APPENDIX

CODE

```
118     import pandas as pd
119     import numpy as np
120     import math
121
122     class CF:
123
124         def ExplodeRatingSet(data):
125             return data.split(',')
126
127         def ExplodeGenre(data):
128             return data.split('|')
129
130         def CountAverageRating(data):
131             DataRate = []
132
133             for i in range(len(data)):
134                 DataTemp                                    =[CF.ExplodeRatingSet(data[i][0])[1],
str(CF.ExplodeRatingSet(data[i][0])[2]),                        CF.ExplodeRatingSet(data[i][0])[3],
CF.ExplodeRatingSet(data[i][0])[4]]
135
136                 count = 0
137
138                 for a in range(len(DataRate)):
139                     if DataRate[a][0] == DataTemp[0] and DataRate != []:
140
141                         Ratings = str(DataRate[a][1]) + ',' + str(DataTemp[1])
142                         DataTemp = [DataTemp[0], Ratings, DataTemp[2], DataTemp[3]]
143                         DataRate[a] = DataTemp
144                         count = 1
145
146                 if count == 0:
147                     DataRate.append(DataTemp)
148                     #print('Data Appendedd')
149
150             return DataRate
151
152         def CosineSimilarity(id, data):
153             Ranking = []
154
155             #Get Movie Average
156             for a in range(len(data)):
157                 if data[a][0] == id:
158                     Movie = data[a]
159                     break
160
161             MovieAverage = CF.ExplodeRatingSet(Movie[1])
162
163             #Scoring Ranking With Cosine Similarity
164             for a in data:
```

```
            OtherMovie = CF.ExplodeRatingSet(a[1])
            B = 1
            Pb = 0
            C = 1
            Pc = 0

            #Hitung B
            for b in MovieAverage:
                B = float(B) * float(b)
                Pb = float(Pb) + float(b)**2

            for c in OtherMovie:
                C = float(C) * float(c)
                Pc = float(Pc) + float(c)**2

            Pb = round(math.sqrt(Pb), 3)
            Pc = round(math.sqrt(Pc), 3)

            BxC = round(B + C, 3)
            BC = round(Pb * Pc, 3)

            Score = round(BxC / BC, 3)

            Score = round(Score / 1000, 2)

            Ranking.append([a[0], a[1], a[2], Score, a[3]])


        return Ranking

    def sortingRanking(val):
        return val[3]

    def printScore(data, limit):
        print('\n-----Movie Ranking-----')
        for i in range(limit):
            print('\nTitle : ', data[i][2], ' Genres : ', data[i][4])

    def CountValidation(ScoreValue, DataActual):

        MSE = 0
        for count in range(len(ScoreValue)):
            rates = CF.ExplodeRatingSet(ScoreValue[count][1])
            AverageRating = 0
            for value in rates:
                AverageRating += float(value)
            AverageRating = AverageRating/len(rates)
            ActualRating = CF.ExplodeRatingSet(DataActual[count][0])
            MSE += pow((AverageRating-float(ActualRating[1])) , 2)

        MSE = 1/len(ScoreValue) * float(MSE)
        print('\n MSE Score : ', round(MSE, 3))
        print('\n RMSE Score : ', round(math.sqrt(MSE), 3))


class NB:
```

```
221         def ExplodeRatingSet(data):
222             return data.split(',')
223 
224         def SplitingData(data, seperator):
225             return data.split(seperator)
226 
227         def PreprocessingData(data):
228             MovieData = []
229             UserData = []
230 
231             CountCoba = 0
232 
233             for value in data:
234                 value = NB.SplitingData(value[0], ',')
235                 #print('\nValue : ', value)
236 
237                 conditionMovie = 0
238                 for countMovie in range(len(MovieData)):
239 
240                     if MovieData[countMovie][0] == value[1]:
241                         MovieData[countMovie][1].append([value[0], value[2]])
242                         MovieData[countMovie][2] = round((MovieData[countMovie][2] +
float(value[2])) / 2, 3)
243                         conditionMovie = 1
244                         CountCoba += 1
245 
246                 if conditionMovie == 0:
247                     MovieData.append( [ value[1], [[value[0], value[2] ]], float(value[2]), value[3] ])
248 
249                 conditionUser = 0
250                 for countUser in range(len(UserData)):
251 
252                     if UserData[countUser][0] == value[0]:
253                         UserData[countUser][1].append([value[1], value[2]])
254                         conditionUser = 1
255 
256                 if conditionUser == 0:
257                     UserData.append([value[0], [ [value[1], value[2] ] ] ])
258 
