Scholar English Academy

SURAT

A PROJECT REPORT ON

Movie Categorization Based on Genres, Ratings, and Popularity

Scholar English Academy Surat

SUBMITTED TO	**SUBMITTED BY:**
MR. Ravi Agarwal	**Arya Amin**
P.G.T.(COMP. SC)	

<u>CERTIFICATE</u>

This is to certify that **Arya Amin** of class **XII B** of **Scholar English Academy** has done his/her project on **MOVIE CATEGORIZATION BASED ON GENRES, RATINGS, AND POPULARITY** under my supervision. He/She has taken interest and has shown at most sincerity in completion of this project.

I certify this Project up to my expectation & as per guidelines issued by **CBSE, NEW DELHI**.

Internal Examiner *External Examiner*

Principal

ACKNOWLEDGMENT

It is with pleasure that I acknowledge my sincere gratitude to our teacher, Mr. Ravi Agarwal who taught and undertook the responsibility of teaching the subject of computer science. I have been greatly benefited from his classes.

I am especially indebted to our Principal Dr. Vatsal Bhatt who has always been a source of encouragement and support and without whose inspiration this project would not have been a success I would like to place on record heartfelt thanks to him.

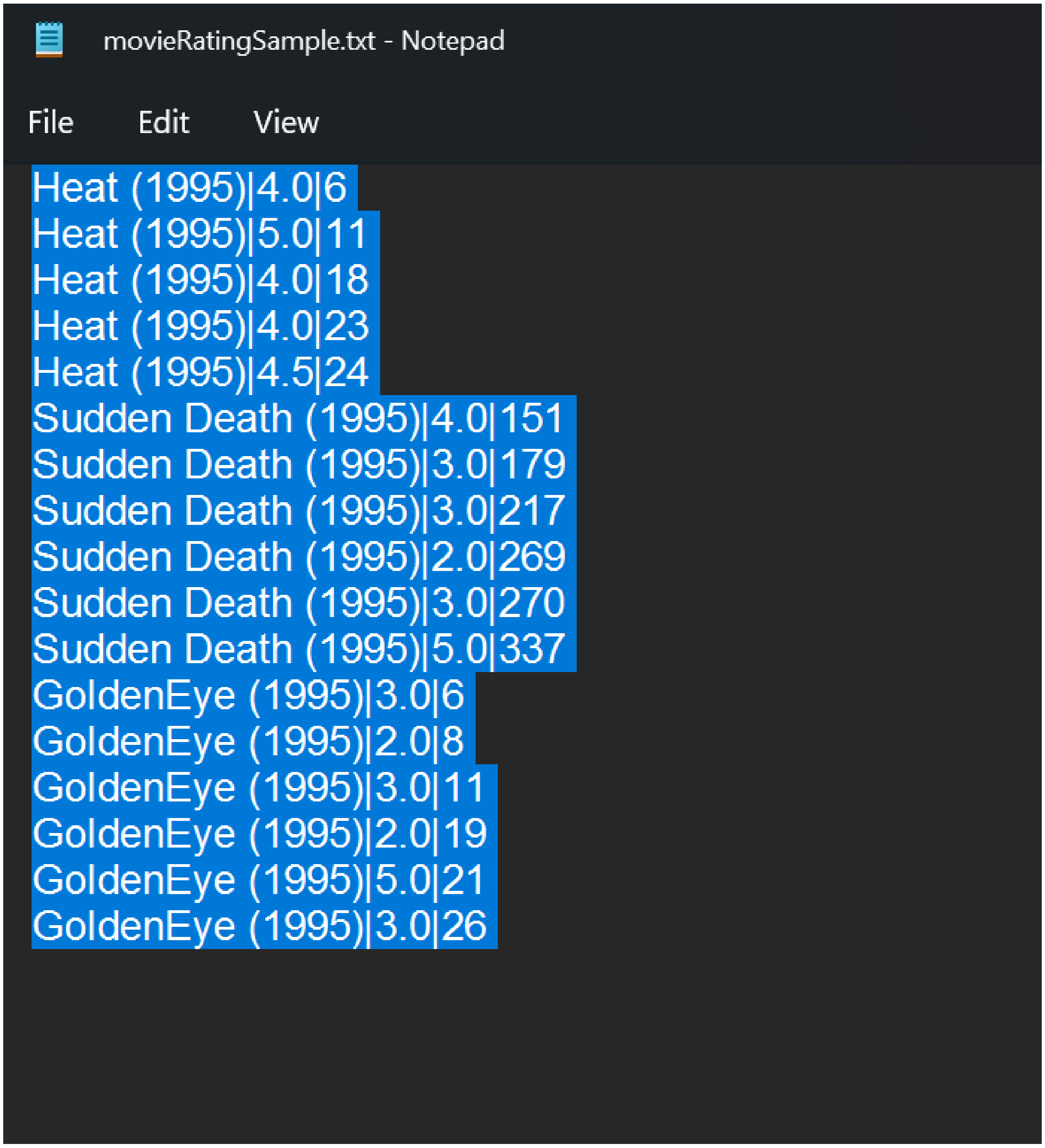
Finally, I would like to express my sincere appreciation for all the other students for my batch their friendship & the fine times that we all shared together.

