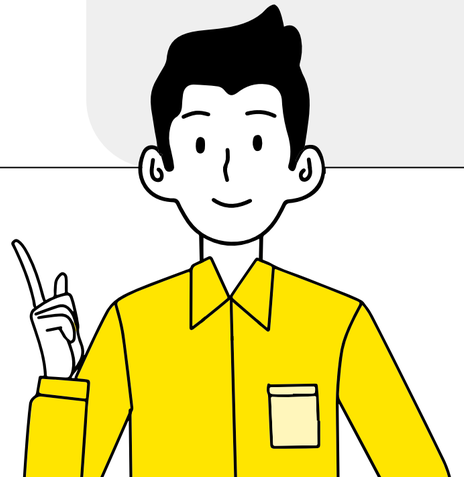
Santé publique France

Parcours
AI Engineer

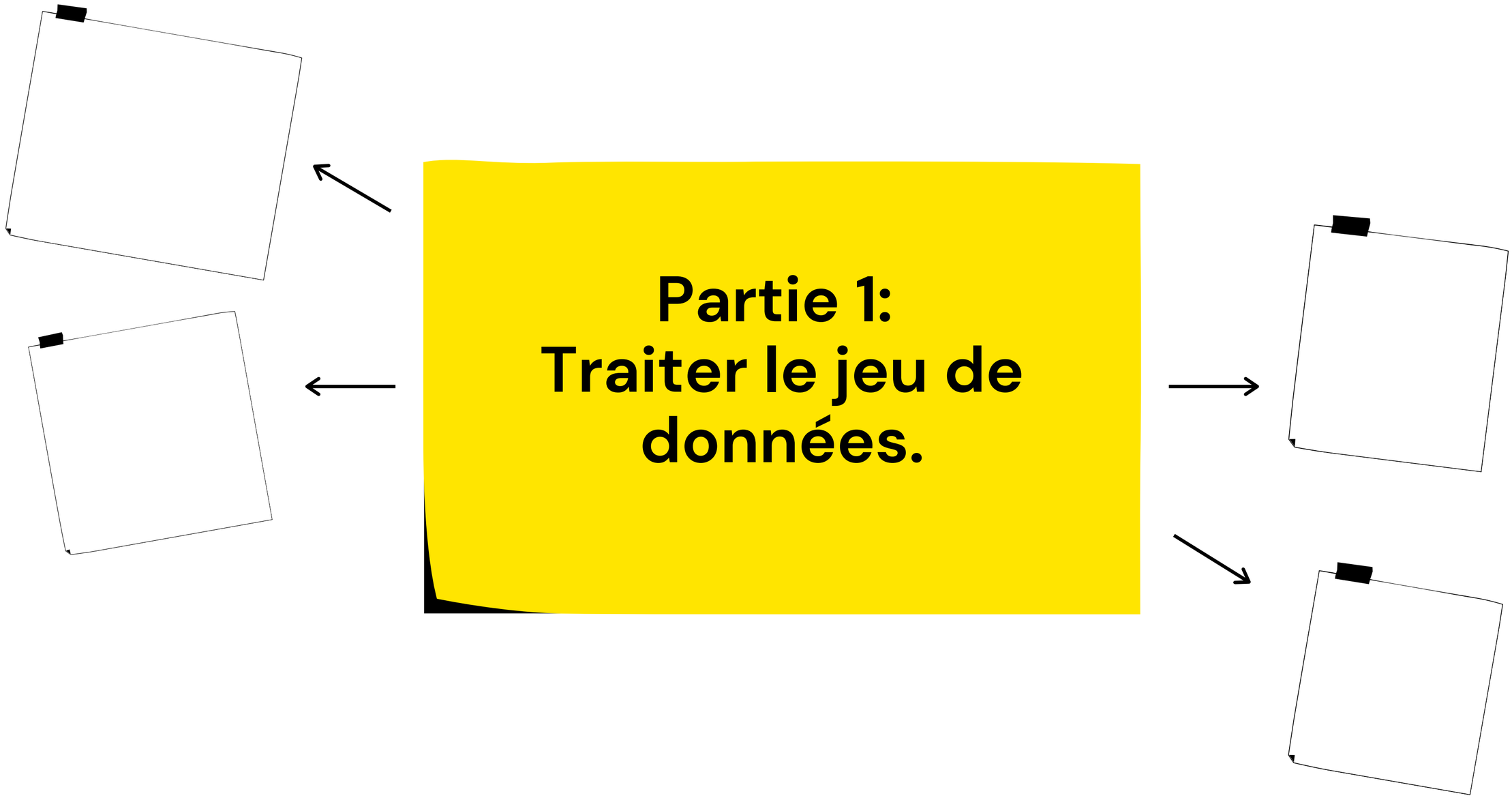
Notre mission:

Nettoyer et explorer les données pour la faisabilité d’une application.



Déroulement

1. Traiter le jeu de données.
2. Observation -tendance-
3. Observation -Corrélation-
4. Rapport d’exploration.



Partie 1:
Traiter le jeu de données.

Partie 1: Traiter le jeu de données.

Source externe

- Saisie manuelle
- Plusieurs unités de mesure

Partie 1: Traiter le jeu de données.

Jeu de données

- 320772 rows x 162 columns
- International
- Nombreuse colonnes ‘doublons’
- Nombreuse colonnes pour 100g’

Piste d’analyse

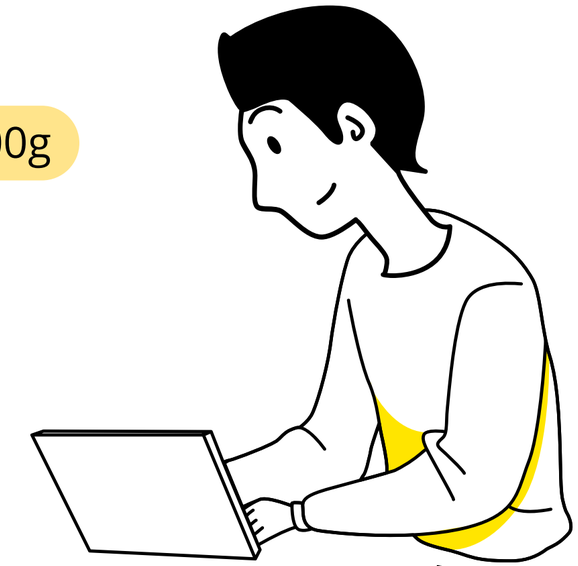
nutrition_grade_fr

allergens

additives

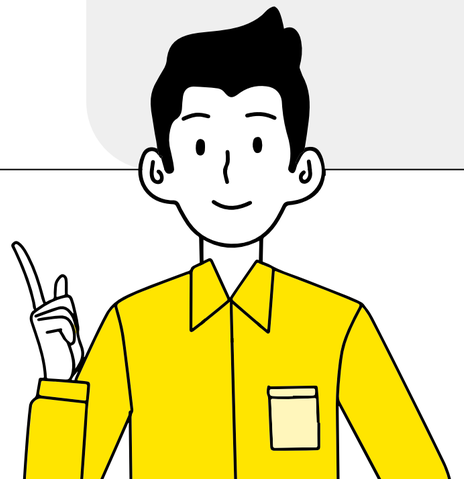
carbon-footprint-100g

countries



Question.

Il y a-t-il un lien entre le Nutri Score et les allergènes ?



Partie 1: Traiter le jeu de données.

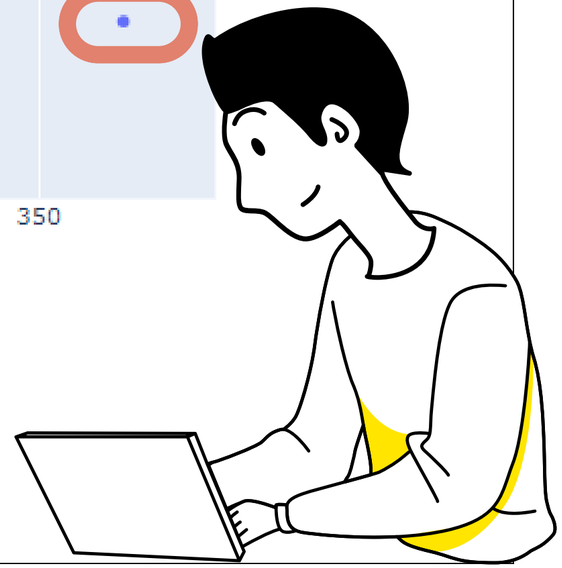
Principe RGPD

1. Licéité, loyauté et transparence
2. Limitation des finalités
3. Minimisation des données
4. Exactitude des données
5. Limitation de la conservation

Partie 1: Traiter le jeu de données.

