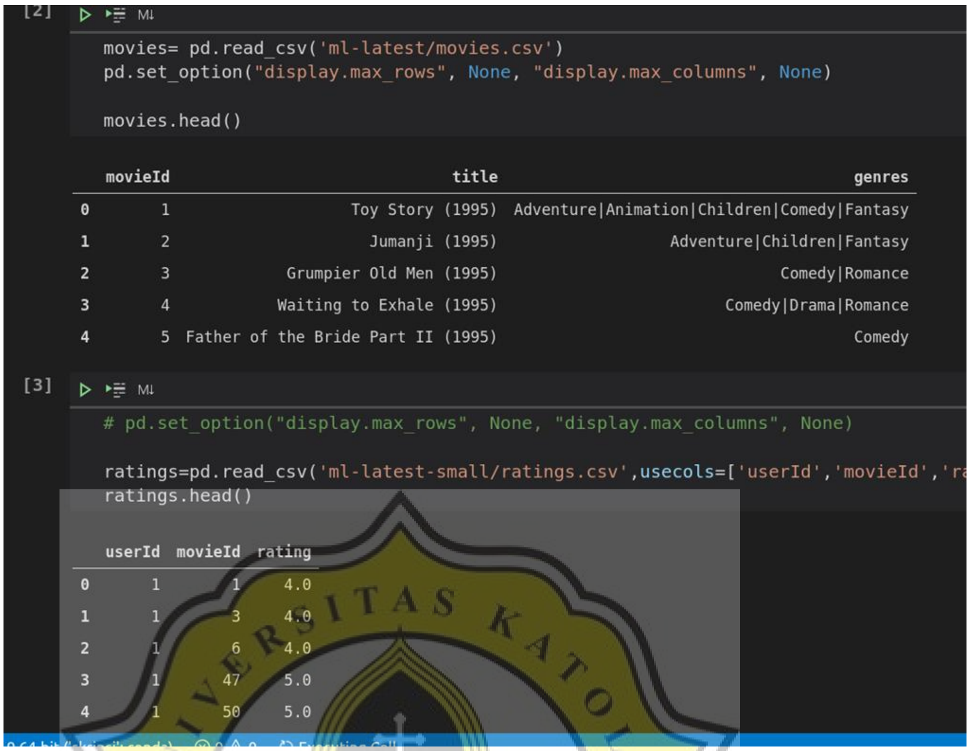
CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Implementation

The project utilizes Python code as its main programming language, the reason is Python is very practical in data science field, easier and more popular among data scientist compared to other language such as Java and C++ that may need more code definition, class, etc. just for the data preprocessing. This in fact can save a lot of time for implementation.

In every data science related field data preprocessing is a very serious thing that must be done firsthand before continuing the work, pre – processing will show all the possibilities so that we know how the dataset should behave and modeled according to our needs for the project as mentioned in Chapter 4.



```
[2]
movies= pd.read_csv('ml-latest/movies.csv')
pd.set_option("display.max_rows", None, "display.max_columns", None)

movies.head()
```

```
   movieId                               title                                       genres
0        1                    Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
1        2                      Jumanji (1995)                   Adventure|Children|Fantasy
2        3             Grumpier Old Men (1995)                               Comedy|Romance
3        4            Waiting to Exhale (1995)                         Comedy|Drama|Romance
4        5  Father of the Bride Part II (1995)                                       Comedy
```

```
[3]
# pd.set_option("display.max_rows", None, "display.max_columns", None)

ratings=pd.read_csv('ml-latest-small/ratings.csv',usecols=['userId','movieId','ra
ratings.head()
```

```
   userId  movieId  rating
0       1        1     4.0
1       1        3     4.0
2       1        6     4.0
3       1       47     5.0
4       1       50     5.0
```

Figure 5.1.1 Data Pre-Processing

Average User rated a movie differently for one user to another, in this page the average rating and number of movies rated by each user is calculated with a group by clause just like in database to show how many movies rated by the users and what is the average ratings of rated movies.

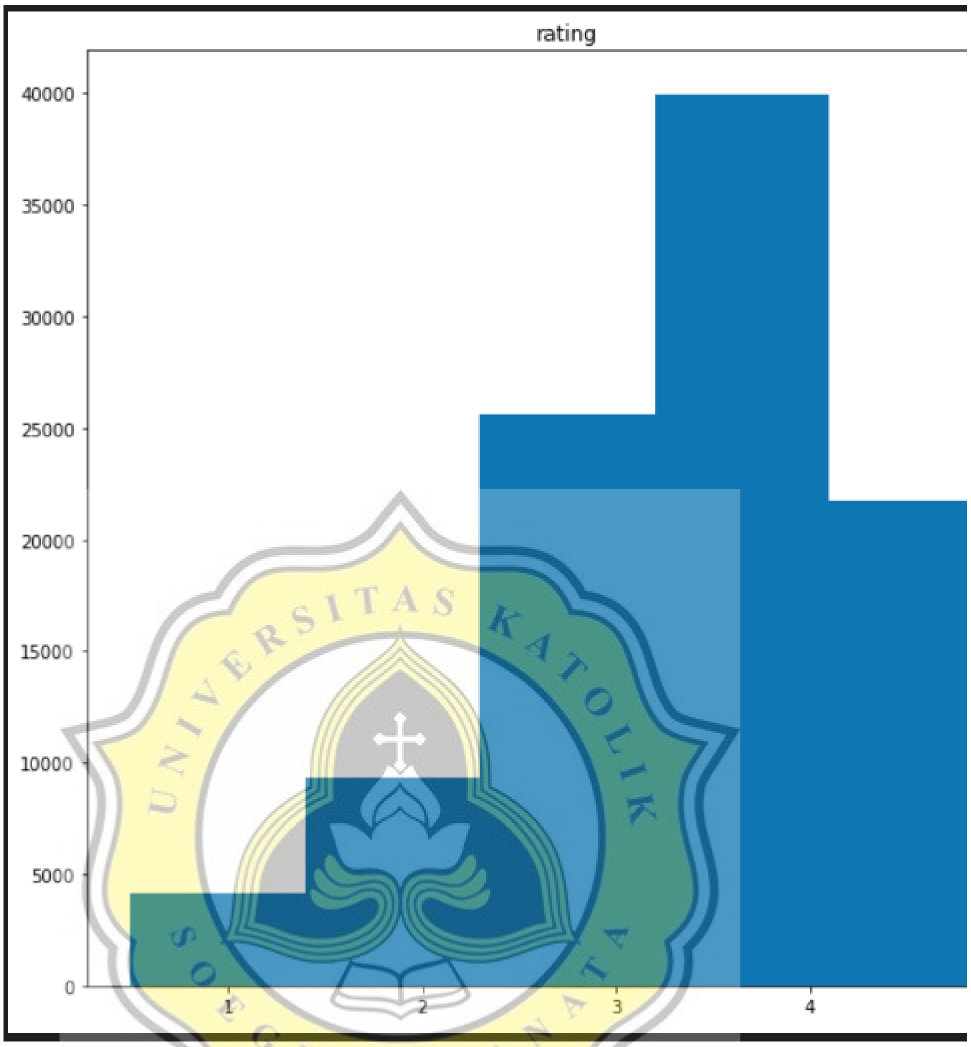
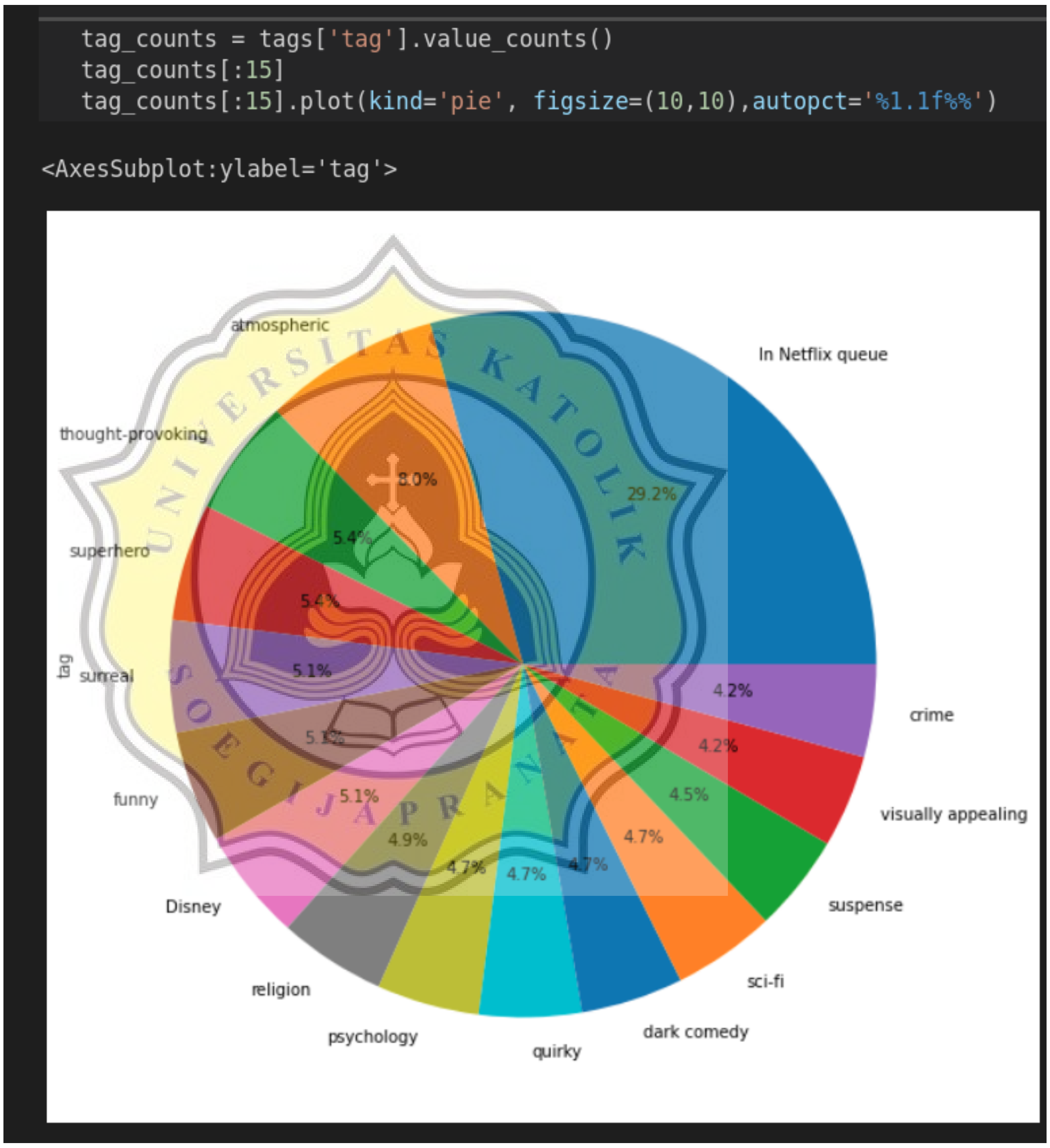


Figure 5.1.2 Average User's rating

In this figure we see from above we can conclude that the most given ratings the rating average is about 3.5, the second group of user’s voted about 3 to 3.5 and the least or fewest group of users throw a score of 1 and less than 2 for the given record.

