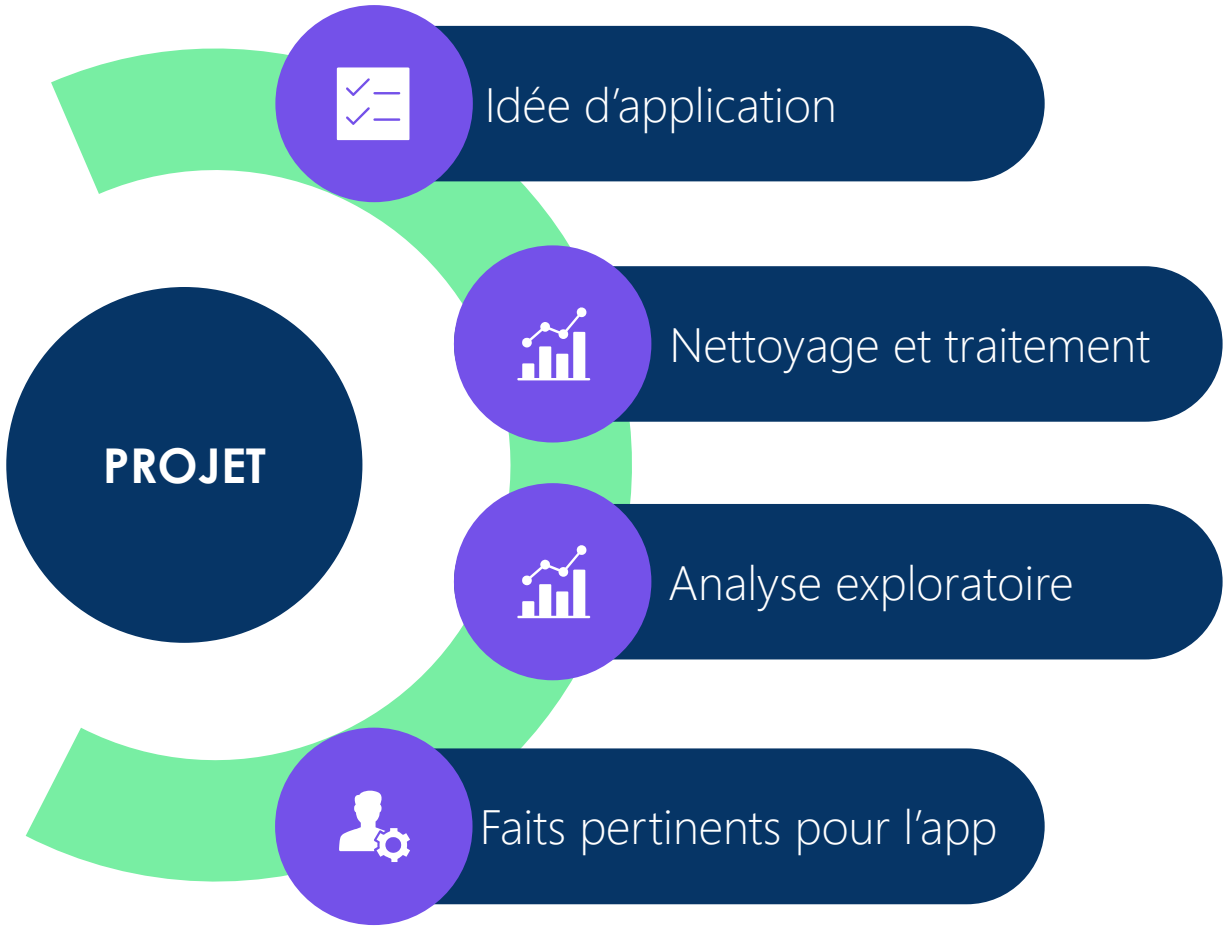
Projet N°3 :
Concevez une application au service de la santé publique

Agustin Bunader (autofinancé)
Soutenance de Projet
Décembre 2020

Programme


PROJET
Idée d'application
Nettoyage et traitement
Analyse exploratoire
Faits pertinents pour l'app

Idée d'application

Une application qu'indique à l'utilisateur quels nutriments sont en excès et si l'utilisateur a encore une marge de consommation selon les *Apports Journaliers Recommandés (AJR)*. Les données utilisées seront limitées uniquement à la France.

L'utilisateur saisit ce qu'il a consommé au cours de la journée, en respectant les catégories *PNNS (Programme national nutrition santé)* et leur quantité correspondante en grammes.

Les valeurs utilisées pour les calculs internes proviennent d'*Open Food Facts* et sont travaillées en respectant les critères suivants :

- Pour chaque groupe *PNNS*, son *grade* de *Nutri-Score* sera calculé en prenant la moyenne des scores en effectuant un filtre sur la colonne des scores avec les groupes *PNNS* comme condition.

- Les nutriments correspondant à chaque groupe *PNNS* sont calculés avec les mêmes critères que le *Nutri-Score* détaillé ci-dessus.

Nettoyage et traitement - Downcast

Un premier *downcast* (bibliothèque Pandas) sur les int et float dans le *dataframe* généré par le csv source afin de réduire la mémoire utilisée lors des opérations de nettoyage et d'analyse exploratoire.

```
In [6]: def downCast(df):
            cols = df.select_dtypes(include=['float64']).columns.tolist()
            for col in cols:
                df[col] = pd.to_numeric(df[col], downcast='float')
            cols = df.select_dtypes(include=['int64']).columns.tolist()
            for col in cols:
                df[col] = pd.to_numeric(df[col], downcast='integer')
            df.info(verbose=True,null_counts=True)
            return df
```

Avant

```
dtypes: float64(122), int64(2), object(58)
memory usage: 2.0+ GB
```

Après

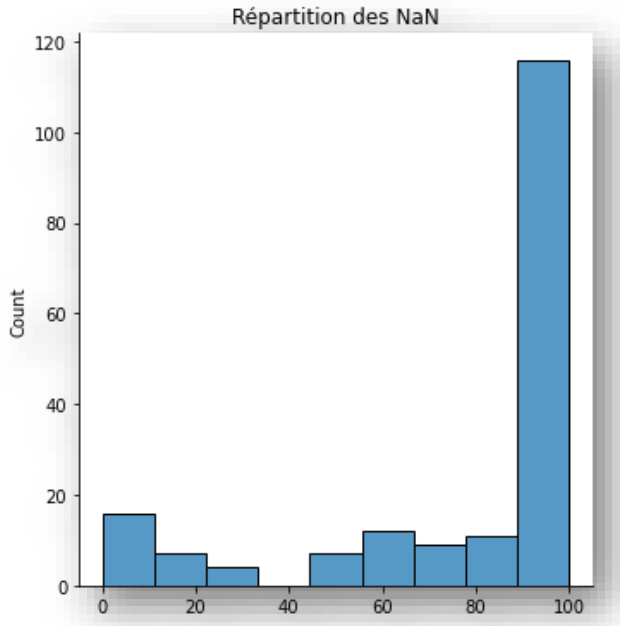
```
dtypes: float32(122), int32(2), object(58)
memory usage: 1.3+ GB
```

