Concevez une application au service de la santé public

Santé publique France

OPENCLASSROOMS

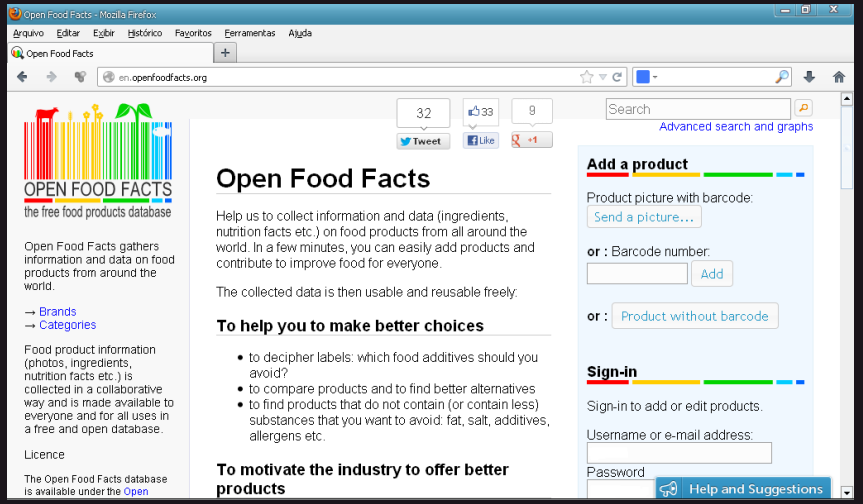
Victoire Mohebi
Décembre 2021

SOMMAIRE

1. Rappel de Problématique

- **Un appel à projets de « l’agence santé publique France »**
- **Trouver des idées innovantes d’applications en lien avec l'alimentation**

2.Présentation des données



Open Food Facts

- **Une base de données libre sur les produits alimentaires**
- **Un projet collaboratif citoyen à but non lucratif**

La dimension de jeu de donnés

2.011.182

Nombre de lignes

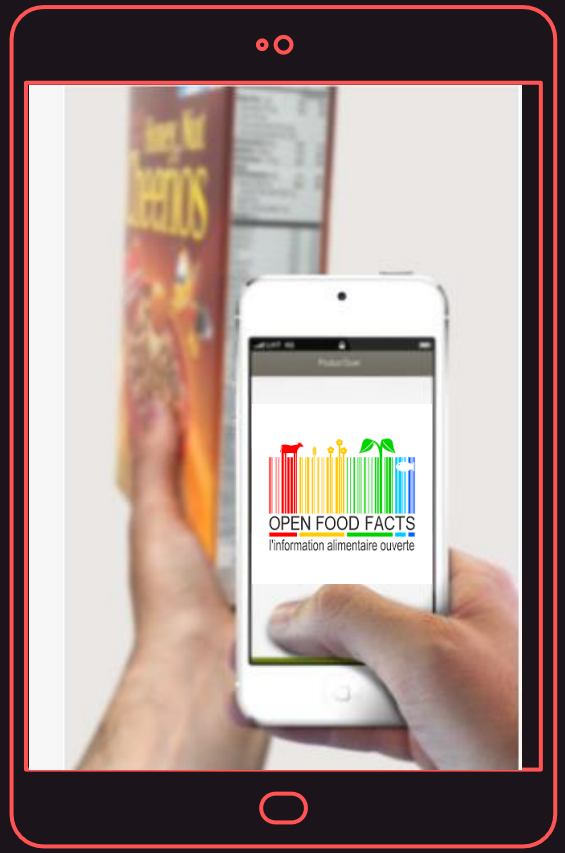
187

Nombre des colonnes

Les sections des variables

1. Les informations générales sur les produits: nom, code_bara, etc
2. Des tags : catégorie, pays d'origine, etc
3. Les ingrédients et les additifs composant des produits
4. Des informations nutritionnelles en 100gr pour chaque produit

3. Idée d’application



Introduction au régime cétogénqiue

- **Le régime cétogène est un régime riche en lipides.**
- **Ce régime vise à réduire considérablement la consommation de glucides au profit des lipides pour provoquer un état de cétose**

Application « keto_score »

Le « keto_score » est une application qui pourrait aider les consommateurs qui souhaite suivre un régime cétogène

Conception de l’application keto_score

- **Code bar**
- **Variables sélectionnées**

- **Requête dans la base de donnée pour calculer le « keto_score »**

- **Indication de qualité du produit pour le régime Cétogène**

4.Nettoyage de données

Nettoyage effectué

1) La mise en forme de nom des colonnes
2) Typage de données
3) Traitement des doublons sur le variable code
4) Traitement des valeurs aberrantes
5) Traitement des valeurs manquantes

1) La mise en forme de nom des colonnes

```python
# repmlacer les '-' par '_'
df.columns = df.columns.str.replace("-", "_")

# supprimer les tirets au début de nom des colonnes
def clean_col_name(dataframe):

    columns = dataframe.columns
    columns_treated = []
    for column in columns:
        if column[0] == '_':
            column = column[1:]
        columns_treated.append(column)
    dataframe.columns = columns_treated
    return dataframe

df = clean_col_name(df)
```

2) Typage de donnée

```python
# Convertir à datatime
df['last_modified_datetime'] = pd.to_datetime(df['last_modified_datetime'],utc=True)
df['created_datetime'] = pd.to_datetime(df['created_datetime'],utc=True)
```

```python
# Vérifier la conversion à datetime
df.dtypes.value_counts()
```

```
float64                123
object                  58
datetime64[ns, UTC]      2
dtype: int64
```

3. Traitement des doublons sur le variable code

- **Taux d'occurrence de doublon sur le variable code : 0.012%**
- **Nombre des lignes en doublon supprimés : 243**

```
# Pourcentage de duplicats sur le variables "code"
print("Pourcentage de duplicats sur le variable 'code' : ", df.duplicated(subset =['code']).astype(int).mean())


# Ordonner le jeu de donne par la date de drnière modification
df = df.sort_values(by=["last_modified_datetime"], ascending= False)

df1 = df.drop_duplicates(subset = ['code'], keep = 'last')

# Vérifier que les linges en doublons ont été supprimés
print("Pourcentage de duplicats sur le variable 'code' :",df1[df1['code'].duplicated() == True].shape[0])

# Afficher le nombre de code en doublons supprimés
print("Nombre de ligne en doublon supprimées :", df.shape[0] - df1.shape[0], "\n")
```

```
Pourcentage de duplicats sur le variable 'code' :  0.00012082447038607148
Pourcentage de duplicats sur le variable 'code' : 0
Nombre de ligne en doublon supprimées : 243
```

4) Traitement des valeurs aberrantes

- Les valeurs > 100 et < 0 pour les nutriments en 100g
- Les valeurs qui ne s'accordent pas à cette formule :

