URCA Reims

RAPPORT FINAL

PRÉDICTION SUR LE NUTRI-SCORE

2023

Brunet Alexandre
Ertas Elif
Jupin Manon
Gabet Léo
Kpadondou Carlos
Otogondoua Ememag Jordhy Jean Jaurès

SOMMAIRE

INTRODUCTION

Depuis quelques années, l'émergence des enjeux environnementaux s'est accompagnée de la question de la santé et du bien-être physique par l'alimentation. Ces deux questions combinées, en plus de la volonté d'informer le consommateur, a poussé l'Etat a instauré une norme d'étiquetage des produits transformés : le nutri-score.

Cette norme classe les produits selon leur impact positif ou négatif pour la santé des consommateurs. Le classement en question est illustré par des lettres allant de A (très bon pour la santé) à E (très mauvais pour la santé). La base de ce classement repose sur les composants nutritionnels des produits.

C'est pourquoi notre application porte sur la prédiction du nutri-score des produits à partir de leurs valeurs nutritionnelles. Cette application s'adresse à deux groupes de personnes. La première regroupe les consommateurs qui souhaite soit vérifier le nutri-score qui se trouve sur les produits, soit savoir le nutri-score à partir d'un produit qui n'en possède pas. Le second groupe regroupe les producteurs qui souhaitent évaluer le nutri-score de leurs produits sans coût et rapidement. Cette distinction a été réalisé pour deux raisons. D'abord, pour toucher un plus large panel d'individus, mais aussi (et surtout) car nous utiliserons des informations qu'un consommateur lambda n'a pas accès, seul les producteurs y ont accès.

Pour réaliser cette application, nous nous sommes appuyés sur une base de données regroupant un peu plus de 52 260 produits ayant chacun leurs caractéristiques nutritionnelles (variables explicatives) et leur nutri-score variable cible (tableau 1).

Variables	**Description**
energy_100g	Quantité de kilocalories pour 100g
fat_100g	Quantité de matières grasses pour 100g
saturated-fat_100g	Quantité de graisses saturées pour 100g
trans-fat_100g	Quantité de graisses transformées pour 100g
cholesterol_100g	Quantité de cholestérol pour 100g
carbohydrates_100g	Quantité de carbohydrates pour 100g
sugars_100g	Quantité de sucres pour 100g
fiber_100g	Quantité de fibres pour 100g
proteins_100g	Quantité de protéines pour 100g
salt_100g	Quantité de sel pour 100g
sodium_100g	Quantité de sodium pour 100g

vitamin-a_100g	Quantité de vitamine A pour 100g
vitamin-c_100g	Quantité de vitamine C pour 100g
calcium_100g	Quantité de calcium pour 100g
iron_100g	Quantité de fer pour 100g
nutrition_grade_fr	**Notre variable cible qui est décrite par des lettres allant de A à E exprimant la qualité nutritionnelle du produit.**

Tableau 1 : Description de toutes les variables utilisées.

Cette base est présente sur le site de l'open-data du gouvernement[1]. Grâce à cela, nous allons mobiliser des outils d'apprentissage statistique pour qu'un consommateur ou producteur, en entrant les informations du produit, puisse évaluer de la pertinence du nutri-score d'un produit (ou alors connaître le son nutri-score dans le cas où il n'y a pas accès).

[1] https://www.data.gouv.fr/fr/datasets/open-food-facts-produits-alimentaires-ingredients-nutrition-labels/

1. Interfaces utilisateur

1.1. Interface d'entrée

Le projet débute avec une page d'accueil (figure 1) qui présente notre application, à qui est à destination de cette application et pourquoi, un message d'avertissement qui s'adresse aux utilisateur disant qu'il faut rentrer le données d'une certaine manière : réfléchir par 100g. Autrement dit, considérer les valeurs comme celles qui sont inscrits sur les paquet que l'on trouve au supermarché. Nous avons également les différents historiques et dashboard (nous y reviendrons plus tard), et un message adressée spécifiquement aux consommateurs. Enfin, nous avons un bouton qui permet d'ouvrir le formulaire dans lequel les valeurs seront rentrées par les utilisateurs.

Figure 1 : Page d'accueil.

Poursuivons avec la création d'une interface utilisateur sous forme d'un formulaire dans Excel qui a plusieurs parties (figure 2). La première, la plus importante, est celle du choix entre consommateur et producteur. Nous avons fait cette distinction car dans notre base de données, certaines informations ne se trouvent pas sur les paquets des produits (comme le cholestérol),

ou en tout cas pas à tous les coups. Ensuite, nous avons le nom du produit que l'utilisateur doit insérer. Nous avons également la partie de l'historique : l'utilisateur peut visualiser les différents produits qu'il a déjà inséré. Enfin, nous avons la plus grande partie qui concerne les informations que l'utilisateur veut insérer.

Dans le cas où un produit est déjà présent dans l'historique et que l'utilisateur insère à nouveau le nom de ce produit, un message s'afficher en disant que le produit existe déjà et donne également le choix de modifier ou non les informations de ce produits. Aussi, lorsque l'utilisateur fait le choix entre consommateur et producteur (choix qu'il doit faire en premier, sans quoi aucune information ne peut être insérée), les informations grisées sont celles qui ne sont pas disponibles pour le type choisi. La raison de cette sécurité tient dans le modèle statistique. En effet, nous avons générer 2 modèles différents, basés sur des variables différentes. Cela s'exprime sur le formulaire par la disponibilité ou non d'insérer certaines valeurs. En parlant de ces valeurs, puisqu'elles sont exprimées par 100g, aucune valeur ne doit excéder 100g (sauf pour les kilocalories).

Naturellement, nous trouvons le bouton de confirmation (« Calculer Nutri-Score ») de couleur verte qui va calculer le Nutri-Score avec les informations insérées, puis le bouton pour quitter le formulaire, en rouge. Lorsque l'utilisateur veut confirmer ses informations grâce au bouton vert, un récapitulatif ainsi qu'une demande de (re)confirmation apparaissent. De plus, le bouton vert ne sera actif que lorsque toutes les variables seront entrées dans le formulaire

Pour finir, la vérification des valeurs est importante pour Python dans sa lecture des données des entrées. Par exemple, le formulaire supprime automatiquement un caractère, car nous attendons des valeurs numériques au format « 0.0 ». Ainsi, si « 0,0 » est saisi, notre code le met à jour avec le format « 0.0 », le seul format numérique accepté en langage Python. Cela nous a permis d'éviter d'afficher des messages à chaque case, car nous trouvons ce principe très lourd pour l'utilisateur. Avec notre code, les entrées et corrections sont plus fluides et l'utilisateur peut poursuivre son activité, sans à se soucier des détails comme celui-ci.

