Projet 3 : Concevez une application au service de la santé publique

Sommaire

1. Idée d’application
2. Opérations de nettoyage
3. Analyse exploratoire
4. Analyse univariée/ multivariée
5. PCA
6. ANOVA
7. Calcul automatique de nutri score par régression linéaire
8. Synthèse

1. Idée d’application

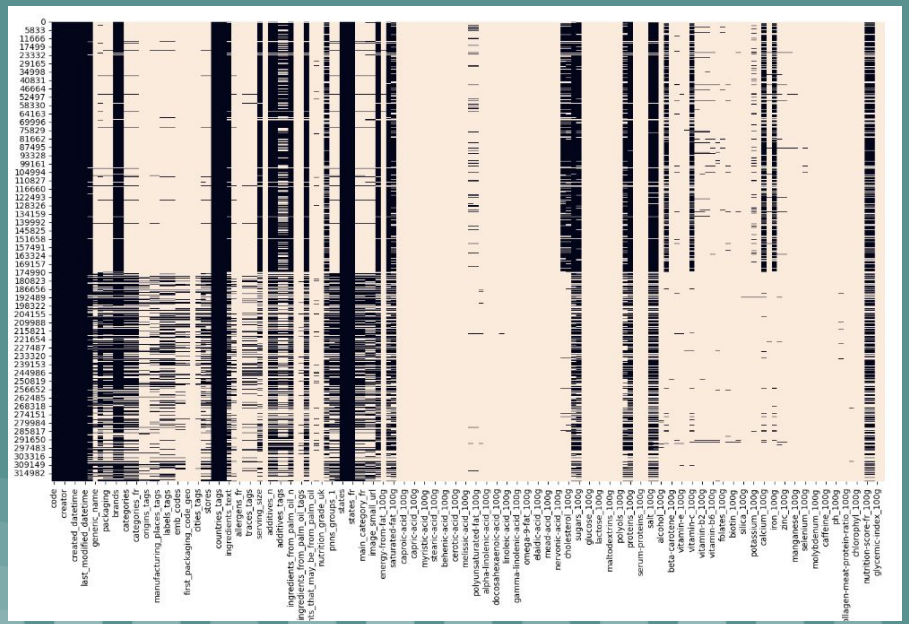
Proposer une application pour exploiter les données d'Open Food Facts en identifiant les caractéristiques nutritionnelles des produits ayant des nutriscores faibles et en étiquetant automatiquement les produits ayant des caractéristiques similaires.

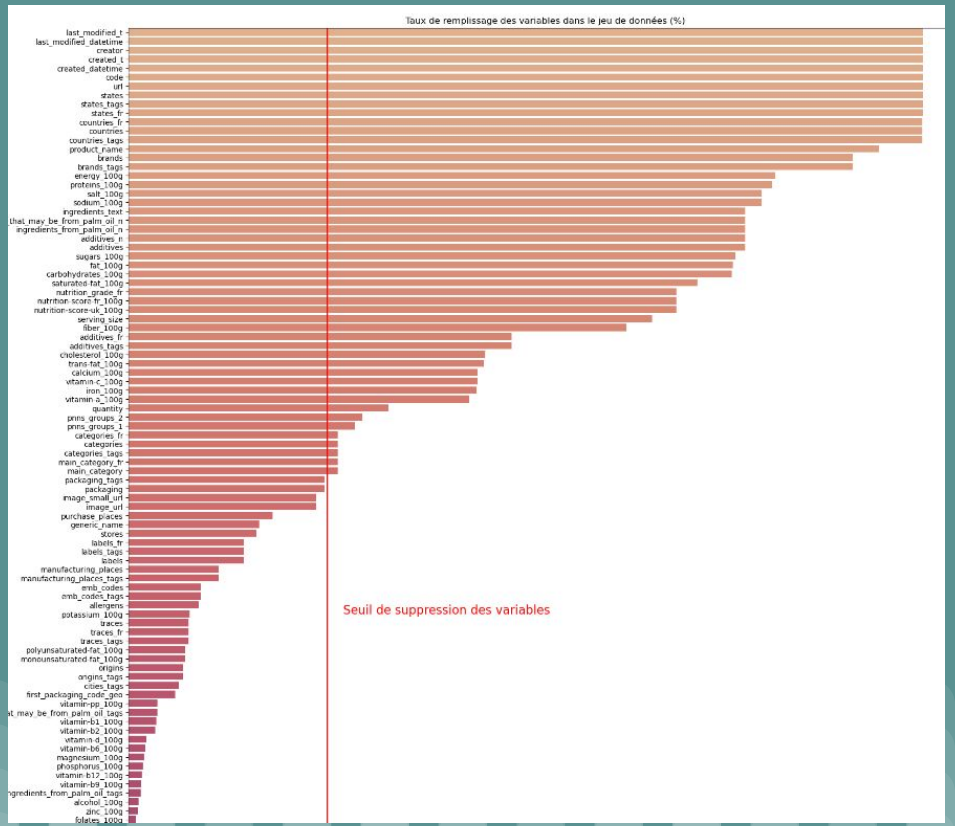
l'application pourrait également inclure le calcul automatique du nutriscore et un indicateur pour le nombre d'additifs dans les produits afin de vérifier si ces deux variables sont liées au nutriscore.

2. Opérations de nettoyage

```
print ("Le dataset compte {} lignes et {} variables"
```

```
Le dataset compte 320772 lignes et 162 variables
```





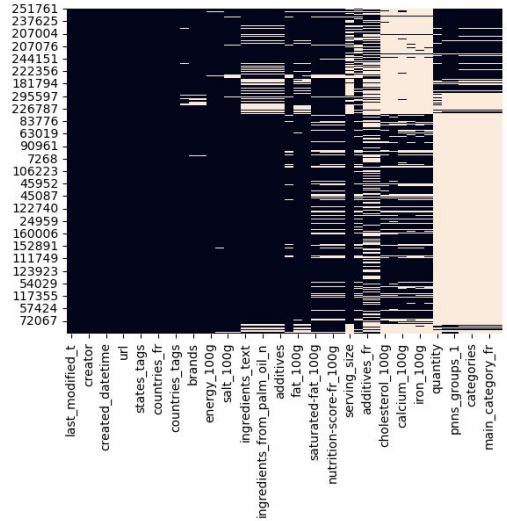
```
#Liste des variables à conserver
features_to_conserve = list(filling_features.loc[filling_features['Taux_de_Null']>=sup_threshold, 'Variable'].values)
#Liste des variables supprimées
deleted_features = list(filling_features.loc[filling_features['Taux_de_Null']<sup_threshold, 'Variable'].values)

#Nouveau Dataset avec les variables conservées
datas = datas[features_to_conserve].sort_values(["created_datetime","last_modified_datetime"], ascending=True)
datas.sample(5)
```

```
datas = datas[df_subset_nutriments.notnull().any(axis=1)]
datas.shape
```

```
(262833, 50)
```

```
sns.heatmap(datas.isnull(), cbar=False)
```

```
<AxesSubplot:>
```



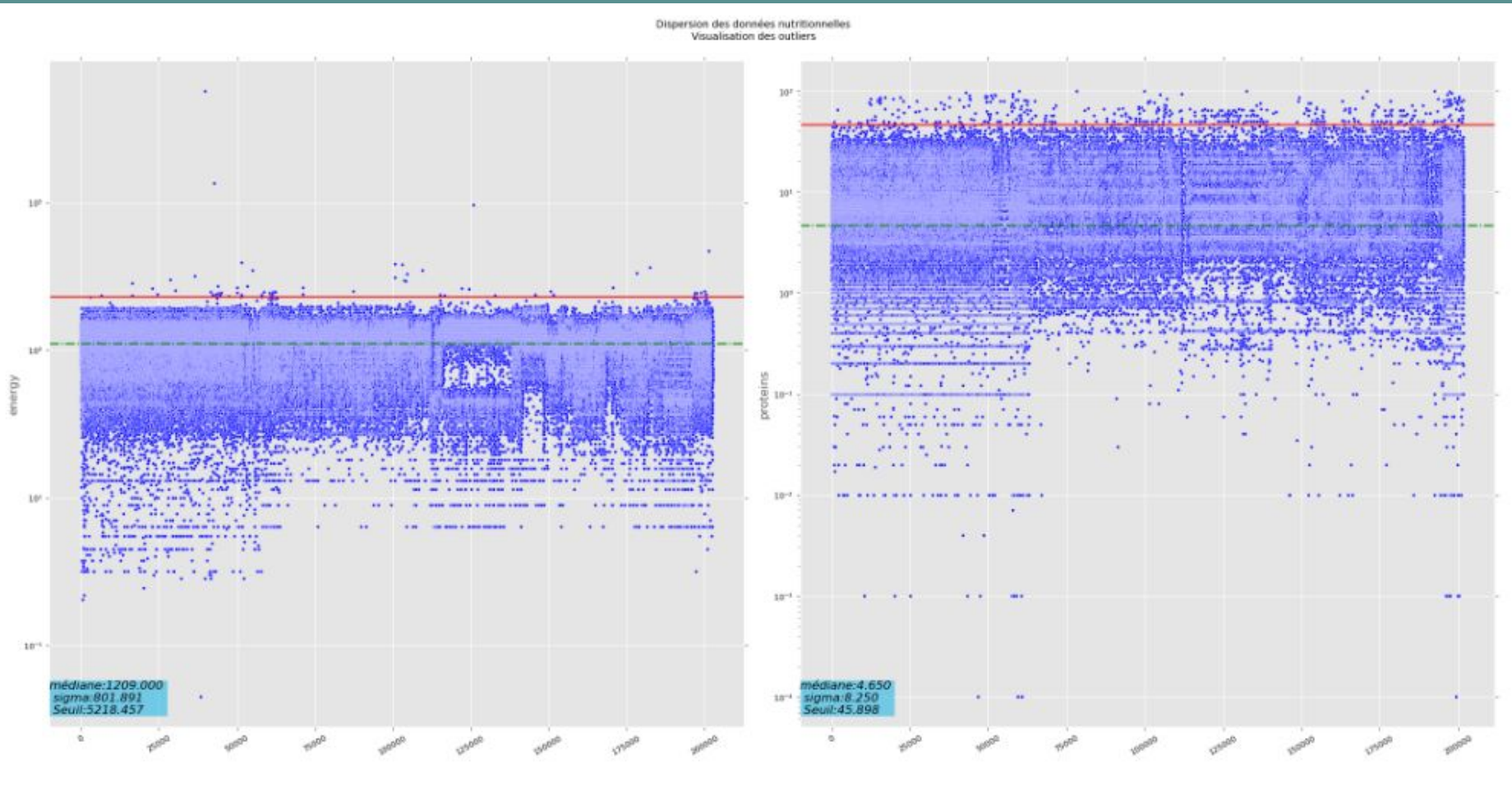

```
#On repère les numerical_features
numerical_features = list(datas_cleaned.select_dtypes(include=["float64","int64"]).columns)
#On supprime les nutriscores qui eux peuvent être négatifs
#numerical_features.remove('nutriscore_score')
numerical_features.remove('nutrition-score-fr_100g')
numerical_features.remove('ingredients_that_may_be_from_palm_oil_n')
numerical_features.remove('ingredients_from_palm_oil_n')
```

```
#On supprime les lignes dont toutes les numerical_features sont à 0 ou nulles
datas_cleaned = datas_cleaned.loc[~((datas_cleaned[numerical_features]==0) | (datas_cleaned[numerical_features].isnull())).all(a
```

```
#On supprime les lignes contenant des valeurs négatives et des max aberrants
datas_cleaned = datas_cleaned[~(datas_cleaned[numerical_features] < 0).any(axis=1)]
datas_cleaned = datas_cleaned[~(datas_cleaned[numerical_features].isin([999999,9999999])).any(axis=1)]
```

```
datas_cleaned.shape
```

```
(204636, 45)
```