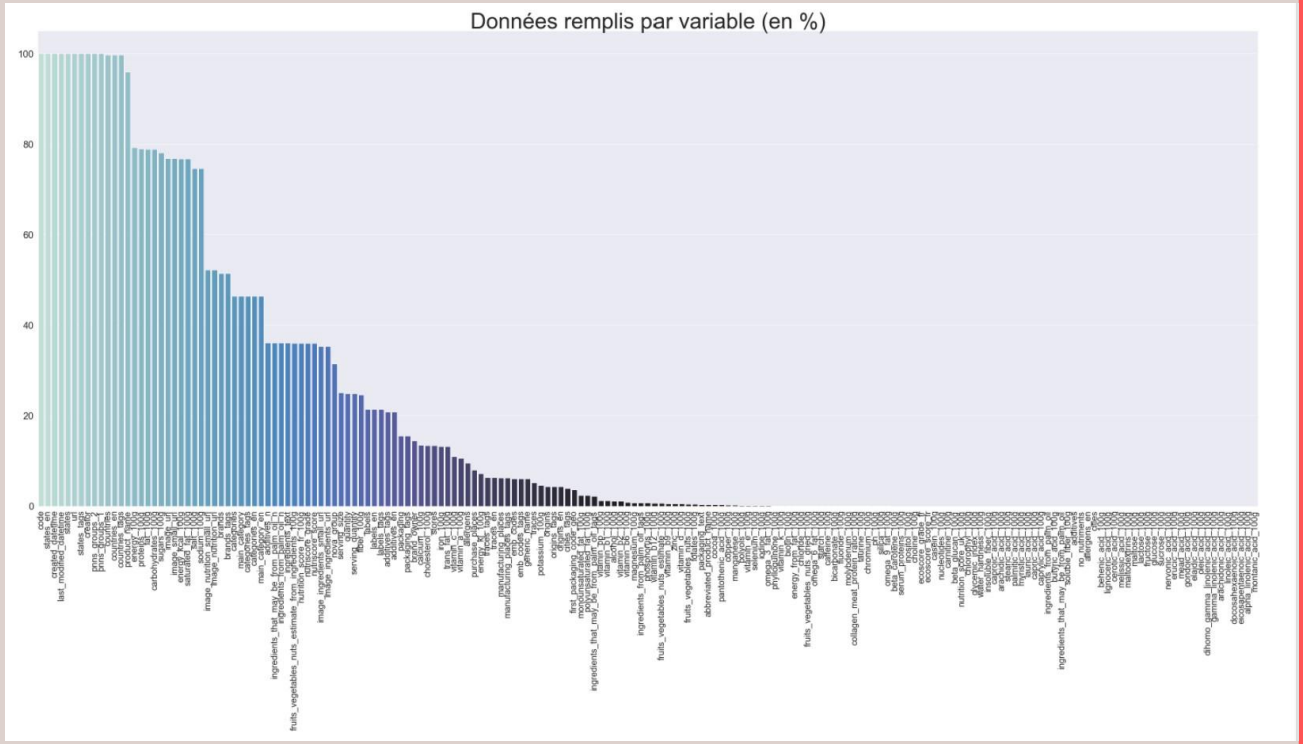
$$energy\_100g = (4 \times proteins) + (4 \times carbohydrate) + (9 \times fat)$$

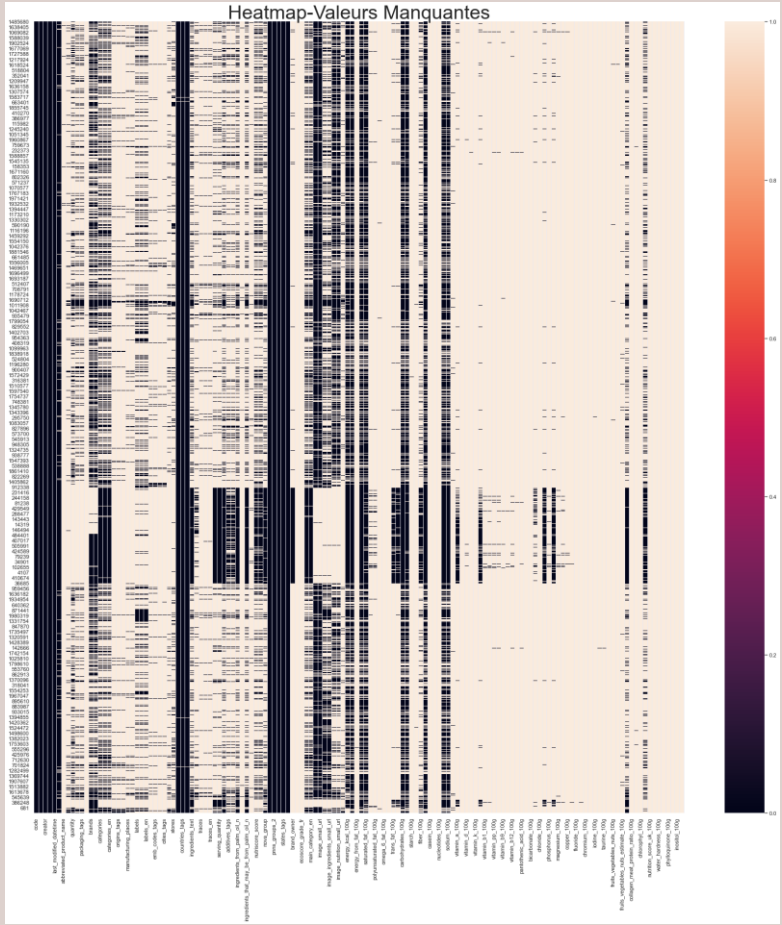
-Les valeur aberrante pour « energy_kcal_100g » avec IQRscore Méthode
-Les valeurs aberrantes pour « serving_quanity » avec IQRscore Méthode
-175069 lignes supprimées après le traitement des valeurs aberrantes

5) Traitement des valeurs manquantes

I- Environ 80.7% de jeu de donnés n'est pas renseigné

II- Supprimer 41 variables composée uniquement de Null





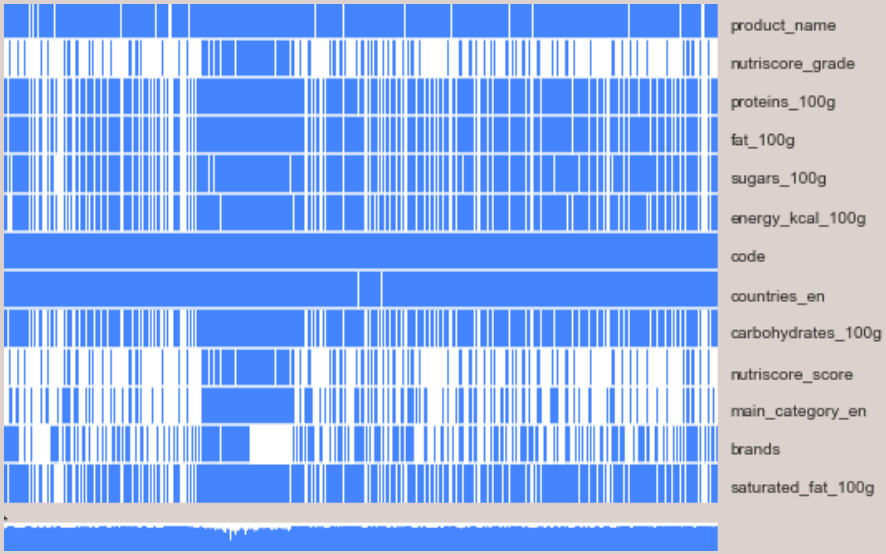
III- Supprimer 129 variables avec plus de 80% des valeurs manquantes

IV- Supprimer 360 lignes avec plus de 80% des valeurs manquantes

Filtrer le jeu de donnée en fonction d'idée de l'application

- Choisir des variables potentiellement pertinentes pour l’applicataion keto_score
- Faire le tri et choisir définitement des variables pour l’application keto_score

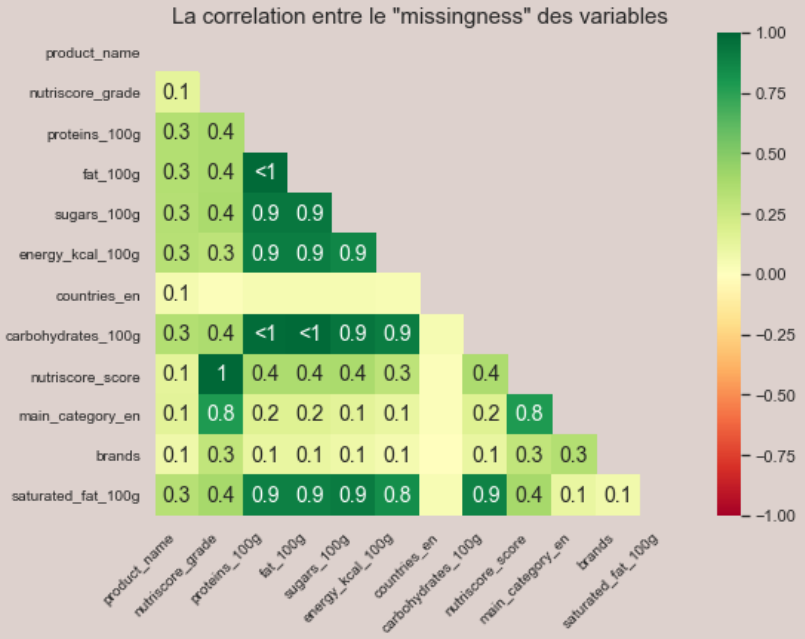
Identifier le type de « missingness » des données