In the implementation stage movie genre is also one of the main factor a recommender engine is created, even though the fucus of this project will be the algorithm it is worth mentioning that genre chart also play an important role for the algorithm or method to perform with the expected or unexpected result, remember this project is a measurement between two algorithms so similar or not similar result may be happening.



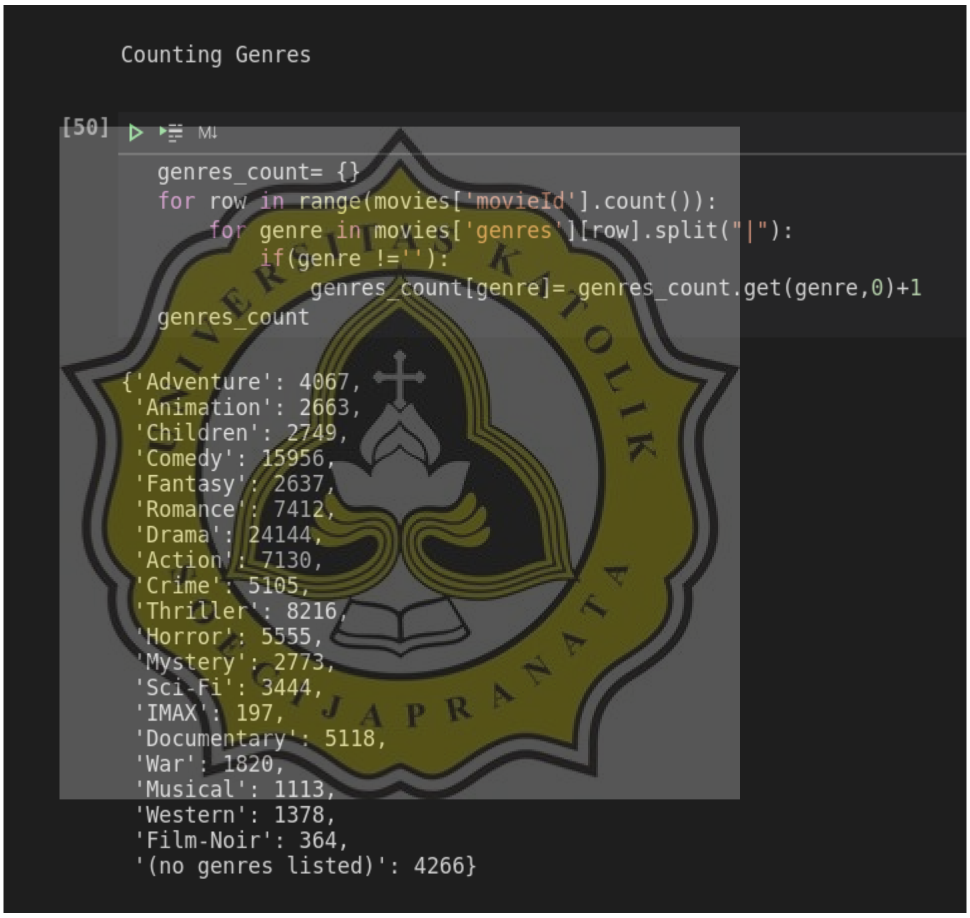
```
tag_counts = tags['tag'].value_counts()
tag_counts[:15]
tag_counts[:15].plot(kind='pie', figsize=(10,10),autopct='%1.1f%%')
```

```
<AxesSubplot:ylabel='tag'>
```

Figure 5.1.3 Tag / genre Pie Chart

In the above chart we can see that Netflix queue have the highest number of tag that is yet to be determined, this is a good picture of how the recommender system will behave based on the provided data from the csv files.

Genre in each movie is very crucial thing and will affect the recommender as a whole so I decided to count every genre inside the dataset



Counting Genres

```
[50]
genres_count= {}
for row in range(movies['movieId'].count()):
    for genre in movies['genres'][row].split("|"):
        if(genre !=''):
            genres_count[genre]= genres_count.get(genre,0)+1
genres_count
```

```
{'Adventure': 4067,
 'Animation': 2663,
 'Children': 2749,
 'Comedy': 15956,
 'Fantasy': 2637,
 'Romance': 7412,
 'Drama': 24144,
 'Action': 7130,
 'Crime': 5105,
 'Thriller': 8216,
 'Horror': 5555,
 'Mystery': 2773,
 'Sci-Fi': 3444,
 'IMAX': 197,
 'Documentary': 5118,
 'War': 1820,
 'Musical': 1113,
 'Western': 1378,
 'Film-Noir': 364,
 '(no genres listed)': 4266}
```

Figure 5.1.4 Count the number of genre

In the above image the first code initiate genre as an empty variable and the next row it count the rows in movies dataset to be counted along with the nested for loop statement to count the genre and the output will be how many movies are 'Adventure', 'Drama', 'Romance', 'Horror', etc. and the movie that have no genres. It is a lot of genres.

The following bar chart best explain the current condition of dataset regarding the genres being the most watched and the least.

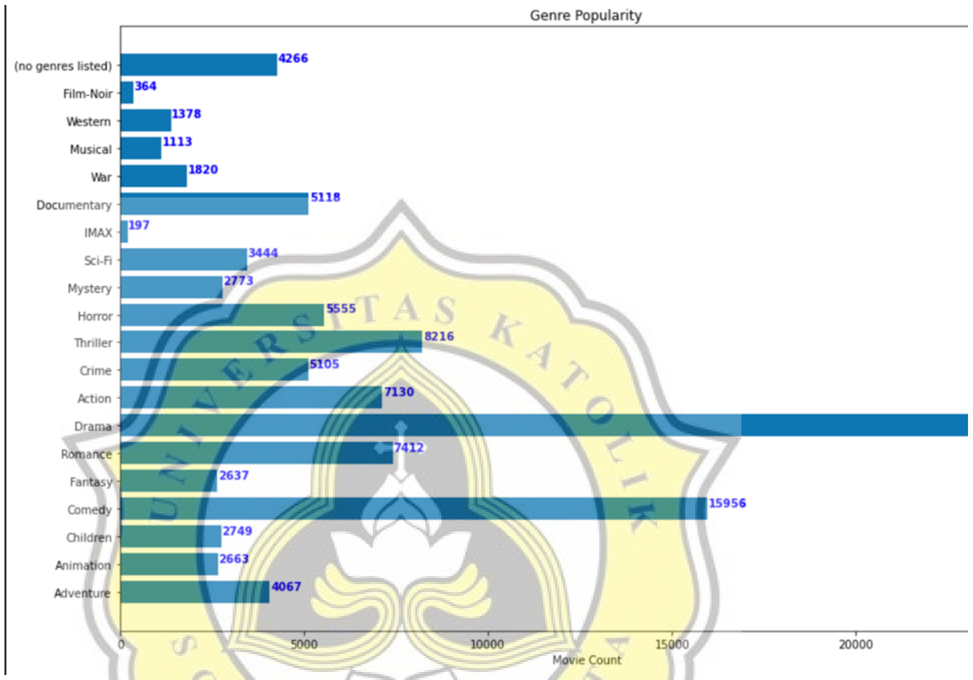
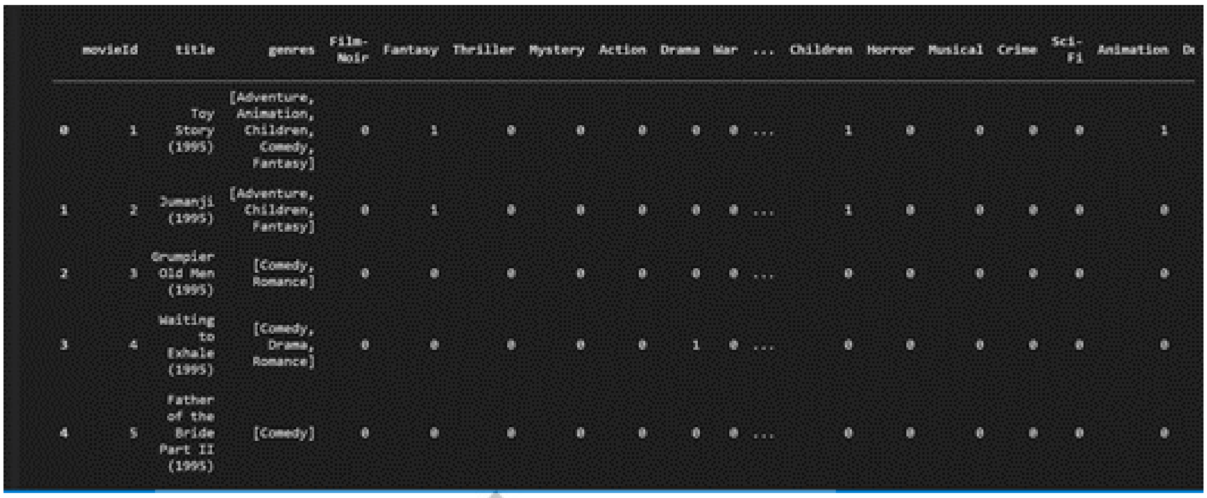


Figure 5.1.5 Genre of watched movies

The most watched movies based on this plotted data is 'Drama', the second favorite is in the place of 'Comedy' and the least favorite genre is IMAX with the score of 197 movie count.