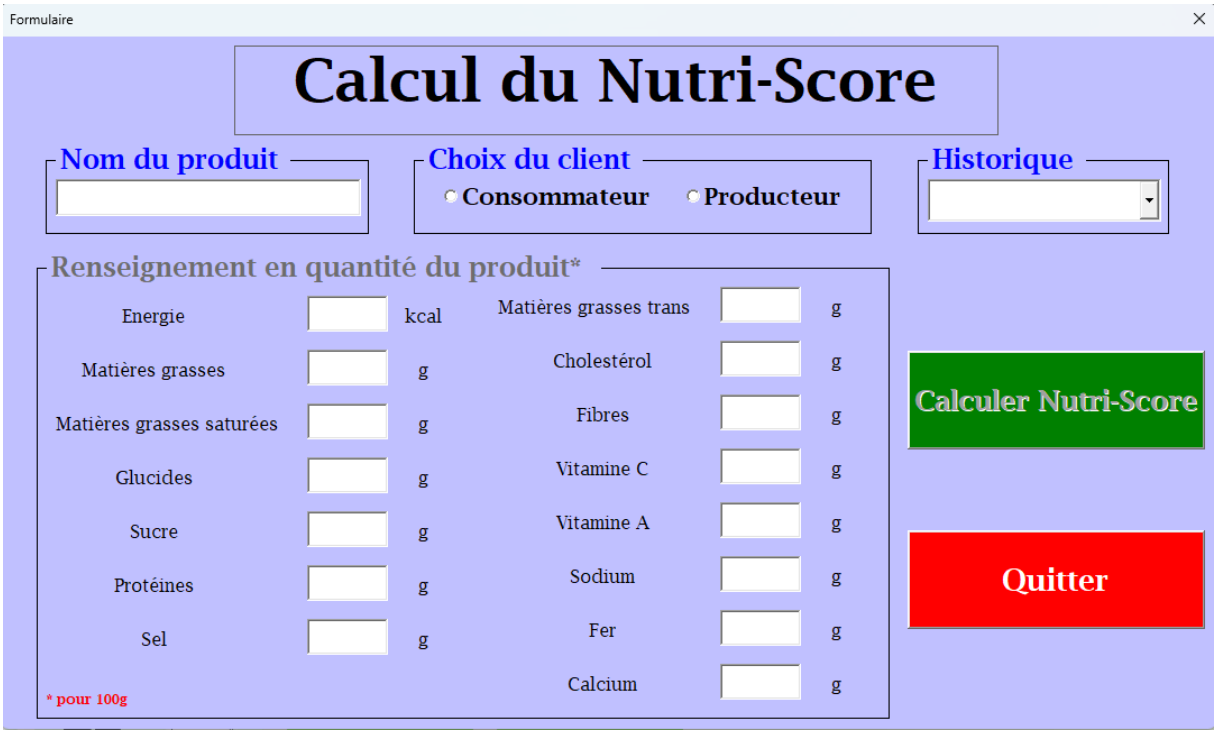


Figure 2 : Formulaire d'entrée.

1.2. Interface de sortie

Quand l'utilisateur a réussi à entrer les valeurs de son produit, le Nutri-Score se calcul grâce aux résultats obtenus par Python, via l'API. En fonction de ce que l'utilisateur à cocher, il serait dirigé vers le dashboard du producteur (figure 3) ou celui du consommateur (figure 4) dans lesquels les messages sont personnalisés en fonction du public.

A l'intérieur de ces dashboard, nous trouvons le plus important : le Nutri-Score que retourne l'algorithme. Evidemment, l'algorithme ne fait pas des choix catégorique, mais estime des probabilités d'appartenance. C'est le Nutri-Score qui obtient la plus grande probabilité qui est mis en avant avec une jauge et un message qui commente le résultat. Quand aux autres probabilités, ils sont résumés dans le diagramme circulaire situé en dessous de la jauge qui résume toutes les probabilités. L'intérêt est de voir si l'algorithme est plutôt « confiant » dans son résultat. Par exemple un produit qui a 68% de chance d'être classé dans la catégorie B, veut dire qu'il y a peu de doute sur son classement. Ensuite, dans la même partie des dashboard, nous avons le top 3 des nutriments qui influencent le plus le calcul des probabilités et les performances des modèles (nous en parlerons en détails dans la partie sur les modèles utilisés). A droite du dashboard, nous avons l'erreur de classification des modèles utilisés, indiquant ainsi la « fiabilité » de celui-ci (spoiler : le modèle du producteur est beaucoup plus fiable). Enfin,

quelques recommandations apparaissent, à la fois sur la composition des produits des producteurs (pour qu'ils fassent attention) et sur les limites que posent l'OMS pou avertir les consommateurs des « risques » nutritionnels.

Hors du dashboard, un historique apparait de la même façon que pour le formulaire d'entrée, ainsi si l'utilisateur veut revoir le résultat de ses anciens produits, il n'est pas obligé de revenir sur le formulaire. L'objectif est bien de fluidifier l'expérience de l'utilisateur.

