Recommender System – Technical Report

Table of contents

Introduction

The goal of this mission is to build a movies and series recommender system from scratch based on the data from the French Website AlloCiné.fr.

There are 3 main steps to achieve this goal:

- **Step 1:** Webscraping the data from the AlloCiné website.
- **Step 2:** Cleaning and analysing the data to better understand it.
- **Step 3:** Building different recommender systems and comparing their efficiency.

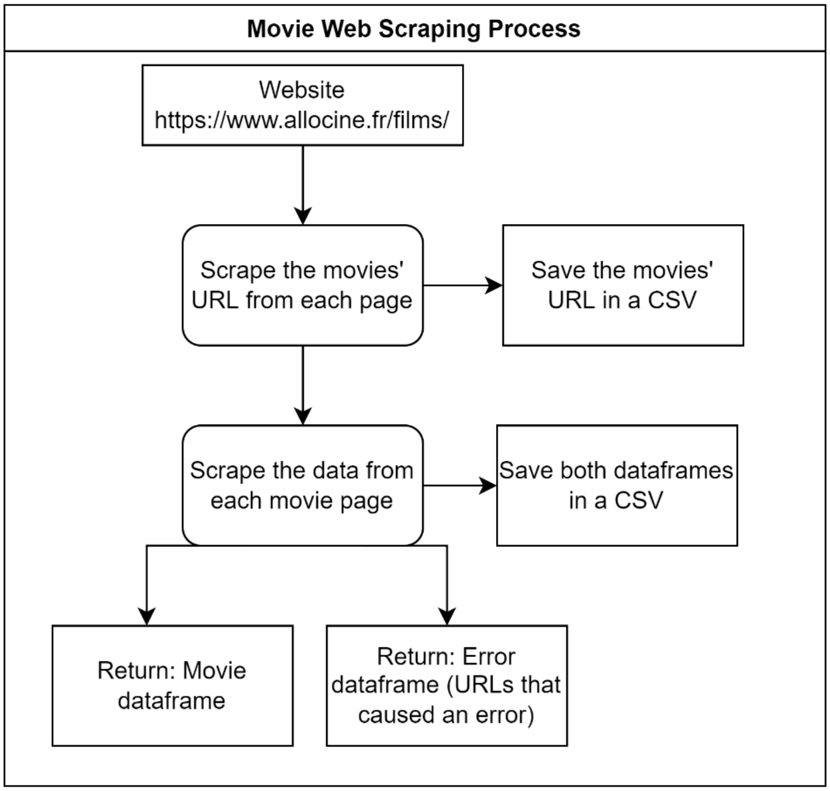
Step 1: Webscraping

References

Former projects found on GitHub were used as references to build this webscraping system. You can find those project made by Mr. Olivier MAILLOT here and there.

Movies Scraping

We proceed to the movies data scraping in the Webscraping Movies From AlloCine.ipynb notebook, which follows the following steps:


Movie Web Scraping Process
Website
https://www.allocine.fr/films/
Scrape the movies' URL from each page
Save the movies' URL in a CSV
Scrape the data from each movie page
Save both dataframes in a CSV
Return: Movie dataframe
Return: Error dataframe (URLs that caused an error)

For the **movies** data, we have the following columns to scrape:

- **id**: AlloCiné movie id
- **title**: the movie title (in French)
- **release_date**: the movie release date
- **duration**: the movie length (in minutes)
- **genres**: the movie genres (as a CSV string)
- **directors**: movie directors (as a CSV string)
- **actors**: main movie actors (as a CSV string)
- **nationality**: nationality of the movie (as a CSV string)
- **press_rating**: average press rating (from 0.5 to 5 stars ☆☆☆☆☆)
- **nb_press_rating**: number of ratings made by the press
- **spect_rating**: average AlloCiné users rating (from 0.5 to 5 stars ☆☆☆☆☆)
- **nb_spect_rating**: number of ratings made by the users/spectators
- **summary**: the movie summary (in French)
- **poster_link**: url of the movie poster

Libraries to import:

```
# Import libs
import pandas as pd
import numpy as np
import re
import unicodedata
from tqdm.notebook import tqdm_notebook as tqdm
from time import time
from time import sleep
from datetime import timedelta, datetime, date
from urllib.request import urlopen
from random import randint
from bs4 import BeautifulSoup
import dateparser
import os
from warnings import warn, filterwarnings
from IPython.core.display import clear_output
import traceback
```

