

PROJET N°3 :

CONCEVEZ UNE APPLICATION AU SERVICE DE LA SANTÉ PUBLIQUE

Soutenance du P3: le 22/02/2022

Version notebook : **6.3.0**
Version Python : **3.8.8**
Version Pandas : **1.2.4**
Version Seaborn : **0.11.1**
Version Matplotlib: **3.3.4**



- ❖ **Idée d'application : Nutri-Sport**
- ❖ **Nettoyage du data-set**
- ❖ **Analyse exploratoire**
- ❖ **Conclusion**



A. Idée d'application

Nutri-Sport:



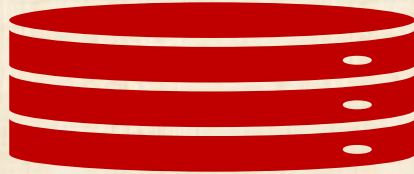
L'idée d'application est de calculer un score en scannant un produit, ce score est basé sur les produits qui contiennent plus de protéines ainsi que des nutriments énergétiques faciles à absorber dans le but de couvrir le besoin des sportifs, puis l'application va suggérer des produits de même catégorie et score afin d'avoir plus de choix de produits. Le score sera représenté par trois modalités: **Mauvais, Moyen, Excellent.**

Le choix de variables: vu que l'application va traiter un besoin sportif, donc on doit se focaliser sur les valeurs nutritionnelles comme la protéine, le sucre, la matière grasse et d'autres, qui sont nécessaires pour calculer le score.



B. Nettoyage du data-set

B.1. Présentation du jeu données:



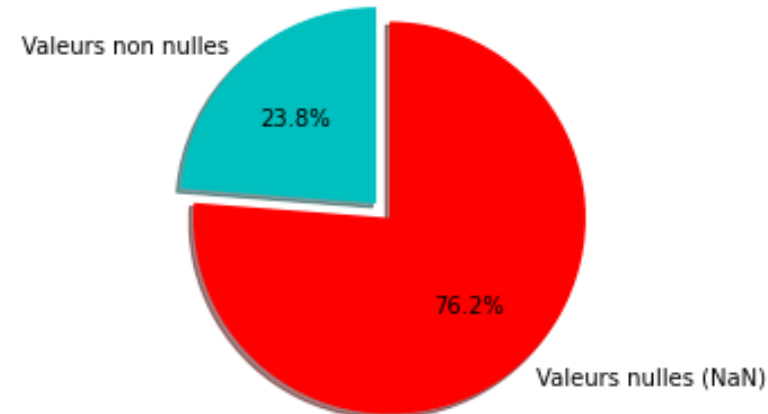
fr.openfoodfacts.org.products.csv

fr.openfoodfacts.org.products:

- Le fichier est volumineux, il contient plus de 320 000 produits alimentaires.
- Taille : 320772 lignes, 162 colonnes, nombre de cases: 51965064
- Nombre de valeurs nulles: 39608589
- Nombre de valeurs non nulles: 12356475
- Le pourcentage des valeurs nulles: 76.2 %
- Le pourcentage des valeurs non nulles: 23.8 %

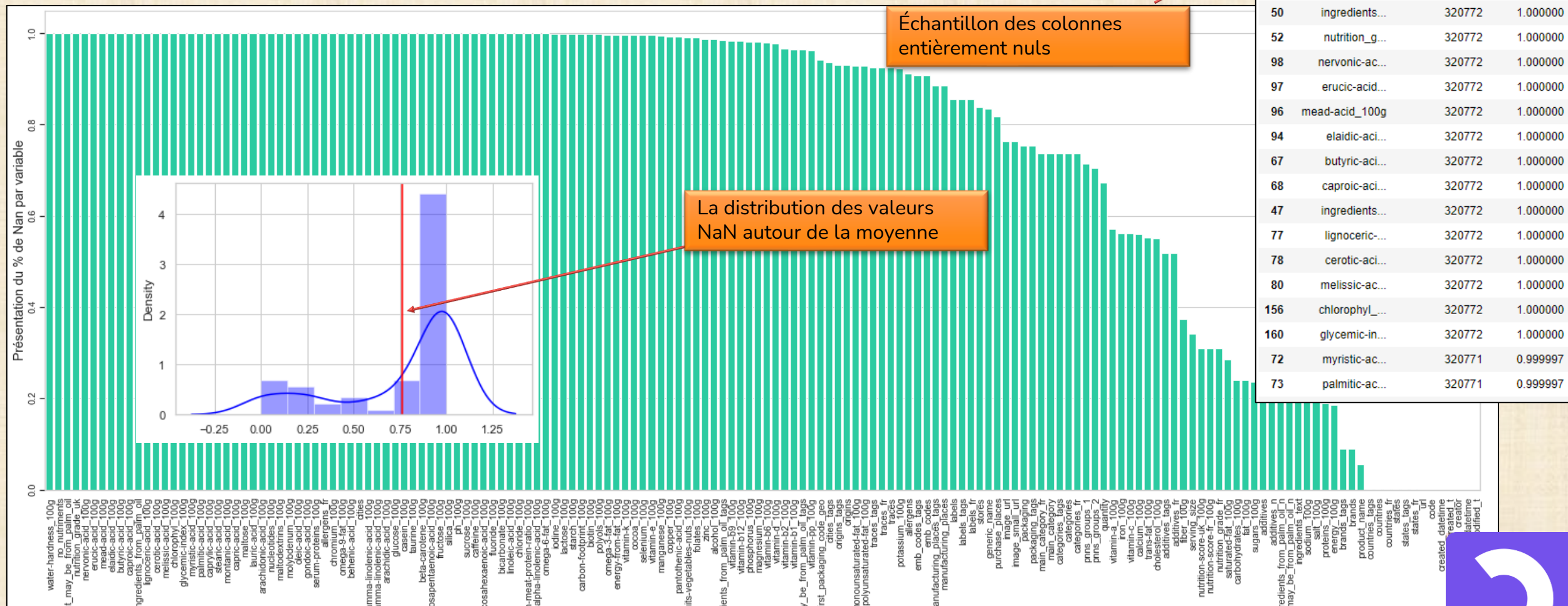
```
* Nombre de colonnes sans NaN -----: 2
* Nombre de colonnes NaN -----: 16
* Nombre de colonnes mixtes-----: 144
* Nombre de ligne entièrement nulles : 0
* Nombre de ligne mixtes ----- : 320772
* Nombre de ligne sans NaN -----: 0
* Nombre de lignes -----: 320772
* Nombre de colonnes -----: 162
* Nombre de cases -----: 51965064
* Nombre de valeurs nulles -----: 39608589
* Nombre de valeurs non nulles -----: 12356475
* le pourcentage des valeurs nulles -----: 76.2 %
* le pourcentage des valeurs non nulles --: 23.8 %
```

Le taux de remplissage en %



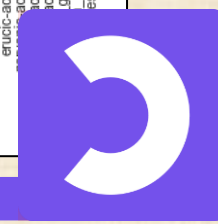
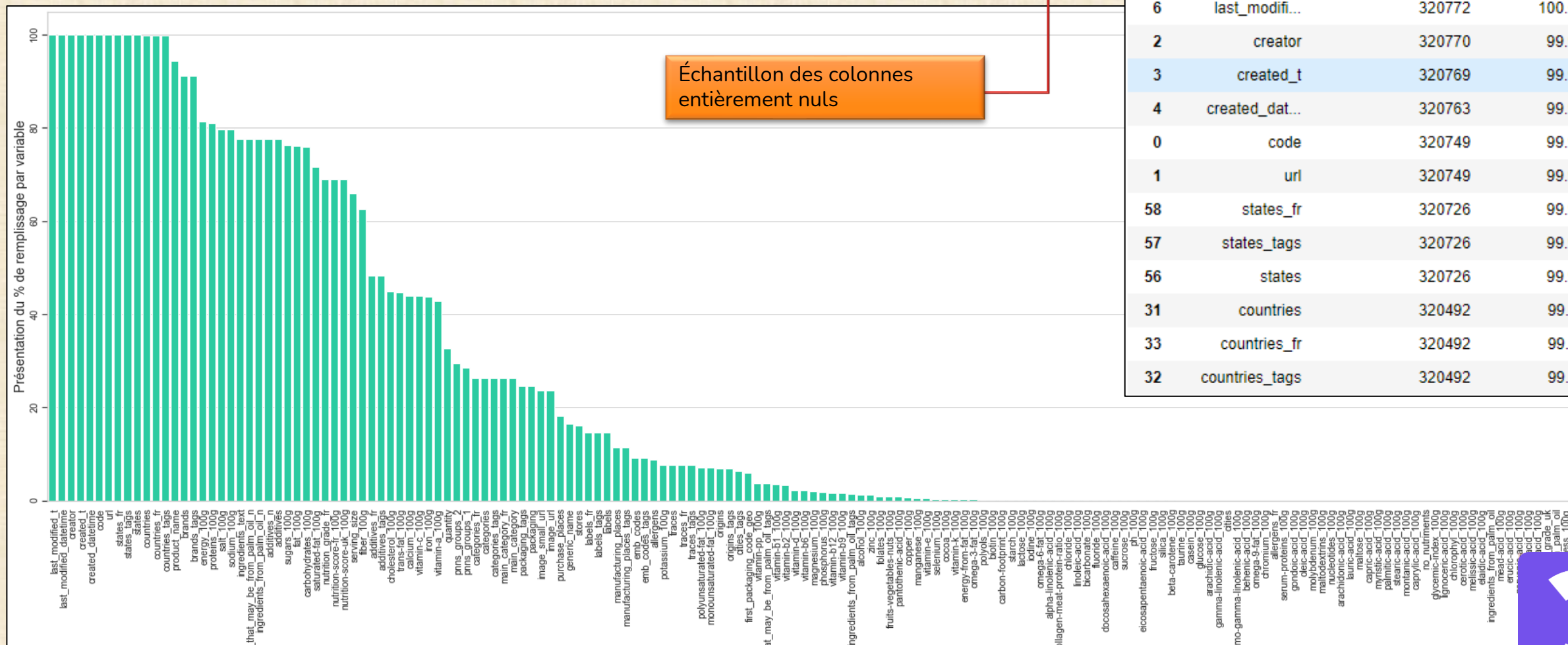
B. Nettoyage du data-set

B.1. Présentation des valeurs NaN de toutes les colonnes:



B. Nettoyage du data-set

B.1. Présentation des valeurs non nulles de toutes les colonnes:



B. Nettoyage du data-set

B.2. Filtration des colonnes au dessus de 20% de NaN:

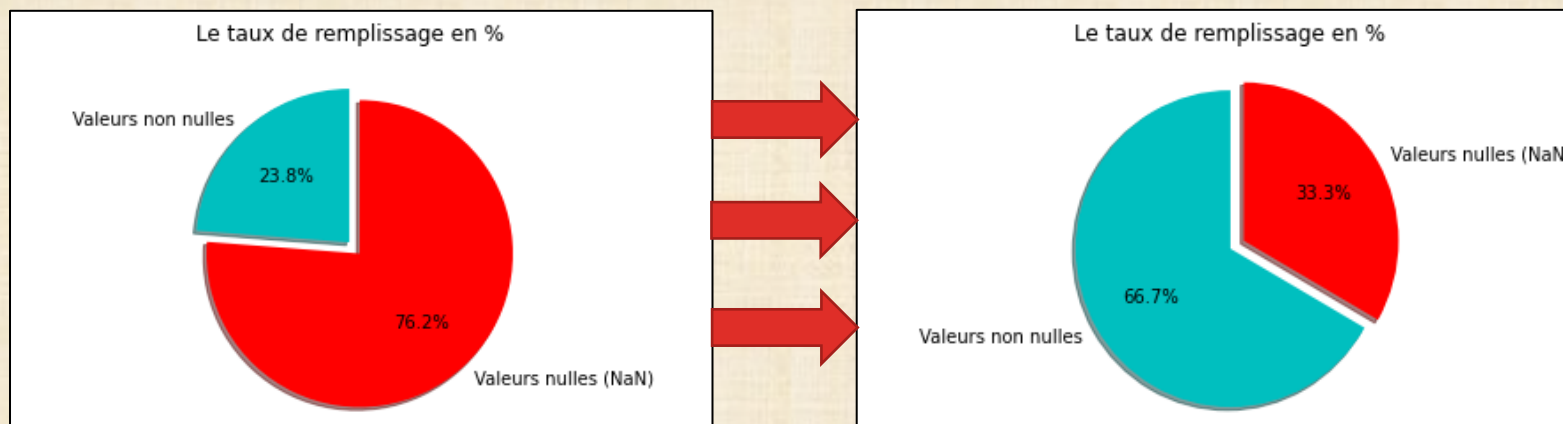
Le Via la fonction « `flt_nan` » on fixe le seuil à 80% pour supprimer les colonnes inférieur ou égale à 80% de NaN

```
prd_flt = flt_nan(prd, 80) # Suppression des colonnes qui contiennent de 80% de NaN  
prd_flt.shape
```

(320772, 54)

Le data-set est passé de 162 colonnes à 54.

