**Assignment 2**

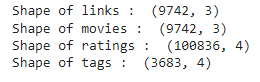
**Amulya Agrawal MT20004**

**Aniket Chauhan MT20030**

**Ques 1**.

Perform exploratory data analysis (EDA) over the shared dataset. Report at least three insights about the dataset.

1. **Finding the Number of Columns and Rows in each file**

****

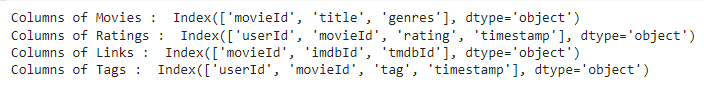
Movies: 3 columns, 9742 rows

Links: 3 columns, 9742 rows

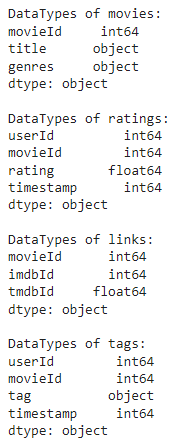
Tags: 4 columns, 3683 rows

Ratings: 4 columns, 100836 rows

1. **Finding the columns**



1. **Finding the datatypes of the columns**

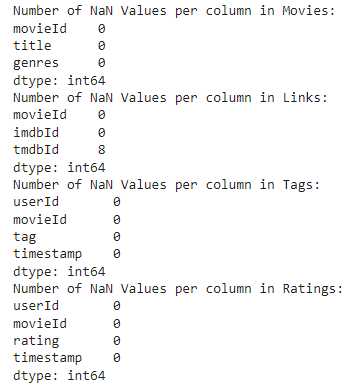
****

1. **Finding frequently occurring values in categorical features**

| **S.No.** | **File** | **Column Name** | **Frequently Occurring Values** | **Count** |
| --- | --- | --- | --- | --- |
| 1 | Movies | Title | Confessions of a Dangerous Mind (2002) Emma (1996)  Eros (2004)  Saturn 3 (1980)  War of the Worlds (2005) | 2 |
| 2 | Movies | Genre | Drama | 1053 |
| 3 | Ratings | Rating | 4 | 26818 |
| 4 | Tags | Tag | In Netflix queue | 131 |

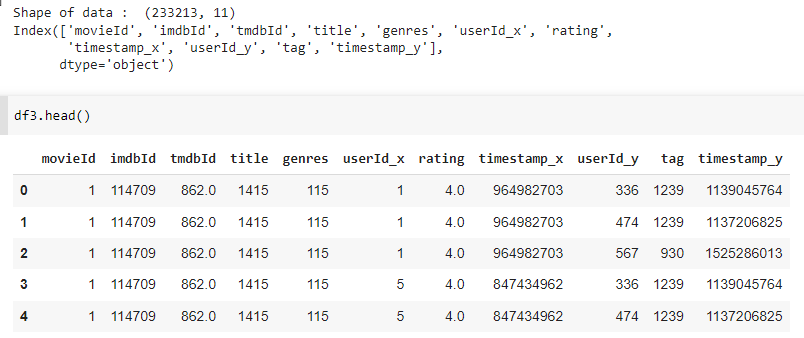
1. **Counting of Nan values per column**

Using **dataframe.isnull().sum()**, we can find the number of NaN values per column

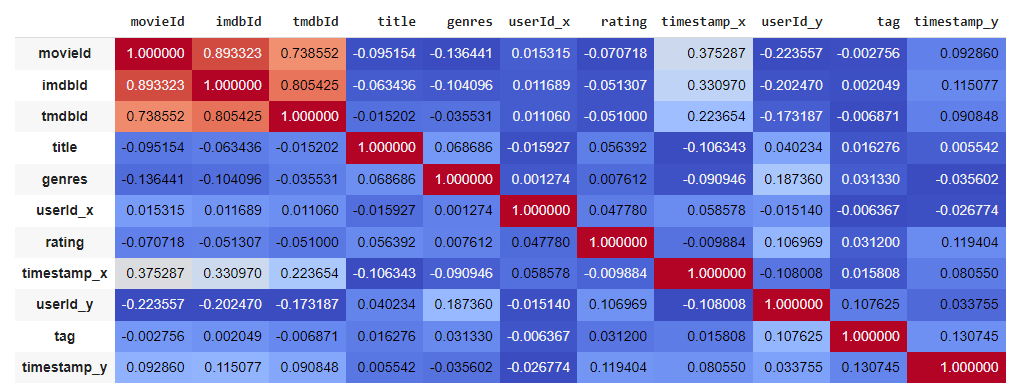


**Inference:** We have 8 NaN values in file “Tags” under the column: “tmdbId”

1. **Merging the files to get a single dataframe**

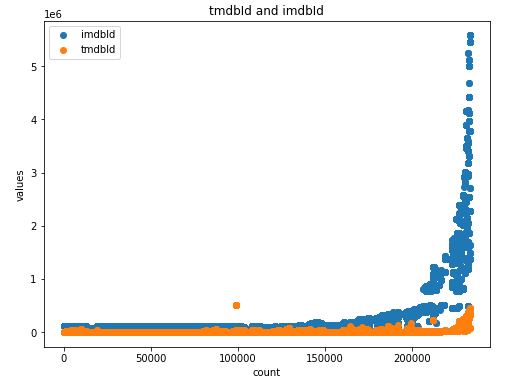
****

1. **HeatMap between the columns of the dataframe**

****

**Inference**: There is a positive correlation between imdbId and tmdbId.

1. **Plotting between features -** imdbId and tmdbId.

