DATA MINING
ASSIGNMENT-2

AASTHA 2019224
SATWIK TIWARI 2019100

1

Insights about dataset

1) Counting NaN values per column

```
Movies
movieId    0
title      0
genres     0
dtype: int64

Links
movieId    0
imdbId     0
tmdbId     8
dtype: int64

Ratings
userId       0
movieId      0
rating       0
timestamp    0
dtype: int64

Tags
userId       0
movieId      0
tag          0
timestamp    0
```

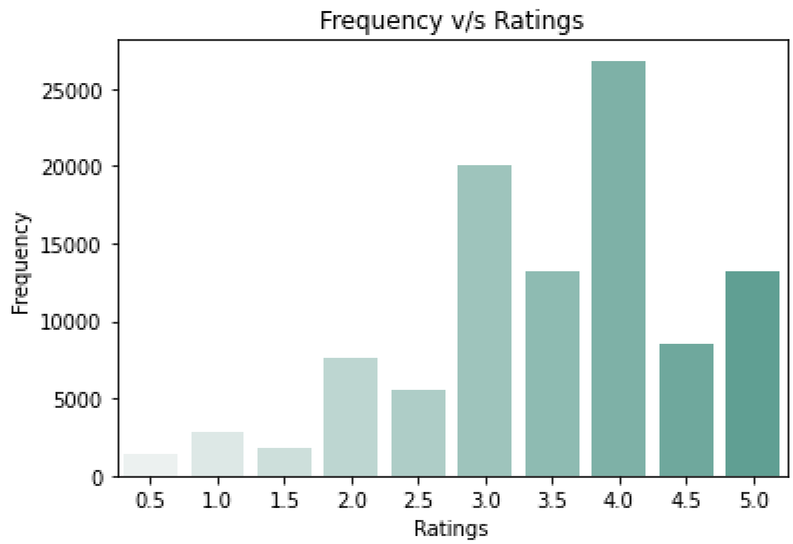
2) Top 4 Movies Based On Average Rating

	title	rating
48	Lamerica (1994)	5.0
87	Heidi Fleiss: Hollywood Madam (1995)	5.0
121	Awfully Big Adventure, An (1995)	5.0
405	Live Nude Girls (1995)	5.0

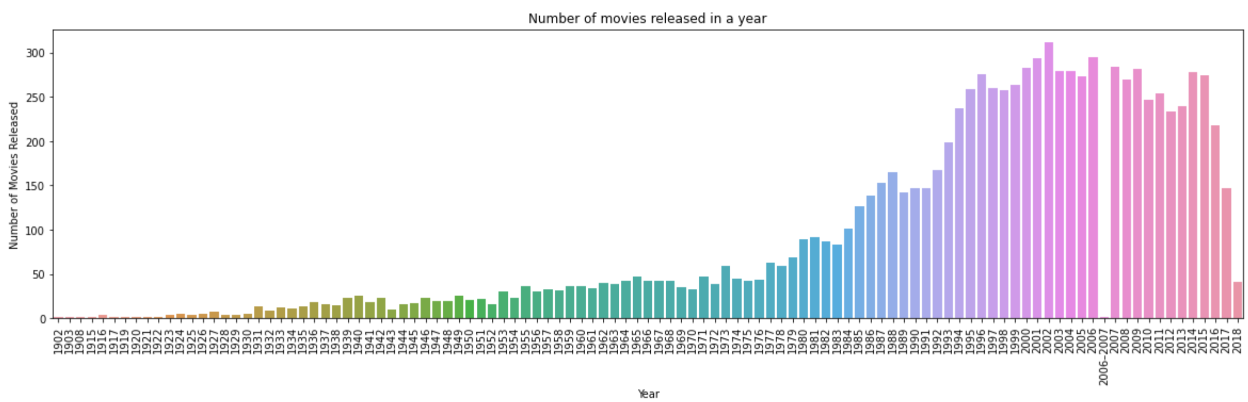
3) Bottom 4 Movies Based On Average Rating

	title	rating
2685	Gypsy (1962)	0.5
2929	Killer Shrews, The (1959)	0.5
3023	Horrors of Spider Island (Ein Toter Hing im Ne...	0.5
3230	Baby Boy (2001)	0.5

