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

Mentor : Morgan MOISON
Etudiant : Marin DUCHEMIN

OPENCLASSROOMS

Plan de la Présentation

I ~ Présentation de l’idée d’application

II ~ Préparation des données

III ~ Analyse exploratoire effectuée

IV ~ Faisabilité de l’application

I ~ Présentation de l’idée d’application

Générer un panier de provisions alimentaires pour une semaine.

Basée sur l’idée du «Panier Confinement » de certaines grandes enseignes commerciales

I ~ Présentation de l'idée d'application

Proposer un choix de types de panier différents selon :

- Ses préférences
- Sa localisation

Basée sur la base de données

OpenFoodFact

I ~ Présentation de l’idée d’application

Exemple de panier de confinement (Carrefour)

2 briques de **Jus d'orange avec pulpe pur jus**, Carrefour, 1L

2 paquets de **Biscottes froment**, Cracotte, 250g

1 paquet de **Céréales trésor chocolat noisette**, Kellogg's, 375g

1 pot de **Confiture de fraise**, Carrefour, 370g

1 pot de **Pâte à tartiner noisettes et cacao**, Nutella, 400g

1 sachet de **Café soluble**, Nescafe, 100g

1 boite de **Thé vert menthe**, Carrefour, 25 sachets

...

Définition des catégories suite à cet exemple

Jus de Fruits, Tartines, Céréales, Laits, Confitures et Pâtes à Tartiner, Cafés, Thés, Biscuits Sucrés, Desserts, Soupes, Pates, Riz, Fruits, Légumes, Sources de Protéines, Sauces, Biscuits Apéritifs, Matières Grasses

I ~ Présentation de l’idée d’application

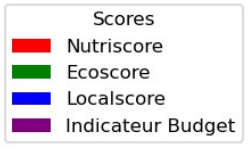
Construction des Profils :

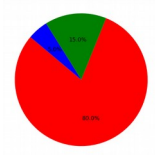
- Choix des indicateurs d’intérêt :
  - Récupérés dans la base de données :
    - le Nutriscore (allant de -15 à +40, classé en lettre de A à E),
    - L’Ecoscore (allant de 0 à +100, classé en lettre de A à E),
  - Créés par la suite :
    - un Localscore (qui sera créé en fonction de l’emplacement du consommateur),
    - un Indicateur Budget (un filtre selon les marques de distribution pour sélectionner les produits moins onéreux)

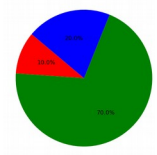
I ~ Présentation de l’idée d’application

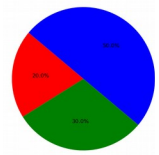
Construction des Profils :

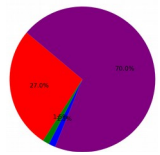
- Choix de différents profils à partir de nos indicateurs correspondant à différents besoins (nutritionnels, écologiques, régionaux ou économiques)











II ~ Préparation des données

Jeu de données très conséquent impliquant :

- Importation au sein de la base de données NoSQL MongoDB

- Filtres successifs pour réduire aux produits d'intérêt :
  - Au départ : 1 601 557 produits.
  - Appel de Santé Publique France donc réduction aux produits vendus en France : 425 015 *(filtre de requête MongoDB)*
  - Pour le localscore, besoin de connaître le lieu de manufacture : 193 747 *(filtre de requête MongoDB)*
  - Produits avec données utilisables : 44 214 *(filtre RegEx sur les noms, lieux de manufacture, codes barres)*

