UNIVERSITY OF MALAYA

EXAMINATION FOR THE DEGREE OF MASTER OF DATA SCIENCE

ACADEMIC SESSION 2019/2020 : SEMESTER II

WQD7005 : Data Mining

June 2020

INSTRUCTIONS TO CANDIDATES :

Answer **ALL** questions (50 marks).

Name : Nurullainy Binti Mat Rashid

ID : 17036591

Link to codes and data:
https://github.com/Nurullainy/Data-Mining-Final-Exam

Link to video :
https://drive.google.com/drive/folders/1hMqLYw5ubSpMRsPvyJwjDG8MuV3GO_Qt?usp=sharing

(This question paper consists of 5 questions on 3 printed pages)

Mini-assignment (50 marks)

Instructions: Work individually, submission via Spectrum.

1. You are required to make a user-agent that will crawl the WWW (your familiar domain) to produce dataset of a particular website.
   - the web site can be as simple as a list of webpages and what other pages they link to
   - the output does not need to be in XHTML (or HTML) form

   a multi-stage approach (e.g. produce the xhtml or html in csv format)

   Topic: Web Crawling Data of TV Shows and Movies at Internet Movie Database (IMDb)

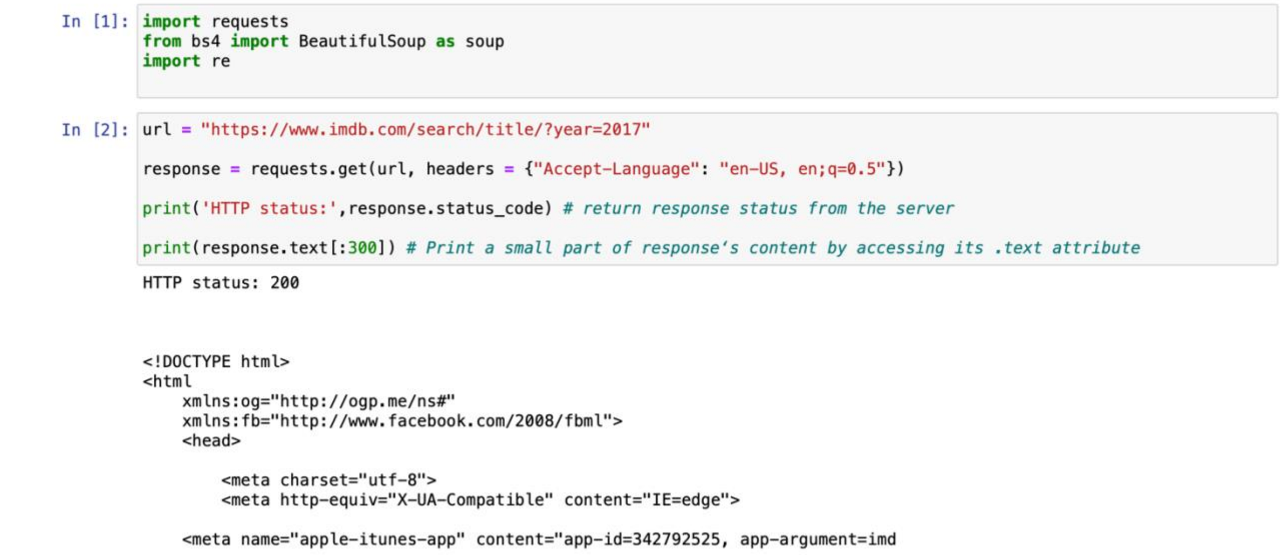
   IMDb is an online database of information related to films, television programs, home videos, video games, and streaming content online – including cast, year released, ratings, production crew, plot summaries, trivia, fan and critical reviews.

   I want to analyze data of TV Show and movie from IMDb. The data can be extracted from this website : https://www.imdb.com/search/title/?year=2017

   1.1 Importing Python Libraries

   First of all, I will do the following step:

   1) Import requests module and BeautifulSoup from bs4
   2) Assign the address of the web page to a variable named ‘url’.
   3) Request the server the content of the web page by using get( ), and store the server’s response in the variable ‘response’.
   4) Print HTTP status code and a small part of response‘s content by accessing its text attribute (response is now a Response object).

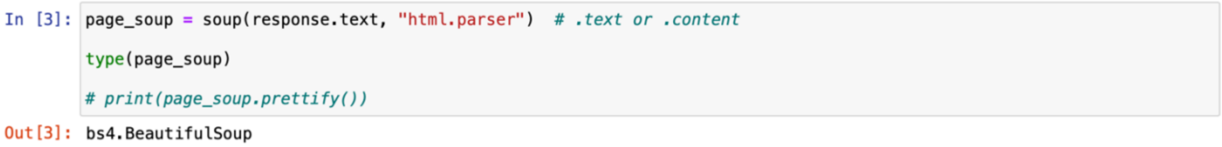


```
In [1]: import requests
        from bs4 import BeautifulSoup as soup
        import re

In [2]: url = "https://www.imdb.com/search/title/?year=2017"

        response = requests.get(url, headers = {"Accept-Language": "en-US, en;q=0.5"})

        print('HTTP status:',response.status_code) # return response status from the server

        print(response.text[:300]) # Print a small part of response‘s content by accessing its .text attribute

        HTTP status: 200


        <!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=imd
```

1.2 Using Beautifulsoup Module To Parse The HTML Content

Parsing HTML document and extract the 50 div containers

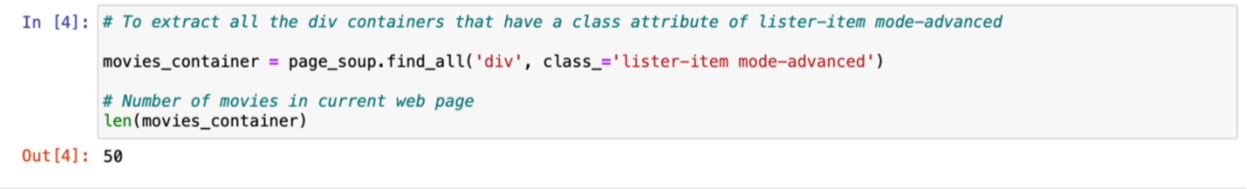
1) Import the BeautifulSoup class creator from the package bs4.
2) Parse response.text by creating a BeautifulSoup object, and assign this object to page_soup.
3) The 'html.parser' argument indicates that we want to do the parsing using Python’s built-in HTML parser.



```
In [3]: page_soup = soup(response.text, "html.parser")  # .text or .content

        type(page_soup)

        # print(page_soup.prettify())

Out[3]: bs4.BeautifulSoup
```

Class attribute has two values; 1) lister-item and 2) mode-advanced.