Functions:

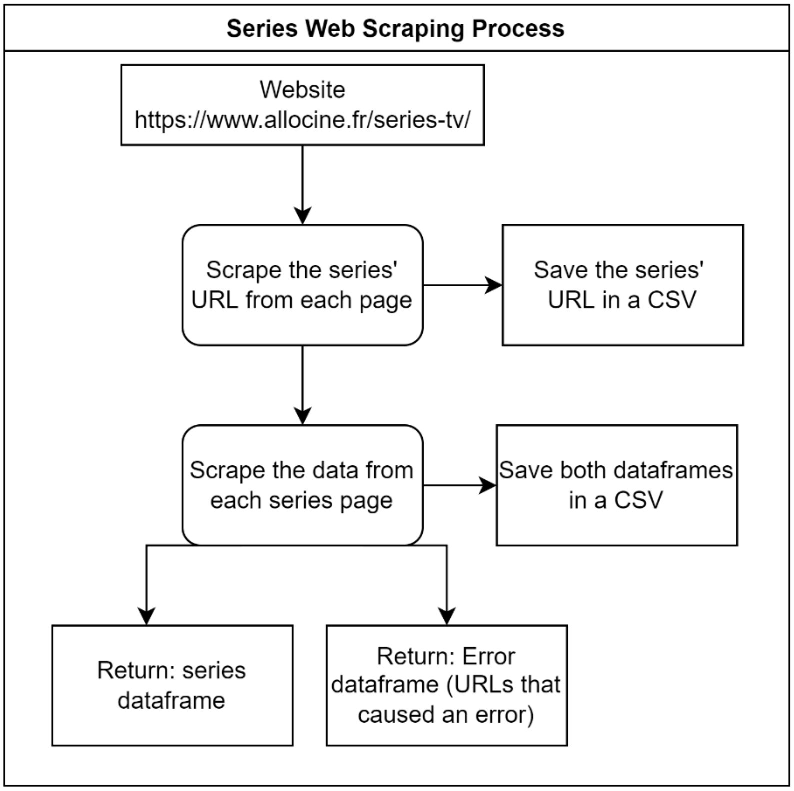
Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
URLs of each individual movie page	getMoviesUrl(start_page, end_page, nb_pages)	1 In the “Launching the script” part of the notebook	Scrape the movies urls from the AlloCine website's movie page (http://www.allocine.fr/films/). The range of pages to scrape goes from start_page to end_page (included) if end_page is not None or <= start_page. Else, the range of pages to scrape goes from start_page to start_page + nb_pages. It will ignore the movies that has not been released yet.	**start_page:** The first page to scrape. **end_page:** The last page to scrape (included) (optional). **nb_pages:** The number of pages to scrape (default 1).	List of movies url and a list of the movies that have not been released yet. Saves the list of movies urls in a csv file.
Movies’ data from the movie page url	ScrapeURL(movie_url, dwld_poster)	2 In the “Launching the script” part of the notebook	Scrape the data from the movies page urls.	**movie_url:** The list of movies page urls. **dwld_poster:** Boolean to download the movie poster (default False).	The dataframe of the movies and the dataframe of the urls that return an error. Both are saved into csv files.
Page content from the URL	get_url(url)	1.1 and 2.1 In the “getMoviesUrl(…)” and “ScrapeURL(…)” functions	Try to get and open the page url.	**url:** url to get the content from.	The content of the page
ID of the movie	get_movie_ID(movie_url)	2.2 In the “ScrapeURL(…)” function	Get the movie ID from the movie page url.	**movie_url:** URL of the movie page	The movie’s ID
Title of the movie	get_movie_title(movie_soup)	2.3 In the “ScrapeURL(…)” function	Scrape the movie title from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie’s title.
Release date of the movie	get_movie_release_date(movie_soup)	2.4 In the “ScrapeURL(…)” function	Scrape the movie release date from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie’s release date.

Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
Date month in English	convert_month(month)	2.4.1 In "get_movie_release_date(movie_soup)"	Convert French months into English months for the dateparser to work.	**month:** French month	English month
Duration of the movie	get_movie_duration (movie_soup)	2.5 In the "ScrapeURL(…)" function	Scrape the movie duration from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's duration. None if nothing is found
Genres of the movie	get_movie_genres(movie_soup)	2.6 In the "ScrapeURL(…)" function	Scrape the movie genres from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's genres.
Directors of the movie	get_movie_directors (movie_soup)	2.7 In the "ScrapeURL(…)" function	Scrape the movie directors from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's directors.
Actors of the movie	get_movie_actors(movie_soup)	2.8 In the "ScrapeURL(…)" function	Scrape the 3 displayed movie actors from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's actors.
Nationality of the movie	get_movie_nationality(movie_soup)	2.9 In the "ScrapeURL(…)" function	Scrape the movie nationality from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's nationality.
Average press rating of the movie	get_movie_ press_rating (movie_soup)	2.10 In the "ScrapeURL(…)" function	Scrape the movie average press rating from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's average press rating. None if no rating is found.
Number of press ratings for the movie	get_movie_press_rating_count(movie_soup)	2.11 In the "ScrapeURL(…)" function	Scrape the movie number of press ratings from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's number of press ratings. None if nothing is found.
Average spectators'/users' rating of the movie	get_movie_ spect_rating (movie_soup)	2.12 In the "ScrapeURL(…)" function	Scrape the movie average spectators'/users' rating from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's average spectators' rating. None if no rating is found.
Number of spectators ratings for the movie	get_movie_spect_rating_count(movie_soup)	2.13 In the "ScrapeURL(…)" function	Scrape the movie number of spectators'/users' ratings from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's number of spectators'/users' ratings. None if nothing is found.

Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
Summary of the movie	get_movie_summary(movie_soup)	2.14 In the "ScrapeURL(…)" function	Scrape the movie summary from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's summary. None if no summary is found
Poster of the movie	get_movie_poster(movie_soup)	2.15 In the "ScrapeURL(…)" function	Scrape the movie poster link from the movie page.	**movie_soup:** BeautifulSoup object of the movie page.	The movie's poster link
Download the movie poster	download_movie_poster(poster_link, movie_title)	2.16 and 0.1 In the "ScrapeURL(…)" and "download_all_posters(…)" functions	Download the movie poster from the poster link.	**poster_link:** The poster link. **movie_title:** The movie title.	Nothing but saves the movie poster as a jpg file.
Download all the posters at once.	download_all_posters(movies_df)	0 No specific place. Can be called anywhere once the movies have been scraped	Download all the movies poster from the posters' links in the dataframe if **dwld_poster** was set as **False** in the "ScrapeURL(…)" function.	**movies_df:** The movies dataframe	Nothing but saves all the movies posters as jpg files.

Series Scraping

We proceed to the series data scraping in the Webscraping_Series_From_AlloCine.ipynb notebook, which follows the following steps:


Series Web Scraping Process
Website https://www.allocine.fr/series-tv/
Scrape the series' URL from each page
Save the series' URL in a CSV
Scrape the data from each series page
Save both dataframes in a CSV
Return: series dataframe
Return: Error dataframe (URLs that caused an error)

This step follows the same process as the one for the movies, but we added some more features to scrape. For the **series** data, we have the following columns :

- **id**: AlloCiné series id
- **title**: the series title (in French)
- **status**: the series status (in French) (En cours|Terminée|Annulée|À venir)
- **release_date**: the series release date
- **duration**: the series average episode length (in minutes)
- **nb_seasons**: the number of seasons
- **nb_episodes**: the number of episodes
- **genres**: the series genres (as a CSV string)
- **directors**: series directors (as a CSV string)
- **actors**: main actors of the series (as a CSV string)
- **nationality**: nationality of the series (as a CSV string)

- **press_rating**: average press rating (from 0.5 to 5 stars ☆☆☆☆☆)
- **nb_press_rating**: number of ratings made by the press
- **spect_rating**: average AlloCiné users rating (from 0.5 to 5 stars ☆☆☆☆☆)
- **nb_spect_rating**: number of ratings made by the users/spectators
- **summary**: the series summary in French
- **poster_link**: url of the series poster

