Département de Mathématique et Informatique

PROJET

Moteur de recherche de films

Réalisé par :

KASDI Oumaima
DOMMANE Hajar

2020-2021

Contents

Présentation du projet:

On peut définir un **système de recherche** comme une forme spécifique de filtrage de l'information visant à présenter les éléments d'information (films, musique, livres, news, images, pages Web, etc.) qui sont susceptibles d'intéresser l'utilisateur dans sa recherche.

les systèmes de recherches se basent principalement sur Le traitement automatique du langage **Natural Language Processing (NLP)** qui est généralement composé de deux à trois grandes étapes:

- Pré-traitement : une étape qui cherche à standardiser du texte afin de rendre son usage plus facile.
- Représentation du texte comme un vecteur : Cette étape peut être effectuée via des techniques de sac de mots (Bag of Words) ou Term Frequency-Inverse Document Frequency (Tf-IdF).
- Classification, trouver la phrase la plus similaire….

Pour ce projet nous allons analyser les critiques des **top 100 films sur le site IMDb** a fin de répondre a des requêtes de recherches différentes tel que :

- La recherche des informations d'un film par un titre donnée.
- La recherche d'un film en se basant sur l'analyse des sentiments des avis .
- Les films similaires à une description donnée .
- Les films similaires à une catégorie donnée .

Voici donc les étapes que nous allons parcourir :

1. Récupération des données
2. Préparation des données
3. Modélisation
4. Résultat

La solution adoptée :

Nous utiliserons le langage Python pour programmer ce système vu qu'il est riche en termes de packages et librairies , et chacune joue un rôle spécifique dans le domaine du traitement du langage naturel (NLP) et l'analyse des sentiments.

Le Web Scraping :

Le web scraping est la récupération de données de pages web, de façon automatique. C'est une technique, basée sur un principe simple. Qui sert à de nombreuses applications : Moteurs de recherche, comparateurs de prix, outils de monitoring etc...

Le scraping ou crawling se fait en deux étapes : *le téléchargement,* du code HTML de la page à scraper, et la Récupération d'informations (Parsing) en se basant principalement sur le package Python **requests**.

On va voir en détails les étapes effectués dans le chapitre suivant.

Les packages Python :

SpaCy : SpaCy est une librairie qui offre des modèles pré-entraînés pour diverses applications et effectuer les opérations d'analyse sur les textes. . SpaCy est la principale alternative à NLTK.

NLTK : (Natural Language Tool Kit) il va nous servir a nettoyer notre corpus.

sklearn.feature_extraction: Pour la transformation (tf-idf) Term Frequency - Inverse Document Frequency . Avec tf-idf, au lieu de représenter un terme dans un document par sa fréquence brute (nombre d'occurrences) ou sa fréquence relative (nombre de termes divisé par la longueur du document), chaque terme est pondéré en divisant la fréquence des termes par le nombre de documents dans le corpus contenant le mot.

pandas : Pour manipulation et l'analyse des données.
numpy : Pour manipuler des matrices ou tableaux multidimensionnel.
Textblob : Pour le traitement de données textuelles. Il fournit une API simple pour des tâches courantes de traitement du langage naturel (NLP) telles que le balisage d'une partie du discours, l'extraction de phrases nominales, l'analyse des sentiments, la classification etc.

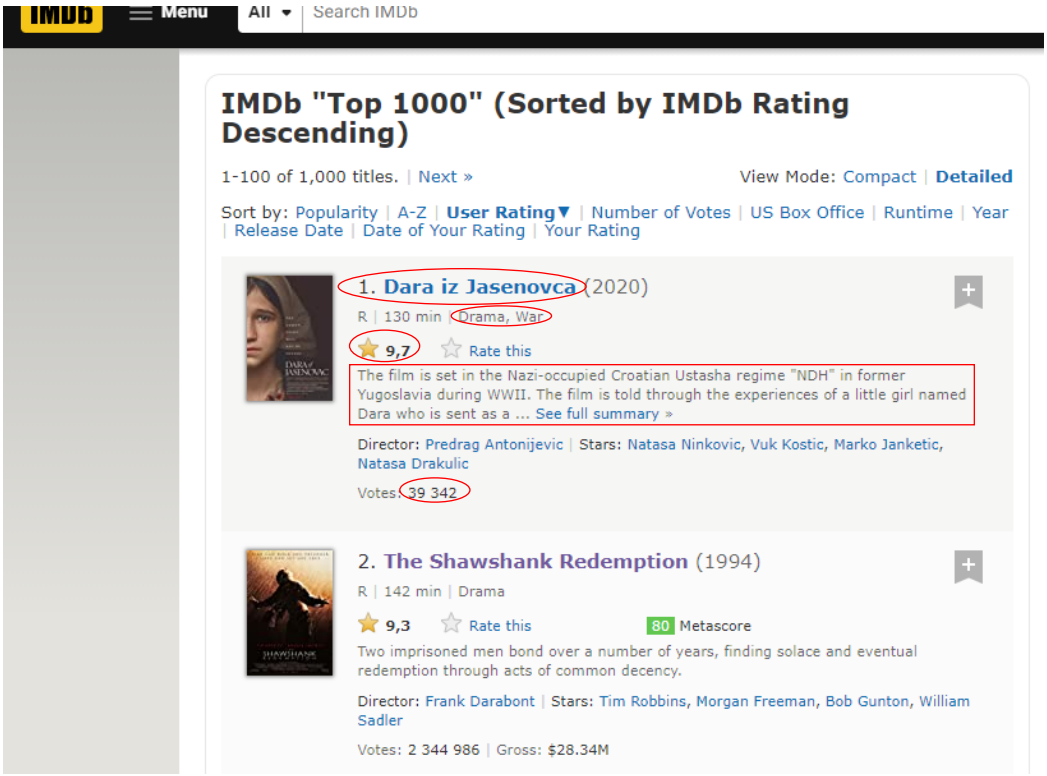
Démonstration du système :