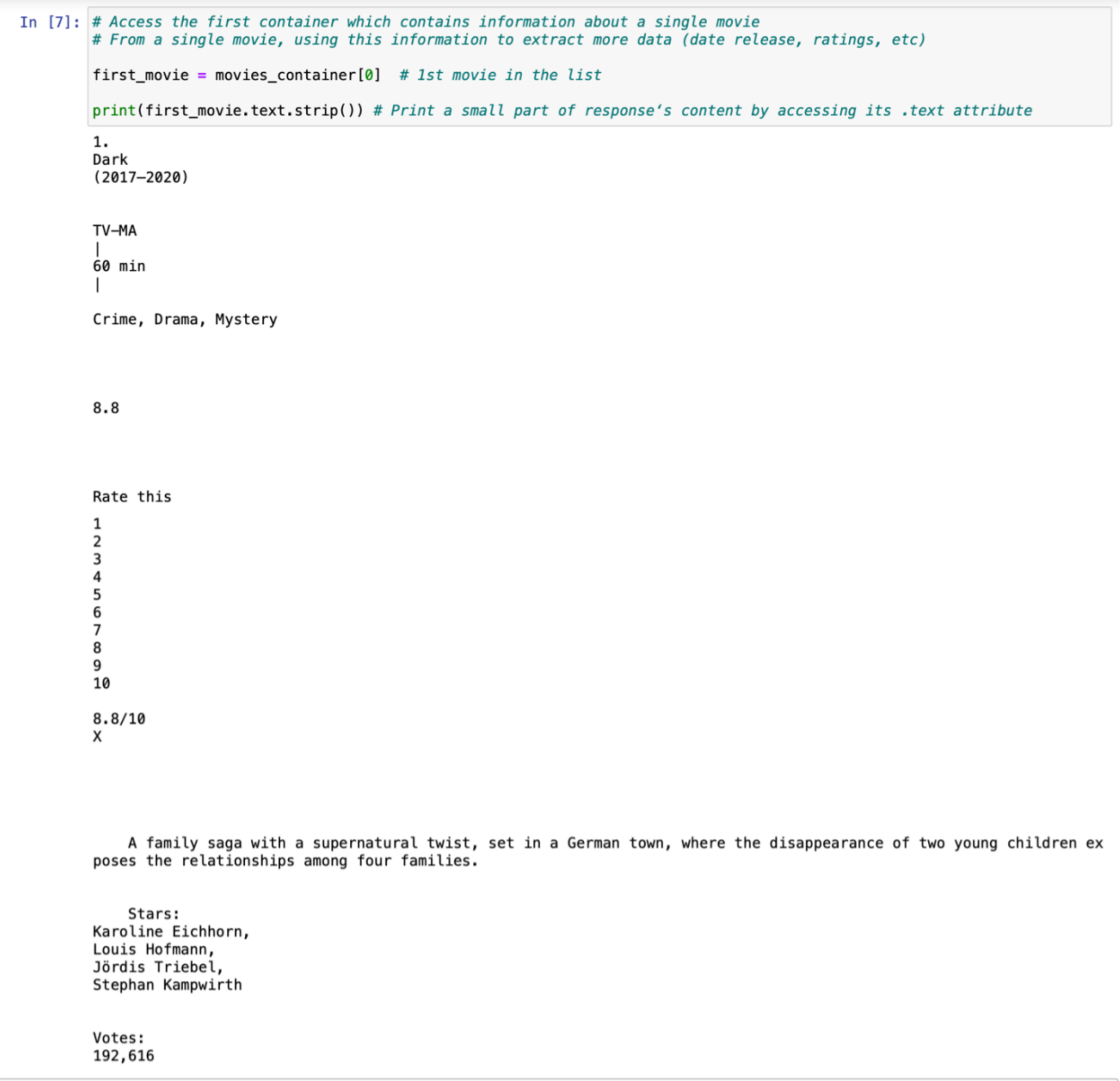
This combination is unique to these div containers. Use the find_all() method to extract all the div containers that have a class attribute of *lister-item mode-advanced* and assign it to variable *movie_container*. It will return a ResultSet object which is a list containing all the 50 divs



```
In [4]: # To extract all the div containers that have a class attribute of lister-item mode-advanced

        movies_container = page_soup.find_all('div', class_='lister-item mode-advanced')

        # Number of movies in current web page
        len(movies_container)

Out[4]: 50
```

1.3 Extracting The Data For A Single Movie

Now I'm selecting one movie container (let say the first container) to extract 9 attributes that I am interested with for next data mining purposes:



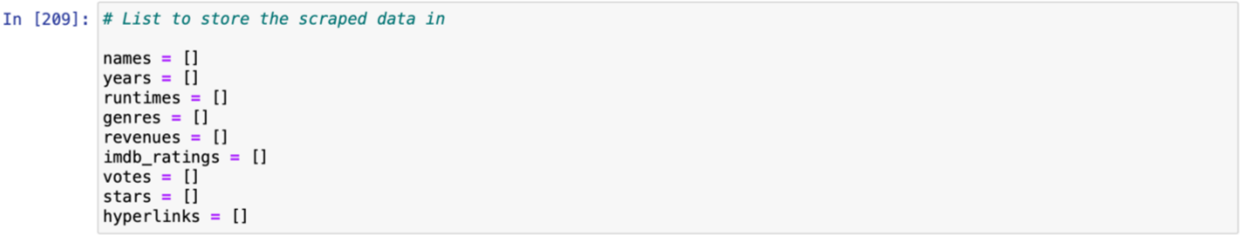
```
In [7]: # Access the first container which contains information about a single movie
        # From a single movie, using this information to extract more data (date release, ratings, etc)

        first_movie = movies_container[0]  # 1st movie in the list

        print(first_movie.text.strip()) # Print a small part of response's content by accessing its .text attribute

        1.
        Dark
        (2017–2020)


        TV-MA
        |
        60 min
        |

        Crime, Drama, Mystery




        8.8




        Rate this

        1
        2
        3
        4
        5
        6
        7
        8
        9
        10

        8.8/10
        X





            A family saga with a supernatural twist, set in a German town, where the disappearance of two young children ex
        poses the relationships among four families.


            Stars:
        Karoline Eichhorn,
        Louis Hofmann,
        Jördis Triebel,
        Stephan Kampwirth


        Votes:
        192,616
```

The 9 following attributes are as follows:

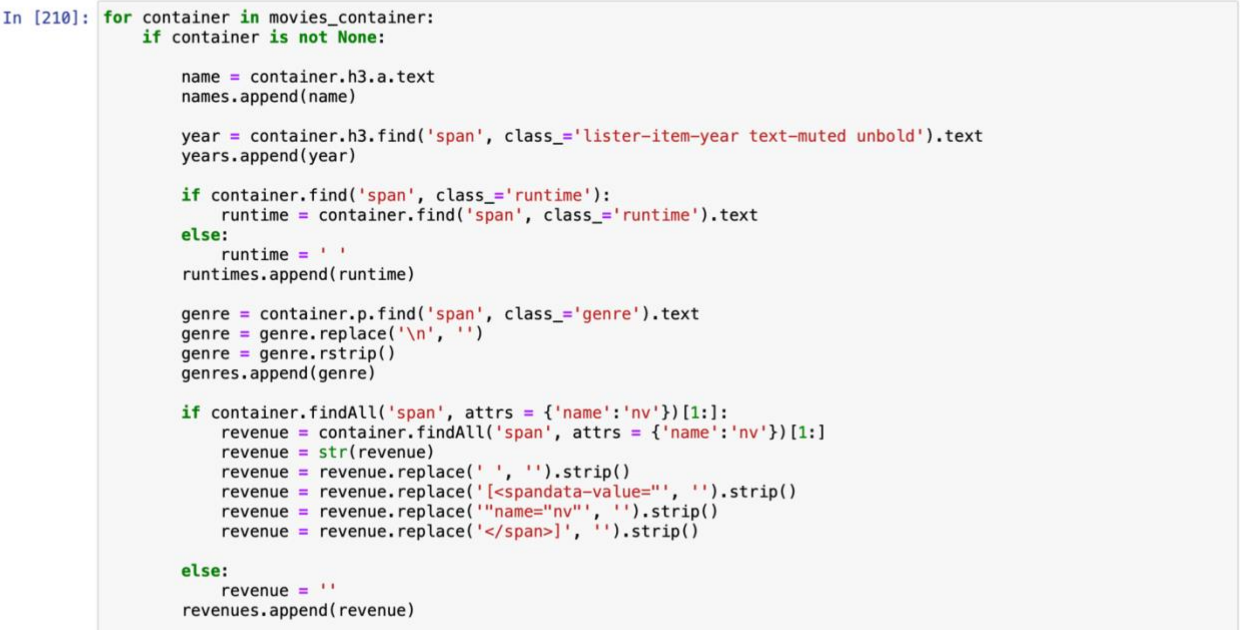
1) The name of the TV show or movie
2) The year of release
3) Runtime of each TV show or movie
4) Genre of TV show or movie
5) Revenues from the movie released
6) The IMDB rating
7) The number of votes from user
8) Stars of the TV show or movie (name of director and main cast)
9) Hyperlink to the TV show or movie

1.4 Extracting Information For All The Tv Shows And Movies In A Single Page

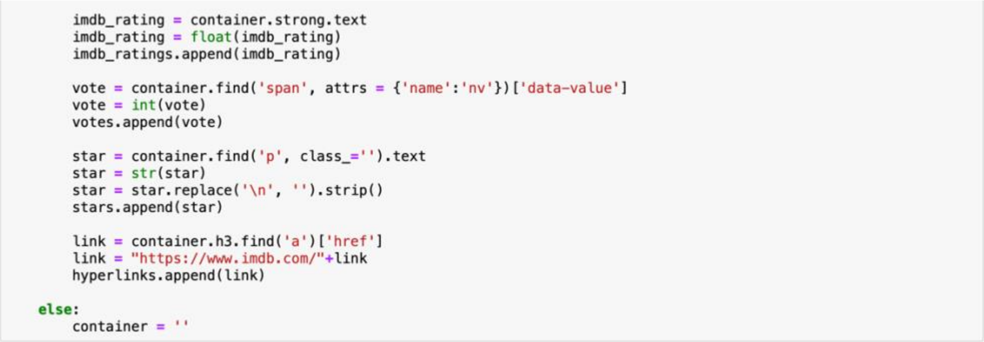
1) Declare list of variables to have something to store the extracted data in.
2) Loop through each container in a web page (the variable which contains all the 50 movie containers).
3) Extract the data points of interest only if the container is True



```
In [209]: # List to store the scraped data in

          names = []
          years = []
          runtimes = []
          genres = []
          revenues = []
          imdb_ratings = []
          votes = []
          stars = []
          hyperlinks = []
```

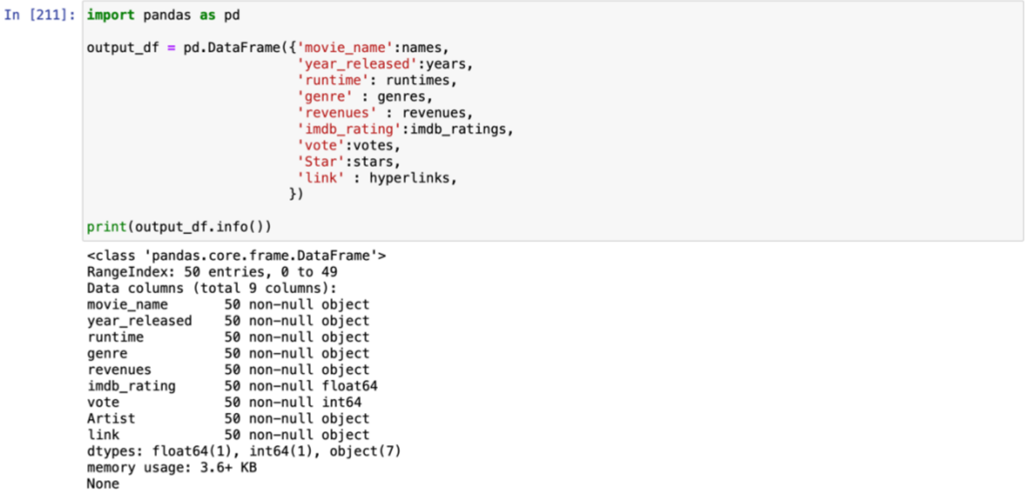


```
In [210]: for container in movies_container:
              if container is not None:

                  name = container.h3.a.text
                  names.append(name)

                  year = container.h3.find('span', class_='lister-item-year text-muted unbold').text
                  years.append(year)

                  if container.find('span', class_='runtime'):
                      runtime = container.find('span', class_='runtime').text
                  else:
                      runtime = ' '
                  runtimes.append(runtime)

                  genre = container.p.find('span', class_='genre').text
                  genre = genre.replace('\n', '')
                  genre = genre.rstrip()
                  genres.append(genre)

                  if container.findAll('span', attrs = {'name':'nv'})[1:]:
                      revenue = container.findAll('span', attrs = {'name':'nv'})[1:]
                      revenue = str(revenue)
                      revenue = revenue.replace(' ', '').strip()
                      revenue = revenue.replace('[<spandata-value="', '').strip()
                      revenue = revenue.replace('"name="nv"', '').strip()
                      revenue = revenue.replace('</span>]', '').strip()

                  else:
                      revenue = ''
                  revenues.append(revenue)
```