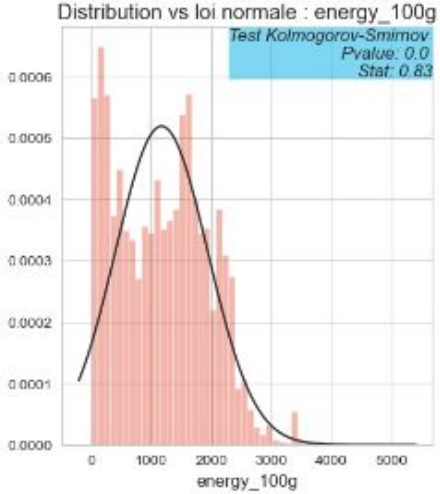

Dispersion des données nutritionnelles
Visualisation des outliers
energy
médiane:1209.000
sigma:801.891
Seuil:5218.457
proteins
médiane:4.650
sigma:8.250
Seuil:45.898

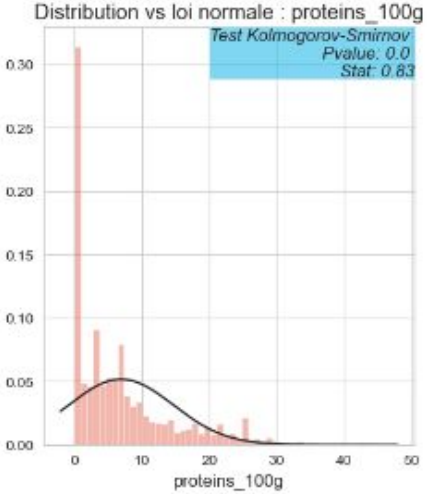
```python
for i in range(len(sigma_features)):
    col = sigma_features[i]
    threshold = (median[i] + 5*sigma[i])
    print('{:40}: suppression de la ligne si valeur > {}'.format(col, round(threshold,3)))
    mask = datas_cleaned[col] > threshold
    datas_cleaned = datas_cleaned.drop(datas_cleaned[mask].index)
```

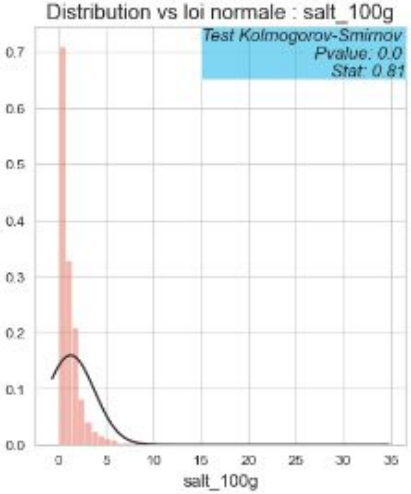
```
energy_100g                              : suppression de la ligne si valeur > 5218.457
proteins_100g                            : suppression de la ligne si valeur > 45.898
salt_100g                                : suppression de la ligne si valeur > 34.186
sodium_100g                              : suppression de la ligne si valeur > 13.46
ingredients_that_may_be_from_palm_oil_n : suppression de la ligne si valeur > 1.397
ingredients_from_palm_oil_n              : suppression de la ligne si valeur > 0.746
additives_n                              : suppression de la ligne si valeur > 13.884
sugars_100g                              : suppression de la ligne si valeur > 118.113
fat_100g                                 : suppression de la ligne si valeur > 98.664
carbohydrates_100g                       : suppression de la ligne si valeur > 170.28
saturated-fat_100g                       : suppression de la ligne si valeur > 44.614
nutrition-score-fr_100g                  : suppression de la ligne si valeur > 50.811
nutrition-score-uk_100g                  : suppression de la ligne si valeur > 51.436
fiber_100g                               : suppression de la ligne si valeur > 20.742
cholesterol_100g                         : suppression de la ligne si valeur > 0.189
trans-fat_100g                           : suppression de la ligne si valeur > 3.566
calcium_100g                             : suppression de la ligne si valeur > 1.108
vitamin-c_100g                           : suppression de la ligne si valeur > 0.494
iron_100g                                : suppression de la ligne si valeur > 0.019
vitamin-a_100g                           : suppression de la ligne si valeur > 0.003
```

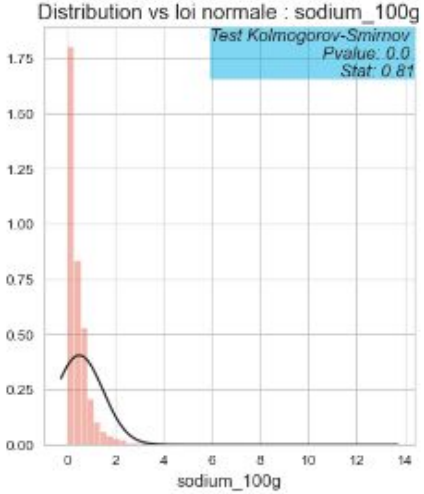
```python
datas_cleaned.shape
```

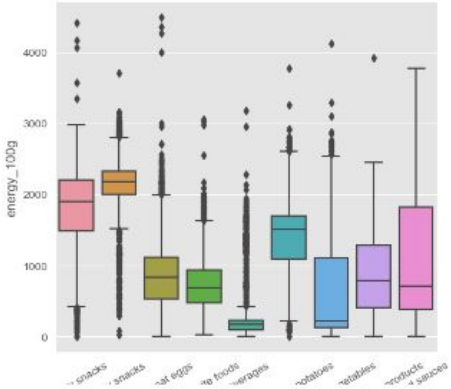
```
(189595, 45)
```

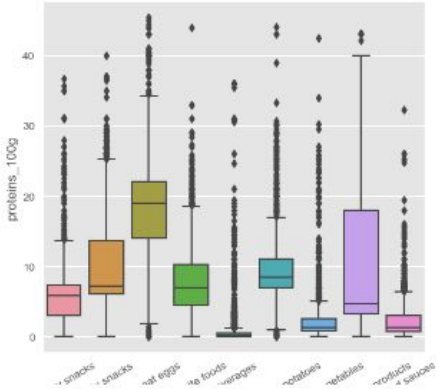

Distribution vs loi normale : energy_100g
Test Kolmogorov-Smirnov
Pvalue: 0.0
Stat: 0.83
energy_100g

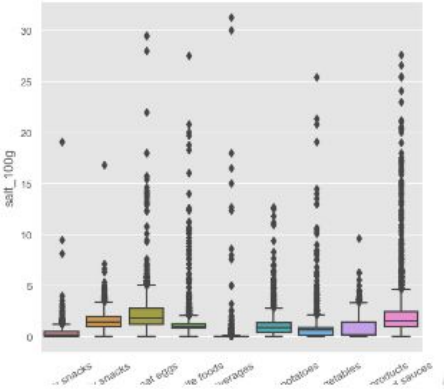

Distribution vs loi normale : proteins_100g
Test Kolmogorov-Smirnov
Pvalue: 0.0
Stat: 0.83
proteins_100g


Distribution vs loi normale : salt_100g
Test Kolmogorov-Smirnov
Pvalue: 0.0
Stat: 0.81
salt_100g


Distribution vs loi normale : sodium_100g
Test Kolmogorov-Smirnov
Pvalue: 0.0
Stat: 0.81
sodium_100g


energy_100g


proteins_100g


salt_100g

En se basant sur les projections obtenus et les résultats des tests de Kolmogorov-Smirnov (Pvalue < au niveau de test de 5%) on rejette donc l'hypothèse de normalité des distributions de ces variables. Il serait donc inexacte d'imputer les valeurs manquantes par la moyenne.

