Dataset Analysis

For our system implementation we needed a dataset to use for our sequential recommendation algorithm for movies. We made use of the movielens 1 million[1] dataset. This is a stable benchmark dataset and thus often used for recommender system testing. Movielens is a website where users can rate movies they watched to create a taste profile which can be used to provide recommendations. It's a set of 1 million ratings made by 6000 users on 4000 movies.

We plotted the distribution of the amount of ratings done by each user. We can see from this in the Figure 1 below that the rating count follows a Pareto like distribution, where a small number of users do perform a large amount of ratings while the majority of users only stick to a few ratings.

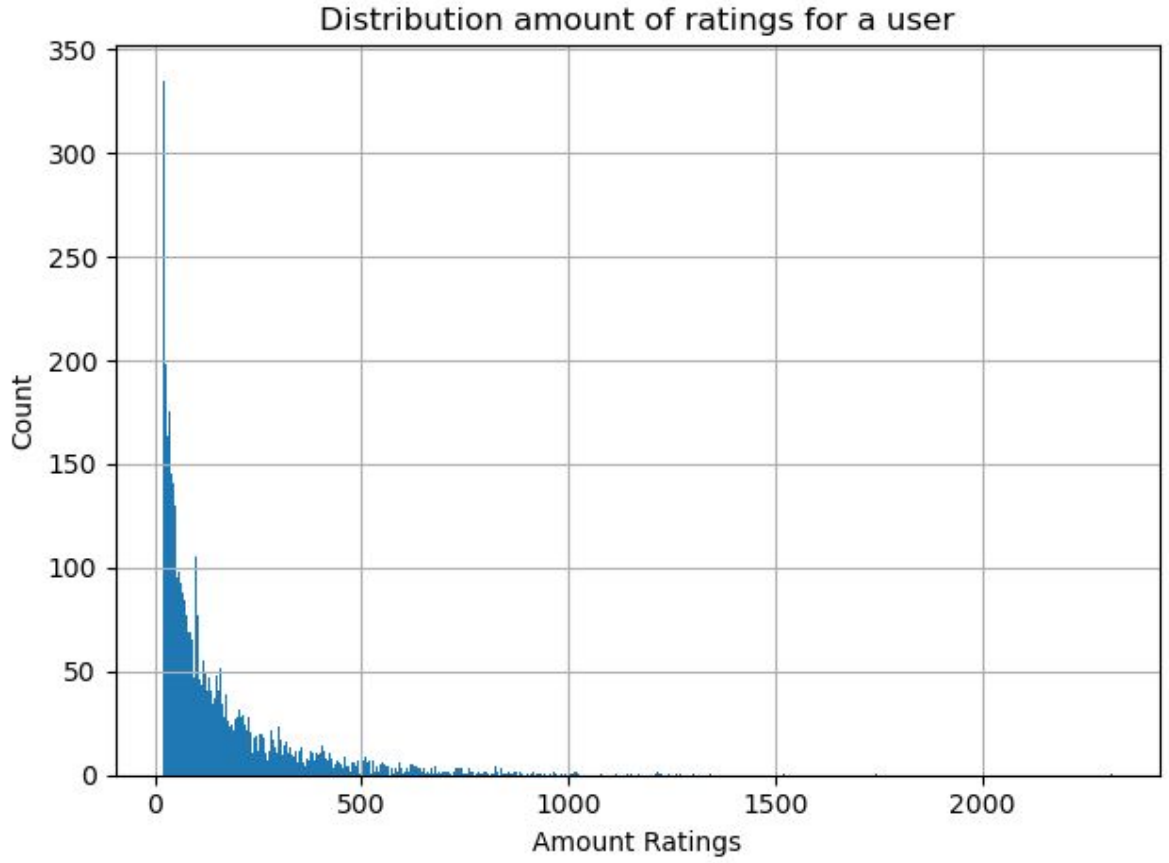


Figure 1: Distribution of user ratings

[1] https://grouplens.org/datasets/movielens/1m/

Further we looked at the distribution of the rating scores themselves. These are shown in Figure 2. From the figure we can see that the distribution has its peak around the score of 4. The distribution seems to have a shift to the higher scores. Which can be explained by users of the site usually only giving ratings for movies they liked, as they want to see similar movies recommended to those.