II ~ Préparation des données

Plusieurs étapes d'uniformisation des données

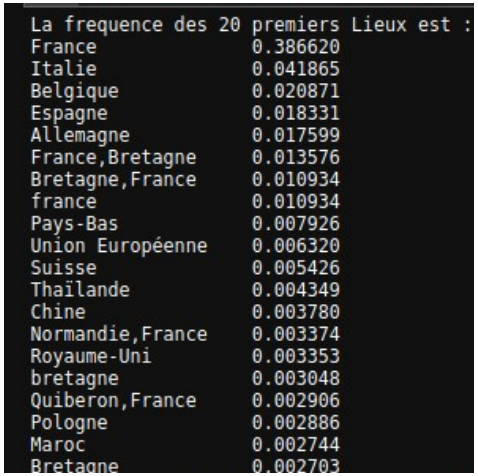
- Noms des produits
- Noms des lieux
- etc

```python
# Uniformisation de la colonne ecoscore_data

for i in range(len(name_df)):
    if type(name_df.loc[i,'ecoscore_data']) == dict:
        if name_df.loc[i,'ecoscore_data']['agribalyse']:
            name_df.loc[i,'ecoscore_data'] = name_df.loc[i,'ecoscore_data']['agribalyse']['name_fr']
        else:
            name_df.loc[i,'ecoscore_data'] = np.nan
    else:
        continue
```

```python
# Uniformisation des noms

for i in range(len(name_df)) :
    if type(name_df.loc[i,'generic_name_fr']) == str and len(name_df.loc[i,'generic_name_fr']) == 0:
        name_df.loc[i,'generic_name_fr'] = np.nan
    elif type(name_df.loc[i,'generic_name']) == str and len(name_df.loc[i,'generic_name']) == 0:
        name_df.loc[i,'generic_name'] = np.nan
    elif type(name_df.loc[i,'product_name_fr']) == str and len(name_df.loc[i,'product_name_fr']) == 0:
        name_df.loc[i,'product_name_fr'] = np.nan
    elif type(name_df.loc[i,'product_name']) == str and len(name_df.loc[i,'product_name']) == 0:
        name_df.loc[i,'product_name'] = np.nan
    elif type(name_df.loc[i,'ecoscore_data']) == str and len(name_df.loc[i,'ecoscore_data']) == 0:
        name_df.loc[i,'ecoscore_data'] = np.nan
    else:
        continue
```

II ~ Préparation des données

Enrichissement avec des coordonnées GPS :



```
La frequence des 20 premiers Lieux est :
France              0.386620
Italie              0.041865
Belgique            0.020871
Espagne             0.018331
Allemagne           0.017599
France,Bretagne     0.013576
Bretagne,France     0.010934
france              0.010934
Pays-Bas            0.007926
Union Européenne    0.006320
Suisse              0.005426
Thaïlande           0.004349
Chine               0.003780
Normandie,France    0.003374
Royaume-Uni         0.003353
bretagne            0.003048
Quiberon,France     0.002906
Pologne             0.002886
Maroc               0.002744
Bretagne            0.002703
```

Première étape de normalisation des lieux de manufacture

(n'avoir que des Pays et, pour la France, des Régions, Villes ou Code Postaux)

Recherche des coordonnées via des bases de données disponibles sur le site du gouvernement et Kaggle

- https://www.data.gouv.fr/fr/datasets/regions-departements-villes-et-villages-de-france-et-doutre-mer/
- https://www.kaggle.com/paultimothymooney/latitude-and-longitude-for-every-country-and-state

