Data Gathering

Sources of Data

A vast amount of historical data can be found in files such as:

- MS Word documents
- Emails
- Spreadsheets
- MS PowerPoints
- PDFs
- HTML
- and plaintext files

Public and Private Archives

CSV, JSON, and XML files use plaintext, a common format, and are compatible with a wide range of applications

The Web can be mined for data using a web scraping application

The IoT uses sensors create data

Sensors in smartphones, cars, airplanes, street lamps, and home appliances capture raw data

Open Data and Private Data

1. Open Data

> The Open Knowledge Foundation describes Open Data as “any content, information or data that people are free to use, reuse, and redistribute without any legal, technological, or social restriction.”

1. Private Data

> Data related to an expectation of privacy and regulated by a particular country/government

Structured and Unstructured Data

1. Structured Data

> Data entered and maintained in fixed fields within a file or record Easily entered, classified, queried, and analyzed Relational databases or spreadsheets

2. Unstructured Data Lacks organization

> Raw data Photo contents, audio, video, web pages, blogs, books, journals, white papers, PowerPoint presentations, articles, email, wikis, word processing documents, and text in general

Example of gathering image data using webcam

Note: Run this snippet using local jupyter notebook

In [ ]:

```
import cv2
from google.colab.patches import cv2_imshow
key = cv2. waitKey(1)
webcam = cv2.VideoCapture(0)
while True:
    try:
        check, frame = webcam.read()
        print(check) #prints true as long as the webcam is running
        print(frame) #prints matrix values of each framecd
        cv2.imshow("Capturing", frame)
        key = cv2.waitKey(1)
        if key == ord('s'):
            cv2.imwrite(filename='saved_img.jpg', img=frame)
            webcam.release()
            img_new = cv2.imread('saved_img.jpg', cv2.IMREAD_GRAYSCALE)
            img_new = cv2.imshow("Captured Image", img_new)
            cv2.waitKey(1650)
            cv2.destroyAllWindows()
            print("Processing image...")
            img_ = cv2.imread('saved_img.jpg', cv2.IMREAD_ANYCOLOR)
            print("Converting RGB image to grayscale...")
            gray = cv2.cvtColor(img_, cv2.COLOR_BGR2GRAY)
            print("Converted RGB image to grayscale...")
            print("Resizing image to 28x28 scale...")
            img_ = cv2.resize(gray,(28,28))
            print("Resized...")
```

```
            img_resized = cv2.imwrite(filename='saved_img-final.jpg', img=img_)
            print("Image saved!")

            break
        elif key == ord('q'):
            print("Turning off camera.")
            webcam.release()
            print("Camera off.")
            print("Program ended.")
            cv2.destroyAllWindows()
            break

    except(KeyboardInterrupt):
        print("Turning off camera.")
        webcam.release()
        print("Camera off.")
        print("Program ended.")
        cv2.destroyAllWindows()
        break
```

Example of gathering voice data using microphone

Note: Run the snippet of codes using local jupyter notebook

In [ ]:
```
!pip3 install sounddevice
```

```
Requirement already satisfied: sounddevice in /usr/local/lib/python3.7/dist-packages (0.4.1)
Requirement already satisfied: CFFI>=1.0 in /usr/local/lib/python3.7/dist-packages (from sounddevice) (1.14.5)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from CFFI>=1.0->sounddevice) (2.20)
```

In [ ]:
```
!pip3 install wavio
```

```
Requirement already satisfied: wavio in /usr/local/lib/python3.7/dist-packages (0.0.4)
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from wavio) (1.19.5)
```

In [ ]:
```
!pip3 install scipy
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scipy) (1.19.5)
```

In [ ]:
```
!apt-get install libportaudio2
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
The following NEW packages will be installed:
  libportaudio2
0 upgraded, 1 newly installed, 0 to remove and 34 not upgraded.
Need to get 64.6 kB of archives.
After this operation, 215 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 libportaudio2 amd64 19.6.0-1 [64.6 kB]
Fetched 64.6 kB in 0s (161 kB/s)
Selecting previously unselected package libportaudio2:amd64.
(Reading database ... 160706 files and directories currently installed.)
Preparing to unpack .../libportaudio2_19.6.0-1_amd64.deb ...
Unpacking libportaudio2:amd64 (19.6.0-1) ...
Setting up libportaudio2:amd64 (19.6.0-1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.2) ...
/sbin/ldconfig.real: /usr/local/lib/python3.7/dist-packages/ideep4py/lib/libmkldnn.so.0 is not a symbolic link
```

In [ ]:

```python
# import required libraries
import sounddevice as sd
from scipy.io.wavfile import write
import wavio as wv

# Sampling frequency
freq = 44100

# Recording duration
duration = 5

# Start recorder with the given values
# of duration and sample frequency
recording = sd.rec(int(duration * freq),
                   samplerate=freq, channels=2)

# Record audio for the given number of seconds
sd.wait()

# This will convert the NumPy array to an audio
# file with the given sampling frequency
write("recording0.wav", freq, recording)
```

```
# Convert the NumPy array to audio file
wv.write("recording1.wav", recording, freq, sampwidth=2)
```

Web Scraping

Web scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. The web scraping software may directly access the World Wide Web using the Hypertext Transfer Protocol or a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis.

Reference: link text

Image Scraping using BeautifulSoup and Request

In [ ]:
```
!pip install bs4
```

```
Requirement already satisfied: bs4 in /usr/local/lib/python3.7/dist-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.7/dist-packages (from bs4) (4.6.3)
```

In [ ]:
```
pip install requests
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (2.23.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests) (2020.12.5)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests) (3.0.4)
```

In [ ]:
```
import requests
from bs4 import BeautifulSoup

def getdata(url):
    r = requests.get(url)
    return r.text

htmldata = getdata("https://www.google.com/")
soup = BeautifulSoup(htmldata, 'html.parser')
for item in soup.find_all('img'):
    print(item['src'])
```

```
/images/branding/googlelogo/1x/googlelogo_white_background_color_272x92dp.png
```

