DEVOIR FINAL – LANGAGE DE PROGRAMMATION

- SUJET 2 – IMDB -

CECILE BRISSARD

MARIE-LOU BAUDRIN

SERENA GRUARIN

Sommaire

I. Introduction : Pourquoi et Comment ?

Nous avons choisi le Sujet 2 – IMDb, celui-ci nous apparaissant plus intéressant mais également plus complexe. Nous pensons qu'il sera plus valorisable sur un CV qu'une simple analyse de données, ce que nous traitons, par ailleurs dans d'autres matières du Master Econométrie-Statistiques.

Le sujet étant découpé en trois parties, nous avons décidé que chacune d'entre nous prendrait en charge une partie. Néanmoins, nous nous sommes entraidées et concertées tout au long de l'élaboration du code et de ce dossier afin que le rendu en soit optimal.

Enfin, ce sujet nous aura permis de maitriser les librairies Pandas, Seaborn et Tkinter, ce qui est un avantage pour la poursuite de nos études.

II. Partie 1 – Analyse descriptive des bases et visualisation

MARIE-LOU BAUDRIN

Dans cette partie, il s'agissait dans un premier lieu de décrire les données, la taille des dataframes, le nombre de variables, d'observations, etc… Avant de commencer, nous transformons les fichiers csv en DataFrame à l'aide de la librairie Pandas. Nous les avons appelés : *df_movies, df_names, df_title* et *df_ratings.*

Pour connaître les dimensions des DataFrames, on utilise **.***shape()* qui renvoie le nombre de lignes suivi du nombre de colonnes.

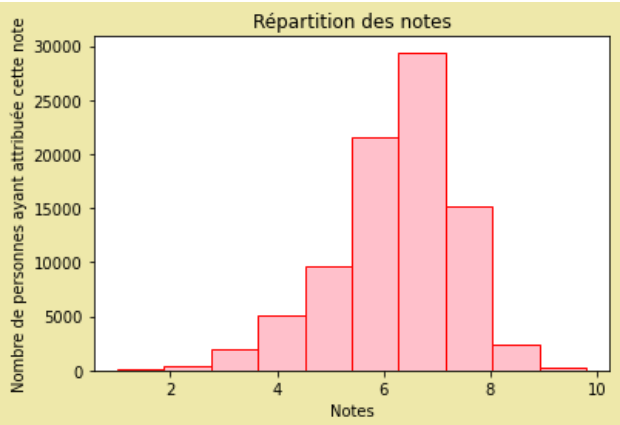
Ainsi, on sait que le DataFrame *df_movies* possède 85 855 lignes et 22 colonnes. On a donc dans ce DataFrame 22 variables pour 85 855 observations qui correspondent à 85 855 films référencés sur la plateforme IMDb.

De même, le DataFrame df_title possède 6 variables et 835 513 observations qui correspondent à 835 513 données en tous genres référencés sur IMDb qui concernent nos films, comme des acteurs, personnage, etc.

Le DataFrame *df_names* comprend 297 705 observations pour 17 variables. Notre base de données dispose donc d'informations sur 297 705 personnes du monde cinématographiques : acteurs, producteurs, scénaristes, etc.

Enfin le DataFrame *df_ratings* contient 85 855 observations pour 49 variables, soit 85 855 films. En appliquant **.***dtypes*, à nos DataFrames, il nous est retourné la liste des variables et leur type.

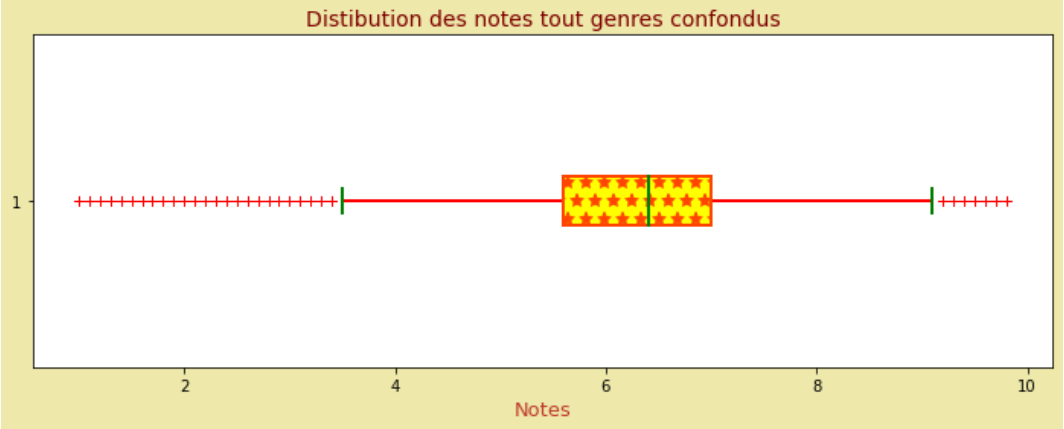
Notre premier graphique sera ce simple histogramme des notes attribuées sur IMDb. Remarquons alors que sur 85 855 votes, près de 30 000 ont eu des notes entre 6.2 et 7.1, ces notes représentent ainsi près d'un tiers des notes attribuées.



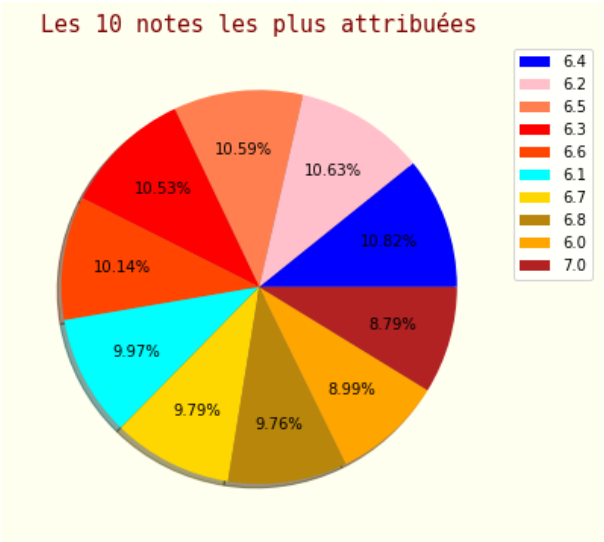
Afin d'en savoir un peu plus sur la distribution des notes, réalisons avec les mêmes données une boîte à moustache horizontale à l'aide de *plt.boxplot*. Nous en profitons pour ajouter des options pour que cette dernière soit plus esthétique. On peut voir ci-dessous que 50% des notes sont comprises entre environ 5.7 et 7.1, ce qui est un petit intervalle. La médiane se situe à environ 6.5 ce qui est relativement bon, la moitié des films ont reçu une note au-dessus et l'autre moitié en dessous.