- « product_name » : MCAR (Missing Completely At Random)
- Pour les nutriments_100gr : MAR (missing at random)

Heatamp de missingness

- **Une forte relation de nullité entre toutes les variable de nutriment_100g**
- **Une forte relation de nullité entre « nutri_grade » et « category »**



Imputation des valeurs manquantes

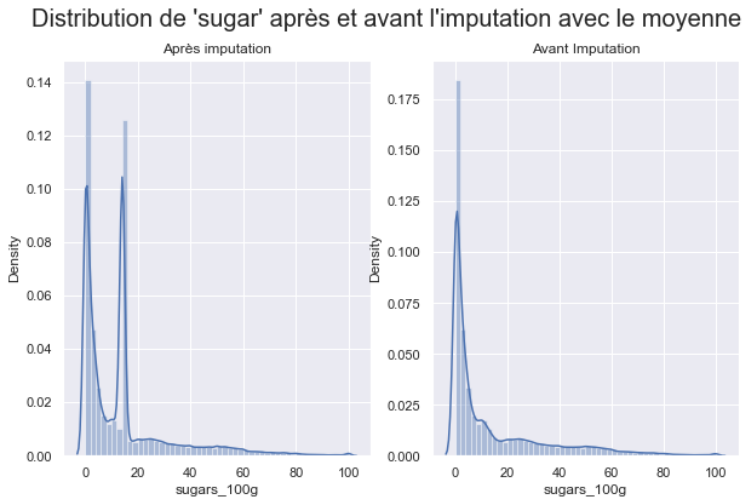
Les stratégies

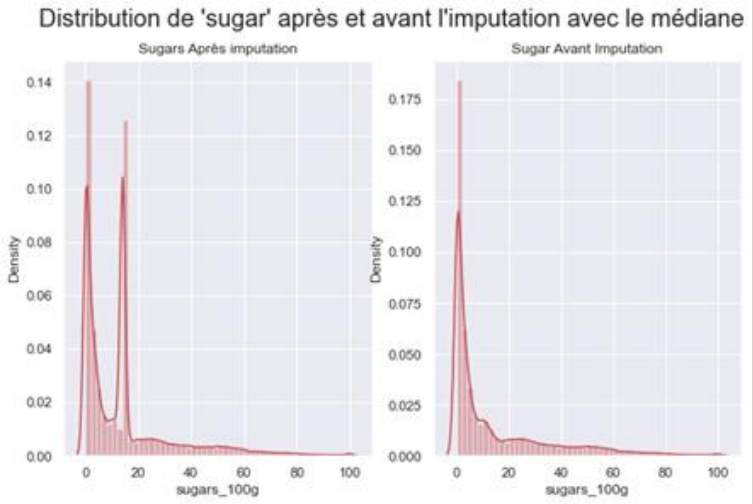
1. Par le moyenne
2. Par le médiane
3. Par le « Multivariate feature imputation » (« K-Nearest Neighbor Imputation »)

Evaluation de la stratégie

1. Le temps de l'exécution des code
2. Comparer la distribution de donnée avant et après l'imputation

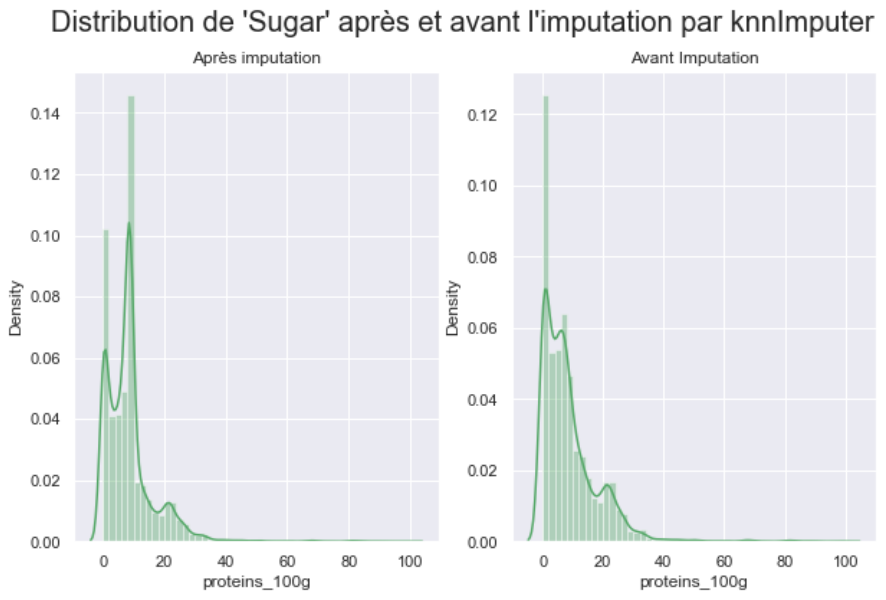
Comparer la distribution de donnée





La distribution de donnée a été changée (ex. variable sucre)

Imputer les valeurs manquantes par k-Nearest Neighbors algorithme



« KnnImputer » : la méthode pertinente

Inconvénient majeur :
Le temps d'exécution très long
4.81 s sur une échantillon de 0.05% de donnée)

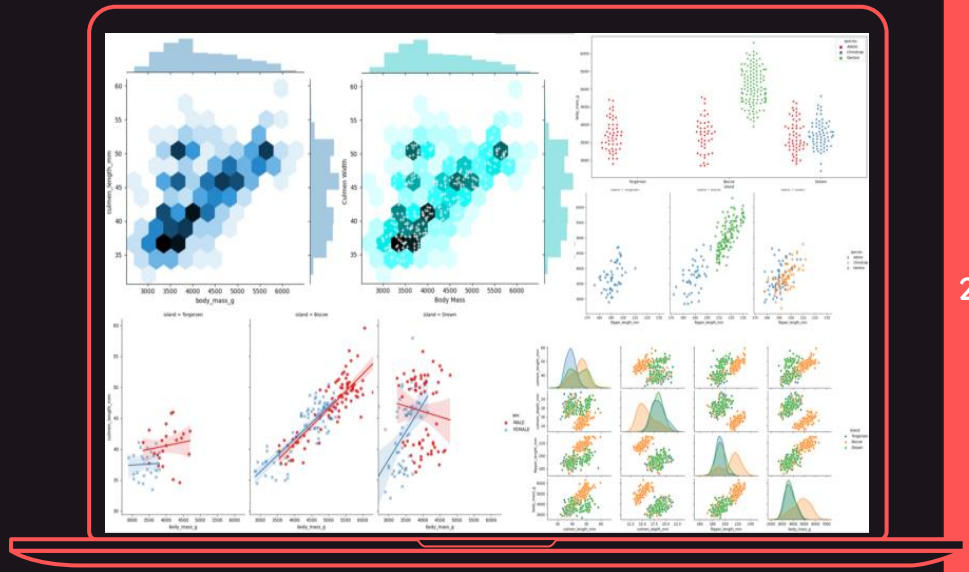
Alternative :
Sélectionner des lignes qui n'ont pas de valeurs manquantes

Filtrer le jeu de donnée pour l’applicatio

Ligne : 385.652
Colonne : 13
Taux de valeur manquantes : 0%

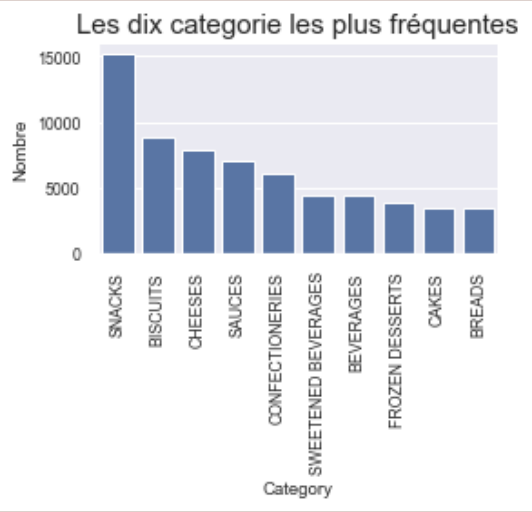
	Nom	Dimensions	Proportion de NaN (en %)
0	données initiales	(2011182, 187)	80.02982
1	données nettoyées	(385652, 13)	0.00000

