Phase 1 - Combination and preparation of the data

Team #5 | Movie Pruners

Zachary Chang - Hugo Tessier - Jameson Toper - Ellis Wright

For this phae 4 files need to be imported and merged:

- `title.akas.tsv`
- `title.basics.tsv`
- `title.ratings.tsv`
- `name.basics.tsv`

---

1 - IMPORT

Libraries used to import and treat the datasets. All the files has to be in a `data/` folder. `import_tsv` is a function

Entrée [1]:

```python
import pandas as pd
import numpy as np
```

Entrée [2]:

```python
datapath = "data/"
def import_tsv(filename):
    return pd.read_csv( datapath + filename, sep='\t', encoding='utf-8', dtype=str)
```

Notes: Few things can be noted after a first glance at the files and are considered in the `import_tsv` function.

- `sep='\t'` : the values are separated by tabulates (TSV files)
- `encoding='utf-8'` : the files are encoding in UTF-8
- `dtype=str` as we won't use datatypes in this phase, it's easier and faster to consider every columns as strings

Import `title.akas.tsv`

Entrée [3]:

```python
title_akas = import_tsv("title.akas.tsv")
```

Entrée [4]:

```
len(title_akas)
```

Out[4]:

```
28881100
```

Entrée [5]:

```
title_akas.head(10)
```

Out[5]:

	titleId	ordering	title	region	language	types	attributes	isOriginalTitle
0	tt0000001	1	Карменсіта	UA	\N	imdbDisplay	\N	0
1	tt0000001	2	Carmencita	DE	\N	\N	literal title	0
2	tt0000001	3	Carmencita - spanyol tánc	HU	\N	imdbDisplay	\N	0
3	tt0000001	4	Καρμενσίτα	GR	\N	imdbDisplay	\N	0
4	tt0000001	5	Карменсита	RU	\N	imdbDisplay	\N	0
5	tt0000001	6	Carmencita	US	\N	imdbDisplay	\N	0
6	tt0000001	7	Carmencita	\N	\N	original	\N	1
7	tt0000001	8	カルメンチータ	JP	ja	imdbDisplay	\N	0
8	tt0000002	1	Le clown et ses chiens	\N	\N	original	\N	1
9	tt0000002	2	Le clown et ses chiens	FR	\N	imdbDisplay	\N	0

Import `title.basics.tsv`

Entrée [6]:

```
title_basics = import_tsv("title.basics.tsv")
```

Entrée [7]:

```
len(title_basics)
```

Out[7]:

```
8213171
```

Entrée [8]:

```
title_basics.head(10)
```

Out[8]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1
5	tt0000006	short	Chinese Opium Den	Chinese Opium Den	0	1894	\N	1
6	tt0000007	short	Corbett and Courtney Before the Kinetograph	Corbett and Courtney Before the Kinetograph	0	1894	\N	1
7	tt0000008	short	Edison Kinetoscopic Record of a Sneeze	Edison Kinetoscopic Record of a Sneeze	0	1894	\N	1
8	tt0000009	short	Miss Jerry	Miss Jerry	0	1894	\N	40
9	tt0000010	short	Leaving the Factory	La sortie de l'usine Lumière à Lyon	0	1895	\N	1

Explode multiple valued cells

Some cells have multiple values separated by commas and need to by plit into several rows.

genres explosion

Entrée [9]:

```
title_basics_exploded = title_basics.assign(
    genres = title_basics['genres'].str.split(',')
).explode(
    'genres'
)
```

Entrée [10]:

```
len(title_basics_exploded)
```

Out[10]:

```
13012039
```

Entrée [11]:

```
title_basics_exploded.head(10)
```

Out[11]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1

Remark:

as a point of comparison we can count the number of different values in the multi-valued cells by counting the number of commas and compare the number of values to the number of rows in the final dataset.

Entrée [12]:

```
title_basics.assign(
    valueCount  = title_basics['genres'].map(lambda x: 1 + str(x).count(','))
)['valueCount'].sum()
```

Out[12]:

```
13012039
```

There is no difference. This is a simple way to validate our number of records.

