HelbFour

- Rapport de projet -

Analyse de données et intelligences artificielle

Écrit par Jessy ADAM

Professeur : M. Riggio

Dall.E : very tall shopping aisle, dim lighting, hyper realistic, perspective

Introduction

Dans le cadre du cours d'Analyse de données et intelligence artificielle, il nous a été demandé d'aider la société HelbFour dans sa conquête du monde numérique. Pour cela, il a fallu analyser un jeu de données représentant les ventes de la société au fil des années et en tirer de la valeur en répondant à certaines questions.

Ce rapport passera sur la procédure que j'ai personnellement utilisé pour réaliser ce projet qui a pu nous initialiser au côté pratique de l'analyse de données. Celui-ci parcourera les technologies utilisées, le nettoyage des données, l'analyse de celles-ci avec les résultats, le feedback que le client a pu donner et les limitations du code écrit.

Technologies utilisées

Diverses technologies ont été utilisées pour la réalisation de ce projet, les principales étant Python et Microsoft Excel.

Langage utilisé

Par recommandation du professeur, j'ai utilisé Python pour écrire le programme. La raison principale étant que, comme les transactions du fichiers sont au format d'une liste Python, il est possible d'utiliser la fonction **eval()** pour les transformer d'une simple chaîne de caractères à une liste, ce qui facilite grandement le traitement des transactions.

Python est aussi un langage très versatile fait que c'est un langage de haut niveau, ce qui permet de faire des opérations comme l'ouverture de fichier, ou la conversion de variable ainsi que l'installation de librairies tierces plus facilement.

Python est aussi un langage très populaire ce qui rend la documentation facile d'accès et les librairies nécessaires à la réalisation du projet aussi.

En plus d'avoir utilisé Python comme langage j'ai aussi utilisé certains modules préinstallés pour faire certaines tâches :

- re - pour utiliser des expressions régulières dans le nettoyage des données;
- difflib - pour faire de la correction automatique sur les mot-clés erronés du fichier;
- threading - pour utiliser du multithreading pour faire une animation de chargement;
- time - pour pouvoir mettre le thread d'animation en pause, pour qu'il n'utilise pas toutes les ressources;
- random - pour choisir des animations au hasard

À part ça, je n'ai utilisé qu'un module tiers, openpyxl, afin de pouvoir utiliser des fichier xlsx pour générer les graphiques qui apparaîtront dans ce rapport.

Génération des graphiques

Pour la création des graphique eux-mêmes, j'utilise Microsoft Excel afin de pouvoir customiser les graphiques à ma guise sans devoir relancer le code constamment et en utilisant l'interface graphique qu'Excel offre.

Ne vous inquiétez pas, je ne mets pas les données manuellement dans le fichier, c'est le role d'openpyxl. J'ai fait un fichier Python qui me permet avec openpyxl de mettre les données dans le fichier dans un format correct que je peux ensuite utiliser pour les graphiques.

Outils de développement

Finalement comme autres outils de développements, j'ai utilisé Visual Studio Code comme IDE car c'est celui auquel je suis le plus habitué. Il est aussi assez simple d'utilisation et assez léger ce qui aide le développement sur mon portable.

Pour le versionning et la synchronisation de travail j'ai utilisé GitHub, car c'est ce que j'utilise depuis la première.

Nettoyage des données

La première étape à la réalisation du projet a été le nettoyage de données. On ne peut pas traiter n'importe quoi, non plus. Je vais donc expliquer le processus que j'ai suivi pour avoir un set de données "acceptable" à analyser.

Par manque de temps, je n'ai malheureusement pas pu traiter autant que ce que j'aurais voulu, mais je pense que les erreurs qui restent sont assez insignifiantes pour pouvoir l'ignorer dans le cadre du projet.

Lecture du fichier

Pour commencer, il fallait lire le fichier de fond en comble pour cela j'ouvre le fichier via la méthode **open()**. Je prend le nombre de lignes pour pouvoir lancer une animation pendant le traitement des données afin de savoir où on en est.

Après cela, j'utilise une boucle for pour lire le fichier. J'ai fait en sorte de pouvoir spécifier le nombre de lignes du fichier qu'il faut lire afin de pouvoir faire plus rapidement des tests.

Pendant la lecture, le code est divisé en deux grandes sections basées sur le fait que la ligne commence avec un crochet ou non. Si on commence avec un crochet, c'est une transaction à compter pour l'analyse, sinon il faut un traitement différent c'est notamment là qu'on effectue les changements de dates.

Séparation de lignes collées

Avant d'aller plus loin, il y a quelque chose d'important dans la manière dont je parcours le fichier que j'aimerais expliquer : Comment traiter deux choses différentes sur la même ligne, comme deux transactions ou une transaction et un changement de date ? C'est important à savoir pour comprendre un peu plus la manière dont j'ai procédé.

Pour faire cela, j'ai une boucle while imbriquée qui tourne jusqu'à ce que la ligne soit vide. Cette ligne devient vide car je traite chaque partie de la ligne séparément de manière consécutive. Et au fur et à mesure que je fais chaque partie, je l'enlève de la ligne pour pouvoir continuer.

Ceci fait que je dois bien définir ou commencer et arrêter avant de refaire la boucle.

Le traçage de la date

Après avoir mis en place la lecture du fichier, j'ai commencé par m'occuper de la date, car c'était une partie importante du set données et serait crucial pour catégoriser les informations que l'on tire du fichier.

Pour faire ça j'ai fait 3 variables : **year**, **week**, et **day**. Qui sont mises à jour au fur et à mesure de la lecture du fichier, à chaque fois que je retrouve les préfixes correspondants ("**YEAR:"**, "**WEEK:"**, et **"DAY:"**) qui sont aussi stockés dans des variables.

Pour cela, je prend ce qui est entre la fin du préfixe et le premier des symboles qui est toujours à la fin des lignes liées à la date, à savoir : *****, **+** ou **-**. Je retire ensuite de la ligne tout jusqu'au dernier de ces symboles.

Comptage des transactions

Juste après, j'ai fait un comptage rudimentaire des transactions faites en prenant du crochet ouvrant qui détermine si on rentre dans cette partie du code, jusqu'au prochain crochet fermant. S'il n'y a pas de crochet fermant ou qu'il se trouve après un autre crochet ouvrant, je l'ignore et passe directement à la suite, que se soit la fin de la ligne ou la prochaine transaction.

Je regarde ensuite si cette partie de la ligne peut passer dans la méthode **eval()** sans causer de soucis via un **try/except** caché dans une méthode pour éviter de couvrir d'autres erreurs. Si tout se passe bien, je fais le compte de la transaction dans des dictionnaires qui servent à compter les occurrences des produits en fonction de la date et au total. Pour que je puisse les afficher à la fin de la lecture du fichier.

