Institute of Technology of Cambodia

Programming For Data Science
2020-2021

3rd year Engineer's Degree in Data Science

Department of Applied Mathematics and Statistics

Movie Recommendation System

Group members:

Name	ID
Tang Piseth	e20201634
Set Mongkol	e20201255
Thornthea Gechhai	e20201321
You Phakkorn	e20200727
Sreng Seangleng	e20200840
Thong Chhunher	e20200711

Lecturers:

Prof.Chan Sophal

Contents

1 Data preparation

In order to use data in a proper way, the raw data as described in a detailed format (in our last paper) has to be preprocessed, explored and engineered in several ways. In this section, we will elaborate further on this process.

1.1 Data preprocessing

In the early stage , we decided to cover small changes such as replacing whites- pace character by "_" in every column's name(we believe this is a good prac- tice), changing " Gross_Collection($)" to Gross_ Collection(M$)" since the gross collection of each movie is in millions of dollars(M$), replacing '****' by NaN val- ues as we believe that the built-in pan- das functions will have a much easier time dealing with NaN values instead of the string "****", .

```
0        II 2014
1           2020
2           2008
3         I 2017
4           2017
           ...
9834      I 2009
9835      I 2012
Name: Year_of_Release,
Length: 9836,
dtype: object
```

Thus, we see that there are some problems with the data types of the following features: "Year_of_Release", "Watch_time", "Meta_Score", "Votes", and "Gross_Collection(M$)". We will deal with them(changing their data types to appropriate one, numerical) one by one by using 2 functions below for cleaning the year releases and votes:

After trials and errors, we were able to Gross_Collection(M$) and Meta_Score just by manually changing their data types.

1.1.1 Analysis on missing / null values

We will provide details on how we dealt with values not present in our dataset. Firstly, let's check the number of miss- ing values. We check for any rows whose features' values contain 'NaN'. Even if one of the movie's feature contains 'Nan', we will also include that into our "num_missing_rows" variables.

```
# count the number of rows with at least one
mis
missing_rows = df.isna().any(axis=1).sum()
# count the number of rows without missing
value
non_missing_rows = len(df) - missing_rows
missing_rows
```

```
[Output] : 3351
```

Remember that our dataset contains 9836 records, and what is the proportion of 3351 out of 9836?

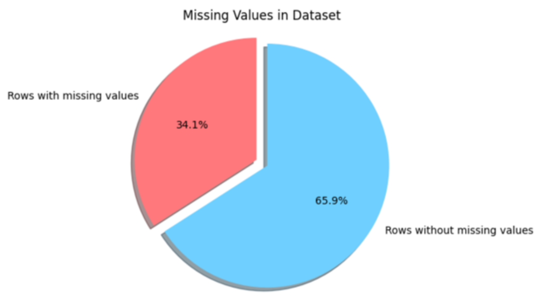


Figure 1: Missing Values in Data-set

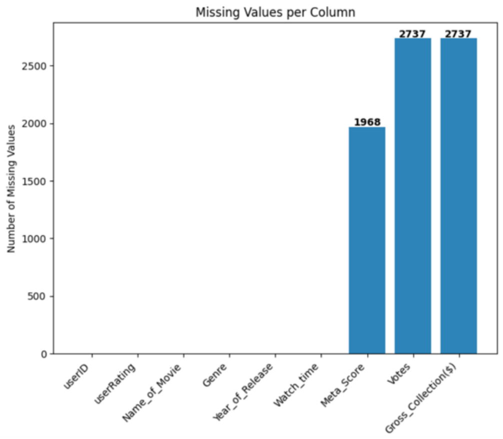
There are so many missing values (by our definition of missing values) in our entire dataset, more than 34%, we certainly do not blindly remove 34% of our dataset.

Thus, we will perform further analysis into the missing values present in each column

```
Number of rows with missing data:
userID                 0
userRating             0
Name_of_Movie          0
Genre                  0
Year_of_Release        0
Watch_time             0
Meta_Score          1968
Votes               2737
Gross_Collection($) 2737
```

And in percentage, that is:

```
userID                  0.000000
userRating              0.000000
Name_of_Movie           0.000000
Genre                   0.000000
Year_of_Release         0.000000
Watch_time              0.000000
Meta_Score             20.008133
Votes                  27.826352
Gross_Collection($)    27.826352
```



As we can see so far, there are 34 % of the entire movies that contain missing values in at least one of its features, that is a lot. And column-wise we have 20.008133% of movies which do have missing meta scores and 27.826352 % of movies which have no gross collection and votes. That is a quarter of all our data set, removing those movies will heavily im- pact the data analysis of the features' distribution. Moreover, we cannot just ignore missing values. So, we will resort to imputing them with either the mean, median or the mode. Here is a brief overview of when to impute missing values with the mean and median and the mode but in order to know which type of it we should replace all the missing value first we have to plot the data distributions of the movies' Meta Score, Votes, and Gross Collection(M$). Firstly, it is obvious that since the 3 mentioned features are numerical, then imputing their instances with the 'mode' is not logical so 'mode' is out of the option leaving only median and mean.

