

# **Présentation du projet 3 : Concevez une application au service de la santé publique**



**Préparé par :**  
**MAGHLAZI Hanane**

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

- **Introduction**
- **Définition de l'idée d'application**
- **Inspection du jeu de données**
- **Exploration du jeu de données :**
  - **Lecture des données**
  - **Vérification des données manquantes**
  - **Suppression des colonnes inutiles**
- **Vérification des doublons et gestion des outliers**
- **Analyse exploratoire**
- **Matrice des corrélations entre les variables**
- **Imputation des données :**
  - **Régression linéaire pour la variable fat\_100g**
  - **KNN imputer pour les nutriments**
  - **KNN regressor pour le nutriscore**
  - **KNN classifier pour le nutrigrade**
- **Analyses univariées et bivariées**
- **Anova**
- **Réalisation de l'ACP**
- **Réalisation de l'idée d'application**
- **Conclusion**

# Introduction :

- L'agence "Santé publique France" a lancé un appel à projets pour trouver des idées innovantes d'applications en lien avec l'alimentation.
- Le jeu de données Open Food Facts est disponible sur le site officiel
- Afin de bien mener ce projet quelques recherches sur les nutriments et le nutrition score ont été faites.



# Définir l'idée d'application :

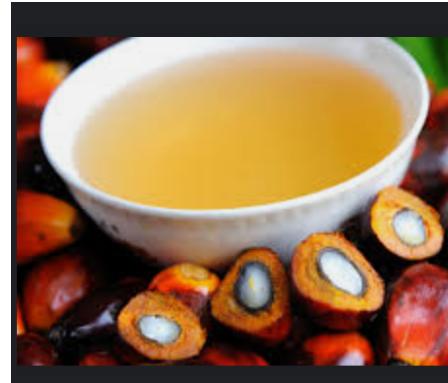
- Je souhaite aider les mamans à trouver des produits plus **sains** pour leurs **enfants** dans leur **alimentation**.
- Pour cela je propose une application qui retourne après avoir identifié le produit :
  - Une **note** entre 0 et 100 représentant le caractère nutritionnel sain.
  - Une **appréciation** sur le produit est également retournée selon sa note : Excellent, bon, médiocre, mauvais.
- Cette application affichera aussi les **caractéristiques** du produit :
  - **Défauts** : A savoir trop gras, trop sucré, trop calorique, graisses saturés et présences d'additifs
  - **Qualités** : Peu de sel, BIO, Sans gluten, Sans huile de palme, Végétarien, absence d'additifs, présence de fibres
  - Présences **d'additifs** spécifiques causant quelques maladies comme **l'hyperactivité**



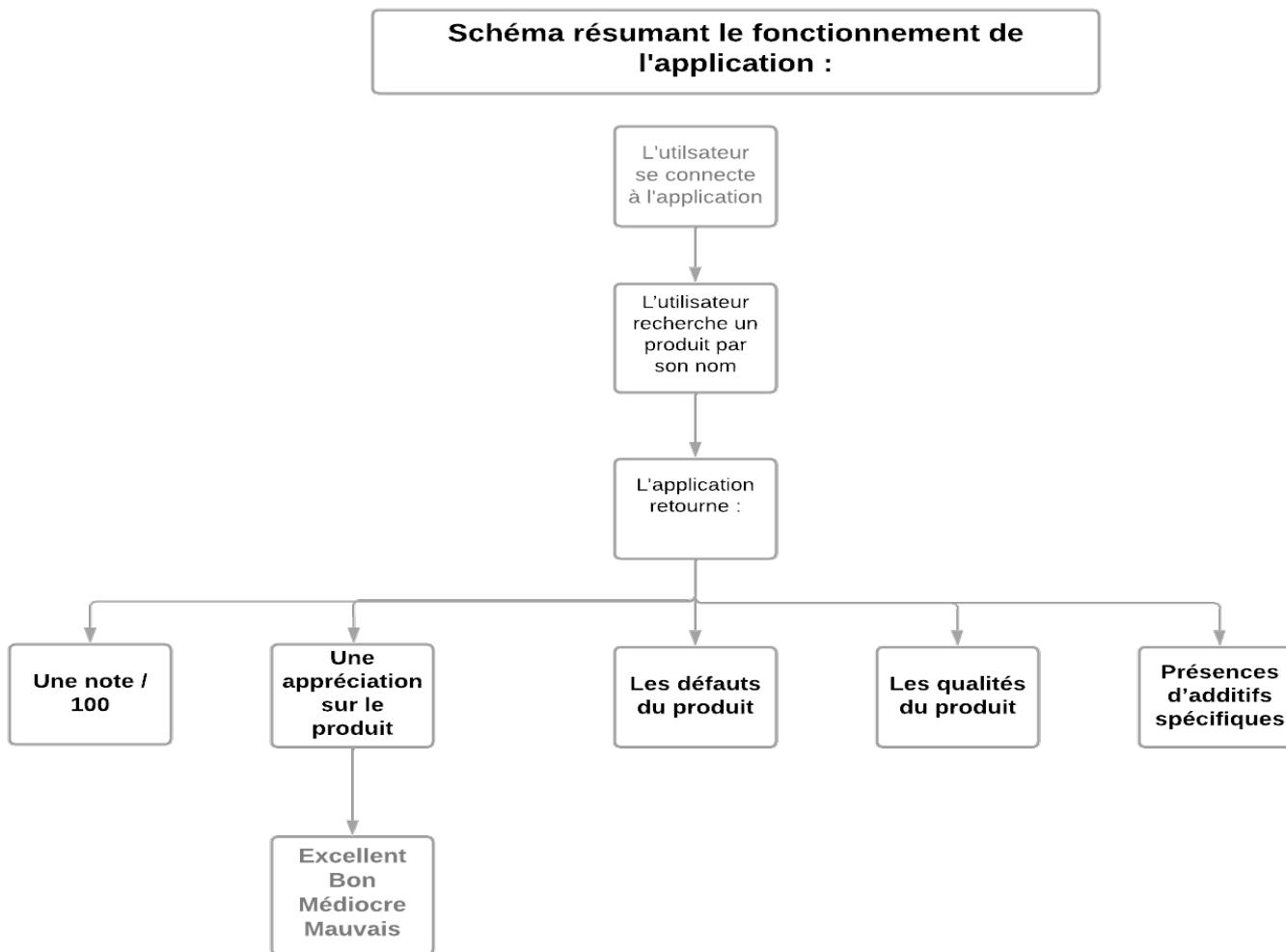
# Définir l'idée d'application :

La note est construite en agrégeant les indicateurs suivants :

- **Le nombre d'additifs**
- **Le type d'additifs**
- **La présence d'ingrédients issus de l'huile de palme**
- **La présence d'un label biologique**
- **Le nutriscore**



# Schéma résumant le fonctionnement :



# Inspection des données

- Présentation du jeu de données :

Dimensions	NB_lignes	NB_colonnes	Total remplissage	Description
food (320772, 162)	320772	162	12356406	Il s'agit d'un fichier de produits avec les ingrédients composant les produits, leurs additifs et des informations nutritionnelles

```
print(missing_data(df_food), "données manquantes")
```

39608658 données manquantes

```
print(format_pourcentage(missing_percent(df_food)), "est le pourcentage de données manquantes")
```

76.2% est le pourcentage de données manquantes

# Inspection des données

- Les données de ce fichier sont bien détaillés .
- Les champs qui se terminent par \_t sont des dates
- les champs qui se terminent par \_datetime sont des dates au format iso8601
- les champs qui se terminent par \_tags sont une liste de balises séparées par des virgules (par exemple, categories\_tags est l'ensemble de balises normalisées de l'ordinateur du champ catégories)
- les champs qui se terminent par un code de langue à 2 lettres (par exemple Fr pour le français) sont l'ensemble des balises dans cette langue
- les champs qui se terminent par **\_100g** correspondent à la quantité d'un nutriment (en g, ou kJ pour l'énergie) pour **100 g** ou **100 ml** de produit
- Un **Nutriscore** allant de **-15 à 40** qui constitue une note selon les nutriments contenus dans les aliments.
- Un **Nutrigrade** qui est un étiquetage nutritionnel coloré qui classe de **A à E** les aliments.
- les champs qui se terminent par \_portion correspondent à la quantité d'un nutriment (en g, ou kJ pour l'énergie) pour 1 portion du produit
- A ce stade je cherche à déterminer les **variables** qui sont à la fois **pertinentes et utiles** dans le cadre du **projet** et d'idée d'application et ainsi **exclure** les **variables** inutiles et qui ont un taux de **nan élevé**.
- Par la suite il sera possible d'envisager des **stratégies d'imputation** pour compléter les valeurs **manquantes** pour les données nécessaires à l'analyse

# Nettoyage du jeu de données :

- Beaucoup de données manquantes sur plusieurs variables, il y a même des colonnes vides à **100%**
- Je crée un premier filtre en supprimant les données avec **90%** de données **manquantes** et plus

```
# Supprimer colonnes
def supp_col_null(df):
    column_with_nan = df.columns[df.isnull().any()]
    for column in column_with_nan:
        if df[column].isnull().sum()*100.0/df.shape[0] > 90:
            df.drop(column,1, inplace=True)
```

```
supp_col_null(df_food)
```

```
print(df_food.shape[1],"colonnes restantes")
```

62 colonnes restantes

- 100 variables ont été supprimés

# Nettoyage du jeu de données :

- Lecture des colonnes et suppression des colonnes inutiles à l'analyse:

```
# Supprimer les colonnes inutiles
df_food.drop(["url","creator","created_t","created_datetime","last_modified_t","last_modified_datetime",
             "image_url", "image_small_url","packaging","states","generic_name"], axis = 1, inplace = True)
print(df_food.shape[1],"colonnes restantes")
```

51 colonnes restantes

11 variables supprimés

# Nettoyage du jeu de données :

- Sur plusieurs colonnes il y a la même variable avec \_tags
- Sur plusieurs colonnes il y a la même variable avec \_fr
- Ce sont des colonnes redondantes

- Je supprime les colonnes avec \_tags car ce ne sont que des répétitions des colonnes

```
def delete_specific_df_columns(df,suffix:str):
    suffix_columns=[]
    for col in df.columns:
        if suffix in col:
            suffix_columns.append(col)
    suffix_columns_df=df[suffix_columns]
    df.drop(df[suffix_columns], axis=1,inplace=True)
    return(print("colonnes restantes",df.shape[1]))
```

```
delete_specific_df_columns(df_food,'_tags')
```

colonnes restantes 43

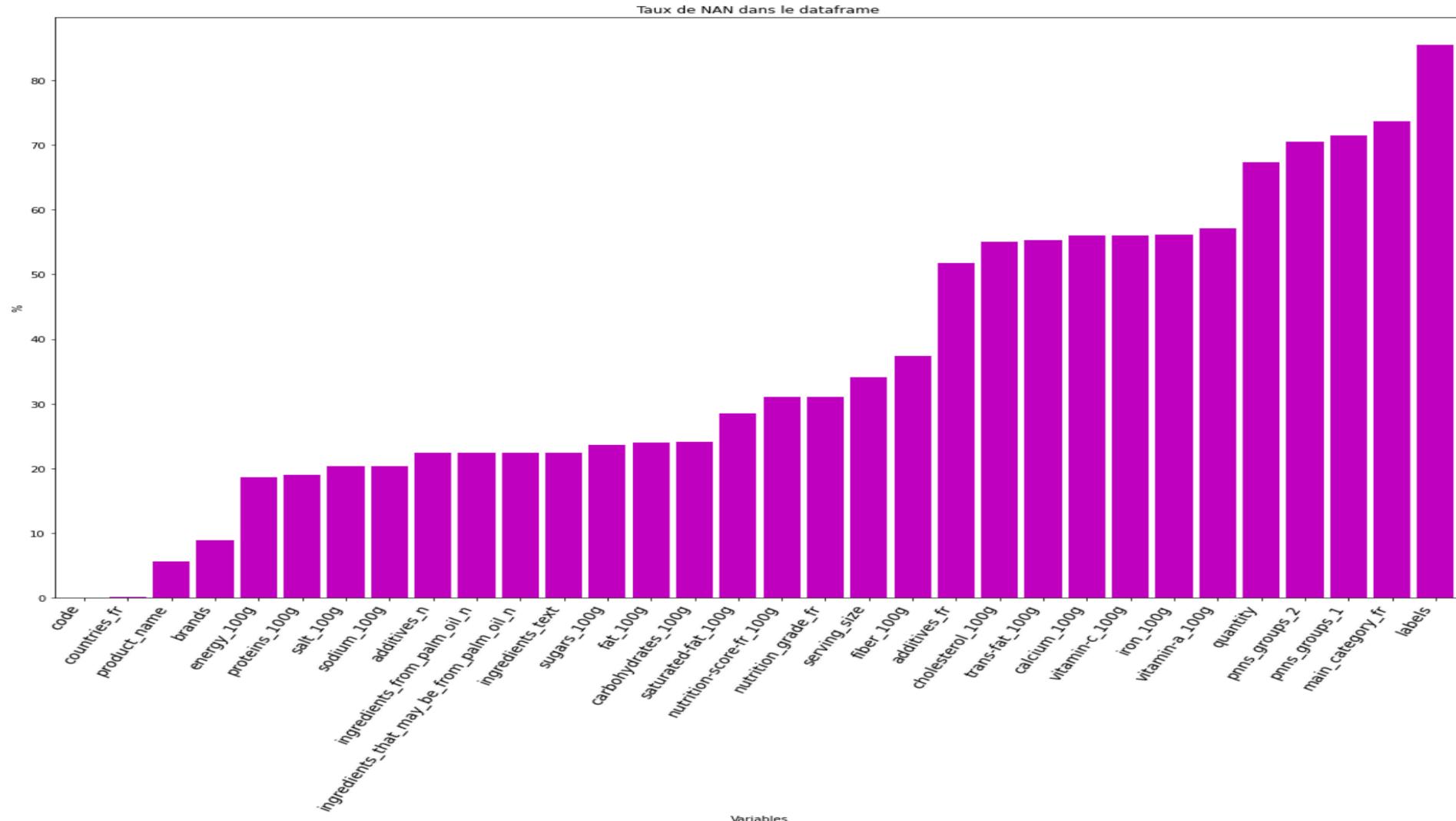
```
# Supprimer les colonnes inutiles
df_food.drop(["states_fr","categories_fr","labels_fr","additives","main_category",
              "countries","nutrition-score-uk_100g","stores","purchase_places","manufacturing_places","categories"], axis = 1, inplace = True)
print(df_food.shape[1],"colonnes restantes")
```