Correction des fautes de frappes

J'ai ensuite remarqué que certains jours étaient mal orthographiés ce qui causait des problèmes au niveau du comptage, c'est là que j'ai découvert **difflib**, qui sert de correcteur automatique. Je l'ai donc utilisé pour corriger les jours en utilisant une liste de jours valides que j'ai stocké dans une variable.

J'ai aussi essayé de l'utiliser pour les produits mais j'ai fini par ne pas le faire car une faute de frappe et même sa correction peut facilement fausser le produit dont il s'agit.

Bien plus tard, j'ai aussi découvert que c'était le cas avec les préfixes de date, j'ai donc utilisé une tactique similaire pour vérifier s'il y avait une balise ou non, quand la ligne ne commençait pas par une transaction.

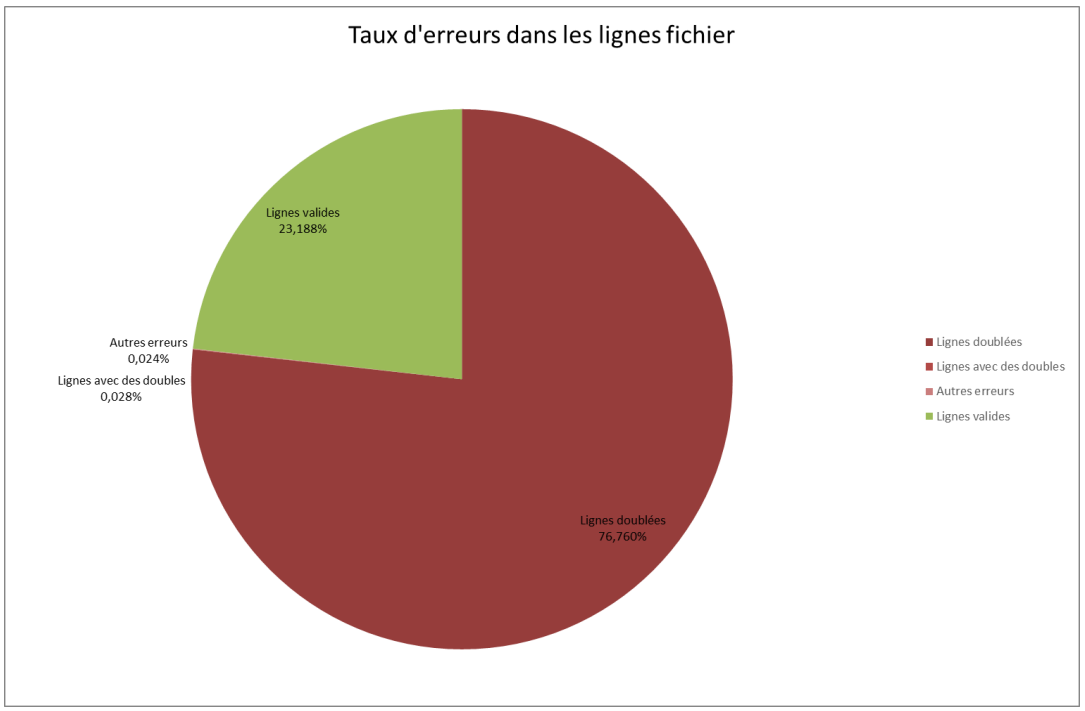
Les doublons et lignes “infinies”

Par après, en testant le programme j'avais remarqué qu'il restait bloqué sur certaines lignes pendant un relativement long moment. J'ai donc investigué et trouvé des horreurs au-delà de mon imagination : des lignes qui, à elles seules, faisaient plus de 100 ko composées d'un produit répété en boucle.

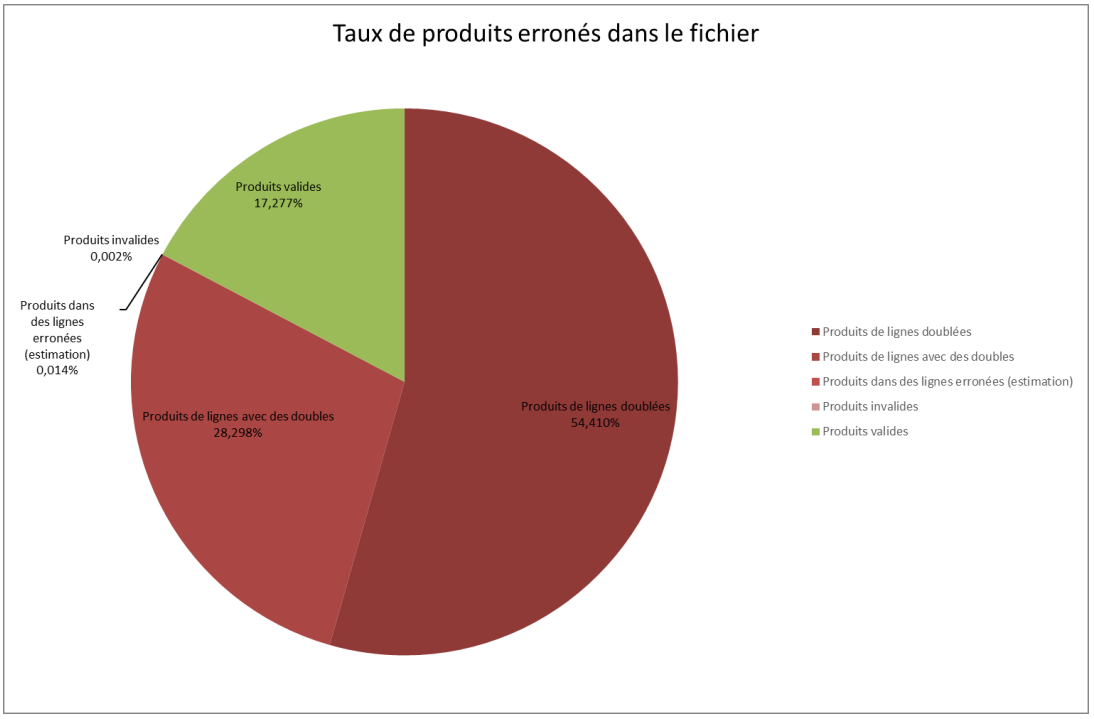
Pour pallier cela j'ai utilisé les sets, qui ne peuvent avoir qu'une fois la même chose, pour comparer les tailles. Si le set était plus petit, je le prenais à la place.

En plus de cela j'ai découvert que la même transaction pouvait se répéter en boucle aussi sur plusieurs lignes, ce qui pourrait sûrement fausser l'analyse une fois faite. J'ai donc utilisé une variable pour garder la dernière transaction faite et vérifier que l'on n'aie pas un doublon qui est passé à la place.

