Batta.jpg

/TECHNIQUE DE VEILLE

Web Scrapping and Sentiment Analysis

Provided by :

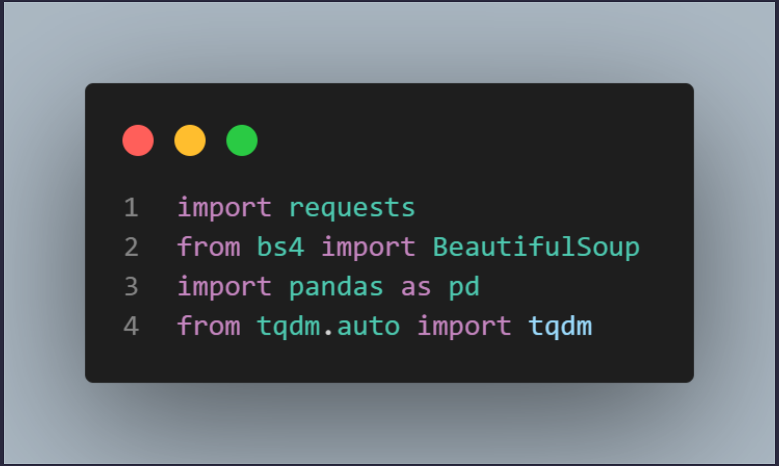
Ameny IHKAF
Mehdi SOMRANI

INDEX.HTML

/Scrapping movies

Used Libraries

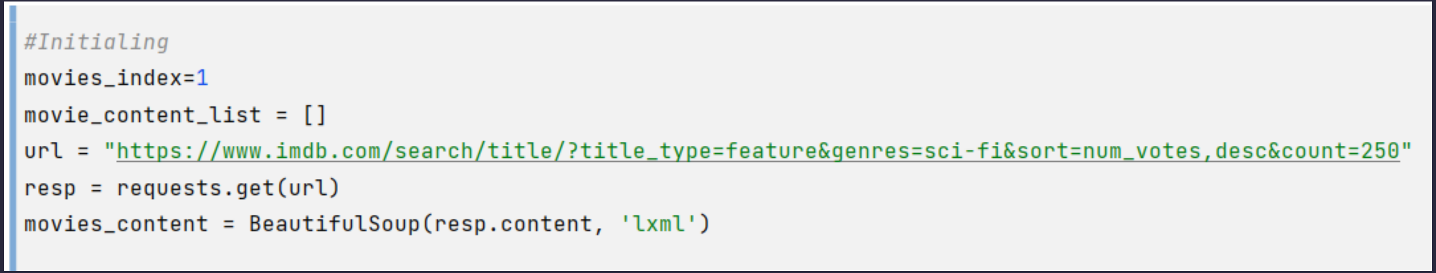
- `import requests`
  - Used to get HTTP requests
- `from bs4 import BeautifulSoup`
  - Used to scrap web page
- `import pandas as pd`
  - Used to manipulate dataframes
- `from tqdm.auto import tqdm`
  - Used to implement progress bar in loops



```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
from tqdm.auto import tqdm
```

/Scrapping movies

Initializing variables to use late such as the url we are going to request data from and converting the **request.get** response to **LXML** text type using **BeautifulSoup**