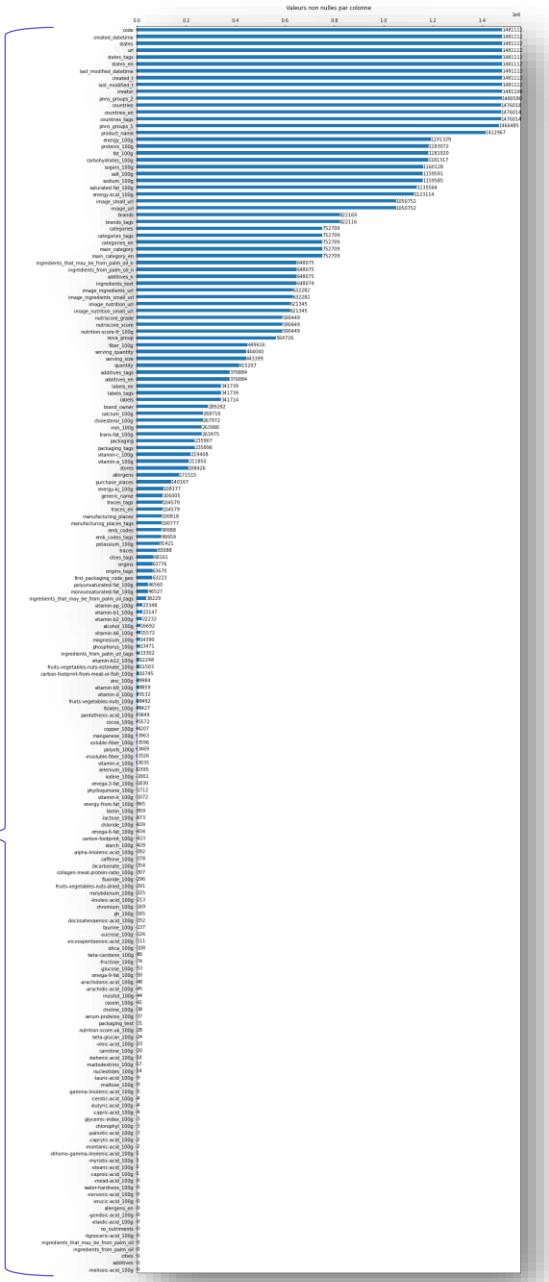
Nettoyage et traitement - NaN

Les colonnes contenant plus de 90% de données manquantes sont effacées.
Taille originale : 182 colonnes
Taille finale : 66 colonnes
Colonnes supprimées : 116

```
In [14]: df = delete_nan(data,0.1)

Colonnes supprimées : 116
['generic_name', 'packaging_text', 'origins', 'origins_tags', 'manufacturing_places', 'manufacturing_
places_tags', 'emb_codes', 'emb_codes_tags', 'first_packaging_code_geo', 'cities', 'cities_tags', 'pu
rchase_places', 'allergens_en', 'traces', 'traces_tags', 'traces_en', 'no_nutriments', 'additives',
'ingredients_from_palm_oil', 'ingredients_from_palm_oil_tags', 'ingredients_that_may_be_from_palm_oi
l', 'ingredients_that_may_be_from_palm_oil_tags', 'energy-kj_100g', 'energy-from-fat_100g', '-butyric
-acid_100g', '-caproic-acid_100g', '-caprylic-acid_100g', '-capric-acid_100g', '-lauric-acid_100g',
'-myristic-acid_100g', '-palmitic-acid_100g', '-stearic-acid_100g', '-arachidic-acid_100g', '-behenic
-acid_100g', '-lignoceric-acid_100g', '-cerotic-acid_100g', '-montanic-acid_100g', '-melissic-acid_10
0g', 'monounsaturated-fat_100g', 'polyunsaturated-fat_100g', 'omega-3-fat_100g', '-alpha-linolenic-ac
id_100g', '-eicosapentaenoic-acid_100g', '-docosahexaenoic-acid_100g', 'omega-6-fat_100g', '-linoleic
-acid_100g', '-arachidonic-acid_100g', '-gamma-linolenic-acid_100g', '-dihomo-gamma-linolenic-acid_10
0g', 'omega-9-fat_100g', '-oleic-acid_100g', '-elaidic-acid_100g', '-gondoic-acid_100g', '-mead-acid_
100g', '-erucic-acid_100g', '-nervonic-acid_100g', '-sucrose_100g', '-glucose_100g', '-fructose_100
g', '-lactose_100g', '-maltose_100g', '-maltodextrins_100g', 'starch_100g', 'polyols_100g', '-soluble
-fiber_100g', '-insoluble-fiber_100g', 'casein_100g', 'serum-proteins_100g', 'nucleotides_100g', 'alc
ohol_100g', 'beta-carotene_100g', 'vitamin-d_100g', 'vitamin-e_100g', 'vitamin-k_100g', 'vitamin-b1_1
00g', 'vitamin-b2_100g', 'vitamin-pp_100g', 'vitamin-b6_100g', 'vitamin-b9_100g', 'folates_100g', 'vi
tamin-b12_100g', 'biotin_100g', 'pantothenic-acid_100g', 'silica_100g', 'bicarbonate_100g', 'potassiu
m_100g', 'chloride_100g', 'phosphorus_100g', 'magnesium_100g', 'zinc_100g', 'copper_100g', 'manganese
_100g', 'fluoride_100g', 'selenium_100g', 'chromium_100g', 'molybdenum_100g', 'iodine_100g', 'caffein
e_100g', 'taurine_100g', 'ph_100g', 'fruits-vegetables-nuts_100g', 'fruits-vegetables-nuts-dried_100
g', 'fruits-vegetables-nuts-estimate_100g', 'collagen-meat-protein-ratio_100g', 'cocoa_100g', 'chloro
phyl_100g', 'carbon-footprint_100g', 'carbon-footprint-from-meat-or-fish_100g', 'nutrition-score-uk_1
00g', 'glycemic-index_100g', 'water-hardness_100g', 'choline_100g', 'phylloquinone_100g', 'beta-gluca
n_100g', 'inositol_100g', 'carnitine_100g']
```





Nettoyage et traitement – Time & Doubles

Unification des *timestamps* sur le format iso8601 yyyy-mm-ddThh:mn:ssZ

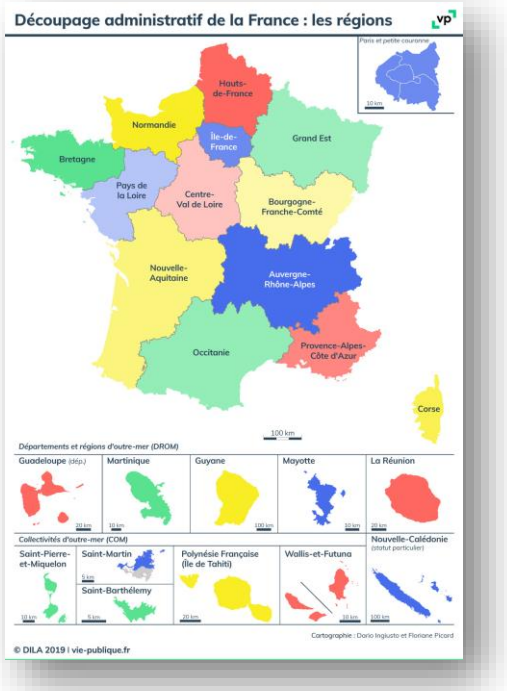
In [17]:
```
from datetime import datetime
import time
```

In [18]:
```
def unify_datetime(df):
    cols = df.columns
    colsViewed = []
    for col in cols:
        if col[-2:] == '_t': #_t dans la colonne ?
            newCol = col[:-2] #Le nom du colonne sans _t
            df[newCol] = pd.to_datetime(df[col],unit='s')
            df = df.drop(col,axis=1)
    return df
```

Effacer les doublons, en gardant les valeurs dans la colonne « code » les plus remplies par rapport aux autres colonnes. 592 lignes effacées.

In [22]:
```
def onlyOne(df):
    temp = df.copy()
    doublons = temp[temp.groupby('code')['code'].transform('size')>1].sort_values("code")
    codes = pd.unique(doublons['code'])
    indexs = doublons.index
    plusRemplie = []
    for code in codes:
        remplies = doublons[doublons['code']==code].count(axis=1)
        plusRemplie.append(remplies.idxmax) #Return index of first occurrence of maximum c
    moinsRemplie = doublons[~doublons.index.isin(plusRemplie)].index
    temp.drop(moinsRemplie,inplace=True)
    return temp
```

Nettoyage et traitement – Country & Tags



Comme l'app est destinée uniquement au marché français, nous allons garder uniquement les informations provenant de la France et ses territoires d'outre-mer.

Lignes avant le filtrage : 1480520
Lignes après le filtrage : 652914
Lignes effacées : 827606

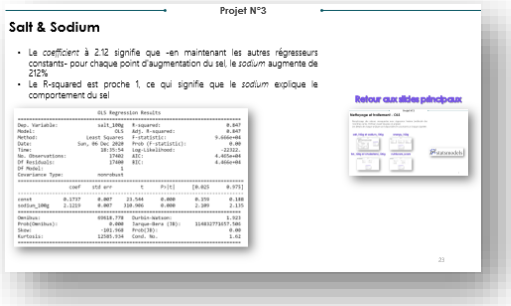
De l'analyse de la colonne « states_tags » (qui contient les données d'état de chaque ligne) il ressort que le tag **empty** et les tags contenant le string **to be completed** ne contiennent pas d'informations intéressantes. Donc, les dites lignes seront supprimées.

Lignes avant le filtrage des tags : 652914
Lignes après le filtrage des tags : 18697
Lignes effacées : 634217

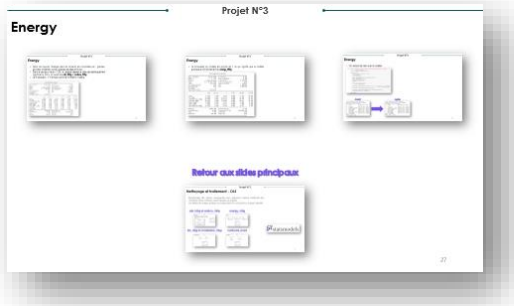
Nettoyage et traitement - OLS

Remplissage des valeurs manquantes avec régression linéaire (méthode des moindres carrés, Ordinary Least Squares en anglais).
Les détails de chaque analyse sont disponibles en zoomant sur chaque vignette.

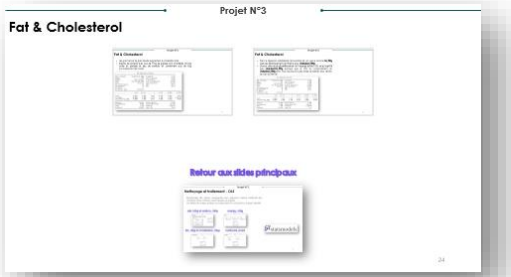
salt_100g et sodium_100g



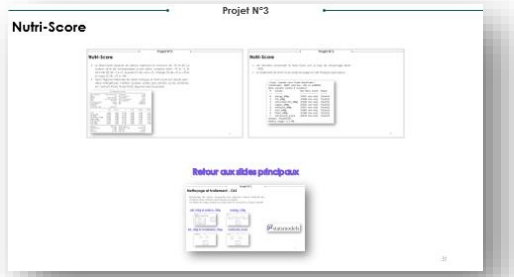
energy_100g



fat_100g et cholesterol_100g



nutriscore_score



statsmodels

Nettoyage et traitement – Quantity, Ingredients & Outliers

Unifications des valeurs et des unités sur la colonne « quantity ». Les unités sont unifiées sur grammes (g) et millilitres (ml) selon le cas avec 3000 comme valeur maximale.

```
for val, unit in zip(numFiltered, unitFiltered):
    if unit == 'kg':
        cleanUnits.append('g')
        cleanVals.append(val*1000)
    elif unit == 'g':
        cleanUnits.append('g')
        cleanVals.append(val)
    elif unit == 'mg':
        cleanUnits.append('g')
        cleanVals.append(val/1000)
    elif unit == 'l':
        cleanUnits.append('ml')
        cleanVals.append(val*1000)
    elif unit == 'cl':
        cleanUnits.append('ml')
        cleanVals.append(val*10)
    elif unit == 'ml':
        cleanUnits.append('ml')
        cleanVals.append(val)
    else :
        cleanUnits.append(None)
        cleanVals.append(None)
```

Unification des formats des *strings* contenant des ingrédients vers utf-8.

Correction des valeurs aberrantes de chaque colonne vers la valeur moyenne de chacune. Notamment dans les colonnes :