Nettoyage et complétions

Exemple de variable pour 100g



Partie 1: Traiter le jeu de données.

Nettoyage et complétions

Solution pour les variables “pour 100g”

```
# liste de mots-clés que vous souhaitez rechercher dans les noms de colonnes
keywords = ['_100g']

# Parcourez les colonnes du DataFrame
for col_name in data.filter(like='_100g'):
    # Appliquez la transformation
    data[col_name] = data[col_name].apply(lambda x: x / 1000 if x > 100 else 0 if x < 0 else x)
```

Suppression

```
#Suppression des variables à 100% de Val Manquante
data = data.dropna(axis=1, how='all')
```

Partie 1: Traiter le jeu de données.

Nettoyage

Doublons via 'code'

```
l]: dataFrance.duplicated('code').sum()
```

```
l]: 4
```

```
Traitement des doublons :
    -On va dans un premier temps fusionner les doublons pour remplacer un maximum de valeur manquante
    -Ensuite on conserve :  -1 Celui qui possède déjà un résultat nutriscore.
                            -2 Ou l'individu qui a le plus de valeur à la fin de la fusion
                            -3 La date de création la plus récente.
```

```
# Parcourez le DataFrame et effectuez la fusion conditionnelle
for index, row in dataFrance.iterrows():
    if row['code'] == 722810:
        # Récupérez l'indice de l'autre individu avec le même 'code'
        other_index = dataFrance[(dataFrance['code'] == 722810) & (dataFrance.index != index)].index

        if len(other_index) > 0:
            other_index = other_index[0]  # Prenez le premier indice trouvé

            # Parcourez les colonnes et remplacez les valeurs manquantes
            for column in dataFrance.columns:
                if pd.notna(row[column]) and pd.isna(dataFrance.at[other_index, column]):
                    dataFrance.at[other_index, column] = row[column]
```

Partie 1: Traiter le jeu de données.

Liste de nettoyage rapide

- **Simplification** des informations sur **l'huile de palme**
- **Suppression** des lignes **sans le nom du produit**
- **Suppression** des lignes **de boissons alcoolisées**
- **Fusion** des colonnes sur le **même sujet**
- **Suppression** des colonnes après **fusion** et **obsolètes**
- **Création** de la variable **DataFrance**

Partie 1: Traiter le jeu de données.

Calcul de Nutri-Score avec du machine learning -Préparation-

1 - Pré-remplissage

```
for col in dataFrance.columns:
     #Si le type de la colonne est un objet (chaîne de caractères), remplacer les valeurs manquantes par 'Abs'
    if dataFrance[col].dtype == 'O':  # 'O' est le code pour le type d'objet
        dataFrance[col] = dataFrance[col].fillna('Abs')
     #Si le type de la colonne est un entier ou un flottant, remplacer les valeurs manquantes par 0
    if dataFrance[col].dtype in ['int64', 'float64']:  # 'int64' et 'float64' sont les types d'entier et de flottant
        dataFrance[col] = dataFrance[col].fillna(0)
```

2 - Numérisation

```
def convert_nutrition_grade(grade):
    mapping = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
    return mapping.get(grade, None)  # Retourne None si la note n'est pas trouvée

# Appliquer la fonction de conversion à la colonne 'nutrition_grade_fr'
dataFrance['nutrition_grade_fr'] = dataFrance['nutrition_grade_fr'].apply(convert_nutrition_grade)
```

3 - Création de variable

```
# Séparation du jeu de données en deux parties
dataFrance_with_y = dataFrance.dropna(subset=['nutrition_grade_fr'])
dataFrance_without_y = dataFrance[dataFrance['nutrition_grade_fr'].isna()]

# Définition de X_train et y_train
X_train = dataFrance_with_y[['salt_100g', 'proteins_100g', 'fiber_100g', 'sugars_100g', 'carbohydrates_100g', 'saturated-f
y_train = dataFrance_with_y['nutrition_grade_fr']
```

Partie 1: Traiter le jeu de données.

Calcul de Nutri-Score avec du machine learning -Application-

1 - Utilisation d'un modèle

2 - Entrainement et application

3 - Harmonise et valide

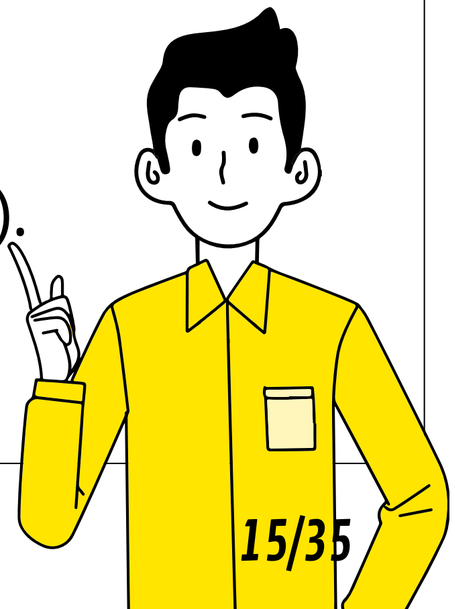
```python
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression

# Création de l'imputer
imputer = SimpleImputer(strategy='mean')  # Vous pouvez choisir 'median' ou une autre stratégie si nécessaire

# Application de l'imputation sur X_train
X_train_imputed = imputer.fit_transform(X_train)

# Entraînement du modèle sur les données imputées
model = LinearRegression()
model.fit(X_train_imputed, y_train)

# Utiliser le modèle pour prédire les valeurs manquantes de y
X_missing = dataFrance_without_y[['salt_100g', 'proteins_100g', 'fiber_100g', 'sugars_100g', 'carbohydrates_100g', 'saturated-fat_100g', 'fat_100g', 'en
X_missing_imputed = imputer.transform(X_missing)  # Utiliser le même imputer pour garantir la cohérence
predicted_y = model.predict(X_missing_imputed)

# Utiliser le modèle pour prédire les valeurs manquantes de y
predicted_y = model.predict(X_missing_imputed)

# Arrondir les prédictions au chiffre supérieur et appliquer les contraintes de plage
predicted_y_rounded = np.ceil(predicted_y)
predicted_y_constrained = np.clip(predicted_y_rounded, 1, 5)

# Ajouter les prédictions ajustées à l'ensemble de données sans y
dataFrance_without_y['nutrition_grade_fr'] = predicted_y_constrained

# Combiner les deux parties du jeu de données
dataFrance = pd.concat([dataFrance_with_y, dataFrance_without_y])
```

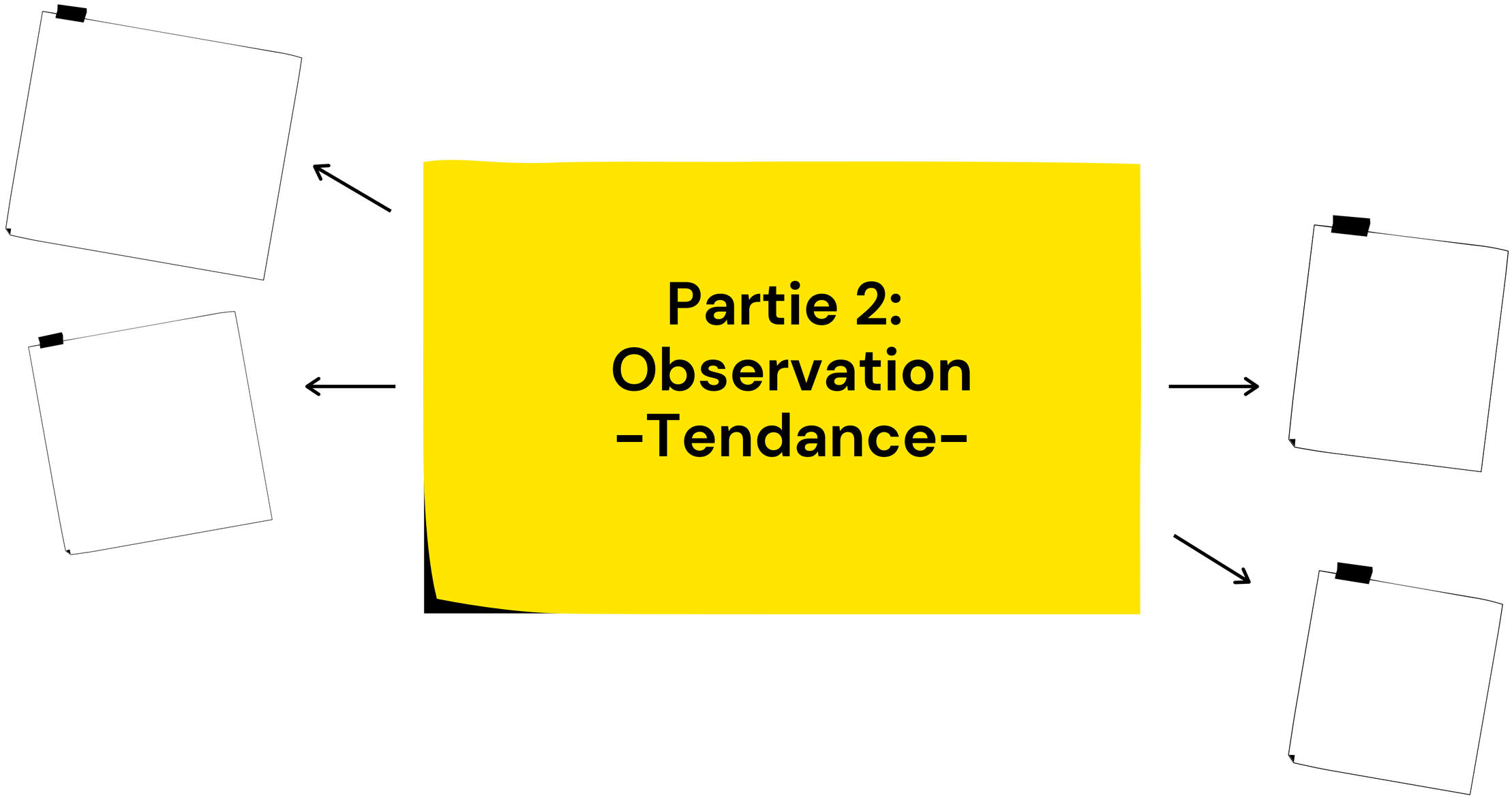
Partie 1: Traiter le jeu de données.

