

“Concevez une application pour Santé Publique France”

Projet n°3

Léa ZADIKIAN
Parcours Data Scientist
20 Novembre 2022

Concevez une application pour Santé Publique France

- I. Rappel de l'appel à projet et présentation de l'idée d'application
- II. Opérations de nettoyages effectuées sur le jeu de données
- III. Analyse exploratoire des données



I. Rappel de l'appel à projet et idée d'application



Un projet d'application en lien avec la nutrition


- L'agence **Santé publique France** a lancé un appel à projet pour trouver des idées innovantes d'applications en lien avec l'alimentation.



- Informations de la base de données de produits alimentaires du site **Open Food Facts** :



- Ingrédients et additifs éventuels
- Valeurs nutritionnelles et Nutri-Score
- Le packaging, le pays de vente...



Muesli Superfruits - Bjorg - 375 g

Barcode: 3229620782560 (EAN) / EAN-13

Common name: Mélange biologique de céréales et de fruits


Quantity: 375 g

Packaging: 1x Sachet plastique à jeter

Brand: Bjorg

Categories: Plant-based foods and beverages, Plant-based foods, Breakfasts, Cereals and pastas, Cereals and their products, Breakfast cereals, Cereals with fruits, Muesli, Muesli with fruits

Labels, certifications, awards: Organic, Certified by Ecocert, EU Organic, Non-EU Agriculture, Source of fibre, DE-ÖKO-001, EU Agriculture, EUMon-EU Agriculture, Green Dot, High fibre, Made in Germany, No added sugar, Nutriscore, Nutriscore Grade A, AB Agriculture Biologique, 100% European Certified #



Origin of ingredients: Unspecified

Manufacturing or processing places: Allemagne

EMV code: A1 05/05

Link to the product page on the official site of the producer: <https://www.bjorg.fr/products/muesli-sfp...>

Stores: Lidl.fr, Auchan, Mappesa U, carrefour.fr

Countries where sold: Belgium, France, Netherlands, Mexico, Morocco, Norway, Spain, Switzerland

Idée d'application

- Identifier le produit à partir de son code-barres et obtenir :
 - Ses informations nutritionnelles (quantité sel, sucre, protéines, etc...) pour 100g de produit
 - Son Nutri-Score : qualité nutritionnelle notée de A à E
- Si le Nutri-Score du produit est "médiocre" ou "mauvais" (C,D ou E), **proposer une alternative de produit**, c'est-à-dire un produit de la même catégorie mais ayant un meilleur Nutri-Score (A ou B).



Tableau nutritionnel

Tableau nutritionnel	Tel que vendu pour 100 g / 100 ml
Énergie	1 711 kJ (401 kcal)
Matières grasses	9,1 g
Acides gras saturés	2,1 g
Glucides	67 g
Sucres	23 g
Fibres alimentaires	7,6 g
Protéines	8,8 g
Sel	0,04 g
Fruits, légumes, noix et huiles de colza, noix et olive (estimation par analyse de la liste des ingrédients)	4,9 %

Environnement technique

- Notebook Jupyter 6.4.8
- Python 3.9.12
- Librairies utilisées :
 - pandas
 - numpy
 - missingno
 - matplotlib et seaborn
 - sklearn

Jeu de données Open Food Facts

Open Food Facts



- **320.772 lignes** ⇒ produits alimentaires
- **162 colonnes** ⇒ informations sur les produits, séparées en 4 sections :
 1. Informations générales : code-barres, nom, date de modification...
 2. Tags : catégorie du produit, origine, pays de vente, etc...
 3. Ingrédients et additifs éventuels
 4. Informations nutritionnelles : quantité de nutriments pour 100g de produit, Nutri-score...
- Jeu de données avec de nombreuses variables peu renseignées : **+ de 75% de valeurs manquantes**

II. Opérations de nettoyage

II. Opérations de nettoyage

1. Opérations de filtrage

- Suppression des produits sans code-barres et recherche de doublons
- Produits vendus en France
- Filtrage des colonnes

2. Gestion des valeurs aberrantes

3. Gestion des valeurs manquantes

Opérations de filtrage : Code-barres

- Code-barres nécessaire pour identifier le produit \Rightarrow suppression des produits de 23 produits sans code-barre.

```
Entrée [110]: # Nous supprimons Les lignes de produits sans code-barres
avec_code_barre_foodData=foodData.dropna(axis=0,subset='code')
avec_code_barre_foodData
```

- Étude des doublons \Rightarrow Aucune ligne identique, aucun doublon au niveau du code-barres.

```
Entrée [14]: # Existe-t-il des lignes de produits exactement identiques ? --> Non
avec_code_barre_foodData.duplicated().sum()
```

Out[14]: 0

```
Entrée [126]: # Existe-t-il des produits ayant Le même code-barres ?
avec_code_barre_foodData.loc[((avec_code_barre_foodData['code'].duplicated(keep=False))),:]
```

Out[126]:

code	url	creator	created_t	created_datetime	last_modified_t	last_modified_datetime	product_name	generic_name
------	-----	---------	-----------	------------------	-----------------	------------------------	--------------	--------------