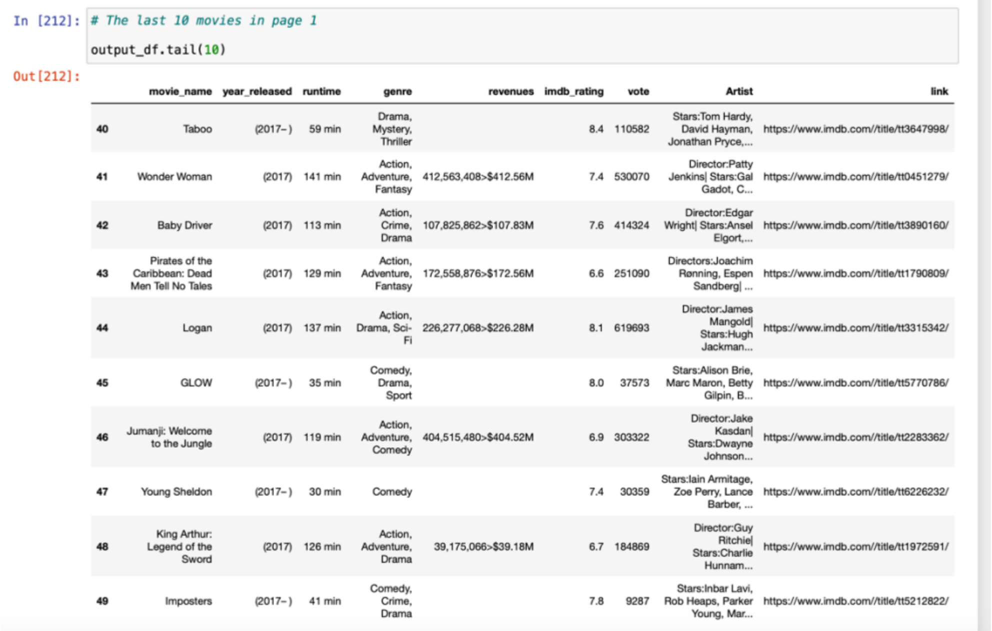
```
        imdb_rating = container.strong.text
        imdb_rating = float(imdb_rating)
        imdb_ratings.append(imdb_rating)

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

        star = container.find('p', class_='').text
        star = str(star)
        star = star.replace('\n', '').strip()
        stars.append(star)

        link = container.h3.find('a')['href']
        link = "https://www.imdb.com/"+link
        hyperlinks.append(link)

    else:
        container = ''
```

Print last 10 movie_container in first web page



```
In [211]: import pandas as pd

          output_df = pd.DataFrame({'movie_name':names,
                                    'year_released':years,
                                    'runtime': runtimes,
                                    'genre' : genres,
                                    'revenues' : revenues,
                                    'imdb_rating':imdb_ratings,
                                    'vote':votes,
                                    'Star':stars,
                                    'link' : hyperlinks,
                                   })

          print(output_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 9 columns):
movie_name       50 non-null object
year_released    50 non-null object
runtime          50 non-null object
genre            50 non-null object
revenues         50 non-null object
imdb_rating      50 non-null float64
vote             50 non-null int64
Artist           50 non-null object
link             50 non-null object
dtypes: float64(1), int64(1), object(7)
memory usage: 3.6+ KB
None
```



```
In [212]: # The last 10 movies in page 1

          output_df.tail(10)
```

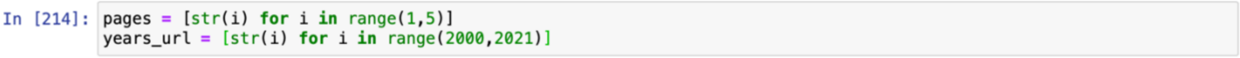
Out[212]:

	movie_name	year_released	runtime	genre	revenues	imdb_rating	vote	Artist	link
40	Taboo	(2017–)	59 min	Drama, Mystery, Thriller		8.4	110582	Stars:Tom Hardy, David Hayman, Jonathan Pryce,...	https://www.imdb.com//title/tt3647998/
41	Wonder Woman	(2017)	141 min	Action, Adventure, Fantasy	412,563,408>$412.56M	7.4	530070	Director:Patty Jenkins| Stars:Gal Gadot, C...	https://www.imdb.com//title/tt0451279/
42	Baby Driver	(2017)	113 min	Action, Crime, Drama	107,825,862>$107.83M	7.6	414324	Director:Edgar Wright| Stars:Ansel Elgort,...	https://www.imdb.com//title/tt3890160/
43	Pirates of the Caribbean: Dead Men Tell No Tales	(2017)	129 min	Action, Adventure, Fantasy	172,558,876>$172.56M	6.6	251090	Directors:Joachim Rønning, Espen Sandberg| ...	https://www.imdb.com//title/tt1790809/
44	Logan	(2017)	137 min	Action, Drama, Sci-Fi	226,277,068>$226.28M	8.1	619693	Director:James Mangold| Stars:Hugh Jackman...	https://www.imdb.com//title/tt3315342/
45	GLOW	(2017–)	35 min	Comedy, Drama, Sport		8.0	37573	Stars:Alison Brie, Marc Maron, Betty Gilpin, B...	https://www.imdb.com//title/tt5770786/
46	Jumanji: Welcome to the Jungle	(2017)	119 min	Action, Adventure, Comedy	404,515,480>$404.52M	6.9	303322	Director:Jake Kasdan| Stars:Dwayne Johnson...	https://www.imdb.com//title/tt2283362/
47	Young Sheldon	(2017–)	30 min	Comedy		7.4	30359	Stars:Iain Armitage, Zoe Perry, Lance Barber, ...	https://www.imdb.com//title/tt6226232/
48	King Arthur: Legend of the Sword	(2017)	126 min	Action, Adventure, Drama	39,175,066>$39.18M	6.7	184869	Director:Guy Ritchie| Stars:Charlie Hunnam...	https://www.imdb.com//title/tt1972591/
49	Imposters	(2017–)	41 min	Comedy, Crime, Drama		7.8	9287	Stars:Inbar Lavi, Rob Heaps, Parker Young, Mar...	https://www.imdb.com//title/tt5212822/

1.5 Extracting data for all TV shows and movies from multiple pages from year 2000 – 2020

/3

1) Create a list called *pages*, and populate it with the strings corresponding to the first 4 pages.
2) Create a list called *years_url* and populate it with the strings corresponding to the years 2000 - 2020



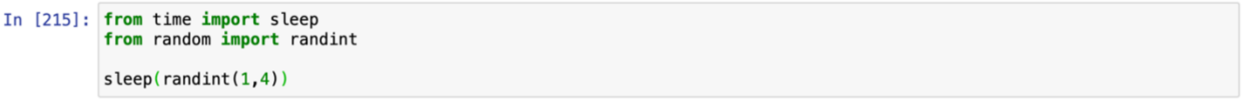
```
In [214]: pages = [str(i) for i in range(1,5)]
          years_url = [str(i) for i in range(2000,2021)]
```

1.6 Controlling the crawl-rate

Controlling the rate of crawling is important for the website that I will be scraping. If I not controlling the rate of crawling, much less likely to get my IP address banned.

