P3_01_notebook

September 8, 2021

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# explicitly require this experimental feature
from sklearn.experimental import enable_iterative_imputer  # noqa
# now you can import normally from sklearn.impute
from sklearn.impute import IterativeImputer
from sklearn import decomposition, preprocessing
from functools import *
from scipy.stats.mstats import kruskal
import plotly.graph_objects as go
import seaborn as sns
sns.set()
```

1 Phase préliminaire

1.1 Chargement du fichier de données brutes

[2]:
```
food = pd.read_csv("fr.openfoodfacts.org.products.csv/fr.openfoodfacts.org.
 ↪products.csv", sep='\t'); #le séparateur '\t' correspond à la tabulation ou␣
 ↪shift
```

```
       ␣
↪---------------------------------------------------------------------------

       FileNotFoundError                         Traceback (most recent call␣
↪last)

       <ipython-input-2-57f8b44fdad1> in <module>
   ----> 1 food = pd.read_csv("fr.openfoodfacts.org.products.csv/fr.
↪openfoodfacts.org.products.csv", sep='\t'); #le séparateur '\t' correspond à␣
↪la tabulation ou shift
```

```
       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪parser_f(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
↪usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters,␣
↪true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,␣
↪na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates,␣
↪infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates,␣
↪iterator, chunksize, compression, thousands, decimal, lineterminator,␣
↪quotechar, quoting, doublequote, escapechar, comment, encoding, dialect,␣
↪error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map,␣
↪float_precision)
       674         )
       675 
   --> 676         return _read(filepath_or_buffer, kwds)
       677 
       678     parser_f.__name__ = name


       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪_read(filepath_or_buffer, kwds)
       446 
       447     # Create the parser.
   --> 448     parser = TextFileReader(fp_or_buf, **kwds)
       449 
       450     if chunksize or iterator:


       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪__init__(self, f, engine, **kwds)
       878             self.options["has_index_names"] = kwds["has_index_names"]
       879 
   --> 880         self._make_engine(self.engine)
       881 
       882     def close(self):


       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪_make_engine(self, engine)
      1112     def _make_engine(self, engine="c"):
      1113         if engine == "c":
   -> 1114             self._engine = CParserWrapper(self.f, **self.options)
      1115         else:
      1116             if engine == "python":


       ~/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in␣
↪__init__(self, src, **kwds)
      1889         kwds["usecols"] = self.usecols
      1890
   -> 1891         self._reader = parsers.TextReader(src, **kwds)
      1892         self.unnamed_cols = self._reader.unnamed_cols
      1893


       pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()


       pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.
   ↪_setup_parser_source()


       FileNotFoundError: [Errno 2] File fr.openfoodfacts.org.products.csv/fr.
   ↪openfoodfacts.org.products.csv does not exist: 'fr.openfoodfacts.org.products.
   ↪csv/fr.openfoodfacts.org.products.csv'
```

2 Présentation générale du jeu de données

2.1 Présentation résumée du fichier brute de données

[ ]:
```
food.head()
food.describe()

## Recuperation de la forme courante du tableau de données
np.shape(food)
form_df=pd.DataFrame(data={'num_rows':[np.shape(food)[0]],
                           'num_columns':[np.shape(food)[1]],
                           'clean_step':'Raw'})
form_df
```

2.1.1 Répartition par type de variable

[ ]:
```
pres_gen = pd.DataFrame(data =[10, 24, 3, 32, 111],
                        index=["General information",
                               "Tags",
                               "Ingredients",
                               "Miscellanous data",
                               "Nutrition facts"],
                        columns=["Count"])

##Tracé du pie chart des types de variables
uneliste= pres_gen["Count"].sort_values(ascending=False).index
```

```
plt.figure(figsize=(4,4))
plt.pie(pres_gen["Count"].sort_values(ascending=False),
       labels= uneliste);
plt.title('Counts acc. to variables types');
plt.savefig('./VarTypeCounts.png', bbox_inches='tight')
```

3 Démarche méthodologique d'analyse des données

3.1 Pré-traitement du fichier brut

3.1.1 Détermination du taux de remplissage et détermination de la valeur plancher

[ ]:
```
na_rate = 1 - food.isna().mean()
#print(na_rate.index[condition][0:90])

fill_dist = []
fill_values = np.arange(0.01, 1.0, 0.005)
for val in fill_values :
    condition = na_rate.values <= val
    fill_dist.append(len(na_rate.index[condition]))

##Tracé du nombre cumulé de colonnes ayant un plafond
plt.barh(fill_values, width=fill_dist);
plt.axis([130, 180, 0, 0.65]);
plt.title('Cumulative number of colums with fill rate within a cap');
plt.xlabel('Number of columns');
plt.ylabel('Fill rate');

plt.savefig('./CumulFillRate.png', bbox_inches='tight')
```

[ ]:
```
len(na_rate.index[na_rate.values <= 0.995])
```

Il est maintnenant su que sur les 180 colonnes, il y en a 167 qui ont au plus 60% de remplissage, et 13 ont un taux de rermplissage de plus de 99.5%. Il faut donc maintenant trouver un seuil en dessous duquel les variables seront supprimées. On pourrait par exemple, identifier un ensemble de variables non-pertinentes - eu égard aux objectifs/cibles de l'analyse - et déterminer le seuil qui élimine un nombre important de variables non-pertinentes, rapporté au nombre de variables pertinentes ou intéressantes également éliminées. Par exemple, on a à peu près 165 colonnes qui ont moins de 40% de remplissage. Si on a var. pert. = 140

-> *ratio var. pert./165 = 0.85 >* **80%**

Alors, on peut considérer qu'on perd peu ! 0.4 sera alors un bon seuil pour la suppression des valeurs manquantes. Sinon, on fera varier ce seuil dans un sens ou dans l'autre!

```
[ ]: ## Taux de remplissage inférieur à 0.1
na_rate.index[na_rate.values <= 0.1][0:99];
#na_rate.index[na_rate.values <= 0.1][99:115];

## Taux de remplissage inférieur entre 0.1 et 0.4
na_rate.index[(na_rate.values > 0.1) & (na_rate.values <= 0.4)]

## Taux de remplissage supérieur à 0.4
#na_rate.index[na_rate.values > 0.4]
```

3.1.2 Fill rate cap based drop

```
[ ]: fill_rate_cap = 0.4

# Definition of the columns to drop, based on the chosen fill rate cap chosen␣
 ↪above
col_to_drop_fillratebased = na_rate.index[na_rate.values <= fill_rate_cap]
col_excepted = ['quantity', 'allergens', 'serving_size', 'serving_quantity',␣
 ↪'additives_fr', 'nova_group', 'brand_owner','fiber_100g', 'vitamin-c_100g']
col_to_drop_fillratebased.drop(labels=col_excepted)

#Dropping the columns
food = food.drop(columns=col_to_drop_fillratebased)
```

```
[ ]: ##Connaitre à titre indicatif les colonnes ayant un taux de remplissage de plus␣
 ↪de 99.5% avant imputation
na_rate.index[na_rate.values > 0.995];
```

```
[ ]: ## Mise à jour de l'évolution de la forme du dataframe principal
np.shape(food)
form_df = form_df.append(pd.DataFrame(data={'num_rows':[np.shape(food)[0]],
                                            'num_columns':[np.shape(food)[1]],
                                            'clean_step':
 ↪'FillRateCapBased_drop'}),
                                      ignore_index=True #argument permettant␣
 ↪d'éviter que l'index du df ajouté soit 0
                         )
```

3.1.3 Suppression des colonnes non-pertinentes

Suppression des métadonées non pertinentes

```
[ ]: ##Suppression Manuelle
col_to_drop_non_pertinentes = ['created_t', 'created_datetime',
       'last_modified_t', 'last_modified_datetime',
                             'categories', 'categories_tags', 'categories_fr',
                             'countries', 'countries_tags',
                             'ingredients_text',
                             'states', 'states_tags', 'states_fr',
                             'main_category', 'main_category_fr',
       'image_url', 'image_small_url', 'image_ingredients_url',
       'image_ingredients_small_url', 'image_nutrition_url',
       'image_nutrition_small_url',
                              'brands_tags'
                             ]
food = food.drop(columns = col_to_drop_non_pertinentes)
```

[ ]:
```
food.columns
```

Suppression de valeurs redondantes sur la base du calcul de corrélations linéaires - Corrélations entre les variables _100g

[ ]:
```
##Création de la matrice de corrélation
nutrifacts_cols= food.columns[food.columns.str.endswith('_100g')]
nutrifacts_cols= nutrifacts_cols.difference(['nutrition-score-fr_100g'])␣
 ↪#Exemption du nutrition-score-fr_100g
nutrifacts_df = pd.DataFrame(food,columns=nutrifacts_cols)
corrMatrix_100g = nutrifacts_df.corr()

#Création de la heatmap
plt.figure(figsize=(12,10));
sns.heatmap(corrMatrix_100g, annot=True) ;#
plt.title('Heatmap-styled correlation matrix of the proportions');
plt.show();

plt.savefig('./CorrMatrix.png', bbox_inches='tight');
#food[nutrifacts_cols.difference('nutrition-score-fr_100g')].corr()
```

[ ]:
```
##Suppressions manuelles des variables redondantes

print('salt_100g : ', 1- food['salt_100g'].isna().mean())
print('sodium_100g : ', 1- food['sodium_100g'].isna().mean())
print('energy_100g : ', 1- food['energy_100g'].isna().mean())
print('energy-kcal_100g : ', 1- food['energy-kcal_100g'].isna().mean())

col_to_drop_redundant = ['sodium_100g', 'energy-kcal_100g']
food = food.drop(columns = col_to_drop_redundant)
```

[ ]:
```
## Mise à jour de l'évolution de la forme du dataframe principal
np.shape(food)
form_df = form_df.append(pd.DataFrame(data={'num_rows':[np.shape(food)[0]],
                                                    'num_columns':[np.shape(food)[1]],
                                                    'clean_step':
↪'NonRelevantBased_drop'}),
                                             ignore_index=True #argument permettant␣
↪d'éviter que l'index du df ajouté soit 0
                              )
```

3.1.4 Suppression des valeurs aberrantes

Calcul des quantiles de distribution de l'énergie & détermination de la valeur limite supérieure admissible

[ ]:
```
energy_cols = food.columns[food.columns.str.startswith('energy')]

q1_energy = food[energy_cols].quantile(0.25)
q3_energy = food[energy_cols].quantile(0.75)
energy_max_stat = q3_energy + 16*q1_energy #Calcul de la borne interquantile

print(energy_max_stat,"\n")
```

Nutrition facts

[ ]:
```
##Definition du filtre pass_bande de la variable
def filtre_pass_band(var, val_inf, val_sup):
    return (food.loc[:, [var]] >= val_inf).any(axis=1) & (food.loc[:, [var]] >=␣
↪val_sup).any(axis=1)

