In [ ]:
```
# Import du jeu de donnée
import pandas as pd
import numpy as np
df = pd.read_csv('fr.openfoodfacts.org.products.csv',sep = '\t',low_memory=False
# On affiche les 5 premières lignes
print(df.head())
# On affiche les colonnes
df.describe()
```

```
           code                                               url  \
0  0000000003087  http://world-fr.openfoodfacts.org/produit/0000...
1  0000000004530  http://world-fr.openfoodfacts.org/produit/0000...
2  0000000004559  http://world-fr.openfoodfacts.org/produit/0000...
3  0000000016087  http://world-fr.openfoodfacts.org/produit/0000...
4  0000000016094  http://world-fr.openfoodfacts.org/produit/0000...

                     creator   created_t      created_datetime  \
0  openfoodfacts-contributors  1474103866  2016-09-17T09:17:46Z
1             usda-ndb-import  1489069957  2017-03-09T14:32:37Z
2             usda-ndb-import  1489069957  2017-03-09T14:32:37Z
3             usda-ndb-import  1489055731  2017-03-09T10:35:31Z
4             usda-ndb-import  1489055653  2017-03-09T10:34:13Z

  last_modified_t last_modified_datetime                  product_name  \
0      1474103893   2016-09-17T09:18:13Z              Farine de blé noir
1      1489069957   2017-03-09T14:32:37Z  Banana Chips Sweetened (Whole)
2      1489069957   2017-03-09T14:32:37Z                         Peanuts
3      1489055731   2017-03-09T10:35:31Z          Organic Salted Nut Mix
4      1489055653   2017-03-09T10:34:13Z                 Organic Polenta

  generic_name quantity  ... ph_100g fruits-vegetables-nuts_100g  \
0          NaN      1kg  ...     NaN                         NaN
1          NaN      NaN  ...     NaN                         NaN
2          NaN      NaN  ...     NaN                         NaN
3          NaN      NaN  ...     NaN                         NaN
4          NaN      NaN  ...     NaN                         NaN

  collagen-meat-protein-ratio_100g cocoa_100g chlorophyl_100g  \
0                              NaN        NaN             NaN
1                              NaN        NaN             NaN
2                              NaN        NaN             NaN
3                              NaN        NaN             NaN
4                              NaN        NaN             NaN

  carbon-footprint_100g nutrition-score-fr_100g nutrition-score-uk_100g  \
0                   NaN                     NaN                     NaN
1                   NaN                    14.0                    14.0
2                   NaN                     0.0                     0.0
3                   NaN                    12.0                    12.0
4                   NaN                     NaN                     NaN

  glycemic-index_100g water-hardness_100g
0                 NaN                 NaN
1                 NaN                 NaN
2                 NaN                 NaN
3                 NaN                 NaN
4                 NaN                 NaN

[5 rows x 162 columns]
```

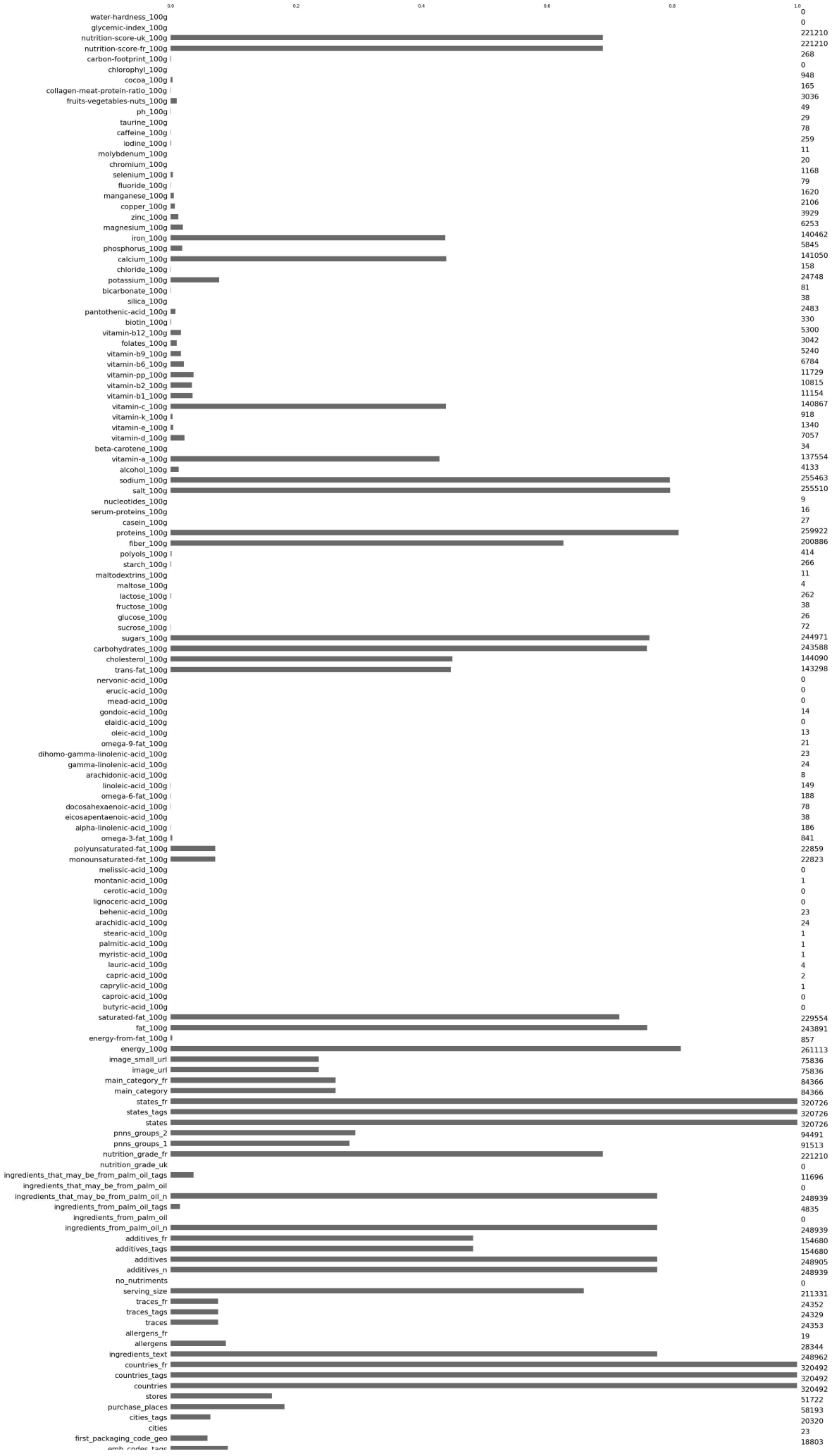
Out[ ]:

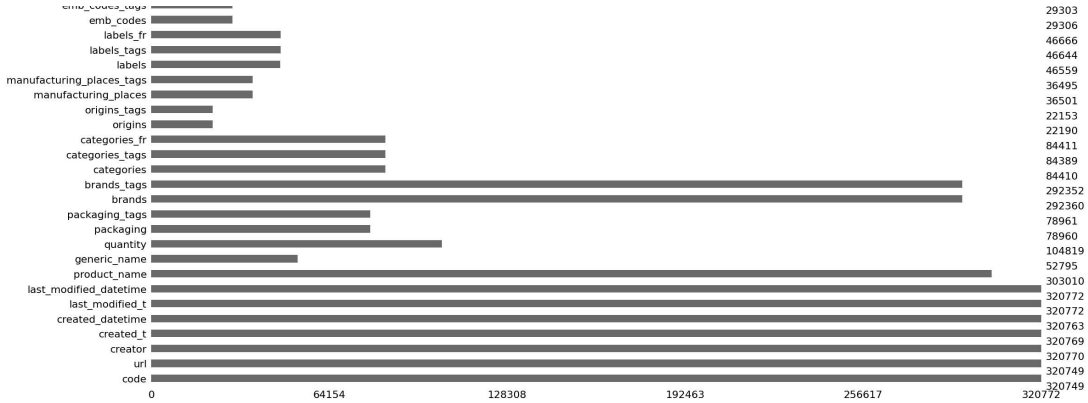
	no_nutriments	additives_n	ingredients_from_palm_oil_n	ingredients_from_palr
count	0.0	248939.000000	248939.000000	
mean	NaN	1.936024	0.019659	
std	NaN	2.502019	0.140524	
min	NaN	0.000000	0.000000	
25%	NaN	0.000000	0.000000	
50%	NaN	1.000000	0.000000	
75%	NaN	3.000000	0.000000	
max	NaN	31.000000	2.000000	

8 rows × 106 columns

In [ ]:
```
# Pas très parlant on regarde les msno
import missingno as msno
msno.bar(df)
```

Out[ ]: <Axes: >

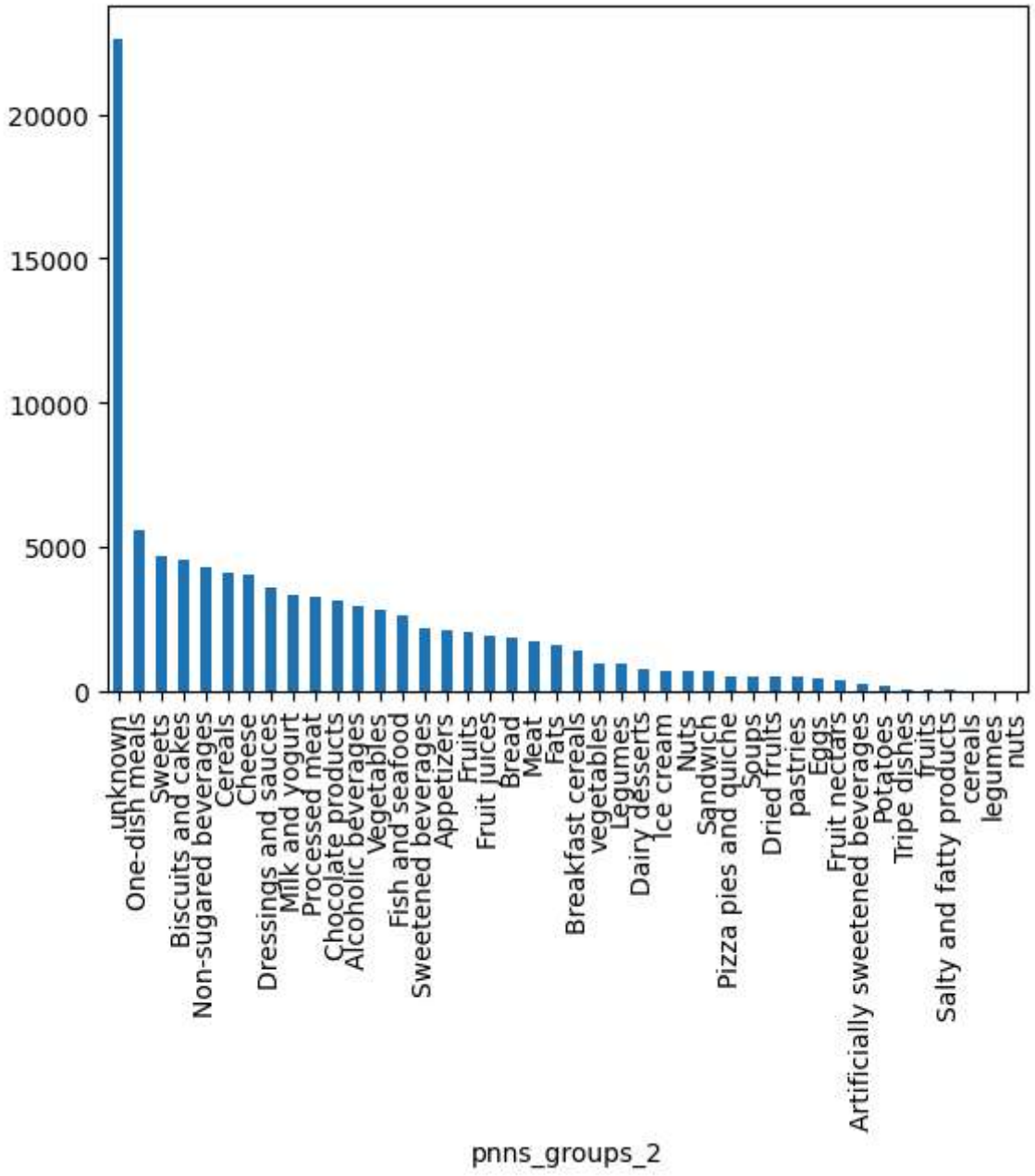

0.0
0.2
0.4
0.6
0.8
1.0
water-hardness_100g 0
glycemic-index_100g 0
nutrition-score-uk_100g 221210
nutrition-score-fr_100g 221210
carbon-footprint_100g 268
chlorophyl_100g 0
cocoa_100g 948
collagen-meat-protein-ratio_100g 165
fruits-vegetables-nuts_100g 3036
ph_100g 49
taurine_100g 29
caffeine_100g 78
iodine_100g 259
molybdenum_100g 11
chromium_100g 20
selenium_100g 1168
fluoride_100g 79
manganese_100g 1620
copper_100g 2106
zinc_100g 3929
magnesium_100g 6253
iron_100g 140462
phosphorus_100g 5845
calcium_100g 141050
chloride_100g 158
potassium_100g 24748
bicarbonate_100g 81
silica_100g 38
pantothenic-acid_100g 2483
biotin_100g 330
vitamin-b12_100g 5300
folates_100g 3042
vitamin-b9_100g 5240
vitamin-b6_100g 6784
vitamin-pp_100g 11729
vitamin-b2_100g 10815
vitamin-b1_100g 11154
vitamin-c_100g 140867
vitamin-k_100g 918
vitamin-e_100g 1340
vitamin-d_100g 7057
beta-carotene_100g 34
vitamin-a_100g 137554
alcohol_100g 4133
sodium_100g 255463
salt_100g 255510
nucleotides_100g 9
serum-proteins_100g 16
casein_100g 27
proteins_100g 259922
fiber_100g 200886
polyols_100g 414
starch_100g 266
maltodextrins_100g 11
maltose_100g 4
lactose_100g 262
fructose_100g 38
glucose_100g 26
sucrose_100g 72
sugars_100g 244971
carbohydrates_100g 243588
cholesterol_100g 144090
trans-fat_100g 143298
nervonic-acid_100g 0
erucic-acid_100g 0
mead-acid_100g 0
gondoic-acid_100g 14
elaidic-acid_100g 0
oleic-acid_100g 13
omega-9-fat_100g 21
dihomo-gamma-linolenic-acid_100g 23
gamma-linolenic-acid_100g 24
arachidonic-acid_100g 8
linoleic-acid_100g 149
omega-6-fat_100g 188
docosahexaenoic-acid_100g 78
eicosapentaenoic-acid_100g 38
alpha-linolenic-acid_100g 186
omega-3-fat_100g 841
polyunsaturated-fat_100g 22859
monounsaturated-fat_100g 22823
melissic-acid_100g 0
montanic-acid_100g 1
cerotic-acid_100g 0
lignoceric-acid_100g 0
behenic-acid_100g 23
arachidic-acid_100g 24
stearic-acid_100g 1
palmitic-acid_100g 1
myristic-acid_100g 1
lauric-acid_100g 4
capric-acid_100g 2
caprylic-acid_100g 1
caproic-acid_100g 0
butyric-acid_100g 0
saturated-fat_100g 229554
fat_100g 243891
energy-from-fat_100g 857
energy_100g 261113
image_small_url 75836
image_url 75836
main_category_fr 84366
main_category 84366
states_fr 320726
states_tags 320726
states 320726
pnns_groups_2 94491
pnns_groups_1 91513
nutrition_grade_fr 221210
nutrition_grade_uk 0
ingredients_that_may_be_from_palm_oil_tags 11696
ingredients_that_may_be_from_palm_oil 0
ingredients_that_may_be_from_palm_oil_n 248939
ingredients_from_palm_oil_tags 4835
ingredients_from_palm_oil 0
ingredients_from_palm_oil_n 248939
additives_fr 154680
additives_tags 154680
additives 248905
additives_n 248939
no_nutriments 0
serving_size 211331
traces_fr 24352
traces_tags 24329
traces 24353
allergens_fr 19
allergens 28344
ingredients_text 248962
countries_fr 320492
countries_tags 320492
countries 320492
stores 51722
purchase_places 58193
cities_tags 20320
cities 23
first_packaging_code_geo 18803



In [ ]:
```
# On regarde les colonnes avec moins de 50% de valeurs manquantes
missing_values = df.isnull().sum()
missing_percentage = (missing_values / len(df)) * 100
columns_with_fewer_missing_values = missing_percentage[missing_percentage <= 50]
print("Colonnes avec moins de 50% de valeurs manquantes:", columns_with_fewer_mi
```

```
Colonnes avec moins de 50% de valeurs manquantes: ['code', 'url', 'creator', 'cre
ated_t', 'created_datetime', 'last_modified_t', 'last_modified_datetime', 'produc
t_name', 'brands', 'brands_tags', 'countries', 'countries_tags', 'countries_fr',
'ingredients_text', 'serving_size', 'additives_n', 'additives', 'ingredients_from
_palm_oil_n', 'ingredients_that_may_be_from_palm_oil_n', 'nutrition_grade_fr', 's
tates', 'states_tags', 'states_fr', 'energy_100g', 'fat_100g', 'saturated-fat_100
g', 'carbohydrates_100g', 'sugars_100g', 'fiber_100g', 'proteins_100g', 'salt_100
g', 'sodium_100g', 'nutrition-score-fr_100g', 'nutrition-score-uk_100g']
```

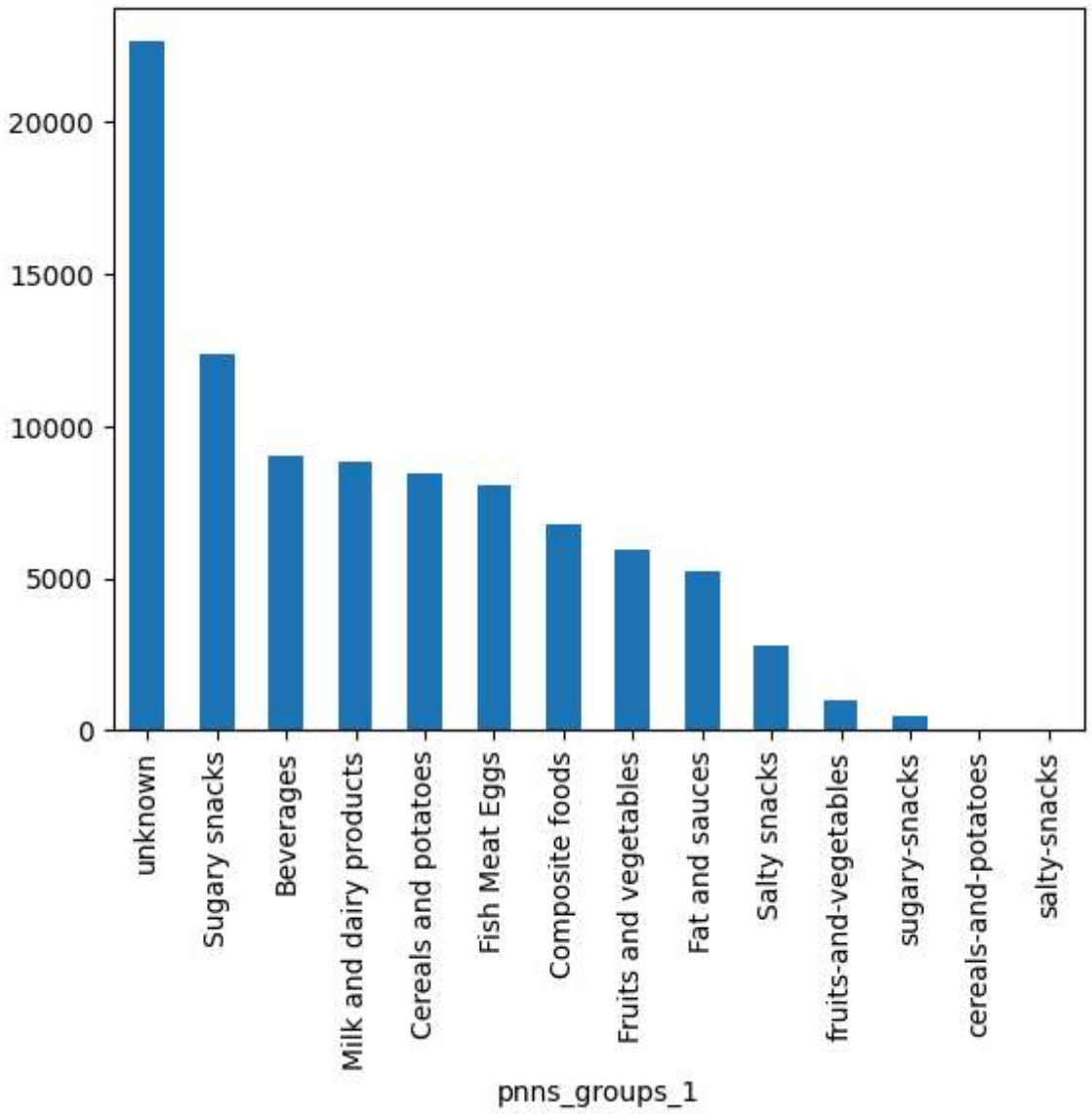
In [ ]:
```
# On s'intéresse à des valeurs inférieur à 50% que l'on pourrait reconstitué à p
# value count en graphique
import matplotlib.pyplot as plt
df['pnns_groups_2'].value_counts().plot(kind='bar')
plt.show()
# On regarde les valeurs uniques
print(df['pnns_groups_2'].unique())
```




```
[nan 'unknown' 'Vegetables' 'Biscuits and cakes' 'Bread' 'Legumes'
 'Pizza pies and quiche' 'Meat' 'Sweets' 'Non-sugared beverages'
 'Sweetened beverages' 'Dressings and sauces' 'One-dish meals'
 'vegetables' 'Soups' 'Chocolate products' 'Alcoholic beverages' 'Fruits'
 'Sandwich' 'Cereals' 'Milk and yogurt' 'Fats' 'Cheese' 'Appetizers'
 'Nuts' 'Breakfast cereals' 'Artificially sweetened beverages'
 'Fruit juices' 'Eggs' 'Fish and seafood' 'Dried fruits' 'Ice cream'
 'Processed meat' 'Potatoes' 'Dairy desserts' 'Fruit nectars' 'pastries'
 'Tripe dishes' 'fruits' 'Salty and fatty products' 'cereals' 'legumes'
 'nuts']
```

In [ ]:

In [ ]:
```
# même chose pour pnns_groups_1
df['pnns_groups_1'].value_counts().plot(kind='bar')
plt.show()
print(df['pnns_groups_1'].unique())
```



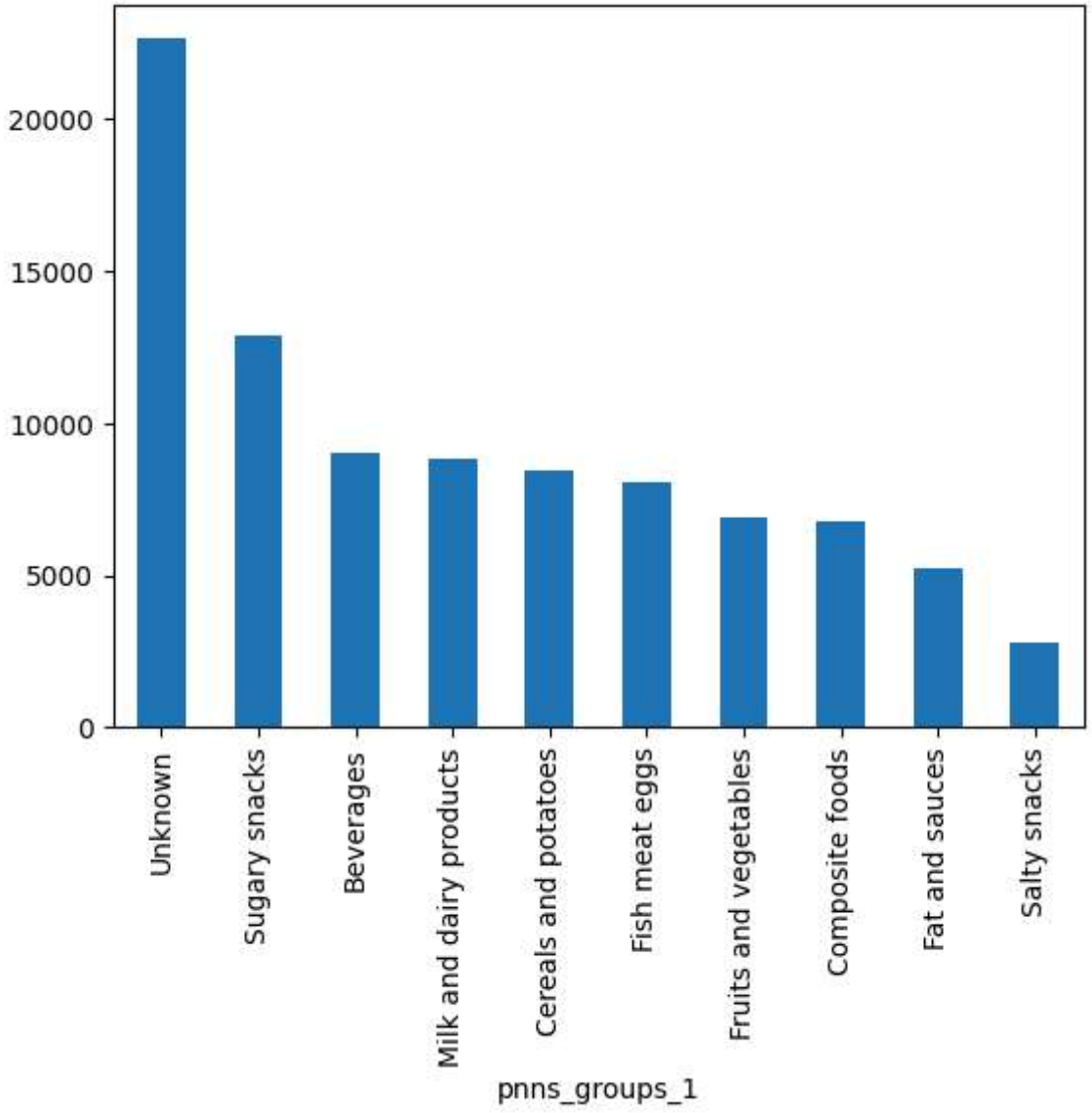

```
[nan 'unknown' 'Fruits and vegetables' 'Sugary snacks'
 'Cereals and potatoes' 'Composite foods' 'Fish Meat Eggs' 'Beverages'
 'Fat and sauces' 'fruits-and-vegetables' 'Milk and dairy products'
 'Salty snacks' 'sugary-snacks' 'cereals-and-potatoes' 'salty-snacks']
```

Il est peut être un peu moins fourni mais servirait de base plus facile pour tenter de reconstituer notre jeu de donnée. A voir avec la documentation en ligne si c'est faisable aisément.