II ~ Préparation des données

Etape de standardisation des lieux de manufacture :

```
# Actualisation du tableau

for i in range(len(coordonnees_df)):
# Analyse de la cellule
    if coordonnees_df.loc[i,'Pays'][0] in ['france'] and len(coordonnees_df.loc[i,'Pays']) == 1:
        coordonnees_df.loc[i,'Pays'] = 'FRANCE'
    else:
        test = [j for j in coordonnees_df.loc[i,'Pays'] if(j not in ['france'])]

# Tri des Pays
        pays = [k for k in paysList if (k in test)]
        pays = list(set(s for s in pays if len(s) == max(len(s) for s in pays)))
        if len(pays) != 0:
            coordonnees_df.loc[i,'Pays'] = pays[0].upper()

#Tri des Regions
        else:
            region = [k for k in regionList if (k in test)]
            region = list(set(s for s in region if len(s) == max(len(s) for s in region)))
            if len(region) != 0:
                coordonnees_df.loc[i,'Pays'] = 'FRANCE'
                coordonnees_df.loc[i,'Region'] = region[0]

#Tri des Codes Postaux
            else:
                codePostal = [k for k in codePostalList if (k in test)]
                codePostal = list(set(s for s in codePostal if len(s) == max(len(s) for s in codePostal)))
                if len(codePostal) != 0:
                    coordonnees_df.loc[i,'Pays'] = 'FRANCE'
                    coordonnees_df.loc[i,'Region'] = codePostal[0]

#Tri des Villes
                else:
                    ville = [k for k in villeList if (k in test)]
                    ville = list(set(s for s in ville if len(s) == max(len(s) for s in ville)))
                    if len(ville) != 0:
                        coordonnees_df.loc[i,'Pays'] = 'FRANCE'
                        coordonnees_df.loc[i,'Region'] = ville[0]
                    else:
                        if np.vectorize(lambda x: bool(re.compile(r'.*france.*').match(x)))(coordonnees_df.loc[i,'Pays']).any():
                            coordonnees_df.loc[i,'Pays'] = 'FRANCE'
```

Suivie d’une étape d’uniformisation pour n’avoir que des pays ou des regions

II ~ Préparation des données

Comment associer les catégories aux données:

- Sélection de mots-clés correspondant aux catégories définies *(diapo 5)*

```
# On liste le type d'ingredients pouvant aller dans le panier de confinement

jusList = ["jus d'orange", 'jus de pomme', 'jus multifruit', "jus d'ananas", 'jus de goyave']
tartineList = ['pain de mie', 'biscotte', 'pain tranche']
cerealeList = ['cereales', 'petales de ble', 'petales', 'muesli', 'petales']
laitList = ['lait entier', 'lait demi-ecreme', 'lait-ecreme', 'lait de coco', "lait d'amande", 'lait de riz']
tartinableList = ['confiture', 'pate a tartiner']
cafeList = ['cafe']
theList = ['the']
biscuitList = ['cookie', 'biscuit sec chocolate', 'biscuit sec petit beurre', 'biscuit sec feuillete', 'sable']
dessertList = ['compote', 'yaourt', 'dessert']
soupeList = ['soupe']
pateList = ['pates', 'pates fraiches', 'pates seches']
rizList = ['riz basmati', 'riz thai', 'riz gluant', 'riz melange', 'riz complet', 'riz blanc']
fruitList = ['orange', 'pomme, crue', 'poire', 'banane, crue', 'ananas, cru', 'peche', 'abricot', 'cerise', 'framboise', 'mure', 'prune']
legumeList = ['carotte', 'patate', 'oignon', 'haricot', 'mais', 'champignon', 'tomate', 'concombre', 'epinard']
proteineList = ['poulet', 'boeuf', 'jambon','sardine', 'thon', 'colin', 'oeuf']
sauceList = ['pesto', 'sauce tomate', 'ketchup', 'mayonnaise']
aperitifList = ['cacahuete', 'pistache', 'chips', 'noix de cajou']
matiereList = ['beurre doux', 'beurre demi-sel', 'beurre demi sel', 'beurre sale', 'creme fraiche', "huile d'olive", 'huile de colza', 'huile de tournesol']

