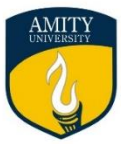

AMITY UNIVERSITY
UTTAR PRADESH

Major Project Report on

Simple GUI for Movie Recommendation System Using Machine Learning

Submitted to

Amity Institute of Information Technology In partial fulfilment of the requirements for the award of the degree of

Master of Computer Application

Submitted To

Dr Ajay Rana

Director

Submitted By

Priyaank Sinha

A1000718022

2018 – 2021

DECLARATION BY STUDENT

I Priyaank Sinha, student of MCA hereby declare that the Seminar titled "**Simple GUI for Movie Recommendation System Using Machine Learning**" which is submitted by me to Dr Ajay Rana, Amity Institute of Information Technology, Amity University Uttar Pradesh, Noida, in partial fulfilment of the requirement for the award of the degree of master of computer applications, has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the Dissertation / Project report other than brief excerpts requiring only proper acknowledgement in scholarly writing and all such use is acknowledged.

Place: Noida

Priyaank Sinha

A1000718022

Semester: MCA 5th

GUIDE CERTIFICATE

I hereby certify that the Seminar Report by Priyaank Sinha, student of MCA 5th (Enrolment No: A1000718022) with the title "**Simple GUI for Movie Recommendation System Using Machine Learning**" which is submitted to Amity Institute of Information Technology, Amity University Uttar Pradesh, Noida in partial fulfilment of the requirement for the award of the degree of master of computer applications is an original contribution with existing knowledge and faithful record of work carried out by him/her under my guidance and supervision and to the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Noida

Date:

Dr Ajay Rana

Director

Amity Institute of Information Technology
AUUP, Noida

ACKNOWLEDGEMENT

It is a high privilege for me to express my deep sense of gratitude to those entire faculty Members who helped me in the completion of the project, especially my internal guide Dr Ajay Rana who was always there at the hour of need.

My special thanks to particular faculty members and Batch mate of Amity Institute of Information Technology, Amity University, Uttar Pradesh for helping me in the completion of project work and its report submission.

Priyaank Sinha

A1000718022

AMITY UNIVERSITY

-----UTTAR PRADESH-----

Amity Institute of Information Technology

Major Project

Student Name	PRIYAANK SINHA
Enrollment No	A1000718022
Programme	Master of Computer Applications
Company's Name and Address	AIIT Sector 125, amity university Noida 250004
Industry Guide	
Name	
Designation	
Contact Number	
	Ph.(O) : **(R) :**
	Mobile :
	Fax :
	E-mail : aiit@amity.edu

Project Information

1) Project Duration : **(112 Days)**

a) Date of Summer Internship commencement **(16/12/2020)**

a) Date of Summer Internship Completion **(07/04/2021)**

2) Topic

Simple GUI For Movie Recommendation System Using Machine Learning

3) Project Objective

Try to improve the movie recommender system used by many companies

4) Methodology to be adopted

Data gathering -> data cleaning -> data visualization -> Machine Learning Algorithms for recommending movies

5) **Brief Summery of project(*to be duly certified by the industry guide*)**

Recommendation systems that are in use these days are: 1) Content-Based Recommendation system 2) Collaborative Recommendation system With the help of which I am going to make the user interface and try to merge these two systems to make a new hybrid recommendation system which will predict and recommend movies more accurately.

Signature (Student)	Signature (Industry Guide)	Signature (Faculty Guide)

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The overwhelming increase in Data in today's world demands more experts to handle this massive amount of data in all industries. It stands out that the companies need more Data Scientists and Data Engineers to design and code the algorithms. Having said that, we often confuse ourselves in the responsibilities shared between these two groups of experts. "*Data Scientists*" are the ones who primarily design algorithms that can serve as the foundation for different prediction models such as *Neural Network, Linear and Logistic regression*, etc. However, "*Data Engineers*" are the ones who code using the predefined prediction models and try to increase the accuracy in the results and sometimes need to create recommendation systems as per the requirement. This clearly shows that these two groups of experts i.e. Data Scientists and Data Engineers works in perfect harmony side-by-side to create impressive solutions to handle the ginormous amount of data produced each day.

This project provides a thorough walk-through of the steps required for developing a *hybrid algorithm* (which combines the concept of *content-based and collaborative* recommender systems) to create a successful movie recommender system (Using *Natural Language Processing*, a field in Machine Learning). This effort of making a hybrid algorithm will predict and recommend movies more accurately.

CHAPTER 1: PROBLEM STATEMENT

Problem: To generate the algorithm for the movie recommender system and possibly make a simple GUI for the same.

Explanation: The demand for Machine Learning is remarkably high. This project is an attempt to develop the hybrid model by combining the concepts of content-based and collaborative recommender system. For exploring the dataset, we implemented the visualisation techniques.

A) Pre-requisites:

- ✓ **Tools:** Jupiter Notebook (for visualization) and Geany IDE
- ✓ **Language:** Python Programming

B) Source of Dataset:

- ➢ https://grouplens.org/datasets/movielens/25m/

CHAPTER 2: INTRODUCTION

In today's era, one must have experienced Data Science and Machine Learning in several forms. Have anyone ever thought, from search engines to Global Positioning System (GPS); everyone is communicating with data science and machine learning tools? It is one of the powerful sciences in the world of technologies. At this point, one might be suffering from a question that, what a *Data Science* is? The fusion of multiple tools, machine learning algorithms and principles that are required for investigating meaningful patterns in the raw data, is *Data Science*.

With time, Natural Language Processing, a field in machine learning is becoming a critical tool for predicting and analysing data. For that reason, machine learning plays a very important role in breaking and extracting useful data from big data that can be used to develop search engines, recommender systems, etc. To understand these concepts more deeply, let us first go through the basic ideas and technical aspects of data science used in this project.

2.1 Data Exploration and Visualization

It is the inaugural step in data science, where the data engineer explores the data set in an unstructured way to divulge the initial patterns, characteristics and points of interest. Moreover, we can utilize **data visualization** techniques like *histograms, bar graphs, pie charts, etc.* to scrutinize the data with more accuracy. Data visualization is a crucial step for any data engineer to extract information that can be easily missed while scrutinizing a large amount of data.

2.2 Data Cleaning

The process of preparing the data for analysis by eliminating or modifying irrelevant, duplicated, incorrect, or improperly formatted (i.e.; NULL or VOID spaces) in data. Data cleaning is not all about removing data but it is a way to maximize the accuracy of the dataset without deleting crucial information. While going through the string data it is very important for the data engineer, to either remove or replace the duplicity (if any).

2.3 Natural Language Processing

Natural Language Processing (**NLP**) is a branch of artificial intelligence or a field in machine learning that provides an ability to a machine to understand human written texts or speech. Some of its tasks are sentimental analysis, named entity recognition, and many more. It helps computers to understand, manipulate and interpret human language.

2.4 Fuzzywuzzy

Fuzzywuzzy is a python library, which works on the concept of *Levenshtein Distance* to calculate the differences between the sequences and patterns that were developed while working on the string data. The concept of fuzzy-wuzzy came from the concept of Fuzzy Logic.

2.5 EasyGUI

EasyGUI is a module in python to develop very simple graphical user interface programming in python. This *EasyGUI* is not similar to other GUI libraries as it is not event-driven instead all the instructions are invoked by just using function calls.

2.6 Complexity

An algorithm complexity can be defined as the analysis of the performance of an algorithm in terms of time or space of the given size "n". The main consideration while developing an algorithm for this project is *Time Complexity*. In simple words, we can say that algorithm complexity measures the number of steps required by the algorithm to solve the given problem.