- saturated-fat_100g
- serving_quantity
- Energy_100g

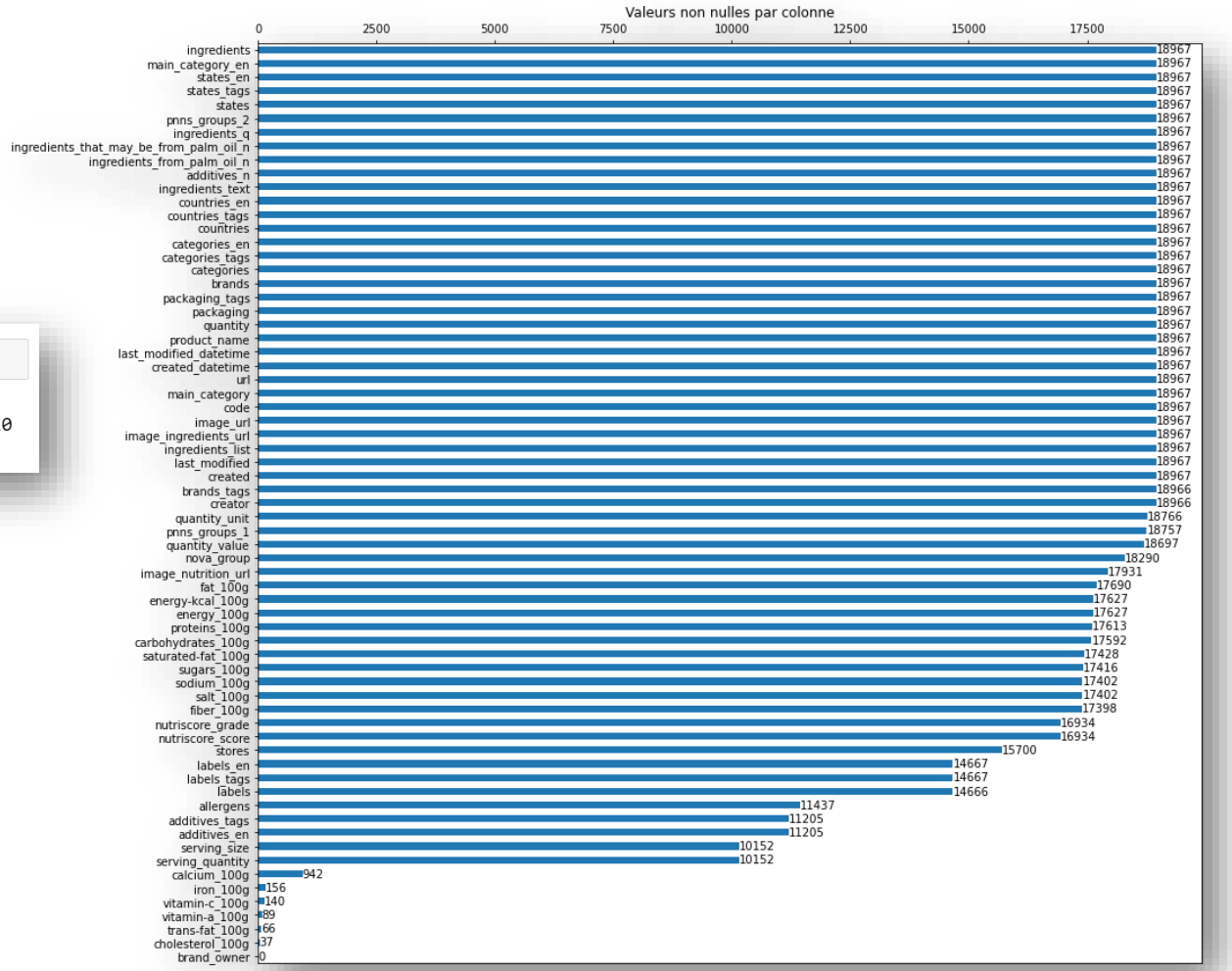
```
def overXvalue_toMean(df,cols,x=100):
    for col in cols:
        median = df.loc[df[col] < x, col].median()
        df[col] = np.where(df[col] >= x, median, df[col])
    return df
```

Nettoyage et traitement - NaN

Les colonnes contenant plus de 60% de données manquantes sont effacées.
Taille originale : 182 colonnes
Taille finale : 60 colonnes
Colonnes supprimées en totale: 122

```
In [119]: df = delete_nan(df,0.4)

          Colonnes supprimées : 7
          ['brand_owner', 'trans-fat_100g', 'cholesterol_100g', 'vitamin-a_100g', 'vitamin-c_100g', 'calcium_10
          0g', 'iron_100g']
```



Nettoyage et traitement – Memory size

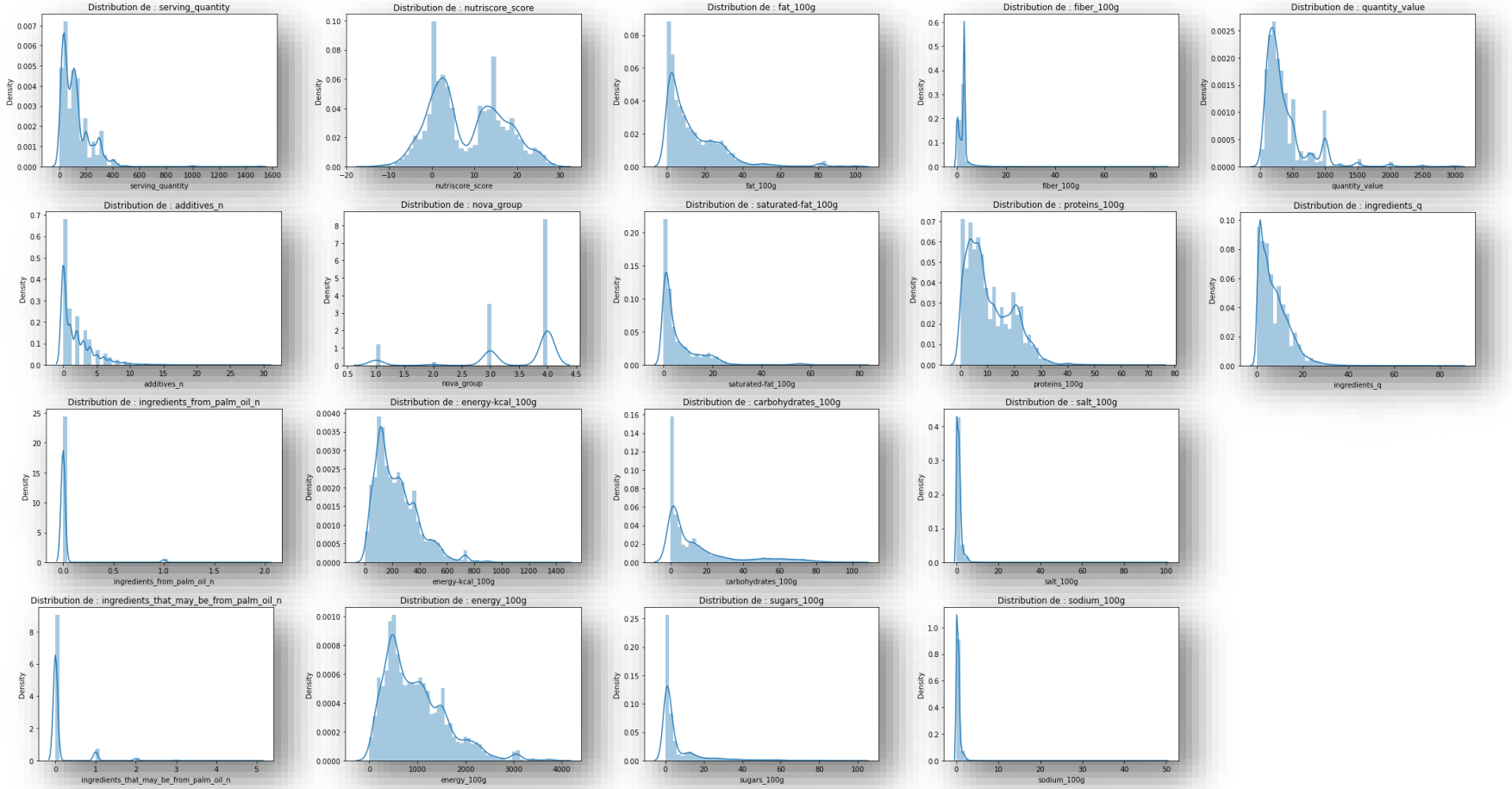
Après toutes les actions de nettoyage, le rapport entre la taille originale du csv et la taille finale est le suivant :

Out[120]:

	CSV	Shape	% NaN	Memory Usage in MB
0	initial	(1481112, 182)	78.86	2056.60
1	final	(18967, 60)	6.84	8.29

Analyse exploratoire – Analyse univariée

Des distributions des variables on peut observer que certaines variables sont discrètes. Aussi on peut observer que certaines variables ont une distribution très similaires.



Variables discrètes

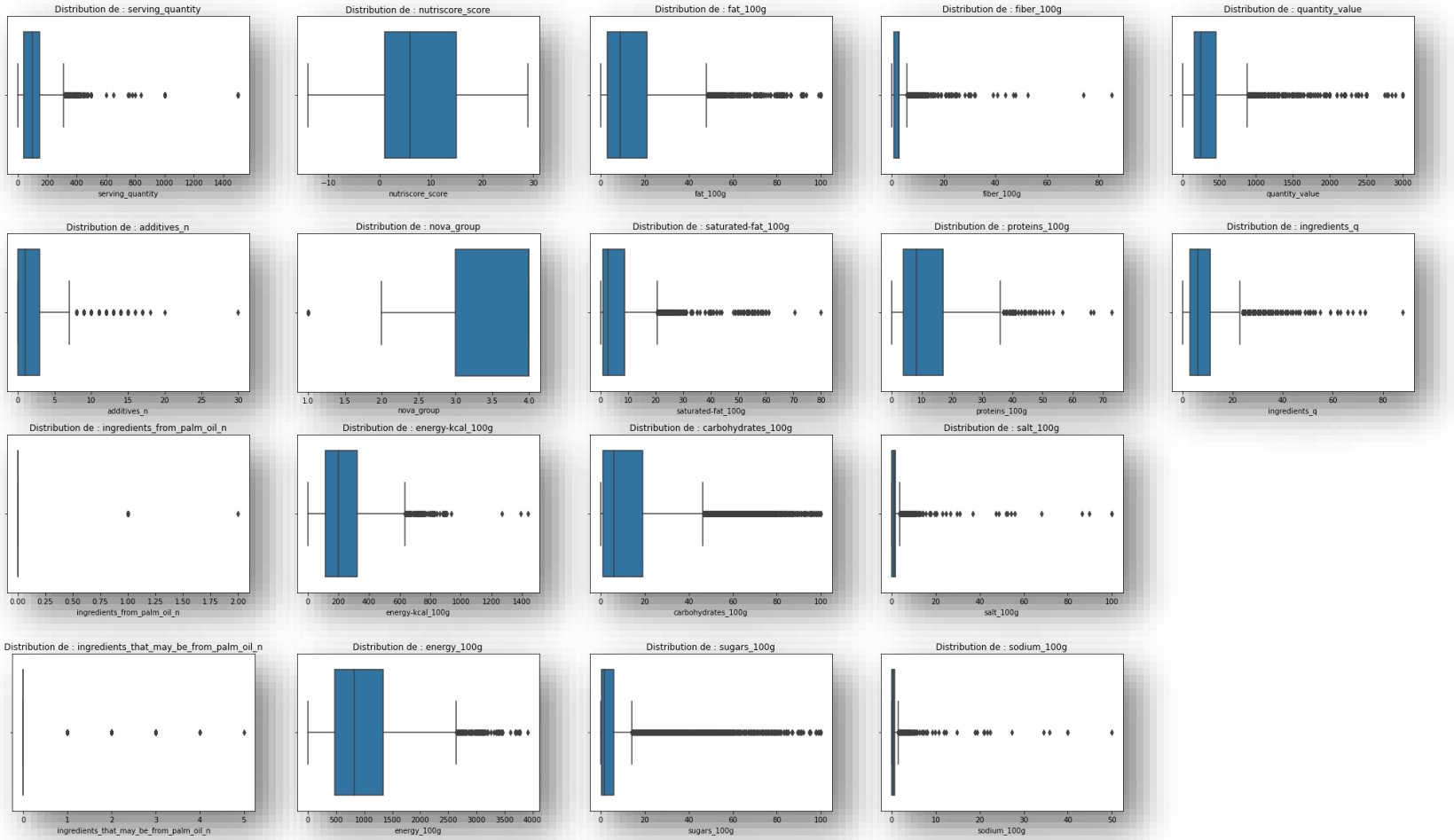
- additives_n
- ingredients_from_palm_oil_n
- ingredients_that_may_be_from_palm_oil_n
- nutriscore_score
- nova_group
- ingredients_q

