**Codsoft**

**Artificial Intelligence**

**project**

**Name- Sudipta Samanta**

**TASK 2: TIC-TAC-TOE AI**

Implement an AI agent that plays the classic game of Tic-Tac-Toe against a human player. You can use algorithms like Minimax with or without Alpha-Beta Pruning to make the AI player unbeatable. This project will help you understand game theory and basic search algorithms

**Source Code**:

BOARD\_EMPTY = 0

PLAYER\_X = 1

PLAYER\_O = -1

def print\_board(s):

def convert(num):

    if num == PLAYER\_X:

      return 'X'

    if num == PLAYER\_O:

      return 'O'

    return '\_'

  i = 0

  for \_ in range(3):

    for \_ in range(3):

      print(convert(s[i]), end=' ')

      i += 1

    print()

from collections import Counter

def player(s):

  counter = Counter(s)

  x\_places = counter[1]

  o\_places = counter[-1]

  if x\_places + o\_places == 9:

    return None

  elif x\_places > o\_places:

    return PLAYER\_O

  else:

    return PLAYER\_X

def actions(s):

  play = player(s)

  actions\_list = [(play, i) for i in range(len(s)) if s[i] == BOARD\_EMPTY]

  return actions\_list

def result(s, a):

  (play, index) = a

  s\_copy = s.copy()

  s\_copy[index] = play

  return s\_copy

def terminal(s):

  for i in range(3):

    # Checking if a row is filled and equal.

    if s[3 \* i] == s[3 \* i + 1] == s[3 \* i + 2] != BOARD\_EMPTY:

      return s[3 \* i]

    # Checking if a column is filled and equal.

    if s[i] == s[i + 3] == s[i + 6] != BOARD\_EMPTY:

      return s[i]

  # Checking if a diagonal is filled and equal.

  if s[0] == s[4] == s[8] != BOARD\_EMPTY:

    return s[0]

  if s[2] == s[4] == s[6] != BOARD\_EMPTY:

    return s[2]

  # Checking if the game has no more moves available

  if player(s) is None:

    return 0

  # Return None if none of the previous conditions satisfy.

  return None

def utility(s):

  term = terminal(s)

  # Return who wins the game if the game has terminated

  if term is not None:

    return term

  # Get the list of actions available

  action\_list = actions(s)

  utils = []

  for action in action\_list:

    # Create a new state applying the action to current state

    new\_s = result(s, action)

    # Add the score of the new state to a list

    utils.append(utility(new\_s))

  score = utils[0]

  play = player(s)

  # Calculate the max score if X is playing

  if play == PLAYER\_X:

    for i in range(len(utils)):

      if utils[i] > score:

        score = utils[i]

  # Calculate the min score if O is playing

  else:

    for i in range(len(utils)):

      if utils[i] < score:

        score = utils[i]

  return score

def utility(s, cost):

  term = terminal(s)

  if term is not None:

    # Return the cost of reaching the terminal state

    return (term, cost)

  action\_list = actions(s)

  utils = []

  for action in action\_list:

    new\_s = result(s, action)

    # Every recursion will be an increment in cost

    utils.append(utility(new\_s, cost + 1))

  # Remember the associated cost with the score of the state.

  score = utils[0][0]

  idx\_cost = utils[0][1]

  play = player(s)

  if play == PLAYER\_X:

    for i in range(len(utils)):

     if utils[i][0] > score:

       score = utils[i][0]

       idx\_cost = utils[i][1]

  else:

    for i in range(len(utils)):

      if utils[i][0] < score:

        score = utils[i][0]

        idx\_cost = utils[i][1]

  # Return the score with the associated cost.

  return (score, idx\_cost)

def minimax(s):

  action\_list = actions(s)

  utils = []

  for action in action\_list:

    new\_s = result(s, action)

    utils.append((action, utility(new\_s, 1)))

  # the score and "cost" of that action.

  if len(utils) == 0:

    return ((0, 0), (0, 0))

  # Sort the list in ascending order of cost.

  sorted\_list = sorted(utils, key=lambda l : l[0][1])