32 colonnes restantes

11 variables supprimées

- De même pour les colonnes avec -fr

# Nettoyage du jeu de données :



# Gestion des doublons

- **Vérification des doublons :**

```
# Doublons
doublons = df_food_clean[df_food_clean.duplicated(['product_name'], keep=False)]
nb_doublons_name = doublons.shape[0]
print(f'Nombre de doublons sur le nom du produit : {nb_doublons_name}')
```

Nombre de doublons sur le nom du produit : 117435

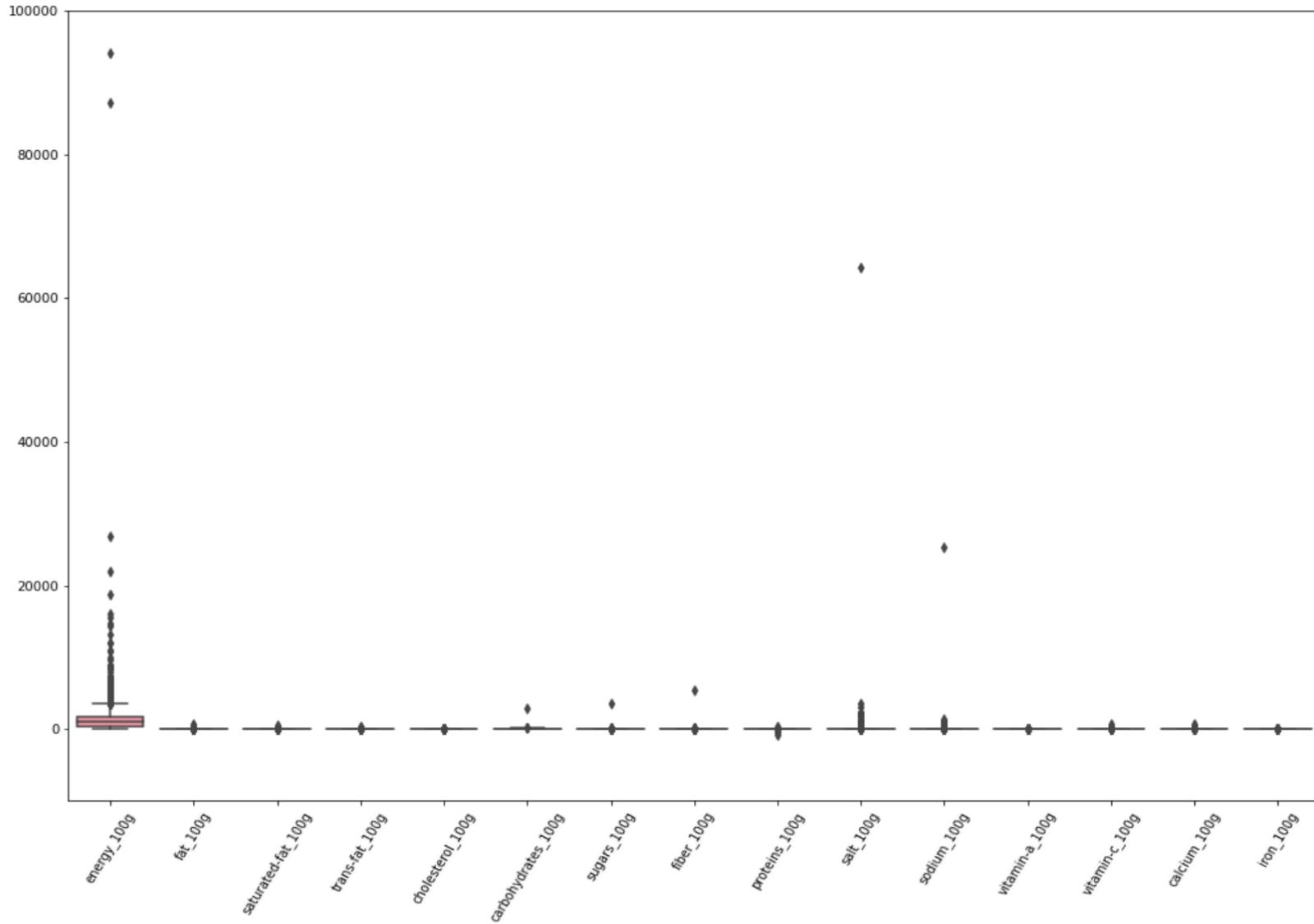
- Des doublons concernant le nom des produits, après vérification il s'agit du même nom de produit mais de marque ou quantité différente

# Gestion des outliers

```
nutriments.describe()
```

	energy_100g	fat_100g	saturated-fat_100g	trans-fat_100g	cholesterol_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g	sodium_100g	vitamin-a_100g	vitamin-c_100g	calcium_100g	iron_100g
count	248332.000000	232186.000000	217900.000000	142658.000000	143426.000000	231921.000000	233215.000000	192801.000000	247330.000000	243648.000000	243604.000000	136847.000000	140039.000000	139991.000000	139551.000000
mean	1128.311159	12.710996	5.137230	0.073394	0.020105	32.163226	16.108056	2.846655	7.053529	2.014078	0.793087	0.000201	0.021922	0.124015	0.003498
std	992.845191	17.678322	8.054879	1.543113	0.358882	29.835565	22.448476	13.084751	8.388287	131.337425	51.712315	0.013715	2.220205	3.325242	0.208808
min	0.000000	0.000000	0.000000	-3.570000	0.000000	0.000000	-17.860000	-6.700000	-800.000000	0.000000	0.000000	-0.000340	-0.002100	0.000000	-0.000260
25%	372.000000	0.000000	0.000000	0.000000	0.000000	6.000000	1.365000	0.000000	0.600000	0.060960	0.024000	0.000000	0.000000	0.000000	0.000000
50%	1100.000000	4.900000	1.790000	0.000000	0.000000	20.690000	5.880000	1.500000	4.700000	0.580000	0.229000	0.000000	0.000000	0.035000	0.001000
75%	1674.000000	20.000000	7.140000	0.000000	0.020000	58.700000	24.140000	3.600000	10.000000	1.379220	0.543000	0.000107	0.003600	0.105000	0.002400
max	231199.000000	714.290000	550.000000	369.000000	95.238000	2916.670000	3520.000000	5380.000000	430.000000	64312.800000	25320.000000	5.000000	716.981100	694.737000	50.000000

# Gestion des outliers



- les Boxplots montrent qu'il y a beaucoup d'outliers

# Gestion des outliers

```
# Pour la colonne energy > 3700 je les remplace par des NAN
nutriments.loc[nutriments.energy_100g > 3700, 'energy_100g'] = np.nan

# Pour le reste des nutriments les valeurs inférieures à 0 et supérieurs à 0 je les remplace par des nan

for col in nutriments.columns:
    nutriments.loc[nutriments[col]<0, col] = np.nan

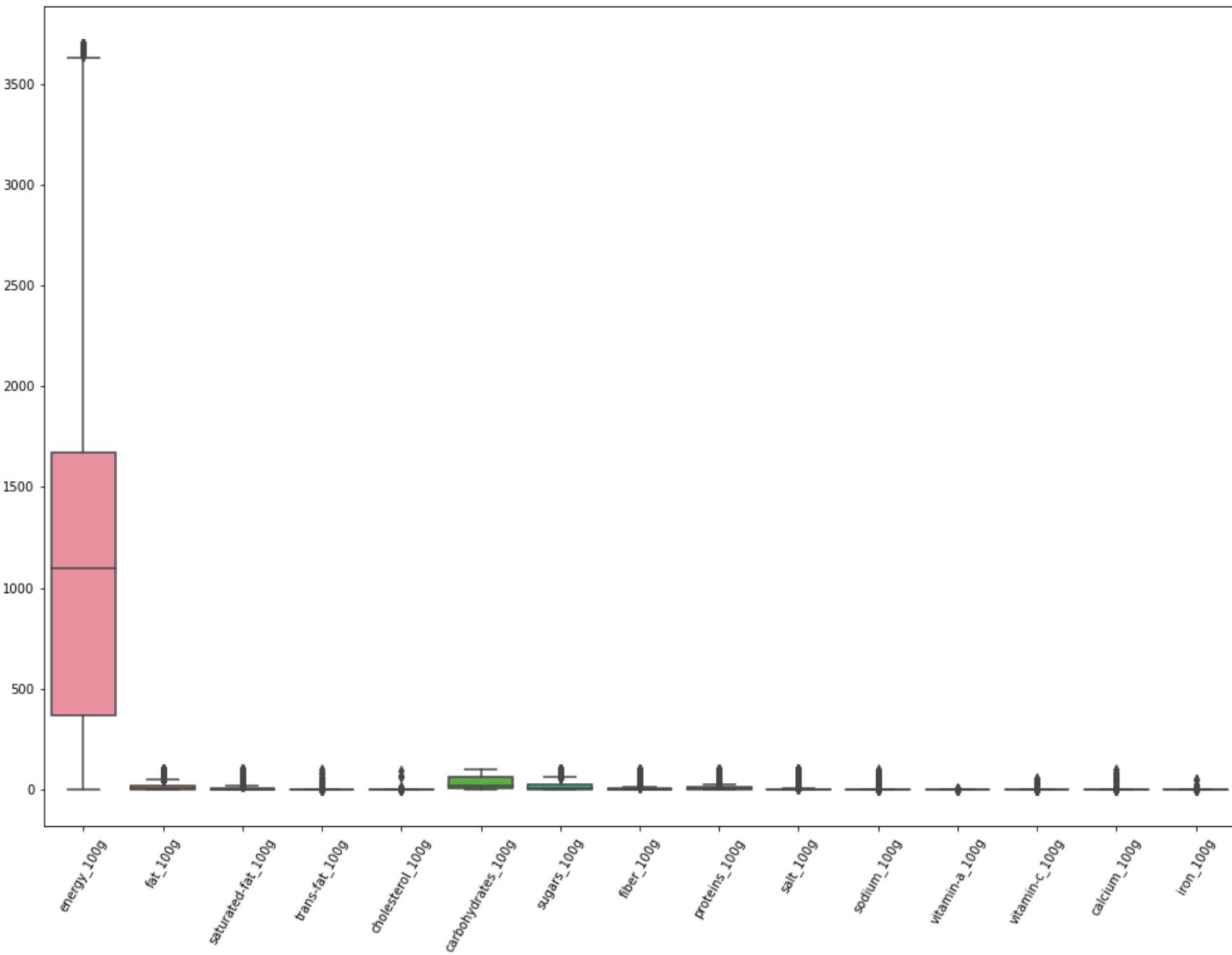
def outlier(df):
    for col in df.iloc[:,1:15]:
        df.loc[df[col]>100, col] = np.nan
    return df

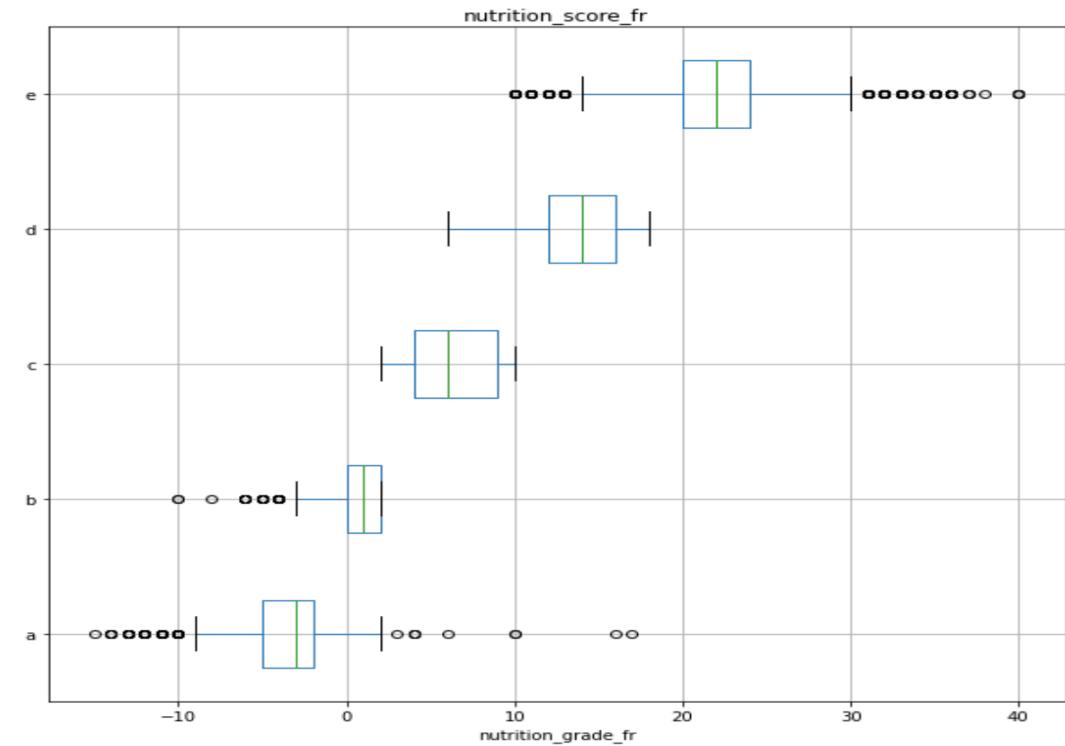
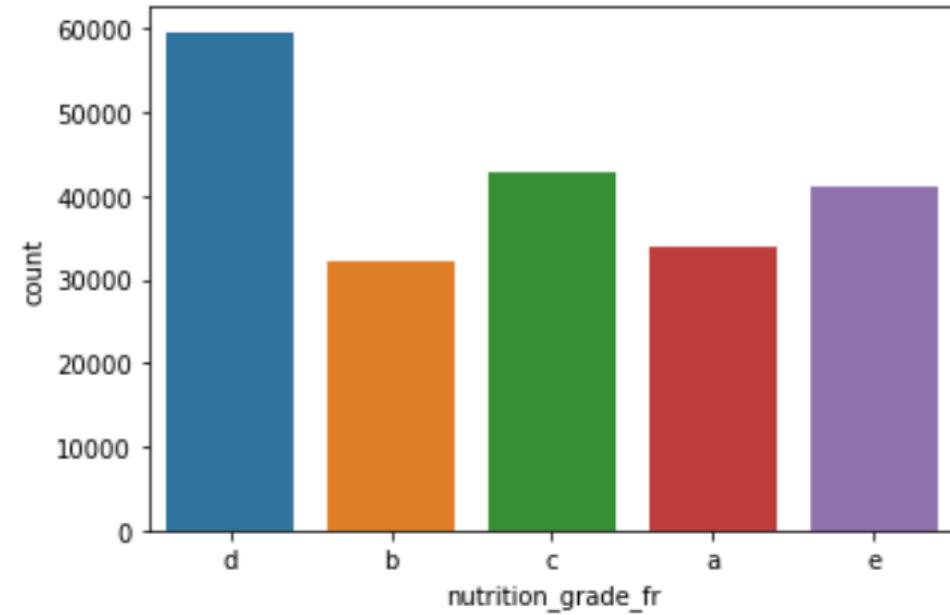
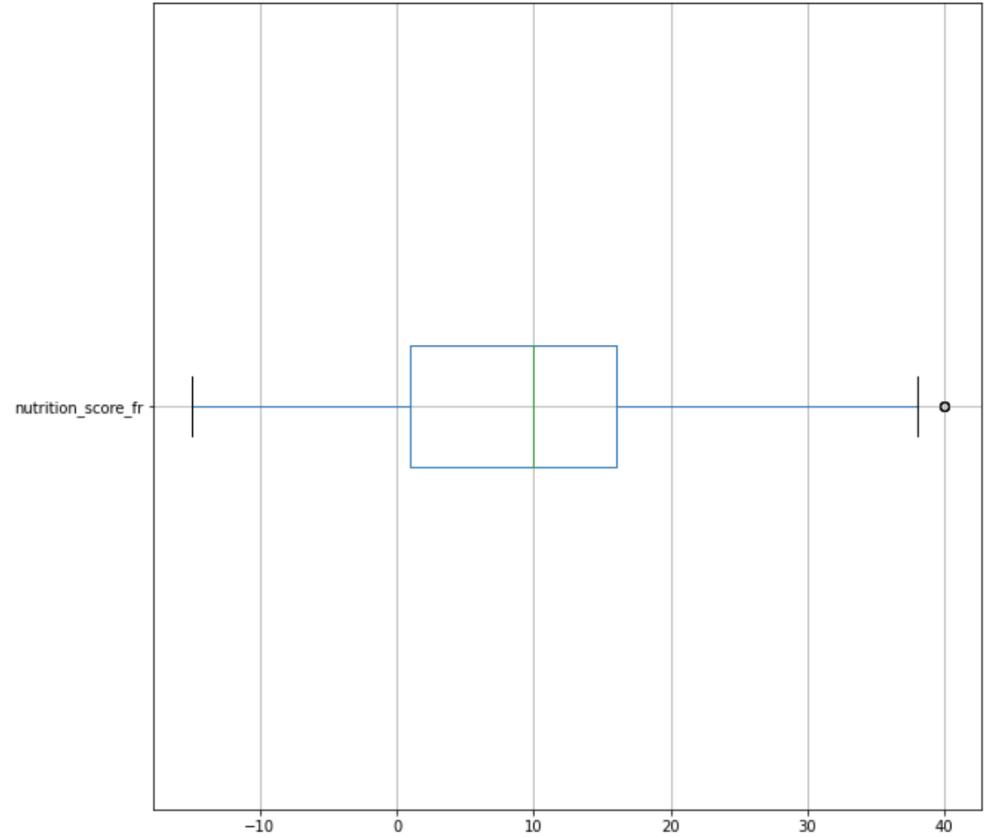
outlier(nutriments)
```