Need to avoid activity disruption of the scraped website by allowing the server to respond to other users' requests too.

Control the loop's rate by using the *sleep()* function from Python's time module. *sleep()* will pause the execution of the loop for a specified amount of seconds. To mimic human behavior, I'll vary the amount of waiting time between requests by using the *randint()* function from the Python's random module. *randint()* randomly generates integers within a specified interval.



```
In [215]: from time import sleep
          from random import randint

          sleep(randint(1,4))
```

Since the web scraping is more than 10 pages, it would be nice to find a way to monitor the scraping process as it's still going. The greater the number of pages, the more helpful the monitoring becomes.

For my script, I'll make use of this feature, and monitor the following parameters:

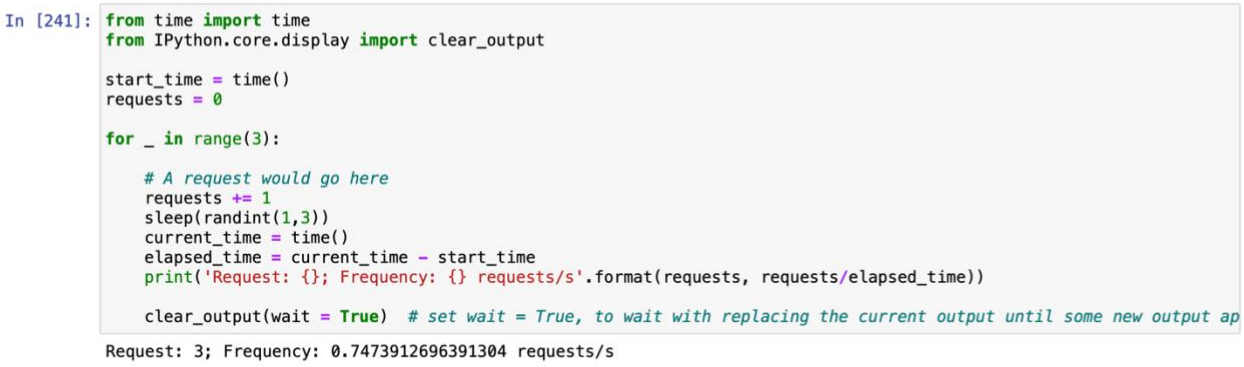
1) The *frequency (speed) of requests*, to ensure our program is not overloading the server
2) The *number of requests*, so I can halt the loop in case the number of expected requests is exceeded
3) The *status code of our requests*, to make sure the server is sending back the proper responses

To get a frequency value, I divide the number of requests by the time elapsed since the first request.

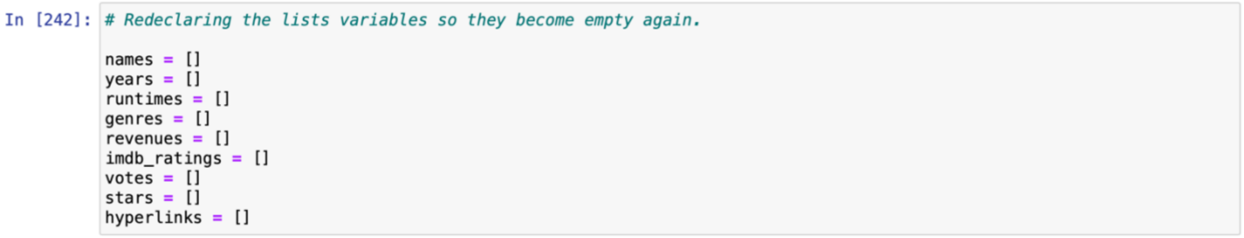
1) Set a starting time using the *time()* function from the time module, and assign the value to start_time.
2) Assign 0 to the variable requests which to be use to count the number of requests.
3) Start a loop, and then with each iteration:

- Simulate a request
- Increment the number of requests by 1
- Pause the loop for a time interval between 8 and 15 seconds
- Calculate the elapsed time since the first request, and assign the value to elapsed_time
- Print the number of requests and the frequency

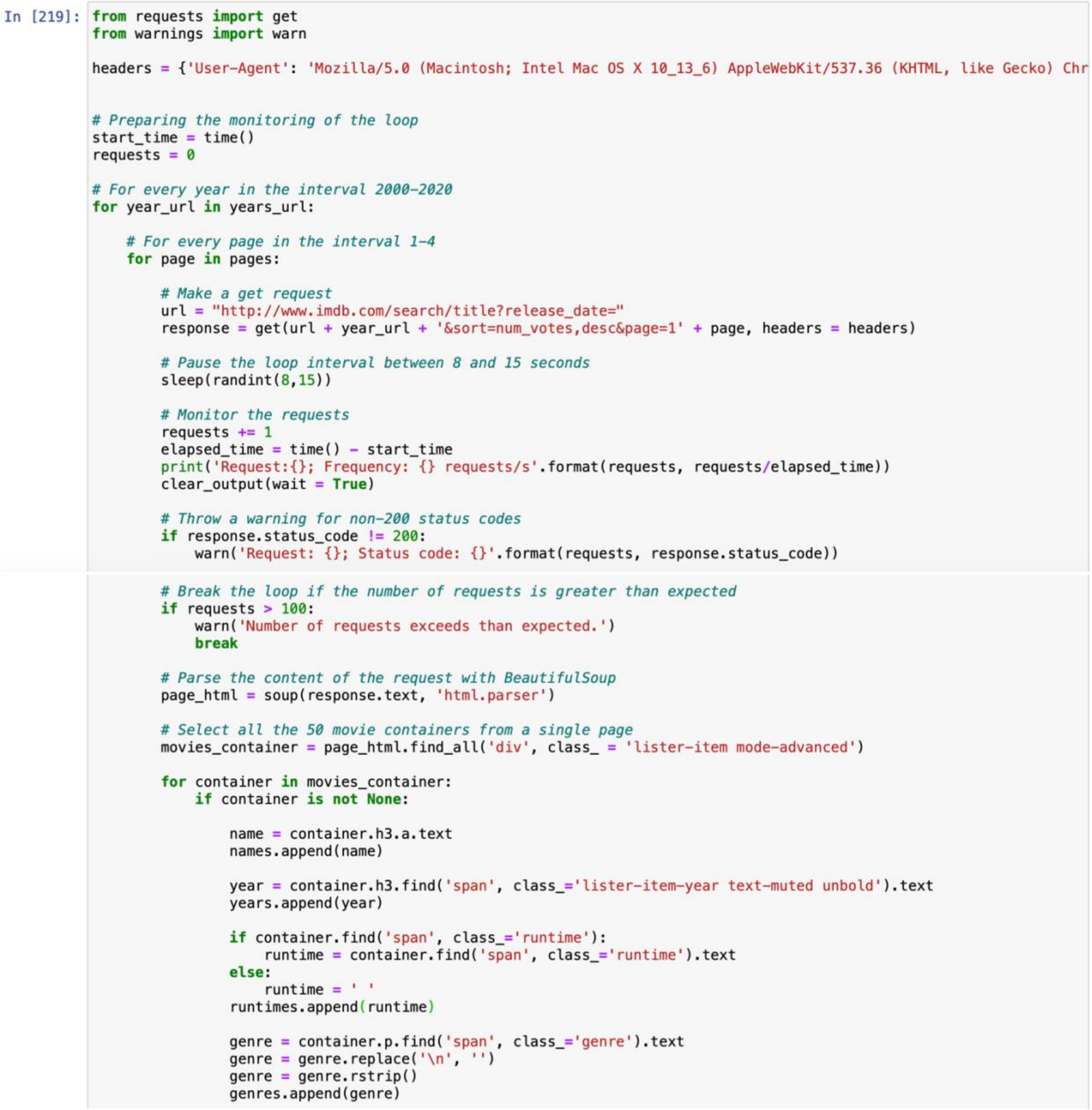
If I set my request to 100 requests, my output return will look a lengthy and a bit untidy as the output accumulates. To avoid that, I'll clear the output after each iteration, and replace it with information about the most recent request. I use the *clear_output()* function from the IPython's core.display module. Then set the wait parameter of clear_output() to True to wait with replacing the current output until some new output appears.