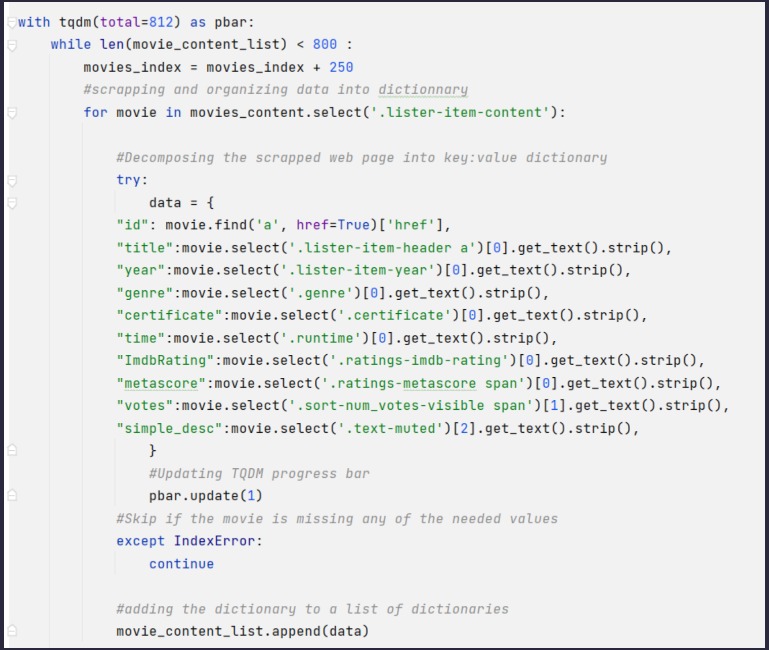


```
#Initialing
movies_index=1
movie_content_list = []
url = "https://www.imdb.com/search/title/?title_type=feature&genres=sci-fi&sort=num_votes,desc&count=250"
resp = requests.get(url)
movies_content = BeautifulSoup(resp.content, 'lxml')
```

Batta.jpg

/Scrapping movies

Starting the scrapping process with integrating the TQDM library to track the progress in addition to getting needed data while removing unneeded characters and spaces using **strip()** in a try exception method. Furthermore, we used a **Movie_index** variable in the url to jump to next page to scrap more movies.
Excluding the movies that misses one or more of the needed values and appending the result to list of dictionaries named **movie_content_list**

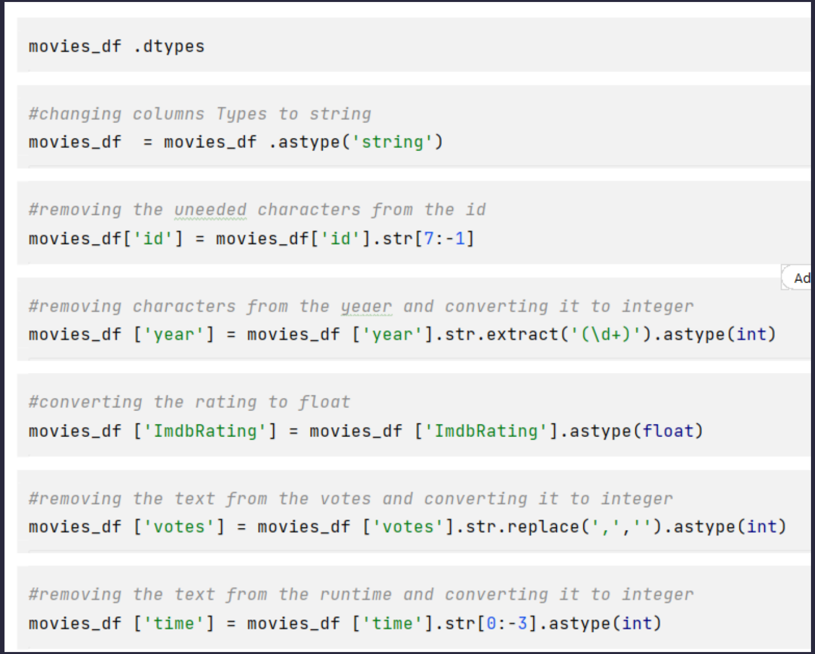


```
with tqdm(total=812) as pbar:
    while len(movie_content_list) < 800 :
        movies_index = movies_index + 250
        #scrapping and organizing data into dictionnary
        for movie in movies_content.select('.lister-item-content'):

            #Decomposing the scrapped web page into key:value dictionary
            try:
                data = {
            "id": movie.find('a', href=True)['href'],
            "title":movie.select('.lister-item-header a')[0].get_text().strip(),
            "year":movie.select('.lister-item-year')[0].get_text().strip(),
            "genre":movie.select('.genre')[0].get_text().strip(),
            "certificate":movie.select('.certificate')[0].get_text().strip(),
            "time":movie.select('.runtime')[0].get_text().strip(),
            "ImdbRating":movie.select('.ratings-imdb-rating')[0].get_text().strip(),
            "metascore":movie.select('.ratings-metascore span')[0].get_text().strip(),
            "votes":movie.select('.sort-num_votes-visible span')[1].get_text().strip(),
            "simple_desc":movie.select('.text-muted')[2].get_text().strip(),
                }
                #Updating TQDM progress bar
                pbar.update(1)
            #Skip if the movie is missing any of the needed values
            except IndexError:
                continue

            #adding the dictionary to a list of dictionaries
            movie_content_list.append(data)
```