En terme de pourcentage, le Data-set maintenant est à 66.7% de valeurs non nulles

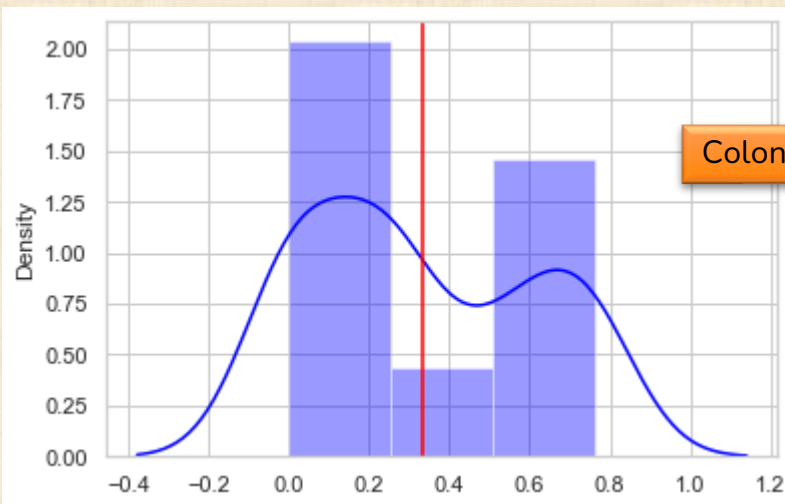


B. Nettoyage du data-set

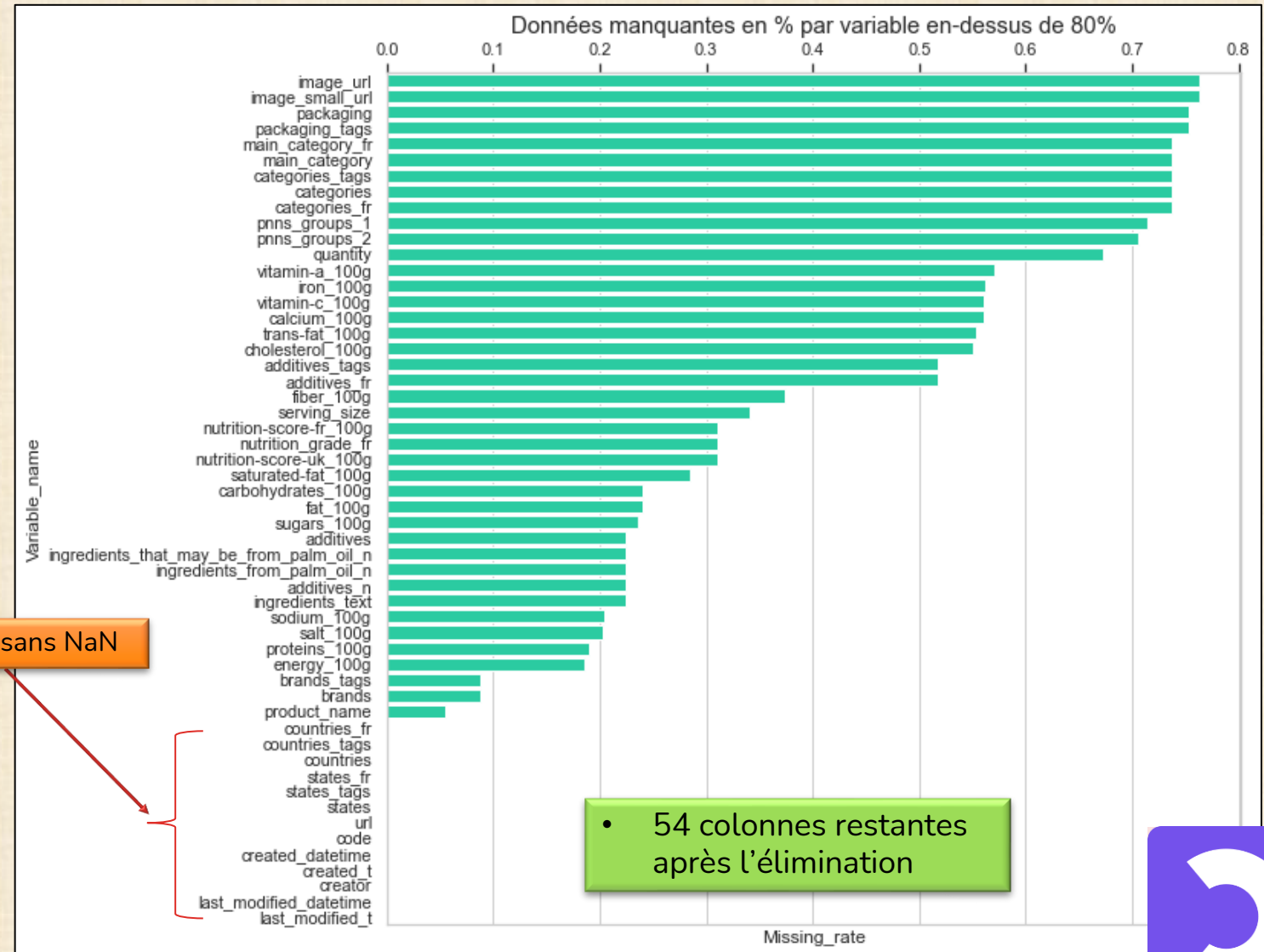
B.2. Filtration des colonnes NaN:

Sur le « Barplot » à droit, la variable 'image_url' a un pourcentage 75% de NaN, qui représentait 20% de NaN dans le DataSet non filtré

La distribution des valeurs NaN autour de la moyenne



Colonnes sans NaN



- 54 colonnes restantes après l'élimination



B. Nettoyage du data-set

B.3. Sélection des variable pour l'appli:

Après le premier filtrage, maintenant on doit garder que les variables qui nous intéressent pour notre application, en dessous la liste des variables potentielles

Des variables qui concernent le code, le nom du produit, la date de création, le producteur, la marque ainsi que le pays de provenance (Object)

Les modalités nutri-score, classés come suite :
'A', 'B', 'C', 'D', 'E', (variable catégorique)

'pnns_groups_1' et 'pnns_groups_2' concernent les catégories des produits ainsi les sous groupe (Object)

Les 10 variables concernent: l'énergie calculée pour 100g, 8 variables nutritionnelles, et en dernier le score nutritionnel (Int, Floats)

```
food_facts[list_food_facts].isna().sum().sum()
1504575

food_facts[list_food_facts].notna().sum()

code          320749
creator        320770
created_datetime 320763
product_name   303010
brands         292360
countries_fr   320492
nutrition_grade_fr 221210
pnns_groups_1  91513
pnns_groups_2  94491
energy_100g    261113
fat_100g       243891
saturated-fat_100g 229554
cholesterol_100g 144090
carbohydrates_100g 243588
sugars_100g    244971
fiber_100g     200886
proteins_100g  259922
salt_100g      255510
nutrition-score-fr_100g 221210
dtype: int64
```



B. Nettoyage du data-set

B.4. Traitement des valeurs aberrantes:

Pour traiter les valeurs aberrantes il existe plusieurs méthodes comme z-score, Interquartiles, percentile..., dans un premier temps on va prendre les seuils suivants:

Pour l'énergie **4000**, le reste des variables entre **0 et 100g**, **3.1g** Cholestérol, sauf nutri-score.

* on constate que toutes les nutriments maintenant n'ont pas de valeurs négatives sauf le nutrition score qui est normal car le nutri grade A contient des valeurs négatives

```
def min_max(df):
    for col in df.columns:
        if col == "energy_100g": seuil = 4000 # Le max q'on peut avoir dans 100g de nut
        elif col == "cholesterol_100g": seuil = 3.1 # Le max q'on peut avoir dans 100g de nut
        else: seuil = 100
        if df[col].dtypes != 'object':
            df[col] = df[col].apply(lambda x : np.nan if x < 0 or x > seuil else x)
    return df
```

```
%timeit min_max(food_facts[slct_ingr])
```

food_facts.describe()

	energy_100g	fat_100g	saturated-fat_100g	cholesterol_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g
count	2.611130e+05	243891.000000	229554.000000	144090.000000	243588.000000	244971.000000	200886.000000	259922.000000	255510.000000
mean	1.141915e+03	12.730379	5.129932	0.020071	32.073981	16.003484	2.862111	7.075940	2.028624
std	6.447154e+03	17.578747	8.014238	0.358062	29.731719	22.327284	12.867578	8.409054	128.269454
min	0.000000e+00	0.000000	0.000000	0.000000	0.000000	-17.860000	-6.700000	-800.000000	0.000000
25%	3.770000e+02	0.000000	0.000000	0.000000	6.000000	1.300000	0.000000	0.700000	0.063500
50%	1.100000e+03	5.000000	1.790000	0.000000	20.600000	5.710000	1.500000	4.760000	0.581660
75%	1.674000e+03	20.000000	7.140000	0.020000	58.330000	24.000000	3.600000	10.000000	1.374140
max	3.251373e+06	714.290000	550.000000	95.238000	2916.670000	3520.000000	5380.000000	430.000000	64312.800000

	energy_100g	fat_100g	saturated-fat_100g	cholesterol_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g
count	260959.000000	243887.000000	229551.000000	144085.000000	243569.000000	244952.000000	200881.000000	259918.000000	255352.000000
mean	1123.689921	12.725256	5.125925	0.018316	32.055323	15.985449	2.832473	7.079409	1.588497
std	795.927819	17.503695	7.913684	0.038483	29.140680	21.165362	4.588272	8.156552	6.242531
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	377.000000	0.000000	0.000000	0.000000	6.000000	1.300000	0.000000	0.700000	0.063500
50%	1100.000000	5.000000	1.790000	0.000000	20.600000	5.710000	1.500000	4.760000	0.580000
75%	1674.000000	20.000000	7.140000	0.020000	58.330000	24.000000	3.600000	10.000000	1.371600
max	4000.000000	100.000000	100.000000	1.580000	100.000000	100.000000	100.000000	100.000000	100.000000

