Movie Recommender System

Introduction to Data Mining Project Presentation

Group Members
Robert Doxey
Jordan Fuchs
Dylan McKenna
Brian Suriel
Aloysius Arno Wiputra

Introduction - Motivating The Audience

- Have you ever had a time where you were looking for new movies to watch but were unsure if you would like the movie you selected?
- Do you wish there were accurate recommender systems for movies that you can use to find new movies you will enjoy watching?
- Our project aims to help those who are looking for new movies to watch based on movies they have watched in the past.

Introduction - The Problem

- It can be hard to tell which movies you will enjoy watching without recommendations.
- People don’t want to waste time trying movies only to find out they don’t like the movies they are trying to watch.

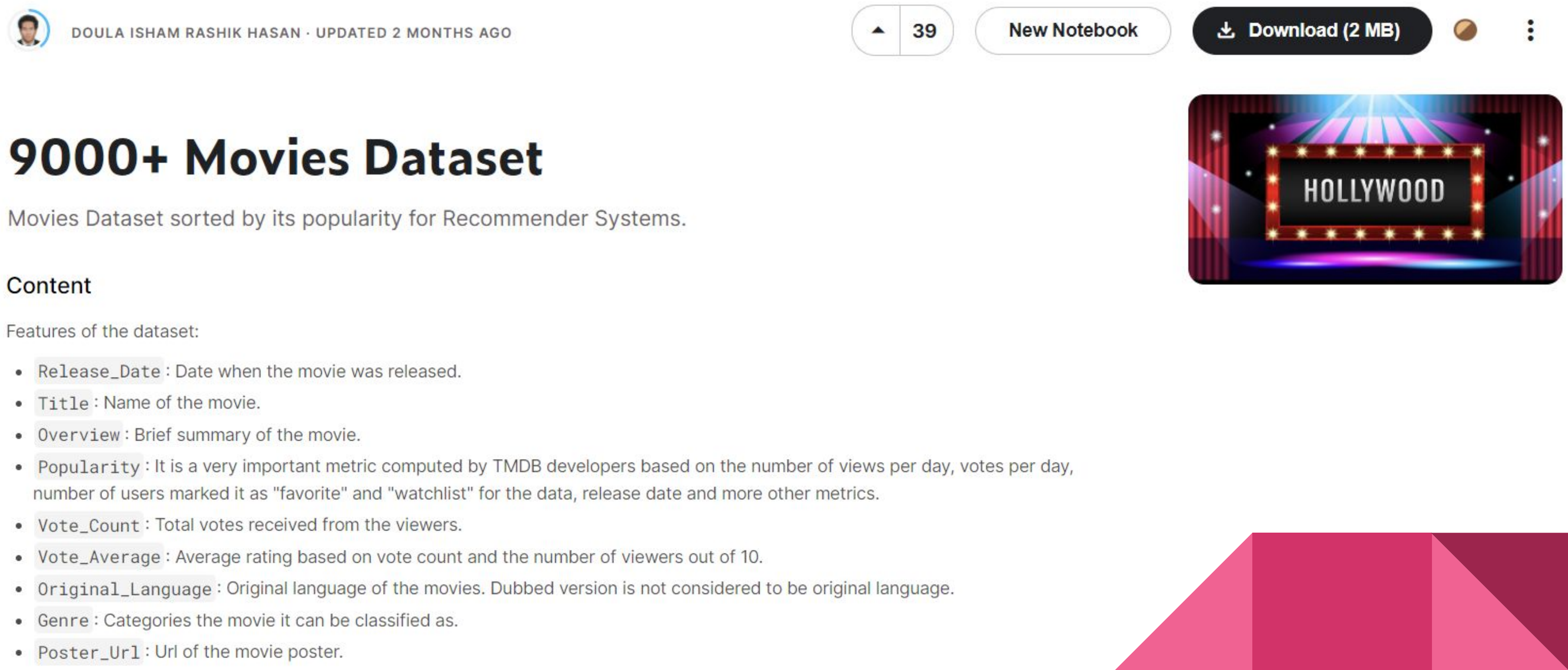
Formalization

- Our Chosen Data Set
  - 9000+ Movies
  - Features of the dataset: Release_Date, Title, Overview, Popularity, Vote_Count, Vote_Average, Original_Language, Genre, and Poster_Url
  - Data converted into a .csv file for easier analysis and pre-processing

- Function - Recommender System

OUR DATASET

the introduction from the kaggle source



9000+ Movies Dataset

Movies Dataset sorted by its popularity for Recommender Systems.

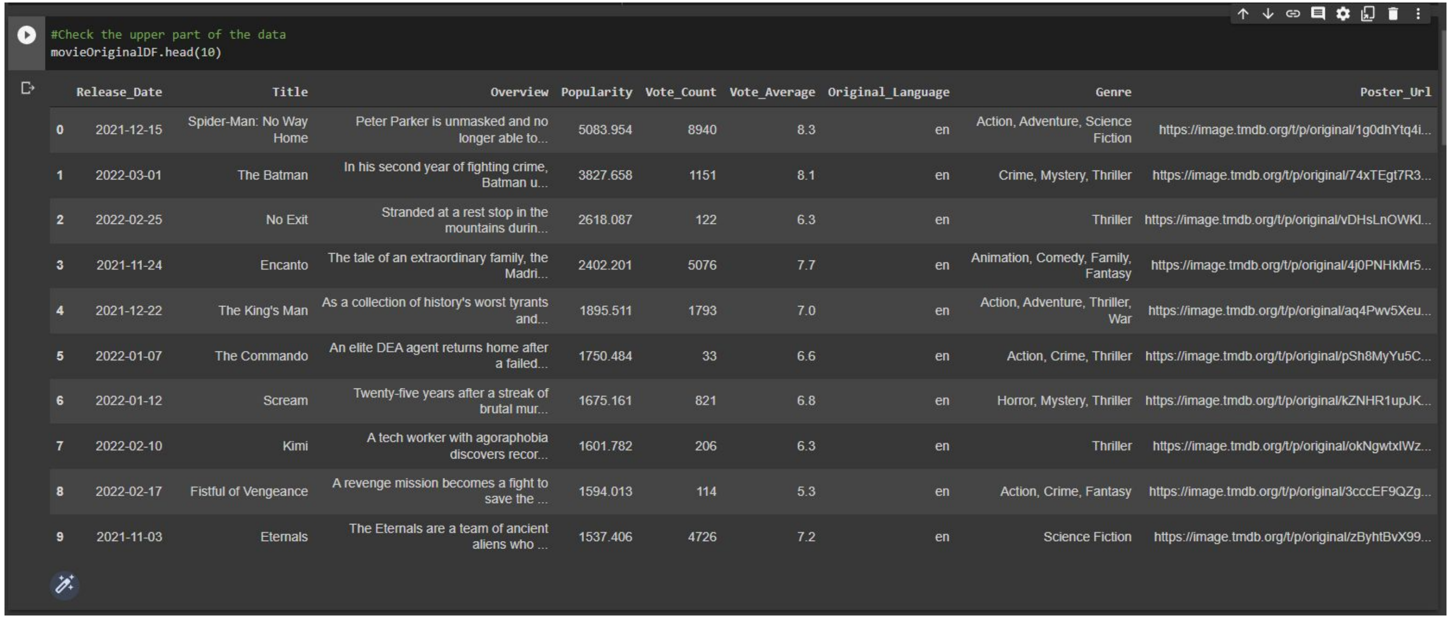
Content

Features of the dataset:

- `Release_Date` : Date when the movie was released.
- `Title` : Name of the movie.
- `Overview` : Brief summary of the movie.
- `Popularity` : It is a very important metric computed by TMDB developers based on the number of views per day, votes per day, number of users marked it as "favorite" and "watchlist" for the data, release date and more other metrics.
- `Vote_Count` : Total votes received from the viewers.
- `Vote_Average` : Average rating based on vote count and the number of viewers out of 10.
- `Original_Language` : Original language of the movies. Dubbed version is not considered to be original language.
- `Genre` : Categories the movie it can be classified as.
- `Poster_Url` : Url of the movie poster.

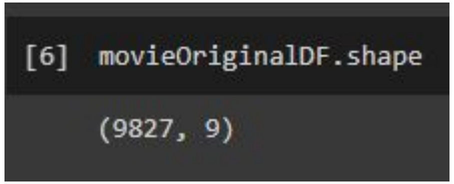
OUR DATASET

example entries and shape



```
#Check the upper part of the data
movieOriginalDF.head(10)
```

	Release_Date	Title	Overview	Popularity	Vote_Count	Vote_Average	Original_Language	Genre	Poster_Url
0	2021-12-15	Spider-Man: No Way Home	Peter Parker is unmasked and no longer able to...	5083.954	8940	8.3	en	Action, Adventure, Science Fiction	https://image.tmdb.org/t/p/original/1g0dhYtq4i...
1	2022-03-01	The Batman	In his second year of fighting crime, Batman u...	3827.658	1151	8.1	en	Crime, Mystery, Thriller	https://image.tmdb.org/t/p/original/74xTEgt7R3...
2	2022-02-25	No Exit	Stranded at a rest stop in the mountains durin...	2618.087	122	6.3	en	Thriller	https://image.tmdb.org/t/p/original/vDHsLnOWKl...
3	2021-11-24	Encanto	The tale of an extraordinary family, the Madri...	2402.201	5076	7.7	en	Animation, Comedy, Family, Fantasy	https://image.tmdb.org/t/p/original/4j0PNHkMr5...
4	2021-12-22	The King's Man	As a collection of history's worst tyrants and...	1895.511	1793	7.0	en	Action, Adventure, Thriller, War	https://image.tmdb.org/t/p/original/aq4Pwv5Xeu...
5	2022-01-07	The Commando	An elite DEA agent returns home after a failed...	1750.484	33	6.6	en	Action, Crime, Thriller	https://image.tmdb.org/t/p/original/pSh8MyYu5C...
6	2022-01-12	Scream	Twenty-five years after a streak of brutal mur...	1675.161	821	6.8	en	Horror, Mystery, Thriller	https://image.tmdb.org/t/p/original/kZNHR1upJK...
7	2022-02-10	Kimi	A tech worker with agoraphobia discovers recor...	1601.782	206	6.3	en	Thriller	https://image.tmdb.org/t/p/original/okNgwtxIWz...
8	2022-02-17	Fistful of Vengeance	A revenge mission becomes a fight to save the ...	1594.013	114	5.3	en	Action, Crime, Fantasy	https://image.tmdb.org/t/p/original/3cccEF9QZg...
9	2021-11-03	Eternals	The Eternals are a team of ancient aliens who ...	1537.406	4726	7.2	en	Science Fiction	https://image.tmdb.org/t/p/original/zByhtBvX99...

