Ostbayerische Technische Hochschule Regensburg
Faculty for Mathematics and Computer Science

IMDB Software of Hollywood Actors and Actresses

Applied Data Science with Python

January 12, 2022

Author: Anna Breithaupt
MatNr. 3202199

Contents

1 Task 1

1.1 Short Description of the Problem

The goal of the project is to create a user-friendly application that provides information about the top 50 actors and actresses.
The information about the movies can be accessed through the IMDb page of the 50 most popular actors (Top 50 Popular Hollywood Actors and Actresses | IMDb, 2013).

1.2 General Approach

To implement this task, I want to scrape the required information from the website, sort it and then store it in a database.

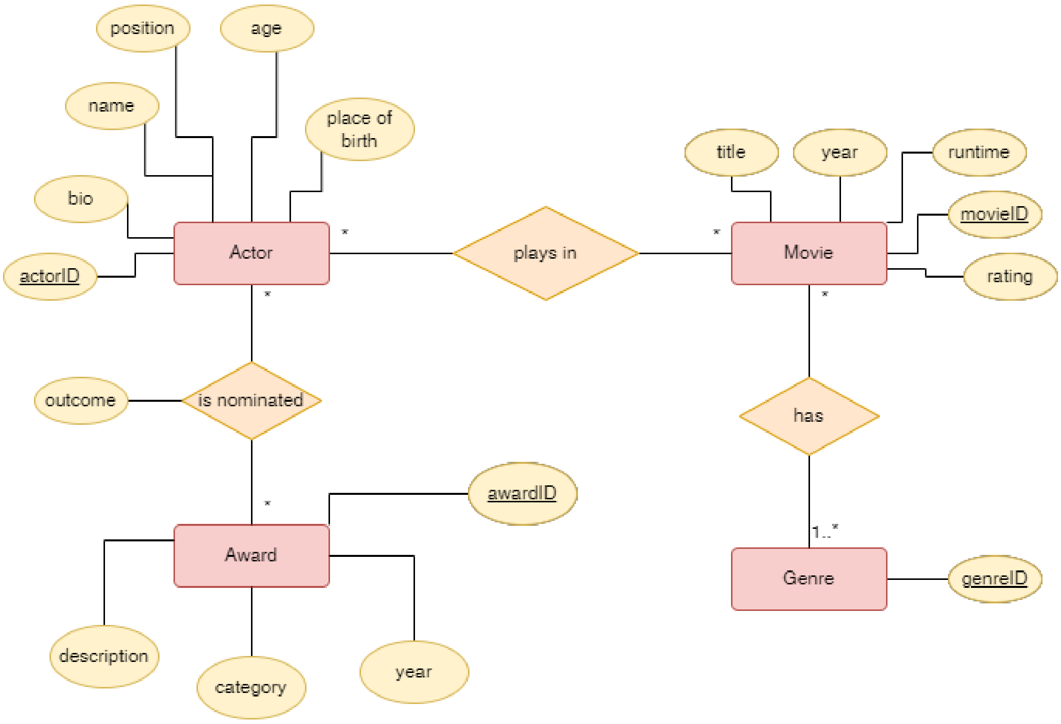


Figure 1: Entity Relationship Sketch

I considered the schema of the database for the beginning as seen in figure 1.
The stored data can then be used to retrieve specific information, statistics as well as to create graphs. The data obtained with this will be displayed to the user in a web application.

1.3 Tools

As development environment I will use **PyCharm**. Besides refactoring and debugging possibilities PyCharm also supports me in the organization and structuring of my project.
As versioning tool I will use **git**, whereby PyCharm also supports versioning with git.

1.3.1 Documentation

For documentation of the code I plan to use **Sphinx**, if more diagrams are needed for planning I will use **draw.io**. Additionally I will use **LaTeX** and **TeXstudio** to write the documentation.

1.3.2 Scraping, Storing and Processing the Data

To get the data from the website I plan to use **Beautiful Soup**. In order to find the right information I will also use the **Google Inspector Tool** provided by Chrome to locate the HTML tags and classes.
The data then will be stored in a database, for which I will use **MySQL**.
To process the data, I will access **Numpy** and **Pandas**. Since I also want to plot, I need **matplotlib.pyplot** as well.

1.3.3 User Interface

For the user to view the data in a descriptive manner, I will use a web interface. For this I utilize the web framework **Django**, **HTML5** and for the graphical elaboration **CSS** and **Bootstrap**.

1.4 Algorithms and Data Structures

1.4.1 Scraping and Storing the Data

The IMDb page lists the top 50 actors (Top 50 Popular Hollywood Actors and Actresses | IMDb, 2013). For each actor a detail page with further information is linked, where query parameters are used. To get the information I need, I thought of the following algorithm. After I have found the first actor, I search for the corresponding query parameter. This will also be used later in my database as ID for the actor. The query takes me to the details page to get all further information about the actor and his awards. On the details page all movies of the actor are listed. The movies are

linked again and I can repeat the procedure for the movies. So if the movie is not yet saved, it will be persisted in the database. The movie now can be linked to the actor. The procedure is repeated for all other actors subsequently.
I will mostly use the **find** and **find_all** function of Beautiful Soup.

The information I find with **JSON** formats can be stored in a **dictionary**. The rest of the information can be added to the dictionary and then simply stored in the database using a function. In addition to the lector materials I also used realpython.com as source of information (Beautiful Soup: Build a Web Scraper With Python | Real Python, 2021).

1.4.2 Processing und Evaluation

For the evaluation of the data I would like to use **diagrams**, among other things. To display the awards of the actors I would like to use a **histogram**. So the difference between nominations and wins can be shown well. A **scatter plot** can be used for the average rating of the films in the respective years. The genres of the movies could be shown well in a pie chart or in a **wordcloud mask**. For this I can use different methods from the matplotlib.pyplot.

For returning all actors or movies I will use **lists**, which will then be sorted with a **QuickSort algorithm**.

1.5 Modules

In order to think of suitable modules, I first thought about how to structure the project. For this I roughly fall back on the three layer model, where an application layer, a persistence layer and a presentation layer is implemented. Additionally I was inspired by Hitchhiker's Guide to Python(Structuring Your Project | Hitchhiker's Guide to Python).
In appendix A you can see a rough logic, how I plan to structure the project with corresponding modules.

2 Task 2

2.1 Current state of the project

In the last few weeks I was able to implement large parts of my project. Among other things, I created the databases where the scraped data will be stored. I was able to scrape the necessary information from the IMDb site [?] and store it in the database. Furthermore, I implemeted queries to retrieve the data from the database again and pass it to the web interface. I will explain my exact procedure on the following pages.

2.2 Database creation

2.2.1 Updated entity relationship model

I was able to implement the databases mostly as described in chapter 1.2. On figure 2 you can see the updated entity relationship model. The changes are highlighted with red color.

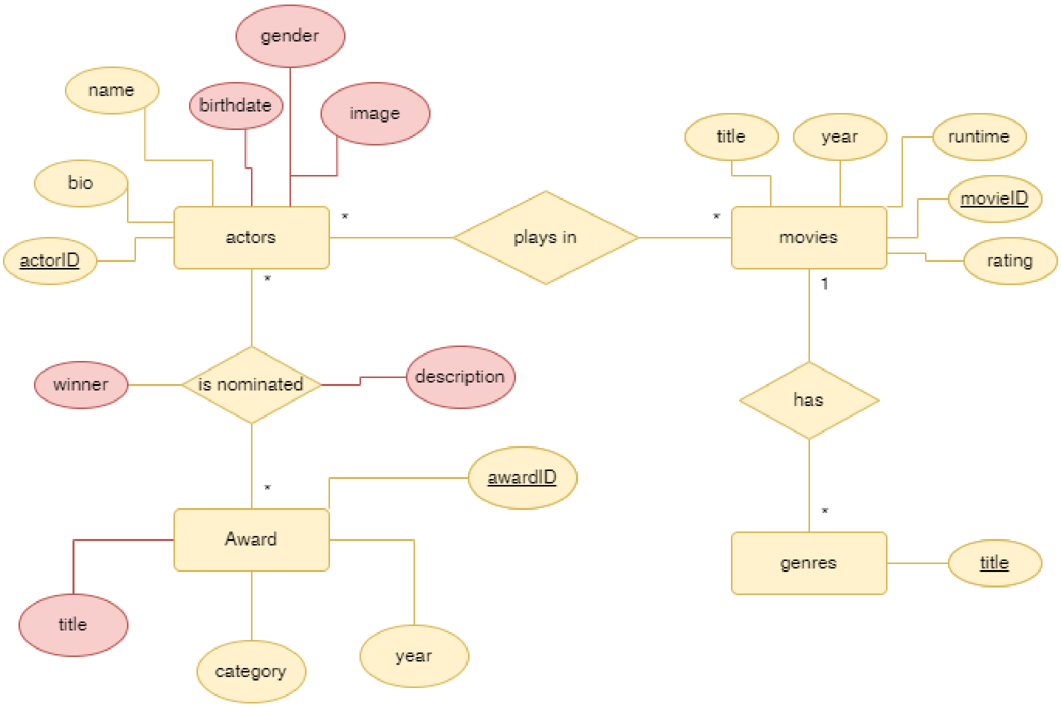
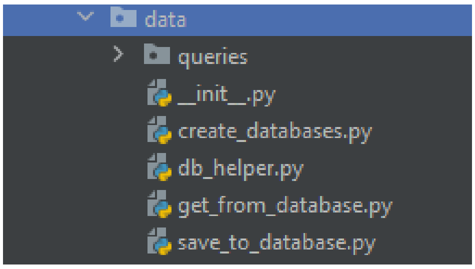


Figure 2: Updated ER diagram

Since an actor can be nominated several times for the same award in one year for different roles or categories, I have stored another attribute with the name *description* in the relationship between actor and award. Here, among other things, the nominated role or co-nominated actors can be stored.

I changed the relation between movie and genre from a many to many relation to a one to many relation, because the genre has no attributes except the title. This saves me an additional table that would reference both.
For the actor, I added image and gender as attributes and changed age to birthdate, which makes it easy to determine the age.

2.2.2 Creation of the database



data
queries
__init__.py
create_databases.py
db_helper.py
get_from_database.py
save_to_database.py

Figure 3: data layer structure

I started to create the database in the data layer of my project. As you can see in figure 3 I created a script called *create_databases.py*. The first thing I want to do is to establish a connection to a MySQL database. Since I will need the connection in the course of my project again and again, I have decided to outsource the function in a separate module *db_helper*. After a connection was established, a new schema will be created, if it does not exist yet. Then a new connection is established, this time directly to the schema.
Then the tables are to be generated. For this purpose, each table is first checked to see if it already exists in the database and dropped if necessary. This way I can avoid errors and make sure that there is no unwanted data in the table. Afterwards the table can be created.

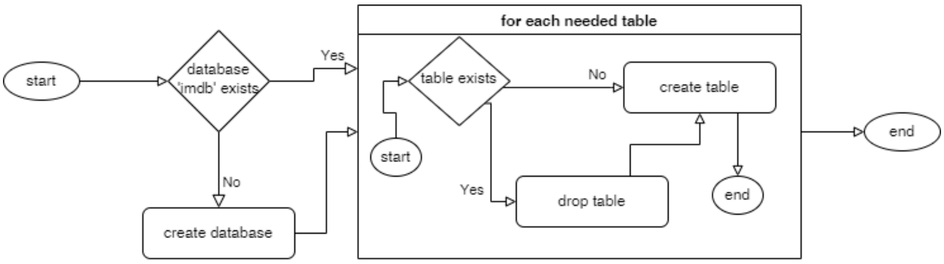
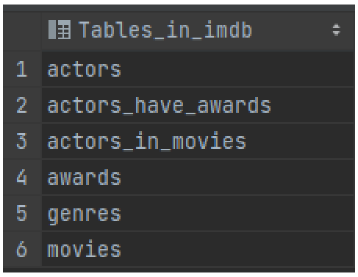


Figure 4: Flowchart create databases

In Figure 4 you can see the described chronology as a flow chart.

