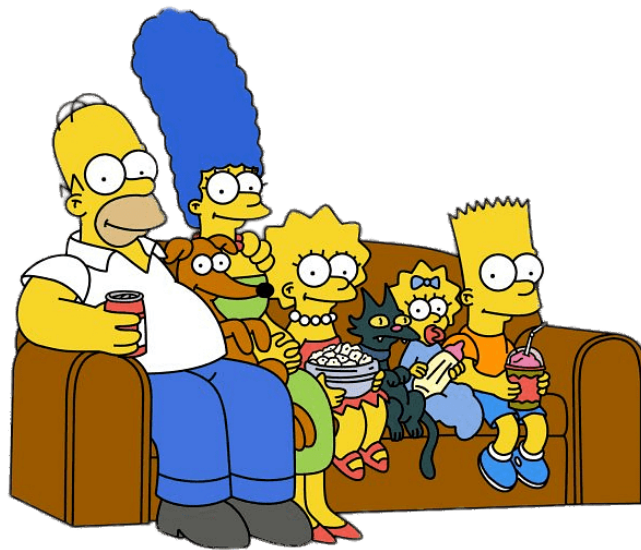




# *Best and worst of “The Simpsons” characters*



# Contents

<b>1. Introduction</b>	<b>3</b>
1.1 Objective	3
1.2 Solution	4
<b>2. Data acquisition</b>	<b>5</b>
2.1 Fandom characters scraping	6
2.2 Fandom episodes scraping	8
2.3 IMDb episodes scraping	10
<b>3. Raw data cleaning</b>	<b>13</b>
3.1 Fandom raw data	13
3.3 IMDb raw data	13
<b>4. CSV fix</b>	<b>15</b>
<b>5. Database</b>	<b>16</b>
5.1 Database choice	16
5.2 Database modeling	16
<b>6. Data cleaning and integration</b>	<b>20</b>
6.1 Character table	20
6.2 Alias table	22
6.3 Episode table	23
6.4 Main characters table	24
<b>7. Data quality</b>	<b>25</b>
7.1 Accuracy	25
7.2 Completeness	25
7.3 Currency	27
7.4 Consistency	27
<b>8. Data exploration</b>	<b>29</b>
<b>9. Conclusion and future developments</b>	<b>33</b>

## Roles

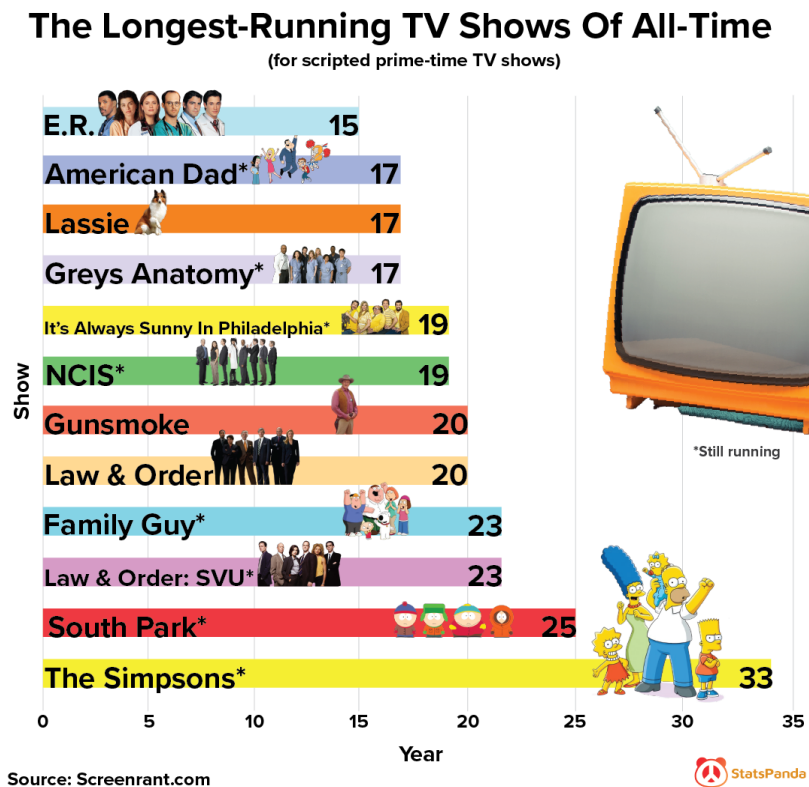
Paola Cavana: all Fandom episode related actions, episode integration and cleaning, quality assessment

Yuliia Tsymbal: all IMDb episode scraping related actions, episode integration and cleaning, quality assessment

Edoardo Fava: all Fandom character related actions, character deduplication and cleaning, database core modeling

# 1. Introduction

This project is about the best and the worst characters in “The Simpsons” tv series. It’s a show about parodies of American culture and society. The Simpsons is very popular, and it’s actually the longest-running American animated series, both in terms of seasons and number of episodes.



## 1.1 Objective

The objective of this project is to individuate, among all characters, the ones that makes the episode more appreciated and less appreciated, in other words, the characters that can make an episode good or bad.

So the questions we want to answer are:

1. Which character(s) makes the episodes **more** enjoyable?
2. Which character(s) makes the episodes **less** enjoyable?

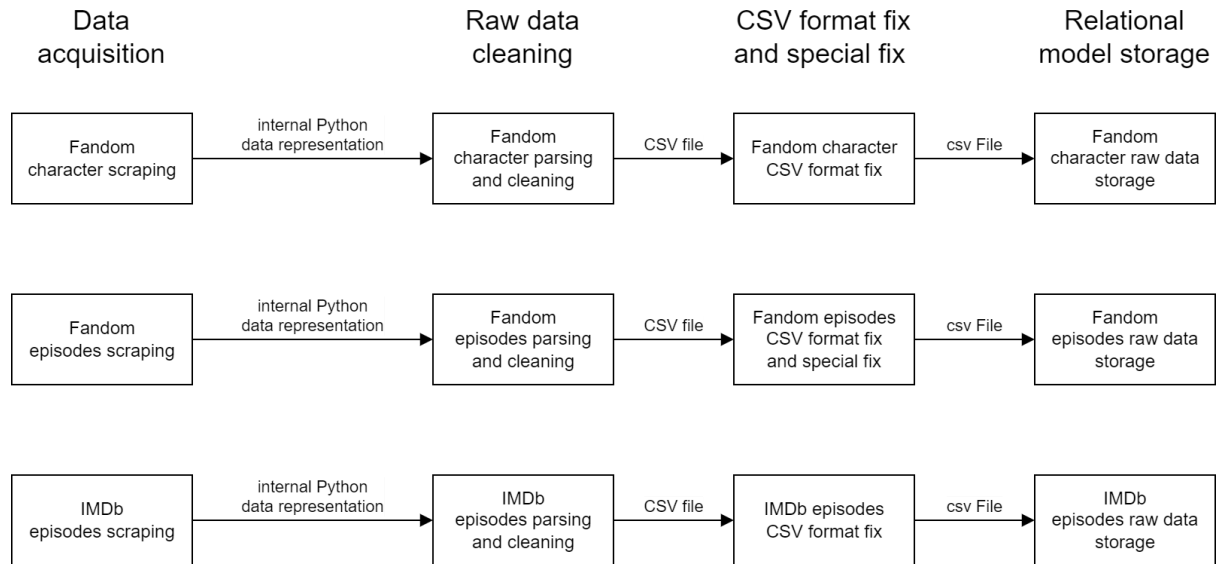
The objective and the topic are funny, but the research could also lead to more serious outcomes such as deciding to remove some characters or make a secondary character appear more in the future episodes.

In the best case, the changes to the show based on this project could lead to an increase in audience and appreciation.

