UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

M1 - Économétrie, Statistique

Projet encadré par Madame Imane Loukah,

Hella BOUHADDA
Charlotte CEGARRA
Manal GHORAFI
Aya MOKHTAR

Table des matières

I - INTRODUCTION

Les MarketPlace, ces plateformes Internet mettant à disposition des espaces de vente de services ou de produits à des clients, font face à un succès grandissant ces dernières années. Aujourd'hui ces dernières sont devenues plus qu'incontournables dans le paysage du e-commerce. En France, ces dernières façonnent la majorité du top 10 des e-commerce les plus visités. Leur analyse est donc plus qu'essentielle pour appréhender les évolutions marketing & commerciales actuelles.
Parmi ces dernières, nous retrouvons Amazon, cette entreprise américaine , faisant partie des grands "géants de l'e-commerce" à amasser pas loin de 135 milliards de dollars de chiffre d'affaires cette année avec plus de 240 millions de clients à son actif. Ceci découle d'une ascension exponentielle du e-commerce au fil du temps.

Cependant, il est important de notifier que la majorité des produits d'Amazon sont vendus par des tiers ainsi sur la plate-forme, les différents fournisseurs se font la guerre vendant les mêmes produits à des prix différents, en jouant parfois la qualité ou sur d'autres critères tout aussi importants. Nous le comprenons donc, Amazon est une plate-forme vaste dans laquelle le traitement donné est plus que nécessaire.

En effet, la plate-forme doit nécessairement pouvoir proposer des offres adaptées aux besoins de chaque client. De plus, les récentes études marketing témoignent de l'intérêt de proposer des expériences uniques à chaque consommateur. Et pour finir pour le bien du développement de l'affaire, les équipes se doivent de comprendre au mieux les dessous de la plateforme. Tout ceci serait impraticable sans l'analyse de données mais néanmoins plus que nécessaires.

Finalement, Tous ces enjeux basés sur ses Marquette Place nous ont poussait à choisir Amazon comme sujet de notre projet afin de nous intéresser à une partie de cette analyse et de comprendre quels sont les différents rouages analytiques se cachant derrières cette marketplace.

II - ANALYSES ET INTERPRÉTATIONS

i. Les Données

Pour réaliser ces Projets nous avons utilisé 2 bases de données provenant de Kaggle. Ces deux Datasets recensent des informations récentes sur les produits vendus par Amazon. Mais, avant de les manipuler, il est nécessaire d'en comprendre les composantes et les éventuelles failles.

i.a Descriptions des données:

Les deux datasets mis à notre disposition se nomment « amazon_categories.csv » et « amazon_products » il s'agit de deux fichiers Excel pour en faire une brève description nous avons créé la fonction suivante:

```
def description(df):
    print("Le fichier possède {} observations et {} variables"\
          .format(df.shape[0], df.shape[1]))
    print("\nVoici les différentes colonnes du dataframe: ")
    print(df.columns)
    print("\nVoici les noms et types de variables :")
    print(df.dtypes)
    return

description(A_categories)
description(A_products)
```

Dans un premier temps, faisons une brève description de cette fonction. Dans la première ligne on affiche le premier et le second élément du tuple donnant la taille du data frame, soit le nombre de lignes et de colonnes correspondant respectivement au nombre de variables et d'observations.

On affiche ensuite le nom de chacune des colonnes soit le nom de chacune des variables ainsi que leur type ce qui nous permettra de mieux procéder pour la suite.

Finalement, on en conclut les information suivantes pour les deux bases de données: pour la base de données contenant les catégories, le fichier comporte 248 observations et 2 variables. La première avec les identifiants de chaque catégorie (type : int64) et la seconde avec les noms (type : object) de celles-ci.

Libellé de la variable	Définition
id	Identifiant de la catégorie du produit
category_name	Libellé de la catégorie du produit

Pour la seconde base de données contenant les produits, on en conclut qu'elle comporte 1426337 observations et 11 variables nommées, 'asin', 'title', 'imgUrl', 'productURL', 'stars', 'reviews', 'price', 'listPrice', 'category_id', 'isBestSeller', 'boughtInLastMonth' dont les types sont respectivement les suivants: object , object, object, object, float64, int64, price, float64, float64, int64, bool et int64.

Libellé de la variable	Définition
asin	Identifiant Amazon du produit
title	Libellé du produit
imgUrl	URL de l'image du produit
productURL	URL du produit sur Amazon
stars	Note du produit. Si la valeur est égale à 0, cela signifie qu'aucune note n'a été trouvée
reviews	Nombre d'avis (commentaires). Si la valeur est égale à 0, cela signifie qu'aucun avis n'a été trouvé
price	Prix du produit en USD. Si la valeur est égale à 0, cela signifie que le prix n'était pas disponible
listPrice	Prix original du produit, avant réduction. Si la valeur est égale à 0, cela signifie qu'aucune réduction n'est en cours sur le produit
category_id	Identifiant de la catégorie du produit
isBestSeller	True si le produit a eu le label « BestSeller » False sinon
boughtInLastMonth	Nombre de produits vendus le mois dernier, selon Amazon

i.b Visualisation des données

Après avoir fait une brève description des bases de données, il paraît nécessaire de visualiser plus de détails sur la base en imprimant quelques caractéristiques des deux bases de données : on a utilisé dans notre cas la fonction describe()

```
def stats_descrip(df):
    return df.describe()
stats_descrip(A_categories)
stats_descrip(A_products)
```

Cette fonction prend en entrée une table et lui applique la fonction describe() pour avoir toutes les statistiques descriptives sur les variables numériques telles que la moyenne, l'écart-type, les valeurs minimales et maximales, les quantiles etc. Voici ci-dessous le résultat obtenu en appliquant cette fonction sur la dataframe A_products qui correspond à la base de données Amazon_products:

	stars	reviews	price	listPrice	category_id	boughtInLastMonth
count	1.426337e+06	1.426337e+06	1.426337e+06	1.426337e+06	1.426337e+06	1.426337e+06
mean	3.999512e+00	1.807508e+02	4.337540e+01	1.244916e+01	1.237409e+02	1.419823e+02
std	1.344292e+00	1.761453e+03	1.302893e+02	4.611198e+01	7.311273e+01	8.362720e+02
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00
25%	4.100000e+00	0.000000e+00	1.199000e+01	0.000000e+00	6.500000e+01	0.000000e+00
50%	4.400000e+00	0.000000e+00	1.995000e+01	0.000000e+00	1.200000e+02	0.000000e+00
75%	4.600000e+00	0.000000e+00	3.599000e+01	0.000000e+00	1.760000e+02	5.000000e+01
max	5.000000e+00	3.465630e+05	1.973181e+04	9.999900e+02	2.700000e+02	1.000000e+05

On constate ici que la valeur minimale de la variable « stars » est de 0 et la valeur maximale est de 5. Rien d'étonnant pour cette variable discrète donc. La moyenne de cette variable est de 3,99, on peut en conclure que la grande majorité des produits constituant ce Dataframe sont assez bien noté. Ça médiane est quant à elle de 4,4 ce qui reste correcte.
Pour les prix cependant, moyenne et médiane sont tout de même plus éloigné ce que l'on constate également à la lecture de l'écart type. Une étude plus approfondit de ces derniers pourrait donc être intéressent mais nous verrons cela plus en détail par la suite.
Concernant mes autres variables, leurs interprétations sont légèrement plus délicates et les mettre en contexte pourrait être plus prenant ce que nous verrons plus tard dans le rapport.

i.c Nettoyage de nos bases de données

Le nettoyage des DataFrame est cruciale pour une analyse de données fiables. La présence d'erreurs pourrait entraîner une perte d'information consequante pouvant ainsi entrainer des résultats statistiques faux. Il est donc important de vérifier ces valeurs pour garantir la fiabilité de nos résultats.

Voici comment nous avons decidé de procéder dans le module « Data_Clean » :

```python
def nettoyage(df):
    print(df.isna().sum())

    df = df.fillna("missing")

    if 'id' in df.columns:
        df = df.rename(columns={'id': 'category_id'})

    if 'title' in df.columns:
        df['title'] = df['title'].str.lower()

    if 'category_name' in df.columns:
        df['category_name'] = df['category_name'].str.lower()

    for col in df.columns:
        df = df.rename(columns={col: col.lower()})

    df = df.drop_duplicates()

    return df
```

Nous avons commencé par définir une fonction nommé nettoyage. Elle permet de nettoyer un dataframe, plus précisément, elle en compte les données manquantes et en retire les majuscules ainsi que les doublons.
En appelant la fonction pour nos deux bases de données initiale nous nous sommes rendu compte que la seule donnée manquante était le titre d'un des produits. Vu que nous possédions l'url de celui-ci nous sommes directement aller récupérer le titre sur Amazon afin de complet le vide de notre base de données.
En effet, nous avons préféré procéder ainsi pour afin de ne pas fausser nos résultats par la suite.

III - Les analyses demandées en première partie des sujets avec interprétations

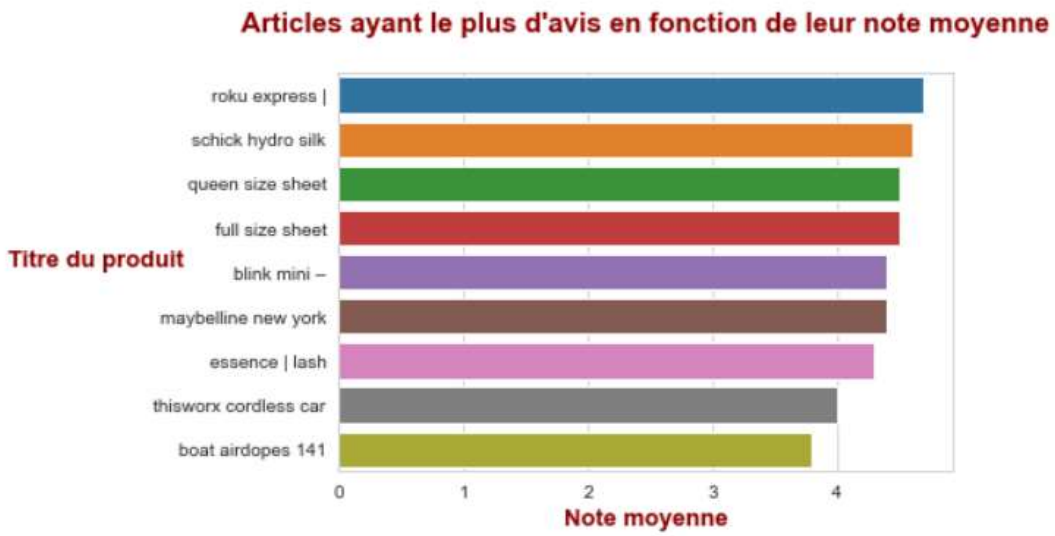
i. Quels sont les 10 articles ayant le plus d'avis ? Afficher graphiquement leur note moyenne respective par ordre décroissant.

Pour répondre à cette première question, nous avons créé un DataFrame en triant les données par le nombre d'avis ("Reviews"). Les colonnes "category_name" et "title" ont été ajustées pour afficher uniquement les trois premiers mots et ne pas rendre le graphique illisible. Par la suite, nous avons formé un nouveau DataFrame comprenant les colonnes "title", "reviews", et "category_name" la colonne "title" ayant été définie comme index principal pour ne pas alourdir la visualisation des résultats. Puis nous avons affiché les 10 premières lignes du résultat, présentant les titres, le nombre d'avis, et les catégories respectives. Enfin, ces dix lignes ont été réorganisées en fonction de la note moyenne ('stars') de manière décroissante.