Data_Scrapping :

PARTIE1 : Les données des films :

La 1ére partie est le scraping Web, nous allons créer un ensemble de données à partir des 100 films les mieux notés par les utilisateurs sur IMDB:

Connection à la page Web :



Premièrement il faut vérifier que le package beautifulsoup1 est installé .

beautifulsoup4: nous permet d'analyser le HTML du site et de le convertir en un objet BeautifulSoup, qui représente le HTML sous la forme d'une structure de données imbriquée.

```
[ ]  import bs4
     import pandas as pd
     import requests
```

requests: Le package qui nous permet de connecter le site de choix.

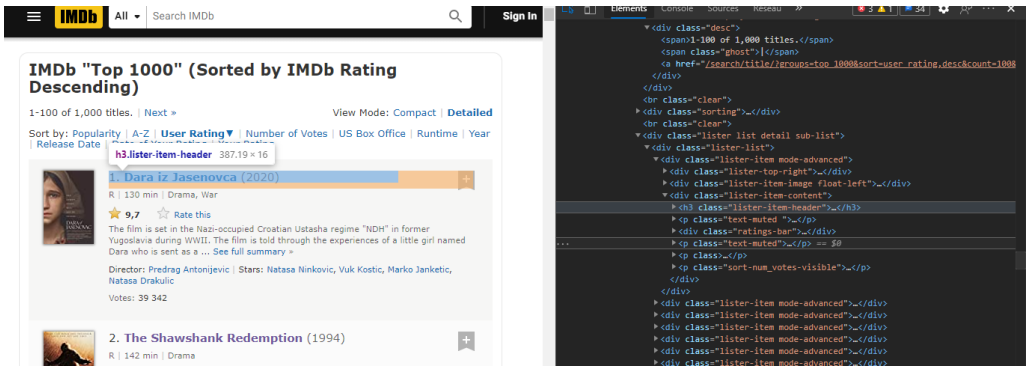
pandas: le package Python pour la manipulation des ensembles de données.

On se connecte à la page Web des 100 top films IMDb, extraire le code HTML et le convertir en objet BeautifulSoup:

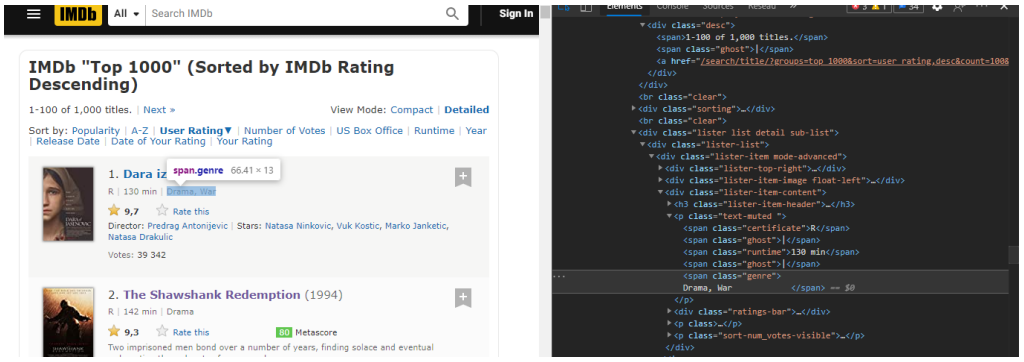
```
[18] url = 'https://www.imdb.com/search/title/?count=100&groups=top_1000&sort=user_rating'
     def get_page_contents(url):
         page = requests.get(url, headers={"Accept-Language": "en-US"})
         return bs4.BeautifulSoup(page.text, "html.parser")

     soup = get_page_contents(url)
```