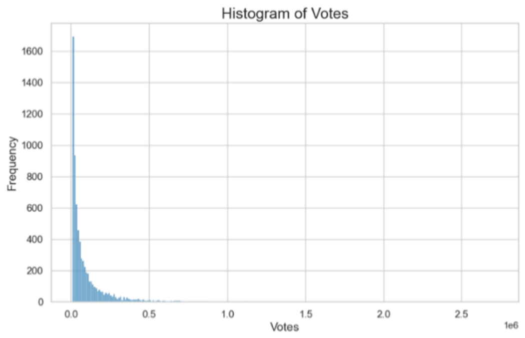


Figure 2: Histogram of Votes

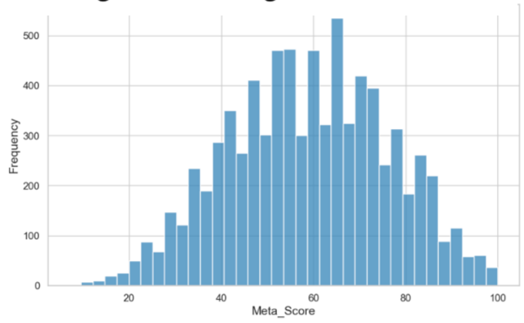


Figure 3: Histogram of Mela_Score

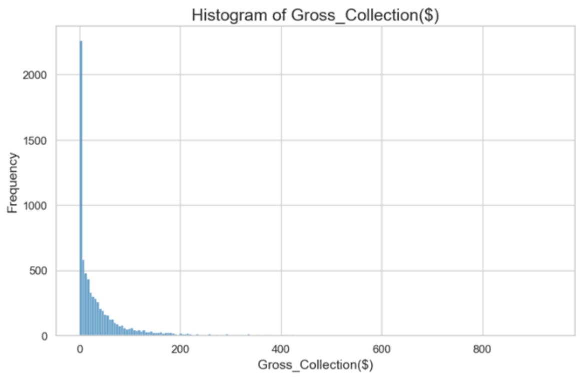


Figure 4: Histogram of Gross_Collection ($)

We can see skewness coming from Votes, and Gross_Collection(M$) and little skewness from Meta_Score. We will square-root-transform them and compare with the distribution of the originals:

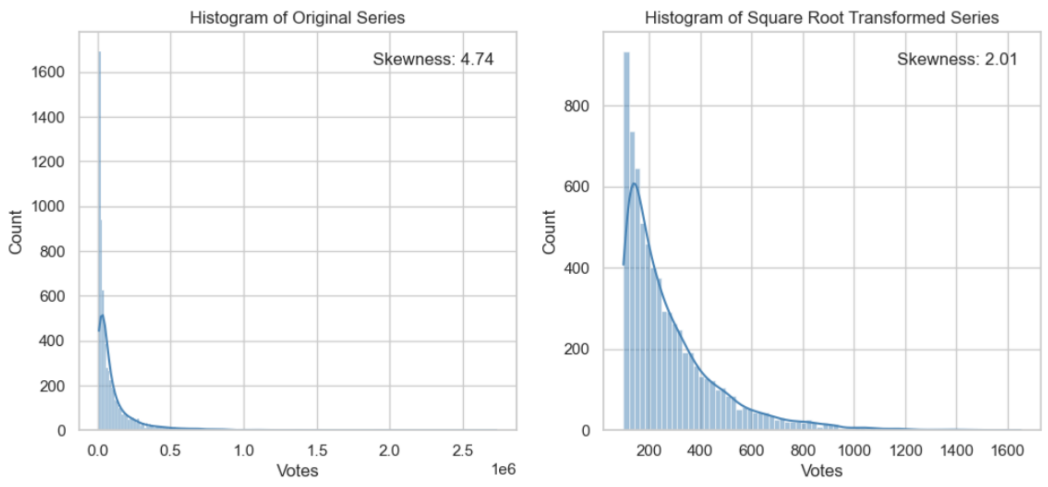


Figure 5

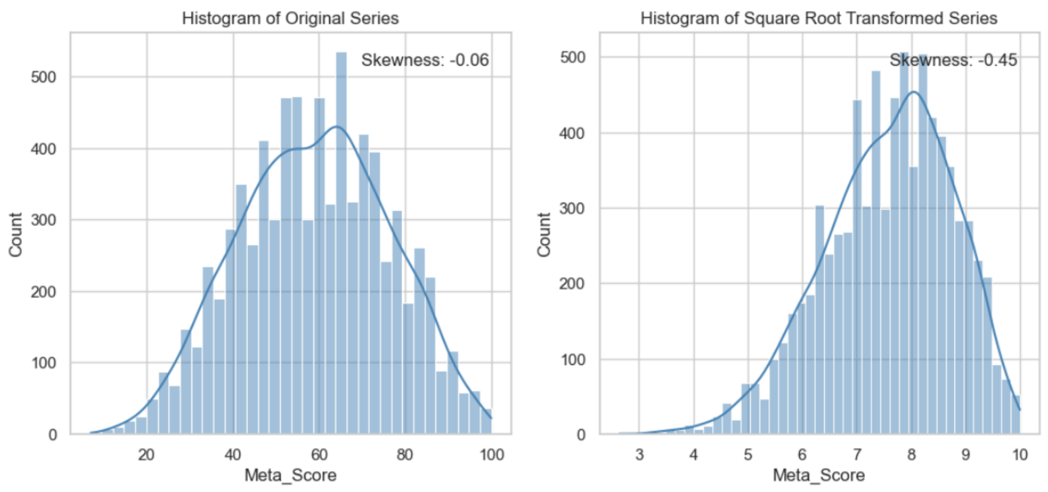


Figure 6

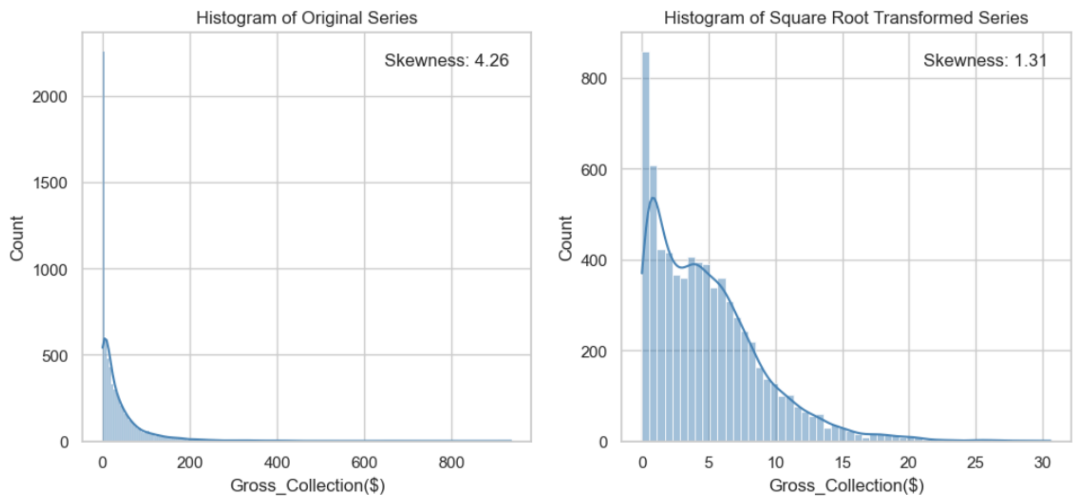
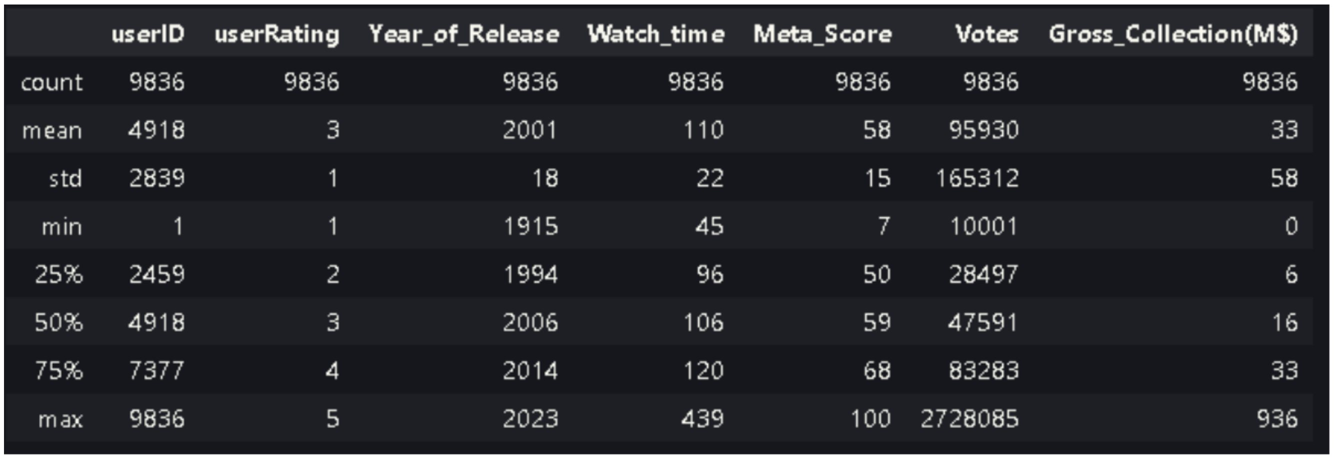


Figure 7

Although the skewness has gotten better, we believe that working with the transformed data will bring more challenges.

Here are the descriptive statistics:



	userID	userRating	Year_of_Release	Watch_time	Meta_Score	Votes	Gross_Collection(M$)
count	9836	9836	9836	9836	9836	9836	9836
mean	4918	3	2001	110	58	95930	33
std	2839	1	18	22	15	165312	58
min	1	1	1915	45	7	10001	0
25%	2459	2	1994	96	50	28497	6
50%	4918	3	2006	106	59	47591	16
75%	7377	4	2014	120	68	83283	33
max	9836	5	2023	439	100	2728085	936

Figure 11

1.1.2 Outlier Analysis

Here is the number of outliers that we found in each numerical column:

```
userID -> 0
userRating -> 0
Year_of_Release -> 567
Watch_time -> 417
Meta_Score -> 190
Votes -> 1312
Gross_Collection(M$) -> 1104
```

Again, visualization is our friend. We will plot histograms of numerical columns with outliers:

1.2 Exploratory Data Analysis

We actually took a detour and perform a feature engineering process (called feature creation) ahead of time. We believe that a data science project's path is not always fixed and the steps can be modified for the project's best interest.

