Artificial Intelligence & Machine Learning Assignment AL23431 (Batch-02)



**Submitted by: Antonysachin S**

**Regno: 2117230040011**

**Course: Artificial Intelligence & Machine Learning**

**Class: ECE-A 2nd batch**

**Date: June 16, 2025**

8. (i)Movie Recommender:

**Program:**

import pandas as pd

import tkinter as tk

from tkinter import ttk

from PIL import Image, ImageTk

import io, requests

# ── Data ──────────────────────────────────────────────────────────────────────

df = pd.read\_csv(r"C:\Users\anton\OneDrive\Documents\bhaves\imdb\_top\_1000.csv")

df = df.dropna(subset=["Series\_Title", "Genre", "Released\_Year", "IMDB\_Rating", "Poster\_Link"])

df["Released\_Year"] = pd.to\_numeric(df["Released\_Year"], errors="coerce")

df["IMDB\_Rating"] = pd.to\_numeric(df["IMDB\_Rating"], errors="coerce")

# ── Theme colors ─────────────────────────────────────────────────────────────

BG\_MAIN      = "#0d0d16"   # deep charcoal

BG\_PANEL     = "#1a1a28"

BG\_CARD      = "#222233"

FG\_TEXT      = "#e5e5e5"

NEON\_GREEN   = "#39ff14"

NEON\_CYAN    = "#00e5ff"

NEON\_PINK    = "#ff41ff"

FONT\_SANS = ("Segoe UI", 10)

FONT\_BOLD = ("Segoe UI", 10, "bold")

# ── GUI root & style ─────────────────────────────────────────────────────────

root = tk.Tk()

root.title("🎬 Movie Recommender")

root.geometry("750x620")

root.configure(bg=BG\_MAIN)

style = ttk.Style()

style.theme\_use("clam")  # neutral base

style.configure(".", background=BG\_MAIN, foreground=FG\_TEXT, font=FONT\_SANS)

style.configure("Accent.TButton",

                background=NEON\_CYAN,

                foreground=BG\_MAIN,

                font=FONT\_BOLD,

                padding=6,

                borderwidth=0)

style.map("Accent.TButton",

          background=[("active", NEON\_PINK), ("pressed", NEON\_GREEN)])

style.configure("TLabel", background=BG\_MAIN, foreground=FG\_TEXT)

style.configure("Card.TLabel", background=BG\_CARD, foreground=FG\_TEXT, anchor="w")

style.configure("Custom.TEntry",

                fieldbackground=BG\_PANEL,

                foreground=FG\_TEXT,

                padding=4,

                relief="flat")

# ── State vars ───────────────────────────────────────────────────────────────

genre\_var  = tk.StringVar()

year\_var   = tk.StringVar()

rating\_var = tk.StringVar()

filtered\_df = pd.DataFrame()

recommend\_index = 0

# ── Auto‑suggest helpers ─────────────────────────────────────────────────────

def update\_suggestions(var, suggestion\_box, column):

    value = var.get().lower()

    suggestions = sorted(set(df[column].dropna()))

    matches = [s for s in suggestions if value in str(s).lower()]

    suggestion\_box.delete(0, tk.END)

    for m in matches[:10]:

        suggestion\_box.insert(tk.END, m)

def on\_select\_suggestion(var, box):

    var.set(box.get(tk.ACTIVE))

    box.pack\_forget()

def suggest\_entry(frame, label\_text, var, column):

    ttk.Label(frame, text=label\_text + ":").pack(anchor="w", pady=(4,0))

    entry = ttk.Entry(frame, textvariable=var, style="Custom.TEntry")

    entry.pack(fill="x")

    box = tk.Listbox(frame, height=5, bg=BG\_PANEL, fg=FG\_TEXT,

                     selectbackground=NEON\_CYAN, activestyle="none",

                     borderwidth=0, highlightthickness=0)

    box.pack(fill="x")

    box.pack\_forget()

    entry.bind("<KeyRelease>",

               lambda e: (box.pack(fill="x"),

                          update\_suggestions(var, box, column)))

    box.bind("<<ListboxSelect>>",

             lambda e: on\_select\_suggestion(var, box))

# ── Recommendation logic ─────────────────────────────────────────────────────

def recommend\_movies():

    global filtered\_df, recommend\_index

    genre  = genre\_var.get().lower()

    rating = rating\_var.get()

    year\_input = year\_var.get()

    try:

        rating = float(rating)

    except:

        rating = 0

    if "-" in year\_input:

        y\_start, y\_end = map(int, year\_input.split("-"))

    else:

        try:

            y\_start = y\_end = int(year\_input)

        except:

            y\_start, y\_end = 1900, 2100

    filtered\_df = df[

        df["Genre"].str.lower().str.contains(genre) &

        (df["IMDB\_Rating"] >= rating) &

        (df["Released\_Year"].between(y\_start, y\_end))

