# Projet 3 : Concevez une application de la santé publique .

Ilham NOUMIR | Parcours Data Science | Date : 16/08/2021

# Contexte du projet :

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

## Mission :

1. Traitement du jeu de données afin de repérer des variables pertinentes pour les traitements à venir.
2. Production  des visualisations afin de mieux comprendre les données.
3. Analyse multivariée et production  des tests statistiques appropriés.
4. Elaboration des idées d'application .
5. Rédaction d'un rapport d'exploration.

# Plan de présentation :


1. Idée d'application
2. Nettoyage des données
3. Analyse des données
4. Conclusion

# 1. Idée d’application :

Prévoir le nutri score pour un utilisateur qui cherche à savoir la valeur des aliments achetés avant la consommation

# 2. Nettoyage des données :

**Jeu de données initial**

**Dimension :**
(1856452, 186)
**%Valeurs manquantes:**
79.68%

**10 Fonctions pour le nettoyage des données**

**Jeu de données après nettoyage**

**Dimension :**
(659180, 27)
**%Valeurs manquantes:**
18.62%

**Une fonction globale qui automatise le nettoyage des données**

# 2. Nettoyage des données :

1

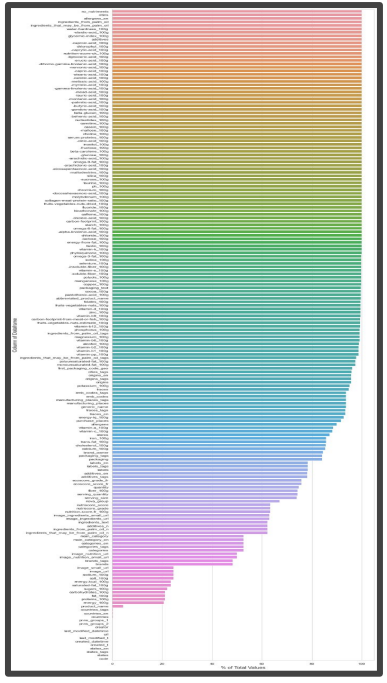
Vue globale de la qualité des données :

```
def nombre_percentage_nan(df):
    total_NaN= df.isnull().sum().sum()
    print('Nombre total des NaN:', total_NaN)
    percentage_NaN= round(total_NaN*100/((df.shape[0]*df.shape[1])),2)
    print('Percentage total des NaN:', percentage_NaN)
```

```
Nombre total des NaN: 275140549
Percentage total des NaN: 79.68
```

# 2. Nettoyage des données :

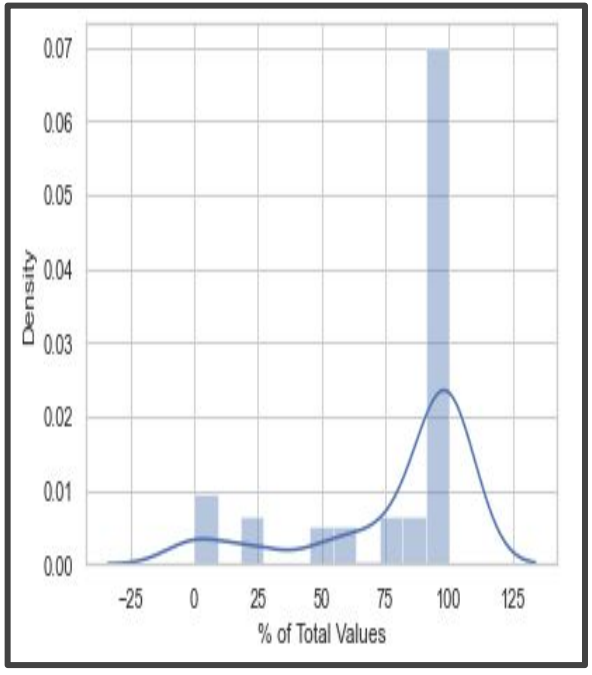
## Vue globale du pourcentage de NaN par colonne



## Dataframe du pourcentage de NaN par colonne

| | index | % of Total Values |
|---|---|---|
| 0 | no_nutriments | 100.000000 |
| 1 | cities | 100.000000 |
| 2 | allergens_en | 100.000000 |
| 3 | ingredients_from_palm_oil | 100.000000 |
| 4 | ingredients_that_may_be_from_palm_oil | 100.000000 |
| 5 | water-hardness_100g | 99.999946 |
| 6 | -elaidic-acid_100g | 99.999892 |
| 7 | glycemic-index_100g | 99.999785 |
| 8 | additives | 99.999785 |
| 9 | -caproic-acid_100g | 99.999731 |
| 10 | chlorophyl_100g | 99.999731 |
| 11 | -caprylic-acid_100g | 99.999731 |
| 12 | nutrition-score-uk_100g | 99.999569 |
| 13 | -lignoceric-acid_100g | 99.999569 |
| 14 | -erucic-acid_100g | 99.999515 |
| 15 | -dihomo-gamma-linolenic-acid_100g | 99.999407 |
| 16 | -nervonic-acid_100g | 99.999407 |
| 17 | -capric-acid_100g | 99.999407 |
| 18 | -stearic-acid_100g | 99.999354 |
| 19 | -cerotic-acid_100g | 99.999300 |
| 20 | -melissic-acid_100g | 99.999300 |
| 21 | -myristic-acid_100g | 99.999246 |
| 22 | -gamma-linolenic-acid_100g | 99.999246 |
| 23 | -mead-acid_100g | 99.999138 |
| 24 | -lauric-acid_100g | 99.999084 |
| 25 | -montanic-acid_100g | 99.998869 |
| 26 | -palmitic-acid_100g | 99.998492 |
| 27 | -butyric-acid_100g | 99.998492 |
| 28 | -gondoic-acid_100g | 99.998169 |

## Répartition du pourcentage des NaN

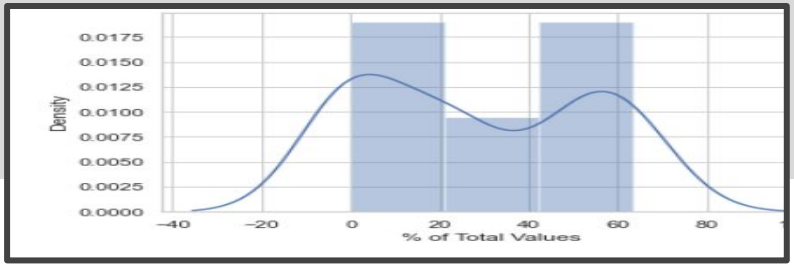


# 2. Nettoyage des données :

2

Traitement des colonnes dépassant un taux de NAN de 65 % :

```
def delete_nan(data, taux_nan_admis):
    missing= pd.DataFrame((((data.isnull().sum()/len(data))*100 ).sort_values(ascending=False))).reset_index()
    missing.rename(columns = {0:'% of Total Values'} , inplace= True)
    missing= missing.loc[missing['% of Total Values']> taux_nan_admis]
    missing= missing.reset_index().rename(columns= {'index':'name_of_colone'})
    for row in missing.itertuples():
        name= row.name_of_colone
        data.drop([name], axis='columns', inplace=True)
    return data
```



# 2. Nettoyage des données :

Traitement des colonnes non pertinents à l'étude :

3

```
def delete_non_relevant_column(data):
    searchfor= ['url','Unnamed','states','image']
    for i in data.loc[:, data.columns.str.contains('|'.join(searchfor))].columns:
        data.drop(columns=i, inplace=True)
    return data
```