HARDWARE AND SOFTWARE REQUIREMENTS

There were some prerequisites to build this project:

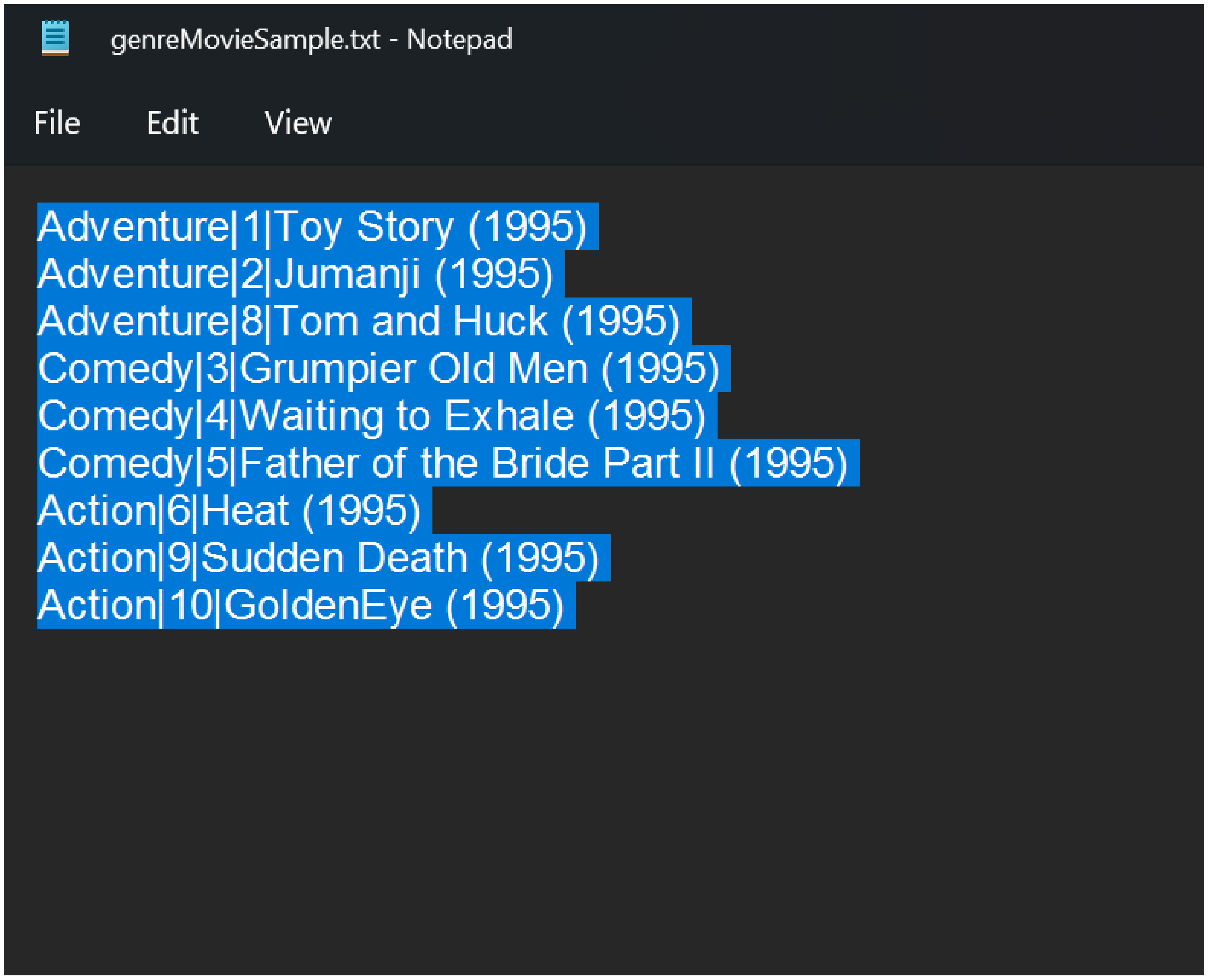
- Knowledge of basic Python data structures and their manipulation
- Some advanced functions and syntax of Python
- Handling of text files and their manipulation
- Sorting techniques and standard built-in Python libraries