0 rows × 162 columns

Filtrage sur les produits vendus en France

- Recherche de la chaîne de caractère **"France"** dans les 3 variables :
 - *countries_fr*
 - *purchase_places*
 - *cities_tags*

```
Entrée [44]: # Filtrage sur Les produits vendus en France.
# Pour chaque produit nous allons donc chercher si on retrouve le mot "france" dans l'une des 3 colonnes :
# 'purchase_places' ou 'cities_tags' ou 'countries_fr'.

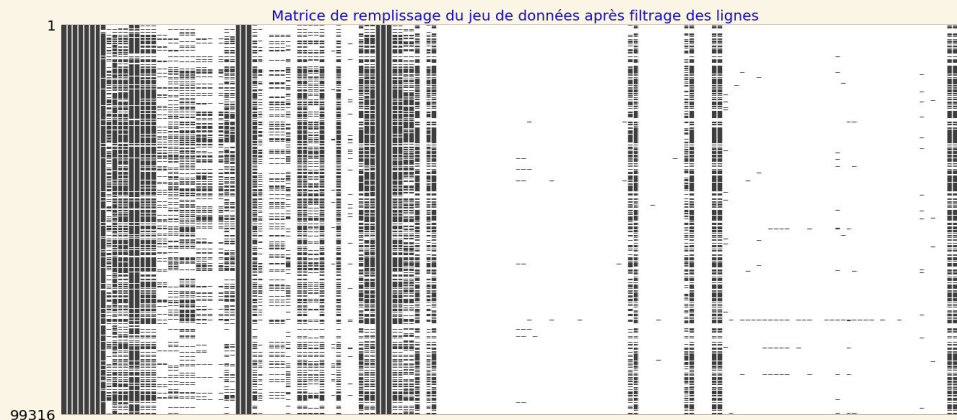
# Si l'un des de ces 3 champs comporte le mot "france" (avec ou sans majuscule),
# On affecte la valeur "France" dans la colonne 'countries_fr'
mask1=(data['purchase_places'].str.contains('France',case=False,na=False)
| data['cities_tags'].str.contains('France',case=False,na=False)
| data['countries_fr'].str.contains('France',case=False,na=False))

data.loc[mask1,'countries_fr']='France'
```

- Intégration des DOM-TOM avec la même méthode.
- Jeu de données après filtrage : 99.316 lignes.

```
Entrée [28]: # Filtrage des lignes de produits ayant "France" pour pays de vente
french_data=data.loc[data['countries_fr']=='France']
french_data
```

Filtrage des colonnes



1. Identification des colonnes ayant un taux de remplissage > à 50%

⇒ passage de 162 à 42 colonnes

2. Choix des colonnes pertinentes :

- Code-barres et nom du produit
- Catégorie
- Valeurs nutritionnelles pour 100g
- Nutri-score

```
#En utilisant un taux de remplissage de colonnes au moins égale à 50%
# Parmi ces colonnes, on choisit les indicateurs (colonnes) pertinents

indicateursRetenus=['code',
                    'product_name',

                    'pnns_groups_1',
                    'pnns_groups_2',

                    'energy_100g',
                    'fat_100g',
                    'saturated-fat_100g',
                    'sugars_100g',
                    'carbohydrates_100g',
                    'salt_100g',
                    'sodium_100g',

                    'fiber_100g',
                    'proteins_100g',

                    'nutrition-score-fr_100g',
                    'nutrition_grade_fr']

short_data=french_data.filter(items=indicateursRetenus)
```

Synthèse du filtrage

Caractéristiques du jeu de données issu de l'étape de filtrage :

- **Passage de 320.772 lignes à 99.316 lignes :**
 - produits avec code-barres unique
 - produits vendus en France

- **Passage de 162 colonnes à 15 colonnes :**
 - taux de remplissage des données > 50%
 - contient uniquement les 15 features nécessaires

Étapes de filtrage des lignes	Nb de lignes
Dataframe initial	320.772
Suppression des produits sans code-barres (23 produits)	320.749
Etude des doublons (0 doublon)	320.749
Produits vendus en France métropolitaine	98.832
Produits vendus en France (y compris DOM TOM)	99.316

II. Opérations de nettoyage effectuées

1. Opérations de filtrage

- Suppression des produits sans code-barres et recherche de doublons
- Produits vendus en France
- Filtrage des colonnes

2. Gestion des valeurs aberrantes

- Contrôles de cohérences sur les données
- Méthode des écarts interquartiles

3. Gestion des valeurs manquantes

Gestion des valeurs aberrantes

■ Opérations de contrôle de cohérence :

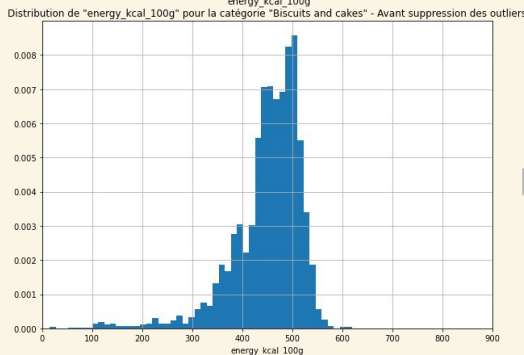
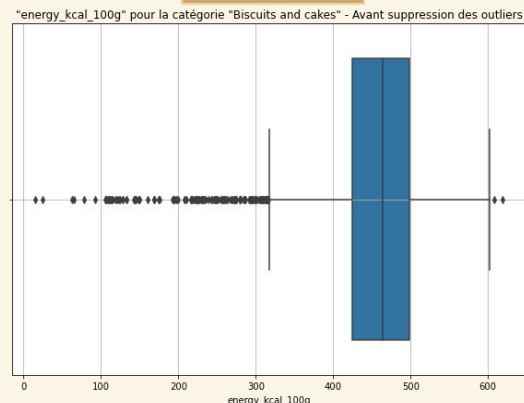
- Les valeurs nutritionnelles pour 100g de produit doivent rester comprises entre 0g et 100g.
- Le Nutri-Score doit être compris entre -15 et 40.
- La valeur énergétique maximale possible est de 900 kcal (correspond à l'huile).
- La quantité de graisses saturées doit rester inférieure à la quantité de graisses.
- La quantité de sucre doit rester inférieure à la quantité de carbohydrates.
- La somme des valeurs nutritionnelles doit rester inférieure à 100g.