# 2. Nettoyage des données :

4

Traitement des colonnes format Date :

```
def convet_datetime(data):
    for col in data.columns:
        if col.endswith('_t'):
            data[col]= pd.to_datetime(data[col], unit='s')
        elif col.endswith('_datetime'):
            data[col]= pd.to_datetime(data[col], infer_datetime_format=True, format = "%Y-%m-%dT%h:%M:%s").dt.tz_localiz
    return data
```

5

Correction des types de données :

```
def convert_to_category(df):
    categories_columns= ['ingredients_that_may_be_from_palm_oil_n','ingredients_from_palm_oil_n']
    for column in categories_columns:
        df[column]= df[column].astype('category')
    return df
```

# 2. Nettoyage des données :

6

Choix de la zone d'étude (La france) :

```python
def tri_pays(dataframe):
    '''Réduction du dataframe à la France'''
    liste_pays = ['France', 'FR', 'en:FR', 'en:fr', 'en:France', 'Frankreich',
                  'france', 'Réunion', 'Francia', 'French Polynesia', 'Frankrijk',
                  'Nouvelle-Calédonie', 'Martinique', 'Guadeloupe',
                  'Polynésie Française', 'Mayotte']
    return dataframe[dataframe['countries'].isin(liste_pays)]
```

7

Traitement des colonnes ayant des informations redondantes :

```python
def remove_columns(df):
    colone_a_supprimer = ['last_modified_datetime', 'created_datetime' ,'categories_tags','categories_en'
                          ,'brands_tags','countries_tags' ,'countries_en','origins_tags','traces_tags' ,
                          'traces_en', 'main_category_en']
    for col in colone_a_supprimer :
        if col in df.columns :
            df.drop(columns=[col],axis='columns', inplace=True)
    return df
```

# 2. Nettoyage des données :

| | energy_100g | fat_100g | saturated-fat_100g | carbohydrates_100g | sugars_100g | proteins_100g | salt_100g | sodium_100g | nutrition-score-fr_100g |
|---|---|---|---|---|---|---|---|---|---|
| count | 5.276660e+05 | 523912.000000 | 527379.000000 | 523772.000000 | 526466.000000 | 525538.000000 | 511018.000000 | 511017.000000 | 236940.000000 |
| mean | 1.170565e+03 | 14.235537 | 5.425011 | 26.914975 | 13.610135 | 9.154988 | 1.275964 | 0.510508 | 9.453305 |
| std | 1.115756e+04 | 43.651358 | 8.388901 | 266.687252 | 42.226931 | 101.180241 | 20.303175 | 8.121605 | 8.759001 |
| min | 0.000000e+00 | 0.000000 | 0.000000 | -1.000000 | -1.000000 | 0.000000 | 0.000000 | 0.000000 | -15.000000 |
| 25% | 4.730000e+02 | 1.000000 | 0.200000 | 2.300000 | 0.600000 | 1.500000 | 0.060000 | 0.024000 | 2.000000 |
| 50% | 1.088000e+03 | 8.000000 | 2.000000 | 13.300000 | 3.300000 | 6.300000 | 0.550000 | 0.220000 | 10.000000 |
| 75% | 1.669000e+03 | 22.000000 | 8.000000 | 51.000000 | 19.000000 | 13.000000 | 1.300000 | 0.520000 | 16.000000 |
| max | 8.010000e+06 | 29000.000000 | 2000.000000 | 192000.000000 | 27000.000000 | 73000.000000 | 14000.000000 | 5600.000000 | 40.000000 |

# 2. Nettoyage des données :

8

## Traitement des valeurs négatives :

```
def negative_values(df) :
    for column in df.select_dtypes(include = ['int32','float64']).columns :
        df.loc[df[column] < 0] = np.nan
    return df
```

9

## Traitement des colonnes 100_g :

```
def column_100g(df) :
    col_100g = [c for c in df.columns if c.endswith('_100g') and c!='energy_100g'
                and c!='energy-kj_100g' and c!='energy-kcal_100g'
                and c!='nutrition-score-fr_100g']
    for i in range(len(col_100g)):
        colonne = col_100g[i]
        mask = df[colonne] > 100.0
        df = df.drop(df[mask].index)
    return df
```

# 2. Nettoyage des données :

10

Traitement des outliers :

```
def treat_outliers_std(df) :

    numeric_columns= df.select_dtypes(['float64', 'int64'])
    no_numeric_columns = df.select_dtypes(exclude=['float64', 'int64'])
    data_mean, data_std = np.mean(numeric_columns), np.std(numeric_columns)
    cut_off = data_std * 3
    lower, upper = data_mean - cut_off, data_mean + cut_off
    idx = ~((numeric_columns < lower) | (numeric_columns > upper)).any(axis=1)
    df_final =  pd.concat([numeric_columns.loc[idx], no_numeric_columns.loc[idx]], axis=1)
    return df_final
```

11

Traitement des valeurs manquantes:

```
def impute_NaN_iterative(df):
    numeric_columns= df.select_dtypes(['int64','float64'])
    colone= numeric_columns.columns
    no_numeric_columns= df.select_dtypes(exclude=['int64','float64'])
    no_numeric_columns.reset_index(drop=True,inplace=True)
    imputer = IterativeImputer()
    X= numeric_columns.values
    imputer.fit(X)
    X_trans= imputer.transform(X)
    numeric_columns= pd.DataFrame(X_trans , columns=colone)
    numeric_columns.reset_index(drop=True,inplace=True)
    df_final = pd.concat([numeric_columns, no_numeric_columns], axis= 1)
    df= df_final
    return df
```

# 2. Nettoyage des données : Automatisation du processus de nettoyage

jupyter functions.py il y a une heure

File Edit View Language

```python
##### Ensemble des fonctions pour le nettoyage et la préparation de la dataframe


import numpy as np
import pandas as pd
import seaborn as sns
import datetime

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

#### Calcul du nombre et du pourcentage des NAN :

def nombre_percentage_nan(df):
    total_NaN= df.isnull().sum().sum()
    print('Nombre total des NaN:', total_NaN)
    percentage_NaN= round(total_NaN*100/((df.shape[0]*df.shape[1])),2)
    print('Percentage total des NaN:', percentage_NaN)

#### Suppression des colones avec beaucoup de valeurs manquantes :

def delete_nan(data, taux_nan_admis):
    missing= pd.DataFrame((((data.isnull().sum()/len(data))*100 ).sort_values(ascending=False))).reset_index()
    missing.rename(columns = {0:'% of Total Values'} , inplace= True)
    missing= missing.loc[missing['% of Total Values']> taux_nan_admis]
    missing= missing.reset_index().rename(columns= {'index':'name_of_colone'})
    for row in missing.itertuples():
        name= row.name_of_colone
        data.drop([name], axis='columns', inplace=True)
    return data

#### Suppression des colones non petinents à l'étude :

def delete_non_relevant_column(data):
    searchfor= ['url','Unnamed','states','image']
    for i in data.loc[:, data.columns.str.contains('|'.join(searchfor))].columns:
        data.drop(columns=i, inplace=True)
    return data
```

```python
def total_clean(df):
    print('Le nettoyage Globale de la dataframe .....')
    print('-'*60)
    print(nombre_percentage_nan(df))
    print('La dimension de la dataframe : ' , df.shape)

    print('-'*60)

    print('-'*60)
    print('Nettoyage des colones avec plus de 65 % des NaN')
    try :
        df= delete_nan(df, 65)
    except Exception as e :
        print(e)
        print('Réduction des colones de la dataframe est échoué ')

    print('-'*60)
    print('Mise en conformité du format de la date')
    try :
        df = convet_datetime(df)
    except Exception as e:
        print(e)
        print('Erreur dans la mise en conformité des dates')


    print('-'*60)
    print('Suppression des colones non pertinents')
    try :
        df = delete_non_relevant_column(df)
    except Exception as e  :
        print(e)
        print('Suppression des colones non pertinents a échoué')

    print('-'*60)
    print('Conversion des colones objet en category')
    try :
        df= convert_to_category(df)
    except Exception as e :
        print(e)
        print("La conversion au type 'category' a échoué")
```

