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

QUENTIN STEPNIEWSKI

OPENCLASSROOMS

26 JUIN 2020

Sommaire

1. Introduction

Problématique principale

- Répondre à l'appel à projets lancé par Santé publique France

Objectif de l'étude

- Trouver une idée innovante d'application en lien avec l'alimentation

➢ Exploitation du jeu de données disponible sur Open Food Fact

2. Présentation BDD et idée d'application

a. Présentation de la base de données :

Les champs sont séparés en quatre sections :

- Les informations générales sur la fiche du produit : nom, date de modification, etc.
- Un ensemble de tags : catégorie du produit, localisation, origine, etc.
- Les ingrédients composant les produits et leurs additifs éventuels.
- Des informations nutritionnelles : quantité en grammes d'un nutriment pour 100 grammes du produit.

```
print('lignes: ',data.shape[0],'\n colonnes: ',data.shape[1])
```

```
lignes:  1383645
 colonnes:  181
```

	code	url	creator	created_t	created_datetime	last_modified_t	last_modified_datetime		phylloquinone_100g	beta-glucan_100g	inositol_100g	carnitine_100g
0	0000000000017	http://world-en.openfoodfacts.org/product/0000...	kiliweb	1529059080	2018-06-15T10:38:00Z	1561463718	2019-06-25T11:55:18Z		NaN	NaN	NaN	NaN
1	0000000000031	http://world-en.openfoodfacts.org/product/0000...	isagoofy	1539464774	2018-10-13T21:06:14Z	1539464817	2018-10-13T21:06:57Z		NaN	NaN	NaN	NaN
2	000000000003327986	http://world-en.openfoodfacts.org/product/0000...	kiliweb	1574175736	2019-11-19T15:02:16Z	1574175737	2019-11-19T15:02:17Z	(...)	NaN	NaN	NaN	NaN
3	0000000000100	http://world-en.openfoodfacts.org/product/0000...	del51	1444572561	2015-10-11T14:09:21Z	1444659212	2015-10-12T14:13:32Z		NaN	NaN	NaN	NaN
4	0000000001111111111	http://world-en.openfoodfacts.org/product/0000...	openfoodfacts-contributors	1560020173	2019-06-08T18:56:13Z	1560020173	2019-06-08T18:56:13Z		NaN	NaN	NaN	NaN

2. Présentation BDD et idée d'application

b. Présentation de l'idée d'application :

Mettre en place un indicateur de qualité de produit alimentaire (nutriscore) pour un utilisateur n'ayant que quelques informations sur un produit

2. Présentation BDD et idée d'application

b. Présentation de l'idée d'application :

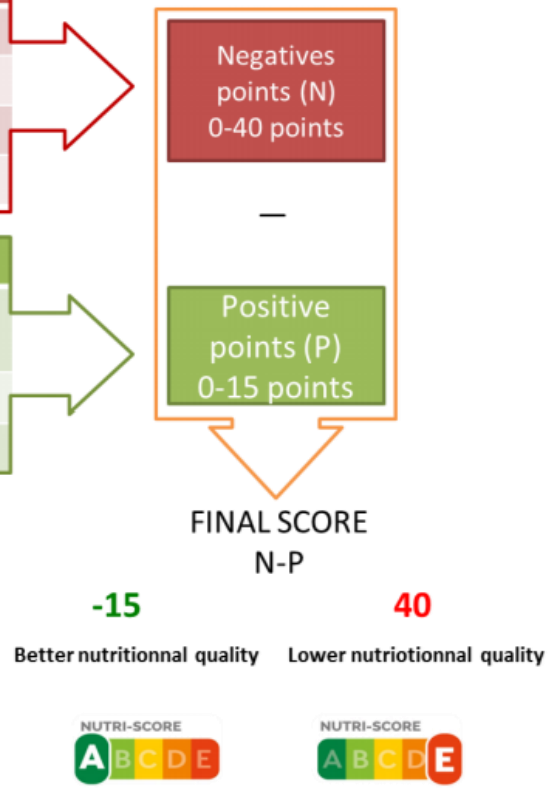
2 questions à se poser avant d'entrer dans la réalisation d'une proof of concept :

Q1. Comment calcule t-on le nutriscore de manière « exacte » ?

Nutriment /100g	Points
Energy (KJ)	0-10
Sugars (g)	0-10
Saturated Fatty acids (g)	0-10
Sodium (g)	0-10

Element /100g	Points
Fruits, vegetables, pulses, nuts, and rapeseed, walnut and olive oils, (%)	0-5
Fibres (g)	0-5
*Proteins (g)**	*0-5*

**Depending on the number of negative points and the content in « fruits, vegatables, pulses, nuts and oils », proteins are taken into account or not.*



2. Présentation BDD et idée d'application

b. Présentation de l'idée d'application :

2 questions à se poser avant d'entrer dans la réalisation d'une proof of concept :

Q2. Notre jeu de données est-il suffisamment consistant pour réaliser cette application ?

- Etude des individus ayant un nutriscore renseigné et un bon taux de remplissage général (ici arbitrairement moins de 80% de NaN)

```
df=df[df['nutriscore_score'].notnull()]
df.shape[0]
```

```
567328
```

- On obtient une base de données avec un niveau de remplissage très satisfaisant :

