1. Exercise

Data Science mit Python

Deadline: Tue, 18.11.25, Midnight

Wintersemester 2025/26

Patrick Schäfer, patrick.schaefer@hu-berlin.de

Agenda

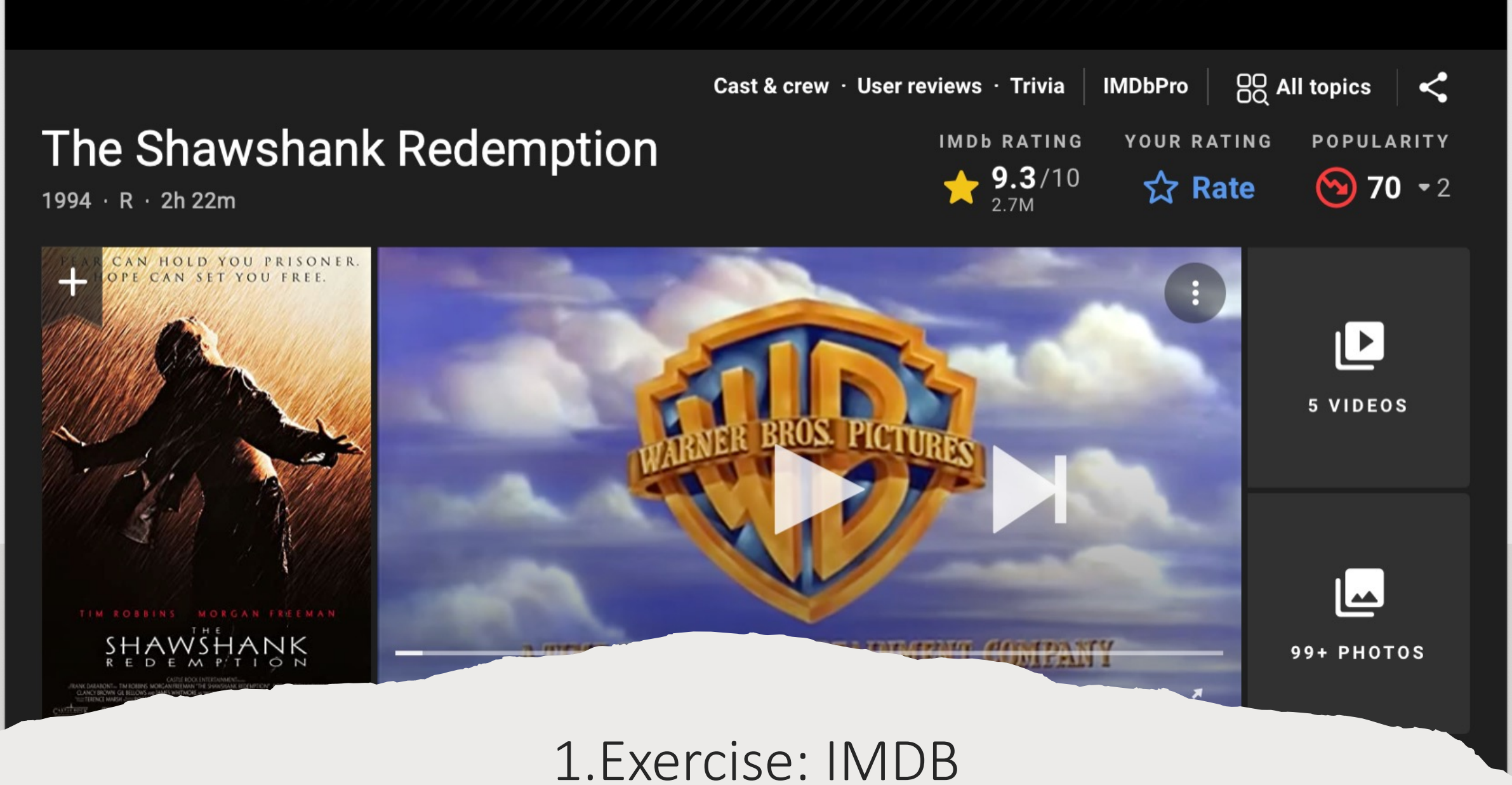
- Questions?
- 1. Exercise
- Background
  - Beautiful Soup
  - Pandas
  - OpenAI / Mistral
  - Vibe Coding
  - Streamlit

Any Question?

Organization

- Team size must be 2
  - Smaller teams are not allowed
  - Use Moodle for building teams

- One Assignment every 2-3 weeks:
  - **Exploratory Data Science Programming Assignments**
  - Jupyter Notebooks



1.Exercise: IMDB

IMDB

- Task:
  - Get a list of TOP-250 movies, do **exploratory data analysis**
- Problem:
  - IMDB data is human-readable, but semi-structured
- Idea:
  - We "*scrap*"* data from each movie on IMDB

    * Data scrapping is a technique in which a program extracts data from human-readable output
  - Then, we perform analysis on the scrapped data

Concrete Tasks

1. Part 1: Scraping IMDB
   - Implement a Jupyter Notebook that scraps a list of 250 movie titles from a JSON file
   - For each movie URL, extract data from HTML and store to a CSV file

2. Part 2: Exploratory Data Analysis (EDA)
   - Implement queries on the extracted movies' metadata

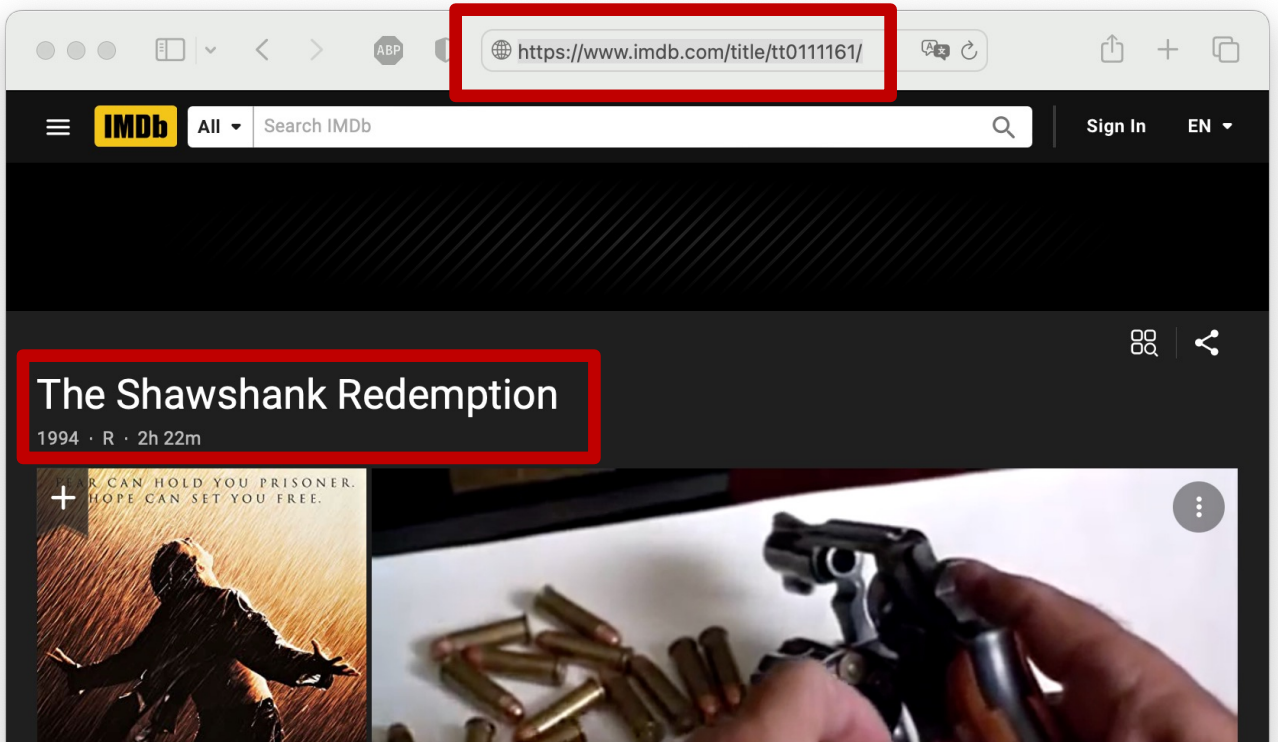
3. Part 3: Vibe Coding – Exploratory Analysis APP
   - Implement a Streamlit APP that shows an EDA on the TOP-250 movies

Part 1 – Scraping IMDB

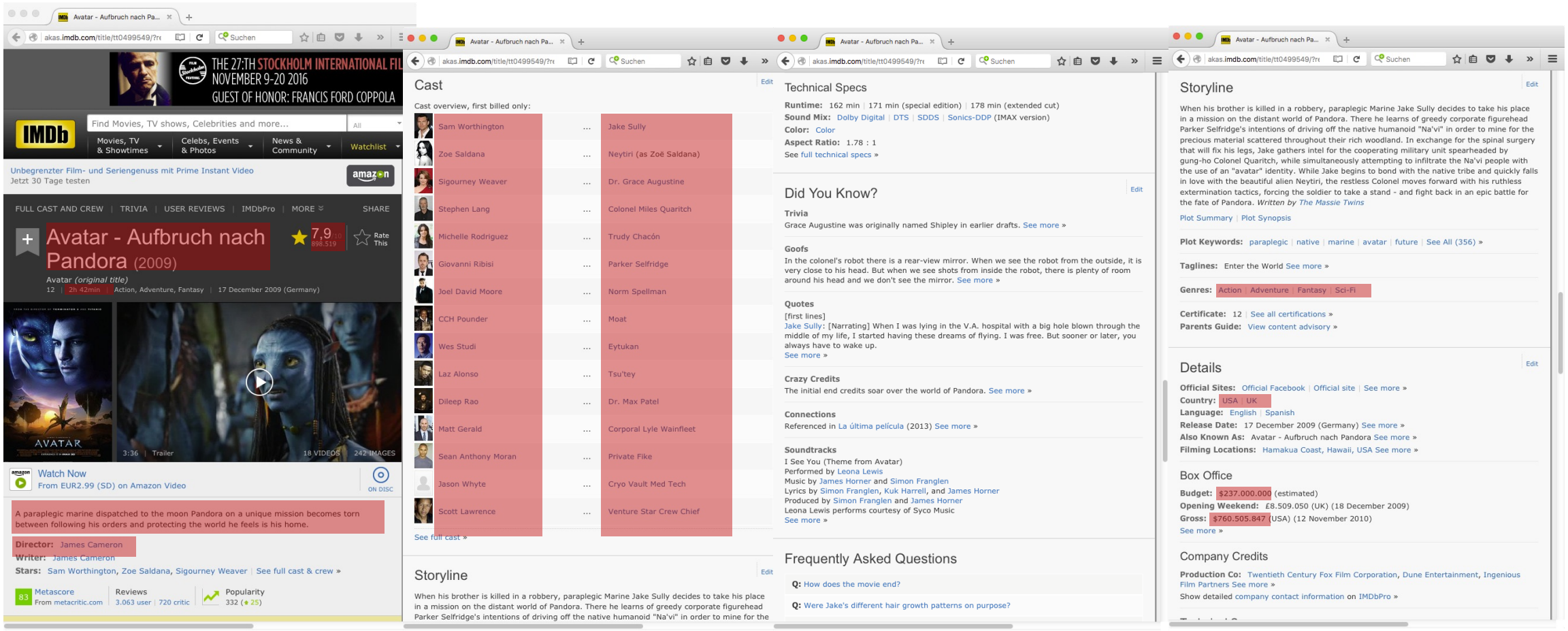
- The Primer Jupyter Notebook retrieves the HTML given an URL

```
response = requests.get('https://www.imdb.com/' + href)
```

- Tasks
  1. Parse html file using Beautiful Soup
  2. Apply data conversion
  3. Finally, write dataset to JSON file



Task 1.1 – Extract Data from URL



- Use **Beautiful Soup** to parse and extract the highlighted data from the movie webpages

Task 1.1 – Extract Data from URL

- Extract the following information from each movie and store it to a separate JSON file:
  - **Primitive Types:**
    `url, title, year, description, budget, gross, ratingValue, ratingCount, duration`
  - **List-Types:**
    `genreList, countryList, castList, characterList, directorList`

  - A variable ending in *List* such as *genreList* refers to the extracted data type being a list, e.g. it has multiple values

- Stick to exactly these names!

Task 1.1 – Data Cleansing

- Some datatypes are not in a format, we can use to process the data such as the `duration`, the `ratingCount` or `gross`
  - We will ignore `budget`, as it contains different currencies, too

your result:

	url	title	ratingValue	ratingCount	year	description	budget	gross	duration	genreList	countryList	castList	characterList
0	/title/tt0111161/	Die Verurteilten	9.3	2.7M	994	Two imprisoned men bond over a number of years...	$25,000,000 (estimated)	$28,884,504	2h 22m	[Drama]	[United States]	[Tim Robbins, Morgan Freeman, Bob Gunton, Will...	[Andy Dufresne, Ellis Boyd 'Red' Redding, Ward...

expected (converted) result – we will do conversion next:

	url	title	ratingValue	ratingCount	year	description	budget	gross	duration	genreList	countryList	castList	characterList	di
0	/title/tt0111161/	Die Verurteilten	9.3	2700000	1994	Two imprisoned men bond over a number of years...	$25,000,000 (estimated)	28884504	142	[Drama]	[United States]	[Tim Robbins, Morgan Freeman, Bob Gunton, Will...	[Andy Dufresne, Ellis Boyd 'Red' Redding, Ward...	