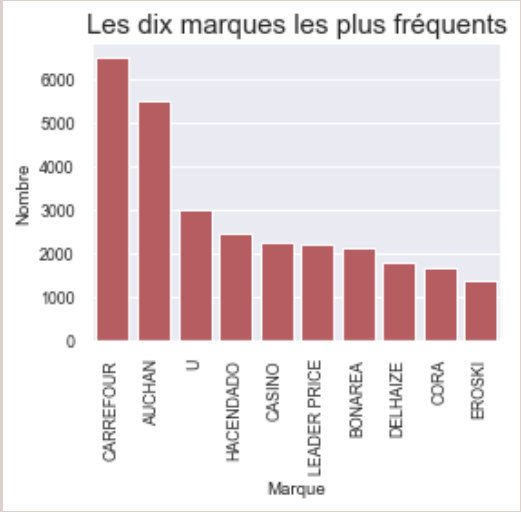
5. Analyse exploratoire

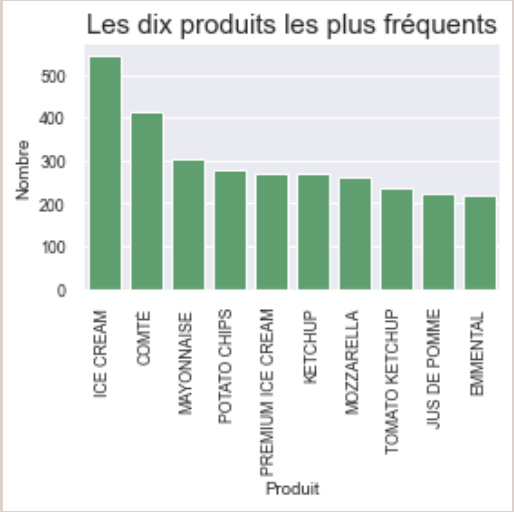


5.1 Analyse univariée

variables catégorielles

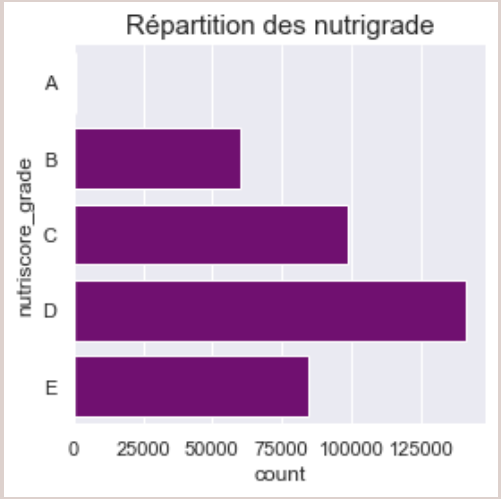


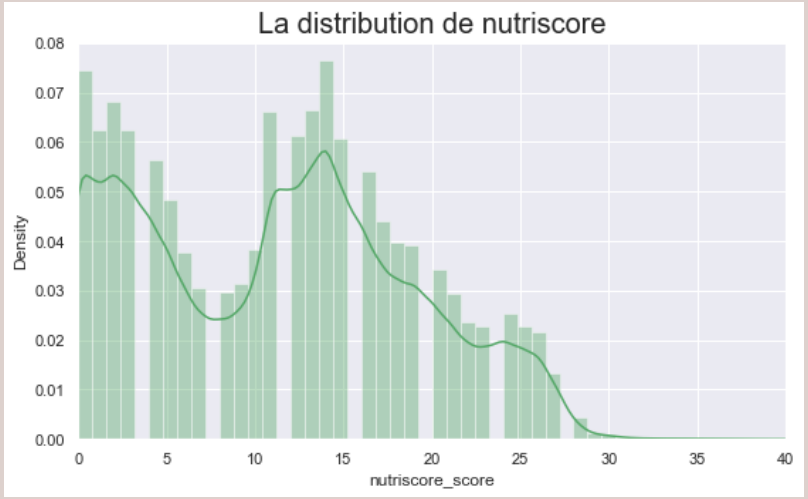




5.1 Analyse univariée

« nutriscore »



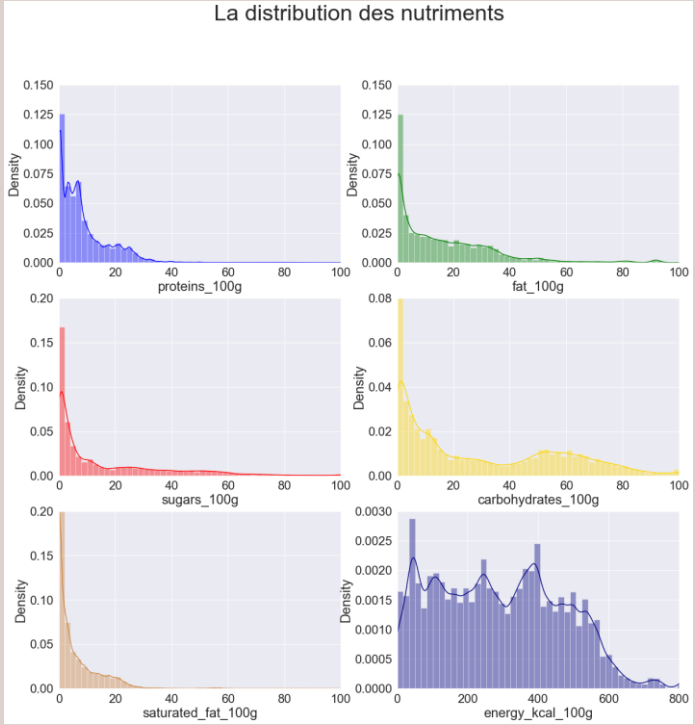


- **Pas beaucoup de produits avec le nutri_grade de "A"**
- **Les produits avec le nutri_grade "D" plus nombreux**

- **Le coefficient d'asymétrie : 0.19**
- **Le moyenne de nutri_score : 11.4**

5.1. Analyse univariée

variables numériques



Le coefficient d'asymétrie

proteins : 2.091

fat : 1.63

sugars : 1.57

carbohydrates : 0.6

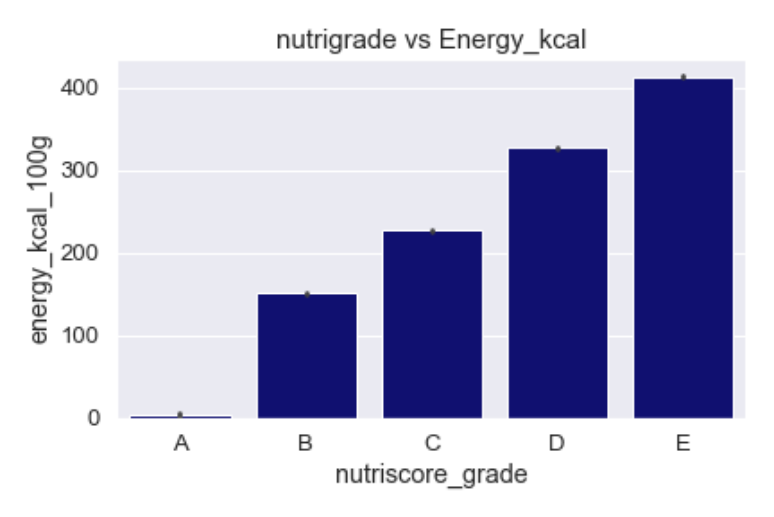
saturated_fat : 2.55

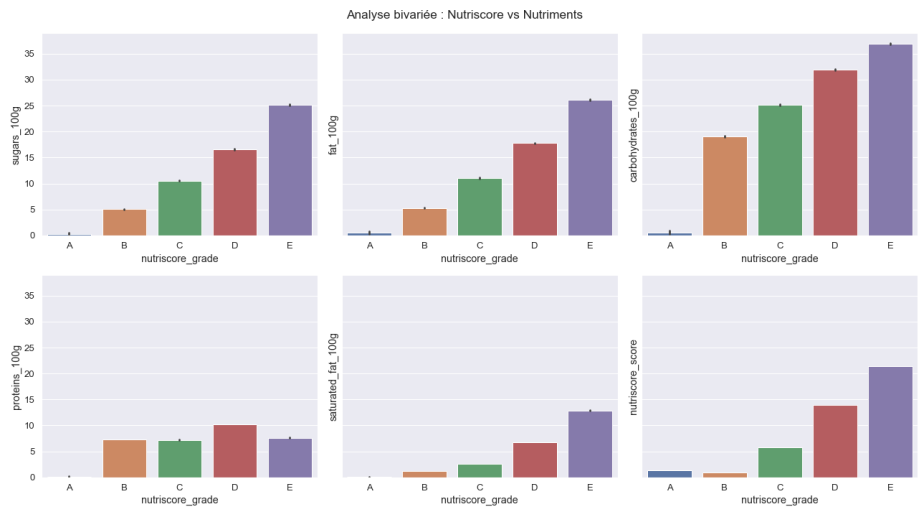
energy_kcal : 0.30

L'écart_type élevé : «energy_ kcal» «carbohydrate»