In [ ]:
```
pip install selenium
```

```
Collecting selenium
  Downloading https://files.pythonhosted.org/packages/80/d6/4294f0b4bce4de0abf13e17190289f9d0613b0a44e5dd6a7f5ca98459853/selenium-3.141.0-py2.py3-none-any.whl (904kB)
     |████████████████████████████████| 911kB 3.8MB/s
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from selenium) (1.24.3)
Installing collected packages: selenium
Successfully installed selenium-3.141.0
```

Image Scraping using Selenium

Note: Run the snippet of code using local jupyter notebook

In [ ]:

```
!pip install selenium
!apt-get update # to update ubuntu to correctly run apt install
!apt install chromium-chromedriver
!cp /usr/lib/chromium-browser/chromedriver /usr/bin
import sys
sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')

from selenium import webdriver
import time
import requests
import shutil
import os
import getpass
import urllib.request
import io
import time
from PIL import Image

user = getpass.getuser()
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')
driver = webdriver.Chrome('chromedriver',chrome_options=chrome_options)

search_url = "https://www.google.com/search?q={q}&tbm=isch&tbs=sur%3Afc&hl=en&ved=0CAIQpwVqFwoTCKCa1c6s4-oCFQAAAAAdAAAAABAC&biw=1251&bih=568"
driver.get(search_url.format(q='Car'))

def scroll_to_end(driver):
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(5)#sleep_between_interactions
```

```python
def getImageUrls(name,totalImgs,driver):

    search_url = "https://www.google.com/search?q={q}&tbm=isch&tbs=sur%3Afc&hl=en&ved=0CAIQpwVqFwoTCKCa1c6s4-oCFQAAAAAdAAAAABAC&biw=1251&bih=568"
    driver.get(search_url.format(q=name))
    img_urls = set()
    img_count = 0
    results_start = 0

    while(img_count<totalImgs): #Extract actual images now

        scroll_to_end(driver)

        thumbnail_results = driver.find_elements_by_xpath("//img[contains(@class,'Q4LuWd')]")
        totalResults=len(thumbnail_results)
        print(f"Found: {totalResults} search results. Extracting links from{results_start}:{totalResults}")

        for img in thumbnail_results[results_start:totalResults]:

            img.click()
            time.sleep(2)
            actual_images = driver.find_elements_by_css_selector('img.n3VNCb')
            for actual_image in actual_images:
                if actual_image.get_attribute('src') and 'https' in actual_image.get_attribute('src'):
                    img_urls.add(actual_image.get_attribute('src'))

            img_count=len(img_urls)

            if img_count >= totalImgs:
                print(f"Found: {img_count} image links")
                break
            else:
                print("Found:", img_count, "looking for more image links ...")
                load_more_button = driver.find_element_by_css_selector(".mye4qd")
                driver.execute_script("document.querySelector('.mye4qd').click();")
                results_start = len(thumbnail_results)
    return img_urls


def downloadImages(folder_path,file_name,url):
    try:

        image_content = requests.get(url).content
    except Exception as e:
        print(f"ERROR - COULD NOT DOWNLOAD {url} - {e}")
    try:
```

```
        image_file = io.BytesIO(image_content)
        image = Image.open(image_file).convert('RGB')

        file_path = os.path.join(folder_path, file_name)

        with open(file_path, 'wb') as f:
            image.save(f, "JPEG", quality=85)
        print(f"SAVED - {url} - AT: {file_path}")
    except Exception as e:
        print(f"ERROR - COULD NOT SAVE {url} - {e}")

def saveInDestFolder(searchNames,destDir,totalImgs,driver):
    for name in list(searchNames):
        path=os.path.join(destDir,name)
        if not os.path.isdir(path):
            os.mkdir(path)
        print('Current Path',path)
        totalLinks=getImageUrls(name,totalImgs,driver)
        print('totalLinks',totalLinks)

    if totalLinks is None:
            print('images not found for :',name)

    else:
        for i, link in enumerate(totalLinks):
            file_name = f"{i:150}.jpg"
            downloadImages(path,file_name,link)

searchNames=['cat']
destDir=f'/content/drive/My Drive/Colab Notebooks/Dataset/'
totalImgs=5

saveInDestFolder(searchNames,destDir,totalImgs,driver)
```

```
Requirement already satisfied: selenium in /usr/local/lib/python3.7/dist-packages (4.0.0)
Requirement already satisfied: trio~=0.17 in /usr/local/lib/python3.7/dist-packages (from selenium) (0.19.0)
Requirement already satisfied: urllib3[secure]~=1.26 in /usr/local/lib/python3.7/dist-packages (from selenium) (1.26.7)
Requirement already satisfied: trio-websocket~=0.9 in /usr/local/lib/python3.7/dist-packages (from selenium) (0.9.2)
Requirement already satisfied: async-generator>=1.9 in /usr/local/lib/python3.7/dist-packages (from trio~=0.17->selenium) (1.10)
Requirement already satisfied: idna in /usr/local/lib/python3.7/dist-packages (from trio~=0.17->selenium) (2.10)
Requirement already satisfied: outcome in /usr/local/lib/python3.7/dist-packages (from trio~=0.17->selenium) (1.1.0)
Requirement already satisfied: sniffio in /usr/local/lib/python3.7/dist-packages (from trio~=0.17->selenium) (1.2.0)
Requirement already satisfied: sortedcontainers in /usr/local/lib/python3.7/dist-packages (from trio~=0.17->selenium) (2.4.0)
Requirement already satisfied: attrs>=19.2.0 in /usr/local/lib/python3.7/dist-packages (from trio~=0.17->selenium) (21.2.0)
Requirement already satisfied: wsproto>=0.14 in /usr/local/lib/python3.7/dist-packages (from trio-websocket~=0.9->selenium) (1.0.0)
Requirement already satisfied: pyOpenSSL>=0.14 in /usr/local/lib/python3.7/dist-packages (from urllib3[secure]~=1.26->selenium) (21.0.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from urllib3[secure]~=1.26->selenium) (2021.10.8)
Requirement already satisfied: cryptography>=1.3.4 in /usr/local/lib/python3.7/dist-packages (from urllib3[secure]~=1.26->selenium) (35.0.0)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.7/dist-packages (from cryptography>=1.3.4->urllib3[secure]~=1.26->selenium)
(1.15.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from cffi>=1.12->cryptography>=1.3.4->urllib3[secure]~=1.26->s
elenium) (2.20)
Requirement already satisfied: six>=1.5.2 in /usr/local/lib/python3.7/dist-packages (from pyOpenSSL>=0.14->urllib3[secure]~=1.26->selenium) (1.15.
0)
Requirement already satisfied: h11<1,>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from wsproto>=0.14->trio-websocket~=0.9->selenium) (0.12.
0)
Hit:1 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease
Ign:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  InRelease
Hit:3 http://security.ubuntu.com/ubuntu bionic-security InRelease
Ign:4 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  InRelease
Hit:5 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64  Release
Hit:6 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86_64  Release
Hit:7 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease
Hit:8 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:9 http://archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:10 http://ppa.launchpad.net/cran/libgit2/ubuntu bionic InRelease
Hit:11 http://archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:13 http://ppa.launchpad.net/deadsnakes/ppa/ubuntu bionic InRelease
Hit:15 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
chromium-chromedriver is already the newest version (95.0.4638.69-0ubuntu0.18.04.1).
0 upgraded, 0 newly installed, 0 to remove and 57 not upgraded.
cp: '/usr/lib/chromium-browser/chromedriver' and '/usr/bin/chromedriver' are the same file
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:24: DeprecationWarning: use options instead of chrome_options
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-fecd9ead3206> in <module>()
    108 totalImgs=5
    109
--> 110 saveInDestFolder(searchNames,destDir,totalImgs,driver)

<ipython-input-2-fecd9ead3206> in saveInDestFolder(searchNames, destDir, totalImgs, driver)
     91         path=os.path.join(destDir,name)
     92         if not os.path.isdir(path):
---> 93             os.mkdir(path)
     94         print('Current Path',path)
     95         totalLinks=getImageUrls(name,totalImgs,driver)

FileNotFoundError: [Errno 2] No such file or directory: '/content/drive/My Drive/Colab Notebooks/Dataset/cat'
```