##Definition des limites
inf= 0.0
sup= 100
sup_energy = energy_max_stat

food = food[filtre_pass_band('proteins_100g', inf, sup) &
            filtre_pass_band('fat_100g', inf, sup) &
            filtre_pass_band('saturated-fat_100g', inf, sup) &
            filtre_pass_band('carbohydrates_100g', inf, sup) &
            filtre_pass_band('salt_100g', inf, sup) &
            filtre_pass_band('sugars_100g', inf, sup) &
            filtre_pass_band('sugars_100g', inf, sup_energy)
           ]
```

##Sélection des colonnes à retenir nutrifacts_cols= food.columns[food.columns.str.endswith('_100g')] nutrifacts_cols= nutrifacts_cols.difference([‘nutrition-score-fr_100g’]) #Suppression du nutrition-score-fr_100g energy_cols = food.columns[food.columns.str.startswith(‘energy’)]

def filtre_pass_band(var, val_inf, val_sup): return (food.loc[:, [var]] >= val_inf).any(axis=1) & (food.loc[:, [var]] >= val_sup).any(axis=1)

##Filtres généraux des nutrition facts filter_pass_haut_proteins = (food.loc[:, [‘proteins_100g’]] >= 0.0).any(axis=1) #PASS TEST filter_pass_haut_energy = (food.loc[:, [‘energy_100g’]] >= 0.0).any(axis=1)#PASS TEST filter_pass_haut_fat = (food.loc[:, [‘fat_100g’]] >= 0.0).any(axis=1)#PASS TEST filter_pass_haut_saturated_fat = (food.loc[:, [‘saturated-fat_100g’]] >= 0.0).any(axis=1)#PASS TEST filter_pass_haut_carbohydrates = (food.loc[:, [‘carbohydrates_100g’]] >= 0.0).any(axis=1)#PASS TEST filter_pass_haut_salt = (food.loc[:, [‘salt_100g’]] >= 0.0).any(axis=1)#PASS TEST filter_pass_haut_sugars = (food.loc[:, [‘sugars_100g’]] >= 0.0).any(axis=1)#PASS TEST

##Filtres pass-bas filter_pass_bas_proteins = (food.loc[:, [‘proteins_100g’]] <= 100.0).any(axis=1) #PASS TEST filter_pass_bas_fat = (food.loc[:, [‘fat_100g’]] <= 100.0).any(axis=1)#PASS TEST filter_pass_bas_saturated_fat = (food.loc[:, [‘saturated-fat_100g’]] <= 100.0).any(axis=1)#PASS TEST filter_pass_bas_carbohydrates = (food.loc[:, [‘carbohydrates_100g’]] <= 100.0).any(axis=1)#PASS TEST filter_pass_bas_salt = (food.loc[:, [‘salt_100g’]] <= 100.0).any(axis=1)#PASS TEST filter_pass_bas_sugars = (food.loc[:, [‘sugars_100g’]] <= 100.0).any(axis=1)#PASS TEST

filter_pass_bas_nrj = (food.loc[:, energy_cols] <= energy_max_stat).any(axis=1)

food = food[filter_pass_haut_proteins & filter_pass_haut_sugars & filter_pass_haut_energy & filter_pass_haut_fat & filter_pass_haut_saturated_fat & filter_pass_haut_carbohydrates & filter_pass_haut_salt &

```
         filter_pass_bas_proteins &
         filter_pass_bas_sugars &
         filter_pass_bas_fat &
         filter_pass_bas_saturated_fat &
         filter_pass_bas_carbohydrates &
         filter_pass_bas_salt &

         filter_pass_bas_nrj
        ]
```

[ ]:
```
food.describe()
#food[~filter_pass_bas_nrj]
#filter_pass_haut
#food.loc[:, nutrifacts_cols] >= 0
```

[19]:
```
## Mise à jour de l'évolution de la forme du dataframe principal
np.shape(food)
form_df = form_df.append(pd.DataFrame(data={'num_rows':[np.shape(food)[0]],
                                             'num_columns':[np.shape(food)[1]],
                                             'clean_step':'PostOutliers'}),
                         ignore_index=True #argument permettant d'éviter que␣
 ↪l'index du df ajouté soit 0
                        )
```

3.1.5 Identification et élimination des doublons

[20]:
```
## Ordonner le df suivant les taux de remplissages décroissant (plus␣
↪précisément -dans le code- les taux de na croissants), puis éliminer les␣
↪doublons en gardant le premier (moins vide).
food["fill_level"]=food.isna().mean(axis=1).sort_values(ascending=True)
food = food.sort_values(by=["fill_level"],ascending=True).
↪drop_duplicates(subset=['code'],keep='first')
```

[21]:
```
## Restauration du dataframe vers les forme et ordre précédant la manipulation
food = food.drop(columns=["fill_level"])
food = food.sort_index()
```

[22]:
```
## Mise à jour de l'évolution de la forme du dataframe principal
np.shape(food)
form_df = form_df.append(pd.DataFrame(data={'num_rows':[np.shape(food)[0]],
                                            'num_columns':[np.shape(food)[1]],
                                            'clean_step':'Duplicates_drop'}),
                         ignore_index=True #argument permettant d'éviter que␣
↪l'index du df ajouté soit 0
                        )
```

3.2 Imputation Statistique

3.2.1 Imputation des nutrifacts _100g

[23]:
```
1 - food.isna().mean()
```

[23]:
```
code                                     1.000000
url                                      1.000000
creator                                  0.999998
product_name                             0.996308
brands                                   0.587707
countries_fr                             0.999362
additives_n                              0.508943
ingredients_from_palm_oil_n              0.508943
ingredients_that_may_be_from_palm_oil_n  0.508943
nutriscore_score                         0.546107
nutriscore_grade                         0.546107
pnns_groups_1                            0.998302
pnns_groups_2                            1.000000
energy_100g                              1.000000
fat_100g                                 1.000000
saturated-fat_100g                       1.000000
carbohydrates_100g                       1.000000
```

```
sugars_100g                                 1.000000
proteins_100g                               1.000000
salt_100g                                   1.000000
nutrition-score-fr_100g                     0.546107
dtype: float64
```

[24]:
```
imputer = IterativeImputer(min_value=0)
imputer.fit(food[nutrifacts_cols])
food.loc[:,nutrifacts_cols] = imputer.transform(food[nutrifacts_cols])
```

[26]:
```
## Mise à jour de l'évolution de la forme du dataframe principal
np.shape(food)
form_df = form_df.append(pd.DataFrame(data={'num_rows':[np.shape(food)[0]],
                                            'num_columns':[np.shape(food)[1]],
                                            'clean_step':'PostImputation'}),
                         ignore_index=True #argument permettant d'éviter que␣
 ↪l'index du df ajouté soit 0
                        )
```

3.2.2 Suppression post-imputation des na

[27]:
```
food = food.dropna(axis='rows')
```

[28]:
```
## Mise à jour de l'évolution de la forme du dataframe principal
np.shape(food)
form_df = form_df.append(pd.DataFrame(data={'num_rows':[np.shape(food)[0]],
                                            'num_columns':[np.shape(food)[1]],
                                            'clean_step':'ResidualNA_drop'}),
                         ignore_index=True #argument permettant d'éviter que␣
 ↪l'index du df ajouté soit 0
                        )
```

[29]:
```
1 - food.isna().mean()
```

[29]:
```
code                                        1.0
url                                         1.0
creator                                     1.0
product_name                                1.0
brands                                      1.0
countries_fr                                1.0
additives_n                                 1.0
ingredients_from_palm_oil_n                 1.0
ingredients_that_may_be_from_palm_oil_n     1.0
nutriscore_score                            1.0
nutriscore_grade                            1.0
```

```
pnns_groups_1                              1.0
pnns_groups_2                              1.0
energy_100g                                1.0
fat_100g                                   1.0
saturated-fat_100g                         1.0
carbohydrates_100g                         1.0
sugars_100g                                1.0
proteins_100g                              1.0
salt_100g                                  1.0
nutrition-score-fr_100g                    1.0
dtype: float64
```

3.2.3 Bilan de la phase de Nettoyage

[30]:
```
form_df
```

[30]:
```
   num_rows  num_columns             clean_step
0   1437214          181                    Raw
1   1437214           45  FillRateCapBased_drop
2   1437214           21  NonRelevantBased_drop
3    980865           21           PostOutliers
4    980665           21        Duplicates_drop
5    980665           21         PostImputation
6    304490           21        ResidualNA_drop
```

[31]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(y=form_df.sort_index(ascending=True)['num_rows'],
                         x=form_df.sort_index(ascending=True)['num_columns'],
                         mode= 'markers',
                         hovertext= form_df.
 ↪sort_index(ascending=True)['clean_step']
                        )
             )
fig.show()
```

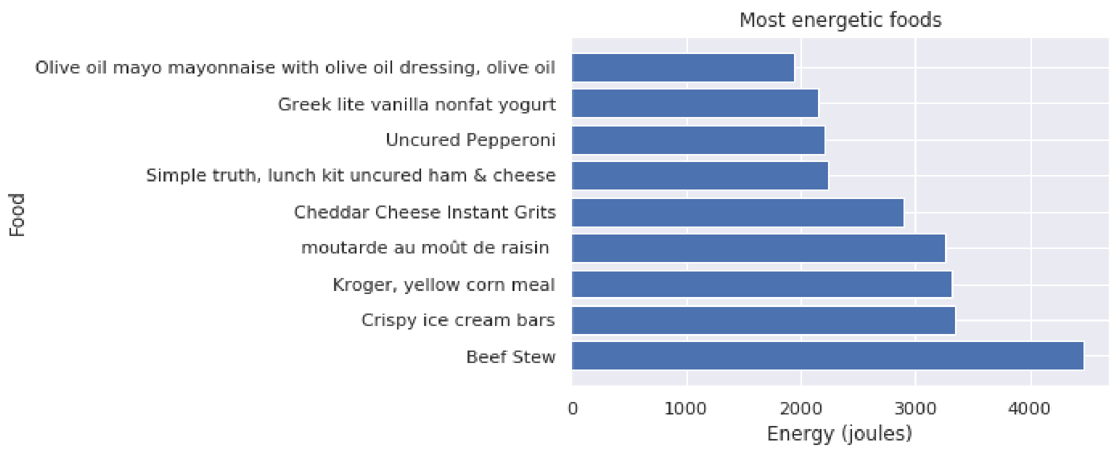
[57]:
```
## Extraction du dataframe form_df
form_df.to_csv('form_df.zip', index=False)
```

3.3 Analyse univariée

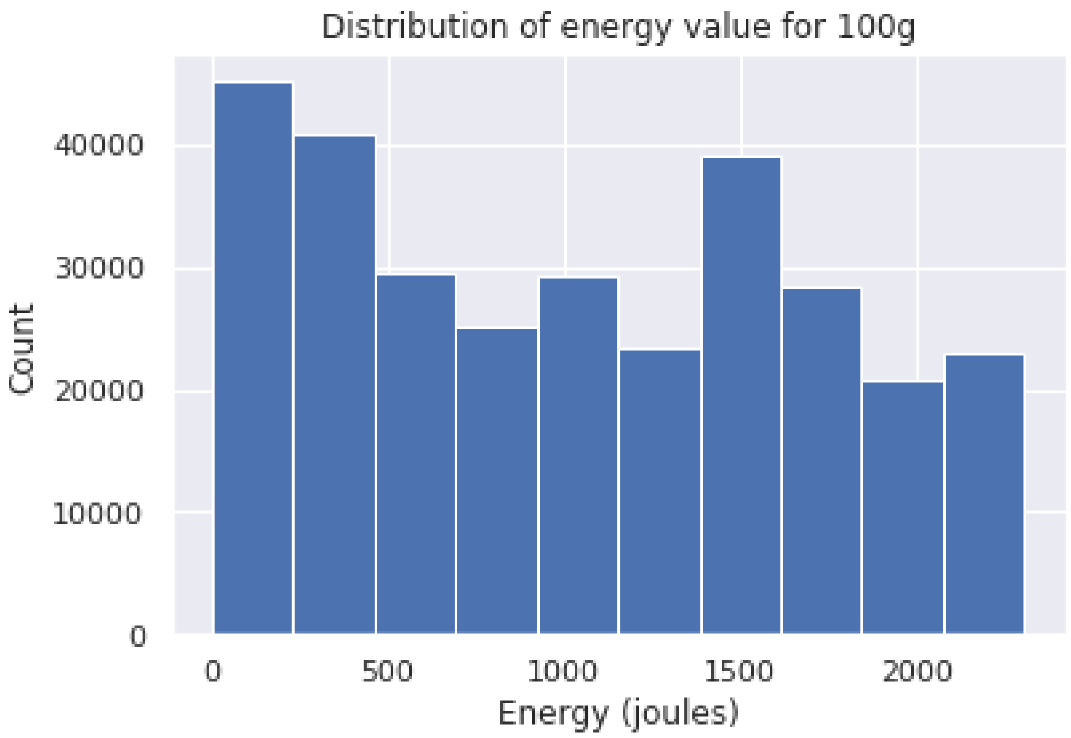
Energy

[32]:

```
ax = plt.barh(food.iloc[food.energy_100g.value_counts().
↪sort_values(ascending=False).index[0:9]].product_name,
        width=food.energy_100g.value_counts().sort_values(ascending = False).
↪values[0:9])

plt.title('Most energetic foods');
plt.xlabel('Energy (joules)');
plt.ylabel('Food');


plt.savefig('./EnergyTopScorers.png', bbox_inches = 'tight')
```