    ]

    recommend\_index = 0

    show\_next\_movies()

def show\_next\_movies():

    global recommend\_index

    for widget in result\_frame.winfo\_children():

        widget.destroy()

    batch = filtered\_df.iloc[recommend\_index:recommend\_index + 3]

    if batch.empty:

        ttk.Label(result\_frame, text="No more recommendations!", style="Card.TLabel",

                  font=FONT\_BOLD).pack(pady=10)

        return

    for \_, row in batch.iterrows():

        card = tk.Frame(result\_frame, bg=BG\_CARD, bd=0, pady=6, padx=6)

        card.pack(fill="x", padx=10, pady=8)

        # Poster

        try:

            response = requests.get(row["Poster\_Link"])

            img\_data = response.content

            img = Image.open(io.BytesIO(img\_data)).resize((90, 135))

            photo = ImageTk.PhotoImage(img)

            img\_label = tk.Label(card, image=photo, bg=BG\_CARD)

            img\_label.image = photo

            img\_label.pack(side="left", padx=(0, 10))

        except:

            pass

        # Details

        info = (f"{row['Series\_Title']} ({int(row['Released\_Year'])})\n"

                f"Genre:  {row['Genre']}\n"

                f"Rating: {row['IMDB\_Rating']}")

        ttk.Label(card, text=info, style="Card.TLabel", justify="left").pack(anchor="w")

    recommend\_index += 3

# ── Layout ───────────────────────────────────────────────────────────────────

top\_frame = tk.Frame(root, bg=BG\_MAIN, padx=12, pady=12)

top\_frame.pack(fill="x")

suggest\_entry(top\_frame, "Genre", genre\_var, "Genre")

suggest\_entry(top\_frame, "Release Year (e.g., 2000 or 1990‑2010)", year\_var, "Released\_Year")

suggest\_entry(top\_frame, "Min IMDB Rating", rating\_var, "IMDB\_Rating")

ttk.Button(top\_frame, text="🎯 Recommend", style="Accent.TButton",

           command=recommend\_movies).pack(pady=8, fill="x")