Task 1.1 – Data Cleansing

- Implement the following three function:
  1. `convert_duration():`
     - Converts from the duration format to minutes.
     - *E.g. from 2h 22m to 144*

  1. `convert_rating_count():`
     - Converts from the human readable format to an integer.
     - *E.g. 2.7M to 2.700000 and 2.7k to 2.700*

  1. `convert_gross():`
     - Converts the gross to an integer.
     - *E.g. $28,884,504 to 28884504*

Task 1.1 – Cleansed Data

- The TOP-250 data will be stored in a Pandas DataFrame
- **Nothing to be implemented**
  - **Just make sure, there are no errors when parsing all files**
  - **E.g. catch all special cases**

- Finally, the DataFrame is written to a json file:

  part1_submission.json

]:

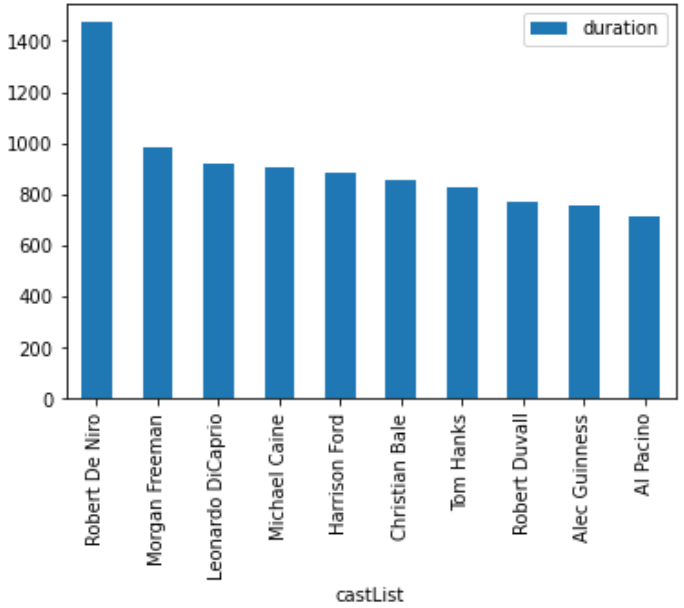
	url	title	ratingValue	ratingCount	year	description	budget	gross	duration	genreList	countryList	castList	characterList	
0	/title/tt0111161/	Die Verurteilten	9.3	2700000	1994	Two imprisoned men bond over a number of years...	$25,000,000 (estimated)	0	142	[Drama]	[United States]	[Tim Robbins, Morgan Freeman, Bob Gunton, Will...	[Andy Dufresne, Ellis Boyd 'Red' Redding, Ward...	
1	/title/tt0068646/	Der Pate	9.2	1800000	1972	The aging patriarch of an organized crime dyna...	$6,000,000 (estimated)	0	175	[Crime, Drama]	[United States]	[Marlon Brando, Al Pacino, James Caan, Diane K...	[Don Vito Corleone, Michael, Sonny, Kay Adams,...	
2	/title/tt0468569/	The Dark Knight	9.0	2600000	2008	When the menace known as the Joker wreaks havo...	$185,000,000 (estimated)	0	152	[Action, Crime, Drama]	[United States, United Kingdom]	[Christian Bale, Heath Ledger, Aaron Eckhart, ...	[Bruce Wayne, Joker, Harvey Dent, Alfred, Rach...	[
3	/title/tt0071562/	Der Pate 2	9.0	1300000	1974	The early life and career of Vito Corleone in ...	$13,000,000 (estimated)	0	202	[Crime, Drama]	[United States]	[Al Pacino, Robert De Niro, Robert Duvall, Dia...	[Michael, Vito Corleone, Tom Hagen, Kay, Fredo...	
4	/title/tt0050083/	Die zwölf Geschworenen	9.0	784000	1957	The jury in a New York City murder trial is fr...	$350,000 (estimated)	0	96	[Crime, Drama]	[United States]	[Henry Fonda, Lee J. Cobb, Martin Balsam, John...	[Juror 8, Juror 3, Juror 1, Juror 2, Juror 4, ...	

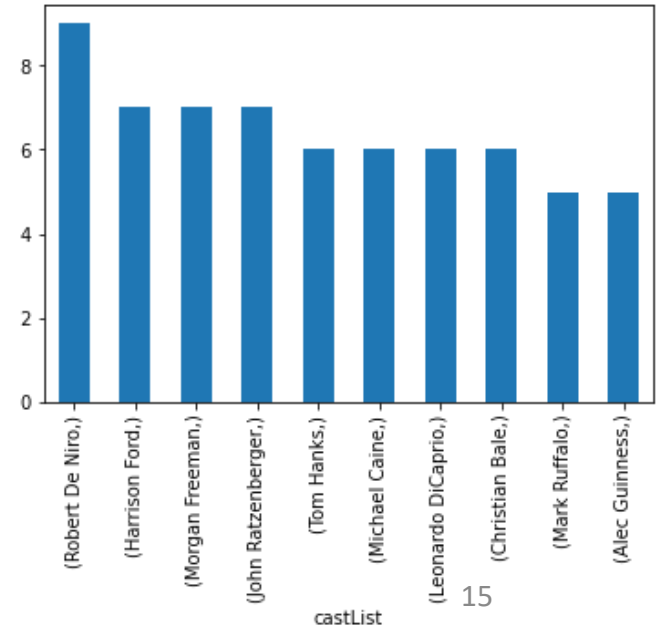
Part 2 – Exploratory Data Analysis

- The primer Notebook loads the TOP-250 movies in JSON-format
  - **`primer_part2_imdb_eda.ipynb`**
- **You can solve this exercise without solving part 1**
- You have to correctly implement (at least) **five** out of **ten** queries:
  1. Patience
  2. Binge Watching Steven Spielberg
  3. This is about me
  4. Workhorse
  5. Cash horse
  6. Beloved Genres
  7. Must See
  8. Uncreative writers
  9. Glamour of Hollywood
  10. **LLM Chat Client**

Queries

1. Patience
   - Select all movies with a duration above or equal to 220 minutes.
2. Binge Watching Steven Spielberg
   - Select the total duration of all movies directed by Steven Spielberg
3. This is about me
   - Provide a ranked list of the TOP-10 actors with the most screen-time, as counted by total duration of the movies they are starring in
4. Workhorse
   - Provide a ranked list of the TOP-10 most active actors (cast), i.e., those actors which have starred in most movies
5. Cash horse
   - Provide a ranked list of the TOP-10 most successful actors (cast), i.e., those actors which have generated the most gross





Queries

6. Beloved Genres
   - Provide a ranked list of the TOP-10 most successful genres, i.e., those genres which have generated the most gross
7. Must See
   - List the best rated movie of each year starting from (including) 1990 until (including) 2000. Order the movies by increasing year. The result must contain exactly 11 rows.
8. Uncreative writers
   - Provide a ranked list of the TOP-10 most frequent character names of all times ordered by the frequency of occurrence in movies
9. Glamour of Hollywood
   - Compute the cumulative gross over the years
10. **Chat Client**
    - **Write a simple chat client that queries your IMDB data**

Optional: Chat-Client

- Write a simple chat client to query the data
- It must connect to the HU MISTRAL API

```
from openai import OpenAI
base_url = "https://llm3-compute.cms.hu-berlin.de/v1/"
OPENAI_API_KEY = "secret but not used"
client = OpenAI(
    base_url=base_url, api_key=OPENAI_API_KEY)
```

- For this to work, you must be connected to the HU Intranet (using either HU VPN or Intranet)
  - https://amor.cms.hu-berlin.de/account/ip.cgi

```
user_question = "Which movies contain animals?"
ask_user_question(user_question)
print("\n")
```

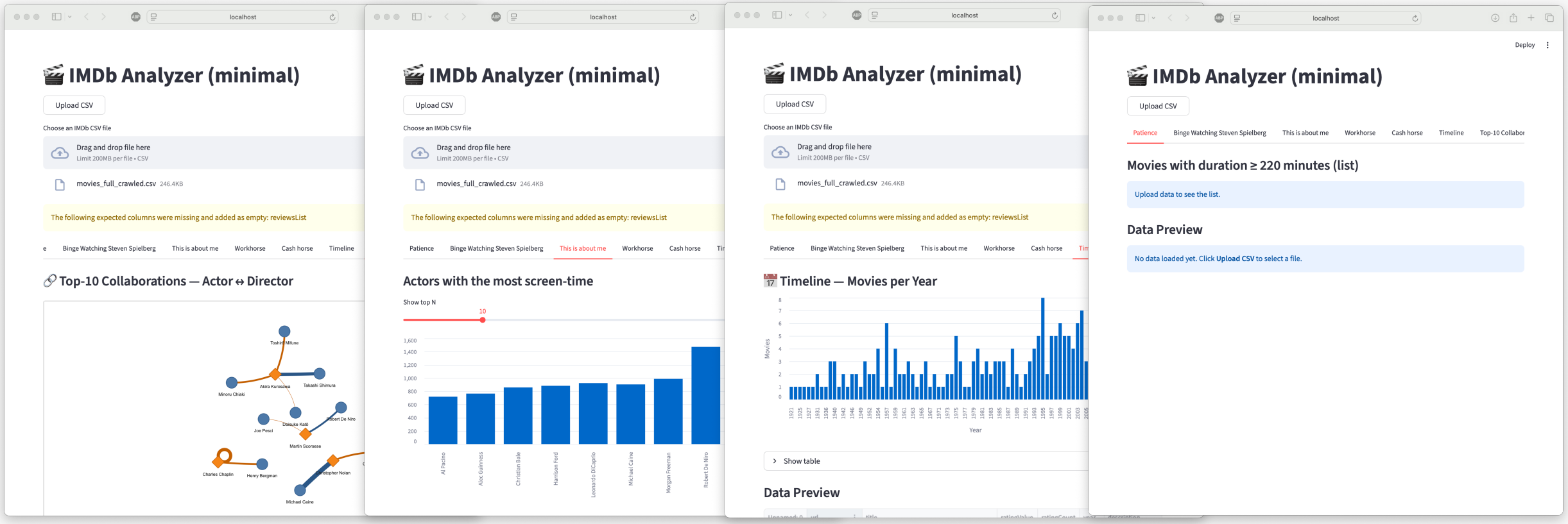
```
Question: 'What is the best rated movie?'
Answer:   'the best rated movie is "the shawshank redemption" with a rating of
9.3.'


Question: 'What is the most expensive movie?'
Answer:   'the most expensive movie in the dataset is "the dark knight" with a b
udget of $185,000,000 (estimated).'


Question: 'Which movie is your favorite? You must pick one!'
Answer:   'based on the provided data, the movie with the highest rating is "the
shawshank redemption" with a rating of 9.3. therefore, my favorite movie is:

**the shawshank redemption**'


Question: 'Which movies contain animals?'
Answer:   'based on the provided data, the movies that contain animals are:

1. **aladdin** (genie is a magical creature, and there are other animals like ab
u the monkey)
2. **the iron giant** (features a giant robot that is often considered an animal
-like character)
3. **dersu uzala** (involves an explorer and a native of the siberian wildernes
s, featuring interactions with wildlife)'
```

Background – Basic Operations

- Obviously, the queries follow a SQL-like form
- Some useful operations on DataFrames include
  - Reading JSON-files
  - Handling List Columns
  - Indexing and Slicing
  - Boolean Filter
  - Aggregation
  - Sorting
  - Ranking

Jupyter Notebooks

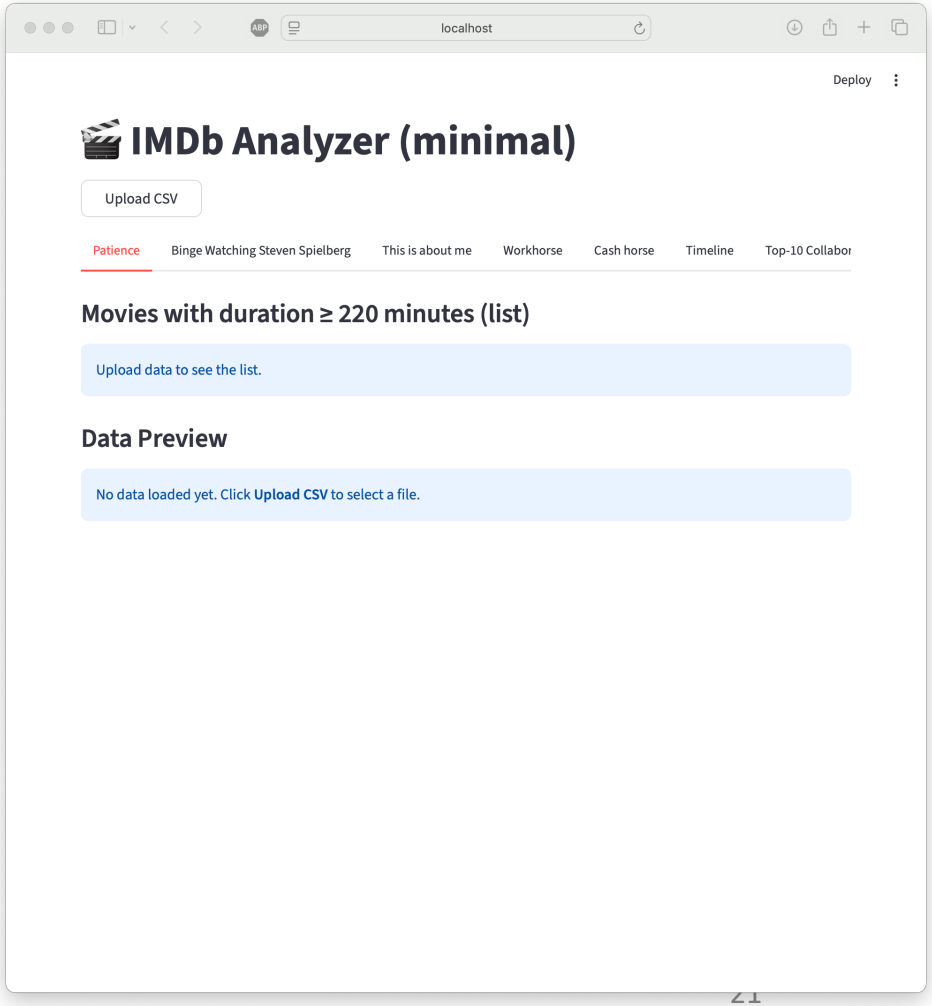
- To get started for parts 1 and 2
  - Download the two **Jupyter Notebooks in Moodle**
    - **primer_part1_imdb_scraping.ipynb**
    - **primer_part2_imdb_eda.ipynb**
  - Solve the exercise