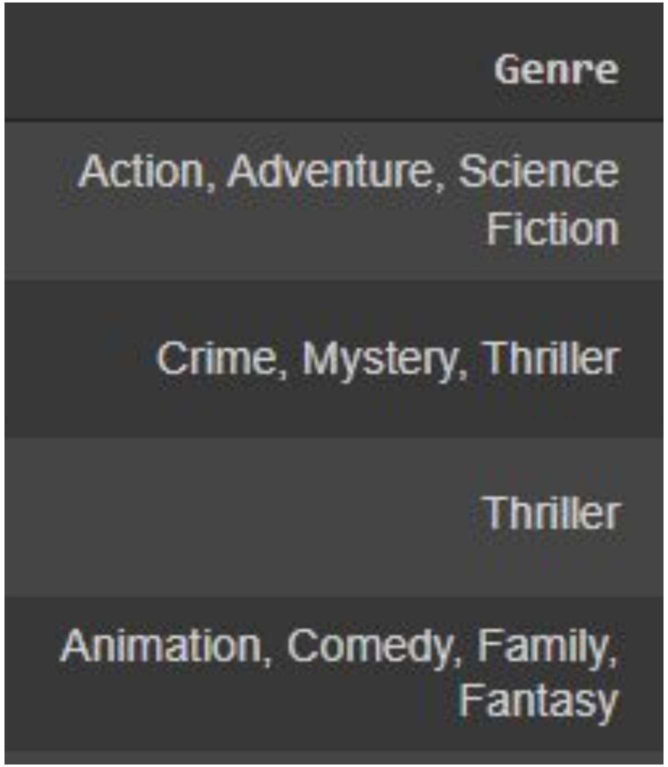


```
[6] movieOriginalDF.shape

    (9827, 9)
```

9827 movies
9 categories

OUR DATASET

the problems with the genre

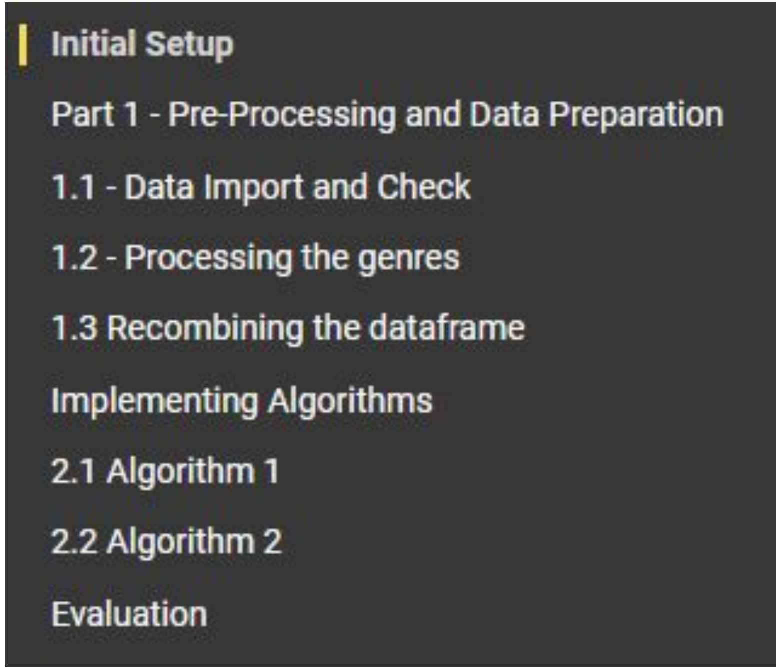


Genre
Action, Adventure, Science Fiction
Crime, Mystery, Thriller
Thriller
Animation, Comedy, Family, Fantasy

- different genres combined into **one value** per entry in one column
- needs to be **processed** so that it can be used by the algorithm

OUR APPROACH

the notebook itself



Initial Setup

Part 1 - Pre-Processing and Data Preparation

1.1 - Data Import and Check

1.2 - Processing the genres

1.3 Recombining the dataframe

Implementing Algorithms

2.1 Algorithm 1

2.2 Algorithm 2

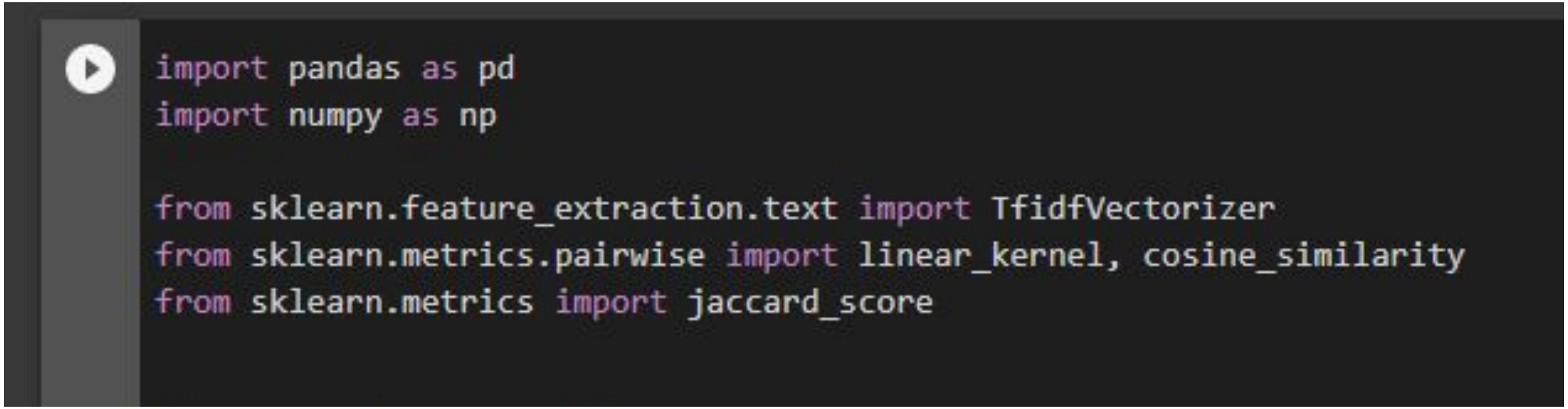
Evaluation

segmented into **parts**

- better organization
- allows smoother workflow

OUR APPROACH

the libraries that we used

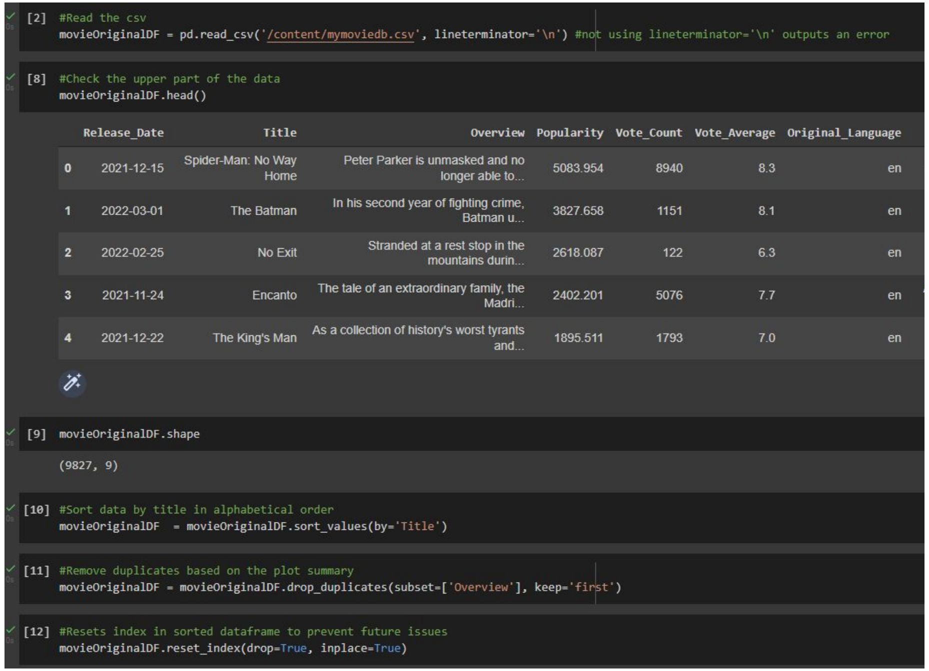


```
import pandas as pd
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
from sklearn.metrics import jaccard_score
```

- **Pandas** and **Numpy** for **data analysis** and **processing**
- **sklearn** for the basis of our algorithms

OUR APPROACH

pre-processing - converting into a DF and doing some groundwork



```
[2]  #Read the csv
     movieOriginalDF = pd.read_csv('/content/mymoviedb.csv', lineterminator='\n') #not using lineterminator='\n' outputs an error
```

```
[8]  #Check the upper part of the data
     movieOriginalDF.head()
```

	Release_Date	Title	Overview	Popularity	Vote_Count	Vote_Average	Original_Language
0	2021-12-15	Spider-Man: No Way Home	Peter Parker is unmasked and no longer able to...	5083.954	8940	8.3	en
1	2022-03-01	The Batman	In his second year of fighting crime, Batman u...	3827.658	1151	8.1	en
2	2022-02-25	No Exit	Stranded at a rest stop in the mountains durin...	2618.087	122	6.3	en
3	2021-11-24	Encanto	The tale of an extraordinary family, the Madri...	2402.201	5076	7.7	en
4	2021-12-22	The King's Man	As a collection of history's worst tyrants and...	1895.511	1793	7.0	en

```
[9]  movieOriginalDF.shape

     (9827, 9)
```

```
[10] #Sort data by title in alphabetical order
     movieOriginalDF  = movieOriginalDF.sort_values(by='Title')
```

