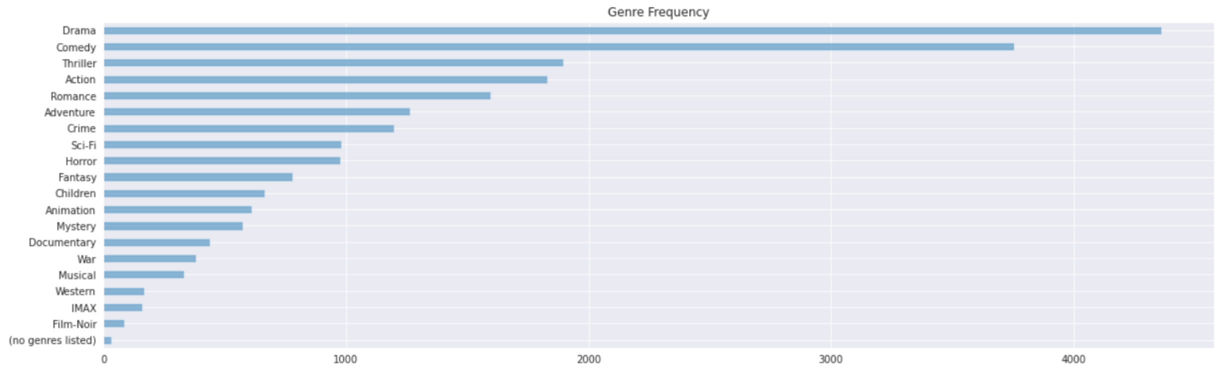
Pre-processing Steps:

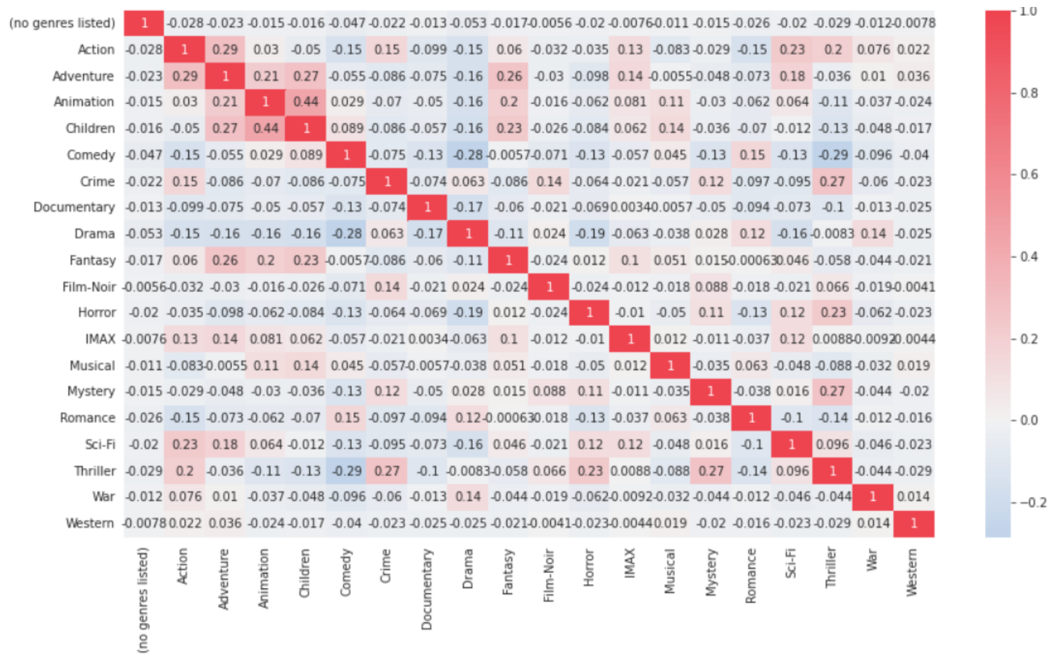
1. Loaded the 4 datasets *links_df*, *movies_df*, *ratings_df*, *tags_df*
2. **NOTE:** We have not used *links_df* as it only contains the *imdbId, tmdbId* which is not useful in recommending movies to a user.
3. *movies_df*:
    a. Extracted '*release year*' from '*title*', and used it to create '*release period*' (eg: 1995-2006) which can be used to recommend movies based on the same release period.
    b. One-hot encoded the '*genre*' feature to increase its usability for further computations.
4. *ratings_df:*
    a. Converted *timestamp* from UNIX to UTC format to extract the rating year which might be helpful in selecting the ratings.
5. *tags_df:*
    a. Dropped *timestamp* as it is an unnecessary feature for the system.
    b. One-hot encoded the '*tag*' feature to increase its usability for further computations.