PLAN.DATL

/Cleaning DataFrame

Using pandas, we’ve changed the types of the columns and removing the unnecessary characters.
We didn’t check for null since we skipped every movie that have any null value



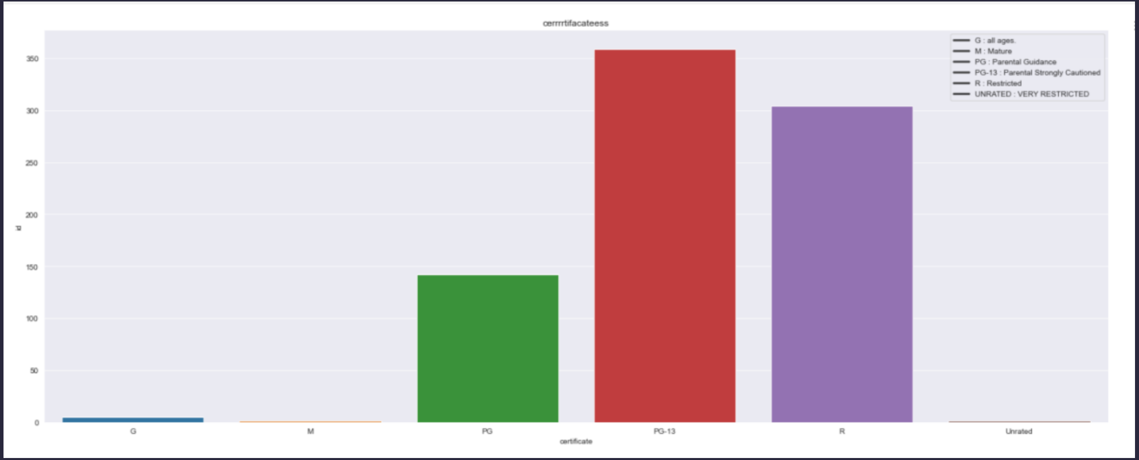
```
movies_df .dtypes

#changing columns Types to string
movies_df  = movies_df .astype('string')

#removing the uneeded characters from the id
movies_df['id'] = movies_df['id'].str[7:-1]

#removing characters from the yeqer and converting it to integer
movies_df ['year'] = movies_df ['year'].str.extract('(\d+)').astype(int)

#converting the rating to float
movies_df ['ImdbRating'] = movies_df ['ImdbRating'].astype(float)

#removing the text from the votes and converting it to integer
movies_df ['votes'] = movies_df ['votes'].str.replace(',','').astype(int)

#removing the text from the runtime and converting it to integer
movies_df ['time'] = movies_df ['time'].str[0:-3].astype(int)
```

Batta.jpg

/Data Visualization

Using Matplotlib and Seaborn, we visualized the number of sci-movie made in each certificate rate such as PG-13 and R rated movies

