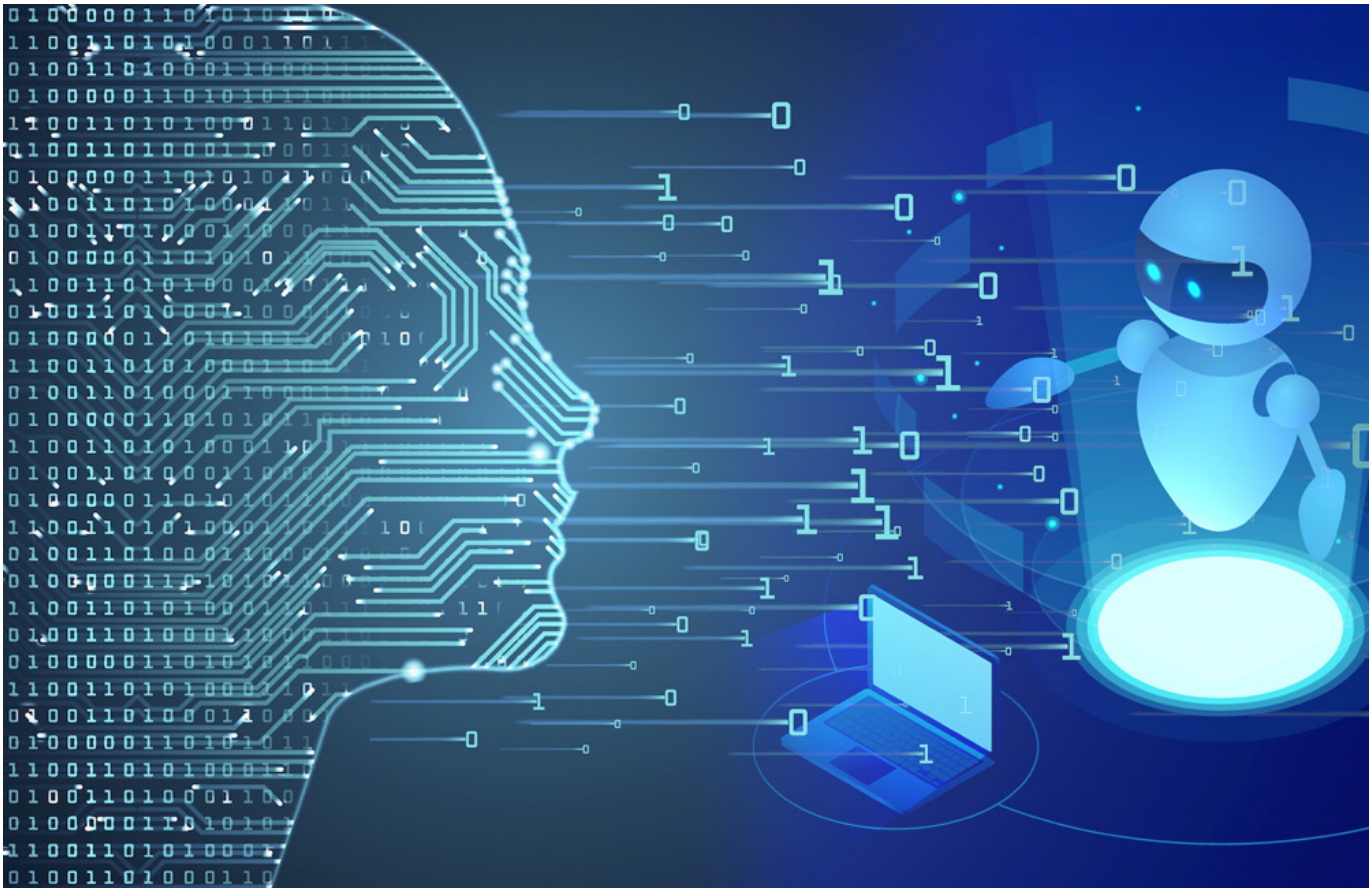
Rapport du projet NLP OpenFoodFacts



Membres de l’équipe :

- GUO Zhaojun
- DENG Muchan

Notre colab: https://colab.research.google.com/drive/1ZIwOkdDgJlNCpHtMxQXm5JF9k-k43PE8?usp=sharing

Introduction :

Nous travaillons sur OpenFoodFacts, une base de données considérée comme un Wikipédia pour la nourriture. L'objectif du projet est d'utiliser ce dataset contenant plus de 800000 produits pour de partager avec tout le monde un maximum d'informations sur les produits alimentaires.