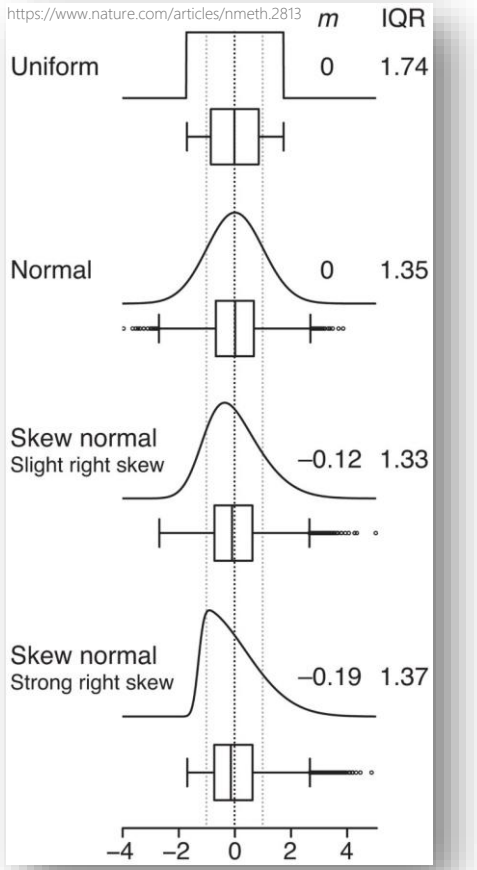
Distributions similaires

- energy-kcal_100g et energy_100g
- salt_100g et sodium_100g
- carbohydrates_100g et sugars_100g
- fat_100g et saturated-fat_100g

Analyse exploratoire – Analyse univariée

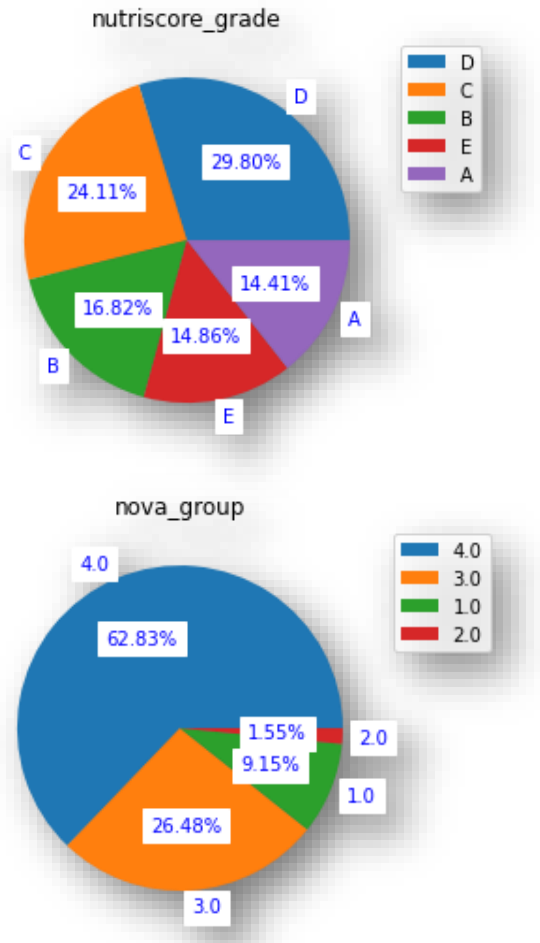
Test de normalité avec *Boxplot* : les variables ne semblent pas suivre la loi normale



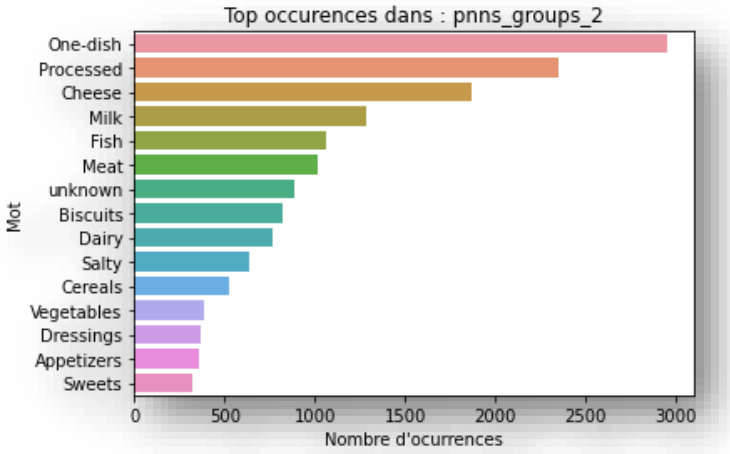


Analyse exploratoire – Analyse univariée

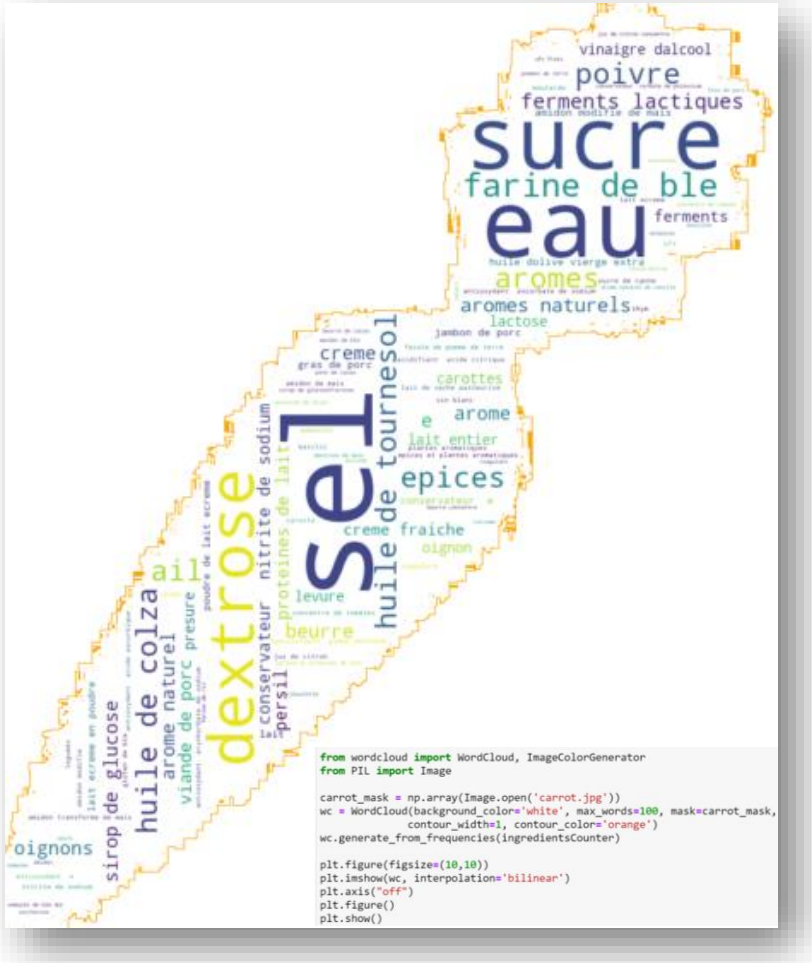
Répartition des Nutri-Score et Nova group



Occurrences des PNNS groups

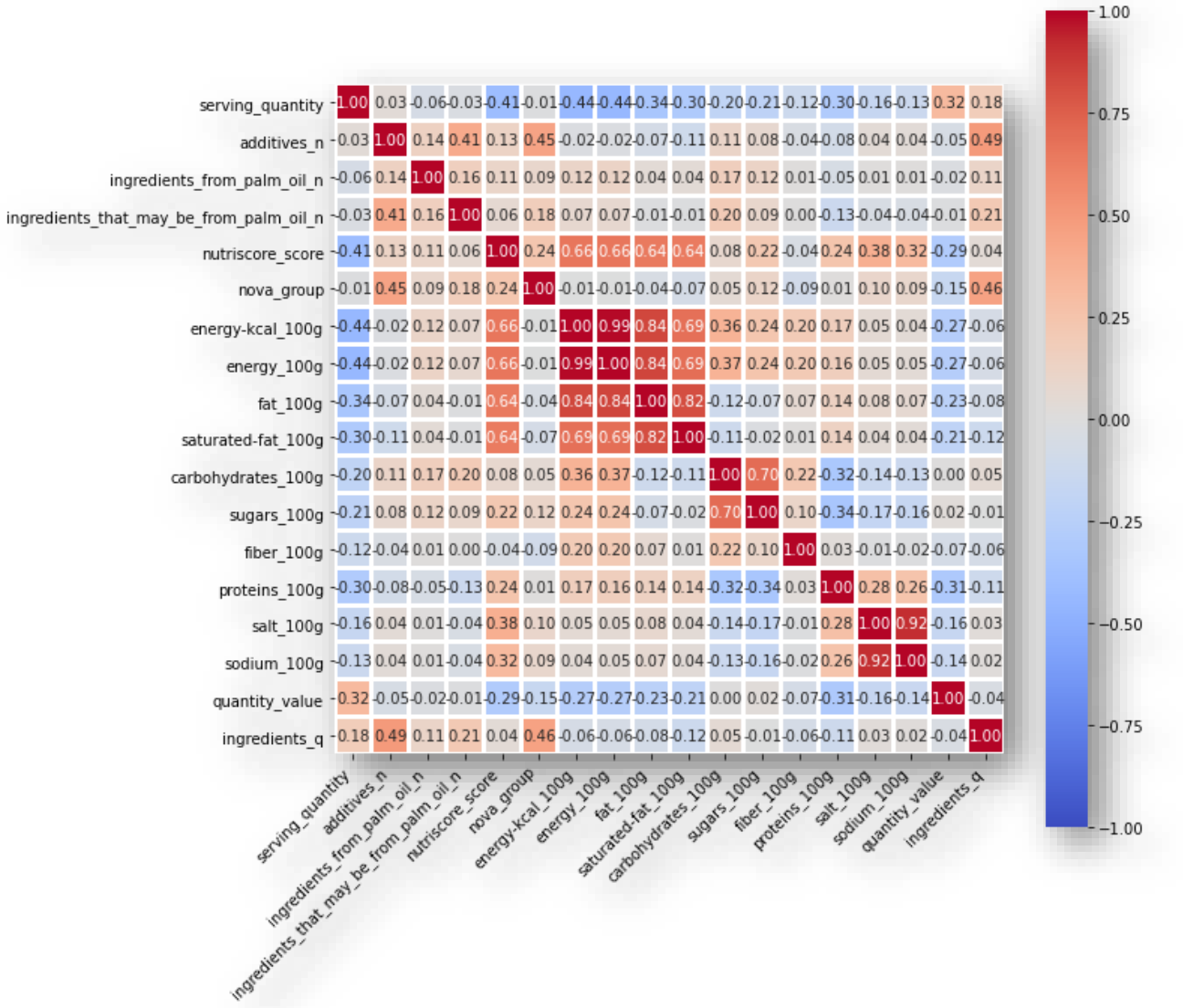


Wordcloud en forme de carotte avec les 100 plus grandes occurrences



Analyse exploratoire – Analyse bivariée

Pearson's r : corrélation linéaire entre deux variables.