Entrée [13]:

```
title_basics = title_basics_exploded
```

Import `title.ratings.tsv`

Entrée [14]:

```
title_ratings = import_tsv("title.ratings.tsv")
```

Entrée [15]:

```
len(title_ratings)
```

Out[15]:

```
1180677
```

Entrée [16]:

```
title_ratings.head(10)
```

Out[16]:

	tconst	averageRating	numVotes
0	tt0000001	5.7	1808
1	tt0000002	6.0	233
2	tt0000003	6.5	1559
3	tt0000004	6.1	152
4	tt0000005	6.2	2380
5	tt0000006	5.1	156
6	tt0000007	5.4	744
7	tt0000008	5.5	1963
8	tt0000009	5.8	189
9	tt0000010	6.9	6528

Import `name.basics.tsv`

Entrée [17]:

```
name_basics = import_tsv("name.basics.tsv")
```

Entrée [18]:

```
len(name_basics)
```

Out[18]:

```
11180384
```

Entrée [19]:

```
name_basics.head(10)
```

Out[19]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt0050419,
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0038355,
2	nm0000003	Brigitte Bardot	1934	\N	actress,soundtrack,music_department	tt0057345,
3	nm0000004	John Belushi	1949	1982	actor,soundtrack,writer	tt0078723,
4	nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	tt0050986,
5	nm0000006	Ingrid Bergman	1915	1982	actress,soundtrack,producer	tt0036855,
6	nm0000007	Humphrey Bogart	1899	1957	actor,soundtrack,producer	tt0033870,
7	nm0000008	Marlon Brando	1924	2004	actor,soundtrack,director	tt0047296,
8	nm0000009	Richard Burton	1925	1984	actor,soundtrack,producer	tt0057877,
9	nm0000010	James Cagney	1899	1986	actor,soundtrack,director	tt0031867,

Explode multiple valued cells

Two variables (`knownForTitles` and `primaryProfession`) have multiple values and nedd to be treated.

`knownForTitles` explosion

Entrée [20]:

```
name_basics_exploded_titles = name_basics.assign(
    knownForTitles = name_basics['knownForTitles'].str.split(',')
).explode(
    'knownForTitles'
)
```

Entrée [21]:

```
len(name_basics_exploded_titles)
```

Out[21]:

```
19504710
```

Entrée [22]:

```
name_basics_exploded_titles.head(10)
```

Out[22]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownFor
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt005
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt007
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt005
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt003
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt003
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt003
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt01
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt007
2	nm0000003	Brigitte Bardot	1934	\N	actress,soundtrack,music_department	tt005
2	nm0000003	Brigitte Bardot	1934	\N	actress,soundtrack,music_department	tt005

Remark:

as a point of comparison we can count the number of different values in the multi-valued cells by counting the number of commas.

Entrée [23]:

```
name_basics.assign(
    valueCount  = name_basics['knownForTitles'].map(lambda x: 1 + str(x).count(','))
)['valueCount'].sum()
```

Out[23]:

```
19504710
```

There isn't any difference.

`primaryProfession` explosion

Entrée [24]:

```
name_basics_exploded_titles_profession = name_basics_exploded_titles.assign(
    primaryProfession = name_basics_exploded_titles['primaryProfession'].str.split(',')
).explode(
    'primaryProfession'
)
```

Entrée [25]:

```
len(name_basics_exploded_titles_profession)
```

Out[25]:

```
29885305
```

Entrée [26]:

```
name_basics_exploded_titles_profession.head(10)
```

Out[26]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
0	nm0000001	Fred Astaire	1899	1987	soundtrack	tt0050419
0	nm0000001	Fred Astaire	1899	1987	actor	tt0050419
0	nm0000001	Fred Astaire	1899	1987	miscellaneous	tt0050419
0	nm0000001	Fred Astaire	1899	1987	soundtrack	tt0072308
0	nm0000001	Fred Astaire	1899	1987	actor	tt0072308
0	nm0000001	Fred Astaire	1899	1987	miscellaneous	tt0072308
0	nm0000001	Fred Astaire	1899	1987	soundtrack	tt0053137
0	nm0000001	Fred Astaire	1899	1987	actor	tt0053137
0	nm0000001	Fred Astaire	1899	1987	miscellaneous	tt0053137
0	nm0000001	Fred Astaire	1899	1987	soundtrack	tt0031983

Remark :

We now compare with an estimation of the number of rows. As in the previous explosion, we want to estimate the number of rows by counting the number of values in the cells.