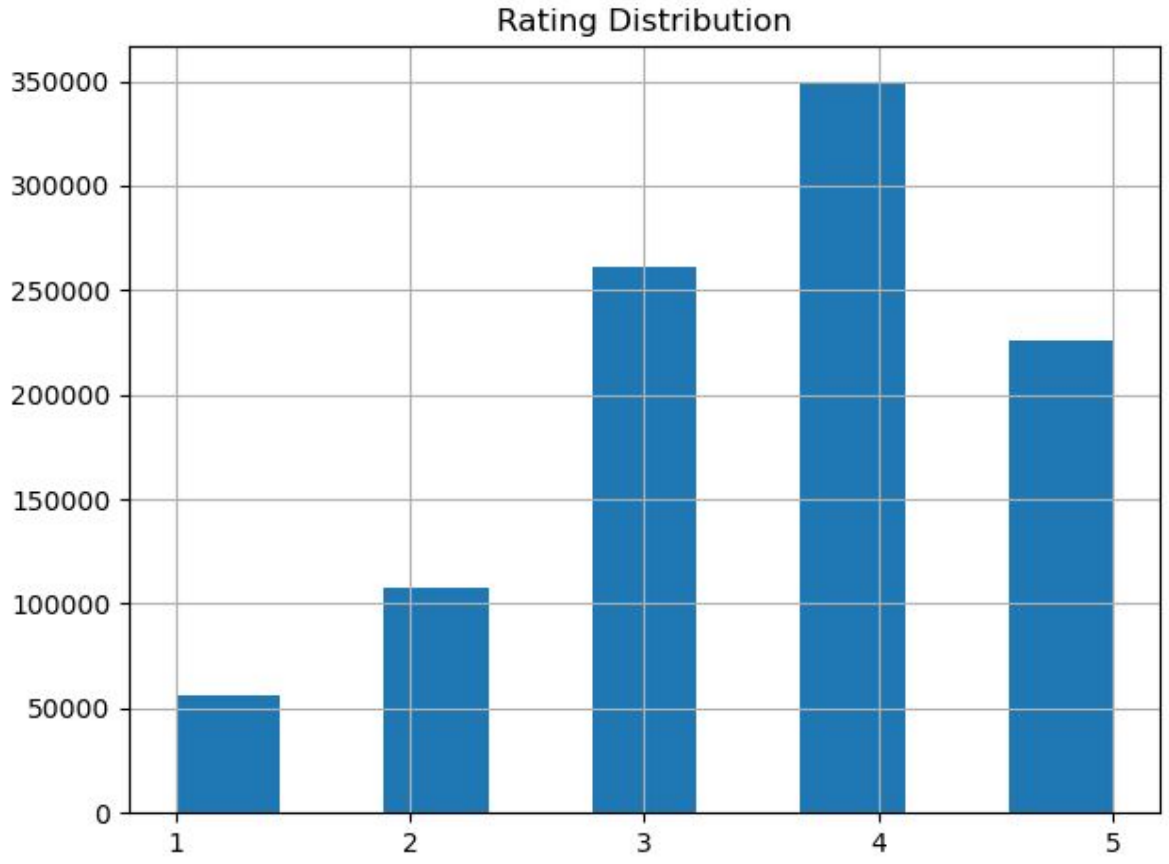


Figure 2: Distribution of rating scores

To perform an analysis of the demographic of the users we show the distribution of the age ranges in Figure 3. This shows a distribution as you would expect, where the majority of users fall within the 25-34 age range.

