quantite totale vendue pour ce produit
        avg_sales = 100
        ecart_type = 10

        num_sales = int(np.clip(np.random.normal(loc=avg_sales, scale=ecart_type), avg_sales - 2*ecart_type, avg_sales + 2*ecart_type))
        for _ in range(num_sales):
            distributer_id = np.random.choice(distributer_ids, p=distributer_proba).item()
            sale_date = GenerateDate(month_proba)
            # Calcul de la quantite maximale pouvant etre vendue pour cette vente
            max_quantity = min(100 - total_quantity, int(np.clip(np.random.normal(loc=3, scale=1.0), 1, 5)))
            quantity = max_quantity if max_quantity >= 0 else 0  # Assure que la quantite est positive
            if quantity > 0:  # Verifie si la quantite est non nulle avant d'inserer la vente
                cursor.execute("INSERT INTO VENTE (IDPRODUIT, IDDISTRIBUTEUR, DATEVENTE, QUANTITE) VALUES (%s, %s, %s, %s)", (int(product_id), int(distributer_id), sale_date,
                total_quantity += quantity  # Met a jour la quantite totale vendue pour ce produit
                nb += 1
```

```python
def GenerateDate(month_probs):
    month = np.random.choice(np.arange(1, 13), p=month_probs)
    year = datetime.now().year - np.random.randint(0, 3)
    if month == 2:
        day = np.random.randint(1, 28)
    else:
        day = np.random.randint(1, 30)
    return datetime(year, month, day)

def connect_to_database():
    try:
        connection = psycopg2.connect(
            dbname="BDOFF",
            user="postgres",
            password="postgres",
            host="localhost"
        )
        cursor = connection.cursor()
        return connection, cursor
    except psycopg2.Error as e:
        print("Erreur lors de la connexion à la base de données PostgreSQL:", e)
        return None, None

def close_connection(connection, cursor):
    if cursor:
        cursor.close()
    if connection:
        connection.commit()
        connection.close()

if __name__ == "__main__":
    connection, cursor = connect_to_database()
    if connection and cursor:
        InsertSales(cursor)
        close_connection(connection, cursor)

```

I.II) Base de données en étoile :

Après la finalisation de notre base de données OLTP, nous avons entrepris la création d'une base de données suivant le modèle en étoile, reprenant ainsi les données de notre base OLTP :



Dans la construction du modèle en étoile, deux types de tables sont utilisés : les "Dimensions" et les "Faits". Les Dimensions représentent différents angles à partir desquels les données sont observées, tels que les clients, les produits ou les lieux. Les Faits correspondent aux informations que nous souhaitons comprendre à partir de ces données, comme le nombre de produits vendus. En utilisant ces deux types de tables, les analyses deviennent plus claires et plus faciles à interpréter, car elles fournissent le contexte (les Dimensions) nécessaire pour interpréter les chiffres (les Faits).

Justification des choix de tables de Fait et de Dimensions :

- La table **DIM_DISTRIBUTEUR** : Elle enregistre des informations sur les distributeurs, permettant de comprendre l'origine des ventes.
- La table **DIM_GEOGRAPHIQUE** : Elle fournit des détails sur les lieux géographiques des produits, offrant ainsi une vision des ventes par pays.
- La table **DIM_PRODUIT** : Elle décrit les produits vendus, facilitant ainsi l'analyse des ventes par catégories de produits, par exemple.
- La table **DIM_INGREDIENT** : Elle enregistre les ingrédients présents dans les produits, tandis que la table FAIT_COMPOSITION permet de lier les produits aux ingrédients.
- La table **DIM_ADDITIF** : Similaire à DIM_INGREDIENT, elle enregistre les additifs.
- La table **DIM_TEMPS** : Elle fournit un contexte temporel aux ventes, permettant de suivre l'évolution des ventes au fil du temps.
- La table **FAIT_EST_COMPOSE** : Elle enregistre des détails en plus de produits, fournissant ainsi une vue complète des produits sous forme de table en plus de la table DIM_PRODUIT.
- La table **FAIT_VENTE** : Elle enregistre tous les détails des ventes.

Pour les insertions dans le modèle en étoile, la procédure suivante a été suivie :

1) La base de données en étoile a été insérée dans la base de données OLTP.

2) Ensuite, à l'aide de procédures SQL, les données de la base OLTP ont été insérées dans les tables du modèle en étoile. Exemple pour la table distributeur :

```sql
--Procédure Table Dim_Distributeur

CREATE OR REPLACE PROCEDURE insert_into_dim_distributeur()
AS $$
DECLARE
  rec record;
BEGIN
  FOR rec IN (SELECT iddistributeur, libelledistributeur FROM distributeur)
  LOOP
    INSERT INTO dim_distributeur (iddimdistributeur, libelledistributeur) VALUES (rec.iddistributeur, rec.libelledistributeur);
  END LOOP;
END;
$$ LANGUAGE plpgsql;

CALL insert_into_dim_distributeur();
```

3) Par la suite, le modèle OLTP a été entièrement supprimé, ne laissant dans la base de données que le modèle en étoile avec toutes les données insérées.

4) Suite à cela, une nouvelle base de données OLTP a été créée, les données y ont été réinsérées, puis les deux bases de données séparées ont été hébergées avec Azure à l'adresse suivante :

Adresse : mrab-pgsql.postgres.database.azure.com
Mot de passe : Mi1lh2an%

I.III) Que choisir ?

La requête SQL suivante a été exécutée sur chaque base de données dans PgAdmin afin de déterminer laquelle est la moins volumineuse. Les résultats indiquent que la base de données OLTP contient 257 MB de stockage, tandis que la base de données en étoile contient 190 MB de stockage.

	taille text		taille text
1	190 MB	1	257 MB

Requête :

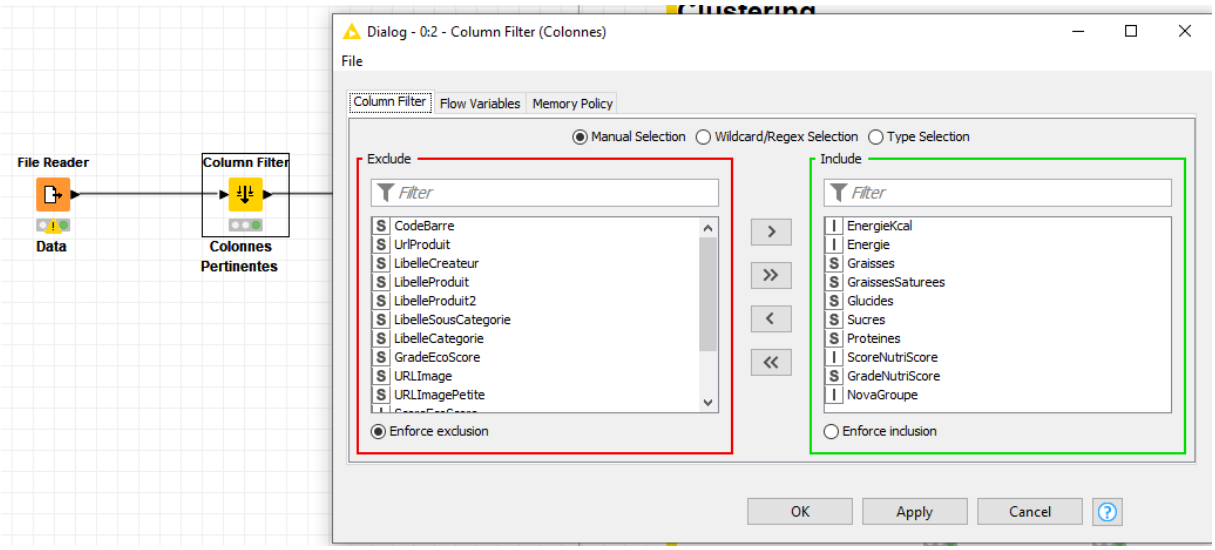
```sql
SELECT pg_size_pretty(pg_database_size('nom_de_votre_base_de_données')) AS taille;
```

II) Visualisation des données :

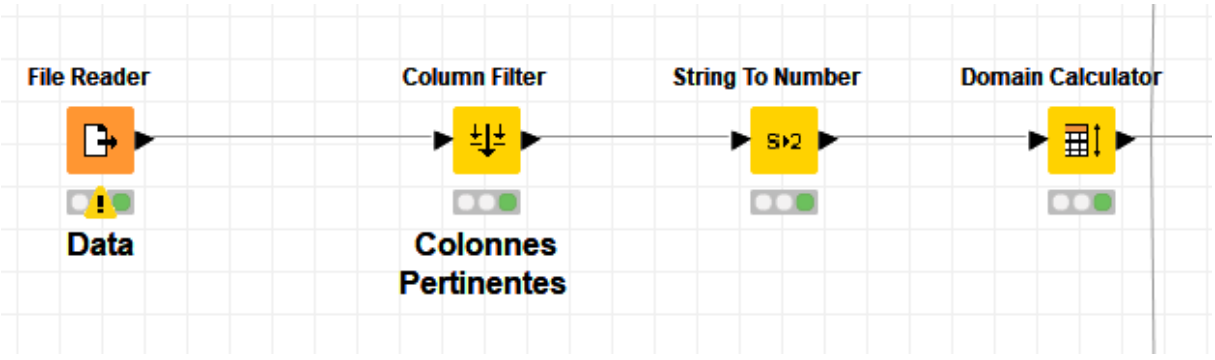
II.I) KNIME

L'objectif est de créer un nouveau Nutri-Score en utilisant les données du fichier CSV traitées avec le logiciel Knime. Ce logiciel permet de réaliser une série de traitements et d'analyses de données.