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 567328 entries, 3 to 1383639
Data columns (total 10 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   nutriscore_score    567328 non-null  float64
 1   energy_100g         565846 non-null  float64
 2   fat_100g            565805 non-null  float64
 3   saturated-fat_100g  565790 non-null  float64
 4   carbohydrates_100g  565544 non-null  float64
 5   sugars_100g         565796 non-null  float64
 6   fiber_100g          353943 non-null  float64
 7   proteins_100g       565808 non-null  float64
 8   salt_100g           566208 non-null  float64
 9   sodium_100g         566207 non-null  float64
dtypes: float64(10)
memory usage: 47.6 MB
```

3. Nettoyage de la base de données

Séparation de notre base de données en 2 sets distincts :

- Un dataset “train” afin de pouvoir entrainer notre futur modèle de prédiction du nutriscore (70% des données)
- Un dataset “test” afin de tester le niveau de précision de notre modèle en “ mise en production” (30% des données)

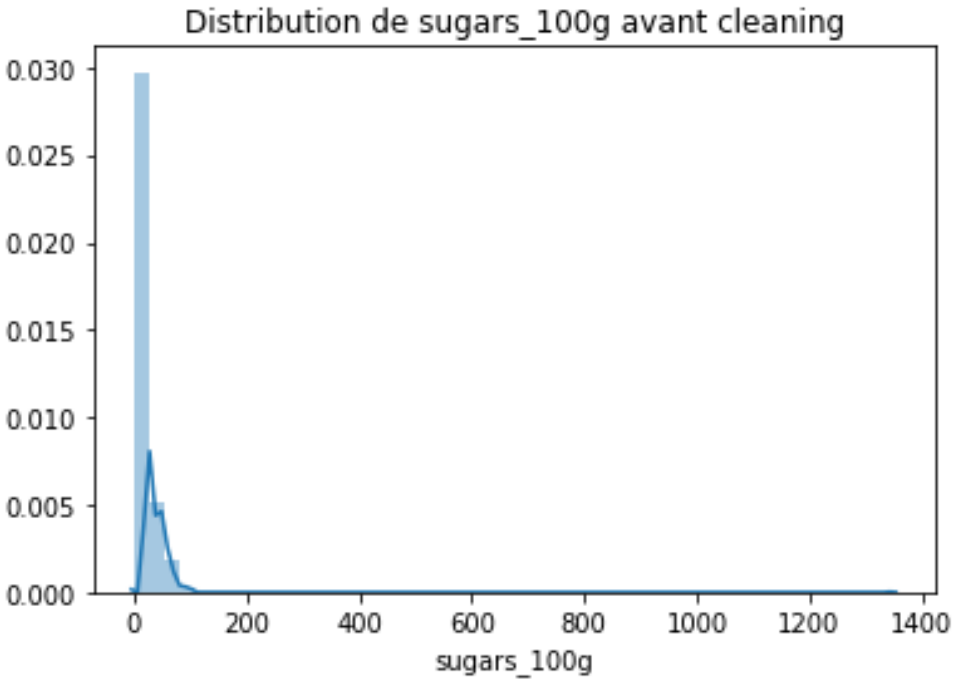
```
train, test = train_test_split(df, test_size=0.3)
train.shape[0]
```

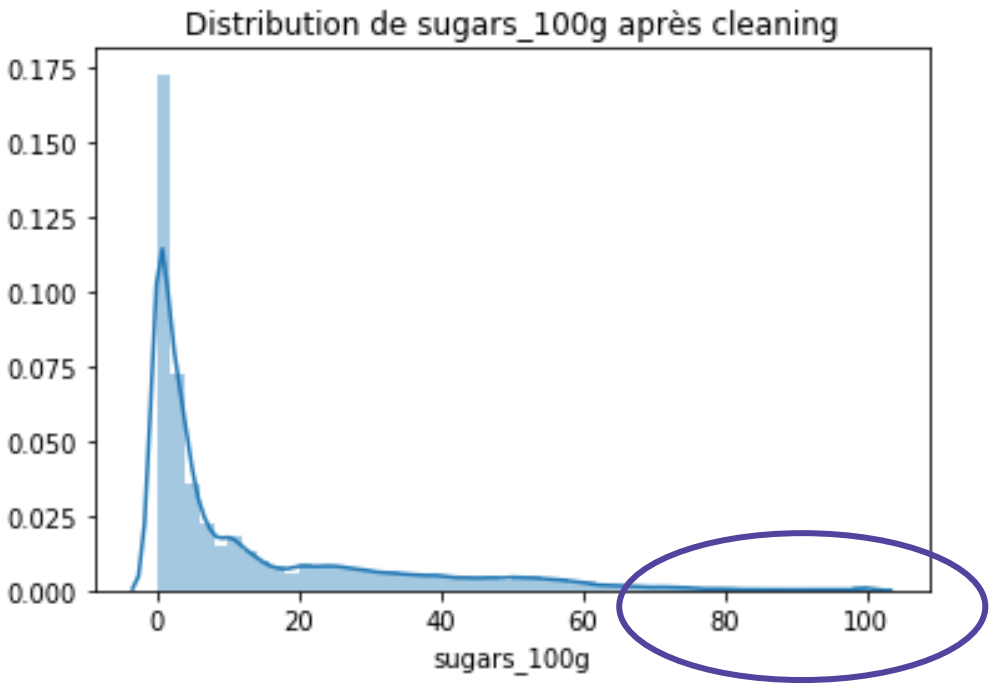
```
397129
```

3. Nettoyage de la base de données

Liste des différentes étapes de cleaning :

1. Mise à NaN des valeurs < 0g et > 100g



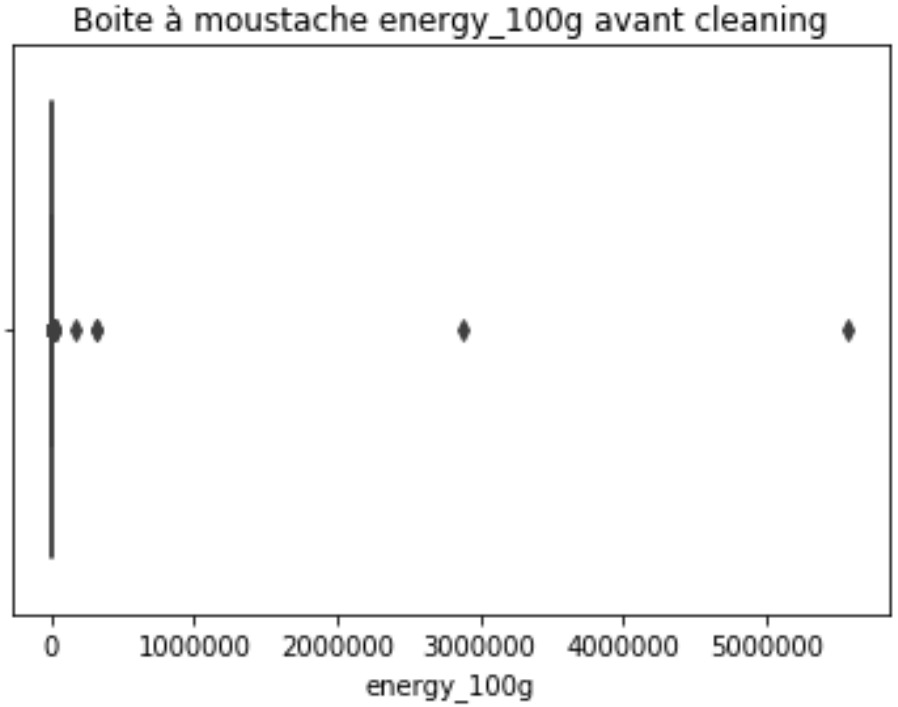


