Computer science Internal Assessment
CRITERION C: Development

Word Count: 637

Libraries Imported:

```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix #compress the memory that a matrix uses
from sklearn.neighbors import NearestNeighbors #knn distance algorithm
import matplotlib.pyplot as plt #plotting graph
```

From scipy library, compressed sparse row is imported to reduce the memory a matrix uses. Sklearn library is used to import the main knn distance algorithm used to recommend movies. Pandas and Numy are for analysing and iterating datasets, while matplotlib is for plotting graphs and understanding data.

Creating the Movie Recommendation method:

Step 1: Importing the datasets from device

```
ratings = pd.read_csv("/Users/himangiparekh/Desktop/COMPUTER SCIENCE/CS IA/movieListSmall/
# imported dataset ratings from computer
# contains information on the ratings given to each movie by each user
```

```
ratings = ratings.rename(columns={'userId': 'userKey', 'movieId': 'movieKey', 'rating': '
# renaming the columns
```

```
ratings.head()
# printing the first five rows of the dataset
```

	userKey	movieKey	ratingGiven	duration
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Step 2: Removing characters like colen, dashes etc from the movie titles to make case-less and character-less comparison with user's input.

```
movies = pd.read_csv("/Users/himangiparekh/Desktop/COMPUTER SCIENCE/CS IA/movieListSmall/r
# importing movies dataset from computer
# contains information regarding the movieId and movieName + genre
# important to connect the two datasets and user will eventually use movie name not ID
# also will be used for allowing user to subset the recommendations based on genre
movies = movies.rename(columns={'movieId': 'movieKey', 'title': 'nameOfMovie', 'genres':
# renaming the columns
movies_list = movies.iloc[:, 2].values
for i in movies_list:
    movies['nameOfMovie'] = movies['nameOfMovie'].str.replace(':', '')
movies
# printing the first five rows of the dataset
```

```
movies_list = movies.iloc[:, 2].values
for i in movies_list:
    movies['nameOfMovie'] = movies['nameOfMovie'].str.replace('-', ' ')
movies
```

Step 3: Presenting 'ratings' database in a comprehensible and concise matrix.

```
matrix = ratings.pivot(index='movieKey',columns='userKey',values='ratingGiven')
# changing the organisation of the 'ratings' dataset into a matrix:
# with columns as the movieId
# with rows as the userId
# with each cell representing the ratings given by the particular user to the particular
# movie
matrix
```

userKey	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609
movieKey																				
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193583	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193585	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193587	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193609	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

9724 rows × 610 columns

Step 4: Reducing the sparsity of the dataset.

```
# now we want to reduce the sparsity of the dataset
# we will check to see if each movie has been rated a certain number of times
# if the respective conditions are not satisfied, the row will be erased

non_nan_movies = matrix.count(1)
# storing the number of non 'NAN' in each row (number of ratings given to each movie)
non_nan_movies
```

```
non_nan_movies_list = non_nan_movies.tolist()
# storing the number of ratings given to each movie as a list
print(non_nan_movies_list)
```

Step 6: Summarising the data to check for outliers which may skew the results.

```
# we can see that there are several outliers, so setting the benchmark as the average
# will be of little use
# let us find the range, mode and median

import statistics

print("range =", range(min(non_nan_movies_list), max(non_nan_movies_list)))

def Median_movies(non_nan_movies_list):
    return statistics.median(non_nan_movies_list)

median_movies = Median_movies(non_nan_movies_list)
print("median =", median_movies)

mode_movies = statistics.mode(non_nan_movies_list)
print("mode =", mode_movies)

# since the mode is 1 and the median is 3, there is minimal scope to reduce
# the sparsity
# so let us follow the trial and error method to decide the benchmark
```

```
range = range(1, 329)
median = 3.0
mode = 1
```

Step 7: Finding how many movies are rated less than three times (as three is the medium).

```
# if statement inside for loop:
# let us start with 3 as it is the median
# how many indices (which is the number of rows) have values < 3 in list?

counter =  0 # Counter Variable
number_of_rows = 0
for i in non_nan_movies_list:
    if i < 3:
        #print(counter)
        number_of_rows+=1
    counter+=1 # increments counter
print("Total Rows with less than 3 values: ", number_of_rows)
```

```
Total Rows with less than 3 values:  4744
```

```
# since around half the no. of rows have < 3 values,
# it does not make sense to remove all these rows, because then the number of movies
# that the user will have access to will be very low
# let us try 2
# how many indices (which is the number of rows) have values < 2 in list?

counter =  0 # Counter Variable
number_of_rows = 0
for i in non_nan_movies_list:
    if i < 2:
        #print(counter)
        number_of_rows+=1
    counter+=1 # increments counter
print("Total Rows with less than 2 values: ", number_of_rows)
```

```
Total Rows with less than 2 values:  3446
```

Since the mode is 1 and there are 3446 movies rates less than three times, it means that there is minimal scope to reduce the sparsity of the dataset, as removing these rows will most will deem fewer choices for out users.

Step 8: So, the compressed sparse matrix function war invoked to reduce the memory used by the NaN or 0 values such that future operations become faster and more efficient.

```
compress_data = csr_matrix(matrix.values) # basically used to compress the memory used by
matrix.reset_index(inplace=True) #creating index column
matrix
```

Step 9: Knn (nearest neighbour algorithm) was researched and invoked on the compressed data. Knn basically plots the ratings as points on an *n* dimensional graph and uses metrics to find the distance from one to another. The less the distance, the more similar the two points are. ‘Cosine’ indicates that eucledian distance was used. ‘Brute’ is specially designed for sparse matrices.

```
from sklearn.neighbors import NearestNeighbors
# here, we are using the knn algorithm which finds the nearest neighbor/s of a moive
# it plots all the movies in the dataset based on the ratings on a graph
# then, when the user inputs a movie, it will return the 'nearest neighbors' of that movie

knn_algorithm = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jo
# the metric = the way that the algorithm will measure the distance
# minkowski is combination of manhattan and euclidean distance
# n_neighbours is the number of neighbours we want the algorithm the find
# algorithm brute is designed for sparse matrices
# which will be useful for accurately plotting the movies, considering ratings
# and other features (which program will decide)

knn_algorithm.fit(compress_data)
# .fit is a sklearn method which best estimates the representative model of the data
# the function can help the 'metric' part of the algorithm to plot the points most
# efficiently and effectively
```

```
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

Finally creating a method to recommend movie:

A) Initialising a list of movies which contains the user input (like spider man, spider man two etc.)