# Gestion des outliers

```
# Je vérifie les stats  
nutriments.describe()
```

	energy_100g	fat_100g	saturated-fat_100g	trans-fat_100g	cholesterol_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g	sodium_100g	vitamin-a_100g	vitamin-c_100g	calcium_100g
count	247726.000000	232182.000000	217897.000000	142649.000000	143426.000000	231911.000000	233203.000000	192797.000000	247326.000000	243496.000000	243570.000000	136846.000000	140035.000000	139981.000000
mean	1118.360562	12.705615	5.133008	0.067061	0.020105	32.146727	16.091733	2.816682	7.057175	1.555390	0.634031	0.000201	0.013066	0.100700
std	791.706344	17.599921	7.949457	0.946223	0.358882	29.220637	21.238399	4.559161	8.122050	6.052575	2.586650	0.013715	0.334911	0.555123
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	370.000000	0.000000	0.000000	0.000000	0.000000	6.000000	1.370000	0.000000	0.600000	0.060960	0.024000	0.000000	0.000000	0.000000
50%	1100.000000	4.900000	1.790000	0.000000	0.000000	20.690000	5.880000	1.500000	4.700000	0.580000	0.228346	0.000000	0.000000	0.035000
75%	1674.000000	20.000000	7.140000	0.000000	0.020000	58.700000	24.140000	3.600000	10.000000	1.374140	0.542000	0.000107	0.003600	0.105000
max	3700.000000	100.000000	100.000000	100.000000	95.238000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	5.000000	57.140000	99.006000





# Sélection des variables

```
1 | missing_values(food)
```

	Variable	nan	%nan
11	labels_a	222966	0.849980
9	pnns_groups_1	193986	0.739504
10	pnns_groups_2	191752	0.730988
15	trans-fat_100g	119774	0.456597
4	additives_fr	115364	0.439785
17	fiber_100g	71541	0.272725
7	nutrition_grade_fr	54600	0.208144
8	nutrition_score_fr	54600	0.208144
14	saturated-fat_100g	46864	0.178653
13	fat_100g	32953	0.125622
16	sugars_100g	31696	0.120830
3	additives_n	27494	0.104811
5	ingredients_from_palm_oil_n	27494	0.104811
6	ingredients_that_may_be_from_palm_oil_n	27494	0.104811
19	sodium_100g	21243	0.080982
18	proteins_100g	17813	0.067906
12	energy_100g	17428	0.066438
2	countries_fr	46	0.000175
1	product_name	0	0.000000
0	code	0	0.000000

Beaucoup de NAN pour la variable trans-fat\_100g, je la supprime, en plus elle ne rentre pas dans le calcul du nutriscore

# Sélection des variables

- Le data frame est à présent nettoyé et prêt pour l'analyse et ainsi définir une application qui détermine le nutriscore en choisissant les variables nécessaires :
- Les variables à utiliser sont : energy\_100g, fat\_100g, saturated\_fat\_100g, sugars\_100g, fiber\_100g, proteins\_100g, additives\_fr, ingredients\_from\_palm\_oil\_n, labels\_a
- J'écarte le sel car il n'est pas pris en charge dans le calcul du nutriscore mais je garde plutôt le sodium

```
1 | food.columns  
  
dex(['code', 'product_name', 'countries_fr', 'additives_n', 'additives_fr',  
     'ingredients_from_palm_oil_n',  
     'ingredients_that_may_be_from_palm_oil_n', 'nutrition_grade_fr',  
     'nutrition_score_fr', 'pnns_groups_1', 'pnns_groups_2', 'labels_a',  
     'energy_100g', 'fat_100g', 'saturated_fat_100g', 'sugars_100g',  
     'fiber_100g', 'proteins_100g', 'sodium_100g'],  
     dtype='object')
```

# Filtre sur les lignes

- Supprimer les lignes trop pauvres en données quantitatives:

```
1 | missing_values(df_quant)
```

	Variable	nan	%nan
3	trans-fat_100g	155499	0.521550
5	fiber_100g	105351	0.353351
3	nutrition_score_fr	88145	0.295642
2	saturated-fat_100g	80251	0.269165
1	fat_100g	65966	0.221253
1	sugars_100g	64945	0.217828
1	sodium_100g	54578	0.183057
3	proteins_100g	50822	0.170459
1	energy_100g	50422	0.169117

```
# Je supprime les produits sans nom
food = food.dropna(subset = ['product_name'])
#df_categ=df_categ.dropna(subset=['product_name'])
```

```
def delete_NA(df, perc=80):
```

```
#Supprime les lignes avec 80% de valeurs manquantes
min_count = int(((100-perc)/100)*df.shape[1] + 1)
```

```
df = df.dropna(axis=0, thresh=min_count)
```

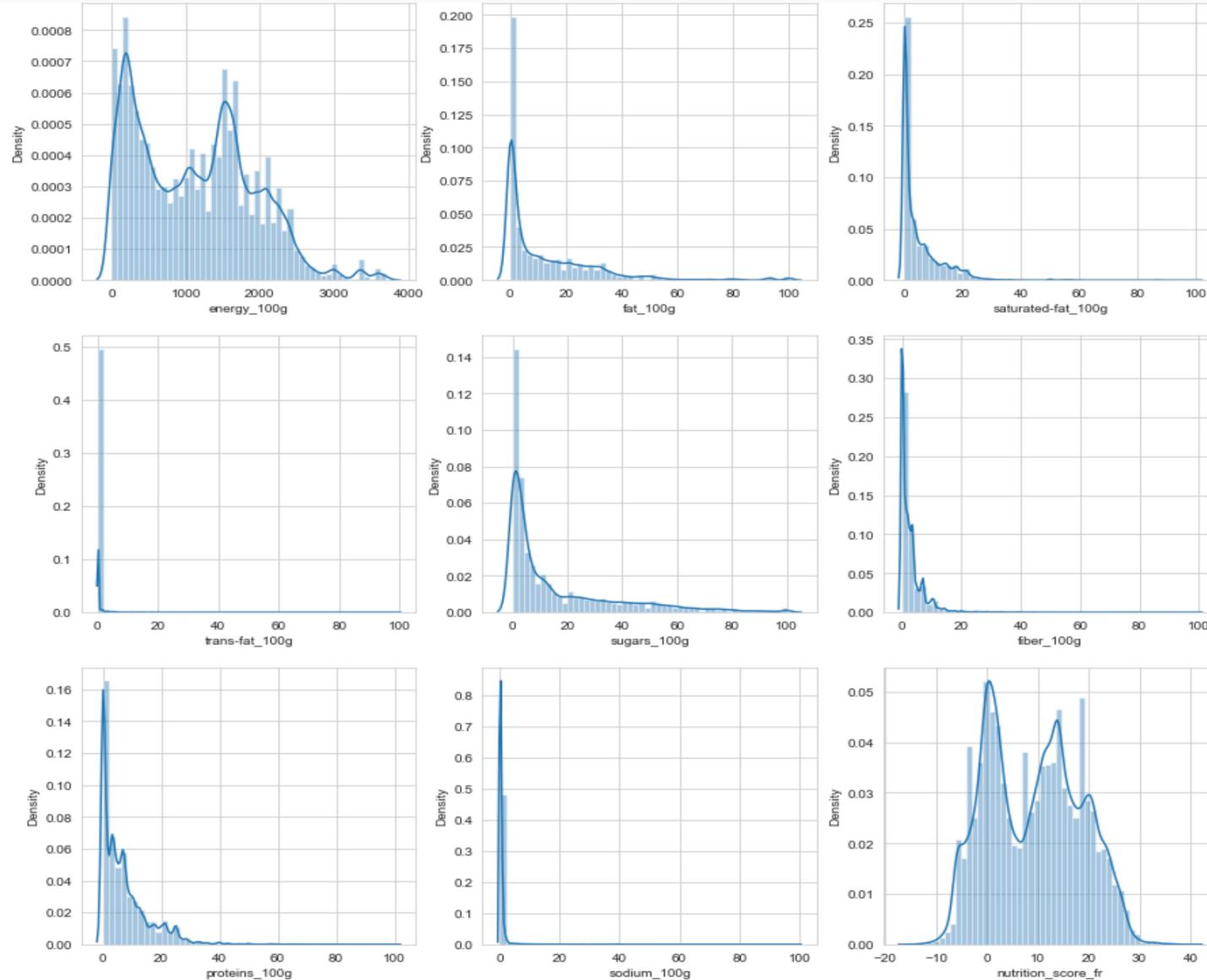
```
return(df)
```

```
:5]: 1 | food.shape
```

```
: (262319, 20)
```