Tout d'abord, nous pouvons voir que l'article qui a le plus grand nombre d'avis est "Essence | lash" qui compte 346 563 avis. Il fait partie de la catégorie "Makeup". Puis, nous avons le produit "thisworx cordless car" (avec 292 474 avis) qui appartient à la catégorie "vacuum cleaners", le produit "queen size sheet" (avec 281 661 avis) qui appartient à la catégorie "kids' home store", le produit "full size sheet" (avec 281 661 avis) qui appartient également à la catégorie "kids' home store", le produit "blink mini" (avec 260 658 avis) qui appartient à la catégorie "smart home : security", le produit "maybelline new york" (avec 194 659 avis) qui appartient à la catégorie "makeup", le produit "schick hydro silk" (avec 183 726 avis) qui appartient à la catégorie "shaving & hair", le produit "roku express" (avec 177 760 avis) qui appartient à la catégorie "televisions & video". Enfin, le dixième article ayant le plus d'avis est le "boat airdopes 141" (avec 174 524 avis) qui appartient à la catégorie "headphones & earbuds".
Le nombre d'avis élevés montre que ces articles sont très populaires auprès des consommateurs.

title	reviews	category_name
essence | lash	346563	makeup
thisworx cordless car	292474	vacuum cleaners &
queen size sheet	281661	kids' home store
full size sheet	281661	kids' home store
queen size sheet	281661	kids' home store
blink mini –	260659	smart home: security
maybelline new york	194051	makeup
schick hydro silk	183726	shaving & hair
roku express |	177760	televisions & video
boat airdopes 141	174524	headphones & earbuds

Voici le graphique obtenu :



- <u>Interprétations des résultats :</u>

Nous avons choisi de représenter les dix produits ayant le plus d'avis, en fonction de leur note moyenne avec un diagramme en barre. On peut voir que le produit qui a le plus d'avis n'est pas celui qui a la note moyenne la plus haute. Le produit "roku express", avec une note d'environ 4,6/5, est le produit qui a la note moyenne la plus élevée. Or, ce produit était classé 9e parmis les 10 produits qui avaient le plus d'avis. De plus, le produit "thisworx cordless car" était le 7e produit avec le plus d'avis et il est désormais classé à la deuxième position lorsque l'on tient compte de sa note moyenne.
Cependant, le produit qui a la note moyenne la moins élevée est celui qui avait le moins d'avis parmi les dix. Globalement, les dix produits ont une note comprise entre 3,6/5 et 4,6/5, ce qui montre que les produits ayant le plus d'avis ont généralement des bonnes notes. On observe donc une corrélation entre le nombre d'avis et la note moyenne du produit.

ii. Calculer le prix moyen des produits par catégorie. Afficher graphiquement les prix moyens pour les 15 catégories les plus chères.

```
merge = fusion.groupby(["category_id","category_name"]).agg({'price': 'mean'}).reset_index()
merge=merge.set_index('category_id')
merge['category_name'] = merge['category_name'].str.split().str.slice(0, 3).str.join(' ')
merge = merge.sort_values(by="price",ascending=(False))


sns.barplot(x="category_name",y='price', data=merge.head(15))
plt.xticks(rotation='vertical')
plt.show()
```

Dans cette question, on a créé une variable merge qui prend le résultat du regroupement de la DataFrame fusion en fonction des colonnes “category_id” et “category_name” en calculant la moyenne de prix pour chaque groupe. Ensuite on a changé l’index du dataframe pour qu'il soit indexé sur la colonne “category_id”. Pour ne pas avoir des longs noms de catégorie on a procédé à diviser le contenu de la colonne “category_name” pour avoir que les 3 premiers mots pour plus de visibilité et de clarté.
Après avoir les 3 premiers mots de chaque observation de “category_name” on trie la DataFrame par prix de manière décroissante. Ainsi on obtiendra le prix moyen des produits par catégorie.

Pour l’affichage des prix pour les 15 catégories les plus chères, on effectue un barplot en utilisant la librairie seaborn de python en mettant en abscisse “category_name” et en ordonné “price”
Voici le résultat obtenu:

- <u>Interprétation des résultats:</u>

Le graphique représente le prix moyen des produits pour les 15 catégories les plus chères. Ce graphique est un diagramme en bâton avec l'axe des abscisses représentés par les différentes catégories et l'axe des ordonnées représentés par les prix

Globalement, on remarque un écart significatif entre le prix moyen des produits de la catégorie "computer servers" et les autres catégories. En effet, le prix moyen des produits de cette catégorie se comptabilisent à environ 1 550€ alors que le prix moyen des produits de la deuxième catégorie la plus chère se situe aux alentours de 550€. On observe donc un écart de plus de 1000€ entre la catégorie "computer servers" et "computers and tablets", ce qui n'est pas très étonnant au vu des prix des appareils électroniques dû à leur coût de production et de la recherche pour y aboutir. Mais aussi on peut apercevoir l'étendu et la disparité des prix en comparant la première et la dernière catégorie (soit computer servers et smart home : news) qui comptabilise un écart de 1400€.

On peut donc scinder ce graphique en trois groupes :

- la catégorie computer servers qui comptabilise les prix moyens les plus chers soit 1550€
- la catégorie computer & tablets et smart home (home) qui comptabilisent en moyenne 400€ de prix moyens des produits
- Puis tout le reste soit les catégories: luggage sets, smart home (vacuums), computer monitors, computer external components, video projectors, camera & photo, data storage, smart home (smart, thermostats, lawn et new)

Soit en moyenne, les prix moyens de ces catégories sont de 200€.

On peut donc s'apercevoir que l'écart des prix moyens entre les catégories sont parfois du double (200€ à 400€) au quadruple (400€ à 1500€) ce qui montre une grande disparité des prix entre les catégories.

iii. Quels sont les produits Best Seller ayant une note inférieure à 4/5 ? Afficher la distribution des prix pour ces produits. Quels sont les 5 qui ont été le plus vendus ? Combien d'unités ?

- Explication du code :

Ce code crée un graphique de boîte à moustaches pour la distribution des prix des produits best-sellers ayant une note inférieure à 4/5, en utilisant les bibliothèques seaborn et matplotlib.

En premier temps, on filtre les données en sélectionnant les lignes où la colonne "isbestseller" est vraie et la colonne "stars" est inférieure à 4 et on réinitialise l'index de notre dataframe trois. Ensuite on modifie la colonne « title » en conservant les 3 premiers mots de chaque libellé de produit pour plus de lisibilité. A l'aide de la bibliothèque seaborn (sns) on crée un graphique de boîte à moustaches. Les données utilisées sont celles du DataFrame "trois", où la constante est ajoutée à chaque ligne pour créer une seule boîte à moustaches. Enfin, on affiche le résultat avec plt.show().

- Interprétation :

Ce graphique représente un boxplot des prix des produits Best Sellers ayant des notes inférieures à 4 sur 5. D'après le graphique obtenu on pourra constater que les produits Best Sellers ayant obtenu une note inférieure à 4 sur 5 affichent un prix maximal environ de 66 USD. Il est à noter que la dispersion des prix apparaît notable, comme illustré par le graphique. En effet, la médiane des prix, représentant la valeur au centre de la distribution des produits Best Sellers avec une note inférieure à 4 sur 5, se situe aux environs de 22 USD. Ce point médian, qui divise équitablement les prix en deux parties égales, semble se rapprocher du prix minimum qui est égale à 0. Toutefois, la présence de quelques valeurs extrêmes peut accentuer cette dispersion des prix.
En effet, une explication plausible de cette tendance pourrait être que les entreprises choisissent de réduire légèrement leurs prix pour les produits devenus best-sellers. Cette stratégie vise à stimuler la demande en attirant davantage de consommateurs, ce qui, à son tour, pourrait générer des retours positifs et accroître la notoriété des produits.

title	boughtinlastmonth
hismile v34 colour	100000
super mario bros.™	20000
nad's facial wax	20000
pure instinct roll-on	20000
hair catcher durable	20000

- Interprétation :

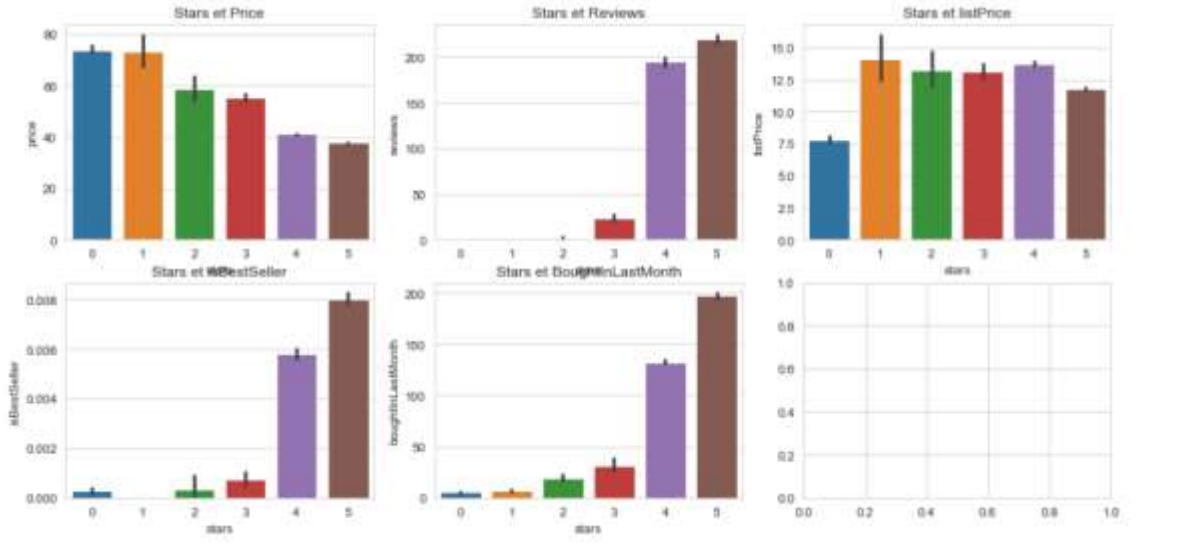
Ce tableau illustre les 5 produits ayant une note inférieure à 4 sur 5. Parmi les cinq premiers produits, quatre ont été vendus de manière similaire et il y a un de ces produits qui se distingue des autres. Le produit « hismile hismile v34 colour » a été vendu 5 fois plus que les autres (100000/20000).

iv. Quels sont les 5 qui ont été le plus vendus ? Combien d'unités ?

```
fusionS=fusion

fusionS['stars'] = fusionS['stars'].round().astype(int)

# Créer une seule figure et plusieurs sous-graphiques
fig, axes = plt.subplots(2, 3, figsize=(15, 7))

# Tracer les barplots sur les sous-graphiques spécifiques
sns.barplot(x='stars', y='price', data=fusionS, ax=axes[0, 0])
axes[0, 0].set_title('Stars et Price')

sns.barplot(x='stars', y='reviews', data=fusionS, ax=axes[0, 1])
axes[0, 1].set_title('Stars et Reviews')

sns.barplot(x='stars', y='isbestseller', data=fusionS, ax=axes[1, 0])
axes[1, 0].set_title('Stars et isBestSeller')

sns.barplot(x='stars', y='boughtinlastmonth', data=fusionS, ax=axes[1, 1])
axes[1, 1].set_title('Stars et BoughtInLastMonth')

sns.barplot(x='stars', y='listprice', data=fusionS, ax=axes[0, 2])
axes[0, 2].set_title('Stars et listPrice')

plt.show()
```

- <u>Explication du code :</u>

Ce bout de code prend toutes valeurs de la colonne “stars” et les a arrondis à des entiers pour plus de clarté. Ensuite, à l’aide de la librairie matplotlib. On crée une grille de sous-graphiques de dimensions 2x3. Dans notre cas, on en a créé 5 barplots en utilisant la librairie seaborn de Python entre les notes du produit (“stars”) et les différentes variables (“Price”, “Reviews”, “isBestSeller” , “BoughtInLastMonth”, “listPrice”).



- Interprétation des résultats :

Tout d'abord, le premier graphique représente la relation qu'il existe entre le nombre d'étoiles d'un produit et son prix. On observe une relation négative: plus un produit est cher, moins il a d'étoiles. Cela peut s'expliquer par le fait que les consommateurs ont des attentes de plus en plus élevées à mesure qu'il débourse une somme importante pour l'acquisition d'un produit. Par ailleurs, le consommateur est souvent tenté de comparer le bien acheté avec un substitut moins coûteux, ce qui l'amène à évaluer son rapport qualité-prix à la baisse. Parallèlement, un produit payé moins cher nous donne l'impression d'avoir fait une "bonne affaire" et nous conduit à être moins exigeant quant à sa qualité et ainsi à lui attribuer une meilleure note.

Le second graphique, lui, évoque le lien entre le nombre d'avis d'un produit et son nombre d'étoiles. Et, on constate que plus un produit possède d'avis recensés, plus il a d'étoiles. En effet, plus un produit nous plaît, plus on est amené à lui laisser un avis. La satisfaction du client est telle qu'il prend le temps de laisser un avis pour convaincre les autres de se le procurer. Toutefois, nous nous attendions à ce que ce phénomène se produise dans le sens inverse: plus un produit déplaît, plus il recense d'avis dans la mesure où le mécontentement des consommateurs les pousse à laisser un avis négatif de sorte à désinciter les autres de se le procurer, mais nous n'observons pas cela.

Ensuite, le troisième graphique relate l'effet d'une promotion sur le nombre d'étoiles accordées à un produit. De prime abord, on remarque que la distribution est relativement homogène, aucune tendance ne se dégage d'emblée. Le fait qu'un produit bénéficie d'un rabais ne semble pas affecter le nombre d'étoiles qu'il détient. Néanmoins, on observe que les produits notés une étoile sont ceux qui bénéficient de la promotion la moins importante.

Le quatrième graphique révèle une tendance assez flagrante: plus un bien est un best seller, plus il possède d'étoiles. Cela semble logique puisque si un produit est considéré comme best seller, cela sous-entend qu'il est fortement apprécié des consommateurs. Réciproquement, un produit peut être considéré comme best seller dans la mesure qu'il possède un nombre d'étoiles élevé.

Pour finir, le dernier graphique représente le lien entre la nouveauté d'un produit et la note qu'il lui est attribuée. On constate que plus un produit a été acheté récemment, plus il a un nombre élevé d'étoiles. Cela peut provenir du fait que les individus apprécient la nouveauté et le changement à tel point qu'ils attribuent une meilleure note aux produits achetés récemment.

IV - DÉMARCHE DU PROGRAMME:

Avant d'entamer la résolution des questions, Il était nécessaire d'importer les bases de données en utilisant la fonction "read.csv" de la célèbre bibliothèque Pandas. On a importé les deux bases de données Amazon_categories et Amazon_products de la manière suivante :

```
A_categories_Base = pd.read_csv("../Data/amazon_categories.csv")
A_products_Base = pd.read_csv( "../Data/amazon_products.csv")
```

Par la suite, nous avons effectué une copie intégrale des deux bases de données initiale en les nommant A_categories et A_products. Les bases de données importées sont stockées dans le module appelé “data_load.py”.