Web Scraping of Movies Information using BeautifulSoup

We want to analyze the distributions of IMDB and Metacritic movie ratings to see if we find anything interesting. To do this, we'll first scrape data for over 2000 movies.

Most Voted Titles Released 2017-01-01 to 2017-12-31

1 to 50 of 117,062 titles | Next »

View Mode: Compact | **Detailed**

Sort by: Popularity | Alphabetical | IMDb Rating | **Number of Votes ▾** | US Box Office | Runtime | Year | Release Date

1. **Logan** (2017)

R | 137 min | Action, Drama, Sci-Fi

8.3 Rate this 77 Metascore

In the near future, a weary Logan cares for an ailing Professor X somewhere on the Mexican border. However, Logan's attempts to hide from the world and his legacy are upended when a young mutant arrives, pursued by dark forces.

Director: James Mangold | Stars: Hugh Jackman, Patrick Stewart, Dafne Keen, Boyd Holbrook

Votes: 309,245 | Gross: $226.23M

2. **Guardians of the Galaxy Vol. 2** (2017)

PG-13 | 136 min | Action, Adventure, Sci-Fi

8.1 Rate this 67 Metascore

The Guardians must fight to keep their newfound family together as they unravel the mystery of Peter Quill's true parentage.

Director: James Gunn | Stars: Chris Pratt, Zoe Saldana, Dave Bautista, Vin Diesel

Identifying the URL structure

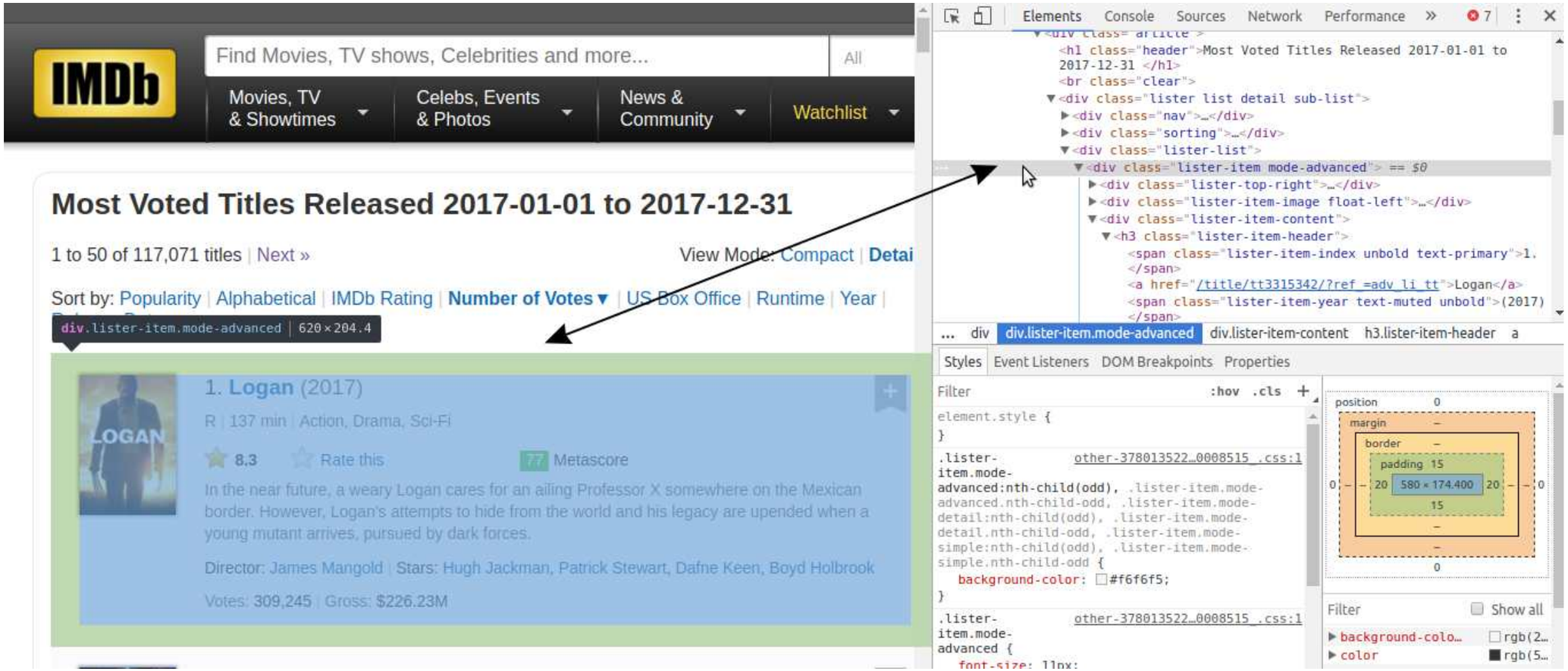
In the image above, you can see that the URL has several parameters after the question mark:

- release_date — Shows only the movies released in a specific year.
- sort — Sorts the movies on the page. sort=num_votes,desc translates to sort by number of votes in a descending order.
- page — Specifies the page number.
- ref_ — Takes us to the the next or the previous page. The reference is the page we are currently on. adv_nxt and adv_prv are two possible values. They translate to advance to the next page, and advance to the previous page, respectivel

In [ ]:
```
from requests import get
url = 'https://www.imdb.com/search/title?release_date=2017&sort=num_votes,desc&page=1'
response = get(url)
print(response.text[:500])
```

```
<!DOCTYPE html>
<html
    xmlns:og="http://ogp.me/ns#"
    xmlns:fb="http://www.facebook.com/2008/fbml">
    <head>

        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="apple-itunes-app" content="app-id=342792525, app-argument=imdb:///?src=mdot">



        <script type="text/javascript">var IMDbTimer={starttime: new Date().getTime(),pt:'java'};</script>

<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadTitle",
```

Understanding the HTML structure of a single page



Using BeautifulSoup to parse the HTML content

To parse our HTML document and extract the 50 div containers, we'll use a Python module called BeautifulSoup, the most common web scraping module for Python.

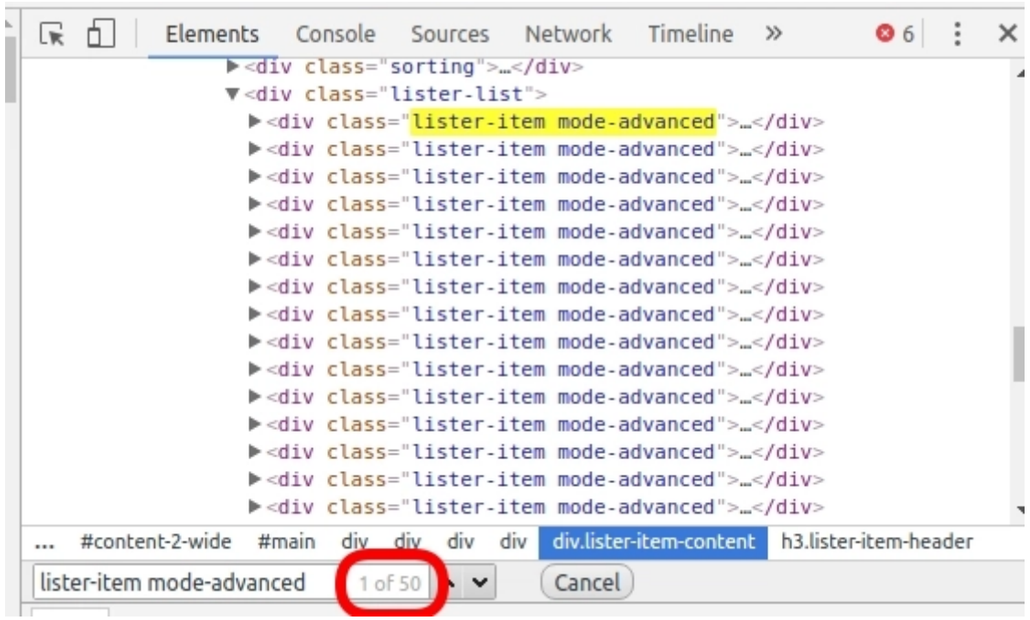
In the following code cell we will:

- Import the BeautifulSoup class creator from the package bs4.
- Parse response.text by creating a BeautifulSoup object, and assign this object to html_soup. The 'html.parser' argument indicates that we want to do the parsing using Python's built-in HTML parser.

In [ ]:
```
from bs4 import BeautifulSoup
html_soup = BeautifulSoup(response.text, 'html.parser')
headers = {'Accept-Language': 'en-US,en;q=0.8'}
type(html_soup)
```

Out[ ]:
```
bs4.BeautifulSoup
```

Before extracting the 50 div containers, we need to figure out what distinguishes them from other div elements on that page. Often, the distinctive mark resides in the class attribute. If you inspect the HTML lines of the containers of interest, you'll notice that the class attribute has two values: lister-item and mode-advanced. This combination is unique to these div containers. We can see that's true by doing a quick search (Ctrl + F). We have 50 such containers, so we expect to see only 50 matches:



Now let's use the find_all() method to extract all the div containers that have a class attribute of lister-item mode-advanced:

In [ ]:
```
movie_containers = html_soup.find_all('div', class_ = 'lister-item mode-advanced')
print(type(movie_containers))
print(len(movie_containers))
```

```
<class 'bs4.element.ResultSet'>
50
```

find_all() returned a ResultSet object which is a list containing all the 50 divs we are interested in.

Now we'll select only the first container, and extract, by turn, each item of interest:

- The name of the movie.
- The year of release.
- The IMDB rating.
- The Metascore.
- The number of votes.

Extracting the data for a single movie

We can access the first container, which contains information about a single movie, by using list notation on movie_containers.

