Projet 3: Développez un moteur de recommandations de films

Réalisé par: Taher Haggui

Problématique :

Elaborer une API capable de retourner 5 films similaires et intéressants pour les visiteurs de grand prince.

Comment ?

1.Recenser une base de données de milliers de produit.

2.Nettoyage et exploration des données .

3.Modélisation des données .

4.devellopement d’une API avec le framework flask .

Résultats attendues ?

Un meilleur partitionnement des films.

Une API, permettant de recommander 5 films similaires au choix préalable de visiteur.

Nettoyage des données :

Data Existing state

Statistiques sur les différentes colonnes de la base de données à l'état initial.

```
# Data columns statistics
data.describe()
```

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users	cast_tot
count	4993.000000	5028.000000	4939.000000	5020.000000	5036.000000	4.159000e+03	5.043000e+03	
mean	140.194272	107.201074	686.509212	645.009761	6560.047061	4.846841e+07	8.366816e+04	
std	121.601675	25.197441	2813.328607	1665.041728	15020.759120	6.845299e+07	1.384853e+05	
min	1.000000	7.000000	0.000000	0.000000	0.000000	1.620000e+02	5.000000e+00	
25%	50.000000	93.000000	7.000000	133.000000	614.000000	5.340988e+06	8.593500e+03	
50%	110.000000	103.000000	49.000000	371.500000	988.000000	2.551750e+07	3.435900e+04	
75%	195.000000	118.000000	194.500000	636.000000	11000.000000	6.230944e+07	9.630900e+04	
max	813.000000	511.000000	23000.000000	23000.000000	640000.000000	7.605058e+08	1.689764e+06	

Nettoyage des données : Colonnes inutiles

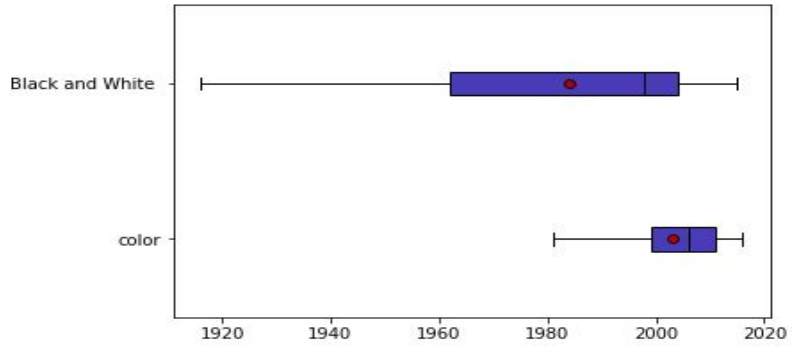
Useless Columns

The goal of this project is to provide an API,which can suggest 5 films similar to the film choiced by a customer.To achieve this goal, we find that we can get red of the following useless columns:
-director_facebook_likes
-actor_3_facebook_likes
-actor_1_facebook_likes
-cast_total_facebook_likes -budget
-actor_2_facebook_likes
-movie_facebook_likes
-facenumber_in_poster

:
```
del data["director_facebook_likes"]
del data["actor_3_facebook_likes"]
del data["actor_1_facebook_likes"]
del data["cast_total_facebook_likes"]
del data["budget"]
del data["actor_2_facebook_likes"]
del data["movie_facebook_likes"]
del data["facenumber_in_poster"]
```

Nettoyage des données : Valeurs manquantes

Color missing Values



So the most likely color for a movie created after 1980, is Color.Above all the 95% of movie mentionned in our database have value of Color in the column namely color.

Remplacer les valeurs manquantes par Color si la date de sortie de film est aprés 1980, par Black and White si non.

```
In [13]: data["color"]=[fill_color(c[0],c[1]) for c in zip(data['color'],data['title_year'])]# imputing missed value
```