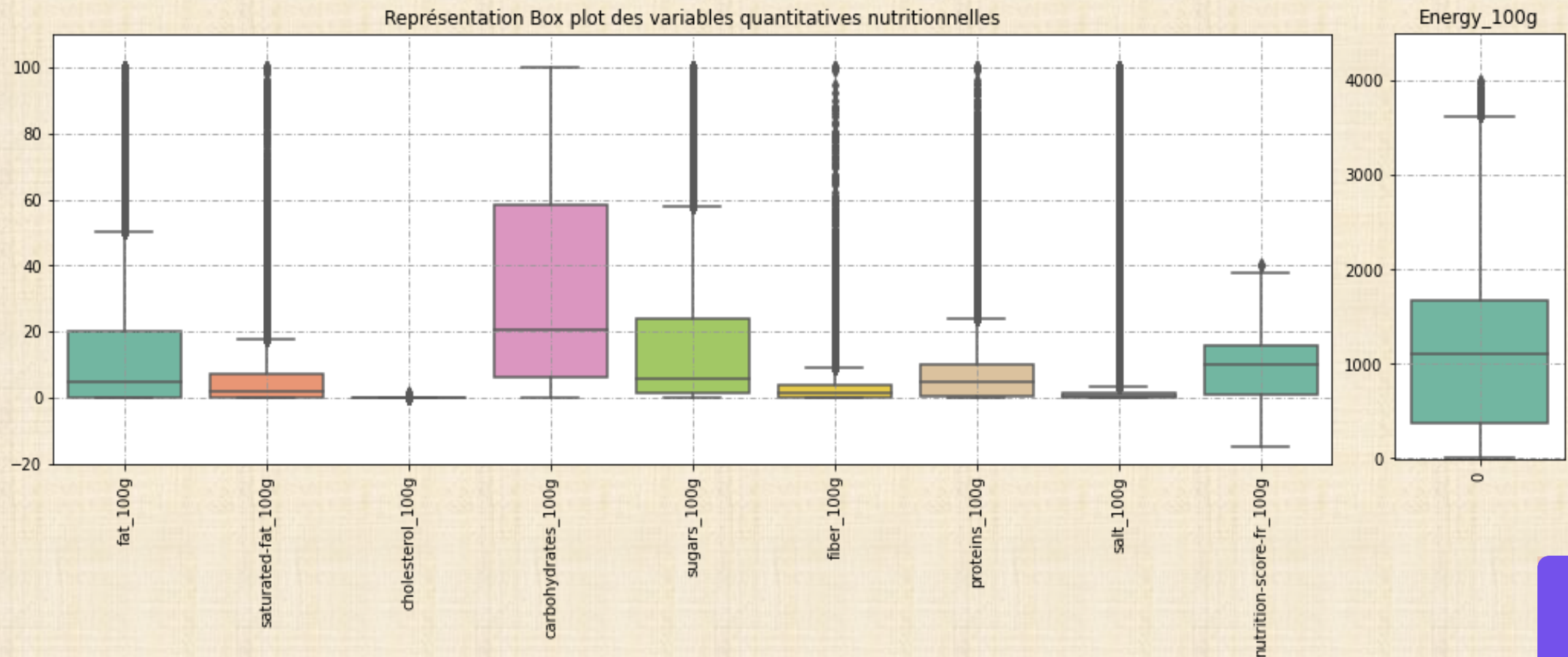
* Les valeurs au dessus des seuils sont remplacées par des Nan pour pouvoir après les remplacer par la moyenne de chaque sous catégorie



B. Nettoyage du data-set

B.4. Traitement des valeurs aberrantes: on va revenir sur ce point avec la méthode « **Percentile** » *

Le Boxplot en dessous représente les nutriments après le filtrage, maintenant il n'y a pas de valeurs négatives sauf le nutrition score qui est normal car le nutri grade A contient des valeurs négatives, Ainsi l'Energie est fixée à 4000



B. Nettoyage du data-set

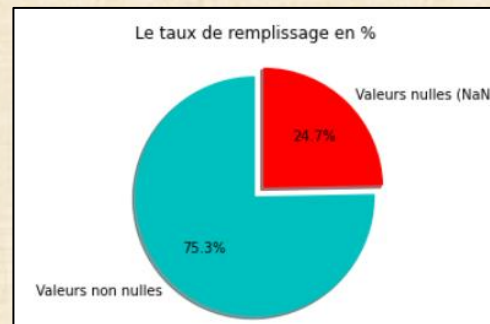
B.5. Traitement des lignes dupliquées et entièrement NaN:

- Suppression des lignes dupliquées

```
food_facts[food_facts.duplicated() == True].shape[0]
2

food_facts = food_facts.drop_duplicates()
food_facts[food_facts.duplicated() == True].shape[0]
0
```

Cela concerne les variables nutritionnelles sélectionnées, en dessous on constate que le dataset food_facts est passé de 320753 lignes à 262744 * de 74.7% de données ----> à 84.0% de données



- Suppression des lignes dupliquées en fonction des variables

"code", "product_name", "brands", "nutrition_grade_fr", "nutrition-score-fr_100g

```
food_facts[food_facts.duplicated(subset = ['code', 'product_name', 'brands', 'nutrition_grade_fr', 'nutrition-score-fr_100g'])]
19

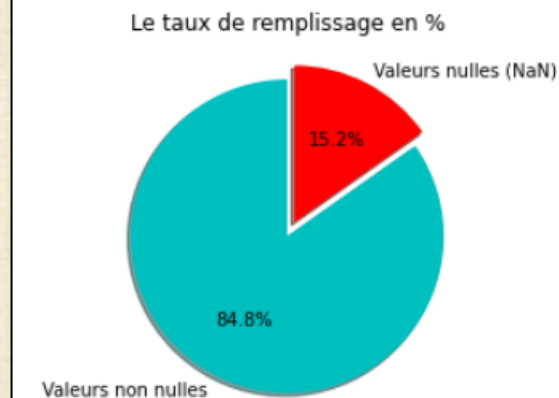
food_facts = food_facts.drop_duplicates(subset = ["code", "product_name", "brands", "nutrition_grade_fr", "nutrition-score-fr_100g"])
print(food_facts[food_facts.duplicated() == True].shape[0])
0
```

- Suppression des lignes entièrement NaN

```
food_facts = food_facts.dropna(how='all', subset = slct_ingr)
```

```
calc_inf(food_facts, False)
```

```
* Nombre de colonnes sans NaN -----: 1
* Nombre de colonnes NaN -----: 0
* Nombre de colonnes mixtes-----: 18
* Nombre de lignes -----: 262560
* Nombre de colonnes -----: 19
* Nombre de cases -----: 4988640
* Nombre de valeurs nulles -----: 760143
* Nombre de valeurs non nulles -----: 4228497
* le pourcentage des valeurs nulles -----: 15.2 %
* le pourcentage des valeurs non nulles --: 84.8 %
```



B. Nettoyage du data-set

B.6. Remplacement des valeurs NaN par la moyenne:

* Correction des noms de catégories 'pnns_groups_1' et 'pnns_groups_2' et remplacer les 'NaN' et 'Unknown' par 'To Define'

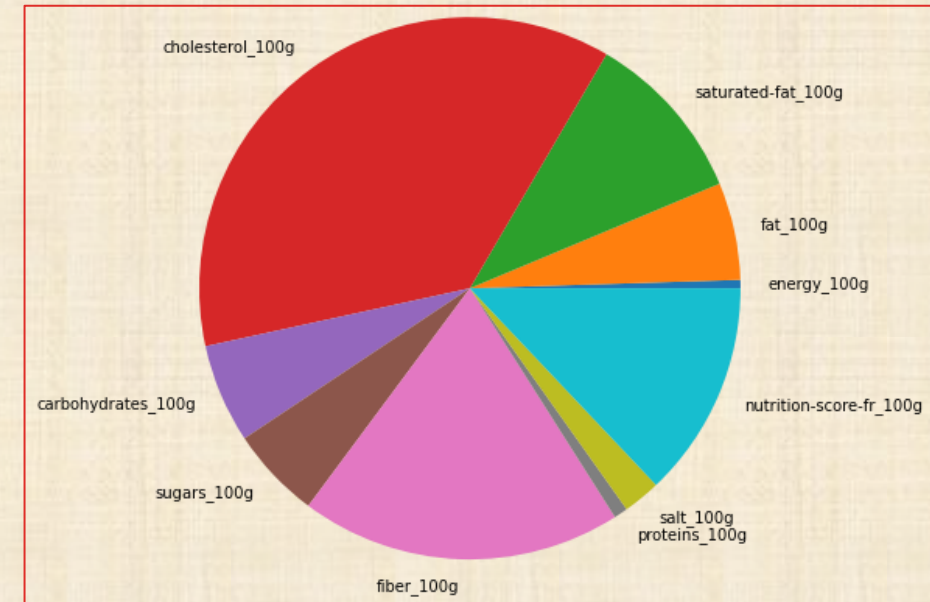
* De chaque sous catégorie pnns_groups_2 on remplace les NaN par la moyenne si la moyenne de chaque sous catégorie de produit par variable est différent de NaN

```
threshold = 0
for cat in categories:
    y = food_facts.pnns_groups_2 == cat
    for col in food_facts[y][select_ingr_1].columns:
        #
        mean_val = food_facts[food_facts['pnns_groups_2'] == cat][col].mean()
        std_val = food_facts[food_facts['pnns_groups_2'] == cat][col].std()
        if not np.isnan(mean_val) and std_val != 0:
            score = float(mean_val/std_val)
            #print(std_val)
            if score >= threshold:
                food_facts[col][y] = food_facts[col][y].apply(lambda x : mean_val if np.isnan(x) else x)
```

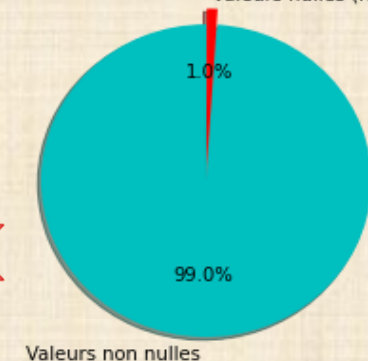
- On remarque qu'il reste que le 'cholesterol_100g' avec des NaN car la moyenne de cette variable dans plusieurs catég-produit égale à NaN, dans ce cas on remplace par la moyenne de la variable même.

```
* Nombre de colonnes sans NaN -----: 19
* Nombre de colonnes NaN -----: 0
* Nombre de colonnes mixtes-----: 0
* Nombre de lignes -----: 215764
* Nombre de colonnes -----: 19
* Nombre de cases -----: 4099516
* Nombre de valeurs nulles -----: 0
* Nombre de valeurs non nulles -----: 4099516
* le pourcentage des valeurs nulles -----: 0.0 %
* le pourcentage des valeurs non nulles --: 100.0 %
```

Distribution des NaN / variable



Le taux de remplissage en %
Valeurs nulles (NaN)



Pour le 1% des NaN qui reste on va le supprimer



B. Nettoyage du data-set

B.6. Remplacement la modalité To Define:

En dessous on a créé un dictionnaire qui contient les noms des catégories pnns1 et pnns2 pour remplacer les modalité 'To Define' aux produits non définis, cela concerne presque 5500 produits.

```
col_1 = ['pnns_groups_1']
col_2 = ['pnns_groups_2']
for k, v in diction.items():
    if k in diction:
        y = food_facts.product_name == k
        # v = ['pnns_groups_1', 'pnns_groups_2']
        b = v[0] # ['Milk and dairy products']
        a = v[1] # ['Ice Cream']
        #print(k, a, b)
        for index in food_facts[y].index: #[col]
            food_facts.at[index, col_1] = a
            food_facts.at[index, col_2] = b
```