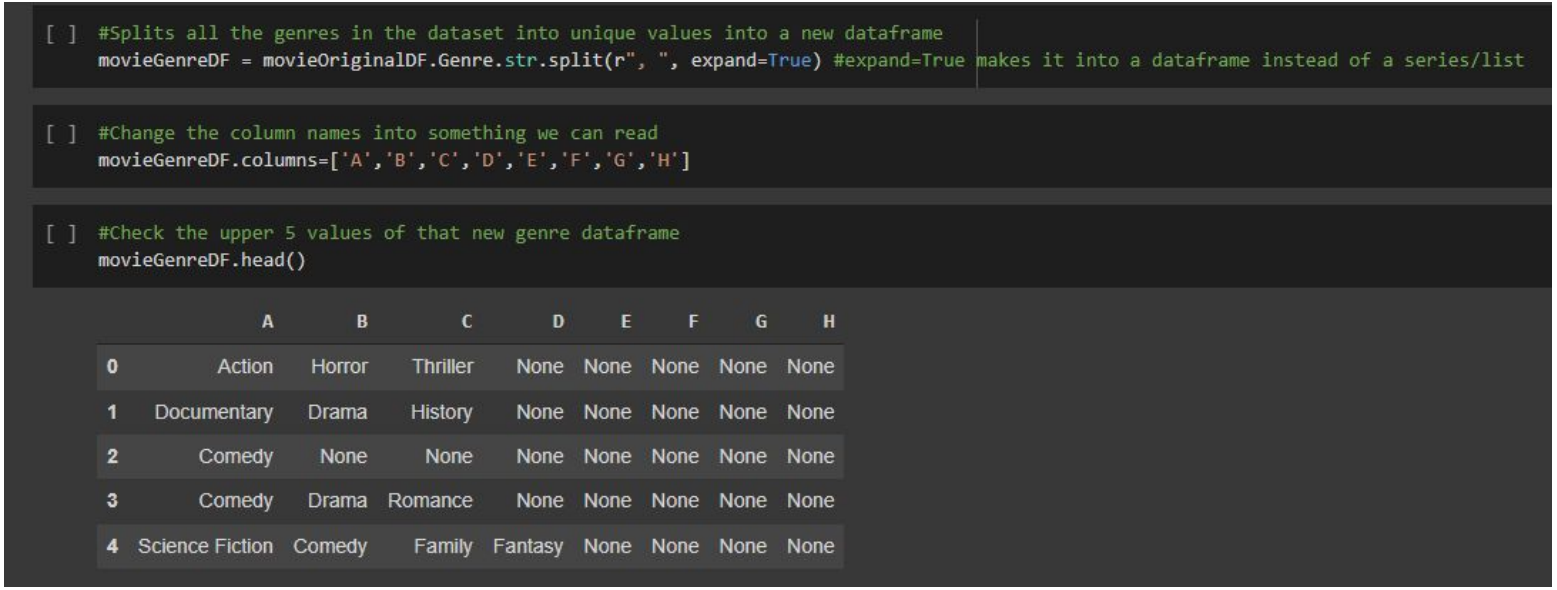
```
[11] #Remove duplicates based on the plot summary
     movieOriginalDF = movieOriginalDF.drop_duplicates(subset=['Overview'], keep='first')
```

```
[12] #Resets index in sorted dataframe to prevent future issues
     movieOriginalDF.reset_index(drop=True, inplace=True)
```

- reading csv into a **dataframe**
  - for easier manipulation
- **sorting** by title
- dropping **duplicates**
  - uses **'Overview'** because there are movies with the same titles
- **resetting** the index
  - index restarts at 0 by title
- processed dataframe gets **saved to a new csv**
  - ensures we can **continue working** without having to run everything from the start

OUR APPROACH

the genre problem – how we processed it



```
[ ] #Splits all the genres in the dataset into unique values into a new dataframe
    movieGenreDF = movieOriginalDF.Genre.str.split(r", ", expand=True) #expand=True makes it into a dataframe instead of a series/list
```

```
[ ] #Change the column names into something we can read
    movieGenreDF.columns=['A','B','C','D','E','F','G','H']
```

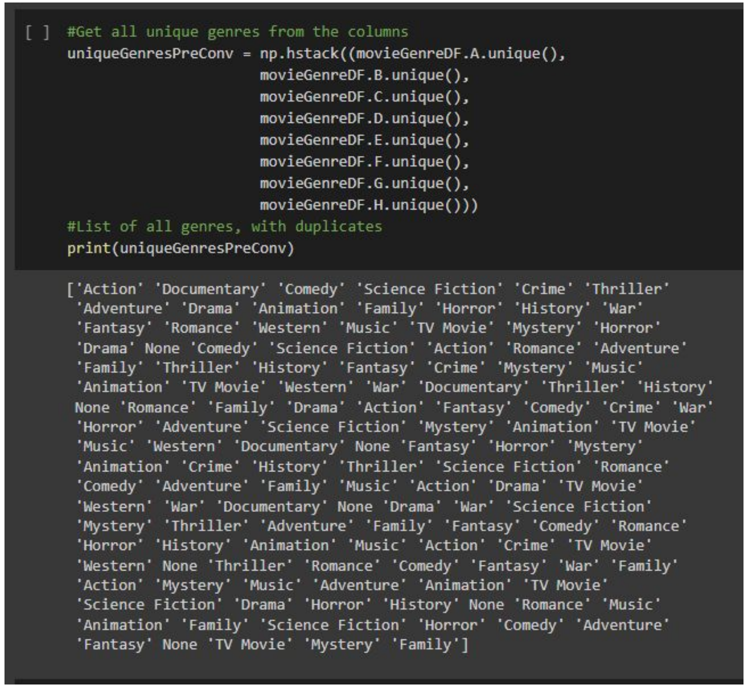
```
[ ] #Check the upper 5 values of that new genre dataframe
    movieGenreDF.head()
```

	A	B	C	D	E	F	G	H
0	Action	Horror	Thriller	None	None	None	None	None
1	Documentary	Drama	History	None	None	None	None	None
2	Comedy	None	None	None	None	None	None	None
3	Comedy	Drama	Romance	None	None	None	None	None
4	Science Fiction	Comedy	Family	Fantasy	None	None	None	None

- making a **new dataframe** with **just the genre**
  - the genre is split into **different columns**
- change the column titles into something easier to be referenced
- index is kept so the values should match

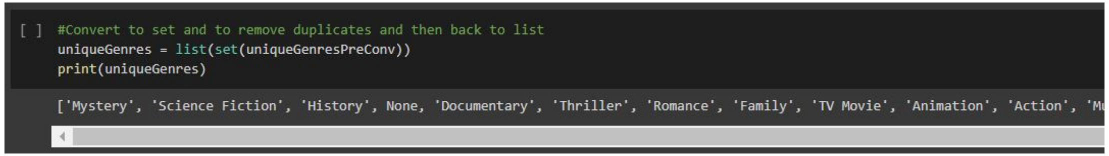
OUR APPROACH

the genre problem – how we processed it



```
[ ] #Get all unique genres from the columns
    uniqueGenresPreConv = np.hstack((movieGenreDF.A.unique(),
                                     movieGenreDF.B.unique(),
                                     movieGenreDF.C.unique(),
                                     movieGenreDF.D.unique(),
                                     movieGenreDF.E.unique(),
                                     movieGenreDF.F.unique(),
                                     movieGenreDF.G.unique(),
                                     movieGenreDF.H.unique()))
    #List of all genres, with duplicates
    print(uniqueGenresPreConv)
```

```
['Action' 'Documentary' 'Comedy' 'Science Fiction' 'Crime' 'Thriller'
 'Adventure' 'Drama' 'Animation' 'Family' 'Horror' 'History' 'War'
 'Fantasy' 'Romance' 'Western' 'Music' 'TV Movie' 'Mystery' 'Horror'
 'Drama' None 'Comedy' 'Science Fiction' 'Action' 'Romance' 'Adventure'
 'Family' 'Thriller' 'History' 'Fantasy' 'Crime' 'Mystery' 'Music'
 'Animation' 'TV Movie' 'Western' 'War' 'Documentary' 'Thriller' 'History'
 None 'Romance' 'Family' 'Drama' 'Action' 'Fantasy' 'Comedy' 'Crime' 'War'
 'Horror' 'Adventure' 'Science Fiction' 'Mystery' 'Animation' 'TV Movie'
 'Music' 'Western' 'Documentary' None 'Fantasy' 'Horror' 'Mystery'
 'Animation' 'Crime' 'History' 'Thriller' 'Science Fiction' 'Romance'
 'Comedy' 'Adventure' 'Family' 'Music' 'Action' 'Drama' 'TV Movie'
 'Western' 'War' 'Documentary' None 'Drama' 'War' 'Science Fiction'
 'Mystery' 'Thriller' 'Adventure' 'Family' 'Fantasy' 'Comedy' 'Romance'
 'Horror' 'History' 'Animation' 'Music' 'Action' 'Crime' 'TV Movie'
 'Western' None 'Thriller' 'Romance' 'Comedy' 'Fantasy' 'War' 'Family'
 'Action' 'Mystery' 'Music' 'Adventure' 'Animation' 'TV Movie'
 'Science Fiction' 'Drama' 'Horror' 'History' None 'Romance' 'Music'
 'Animation' 'Family' 'Science Fiction' 'Horror' 'Comedy' 'Adventure'
 'Fantasy' None 'TV Movie' 'Mystery' 'Family']
```