Donnons une explication : il est possible que les films reçoivent en moyenne une assez bonne note car les consommateurs ont tendance à faire la démarche de noter un film lorsqu'ils l'ont aimé.

Les extrémités des moustaches sont calculées en utilisant 1.5 fois l'espace interquartile (la distance entre le 1er et le 3ème quartile), les observations en dehors sont marquées ici par des +.



A présent, appliquons à la variable "mean_vote" du dataframe df_ratings, *value_counts().head(10)* qui nous retourne les dix notes les plus attribuées sur IMDb. Il s'agit ici de notes comprises entre 6 et 7, ce qui est cohérent avec notre graphique précédent. Nous en faisons un Camembert avec les pourcentages de chaque note, sachant qu'ici le pourcentage est calculé sur la base des 10 notes les plus attribuées et non sur la base de toutes les notes.

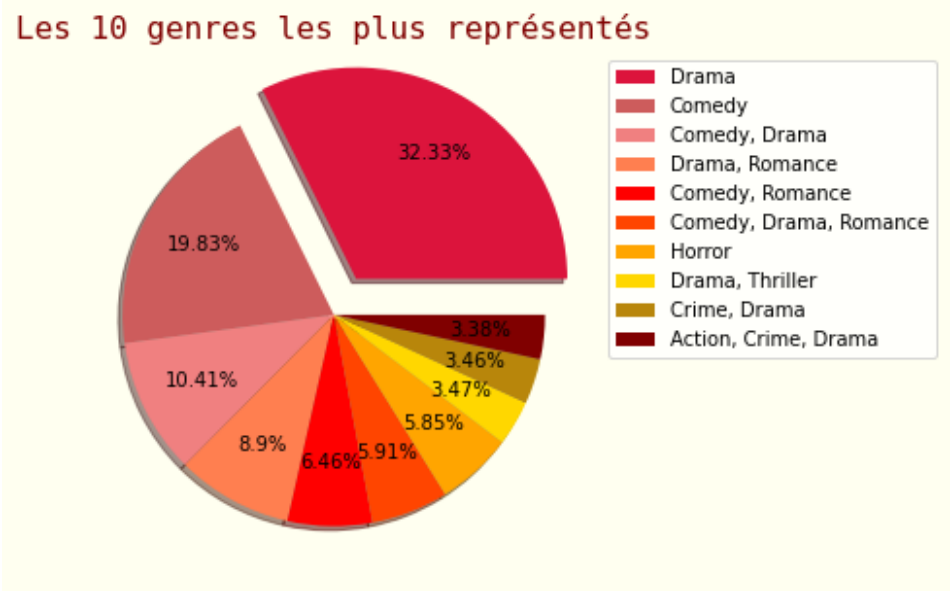


Ces dix notes représentent près de 40% des notes attribuées totales, ce qui est beaucoup, il s'agit ici de notes relativement bonnes.
Nous en savons maintenant un peu plus sur la répartition des notes tout films confondus, nous allons maintenant étudier la répartition des notes pour chaque genre.

Pour ce qui est demandé dans le sujet, très peu de variables sont utiles et elles sont toutes

comprises dans le DataFrame *df_movies*. On crée donc un nouveau DataFrame *dfm*, à partir du premier en y conservant seulement quatre variables, celles dont nous aurons besoin : le titre - 'title', l'année de sortie - 'year', la note moyenne sur IMDb – 'avg_vote' et le genre du film - 'genre'. Nous avons donc un DataFrame *dfm*, à quatre variables pour toujours 85 855 films.

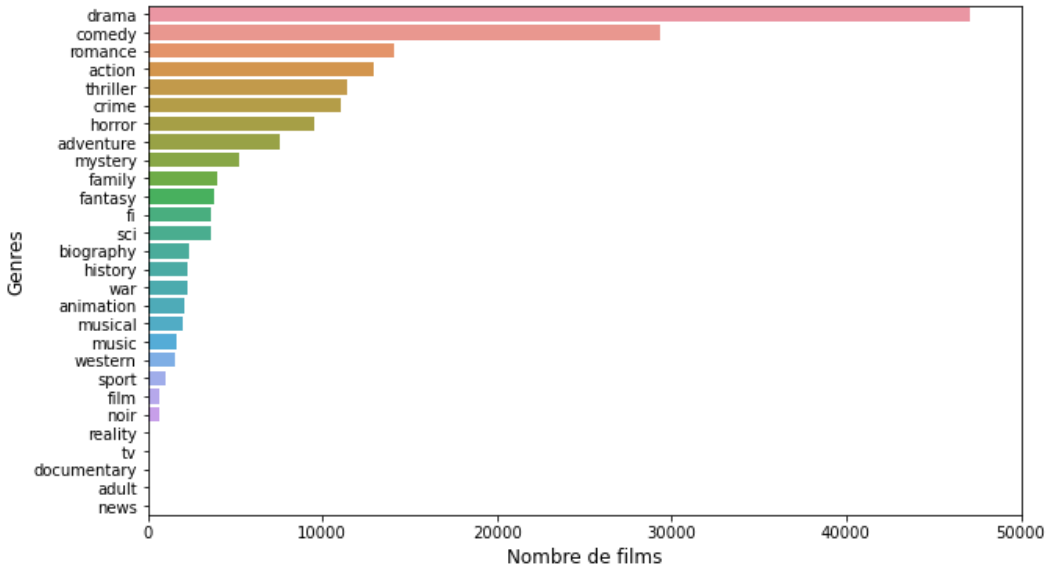
Nous voudrions connaître le genre de film le plus représenté dans les données. Comme on l'a utilisé pour les notes, on applique maintenant *.value_counts()* à la variable « genre » du DataFrame dfm. On lui demande les dix premiers genres les plus représentés dans l'ordre décroissant avec *.head(10).* On en fait un graphique avec *plt.pie(),* on voit ici que le premier genre est Drama avec 12 543 films, représentant, comme on peut le voir sur le camembert, 32,33% des films dans notre base de données.



Le problème ici est qu'il existe 1257 genres différents. En effet, certains films se voit attribué plusieurs genres (Ex : « Comedy, Romance », ou « Comedy,Aventure »), et cela compte pour un genre différent.

Importons donc *CountVectorizer* de *sklearn.feature_extraction.text* qui va séparer chaque genre et compter 1 pour chacun. En faisant analyser notre DataFrame par 'word', nous avons une liste de 28 genres "uniques" ici. Créons ensuite un DataFrame de genres uniques qu'on appelle *genres***,** puis à partir de ce nouveau DataFrame il sort une série de genres et leur compte dans l'ordre décroissante, que nous appelons *sorted_genres*.