# 2. Nettoyage des données : Automatisation du processus de nettoyage

```
%%time
df = total_clean(df)
```

```
Le nettoyage Globale de la dataframe .....
------------------------------------------------------------
Nombre total des NaN: 275140549
Percentage total des NaN: 79.68
None
La dimension de la dataframe :  (1856452, 186)
------------------------------------------------------------
------------------------------------------------------------
Nettoyage des colones avec plus de 65 % des NaN
------------------------------------------------------------
Mise en conformité du format de la date
------------------------------------------------------------
Suppression des colones non pertinents
------------------------------------------------------------
Conversion des colones objet en category
------------------------------------------------------------
Réduction de la dataset
------------------------------------------------------------
Suppression des colones redondants:
------------------------------------------------------------
Traitement des valeurs négatives
------------------------------------------------------------
Traitement des colones 100_g
------------------------------------------------------------
Traitement des outliers
------------------------------------------------------------
Imputation des NaN
------------------------------------------------------------
------------------------------------------------------------
Nombre total des NaN: 3195795
Percentage total des NaN: 18.69
le pourcentage actuel des NaN dans la dataframe None
La dimension de la dataframe :  (633148, 27)
Wall time: 3min 19s
```

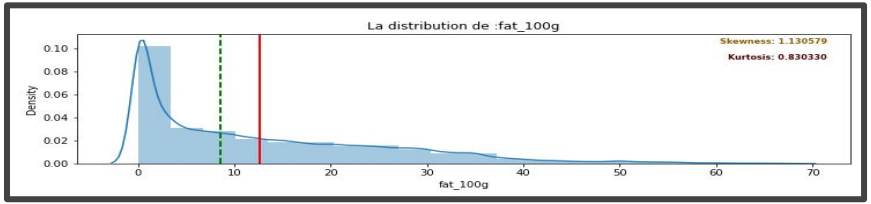
# 3. Analyse des données :

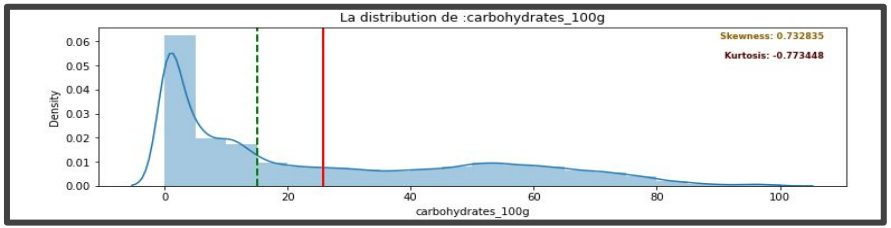
**La description des données numériques : avec la fonction describe()**

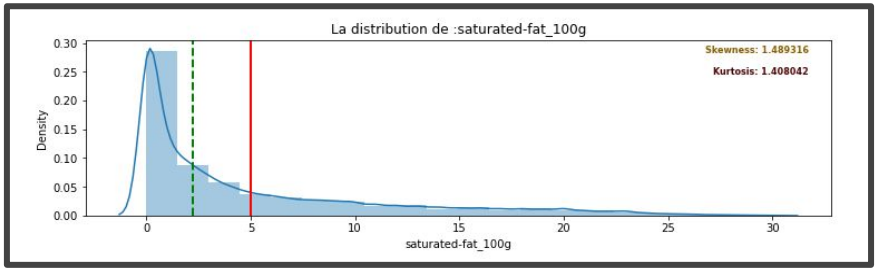
| | fat_100g | saturated-fat_100g | carbohydrates_100g | sugars_100g | proteins_100g |
|---|---|---|---|---|---|
| count | 633146.000000 | 633146.000000 | 633146.000000 | 633146.000000 | 633146.000000 |
| mean | 12.648660 | 4.982705 | 25.704159 | 12.575720 | 8.470834 |
| std | 13.008964 | 6.283067 | 25.566041 | 17.366369 | 7.864868 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 1.400000 | 0.300000 | 2.900000 | 0.700000 | 2.000000 |
| 50% | 8.573333 | 2.200000 | 14.966667 | 3.700000 | 6.333333 |
| 75% | 21.000000 | 7.600000 | 48.000000 | 18.400000 | 12.800000 |
| max | 67.600000 | 29.900000 | 100.000000 | 75.500000 | 39.100000 |

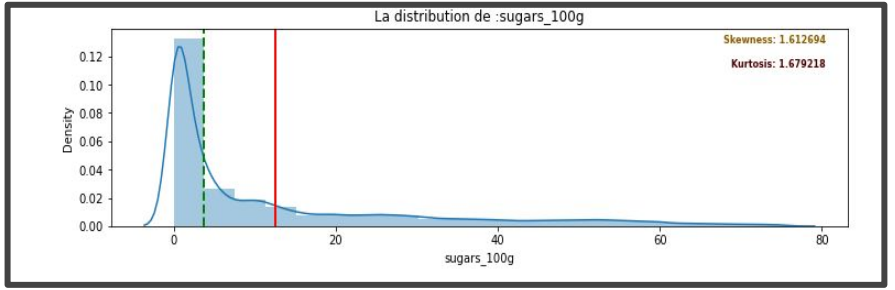
# 3. Analyse des données :

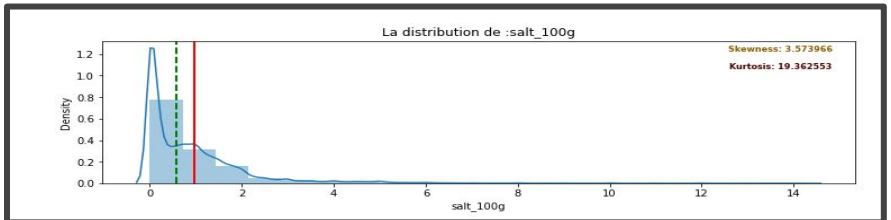
## Analyse univariée des variables quantitatives : Distribution des variables