Dans un deuxieme script “data_clean.py” nous avons effectué un nettoyage de la base de données en créant une fonction “nettoyage()” qui prend en entrée un dataframe et lui applique quelques modifications de la base de données d’entrée. A l’intérieur de cette fonction, on remplace dans un premier temps toutes les valeurs manquantes par le mot “missing” ainsi qu’on a renommé la colonne “id” par “category_id” nous faciliter la fusion des deux dataframes par la suite. De plus, nous avons essayé d’appliquer la fonction lower() pour toutes les observations de “category_name” et “title” pour éliminer les problèmes de cassation (Miniscules/Majuscules). Enfin nous avons effacé toutes les lignes dupliquées pour simplifier au max la base de données et de ne pas avoir de confusion par la suite.
Donc le script data_clean.py contient la fonction nettoyage() qui applique les instructions qui se trouvent à l’intérieur de cette fonction sur les deux dataframe A_categories et A_products avec une fusion des deux dataframes obtenus suite à l’application de la fonction de nettoyage()

Afin de répondre aux questions proposées, on a essayé de diviser les questions en 3 scripts différents :

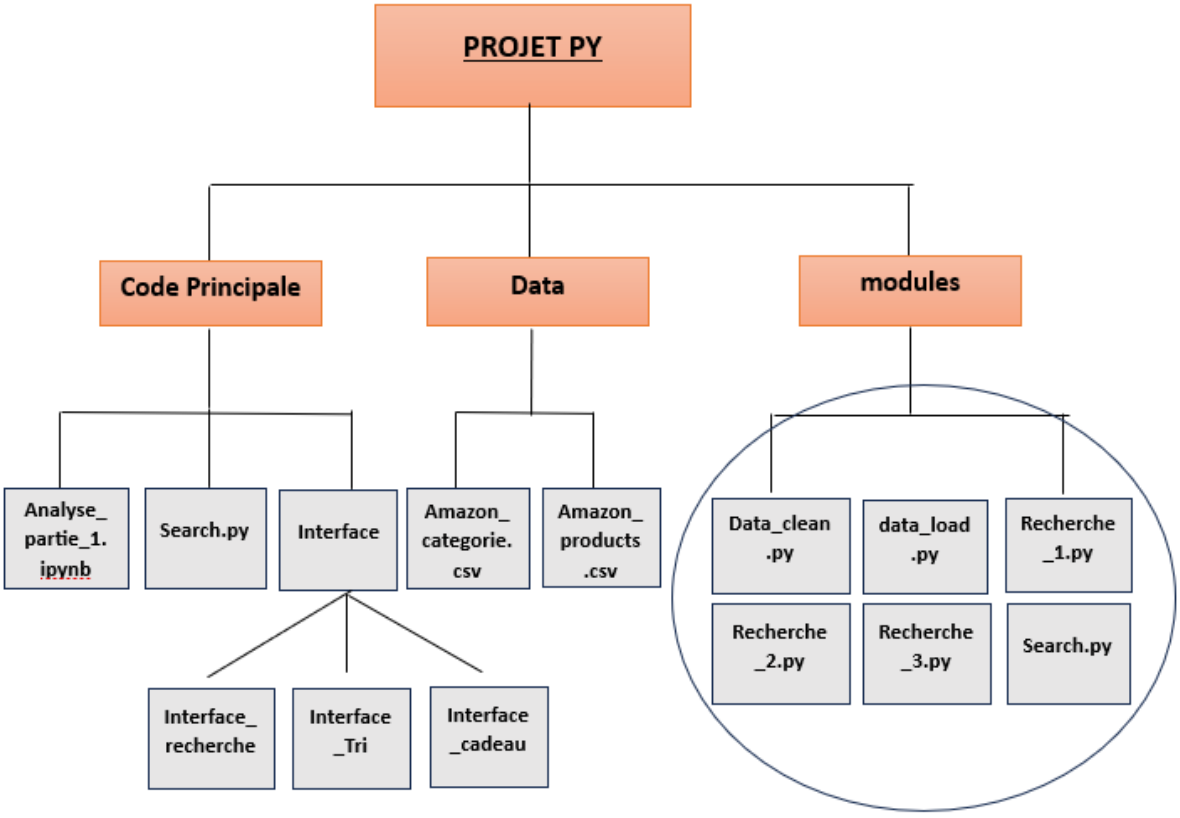
Dans Le premier script “recherche_1.py”, nous avons cherché à offrir à l'utilisateur la possibilité de saisir un nom de produit et une catégorie, obtenant ainsi les produits correspondants, triés selon son choix par notes, nombre d'avis décroissants ou par prix et en afficher que les 20 premiers. Pour ce faire, dans un premier temps, On a importé le module data_clean sous la forme “import data_clean as dc” pour qu’on puisse récupérer nos deux dataframe après avoir appliqué la fonction nettoyage du module “data_clean” ainsi que leur fusion avec l’instruction “fusion= dc.fusion”. A l’intérieur du module recherche_1, on a créé une fonction “recherche_produit” qui prend en entrée un dataframe et qui permet de retourner des produits triés par le nom de catégorie ,le nom du produit et le critère choisi entre notes, avis ou prix, dépendant du choix du consommateur. A l’intérieur, de la fonction, Nous avons demandé à l’utilisateur de faire ses choix par input. Le output de cette fonction est les 20 premiers produits triés selon ce que l’utilisateur aura choisi entre notes (ordre croissant, avis (ordre décroissant) et prix (ordre croissant)

Ensuite, nous avons cherché à réaliser un programme permettant de retourner les produits correspondant à une note minimale à un nombre d'unités vendues ainsi qu’à une catégorie le tout saisi par l’utilisateur et trié par prix ou par note selon son choix.

Pour ce faire, on a créé un nouveau script recherche_2 dans lequel on a déclaré une fonction qui répond bien à la question précédente , la fonction Tri(). Cette fonction prend un dataframe en paramètre d’entrée et retourne à la fin le dataframe triée en fonction du prix ou des notes de produits qui pourraient intéresser l’utilisateur en accord avec la catégorie, la note minimale et le nombre minimal d’unités vendues qu’il a entrées.
Nous avons, dans un premier temps, demandé à l’utilisateur de faire ses choix par input. On filtre ensuite le data Frame pour ne garder que les produits respectant ses conditions.
Ensuite, c’est l'utilisateur qui choisit de trier par prix montriez la valeur du data Frame par prix croissants et si il choisit de triés par note la base de données sera triée par avis décroissant (généralement, l’utilisateur préfère un produit dont le prix est plus faible ou un produit dans la note et la plus élevée)

V - SCHEMA ET STRUCTURE DU PROGRAMME:

i. Schéma de la structure du Projet



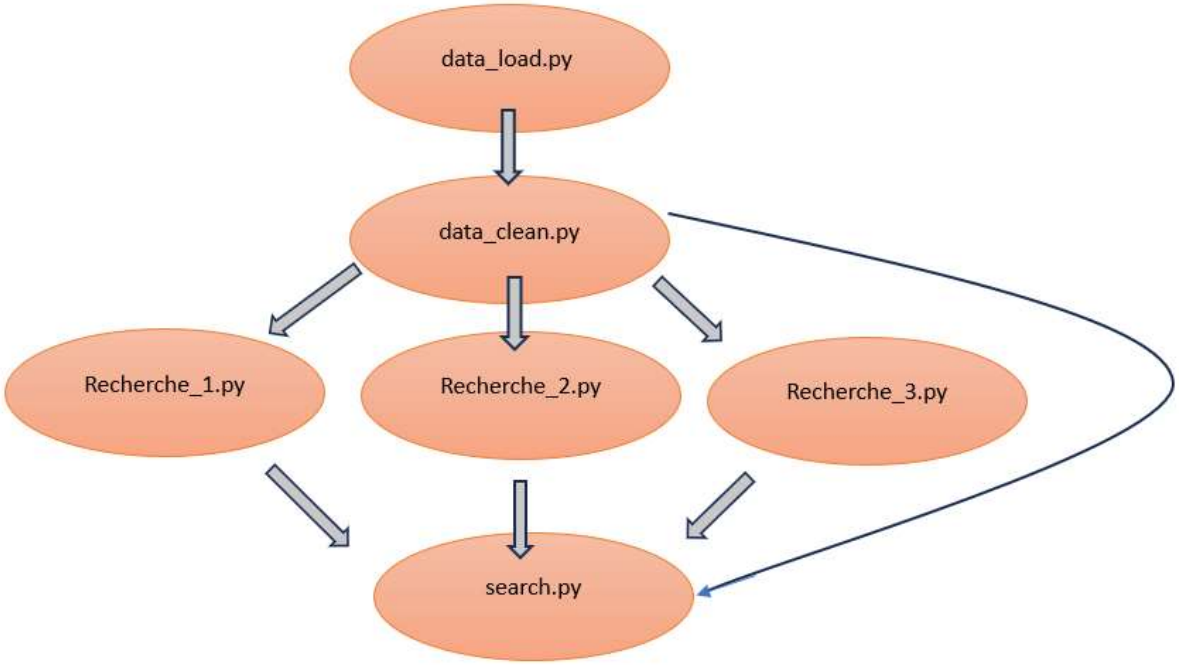
Pour assurer une meilleure lisibilité et efficacité dans l'exécution du code principal, il est essentiel d'organiser de manière ordonnée notre répertoire. Notre point de départ est le dossier "projet py", divisé en trois parties principales : "Code_Principal", "modules" et "Data".

A l'intérieur du Code Principale, il y a 3 fichiers : "analyse_partie1.ipynb" où on a travaillé sur la partie 1 de l'analyse descriptives des bases de données et visualisation, le fichier "search.py" qui représente le programme principal regroupant toutes la partie 2, et enfin le fichier "interface" qui contient 3 scripts des 3 différentes fonctions appliquées dans le cadre de la Partie 3 (création de l'interface graphique).

Le fichier Data contient nos deux bases de données de type csv : Amazon_catégorie.csv et Amazon_products.csv, à importer plus tard

Enfin, le fichier modules où on a mis tous les modules créés tout au long de la construction de notre programme.

ii. Interaction des scripts



Les scripts dans ces dossiers interagissent de manière coordonnée, chacun contribuant à une fonction spécifique du système. Cette structuration simplifiée des dossiers vise à rendre le code plus clair, facilitant ainsi la compréhension du programme.

- Le script "data_load.py" se charge d'importer les bases de données.
- Dans le script "data_clean.py", la fonction "nettoyage()" prend en charge le nettoyage de la base de données (gérer les valeurs manquantes, renommer des variables, gérer les doublons..). A l'intérieur du même module on effectue également la fusion. Pour ce faire, le module "data_load.py" est importé.

Les scripts “recherche_1.py”, “recherche_2.py” et “recherche_3.py” exigent l'appel préalable du module “data_clean.py” afin d’en extraire la fusion. Chacun de ces scripts contient une fonction dédiée à l’exécution de la demande en partie 2. Par exemple, “recherche_1.py” qui comprend la fonction “recherche_produit()” répond à la question : "L'utilisateur saisit un nom de produit et une catégorie ; le programme retourne les produits triés par notes/nombre d'avis décroissants ou par prix, selon la préférence de l'utilisateur. Si les résultats sont nombreux, seuls les 20 premiers sont affichés." Enfin, le script `search` agit comme un chef d'orchestre en appelant les trois modules nécessaires pour son exécution.

VI - CONSTRUCTION DU PROGRAMME:

Avant de répondre aux questions, Il est nécessaire d’importer les bases de données en utilisant la fonction "read.csv" de la célèbre bibliothèque Pandas. Nous avons importé les deux bases de données Amazon_categories et Amazon_products de la manière suivante :

A_categories_Base = pd.read_csv("../Data/amazon_categories.csv")

A_products_Base = pd.read_csv("../Data/amazon_products.csv")

Ensuite, nous avons effectué une copie intégrale des deux bases de données en les nommant A_categories et A_products. Les bases de données importées sont stockées dans le module appelé “data_load.py”.

Dans le second script “data_clean.py” nous avons effectué un nettoyage de la base de données en créant une fonction “nettoyage()” qui prend en entrée un dataframe et lui applique quelques. A l’intérieur de cette fonction, on remplace dans un premier temps toutes les valeurs manquantes par le mot “missing” ainsi qu’on a renommé la colonne “id” par “category_id” pour que ça nous facilite la fusion des deux dataframes après. De plus, on a essayé d’appliquer la fonction lower() pour toutes les observations de “category_name” et “title” pour éliminer les problèmes de cassation (Miniscules/Majuscules). et enfin on a effacé toutes les lignes dupliquées pour simplifier au max la base de données et de ne pas avoir des observations qui nous serveront à rien.