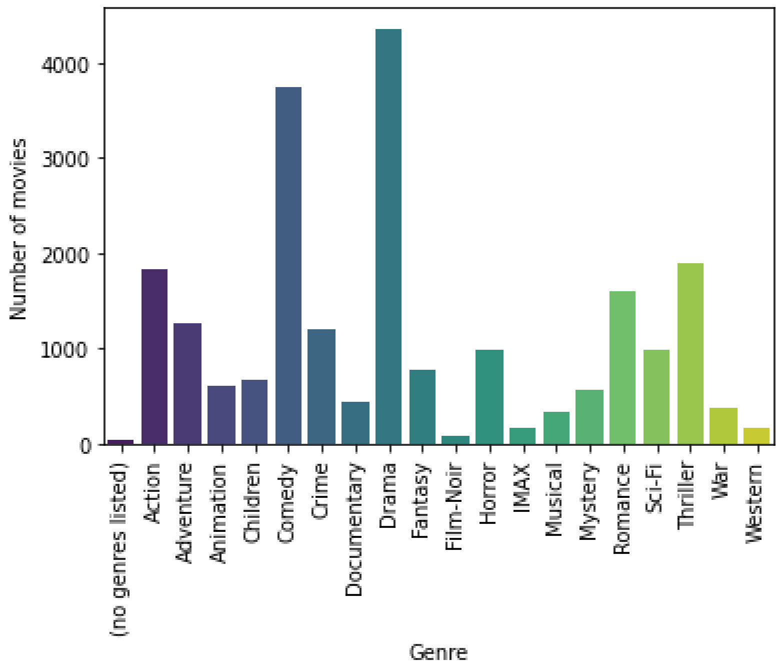
4) Frequency of Ratings - 4 is the most frequent rating given by users followed by 3 and 5



5) Number of movies released in a year-Highest number of movies released in 2002



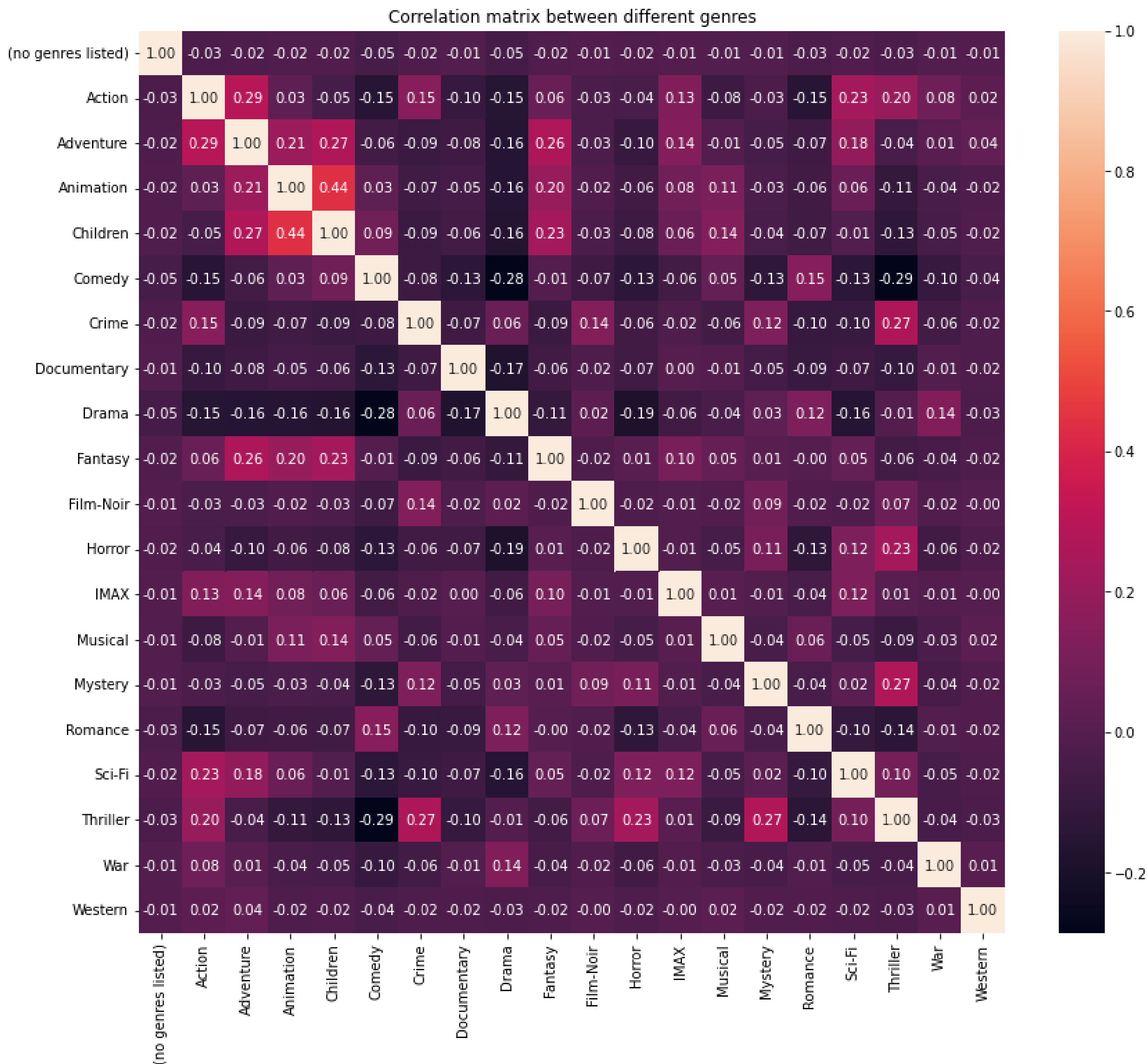
6) Number of movies for each genre - Drama had the highest number of movie releases