Nettoyage des données : Valeurs manquantes

num_critic_for_reviews / num_user_for_reviews missing Values

Choix : Définir un coefficient de proportionnalité entre ces deux variables.

```
: # Thisfunction allow to remplace missing values in column of num_critic_for_reviews and num_user_for_reviews.
def fill_reviews(cr_reviews,us_reviews):
    """ This function examinate if the value of the introduced arguments is null. the function uses the coefficient
        that we had estimated between the two variables 'num_user_for_reviews' and 'num_critic_for_reviews' to rempl
        the value of one argument when the other is not null. In the case of the two arguments are null, the functio
        begin by give the argument us_reviews the mean of the variable 'num_user_for_reviews', then the function rem
        the first argument as defined above.

        Args:

        cr_reviews(int): The argument that the function examinated and remplace it when it is null.
        us_reviews(int): The argument that the function examinated and remplace it when it is null.

        Returns:

        (cr_reviews,us_reviews): the function return a not null tuple.
        """
    if pd.isnull(cr_reviews):
        if pd.isnull(us_reviews):
            us_reviews=data['num_user_for_reviews'].mean()
        return(coef*us_reviews,us_reviews)


    elif pd.isnull(us_reviews):
        return (cr_reviews,(1/coef)*cr_reviews)
    else:
        return(cr_reviews,us_reviews)
```

```
: # imputation of misssing values in num_critic_for_reviews.
data['num_critic_for_reviews']=[fill_reviews(a[0],a[1])[0] for a in zip(data['num_critic_for_reviews'],data['num_use
```

Nettoyage des données : Valeurs manquantes

gross missing Values

```
In [26]: gross_corr #display the table of correlation
```

Out[26]:

	num_critic_for_reviews	duration	gross	num_user_for_reviews	title_year	imdb_score	aspect_ratio
gross	0.480599	0.250298	1.0	0.559941	0.030886	0.198021	0.069346

Absence d'un modèle ,permettant de remplacer efficacement les valeurs manquantes.

Choix : Supprimer cette variable.

Others missing Values

Choix: Supprimer toutes les lignes contenant des valeurs manquantes.

```
In [32]: data=data.dropna()
```

Préparation des données

Action 1: Organiser les colonnes par type. Mettre les variables quantitatives au début, puis les variables qualitatives.

```
li_q=[] # names's list of quantitatives variables.
li_r=[] # names's list of rest of variables.
for c in data.columns:
    if data[c].dtype == float or data[c].dtype==int:
        li_q.append(c)
    else:
        li_r.append(c)
```

```
li=li_q+li_r
data=data[li]
data.head()
```

Action 2: Supprimer les lignes dupliquées.

Duplicated rows

```
In [35]: len(data) # length of our data
Out[35]: 4429

In [36]: data=data.drop_duplicates()

In [37]: len(data) # length of data after deleting duplicated rows
Out[37]: 4389
```

Valeurs aberrantes:

- **Choix 1: La durée d'un film ne dépasse pas généralement 2 heures.Donc, nous avons considérés les durées supérieur à (Q3+1.5IQR=160min) comme valeurs aberrantes.**
- **Choix 2: La durée d'un film ne peut pas être une valeur nulle.Donc, nous allons considérer les durées nulles comme valeurs aberrantes.**
- **Choix 3: Nous supposons que ce jeu de données a été collecté depuis au moins un an. Donc , nous avons considéré les films dont la variable 'title_year' supérieur à 2017 comme valeurs aberrantes.**

-We know that the duration of a film can not be more than two hours at maximum. So we will consider that values of films's duration superior than (Q3+1.5IQR) are outliers.
-The duration of film can not be zero .So we will consider all films which have zero duration as outliers.
-We suppose that this database are collected for more than year. So will consider all films who have title year above than 2017 as outliers.

```
data=data[(data['duration']>0) & (data['duration']<(data['duration'].quantile(0.75)+1.5*(data['duration'].quantile(0
```

```
data=data[data['title_year']<2017]
```

Résultats de Nettoyage de données:

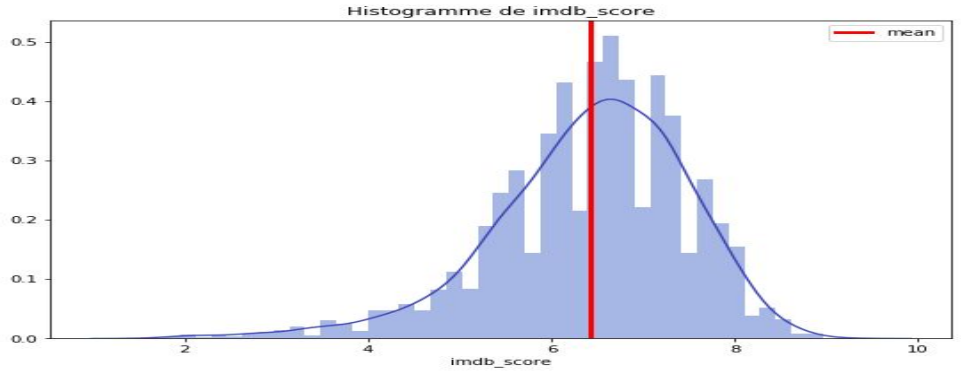
Avant

	number
rows	5043
columns	28
not_missing_values	138506

Après

	number
rows	4239
columns	19
not_missing_values	80541

Analyse univariée: Variables quantitatives




```
: from scipy.stats import ks_2samp
  ks_2samp(data['imdb_score'],list(np.random.normal(data['imdb_score'].mean(),data['imdb_score'].std(),len(data))))
```

```
: Ks_2sampResult(statistic=0.06369426751592355, pvalue=6.201662305114717e-08)
```

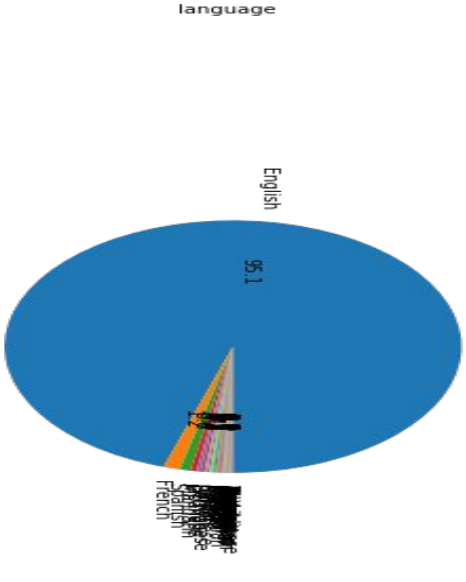
```
: print(data['imdb_score'].mean())
  print(data['imdb_score'].std())
```

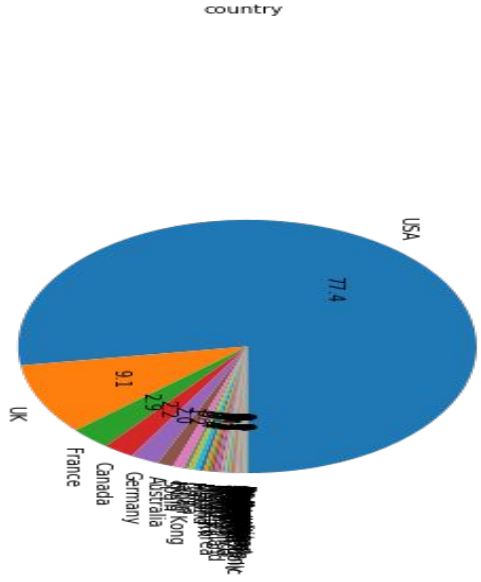
```
6.423543288511426
1.0668113474792835
```

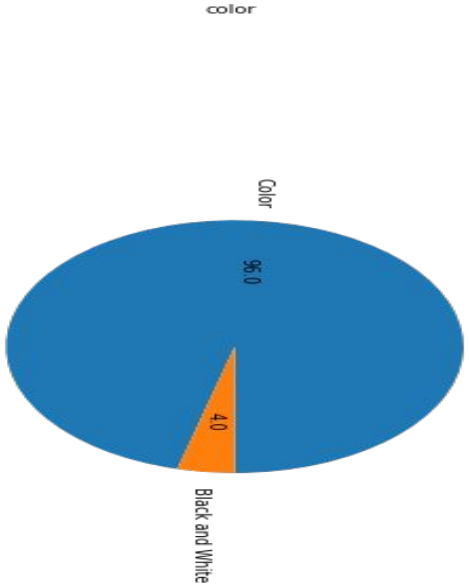
So we can accept that imdb_score is a gaussian variable with a about 50% of risk to reject this hypothesis even when it is true.