2.2.3 Result of the database creation

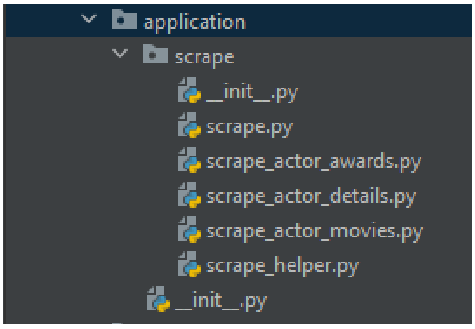
After the script was called, a database with six tables is created.
The *actors*, *awards* and the *movies* with the *genres* tables contain the main information, the others display the relationships.



```
   Tables_in_imdb
1  actors
2  actors_have_awards
3  actors_in_movies
4  awards
5  genres
6  movies
```

Figure 5: created databases

2.3 Scraping of the data



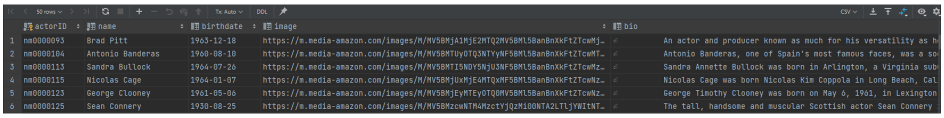
```
application
  scrape
    __init__.py
    scrape.py
    scrape_actor_awards.py
    scrape_actor_details.py
    scrape_actor_movies.py
    scrape_helper.py
  __init__.py
```

Figure 6: application layer structure

To get the data we want to store in our created database, we switch to the application layer of the project. The script responsible for scraping the data is *scrape.py*. Here we first use BeautifulSoup to fetch a JSON object with all actors. Among other things, the id of each actor is contained here. Via the JSON object I can iterate through all actors in the list. For the exact information I created more scripts, which should give the project clarity. So for each actor you can first scrape the awards and the movies next. In the appendix B, figure 28 you can see a flowchart of the general scraping process in *scrape.py*

2.3.1 Scrape Actor Information

To find out more about the actor, I used the URL *https://www.imdb.com/name/id* and *https://www.imdb.com/name/id/bio* . This turned out to be quite easy. I was able to retrieve some data using the JSON format. The remaining data was found by using html tags. I wrote the data into a dictionary and stored it in the database.
In the following figure 7 you can see some lines of the extracted data.



	actorID	name	birthdate	image	bio
1	nm0000093	Brad Pitt	1963-12-18	https://m.media-amazon.com/images/M/MV5BMjA1MjE2MTQ2MV5BMl5BanBnXkFtZTcwMj…	An actor and producer known as much for his versatility as h
2	nm0000104	Antonio Banderas	1960-08-10	https://m.media-amazon.com/images/M/MV5BMTUyOTQ3NTYyNF5BMl5BanBnXkFtZTcwMT…	Antonio Banderas, one of Spain's most famous faces, was a so
3	nm0000113	Sandra Bullock	1964-07-26	https://m.media-amazon.com/images/M/MV5BMTI5NDY5NjU3NF5BMl5BanBnXkFtZTcwMz…	Sandra Annette Bullock was born in Arlington, a Virginia sub
4	nm0000115	Nicolas Cage	1964-01-07	https://m.media-amazon.com/images/M/MV5BMjUxMjE4MTQxMF5BMl5BanBnXkFtZTcwNz…	Nicolas Cage was born Nicolas Kim Coppola in Long Beach, Cal
5	nm0000123	George Clooney	1961-05-06	https://m.media-amazon.com/images/M/MV5BMjEyMTEyOTQ0MV5BMl5BanBnXkFtZTcwNz…	George Timothy Clooney was born on May 6, 1961, in Lexington
6	nm0000125	Sean Connery	1930-08-25	https://m.media-amazon.com/images/M/MV5BMzcwNTM4MzctYjQzMi00NTA2LTljYWItNT…	The tall, handsome and muscular Scottish actor Sean Connery

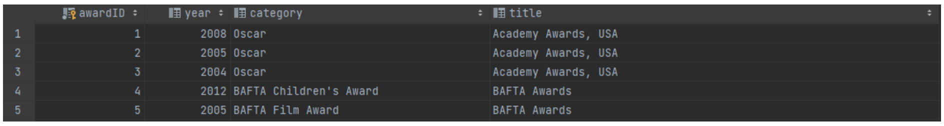
Figure 7: actors table

2.3.2 Scrape Actor Awards

Getting the awards was a bit more complicated. For this I used the URL *https://www.imdb.com/name/id/awards.*
The biggest difficulty was to include the awards where the actor was nominated several times for different roles. Since the table was then structured differently, the same attributes, such as year and outcome, remained empty. To get around this, I used the attributes of the predecessor in this case. My procedure can be seen in the following pseudo code.

```
soup = findSoup(url);
awardNames[ ] = soup.findList.findAll(h3);                    ▷ Find titles
awardList = [mockActor];                ▷ Create a mock award for the first entry
for all title in awardNames do
    award = {};
    award['name'] = title;
    award['actorID'] = actorID ▷ is the same for each award, since script gets called
for every actor
    tableRows = title.findNext(table).findAll(tr);
    for all row in tableRows do                        ▷ for every other attribute
        attribute = findAttribute(row);
        if attribute is empty then
            attribute = awardList[-1]['attribute'];          ▷ use previous if empty
        end if
        award['attribute'] = attribute;
    end for
    awardList.append[award];
end for
return awardList[1:];                                       ▷ remove mock award
```

If the award was not already stored in the database by a previous actor, the values had to be stored in the database. Finally, the actor and the award are linked in the relationship table.
The result can be seen in the following excerpt from the database, figure 8

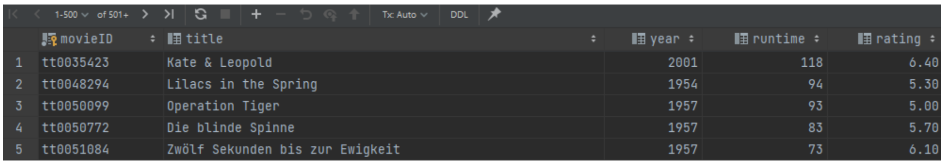


	awardID	year	category	title
1	1	2008	Oscar	Academy Awards, USA
2	2	2005	Oscar	Academy Awards, USA
3	3	2004	Oscar	Academy Awards, USA
4	4	2012	BAFTA Children's Award	BAFTA Awards
5	5	2005	BAFTA Film Award	BAFTA Awards

Figure 8: awards table

2.3.3 Scrape Movies

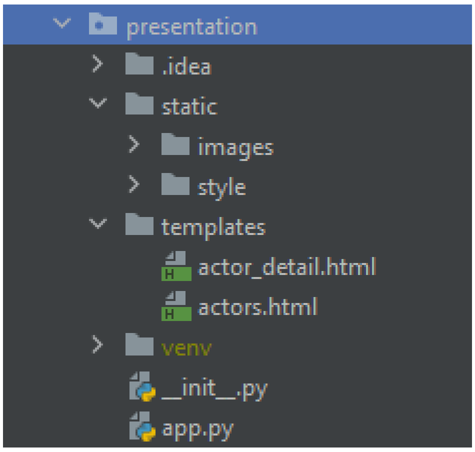
Scraping the films was done in a similar way as with the actors. The only challenge here was that some attributes were not specified for some films. To avoid NULL values in the database, I stored mock values in the database in this case. Finally the movies and the links to the actors were stored in the database again.
You can see the result in figure 9.



	movieID	title	year	runtime	rating
1	tt0035423	Kate & Leopold	2001	118	6.40
2	tt0048294	Lilacs in the Spring	1954	94	5.30
3	tt0050099	Operation Tiger	1957	93	5.00
4	tt0050772	Die blinde Spinne	1957	83	5.70
5	tt0051084	Zwölf Sekunden bis zur Ewigkeit	1957	73	6.10

Figure 9: movies table

2.4 User Interface

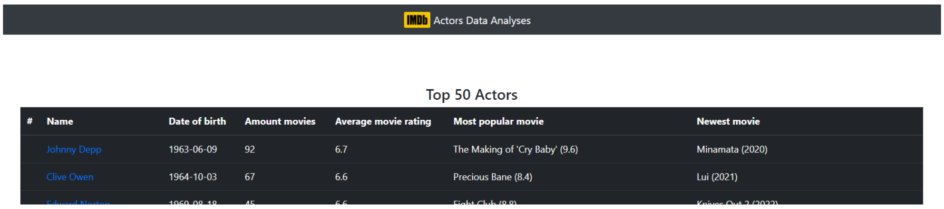


```
presentation
    .idea
    static
        images
        style
    templates
        actor_detail.html
        actors.html
    venv
    __init__.py
    app.py
```

Figure 10: presentation layer structure

The data was scraped from the website and persisted into the database. In the next step I want to display the data in a user-friendly way. Although I had initially considered using the Django web framework for the implementation of the web interface, I realized during the first implementation attempt that this would not benefit my already existing databases. With Django, databases are created automatically. I preferred to implement it in a way, so the data could be fetched from the database with an already fitting selection and then just pass the needed data to the web interface. Therefore I switched to the web framework Flask. For this I used the Flask documentation [?] and a tutorial [?]. In figure 10 you can see the adapted structure. So far I already created a template, on which all actors are displayed as a list. You can see the result so far in figure 11.

To get the required data, I created a module in the application layer that fetches the data from the database via the persistence layer and returns it in a dictionary.



Top 50 Actors

#	Name	Date of birth	Amount movies	Average movie rating	Most popular movie	Newest movie
	Johnny Depp	1963-06-09	92	6.7	The Making of 'Cry Baby' (9.6)	Minamata (2020)
	Clive Owen	1964-10-03	67	6.6	Precious Bane (8.4)	Lui (2021)
	Edward Norton	1969-08-18	45	6.6	Fight Club (8.8)	Knives Out 2 (2022)

Figure 11: web actors list

2.5 Outlook

So far, the data is scraped from the IMDb site, persisted in a database and displayed as a list on a web interface.
The next task is to evaluate and display the data for each actor as well. For this the data must be selected from the database and evaluated in the form of diagrams. In addition, html templates are to be created.

3 Task 3

3.1 Introduction

As already explained in section 1.1, the goal of the project is to provide a user-friendly application for obtaining and presenting information about the top 50 actors and actresses.
The information about the actors should be scraped from an IMDb page with the 50 most popular actors. [?]
Furthermore, functionalities are to be made available with which the following information can be retrieved:

- List of all available actors and actresses
- About the actor/actresses
- All time movie names and years
- Awards to actor/actresses in different years
- Movie genre of actor/actresses
- Average rating of their movies (overall and each year)
- Top 5 movies, their respective years and genre

The implementation of the project can be roughly divided into the following problems

1. Configuration and creation of the database
2. Scraping the information from the web page
3. Saving the data to the database
4. Implementation of a web interface
5. Getting and analyzing the data from the database

3.2 Modules, Datastructures and Tools

3.2.1 Modules

During the implementation of the project I was able to use some python modules.
To persist the data I used the python ***mysql*** module. When saving the awards I wanted to check which awards are already in the database, but I didn't have a primary key like the actors and awards, so I used the ***hashlib*** module to generate a unique primary key for each award.
For scraping the information I used ***BeautifulSoup*** and ***request*** to connect to the IMDb site. For the analysis of the data I used the ***json*** module. I also used it to save and read the database configuration.
To analyze the data and to generate the diagrams I used the modules ***pandas***, ***pylab***, ***matplotlib*** and ***wordcloud***.
To check if the diagrams have already been generated, I used ***os***.
For the display of the data I used ***flask***. I also use the ***threading*** module so that the user can still use the console even when the web page is displayed.