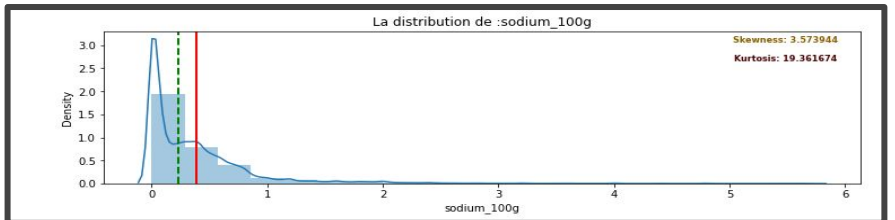


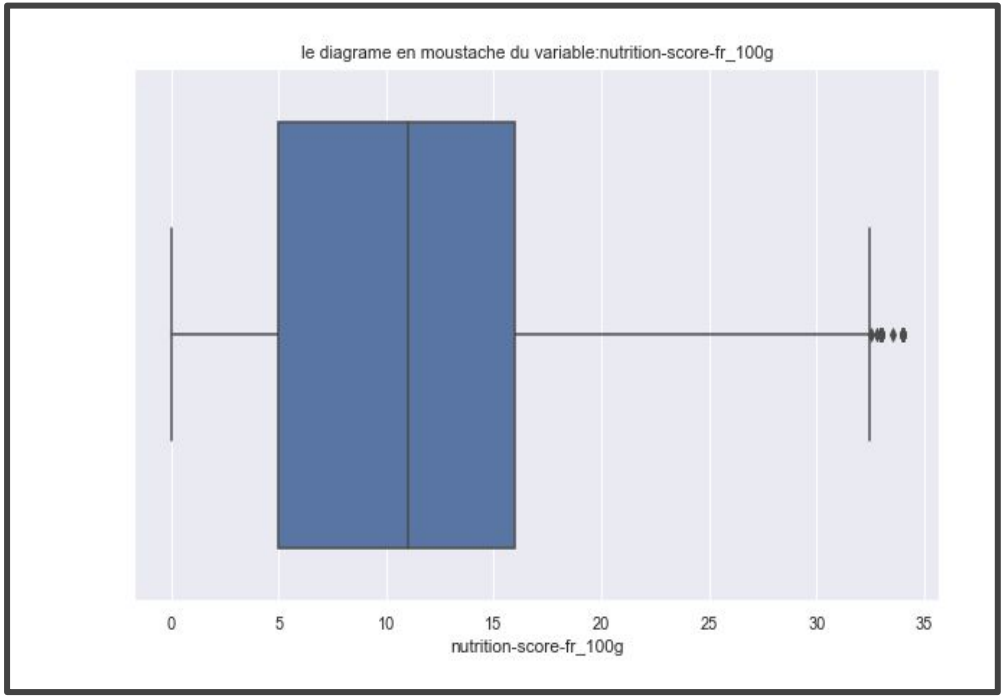



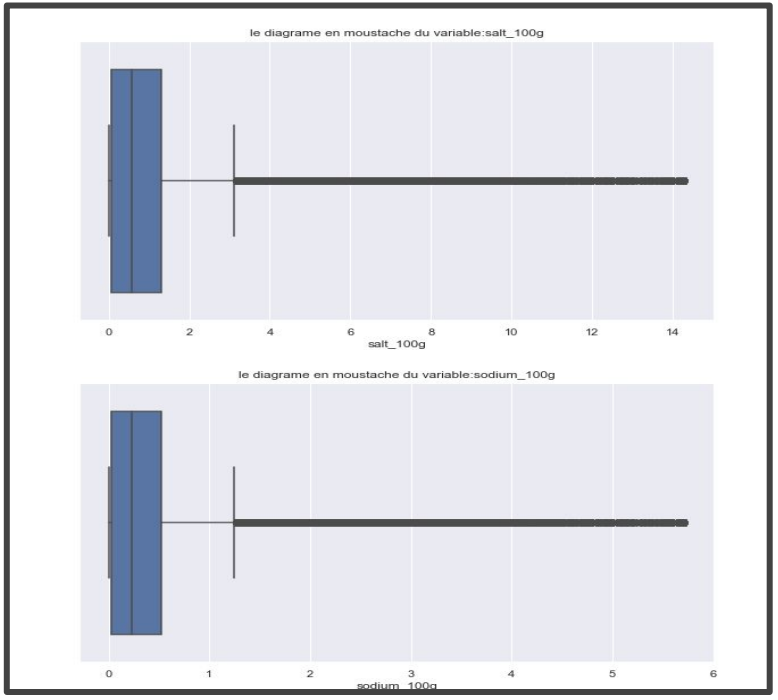



# 3. Analyse des données :

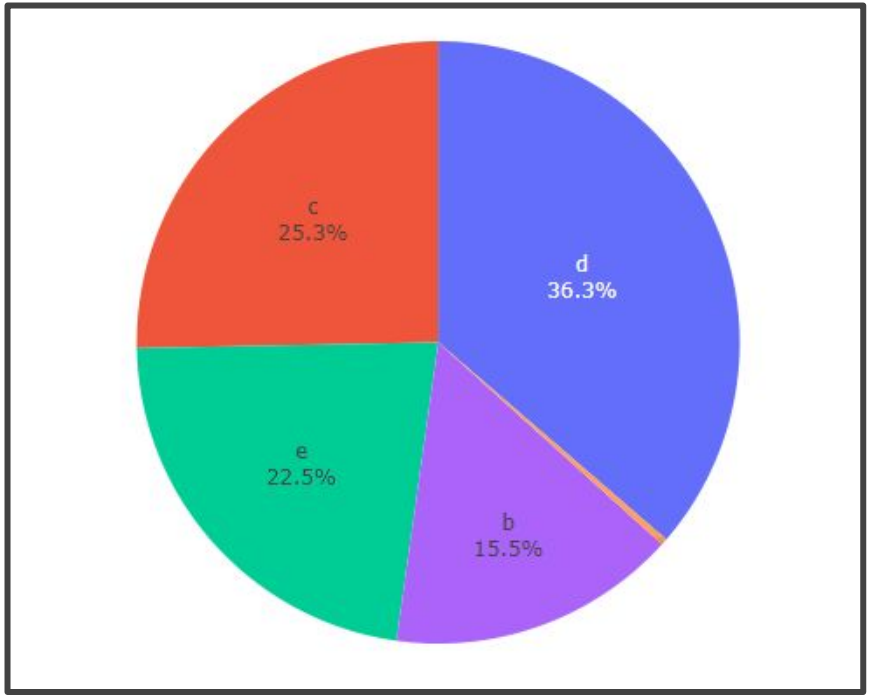
## Analyse univariée des variables quantitatives : Boxplot des variables