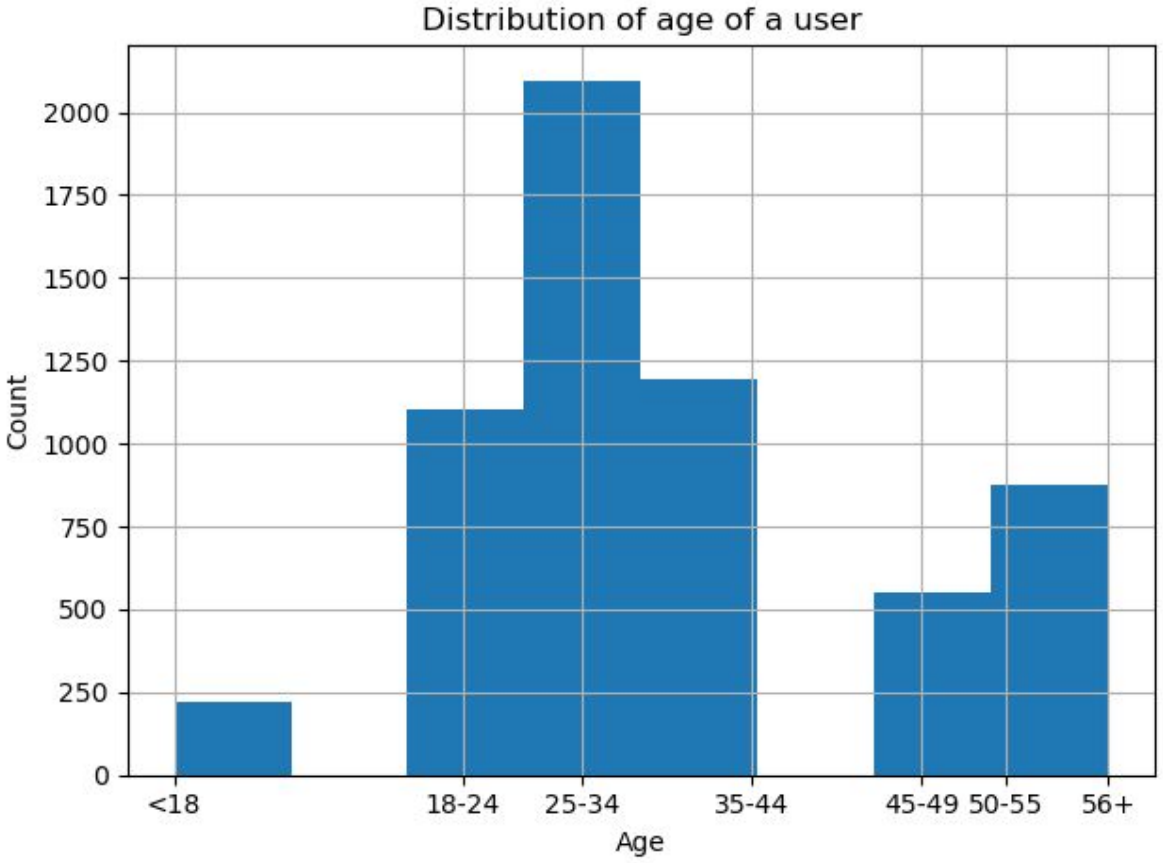


Figure 3: Age group distribution

The Distribution of genres in the dataset seems to follow a distribution where Drama and Comedy are the most common with Action, Romance and Thriller being the next highest. This is similar to the distribution of all global movie genre trends[2].