Part 3 – Vibe Coding – EDA

Part 3 – Vibe Coding Task:
Build a Minimalistic Streamlit App

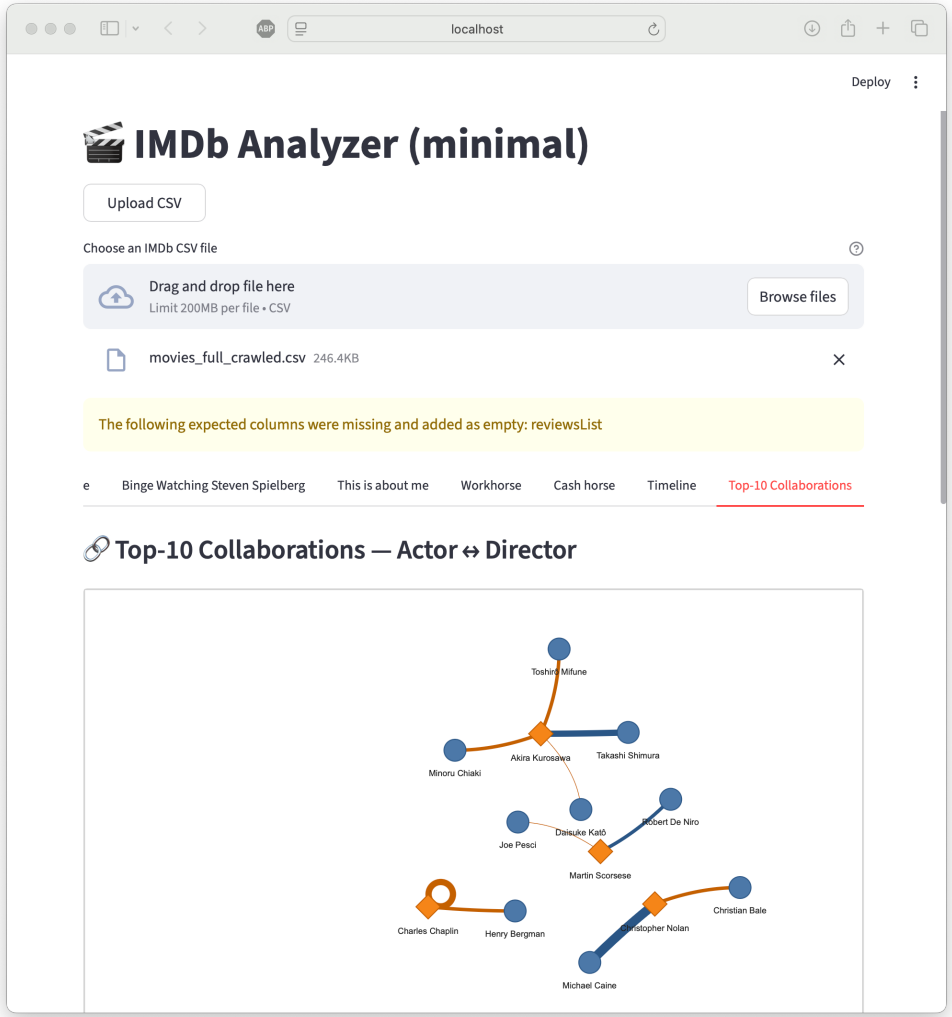
- It should include an **upload button** for **CSV or JSON files containing the TOP-250 movies**, and **five tabs**:
  1. Patience
  2. Binge Watching Steven Spielberg
  3. This is about me
  4. Workhorse
  5. Cash horse
- After upload, display the data in a **table view**.
- Each **tab** should display one of the **five predefined queries**
- **Add at least one WOW-feature (next)**



WOW-Features – Some examples

1. **Timeline View**
   - Plot a bar chart showing the number of movies per year
2. **Genre Word Cloud**
   - Generate a word cloud from genreList using the wordcloud library
3. **Top Collaborations**
   - Create a network graph of actors and directors
   - Draw an edge if an actor and director collaborated on a movie
   - Display only the Top-10 collaborations
   - Use i.e. networkx and pyvis libs for visualization
4. **Interactivity**
   - Add a slider to let the user choose how many elements (e.g., Top 5 to Top 20) to display

- ...?



Submission Deadline: Tue, 18.11.25, Midnight

- Submission via Moodle
- The task is considered to have been **successfully solved if**
  - Task 1: Scraping and cleansing the **first webpage is correct**
  - Task 2: **Five out of 10** queries are implemented correctly
  - Task 3: Your streamlit solution **can be executed** and contains **one WOW-features**

- **Deliverables:**
  - **a) Task 1 / Task 2 Exploratory Data Analysis (5 Files!)**
    - **Two** Jupyter-Notebooks
    - **Two** html-exports
    - **One** json-file: part1_submission.json
  - **b) Task 3: Vibe Coding (>3 Files!):**
    - **The file** with **python code**
    - **One file** documenting the **used prompts** and the **used LLM**
    - **Screenshot(s)** or a small video

Agenda

- Questions?
- 1. Exercise
- Background
  - Beautiful Soup
  - Pandas
  - OpenAI / Mistral
  - Vibe Coding
  - Streamlit

Background – HTML

- *Hypertext Markup Language* (HTML) is the standard markup language for documents designed to be displayed in a web browser

- <html> <head> <title> My beautiful web page! </title> </head> <body> Here is my content. </body> </html>

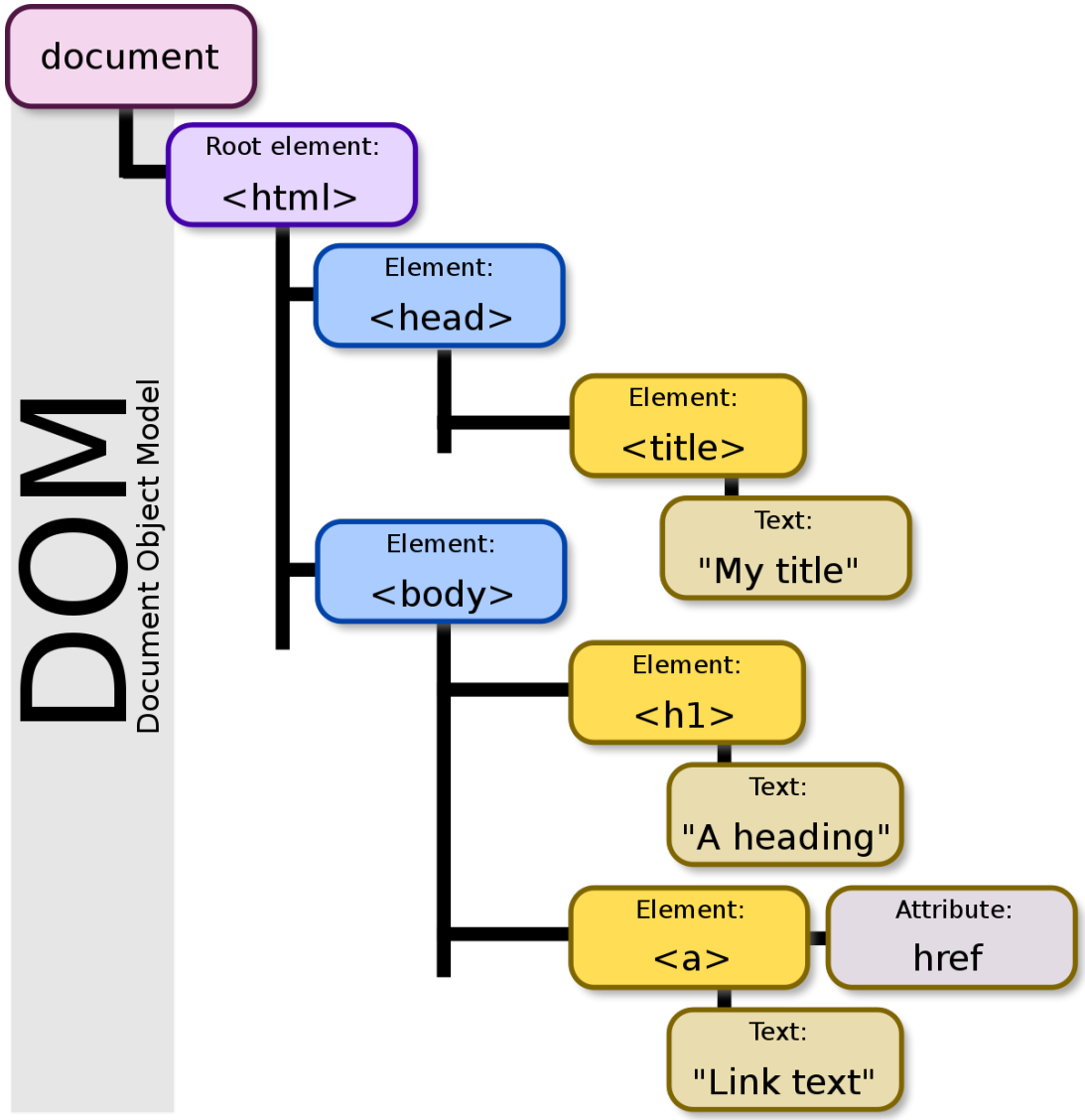
Namespace of HTML-Elements

- Tags begin with < and end with >
- Usually occur in pairs:
   Wow!
- Elements can be nested:
  This is a really interesting paragraph.
- Some tags never occur in pairs

- Tags may have one ore more attributes
  Contact us

DOM-Model

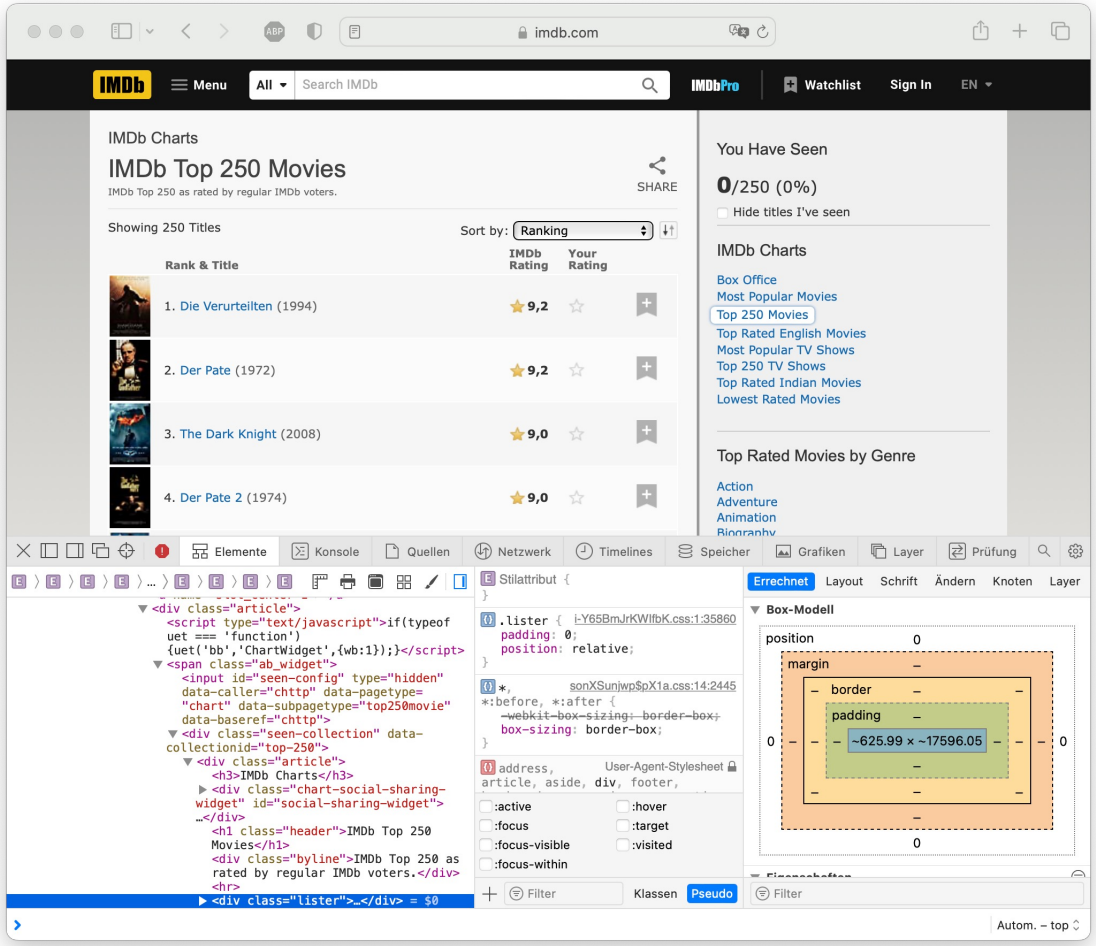
- We represent XML and HTML files using the Document Object Model (DOM)
- The DOM is a logical tree
- Allows hierarchical navigation