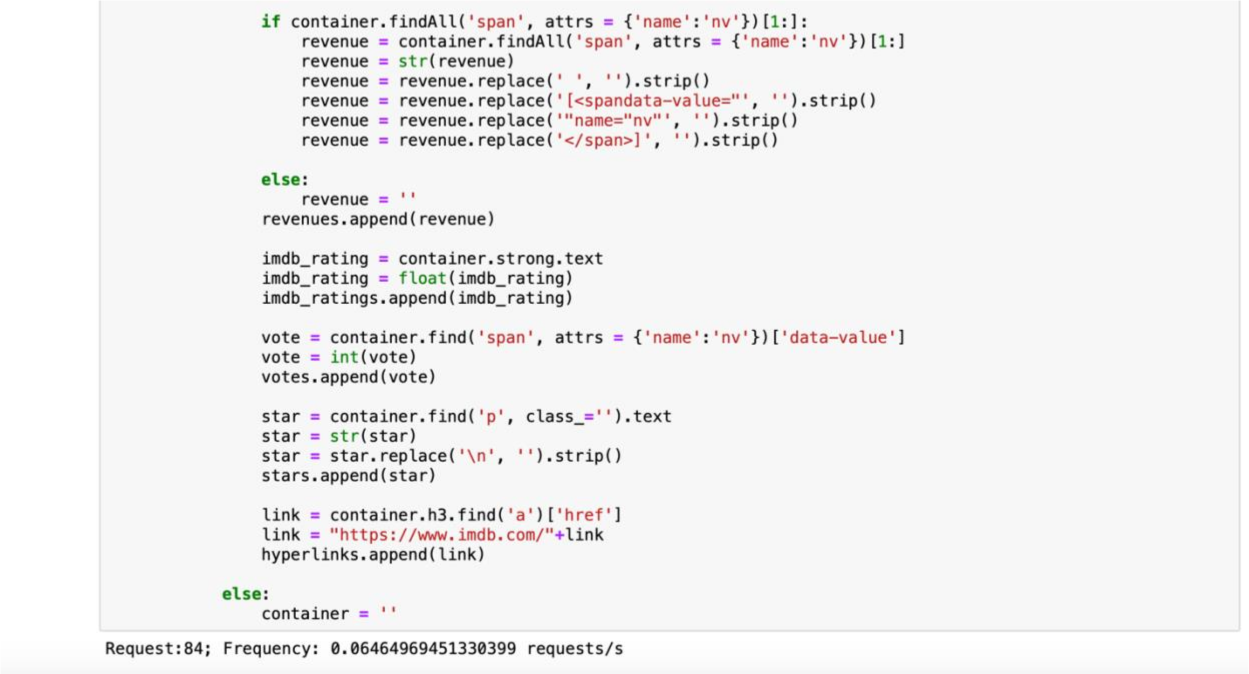
```
In [241]: from time import time
          from IPython.core.display import clear_output

          start_time = time()
          requests = 0

          for _ in range(3):

              # A request would go here
              requests += 1
              sleep(randint(1,3))
              current_time = time()
              elapsed_time = current_time - start_time
              print('Request: {}; Frequency: {} requests/s'.format(requests, requests/elapsed_time))

              clear_output(wait = True)  # set wait = True, to wait with replacing the current output until some new output ap

          Request: 3; Frequency: 0.7473912696391304 requests/s
```



```
In [242]: # Redeclaring the lists variables so they become empty again.

          names = []
          years = []
          runtimes = []
          genres = []
          revenues = []
          imdb_ratings = []
          votes = []
          stars = []
          hyperlinks = []
```



```
In [219]: from requests import get
          from warnings import warn

          headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chr


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

          # For every year in the interval 2000-2020
          for year_url in years_url:

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

                  # Make a get request
                  url = "http://www.imdb.com/search/title?release_date="
                  response = get(url + year_url + '&sort=num_votes,desc&page=1' + page, headers = headers)

                  # Pause the loop interval between 8 and 15 seconds
                  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 > 100:
                      warn('Number of requests exceeds than expected.')
                      break

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

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

                  for container in movies_container:
                      if container is not None:

                          name = container.h3.a.text
                          names.append(name)

                          year = container.h3.find('span', class_='lister-item-year text-muted unbold').text
                          years.append(year)

                          if container.find('span', class_='runtime'):
                              runtime = container.find('span', class_='runtime').text
                          else:
                              runtime = ' '
                          runtimes.append(runtime)

                          genre = container.p.find('span', class_='genre').text
                          genre = genre.replace('\n', '')
                          genre = genre.rstrip()
                          genres.append(genre)
```

