PREDICTION FUNCTION

FUNCTION:

def predict(ratings,similarity,type='user'):

if(type=='user'):

mean\_user\_rating=ratings.mean(axis=1)

#We use np.newaxis so that mean\_user\_rating has same format as ratings

ratings\_diff=(ratings-mean\_user\_rating[:,np.newaxis])

pred=mean\_user\_rating[:,np.newaxis] + similarity.dot(ratings\_diff) / np.array([np.abs(similarity).sum(axis=1)]).T

elif(type=='item'):

pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])

return pred

User-based Collaborative Filtering (type='user'):

Step 1: Compute the Mean Rating for Each User:

mean\_user\_rating = ratings.mean(axis=1)

Step 2:Reshapes the 1D array into a 2D column vector By adding [:, np.newaxis]:

mean\_user\_rating[:, np.newaxis]

Step 3: Center the Ratings by Subtracting the Mean:

ratings\_diff = (ratings - mean\_user\_rating[:, np.newaxis])

Step 4:Apply np.abs(similarity) to take the absolute values of all elements:

np.abs(similarity)

Step 5: Sum Across Each Row (axis=1)

np.abs(similarity).sum(axis=1)

Step 6:Convert to a NumPy Array:

np.array([np.abs(similarity).sum(axis=1)])

Step 7:Compute the normalization term

np.array([np.abs(similarity).sum(axis=1)]).T

Step 8: Multiplies the similarity matrix by the ratings\_diff matrix using matrix multiplication (the .dot() method).

similarity.dot(ratings\_diff)

Step 9:Normalize the weighted sum:

similarity.dot(ratings\_diff) / np.array([np.abs(similarity).sum(axis=1)]).T

Step 10:Add the user's mean rating back to get pred:

pred=mean\_user\_rating[:,np.newaxis] + similarity.dot(ratings\_diff) / np.array([np.abs(similarity).sum(axis=1)]).T

Item-based Collaborative Filtering (type='item')

Step 1:Compute the Weighted Sum of Ratings

multiplies the ratings matrix with the similarity matrix.

ratings.dot(similarity)

Step 2:Normalize the Weighted Sum of Ratings

pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])