```

- *Recherche de ces mots-clés parmis les noms des produits*

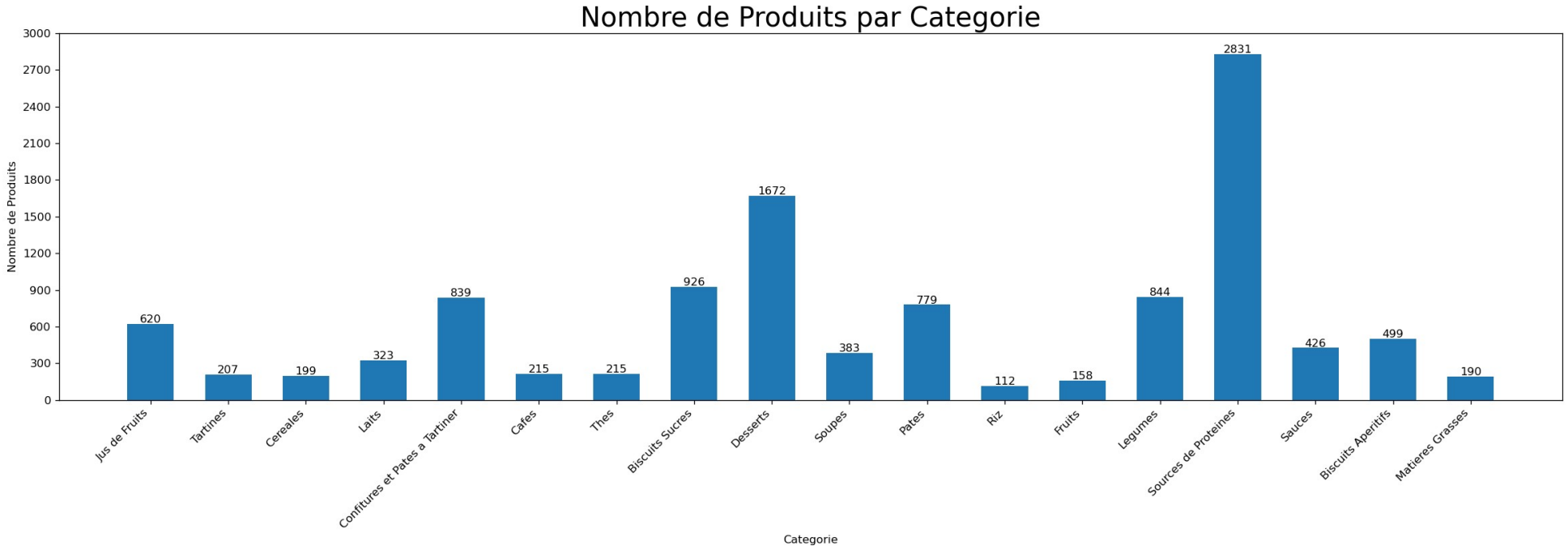
II ~ Préparation des données

- Construction des catégories en jeux de données :

	Code Barre	Nom du Produit	Pays de Manufacture	Region de Manufacture	Latitude GPS	Longitude GPS	Nutriscore	Ecoscore	Marque de Distribution
0	25110001525	Huile d'olive vierge extra	Italie	NaN	41.871940	12.567380	6.0	49.531594	Non
1	319110413922	Huile d'olive vierge extra	France	NaN	NaN	NaN	NaN	65.766726	Non
2	637913429156	Huile de colza	France	Hauts-de-France	50.614407	3.118079	NaN	47.450168	Non
3	764549058679	Huile d'olive vierge extra	Espagne	NaN	40.463667	-3.749220	6.0	64.472459	Non
4	11972464	Huile d'olive vierge extra	Italie	NaN	41.871940	12.567380	NaN	41.481594	Non