imdb_score ~ N(6.42,1.06)

Analyse univariée: Variables qualitatives







Analyse multivariée

In order to examinate the possible relations between variables. We will calculate the table of correlation between them.

```
data[['num_critic_for_reviews','duration','num_voted_users','title_year','imdb_score','aspect_ratio']].corr()
```

	num_critic_for_reviews	duration	num_voted_users	title_year	imdb_score	aspect_ratio
num_critic_for_reviews	1.000000	0.260038	0.607177	0.364459	0.330274	0.105293
duration	0.260038	1.000000	0.319825	-0.049988	0.351430	0.113714
num_voted_users	0.607177	0.319825	1.000000	0.047982	0.437528	0.050117
title_year	0.364459	-0.049988	0.047982	1.000000	-0.179491	0.205438
imdb_score	0.330274	0.351430	0.437528	-0.179491	1.000000	-0.010441
aspect_ratio	0.105293	0.113714	0.050117	0.205438	-0.010441	1.000000

So we can conclude from the table above that there is a high correlation of num_voted_users with Simultaneously num_critic_for_reviews and imdb_score.

Variables catégorielles

categorical variables

As it had showed in the univariate analyse, more than 95% of films are in english and about 96% of films are colored films. So we will create a new variables correspondent to three most used languages and new variables for each color films.

```
li_color=list(data['color'].unique()) # list of used color
li_lang=list(data['language'].value_counts().index)[:3] # list of used language
```

```
#create new variables correspond to each color.
for c in li_color:
    data[c]=[1 if c==m else 0 for m in data['color']]
del(data['color']) #delete the variable color
```

```
#create  new variables correpond to each language.
for c in li_lang:
    data[c]=[1 if c==m else 0 for m in data['language']]
del(data['language']) # delete the variable language.
```

Features engineering

Création des nouvelles variables correspondant aux différents genres, mentionnés dans notre jeu de donnés.

```
genres=[]# list of various genres mentionned in this data.
li=[]
for c in data['genres']:
    li=c.split('|')
    genres+=li
genres=np.array(genres)
genres=np.unique(genres)
genres=list(genres)
```

```
#Create the new variables
for c in genres:
    data[c]=[1 if c in m else 0 for m in data['genres']]
```

```
del(data['genres']) # delete the column "genres" which will be useless.
```

Création des nouvelles variables correspondant aux 20% des mots clés, les plus utilisés dans notre jeu de donnés.

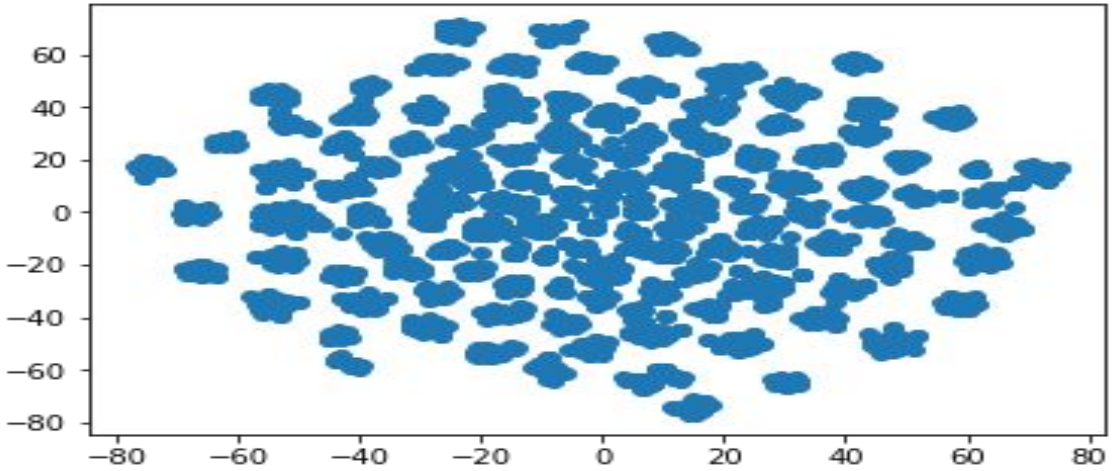
```
li_mot=[]# list of key words
for c in data['plot_keywords']:
    li_mot+=c.split('|')
li_key=pd.DataFrame({"mot_clés":li_mot})
li=li_key['mot_clés'].value_counts(normalize=True).cumsum()
i=0
for c in li :
    if c < 0.2:
        i+=1
    else:
        break
mot_retenue=list(li.index)[0:i] # retained words
```

```
# Create new variables for the retained key words.
for c in mot_retenue:
    data[c]=[1 if c in m else 0 for m in data['plot_keywords']]
```