As we see most of sci-fi movies are PG-13 rated

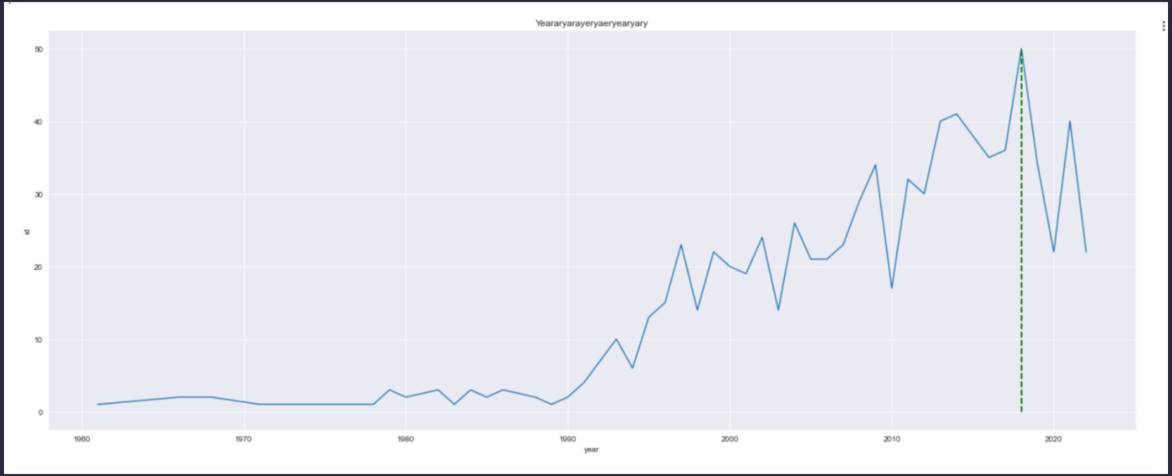


PLAN.DATL

/Data Visualization

Using Matplotlib and Seaborn again, We plotted the number of movies made each year with highlighting the peak year, which it was 2019.