[33]:
```
plt.hist(food.energy_100g);

plt.title('Distribution of energy value for 100g');
plt.xlabel('Energy (joules)');
plt.ylabel('Count');

plt.savefig('./EnergyValuesDistribution.png', bbox_inches='tight')
```

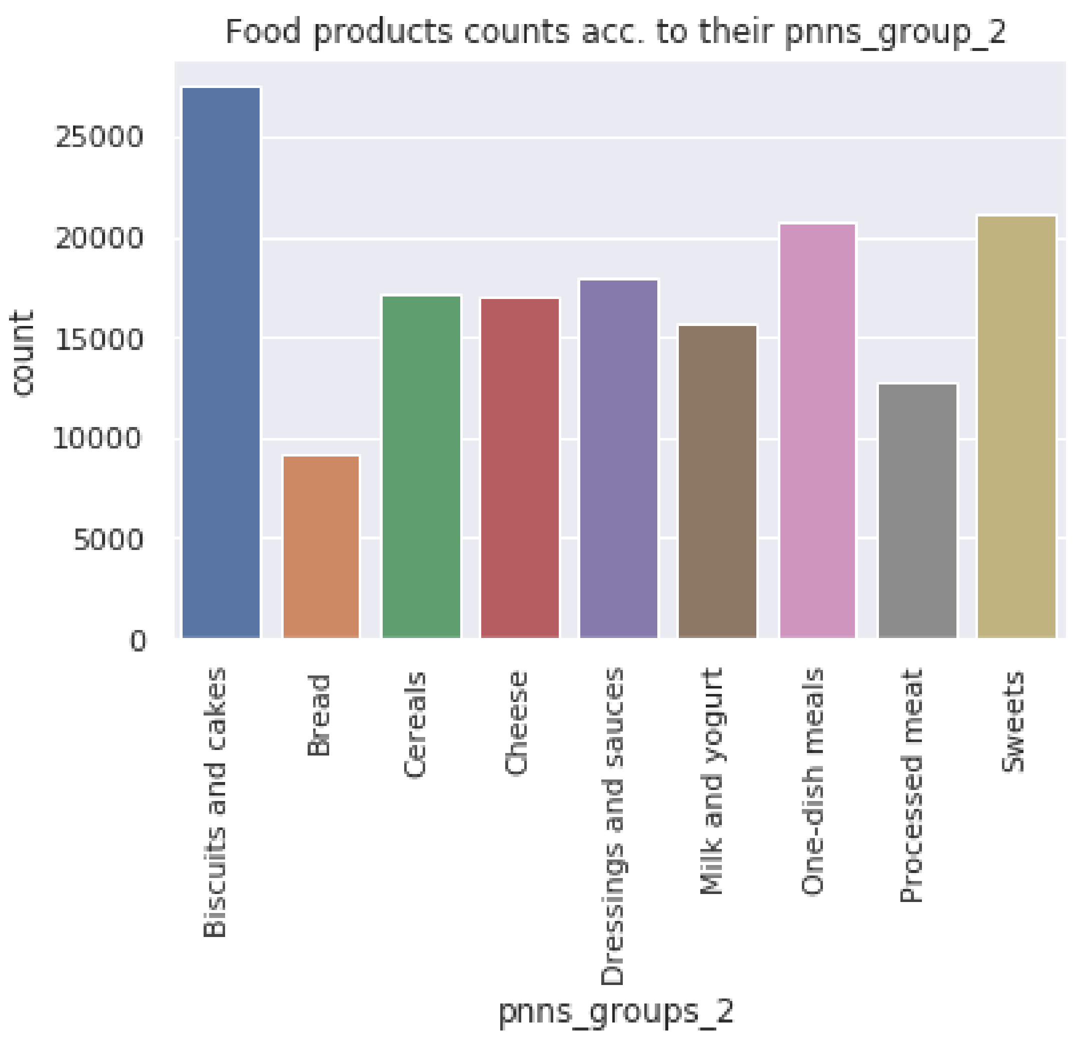


Répartition par Pnns group 2

Commentaire: Modalité "Unknown" Le mode de la distribution est de valeur "Unknown". Au départ, j'avais gardé cette valeur (avec un semilogy), mais finalement je l'ai retirée pour avoir les valeurs effectives.

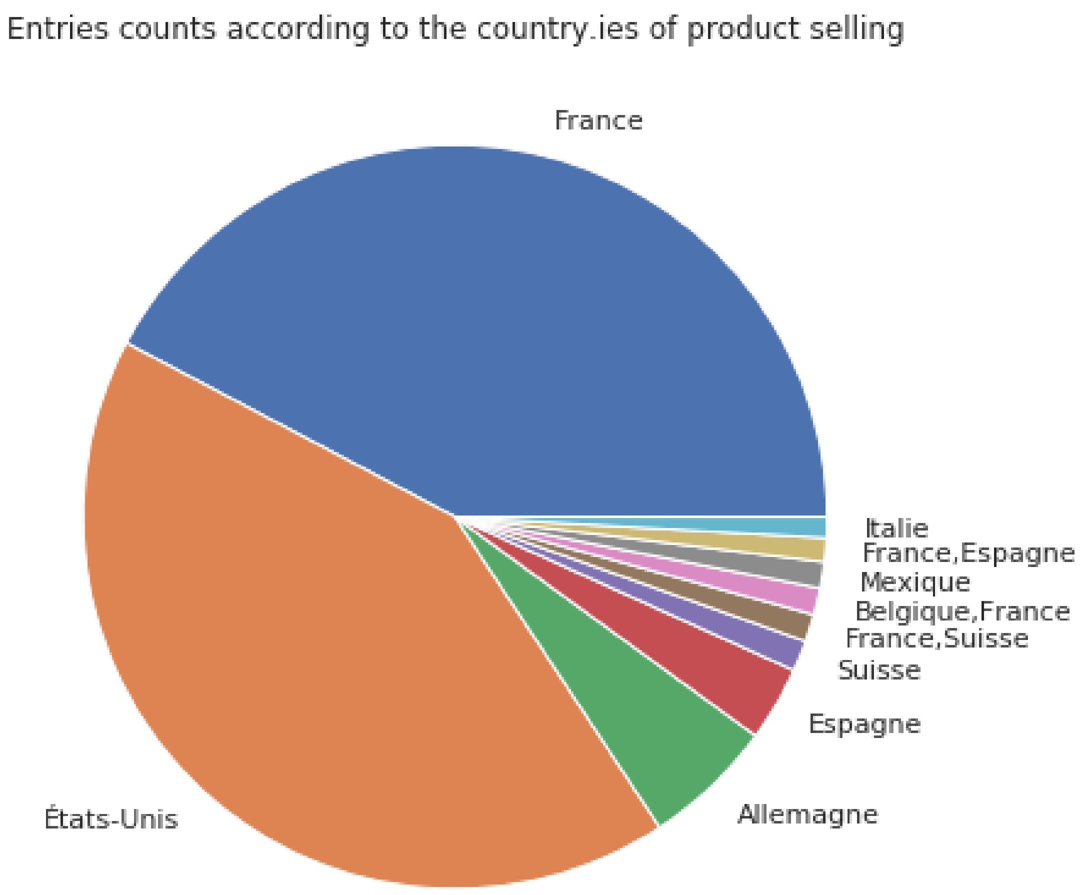
Question Comment faire pour garder l'ordre après l'application de *.difference([unknown])* ?

[34]:
```
uneliste =food['pnns_groups_2'].value_counts().sort_values(ascending=False).
 ↪head(10).index.difference(['unknown']).tolist()
#uneliste =food['pnns_groups_2'].value_counts().sort_values(ascending=False)[1:
 ↪10].index.tolist()
ax = sns.countplot(x = 'pnns_groups_2',
                   order= uneliste,
                   data= food[food['pnns_groups_2'].isin(uneliste)])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
plt.title('Food products counts acc. to their pnns_group_2');

plt.savefig('./pnns_groups.png', bbox_inches='tight')
```



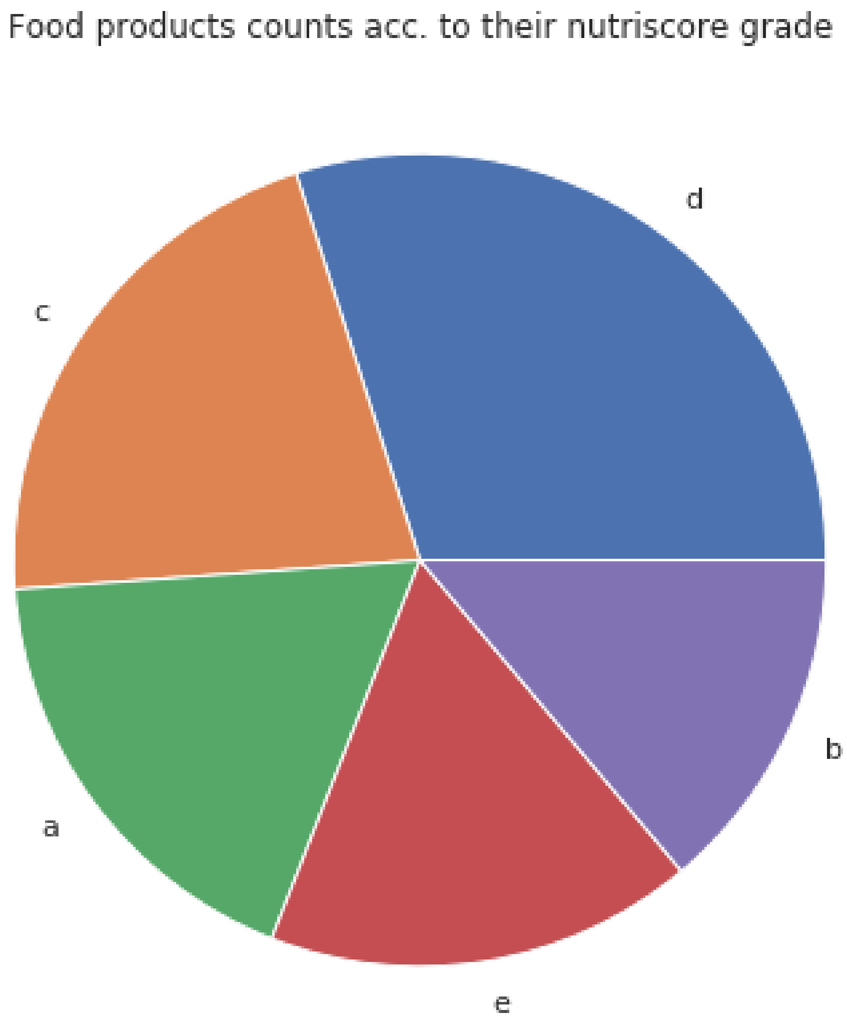
Répartition par pays

[35]:
```
uneliste =food['countries_fr'].value_counts().sort_values(ascending=False).
 ↪head(10).index.tolist()

plt.figure(figsize=(7,7))
plt.pie(food['countries_fr'].value_counts().sort_values(ascending=False).
 ↪head(10),
        labels = uneliste);

plt.title('Entries counts according to the country.ies of product selling');

plt.savefig('./SellingCountriesCounts.png', bbox_inches = 'tight')
```



Répartition par nutrigrade

[36]:
```python
uneliste =food['nutriscore_grade'].value_counts().sort_values(ascending=False).
 ↪head(10).index.tolist()

plt.figure(figsize=(7,7))
plt.pie(food['nutriscore_grade'].value_counts().sort_values(ascending=False).
 ↪head(10),
        labels = uneliste);

plt.title('Food products counts acc. to their nutriscore grade');

plt.savefig('./NutrigradeCounts.png', bbox_inches = 'tight')
```