```
def recommend_movie():
    list_of_mov = movies[movies['nameOfMovie'].str.contains(entry_movie_name_home.get(), case=False)]
```

B) Finding the movieKey of the 1st element in the list

```
if len(list_of_mov): # checking for whether the list is empty or not
    index_mov = list_of_mov.iloc[0]['movieKey']   # finding the movieKey of the 1st element in list
    index_mov = matrix[matrix['movieKey'] == index_mov].index[0]
    # finding the index from matrix dataframe of the movie key inputted from movies dataset
```

C) Implementing the KNN algorithm on the compressed matrix

```
knn_distance , knn_index = knn_algorithm.kneighbors(compress_data[index_mov],
                                                    n_neighbors=int(entry_number_mov_rec_home.get())+1)
# 2 aspects will be returned when this method is run
# finding neighbours of the 'movie index' using the knn algorithm and in-built kneighbors method
# the number of neighbours to find is no. movies to rec + 1 because it also includes the movie
# that the user inputs (which cannot be considered as a rec)
```

D) creating a tuple with 1st element as index of the movie (movie found by implementing knn algorithm) and second as the distance of that movie from the inputted movie then sorting this tuple based on the second element in each bracket (basically based on the distance)

a) .squeeze reduces the dimensionality of an array/list
b) zip: making pairs of index and distance
c) sorting it based on lamda which is a function to iterate on each element in the list: each variable second element

```
mov_rec_index = sorted(list(zip(knn_index.squeeze().tolist(),knn_distance.squeeze().tolist())),
                       key=lambda x: x[1])
```

E) Initialising an empty array which will have recommended movies

a) finding the movie key of the first element of the tuple from the matrix dataset
b) adding the array with name of movie from the movies dataset of the movie key
c) making the array into dataframe and adding an index to it
d) removing the first row of create data (as it is the movie itself)

```
for i in mov_rec_index: # looping through the neighbours we stored as tuple
    index_mov = matrix.iloc[i[0]]['movieKey']
    # finding the movie key of the first element of the tuple from the matrix dataset
    index = movies[movies['movieKey'] == index_mov].index
    # finding the index of the moviekey from the movie database
    arr_recommend.append({'nameOfMovie':movies.iloc[index]['nameOfMovie'].values[0]})
    # adding the array with name of movie from the movies dataset of the movie key
create_data = pd.DataFrame(arr_recommend,index=range(0,int(entry_number_mov_rec_home.get())+1))
# making the array into dataframe and adding an index to it
create_data2 = create_data.iloc[1:, 0]
```

F) Closing the loop and method

```
    tkinter.messagebox.showinfo("Recommendations", ' '.join(create_data2))
else:
    tkinter.messagebox.showinfo("Error", "Please input another movie.")
```

If the sample input is ‘spider man’ and user wants 15 movies, the output will be:

```
In [19]: recommend_movie('spider man', 15)
```

Out[19]:

	nameOfMovie
1	Spider Man 2 (2004)
2	Star Wars Episode II Attack of the Clones (2...
3	Minority Report (2002)
4	X2 X Men United (2003)
5	X Men (2000)
6	Pirates of the Caribbean The Curse of the Blac...
7	Lord of the Rings The Fellowship of the Ring, ...
8	Matrix Reloaded, The (2003)
9	Lord of the Rings The Two Towers, The (2002)
10	Bourne Identity, The (2002)
11	Incredibles, The (2004)
12	Star Wars Episode III Revenge of the Sith (2...
13	Lord of the Rings The Return of the King, The ...
14	Shrek (2001)
15	Fifth Element, The (1997)

Creating User Interface:

Libraries Imported:

Tkinter library is used to create UI as it has useful in-built functionality like buttons etc. Pymsql is required to connect to the database and PIL is needed to import images.

```
import tkinter
from tkinter import *
from tkinter import messagebox

import pandas as pd
import pymysql
from PIL import Image, ImageTk
```

Step 1: invoke base methods for the login page

```
def clear_log():
    un_log.delete(0,END)
    name_log.delete(0, END)
    ps_log.delete(0,END)

def clear_sign():
    un_sign.delete(0,END)
    name_sign.delete(0,END)
    ps_sign.delete(0,END)

def exit_log():
    login.destroy()

def login_database():

    if un_log.get() == "" or ps_log.get() == "":
        messagebox.showerror("Error!", "Please enter user name and password", parent=login)
```

Step 2: connect login inputs to database: when this is invoked via button, the code will connect to the database

A) Verifying whether entryboxes are empty:

```
def login_database():

    if un_log.get() == "" or ps_log.get() == "":
        messagebox.showerror("Error!", "Please enter user name and password", parent=login)
```

B) Checking whether the selected information from the database matches the inputs:

```
    else:
        try:
            connect_log = pymysql.connect(host="192.168.64.2", user="root", password="123",
                                          database="user_information")
            cursor_log = connect_log.cursor()

            cursor_log.execute("select * from user_information where username=%s and password = %s",(un_log.get(),
                                                                                                    ps_log.get()))
            row = cursor_log.fetchone()

            if row == NONE:
                messagebox.showerror("Error", "Invalid User Name And Password", parent=login)
```

C) Verifying and logging the user in

```
            else:
                messagebox.showinfo(f"Hello {name_log}!",  "You are successfully logged in!", parent=login)
                exit_log()
                homepage()

        except Exception as ErrorL:
            return ErrorL
```

Step 3: creating homepage function and embedding the recommend movie method in the function (which is explained formerly). When homepage() is called, the homepage will openup.

```
def homepage():

    def recommend_movie():
        list_of_mov = movies[movies['nameOfMovie'].str.contains(entry_movie_name_home.get(), case=False)]
        # subsets the movies dataset based on user input movie
        if len(list_of_mov): # checking for whether the list is empty or not
            index_mov = list_of_mov.iloc[0]['movieKey']   # finding the movieKey of the 1st element in list
            index_mov = matrix[matrix['movieKey'] == index_mov].index[0]
            # finding the index from matrix dataframe of the movie key inputted from movies dataset
            knn_distance , knn_index = knn_algorithm.kneighbors(compress_data[index_mov],
                                                                n_neighbors=int(entry_number_mov_rec_home.get())+1)
            # 2 aspects will be returned when this method is run
            # finding neighbours of the 'movie index' using the knn algorithm and in-built kneighbors method
            # the number of neighbours to find is no. movies to rec + 1 because it also includes the movie
            # that the user inputs (which cannot be considered as a rec)
            mov_rec_index = sorted(list(zip(knn_index.squeeze().tolist(),knn_distance.squeeze().tolist())),
                                   key=lambda x: x[1])
            # .squeeze reduces the dimensionality of an array/list
            # zip: making pairs of index and distance
            # creating a tuple with 1st element as index of the movie
            # (movie found by implementing knn algorithm) and second as the distance of that movie
            # from the inputted movie)
            # then sorting this tuple based on the second element in each bracket (basically based on the distance)
            # sorting it based on lamda which is a function to iterate on each element in the list:
            each variable second element
            # x:x[1] basically means that the sorting is happening in second element of tuple
            arr_recommend = []
            # initialising an empty array which will have recommended movies
            for i in mov_rec_index: # looping through the neighbours we stored as tuple
                index_mov = matrix.iloc[i[0]]['movieKey']
                # finding the movie key of the first element of the tuple from the matrix dataset
                index = movies[movies['movieKey'] == index_mov].index
                # finding the index of the moviekey from the movie database
                arr_recommend.append({'nameOfMovie':movies.iloc[index]['nameOfMovie'].values[0]})
                # adding the array with name of movie from the movies dataset of the movie key
            create_data = pd.DataFrame(arr_recommend,index=range(0,int(entry_number_mov_rec_home.get())+1))
            # making the array into dataframe and adding an index to it
            create_data2 = create_data.iloc[1:, 0]
            tkinter.messagebox.showinfo("Recommendations", ' '.join(create_data2))
        else:
            tkinter.messagebox.showinfo("Error", "Please input another movie.")
```

Step 4: creating home window and variables used locally:

```
home = Tk()
home.title("Homepage")
home.geometry("2000x2000")

mov_name = StringVar()
mov_number = StringVar()
```

Step 5: creating introduction labels, entry buttons and background image:

```
intro_home = Label(home, text="Welcome to Movie Recommender!", fg='blue', font=("arial", 30, "bold")).pack()

bag_image = Image.open("/Users/himangiparekh/Desktop/COMPUTER SCIENCE/CS IA/image.png")
background_image = ImageTk.PhotoImage(bag_image)
label1 = Label(image=background_image)                    # pasting the background image on the login window
label1.pack(fill=BOTH, expand=TRUE)

instruction_home = Label(home, text="Please input a movie name along with the number of recommendations that you
                         fg='black', font=("arial", 15, "bold"))
instruction_home.place(relx=0.27, rely=0.56)

movie_name_home = Label(home, text="Movie Name : ", font=("arial", 16, "bold"))
movie_name_home.place(relx=0.34, rely=0.33)

entry_movie_name_home = Entry(home, textvar=mov_name)
entry_movie_name_home.place(relx=0.55, rely=0.33)

number_mov_rec_home = Label(home, text="Number of Movies : ", font=("arial", 16, "bold"))
number_mov_rec_home.place(relx=0.34, rely=0.4)

entry_number_mov_rec_home = Entry(home, textvar=mov_number)
entry_number_mov_rec_home.place(relx=0.55, rely=0.4)
```

Step 6: create menu button for homepage:

```
    def abt_home():
        # initialising a method which shows a message in the home page

        tkinter.messagebox.showinfo("About", "Welcome! All you have to do is input a movie which you want your sugge



    menu_home = Menu(home)                                    # creating menu bar in homepage
    home.config(menu=menu_home)

    About_home = Menu(home)                                       # creating 'about' in menu
    menu_home.add_cascade(label="About", menu=About_home)         # adding dropdown in about
    About_home.add_command(label="What is this program?", command=abt_home)
    # when clicked, the abt command messagebox will open
    About_home.add_command(label="test", command=recommend_movie)
    home.mainloop()
```

Step 6: create signup method with variables used:

```
def signup_method():
    un_sign = StringVar()
    name_sign = StringVar()
    ps_sign = StringVar()
```

Step 7: connect signup variables to database: when signup is called, the signup window will open.

A) Verifying whether inputs are empty

```
def signup_database():

    if entry_un_signup.get() == "" or entry_name_signup.get() =="" or entry_password_signup.get() == "":
        messagebox.showerror("Error!", "Please input all sections.", parent=signup)
```

B) Checking whether username already exists:

```
    else:
        try:
            connect_sign = pymysql.connect(host="192.168.64.2", user="root", password="123",
                                           database="user_information")
            cursor_sign = connect_sign.cursor()
            cursor_sign.execute("select * from user_information where username=%s", entry_un_signup.get())
            find = cursor_sign.fetchone()

            if find!=None:
                messagebox.showerror("Error!", "User Name Already Exits. Please Select Another User Name.",
                                     parent=signup)
```

C) Entering information into database:

```
            else:
                cursor_sign.execute("insert into user_information(username, name, password) values(%s, %s, %s)",
                    (
                    entry_un_signup.get(),
                    entry_name_signup.get(),
                    entry_password_signup.get()
                    ))
                connect_sign.commit()
                messagebox.showinfo(f"Hello {entry_name_signup}!", "You are successfully signed up!",
                                    parent=signup)
                clear_sign()
                connect_sign.exit_sign()

        except Exception as Error:
            return Error
```

Step 8: make signup window and exit method for signup:

```
signup = Tk()  # creating window
signup.title("Sign Up Page")  # giving title
signup.geometry('2000x2000')  # size

def exit_sign():
    signup.destroy()
```

Step 9: buttons and instructions for the signup page, and closing the signup window with signup.mainloop()

```
    un_signup = Label(signup, text="Username : ", font=("arial", 16, "bold"))
    un_signup.place(relx=0.34, rely=0.33)

    entry_un_signup = Entry(signup, textvar=un_sign)
    entry_un_signup.place(relx=0.55, rely=0.33)

    name_signup = Label(signup, text="Name : ", font=("arial", 16, "bold"))
    name_signup.place(relx=0.34, rely=0.4)

    entry_name_signup = Entry(signup, textvar=name_sign)
    entry_name_signup.place(relx=0.55, rely=0.4)

    password_signup = Label(signup, text="Password : ", font=("arial", 16, "bold"))
    password_signup.place(relx=0.34, rely=0.47)

    entry_password_signup = Entry(signup, textvar=ps_sign, show='*')
    entry_password_signup.place(relx=0.55, rely=0.47)

    intro_signup = Label(signup, text="Welcome to Movie Recommender! Please sign up.", fg='blue',
                         relief=RAISED, font=("arial", 30, "bold")).pack()

    quit_button_signup = Button(signup, text="Quit", width=12, fg='brown', command=exit_sign)
    quit_button_signup.place(relx=0.53, rely=0.6)

    signup_button_signup = Button(signup, text="Sign Up", width=12, fg='brown', command=signup_database)
    signup_button_signup.place(relx=0.39, rely=0.6)

    login_instruction_signup = Label(signup, text="If you already have an account, please click on 'Login'.",
                                     fg='black', font=("arial", 14)).place(relx=0.38, rely=0.76)

    login_button_signup = Button(signup, text="Login", width=12, fg='brown', command=signup.destroy)
    login_button_signup.place(relx=0.46, rely=0.675)

    signup.mainloop()
```

Step 10: finally creating the login window and initialising variables:

```
login = Tk()                                    # creating a login window
login.geometry("2000x2000")                     # defining the size of the window
login.title("Login Page")                       # giving the window a title

un_log = StringVar()
# username (login page) as a string when user enters it in entry box
name_log = StringVar()
# name (login page) as a string when user enters it in entry box
ps_log = StringVar()
# password  (login page) as a string when user enters it in entry box
```

Step 11: creating menu for login page:

```
def abt():
    # initialising a method which shows a message in the login page
    tkinter.messagebox.showinfo("About", "This is a movie recommender. You must first log in to create your account.
```

```
menu_login = Menu(login)                                # creating menu bar in login page
login.config(menu=menu_login)

About = Menu(login)                                     # creating 'about' in menu
menu_login.add_cascade(label="About", menu=About)       # adding dropdown in about
About.add_command(label="What is this program?", command=abt)
# when clicked, the abt command messagebox will open
```

Step 12: making buttons and entry boxes on the login page. Connecting them with previous methods like signup and exit using “command” attribute in the syntax, before closing the login mainloop window:

```
Intro_login = Label(login, text="Welcome to Movie Recommender! Please login.", fg='blue',
                    font=("arial", 30, "bold")).pack()
# the head-label of the login page
# pack essentially means printing in the centre of the page
```

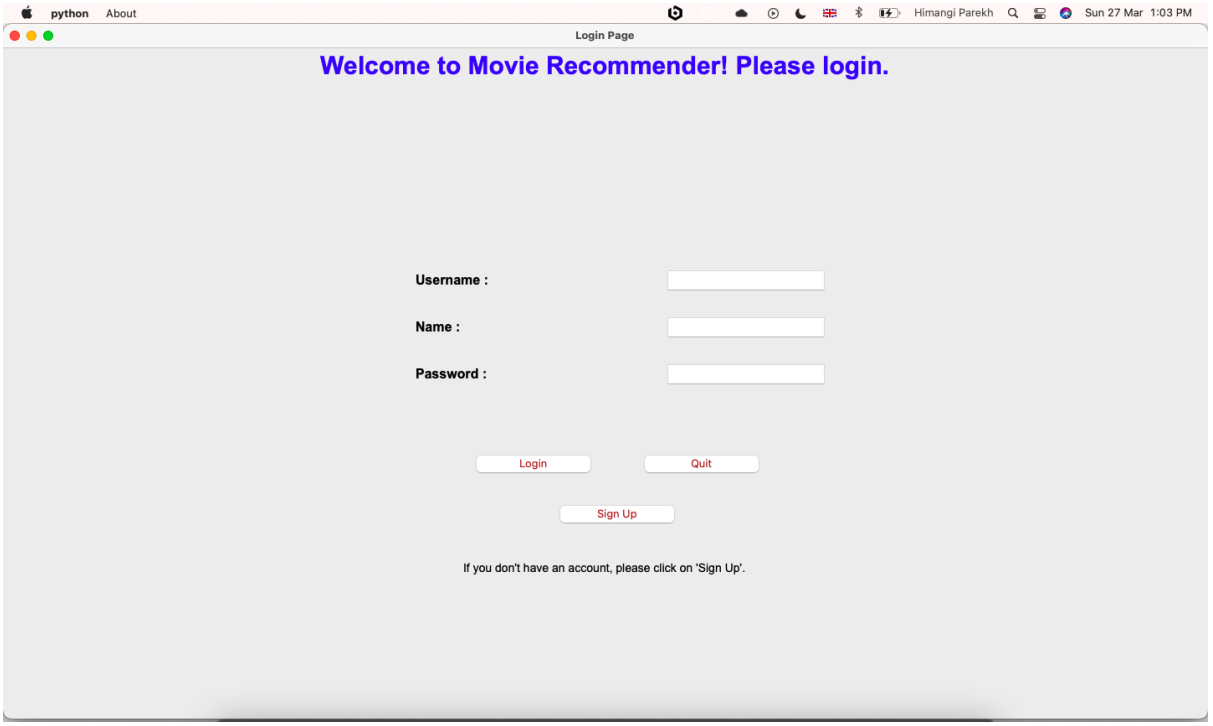
```
userName_login = Label(login, text="Username : ", font=("arial", 16, "bold"))
# create a first name label for login page
userName_login.place(relx=0.34, rely=0.33)

entry_userName_login = Entry(login, textvar=un_log)
# entrybox for first name login page
entry_userName_login.place(relx=0.55, rely=0.33)

Name_login = Label(login, text="Name : ", font=("arial", 16, "bold"))
# last name label login page
Name_login.place(relx=0.34, rely=0.4)

entry_Name_login = Entry(login, textvar=name_log)
# entry box last name login page
entry_Name_login.place(relx=0.55, rely=0.4)

Password_login = Label(login, text="Password : ", font=("arial", 16, "bold"))
# password label login page
Password_login.place(relx=0.34, rely=0.47)

entry_password_login = Entry(login, textvar=ps_log, show='*')
# entry box password login page
entry_password_login.place(relx=0.55, rely=0.47)

Quit_button_login = Button(login, text="Quit", width=12, fg='brown', command=login.quit)
# quit button login page
Quit_button_login.place(relx=0.53, rely=0.6)

Signup_instruction = Label(login, text="If you don't have an account, please click on 'Sign Up'.", fg='black',
                           font=("arial", 14)).place(relx=0.38, rely=0.76)
# signup text on login page

signUp_button = Button(login, text="Sign Up", width=12, fg='brown', command=signup_method)
# signup button login page (command which will open signup window)
signUp_button.place(relx=0.46, rely=0.675)