Certaines catégories de pnns_group_1 sont ortographiés différemments, à retravailler

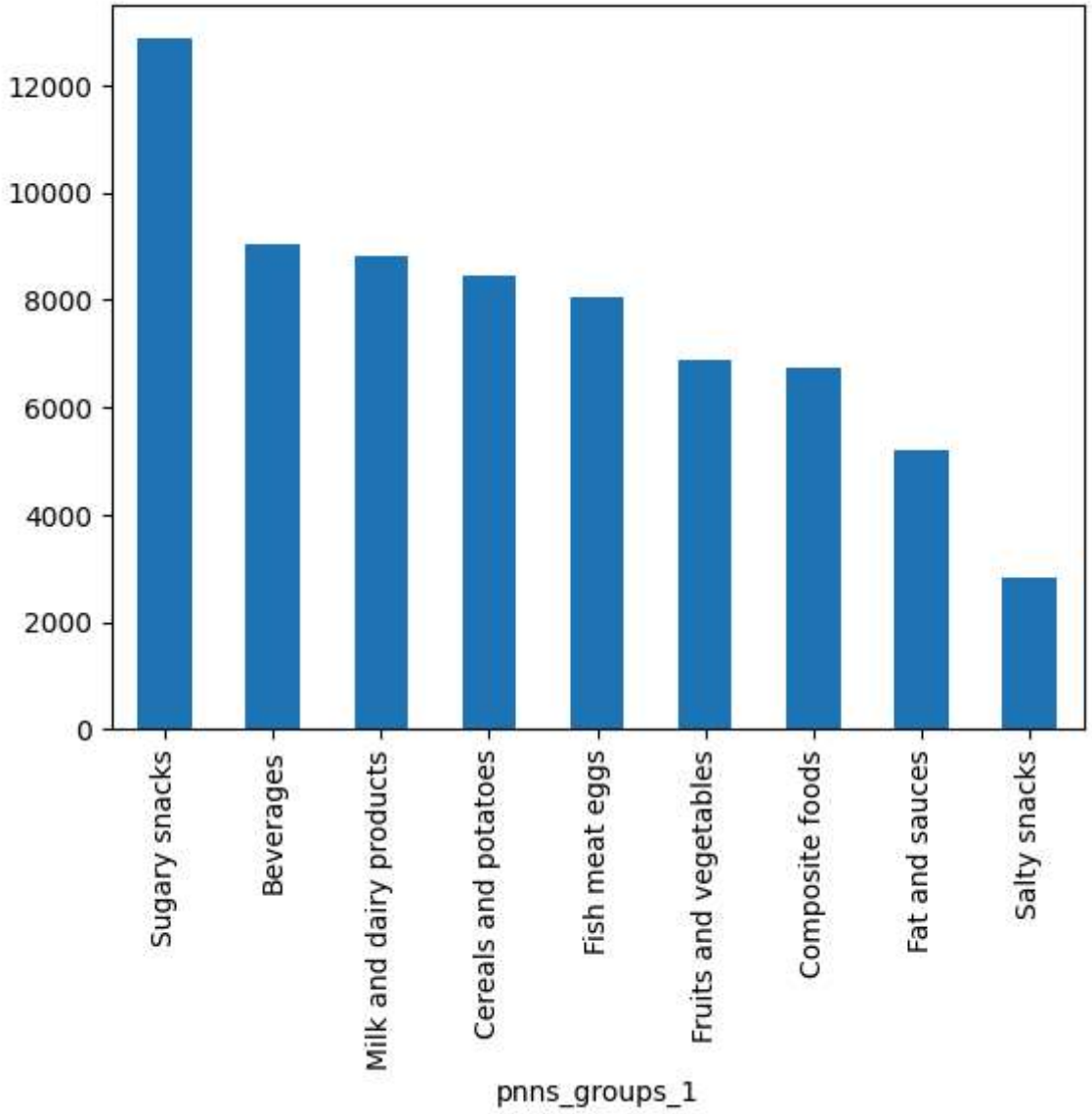
In [ ]:
```
# Mettre tout en capitalize
df['pnns_groups_1'] = df['pnns_groups_1'].str.capitalize()
# Enlever les tirets
df['pnns_groups_1'] = df['pnns_groups_1'].str.replace('-', ' ')
```

In [ ]:
```
# Vérification
df['pnns_groups_1'].value_counts().plot(kind='bar')
plt.show()
print(df['pnns_groups_1'].unique())
```




```
[nan 'Unknown' 'Fruits and vegetables' 'Sugary snacks'
 'Cereals and potatoes' 'Composite foods' 'Fish meat eggs' 'Beverages'
 'Fat and sauces' 'Milk and dairy products' 'Salty snacks']
```

In [ ]:
```
# On remplace l'étiquette catégorie unknown par un vide et on essaiera de la rec
df['pnns_groups_1'] = df['pnns_groups_1'].replace('Unknown', np.nan)
# Vérification
df['pnns_groups_1'].value_counts().plot(kind='bar')
plt.show()
print(df['pnns_groups_1'].unique())
```

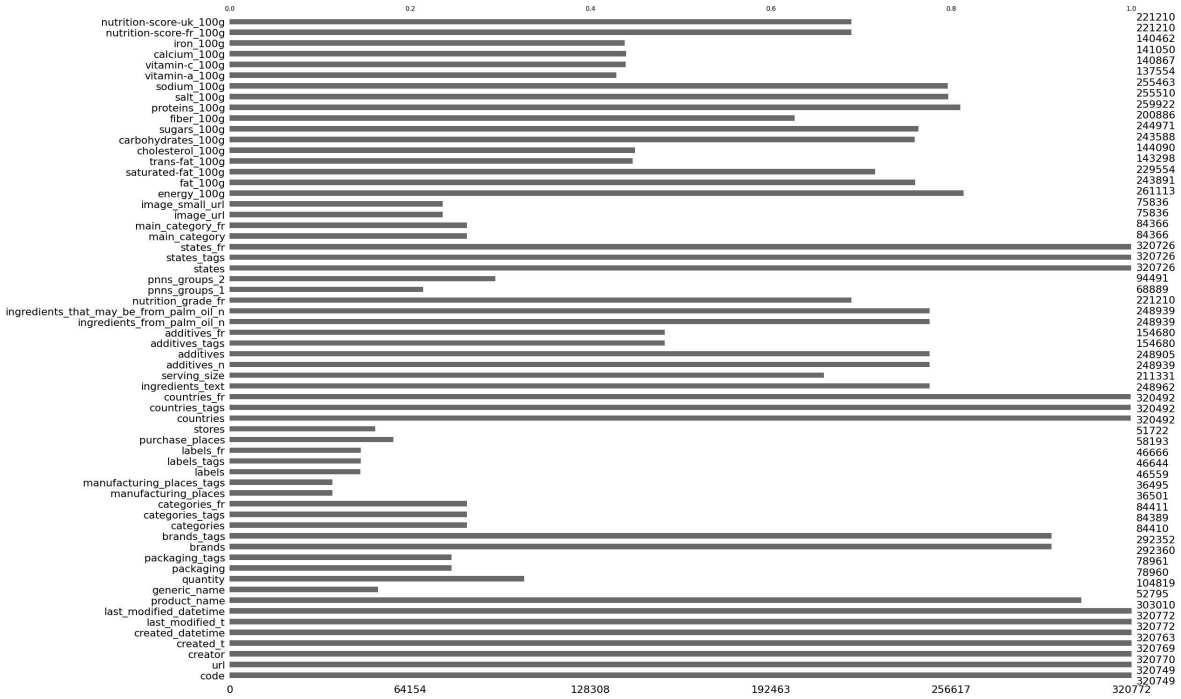



```
[nan 'Fruits and vegetables' 'Sugary snacks' 'Cereals and potatoes'
 'Composite foods' 'Fish meat eggs' 'Beverages' 'Fat and sauces'
 'Milk and dairy products' 'Salty snacks']
```

In [ ]:
```
# On enlève les colonnes avec moins de 10% de valeurs présentes
columns_to_keep = missing_percentage[missing_percentage <= 90].index.tolist()
df_under10_deleted = df[columns_to_keep]
print(df_under10_deleted.head())
msno.bar(df_under10_deleted)
```

```
            code                                               url  \
0  0000000003087  http://world-fr.openfoodfacts.org/produit/0000...
1  0000000004530  http://world-fr.openfoodfacts.org/produit/0000...
2  0000000004559  http://world-fr.openfoodfacts.org/produit/0000...
3  0000000016087  http://world-fr.openfoodfacts.org/produit/0000...
4  0000000016094  http://world-fr.openfoodfacts.org/produit/0000...

                     creator   created_t      created_datetime  \
0  openfoodfacts-contributors  1474103866  2016-09-17T09:17:46Z
1             usda-ndb-import  1489069957  2017-03-09T14:32:37Z
2             usda-ndb-import  1489069957  2017-03-09T14:32:37Z
3             usda-ndb-import  1489055731  2017-03-09T10:35:31Z
4             usda-ndb-import  1489055653  2017-03-09T10:34:13Z

  last_modified_t last_modified_datetime                   product_name  \
0      1474103893   2016-09-17T09:18:13Z               Farine de blé noir
1      1489069957   2017-03-09T14:32:37Z  Banana Chips Sweetened (Whole)
2      1489069957   2017-03-09T14:32:37Z                         Peanuts
3      1489055731   2017-03-09T10:35:31Z          Organic Salted Nut Mix
4      1489055653   2017-03-09T10:34:13Z                 Organic Polenta

  generic_name quantity  ... fiber_100g proteins_100g salt_100g sodium_100g  \
0          NaN      1kg  ...        NaN           NaN       NaN         NaN
1          NaN      NaN  ...        3.6          3.57   0.00000       0.000
2          NaN      NaN  ...        7.1         17.86   0.63500       0.250
3          NaN      NaN  ...        7.1         17.86   1.22428       0.482
4          NaN      NaN  ...        5.7          8.57       NaN         NaN

  vitamin-a_100g vitamin-c_100g calcium_100g iron_100g  \
0            NaN            NaN          NaN       NaN
1            0.0         0.0214        0.000   0.00129
2            0.0         0.0000        0.071   0.00129
3            NaN            NaN        0.143   0.00514
4            NaN            NaN          NaN       NaN

  nutrition-score-fr_100g nutrition-score-uk_100g
0                     NaN                    NaN
1                    14.0                   14.0
2                     0.0                    0.0
3                    12.0                   12.0
4                     NaN                    NaN

[5 rows x 62 columns]
```

Out[ ]: <Axes: >

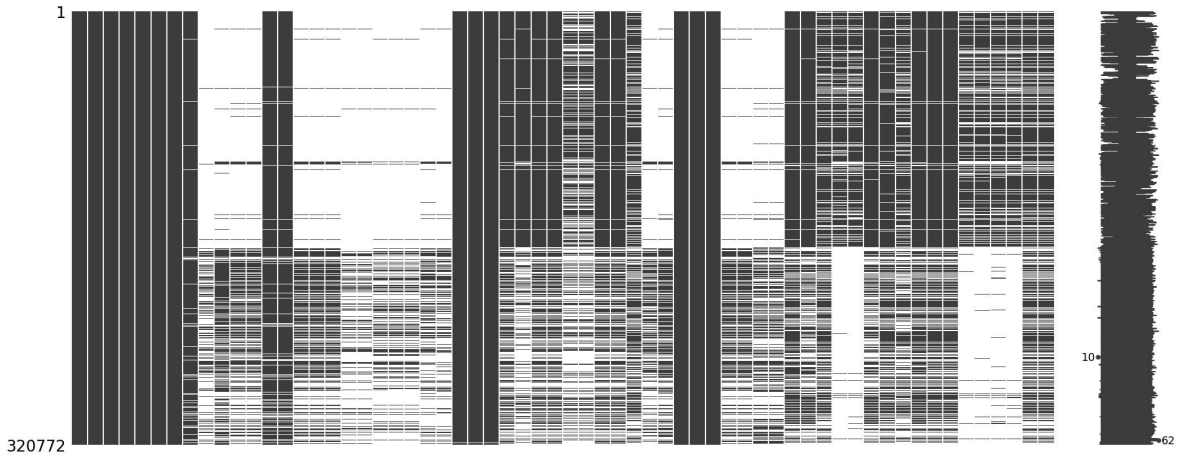


In [ ]:
```
# Montrer les valeurs uniques pour les catégories nutrition-score-fr_100g et nut
print(f' Catégorie nutrition-score-fr_100g : b/ {df["nutrition-score-fr_100g"].u
print(f' Catégorie nutrition_grade_fr : {df["nutrition_grade_fr"].unique()}')
```

```
 Catégorie nutrition-score-fr_100g : b/ [ nan  14.   0.  12.   7.  -6.  15.  11.
 9.  16.  19.  -1.   6.  26.
  13.  10.  17.  22.  23.  30.   8.  25.  -2.   1.   2.  18.  21.  20.
   3.   4.  24.  27.  28.  -3.  -4.  -7.  -5.   5.  29.  31.  -8.  -9.
  35.  33.  32.  34. -10.  40.  37.  38. -11.  36. -13. -12. -14. -15.]
 Catégorie nutrition_grade_fr : [nan 'd' 'b' 'c' 'a' 'e']
```

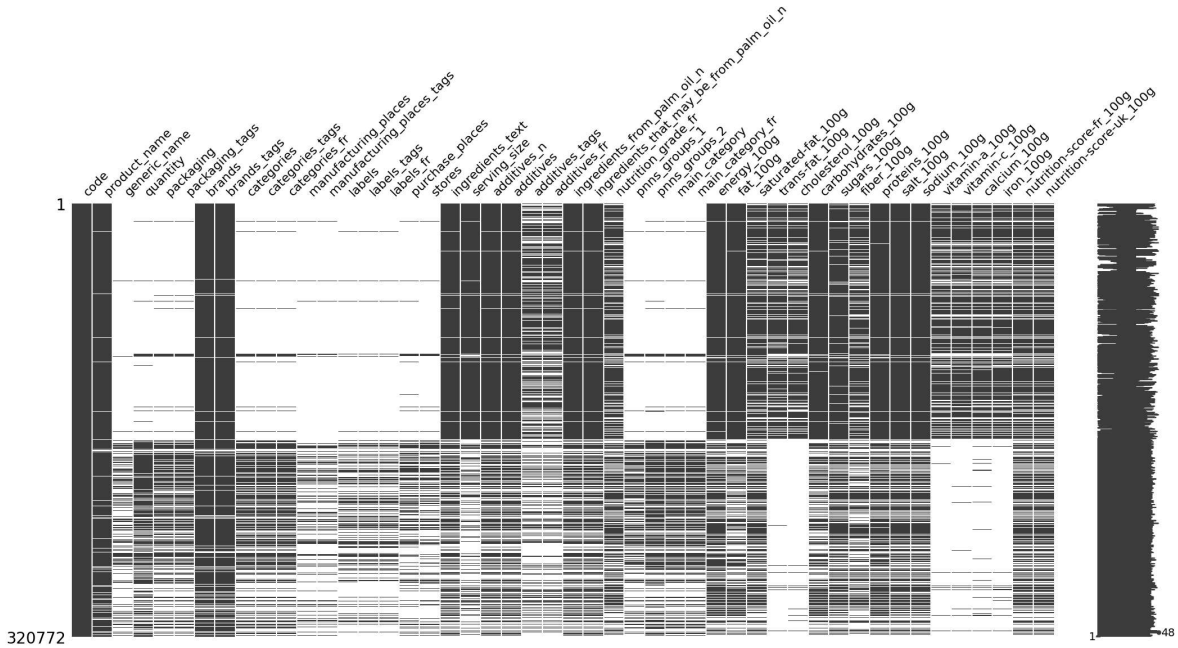
In [ ]:
```
msno.matrix(df_under10_deleted)
```

Out[ ]: <Axes: >



In [ ]:
```
# Réaliser un drop des colonnes inutiles
df_drop = df_under10_deleted.drop(columns=['url', 'creator', 'created_t', 'creat
msno.matrix(df_drop)
```

Out[ ]: <Axes: >



In [ ]:
```
# afficher les catégories unique de pnns group 1
print(df_drop['pnns_groups_1'].unique())
print(df_drop['pnns_groups_1'].nunique())
```

```
[nan 'Fruits and vegetables' 'Sugary snacks' 'Cereals and potatoes'
 'Composite foods' 'Fish meat eggs' 'Beverages' 'Fat and sauces'
 'Milk and dairy products' 'Salty snacks']
9
```

In [ ]:
```
# Valeurs unique pour categories et le nombre de valeurs uniques
print(df_drop['categories'].unique())
print(df_drop['categories'].nunique())
```

```
[nan 'Filet de bœuf' 'Légumes-feuilles' ...
 'Boissons,Boissons alcoolisées,Vins,Vins blancs,Roussette du Bugey'
 'pl:Szprot'
 "Viandes,Produits à tartiner,Charcuteries,Produits à tartiner salés,Rillettes,Ri
llettes de viande,Rillettes de viande blanche,Rillettes de volaille,Rillettes d'o
ie"]
36982
```

On va se focus sur le le pnns group 1 car ça a l'air d'être le plus simple !

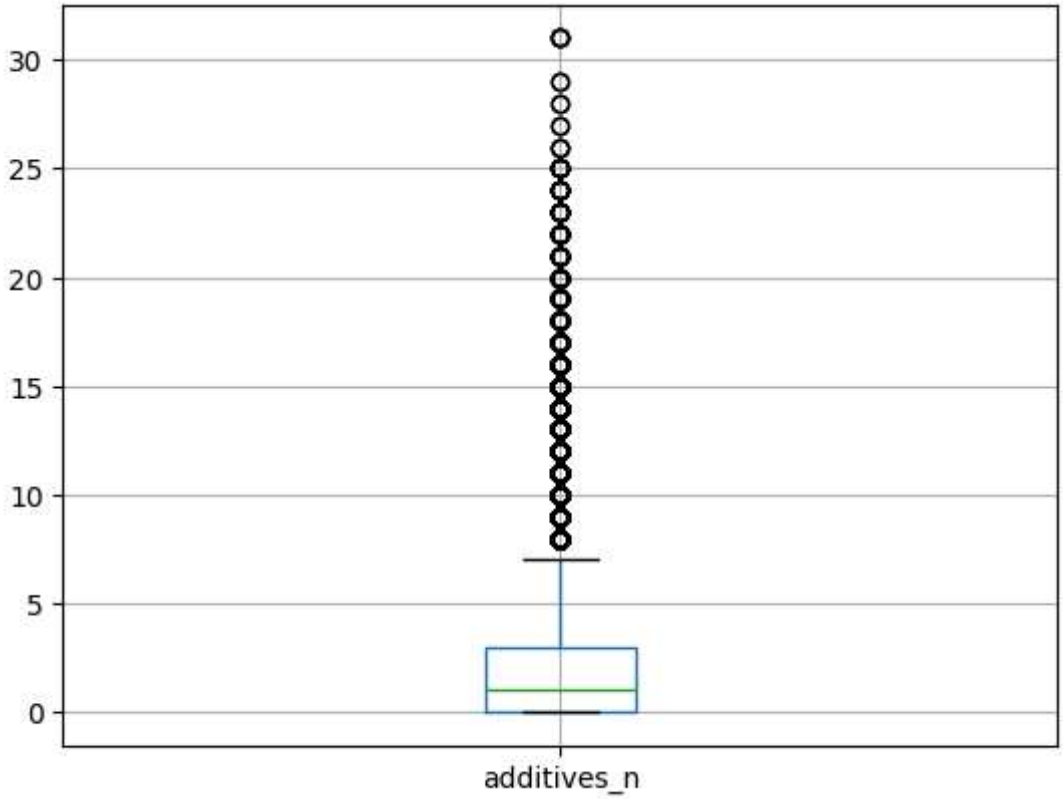
Vérification d'éventuels doublons

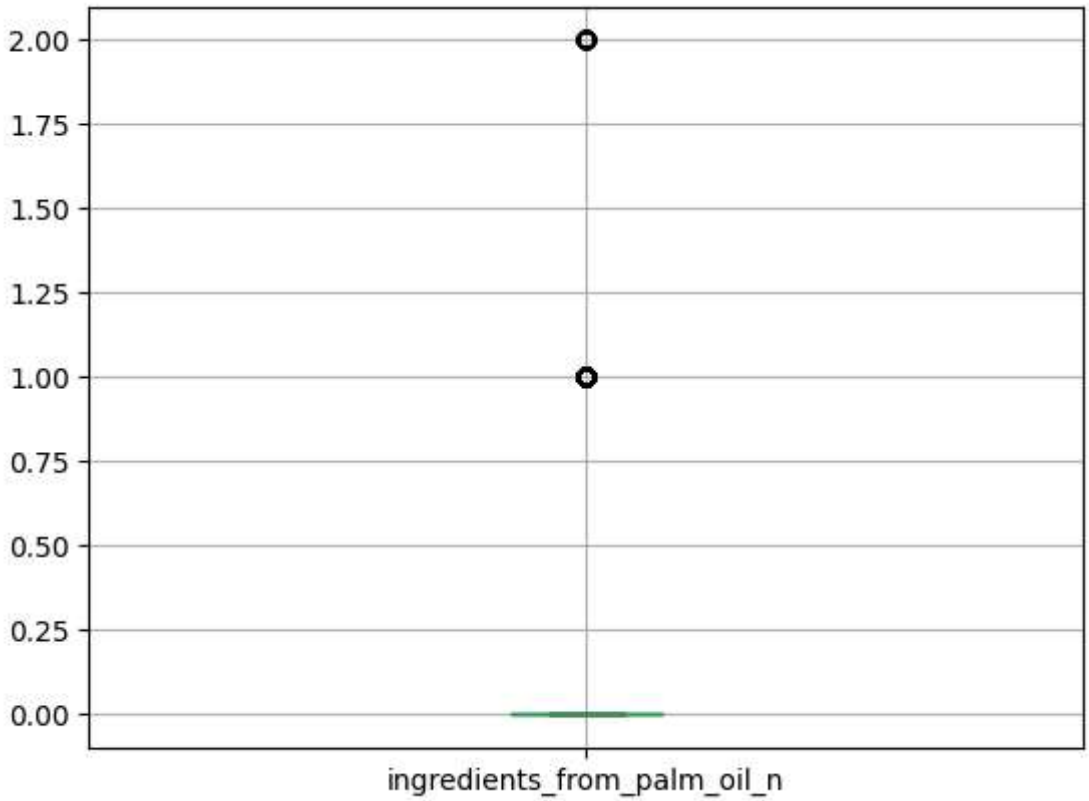
In [ ]:
```
# Recherche des doublons par code afin de les supprimer
sansdoublons = df_drop.drop_duplicates(subset='code', keep='first')
# On affiche le nombre de lignes supprimées
print(len(df_drop) - len(sansdoublons))
print(sansdoublons.shape)
```