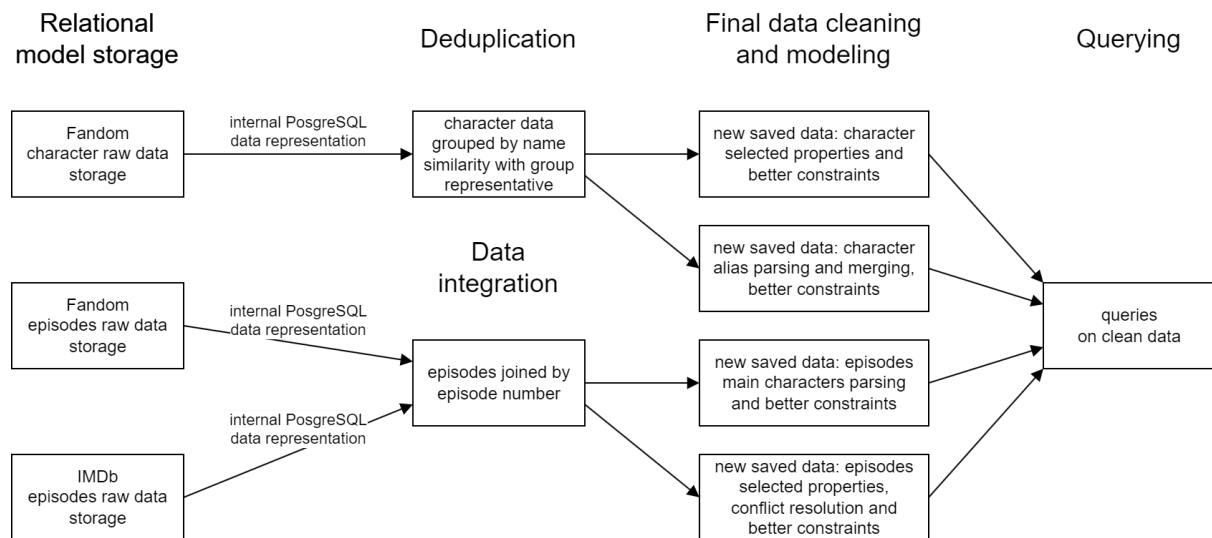
## 1.2 Solution

This is the workflow of our proposed solution. In the following chapters we will explain every step and the choices we made.

- **Part 1:** data retrieval and storage



- **Part 2:** data cleaning and better representation



## 2. Data acquisition

There is much data on the internet about The Simpsons, but some is not far close to completeness, much is not useful to connect characters and episodes, and almost all is user-generated and not verified.

We ignored the fact that content is not official or verified, because, at the end, we want to analyze the people's opinion, but our data sources must be complete or near-complete, because we need to query every episode. We essentially need two data for every episode: a measure of people's appreciation, and the main characters that appeared in that episode.

For people's appreciation, we decided to scrape data from the most popular website in the TV and movie field: [IMDb](#). For the episode main characters, we decided to scrape data from the largest wiki about The Simpsons (that also provided that field directly): the [Simpsons Fandom Wiki](#).

But we're not finished, because we have an issue with the characters: the same character with roles different from the usual, can have dedicated pages and result in completely different characters. Take as example Seymour Skinner:



The first image represents the original version of the character, the second is him in the role of Springfield Volunteer of Fire Department and the third is Stonecutter Skinner. All the three have different wiki pages.

In order to merge or link the characters correctly (and in order to take into account more data on a character than just his/her name) we decided to also scrape the characters from the [Simpsons Fandom Wiki](#).

## 2.1 Fandom characters scraping

The website in the section of characters is organized as follows:

(1) Pages containing a list of max 200 characters, from now on called *characters pages* (thematic, based on categories, etc.). There is the option to list all characters in alphabetical order, in that case a page with the first 200 characters is provided along with a “next” button, available until the last page.

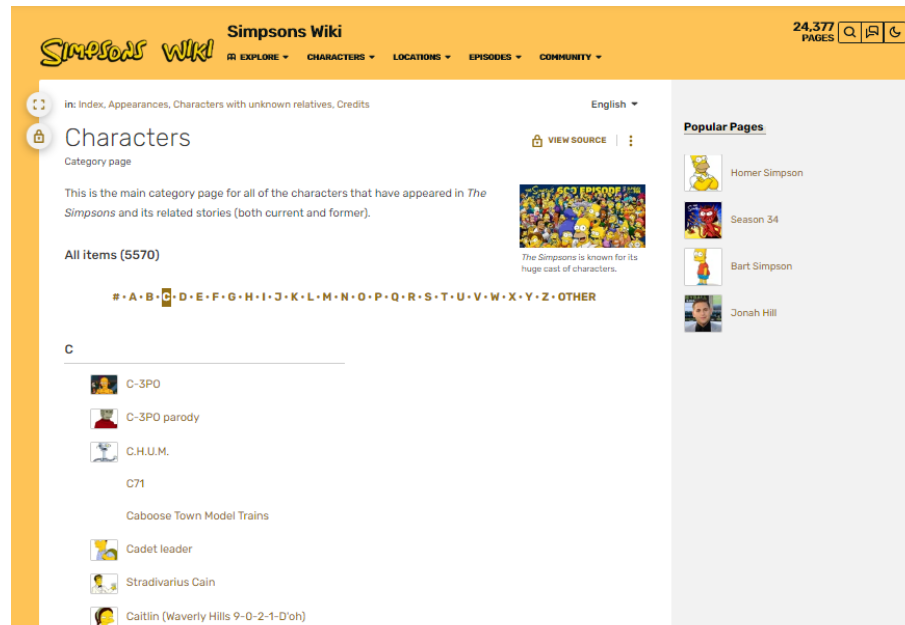


Figure 1. Characters list page example

(2) A page for every character, from now on called *character page*. Every character page usually contains a description and always contains an info-box (Wikipedia style) with useful information that is nearly structured.

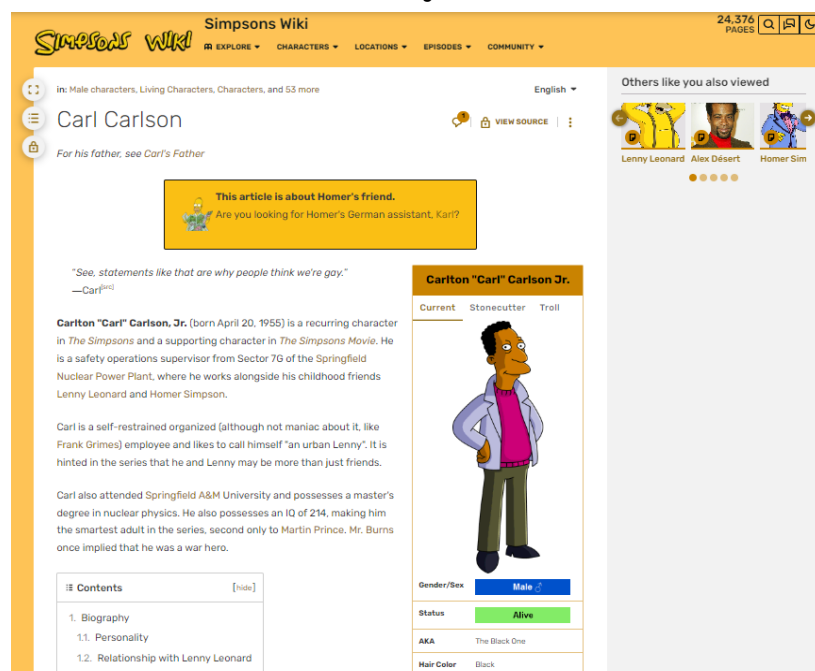


Figure 2. Character page example

The Fandom character scraper is a Python program, and has the structure explained in the following UML package diagram.