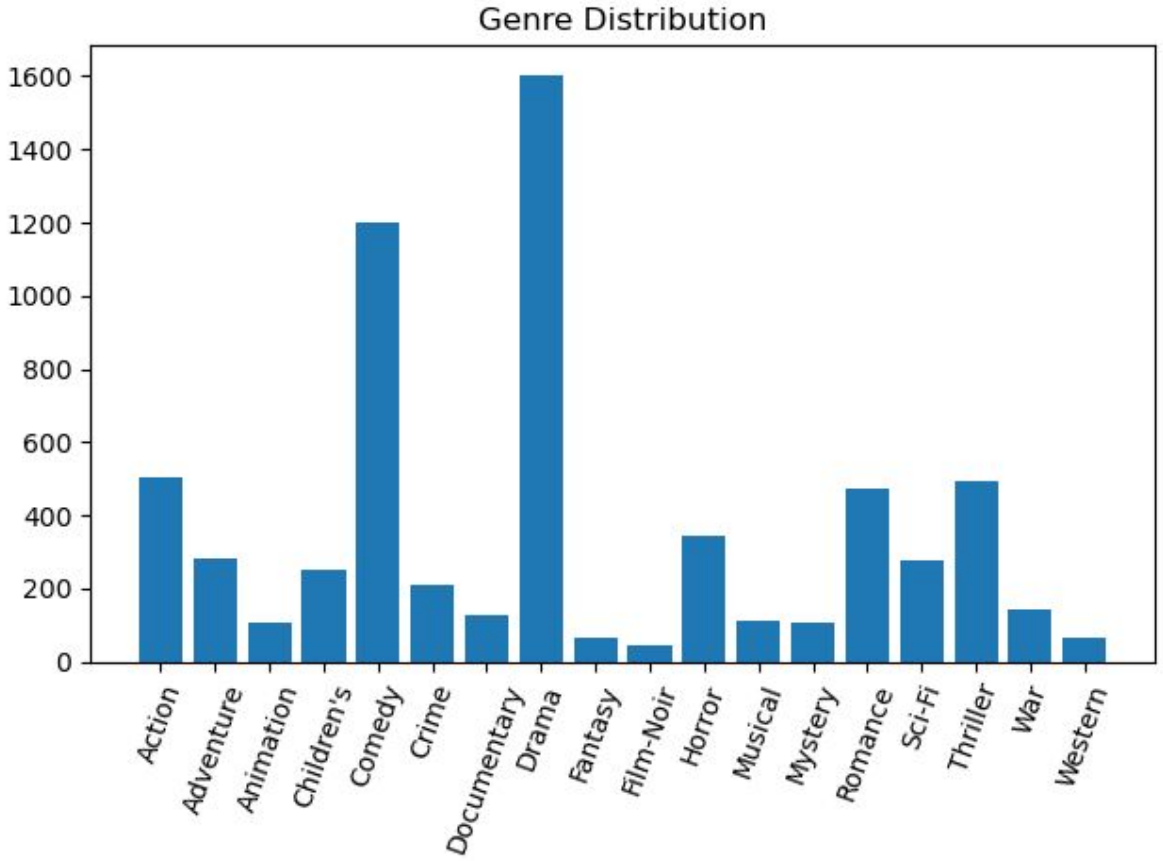


Figure 4: Genre Distribution

[2] https://stephenfollows.com/genre-trends-global-film-production/

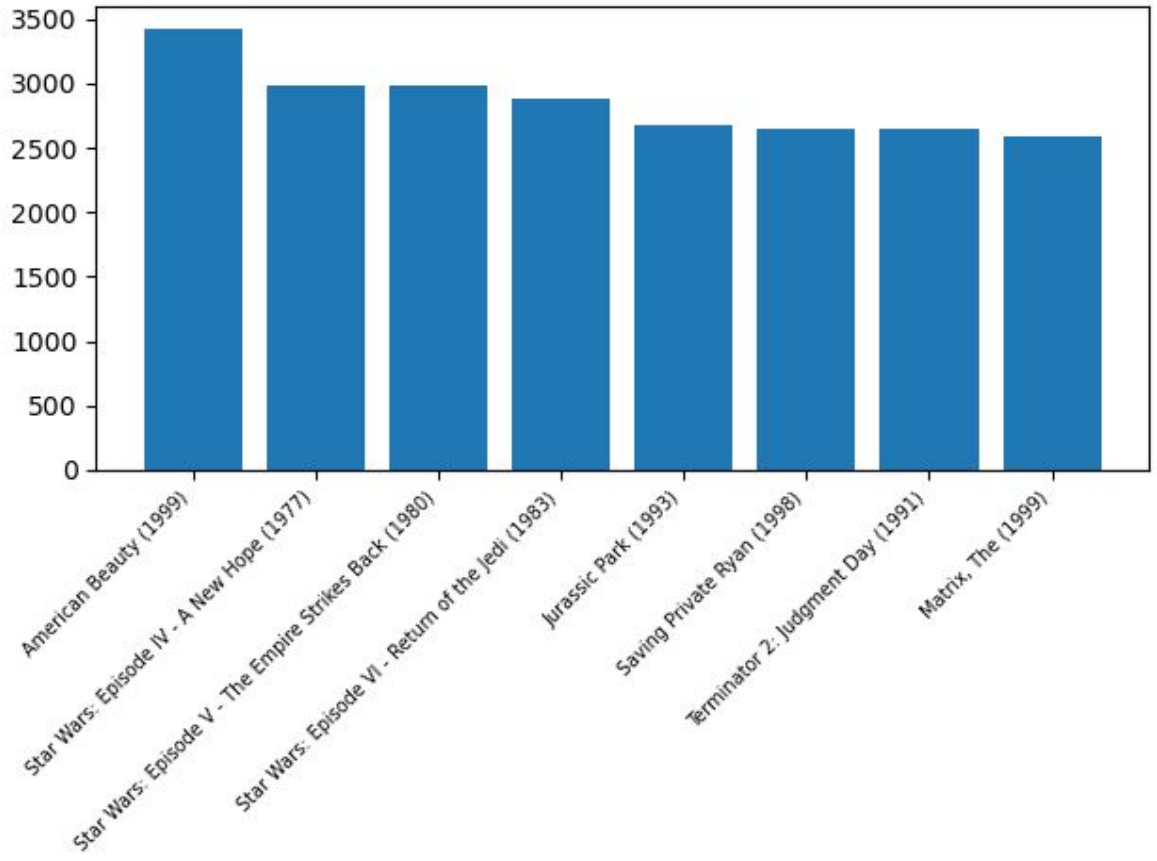