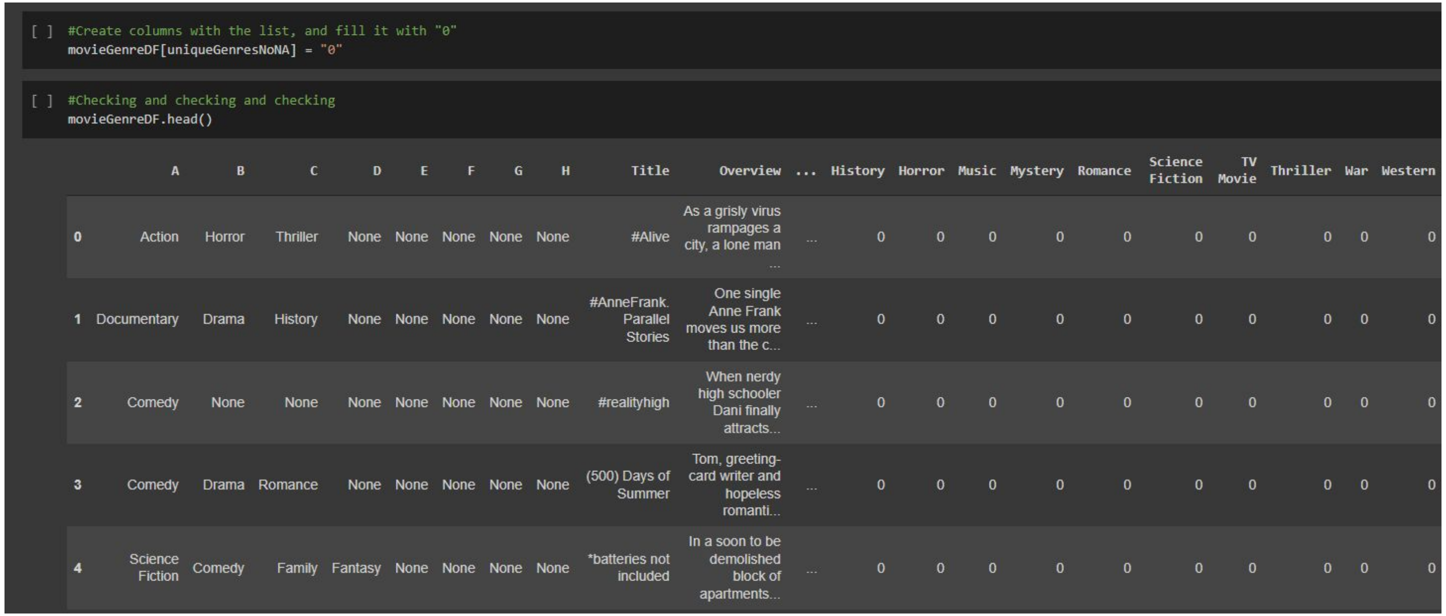
```
[ ] #Convert to set and to remove duplicates and then back to list
    uniqueGenres = list(set(uniqueGenresPreConv))
    print(uniqueGenres)
```

```
['Mystery', 'Science Fiction', 'History', None, 'Documentary', 'Thriller', 'Romance', 'Family', 'TV Movie', 'Animation', 'Action', 'Mu
```

- Use **numpy** to **stack arrays into a tuple** with the **unique value from each column**
- Convert it back to **set** and then **list**
  - **Set does not allow duplicates**, entries with duplicate value are purged in the process
  - Convert to list to be sent back to the dataframe

OUR APPROACH

the genre problem – how we processed it



```
[ ] #Create columns with the list, and fill it with "0"
    movieGenreDF[uniqueGenresNoNA] = "0"
```

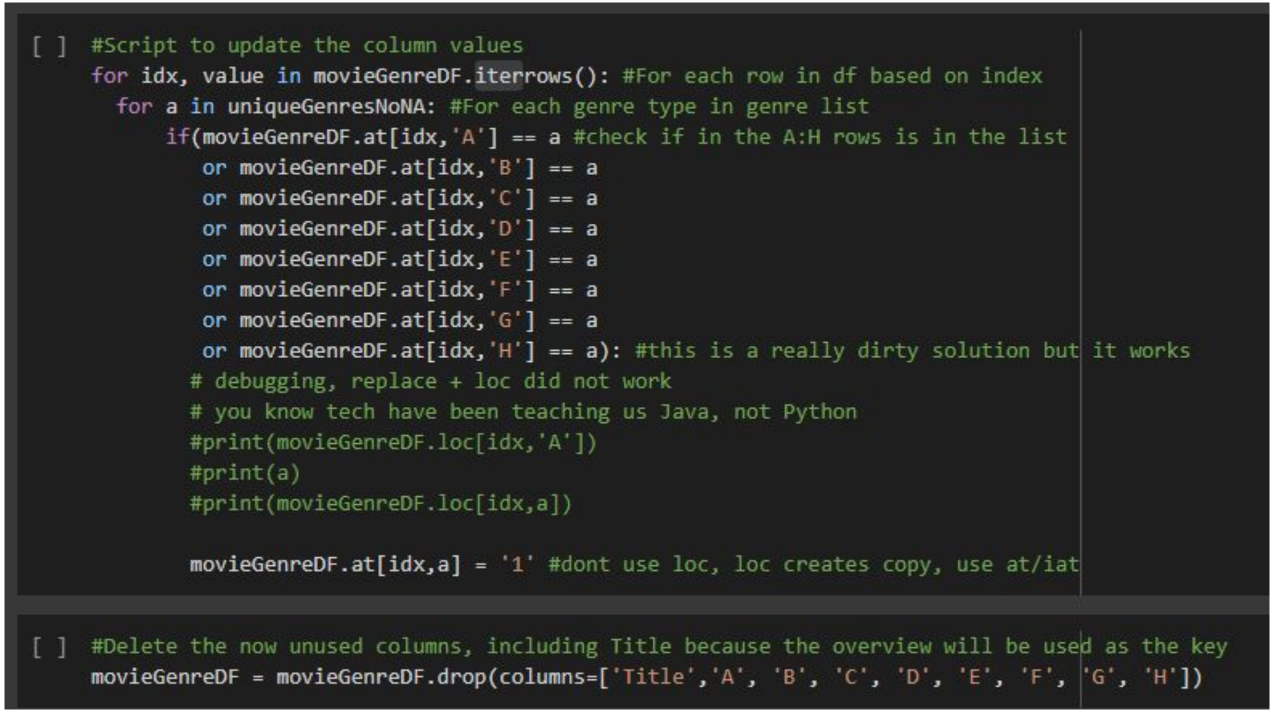
```
[ ] #Checking and checking and checking
    movieGenreDF.head()
```

	A	B	C	D	E	F	G	H	Title	Overview	...	History	Horror	Music	Mystery	Romance	Science Fiction	TV Movie	Thriller	War	Western
0	Action	Horror	Thriller	None	None	None	None	None	#Alive	As a grisly virus rampages a city, a lone man ...	...	0	0	0	0	0	0	0	0	0	0
1	Documentary	Drama	History	None	None	None	None	None	#AnneFrank. Parallel Stories	One single Anne Frank moves us more than the c...	...	0	0	0	0	0	0	0	0	0	0
2	Comedy	None	None	None	None	None	None	None	#realityhigh	When nerdy high schooler Dani finally attracts...	...	0	0	0	0	0	0	0	0	0	0
3	Comedy	Drama	Romance	None	None	None	None	None	(500) Days of Summer	Tom, greeting-card writer and hopeless romanti...	...	0	0	0	0	0	0	0	0	0	0
4	Science Fiction	Comedy	Family	Fantasy	None	None	None	None	*batteries not included	In a soon to be demolished block of apartments...	...	0	0	0	0	0	0	0	0	0	0

make **new columns** in the dataframe based on the list and **fill with zero**

OUR APPROACH

the genre problem – how we processed it



```
[ ]  #Script to update the column values
     for idx, value in movieGenreDF.iterrows(): #For each row in df based on index
       for a in uniqueGenresNoNA: #For each genre type in genre list
           if(movieGenreDF.at[idx,'A'] == a #check if in the A:H rows is in the list
              or movieGenreDF.at[idx,'B'] == a
              or movieGenreDF.at[idx,'C'] == a
              or movieGenreDF.at[idx,'D'] == a
              or movieGenreDF.at[idx,'E'] == a
              or movieGenreDF.at[idx,'F'] == a
              or movieGenreDF.at[idx,'G'] == a
              or movieGenreDF.at[idx,'H'] == a): #this is a really dirty solution but it works
             # debugging, replace + loc did not work
             # you know tech have been teaching us Java, not Python
             #print(movieGenreDF.loc[idx,'A'])
             #print(a)
             #print(movieGenreDF.loc[idx,a])

             movieGenreDF.at[idx,a] = '1' #dont use loc, loc creates copy, use at/iat
```

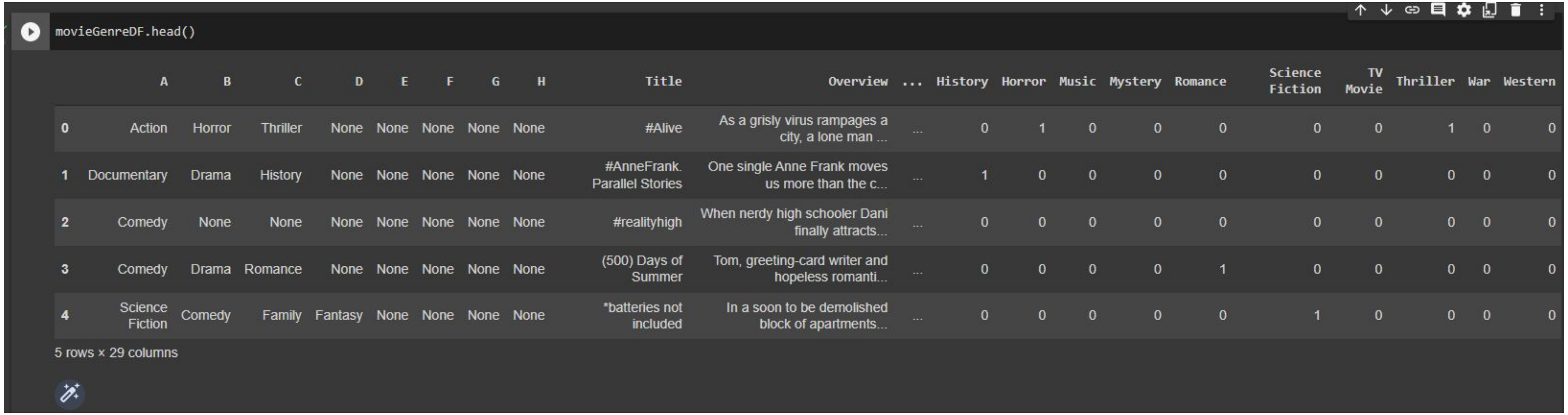
```
[ ]  #Delete the now unused columns, including Title because the overview will be used as the key
     movieGenreDF = movieGenreDF.drop(columns=['Title','A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'])
```

iterate through the dataframe

- if **columns A:H** at each row **has a value matching the list**, **update the corresponding column** with ‘1’