Pour confirmer cette approche, regardons à présent quelque unes de ces distributions en fonction de la catégorie pnns_groups_1 :

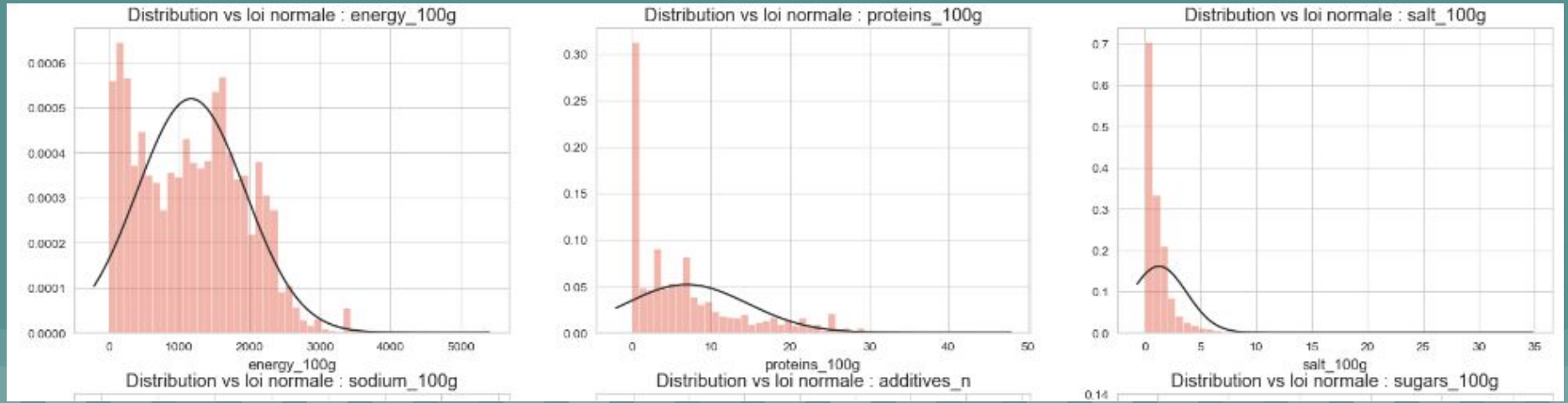
Pour ces valeurs nulles ci-dessus, les variables serving_size et additives_n sont très peu renseignées, nous allons donc les supprimer de notre jeu de données. fiber_100g est également mal renseigné mais nous en aurons besoin pour la suite. Nous allons donc compléter les valeurs nulles par la médiane de la catégorie pnns_groups_2. Enfin, pour les autres variables, avec peu de null et dont les distributions ne suivent pas la loi gaussiene, nous allons imputer avec l'algorithme des K Nearest Neighbours (KNN).

```
# KNN pour les autres variables
from sklearn.impute import KNNImputer

# On entraine le modèle d'imputation sur un échantillon de données
knn_features = ['energy_100g','proteins_100g','saturated-fat_100g','sugars_100g','salt_100g']
sample_datas = datas_cleaned[knn_features].sample(frac=0.25, random_state=1)
imputer = KNNImputer(n_neighbors=5, missing_values=np.nan)
imputer.fit(sample_datas)
```

```
KNNImputer()
```

```
# Puis on applique le modèle sur l'ensemble des données
datas_imputed = imputer.transform(datas_cleaned[knn_features])
df_datas_imputed = pd.DataFrame(datas_imputed, columns=knn_features)
```

```
for col_knn in knn_features:
    datas_cleaned[col_knn] = df_datas_imputed[col_knn].values
```



3. Analyse exploratoire
Connaissance des données

Connaissance des données

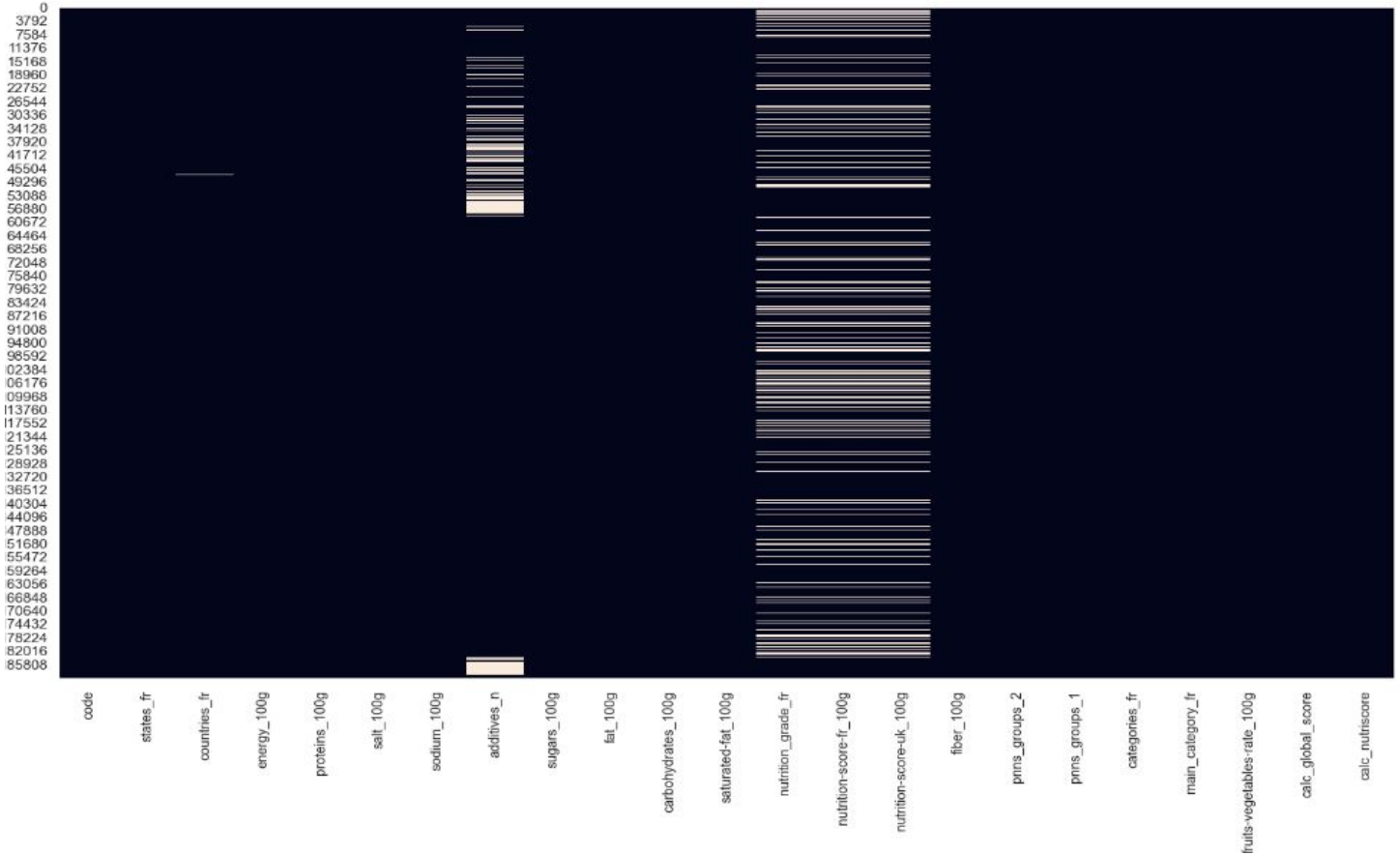
```
Entrée [136]: data_ana.shape

Out[136]: (189590, 23)
```

```
Entrée [137]: data_ana.isna().sum()

Out[137]: code                            1
          states_fr                       1
          countries_fr                   49
          energy_100g                     0
          proteins_100g                   0
          salt_100g                       0
          sodium_100g                     0
          additives_n                 20555
          sugars_100g                     0
          fat_100g                        0
          carbohydrates_100g              0
          saturated-fat_100g              0
          nutrition_grade_fr          32000
          nutrition-score-fr_100g     32000
          nutrition-score-uk_100g     32000
          fiber_100g                      0
          pnns_groups_2                   0
          pnns_groups_1                   0
          categories_fr                   0
          main_category_fr                0
          fruits-vegetables-rate_100g     0
```