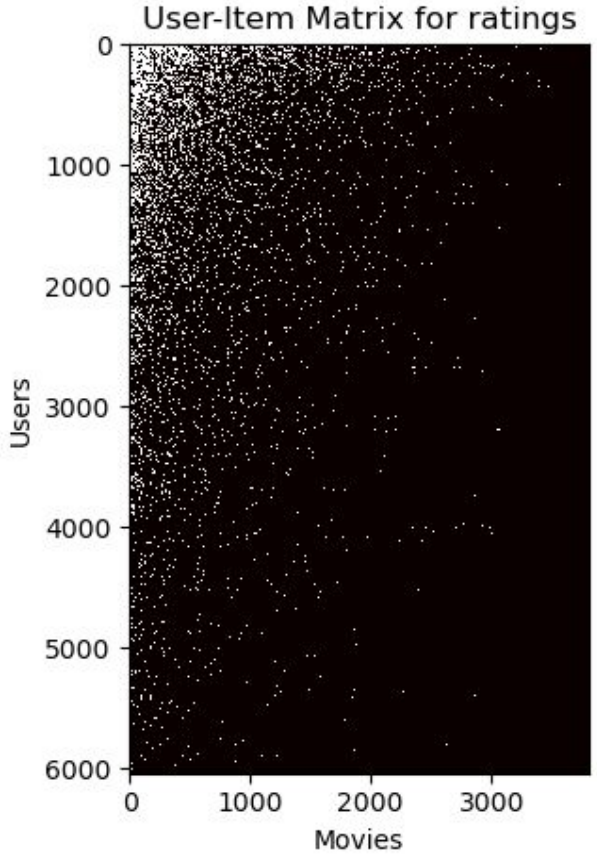


Figure 5: Movies that received the highest amount of ratings


User-Item Matrix for ratings
0
1000
2000
3000
4000
5000
6000
Users
0
1000
2000
3000
Movies