J’ai bien fait d’avoir remarqué ces bugs, car leur envergure est énorme, ils constituent la grande majorité des lignes et surtout des produits contenus dans le fichier, comme vous pouvez le voir ci-dessous.



Approximativement 76% des lignes du fichier font partie de murs de transactions toutes identiques les unes aux autres.



Et cela ne s’arrange pas quand on regarde les produits individuels car à cause des lignes “infinies”, ce chiffre monte à 82%, ce qui pourrait massivement changer la donne.

Complétion des transactions sans début

Ensuite, pour essayer d'exploiter le plus de données possibles, j'ai fait en sorte que quand une transaction n'a pas de début mais à une fin, je puisse le détecter et en ajouter un.

Au début, je l'ajoutais en plus de ce qui avait remplacé le crochet de début, ce qui causait une erreur lors de l'évaluation. J'ai donc fait en sorte de le mettre juste avant la première apostrophe, ce qui a grandement amélioré le traitement de ce type d'erreur.

Retirage des produits erronés

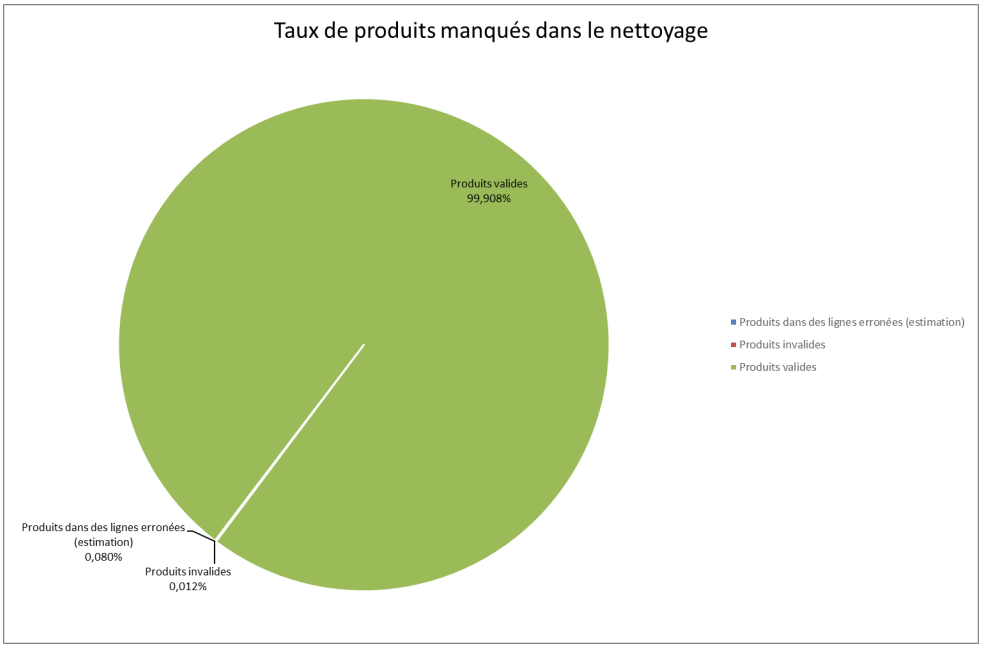
Après celà, j'ai essayé de sauver le plus de produits possible en utilisant une expression régulière pour enlever les caractères spéciaux dans les produits, ne laissant que les caractères : **p**, **_**, les chiffres et ce qui compose une liste. C'est un peu extrême mais ça sauve tout de même quelques produits.

S'il reste des produits qui ne sont pas exactement comme ceux que l'on a déjà, ils seront ignorés, ce qui est dommage mais cela reste une très petite partie des données. Néanmoins, on pourrait faire plus d'opérations pour conformer les produits au format que l'on utilise (**p_XX**), mais cela compliquerait les changements.

Je n'utilise pas **difflib** car les produits sont trop similaires sinon il serait intéressant de l'appliquer.

Conclusion du nettoyage

Je pense avoir fait du bon travail, étant donné que moins d'un dixième de pourcent des données sont irrécupérable pour l'instant, comme vous pouvez le voir dans le graphique ci-dessous. Il y a moyen de faire un travail plus en profondeur mais pour le temps consacré c'est suffisant car il fallait passer à l'analyse tôt ou tard.



Processus et résultats d’analyse

Il a fallu ensuite passer à l’analyse pour pouvoir tirer quoi que ce soit de nos données. Je vais donc d’abord expliquer mon procédé puis vous partager mes résultats.

Processus d’analyse

J’avais déjà fait des outils d’analyse rudimentaire pendant la phase de nettoyage pour avoir quelque chose à afficher à la fin.

J’ai fait un dictionnaire pour chaque chose à compter comme les jours, par exemple. J’ai utilisé des dictionnaires car je peux directement mettre le produit comme clé et savoir le nombre d'occurrences dans le fichier.

Comme toutes mes données sont mises dans des dictionnaires, j’ai donc fait des fonctions pour en afficher le meilleur et le pire, que ce soit produit, jour, semaine ou autre. J’ai aussi fait des fonctions pour afficher les résultats dans l’ordre, ou faire un top 10 (ou autre nombre).

Finalement j’ai fait un fichier qui utiliserait **openpyxl** pour mettre ces données dans un fichier xlsx pour que je puisse en faire des graphiques dans ce format ci :

```
# Wished structure
# TITLE   LABEL 1 LABEL 2 LABEL 3
# SERIE 1 DATA 11 DATA 12 DATA 13
# SERIE 2 DATA 21 DATA 22 DATA 23
# SERIE 3 DATA 31 DATA 32 DATA 33
```