■ Les valeurs aberrantes détectées sont remplacées par des NaN.

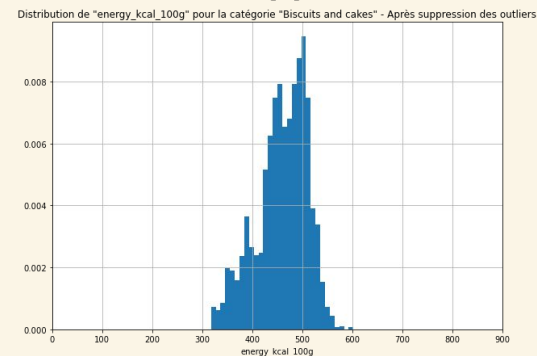
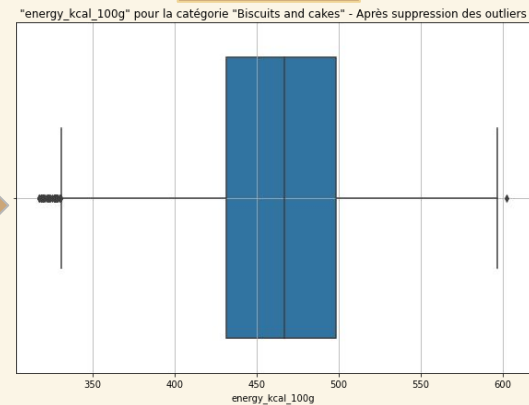
Méthode des écarts interquartiles

- Mise en oeuvre sur des **catégories homogènes** de produits :
 - Borne inf = $Q1 - 1.5 * (Q3 - Q1)$
 - Borne sup = $Q3 + 1.5 * (Q3 - Q1)$
- En dehors de cet intervalle, **les valeurs aberrantes détectées sont remplacées par NaN.**
- Exemple :
 - *Variable* : valeur énergétique
 - *Catégorie* : gâteaux

AVANT



APRÈS



Synthèse du traitement des valeurs aberrantes

- Méthode des écarts interquartiles automatisée : pour toutes les variables numériques et sur l'ensemble des catégories de produits.

Valeurs manquantes	Traitement des outliers	
	AVANT	APRÈS
<i>En nombre</i>	168.717	194.348
<i>En % des données</i>	20,46 %	23.57 %

Valeurs manquantes avant et après traitement des outliers



Introduction de 3% de NaN supplémentaires

- Dernier filtrage sur les lignes trop peu remplies ⇒ Passage à 42.269 lignes et 54.471 NaN

II. Opérations de nettoyage effectuées

1. Opérations de filtrage

- Suppression des produits sans code-barres et recherche de doublons
- Produits vendus en France
- Filtrage des colonnes

2. Gestion des valeurs aberrantes

- Contrôles de cohérences sur les données
- Méthode des écarts interquartiles

3. Gestion des valeurs manquantes

- Imputation par la moyenne
- fonction `IterativeImputer()`
- Algorithme kNN

Imputation par la moyenne

■ Mise en oeuvre pour les variables :

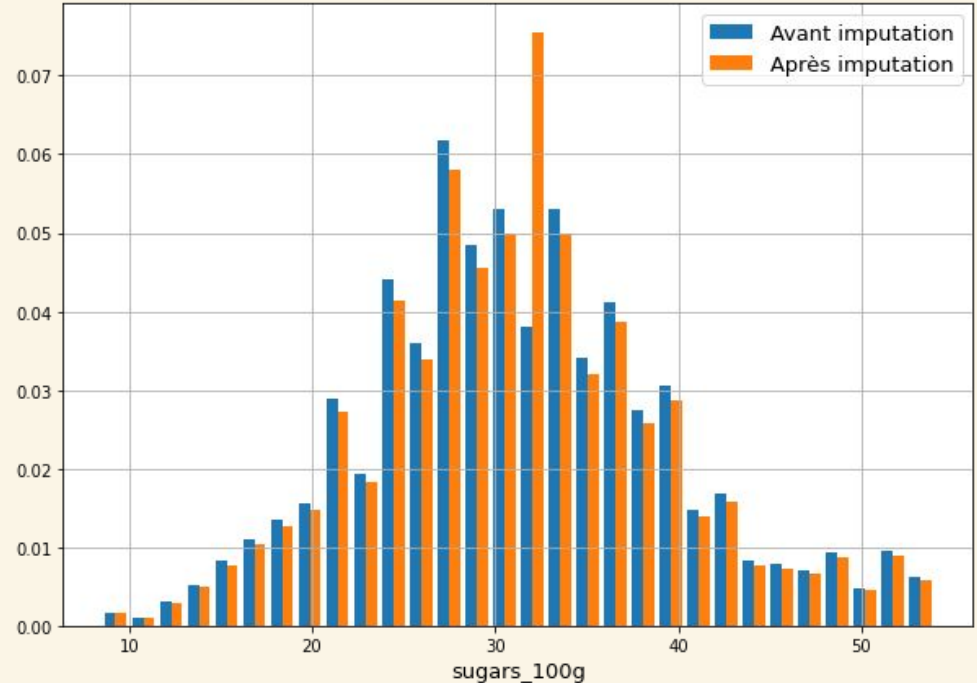
- *sugar_100g*
- *carbohydrates_100g*
- *salt_100g*
- *sodium_100g*
- *proteins_100g*

■ Sur des catégories homogènes de produits.