Travail sur les allergènes

Objectif : Savoir le nombre d'allergènes d'un produit.

- **Mettre en minuscule.**
- **Supprimer les espaces.**
- **Remplacer** les termes **inappropriés :** 'creme' → 'lait'**.**
- **Supprimer** les termes **non remplaçable :** 'couscous' **.**
- **Supprimer les répétitions.**
- **Standardiser l'énumération** → **', '** (virgule + espace).
- **Création** de **Nbr_Allergènes** et **calcul.**
- **Supprimer** les lignes avec **+ de 15.**

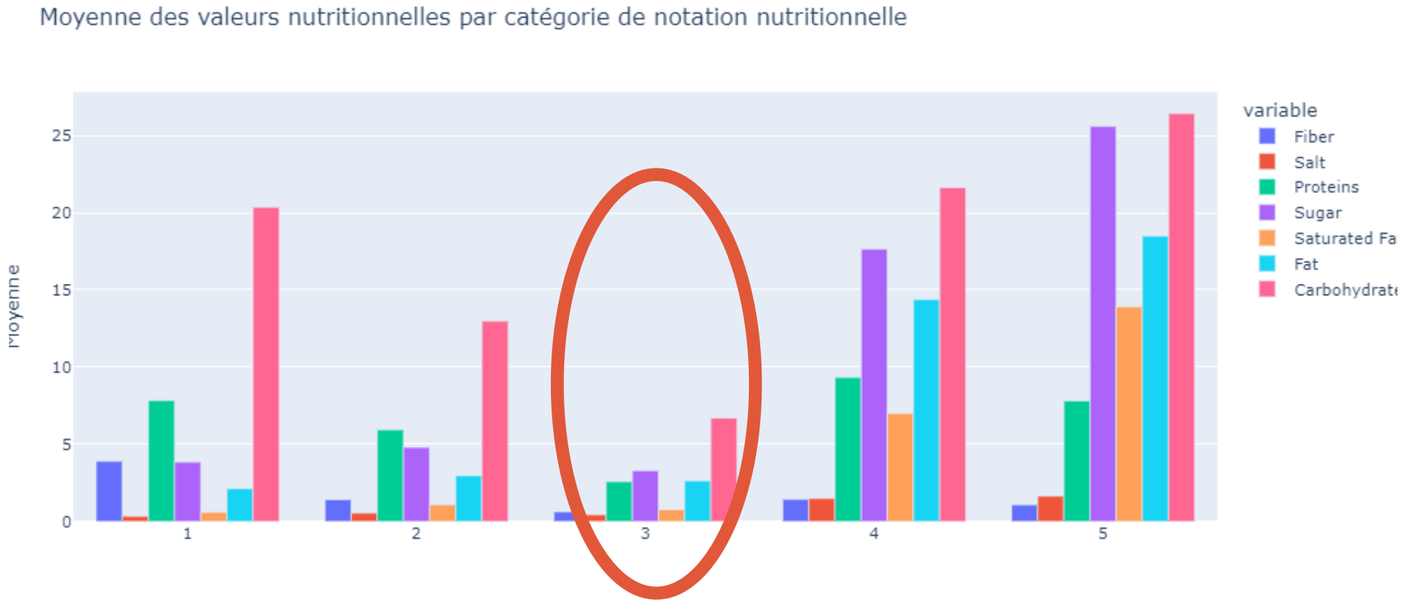




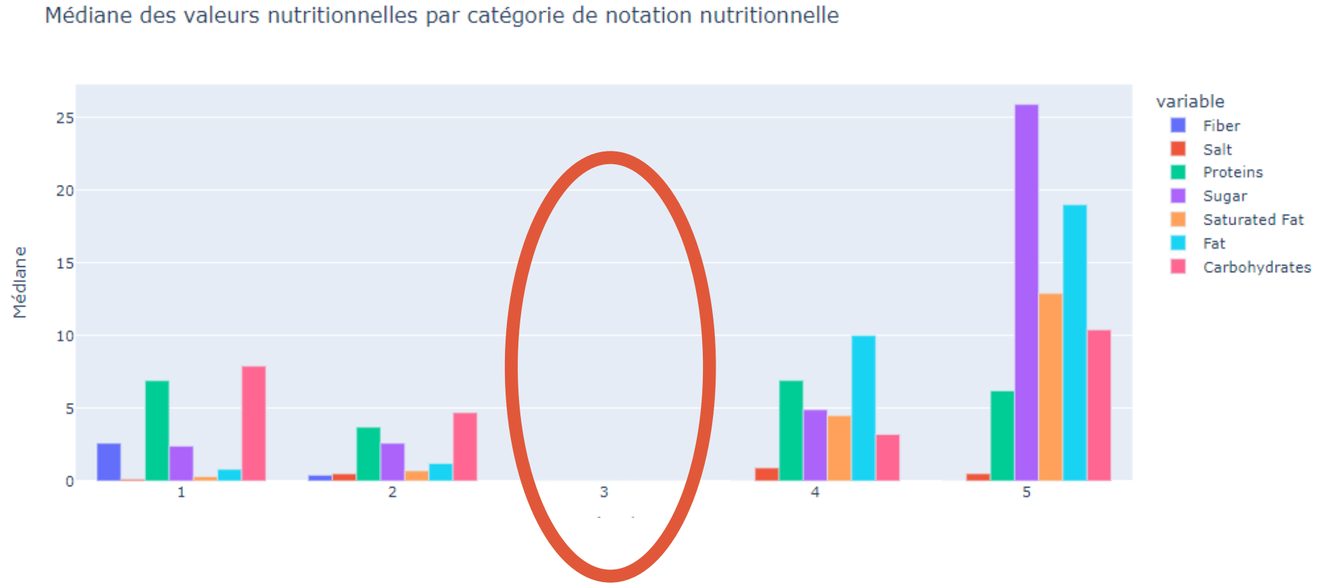
Partie 2:
Observation
-Tendance-

Partie 2: Observation –Tendance–

Moyennes ou Médianes ?