Pour cela, nous allons dans un premier temps nettoyer le dataset pour le rendre plus exploitable, ensuite trouver une approche de clustering ainsi qu'une approche visuelle basant sur les valeurs nutritionnelles et les différentes catégories de nourriture, enfin pour terminer tenter de proposer un model permettant d'améliorer le projet OpenFoodFacts.

Data cleaning :

Dû à la taille volumineuse du dataset, nous avons fait le choix de d'abord l'importer sur Google drive pour pouvoir y accéder par la suite sur Google colab.

Nous constatons qu'il s'agit d'un ensemble de données de produits alimentaires répertoriant les ingrédients et les valeurs nutritionnelles (_100g) de plus de 300 000 aliments provenant de plus de 150 pays dans le monde. Les données sont censées être gratuites pour le public afin d'aider les utilisateurs à décoder les étiquettes des aliments et à faire de meilleurs choix alimentaires en général. L'ensemble de données contient plus de 300 000 lignes sur 163 colonnes. Cependant, comme nous le verrons, il y a beaucoup de valeurs manquantes ou manifestement incorrectes.

- Les champs(colonnes) finissant par **_t** sont des dates au format UNIX timestamp (le nombre de secondes écoulées depuis le 1er janvier 1970 00:00:00 UTC).
- Les champs finissant par **_datetime** sont des dates au format ISO 8601 (yyyy-mm-ddThh:mn:ssZ).
- Les champs finissant par **_tags** sont des listes de balises séparées par des virgules (e.g. categories_tags est l'ensemble des balises normalisés du champ categories).
- Les champs finissant par un code de langue à 2 lettres (e.g. fr pour le français) est l'ensemble des balises dans cette langue.
- Les champs finissant par **_100g** correspondent à la quantité d'un nutriment (en g, ou kJ pour l'énergie) pour 100 g ou 100 ml de produit.
- Les champs finissant par **_serving** correspondent à la quantité d'un nutriment (en g, ou kJ pour l'énergie) pour 1 portion du produit.
- nutrition-score-fr_100g : score nutritionnel expérimental dérivé du score UK FSA et adapté au marché français (ces champs sont de type numérique, allant de -15 à 40. Plus le score est bas, mieux c'est).
- nutrition-score-uk_100g : score nutritionnel défini par la UK Food Standards Administration (ces champs sont de type numérique, allant de -15 à 40. Plus le score est bas, mieux c'est).
- nutrition_grade_fr_100g : Semble être une simple catégorisation du score en A, B, C, D et E (L'équivalent britannique contient principalement des NaNs).

Nous avons repéré différents types d'erreur sur le dataset, il y a par exemple la présence de 4301 « unique » pays. Il faut alors réunir les différentes notations d'un même pays.

A cause de la taille volumineuse du dataset, nous avons décidé de travailler uniquement sur les Etats-Unis, en prenant en compte toutes les lignes dont la valeur du champ countries est égale à « United States » ou « en :us ».

```
[ ]  openfoods_us=openfoods[(openfoods['countries']=='United States')|(openfoods['countries']=='en:us')]
```

Malgré le nombre considérable de colonnes présentes dans le dataset, elles ne sont pas toutes intéressantes pour notre projet, nous allons donc travailler sur des colonnes que nous considérons suffisamment intéressant pour la suite à partir de openfoods_us.

```
[ ]  nutrition_table_cols = ["product_name","categories","ingredients_text","fat_100g", "carbohydrates_100g", "sugars_100g", "proteins_100g", "salt_1
     new_openfoods = openfoods_us[nutrition_table_cols].copy()
```

Ensuite nous supprimons les éléments répétés ainsi que les éléments vides.

```
[ ]  # We drop all duplicates from our data
     new_openfoods.drop_duplicates(inplace=True)
```

```
[ ]  new_openfoods=new_openfoods.dropna()
```

```
[ ]  len(new_openfoods)

     252265
```

Et nous obtenons notre dataset nettoyé et réduit, nous allons ainsi commencer le traitement des ingrédients à partir de ce dataset.

Traitement de texte des ingrédients :

Pour cette partie, nous effectuons le traitement de texte principalement via NLTK (Natural Language Toolkit).