L'analyse des données commence par l'utilisation du fichier CSV "nettoyé" à l'aide du File Reader. Ensuite, seules les colonnes pertinentes pour le calcul d'un Nutri-score sont conservées en utilisant un Colonne Filter :

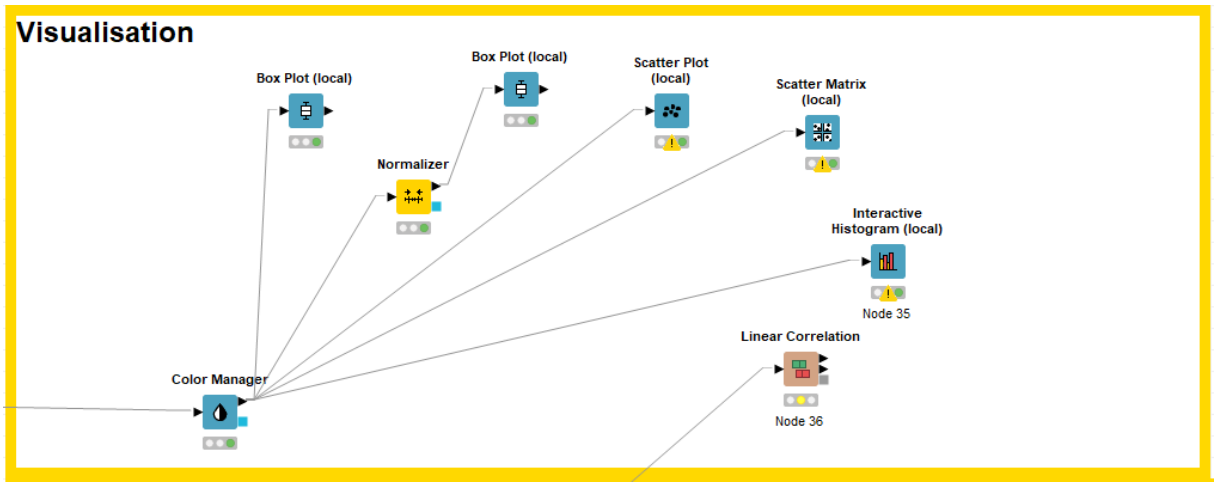


Ensuite, les valeurs nutritionnelles sont ajustées au bon format à l'aide du String To Number. Le Domaine Calculator permet ensuite de définir la table qui sera soumise aux traitements ultérieurs :

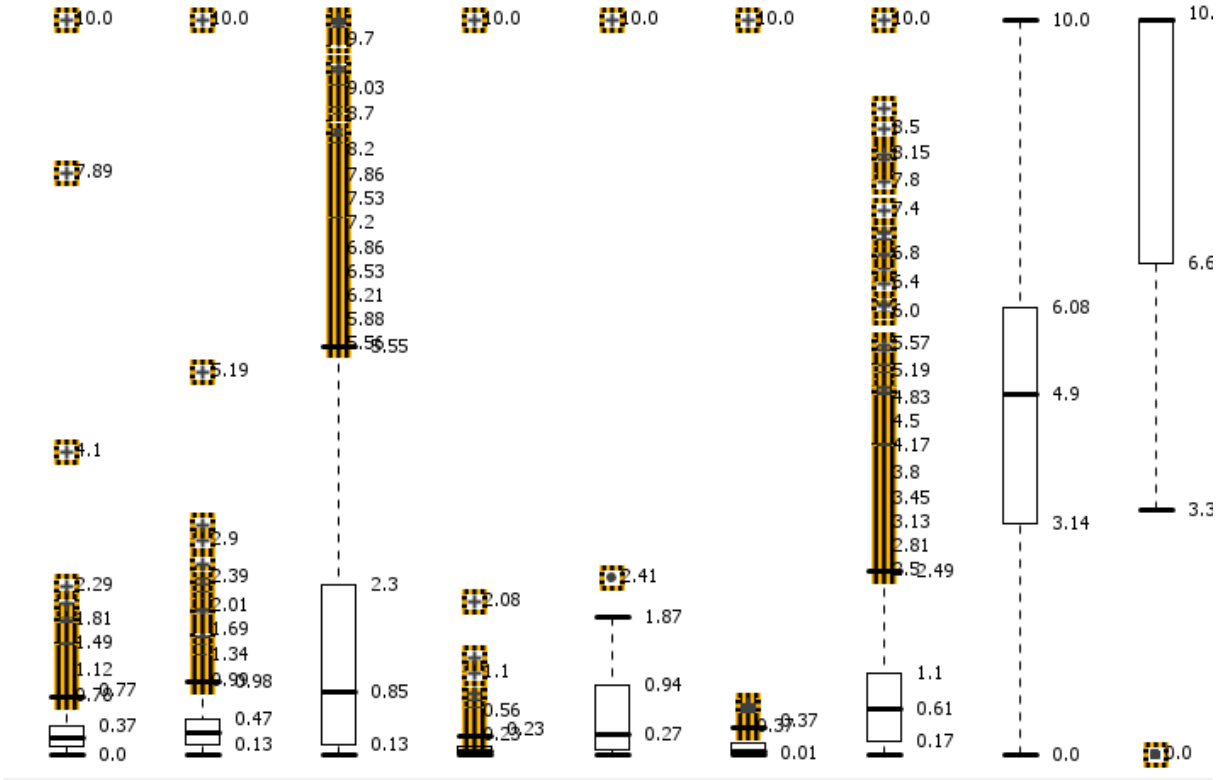


Visualisation :

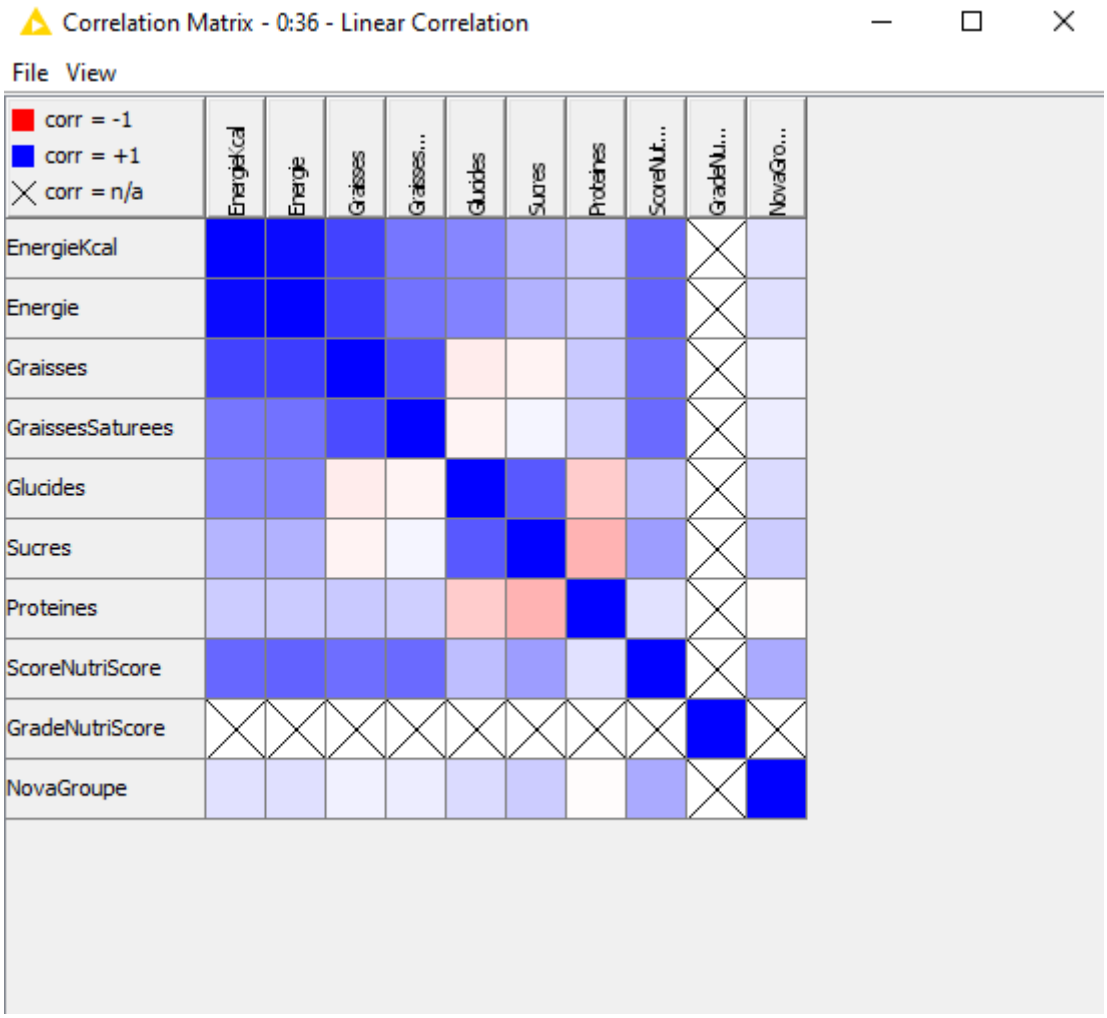
Ensuite, pour visualiser les données, elles ont été affichées de différentes manières : Box Plot, le Scatter Plot, le Scatter Matrix, Histogramme et la Linear Correlation, avec l'application préalable de couleurs à chaque colonne à l'aide du Color Manager.