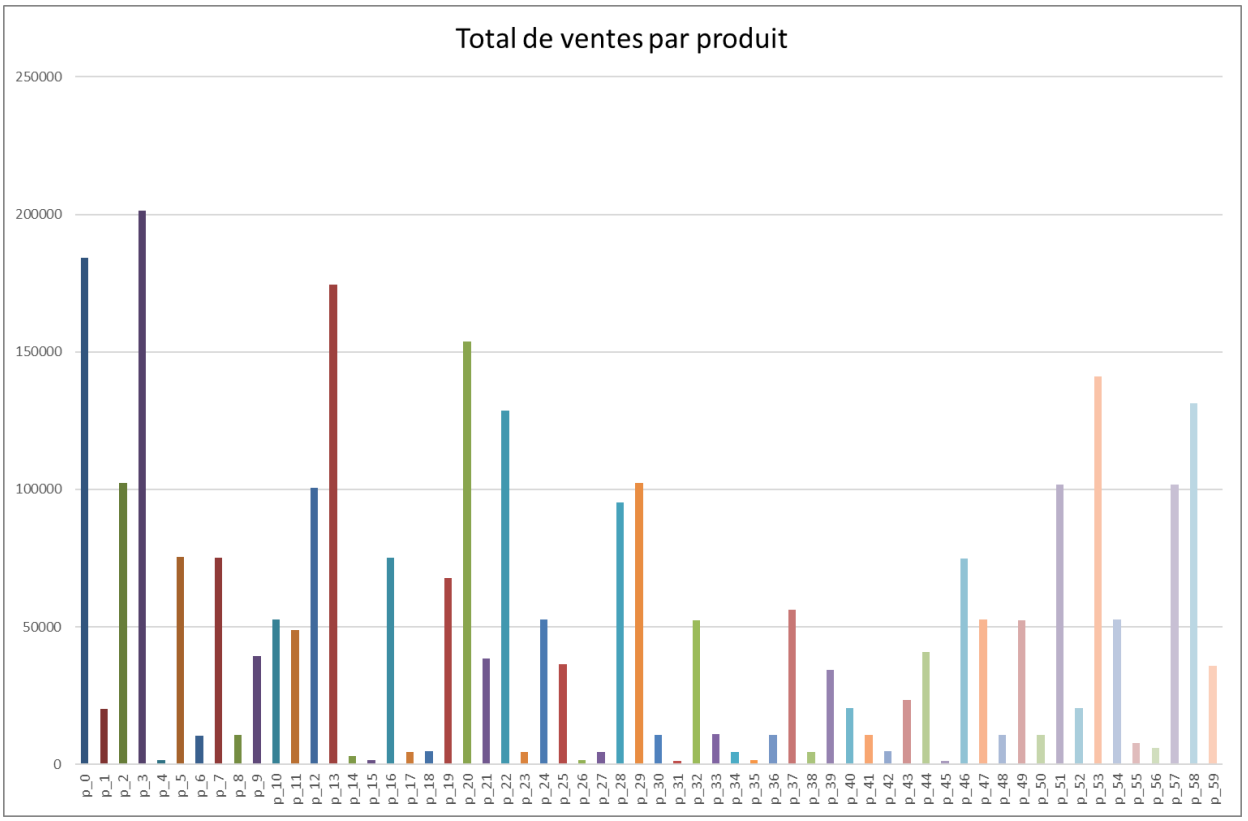
Je suis ensuite allé dans Excel a chaque fois que j’ai un nouveau graphique à faire pour le configurer, ce format là permettant de juste devoir sélectionner la zone des données pour facilement générer le graphique.

Résultats

Voici donc les résultats de toute mon analyse, j’ai essayé de faire des graphiques là où c’était pertinent. L’analyse aurait définitivement été plus en profondeur mais le temps est une denrée rare de nos jours. J’espère tout de même répondre à la plupart de vos questions.

Total des ventes

La première chose que j'ai analysée a été le nombre de ventes toutes années confondues :



Comme vous pouvez le voir les différences sont assez grandes avec des produits qu'on voit très peu et d'autres qu'on voit constamment.

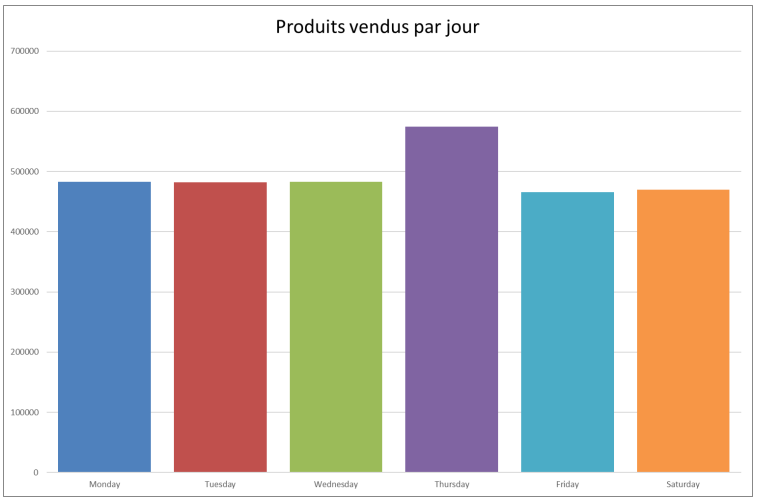
Le meilleur produit est le p_3 avec 201236 ventes, la suite du top 10 est :

- p_0 : 184304 ventes
- p_13 : 174502 ventes
- p_20 : 153723 ventes
- p_53 : 140973 ventes
- p_58 : 131167 ventes
- p_22 : 128526 ventes
- p_2 : 102342 ventes
- p_29 : 102302 ventes
- p_51 : 101770 ventes

Le produit ayant le moins vendu est le p_45 avec seulement 1320 ventes soit à peine 165 ventes par années en moyenne, ce qui n'est pas grand chose.

Vente de produits par jour

Ensuite, voyons la différences entre les jours de la semaine:



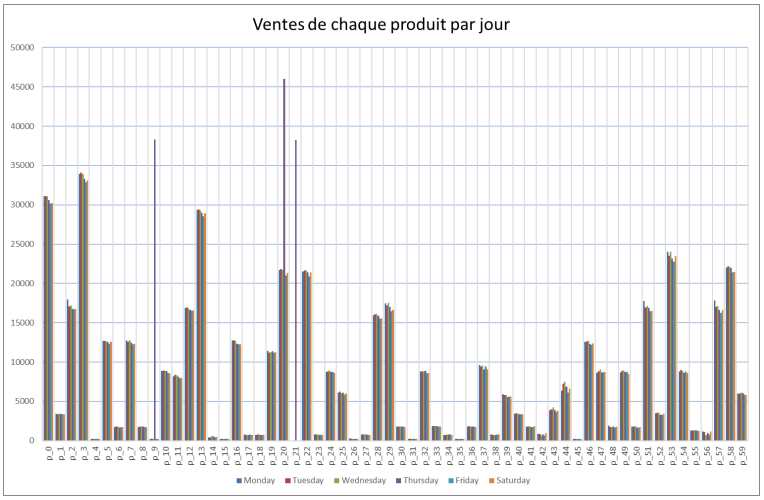
Comme vous pouvez le voir, le jeudi est le meilleur jour de la semaine, avec une assez grande marge en plus, avec 574790 ventes toutes années confondues.

Le reste des jours sont assez similaires mais le vendredi est le jour le moins lucratif avec seulement 465695 depuis 2014.