On remarque que après le remplacement des modalités 'To Define' par leur catégories et sous catégories créées auparavant, le nombre de valeurs a diminué de 165763 à 160298

Un échantillon du dictionnaire créé

```
diction = {'Ice Cream': ['Ice Cream', 'Milk And Dairy Products'],
           'Extra Virgin Olive Oil': ['Fats', 'Fat And Sauces'],
           'Potato Chips': ['Salty And Fatty Products', 'Salty Snacks'],
           'Premium Ice Cream': ['Ice Cream', 'Milk And Dairy Products'],
           'Tomato Ketchup': ['Dressings And Sauces', 'Fat And Sauces'],
           'Beef Jerky': ['Processed Meat', 'Fish Meat Eggs'],
           'Pinto Beans': ['Legumes', 'Cereals And Potatoes'],
           'Cookies': ['Biscuits And Cakes', 'Sugary Snacks'],
           'Popcorn': ['Cereals', 'Cereals And Aotatoes'],
           'Salsa': ['Dressings And Sauces', 'Fat And Sauces'],
           'Tomato Sauce': ['Dressings And Sauces', 'Fat And Sauces'],
           'Cut Green Beans': ['Legumes', 'Cereals And Potatoes'],
           'Black Beans': ['Legumes', 'Cereals And Potatoes'],
           'Creamy Peanut Butter': ['Legumes', 'Cereals And Potatoes'],
           'Tortilla Chips': ['Appetizers', 'Salty Snacks'],
           'Apple Sauce': ['Dressings And Sauces', 'Fat And Sauces'],
           'Candy': ['Sweets', 'Sugary Snacks'],
           'Milk Chocolate': ['Chocolate Products', 'Sugary Snacks'],
           'Pasta Sauce': ['Dressings And Sauces', 'Fat And Sauces'],
           'Chicken Broth': ['Processed Meat', 'Fish Meat Eggs'],
           'Cottage Cheese': ['Cheese', 'Milk And Dairy Products'],
           '2% Reduced Fat Milk': ['Milk And Yogurt', 'Milk And Dairy Products'],
           'Greek Nonfat Yogurt': ['Milk And Yogurt', 'Milk And Dairy Products'],
```

```
food_facts['pnns_groups_1'].value_counts()
```

To Define	165763
-----------	--------

```
food_facts['pnns_groups_1'].value_counts()
```

To Define	160298
-----------	--------



B. Nettoyage du data-set

la méthode « **Percentile** » : *

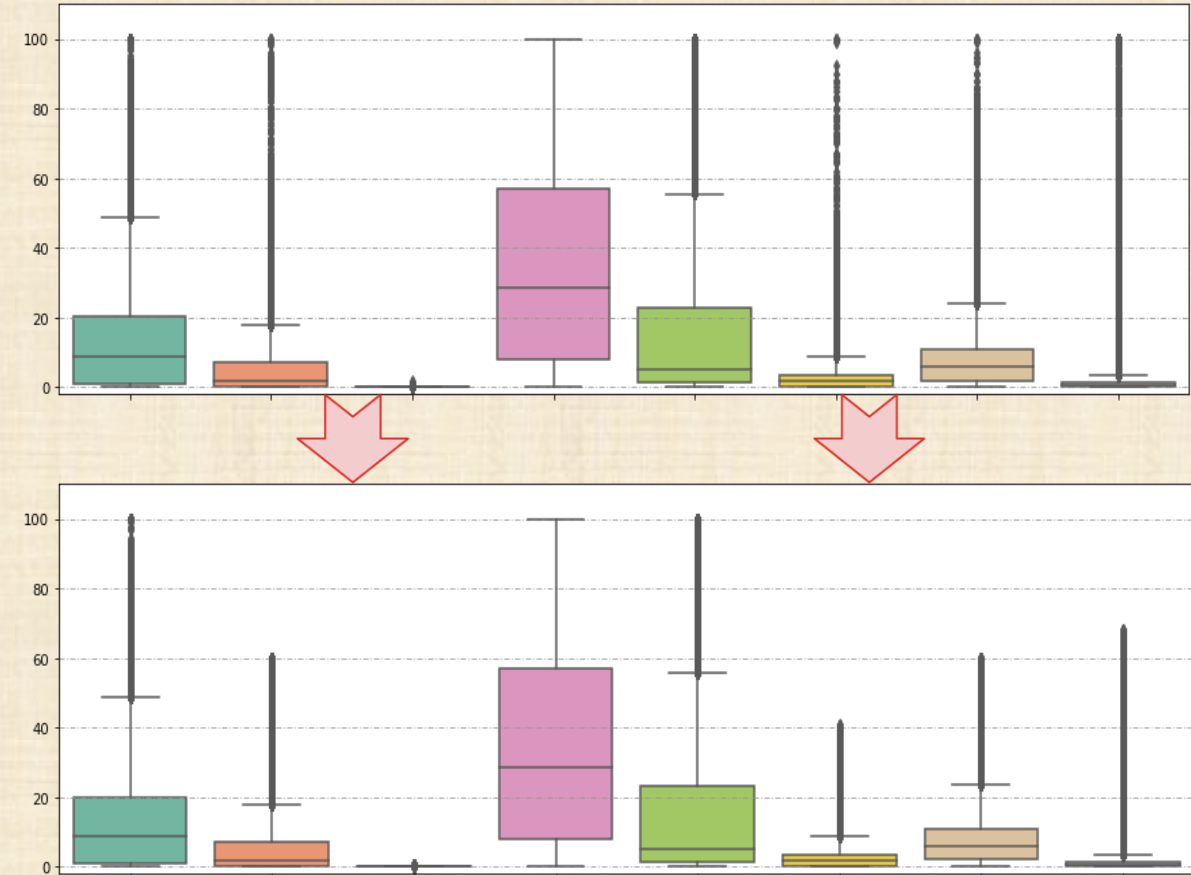
On remarque que malgré les seuils qu'on a pris (0 et 100g), on a toujours des valeurs atypiques comme illustré par exemple il y a des produits qui ont 100% de la protéine ou de la fibre ce qui n'est pas normale, donc pour supprimer ces valeurs on utilise la méthode percentile pour les éliminer

product_name	brands	countries_fr	fiber_100g	proteins_100g
Eau	Delhaize	France	0.0	0.0
Eau	Marque Repère	France	0.0	0.0
Eau	Vital	France	100.0	100.0

Au dessus l'exemple de l'eau qui contient 100g de protéines et de fibre, avec la méthode percentile on va supprimer toutes ces valeurs

fiber_100g	proteins_100g
215764.000000	215764.000000
2.720174	7.779048
4.236390	8.078766
0.000000	0.000000
0.000000	1.900000
1.600000	5.700000
3.500000	10.710000
100.000000	100.000000

proteins_100g	salt_100g
214722.000000	214722.000000
7.715329	1.151512
7.775538	2.832366
0.000000	0.000000
2.000000	0.100000
5.700000	0.650240
10.710000	1.361440
60.000000	68.200000

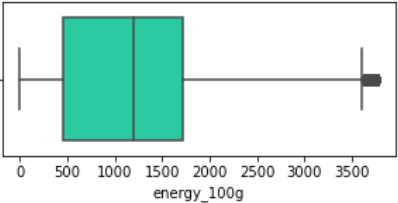
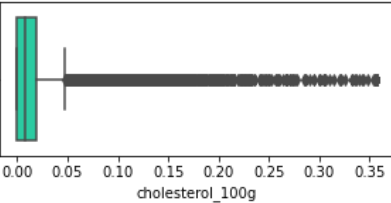
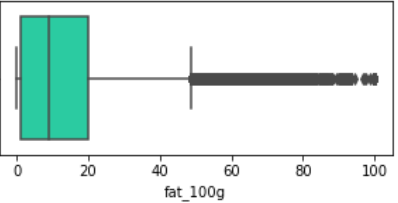
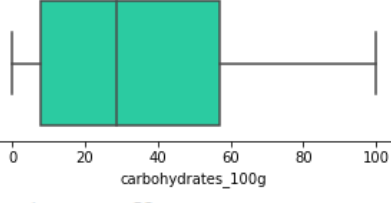
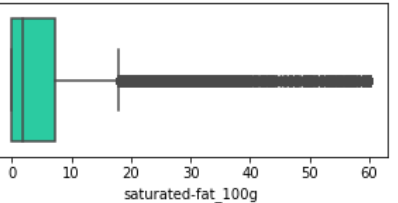
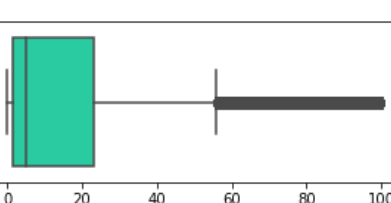


```
for col in food_facts_clean[slct_ingr].columns:
    if food_facts_clean[col].dtypes != 'object':
        food_facts_clean.loc[food_facts_clean[col] > food_facts_clean[col].quantile(0.999)] = np.nan
        #food_facts_clean.loc[food_facts_clean[col] < 0] = np.nan
food_facts_clean = food_facts_clean.dropna()
```

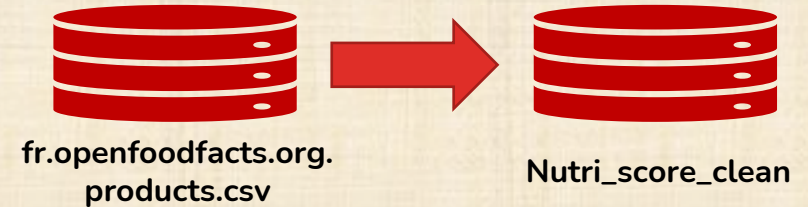


C. Analyse exploratoire

C.1. Analyse univariée: les indicateurs de distribution quantitatives et qualitatives

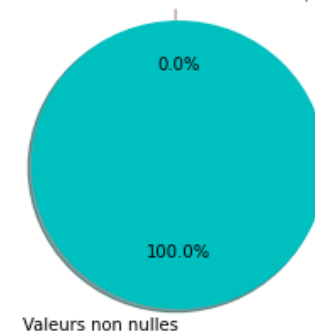
Indicateurs de distribution pour energy_100g count 214722.000000 mean 1173.973902 std 755.686635 min 0.000000 25% 452.000000 50% 1191.049891 75% 1712.675000 max 3766.000000 Name: energy_100g, dtype: float64 	Indicateurs de distribution pour cholesterol_100g count 214722.000000 mean 0.018251 std 0.029116 min 0.000000 25% 0.000000 50% 0.007369 75% 0.019019 max 0.356400 Name: cholesterol_100g, dtype: float64 
Indicateurs de distribution pour fat_100g count 214722.000000 mean 13.204849 std 15.323320 min 0.000000 25% 1.110000 50% 8.930000 75% 20.100000 max 100.000000 Name: fat_100g, dtype: float64 	Indicateurs de distribution pour carbohydrates_100g count 214722.000000 mean 33.225339 std 27.298482 min 0.000000 25% 7.900000 50% 28.570000 75% 57.140000 max 100.000000 Name: carbohydrates_100g, dtype: float64 
Indicateurs de distribution pour saturated-fat_100g count 214722.000000 mean 4.872341 std 7.094718 min 0.000000 25% 0.000000 50% 1.790000 75% 7.140000 max 60.000000 Name: saturated-fat_100g, dtype: float64 	Indicateurs de distribution pour sugars_100g count 214722.000000 mean 15.029976 std 19.810721 min 0.000000 25% 1.300000 50% 5.000000 75% 23.080000 max 100.000000 Name: sugars_100g, dtype: float64 

Après l'enregistrement du Dataset la taille du fichier est passée de **827Mo** à **43Mo**