Browser Developer Tools

- We may use the browser developer tools to inspect and locate elements of a webpage

- You can locate elements in a browser using XPATH



XPATH

- XPATH is a syntax for navigating parts of an XML document
- Has a directory-path-like syntax

```
<table class=”list">
    <tr>
        <TD class = ”result">Avatar</TD>
    </tr>
</table>
```

- XPATH:
  `/table[@class='list']//td[@class='result']/text()`

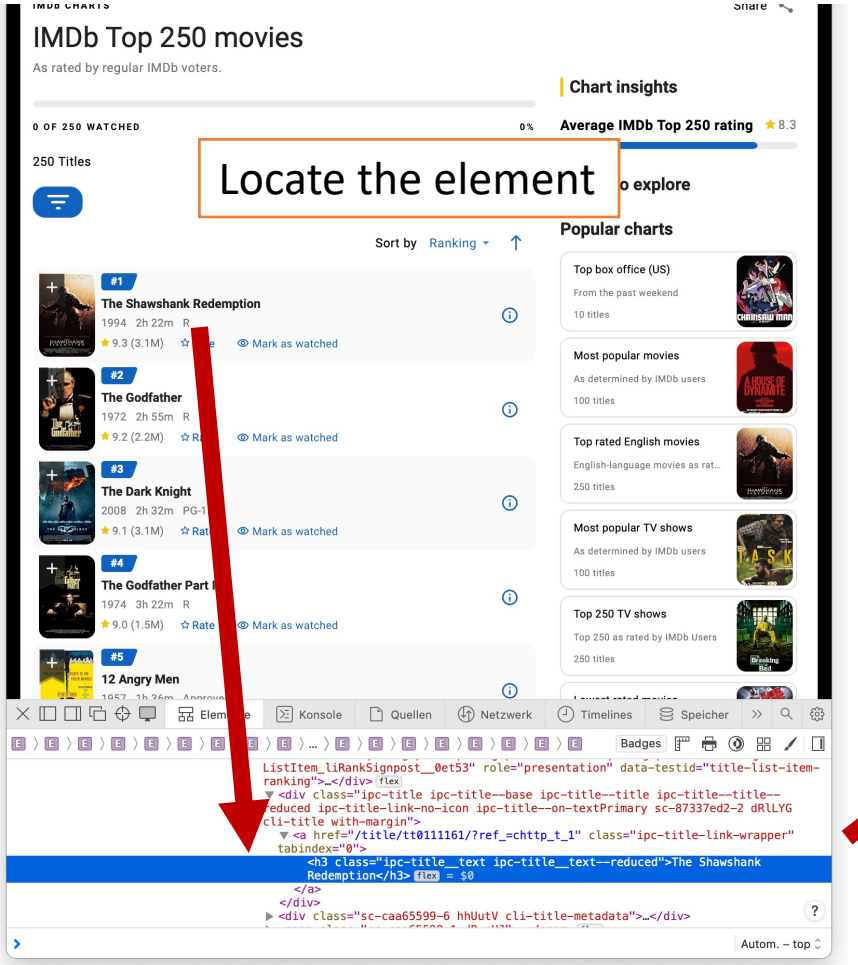
  => Avatar

Beautiful Soup

- Beautiful Soup (BS4) is a html parsing library
  - https://www.crummy.com/software/BeautifulSoup/bs4/doc/

In [6]:
```
# Search the IMDB websize for the TOP 250 movies
response = requests.get('https://www.imdb.com/chart/top/?ref_=nv_mv_1000')

# Throw a warning for non-200 status codes
if response.status_code != 200:
    warn('Request: {}; Status code: {}'.format(requests, response.status_code))

# Parse the content of the request with BeautifulSoup
page = BeautifulSoup(response.text, 'lxml')
```

Example Notebook
Extracting all elements from the Webpage

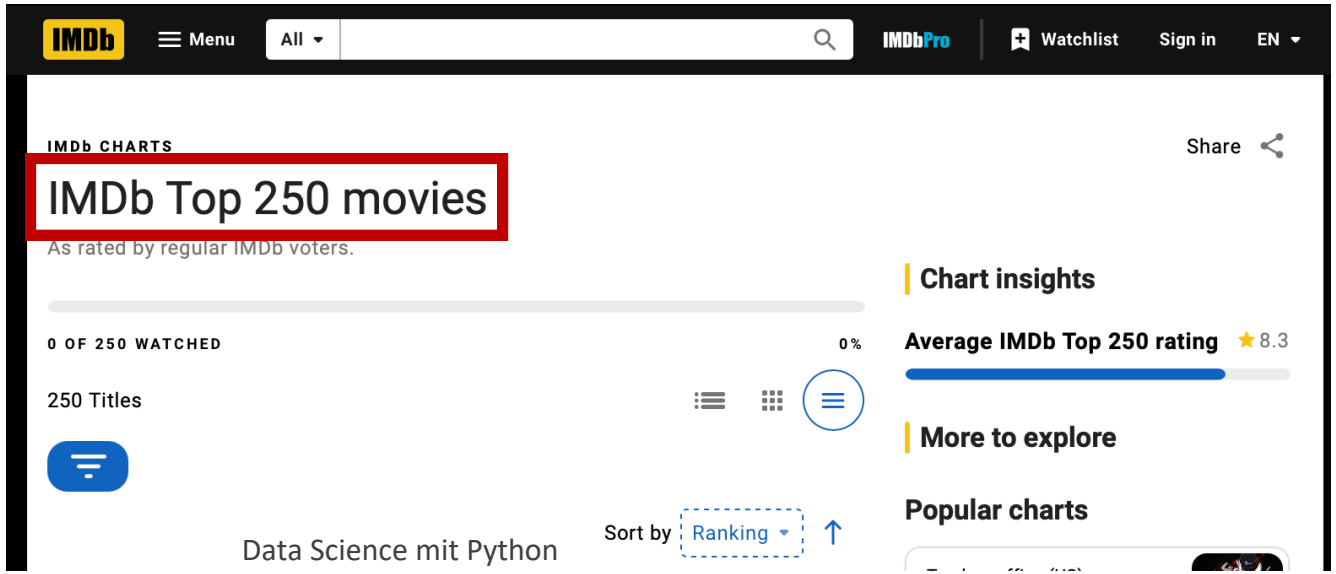



```python
# Parse the movies list
top_movies = []
for i, li in enumerate(page.find_all("li", class_="ipc-metadata-list-summary-item")):
    # parse Title
    title = li.find("h3", class_="ipc-title__text").text

    # parse HREF
    href = li.find("a", class_="ipc-title-link-wrapper")["href"].split('?')[0]

    # parse Rating
    rating = li.find("span", class_="ipc-rating-star--rating").text

    top_movies.append(
        {
            "title" : title,
            "rating" : rating,
            "href" : href,
        }
    )

movies = pd.DataFrame(top_movies)
movies
```

Extract the element

Get the header

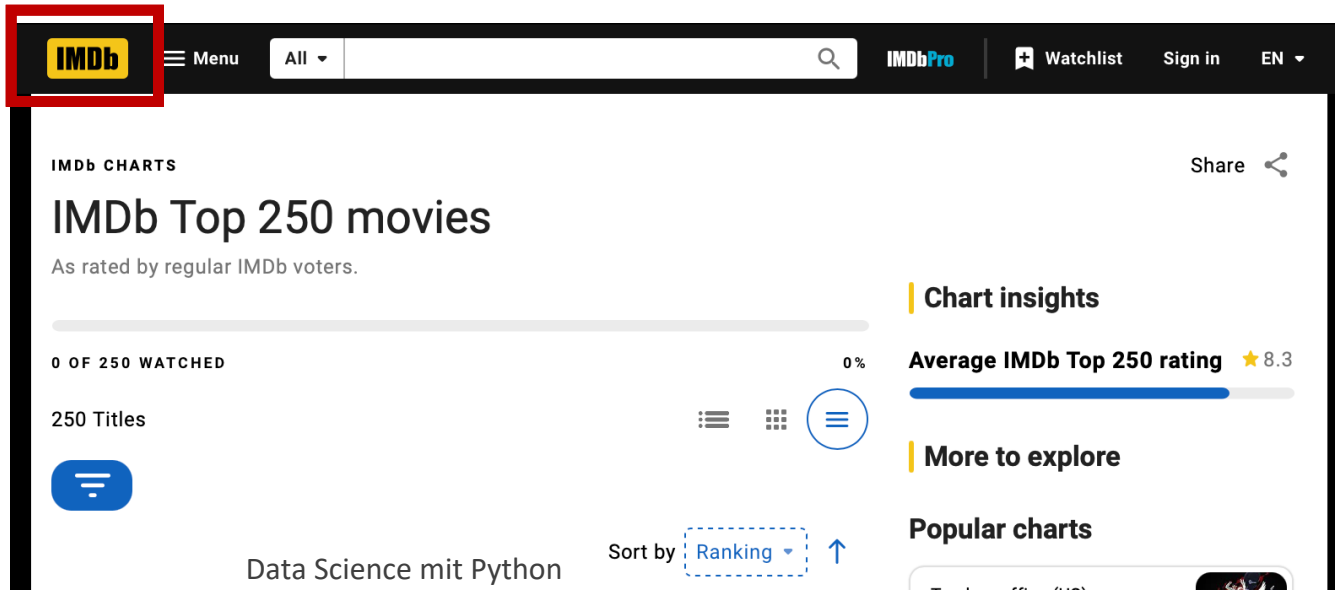
```
In [10]: page.h1
Out[10]: <h1 class="header">IMDb Top 250 Movies</h1>
```



Get the first svg-image

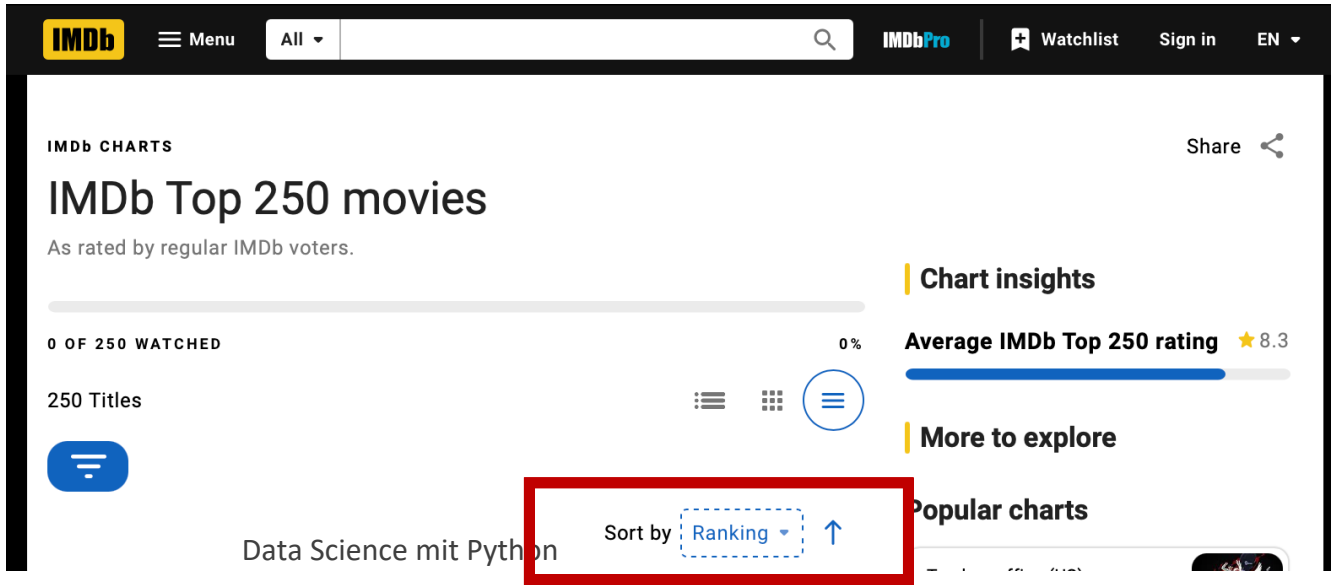
In [31]:
```
page.svg
```

Out[31]:
```
<svg class="ipc-icon ipc-icon--menu" fill="currentColor" height="24" id="iconContext-menu" role="presentation" viewbox="0 0 24 24" width="24" xmlns="http://www.w3.org/2000/svg"><path d="M0 0h24v24H0V0z" fill="none"></path><path d="M4 18h16c.55 0 1-.45 1-1s-.45-1-1-1H4c-.55 0-1 .45-1 1s.45 1 1 1zm0-5h16c.55 0 1-.45 1-1s-.45-1-1-1H4c-.55 0-1 .45-1 1s.45 1 1 1zM3 7c0 .55.45 1 1 1h16c.55 0 1-.45 1-1s-.45-1-1-1H4c-.55 0-1 .45-1 1z"></path></svg>
```



```
page.find_all("option")
```

```
[<option selected="" value="RANKING">Ranking</option>,
 <option value="USER_RATING">IMDb rating</option>,
 <option value="RELEASE_DATE">Release date</option>,
 <option value="USER_RATING_COUNT">Number of ratings</option>,
 <option value="TITLE_REGIONAL">Alphabetical</option>,
 <option value="POPULARITY">Popularity</option>,
 <option value="RUNTIME">Runtime</option>]
```



Find – ID and Class

- *Class* and *id* attributes allow to identify an element

- This is button
- Beautiful Soup:

```
soup.find(”div", id=”button").text
```

- Awe-inspiring paragraph
- Beautiful Soup:

```
soup.find(”p", class_=”awesome").text
```