20536 lignes supprimées

# Vérification des distributions



- Visuellement les distributions semblent asymétriques

# Vérification des distributions

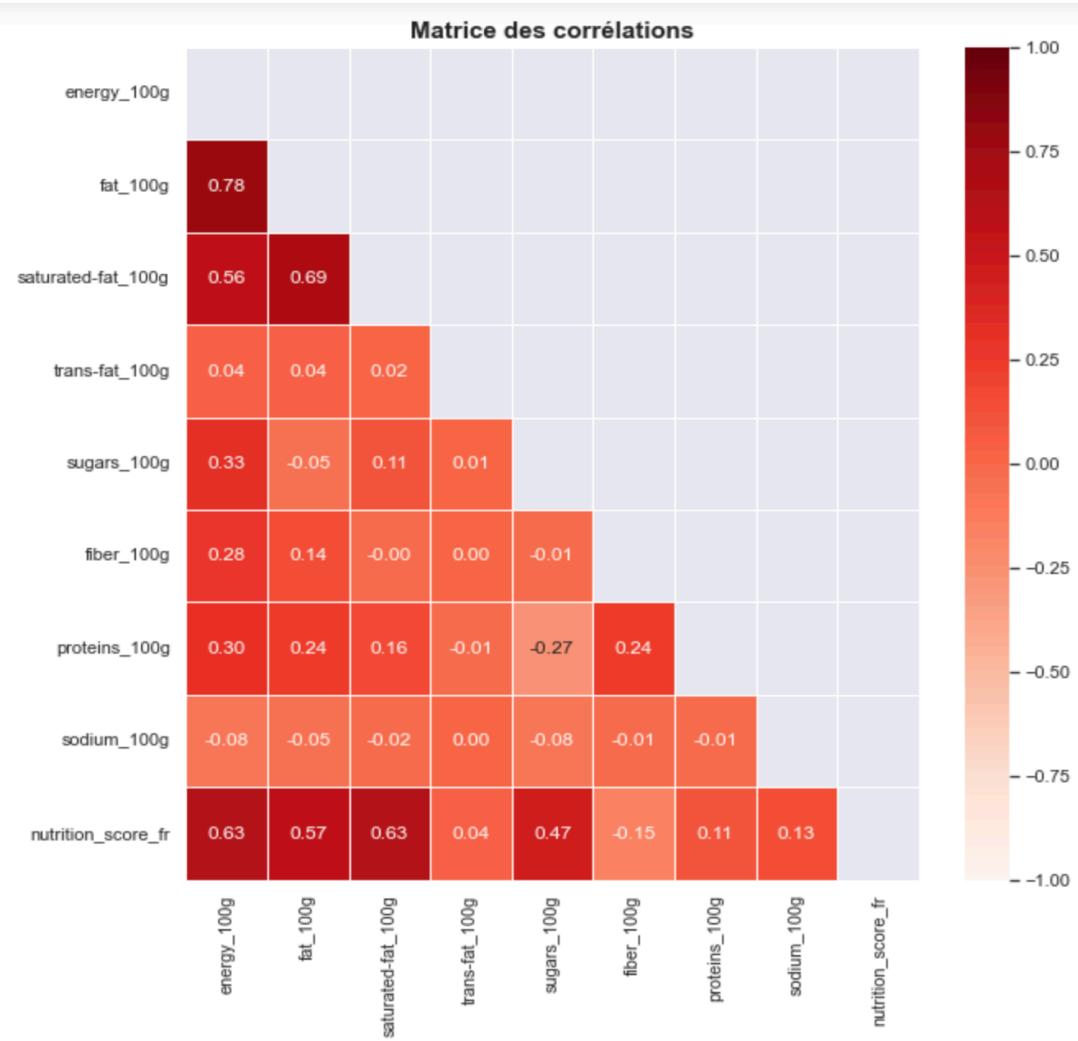
Test de normalité Kolmogorov Smirnov

```
]: 1 alpha=0.05
 2 for col in rows:
 3     print([col])
 4     a,b= stats.kstest( rows[[col]],'norm' )
 5     print("statistics",a,"p-value",b)
 6     if b < alpha:
 7         print("H0 rejetée")
 8     else:
 9         print("H0 non rejetée")
```

```
['energy_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['fat_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['saturated-fat_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['trans-fat_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['sugars_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['fiber_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['proteins_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['sodium_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['nutrition_score_fr']
statistics 1.0 p-value 0.0
H0 rejetée
```

- En se basant sur les projections obtenus et les résultats des tests de Kolmogorov-Smirnov (P-value < au niveau de test de 5%) on rejette donc l'hypothèse de normalité des distributions de ces variables.

# Matrice des corrélations



- Une relation linéaire **positive** entre :
  - fat\_100g et energy\_100
  - fat\_100g et saturated\_fat\_100g
  - saturated\_fat et energy\_100
  - fiber\_100g et proteins\_100g
  - nutriscore et fat\_100
  - nutriscore et energy\_100
  - nutriscore et sugars\_100
  - nutriscore et sodium\_100 (faible)
- Une relation linéaire **négative** entre :
  - proteins\_100g et sugars\_100g
  - sugars\_100g et fat\_100g (faible relation)
  - fiber\_100 et sugars\_100
  - nutriscore et fiber\_100

# Imputation des données

- Imputations de données :

Régression linéaire : pour la variable fat\_100g

```
] : 1 #On régresse fat_100g en fonction des autres variables de l'échantillon
 2 reg_multi = smf.ols('fat_100g~saturated_fat_100g+energy_100g', data=df_quant).fit()
 3 print(reg_multi.summary())

OLS Regression Results
=====
Dep. Variable:          fat_100g    R-squared:                   0.706
Model:                          OLS    Adj. R-squared:                 0.706
Method:                     Least Squares    F-statistic:            2.385e+05
Date:                Wed, 29 Jun 2022    Prob (F-statistic):        0.00
Time:                      00:45:19    Log-Likelihood:         -7.3024e+05
No. Observations:      198349    AIC:                         1.460e+06
Df Residuals:           198346    BIC:                         1.461e+06
Df Model:                           2
Covariance Type:            nonrobust
=====
              coef    std err          t      P>|t|      [ 0.025   0.975 ]
-----  
Intercept     -5.9448      0.040     -149.627      0.000     -6.023     -5.867
saturated_fat_100g    0.8202      0.003      237.669      0.000      0.813     0.827
energy_100g       0.0133    3.35e-05      397.883      0.000      0.013     0.013
-----  
Omnibus:            65463.921    Durbin-Watson:             0.894
Prob(Omnibus):        0.000    Jarque-Bera (JB):        375777.856
Skew:                  1.476    Prob(JB):                  0.00
Kurtosis:                 9.062    Cond. No.            2.63e+03
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
# Remplacer les NAN de la colonne fat_100g par une régression linéaire multiple
food.loc[food.fat_100g.isna(), 'fat_100g'] = reg_multi.predict(food)
```

# Imputation des données

- Imputation des données quantitatives : nutriments par un KNNImputer

```
imputer = KNNImputer(n_neighbors=5,missing_values=np.nan)
df_knn = pd.DataFrame(imputer.fit_transform(df_nutriment),columns = df_nutriment.columns)
```

```
1 missing_values(df_nutriment)
```

Variable	nan	%nan	energy_100g	fat_100g	saturated_fat_100g	sugars_100g	fiber_100g	proteins_100g	sodium_100g
energy_100g	0	0.0	count	262319.000000	262319.000000	262319.000000	262319.000000	262319.000000	262319.000000
fat_100g	0	0.0	mean	1118.634755	12.772761	4.740625	15.556070	2.560239	7.051389
saturated_fat_100g	0	0.0	std	768.410260	16.881362	7.465569	20.246827	4.054738	7.870605
sugars_100g	0	0.0	min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
fiber_100g	0	0.0	25%	406.000000	0.000000	0.000000	1.300000	0.000000	0.800000
proteins_100g	0	0.0	50%	1117.534798	6.800000	1.670000	6.670000	1.500000	5.260000
sodium_100g	0	0.0	75%	1644.000000	19.230000	6.200000	21.050000	3.300000	9.700000
			max	3700.000000	100.000000	100.000000	100.000000	100.000000	100.000000

# Imputation des données

- Afin de prédire la variable nutriscore, j'utilise un KNN regressor

```
7]: 1 # La variable prédictive : nutrition_score_fr
    2 y=df_nutriment_complet_full["nutrition_score_fr"].values
    3 # Les features
    4 x=df_nutriment_complet_full.drop("nutrition_score_fr",axis=1).values

3]: 1 xtrain, xtest, ytrain, ytest = train_test_split(x, y, train_size=0.8, random_state=42)
    2 print("shape of original dataset :", df_nutriment_complet_full.shape)
    3 print("shape of input - training set", xtrain.shape)
    4 print("shape of output - training set", ytrain.shape)
    5 print("shape of input - testing set", xtest.shape)
    6 print("shape of output - testing set", ytest.shape)

shape of original dataset : (207719, 8)
shape of input - training set (166175, 7)
shape of output - training set (166175,)
shape of input - testing set (41544, 7)
shape of output - testing set (41544,)
```

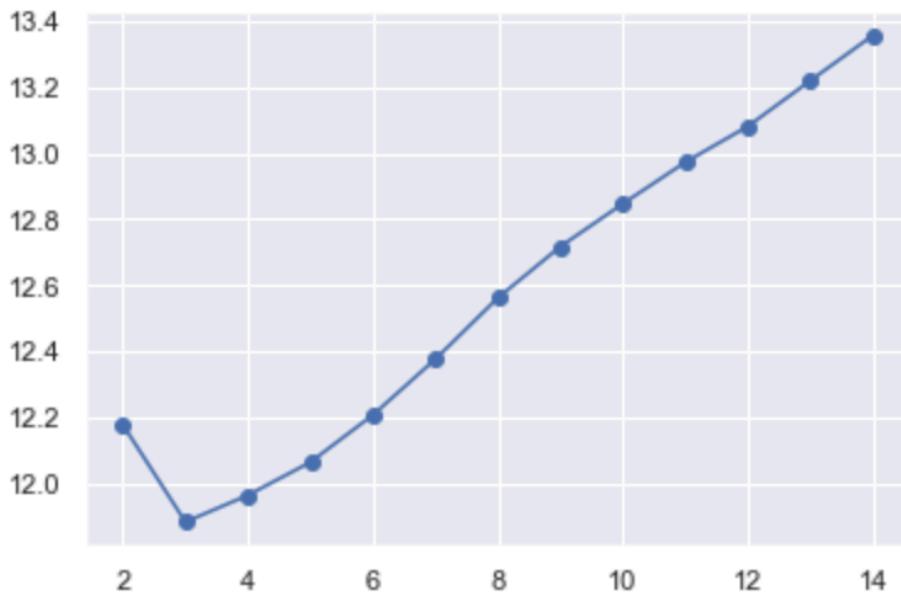
# Imputation des données

```
30]: 1 knn_reg = neighbors.KNeighborsRegressor(n_neighbors=5)
      2 knn_reg.fit(xtrain,ytrain)
      3 y_pred=knn_reg.predict(xtest)
```

```
31]: 1 print("train score",knn_reg.score(xtrain,ytrain))
      2 print("test score",knn_reg.score(xtest,ytest))
```

train score 0.9230900352420074

test score 0.8793709255519275



```
33]: 1 knn_reg = neighbors.KNeighborsRegressor(n_neighbors=3)
      2 knn_reg.fit(xtrain,ytrain)
      3 y_pred=knn_reg.predict(xtest)
```

```
34]: 1 print("train score",knn_reg.score(xtrain,ytrain))
      2 print("test score",round(knn_reg.score(xtest,ytest),2))
```

train score 0.9448591585603334

test score 0.88

# Imputation des données

- Amélioration de l'algorithme KNN par un GridsearchCV

```
|: 1 param_grid={"n_neighbors":np.arange(1,9),"metric":["euclidean","manhattan"]}
|: 2 knn_cv_reg=GridSearchCV(KNeighborsRegressor(),param_grid, verbose=1 ,cv=10)
|: 3 knn_cv_reg.fit(xtrain,ytrain)
|: 4 print(knn_cv_reg.best_params_)
|: 5 print(knn_cv_reg.best_score_)
```

```
] : 1 round(knn_cv_reg.score(xtest,ytest),2)
```

0.89

```
1 # J'applique le KNN pour avoir le nutriscore
2
3 predict = knn_cv_reg.predict(df_nutriments_complet_nan.drop("nutrition_score_fr",axis=1))
4 df_nutriments_complet_nan['nutrition_score'] = predict
```

# Imputation des données

- Afin de prédire le nutrition grade, j'utilise un KNN Classifier :

```
1 # Target  
2 y=df_food_complet_full["nutrition_grade_fr"].values  
3 # Les features  
4 x=df_food_complet_full.drop("nutrition_grade_fr",axis=1).values
```

```
35]: 1 # Fonction qui va entraîner le modèle, prédire et évaluer l'ensemble des modèles  
2 def evaluation(model):  
3     model.fit(xtrain, ytrain)  
4     y_pred=model.predict(xtest)  
5     y_true = ytest  
6     print("train score",model.score(xtrain,ytrain))  
7     print("test score",model.score(xtest,ytest))
```

```
040]: 1 model = neighbors.KNeighborsClassifier(n_neighbors=4)
```

```
041]: 1 evaluation(model)
```

```
train score 0.9283406047841132  
test score 0.8822934719815135
```

# Imputation des données

```
1045]: 1 from sklearn.neighbors import KNeighborsClassifier  
2 param_grid={"n_neighbors":np.arange(1,20),"metric":["euclidean","manhattan"]}  
3 knn_cv_class=GridSearchCV(KNeighborsClassifier(),param_grid, verbose=1 ,cv=10)  
4 knn_cv_class.fit(xtrain,ytrain)  
5 print(knn_cv_class.best_params_)  
6 print(knn_cv_class.best_score_)
```