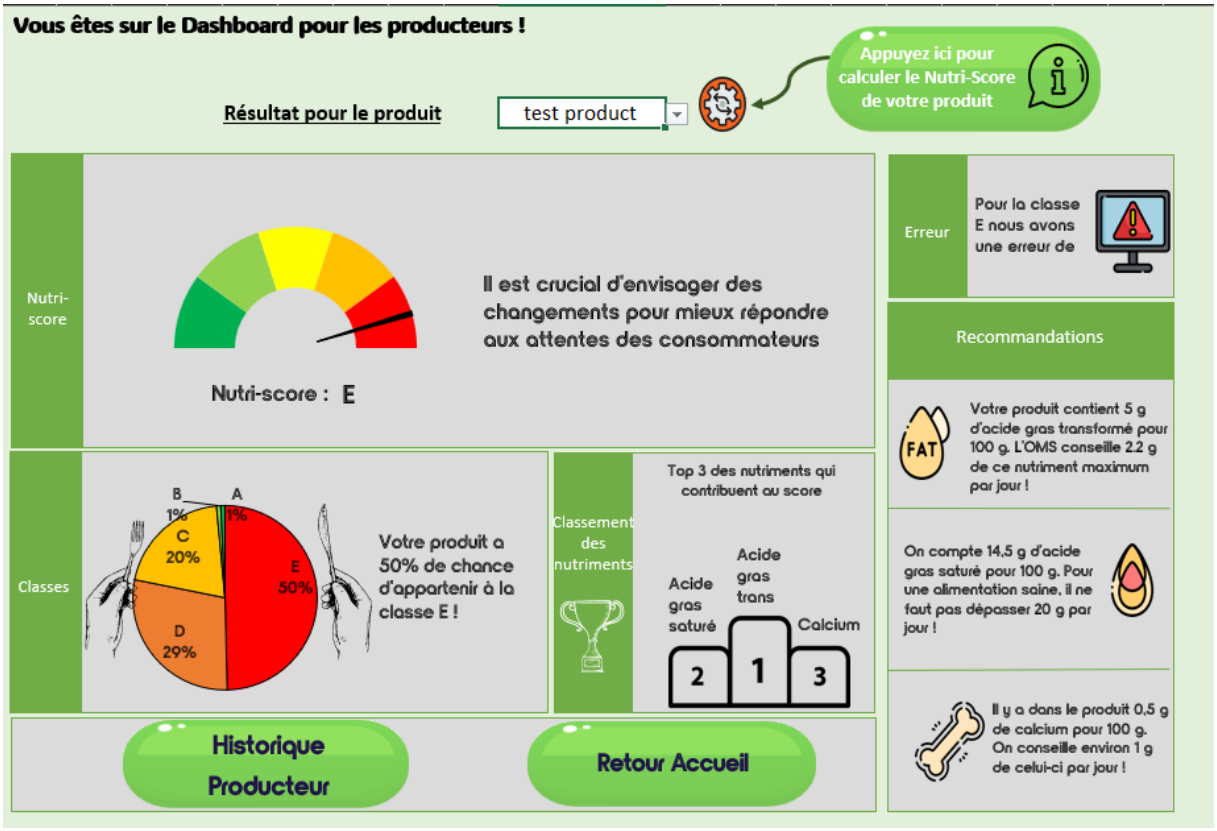


Figure 3 : Dashboard du producteur.

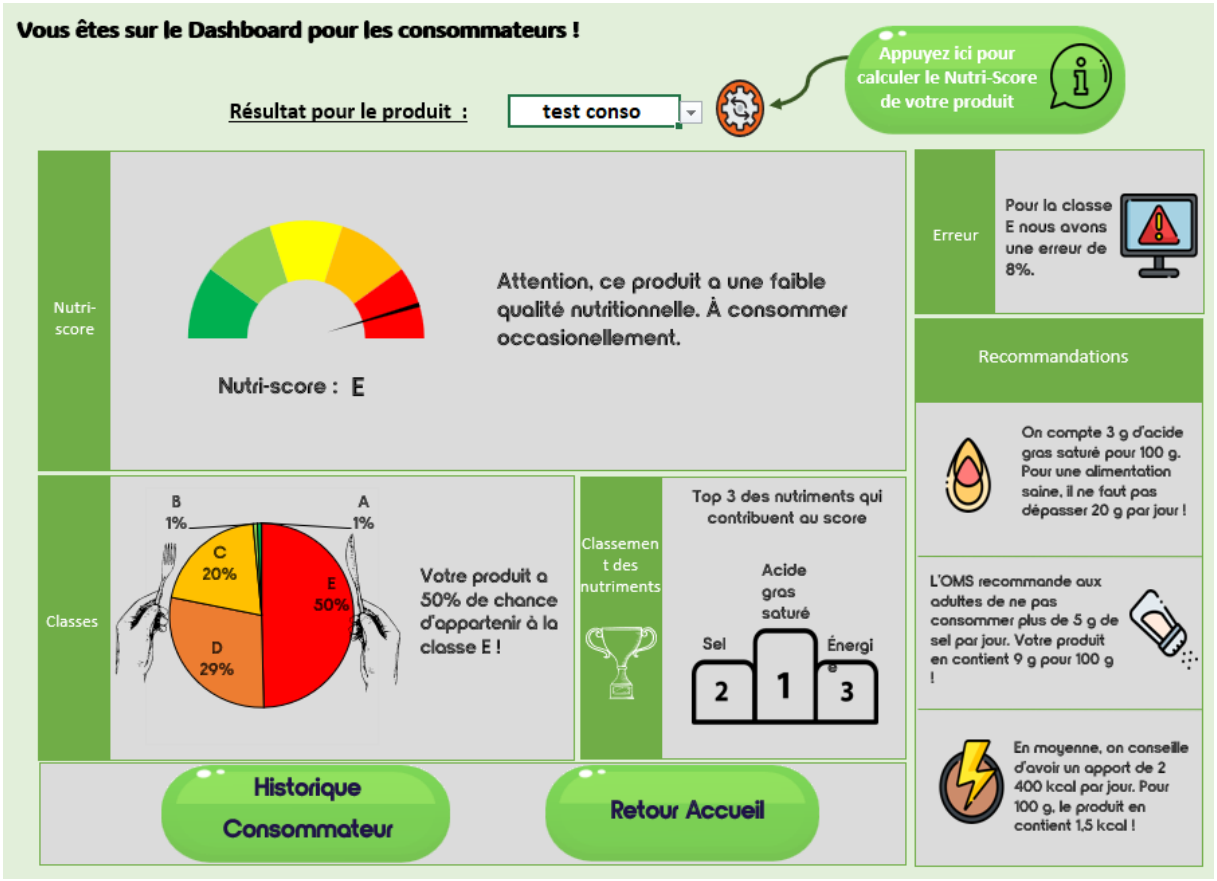
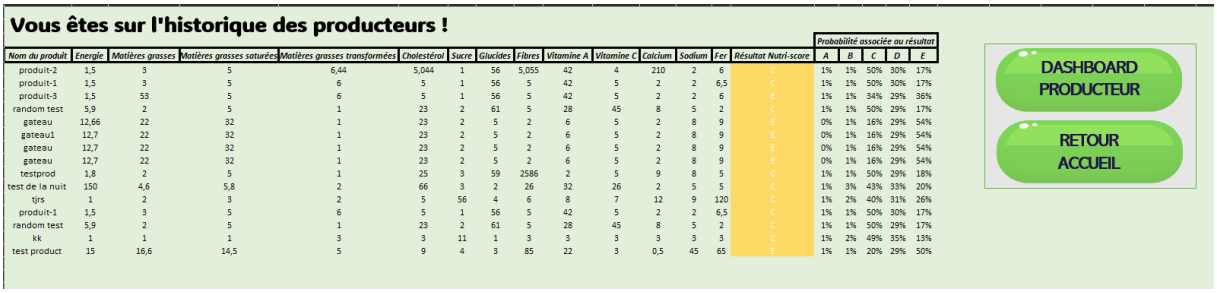


Figure 4 : Dashboard du consommateur.

En outre, nous avons mis à disposition des boutons lui permettant de revenir à l'accueil et de consulter l'historique (figure 5 et 6). Dans ces historique, nous retrouvons toutes les informations insérer par l'utilisateur dans sa page dédiée (consommateur ou producteur) avec les différentes probabilités ainsi que le Nutri-Score le plus probable pour chaque produit. Deux boutons sont également mis à disposition : le retour l'accueil et la visualisation du dashboard. L'utilité de tous ces boutons c'est d'améliorer l'expérience de l'utilisateur en lui évitant les désagréments induit par la navigation entre les différentes feuilles du fichier Excel.