As stated, we will perform feature creation ahead of time by creating additional genre columns and using One-Hot-Encoding to indicate that each unique value in a categorical variable is transformed into a binary vector where each vector has a 1 in the position corresponding to the original value and 0s in all other positions. This technique helps to avoid assigning any arbitrary ordering to categorical variables and allows machine learning models to interpret the categories as **independent** features.

1.2.1 Correlation Analysis

The Gross Collection and Votes values have correlation coefficient up to 0.63 and every other correlation coefficient in the array are below 0.35 so according to that, there is no multi-collinearity between every other feature except between Gross_Collection($) and Votes($). But this is only by the heat map, we will try to test for multi-collinearity with other statistical methods.

1.2.2 Hypothesis Testing

In this section, we use statistical tests to test your hypotheses about the data. For example, you can use a t-test to compare the means of two groups/columns. More details: hypothesis testing can help you to determine the significance of the relation- ship between variables in your dataset. For

example, you could use hypothesis testing to determine whether there is a statistically significant relationship between a movie's genre and its gross revenue.

$H\_0: \mu_1 = \mu_2 = \cdots = \mu_k$
(The mean gross revenue collection for all movie genres are all equal to each other.)

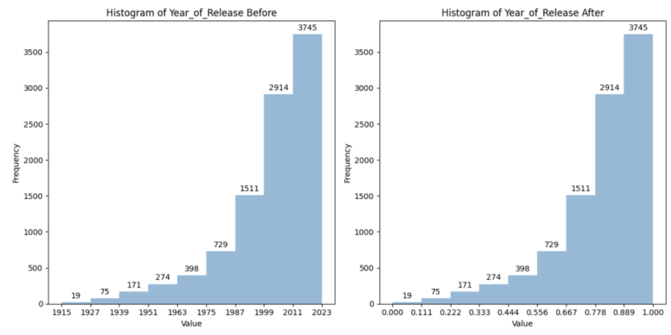
If the hypothesis testing and ANOVA re- veal that certain movie genres are statistically significant with gross collection, it means that there is a significant association between those movie genres and the revenue generated by the movie. This in- formation can be useful for movie studios, producers, and investors in making decisions about which genres to invest in and produce. For example, if action movies are found to be significantly associated with higher gross collection, a movie stu- dio might decide to invest more resources into producing action movies in the future. On the other hand, if a particular genre is found to have a negative association with gross collection, the studio might decide to reduce investments in that genre. Based on the updated dictionary, all of the genres have p-values less than 0.05, which means they are all statistically significant. This suggests that each genre is associated with the revenue generated by a movie. How- ever, some genres have much smaller p- values than others, indicating a stronger association. The genres with the smallest p-values (i.e., the most significant as- sociations) are Action, Adventure, Anima- tion, ... War. The genres with larger p- values (i.e., less significant associations) are Adult, Comedy, Musical, Sport, and Western.

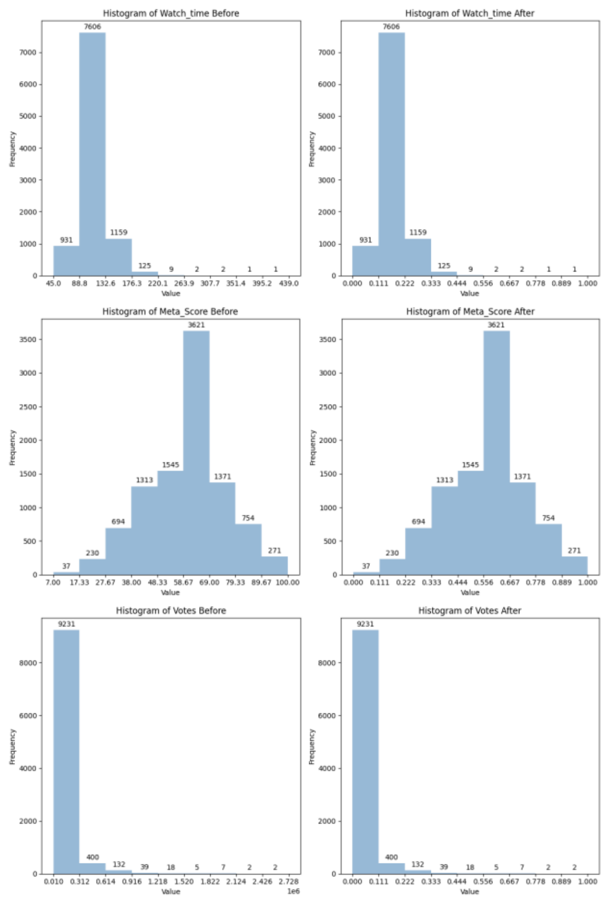
1.3 Feature Engineering

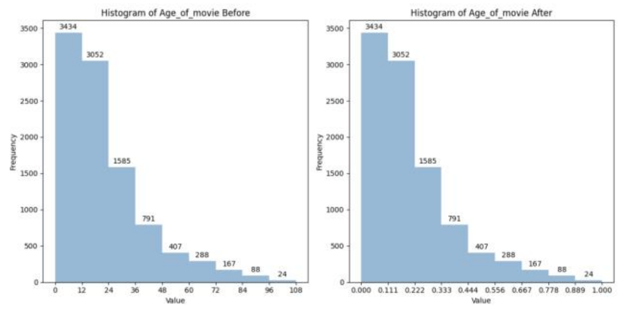
1.3.1 Feature Scaling

Two Common Types of Scaling

- Min-Max Normalization (Normalization): Between 0-1
- Standardization: Representing values in standard deviations from the Mean likely in the ballpark of between -3 to 3.