Further, I created two text files (movieRatingSample.txt and genreRatingSample.txt) for using its contents as sample data. Refer to the files below.


movieRatingSample.txt - Notepad

File Edit View

```
Heat (1995)|4.0|6
Heat (1995)|5.0|11
Heat (1995)|4.0|18
Heat (1995)|4.0|23
Heat (1995)|4.5|24
Sudden Death (1995)|4.0|151
Sudden Death (1995)|3.0|179
Sudden Death (1995)|3.0|217
Sudden Death (1995)|2.0|269
Sudden Death (1995)|3.0|270
Sudden Death (1995)|5.0|337
GoldenEye (1995)|3.0|6
GoldenEye (1995)|2.0|8
GoldenEye (1995)|3.0|11
GoldenEye (1995)|2.0|19
GoldenEye (1995)|5.0|21
GoldenEye (1995)|3.0|26
```



```
Adventure|1|Toy Story (1995)
Adventure|2|Jumanji (1995)
Adventure|8|Tom and Huck (1995)
Comedy|3|Grumpier Old Men (1995)
Comedy|4|Waiting to Exhale (1995)
Comedy|5|Father of the Bride Part II (1995)
Action|6|Heat (1995)
Action|9|Sudden Death (1995)
Action|10|GoldenEye (1995)
```

CODING

Task1: Reading Data

Write a function read_ratings_data(f) that takes in a ratings file, and returns a dictionary. The dictionary should have movie as key, and the corresponding list of ratings as value.
For example: movie_ratings_dict = { "The Lion King (2019)" : [6.0, 7.5, 5.1], "Titanic (1997)": [7]

```
def read_ratings_data(f):
    dict = {}
    lst=f.readlines()
    ratings=[]
    for j in range(0,len(lst),6):
        rating=[]
        for i in range(6):
            mov=lst[i+j].split("|")
            rating.append(eval(mov[1]))
        ratings.append(rating)
    for i in range(0,len(lst),6):
        mov=(lst[i].split("|"))
        dict[mov[0]]=ratings[(i//6)]
    return dict
```

Write a function read_movie_genre(f) that takes in a movies file and returns a dictionary. The dictionary should have a one-to-one mapping between movie and genre.
For example { "Toy Story (1995)" : "Adventure", "Golden Eye (1995)" : "Action" }

```
def read_movie_genre(f):
    dict = {}
    lst=f.readlines()
    for i in range(0,len(lst)):
        mov=lst[i].split("|")
        dict[mov[2].rstrip()]=mov[0]
    return dict
```