Moyenne



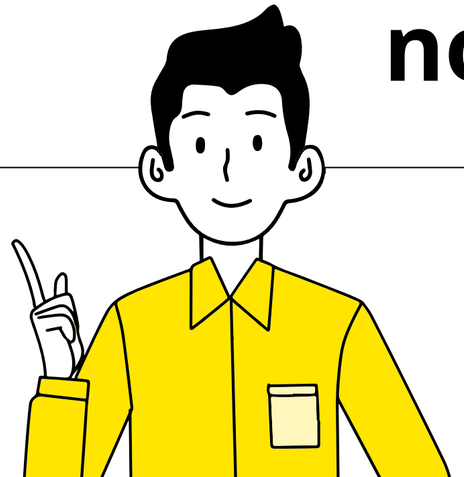
Médiane

La situation

Nous avons tous les produits vendu en France

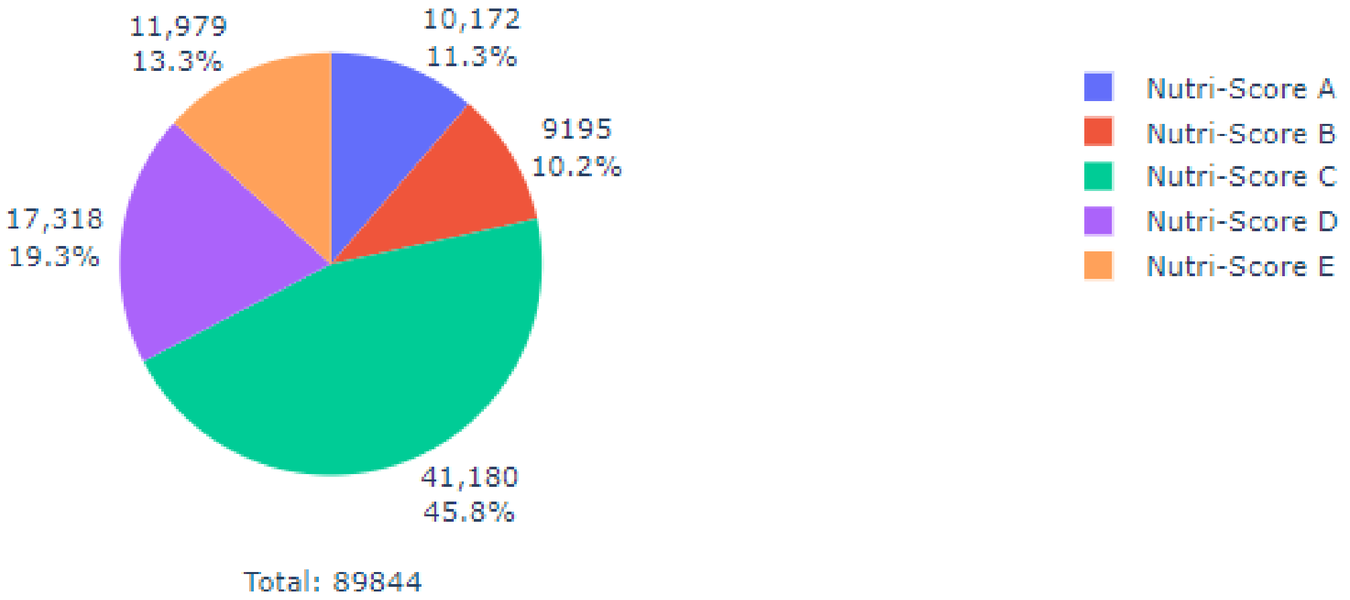
Nous avons calculé tous les Nutri-Scores

Nous avons connaissons les allergènes et leurs nombres pour chaque produits.