3.2.2 Datastructures

As data structures I used some ***dictionaries***. These were very helpful especially for storing and reading data from the database.
Also ***lists*** were often used to store data.
For actors, awards, movies and the connection to the database I created own ***classes***.
I also used ***collections*** to generate charts and ***regular expressions*** to format scrapped strings so that I could store them in the database.

3.2.3 Tools

As development environment I used ***PyCharm***, for the versioning of my project ***Git***.
For saving the data I used ***MySQL*** and for checking the databases ***MySQLWorkbench***.
For generating the documentation I used ***Sphinx***, which was able to convert my comments in the code, written in ***reStructured Text***, directly into a LaTeXdocument. For the documentation I also used LaTeXand ***draw.io*** for diagrams.
For the website I used ***CSS*** and ***HTML5*** as well as ***Bootstrap***.

3.3 Project design

As project structure I wanted to access the three layer architecture. Since in one case a module from the presentation layer directly accesses the application layer (see figure 12), this was slightly violated here. However, this does not affect the structure of the project.

3.3.1 Top level design

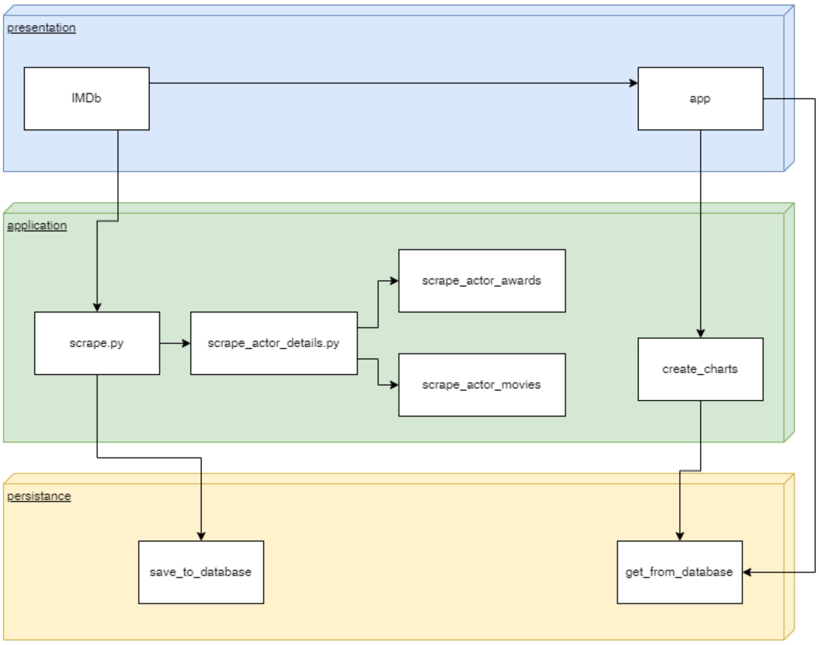


Figure 12: High level design of the project

In the figure 12 you can see well how the individual modules communicate with each other. There are additionall auxiliary libraries in my project that are called by the modules.

3.4 Project Structure

In the following picture you can see the final structure of the project with all modules.

```
|   constants.py
|   IMDb.py
|   README.md
|   __init__.py
|
+---data
|   |   db_config.json
|   |   db_config.py
|   |   db_connection.py
|   |   get_from_database.py
|   |   save_to_database.py
|   |   __init__.py
|   |
|   +---queries
|   |   |   create_queries.py
|   |   |   insert_queries.py
|   |   |   select_queries.py
|   |   |   __init__.py
|   |
|
+---presentation
|   |   app.py
|   |   __init__.py
|   |
|   +---static
|   |   +---images
|   |   |   |   IMDB_Logo_.png
|   |   |   +---award_charts
|   |   |   +---charts
|   |   |   \---movie_charts
|   |   |
|   |   \---style
|   |           style.css
|   |
|   +---templates
|   |       actors.html
|   |       actor_awards.html
|   |       actor_detail.html
|   |       actor_movies.html
|
\---application
    |   create_charts.py
    |   __init__.py
    |
    +---scrape
    |   |   scrape.py
    |   |   scrape_actor_awards.py
    |   |   scrape_actor_details.py
    |   |   scrape_actor_movies.py
    |   |   scrape_helper.py
    |   |   __init__.py
    |   |
    |
```

3.5 My Solution of the project

In the following section, my solutions for the individual problems of the project will be presented. First a general flow chart is used, which shows the order the steps are processed in.
Afterwards I go into detail about the individual functions. The modules and functions are linked to the attached documentation, so you can get details, parameters and the return value about the functions by clicking on them.

3.5.1 Configuration and creation of the database

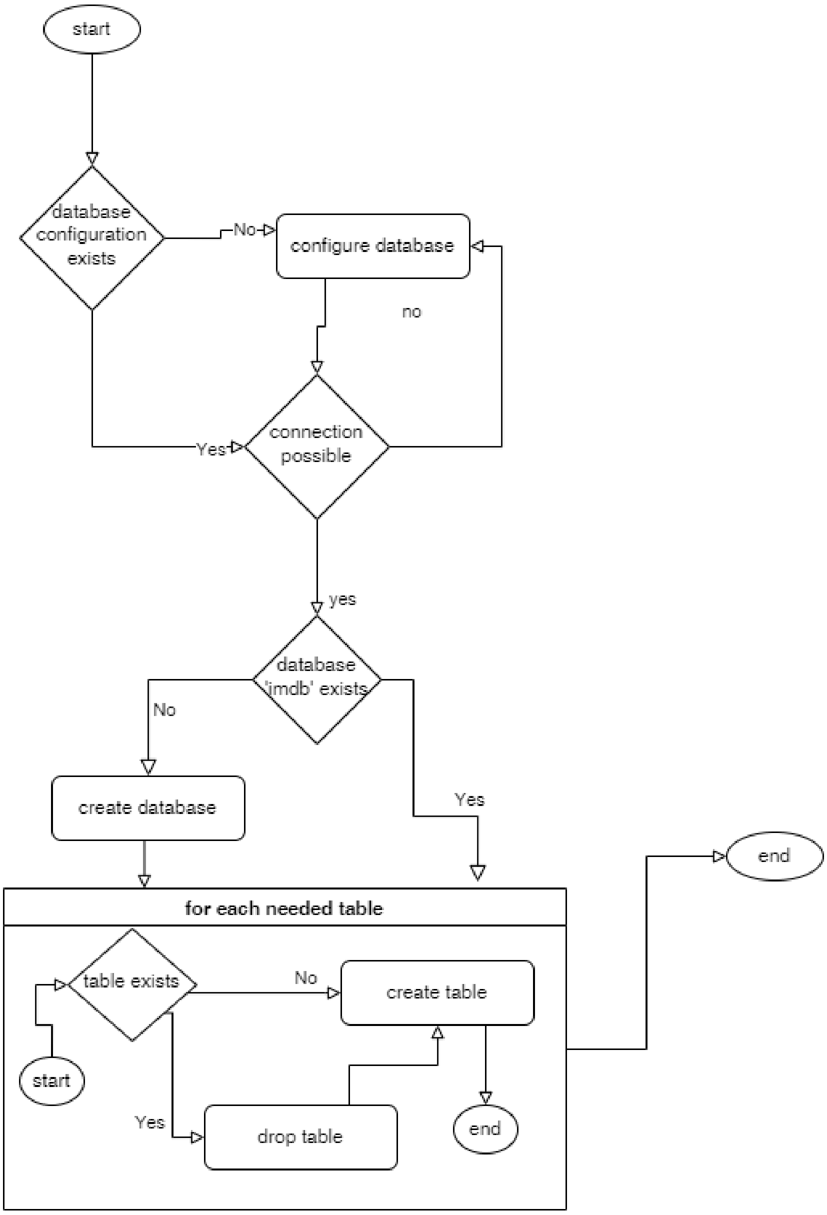


Figure 13: Flowchart database initialization

In Figure 13 you can see the final chronology for the creation of the database.
To create a database, you must first create a configuration for the database. For this, the module ***IMDb.py*** calls the function ***configure_database***. Here the user is asked for the required data for the configuration.
The received data will be processed with the function ***init_config*** in the module ***db_config*** in a configuration file *db_config.json*. Then a new ***Connection*** to the database is initialized and created and with the ***create_connection*** function is checked whether a connection to the database is possible. If necessary, the configuration must be adjusted.
If a connection is possible, ***create_db***. in ***IMDb.py*** is executed next. The ***Connection*** first checks with ***database_exists*** whether a database with this name already exists. Since tables can be deleted and data overwritten during the initialization of the database, the user is asked in this case whether this is intended. If necessary, the user can rename the database (***rename_database***.).
Afterwards the connection calls its function ***init_data_base*** and creates the required tables. For this the queries from the module ***create_queries*** are used.

3.5.2 Scraping the information from the web page

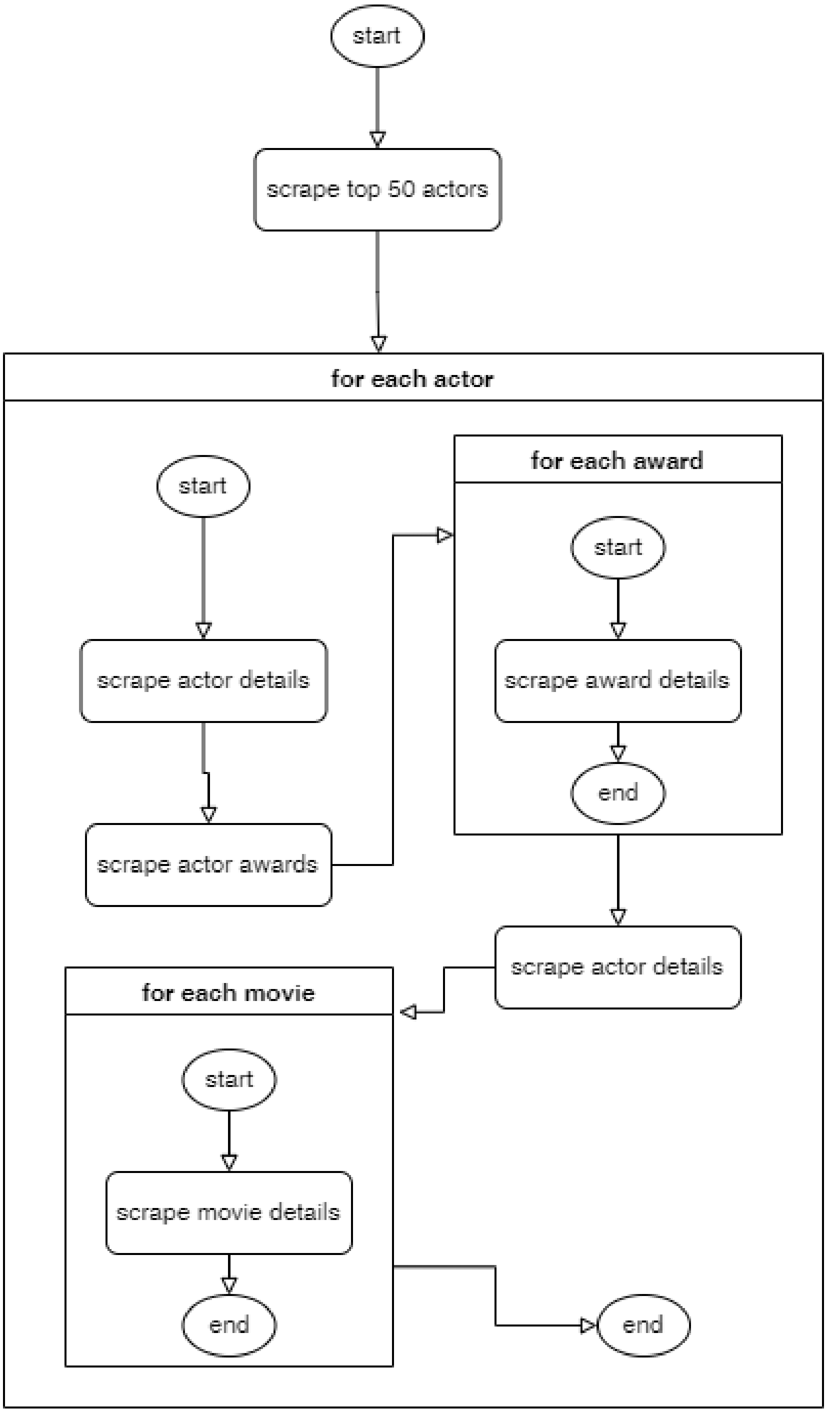


Figure 14: Flowchart Scraping Data

In Figure 14 you can see the final chronology for scraping the information.
For scraping from the web page, the function ***scrape_information*** from the module ***IMDb.py*** the script ***scrape.py*** is called. From here, the information about the top 50 actors is scraped. For each actor, a new ***Actor*** object is created. In its init function all awards of the actor are searched. ***scrape_all_awards_of_actor*** is called in ***scrape_actor_awards.py*** . For each award, a new ***award*** object is created. The