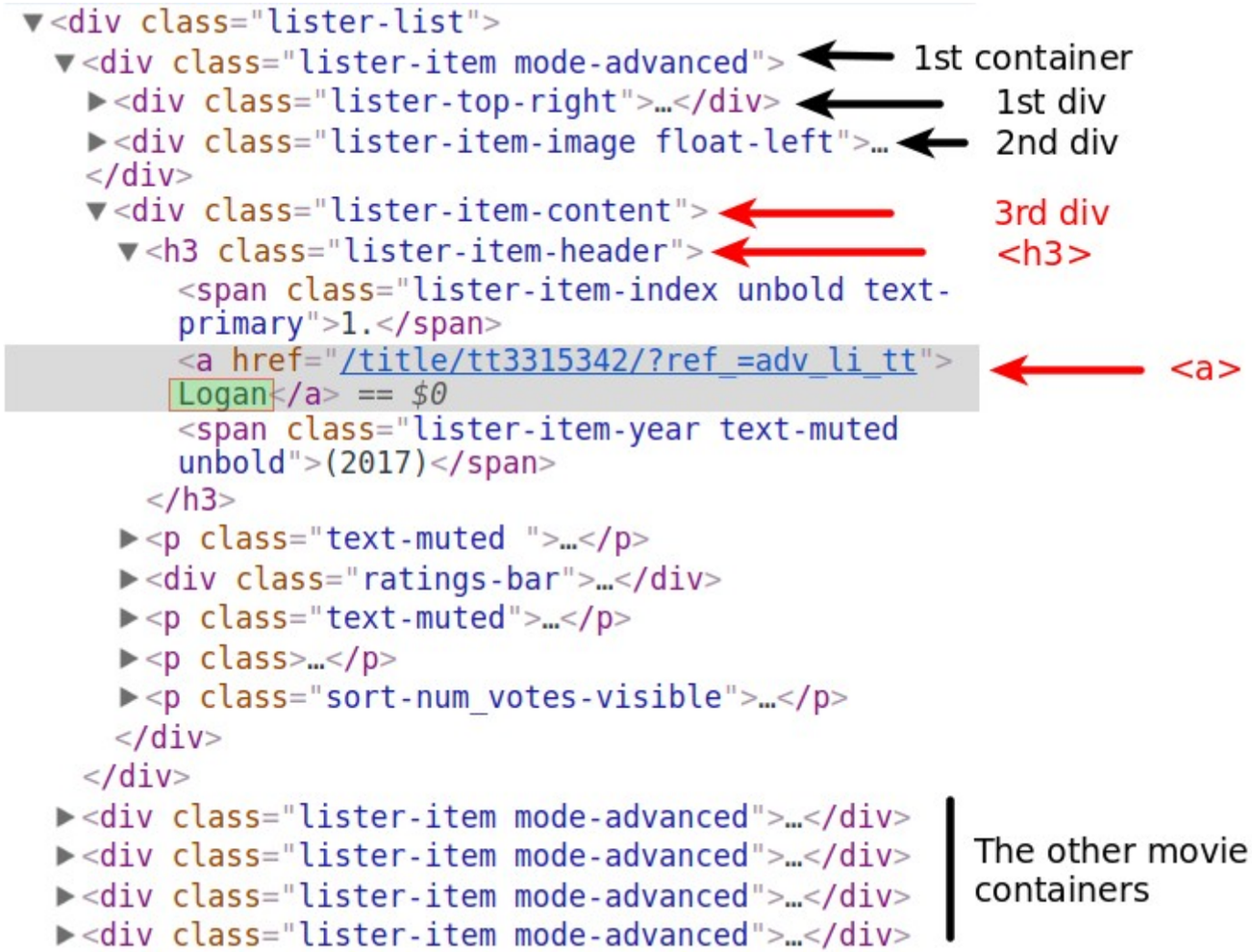
In [ ]:
```
first_movie = movie_containers[0]
first_movie
```

Out[ ]:

```
<div class="lister-item mode-advanced">
<div class="lister-top-right">
<div class="ribbonize" data-caller="filmosearch" data-tconst="tt3315342"></div>
</div>
<div class="lister-item-image float-left">
<a href="/title/tt3315342/"> <img alt="Logan" class="loadlate" data-tconst="tt3315342" height="98" loadlate="https://m.media-amazon.com/images/M/M
V5BYzc5MTU4N2EtYTkyMi00NjdhLTg3NWEtMTY4OTEyMzJhZTAzXkEyXkFqcGdeQXVyNjc1NTYyMjg@._V1_UX67_CR0,0,67,98_AL_.jpg" src="https://m.media-amazon.com/imag
es/S/sash/4FyxwxECzL-U1J8.png" width="67"/>
</a> </div>
<div class="lister-item-content">
<h3 class="lister-item-header">
<span class="lister-item-index unbold text-primary">1.</span>
<a href="/title/tt3315342/">Logan</a>
<span class="lister-item-year text-muted unbold">(2017)</span>
</h3>
<p class="text-muted ">
<span class="certificate">R</span>
<span class="ghost">|</span>
<span class="runtime">137 min</span>
<span class="ghost">|</span>
<span class="genre">
Action, Drama, Sci-Fi            </span>
</p>
<div class="ratings-bar">
<div class="inline-block ratings-imdb-rating" data-value="8.1" name="ir">
<span class="global-sprite rating-star imdb-rating"></span>
<strong>8.1</strong>
</div>
<div class="inline-block ratings-user-rating">
<span class="userRatingValue" data-tconst="tt3315342" id="urv_tt3315342">
<span class="global-sprite rating-star no-rating"></span>
<span class="rate" data-no-rating="Rate this" data-value="0" name="ur">Rate this</span>
</span>
<div class="starBarWidget" id="sb_tt3315342">
<div class="rating rating-list" data-csrf-token="" data-ga-identifier="" data-starbar-class="rating-list" data-user="" id="tt3315342|imdb|8.1|8.1|
adv_li_tt||advsearch|title" itemprop="aggregateRating" itemscope="" itemtype="http://schema.org/AggregateRating" title="Users rated this 8.1/10 (6
67,461 votes) - click stars to rate">
<meta content="8.1" itemprop="ratingValue"/>
<meta content="10" itemprop="bestRating"/>
<meta content="667461" itemprop="ratingCount"/>
<span class="rating-bg"> </span>
<span class="rating-imdb " style="width: 113.4px"> </span>
<span class="rating-stars">
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>1</span></a>
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>2</span></a>
```

```
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>3</span></a>
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>4</span></a>
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>5</span></a>
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>6</span></a>
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>7</span></a>
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>8</span></a>
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>9</span></a>
<a href="/register/login?why=vote" rel="nofollow" title="Register or login to rate this title"><span>10</span></a>
</span>
<span class="rating-rating "><span class="value">8.1</span><span class="grey">/</span><span class="grey">10</span></span>
<span class="rating-cancel "><a href="/title/tt3315342/vote" rel="nofollow" title="Delete"><span>X</span></a></span>
 </div>
</div>
</div>
<div class="inline-block ratings-metascore">
<span class="metascore favorable">77        </span>
        Metascore
            </div>
</div>
<p class="text-muted">
    In a future where mutants are nearly extinct, an elderly and weary Logan leads a quiet life. But when Laura, a mutant child pursued by scienti
sts, comes to him for help, he must get her to safety.</p>
<p class="">
    Director:
<a href="/name/nm0003506/">James Mangold</a>
<span class="ghost">|</span>
    Stars:
<a href="/name/nm0413168/">Hugh Jackman</a>,
<a href="/name/nm0001772/">Patrick Stewart</a>,
<a href="/name/nm6748436/">Dafne Keen</a>,
<a href="/name/nm2933542/">Boyd Holbrook</a>
</p>
<p class="sort-num_votes-visible">
<span class="text-muted">Votes:</span>
<span data-value="667461" name="nv">667,461</span>
<span class="ghost">|</span> <span class="text-muted">Gross:</span>
<span data-value="226,277,068" name="nv">$226.28M</span>
</p>
</div>
</div>
```

