Analyse de tableaux de données avec pandas

1 Introduction

1.1 Objectif

L'objectif de cette séance est de se familiariser avec la manipulation et le traitement de (potentiellement gros) tableaux de données en python, pour en ressortir une information intéressante ou recherchée. Par tableau on entend "une liste d'éléments possédant tous les mêmes attributs", à l'instar de ce que l'on peut trouver dans un classeur Excel ou une table de base de données SQL.

Pour cet exercice, nous utiliserons le package python pandas en chargeant des tableaux depuis des fichiers csv. A noter que pandas peut également charger des tableaux de données depuis des bases de données SQL (non abordé ici).

1.2 Présentation et récupération des données

Les données que nous allons manipuler pour cet exercice est un ensemble de fichiers csv (ou plus précisément "tsv" car le séparateur est le caractère "tab" et non pas une virgule) mis à disposition gratuitement par IMDB.com. Une description rapide par fichier est disponible ici.

Créez un dossier IMDB. Recuperer les archives title.ratings, title.basics, title.principals et name.basics. Décompressez-les dans le dossier IMDB. Celles-ci contiennent chacune un fichier nommé data.tsv dans un dossier portant le nom de l'archive.

```
clement@clement-HP-250-G6:~/Documents/SupOp/IMDB$ tree
.
├── name.basics.tsv
│   └── data.tsv
├── name.basics.tsv.gz
├── title.basics.tsv
│   └── data.tsv
├── title.basics.tsv.gz
├── title.principals.tsv
│   └── data.tsv
├── title.principals.tsv.gz
├── title.ratings.tsv
│   └── data.tsv
└── title.ratings.tsv.gz
```

Ouvrez votre IDE favori, placez vous dans le dossier IMDB et créez un notebook ou fichier python dans lequel vous travaillerez.

Vous aurez besoin d'au moins 5 Go de RAM disponible.

2 Chargement avec pandas

2.1 Chargement d'un csv simple (10 min)

Commençons par charger un premier csv dans un DataFrame pandas grâce à sa fonction read_csv.

```
>>> ratings = pandas.read_csv("title.ratings.tsv/data.tsv")
```

Affichez un aperçu de ce DataFrame :

```
>>> ratings
        tconst\taverageRating\tnumVotes
0                   tt0000001\t5.7\t1988
1                    tt0000002\t5.8\t265
2                   tt0000003\t6.5\t1849
3                    tt0000004\t5.5\t178
4                   tt0000005\t6.2\t2632
...                                  ...
1331788               tt9916730\t8.3\t10
1331789               tt9916766\t7.0\t21
1331790               tt9916778\t7.2\t36
1331791                tt9916840\t7.5\t7
1331792                tt9916880\t7.0\t7
[1331793 rows x 1 columns]
```

Vous constaterez que pandas n'a pas interprété les tabs présents dans le fichier comme des séparateurs de colonne.

Q1) En vous aidant de la documentation de la fonction read_csv, ajoutez l'option adéquate afin de charger correctement le csv et ainsi obtenir cet aperçu :

```
>>> ratings
             tconst  averageRating  numVotes
0         tt0000001            5.7      1988
1         tt0000002            5.8       265
2         tt0000003            6.5      1849
3         tt0000004            5.5       178
4         tt0000005            6.2      2632
...             ...            ...       ...
1331788   tt9916730            8.3        10
1331789   tt9916766            7.0        21
1331790   tt9916778            7.2        36
1331791   tt9916840            7.5         7
1331792   tt9916880            7.0         7
[1331793 rows x 3 columns]
```

Vous pouvez constater que pandas a automatiquement interprété le type de chaque colonne (float, int, …) en affichant l'attribut dtypes du DataFrame (par défaut, pandas stock les chaines de caracteres dans un champs de type object) :

```
>>> ratings.dtypes
tconst            object
averageRating    float64
numVotes           int64
```

2.2 Chargement d'un csv plus complexe (20 min)

Chargez maintenant le csv "title.basics.tsv/data.tsv" dans un DataFrame nommé basics, de la même manière que pour "title.ratings.tsv/data.tsv".