Libraries to import:

```
# Import libs
import pandas as pd
import numpy as np
import re
import unicodedata
from tqdm.notebook import tqdm_notebook as tqdm
from time import time
from time import sleep
from datetime import timedelta
from urllib.request import urlopen
from random import randint
from bs4 import BeautifulSoup
import os
from warnings import warn
from IPython.core.display import clear_output
import traceback
```

Functions:

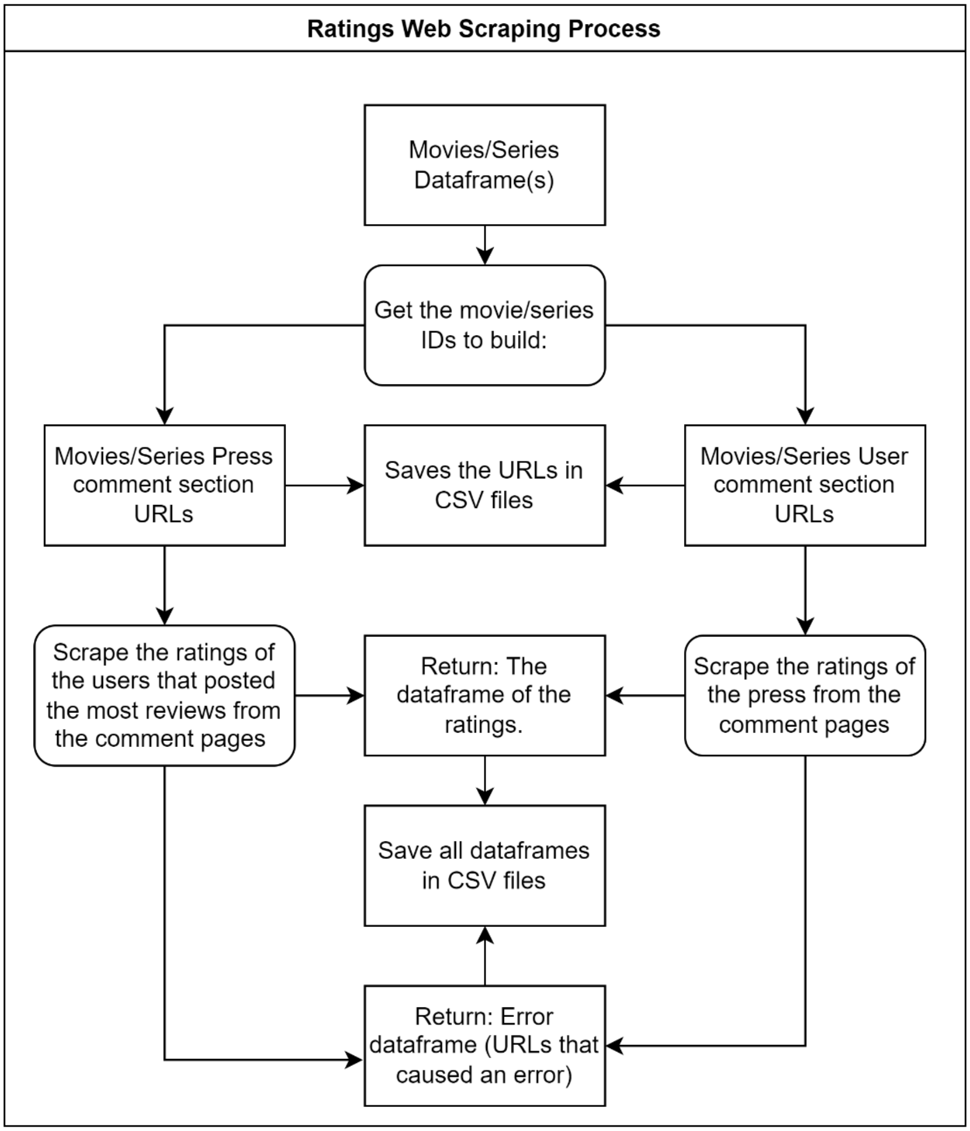
Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
URLs of each individual series page	getSeriesUrl(start_page, end_page, nb_pages)	1 In the "Launching the script" part of the notebook	Scrape the series urls from the AlloCine website's series page (http://www.allocine.fr/series-tv/). The range of pages to scrape goes from start_page to end_page (included) if end_page is not None or <= start_page. Else, the range of pages to scrape goes from start_page to start_page + nb_pages. It will ignore the series that has not been released yet.	**start_page:** The first page to scrape. **end_page:** The last page to scrape (included) (optional). **nb_pages:** The number of pages to scrape (default 1).	List of series url and a list of the series that have not been released yet. Saves the list of series urls in a csv file.
Series' data from the series page url	ScrapeURL(series_url, dwld_poster)	2 In the "Launching the script" part of the notebook	Scrape the data from the series page urls.	**series_url:** The list of series page urls. **dwld_poster:** Boolean to download the series poster (default False).	The dataframe of the series and the dataframe of the urls that return an error. Both are saved into csv files.
Page content from the URL	get_url(url)	1.1 and 2.1 In the "getSeriesUrl(…)" and "ScrapeURL(…)" functions	Try to get and open the page url.	**url:** url to get the content from.	The content of the page.
ID of the series	get_series_ID(series_url)	2.2 In the "ScrapeURL(…)" function	Get the series ID from the series page url.	**series_url:** URL of the series page.	The series ID.
Title of the series	get_series_title(series_soup)	2.3 In the "ScrapeURL(…)" function	Scrape the series title from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series title.
Status of the series	get_series_status(series_soup)	2.4 In the "ScrapeURL(…)" function	Scrape the series status from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series status.

Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
Release date of the series	get_series_release_date(series_soup)	2.5 In the "ScrapeURL(…)" function	Scrape the series release date from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series release date.
Duration of the series	get_series_duration(series_soup)	2.6 In the "ScrapeURL(…)" function	Scrape the series duration from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series duration. None if nothing is found
Number of seasons and episodes of the series	get_series_nb_seasons_episodes(series_soup)	2.7 In the "ScrapeURL(…)" function	Scrape the series number of seasons and episodes from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series number of seasons and episodes.
Genres of the series	get_series_genres(series_soup)	2.8 In the "ScrapeURL(…)" function	Scrape the series genres from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series genres.
Directors of the series	get_series_directors(series_soup)	2.9 In the "ScrapeURL(…)" function	Scrape the series directors from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series directors.
Actors of the series	get_series_actors(series_soup)	2.10 In the "ScrapeURL(…)" function	Scrape the 3 displayed series actors from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series actors.
Nationality of the series	get_series_nationality(series_soup)	2.11 In the "ScrapeURL(…)" function	Scrape the series nationality from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series nationality.
Average press rating of the series	get_series_press_rating(series_soup)	2.12 In the "ScrapeURL(…)" function	Scrape the series average press rating from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series average press rating. None if no rating is found.
Number of press ratings for the series	get_series_press_rating_count(series_soup)	2.13 In the "ScrapeURL(…)" function	Scrape the series number of press ratings from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series number of press ratings. None if nothing is found.
Average spectators'/users' rating of the series	get_series_spect_rating(series_soup)	2.14 In the "ScrapeURL(…)" function	Scrape the series average spectators'/users' rating from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series average spectators' rating. None if no rating is found.

Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
Number of spectators ratings for the series	get_series_spect_rating_count(series_soup)	2.15 In the "ScrapeURL(…)" function	Scrape the series number of spectators'/users' ratings from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series number of spectators'/users' ratings. None if nothing is found.
Summary of the series	get_series_summary(series_soup)	2.16 In the "ScrapeURL(…)" function	Scrape the series summary from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series summary. None if no summary is found
Poster of the series	get_series_poster(series_soup)	2.17 In the "ScrapeURL(…)" function	Scrape the series poster link from the series page.	**series_soup:** BeautifulSoup object of the series page.	The series poster link
Download the series poster	download_series_poster(poster_link, series_title)	2.18 and 0.1 In the "ScrapeURL(…)" and "download_all_posters(…)" functions	Download the series poster from the poster link.	**poster_link:** The poster link. **movie_title:** The movie name.	Nothing but saves the movie poster as a jpg file.
Download all the posters at once.	download_all_posters(series_df)	0 No specific place. Can be called anywhere once the series have been scraped	Download all the series poster from the posters' links in the dataframe if **dwld_poster** was set as **False** in the ScrapeURL(…) function.	**series_df:** The series dataframe	Nothing but saves all the series posters as jpg files.