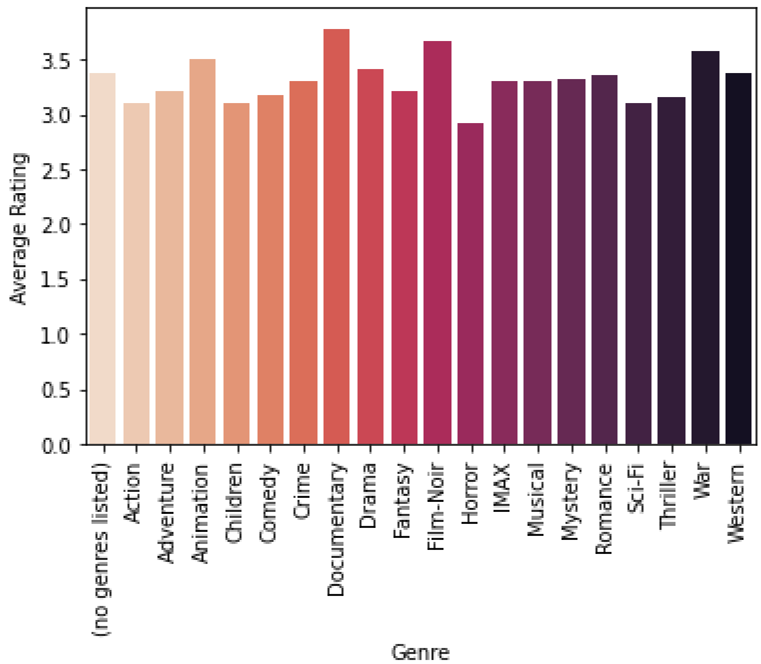
7) 3 genres with most movie releases

```
Drama        4361
Comedy       3756
Thriller     1894
```

8) Correlation Matrix Between Different Genres-High correlation can be observed between children and animation



9) Average rating for each genre



10) 3 Genres With Highest Average Rating

	0
Documentary	3.781682
Film-Noir	3.670471
War	3.571655

11) 3 Genres With Lowest Average Rating

	0
Horror	2.918965
Action	3.094498
Sci-Fi	3.102637

2

Model Training

First, we have grouped the movies by userId. So now we have a set of movies that each user has watched. Based on that we'll have a set of genres and tags, we have trained our model on such sets using apriori rule of association.