#Task 2: Processing Data

1. Genre dictionary
Write a function create_genre_dict that takes as a parameter a movie-to-genre dictionary, of the kind created in Task 1.2. The function should return another dictionary in which a

genre is mapped to all the movies in that genre.
For example: { genre1: [m1, m2, m3], genre2: [m6, m7] }

```
def create_genre_dict(d):
    dict = {}
    genres=list(d.values())
    movs=list(d.keys())
    for i in range(0,9,3):
        movs_list=[]
        for j in range(0,3):
            movs_list.append(movs[i+j])
        dict[genres[i]]=movs_list
    return dict
```

2. Average Rating
Write a function calculate_average_rating that takes as a parameter a ratings dictionary, of the kind created in Task 1.1. It should return a dictionary where the movie is mapped to its average rating computed from the ratings list.
For example: {"Spider-Man (2002)": [3,2,4,5]} ==> {"Spider-Man (2002)": 3.5}

```
def calculate_average_rating(d):
    dict = {}
    for i in d:
        dict[i]= round(sum(d.get(i))/len(d.get(i)),2)
    return dict
```

#Task 3: Recommendation

1. Popularity based
In services such as Netflix and Spotify, you often see recommendations with the heading "Popular movies" or "Trending top 10".
Write a function get_popular_movies that takes as parameters a dictionary of movie-to-average rating (as created in Task 2.2), and an integer n (default should be 10). The function should return a dictionary (movie:average rating, same structure as input dictionary) of top n movies based on the average ratings. (If there are fewer movies than n, it should all return all movies in order of top average ratings.)

```
def get_popular_movies(d,n=10):
    dict={}
    if n > len(d):
        n=len(d)
    mov_lst=list(d.items())
    for i in range(len(mov_lst)):
        for j in range(len(mov_lst)-i-1):
            if mov_lst[j][1] < mov_lst[j+1][1]:
                mov_lst[j],mov_lst[j+1]=mov_lst[j+1],mov_lst[j]
    for i in range(n):
        dict[mov_lst[i][0]]=mov_lst[i][1]
    return dict
```

#2. Threshold Rating
Write a function filter_movies that takes as parameters a dictionary of movie-to-average rating (same as for the popularity based function above), and a threshold rating with default value of 3. The function should filter movies based on the threshold rating, and return a dictionary with the same structure as the input. For example, if the threshold rating is 3.5, the returned dictionary should have only those movies from the input whose average rating is equal to or greater than 3.5.

```
def filter_movies(d,t=3):
    dict={}
    for i in d:
        if d[i]>=t:
            dict[i]=d[i]
    return dict
```

3. Popularity + Genre based
In most recommendation systems, genre of the movie/song/book plays an important role. Often features like popularity, genre, artist are combined to present recommendations to a user.
Write a function get_popular_in_genre that, given a genre, a genre-to-movies dictionary (as created in Task 2.1), a dictionary of movie:average rating (as created in Task 2.2), and an integer n (default 5), returns the top n most popular movies in that genre based on the average ratings. The return value should be a dictionary of movie-to-average rating of movies that make the cut. Genre categories will be from those in the movie:genre dictionary created in Task 1.2. Your code should handle the case when there are fewer than n movies in the data, as in Task 3.1 above.

```
def get_popular_in_genre(g,d1,d2,n=5):
    dict={}
    movs=[]
    for i in d1:
        if g==i:
            for j in d1[i]:
                movs.append(j)

    d3=get_popular_movies(d2)
    if n > len(movs):
        n=len(movs)

    for i in d3:
        if i in movs:
            dict[i]=d3[i]
            n-=1
        if n==0:
            break

    return dict
```

4. Genre Rating
One important analysis for the content platforms is to determine ratings by genre. Write a function get_genre_rating that takes the same parameters as get_popular_in_genre above, except for n, and returns the average rating of the movies in the given genre.

```
def get_genre_rating(g,d1,d2):
    movs=[]
    for i in d1:
        if g==i:
            for j in d1[i]:
                movs.append(j)

    sum=0
    for i in d2:
        if i in movs:
            sum+=d2[i]
    avg=round(sum/len(movs),2)
    return avg
```