3.4 Analyse Bivariée

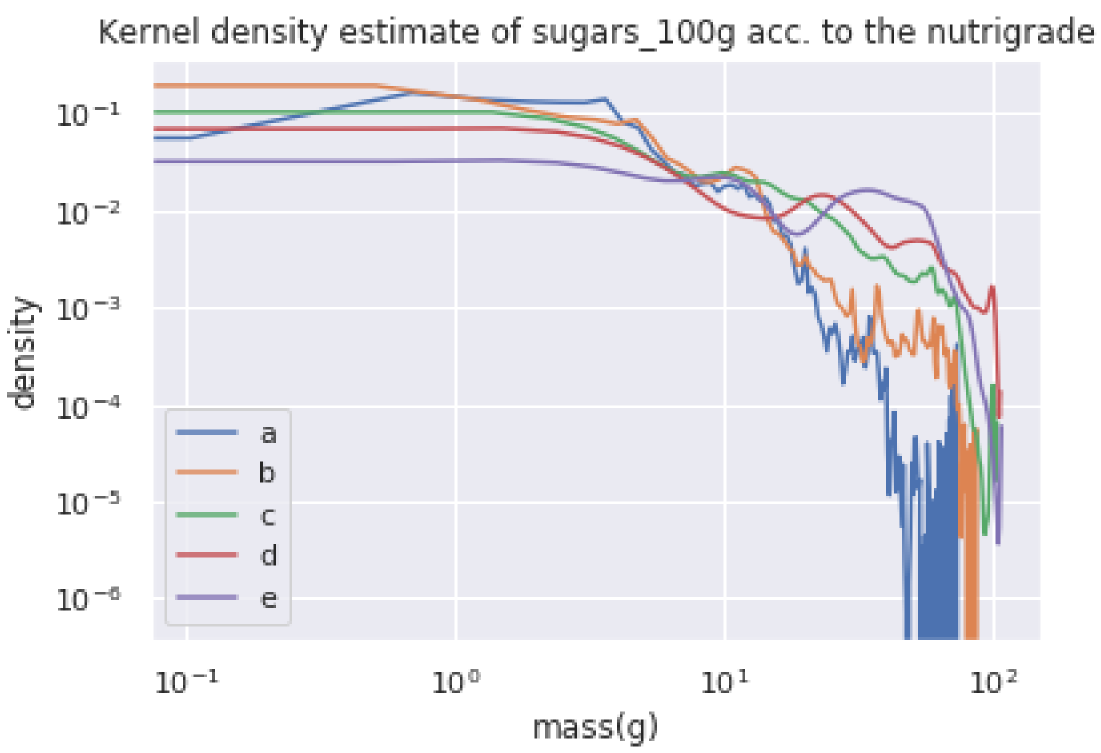
3.4.1 Kernel Density Estimates PAG: 30/08/2020 Modifier le paramètre "kernel=" pour régler souci de "sauts" dans le graphe

Sugars

[37]:
```
nutrigrade_values = ['a','b', 'c', 'd', 'e']
proceeded_variable = 'sugars_100g'

plt.semilogy(); #plus parlant car on voit mieux l'aire sous la courbe
plt.semilogx();
```

```
for grade in nutrigrade_values:
    sns.kdeplot(data=food[food['nutriscore_grade']==grade][proceeded_variable]);

plt.title('Kernel density estimate of '+proceeded_variable+' acc. to the␣
 ↪nutrigrade' );
plt.xlabel('mass(g)')
plt.ylabel('density')
plt.legend(nutrigrade_values);
```

```
/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
```

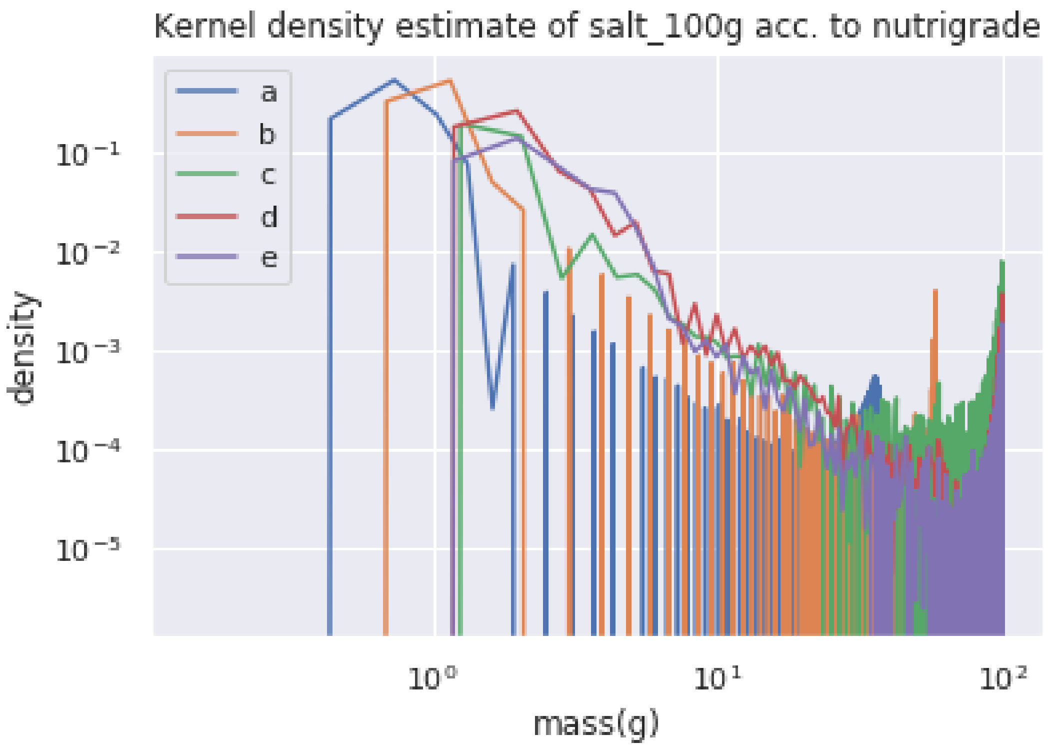


Salt

```
[38]: nutrigrade_values = ['a','b', 'c', 'd', 'e']
proceeded_variable = 'salt_100g'

plt.semilogy();
plt.semilogx();

for grade in nutrigrade_values:
    sns.kdeplot(data=food[food['nutriscore_grade']==grade][proceeded_variable]);

plt.title('Kernel density estimate of '+proceeded_variable+' acc. to␣
 ↪nutrigrade' );
plt.xlabel('mass(g)')
plt.ylabel('density')
plt.legend(nutrigrade_values);

plt.savefig('./kdeplot_sugars.png', bbox_inches='tight')
```

```
/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
```

```
/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
```

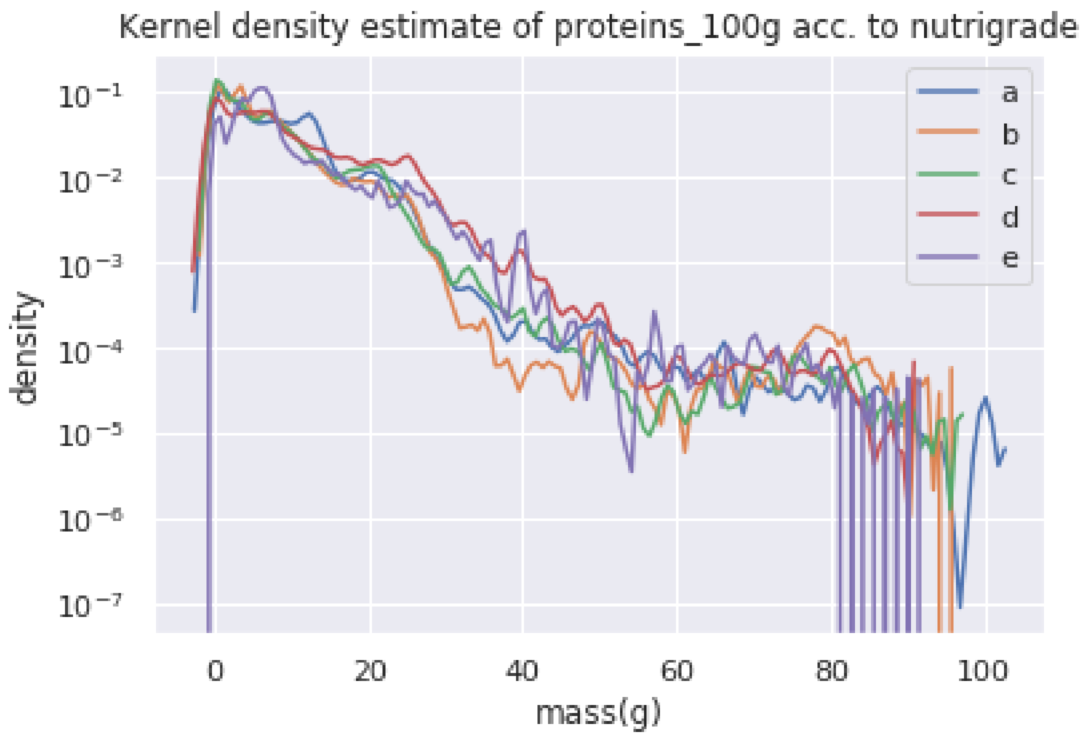


Proteins

```
[39]: nutrigrade_values = ['a','b', 'c', 'd', 'e']
      proceeded_variable = 'proteins_100g'

      plt.semilogy();
      #plt.semilogx();

      for grade in nutrigrade_values:
          sns.kdeplot(data=food[food['nutriscore_grade']==grade][proceeded_variable]);

      plt.title('Kernel density estimate of '+proceeded_variable+' acc. to␣
       ↪nutrigrade' );
      plt.xlabel('mass(g)')
      plt.ylabel('density')
      plt.legend(nutrigrade_values);

      plt.savefig('./kdeplot_proteins.png', bbox_inches='tight')
```

```
/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:
```

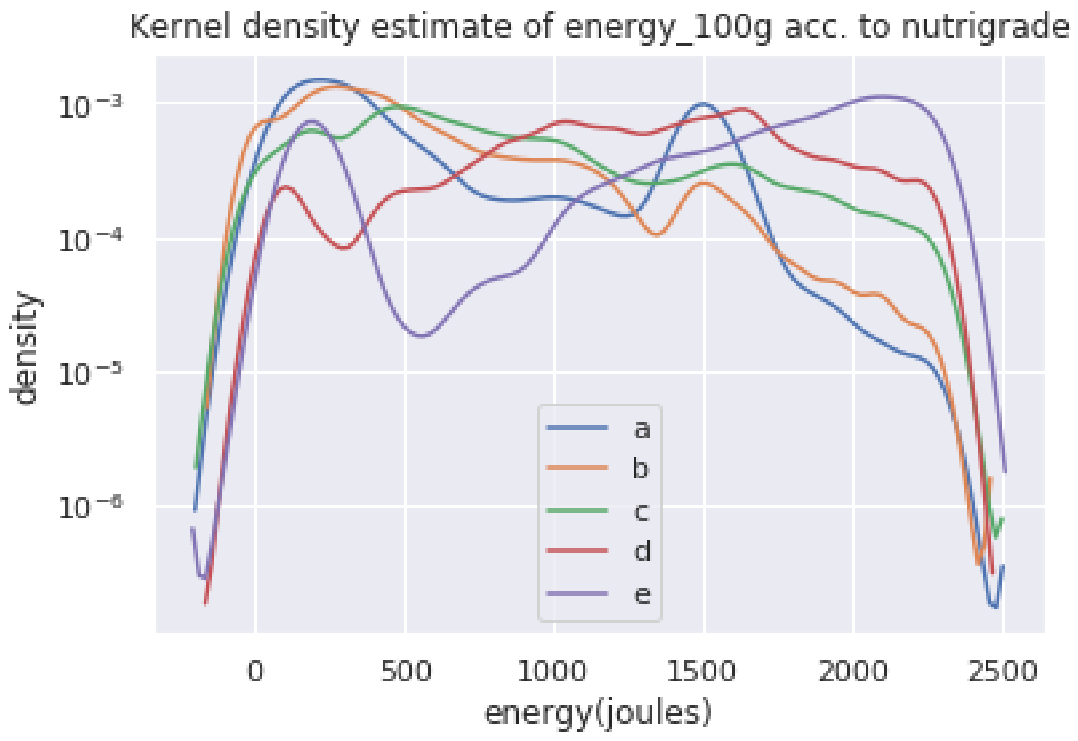
```
Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
```