Find

- We can access non-standard attributes

- Awe-inspiring paragraph</p>
  - Beautiful Soup:

```
soup.find(”div", attrs={”test_id": ”button"}).text
```

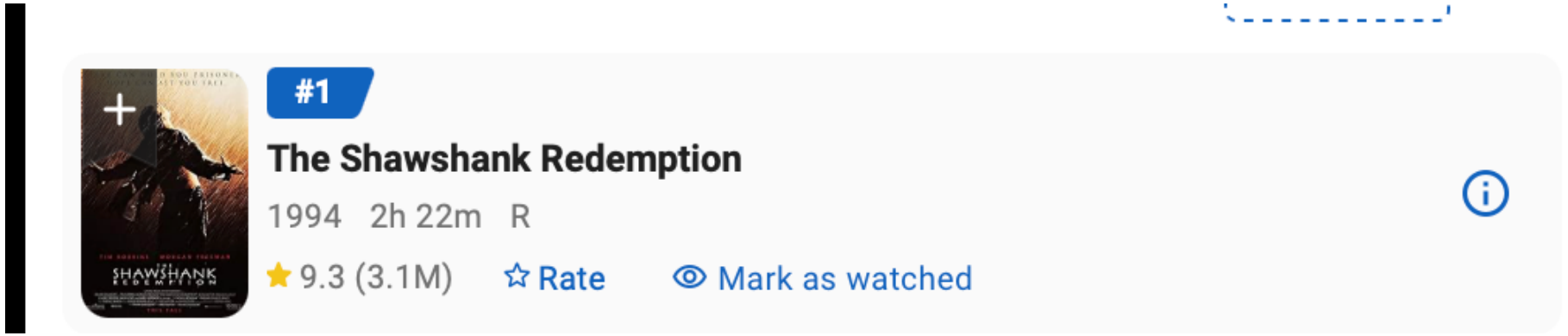
Find one element by class

[7]:
```
page.find("div", class_="cli-title")
```

[7]:
```
<div class="ipc-title ipc-title--base ipc-title--title ipc-title--title--reduced ipc-
title-link-no-icon ipc-title--on-textPrimary sc-87337ed2-2 dRlLYG cli-title with-marg
in"><a class="ipc-title-link-wrapper" href="/title/tt0111161/?ref_=chttp_t_1" tabinde
x="0"><h3 class="ipc-title__text ipc-title__text--reduced">1. The Shawshank Redemptio
n</h3></a></div>
```

[8]:
```
page.find("div", class_="cli-title").a.text
```

[8]:
```
'1. The Shawshank Redemption'
```



Find all elements by class

```
[9]: for td in page.find_all("div", clas
         print (td.a.text)
```

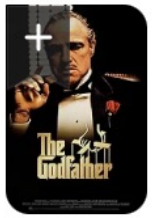
```
1. The Shawshank Redemption
2. The Godfather
3. The Dark Knight
4. The Godfather Part II
5. 12 Angry Men
6. The Lord of the Rings: The Retu
7. Schindler's List
8. The Lord of the Rings: The Fello
9. Pulp Fiction
```

#1

The Shawshank Redemption

1994 2h 22m R

9.3 (3.1M) Rate Mark as watched



#2

The Godfather

1972 2h 55m R

9.2 (2.2M) Rate Mark as watched

#3

The Dark Knight

2008 2h 32m PG-13

9.1 (3.1M) Rate Mark as watched

#4

The Godfather Part II

1974 3h 22m R

9.0 (1.5M) Rate Mark as watched

#5

12 Angry Men

1957 1h 36m Approved

9.0 (953K) Rate Mark as watched

#6

The Lord of the Rings: The Return of the King

2003 3h 21m PG-13

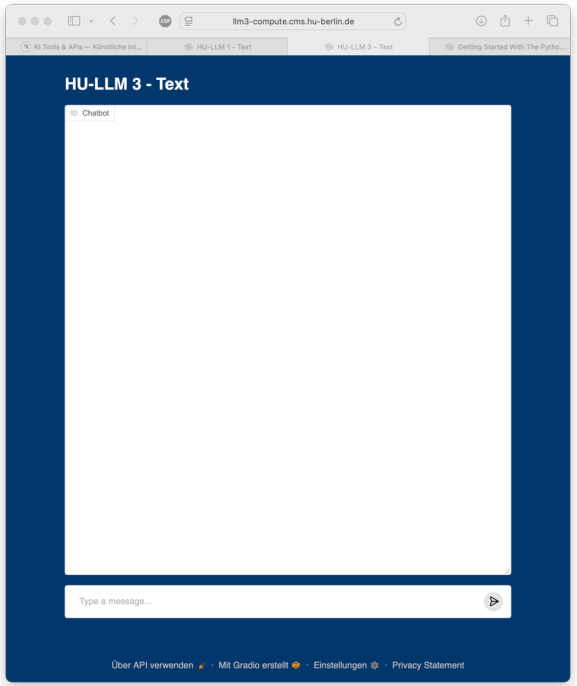
9.0 (2.1M) Rate Mark as watched

Agenda

- Questions?
- 1. Exercise
- Background
  - Beautiful Soup
  - Pandas
  - OpenAI / Mistral
  - Vibe Coding

Mistral

- Is an LLM
  - HU hosts two models, but only one is accessible
  - ~~HU-LLM1-Text - Mistral Large (Context of 15k Tokens)~~
  - HU-LLM3-Text - Mistral Small (Context of ~~128k~~ 32k Tokens)



HU Mistral Chat Interface

- There is an API Access to Mistral
  - https://ki.cms.hu-berlin.de/de/ki-tools

- You need a HU internal IP-Address (via VPN or WLAN) for Mistral
  - Test via: https://amor.cms.hu-berlin.de/account/ip.cgi
  - **Sie befinden sich im HU-Netz!**

API-Call Key Ingrediens

- API Client Setup
  - Passes base_url and api_key
- Prompt construction
  - Explicit instructions what to do
- API Call for Chat Completion
  - The model used (i.e. llm3)
  - The message includes a system and user specification
  - Temperature 0 defines deterministic output

HU – Mistral API Access

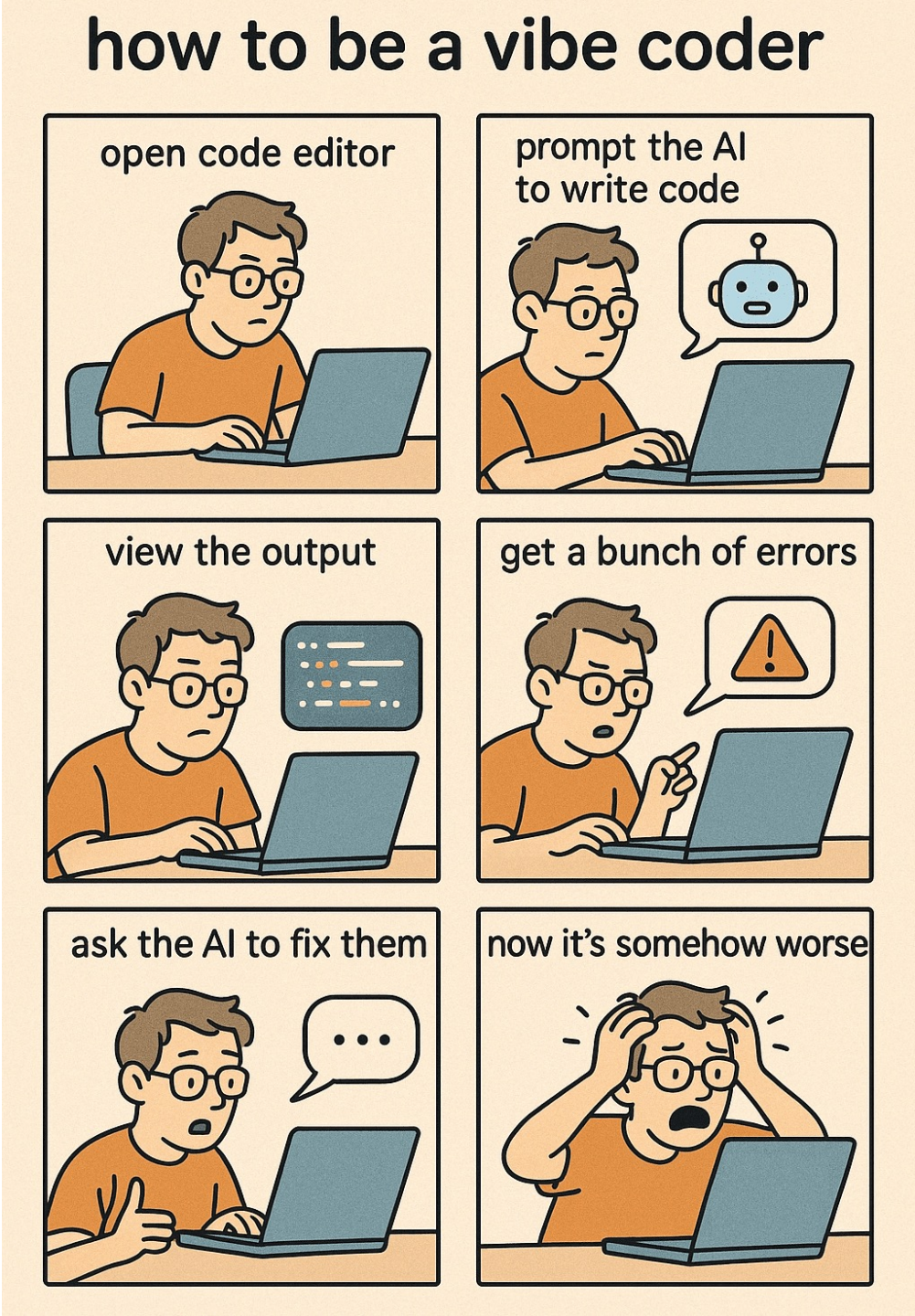
```
from openai import OpenAI
client = OpenAI(
        base_url="https://llm3-compute.cms.hu-berlin.de/v1/",
        api_key="required-but-not-used")

prompt = (  f"What's the sentiment of this review? {text}"
            f"You must answer with 'Positive' or 'Negative' or 'Neutral'.")

stream = client.chat.completions.create(
        model="llm3",
        messages=[
        {"role": "system", "content": "You are a precise sentiment classifier."},
        {"role": "user", "content": prompt}],
        max_tokens=500,             # length of the response
        stream=True,                # incrementally build the response
        temperature=0)              # creativity

for chunk in stream:
        if chunk.choices[0].delta.content is not None:
                content = chunk.choices[0].delta.content
                print(content, end='', flush=True)  # Print tokens as they arrive
```

Agenda

- Questions?
- 1. Exercise
- Background
  - Beautiful Soup
  - Pandas
  - OpenAI / Mistral
  - Vibe Coding
  - Streamlit

Vibe Coding

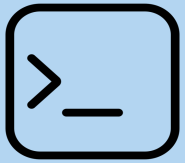
- Vibe coding is a **software development technique** that uses AI to generate code almost exclusively by entering text prompts in natural language
- The term was coined in February 2025 by Andrej Karpathy



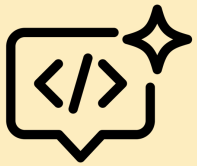
Vibe Coding with GenAI

App that analyzes product reviews

With GenAI:



“Create a Streamlit app that lets users paste in reviews, run a sentiment analysis and show the results as a pie chart”



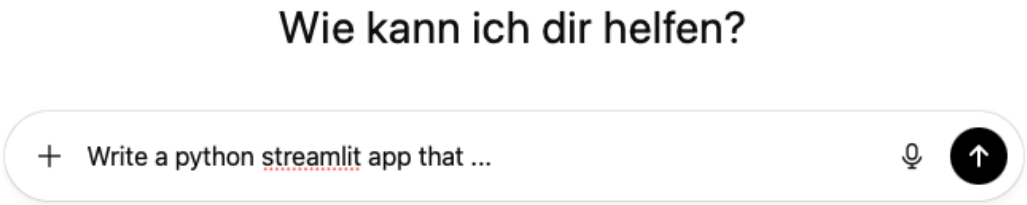
AI writes the code

You click **RUN** →
and you’re already **testing** your idea 🎉

⇨

- import libraries
- set up the interface
- run the analysis
- create the chart

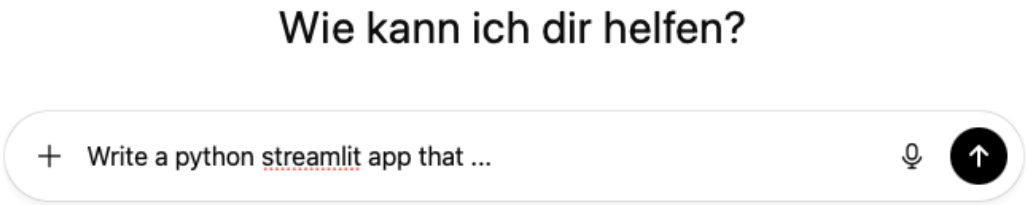
Vibe Coding



- You should be using an LLM (ChatGPT) to generate Python code
- The main challenge lies in crafting effective prompts

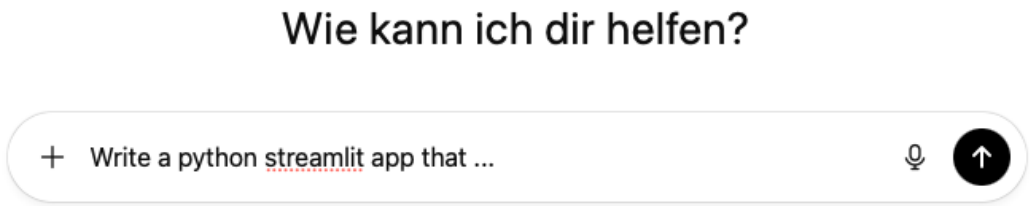
- Workflow:
  - Write a **clear and precise prompt** to generate the desired features
  - Copy the generated code and run it locally
  - Refine the code and add new features as needed
  - Use the LLM to help debug errors and improve the code iteratively

Vibe Coding



- Hints:
  - Do not put too many features into the first prompt
  - Keep adding features incrementally
  - Be **very precise** in what you want

Be as precise as possible



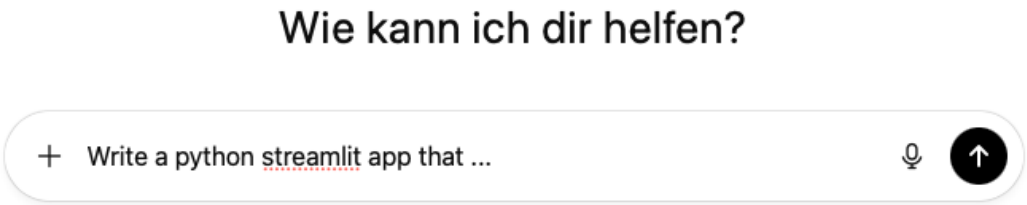
DON'T (might do what is intended or not):

- Offer an upload for a csv-file. The data should be displayed in tabular form.