■ Exemple de distribution avant/après imputation par la moyenne :

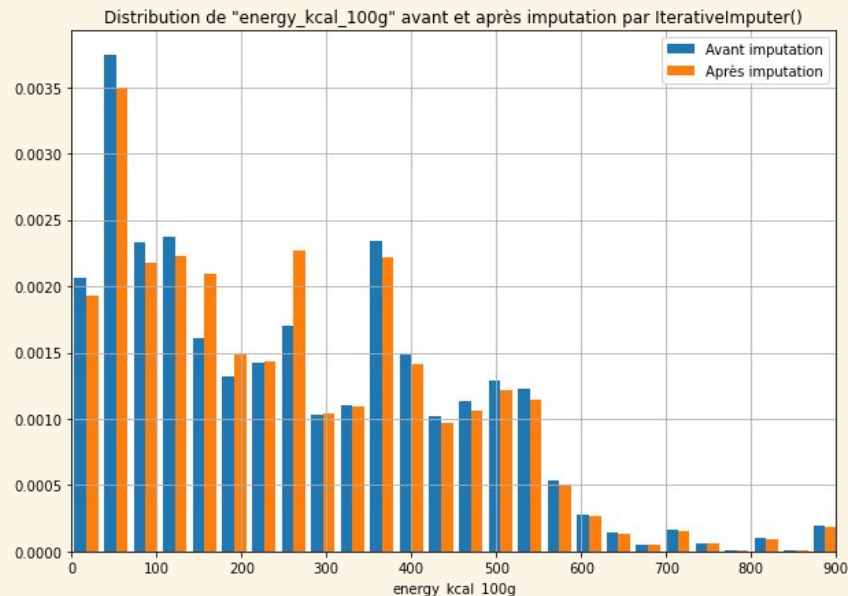
- *Variable* : sucre
- *Catégorie* : gâteaux

Distribution de "sugars_100g" pour la catégorie "Biscuits and cakes" : avant et après imputation par la moyenne



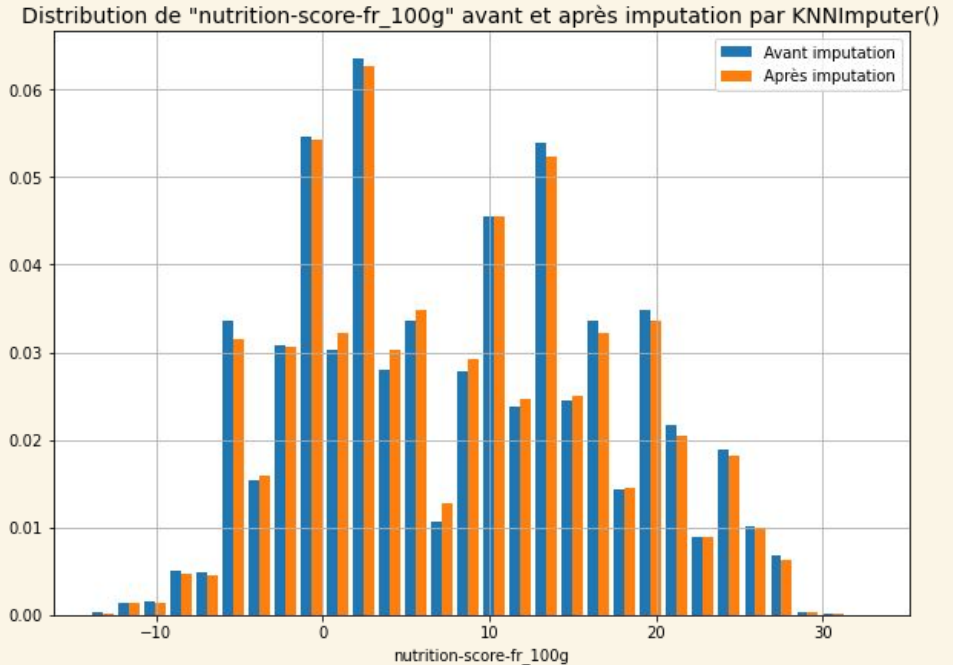
Imputation par IterativeImputer()

- **Stratégie d'imputation itérative** : modélise, tour à tour, chaque variable en fonction des autres, à l'aide d'une régression qui permet ensuite de prédire les valeurs manquantes de la variable.
- **Mise en oeuvre pour 3 variables** qui semblent corrélées (d'après heatmap des corrélations) :
 - *energy_kcal_100g*
 - *fat_100g*
 - *saturated-fat_100g*
- **Exemple de distribution avant/après imputation par IterativeImputer()**
 - variable : *energy_kcal_100g*