  # Since the computer shall be Player O,

  # It is safe to return the object with minimum score.

  action = min(sorted\_list, key = lambda l : l[1])

  return action

if \_\_name\_\_ == '\_\_main\_\_':

  # Initializing the state

  s = [BOARD\_EMPTY for \_ in range(9)]

  print('|------- Welcome to Tic Tac Toe! -----------|')

  print('You are X while the Computer is O. Lets play!\n')

  # Run the program while the game is not terminated

  while terminal(s) is None:

    play = player(s)

    if play == PLAYER\_X:

      # Take input from user

      print('\n\nIt is your turn', end='\n\n')

      x = int(input('Enter the x-coordinate [0-2]: '))

      y = int(input('Enter the y-coordinate [0-2]: '))

      index = 3 \* x + y

      if not s[index] == BOARD\_EMPTY:

        print('Oops! That coordinate is already taken. Try again.\n')

        continue

      # Apply the action and print the board

      s = result(s, (PLAYER\_X, index))

      print\_board(s)

    else:

      print('\n\nThe is computer is playing its turn')

      # Get the action by running the minimax algorithm

      action = minimax(s)

      # Apply the returned action to the state and print the board

      s = result(s, action[0])

      print\_board(s)

  # determine the winner

  winner = terminal(s)

  if winner == PLAYER\_X:

    print("You have won!")

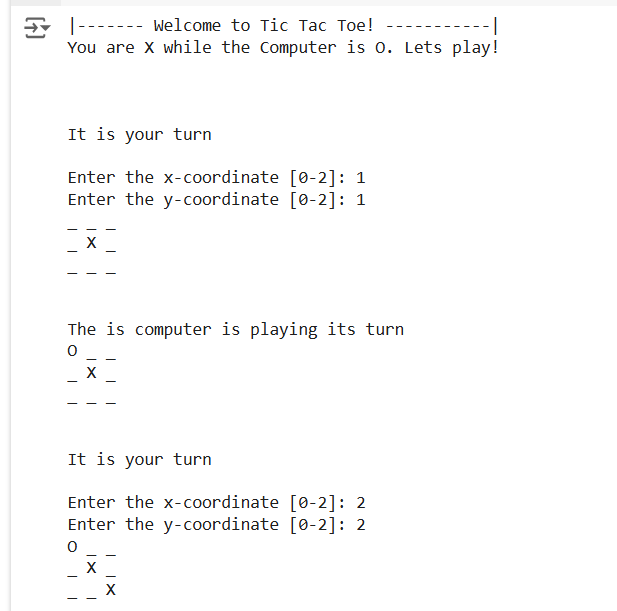
  elif winner == PLAYER\_O:

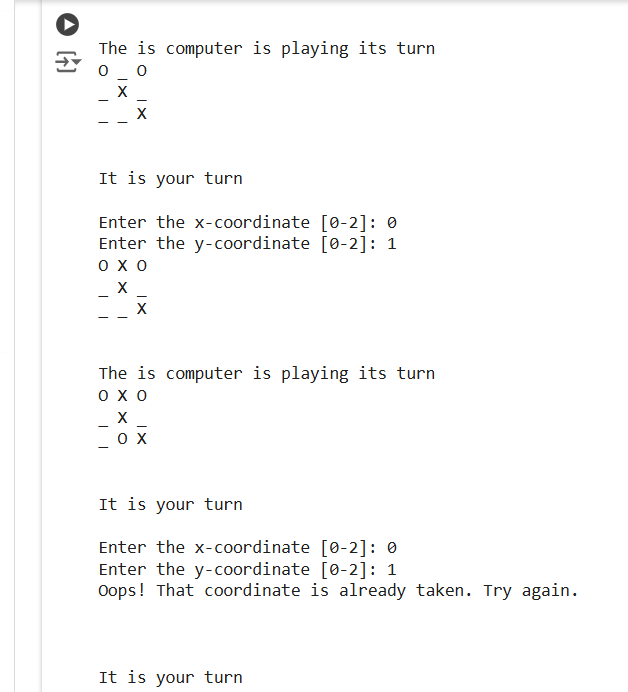
    print("You have lost!")