CHAPTER 3: BACKGROUND

This chapter comprises detailed information on all the important technical aspects, which are the pillars of this project.

3.1 Machine Learning

Machine Learning is a science that allows computers to proceed without explicitly programmed. The major applications of ML are *supervised learning* and *Unsupervised learning*. In unsupervised learning, unlabelled data was fed into a training algorithm to discover relationships and patterns between the data attributes. While supervised learning is human-labelled data.

ML consists of multiple classification algorithms such as:

- Linear and Logistic Regression
- Neural Networks
- Support Vector Machine (SVM)
- K-Nearest Neighbour (KNN)
- Random Forest and many more.

In this project, we are using the field of Machine learning that is, *Natural Language Processing (NLP)* to create an algorithm for a movie recommendation system.

3.2 Natural Language Processing

Natural Language Processing (**NLP**) is a branch of artificial intelligence or a field in machine learning that provides an ability to a machine to understand human written texts or speech. NLP enables machines to act on human language in the form of voice or text data so that the machine can understand its complete meaning as per the speaker or writer's intent and sentiment. Several NLP task breakdown is as follows:

a) *Speech Recognition:* In this section, NLP tries to convert voice data into text data which is required for any application that needs to answer spoken questions (like chatbots) or follow voice commands (like Alexa).
b) *Named entity recognition:* It identifies words or phrases as a useful entity in either the searching process or prediction analysis. For example, NEM identifies "Noida" as a place or "Hanumana" as a name.
c) *Co-reference resolution:* It is the task to identify when two or more words refer to the same entity. In addition, it can identify idioms or metaphors in the text.
d) *Sentiment analysis:* The task of sentiment analysis is to determine whether the piece of text is emotionally positive, negative or neutral.

There are several libraries in python that works on the concept of Natural Language Processing. One of the most important libraries that I have used in this project is "**Fuzzywuzzy**"

3.2.1 FuzzyWuzzy Library

As a data scientist, one should need to retrieve data from various sources either by conducting surveys through application APIs or by scraping data from websites. All this information is useful by combining it and apply data cleaning techniques to remove duplicates, spelling correction, etc. This is where we introduce *Fuzzy String Matching* functions in python.

The **Fuzzywuzzy** library uses ***Levenshtein Distance*** to calculate the differences between the sequences and patterns to match the strings approximately rather than exactly (Hence, it works on the concept of ***Fuzzy Logic***).

Python Commands: pip3 install fuzzy-wuzzy

pip3 install Levenshtein-distance

from fuzzy-wuzzy import fuzz

The two more important commands in the fuzzy-wuzzy library to check the accuracy of the programs similarity test functions, which was written for, string matching in this project.

Python Commands: fuzz.token_sort_ratio

Fuzz.token_set_ratio

Where token_set_ratio command provides better results.

3.2.1(a) Levenshtein Distance

Informally, it is the distance between the two words with the minimum number of single-character edits that required to change one word into the other.

Mathematical Definition:

"The Levenshtein Distance between two strings *a, b* (of length |a| and |b| respectively) is by **lev(a, b)** where,

$$lev(a,b) = \begin{cases} |a|, & if\ b = 0 \\ |b|, & if\ a = 0 \\ lev\big(tail(a), tail(b)\big), & if\ a[0] = b[0] \\ 1 + min\begin{cases} lev(tail(a), b) \\ lev\big(a, tail(b)\big) \\ lev(tail(a), tail(b)) \end{cases}, & otherwise \end{cases}$$

For example,

The distance between the two words 'eaten' and 'beating' is 3 as shown below,

1. Eaten → **B**eaten (inserting character 'B' in the start)
2. Beat**e**n → Beat**i**n (replacing character 'e' with 'i')
3. Beatin → Beatin**g** (inserting character 'g' in the end)

3.2.1(b) Fuzzy Logic

Fuzzy logic has a similar structure of layers as Neural Network, which is composed of, input layer, hidden layers and output layer (known as target layer in fuzzy logic). It also consists of the root node (because of which fuzzy logic is also known as directed rooted graph) which represents the decision parameter. The decision inputs remain stockpiled in the root node before the training starts. The classification decision starts from the root node and traverses the graph until it reaches the target nodes. A hidden layer no. 2 (in fig. 1) represents the possible combinations with the components of the input vector. The target layer works similar to the output layer of Neural Network. Unlike NN, here the output is classified based on the **degree of truth**.

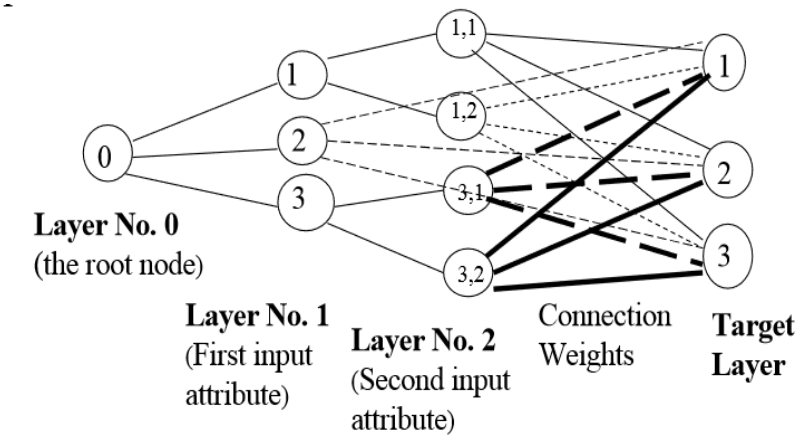

Fig.1. Fuzzy Logic Architecture

3.3 Neural Network

It named *Neural* because the design of the algorithm is based on the basic functionality of the *neurons.* Every neuron is connected with the weighted links. According to fig.2, the

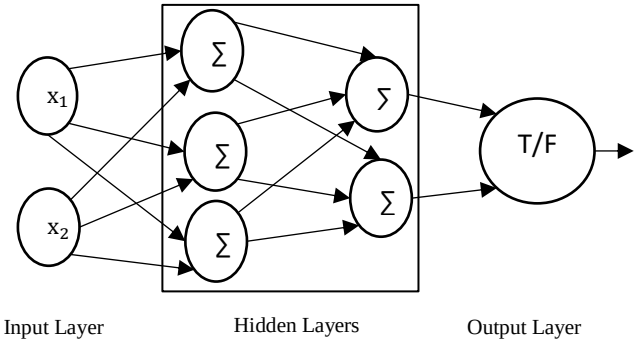

Fig.2. The architecture of the Neural Network

The neural network comprises three layers:

a) *Input Layer:* The number of neurons generated depends on the number of input variables.
b) *Hidden Layers:* The neurons generated in this layer was defined by the user according to the dataset or the usage of the model in the sequence like "*n, (n-1), (n-2)… 3, 2, 1*". For instance, if the user-defined sequence **"c(2, 1)" (R command)** then we can say that the hidden layer consists of two layers, one is having two neurons and the other is having one neuron.
c) *Output Layer:* It comprises only one neuron that generates the result of the whole model.

The formula used in Neural Network:

$$\boldsymbol{Output} = \sum_{\boldsymbol{i=0,a=0}}^{\boldsymbol{n,\infty}} \boldsymbol{w_i} * \boldsymbol{v_{i_a}}$$

Where $\boldsymbol{n}$ = Total number of variables in the dataset

$\boldsymbol{w_i}$ = Weight on the links generated by the neural network

$\boldsymbol{v_{i_a}}$ = Input values of the variable of the normalized dataset, such that

$\boldsymbol{i}$ = Number of variables & $\boldsymbol{a}$ = Number of observation of that particular variable.