Entrée [27]:

```
name_basics_exploded_titles.assign(
    valueCount  = name_basics_exploded_titles['primaryProfession'].map(lambda x: 1 + str(x)
)['valueCount'].sum()
```

Out[27]:

```
29885305
```

No difference.

Entrée [28]:

```
name_basics = name_basics_exploded_titles_profession
```

---

2 - FILTERING

Doing this stage before the merge makes it less memory-consuming. We need to filter several categories:

- filter on the **US** region
- filter on the type **movie**
- filter on the profession **actress/actor**

Filter on the *US* region

This informations is contained in the `region` variable of `title_akas` dataset.

Entrée [29]:

```
c_bold = "\033[1m"
c_end = "\033[0m"

# title_akas.region
print(c_bold + "title_akas.region categories\n" + c_end, title_akas.region.unique(), "\n")
```

```
title_akas.region categories
 ['UA' 'DE' 'HU' 'GR' 'RU' 'US' '\\N' 'JP' 'FR' 'RO' 'GB' 'PT' 'RS' 'ES'
 'UY' 'IT' 'AR' 'FI' 'PL' 'BR' 'DK' 'TR' 'XWW' 'XEU' 'SK' 'CZ' 'SE' 'MX'
 'NO' 'XYU' 'AT' 'VE' 'CSHH' 'SI' 'AU' 'TW' 'LT' 'IN' 'CA' 'NL' 'CO' 'IR'
 'BG' 'BE' 'SUHH' 'DZ' 'CH' 'NZ' 'BF' 'XWG' 'VN' 'CN' 'XSA' 'EE' 'IS' 'HR'
 'DDDE' 'HK' 'XKO' 'CL' 'IE' 'JM' 'PE' 'EG' 'GE' 'BY' 'BA' 'PA' 'TJ' 'XSI'
 'YUCS' 'ZA' 'MY' 'IL' 'PH' 'LV' 'PK' 'SG' 'BD' 'ID' 'CU' 'AL' 'BO' 'KR'
 'UZ' 'BUMM' 'XPI' 'TH' 'BJ' 'PR' 'CM' 'AZ' 'XAS' 'DO' 'EC' 'NG' 'MA' 'GL'
 'MN' 'LI' 'LU' 'PY' 'MZ' 'GT' 'BM' 'KZ' 'MD' 'CR' 'LB' 'IQ' 'TM' 'MK'
 'TN' 'HT' 'AM' 'SN' 'GH' 'CI' 'JO' 'KG' 'LK' 'NE' 'GN' 'VDVN' 'TD' 'SO'
 'SD' 'MC' 'CG' 'TT' 'GA' 'AE' 'BS' 'LY' 'SY' 'AO' 'KH' 'SV' 'MR' 'AF'
 'MG' 'ML' 'GY' 'CY' 'ET' 'GU' 'SR' 'MT' 'TG' 'PG' 'MU' 'BI' 'CF' 'NI'
 'ZW' 'ZM' 'GW' 'DJ' 'KP' 'RW' 'TZ' 'GI' 'LA' 'SC' 'NP' 'GP' 'FO' 'PS'
 'ZRCD' 'MO' 'AW' 'KW' 'CV' 'SL' 'SM' 'CD' 'TO' 'BT' 'LS' 'HN' 'KE' 'MQ'
 'AD' 'ER' nan 'SA' 'CSXX' 'IM' 'XKV' 'BH' 'BB' 'BZ' 'UG' 'AG' 'NU' 'OM'
 'QA' 'BW' 'LR' 'VC' 'AN' 'MV' 'YE' 'GM' 'KY' 'MM' 'ME' 'NC' 'DM' 'MP'
 'VA' 'GQ' 'FJ' 'SZ' 'RE' 'EH' 'PF' 'VG' 'LC' 'XAU' 'MW' 'BN' 'TL' 'ST'
 'KM' 'FM' 'AI' 'GD' 'VI' 'SB' 'GF' 'AQ' 'MH' 'CW' 'WS' 'AS' 'XNA' 'MS'
 'VU' 'SH' 'TV' 'CK' 'PW' 'NR' 'KN' 'JE' 'KI' 'TC']
```

The only category that corresponds to the US region is the category `US` .