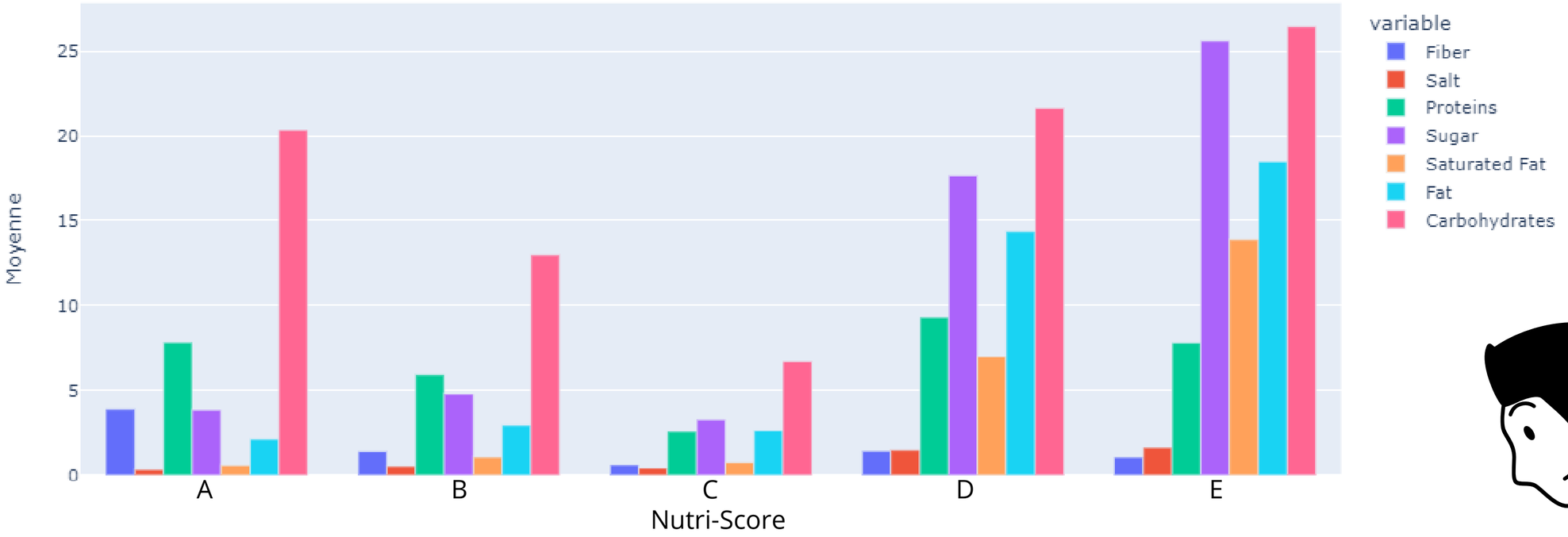
Partie 2: Observation –Tendance–

Répartition des Nutri-Scores



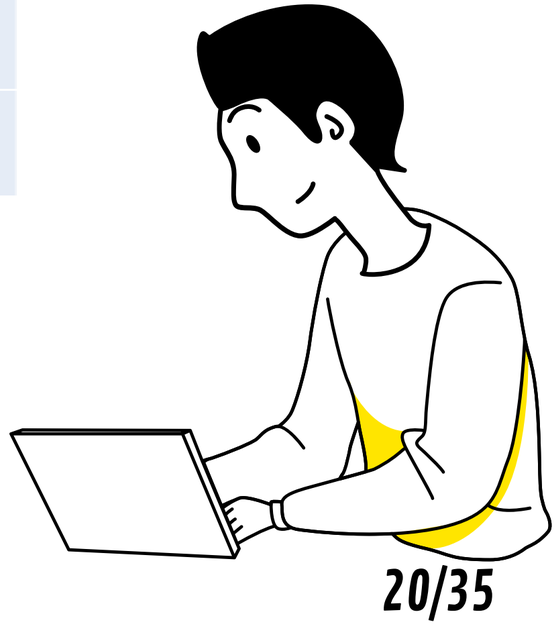
Partie 2: Observation –Tendance–

Moyennes des valeurs nutritionnelles par Nutri-Score



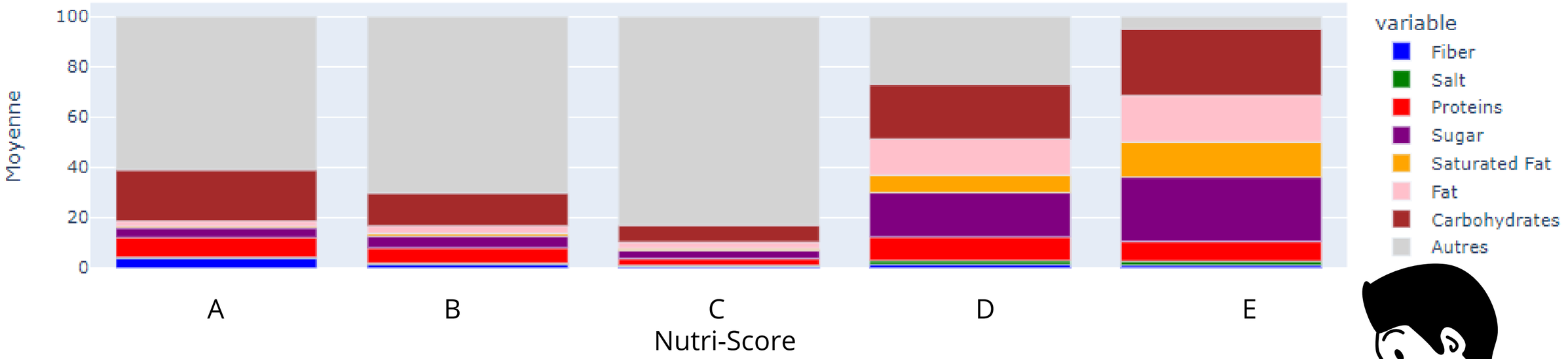
Fibre descend

Sucre monte



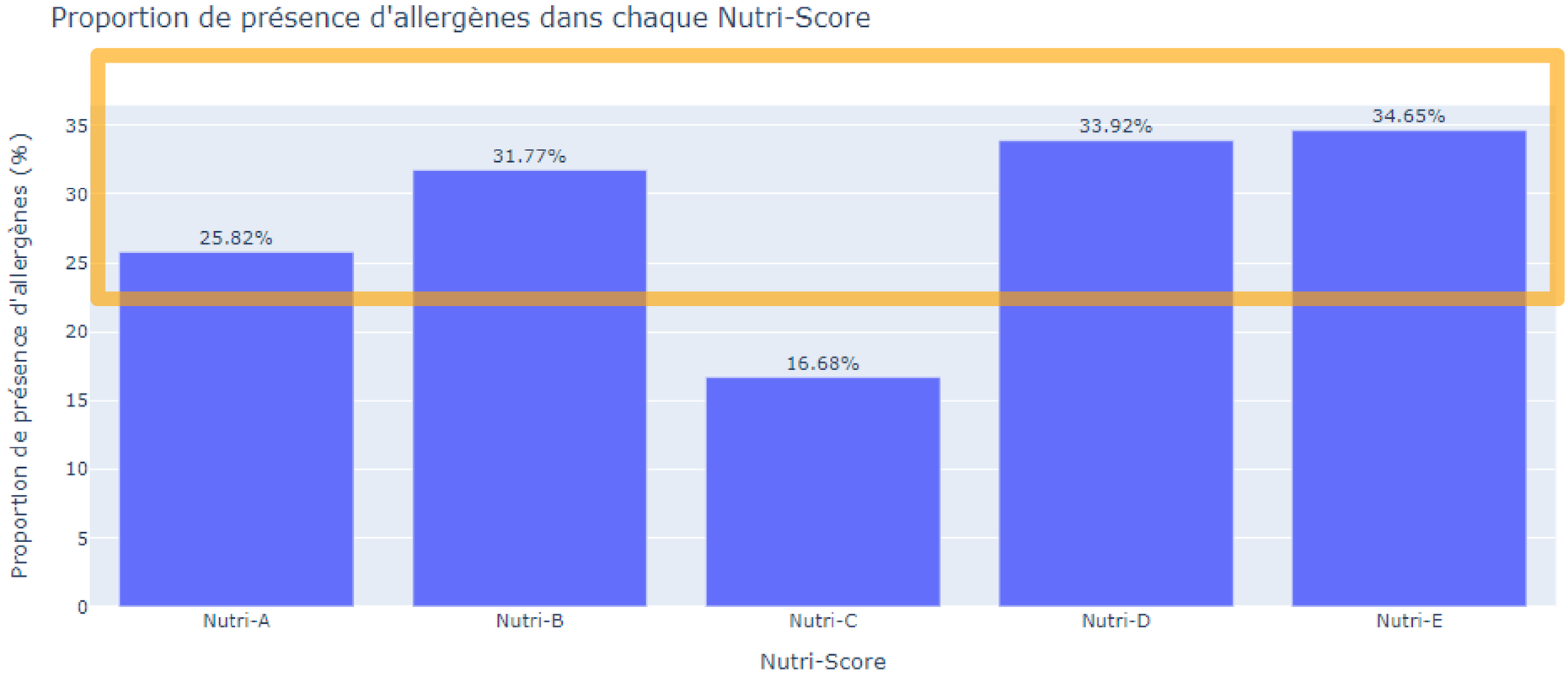
Partie 2: Observation –Tendance–

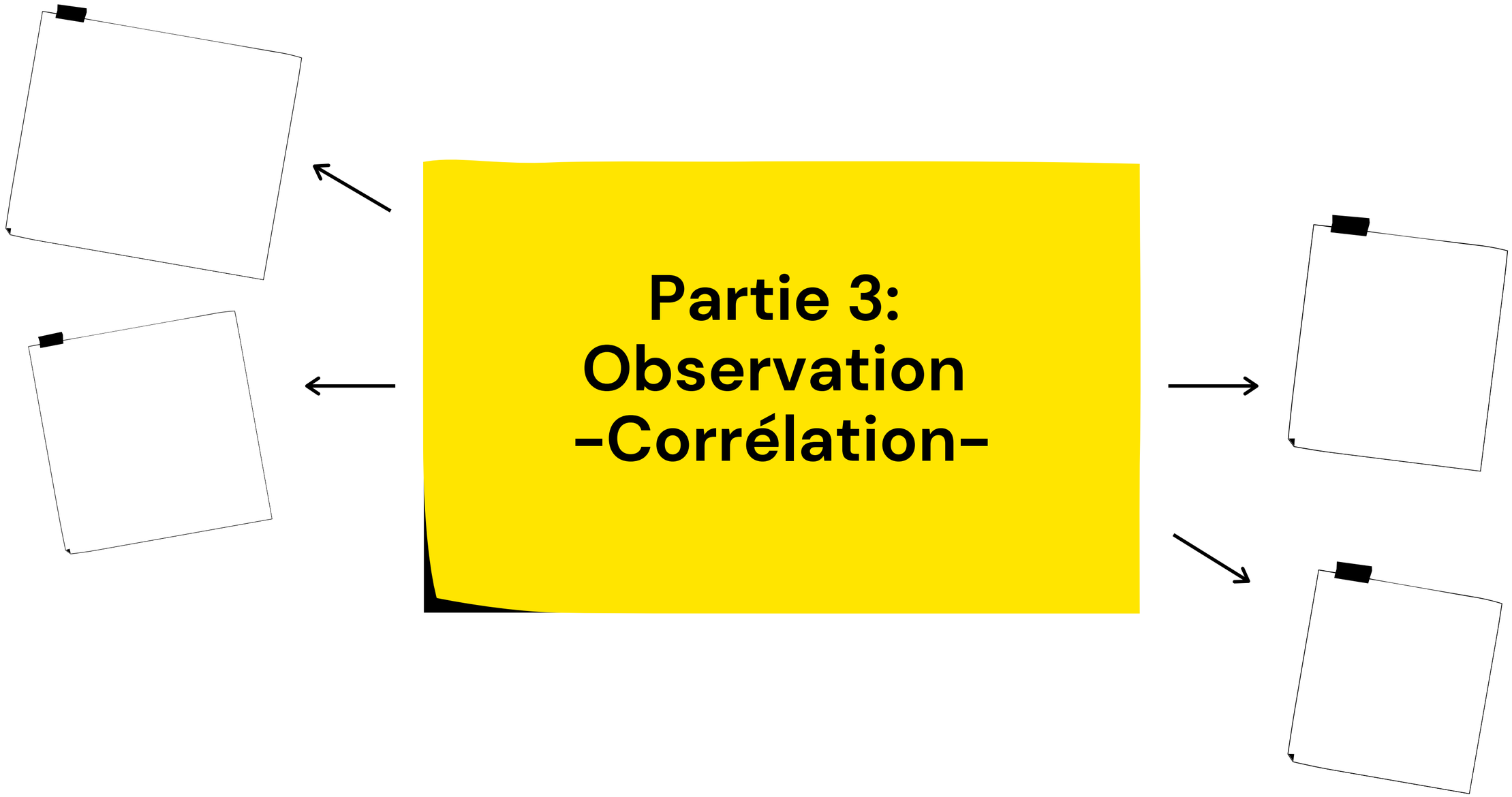
Moyennes des valeurs nutritionnelles par Nutri-Score



La part des Graisses et Graisse Saturées augmente

Partie 2: Observation –Tendance–





Partie 3:
Observation
-Corrélation-

Partie 3: Observation –Corrélation–

Mise en place

```
names = dataFrance.nutrition_grade_fr
names
```

```
2         5.0
9         4.0
10        4.0
14        5.0
16        4.0
          ...
```

Par rapport au
Nutri-Score
du produit

salt_100g	proteins_100g	fiber_100g	sugars_100g	saturated-fat_100g	fat_100g	energy-100g	carbohydrates_100g	Nbr_Allergenes
0.09652	2.5	2.5	57.5	12.5	20.0	1883.0	70.0	0
0.01000	0.6	0.9	87.7	0.8	0.0	1753.0	0.0	0
0.00300	9.5	3.9	50.3	2.9	0.0	2406.0	0.0	0
0.02540	0.0	0.0	10.4	0.0	0.0	177.0	10.4	0
0.80000	7.5	1.4	1.0	11.0	0.0	1079.0	0.0	0
...	...	...	...	...	...	...	...	...
0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

A quoi correspond :
sel, protéine, fibre, graisse, graisse saturée, énergie, glucide et d'allergènes

Partie 3: Observation –Corrélation–

Observation de tendance.