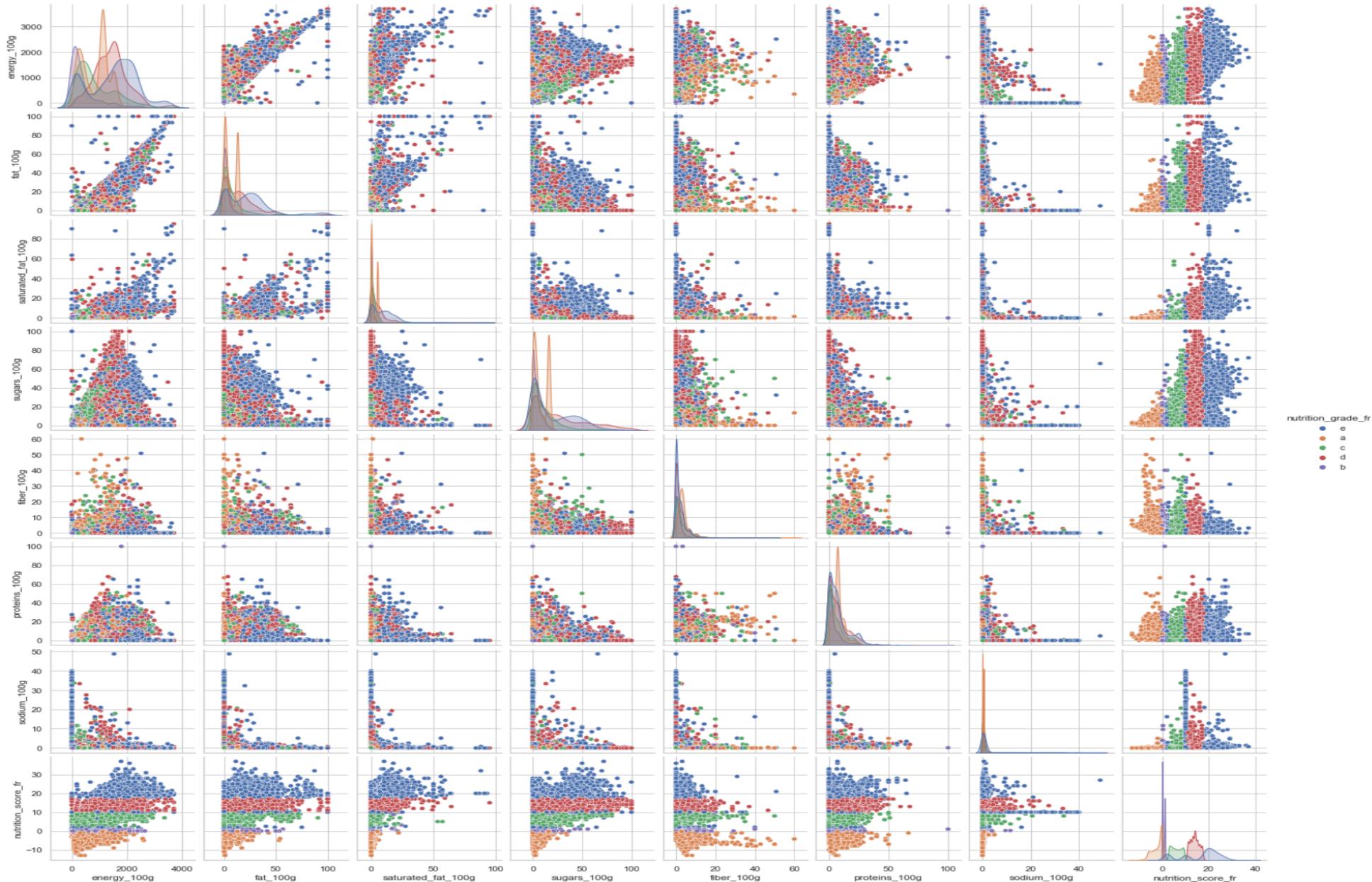
```
Fitting 10 folds for each of 38 candidates, totalling 380 fits  
{'metric': 'manhattan', 'n_neighbors': 1}  
0.9022175556017684
```

```
1 # J'applique le KNN pour avoir le nutrigrade  
2 predict = knn_cv_class.predict(df_food_complet_nan.drop("nutrition_grade_fr",axis=1))  
3 df_food_complet_nan['nutrition_grade'] = predict
```

# Imputation des données

	energy_100g	fat_100g	saturated_fat_100g	sugars_100g	fiber_100g	proteins_100g	sodium_100g	nutrition_score_fr	nutrition_grade_fr
0	2243.0	28.570000		28.57	14.29	3.60	3.57	0.000000	14.0
1	1941.0	17.860000		0.00	17.86	7.10	17.86	0.250000	0.0
2	2540.0	57.140000		5.36	3.57	7.10	17.86	0.482000	12.0
3	2632.0	60.710000		3.57	3.57	10.70	14.29	0.004000	0.0
4	2372.0	50.000000		3.33	6.67	10.00	6.67	0.500000	11.0
5	2372.0	46.670000		8.33	6.67	3.30	16.67	0.400000	16.0
6	2372.0	43.330000		6.67	6.67	3.30	16.67	0.533000	15.0
7	1887.0	21.570000		3.92	3.92	9.80	11.76	0.108000	-1.0
8	1674.0	9.090000		0.91	29.09	7.30	10.91	0.018000	0.0
9	2761.0	65.000000		57.00	7.00	16.00	6.00	0.037000	14.0
10	1941.0	17.860000		2.68	21.43	7.10	10.71	0.018000	6.0

# Analyse bi variée



# Anova

- L'analyse de la variance (ANOVA) est une méthode analytique qui sert à mettre en avant des différences ou des dépendances entre plusieurs groupes statistiques
- On effectue une analyse de variance pour mesurer l'indépendance entre une variable qualitative et une quantitative.
- Je teste l'indépendance entre la variable qualitative nutrigrade et les variables quantitatives

```
Anova avec la variable energy_100g
      sum_sq      df      F  PR(>F)
nutrition_grade_fr 3.859683e+10      4.0 21765.609471 0.0
Residual           1.162900e+11  262314.0          NaN    NaN

Anova avec la variable fat_100g
      sum_sq      df      F  PR(>F)
nutrition_grade_fr 1.384282e+07      4.0 14903.167025 0.0
Residual            6.091266e+07  262314.0          NaN    NaN

Anova avec la variable saturated_fat_100g
      sum_sq      df      F  PR(>F)
nutrition_grade_fr 3.374984e+06      4.0 19681.7941 0.0
Residual            1.124523e+07  262314.0          NaN    NaN

Anova avec la variable sugars_100g
      sum_sq      df      F  PR(>F)
nutrition_grade_fr 1.577066e+07      4.0 11270.589358 0.0
Residual            9.176241e+07  262314.0          NaN    NaN

Anova avec la variable fiber_100g
      sum_sq      df      F  PR(>F)
nutrition_grade_fr 1.754568e+05      4.0 2781.096024 0.0
Residual            4.137288e+06  262314.0          NaN    NaN

Anova avec la variable proteins_100g
      sum_sq      df      F  PR(>F)
nutrition_grade_fr 3.891343e+05      4.0 1608.953242 0.0
Residual            1.586053e+07  262314.0          NaN    NaN

Anova avec la variable sodium_100g
      sum_sq      df      F  PR(>F)
nutrition_grade_fr 3.292343e+04      4.0 1347.479461 0.0
Residual            1.602302e+06  262314.0          NaN    NaN

Anova avec la variable nutrition_score_fr
      sum_sq      df      F  PR(>F)
nutrition_grade_fr 1.484314e+07      4.0 175239.231399 0.0
Residual            5.554640e+06  262314.0          NaN    NaN
```

- H<sub>0</sub> : Les moyennes de chaque groupe sont égales si p-value > 5%
- H<sub>1</sub> : Les moyennes de chaque groupe ne sont pas toutes égales < 5%
- D'après le test les moyennes de chaque groupe ne sont pas toutes égales

- Les moyennes de chaque groupe ne sont pas toutes égales car p-value < 5%

# ANOVA

- la pertinence de ce test repose sur la validation de plusieurs hypothèses :
- l'indépendance entre les échantillons de chaque groupe
- l'égalité des variances que l'on peut vérifier avec un test de Bartlett.
- la normalité des résidus avec un test de Kolmogorov.

# Anova

- Le test de Bartlett qui permet de tester si les variances sont significativement différentes ou non avec :  
H0 : Les variances de chaque groupe sont égales si p-value > 5%  
H1 : Les variances de chaque groupe ne sont pas toutes égales < 5%

```
56]: 1 from scipy.stats import bartlett
2 for col in df_food_app.select_dtypes(exclude=['object']):
3     bar=bartlett(df_food_app[col][df_food_app.nutrition_grade_fr == 'a'],
4                  df_food_app[col][df_food_app.nutrition_grade_fr == 'b'],
5                  df_food_app[col][df_food_app.nutrition_grade_fr == 'c'],
6                  df_food_app[col][df_food_app.nutrition_grade_fr == 'd'],
7                  df_food_app[col][df_food_app.nutrition_grade_fr == 'e'])
8     print("Variances avec la variable ",col," \n",bar," \n")
```

```
Variances avec la variable energy_100g
BartlettResult(statistic=19562.933878041422, pvalue=0.0)
```

```
Variances avec la variable fat_100g
BartlettResult(statistic=97468.89898083298, pvalue=0.0)
```

```
Variances avec la variable saturated_fat_100g
BartlettResult(statistic=204209.5145432176, pvalue=0.0)
```

```
Variances avec la variable sugars_100g
BartlettResult(statistic=141931.28030998132, pvalue=0.0)
```

```
Variances avec la variable fiber_100g
BartlettResult(statistic=26959.219691268943, pvalue=0.0)
```

```
Variances avec la variable proteins_100g
BartlettResult(statistic=13010.238242109785, pvalue=0.0)
```

```
Variances avec la variable sodium_100g
BartlettResult(statistic=344778.1990802859, pvalue=0.0)
```

```
Variances avec la variable nutrition_score_fr
BartlettResult(statistic=281162.5731600907, pvalue=0.0)
```

- Les variances de chaque groupe ne sont pas toutes égales car la p-value est < à 5%

# Anova

- Normalité des résidus : C'est une des 3 conditions de validité d'une ANOVA. L'objectif est de s'assurer que les résidus suivent une loi normale afin de ne pas affirmer qu'il existe une différence de moyenne entre les groupes qui serait causée par le hasard.
- J'utilise le test de Kolmogorov pour tester la normalité des résidus où :
- H<sub>0</sub> : Les résidus suivent une loi normale si p-value > 5% H<sub>1</sub> : Les résidus ne suivent pas une loi normale si p-value < 5%