login_button_login = Button(login, text="Login", width=12, fg='brown', command=login_database)
# login button on login page (which will open homepage)
login_button_login.place(relx=0.39, rely=0.6)
```

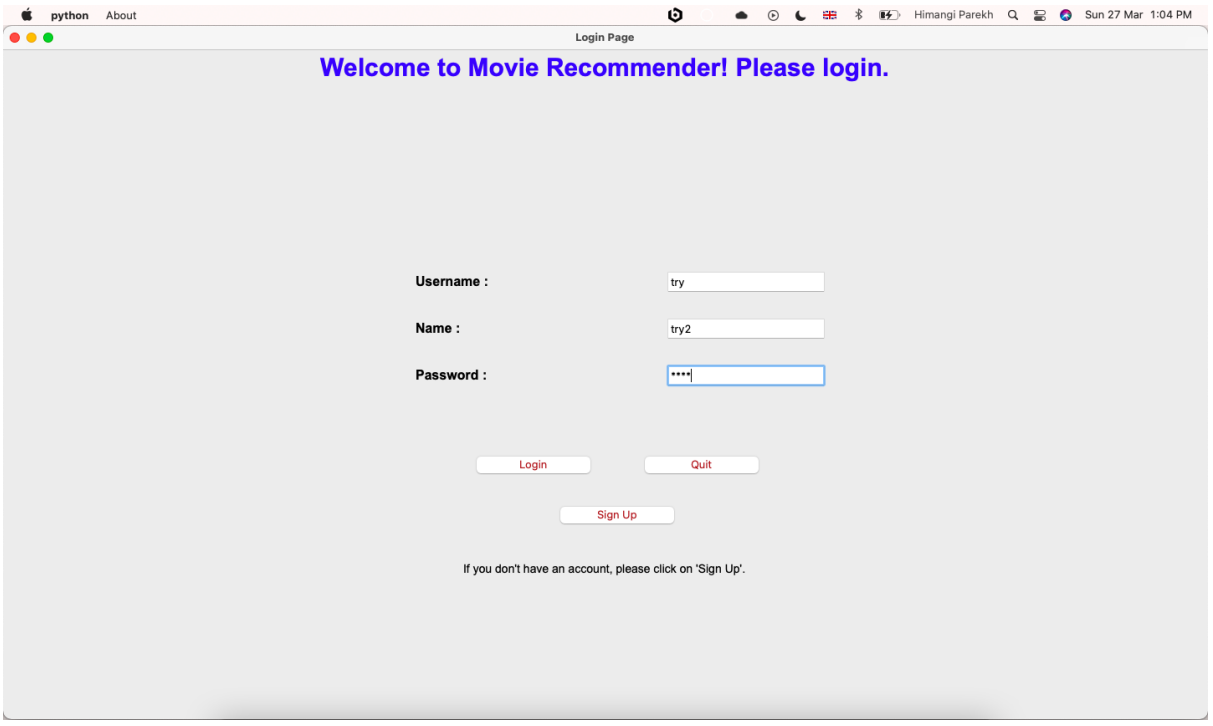
```
login.mainloop()
exit(0)
```

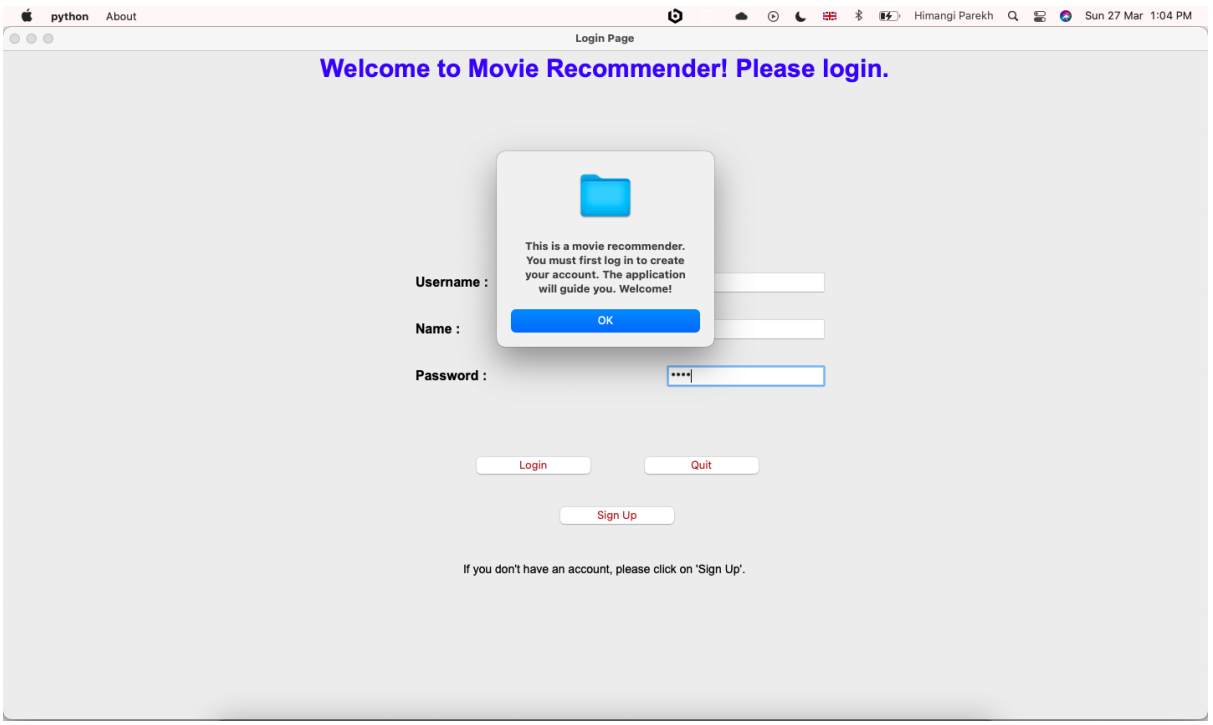
Citing Skit-learn:

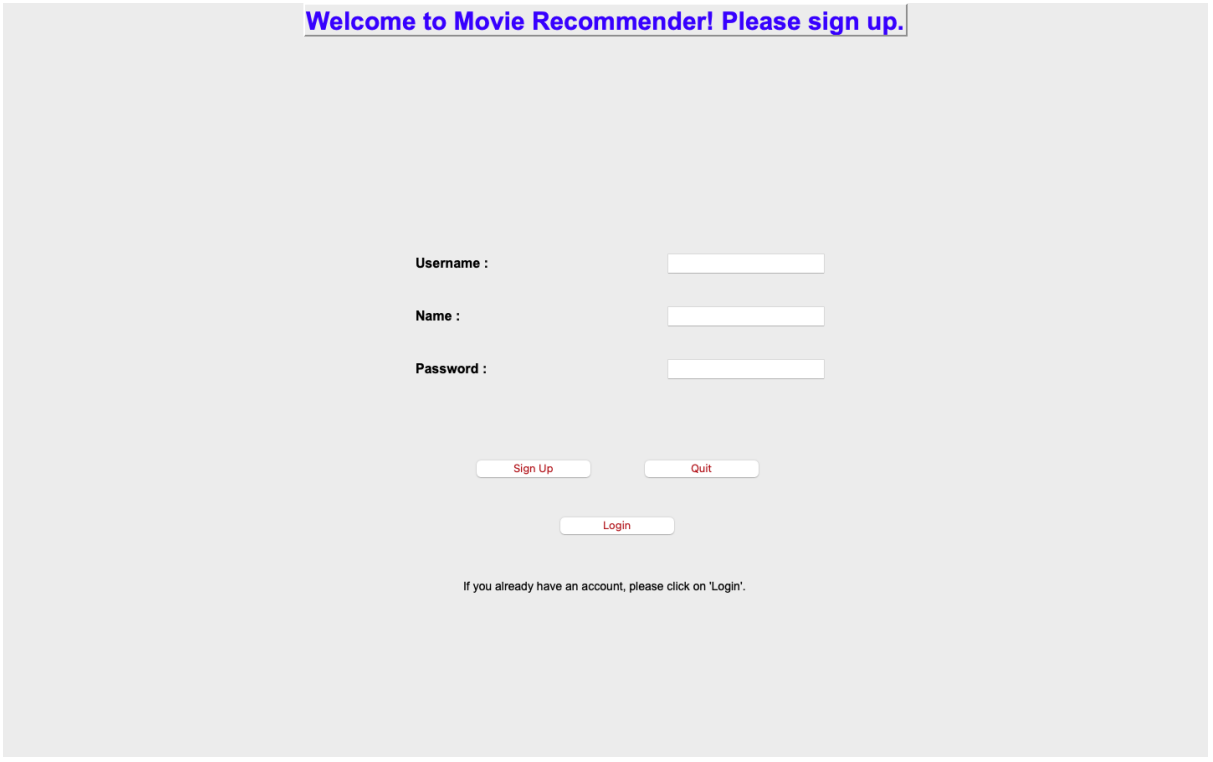
```
@article{scikit-learn,
    title={Scikit-learn: Machine Learning in {P}ython},
    author={Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V.
            and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P.
            and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and
            Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E.},
    journal={Journal of Machine Learning Research},
    volume={12},
    pages={2825--2830},
    year={2011}
   }
```

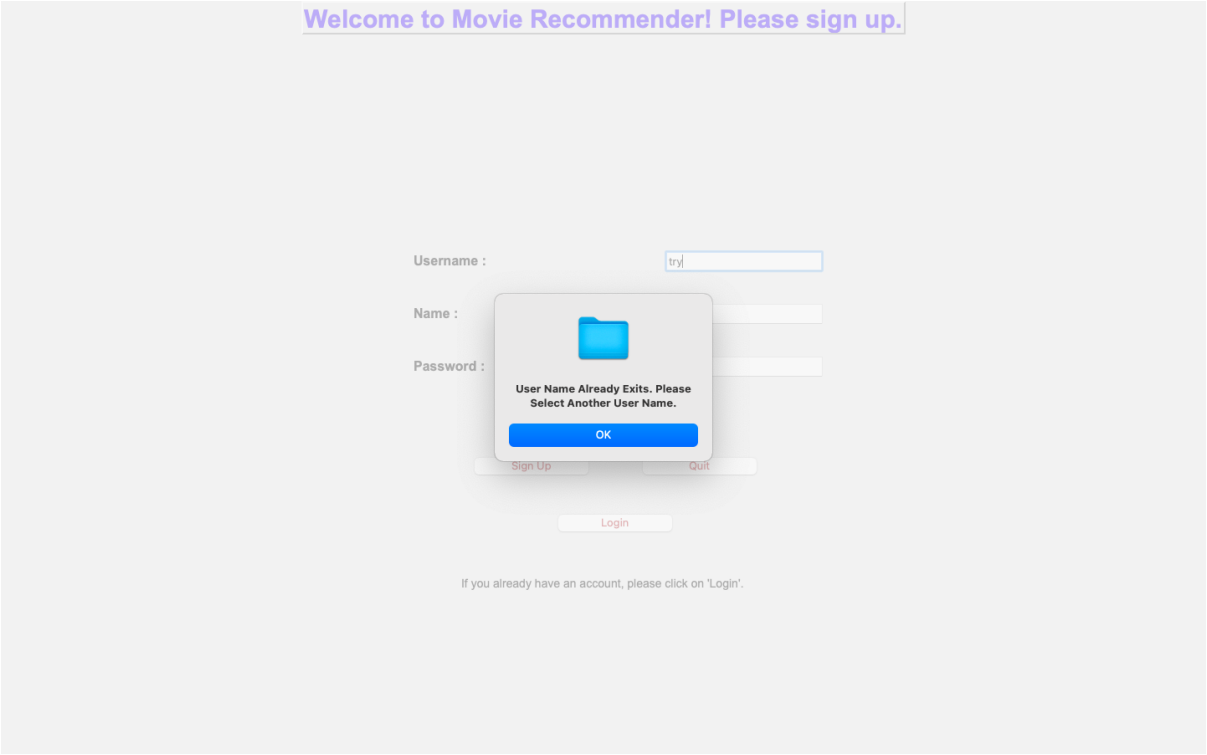
Output

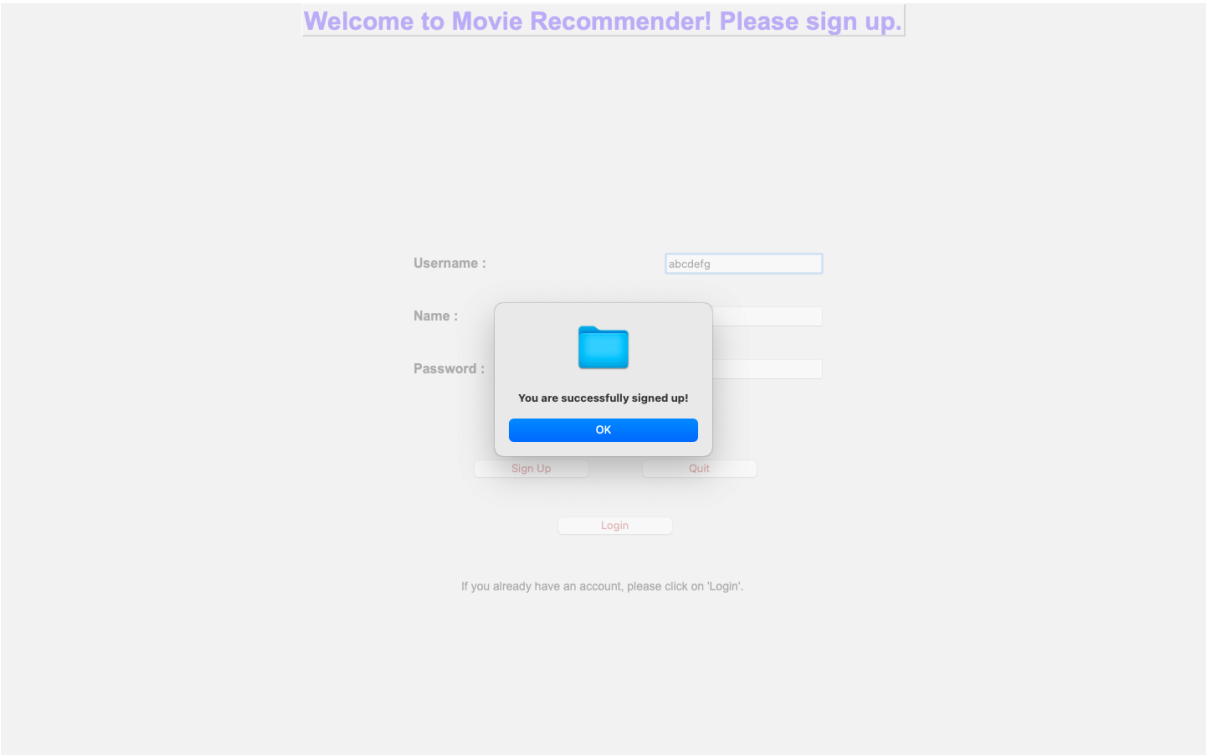

python About
Himangi Parekh
Sun 27 Mar 1:03 PM
Login Page
Welcome to Movie Recommender! Please login.
Username :
Name :
Password :
Login
Quit
Sign Up
If you don't have an account, please click on 'Sign Up'.

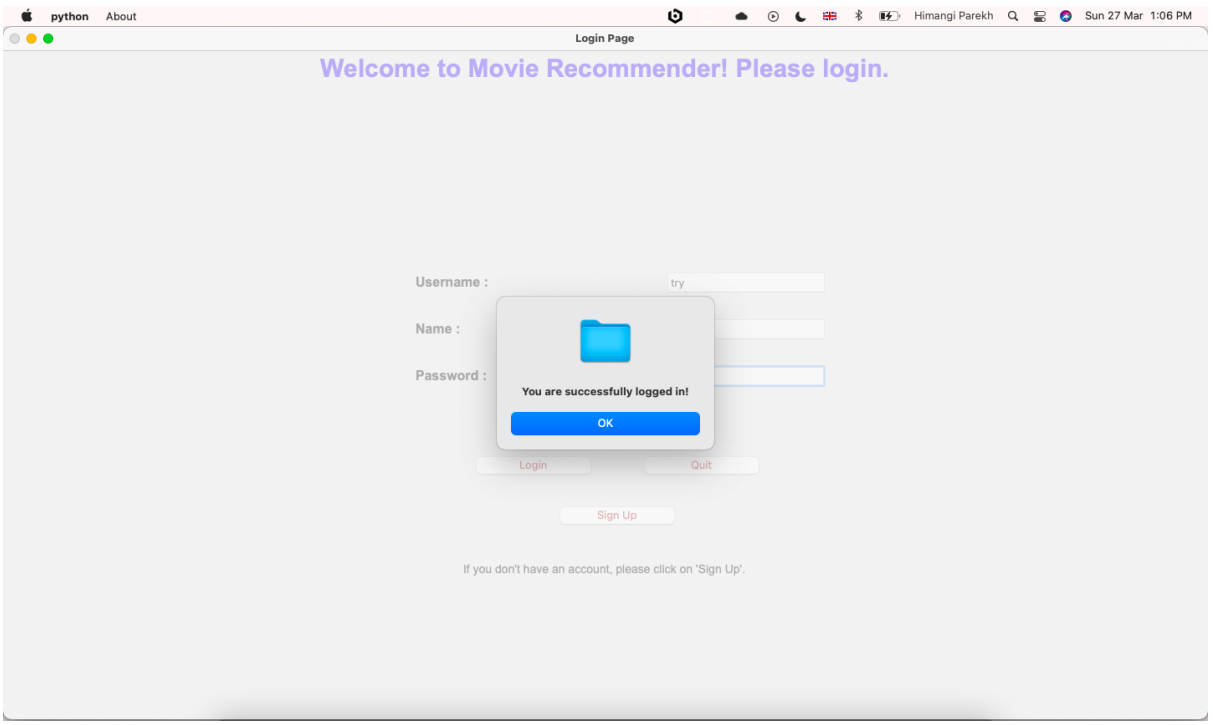

python About
Himangi Parekh
Sun 27 Mar 1:04 PM
Login Page
Welcome to Movie Recommender! Please login.
Username :
try
Name :
try2
Password :
Login
Quit
Sign Up
If you don't have an account, please click on 'Sign Up'.