Par ordre décroissant nous avons donc :

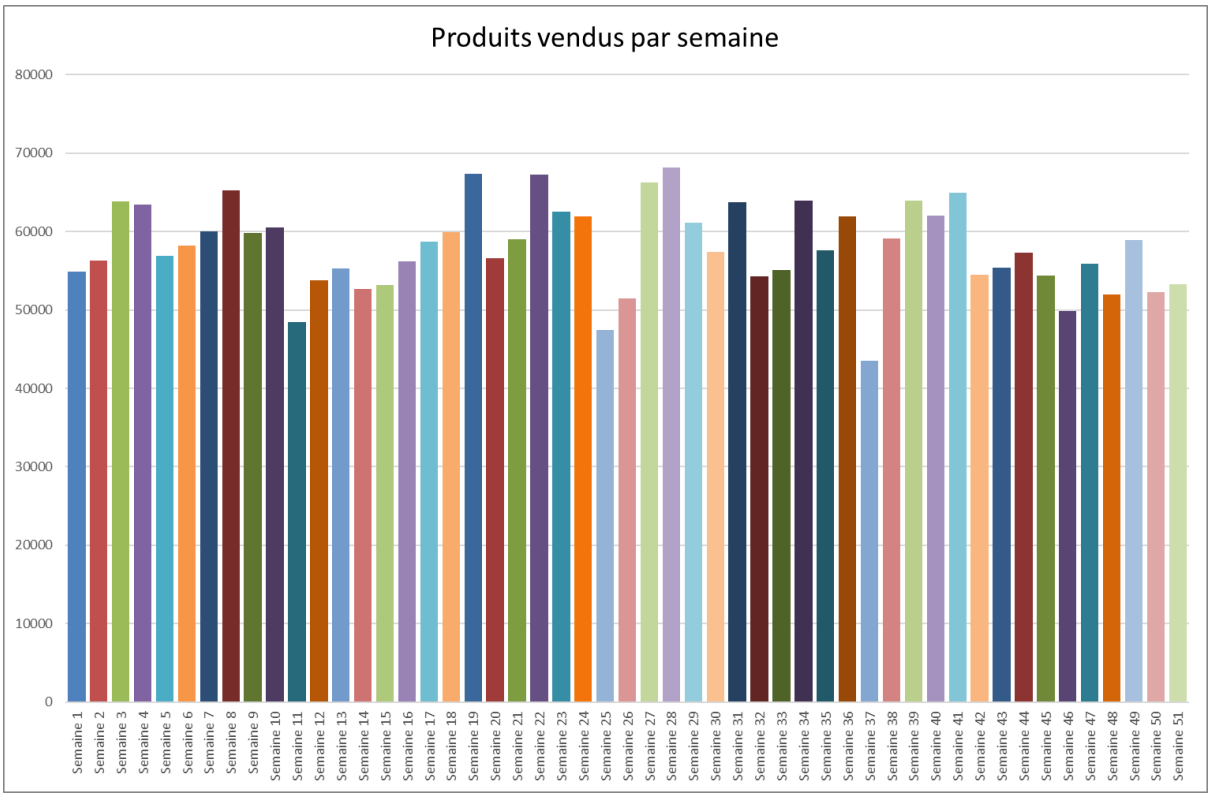
- Vendredi : 465695 ventes
- Samedi : 469897 ventes
- Mardi : 481805 ventes
- Lundi : 483011 ventes
- Mercredi : 483147 ventes
- Jeudi : 574790 ventes
-

Pour ce qui est des produits individuels, vous pouvez voir sur ce graphique un peu indigeste que pas grand chose ne change. A part pour l'énorme hausse en vente de p_9, P_20 et P_21, ce qui est intéressant.



Ventes de produits par semaine

Voici le nombre de produits par semaine, toutes années confondues :

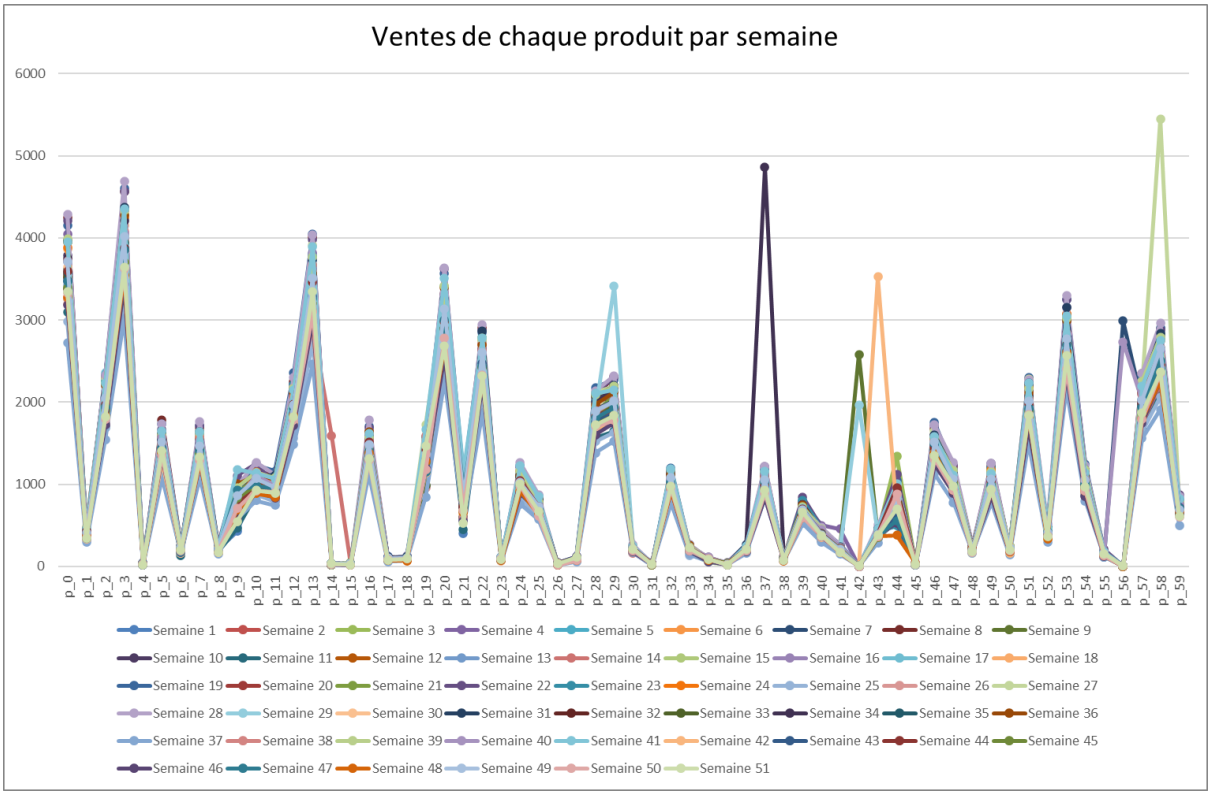


Vous pouvez voir qu'il y a une différence notable entre certaines semaines, notamment que la semaine de noël n'est pas particulièrement lucrative.