OUR APPROACH

the genre problem – how we processed it



```
movieGenreDF.head()
```

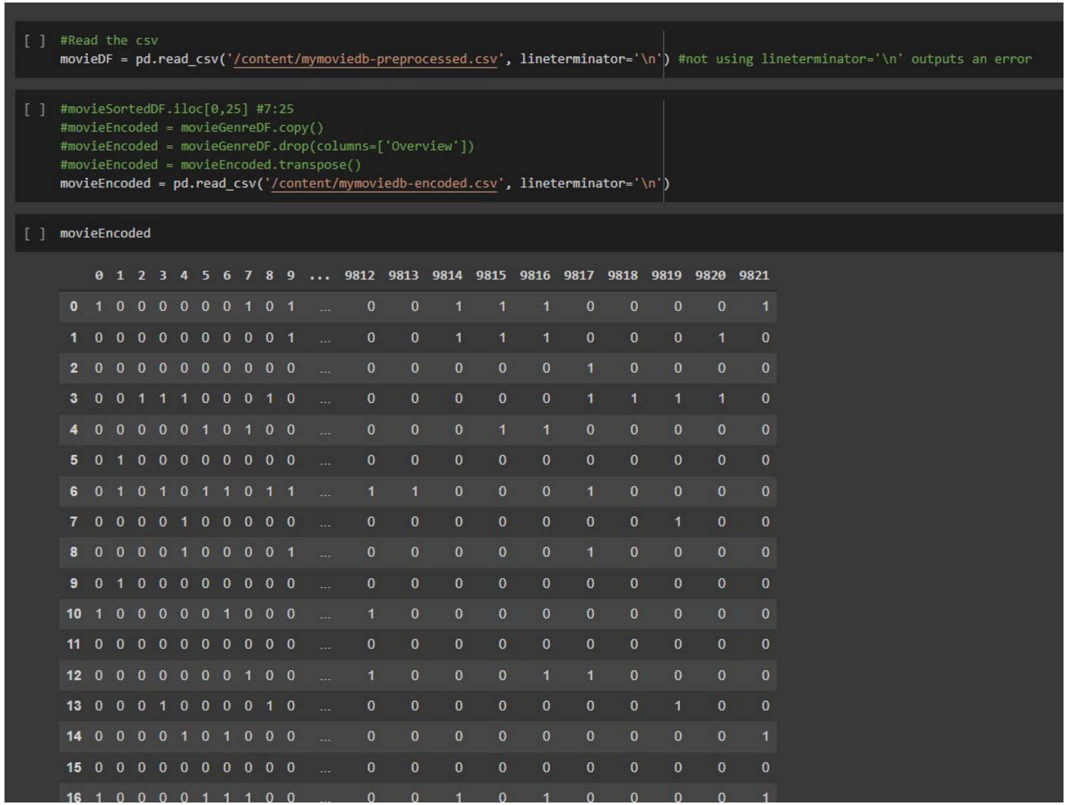
	A	B	C	D	E	F	G	H	Title	Overview	...	History	Horror	Music	Mystery	Romance	Science Fiction	TV Movie	Thriller	War	Western
0	Action	Horror	Thriller	None	None	None	None	None	#Alive	As a grisly virus rampages a city, a lone man ...	...	0	1	0	0	0	0	0	1	0	0
1	Documentary	Drama	History	None	None	None	None	None	#AnneFrank. Parallel Stories	One single Anne Frank moves us more than the c...	...	1	0	0	0	0	0	0	0	0	0
2	Comedy	None	None	None	None	None	None	None	#realityhigh	When nerdy high schooler Dani finally attracts...	...	0	0	0	0	0	0	0	0	0	0
3	Comedy	Drama	Romance	None	None	None	None	None	(500) Days of Summer	Tom, greeting-card writer and hopeless romanti...	...	0	0	0	0	1	0	0	0	0	0
4	Science Fiction	Comedy	Family	Fantasy	None	None	None	None	*batteries not included	In a soon to be demolished block of apartments...	...	0	0	0	0	0	1	0	0	0	0

5 rows × 29 columns

- sample showing that the script works

THE ALGORITHM USED

the first one – using Jaccard Similarity



```
[ ] #Read the csv
    movieDF = pd.read_csv('/content/mymoviedb-preprocessed.csv', lineterminator='\n') #not using lineterminator='\n' outputs an error
```

```
[ ] #movieSortedDF.iloc[0,25] #7:25
    #movieEncoded = movieGenreDF.copy()
    #movieEncoded = movieGenreDF.drop(columns=['Overview'])
    #movieEncoded = movieEncoded.transpose()
    movieEncoded = pd.read_csv('/content/mymoviedb-encoded.csv', lineterminator='\n')
```

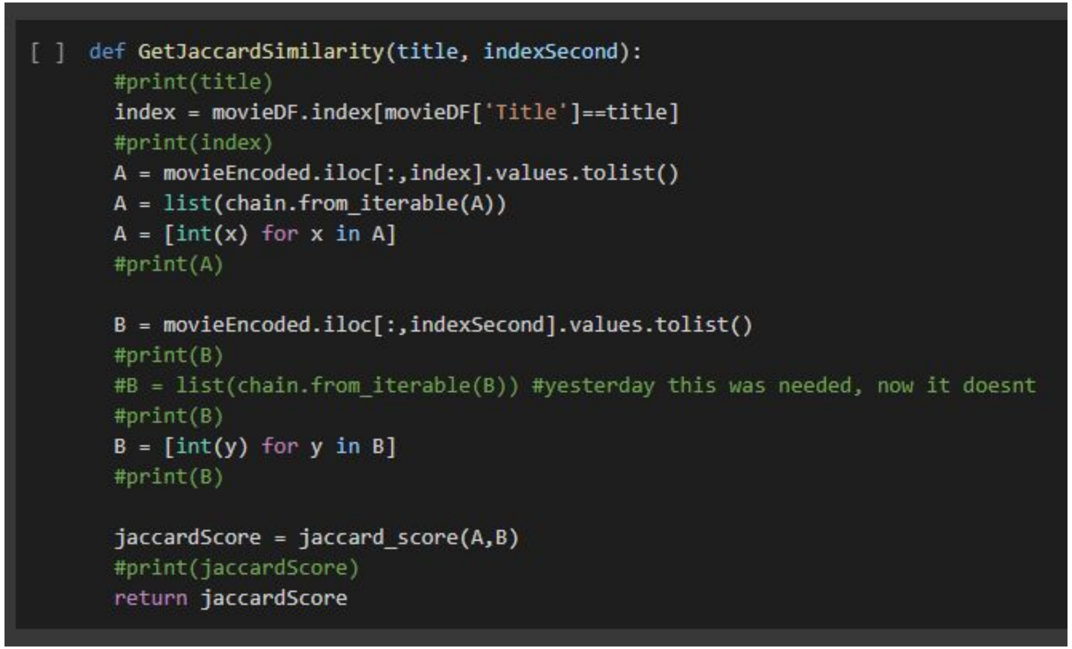
```
[ ] movieEncoded
```

	0	1	2	3	4	5	6	7	8	9	...	9812	9813	9814	9815	9816	9817	9818	9819	9820	9821
0	1	0	0	0	0	0	0	1	0	1	...	0	0	1	1	1	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	1	...	0	0	1	1	1	0	0	0	1	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0
3	0	0	1	1	1	0	0	0	1	0	...	0	0	0	0	0	1	1	1	1	0
4	0	0	0	0	0	1	0	1	0	0	...	0	0	0	1	1	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
6	0	1	0	1	0	1	1	0	1	1	...	1	1	0	0	0	1	0	0	0	0
7	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	1	0	0	0	0	1	...	0	0	0	0	0	1	0	0	0	0
9	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	1	0	0	0	...	1	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	...	1	0	0	0	1	1	0	0	0	0
13	0	0	0	1	0	0	0	0	1	0	...	0	0	0	0	0	0	0	1	0	0
14	0	0	0	0	1	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	1
15	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
16	1	0	0	0	0	1	1	1	0	0	...	0	0	1	0	1	0	0	0	0	1

- copy that genre dataframe
- drop the unused keys
- transpose it

THE ALGORITHM USED

the first one – using Jaccard Similarity

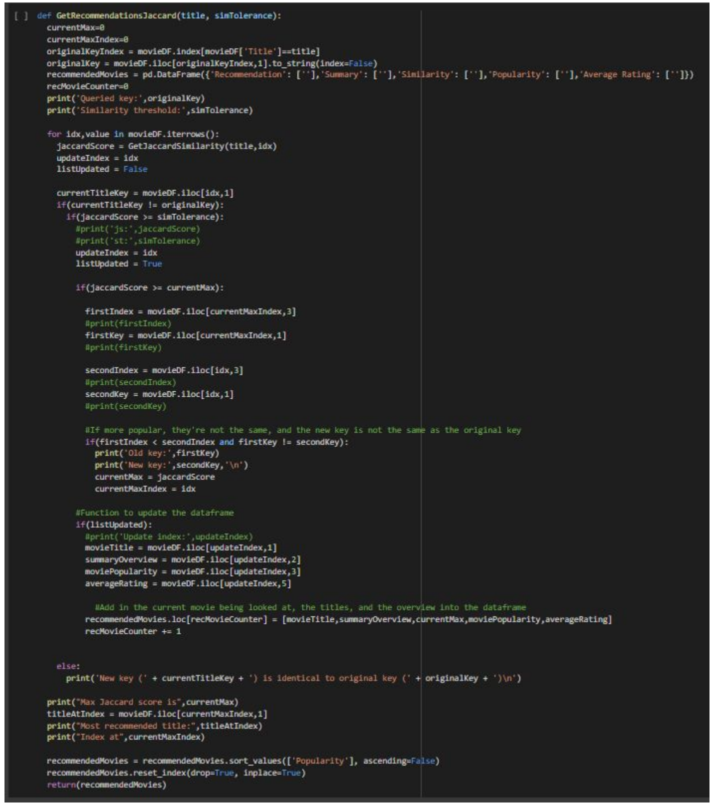


```
[ ]  def GetJaccardSimilarity(title, indexSecond):
       #print(title)
       index = movieDF.index[movieDF['Title']==title]
       #print(index)
       A = movieEncoded.iloc[:,index].values.tolist()
       A = list(chain.from_iterable(A))
       A = [int(x) for x in A]
       #print(A)

       B = movieEncoded.iloc[:,indexSecond].values.tolist()
       #print(B)
       #B = list(chain.from_iterable(B)) #yesterday this was needed, now it doesnt
       #print(B)
       B = [int(y) for y in B]
       #print(B)

       jaccardScore = jaccard_score(A,B)
       #print(jaccardScore)
       return jaccardScore
```

- Script to take two arguments, the title and index
- Run through the encoded dataframe and database
- Calculate their similarity, assign a score