Making Rules For Genre and Tags

```
[ ] te = TransactionEncoder()
    te_ary = te.fit(df_genre_list).transform(df_genre_list)
    df = pd.DataFrame(te_ary, columns=te.columns_)
```

```
[ ] apriori_frequent_itemsets = apriori(df, min_support=0.2,use_colnames=True,max_len=10)
```

```
[ ] rules_genre = association_rules(apriori_frequent_itemsets,metric="lift",min_threshold=1)
    rules_genre = rules_genre.sort_values(by=['lift'], ascending=False)
```

```
[ ] rules_genre
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2829	(Mystery, Adventure)	(Action, Fantasy, Thriller)	0.241379	0.206897	0.206897	0.857143	4.142857	0.156956	5.551724
4940	(Adventure, Mystery, Drama)	(Action, Romance, Thriller)	0.241379	0.224138	0.224138	0.928571	4.142857	0.170036	10.862069
4874	(Action, Fantasy, Thriller)	(Adventure, Mystery, Drama)	0.206897	0.241379	0.206897	1.000000	4.142857	0.156956	inf
2834	(Fantasy, Thriller)	(Action, Mystery, Adventure)	0.206897	0.241379	0.206897	1.000000	4.142857	0.156956	inf
4954	(Mystery, Adventure)	(Action, Romance, Drama, Thriller)	0.241379	0.224138	0.224138	0.928571	4.142857	0.170036	10.862069
...	...	...	...	...	...	...	...	...	...
56	(Comedy)	(Sci-Fi)	0.603448	0.465517	0.293103	0.485714	1.043386	0.012188	1.039272
883	(Drama)	(Action, Adventure, Sci-Fi)	0.672414	0.379310	0.258621	0.384615	1.013986	0.003567	1.008621
874	(Action, Adventure, Sci-Fi)	(Drama)	0.379310	0.672414	0.258621	0.681818	1.013986	0.003567	1.029557
124	(Drama)	(Action, Adventure)	0.672414	0.431034	0.293103	0.435897	1.011282	0.003270	1.008621
121	(Action, Adventure)	(Drama)	0.431034	0.672414	0.293103	0.680000	1.011282	0.003270	1.023707

5904 rows × 9 columns

```
[14] te = TransactionEncoder()
     te_ary = te.fit(df_tags_list).transform(df_tags_list)
     df = pd.DataFrame(te_ary, columns=te.columns_)
```

```
apriori_frequent_itemsets = apriori(df, min_support=0.62, use_colnames=True, max_len = 8)
apriori_frequent_itemsets.head()
print(apriori_frequent_itemsets.size)

6650
```

```
[16] rules_tags = association_rules(apriori_frequent_itemsets,metric="lift",min_threshold=0.8)
     rules_tags = rules_tags.sort_values(by=['lift'], ascending=False)
```

```
[17] rules_tags
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
97900	(psychological, atmospheric, classic, stylized)	(quirky, thought-provoking, imdb top 250, action)	0.625000	0.631944	0.621528	0.994444	1.573626	0.226562	66.250000
97837	(quirky, thought-provoking, imdb top 250, action)	(psychological, atmospheric, classic, stylized)	0.631944	0.625000	0.621528	0.983516	1.573626	0.226562	22.750000
99558	(quirky, twist ending, thought-provoking, imdb...	(psychological, atmospheric, stylized)	0.625000	0.631944	0.621528	0.994444	1.573626	0.226562	66.250000
99735	(psychological, atmospheric, stylized)	(quirky, twist ending, thought-provoking, imdb...	0.631944	0.625000	0.621528	0.983516	1.573626	0.226562	22.750000
82949	(atmospheric, dark, stylized)	(quirky, psychological, imdb top 250, action)	0.628472	0.628472	0.621528	0.988950	1.573578	0.226550	33.623264
...	...	...	...	...	...	...	...	...	...
396	(suspense)	(time travel)	0.798611	0.732639	0.638889	0.800000	1.091943	0.053795	1.336806
401	(time travel)	(thought-provoking)	0.732639	0.795139	0.635417	0.867299	1.090751	0.052867	1.543775
400	(thought-provoking)	(time travel)	0.795139	0.732639	0.635417	0.799127	1.090751	0.052867	1.330993
379	(twist ending)	(sci-fi)	0.791667	0.750000	0.638889	0.807018	1.076023	0.045139	1.295455
378	(sci-fi)	(twist ending)	0.750000	0.791667	0.638889	0.851852	1.076023	0.045139	1.406250

101298 rows × 9 columns

Prediction

Given some movies. We have extracted both its genre and tags and based on that we'll have a set of genres and tags from the given movies.