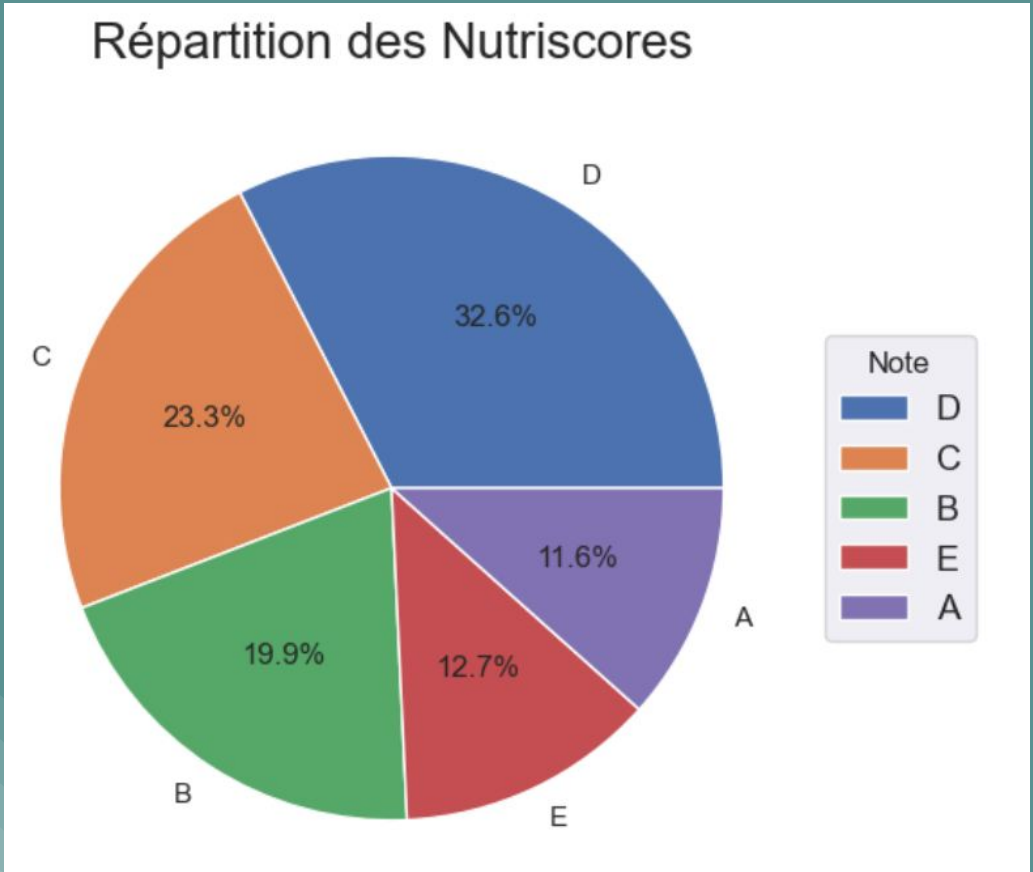
```
Entrée [134]: data_ana.columns.sort_values().to_list()

Out[134]: ['additives_n',
           'calc_global_score',
           'calc_nutriscore',
           'carbohydrates_100g',
           'categories_fr',
           'code',
           'countries_fr',
           'energy_100g',
           'fat_100g',
           'fiber_100g',
           'fruits-vegetables-rate_100g',
           'main_category_fr',
           'nutrition-score-fr_100g',
           'nutrition-score-uk_100g',
           'nutrition_grade_fr',
           'pnns_groups_1',
           'pnns_groups_2',
           'proteins_100g',
           'salt_100g',
           'saturated-fat_100g',
           'sodium_100g',
           'states_fr',
           'sugars_100g']
```

Connaissance des données

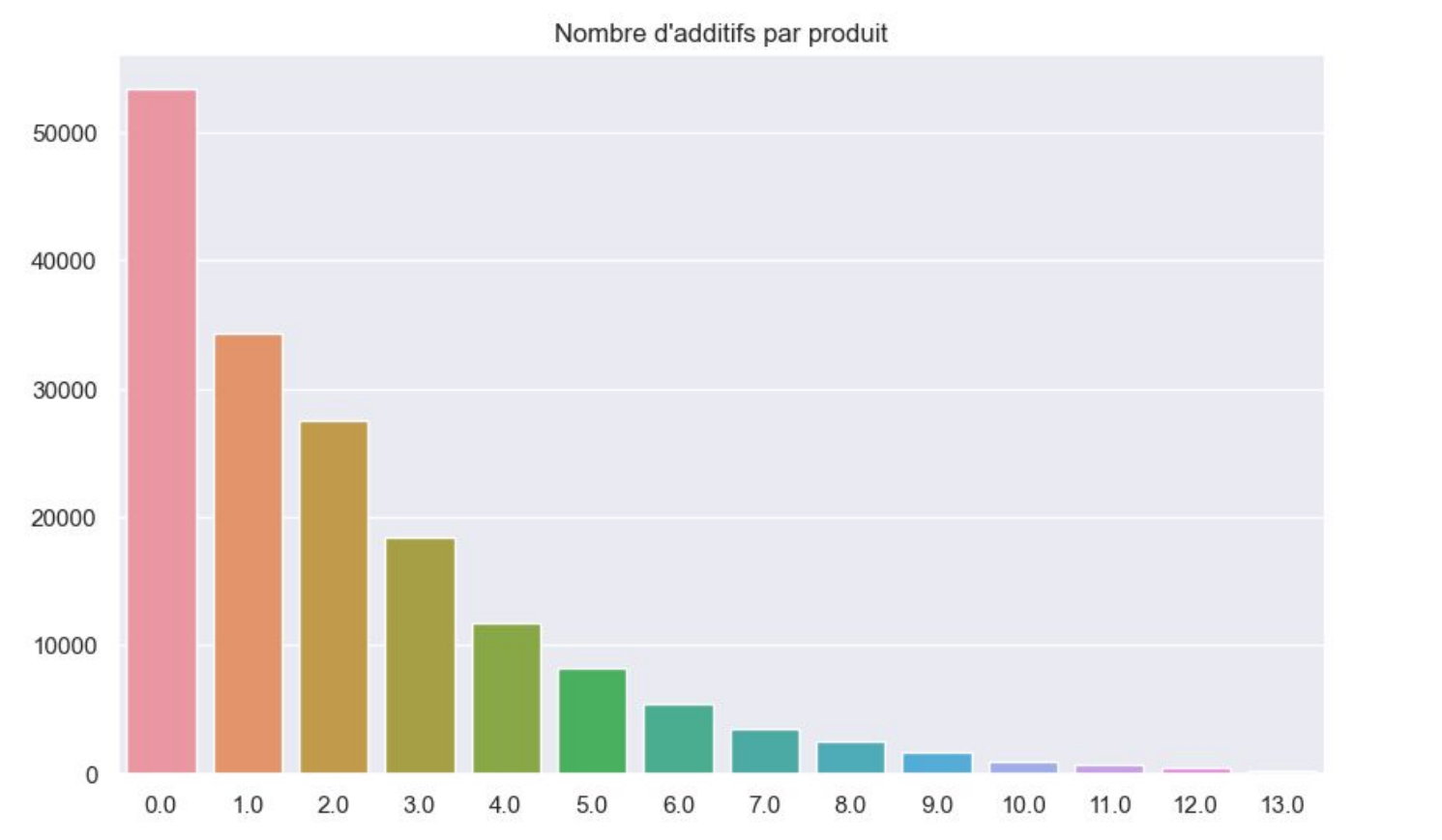
```
Entrée [138]: plt.figure(figsize=(18,10))
              sns.heatmap(data_ana.isnull(), cbar=False)
```



Connaissance des données: Répartition des Nutriscores



Connaissance des données: Additifs

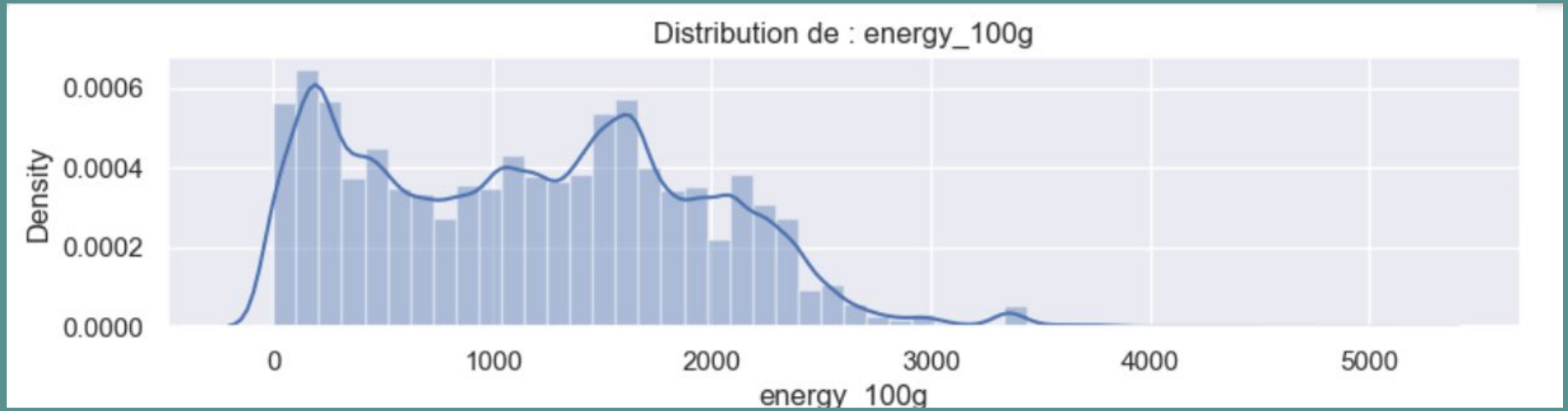


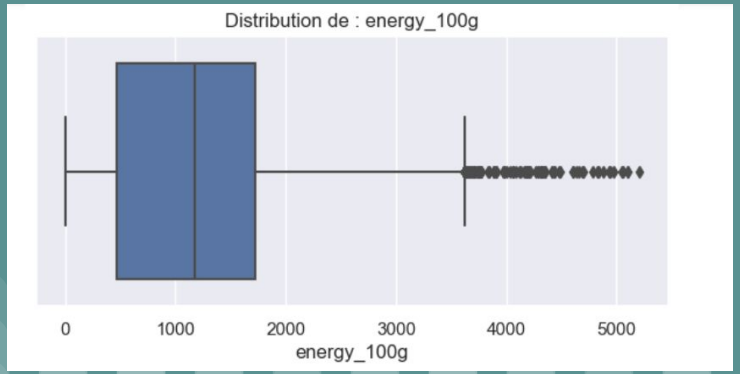
Connaissance des données:

Métriques des données numériques

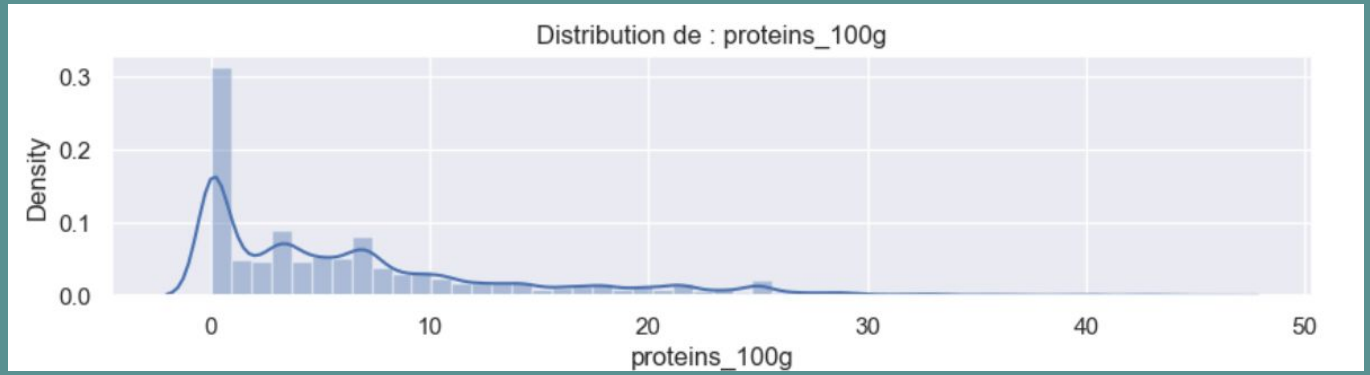
	energy_100g	fat_100g	saturated-fat_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g	sodium_100g	nu score-
count	189590.000000	189590.000000	189590.000000	189590.000000	189590.000000	189590.000000	189590.000000	189590.000000	189590.000000	157590.
mean	1173.558648	13.155236	5.156561	32.489628	17.769646	2.003084	6.901639	1.278906	0.496820	11.
std	767.439249	15.922208	6.442385	28.305708	21.913568	2.893907	7.710484	2.477649	0.972316	7.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-2.
25%	467.000000	0.100000	0.000000	6.670000	1.700000	0.000000	0.500000	0.100000	0.039370	4.
50%	1179.000000	7.320000	2.830000	26.400000	7.500000	1.000000	4.600000	0.741680	0.288000	12.
75%	1728.000000	21.400000	7.200000	57.140000	27.600000	2.500000	10.000000	1.541780	0.600000	17.
max	5205.000000	98.500000	44.440000	100.000000	100.000000	20.700000	45.830000	34.150000	13.444882	40.