Corrélation forte et positive

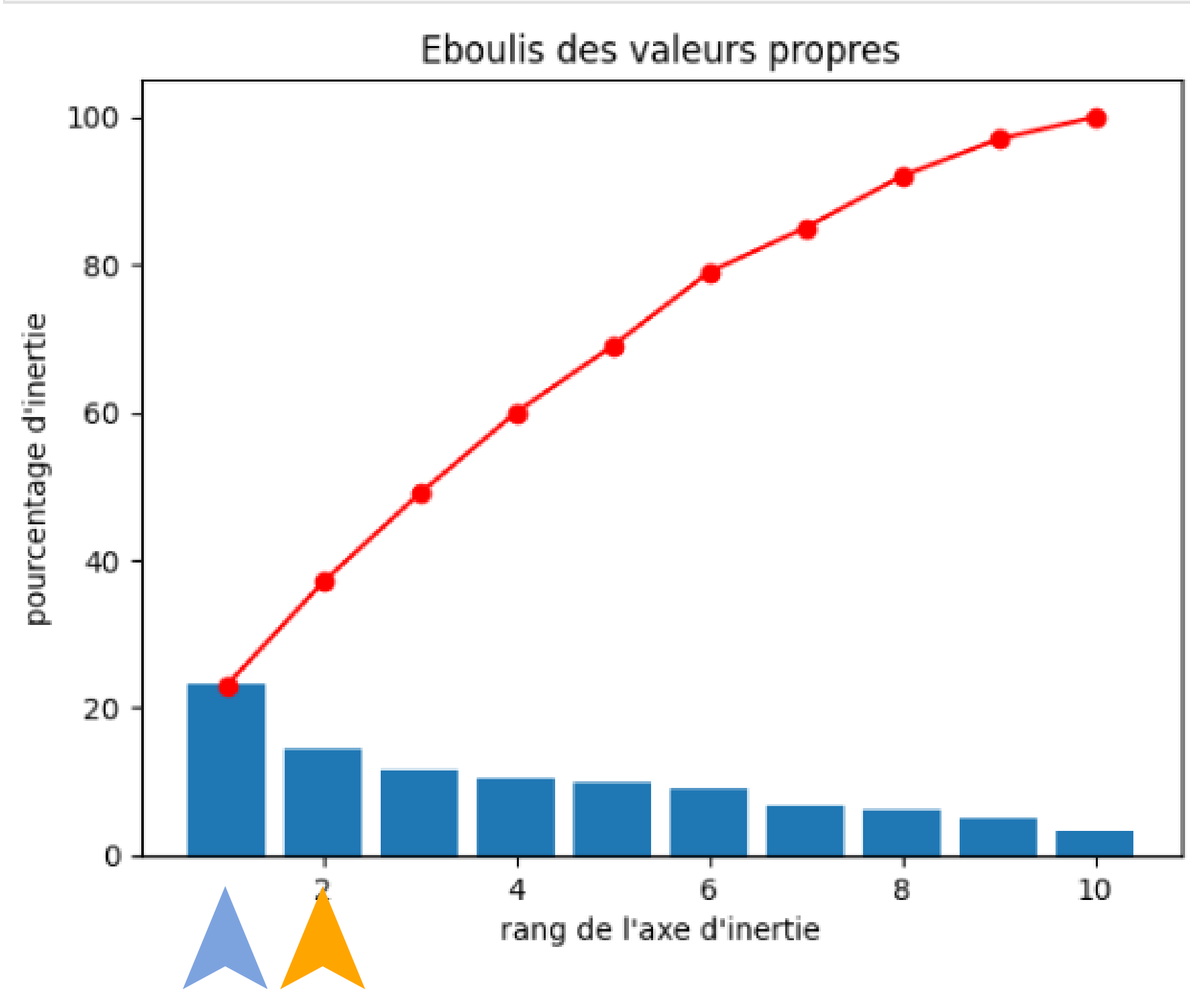
<Axes: >

	0	1	2	3	4	5	6	7	8	9
0	0.36	0.07	0.25	0.20	0.33	0.39	0.43	0.07	0.43	0.35
1	-0.27	0.26	0.46	0.06	-0.40	0.41	0.35	0.05	-0.40	-0.15
2	-0.28	0.18	0.30	0.68	0.24	-0.18	-0.28	0.15	0.22	-0.32
3	-0.34	-0.58	-0.20	-0.08	0.33	0.35	0.22	0.27	0.01	-0.7
4	0.13	0.10	-0.02	-0.10	-0.15	-0.13	-0.07	0.95	-0.04	
5	-0.11	0.72	-0.25	-0.38	0.40	0.14	0.02	0.02	0.07	-0.2
6	0.39	0.13	-0.59	0.51	-0.20	0.12	0.21	0.01	-0.18	-0.28
7	-0.59	0.12	-0.39	0.25	0.05	0.06	0.08	0.01	-0.10	0.63
8	0.25	-0.04	0.08	0.09	0.38	0.40	-0.52	0.01	-0.57	0.17
9	-0.07	0.01	-0.13	-0.04	-0.44	0.55	-0.49	-0.00	0.48	-0.02

1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00

Partie 3: Observation –Corrélation–

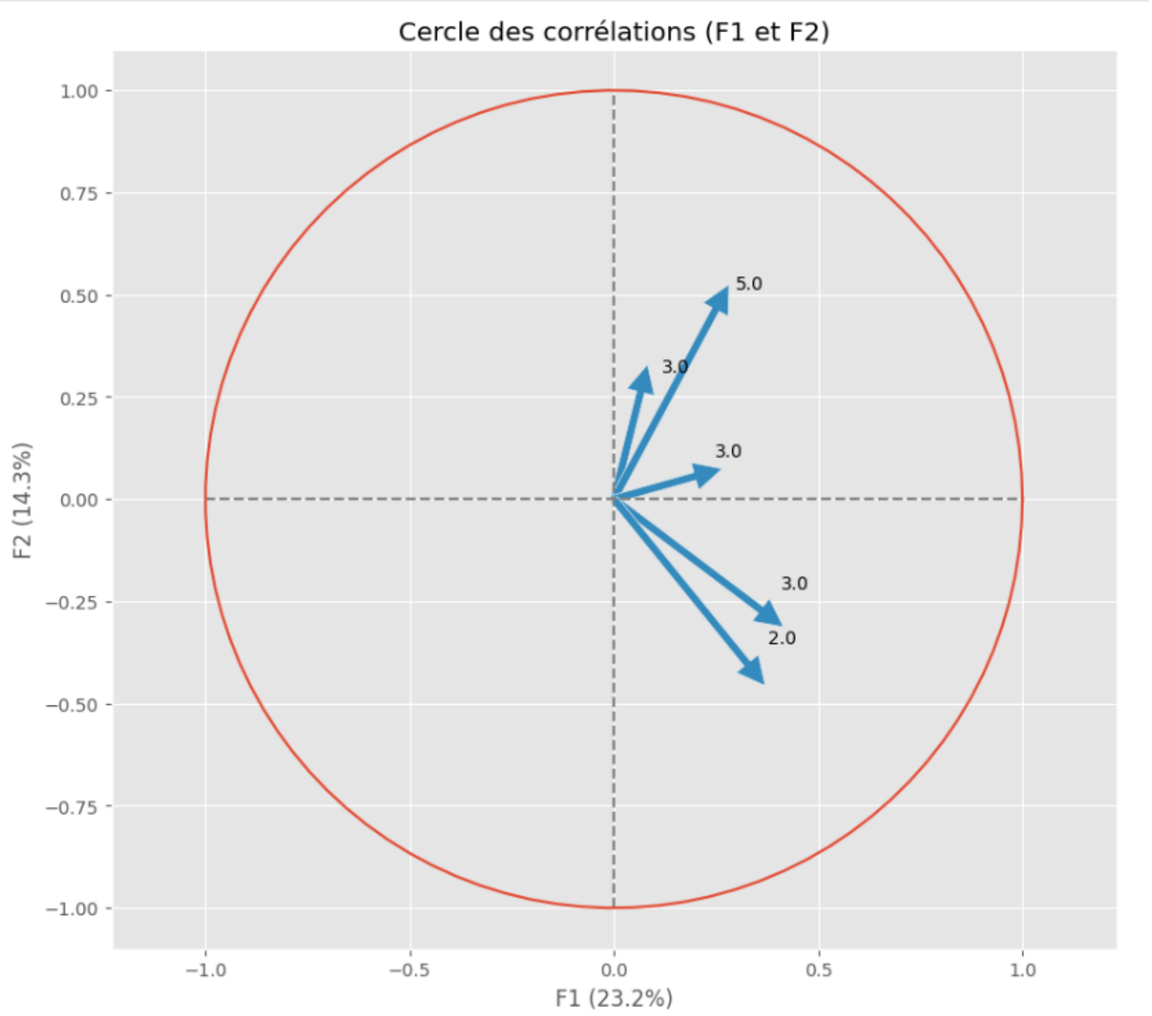
Observation de tendance.



La majorité de l'information est capturée par le premier composant principal (F1) et, dans une moindre mesure, par le deuxième (F2).