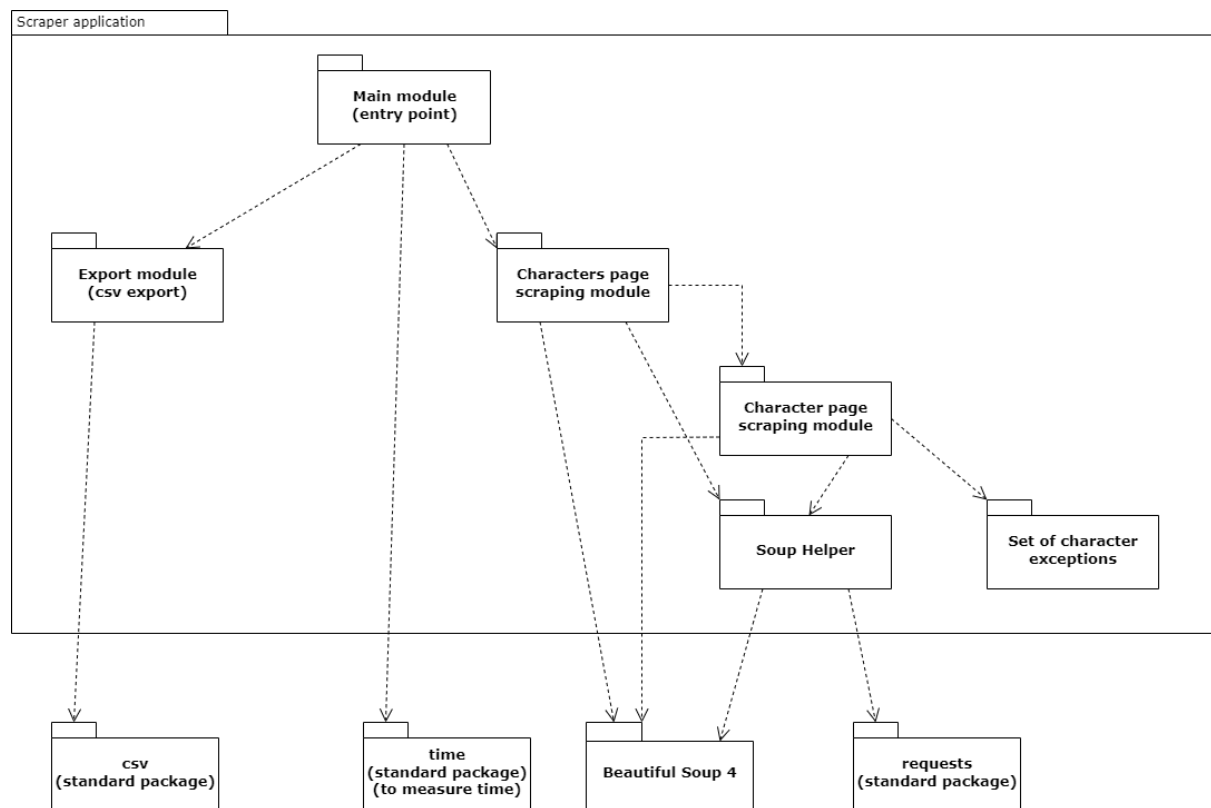


Figure 3. Structure of Fandom Character Scraper

The main module is responsible for the execution of the program, in particular the outer loop on all characters pages. The characters page module is responsible for scraping all the single characters urls from a page, and scrape every character calling the character page module. Note that the main module has not a dependency on BeautifulSoup, the library used to scrape information from HTML documents.

The main work of the scraping program resides in the character page scraper and the soup helper module. Almost every property of the characters has a format different from the others, and in the same property many different cases of HTML tag usage can appear. The character module handles every property separately in different ways, while the soup helper module handles the removal or replacement of the different HTML tags in the various necessary ways (independently from the context of the property).

A set of character exceptions became necessary when the scraping failed (mainly) for some syntax errors that originated from human error. For example, [Aunt Edith](#) has gender and status swapped. We are considering fixing these issues directly given the possibility to edit data by the Fandom wiki.

The scraped properties are

- `known_as`: the usual name of the character
- `full_name`: the complete name
- `image_url`: url of the first image of the character
- `age`: age parsed to integer
- `species`: attribute for animal characters
- `gender`: mainly “male” or “female” or “unknown”, multiple options are allowed and comma separated (example: [old opening sequence Lisa's classmates](#) is a group composed by both females and males)
- `status`: mainly “alive” or “deceased”, multiple options are allowed and comma separated (example: [Thanos](#) because he died in the Marvel universe only)
- `fictional`: if the character is fictional inside or outside the world of The Simpsons (example inside: [Poochie](#) from The Simpsons television) (example outside: [Yoda](#) from Star Wars)
- `alias`: a list of comma separated aliases
- `hair_color`: hair color or status (like bald), multiple values are allowed and comma separated
- `color`: attribute for non-human characters (it is not the skin color)
- `birth_country`: country of birth of the character
- `job`: list of comma separated jobs of the character
- `first_appearance`: name of the first episode where this character appeared
- `first_mentioned`: name of the first episode where this character was mentioned
- `voice`: voice actor or list of comma separated voice actors

The csv file columns are all the above attributes plus the page URL (`fandom_url`). Given the various comma separated fields, we decided semicolon “;” as the separator.

## 2.2 Fandom episodes scraping

Fandom has a page dedicated to all episodes of the series, which contains a complete list of them from season 1 through season 34. Typically a season contains 22 episodes, but there are exceptions.

The seasons are divided into tables, where each table contains rows of how many episodes of the season. For each episode there is the title, the original airdate, the production code and the episode number, both overall and related to the season.



## Season 1 (1989/90)

Main article: *Season 1*

Title	Original airdate	Production code	#
"Simpsons Roasting on an Open Fire"	December 17, 1989	7G08	1-001
"Bart the Genius"	January 14, 1990	7G02	2-002
"Homer's Odyssey"	January 21, 1990	7G03	3-003
"There's No Disgrace Like Home"	January 28, 1990	7G04	4-004
"Bart the General"	February 4, 1990	7G05	5-005
"Moaning Lisa"	February 11, 1990	7G06	6-006
"The Call of the Simpsons"	February 18, 1990	7G09	7-007
"The Telltale Head"	February 25, 1990	7G07	8-008
"Life on the Fast Lane"	March 18, 1990	7G11	9-009
"Homer's Night Out"	March 25, 1990	7G10	10-010
"The Crepes of Wrath"	April 15, 1990	7G13	11-011
"Krusty Gets Busted"	April 29, 1990	7G12	12-012
"Some Enchanted Evening"	May 13, 1990	7G01	13-013

Figure 4. Episode table on Fandom site example

Each episode has its own page, where we first find the title, after that a particular quote of the episode and other details, such as a: the plot, the complete story, references, the production, the reception, etc. The most important part for scraping the episodes was the infobox on the right, which contains the most useful information for our purpose. For each episode are specified: the episode number, the production code, the original airdate, main character(s), the writer and the director of the episode. In our case the most useful attribute for our purpose was precisely that which refers to the main character(s), but anyway in the database of Fandom episodes were also inserted the other attributes.











url	title	image_url	season	episode_number_...	production...	airdate	main_characters	written_by	directed_by
									
<a href="https://si...">https://si...</a>	Principal ...	<a href="https://stat...">https://stat...</a>	2	27	7F15	February ...	Seymour Skinner,Pat...	David M. St...	Mark Kirkland...
<a href="https://si...">https://si...</a>	Oh Broth...	<a href="https://stat...">https://stat...</a>	2	28	7F16	February ...	Homer Simpson,Her...	Jeff Martin	Wes Archer
<a href="https://si...">https://si...</a>	Bart's Do...	<a href="https://stat...">https://stat...</a>	2	29	7F14	March 7, ...	Bart Simpson,Santa'...	Jon Vitti	Jim Reardon,...
<a href="https://si...">https://si...</a>	Old Money	<a href="https://stat...">https://stat...</a>	2	30	7F17	March 28...	Grampa Simpson,H...	Jay Kogen ...	David Silverm...