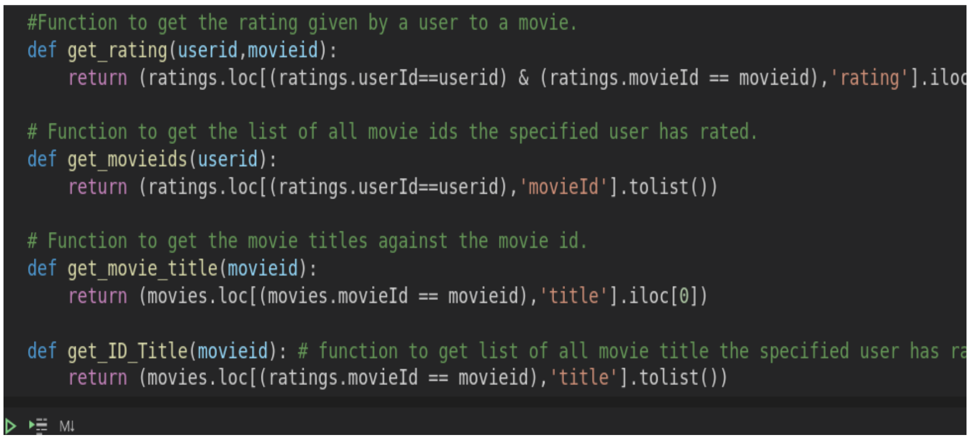
In a Recommender System usually there is a pivot table of movies name and a genre on the top rows to confirm the property of the movie. A pivot table by definition is a grouped values that aggregates the individual items of a more extensive table within one or more discrete categories. The summary can include sums, average, or more statistics.



	movieId	title	genres	Film-Noir	Fantasy	Thriller	Mystery	Action	Drama	War	...	Children	Horror	Musical	Crime	Sci-Fi	Animation	Dc
0	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	0	1	0	0	0	0	0	...	1	0	0	0	0	1	
1	2	Jumanji (1995)	[Adventure, Children, Fantasy]	0	1	0	0	0	0	0	...	1	0	0	0	0	0	
2	3	Grumpier Old Men (1995)	[Comedy, Romance]	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
3	4	Waiting to Exhale (1995)	[Comedy, Drama, Romance]	0	0	0	0	0	1	0	...	0	0	0	0	0	0	
4	5	Father of the Bride Part II (1995)	[Comedy]	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

Figure 5.1.6 Pivot Table Genre

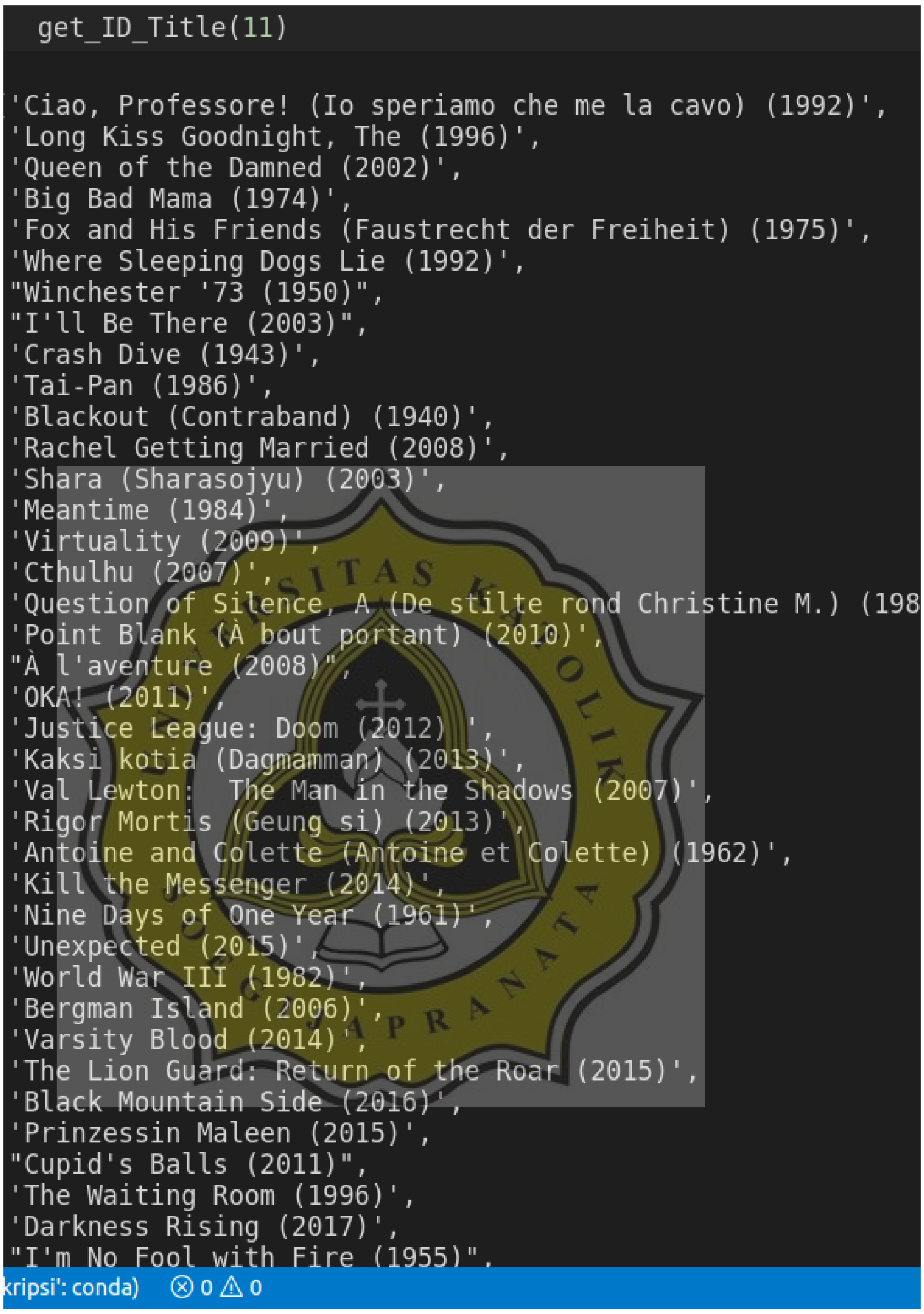
Here we can see how the movie can be confirmed for its genre easily by visually checking is this movie matched with this genre or not for example if Toy Story a funny movie? The answer is yes, then it matched the Comedy Segment, if it is a fantasy movie it will also match the fantasy category along with many more category and the matched category will show a Boolean value of 1 and the unmatched category will show 0 this is the whole picture of a pivot table for genre classification.



```
#Function to get the rating given by a user to a movie.
def get_rating(userid,movieid):
    return (ratings.loc[(ratings.userId==userid) & (ratings.movieId == movieid),'rating'].iloc

# Function to get the list of all movie ids the specified user has rated.
def get_movieids(userid):
    return (ratings.loc[(ratings.userId==userid),'movieId'].tolist())

# Function to get the movie titles against the movie id.
def get_movie_title(movieid):
    return (movies.loc[(movies.movieId == movieid),'title'].iloc[0])

def get_ID_Title(movieid): # function to get list of all movie title the specified user has ra
    return (movies.loc[(ratings.movieId == movieid),'title'].tolist())
```

Figure 5.1.7 Get Movie Title

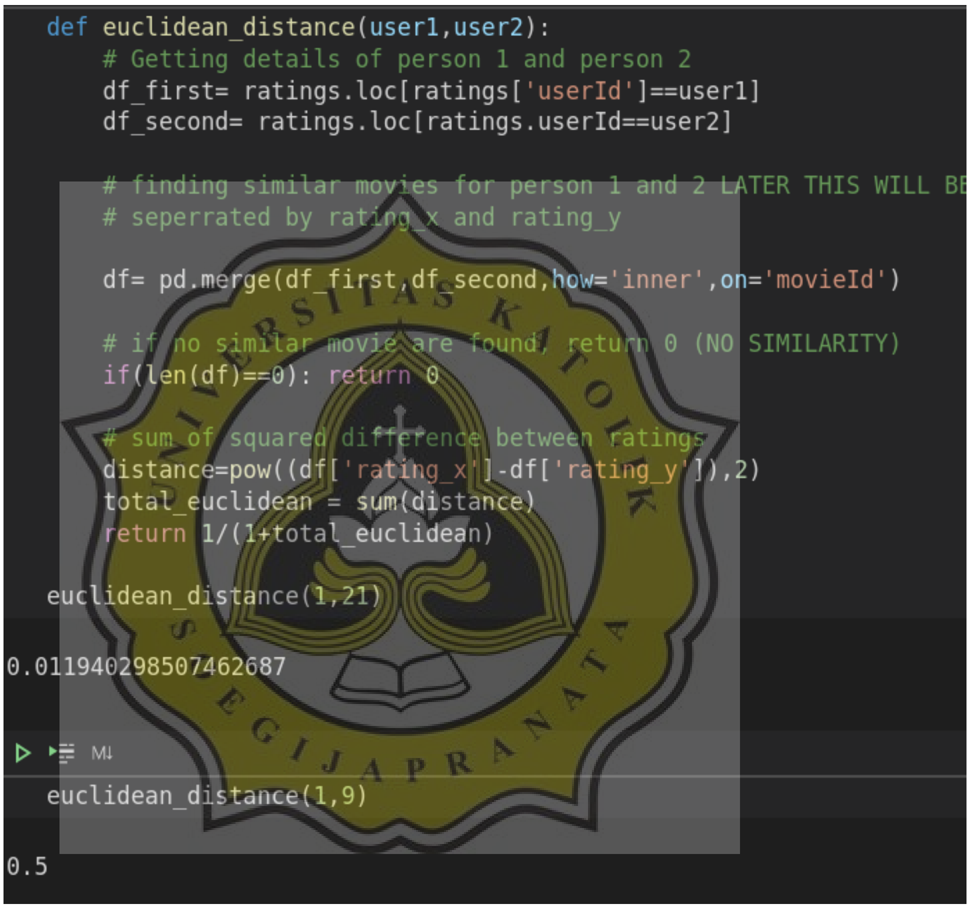
This is a function used to find rating, movieId, and most importantly movie title of the movies that are watched by a specific user.



```
get_ID_Title(11)

'Ciao, Professore! (Io speriamo che me la cavo) (1992)',
'Long Kiss Goodnight, The (1996)',
'Queen of the Damned (2002)',
'Big Bad Mama (1974)',
'Fox and His Friends (Faustrecht der Freiheit) (1975)',
'Where Sleeping Dogs Lie (1992)',
"Winchester '73 (1950)",
"I'll Be There (2003)",
'Crash Dive (1943)',
'Tai-Pan (1986)',
'Blackout (Contraband) (1940)',
'Rachel Getting Married (2008)',
'Shara (Sharasojyu) (2003)',
'Meantime (1984)',
'Virtuality (2009)',
'Cthulhu (2007)',
'Question of Silence, A (De stilte rond Christine M.) (198
'Point Blank (À bout portant) (2010)',
"À l'aventure (2008)",
'OKA! (2011)',
'Justice League: Doom (2012) ',
'Kaksi kotia (Dagmamman) (2013)',
'Val Lewton: The Man in the Shadows (2007)',
'Rigor Mortis (Geung si) (2013)',
'Antoine and Colette (Antoine et Colette) (1962)',
'Kill the Messenger (2014)',
'Nine Days of One Year (1961)',
'Unexpected (2015)',
'World War III (1982)',
'Bergman Island (2006)',
'Varsity Blood (2014)',
'The Lion Guard: Return of the Roar (2015)',
'Black Mountain Side (2016)',
'Prinzessin Maleen (2015)',
"Cupid's Balls (2011)",
'The Waiting Room (1996)',
'Darkness Rising (2017)',
"I'm No Fool with Fire (1955)",
kripsi': conda)   ⓧ 0 ⚠ 0
```