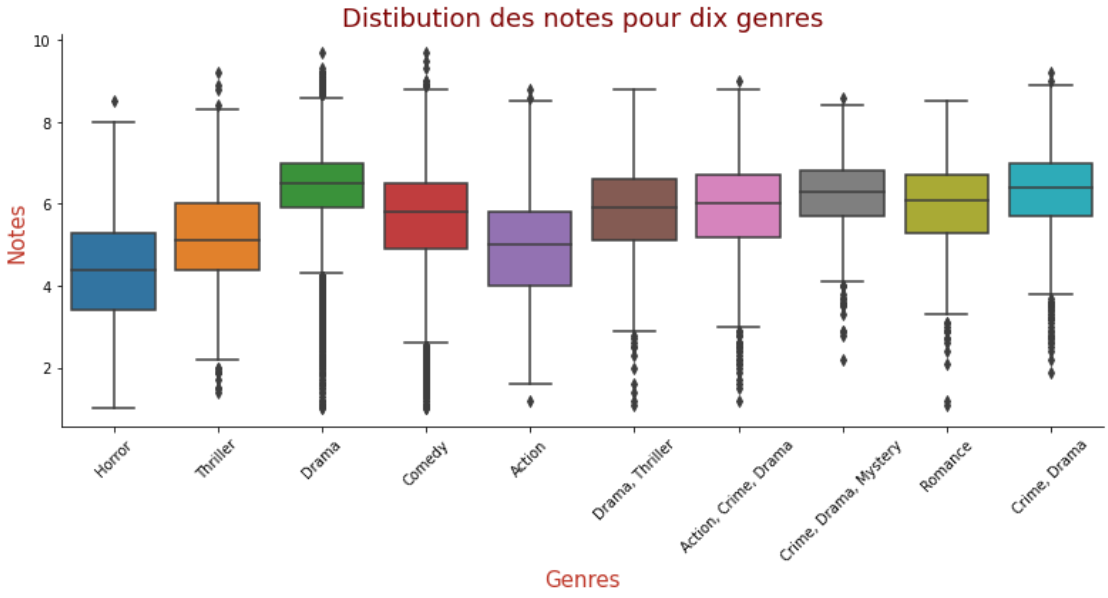
En réalisant un histogramme avec *sns.barplot()*, nous pouvons ainsi voir que le genre Drama est encore une fois, et largement, en tête.



Il est suivi de Comedy et Romance. Avec *sorted_genres.head(),* 47 110 films ont visiblement Drama comme genre, soit plus de la moitié des films de notre base de données. De même, 29 367 films ont comme genre associé Comed, contre seulement 14 127 films pour Romance. *Le genre Drama est donc le genre le plus largement représenté dans notre base de données IMDb.*

Intéressons-nous à présent à la répartition des notes pour chaque genre. Ayant 1 257 genres dans notre base de données, il serait illisible de faire un graphique de la répartition des notes pour chacun. Nous essayerons deux manières pour s'approcher au mieux de ce qui est demandé.

Dans un premier temps, récupèrons les vingt genres les plus représentés et créons, pour chacun, un sous-DataFrame avec les films appartenant à ce genre seulement.
Puis concatènons dix de ces sous-DataFrame de genres différents et réalisons la boîte à moustache ci-dessous pour chacun des genres. Nous constatons que le genre le moins bien noté est Horror.

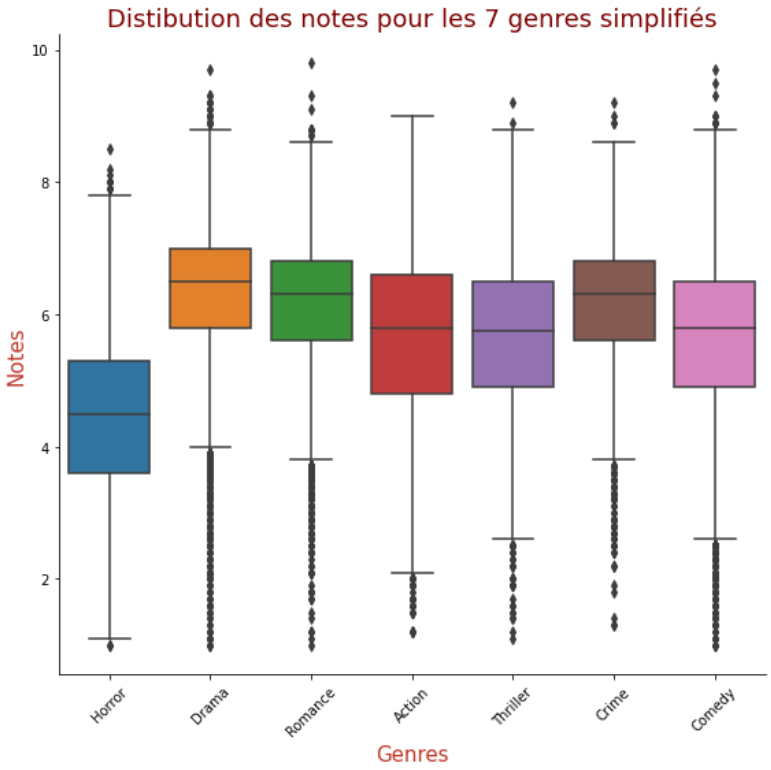


Il est aussi ici possible tenter une interprétation : peut-être que les effets spéciaux sont mal réalisés, ou que les spectateurs ont trop peur et ont donc attribué une mauvaise note. De plus, les films d'horreur ont généralement un scénario moins bien travaillé. Les films d'action et les thrillers sont également moins bien notés que la moyenne, encore une fois, peut-être que le scénario est moins bien travaillé pour les films d'action, tandis que pour les Thrillers, ils peuvent être trop "perchés" ce qui peut déplaire.

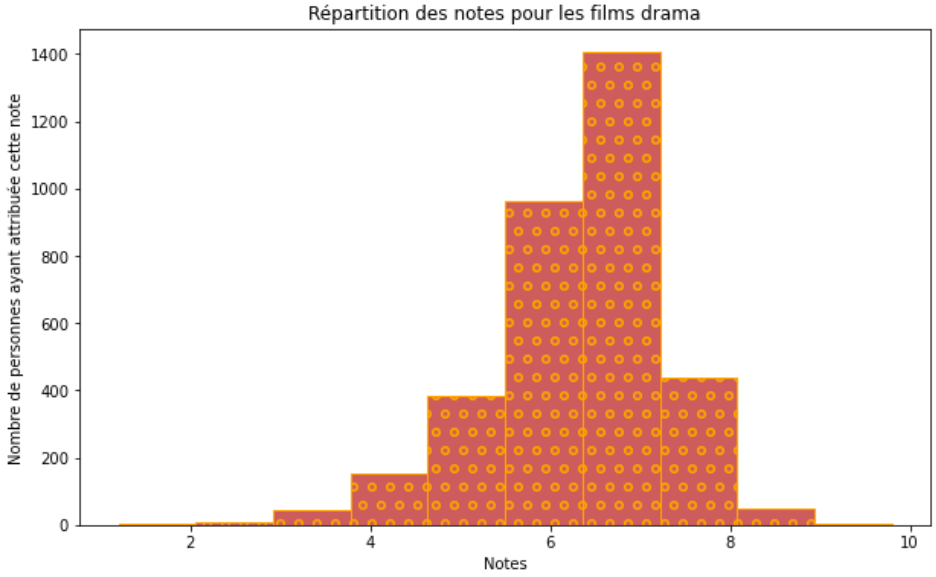
Ici, nous n'avons pas répertorié tous les genres et notre DataFrame ne contient que 29 570 films, soit un peu plus d'un tiers de notre base de données, ce qui n'est pas vraiment suffisant. Cela nous permet cependant d'avoir un bon aperçu de la répartition des notes par genre.