Donc le script data_clean.py contient la fonction nettoyage() qui applique les instructions qui se trouvent à l’intérieur de cette fonction sur les deux dataframe A_categories et A_products avec une fusion des deux dataframes obtenus suite à l’application de la fonction de nettoyage()

Afin de répondre aux problèmes proposées, nous avons essayé de diviser les questions en 3 scripts différents:

Dans Le premier script “recherche_1.py”, nous avons cherché à offrir à l'utilisateur la possibilité de saisir un nom de produit et une catégorie, obtenant ainsi les produits correspondants, triés selon son choix par notes, nombre d'avis décroissants ou par prix et en afficher que les 20 premiers. Pour ce faire, dans un premier temps, On a importé le module data_clean sous la forme “import data_clean as dc” pour qu’on puisse récupérer nos deux dataframe après avoir appliqué la fonction nettoyage du module “data_clean” ainsi que leur fusion avec l’instruction “fusion= dc.fusion”. A l’intérieur du module recherche_1, on a créé une fonction “recherche_produit” qui prend en entrée un dataframe

et qui permet de retourner des produits triés par le nom de catégorie ,le nom du produit et le critère choisi entre notes, avis ou prix, dépendant du choix du consommateur. A l'intérieur, de la fonction, Nous avons demandé à l'utilisateur de faire ses choix par input. Le output de cette fonction est les 20 premiers produits triés selon ce que l'utilisateur aura choisi entre notes (ordre croissant, avis (ordre décroissant) et prix (ordre croissant)

```
import pandas as pd
import data_clean as dc

fusion= dc.fusion

def recherche_produit(df1): #prend en défaut la table fusion

    nom_produit_0=input("saisir le nom du produit :").lower()
    nom_categorie_0=input('saisir le nom d une catégorie :').lower()
    choix_0 = input("Choisissez notes, avis ou prix : ").lower()


    df1 = df1.loc[df1["category_name"] == nom_categorie_0]
        #filtrer le dataframe df1 en fonction de la colonne "nom_categorie" choisie par l'utilisateur
    df1 = df1[df1['title'].str.contains(nom_produit_0, case=False)]
        #conserver les lignes de la colonne 'title' contenant que le nom du produit saisie par l'utilisateur(nom_produit)
        #en fonction du critère de tri choisi par l'utilisateur, on trie en fonction de la colonne associée
    if choix_0=='notes':
       df1 = df1.sort_values(by="stars",ascending=(True))
    elif choix_0=='avis':
       df1  = df1.sort_values(by="reviews",ascending=(False))
    elif choix_0 =='prix':
        df1 = df1.sort_values(by="price",ascending=(True))

    else:
        print("Le choix saisi est invalide, les produits ne sont pas triés. Voici les " + str(nom_produit_0) +
              " appartenant à la catégorie " + str(nom_categorie_0))
        #si l'utilisateur ne rentre pas de critère de tri ou rentre une mauvaise syntaxe
        #alors un message sera affiché et retournera les libellés des produits triés seulement
        #par le nom de produit et le nom de catégorie
    df1=df1 ['title'].reset_index(drop=True)

    return df1.head(20)
```

Ensuite, nous avons cherché à réaliser un programme permettant de retourner les produits correspondant à une note minimale à un nombre d'unités vendues ainsi qu'à une catégorie le tout saisi par l'utilisateur et trié par prix ou par note selon son choix.

Pour ce faire, on a créé un nouveau script recherche_2 dans lequel on a déclaré une fonction qui répond bien à la question précedente , la fonction Tri(). Cette fonction prend un dataframe en paramètre d'entrée et retourne à la fin le dataframe triée en fonction du prix ou des notes de produits qui pourraient intéresser l'utilisateur en accord avec la catégorie, la note minimale et le nombre minimal d'unités vendues qu'il a entrées.
Nous avons, dans un premier temps, demandé à l'utilisateur de faire ses choix par input. On filtre ensuite le data Frame pour ne garder que les produits respectant ses conditions.

Ensuite, c'est l'utilisateur qui choisit de trier par prix montriez la valeur du data Frame par prix croissants et si il choisit de triés par note la base de données sera triée par avis décroissant (généralement, l'utilisateur préfère un produit dont le prix est plus faible ou un produit dans la note et la plus élevée)

Voici le programme réalisé :

```
def Tri(df):

    Tri_par = input("voulez vous trier par prix ou par note (choisissez 'prix' ou 'note') :").lower()
    categorie=input("Quelle catégorie voulez-vous ?").lower() #on demande au Cr de choisir une catégor

    note_min=int(input("Choisissez une note minimale entre 1 et 5 : ")) #On demande au Cr quelle note

    nb_unite_min=int(input("Quel doit être le nombre minimal d'unités vendues ? ")) #Le Cr choisit le

    df = df.loc[(df["category_name"] == categorie) & (df["stars"]>=note_min) &
                (df["boughtinlastmonth"]>=nb_unite_min)]  #on garde qu'une partie de la DataFrame init

    if Tri_par=='prix': #si le Cr a choisi de trier par prix le dataFrame contenant les produits qui p
        df = df.sort_values(by="price",ascending=(True)) #on trie les valeurs du dataFrame par prix cr

    elif(Tri_par=='note'): #si le Cr a choisi de trier par note le dataFrame contenant les produits qu
        df = df.sort_values(by="stars",ascending=(False)) #on trie les valeurs du dataFrame par avis d

    return df.head(20)

print(Tri(fusion))
```

Dans un 3ème script "recherche_3" on a cherché à répondre à la dernière question.
Pour celle-ci, nous avions cherché à réaliser un code permettant de recommander un cadeau à un utilisateur en créant une fonction cadeau() qui prend un dataframe (dans notre cas la table fusion) et retourne un autre data frame AA où y a tous les produits répondant à aux choix de l'utilisateur.
Pour ce faire, il devra pouvoir spécifier ses préférences concernant une ou plusieurs catégories, son budget, ainsi que le type de produits qu'il souhaite offrir.
le tout en favorisant, dans un premier temps, les produits, les mieux notés dans un second temps, ce ayant le plus d'avis, puis dans un dernier temps, les produits best-sellers (nous avons choisi c'est ordre, car selon nous il est préférable, soit tout d'abord bien noté ensuite qu'il est beaucoup d'avis pour que cette note ne soit pas biaisé, et enfin qu'il soit un best-seller étant plus). Pour ce faire, on a importé le module data_clean sous la forme "import data_clean as dc" pour qu'on puisse récupérer nos deux dataframe après avoir appliqué la fonction nettoyage du module "data_clean" ainsi que leur fusion avec l'instruction "fusion= dc.fusion"

Voici le programme que nous avons réalisé :

```
def cadeau(df):
    result = {'asin': [],
        'title': [],
        'imgUrl': [],
        'productURL': [],
        'stars': [],
        'reviews': [],
        'price': [],
        'listPrice': [],
        'category_id': [],
        'isBestSeller': [],
        'boughtInLastMonth': [],
        'category_name': []}
    result = pd.DataFrame(result) #on crée un dataFrame vide

#on met choisit le type de chaque variable
    result['asin'] = result['asin'].astype(str)
    result['title'] = result['title'].astype(str)
    result['imgUrl'] = result['imgUrl'].astype(str)
    result['productURL'] = result['productURL'].astype(str)
    result['stars'] = result['stars'].astype(float)
    result['reviews'] = result['reviews'].astype(int)
    result['price'] = result['price'].astype(float)
    result['listPrice'] = result['listPrice'].astype(float)
    result['category_id'] = result['category_id'].astype(int)
    result['isBestSeller'] = result['isBestSeller'].astype(bool)
    result['boughtInLastMonth'] = result['boughtInLastMonth'].astype(int)
    result['category_name'] = result['category_name'].astype(str)

while True: #boucle qui continue tant qu'on l'arrête pas avec break
    categorie = input("Entrez une catégorie (ou 'non' pour terminer) : ").lower() #on ajoute
    if categorie == 'non':#ajouter len()
        break  #si le consommateur ne veut pas entrer de catégorie on arrête la boucle
    CAT.append(categorie) #on ajoute les catégories que le Cr veut dans la liste vide qu'on
    if len(CAT) != 0: #s'il y'a des catégories dans la liste (donc si le Cr a fait un choix)
        for i in CAT: #pour chaque élément de la liste donc pour chaque catégorie
            df = df.loc[df["category_name"] == i] # à chaque fois on garde que les produits
            result = pd.concat([result, df]) #on ajoute dans les result tous les produits re
    else:
        result=df

budget=float(input("quel est votre budget ?"))#ajouter la possibilité de ne pas en mettre

A=result.loc[result["price"]<= budget] #Parmi les produits qui respectent les catégories cho
choix_T = input("voulez vous préciser le type du produit( oui / non) : ").lower()

if choix_T == 'oui':
    Type=input("choisissez un type : ").lower()
    d1 = A[A['title'].str.contains(Type, case=False)] #on garde que les produits dont le tit
    d1 = d1.sort_values(['stars','reviews','isBestSeller'], ascending=[False,False,False])
     #on trie tous les produits dont le titre contient le type choisi par le Cr, d'abord par

    d2 = A[~A['title'].str.contains(Type)] #on récupère tous les produits dont le titre ne c
    d2 = d2. sort_values(['stars','reviews','isBestSeller'], ascending=[False,False,False])
     #on trie tous les produits dont le titre ne contient pas le type choisi par le Cr, d'ab

    AA = pd.concat([d1, d2]) #on rassemble les deux dataframe en mettant  au dessus celui co

else :
    AA=A.sort_values(['stars','reviews','isBestSeller'], ascending=[False,False,False]) #si

return AA.head(20)
```

VII - 4 - LA DÉMARCHE DE CONSTRUCTION DE L'INTERFACE (AVEC DES CAPTURES D'ÉCRAN DU RENDU VISUEL)

Pour créer une interface graphique on a importé deux bibliothèques supplémentaires : Tkinter (qui est la bibliothèque d'interface graphique) et webbrowser (pour ouvrir des liens dans le navigateur par défaut).

i. Interface graphique et liens cliquables avec la fonction recherche_produit :

Dans cette fonction, l'utilisateur saisit un nom de produit et une catégorie sur l'interface graphique et le programme doit lui retourner les produits par notes/ nombre d'avis décroissants ou par prix, selon ce qu'il aura choisi avec des liens le dirigeant soit vers l'image ou soit sur le site Amazon.

- **Création de l'interface graphique grâce à l'importation de Tkinter :**

Tout d'abord nous avons construit un programme permettant l'affichage d'une interface graphique pour que l'utilisateur puisse saisir directement ses critères.
Toutes les fonctions suivantes sont possibles grâce à l'utilisation de la libraire Tkinter.

Pour cela, nous avons repris le code ***recherche_produit*** de la partie deux puis nous avons créé une nouvelle fenêtre permettant la création de l'interface.
L'objectif de ce code est que l'utilisateur puisse saisir trois critères : un produit, une catégorie et un critère de tri. Pour permettre cela, nous avons créé trois labels respectifs à chacun, ainsi que trois zones de saisie grâce au widget **entry()**.

Nous avons créé un bouton "rechercher" avec le widget **button()** rattaché à notre fonction *recherche_produit* permettant la recherche des résultats en fonction de la saisie de l'utilisateur (nous avons rajouté un bout de code dans *recherche_produit* expliqué plus bas).

Nous avons aussi créé une zone de texte avec la widget **text()** pour pouvoir afficher les résultats de notre recherche.

L'affichage de chaque fonction (bouton, zone de saisie (entrée), zone de texte) se fait grâce à la méthode **pack()**.

Pour une meilleure lisibilité et un peu d'amusement, nous avons ajouté quelques options graphiques tels que : la taille de la fenêtre, la couleur du texte, la couleur du fond etc….

L'affichage de la fenêtre est possible grâce à la méthode **mainloop()**

Voici le code sur la partie interface graphique :

```
# Création la fenêtre principale
fenetre = tk.Tk() #création de la fenetre principale
fenetre.title("Choisir un produit, une catégorie et un critère") #on donne un titre à cette fenetre
fenetre.geometry('900x1200') #on choisit la dimension

Titre_label = tk.Label(fenetre, text="TRIER LES PRODUIT !", font=("Palatino", 20, "bold"), fg='MediumPurple1')  #on crée un

label_produit = tk.Label(fenetre, text='Saisir un produit',font=("Palatino", 20, "bold"), fg='MediumPurple1') #on crée un la
label_produit.pack()  #on ajoute le label dans la fenetre principale
choix_produit = tk.Entry(fenetre)  #l'utilisateur doit entrer son choix de produit ici
choix_produit.pack()

label_categorie = tk.Label(fenetre, text='Saisir une catégorie', font=("Palatino", 20, "bold"), fg='MediumPurple1')
label_categorie.pack()
choix_categorie = tk.Entry(fenetre)
choix_categorie.pack()

label_critere = tk.Label(fenetre, text='Saisir notes, avis ou prix :', font=("Palatino", 20, "bold"), fg='MediumPurple1')
label_critere.pack()
choix_critere = tk.Entry(fenetre)
choix_critere.pack()

#création d'un bouton de recherche
bouton_rechercher = tk.Button(fenetre, text="Rechercher",font=("Palatino", 20, "bold"), fg='MediumPurple1', command=recherch
bouton_rechercher.pack()

# Création d'une zone de texte pour afficher les résultats
zone_texte = tk.Text(fenetre, height=150, width=140) #création de la zone de texte qui contient les résultats
zone_texte.pack()

fenetre.mainloop()
```

Résultat de l'affichage de l'interface :

Pour permettre l'affichage dans la zone de texte au moment d'enclencher le bouton recherche, nous avons ajouté dans la fonction *recherche_produit* la méthode **insert()**. Cette méthode permet

d'insérer les résultats de notre recherche dans la zone de texte, en chaine de caractère et sans les index pour une meilleure lisibilité.