Now we have used association rules to find out if the user had watched such genres and tags then what else is also recommended using the model we had trained above.

As the dataset is very large, The maximum number of items in a set is limited so in case we have a large set of genres or tags, We are iterating through the association rules and picking up the best rule which has the maximum intersection with our set and predictions are made accordingly.

```
input_tags = set(curr_tags)
maximal_intersection = 0
best_pred_tags = set()

# iterating through rules

for i in rules_tags.values:
    temp_intersection = 0
    for j in i[0]:
        if j in input_tags:
          temp_intersection += 1

    if(temp_intersection > maximal_intersection):
      maximal_intersection = temp_intersection
      best_pred_tags = set()
      for j in i[1]:
          best_pred_tags.add(j)


for i in best_pred_tags:
    input_tags.add(i)
```

Based on the final set of genres and tags, we had implemented a scoring system (to be discussed in the scoring section) and scored each of the movies.

Now we can simply recommend the top 4 movies based on scores.

Scoring System -

For scoring, we have assigned some priority to tags, genre and ratings and based on that each movie has been assigned some score.

We have also taken one additional parameter which is the similarity between the movie titles. Though the priority of this is pretty low as compared to the genre, tag and rating.

For the sample, These were our predictions.

output.csv

1 to 9 of 9 entries Filter

movies	recommendation
the godfather (1969)	out of sight (1998) negotiator, the (1998) beat the devil (1953) charade (1963)
the dark knight (2008) the dark knight rises (2012)	batman forever (1995) batman (1989) inception (2010) negotiator, the (1998)
jfk (1991) the file on thelma jordon (1950) a love song for bobby long (2004)	inception (2010) 21 grams (2003) out of sight (1998) negotiator, the (1998)
bobby (1973)	out of sight (1998) charade (1963) femme nikita, la (nikita) (1990) inception (2010)
little miss broadway (1938)	21 grams (2003) charade (1963) how to steal a million (1966) out of sight (1998)
frankenstein meets the wolf man (1943)	out of sight (1998) charade (1963) femme nikita, la (nikita) (1990) inception (2010)
american movie (1999) collapse (2006) revenge of the green dragons (2014) pain & gain (2013)	out of sight (1998) how to steal a million (1966) talk of the town, the (1942) charade (1963)
the mambo kings (1992)	out of sight (1998) charade (1963) femme nikita, la (nikita) (1990) inception (2010)
the joneses (2009)	out of sight (1998) negotiator, the (1998) beat the devil (1953) charade (1963)

Show 10 per page

3

Generating Frequent and Maximal Frequent Itemsets

For generating frequent itemsets, we have used fpgrowth from mlxtend library.

```
[ ] te = TransactionEncoder()
    te_ary = te.fit(df_genre_list).transform(df_genre_list)
    df = pd.DataFrame(te_ary, columns=te.columns_)
```

```
[ ] frequent_genre = fpgrowth(df, min_support=0.04, use_colnames=True,max_len = 6)
```

Now for Maximal frequent itemsets, we're iterating among all the generated frequent itemsets and picking out the itemsets for which none of its immediate supersets are frequent.

```
def maximal_itemset(frequent):
  su = frequent.support.unique()#all unique support count

  #Dictionay storing itemset with same support count key
  fredic = {}
  for i in range(len(su)):
      inset = list(frequent.loc[frequent.support ==su[i]]['itemsets'])
      fredic[su[i]] = inset

  #Dictionay storing itemset with  support count <= key
  fredic2 = {}
  for i in range(len(su)):
      inset2 = list(frequent.loc[frequent.support<=su[i]]['itemsets'])
      fredic2[su[i]] = inset2

  ml = []
  for index, row in frequent.iterrows():
    isclose = True
    cli = row['itemsets']
    cls = row['support']
    checkset = fredic2[cls]
    for i in checkset:
        if (cli!=i):
            if(frozenset.issubset(cli,i)):
                isclose = False
                break

    if(isclose):
        ml.append(row['itemsets'])
  return ml
```

Generating Tree

As the data is very large, we have taken a subset of items and visualized the tree based on these itemsets.