The name of the movie



In [ ]:
```
first_movie.div
```

Out[ ]:
```
<div class="lister-top-right">
<div class="ribbonize" data-caller="filmosearch" data-tconst="tt3315342"></div>
</div>
```

In [ ]:
```
first_movie.a
```

Out[ ]:
```
<a href="/title/tt3315342/"> <img alt="Logan" class="loadlate" data-tconst="tt3315342" height="98" loadlate="https://m.media-amazon.com/images/M/MV5BYzc5MTU4N2EtYTkyMi00NjdhLTg3NWEtMTY4OTEyMzJhZTAzXkEyXkFqcGdeQXVyNjc1NTYyMjg@._V1_UX67_CR0,0,67,98_AL_.jpg" src="https://m.media-amazon.com/images/S/sash/4FyxwxECzL-U1J8.png" width="67"/>
</a>
```

In [ ]:
```
first_movie.h3
```

Out[ ]:
```
<h3 class="lister-item-header">
<span class="lister-item-index unbold text-primary">1.</span>
<a href="/title/tt3315342/">Logan</a>
<span class="lister-item-year text-muted unbold">(2017)</span>
</h3>
```

In [ ]:
```
first_movie.h3.a
```

Out[ ]:
```
<a href="/title/tt3315342/">Logan</a>
```

In [ ]:
```
first_name = first_movie.h3.a.text
first_name
```

Out[ ]:
```
'Logan'
```

The year of the movie's release

In [ ]:
```
first_year = first_movie.h3.find('span', class_ = 'lister-item-year text-muted unbold')
first_year
```

Out[ ]:
```
<span class="lister-item-year text-muted unbold">(2017)</span>
```

In [ ]:
```
first_year = first_year.text
first_year
```

Out[ ]:
```
'(2017)'
```

The IMDB rating

In [ ]:
```
first_movie.strong
```

Out[ ]:
```
<strong>8.1</strong>
```

In [ ]:
```
first_imdb = float(first_movie.strong.text)
first_imdb
```

Out[ ]:
```
8.1
```

The Metascore

In [ ]:
```
first_mscore = first_movie.find('span', class_ = 'metascore favorable')
first_mscore = int(first_mscore.text)
print(first_mscore)
```

```
77
```

The number of votes

In [ ]:
```
first_votes = first_movie.find('span', attrs = {'name':'nv'})
first_votes
```

Out[ ]:
```
<span data-value="667461" name="nv">667,461</span>
```

In [ ]:
```
first_votes['data-value']
```

Out[ ]:
```
'667461'
```

In [ ]:
```
first_votes = int(first_votes['data-value'])
```

The script

In [ ]:
```
# Lists to store the scraped data in
names = []
years = []
imdb_ratings = []
metascores = []
votes = []
# Extract data from individual movie container

for container in movie_containers:

# If the movie has Metascore, then extract:
    if container.find('div', class_ = 'ratings-metascore') is not None:
# The name
      name = container.h3.a.text
      names.append(name)
# The year
      year = container.h3.find('span', class_ = 'lister-item-year').text
```

```
        years.append(year)
# The IMDB rating
        imdb = float(container.strong.text)
        imdb_ratings.append(imdb)
# The Metascore
        m_score = container.find('span', class_ = 'metascore').text
        metascores.append(int(m_score))
# The number of votes
        vote = container.find('span', attrs = {'name':'nv'})['data-value']
        votes.append(int(vote))
```

In [ ]:

```
import pandas as pd
test_df = pd.DataFrame({'movie': names,
'year': years,
'imdb': imdb_ratings,
'metascore': metascores,
'votes': votes
})
print(test_df.info())
test_df
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   movie      42 non-null     object 
 1   year       42 non-null     object 
 2   imdb       42 non-null     float64
 3   metascore  42 non-null     int64  
 4   votes      42 non-null     int64  
dtypes: float64(1), int64(2), object(2)
memory usage: 1.8+ KB
None
```

Out[ ]:

	movie	year	imdb	metascore	votes
0	Logan	(2017)	8.1	77	667461
1	Thor: Ragnarok	(2017)	7.9	74	615570
2	Wonder Woman	(2017)	7.4	76	594410
3	Guardians of the Galaxy Vol. 2	(2017)	7.6	67	589160
4	Star Wars: Episode VIII - The Last Jedi	(2017)	7.0	84	577006
5	Dunkirk	(2017)	7.8	94	575108
6	Spider-Man: Homecoming	(2017)	7.4	73	532718
7	Get Out	(I) (2017)	7.7	85	509953
8	Blade Runner 2049	(2017)	8.0	81	479952
9	It	(I) (2017)	7.3	69	471568
10	Baby Driver	(2017)	7.6	86	457272
11	Three Billboards Outside Ebbing, Missouri	(2017)	8.1	88	451012
12	Justice League	(2017)	6.1	45	415247
13	Coco	(I) (2017)	8.4	81	407616
14	The Shape of Water	(2017)	7.3	87	382846
15	John Wick: Chapter 2	(2017)	7.5	75	379693
16	Jumanji: Welcome to the Jungle	(2017)	6.9	58	330889
17	Kong: Skull Island	(2017)	6.6	62	289484
18	Kingsman: The Golden Circle	(2017)	6.7	44	282140
19	Beauty and the Beast	(I) (2017)	7.1	65	280830
20	Pirates of the Caribbean: Dead Men Tell No Tales	(2017)	6.5	39	271915
21	Alien: Covenant	(2017)	6.4	65	259558
22	Lady Bird	(2017)	7.4	94	251279
23	The Greatest Showman	(2017)	7.6	48	247712
24	War for the Planet of the Apes	(2017)	7.4	82	233250
25	Call Me by Your Name	(2017)	7.9	93	224331