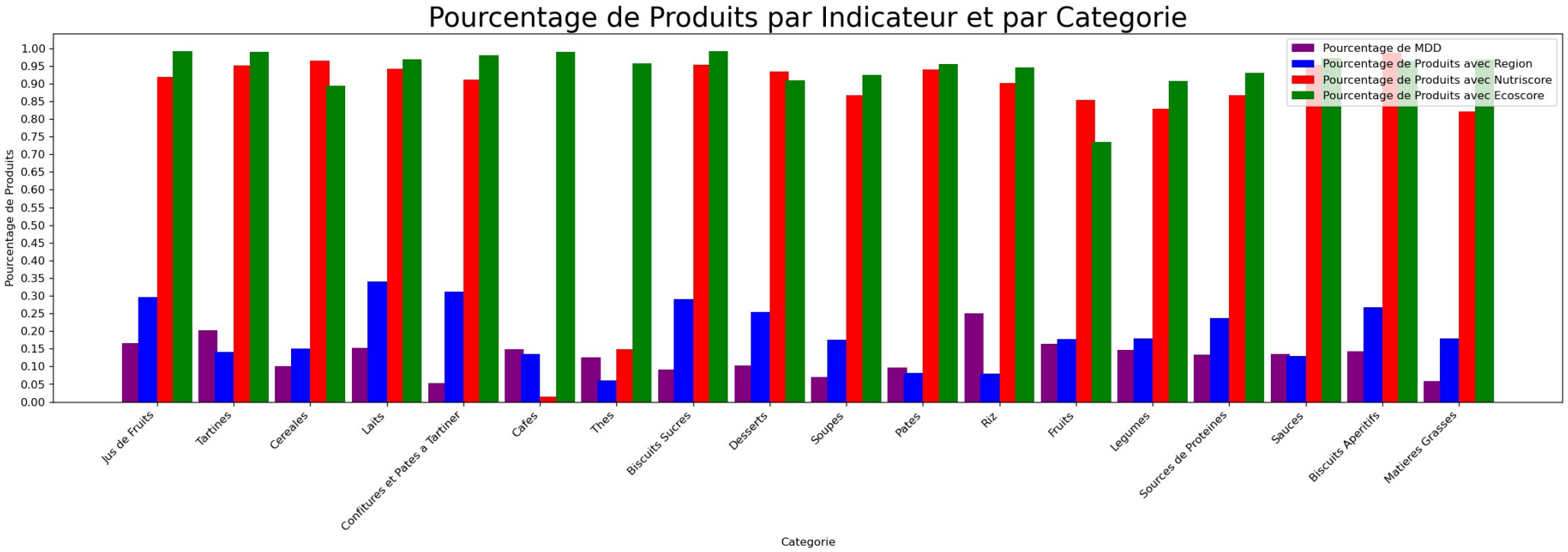
- Enregistrement sous format csv pour les réutiliser dans les autres NoteBooks.