The most important step before the neural network is to normalize the data, now again the question arises, ***what is normalized data*?** Well, data normalization is the process in which we will apply the normalised formula and convert all the values of the dataset between "-1 to +1".

Formula Used:

$$\boldsymbol{Norm} = \frac{\boldsymbol{X - Xmax}}{\boldsymbol{Xmax - Xmin}}$$

3.4 EasyGUI

In easygui, the simple function calls invoke all the GUI interactions. It provides a simple Graphical User Interface to the user for easy-to-use.

Python commands: pip3 install easygui

```
from easygui import *
import sys
```

Some of the important easyGUI Application Interface (API) with syntaxes are:

a) *Button box:* easygui.buttonbox(msg='', title=' ', choices=('Button[1]', 'Button[2]', 'Button[3]'), image=None, images=None, default_choice=None, cancel_choice=None, callback=None, run=True)

b) *Text box:* easygui.textbox(msg='', title=' ', text='', codebox=False, callback=None, run=True)
c) *Yes/No box:* easygui.ynbox(msg='Shall I continue?', title=' ', choices=('[]Yes', '[]No'), image=None, default_choice='[]Yes', cancel_choice='[]No')
d) *Continue/cancel box:* easygui.ccbox(msg='Shall I continue?', title=' ', choices=('C[o]ntinue', 'C[a]ncel'), image=None, default_choice='Continue', cancel_choice='Cancel')
e) *Message box:* easygui.msgbox(msg='(Your message goes here)', title=' ', ok_button='OK', image=None, root=None)
f) *Choice box:* easygui.choicebox(msg='Pick an item', title='', choices=[], preselect=0, callback=None, run=True)

3.5 Complexity

An algorithm complexity can be defined as the analysis of the performance of an algorithm in terms of time or space of the given size "n". Let us go through these two complexities one by one:

a) Time complexity: It quantifies the amount of time taken by an algorithm at the time of execution as per the length of the function inputs.
b) Space complexity: It quantifies the amount of space occupied by an algorithm as per the size of the function inputs.

As per the project requirement, we will only focus on the "Time complexity" of my generated algorithm, but before that, we need to understand some basic and important concepts of complexity.

a) *Big - Ω notation (Best Case)*: It denotes asymptotic lower bound.
$$\Omega(g(n)) = \{f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq c * g(n) \leq f(n) \text{ for all } n \geq n_0\}$$
b) *Big – Θ notation (Average Case)*: It denotes asymptotic tight bound.
$$\Theta(g(n)) = \{f(n): \text{there exist positive constants } c_1, c_2 \text{and } n_0 \text{ such that } 0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n) \text{ for all } n \geq n_0\}$$
c) *Big – O notation (Worst Case)*: It denotes asymptotic upper bound.
$$O(g(n)) = \{f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c * g(n) \text{ for all } n \geq n_0\}$$

Calculating Time complexity of my algorithm:

The complete algorithm works on the *Levenshtein Distance* string matching technique. As per the code, the following dataset is imported using 'pandas' and cleaning was done by using 'numpy'. The size of this movie dataset is 25 million. Hence, the time complexity is very high.