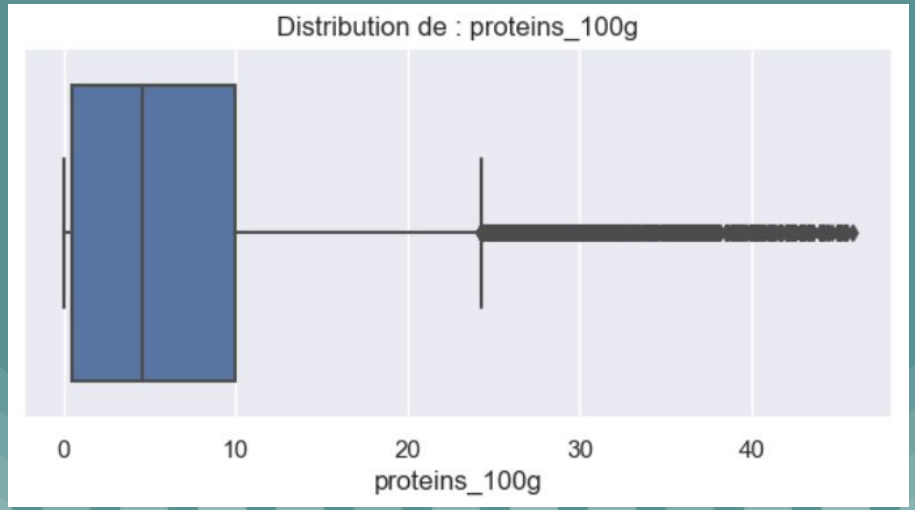
Connaissance des données: Distribution



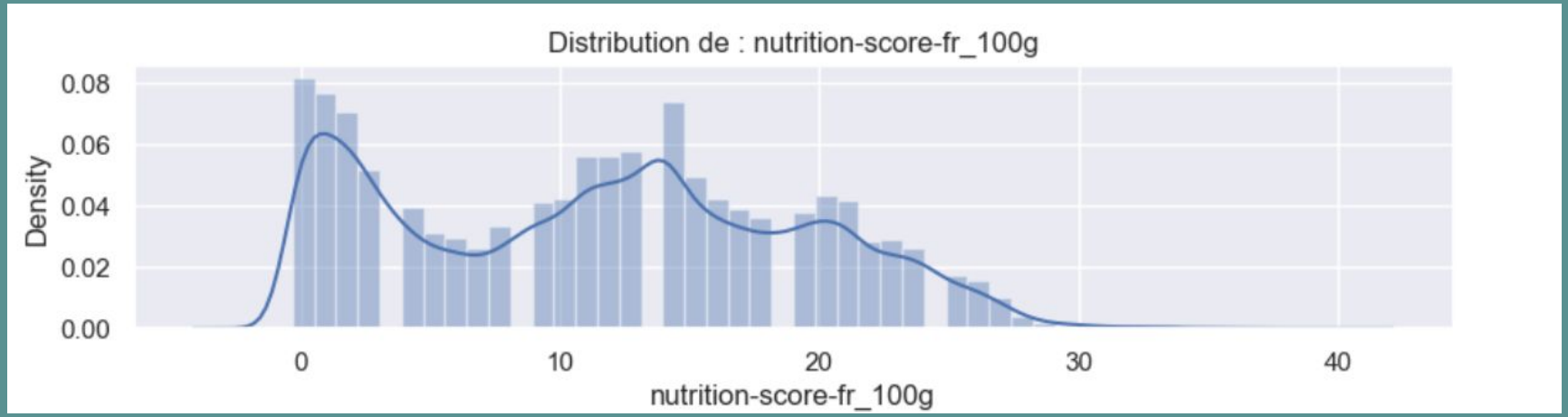


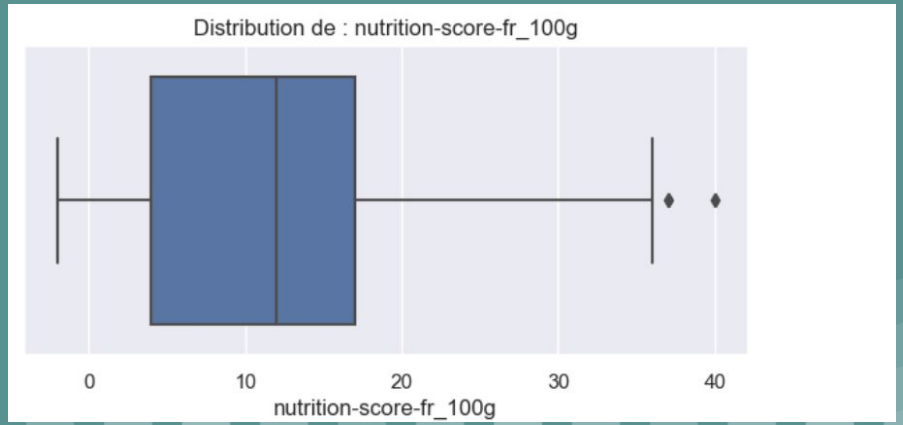
Connaissance des données: Distribution



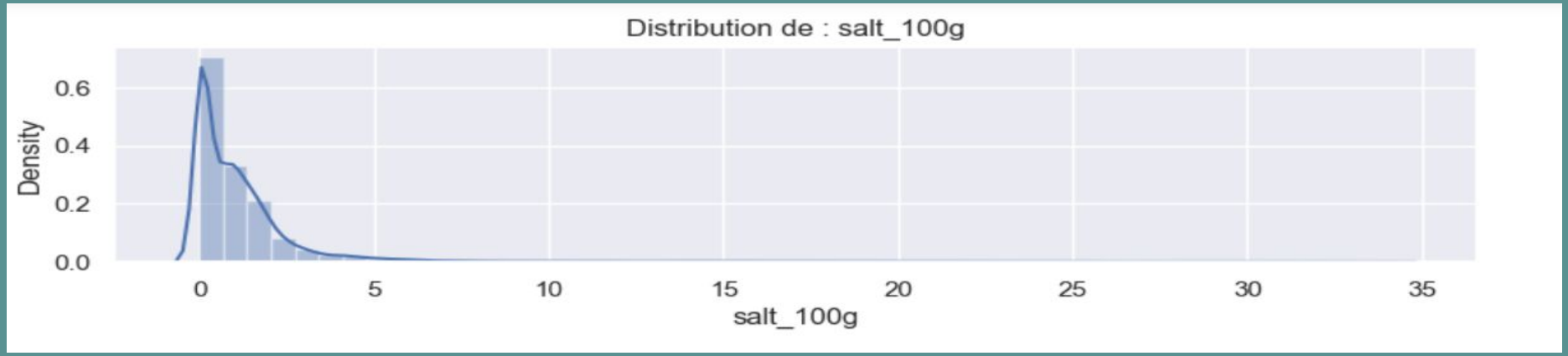


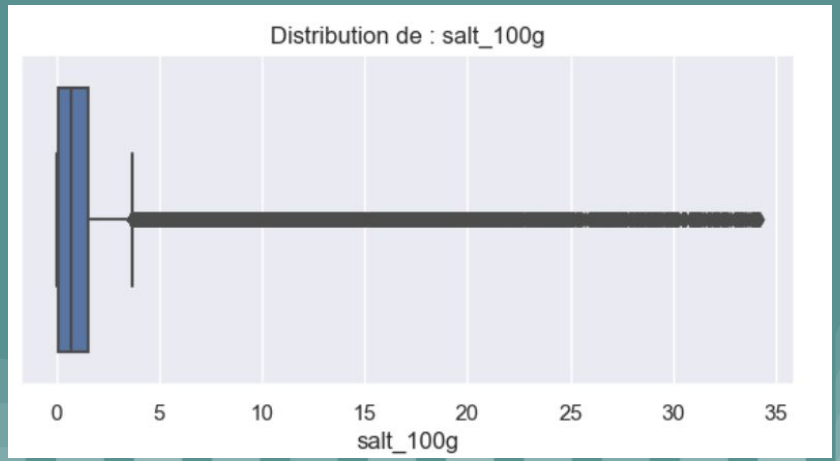
Connaissance des données: Distribution





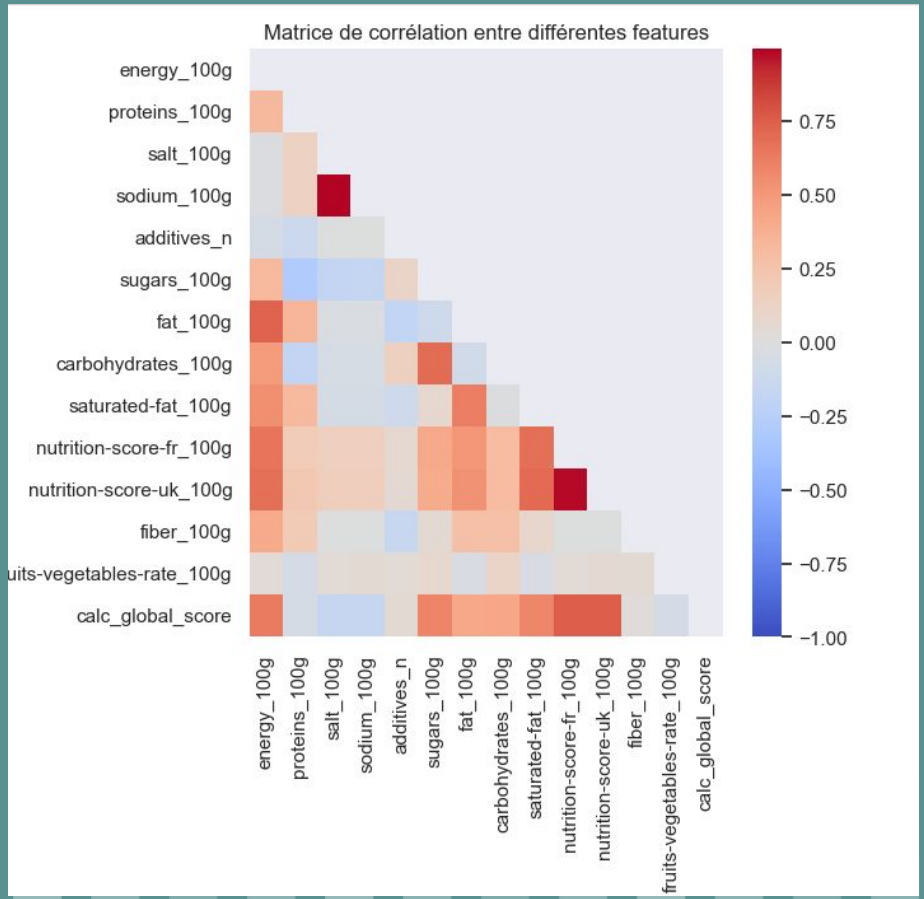
Connaissance des données: Distribution





4. Analyse multivariée

4. Analyse multivariée: corrélations



Analyse du tableau :

- additives_n : pas de correlation remarquable
- additives_tags : pas de correlation remarquable

- energy_100g : forte corrélation avec:
  - fat_100g
  - saturated-fat_100g
  - carbohydrates_100g
  - nutrition-score-fr_100g
- fat_100g et saturated-fat_100g fortement corrélés
- nutrition-score-uk_100g : ont corrélation avec:
  - saturated_fat_100g
  - fiber_100g
- fiber_100g et trans-fat_100g ont corrélation

4. Analyse multivariée: indépendance des variables Test du CHI 2 :

Application du Test du CHI 2

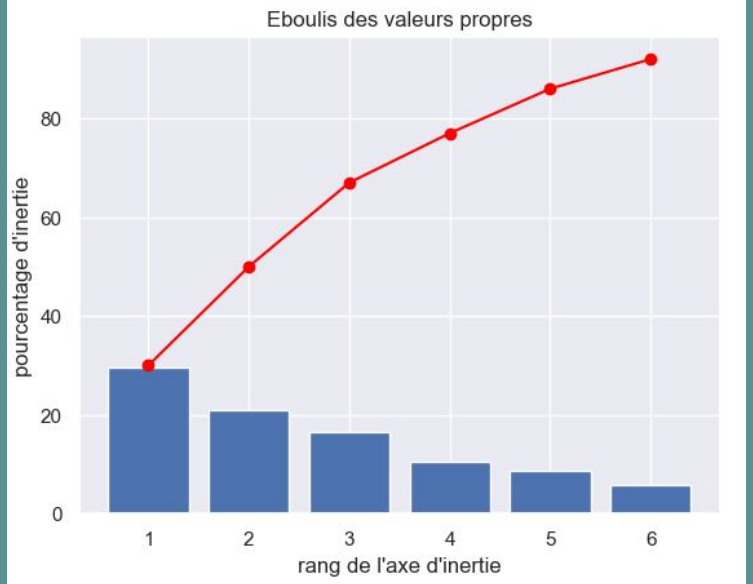
Conclusions

```
import seaborn as sns
x = pd.cut(data_ana['nutrition-score-fr_100g'], 20)
for column in data_ana.select_dtypes(include = ['int32','float64','int32']).columns:
    print('test d\'indépendance {} / {}'.format('nutriscore',column))
    if data_ana[column].nunique() > 20 :
        y = pd.cut(data_ana[column], 20).astype('category')
    else:
        y = data_ana[column].astype('category')
    test_chi2(x, y)

    print('_____')
```

```
test d'indépendance nutriscore / energy_100g
chi2 : 123795.14241,
p : 123795.14241,
dof : 123795.14241

Variables non indépendantes (H0 rejetée) car p = 0.0 <= alpha = 0.03
_____

test d'indépendance nutriscore / proteins_100g
chi2 : 29527.19849,
p : 29527.19849,
dof : 29527.19849

Variables non indépendantes (H0 rejetée) car p = 0.0 <= alpha = 0.03
_____

test d'indépendance nutriscore / salt_100g
chi2 : 18322.84963,
p : 18322.84963,
dof : 18322.84963

Variables non indépendantes (H0 rejetée) car p = 0.0 <= alpha = 0.03
```

5. PCA Réduction de dimension

```
plt.bar(x_list, scree)
plt.plot(x_list, scree_cum,c="red",marker='o')
plt.xlabel("rang de l'axe d'inertie")
plt.ylabel("pourcentage d'inertie")
plt.title("Eboulis des valeurs propres")
plt.show(block=False)
```