III ~ Analyse exploratoire effectuée



Nombre total de produits conservés : 11 438

III ~ Analyse exploratoire effectuée



III ~ Analyse exploratoire effectuée

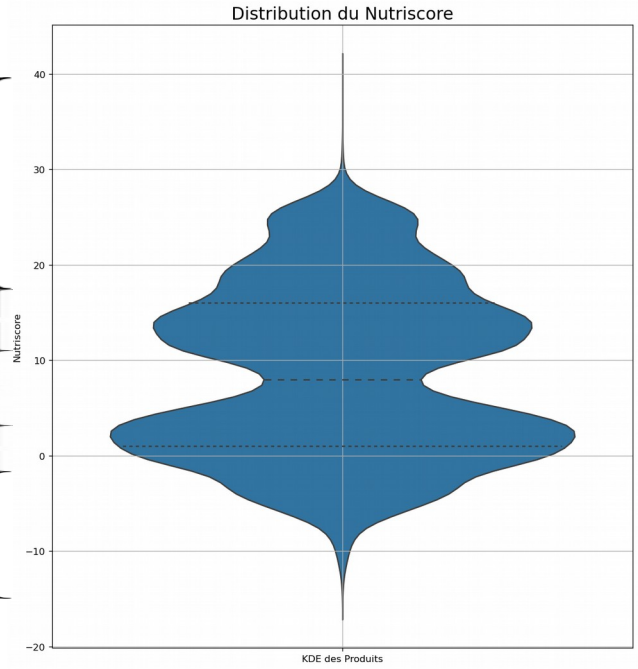
+17 à +40 NUTRI-SCORE E

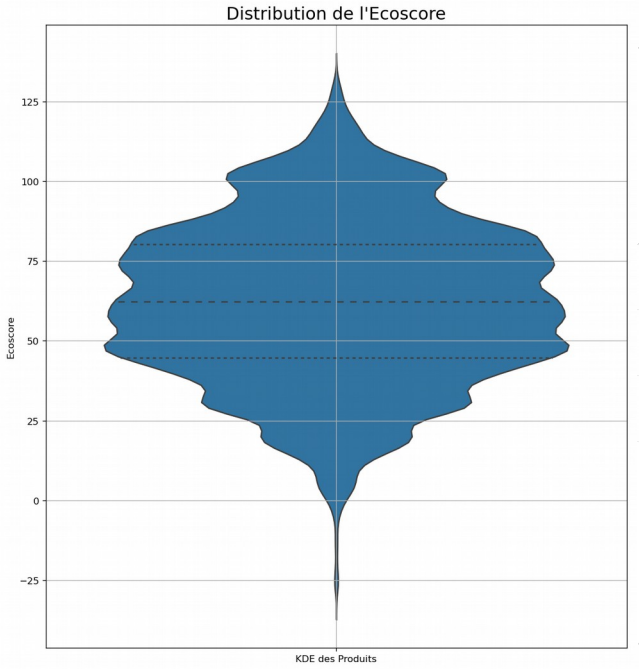
+12 à +16 NUTRI-SCORE D

+4 à +11 NUTRI-SCORE C

-1 à +3 NUTRI-SCORE B

-15 à -2 NUTRI-SCORE A





ECO-SCORE A >80

ECO-SCORE B +60 à +80

ECO-SCORE C +40 à +60

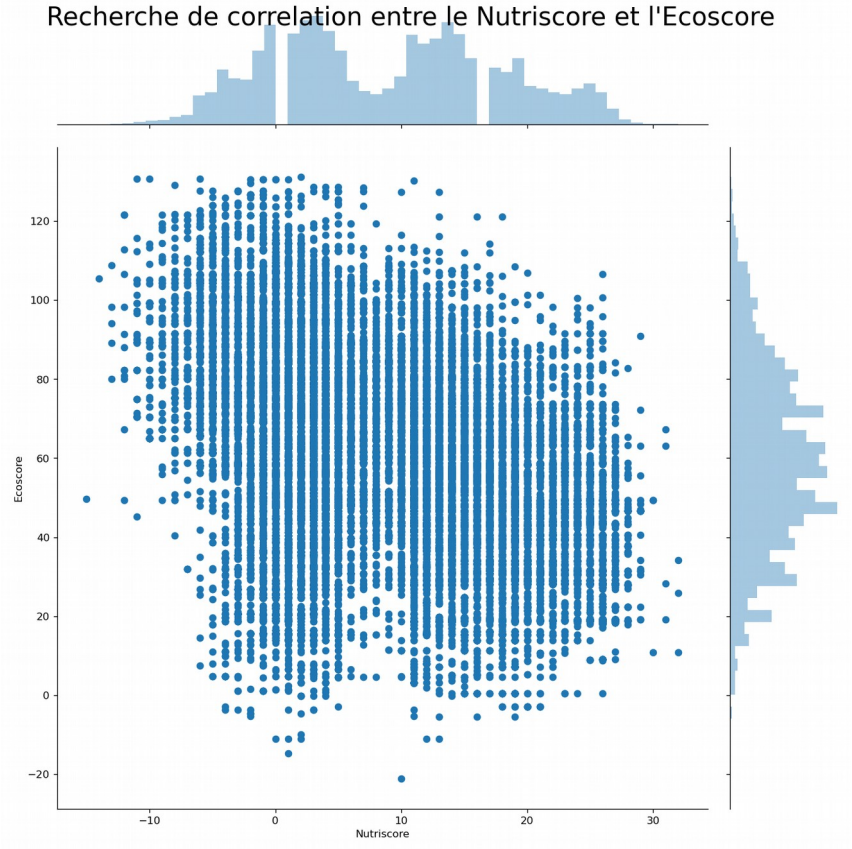
ECO-SCORE D +20 à +40

ECO-SCORE E <20

Distribution bimodale du Nutriscore

Distribution monomodale de l'Ecoscore

III ~ Analyse exploratoire effectuée



Aucune corrélation visible entre ces deux scores.

III ~ Analyse exploratoire effectuée

Synthèse après la partie exploratoire:

- Données suffisantes et uniformes pour exploiter les indicateurs
- Catégories établies, créées et fonctionnelles
- Indicateurs récupérés non-corrélés et exploitables et
  Indicateurs non récupérés prêt à être dynamiquement créés

Le projet d’application paraît donc faisable.