Nous remarquons dans un premier temps, différents cas à traiter pour chaque ingrédient : les symboles boursiers comme $GE, les hyperliens, les caractères spécifiques, les mots répètes, la présence des chiffres (format us ou fr) suivant des mots, nous empêchent d'effectuer des opérations par la suite.

Apres avoir traiter manuellement ces cas, nous utilisons 3 modules de NLTK (corpus,stem et tokenize) pour transformer le string ingrédient en une liste de mot bien traité, pour être plus précis nous utilisons les packages:

- Stopwords du module **corpus** qui nous donne une liste de mots à partir desquels nous devons séparer, par exemple « or ».
- PorterStemmer du module **stem** qui nous permet d'effectuer de la stemming (racinisation en français) vise à garder la racine du mot, c'est à dire le tronquer de toute déclinaison, accord (flexions) et dérivations.
- WordNetLemmatizer du module **stem** qui nous permet d'effectuer de la lemmatisation, c'est à dire transformer la forme dérivée d'un mot à sa forme la plus simple, par exemple transformer cats en cat.
- TweetTokenizer du module **tokenize** qui permet de diviser une chaine de caractères en une liste de sous-chaines.

Ainsi, nous aurons a priori traité le texte des ingrédients, et nous pouvons voir quels sont les mots les plus fréquemment apparus.

```
# This was done once I had already preprocessed the ingredients
new_list=[]
for ingredients in new_ingredient:
    for i in ingredients:
      new_list.append(i)
    vocabulary.update(ingredients)
for word, frequency in vocabulary.most_common(200):
    print(f'{word};{frequency}')
```

```
salt;164693
sugar;136679
flavor;125264
water;117743
acid;115598
oil;113077
natur;110068
corn;91625
milk;79667
flour;70314
syrup;68191
citric;67288
sodium;67211
color;66312
wheat;63561
starch;59055
soy;57015
gum;56031
less;54593
powder;54033
```

Nous pouvons tout de suite voir que certains mots ne sont pas des ingrédients, par exemple « less », nous allons donc les supprimer manuellement.

```
[ ]  def ingredient_parser(ingredients):
         # measures and common words (already lemmatized)
         measures = ['fdc','ad','follow','brown','mono','high','white','dri','ad','less','green','et','red','yellow','color','blue','black']
         ingred_list = []
         for word in ingredients:
           if word not in measures:
             ingred_list.append(word)
         return ingred_list
```

```
[ ]  for i in range(len(new_ingredient)):
       new_ingredient[i]=ingredient_parser(new_ingredient[i])
```

Nous avons donc nos données nettoyées avec lesquelles nous pouvons vraiment travailler ! Mais avant de commencer, nous voulons examiner de plus près les caractéristiques individuelles. Nous savons que l'énergie contenue dans un produit peut être principalement calculée à travers sa quantité de glucides, de lipides et de protéines. Nous savons également que : 1 g de graisse contient environ 39 kJ d'énergie ; 1 g de glucides et de protéines contiennent tous deux environ 17 kJ d'énergie.

En raison du processus de saisie compliqué dans l'application OpenFoodFacts, certains utilisateurs saisissent des valeurs erronées. De plus l'unité de l'énergie est également donnée en kJ et kcal, ce qui augmente encore plus la chance que l'utilisateur se trompe lors de saisi de données.