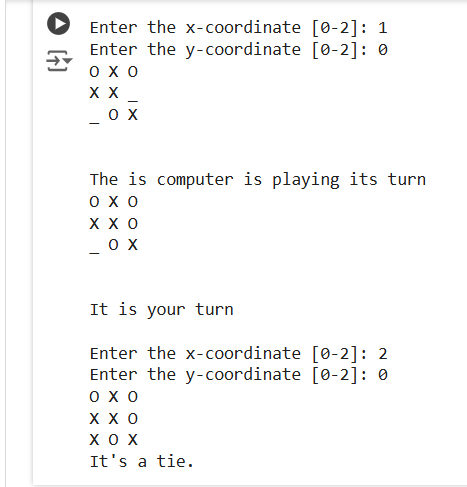
  else:

    print("It's a tie.")

**Output:**







**TASK 1: CHATBOT WITH RULE-BASED RESPONSES**

Build a simple chatbot that responds to user inputs based on predefined rules. Use if-else statements or pattern matching techniques to identify user queries and provide appropriate responses. This will give you a basic understanding of natural language processing and conversation flow

**Source Code**:

# Install necessary libraries

!pip install nltk

# Import necessary modules

import nltk

import re

from nltk.chat.util import Chat, reflections

# Download NLTK data

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

# Define patterns and responses

pairs = [

    [r"My name is (.\*)", ["Hello %1, nice to meet you, how can I assist you today?",]],

    [r"Hi|Hey|Hello", ["Hello, how can I help you?", "Hi! How can I assist you today?",]],

    [r"What is your name?", ["My name is chatbot created to assist you. what's on your mind today?",]],

    [r"How are you?", ["I'm good! what about you? ",]],

    [r"Can you help me with (.\*)", ["Sure, I can help you with %1. Please provide more details.",]],

    [r"Thankyou|Thanks", ["You're welcome!", "Happy to help you! Let me know if you need anything else.",]],

    [r"Bye", ["Bye! Have a nice day!", "Goodbye!",]],

    [r"What is your purpose?", ["I'am here to assist with your queries, provide information according to the pattern I have been created with!",]],

    [r"(.\*)", ["I'm sorry, I don't understand that. Could you clarify?",]],

]

# Define the chatbot class

class RuleBasedChatbot:

    def \_\_init\_\_(self, pairs):

        self.chat = Chat(pairs, reflections)

    def respond(self, user\_input):

        return self.chat.respond(user\_input)

# Initialize the chatbot

chatbot = RuleBasedChatbot(pairs)

# Function to chat with the bot

def chat\_with\_bot():

    print("Hi, I'm your chatbot. Type 'exit' to exit chat with chatbot.")

    while True:

        user\_input = input("You: ")

        if user\_input.lower() == 'exit':

            print("Chatbot: Bye! Have a great day!")

            break

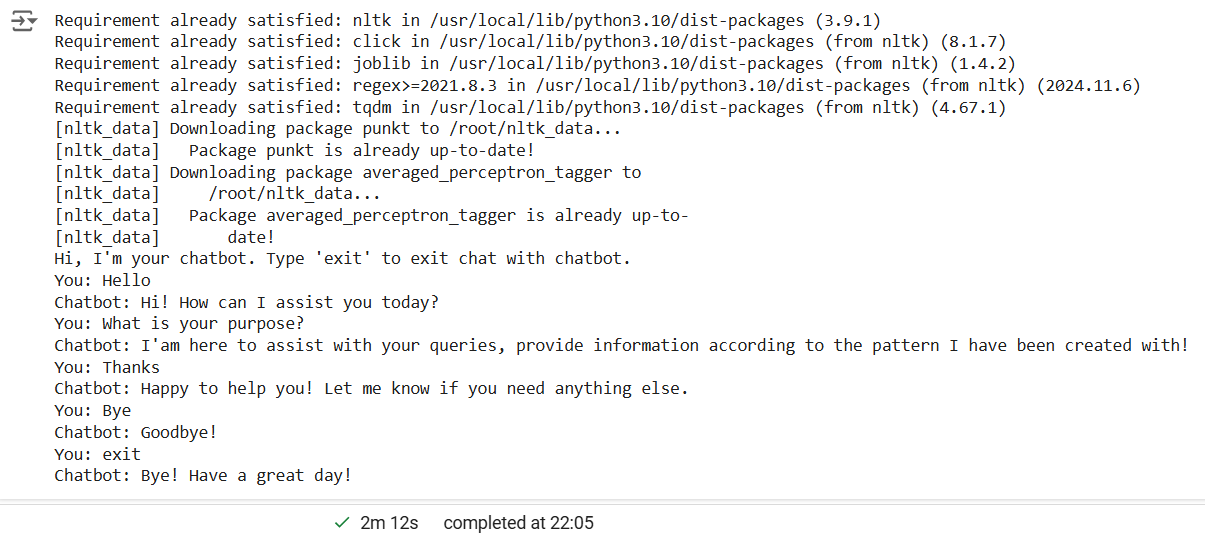
        response = chatbot.respond(user\_input)