Le coefficient R de Pearson peut être interprété (en valeurs absolues) comme :

Rang	Correlation
1~0.9	corrélation très élevée
0.9~0.7	corrélation élevée
0.7~0.5	corrélation modérée
0.5~0.3	faible corrélation
0.3~0	corrélation négligeable

	serie1	serie2	abs_corr	type
35	nutriscore_score	energy_100g	0.66	moderate
36	nutriscore_score	energy-kcal_100g	0.66	moderate
39	nutriscore_score	fat_100g	0.64	moderate
40	nutriscore_score	saturated-fat_100g	0.64	moderate
53	nutriscore_score	serving_quantity	0.41	low
56	nutriscore_score	salt_100g	0.38	low
69	nutriscore_score	sodium_100g	0.32	low

Analyse exploratoire – Analyse bivariée

Chi-squared : vérifie s'il existe une relation significative entre deux variables catégorielles

- H0 : Il n'y a pas de relation entre la variable 1 et la variable 2
- H1 : il existe une relation entre la variable 1 et la variable 2

Si le p-value est significative, on peut rejeter l'hypothèse nulle et affirmer que les résultats soutiennent l'hypothèse alternative.

Le test chi-squared conclut au rejet de l'Hypothèse 0 pour toutes les variables analysées. Donc, on peut dire qu'il y a un lien entre le nutriscore_score et la valeur des variables analysées.

```python
def chi2_table(series1, series2, alpha=0.05):

    if type(series1) != list:
        crosstab = pd.crosstab(series1, series2)
        chi2q, p, dof, expected = stats.chi2_contingency(crosstab)

    elif type(series1) == list and type(series2) == list:
        for entry2 in series2:
            for entry1 in series1:
                crosstab = pd.crosstab(entry1, entry2)
                chi2q, p, dof, expected = stats.chi2_contingency(crosstab)

    elif type(series1) == list:
        for entry in series1:
            crosstab = pd.crosstab(entry, series2)
            chi2q, p, dof, expected = stats.chi2_contingency(crosstab)

    prob = 1.0-alpha
    critical = chi2.ppf(prob, dof)
    print('probability=%.3f,\ncritical=%.3f,\nchi2=%.3f\n' % (prob, critical, chi2q))
    if abs(chi2q) >= critical:
        print('Dependent (reject H0)')
    else:
        print('Independent (fail to reject H0)')
    # interpret p-value
    print('\nsignificance=%.3f,\np=%.3f\n' % (alpha, p))
    if p <= alpha:
        print('Dependent (reject H0)')
    else:
        print('Independent (fail to reject H0)')

    if p <= alpha:
        return False
    else:
        return True
```

```python
x = data['nutriscore_score']
for col in data.select_dtypes(include = ['int64','float64']).columns:
    print("Chi-squared : {} / {}".format('nutriscore_score',col))
    if data[col].nunique() > 20 :
        y = data[col].astype('category')
    else:
        y = data[col].astype('category')
    chi2_table(x, y)
    print('_'*90,'\n')
```

```
Chi-squared : nutriscore_score / serving_quantity
probability=0.950,
critical=16763.609,
chi2=26577.416

Dependent (reject H0)

significance=0.050,
p=0.000

Dependent (reject H0)
```

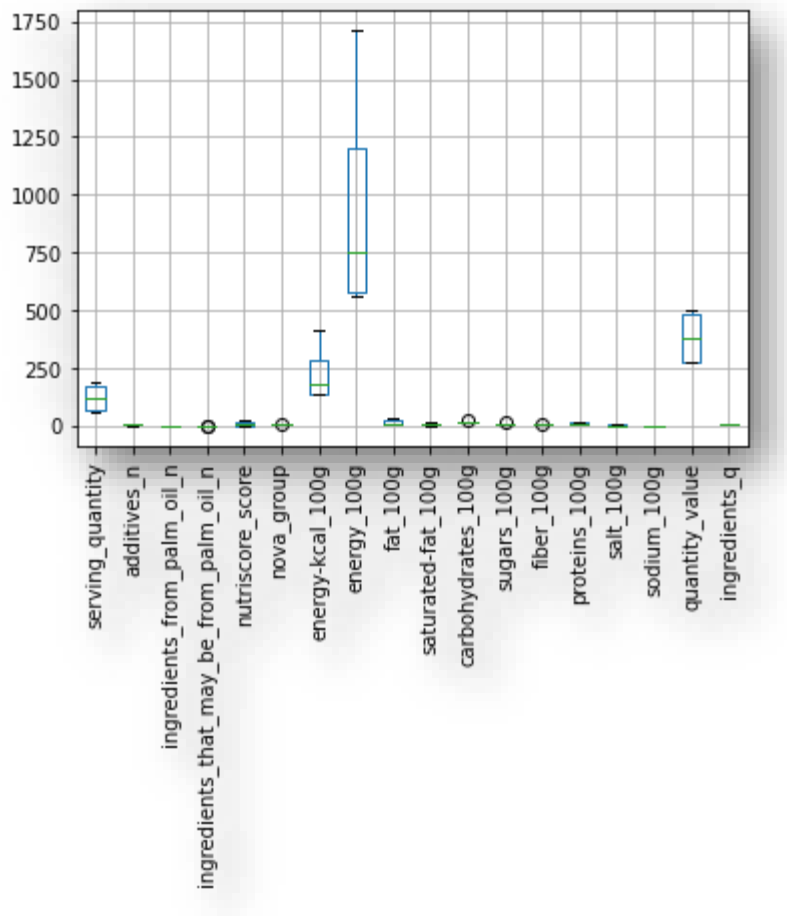
Analyse exploratoire – Analyse bivariée

ANOVA : ensemble des modèles statistiques utilisés pour vérifier si les moyennes des groupes proviennent d'une même population.

- H0 : m1 = m2 = … = mk = m
- H1 : Il existe un (i,j) telle que mi =! mj

Comme nous le soulignerons précédemment, les variables semblent ne pas suivre une distribution normale.

Ce phénomène est dû à la variation Inter-Groupe (Between-Group) : une quantification de la variation expliquée par notre variable **nutriscore_grade**. Cependant, il y a aussi une partie de la variation qui ne peut pas expliquer la variable **nutriscore_grade**, nous aurions besoin de plus de variables pour l'expliquer. Puisque nous n'avons pas ces nouvelles variables, la variation reste inexpliquée et est appelée la variation Intra-groupe (Within-Group).



Analyse exploratoire – Analyse bivariée

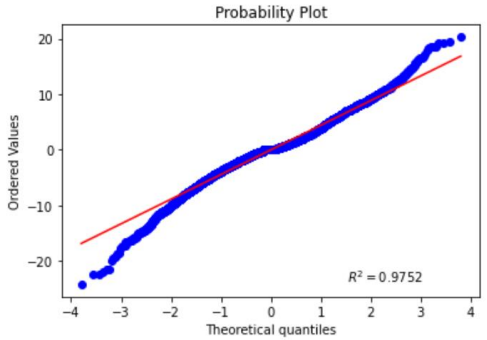
ANOVA : Hard fail

Après avoir appliqué les tests ANOVA Table, QQ PLOT, Shapiro-Wilk, K-squared et Anderson-Darling, on peut conclure un hard fail sur la conclusion de normalité. Les données ne suivent pas une distribution normale, donc il n'y a aucune raison de faire une analyse de variance.

```
keys = []
tables = []
for var in num_data.columns:
    # Ordinary Least Squares (OLS) model
    model = ols('{} ~ C({})'.format('nutriscore_score',var), data=num_data).fit()
    aov_table = sm.stats.anova_lm(model, typ=2)
    keys.append(var)
    tables.append(aov_table)
    print(anova_table(aov_table))
    #print(aov_table)
    print('_'*90,'\n')

aov_data = pd.concat(tables, keys=keys, axis=0)
```

```
QQ Plot and Shapiro-Wilk test: energy_100g
shapiro_test.statistic : 0.975
shapiro_test.pvalue : 0.000
H0 hypothesis rejected. Data is not drawn from a normal distribution.
```




```
from scipy.stats import shapiro
for col in num_data.columns:
    stat, p = shapiro(num_data[col])
    print(col)
    print('Statistics=%.3f, p=%.3f' % (stat, p))
    alpha = 0.05
    if p > alpha:
        print('Sample looks Gaussian (fail to reject H0)')
    else:
        print('Sample does not look Gaussian (reject H0)')
    print('_'*90,'\n')
```

```
serving_quantity
Statistics=0.781, p=0.000
Sample does not look Gaussian (reject H0)
__________________________________________________________________________________________

additives_n
Statistics=0.814, p=0.000
Sample does not look Gaussian (reject H0)
__________________________________________________________________________________________

ingredients_from_palm_oil_n
Statistics=0.150, p=0.000
Sample does not look Gaussian (reject H0)
```

```
for col in num_data.columns:
    print("K-squared :".format(col))
    k2, p = stats.normaltest(num_data[col], axis=0, nan_policy = 'omit')
    alpha = 0.05
    print("p = {:g}".format(p))
    if p < alpha:  # H0 : la colonne a une distribution normale
        print("Reject H0: {} doesn't have a normal distribution ".format(col))
    else:
        print("Reject H1: {} has a normal distribution".format(col))
    print('_'*90,'\n')
```

```
K-squared :
p = 0
Reject H0: serving_quantity doesn't have a normal distribution
__________________________________________________________________________________________

K-squared :
p = 0
Reject H0: additives_n doesn't have a normal distribution
__________________________________________________________________________________________

K-squared :
p = 0
Reject H0: ingredients_from_palm_oil_n doesn't have a normal distribution
```

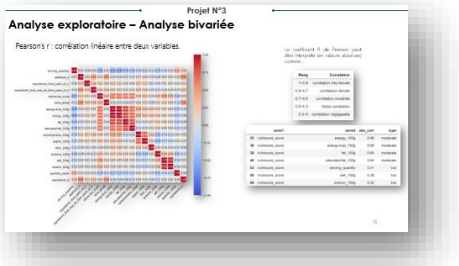
```
from scipy.stats import anderson
for col in num_data.columns:
    print(col)
    result = anderson(num_data[col])
    print('Statistic: %.3f' % result.statistic)
    p = 0
    for i in range(len(result.critical_values)):
        sl, cv = result.significance_level[i], result.critical_values[i]
        if result.statistic < result.critical_values[i]:
            print('%.3f: %.3f, data looks normal (fail to reject H0)' % (sl, cv))
        else:
            print('%.3f: %.3f, data does not look normal (reject H0)' % (sl, cv))
    print('_'*90,'\n')
```

```
serving_quantity
Statistic: 400.628
15.000: 0.576, data does not look normal (reject H0)
10.000: 0.656, data does not look normal (reject H0)
5.000: 0.787, data does not look normal (reject H0)
2.500: 0.918, data does not look normal (reject H0)
1.000: 1.092, data does not look normal (reject H0)
```