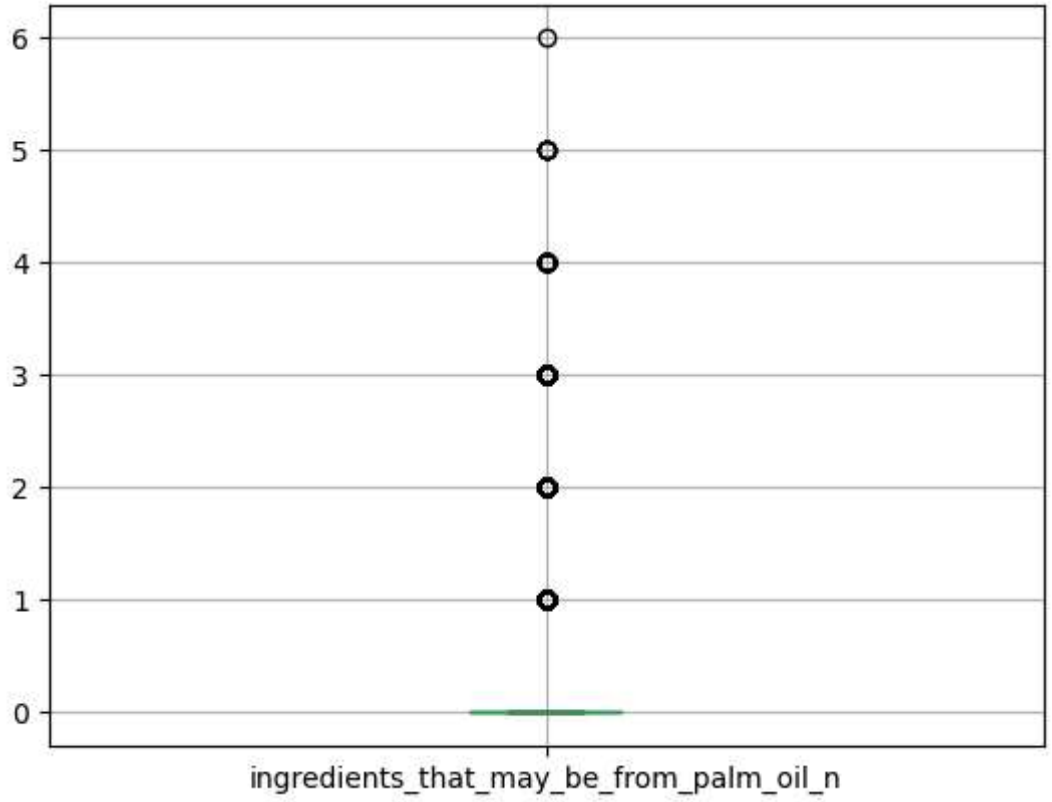
```
22
(320750, 48)
```

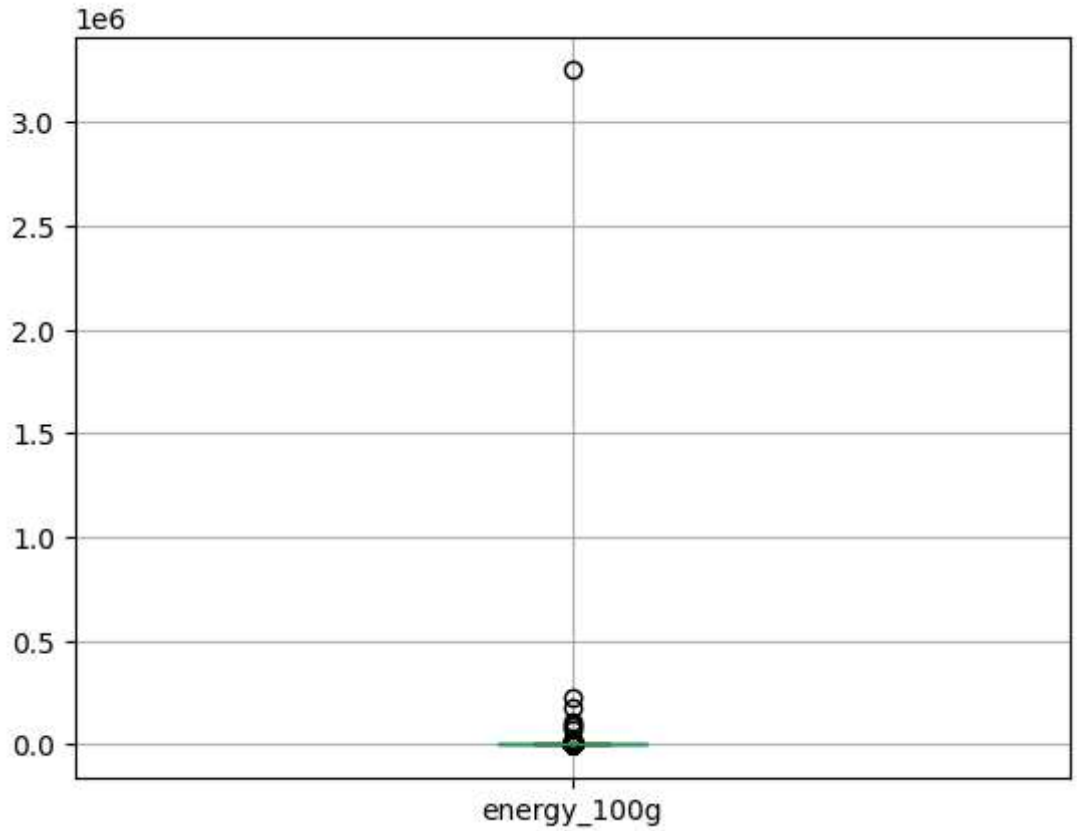
Vérification d'éventuels outliers

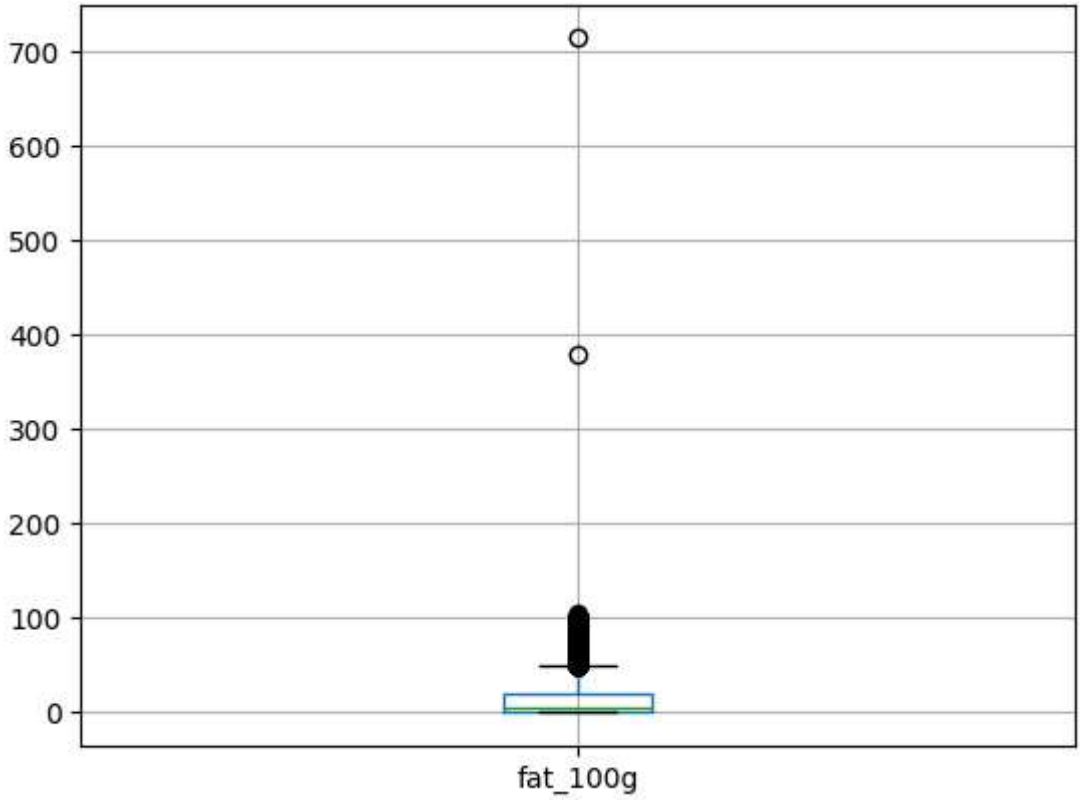
In [ ]:
```
# Boucle for pour faire un box plot de chaque variable numérique
for col in sansdoublons.select_dtypes(include=['float64']).columns:
    sansdoublons.boxplot(column=col)
    plt.show()
```

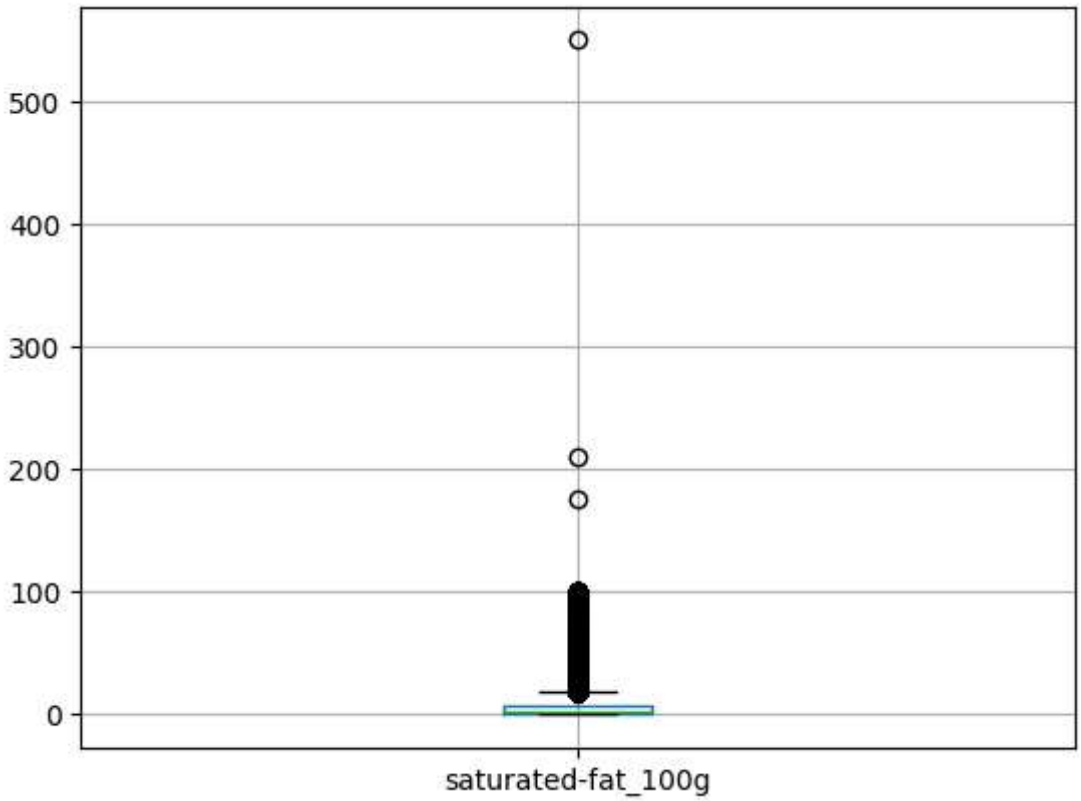

30
25
20
15
10
5
0
additives_n

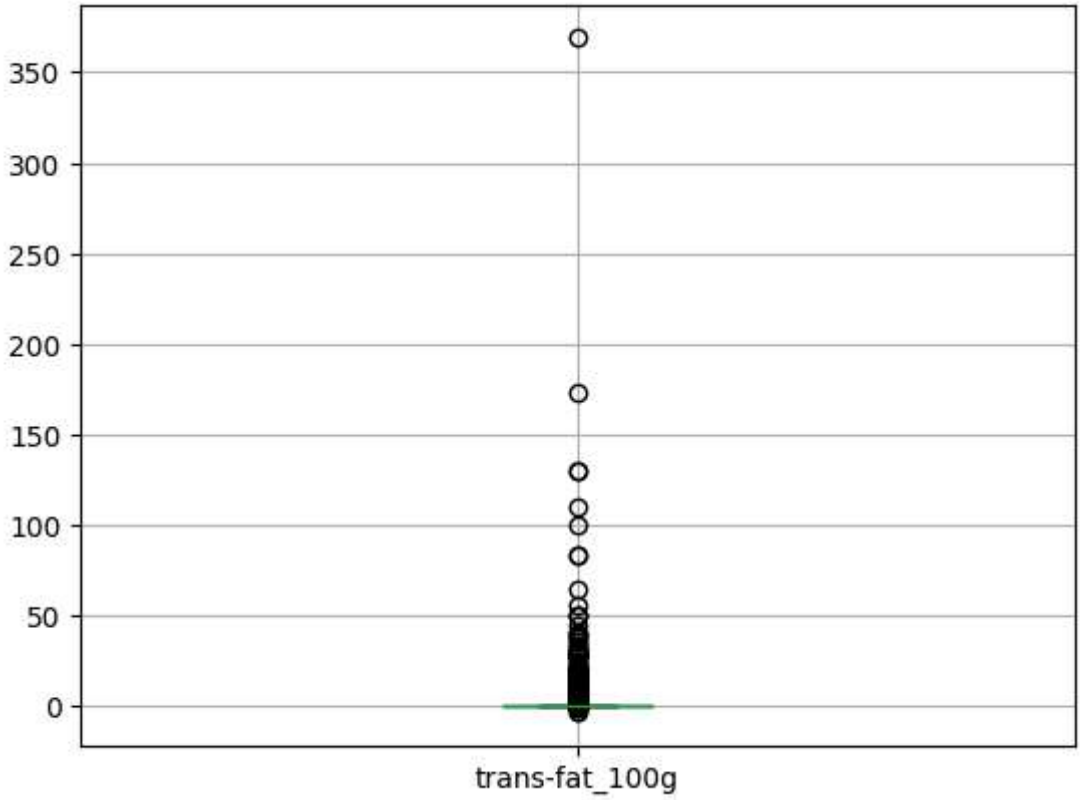

2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
ingredients_from_palm_oil_n