Nous allons commencer à extraire (les tags capturés ci-dessous) :

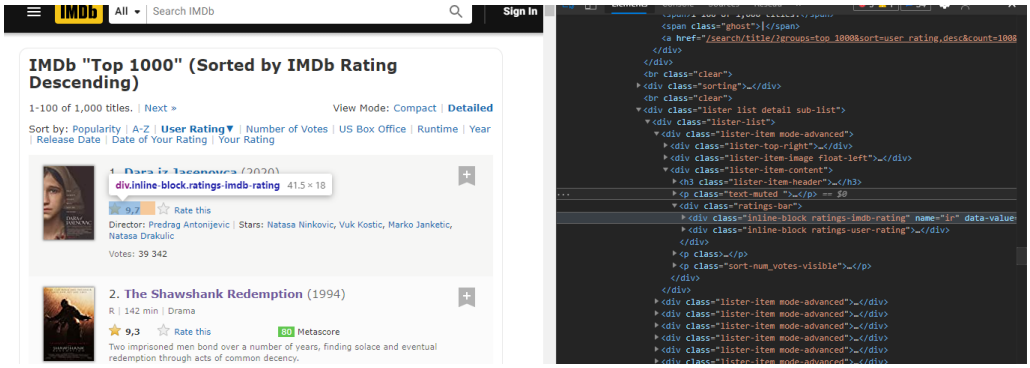
- Movie title



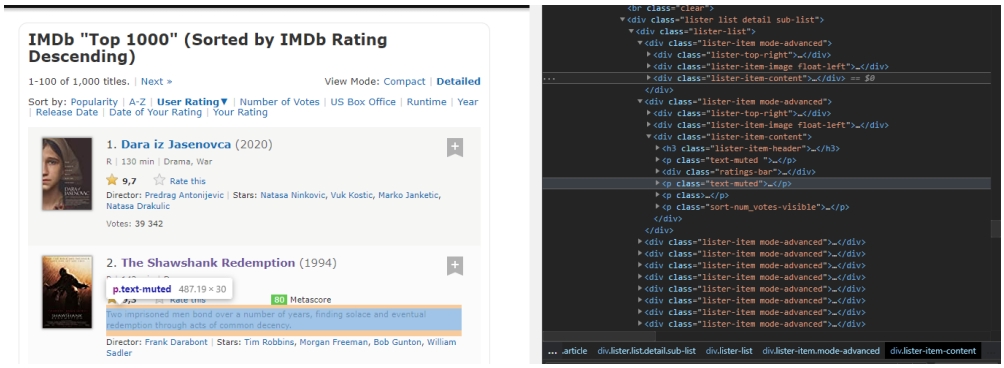
- Genre



- IMDB rating



- Description



Fonctions d'extraction :

Nous allons créer quelques fonctions d'assistance pour extraire les informations correctes selon que nous saisissons des valeurs de chaîne, numériques ou imbriquées.

- Numeric_value()
- Text_value()
- Nested_text_value()

En plus de cela, nous allons créer quelques conditions pour renvoyer None dans le cas où un film n'a pas de genre, de description ...

Et en fin la fonction d'extraction

- Extract_attribute()

```
[6] def numeric_value(movie, tag, class_=None, order=None):
        if order:
            if len(movie.findAll(tag, class_)) > 1:
                to_extract = movie.findAll(tag, class_)[order]['data-value']
            else:
                to_extract = None
        else:
            to_extract = movie.find(tag, class_)['data-value']

        return to_extract


    def text_value(movie, tag, class_=None):
        if movie.find(tag, class_):
            return movie.find(tag, class_).text
        else:
            return


    def nested_text_value(movie, tag_1, class_1, tag_2, class_2, order=None):
        if not order:
            return movie.find(tag_1, class_1).find(tag_2, class_2).text
        else:
            return [val.text for val in movie.find(tag_1, class_1).findAll(tag_2, class_2)[order]]


    def extract_attribute(soup, tag_1, class_1='', tag_2='', class_2='',
                          text_attribute=True, order=None, nested=False):
        movies = soup.findAll('div', class_='lister-item-content')
        data_list = []
        for movie in movies:
            if text_attribute:
                if nested:
                    data_list.append(nested_text_value(movie, tag_1, class_1, tag_2, class_2, order))
                else:
                    data_list.append(text_value(movie, tag_1, class_1))
            else:
                data_list.append(numeric_value(movie, tag_1, class_1, order))

        return data_list
```