5.2 Analyse bivarié

Variables numérique vs. variables catégorielle



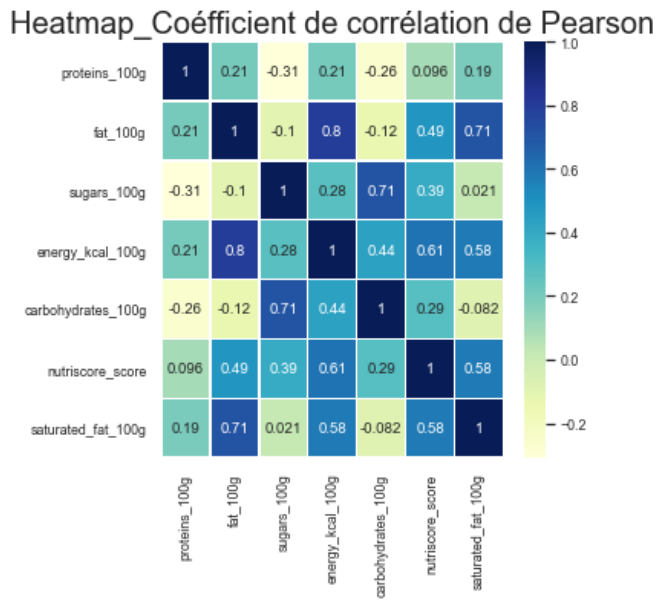


- Les produits les moins caloriques sont de group « A »
- Les produits les plus caloriques sont de group « E »

- Les produits les plus riches en carbohydrate sont de group « E »
- Les produits de group « D » sont plus riche en protéines que ceux d'autre group

5.2 Analyse bivariée

Corrélation entre variables

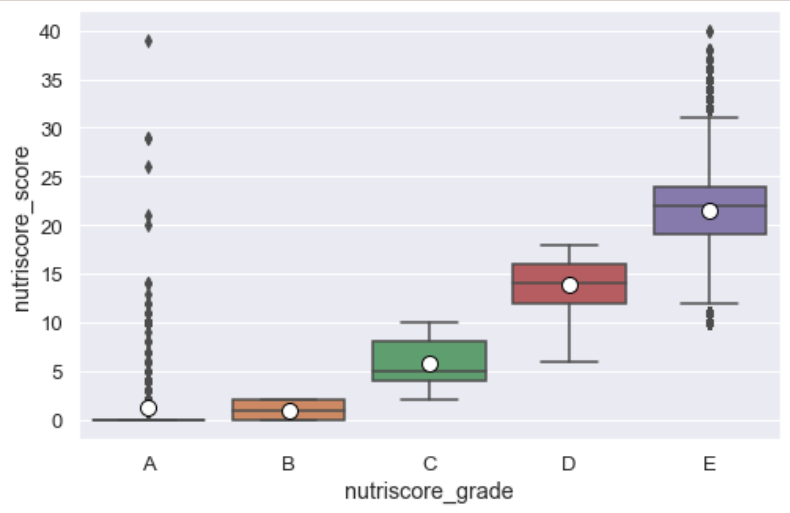


- Une relation linéaire forte entre les variables « fat/saturated fat » , « carbohydrate/sugars » et « fat"/"energy_kcal »

- « proteins/sugars » et « protéine/carbohydrate » sont négativement corrélée

5.3 Analyse explicative

Variable numérique vs variable catégorielle



- **Selon le "nutriscore_grade", la variable "nutriscore_score" a des moyennes différentes**

- **Les deux variables ne semblent pas être indépendantes.**

Test statistique analyse des variance (ANOVA)

L’hypothèse null :

Les moyennes de deux variables "nutri_grade" et "nutri_score" ne sont pas différentes

Les conditions de test Anova

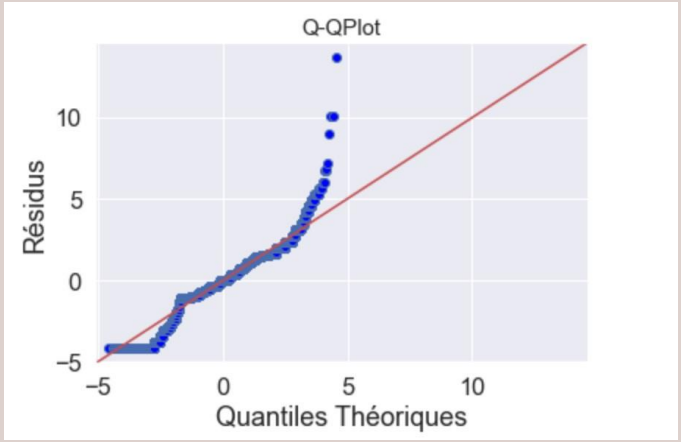
I. **Independence des échantillons**
II. **La distribution normale des résidus de modèle**
III. **La Homoscédasticité (La variance des erreurs de modèle est la même)**

I. Independance des échantillon

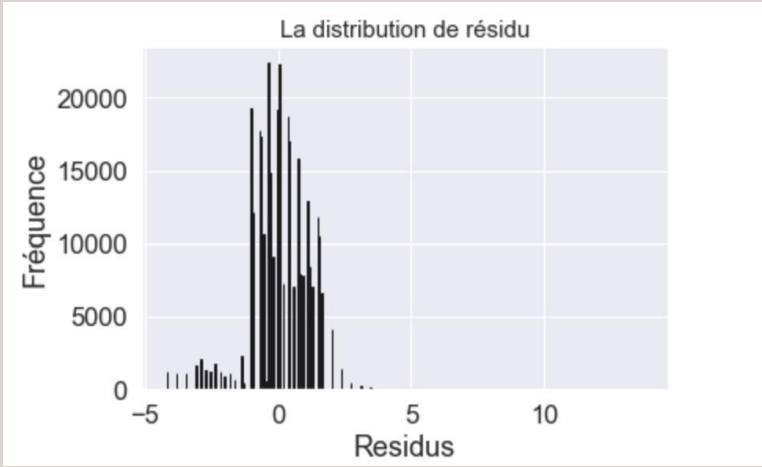
- **Les données sur chaque produit sont collectées indépendamment.**
- **Donc, on peut considérer que les échantillons sont choisis de manière indépendante.**

II. La normalité des résidus

"Quantile-Quantile" plot



La distribution des résidus



Les résidus s'écarte de la quantile théorique

Vérification de normalité

Test de "Shapiro_Wilk"

- **Le test de Shapiro-Wilk donne une p-value < 0.05**
- **On ne peut pas rejeter l'hypothèse nulle du test et les résidus ne sont pas normaux.**

```
stats.shapiro(model.resid_pearson)
```

```
ShapiroResult(statistic=0.9566909074783325, pvalue=0.0)
```

Vérification de normalité avec test non_parametric

Kruskal-Wallis

- **Le test de Kruskal-Wallis donne une p-value de 0**
- **on peut avec un certain degré de confiance rejeter l'hypothèse null (les moyennes sont égaux)**

```
stats.kruskal(appli_score2["nutriscore_grade"], appli_score2["nutriscore_score"])
```

```
KruskalResult(statistic=257107.11034744608, pvalue=0.0)
```