Figure 5. Some row of Fandom Episodes Dataset

The image below shows a typical episode page on the Fandom site:

The screenshot shows the Fandom page for the episode "Burger Kings". At the top, the title "Burger Kings" is displayed with a "VIEW SOURCE" link. Below the title is a navigation bar with tabs: "Episode" (selected), "References", "Gags", "Appearances", "Gallery", "Quotes", and "Credits". A secondary navigation bar shows episode thumbnails: "Uncut Femmes", "Burger Kings" (selected), and "Panic on the Streets of Springfield".

Below the navigation bar, a quote is displayed: "I did it again. I won by doing nothing." —Krusty The Clown. This is followed by the text: "Burger Kings is the eighteenth episode of Season 32."

A "Contents" section is visible, listing: 1. Synopsis, 2. Full story, 3. Production, and 4. Reception. The "Synopsis" section is expanded, showing the text: "Mr. Burns gets into the plant-based burger business. Lisa refuses to".

On the right side, there is a table with episode details:

Burger Kings	
Episode Number	702
Production Code	QABF11
Original Airdate	April 11, 2021
Main character(s)	Lisa Simpson Charles Montgomery Burns
Couch Gag	Swiss Pocket Knife couch gag

Figure 6. Episode page example

The Fandom episodes scraper is a Python program, is a scraping done on the same site as the characters, so it has the same structure as the Fandom characters scraper.

## 2.3 IMDb episodes scraping

The IMDb resource contains, like the Fandom, a special section with a list of seasons and episodes in each of them. Each page corresponds to one season of the series. The main information contained on this home page is the title of the episode, the rating, the number of votes forming the rating, the release date and a short description. The page of each episode contains similar information, but with a larger description and actors who voiced the roles, etc.

The main information that must be obtained by scraping is the link for each episode page, the name of the episode, the season number, the episode number, the release date, the received rating and the number of votes that formed this rating.

Taking into account the structure of the site and in order to ensure more convenient and universal access for each element of information on the page, the following scraping algorithm was written. To begin with, we open a page with a list of all episodes of one season (fig. 1) and receive links to the personal page of

each of the episodes. Then, using the received links, we go to the page (fig. 2) of each individual episode and save the data of the web elements that we are interested in (fig. 3).

**Симпсоны** (1989– )  
Episode List

Season:  OR Year:

**Season 34**

**Habeas Tortoise** 25 Sep. 2022  
★ 6.5 (488) ☆ Rate  
After humiliating himself at a town meeting, Homer looks for acceptance in an internet group on the hunt for a missing tortoise and finds something much more sinister.

**One Angry Lisa** 2 Oct. 2022  
★ 6.3 (393) ☆ Rate  
Lisa gets called for jury duty, while Marge becomes obsessed with her exercise bike.

Figure 7. Episode list web page example

< Симпсоны

**S34.E3** < All episodes >

**Lisa the Boy Scout**

Episode aired Oct 9, 2022 · TV-PG · 22m

IMDb Rating: ★ 6.6/10 (528) YOUR RATING: ☆ Rate

+ Add to Watchlist

13 User reviews 1 Critic review

The show is hijacked by cyber terrorists.

**Directors** Mike B. Anderson (supervising director) · Timothy Bailey

**Writers** Matt Groening · James L. Brooks · Sam Simon >

**Stars** Dan Castellaneta (voice) · Julie Kavner (voice) · Nancy Cartwright (voice) >

IMDbPro See production, box office & company info

Figure 8. Episode web page example

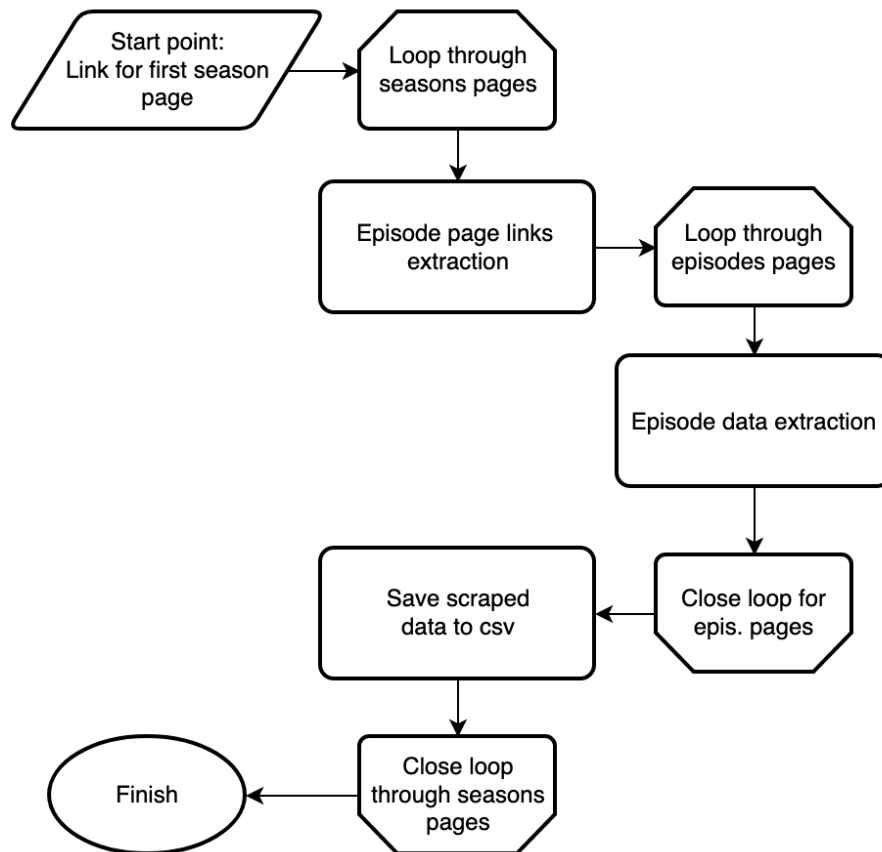


Figure 9. Scraping algorithm scheme for IMDb

After the completion of the developed algorithm, we receive a csv file with the desired data (fig. 4).

imdb_url	season	episode	episode_number_absolute	title	airdate	rating	reviews_amount
<a href="https://www.imdb.com/title/">https://www.imdb.com/title/</a>	1	1	1	Simpsons Roasting on an Open Fire	1994-06-23	81	7800
<a href="https://www.imdb.com/title/">https://www.imdb.com/title/</a>	1	2	2	Bart the Genius	1990-01-14	77	5200
<a href="https://www.imdb.com/title/">https://www.imdb.com/title/</a>	1	3	3	Homer's Odyssey	1990-01-21	73	4600
<a href="https://www.imdb.com/title/">https://www.imdb.com/title/</a>	1	4	4	There's No Disgrace Like Home	1990-01-28	77	4500
<a href="https://www.imdb.com/title/">https://www.imdb.com/title/</a>	1	5	5	Bart the General	1990-02-04	79	4900

Figure 10. IMDb Dataset example

## 3. Raw data cleaning

### 3.1 Fandom raw data

Both the Fandom scrapers retrieve, parse and clean data. The parsing is the first step of cleaning, because we need text data but we are given an HTML document. Parsing character properties is difficult because the website is a Wiki where everyone can change everything. Fortunately, the infobox structure is equal for all characters pages (but sometimes it was missing). As mentioned above, every property has to be handled differently. After getting a property value, in the majority of cases it's still full of other HTML tags. Some examples of this complexity are the following

- `<a><img></a><p><a><img></a></p>` for some double gender characters
- alias and main character fields containing `<p>`, `<small>`, `<sup>`, `<a>`, `<strong>`, `<i>` and `<br>`
- first mentioned property sometimes `<a></a>`, sometimes `<span>name</span>`, sometimes just `name`.

The cleaning at property level happens during the parsing; it happens after the parsing for rare cases, like the casting to integer value of the age.