DO (be very precise with your intentions):

- When the upload button is pressed, a dialog for uploading a csv-file should be offered.
- The data has the following keys and data types in brackets:
  url (string), title (string), ratingValue (float64), ratingCount (int64), year (int64), description (string), budget (object), gross (int64), duration (int64), genreList (list), countryList [...]
- After uploading the file, the data should be displayed in tabular form.

Be as precise as possible



DON'T (might do what is intended or not):

- It should show TOP-10 actors with most screen-time.

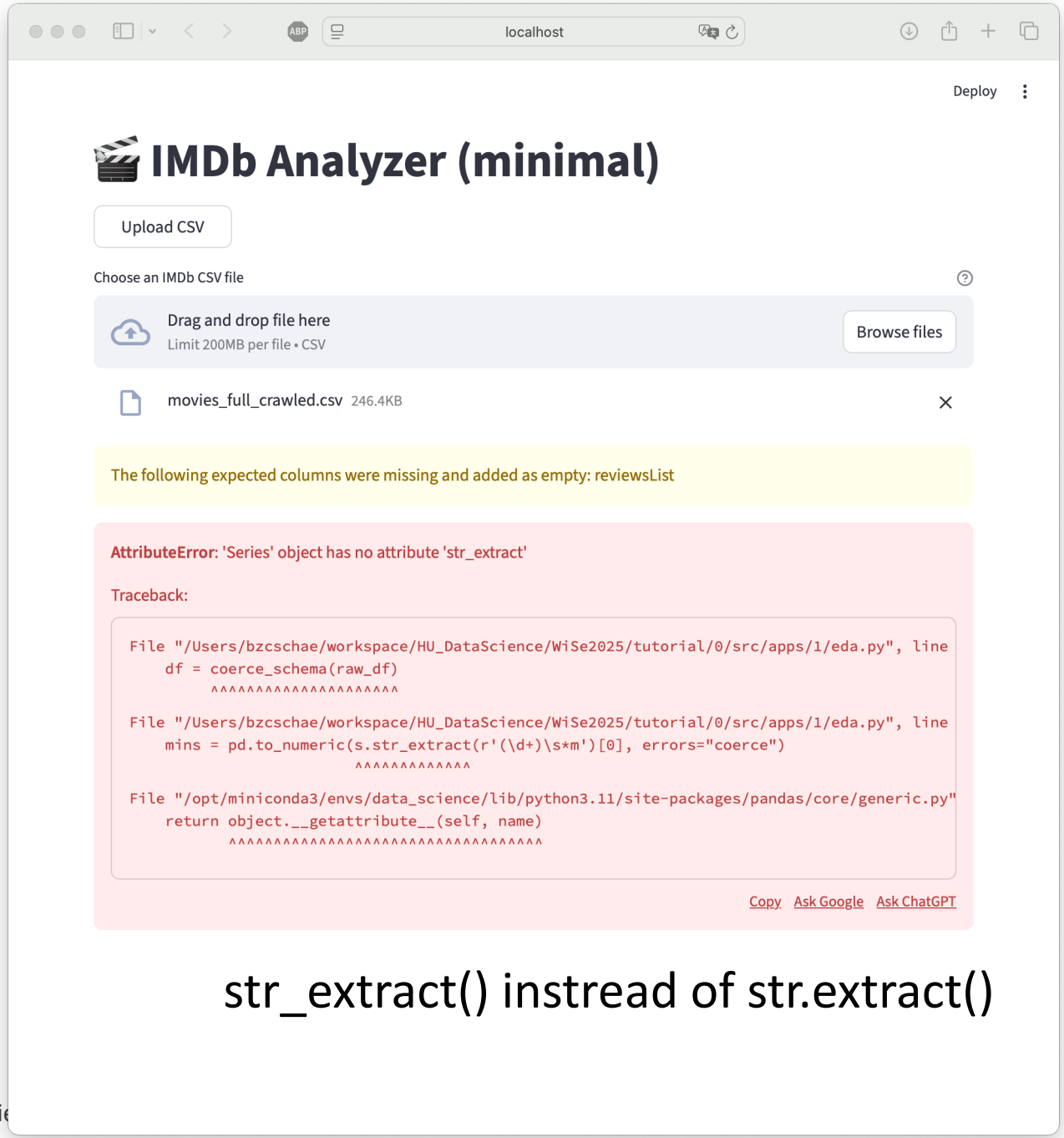
DO (be very precise with your intentions):

- It should show a **barplot** of the TOP-10 actors with the most screen-time, as **counted by total duration of the movies they are starring in**, sorted by screen-time.

Debugging

- If you run into a bug, make a screenshot or copy paste it into the LLM
- Ask the LLM to fix it

- If all goes wrong, you can try to debug it, too.

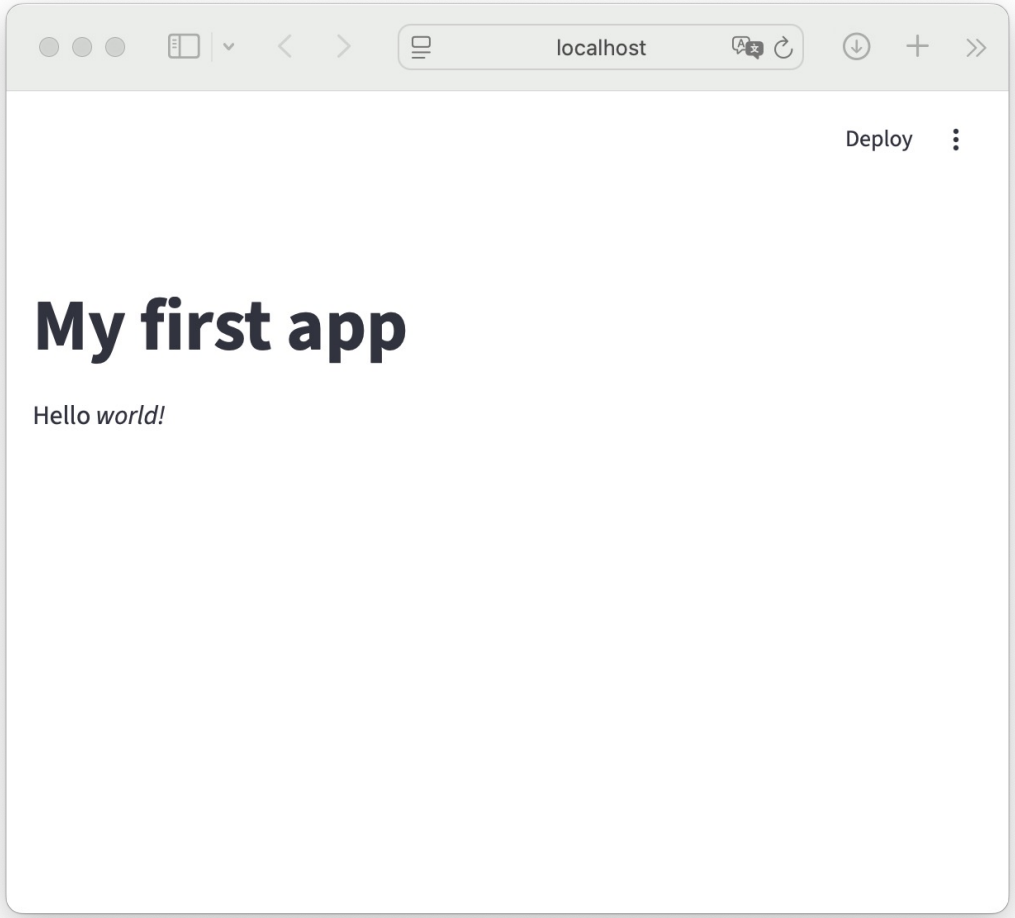


str_extract() instread of str.extract()

Agenda

- Questions?
- 1. Exercise
- Background
  - Beautiful Soup
  - Pandas
  - OpenAI / Mistral
  - Vibe Coding
  - Streamlit

Streamlit – Hello World



- Streamlit is an open-source framework for data scientists to write interactive apps in python
  - https://streamlit.io

- Hello World-App:

```
import streamlit as st
st.write("""
    # My first app
    Hello *world!*
""")
```

- Store in a file `app.py`, and run with `streamlit run app.py`

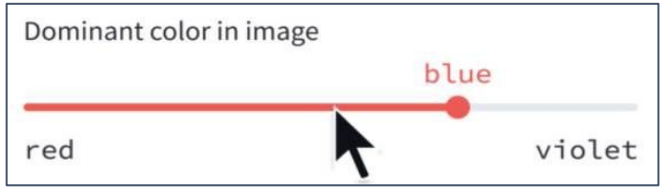
Widgets

st.text_input()

st.button()

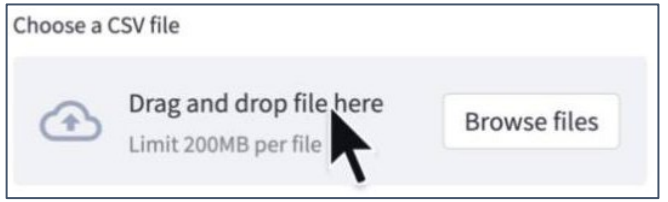
st.selectbox()

st.slider()



st.checkbox()

st.file_uploader()

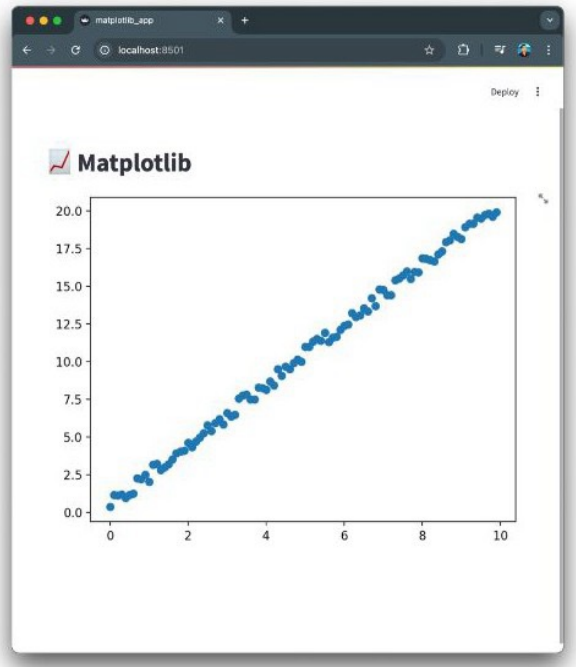


st.spinner()

https://docs.streamlit.io/develop/api-reference/widgets

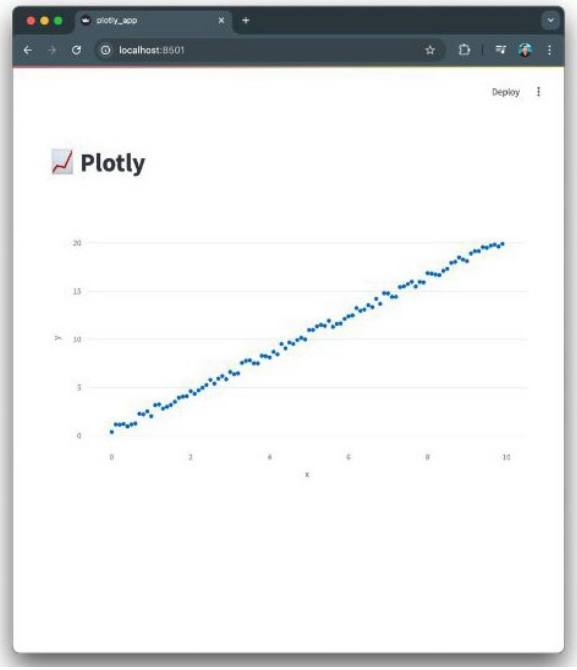
Streamlit – Plotting Backends

matplotlib



st.pyplot()

plotly

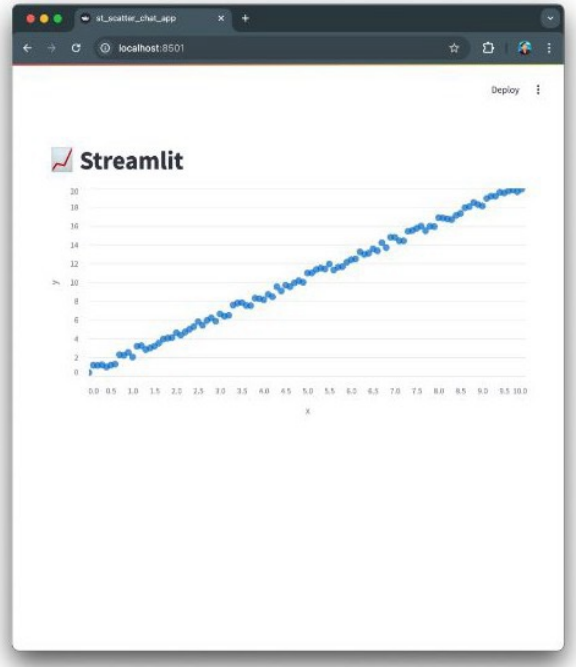


st.plotly_chart()