Lors du chargement, pandas émet l'avertissement suivant :

```
<stdin>:1: DtypeWarning: Columns (4) have mixed types. Specify dtype option on import or
set low_memory=False.
```

Et si vous affichez les types déduits par pandas pour chaque colonne, vous constaterez qu'en effet, pandas n'a pas réussi à inférer le bon type des colonne isAdult (colonne 4, normalement booléen), startYear, endYear et runtimeMinutes (normalement des int).

```
>>> basics.dtypes
tconst            object
titleType         object
primaryTitle      object
originalTitle     object
isAdult           object
startYear         object
endYear           object
runtimeMinutes    object
genres            object
```

La première raison à cela vient de la manière que IMDB utilise pour indiquer qu'un champ n'est pas renseigné. En effet, comme ils le spécifient sur la page de présentation des données, il utilisent la chaine de caractère "\N", qui ne fait pas partie des symboles par défaut que pandas interprète automatiquement en tant que champs vide (la liste par défaut est renseignée dans la documentation de l'option na_values de la fonction read_csv).

Q2) En utilisant l'option na_values de read_csv, rechargez le csv basics.

L'avertissement doit maintenant être :

```
<stdin>:1: DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
```

Remarquez que le problème a maintenant lieu à la colonne 7 et non plus 4.

Si vous affichez le type inféré par pandas pour chaque colonne, vous constaterez en effet qu'il a réussi sauf pour la colonne 7 (tuntimeMinutes) :

```
>>> basics.dtypes
tconst             object
titleType          object
primaryTitle       object
originalTitle      object
isAdult           float64
startYear         float64
endYear           float64
runtimeMinutes     object
genres             object
```

Pour comprendre le problème, affichons une des lignes pour lesquelles pandas n'a pas su interpréter le champs runtimeMinutes comme un nombre :

```
>>> basics[basics.tconst == "tt10970874"].iloc[0]
tconst                                                tt10970874
titleType                                              tvEpisode
primaryTitle      Die Bauhaus-Stadt Tel Aviv - Vorbild für die M...
originalTitle                                                 0
isAdult                                                  2019.0
startYear                                                   NaN
endYear                                                     NaN
runtimeMinutes                                        Talk-Show
genres                                                      NaN
```

Il semble en fait y avoir un décalage à partir du champ "originalTitle" (originalTitle contient la valeur qui devrait être dans isAdult, qui lui-même contient la valeur qui devrait être dans le champ

suivant, etc ...) . Si vous affichez dans un terminal cette ligne, vous constaterez en effet une particularité :

```
>>> grep -rine 'Bauhaus-Stadt' title.basics.tsv/data.tsv
1507381:tt10970874  tvEpisode    "Die Bauhaus-Stadt Tel Aviv - Vorbild für die
Metropolen der Moderne?    "Die Bauhaus-Stadt Tel Aviv - Vorbild für die Metropolen der
Moderne?     0     2019   \N     \N     Talk-Show
```

Les valeurs des colonnes primaryTitle et originalTitle commencent toutes les deux par un guillemet qui n'est pas refermé. Pandas par défaut considère tous les caractères (y compris un délimiteur de champs) compris entre deux guillemets comme un unique champ. Par conséquent, il a fusionné les deux champs en un seul.

Q3) Pour remédier à cela, recharger le csv en utilisant l'option `quoting=csv.QUOTE_NONE` de la fonction read_csv, qui permet de ne donner aucun sens particulier au caractère guillemet. Note : comme le DataFrame "basics" est très gros, mieux vaut démarrer un nouvel interpréteur python pour libérer la RAM utilisée par basics.

3 Accès aux éléments d'un DataFrame (10 min)

3.1 df.loc[] , df.iloc[] , df[]

Les DataFrame possèdent deux dimensions. L'attribut "loc" d'un DataFrame permet d'adresser la première (les lignes) via un nom d'index et la seconde (les colonnes) via un nom de colonne.