5. Genre Popularity
Write a function genre_popularity that takes as parameters a genre-to-movies dictionary (as created in Task 2.1), a movie-to-average rating dictionary (as created in Task 2.2), and n (default 5), and returns the top-n rated genres as a dictionary of genre:average rating. Hint: Use the above get_genre_rating function as a helper.

```
def genre_popularity(d1,d2,n=5):
    gen_lst=[]
    dict={}
    for i in d1:
        gen_lst.append((i,get_genre_rating(i,d1,d2)))

    for i in range(len(gen_lst)):
        for j in range(len(gen_lst)-i-1):
            if gen_lst[j][1] < gen_lst[j+1][1]:
                gen_lst[j],gen_lst[j+1]=gen_lst[j+1],gen_lst[j]

    if n > len(gen_lst):
        n=len(gen_lst)

    for i in range(n):
        dict[gen_lst[i][0]]= gen_lst[i][1]

    return dict

    #Task 4 (User-Focused)
```

1. Read the rating file to return a user-to-movies dictionary that maps user ID to the associated movies and the corresponding ratings. Write a function named read_user_ratings for this, with the rating file as the parameter.
For example: { u1: [(m1, r1), (m2, r2)], u2: [(m3, r3), (m8, r8)] }
where ui is user ID, mi is movie, ri is corresponding rating.

```
def read_user_ratings(f):
    dict = {}
    f1.seek(0)
    lst=f.readlines()

    for i in range(len(lst)):
        s=lst[i].rstrip()
        mov=s.split("|")
        u=eval(mov[2])
        dict[u]=[]

    for i in range(len(lst)):
        mov=lst[i].rstrip().split("|")
        m=mov[0]
        r=eval(mov[1])
        u=eval(mov[2])
        tup=(m,r)
        dict[u].append(tup)

    return dict
```

2. Write a function get_user_genre that takes as parameters a user id, the user-to-movies dictionary (as created in Task 4.1 above), and the movie-to-genre dictionary (as created in Task 1.2), and returns the top genre that the user likes based on the user's ratings.

Here, the top genre for the user will be determined by taking the average rating of the movies genre-wise that the user has rated.

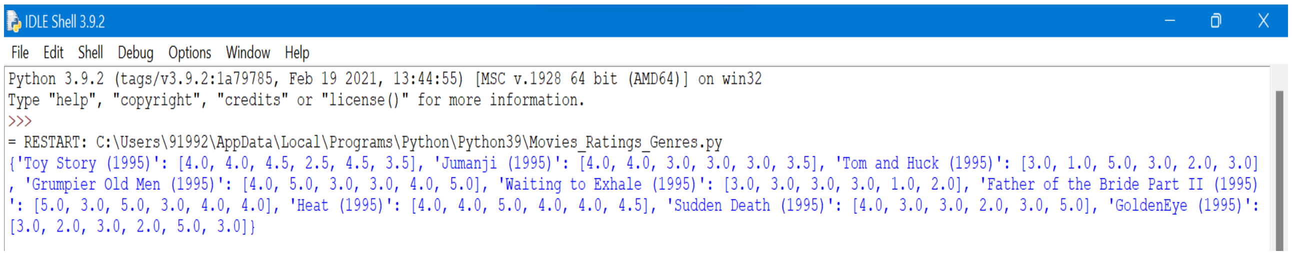
```
def get_user_genre(u,d1,d2):
    u_movs=d1[u]
    u_genres=[]
    for i in u_movs:
        u_genres.append(d2[i[0]])
    dict_genre_to_movrat={}
    for i in range(len(u_genres)):
        dict_genre_to_movrat[u_genres[i]]=[]
    for i in dict_genre_to_movrat:
        for j in u_movs:
            if i==d2[j[0]]:
                dict_genre_to_movrat[i].append(j)
    dict_genre_to_rat={}
    for i in dict_genre_to_movrat:
        sum=0
        for j in dict_genre_to_movrat[i]:
            sum+=j[1]
        avg=round(sum/len(dict_genre_to_movrat[i]),2)
        dict_genre_to_rat[i]=avg

    max=None
    for i in dict_genre_to_rat:
        for j in dict_genre_to_rat:
            if dict_genre_to_rat[i]<=dict_genre_to_rat[j]:
                max=j

    return max
```