ttk.Button(top\_frame, text="📩 More Suggestions", style="Accent.TButton",

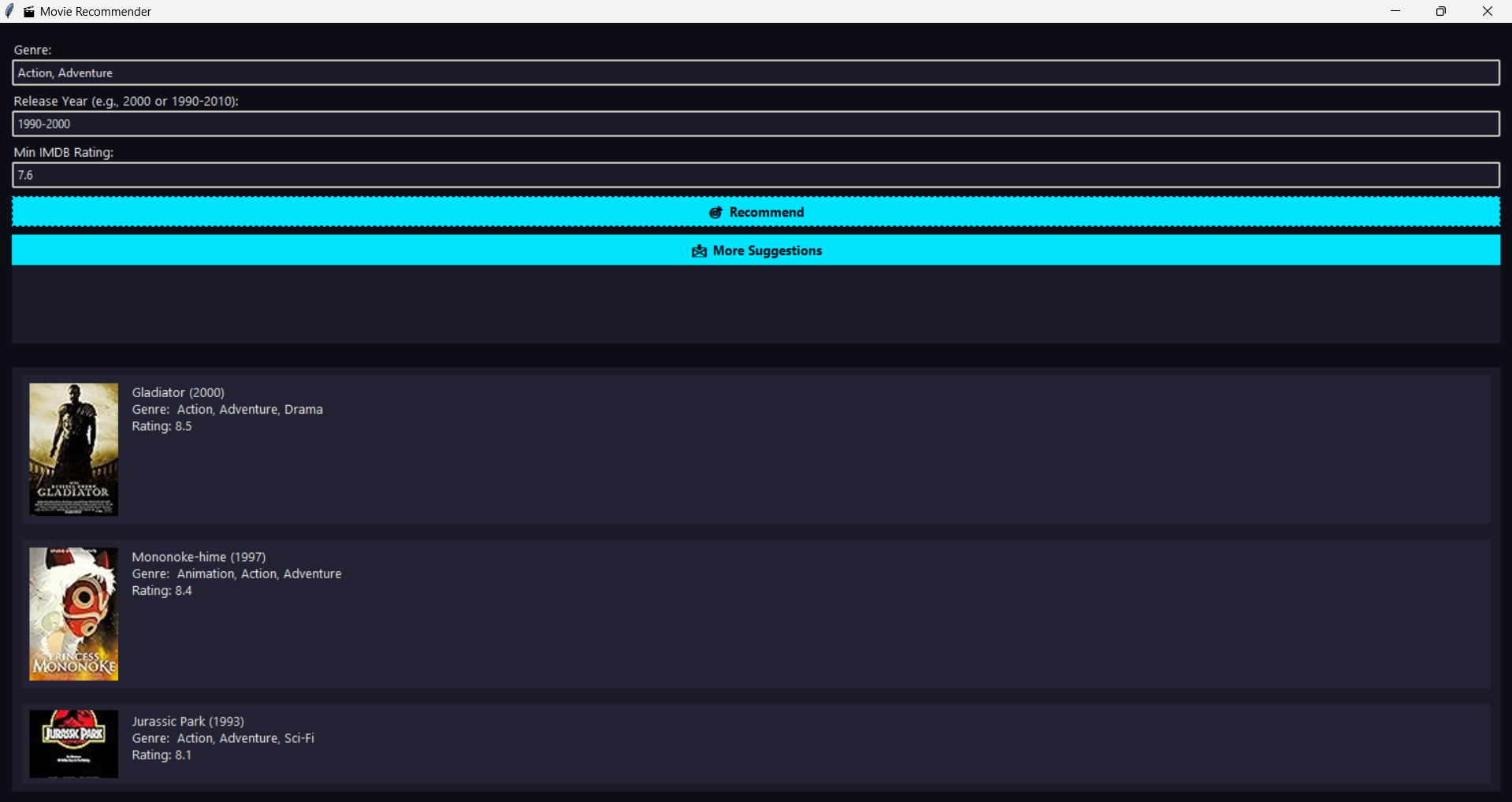
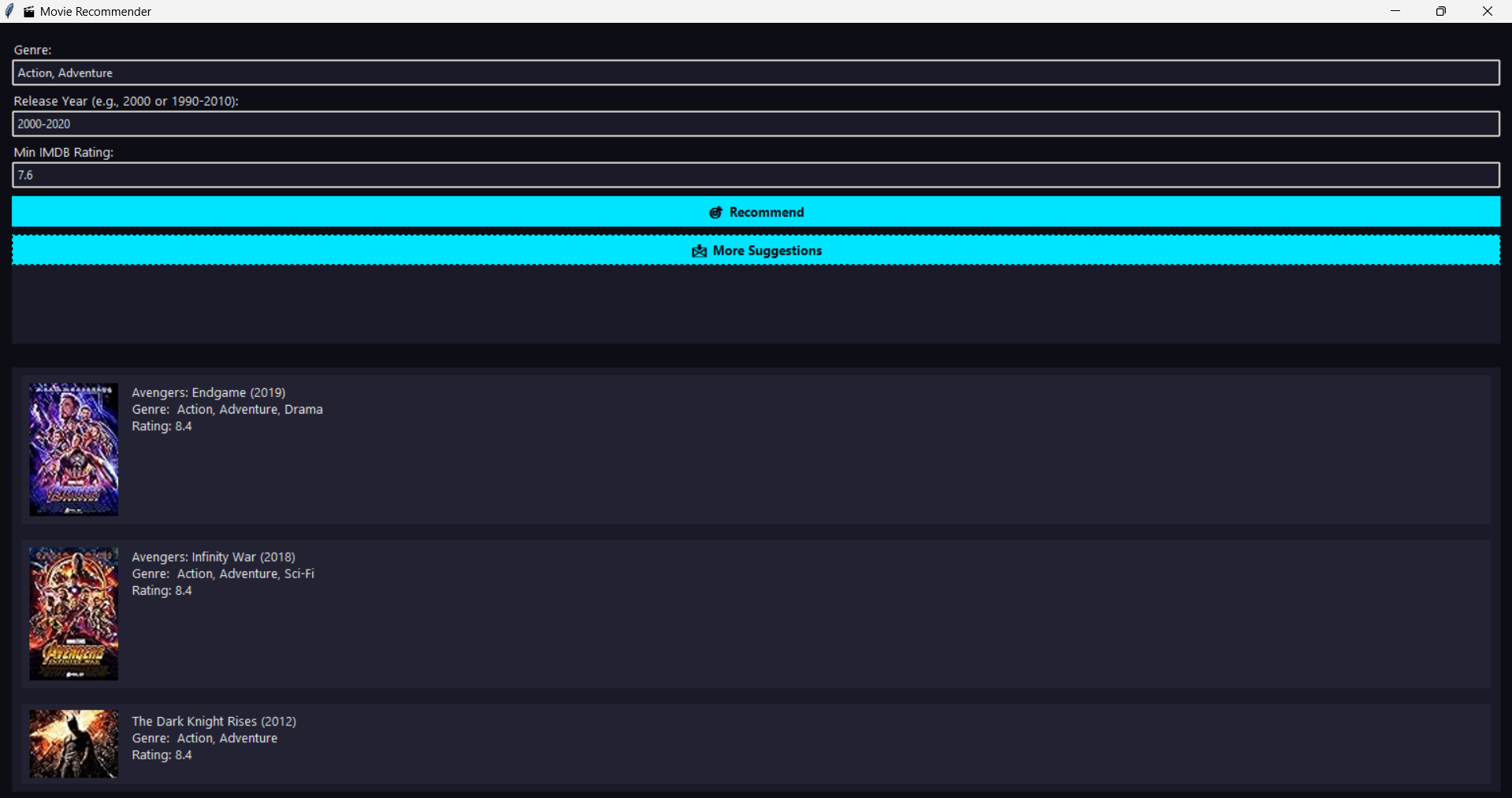
           command=show\_next\_movies).pack(fill="x")