Pour se rapprocher davantage du résultat désiré, nous rassemblons dans un même *DataFrame* plusieurs sous-DataFrame de genres en choisissant les vingt genres les plus représentés. Par exemple, plusieurs genres appartenant à la dénomination *Thriller* sont rassemblés, de même pour *Romance*, etc.. On a alors six sous-DataFrames suivant un classement par genre : Thriller, Romance, Drama, Horror, Action et Crime ; auquel s'ajoute Comedy, aucunement modifié par nos soins.
Pour chacun de ces six DataFrames, nous effaçons la colonne de genre pour ajouter une colonne de constante que nommée genre - la constante étant le genre qui rassemble les films du sous-DataFrame. Ainsi, pour reprendre l'exemple ci-dessus, pour tous les films contenant Thriller dans leur genre, nous n'avons plus qu'un genre unique, Thriller. Il en est de même pour les cinq autres. Enfin, ces six DataFrames au genre simplifié sont concaténés, et y ajoutant le sous-DataFrame Comedy, nous obtenons un DataFrame 47 468 films, sur les 85 000 initialement présent.

Ce graphique ne peut pas être défini comme représentatif, mais il permet de donner une représentation de plus de la moitié des films parmi les genres le plus populaires.



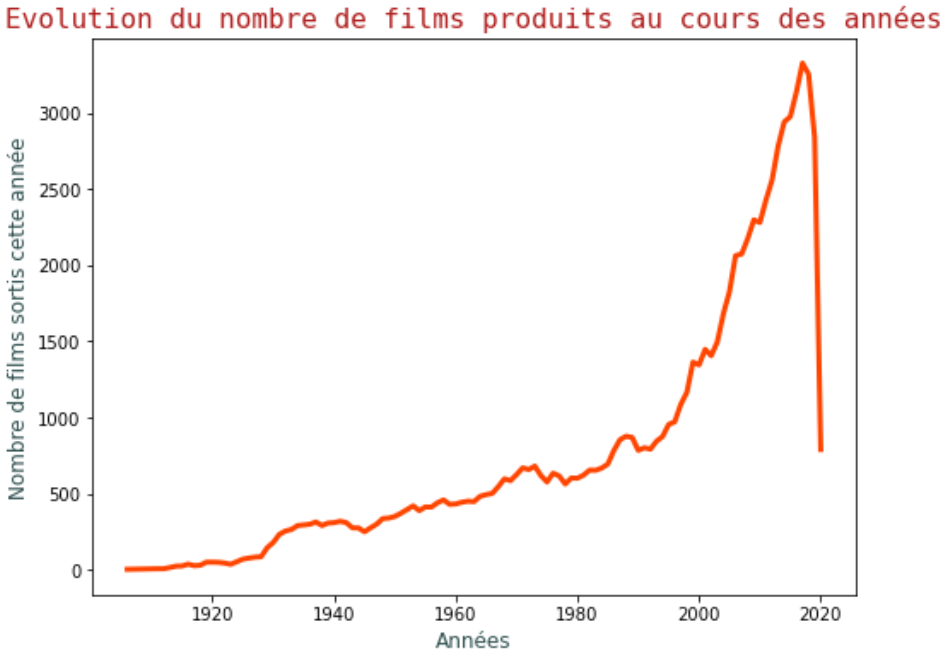
Là encore, le genre *horror* est le moins bien noté : plus de la moitié des notes sont inférieures à 5. Le genre le plus apprécié reste *Drama* avec une médiane d'environ 6.5, suivi de près par les genres *Crime* et *Romance.*
Étant le plus représenté dans notre base de données, on peut s'intéresser à la répartition des notes uniquement pour les films *Drama*.



La majorité des notes se situe entre 6.2 et 7.1, ce qui suit la répartition des notes réalisée au tout début (avec tous les genres confondus).

A présent, nous allons étudier l'évolution du nombre de films produits au cours des années. Pour cela nous utilisons *.astype(int)* pour pouvoir changer le type de la variable "year". Nous avions déjà enlevé Year Movie 2019 qui était répertorié comme une année, enlever cette variable n'est pas problématique car seulement un film y était répertorié. Avec *value_counts(),* nous avons la liste des années avec le nombre de films produit chaque année dans l'ordre décroissant. On crée un *DataFrame dfy* à partir de ces résultats, avec une colonne 'Annee' qui correspond aux index des *value_counts* et une colonne 'Nombre de films sortis cette année' qui correspond aux *value_counts*. Trions ensuite ce nouveau DataFrame par année avec *sort_values* comme nous l'avons déjà fait précédemment.

Nous visualisons à présent l'évolution du nombre de films produits par an, en réalisant un graphique avec *plt.plot* nous obtenons le visuel suivant :