Vous pouvez afficher la liste des index et des colonnes via les attributs "index" et "colonne" d'un DataFrame :

```
>>> ratings.index
RangeIndex(start=0, stop=1331793, step=1)
>>> ratings.columns
Index(['tconst', 'averageRating', 'numVotes'], dtype='object')

>>> basics.loc[42, "startYear"]
1896
```

Il est possible d'accéder à plusieurs colonnes pour un index donné, ou plusieurs index pour une colonne donné, auquel cas pandas retournera une Series :

```
>>> basics.loc[42, ["titleType", "startYear"]]
titleType    short
startYear     1896

>>> basics.loc[[42, 2023], "titleType"]
42      short
2023    short
```

Il est possible d'extraire plusieurs index et colonnes à la fois, auquel cas pandas retournera un DataFrame :

```
>>> basics.loc[[42, 2023], ["titleType", "startYear"]]
     titleType  startYear
42       short       1896
2023     short       1912
```

Il est également possible de faire les mêmes accès à des éléments/lignes/colonnes/sous-DataFrame via les numéros de lignes et colonnes grâce à l'attribut iloc :

```
>>> basics.iloc[[42, 2023], [5,8]]
      startYear     genres
42         1896      Short
2023       1912  Short,War
```

NB : les index sont ici égaux aux numéros de lignes, mais il est possible de construire un DataFrame possédant un index plus complexe. Vous pouvez par exemple essayer d'utiliser loc[] sur le DataFrame ratings2 créé ainsi : ratings2 = ratings.set_index("tconst"). Ici, les index sont des chaînes de caractères !

Vous pouvez utiliser le symbole ":" sur une des dimensions de l'attribut loc ou iloc afin de sélectionner tous les champs d'une dimension et ainsi récupérer une ligne ou colonne entière.

```
>>> ratings.loc[:, "numVotes"]
0          1988
1           265
2          1849
3           178
4          2632
           ...
1331788      10
1331789      21
1331790      36
1331791       7
1331792       7

>>> ratings.loc[5, :]
tconst           tt0000006
averageRating          5.1
numVotes               182
```

Dans le cas des colonnes, il existe une possibilité supplémentaire : l'opérateur [] ou "." directement appliqué au DataFrame :

```
>>> basics.startYear
0          1894
1          1892
2          1892
3          1892
4          1893
           ...
4234319    2021
4234320    2010
4234321    2007
4234322    2001
4234323    <NA>

>>> basics["titleType"]
0              short
1              short
2              short
3              short
4              short
             ...
4234319    tvEpisode
```

```
4234320     tvSeries
4234321    tvEpisode
4234322    tvEpisode
4234323    tvEpisode
```

4 Opérations sur les colonnes (10 min)

4.1 Opérations arithmétiques entre colonnes

```
>>> ratings.numVotes * ratings.averageRating
0          11331.6
1           1537.0
2          12018.5
3            979.0
4          16318.4
            ...
1331788       83.0
1331789      147.0
1331790      259.2
1331791       52.5
1331792       49.0
```

4.2 Statistiques

```
>>> ratings.numVotes.sum()
1383437867
>>> ratings.averageRating.mean()
6.9557419208540665
>>> ratings.averageRating.std()
1.3821287797981858
```

La méthode describe permet d'avoir rapidement des informations sur la distributions des valeurs dans une colonne de valeurs numériques :

```
>>> ratings.describe()
       averageRating      numVotes
count   1.331793e+06  1.331793e+06
mean    6.955742e+00  1.038778e+03
std     1.382129e+00  1.749255e+04
min     1.000000e+00  5.000000e+00
25%     6.200000e+00  1.100000e+01
50%     7.200000e+00  2.600000e+01
75%     7.900000e+00  1.010000e+02
max     1.000000e+01  2.768047e+06
```

4.3 Application de fonctions numpy

```
>>> np.mod(basics.startYear, 10)
0          4.0
1          2.0
2          2.0
3          2.0
4          3.0
          ...
4234319    1.0
```

```
4234320    0.0
4234321    7.0
4234322    1.0
4234323    NaN
```

4.4 Application de fonctions personnalisées

```
>>> basics.startYear.apply(lambda x: x/2 if x%2==0 else 2*x+1)
0           947.0
1           946.0
2           946.0
3           946.0
4          3787.0
            ...
4234319    4043.0
4234320    1005.0
4234321    4015.0
4234322    4003.0
4234323       NaN
```

5 Ajout d'une colonne dans un DataFrame (2 min)

```
>>> basics["runtimeHours"] = basics["runtimeMinutes"] / 60
>>> basics["isOld"] = basics["startYear"] < 1992
>>> basics[["runtimeHours", "isOld"]]
         runtimeHours  isOld
0            0.016667   True
1            0.083333   True
2            0.066667   True
3            0.200000   True
4            0.016667   True
...               ...    ...
4234319           NaN  False
4234320      0.500000  False
4234321           NaN  False
4234322           NaN  False
4234323           NaN  False
```