return value of ***scrape_all_awards_of_actor*** is then stored in the ***Actor*** object. Similar procedure is found again for movies. First, ***scrape_all_movies_of_actor*** is called in ***scrape_actor_movies.py***. Individual ***Movie*** Objects created, which in turn scrape all important information in their init function. The ***Actor*** is returned a list of movies.

3.5.3 Saving the data to the database

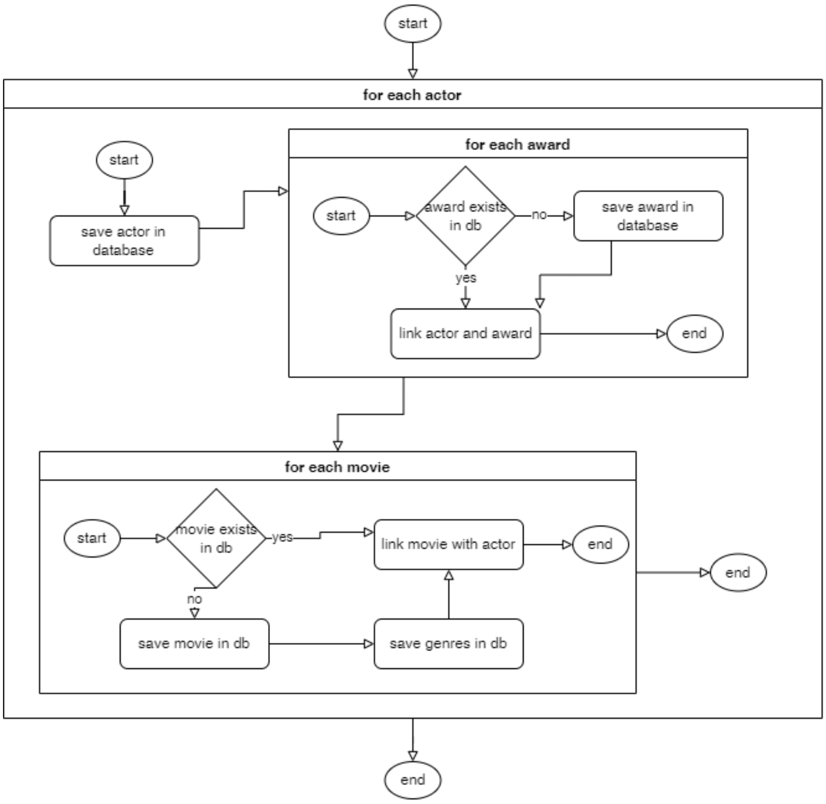


Figure 15: Flowchart Saving Data

In Figure 15 you can see the final chronology for saving the data in the database. After the information is found in ***scrape.py*** this script calls the ***persist_information*** function in ***save_to_database.py*** . After a new ***Connection*** is established, the ***Actor*** is saved first. To save, the appropriate information of the actor is obtained with its method ***get_actor_information*** as a dict. Afterwards this is saved from the ***Connection*** with ***save_value*** into the database. Afterwards a similar procedure follows for the awards. Here it is checked with ***entry_exists*** whether the Award was already stored. Then the connection to the actor is saved. The same procedure follows for the movies. The genres of the film are also persisted.

3.5.4 Implementation of a web interface

Since flask could do some of the work for me, my main task in implementing the web interface was to create html templates.
I decided to use four different pages. In **actors.html** all actors are shown. **actor_detail.html** should be able to show more information, like the bio, of the actor. For the movies of an actor I created **actor_movies.html** and information about the awards can be found on **actor_awards.html**.
From the ***app.py*** module the respective templates can be rendered. The application is started by the ***start_web_app*** function in the ***IMDb.py*** module.

3.5.5 Getting and analyzing the data from the database

For every html template different information needs to be called from the databases. When charts are created, they are stored in a static folder. Everytime a chart should be created, it is first checked if the chart already exists.

actors.html

actors.html needs a list of all actors and information about them. For this the function ***get_all_actors*** is called in the module ***get_from_database.py***, which returns all actors as a list of dicts.

actor_detail.html

To get the details about an actor, ***get_single_actor*** is called in ***get_from_database.py***.
In addition, a chart is created that compares the average number of nominations and award wins as well as the amount of movies of all actors with the current one. For this purpose ***avg_awards_movies_bar*** in ***create_charts.py*** is called.

actor_awards.html

To get the awards of an actor, ***get_awards_of*** is called.
After that two plots about the awards over the years are returned. For this purpose ***awards_plot*** and ***avg_awards_plot*** are called.

actor_movies.html

To get the movies of an actor, ***get_movies_of*** is called.
After that two charts about the genres of the actor and two plots of his average movie rating per year are generated with ***genres_pie_chart***, ***genres_wordcloud_chart*** and ***movie_rating_per_year***

3.6 Result

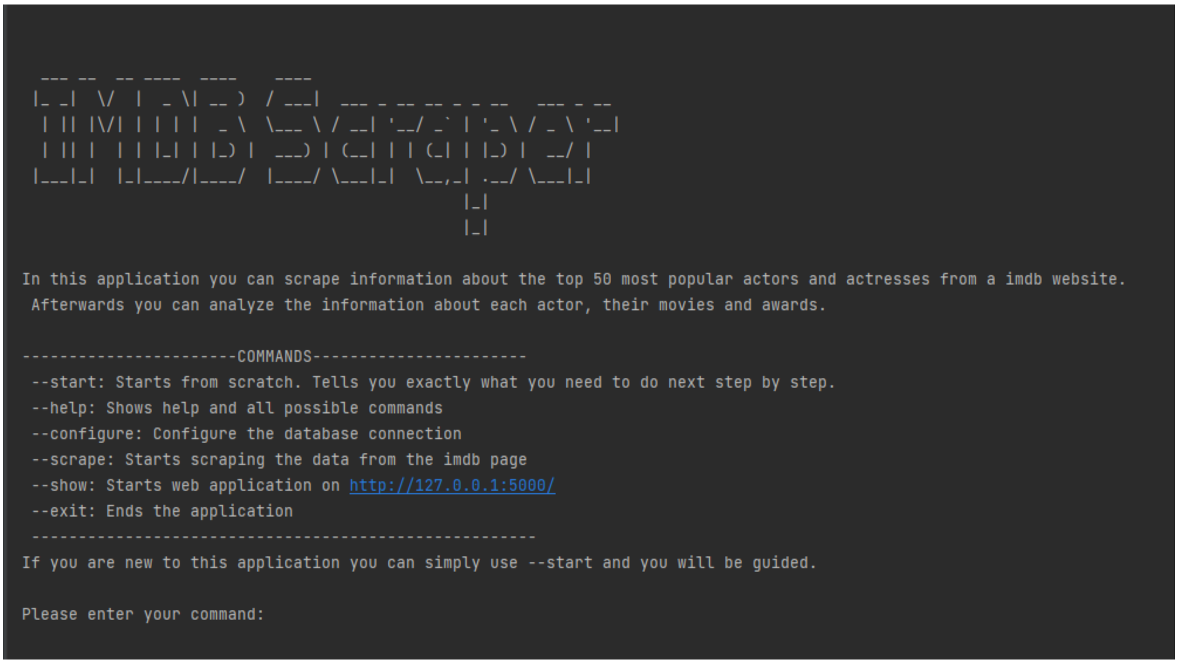
3.6.1 Scraped data in the database

For examples of the scraped data you can still have a look at 7, 8 and 9 since the result did not change much since the last task.

3.6.2 User Interface

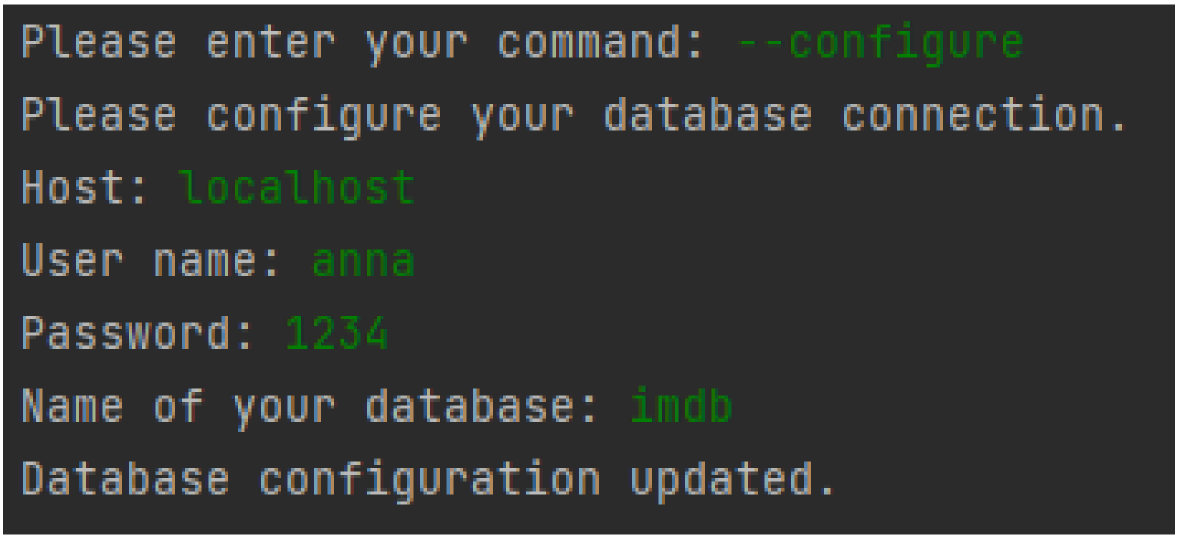
Command line

When you start the application, you have the possibility to execute different commands. The following illustrations show the execution of different commands in the command line.