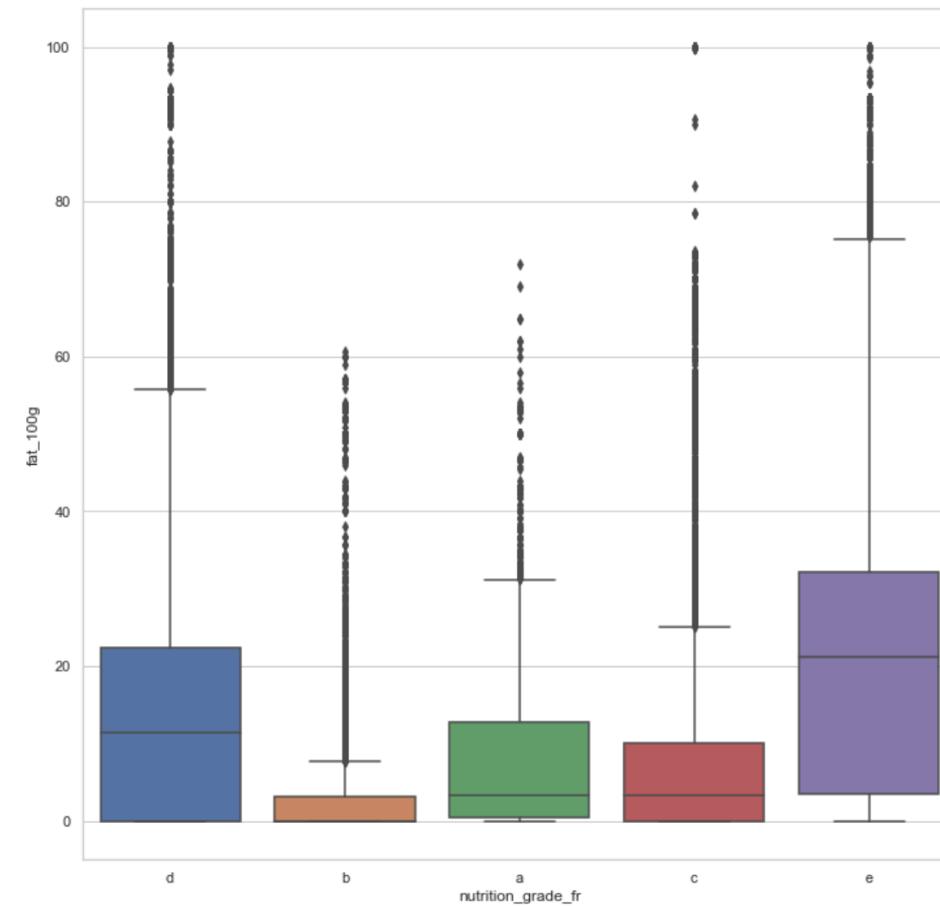
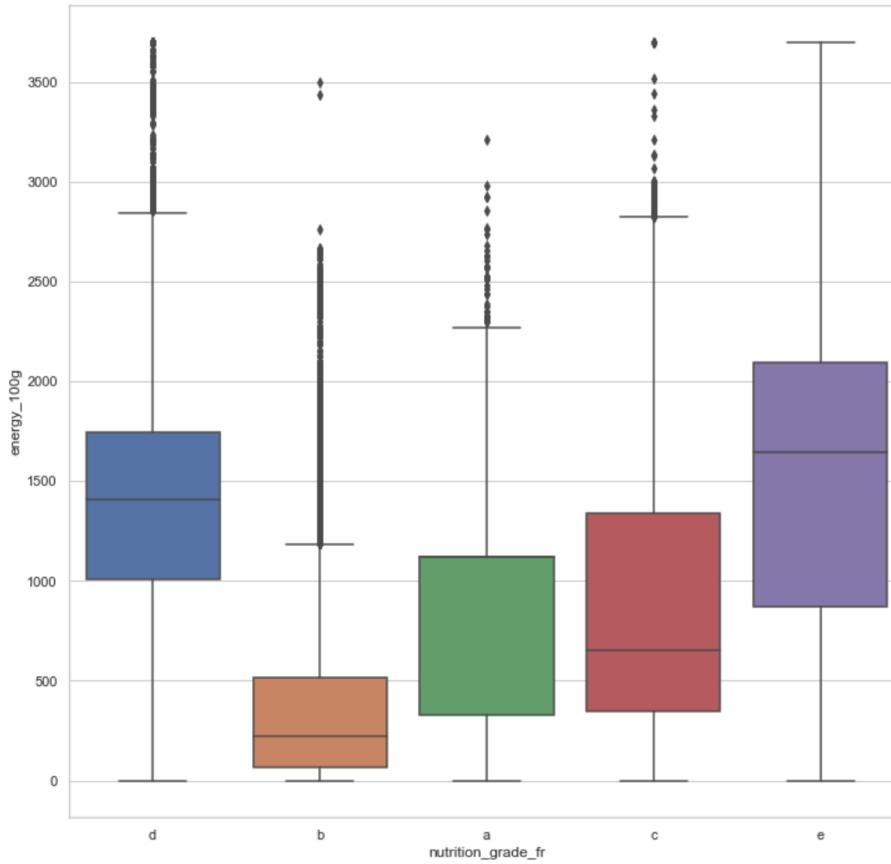
## Test de normalité Kolmogorov Smirnov

```
[1]: alpha=0.05
[2]: for col in rows:
[3]:     print([col])
[4]:     a,b= stats.kstest( rows[[col]],'norm' )
[5]:     print("statistics",a,"p-value",b)
[6]:     if b < alpha:
[7]:         print("H0 rejetée")
[8]:     else:
[9]:         print("H0 non rejetée")

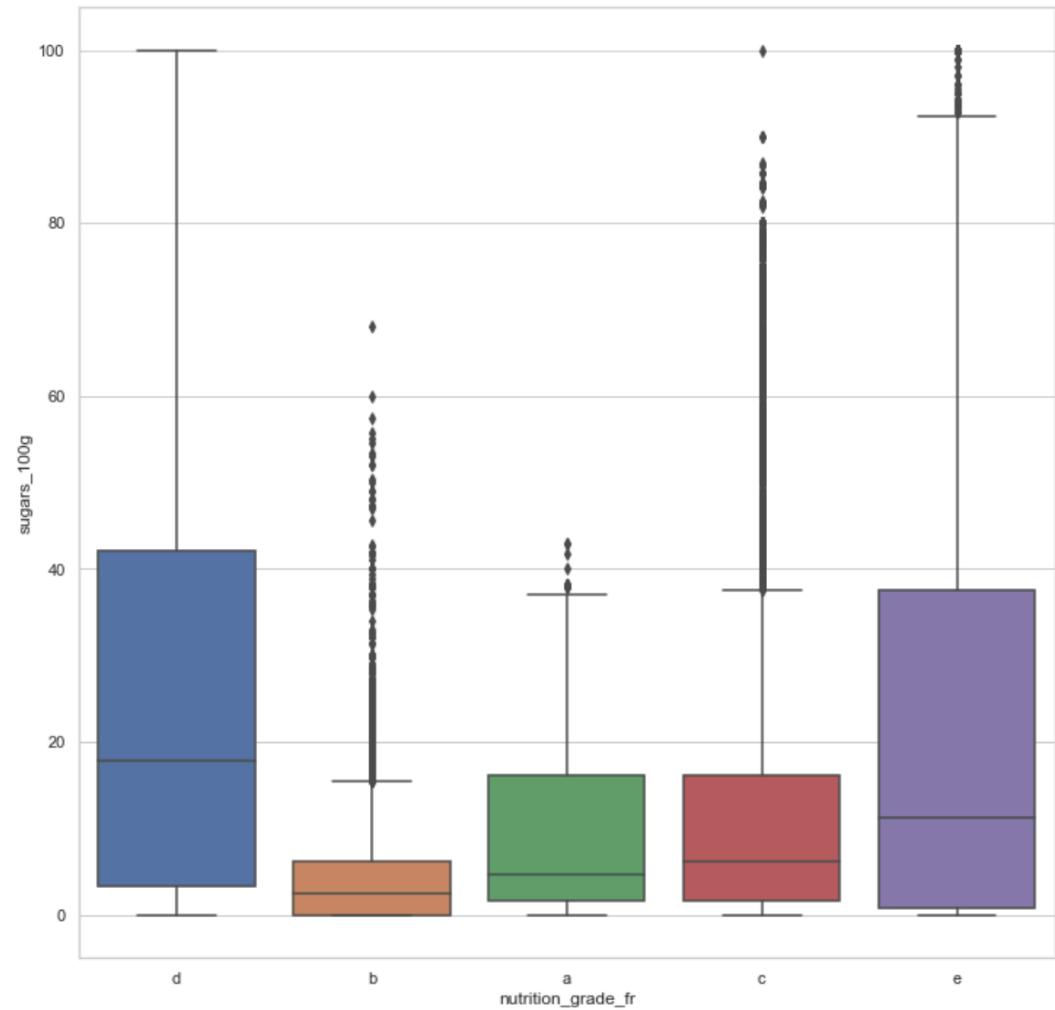
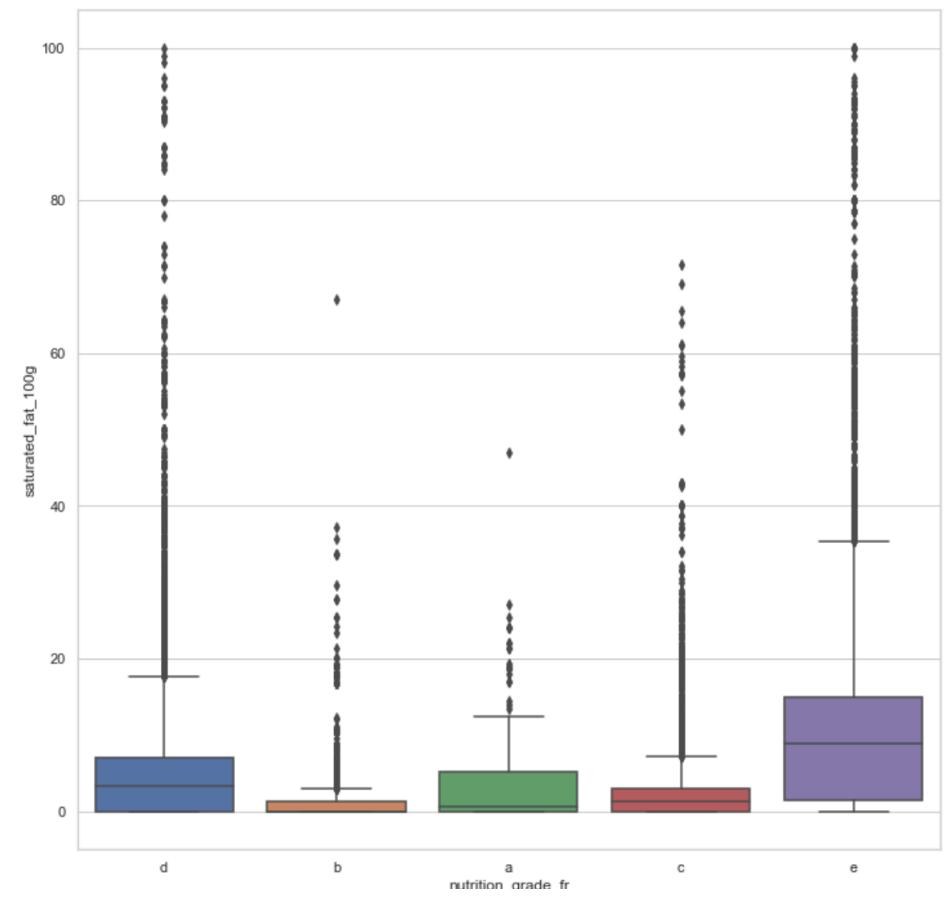
['energy_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['fat_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['saturated-fat_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['trans-fat_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['sugars_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['fiber_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['proteins_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['sodium_100g']
statistics 1.0 p-value 0.0
H0 rejetée
['nutrition_score_fr']
statistics 1.0 p-value 0.0
H0 rejetée
```

# Anova

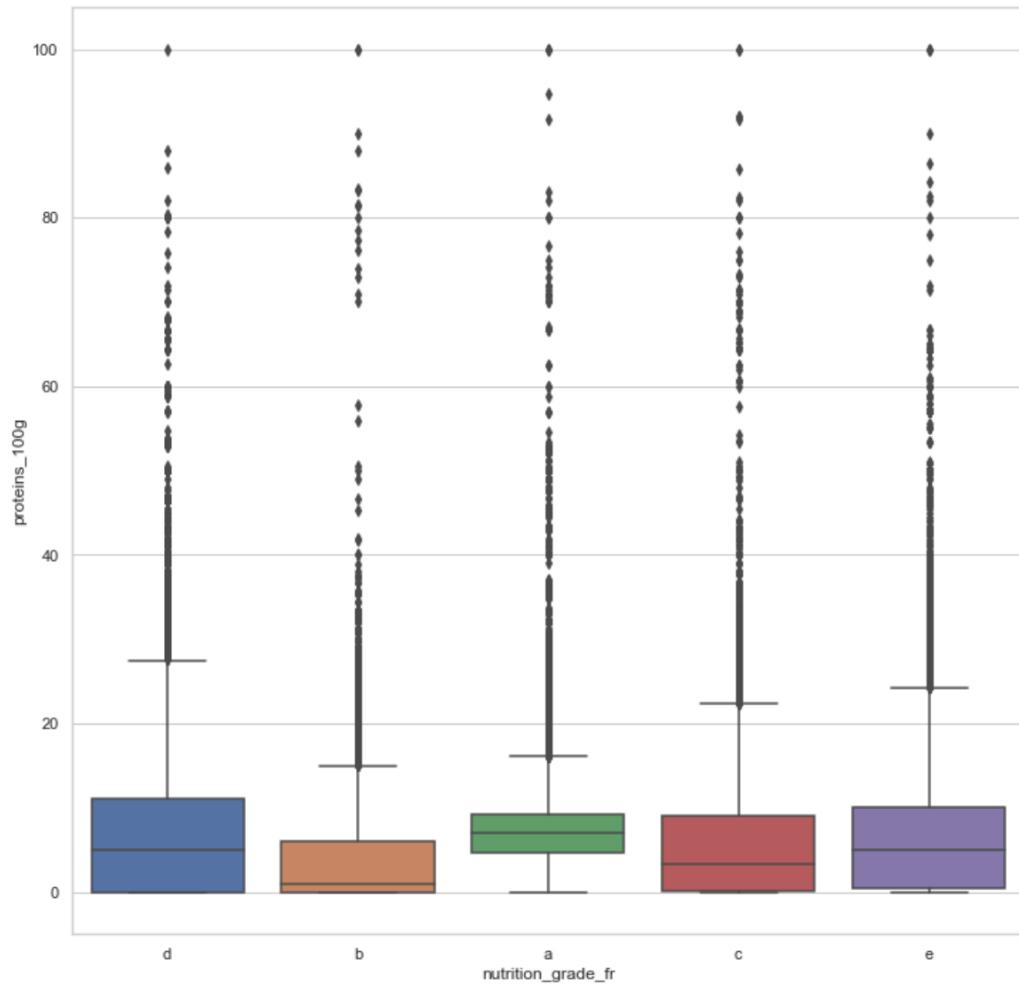
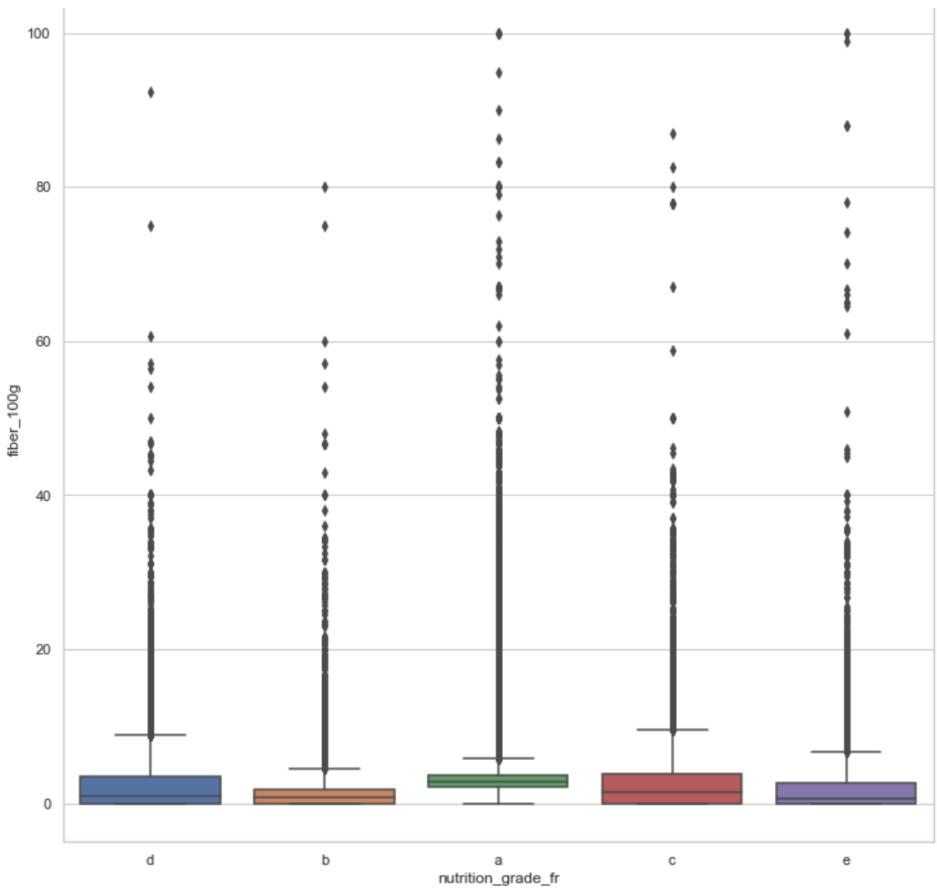
- Cas de variances inégales entre chaque groupe