Apres l'extraction on va transformer tout en une trame de données pandas propre via le script suivant:

```
[13] titles = extract_attribute(soup, 'a')
```

```
[14] genre = extract_attribute(soup, 'span', 'genre')
```

```
[15] imdb_rating = extract_attribute(soup, 'div', 'inline-block ratings-imdb-rating', False)
```

```
[16] df_dict = {'Title': titles,'Genre': genre, 'IMDB Rating': imdb_rating,'description':description}
```

```
[17] df= pd.DataFrame(df_dict)
     df
```

Résultat:

Ca nous donne :

	Title	Genre	IMDB Rating	description
0	Dara of Jasenovac	\nDrama, War	\n\n9.7\n	\n The film is set in the Nazi-occupied Cro...
1	The Shawshank Redemption	\nDrama	\n\n9.3\n	\n Two imprisoned men bond over a number of...
2	The Godfather	\nCrime, Drama	\n\n9.2\n	\n An organized crime dynasty's aging patri...
3	The Dark Knight	\nAction, Crime, Drama	\n\n9.0\n	\n When the menace known as the Joker wreak...
4	The Godfather: Part II	\nCrime, Drama	\n\n9.0\n	\n The early life and career of Vito Corleo...
...	...	...	...	...
95	Eternal Sunshine of the Spotless Mind	\nDrama, Romance, Sci-Fi	\n\n8.3\n	\n When their relationship turns sour, a co...
96	Amélie	\nComedy, Romance	\n\n8.3\n	\n Amélie is an innocent and naive girl in ...
97	Snatch	\nComedy, Crime	\n\n8.3\n	\n Unscrupulous boxing promoters, violent b...
98	Requiem for a Dream	\nDrama	\n\n8.3\n	\n The drug-induced utopias of four Coney I...
99	American Beauty	\nDrama	\n\n8.3\n	\n A sexually frustrated suburban father ha...

100 rows × 4 columns

Sauvgarder le resultat:

Il faut créer un repertoire sur votre Drive où le fichier Data.CSV va se sauvgarder :

```
[ ]  df.to_csv('/content/drive/My Drive/projet_TM/Data.csv' , encoding="UTF-8",sep =";" , index = False)
```

PARTIE2: Les données des reviews :

Connection à la page et collecter les liens URL de chaque film :

```
[27] url="https://www.imdb.com/search/title/?count=100&groups=top_1000&sort=user_rating"
```

```
# find all a-tags with class:None
movie_tags = soup.find_all('a', attrs={'class': None})

# filter the a-tags to get just the titles
movie_tags = [tag.attrs['href'] for tag in movie_tags
              if tag.attrs['href'].startswith('/title') & tag.attrs['href'].endswith('/')]

# remove duplicate links
movie_tags = list(dict.fromkeys(movie_tags))

print("There are a total of " + str(len(movie_tags)) + " movie titles")
print("Displaying 10 titles")
movie_tags[:10]
```

```
[29] # movie links
     base_url = "https://www.imdb.com"
     movie_links = [base_url + tag + 'reviews' for tag in movie_tags]
     print("There are a total of " + str(len(movie_links)) + " movie user reviews")
     print("Displaying 10 user reviews links")
     movie_links[:10]
```

Ça donne :

```
There are a total of 100 movie user reviews
Displaying 10 user reviews links
['https://www.imdb.com/title/tt10554232/reviews',
 'https://www.imdb.com/title/tt0111161/reviews',
 'https://www.imdb.com/title/tt0068646/reviews',
 'https://www.imdb.com/title/tt0468569/reviews',
 'https://www.imdb.com/title/tt0071562/reviews',
 'https://www.imdb.com/title/tt0050083/reviews',
 'https://www.imdb.com/title/tt0167260/reviews',
 'https://www.imdb.com/title/tt0110912/reviews',
 'https://www.imdb.com/title/tt0108052/reviews',
 'https://www.imdb.com/title/tt1375666/reviews']
```

Lister les review :

Pour chaque lien de film on va collecter les avis dans une liste **list_review[]** et pour chaque avis le score associé dans **r_view[] :**

```
[30] # get a list of soup objects
     movie_soups = [get_page_contents(link) for link in movie_links]
```

```
[31] pages_reviews=[msoup.findAll('div', class_='lister-item-content') for msoup in movie_soups]
```

```
[32] list_revews=[]
     for page_reviews in pages_reviews:
       list_revews.append([rf.find('a','title').text for rf in page_reviews])
```

```
[33] r_view=[]

     for page_reviews in pages_reviews:
       r_view.append([rf.find('span').text for rf in page_reviews])
```