****

**Ques 2**

Using the association rule mining techniques, build a recommendation system for the dataset provided to you. Your system should be able to recommend movies given a customer profile. I.e. What set of movies has a customer watched in the past? The set can contain any non-zero number of movies. Let us say a customer has seen "avengers, iron man", the system can recommend the following movies "Captain America, hulk, thor, doctor strange". The final scores will be computed using the metric "Precision@k". For a given customer profile, you will recommend the four best movies, i.e. k=4.

1. Processing\_Movies(movies):

mdf = movies.drop('genres',axis = 1)

mdf = mdf.join(movies.genres.str.get\_dummies())

mdf = mdf.drop([ 'movieId' ,'title'],axis = 1)

mdf.head()

mdf\_n = []

mdf\_a = np.array(mdf)

mdf\_c = list(mdf.columns)

mdf\_len = len(mdf\_c)

for i in range(len(mdf)):

t = []

for j in range(mdf\_len):

if mdf\_a[i,j]==1:

t.append(mdf\_c[j])

mdf\_n.append(t)

return mdf\_n

1. Rule\_Mining(mdf\_n,min\_length=2,min\_support=0.0005, min\_confidence=0.3, min\_lift=3):

arules = apriori(mdf\_n, min\_length=min\_length,min\_support=min\_support, min\_confidence=min\_confidence, min\_lift=min\_lift)

afinal = list(arules)

rules = []

for i in range(len(afinal)):

x = list(list(list(afinal[i][2])[0])[0])

y = list(list(list(afinal[i][2])[0])[1])

rules.append((x,y))

return rules,afinal

1. getactualids(ratings):

ratings = ratings.sort\_values(by=['userId','movieId'])

actual\_ids = {}

user\_ids = list(set(ratings['userId']))

for i in range(len(user\_ids)):

actual\_ids[user\_ids[i]] = list(ratings[ratings['userId']==user\_ids[i]]['movieId'])

return actual\_ids,user\_ids

1. getmoviegenre(movies):

mdf = movies.drop('genres',axis = 1)

mdf = mdf.join(movies.genres.str.get\_dummies())

col = list(mdf.columns)

col = col[2:]

Genre\_Movie\_ID = {}

for i in range(len(col)):

colv = list(mdf[mdf[col[i]]==1]['movieId'])

Genre\_Movie\_ID[col[i]] = colv

return Genre\_Movie\_ID

1. getusergenre(df3,user\_ids):

User\_Genre = df3[['genres','userId\_x']]

User\_Genre\_dict = {}

for i in range(len(user\_ids)):

User\_Genre\_dict[user\_ids[i]] = list(User\_Genre[User\_Genre['userId\_x']==user\_ids[i]]['genres'])

for i in list(User\_Genre\_dict.keys()):

v = User\_Genre\_dict[i]

temp = []

for j in v:

temp = temp + j.split('|')

temp = list(set(temp))

User\_Genre\_dict[i] = temp

return User\_Genre\_dict

1. getpredictedids(user\_ids,User\_Genre\_dict,rules):

predicted\_ids = {}

for i in range(len(user\_ids)):

watched\_genre = User\_Genre\_dict[user\_ids[i]]

pred\_movie = []

for j in range(len(rules)):

rule = rules[j]

l1 = rule[0]

l2 = rule[1][0]

contained = all(k in watched\_genre for k in l1)

if contained:

predgenre = l2

pred\_movie = pred\_movie + Genre\_Movie\_ID[predgenre]

predicted\_ids[user\_ids[i]] = pred\_movie

return predicted\_ids

1. Pecision\_k(actual\_ids,predicted\_ids,k,):

pre = 0

for i in actual\_ids:

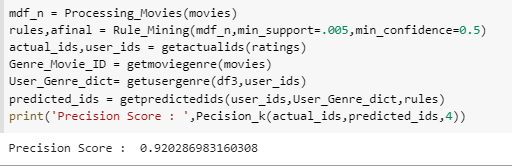
ints = len(set.intersection(set(actual\_ids[i]),set(predicted\_ids[i])))

div = (len(set(actual\_ids[i]))\*((k/10)\*const))

pre = pre + ints/div

pre = pre/len(list(actual\_ids.keys()))

return pre



**Ques 3**

Visualize the maximal frequent pattern set.

def getfrequentitems(afinal,rules):

data = []

for i in range(len(afinal)):

x = rules[i][0]

y = rules[i][1]

s = afinal[i][1]

data.append([x+y,s,len(x+y)])

freq = pd.DataFrame(data,columns = ['Itemsets','Support','Length'])

return freq