EDA:

1. *movies_df*:
    a. Counting of Nan values per column: None of the columns contained any nan values, except for 'release period' which contained 12 nan values. We have not dropped the column since the count is insignificant with respect to the total number of samples. Also we have not dropped the rows corresponding to the nan values of release period as it would lead to removal for rows from other tables as well (due to *movieId* column). We therefore simply ignored the nan values while using them as a part of our analysis.
    b. Counting duplicate rows: Since the count came out to be zero, we need not to remove any row.
    c. Genre frequency plot: As shown by the following plot, Drama and Comedy are two most popular genres. Also there are insignificant movies with No listed genres.



d. Correlation matrix among genres: As shown below, none of the features are highly correlated. The most correlated features are ‘Children’ and ‘Animation’ which shows that most of the children's movies are animated or vice versa.

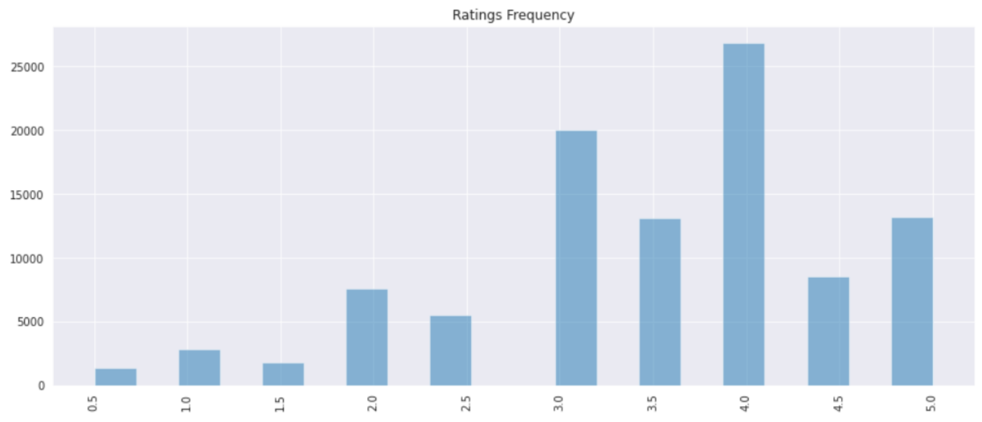


2. *ratings_df*:

   a. Counting of Nan values per column: None of the columns contained any nan values, so we need not to drop any column or row.

   b. Counting duplicate rows: Since the count came out to be zero, we need not to remove any row. Also every user has given only a single rating to a particular movie, therefore we don’t need *rating year.*

   c. Ratings frequency plot: As shown by the following plot (and mean user ratings and mean movie ratings plot), most of the users have given 3.0 to 4.0 ratings to the movies.