Au moment d'effectuer une seconde recherche, nous nous sommes rendu compte que la recherche précédente restait affichée. Nous avons donc rajouté une méthode **delete()** prenant en paramètre deux positions (la première ligne et la dernière ligne) permettant l'effacement du contenu précédent, de la première ligne à la dernière ligne.

```
position1= 1.0 #on définit la position 1 dans la zone de texte (c'est le début)
position2= tk.END #on définit la position 2 dans la zone de texte (c'est la fin)
zone_texte.delete(position1, position2) #on supprime le contenu actuel de la zone de texte
```

- **Liens cliquables :**

Nous avons commencé par récupérer les valeurs saisies par l'utilisateur grâce à la fonction **get()**.

```
nom_produit = choix_produit.get() #la j
nom_categorie = choix_categorie.get()
choix = choix_critere.get()
```

Puis nous avons créé une boucle à la fin de notre fonction recherche_produit pour parcourir les lignes du DataFrame (la boucle itère à travers les lignes du DataFrame).
On souhaite récupérer les liens, les images et les titres des produits.
On affiche d'abord le titre de chaque produit dans la zone de texte puis on veut afficher les liens cliquables des produits respectifs dans cette même zone.

Pour chaque produit, le code crée deux labels (l'un pour l'URL du produit et l'autre pour l'URL de l'image). Puis, ces labels sont configurés pour être cliquables. Et, lorsqu'ils sont cliqués par l'utilisateur, la fonction lien_cliquable» est appelée pour afficher le lien dans un navigateur (fonction expliquée plus bas).

Nous avons créé un premier label qui contient le « lien1 » (c'est-à-dire l'URL du produit) en couleur bleu avec un curseur de type main. Puis, nous avons créé un deuxième label qui contient le « lien2 » (c'est-à-dire l'URL image) également en couleur bleu avec un curseur de type main.

On utilise la méthode "bind" pour que la fonction « lien_cliquable» soit appelée à chaque fois qu'un lien est cliqué. Ainsi, on lie un événement à une fonction (ici : un clic gauche avec la fonction « lien_cliquable»). On utilise pour cela la fonction lambda qui prend deux arguments : « e » qui est l'événement (ici le clic gauche) et « l » (c'est-à-dire les liens). La fonction lambda appelle la fonction «lien_cliquable » (avec « l ») lorsque l'événement est déclenché (ici le clic gauche).
Après l'ajout des informations de chaque produit dans la zone de texte on insère un saut de ligne.

Ainsi, pour chaque produit, le code crée deux labels (un pour l'URL produit et l'autre pour l'URL image) pour afficher les liens dans la zone de texte, qui sont configurés pour être cliquables.

Pour pouvoir ouvrir le lien dans un navigateur, les labels doivent être liés à la fonction “lien_cliquable” utilisable grâce au module ‘webbrowser’. Nous avons donc créé une fonction lien_cliquable qui permet l’ouverture du lien sur le navigateur.

```
def lien_cliquable(l):
    webbrowser.open_new(l)
```

Voici le code dans son entièreté :

```
import os
import pandas as pd
import webbrowser
import tkinter as tk

# avoir le répertoire de travail
os.getcwd()

# Importation des données
A_categories = pd.read_csv("../../Data/amazon_categories.csv", sep=",")
A_products = pd.read_csv("../../Data/amazon_products.csv")

A_categories=A_categories.rename(columns = {'id': 'category_id'})
fusion = pd.merge(A_products,A_categories, how="outer", on=["category_id"])

def recherche_produit():
    nom_produit = choix_produit.get() #la fonction get permet de récupérer la valeur
    nom_categorie = choix_categorie.get()
    choix = choix_critere.get()

    # Filtrer le dataframe en fonction des valeurs saisies
    df1 = fusion.loc[fusion["category_name"] == nom_categorie]
    df1 = df1[df1['title'].str.contains(nom_produit, case=False)]

    # En fonction du critère de tri choisi par l'utilisateur, trier le dataframe
    if choix == 'notes':
        df1 = df1.sort_values(by="stars", ascending=True)
    elif choix == 'avis':
        df1 = df1.sort_values(by="reviews", ascending=False)
    elif choix == 'prix':
        df1 = df1.sort_values(by="price", ascending=True)
    else:
        print("Le choix saisi est invalide. Les produits ne sont pas triés.")

    position1= 1.0 #on définit la position 1 dans la zone de texte (c'est le début)
    position2= tk.END #on définit la position 2 dans la zone de texte (c'est la fin)
    zone_texte.delete(position1, position2) #on supprime le contenu actuel de la zone

    for index, row in df1.iterrows():  #on parcours les lignes du DataFrame
#df.iterrows() retourne l'index de la ligne et toutes les données de la ligne sous for
        lien1 = row['productURL'] #on garde pour chaque ligne que l'element "productUR
        lien2 = row['imgUrl'] #on garde pour chaque ligne que l'élement "imgUrl" pour
        titre = row['title'] #on récupère le nom de chaque produit
        zone_texte.insert(position2, row['title']) #on affiche le nom de chaque produi

        lien_label1 = tk.Label(zone_texte, text=lien1, fg="blue", cursor="hand1") #on
        lien_label1.bind("<Button-1>", lambda e, l=lien1: lien_cliquable(l)) #On lie l
        #La fonction lambda appelle la fonction "lien_ouvert" lorsque l'événement (cli
        zone_texte.window_create(position2, window=lien_label1) #on ajoute le label1 c
        #à la position 2 (c'est-à-dire à la fin du texte actuel) #on affiche donc des

        lien_label2 = tk.Label(zone_texte, text=lien2, fg="blue", cursor="hand1")  #on
        lien_label2.bind("<Button-1>", lambda e, l=lien2: lien_cliquable(l)) #On lie l
        #La fonction lambda appelle la fonction "lien_ouvert" lorsque l'événement (cli
        zone_texte.window_create(position2, window=lien_label2) #on affiche donc des l

        zone_texte.insert(tk.END, "\n") #à chaque fois qu'on insère les informations p
```

```
def lien_cliquable(l):
    webbrowser.open_new(l) #on ouvre chaque lien dans le navigateur (lorsqu'un clique gauche a lieu)

# Création la fenêtre principale
fenetre = tk.Tk() #création de la fenetre principale
fenetre.title("Choisir un produit, une catégorie et un critère") #on donne un titre à cette fenetre
fenetre.geometry('980x1200') #on choisit la dimension

Titre_label = tk.Label(fenetre, text="TRIER LES PRODUIT !", font=("Palatino", 20, "bold"), fg='MediumPurple1')  #on crée un la

label_produit = tk.Label(fenetre, text='Saisir un produit',font=("Palatino", 20, "bold"), fg='MediumPurple1') #on crée un labe
label_produit.pack()  #on ajoute le label dans la fenetre principale
choix_produit = tk.Entry(fenetre)  #l'utilisateur doit entrer son choix de produit ici
choix_produit.pack()

label_categorie = tk.Label(fenetre, text='Saisir une catégorie', font=("Palatino", 20, "bold"), fg='MediumPurple1')
label_categorie.pack()
choix_categorie = tk.Entry(fenetre)
choix_categorie.pack()

label_critere = tk.Label(fenetre, text='Saisir notes, avis ou prix :', font=("Palatino", 20, "bold"), fg='MediumPurple1')
label_critere.pack()
choix_critere = tk.Entry(fenetre)
choix_critere.pack()

#création d'un bouton de recherche
bouton_rechercher = tk.Button(fenetre, text="Rechercher",font=("Palatino", 20, "bold"), fg='MediumPurple1', command=recherche_
bouton_rechercher.pack()

# Création d'une zone de texte pour afficher les résultats
zone_texte = tk.Text(fenetre, height=150, width=140) #création de la zone de texte qui contient les résultats
zone_texte.pack()

fenetre.mainloop()
```

Résultat de l'interface après saisie de l'utilisateur : par exemple, si l'utilisateur saisie *«Battery»* dans la catégorie *« Laptop Accessories »* et souhaite un *tri par notes*, alors il obtiendra les résultats suivants :

Le premier lien de chaque produit indique la page Amazon du produit, l'utilisateur peut le commander directement. Et le deuxième lien indique son image.

ii. Interface graphique et liens cliquables avec la fonction tri :

Dans cette question, l'utilisateur devait entrer une catégorie, une note minimale et un nombre minimal d'unités vendues et le programme devait retourner les produits correspondants triés par prix ou par note. Premièrement, nous avons essayé de construire un programme qui permettait de construire une interface graphique où l'utilisateur pouvait entrer ses critères et obtenir en retour les produits qui pourraient l'intéresser (dans un premier temps sans incorporer les liens vers les pages des produits sur Amazon ou les images).

Premièrement, nous avons créé notre fenêtre principale (« fenêtre »). Puis, nous avons créé des Labels pour permettre à l'utilisateur d'entrer ses souhaits dans l'interface graphique. Le bouton « Trier » lance ensuite la fonction Tri de la partie 2 (que nous avons modifié) pour avoir les résultats. La zone de texte que nous avons créé contiendra les résultats de la fonction qui est lancée lorsque l'utilisateur appuie sur le bouton "Trier".

```
fenetre = tk.Tk () #création de la fenetre principale
fenetre.title("tri") #on donne un titre à cette fenetre
fenetre.geometry('900x1200') #on choisit la dimension

Titre_label = tk.Label(fenetre, text="Trouver le Cadeau Idéal !", font=("Palatino", 30, "bold"), fg='MediumPurple1') #on

Tri_par = tk.Label(fenetre, text="tri par prix ou note ?",font=("Palatino", 30, "bold"), fg='MediumPurple1') #on crée un
Tri_par.pack() #on ajoute le label dans la fenetre principale
choix_tri = tk.Entry(fenetre) #l'utilisateur doit entrer son choix de tri ici
choix_tri.pack()

categorie = tk.Label(fenetre, text="Saisir une catégorie ?",font=("Palatino", 30, "bold"), fg='MediumPurple1')
categorie.pack()
choix_cat = tk.Entry(fenetre)
choix_cat.pack()

note_min = tk.Label(fenetre, text="Saisir une note entre 1 à 5",font=("Palatino", 30, "bold"), fg='MediumPurple1')
note_min.pack()
choix_note = tk.Spinbox(fenetre, from_=1, to=5)
choix_note.pack()

nb_unite = tk.Label(fenetre, text="Saisir un nombre minimum d'unités vendues",font=("Palatino", 30, "bold"), fg='MediumPu
nb_unite.pack()
choix_nb = tk.Entry(fenetre)
choix_nb.pack()

Bouton_trier = tk.Button(fenetre, text="Trier", font=("Palatino", 30, "bold"), fg='MediumPurple1', command=Tri) #on crée
Bouton_trier.pack()

zone_texte = tk.Text(fenetre, width=150, height=40) #création de la zone de texte qui contient les résultats
zone_texte.pack()

fenetre.mainloop()
```

Dans la fonction Tri, nous voulions d'abord récupérer les critères que l'utilisateur a entré dans l'interface graphique. Cela est en effet possible grâce à la fonction « get( ) ».
La suite de la fonction se déroule comme avec des inputs habituels (c'est-à-dire comme à la partie 2). Ainsi, le résultat final qui doit être affiché (en fonction des choix du l'utilisateur) est contenu dans le DataFrame « df ».
Avant d'afficher les résultats dans l'interface graphique, nous avons défini deux positions : la « position1 » et la « position2 ». Ces positions sont notamment utilisées pour ajouter ou supprimer du texte dans la zone de texte. La méthode « delete » permet de supprimer le contenu actuel de la zone de texte (de la première ligne à la dernière) pour ainsi afficher de nouvelles données lorsque l'utilisateur souhaite changer ses critères et afficher de nouveaux produits.
La méthode « insert » permet d'insérer un texte dans la zone de texte à une position donnée. Ainsi, on affiche le DataFrame convertit en chaîne de caractère (et sans les indices) dans la zone de texte.

```python
def Tri(df=fusion):
    Tri_par = choix_tri.get()#la fonction get permet de récupérer la valeur entrée dans l'i
    categorie = choix_cat.get()
    note_min = int(choix_note.get())
    nb_unite_min = float(choix_nb.get())
    df = df.loc[(df["category_name"] == categorie) & (df["stars"]>=note_min) &
                (df["boughtInLastMonth"]>=nb_unite_min)]

    if Tri_par=='prix':
        df = df.sort_values(by="price",ascending=(True))

    elif(Tri_par=='note'):
        df = df.sort_values(by="stars",ascending=(False))

    position1 = 1.0 #on définit la position 1 dans la zone de texte (c'est le début)
    position2 = tk.END #on définit la position 2 dans la zone de texte (c'est la fin)
    zone_texte.delete(position1, position2) #on supprime le contenu actuel de la zone de te
```

Voici un aperçu du rendu visuel de l'interface :

Voici exemple de critères que l'utilisateur pourrait entrer :

Ce que l'utilisateur obtient à la fin :

Après avoir réussi à afficher les résultats dans l'interface, nous avons essayé de remplacer ces résultats par le nom de chaque produit suivi de son URL produit et de l'URL de l'image.
Nous n'avons pas modifié la partie sur la création de la fenêtre principale et des différents widgets de l'interface.

Cependant, la fonction "Tri" a dû être modifiée. Elle est appelée lorsque l'utilisateur appuie sur le bouton « trier ». Elle récupère les valeurs des widgets de l'utilisateur (notamment la catégorie, la note minimale, le nombre d'unités minimales, le type de tri). On filtre ensuite le DataFrame en fonction des critères de l'utilisateur et du type de tri choisi.
On veut afficher les résultats dans la zone de texte de l'interface graphique. On veut afficher les noms des produits, leurs liens vers le site d'Amazon, et leurs images.
Nous avons décidé de faire une boucle pour parcourir les lignes du DataFrame (la boucle itère à travers les lignes du DataFrame). On souhaite récupérer les liens, les images et les titres des produits. On affiche d'abord le titre de chaque produit dans la zone de texte. On veut maintenant afficher des liens cliquables pour chaque produit dans la zone de texte. Pour chaque produit, le code crée deux labels (l'un pour l'URL du produit et l'autre pour l'URL de l'image). Puis, ces labels sont configurés pour être cliquables. Et, lorsqu'ils sont cliqués par l'utilisateur, la fonction «lien_ouvert » est appelée pour afficher le lien dans un navigateur.