Entrée [30]:

```
title_akas_filtered = title_akas.loc[title_akas['region'] == 'US', ]
```

Entrée [32]:

```python
print('from', len(title_basics), 'to', len(title_akas_filtered), 'rows')

title_akas_filtered
```

```
from 13012039 to 1283903 rows
```

Out[32]:

	titleId	ordering	title	region	language	types	attributes	isOrigina
5	tt0000001	6	Carmencita	US	\N	imdbDisplay	\N	
14	tt0000002	7	The Clown and His Dogs	US	\N	\N	literal English title	
35	tt0000005	1	Blacksmithing Scene	US	\N	alternative	\N	
39	tt0000005	5	Blacksmith Scene #1	US	\N	alternative	\N	
40	tt0000005	6	Blacksmithing	US	\N	\N	informal alternative title	
...	...	...	...	...	...	...	...	
28880807	tt9916720	10	The Demonic Nun	US	\N	tv	\N	
28880809	tt9916720	12	The Nun 2	US	\N	imdbDisplay	\N	
28880824	tt9916734	1	Manca: Peleo	US	\N	imdbDisplay	\N	
28880828	tt9916756	1	Pretty Pretty Black Girl	US	\N	imdbDisplay	\N	
28880844	tt9916764	1	38	US	\N	imdbDisplay	\N	

1283903 rows × 8 columns

Filter on the *movie* type

We searched for this information in 3 differents variables

Entrée [33]:

```
# title_akas.types
print(c_bold + "title_akas.types categories\n" + c_end, title_akas.types.unique(), "\n")

# title_basics.genres
print(c_bold + "title_basics.genres categories\n" + c_end, title_basics.genres.unique(), "\

# title_basics.titleType
print(c_bold + "title_basics.titleType categories\n" + c_end, title_basics.titleType.unique
```

```
title_akas.types categories
 ['imdbDisplay' '\\N' 'original' 'alternative' 'dvd' 'festival' 'working'
 'tv' 'video' 'imdbDisplay\x02tv' 'alternative\x02tv'
 'imdbDisplay\x02working' 'festival\x02imdbDisplay' 'tv\x02working'
 'imdbDisplay\x02video' 'alternative\x02dvd' 'tv\x02video'
 'dvd\x02imdbDisplay' 'alternative\x02working' 'video\x02working'
 'alternative\x02video' 'festival\x02working' 'dvd\x02video'
 'alternative\x02festival']

title_basics.genres categories
 ['Documentary' 'Short' 'Animation' 'Comedy' 'Romance' 'Sport' 'News'
 'Drama' 'Fantasy' 'Horror' 'Biography' 'Music' 'War' 'Crime' 'Western'
 'Family' 'Adventure' 'Action' 'History' 'Mystery' '\\N' 'Sci-Fi'
 'Musical' 'Thriller' 'Film-Noir' 'Talk-Show' 'Game-Show' 'Reality-TV'
 'Adult' nan]

title_basics.titleType categories
 ['short' 'movie' 'tvEpisode' 'tvSeries' 'tvShort' 'tvMovie' 'tvMiniSeries'
 'tvSpecial' 'video' 'videoGame' 'radioSeries' 'radioEpisode' 'tvPilot']
```

After a search in `title_akas.types` , `title_basics.genres` and `title_basics.titleType` it seems that the only interesting way to filter movies is with the last variable. We can keep the categories `movie` and `tvMovie` .