At a tuple level the only cleaning is a page filter: are discarded all the pages representing Wiki users (that somehow appear in the list of characters), categories of characters pages (like the category of all [The Simpsons fathers](#)) and all the pages in the list of exceptions.

There are web pages on the fandom for episodes scheduled for release in 2023, but they have almost no information filled in and there is no corresponding data on the IMDb. Accordingly, we decided not to include these episodes in the database.

### 3.3 IMDb raw data

During the scraping data process from the web page, it was necessary to clean and transform the data for further convenient and efficient use. Almost every feature had to use appropriate transformations and cleanings. We will consider each of them further. Only such attributes as the title of the episode and its link remained unchanged.

- **Season and episode number:** on IMDb web resource season number and episode number contains in one single string. In order to get the season and episode number separately, the function of dividing the string by the occurrence of a specific character was used.

- **Air date:** on the page, the episode release date is written in a non-digital format, and by the scraping process, the date was read as text rather than a date. Therefore, data type and format changes to fully digital were required.
- **Rating:** the website indicates a rating on a scale from 0 to 10, but for more convenient further use it was decided to convert the rating to a scale from 0 to 100. Also during the scraping process, the data is stored in text format, so conversion to numeric is necessary.
- **Reviews amount:** to indicate the number of votes, the symbolic notation of thousands and millions, such as K and M, respectively, is used. This format is not acceptable for our task. A function was written to convert the text notation of the amount of votes into the corresponding numbers.

## 4.CSV fix

For both Fandom scrapers, the CSV file was not quoted in the url field and this caused issues while importing the data to the database. To fix the problem we forced the quoting of all strings, but another problem appeared: the database was reading "" as empty strings instead of null values.

We couldn't find a quick solution in Python, so we made the following find replace operations with the help of a text editor:

- find all ";" replace with ";", repeated until no matches are found
- find all ";"\n replace with ";\n" for the last CSV column

For the Fandom character scraper another find-replace operation is needed, because it was discovered printed Python None values as "None":

- find all "None" replace with (nothing)

All this operation could be made automatic without much effort, but since we needed to scrape data just a few times we decided to use our time in other sections of the project.

During the integration of two episode datasets collected from different websites, a problem with one episode was noticed. This episode consists of two separate parts and was recorded in the fandom dataset as a double episode; it had two episode numbers at once. While in the dataset from the IMDb, the same episode has only one episode number. Therefore, in order to improve the data integration process, we decided to manually solve this problem. We assigned a single episode number to the problematic episode in the fandom dataset and changed the subsequent absolute episode numbers for further episodes. After such a decision, two datasets from different resources have the same episodes absolute numbers and we can combine them according to this criterion.

## 5. Database

### 5.1 Database choice

Once the data has been properly retrieved, we move to the data modeling phase. To understand which characters make the episodes more enjoyable, we clearly need a relation between episodes and characters. This is a many-to-many relation, because an episode can have more than one main character and a character can be a main one in multiple episodes. Based on this information we initially decided to use a relational database. The relational paradigm is the best option because relational databases are the best at managing (a finite number of) many-to-many relationships thanks to an efficient indexing, even if it's based on text keys. Additionally, all the project members already knew SQL language.

Analyzing all possibilities, we confirmed by exclusion that our choice is correct:

- Key-Value Stores: not suitable for relations
- Column Family Stores: not suitable for relations
- Graph databases: we don't need rich relations, it's not optimal for many data join
- RDF databases and related: we don't need such expressive power, less performant
- The document database option should be discussed more in detail, because it's a good option: it can index object properties, and characters have multi level data. A critical example is the alias property, containing other ways that character is called in specific contexts.

To make real-world comparisons, we considered the biggest open source alternative for each database type: [MongoDB](#) for the document based database and [PostgreSQL](#) for the relational database. Both MongoDB and PostgreSQL support indexing on array fields, the first with multikey indexes, the second with GIN index type. Both can split a string based on a separator. We compared other features, and the two options always were on the same level, but in the end the relational model prevailed for a simple reason: familiarity.

The database we chose is [PostgreSQL](#) because it is the major open source option, it's powerful with a strong reputation for reliability, feature robustness, and performance; and not less important, it can be used with the [pgAdmin](#) user interface to make the user interaction (not automated) more friendly.

### 5.2 Database modeling

The entities we deal with are just two: character and episode, with a many-to-many relationship between them representing the main characters. The



previous statement is exactly the description of the ER diagram related to the model.

As mentioned in chapter 2.1 ([Data Sources](#)) we want to merge all the characters representing the same person into one, but we still need to create the main character relationship. As a study case let's consider the two characters [Ned Flanders](#) and [Ned Flandish](#), and an episode called Flandish episode that has Ned Flandish as main character. We considered two options:

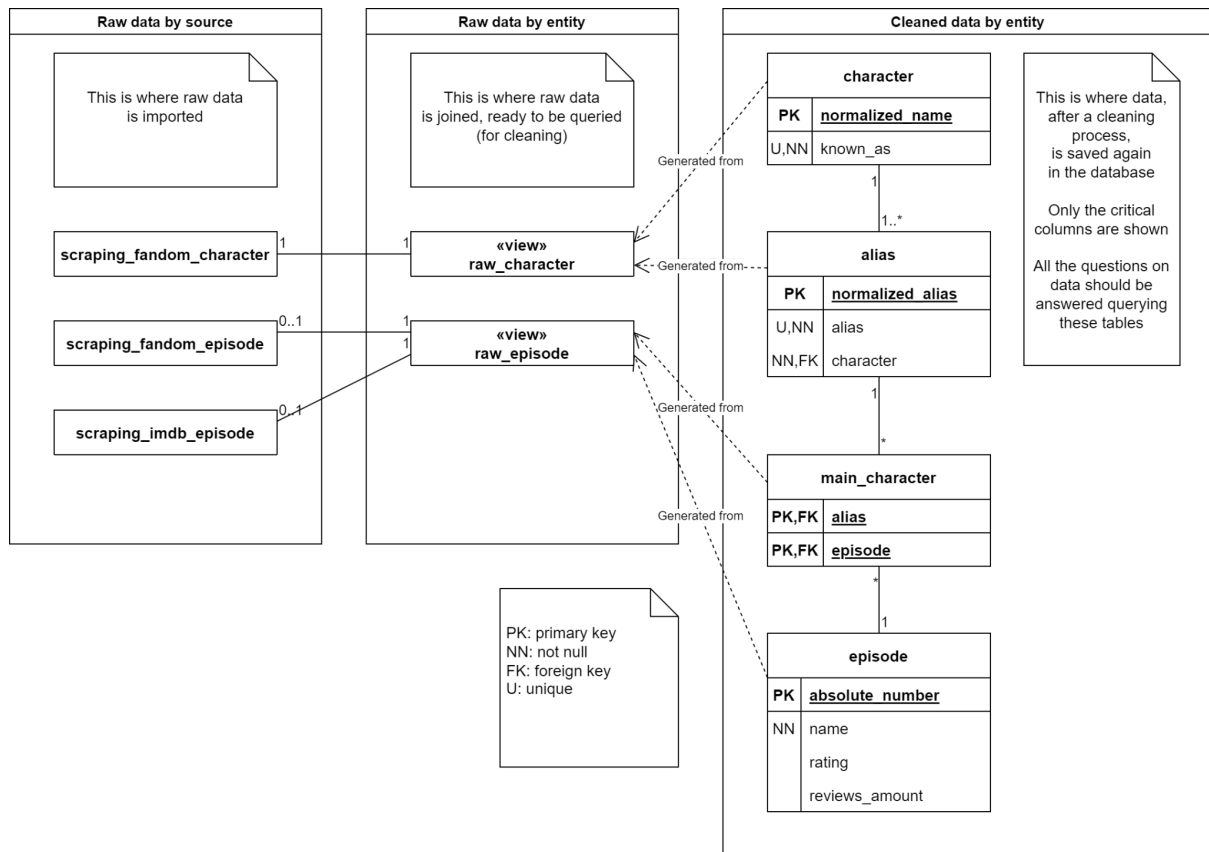
- Collapse also all the main character values of the episode data in the same way similar characters are collapsed. In the example the character table will include Ned Flanders and will not include Ned Flandish, and the main character of the Flandish episode will become Ned Flanders.
- Keep all the characters versions, introducing the concept of alias, the main character will reference alias, and an alias will reference a character. In the example the character table will include Ned Flanders and will not include Ned Flandish, the alias table will link Ned Flanders to the alias Ned Flandish, and the main character of the Flandish episode will remain Ned Flandish.