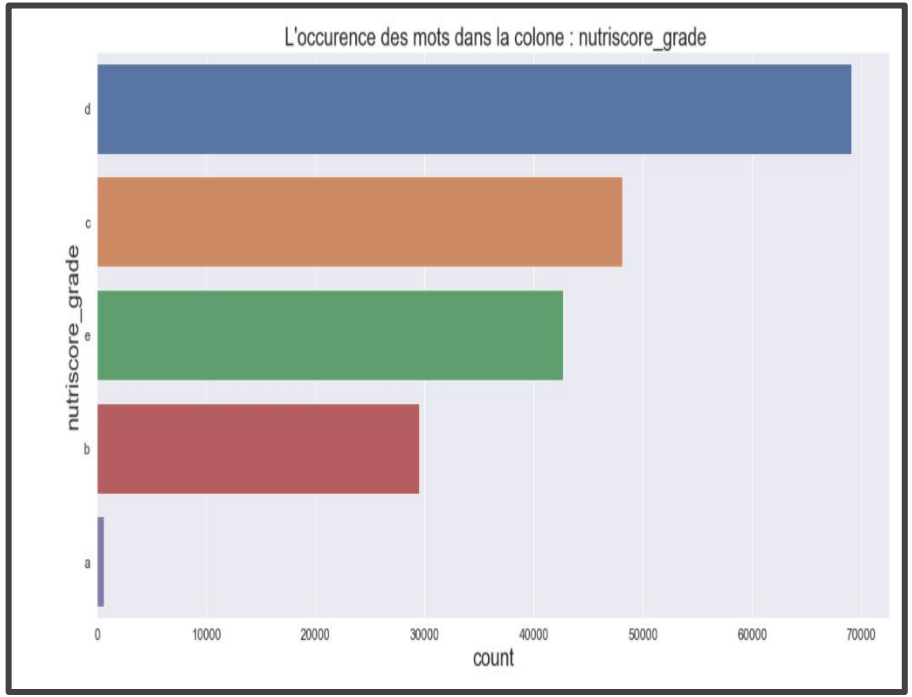


## Répartition des grades de nutri score

La répartition des grades du nutri score nous montre que le



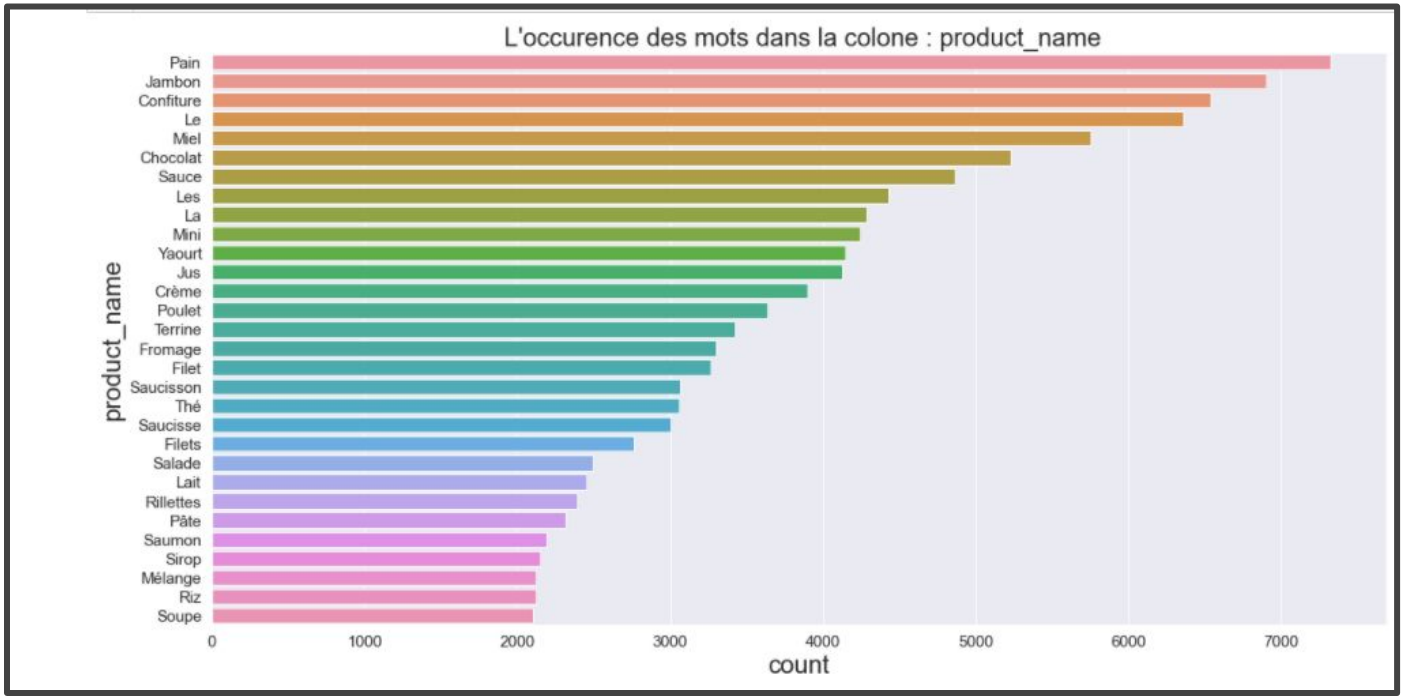



# 3. Analyse des données :

## Le nombre de produits par catégorie:

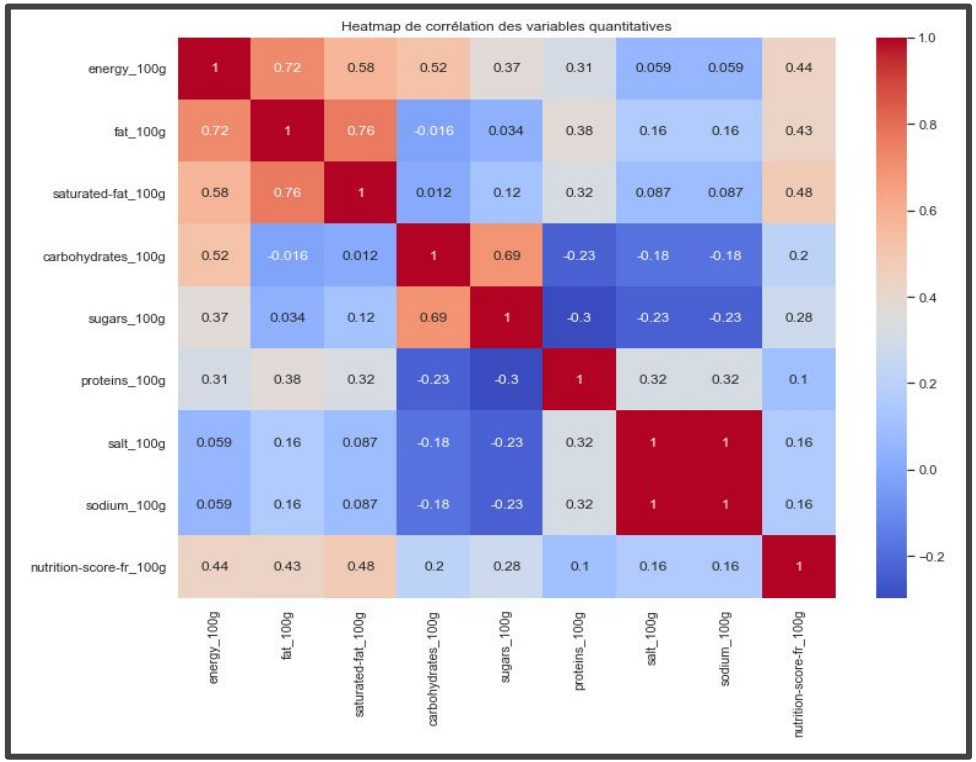
# 3. Analyse des données :

## Occurrence des mots dans la colonne product_name :



# 3. Analyse des données :

## Analyse bivariée




* energy_100g a une corrélation forte avec : fat_100g , saturated_fat_100g , carbohydrates_100g et moins avec nutriscore_100g

* fat_100g a une corrélation forte avec energy_100g , saturated_fat_100g et mois avec nutriscore_100g