Altair

st.altair_chart (chart, use_container_width= True)

Streamlit



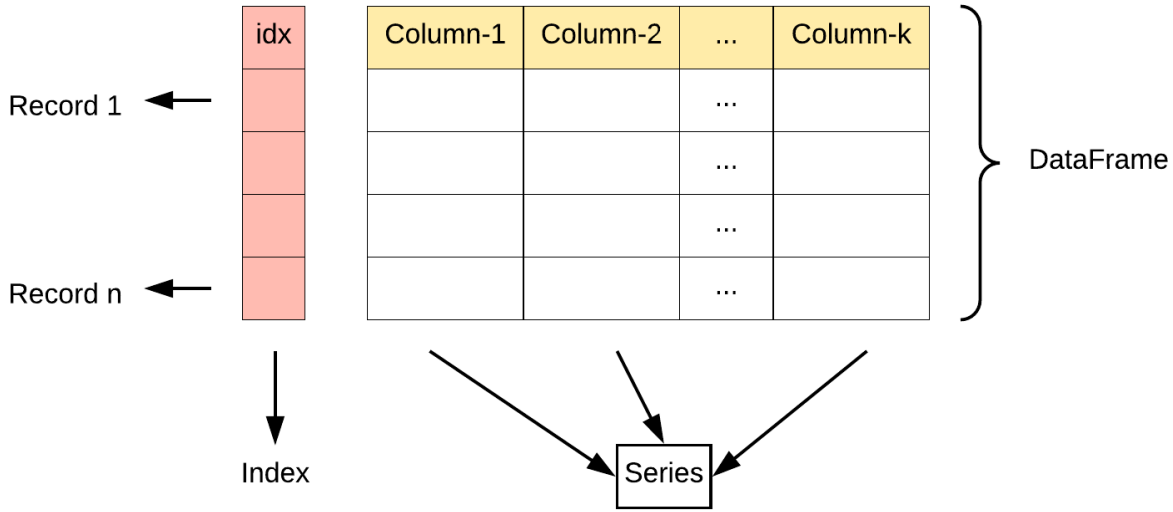
st.scatter_chart(df)

Agenda

- Questions?
- 1. Exercise
- Background
  - Beautiful Soup
  - Pandas
  - OpenAI / Mistral
  - Vibe Coding
  - Streamlit

Pandas DataFrame

- Tabular data structure comprised of rows and columns
  - Conceptually like a spreadsheet or a database table (without a primary key)
- Represents a collection of Series (the columns) that share an index



Data Wrangling

with pandas Cheat Sheet
http://pandas.pydata.org

Pandas API Reference Pandas User Guide

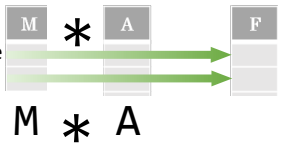
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4, 5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
N	v			
D	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

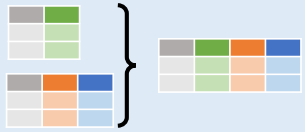
```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change layout, sorting, reindexing, renaming

pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames



pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)
Order rows by values of a column (high to low).

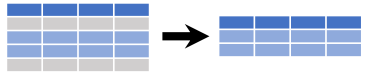
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length', 'Height'])
Drop columns from DataFrame

Subset Observations - rows



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10) Randomly select n rows.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

Subset Variables - columns

df[['width', 'length', 'species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

Using query

query() allows Boolean expressions for filtering rows.

df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")', engine="python")

Subsets - rows and columns

Use **df.loc[]** and **df.iloc[]** to select only rows, only columns or both.
Use **df.at[]** and **df.iat[]** to access a single value by row and column.
First index selects rows, second index columns.

df.iloc[10:20]
Select rows 10-20.

df.iloc[:, [1, 2, 5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[:, 'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.loc[df['a'] > 10, ['a', 'c']]
Select rows meeting logical condition, and only the specific columns .

df.iat[1, 2] Access single value by index

df.at[4, 'A'] Access single value by label

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

Cheatsheet for pandas (http://pandas.pydata.org/ originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

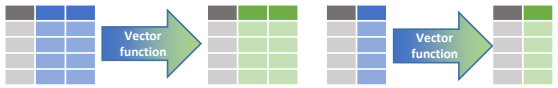
`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

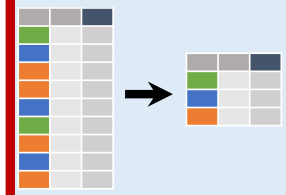
`max(axis=1)`
Element-wise max.

`min(axis=1)`
Element-wise min.

`clip(lower=-10,upper=10)`
Trim values at input thresholds

`abs()`
Absolute value.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group.
Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

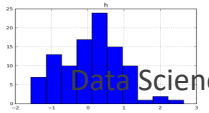
Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

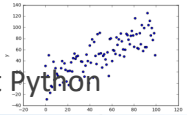
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h'`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

`pd.merge(ydf, zdf, how='outer', indicator=True) .query('_merge == "left_only"') .drop(columns=['_merge'])`
Rows that appear in ydf but not zdf (Setdiff).

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

JSON to DataFrame

- Loading a JSON file is as simple as calling `read_json()`:

```
athletes = pd.read_json(URL)
```

- List types are simply stored as lists in each cell

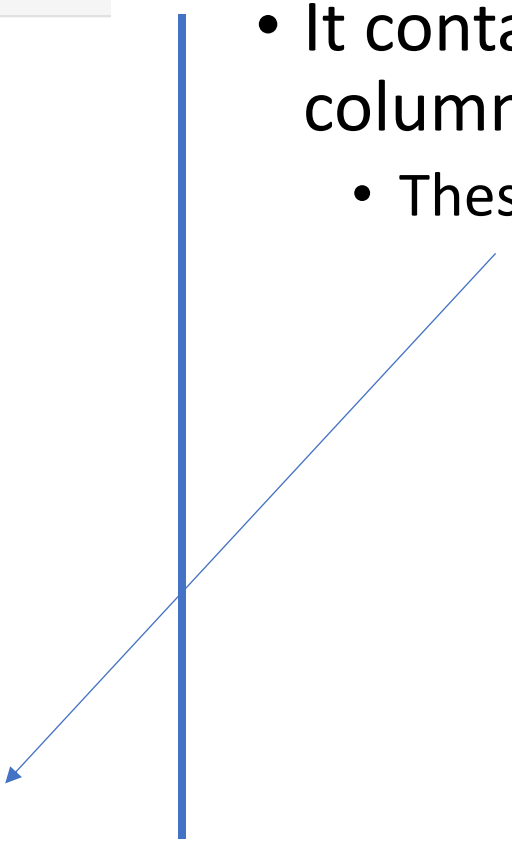
The Dataset

	url	title	ratingValue	ratingCount	year	description	budget	gross	duration	genreList	countryList	castList	characterList	directorList
0	/title/tt0111161/	Die Verurteilten	9.3	2700000	1994	Two imprisoned men bond over a number of years...	$25,000,000 (estimated)	28884504	142	[Drama]	[United States]	[Tim Robbins, Morgan Freeman, Bob Gunton, Will...	[Andy Dufresne, Ellis Boyd 'Red' Redding, Ward...	[Frank Darabont]
1	/title/tt0068646/	Der Pate	9.2	1800000	1972	The aging patriarch of an organized crime dyna...	$6,000,000 (estimated)	250341816	175	[Crime, Drama]	[United States]	[Marlon Brando, Al Pacino, James Caan, Diane K...	[Don Vito Corleone, Michael, Sonny, Kay Adams,...	[Francis Ford Coppola]
...														

The Dataset

```
movies = movies.convert_dtypes()
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 250 entries, 0 to 249
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   url            250 non-null    string
 1   title          250 non-null    string
 2   ratingValue    250 non-null    Float64
 3   ratingCount    250 non-null    Int64
 4   year           250 non-null    Int64
 5   description    250 non-null    string
 6   budget         250 non-null    object
 7   gross          250 non-null    Int64
 8   duration       250 non-null    Int64
 9   genreList      250 non-null    object
 10  countryList    250 non-null    object
 11  castList       250 non-null    object
 12  characterList  250 non-null    object
 13  directorList   250 non-null    object
dtypes: Float64(1), Int64(4), object(6), string(3)
memory usage: 30.5+ KB
```



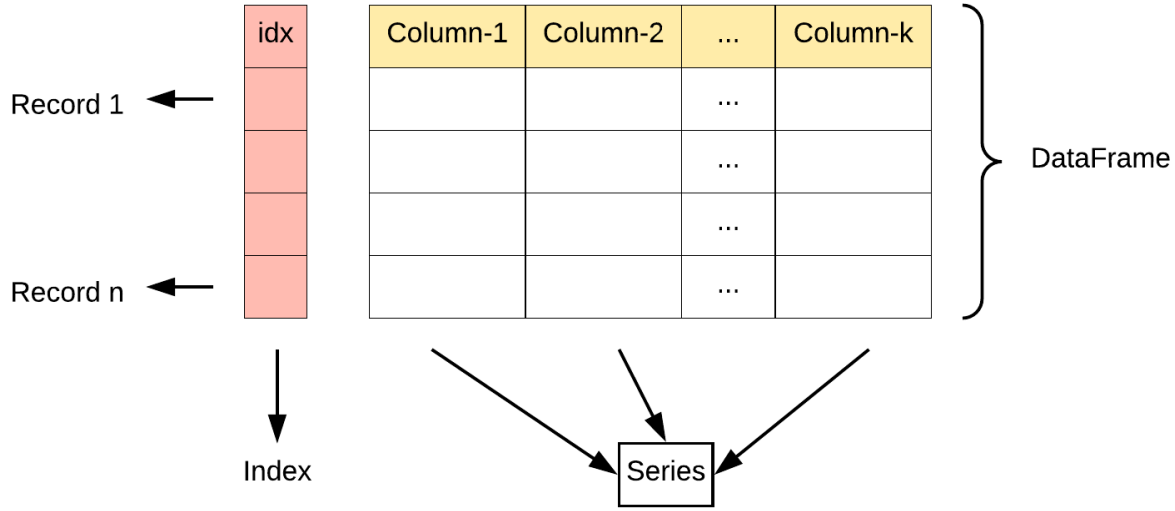
- It contains List and String-columns
  - These are stored as "`object`"

Slicing

Get the first 2 entries from the dataframe

```
result = movies.iloc[:2]
result[["title", "url"]]
```

	title	url
0	Die Verurteilten	/title/tt0111161/
1	Der Pate	/title/tt0068646/



Exploratory Data Analysis

- We want to find the: **TOP-5 movies by gross**
  - We must apply sorting and ranking

TOP-5 Ranking

TOP-5 movies by largest gross

```
result = movies.sort_values(by="gross", ascending=False).iloc[:5]
result[["title", "gross"]]
```

	title	gross
77	Avengers: Endgame	2797501328
63	Avengers: Infinity War	2048359754
140	Spider-Man: No Way Home	1920618834
78	Top Gun: Maverick	1484620609
182	Harry Potter und die Heiligtümer des Todes - T...	1342359942

Exploratory Data Analysis

- We want to compute: **Total Number of Movies with a rating above or equal to 9**

- We have to apply a boolean filter

Boolean Filtering

```
movies["ratingValue"] >= 9.0
```

	title	ratingValue	condition
0	Die Verurteilten	9.3	**True**
1	Der Pate	9.2	**True**
2	The Dark Knight	9.0	**True**
3	Der Pate 2	9.0	**True**
4	Die zwölf Geschworenen	9.0	**True**
...	...	...	**...**
245	Uzala, der Kirgise	8.2	**False**
246	The Help	8.1	**False**
247	Gandhi	8.1	**False**
248	Aladdin	8.0	**False**
249	Der Gigant aus dem All	8.1	**False**

Boolean Filtering

```
movies["ratingValue"] >= 9.0
```

	title	ratingValue
0	Die Verurteilten	9.3
1	Der Pate	9.2
2	The Dark Knight	9.0
3	Der Pate 2	9.0
4	Die zwölf Geschworenen	9.0
...	...	...
245	Uzala, der Kirgise	8.2
246	The Help	8.1
247	Gandhi	8.1
248	Aladdin	8.0
249	Der Gigant aus dem All	8.1

condition
True
True
True
True
...
False
False
False
False
False

```
movies[ movies["ratingValue"] >= 9.0 ]
```

	title	ratingValue
0	Die Verurteilten	9.3
1	Der Pate	9.2
2	The Dark Knight	9.0
3	Der Pate 2	9.0
4	Die zwölf Geschworenen	9.0
5	Schindlers Liste	9.0
6	Der Herr der Ringe: Die Rückkehr des Königs	9.0

Boolean Filtering