Energy

```
[40]: nutrigrade_values = ['a','b', 'c', 'd', 'e']
      proceeded_variable = 'energy_100g'

      plt.semilogy();
      #plt.semilogx();

      for grade in nutrigrade_values:
          sns.kdeplot(data=food[food['nutriscore_grade']==grade][proceeded_variable]);

      plt.title('Kernel density estimate of '+proceeded_variable+' acc. to␣
       ↪nutrigrade' );
      plt.xlabel('energy(joules)')
      plt.ylabel('density')
      plt.legend(nutrigrade_values);

      plt.savefig('./kdeplot_energy.png', bbox_inches='tight')
```

```
/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.

/home/erbadi/anaconda3/lib/python3.7/site-packages/seaborn/distributions.py:352:
UserWarning:

Attempted to set non-positive bottom ylim on a log-scaled axis.
Invalid limit will be ignored.
```

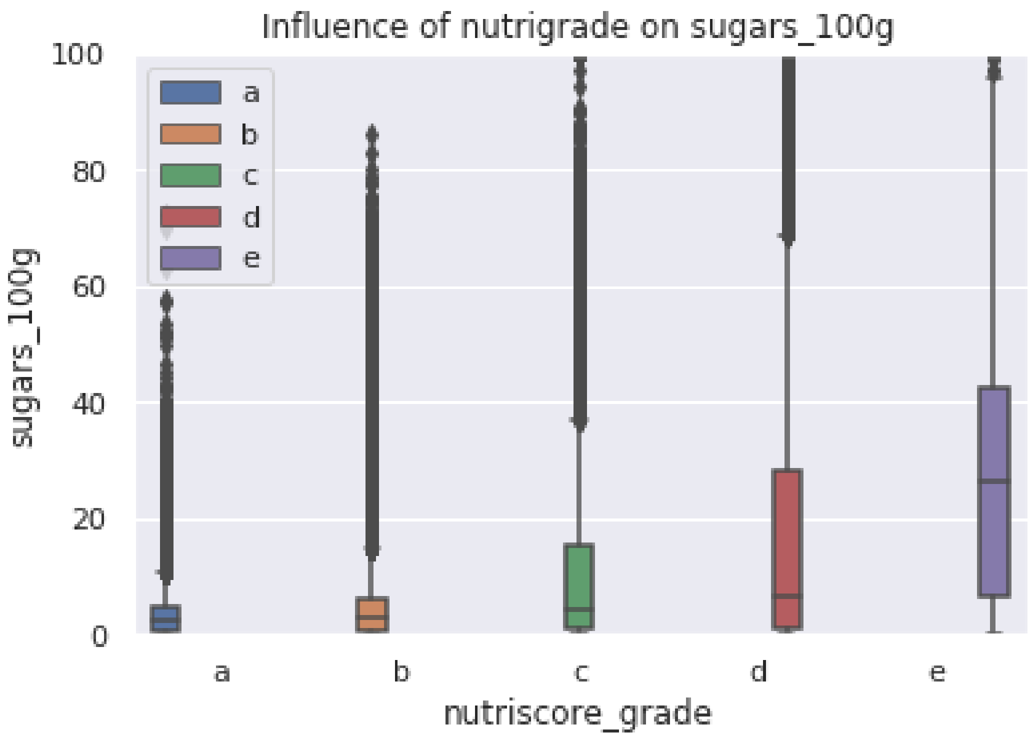


Question Que fait le Kdeplot exactement ?

3.4.2 Grouped Boxplot

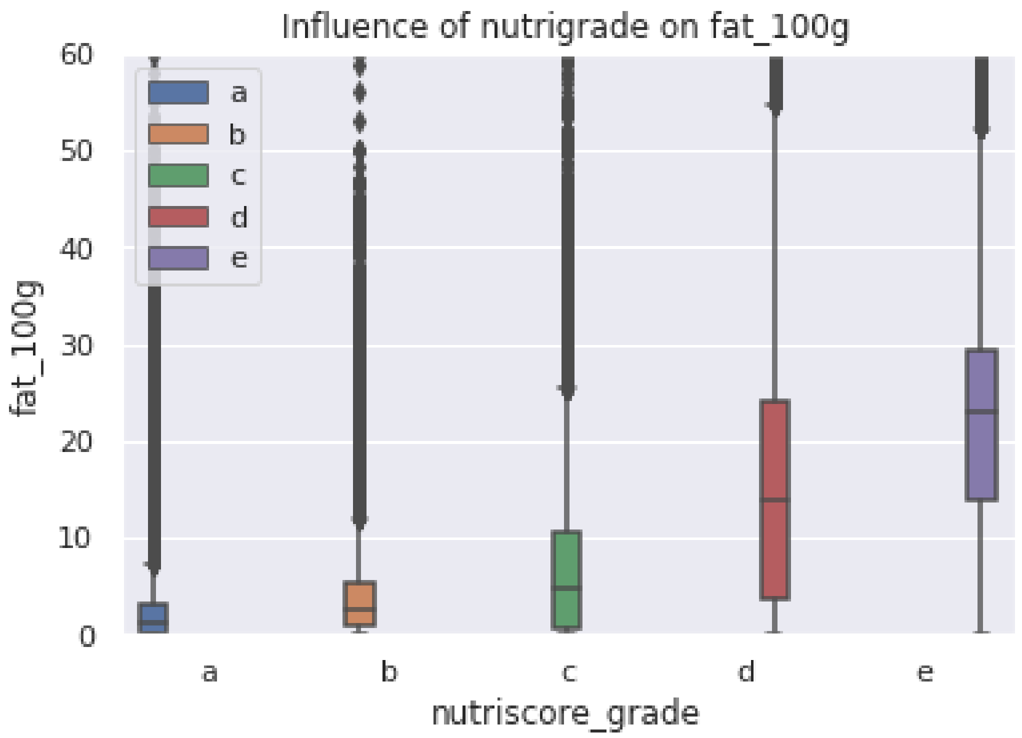
Sugars

[41]:
```
proceeded_variable = 'sugars_100g'

ax = sns.boxplot(x="nutriscore_grade",y=proceeded_variable,
            hue="nutriscore_grade",
            hue_order=nutrigrade_values,
            order=nutrigrade_values,
            data=food);

plt.title('Influence of nutrigrade on '+proceeded_variable);
plt.legend(loc='upper left');
ax.set(ylim=(0, 100));# Il y quelques outliers en dehors au dessus de la valeur␣
 ↪de 150 que j'ai prise comme limite supérieure de la fenêtre de tracé!
```



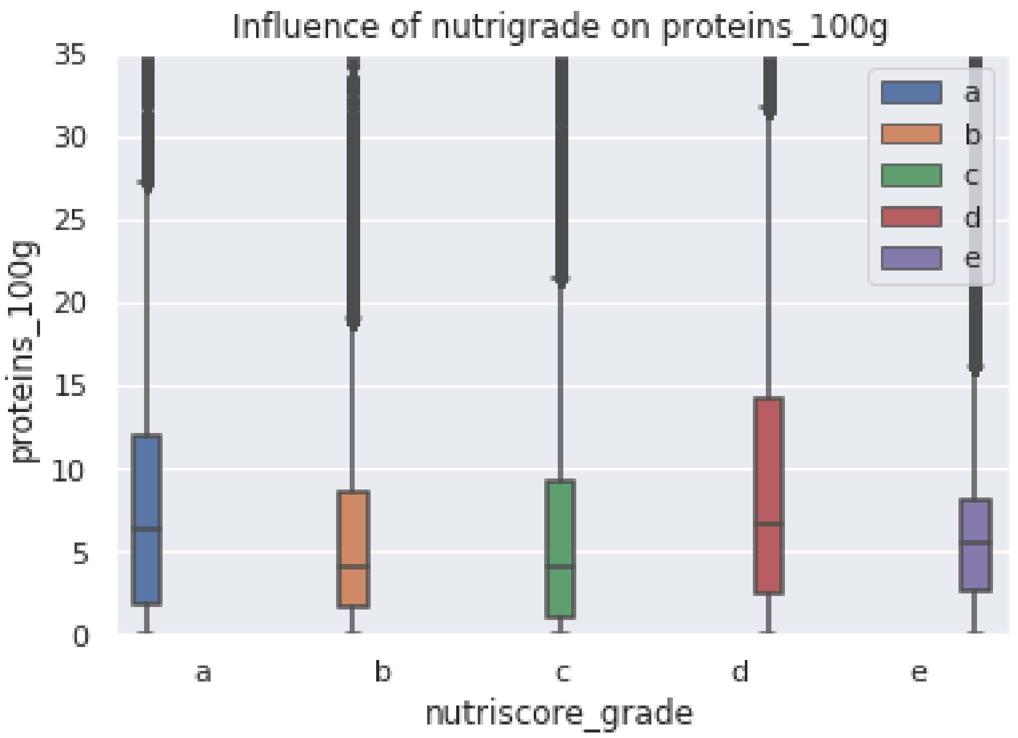
Fat

[42]:
```
proceeded_variable = 'fat_100g'

ax = sns.boxplot(x="nutriscore_grade",y=proceeded_variable,
            hue="nutriscore_grade",
            hue_order=nutrigrade_values,
            order=nutrigrade_values,
            data=food);

plt.title('Influence of nutrigrade on '+proceeded_variable);
plt.legend(loc='upper left');
ax.set(ylim=(0, 60));# Il y quelques outliers en dehors au dessus de la valeur␣
 ↪de 150 que j'ai prise comme limite supérieure de la fenêtre de tracé!
```



Proteins

[43]:
```
proceeded_variable = 'proteins_100g'

ax = sns.boxplot(x="nutriscore_grade",y=proceeded_variable,
            hue="nutriscore_grade",
            hue_order=nutrigrade_values,
            order=nutrigrade_values,
            data=food);

plt.title('Influence of nutrigrade on '+proceeded_variable);
plt.legend(loc='upper right');
ax.set(ylim=(0, 35));# Il y quelques outliers en dehors au dessus de la valeur␣
↪de 150 que j'ai prise comme limite supérieure de la fenêtre de tracé!

plt.savefig('./gboxplot_proteins_nutrigr.png', bbox_inches='tight')
```



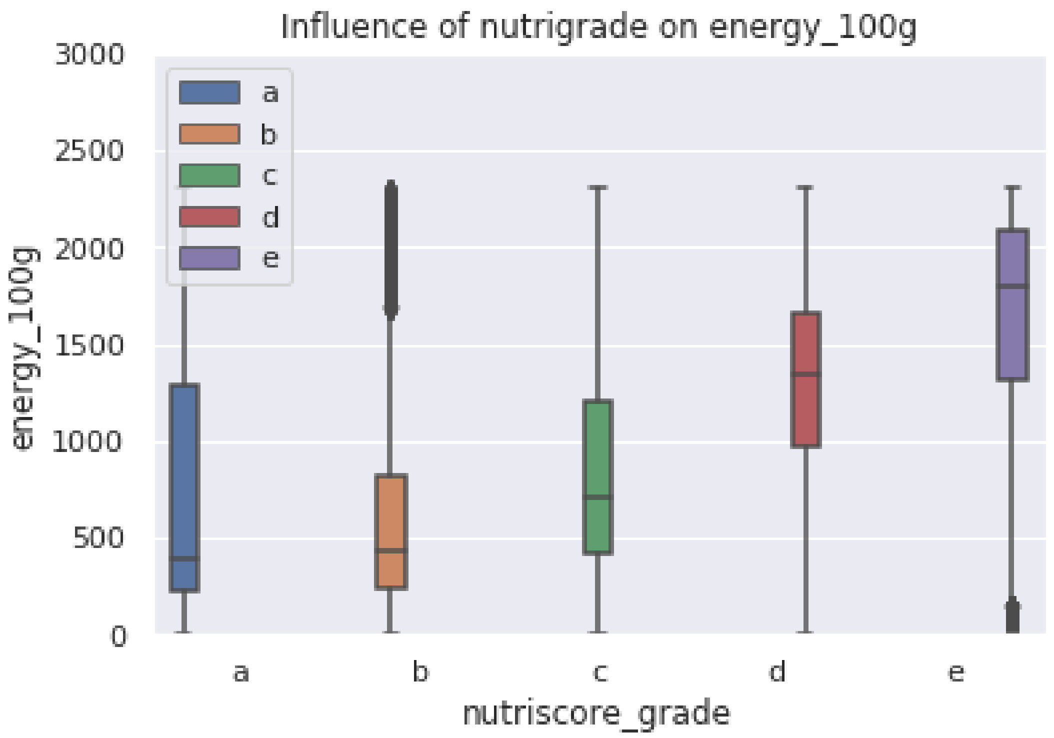
Energy

[ ]:

```
```

[44]:

```
proceeded_variable = 'energy_100g'

ax = sns.boxplot(x="nutriscore_grade",y=proceeded_variable,
            hue="nutriscore_grade",
            hue_order=nutrigrade_values,
            order=nutrigrade_values,
            data=food);

plt.title('Influence of nutrigrade on '+proceeded_variable);
plt.legend(loc='upper left');
ax.set(ylim=(0, 3000));# Il y un outlier  au dessus de la valeur de 5 000 000␣
 ↪que j'ai prise comme limite supérieure de la fenêtre de tracé!

plt.savefig('./gboxplot_energy_nutrigr.png', bbox_inches='tight')
```