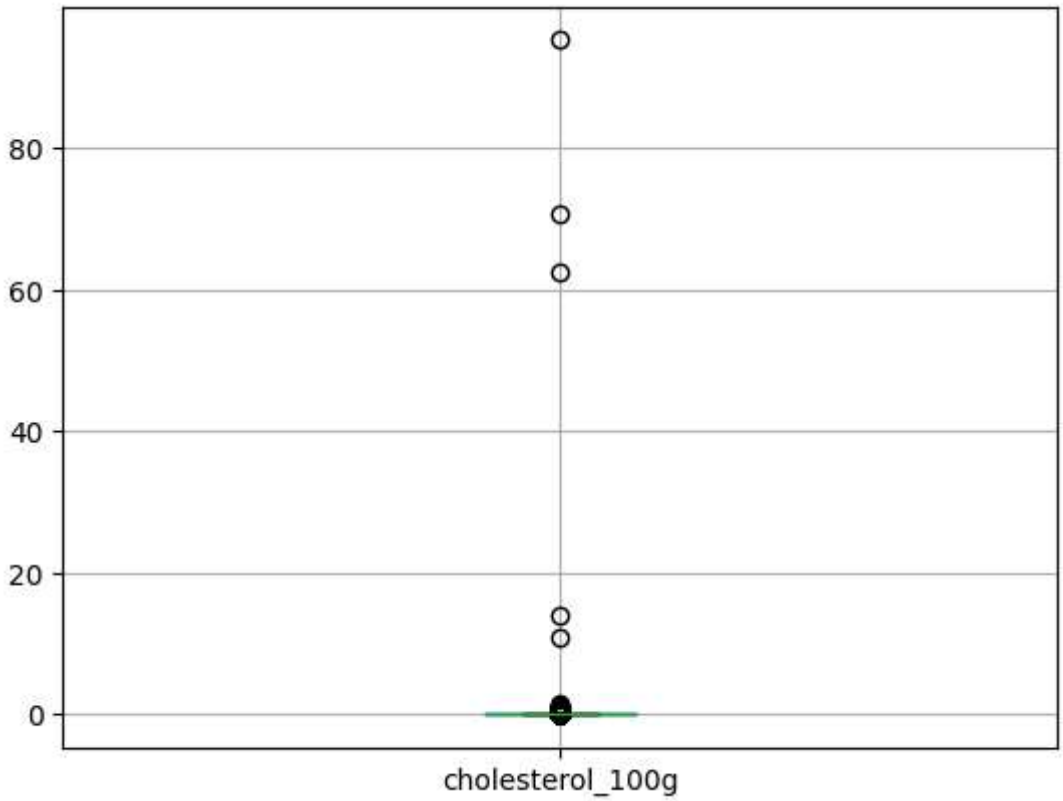

6
5
4
3
2
1
0
ingredients_that_may_be_from_palm_oil_n

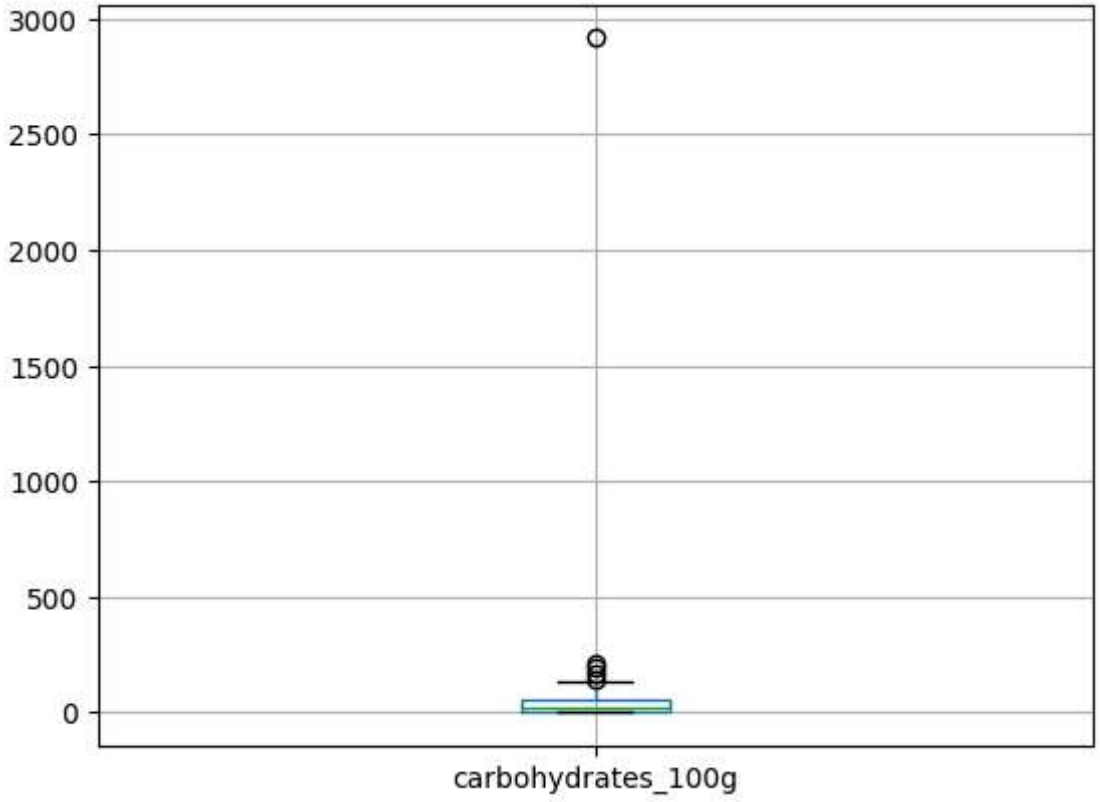

1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
energy_100g

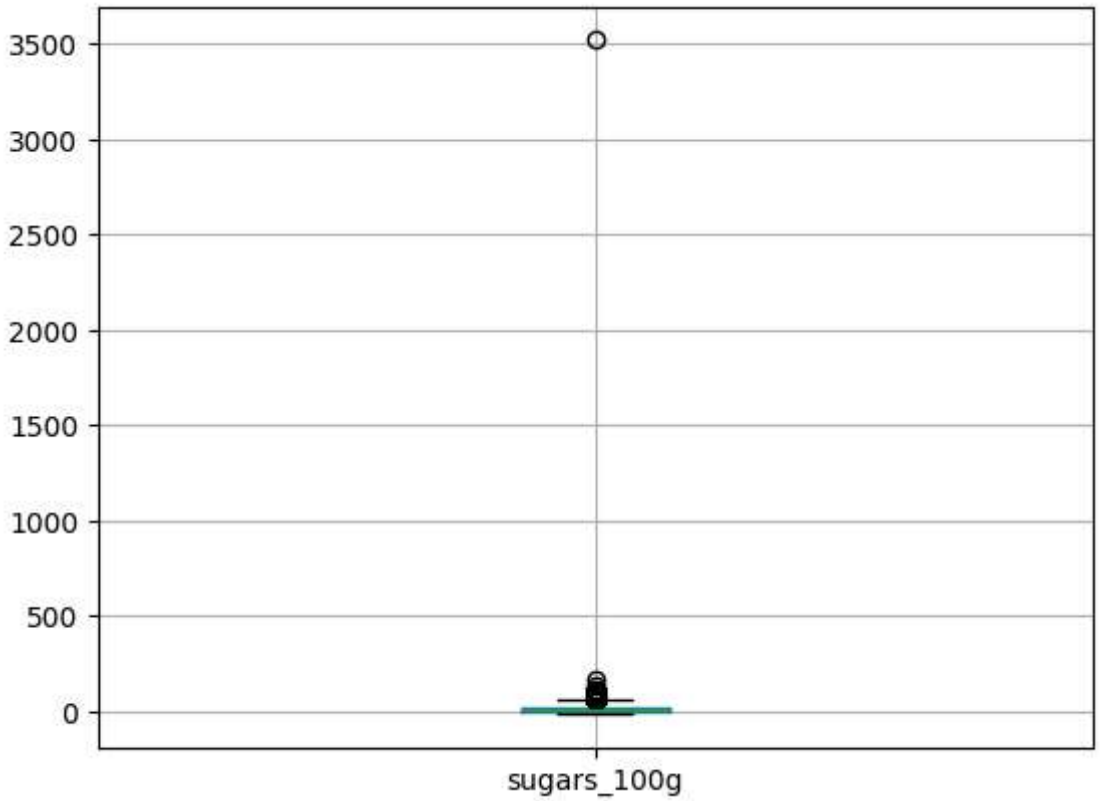

700
600
500
400
300
200
100
0
fat_100g

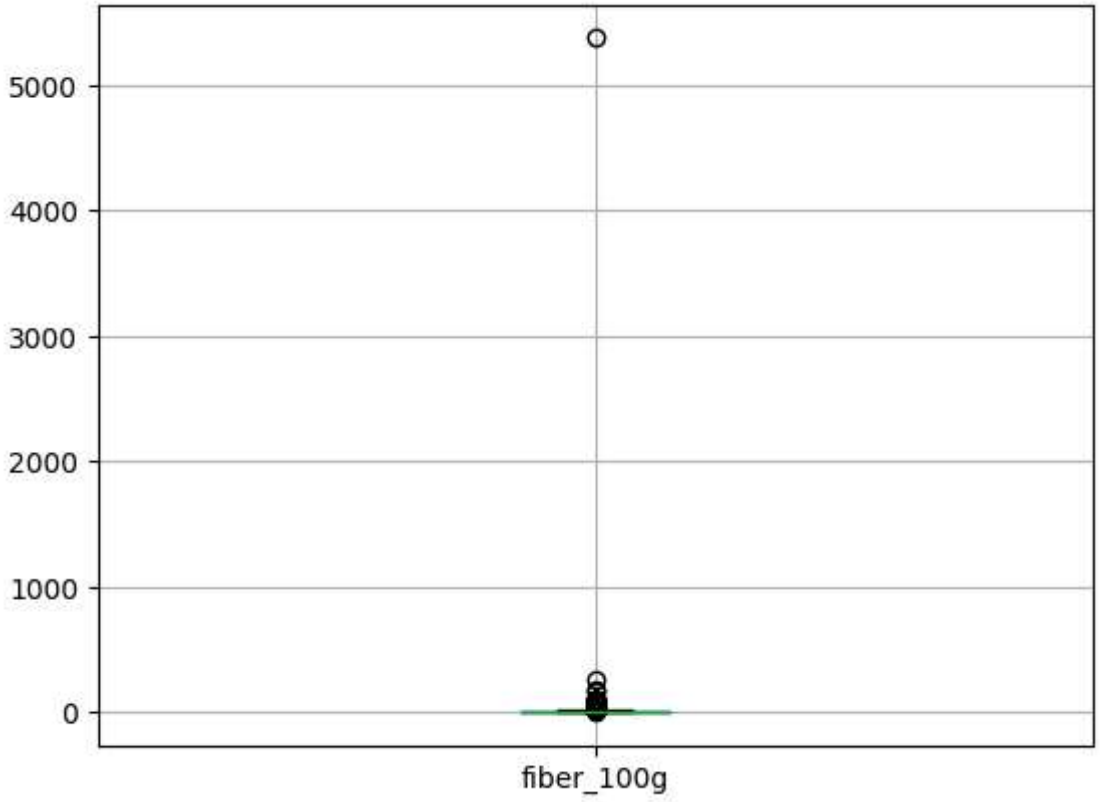

500
400
300
200
100
0
saturated-fat_100g

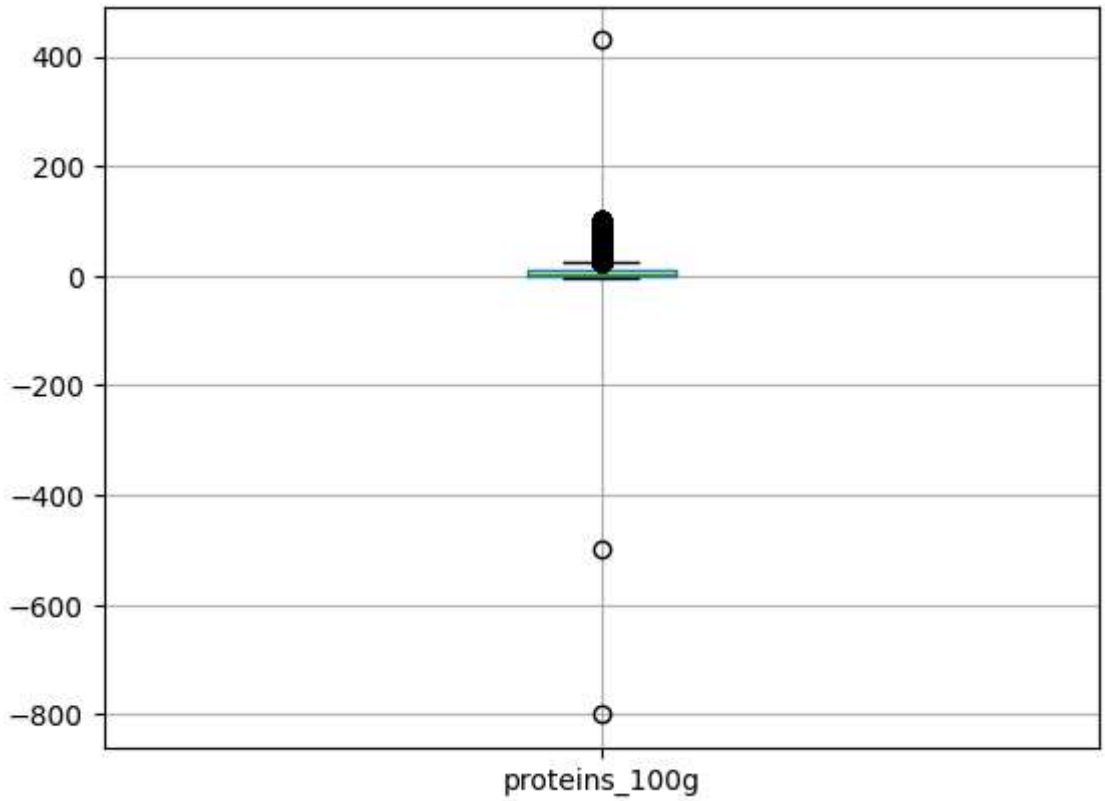

350
300
250
200
150
100
50
0
trans-fat_100g

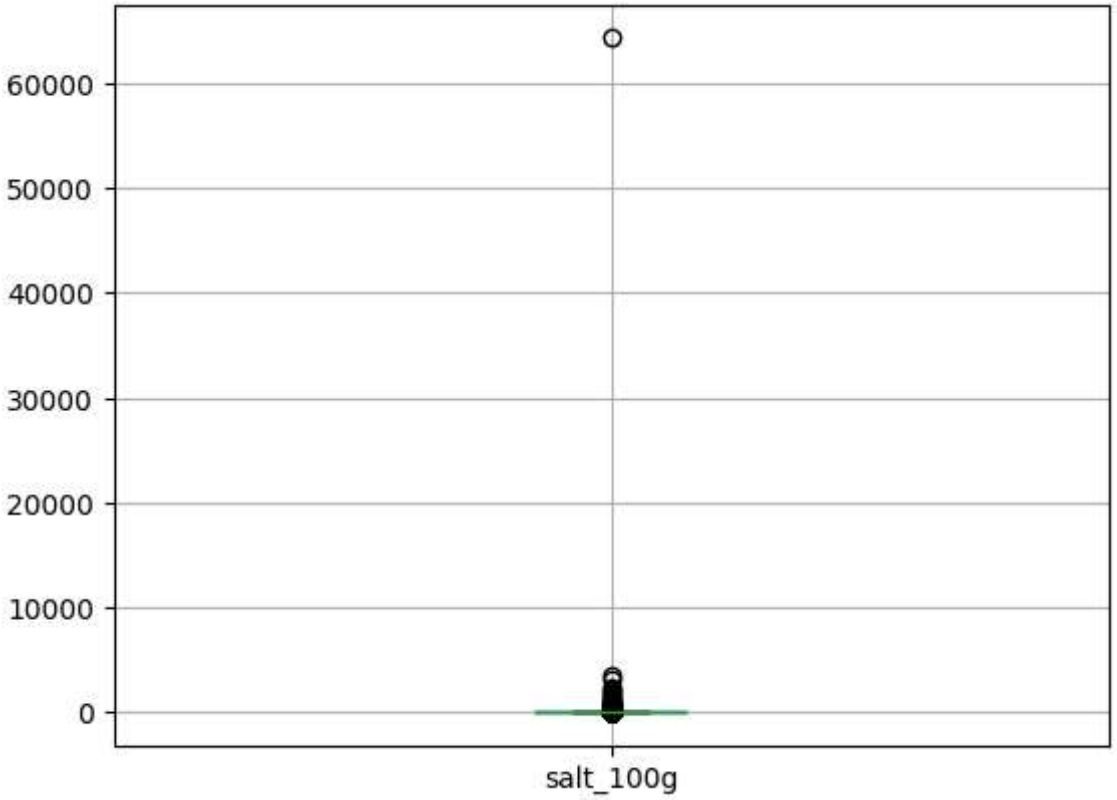

80
60
40
20
0
cholesterol_100g

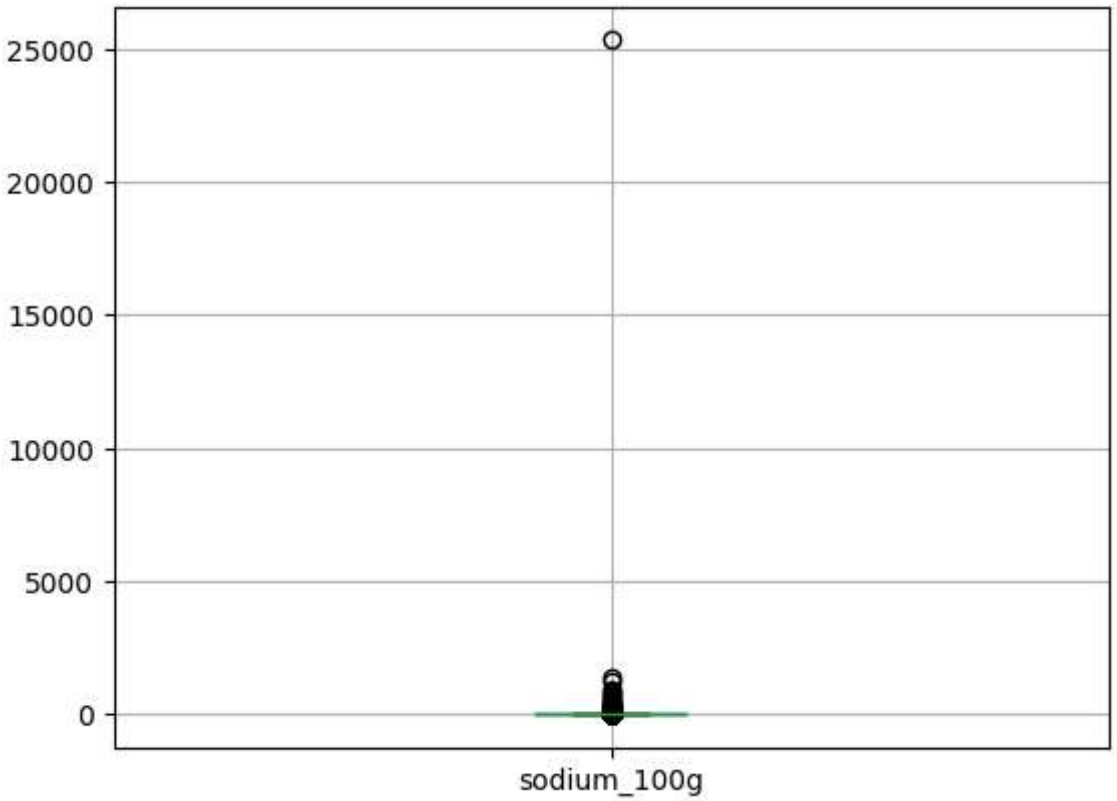

3000
2500
2000
1500
1000
500
0
carbohydrates_100g

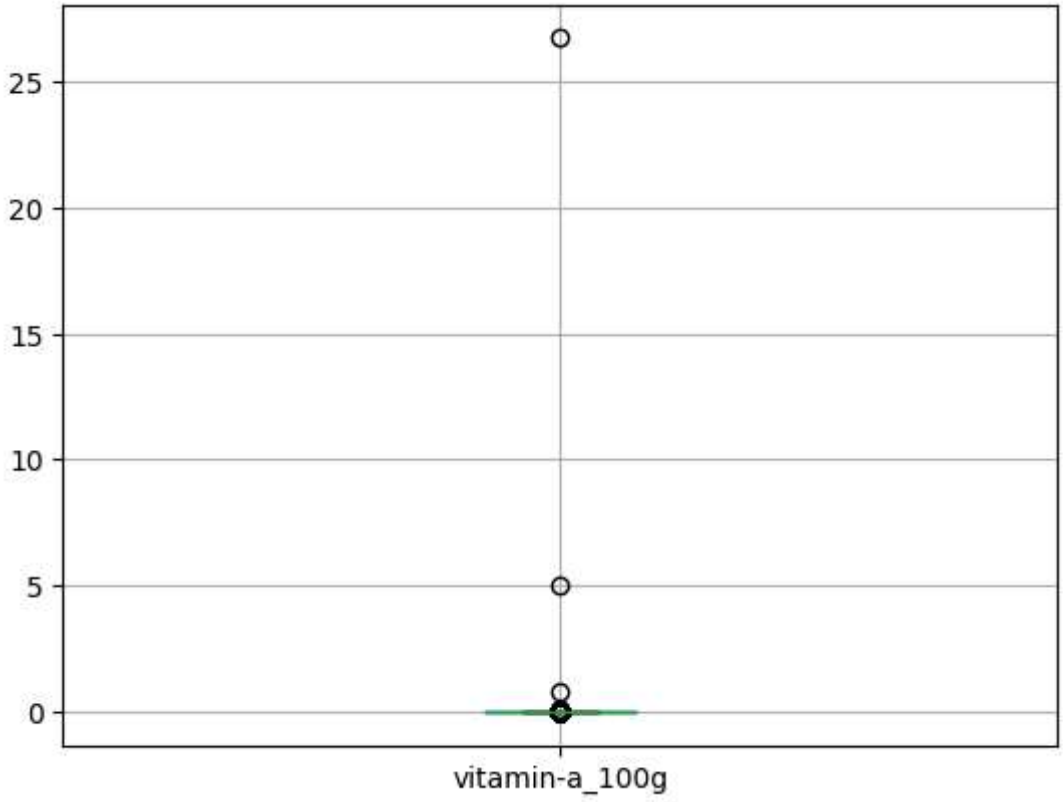

3500
3000
2500
2000
1500
1000
500
0
sugars_100g

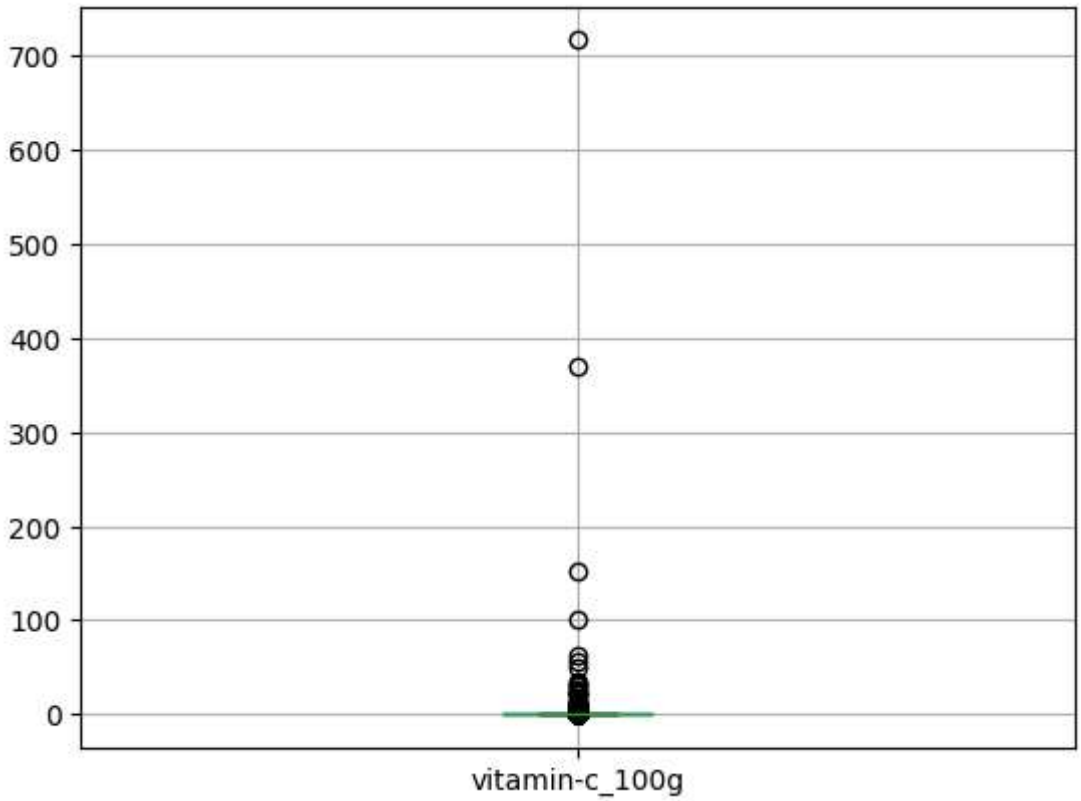

5000
4000
3000
2000
1000
0
fiber_100g

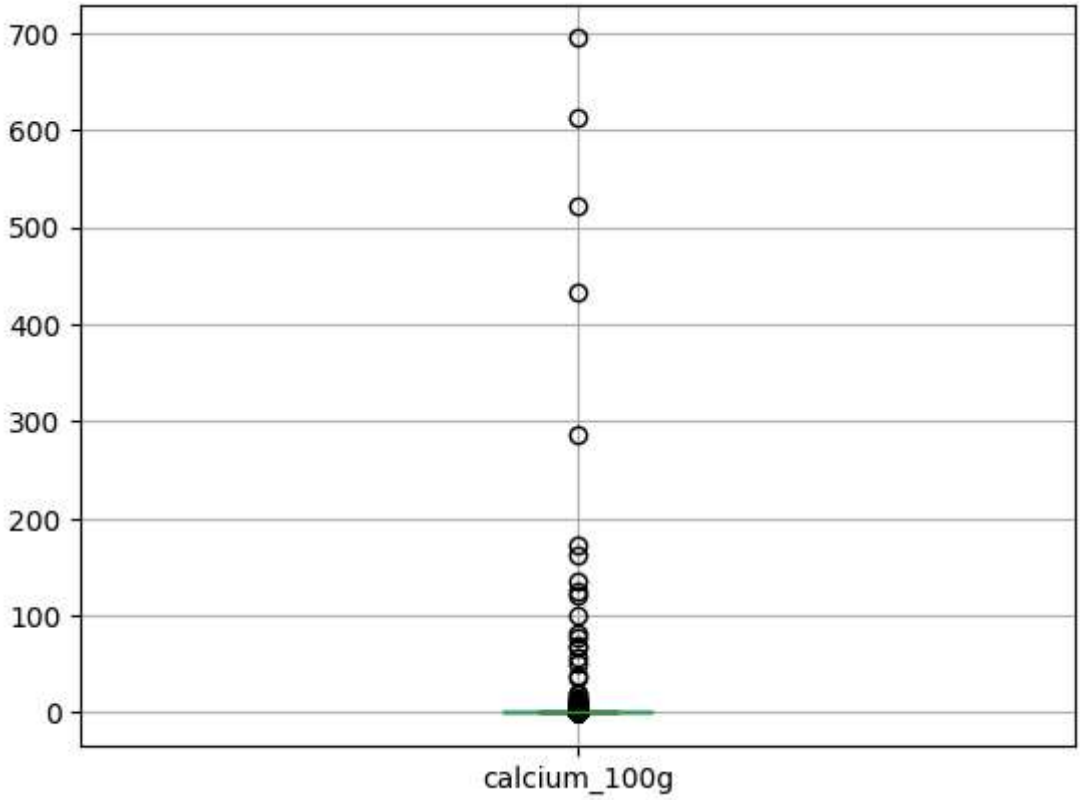

400
200
0
−200
−400
−600
−800
proteins_100g

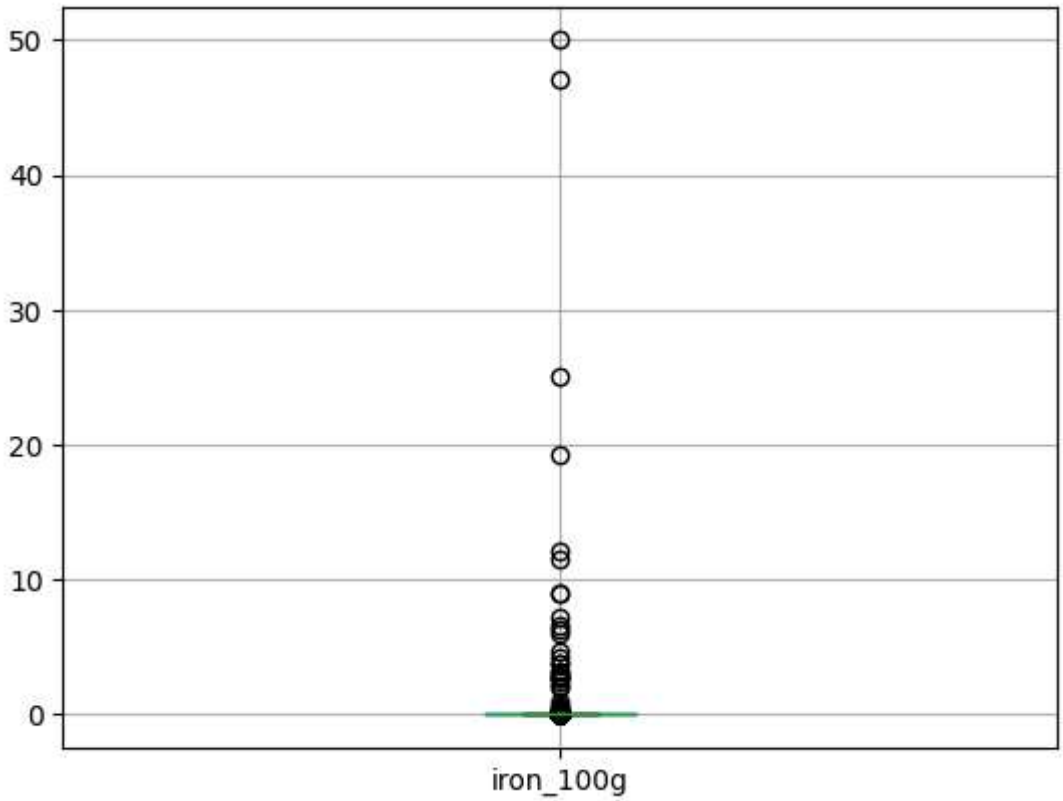

60000
50000
40000
30000
20000
10000
0
salt_100g

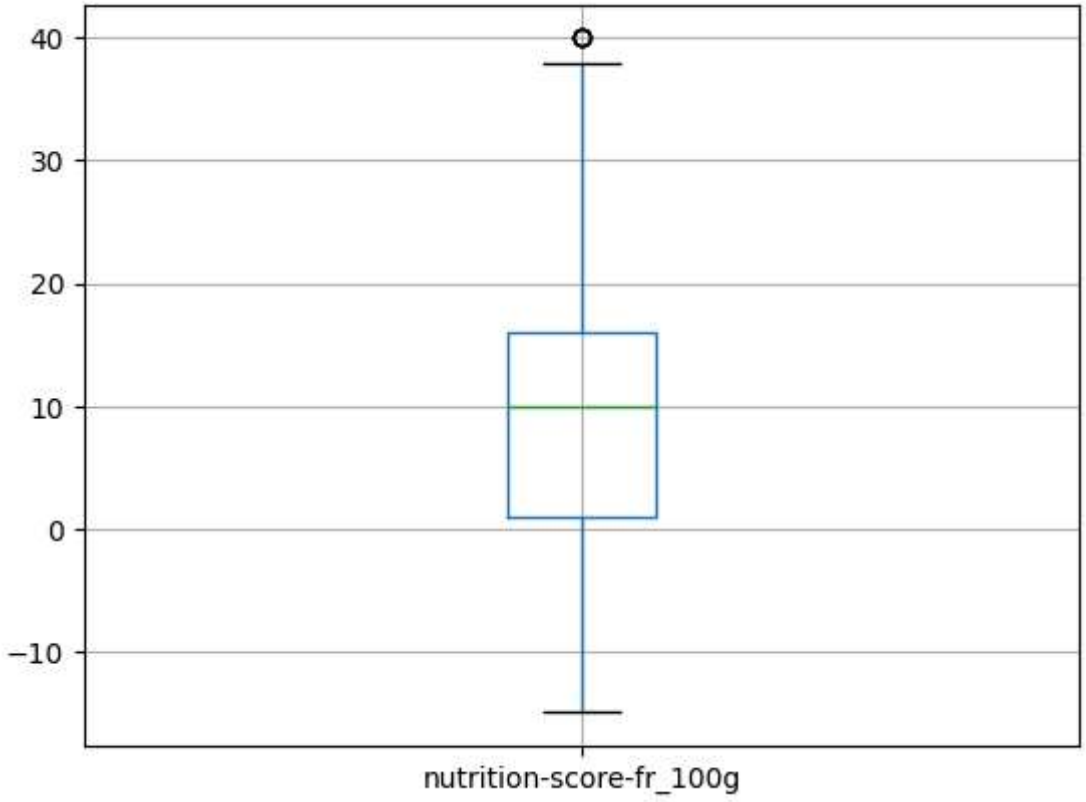

25000
20000
15000
10000
5000
0
sodium_100g

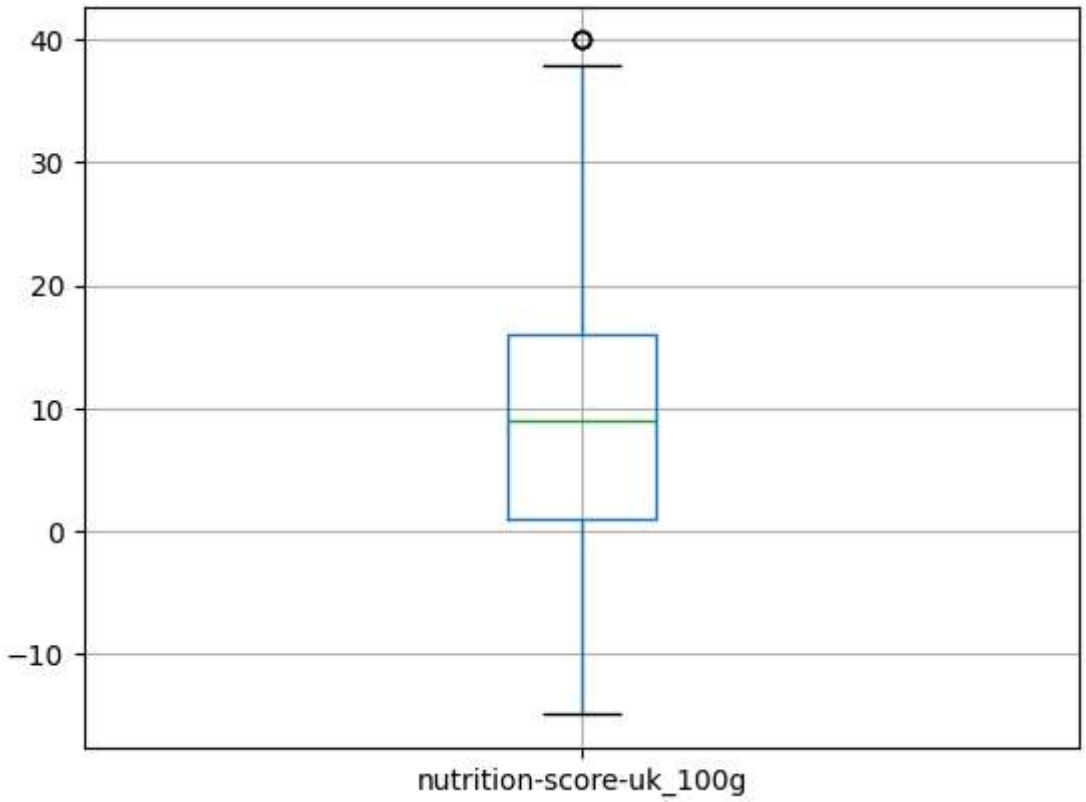

25
20
15
10
5
0
vitamin-a_100g


700
600
500
400
300
200
100
0
vitamin-c_100g


700
600
500
400
300
200
100
0
calcium_100g


50
40
30
20
10
0
iron_100g





Bien que biologiste de formation, je ne pense pas m'amuser à faire un tri parmi les valeurs qui cassent le planché sans consultation de collégue spécialisé. Je vais me contenter d'éliminer le subset ou il y a des outliers.