Entrée [34]:

```
title_basics_filtered = title_basics.loc[title_basics['titleType'].isin({'movie', 'tvMovie'
```

Entrée [35]:

```
print('from', len(title_basics), 'to', len(title_basics_filtered), 'rows')

title_basics_filtered
```

```
from 13012039 to 1032623 rows
```

Out[35]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMir
498	tt0000502	movie	Bohemios	Bohemios	0	1905	\N	
570	tt0000574	movie	The Story of the Kelly Gang	The Story of the Kelly Gang	0	1906	\N	
570	tt0000574	movie	The Story of the Kelly Gang	The Story of the Kelly Gang	0	1906	\N	
570	tt0000574	movie	The Story of the Kelly Gang	The Story of the Kelly Gang	0	1906	\N	
587	tt0000591	movie	The Prodigal Son	L'enfant prodigue	0	1907	\N	
...	...	...	...	...	...	...	...	
8213087	tt9916680	movie	De la ilusión al desconcierto: cine colombiano...	De la ilusión al desconcierto: cine colombiano...	0	2007	\N	
8213092	tt9916692	tvMovie	Teatroteka: Czlowiek bez twarzy	Teatroteka: Czlowiek bez twarzy	0	2015	\N	
8213099	tt9916706	movie	Dankyavar Danka	Dankyavar Danka	0	2013	\N	
8213110	tt9916730	movie	6 Gunn	6 Gunn	0	2017	\N	
8213121	tt9916754	movie	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	0	2013	\N	

1032623 rows × 9 columns

Filter on the *actress/actor* profession

Entrée [36]:

```
# name_basics.primaryProfession
print(c_bold + "name_basics.primaryProfession categories\n" + c_end, name_basics.primaryPro
```

```
name_basics.primaryProfession categories
 ['soundtrack' 'actor' 'miscellaneous' 'actress' 'music_department'
 'writer' 'director' 'producer' 'make_up_department' 'composer'
 'assistant_director' 'camera_department' 'editor' 'cinematographer'
 'casting_director' 'script_department' 'art_director' 'stunts'
 'editorial_department' 'costume_department' 'animation_department'
 'art_department' 'executive' 'special_effects' 'production_designer'
 'production_manager' 'sound_department' 'talent_agent'
 'casting_department' 'costume_designer' 'visual_effects'
 'location_management' 'set_decorator' 'transportation_department' nan
 'manager' 'legal' 'publicist' 'assistant' 'production_department'
 'electrical_department' 'choreographer']
```

Information contained in the `primaryProfession` variable of `name_basics` dataset. From this list we extract 2 categories : `actor` and `actress` . We decided not to keep `miscellaneous` category which seemed too vague.

We also kept all the other professions of the actress/actor and as additionnal information on them. To do so, we first identify the persons ID and keep all the rows with these IDs.

Entrée [37]:

```
kept_person = name_basics.loc[name_basics['primaryProfession'].isin({'actor', 'actress'}),

name_basics_filtered = name_basics[name_basics['nconst'].isin(list(kept_person))]
```

Entrée [38]:

```
print('from', len(name_basics), 'to', len(name_basics_filtered), 'rows')

name_basics_filtered
```

```
from 29885305 to 11633693 rows
```

Out[38]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
0	nm0000001	Fred Astaire	1899	1987	soundtrack	tt0050419
0	nm0000001	Fred Astaire	1899	1987	actor	tt0050419
0	nm0000001	Fred Astaire	1899	1987	miscellaneous	tt0050419
0	nm0000001	Fred Astaire	1899	1987	soundtrack	tt0072308
0	nm0000001	Fred Astaire	1899	1987	actor	tt0072308
...	...	...	...	...	...	...
11180366	nm9993701	Sanjai Kuriakose	\N	\N	actor	tt8736744
11180368	nm9993703	James Craigmyle	\N	\N	actor	tt10627062
11180368	nm9993703	James Craigmyle	\N	\N	actor	tt11212278
11180368	nm9993703	James Craigmyle	\N	\N	actor	tt6225166
11180368	nm9993703	James Craigmyle	\N	\N	actor	tt6914160

11633693 rows × 6 columns

3 - MERGE

Merge between `title.basics` , `title.akas` , `title.ratings` and `name.basics` can be made with the variable `tconst` (respectively `titleId` and `knownForTitles` in other tables) which corresponds to an alphanumeric unique identifier of the title according to the IMDb Datasets reference (https://www.imdb.com/interfaces/ (https://www.imdb.com/interfaces/)).