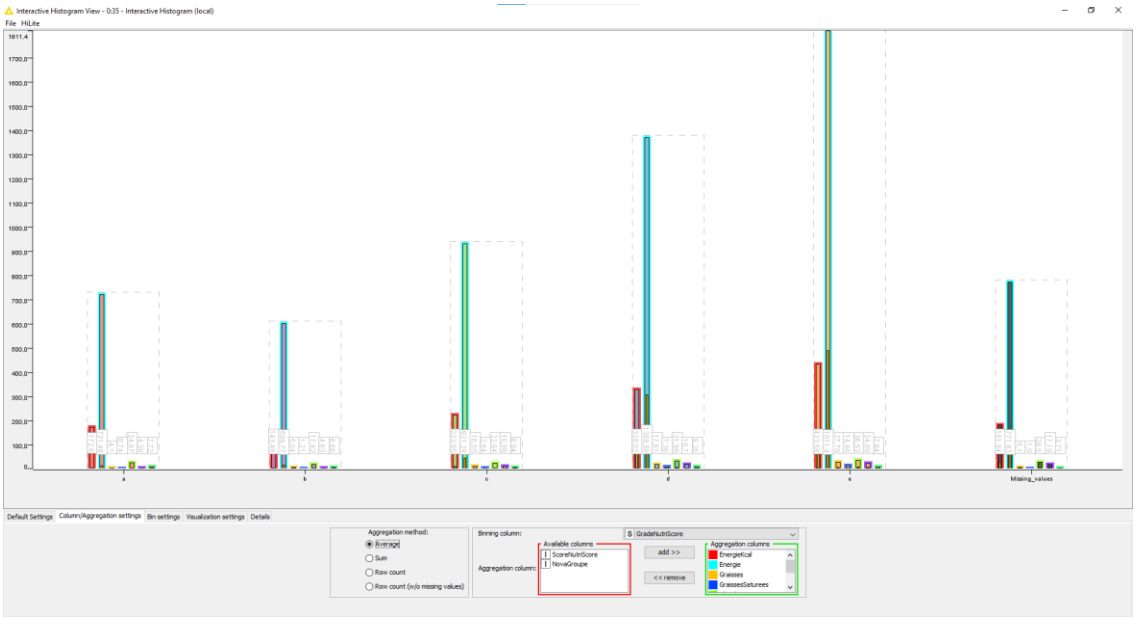
Le Box Plot a permis d'identifier les valeurs aberrantes ainsi que les données les plus complètes, qui ont été mises en évidence à l'aide d'un Hilite (en jaune ci-dessous) :



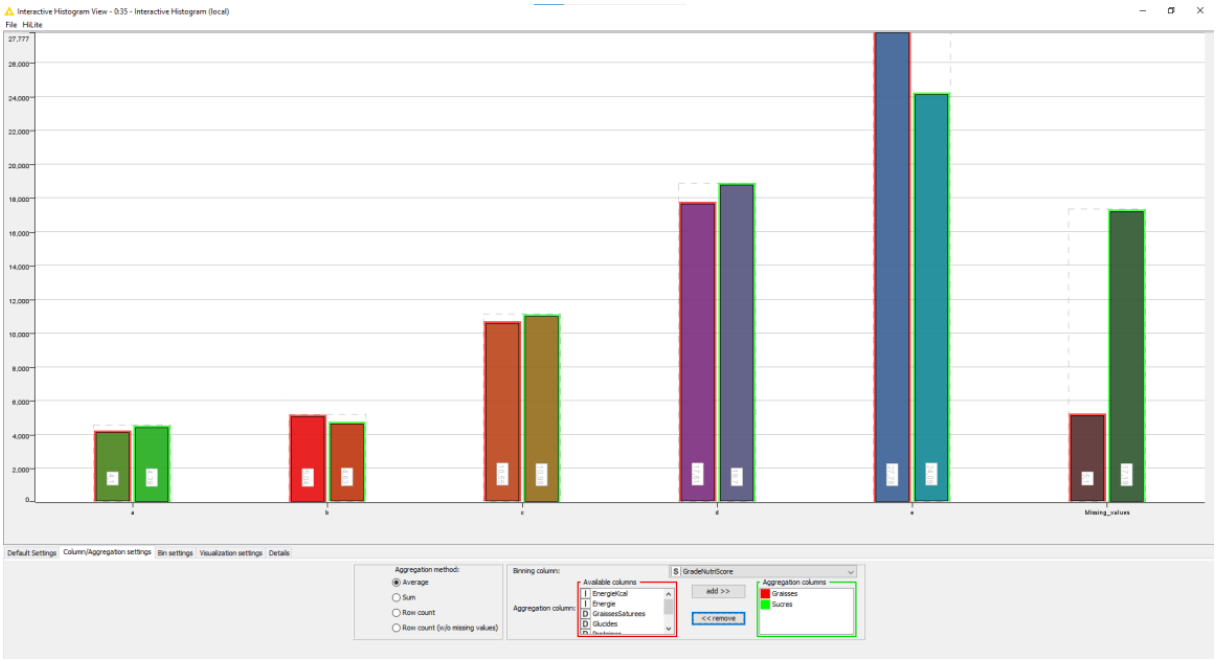
Le Lineair Corelation nous a permis d'identifier les colonnes qui présentaient un lien entre elles :



Enfin, l'histogramme nous a permis de visualiser les valeurs nutritionnelles selon leur Nutri-score :

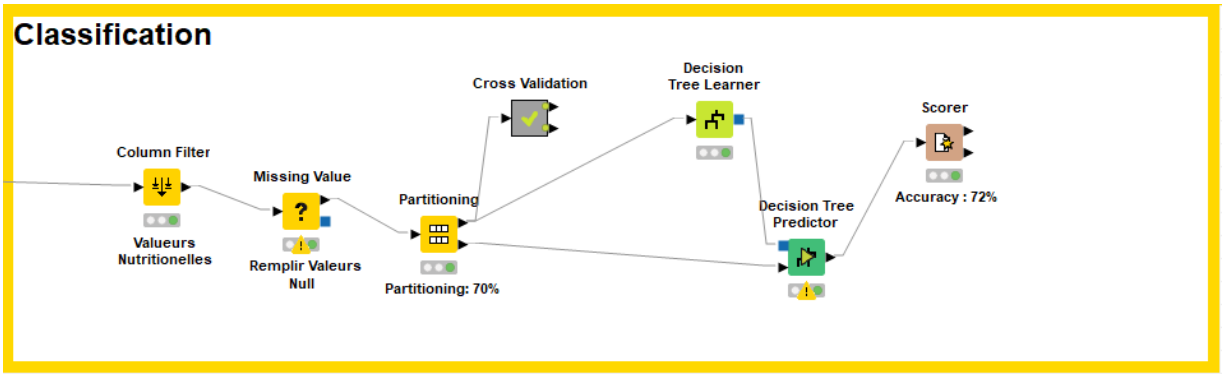


On remarque que les valeurs de graisse et de sucre augmentent proportionnellement au Nutri-score :

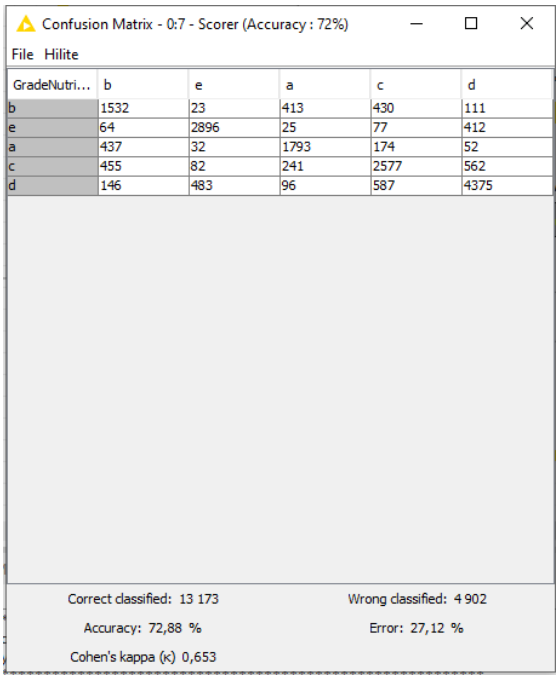


Classification :

La qualité des données a été évaluée en envisageant une éventuelle prédiction du Nutri-score. Le score du Nutri-score a été retiré de la table pour éviter tout biais dans les prédictions. Ensuite, les champs vides ont été remplis par des valeurs (0 ou "inconnue") à l'aide du traitement des valeurs manquantes. Un Partitioning et un Decision Tree Learner ont été utilisés pour l'apprentissage et la prédiction. Enfin, la précision de cette prédiction a été calculée à l'aide d'un score pour évaluer la qualité des données et des prédictions, avec un taux d'accuracy de 72 % :