Mis à part ça les semaines les plus lucratives sont :

- Semaine 28 : 68185 ventes
- Semaine 19 : 67338 ventes
- Semaine 22 : 67246 ventes
- Semaine 27 : 66308 ventes
- Semaine 8 : 65285 ventes
- Semaine 41 : 64938 ventes
- Semaine 34 : 63940 ventes
- Semaine 39 : 63913 ventes
- Semaine 3 : 63885 ventes
- Semaine 31 : 63728 ventes

La pire semaine est la semaine 37 avec seulement 43567 ventes.

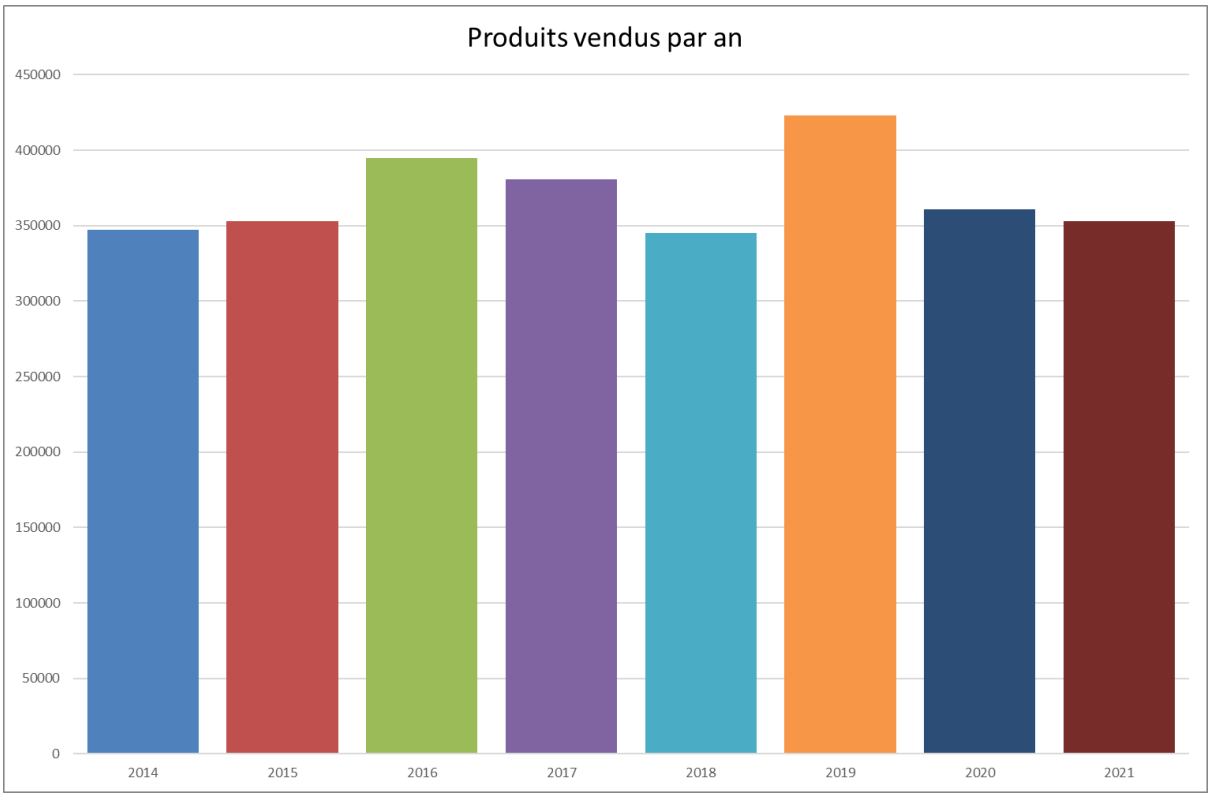


Pour les produits individuels, il n'y a pas beaucoup de différences à part :

- Le p_29 qui est beaucoup vendu en semaine 29 ainsi que;
- Le p_43 en semaine 42;
- Le p_37 en semaine 34;
- et surtout le p_58 en semaine 27.

Ventes de produits par années

Voici les produits vendus par années:

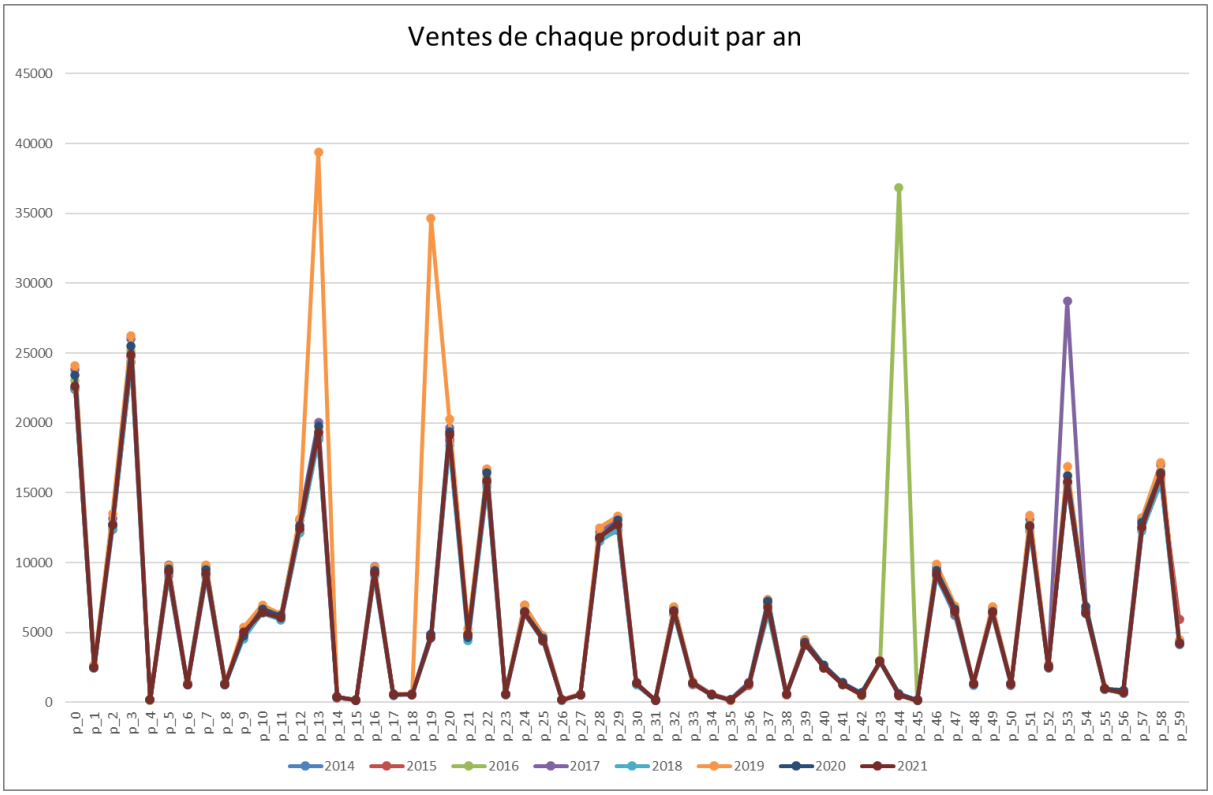


On peut voir sur le graphique que 2019 est l'année record avec 423063 ventes, jusqu'à ce que la pandémie fasse régresser les choses en 2020 et 2021.