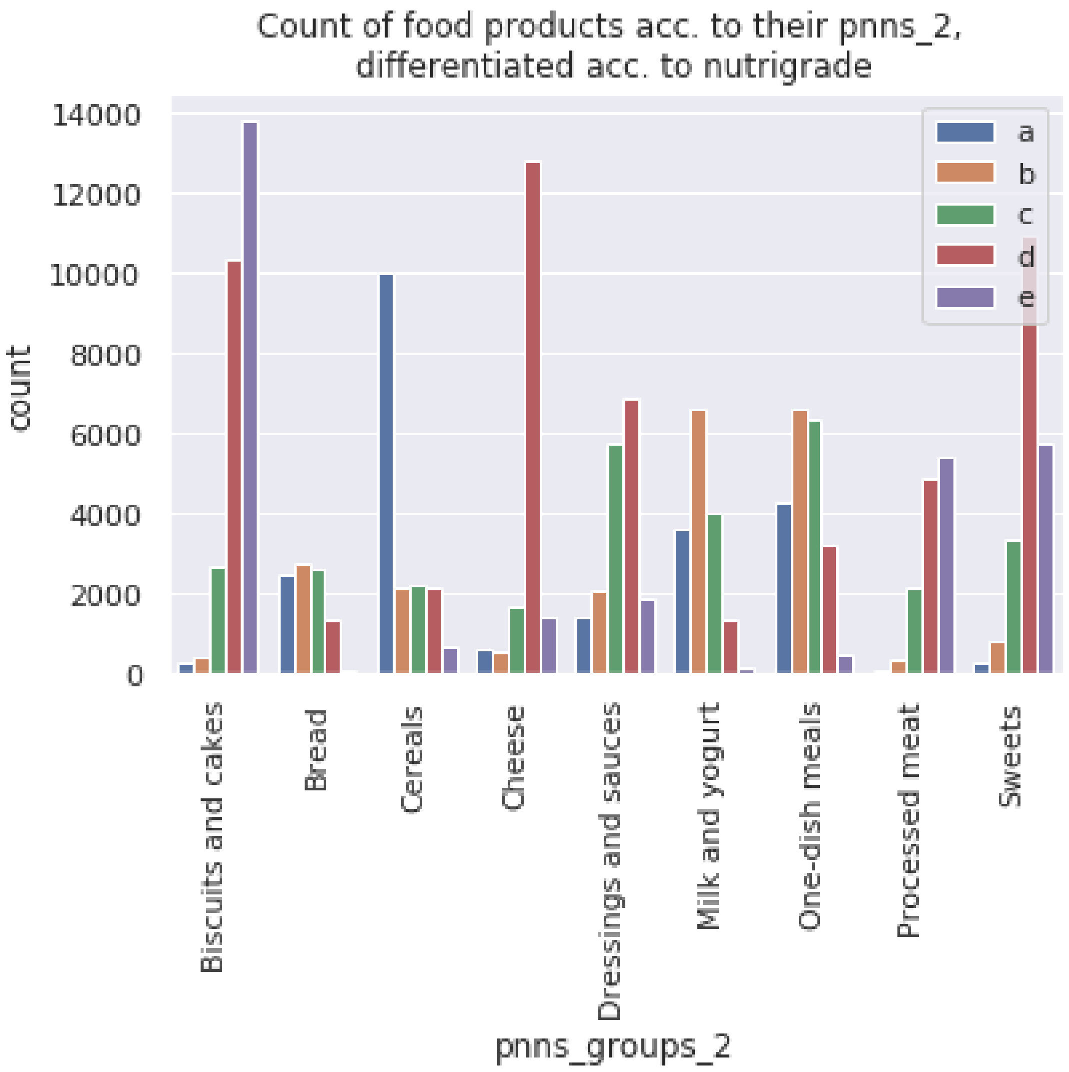
3.4.3 Countplot/ Nutrigrade - pnns_groups_2

PAG Palette de couleur custom avec le code donné sur slack.

PAG **Virer le cas "unknown"**

[45]:
```
uneliste1 =food['pnns_groups_2'].value_counts().sort_values(ascending=False).
 ↪head(10).index.difference(['unknown']).tolist()
uneliste2 =food['nutriscore_grade'].value_counts().sort_values(ascending=False).
 ↪head(5).index.tolist()
ax = sns.countplot(x='pnns_groups_2',
                   hue='nutriscore_grade',
                   order= uneliste1,
                   hue_order=['a', 'b', 'c', 'd', 'e'],
                   data=food[(food['pnns_groups_2'].isin(uneliste1)) &
                                     (food['nutriscore_grade'].
 ↪isin(uneliste2))])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);# Sans le semi-colon, des␣
 ↪chaines de caractère apparaissent avant le graphe

plt.title('Count of food products acc. to their pnns_2, \ndifferentiated acc.␣
 ↪to nutrigrade');
```

```
plt.legend(loc='upper right')

plt.savefig("./figDamArr.png", bbox_inches='tight') # le secnd param permet␣
↪d'éviter un rognement de l'image en
```



3.5 ACP sur les nutrifacts "_100g"

3.5.1 Données centrées réduites

[46]:
```
X_scaled = preprocessing.StandardScaler().fit_transform(food[nutrifacts_cols])
nutrifacts_scaled = pd.DataFrame(X_scaled, columns=nutrifacts_cols,␣
↪index=food[nutrifacts_cols].index)
```

```
#Display
nutrifacts_scaled
```

```
[46]:          carbohydrates_100g  energy_100g  fat_100g  proteins_100g  salt_100g  \
3                   -0.037831    -0.147196 -0.209661      -0.319534   1.034842
64                  -0.173457    -0.546922 -0.716755       0.225651  -0.229025
265                  1.901263     0.330379 -0.902690      -0.951453  -0.335688
317                 -0.526799    -0.874787 -0.716755      -0.108894  -0.127129
320                 -0.251978     0.263010  0.111499       1.774472   0.307865
…                         …           …        …             …          …
1437117             -0.641010    -1.229600 -0.860432      -0.889500  -0.326750
1437135             -0.715961    -1.284993 -0.902690      -0.889500  -0.335688
1437137             -0.573197    -1.121809 -0.885787      -0.703642  -0.335688
1437165              1.104282     1.559499  1.125688      -0.096504  -0.049664
1437207              0.176315    -0.163664 -0.260370      -0.629298  -0.252264

         saturated-fat_100g  sugars_100g
3                 -0.344558     0.432870
64                -0.624380    -0.697813
265               -0.706680     2.793038
317               -0.558540    -0.703096
320                0.445526    -0.729514
…                         …           …
1437117           -0.706680    -0.090203
1437135           -0.706680    -0.201158
1437137           -0.706680     0.010185
1437165           -0.393939    -0.650261
1437207            0.083404     0.538541

[304490 rows x 7 columns]
```

3.5.2 Calcul des composantes principales

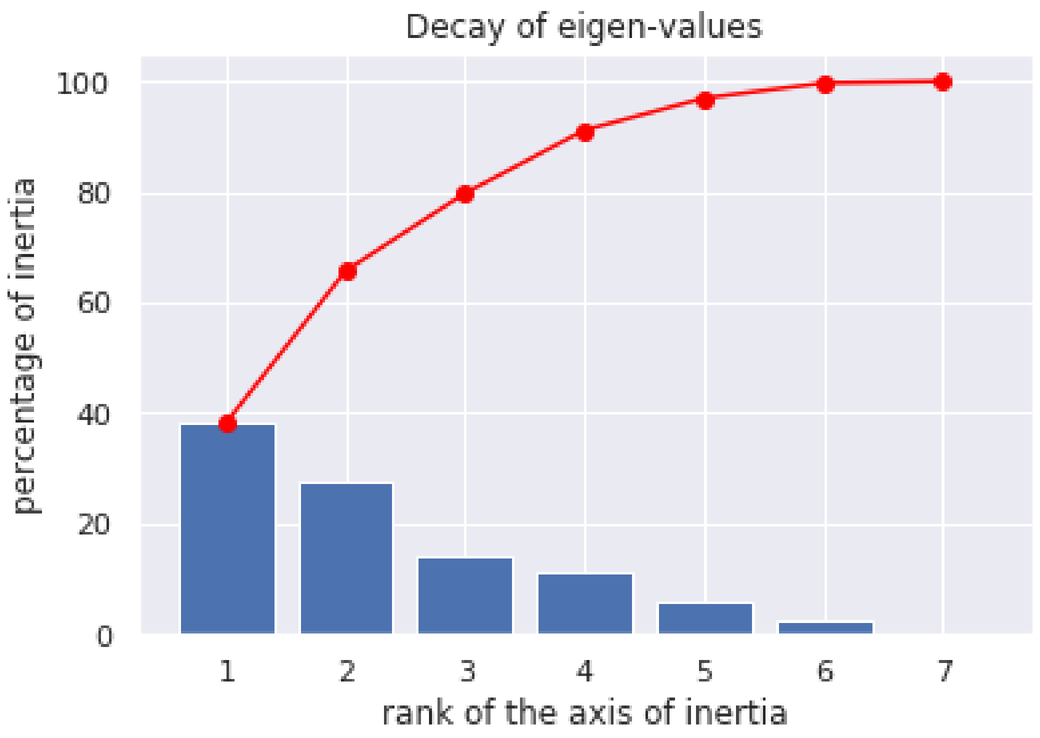
```
[47]: pca = decomposition.PCA(n_components=7)
X_projected = pca.fit_transform(X_scaled)
nutrifacts_pc = pd.DataFrame(X_projected, index=food[nutrifacts_cols].index,␣
 ↪columns=["F"+str(i+1) for i in range(7)])

#Display
nutrifacts_pc
```

```
[47]:            F1        F2        F3        F4        F5        F6        F7
3         -0.321800 -0.321424  1.022933  0.432133  0.236518 -0.216740 -0.056888
64        -1.155038  0.200581 -0.115608 -0.599315 -0.216033  0.171813  0.005896
265        0.506345 -3.554387  0.281700 -0.076467  0.913471 -0.163249  0.144490
```

```
317     -1.485755  0.325283 -0.164914 -0.130875 -0.108745  0.145842  0.011846
320      0.494773  1.484319  0.364769 -1.176008  0.267443  0.289221 -0.009426
…              …        …        …        …        …        …        …
1437117 -1.859707 -0.365458 -0.434808  0.508658  0.164397 -0.035316  0.022984
1437135 -1.963600 -0.271633 -0.462043  0.511119  0.134721  0.001103  0.005349
1437137 -1.728655 -0.398907 -0.391928  0.330526  0.257819 -0.046817 -0.005965
1437165  1.428814 -0.235312  0.200314 -0.457776 -1.708483 -0.492771 -0.012479
1437207 -0.114789 -0.743460 -0.260455  0.431051  0.209307  0.136855 -0.014572

[304490 rows x 7 columns]
```