	movie	year	imdb	metascore	votes
26	Murder on the Orient Express	(2017)	6.5	52	219034
27	Wind River	(2017)	7.7	73	214702
28	Life	(I) (2017)	6.6	54	211675
29	The Fate of the Furious	(2017)	6.6	56	210485
30	Ghost in the Shell	(2017)	6.3	52	200056
31	King Arthur: Legend of the Sword	(2017)	6.7	41	197543
32	Mother!	(2017)	6.6	75	196640
33	The Hitman's Bodyguard	(2017)	6.9	47	194477
34	I, Tonya	(2017)	7.5	77	189693
35	Atomic Blonde	(2017)	6.7	63	180339
36	Darkest Hour	(2017)	7.4	75	177390
37	The Mummy	(2017)	5.4	34	176086
38	Bright	(I) (2017)	6.3	29	172818
39	Baywatch	(2017)	5.5	37	166656
40	Valerian and the City of a Thousand Planets	(2017)	6.5	51	166515
41	American Made	(2017)	7.2	65	163409

The script for multiple pages

In [ ]:
```
from time import time
from time import sleep
from random import randint

from IPython.core.display import clear_output
pages = [ '1','2','3','4','5']
years_url = [ '2017', '2018', '2019', '2020']

# Redeclaring the lists to store data in
names = []
years = []
imdb_ratings = []
metascores = []
```

```
votes = []

# Preparing the monitoring of the loop
start_time = time()
requests = 0

# For every year in the interval 2000-2017
for year_url in years_url:

    # For every page in the interval 1-4
    for page in pages:

        # Make a get request
        response = get('https://www.imdb.com/search/title?release_date=' + year_url +
        '&sort=num_votes,desc&page=' + page, headers = headers)

        # Pause the loop
        sleep(randint(8,15))

        # Monitor the requests
        requests += 1
        elapsed_time = time() - start_time
        print('Request:{}; Frequency: {} requests/s'.format(requests, requests/elapsed_time))
        clear_output(wait = True)

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

        # Break the loop if the number of requests is greater than expected
        if requests > 72:
            warn('Number of requests was greater than expected.')
            break

        # Parse the content of the request with BeautifulSoup
        page_html = BeautifulSoup(response.text, 'html.parser')

        # Select all the 50 movie containers from a single page
        mv_containers = page_html.find_all('div', class_ = 'lister-item mode-advanced')

        # For every movie of these 50
        for container in mv_containers:
            # If the movie has a Metascore, then:
            if container.find('div', class_ = 'ratings-metascore') is not None:
```

```
                # Scrape the name
                name = container.h3.a.text
                names.append(name)

                # Scrape the year
                year = container.h3.find('span', class_ = 'lister-item-year').text
                years.append(year)

                # Scrape the IMDB rating
                imdb = float(container.strong.text)
                imdb_ratings.append(imdb)

                # Scrape the Metascore
                m_score = container.find('span', class_ = 'metascore').text
                metascores.append(int(m_score))

                # Scrape the number of votes
                vote = container.find('span', attrs = {'name':'nv'})['data-value']
                votes.append(int(vote))
```

```
Request:20; Frequency: 0.08430509283518908 requests/s
```

In [ ]:

```
movie_ratings = pd.DataFrame({'movie': names,
'year': years,
'imdb': imdb_ratings,
'metascore': metascores,
'votes': votes
})
print(movie_ratings.info())
movie_ratings.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 830 entries, 0 to 829
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   movie       830 non-null    object 
 1   year        830 non-null    object 
 2   imdb        830 non-null    float64
 3   metascore   830 non-null    int64  
 4   votes       830 non-null    int64  
dtypes: float64(1), int64(2), object(2)
memory usage: 32.5+ KB
None
```

Out[ ]:

	movie	year	imdb	metascore	votes
0	Logan	(2017)	8.1	77	667461
1	Thor: Ragnarok	(2017)	7.9	74	615570
2	Wonder Woman	(2017)	7.4	76	594410
3	Guardians of the Galaxy Vol. 2	(2017)	7.6	67	589160
4	Star Wars: Episode VIII - The Last Jedi	(2017)	7.0	84	577006
5	Dunkirk	(2017)	7.8	94	575108
6	Spider-Man: Homecoming	(2017)	7.4	73	532718
7	Get Out	(I) (2017)	7.7	85	509953
8	Blade Runner 2049	(2017)	8.0	81	479952
9	It	(I) (2017)	7.3	69	471568

In [ ]:
```
movie_ratings.tail(10)
```

Out[ ]:

	movie	year	imdb	metascore	votes
820	Spenser Confidential	(2020)	6.2	49	75101
821	The Midnight Sky	(2020)	5.6	58	74202
822	The Social Dilemma	(2020)	7.6	78	70702
823	I'm Thinking of Ending Things	(2020)	6.6	78	69026
824	Underwater	(2020)	5.8	48	67988
825	Hamilton	(2020)	8.5	90	65067
826	Bloodshot	(2020)	5.7	44	64790
827	News of the World	(2020)	6.8	73	64451
828	The Father	(I) (2020)	8.3	88	63849
829	Mank	(2020)	6.9	79	61200

In [ ]:
```
movie_ratings.to_csv('/content/drive/My Drive/Colab Notebooks/Dataset/movie_ratings.csv')
```

Data Preparation

- Collected data may not be compatible or formatted correctly
- Data must be prepared before it can be added to a data set
- Extract, Transform and Load (ETL)

> process for collecting data from a variety of sources, transforming the data, and then loading the data into a database

Data preprocessing

Data Processing is a process of cleaning the raw data i.e. the data is collected in the real world and is converted to a clean data set. In other words, whenever the data is gathered from different sources it is collected in a raw format and this data isn't feasible for the analysis. Therefore, certain steps are executed to convert the data into a small clean data set, this part of the process is called as data preprocessing.