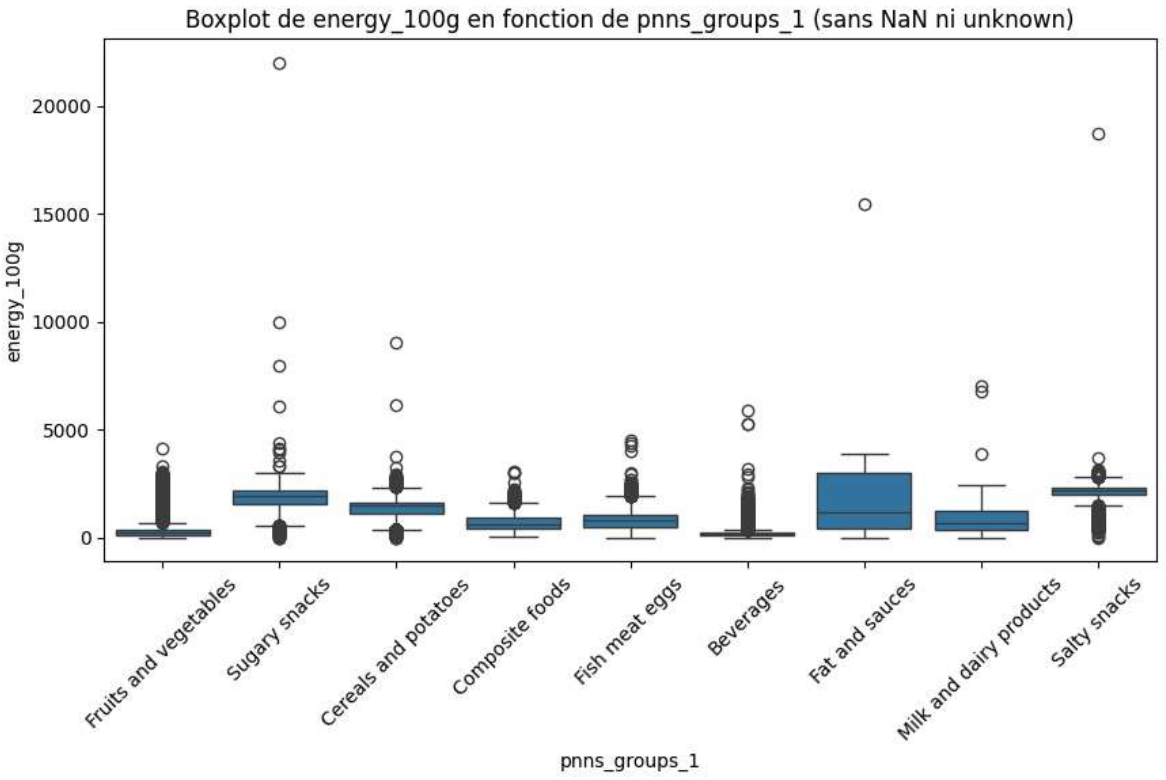
In [ ]:
```
# Réalisation d'un z-score pour chaque variable numérique
from scipy import stats
from scipy.stats import zscore

df_sans_outliers = sansdoublons.copy()
```

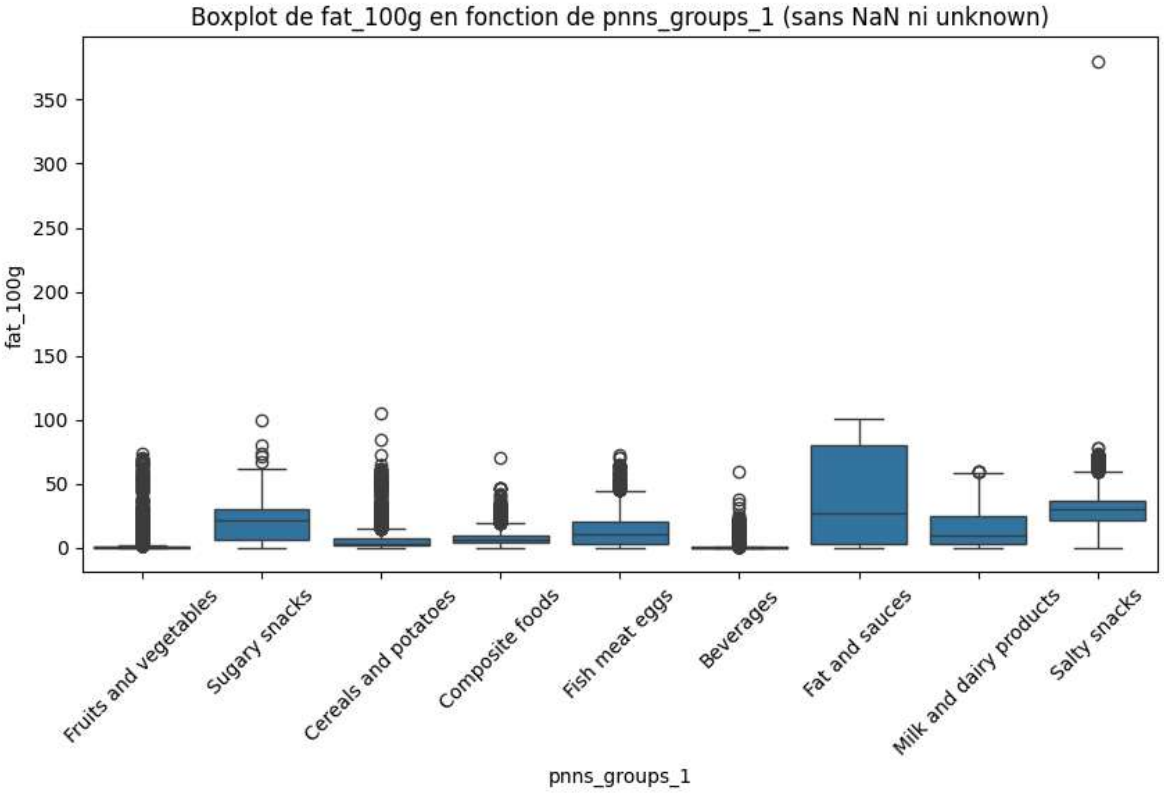
```
for col in df_sans_outliers.select_dtypes(include=['float64']).columns:
    z_scores = zscore(df_sans_outliers[col])
    threshold = 3
    # Filtrage des outliers en utilisant une indexation booléenne
    df_sans_outliers = df_sans_outliers[(z_scores > -threshold)]

# On affiche le nombre de lignes supprimées
print(len(sansdoublons) - len(df_sans_outliers))
print(df_sans_outliers.shape)
```

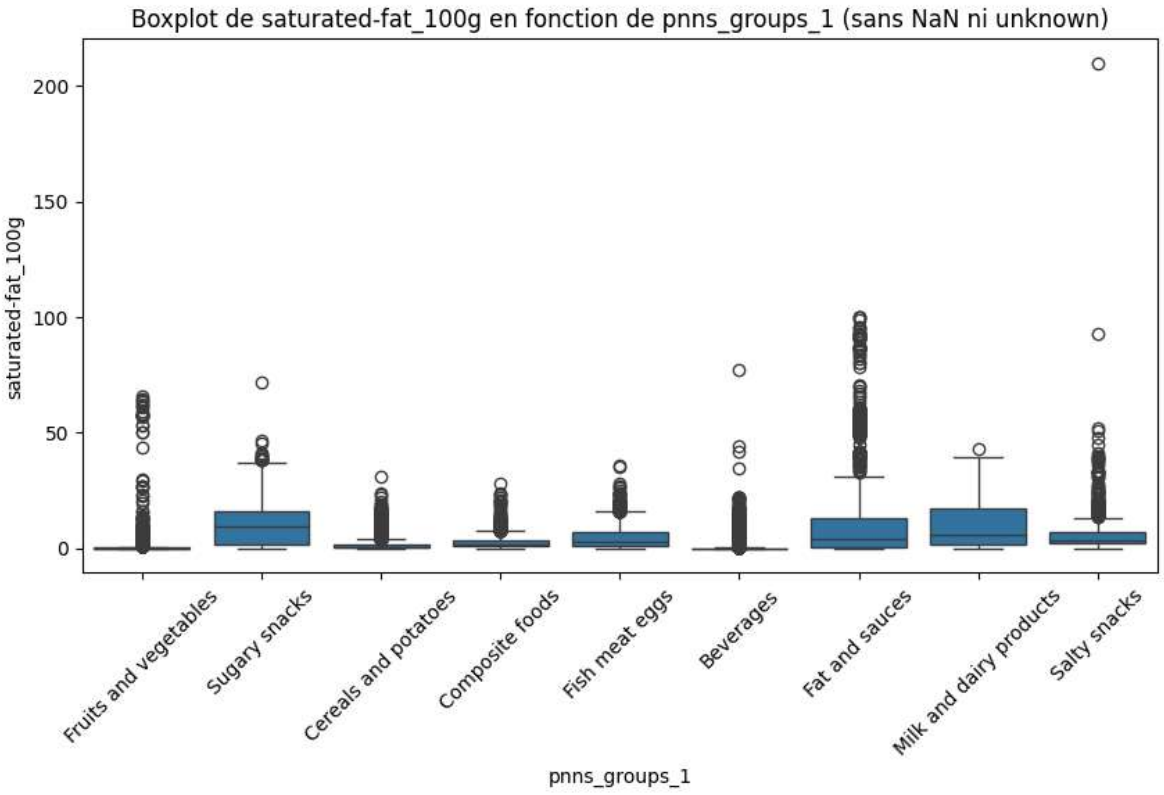
```
320750
(0, 48)
```

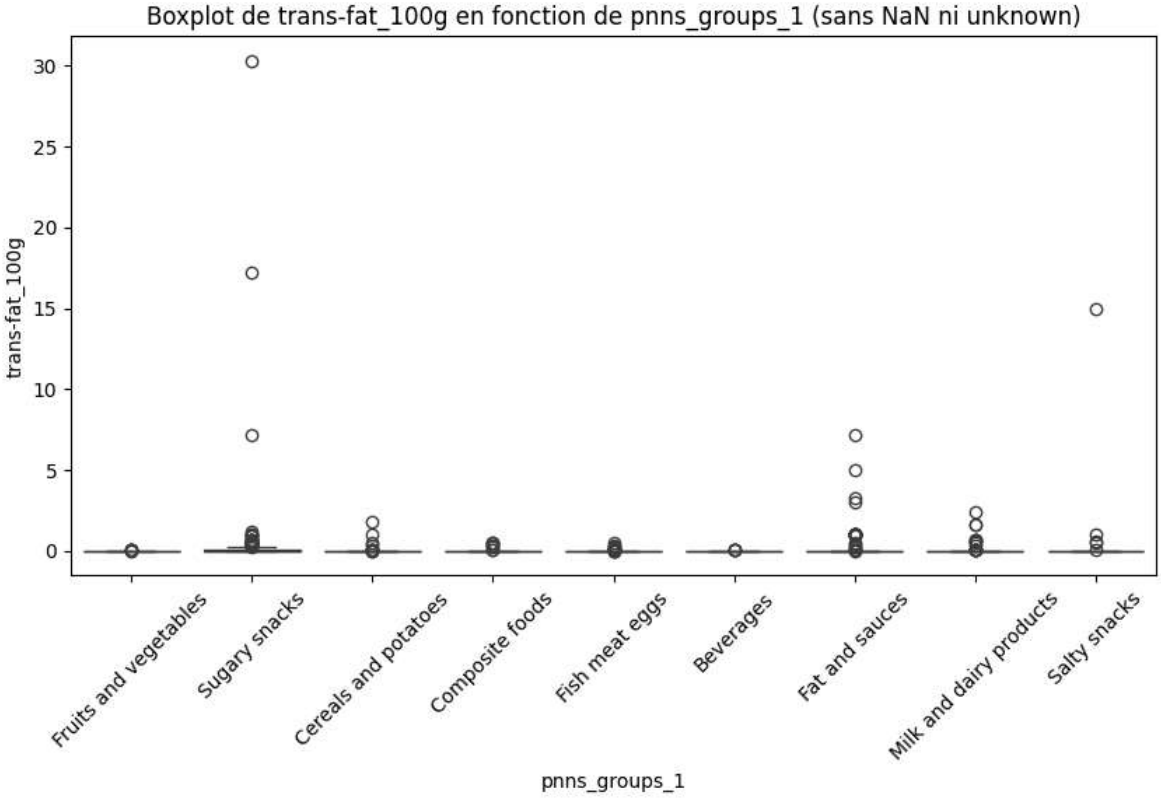
Bon ça va être dur de toucher aux outliers de manières si large... Je vais continuer de travailler avec le sample original car je pense qu'il y a trop de valeurs numériques susceptible d'être un outliers selon les catégories.

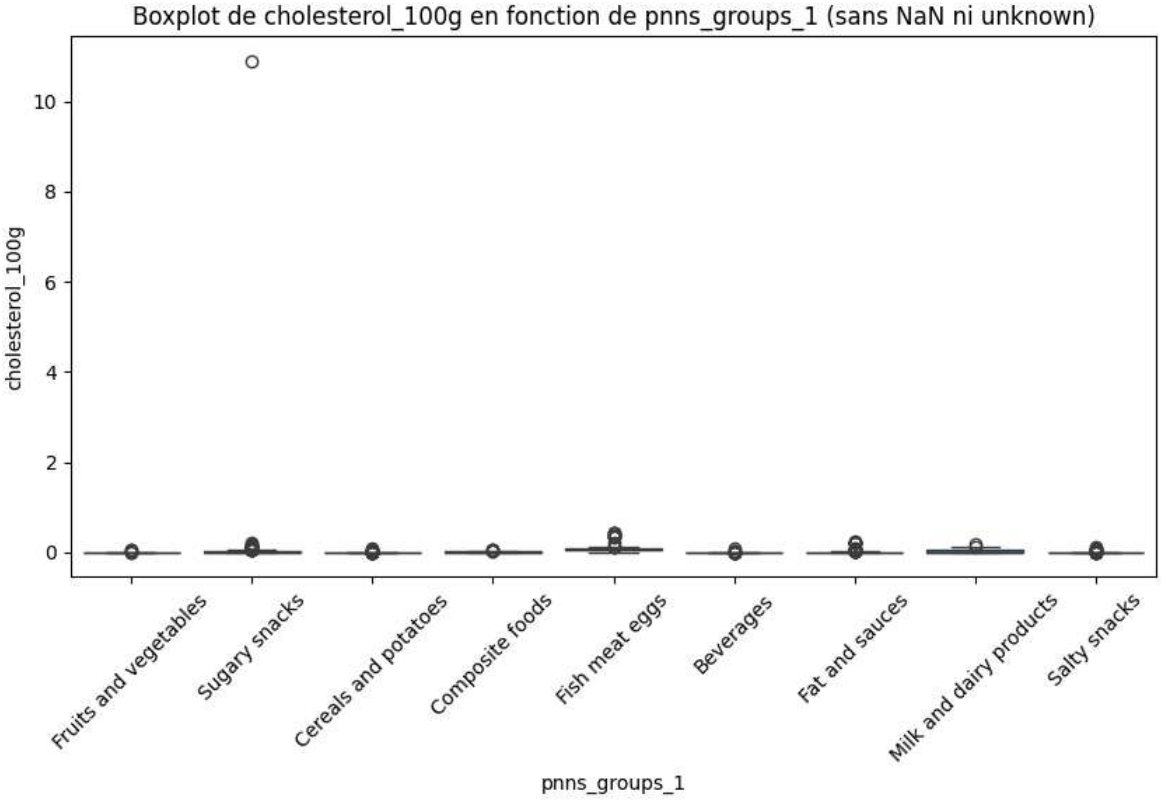
In [ ]:
```
import seaborn as sns

columns_100g = [col for col in sansdoublons.columns if '_100g' in col]

for col in columns_100g:
    plt.figure(figsize=(10, 5))

    # Créer le boxplot en utilisant le DataFrame filtré
    sns.boxplot(x="pnns_groups_1", y=col, data=sansdoublons)

    plt.title(f'Boxplot de {col} en fonction de pnns_groups_1 (sans NaN ni unkno
    plt.xlabel('pnns_groups_1')
    plt.ylabel(col)
    plt.xticks(rotation=45)
    plt.show()
```

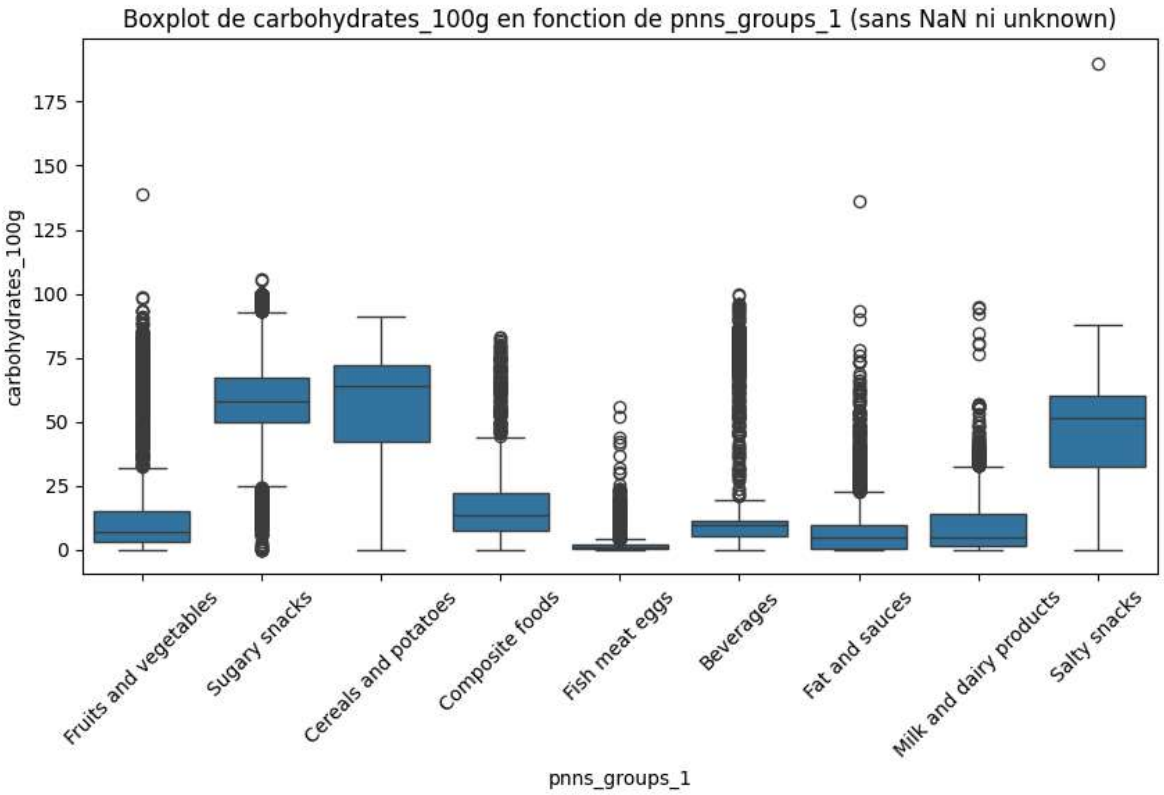


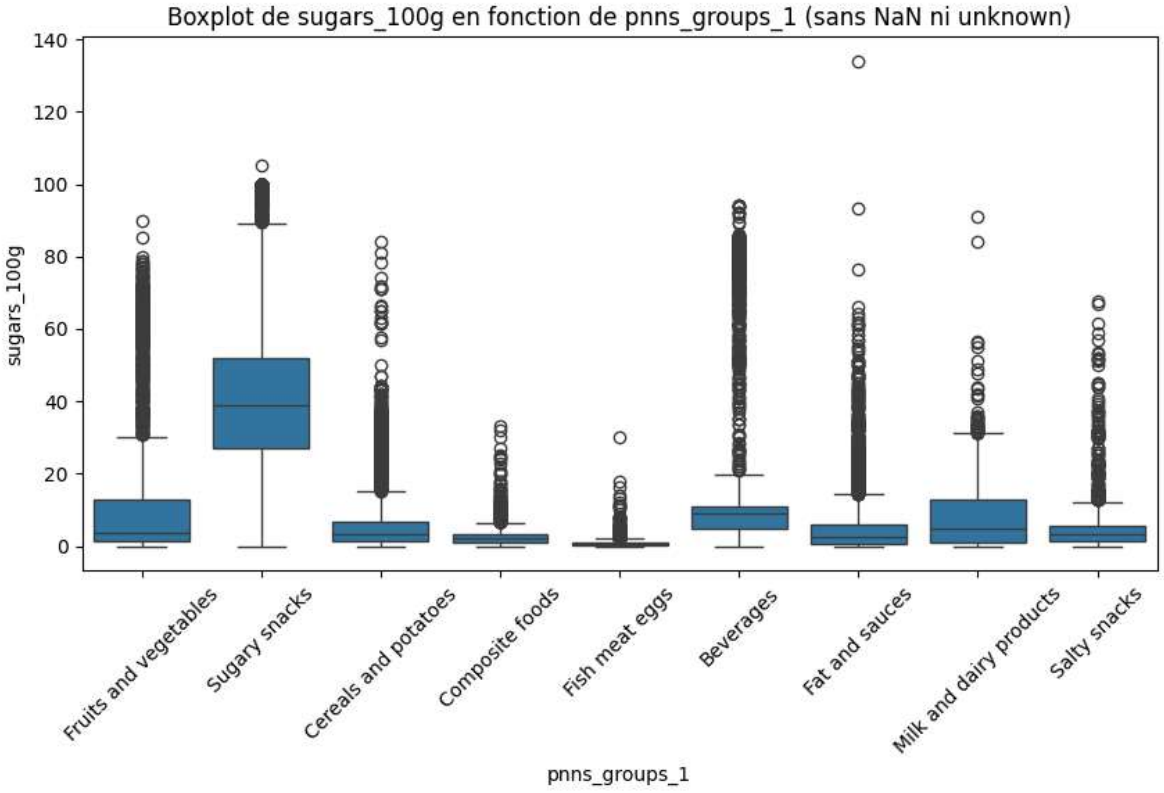

Boxplot de fat_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
fat_100g
0
50
100
150
200
250
300
350
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