* Une partie de ces valeurs extrêmes devront être traitées via le cleaning d'outliers multivariés (partie 5)

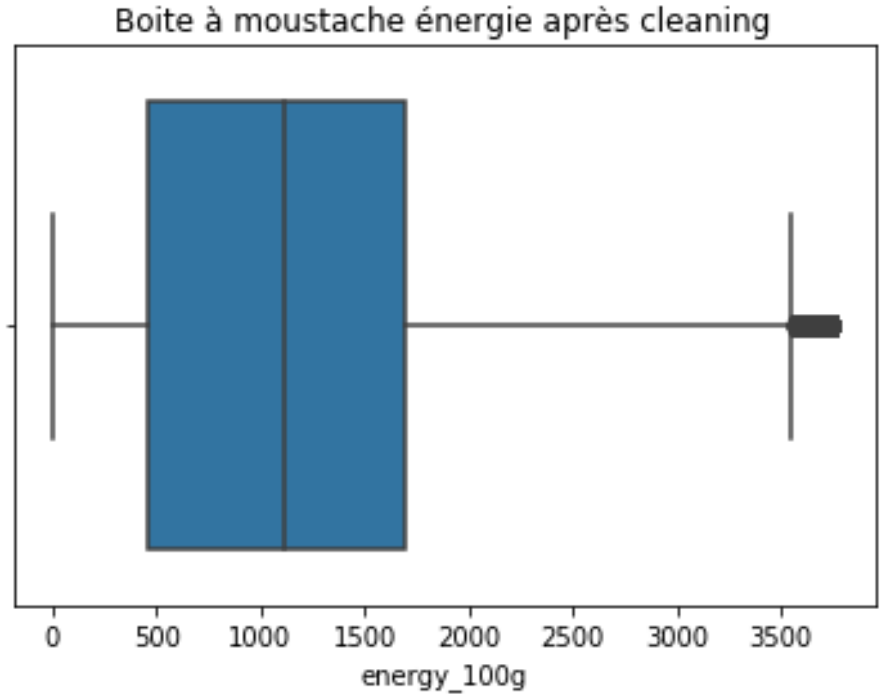
3. Nettoyage de la base de données

Liste des différentes étapes de cleaning :

1. Mise à NaN des valeurs < 0g et > 100g
2. Mise à NaN des valeurs extrêmes pour l'énergie via la méthode des percentiles



99,5% Percentile :
3761 Kj



3. Nettoyage de la base de données

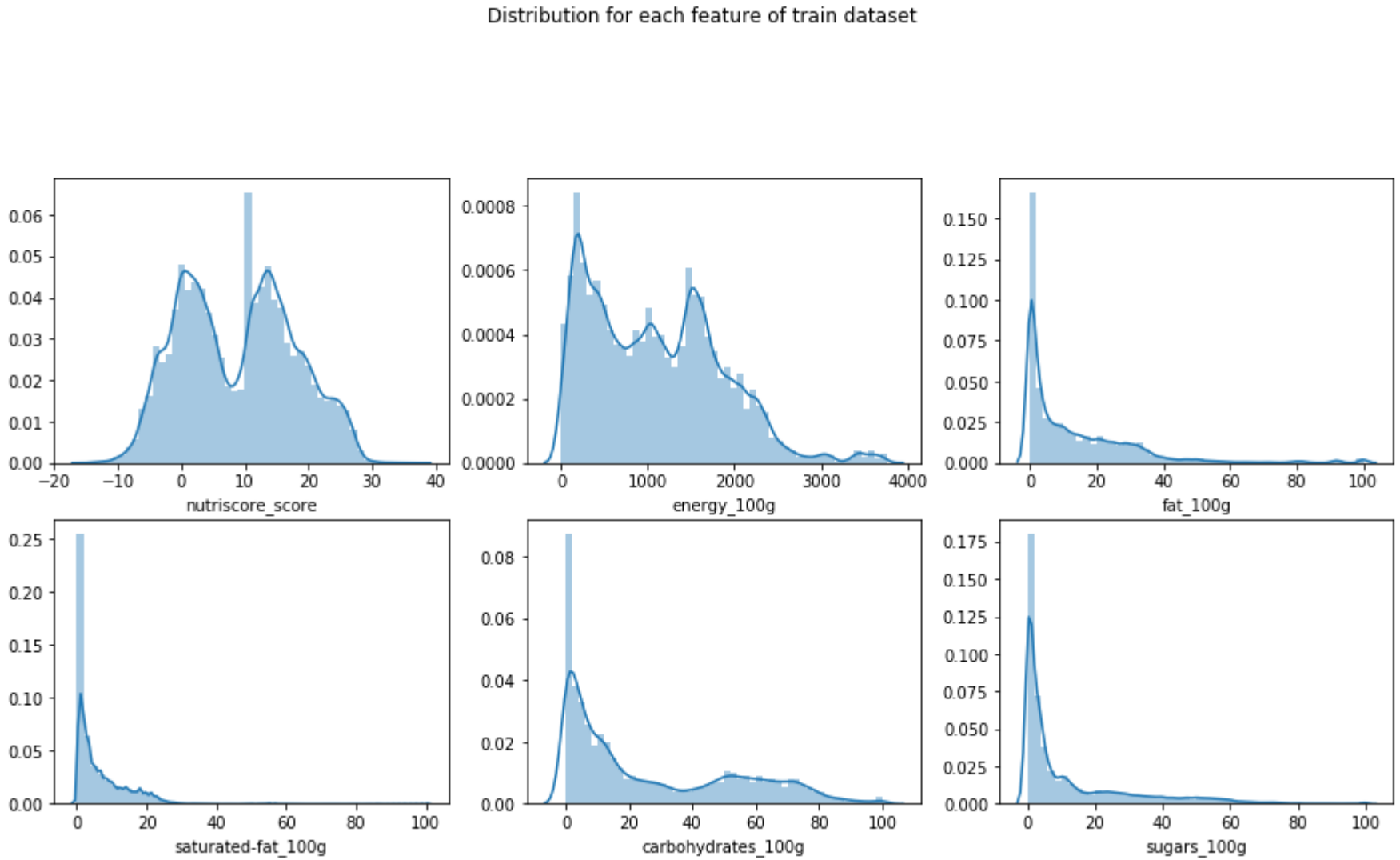
Liste des différentes étapes de cleaning :

1. Mise à NaN des valeurs < 0g et > 100g
2. Mise à NaN des valeurs extrêmes pour l'énergie via la méthode des percentiles
3. Suppression des individus ayant plus de 2 NaN par ligne
4. Imputation des NaN restants via KNN Imputer
5. Suppression des valeurs aberrantes par somme (somme supérieure à 100g)
6. Suppression des outliers multivariés par distance euclidienne (KDTree)

3. Nettoyage de la base de données

Liste de

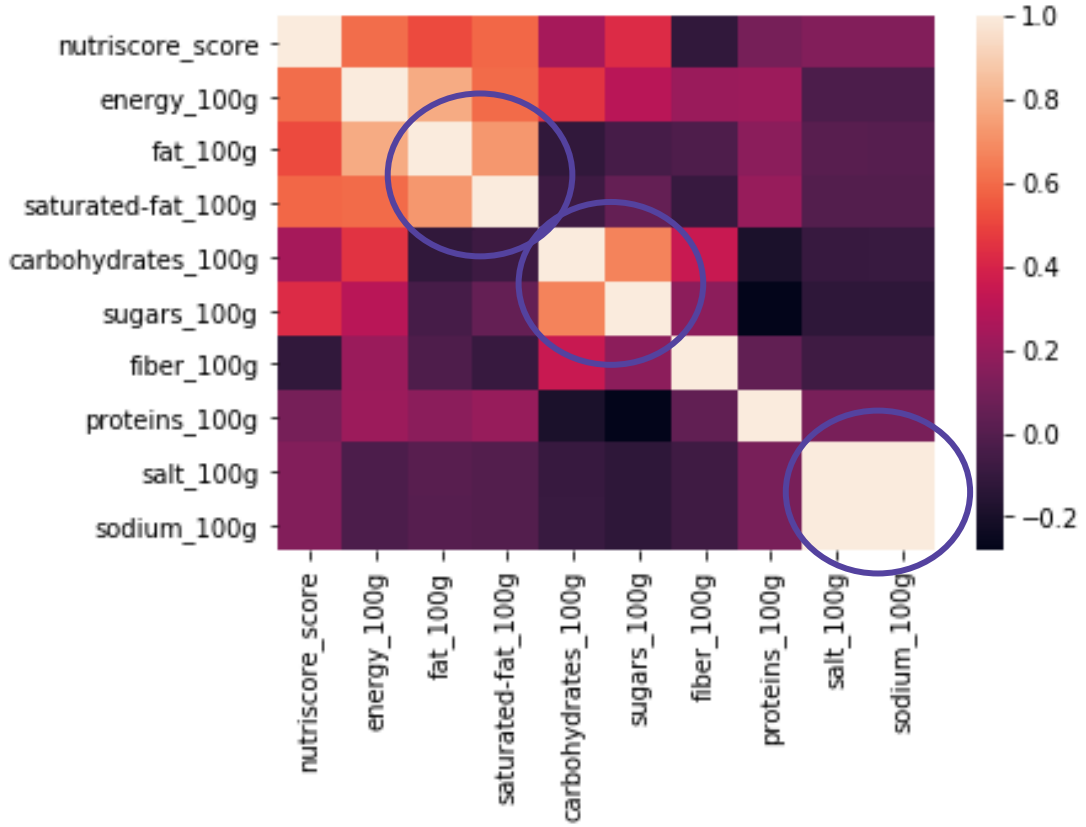
1. Mis
2. Mis
3. Sup
4. Imp
5. Sup
6. Sup



4. Analyse et exploitation

a. Analyse :

1. Matrice de corrélation (coefficient de Pearson)
2. Test de normalité (Kosmogorov-Smirnov)
3. Test ANOVA
4. ACP



On observe une assez forte corrélation pour les couples suivants:

- fat/ saturated-fat
- carbohydrates/ sugars
- salt/sodium

4. Analyse et exploitation

a. Analyse :

1. Matrice de corrélation (coefficient de Pearson)

2. Test de normalité (Kosmogorov-Smirnov)

3. Test ANOVA

4. ACP