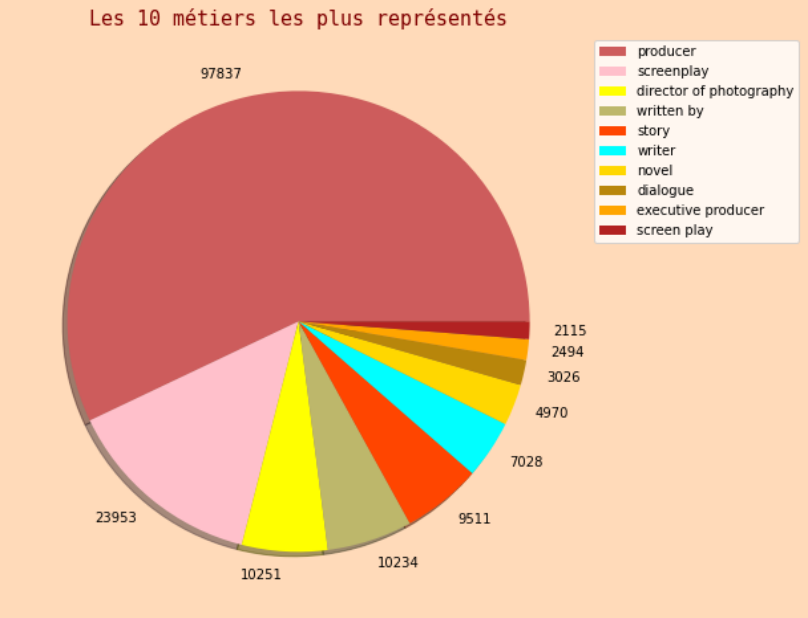


Ici, l'évolution croissante du nombre de films produits par an et ce jusqu'en 2017/2018, s'interrompt en 2020 pour finalement laisser place à une forte chute cette même année. Pour les données que nous avons, autant de films ont été produits en 2020 qu'au début des années 90. Ces résultats peuvent s'expliquer par le manque de données répertorié dans la base de données IMDb. En effet, la base de données a peut-être été réalisée en début d'année 2020 et donc les films produits au cours de cet année n'ont pas encore été répertoriés. Nous nous permettons de penser qu'avec le coronavirus beaucoup de films qui devaient être produits n'ont pas pu l'être cette année.

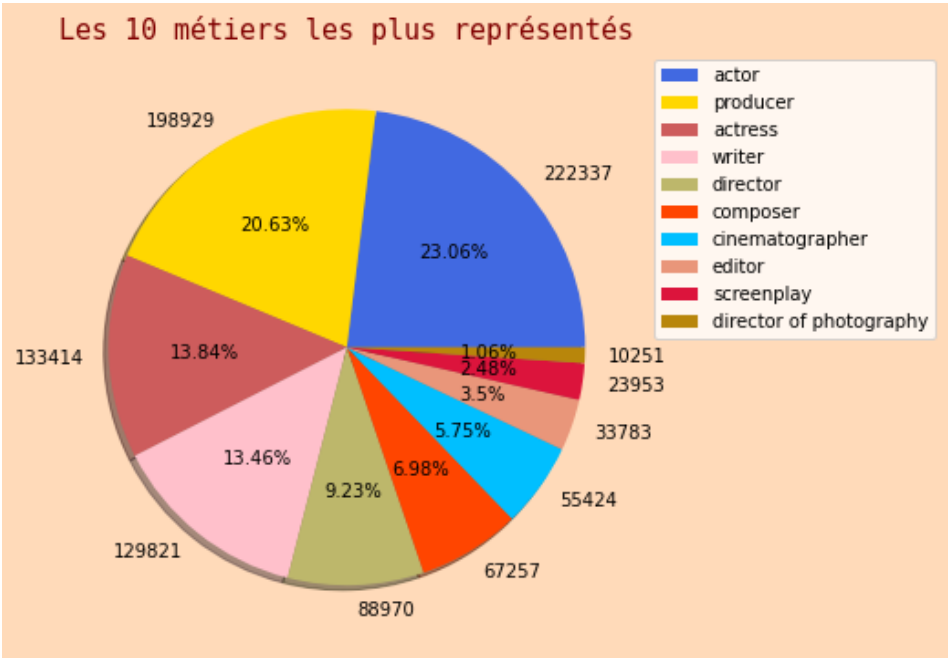
Ce graphique nous donne une réelle idée de l'évolution du nombre de films produits par an, il est évident que nous produisons plus de films maintenant qu'au siècle dernier. Au début du 20^{e} siècle, nous produisions 200 fois moins de films qu'aujourd'hui.

Nous pourrions aussi regarder les fonctions qu'exercent le personnel cinématographique, répertoriées dans le DataFrame *df_title*, pour voir quel est le métier le plus représenté dans notre base de données.

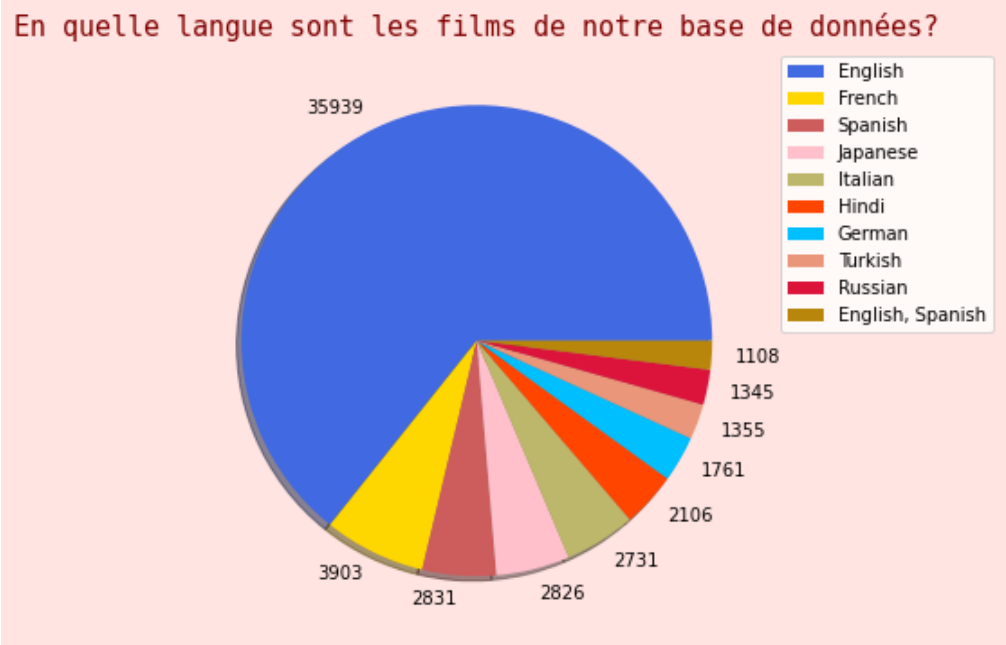
Comme pour les années ou les genres, on utilise *value_counts.head(10)* et on trouve les dix métiers les plus exercés dans notre base de données. Suite à la réalisation du camembert, nous constatons que le premier métier de producteur devance très largement les autres. Pourquoi n'avons-nous pas d'acteur dans le top 10 ?



Dans notre DataFrame, nous avons la colonne job mais aussi la colonne catégorie qui peut faire office de métier donc nos résultats sont faussés. Pour pallier à cela, on rassemble en *job*, category et job et on recommence avec *value_counts.head(10)* qui nous donne les dix métiers les plus représentés et nous en faisons un camembert.
Attention, ici les pourcentages sont calculés par rapport à la liste des dix premiers métiers et non de tous les métiers existants.