IV ~ Faisabilité de l’application

Mise en pratique pour tester un prototype d’application.

IV ~ Faisabilité de l'application

Fonction pour trouver le score de proximité

```python
# Fonction pour trouver la region la plus lointaine (DOM TOM exclus)

def find_further_region(region):
    distanceList = []

    for item in regionList:
        distLatCarre = np.square(latitudeDict[item] - latitudeDict[region])
        distLongCarre = np.square(longitudeDict[item] - longitudeDict[region])
        distanceList.append(np.sqrt(distLatCarre + distLongCarre))

    proximiteDict = dict(zip(regionList,distanceList))

    global furtherRegion
    furtherRegion = (list(proximiteDict.keys())[list(proximiteDict.values()).index(max(proximiteDict.values()))], max(proximiteDict.values()))
    return furtherRegion
```

```python
# Fonction pour construire le localscore en fonction de la region ou on se trouve

def find_localscore(region, categorie):
    categorie['Localscore'] = ''
    for i in range(len(categorie)):
# Si on a l'information de la region ou l'information du pays (autre que la France)
        if type(categorie.loc[i,'Region de Manufacture']) == str or categorie.loc[i,'Pays de Manufacture'] not in ['France']:
            distLatCarre = np.square(categorie.loc[i,'Latitude GPS'] - latitudeDict[region])
            distLongCarre = np.square(categorie.loc[i,'Longitude GPS'] - longitudeDict[region])
            categorie.loc[i,'Localscore'] = (np.sqrt(distLatCarre + distLongCarre))

# Si le produit est manufacture en France
        else:
            categorie.loc[i,'Localscore'] = find_further_region(region)[1]

```

IV ~ Faisabilité de l'application

Classement des produits selon le profil choisi

```python
# Fonction pour classer les produits selon le profil

def find_rank(profil, categorie):
    profilRatio = [profilDict[profil]['Nutriscore'], profilDict[profil]['Ecoscore'], profilDict[profil]['Localscore']]

# Pour ne pas avoir de nutriscore negatif ce qui nous embeterait pour le calcul du ratio
    categorie['Nutriscore']= categorie['Nutriscore'].add(abs(categorie['Nutriscore'].min()))

# Creation du nouveau score global
    for i in range(len(categorie)):
        if categorie.loc[i,'Nutriscore'] == 0:
            scoreNut = 1
        else:
            scoreNut = categorie['Nutriscore'].min() / categorie.loc[i,'Nutriscore']

        scoreEco = categorie.loc[i,'Ecoscore'] / categorie['Ecoscore'].max()

        if categorie.loc[i,'Localscore'] == 0:
            scoreLoc = 1
        else:
            scoreLoc = categorie['Localscore'].min() / categorie.loc[i,'Localscore']

        categorie.loc[i,'Score pour le Profil'] =  np.average([scoreNut, scoreEco, scoreLoc], weights=profilRatio)


# Reorganisation du jeu de donnees
    if profil in ['Petit Prix']:
        categorie.sort_values(['Marque de Distribution', 'Score pour le Profil'], ascending = False, ignore_index = True, inplace = True)
    else:
        categorie.sort_values(['Score pour le Profil'], ascending = False, ignore_index = True, inplace = True)
```

IV ~ Faisabilité de l'application

Création d'un panier selon le profil et la région

```
1  # Fonction pour faire un panier
2
3  def create_basket(region, profil):
4
5      for cat in categ_df_list:
6          find_localscore(region, cat)
7          find_rank(profil, cat)
8
9      global basket_df
10     basket_df = pd.DataFrame(columns = list(categ_df_list[0].columns))
11
12     for i in range(len(categ_df_list)):
13         basket_df.loc[i] = dict(categ_df_list[i].loc[0])
14
15
```

IV ~ Faisabilité de l’application

Test en direct parce que c’est plus sympa.

Merci de votre Attention !