We chose the second option because the first loses lots of data, and the alias table could be useful for different objectives, like a better understanding of the existent versions of a character and statistics on the aliases instead of the characters.

This is how clean data is organized, but we decided to store and keep also the raw data that generated the clean data. This way, using more space, we can add a checkpoint to the automatization pipeline that is useful for development, debugging and production. We add a checkpoint in the sense that we make the process from raw data to clean data repeatable starting from the tables instead of csv files; this is useful for production. The checkpoint is also useful for production and debugging because we can make a relation of clean data to raw data, that allows us to compare the data (debugging) and to get additional information (production).

We now have tables for raw data and tables for clean data. We decided to add a middle step between raw and clean data. Raw data tables are one per source, but we need to work on entries for the cleaning process. The middle step is a series of views, one per entity (character and episode) where data from different sources is put side to side. This will facilitate comparisons. The character view is useless because there is only one character data source, but we realized that anyway for a better conceptual organization of the tables.

The following image represents the final database structure, that is the outcome of all previous considerations. For the creation of the clean data tables are used other temporary tables and views not shown for the sake of simplicity.



The “raw data by source” columns are the same as the respective csv files, with an added attribute `creation_time` that represents the date and time the entry was added to the database.

The “raw data by entity” columns are the same as the raw data by source columns, but whenever there is an attribute present in both, that column will be appended with the source name like `fandom_airstate` and `imdb_airstate`.

`character` table columns:

- `normalized_name`: the `known_as` property after a normalization process, used as primary key
- `known_as`: the original property, how the character is usually called
- `fandom_url`: the original property, used as foreign key to raw data
- `creation_time`: time at which the entry was generated from the cleaning process

`alias` table columns:

- `normalized_alias`: the `alias` property after a normalization process, used as primary key
- `alias`: a way the character character is called
- `character`: foreign key to `character` table primary key
- `creation_time`: time at which the entry was generated from the cleaning process

`main_character` table columns:

- `alias`: foreign key to alias table primary key
- `episode`: foreign key to episode table primary key

`episode` table columns:

- `episode_number_absolute`: absolute episode number, used as primary key
- `episode_number_relative`: relative episode number
- `season`: season number
- `title`: episode title
- `rating`: the original property, rating from 0 to 100
- `reviews_amount`: the original property, amount of reviewers that contributed to the value of rating
- main characters ???
- `fandom_url`: the original property, used as foreign key to fandom raw data
- `imdb_url`: the original property, used as foreign key to IMDb raw data
- `creation_time`: time at which the entry was generated from the cleaning process

## 6. Data cleaning and integration

Cleaning and preparing data during the data integration stage is one of the most important tasks. During the integration it will ensure that we merge the two datasets seamlessly. For integration and 100% string match, we need to make sure that one attribute is consistent, in particular primary keys but also the title of the episodes. String can cause lots of problems because of encoding, cases, special characters, etc. We built a cleaning algorithm for normalizing strings, which basically consists in finding and removing unwanted characters in a string and transforming all to lowercase. Some examples given in the table.

Original title	Cleaned title
Homer's Odyssey	homers odyssey
Itchy & Scratchy & Marge	itchy scratchy marge
King-Size Homer	king size homer

Therefore, all these manipulations were made in order to speed up and simplify the integration process.

In detail, the cleaning algorithm used in all the database is defined as follows:

1. [Unicode Normalization Form](#) NFKD
2. To lowercase
3. Trim initial and final space separator characters
4. Remove characters like brackets, apostrophes, punctuation, and other

The full list of space separator characters and to-remove characters can be found in the `readme.md` file of the `simpsons-db` repository.

### 6.1 Character table

The same character with different versions in raw data tables is ideally merged into one tuple in the `character` table, and all his versions and aliases are shown in a different table, the `alias` table. This process is a sort of deduplication.

The primary key is the normalized `known_as` property, because it can be generated, is source independent, and it's not null. A generated id is not suitable because it cannot be generated; this means that if the raw data changes and a new `character` table is built, Homer Simpson's id will probably be completely different, and this should be avoided. Additionally, this way the creation of the `alias` table is going to be easier instead of retrieving the generated character id. Many attributes are not considered because they are not useful for the objective, to see the full list of attributes go to the database modeling chapter.

Characters are merged through `known_as` string comparison. We do not compare the full name because it can be null and there are characters with very long surnames that are explicit only in the more classical version of the character (so no match will be found). An example is Marge Simpsons whose full name is Marjorie Jacqueline "Marge" Simpson.

We discarded in advance matching based on sound similarity because we need to match real different names like [Marge Simpson](#) and [Marnie Simpson](#) (her British counterpart). Between the remaining options Levenshtein and trigram matching, we chose the trigram matching because it worked better on a few manual tests. Unfortunately, the matching is not as easy as expected. To distinguish Apu ([Apu Nahasapeemapetilon](#)) from his brother's wife ([Ms. Nahasapeemapetilon](#)) a high threshold of 0.9 is needed; but setting this threshold prevents many matches, like the two Marge above that requires a threshold near to 0.6. To make [Homer Simpson](#) and [Other Homer Simpson](#) match, a threshold not higher than 0.7 is needed. Knowing that similar characters will not be lost but inserted in the alias list, we chose a trigram matching threshold of 0.6, achieving 519 merged characters, that is 10% of the raw characters.

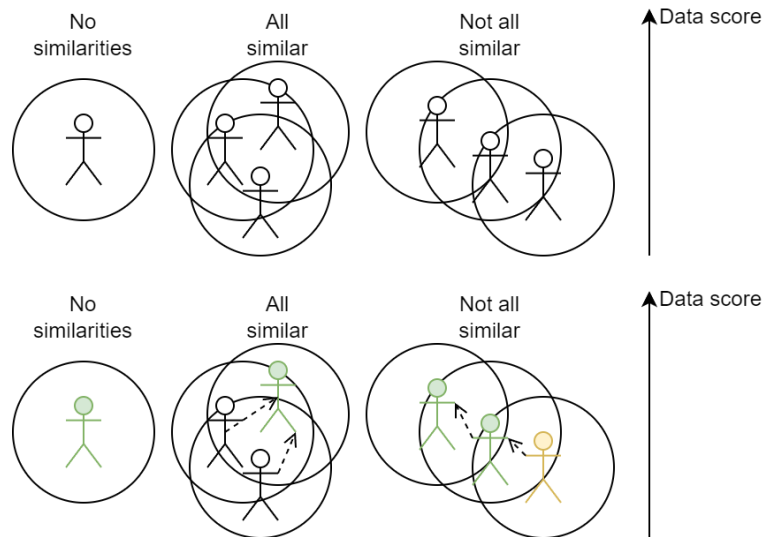
To solve this threshold problem, it seems that the surname should be removed, but unfortunately that is impossible now.

Now we can group, given one character, all similar characters. Two questions still remain:

- What is the criteria to choose the only real character in the group?
- Is the similarity transitive? For example, with a threshold of 0.6 Marnie Simpson is similar to Marge Simpson and is not similar to Other Marge Simpson, but with the same threshold Marge Simpson is similar to Marnie Simpson and Other Marge Simpson: should be the 3 characters in the same group?