        print(f"Chatbot: {response}")

# Start chatting with the bot

chat\_with\_bot()

**Output:**



**TASK 4: RECOMMENDATION SYSTEM**

Create a simple recommendation system that suggests items to users based on their preferences. You can use techniques like collaborative filtering or content-based filtering to recommend movies, books, or products to users.

**Source Code:**

**Cell1**

#Import all necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from IPython.html.widgets import \*

sns.set\_style('white')

%matplotlib inline

**Cell2**

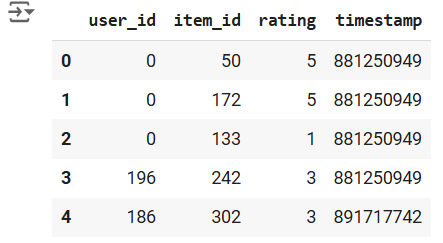
#Get the data into Pandas Dataframe object

import pandas as pd

column\_names = ['user\_id', 'item\_id', 'rating', 'timestamp']

df = pd.read\_csv('dataset.csv', sep = '\t', names = column\_names)

df.head()



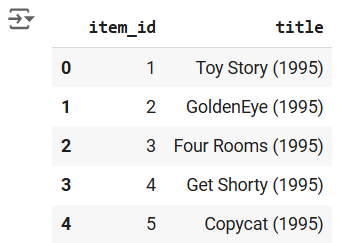
**Cell3**

#Get the Movie Titles

import pandas as pd

movie\_titles = pd.read\_csv('movieIdTitles.csv')

movie\_titles.head()

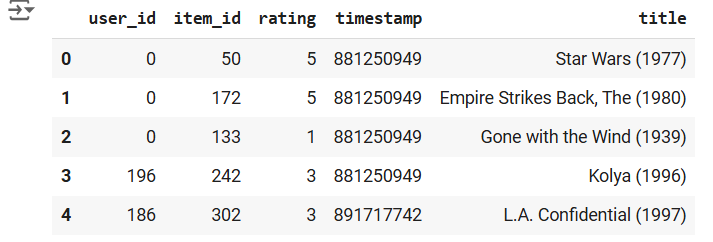


**Cell4**

#Merge the dataset with movie titles

df = pd.merge(df, movie\_titles, on = 'item\_id')

df.head()



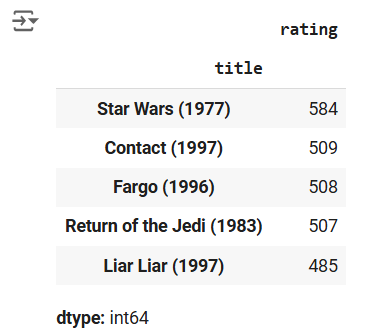
**Cell5**

df.groupby('title')['rating'].mean().sort\_values(ascending = False).head()

****

**Cell6**

df.groupby('title')['rating'].count().sort\_values(ascending = False).head()



**Cell7**

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())

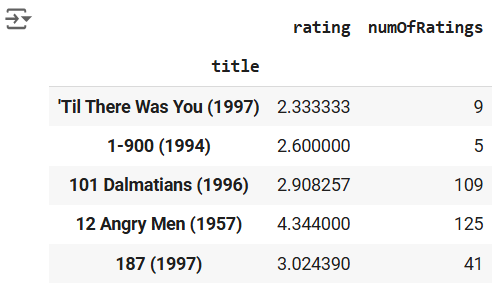
ratings.head()