Most of the real-world data is messy, some of these types of data are: 1. **Missing data**: Missing data can be found when it is not continuously created or due to technical issues in the application (IOT system). 2. **Noisy Data** This type of data is also called outliners, this can occur due to human errors (human manually gathering the data) or some technical problem of the device at the time of collection of data. 3. **Inconsistent data:** This type of data might be collected due to human errors (mistakes with the name or values) or duplication of data.

These are some of the basic pre processing techniques that can be used to convert raw data. 1. **Conversion of data:** As we know that Machine Learning models can only handle numeric features, hence categorical and ordinal data must be somehow converted into numeric features. 2. **Ignoring the missing values:** Whenever we encounter missing data in the data set then we can remove the row or column of data depending on our need. This method is known to be efficient but it shouldn't be performed if there are a lot of missing values in the dataset. 3. **Filling the missing values:** Whenever we encounter missing data in the data set then we can fill the missing data manually, most commonly the mean, median or highest frequency value is used.

1. **Machine learning:** If we have some missing data then we can predict what data shall be present at the empty position by using the existing data. 5. **Outliers detection:** There are some error data that might be present in our data set that deviates drastically from other observations in a data set. [Example: human weight = 800 Kg; due to mistyping of extra 0]

Example of Data Preparation of movie_rating.csv

In [ ]:
```
movie_ratings['year'].unique()
```

Out[ ]:
```
array(['(2017)', '(I) (2017)', '(2018)', '(I) (2018)', '(III) (2018)',
       '(2019)', '(II) (2019)', '(I) (2019)', '(2020)', '(I) (2020)',
       '(II) (2020)'], dtype=object)
```

In [ ]:
```
movie_ratings.dtypes
```

Out[ ]:
```
movie         object
year          object
imdb         float64
metascore      int64
votes          int64
dtype: object
```

In [ ]:
```
movie_ratings['year'] = (movie_ratings.year.apply(lambda x:x.replace('(I)','')))
```

In [ ]:
```
movie_ratings['year'].unique()
```

Out[ ]:
```
array(['(2017)', ' (2017)', '(2018)', ' (2018)', '(III) (2018)', '(2019)',
       '(II) (2019)', ' (2019)', '(2020)', ' (2020)', '(II) (2020)'],
      dtype=object)
```

In [ ]:
```
movie_ratings['year'] = (movie_ratings.year.apply(lambda x:x.replace('(II)','')))
```

In [ ]:
```
movie_ratings['year'] = (movie_ratings.year.apply(lambda x:x.replace('(III)','')))
```

In [ ]:
```
movie_ratings['year'].unique()
```

Out[ ]:
```
array(['(2017)', ' (2017)', '(2018)', ' (2018)', '(2019)', ' (2019)',
       '(2020)', ' (2020)'], dtype=object)
```

In [ ]:
```
movie_ratings['year'] = (movie_ratings.year.apply(lambda x:x.replace('(','')))
```

In [ ]:
```
movie_ratings['year'].unique()
```

Out[ ]:
```
array(['2017)', ' 2017)', '2018)', ' 2018)', '2019)', ' 2019)', '2020)',
       ' 2020)'], dtype=object)
```

In [ ]:
```
movie_ratings['year'] = (movie_ratings.year.apply(lambda x:x.replace(')','')))
```

In [ ]:
```
movie_ratings['year'].unique()
```

Out[ ]:
```
array(['2017', ' 2017', '2018', ' 2018', '2019', ' 2019', '2020', ' 2020'],
      dtype=object)
```

In [ ]:
```
movie_ratings['year'] = movie_ratings['year'].astype(int)
```

In [ ]:
```
movie_ratings['year'].unique()
```

Out[ ]:
```
array([2017, 2018, 2019, 2020])
```

In [ ]:
```
movie_ratings.dtypes
```

Out[ ]:
```
movie         object
year           int64
imdb         float64
metascore      int64
votes          int64
dtype: object
```

In [ ]:
```
movie_ratings.head(10)
```

Out[ ]:

	movie	year	imdb	metascore	votes
0	Logan	2017	8.1	77	667461
1	Thor: Ragnarok	2017	7.9	74	615570
2	Wonder Woman	2017	7.4	76	594410
3	Guardians of the Galaxy Vol. 2	2017	7.6	67	589160
4	Star Wars: Episode VIII - The Last Jedi	2017	7.0	84	577006
5	Dunkirk	2017	7.8	94	575108
6	Spider-Man: Homecoming	2017	7.4	73	532718
7	Get Out	2017	7.7	85	509953
8	Blade Runner 2049	2017	8.0	81	479952
9	It	2017	7.3	69	471568

In [ ]:
```
movie_ratings.tail(10)
```

Out[ ]:

	movie	year	imdb	metascore	votes
820	Spenser Confidential	2020	6.2	49	75101
821	The Midnight Sky	2020	5.6	58	74202
822	The Social Dilemma	2020	7.6	78	70702
823	I'm Thinking of Ending Things	2020	6.6	78	69026
824	Underwater	2020	5.8	48	67988
825	Hamilton	2020	8.5	90	65067
826	Bloodshot	2020	5.7	44	64790
827	News of the World	2020	6.8	73	64451
828	The Father	2020	8.3	88	63849
829	Mank	2020	6.9	79	61200

In [ ]:

```
movie_ratings
```

Out[ ]:

	movie	year	imdb	metascore	votes
0	Logan	2017	8.1	77	667461
1	Thor: Ragnarok	2017	7.9	74	615570
2	Wonder Woman	2017	7.4	76	594410
3	Guardians of the Galaxy Vol. 2	2017	7.6	67	589160
4	Star Wars: Episode VIII - The Last Jedi	2017	7.0	84	577006
...	...	...	...	...	...
825	Hamilton	2020	8.5	90	65067
826	Bloodshot	2020	5.7	44	64790
827	News of the World	2020	6.8	73	64451
828	The Father	2020	8.3	88	63849
829	Mank	2020	6.9	79	61200

830 rows × 5 columns