```
====================
null hypothesis : studied feature follow a normal distribution.
 ====================


D= 0.718031325990802
P-value= 0.0
P-value is lower than alpha (0.05) we can reject the null hypothesis
 => nutriscore_score does not follow normal distribution
----------------

D= 0.9882169779727689
P-value= 0.0
P-value is lower than alpha (0.05) we can reject the null hypothesis
 => energy_100g does not follow normal distribution
----------------

D= 0.6546795418476797
P-value= 0.0
P-value is lower than alpha (0.05) we can reject the null hypothesis
 => fat_100g does not follow normal distribution
----------------
```

4. Analyse et exploitation

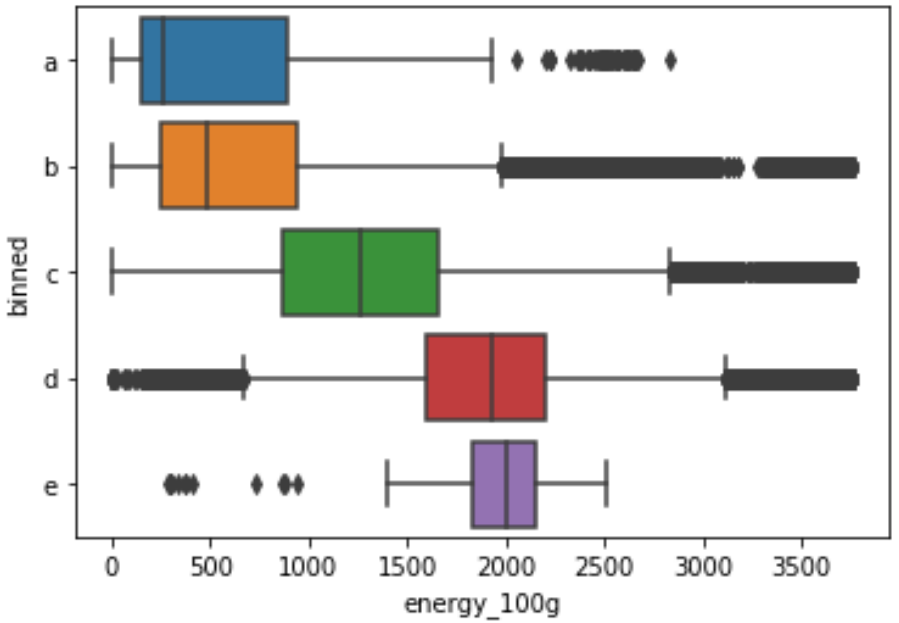
a. Analyse :

1. Matrice de corrélatic
(coefficient de Pears

2. Test de normalité
(Kosmogorov-Smirn

3. Test ANOVA

4. ACP



	F	P_value
energy_100g	48402.018888	0.0
fat_100g	29275.871175	0.0
saturated-fat_100g	47963.338632	0.0
carbohydrates_100g	6747.897986	0.0
sugars_100g	18350.684481	0.0
fiber_100g	3257.267355	0.0
proteins_100g	1461.444236	0.0
salt_100g	2663.101043	0.0
sodium_100g	2623.958628	0.0

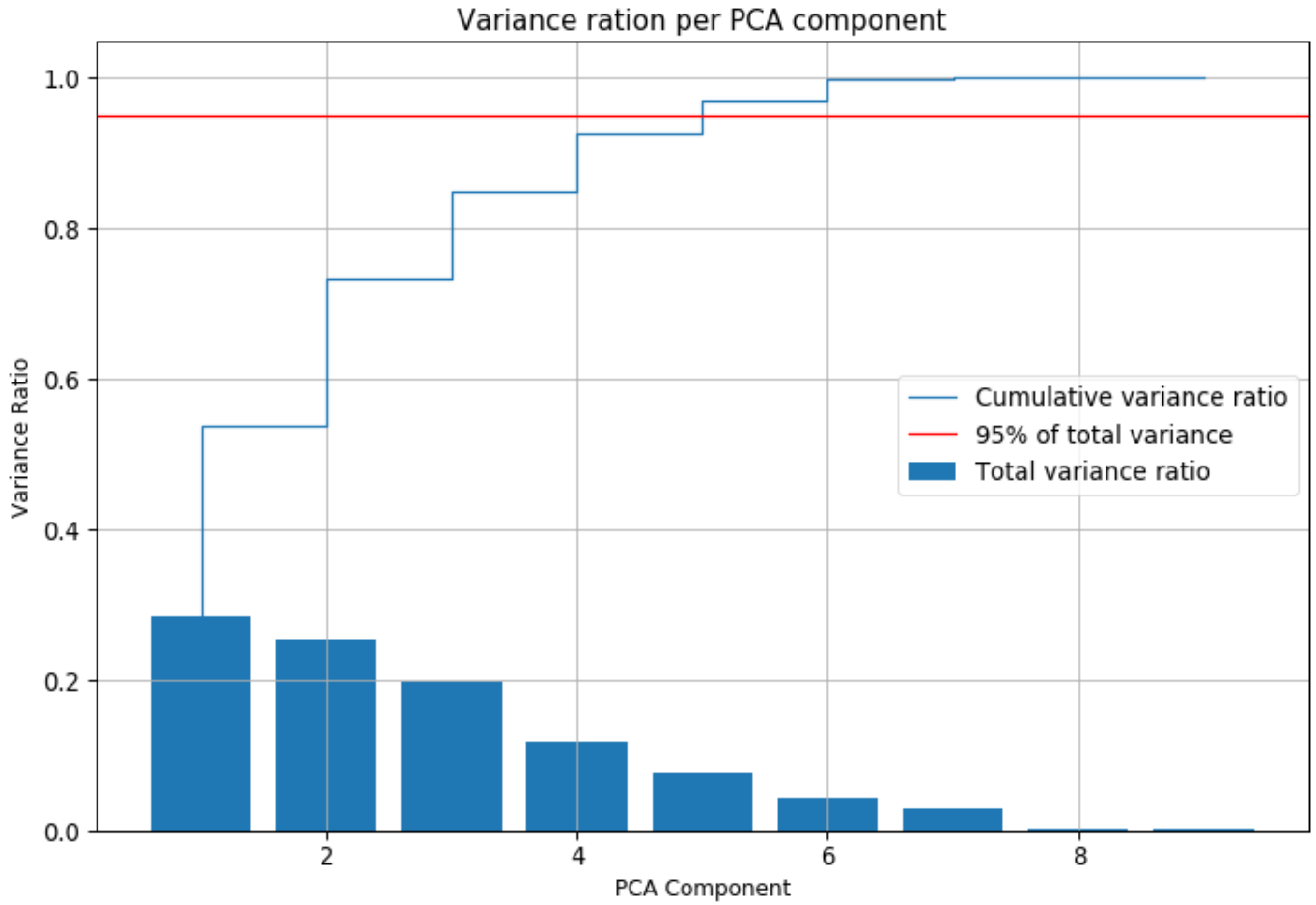
P Value= 0 (on peut rejeter l'hypothèse H0 avec 0% de risque, les catégories ont donc une influence sur la distribution des variables étudiées)

**Résultat du test à relativiser, on ne respecte pas la condition normalité de nos données (nécessaire pour un test ANOVA)*

4. Analyse et exploitation

a. Analyse :