**Cell8**

ratings['numOfRatings'] = pd.DataFrame(df.groupby('title')['rating'].count())

ratings.head()

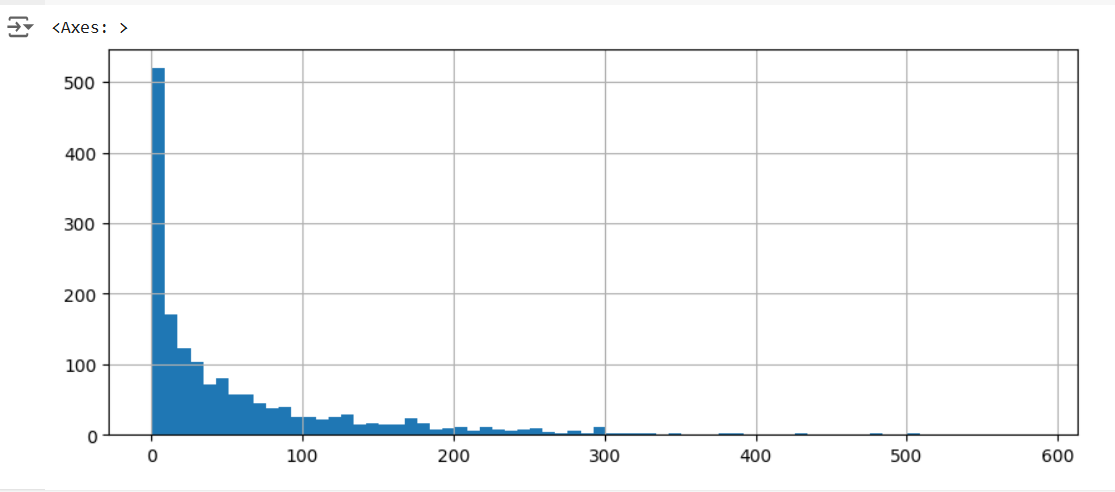


**Cell9**

import matplotlib.pyplot as plt

plt.figure(figsize = (10,4))

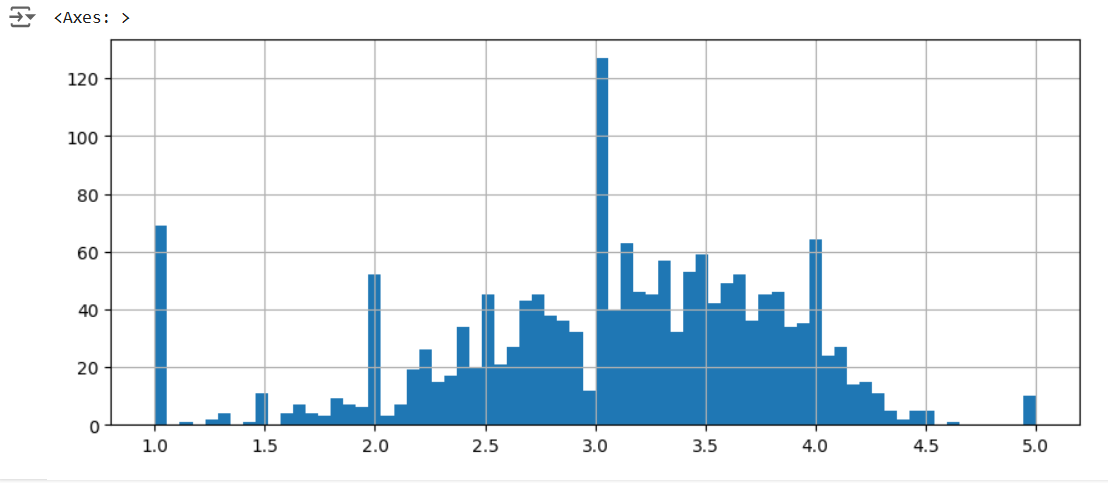
ratings['numOfRatings'].hist(bins = 70)