Nom du produit	Energie	Matières grasses	Matières grasses saturées	Matières grasses transformées	Cholestérol	Sucre	Glucides	Fibres	Vitamine A	Vitamine C	Calcium	Sodium	Fer	Résultat Nutri-score	Probabilité associée au résultat				
															A	B	C	D	E
produit-2	1,5	3	5	6,44	5,044	1	56	5,055	42	4	210	2	6	C	1%	1%	50%	30%	17%
produit-1	1,5	3	5	6	5	1	56	5	42	5	2	2	6,5	C	1%	1%	50%	30%	17%
produit-3	1,5	53	5	6	5	1	56	5	42	5	2	2	6	E	1%	1%	34%	29%	36%
random test	5,9	2	5	1	23	2	61	5	28	45	8	5	2	C	1%	1%	50%	29%	17%
gateau	12,66	22	32	1	23	2	5	2	6	5	2	8	9	E	0%	1%	16%	29%	54%
gateau1	12,7	22	32	1	23	2	5	2	6	5	2	8	9	E	0%	1%	16%	29%	54%
gateau	12,7	22	32	1	23	2	5	2	6	5	2	8	9	E	0%	1%	16%	29%	54%
gateau	12,7	22	32	1	23	2	5	2	6	5	2	8	9	E	0%	1%	16%	29%	54%
testprod	1,8	2	5	1	25	3	59	2586	2	5	9	8	5	C	1%	1%	50%	29%	18%
test de la nuit	150	4,6	5,8	2	66	3	2	26	32	26	2	5	5	C	1%	3%	43%	33%	20%
tjrs	1	2	3	2	5	56	4	6	8	7	12	9	120	C	1%	2%	40%	31%	26%
produit-1	1,5	3	5	6	5	1	56	5	42	5	2	2	6,5	C	1%	1%	50%	30%	17%
random test	5,9	2	5	1	23	2	61	5	28	45	8	5	2	C	1%	1%	50%	29%	17%
kk	1	1	1	3	3	11	1	3	3	3	3	3	3	C	1%	2%	49%	35%	13%
test product	15	16,6	14,5	5	9	4	3	85	22	3	0,5	45	65	E	1%	1%	20%	29%	50%

Figure 5 : Historique du producteur.

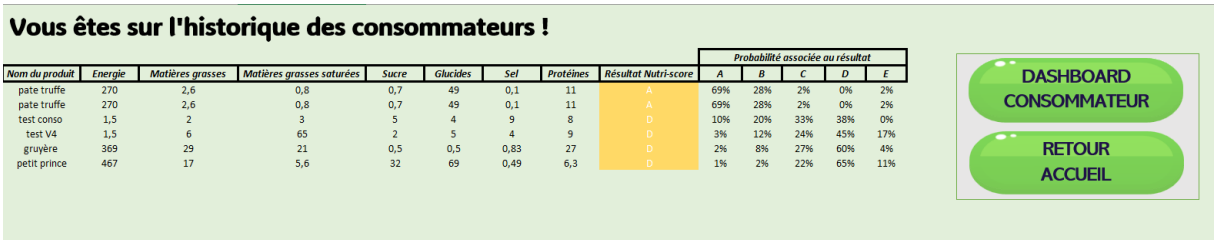


Figure 6 : Historique du consommateur.

2. Préparation des données

2.1. Lien VBA-Python

Notre application a pour objet d'utiliser à la fois VBA (pour la création d'interfaces), et à al fois Python (pour le développement d'un algorithme de prédiction). Or, ce sont deux logiciels qui ne sont pas par nature liés. C'est pourquoi nous devons faire la liaison nous-même. Certes, nous avons développer du code VBA pour la création du formulaire etc. Mais nous avons également fait en sorte que le code VBA gère le stockage des valeurs dans un historique et qui les envoie à Python pour appliquer le modèle (selon si l'utilisateur à cocher consommateur ou producteur). Il reste maintenant à savoir comment ce lien à pu s'établir. En fait, c'est grâce à un API. La relation Python-VBA s'effectue selon le processus suivant :

1. Les utilisateurs saisissent des données dans un formulaire Excel.
2. Les données saisies sont stockées dans des variables VBA pour une manipulation ultérieure.
3. Ces variables VBA sont utilisées pour créer une structure de données JSON (format {"clé": valeur}).
4. Une API Python a été développée pour réaliser la prédiction du Nutri-score. Cette API est hébergée à l'adresse http://kpcarlos.pythonanywhere.com/test/ .
5. Le script de l'API décharge les deux modèles python contenus dans le pickle
6. Puis récupère les données issus de la requête reçue
7. Effectue la prédiction des probabilité d'appartenance à chaque groupe
8. Enfin ces probabilités sont stocké dans des variables et mis au format JSON pour être retourné à Excel par l'API
9. Les données au format JSON sont envoyées à cette API Python pour effectuer la prédiction.
10. L'API Python renvoie une réponse au format JSON contenant les probabilité d'appartenance à chaque groupe de Nutri-score.
11. Une fonction appelée "jsonconverter" est utilisée pour interpréter le format JSON retourné et extraire la valeur du Nutri-score prédit.

Mais pour cela, il est nécessaire d'avoir, en plus de Microsoft Excel, Reference VBA project Microsoft scripting Runtime ainsi qu'une connexion internet, sans quoi l'API ne peut fonctionner et que donc aucune prédiction ne peut s'effectuer.

2.2. Nettoyage des données pour l’algorithme de prédiction

Le nettoyage de nos données a été relativement rapide et simple. En effet, notre base, en plus des valeur nutritionnelles des produits, contenait d’autres informations comme la présence d’additifs par exemple. Cependant, étant donné que le Nutri-Score ne prend en compte que les valeurs nutritionnelles, nous avons décidé de toute information ne relatant pas des variables que l’on étudiait. De plus, ces autres informations étaient inexploitables car manquaient cruellement de sens.

Nous avons réussi à rendre notre base de données entièrement exploitable par nos algorithmes. Cependant, une étape cruciale restait à accomplir : traiter les données manquantes. C'est la raison pour laquelle nous avons choisi de retirer les produits pour lesquels des informations étaient absentes. Finalement, nous avons pu conserver environ 29 000 produits, ce qui équivaut à un peu plus de la moitié de notre base initiale, qui comptait environ 52 000 éléments.

Par ailleurs, nous avons pris soin d'équilibrer nos données afin qu'aucune classe ne soit sur-représentée par rapport aux autres. En effet, la classe la moins représentée compte désormais 19% des données, tandis que la classe la mieux représentée en compte 22%, assurant ainsi une répartition équitable des informations dans l'ensemble de notre base de données.