3.5.3 Eboulis des valeurs propres

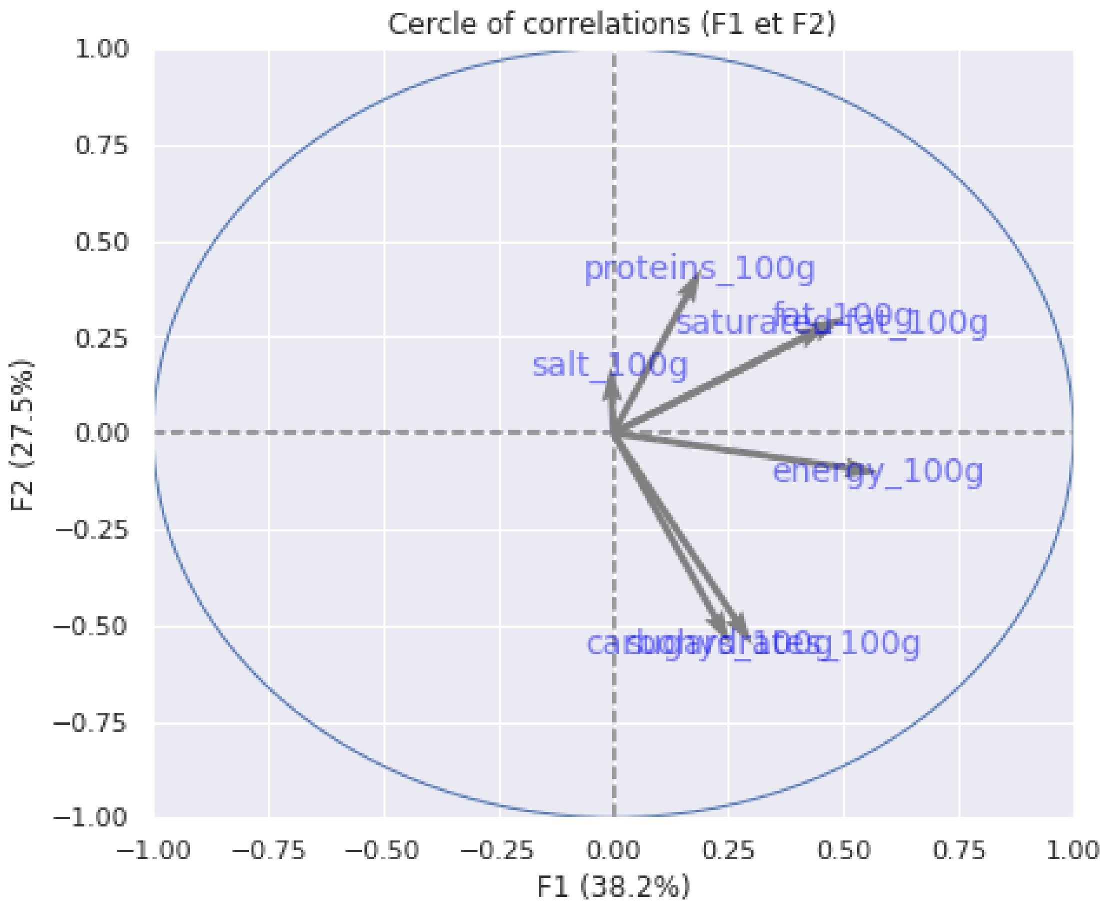
[48]:
```
##Definition de la fonction d'affichage des valeurs propres
def display_scree_plot(pca):
    scree = pca.explained_variance_ratio_*100
    plt.bar(np.arange(len(scree))+1, scree);
    plt.plot(np.arange(len(scree))+1, scree.cumsum(),c="red",marker='o');
    plt.xlabel("rank of the axis of inertia");
    plt.ylabel("percentage of inertia");
    plt.title("Decay of eigen-values");

    plt.savefig('./EigenValuesDecay.png', bbox_inches = 'tight');
    plt.show(block=False);

##Tracé de l'éboulis
display_scree_plot(pca)
```