Réduction dimensionnelle

Objectif: Rechercher les similarités entre les films pour pouvoir recommander 5 films similaires pour le visiteur.

- Nous avons intérêt à utiliser une technique, permettant de garder la structure locale.
- La technique TSNE.



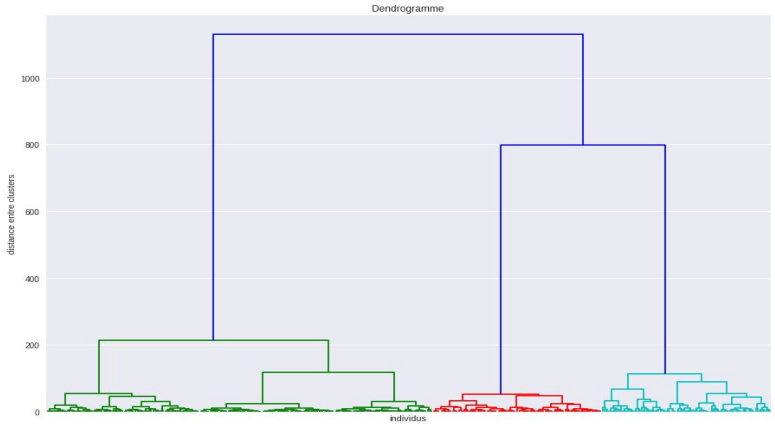
Clustering

Objectif: Recommander 5 films pour chaque utilisateur.

Contraintes:

- Les méthodes non supervisés standards , optimisent des critères tels que l'homogénéité et la distance inters clusters.
- Ils débouchent sur des clusters de taille différents .

Définir une fonction:



Clustering

KMeans method:

```
from sklearn.cluster import KMeans
```

```
label=clustering(KMeans(n_clusters=2),df)
```

DBSCAN method:

```
from sklearn.cluster import DBSCAN
```

```
dbscan=DBSCAN(min_samples=5)
dbscan.fit(df)
lb=dbscan.labels_
```

AgglomerativeClustering method:

```
from sklearn.cluster import AgglomerativeClustering
```

```
label_agg=clustering(AgglomerativeClustering(n_clusters=2),df)
```

Clustering: évaluation de performances

Silhouette Score:

```
score_KMeans=silhouette_score(df,label) # computing silhouette score
score_KMeans
```

```
0.39354599672346385
```

```
score_DB=silhouette_score(df,lb) # computing silhouette_score
score_DB
```

```
-0.6355881273578893
```

```
score_agg=silhouette_score(df,label_agg) # computing silhouette_score
score_agg
```

```
0.38206823789715133
```

Nbre clusters contiennent moins de 6 points:

```
#compute the number of clusters which contains less than 6 points.
t_clusters=[] #Size for the differents obtained clusters.
for lb in np.unique(label_agg):
    t_clusters.append(len(np.where(label_agg==lb)[0]))
t_clusters=np.array(t_clusters)
cl_AG_6=len(np.where(t_clusters<=6)[0])
cl_AG_6
```

Clustering: évaluation de performances

Silhouette Score:

```
score_KMeans=silhouette_score(df,label) # computing silhouette score
score_KMeans
```

```
0.39354599672346385
```

```
score_DB=silhouette_score(df,lb) # computing silhouette_score
score_DB
```

```
-0.6355881273578893
```

```
score_agg=silhouette_score(df,label_agg) # computing silhouette_score
score_agg
```

```
0.38206823789715133
```