We have used the networkx library for constructing the tree. Given the items, I've first created all the subsets of the items and then appropriate edges are added among the nodes.

```
#creating edge list
G = nx.Graph()
edges = deepcopy(id)

for i in range(2, len(id) + 1):
  temp = list(combinations(id,i))
  for j in temp: edges.append(j)

label = {}
for i in range(len(edges)):
  if(type(edges[i]) == type(2)): label[i] = items[edges[i]]
  else:
    curr = []
    for j in edges[i]:
      curr.append(items[j])
    label[i] = curr

edge_list = []
for i in range(len(edges)):
  for j in range(i + 1, len(edges)):
    if(type(edges[i]) == type(2)): a = set([edges[i]])
    else: a = set(edges[i])
    if(type(edges[j]) == type(2)): b = set([edges[j]])
    else: b = set(edges[j])
    if(a.intersection(b) != set()): edge_list.append((i,j))

G.add_edges_from(edge_list)
```

Now to highlight all the maximal frequent itemsets, We are iterating among the nodes of the tree and checking if this itemset is present in the maximal frequent itemsets generated above and changing its color accordingly (Please refer to the screenshots of the tree below)

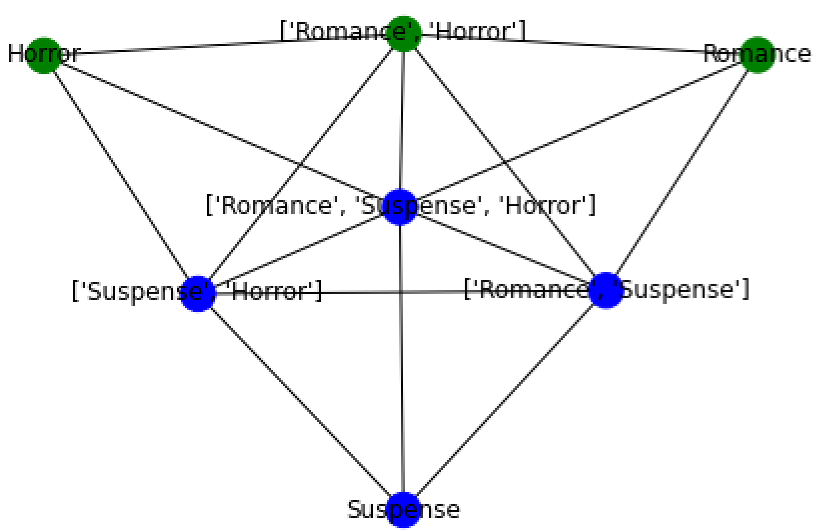
```
#selecting colors
color = []
for node in G:
  curr = edges[node]
  tags = set()
  if(type(curr) == type(2)): tags.add(items[curr])
  else:
    for j in curr:
      tags.add(items[j])

  if(check_in_MFS(tags, item_set)):
    color.append('green')
  else: color.append('blue')
```

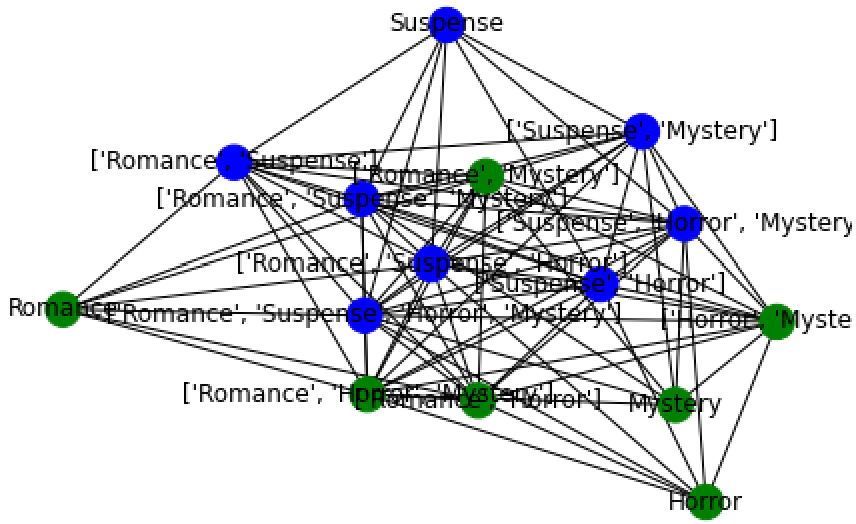
Visualization

The items in green represent that the item is in Maximal Frequent itemsets.