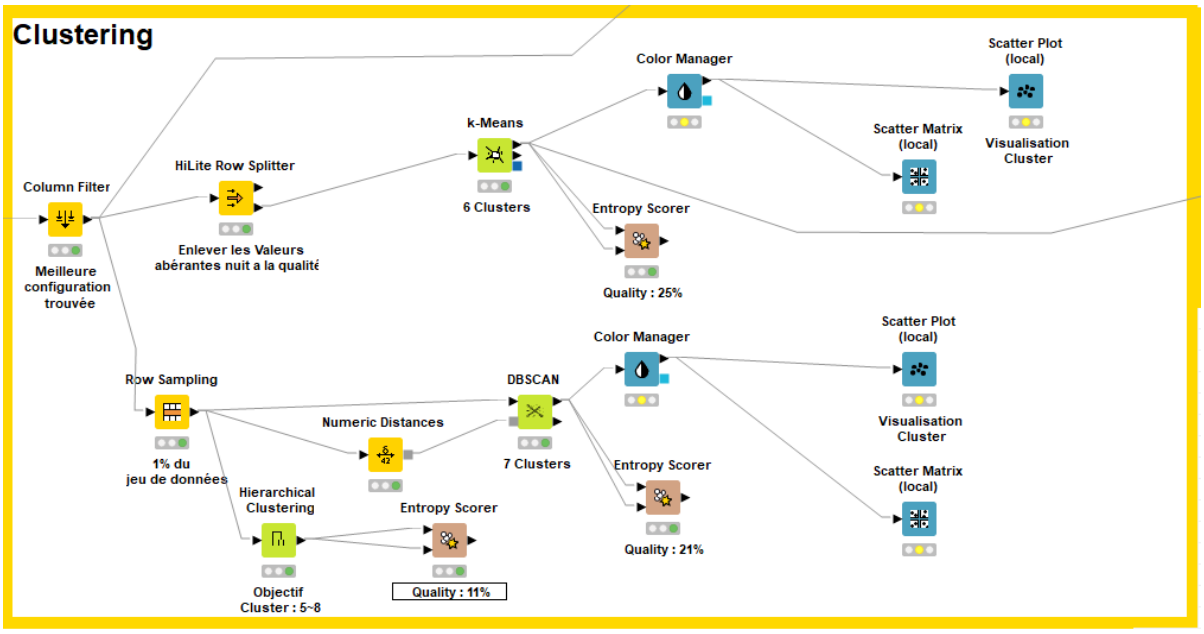


BACHAR Bilal | DEDDOUCHE Yanisse | MIHOUBI Marouane | RABEHI Milhane

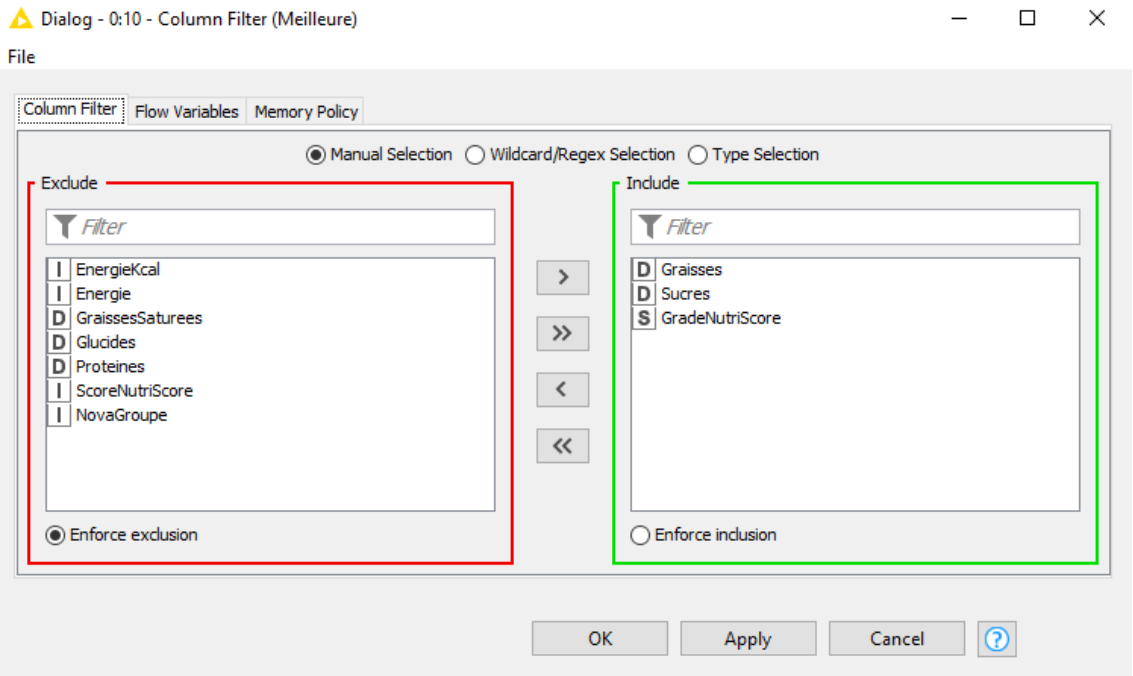


Clustering :

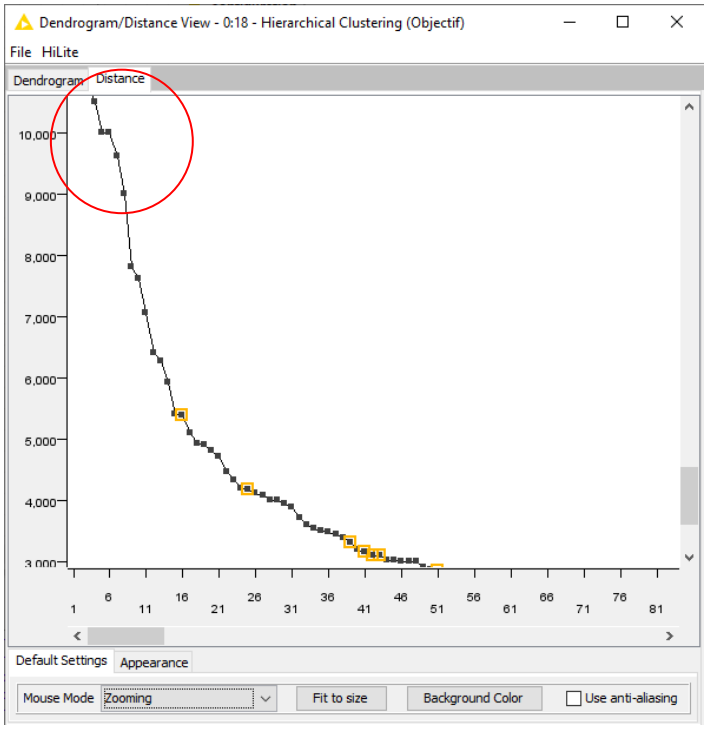
Pour créer le nouveau Nutri-score, des outils de clustering (K-Means, DBScan et Hierarchical) ont été utilisés pour déterminer la méthode de clustering la plus adaptée à nos données, ainsi que les colonnes les plus utiles pour calculer un Nutri-score :



Après avoir réalisé une batterie de tests et visualisé nos données, la meilleure combinaison de colonnes trouvée s'avère être Graisse et Sucre. Les images et résultats suivants seront donc basés sur cette configuration :



Hierarchical : Le processus de clustering a débuté par le Hierarchical Clustering afin de déterminer le nombre de clusters le plus approprié pour nos données. Cependant, cette méthode ne peut être utilisée efficacement qu'avec un faible nombre de données. Par conséquent, nous avons réduit les données en ne sélectionnant que 1 % de l'ensemble des données via un Row Sampling. Comme il est observé, le nombre optimal de clusters se situe entre 5 et 8.