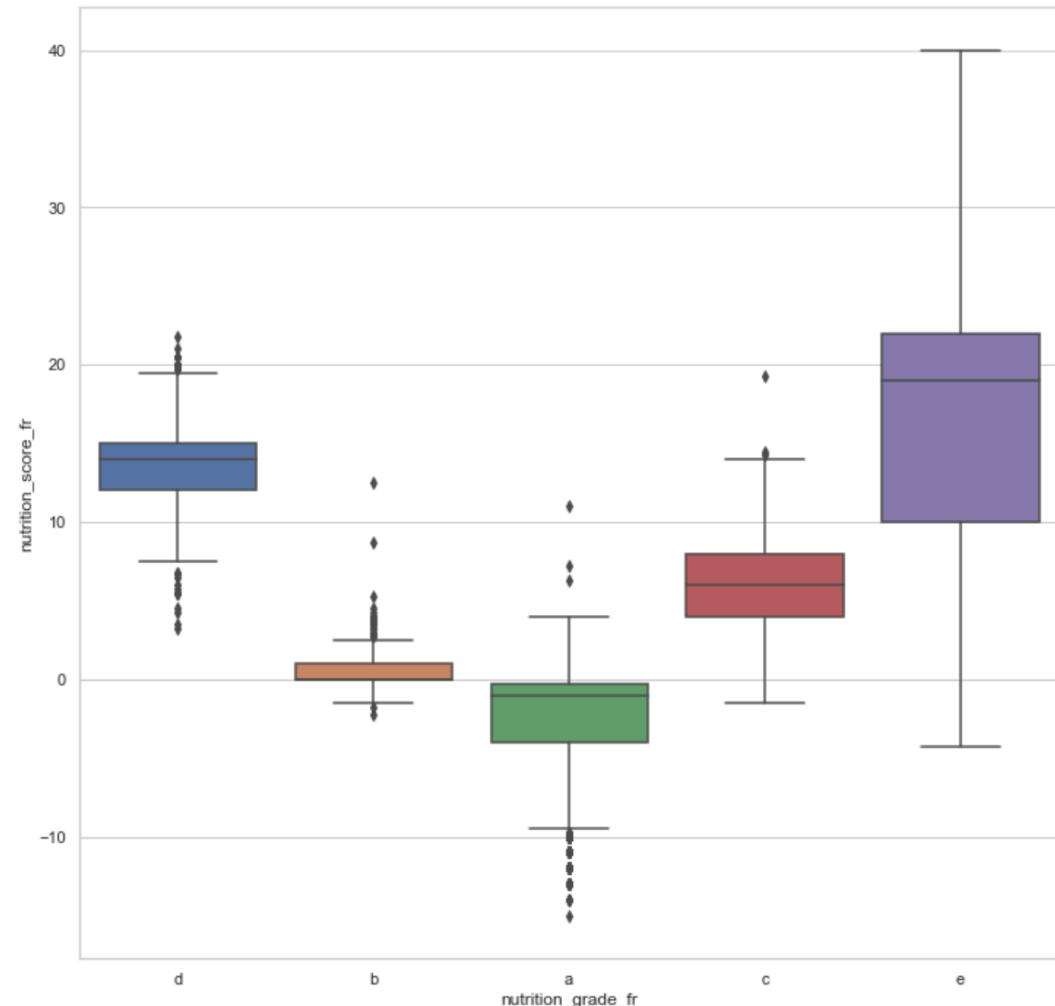
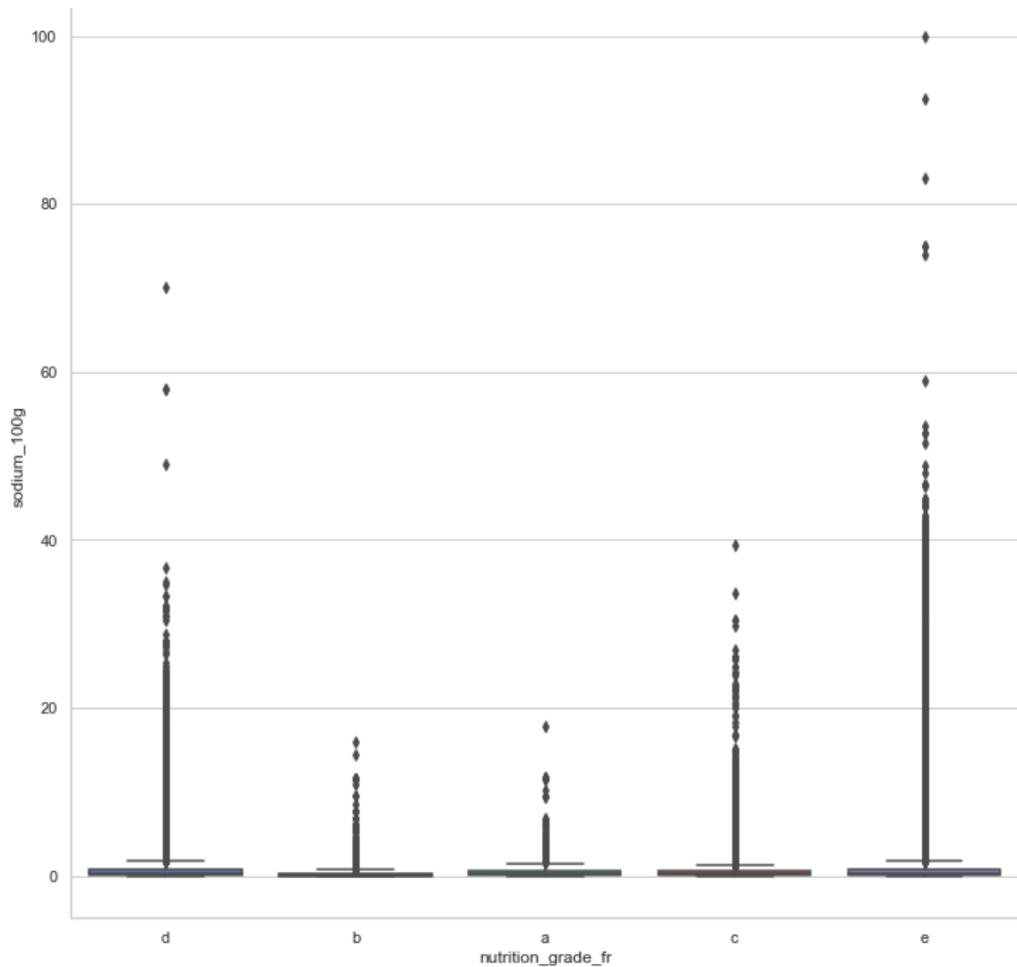
# Anova



# Anova



# Anova



# Anova

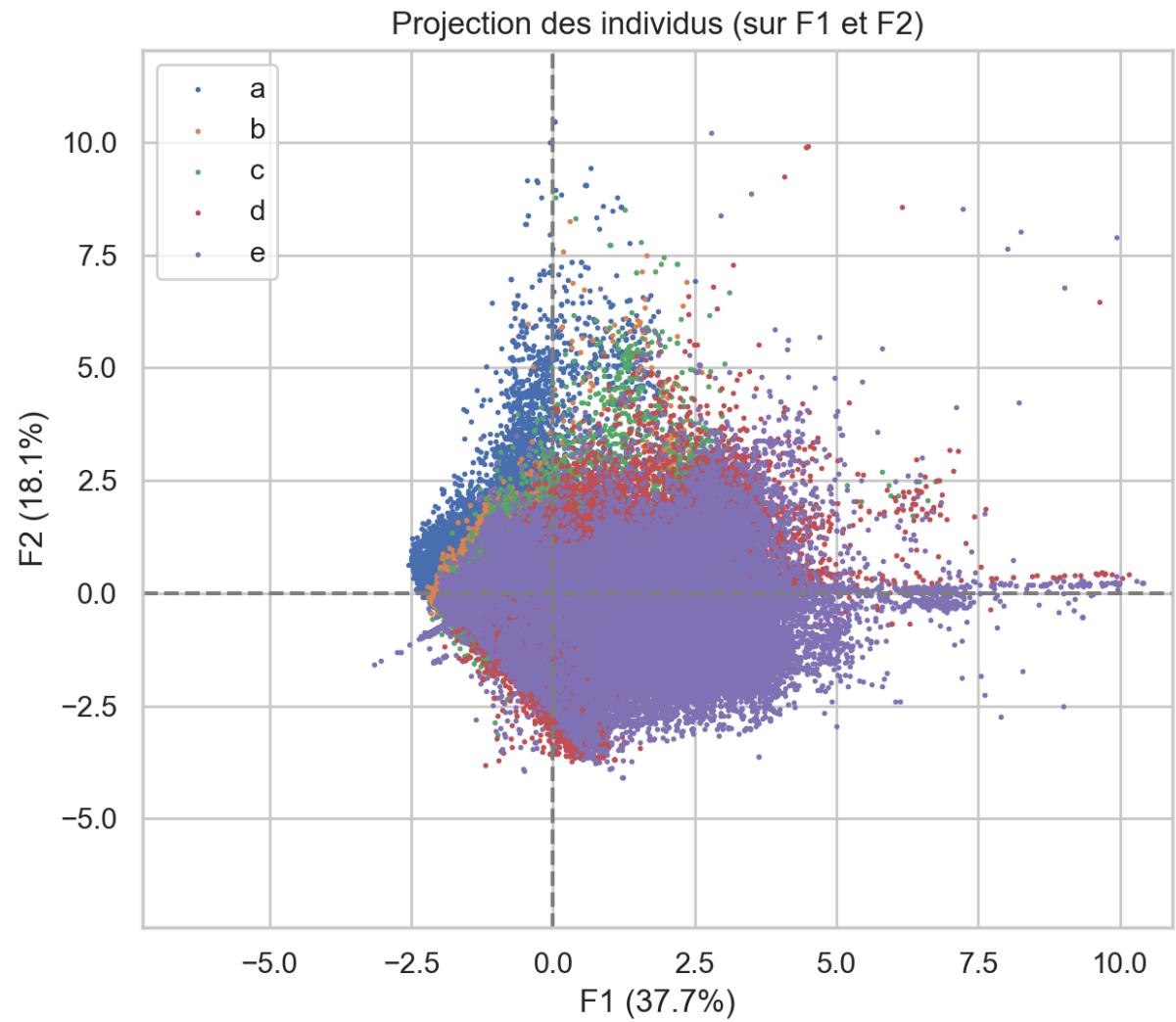
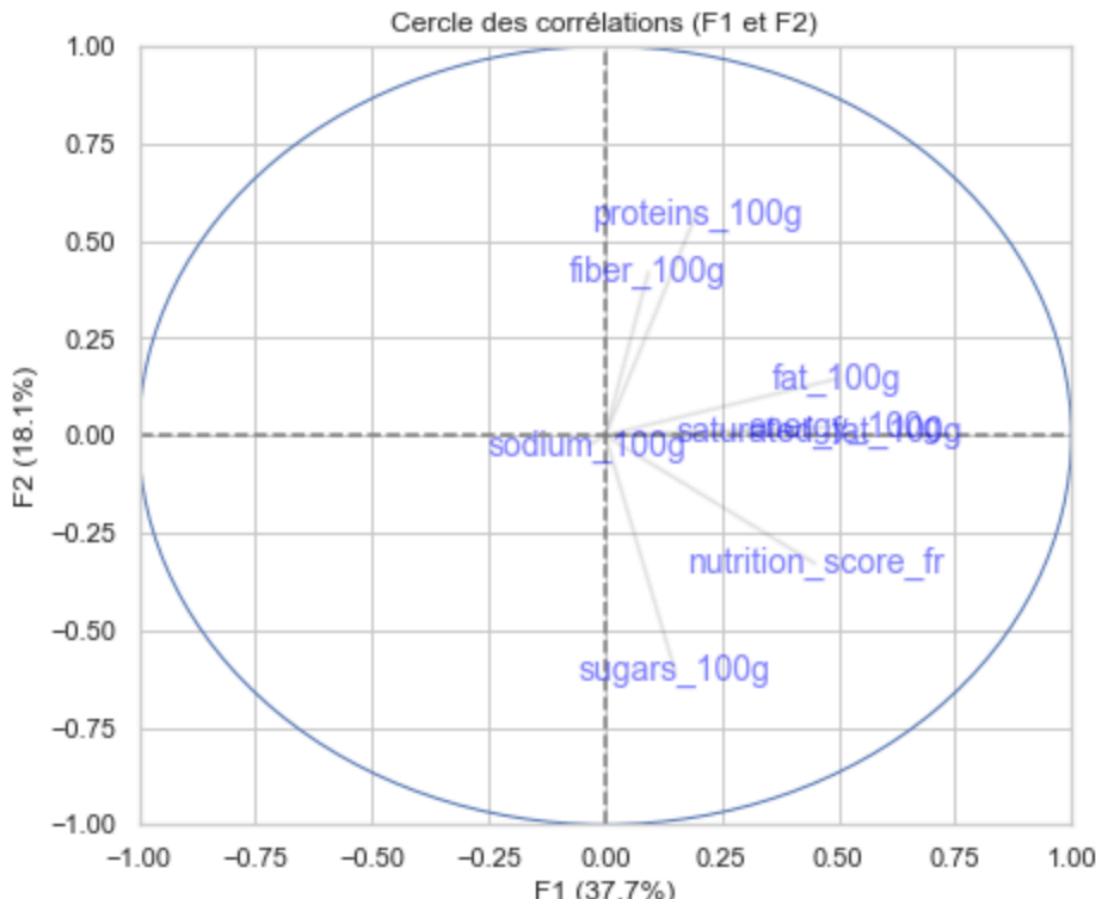
- Comme alternative de l'anova, je fais un test de Kruskal qui est un test non paramétrique

```
191]: 1 for col in df_food_app.columns:  
2     kstat, pval = scipy.stats.kruskal(*[group[col].values for name, group in df_food_app.groupby("nutrition_gra  
3     print(pval)  
  
0.0
```