THE ALGORITHM USED

the first one – using Jaccard Similarity

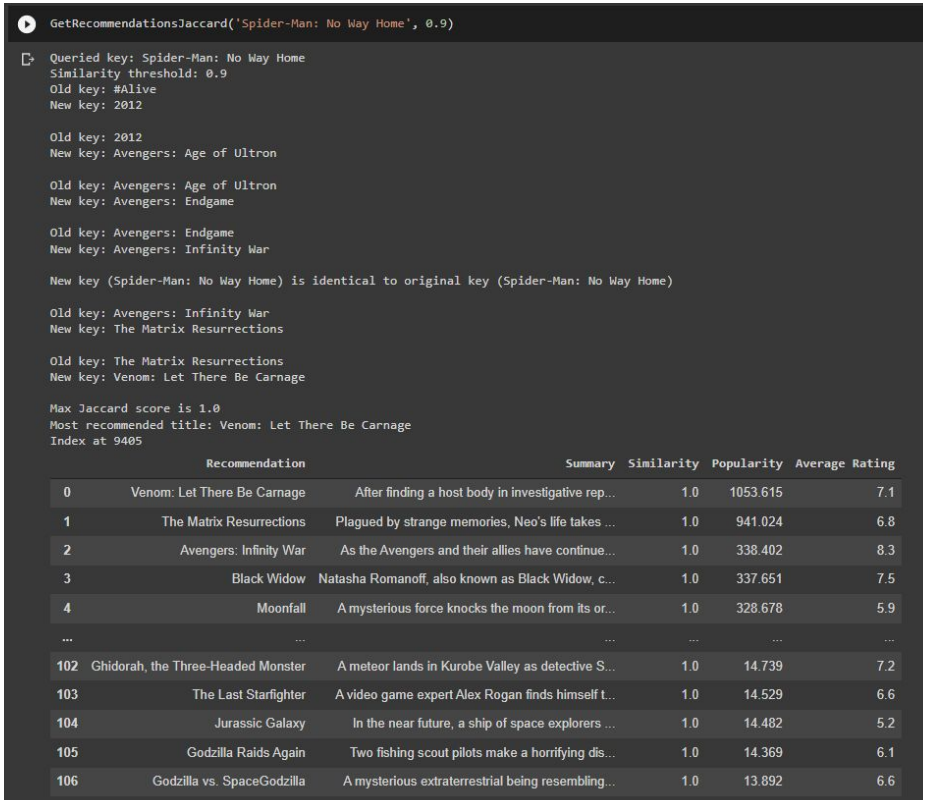


```
[ ] def GetRecommendationsJaccard(title, simTolerance):
      currentMax=0
      currentMaxIndex=0
      originalKeyIndex = movieDF.index[movieDF['Title']==title]
      originalKey = movieDF.iloc[originalKeyIndex,1].to_string(index=False)
      recommendedMovies = pd.DataFrame({'Recommendation': [''],'Summary': [''],'Similarity': [''],'Popularity': [''],'Average Rating': ['']})
      recMovieCounter=0
      print('Queried key:',originalKey)
      print('Similarity threshold:',simTolerance)

      for idx,value in movieDF.iterrows():
        jaccardScore = GetJaccardSimilarity(title,idx)
        updateIndex = idx
        listUpdated = False

        currentTitleKey = movieDF.iloc[idx,1]
        if(currentTitleKey != originalKey):
          if(jaccardScore >= simTolerance):
            #print('js:',jaccardScore)
            #print('st:',simTolerance)
            updateIndex = idx
            listUpdated = True

            if(jaccardScore >= currentMax):

              firstIndex = movieDF.iloc[currentMaxIndex,3]
              #print(firstIndex)
              firstKey = movieDF.iloc[currentMaxIndex,1]
              #print(firstKey)

              secondIndex = movieDF.iloc[idx,3]
              #print(secondIndex)
              secondKey = movieDF.iloc[idx,1]
              #print(secondKey)

              #If more popular, they're not the same, and the new key is not the same as the original key
              if(firstIndex < secondIndex and firstKey != secondKey):
                print('Old key:',firstKey)
                print('New key:',secondKey,'\n')
                currentMax = jaccardScore
                currentMaxIndex = idx

            #Function to update the dataframe
            if(listUpdated):
              #print('Update index:',updateIndex)
              movieTitle = movieDF.iloc[updateIndex,1]
              summaryOverview = movieDF.iloc[updateIndex,2]
              moviePopularity = movieDF.iloc[updateIndex,3]
              averageRating = movieDF.iloc[updateIndex,5]

                #Add in the current movie being looked at, the titles, and the overview into the dataframe
              recommendedMovies.loc[recMovieCounter] = [movieTitle,summaryOverview,currentMax,moviePopularity,averageRating]
              recMovieCounter += 1


        else:
          print('New key (' + currentTitleKey + ') is identical to original key (' + originalKey + ')\n')

      print("Max Jaccard score is",currentMax)
      titleAtIndex = movieDF.iloc[currentMaxIndex,1]
      print("Most recommended title:",titleAtIndex)
      print("Index at",currentMaxIndex)

      recommendedMovies = recommendedMovies.sort_values(['Popularity'], ascending=False)
      recommendedMovies.reset_index(drop=True, inplace=True)
      return(recommendedMovies)
```

- TL;DR
  - Takes **two arguments**, the **title to be searched** and **similarity tolerance**
  - Runs through the dataframe, **passes the title and the current index** to the **instantiated method** from the previous slide
  - Keeps track of **highest similarity**, if the same has been reached then refers to the **popularity score**
  - If there is a **new entry** passing the threshold, **updates the recommendation dataframe**

THE ALGORITHM USED

the first one – using Jaccard Similarity



```
GetRecommendationsJaccard('Spider-Man: No Way Home', 0.9)
```

```
Queried key: Spider-Man: No Way Home
Similarity threshold: 0.9
Old key: #Alive
New key: 2012

Old key: 2012
New key: Avengers: Age of Ultron

Old key: Avengers: Age of Ultron
New key: Avengers: Endgame

Old key: Avengers: Endgame
New key: Avengers: Infinity War

New key (Spider-Man: No Way Home) is identical to original key (Spider-Man: No Way Home)

Old key: Avengers: Infinity War
New key: The Matrix Resurrections

Old key: The Matrix Resurrections
New key: Venom: Let There Be Carnage

Max Jaccard score is 1.0
Most recommended title: Venom: Let There Be Carnage
Index at 9405
```

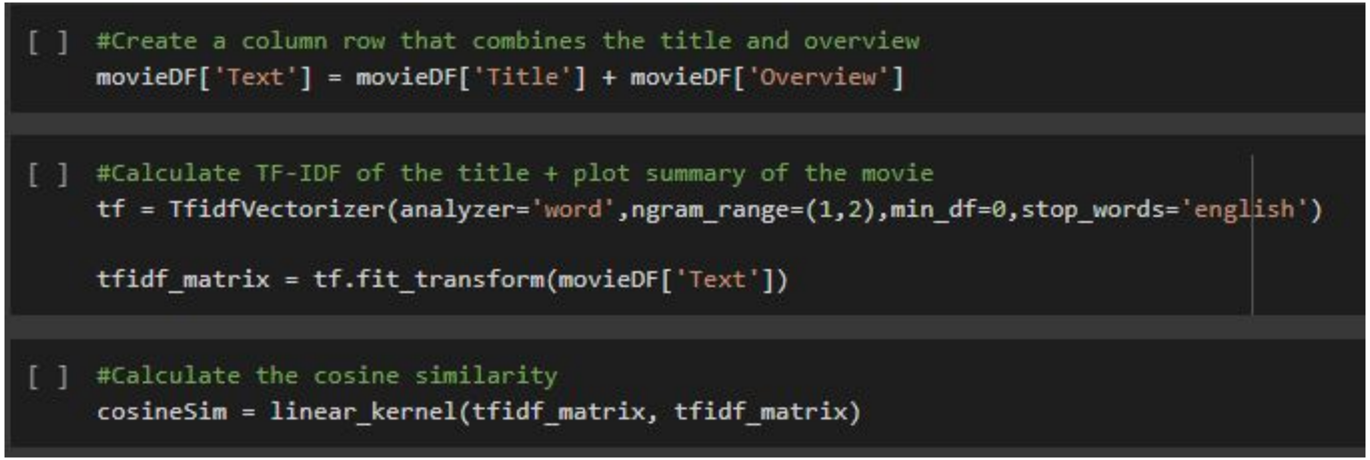
	Recommendation	Summary	Similarity	Popularity	Average Rating
0	Venom: Let There Be Carnage	After finding a host body in investigative rep...	1.0	1053.615	7.1
1	The Matrix Resurrections	Plagued by strange memories, Neo's life takes ...	1.0	941.024	6.8
2	Avengers: Infinity War	As the Avengers and their allies have continue...	1.0	338.402	8.3
3	Black Widow	Natasha Romanoff, also known as Black Widow, c...	1.0	337.651	7.5
4	Moonfall	A mysterious force knocks the moon from its or...	1.0	328.678	5.9
...	...	...	...	...	...
102	Ghidorah, the Three-Headed Monster	A meteor lands in Kurobe Valley as detective S...	1.0	14.739	7.2
103	The Last Starfighter	A video game expert Alex Rogan finds himself t...	1.0	14.529	6.6
104	Jurassic Galaxy	In the near future, a ship of space explorers ...	1.0	14.482	5.2
105	Godzilla Raids Again	Two fishing scout pilots make a horrifying dis...	1.0	14.369	6.1
106	Godzilla vs. SpaceGodzilla	A mysterious extraterrestrial being resembling...	1.0	13.892	6.6

Sample result

- Shows when the highest is updated
- The movie finally recommended, its index location and similarity score
- Shows the dataframe with the movies with high similarity and sorted by popularity, with its title and plot summary