The criteria used to choose the group representative is the amount of data available: it is reasonable to think that the standard version of a character is more used in the episodes with respect to aliases, then it is more known and popular, then there is more data about his/her on the wiki. The amount of data is a proxy for the original character version, but after a few tests, it seems to work very well. The amount of data is actually computed as the length of the majority of text properties concatenated.

The second question raises a more complex and equally critical problem. In SQL we know that all rows are treated equally, we cannot write a loop on the table rows, so we need to calculate the similarity on all tuples. Consider this example where the distance between characters is the similarity, and the circle represents the similarity (what is inside the circle of a character is considered similar with a fixed threshold).



In the second part the green characters are the representatives, the normal characters are not representatives and the yellow character is not represented; he/she has a group without representatives because the representative is in another group. We decided to answer yes to the transitivity question because answering no would have arised another question: “should the yellow character be alone or with the original representative?”. Then, after the group initial formation, we scan again all the characters to find the not represented ones and put them in the group of their fugitive representative. In order to do this, another temporary table not shown in the database is created and called `unmerged_character`. Note that we can be sure that each group cannot have more than one representative with this algorithm, so zero representatives is the only case to handle.

## 6.2 Alias table

We add data to the alias table after adding data to the character table because of the foreign key. The insertion of the aliases in the table is divided into 3 steps:

1. Add the characters as alias of themselves
2. Add the characters that are part of a group but are not representative, linking their name to the character that is representative
3. Add each of the aliases in the `alias` field of raw character data (split by comma) linking them to the clean character obtained considering the raw character name as alias name

In the steps 2 and 3, the existent aliases are discarded to ensure that one alias matches only one character.

The steps order is important because step 3 uses the information stored in the alias table in steps 1 and 2.

Since the character group data is queried multiple times to populate character and alias tables, a temporary table called “unmerged character” was created with the fields

- `group`: the the group representative
- `representative`: if the tuple is the group representative
- `known_as`: the original value
- `fandom_url`: the original value
- `data_score`: the amount of data as described in chapter 6.1 Character table

## 6.3 Episode table

In this section, we finally come to merging the two episode datasets into one clean data. Initially, our team planned to do the integration inside the database, where the raw data was loaded with minimal preparation. However, after conducting some research and analysis of means that could solve the problem of comparing titles name strings, it turned out to be too complex and would take a lot of time to implement.

Therefore, we chose the easier way, which is to merge the two datasets by episode title and check those titles that were not matched. This was done using a program written in Python. As a result of this, we received nearly 30 titles that I have no match for. At the same time, if you check these titles, it turns out that they are equal in meaning, but lexically arranged in a different way. Examples of these titles are given in the table.

Fandom title	IMDb title
bye bye nerdie	bye bye nerdy
the homer of seville	homer of seville
a serious flanders part one	a serious flanders
a serious flanders part two	a serious flanders part 2
gi doh	gi annoyed grunt

From there, we checked manually that the titles of the same episode from two different resources basically equal and and none of them is more preferable to choose.

Thus, such a manipulation allowed us to integrate two episode datasets from two resources, simply by selecting one of the two attributes at our choice. An attempt was also made to find the official names to decide which of the datasets has more

accurate information. However, as mentioned earlier in the report, we have no source of truth. Neither Wikipedia nor FOX TV has a consistent database with title names.

As a result of data integration, we obtained a table with the following attributes:

- `episode_number_absolute`: as `episode_number_absolute` (it was the join key) the same for Fandom dataset and IMDb dataset.
- `episode_number_relative`: as IMDb `episode_number_relative` (in Fandom ds there is no such a attribute)
- `season`: as IMDb `season` (Fandom ds has a lot of NULL value for this attribute)
- `title`: as Fandom `fandom_title` (we decided that it is more complete in terms of meaning)
- `rating`: as IMDb `rating` (in Fandom ds there is no such a attribute)
- `reviews_amount`: as IMDb `reviews_amount` (in Fandom ds there is no such a attribute)
- `fandom_url`: as Fandom `fandom_url`
- `imdb_url`: as IMDb `imdb_url`

## 6.4 Main characters table

The main characters table is the last to add data to, because it has foreign keys for the alias table and for the episode table.

To populate the table we add each of the main characters in the `main_characters` field of raw episode data (split by comma). We link the values to the episode table through episode absolute number; and we link the values to the alias table by normalizing the main character name and then checking the availability in the alias table. The match of main characters with an existing alias is 1007/1084 (92.9%), this means that 77 main characters (7.1%) are lost in this process. This is a good result.



## 7. Data quality

The quality dimensions that have been taken into account are: accuracy, completeness, currency and consistency.

### 7.1 Accuracy

According to the definition of accuracy, which tells us that this is the correct representation of the real world phenomenon. In our case, it is currently impossible to establish a source of truth for the collected data. We do not have reference values or validated sources where we can get extra data for comparison. Fandom and IMDb are not a hundred percent guarantor of the information presented on the site.

The only thing we can do at this stage of working with our datasets is to check the accuracy of the coincidence of episodes collected from two independent resources: Fandom and IMDb. At the same time this approach is more about consistency, therefore, we will provide this review in the next subsection. For now it is impossible to do this with regard to the dataset for the characters of the series.

### 7.2 Completeness

Completeness has been evaluated at the single attribute level for each dataset.

- For the dataset of **characters** obtained from the **Fandom** site, on a total of 5206 rows::

Attribute	Total Nan	% Nan
url	0	0.00
know_as	0	0.00
full name	245	4.71
image_url	444	8.53
age	4677	89.84
species	4997	95.99
gender	464	8.91
status	1842	35.38
fictional	0	0.00
alias	4771	91.64

hair color	1256	24.13
color	5005	96.14
birth country	4938	94.85
job	1845	35.44
first appearance	534	10.26
first mentioned	5042	96.85
voice	2636	50.63

The percentage of completeness of this dataset is 56.2%. This is not a concern because many attributes are not useful for our purpose.

- For the dataset of **episodes** obtained from the **Fandom** site, on a total of 753 rows:

Attribute	Total Nan	% Nan
url	0	0.00
title	0	0.00
image_url	14	1.86
season	141	18.73
episode_number_absolute	0	0.00
production_code	0	0.00
airdate	0	0.00
main_characters	315	41.83
written_by	0	0.00
directed_by	0	0.00

The percentage of completeness of this dataset is 93.7%. If we only had this dataset for the episodes, it would have been necessary to retrieve the missing values for the attribute "season".

In our case we already have all the necessary values thanks to the dataset from the imdb site.

- For the dataset of **episodes** obtained from the **IMDb** site, on a total of 739 rows:

Attribute	Total Nan	% Nan
imdb_url	0	0.00
title	0	0.00
season	0	0.00
episode_number_absolute	0	0.00
episode_number_relative	0	0.00
airdate	0	0.00
rating	0	0.00
reviews_amount	0	0.00

The percentage of completeness of this dataset is 100%

## 7.3 Currency

We know that currency measures how quickly data is updated, so, in our case the series "The Simpsons" is in continuous production and that's why the currency of our database will increase every week.

To try to ensure currency data should be collected at least weekly from the first scraping.

We also know that the web resources that were chosen for data collection are open and unofficial sources, and therefore the data may be updated with an unknown frequency. Therefore, to monitor all changes, it is necessary to scrape data more often, possibly daily.

Another interesting note we made related to the fandom website update. In the metadata of the web pages there is a property for the updates (HTTP last modifie). And the update date of each episode changes every day. In our opinion, this is related to the goal of increasing search relevance, in order to be in the top results of Google searches. The more often the site is updated and the more recent data is contained on the site, the higher Google promotes it.

## 7.4 Consistency

Consistency of the different representations of the same object of reality present in the database. In our case, we can check the consistency of the data formats that we integrate into one dataset. This applies to episode datasets. In the information for each of the data sets, all fields are presented in the corresponding

formats, which are mutually agreed upon. The comparison is given in the table. So the integration went smoothly in that regard.