Voici les résultats de la qualité de ce clustering fournis par un Entropy Scorer :

Clustering statistics

Data Statistics

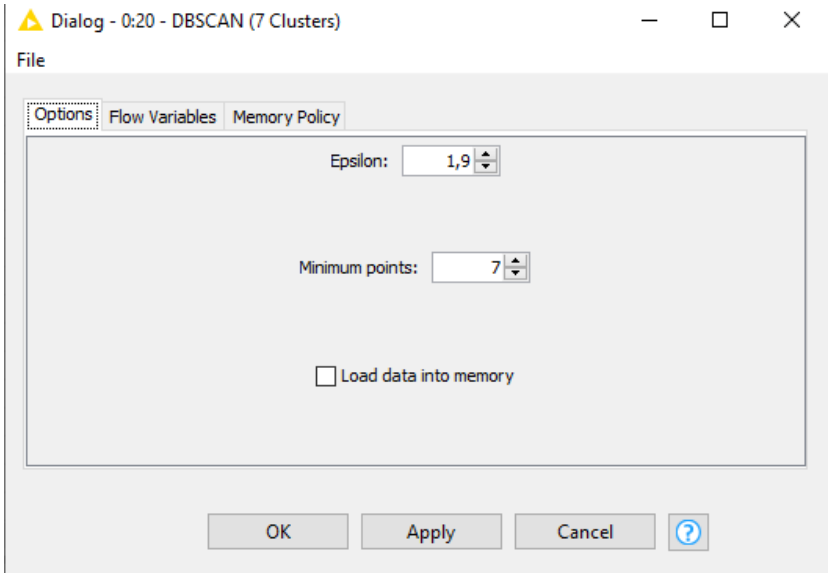
Statistics	Value
Number of clusters found:	8
Number of objects in clusters:	624
Number of reference clusters:	6
Total number of patterns:	624

Data Statistics

Score	Value
Entropy:	2,293
Quality:	0,1129

Row ID	Size	Entropy	Normali...	Quality
cluster_0	1	0	0	?
cluster_1	1	0	0	?
cluster_2	1	0	0	?
cluster_3	1	0	0	?
cluster_4	2	0	0	?
cluster_7	3	0	0	?
cluster_5	13	1.669	0.646	?
cluster_6	602	2.341	0.906	?
Overall	624	2.293	0.887	0.113

DBScan : Avant d'effectuer le clustering par DBScan, la taille du jeu de données a été réduite via un Row Sampling, comme pour le Hierarchical Clustering précédent. Un epsilon de 1,9 et un nombre maximum de points de 7 ont été choisis comme paramètres du DBScan afin d'obtenir un nombre de clusters correspondant à la fourchette indiquée par le Hierarchical Clustering.



Voici les résultats de la qualité de ce clustering fournis par un Entropy Scorer :

Clustering statistics

Data Statistics

Statistics	Value
Number of clusters found:	7
Number of objects in clusters:	624
Number of reference clusters:	6
Total number of patterns:	624

Data Statistics

Score	Value
Entropy:	2,0188
Quality:	0,219

Row ID	I Size	D Entropy	D Normali...	D Quality
Cluster_3	9	0.503	0.195	?
Cluster_1	7	0.592	0.229	?
Cluster_2	11	0.684	0.265	?
Cluster_4	14	1.296	0.501	?
Cluster_0	6	1.459	0.564	?
Noise	255	1.818	0.703	?
Cluster_5	322	2.338	0.905	?
Overall	624	2.019	0.781	0.219

K-Means : Le processus de clustering K-Means a finalement été utilisé, avec 6 clusters en paramètre de sortie. Voici les résultats :

Clustering statistics

Data Statistics

Statistics	Value
Number of clusters found:	6
Number of objects in clusters:	62413
Number of reference clusters:	6
Total number of patterns:	62413

Data Statistics

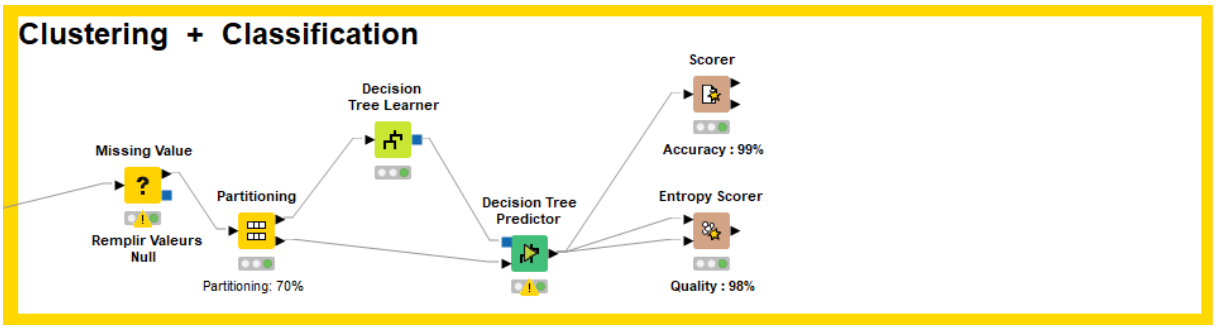
Score	Value
Entropy:	1,9221
Quality:	0,2564

Row ID	Size	Entropy	Normali...	Quality
cluster_4	10408	1.334	0.516	?
cluster_5	8092	1.531	0.592	?
cluster_3	5197	1.539	0.595	?
cluster_2	2255	1.908	0.738	?
cluster_0	11483	2.072	0.802	?
cluster_1	24978	2.306	0.892	?
Overall	62413	1.922	0.744	0.256

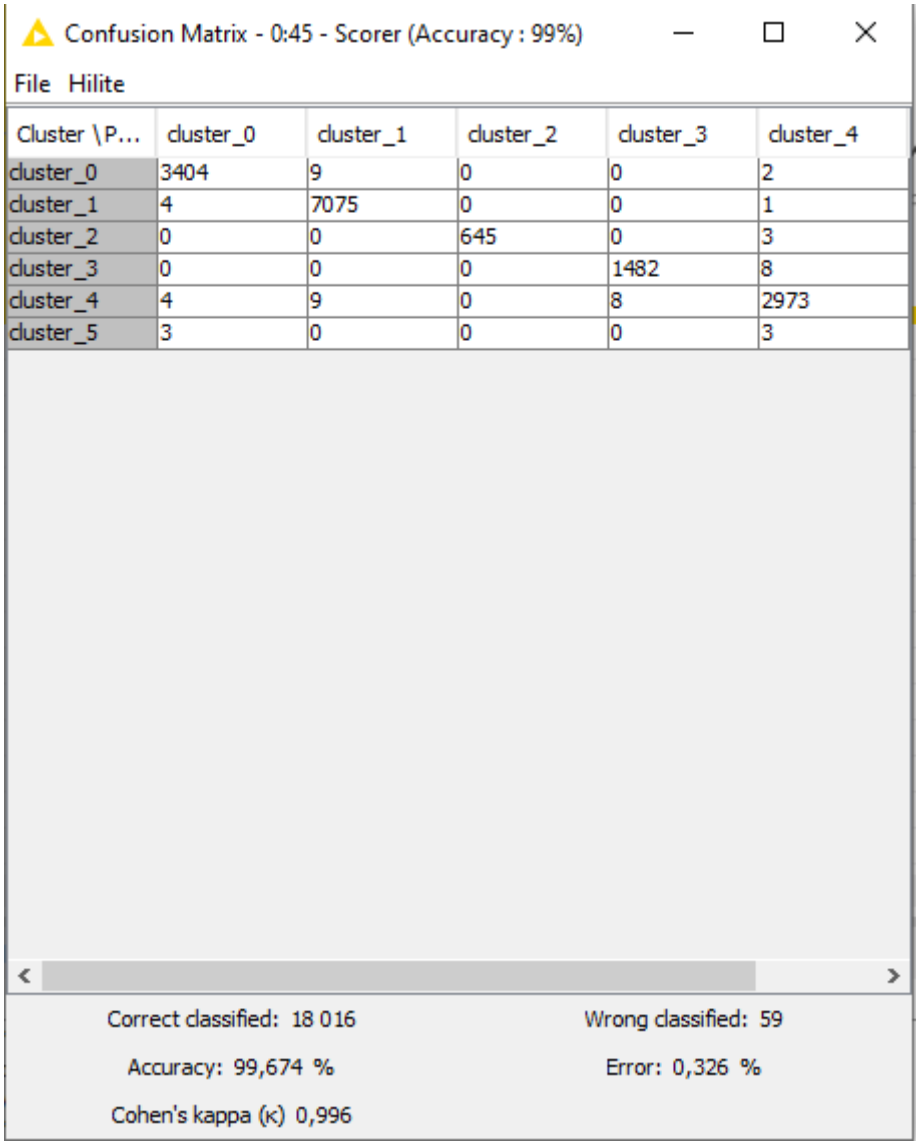
Pour conclure sur les clustering, il a été établi que le K-means était le plus adapté pour nos données. En effet, la meilleure qualité et entropy ont été obtenues avec celui-ci. Cela semble logique car les points de données étant très proches les uns des autres, DBScan devient moins optimal.