259             print('\n Count : ', CountCoba)
260 
261             return MovieData, UserData
262 
263         def NaiveBayes(MovieData, UserData):
264             result = []
265             tempAtas = 1.0
266             tempBawah = 1.0
267             #print('\nNaive Bayes')
268 
269             SumValueRating = 0
270             for movie in MovieData:
271                 #print('\nRating Luar : ', movie[1])
272                 for rating in movie[1]:
273 
274                     SumValueRating += float(rating[1])
275 
```

```
276             for MovieValue in MovieData:
277
278                 MovieId = MovieValue[0]
279
280                 SumMovieRating = 0
281                 for Rating in MovieValue[1]:
282                     SumMovieRating += float(Rating[1])
283
284                 tempAtas = 1.0
285                 tempBawah = 1.0
286
287                 for Rating in MovieValue[1]:
288
289                     UserId = Rating[0]
290
291                     TotalUserRating = 0
292                     for UserValue in UserData:
293
294                         if UserValue[0] == UserId:
295
296                             for UserRating in UserValue[1]:
297
298                                 TotalUserRating += float(UserRating[1])
299
300                 tempAtas = tempAtas * (SumMovieRating/SumValueRating)
301
302
303                 total = round(tempAtas/tempBawah, 3)
304
305                 result.append([MovieId, total, MovieValue[2], MovieValue[3]])
306
307             return result
308
309         def CountValidation(NaiveBayes, DataActual):
310             MSE = 0
311
312             for count in range(len(NaiveBayes)):
313
314                 MSE += pow( (float(NaiveBayes[count][2]) - float(
NB.ExplodeRatingSet(DataActual[count][0])[1] ) ) , 2)
315
316             MSE = 1/len(NaiveBayes) * MSE
317             RMSE = math.sqrt(MSE)
318
319             print('\nMSE Score : ', MSE)
320             print('\nRMSE Score : ', RMSE)
321
322             return
323
324
325     # Raw Data
326
327     nRows = 10000
328     RatingSet = pd.read_csv("ratings_movies.csv",delimiter=";", nrows=nRows)
329     RatingSet = RatingSet.values
330
```

```
331
332     # Actual Data
333     DataActual = pd.read_csv("ranking.csv",delimiter=";", nrows=nRows)
334     DataActual = np.array(DataActual)
335     print('\nData Actual :', DataActual[1][0])
336
337
338     # Collaborative Filtering
339
340     Average = CF.CountAverageRating(RatingSet)
341     pd.set_option('display.max_columns', None)
342     print('\nAverage : ', pd.DataFrame(Average).head())
343
344     Ranking = CF.CosineSimilarity('16', Average)
345
346     Ranking = np.array(Ranking)
347     print('\n\nRanking :\n',Ranking)
348     Ranking = Ranking[np.argsort(Ranking[:, 3])]
349
350     CF.printScore(Ranking, 10)
351
352     CF.CountValidation(Ranking, DataActual)
353
354
355     # Naive Bayes
356     MovieData, UserData = NB.PreprocessingData(RatingSet)
357     pd.set_option('display.max_columns', None)
358     print('\nMovie Data : \n ', pd.DataFrame(MovieData).head(10))
359     print('\nUser Data : \n ', pd.DataFrame(UserData).head(10))
360     print('\n Movie Data : ', MovieData[1])
361
362     NaiveBayes = NB.NaiveBayes(MovieData, UserData)
363     NaiveBayes = np.array(NaiveBayes)
364     NaiveBayes = NaiveBayes[np.argsort(NaiveBayes[:, 1])]
365     print('\nOutPut Naive Bayes :\n', pd.DataFrame(NaiveBayes).tail(100))
366
367     NB.CountValidation(NaiveBayes, DataActual)
368
369
370     # Comparing Ranking Movies
371
372     for count in range(10):
373
374         print('\nCF : ', Ranking[count][2], ' NB : ', NaiveBayes[count][3], ' Actual : ',
NB.ExplodeRatingSet(DataActual[count][0])[2] )
375
```

Report #14327927

1. CHAPTER 1 INTRODUCTION Background Movies have become our daily needs in terms of entertainment. It's no secret that several well-known platforms such as Netflix, Youtube, Disney+ Hotstar, and others have more or fewer millions and even tens of millions of films on each of these platforms. Therefore, the recommendation engine is at the heart of various movie provider platforms. The number of these films is what causes a new problem, namely what film to watch next. Almost all platforms must collect data on movies watched by their users, the problem is if a platform still doesn't have data from that user or a new user. Then what are the movie recommendations that should be presented to the new user if a platform still doesn't have viewing data from that user or still has very little data from that user. Of course, users from each platform want to get movie recommendations that they think are most suitable for them, if not, then it's likely that these users will not use the platform anymore. However, this