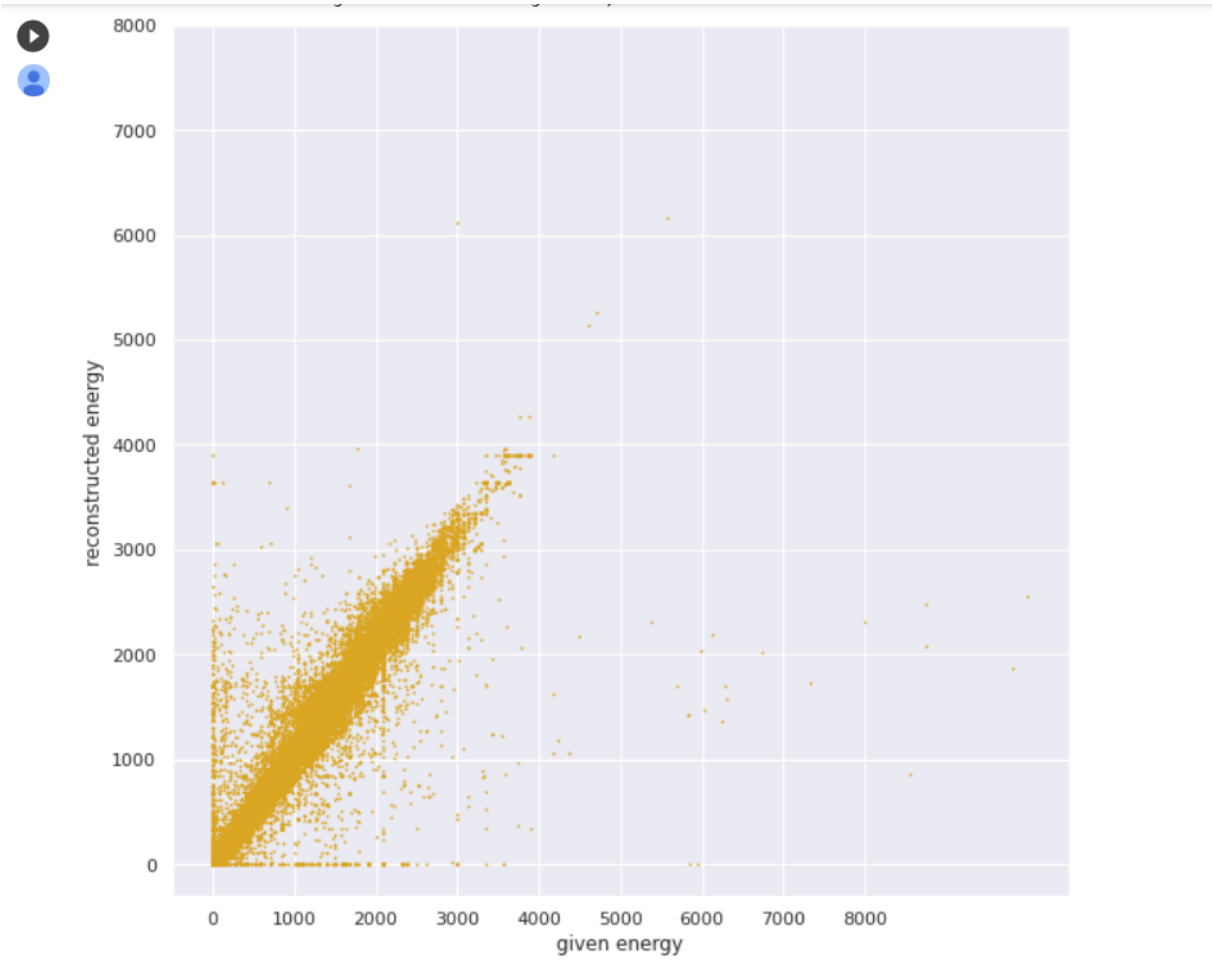
En calculant l'énergie sur la base des caractéristiques des graisses, des glucides et des protéines et en la comparant à la valeur d'énergie donnée, nous pouvons vérifier au préalable si certaines données sont incorrectes et pourrons même tenter de les corriger.

Pour ce faire, nous allons calculer la reconstructed_energy à partir de la formule ci-dessus soulignée.

```
[ ]  new_openfoods['fat_100g']=abs(new_openfoods['fat_100g'])
     new_openfoods['carbohydrates_100g']=abs(new_openfoods['carbohydrates_100g'])
     new_openfoods['sugars_100g']=abs(new_openfoods['sugars_100g'])
     new_openfoods['proteins_100g']=abs(new_openfoods['proteins_100g'])
     new_openfoods['salt_100g']=abs(new_openfoods['salt_100g'])
```

```
[ ]  new_openfoods["reconstructed_energy"] = new_openfoods["fat_100g"] * 39 + new_openfoods["carbohydrates_100g"] * 17 + new_openfoods["proteins_100g
```

Il semblerait à première vue que l'énergie reconstruite correspond presque à la quantité d'énergie donnée. Mais regardons un graphique des deux quantités d'énergie pour une meilleure conclusion :



Les points sont quasiment tous autour de la droite y=x donc nous pouvons dire que la plupart des données sont correctes. Mais les points se situant sur l'axe x et l'axe y sont clairement des erreurs à faire attention.