1.3.2 Feature Selection

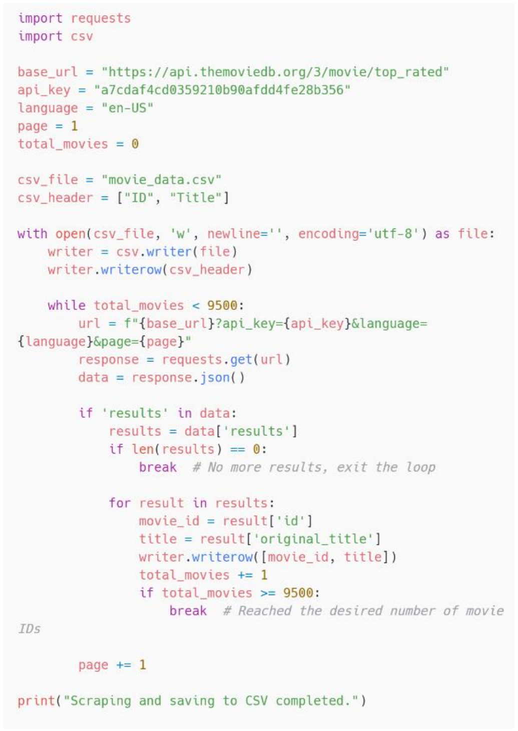
Feature Selection is the process of reducing the input variable to your model by using only relevant data and getting rid of noise in data (not useful data).

Recursive Feature Elimination (Wrapper Method)

After did the Recursive Feature Elimination on the features. Here the result:
>> Selected Features:
'Watch_time', 'Meta_Score', 'Votes', 'Gross_Collection(M$)'

1.3.3 Feature Augmentation

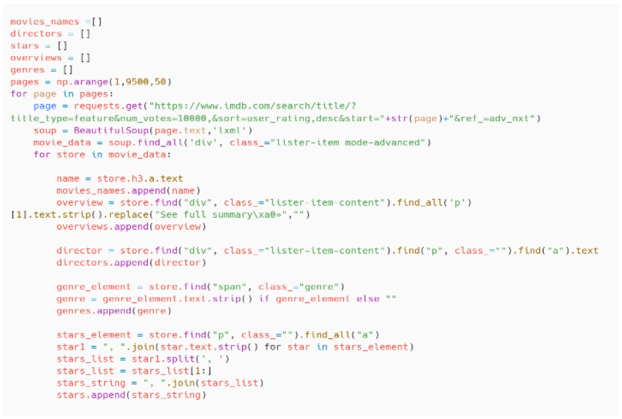
Feature Augmentation, typically involves incorporating additional external data sources or collecting new features to supplement your existing feature set. It aims to enrich the information available to your model and improve its performance.
We will explore a feature augmentation technique by utilizing an API to scrape movie_id from TMDB website. The objective is to collect the movie_id about the top-reated movies from a popular movie database.



```python
import requests
import csv

base_url = "https://api.themoviedb.org/3/movie/top_rated"
api_key = "a7cdaf4cd0359210b90afdd4fe28b356"
language = "en-US"
page = 1
total_movies = 0

csv_file = "movie_data.csv"
csv_header = ["ID", "Title"]

with open(csv_file, 'w', newline='', encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerow(csv_header)

    while total_movies < 9500:
        url = f"{base_url}?api_key={api_key}&language=
{language}&page={page}"
        response = requests.get(url)
        data = response.json()

        if 'results' in data:
            results = data['results']
            if len(results) == 0:
                break  # No more results, exit the loop

            for result in results:
                movie_id = result['id']
                title = result['original_title']
                writer.writerow([movie_id, title])
                total_movies += 1
                if total_movies >= 9500:
                    break  # Reached the desired number of movie
IDs

        page += 1

print("Scraping and saving to CSV completed.")
```

This code utilizes the TMDB (The Movie Database) API to scrape and collect information about top-rated movies. It makes HTTP requests to the API, retrieves the movie data in JSON format, and extracts

the movie IDs and titles. The collected data is then stored in a CSV file for further analysis or storage. The code continues fetching movie data until it has obtained 9,500 movie IDs. Finally, it prints a completion message.
After finished scraping movie_id, we move on into scraping for more features which are more important such as Overview,Star,Director,Genre.



```
movies_names =[]
directors = []
stars = []
overviews = []
genres = []
pages = np.arange(1,9500,50)
for page in pages:
    page = requests.get("https://www.imdb.com/search/title/?
title_type=feature&num_votes=10000,&sort=user_rating,desc&start="+str(page)+"&ref_=adv_nxt")
    soup = BeautifulSoup(page.text,'lxml')
    movie_data = soup.find_all('div', class_="lister-item mode-advanced")
    for store in movie_data:

        name = store.h3.a.text
        movies_names.append(name)
        overview = store.find("div", class_="lister-item-content").find_all('p')
[1].text.strip().replace("See full summary\xa0»","")
        overviews.append(overview)

        director = store.find("div", class_="lister-item-content").find("p", class_="").find("a").text
        directors.append(director)

        genre_element = store.find("span", class_="genre")
        genre = genre_element.text.strip() if genre_element else ""
        genres.append(genre)

        stars_element = store.find("p", class_="").find_all("a")
        star1 = ", ".join(star.text.strip() for star in stars_element)
        stars_list = star1.split(', ')
        stars_list = stars_list[1:]
        stars_string = ", ".join(stars_list)
        stars.append(stars_string)
```