3. Modèles utilisés

3.1. Régression logistique ordinale

La régression logistique ordinale est une extension de la régression logistique binaire pour le cas où la variable dépendante (cible) est ordonnée. Dans notre cas, notre variable cible a 5 modalités, cela signifie que nous avons 4 seuils qui délimitent les différentes catégories. Le modèle de régression logistique ordinale peut être formulé comme suit. Supposons que Y soit la variable dépendante ordinale avec K catégories ordonnées, et X soit le vecteur des variables explicatives (15 dans notre cas). Les probabilités conditionnelles de Y étant dans chaque catégorie k par rapport à une catégorie de référence sont données par :

$$P(Y \leq k|X) := \frac{1}{1 + exp^{-(\alpha_k - X\beta)}}$$

Où α_k est le seuil associé à la catégorie k et β est le vecteur des coefficients des variables explicatives. La fonction du maximum de la log-vraisemblance pour le modèle peut être exprimée comme le produit des probabilités conditionnelles. Elle sert surtout à estimer les différents paramètres de nos modèles.

$$\ell(\alpha, \beta|Y, X) := \sum_{i=1}^{N} log[P(Y_i|X_i, \alpha, \beta)]$$

Nous avons dans un premier temps estimé les paramètres du modèle initial. Cela a pu se faire en maximisant la fonction précédente. Dans un second temp, nous avons utilisé des critères de sélection pour supprimer les variables qui semblent, selon ces critères peu pertinenets. Ces critères sont ceux de l'AIC (Critère d'Information Akaike) et du BIC (Critère d'Information Bayésien). Ils prennent en compte la qualité du modèle en termes de vraisemblance et la pénalisation en fonction du nombre de paramètres (de variables). Cependant, le BIC pénalise d'autant plus les modèles complexes que l'AIC. Ces deux critères sont définis comme ceci :

$$AIC(M_k) := -2 \cdot \log\left[l(\alpha, \beta|Y, X)\right] + 2k$$

$$BIC(M_k) := -2 \cdot log(l(\alpha, \beta|Y, X)) + k \cdot log(N)$$

Où k est le nombre de variables dans le modèle et N est le nombre d'observations. Pour trouver les modèles optimaux au sens de ces deux critères, nous avons calculé quel sont les modèles qui minimisent les deux critères :

$$k^*_{AIC} := \underset{k\in\{1,\dots,m\}}{\arg\min}\, AIC(M_k)$$

$$k^*_{BIC} := \underset{k\in\{1,\dots,m\}}{\arg\min}\, BIC(M_k)$$

Enfin, pour élargir le choix du modèle optimal, nous avons également utiliser les méthodes de régularisation Ridge et Lasso. Ils sont utilisés pour résoudre le problème de multicollinéarité et pour sélectionner les variables importantes. Dans les deux cas, ces méthodes de régularisation peuvent aider à prévenir le surajustement du modèle et à sélectionner les variables importantes tout en ajustant les modèles. La régression Ridge et Lasso introduisent respectivement une pénalité L2 et une pénalité L1 sur les coefficients du modèle. Les fonction objective à optimiser sont les suivantes :

$$Minimiser\ J(\alpha,\beta) := \sum_{i=1}^{N} \log\left[P(Y_i|X_i,\alpha,\beta)\right] + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

$$Minimiser\ J(\alpha,\beta) := \sum_{i=1}^{N} log\ \left[P(Y_i|X_i,\alpha,\beta)\right] + \lambda_1 \sum_{j=1}^{p} \left|\beta_j\right|$$

Avec N est le nombre d'observations, Y_i est la variable dépendante pour l'observation i, X_i est le vecteur des variables explicatives pour l'observation i, α sont les seuils de catégorie, β sont les coefficients des variables explicatives, et λ_i pour tout $i \in \{1,2\}$ étant le paramètre de régularisation L1 ou L2. Le choix de ce dernier est crucial pour garantir que les modèles obtenus présentent une erreur minimale.

Maintenant, il nous restait plus qu'à calculer l'erreur de classification de chaque modèle. Cette erreur peut se calculer tout simplement avec la précision (ou accuracy) qui représente le taux d'observations correctement classées sur le total des observations sur la base de test :

$$Précision_i = \frac{a_{ii}}{\sum_{j=1}^{k} a_{ij}}$$

Avec a_{ii} le nombre d'observations de la classe *i* correctement prédites (aussi appelés « vrais positifs ») et $\sum_{j=1}^{k} a_{ij}$ est le nombre total d'observations prédites comme appartenant à la classe i (comprenant les vrais et faux positifs).

Finalement, pour choisir le modèle qui classifie le mieux en fonction des variables que disposent le producteur, nous avons comparé tous les modèles optimaux et le modèle initial (tableau 2). C'était ce dernier que nous avions décidé de retenir.

Modèle optimaux	Initial	Au sens de l'AIC	Au sens du BIC	Ridge	Lasso
Erreur de classification (en %)	26,1	28	28	43,5	27,5

Tableau 2 : Comparaisons des erreurs de classification des différents modèles optimaux.

3.2. Random Forest

Etant donné l'erreur relativement élevée qu'obtient la régression logistique ordinale, nous avons décidé de nous tourner vers une autre méthode de classification : le Random Forest. Globalement, le Random Forest est une collection d'arbres de décision aléatoires décorrélés qui est beaucoup moins sensible aux données d'entraînement et qui est également beaucoup plus précis dans ses prédictions (car moins de chance de sur-ajustement).

Le Random Forest s'appuie sur la méthode du bagging (bootstrap aggregating). Elle consiste à entraîner aléatoirement nos arbres de décision sur une partie des données et une partie des variables sur le jeu de données d'entraînement, ce qui donnera des échantillon bootstrap. Pour chaque échantillon bootstrap, un arbre de décision est créé. Chacun divise les données en fonction des caractéristiques qui maximisent la pureté des nœuds. La pureté peut être mesurée grâce à l'indice de Gini :

$$Gini(t) := 1 - \sum_{i=1}^{C} p(i|t)^2$$

Avec C le nombre de classes (5 dans notre cas) et $p(i|t)$ le proportion d'observation de la classe i dans le nœud t. Plus précisément, nous calculons cette proportion ainsi :

$$p(i|t) := \frac{Nombre\ d'observation\ de\ la\ classe\ i\ dans\ le\ noeud\ t}{Nombre\ total\ d'observation\ dans\ le\ noeud\ t}$$

Chaque arbre de décision, qui peuvent être différents ou similaires, vont voter. Pour la classification du Random Forest, la prédiction est basée sur le vote majorité des arbres de décision :

$$Prédiction\ du\ Random\ Forest := argmax(\sum_{i=1}^{N} Prédiction\ de\ l'arbre_{i,c})$$

Avec N est le nombre d'arbres dans le Random Forest et $Prédiction\ de\ l'arbre_{i,c}$ est la prédiction de la classe c pour l'arbre i.

Maintenant, une question est encore en suspens : combien d'arbre nous devons avoir dans notre forêt et quelle doit être leur profondeur afin d'obtenir de meilleures performances ? Pour répondre à cette question, nous devons chercher ces hyperparamètres de façon appropriée. Autrement dit, nous devons trouver un nombre d'arbre et une profondeur appropriés qui permettent de maximiser la précision du modèle (et donc minimiser l'erreur de classification). La précision globale du Random Forest est exprimée ainsi :

$$Précision := \frac{\sum_{i=1}^{C} C_{i,i}}{N}$$

Avec $C_{i,i}$ les observations correctement classées.