Nous pouvons aussi nous intéresser à la langue des films référencés. Dans le DataFrame *df_names*, on s'intéresse à la variable 'language' qui correspond à la langue du film et on enlève toutes les valeurs manquantes. En utilisant, une fois de plus *value_counts*, nous avons le nombre de films par langue, nous garderons ici les dix langues les plus utilisées dans les films de notre base de données. La première est l'anglais avec près de 36000 films. La seconde est le français, mais on compte dix fois moins de films dans cette langue.



Cette première partie nous a permis d'appréhender nos quatre bases de données par l'analyse de leur contenu.

Pour conclure, nous pouvons alors affirmer que nous disposons pour la suite de 85 855 films, majoritairement anglophones, 297 705 personnes du monde cinématographiques, dont 97 837 producteurs. On sait également que les films peuvent contenir plusieurs genres, mais qu'au total, la majorité sont des Drames, des Comédies, des Comédies dramatiques et des Romances dramatiques.

III. Partie 2 – Moteur de recherche

SERENA GRUARIN

Il s'agissait donc dans cette partie de créer le moteur de recherche, sur lequel on se basera dans la Partie III pour créer l'interface.
Nous avons décidé de créer cela via un script sur Spyder.

Grâce à l'analyse des quatre bases de données effectuées à la Partie I, nous avions des connaissances sur les informations dont nous disposions. Pour chaque fonctionnalité demandée, nous savions donc quelles informations piochées dans les différents csv.
Ces derniers ont dû donc tout d'abord être convertis en DataFrame à l'aide de la librairie Pandas et renommés : *df_movies, df_names, df_title et df_ratings*.

La principale difficulté rencontrée dès le début était de trouver comment appeler telle ou telle information relative à un film ou un acteur. Ne sachant pas comment faire en sorte de retourner une information (par exemple l'année de sortie) en ne rentrant que le titre d'un film mais néanmoins qu'il est possible d'appeler une valeur d'un dataframe à l'aide de l'index et du nom de la colonne : df.loc['Index', 'nomdelacolonne'], nous avons voulu récupérer la valeur de l'index. Or, il est impossible de connaître l'index, c'est-à-dire le numéro de la ligne d'un film particulier. Nous avons donc contourné ce problème, de différentes manières selon les fonctionnalités.

La première fonctionnalité que nous avons créée était celle de retourner des informations précises relatives à un film, qui est la fonction information(). Cette fonction n'a pas été construite comme les autres car nous n'étions pas encore très à l'aise avec la manipulation de DataFrame. La solution que nous avons trouvée en premier était de fonctionner par des dictionnaires. Nous avions vu, lors des cours, que le dictionnaire fonctionne sous forme de couple clé : valeur, et qu'en appelant une clé, cela retourne la valeur. Il suffisait donc de créer plusieurs dictionnaires avec comme clé le nom du film, et comme valeur l'information que l'on souhaitait obtenir.
Après avoir extrait la colonne 'title' de df_movies en une liste, nous avons créé une fonction permettant pour n'importe quelle colonne de ce DataFrame de l'extraire en liste, et de la fusionner en dictionnaire avec la liste des titres (fonction fusion()).

Nous voulions également récupérer certaines informations du DataFrame df_ratings. Or, les différents films sont représentés par leur identifiant, que l'utilisateur ne peut pas connaître. De plus, on ne pouvait pas fusionner notre liste de titres avec les colonnes qui nous intéressaient puisque les films ne sont pas dans le même ordre.
En effectuant la même procédure que précédemment mais en fusionnant cette fois-ci, la liste des identifiants avec les informations qu'on souhaitait (fonction fusio()), nous avons également créé un dictionnaire associant les titres à son identifiant. Ainsi, si on appelle la clé de ce dictionnaire qui est le titre, cela nous retourne son identifiant, qu'on peut ensuite appeler en tant que clé pour avoir les informations de df_ratings.

Pour la fonction information il suffisait donc d'appeler la clé. Par exemple, y est le dictionnaire associant les titres à leur date de sortie, donc pour avoir la date de sortie d'un film il faut écrire : y['titredufilm'].

Par la suite, nous avons trouvé une manière plus simple d'appeler des informations, mais nous avons souhaité garder cette fonction comme telle pour montrer qu'il y a plusieurs façons de coder et d'obtenir les résultats.

Pour la seconde fonctionnalité common_actors(), qui permet de retourner le ou les acteurs communs à deux films, nous avons procédé différemment : par création d'un unique DataFrame regroupant les quatre, en les fusionnant par les identifiants des films et des noms. Nous avons également changé le type de la colonne title, en string, pour nous permettre d'effectuer la fonction sans soucis (à la base, il y avait des inputs mais on les a enlevés pour faire l'interface en partie III).
Common_actors fonctionne en créant un sous-DataFrame contenant seulement les valeurs liées au film que l'utilisateur entre, et en extrait la colonne name en liste, qui est donc le nom des acteurs. Il suffit ensuite de retourner les éléments communs aux deux listes.

Pour la fonction common_movie, qui permet de retourner le ou les films communs à deux acteurs, nous avons procédé de la même manière, mais à l'inverse.

La troisième fonctionnalité permet de retourner des informations sur un acteur, ainsi que les films dans lesquels il a joué, ses co-acteurs et les notes reçues.
Nous avons ici récupéré ces informations en changeant l'index des DataFrames de sorte à retourner les valeurs par df.loc['Index', 'Nomdelacolonne'].
Pour les co-acteurs et les notes reçues, on a simplement créé des boucles qui parcourent la liste des films.

Enfin, pour top_movie et single_or_not, nous avons utilisé le même procédé de création de sous dataframe relatif à une valeur précise (df[df['Nomdelacolonne']=='valeurprécise']).

L'ensemble de ce script constitue alors le module sur lequel on se base dans la partie III.

IV. Partie 3 – Interface

CECILE BRISSARD

Dans cette troisième et dernière partie de code, l'objectif est de créer une interface représentant le moteur de recherche avec le module Tkinter de Python. Celle-ci est ensuite associée aux fonctions créées dans la partie précédente.

L'interface tkinter demande l'importation du module en question via *import tkinter as TK* ou *from tkinter import* *. Une fois cette première commande effectuée, nous disposons de tous les objets et fonctions nécessaires à l'élaboration de la page de recherche.
Nous avons choisi d'effectuer une *class*, appelée SearchEngine() comprenant toutes les fonctions utilisées pour la construction de l'interface et servant son efficacité en tant que moteur de recherche cinématographique. La classe implique que chacun des objets soit appelé à l'aide de *self*., c'est pourquoi il sera l'argument de toutes les fonctions.

La première fonction *__init__(self)* vise à créer la fenêtre Tkinter nommée fen1 et permet notamment de paramétrer sa taille, sa couleur et si elle est redimensionnable ou non. Dans cette fonction, on appelle d'ores et déjà la fonction suivante beau_ty(self) afin qu'elle s'implémente. Enfin le *mainloop()* est une fonction qui permet de lancer l'ouverture de la fenêtre.