Clustering + Classification :

Maintenant que le meilleur clustering (K-means) a été identifié, les clusters fournis par celui-ci deviennent notre nouveau Nutri-Score. Nous avons testé l'accuracy de notre nouveau Nutri-Score en sachant d'avance qu'il sera bon, car nous l'avons fabriqué à nouveau grâce au partitioning et au Decision Tree Learner :



Voici les résultats qui sont pratiquement parfaits, car ils proviennent de nous :



Cluster \P...	cluster_0	cluster_1	cluster_2	cluster_3	cluster_4
cluster_0	3404	9	0	0	2
cluster_1	4	7075	0	0	1
cluster_2	0	0	645	0	3
cluster_3	0	0	0	1482	8
cluster_4	4	9	0	8	2973
cluster_5	3	0	0	0	3

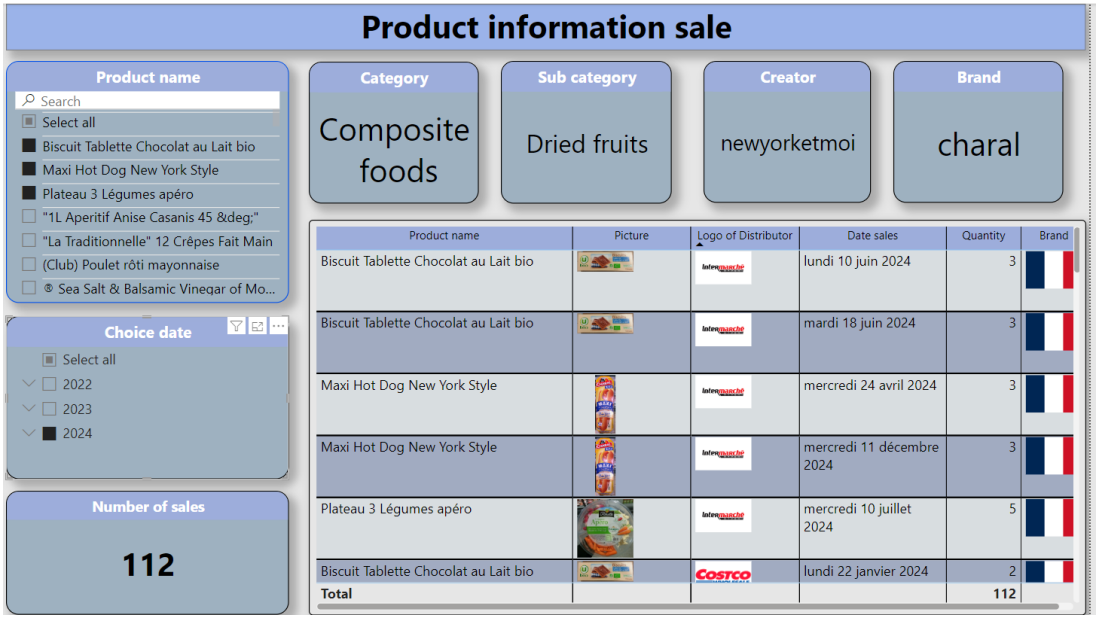
II.II) Power BI

Analyse des données de la base de données en étoile vue précédemment :

D'abord la page Recherche produit :

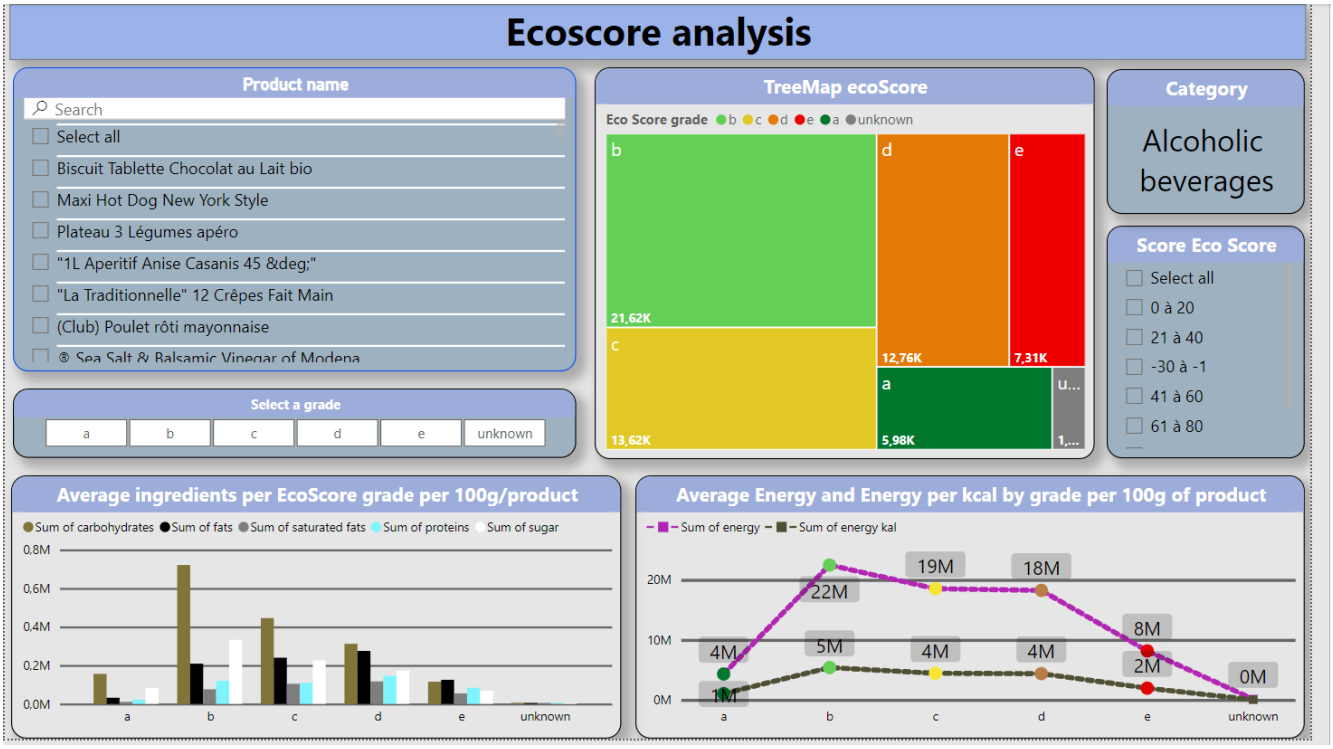
Il est possible de rechercher et/ou sélectionner le produit désiré, ainsi que la marque, la catégorie et la sous-catégorie pertinente pour une recherche de produit. Dans la table de données du produit, des informations telles que le nom du produit, l'image, le pays d'origine, la légende du groupe NOVA, l'image de l'éco-score et du Nutri-Score correspondant, ainsi que l'URL de l'image sont disponibles.