1. Matrice de corrélation (coefficient de Pearson)
2. Test de normalité (Kosmogorov-Smirnov)
3. Test ANOVA
4. ACP

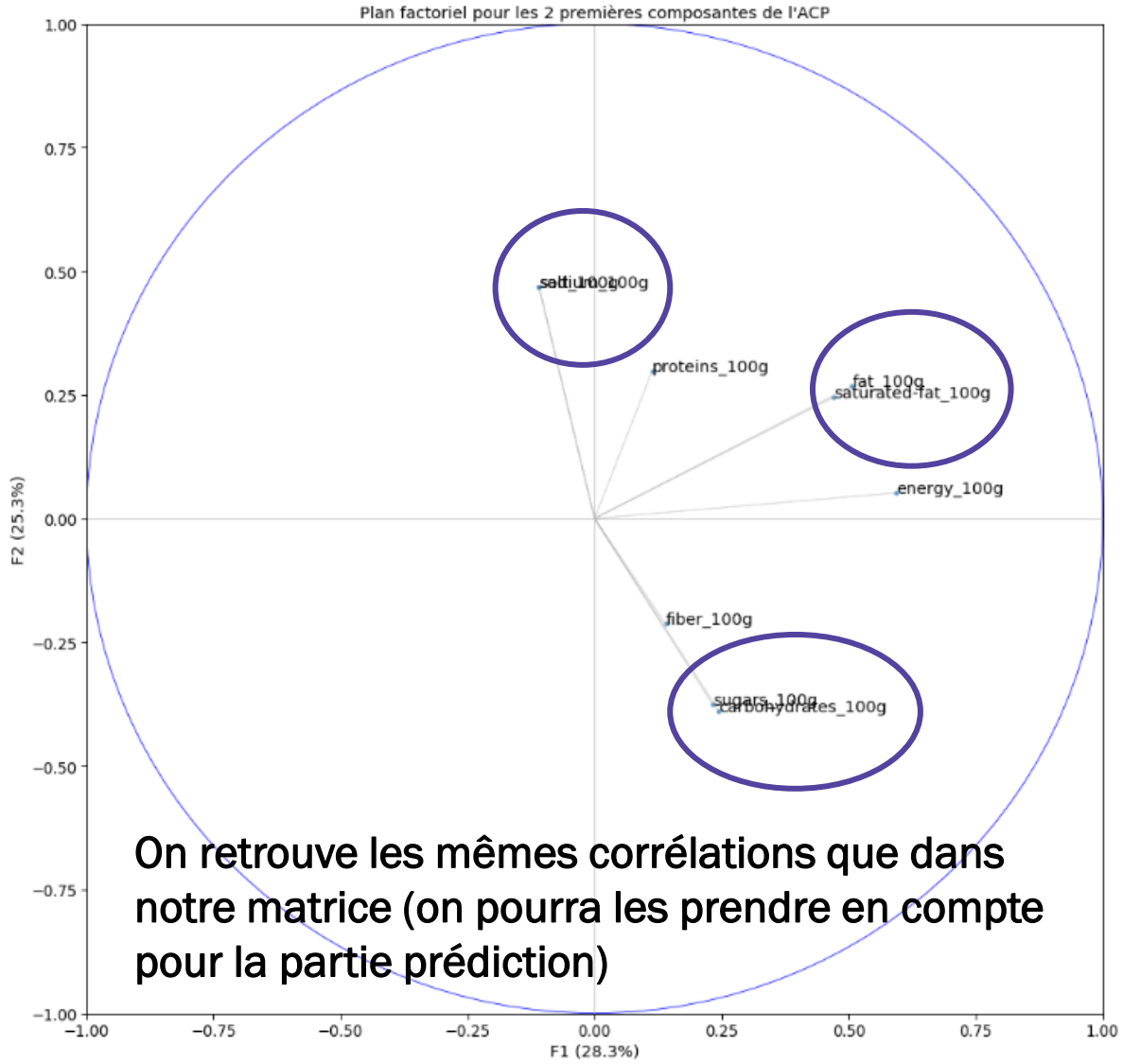


D'après cette vue, on pourrait représenter ~95% de nos données uniquement avec 5 features.
Ici, notre nombre de features est déjà suffisamment petit.

4. Analyse et exploitation

a. Analyse :

1. Matrice de corrélation (coefficient de Pearson)
2. Test de normalité (Kosmogorov-Smirnov)
3. Test ANOVA
4. ACP



On retrouve les mêmes corrélations que dans notre matrice (on pourra les prendre en compte pour la partie prédiction)

4. Analyse et exploitation

b. Exploitation :

Création d'une fonction de cleaning similaire au cleaning du train :

Liste des différentes étapes de cleaning :

1. Mise à NaN des valeurs < 0g et > 100g
2. Mise à NaN des valeurs extrêmes pour l'énergie via la méthode des percentiles
3. ~~Suppression des individus ayant plus de 2 NaN par ligne~~
4. Imputation des NaN restants via KNN Imputer
5. Suppression des valeurs aberrantes par somme (somme supérieure à 100g)
6. Suppression des outliers multivariés par distance euclidienne (KDTree)

Fitté sur le train dataset

4. Analyse et exploitation

b. Exploitation :

Création d'une fonction de cleaning similaire au cleaning du train :

Liste des diffé

1. Mise à Nal
2. Mise à Nal ercentiles
3. ~~Suppressic~~
4. Imputatior
5. Suppressic 100g)
6. Suppression des outliers multivariés par distance euclidienne (KDTree)