III-Homoscédasticité

- **Le test de Levene donne une p-value de 0**
- **On peut avec affirmer l'uniformité de la variance de l'erreur de nos modèles**

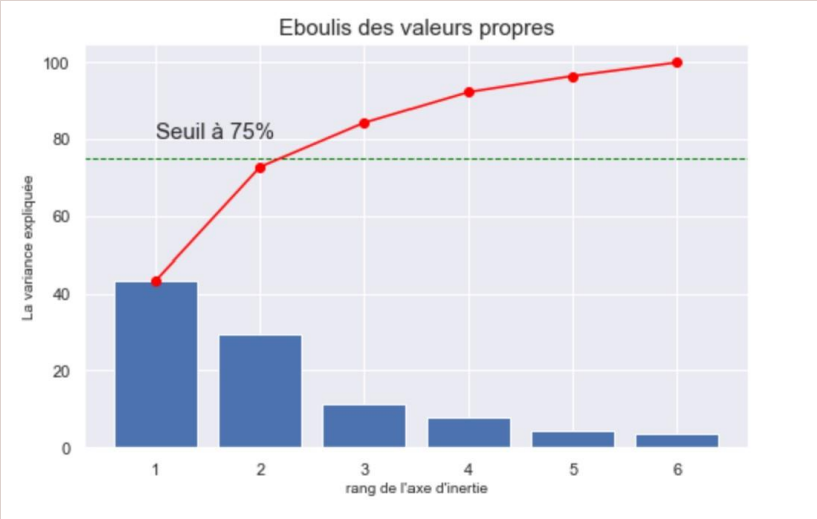
```
stats.levene(appli_score['nutriscore_score'][appli_score['nutriscore_grade'] == 'a'],
             appli_score['nutriscore_score'][appli_score['nutriscore_grade'] == 'b'],
             appli_score['nutriscore_score'][appli_score['nutriscore_grade'] == 'c'],
             appli_score['nutriscore_score'][appli_score['nutriscore_grade'] == 'd'],
             appli_score['nutriscore_score'][appli_score['nutriscore_grade'] == 'e'])
```

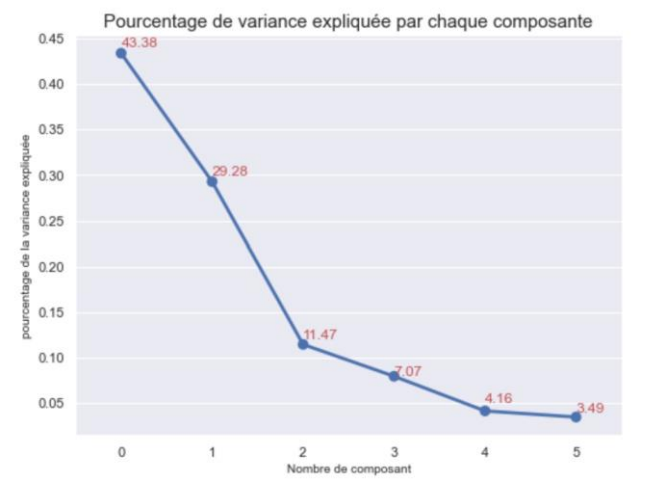
```
LeveneResult(statistic=17058.0279205483, pvalue=0.0)
```

Bilan sur le test d'ANOVA unidirectionnel

- **Les différences entre les moyennes des de deux variables "nutriscore_score" et "nutriscore_grade" sont statistiquement significatives**
- **Ces deux variables sont corrélées.**

5.4. Analyse multivariée
Analyse des composantes principales

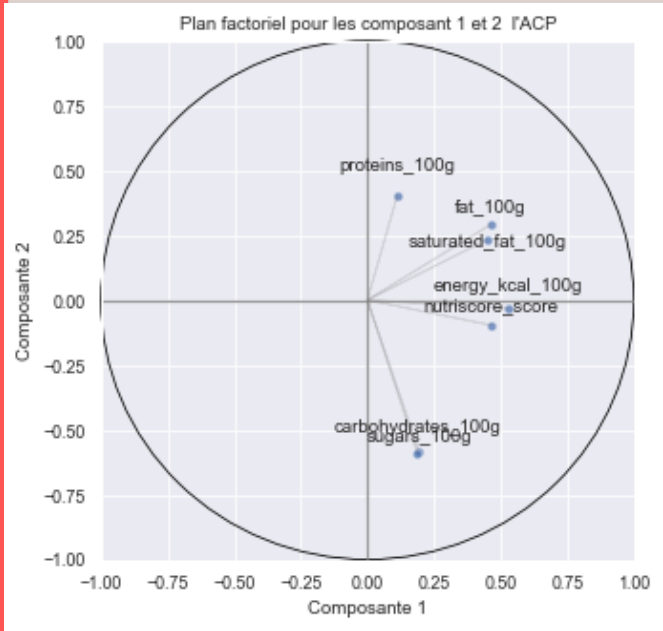




Avec les deux premières composantes principales on a une variance cumulée d'environs 75 %

Cercle de corrélations

Les composante 1 et 2



Les variables «fat», «energy », « saturated fat » et « nutriscore » sont fortement corrélées à l'axe principale 1.

- Le variables "protéin" est corrélées positivement à l'axe principale 2
- Les deux variables "sugars" et « carbohydrate » qui sont corrélées entre eux, sont anti-corrélée à l'axe principale 2

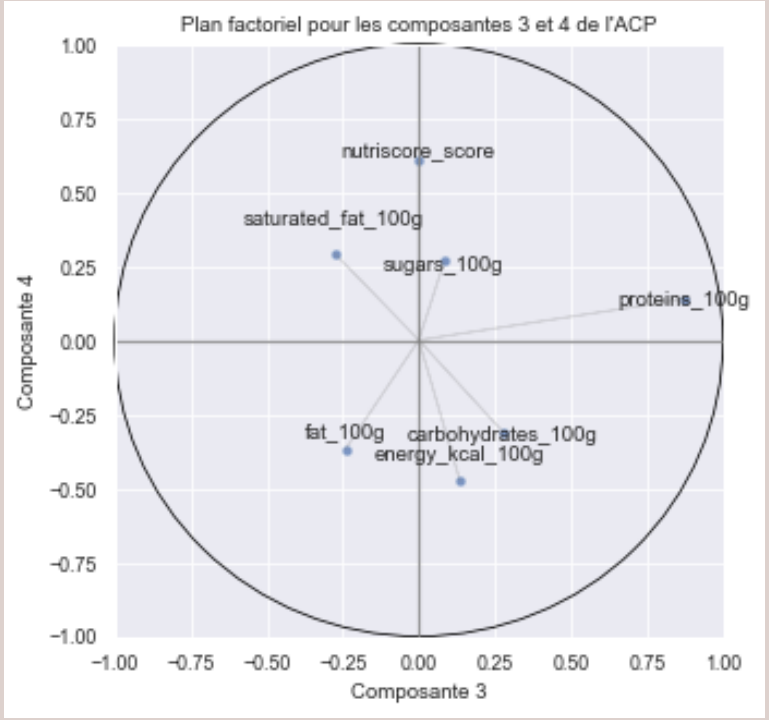
Cercle de corrélations

Les composante 3 et 4

- Le variable « protéines » est corrélées positivement à l'axe principal 3. Elle est de plus, assez proche du cercle des corrélations et donc très bien représentées.