Transformer en dataframe:

```
[34] import numpy as np

     l1=[]
     l2=[]
     for i in range(len(titles)):
        for j in range(len(r_view[i])):
         z=(titles[i],r_view[i][j])
         l1.append(z)
         df2 = pd.DataFrame(np.array(l1),columns=['titres', 'rates'])
        for j in range(len(list_revews[i])):
          z=(list_revews[i][j])
          l2.append(z)
          df3 = pd.DataFrame(np.array(l2),columns=['avis'])
```

```
[35] data_reviews=pd.concat([df2, df3], axis=1, ignore_index=True)
     data_reviews.columns=["title","rate","review"]
```

Résultat:

```
[36] data_reviews
```

	title	rate	review
0	Dara of Jasenovac	\n\n\n\n\n\n10/10\n	Extremely hartbreaking movie about one of the...
1	Dara of Jasenovac	\n\n\n\n\n\n10/10\n	Incredible.\n
2	Dara of Jasenovac	\n\n\n\n\n\n10/10\n	Facts\n
3	Dara of Jasenovac	\n\n\n\n\n\n10/10\n	Amazing Truth\n
4	Dara of Jasenovac	\n\n\n\n\n\n10/10\n	Dara iz Jasenovca\n
...	...	...	...
2491	American Beauty	\n\n\n\n\n\n9/10\n	The best of 1999.\n
2492	American Beauty	\n\n\n\n\n\n10/10\n	Look Closer...\n
2493	American Beauty	Jason Morales	LOOK CLOSER at this BEAUTiful AMERICAN film.\n
2494	American Beauty	\n\n\n\n\n\n10/10\n	The Beautiful Truth\n
2495	American Beauty	\n\n\n\n\n\n10/10\n	A different kind of American family\n

2496 rows × 3 columns

PARTIE2 : Search_Engine :

La deuxième partie c'est la partie de manipulations , on commence par importer la packages nécessaires :

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
import nltk
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
nltk.download('vader_lexicon')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
!pip install spacy
import spacy
```

Les fonctions de nettoyage :

Stop_word :

Pour extraire les jetons d'une chaîne de caractères en utilisant t **word_tokenize** , Il renvoie en fait les syllabes d'un seul mot. Puis **stopwords** pour éliminer les stop words du lange anglaise.

```
51]
    def remove_stop_words(corpus):
        english_stop_words = stopwords.words('english')
        word_tokens = word_tokenize(corpus)
        filtered_sentence = []
        for w in word_tokens:
            if w not in english_stop_words:
                filtered_sentence.append(w)
        return ' '.join(filtered_sentence)
```

Stemmalisation :

Nous utiliserons ici le Porter stemming algorithm **porter.stem()** pour extraire la forme de base des mots en supprimant les affixes.

```
[52] def get_stemmed_text(sentence):
        token_words=word_tokenize(sentence)
        stem_sentence=[]
        porter=PorterStemmer()
        for word in token_words:
            stem_sentence.append(porter.stem(word))
        return ' '.join(stem_sentence)
```

Lemmatisation :

Afin de lemmatiser,nous devons créer une instance du **WordNetLemmatizer () et** appeler la fonction **lemmatize ()** sur chaque mot du corpus.

```
53] def get_lemmatized_text(corpus):
        wordnet_lemmatizer = WordNetLemmatizer()
        token_words=word_tokenize(corpus)
        lem_sentence=[]
        for w in token_words:
            lem_sentence.append(wordnet_lemmatizer.lemmatize(w))
        return ' '.join(lem_sentence)
```

Lower_case :

```
[54] def lower_case(value):
        return value.lower()
```

Suppression des nombres :

```
[55] def remove_number(corpus):
       word_tokens = word_tokenize(corpus)
       caractres=[]
       for i in word_tokens:
         if not isnumeric(i):
           caractres.append(i)
       return ' '.join(filtered_sentence)
```

Suppression de ponctuation :

La méthode **isalpha ()** renvoie «True» si tous les caractères de la chaîne sont des alphabets, sinon, elle renvoie «False »

```
[56] def get_unpuncuated_text(corpus):
         words = nltk.word_tokenize(corpus)
         words=[word.lower() for word in words if word.isalpha()]
         return ' '.join(words)
```

Appel des fonctions (preprocessing) :

La fonction preprocessing fait appel a toutes les fonctions de nettoyage pour les utilisés facilement dans la suite :

```
[103]
     def preprocessing(corpus):
       for i in corpus :
         lower_case(i)
       return corpus
       corpus= remove_stop_words(corpus)
       corpus= remove_number(corpus)
       corpus=get_lemmatized_text(corpus)
       corpus=get_stemmed_text(corpus)
       corpus =get_unpuncuated_text(corpus)
       return corpus
```

Nettoyage du corpus :

Création des corpus :

La première étape de manipulation c'est de charger le corpus Nous avons donc créer des listes (titre – genre – description – avis – rating – titre_r) où nous allons stocker les données nécessaires pour le traitement :