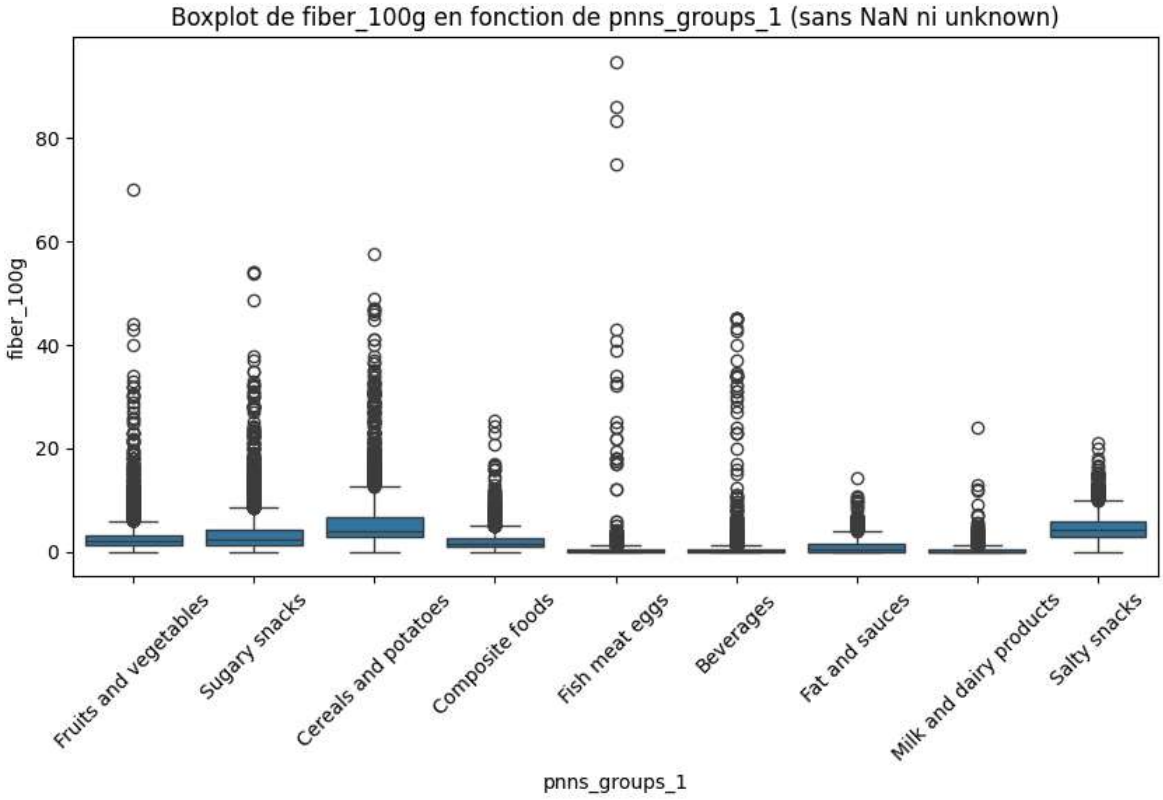

Boxplot de saturated-fat_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
saturated-fat_100g
0
50
100
150
200
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

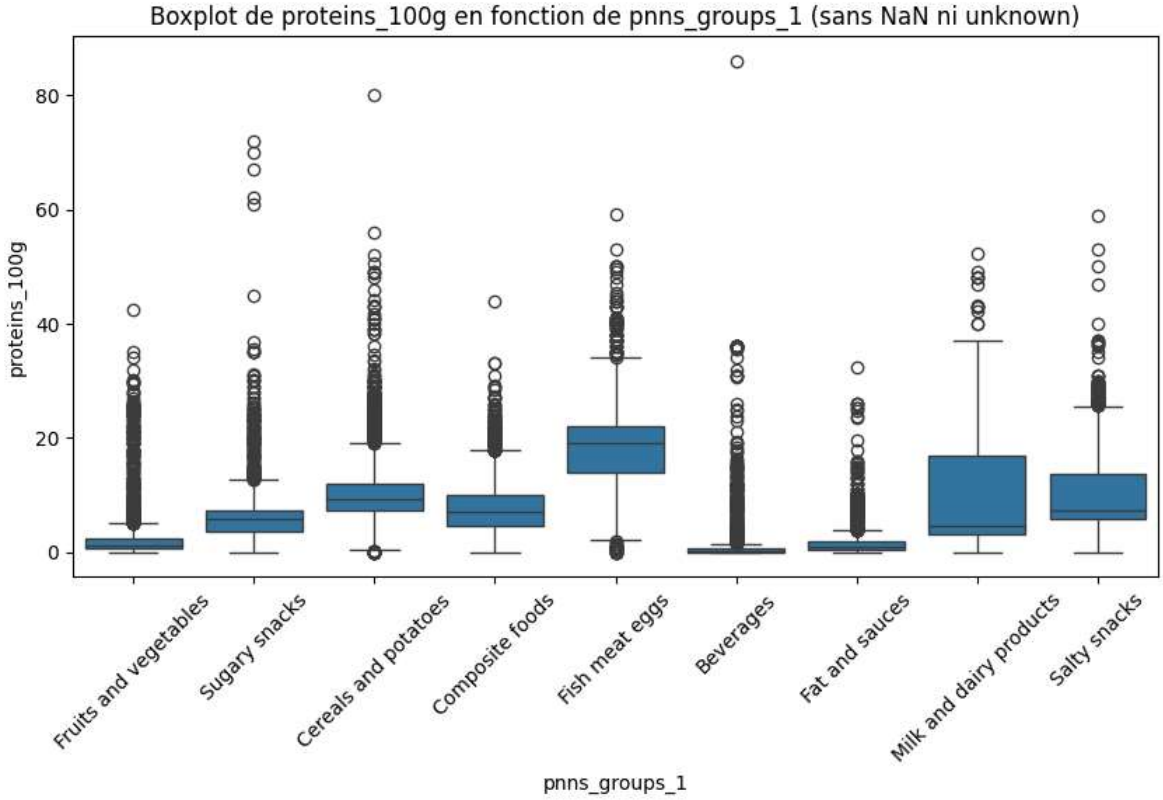

Boxplot de trans-fat_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
trans-fat_100g
30
25
20
15
10
5
0
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

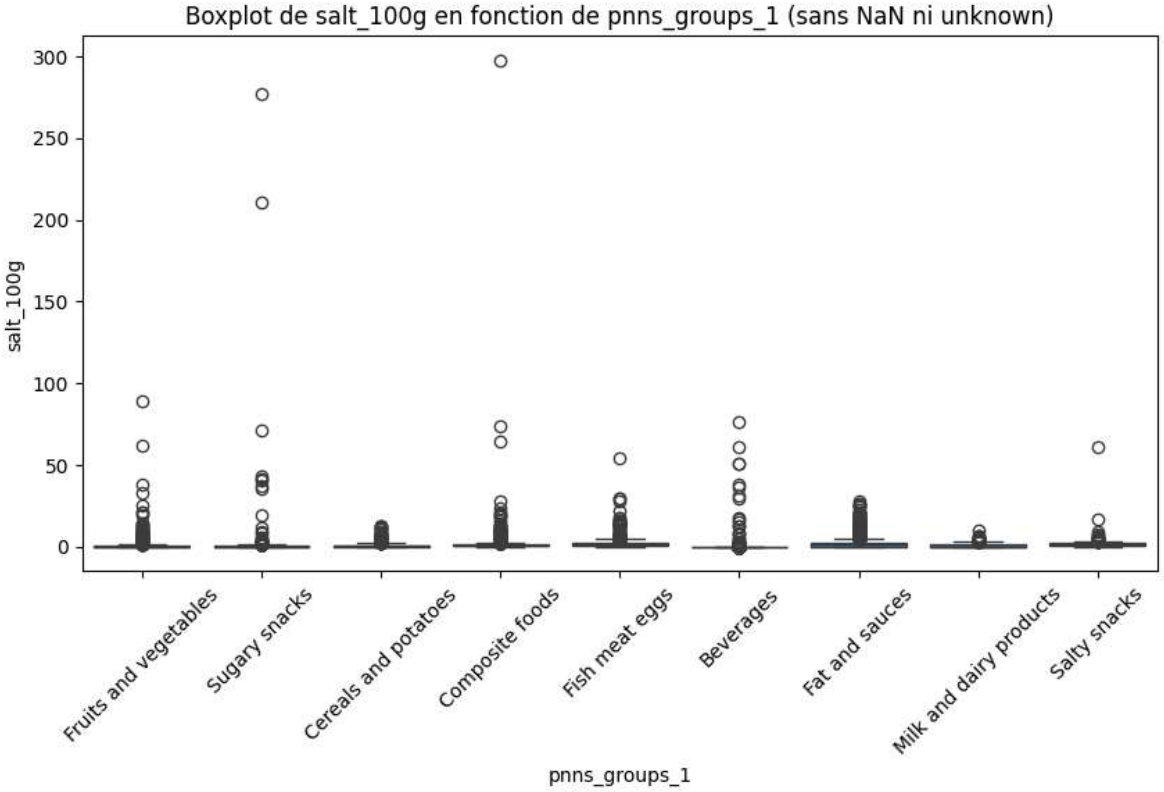

Boxplot de cholesterol_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
cholesterol_100g
10
8
6
4
2
0
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

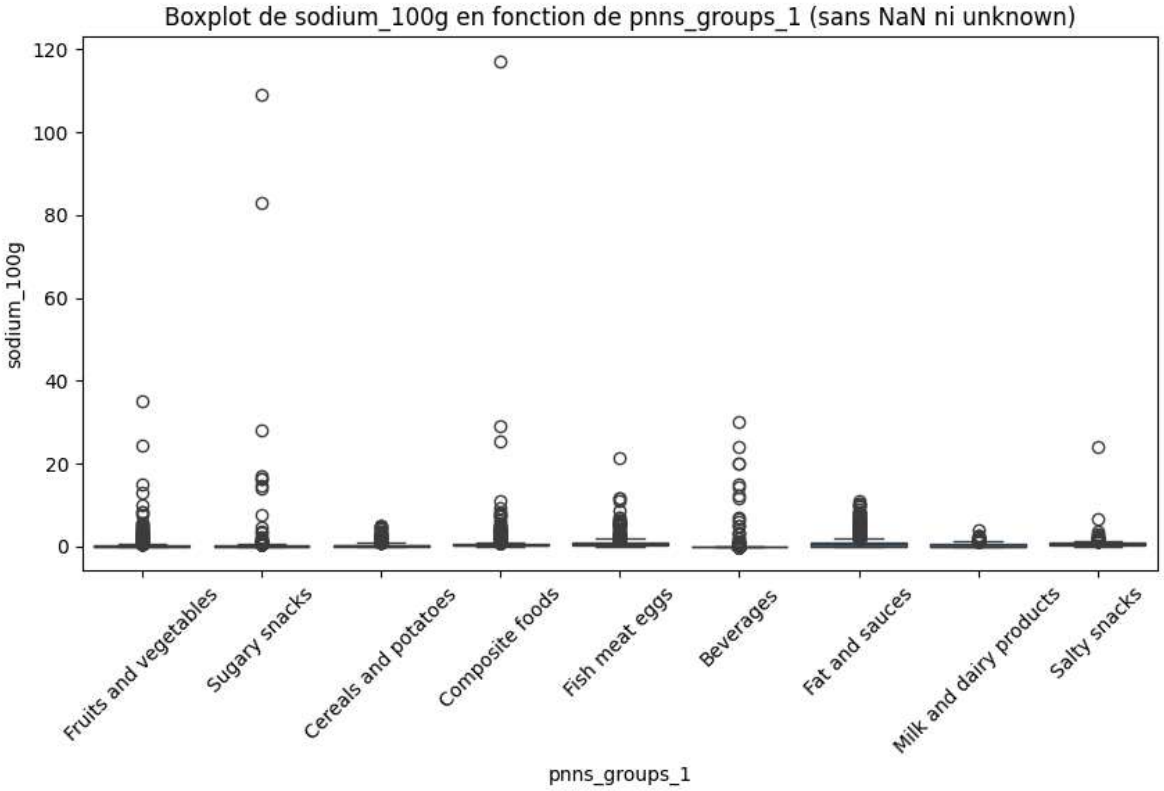

Boxplot de carbohydrates_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
carbohydrates_100g
0
25
50
75
100
125
150
175
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

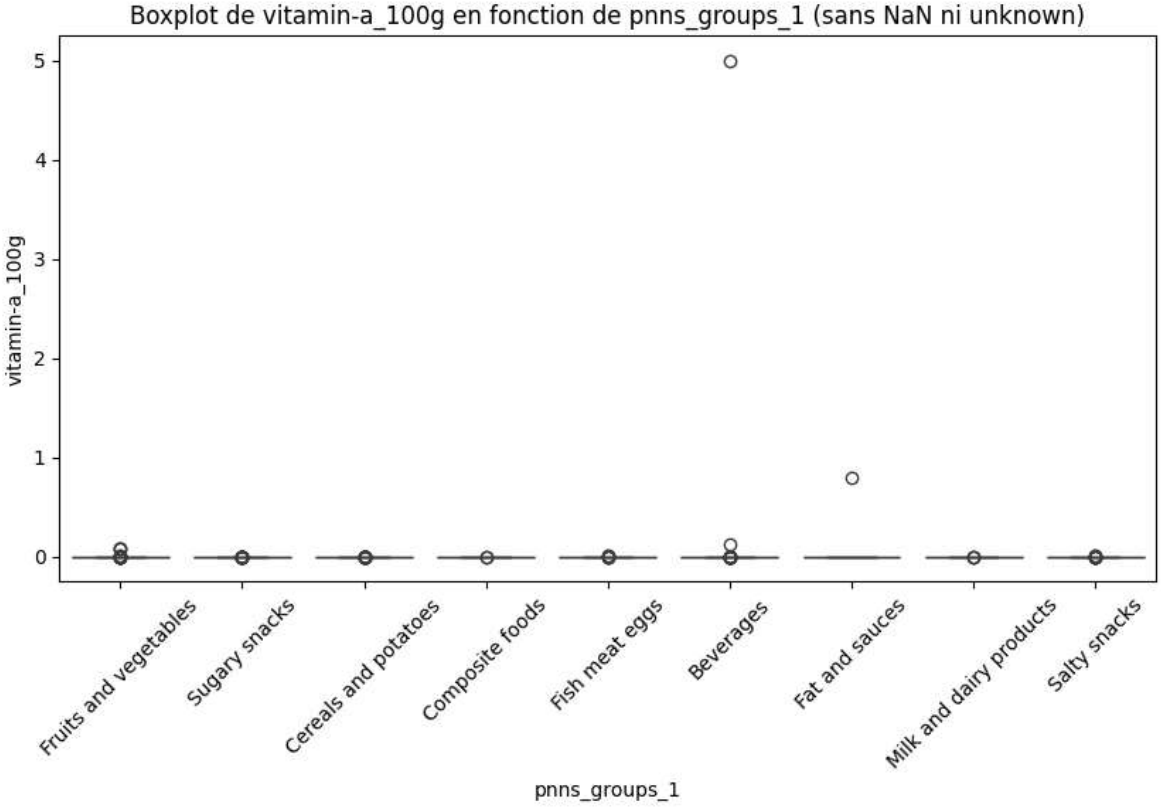

Boxplot de sugars_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
sugars_100g
0
20
40
60
80
100
120
140
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

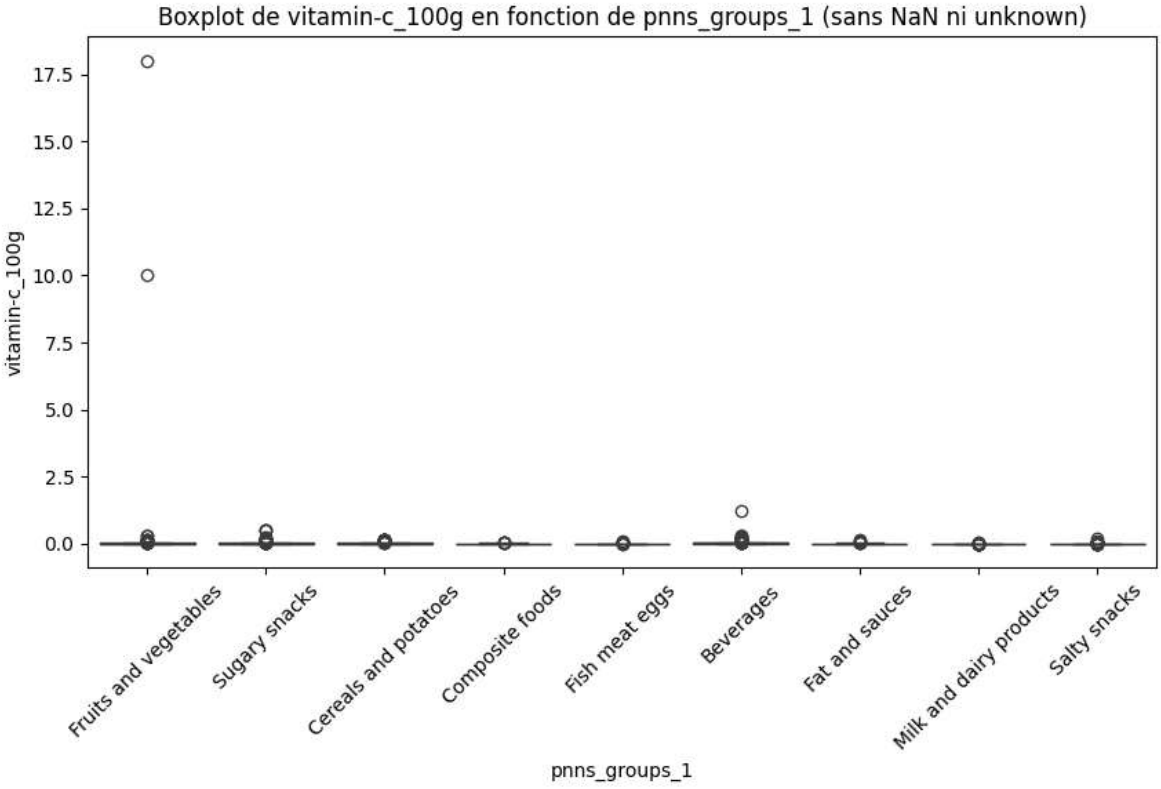

Boxplot de fiber_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
fiber_100g
0
20
40
60
80
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

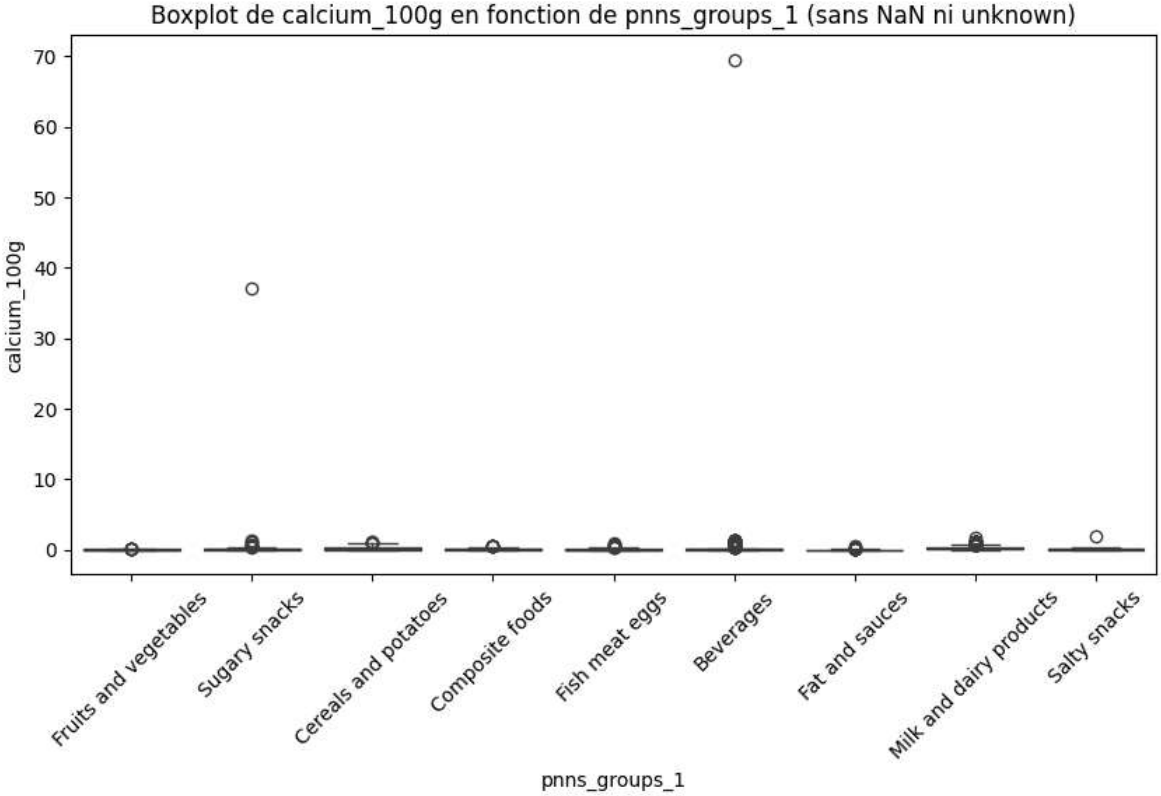

Boxplot de proteins_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
proteins_100g
0
20
40
60
80
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

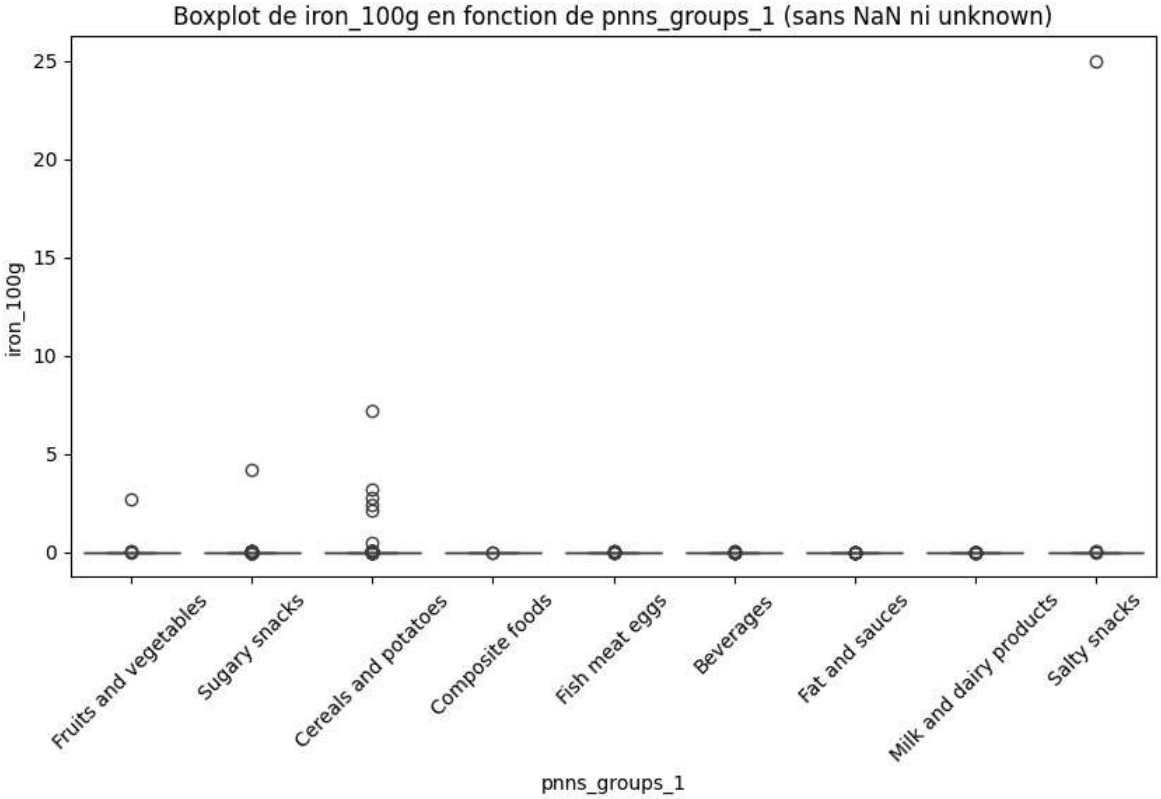

Boxplot de salt_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
salt_100g
300
250
200
150
100
50
0
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