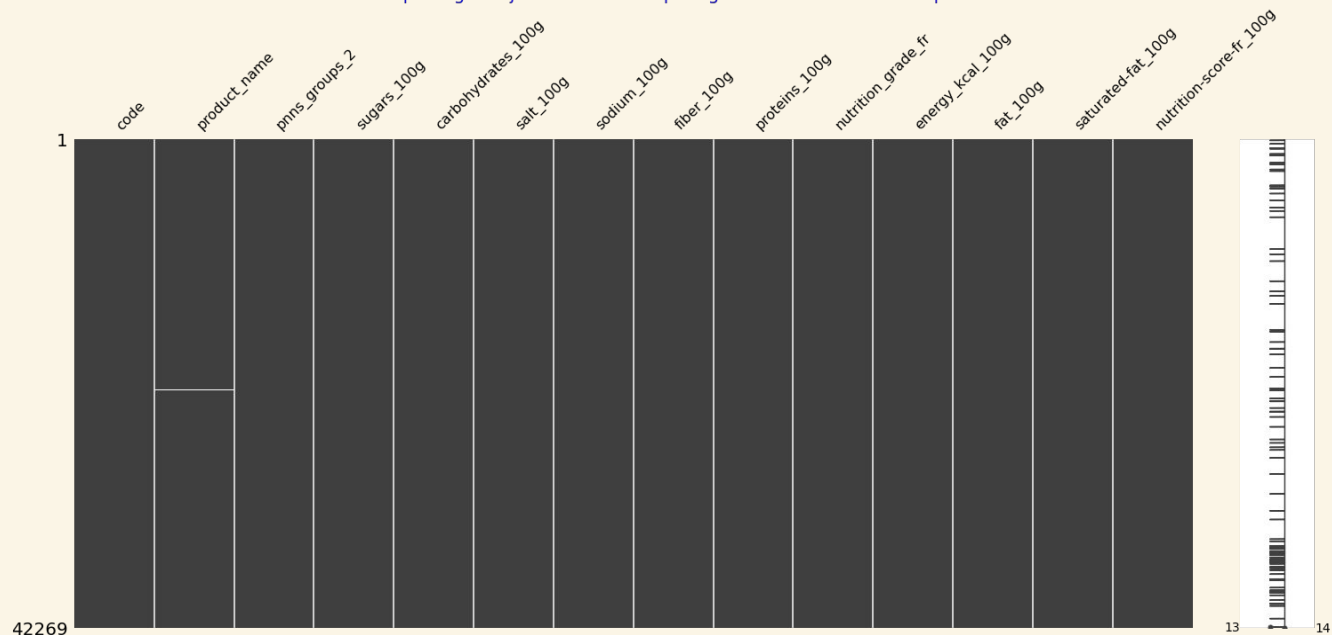
Imputation par l'algorithme kNN

- **Algorithme supervisé kNN** (k-Nearest Neighbors) : choisit les k données les plus proches du point étudié et remplace la valeur manquante par la moyenne de ces k plus proches voisins.
- Mise en oeuvre pour la variable : *'nutrition-score-fr_100g'*, en utilisant les variables qui entrent en jeu dans le calcul du Nutri-Score :
 - *'sugars_100g'*
 - *'salt_100g'*
 - *'saturated-fat_100g'*,
 - *'energy_kcal_100g'*
 - *'fiber_100g'*
 - *'proteins_100g'*
- **Résultat avant / après kNNImputer** pour la variable : *'nutrition-score-fr_100g'* avec $k=5$



Synthèse des opérations de nettoyage

Matrice de remplissage du jeu de données après gestion des valeurs manquantes



42.269 lignes / Aucune valeur manquante (sauf 98 noms de produits, mais non bloquant)

III. Analyse exploratoire des données

Analyse exploratoire des données

1. Analyse univariée

- Distribution des variables numériques
- Distribution des variables catégorielles

2. Analyse bivariée

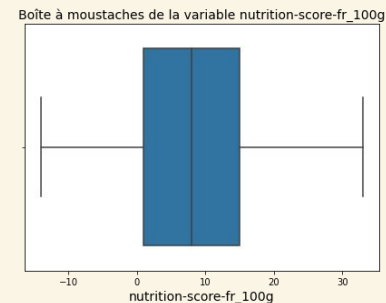
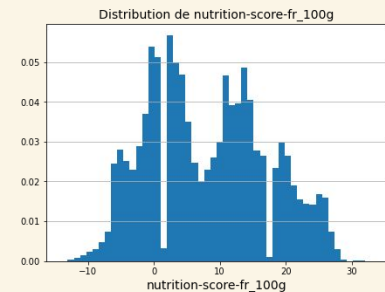
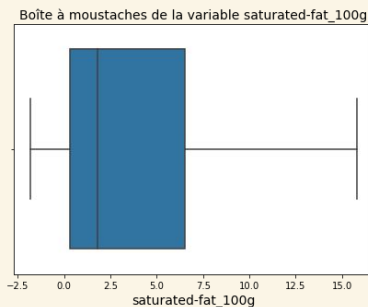
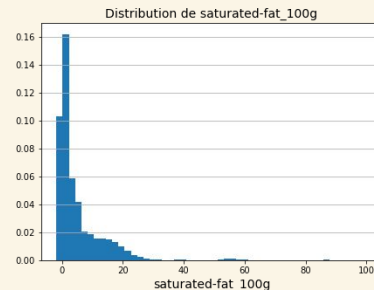
3. Analyse multivariée