Malgré cela, 2018 reste la pire années avec seulement 345396 ventes.

Voici le classements des années, par ordre décroissant :

- 2018 : 345396 ventes
- 2014 : 347261 ventes
- 2015 : 352852 ventes
- 2021 : 353068 ventes
- 2020 : 361010 ventes
- 2017 : 380845 ventes
- 2016 : 394850 ventes
- 2019 : 423063 ventes



Pour ce qui est des produits individuels par années, certains produits ont eu leur moment de gloire comme :

- Le p_54 en 2017;
- Le p_44 en 2016;
- et le p_13 et p_19 en 2019.

Combinaisons

Pour ce qui est des combinaisons, je les ai calculé jusqu'à des combinaisons de 5, voici les combinaisons les plus populaires pour chaque.

Combinaisons de 2

Pour les combinaisons de 2 les plus populaires sont :

- ['p_0', 'p_3'] : 147612 occurrences
- ['p_13', 'p_3'] : 142247 occurrences
- ['p_0', 'p_13'] : 138380 occurrences
- ['p_20', 'p_3'] : 133935 occurrences
- ['p_0', 'p_20'] : 131781 occurrences
- ['p_13', 'p_20'] : 130443 occurrences
- ['p_3', 'p_53'] : 88151 occurrences
- ['p_3', 'p_58'] : 83069 occurrences
- ['p_22', 'p_3'] : 81619 occurrences
- ['p_28', 'p_51'] : 81436 occurrences

Les plus pertinentes sont :

- p_21 menant à ['p_20', 'p_21'] donne une confiance de 99%
- p_21 menant à ['p_21', 'p_9'] donne une confiance de 99%

- p_55 menant à ['p_53', 'p_55'] donne une confiance de 98%
- p_9 menant à ['p_20', 'p_9'] donne une confiance de 98%
- p_9 menant à ['p_21', 'p_9'] donne une confiance de 96%
- p_20 menant à ['p_20', 'p_3'] donne une confiance de 87%
- p_20 menant à ['p_0', 'p_20'] donne une confiance de 85%
- p_28 menant à ['p_28', 'p_51'] donne une confiance de 85%
- p_20 menant à ['p_13', 'p_20'] donne une confiance de 84%
- p_13 menant à ['p_13', 'p_3'] donne une confiance de 81%

Combinaisons de 3

Pour les combinaisons de 3 les plus populaires sont :

- ['p_0', 'p_13', 'p_3'] : 130034 occurrences
- ['p_0', 'p_20', 'p_3'] : 127124 occurrences
- ['p_13', 'p_20', 'p_3'] : 126353 occurrences
- ['p_0', 'p_13', 'p_20'] : 125856 occurrences
- ['p_0', 'p_3', 'p_53'] : 66656 occurrences
- ['p_13', 'p_3', 'p_53'] : 63603 occurrences
- ['p_0', 'p_3', 'p_58'] : 62854 occurrences
- ['p_0', 'p_13', 'p_53'] : 61892 occurrences
- ['p_0', 'p_22', 'p_3'] : 61871 occurrences
- ['p_13', 'p_3', 'p_58'] : 60217 occurrences

Les plus pertinentes sont :

- p_21 menant à ['p_20', 'p_21', 'p_9'] donne une confiance de 99%
- p_9 menant à ['p_20', 'p_21', 'p_9'] donne une confiance de 96%
- p_20 menant à ['p_0', 'p_20', 'p_3'] donne une confiance de 82%
- p_20 menant à ['p_13', 'p_20', 'p_3'] donne une confiance de 82%
- p_20 menant à ['p_0', 'p_13', 'p_20'] donne une confiance de 81%
- p_13 menant à ['p_0', 'p_13', 'p_3'] donne une confiance de 74%
- p_13 menant à ['p_13', 'p_20', 'p_3'] donne une confiance de 72%
- p_13 menant à ['p_0', 'p_13', 'p_20'] donne une confiance de 72%
- p_0 menant à ['p_0', 'p_13', 'p_3'] donne une confiance de 70%
- p_0 menant à ['p_0', 'p_20', 'p_3'] donne une confiance de 68%

Combinaisons de 4

Pour les combinaisons de 4 les plus populaires sont :

- ['p_0', 'p_13', 'p_20', 'p_3'] : 124755 occurrences
- ['p_0', 'p_13', 'p_3', 'p_53'] : 58404 occurrences
- ['p_0', 'p_20', 'p_3', 'p_53'] : 57042 occurrences
- ['p_13', 'p_20', 'p_3', 'p_53'] : 56578 occurrences
- ['p_0', 'p_13', 'p_20', 'p_53'] : 56400 occurrences
- ['p_0', 'p_13', 'p_3', 'p_58'] : 55091 occurrences
- ['p_0', 'p_13', 'p_22', 'p_3'] : 54195 occurrences
- ['p_0', 'p_20', 'p_3', 'p_58'] : 53782 occurrences
- ['p_13', 'p_20', 'p_3', 'p_58'] : 53396 occurrences
- ['p_0', 'p_13', 'p_20', 'p_58'] : 53148 occurrences

Les plus pertinentes sont :

- p_20 menant à ['p_0', 'p_13', 'p_20', 'p_3'] donne une confiance de 81%
- p_13 menant à ['p_0', 'p_13', 'p_20', 'p_3'] donne une confiance de 71%
- p_0 menant à ['p_0', 'p_13', 'p_20', 'p_3'] donne une confiance de 67%
- p_3 menant à ['p_0', 'p_13', 'p_20', 'p_3'] donne une confiance de 61%
- p_21 menant à ['p_20', 'p_21', 'p_3', 'p_9'] donne une confiance de 55%
- p_9 menant à ['p_20', 'p_21', 'p_3', 'p_9'] donne une confiance de 53%
- p_21 menant à ['p_0', 'p_20', 'p_21', 'p_9'] donne une confiance de 50%
- p_9 menant à ['p_0', 'p_20', 'p_21', 'p_9'] donne une confiance de 49%
- p_55 menant à ['p_0', 'p_3', 'p_53', 'p_55'] donne une confiance de 48%
- p_21 menant à ['p_13', 'p_20', 'p_21', 'p_9'] donne une confiance de 47%

Combinaisons de 5

Pour les combinaisons de 2 les plus populaires sont :