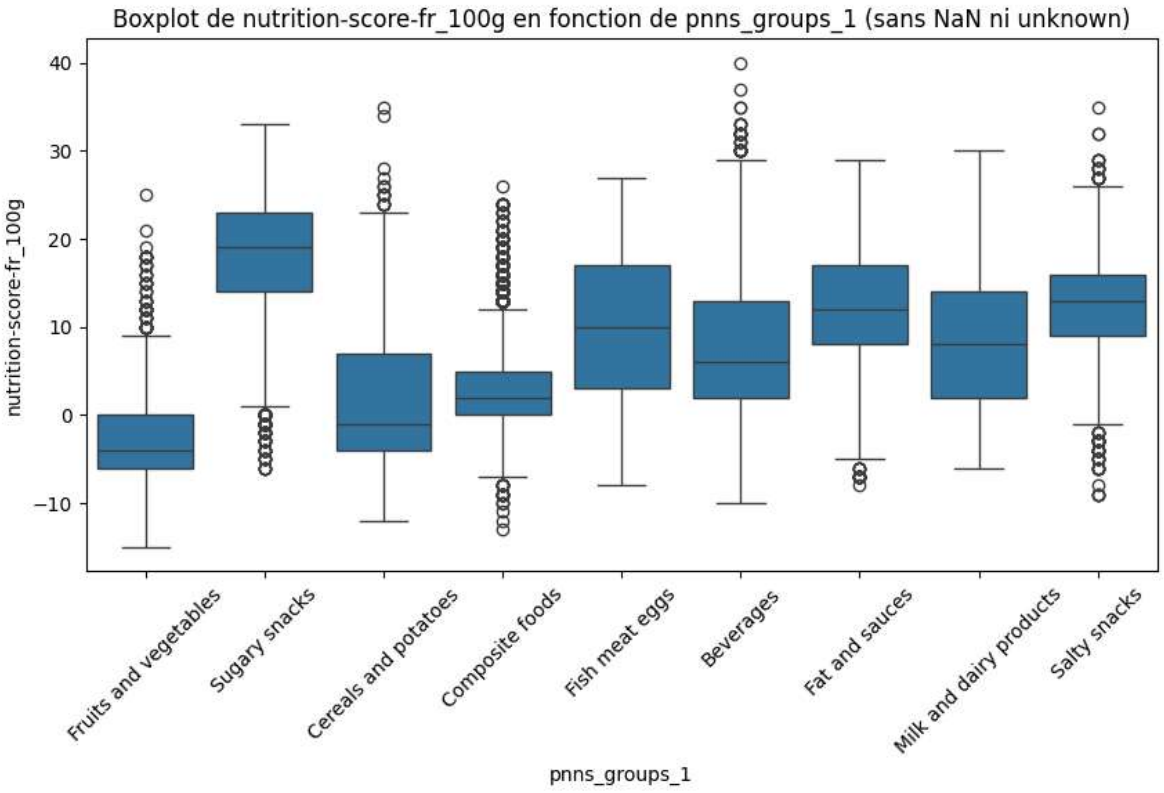

Boxplot de sodium_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
sodium_100g
120
100
80
60
40
20
0
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1

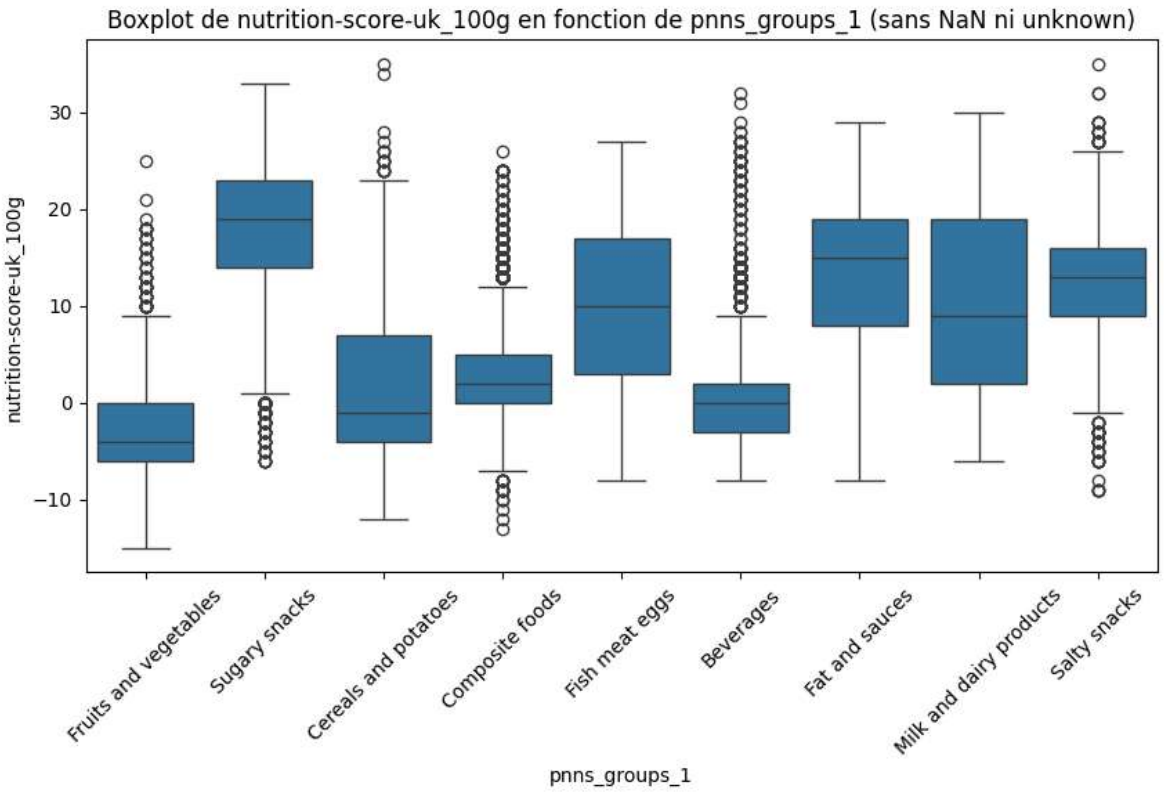

Boxplot de vitamin-a_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
vitamin-a_100g
0
1
2
3
4
5
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1


Boxplot de vitamin-c_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
vitamin-c_100g
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1


Boxplot de calcium_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
calcium_100g
0
10
20
30
40
50
60
70
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1


Boxplot de iron_100g en fonction de pnns_groups_1 (sans NaN ni unknown)
iron_100g
0
5
10
15
20
25
Fruits and vegetables
Sugary snacks
Cereals and potatoes
Composite foods
Fish meat eggs
Beverages
Fat and sauces
Milk and dairy products
Salty snacks
pnns_groups_1





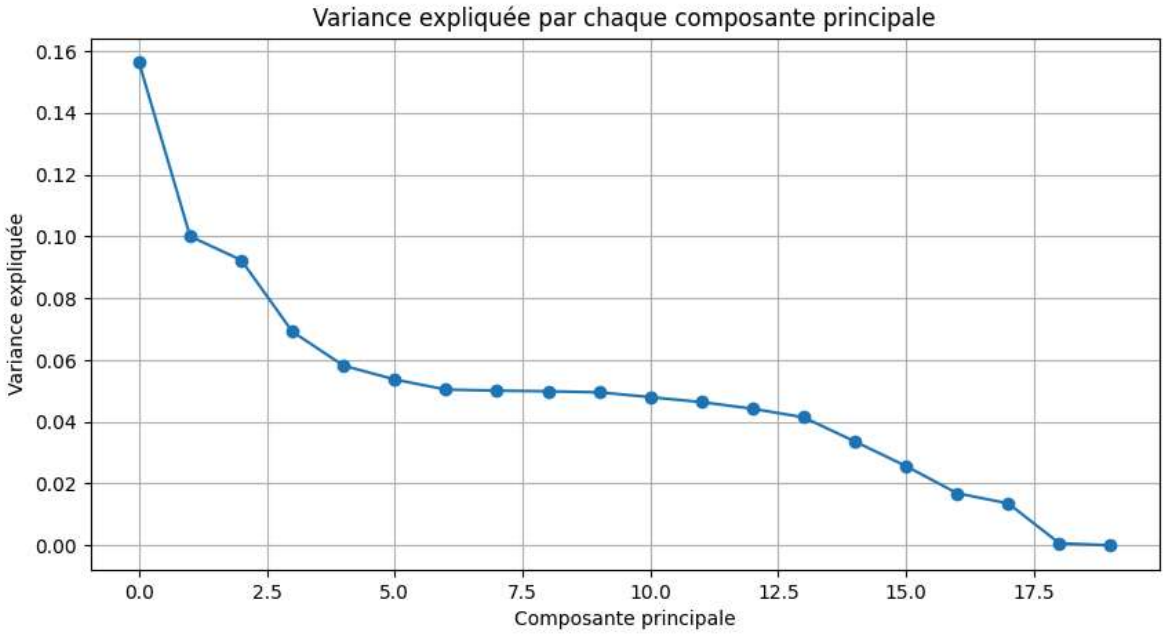
Bon la première ACP n'explique pas grand chose et je pense que c'est normal, déjà il y a des outliers et en plus après la vue des boxplots précédents on peut voir qu'il y a beaucoup de paramètres qui a eux seuls n'expliquent pas grand chose sur la variation.

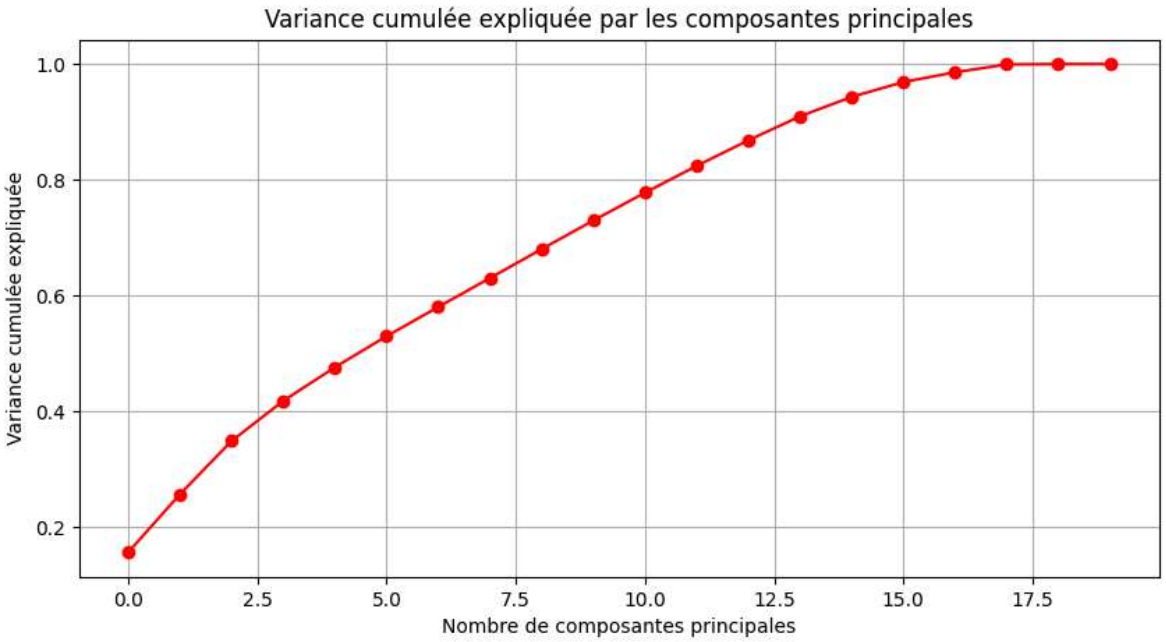
In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Sélection des colonnes numériques
df_numeric = sansdoublons.select_dtypes(include=['float64'])

# Remplacement des NaN par la moyenne de la colonne
df_numeric = df_numeric.fillna(df_numeric.mean())

# Standardisation des données
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_numeric)
```

In [ ]:
```
# Création de l'objet PCA
pca = PCA()

# Ajustement de l'ACP sur les données standardisées
pca.fit(df_scaled)

# Création d'un DataFrame avec les composantes principales
df_pca_components = pd.DataFrame(pca.components_, columns=df_numeric.columns)
```

In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(pca.explained_variance_ratio_, marker='o')
plt.title('Variance expliquée par chaque composante principale')
plt.xlabel('Composante principale')
plt.ylabel('Variance expliquée')
plt.grid(True)
plt.show()

# Affichage de la variance cumulée expliquée par les composantes principales
plt.figure(figsize=(10, 5))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o', color='red')
plt.title('Variance cumulée expliquée par les composantes principales')
plt.xlabel('Nombre de composantes principales')
plt.ylabel('Variance cumulée expliquée')
plt.grid(True)
plt.show()
```



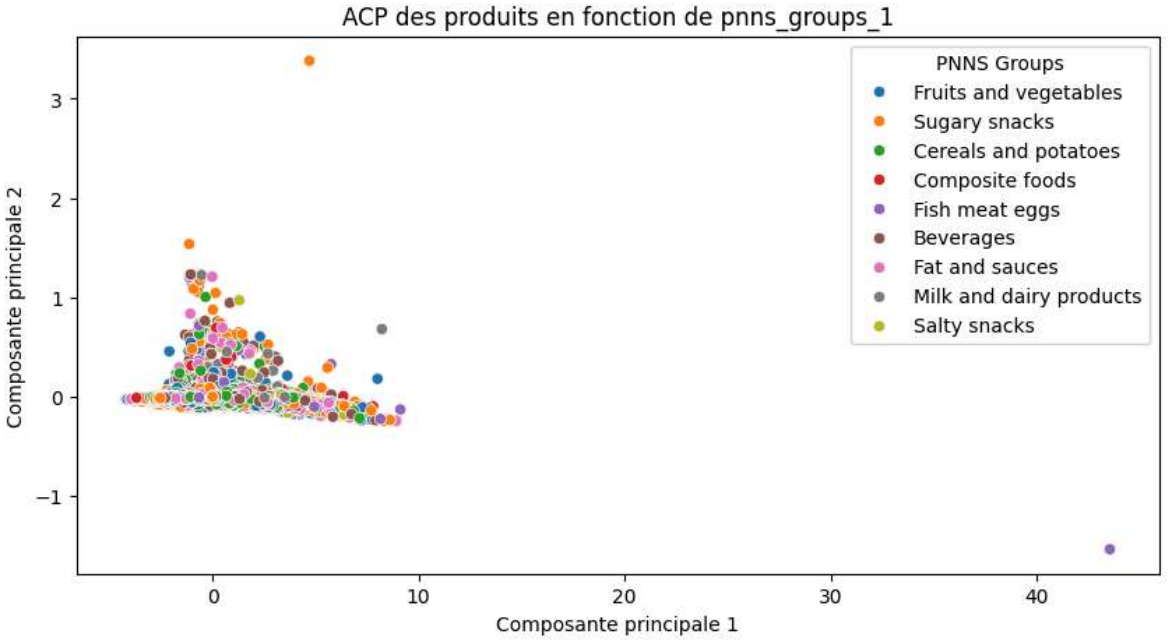


In [ ]:
```
df_pca = pd.DataFrame(pca.transform(df_scaled), columns=[f'PC{i+1}' for i in ran
df_pca['pnns_groups_1'] = sansdoublons['pnns_groups_1']
print(df_pca.head())
```

```
            PC1           PC2           PC3           PC4           PC5  \
0  2.821034e-17  3.049901e-18  9.514832e-17 -2.708483e-17  4.003322e-17
1  2.588463e+00 -6.618238e-02 -8.743787e-01 -7.571375e-01 -4.263940e-01
2 -1.185503e+00 -3.053329e-02 -1.254329e-01 -1.367027e+00  1.021150e+00
3  1.262482e+00 -6.771808e-02 -2.499804e+00 -5.304315e-01  6.482574e-01
4 -1.997980e-02  1.193878e-02  7.848449e-01 -1.133542e+00 -4.635010e-02

            PC6           PC7           PC8           PC9          PC10  ...  \
0  1.206031e-17 -1.831968e-18 -1.026490e-17  2.526684e-17 -4.571110e-18  ...
1 -1.264623e-01  3.116297e-02  6.452588e-02 -2.292983e-01  2.850353e-03  ...
2 -7.758886e-02 -2.129636e-02 -3.379853e-02 -3.627640e-02 -2.709346e-02  ...
3 -1.114060e-01  4.430464e-02  1.324385e-01 -1.489014e-01 -1.596639e-02  ...
4  4.397710e-02 -1.135888e-02 -6.564122e-02  6.094574e-02 -1.112197e-02  ...

           PC12          PC13          PC14          PC15          PC16  \
0  1.501557e-17 -4.864980e-17  2.108524e-16  1.045098e-16 -1.103796e-16
1 -1.949373e-01  6.305804e-01 -9.814623e-01  5.404137e-01  1.203332e+00
2  3.607825e-02  8.407875e-02  1.123127e+00  1.054400e+00  9.965575e-01
3 -5.094151e-02 -1.653948e-02  3.106138e-01  5.202356e-01  9.906316e-01
4  2.826011e-02  2.808077e-01  7.065430e-01  8.888865e-01  1.500903e-01

           PC17          PC18          PC19          PC20  pnns_groups_1
0 -4.255193e-18  4.908850e-17 -3.991642e-17  5.929201e-18            NaN
1  2.186773e+00 -5.478065e-01 -4.178572e-02  3.526768e-06            NaN
2 -1.756399e-01  3.246077e-01 -1.368963e-02  1.879019e-06            NaN
3 -1.585226e+00  1.776872e-01 -3.475797e-02 -8.385474e-07            NaN
4  6.653560e-01 -8.894491e-01 -3.125630e-03  1.491021e-06            NaN

[5 rows x 21 columns]
```

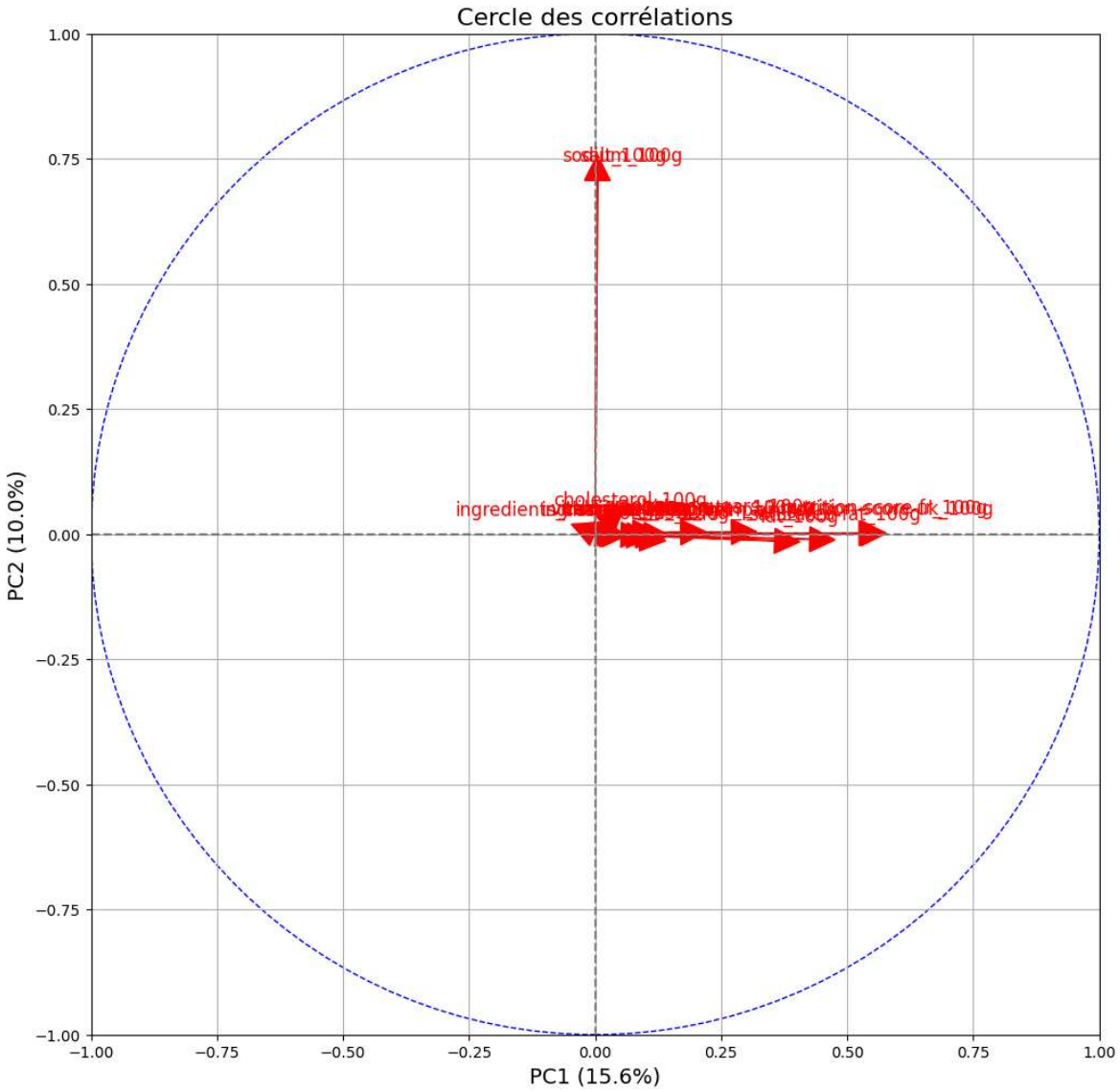
In [ ]:
```
plt.figure(figsize=(10, 5))
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='pnns_groups_1', palette='tab
plt.title('ACP des produits en fonction de pnns_groups_1')
plt.xlabel('Composante principale 1')
plt.ylabel('Composante principale 2')
```

```
plt.legend(title='PNNS Groups')
plt.show()
```



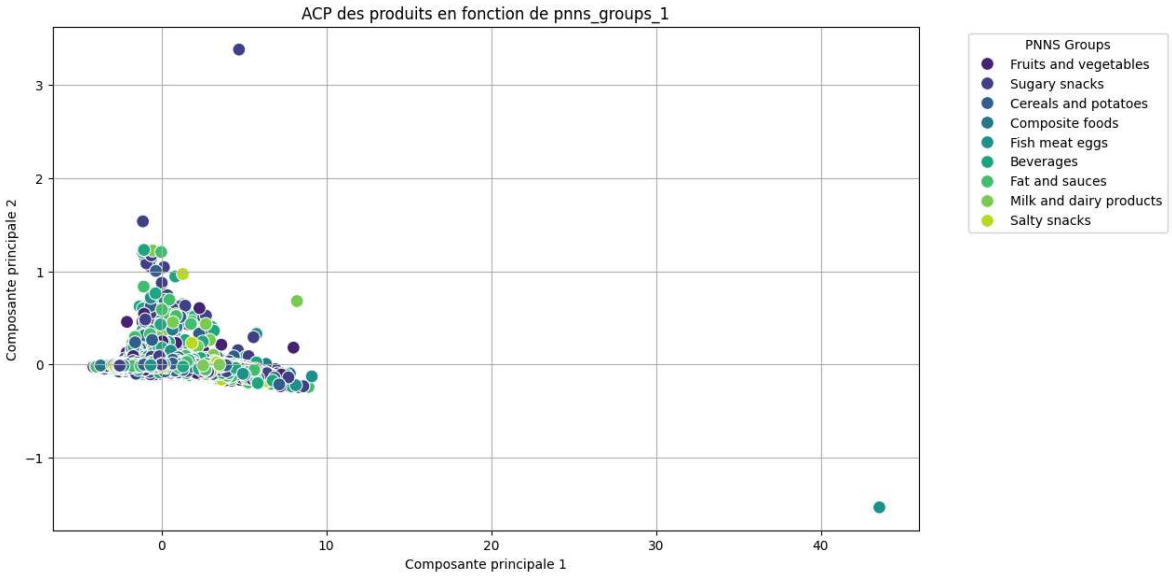
In [ ]:

```
def correlation_graph(pca, features):
    plt.figure(figsize=(12, 12))
    for i in range(pca.components_.shape[1]):
        plt.arrow(0, 0, pca.components_[0, i], pca.components_[1, i],
                  head_width=0.05, head_length=0.05, color='red')
        plt.text(pca.components_[0, i] + 0.05, pca.components_[1, i] + 0.05,
                 features[i], color='red', ha='center', va='center', fontsize=12

    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]*100:.1f}%)', fontsize=14
    plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]*100:.1f}%)', fontsize=14
    plt.grid()
    plt.axhline(0, color='grey', ls='--')
    plt.axvline(0, color='grey', ls='--')
    circle = plt.Circle((0, 0), 1, facecolor='none', edgecolor='b', ls='--')
    plt.gca().add_artist(circle)
    plt.title('Cercle des corrélations', fontsize=16)
    plt.show()

# Appel de la fonction avec les colonnes numériques de votre DataFrame
correlation_graph(pca, df_numeric.columns)
```



Pas très lisible, j'ai cherché avec chat gpt pour faire un plotly plus joli mais je n'ai pas réussi à implémenter la solution

In [ ]:
```
# Nuage de points des deux premières composantes principales
plt.figure(figsize=(12, 7))
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='pnns_groups_1', palette='vir
plt.title('ACP des produits en fonction de pnns_groups_1')
plt.xlabel('Composante principale 1')
plt.ylabel('Composante principale 2')
plt.legend(title='PNNS Groups', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
```



Il faut trouver les coupables parmi les outliers mais la dernière fois ça n'avait pas fonctionner correctement. Trouvons une méthode de filtrage un peu moins agressives.