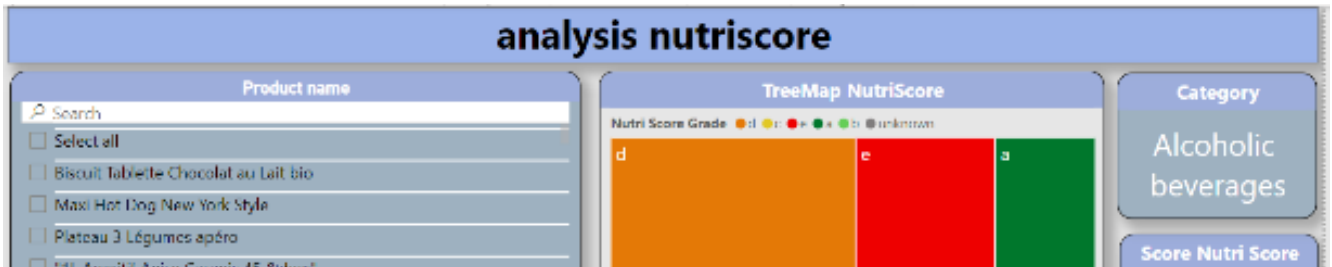
Ensuite la page des informations sur les ventes des produits :



Dans cette interface, il est possible de rechercher ou sélectionner le nom du produit ainsi que la date de vente. Une fois sélectionné, on peut observer le nombre de ventes pour la date choisie, ainsi que la catégorie, la sous-catégorie, le créateur et la marque. La table contient des informations telles que le nom du produit, l'image, les dates précises de ventes, le distributeur, la quantité et le pays d'importation.

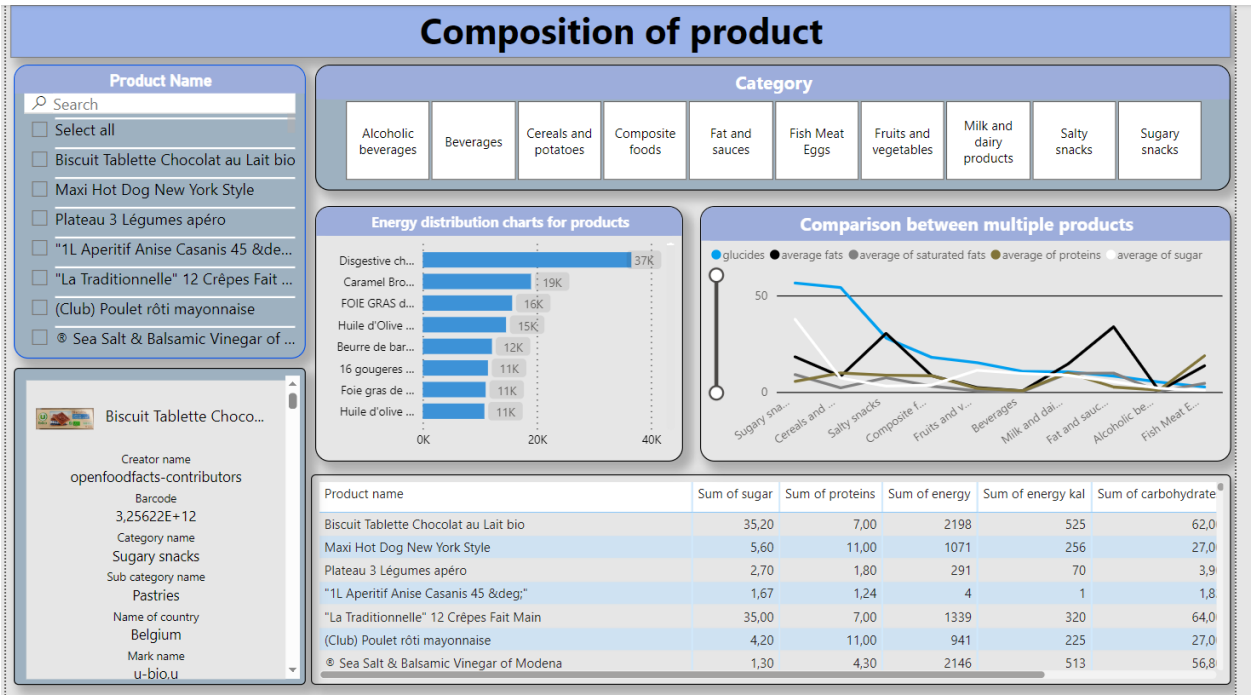
De plus, les deux pages fournissent des analyses détaillées concernant l'éco-score et le nutri-score.





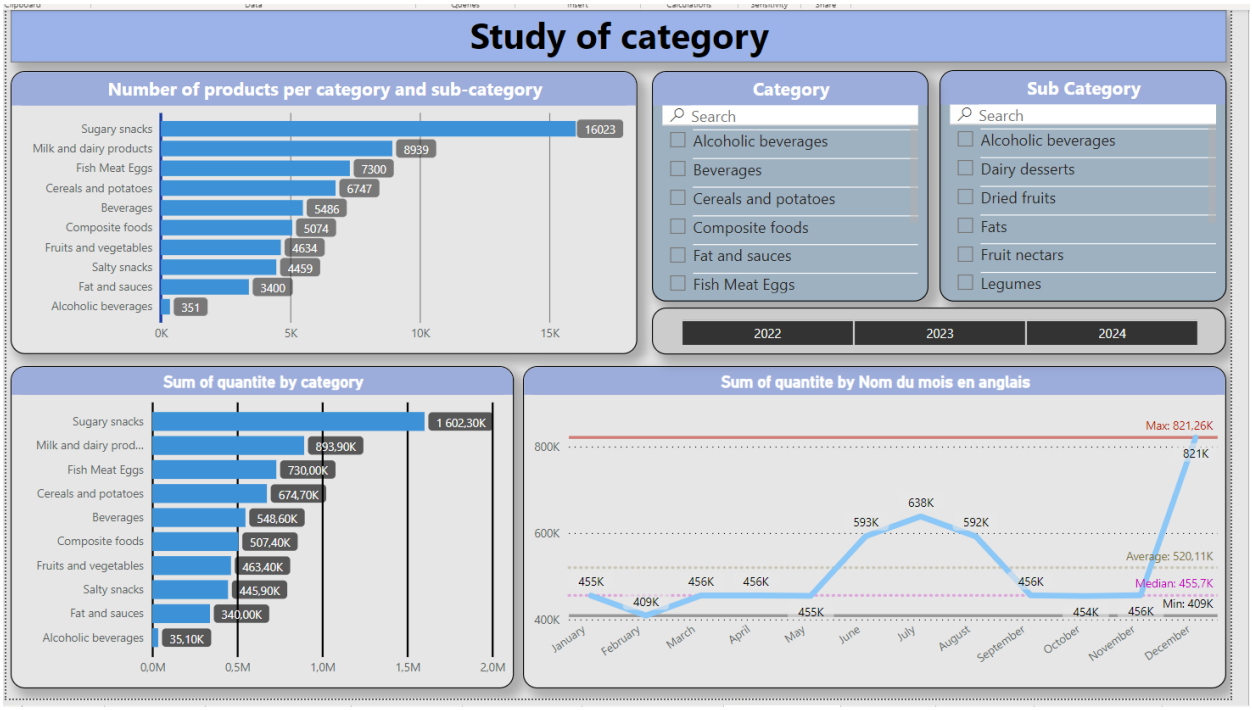
Dans notre interface, un treemap représente la répartition des différents grades par produit. Vous pouvez sélectionner le nom du produit et le score Nutri-Score, c'est-à-dire le grade du produit. Ensuite, vous visualisez la catégorie et, sur les deux graphiques, vous avez la possibilité de voir la répartition en grammes pour 100 grammes de produit pour le sucre, les protéines, les graisses, les graisses saturées et les glucides en fonction du grade et des éléments présélectionnés. Sur le deuxième graphique, vous pouvez observer la somme des répartitions de l'énergie et de l'énergie en kilocalories.

Voici la page de composition de chaque produit :



Dans cette section, vous avez la possibilité de sélectionner le nom du produit, ce qui déclenche une visualisation sous forme de cartes des données du produit, y compris le choix de la catégorie. Vous pouvez également observer la distribution d'énergie en moyenne par produit, représentée graphiquement sous forme de ligne par rapport aux moyennes des composants. Enfin, une table détaille précisément les compositions de chaque donnée pour une analyse approfondie.