```
58] titre=[]
    genre=[]
    description=[]
    avis=[]
    rating=[]
    titre_r=[]

    data_film1=pd.read_csv('/content/drive/My Drive/projet_TM/Data.csv' , encoding = "UTF-8",sep =";" )
    data_film2=pd.read_csv('/content/drive/My Drive/projet_TM/Data_Reviews.csv' , encoding = "UTF-8",sep =";" )
    titre.extend(data_film1["Title"])
    genre.extend(data_film1["Genre"])
    description.extend(data_film1["description"])
    avis.extend(data_film2["review"])
    rating.extend(data_film2["rate"])
    titre_r.extend(data_film2["title"])
```

Nettoyage :

On fait nettoyer les descriptions en utilisant la fonction preprocessing

Pour le score , le genre , et les avis nous avons besoin d'éliminer quelques caractères et les espaces vides :

Preprocessing de description (nettoyage)

```
[59] clean_description=[]
     for i in description:
      clean_description.append(preprocessing(i))
```

nettoyage des avis

```
[60] def clean_Avis(a):
        c=[]
        for i in a:
           x=i.replace("\n","")
           for j in range(20):
               if "  " in x:
                    x=x.replace("  ","")
           c.append(x)
        for k in range(0,len(c)):
             if c[k] == "":
               c[k]=None

        return c
     clean_avis=clean_Avis(avis)
```

Nettoyage des catégories

```
[61] def clean_genre(a):
        c=[]
        for i in a:
           x=i.replace("\n","")
           for j in range(20):
               if "  " in x:
                    x=x.replace("  ","")
           c.append(x)
        for k in range(0,len(c)):
             if c[k] == "":
               c[k]=None

        return c
     clean_genre=clean_genre(genre)
```

nettoyage de rating

```
[74] def clean_rating(a):
        c=[]
        for i in a:
           x=i.replace("\n","")
           x=x.replace("/10","")
           for j in range(20):
               if "  " in x:
                    x=x.replace("  ","")

           c.append(x)
        for k in range(0,len(c)):
             if c[k] == "":
               c[k]=None

        return c
     clean_rating=clean_rating(rating)
```

Les fonctions de recherche :

Chercher un film par son titre :

La première fonction **grouper_par_score()** va regrouper les avis du film selon le score par ordre décroissant.

La deuxieme fonction **rechercher_par_titre()** il verifie si le titre existe et affiche le nom du film , son genre ; sa description et les avis groupés par score .

```
[163] def grouper_par_score(t):
        d={"Titre":titre_r,"Avis":clean_avis,"Score":clean_rating}
        a= pd.DataFrame(d)
        a= a[a["Titre"] == t]
        a=a.dropna(axis=0)
        a=a.sort_values(by=['Score'])
        return a
```

```
#retourner les infos sur un film
def rechercher_par_titre(t):
  for i in titre:
    if i == t:
      print("Titre :"+t)
      print("Genre :"+clean_genre[titre.index(i)])
      print("Description :"+clean_description[titre.index(i)])
      print("Avis :")
      print(grouper_par_score(t))
```

Résultat :

```
167] rechercher_par_titre("The Godfather")

    Titre :The Godfather
    Genre :Crime, Drama
    Description :
        An organized crime dynasty's aging patriarch transfers control of his clandestine empire to his re
    Avis :
              Titre                                              Avis    Score
    74  The Godfather                          A Legitimately Perfect Film       10
    69  The Godfather                                       Simply the best       10
    68  The Godfather   An exquisite Mafia epic with outstanding perf...       10
    67  The Godfather                                       The godfather       10
    66  The Godfather                        How things were done back then!       10
    65  The Godfather                      The world inside the underworld!       10
    64  The Godfather                                               perfect       10
    63  The Godfather                                       A Masterpiece       10
    61  The Godfather   An Epic, Masterful Look into the Underground ...       10
    72  The Godfather                         The Pioneer of All Filmmaking       10
    60  The Godfather                    This Movie Has Haunted My Life...       10
    58  The Godfather                                               Top 5       10
    57  The Godfather                       Another kind of "family movie"       10
    56  The Godfather                                       Amazing movie       10
    55  The Godfather                                      An Iconic Film       10
    54  The Godfather   "The Godfather" is pretty much flawless, and ...       10
    53  The Godfather                 An offer so good, I couldn't refuse       10
    52  The Godfather                     The greatest movie of all time!       10
    51  The Godfather                  For Me, This Is The Definitive Film       10
    59  The Godfather   Initially, I wasn't a fan... but then I realised       10
    71  The Godfather                                       The Godfather       10
    73  The Godfather   A film of great power and a milestone in the ...        8
    62  The Godfather            Magnificent portrait of organized crime        9
    70  The Godfather                          The Greatest Movie Ever Made        9
    50  The Godfather                        The Pinnacle Of Flawless Films!   CalPhys
```

Analyse de sentiment des avis d'un film :

TextBlob est une excellente bibliothèque open-source pour effectuer des tâches NLP.

C'est un lexique des sentiments (sous la forme d'un fichier XML) qu'il exploite pour donner à la fois des scores de polarité et de subjectivité.

La sortie de **TextBlob** pour une tâche de polarité est un flottant dans la plage [-1,0, 1,0] où -1,0 est une polarité négative et 1,0 est positive. Ce score peut également être égal à 0, ce qui correspond à une évaluation neutre d'une instruction car elle ne contient aucun mot de l'ensemble d'apprentissage.