6 Filtrage / Indexage logique (10 min)

Directement avec l'opérateur DataFrame[]

```
>>> ratings[ratings.numVotes > 1e6]
            tconst  averageRating  numVotes
46297    tt0068646            9.2   1926778
48853    tt0071562            9.0   1310715
50510    tt0073486            8.7   1034883
53325    tt0076759            8.6   1402275
56733    tt0080684            8.7   1330032
...            ...            ...       ...
887903   tt2267998            8.1   1016540
1049760  tt4154756            8.4   1137376
1049762  tt4154796            8.4   1195096
1074961  tt4574334            8.7   1257999
1227582  tt7286456            8.4   1372661
[74 rows x 3 columns]
```

Ou bien avec l'opérateur DataFrame.loc[]

```
>>> basics.loc[basics.startYear > 2020, "runtimeMinutes"]
13082      94.0
39544       6.0
43546       6.0
51461      95.0
53584      26.0
           ...
4234308     NaN
4234311     NaN
4234313     NaN
4234318     NaN
4234319     NaN
```

Q4) Extraire dans un DataFrame “movies” les titres dont l’attribut “titleType” est “movie”.

7 Fusion de DataFrame (15 min)

Il est possible de croiser les informations de deux DataFrame différents. Pour cela, pandas possède entre autres la fonction merge.

Q5) Après avoir étudié la documentation de la fonction merge, créer le DataFrame “rated_movies” qui regroupe les informations des DataFrames “movies” et “ratings” uniquement pour les films possédant une note sur IMDB.

Les cinq premiers éléments de ce DataFrame doivent être ceux-ci :

```
>>> rated_movies.loc[:5, ["averageRating","primaryTitle"]]
   averageRating                   primaryTitle
0            5.3                     Miss Jerry
1            5.3  The Corbett-Fitzsimmons Fight
2            4.1                       Bohemios
3            6.0    The Story of the Kelly Gang
4            4.4                The Prodigal Son
5            4.3              Robbery Under Arms
```

8 Regroupement de lignes (5 min)

Il est possible de regrouper les lignes d’un DataFrame qui possède la même valeur pour une colonne donnée, et d’ensuite extraire des statistiques sur ces groupes.

```
>>> movies.groupby('startYear').tconst.count()
startYear
1894.0     1
1897.0     2
1898.0     7
1899.0     8
1900.0     6
          ..
2025.0    43
2026.0     5
2027.0     3
2029.0     1
2030.0     1

>>> movies.groupby('startYear').runtimeMinutes.mean()
startYear
1894.0     45.0
1897.0    100.0
1898.0      NaN
```

```
1899.0    135.0
1900.0     59.5
          ...
2025.0    110.0
2026.0     94.0
2027.0      NaN
2029.0      NaN
2030.0      NaN
```