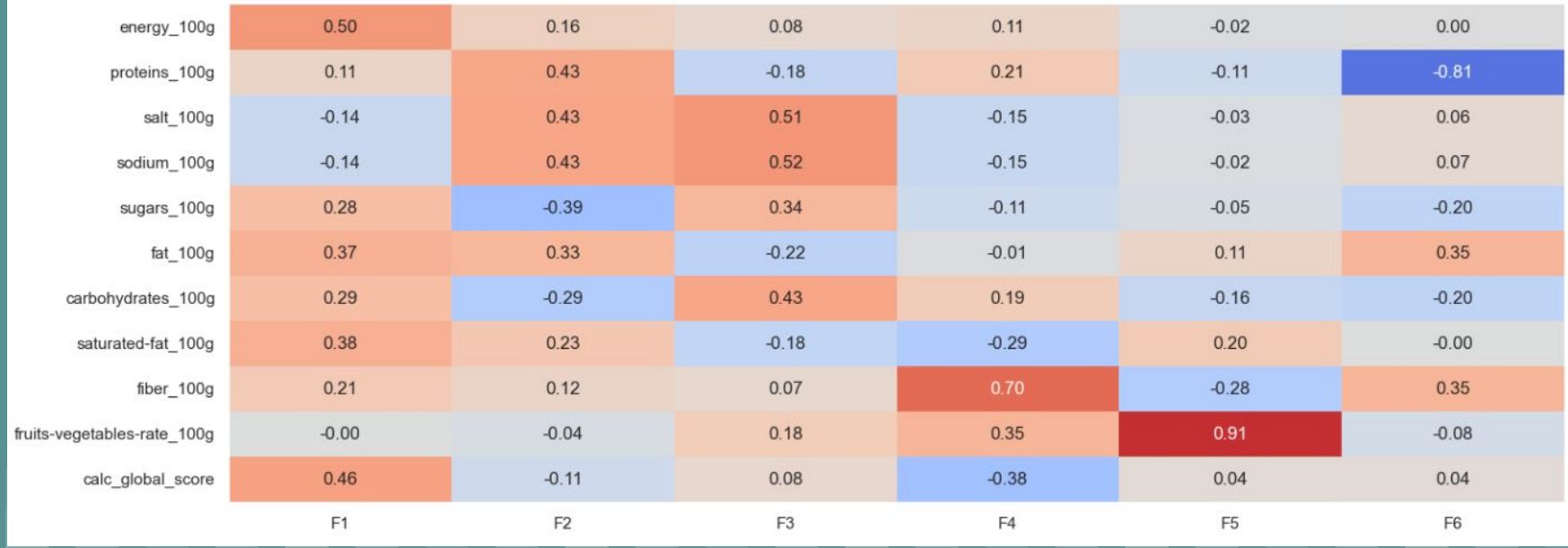

```
pcs.T
```

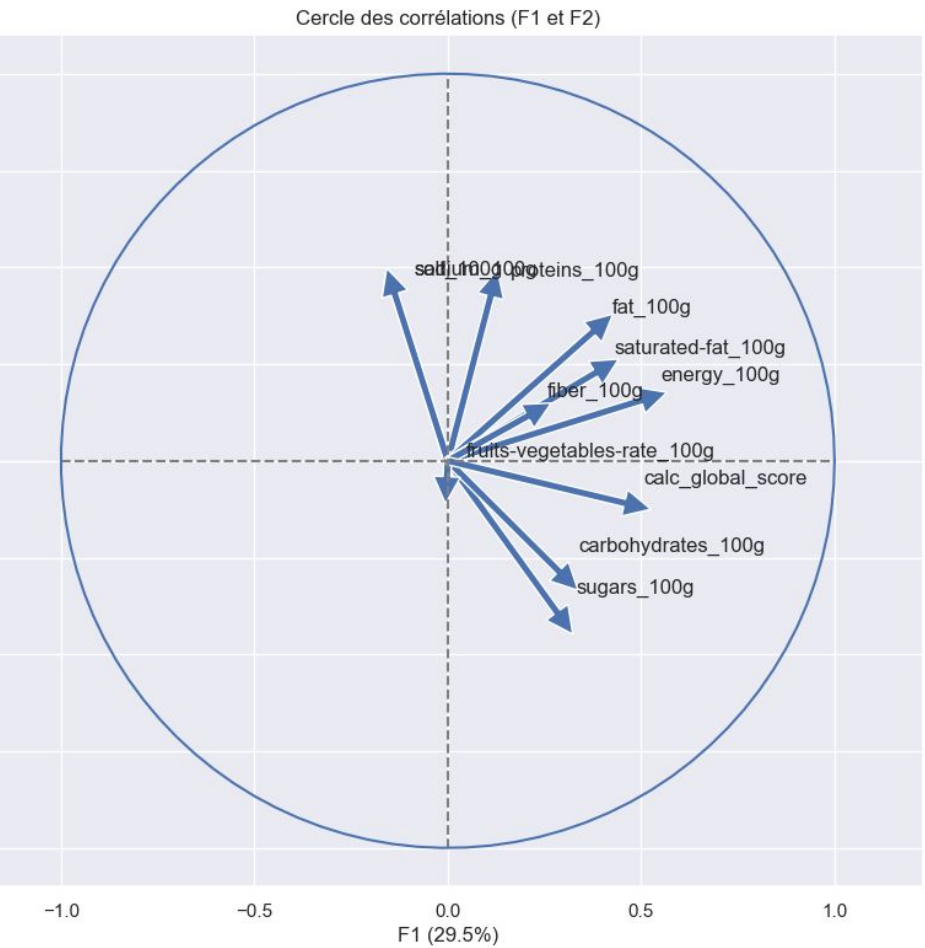
	F1	F2	F3	F4	F5	F6
energy_100g	0.500155	0.156052	0.077080	0.114587	-0.020688	0.000548
proteins_100g	0.110767	0.425232	-0.179364	0.205083	-0.113387	-0.812712
salt_100g	-0.137153	0.431201	0.514545	-0.153210	-0.025393	0.063738
sodium_100g	-0.136682	0.430743	0.515215	-0.152791	-0.018829	0.067916
sugars_100g	0.282414	-0.391167	0.341620	-0.113684	-0.045079	-0.196704
fat_100g	0.374691	0.332511	-0.217199	-0.012494	0.112826	0.351422
carbohydrates_100g	0.287334	-0.285391	0.434112	0.193913	-0.157329	-0.196132
saturated-fat_100g	0.381861	0.227020	-0.183866	-0.292217	0.196186	-0.002607
fiber_100g	0.206293	0.115146	0.073254	0.703923	-0.283360	0.350552
fruits-vegetables-rate_100g	-0.002129	-0.040903	0.182432	0.349256	0.908775	-0.077240
calc_global_score	0.457156	-0.108897	0.083045	-0.380043	0.039345	0.036136

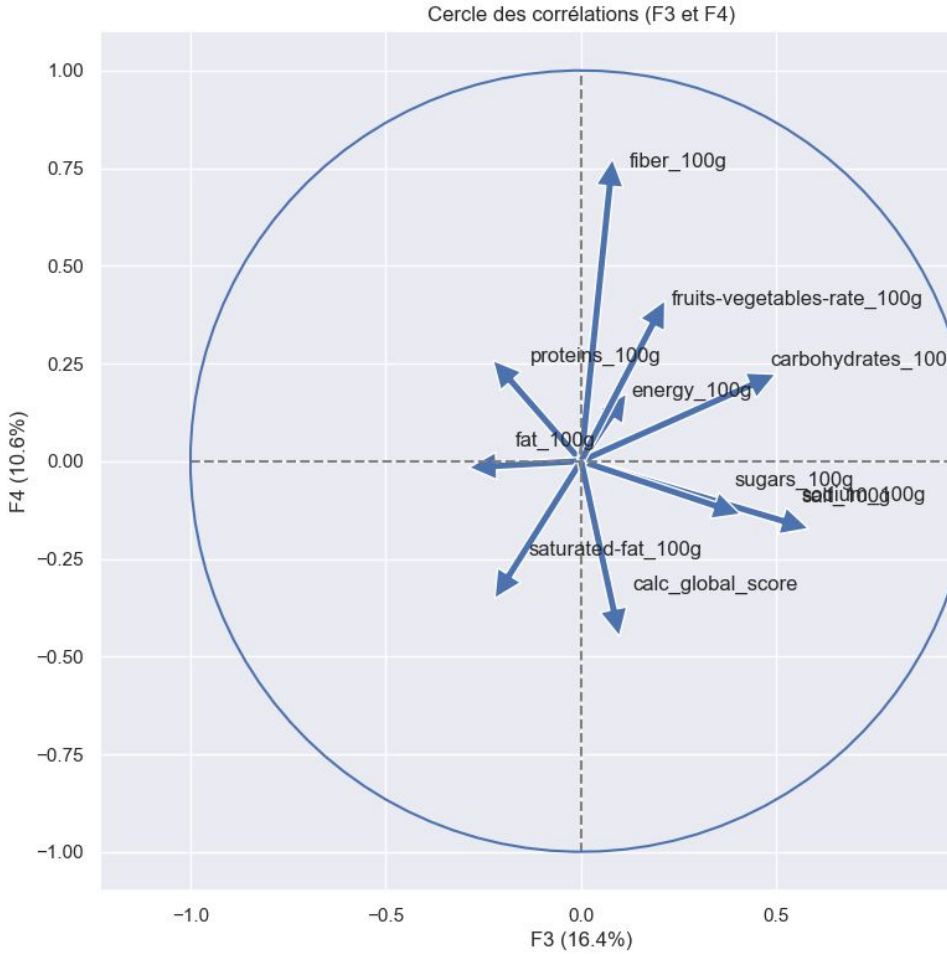
5. PCA Réduction de dimension

```
fig, ax = plt.subplots(figsize=(20, 6))
sns.heatmap(pcs.T, vmin=-1, vmax=1, annot=True, cmap="coolwarm", fmt="0.2f")
```

```
<AxesSubplot:>
```




Cercle des corrélations (F1 et F2)
proteins_100g
fat_100g
saturated-fat_100g
energy_100g
fiber_100g
fruits-vegetables-rate_100g
calc_global_score
carbohydrates_100g
sugars_100g
−1.0
−0.5
0.0
0.5
1.0
F1 (29.5%)


Cercle des corrélations (F3 et F4)
fiber_100g
fruits-vegetables-rate_100g
proteins_100g
energy_100g
fat_100g
sugars_100g
saturated-fat_100g
calc_global_score
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
F4 (10.6%)
−1.0
−0.5
0.0
0.5
F3 (16.4%)

06. Analyse exploratoire avec l'ANOVA

Est-ce que les variables quantitatives suivent une distribution normale ? Cette hypothèse doit être validée afin de pouvoir appliquer le test d'indépendance par l'analyse de la variance ANOVA.