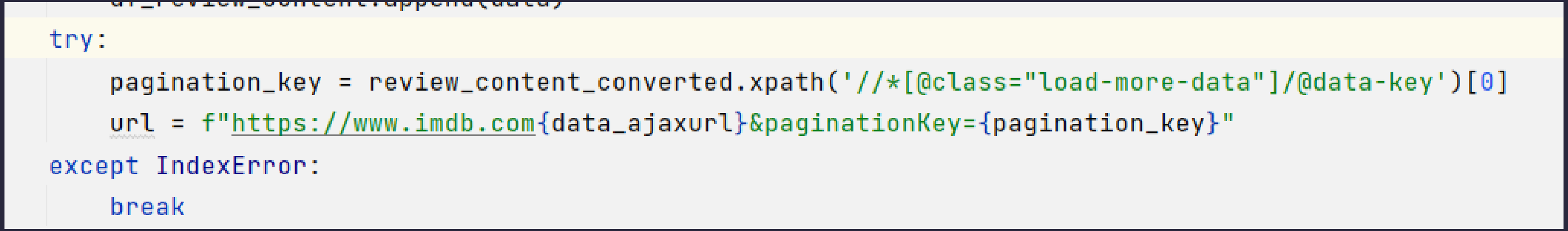
2019 was a year full of sci-fi movies



Batta.jpg

/Scrapping reviews

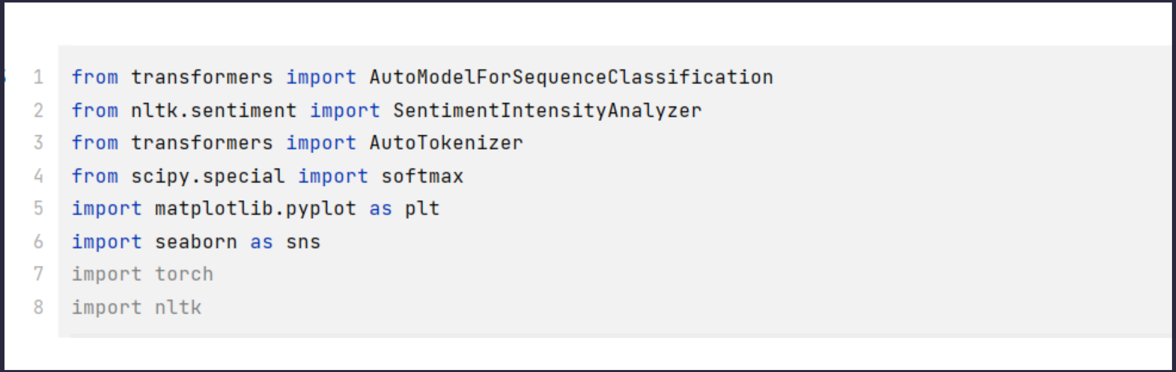
To scrap reviews, we used the same scrapping movie code but instead of changing the url with starting movie rank, we used the **ajax-url,** which is the id on the button of **loadmore.** And the **paginationkey,** which is the key that takes us to load more data. We used a while true loop to scrap all the reviews of the movie.



```
try:
    pagination_key = review_content_converted.xpath('//*[@class="load-more-data"]/@data-key')[0]
    url = f"https://www.imdb.com{data_ajaxurl}&paginationKey={pagination_key}"
except IndexError:
    break
```

/Sentiment analysis

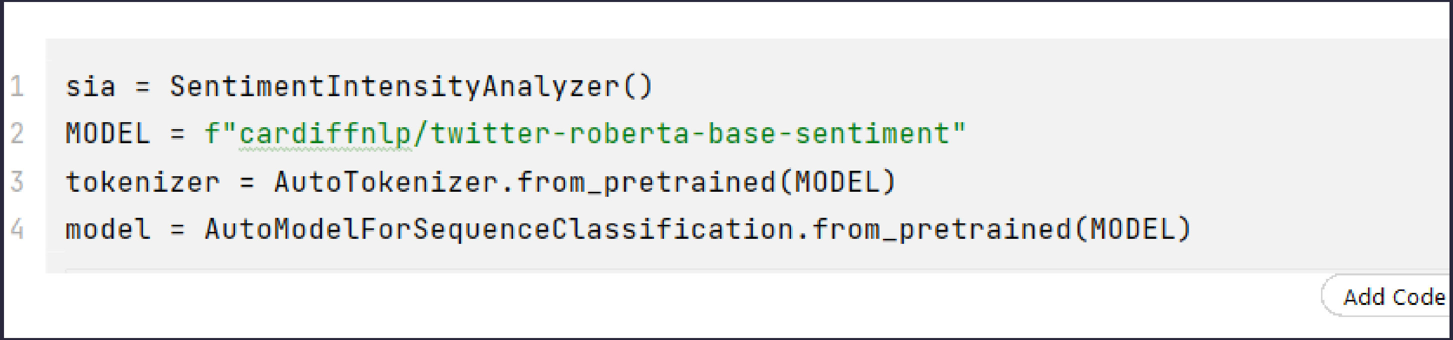
In our sentiment analysis mission, we used the **Roberta model** to get the positive, negateive and neuteral score of each review. This model requires **NLTK,Scipy** and **Transformers, Pytorch** libraries to use **SentimentIntensityAnalyser, AutoTokenizer, Softmax** and we used **pytorch** library since softmax depends on it



```
from transformers import AutoModelForSequenceClassification
from nltk.sentiment import SentimentIntensityAnalyzer
from transformers import AutoTokenizer
from scipy.special import softmax
import matplotlib.pyplot as plt
import seaborn as sns
import torch
import nltk
```