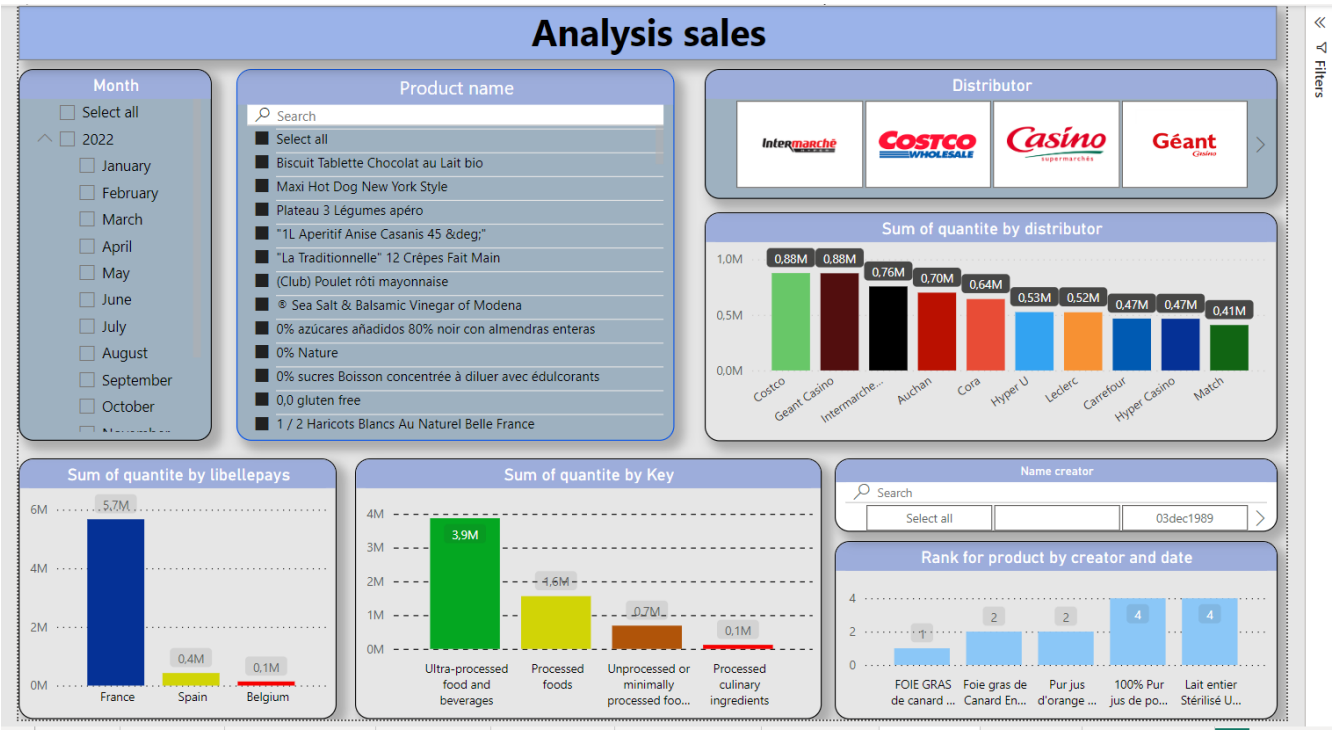
Voici la page de l'étude sur les catégories :



Dans cette section, vous avez la possibilité de choisir la catégorie désirée ainsi que la sous-catégorie et l'année correspondante. Un premier graphique affiche la répartition des produits par catégorie et par sous-catégorie, ainsi que la somme des quantités vendues par catégorie. De

plus, un graphique montre la somme des quantités vendues par mois pour chaque année sélectionnée.

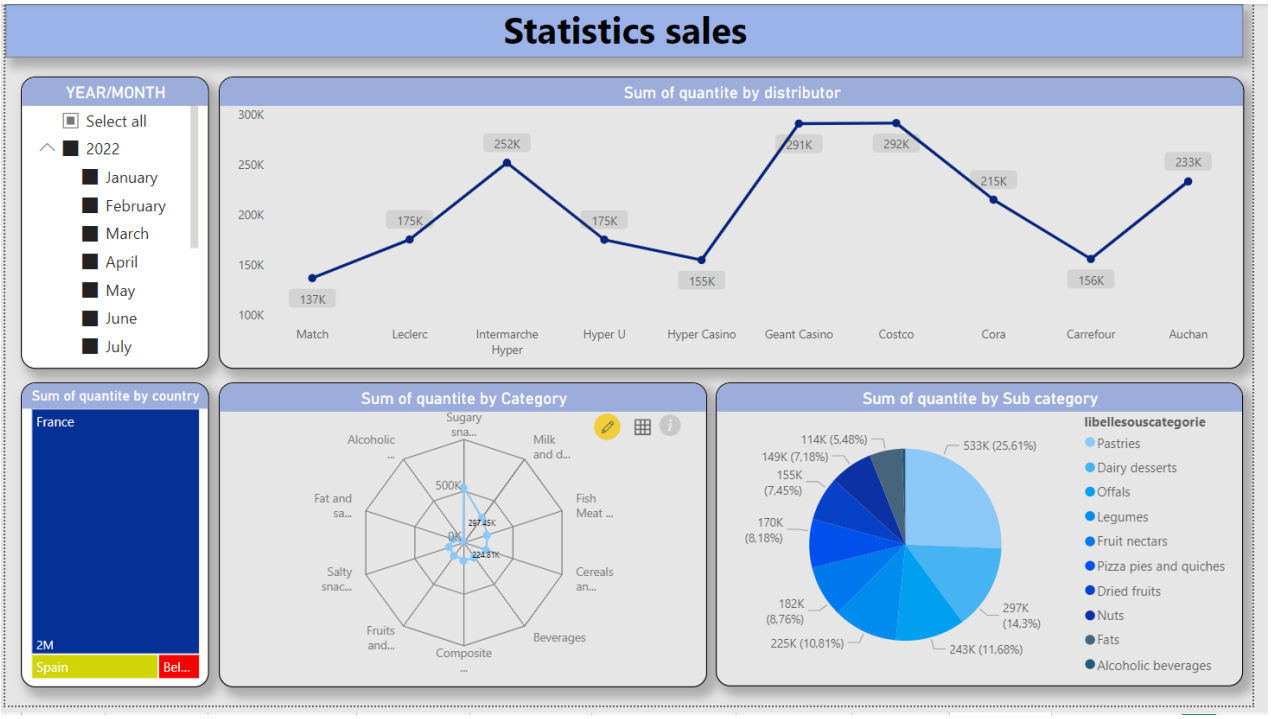
Voici l'analyse des ventes :



Dans cette interface, vous pouvez choisir le mois et l'année de la vente, ainsi que le nom du produit et le distributeur souhaité. Cela vous permet de visualiser la somme des quantités vendues par distributeur, la somme des quantités par pays et par groupe NOVA. De plus, vous

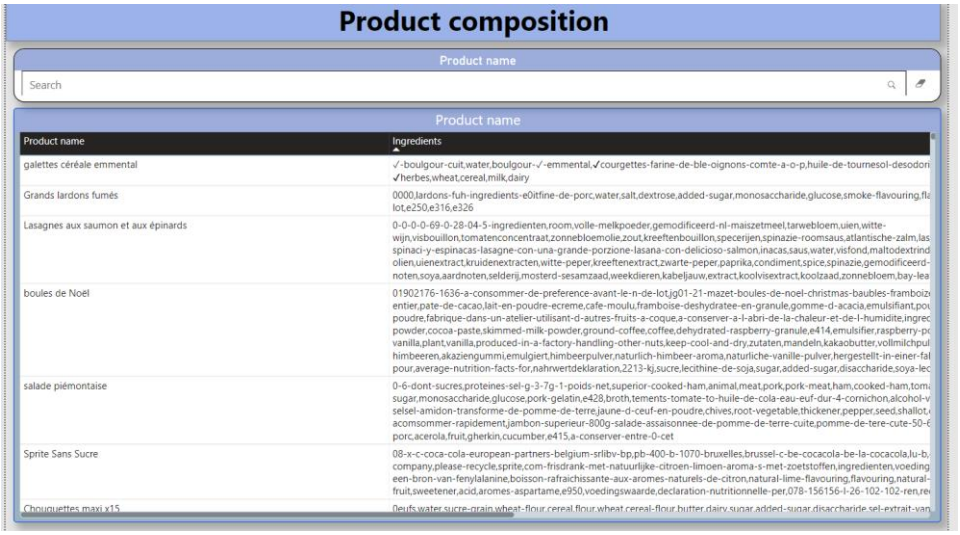
avez la possibilité de sélectionner la marque pour classer les produits en fonction des quantités vendues.

Voici la statistique sur les ventes :



Dans cette section, vous avez la possibilité de sélectionner l'année et le mois souhaités. Ensuite, vous pouvez explorer un treemap représentant les pays d'importation, la somme des quantités par distributeur, un diagramme en radar illustrant les quantités par catégorie, ainsi qu'un diagramme circulaire montrant les sommes des quantités par sous-catégorie.

Enfin voici une page sur la recherche de composition par produit :



En recherchant un produit on retrouve sa composition au niveau des ingrédients et des additives.

Adresse de diffusion du rapport Power BI :
https://app.powerbi.com/view?r=eyJrIjoiMTBjYjFiZjctZDRhOC00NjJhLTg2NjYtMmJhZWU1ZGYwMjQwIiwidCI6ImUyMWU5NzgzLWQwYTAtNDhmOC04NTBlLTBiMDgxYjQ2ZDc4OCIsImMiOjh9