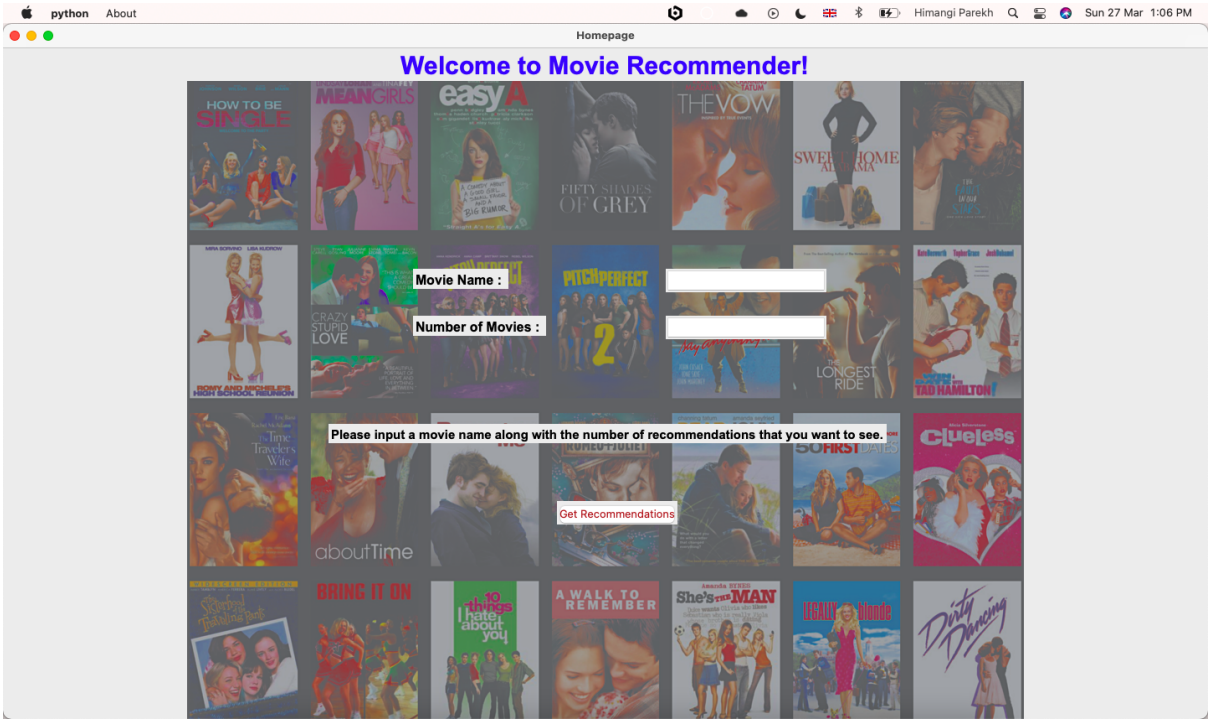

python About
Himangi Parekh
Sun 27 Mar 1:04 PM
Login Page
Welcome to Movie Recommender! Please login.
This is a movie recommender. You must first log in to create your account. The application will guide you. Welcome!
OK
Username :
Name :
Password :
Login
Quit
Sign Up
If you don't have an account, please click on 'Sign Up'.

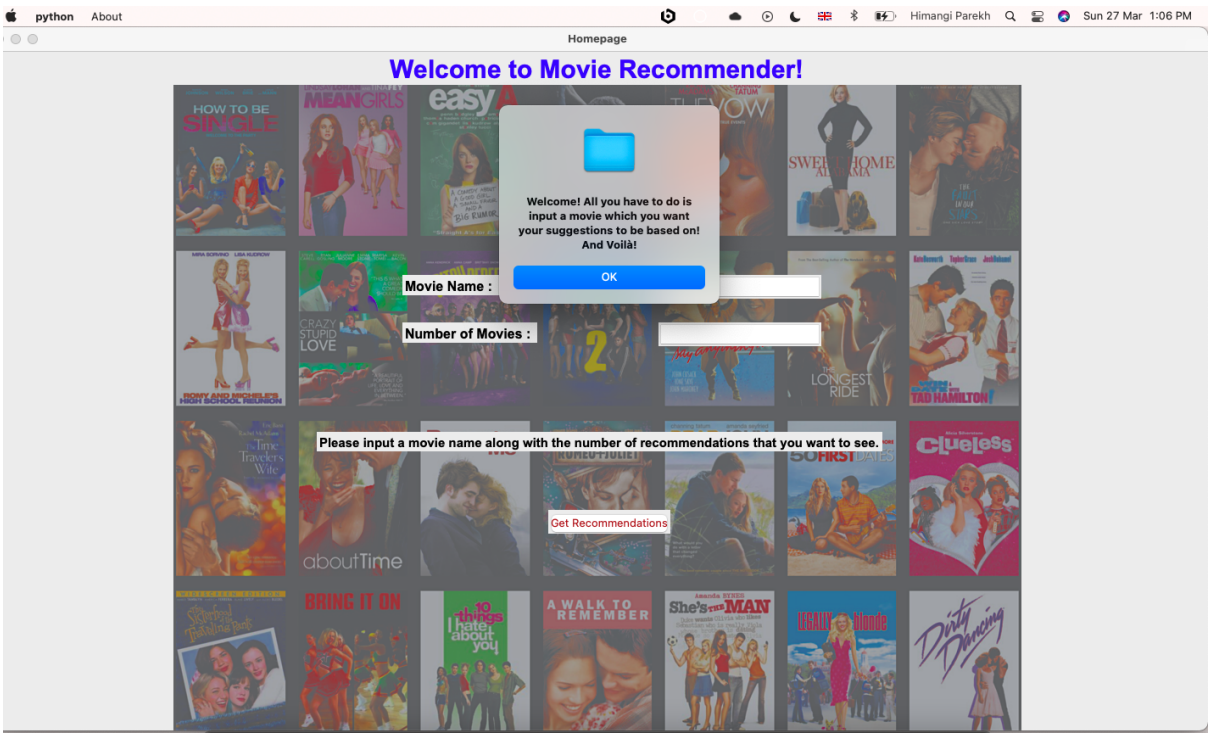

Welcome to Movie Recommender! Please sign up.
Username :
Name :
Password :
Sign Up
Quit
Login
If you already have an account, please click on 'Login'.

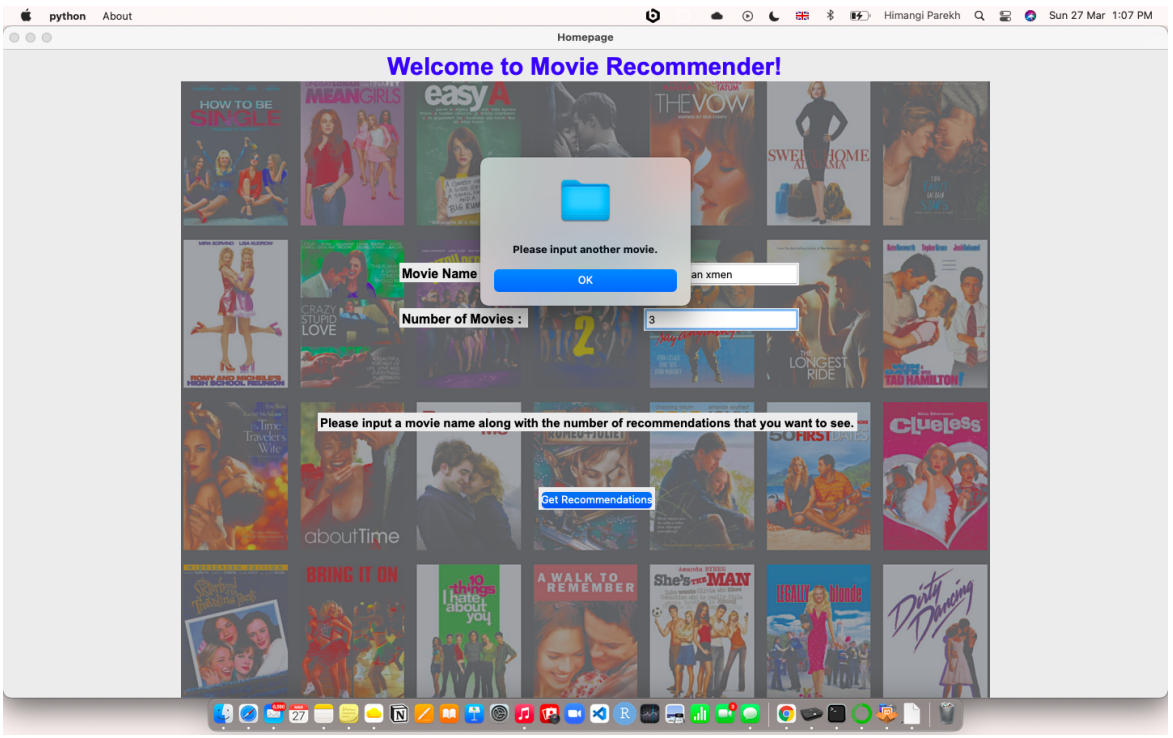

Welcome to Movie Recommender! Please sign up.
Username :
try
Name :
Password :
User Name Already Exits. Please Select Another User Name.
OK
Sign Up
Quit
Login
If you already have an account, please click on 'Login'.

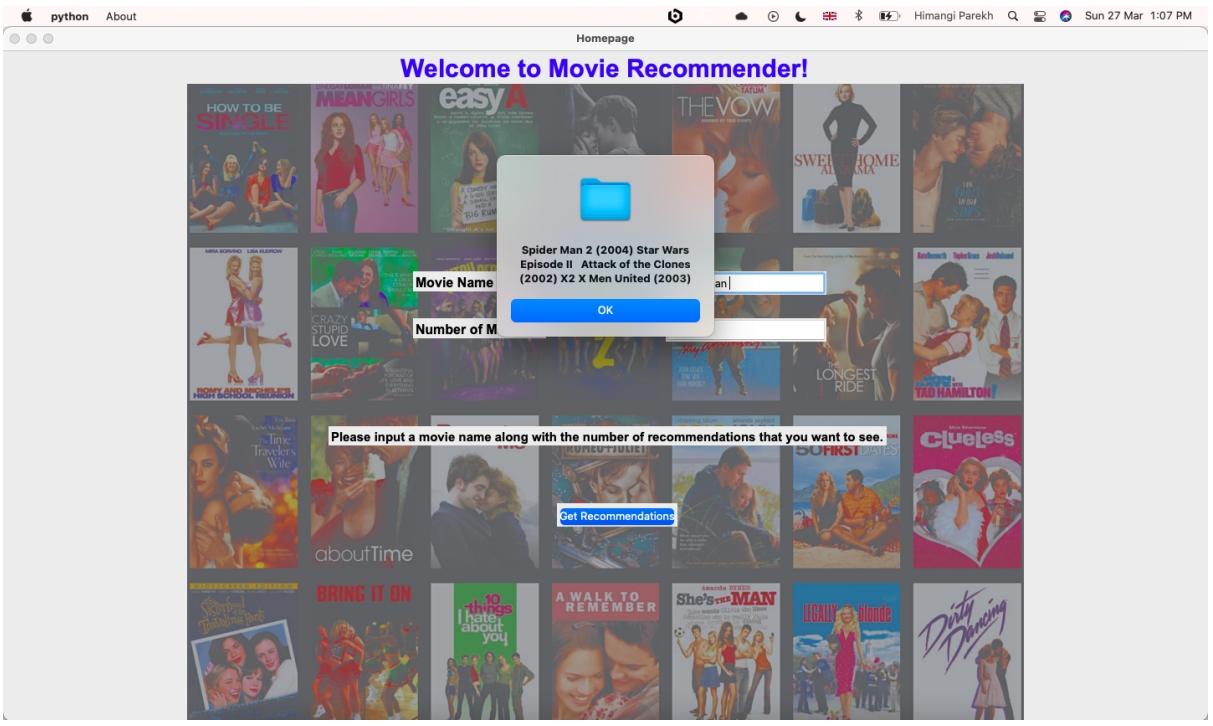

Welcome to Movie Recommender! Please sign up.
Username :
abcdefg
Name :
Password :
You are successfully signed up!
OK
Sign Up
Quit
Login
If you already have an account, please click on 'Login'.


python About
Himangi Parekh
Sun 27 Mar 1:06 PM
Login Page
Welcome to Movie Recommender! Please login.
Username :
try
Name :
Password :
You are successfully logged in!
OK
Login
Quit
Sign Up
If you don't have an account, please click on 'Sign Up'.


python About
Himangi Parekh
Sun 27 Mar 1:06 PM
Homepage
Welcome to Movie Recommender!
Movie Name :
Number of Movies :