result\_frame = tk.Frame(root, bg=BG\_PANEL)

result\_frame.pack(fill="both", expand=True, padx=12, pady=12)

root.mainloop()

**Output:**

****

**10.(i) Tic-Tac-Toe game:**

**Program:**

import tkinter as tk

from tkinter import messagebox

import math

# Initialize the game board

board = [' ' for \_ in range(9)]

# Check for winner

def check\_winner(b, player):

    win\_combinations = [

        [0,1,2], [3,4,5], [6,7,8],

        [0,3,6], [1,4,7], [2,5,8],

        [0,4,8], [2,4,6]

    ]

    return any(all(b[i] == player for i in combo) for combo in win\_combinations)

# Check for draw

def is\_draw():

    return ' ' not in board

# Get all valid moves

def available\_moves():

    return [i for i in range(9) if board[i] == ' ']

# Minimax Algorithm

def minimax(b, depth, is\_maximizing):

    if check\_winner(b, 'O'):

        return 1

    elif check\_winner(b, 'X'):

        return -1

    elif is\_draw():

        return 0

    if is\_maximizing:

        best\_score = -math.inf

        for move in available\_moves():

            b[move] = 'O'

            score = minimax(b, depth + 1, False)

            b[move] = ' '

            best\_score = max(score, best\_score)

        return best\_score

    else:

        best\_score = math.inf

        for move in available\_moves():

            b[move] = 'X'

            score = minimax(b, depth + 1, True)

            b[move] = ' '

            best\_score = min(score, best\_score)

        return best\_score

# Computer's move using Minimax

def computer\_move():

    best\_score = -math.inf

    best\_move = None

    for i in available\_moves():

        board[i] = 'O'

        score = minimax(board, 0, False)

        board[i] = ' '

        if score > best\_score:

            best\_score = score

            best\_move = i

    board[best\_move] = 'O'

    buttons[best\_move].config(text='O', state='disabled')

    check\_game\_over()

# Handle player's move

def on\_button\_click(index):

    if board[index] == ' ':

        board[index] = 'X'

        buttons[index].config(text='X', state='disabled')

        check\_game\_over()

        if not check\_winner(board, 'X') and not is\_draw():

            computer\_move()

# Check game result

def check\_game\_over():

    if check\_winner(board, 'X'):

        messagebox.showinfo("Game Over", "You Win! 🎉")

        reset\_game()

    elif check\_winner(board, 'O'):

        messagebox.showinfo("Game Over", "Computer Wins! 🤖")

        reset\_game()

    elif is\_draw():

        messagebox.showinfo("Game Over", "It's a Draw!")

        reset\_game()

# Reset the game

def reset\_game():

    global board    board = [' ' for \_ in range(9)]

    for button in buttons:

        button.config(text=' ', state='normal')

# Tkinter GUI setup

root = tk.Tk()

root.title("Tic-Tac-Toe - Minimax AI")

buttons = []

for i in range(9):

    btn = tk.Button(root, text=' ', font=('Arial', 24), height=2, width=5,

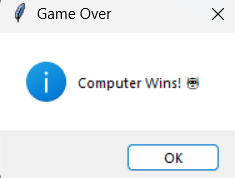
                    command=lambda i=i: on\_button\_click(i))

    btn.grid(row=i//3, column=i%3)

    buttons.append(btn)

root.mainloop()

**Output:**

****