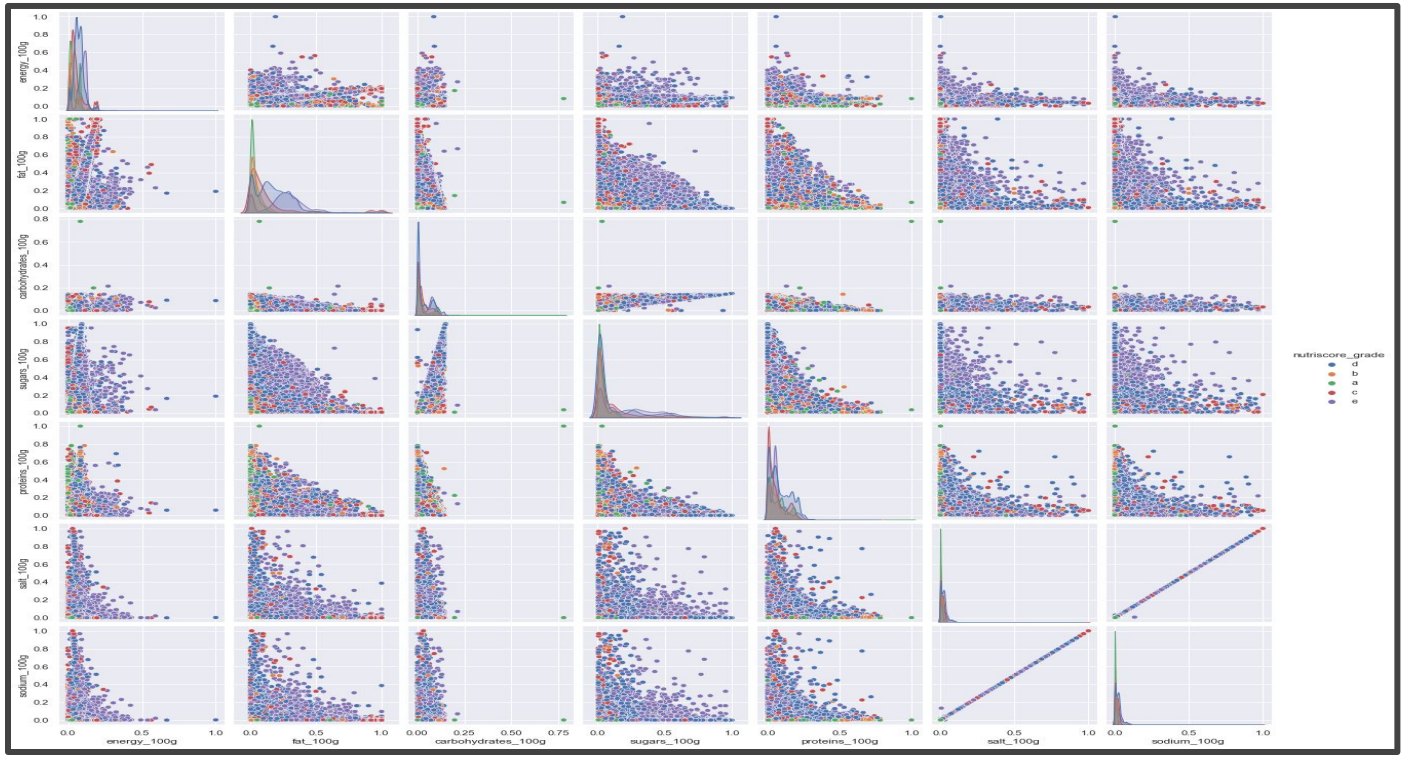
* saturated_fat_100g a une corrélation energy_100g , fat_100g

* carbohydrates_100g a une correlation forte avec sugars_100g , energy_100g

* nutriscore_100g a une corrélation moyenne avec energy_100g , fat_100g n saturatedfat_100g .

# 3. Analyse des données :

## Analyse bivariée



# 3. Analyse des données :

## Tests statistiques : Test d'indépendance (Khi2 )

Test d'indépendance entre la variable nutrition_score et les autres variables :

- additives_n;
- carbohydrates_100g;
- energy_kcal_100g;
- energy_100g ;
- fat_100g ;
- proteins_100g ;
- salt_100g ;
- saturated_fat_100g ;
- sodium_100g ;
- sugars_100g .

```
--------------------
test de khi2 entre 'nutrition-score'  et  additives_n
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  carbohydrates_100g
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  energy-kcal_100g
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  energy_100g
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  fat_100g
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  proteins_100g
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  salt_100g
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  saturated-fat_100g
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  sodium_100g
Variables non indépendants (H0 Rejetée)
--------------------
test de khi2 entre 'nutrition-score'  et  sugars_100g
Variables non indépendants (H0 Rejetée)
```

# 3. Analyse des données :

## Tests statistiques : Test de normalité (Kolmogorov Smirnov)

```
------------------------------------------------------------
Test de normalité de la colone:  energy_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
------------------------------------------------------------
Test de normalité de la colone:  fat_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
------------------------------------------------------------
Test de normalité de la colone:  saturated-fat_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
------------------------------------------------------------
Test de normalité de la colone:  carbohydrates_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
------------------------------------------------------------
Test de normalité de la colone:  sugars_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
------------------------------------------------------------
Test de normalité de la colone:  proteins_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
------------------------------------------------------------
Test de normalité de la colone:  salt_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
------------------------------------------------------------
Test de normalité de la colone:  sodium_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
------------------------------------------------------------
Test de normalité de la colone:  nutrition-score-fr_100g
l'hypothèse nulle est rejetée, La distribution n'est pas normale
```

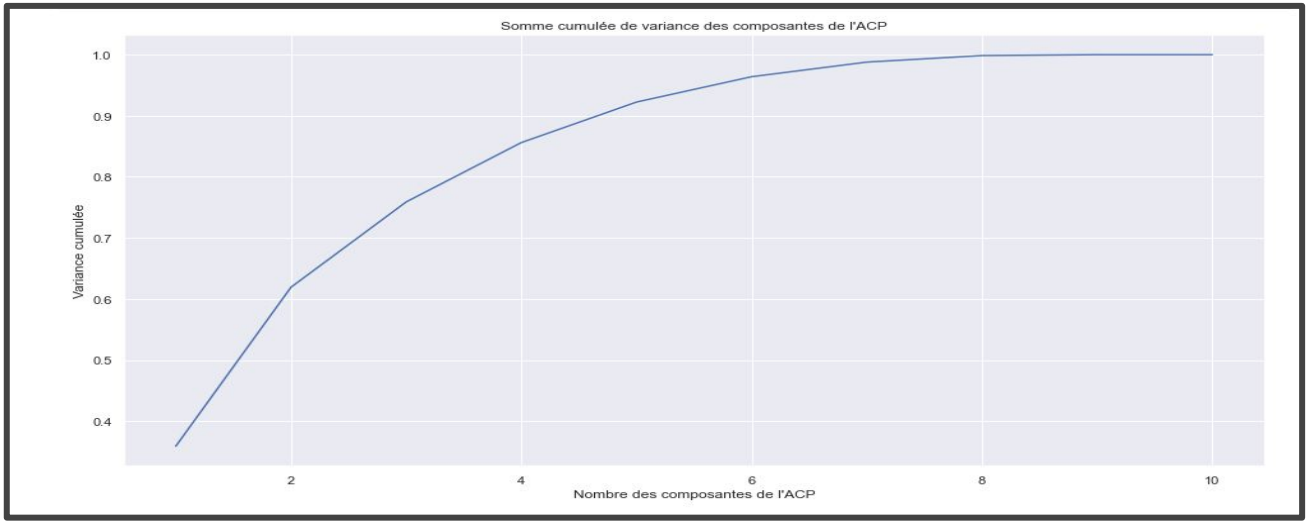
# 3. Analyse des données :

## Analyse multivariée et réduction de dimension par l'analyse de composantes principales(ACP):

1. Choix des variables quantitatives;
2. Réduction et centrage des données : StandardScaler();
3. Entraînement du modèle ;
4. Visualisation des variances cumulées
5. Visualisations du diagramme d'éboulis ;
6. Visualisation du cercle de corrélations .