Pour ce faire, une technique habituelle consiste à définir une grille de valeurs possibles pour les hyperparamètres, puis de spécifier le nombre d'itérations ou d'évaluations à effectuer de manière aléatoire. L'algorithme sélectionne ensuite des combinaisons aléatoires d'hyperparamètres à partir de cette grille et évalue les performances du modèle avec chaque combinaison grâce à une validation croisée. À la fin du processus, les hyperparamètres produisant les meilleures performances sont identifiés, offrant ainsi une approche plus rapide et efficace pour l'optimisation des paramètres de Random Forest. Cette approche nous a permis de trouver rapidement des combinaisons d'hyperparamètres qui maximisent les performances de nos modèles tout en évitant la lourdeur d'une recherche exhaustive. Maintenant, avec les hyperparamètres optimisés en main, on peut maintenant présenter les performances obtenues sur les données de test pour nos deux modèles (figures 1 et 2).

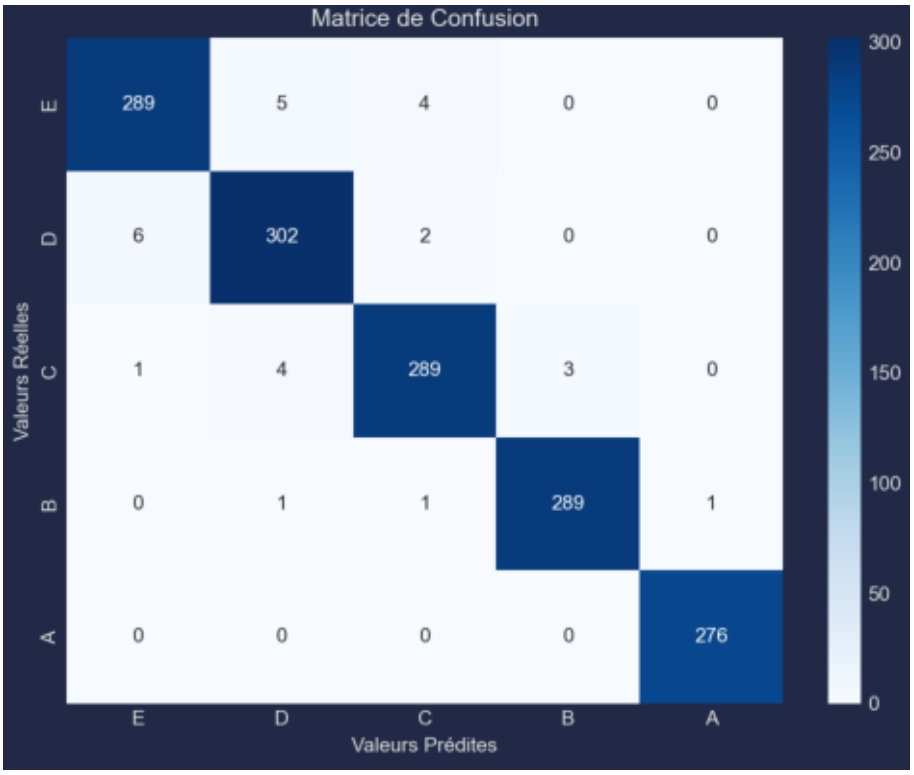


Figure 7 : Matrice de confusion pour le modèle des producteurs. **Lecture** : 276 produits ont été correctement classés dans le Nutri-Score A contre un produit qui a été classé dans le A alors qu'il était noté B.

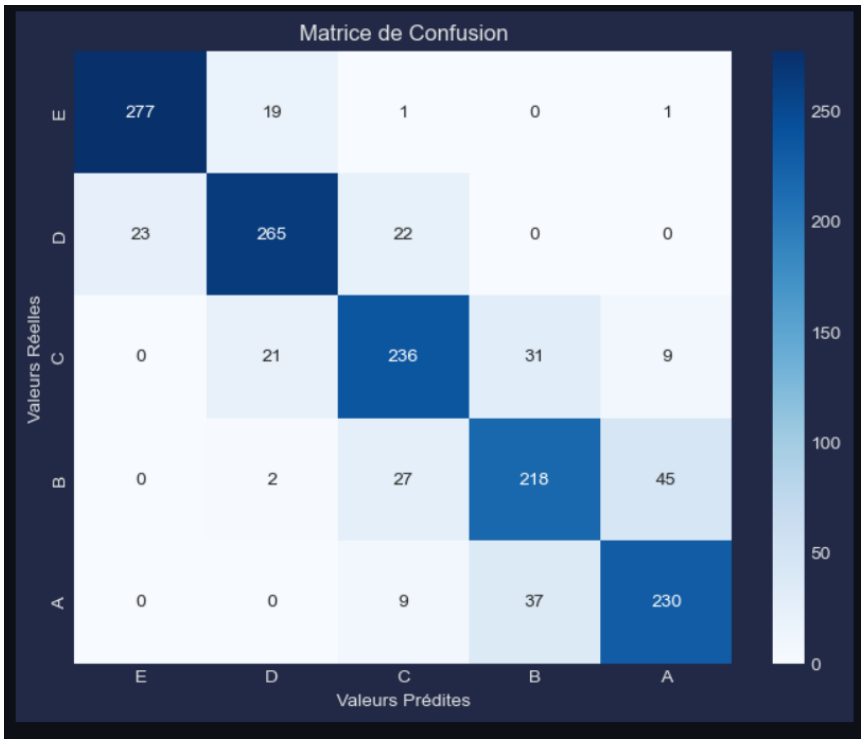


Figure 8 : Matrice de confusion pour le modèle des consommateurs. **Lecture** : 236 produits ont été correctement classés dans le Nutri-Score C contre 21 produit qui a été classé dans le D alors qu'il était noté C et 31 produits noté C mais qui ont été classé B.

Avec ces deux différentes matrices, nous pouvons établir un tableau résumant la précision globale des 2 modèles ainsi que la précision de chaque classe à partir des données de test (tableau 3).

Modèles	A	B	C	D	E	Total
Producteur	1	1	2	4	2	2
Consommateur	18	24	20	14	8	17

Tableau 3 : Précision des modèles après évaluation du Random Forest sur la base de test. **Lecture** : Un produit classé A par le modèle de Random Forest sur les caractéristiques du producteur a une chance de 1% d'être mal classé.

Afin de tester la robustesse des hyperparamètres sélectionnés, on a mis en place une approche itérative. Dans cette démarche, on entraîne le modèle avec les hyperparamètres optimisés en utilisant une base d'entraînement équilibrée pour toutes les classes. Cette base d'entraînement est tirée aléatoirement de la base de données globale à chaque itération. De manière complémentaire, on constitue une base de test également tirée aléatoirement de la base globale, sans nécessairement maintenir un équilibre entre les classes. Ce processus itératif a été répété plusieurs fois pour observer la valeur moyenne vers laquelle l'erreur de classification converge. Cette approche permet de garantir que les performances du modèle ne sont pas simplement le résultat d'une bonne adaptation à une configuration spécifique des données, mais qu'elles demeurent stables et fiables sur des ensembles de données variés. Les résultats obtenus à partir de cette validation itérative renforcent la confiance dans la capacité du modèle à généraliser de manière efficace à de nouvelles données.