3. Recommend 3 most popular (highest average rating) movies from the user's top genre that the user has not yet rated. Write a function recommend_movies for this, that takes the parameters a user id, the user-to-movies dictionary (as created in Task 4.1 above), the movie-to-genre dictionary (as created in Task 1.2), and the movie-to-average rating dictionary (as created in Task 2.2). The function should return a dictionary of movie-to-average ratings. (Return all if fewer than 3 movies make the cut.)

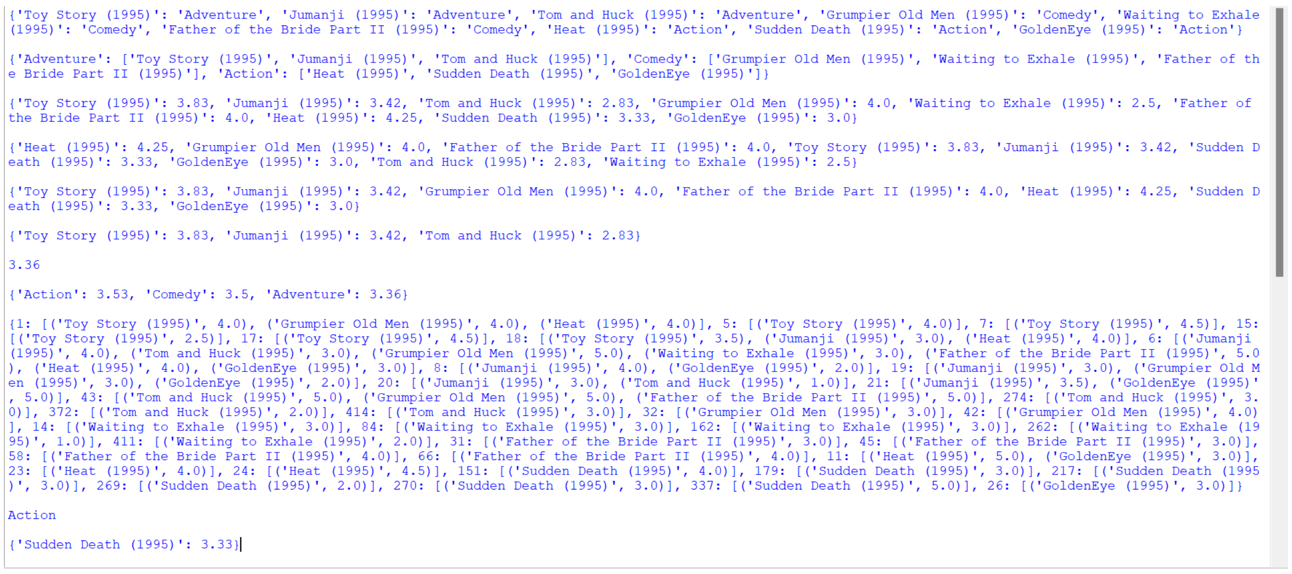
```
def recommend_movies(u,d1,d2,d3):
    dict={}
    u_movs=[]
    for i in d1[u]:
        u_movs.append(i[0])
    top_genre=get_user_genre(u,d1,d2)
    d4=create_genre_dict(d2)
    dict=get_popular_in_genre(top_genre,d4,d3,len(d4[top_genre]))
    dict_new={}
    for i in dict:
        if i in u_movs:
            pass
        else:
            dict_new[i]=dict[i]

    if len(dict_new)<3:
        n=len(dict_new)
    else:
        n=3

    dict_final={}
    lst=list(dict_new.items())
    for i in range(n):
        dict_final[lst[i][0]]=lst[i][1]

    return dict_new


f1=open("movieRatingSample.txt","r")
dict_read_ratings_data=read_ratings_data(f1)
print(dict_read_ratings_data)
print()

f2=open("genreMovieSample.txt","r")
dict_read_movie_genre=read_movie_genre(f2)
print(dict_read_movie_genre)
print()

dict_create_genre_dict=create_genre_dict(dict_read_movie_genre)
print(dict_create_genre_dict)
print()

dict_calculate_average_rating=calculate_average_rating(dict_read_ratings_data)
print(dict_calculate_average_rating)
print()

dict_get_popular_movies=get_popular_movies(dict_calculate_average_rating)
print(dict_get_popular_movies)
print()

dict_filter_movies=filter_movies(dict_calculate_average_rating)
```

```
print(dict_filter_movies)
print()

print(get_popular_in_genre("Adventure",dict_create_genre_dict,dict_calculate_average_rating,4))
print()

print(get_genre_rating("Adventure",dict_create_genre_dict,dict_calculate_average_rating))
print()

print(genre_popularity(dict_create_genre_dict,dict_calculate_average_rating))
print()

dict_read_user_ratings=read_user_ratings(f1)
print(dict_read_user_ratings)
print()

print(get_user_genre(6,dict_read_user_ratings,dict_read_movie_genre))
print()

print(recommend_movies(6,dict_read_user_ratings,dict_read_movie_genre,dict_calculate_averag
e_rating))
print()

f1.close()
f2.close()
```