Par ailleurs, en dehors des erreurs que nous avons détectées grâce à notre énergie reconstruite, nous pouvons également facilement détecter une autre erreur : nos caractéristiques en matières grasses, glucides et protéines sont données sur la base de 100g. Donc, si la somme de ces caractéristiques est supérieure à 100, nous saurons également qu'il y a quelque chose qui ne va pas avec nos données. Voyons donc si nous pouvons trouver l'une de ces erreurs :

```
[ ] new_openfoods["g_sum"] = new_openfoods.fat_100g + new_openfoods.carbohydrates_100g + new_openfoods.proteins_100g

    new_openfoods["exceeded"] = np.where(new_openfoods.g_sum.values > 100, 1, 0)
    new_openfoods[new_openfoods["exceeded"] == 1].head()
```

	product_name	categories	ingredients_text	fat_100g	carbohydrates_100g	sugars_100g	proteins_100g	salt_100g	energy_100g	recons
3393	Meiji, hello panda, choco biscuits with choco ...	Snacks, Sweet snacks, Biscuits and cakes, Bisc...	[wheat, oil, cream, salt, yeast, mass, butter,...	26.67	66.67	26.67	6.67	0.8325	2230.0	
3395	Biscuit	Snacks, Sweet snacks, Biscuits and cakes, Bisc...	[leav, malt, natur, wheat, oil, salt, bean, ci...	26.67	66.67	26.67	6.67	0.9175	2230.0	
3396	Biscuits	Snacks, Sweet snacks, Biscuits and cakes, Bisc...	[malt, natur, wheat, oil, salt, skim, canola, ...	26.67	66.67	33.33	6.67	0.9175	2230.0	
3662	Peanut butter cup optimal low-carb ketogenic n...	Snacks, Sweet snacks, Confectioneries	[natur, oil, salt, vanilla, butter, isol, coat...	47.06	41.18	5.88	11.76	0.2200	2213.0	
3663	Peanut butter cup fat bomb snack	Snacks	[natur, oil, salt, vanilla, butter, isol, coat...	47.06	41.18	5.88	11.76	0.2200	2213.0	

```
[ ] new_openfoods.exceeded.value_counts()

    0    250841
    1      1420
    Name: exceeded, dtype: int64
```

Il y a donc a priori 1420 produits dont la somme des ingrédients 100g dépasse les 100g, on les exclue bien évidamment par la suite.

```
[ ] new_openfoods=new_openfoods.drop(index=exceeded.index)
```

Clustering :

```
[ ] new_openfoods.categories.value_counts()

Snacks
Snacks, Sweet snacks, Confectioneries
Groceries, Sauces
Dairies, Fermented foods, Fermented milk products, Cheeses
Desserts, Frozen foods, Frozen desserts

Plant-based foods and beverages, Plant-based foods, Snacks, Sandwiches, Meat analogues, Veggie patties, Veggie burgers
Groceries,Sauces,Pestos,Green pestos
Frozen foods, Frozen-fried-chicken
Plant-based foods and beverages, Plant-based foods, Fruits and vegetables based foods, Spreads, Breakfasts, Fruits based foods, Plant-based sprea
Plant-based foods and beverages, Plant-based foods, Nuts and their products, Nuts, Roasted nuts
Name: categories, Length: 4340, dtype: int64
```

On trouve que les colonnes de catégories assez désordonnées pour pouvoir utiliser plus tard, nous avons donc décidé de faire un peu de nettoyage manuel.

```
[ ] x=new_openfoods.apply(lambda x: change_product_name(x['categories']), axis=1)

[ ] new_openfoods.categories=x

[ ] new_openfoods=new_openfoods[new_openfoods.categories!='Other']
    new_openfoods.head()
```

	product_name	categories	ingredients_text	fat_100g	carbohydrates_100g	sugars_100g	proteins_100g	salt_100g	energy_100g	reconstructed_e
939	Piasten, Chocolate Assortment	Snack	[wheat, oil, min, humect, mass, butter, citric...	26.25	58.75	6.5	8.75	0.0300	2029.0	2
948	Nestle, dark truffles grand chocolate	Snack	[sunflow, alkali, butter, miner, sorbitol, mil...	35.00	50.00	35.0	5.00	0.1250	1987.0	2
953	Mt. olive, sweet 'n' hot salad peppers	Snack	[banana, natur, salt, calcium, vinegar, benoat...	0.00	32.14	25.0	0.00	1.6075	598.0	
980	Ritter sport, knusperflakes mit knusprigen cor...	Snack	[mass, malt, lactos, natur, cream, salt, cocoa...	28.00	60.00	49.6	8.00	0.4000	2209.0	2
990	Funsch, High Quality Marzipan	Snack	[almond, carmin, invert, food, water, sorbitol...	21.40	55.20	51.6	9.60	0.2500	1929.0	1

Nous voilà maintenant avec des valeurs de catégories composé d’un seul mot.