Analyse univariée : variables numériques

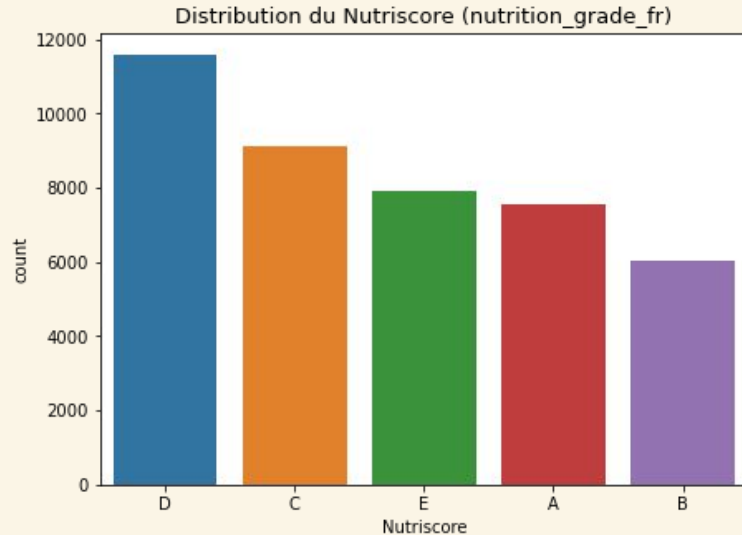
- Description statistique avec **.describe()**
- Représentation des distributions :
 - Diagramme en barres
 - Boîte à moustaches

```
# Statistiques descriptives des variables numériques
numerical_columns_df.describe().round(2)
```

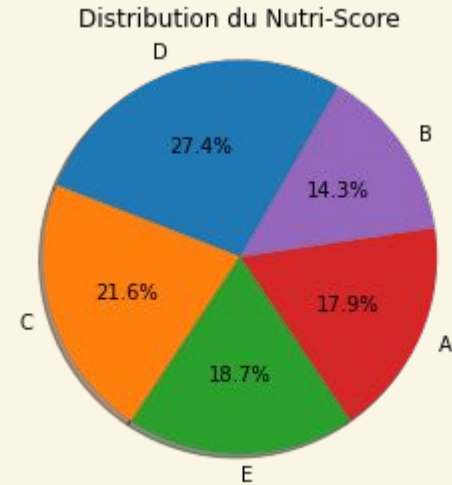
	sugars_100g	carbohydrates_100g	salt_100g	sodium_100g	fiber_100g	proteins_100g	energy_kcal_100g	fat_100g	saturated-fat_100g	nutrition-score-fr_100g
count	42256.00	42256.00	42256.00	42256.00	42256.00	42256.00	42256.00	42256.00	42256.00	42256.00
mean	11.81	25.95	0.74	0.29	1.26	7.33	256.21	12.72	5.04	8.21
std	16.90	26.98	0.85	0.33	2.14	6.99	183.74	16.57	7.90	8.94
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-6.05	-1.84	-14.00
25%	1.00	3.50	0.06	0.02	0.00	1.40	98.17	1.00	0.30	1.00
50%	3.70	12.00	0.53	0.21	0.00	6.00	233.59	6.50	1.80	8.00
75%	13.96	52.20	1.20	0.47	1.90	10.20	384.78	20.00	6.50	15.00
max	92.00	100.00	6.20	2.44	19.20	34.00	1043.91	124.63	100.00	33.00



Analyse univariée : variable catégorielle



Représentation de la distribution du Nutri-Score avec un diagramme en barres



Représentation de la distribution du Nutri-Score avec un diagramme en secteur

Analyse exploratoire des données

1. Analyse univariée

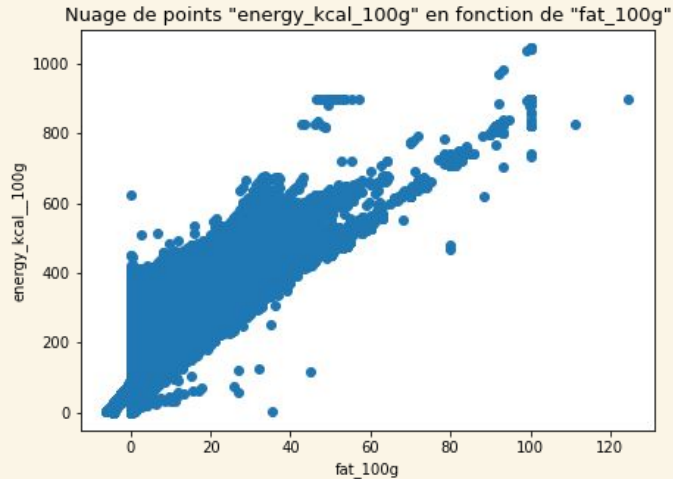
- Distribution des variables numériques
- Distribution des variables catégorielles

2. Analyse bivariée

- 2 variables numériques
- Variable numérique et variable catégorielle (ANOVA)

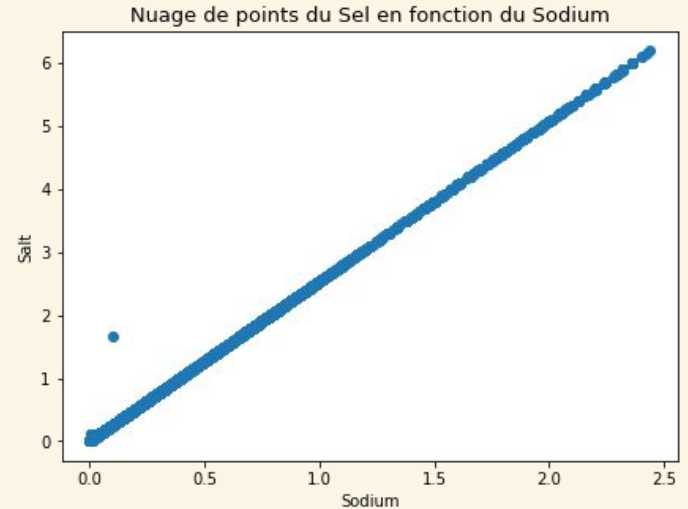
3. Analyse multivariée

Analyse de 2 variables numériques



```
# calculer Le coefficient de Pearson ou coeffecient de corrélation  
R=st.pearsonr(data['energy_kcal_100g'],data['fat_100g'])[0]  
round(R,5)
```