Nous avons créé un premier label qui contient le « lien1 » (c'est-à-dire l'URL du produit) en couleur bleu et avec un curseur de type main. Puis, nous avons créé un deuxième label qui contient le "lien 2" (c'est-à-dire l'URL image) en couleur bleu également et avec un curseur de type main. On utilise la méthode "bind" pour que la fonction « lien_ouvert » soit appelée à chaque fois qu'un lien est cliqué. Ainsi, on lie un événement à une fonction (ici : un clic gauche avec la fonction « lien_ouvert »). On utilise pour cela la fonction lambda qui prend deux arguments : « e » qui est l'événement (ici le clic gauche) et « l » (c'est-à-dire les liens). La fonction lambda appelle la fonction «lien_ouvert » (avec « l ») lorsque l'événement est déclenché (ici le clic gauche). La fonction « lien_ouvert » utilise le module 'webbrowser' pour ouvrir chaque lien dans un nouveau navigateur.

Après l'ajout des informations de chaque produit dans la zone de texte on insère un saut de ligne. Ainsi, pour chaque produit, le code crée deux labels (un pour l'URL produit et l'autre pour l'URL image) pour afficher les liens dans la zone de texte, qui sont configurés pour être cliquables. Pour ensuite ouvrir ce lien dans un navigateur, ces labels vont être liés à la fonction "lien_ouvert".

```
def Tri(df=fusion):
    Tri_par = choix_tri.get()#la fonction get permet de récupérer la valeur ent
    categorie = choix_cat.get()
    note_min = int(choix_note.get())
    nb_unite_min = float(choix_nb.get())
    df = df.loc[(df["category_name"] == categorie) & (df["stars"]>=note_min) &
                (df["boughtInLastMonth"]>=nb_unite_min)]

    if Tri_par=='prix':
        df = df.sort_values(by="price",ascending=(True))

    elif(Tri_par=='note'):
        df = df.sort_values(by="stars",ascending=(False))

    position1 = 1.0 #on définit la position 1 dans la zone de texte (c'est le c
    position2 = tk.END #on définit la position 2 dans la zone de texte (c'est l
    zone_texte.delete(position1, position2) #on supprime le contenu actuel de l

    for index, row in df.iterrows(): #on parcours les lignes du DataFrame
#df.iterrows() retourne l'index de la ligne et toutes les données de la ligne s
        lien1 = row['productURL'] #on récupère le lien de chaque produit
        lien2 = row['imgUrl'] #on récupère les liens des images de chaque produ
        titre = row['title'] #on récupère le nom de chaque produit
        zone_texte.insert(position2, titre) #on affiche le nom de chaque produi

        label1 = tk.Label(zone_texte, text=lien1, fg="blue", cursor="hand1") #c
        label1.bind("<Button-1>", lambda e, l=lien1: ouvrir_lien(l)) #On lie l'
        zone_texte.window_create(position2, window=label1) #on ajoute le label1
#on affiche donc des liens cliquables pour chaque produit

        label2 = tk.Label(zone_texte, text=lien2, fg="blue", cursor="hand1") #c
        label2.bind("<Button-1>", lambda e, l=lien2: ouvrir_lien(l)) #On lie l'
        zone_texte.window_create(position2, window=label2) #on affiche donc des

        zone_texte.insert(position2, "\n") #à chaque fois qu'on insère les infor

def ouvrir_lien(l):
    webbrowser.open_new(l) #on ouvre chaque lien dans le navigateur (lorsqu'un c
```

Voici un aperçu des résultats que l'utilisateur pourrait obtenir :

iii. Interface graphique et liens cliquables avec la fonction cadeau :

Pour la question 3, le programme doit aider l'utilisateur à choisir un cadeau. Ainsi, l'utilisateur saisit sa ou ses catégories préférée(s), son budget et le type de produit qu'il cherche, dans l'interface.

Dans la même logique que les deux questions précédentes, nous avons créé une fenêtre principale et plusieurs widgets dans notre interface graphique.
Nous avons créé des labels avec des zones de saisies pour que l'utilisateur puisse rentrer ses critères ainsi que deux boutons : bouton_cadeau qui appelle la fonction cadeau et bouton_ajout_cat qui appelle la fonction *ajouter_categorie*.

```
#création de l'interface
fenetre = tk.Tk()
fenetre.title("cadeau")
fenetre.geometry('900x1200')

fenetre.configure(bg='pink')

Titre_label = tk.Label(fenetre, text="Trouver le Cadeau Idéal !", font=("Palatino", 30, "bold"), fg='MediumPurple1')
Titre_label.pack()

#Création du label catégorie (correspond au 1er choix de catégorie)
cat_label = tk.Label(fenetre, text="Saisir une catégorie :", font=("Arial", 16, "bold"), fg='dark violet')  # Violet foncé
cat_label.pack()
choix_cat = tk.Entry(fenetre) #création d'une zone de saisie permettant la saisie de l'utilisateur
choix_cat.pack()
CAT.append(choix_cat) #cette nouvelle entrée est ajouté à la liste CAT

#Bouton qui permet l'ajout d'une ou plusieurs catégorie selon le choix de l'utilisateur
bouton_ajout_cat= tk.Button(fenetre, text="Ajouter une catégorie supplémentaire",  font=("Arial", 10, "bold"), fg='dark violet',
                            command=ajouter_categorie)
bouton_ajout_cat.pack()

#Création du label budget dans lequel l'utilisateur pourra saisir son budget
budget = tk.Label(fenetre, text="Saisir votre budget :", font=("Arial", 16, "bold"), fg='dark violet')
budget.pack()
choix_budget = tk.Entry(fenetre)
choix_budget.pack()

#Création du label type qui permet à l'utilisateur de préciser si il veut un type de produit ou non
Proposition_produit = tk.Label(fenetre, text="voulez-vous préciser le type de produit (oui/non) :",  font=("Arial", 16, "bold"), fg='dark violet')
Proposition_produit.pack()
choix_produit = tk.Entry(fenetre)
choix_produit.pack()
type_produit = tk.Label(fenetre, text="Si oui choisissez un type de produit :", font=("Arial", 16, "bold"), fg='dark violet')
type_produit.pack()
choix_Type = tk.Entry(fenetre)
choix_Type.pack()

#Création d'un bouton recherche qui appelle la fonction cadeau :
#lorsque l'utilisateur aura saisit ses critères, il appuiera sur ce bouton afin de lancer la recherche
Bouton_cadeau = tk.Button(fenetre, text="C'est parti !", font=("Arial", 18, "bold"), fg='dark violet', command=cadeau)
Bouton_cadeau.pack()

#Création d'une zone de texte qui affichera les résultats de la recherche (liée à la fonction cadeau)
zone_texte = tk.Text(fenetre, width=90, height=30)
zone_texte.pack()

fenetre.mainloop()

zone_texte.configure(font=("Times New Roman", 25, "italic")) #configuration d'une police de texte

fenetre.mainloop()
```

Voici un aperçu de l'affichage de l'interface graphique :

En effet, nous avons dû créer une nouvelle fonction *ajouter_categorie* qui est appelé lorsque l'utilisateur souhaite saisir une autre catégorie en cliquant sur le bouton *"Saisir une catégorie supplémentaire*". Au sein de cette fonction, nous avons créé un label et une zone de saisie qui sera ajoutée à chaque fois que l'utilisateur voudra ajouter une catégorie supplémentaire.

```
def ajouter_categorie(): #fonction appelée lorsque l'utilisateur clique sur le bouton "Saisir une catégorie suppléme
    #on crée un nouveau label et une nouvelle zone de saisie pour que l'utilisateur puisse ajouter une catégorie
    label = tk.Label(fenetre, text="Saisir une nouvelle catégorie :", font=("Arial", 16, "bold"), fg='dark violet')
    label.pack()
    entree = tk.Entry(fenetre)
    entree.pack()
    CAT.append(entree) #cette nouvelle entrée est ajouté à la liste CAT
```

Cependant, nous avons dû modifier notre fonction cadeau initiale car la boucle "while True" n'était pas compatible avec l'interface graphique. En effet, la boucle continue de demander des catégories jusqu'à ce que l'utilisateur entre "non". Or dans l'interface, ceci ne se génère pas de cette manière. Et, si l'utilisateur ne saisit jamais "non", la boucle continue de tourner.

Nous avons donc commencé par mettre notre liste CAT en variable globale car elle est utilisée plusieurs fois dans notre programme, sans cela, CAT en variable locale n'aurait pas fonctionné.
Puis nous avons créé un dataframe vide reprenant les colonnes de fusion qui nous sera utile dans la suite du code.
Ensuite, nous avons créé une boucle à travers la liste CAT pour récupérer les catégories saisies par l'utilisateur.
Nous avons stocké les lignes qui respectent le budget dans un nouveau dataframe A.

```
def cadeau():
    global CAT #variable globale

    #Création d'un dataframe vide avec les colonnes souhaités
    result = pd.DataFrame(columns=fusion.columns)

    #On fait une boucle à travers la liste CAT pour récupérer les catégories saisies par l'utilisateur
    for i in CAT:
        categorie = i.get()
        if categorie:
            df1 = fusion.loc[fusion["category_name"] == categorie] #Filtrer fusion pour ne garder que les lignes
            result = pd.concat([result, df1]) #concatenation
        else :
            result = fusion


    #On récupère le choix du budget
    budget = choix_budget.get()
    A = result.loc[result["price"] <= float(budget)] # Garder les lignes avec un prix inférieur ou égal au budget
```

Par la suite, on crée une variable proposition_produit qui récupère le choix de l'utilisateur concernant la spécification ou non d'un type de produit.
Si l'utilisateur veut choisir un type de produit, alors on filtre les produits selon le type choisi et on tri en fonction des notes, des avis et des meilleures ventes.
Dans le cas inverse, si l'utilisateur ne spécifie pas de type, les résultats sont simplement triés en fonction des mêmes critères précédents.

```
if Proposition_produit == 'oui':
    Type = choix_Type.get()
    #On filtre les libellés de produit pour garder que le type choisi par l'utilisateur
    #Puis on trie en fonction des notes, des avis, et des meilleurs ventes
    d1 = A[A['title'].str.contains(Type, case=False)]
    d1 = d1.sort_values(['stars', 'reviews', 'isBestSeller'], ascending=[False, False, False])

    #On filtre les libellés de produits pour ne garder que les produits dont le titre n'a pas le type spécifié
    #Puis on trie en fonction des notes, des avis, et des meilleurs ventes
    d2 = A[~A['title'].str.contains(Type)]
    d2 = d2.sort_values(['stars', 'reviews', 'isBestSeller'], ascending=[False, False, False])

    AA = pd.concat([d1, d2]) #concaténation

# Si l'utilisateur ne veut pas spécifier de type, on trie juste les résultats
else:
    AA = A.sort_values(['stars', 'reviews', 'isBestSeller'], ascending=[False, False, False])
```

Comme pour les deux fonctions précédentes, on a créé une boucle for qui parcours le dataframe AA avec *itterows()*. Pour chaque ligne, les valeurs des colonnes 'productURL', 'imgURL' et 'title' sont

extraites puis utilisées pour créer des labels qui affichent les liens des produits et de l'image sur l'interface graphique. Les labels sont ensuite ajoutés à la zone de texte grâce à la méthode *window_create*.

```
#création de la zone de texte
position1 = 1.0 #position de départ
position2 = tk.END #position d'arrivée
zone_texte.delete(position1, position2) #Supprimer la recherche précédente de la 1ère ligne à la dernière ligne

for index, row in AA.iterrows(): #retourne l'index de la ligne et toutes les données de la ligne sous forme de séries
#on parcours les lignes du dataframe
    lien1 = row['productURL'] #on garde pour chaque ligne/série que l'element "productURL" pour avoirs les liens
    lien2 = row['imgUrl']
    zone_texte.insert(position2, row['title']) #on insère à la fin de la zone de texte une seule chaîne de caractère q

    lien_label1 = tk.Label(zone_texte, text=lien1, fg="blue", cursor="hand1")
    #on crée un label qui doit être ajouté à la zone de texte
    lien_label1.bind("<Button-1>", lambda e, l=lien1: lien_cliquable(l))
     #On lie l'évenement "clic gauche" à la fonction "lien_cliquable". La fonction lambda appelle la fonction "lien_cl

    zone_texte.window_create(position2, window=lien_label1)

    lien_label2 = tk.Label(zone_texte, text=lien2, fg="blue", cursor="hand1")
    #on crée un label qui doit etre ajouté à la zone de texte
    lien_label2.bind("<Button-1>", lambda e, l=lien2: lien_cliquable(l))
      #On lie l'évenement "clic gauche" à la fonction "lien_cliquablet". La fonction lambda appelle la fonction "lien_

    zone_texte.window_create(position2, window=lien_label2)

    zone_texte.insert(tk.END, "\n")
```

Pour que ces liens soient cliquables, nous avons, comme pour les autres fonctions, créé une fonction lien_cliquable avec le module webbrowser qui permet d'ouvrir le lien dans le navigateur.

```
def lien_cliquable(lien): #permet de rendre le lien cliquable
    webbrowser.open_new(lien)
```