THE ALGORITHM USED

the second one – using TF/IDF and Cosine Similarity

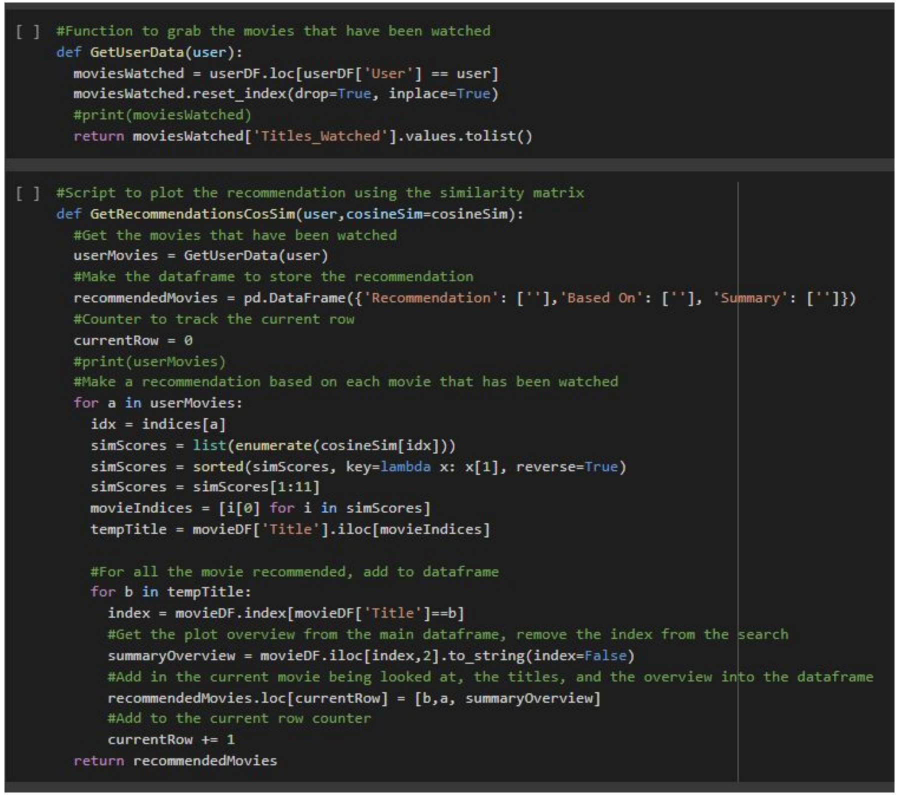


```
[ ]  #Create a column row that combines the title and overview
     movieDF['Text'] = movieDF['Title'] + movieDF['Overview']

[ ]  #Calculate TF-IDF of the title + plot summary of the movie
     tf = TfidfVectorizer(analyzer='word',ngram_range=(1,2),min_df=0,stop_words='english')

     tfidf_matrix = tf.fit_transform(movieDF['Text'])

[ ]  #Calculate the cosine similarity
     cosineSim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

- Combines the ‘Title’ and ‘Overview’ into a new column to be processed
- Calculates TF-IDF and transforms it into a matrix
- Calculates the cosine similarity matrix

THE ALGORITHM USED

the second one – using TF/IDF and Cosine Similarity



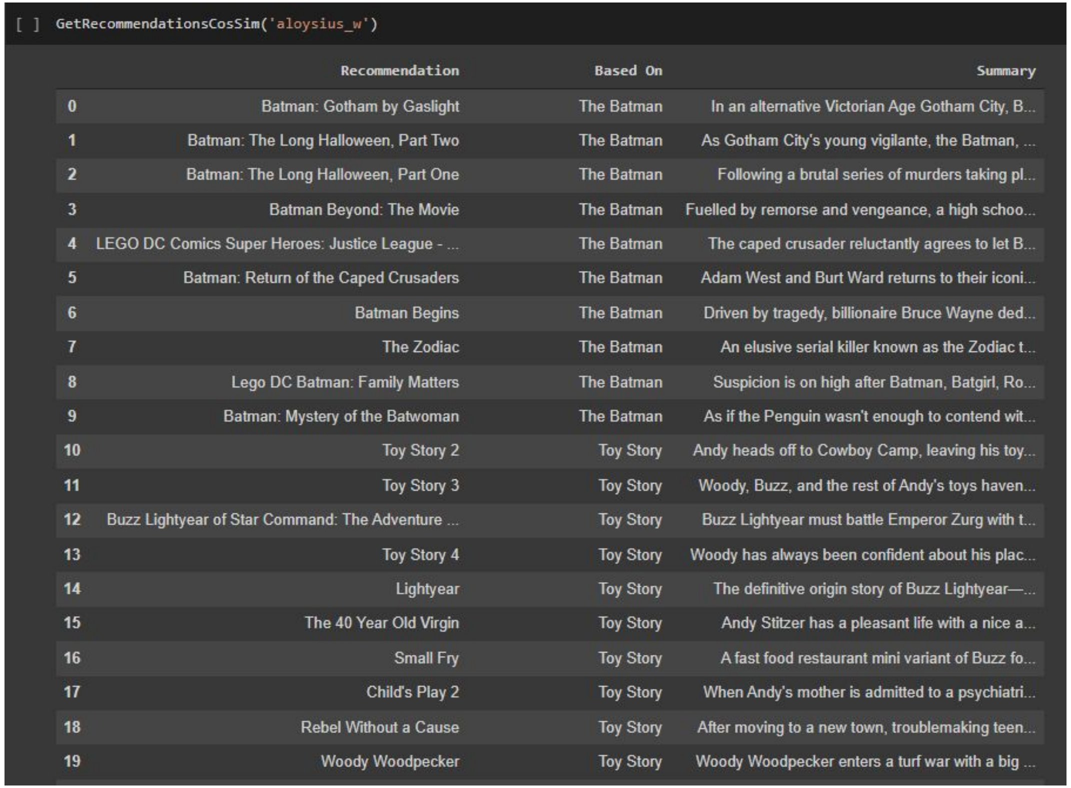
```
[ ] #Function to grab the movies that have been watched
    def GetUserData(user):
      moviesWatched = userDF.loc[userDF['User'] == user]
      moviesWatched.reset_index(drop=True, inplace=True)
      #print(moviesWatched)
      return moviesWatched['Titles_Watched'].values.tolist()
```

```
[ ] #Script to plot the recommendation using the similarity matrix
    def GetRecommendationsCosSim(user,cosineSim=cosineSim):
      #Get the movies that have been watched
      userMovies = GetUserData(user)
      #Make the dataframe to store the recommendation
      recommendedMovies = pd.DataFrame({'Recommendation': [''],'Based On': [''], 'Summary': ['']})
      #Counter to track the current row
      currentRow = 0
      #print(userMovies)
      #Make a recommendation based on each movie that has been watched
      for a in userMovies:
        idx = indices[a]
        simScores = list(enumerate(cosineSim[idx]))
        simScores = sorted(simScores, key=lambda x: x[1], reverse=True)
        simScores = simScores[1:11]
        movieIndices = [i[0] for i in simScores]
        tempTitle = movieDF['Title'].iloc[movieIndices]

        #For all the movie recommended, add to dataframe
        for b in tempTitle:
          index = movieDF.index[movieDF['Title']==b]
          #Get the plot overview from the main dataframe, remove the index from the search
          summaryOverview = movieDF.iloc[index,2].to_string(index=False)
          #Add in the current movie being looked at, the titles, and the overview into the dataframe
          recommendedMovies.loc[currentRow] = [b,a, summaryOverview]
          #Add to the current row counter
          currentRow += 1
      return recommendedMovies
```

- Two methods
  - One to get the movies that the user have watched
  - One to perform the recommendation
- TL;DR
  - Iterate through the movies that the user have watched
  - Uses the similarity matrix to get movie with similar overall description
  - Updates a recommendation dataframe

THE ALGORITHM USED

the second one – using TF/IDF and Cosine Similarity



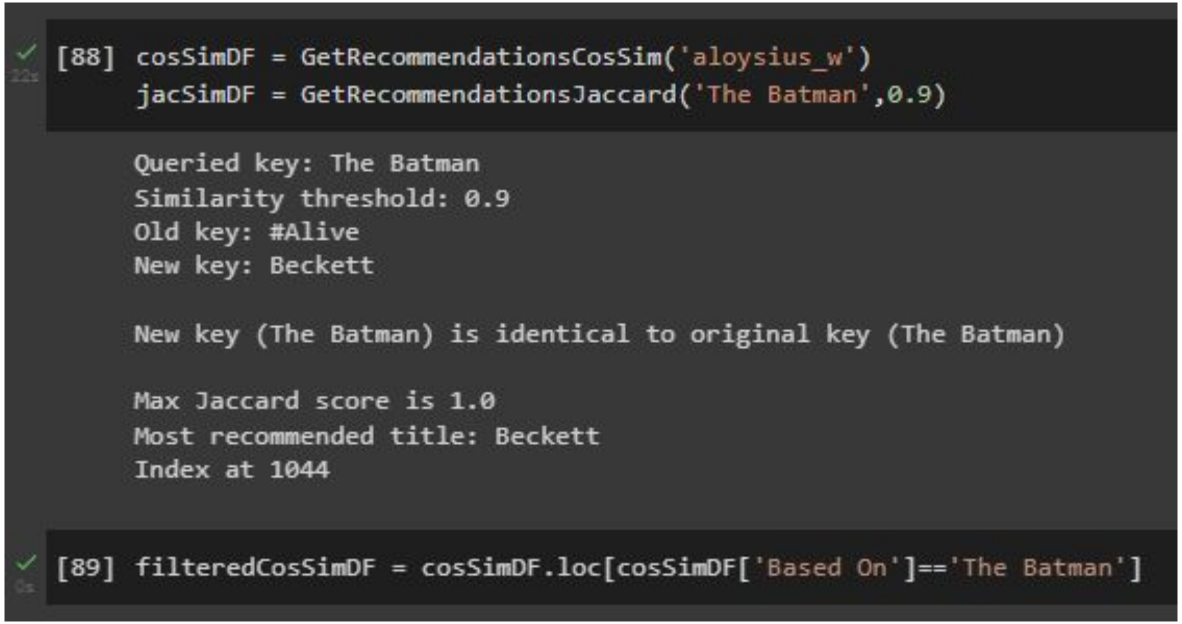
```
[ ] GetRecommendationsCosSim('aloysius_w')
```

	Recommendation	Based On	Summary
0	Batman: Gotham by Gaslight	The Batman	In an alternative Victorian Age Gotham City, B...
1	Batman: The Long Halloween, Part Two	The Batman	As Gotham City's young vigilante, the Batman, ...
2	Batman: The Long Halloween, Part One	The Batman	Following a brutal series of murders taking pl...
3	Batman Beyond: The Movie	The Batman	Fuelled by remorse and vengeance, a high schoo...
4	LEGO DC Comics Super Heroes: Justice League - ...	The Batman	The caped crusader reluctantly agrees to let B...
5	Batman: Return of the Caped Crusaders	The Batman	Adam West and Burt Ward returns to their iconi...
6	Batman Begins	The Batman	Driven by tragedy, billionaire Bruce Wayne ded...
7	The Zodiac	The Batman	An elusive serial killer known as the Zodiac t...
8	Lego DC Batman: Family Matters	The Batman	Suspicion is on high after Batman, Batgirl, Ro...
9	Batman: Mystery of the Batwoman	The Batman	As if the Penguin wasn't enough to contend wit...
10	Toy Story 2	Toy Story	Andy heads off to Cowboy Camp, leaving his toy...
11	Toy Story 3	Toy Story	Woody, Buzz, and the rest of Andy's toys haven...
12	Buzz Lightyear of Star Command: The Adventure ...	Toy Story	Buzz Lightyear must battle Emperor Zurg with t...
13	Toy Story 4	Toy Story	Woody has always been confident about his plac...
14	Lightyear	Toy Story	The definitive origin story of Buzz Lightyear—...
15	The 40 Year Old Virgin	Toy Story	Andy Stitzer has a pleasant life with a nice a...
16	Small Fry	Toy Story	A fast food restaurant mini variant of Buzz fo...
17	Child's Play 2	Toy Story	When Andy's mother is admitted to a psychiatri...
18	Rebel Without a Cause	Toy Story	After moving to a new town, troublemaking teen...
19	Woody Woodpecker	Toy Story	Woody Woodpecker enters a turf war with a big ...

- Sample result for user **'aloysius_w'**

THE RESULTS

the two algorithms compared

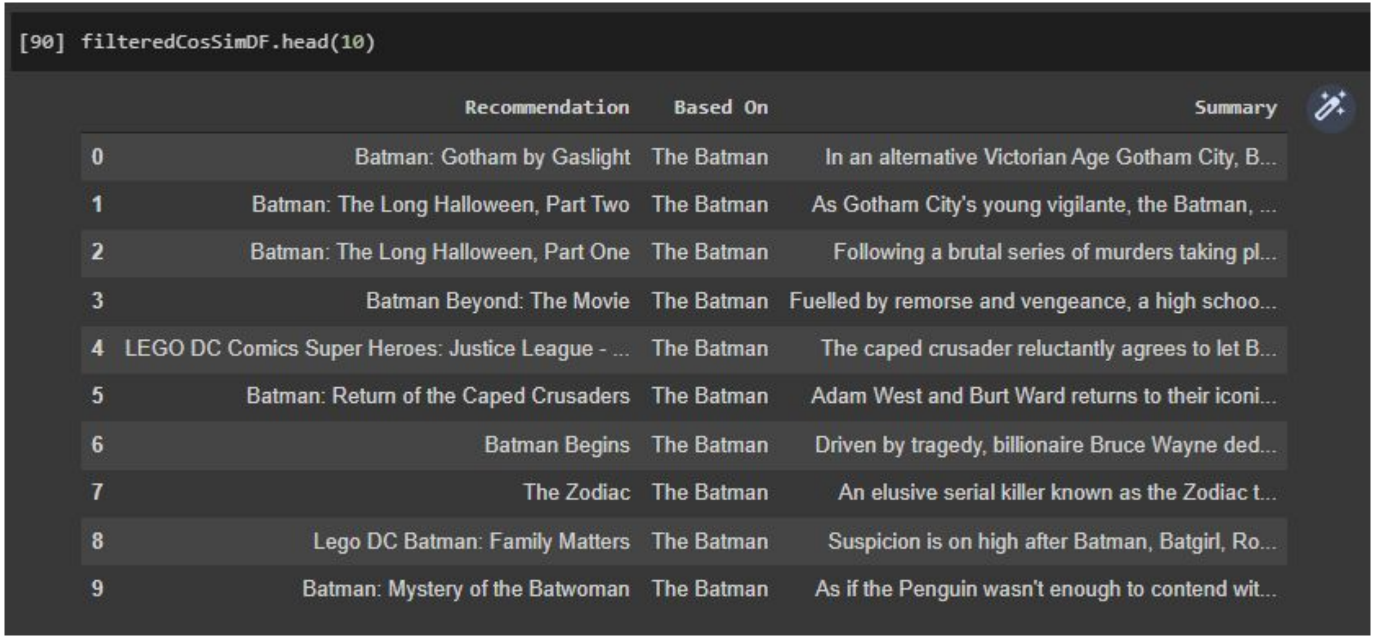