This code utilizes web scraping techniques to extract movie data from IMDb's advanced search feature. It collects information about highly rated feature films with a minimum of 10,000 votes. The code iterates through multiple pages of search results, retrieves details such as movie names, overviews, directors, stars, and genres, and stores them in separate lists. This data can be further analyzed or stored for various purposes, such as research or creating a movie database.
After done with scraping for new features, we then merge the new features into the old dataset.

2. Content Based Recommender

The Content Based Recommender offers generalized recommendations to user based on a particular item. This system uses item metadata such as:
- Movie Star, Director, Overview, Genre, etc.

For movies, to make this recommendation. The general idea behind these recommender systems is that if a person liked a particular item, he or she will also like an item that is similar to it.

2.1 Recommendation Approach

So, after having some new features for our model. We want to combine various movie features, including Genre, Directors, Stars, Watch Time, Year of Release, Meta Score, Votes, Gross Collection, and Overviews, into a single column called 'tags’. By consolidating these features, we aim to perform PorterStemmer,a Bag-of-Words analysis using CountVectorizer. This approach allows us to represent movies as a collection of words or tags, enabling further analysis, such as classification, or recommendation systems.

PortStemmer:

PortStemmer is a powerful text processing technique that can be applied to the 'Tags' column in movie analysis. Stemming is the process of reducing words to their base or root form, enabling better grouping and analysis of related terms. By applying PortStemmer to the 'Tags' column, we can further enhance our understanding and analysis of movie data.

The Bag-of-Words Technique:

The Bag-of-Words technique is a commonly used method in natural language processing and text analysis. It represents a piece of text, in this case, the 'tags' column, as a collection of words, ignoring grammar, word order, and context. Each unique word in the 'tags' column becomes a feature, and its frequency is used as a numerical value representing the importance of that word in a particular movie.

CountVectorizer:

Once we have the 'Tags' column, we can apply the CountVectorizer technique to convert the textual data into numerical feature vectors. CountVectorizer will split it into individual words or tags, and build a vocabulary of unique words. It will then transform each movie's 'tags' into a numerical vector representation, with each entry representing the frequency of a specific word or tag.

Applications and Benefits:
By combining movie features into 'tags' and applying CountVectorizer, we can gain valuable insights and perform various analyses. This approach enables us to identify frequently occurring words or tags across movies, discover patterns or themes, and group movies with similar characteristics. It also provides a foundation for building recommendation systems based on movie similarities.

2.2. Implementation of the Approach

To able to concatenate all the features into one column 'tags'. We write a function which converting every other features into list and then concatenate them.

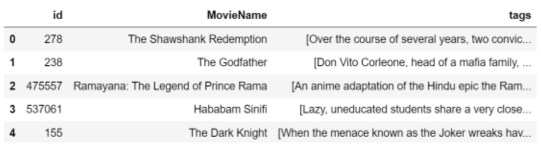
```
def convert_int_columns_to_string(movies, column_names):
    for column in column_names:
        movies[column] = movies[column].apply(lambda x: str(x))
    return movies
columns_to_convert =
['Watch_time','Year_of_Release','Meta_Score','Votes','Gross_Coll
ection(M$)']
convert_int_columns_to_string(movies, columns_to_convert)
```

```
def convert_string_columns_to_list(movies, column_names):
    for column in column_names:
        movies[column] = movies[column].apply(lambda x:
x.split(','))
    return movies

columns_to_convert = ['Genre',
'Directors','Stars','Watch_time','Year_of_Release','Meta_Score',
'Votes','Gross_Collection(M$)','Overviews']
convert_string_columns_to_list(movies, columns_to_convert)
```

After converting all the features into list then we combine them into one column 'tags'.

```
movies['tags'] =movies['Overviews'] + movies['Directors'] + movies['Genre']+ movies['Stars'] +
movies['Watch_time']+ movies['Year_of_Release']+ movies['Meta_Score']+
movies['Votes']+movies['Gross_Collection(M$)']
```

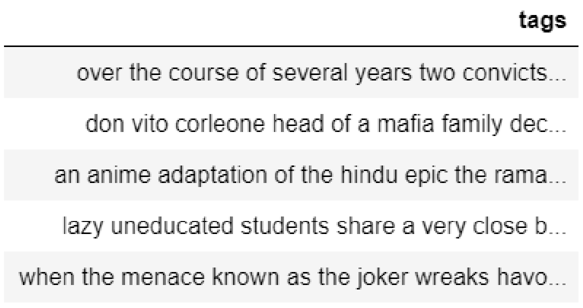
Here the result:



	id	MovieName	tags
0	278	The Shawshank Redemption	[Over the course of several years, two convic...
1	238	The Godfather	[Don Vito Corleone, head of a mafia family, ...
2	475557	Ramayana: The Legend of Prince Rama	[An anime adaptation of the Hindu epic the Ram...
3	537061	Hababam Sinifi	[Lazy, uneducated students share a very close...
4	155	The Dark Knight	[When the menace known as the Joker wreaks hav...