0.81467



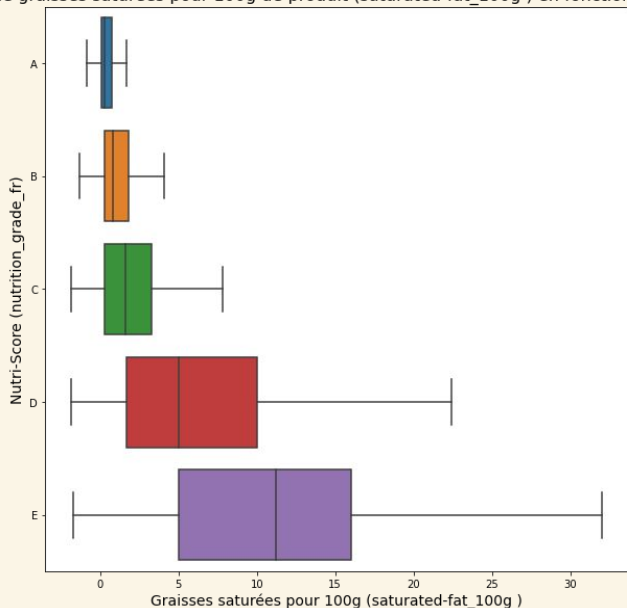
```
# calculer Le coefficient de Pearson ou coeffecient de corrélation  
R=st.pearsonr(data['salt_100g'],data['sodium_100g'])[0]  
round(R,5)
```

0.99997

Variable numérique et catégorielle : ANOVA

ANOVA : Variation d'une variable numérique en fonction des modalités de la variable catégorielle

Quantité de graisses saturées pour 100g de produit (saturated-fat_100g) en fonction du Nutri-Score



```
# fonction permettant de calculer Le rapport de corrélation2 (eta carré ou eta squared)
def eta_squared(x,y):
    moyenne_y = y.mean()
    classes = []
    for classe in x.unique():
        yi_classe = y[x==classe]
        classes.append({'ni': len(yi_classe),
                        'moyenne_classe': yi_classe.mean()})

    #Variation totale
    SCT = sum([(yj-moyenne_y)**2 for yj in y])
    #Variation interclasse
    SCE = sum([c['ni']* (c['moyenne_classe']-moyenne_y)**2 for c in classes])
    return SCE/SCT
```

```
# Calcul Le rapport de corrélation2 (eta carré ou eta squared) pour energy et nutriscore
```

```
X=data['nutrition_grade_fr']# qualitative
Y=data['saturated-fat_100g']# quantitative
```

```
round(eta_squared(X,Y),4)
```

0.2891

Rapport de corrélation : 0.3

Analyse exploratoire des données

1. Analyse univariée

- Distribution des variables numériques
- Distribution des variables catégorielles

2. Analyse bivariée

- 2 variables numériques
- Variable numérique et variable catégorielle (ANOVA)

3. Analyse multivariée

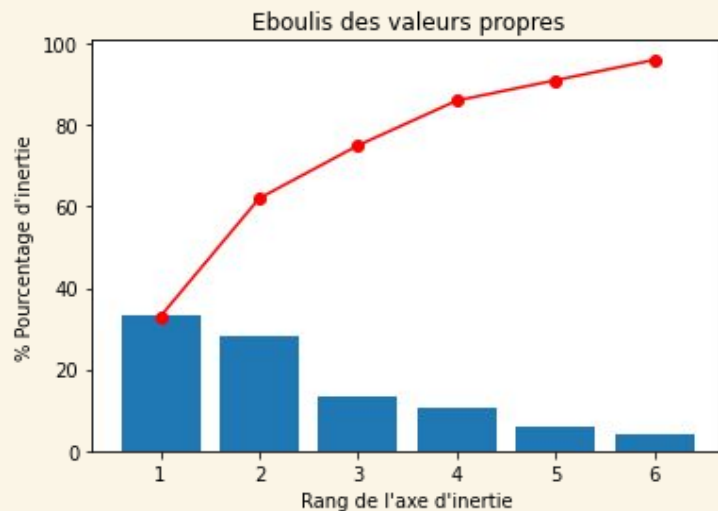
- Analyse en composantes principales

Analyse en composantes principales (ACP)

Objectif : Regrouper les variables liées en nouvelles variables synthétiques pour réduire le nombre de dimensions des données.