```
======================
Début de la phase de cleaning
======================
Gestion des valeurs inférieures à 0g et supérieures à 100g:
138  valeurs aberrantes mises à NaN
_____________________________
Gestion des outliers pour la variable: energy_100g:
850  valeurs aberrantes mises à NaN
_____________________________
Remplissage des NaNs via KNN Imputer (fitté sur le train)
Les Nans du dataframe ont été imputés via KNNImputer
_____________________________
Suppression des lignes avec une somme des macronutriments supérieure à 100g:
11106  lignes supprimées
_____________________________
Suppression outliers multivariés avec KDTree:
2351  lignes supprimées
_________________________
```

Fitté sur le train dataset

4. Analyse et exploitation

b. Exploitation :

Prédiction via régression linéaire multiple :

Score de la prédiction (R^2) : 0,626

	prédiction	nutriscore
count	159037.000000	148610.000000
mean	9.220771	9.030186
std	7.111163	8.893977
min	-44.044231	-15.000000
25%	3.660006	1.000000
50%	7.293254	10.000000
75%	13.987662	16.000000
max	58.235002	36.000000

Poids de chaque variable dans la régression:

	Feature	Coef
0	energy_100g	3.099802
1	fat_100g	0.010714
2	saturated-fat_100g	2.877117
3	carbohydrates_100g	-0.724187
4	sugars_100g	3.840426
5	fiber_100g	-1.707929
6	proteins_100g	0.554542
7	salt_100g	2.751464
8	sodium_100g	-1.190483

4. Analyse et exploitation

b. Exploitation :

Prédiction via régression KNN Regressor :

Ici, l'optimisation a été plus poussée

1) Utilisation d'une cross validation via GridSearchCV (5 Split)

4. Analyse et exploitation

b. Exploitation :

Prédiction via régression KNN Regressor :

Ici, l'optimisation a été plus poussée

2) GridSearchCV va également nous permettre de chercher les paramètres optimum pour notre modèle (ici uniquement le nombre de voisins)

3) Etude des features "trop corrélées" (non-concluante)

```
for i in range(len(scoring)):

    grid= GridSearchCV(KNeighborsRegressor(),param_grid,cv=5,scoring=scoring[i])
    grid.fit(xtrain,ytrain)

    print('Methode de Scoring: ',scoring[i])
    print('Meilleur score :',grid.best_score_)
    print('Meilleurs Paramètres :',grid.best_params_,'\n_________________')

    if i == 0 :
        model_1 = grid.best_estimator_
    elif i == 1 :
        model_2 = grid.best_estimator_
    else:
        model_3 = grid.best_estimator_
```