Nex, We want to convert 'tags' feature into string so we able to perform Bag-of-Word technique on it and also make the feature lower case because it can be beneficial in the context of text analysis and NLP.

```
new_df['tags'] = new_df['tags'].apply(lambda x:" ".join(x))
new_df['tags'] = new_df['tags'].apply(lambda x:x.lower())
```



tags
over the course of several years two convicts...
don vito corleone head of a mafia family dec...
an anime adaptation of the hindu epic the rama...
lazy uneducated students share a very close b...
when the menace known as the joker wreaks havo...

-Applying PorterStemmer on tags

```
import nltk
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
def stem(text):
    y = []

    for i in text.split():
        y.append(ps.stem(i))

    return " ".join(y)
new_df['tags'] = new_df['tags'].apply(stem)
new_df['tags'][1]
```

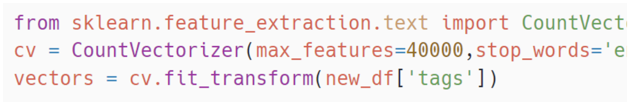
Here a sample after applied PorterStemmer:

“don vito corleon head of a mafia famili decid to hand over hi empir to hi youngest son michael. howev hi decis unintent put the live of hi love one in grave danger. franci ford coppola crime drama marlonbrando alpacino jamescaan dianekeaton 175 1972 100 1896661 134.97”

As you can see, some of the words in the document were converted into their base root.

-Performing Bag of Words Technique

CounterVectorizer


```
from sklearn.feature_extraction.text import CountVect
cv = CountVectorizer(max_features=40000,stop_words='e
vectors = cv.fit_transform(new_df['tags'])
```

If we print out the shape of ‘vectors’ variable the result should be (9497, 40000). The first number 9497 corresponds to the total number of movie samples in the dataset. Each movie is represented das a separate row. The second number, 40000, indicates the total number of unique words or features extracted from the ‘tags’ column. Each entry in the matrix represents the frequency count of a specific word.

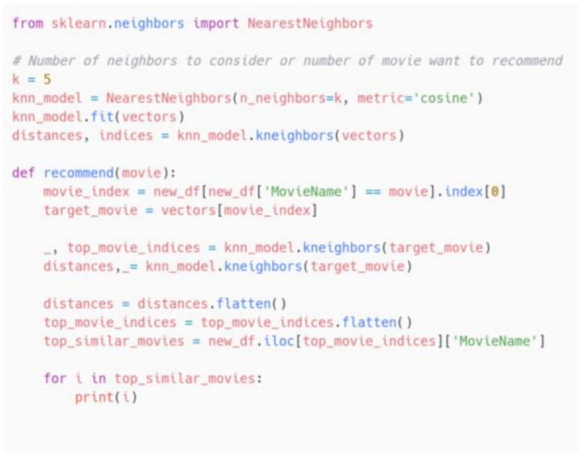
3.Model

Content-based filtering is a popular approach that leverages the characteristics of movies to make recommendations. In this context, the K-nearest neighbors (KNN) algorithm, using the cosine metric, is a powerful technique for content-based movie recommendation.

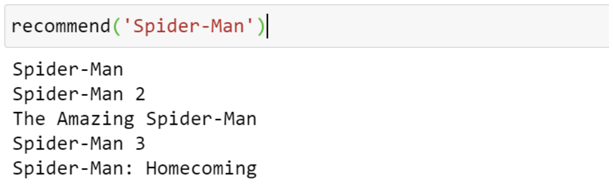
The KNN algorithm is a well-known technique in machine learning for classification and recommendation tasks. It run by measuring the distance or similarity between data points and identifying the k-nearest neighbors to make recommendation. So, in this project, KNN was applied to calculate the similarity between movies based on their feature vectors.
Cosine Metric is commonly used in content-based recommendation systems to measure the similarity between feature vectors. It calculates the cosine of the angle between two vectors.

3.1 Building the Model

By importing ‘NearestNeightbors ’ library from sklearn, we are able to to build KNN model easily and also we build a recommendation function for recommending movie.


```
from sklearn.neighbors import NearestNeighbors

# Number of neighbors to consider or number of movie want to recommend
k = 5
knn_model = NearestNeighbors(n_neighbors=k, metric='cosine')
knn_model.fit(vectors)
distances, indices = knn_model.kneighbors(vectors)

def recommend(movie):
    movie_index = new_df[new_df['MovieName'] == movie].index[0]
    target_movie = vectors[movie_index]

    _, top_movie_indices = knn_model.kneighbors(target_movie)
    distances,_= knn_model.kneighbors(target_movie)

    distances = distances.flatten()
    top_movie_indices = top_movie_indices.flatten()
    top_similar_movies = new_df.iloc[top_movie_indices]['MovieName']

    for i in top_similar_movies:
        print(i)
```

Using the recommendation function:


```
recommend('Spider-Man')
```
```
Spider-Man
Spider-Man 2
The Amazing Spider-Man
Spider-Man 3
Spider-Man: Homecoming
```

3.2 Evaluate the Model

For evaluating the model or the recommendation system, we haven’t found the right method to evaluate the model. So we decided to evaluate the model through doing the survey on the User.