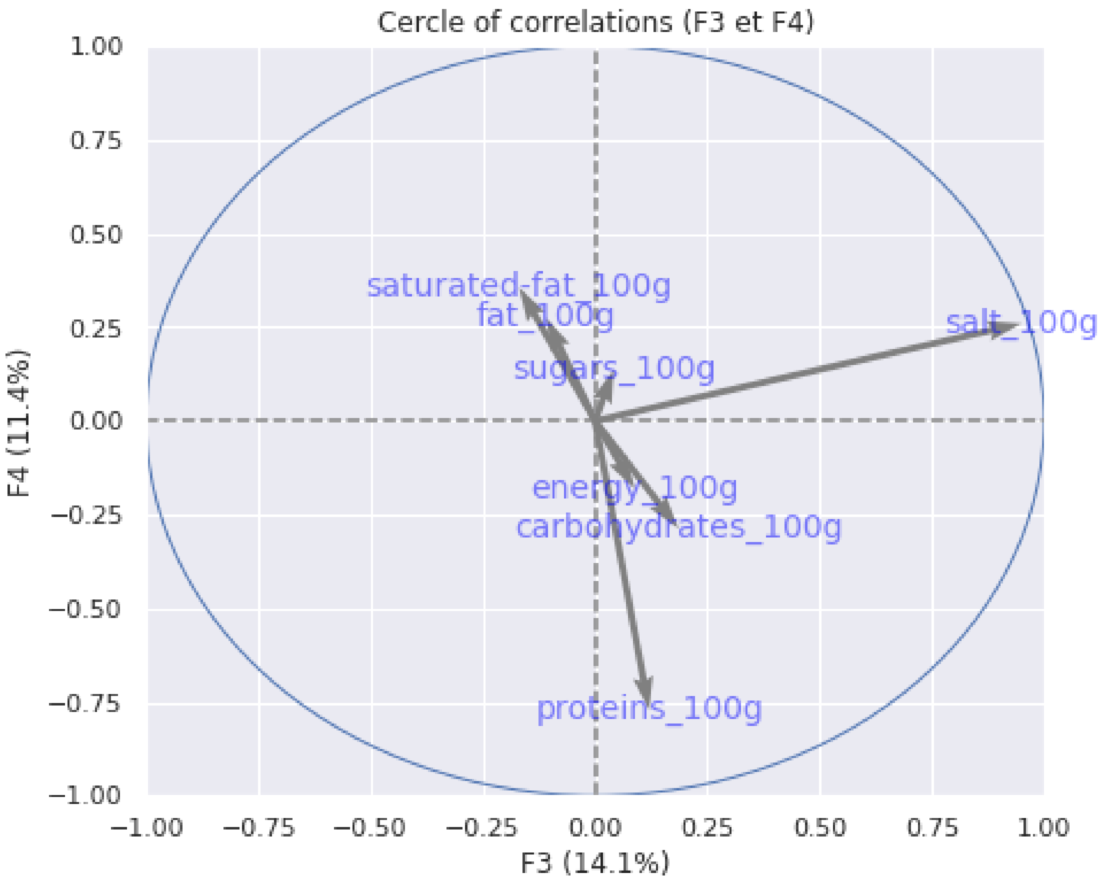


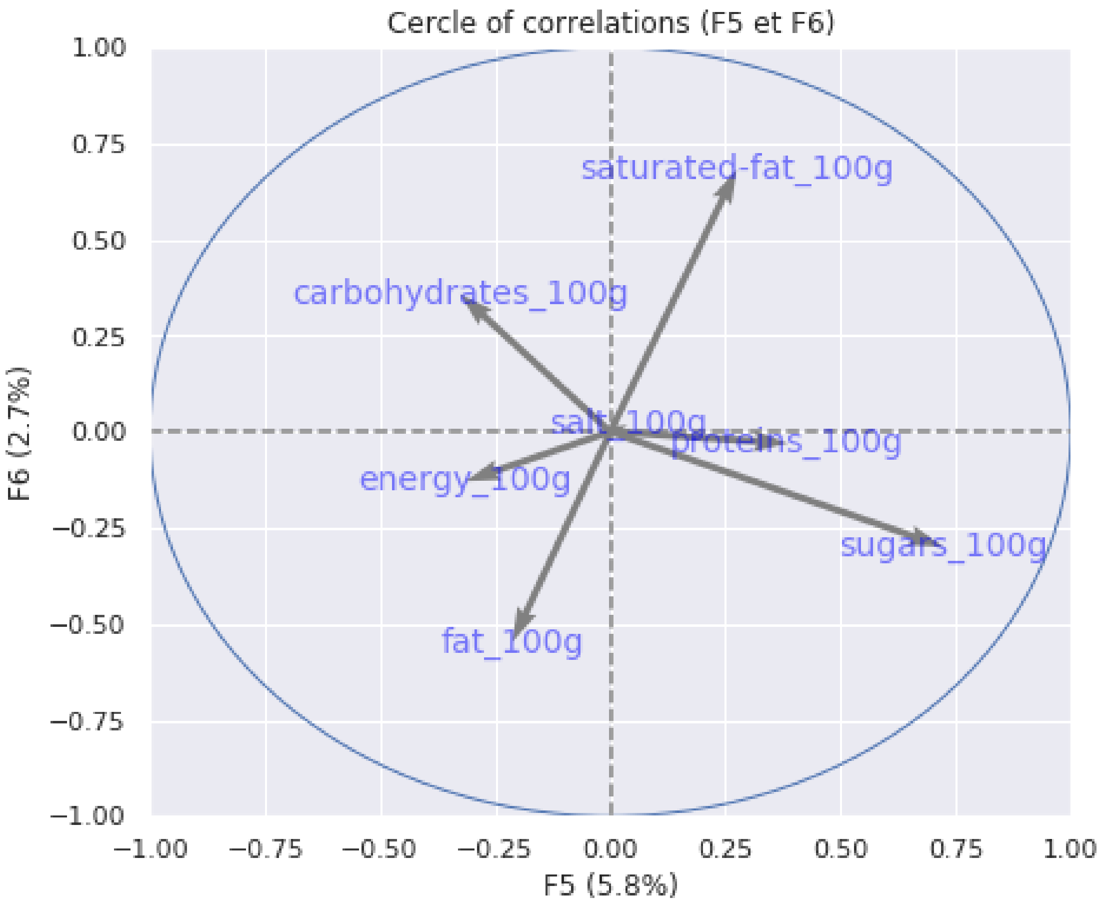
3.5.4 Cercle des corrélations

Question Pourquoi les flèches des cercles de corrélations sont elles de longueurs différentes si je travaille avce les données centrées réduites "X_scaled" ? **Réponse** Elles ont effectivement la même longueur, mais puisqu'elles sont dans une hypersphère, on n'en perçoit qu'une "projection" *en perspective* dans le plan factoriel "courant" d'observation. Elles sont donc "mal repésentées" dans ce plan.

```
[49]: ##Definition de la fonction d'affichage des valeurs propres
dic_graph={}
def display_circles(pcs, n_comp, pca, axis_ranks, labels=None,␣
 ↪label_rotation=0, lims=None):
    for d1, d2 in axis_ranks: # On affiche les 3 premiers plans factoriels,␣
 ↪donc les 6 premières composantes
        if d2 < n_comp:

            # initialisation de la figure
            fig, ax = plt.subplots(figsize=(7,6))

            # détermination des limites du graphique
            if lims is not None :
                xmin, xmax, ymin, ymax = lims
            elif pcs.shape[1] < 30 :
                xmin, xmax, ymin, ymax = -1, 1, -1, 1
            else :
                xmin, xmax, ymin, ymax = min(pcs[d1,:]), max(pcs[d1,:]),␣
↪min(pcs[d2,:]), max(pcs[d2,:])

            # affichage des flèches
            # s'il y a plus de 30 flèches, on n'affiche pas le triangle à leur␣
↪extrémité
            if pcs.shape[1] < 30 :
                plt.quiver(np.zeros(pcs.shape[1]), np.zeros(pcs.shape[1]),
                   pcs[d1,:], pcs[d2,:],
                   angles='xy', scale_units='xy', scale=1, color="grey")
                # (voir la doc : https://matplotlib.org/api/_as_gen/matplotlib.
↪pyplot.quiver.html)
            else:
                lines = [[[0,0],[x,y]] for x,y in pcs[[d1,d2]].T]
                ax.add_collection(LineCollection(lines, axes=ax, alpha=.1,␣
↪color='black'))

            # affichage des noms des variables
            if labels is not None:
                for i,(x, y) in enumerate(pcs[[d1,d2]].T):
                    if x >= xmin and x <= xmax and y >= ymin and y <= ymax :
                        plt.text(x, y, labels[i], fontsize='14', ha='center',␣
↪va='center', rotation=label_rotation, color="blue", alpha=0.5)

            # affichage du cercle
            circle = plt.Circle((0,0), 1, facecolor='none', edgecolor='b')
            plt.gca().add_artist(circle)

            # définition des limites du graphique
            plt.xlim(xmin, xmax)
            plt.ylim(ymin, ymax)

            # affichage des lignes horizontales et verticales
            plt.plot([-1, 1], [0, 0], color='grey', ls='--')
            plt.plot([0, 0], [-1, 1], color='grey', ls='--')

            # nom des axes, avec le pourcentage d'inertie expliqué
            plt.xlabel('F{} ({}%)'.format(d1+1, round(100*pca.
↪explained_variance_ratio_[d1],1)))
            plt.ylabel('F{} ({}%)'.format(d2+1, round(100*pca.
↪explained_variance_ratio_[d2],1)))

            plt.title("Cercle of correlations (F{} et F{})".format(d1+1, d2+1))
            # Enregistrement du tracé du cercle de corr de ce plan factoriel
            plt.savefig('./CercleCorr'+str(d1)+str(d2)+'.png',␣
 ↪bbox_inches='tight')


            plt.show(block=False)


##Tracé du cercle des corrélations
pcs = pca.components_
display_circles(pcs, 7, pca, [(0,1),(2,3),(4,5)], labels=nutrifacts_cols)
```

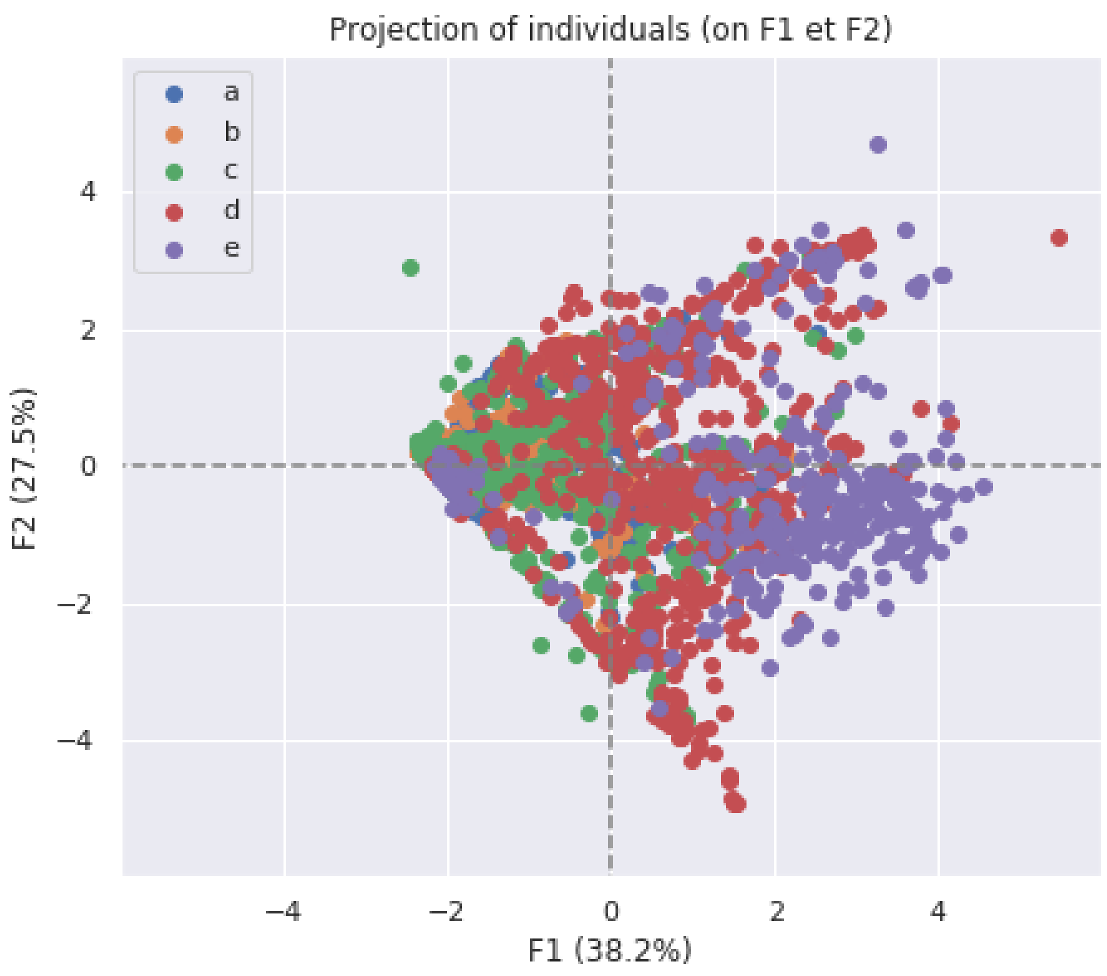



Cercle of correlations (F3 et F4)
saturated-fat_100g
fat_100g
sugars_100g
salt_100g
energy_100g
carbohydrates_100g
proteins_100g
F3 (14.1%)
F4 (11.4%)



3.5.5 Projection des individus , PAG :rajouter les couleurs avec le nutrigrade: exploiter la commande plt.scatter et rajouter/éditer une option –> OK ! (7.09.2020)

[50]:
```python
##Definition de la fonction d'affichage des valeurs propres
def display_factorial_planes(X_projected, n_comp, pca, axis_ranks, labels=None,␣
 ↪alpha=1, illustrative_var=None):
    for d1,d2 in axis_ranks:
        if d2 < n_comp:

            # initialisation de la figure
            fig = plt.figure(figsize=(7,6))

            # affichage des points
            if illustrative_var is None:
                plt.scatter(X_projected[:, d1], X_projected[:, d2], alpha=alpha)
            else:
                illustrative_var = np.array(illustrative_var)
                for value in np.unique(illustrative_var):
                    selected = np.where(illustrative_var == value)
                    plt.scatter(X_projected[selected, d1],␣
 ↪X_projected[selected, d2], alpha=alpha, label=value)
                plt.legend()

            # affichage des labels des points
            if labels is not None:
                for i,(x,y) in enumerate(X_projected[:,[d1,d2]]):
                    plt.text(x, y, labels[i],
                              fontsize='14', ha='center',va='center')

            # détermination des limites du graphique
            boundary = np.max(np.abs(X_projected[:, [d1,d2]])) * 1.1
            plt.xlim([-boundary,boundary])
            plt.ylim([-boundary,boundary])

            # affichage des lignes horizontales et verticales
            plt.plot([-100, 100], [0, 0], color='grey', ls='--')
            plt.plot([0, 0], [-100, 100], color='grey', ls='--')

            # nom des axes, avec le pourcentage d'inertie expliqué
            plt.xlabel('F{} ({}%)'.format(d1+1, round(100*pca.
 ↪explained_variance_ratio_[d1],1)))
            plt.ylabel('F{} ({}%)'.format(d2+1, round(100*pca.
 ↪explained_variance_ratio_[d2],1)))

            plt.title("Projection of individuals (on F{} et F{})".format(d1+1,␣
 ↪d2+1))

            #Enregistrement du tracé
            plt.savefig('./ProjectionOn'+str(d1)+str(d2), bbox_inches='tight')

            plt.show(block=False)


##Tracé du cercle des corrélations
pcs = pca.components_
#display_factorial_planes(X_projected, 7, pca, [(4,5)],␣
 ↪illustrative_var=food['nutriscore_grade'])
sampled = np.random.choice(range(X_projected.shape[0]), 2000, replace=False) #!
 ↪rend une liste d'indices

display_factorial_planes(X_projected[sampled,:], 7, pca, [(0,1)],␣
 ↪illustrative_var=food['nutriscore_grade'].iloc[sampled])
```



4 Tests statistiques

Question: Pour les tests stats, est-ce que j'utilise les données nutrifact de base, les données centrées réduites, ou les directions principales d'inerties ?

[51]:
```
resultat_kw= pd.DataFrame([],columns=['nutrifact','tstat','pvalue'])
```

[52]:
```
illustrative_var = np.array(food['nutriscore_grade'])

for col in nutrifacts_cols:
    dic = {}
    ind=0
    for value in np.unique(illustrative_var):
        selected = np.where(illustrative_var == value)# renders indices "where"␣
 ↪the mask applies
```

```
        dic[nutrigrade_values[ind]]=nutrifacts_scaled.iloc[selected][col]
        ind=ind+1

    print('Le résultat du test Kruskal Wallis pour ', col, ' est:
↪\n',kruskal(dic['a'].values,                                                                  ␣
↪dic['b'].values,                                                                              ␣
↪dic['c'].values,                                                                              ␣
↪dic['d'].values,                                                                              ␣
↪dic['e'].values), '\n')
    tstat_,pvalue_=  kruskal(dic['a'].values, dic['b'].values,dic['c'].
↪values,dic['d'].values,dic['e'].values)



    resultat_kw = resultat_kw.append(pd.DataFrame(data={'nutrifact':[col],
                                        'tstat':[tstat_],
                                        'pvalue':[pvalue_]}
                                 ),
                       ignore_index=True
                      )
```

```
Le résultat du test Kruskal Wallis pour  carbohydrates_100g  est:
 KruskalResult(statistic=11902.754288076138, pvalue=0.0)

Le résultat du test Kruskal Wallis pour  energy_100g  est:
 KruskalResult(statistic=86345.6875580772, pvalue=0.0)

Le résultat du test Kruskal Wallis pour  fat_100g  est:
 KruskalResult(statistic=76882.34626485115, pvalue=0.0)

Le résultat du test Kruskal Wallis pour  proteins_100g  est:
 KruskalResult(statistic=5646.813472497895, pvalue=0.0)

Le résultat du test Kruskal Wallis pour  salt_100g  est:
 KruskalResult(statistic=37006.04175075471, pvalue=0.0)

Le résultat du test Kruskal Wallis pour  saturated-fat_100g  est:
 KruskalResult(statistic=89258.4410517079, pvalue=0.0)

Le résultat du test Kruskal Wallis pour  sugars_100g  est:
 KruskalResult(statistic=40356.20629464346, pvalue=0.0)
```

```
[53]: resultat_kw.to_excel('./KruskalWallisResult.xlsx', index=False)
```

```
[54]: np.shape(X_projected)
      X_projected[sampled,:]
```

```
[54]: array([[-1.28151188e+00,  1.28734456e-01, -1.59962970e-01, …,
              -2.17468245e-01,  2.29823108e-01,  3.85928980e-02],
             [-1.74179974e+00, -5.71691400e-01, -3.99406463e-01, …,
               1.33391947e-01,  9.64885192e-04, -1.10768295e-02],
             [ 1.88837869e+00,  2.74388798e+00,  5.04565619e-01, …,
               8.81413822e-01,  1.77075619e-01,  2.26025854e-02],
             …,
             [-1.98299825e+00,  2.10489642e-01, -4.65967311e-01, …,
               6.00196986e-02, -3.00159709e-02, -5.49080246e-03],
             [ 3.30495535e+00, -1.60881835e+00, -1.94783990e-01, …,
               5.31613962e-01,  1.92455009e-01, -7.47515010e-02],
             [-2.16581594e+00, -5.08921046e-02, -5.23083996e-01, …,
               2.38133212e-02,  3.40437716e-02,  3.79191483e-03]])
```

```
[55]: np.shape(food)
```

```
[55]: (304490, 21)
```

5 Extraction d'une partie des données

```
[56]: food.iloc[0:50000].to_csv('food_extract.zip', index=False)
```

```
[ ]:
```