/Sentiment analysis

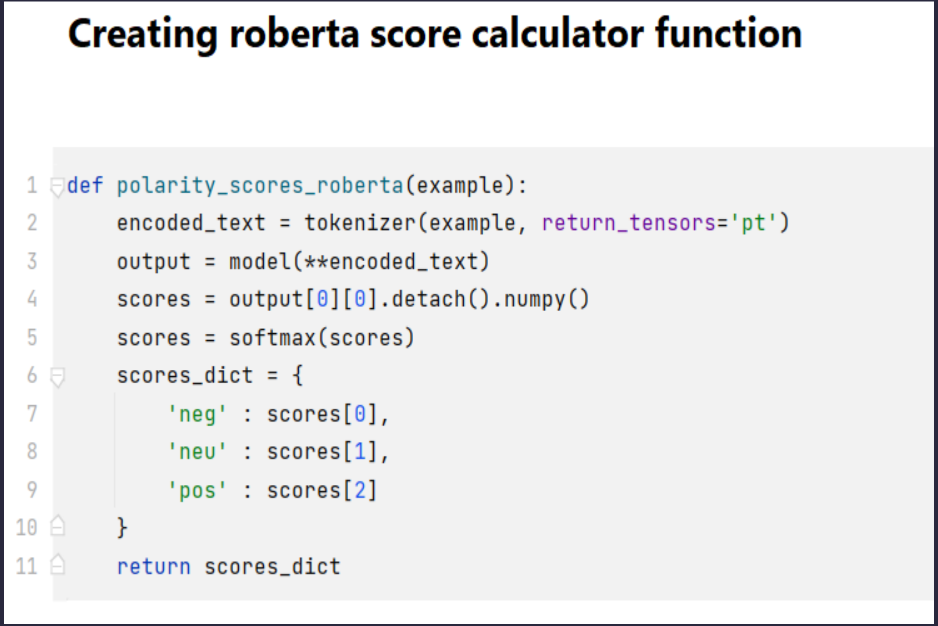
After importing needed libraries, we need to declare our needed variables
We declared the **Sentiment analyzer** and the model we are going to use, which is **Roberta** in our case, **tokenizer** from a pretrained model to scan lexical and the model to get the **pretrained** version of our model



```python
sia = SentimentIntensityAnalyzer()
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
```

/Sentiment analysis

Polarity scores Roberta uses the tokenizer function to scan and encode the text so the model could analyze the sentiments and saving it to output. Then, score detach the output and make it to numpy array, before saving the score to a dictionary we needed to decode the score we found And finally we return the dictionary

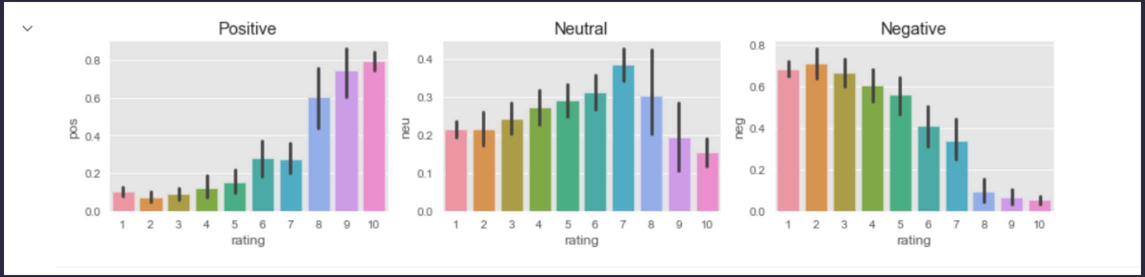


Creating roberta score calculator function

```python
def polarity_scores_roberta(example):
    encoded_text = tokenizer(example, return_tensors='pt')
    output = model(**encoded_text)
    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
    scores_dict = {
        'neg' : scores[0],
        'neu' : scores[1],
        'pos' : scores[2]
    }
    return scores_dict
```

/Sentiment analysis

After scrapping the reviews and configuring the model we calculated the sentiment on each row in the reviews **dataframe** for 5 movies and plotted it using matplotlib and seaborn showing the accuracy between the **positive**, **neutral** and **negative** reviews and their rating of the move


Positive
Neutral
Negative
pos
neu
neg
rating