Nous avons choisi le model de clustering KMeans, et nous l'appliquons directement en utilisant le package KMeans de la bibliothèque sklearn.

Le paramètre est donc choisi pour être égale a 11, le nombre d'espèce de catégorie après le nettoyage de données précédemment effectué.

Et comme nous pouvons le voir, l'algorithme de prédiction donne un chiffre correspondant a une catégorie de nourriture.

```
[ ] features = ["fat_100g", "carbohydrates_100g", "sugars_100g", "proteins_100g", "salt_100g", "energy_100g", "reconstructed_energy", "g_sum"]
```

```
[ ] import matplotlib.pyplot as plt
    import numpy as np
    from sklearn.cluster import KMeans
    #from sklearn import datasets
    from sklearn.datasets import load_iris

    X_train = new_openfoods[features].values

    model = KMeans(n_clusters=11,random_state=42)
    model.fit(X_train)
    results = new_openfoods.copy()
    results["cluster"] = model.predict(X_train)
```

```
results.head(10)
```

ingredients_text	fat_100g	carbohydrates_100g	sugars_100g	proteins_100g	salt_100g	energy_100g	reconstructed_energy	g_sum	exceeded	cluster
[wheat, oil, min, humect, mass, butter, citric...	26.25	58.75	6.50	8.75	0.0300	2029.0	2171.25	93.75	0	2
[sunflow, alkali, butter, miner, sorbitol, mil...	35.00	50.00	35.00	5.00	0.1250	1987.0	2300.00	90.00	0	2
[banana, natur, salt, calcium, vinegar, benoat...	0.00	32.14	25.00	0.00	1.6075	598.0	546.38	32.14	0	5

Examinons maintenant le résultat :

```
cluster_10=results[results.cluster==10]
cluster_10.head(10)
```

	product_name	categories	ingredients_text	fat_100g	carbohydrates_100g	sugars_100g	proteins_100g	salt_100g	energy_100g	reconstructed_
993	Best Sweet-Potato Cookies	Snack	[sweet, proprietari, cranberri, bake, butter, ...	17.65	50.00	17.65	2.94	0.3675	1623.0	
994	Best Ginger Snap Cookies	Snack	[ginger, salt, molass, butter, flour, judy, gr...	17.65	50.00	17.65	2.94	0.3675	1623.0	
1274	Candy crush, sugar crush crunchy and crackling...	Snack	[natur, caramin, carbon, wax, candi, flavor, m...	0.00	100.00	90.00	0.00	0.0250	1674.0	
1310	Welch's, golden apple chips	Snack	[fresh, appl, preserv, sulphit, sodium]	0.00	92.86	78.57	0.00	1.6075	1644.0	
1312	Green apple chips	Snack	[natur, appl, flavor]	0.00	92.86	75.00	0.00	0.0000	1644.0	
1447	Brownie mix	Snack	[soybean, alkali, cottonse, folic, wheat, iron...	3.85	82.05	46.15	7.69	0.9625	1611.0	

Nous constatons que pour les nourritures dont la prédiction est 10, leur catégorie (snack) correspond la plupart du temps a ce nombre.

```
[ ] results.cluster.value_counts()
```

```
10    18690
7     14848
0     14697
6     14278
2     13756
1     12217
8     11021
3     10881
9      7191
5      7059
4      1753
Name: cluster, dtype: int64
```