Figure 5.1.8 Movie Title watched by users

This is an example of getting a movie title from userId (11) and seeing what kinds of movie these users have watched or rated that can be used to get information about the recommender system.



```
def euclidean_distance(user1,user2):
    # Getting details of person 1 and person 2
    df_first= ratings.loc[ratings['userId']==user1]
    df_second= ratings.loc[ratings.userId==user2]

    # finding similar movies for person 1 and 2 LATER THIS WILL BE
    # seperrated by rating_x and rating_y

    df= pd.merge(df_first,df_second,how='inner',on='movieId')

    # if no similar movie are found, return 0 (NO SIMILARITY)
    if(len(df)==0): return 0

    # sum of squared difference between ratings
    distance=pow((df['rating_x']-df['rating_y']),2)
    total_euclidean = sum(distance)
    return 1/(1+total_euclidean)

euclidean_distance(1,21)

0.011940298507462687

euclidean_distance(1,9)

0.5
```

Figure 5.1.9 Euclidean Distance

The Figure above show how Euclidean Distance Algorithm is implemented there will be an explanation on how each line of statement really work in the next paragraph.

The first line is defining the function for Euclidean Distance Algorithm. Next part is where its method is used to separate rating into two group that on later writing will be called rating_x and rating_y. This part is where we merge df_first and df_second into a single dataframe for processing on movieId. This part throws a conditional statement stating if there is no similar user's during looping of dataframe then the code can return 0 or continue to run until a value is found. Meanwhile on the next part if the len() or length of dataframe is 0 then the code will return 0. Calculation is done for this part below is a calculation of Euclidean distance that is the sum of squared difference between ratings. More detailed overwiew of the code will be explained below the code

The code below is the implementation code of Euclidean Distance and Manhattan Distance and there will be more explanation below the printed code and some testing.

Hashtags (#) are used for commenting a code in python for debugging / review process in case we forget what the code does.

FORMULA HERE

Euclidean Distance code

```
def euclidean_distance(user1,user2):
    # Getting details of person 1 and person 2
    df_first= ratings.loc[ratings['userId']==user1]
    df_second= ratings.loc[ratings.userId==user2]
    # finding similar movies for person 1 and 2 LATER THIS WILL BE
    # seperrated by rating_x and rating_y

    df= pd.merge(df_first,df_second,how='inner',on='movieId')

    # if no similar movie are found, return 0 (NO SIMILARITY)
    if(len(df)==0): return 0
    # sum of squared difference between ratings
```

```
11.       distance=pow((df['rating_x']-
  df['rating_y']),2)
12.       total_euclidean = sum(distance)
13.       return 1/(1+total_euclidean)
```

Code Explanation Line 1 is defining the function for Euclidean Distance Algorithm. Line 3 and 4 is used to separate rating into two group that on later writing will be called rating_x and rating_y. Line 7 is to merge df_first and df_second into a single dataframe for processing on movieId. Line 8 is a conditional statement stating if there is no similar user's during looping of dataframe then the code can return 0 or continue to run until a value is found. Meanwhile on Line 9 if the len or length of dataframe is 0 then the code will return 0. Line 10 and below is a calculationh of euclidean distance that is the sum of squared difference between ratings. Line 11 define distance and counting rating_x – rating_y and then squared, Line 12 adds the sum of distance and finnaly Line 13 computed a returned result of 1/(1+total_euclidean) this formula is needed to return a result between -1 and 1 but since any value less than 0 is discarded then this calculation is expected to return a value between 0 and 1 and for easeir readabilty we can also say that the distance between user x and user y is between 0 and 100 % but the result will be returned in a decimal value.

Here is the initial test result of the above code, simple testing is needed to make sure the algorithm specified run well with the input and the test result shows some of the best expected result. This result will be compared with manhattan distance and see wheter they are very close, very far, similar or identical or not similar at all.

Expected Output : Between 0 and 1 Float values between userId's INTEGER Values.