```
Methode de Scoring:  r2
Meilleur score : 0.9485970162173958
Meilleurs Paramètres : {'n_neighbors': 5}
_________________
Methode de Scoring:  neg_mean_absolute_error
Meilleur score : -0.921002810035702
Meilleurs Paramètres : {'n_neighbors': 1}
_________________
Methode de Scoring:  neg_mean_squared_error
Meilleur score : -4.062249569922076
Meilleurs Paramètres : {'n_neighbors': 5}
_________________
```

4. Analyse et exploitation

b. Exploitation :

Prédiction via régression KNN Regressor :

Ici, l'optimisation a été plus poussée

3) Etude des features "trop corrélées" (non-concluante) :

Suppresion pas à pas des variables trop corrélées entre-elles

Rappel score avec toutes les variables

```
Methode de Scoring:  r2
Meilleur score : 0.9485970162173958
Meilleurs Paramètres : {'n_neighbors': 5}
```

	col	r2	neg_mean_absolute_error	neg_mean_squared_error
sans saturated fat	saturated-fat_100g	0.923408	-1.259147	-6.052823
sans sugars	sugars_100g	0.932598	-1.124096	-5.326558
sans sodium	sodium_100g	0.946746	-0.957152	-4.208524
sans saturated fat & sugars	[saturated-fat_100g, sugars_100g]	0.897964	-1.509248	-8.063685
sans saturated fat & sodium	[saturated-fat_100g, sodium_100g]	0.9221	-1.277486	-6.156246
sans sugars et sodium	[sugars_100g, sodium_100g]	0.931532	-1.144292	-5.410799
sans saturates fat & sugars & sodium	[saturated-fat_100g, sugars_100g, sodium_100g]	0.89735	-1.519647	-8.112165

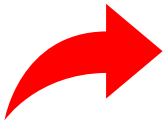
4. Analyse et exploitation

b. Exploitation :

Prédiction via regression KNN Regressor :

Ici, l'optimisation a été plus poussée

- Utilisation d'une cross validation via GridSearchCV (5 Split)
- GridSearchCV va également nous permettre de chercher les paramètres optimum pour notre modèle (ici uniquement le nombre de voisins)
- Etude des features "trop corrélées" (non-concluante)



Score final sur le test dataset (R^2): 0,93

5.

On a

- Alg
- Pré
- Pos

produ

Il fau

poss

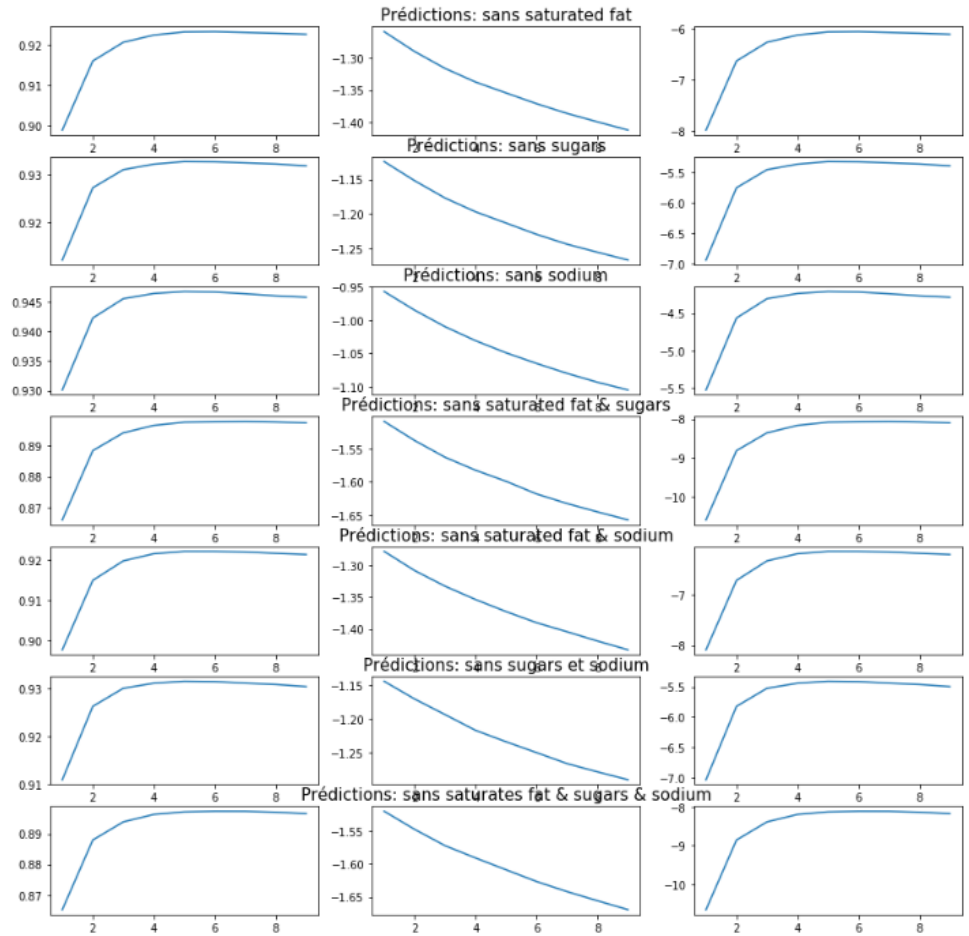
	nutriscore_score	energy_100g	fat_100g	saturated-fat_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g	sodium_100g
0	-6.0	444.0	0.59	0.0	22.35	9.41	2.4	3.53	0.22	0.088
1	-6.0	444.0	0.59	0.0	22.35	NaN	2.4	3.53	0.22	0.088
2	-6.0	444.0	0.59	NaN	NaN	9.41	2.4	3.53	0.22	0.088
3	-6.0	444.0	0.59	0.0	NaN	NaN	2.4	3.53	NaN	0.088
4	-6.0	NaN	0.59	0.0	NaN	9.41	2.4	NaN	NaN	0.088
5	-6.0	NaN	NaN	NaN	22.35	NaN	2.4	3.53	NaN	0.088
6	-6.0	NaN	NaN	NaN	22.35	9.41	2.4	NaN	NaN	NaN
7	-6.0	NaN	NaN	NaN	NaN	9.41	NaN	NaN	NaN	0.088

	nutriscore_score	energy_100g	fat_100g	saturated-fat_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g	sodium_100g	predict
0	-6.0	444.000000	0.590000	0.000000e+00	22.350000	9.41	2.400	3.530000	0.220000	0.088	-6.0
1	-6.0	444.000000	0.590000	0.000000e+00	22.350000	8.43	2.400	3.530000	0.220000	0.088	-6.0
2	-6.0	444.000000	0.590000	-8.881784e-16	22.350000	9.41	2.400	3.530000	0.220000	0.088	-6.0
3	-6.0	444.000000	0.590000	0.000000e+00	22.350000	8.43	2.400	3.530000	0.000000	0.088	-6.0
4	-6.0	185.800000	0.590000	0.000000e+00	12.626667	9.41	2.400	1.546667	0.483483	0.088	-3.8
6	-6.0	444.000000	0.590000	-8.881784e-16	22.350000	9.41	2.400	3.530000	0.000000	0.000	-6.0
7	-6.0	638.666667	7.951667	4.581667e+00	17.390000	9.41	0.926	5.280000	0.217333	0.088	3.8

Merci de votre attention

	col	r2	neg_mean_absolute_error	neg_mean_squared_error
sans saturated fat	saturated-fat_100g	0.923408	-1.259147	-6.052823
sans sugars	sugars_100g	0.932598	-1.124096	-5.326558
sans sodium	sodium_100g	0.946746	-0.957152	-4.208524
sans saturated fat & sugars	[saturated-fat_100g, sugars_100g]	0.897964	-1.509248	-8.063685
sans saturated fat & sodium	[saturated-fat_100g, sodium_100g]	0.9221	-1.277486	-6.156246
sans sugars et sodium	[sugars_100g, sodium_100g]	0.931532	-1.144292	-5.410799
sans saturates fat & sugars & sodium	[saturated-fat_100g, sugars_100g, sodium_100g]	0.89735	-1.519647	-8.112165