9 Tracé de graphes (15 min)

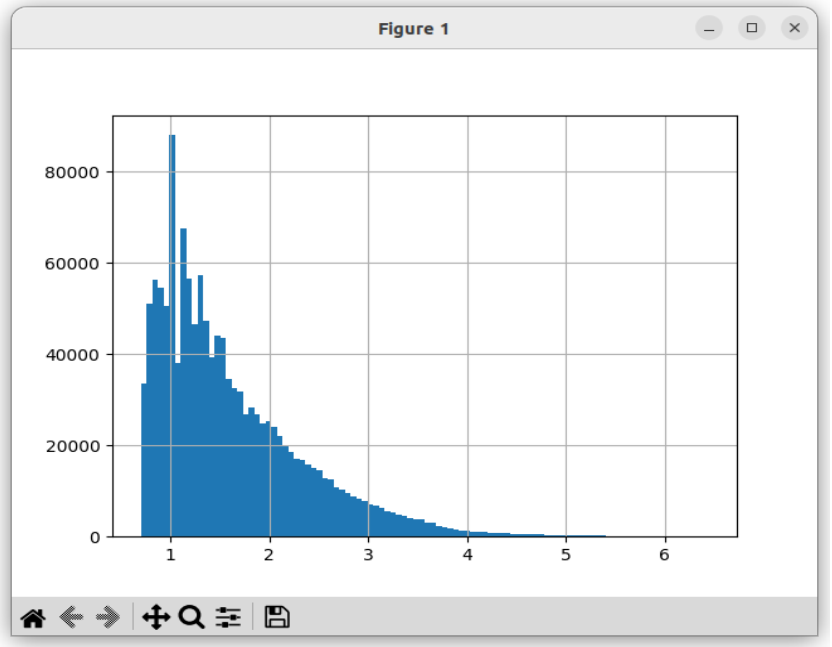
9.1 Histogramme

La méthode hist des DataFrame permet de tracer l'histogramme de toutes les colonnes à valeurs numériques du DataFrame.

```
>>> ratings.hist(bins=100)
array([[<Axes: title={'center': 'averageRating'}>,
        <Axes: title={'center': 'numVotes'}>]], dtype=object)
>>> plt.show()
```

On peut également l'appeler sur une colonne en particulier :

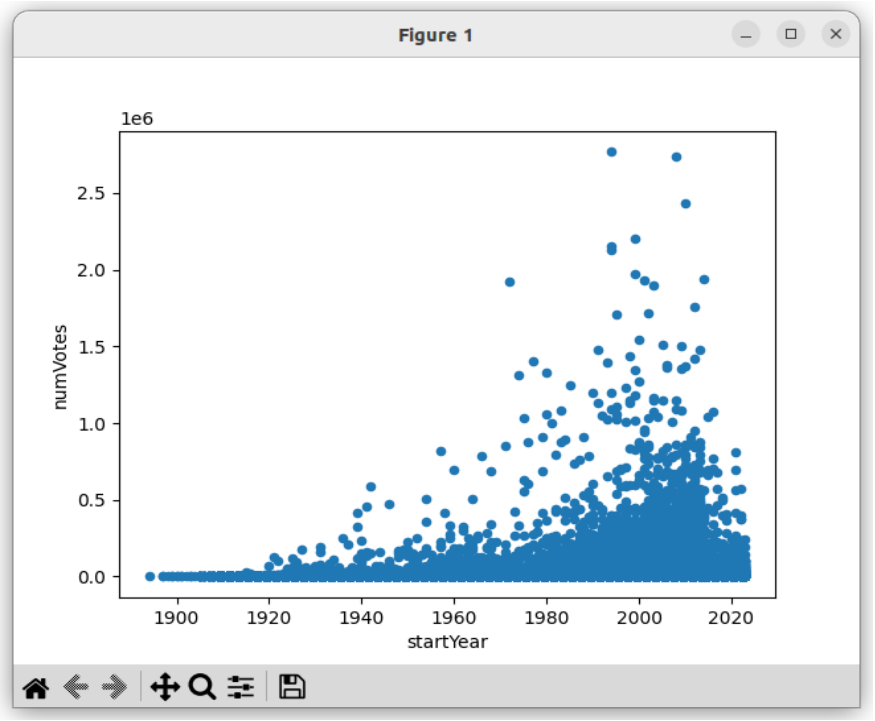
```
>>> np.log10(ratings.numVotes).hist(bins=100)
<Axes: >
>>> plt.show()
```



9.2 Courbes

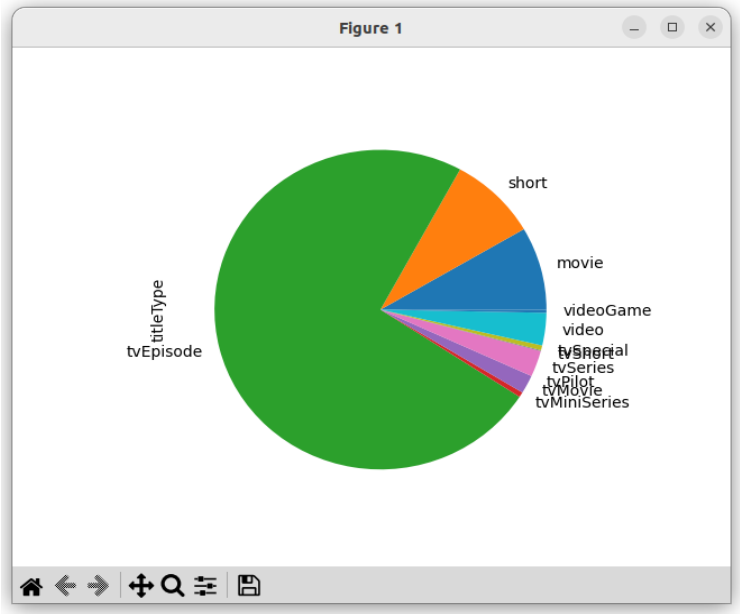
On peut simplement tracer les valeurs d'une colonne en fonction d'une autre colonne grâce à la méthode "plot" de DataFrame. L'option "kind" permet de choisir le type de graphe.

```
>>> rated_movies.plot("startYear", "numVotes", kind="scatter")
<Axes: xlabel='startYear', ylabel='numVotes'>
>>> plt.show()
```