Cette procédure a été effectué à la fois sur le modèle producteur et du consommateur. Et grâce à la loi des grands nombre, nous voyons que la moyenne des erreurs est très proche (dans le cas du modèle du consommateur), voire égale (dans le cas du modèle du producteur) à ce que l'on a obtenu précédemment.

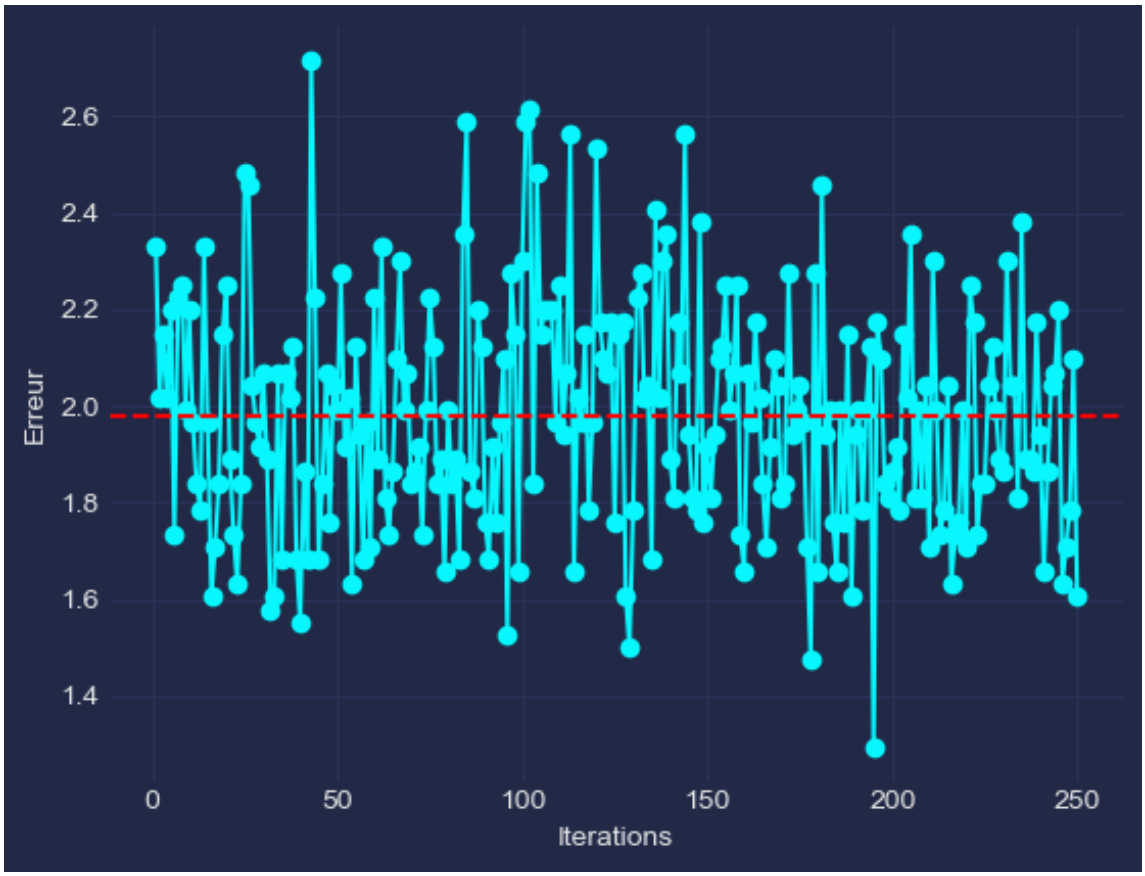


Figure 9 : Evolution de l'erreur du modèle du producteur selon les itérations.

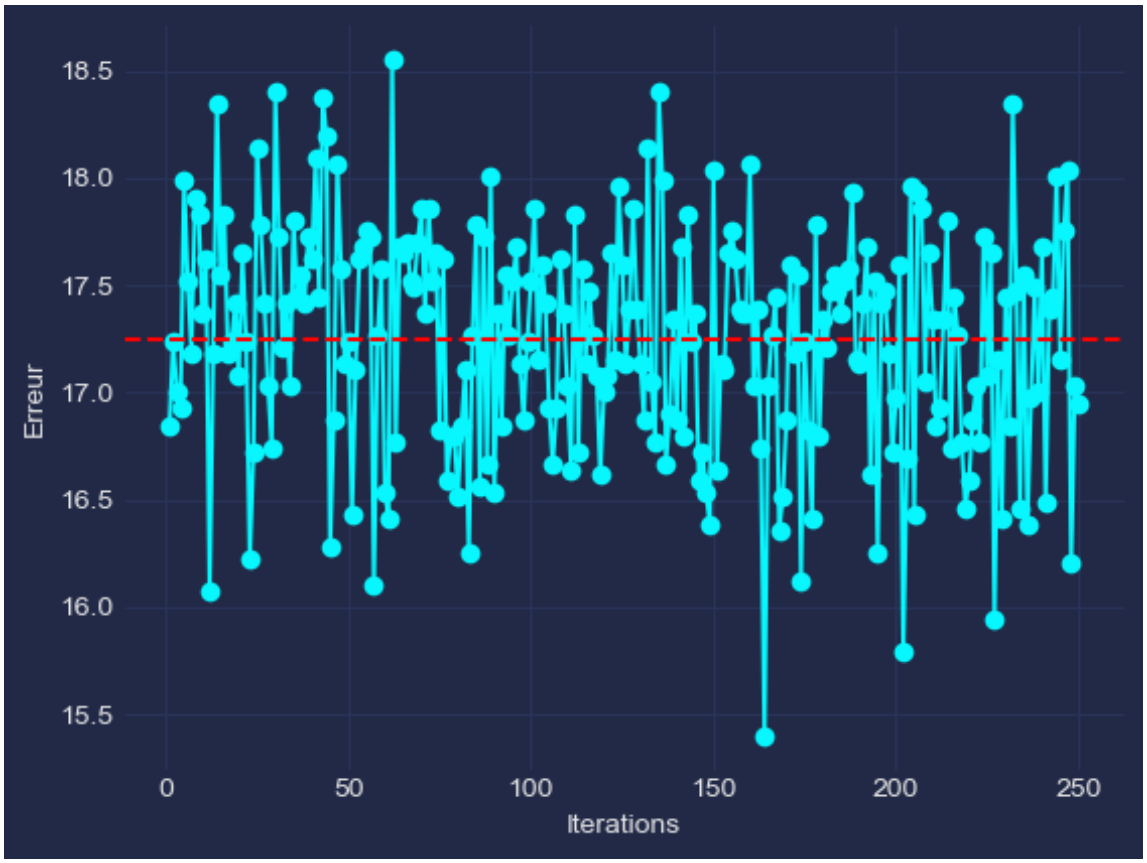


Figure 10 : Evolution de l'erreur du modèle du consommateur selon les itérations.