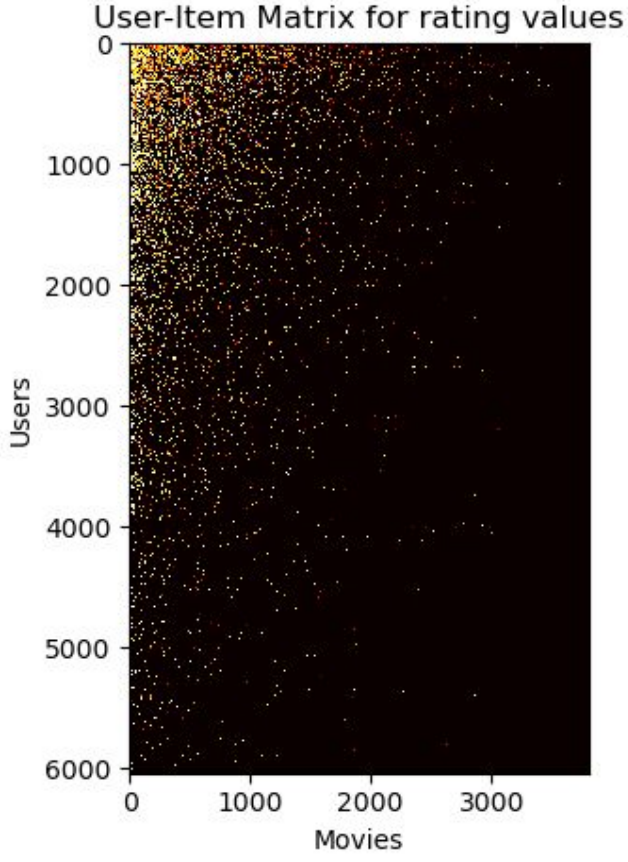

User-Item Matrix for rating values
0
1000
2000
3000
4000
5000
6000
Users
0
1000
2000
3000
Movies

Figure 6: User/Item Matrix heatmap, sorted on most rated movie and user with most ratings

Dataset Problems

We acknowledge that due to the nature of how Movielens works, where a user usually rates movies in batches at a time and not necessarily in the sequence they watched them, that these may not always provide optimal results when used for sequential recommendation purposes. When looking at the difference of the timestamps for the users of ratings made we can see that the average for all users gives a median difference 14 seconds and a mean difference of ~52300 seconds. The mean is prone to outlier values since users take large breaks between subsequent movielens visits.
Even though the dataset suffers from this problem it is still quite an extensive dataset which is able to give a taste profile of the user which can be used to evaluate the effectiveness of the FPMC model for movies.