OUTPUT



```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
= RESTART: C:\Users\91992\AppData\Local\Programs\Python\Python39\Movies_Ratings_Genres.py
{'Toy Story (1995)': [4.0, 4.0, 4.5, 2.5, 4.5, 3.5], 'Jumanji (1995)': [4.0, 4.0, 3.0, 3.0, 3.0, 3.5], 'Tom and Huck (1995)': [3.0, 1.0, 5.0, 3.0, 2.0, 3.0]
, 'Grumpier Old Men (1995)': [4.0, 5.0, 3.0, 3.0, 4.0, 5.0], 'Waiting to Exhale (1995)': [3.0, 3.0, 3.0, 3.0, 1.0, 2.0], 'Father of the Bride Part II (1995)
': [5.0, 3.0, 5.0, 3.0, 4.0, 4.0], 'Heat (1995)': [4.0, 4.0, 5.0, 4.0, 4.0, 4.5], 'Sudden Death (1995)': [4.0, 3.0, 3.0, 2.0, 3.0, 5.0], 'GoldenEye (1995)':
[3.0, 2.0, 3.0, 2.0, 5.0, 3.0]}
```



```
{'Toy Story (1995)': 'Adventure', 'Jumanji (1995)': 'Adventure', 'Tom and Huck (1995)': 'Adventure', 'Grumpier Old Men (1995)': 'Comedy', 'Waiting to Exhale
(1995)': 'Comedy', 'Father of the Bride Part II (1995)': 'Comedy', 'Heat (1995)': 'Action', 'Sudden Death (1995)': 'Action', 'GoldenEye (1995)': 'Action'}

{'Adventure': ['Toy Story (1995)', 'Jumanji (1995)', 'Tom and Huck (1995)'], 'Comedy': ['Grumpier Old Men (1995)', 'Waiting to Exhale (1995)', 'Father of th
e Bride Part II (1995)'], 'Action': ['Heat (1995)', 'Sudden Death (1995)', 'GoldenEye (1995)']}

{'Toy Story (1995)': 3.83, 'Jumanji (1995)': 3.42, 'Tom and Huck (1995)': 2.83, 'Grumpier Old Men (1995)': 4.0, 'Waiting to Exhale (1995)': 2.5, 'Father of
the Bride Part II (1995)': 4.0, 'Heat (1995)': 4.25, 'Sudden Death (1995)': 3.33, 'GoldenEye (1995)': 3.0}

{'Heat (1995)': 4.25, 'Grumpier Old Men (1995)': 4.0, 'Father of the Bride Part II (1995)': 4.0, 'Toy Story (1995)': 3.83, 'Jumanji (1995)': 3.42, 'Sudden D
eath (1995)': 3.33, 'GoldenEye (1995)': 3.0, 'Tom and Huck (1995)': 2.83, 'Waiting to Exhale (1995)': 2.5}

{'Toy Story (1995)': 3.83, 'Jumanji (1995)': 3.42, 'Grumpier Old Men (1995)': 4.0, 'Father of the Bride Part II (1995)': 4.0, 'Heat (1995)': 4.25, 'Sudden D
eath (1995)': 3.33, 'GoldenEye (1995)': 3.0}

{'Toy Story (1995)': 3.83, 'Jumanji (1995)': 3.42, 'Tom and Huck (1995)': 2.83}