Test 1 : **`print(euclidean_distance_score(1,9))`**

Output : `0.5`

Test 2 : `print(euclidean_distance_score(1,310)`

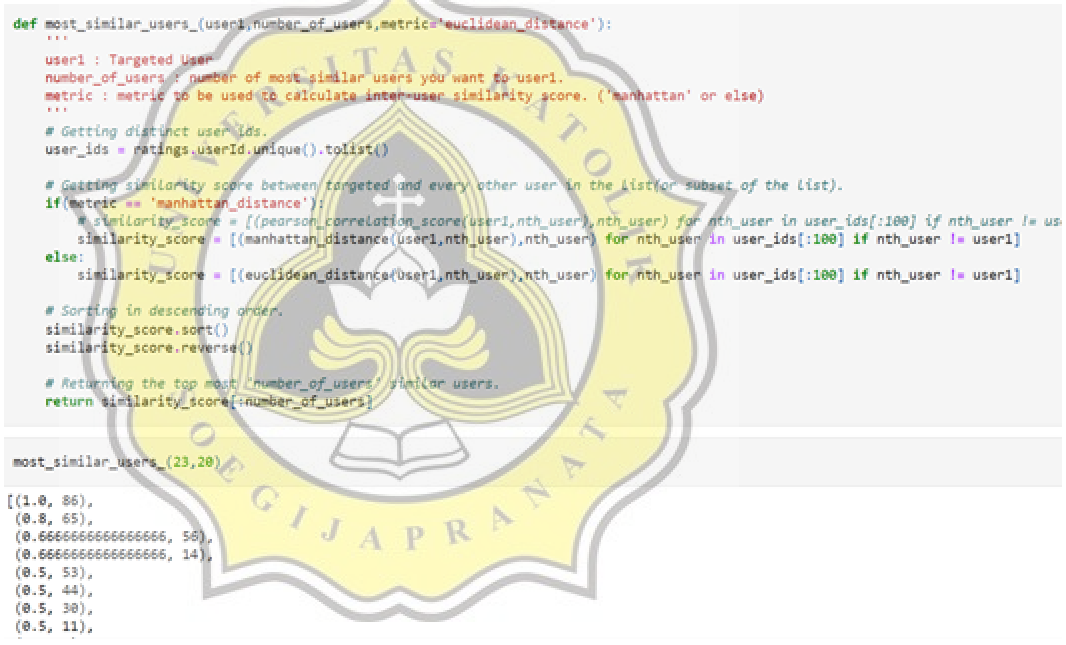
Output : `0.06060606060606061`

Test 3 : `euclidean_distance_score(11,41)`

Output : `0.023121387283236993`

Test 4 : `euclidean_distance_score(61,89)`

Output : `0.13793103448275862`



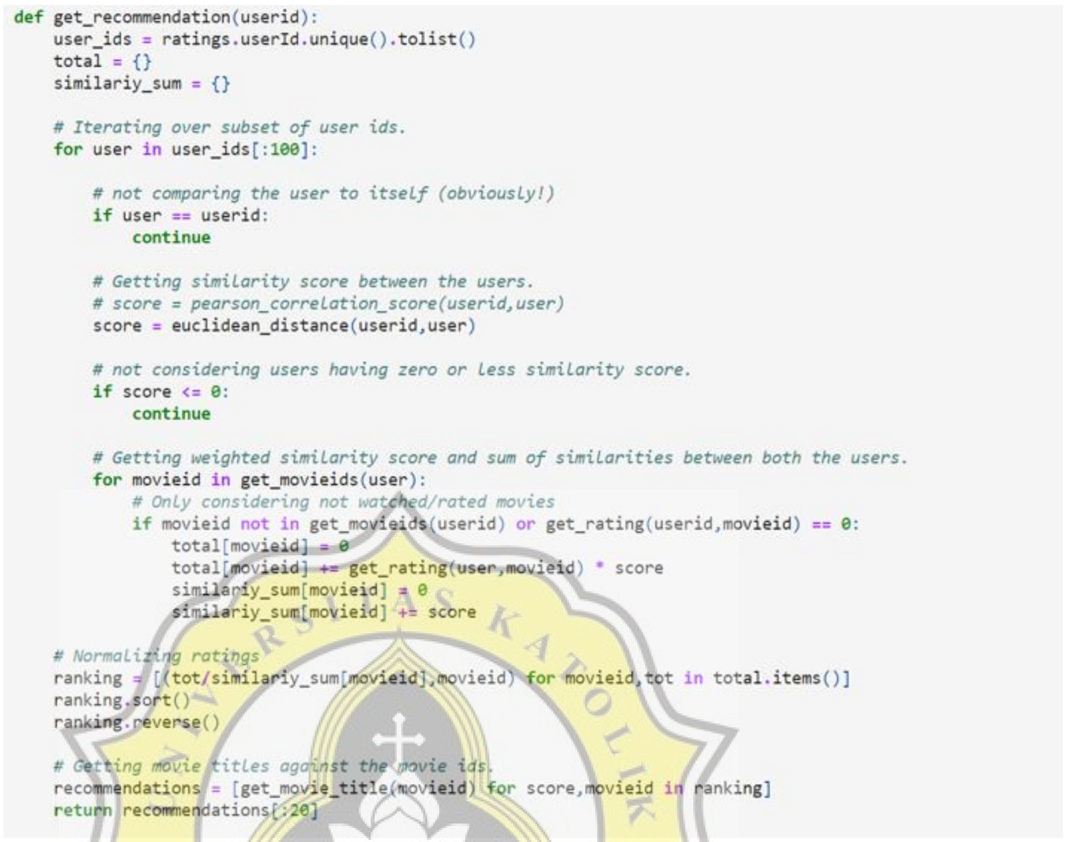
```
def most_similar_users_(user1,number_of_users,metric='euclidean_distance'):
    '''
    user1 : Targeted User
    number_of_users : number of most similar users you want to user1.
    metric : metric to be used to calculate inter-user similarity score. ('manhattan' or else)
    '''
    # Getting distinct user ids.
    user_ids = ratings.userId.unique().tolist()

    # Getting similarity score between targeted and every other user in the list(or subset of the list).
    if(metric == 'manhattan_distance'):
        # similarity_score = [(pearson_correlation_score(user1,nth_user),nth_user) for nth_user in user_ids[:100] if nth_user != us
        similarity_score = [(manhattan_distance(user1,nth_user),nth_user) for nth_user in user_ids[:100] if nth_user != user1]
    else:
        similarity_score = [(euclidean_distance(user1,nth_user),nth_user) for nth_user in user_ids[:100] if nth_user != user1]

    # Sorting in descending order.
    similarity_score.sort()
    similarity_score.reverse()

    # Returning the top most 'number_of_users' similar users.
    return similarity_score[:number_of_users]
```

```
most_similar_users_(23,20)
```

```
[(1.0, 86),
 (0.8, 65),
 (0.6666666666666666, 56),
 (0.6666666666666666, 14),
 (0.5, 53),
 (0.5, 44),
 (0.5, 30),
 (0.5, 11),
```

Figure 5.1.9.a Most Similar User Euclidean Distance

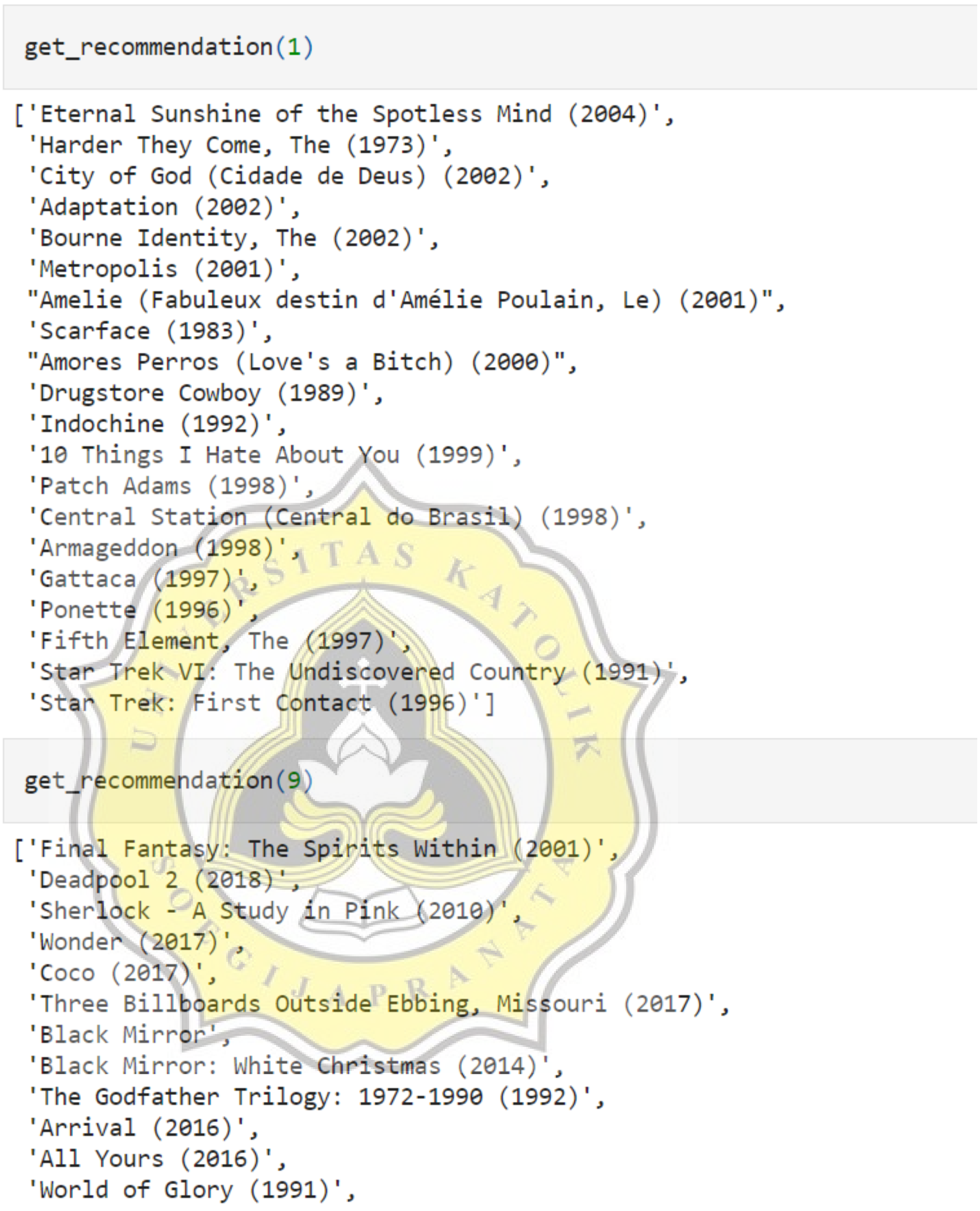
Searching through all of Euclidean distance function the most similar movies. The code runs on the ***else clause*** so this is different than Manhattan metric.



```python
def get_recommendation(userid):
    user_ids = ratings.userId.unique().tolist()
    total = {}
    similariy_sum = {}

    # Iterating over subset of user ids.
    for user in user_ids[:100]:

        # not comparing the user to itself (obviously!)
        if user == userid:
            continue

        # Getting similarity score between the users.
        # score = pearson_correlation_score(userid,user)
        score = euclidean_distance(userid,user)

        # not considering users having zero or less similarity score.
        if score <= 0:
            continue

        # Getting weighted similarity score and sum of similarities between both the users.
        for movieid in get_movieids(user):
            # Only considering not watched/rated movies
            if movieid not in get_movieids(userid) or get_rating(userid,movieid) == 0:
                total[movieid] = 0
                total[movieid] += get_rating(user,movieid) * score
                similariy_sum[movieid] = 0
                similariy_sum[movieid] += score

    # Normalizing ratings
    ranking = [(tot/similariy_sum[movieid],movieid) for movieid,tot in total.items()]
    ranking.sort()
    ranking.reverse()

    # Getting movie titles against the movie ids.
    recommendations = [get_movie_title(movieid) for score,movieid in ranking]
    return recommendations[:20]
```

Figure 5.1.9.b Get Recommendation Euclidean Distance

The code above shows how to iterate through user ID and count the Euclidean distance in order to get the movie recommended appeared in the next dialog box or dataframe.



```
get_recommendation(1)
```