```
fig, axs = plt.subplots(matrix_features.shape[0],3,figsize=(15,15))
axs = axs.ravel()
counter = 0

for k in range(matrix_features.shape[0]):
    scaler = StandardScaler()
    print('Prediction: {}'.format(matrix_features.index[k]))
    train_used = train.drop(columns=matrix_features.col[k])
    xtrain = scaler.fit_transform(train_used.drop(columns=['nutriscore_score']))
    ytrain = train_used['nutriscore_score']

    param_grid = {'n_neighbors':np.arange(1,10)}
    scoring = ['r2','neg_mean_absolute_error','neg_mean_squared_error']

    for i in range(len(scoring)):

        grid= GridSearchCV(KNeighborsRegressor(),param_grid,cv=5,scoring=scoring[i])
        grid.fit(xtrain,ytrain)

#         print('Methode de Scoring: ',scoring[i])
#         print('Meilleur score :',grid.best_score_)
#         print('Meilleurs Paramètres :',grid.best_params_,'\n_____________________')
        matrix_features.iloc[k,i+1]=grid.best_score_

        if i == 0 :
            model_1 = grid.best_estimator_
            result_1 = pd.DataFrame(grid.cv_results_)
            axs[counter].plot(result_1['param_n_neighbors'],result_1['mean_test_score'])
            counter+=1
        elif i == 1 :
            model_2 = grid.best_estimator_
            result_2 = pd.DataFrame(grid.cv_results_)
            axs[counter].set_title('Prédictions: {}'.format(matrix_features.index[k]),fontsize=15)
            axs[counter].plot(result_2['param_n_neighbors'],result_2['mean_test_score'])
            counter+=1
        else:
            model_3 = grid.best_estimator_
            result_3 = pd.DataFrame(grid.cv_results_)
            axs[counter].plot(result_3['param_n_neighbors'],result_3['mean_test_score'])
            counter+=1

#     fig, axs = plt.subplots(3,figsize=(15,15))

matrix_features
```



Additional slide

```
test_nan_bis = test.iloc[170194:170196]
test_nan_bis=test_nan_bis.drop(index=170195)
test_nan_bis = pd.DataFrame(np.repeat(test_nan_bis.values,8,axis=0))
test_nan_bis.columns = test.columns
for i in range(test_nan_bis.shape[0]):
    if i!=0:
        nans = random.sample(range(1,10),i)
        for k in range(i):
            test_nan_bis.iloc[i,nans[k]]=np.nan
test_nan_bis
```

	nutriscore_score	energy_100g	fat_100g	saturated-fat_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g	sodium_100g
0	-6.0	444.0	0.59	0.0	22.35	9.41	2.4	3.53	0.22	0.088
1	-6.0	444.0	0.59	0.0	22.35	NaN	2.4	3.53	0.22	0.088
2	-6.0	444.0	0.59	NaN	NaN	9.41	2.4	3.53	0.22	0.088
3	-6.0	444.0	0.59	0.0	NaN	NaN	2.4	3.53	NaN	0.088
4	-6.0	NaN	0.59	0.0	NaN	9.41	2.4	NaN	NaN	0.088
5	-6.0	NaN	NaN	NaN	22.35	NaN	2.4	3.53	NaN	0.088
6	-6.0	NaN	NaN	NaN	22.35	9.41	2.4	NaN	NaN	NaN
7	-6.0	NaN	NaN	NaN	NaN	9.41	NaN	NaN	NaN	0.088

	nutriscore_score	energy_100g	fat_100g	saturated-fat_100g	carbohydrates_100g	sugars_100g	fiber_100g	proteins_100g	salt_100g	sodium_100g	predict
0	-6.0	444.000000	0.590000	0.000000e+00	22.350000	9.41	2.400	3.530000	0.220000	0.088	-6.0
1	-6.0	444.000000	0.590000	0.000000e+00	22.350000	8.43	2.400	3.530000	0.220000	0.088	-6.0
2	-6.0	444.000000	0.590000	-8.881784e-16	22.350000	9.41	2.400	3.530000	0.220000	0.088	-6.0
3	-6.0	444.000000	0.590000	0.000000e+00	22.350000	8.43	2.400	3.530000	0.000000	0.088	-6.0
4	-6.0	185.800000	0.590000	0.000000e+00	12.626667	9.41	2.400	1.546667	0.483483	0.088	-3.8
6	-6.0	444.000000	0.590000	-8.881784e-16	22.350000	9.41	2.400	3.530000	0.000000	0.000	-6.0
7	-6.0	638.666667	7.951667	4.581667e+00	17.390000	9.41	0.926	5.280000	0.217333	0.088	3.8