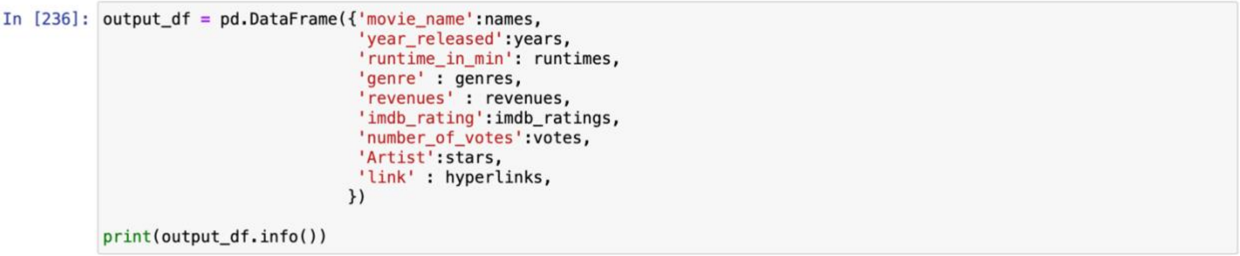


```
            if container.findAll('span', attrs = {'name':'nv'})[1:]:
                revenue = container.findAll('span', attrs = {'name':'nv'})[1:]
                revenue = str(revenue)
                revenue = revenue.replace(' ', '').strip()
                revenue = revenue.replace('[<spandata-value="', '').strip()
                revenue = revenue.replace('"name="nv"', '').strip()
                revenue = revenue.replace('</span>]', '').strip()

            else:
                revenue = ''
            revenues.append(revenue)

            imdb_rating = container.strong.text
            imdb_rating = float(imdb_rating)
            imdb_ratings.append(imdb_rating)

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

            star = container.find('p', class_='').text
            star = str(star)
            star = star.replace('\n', '').strip()
            stars.append(star)

            link = container.h3.find('a')['href']
            link = "https://www.imdb.com/"+link
            hyperlinks.append(link)

        else:
            container = ''
```

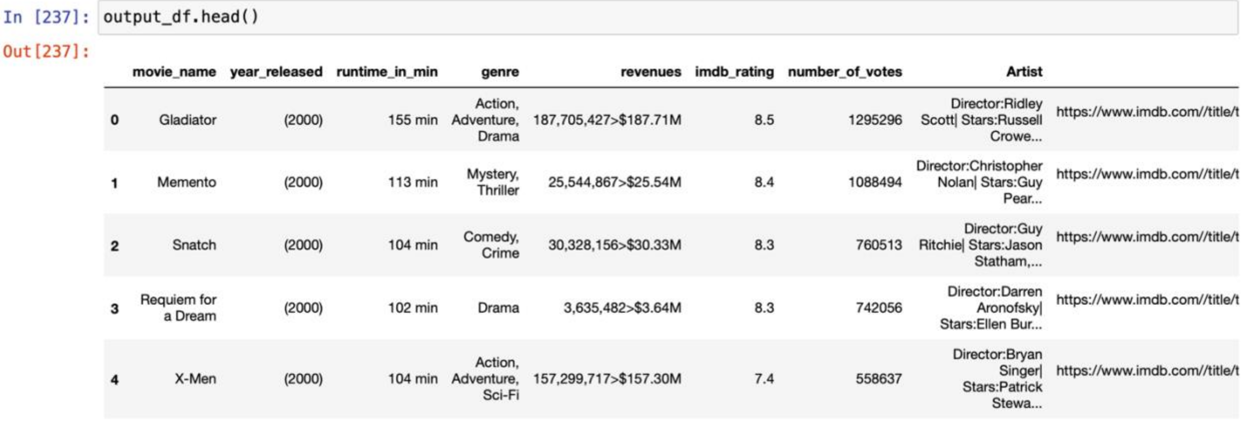
```
Request:84; Frequency: 0.06464969451330399 requests/s
```

I set the loop limit to 100 but the requests stop at request number 84. This indicates that I have collected all TV shows and movies data from year 2000 – 2020.

Print 10 movie_container in first web page



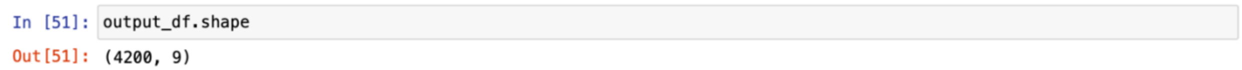
```
In [236]: output_df = pd.DataFrame({'movie_name':names,
                                    'year_released':years,
                                    'runtime_in_min': runtimes,
                                    'genre' : genres,
                                    'revenues' : revenues,
                                    'imdb_rating':imdb_ratings,
                                    'number_of_votes':votes,
                                    'Artist':stars,
                                    'link' : hyperlinks,
                                   })

          print(output_df.info())
```



```
In [237]: output_df.head()
Out[237]:
```

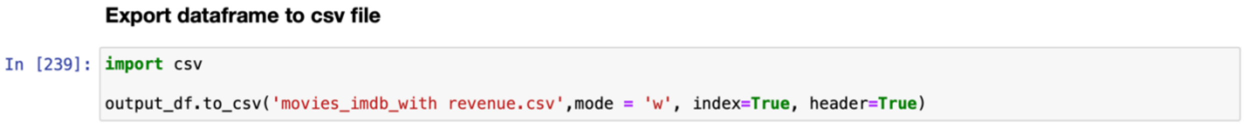
	movie_name	year_released	runtime_in_min	genre	revenues	imdb_rating	number_of_votes	Artist	
0	Gladiator	(2000)	155 min	Action, Adventure, Drama	187,705,427>$187.71M	8.5	1295296	Director:Ridley Scott| Stars:Russell Crowe...	https://www.imdb.com//title/t
1	Memento	(2000)	113 min	Mystery, Thriller	25,544,867>$25.54M	8.4	1088494	Director:Christopher Nolan| Stars:Guy Pear...	https://www.imdb.com//title/t
2	Snatch	(2000)	104 min	Comedy, Crime	30,328,156>$30.33M	8.3	760513	Director:Guy Ritchie| Stars:Jason Statham,...	https://www.imdb.com//title/t
3	Requiem for a Dream	(2000)	102 min	Drama	3,635,482>$3.64M	8.3	742056	Director:Darren Aronofsky| Stars:Ellen Bur...	https://www.imdb.com//title/t
4	X-Men	(2000)	104 min	Action, Adventure, Sci-Fi	157,299,717>$157.30M	7.4	558637	Director:Bryan Singer| Stars:Patrick Stewa...	https://www.imdb.com//title/t

Total number of TV shows and movies collected are 4200 titles.



```
In [51]: output_df.shape
Out[51]: (4200, 9)
```