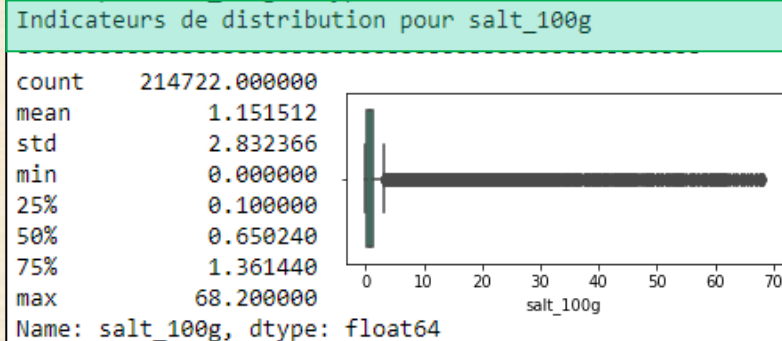
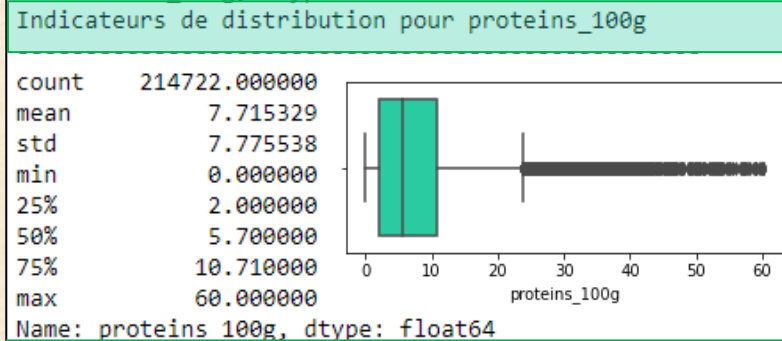
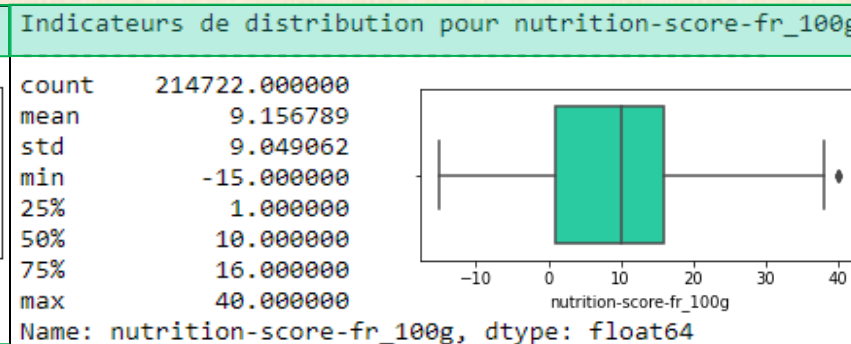
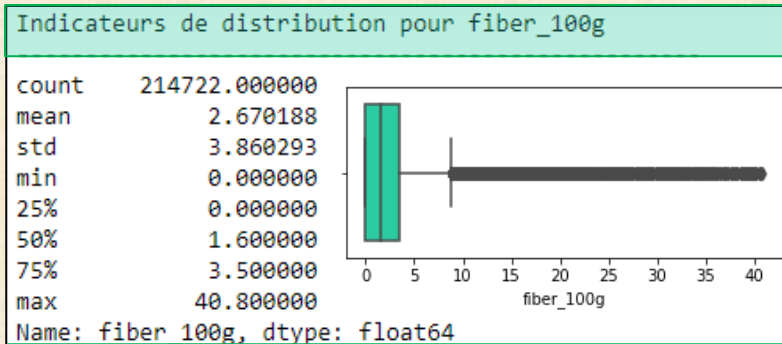
```
* Nombre de colonnes sans NaN -----: 19
* Nombre de colonnes NaN -----: 0
* Nombre de colonnes mixtes-----: 0
* Nombre de ligne entièrement nulles : 0
* Nombre de ligne mixtes ----- : 0
* Nombre de ligne sans NaN -----: 214722
* Nombre de lignes -----: 214722
* Nombre de colonnes -----: 19
* Nombre de cases -----: 4079718
* Nombre de valeurs nulles -----: 0
* Nombre de valeurs non nulles -----: 4079718
* le pourcentage des valeurs nulles -----: 0.0 %
* le pourcentage des valeurs non nulles --: 100.0 %
```

Le taux de remplissage en %
Valeurs nulles (NaN)



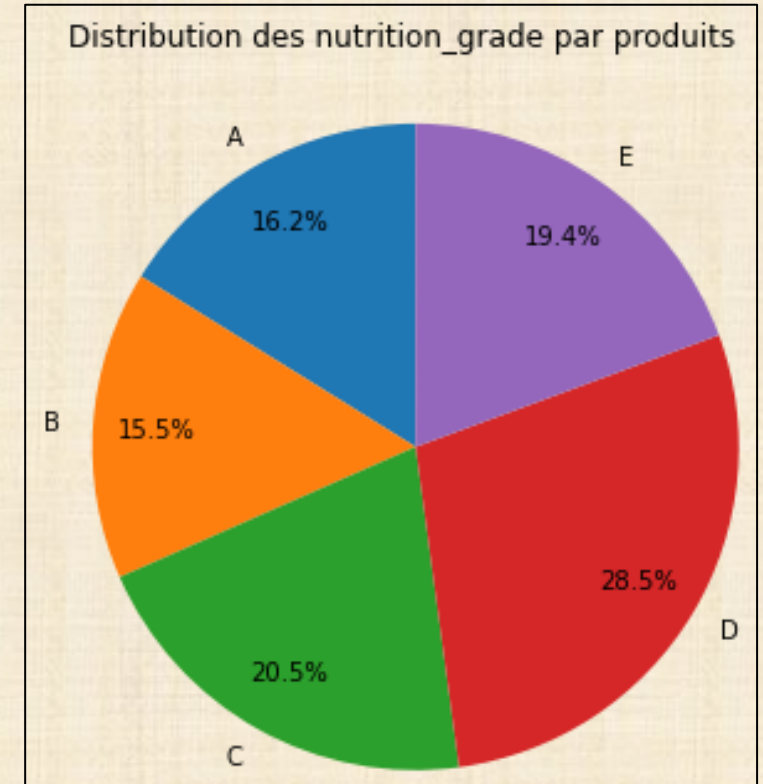
C. Analyse exploratoire

C.1. Analyse univariée: les indicateurs de distribution quantitatives et qualitatives



Variables quantitatives: présenté par le Boxplot et la méthode Describe()

- 'energy_100g',
- 'fat_100g',
- 'saturated-fat_100g',
- 'cholesterol_100g',
- 'carbohydrates_100g',
- 'sugars_100g',
- 'fiber_100g',
- 'proteins_100g',
- 'salt_100g',
- 'nutrition-score-fr_100g'



Variables qualitatives:

le nutrition_grade_fr est une variable qualitative ordinaire, qui indique le nutri-score des individus.



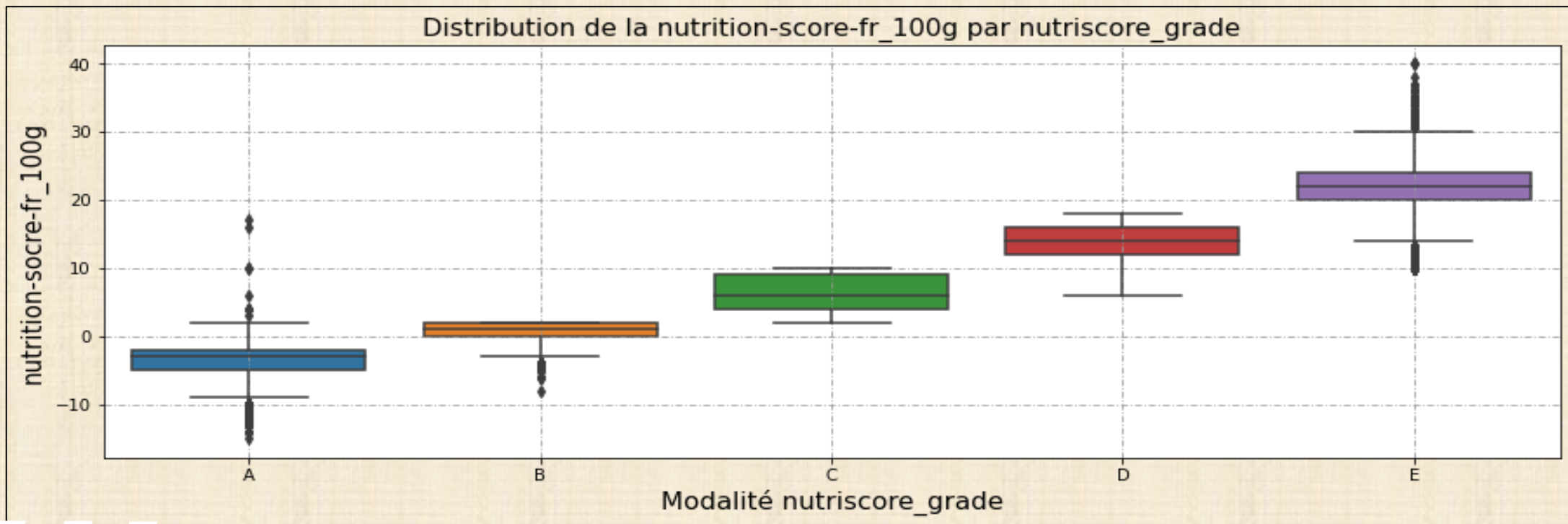
C. Analyse exploratoire

C.2. Analyse bivariable:

- La distribution de nutrition-scor-fr_100g par nutriscore_grade :