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

```
import itertools
from datetime import datetime
import os
from itertools import chain, zip_longest
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
import scipy as sp
import statistics as stat

def get_dataset(filename,set):
    if  (set == 'ratings'):
        columns=['user_id', 'item_id', 'rating', 'timestamp']
    elif(set == 'movies'):
        columns = ['movie_id', 'title', 'genre']
    elif(set == 'users'):
        columns = ['user_id', 'gender', 'age', 'occupation', 'zip']
    df = pd.read_csv(filename, sep='::', engine='python', header=None, names=columns)
    return df
```

```
movie_dir = os.path.join('datasets', 'ml-1m')
ratings_file = os.path.join(movie_dir, 'ratings.dat')
movies_file = os.path.join(movie_dir, 'movies.dat')
users_file = os.path.join(movie_dir, 'users.dat')
ratings_pickle = os.path.join(movie_dir, 'ratings.pickle')
movies_pickle = os.path.join(movie_dir, 'movies.pickle')
users_pickle = os.path.join(movie_dir, 'users.pickle')

#Serializes the data
start = datetime.now()
try:
    df_ratings =  pd.read_pickle(ratings_pickle)
    df_movies = pd.read_pickle(movies_pickle)
    df_users = pd.read_pickle(users_pickle)
    print('{} - Retrieved interactions df.'.format(datetime.now() - start))
except Exception as e:
    print('Error unpickling {}, reconstructing '.format(e))
    df_ratings = get_dataset(ratings_file, 'ratings')
    df_movies = get_dataset(movies_file, 'movies')
    df_users = get_dataset(users_file, 'users')
    print('{} - Processed interactions'.format(datetime.now() - start))
    df_ratings.to_pickle(ratings_pickle)
    df_movies.to_pickle(movies_pickle)
    df_users.to_pickle(users_pickle)
    print('{} - Serialized interactions'.format(datetime.now() - start))


#print('ratings: \n',df_ratings.head(10))
#print('movies: \n',df_movies.head(10))
#print('users: \n',df_users.head(10))
#Do not truncate output
#pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
print("Count of user ratings\n ",df_ratings.groupby(['user_id']).count().sort_values(['item_id']))
df_ratings.groupby(['user_id']).count()['item_id'].hist(bins=755)
plt.title("Distribution amount of ratings for a user")
plt.xlabel("Amount Ratings")
plt.ylabel("Count")
hist = df_ratings.hist(bins=9,column='rating',align='mid')
plt.title("Rating Distribution")
plt.xticks([1,2,3,4,5])
plt.show()

#Get movies that are rated most often
values = df_ratings['item_id'].value_counts().keys().tolist()
value_names = [df_movies['title'].loc[df_movies['movie_id'] == id].iat[0] for id in values]
```

```
counts = df_ratings['item_id'].value_counts().tolist()
dict = {'movie_id': values, 'title': value_names, 'counts': counts}
df_moviecounts = pd.DataFrame(dict)
head= df_moviecounts.head(8)
print(head)
print(type(head.counts))
plt.bar(head.title.values,head.counts)
plt.xticks(rotation=45, size =7, ha='right')
plt.show()

#Analysis of age of the users
print("Count of user ages\n ",df_users.groupby(['age']).count().sort_values(['user_id']))
hist = df_users.hist(bins=8,column='age',align='mid')
#plot = df_users['age'].plot.kde(ind=11)
plt.title("Distribution of age of a user")
plt.xlabel("Age")
plt.ylabel("Count")
plt.xticks([1,18,25,35,45,50,56],["<18","18-24","25-34","35-44","45-49","50-55","56+"])

plt.show()



print("Count of user ages\n ",df_movies.groupby(['genre']).count())

def split_genres(movie):
    movie_col = list(movie.columns)
    movie_genres = [doc.split('|') for doc in movie['genre']]
    movie_genres = pd.DataFrame(movie_genres)
    movie = pd.concat([movie, movie_genres], 1)
    movie_col.extend(['genre' + str(i) for i in range(len(movie_genres.columns))])
    movie.columns = movie_col
    return movie
movie = split_genres(df_movies)
print(movie.head(20))
genrecounts =
movie[['genre0','genre1','genre2','genre3','genre4','genre5']].apply(pd.Series.value_counts).s
um(axis=1)
print("Count of genre\n
",genrecounts)#.groupby(['genre0','genre1','genre2','genre3','genre4']).size())

print(genrecounts.values)
#plt.hist(genrecounts.values)
#genrecounts.hist(bins=17)
plt.bar(genrecounts.index.values, genrecounts.values)
plt.title("Genre Distribution")
plt.xticks(rotation=70)
```

```
plt.show()


#mean time difference
medians = []
means=[]
for user in df_ratings['user_id'].unique():
    print(user)
    oneuser = df_ratings[df_ratings['user_id']==user]
    #print(oneuser)
    timestamps= oneuser['timestamp'].values.tolist()
    timestamps.sort()
    #print(timestamps)
    differences = [x - timestamps[i - 1] for i, x in enumerate(timestamps)][1:]
    means.append(stat.mean(differences))
    medians.append(stat.median(differences))
    print(differences)
    #print("MEAN ",stat.mean(differences))
    #print("MEDIAN ", stat.median(differences))
print(medians,"\n", means)
print("medians",stat.mean(medians))
print("means", stat.mean(means))
```