- ['p_0', 'p_13', 'p_20', 'p_3', 'p_53'] : 55928 occurrences
- ['p_0', 'p_13', 'p_20', 'p_3', 'p_58'] : 52711 occurrences
- ['p_0', 'p_13', 'p_20', 'p_22', 'p_3'] : 51841 occurrences
- ['p_0', 'p_13', 'p_2', 'p_20', 'p_3'] : 41173 occurrences
- ['p_0', 'p_13', 'p_20', 'p_29', 'p_3'] : 41142 occurrences
- ['p_0', 'p_13', 'p_20', 'p_3', 'p_51'] : 40850 occurrences
- ['p_0', 'p_13', 'p_20', 'p_3', 'p_57'] : 40817 occurrences
- ['p_0', 'p_12', 'p_13', 'p_20', 'p_3'] : 40410 occurrences
- ['p_0', 'p_13', 'p_20', 'p_28', 'p_3'] : 38198 occurrences
- ['p_0', 'p_13', 'p_28', 'p_3', 'p_51'] : 34325 occurrences

Les plus pertinentes sont :

- p_15 menant à ['p_0', 'p_13', 'p_15', 'p_20', 'p_3'] donne une confiance de 0.43%
- p_25 menant à ['p_0', 'p_13', 'p_20', 'p_25', 'p_3'] donne une confiance de 0.42%
- p_26 menant à ['p_0', 'p_13', 'p_20', 'p_26', 'p_3'] donne une confiance de 0.42%
- p_55 menant à ['p_0', 'p_13', 'p_3', 'p_53', 'p_55'] donne une confiance de 0.42%
- p_55 menant à ['p_0', 'p_20', 'p_3', 'p_53', 'p_55'] donne une confiance de 0.41%
- p_8 menant à ['p_0', 'p_13', 'p_20', 'p_3', 'p_8'] donne une confiance de 0.40%
- p_32 menant à ['p_0', 'p_13', 'p_20', 'p_3', 'p_32'] donne une confiance de 0.40%
- p_55 menant à ['p_0', 'p_13', 'p_20', 'p_3', 'p_55'] donne une confiance de 0.40%
- p_55 menant à ['p_13', 'p_20', 'p_3', 'p_53', 'p_55'] donne une confiance de 0.40%
- p_36 menant à ['p_0', 'p_13', 'p_20', 'p_3', 'p_36'] donne une confiance de 0.40%

Feedback du client

Le 28 novembre, j'ai eu l'occasion de rencontrer le client et de lui poser des questions avec des collègues. Voici le feedback que j'ai pu en tirer et les impacts que ça a eu sur le projet :

- Les lignes infinies sont, en effet, un bug mais il faudrait les compter pour les produits uniques qui sont présents. Je les ignorais totalement auparavant donc c'est une bonne correction à avoir.
- Pour les blocs de lignes ont devrait pouvoir garder les X premiers, au choix et démontrer la différence que changer cette limite peut faire. Je n'ai malheureusement pas eu le temps d'implémenter cela mais ça reste une amélioration future à faire.
- Il faut exploiter un maximum de données. Ceci à fait que j'ai essayé de catégoriser le type d'erreur que je n'arrivais pas à corriger dans des fichiers séparés pour analyse. j'ai aussi implémenté la complétion de début de tableau vers la fin de projet grâce à cette remarque.
- Privilégier les graphiques pour les choses longues à expliquer, ceci à confirmer mon envie de faire le plus de graphiques possible mais le manque de temps m'a empêché d'en faire beaucoup malheureusement.

Limitations et développements futurs

Comme tout projet, celui-ci vient avec ses limitations et possibilités d'améliorations je vais donc les parcourir ici.

Limitations

Premièrement, ce programme est vraiment calqué sur la structure actuelle du fichier. Si celle-ci venait à changer, il y aurait beaucoup de refactorisations à faire pour l'adapter au nouveau format.

Deuxièmement, ceci n'est pas une IA en train d'apprendre constamment, il y a sûrement plus de types d'erreurs que je n'ai pas vu venir et qui pourraient venir tout casser, en plus de celles que j'ai dû ignorer.

Aussi, ce programme demande d'avoir une liste bien orthographiée et maintenue des produits, jours, et autres facteurs de comparaisons, car c'est sur cela que se base une bonne partie des vérifications.

Ensuite, les combinaisons sont calculées pendant la lecture du fichier, ceci fait qu'on ne peut pas utiliser une limite de support pour optimiser les combinaisons dont on garde une trace, ce qui fait que le programme est probablement plus lent que ce qu'il devrait être.

Après ça, le programme n'est pas le plus optimisé, ni horriblement propre. Bien que j'ai essayé d'éviter des erreurs aberrantes, il y a sûrement beaucoup de moyens d'optimiser le code et de le découpler plus que ce que j'ai fait.

Finalement, comme j'ai beaucoup de dictionnaires, il se peut que ce programme prenne beaucoup de mémoire, surtout avec un fichier avec un nombre de données plus proche du réel, mais cela reste à voir.

Améliorations futures

Pour ce qui est des améliorations, il y a différentes directions où nous pouvons aller, le nettoyage et l'analyse.

Nettoyage

Comme tout n'a pas été traité, une amélioration évidente serait de corriger le reste des bugs, cela va de soi. Cela permettrait d'avoir une analyse plus complète et véridique.

Analyse

Mais c'est surtout au niveau de l'analyse qu'il y a des améliorations à faire.

D'abord les combinaisons sont calculées à la volée pendant la lecture du fichier. Une amélioration serait de les calculer après coup pour pouvoir le faire plus proprement et en utilisant la notion de support.

Ensuite, la pertinence, pour l'instant, n'est calculée que d'un produit singulier. Un amélioration serait de pouvoir la pertinence entre combinaisons, par exemple : [‘p_1’, ‘p_2’] dans [‘p_1’, ‘p_2’, 'p_ 20']

Finalement, il y a toujours plus d'analyse à faire, comme la moyenne de produit dans une transaction, les top 10 des pires produits, la meilleure journée de tous les temps et j'en passe. On peut toujours y tirer plus d'informations.

Conclusion

Je suis content du travail que j'ai pu faire sur ce projet. C'était une bonne introduction au monde du data analysis et ça m'encourage encore plus à faire attention à ce que j'écris dans les formulaire que je remplis.

J'aurais aimé faire plus pour mais, comme je l'ai dit auparavant, un manque d'organisation d‘un côté et une montagne de projets de l'autre m'ont laissé avec assez peu de temps. Le projet en est quand même resté passionnant et intéressant jusqu'au bout.

Je vous remercie pour ce projet et d'avoir été un si bon professeur pendant mes études. Je vous remercie de votre temps de lecture et de votre clémence d'avoir accepté de recevoir ce projet en retard. Je m'excuse, j'essaierai de faire mieux la prochaine fois.