Connectée à cette fonction on trouve entre autre la fonction suivante : beau_ty(self) qui va nous permettre de créer les différents éléments de la fenêtre et de les organiser géographiquement via d'une part, *.pack()* en choisissant l'une des quatre orientations *BOTTOM, TOP, LEFT, RIGHT* ; et d'autre part *.grid()* permettant de placer en fonction du rang (ex: *row=2*) et de la colonne (ex: *column=6*) les *widgets* sur une grille ainsi construite. Ces deux fonctions de placement permettent également d'établir des marges à l'aide de *padx* et *pady* afin de préciser le placement des *widgets* ; il est également possible de choisir une orientation au sein même d'une case de la grid, en y affectant via l'option *sticky* les points cardinaux.

Au sein de la fonction beau_ty() sont initialisés dans la fenêtre fen1, un *Frame*, un *Canvas* et un *Bouton* nommé 'EXIT'. Lorsque l'utilisateur appuie sur ce dernier widget, il déclenche la fonction *exitengine()* qui est construite sur la fonction *destroy()* pour fermer la fenêtre fen1 : l'utilisateur quitte alors le moteur de recherche. Après avoir testé l'utilisation de *.pack()*, nous avons finalement préféré disposer les widgets, les Canvas, le Bouton et le Frame par utilisation de la *.grid()*. La manipulation des placements est plus complexe mais cette complexité ajoute une précision, une fois l'instruction maîtrisée le rendu est donc plus agréable.
De même que pour la fenêtre, Frame, Bouton et Canevas peuvent être colorés, dimensionnés, encadrés, espacés, définis avec divers reliefs grâce aux options disponibles entre les parenthèses lors de leur création.

A l'intérieur du frame ainsi construit et nommé 'Screen', nous avons défini un Canvas contenant le texte portant le nom du moteur de recherche, l'image utilisée comme logo et positionnée en choisissant des points cardinaux (anchor=N,S,SE,NE,NW,SW…) ; ainsi que deux Entry qui nous seront utile comme barre de recherche. Sont également créés via

*.create_*widget une *List* associée à une *Scrollbar* donnant l'illusion d'un menu déroulant et un bouton nommé 'Search'. Ce dernier lance la fonction suivante : search().

Cette dernière fonction compile les fonctions définies dans la Partie II et recherche, pour les différentes variables string (chaîne de caractère, via StringVar()) reconnue dans le DataFrame *data* à partir des variables initialisées, la réponse associée à la demande de l'utilisateur. Il se doit également de choisir le type d'information qu'il attend en retour via la sélection d'un objet dans le menu déroulant. De fait, lorsque l'utilisateur entre - s'il souhaite obtenir les informations d'un film -, un titre, la fonction information() et son résultat sont appelés ; lorsqu'il entre deux films - pour la sélection de « Common actors between two movies » dans la liste - la fonction common_actors() et son résultat sont appelés ; et de même pour chaque argument définit dans les fonctions suivantes.

Enfin la fonction search() donne l'instruction d'afficher l'output dans le second Canvas *can2* situé sous la barre de recherche, et rendu visible dans l'interface par son carré rouge. Pour ce faire, nous utilisons *.create_text* auquel nous affectons le résultat de recherche via l'option text : le résultat *search_result* - associé à l'appel de la fonction choisie par l'utilisateur lorsque la boucle *if/elif* est parcourue -, est affiché dans can2. Pour l'esthétisme, il lui est affecté comme à tous les textes définis par nos soins, une couleur de police ; une *font* : option qui permet de choisir la police, sa taille et les rendus 'italic','bold'… ; une position.
Nous avons ajouté *.strip()* afin d'obtenir une string sans les espaces avant et après la phrase, *self.can2.delete("all")*, après chaque if/elif afin d'effacer les résultats d'une fonction et de les remplacer par le nouveau résultat, ainsi que *.cruselection()* qui permet de garder la sélection de l'utilisateur même s'il touche la Scrollbar est activée.

Ces dernières lignes de code produisent donc l'effet, essentiel, que l'utilisateur se situe bien sur la page d'un moteur de recherche, qui n'est autre que l'interface *Tkinter* construite comme décrit ci-dessus.

V. Difficultés rencontrées

1.1 Au niveau du groupe

Globalement, nous n'avons pas rencontré énormément de difficultés.
Les missions de chacune étaient claires, puisque nous étions chacune en charge d'une partie, mais sans jamais non plus être complétement seules face à nos difficultés étant donné que nous nous sommes entraidées.
Le fait d'être à distance était un peu problématique, notamment pour préparer la soutenance, mais nous avons surmonté cela en restant beaucoup en contact, tant par messages, appels, ou lors de sessions zoom.

1.2 Au niveau individuel

1.2.1 Marie-Lou BAUDRIN

Pour ce qui est de mes difficultés personnelles, j'ai travaillé particulièrement sur la première partie qui est supposée comme la plus simple. J'ai néanmoins eu plusieurs soucis et ai passé beaucoup de temps sur certaines questions. Je me suis donc aidée d'internet et également des cours que vous nous aviez dispensés.

Pour ce qui est de l'évolution du nombre de films produits par an, il m'a au départ été très difficile de réaliser, à partir des value_counts, un dataframe. Après avoir cherché sur différents sites et essayé différentes manières, j'ai fini par y arriver mais le code était beaucoup trop compliqué pour ce qu'il fallait faire. J'ai persisté et ai fini par trouver une manière beaucoup plus simple d'atteindre mon objectif, ce qui m'a pris seulement 20 minutes pour le faire alors que j'y avais passé 3heures la veille. Ceci m'a bien prouvé que c'est en codant qu'on apprend réellement. Je n'avais jamais utilisé Python auparavant et je ne connaissais rien à la programmation mais ce projet m'a montré qu'en persistant, il est possible d'y arriver et surtout que tout vient avec le temps.

Une autre grande difficulté fut les genres : en effet, dans notre base de données nous avions 1257 sous-genres et donc cela était illisible pour faire n'importe quel graphique et faussait les résultats. Pour les notes, j'ai essayé deux manières différentes pour m'en approcher au maximum, la deuxième solution prend en compte plus de la moitié des films de notre base de données. Elle me paraît acceptable car elle réunit les vingt genres les plus représentés et représente plus de la moitié des films.

Mais pour connaître le genre le plus représenté, il a également fallu que je montre si chaque sous genre était découpé, ce que ça donnerait comme résultats. Ici, j'ai trouvé sur internet plusieurs réponses au même souci qui conseillait d'utiliser Countvectorizer afin de séparer chaque sous genre en plusieurs genres uniques. Cela permet de compter chaque unique genre et de voir lequel est réellement le plus représenté. Dans mon cas, cela n'a pas changé la réponse mais ça aurait pu, c'était donc important pour moi de le faire.