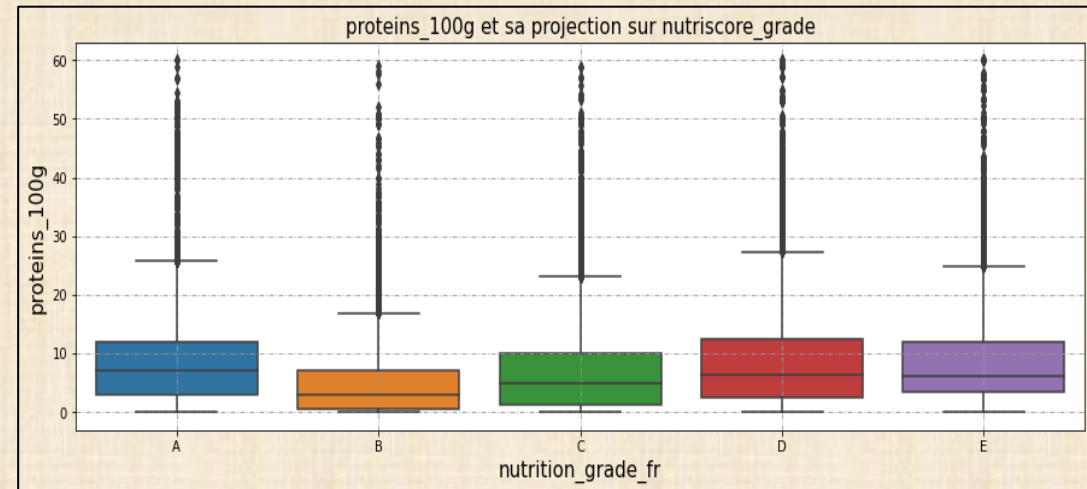
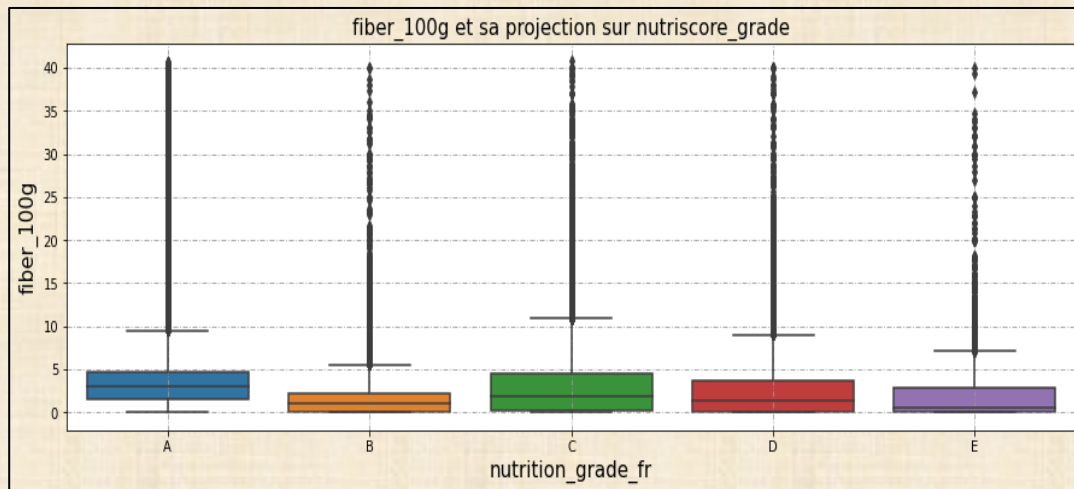
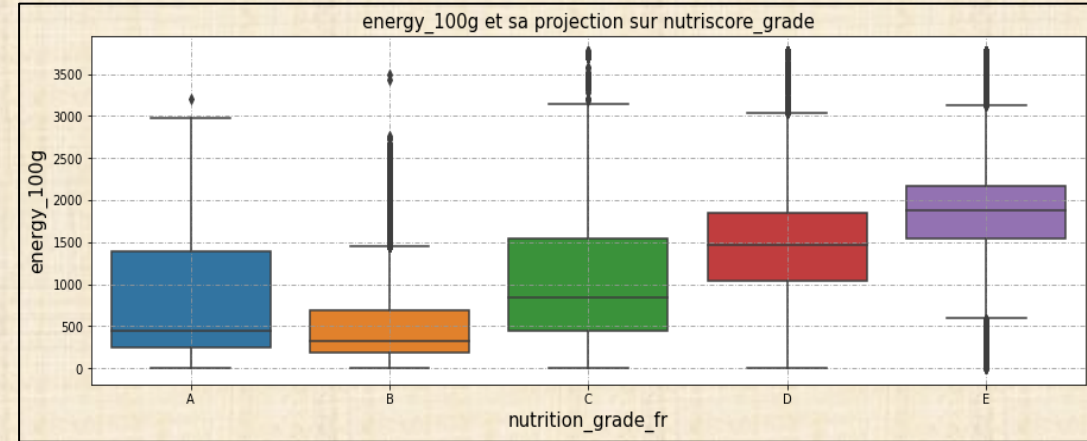
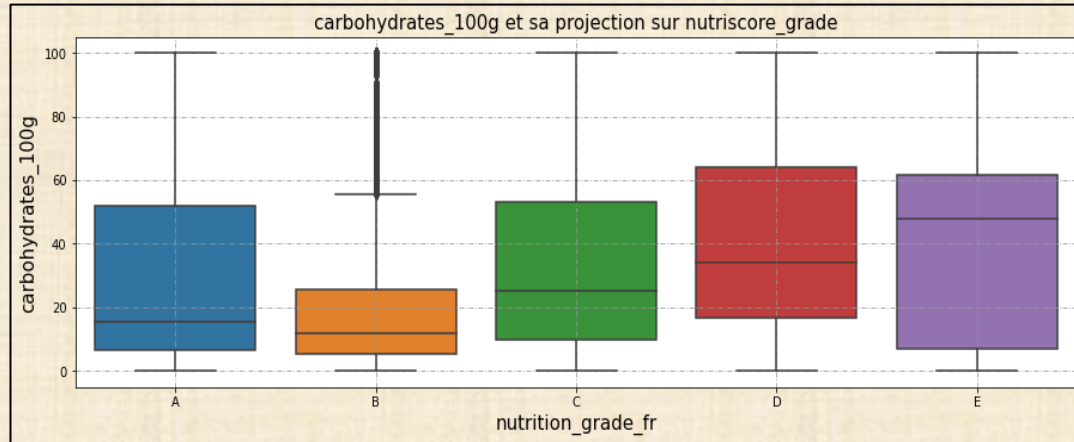
La distribution de la variable nutri-score évolue linéairement avec les modalités de la variable nutri-grade, clairement corrélée selon le Boxplot, à droite les intervalles de chaque nutri-grade en fonction de nutri-score

- A : entre -15 et 2
- B : entre 0 et 2
- C : entre 2 et 10
- D : entre 11 et 20
- E : entre 20 et 40

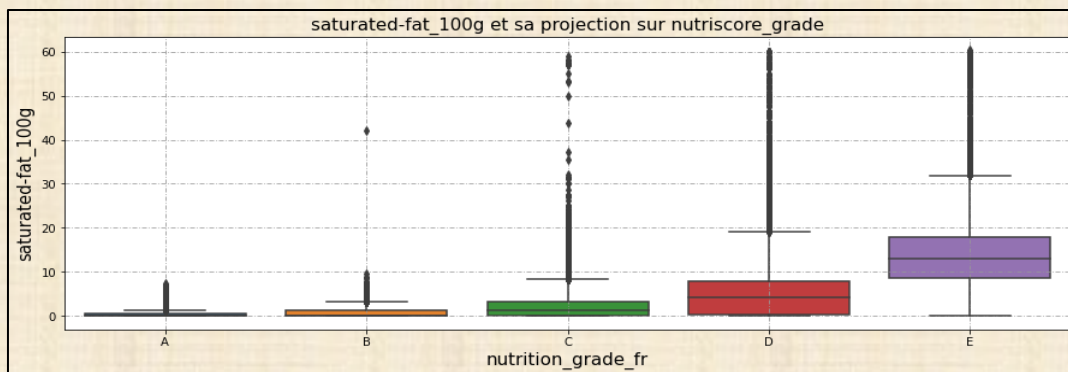
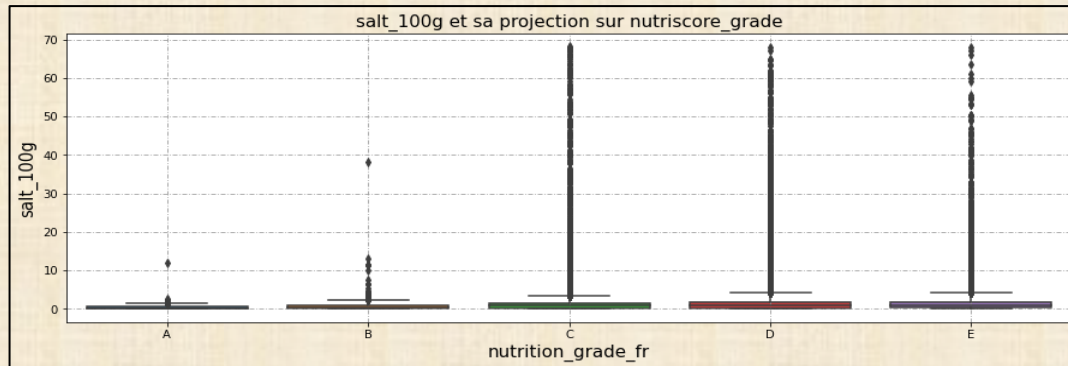
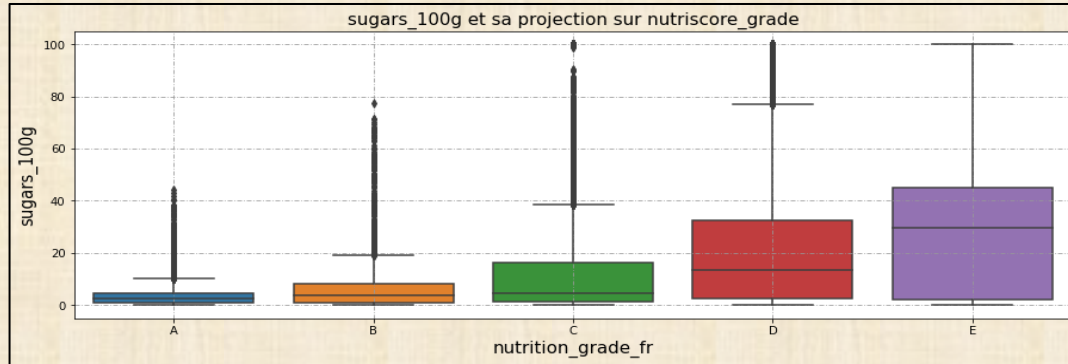


C. Analyse exploratoire

C.2. Analyse bivariée: la distribution des variables par nutriscore_grade

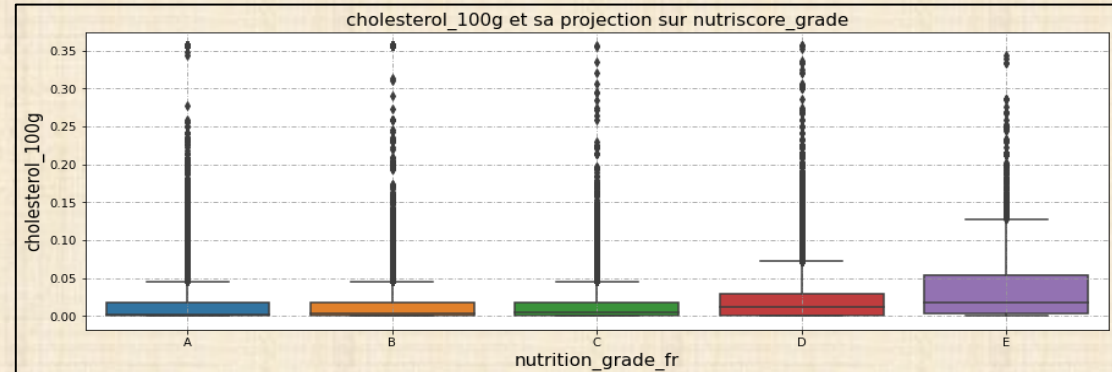
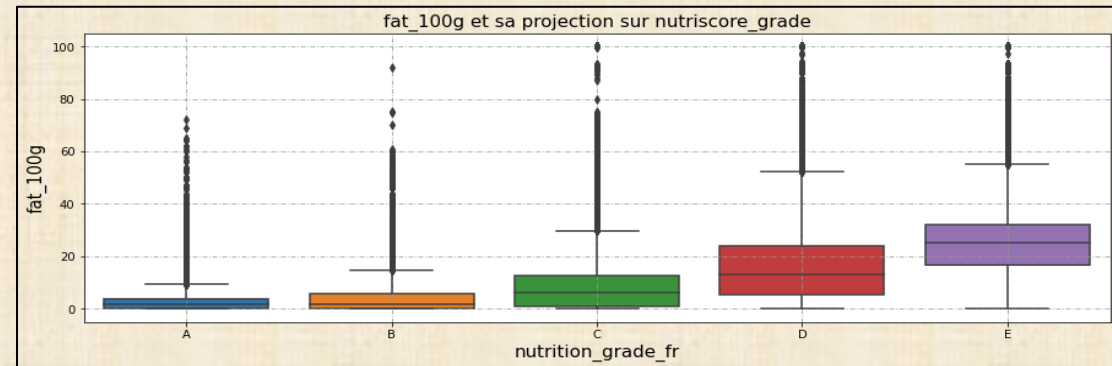