Please input a movie name along with the number of recommendations that you want to see.
Get Recommendations


python About
Himangi Parekh
Sun 27 Mar 1:06 PM
Homepage
Welcome to Movie Recommender!
Welcome! All you have to do is input a movie which you want your suggestions to be based on! And Voilà!
OK
Movie Name :
Number of Movies :
Please input a movie name along with the number of recommendations that you want to see.
Get Recommendations


python About
Himangi Parekh
Sun 27 Mar 1:07 PM
Homepage
Welcome to Movie Recommender!
Please input another movie.
OK
Movie Name
an xmen
Number of Movies :
3
Please input a movie name along with the number of recommendations that you want to see.
Get Recommendations


python About
Himangi Parekh
Sun 27 Mar 1:07 PM
Homepage
Welcome to Movie Recommender!
Spider Man 2 (2004) Star Wars Episode II Attack of the Clones (2002) X2 X Men United (2003)
OK
Movie Name
Number of M
Please input a movie name along with the number of recommendations that you want to see.
Get Recommendations

Matrix/Ratings database:

userKey	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609
movieKey																				
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193583	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193585	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193587	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193609	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

9724 rows × 610 columns

User database: (php myadmin: Xampp)

Username	Name	Password
try	try2	try3
try	try2	try3
hello2	hello3	heloo4
hello7	797	687987
hello9	797	687987
hello799	797	687987
himangi5715	himangiparekh	himangi@5715
himangi	parekh	himangiparekh
hiimangi	parekkh	himangiiii
hellooooo	helo9891	hidefojfe2
hello4567	hello	hiii
please	work	defrgkn
why	ehy3	heyfojrf
hiiiiiiiiii	himangi	parekhparekh
why7897	whydrpfk	heyfojnvf
why7897y76	whydrpfk	heyfojnvf
why7897y76uyu	whydrpfk	heyfojnvf
geet123	Geet Parekh	Geet@1234
harsha78	harsha parekh	harsha@123
shraavsi	Shrawani	hello123A
user1	user2	user3
user5	user6	user7
abcdefg	hello3	hello4

Citations

Scikit-learn:

```
@article{scikit-learn,
    title={Scikit-learn: Machine Learning in {P}ython},
    author={Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V.
            and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P.
            and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and
            Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E.},
    journal={Journal of Machine Learning Research},
    volume={12},
    pages={2825--2830},
    year={2011}
   }
```

Source: Nearest Neighbours, 2022, Scikit Learn, *KNN Algorithm*, https://scikit-learn.org/stable/modules/neighbors.html

Source: Small Movie Lens Dataset from Kaggle Datasets, 2018: *Movie Lens Small Latest Dataset,* Shubham Mehta, https://www.kaggle.com/datasets/shubhammehta21/movie-lens-small-latest-dataset