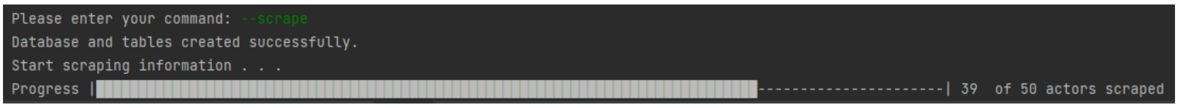
```
  ___ __  __ ____  ____    ____
 |_ _|  \/  |  _ \| __ )  / ___|  ___ _ __ __ _ _ __   ___ _ __
  | || |\/| | | | |  _ \  \___ \ / __| '__/ _` | '_ \ / _ \ '__|
  | || |  | | |_| | |_) |  ___) | (__| | | (_| | |_) |  __/ |
 |___|_|  |_|____/|____/  |____/ \___|_|  \__,_| .__/ \___|_|
                                               |_|
                                               |_|

In this application you can scrape information about the top 50 most popular actors and actresses from a imdb website.
 Afterwards you can analyze the information about each actor, their movies and awards.

----------------------COMMANDS----------------------
 --start: Starts from scratch. Tells you exactly what you need to do next step by step.
 --help: Shows help and all possible commands
 --configure: Configure the database connection
 --scrape: Starts scraping the data from the imdb page
 --show: Starts web application on http://127.0.0.1:5000/
 --exit: Ends the application
 ------------------------------------------------------
If you are new to this application you can simply use --start and you will be guided.

Please enter your command:
```

Figure 16: Command line after start of IMDb.py



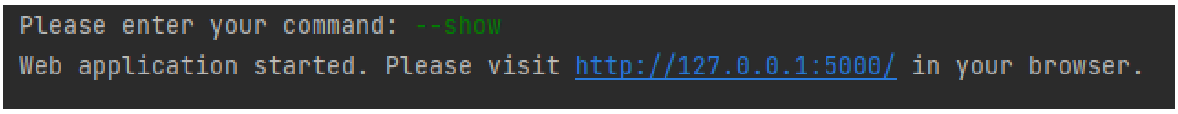
```
Please enter your command: --configure
Please configure your database connection.
Host: localhost
User name: anna
Password: 1234
Name of your database: imdb
Database configuration updated.
```

Figure 17: Command line after command –configure



```
Please enter your command: --scrape
Database and tables created successfully.
Start scraping information . . .
Progress |██████████████████████████████████████████████████████████████████████████████----------------------| 39  of 50 actors scraped
```

Figure 18: Command line after command –scrape



```
Please enter your command: --show
Web application started. Please visit http://127.0.0.1:5000/ in your browser.
```

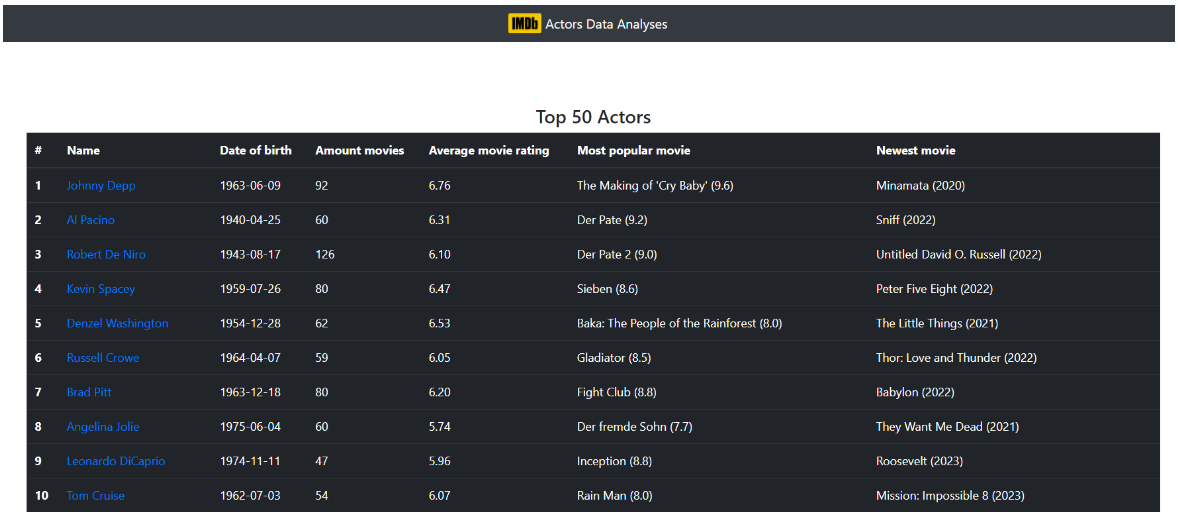
Figure 19: Command line after command –show

Web interface

When you enter the –show command, you can follow the link to http://127.0.0.1:5000/ in your browser. Here all important details are listet in the webinterface. When we take a look at section 3.1 we see the requirements of the project.

My solutions for each task:

- List of all available actors and actresses

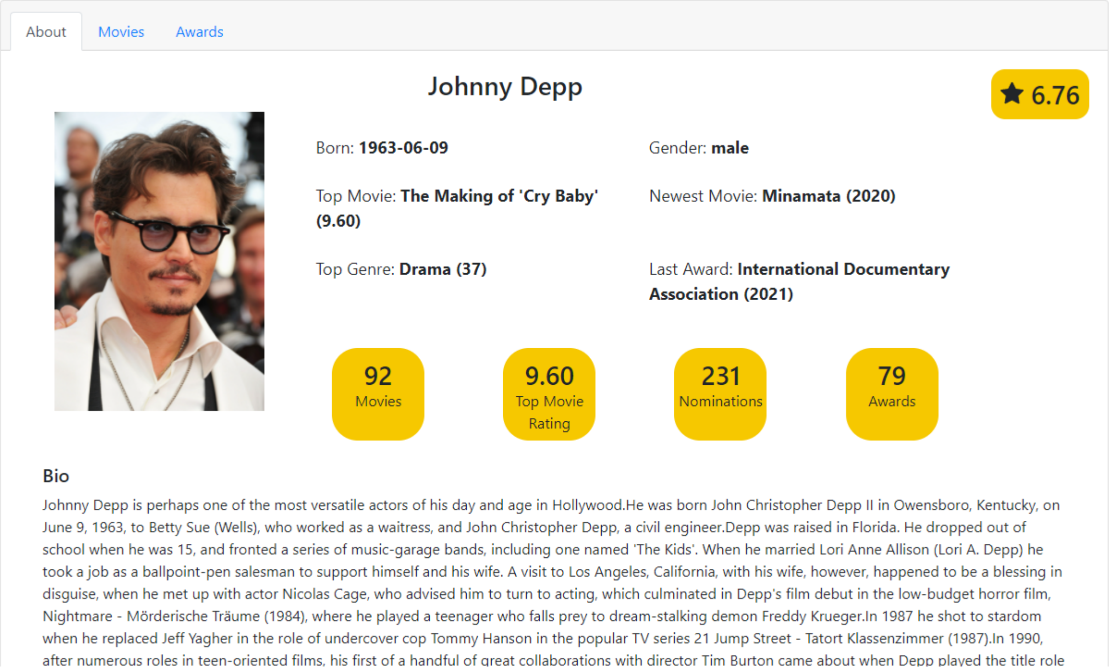


Top 50 Actors

#	Name	Date of birth	Amount movies	Average movie rating	Most popular movie	Newest movie
1	Johnny Depp	1963-06-09	92	6.76	The Making of 'Cry Baby' (9.6)	Minamata (2020)
2	Al Pacino	1940-04-25	60	6.31	Der Pate (9.2)	Sniff (2022)
3	Robert De Niro	1943-08-17	126	6.10	Der Pate 2 (9.0)	Untitled David O. Russell (2022)
4	Kevin Spacey	1959-07-26	80	6.47	Sieben (8.6)	Peter Five Eight (2022)
5	Denzel Washington	1954-12-28	62	6.53	Baka: The People of the Rainforest (8.0)	The Little Things (2021)
6	Russell Crowe	1964-04-07	59	6.05	Gladiator (8.5)	Thor: Love and Thunder (2022)
7	Brad Pitt	1963-12-18	80	6.20	Fight Club (8.8)	Babylon (2022)
8	Angelina Jolie	1975-06-04	60	5.74	Der fremde Sohn (7.7)	They Want Me Dead (2021)
9	Leonardo DiCaprio	1974-11-11	47	5.96	Inception (8.8)	Roosevelt (2023)
10	Tom Cruise	1962-07-03	54	6.07	Rain Man (8.0)	Mission: Impossible 8 (2023)

Figure 20: List of all actors

- About the actor/actresses



Johnny Depp

Born: **1963-06-09**

Gender: **male**

Top Movie: **The Making of 'Cry Baby' (9.60)**

Newest Movie: **Minamata (2020)**

Top Genre: **Drama (37)**

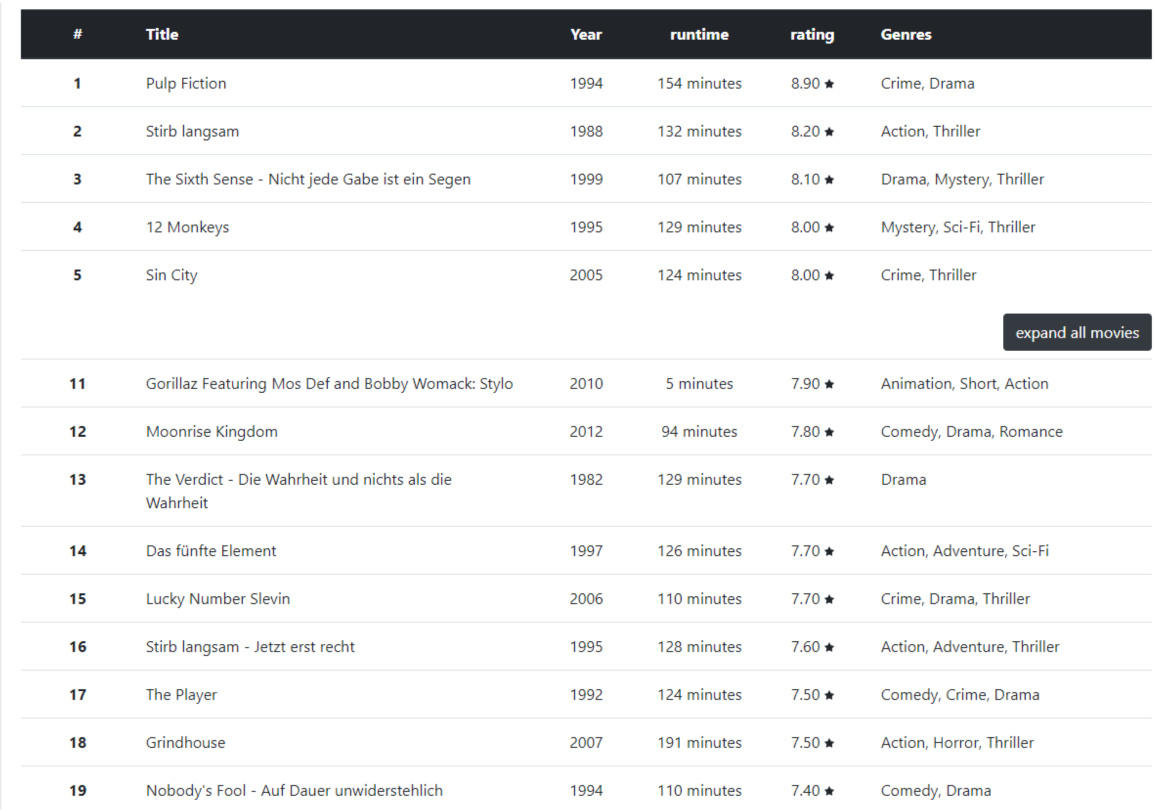
Last Award: **International Documentary Association (2021)**