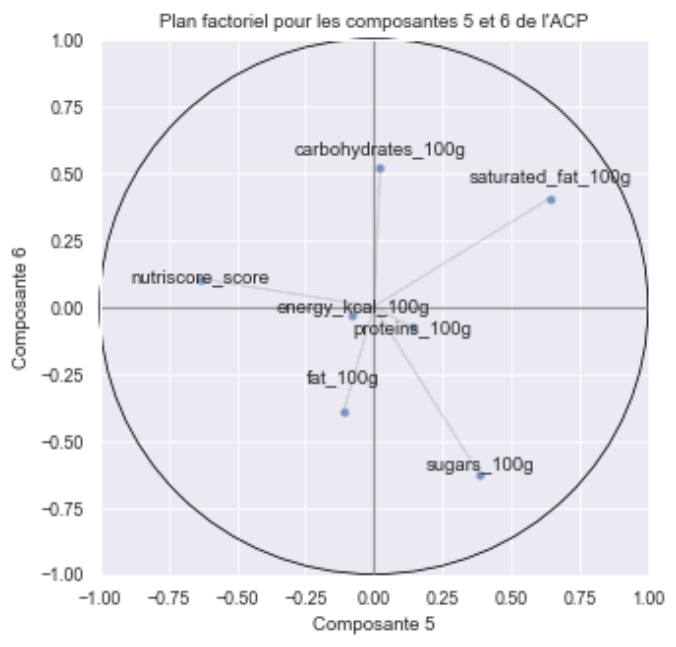
- Le variable « nutriscore_score » est corrélée positivement à pca4. Elle est de plus, assez proche du cercle des corrélations et donc très bien représentée.

- Les variables «fat», «energy» et « carbohydrate » sont corrélées négativement à pca4



Cercle de corrélations

Les composante 5 et 6



Les variables « sugar » et « saturated fat » sont corrélées positivement à pca5. Elles sont de plus, assez proche du cercle des corrélations et donc très bien représentées.

Les variables «energy» et «nutriscore» , fortement corrélées entre eux , sont anti-corrélées aux 5eme composantes.

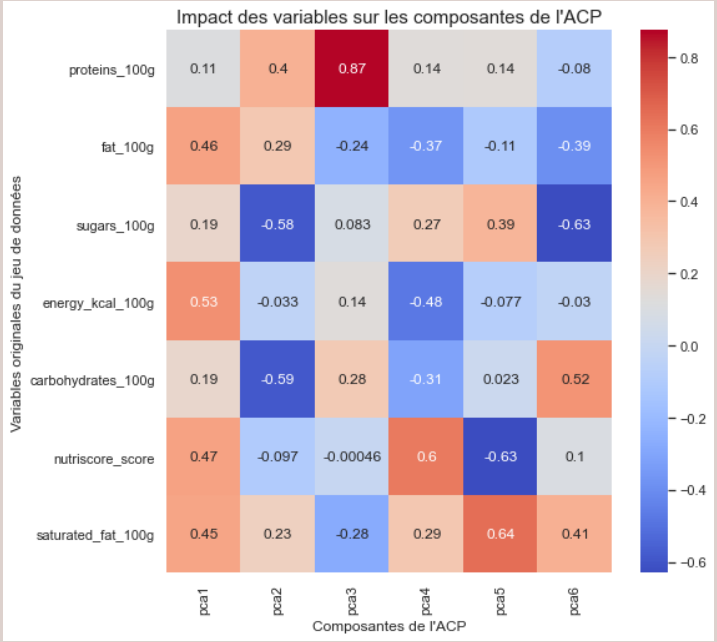
Les variables «carbohydrate» et «saturated_fat» sont corrélées positivement à pca6. Les variable « fat », « sugars » et «protéines» sont corrélées négativement au sixième axe principal.

Coefficient de coorélation

entre les variables et les axe principaux

- Les deux variables « nutriscore » et « protéines » sont les mieux représentées respectivement par les composantes 3 et 4

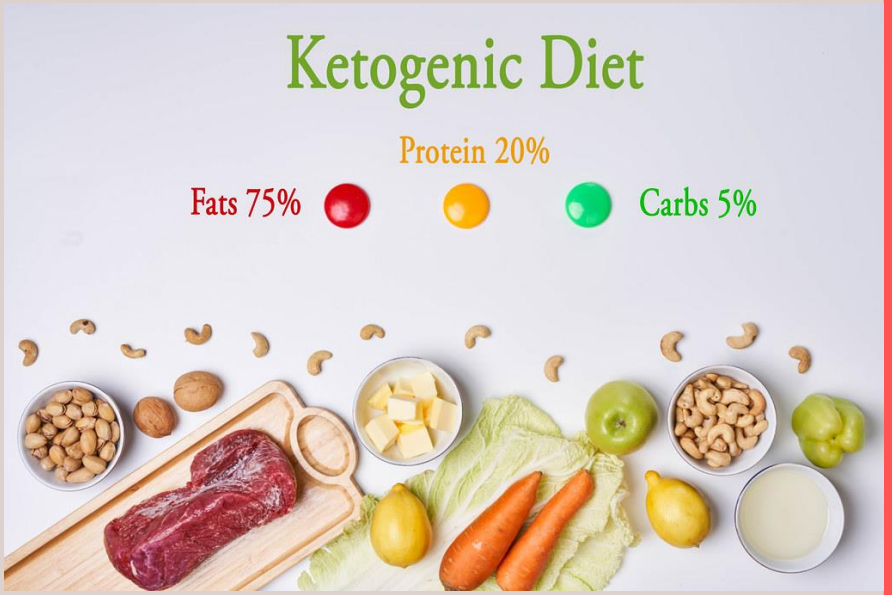
- Les trois variables « sugars », «carbohydrate » et «energy» sont anti_correlées au composant 2



6.Implémentation d’application

***L'apport nutritionnel* dans le régime cétogénique :**

- **75% Lipide**
- **20% Protéines**
- **5% Glucide**



6.1 Création de keto_score entre 0 et 10

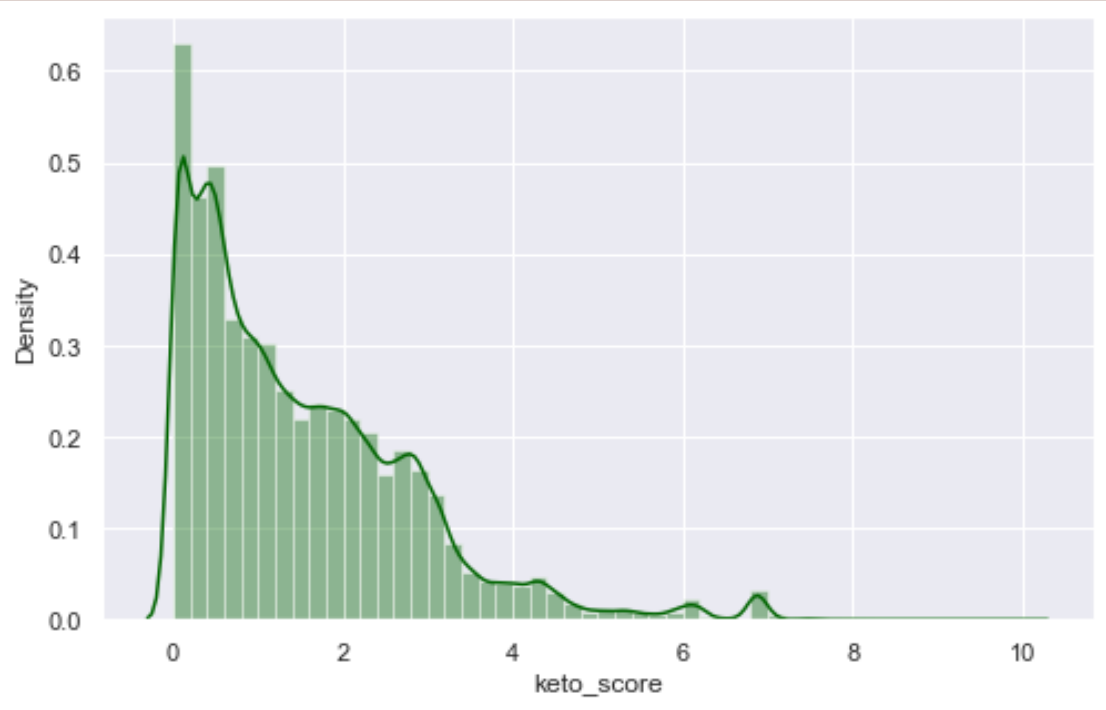
variables

```
# Trier les variables fianle pour tester l'idée de l'application
df_ketoscore= appli_final[["code", "product_name","proteins_100g", "fat_100g","main_category_en",
                           "carbohydrates_100g", "nutriscore_grade","energy_kcal_100g", 'nutriscore_score']]
df_ketoscore.reset_index(drop=True, inplace=True)
```

formule

```
# Créer un colonne de "keto" avec la formule en_dessus
df_ketoscore["keto"] = (df_ketoscore["proteins_100g"]*0.2) + (df_ketoscore['fat_100g']*0.75)\
                                          + (df_ketoscore['carbohydrates_100g']*0.05)

# La mise en echell de keto_score créer avec min/max méthode
df_ketoscore['keto_score'] = (df_ketoscore['keto']- df_ketoscore['keto'].min()) /
                                                    (df_ketoscore['keto'].max()-df_ketoscore['keto'].min())

df_ketoscore['keto_score'] = (df_ketoscore['keto_score']*10).round(decimals = 2)
```