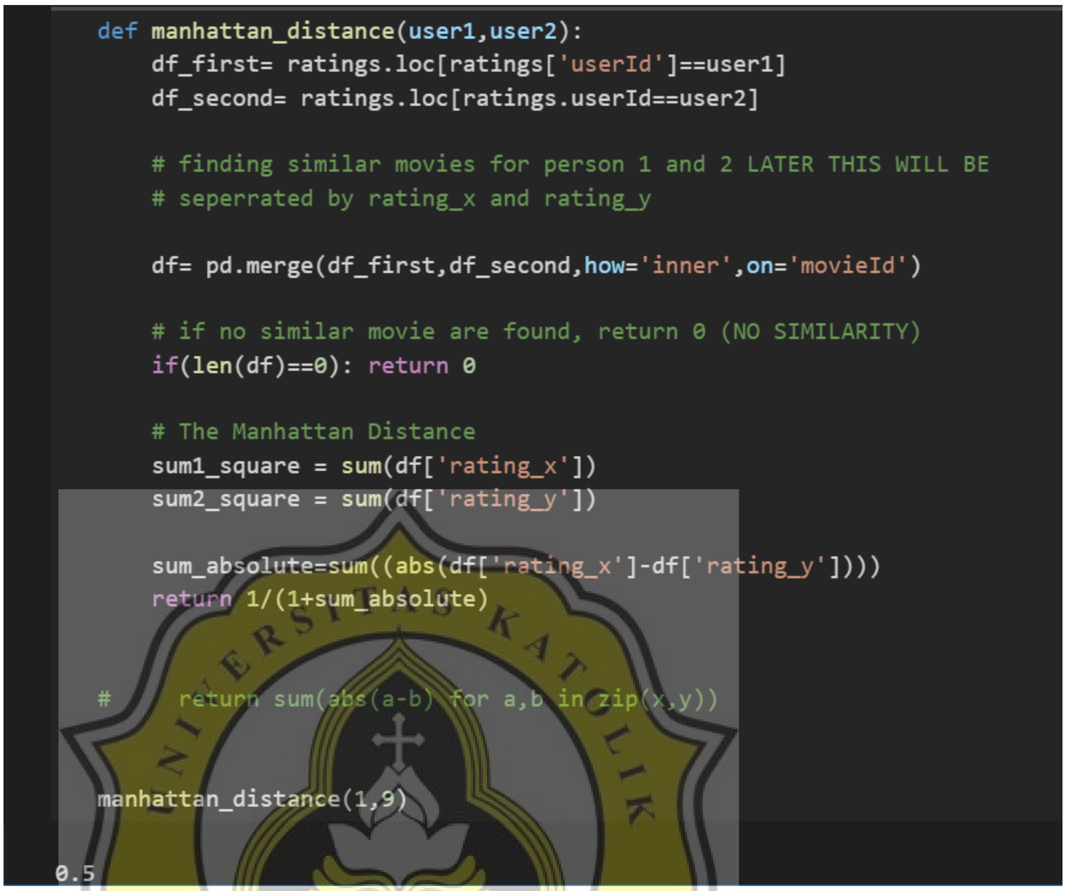
```
['Eternal Sunshine of the Spotless Mind (2004)',
 'Harder They Come, The (1973)',
 'City of God (Cidade de Deus) (2002)',
 'Adaptation (2002)',
 'Bourne Identity, The (2002)',
 'Metropolis (2001)',
 "Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)",
 'Scarface (1983)',
 "Amores Perros (Love's a Bitch) (2000)",
 'Drugstore Cowboy (1989)',
 'Indochine (1992)',
 '10 Things I Hate About You (1999)',
 'Patch Adams (1998)',
 'Central Station (Central do Brasil) (1998)',
 'Armageddon (1998)',
 'Gattaca (1997)',
 'Ponette (1996)',
 'Fifth Element, The (1997)',
 'Star Trek VI: The Undiscovered Country (1991)',
 'Star Trek: First Contact (1996)']
```

```
get_recommendation(9)
```

```
['Final Fantasy: The Spirits Within (2001)',
 'Deadpool 2 (2018)',
 'Sherlock - A Study in Pink (2010)',
 'Wonder (2017)',
 'Coco (2017)',
 'Three Billboards Outside Ebbing, Missouri (2017)',
 'Black Mirror',
 'Black Mirror: White Christmas (2014)',
 'The Godfather Trilogy: 1972-1990 (1992)',
 'Arrival (2016)',
 'All Yours (2016)',
 'World of Glory (1991)',
```

Figure 5.1.9.c Recommended movie list Euclidean

The above code is an example of how the movie is recommended to the specific userID.



```
def manhattan_distance(user1,user2):
    df_first= ratings.loc[ratings['userId']==user1]
    df_second= ratings.loc[ratings.userId==user2]

    # finding similar movies for person 1 and 2 LATER THIS WILL BE
    # seperrated by rating_x and rating_y

    df= pd.merge(df_first,df_second,how='inner',on='movieId')

    # if no similar movie are found, return 0 (NO SIMILARITY)
    if(len(df)==0): return 0

    # The Manhattan Distance
    sum1_square = sum(df['rating_x'])
    sum2_square = sum(df['rating_y'])

    sum_absolute=sum((abs(df['rating_x']-df['rating_y'])))
    return 1/(1+sum_absolute)


#     return sum(abs(a-b) for a,b in zip(x,y))


manhattan_distance(1,9)

0.5
```

Figure 5.1.10 Manhattan Distance

The Figure above show how Manhattan Distance Algorithm is implemented there will be an explanation on how each line of statement really work in the next paragraph.

$$d1(p,q) = ||p - q||i = \sum_{i=1}^{n} |pi - qi|$$

Illustration 5.1.11 Manhattan DIstance Formula

The code for Manhattan Distance

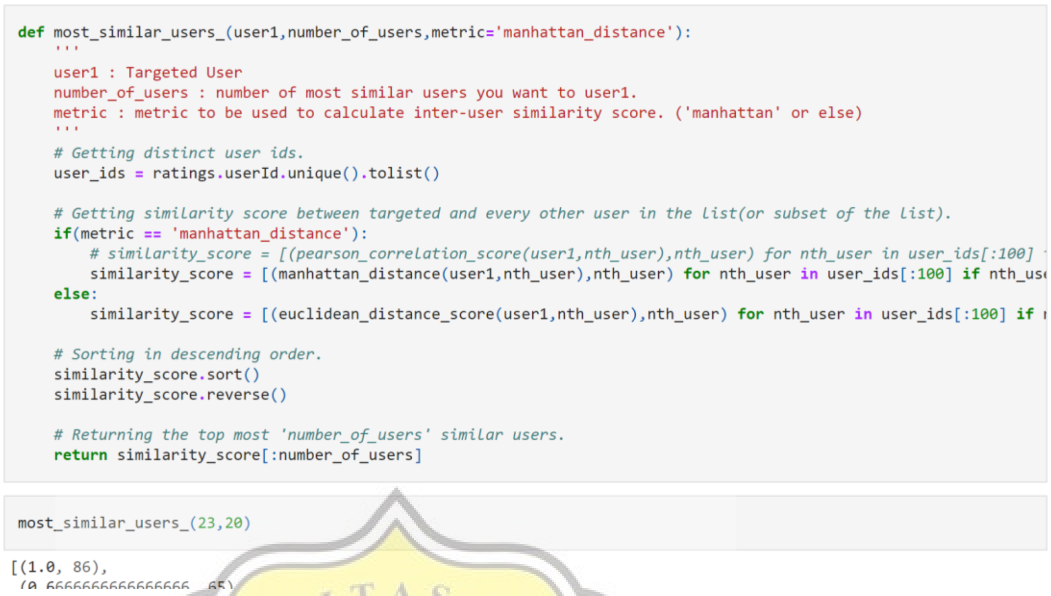
```
def manhattan_distance(user1,user2):
    df_first= ratings.loc[ratings['userId']==user1]
    df_second= ratings.loc[ratings.userId==user2]
    # finding similar movies for user1 and user2 LATER THIS WILL BE
    # seperrated by rating_x and rating_y
pd.merge(df_first,df_second,how='inner',on='movieId')
     # if no similar movie are found, return 0 (NO SIMILARITY)
    if(len(df)==0): return 0
    # The Manhattan Distance
      sum1_square = sum(df['rating_x'])
      sum2_square = sum(df['rating_y'])
  sum_absolute=sum((abs(df['rating_x']-df['rating_y'])))
      return 1/(1+sum_absolute) #       returning a value between 0 1 and 1
 manhattan_score(1,21) # comparison between user 1 and 21
```

Code Explanation Line 1 is defining the function for Manhattan Distance Algorithm. Line 2 and 3 is used to separate rating into two group that on later writing will be called rating_x and rating_y. Line 6 is to merge df_first and df_second into a single dataframe for processing on movieId. Line 8 is a conditional statement stating if there is no similar user's during looping of dataframe then the code can return 0 or continue to run until a value is found. Meanwhile on Line 9 if the len or length of dataframe is 0 then the code will return 0. Line 9 is the actual Manhattan Distance code evaluation. Line 10 and 11 is to count the sum of rating_x and rating_y somehow the code won't execute without this declaration of sum. Line 12 is used to count the absolute value of df['rating_x'] - df['rating_y']. Line 13 is used to make sure whatever the value is it will be between 0 and 1 because absolute keyword already in place.

Here is the initial test result of the above code, simple testing is needed to make sure the algorithm specified run well with the input and the test result shows some of the best expected result. This result will be compared with manhattan distance and see wheter they are very close, very far, similar or identical or not similar at all.