Bio

Johnny Depp is perhaps one of the most versatile actors of his day and age in Hollywood.He was born John Christopher Depp II in Owensboro, Kentucky, on June 9, 1963, to Betty Sue (Wells), who worked as a waitress, and John Christopher Depp, a civil engineer.Depp was raised in Florida. He dropped out of school when he was 15, and fronted a series of music-garage bands, including one named 'The Kids'. When he married Lori Anne Allison (Lori A. Depp) he took a job as a ballpoint-pen salesman to support himself and his wife. A visit to Los Angeles, California, with his wife, however, happened to be a blessing in disguise, when he met up with actor Nicolas Cage, who advised him to turn to acting, which culminated in Depp's film debut in the low-budget horror film, Nightmare - Mörderische Träume (1984), where he played a teenager who falls prey to dream-stalking demon Freddy Krueger.In 1987 he shot to stardom when he replaced Jeff Yagher in the role of undercover cop Tommy Hanson in the popular TV series 21 Jump Street - Tatort Klassenzimmer (1987).In 1990, after numerous roles in teen-oriented films, his first of a handful of great collaborations with director Tim Burton came about when Depp played the title role

Figure 21: About actor

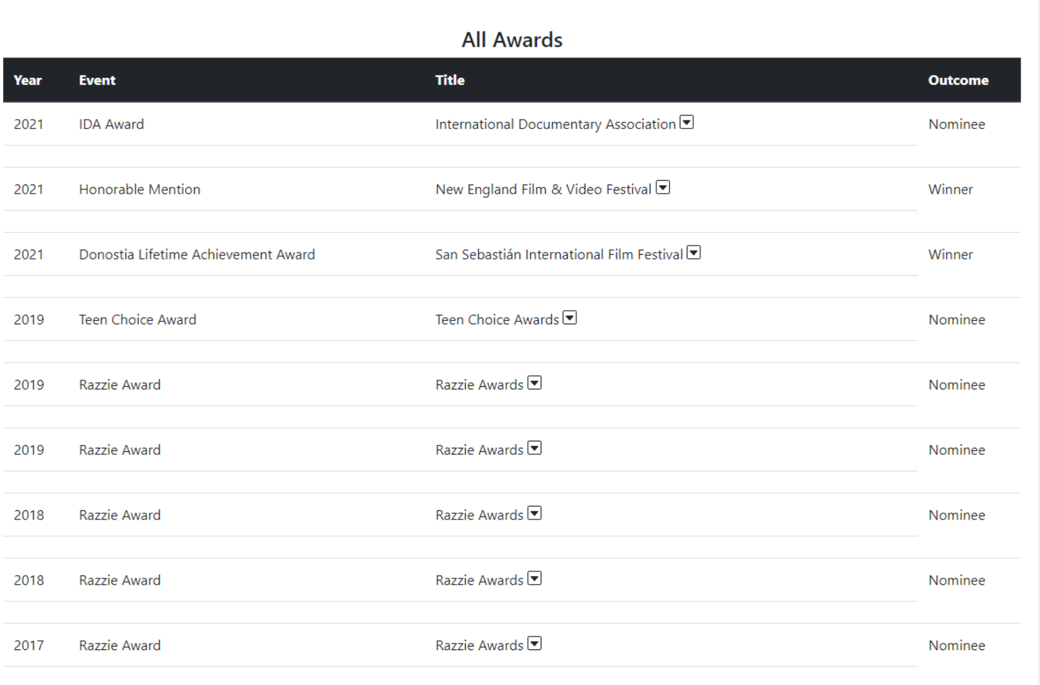
- All time movie names and years



#	Title	Year	runtime	rating	Genres
1	Pulp Fiction	1994	154 minutes	8.90 ★	Crime, Drama
2	Stirb langsam	1988	132 minutes	8.20 ★	Action, Thriller
3	The Sixth Sense - Nicht jede Gabe ist ein Segen	1999	107 minutes	8.10 ★	Drama, Mystery, Thriller
4	12 Monkeys	1995	129 minutes	8.00 ★	Mystery, Sci-Fi, Thriller
5	Sin City	2005	124 minutes	8.00 ★	Crime, Thriller
11	Gorillaz Featuring Mos Def and Bobby Womack: Stylo	2010	5 minutes	7.90 ★	Animation, Short, Action
12	Moonrise Kingdom	2012	94 minutes	7.80 ★	Comedy, Drama, Romance
13	The Verdict - Die Wahrheit und nichts als die Wahrheit	1982	129 minutes	7.70 ★	Drama
14	Das fünfte Element	1997	126 minutes	7.70 ★	Action, Adventure, Sci-Fi
15	Lucky Number Slevin	2006	110 minutes	7.70 ★	Crime, Drama, Thriller
16	Stirb langsam - Jetzt erst recht	1995	128 minutes	7.60 ★	Action, Adventure, Thriller
17	The Player	1992	124 minutes	7.50 ★	Comedy, Crime, Drama
18	Grindhouse	2007	191 minutes	7.50 ★	Action, Horror, Thriller
19	Nobody's Fool - Auf Dauer unwiderstehlich	1994	110 minutes	7.40 ★	Comedy, Drama

Figure 22: All movies of an actor

- Awards to actor/actresses in different years



All Awards

Year	Event	Title	Outcome
2021	IDA Award	International Documentary Association	Nominee
2021	Honorable Mention	New England Film & Video Festival	Winner
2021	Donostia Lifetime Achievement Award	San Sebastián International Film Festival	Winner
2019	Teen Choice Award	Teen Choice Awards	Nominee
2019	Razzie Award	Razzie Awards	Nominee
2019	Razzie Award	Razzie Awards	Nominee
2018	Razzie Award	Razzie Awards	Nominee
2018	Razzie Award	Razzie Awards	Nominee
2017	Razzie Award	Razzie Awards	Nominee

Figure 23: All awards of an actor

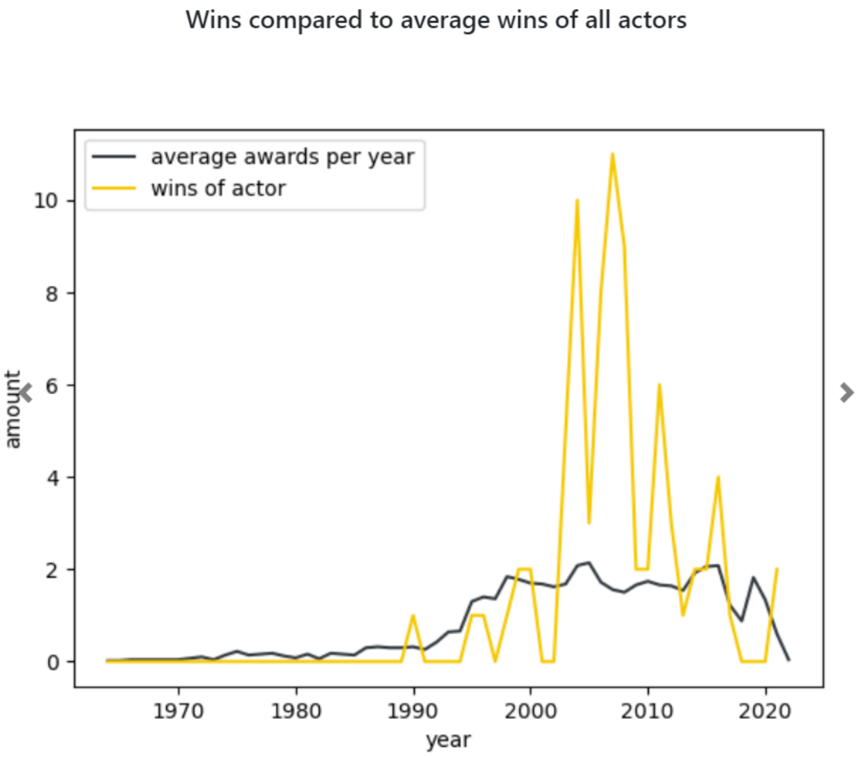
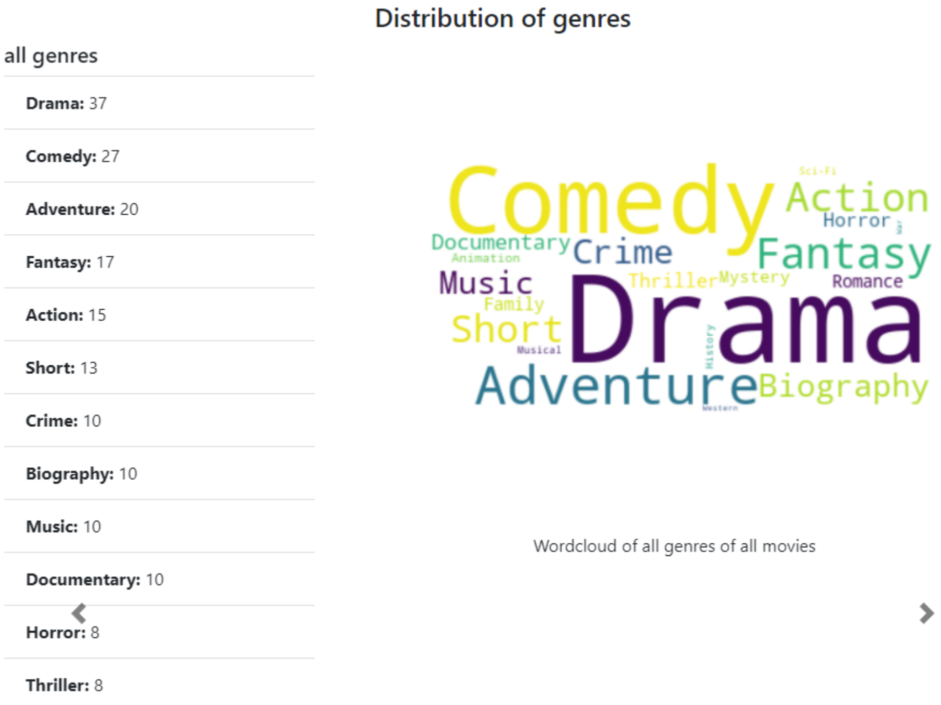


Figure 24: Awards of each year

- Movie genre of actor/actresses



Distribution of genres

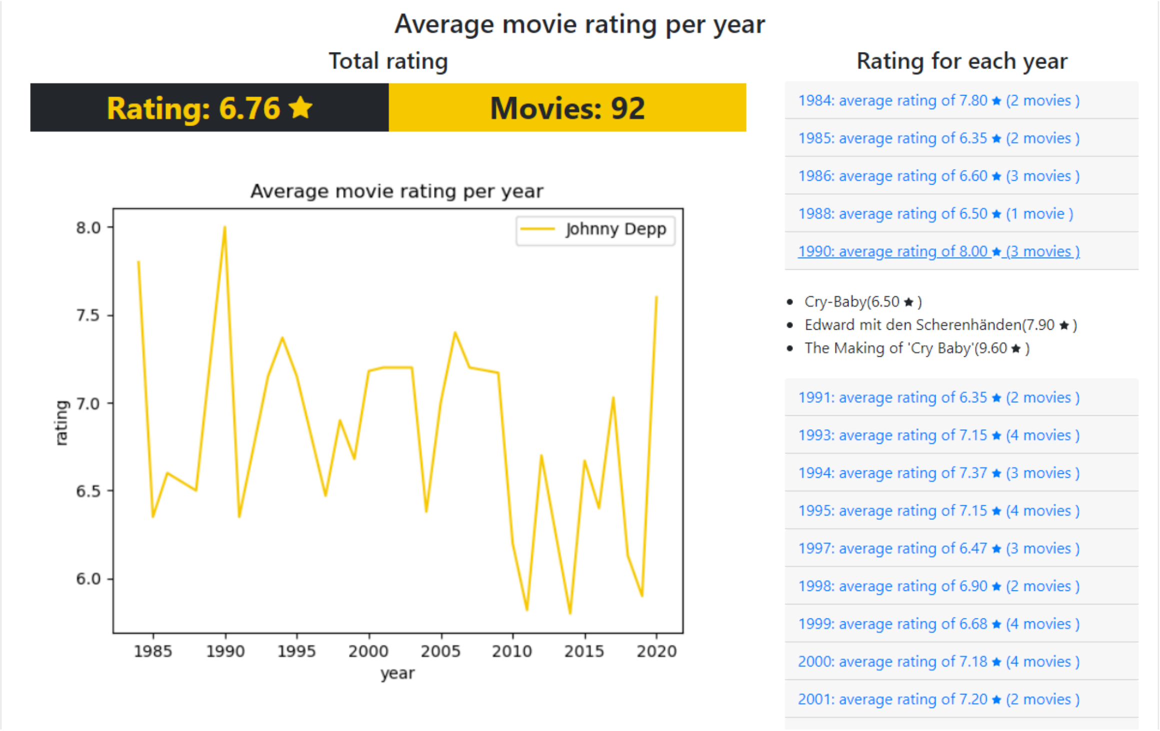
all genres

Drama:	37
Comedy:	27
Adventure:	20
Fantasy:	17
Action:	15
Short:	13
Crime:	10
Biography:	10
Music:	10
Documentary:	10
Horror:	8
Thriller:	8