```
67]  def analyser_avis():
       from textblob import TextBlob
       from textblob import Blobber
       from textblob_fr import PatternTagger, PatternAnalyzer
       tb = Blobber(pos_tagger=PatternTagger(), analyzer=PatternAnalyzer())
       res=[]
       for i in avis:
           text = tb(u""+i)
           res.append(text.sentiment[0])
       ress=[]
       for i in range(0,len(res)):
           res[i]=round(res[i],2)
           if res[i] > 0:
             ress.append(str("Positive de "+ str(int(res[i]*100))+ "%"))
           elif res[i] == 0:
             ress.append(str("Neutre"))
           else :
             ress.append(str("Négative de "+ str(int(res[i]*100*-1))+ "%"))
       return [res,ress]

     #Grouper les avis (une fct qui retourne les avis d'un film donné ordonné par analyse de sens)
     def grouper_par_avis(t):
       av=analyser_avis()
       d={"Titre":titre_r , "Avis":clean_avis , "Sentiment": av[1] , "ordre": av[0]}
       a= pd.DataFrame(d)
       a= a[a["Titre"] == t]
       a=a.dropna(axis=0)
       a=a.sort_values(by=["ordre"], ascending=False)
       a=a.drop(["ordre"], axis='columns')
       return a

     #Recherche par titre (une fct qui retourne toute les infos d'un film donné)
     def recherche_par_titre_AS(t):
       for i in titre:
         if i == t:
           print("Titre :"+t)
           print("Genre :"+clean_genre[titre.index(i)])
           print("Description :"+clean_description[titre.index(i)])
           print("Avis :")
```

Après le calcul de la polarité des avis la fonction **gouper_par_avis()** regroupe ces avis selon le score de polarite en ordre décroissant **:**

Résultat :

```
[68] recherche_par_titre_AS("Inception")
```

```
Titre :Inception
Genre :Action, Adventure, Sci-Fi
Description :a thief steal corporate secret use technology given inverse task
Avis :
         Titre  ...        Sentiment
233  Inception  ...  Positive de 90%
225  Inception  ...           Neutre
238  Inception  ...           Neutre
248  Inception  ...           Neutre
247  Inception  ...           Neutre
245  Inception  ...           Neutre
244  Inception  ...           Neutre
243  Inception  ...           Neutre
242  Inception  ...           Neutre
241  Inception  ...           Neutre
240  Inception  ...           Neutre
239  Inception  ...           Neutre
237  Inception  ...           Neutre
226  Inception  ...           Neutre
236  Inception  ...           Neutre
235  Inception  ...           Neutre
234  Inception  ...           Neutre
232  Inception  ...           Neutre
231  Inception  ...           Neutre
230  Inception  ...           Neutre
229  Inception  ...           Neutre
228  Inception  ...           Neutre
227  Inception  ...           Neutre
249  Inception  ...           Neutre
246  Inception  ...  Négative de 40%

[25 rows x 3 columns]
```

Chercher le film le plus similaire à une description :

TfidfVectorizer : Transforme le texte en vecteurs de caractéristiques qui peuvent être utilisés comme entrée pour l'estimateur.

Cosine_similarity() : pour calculer la similarité entre la requête entrée et les descriptions du corpus.

En fin on affiche le titre du filme dont la description est plus similaire à notre requête.