**Cell10**

plt.figure(figsize = (10,4))

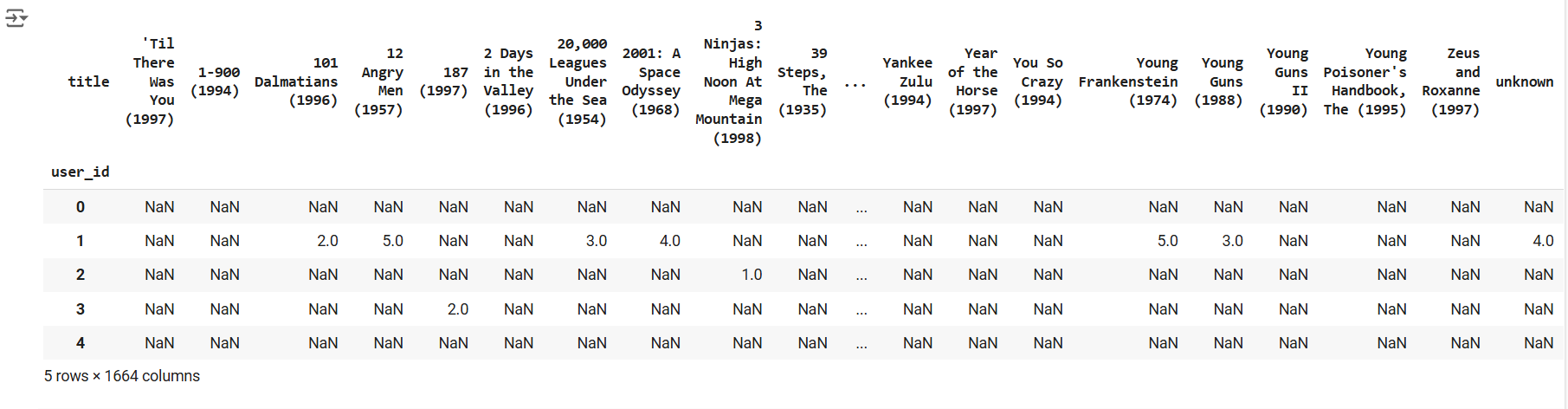
ratings['rating'].hist(bins = 70)



**Cell11**

moviemat = df.pivot\_table(index='user\_id',columns='title',values='rating')

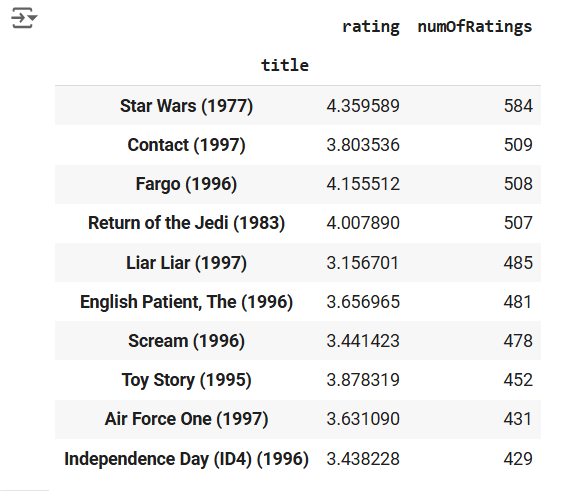
moviemat.head()



**Cell12**

#Most Rated Movies with their Average Ratings

ratings.sort\_values('numOfRatings', ascending = False).head(10)



**Cell13**

for i in ratings.index:

    movieUserRatings = moviemat[i]

    similarToThatMovie = moviemat.corrwith(movieUserRatings)

    corr\_toMovie = pd.DataFrame(similarToThatMovie, columns = ['Correlation'])

    corr\_toMovie.dropna(inplace = True)

    corr\_toMovie = corr\_toMovie.join(ratings['numOfRatings'])

    result = corr\_toMovie[corr\_toMovie['numOfRatings'] > 100].sort\_values('Correlation', ascending = False).head()

    if result['numOfRatings'].count() >= 5:

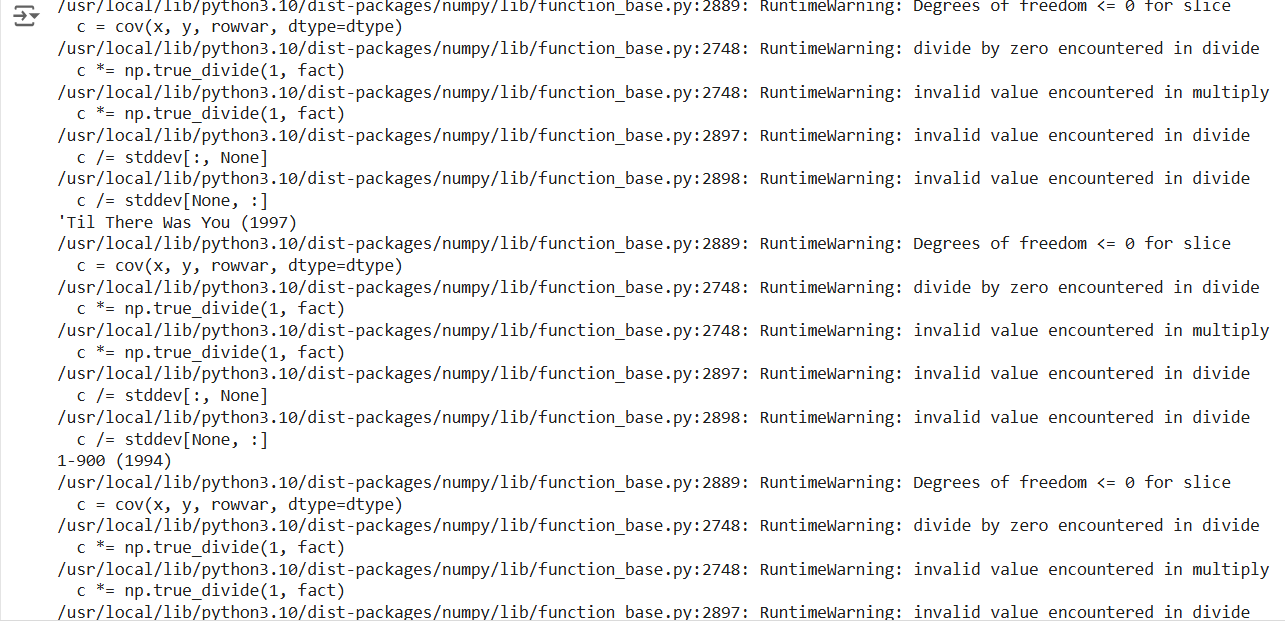
        print(i)

        ratings.loc[i, 'FirstMovieRecommendation'] = result.iloc[1:2].index.values[0]

        ratings.loc[i, 'SecondMovieRecommendation'] = result.iloc[2:3].index.values[0]

        ratings.loc[i, 'ThirdMovieRecommendation'] = result.iloc[3:4].index.values[0]

        ratings.loc[i, 'FourthMovieRecommendation'] = result.iloc[4:5].index.values[0]



**Cell14**

#Check the result

ratings.head()

**Cell15**

ratings = ratings.fillna('-')

**Cell16**

#Save the ratings data for later use

ratings.to\_csv('MovieRecommendations.csv', encoding='utf-8')

**Cell17**

#Load the dataset saved for reusability from this code block onwards

df\_result = pd.read\_csv('MovieRecommendations.csv')

df\_result.head()

**Cell18**

import ipywidgets as widgets

from IPython.display import display

inputMovieName = widgets.Text()

def getRecommendations(sender):

    searchMovie = inputMovieName.value

    list\_result = df\_result[df\_result['title'] == searchMovie]

    fm = list\_result['FirstMovieRecommendation'].values[0]

    sm = list\_result['SecondMovieRecommendation'].values[0]

    tm = list\_result['ThirdMovieRecommendation'].values[0]

    fourthm = list\_result['FourthMovieRecommendation'].values[0]

    finalRecommendationText = '1:' + fm + ' \n2:' + sm + ' \n3:' + tm + ' \n4:' + fourthm

    print('Your Recommendations for the Movie ' + searchMovie + ' are:\n')

    print(finalRecommendationText)