Wordcloud of all genres of all movies

Figure 25: All awards of an actor

- Average rating of their movies (overall and each year)



Average movie rating per year

Total rating

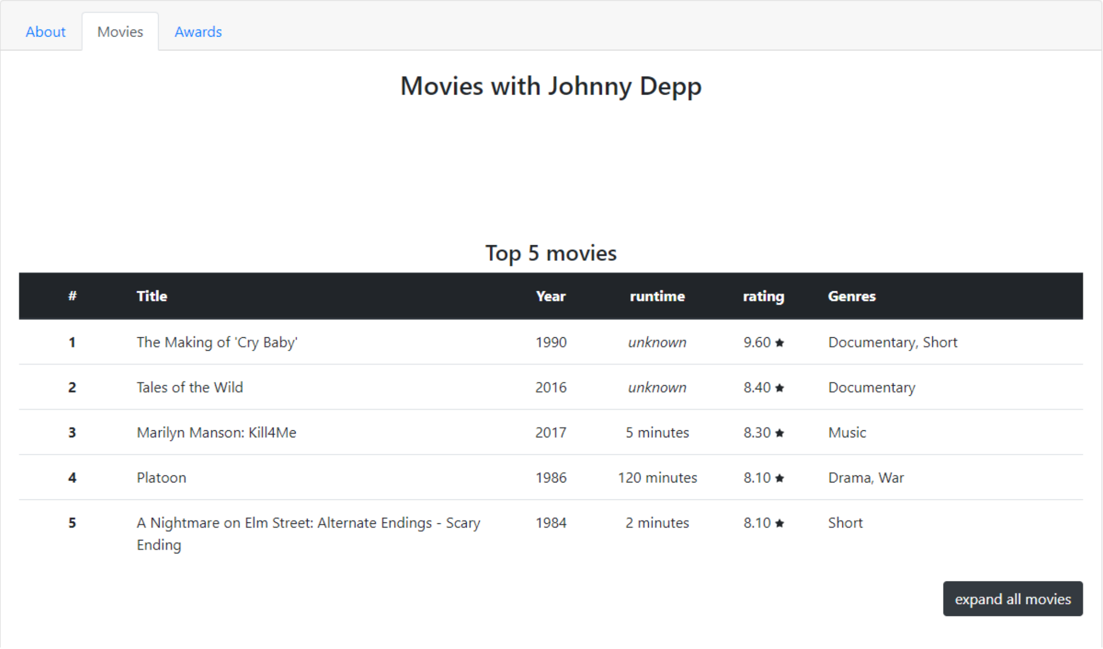
Rating: 6.76 ★ **Movies: 92**

Rating for each year

- 1984: average rating of 7.80 ★ (2 movies)
- 1985: average rating of 6.35 ★ (2 movies)
- 1986: average rating of 6.60 ★ (3 movies)
- 1988: average rating of 6.50 ★ (1 movie)
- 1990: average rating of 8.00 ★ (3 movies)
  - Cry-Baby(6.50 ★)
  - Edward mit den Scherenhänden(7.90 ★)
  - The Making of 'Cry Baby'(9.60 ★)
- 1991: average rating of 6.35 ★ (2 movies)
- 1993: average rating of 7.15 ★ (4 movies)
- 1994: average rating of 7.37 ★ (3 movies)
- 1995: average rating of 7.15 ★ (4 movies)
- 1997: average rating of 6.47 ★ (3 movies)
- 1998: average rating of 6.90 ★ (2 movies)
- 1999: average rating of 6.68 ★ (4 movies)
- 2000: average rating of 7.18 ★ (4 movies)
- 2001: average rating of 7.20 ★ (2 movies)

Figure 26: Actor rating and ratings over the years

- Top 5 movies, their respective years and genre



Movies with Johnny Depp

Top 5 movies

#	Title	Year	runtime	rating	Genres
1	The Making of 'Cry Baby'	1990	*unknown*	9.60 ★	Documentary, Short
2	Tales of the Wild	2016	*unknown*	8.40 ★	Documentary
3	Marilyn Manson: Kill4Me	2017	5 minutes	8.30 ★	Music
4	Platoon	1986	120 minutes	8.10 ★	Drama, War
5	A Nightmare on Elm Street: Alternate Endings - Scary Ending	1984	2 minutes	8.10 ★	Short

Figure 27: Actor rating and ratings over the year

A Planning Project Structure

```
imdb/
│
├── bin/
│
├── webapplikation/
│      ├──app/
│      │      ├── __init__.py
│      │      ├── admin.py
│      │      ├── apps.py
│      │      │
│      │      ├── migrations/
│      │      │      └── __init__.py
│      │      │
│      │      ├── models.py
│      │      ├── tests.py
│      │      └── views.py
│      │
│      ├──docs/
│      │
│      ├──project/
│      │      ├── __init__.py
│      │      ├── settings.py
│      │      ├── urls.py
│      │      └── wsgi.py
│      │
│      ├── static/
│      │      └── style.css
│      │
│      ├── templates/
│      └──    └── base.html
│
│
├── application/
│      ├── __init__.py
│      ├── runner.py
│      ├── scrape/
│      │      ├── __init__.py
│      │      ├── scrape.py
│      │      └── save_information.py
│      │
│      ├── processinformation/
│      │      ├── __init__.py
│      │      ├── process_information.py
│      │      └── get_information.py
│      │
│      ├── tests/
│      │      ├── scraping_tests.py
│      │      └── information_processing_tests.py
│      │
│      ├── docs/
│      │      ├── scrape.md
│      └──    └── processinformation.md
│
├── data/
│      ├── __init__.py
│      ├── create_databases.py
│      ├── save_to_database.py
│      ├── load_from_database.py
│      ├── docs/
│      │      └── . . .
│      ├── tests/
│      └──    └── . . .
│
├── .gitignore
├── LICENSE
└── README.md
```

B Flowchart Scraping

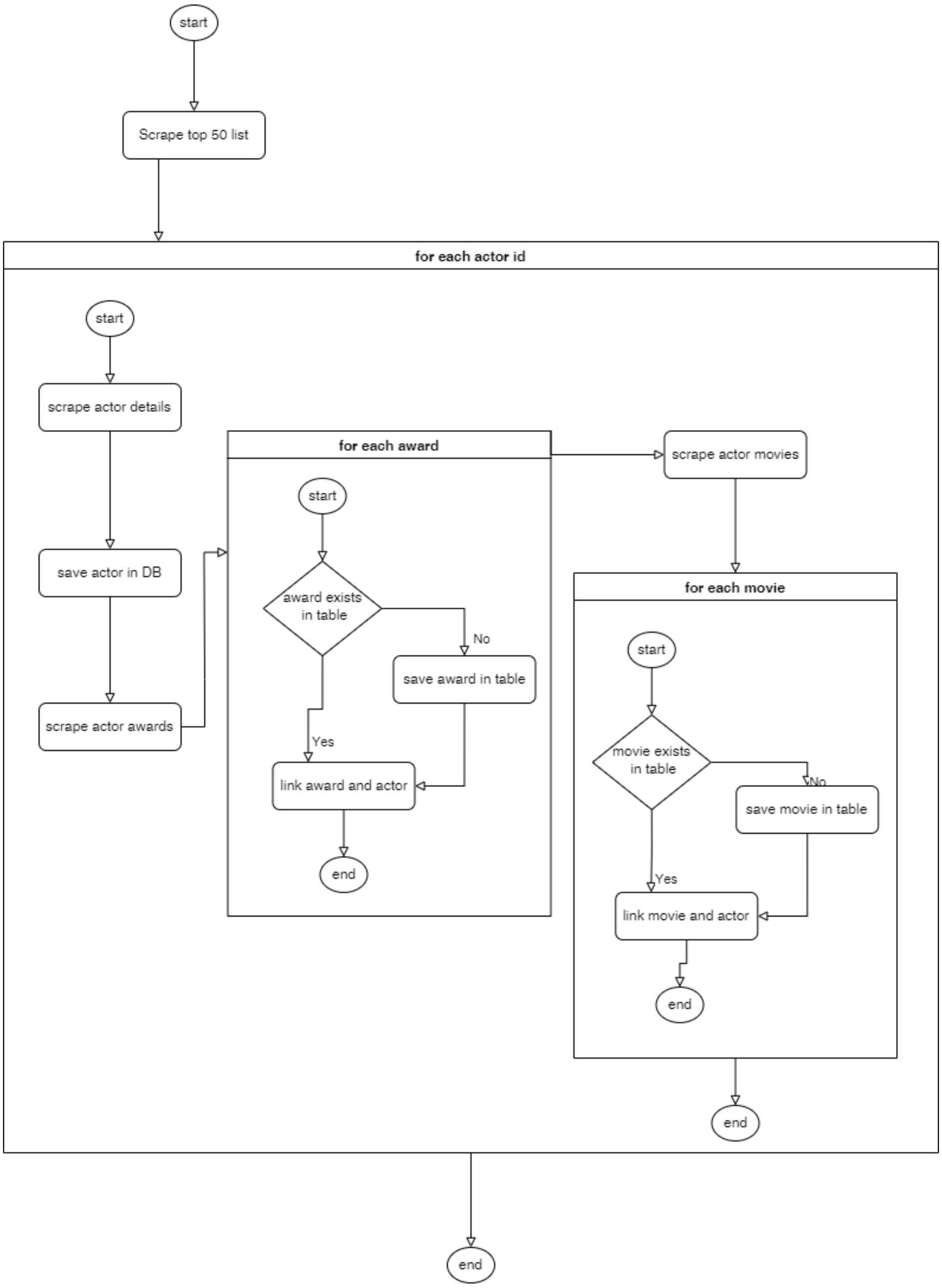


Figure 28: Flowchart scrape.py

C Documentation

C.0.1 IMBd.py

commandline application to control program by user input

`IMDb.`**`check_answer`**`(`*yes_answer*, *no_answer*`)`

Evaluates user input

Parameters

- **`yes_answer`** (*`str`*) – user input if user agrees
- **`no_answer`** (*`str`*) – user input if user disagrees

Returns True if user agrees

Return type bool

`IMDb.`**`configure_database`**`()`

Configure the database connection

Returns True if connection to database is possible with new configuration, else False

Return type bool

`IMDb.`**`create_db`**`(`*con*`)`

Initializes the databases

Parameters **`con`** (`Connection`) – Database Connection

Returns True if database was created successfully, False otherwise

Return type bool

`IMDb.`**`exit_application`**`()`

Ends the application

`IMDb.`**`print_help_text`**`()`

Shows help and all possible commands

`IMDb.`**`rename_database`**`(`*con*`)`

Renames database name in configuration file

Parameters **`con`** (`Connection`) – Database Connection

`IMDb.`**`scrape_information`**`()`

Starts scraping the data from the imdb page

Returns True if scraping worked with no errors, False otherwise

Return type bool

`IMDb.`**`start_from_scratch`**`()`

Starts from scratch. Tells you exactly what you need to do next step by step.