# 3. Analyse des données :

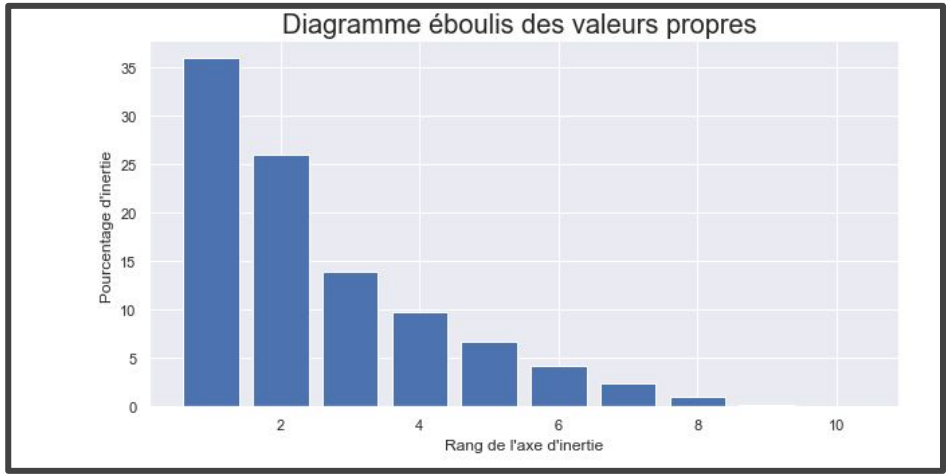
## Analyse en composantes principales: Somme cumulée de variance des composantes de l'ACP




A partir de 7 features on a une variance cumulée de plus de 95 %. On pourrait donc réduire notre jeu de données à 7 dimensions si on souhaitait gagner en temps de calcul / volume de données.

# 3. Analyse des données :

## Analyse en composantes principales : Diagramme éboulis des valeurs propres



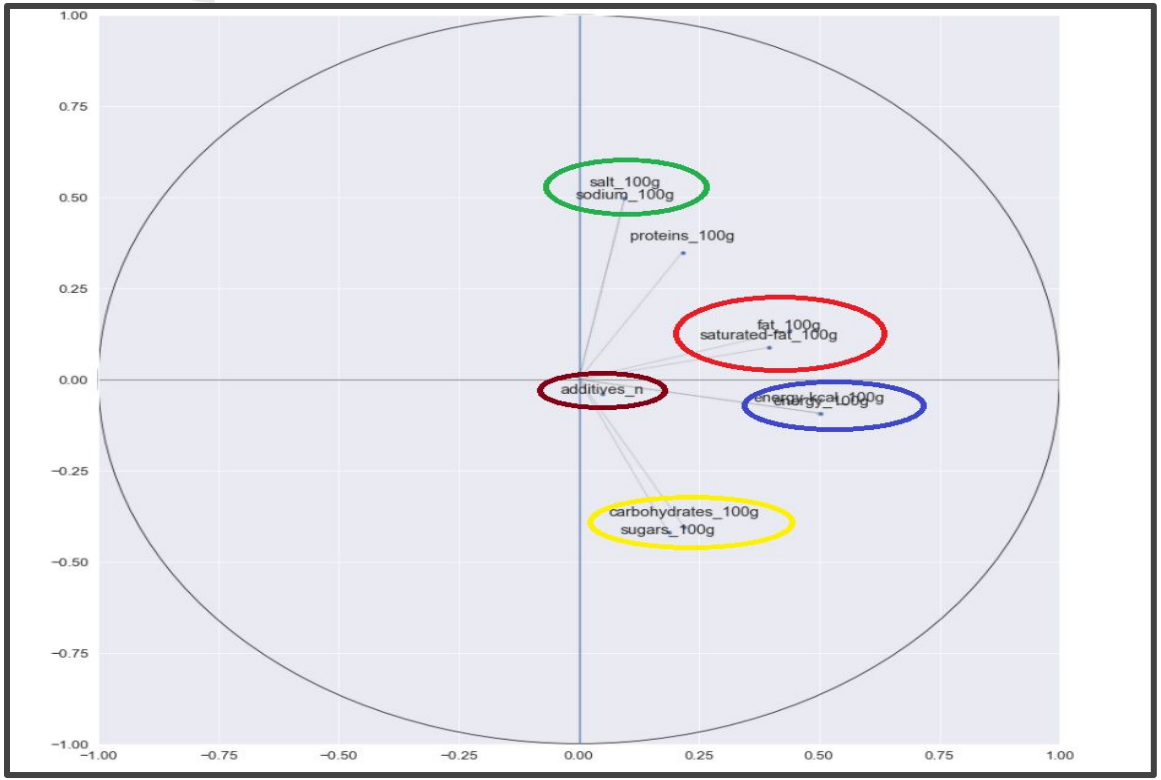

En ACP, on projette les données sur les axes principaux d'inertie, et que ceux-ci sont ordonnés selon l'inertie du nuage projeté : de la plus grande à la plus petite.

Quand on additionne les inerties associées à tous les axes, on obtient l'inertie totale du nuage des individus.

Le diagramme **éboulis des valeurs propres** décrit le pourcentage d'inertie totale associé à chaque axe.

# 3. Analyse des données :

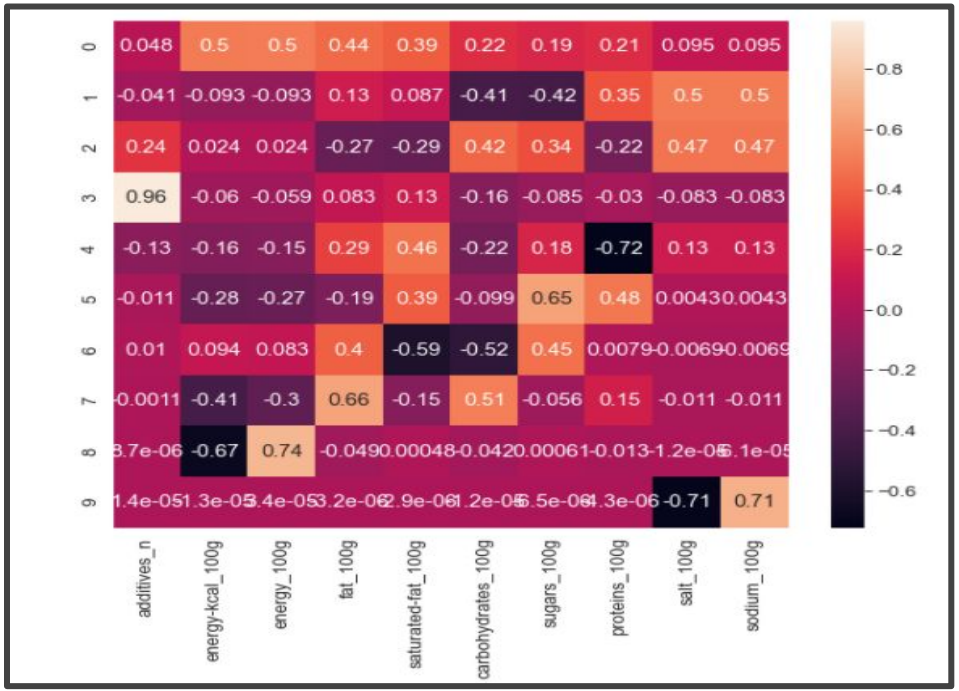
## Analyse en composantes principales : Cercle de corrélations




- **fat_100g** et **saturated_fat_100g** expliquent bien la variance sur la composante 1 mais pas sur la composante 2.
- **additives_n** n'explique pas la variable sur la composante 2 et quasiment pas sur la composante 1
- **sodium_100g** et **salt_100g** sont quasi confondus : les variables sont corrélées positivement.
- **carbohydrates_100g** et **sugars_100g** sont très proches et donc également corrélées positivement
- **carbohydrates_100g**, **sugars_100g**, **sodium_100g** et **salt_100g** expliquent bien la variance sur la composante 2, moins sur la composante 1.