Ratings Scraping

Once we scraped the movies and series data, we can proceed to the ratings scraping in the Webscraping_Ratings_From_AlloCine.ipynb notebook, which follows the following steps:


Ratings Web Scraping Process
Movies/Series Dataframe(s)
Get the movie/series IDs to build:
Movies/Series Press comment section URLs
Saves the URLs in CSV files
Movies/Series User comment section URLs
Scrape the ratings of the users that posted the most reviews from the comment pages
Return: The dataframe of the ratings.
Scrape the ratings of the press from the comment pages
Save all dataframes in CSV files
Return: Error dataframe (URLs that caused an error)

In the process, we extract, if available:

All of these dataframes have the similar following columns:

- user_id: AlloCiné user id (unavailable for the press)
- (user/press)_name: AlloCiné user/press name
- (movie/series)_id: AlloCiné movie/series id
- (user/press)_rating: AlloCiné user/press rating (from 0.5 to 5 stars ☆☆☆☆☆)
- date: date of the rating (unavailable for the press)

Libraries to import:

```
# Import libs
import pandas as pd
import numpy as np
from tqdm.notebook import tqdm_notebook as tqdm
from time import time
from time import sleep
import re
from datetime import timedelta, date
from urllib.request import urlopen
from random import randint
from bs4 import BeautifulSoup
import os
import dateparser
from warnings import warn, filterwarnings
from IPython.core.display import clear_output
import traceback
```

Functions:

Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
Create dataframes from the movies and series data stored in CSV files	loadDataFrames(nrows)	1 In the "Launching the script" section	Load the movies and series dataframes from the csv files.	**nrows:** number of rows to load	The movies and series dataframes
The comments page URLs of the users and the press from the movies and series	getCommentsUrl(movies_df, series_df, spect, press)	2 In the "Launching the script" section	Get the comments section url for each movie and series. You can select the spectators or press section, or both.	**movies_df:** DataFrame of movies **series_df:** DataFrame of series **spect:** Boolean, True if you want to get the spectators section **press:** Boolean, True if you want to get the press section	Nothing but saves the urls in CSV files.
The comments page URLs of the users and the press from the movies	getMoviesCommentsUrl(movies_df, spect, press)	2.1 In the "getCommentsUrl(…)" function	Get the comments section url for each movie. You can select the spectators or	**movies_df:** DataFrame of movies **spect:** Boolean, True if you want to get the spectators section	Nothing but saves the urls in CSV files.

			press section, or both.	**press**: Boolean, True if you want to get the press section	
The comments page URLs of the users and the press from the series	getSeriesCommentsUrl(series_df, spect, press)	2.2 In the "getCommentsUrl(…)" function	Get the comments section url for each series. You can select the spectators or press section, or both.	**series_df:** DataFrame of series **spect**: Boolean, True if you want to get the spectators section **press**: Boolean, True if you want to get the press section	Nothing but saves the urls in CSV files.
Load the comments URLs of the users and the press from movies and series into the script	loadingURL(movies, series, spect, press)	3 In the "Launching the script" section	Load the urls of the movies and series pages.	**movies:** Boolean, if True, load the movies urls **series:** Boolean, if True, load the series urls **spect:** Boolean, if True, load the spectator urls for movies and/or series **press:** Boolean, if True, load the press urls for movies and/or series	The urls lists of the movies and series press and user rating pages
All the rating data from the movies and series	ScrapeURL(press_series_urls, press_movies_urls, user_series_urls, user_movies_urls, nb_users, nb_series_folds, nb_movies_folds, series_fold_start, movies_fold_start)	4 In the "Launching the script" section	Scrape the user and press ratings from the movie and series pages.	**press_series_urls:** list of press series urls **press_movies_urls:** list of press movies urls **user_series_urls:** list of user series urls **user_movies_urls:** list of user movies urls **nb_users:** number of users to scrape (default: 5). **nb_series_folds:** number of folds to split the series scraping (default: 10). **nb_movies_folds:**number of folds to split the movies scraping (default: 10). **series_fold_start:** Number of the fold	2 lists of dataframes of user and press: one for the ratings and the other one for the errors.