Entrée [39]:

```
title_basics_join = title_basics_filtered.set_index('tconst')
title_akas_join = title_akas_filtered.rename({'titleId':'tconst'}, axis = 1).set_index('tco
title_ratings_join = title_ratings.set_index('tconst')
name_basics_join = name_basics_filtered.rename({'knownForTitles':'tconst'}, axis = 1).set_i
```

Test of the keys for all the tables

Entrée [40]:

```
#test of the format of all the key before the join
expr = r'tt[0-9]+' #regular expression to identidy correctly formatted title IDs

print("title.akas:", title_akas_join.index.str.match(expr).all())
print("title.ratings:", title_ratings_join.index.str.match(expr).all())
print("title.basics:", title_basics_join.index.str.match(expr).all())
print("name.basics:", name_basics_join.index.str.match(expr).all())
```

```
title.akas: True
title.ratings: True
title.basics: True
name.basics: False
```

Some *good-formated* keys are missing in the `name_basics` dataset. It may worth investigating.

Incorrect format in `name.basics`

We first count the number of values that don't matching with the regular expression.

Entrée [41]:

```
np.logical_not(name_basics_join.index.str.match(expr)).sum()
```

Out[41]:

```
358325
```

We the most common title identifiers

Entrée [42]:

```
name_basics_join.index.value_counts().sort_values(ascending=False).head()
```

Out[42]:

```
\N           358325
tt0806910     10400
tt0441074      4885
tt0203259      4615
tt0098844      3761
Name: tconst, dtype: int64
```

We find that the most frequent is `'\N'` and that it corresponds exactly to the non-matching values amount. We just have to get rid of these missing values for title IDs.

Check for the column names conflicts

Entrée [43]:

```
set().intersection(
    set(name_basics_join.columns),
    set(title_basics_join.columns),
    set(title_akas_join.columns),
    set(title_ratings_join.columns)
)
```

Out[43]:

```
set()
```

There isn't any conflict with the column names in the different datasets.

join of the tables

Entrée [44]:

```
us_movies_actors = name_basics_join.join(
    title_basics_join, how = "inner"
).join(
    title_akas_join, how = "inner"
).join(
    title_ratings_join, how = "inner"
).reset_index()
```

Note:

For our further study, we assume that the movie needs to be present in all datasets so that we have information on its ratings, its characteristics, its famous actors, etc. That's the reason why we choose to make an `inner` join: we only keep titles that appear in all datasets. With all joins being inner, the order of the intersection doesn't change the result.

Final dataset

Entrée [45]:

```
us_movies_actors
```

Out[45]:

	tconst	nconst	primaryName	birthYear	deathYear	primaryProfession	titleType
0	tt0000591	nm0141150	Michel Carré	1865	1945	director	movie
1	tt0000591	nm0141150	Michel Carré	1865	1945	writer	movie
2	tt0000591	nm0141150	Michel Carré	1865	1945	actor	movie
3	tt0000591	nm1323543	Christiane Mandelys	1873	1957	actress	movie
4	tt0000591	nm1759558	Gilberte Sergy	\N	1924	actress	movie
...	...	...	...	...	...	...	...
9403922	tt9916362	nm8004301	Kiko Rossi	\N	\N	composer	movie
9403923	tt9916362	nm8004301	Kiko Rossi	\N	\N	composer	movie
9403924	tt9916428	nm9445072	Wang Peng Kai	\N	\N	actor	movie
9403925	tt9916428	nm9445072	Wang Peng Kai	\N	\N	actor	movie
9403926	tt9916428	nm9445072	Wang Peng Kai	\N	\N	actor	movie

9403927 rows × 23 columns

File writing

Entrée [46]:

```
filename = 'us.movies.actors.tsv'
us_movies_actors.to_csv( datapath + filename , sep='\t', encoding='utf-8', index = False)
```