Partie 3: Observation –Corrélation–

Observation de tendance.



→ Les variables corrèlent avec F1

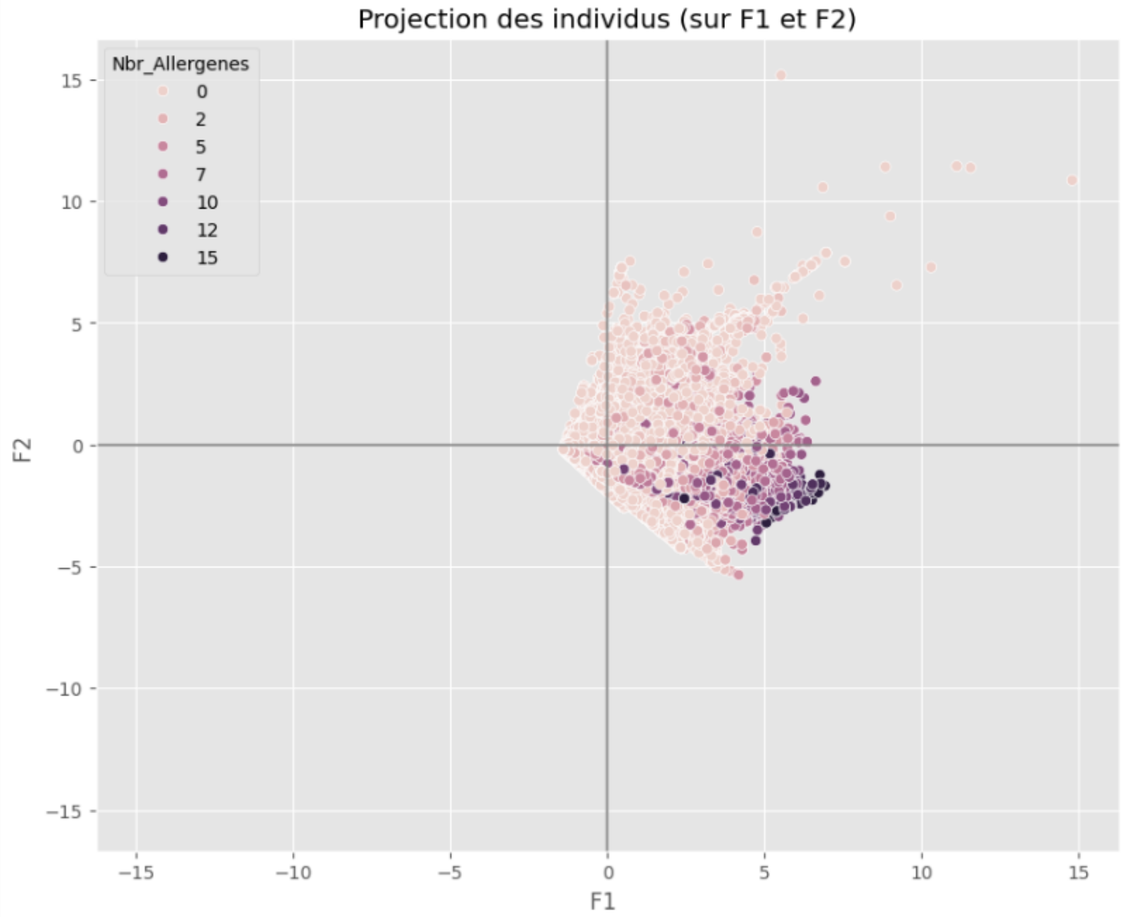
MAIS

Attention aux noms identiques

Partie 3: Observation –Corrélation–

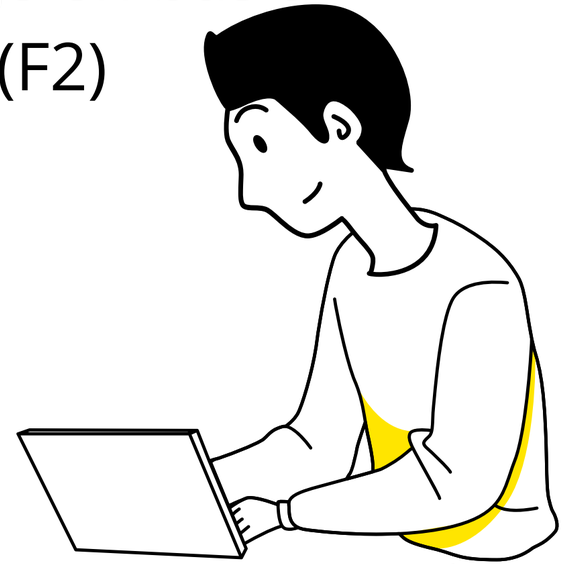
Observation de tendance.

```
[133]: x_y = [0,1]
display_factorial_planes(X_proj, x_y, clusters=dataFrance['Nbr_Allergenes'])
```



Nuage, principalement autour de F1

+ il y a d'allergène, + le nuage se dirige en bas à droite. (F2)



Partie 3: Observation –Corrélation–

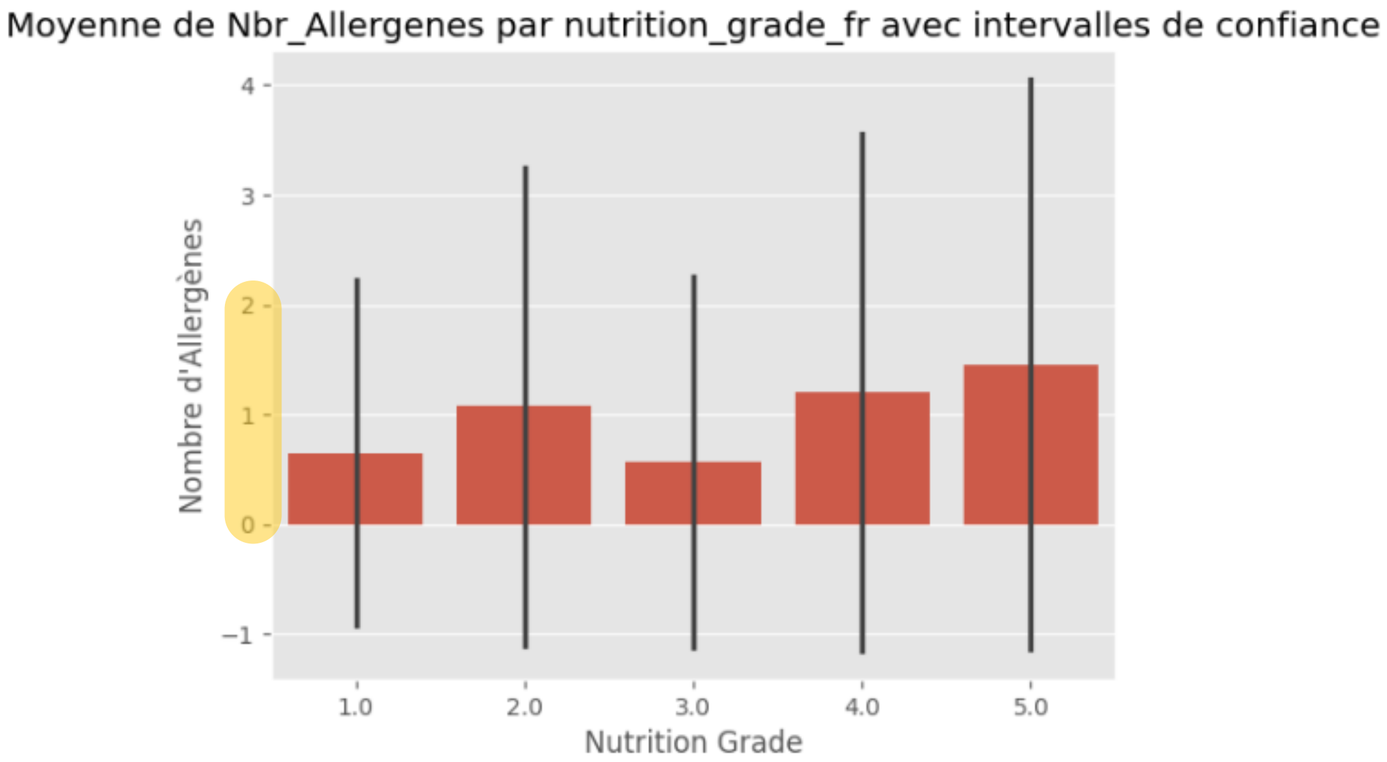
Observation de tendance.