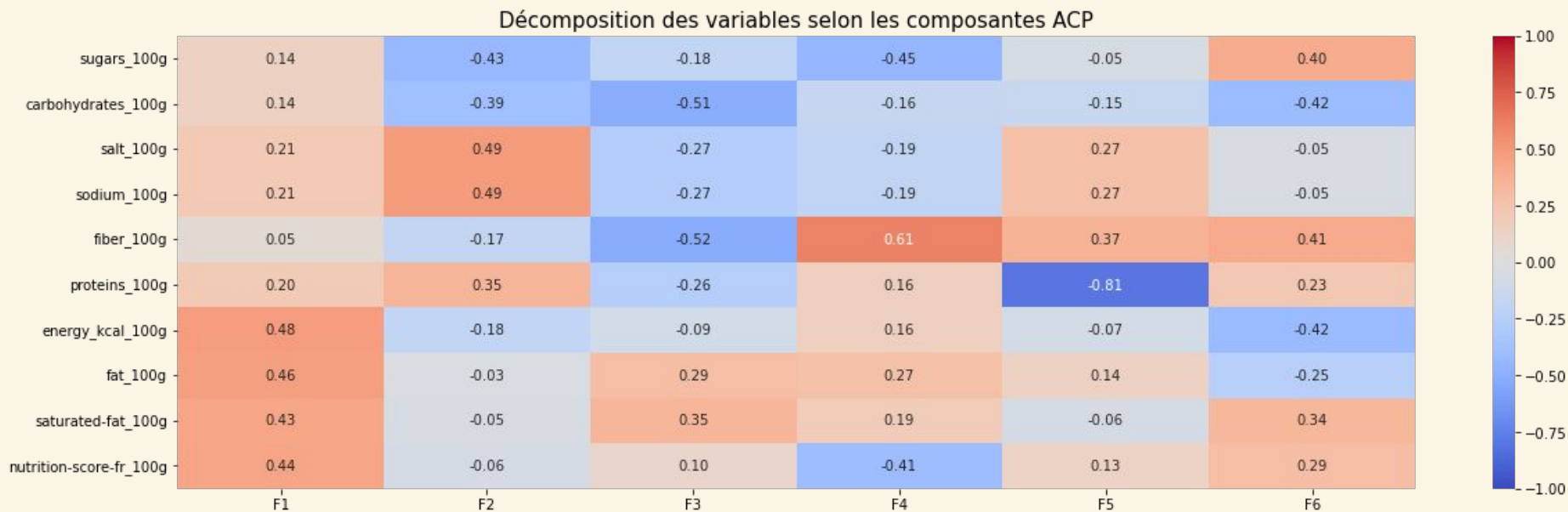
Etapes :

1. Split des données (séparations données, features et index).
2. Scaling : centrer et réduire les données avec *StandardScaler()*.
3. Calcul de l'ACP avec 6 composantes.
4. Calcul de la variance retenue et de l'éboulis des valeurs propres.

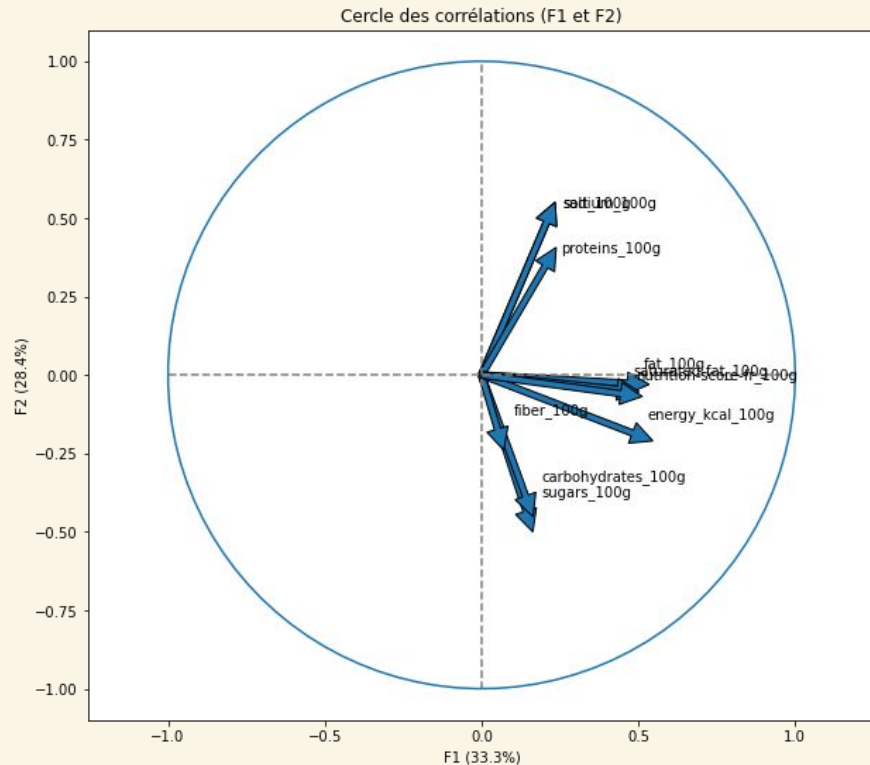


- 75% de la variance captée avec les 3 premières composantes.
- 85% avec les 4 premières composantes.

ACP : Calcul des composantes



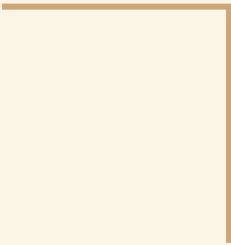
ACP : Cercle des corrélations



- **Les variables les plus corrélées à F1 :**
 - le nutriscore
 - l'énergie
 - les graisses
 - les graisses saturées
- **Les variables les plus corrélées à F2 :**
 - Sel et sodium
 - Sucre et carbohydrates (avec corrélation négative)

Conclusion

Compte tenu des opérations de nettoyage effectuées et de l'analyse réalisée sur le jeu de données, la base Open Food Facts semble pertinente pour mettre en œuvre l'idée d'application.



Merci pour votre attention !
Des questions ?