				to start/resume the scraping for the series. **movies_fold_start:** Number of the fold to start/resume the scraping for the movies.	
All press rating data from the movies and series	ScrapePressURL(series_urls, movies_urls)	4.1 In the “ScrapeURL(…)” function	Scrape the press_ratings from the series and/or movies press comments section urls (if available).	**series_url:** list of series press comments section urls. **movies_url:** list of movies press comments section urls.	Dataframes of series and movies press ratings and the errors.
All user rating data from the movies and series	ScrapeUserURL(series_urls, movies_urls, nb_users, nb_series_folds, nb_movies_folds, series_fold_start, movies_fold_start)	4.2 In the “ScrapeURL(…)” function	Scrape the user_ratings from the series and/or movies press comments section urls (if available). As the process takes a lot of time, we divided the urls into folds so that we can save the data regularly.	**series_url:** list of series press comments section urls. **movies_url:** list of movies press comments section urls. **nb_users:** number of users to scrape (default: 5) **nb_series_folds:** number of folds to split the series scraping (default: 10). **nb_movies_folds:**number of folds to split the movies scraping (default: 10). **series_fold_start:** Number of the fold to start/resume the scraping for the series. **movies_fold_start:** Number of the fold to start/resume the scraping for the movies.	Dataframes of series and movies user ratings and the errors.
Page content from the URL	get_url(url)	4.1.1 and 4.2.1 In the “ScrapePressURL(…)” and	Try to get and open the page url.	**url:** url to get the content from.	The content of the page

		“ScrapeUserURL(…)” functions			
ID of a movie or series	get_ID(url)	4.1.2 and 4.2.2 In the “ScrapePressURL(…)” and “ScrapeUserURL(…)” functions	Get the movie or serie ID from an url.	**url:** url containing the movie or serie ID	The movie or serie ID.
The press rating data of a movie or series page	getPressRatings(press_soup)	4.1.3 In the “ScrapePressURL(…)” function	Get press_ratings from the comments section url for each movie or series.	**press_soup:** BeautifulSoup object of the press comments section url	A dictionary of ratings (key: press name, value: press rating) if the ratings are available, else None.
The user rating data of a movie or series page	getUserRatings(user_rating_url, last_page, nb_users)	4.2.3 In the “ScrapeUserURL(…)” function	Get user_ratings from the comments section url for each movie or series. We will keep only the nb_users first users who have posted the most reviews.	**user_rating_url :** url of the user ratings section. **last_page:** last page of the user ratings section (default 1). **nb_users:** Number of users to keep (default 5).	A dictionary of ratings {key=(user_id, user_name): value=(user rating, date)} if the ratings are available, else None.
The rating given by the press for a movie or series	convertTextToRating(text)	4.1.3.1 In the “getPressRatings(…)” function	Convert the press evaluation text into a rating.	**text:** Evaluation of the movie as a string	The corresponding float rating value.
Date month in English	convert_month(month)	4.2.3.1 In the “getUserRatings(…)” function	Convert French months into English months for the dateparser to work.	**month:** French month	English month

Step 2: Data Analysis

Data Cleaning

After retrieving all the data needed and storing them in a safe place to avoid overwriting them, we can start cleaning the data after a quick analysis to ensure that we do not remove too much information. We execute this process in the notebook Allocine_Data_Analysis.ipynb.

Libraries to import:

```
# import libraries
import pandas as pd
import random as rd
import matplotlib.pyplot as plt
import re
import seaborn as sns
import os
from warnings import filterwarnings
import missingno as msno

# We ignore reindexing warnings
filterwarnings("ignore",message="Boolean Series key will be reindexed")
%matplotlib inline
```

Functions:

Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
Dataframes of the saved CSV files from the last data scraping	load_csv()	1 In the “#Data Cleaning##Load the csv files” section	Load the movies, series and all ratings dataframes from the csv files.		A dict of all the dataframes
Rename some columns and store their values into variables	In-cell execution	2 In the “#Data Cleaning## Rename and store columns names” section	Rename some columns’ names		
Statistics of a dataframe	display_stats(df, chart_type)	3 and 13 Called multiple times in the “##Display the first stats” and “##Display the last stats” sections	Display the statistics of a dataframe about their features and missing values.	**df:** dataframe to display stats from. **chart_type:** type of chart to display (matrix, bar or heatmap).	The "description" dataframe of pandas.