Expected Output : Between 0 and 1 float value

Output Line 16 : 0.020202020202020204



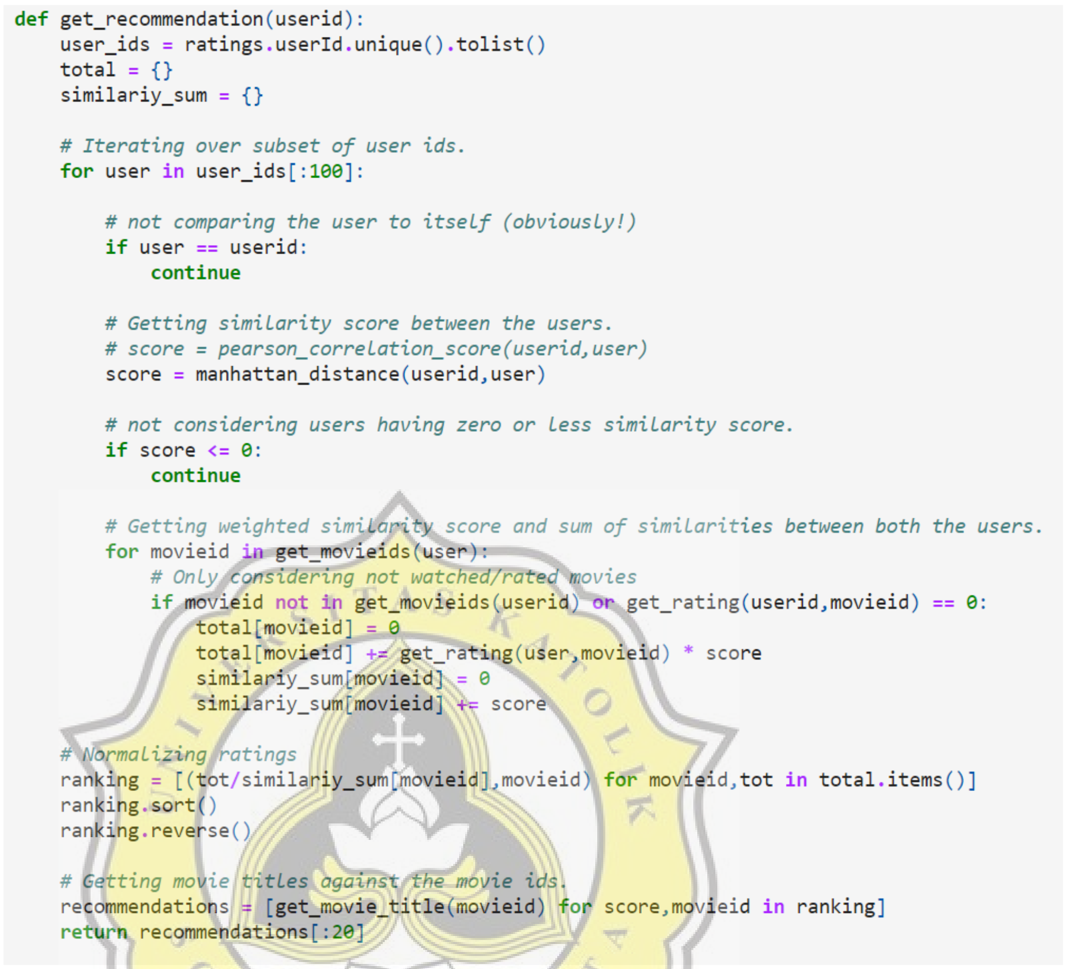
```
def most_similar_users_(user1,number_of_users,metric='manhattan_distance'):
    '''
    user1 : Targeted User
    number_of_users : number of most similar users you want to user1.
    metric : metric to be used to calculate inter-user similarity score. ('manhattan' or else)
    '''
    # Getting distinct user ids.
    user_ids = ratings.userId.unique().tolist()

    # Getting similarity score between targeted and every other user in the list(or subset of the list).
    if(metric == 'manhattan_distance'):
        # similarity_score = [(pearson_correlation_score(user1,nth_user),nth_user) for nth_user in user_ids[:100]
        similarity_score = [(manhattan_distance(user1,nth_user),nth_user) for nth_user in user_ids[:100] if nth_us
    else:
        similarity_score = [(euclidean_distance_score(user1,nth_user),nth_user) for nth_user in user_ids[:100] if

    # Sorting in descending order.
    similarity_score.sort()
    similarity_score.reverse()

    # Returning the top most 'number_of_users' similar users.
    return similarity_score[:number_of_users]
```

```
most_similar_users_(23,20)
```

```
[(1.0, 86),
 (0.6666666666666666, 65)
```

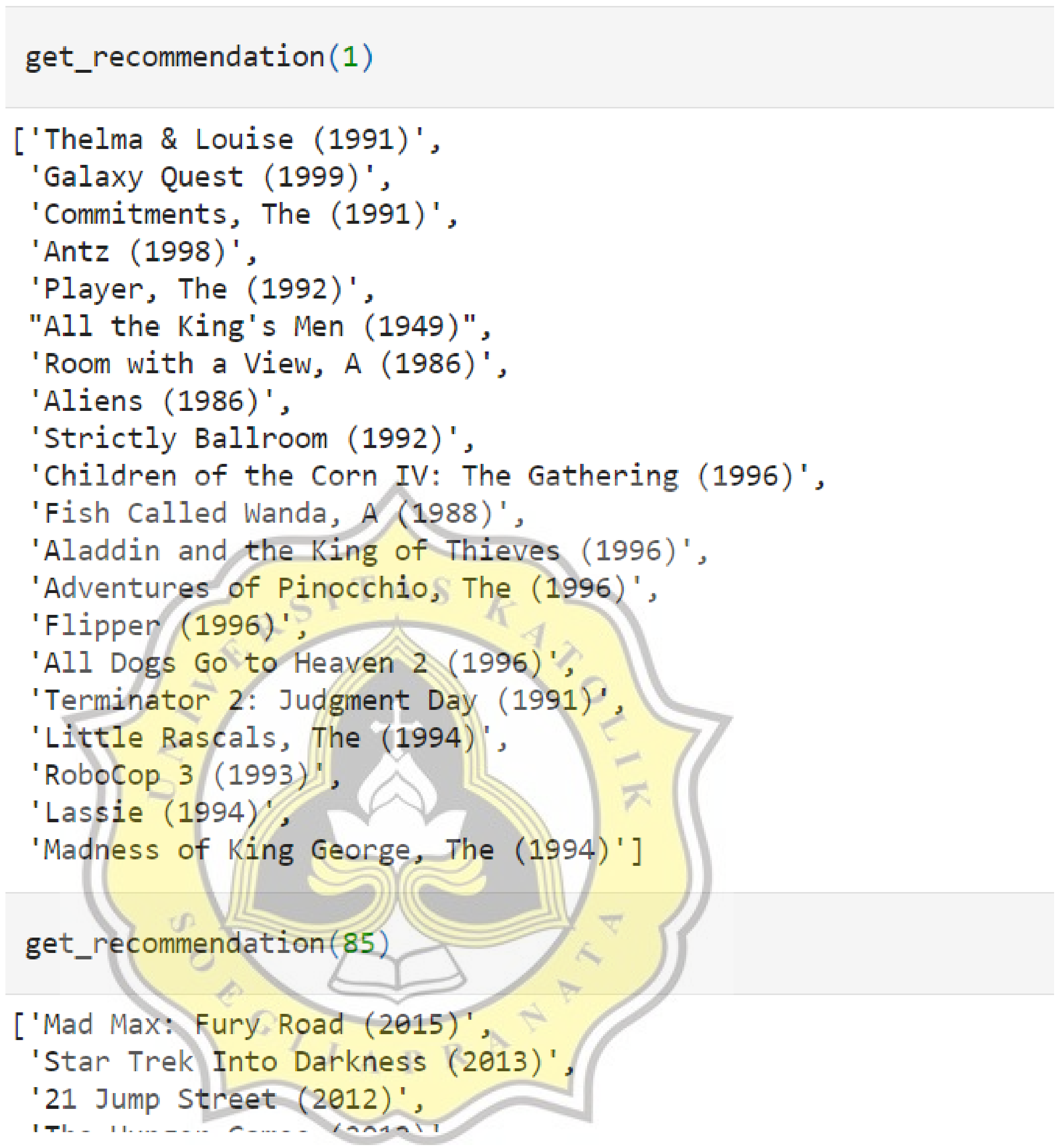
Figure 5.1.11 Most Similar User Manhattan

Searching through all of Manhattan_distance function the most similar movies.



```python
def get_recommendation(userid):
    user_ids = ratings.userId.unique().tolist()
    total = {}
    similariy_sum = {}

    # Iterating over subset of user ids.
    for user in user_ids[:100]:

        # not comparing the user to itself (obviously!)
        if user == userid:
            continue

        # Getting similarity score between the users.
        # score = pearson_correlation_score(userid,user)
        score = manhattan_distance(userid,user)

        # not considering users having zero or less similarity score.
        if score <= 0:
            continue

        # Getting weighted similarity score and sum of similarities between both the users.
        for movieid in get_movieids(user):
            # Only considering not watched/rated movies
            if movieid not in get_movieids(userid) or get_rating(userid,movieid) == 0:
                total[movieid] = 0
                total[movieid] += get_rating(user,movieid) * score
                similariy_sum[movieid] = 0
                similariy_sum[movieid] += score

    # Normalizing ratings
    ranking = [(tot/similariy_sum[movieid],movieid) for movieid,tot in total.items()]
    ranking.sort()
    ranking.reverse()

    # Getting movie titles against the movie ids.
    recommendations = [get_movie_title(movieid) for score,movieid in ranking]
    return recommendations[:20]
```

Figure 5.1.12 get recommendation



```
get_recommendation(1)
```

```
['Thelma & Louise (1991)',
 'Galaxy Quest (1999)',
 'Commitments, The (1991)',
 'Antz (1998)',
 'Player, The (1992)',
 "All the King's Men (1949)",
 'Room with a View, A (1986)',
 'Aliens (1986)',
 'Strictly Ballroom (1992)',
 'Children of the Corn IV: The Gathering (1996)',
 'Fish Called Wanda, A (1988)',
 'Aladdin and the King of Thieves (1996)',
 'Adventures of Pinocchio, The (1996)',
 'Flipper (1996)',
 'All Dogs Go to Heaven 2 (1996)',
 'Terminator 2: Judgment Day (1991)',
 'Little Rascals, The (1994)',
 'RoboCop 3 (1993)',
 'Lassie (1994)',
 'Madness of King George, The (1994)']
```

```
get_recommendation(85)
```

```
['Mad Max: Fury Road (2015)',
 'Star Trek Into Darkness (2013)',
 '21 Jump Street (2012)',
```

Figure 5.1.13 recommended movie returned manhattan

5.2 Testing

Testing for this program is making sure the code supplied is working as inteded, and also a test of the algorithm's performance, score based on the data structure supplied which is the movilens dataset. Testing also conduct an analyzsis of whether a userID can return a list of recommended movie and how they correlate with each other in terms of distance such as euclidean distances and the proposed manhattan distance calculation, see how far the a particular user is from the other supplied user.

Testing the Algorithm with Manhattan distance to see the similarity between user x and user y, x and y will be the UserID to be identified during the execution of the program.

Here is how the program are being tested. Firstly, I will demonstate how Euclidean Distance performs for 2 user's then Secondly I will demonstrate how Manhattan Distance will returns an output the same way as Euclidean Distance score, is it the same, far away or very close to each other *similarity is close enough*. The Final Step would be to check which movie is recommended to which users for example user 1 may get a recommendation of 'Finding Dory' while user 2 may get 'Ice Age' this is done in the getRecommendation function at the end of the program.

The expected result will be a range of 1 and 0 as mentioned in Chapter 3 Activation Function. This formula is needed to compare how the two algorithm will perform and their score, if I chose to not include this function in the code will result a confusion on how the score behave, that is why a standardized metric of activation function in this case is a heuristic function to return a range 0 until 1 is needed. In my code activation function for each algorithm is represented as **return 1 / (1 + sum_of_squares).**

Here is the tesing result of the code.