Voici un aperçu de notre code :

```
import os
import pandas as pd
import tkinter as tk
import webbrowser

os.getcwd()

# Importation des données
A_categories = pd.read_csv("../../Data/amazon_categories.csv", sep=",")
A_products = pd.read_csv("../../Data/amazon_products.csv")

A_categories=A_categories.rename(columns = {'id': 'category_id'})
fusion = pd.merge(A_products,A_categories, how="outer", on=["category_id"])

CAT = [] #création d'une liste vide

def cadeau():
    global CAT #variable globale

    #Création d'un dataframe vide avec les colonnes souhaités
    result = pd.DataFrame(columns=fusion.columns)

    #On fait une boucle à travers la liste CAT pour récupérer les catégories saisies par l'utilisateur
    for i in CAT:
        categorie = i.get()
        if categorie:
            df1 = fusion.loc[fusion["category_name"] == categorie] #Filtrer fusion pour ne garder que les lignes correspondanst à la catégorie saisie
            result = pd.concat([result, df1]) #concatenation
        else :
            result = fusion


    #On récupère le choix du budget
    budget = choix_budget.get()
    A = result.loc[result["price"] <= float(budget)] # Garder les lignes avec un prix inférieur ou égal au budget

    #On récupère le choix du type
    Proposition_produit = choix_produit.get()

    #Si l'utilisateur veut choisir un type de produit
    if Proposition_produit == 'oui':
        Type = choix_Type.get()
        #On filtre les libellés de produit pour garder que le type choisi par l'utilisateur
        #Puis on trie en fonction des notes, des avis, et des meilleurs ventes
        d1 = A[A['title'].str.contains(Type, case=False)]
        d1 = d1.sort_values(['stars', 'reviews', 'isBestSeller'], ascending=[False, False, False])

        #On filtre les libellés de produits pour ne garder que les produits dont le titre n'a pas le type spécifié
        #Puis on trie en fonction des notes, des avis, et des meilleurs ventes
        d2 = A[~A['title'].str.contains(Type)]
        d2 = d2.sort_values(['stars', 'reviews', 'isBestSeller'], ascending=[False, False, False])

        AA = pd.concat([d1, d2]) #concaténation

    # Si l'utilisateur ne veut pas spécifier de type, on trie juste les résultats
    else:
        AA = A.sort_values(['stars', 'reviews', 'isBestSeller'], ascending=[False, False, False])


    #création de la zone de texte
    position1 = 1.0 #position de départ
    position2 = tk.END #position d'arrivée
    zone_texte.delete(position1, position2) #Supprimer la recherche précédente de la 1ère ligne à la dernière ligne

    for index, row in AA.iterrows(): #retourne l'index de la ligne et toutes les données de la ligne sous forme de séries
    #on parcours les lignes du dataframe
        lien1 = row['productURL'] #on garde pour chaque ligne/série que l'element "productURL" pour avoirs les liens
        lien2 = row['imgUrl']
        zone_texte.insert(position2, row['title']) #on insère à la fin de la zone de texte une seule chaîne de caractère qui e

        lien_label1 = tk.Label(zone_texte, text=lien1, fg="blue", cursor="hand1")
        #on crée un label qui doit être ajouté à la zone de texte
        lien_label1.pack()
        lien_label1.bind("<Button-1>", lambda e, l=lien1: lien_cliquable(l))
         #On lie l'évenement "clic gauche" à la fonction "lien_cliquable". La fonction lambda appelle la fonction "lien_cliqua

        zone_texte.window_create(position2, window=lien_label1)

        lien_label2 = tk.Label(zone_texte, text=lien2, fg="blue", cursor="hand1")
        #on crée un label qui doit être ajouté à la zone de texte
        lien_label2.pack()
        lien_label2.bind("<Button-1>", lambda e, l=lien2: lien_cliquable(l))
          #On lie l'évenement "clic gauche" à la fonction "lien_cliquablet". La fonction lambda appelle la fonction "lien_cliq

        zone_texte.window_create(position2, window=lien_label2)

        zone_texte.insert(tk.END, "\n")

def lien_cliquable(lien): #permet de rendre le lien cliquable : on pourra accéder au site Amazon et à l'image
    webbrowser.open_new(lien)
```

```python
def ajouter_categorie(): #fonction appelée lorsque l'utilisateur clique sur le bouton "Saisir une catégorie supplémentaire"
    #on crée un nouveau label et une nouvelle zone de saisie pour que l'utilisateur puisse ajouter une catégorie
    label = tk.Label(fenetre, text="Saisir une nouvelle catégorie :", font=("Arial", 16, "bold"), fg='dark violet')
    label.pack()
    entree = tk.Entry(fenetre)
    entree.pack()
    CAT.append(entree) #cette nouvelle entrée est ajouté à la liste CAT

#création de l'interface
fenetre = tk.Tk()
fenetre.title("cadeau")
fenetre.geometry('900x1200')

fenetre.configure(bg='pink')

Titre_label = tk.Label(fenetre, text="Trouver le Cadeau Idéal !", font=("Palatino", 30, "bold"), fg='MediumPurple1')
Titre_label.pack()

#Création du label catégorie (correspond au 1er choix de catégorie)
cat_label = tk.Label(fenetre, text="Saisir une catégorie :", font=("Arial", 16, "bold"), fg='dark violet')  # Violet foncé
cat_label.pack()
choix_cat = tk.Entry(fenetre) #création d'une zone de saisie permettant la saisie de l'utilisateur
choix_cat.pack()
CAT.append(choix_cat) #cette nouvelle entrée est ajouté à la liste CAT

#Bouton qui permet l'ajout d'une ou plusieurs catégorie selon le choix de l'utilisateur
bouton_ajout_cat= tk.Button(fenetre, text="Ajouter une catégorie supplémentaire",  font=("Arial", 10, "bold"), fg='dark violet',command=ajouter_categ
bouton_ajout_cat.pack()

#Création du label budget dans lequel l'utilisateur pourra saisir son budget
budget = tk.Label(fenetre, text="Saisir votre budget :", font=("Arial", 16, "bold"), fg='dark violet')
budget.pack()
choix_budget = tk.Entry(fenetre)
choix_budget.pack()

#Création du label type qui permet à l'utilisateur de préciser si il veut un type de produit ou non
Proposition_produit = tk.Label(fenetre, text="voulez-vous préciser le type de produit (oui/non) :",  font=("Arial", 16, "bold"), fg='dark violet')
Proposition_produit.pack()
choix_produit = tk.Entry(fenetre)
choix_produit.pack()

#Création d'un nouveau label en réponse au label précédent: si l'utilisateur veut un type de produit, il le précisera dans ce label
type_produit = tk.Label(fenetre, text="Si oui choisissez un type de produit :", font=("Arial", 16, "bold"), fg='dark violet')
type_produit.pack()
choix_Type = tk.Entry(fenetre)
choix_Type.pack()

#Création d'un bouton recherche qui appelle la fonction cadeau :
#lorsque l'utilisateur aura saisit ses critères, il appuiera sur ce bouton afin de lancer la recherche
Bouton_cadeau = tk.Button(fenetre, text="C'est parti !", font=("Arial", 18, "bold"), fg='dark violet', command=cadeau)
Bouton_cadeau.pack()

#Création d'une zone de texte qui affichera les résultats de la recherche (liée à la fonction cadeau)
zone_texte = tk.Text(fenetre, width=60, height=10)
zone_texte.pack()

fenetre.mainloop()

zone_texte.configure(font=("Times New Roman", 25, "italic")) #configuration d'une police de texte

fenetre.mainloop()
```

Voici l'interface graphique finale : *par exemple, si l'utilisateur : saisie la catégorie "Laptop Accessories", souhaite ajouter une nouvelle catégorie "Fabric Decorating", saisie un budget de 78,98€ et choisit un produit de type Battery alors il obtiendra les résultats suivants*.

VIII - INSTRUCTIONS D'EXÉCUTION DU PROGRAMME

Afin d'exécuter notre programme, veuillez ouvrir JupyterLab ou Jupyter Notebook pour accéder au premier fichier nommé "analyse_partie_1.ipynb" situé dans le répertoire code principal". À l'intérieur de ce fichier, vous trouverez l'ensemble de nos analyses de la partie 1 ainsi que les interprétations associées à chacun de nos résultats.

Ensuite, veuillez ouvrir Spyder pour exécuter la partie 2 du programme. Assurez-vous au préalable que tous les dossiers sont dans le même répertoire que celui où vous travaillez. Commencez par exécuter le script "data_load", qui importe les deux bases de données "Amazon_categories" et "Amazon_products". Ensuite, exécutez le script "data_clean()".

Une fois ces deux scripts exécutés, vous devriez disposer des deux dataframes A_products et A_categories, ainsi que du dataframe résultant de leur fusion. Ensuite, exécutez le script "recherche_1". Pour tester la fonction "recherche_produit", saisissez "battery" comme nom de produit et "Laptop accessories" comme nom de catégorie, puis choisissez "prix". Cela devrait vous fournir le résultat attendu.

Ensuite, exécutez le script "recherche_2" en testant la fonction de fusion de la manière suivante : "Voulez-vous trier par notes ou par prix ?" (Par exemple, "prix") et en spécifiant la catégorie ("Laptop Accessories").

Enfin, testez le script "recherche_3" avec la fonction "cadeau". Vous avez également la possibilité de tester directement le script "search", qui englobe l'ensemble des scripts en les utilisant comme modules et en exploitant les fonctions qu'ils contiennent. *Veuillez exécuter les fonctions séparément, car sinon elles se répèteront à l'infini. Nous n'avons pas trouvé de solutions à ce problème, veuillez-nous excusez pour ce désagrément.*

Nous avons choisi de ne pas créer un module pour la partie 3 concernant l'interface graphique utilisant la bibliothèque Tkinter, en raison de difficultés d'organisation. Vous trouverez donc les trois interfaces correspondent aux différentes fonctions de la partie 2, dans le dossier "Interface". Exécutez ce script dans son intégralité pour afficher toutes les fenêtres permettant de répondre aux questions de la partie 2, mais de manière différente.
Chacune de ses interfaces est à exécuter seule une à une.

IX - DIFFICULTÉS RENCONTRÉES

En cette première année de master, nous avons, et ce pour la majorité des groupes, rencontrer les personnes membre de notre équipe que très récemment. Nous avons toutes dû apprendre à travailler ensemble, mais également individuellement. En effet, coder en python ne faisait pas forcément partie de nos acquis.
Ce qui nous a finalement confrontés à des difficultés de groupes mais également individuelles voyons en quoi consistent ces dernières.

i. Les difficultés au sein du groupe

Commençons premièrement par les difficultés au sein de notre groupe.
La première difficulté à laquelle nous avons été confrontés, était celle du travail collaboratif et plus particulièrement au sein du codage. En effet, durant ces dernières années, nous avions pour la plupart, toutes déjà collaboré sur tout un tas de projets. Mais ce fut la première fois que nous avons dû le faire pour coder. Et le problème est que lorsque l'on code nous dépendons très souvent du code précédemment réalisé.
Ainsi, après quelques recherches nous nous sommes tournés vers la solution la plus simple pour nous : un outil de codage collaboratif. Mais avant ceci, comme tout repose sur l'utilisation des bases de données d'Amazon, nous avons créé un dossier contenant ces dernières afin de pouvoir procéder par chemin relatif, fonctionnant ainsi pour nos quatre ordinateurs.

Par la suite, nous nous sommes heurtés à un tout autre problème : le rythme du codage.
Effectivement, nous nous sommes rendu compte de l'importance de la coordination au sein de notre groupe pour ce type de situation. Il nous était impossible de réaliser de notre côté et au moment où on le souhaite notre partie de code, sans en parler entre nous.

La majorité était indépendante, mais pour certains cas, avoir les codes précédents s'avérait être une nécessité. Nous avons également dû assigner les différentes tâches en fonction des disponibilités de chacun, tout en prenant en compte les compétences des uns et des autres.

Nous avons également dû faire face à des soucis de compréhension. Mais pas ce auquel nous pouvons habituellement être confrontés au sein d'un groupe. Dans notre cas, il s'agissait de compréhension de code. En effet un code pouvait se montrer très simple pour la personne ayant codé mais pour les autres membres du groupe, la démarche pouvait paraître difficile à comprendre. Cependant il était d'une importance cruciale de ne pas passer à côté d'une partie du projet pour que chacune d'entre nous puisse en retirer tous les bénéfices. Ainsi nous avions décidé, et ce malgré la distance durant les vacances, de s'organiser des calls pour s'expliquer le plus clairement possible ce qui a été réalisé. Nous nous sommes également retrouvées dans des salles de classe au sein de la MSE pour que chacune d'entre nous puisse expliquer ses idées au reste du groupe.

La quatrième difficulté à laquelle nous avons été confrontés était celle de la confrontation d'idées. En effet, pour faire un code, plusieurs méthodes sont envisageables mais il faut savoir laquelle choisir.
Pour nous, la question n'était pas si simple que ça, nous savions que par la suite, il faudrait expliquer ce code. Nous savions également qu'il était préférable qu'il soit le plus court possible. Ainsi lorsque plusieurs idées étaient proposées pour un même code, nous avons pris le temps de réfléchir au plus favorable. Et, dans certains cas (souvent pour les idées les moins longues à matérialiser) nous avons réalisé les deux codes pour voir en pratique lequel permettra la concrétisation de nos ambitions.