The summary table of all the dataframes.	display_summary_table(data)	4 At the end of the “##Display the first stats” and “##Display the last stats” sections	Display the summary table of the shape and unique values in the dataframes.	**data:** dict of dataframes.	The summary table of the dataframes.
Dropping rows	In-cell execution	5 In the “#Data Cleaning## Dropping rows” section	Drop duplicates in the dataframes and other specific rows in the series dataframe.		
Convert CSV strings values into arrays	convert_to_array(df)	6 In the “#Data Cleaning## Convert CSV strings to arrays” section	Convert CSV strings values into arrays	**df:** dataframe in which we convert CSV strings values into arrays.	
The genres dataframe	create_genre_df(df_key)	7 In the “#Data Cleaning## Create new tables” section	Create a dataframe that contains information about each genre.	**df_key:** The dataframe key of the data dict to get the genres from (movies or series).	The movies or series genre dataframe.
The nationality dataframe	create_nationality_df(df_key)	8 In the “#Data Cleaning## Create new tables” section	Create a dataframe that contains information about each nationality.	**df_key:** The dataframe key of the data dict to get the nationalities from (movies or series).	The movies or series nationality dataframe.
Movies/Series dataframe with missing duration imputed	impute_missing_duration(df_key)	9 Twice in “#Data Cleaning## Improve data format & coherence” section: 1 for movies, 1 for series	From the m/s_genres_df DataFrames, we order the genres by the amount of movies/series with that genre. We then compute the median duration of the movies/series with that genre. Finally, we impute the missing	**df_key:** Key of the dataframe in the data dictionary to impute the missing duration from.	The imputed dataframe.

			duration of the movies/series with the median duration of the movies/series with the most popular genre among their genres. We choose the median as it is more robust than the mean duration.		
Dropping rows with a thresh of missing values	In-cell execution	10 In “#Data Cleaning## Improve data format & coherence###Series” section	Drop series with at least na_thresh missing values		
Change the series release_date format	In-cell execution	11 In “#Data Cleaning## Improve data format & coherence###Series” section	Replace the release_date string by the release year as a date		
Get the movies release season	get_release_season(date)	12 In “#Data Cleaning## Improve data format & coherence###Movies” section	Get the release season date (eg: from "2019-05-10" return "Spring")	**date:** Date to get the release season from.	The release season of the movie.
Saving the Datasets	save_data(df, path, name)	14, 7.1 and 8.1 After creating the genres and nationality df and cleaning all dataframes	Save all dataframes into CSV files.		

The **genres** dataframe is the same for the movies and the series. It contains the following columns:

- **genres** : the genre name
- **nb_(movies/series)** : number of movies/series in this genre
- **avg_duration** : average duration of the movies/series in this genre
- **median_duration** : median duration of the movies/series in this genre
- **nb_press_rating** : number of ratings made by the press in this genre
- **nb_user_rating** : number of ratings made by the users/spectators in this genre
- **total_rating** : total rating made by the press and users in this genre
- **press_rating_percentage** : percentage of the total rating made by the press in this genre
- **user_rating_percentage** : percentage of the total rating made by the users/spectators in this genre
- **(movies/series)_percentage** : percentage of the total number of movies/series in this genre

The **nationality** dataframe is built the same way as the **genres** one mentioned above, except that it lacks the avg_duration and median_duration per nationality.

Data Analysis

In this same notebook, we proceed to a brief but significant analysis of the ratings.

Functions:

Object to retrieve	Function definition	Order and place of declaration	Description	Parameters	Return
Plot of the Rating Distribution	In-cell execution	1 In the “#Section 1. Ratings Distribution” section	Plot the distribution of the press and user ratings for movies and series		Saves the plots.
Plot of the Rating Correlation	In-cell execution	2 In the “#Section 1. Ratings Distribution” section	Plot the correlation between the press and user ratings for movies and series		Saves the plots.
Bar chart of the (user|press) ratings of movies and series compared to the	compare_max_ratings(rating_from, rating_to)	3 In the “#Section 1. Ratings Distribution” section	Compare the (user|press) ratings of movies and series for which the (press|user) gave the	**rating_from:** The dataset to compare the max ratings from (user|press). **rating_to:** The dataset to compare the max ratings to (press|user).	

(press|user) highest rating			highest rating.		