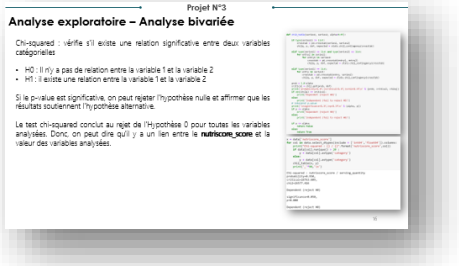
Faits pertinents pour l'app

3 observations :

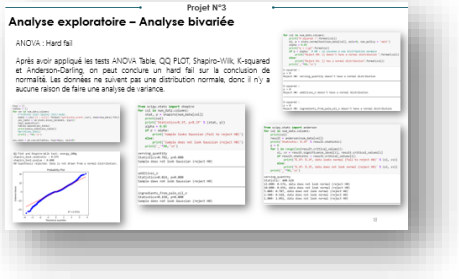
- Corrélation forte de certaines variables avec le *Nutri-Score* (Pearson's r)



- Il y a un lien entre le *Nutri-Score* et les variables analysées (Chi-squared)



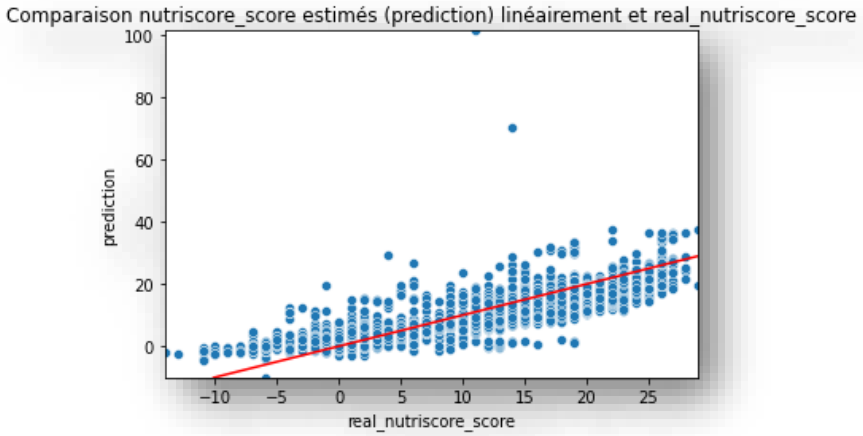
- Les variables ne suivent pas une distribution normale (ANOVA Hard Fail)

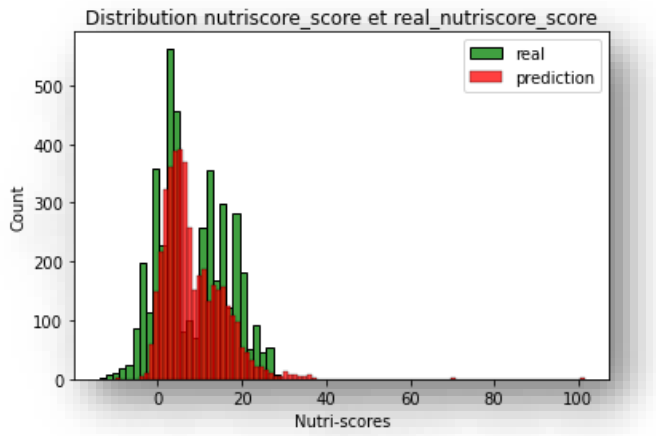


Faits pertinents pour l'app

Estimation linéaire du *Nutri-Score* :

- 7 variables utilisées
- R-squared du jeu testé ~0.7



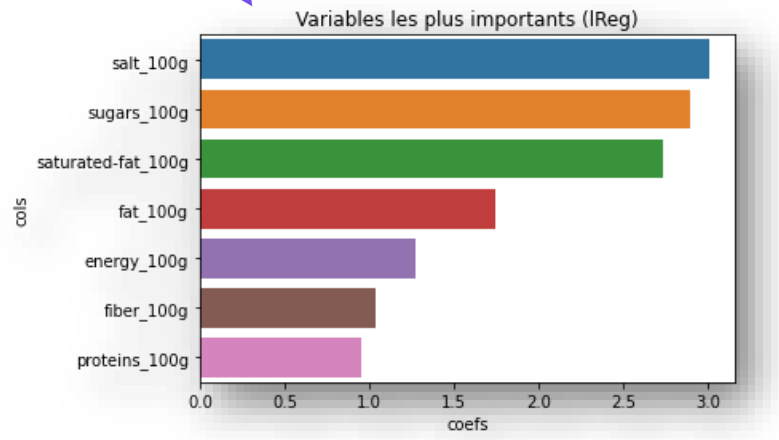


OLS Regression Results

Dep. Variable:	nutriscore_score	R-squared (uncentered):	0.835
Model:	OLS	Adj. R-squared (uncentered):	0.835
Method:	Least Squares	F-statistic:	1.224e+04
Date:	Sun, 06 Dec 2020	Prob (F-statistic):	0.00
Time:	17:16:51	Log-Likelihood:	-50557.
No. Observations:	16926	AIC:	1.011e+05
Df Residuals:	16919	BIC:	1.012e+05
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
energy_100g	0.0014	0.000	9.670	0.000	0.001	0.002
fat_100g	0.1286	0.007	18.827	0.000	0.115	0.142
saturated-fat_100g	0.3203	0.008	41.575	0.000	0.305	0.335
sugars_100g	0.2300	0.004	54.969	0.000	0.222	0.238
salt_100g	2.0065	0.030	66.876	0.000	1.948	2.065
fiber_100g	-0.5795	0.017	-33.249	0.000	-0.614	-0.545
proteins_100g	0.0824	0.005	16.041	0.000	0.072	0.092

Omnibus:	7850.871	Durbin-Watson:	1.391
Prob(Omnibus):	0.000	Jarque-Bera (JB):	503643.993
Skew:	-1.409	Prob(JB):	0.00
Kurtosis:	29.574	Cond. No.	949.



Faits pertinents pour l'app

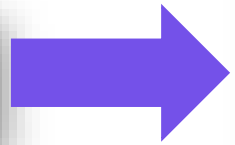
Même si le modèle est relativement pertinent (R-squared ~0.7), l'application est faisable et l'analyse statistique accompagne ce propos. Par contre, compte tenu des connaissances acquises au moment de la réalisation de ce Projet, il n'est pas possible de déterminer les causes de l'écart entre la prédiction du *Nutri-Score* et sa valeur réelle.

Pour le moment, les valeurs à utiliser seront approximées comme suit :

- Pour chaque groupe *PNNS*, son grade de *Nutri-Score* sera calculé en prenant la moyenne des scores en effectuant un filtre sur la colonne des scores avec les groupes *PNNS* comme condition. Les résultats obtenus sont très similaires à ceux représentés dans le heatmap « PNNS Groupes 2 et Nutri-score »

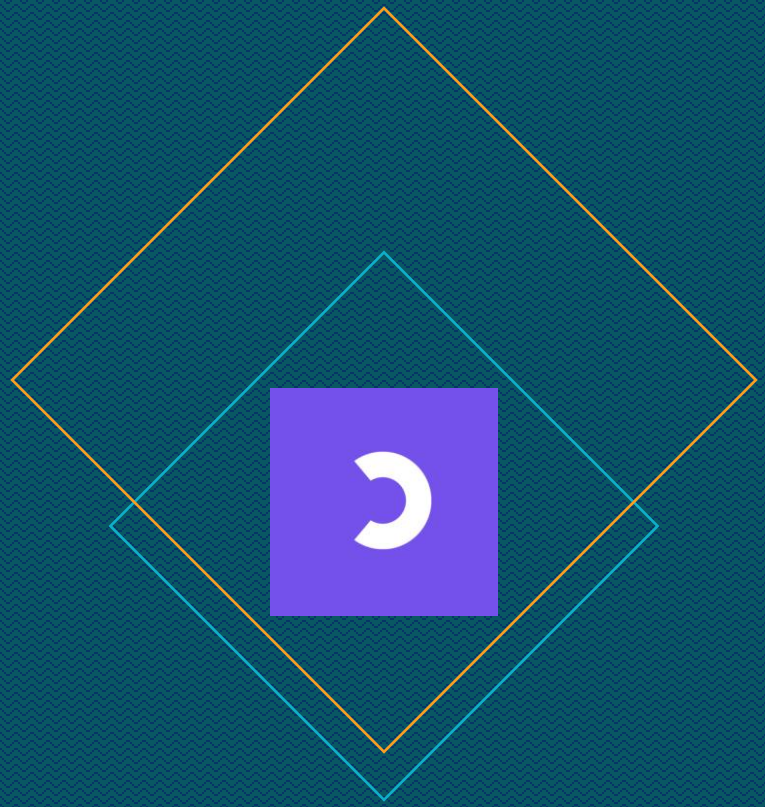
- Les nutriments correspondant à chaque groupe *PNNS* sont calculés avec les mêmes critères que le *Nutri-Score* détaillé ci-dessus.

```
df_input = pd.DataFrame(columns=['indicator','quantity_g'])
df_input['indicator'] = ['sweets','legumes','bread','pastries']
df_input['quantity_g'] = ['50','200','200','300']
```



```
resultStatus = dailyCheck(dailyStatus)
resultStatus
```

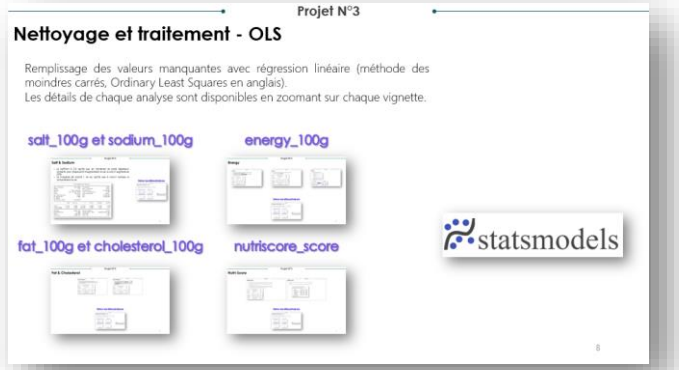
	energy-kcal_100g	fat_100g	saturated-fat_100g	carbohydrates_100g	sugars_100g	proteins_100g	salt_100g
0	2424.65	69.28	29.55	360.22	105.6	73.85	6.51
1	NOPE	OK	NOPE	NOPE	NOPE	NOPE	NOPE