# 3. Analyse des données :

## Analyse en composantes principales : Matrice de corrélation




La matrice de corrélation de l'ACP nous donne une information sur la structure de chaque axe d'inertie
Par exemple la première composante est composée principalement par :

- energy_100g;
- energy_kcal_100g;
- fat_100g;
- saturated_fat_100g

# 4. Idée d’application :

## Régression linéaire multiple :

L'idée d'application vient à partir des analyses réalisées jusqu'à présent et qui montre :
* La corrélation entre les variables ;
* La non indépendance des variables ;
* Le sens métier : le nutri score est forcément calculée à partir des composantes des produits alimentaires.

# 4. Idée d’application :

## Régression linéaire multiple :

### Les étapes de la création du modèle

1. Sélection des variables d’entrée et de la variable de sortie (le nutri score);
2. Standardisation des données;
3. Séparation des données en donnée d'entraînement et données de test ;
4. Calcul du coefficient de détermination;
5. Utilisation du cross validation(Shuffle split)
6. Utilisation de la régularisation L1 et L2 pour augmenter la performance du modèle.
7. Utilisation du Gridsearch avec changement des hyper paramètre

# 4. Idée d'application :

## Régression linéaire multiple :

### 1. Sélection des variables d'entrée et de la variable de sortie (le nutri score);

| | energy_100g | fat_100g | saturated-fat_100g | carbohydrates_100g | sugars_100g | proteins_100g | salt_100g | sodium_100g | nutrition-score-fr_100g |
|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.177916 | -0.341970 | -0.442890 | 0.128919 | 0.542675 | -0.428594 | 2.622311 | 2.622248 | 0.997196 |
| 1 | -0.783993 | -0.657137 | -0.617964 | -0.344366 | -0.079217 | -0.740106 | 1.699905 | 1.699862 | 0.753836 |
| 2 | -1.390070 | -0.972304 | -0.793038 | -0.817651 | -0.701108 | -1.051617 | 0.777500 | 0.777476 | 0.510475 |
| 3 | -1.157074 | -0.741694 | -0.633880 | -0.614256 | -0.551393 | -0.822752 | 0.136140 | 0.136129 | 0.267115 |
| 4 | -1.153357 | -0.787816 | -0.665711 | -0.571230 | -0.407437 | -0.868525 | -0.071041 | -0.071047 | 0.023754 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 633139 | 0.666875 | -0.395779 | -0.655101 | 1.550079 | 1.348830 | -0.246349 | -0.682975 | -0.682968 | -0.505913 |
| 633140 | 0.708329 | -0.126734 | -0.586133 | 1.263239 | 2.385315 | -0.276017 | -0.678171 | -0.678164 | 0.030912 |
| 633141 | 0.475332 | 0.580470 | 0.161909 | 0.203237 | -0.125283 | -0.136155 | 0.100109 | 0.100099 | 0.567737 |
| 633142 | 0.475332 | 0.580470 | 0.161909 | 0.203237 | -0.125283 | -0.136155 | 0.100109 | 0.100099 | 0.567737 |
| 633143 | 0.475332 | 0.580470 | 0.161909 | 0.203237 | -0.125283 | -0.136155 | 0.100109 | 0.100099 | 0.567737 |

# 4. Idée d'application :

Régression linéaire multiple :

## 2. Standardisation des variables :

```
model = LinearRegression()
X = numeric_data.drop(columns='nutrition-score-fr_100g', axis=1)
y = numeric_data['nutrition-score-fr_100g']
#print(X.shape, y.shape)

scaler = StandardScaler()
X = scaler.fit_transform(X)
```

## 3. Séparation des données en entraînement / test :

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model.fit(X_train, y_train)
print("le coefficient de détrmination est :" ,r2_score(y_test, model.predict(X_test)))
print("L'erreur absolue moyenne est : ", metrics.mean_absolute_error(y_test, model.predict(X_test)))
print("Le carré moyen des erreurs",metrics.mean_squared_error(y_test, model.predict(X_test)))
print("L'erreur quadratique moyenne est : ", np.sqrt(metrics.mean_squared_error(y_test, model.predict(X_test))))
```

# 4. Idée d'application :

Régression linéaire multiple :

## 4 . Coefficient de détermination :

```
le coefficient de détrmination est : 0.49399761198537784
L'erreur absolue moyenne est :  0.569652476117747
Le carré moyen des erreurs 0.506383796422167
L'erreur quadratique moyenne est :  0.7116064898679375
```

## 5. Utilisation du ShuffleSplit :

```
cv= ShuffleSplit(30, test_size=0.2)
cross_val_score(model, X_train, y_train , cv=cv).mean()
```

# 4. Idée d'application :

Régression linéaire multiple :

## 6 . Régularisation des données :Lasso

```
hyper_params= [0.005, 0.02, 0.03, 0.05, 0.06 ,1,2,3 , 0.001, 0.00005 ]
for params in hyper_params:
    print('-'*20)
    print("l'hyper paramètre", params)
    model= Lasso(alpha=params)
    model.fit(X_train, y_train)
    print("le coefficient de détrmination est :" ,r2_score(y_test, model.predict(X_test)))

    print("MSE" , metrics.mean_squared_error(y_test, model.predict(X_test)))
```

# 4. Idée d'application :

Régression linéaire multiple :

## 6 . Régularisation des données :Ridge

```
hyper_params= [1e-8, 1e-7, 0.000001, 0.001, 0.005, 0.01, 0.05, 0.5, 0.6 , 500 , 600, 700]
for params in hyper_params:
    print('-'*20)
    print("l'hyper paramètre", params)
    model= Ridge(alpha= params)
    model.fit(X_train, y_train)
    print("le coefficient de détrmination est :" ,r2_score(y_test, model.predict(X_test)))
```

# 4. Idée d'application :

## Régression linéaire multiple :

### 7 . Grid Search

```
def find_best_model_using_gridsearchcv(X,y):
    algos = {
        'linear_regression' : {
            'model': LinearRegression(),
            'params': {
                'normalize': [True, False]
            }
        },
        'lasso': {
            'model': Lasso(),
            'params': {
                'alpha': [0.005, 0.02, 0.03, 0.05, 0.06 ,1,2,3],
                'selection': ['random', 'cyclic']
            }
        },
        'Ridge': {
            'model': Ridge(),
            'params': {
                'alpha': [550, 580, 600, 620, 650],
            }
        },
        'Elasticnet': {
            'model': ElasticNet(),
            'params': {
                'alpha': [1e-8, 1e-7, 0.000001, 0.001, 0.005, 0.01, 0.05, 0.5, 0.6],
            }
        },
        'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {
                'criterion' : ['mse','friedman_mse'],
                'splitter': ['best','random']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=20, test_size=0.2, random_state=0)
    for algo_name, config in algos.items():
        gs =  GridSearchCV(config['model'], config['params'], cv=cv, return_train_score=False)
        gs.fit(X,y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })

    return pd.DataFrame(scores,columns=['model','best_score','best_params'])
```

| | model | best_score | best_params |
|---|---|---|---|
| 0 | linear_regression | 0.504436 | {'normalize': True} |
| 1 | lasso | 0.504206 | {'alpha': 0.005, 'selection': 'random'} |
| 2 | Ridge | 0.504421 | {'alpha': 550} |
| 3 | Elasticnet | 0.504435 | {'alpha': 1e-07} |
| 4 | decision_tree | 0.403330 | {'criterion': 'friedman_mse', 'splitter': 'best'} |

# Merci pour votre attention