Fandom Columns	Fandom DType	IMDb DType	IMDb Columns
url	string	string	imdb_url
title	string	string	title
season	float64	int64	season
episode_number_absolute	int64	int64	episode_number_absolute
airedate	date	date	airdate
main_characters	string	-	-
image_url	string	-	-
production_code	string	-	-
-	-	int64	rating
-	-	int64	episode_number_relative
-	-	int64	reviews_amount
written_by	string	-	-
directed_by	string	-	-

## 8.Data exploration

Our database is ready to provide answers to the questions raised in the work.

1. Which character(s) makes the episodes **more** enjoyable?

SQL query for this question:

```
SELECT
    character.normalized_name,
    character.known_as,
    character.fandom_url,
    COUNT(episode.episode_number_absolute) AS number_of_episodes,
    AVG(episode.rating) AS average_episode_rating,
    COUNT(episode.episode_number_absolute)/2 +
        AVG(episode.rating) AS popularity_score
FROM
    "character"
    INNER JOIN "alias"
        ON character.normalized_name = "alias"."character"
    INNER JOIN main_character
        ON "alias".normalized_alias = main_character."alias"
    INNER JOIN episode
        ON
            main_character.episode=episode.episode_number_absolute
GROUP BY character.normalized_name
HAVING COUNT(episode.episode_number_absolute) > 1
ORDER BY COUNT(episode.episode_number_absolute)/2
        + AVG(episode.rating) DESC
LIMIT 10;
```

Notes for the query: in order to give more complete and weighted scores to the characters, we include an additional rating - popularity score. We take into account the number of episodes, where the character appeared, and the average rating of these episodes, in order to reduce the effect of the number of episodes. We chose such a fuse that we divide the number of episodes by 2. So we got that not only the main characters are the most popular, but also secondary ones.

Result:

	Normalized name	Known as	Fandom url	Number of episodes	Average episode rating	Popularity score
1.	homer simpson	Homer Simpson	<a href="https://simpsons.fandom.com/wiki/Homer_Simpson">https://simpsons.fandom.com/wiki/Homer_Simpson</a>	242	76,30579	197,3058
2.	bart simpson	Bart Simpson	<a href="https://simpsons.fandom.co">https://simpsons.fandom.co</a>	156	76,58974	154,5897

			m/wiki/Bart_Simpson			
3.	lisa simpson	Lisa Simpson	<a href="https://simpsons.fandom.com/wiki/Lisa_Simpson">https://simpsons.fandom.com/wiki/Lisa_Simpson</a>	116	74,11207	132,1121
4.	marge simpson	Marge Simpson	<a href="https://simpsons.fandom.com/wiki/Marge_Simpson">https://simpsons.fandom.com/wiki/Marge_Simpson</a>	90	73,23333	118,2333
5.	charles montgomery burns	Charles Montgomery Burns	<a href="https://simpsons.fandom.com/wiki/Charles_Montgomery_Burns">https://simpsons.fandom.com/wiki/Charles_Montgomery_Burns</a>	30	77,03333	92,03333
6.	ned flanders	Ned Flanders	<a href="https://simpsons.fandom.com/wiki/Ned_Flanders">https://simpsons.fandom.com/wiki/Ned_Flanders</a>	30	76,2	91,2
7.	barney gumble	Barney Gumble	<a href="https://simpsons.fandom.com/wiki/Barney_Gumble">https://simpsons.fandom.com/wiki/Barney_Gumble</a>	4	84,75	86,75
8.	lenny leonard	Lenny Leonard	<a href="https://simpsons.fandom.com/wiki/Lenny_Leonard">https://simpsons.fandom.com/wiki/Lenny_Leonard</a>	3	85,33333	86,33333
9.	milhouse van houten	Milhouse Van Houten	<a href="https://simpsons.fandom.com/wiki/Milhouse_Van_Houten">https://simpsons.fandom.com/wiki/Milhouse_Van_Houten</a>	18	77	86
10.	seymour skinner	Seymour Skinner	<a href="https://simpsons.fandom.com/wiki/Seymour_Skinner">https://simpsons.fandom.com/wiki/Seymour_Skinner</a>	19	77	86

## 2. Which character(s) makes the episodes **less** enjoyable?

SQL query for this question:

```

SELECT
    character.normalized_name,
    character.known_as,
    character.fandom_url,
    COUNT(episode.episode_number_absolute) AS number_of_episodes,
    AVG(episode.rating) AS average_episode_rating,
    COUNT(episode.episode_number_absolute)/2 +
        AVG(episode.rating) AS popularity_score
FROM
    "character"
    INNER JOIN "alias"
        ON character.normalized_name = "alias"."character"
    INNER JOIN main_character
        ON "alias".normalized_alias = main_character."alias"
    INNER JOIN episode
        ON
            main_character.episode=episode.episode_number_absolute
GROUP BY character.normalized_name
HAVING COUNT(episode.episode_number_absolute) > 10
ORDER BY COUNT(episode.episode_number_absolute)/2 +

```

```

AVG(episode.rating) ASC
LIMIT 5;

```

Notes for the query: in this query, we used the condition that the character appeared in at least 10 episodes. That way we choose the lowest rated characters from the regular list, not the one characters that only appeared once or twice in the entire run of the series.

Result:

	Normalized name	Known as	Fandom url	Number of episodes	Average episode rating	Popularity score
1.	apu nahasapeemapetilon	Apu Nahasapeem apetilon	<a href="https://simpsons.fandom.com/wiki/Apu_Nahasapeemapetilon">https://simpsons.fandom.com/wiki/Apu_Nahasapeemapetilon</a>	13	74,84615	80,84615
2.	krusty the clown	Krusty the Clown	<a href="https://simpsons.fandom.com/wiki/Krusty_the_Clown">https://simpsons.fandom.com/wiki/Krusty_the_Clown</a>	15	74,13333	81,13333
3.	moe szyslak	Moe Szyslak	<a href="https://simpsons.fandom.com/wiki/Moe_Szyslak">https://simpsons.fandom.com/wiki/Moe_Szyslak</a>	13	75,15385	81,15385
4.	nelson muntz	Nelson Muntz	<a href="https://simpsons.fandom.com/wiki/Nelson_Muntz">https://simpsons.fandom.com/wiki/Nelson_Muntz</a>	12	75,16667	81,16667
5.	clancy wiggum	Clancy Wiggum	<a href="https://simpsons.fandom.com/wiki/Clancy_Wiggum">https://simpsons.fandom.com/wiki/Clancy_Wiggum</a>	11	77,54545	82,54545

### 3. What are the 10 top rated episodes?

SQL query for this question:

```

SELECT e.episode_number_absolute, e.season, e.title, e.rating
FROM episode AS e
ORDER BY e.rating DESC
LIMIT 10

```

Result:

Episode num abs	Season	Title	Rating
176	8	Homer's Enemy	93
128	6	Who Shot Mr. Burns? (Part One)	92

109	6	Treehouse of Horror V	92
155	8	You Only Move Twice	92
83	5	Cape Feare	91
71	4	Marge vs. the Monorail	91
76	4	Last Exit to Springfield	90
129	7	Who Shot Mr. Burns? (Part Two)	90
115	6	Homer the Great	90
135	7	King-Size Homer	90



## 9. Conclusion and future developments

In conclusion, we created a relational database for “The Simpsons” series from three data sources obtained from the scraping of Fandom and IMDb sites. Our database contains all the characters and episodes of the series, which have useful attributes such as rating and main characters, that allow you to answer many questions. In particular, our questions received a good answer (Chapter 8).

The database created could be a good starting point to create more advanced statistics, for example

- Check if there is a correlation between a character property (like gender) and popularity score
- Make a ranking for directors, writers and voice actors

The database may be updated and expanded whenever new episodes or characters are produced. We are happy that the workflow can be almost fully automated, the only exception is the handling of the two-in-one episode. For future development, full automation surely is interesting.