In [ ]:
```
# Filtrer uniquement les colonnes qui se terminent par '_100g'
df_100g = sansdoublons.filter(like='_100g')

# Calculer les quantiles et l'IQR pour ces colonnes
Q1 = df_100g.quantile(0.25)
Q3 = df_100g.quantile(0.75)
IQR = Q3 - Q1

# Filtrer les outliers
df_outliers = df_100g[~((df_100g < (Q1 - 1.5 * IQR)) | (df_100g > (Q3 + 1.5 * IQ

df_outliers.shape
```

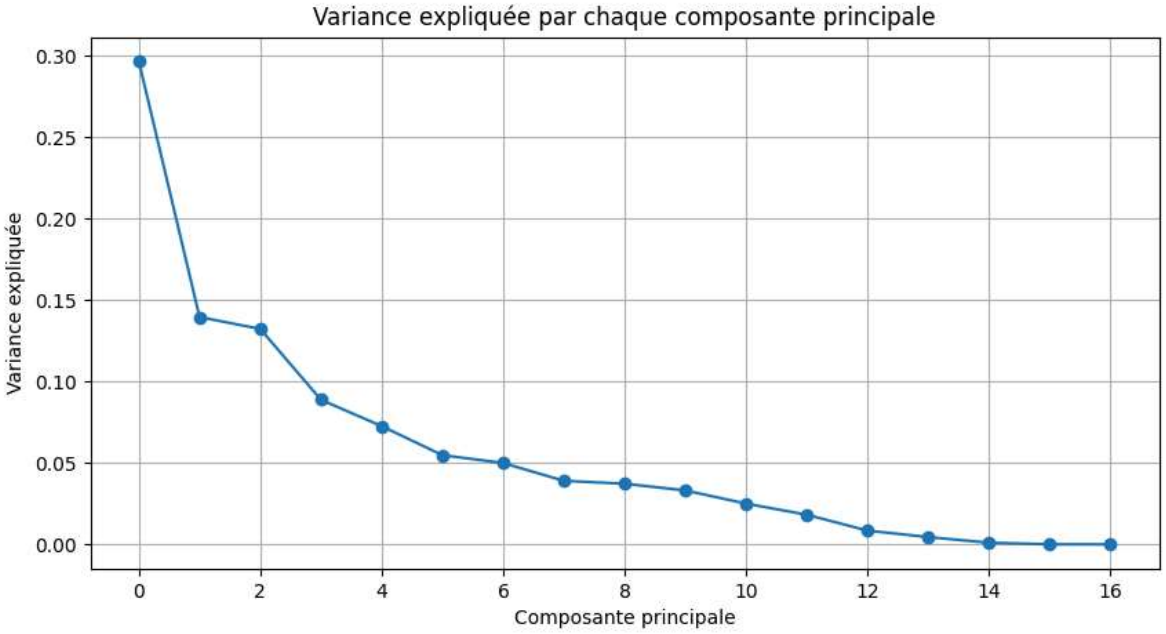
Out[ ]:
```
(211173, 17)
```

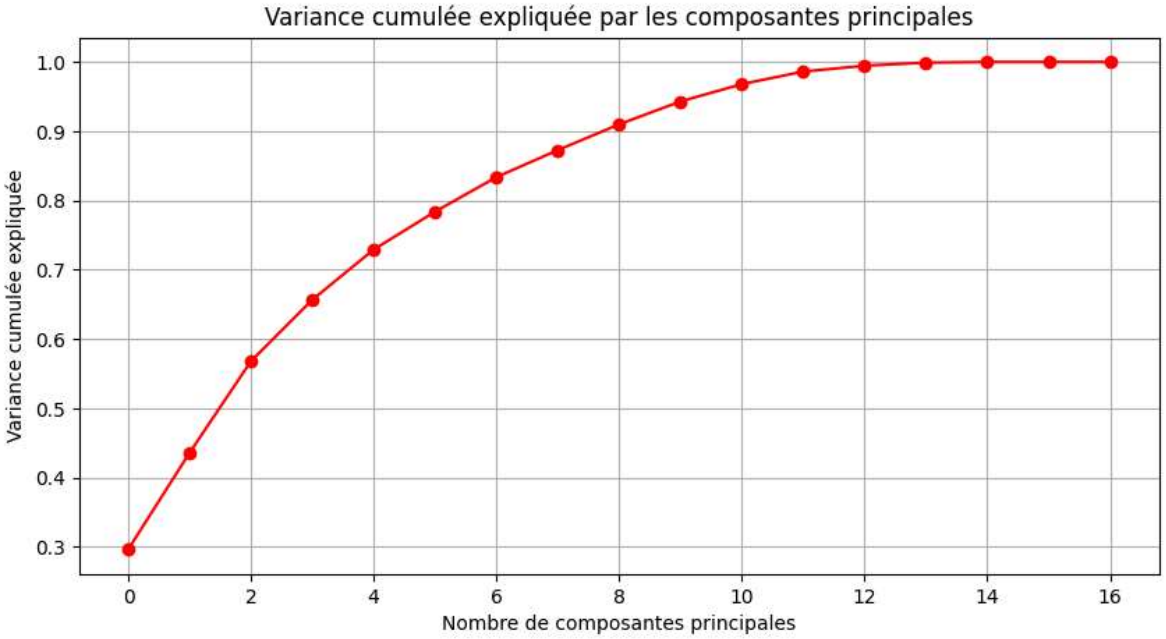
In [ ]:
```
# Sélection des colonnes numériques
df_numeric2 = df_outliers.select_dtypes(include=['float64'])

# Remplacement des NaN par la moyenne de la colonne
df_numeric2 = df_numeric2.fillna(df_numeric2.mean())

# Standardisation des données
scaler2 = StandardScaler()
df_scaled2 = scaler2.fit_transform(df_numeric2)
```

In [ ]:
```
# Création de l'objet PCA 2
pca2 = PCA()

# Ajustement de l'ACP sur les données standardisées
pca2.fit(df_scaled2)

# Création d'un DataFrame avec les composantes principales
df_pca_components2 = pd.DataFrame(pca2.components_, columns=df_numeric2.columns)
```

In [ ]:
```
plt.figure(figsize=(10, 5))
plt.plot(pca2.explained_variance_ratio_, marker='o')
plt.title('Variance expliquée par chaque composante principale')
```

```
plt.xlabel('Composante principale')
plt.ylabel('Variance expliquée')
plt.grid(True)
plt.show()

# Affichage de la variance cumulée expliquée par les composantes principales
plt.figure(figsize=(10, 5))
plt.plot(np.cumsum(pca2.explained_variance_ratio_), marker='o', color='red')
plt.title('Variance cumulée expliquée par les composantes principales')
plt.xlabel('Nombre de composantes principales')
plt.ylabel('Variance cumulée expliquée')
plt.grid(True)
plt.show()
```

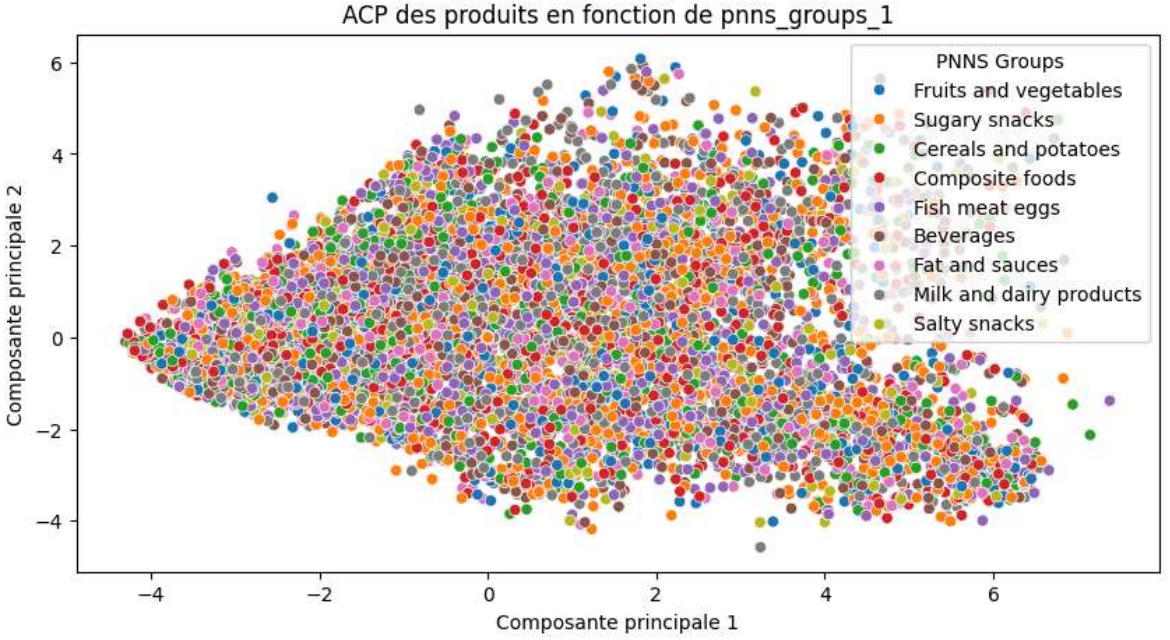




In [ ]:
```
# Résumé des composantes principales

df_pca2 = pd.DataFrame(pca2.transform(df_scaled2), columns=[f'PC{i+1}' for i in
df_pca2['pnns_groups_1'] = sansdoublons['pnns_groups_1']
print(df_pca2.head())

plt.figure(figsize=(10, 5))
```

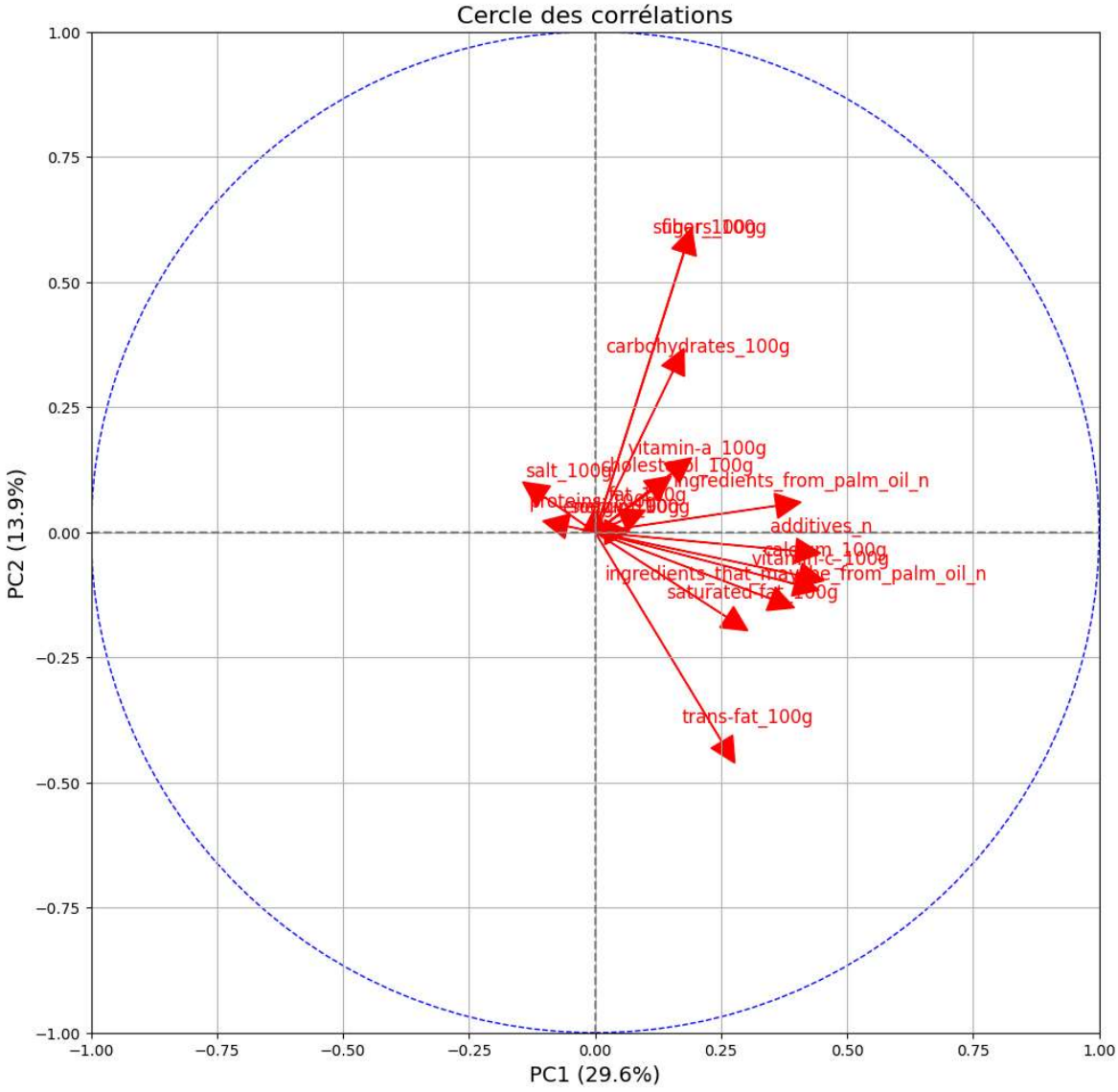
```
sns.scatterplot(data=df_pca2, x='PC1', y='PC2', hue='pnns_groups_1', palette='ta
plt.title('ACP des produits en fonction de pnns_groups_1')
plt.xlabel('Composante principale 1')
plt.ylabel('Composante principale 2')
plt.legend(title='PNNS Groups')
plt.show()
```

```
            PC1           PC2           PC3           PC4           PC5  \
0  1.090646e-16 -3.929691e-17  2.145480e-17  1.925226e-16  8.316210e-18
1  1.184380e+00  7.981989e-01  4.273103e+00  1.343314e+00 -2.554863e-01
2  1.025764e+00 -4.268131e-02  2.095380e+00  3.915615e-01  8.344684e-01
3  2.389097e+00  1.019040e+00  4.024694e+00  1.597253e+00  1.316441e+00
4  9.210240e-01 -8.312783e-02  8.609015e-01  1.005976e-02  1.245200e+00

            PC6           PC7           PC8           PC9           PC10  \
0  2.519255e-17  3.255198e-17 -7.626448e-17  2.302088e-17 -3.823516e-19
1 -1.022058e-01 -7.461429e-01 -1.891992e+00  1.370638e+00  2.599151e-01
2  9.443791e-01 -1.011488e+00 -5.428544e-01  8.325615e-01  6.915393e-01
3 -1.818319e-01 -7.204556e-01  1.118213e+00  1.345028e-01  7.434203e-01
4  5.129629e-01 -1.313125e+00 -1.138607e+00 -8.479621e-01 -2.606522e-01

            PC11          PC12          PC13          PC14          PC15  \
0  9.933307e-17 -1.207655e-16  1.414996e-17  8.630525e-17  5.422428e-18
1  7.337342e-01 -7.787150e-01 -1.144388e-01 -9.764805e-02 -3.975781e-02
2  2.957623e-01  1.341594e+00 -4.857821e-01 -1.213468e-01 -8.170092e-03
3  1.400515e-01 -3.419989e-01 -7.900443e-01 -1.742442e-01  5.579284e-02
4 -6.540934e-01  7.446451e-01  3.379392e-02 -4.576806e-01 -2.488741e-02

            PC16          PC17 pnns_groups_1
0  1.263490e-16 -7.993878e-29           NaN
1  2.692686e-04 -2.072819e-16           NaN
2  1.182574e-05  9.107272e-17           NaN
3 -3.700921e-04  4.789062e-16           NaN
4  3.685435e-04 -1.478651e-15           NaN
```



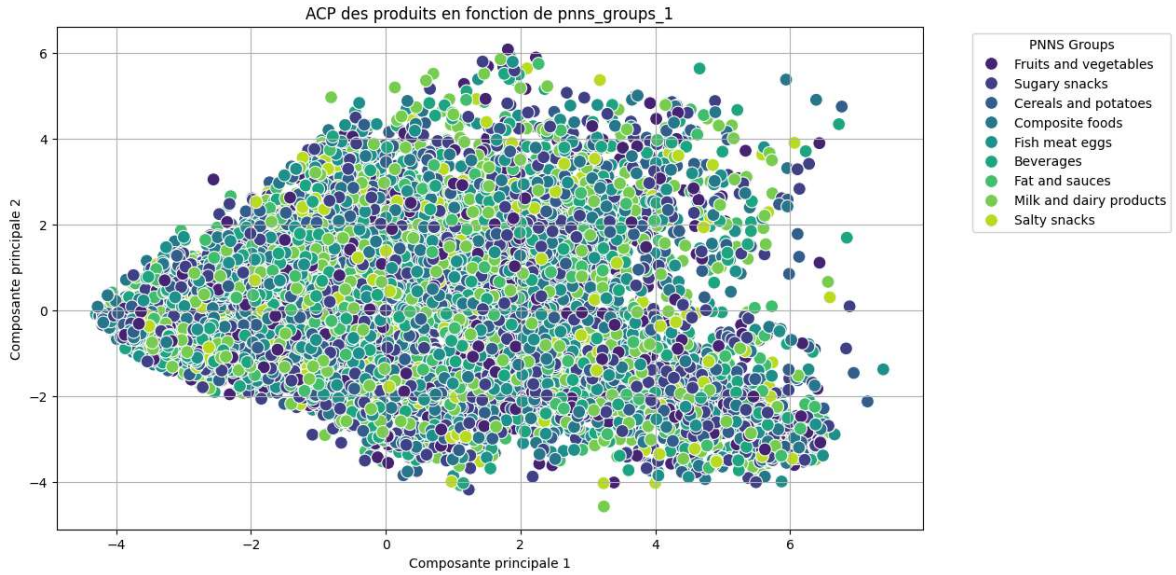
In [ ]:
```
# Graphique des corrélations

def correlation_graph(pca2, features):
    plt.figure(figsize=(12, 12))
    for i in range(pca2.components_.shape[1]):
        plt.arrow(0, 0, pca2.components_[0, i], pca2.components_[1, i],
```

```
                    head_width=0.05, head_length=0.05, color='red')
            plt.text(pca2.components_[0, i] + 0.05, pca2.components_[1, i] + 0.05,
                     features[i], color='red', ha='center', va='center', fontsize=12

    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    plt.xlabel(f'PC1 ({pca2.explained_variance_ratio_[0]*100:.1f}%)', fontsize=1
    plt.ylabel(f'PC2 ({pca2.explained_variance_ratio_[1]*100:.1f}%)', fontsize=1
    plt.grid()
    plt.axhline(0, color='grey', ls='--')
    plt.axvline(0, color='grey', ls='--')
    circle = plt.Circle((0, 0), 1, facecolor='none', edgecolor='b', ls='--')
    plt.gca().add_artist(circle)
    plt.title('Cercle des corrélations', fontsize=16)
    plt.show()

# Appel de la fonction avec les colonnes numériques de votre DataFrame
correlation_graph(pca2, df_numeric.columns)
import plotly.express as px
```



Bon pour ce qui est de la première ACP, on voit déjà que le cercle de corrélation est plus lisible, et on peut en faire quelques analyses. En revanche le nuage de points est imbuvable. IL faudra peut être songer à réaliser un tri. C'est dur d'envisager un regroupement.

In [ ]:
```
# Nuage de points des deux premières composantes principales
plt.figure(figsize=(12, 7))
sns.scatterplot(data=df_pca2, x='PC1', y='PC2', hue='pnns_groups_1', palette='vi
plt.title('ACP des produits en fonction de pnns_groups_1')
plt.xlabel('Composante principale 1')
plt.ylabel('Composante principale 2')
plt.legend(title='PNNS Groups', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
```



In [ ]:
```
import plotly.express as px
# nb format version
fig = px.scatter(df_sans_outliers, x='sugars_100g', y = 'carbohydrates_100g',
                 labels={'value':'sugars','variable': 'Condition'})
fig
fig.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[38], line 6
      3 fig = px.scatter(df_sans_outliers, x='sugars_100g', y = 'carbohydrates_10
0g',
      4                  labels={'value':'sugars','variable': 'Condition'})
      5 fig
----> 6 fig.show()

File ~\AppData\Roaming\Python\Python312\site-packages\plotly\basedatatypes.py:341
0, in BaseFigure.show(self, *args, **kwargs)
   3377 """
   3378 Show a figure using either the default renderer(s) or the renderer(s)
   3379 specified by the renderer argument
   (...)
   3406 None
   3407 """
   3408 import plotly.io as pio
-> 3410 return pio.show(self, *args, **kwargs)

File ~\AppData\Roaming\Python\Python312\site-packages\plotly\io\_renderers.py:39
4, in show(fig, renderer, validate, **kwargs)
    389         raise ValueError(
    390             "Mime type rendering requires ipython but it is not installe
d"
    391         )
    393     if not nbformat or Version(nbformat.__version__) < Version("4.2.0"):
--> 394         raise ValueError(
    395             "Mime type rendering requires nbformat>=4.2.0 but it is not i
nstalled"
    396         )
    398     ipython_display.display(bundle, raw=True)
    400 # external renderers

ValueError: Mime type rendering requires nbformat>=4.2.0 but it is not installed
```

In [ ]:

Out[ ]:

	energy_100g	fat_100g	saturated-fat_100g	trans-fat_100g	cholesterol_100g	carboh
count	151873.000000	139295.000000	133032.000000	69501.0	70192.000000	1
mean	932.764148	8.613637	3.212394	0.0	0.007565	
std	706.819007	10.883736	4.310744	0.0	0.013160	
min	0.000000	0.000000	0.000000	0.0	0.000000	
25%	290.000000	0.100000	0.000000	0.0	0.000000	
50%	810.000000	3.400000	1.200000	0.0	0.000000	
75%	1523.000000	14.145000	5.000000	0.0	0.011000	
max	3619.000000	50.000000	17.840000	0.0	0.050000	