Salt & Sodium

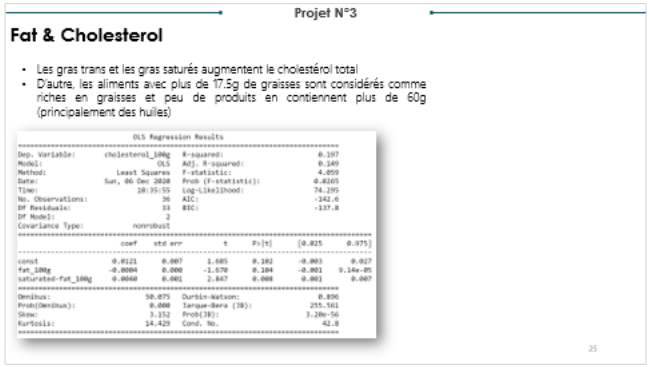
- Le *coefficient* à 2.12 signifie que -en maintenant les autres régresseurs constants- pour chaque point d'augmentation du sel, le *sodium* augmente de 212%
- Le R-squared est proche 1, ce qui signifie que le *sodium* explique le comportement du sel

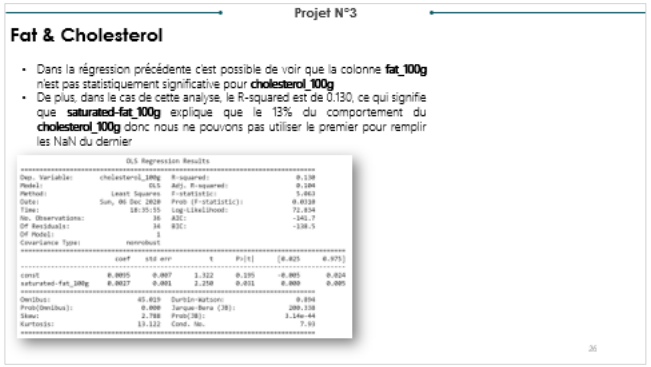
Retour aux slides principaux



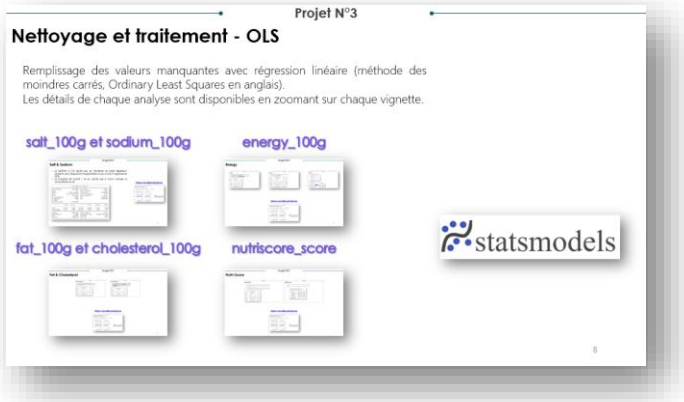

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              salt_100g   R-squared:                       0.847
Model:                            OLS   Adj. R-squared:                  0.847
Method:                 Least Squares   F-statistic:                 9.666e+04
Date:                Sun, 06 Dec 2020   Prob (F-statistic):               0.00
Time:                        18:35:54   Log-Likelihood:                -22322.
No. Observations:               17402   AIC:                         4.465e+04
Df Residuals:                   17400   BIC:                         4.466e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1737      0.007     23.544      0.000       0.159       0.188
sodium_100g    2.1219      0.007    310.906      0.000       2.109       2.135
==============================================================================
Omnibus:                    69618.778   Durbin-Watson:                   1.923
Prob(Omnibus):                  0.000   Jarque-Bera (JB):     114832771657.506
Skew:                        -101.968   Prob(JB):                         0.00
Kurtosis:                   12585.934   Cond. No.                         1.62
==============================================================================
```

Fat & Cholesterol





Retour aux slides principaux



Fat & Cholesterol

- Les gras trans et les gras saturés augmentent le cholestérol total
- D'autre, les aliments avec plus de 17.5g de graisses sont considérés comme riches en graisses et peu de produits en contiennent plus de 60g (principalement des huiles)

OLS Regression Results

Dep. Variable:	cholesterol_100g	R-squared:	0.197
Model:	OLS	Adj. R-squared:	0.149
Method:	Least Squares	F-statistic:	4.059
Date:	Sun, 06 Dec 2020	Prob (F-statistic):	0.0265
Time:	18:35:55	Log-Likelihood:	74.295
No. Observations:	36	AIC:	-142.6
Df Residuals:	33	BIC:	-137.8
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0121	0.007	1.685	0.102	-0.003	0.027
fat_100g	-0.0004	0.000	-1.670	0.104	-0.001	9.14e-05
saturated-fat_100g	0.0040	0.001	2.847	0.008	0.001	0.007

Omnibus:	50.075	Durbin-Watson:	0.896
Prob(Omnibus):	0.000	Jarque-Bera (JB):	255.561
Skew:	3.152	Prob(JB):	3.20e-56
Kurtosis:	14.429	Cond. No.	42.8

Fat & Cholesterol

- Dans la régression précédente c'est possible de voir que la colonne **fat_100g** n'est pas statistiquement significative pour **cholesterol_100g**
- De plus, dans le cas de cette analyse, le R-squared est de 0.130, ce qui signifie que **saturated-fat_100g** explique que le 13% du comportement du **cholesterol_100g** donc nous ne pouvons pas utiliser le premier pour remplir les NaN du dernier

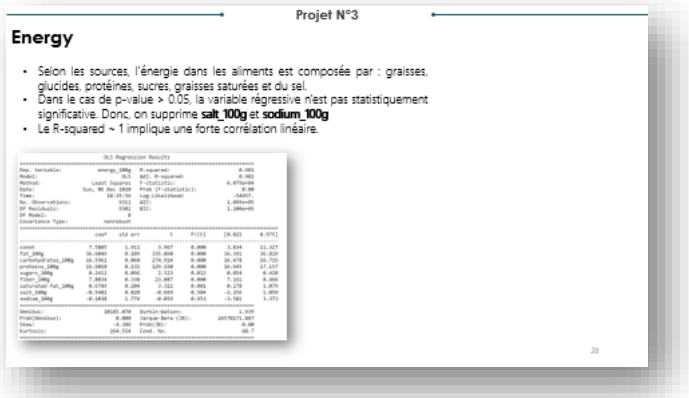
OLS Regression Results

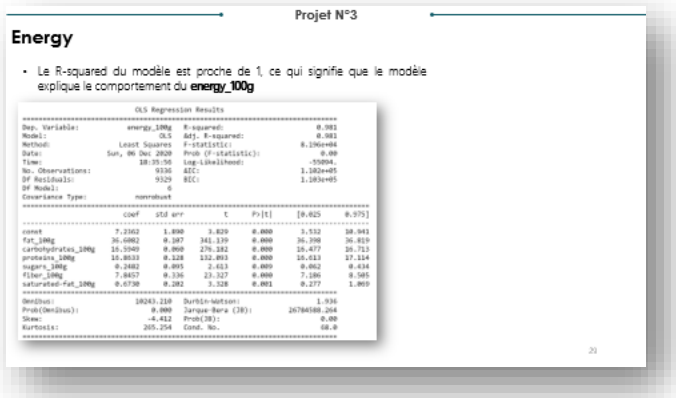
Dep. Variable:	cholesterol_100g	R-squared:	0.130
Model:	OLS	Adj. R-squared:	0.104
Method:	Least Squares	F-statistic:	5.063
Date:	Sun, 06 Dec 2020	Prob (F-statistic):	0.0310
Time:	18:35:55	Log-Likelihood:	72.834
No. Observations:	36	AIC:	-141.7
Df Residuals:	34	BIC:	-138.5
Df Model:	1		
Covariance Type:	nonrobust		

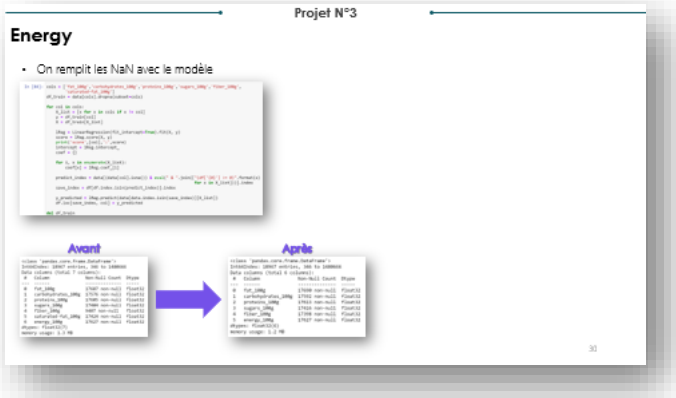
	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0095	0.007	1.322	0.195	-0.005	0.024
saturated-fat_100g	0.0027	0.001	2.250	0.031	0.000	0.005

Omnibus:	45.019	Durbin-Watson:	0.894
Prob(Omnibus):	0.000	Jarque-Bera (JB):	200.338
Skew:	2.788	Prob(JB):	3.14e-44
Kurtosis:	13.122	Cond. No.	7.93

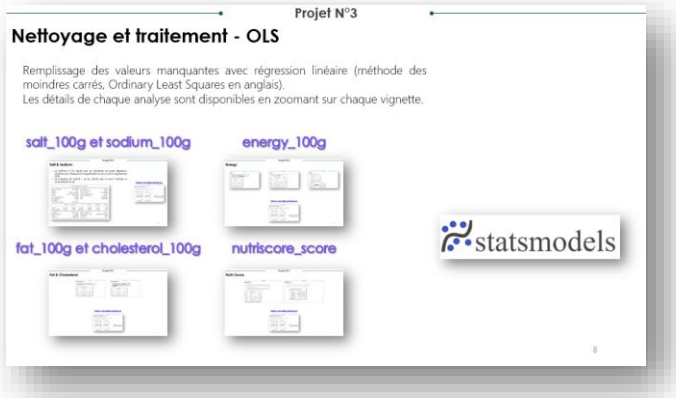
Energy







Retour aux slides principaux



Energy

- Selon les sources, l'énergie dans les aliments est composée par : graisses, glucides, protéines, sucres, graisses saturées et du sel.
- Dans le cas de p-value > 0.05, la variable régressive n'est pas statistiquement significative. Donc, on supprime **salt_100g** et **sodium_100g**
- Le R-squared ~ 1 implique une forte corrélation linéaire.

OLS Regression Results

Dep. Variable:	energy_100g	R-squared:	0.981
Model:	OLS	Adj. R-squared:	0.981
Method:	Least Squares	F-statistic:	6.079e+04
Date:	Sun, 06 Dec 2020	Prob (F-statistic):	0.00
Time:	18:35:56	Log-Likelihood:	-54957.
No. Observations:	9311	AIC:	1.099e+05
Df Residuals:	9302	BIC:	1.100e+05
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	7.5805	1.911	3.967	0.000	3.834	11.327
fat_100g	36.6049	0.109	335.860	0.000	36.391	36.819
carbohydrates_100g	16.5962	0.060	274.914	0.000	16.478	16.715
proteins_100g	16.9010	0.131	129.190	0.000	16.645	17.157
sugars_100g	0.2411	0.096	2.523	0.012	0.054	0.428
fiber_100g	7.8034	0.338	23.087	0.000	7.141	8.466
saturated-fat_100g	0.6789	0.204	3.322	0.001	0.278	1.079
salt_100g	-0.5482	0.820	-0.669	0.504	-2.156	1.059
sodium_100g	-0.1038	1.774	-0.059	0.953	-3.581	3.373

Omnibus:	10185.070	Durbin-Watson:	1.939
Prob(Omnibus):	0.000	Jarque-Bera (JB):	26570271.887
Skew:	-4.386	Prob(JB):	0.00
Kurtosis:	264.554	Cond. No.	68.7

Energy

- Le R-squared du modèle est proche de 1, ce qui signifie que le modèle explique le comportement du **energy_100g**

OLS Regression Results

Dep. Variable:	energy_100g	R-squared:	0.981
Model:	OLS	Adj. R-squared:	0.981
Method:	Least Squares	F-statistic:	8.196e+04
Date:	Sun, 06 Dec 2020	Prob (F-statistic):	0.00
Time:	18:35:56	Log-Likelihood:	-55094.
No. Observations:	9336	AIC:	1.102e+05
Df Residuals:	9329	BIC:	1.103e+05
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	7.2362	1.890	3.829	0.000	3.532	10.941
fat_100g	36.6082	0.107	341.139	0.000	36.398	36.819
carbohydrates_100g	16.5949	0.060	276.182	0.000	16.477	16.713
proteins_100g	16.8633	0.128	132.093	0.000	16.613	17.114
sugars_100g	0.2482	0.095	2.613	0.009	0.062	0.434
fiber_100g	7.8457	0.336	23.327	0.000	7.186	8.505
saturated-fat_100g	0.6730	0.202	3.328	0.001	0.277	1.069

Omnibus:	10243.210	Durbin-Watson:	1.936
Prob(Omnibus):	0.000	Jarque-Bera (JB):	26784588.264
Skew:	-4.412	Prob(JB):	0.00
Kurtosis:	265.254	Cond. No.	68.0

Energy

- On remplit les NaN avec le modèle