Nous avons également calculé l'importance de chaque variables (nutriments). Autrement dit, nous avons mesuré l'impact de chaque nutriment sur la précision du modèle en quantifiant la diminution de l'impureté causée par chaque nutriment lors de la construction des arbres de décision. Ainsi, nous avons :

$$Importance_{j,i} := \sum_{t} (p(t) \cdot \Delta I_{j,t,i})$$

Avec $p(t)$ la proportion du nombre d'observations dans le nœud t par rapport au nombre total d'observations dans l'ensemble d'entraînement et $\Delta I_{j,t,i}$ la diminution d'impureté due à la variable j dans le nœud t de l'arbre i. $\Delta I_{j,t,i}$. Nous mesurons $\Delta I_{j,t,i}$ comme la différence entre l'impureté du nœud avant la division et la somme des impuretés des nœuds fils après la division, pondérée par la taille des nœuds fils[2] :

[2] Un nœud fils est un nœud qui résulte de la division d'un nœud parent. Par exemple : un nœud parent rassemble 100 observations, dans le premier nœud fils nous retrouvons un sous-ensemble de 60 observations et le second nœud fils contient l'autre sous-ensemble avec les 40 observations restantes.

$$\Delta I_{j,t,i} := I(t) - \sum_{fils\ de\ t} (\frac{N_f}{N_t} \times I(f))$$

Avec $I(t)$ est l'impureté du nœud t avant la division ; N_f est le nombre d'observations dans le nœud fils f ; N_t le nombre d'observations dans le nœud t et $I(f)$ l'impureté du nœud fils f après la division. Grâce à cela, nous pouvons déterminer l'importances des différents nutriments pour établir le top 3 pour le modèle du producteur et du consommateur (figures 5 et 4). Ainsi, pour le modèle du producteur les 3 nutriments qui influencent le plus le modèle sont les graisses transformées, les graisses saturées et le calcium. Pour le consommateur ce sont les graisses saturées, le sel et l'énergie. Grâce à cela, on sait maintenant pourquoi e modèle du consommateur est moins précis que celui du producteur. En effet, la variable la plus importante augmente le plus l'erreur du modèle lorsqu'elle est supprimée. C'est bien ce qu'il se passe ici : les graisses transformées et le calcium absent du modèle du consommateur, leur absence augmente l'erreur du modèle.

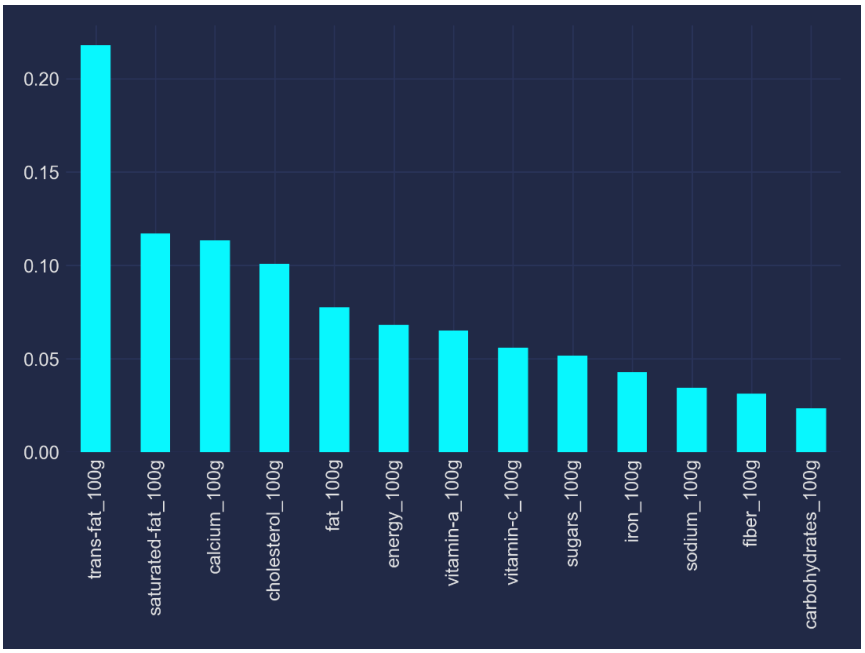


Figure 11 : Importance des variables du modèle du producteur.

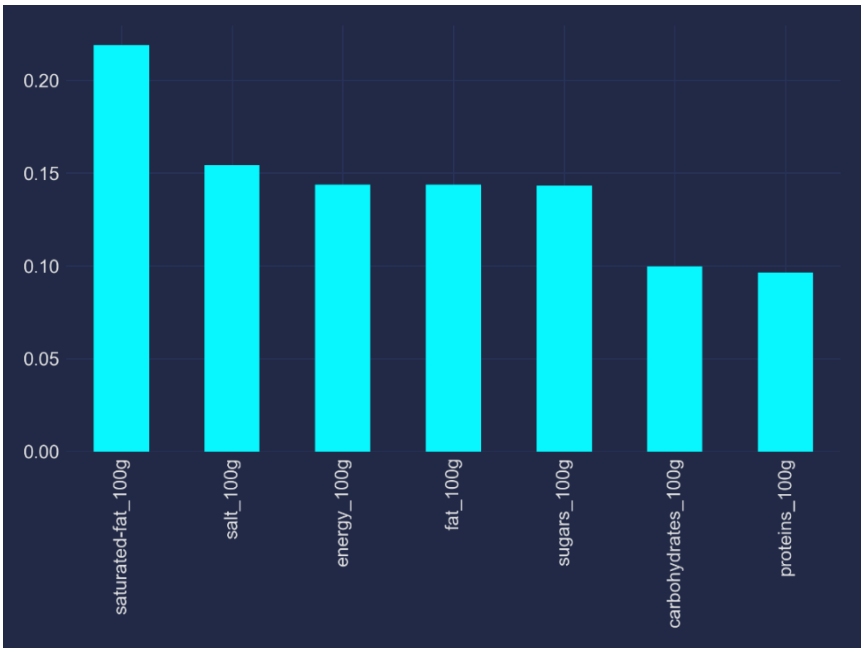


Figure 12 : Importance des variables du modèle du consommateur.

Enfin, lorsque l'on veut faire une prédiction d'un nouveau produit dont les informations seront rentrées dans le formulaire, nous avons besoin d'une fonction de décision notée $f(X)$. Cette fonction est transformée en fonction d'activation softmax (puisque l'on est dans le cas de la classification multi-classe) qui est exprimée ainsi :

$$P(classe\ i|X) := \frac{e^{f_{i(X)}}}{\sum_{i=1}^{C} e^{f_j(X)}}$$

Pour $i = 1,2, \dots, C$ où C est le nombre de classes. La classe prédite est souvent celle ayant la probabilité la plus élevée après l'application de la fonction d'activation : $Prédiction = argmax_i P(classe\ i|X)$.

TABLE DES FIGURES

TABLE DES TABLEAUX