Nbre clusters contiennent moins de 6 points:

```
#compute the number of clusters which contains less than 6 points.
t_clusters=[] #Size for the differents obtained clusters.
for lb in np.unique(label_agg):
    t_clusters.append(len(np.where(label_agg==lb)[0]))
t_clusters=np.array(t_clusters)
cl_AG_6=len(np.where(t_clusters<=6)[0])
cl_AG_6
```

Clustering: évaluation de performances

Score empirique de précision:

```
# We will choice 5 films randomly.
film_alea_choisies=np.random.choice(range(100),5)
```

```
#We will define a method, which allow to recommend 5 films for an choiced film using a defined method.
def recommander_5films(clusters,film):
    """This method allow to recommend 5 films for the introduced film. It uses the data set index of the introduced
       film , then it recuperate the label cluster of this film according to the labels of the used method.
       The function, retrieve all films which have same label cluster with the introduced film.The the function
       return after the 5 first films for them.

        Args:
        clusters(array): label of each film according to the used method.
        film(int): the index of the introduced film in the data set

        """
    cluster_film=clusters[film]
    indices=np.where(clusters==cluster_film)
    indices=indices[0]
    i=0
    for j in indices:
        if j==film:
            break
        i+=1
    indices=np.delete(indices,i)

    indices=indices[0:5]
    recommandations=dff.iloc[indices]
    return recommandations
```

KMean accuracy note

```
KMean_note=np.mean(np.array([pertinence_recommandationKM1,pertinence_recommandationKM2,pertinence_recommandationKM3,
                             pertinence_recommandationKM4,pertinence_recommandationKM5]))
```

AgglomerativeClustering note

```
Agg_note=np.mean(np.array([pertinence_recommandationAgg1,pertinence_recommandationAgg2,pertinence_recommandationAgg3
                           ,pertinence_recommandationAgg4,pertinence_recommandationAgg5]))
```

Clustering: choix du modèle

Selected method:

```
#The performance table of each tested method.
table=pd.DataFrame({'KMeans':[score_KMeans,cl_KM_6,KMean_note],'DBSCAN':[score_DB,'-','-'],'Agglomerative':[score_ag
                   ,index=['silhouette_score','clusters_6','Empirical accuracy note'])
table
```

	KMeans	DBSCAN	Agglomerative
silhouette_score	0.393546	-0.635588	0.382068
clusters_6	1197.000000	-	678.000000
Empirical accuracy note	0.720000	-	0.720000

We find that KMeans and AgglomerativeClustering, give practicaly the same silhoutte score, but the AgglomerativeClustering provide less number of clusters with less than 6 points. So we will use this method as a method clustering of our dataset.

API

id & cluster of each film:

```
data['id']=[c[26:35] for c in data['movie_imdb_link']] # the id of each film
```

```
data['cluster']=label_agg # the cluster of each film on base of AgglomerativeClustering method
```

```
#! usr/bin/python3.6
from flask import Flask,request
import pandas as pd
import os
chemin=os.path.abspath(os.path.dirname(__file__))
fichier=os.path.join(chemin,'data_final.csv')
data=pd.read_csv(fichier)


app=Flask(__name__)

@app.route('/')
@app.route('/recommend/')

def recommend():
   id=request.args.get('id')

   if id is not None:
       if id not in data['id'].values:
          return "This film not available"
       else:

          li=[]
          cluster=data[data['id']==id]['cluster'].iloc[0]
          df=data[data['cluster']==cluster][['id','movie_title']]
          i=0
          for uid,tf in zip (df['id'],df['movie_title']):
              if uid!=id:
                  li.append({"id":uid,"name":tf})
                  i+=1
              if i==5:
                  break
          return str({"_results":li})
   else:
       return " id not found "

if __name__=="__main__":
```

Mise en ligne

- **Heroku**

https://moteur-de-recherche.herokuapp.com/recommend/?id=tt0401729

Conclusion

- **En se basant sur les genres de film, des mots clés de recherche, de la langue et la colère de film. Nous avons pu définir 965 regroupements différents de film .**

- **A l'aide de heroku, le framework flask ainsi que le clustering obtenu. Nous avons pu développer une API permettant de recommander 5 films similaires aux utilisateurs correspond à leur choix initial. dont ci-dessous son lien:**

  https://moteur-de-recherche.herokuapp.com/