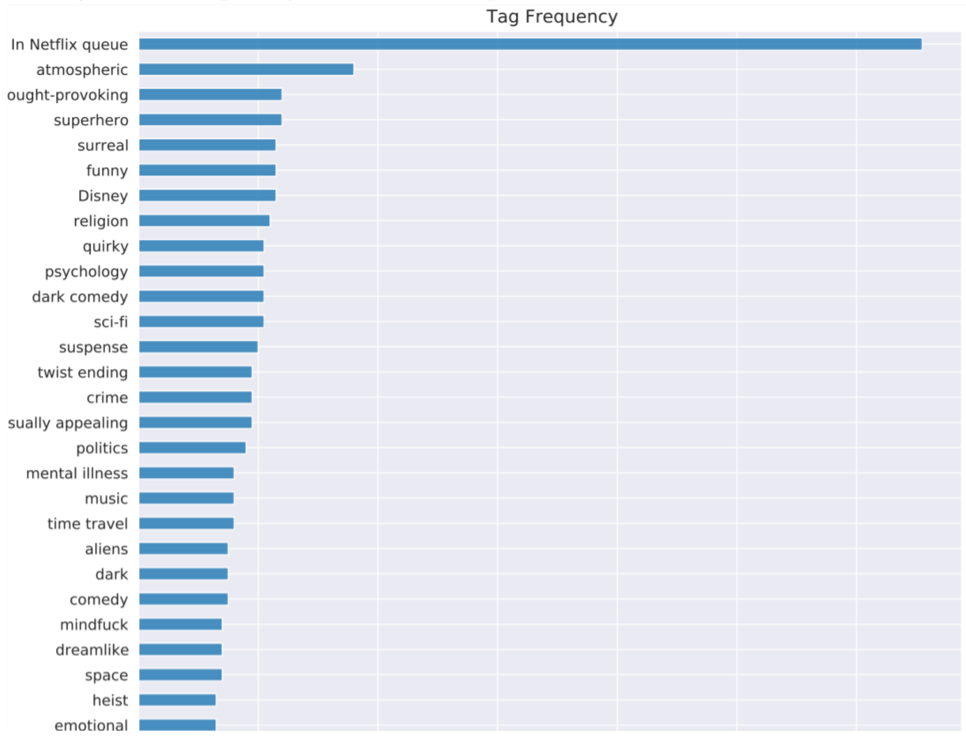


3. *tags_df*:

   a. Counting of Nan values per column: None of the columns contained any nan values, so we need not to drop any column or row.

   b. Counting duplicate rows: Since the count came out to be zero, we need not to remove any row.

   c. Tags frequency plot: As shown by the following plot, ‘In Netflix queue’ is the most popular tag, which shows that many users have simply added the movies in their wishlist. Also since there are more than 1500 tags, we have dropped the tags with frequency less than or equal to 10.



d. Correlation matrix among tags: None of the features comes out to be highly correlated. The most correlated features are ‘Comedy’ and ‘Dark Comedy’ which shows many of the comedy movies involve dark comedy as well.

Transactional Data Preparation (Post-processing):

We need to create a list of list of movies, which we can do by combining the movies in the following 4 ways:

1) Combining the movies that the user has watched and liked (given 3+ rating).
2) Combining the movies based on genre (NOTE: We have not considered the ‘No listed genre’ as it is not a genre).
3) Combining the movies based on user tags (NOTE: We have not considered the ‘In Netflix queue’ as the user has not watched the movie while giving the tag).
4) Combining the movies based on release period (NOTE: We have ignored the Nan values)

We combined all the above lists to get the final transaction list, and one hot encoded it using *TransactionEncoder.*

Association Rules:

For association rules generation we used the mlxtend library. Firstly, we used `fpgrowth` using `min_support` of 0.05 (considering number of frequent itemset and time complexity) for generating frequent itemsets. Here we have taken a frequent itemset of length utmost 2, because we tried to make rules using larger length frequent itemsets which produced a large combination hence it became very cumbersome to predict. Then we used the `association_rules` function of the same library for rules generation using '`lift`' as a matrix with `min_threshold` of 0.01.

Prediction:

- For recommending the movies we take I/P in the form of a csv file which should have a column named '**`movies'.`** If we get a column having multiple movies(should be separated by a new line) then we predict movies using each of the movies and combine and sort them using 'lift' and pick top 4 movies for recommendation.
- And if our rules are not recommending 4 movies for any given I/P then we randomly choose movies from a list which have 50 movies which we have chosen from frequent itemset of length 1.
- After recommending movies we write them in the given csv separated by new lines with column name '**`recommendation_prediction`**'.

Maximal Frequent Itemsets:

- For getting maximal frequent itemsets, we used the fpmax function of mlxtend library (help in extract maximal itemsets for association rule mining).
- Fpmax takes a dataframe(in which you want to find maximal frequent itemset), `min_support = 0.05` (a float between 0 and 1 for minimum support of the itemsets returned), use_colnames (if true, use the DataFrames' column names in the returned DataFrame instead of column indices) as input.
- Since we have a large number of maximal frequent itemset, it is feasible to visualize them all hence we made rules using these itemset setting *min_threshold* = 0.01.Then we used 'networkx' and plotted these rules and saved this visualization in the png file.