Export dataframe to csv file

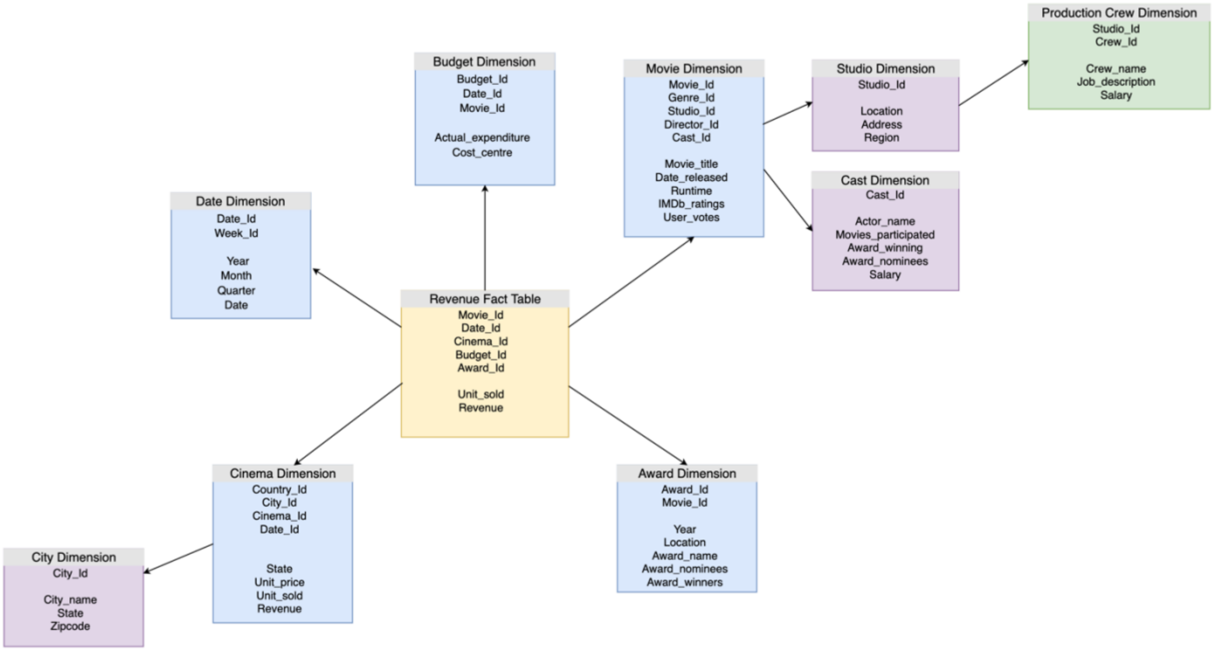


Export dataframe to csv file

```
In [239]: import csv

          output_df.to_csv('movies_imdb_with revenue.csv',mode = 'w', index=True, header=True)
```

(10 marks)

2. Draw snowflake schema diagram for the above dataset. Justify your attributes to be selected in the respective dimensions.



Snowflakes Diagram data model for Movie Data Warehouse

Fact Table:

1) Revenue Fact Table
    a. Primary Key: Movie_Id, Date_Id, Cinema_Id, Budget_Id, Award_Id
    b. Foreign Key: Unit_sold and Revenue

Dimension Table:

1) Movie Dimension Table
    a) Primary Key: Movie_Id, Genre_Id, Studio_Id, Director_Id, Cast_Id
    b) Foreign Key: Movie_title, Date_released, Runtime, IMDb_ratings, User_votes

2) Cast Dimension Table
    a) Primary Key: Cast_Id
    b) Foreign Key: Actor_name, Movies_participated, Award_winning, Award_nominees, Salary

3) Studio Dimension Table
    a) Primary Key: Studio_Id
    b) Foreign Key: Location, Address, Region

4) Production Crew Dimension Table
    a) Composite Key: Studio_Id, Crew_Id
    b) Foreign Key: Crew_name, Job_description, Salary

5) Award Dimension Table
    a) Composite Key: Award_Id, Movie_Id
    b) Foreign Key: Year, Location, Award_name, Award_nominees, Award_winners

6) Cinema Dimension Table
    a) Composite Key: Cinema_Id, City_Id
    b) Primary Key: Country_Id, Date_Id
    c) Foreign Key: State, Unit_price, Unit_sold, Revenue

7) City Dimension Table
    a) Primary Key: City_Id
    b) Foreign Key: City_name, State, Zipcode

8) Budget Dimension Table
    a) Primary Key: Budget_Id, Date_Id, Movie_Id
    b) Foreign Key: Actual_expenditure, Cost_centre

9) Date Dimension Table
    a) Primary Key: Date_Id, Week_Id
    b) Foreign Key: Year, Month, Quarter

Description:

a) In snowflakes schema, **Primary Key** is that column of the table whose every row data is uniquely identified. Every row in the table must have a primary key and no two rows can have the same primary key. Primary key value can never be null nor can be modified or updated. The **Composite Key** is a form of the candidate key where a set of columns will uniquely identify every row in the table. For example, Cinema_Id and City_Id in Cinema dimension table. **Foreign Key** are the columns of a table that points to the key of another table. They act as a cross-reference between tables.
b) OLAP operations consist of 5 types of operation which are
    - Drill down - converting less detailed data into highly detailed data
    - Roll Up - climbing up in the concept hierarchy
    - Dice - select a sub-cube from the OLAP by selecting two or more dimensions
    - Slice - select a sub-cube from the OLAP which results in a new sub-cube creation.
    - Pivot - rotates the current view to get a new view of the representation
c) For my data warehouse, I am using typical OLAP operation which are **Roll Up** and **Drill Down** based on 1 fact table (Revenue) and 9 dimension tables (Movie, Studio, Cast, Production Crew, Award, Cinema, City, Budget and Date)
d) **Drill Down** is to examine my summary data which break out by dimension attributes. From higher level to lower level summary; introducing new dimension. Based on my Movie_Id in Revenue fact table to Movie, Budget and Award dimension table. Another example is Studio_Id in Movie dimension table to Studio and Production Crew dimension table
e) **Roll Up** is operation by climbing up hierarchy or by dimension reduction. For instance, my attribute of City_Id in City dimension table to Cinema dimension table and Studio_Id in Production Crew dimension table to Movie dimension table.

(10 marks)

3. You are required to write code to create a decision tree (DT) model using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:
   - Importing required libraries
   - Loading Data
   - Feature Selection
   - Splitting Data
   - Building Decision Tree Model
   - Evaluating Model
   - Visualizing Decision Trees

(10 marks)

4. You are required to write code to find frequent itemsets using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:
   - Importing required libraries
   - Creating a list from dataset (Question 1)
   - Convert list to dataframe with boolean values
   - Find frequently occurring itemsets using Apriori Algorithm
   - Find frequently occurring itemsets using F-P Growth
   - Mine the Association Rules

(10 marks)

5. It will be appeared in week 14.

(10 marks)

Submissions:

The student is expected to submit answers to each question individually, and submit the document in PDF format. The student can include online materials, screenshots, videos and/or codes (ipynb format) to support your answer