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
5,Father of the Bride Part II (1995),Comedy
6,Heat (1995),Action|Crime|Thriller
7,Sabrina (1995),Comedy|Romance
8,Tom and Huck (1995),Adventure|Children
9,Sudden Death (1995),Action
10,GoldenEye (1995),Action|Adventure|Thriller
11,"American President, The (1995)",Comedy|Drama|Romance
12,Dracula: Dead and Loving It (1995),Comedy|Horror
13,Balto (1995),Adventure|Animation|Children
14,Nixon (1995),Drama
15,Cutthroat Island (1995),Action|Adventure|Romance
16,Casino (1995),Crime|Drama
17,Sense and Sensibility (1995),Drama|Romance
18,Four Rooms (1995),Comedy
19,Ace Ventura: When Nature Calls (1995),Comedy
20,Money Train (1995),Action|Comedy|Crime|Drama|Thriller
21,Get Shorty (1995),Comedy|Crime|Thriller
22,Copycat (1995),Crime|Drama|Horror|Mystery|Thriller
23,Assassins (1995),Action|Crime|Thriller
24,Powder (1995),Drama|Sci-Fi
25,Leaving Las Vegas (1995),Drama|Romance
26,Othello (1995),Drama
27,Now and Then (1995),Children|Drama
28,Persuasion (1995),Drama|Romance
29,"City of Lost Children, The (Cité des enfants perdus, La) (1995)",Adventur
```

Fig.3. A glimpse of the movie dataset

Since Levenshtein distance is the inbuilt function of a fuzzy-wuzzy module, therefore, the time complexity of the string matching algorithm is:

$$\mathbf{O\ (n+}d^{n}\mathbf{)}$$

Where n → is the size of a dataset and d → is the edit distance

Since the mentioned time complexity is in quadratic form but the highest power of **n** is **1**, therefore, it matches with the remaining part of an algorithm whose time complexity is **O (n).**

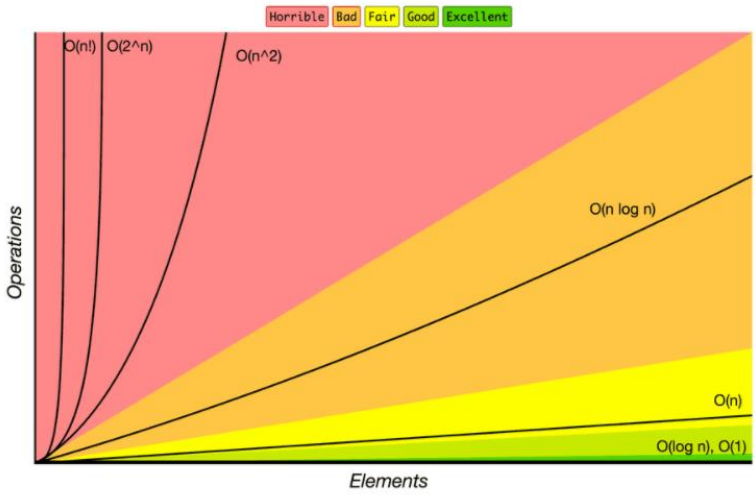

Fig.4. Time complexities variation graph

As we can observe from figure 4, that **O (n)** time complexity is in the fair section of the complexity graph. Hence, we can consider this algorithm as a better movie recommender system algorithm with required improvements in terms of time complexity.

CHAPTER 4: WORKING MODEL

Data Flow Diagram:

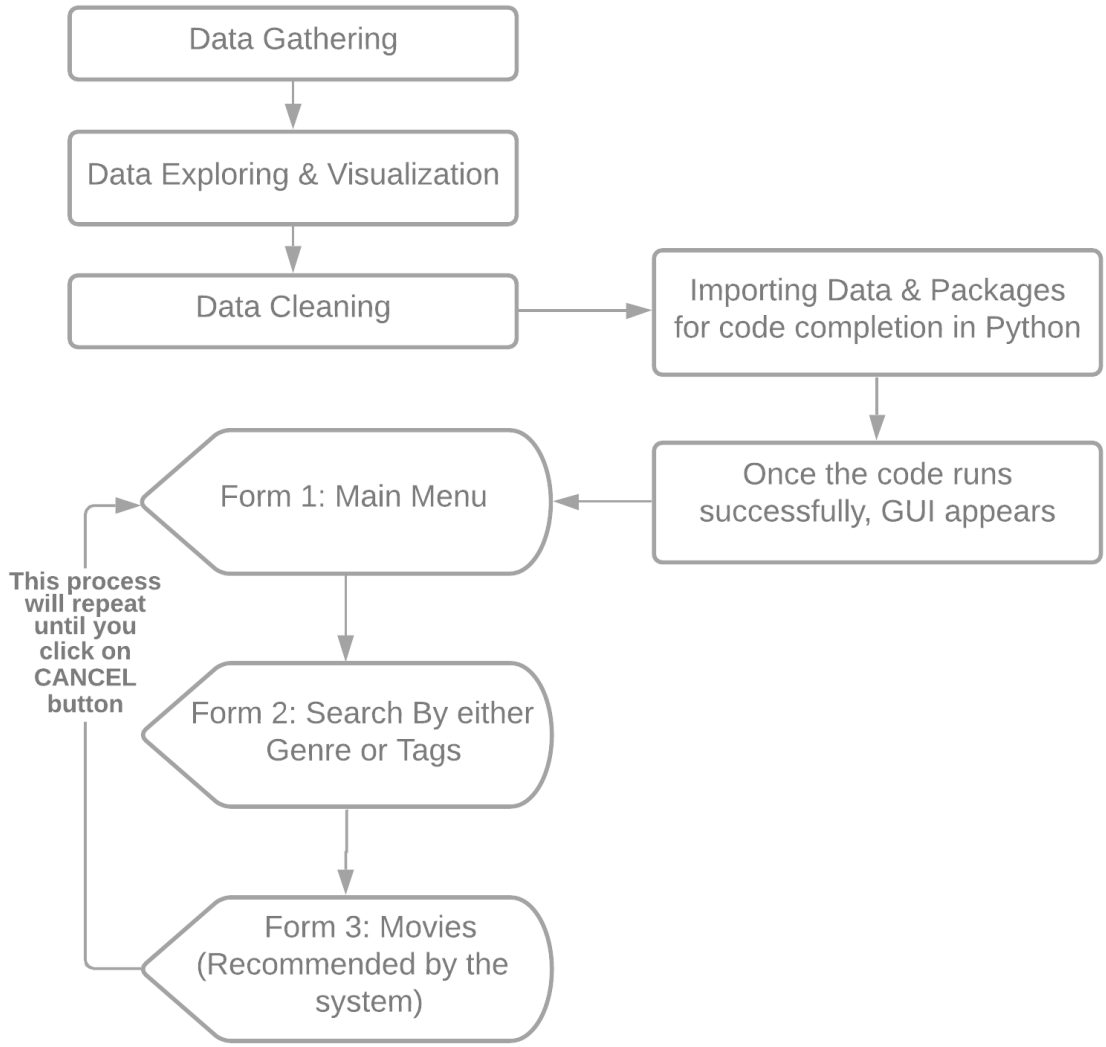

Fig. 5. DFD of Working Model

Step 1: Gathering authentic data is one of the most important and primary steps in the field of data analysis.

Step 2: Exploring the data will allow you to extract meaningful data from a whole lot of raw data. While exploring we can do better scrutiny by visualising the data with the help of a bar graph.

Step 3: In this step, if we can find duplicate data, misspelt data, or missing values then we need to clean the dataset first before put into the Machine Learning models.

Step 4: For writing the python code, we are using Geany IDE. So, we first need to import some important packages and libraries, with the help of which we will able to import the dataset in the IDE as well as able to write the complete program of a movie recommender system using natural language processing and machine learning models. The packages used are:

- Fuzzywuzzy
- Easygui
- Pandas
- Numpy
- Sys
- Progress.bar

Step 5: Once the code runs successfully, the simple user interface will get generated through which the system will ask for the user inputs to provide the desired results.

Step 6: Form 1 → this form titled as 'main menu', in which the user needs to select one option out of two that is, whether the user wants to "search movies by their genre" or wants to "search movies by their tags".

Step 7: Form 2 → in this form the user either needs to enter the *genres* of the movie, if he/she clicked on "search movies by their genre" in the first form or the user needs to enter the *tags* of the movie, if he/she clicked on "search movies by their tags" in the first form.

Step 8: Form 3 → based on the type of search this form titled "*movies*" will generate the list of recommended movies available in the database.

CHAPTER 5: CODE SNIPPETS

Code snippet with explanations:

First Function:

```
def which_way():
    # defining easygui choicebox parameters
    msg = "Choose an option:"
    title = "Main Menu"
    choices = ["Search recommended movies by genre:","Search recommended movies by tag:"]
    fieldValues = choicebox(msg,title, choices)
    # fieldValues variable is the user input which gets returned from the gui
    # conditional statement to guide user to the next interface based on input
    if fieldValues == "Search recommended movies by genre:":
        genre_entry()
    elif fieldValues == "Search recommended movies by tag:":
        tag_entry()
```

Explanation:

The first function of the program, which is executed when running the program. This functions as a crossroad, either you choose to search movies by genre or by tags. Based on user choice, the user gets directed to the next window.

Second Function:

```
def field_check(msg, title, fieldNames):
    fieldValues = multenterbox(msg, title, fieldNames)
    # Loop with conditionals to make sure that none of the user input
    # fields are left blank
    while 1:
        if fieldValues is None: break
        errmsg = ""
        for i in range(len(fieldNames)):
            if fieldValues[i].strip() == "":
```

```
                errmsg += ('"%s" is a required field.\n\n' % fieldNames[i])
        if errmsg == "":
            break # if no empty fields found, proceed with next codeblock
        # Saving user input as list in fieldValues variable
        fieldValues = multenterbox(errmsg, title, fieldNames, fieldValues)
    return fieldValues
```

Explanation:

This function checks for missing user input values in the multi-enter box and returns the user input as a fieldValues variable.

Parameters: msg, title and fieldnames of the multi-enterbox GUI

Third Function:

```
def tag_entry():
    # defining easygui multenterbox parameters
    msg = "Enter movie tag for example: world war 2 | brad pitt | documentary \nIf tag not found you will be returned to this window"
    title = 'Search by tag'
    fieldNames = ["Tag"]
    # calling on field_check() to check for missing user input and to
    # save user input as fieldValues variable
    fieldValues = field_check(msg, title, fieldNames)
    # If user input is not empty, slice list element and save in variable
    if fieldValues != None:
        global user_input_2
        user_input_2 = fieldValues[0]
        # here we call on a function which basically tests for string
        # similarity. if user press cancel, user gets returned to main menu
        similarity_test2(user_input_2)
    else:
        which_way()
```

Explanation:

This function defines the easygui multi-enterbox parameters and calls on the field_check functions, if field/user input is retrieved function calls on similarity test and if a match is not found user gets returned to the same window.

Fourth Function:

```
def tag():
    # append tag matched movies as dataframe object
    final_1 = []
    for i in final_2:
        final_1.append(df_tags.loc[df_tags['tag'].isin(i)])
    # reset df index, drop index column as well as duplicate entries
    lst = final_1[0]
    lst = lst.reset_index()
    lst.drop('index', axis=1, inplace=True)
    lst = lst.drop_duplicates(subset='movieId')
    # merge movieId with movie names and genres + drop tag and movieId column
    df = pd.merge(lst, df_1, on='movieId', how='left')
    df.drop('tag', axis=1, inplace=True)
    df.drop('movieId', axis=1, inplace=True)
    # sort movies by ratings, display only movies rated above or = 2.5 rating
    data = df.sort_values(by='rating', ascending=False)
    data = data[data['rating'] >= 2.5]
    heading = [] # add column names as first dataframe row for easygui display
    heading.insert(0, {'rating': 'Rating', 'title': '----------Title',
    'genres': '----------Genre'})
    data = pd.concat([pd.DataFrame(heading), data], ignore_index=True, sort=True)
    # casting dataframe columns to lists
    rating = data['rating'].tolist()
    title = data['title'].tolist()
    genres = data['genres'].tolist()
    # compiling numpy array from dataframe column lists for easygui display
```

```
        data = np.concatenate([np.array(i)[:,None] for i in [rating,title,genres]], axis=1)
        data = str(data).replace('[','').replace(']','')
        # displaying matched movies to user
        gui.codebox(msg='Movies filtered by tag returned from database:',
        text=(data),title='Movies')
    which_way()
```

Explanation:

This function adds all tag-matched movies to pandas dataframe, alter the dataframe for correct easygui display by dropping some columns, resetting df index, merging df's and sorting movies to display only movies rated >= 2.5, it also casts the final df columns to lists and rearrange them in a NumPy array for easygui display.

Fifth Function:

```
def genre_entry():
        # defining easygui multenterbox parameters
        msg = "Enter movie genre for example: mystery | action comedy | war \nIf genre not found you will be returned to this window"
        title = "Search by genre"
        fieldNames = ["Genre"]
        # calling on field_check() to check for missing user input and to
        # save user input as fieldValues variable
        fieldValues = field_check(msg, title, fieldNames)
        # If user input is not empty, slice list element and save in variable
        if fieldValues != None:
                global user_input
                user_input = fieldValues[0]
        # here we call on a function which basically tests for string
        # similarity. if user press cancel, user gets returned to main menu
                similarity_test1(user_input)
        else:
                which_way()
```

Explanation:

This function defines the easygui multi-enterbox parameters and calls on the field_check functions, if field/user input is retrieved function calls on similarity test and if a match is not found user gets returned to the same window.

Sixth Function:

```
def genre():
    # append genre matched movies as dataframe object
    final_1 = []
    for i in final:
        final_1.append(df_movies.loc[df_movies['genres'].isin(i)])
    # reset df index, drop index columns as well as duplicate entries
    lst = final_1[0]
    lst = lst.reset_index()
    lst.drop('index', axis=1, inplace=True)
    lst.drop('title', axis=1, inplace=True)
    lst.drop('genres', axis=1, inplace=True)
    lst = lst.drop_duplicates(subset='movieId')
    # merge movieId with movie names, rating and genres + drop index, title and genres column
    df = pd.merge(lst, df_1, on='movieId', how='left')
    # sort movies by ratings, display only movies rated above or = 2.5 rating
    data = df.sort_values(by='rating', ascending=False)
    data.drop('movieId', axis=1, inplace=True)
    data = data[data['rating'] >= 2.5]
    heading = [] # add column names as first dataframe row for easygui display
    heading.insert(0, {'rating': 'Rating', 'title': '----------Title','genres': '----------Genre'})
    data = pd.concat([pd.DataFrame(heading), data], ignore_index=True, sort=True)
    # casting dataframe columns to lists
    rating = data['rating'].tolist()
    title = data['title'].tolist()
    genres = data['genres'].tolist()
```

```
        # compiling numpy array from dataframe column lists for easygui display
        data = np.concatenate([np.array(i)[:,None] for i in [rating,title,genres]], axis=1)
        data = str(data).replace('[','').replace(']','')
        # displaying matched movies to user
        gui.codebox(msg='Movies filtered by genre returned from database:',
        text=(data),title='Movies')
    which_way()
```

Explanation:

This function adds all genre-matched movies to pandas dataframe, alter the dataframe for correct easygui display by dropping some columns, resetting df index, merging df's and sorting movies to display only movies rated >= 2.5, it also casts the final df columns to lists and rearrange them in a NumPy array for easygui display.

Seventh Function:

```
def similarity_test1(user_input):
        # saving movie genre/s as test base and user input to test for
        genre_list = df_movies['genres'].unique()
        query = user_input
        choices = genre_list
        # here fuzzywuzzy does its magic to test for similarity
        output = process.extract(query, choices)
        # for accuracy check of our similarity_test1 model
        print (fuzz.token_set_ratio(query, choices))
        # saving returned matches in variable and passing it on to next function
        global final
        final = [i for i in output if i[1] > 90]
        # if no match of > 90%  is found, return to genre entry window for retry
        if final == []:
                genre_entry()
        else:
                genre()
```

Explanation:

This function tests for string similarity by matching user input to movies genres, match/es > 90% gets saved in a variable which then gets passed on to genre function for database match and return unless no match is found.

Eighth Function:

```
def similarity_test2(user_input_2):
    # saving movie tag as test base and user input to test for
    tag_list = df_tags['tag'].unique()
    query = user_input_2
    choices = tag_list
    # here fuzzywuzzy does its magic to test for similarity
    output = process.extract(query, choices)
    # for accuracy check of our similarity_test2 model
    print (fuzz.token_set_ratio(query, choices))
    # saving returned matches in variable and passing it on to next function
    global final_2
    final_2 = [i for i in output if i[1] > 90]
    # if no match of > 90%  is found, return to tag entry window for retry
    if final_2 == []:
        tag_entry()
    else:
        tag()
```

Explanation:

This function tests for string similarity by matching user input to movie tags, match/es > 90% gets saved in the variable which then gets passed on to tag function for database match and return unless no match is found.

CHAPTER 6: RESULTS

ScreenShots with explanation (wherever required):

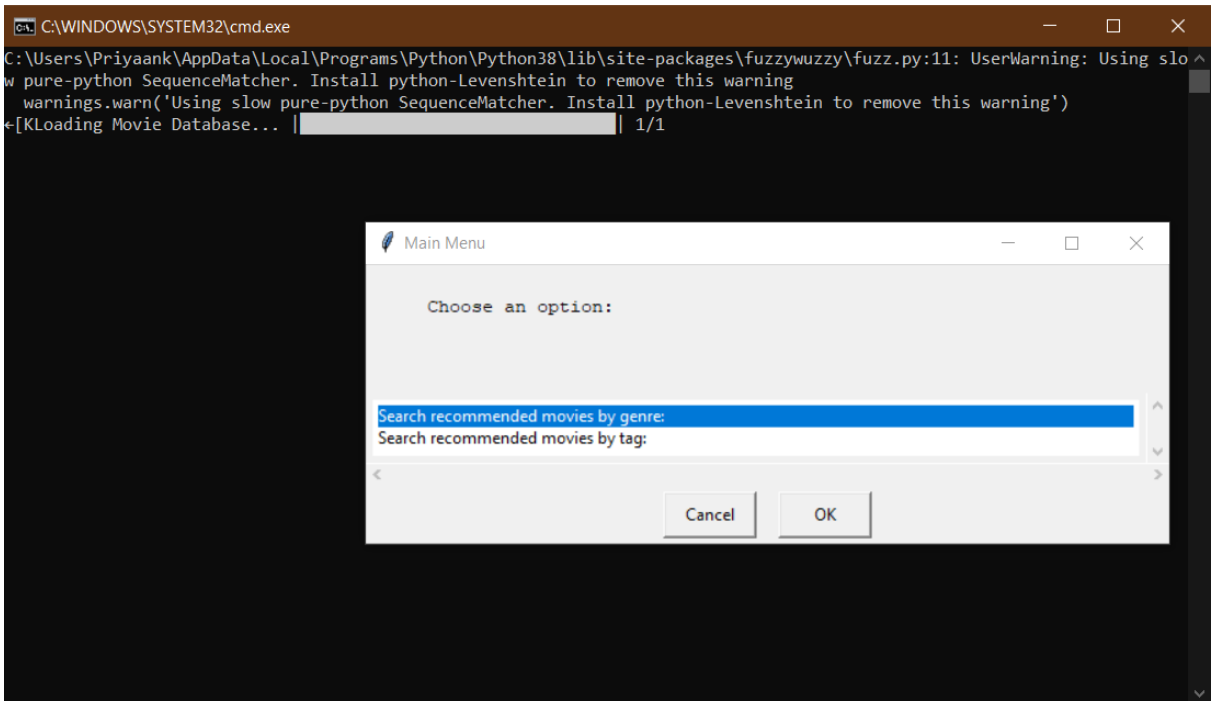

Fig.6. Form 1(a)

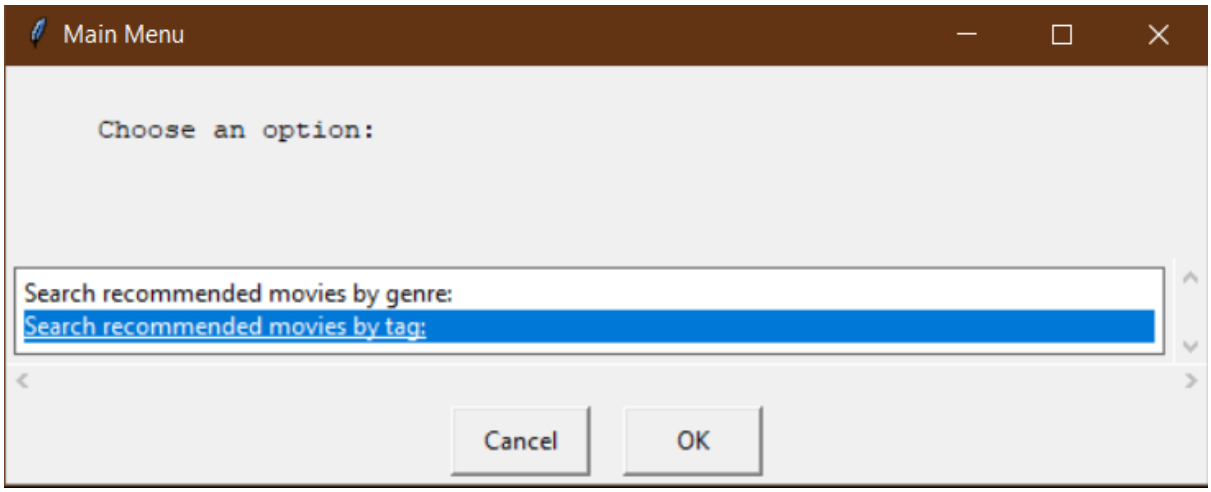

Fig.6. Form 1(b)

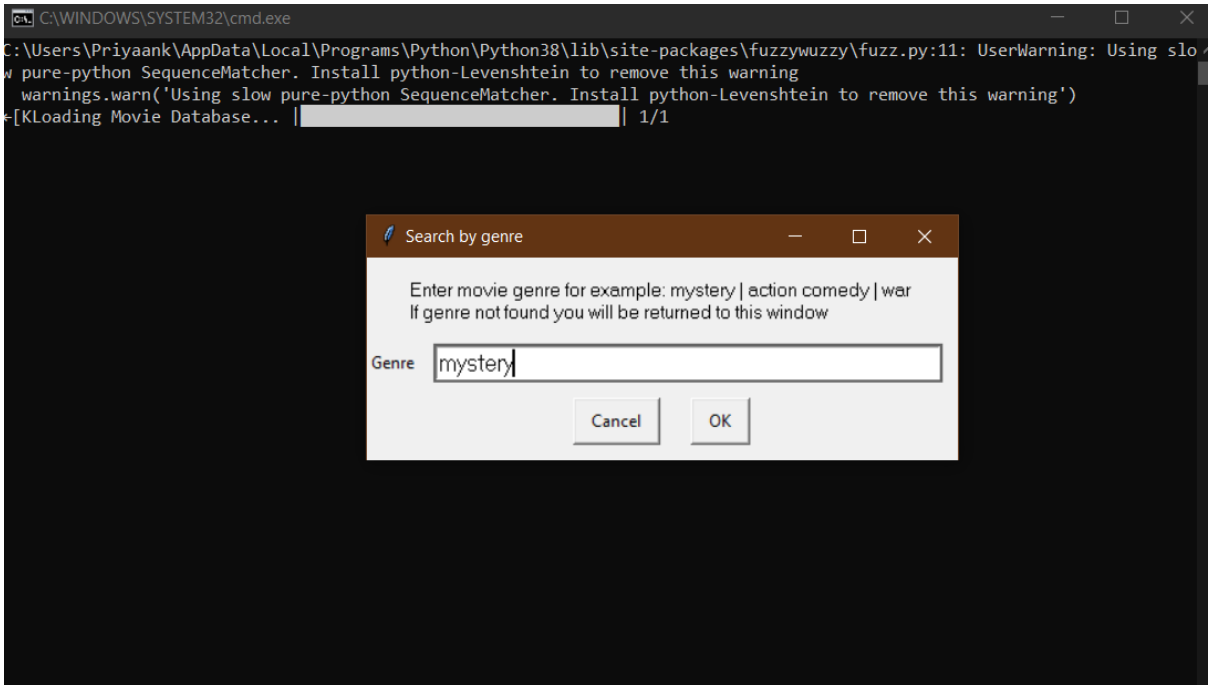

Fig.7. Form 2(a)

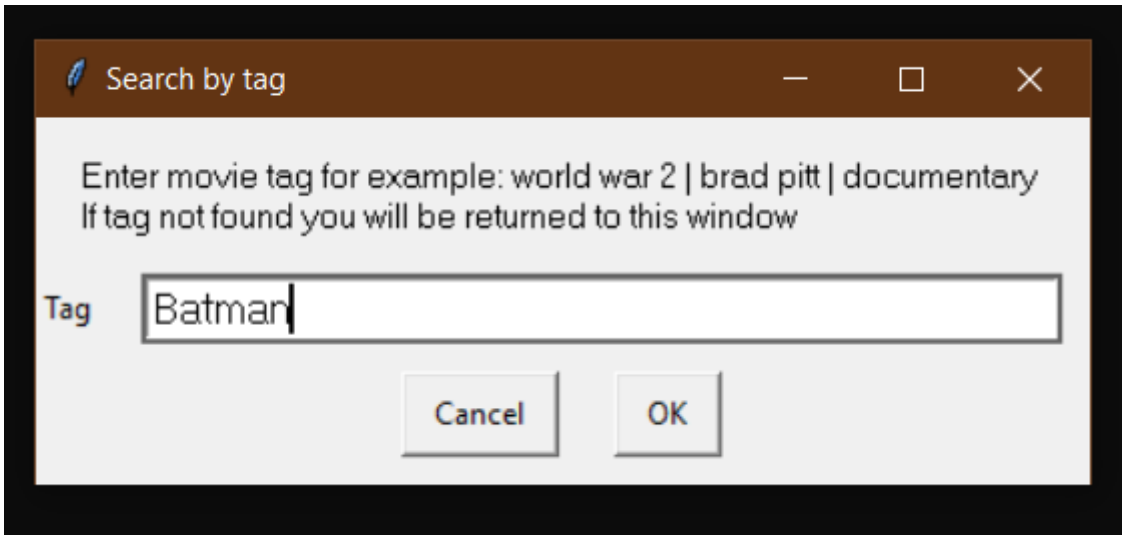

Fig.7. Form 2(b)

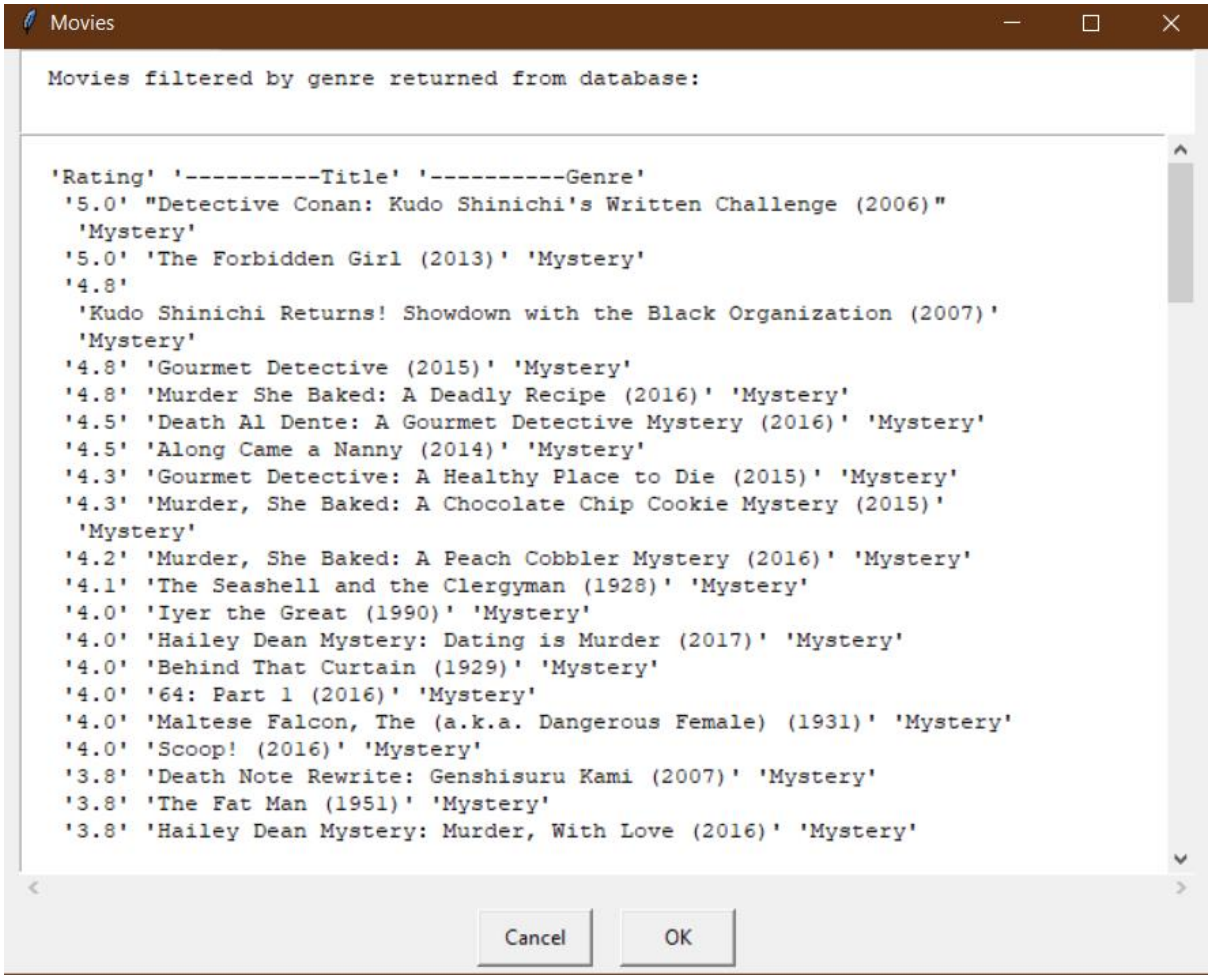

Fig.8. Form 3(a)

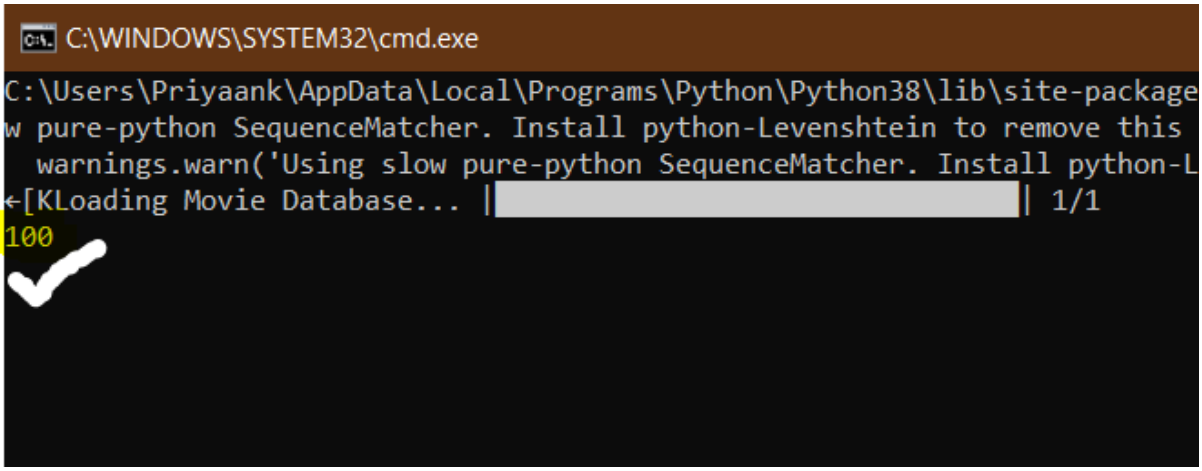

Fig.9. Accuracy rate when searching is by **genre**