A : Test de normalité d'Agostino & Pearson

B. Test de normalité Kolmogorov Smirnov

C. ANOVA : analyse de la variance

06. Analyse exploratoire avec

```
from scipy import stats
numeric_columns = data_ana.select_dtypes(include = ['int32','float64','int64']).columns
numeric_columns = numeric_columns[1:]
for column in numeric_columns:
    print('_______\n{}'.format(column))
    k2, p = stats.normaltest(data_ana[column],
                             axis=0,
                             nan_policy = 'omit')
    alpha = 5e-2
    print("p = {:g}".format(p))
    if p < alpha:  # null hypothese: x a une distribution normale
        print("H0 est rejetée : {} n'est pas de distribution normale".format(column))
    else:
        print("H0 ne peut être rejetée :{}, on considère l'hypothèse de normalité".format(column))
```

```
_______
proteins_100g
p = 0
H0 est rejetée : proteins_100g n'est pas de distribution normale

_______
salt_100g
p = 0
H0 est rejetée : salt_100g n'est pas de distribution normale

_______
sodium_100g
p = 0
H0 est rejetée : sodium_100g n'est pas de distribution normale

_______
additives_n
p = 0
H0 est rejetée : additives_n n'est pas de distribution normale
```

A : Test de normalité d'Agostino & Pearson

On ne peut pas appliquer le test Anova car skewtest n'est pas valide avec moins de 8 échantillons ; 13 échantillons ont été donnés

06. Analyse exploratoire avec:

```python
from scipy import stats
numeric_columns = data_ana.select_dtypes(include = ['int32','float64']).columns
numeric_columns = numeric_columns[1:]

for column in numeric_columns:
    print('________\n{}'.format(column))
    D, p = stats.kstest(data_ana[column].dropna(),
                        'norm',
                        args=(data_ana[column].mean(), data_ana[column].std()))
    alpha = 5e-2
    print("p = {}".format(p))
    if p < alpha:  # null hypothesis: x comes from a normal distribution
        print("The null hypothesis can be rejected : {} is not from a normal distribution".format(column))
    else:
        print("The null hypothesis cannot be rejected")
```

```
________
carbohydrates_100g
p = 0.0
The null hypothesis can be rejected : carbohydrates_100g is not from a normal distribution

________
saturated-fat_100g
p = 0.0
The null hypothesis can be rejected : saturated-fat_100g is not from a normal distribution

________
nutrition-score-fr_100g
p = 0.0
The null hypothesis can be rejected : nutrition-score-fr_100g is not from a normal distribution

________
nutrition-score-uk_100g
p = 0.0
The null hypothesis can be rejected : nutrition-score-uk_100g is not from a normal distribution
```

B. Test de normalité Kolmogorov Smirnov

Le test de Kolmogorov Smirnov confirme la non normalité des distributions de données : on ne peut appliquer l'analyse de la variance

06. Analyse exploratoire avec l'ANOVA

```
pd.DataFrame({'features': data_b.columns.tolist(),
              'p' : p,
              'bool_test' : p<alpha})

#si p<alpha : on rejette l'hypothèse d'indépendance H0 :
#les variables ne sont pas indépendantes
```

	features	p	bool_test
0	energy_100g	0.000000e+00	True
1	proteins_100g	0.000000e+00	True
2	salt_100g	0.000000e+00	True
3	sodium_100g	0.000000e+00	True
4	additives_n	0.000000e+00	True
5	sugars_100g	0.000000e+00	True
6	fat_100g	0.000000e+00	True
7	carbohydrates_100g	0.000000e+00	True
8	saturated-fat_100g	0.000000e+00	True
9	nutrition-score-fr_100g	0.000000e+00	True
10	nutrition-score-uk_100g	0.000000e+00	True
11	fiber_100g	0.000000e+00	True
12	fruits-vegetables-rate_100g	1.346896e-10	True
13	calc_global_score	0.000000e+00	True

C. ANOVA : analyse de la variance

Même si les données ne suivent pas une distribution normale, préparons les opérations pour appliquer le test si c'était le cas, à but d'apprentissage

Ce résultat indique que toutes les variables ont une valeur "p" très faible (inférieure au niveau de signification) et une valeur "bool_test" de "True", ce qui suggère que toutes les variables sont liées les unes aux autres et ne sont pas indépendantes

7. Calcul automatique de nutri score par régression linéaire

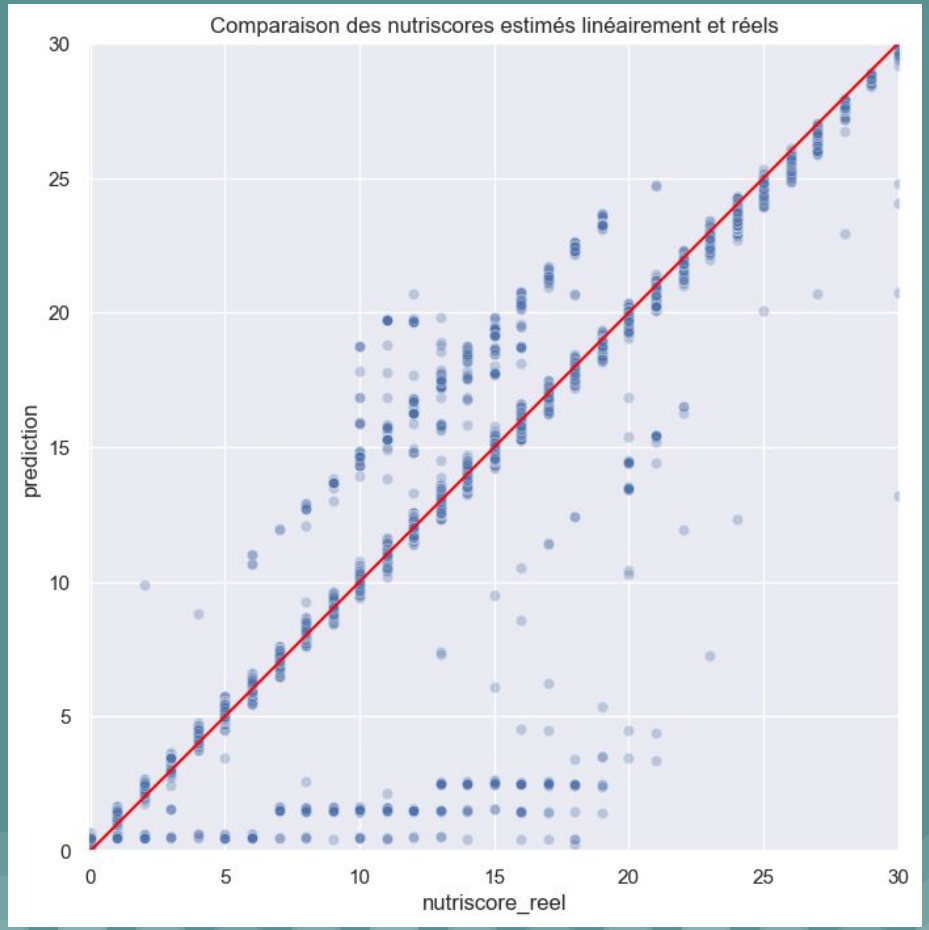
```
#?LinearRegression
```

```
lr = LinearRegression().fit(X_train, y_train)
print('R² sur jeu d\'entraînement : ', lr.score(X_train,y_trai
print('R² sur jeu de test', lr.score(X_test,y_test))
print('Poids de chaque variable dans la régression', lr.coef_)
print('ordonnée à l\'origine', lr.intercept_)
```

```
R² sur jeu d'entraînement :  0.9688428183226634
R² sur jeu de test 0.967480964897748
Poids de chaque variable dans la régression [ 0.01805688 -0.156
 -0.05286353 -0.11242194 -0.11756771  7.73806074  0.04516976]
ordonnée à l'origine 11.46660744201502
```

```
resultat.head()
```

	index	nutriscore_reel	prediction
0	141759	12.0	11.991587
1	62498	0.0	0.519550
2	65635	11.0	11.447803
3	167514	3.0	3.190594
4	136168	8.0	8.105656



On peut dire que le modèle de régression linéaire est très performant car les coefficients de détermination R^2 sont élevés, avec un R^2 de 0,97 pour le jeu de test et un R^2 de 0,97 pour le jeu d'entraînement. De plus, les poids de chaque variable dans la régression sont significatifs, ce qui indique que les variables ont une influence sur la variable cible. La constante de la régression est également significative, avec une ordonnée à l'origine de 11,47.

8. Synthèse

	index	nutriscore_reel	prediction
0	141759	12.0	11.991587
1	62498	0.0	0.519550
2	65635	11.0	11.447803
3	167514	3.0	3.190594
4	136168	8.0	8.105656

On peut dire que le modèle de régression linéaire est très performant car les coefficients de détermination R^2 sont élevés, avec un R^2 de 0,97 pour le jeu de test et un R^2 de 0,97 pour le jeu d'entraînement. De plus, les poids de chaque variable dans la régression sont significatifs, ce qui indique que les variables ont une influence sur la variable cible. La constante de la régression est également significative, avec une ordonnée à l'origine de 11,47. En résumé, on peut dire que la régression linéaire explique très bien la variation de la variable cible en fonction des variables indépendantes et est donc un bon modèle pour la prédiction.