3.36

{'Action': 3.53, 'Comedy': 3.5, 'Adventure': 3.36}

{1: [('Toy Story (1995)', 4.0), ('Grumpier Old Men (1995)', 4.0), ('Heat (1995)', 4.0)], 5: [('Toy Story (1995)', 4.0)], 7: [('Toy Story (1995)', 4.5)], 15:
[('Toy Story (1995)', 2.5)], 17: [('Toy Story (1995)', 4.5)], 18: [('Toy Story (1995)', 3.5), ('Jumanji (1995)', 3.0), ('Heat (1995)', 4.0)], 6: [('Jumanji
(1995)', 4.0), ('Tom and Huck (1995)', 3.0), ('Grumpier Old Men (1995)', 5.0), ('Waiting to Exhale (1995)', 3.0), ('Father of the Bride Part II (1995)', 5.0
), ('Heat (1995)', 4.0), ('GoldenEye (1995)', 3.0)], 8: [('Jumanji (1995)', 4.0), ('GoldenEye (1995)', 2.0)], 19: [('Jumanji (1995)', 3.0), ('Grumpier Old M
en (1995)', 3.0), ('GoldenEye (1995)', 2.0)], 20: [('Jumanji (1995)', 3.0), ('Tom and Huck (1995)', 1.0)], 21: [('Jumanji (1995)', 3.5), ('GoldenEye (1995)'
, 5.0)], 43: [('Tom and Huck (1995)', 5.0), ('Grumpier Old Men (1995)', 5.0), ('Father of the Bride Part II (1995)', 5.0)], 274: [('Tom and Huck (1995)', 3.
0)], 372: [('Tom and Huck (1995)', 2.0)], 414: [('Tom and Huck (1995)', 3.0)], 32: [('Grumpier Old Men (1995)', 3.0)], 42: [('Grumpier Old Men (1995)', 4.0)
], 14: [('Waiting to Exhale (1995)', 3.0)], 84: [('Waiting to Exhale (1995)', 3.0)], 162: [('Waiting to Exhale (1995)', 3.0)], 262: [('Waiting to Exhale (19
95)', 1.0)], 411: [('Waiting to Exhale (1995)', 2.0)], 31: [('Father of the Bride Part II (1995)', 3.0)], 45: [('Father of the Bride Part II (1995)', 3.0)],
58: [('Father of the Bride Part II (1995)', 4.0)], 66: [('Father of the Bride Part II (1995)', 4.0)], 11: [('Heat (1995)', 5.0), ('GoldenEye (1995)', 3.0)],
23: [('Heat (1995)', 4.0)], 24: [('Heat (1995)', 4.5)], 151: [('Sudden Death (1995)', 4.0)], 179: [('Sudden Death (1995)', 3.0)], 217: [('Sudden Death (1995
)', 3.0)], 269: [('Sudden Death (1995)', 2.0)], 270: [('Sudden Death (1995)', 3.0)], 337: [('Sudden Death (1995)', 5.0)], 26: [('GoldenEye (1995)', 3.0)]}

Action

{'Sudden Death (1995)': 3.33}
```

BIBLIOGRAPHY

1. Preeti Arora Class 12