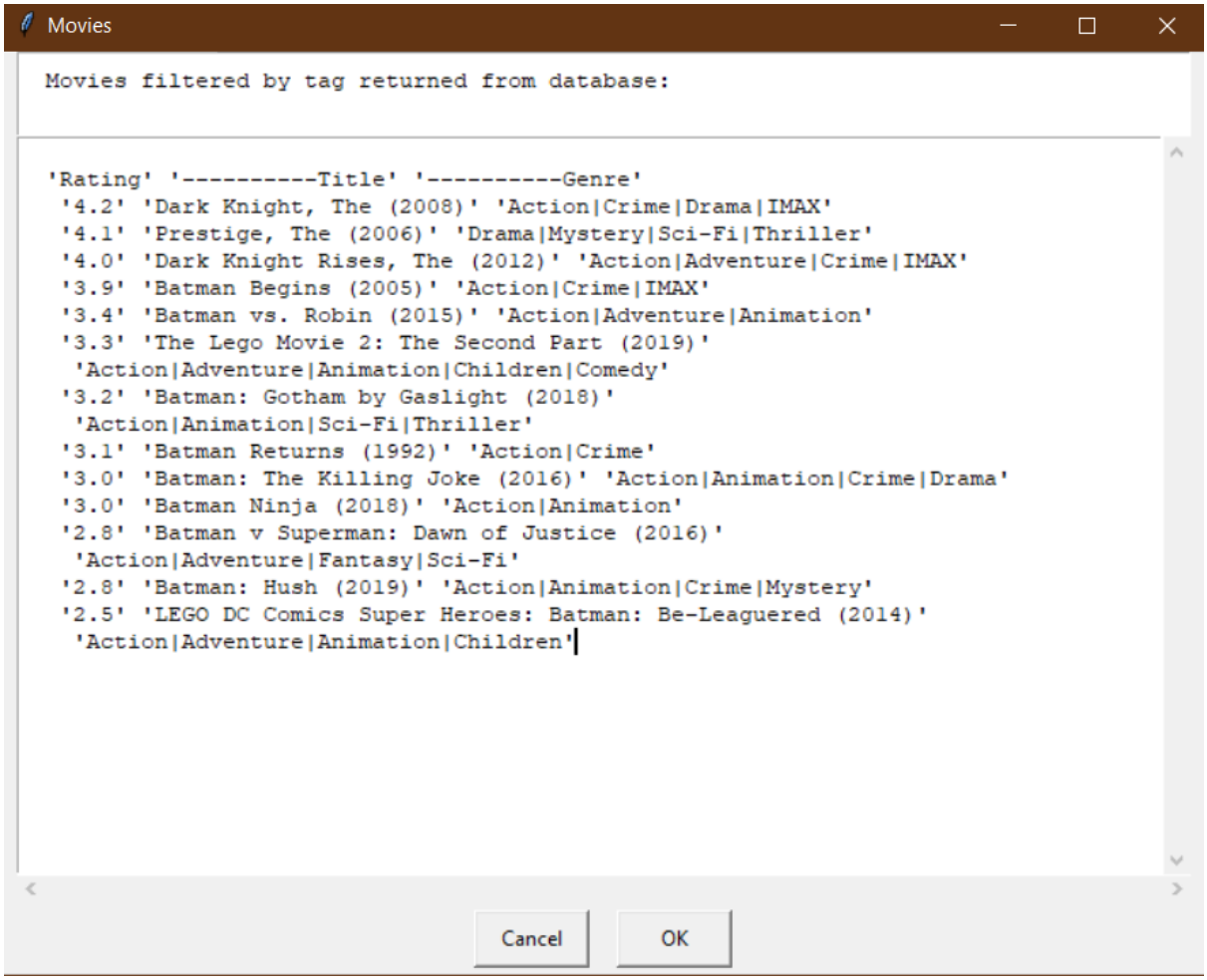

Fig.8. Form 3(b)

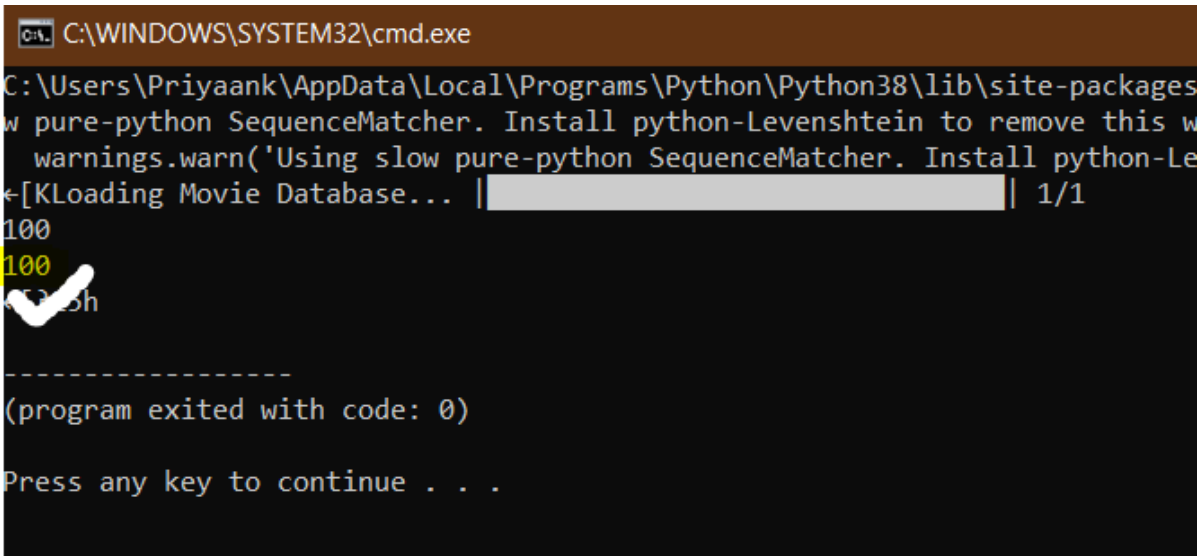

Fig.10. Accuracy rate when searching is by **tag**

CHAPTER 7: CONCLUSION & FUTURE SCOPE

Conclusion:

From small to large industries, data flow is everywhere; hence, data analysis is the deciding factor for any industry success rate. This project is a part of data analysis in which we have successfully explained all the concepts i.e.; from *data exploration* to *accuracy check using fuzzuwuzzy* in detail. From the experimental point of view, we have successfully created a simple graphical user interface (GUI) for recommending movies on the concept of content-based and collaborative recommender systems using Natural Language Processing libraries with an accuracy rate of **100%**. Hence, the demonstrated model in the project can successfully recommend the movies based on the user search either by genre or by tags.

Future Work:

The recommender system is working on 25--million-movie dataset with the time complexity of **O (n),** which can be improved a lot.

REFERENCES

[1] https://www.analyticsindiamag.com/7-types-classification-algorithms/

[2] https://app.creately.com/

[3]https://pub.towardsai.net/recommendation-system-in-depth-tutorial-with-python-for-netflix-using-collaborative-filtering-533ff8a0e444

[4]https://www.edureka.co/blog/tkinter-tutorial/