```
In [84]: cols = ['fat_100g','carbohydrates_100g','proteins_100g','sugars_100g','fiber_100g',
                 'saturated-fat_100g']
         df_train = data[cols].dropna(subset=cols)

         for col in cols:
             X_list = [x for x in cols if x != col]
             y = df_train[col]
             X = df_train[X_list]

             lReg = LinearRegression(fit_intercept=True).fit(X, y)
             score = lReg.score(X, y)
             print('score',[col],':',score)
             intercept = lReg.intercept_
             coef = {}

             for i, x in enumerate(X_list):
                 coef[x] = lReg.coef_[i]

             predict_index = data[(data[col].isna()) & eval(" & ".join(["(df['{0}'] >= 0)".format(x)
                                                                        for x in X_list]))].index
             save_index = df[df.index.isin(predict_index)].index

             y_predicted = lReg.predict(data[data.index.isin(save_index)][X_list])
             df.loc[save_index, col] = y_predicted

         del df_train
```

Avant

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18967 entries, 346 to 1480666
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   fat_100g            17687 non-null  float32
 1   carbohydrates_100g  17576 non-null  float32
 2   proteins_100g       17605 non-null  float32
 3   sugars_100g         17404 non-null  float32
 4   fiber_100g          9407 non-null   float32
 5   saturated-fat_100g  17424 non-null  float32
 6   energy_100g         17627 non-null  float32
dtypes: float32(7)
memory usage: 1.3 MB
```

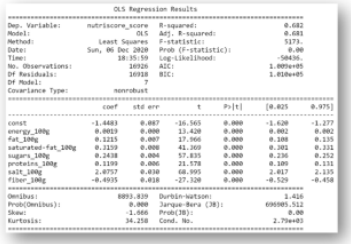
Après

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18967 entries, 346 to 1480666
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   fat_100g            17690 non-null  float32
 1   carbohydrates_100g  17592 non-null  float32
 2   proteins_100g       17613 non-null  float32
 3   sugars_100g         17416 non-null  float32
 4   fiber_100g          17398 non-null  float32
 5   energy_100g         17627 non-null  float32
dtypes: float32(6)
memory usage: 1.2 MB
```

Nutri-Score

Nutri-Score

- Le *Nutri-Score* respecte les valeurs maximum et minimum de -15 et 40. La couleur verte (A) correspondant à une valeur comprise entre –15 et –2, le vert clair (B) de –1 à +3, le jaune (C) de +4 à +11, l'orange (D) de +12 à +16 et le rouge (E) de +17 à +40.
- Selon l'Agence Nationale de Sante Publique, le *Nutri-Score* est calculé avec : valeur énergétique, matières graisses, acides gras saturés, sucres, protéines, sel / sodium, fibres, % des fruits, légumes dans le produit



OLS Regression Results

Dep. Variable:	nutriscore_score	R-squared:	0.682
Model:	OLS	Adj. R-squared:	0.681
Method:	Least Squares	F-statistic:	5173.
Date:	Sun, 06 Dec 2020	Prob (F-statistic):	0.00
Time:	18:35:59	Log-Likelihood:	-50436.
No. Observations:	16926	AIC:	1.009e+05
Df Residuals:	16918	BIC:	1.010e+05
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.4483	0.007	-16.565	0.000	-1.620	-1.277
energy_100g	0.0019	0.000	13.420	0.000	0.002	0.002
fat_100g	0.1215	0.007	17.966	0.000	0.108	0.135
saturated-fat_100g	0.2159	0.008	41.069	0.000	0.201	0.231
sugars_100g	0.2438	0.004	57.835	0.000	0.236	0.252
proteins_100g	0.1199	0.006	21.578	0.000	0.109	0.131
salt_100g	2.0757	0.030	68.995	0.000	2.017	2.135
fiber_100g	-0.4935	0.018	-27.320	0.000	-0.529	-0.458

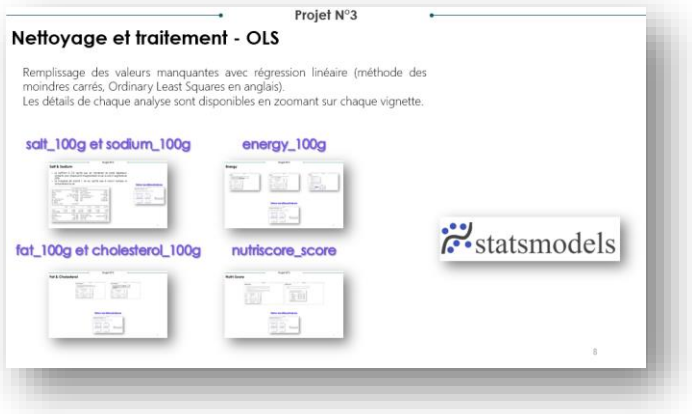
Omnibus:	8093.839	Durbin-Watson:	1.416
Prob(Omnibus):	0.000	Jarque-Bera (JB):	695985.512
Skew:	-1.666	Prob(JB):	0.00
Kurtosis:	34.258	Cond. No.	2.79e+03

Nutri-Score

- Les données concernant le *Nutri-Score* ont un taux de remplissage élevé ~90%
- Le traitement du *Nutri-Score* serait envisagé lors de l'Analyse exploratoire

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18967 entries, 346 to 1480666
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   energy_100g         17627 non-null  float32
 1   fat_100g            17690 non-null  float32
 2   saturated-fat_100g  17428 non-null  float32
 3   sugars_100g         17416 non-null  float32
 4   proteins_100g       17613 non-null  float32
 5   salt_100g           17402 non-null  float32
 6   fiber_100g          17398 non-null  float32
 7   nutriscore_score    16934 non-null  float32
dtypes: float32(8)
memory usage: 1.3 MB
```


Retour aux slides principaux



Nutri-Score

- Le *Nutri-Score* respecte les valeurs maximum et minimum de -15 et 40. La couleur verte (A) correspondant à une valeur comprise entre –15 et –2, le vert clair (B) de –1 à +3, le jaune (C) de +4 à +11, l'orange (D) de +12 à +16 et le rouge (E) de +17 à +40.
- Selon l'Agence Nationale de Sante Publique, le *Nutri-Score* est calculé avec : valeur énergétique, matières graisses, acides gras saturés, sucres, protéines, sel / sodium, fibres, % des fruits, légumes dans le produit

OLS Regression Results

Dep. Variable:	nutriscore_score	R-squared:	0.682
Model:	OLS	Adj. R-squared:	0.681
Method:	Least Squares	F-statistic:	5173.
Date:	Sun, 06 Dec 2020	Prob (F-statistic):	0.00
Time:	18:35:59	Log-Likelihood:	-50436.
No. Observations:	16926	AIC:	1.009e+05
Df Residuals:	16918	BIC:	1.010e+05
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.4483	0.087	-16.565	0.000	-1.620	-1.277
energy_100g	0.0019	0.000	13.420	0.000	0.002	0.002
fat_100g	0.1215	0.007	17.966	0.000	0.108	0.135
saturated-fat_100g	0.3159	0.008	41.369	0.000	0.301	0.331
sugars_100g	0.2438	0.004	57.835	0.000	0.236	0.252
proteins_100g	0.1199	0.006	21.578	0.000	0.109	0.131
salt_100g	2.0757	0.030	68.995	0.000	2.017	2.135
fiber_100g	-0.4935	0.018	-27.320	0.000	-0.529	-0.458

Omnibus:	8893.839	Durbin-Watson:	1.416
Prob(Omnibus):	0.000	Jarque-Bera (JB):	696905.512
Skew:	-1.666	Prob(JB):	0.00
Kurtosis:	34.258	Cond. No.	2.79e+03

Nutri-Score

- Les données concernant le *Nutri-Score* ont un taux de remplissage élevé ~90%
- Le traitement du *Nutri-Score* serait envisagé lors de l'Analyse exploratoire

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18967 entries, 346 to 1480666
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   energy_100g         17627 non-null  float32
 1   fat_100g            17690 non-null  float32
 2   saturated-fat_100g  17428 non-null  float32
 3   sugars_100g         17416 non-null  float32
 4   proteins_100g       17613 non-null  float32
 5   salt_100g           17402 non-null  float32
 6   fiber_100g          17398 non-null  float32
 7   nutriscore_score    16934 non-null  float32
dtypes: float32(8)
memory usage: 1.3 MB
```