```
[88] cosSimDF = GetRecommendationsCosSim('aloysius_w')
     jacSimDF = GetRecommendationsJaccard('The Batman',0.9)

        Queried key: The Batman
        Similarity threshold: 0.9
        Old key: #Alive
        New key: Beckett

        New key (The Batman) is identical to original key (The Batman)

        Max Jaccard score is 1.0
        Most recommended title: Beckett
        Index at 1044

[89] filteredCosSimDF = cosSimDF.loc[cosSimDF['Based On']=='The Batman']
```

- Querying the same title using the two algorithms

THE RESULTS

the two algorithms compared



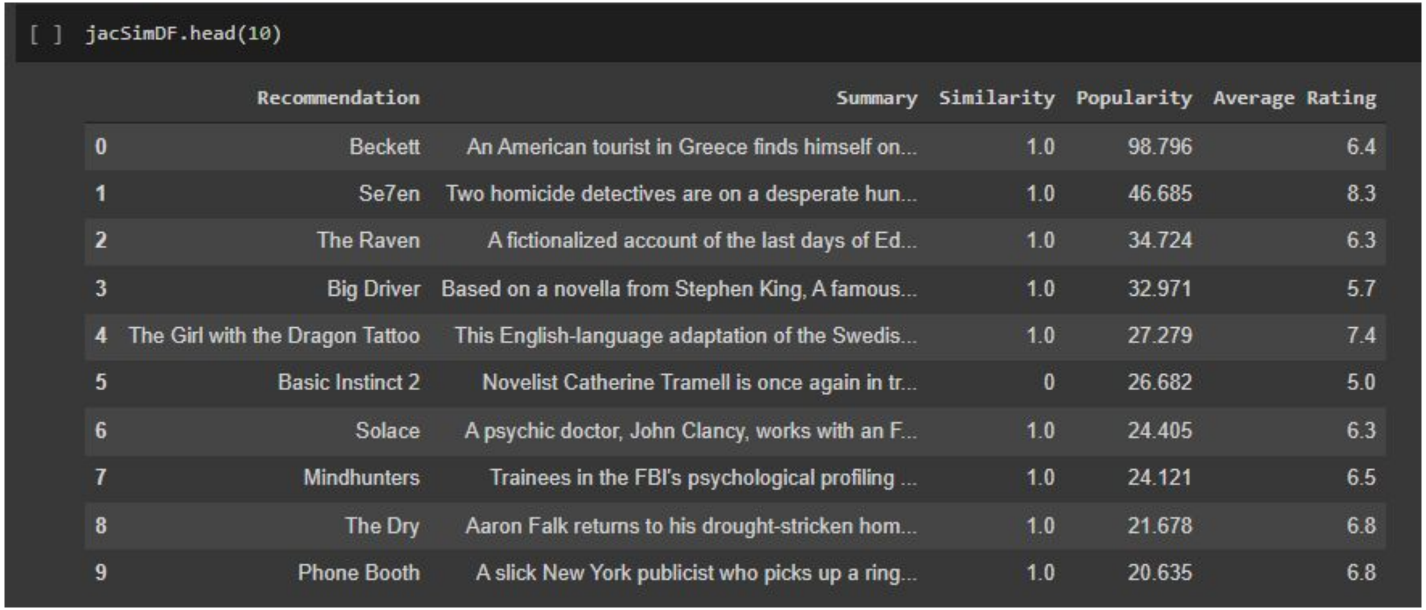
```
[90] filteredCosSimDF.head(10)
```

	Recommendation	Based On	Summary
0	Batman: Gotham by Gaslight	The Batman	In an alternative Victorian Age Gotham City, B...
1	Batman: The Long Halloween, Part Two	The Batman	As Gotham City's young vigilante, the Batman, ...
2	Batman: The Long Halloween, Part One	The Batman	Following a brutal series of murders taking pl...
3	Batman Beyond: The Movie	The Batman	Fuelled by remorse and vengeance, a high schoo...
4	LEGO DC Comics Super Heroes: Justice League - ...	The Batman	The caped crusader reluctantly agrees to let B...
5	Batman: Return of the Caped Crusaders	The Batman	Adam West and Burt Ward returns to their iconi...
6	Batman Begins	The Batman	Driven by tragedy, billionaire Bruce Wayne ded...
7	The Zodiac	The Batman	An elusive serial killer known as the Zodiac t...
8	Lego DC Batman: Family Matters	The Batman	Suspicion is on high after Batman, Batgirl, Ro...
9	Batman: Mystery of the Batwoman	The Batman	As if the Penguin wasn't enough to contend wit...

- Results by the first algorithm
  - Recommends movies with similar keywords

THE RESULTS

the two algorithms compared



```
[ ] jacSimDF.head(10)
```

	Recommendation	Summary	Similarity	Popularity	Average Rating
0	Beckett	An American tourist in Greece finds himself on...	1.0	98.796	6.4
1	Se7en	Two homicide detectives are on a desperate hun...	1.0	46.685	8.3
2	The Raven	A fictionalized account of the last days of Ed...	1.0	34.724	6.3
3	Big Driver	Based on a novella from Stephen King, A famous...	1.0	32.971	5.7
4	The Girl with the Dragon Tattoo	This English-language adaptation of the Swedis...	1.0	27.279	7.4
5	Basic Instinct 2	Novelist Catherine Tramell is once again in tr...	0	26.682	5.0
6	Solace	A psychic doctor, John Clancy, works with an F...	1.0	24.405	6.3
7	Mindhunters	Trainees in the FBI's psychological profiling ...	1.0	24.121	6.5
8	The Dry	Aaron Falk returns to his drought-stricken hom...	1.0	21.678	6.8
9	Phone Booth	A slick New York publicist who picks up a ring...	1.0	20.635	6.8

- Results by the second algorithm
  - Recommends movies that are of similar genre

Conclusion

- With our project and dataset, we can predict and recommend movies that one may enjoy watching.
- People won’t have to waste time trying movies they may not like.
- This resolves the problem of not being able to find new movies to watch.

Thank You!
Any Questions?

References

- Dataset Link: https://www.kaggle.com/datasets/disham993/9000-movies-dataset
- Help with creating a Recommender System Without User Preferences 1: https://towardsdatascience.com/nlp-based-recommender-system-without-user-preferences-7077f4474107
- Help with creating a Recommender System Without User Preferences 2: https://datascience.stackexchange.com/questions/42495/can-a-recommendation-system-be-built-without-any-user-ratings#:~:text=on%20this%20post.-,Yes.,often%20called%20More%20Like%20This
- Hands On Assignments
- Pandas documentation