```
# Using shape
result = movies[ movies["ratingValue"] >= 9.0 ].shape[0]
print(result)
```

```
7
```

List-Cells

- Assume we want to group movies by genre
  - `genreList` is a list-type

- Unfortunately, we can not directly work with complex data types in cells such as lists

- Pandas `df.explode()`:
  transform each element of a list-like cell to a row, replicating rows
  https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.explode.html

Handling List-Cells

```
# First create duplicates to unroll the genre-list into own rows.
df[df.title=="Der Pate"][["title", "genreList"]]
```

	title	genreList
1	Der Pate	[Crime, Drama]

```
# First create duplicates to unroll the genre-list into own rows.
df = movies.explode('genreList')
df[df.title=="Der Pate"][["title", "genreList"]]
```

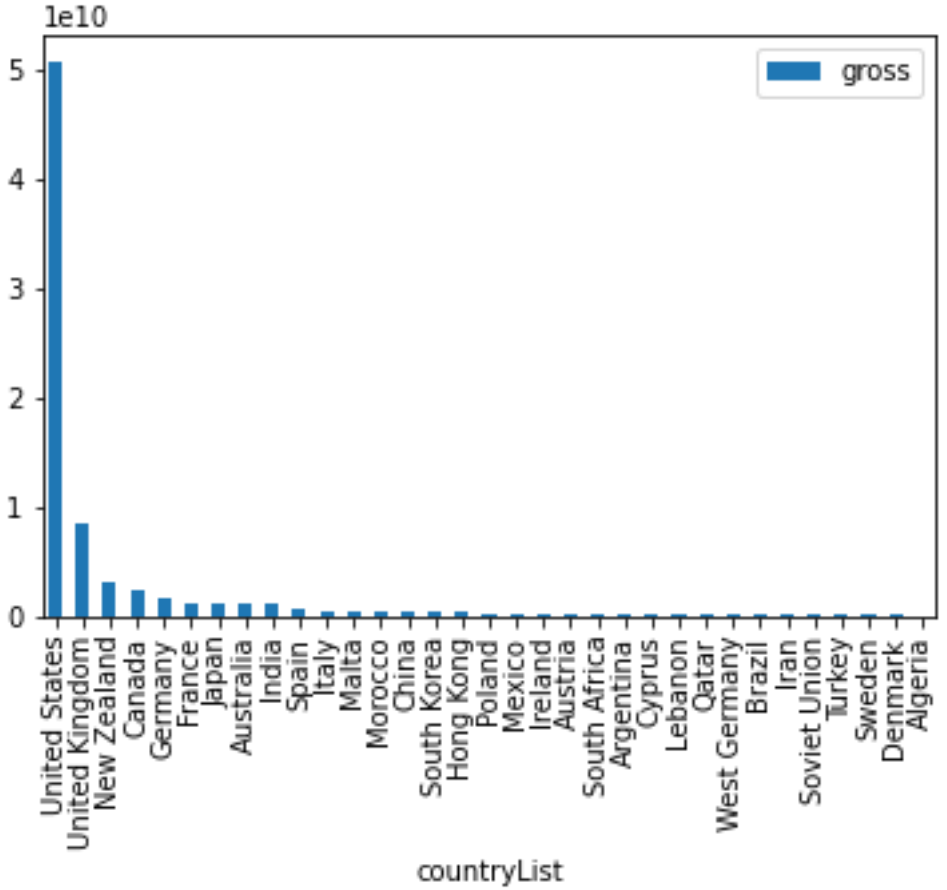
	title	genreList
1	Der Pate	Crime
1	Der Pate	Drama

Exploratory Data Analysis

- We want to find the

  Sum of Gross by all Countries of Origin ordered by total gross

- We apply **grouping, sum and sorting**
  - More about this in the following lectures



First „Explode“ Dataset

title	countryList	gross
Die Verurteilten	[United States]	28884504
Der Pate	[United States]	250341816
The Dark Knight	[United States, United Kingdom]	1006234167
Der Pate 2	[United States]	47961919
Die zwölf Geschworenen	[United States]	955
...	...	...
Uzala, der Kirgise	[Soviet Union, Japan]	14480
The Help	[United States, India]	216639112
Gandhi	[United Kingdom, India, United States, South A...	52767889
Aladdin	[United States]	504050219
Der Gigant aus dem All	[United States]	23335817

Raw data with list type

m=250

title	countryList	gross
Die Verurteilten	**United States**	28884504
Der Pate	**United States**	250341816
The Dark Knight	**United States**	1006234167
The Dark Knight	**United Kingdom**	1006234167
Der Pate 2	**United States**	47961919
...	**...**	...
Gandhi	**India**	52767889
Gandhi	**United States**	52767889
Gandhi	**South Africa**	52767889
Aladdin	**United States**	504050219
Der Gigant aus dem All	**United States**	23335817

Adds duplicates

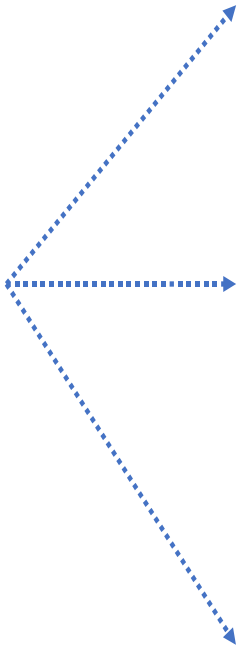
m=360

Grouping and Aggregation

title	countryList
Die Verurteilten	United States
Der Pate	United States
The Dark Knight	United States
The Dark Knight	United Kingdom
Der Pate 2	United States
...	...
Gandhi	India
Gandhi	United States
Gandhi	South Africa
Aladdin	United States
Der Gigant aus dem All	United States

m=360

1. group-by countryList



title	countryList
Die Verurteilten	**United States**
Der Pate	**United States**
The Dark Knight	**United States**
Der Pate 2	**United States**
Die zwölf Geschworenen	**United States**

...

title	countryList
Fight Club	**Germany**
City of God	**Germany**
Der Pianist	**Germany**
Die üblichen Verdächtigen	**Germany**
Das Leben der Anderen	**Germany**
Inglourious Basterds	**Germany**

Grouping and Aggregation

title	countryList
Die Verurteilten	United States
Der Pate	United States
The Dark Knight	United States
The Dark Knight	United Kingdom
Der Pate 2	United States
...	...
Gandhi	India
Gandhi	United States
Gandhi	South Africa
Aladdin	United States
Der Gigant aus dem All	United States

m=360

1. group-by countryList

title	countryList
Die Verurteilten	United States
Der Pate	United States
The Dark Knight	United States
Der Pate 2	United States
Die zwölf Geschworenen	United States

...

title	countryList
Fight Club	Germany
City of God	Germany
Der Pianist	Germany
Die üblichen Verdächtigen	Germany
Das Leben der Anderen	Germany
Inglourious Basterds	Germany

2. Sum of gross

countryList	gross
United States	50602793883

countryList	gross
Germany	1517359823

Grouping and Aggregation

title	countryList
Die Verurteilten	United States
Der Pate	United States
The Dark Knight	United States
The Dark Knight	United Kingdom
Der Pate 2	United States
...	...
Gandhi	India
Gandhi	United States
Gandhi	South Africa
Aladdin	United States
Der Gigant aus dem All	United States

m=360

1. group-by countryList

title	countryList
Die Verurteilten	United States
Der Pate	United States
The Dark Knight	United States
Der Pate 2	United States
Die zwölf Geschworenen	United States

...

title	countryList
Fight Club	Germany
City of God	Germany
Der Pianist	Germany
Die üblichen Verdächtigen	Germany
Das Leben der Anderen	Germany
Inglourious Basterds	Germany

2. Sum of gross

countryList	gross
United States	50602793883

countryList	gross
Germany	1517359823

3. Combine (and sort by group keys)

countryList	gross
United States	50602793883
United Kingdom	8472053024
New Zealand	2992545042
Canada	2262281025
Germany	1517359823
France	1211493670
Japan	1157353754
Australia	1048162339
India	1040364220
Spain	575195077
Italy	485297904
Malta	465380802
Morocco	465380802
China	321752656
South Korea	317001314
Hong Kong	291480452
Poland	120072577
Mexico	104770499
Ireland	101198620
Austria	98168960
South Africa	86650132
Argentina	66141126
Cyprus	64417003
Lebanon	64417003
Qatar	64417003
West Germany	51741563
Brazil	30680793
Iran	23860009
Soviet Union	20944128
Turkey	18612999
Sweden	16348559
Denmark	15886373
Algeria	962002

Sum of Gross by all Countries of Origin

Applying grouping and sorting to the data

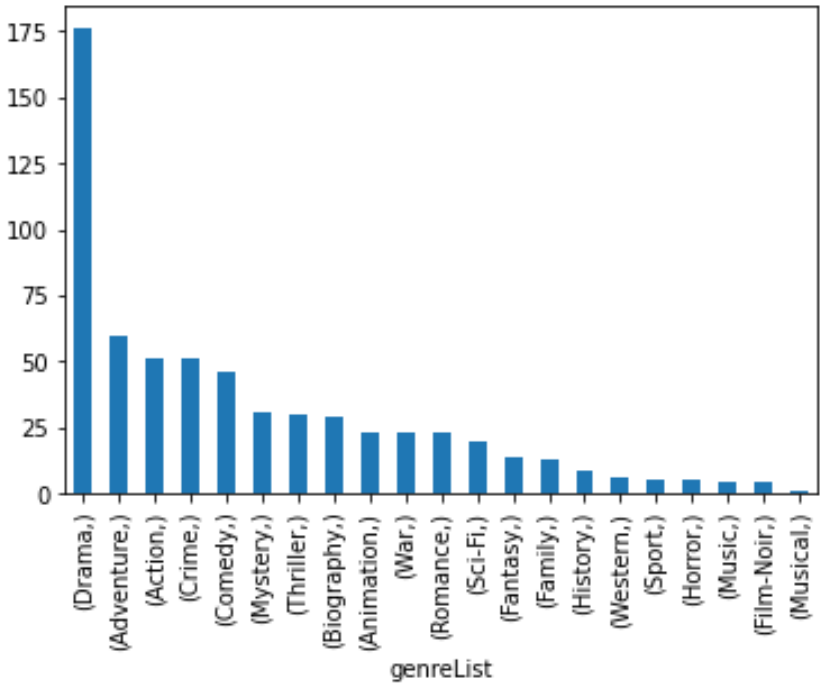
```
# First create duplicates to unroll the country-list into own rows.
df = movies.explode('countryList')

# Group the data, aggregate and sort the result
result = df.groupby("countryList")[["gross"]]\
        .sum() \
        .sort_values(by="gross", ascending=False)
display(result)
```

countryList	gross
United States	50602793883
United Kingdom	8472053024
New Zealand	2992545042
Canada	2262281025
Germany	1517359823
France	1211493670

The result is sorted by CountryList by default (grouby attribute)

Query

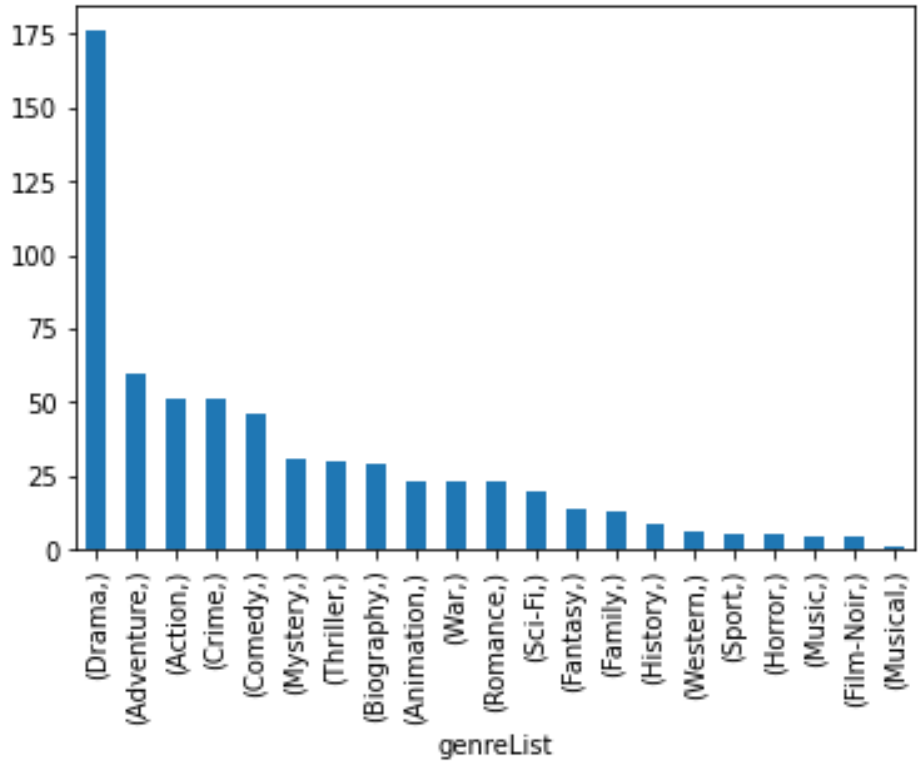
- We want to find the
  - Total count of movies by genre
- We apply **grouping and counting**
  - There is a short way for this in pandas



Count of Movies by Genre

Count the number of movies in each genre.

```
# First create duplicates to unroll the country-list into own rows,
# then count the number of movies in each genre
result = movies.explode('genreList')[["genreList"]].value_counts()

# Finally, plot the result
_ = result.plot(kind="bar")
```



Pandas `df.value_counts()` returns counts of unique values

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.value_counts.html

Questions?