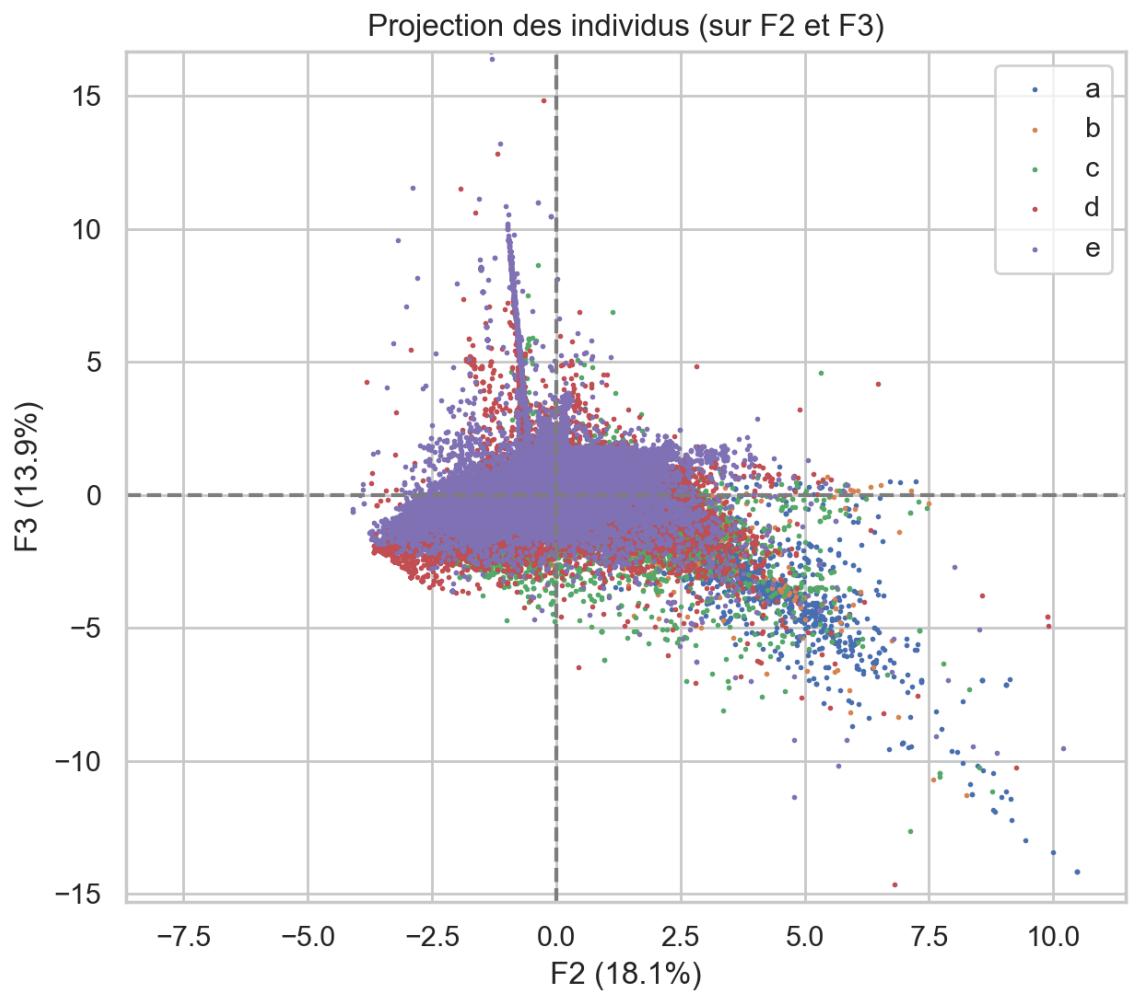
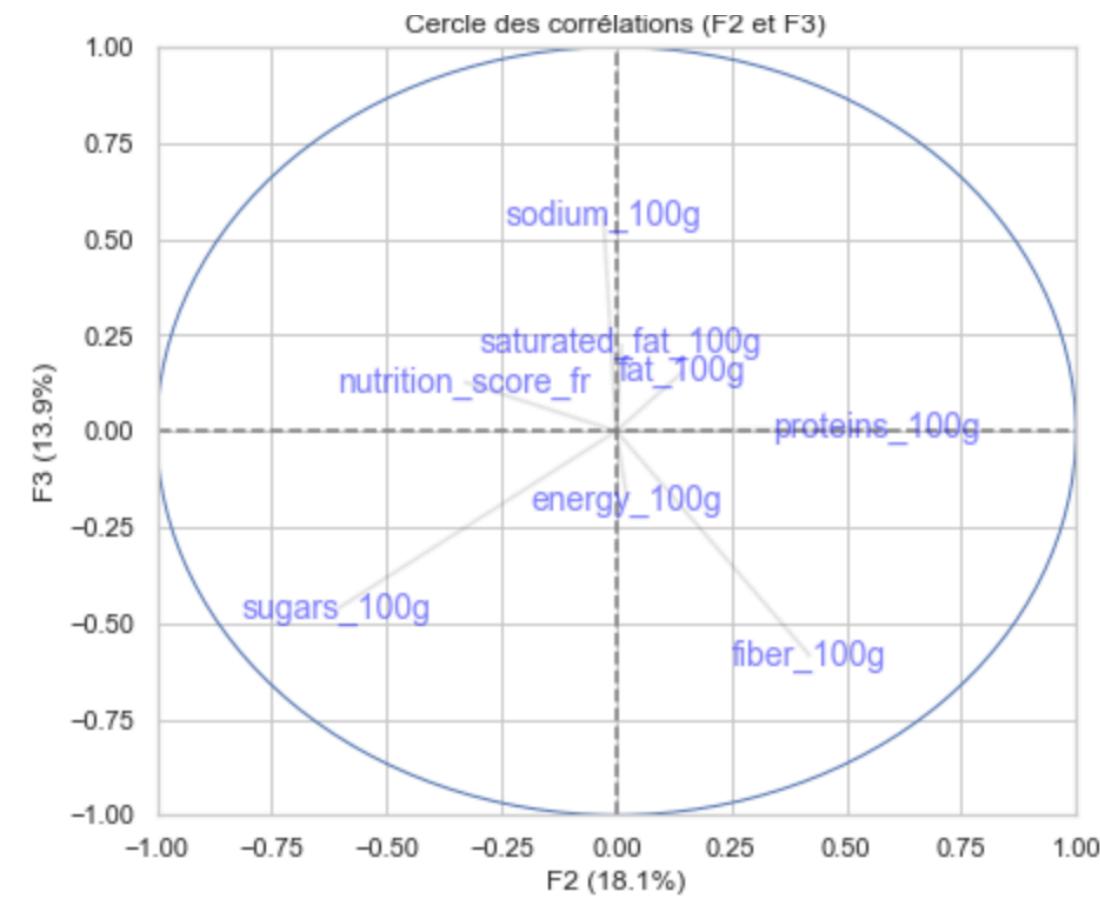
---

- La p-value est inférieure à 0.05, alors on peut dire qu'il y a une différence significative

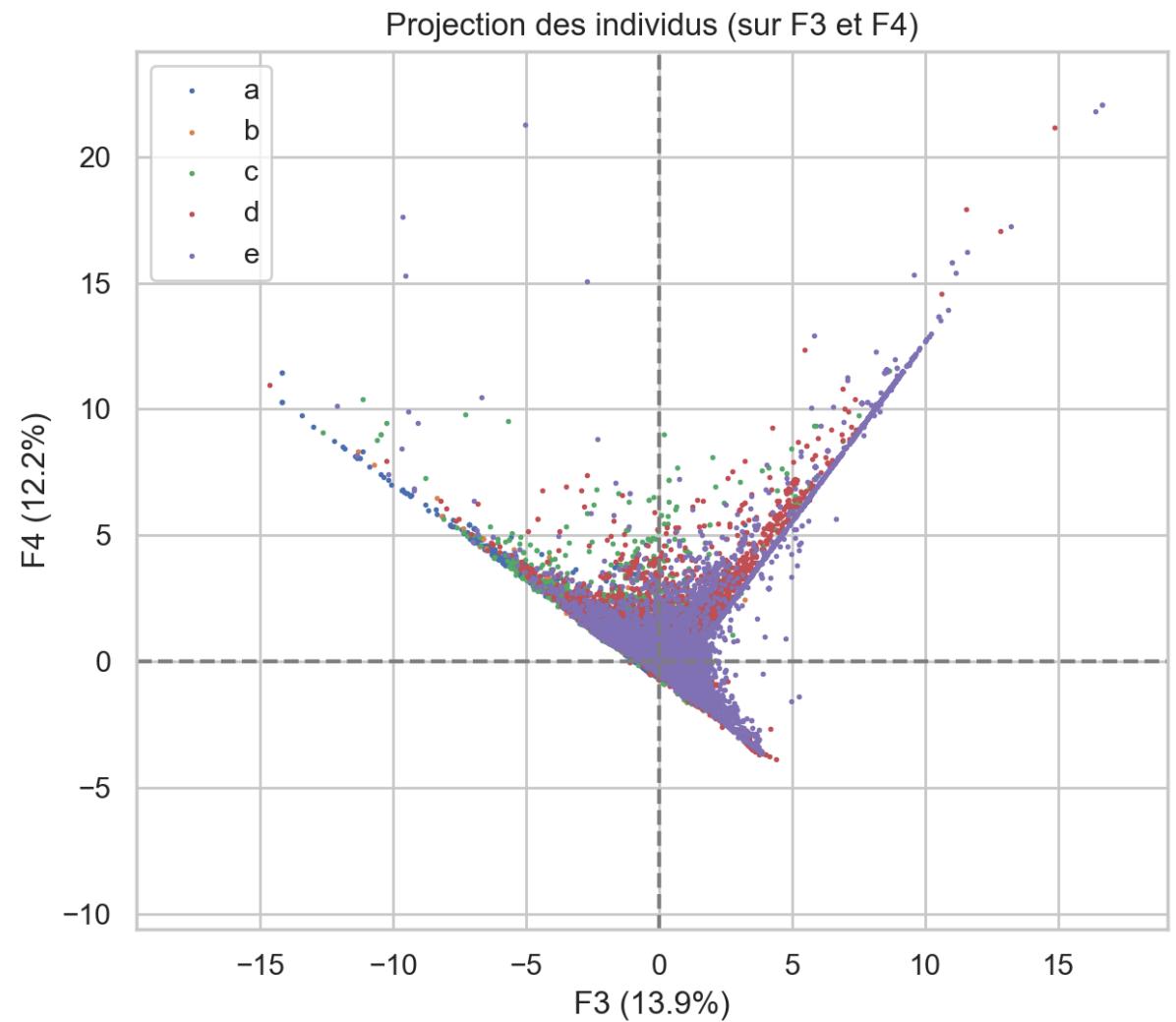
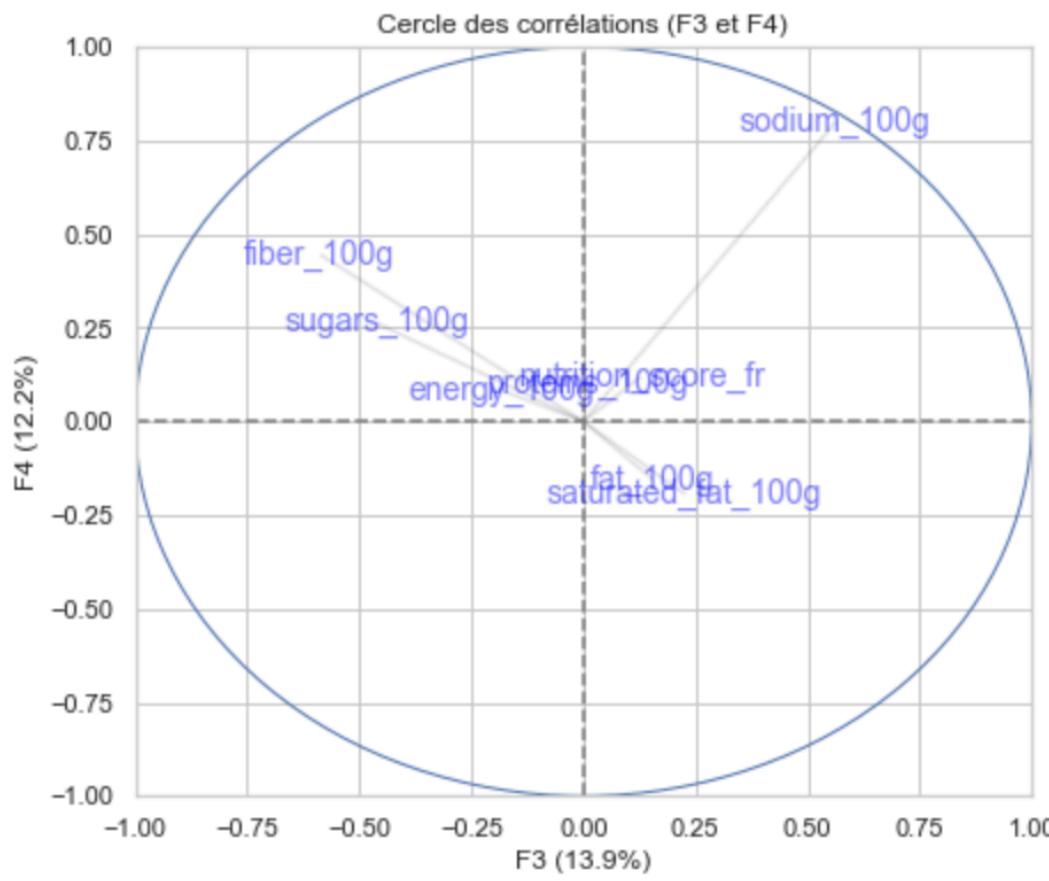
# ACP



# ACP



# ACP



# Application de santé

	additives	TDAH	SAIN	global_score	appreciation	qualite	defaut
	NON	OUI		75	Très Bien	Faible en sel-Pas d'additifs-Absence huile de palme-Produit sain	---Produit Gras--Gras saturé élevé
	NON	NON		80	Très Bien	Faible en sel-Pas d'additifs-Absence huile de palme-	-----
	NON	NON		80	Très Bien	Faible en sel-Pas d'additifs-Absence huile de palme-	-----
	NON	NON		80	Très Bien	Faible en sel---	-----
E330,E951,E150,E150,E950,E552,E102,E129	Risque	NON		49	Médiocre	Faible en sel--Absence huile de palme-	-Présence d'additifs--Produit à risque-Gras saturé élevé

# Conclusion

- Le jeu de données présentait plusieurs données comportant beaucoup de données manquantes ainsi que des erreurs dans les valeurs des variables.
- Le nettoyage du data frame , rectification des textes ainsi que l'ensemble des méthodes d'imputation des valeurs manquantes ont permis d'avoir des données exploitables afin de répondre au projet et exploiter les données en proposant une idée d'application de santé.



# Annexes

- [https://ciqual.anses.fr/#/constituants/10004/sel-chlorure-de-sodium-\(g-100-g\)](https://ciqual.anses.fr/#/constituants/10004/sel-chlorure-de-sodium-(g-100-g))
- <https://www.passeportsante.net/portail/nutriments>
- <https://sante.journaldesfemmes.fr/calories/classement/aliments/acides-gras-satures>
- <https://www.quechoisir.org/action-ufc-que-choisir-additifs-alimentaires-87-molecules-a-eviter-n59897/>
- <https://quoidansmonassiette.fr/comment-est-calcule-le-nutri-score-logo-nutritionnel/>

# Discussion :



**FIN**

• **MERCI**