Enfin, pour pallier les difficultés de chacune, nous avons fait des zooms avec mes camarades, en partageant chacune notre écran nous avons pu expliquer nos problèmes plus facilement et ensuite les résoudre ensemble.

1.2.2 Cecile BRISSARD

Au cours de la réalisation de la troisième partie, la première difficulté est celle que j'ai rencontrée à la lecture du DataFrame IMDb_Movies ; rapidement palliée grâce à l'utilisation du chemin d'accès suivant : *(r'chemin d'accès avec* *', sep=',', dtype=str),* permettant de définir toutes les valeurs du fichier CSV comme chaîne de caractère. J'ai, également, pris soin de ne pas ouvrir le fichier CSV initial mais uniquement sa copie afin de ne pas en déformer le contenu et d'éviter les messages d'erreur au niveau du Tokening ou du Parser des datas.

Par la suite, en tentant de réaliser un merge des quatre fichiers CSV kaggle en un seul et unique DataFrame, il a fallu effectuer beaucoup de recherche, notamment parce que certaines colonnes mélangeaient index et titres, donc des variables différentes au sein de la même colonne ; mais aussi, parce que les colonnes des quatre fichiers ne portaient pas toujours les mêmes noms. C'est finalement, ma collègue Serena qui aura résolu cette problématique par élaboration de listes et de dictionnaires.

Enfin lors de la création de l'interface Tkinter, je n'ai pas rencontré de problème particulier qui me paraisse insurmontable et j'ai tous pu les résoudre en me documentant sur différents forum et sites traitant de ce module. Le placement des Widgets, Frame, Canvas et autres Boutons, fu facilité dés que j'ai pu maîtriser, pas à pas, la *grid().*

La principale difficulté rencontrée restera pour moi l'association des fonctions de Serena à mon interface graphique Tkinter : la connexion entre nos deux scripts a longtemps été un problème de taille. Cependant après l'ajout de .strip() et de .cruselection(), les fonctions se lance bien lorsqu'elles sont appelées !

Malgré les différentes difficultés rencontrées, ma maîtrise des classes et des fonctions, ainsi que du module Panda étant améliorée, je suis ravie d'avoir eu l'opportunité de réaliser un tel projet. J'ai adoré utiliser le module Tkinter que j'ai découvert ses dernières semaines : j'aspire à en approfondir les possibilités.

1.2.3 Serena GRUARIN

J'étais donc en charge de la partie II : la création du moteur de recherche.
La principale difficulté que j'ai dû surmonter reste la manipulation des dataframes. J'ai passé beaucoup de temps sur la documentation de Pandas pour chercher des solutions pour pouvoir piocher les informations que l'on souhaitait quand l'utilisateur rentre tel ou tel film, acteur, genre ect …

J'ai également eu du mal à coder les fonctions demandées pour qu'elles soient ensuite compatible avec l'interface de la 3ème partie. Par exemple, j'avais commencé par utiliser des inputs et des prints, pour vraiment coller à l'idée d'un moteur de recherche (c'est-à-dire créer un environnement pour l'utilisateur qui ne lui demande aucun code à taper), mais, Cécile et moi, nous sommes aperçues, lors du commencement de la partie III, que cela ne fonctionnerait pas. Je devais donc, à chaque avancement de l'interface, modifier mon script.

Cependant, tout ce que j'ai appris lors de ce projet dépasse ces difficultés. Ainsi, je maîtrise maintenant beaucoup mieux la librairie Pandas et donc la manipulation de DataFrame, ce qui est un avantage pour la poursuite de mes études.

De plus, comme nous avons travaillé en étroite collaboration entre nous, je maîtrise également mieux les compétences requises pour les 1ère et 3ème parties.

Je sais donc comment mener une analyse descriptive de données ainsi que visualiser ces données. Et, concernant la partie III, je connais maintenant le module Tkinter qui m'était jusque-là inconnu. J'en possède désormais quelques notions (les fenêtres, Label, Button, Canvas).

VI. Instructions pour exécuter le programme

L'ensemble du livrable comporte plusieurs scripts :

- Un jupyter notebook correspondant à la partie 1, c'est-à-dire à l'analyse descriptive et graphiques des bases de données.
- Un script Spyder nommé script_partie_2 qui correspond donc à la partie 2, un module importé dans la partie 3 : il n'est donc pas nécessaire de l'exécuter seul.
- Et enfin, un script Spyder nommé script_partie_3, qui correspond à la partie 3 et donc à l'interface.

Le moteur de recherche final s'obtient par l'exécution du dernier script, qui se suffit à lui-même. Nous vous envoyons l'ensemble des scripts comme trace de l'évolution de notre réflexion. Mais de fait, la partie 1 n'est pas intégrée dans l'interface, et le script de la partie 2 a été utilisé pour faire celui de la partie 3.

Voici les étapes à suivre pour la bonne exécution du Script :

1. Lors de l'exécution du script, il vous sera demander le chemin d'accès aux 4 fichiers CSV, ainsi que le chemin d'accès à la photo présente dans le livrable (le « logo » de notre moteur de recherche).

2. Vous devrez ensuite, dans l'interface, taper ce que vous chercher dans l'une et/ou l'autre barre de recherche et sélectionner l'information que vous attendez dans le menu déroulant comme suit :

Si vous souhaitez des informations sur un film, il vous faut inscrire le nom d'un film dans la première barre de recherche, puis sélectionner « Information about a movie » dans le menu déroulant.

Si vous souhaitez connaitre le ou les acteurs communs à deux films, vous devrez donc entrer deux films (un dans chaque barre de recherche) et sélectionner « Common actors between two movies ».

Si vous souhaitez au contraire connaitre le ou les films communs à deux acteurs, vous devrez entrer deux noms d'acteurs et sélectionner « Common films between two actors ».

Si vous souhaitez connaitre le top 3 des films les mieux notés selon le genre et l'année, vous devrez entrer le genre (en anglais) dans la première barre de recherche, et une année dans la seconde, puis sélectionner « A top 3 ».

Si vous voulez avoir la biographie d'un acteur et la liste de ses films, ainsi que ses co-acteurs et les notes reçus, vous devrez cette fois sélectionner « An actor's biography », après avoir inscrit le nom de l'acteur dans la première barre de recherche.

Enfin, si vous voulez le statut matrimonial d'un acteur, vous devrez entrer le nom d'un acteur dans la barre de recherche, et sélectionner « Is this actor/actress single ? ».

3. Cliquez sur le bouton 'Search' et voyez les informations apparaître dans le cadre rouge !