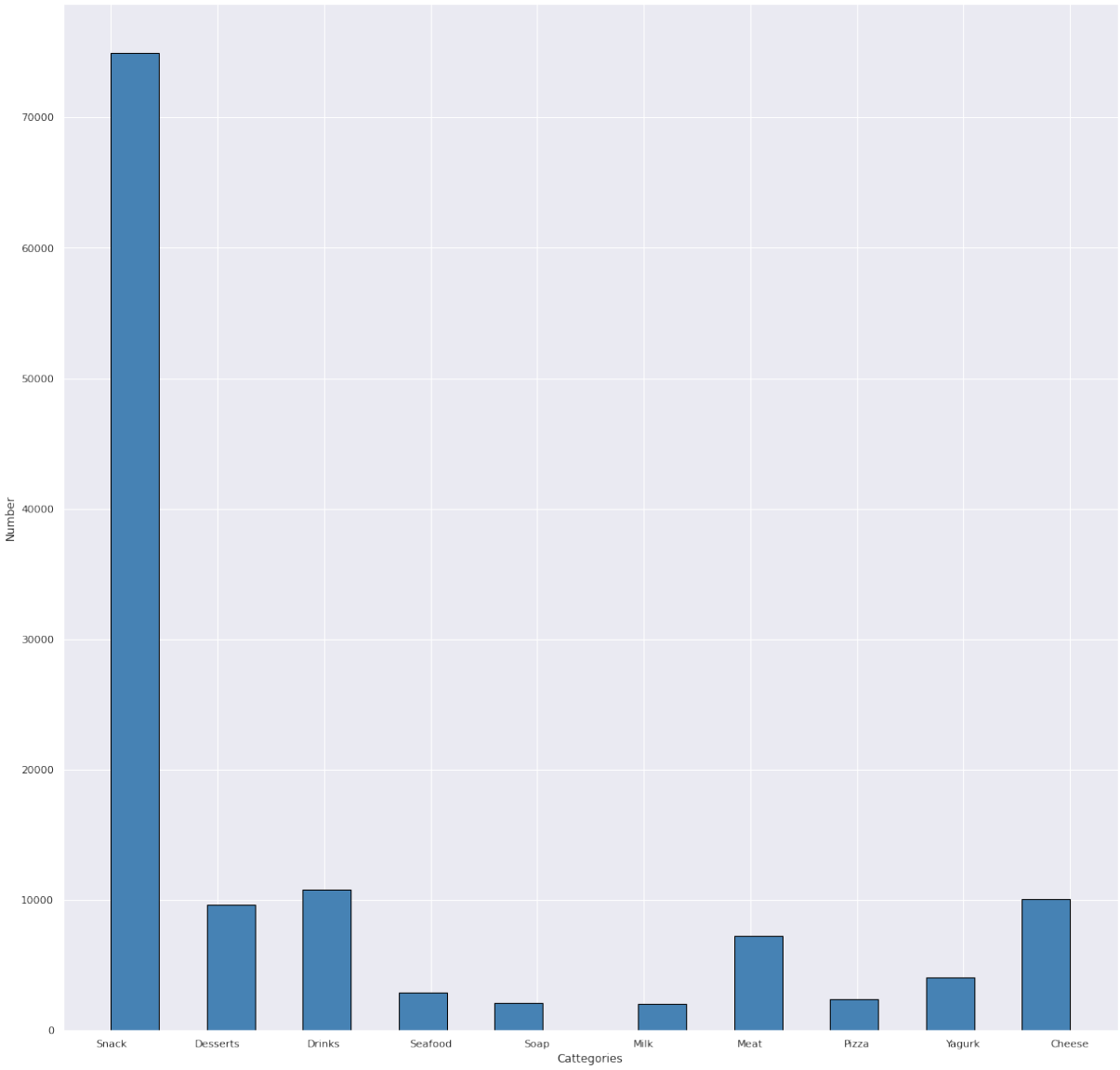
```
[ ] cluster_10.categories.value_counts()
```

```
Snack       14514
Cheese       3332
Meat          307
Drinks        238
Desserts      180
Soap           89
Milk           18
Pizza           5
Seafood         5
Yagurk          2
Name: categories, dtype: int64
```

Nous pouvons voir que parmi les 18690 cluster_10 prédis, 14514 d'entre eux sont réellement des Snack, nous avons donc un taux de prédiction de 0,77

La prédiction est donc a priori bonne.

Visualisation :



Enfin nous avons une visualisation de notre dataset en termes d'effectif de chaque catégorie.

Propose un Model

Pour finir, nous pensons qu'il est possible d'utiliser Word2vec pour effectuer du dataprocessing dans le futur puisque nous avons déjà les ingrédients traités précédemment.

Et basant sur nos travaux sur les catégories (les 11 labels), utiliser RNN ou LSTM en les important de la bibliothèque tensorflow.