`IMDb.`**`start_web_app`**`()`

Starts web application on `http://127.0.0.1:5000/`

`IMDb.`**`switcher_user_input`**`(`*input*`)`

Finds right function for user input

Parameters **`input`** (*`str`*) – user input

Returns corresponding function

Return type function

C.0.2 constants.py

contains string constant

C.1 Application

C.1.1 create_charts.py

`create_charts.`**`avg_awards_movies_bar`**`(`*actor_id*`)`

Generates a bar chart of all awards the actor has won or was nominated for and the amount of movies he played in in comparison to the other actors and saves the chart

Parameters **`actor_id`** (*`str`*) – id of actor

Returns plot data of chart

Return type dict

`create_charts.`**`avg_awards_plot`**`(`*actor_id*`)`

Generates a plot of all awards the actor has won in comparison to the other top 50 actors and saves the chart

Parameters **`actor_id`** (*`str`*) – id of actor

`create_charts.`**`awards_plot`**`(`*actor_id*`)`

Generates a plot of all awards the actor has won and was nominated over the years as well as the movies he played

Parameters **`actor_id`** (*`str`*) – id of actor `create_charts.`**`genres_pie_chart`**`(`*actor_id*`)`

Generates a pie chart of all genres of the top 5 movies with one actor and saves the chart

Parameters **`actor_id`** (*`str`*) – id of actor

Returns dataframe of chart

Return type dict

`create_charts.`**`genres_wordcloud_chart`**`(`*actor_id*`)`

Generates a word cloud of all genres of all 5 movies with one actor and saves the word cloud

Parameters **`actor_id`** (*`str`*) – id of actor

Returns dataframe of chart

Return type dict

`create_charts.`**`movie_rating_per_year`**`(`*actor_id*`)`

Generates a plot of the ratings of the movies the actor played in over the years (in comparison to the other actors) and saves the chart

Parameters **`actor_id`** (*`str`*) – id of actor

Returns plot data of chart

Return type dict

C.1.2 scrape.py

This script allows the user to scrape all actors, movies and awards from the imdb top actors list.

It then stores the values in a database.

C.2 scrape_actor_awards.py

`class` `scrape.scrape_actor_awards.`**`Award`**`(`*title=None*, *award_entry=None*, *last_award=None*`)`

Class that stores all important Award information

`generate_key()`

generates unique primary key for awards table

`get_award_info()`

returns data to save in database

Returns data to save in database

Return type dict

get_linking_information()

returns data to save in database

Returns data to save in database

Return type dict

scrape.scrape_actor_awards.**scrape_all_awards_of_actor**(*actor_id*)

Scrapes all awards of one actor :type actor_id: str :param actor_id: id of the actor the awards are to get scraped of :returns: list of all awards :rtype: list

C.3 scrape_actor_details.py

class scrape.scrape_actor_details.**Actor**(*actor_id*, *pos*)

Scrapes all important information of an actor

get_actor_information()

returns data to save in database

Returns data to save in database

Return type dict

scrape_actor_bio()

Scrapes bio of actor

scrape_actor_information()

Scrapes all important information of an actor

scrape_awards()

Scrapes awards of actor

scrape_movies()

Scrapes movies of actor

C.3.1 scrape_actor_movies.py

`class scrape_actor_movies.`**`Movie`**`(`*actor_id*, *movie_entry*`)`

Class that stores all important movie information

`get_genres()`

returns genres of movie

Returns genres of movie

Return type list(dict)

`get_movie_information()`

returns data to save in database

Returns data to save in database

Return type dict

`scrape_actor_movies.`**`find_genres`**`(`*movie_entry*`)`

Scrape genres from element

Parameters **`movie_entry`** (*`PageElement`*) – element to scrape from

Returns genres

Return type list

`scrape_actor_movies.`**`find_rating`**`(`*movie_entry*`)`

Scrape rating from element

Parameters **`movie_entry`** (*`PageElement`*) – element to scrape from

Returns rating

Return type str

`scrape_actor_movies.`**`find_runtime`**`(`*movie_entry*`)`

Scrape runtime from element

Parameters **`movie_entry`** (*`PageElement`*) – element to scrape from

Returns runtime

Return type str

`scrape_actor_movies.`**`find_year`**`(`*movie_entry*`)`

Scrape year from element

Parameters **`movie_entry`** (*`PageElement`*) – element to scrape from

Returns year

Return type int

`scrape_actor_movies.`**`scrape_all_movies_of_actor`**`(`*actor_id*, *gender*`)`

Scrapes all movies of one actor

Parameters

- **`actor_id`** (*`str`*) – id of actor
- **`gender`** (*`str`*) – gender of actor

Returns list of all movies

Return type list of Movies

`scrape_actor_movies.`**`scrape_movie_from_url`**`(`*url*, *movie_list*, *actor*`)`

Scrapes movies of one actor

Parameters

- **`actor`** (*`str`*) – id of actor
- **`url`** (*`str`*) – url to scrape from
- **`movie_list`** (*`list`*) – list of movies that were already scraped

Returns list of scraped movies

Return type list of Movies

C.3.2 scrape_helper.py

Library that contains functions for the scraping

`scrape_helper.`**`find_soup_from_url`**`(`*url*`)`

creates soup from url

Parameters **`url`** (*`str`*) – url to website

Returns soup of website

Return type BeautifulSoup

`scrape_helper.`**`print_progress_bar`**`(`*iteration*`)`

Call in a loop to create terminal progress bar :param iteration: current iteration :type iteration: int

`scrape_helper.`**`wrap_and_escape_text`**`(`*text*`)`

makes strings easier to dave in databases

Parameters **`text`** (*`str`*) – text to escape

Returns escaped text

Return type str

C.4 Data

C.4.1 create_queries.py

stores all queries needed to create the tables and the database

C.4.2 insert_queries.py

stores all insert queries

`.insert_queries.`**`insert_into_query`**`(`*database*, *key_value_pairs*`)`

creates query to insert values into one table :type database: str :param database: name of database to insert in :type key_value_pairs: dict :param key_value_pairs: keys where to insert, values what should be inserted :returns: insert into query :rtype: str

C.4.3 select_queries.py

stores all select queries

C.4.4 db_config.py

module that saves, updates and reads the database configuration

`db_config.`**`get_config`**`()`

Reads the configuration of the database connection.

Returns the database configuration

Return type dict

Raises FileNotFoundError

`db_config.`**`init_config`**`(`*host*, *user*, *password*, *database*`)`

Initializes the connection configuration for the database

Parameters

- **`host`** (*`str`*) – Host name for the database connection
- **`user`** (*`str`*) – User name for the database connection

- **password** (*str*) – Password for the database connection
- **database** (*str*) – Name of the database

`db_config.`**`update_config`**`(`*key*, *value*`)`

Updates database configuration

Parameters

- **key** (*str*) – key of the value that should be changed
- **value** (*str*) – new value

Raises FileNotFoundError

C.4.5 db_connection.py

`class db_connection.`**`Connection`**

Connection to the database

`create_connection()`

“creates a connection

Returns connection to database

Return type MySQLConnection

Raises ConnectionError

`create_connection_to_database()`

“creates a connection to the database

Returns connection to database

Return type MySQLConnection

Raises ConnectionError

`database_exists()`

“checks if configured database exists

Returns connection to database

Return type MySQLConnection

Raises ConnectionError

`entry_exists``(`*table*, *pk*`)`

Checks if entry already exists

Parameters

- **table** (*str*) – table to check
- **pk** (*str*) – primary key

Returns True if value already exists in db

Return type bool

`execute_query`(*query*, *connect_to_database=True*)

executes query

Parameters

- **query** (*str*) – query to be executed
- **connect_to_database** (*bool*) – connect to specific database if True (True by default)

`execute_read_query`(*query*, *dict_res=True*, *connect_to_database=True*)

executes query with return value

Parameters

- **query** (*str*) – query to be executed
- **connect_to_database** (*bool*) – connect to specific database if True (True by default)
- **dict_res** (*bool*) – returns values as dictionary (True by default)

Returns result of query

Return type list or dict

`get_primary_key_name`(*table*)

primary key of the table

Parameters **table** (*str*) – table to check

Returns primary key

Return type str

`init_data_base`()

initializes database it it does not exist, drops and creates all tables

`save_value`(*value*, *table*, *pk=None*)

saves new value in table

Parameters

- **value** – new values to save

- **table** (*str*) – table in which table the values should be stored
- **pk** (*str*) – pk optional, to check if value already exists

Returns True if insert was successful

Return type bool

C.4.6 get_from_database.py

`get_from_database.`**`get_actor_name`**`(`*actor_id*`)`

Parameters **actor_id** (*str*) – actorID of actor

Returns name of actor

Return type str

`get_from_database.`**`get_all_actors`**`()`

Returns List of all actors

:returns list of all actors :rtype: list(dict)

`get_from_database.`**`get_avg_amounts`**`(`*actor_id*, *amount=None*`)`

extracts average nominations, movies and wins of one or all actors

Parameters

- **amount** (*int*) – amount of all actors (default is None)
- **actor_id** (*str*) – actorID of actor

Returns list of results

Return type list

`get_from_database.`**`get_avg_awards`**`()`

extracts average awards of all actors

Returns average awards of all actors

Return type dict

`get_from_database.`**`get_avg_movie_rating_per_year`**`(`*actor_id*`)`

extracts average movie rating of of specific actor per year

Parameters **actor_id** (*str*) – actorID of actor

Returns average movie rating of of specific actor per year

Return type dict

`get_from_database.`**`get_awards_of`**`(`*actor_id*, *con=None*`)`

extracts awards of a specific actor

Parameters

- **`con`** (`Connection`) – Database Connection
- **`actor_id`** (*`str`*) – actorID of actor

Returns all awards of specific actor

Return type list(dict)

`get_from_database.`**`get_general_rating_dict`**`()`

extracts average movie rating of all actors per year

Returns average movie rating of of all actors per year

Return type dict

`get_from_database.`**`get_genres_of_actor`**`(`*actor_id*`)`

Parameters **`actor_id`** (*`str`*) – actorID of actor

Returns all genres of one actor

Return type list

`get_from_database.`**`get_genres_of_top_movies`**`(`*actor_id*`)`

Parameters **`actor_id`** (*`str`*) – actorID of actor

Returns genres of top movies of one actor

Return type list

`get_from_database.`**`get_movies_of`**`(`*actor_id*, *con=None*`)`

extracts all movies of a specific actor

Parameters

- **`con`** (`Connection`) – Database Connection
- **`actor_id`** (*`str`*) – actorID of actor

Returns all movies of specific actor

Return type list(dict)

`get_from_database.`**`get_new_movie`**`(`*actor_id*, *con*`)`

extracts newest movie of a specific actor

Parameters

- **`con`** (`Connection`) – Database Connection
- **`actor_id`** (*`str`*) – actorID of actor

Returns newest movie of specific actor as dict

Return type dict

`get_from_database.`**`get_pop_movie`**`(`*actor_id*, *con*`)`

extracts most popular movie of a specific actor

Parameters

- **`con`** (`Connection`) – Database Connection
- **`actor_id`** (*`str`*) – actorID of actor

Returns most popular movie of specific actor as dict

Return type dict

`get_from_database.`**`get_single_actor`**`(`*actor_id*`)`

extracts information of a specific actor

Parameters **`actor_id`** (*`str`*) – actorID of actor

Returns all information of one specific actor needed for the 'about' page

Return type dict

C.4.7 save_to_database.py

`save_to_database.`**`persist_information`**`(`*actor*`)`

saves actor in database, as well as his awards and the movies he played in

Parameters **`actor`** (*`Actor`*) – actor to save in db

C.5 Presentation

C.5.1 app.py

starts web application and returns corresponding templates