9.3 Pie Charts

```
>>> basics.groupby("titleType").titleType.count().plot.pie()
<Axes: ylabel='titleType'>
>>> plt.show()
```



10 Filtrages de csv volumineux (15 min)

Comme vous pouvez le constater, certains fichiers sont très gros : jusqu'à 2.7 Go pour la table "title.principals.tsv" listant les acteurs principaux de l'ensemble des titres de IMDB. Cela peut rendre long, voire impossible en fonction de la quantité de mémoire vive que possède votre ordinateur, le chargement de ces tables en RAM.

Pour la suite du TD, afin de ne pas avoir à manipuler l'intégralité de la table title.principal.tsv, nous allons la filtrer pour ne garder que les lignes associées à un titre de type “movie”. Nous écrirons le résultat de ce filtrage dans un fichier tsv plus petit “movie.principals.tsv” afin de pouvoir le recharger facilement ultérieurement.

10.1 Filtrage par morceaux

Avec l’option “chunksize” de la fonction “read_csv”, il est possible de charger un gros fichier csv petit bout par petit bout. Ainsi on peut filtrer chaque petit bout en stockant le résultat de ce filtrage partiel dans une liste temporaire avant de finalement concatener tous ces morceaux filtrés en un unique DataFrame :

```
subchunks = []
movies_ids = movies.tconst.values
with pandas.read_csv("IMDB/title.principals.tsv", sep="\t", na_values=["\\N"],
quoting=csv.QUOTE_NONE, chunksize=100000) as reader:
    for k, chunk in enumerate(reader):
        subchunk = chunk[chunk.tconst.isin(movies_ids)]
        subchunks.append(subchunk)
movies_actor_list = pandas.concat(subchunks)
```

10.2 Ecriture de la table dans un fichier

La méthode “to_csv” des DataFrame permet de simplement écrire un fichier csv :

```
movies_actor_list.to_csv("IMDB/movie.actors.tsv", index=False, sep="\t", na_rep="\\N",
quoting=csv.QUOTE_NONE)
```

Q6) A partir des morceaux de code précédents, créer le DataFrame “movies_actor_list” et le fichier “movie.actors.tsv”, qui ne contiennent que les information de “title.principals.tsv” concernant les rôles d’acteurs/actrices dans des films de cinéma.

Q7) A partir du fichier “name.basics.tsv”, créer un DataFrame “movie_actors_basics”, que vous écrirez dans un fichier “movies_actors.basics.tsv”, ne contenant que les informations personnelles relatives aux acteurs/actrices ayant joué dans un film de cinéma.

11 Exercices (90 min)

1) Combien de films ont obtenu la note de 10/10 ?

2) Afficher l’histogramme des notes des films sortis ces 10 dernières années.

3) Donner les noms, les nombres de votes et les notes moyennes des 5 films les mieux notés et ayant plus de 50000 notes.

4) Donner les noms, notes moyennes, nombres de votes et dates de sortie des 5 comédies avec plus de 50000 notes les mieux notées de ces 20 dernières années.

5) Donner la moyenne et écart-type des notes par genre de film (indice : vous pourrez regarder la méthode Series.str.split et DataFrame.explode).

6) Tracer l'évolution de la note moyenne des films d'action en fonction de l'année de sortie.

7) Donner la liste des 5 acteurs vivants ayant joué dans le plus de films.

8) Donner la liste des 5 acteurs ayant joué dans au moins 5 films, avec la filmographie la mieux notée.

9) Tracer le graphe "Nombre de films joués" versus "Note moyenne de la filmographie".