```
[69] #Le film le plus similaire a une description donnée si la requete entrante e
     def sim_desc(query):
       query=preprocessing(query)
       import numpy as np
       tfidf_vectorizer = TfidfVectorizer()
       tfidf_desc = tfidf_vectorizer.fit_transform(clean_description)
       q = tfidf_vectorizer.transform([query])
       cs = cosine_similarity(q, tfidf_desc)
       ind=np.argmax(cs[0])
       print("Titre  : "+titre[ind])
       print("Description  : "+clean_description[ind])

     sim_desc("love")
```

Résultat :

```
sim_desc("love")
```

```
Titre  : Amélie
Description  : amélie innocent naive girl paris sense justice she decides help around along way discovers love
```

Chercher les 10 films les plus similaires à une description :

La même démarche de la fonction précédente sauf qu'on a pris les 10 premiers résultats , pour cela on a utilisé la méthode **argmax()** une fonction de module **numpy** . Cette fonction renvoie les indices des valeurs maximales renvoyées avec l'axe spécifié.

```
#tf_idf
def tf_idf(query, description):
    query=preprocessing(query)
    desc_clean=[]
    for i in description:
       desc_clean.append(preprocessing(i))
    import numpy as np
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_desc = tfidf_vectorizer.fit_transform(desc_clean)
    q = tfidf_vectorizer.transform([query])
    cs = cosine_similarity(q, tfidf_desc)
    res= cs[0]
    result_list = [] #index
    sim = [] #similarité
    nb = 10
    while nb > 0:
        index = np.argmax(res)
        result_list.append(index)
        sim.append(res[index])
        res[index] = 0
        nb =nb - 1

    print("les 10 films similaires à votre description sont:")
    for i,j in zip(result_list,sim):
            print("Titre :"+titre[i])
            s=int(j*100)
            print("score de similarité :"+str(s)+"%")
            print("Genre :"+clean_genre[i])
            print("Description :"+desc_clean[i])
```

Résultat :

```
tf_idf("love",description)
```

```
les 10 films similaires à votre description sont:
Titre :Amélie
score de similarité :21%
Genre :Comedy, Romance
Description :amélie innocent naive girl paris sense justice she decides help around along
Titre :City Lights
score de similarité :20%
Genre :Comedy, Drama, Romance
Description :with aid wealthy erratic tippler tramp fallen love sightless flower girl accu
Titre :Cinema Paradiso
score de similarité :20%
Genre :Drama, Romance
Description :a filmmaker recall childhood falling love picture cinema home village form de
Titre :7 Kogustaki Mucize
score de similarité :18%
Genre :Drama
Description :a story love father wrongly accused murder lovely six year old daughter the p
Titre :The Hunt
score de similarité :16%
Genre :Drama
Description :a teacher life lonely life struggling son custody his life slowly get better
Titre :Dara of Jasenovac
score de similarité :0%
Genre :Drama, War
Description :the film set croatian ustasha regime ndh former yugoslavia wwii the film told
Titre :Dara of Jasenovac
score de similarité :0%
Genre :Drama, War
Description :the film set croatian ustasha regime ndh former yugoslavia wwii the film told
Titre :Dara of Jasenovac
score de similarité :0%
Genre :Drama, War
```

Chercher les 10 films les plus similaire à une catégorie :

```
.00] #tf_idf
    def tf_idf(query, genre):
        query=preprocessing(query)
        clean_genre=[]
        for i in genre:
           clean_genre.append(preprocessing(i))
        import numpy as np
        tfidf_vectorizer = TfidfVectorizer()
        tfidf_genre = tfidf_vectorizer.fit_transform(clean_genre)
        q = tfidf_vectorizer.transform([query])
        cs = cosine_similarity(q, tfidf_genre)
        res= cs[0]
        result_list = [] #index
        sim = [] #similarité
        nb = 10
        while nb > 0:
            index = np.argmax(res)
            result_list.append(index)
            sim.append(res[index])
            res[index] = 0
            nb =nb - 1

        print("les 10 films similaires à votre categorie sont:")
        for i,j in zip(result_list,sim):
                print("Titre :"+titre[i])
                s=int(j*100)
                print("score de similarité :"+str(s)+"%")
                print("Genre :"+clean_genre[i])
```

Résultat :

```
tf_idf('War',genre)
```

```
les 10 films similaires à votre categorie sont:
Titre :Dara of Jasenovac
score de similarité :93%
Genre :drama war
Titre :Saving Private Ryan
score de similarité :93%
Genre :drama war
Titre :Paths of Glory
score de similarité :93%
Genre :drama war
Titre :Inglourious Basterds
score de similarité :75%
Genre :adventure drama war
Titre :Apocalypse Now
score de similarité :70%
Genre :drama mystery war
Titre :Incendies
score de similarité :70%
Genre :drama mystery war
Titre :The Great Dictator
score de similarité :68%
Genre :comedy drama war
Titre :1917
score de similarité :68%
Genre :drama thriller war
Titre :Grave of the Fireflies
score de similarité :66%
Genre :animation drama war
Titre :Casablanca
score de similarité :65%
Genre :drama romance war
```