C. Analyse exploratoire



C.2. Analyse bivariable:

- L'ensemble de ces variables sont corrélées linéairement avec le nutri-grade, plus il y a de sucre ('sugars_100g'), de graisses ('fat_100g', 'saturated-fat_100g', 'cholesterol_100g') et de sel ('salt_100g') dans un produit et plus son Nutri-Score est mauvais.
- pour le reste des variables ce n'est pas le cas, donc le calcul de nutri-score est basé sur les nutriments qui évoluent de manière linéaire

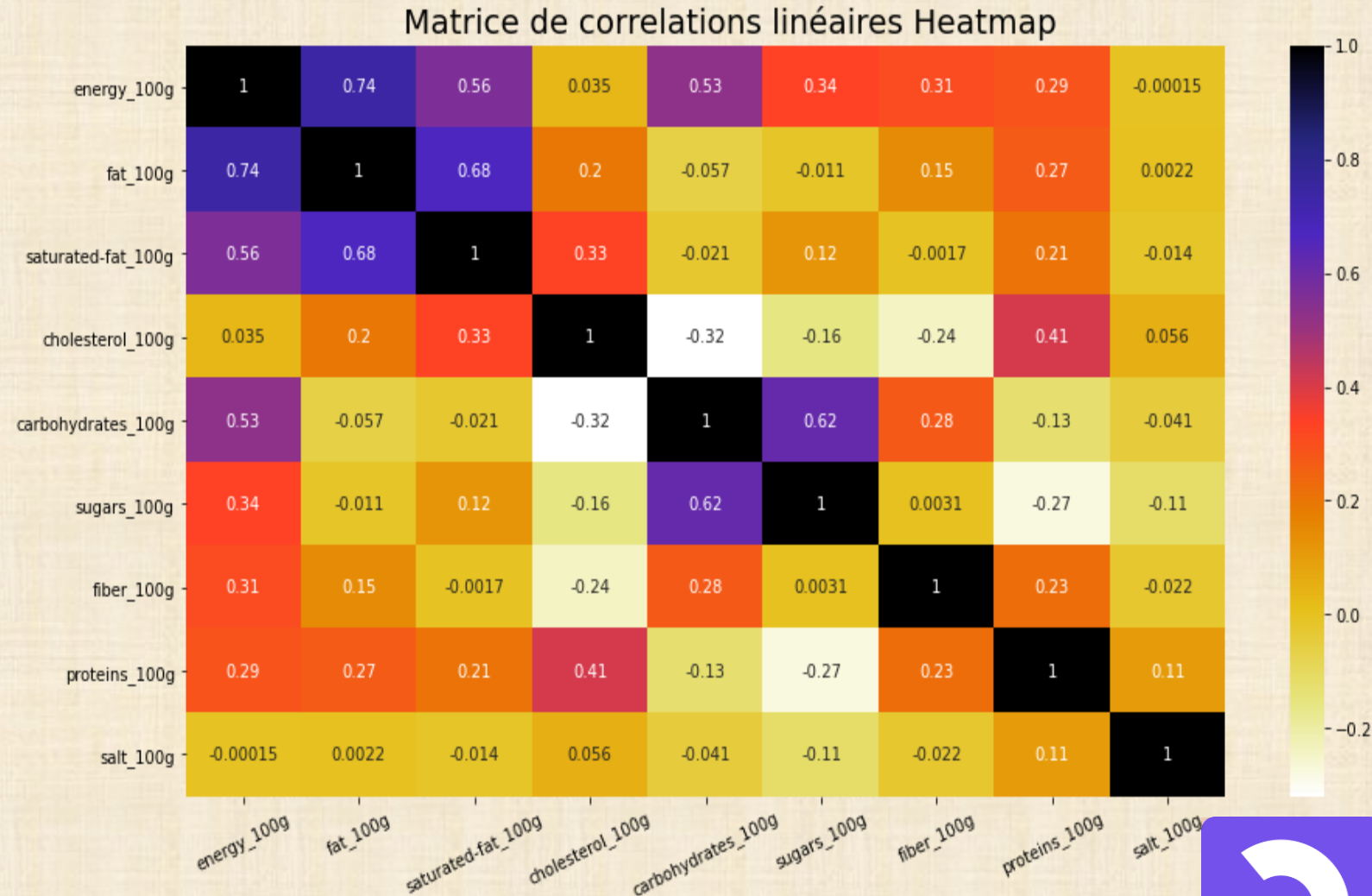


C. Analyse exploratoire

C.2. Analyse bivariée:

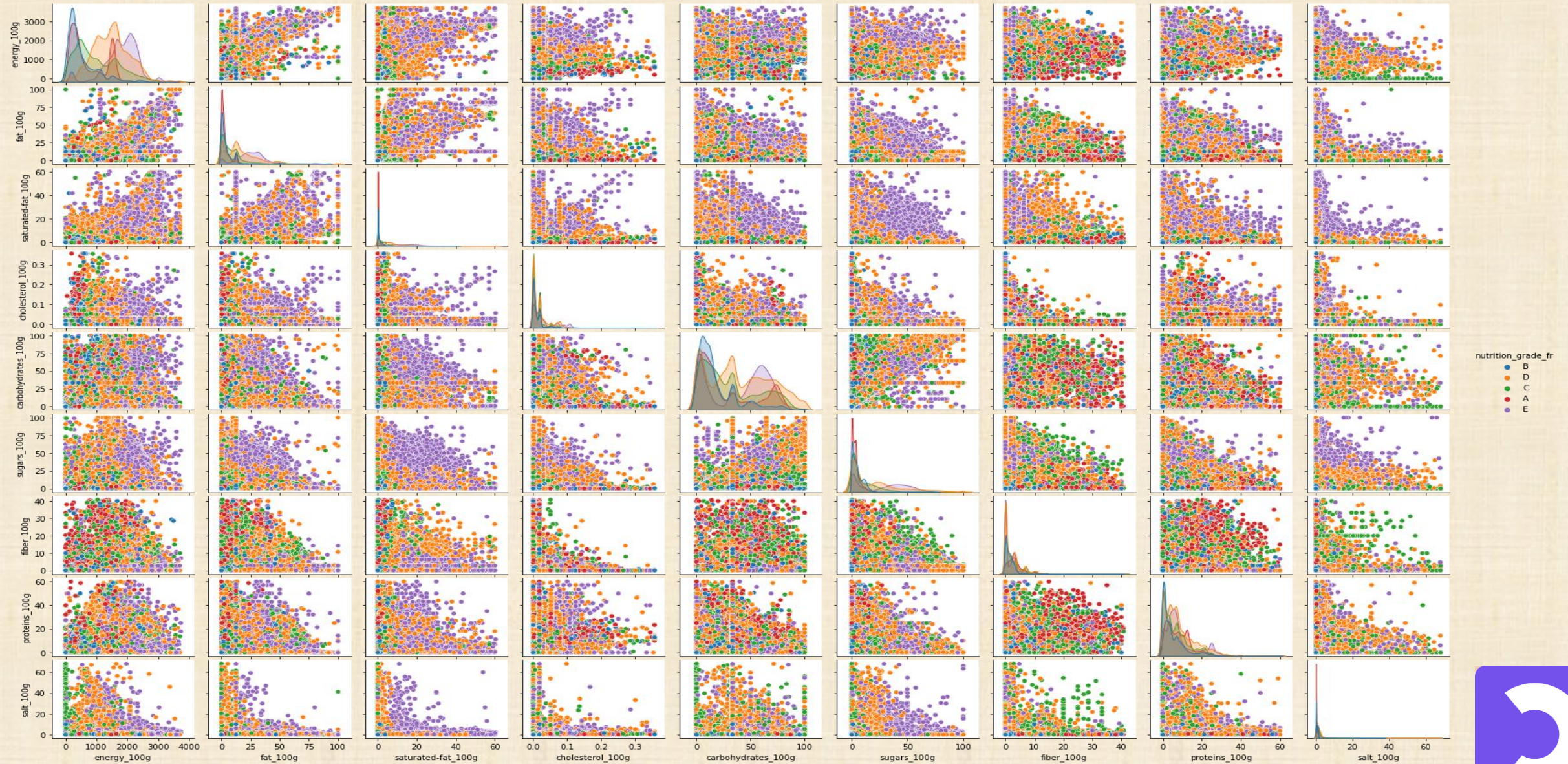
On constate la présence de corrélations entre les variables suivantes:

- energy_100g et fat_100g = 0.74
- energy_100g et nutrition-score = 0.62
- fat_100g et saturated_fat_100g = 0.68
- saturated_fat_100g et nutrition-score = 0.63
- sugars_100g et carbohydrates_100g = 0.62
- energy_100g et carbohydrates_100g = 0.56
- sugars_100g et nutrition-score = 0.41



C. Analyse exploratoire

l'ensemble des variables en fonction de nutri-grade



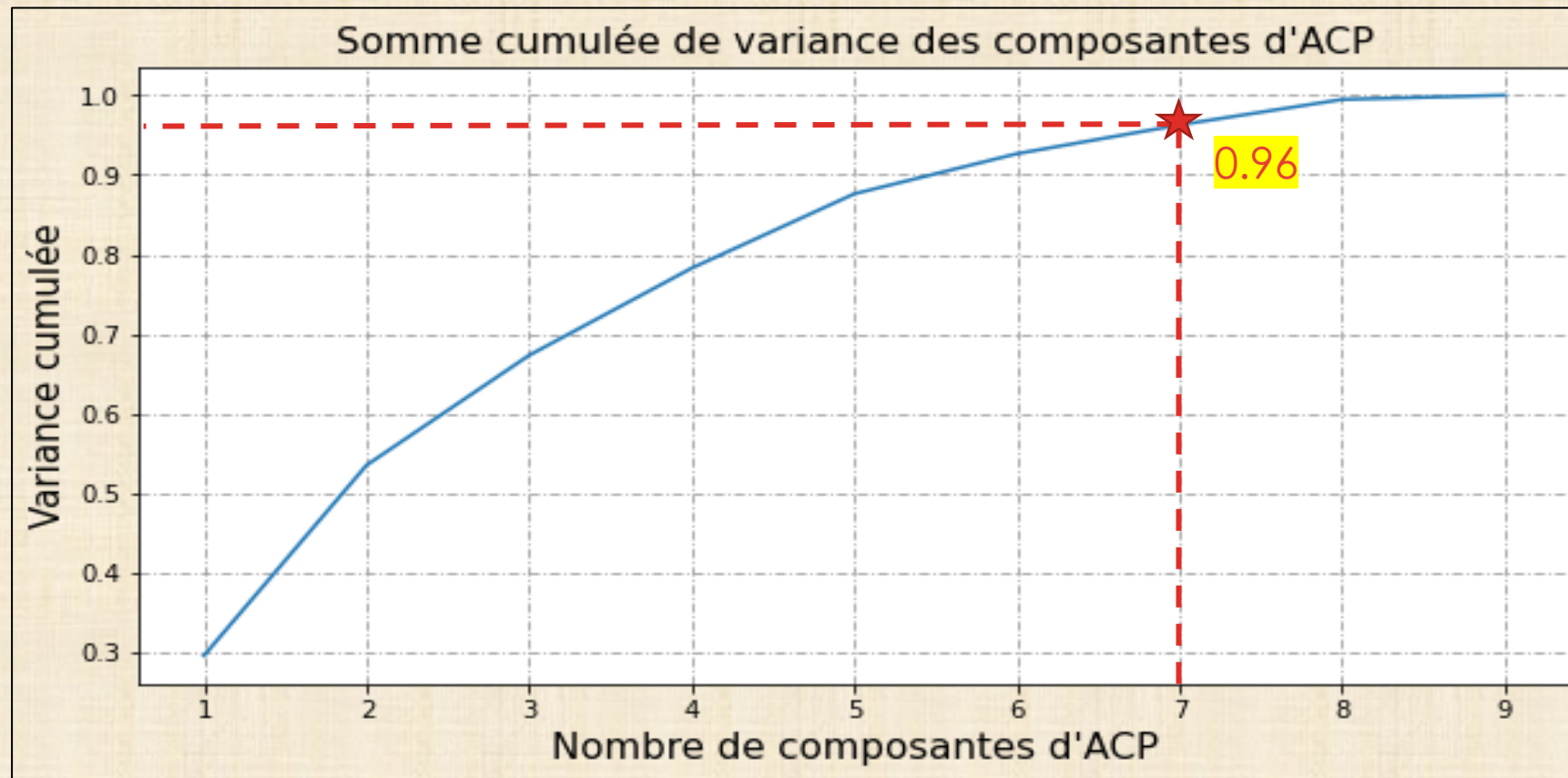
C.2. Analyse bivariable:



C. Analyse exploratoire

C.3.1 Analyse multivariée: Analyse par composantes principales ACP

Selon le cumule de variance on constate qu'on peut réduire les dimension de notre dataset à 7 dimensions



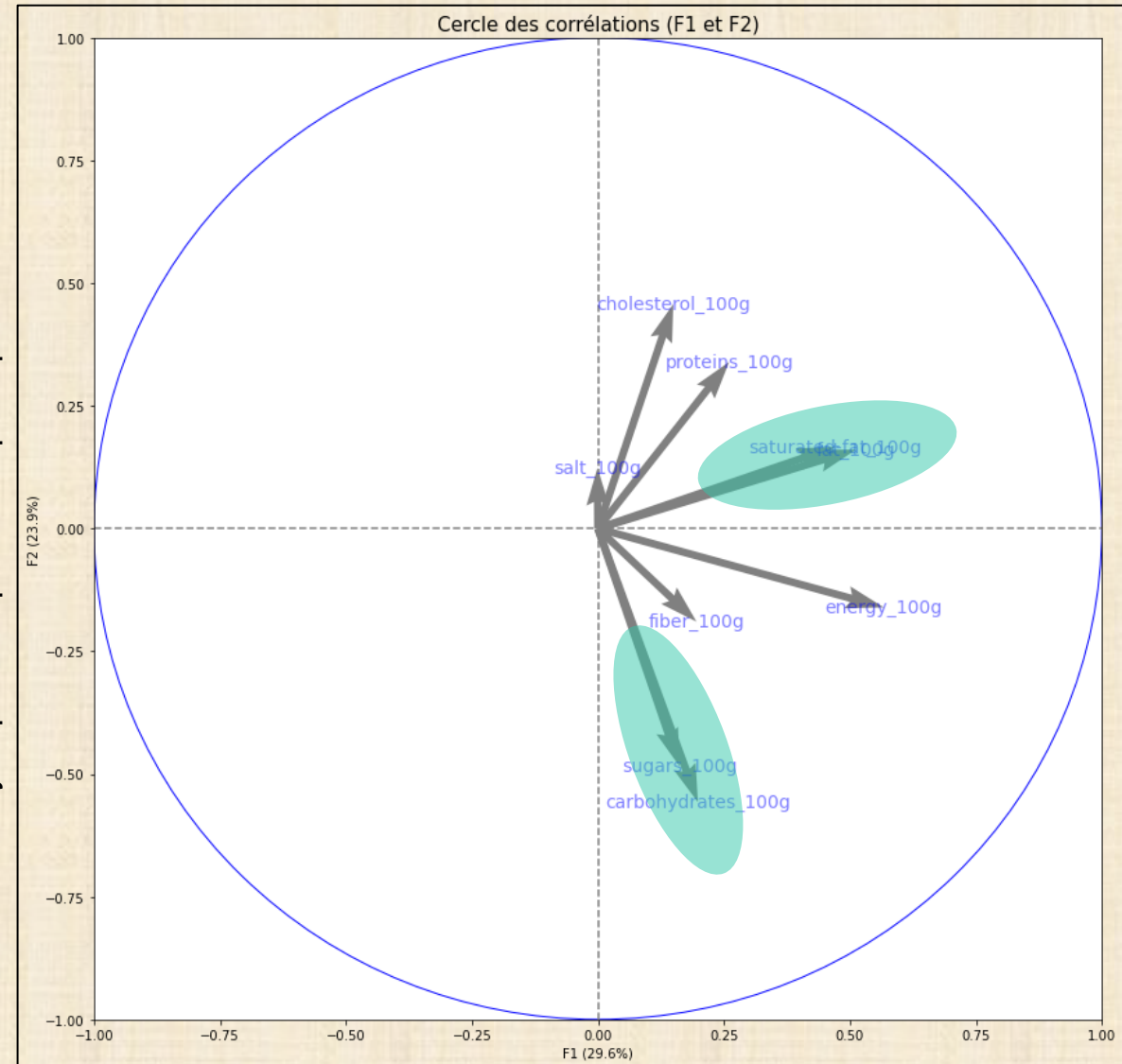
C. Analyse exploratoire

C.3.1 Analyse multivariée: ACP

On constate la présence de corrélations entre les variables suivantes:

- energy_100g et fat_100g = 0.74
- energy_100g et nutrition-score = 0.62
- fat_100g et saturated_fat_100g = 0.68
- saturated_fat_100g et nutrition-score = 0.63
- sugars_100g et carbohydrates_100g = 0.62
- energy_100g et carbohydrates_100g = 0.56
- sugars_100g et nutrition-score = 0.41

Analyse par composantes principales ACP



C. Analyse exploratoire

C.3. Analyse multivariée: Analyse de la variance (ANOVA) indépendance des variables

- Hypothèses stochastiques:

1. les échantillons sont issus d'une population normale (gaussienne): on parle de test paramétrique
2. les variances conditionnelles (variances dans chaque sous-population) sont identiques : homoscédasticité
3. les sous-échantillons sont indépendants

* En toute rigueur, on devrait vérifier les deux premières hypothèses pour que l'analyse ANOVA soit valide

1. Test de normalité avec Shapiro-Wilk test:

Selon le test de normalité avec Shapiro-Wilk la distribution n'est pas normale, dans ce cas l'analyse d'ANOVA n'est pas valide, mais on va vérifier la deuxième condition.

```
*****  
w = 0.9589363932609558 / p = 0.0  
Hypothèse nulle est rejetée : energy_100g n'est pas de distribution normale  
*****
```

2. Test d'homogénéité avec Levene's test:

Selon le test de Levene pvalue < 0.05 donc l'homogénéité n'est pas la même pour les nuri-grade sur la variable energy_100g.

Cela nous confirme encore une fois que le test de ANOVA n'est pas valide car les deux conditions la normalité ainsi que l'homogénéité ne sont pas valides

```
LeveneResult(statistic=2165.7907561155494, pvalue=0.0)
```



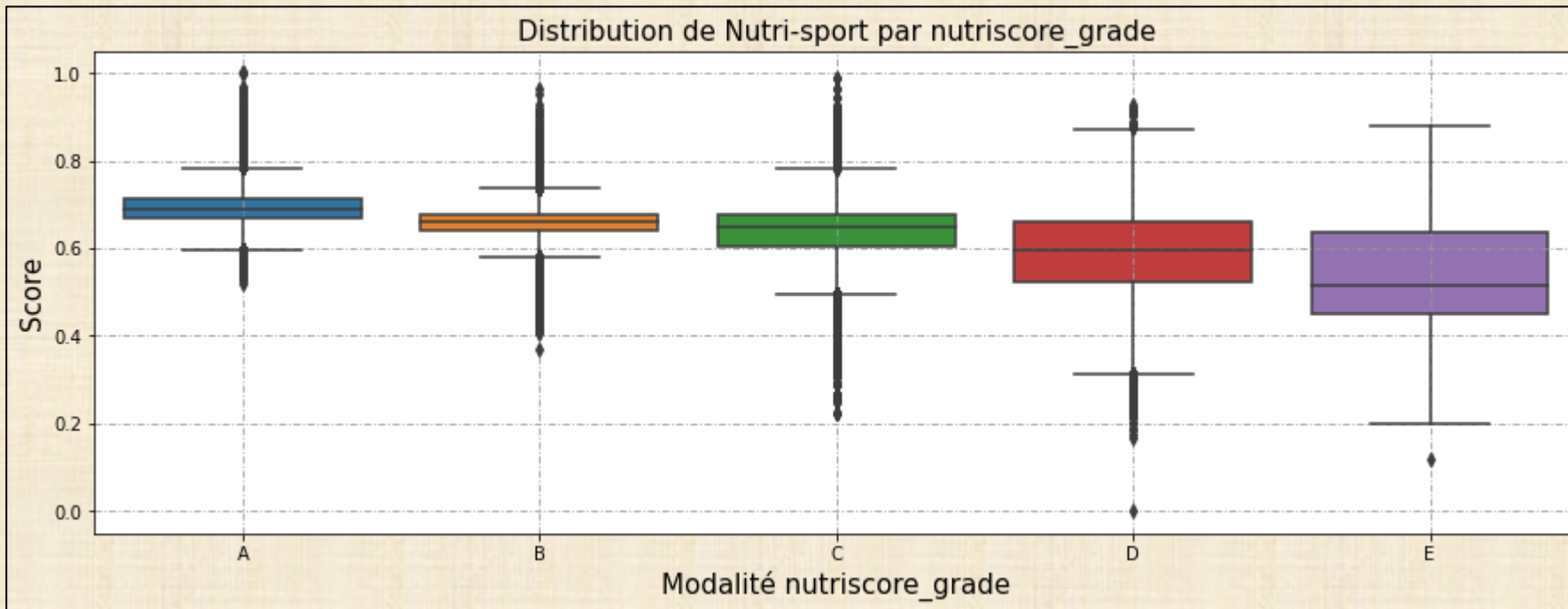
D. Conclusion

D. Faisabilité de l'Appli:

- Calcul de Nutri-Sport:

Le choix des variables potentielles → Calcul de Score → Normalisation de résultats

Sur le Boxplot, on constate que le score calculé est bien corrélé avec le nutri_grade, plus le produit est protéiné et plus il est bien classé, plus de gras et plus est moins classé.



```
nutri_score_fnl.insert(19, 'Score', 0)
```

```
scr = { 'fat_100g':-3,  
        'cholesterol_100g':-1,  
        'sugars_100g':-5,  
        'fiber_100g':5,  
        'proteins_100g':5,  
        'salt_100g':1}
```

```
for index in nutri_score_fnl.index:  
    s = 0  
    for k in scr.keys():  
        if k in nutri_score_fnl:  
            s = s + nutri_score_fnl[k][index]*scr[k]  
        nutri_score_fnl.at[index, 'Score'] = s
```

```
nutri_score_fnl['Score'].max()
```

400

```
nutri_score_fnl['Score'].min()
```

-800



La faisabilité de l'Appli:

- Le dataset contient des données utiles pour répondre à la problématique(Appli)
- La traçabilité des sources de données
- Contient des données détaillées sur chaque produit

Aspect négatif:

- La répartition de NaN distincte sur l'ensemble des variables
- Le manque d'infos sur les catégories pnns1 et pnns1