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Création du modèle OLS pour ANOVA avec 'Nbr_Allergenes' comme variable dépendante
model = ols('Nbr_Allergenes ~ C(nutrition_grade_fr)', data=dataFrance).fit()

# Réalisation de l'ANOVA et affichage du tableau de résultats
anova_results = sm.stats.anova_lm(model, typ=2)
print(anova_results)
```

```
                            sum_sq       df           F  PR(>F)
C(nutrition_grade_fr)   10602.754236      4.0  656.451881     0.0
Residual               362761.104575  89839.0         NaN     NaN
```

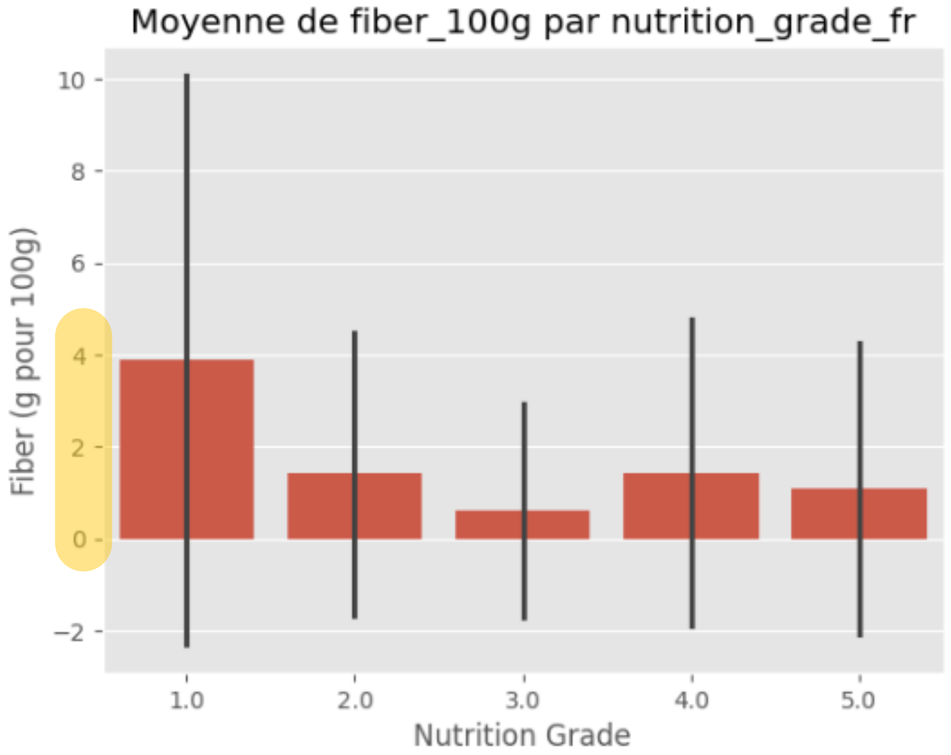
Partie 3: Observation –Corrélation–

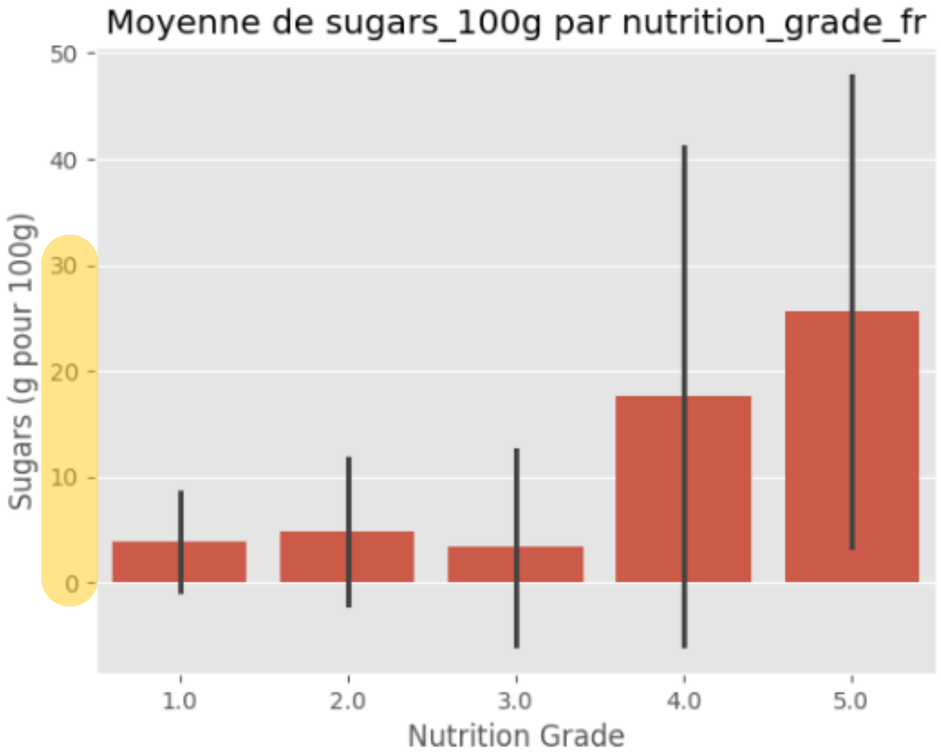
Observation de tendance.

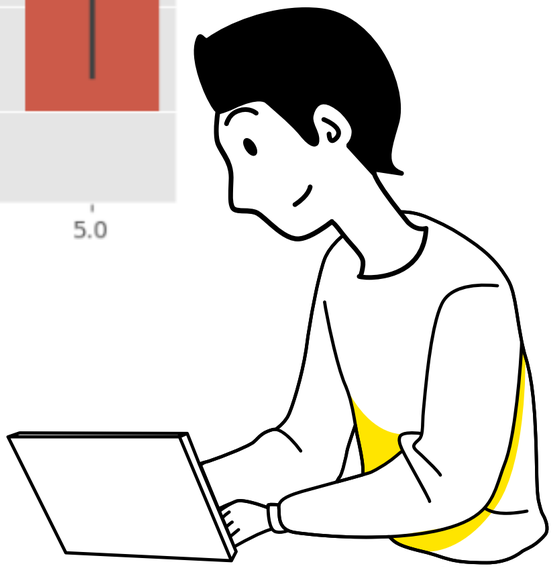


Partie 3: Observation –Corrélation–

Observation de tendance.







Partie 3: Observation –Corrélation–

Observation de tendance.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         Nbr_Allergenes   R-squared:                       0.028
Model:                            OLS   Adj. R-squared:                  0.028
Method:                 Least Squares   F-statistic:                     656.5
Date:                Thu, 18 Jan 2024   Prob (F-statistic):               0.00
Time:                        15:51:10   Log-Likelihood:            -1.9018e+05
No. Observations:               89844   AIC:                         3.804e+05
Df Residuals:                   89839   BIC:                         3.804e+05
Df Model:                           4
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                        0.6509      0.020     32.670      0.000       0.612       0.690
C(nutrition_grade_fr)[T.2.0]     0.4202      0.029     14.533      0.000       0.364       0.477
C(nutrition_grade_fr)[T.3.0]    -0.0828      0.022     -3.720      0.000      -0.126      -0.039
C(nutrition_grade_fr)[T.4.0]     0.5485      0.025     21.852      0.000       0.499       0.598
C(nutrition_grade_fr)[T.5.0]     0.8049      0.027     29.708      0.000       0.752       0.858
==============================================================================
Omnibus:                    56300.928   Durbin-Watson:                   1.517
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           539529.943
Skew:                           2.993   Prob(JB):                         0.00
Kurtosis:                      13.406   Cond. No.                         8.03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# Afficher les résultats du test de Tukey
print(tukey.summary())
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower   upper  reject
---------------------------------------------------
   1.0    2.0   0.4202    0.0  0.3413  0.4991   True
   1.0    3.0  -0.0828 0.0019 -0.1435 -0.0221   True
   1.0    4.0   0.5485    0.0  0.4801   0.617   True
   1.0    5.0   0.8049    0.0   0.731  0.8788   True
   2.0    3.0   -0.503    0.0 -0.5662 -0.4398   True
   2.0    4.0   0.1283    0.0  0.0576   0.199   True
   2.0    5.0   0.3847    0.0  0.3087  0.4607   True
   3.0    4.0   0.6313    0.0  0.5817   0.681   True
   3.0    5.0   0.8877    0.0  0.8308  0.9446   True
   4.0    5.0   0.2564    0.0  0.1912  0.3215   True
---------------------------------------------------
```

Partie 4: Rapport d’exploration.

Partie 4: Rapport d’exploration.

L’application est-elle faisable ?

Oui

Les premières pistes ergonomiques:

- **Liste déroulante**
- **Pré-remplissage**
- **Utilisation d’IA d’analyse d’image**

Pistes pour de futurs analyses du futurs jeu de donnés:

- **Nombres d’ingrédients**
- **Croisement JdD Additif**
- **Simplification**
- **Mise en avant de l’allergènes**

Merci !