Test 1: `print(euclidean_distance_score(1,9))`

Output:
```
0.5
```

Explanation: Here we see that the euclidean distance between user 1 and user 9 is 0.5 meaning they are 50% similar

Test 2: euclidean_distance(1,21)

Output: 0.011940298507462687

Explanation: Here we see that the euclidean distance between user 1 and user 21 is 0.0119 meaning they are 1% similarity or not similar at all.

Test 3: euclidean_distance(1,310)

Output: 0.06060606060606061

Explanation: Here we see that the euclidean distance between user 1 and user 310 is 0.0606 meaning they are 6% similar in distance.

Test 4: euclidean_distance(11,41)

Output: 0.023121387283236993

Explanation: Here we see that the euclidean distance between user 11 and user 41 is 0.0231 meaning they are 2% similar

Test 5: euclidean_distance(61,89)

Output: 0.13793103448275862

Explanation: Here we see that the euclidean distance between user 61 and user 89 is 0.1379 meaning they are 13% similar

Test 6: euclidean_distance(23,86)

Output: 1.0

Explanation: Here we see that the euclidean distance between user 23 and user 86 is 1.0 meaning they are 100% similar meaning they have the same rating type of movie most of the time that they can score 100% similarity score

Test 7: euclidean_distance(1,85)

Output: 1.0

Explanation: Here we see that the euclidean distance between user 1 and user 85 is 1.0 meaning they are 100% similar meaning they have the same rating type of movie most of the time that they are able to score 100% similarity score

Test 8: euclidean_distance(1,77)

Output: 1.0

Explanation: Here we see that the euclidean distance between user 1 and user 77 is 1.0 meaning they are 100% similar meaning they have the same rating type of movie most of the time that they are able to score 100% similarity score

Test 9: euclidean_distance(1,53)

Output: 0.5

Explanation: Here we see that the euclidean distance between user 1 and user 53 is 0.5 meaning they are 50% similar

Test 10: euclidean_distance(1,44)

Output: 0.06666666666666667

Explanation: Here we see that the euclidean distance between user 1 and user 44 is 0.0666 meaning they are about 6% similar

Test 1: manhattan_distance(1,9)

Output: 0.5

Explanation: Here we see that the manhattan distance between user 1 and user 9 is 0.5 meaning they are 50% similar

Test 2: manhattan_distance(1,21)

Output: 0.020202020202020204

Explanation: Here we see that the manhattan distance between user 1 and user 21 is 0.0202 meaning they are 2% similarity or not similar at all a little higher score than Euclidean distance.

Test 3: manhattan_distance(1,310)

Output: 0.1

Explanation: Here we see that the manhattan distance between user 1 and user 310 is 0.1 meaning they are 10% similar in distance.

Test 4: manhattan_distance(11,41)

Output: 0.06060606060606061

Explanation: Here we see that the manhattan distance between user 11 and user 41 is 0.0606 meaning they are 6% similar

Test 5: manhattan_distance(61,89)

Output: 0.2857142857142857

Explanation: Here we see that the manhattan distance between user 61 and user 89 is 0.2857 meaning they are 28% similar, higher score is achieved on Manhattan distance metric.

Test 6: manhattan_distance(23,86)

Output: 1.0

Explanation: Here we see that the manhattan distance between user 23 and user 86 is 1.0 meaning they are 100% similar meaning they have the same rating type

of movie most of the time that they are able to score 100% similarity score

Test 7: manhattan_distance(1,85)

Output: 1.0

Explanation: Here we see that the manhattan distance between user 1 and user 85 is 1.0 meaning they are 100% similar meaning they have the same rating type of movie most of the time that they are able to score 100% similarity score

Test 8: manhattan_distance(1,77)

Output: 1.0

Explanation: Here we see that the manhattan distance between user 1 and user 77 is 1.0 meaning they are 100% similar meaning they have the same rating type of movie most of the time that they are able to score 100% similarity score

Test 9: manhattan_distance(1,53)

Output: 0.5

Explanation: Here we see that the manhattan distance between user 1 and user 53 is 0.5 meaning they are 50% similar

Test 10: manhattan_distance(1,44)

```
Output: 0.09090909090909091
```

Explanation: Here we see that the manhattan distance between user 1 and user 44 is 0.0909 meaning they are 9% similar

Table 5.2 Comparison Table of 100,836 records

Comparison of distance metric		
User ID	***Euclidean Distance***	***Manhattan Distance***
User 1 & User 9	0.5	0.5
User 1 & User 21	0.011940298507462687	0.13793103448275862
User 1 & User 310	0.06060606060606061	0.1
User 11 & User 41	0.023121387283236993	0.06060606060606061
User 61 & User 89	0.13793103448275862	0.2857142857142857
User 23 & User 86	1. 0	1.0
User 1 & User 85	1.0	1.0
User 1 & User 77	1.0	1.0
User 1 & User 53	0.5	0.5
User 1 & User 44	0.06666666666666667	0.09090909090909091

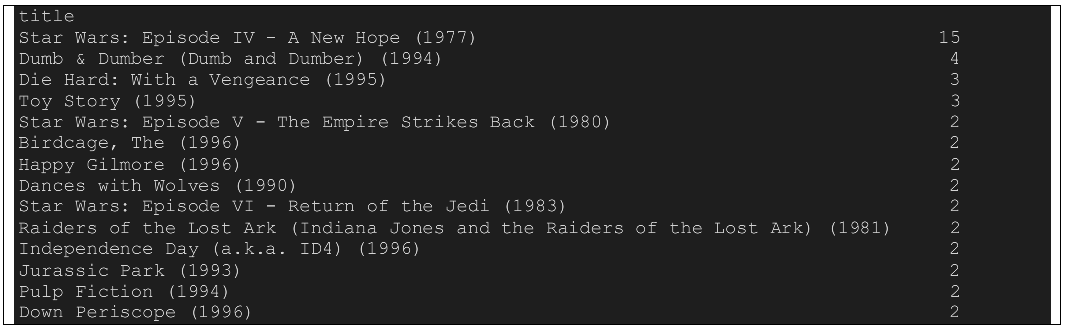
The above table is the same record mentioned in Chapter 4 regarding comparison of a distance between 2 user's in 2 different metric and with that many record it shows that Manhattan distance produce higher score than Euclidean distance.

Below here I will compare the algorithm with fewer ratings at about 200 record and will make a statement on whether Manhattan is better or Euclidean perform better result or returned a higher score.

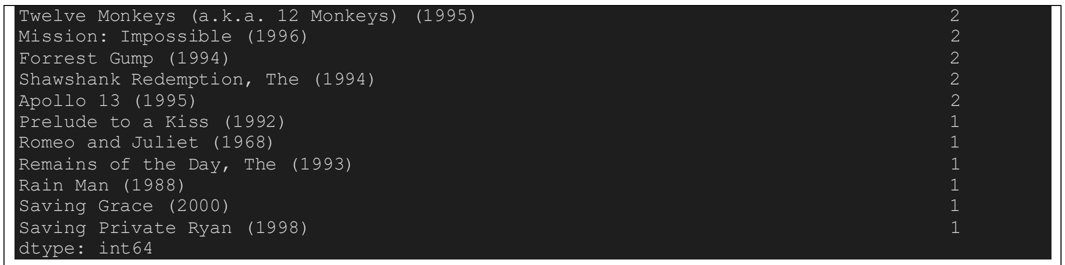
Comparison of distance metric		
User ID	***Euclidean Distance***	***Manhattan Distance***
User 1 & User 85	0	0
User 1 & User 21	0	0
User 85 & User 77	0	0
User 11 & User 41	0	0
User 1 & User 77	1.0	1.0
User 1 & User 44	0	0
User 1 & User 114	0.076923076923077	0.4

The above result is coded in a file named Skripsi_Euclidean1.ipynb and Sripsi_Manhattan1.ipynb in this file the most rated movies from the narrowed down version of this ratings and the movie that rated the most is “Star Wars: Episode IV – A New Hope (1997)” with the ratings given by 15 users.

Table 3 Most rated movies including Star Wars:Episode IV - A New Hope 1997



```
title
Star Wars: Episode IV - A New Hope (1977)                                         15
Dumb & Dumber (Dumb and Dumber) (1994)                                             4
Die Hard: With a Vengeance (1995)                                                  3
Toy Story (1995)                                                                   3
Star Wars: Episode V - The Empire Strikes Back (1980)                              2
Birdcage, The (1996)                                                               2
Happy Gilmore (1996)                                                               2
Dances with Wolves (1990)                                                          2
Star Wars: Episode VI - Return of the Jedi (1983)                                  2
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)     2
Independence Day (a.k.a. ID4) (1996)                                               2
Jurassic Park (1993)                                                               2
Pulp Fiction (1994)                                                                2
Down Periscope (1996)                                                              2
```



```
Twelve Monkeys (a.k.a. 12 Monkeys) (1995)                                     2
Mission: Impossible (1996)                                                    2
Forrest Gump (1994)                                                           2
Shawshank Redemption, The (1994)                                              2
Apollo 13 (1995)                                                              2
Prelude to a Kiss (1992)                                                      1
Romeo and Juliet (1968)                                                       1
Remains of the Day, The (1993)                                                1
Rain Man (1988)                                                               1
Saving Grace (2000)                                                           1
Saving Private Ryan (1998)                                                    1
dtype: int64
```

This is a comparison of Euclidean distance and Manhattan distance without star wars : Episode IV – A New Hope (1997)

Comparison of distance metric		
User ID	***Euclidean Distance***	***Manhattan Distance***
User 1 & User 85	0	0
User 1 & User 21	0	0
User 85 & User 77	0	0
User 11 & User 41	0	0
User 1 & User 77	0.5	0.5
User 1 & User 44	0	0
User 1 & User 114	0.2857142857142857	0.3333333333333333

In the above table we can see that in terms of accuracy the Manhattan distance performs significantly better and produce higher result even though some user similarity become 0 when Star Wars : Episode IV – A New Hope (1997) is removed and the ratings changed to other movies recorded in the dataframe.