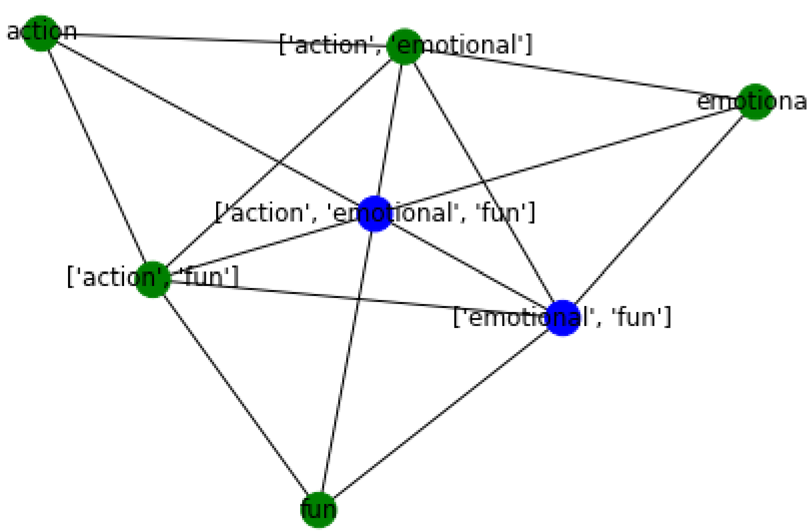
Maximal Frequent Itemsets for 3 items based on Genres



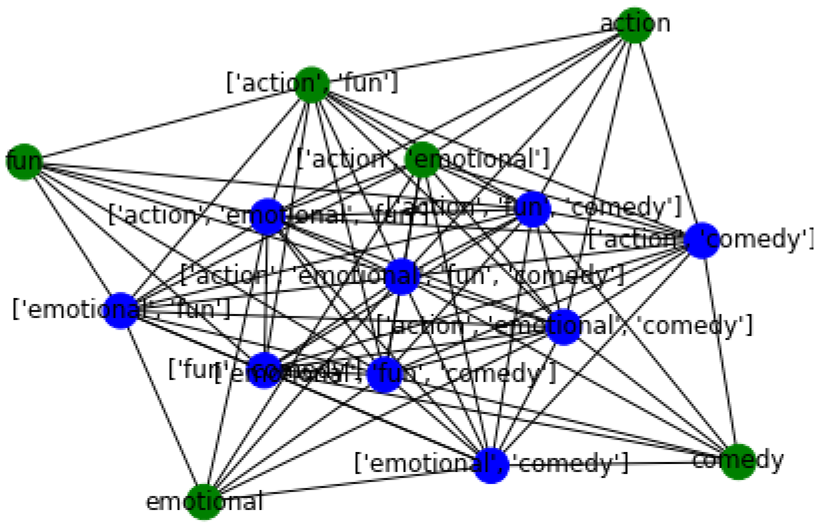
Maximal Frequent Itemsets For 4 items based on Genres



Maximal Frequent Itemsets for 3 Items Based on Tags



Maximal Frequent Itemsets for 4 items based on Tags



Learnings

1. We learned about performing estimated data analysis on a given dataset. It provided a better understanding of the dataset, its features and their values and helped in gaining insights into the dataset. Also learned about using various python libraries like pandas, seaborn and matplotlib.
2. Learned about association rules in data mining like apriori and building recommender systems and working with the association rules on a larger dataset and using the rules with the maximal intersection.
3. Learned about generating maximal frequent pattern sets and visualizing them using networkx and matplotlib. Created tree using the frequent itemsets from scratch and highlighted maximal frequent itemsets among them.

References

1. https://rasbt.github.io/mlxtend/
2. https://pandas.pydata.org/docs/
3. https://numpy.org/doc/stable/
4. https://seaborn.pydata.org/
5. https://towardsdatascience.com/how-to-find-closed-and-maximal-frequent-itemsets-from-fp-growth-861a1ef13e21
6. https://networkx.org/documentation/stable/tutorial.html