User Surveys: Conduct surveys or questionnaires to gather feedback directly from users. Ask users about their satisfaction with the recommended movies.

```
recommend('The Avengers')
```

```
The Avengers
Avengers: Age of Ultron
Captain America: Civil War
This Island Earth
The Arrival
Captain America: The Winter Soldier
The Thing
Edge of Tomorrow
Rampage
X-Men Origins: Wolverine
```

```
recommend('Spider-Man')
```

```
Spider-Man
Spider-Man 2
The Amazing Spider-Man
Spider-Man 3
Spider-Man: Homecoming
Spider-Man: Far from Home
The Amazing Spider-Man 2
Spectral
Krrish
Jumper
```

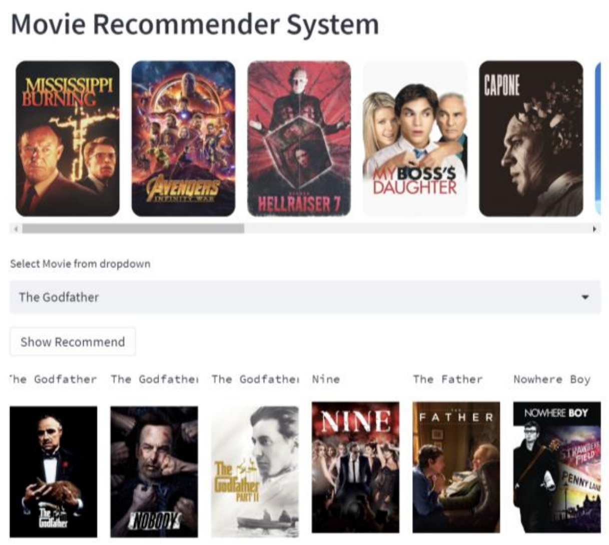
```
recommend('Thor')
```

```
Thor
Thor: The Dark World
Thor: Love and Thunder
Thor: Ragnarok
Hannibal
Bright
Power Rangers
Paranormal Activity
Miami Vice
Eternals
```

After the survey, the users seem to satisfy with the recommendation system as it recommend at least 2 movie which relevant or similar to their favorite movie. Even so, as you can see from the result above, the movie recommendation system seems like doing a great job.

4.Deployment of the Recommendation System

We deploy the movie recommendation system is using the Streamlit framework. The recommendation system use the power of content-based filtering and utilize the TMDB API to fetch movie posters and enhance the user experience.
Deployment with Streamlit:
Streamlit is a popular Python library that simplifies the process of building and deploying data-driven applications. It enables developers to create interactive web applications quickly and efficiently.
Here what the Web Application look like:


Movie Recommender System

Select Movie from dropdown

The Godfather

Show Recommend

he Godfather The Godfathe The Godfathe Nine The Father Nowhere Boy

5.Conclusion

In conclusion, the movie recommendation engine developed in this project uses content-based filtering and various features like Movie Star, Director, Overview, and Genre to enhance user experiences in entertainment.
The engine's ability to understand user preferences and interests through content-based filtering results in personalized movie suggestions. Incorporating Movie Star and Director information further refines the recommendations by considering the influence of specific actors and filmmakers on audience preferences.

The inclusion of Overview and Genre features enriches the recommendation process, presenting movies with similar themes and genres for users to explore.
Testing, user feedback, and comparisons against baseline models validate the engine's accuracy and relevance in delivering engaging movie recommendations.
While there is always room for improvement in feature engineering and model architectures, this project successfully demonstrates a powerful and effective movie recommendation engine that caters to individual tastes and broadens movie horizons. As technology advances, even more sophisticated recommendation engines promise an enjoyable movie-watching experience for users worldwide.

6.Reference

Wang, Z., Yu, X., Feng, N., & Wang, Z. (2014). An improved collaborative movie recommendation system using computational intelligence. Journal of Visual Languages and Computing, 25(6), 667–675. https://doi.org/10.1016/j.jvlc.2014.09.011

Manogaran, G., Logesh, R., Chandrashekhar, M., Challa, A., & Vijayakumar, V. (2017). A personalised movie recommendation system based on collaborative filtering. International Journal of High Performance Computing and Networking, 10(1/2), 54. https://doi.org/10.1504/ijhpcn.2017.083199

https://medium.com/web-mining-is688-spring-2021/content-based-movie-recommendation-system-72f122641eab

https://www.kaggle.com/code/ibtesama/getting-started-with-a-movie-recommendation-system

https://www.researchgate.net/publication/328745963_Content-Based_Movie_Recommendation_System_Using_Genre_Correlation_Proceedings_of_the_Second_International_Conference_on_SCI_2018_Volume_2

https://www.analyticsvidhya.com/blog/2020/08/recommendation-system-k-nearest-neighbors/

https://www.kaggle.com/code/heeraldedhia/movie-ratings-and-recommendation-using-knn