Pour finir, nous nous sommes également rendu compte de l'importance de l'anglais. Malgré que de nombreuses ressources existent pour Python, avec notamment une grande partie en français. Nous avons pris conscience que l'anglais restait une langue importante concernant les ressources liées codage.
De fait, nous avons parfois perdu du temps en passant par des ressources françaises, moins riches que certaines beaucoup plus claires, mais en anglais.

Ce projet nous a donc offert une perspective différente du travail en groupe. Nous avons été confrontées à divers défis, tels que le rythme, l'organisation du code, la langue, les ressources ou encore le choix des concepts. Néanmoins, nous sommes reconnaissantes d'avoir été confrontées à toutes ces difficultés, qui nous ont permis d'apprendre à collaborer en groupe en Python.

ii. Les difficultés individuelles

Nous avons également fait face à des difficultés individuelles. Certaines communes à plusieurs d'entre nous et certaines plus personnelles. En effet, chaque membre du groupe apportait avec elle son environnement de compétences ainsi que sa personnalité. Et c'est dans cette perspective que nous avons pu nous heurter à certains obstacles que voici.

La première consistait à avoir l'idée de départ concernant le code. En cours, nous avions l'habitude de coder directement sur notre ordinateur, mais pour le projet (notamment les questions les plus complexes) nous nous sommes rendu compte qu'il fallait au préalable, structurer nos idées. Ainsi

cette étape, qui pourrait paraître minime de prime abord, permettait un gain de temps considérable par la suite. Parfois, lorsque nous codons, nous oublions l'idée générale du code. Mais lorsque nous passions au préalable sur feuille, l'idée générale était concrète et cela nous permettait de garder le fil conducteur pour aboutir au résultat désiré.

La seconde difficulté était liée aux réflexes de codage. En effet, nous restons débutantes dans le domaine. Nous nous sommes rendu compte qu'après avoir concrétisé notre idée sur feuille, l'envie d'aboutir le plus vite possible aux résultats, nous faisait parfois aller trop vite, à en oublier les réflexes de codage. Souvent, nous laissions en second plan, beaucoup de détails comme par exemple donner des noms parlants à nos fonctions et à nos variables, mettre des inputs (on a préféré au départ commencé par des variables) ou encore commenter notre code.
Seulement, une fois notre travail terminé, tous ces détails omis lors de la création du code nous ont pris un temps considérable. Ce temps aurait pu être réduit si nous avions été pointilleuses, au fur et à mesure. Effectivement les codes réalisés au départ n'étaient plus aussi frais dans notre esprit qu'ils l'étaient auparavant et donc toutes les modifications (même les plus minimes) ont dû être précédées par une re-mémorisation de la question et du but recherché.
Ainsi, dès que nous avons compris cela, nous avons essayé de prendre les bons réflexes dès le début d'une question. Cela pouvait paraître frustrant et être une perte de temps sur le moment, mais il s'agissait néanmoins d'une étape primordiale.

La gestion du stress et l'organisation ont également été importantes durant ce projet. En effet, ce dernier suivait notre première session d'examens au sein de notre master. Ainsi, il pouvait être difficile pour nous de faire la part des choses. Se lancer sur un code pouvait prendre du temps, et nous devions également partager ce temps avec d'autres révisions (qui concernaient d'autres matières). Cependant, la réalisation de ce projet était très importante pour nous car il pouvait nous apporter énormément tant d'un point de vue professionnel, que personnel.
Nous avons donc dû faire des choix et parfois pour certaines d'entre nous délaisser nos temps de repos ou de révision dans d'autres enseignements. Combattre notre stress, tenter de s'organiser au maximum et revoir nos priorités, ont été des enjeux auxquels chacune d'entre nous a dû faire face.

Pour finir, l'apprentissage des interfaces graphiques avec tkinter a constitué une étape cruciale de notre projet, mais elle s'est avérée être la plus grande difficulté individuelle que nous ayons rencontrée. Ce module spécifique nous a plongés dans le domaine complexe du développement d'interfaces utilisateur en Python. Pour maîtriser cet aspect, une compréhension approfondie des concepts liés à la conception d'interfaces graphiques, tels que la gestion des fenêtres, des widgets et la mise en page graphique, étaient essentiels.

Cette complexité a ajouté une couche supplémentaire de défi à notre travail collaboratif. Chaque membre de l'équipe a dû assimiler ces concepts de manière individuelle avant de pouvoir les appliquer dans le contexte du projet global. La coordination des efforts pour assurer une interface utilisateur harmonieuse et fonctionnelle a exigé une communication claire et une compréhension partagée des spécificités de tkinter.

Malgré cette difficulté, cette phase d'apprentissage a été extrêmement formatrice. Elle nous a permis de développer des compétences pratiques dans un domaine essentiel du développement logiciel, renforçant ainsi notre expertise en Python. En fin de compte, surmonter cette difficulté individuelle a contribué à l'enrichissement collectif de l'équipe, démontrant notre capacité à relever des défis complexes et à acquérir de nouvelles compétences au cours du processus collaboratif.

X - Ce que le projet nous a apporté

i. Ce que le projet a apporté à Hella

En réalisant ce projet Python, j'ai appris à travailler en groupe sur des thématiques totalement différentes de ce à quoi j'étais habituée, renforçant ainsi mes compétences en communication et en résolution de problèmes. Une des découvertes les plus gratifiantes a été l'apprentissage de la modélisation des interfaces graphiques. Comprendre comment concevoir et mettre en œuvre des interfaces utilisateurs a été non seulement satisfaisant, mais m'a également permis d'élargir mes compétences en développement logiciel.

De plus, participer à ce projet Python m'a apporté une leçon précieuse sur la relativisation de mes attentes vis-à-vis de la perfection. En temps normal, dans mes projets Java, j'avais souvent tendance à chercher la perfection à tout prix. Cependant, en travaillant avec Python, j'ai rapidement compris que la nature flexible et expressive du langage permettait fréquemment des améliorations, et ce même pour des programmeurs novices comme moi. Cette expérience m'a appris à relativiser mes attentes et à me concentrer sur l'essentiel, en accordant plus d'importance à la fonctionnalité et à la compréhensibilité du code plutôt qu'à une quête incessante de perfection. Cela a non seulement facilité la collaboration au sein du groupe, mais a également ouvert la voie à une amélioration continue, un principe que j'ai depuis intégré à ma pratique du développement logiciel.

En abordant l'un de mes plus grands défis en programmation Python, celui des réflexes de codage, j'ai réalisé l'importance de créer un code clair et compréhensible, surtout lorsqu'il doit être partagé avec d'autres membres du groupe. Cette prise de conscience m'a conduit à adopter de bonnes pratiques de codage, améliorant ainsi la lisibilité et la maintenabilité de mes scripts.

En outre, ce projet Python m'a permis d'explorer des aspects tels que la gestion de projet, la documentation du code et l'optimisation du code. Apprendre à planifier des tâches, rédiger une documentation claire et optimiser le code pour améliorer les performances ont été des compétences précieuses que j'ai développées au cours de ce projet. Finalement, cette expérience m'a non seulement permis d'approfondir mes connaissances en Python, mais aussi de développer des compétences transversales essentielles pour le développement logiciel.

ii. Ce que le projet a apporté à Charlotte

Ce projet de groupe a été très enrichissant pour moi, tant du point de vue personnel que professionnel.

D’un point de vue personnel, ça m’a permis d'apprendre à travailler en groupe, d’avoir un bon esprit d’équipe et d’être toujours solidaire entre nous.
Sans une bonne cohésion d’équipe, ce projet n’aurait pas pu se mener à bien.
Ce projet m’a aussi montré qu’une bonne organisation n’engendre pas de stress. En effet, avec un partage équitable des tâches et une bonne communication, l’avancée du projet ne peut se faire que sereinement.
Par exemple, chaque semaine nous nous fixions un objectif à réaliser, ce qui nous a permis de finir le projet à temps et de ne pas avoir ce stress de bâcler la fin du projet pour le rendre dans les temps.
Ce projet m’a permis aussi de découvrir de nouvelles personnes, avec des parcours et des points forts différents. Cette diversité de connaissances nous a permis de nous compléter au sein du groupe et d’enrichir notre savoir-faire personnel.

D’un point de vue professionnel, python est langage de programmation primordial dans le monde de la data science.
Ce projet m’a permis d’améliorer mon aisance sur python . En effet, j’avais déjà pratiqué python au cours de ma licence mais mes cours été très théoriques, donc je ne pouvais pas appliquer mon savoir-faire et ainsi j’avais oublié une bonne partie de mes connaissances.
Grâce à ce projet, j’ai pu me remémorer d’anciennes connaissances ajoutées aux nouvelles que j’ai pu acquérir lors des TD.
J’ai donc redécouvert Python et ce projet m’a fait beaucoup apprécier ce langage de programmation notamment avec la diversité de choses que l’on peut faire dessus (comme les interfaces graphiques que je ne connaissais pas du tout).
De plus, python est un langage essentiel dans le monde de la data science, ainsi le savoir-faire acquis me sera forcément utile au cours de ma carrière professionnelle et ne serait-ce que pour la fin de l’année, lorsque je chercherais un stage, je pourrais présenter ce projet.

iii. Ce que le projet a apporté à Manal

Ce projet a été une expérience enrichissante qui m’a permis de développer des compétences significatives, tant sur le plan personnel que professionnel. Je pense que ce projet m’a tout d’abord appris à travailler en groupe. En licence, j’avais peu de travaux de groupe et souvent, les quelques travaux de groupe concernaient des matières littéraires (telles que l’anglais par exemple). De plus, j’avais déjà fait du python mais sans projet à rendre à la fin du semestre.

J’ai développé des compétences essentielles en travail de groupe comme la communication, l’organisation, la collaboration et l’adaptabilité. Ce sont des compétences nécessaires pour la poursuite de mes études et ma future carrière.

Premièrement, la répartition des tâches était primordiale pour avancer et rester productif. Régulièrement, nous définissions des objectifs à réaliser, ce qui m'a beaucoup motivé pour réaliser les tâches qui m’étaient attribuées. La communication avec les autres membres du groupe était également nécessaire pour ne pas avoir de soucis de compréhension. En effet, chacune devait par exemple, expliquer la démarche de construction de son code aux autres membres du groupe, ce qui nous permettait à toutes de bien comprendre l’ensemble de notre code. J’ai également appris à

demander de l'aide lorsque mes codes ne fonctionnaient pas. L'objectif était de ne pas rester bloqué et profiter au maximum des connaissances de tous les membres du groupe pour mieux avancer.

En plus des compétences en travail d'équipe, j'ai pu renforcer mes compétences en programmation Python. Le projet nous a fait revoir une bonne partie du cours vu en classe mais également d'apprendre de nouvelles choses. La partie Tkinter par exemple, est la partie sur laquelle j'ai passé le plus de temps et qui m'a appris le plus de choses.

J'ai notamment dû me documenter car je n'avais aucune connaissance sur l'interface graphique. J'ai beaucoup aimé varier mes sources pour comprendre les enjeux et les possibilités liés à l'interface graphique. De plus, j'ai bien compris que
pour coder il ne fallait pas simplement essayer plusieurs codes sur python et regarder lequel fonctionnera. En effet, ill fallait bien comprendre ce qui était demandé et
ce qu'il était possible de faire (par exemple dans l'interface graphique).

J'ai beaucoup aimé faire ce projet qui m'a permis de développer un grand nombre de compétences. Grâce à la bonne cohésion de groupe, nous avons pu mener à bien notre projet, en ne cessant de nous soutenir du début à la fin. Après un début de semestre où j'ai eu quelques difficultés à m'adapter, je pense que le projet m'a rappelé pourquoi j'avais choisi de poursuivre mes études dans ce master.

iv. Ce que le projet a apporté à Aya

Ce projet, sur lequel j'ai travaillé au cours des trois dernières semaines s'est avéré à sa fin enrichissant et très interactif. Ayant déjà suivi un cours « Introduction à la programmation sous python » l'année dernière, je n'avais jamais eu l'opportunité d'appliquer les notions vues en cours dans un cas pratique. Donc je rencontrais toujours des difficultés à digérer les bases de python vu qu'il n'y avait jamais le côté pratique de ce cours. Sur le plan personnel, en travaillant sur ce projet, j'ai pu approfondir ma syntaxe de base de python en revenant sur le cours qu'on a eu tout au long du semestre et utiliser internet de manière efficace ce qui m'a permis de développer ma capacité à penser de manière logique et d'arriver de la manière la plus efficace aux résultats attendus. J'ai appris également à faire des choses que je n'ai jamais faites comme les interfaces graphiques avec la bibliothèque Tkinter qui nous a appris la plupart du temps de travail vu le nombre de recherches effectuées et les problèmes rencontrés lors du codage de cette partie.
Sur le plan collectif, travailler en groupes m'a appris à partager le travail, échanger les idées et voir d'autres méthodes de travail, ainsi que s'entraider en cas de blocage ce qui est été largement le cas du groupe avec qui j'ai travaillé. Je pense qu'une bonne coordination permet de répartir efficacement les tâches entre les membres de l'équipe, en tenant compte des compétences et des forces de chacun.
Pour conclure, ce projet m'a confirmé mon envie d'apprendre davantage le langage de programmation. Elle a également renforcé ma conviction qu'une approche pratique est essentielle pour développer une compétence efficace en programmation. Je vous remercie ainsi pour cette opportunité qui a apporté de nombreux bénéfices pour notre groupe.