[5] http://www.authorstream.com/Presentation/NIKKUTANU-1431448-pradeep-correlation-and-regression/

[6] https://datascienceplus.com/neuralnet-train-and-test-neural-networks-using-r/

[7] https://www.sas.com/en_gb/insights/articles/analytics/a-guide-to-predictive-analytics-and-machine-learning.html

[8] https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

[9] https://www.freecodecamp.org/news/an-introduction-to-web-scraping-using-r-40284110c848/

[10]https://towardsdatascience.com/fuzzy-string-matching-in-python-68f240d910fe

[11]https://towardsdatascience.com/natural-language-processing-for-fuzzy-string-matching-with-python-6632b7824c49

[12]https://www.ibm.com/cloud/learn/natural-language-processing

[13]https://easygui.readthedocs.io/en/master/tutorial.html#using-buttonboxes

[14]https://readthedocs.org/projects/easygui/downloads/pdf/latest/

[15] https://www.datacamp.com/community/tutorials/neural-network-models-r

[16] https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

[17] https://www.kdnuggets.com/2016/08/begineers-guide-neural-networks-r.html

[18] http://uc-r.github.io/ann_classification

[19] https://towardsdatascience.com/build-your-own-neural-network-classifier-in-r-b7f1f183261d

[20] http://www.learnbymarketing.com/tutorials/neural-networks-in-r-tutorial/

[21]https://towardsdatascience.com/recommendation-system-in-python-lightfm-61c85010ce17