Distribution de Keto_score



Skewness : 1.32

Kurtosis : 2.16

Médiane : 1.18

Moyenne : 1.5

Ecart_type : 1.31

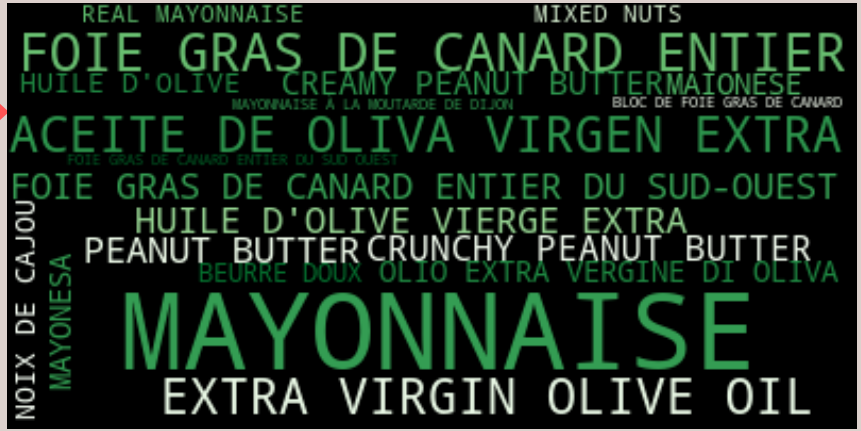
6.2 Création de keto_grade

Score 0-2	Score 2-4	Score 4-6	Score 6-8	Score 8-10
Very Poor	Poor	Good	Very Good	Excellent

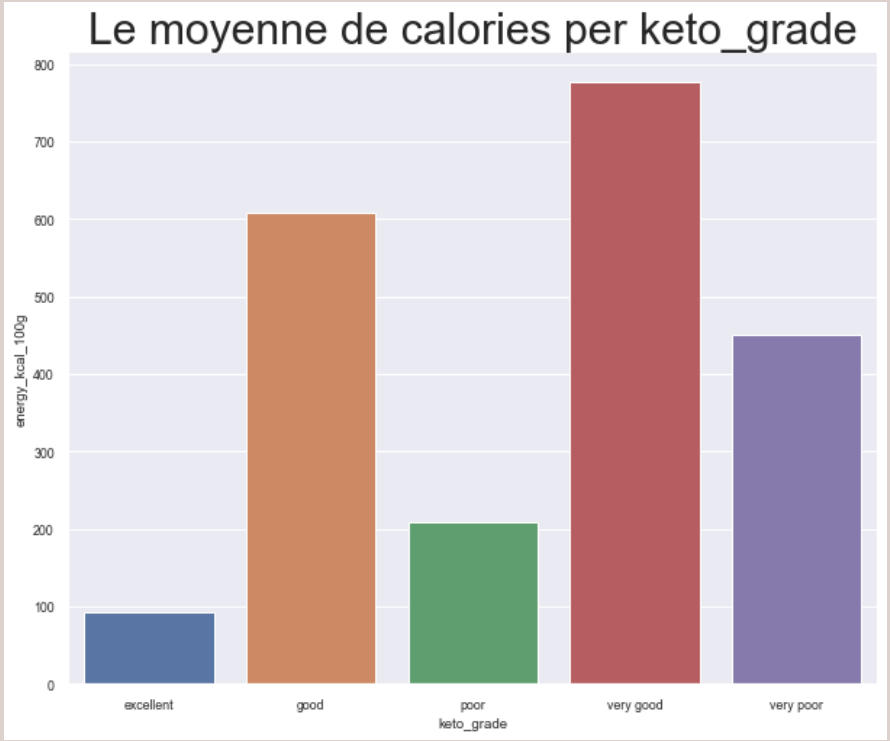
```
ketscore_map = [df_ketoscore["keto_score"].between(0, 2, inclusive= True), df_ketoscore["keto_score"].between(2.01, 4),
            df_ketoscore["keto_score"].between(4.01, 6), df_ketoscore["keto_score"].between(6.01, 8),
                df_ketoscore["keto_score"].between(8.01, 10, inclusive= True)]

keto_grade_values = ['poor', 'very poor', 'good', 'very good', 'excellent']
df_ketoscore["keto_grade"] = np.select(ketscore_map, keto_grade_values, default = 0)
```

***Pour la création keto_grade, on s'est inspiré de la méthodologie de l' Échelle de Likert**

20 produits les plus fréquents avec keto_grade «Goor » or « Very Goor »



20 produits les plus fréquents avec keto_grade «Poor » or « Very Poor »



- Les produits de keto_grade de "excellent" ne sont pas nécessairement les produits caloriques

6.3 Indication de classement cétogénique des aliments

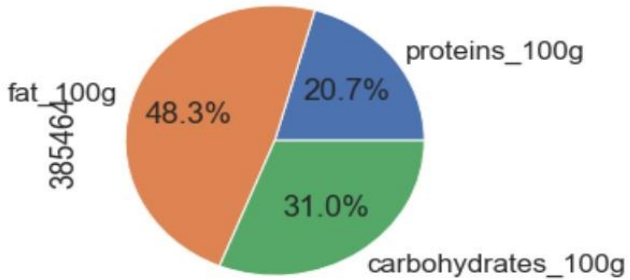
Créatio de la fonction keto_calcul


```
# Function de l'application keto_score
def keto_calcul (i) :
    df_test = df_ketoscore[df_ketoscore['code']== i][['product_name','proteins_100g',
                                                      'fat_100g','carbohydrates_100g',
                                                      'energy_kcal_100g','keto_grade',
                                                      'nutriscore_grade']].squeeze()
    print(df_test)
    return(df_test[['proteins_100g','fat_100g','carbohydrates_100g']].squeeze().plot.pie(autopct='%1.1f%%'))
```

6.4 Tester le résultat

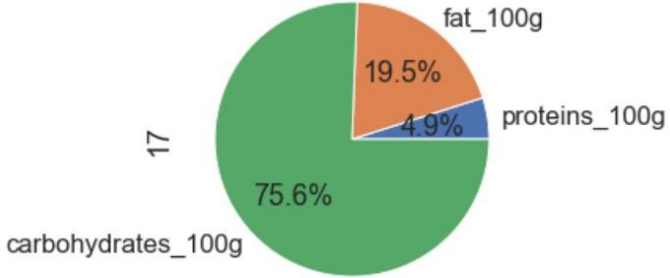
```
my_appli_keto('29212100100')
```

```
product_name           PISTACHIOS
proteins_100g                20.0
fat_100g                     46.7
carbohydrates_100g           30.0
energy_kcal_100g            633.0
keto_grade                   good
nutriscore_grade                C
Name: 385464, dtype: object

<AxesSubplot:ylabel='385464'>
```




```
my_appli_keto('0099482480219')
```

```
product_name           CARAMEL & SEA SALT ITALIAN GELATO
proteins_100g                                        2.0
fat_100g                                             8.0
carbohydrates_100g                                  31.0
energy_kcal_100g                                   200.0
keto_grade                                          poor
nutriscore_grade                                       D
Name: 17, dtype: object

<AxesSubplot:ylabel='17'>
```



7.Conclusion

1. **Possibilité de réaliser l'application keto_score sous réserve d'input:**
   - Protéines
   - Lipides
   - Carbohydrates

2. **Produit déjà présent avec ses variables remplies sur la base Open Food Fact**

8.Ouverture

Méthode de scoring par qualités :
à affiner avec la connaissance métier


Merci de votre attention!