Executive Summary:

Recommendation systems are one of the most popular application of Machine Learning in the business industry. Be it social media or E-commerce, recommendation systems have revolutionized the business dynamics. In addition to this, recommendation systems are also used for entertainment purposes as Netflix, Pandora and Movie Lens. Using "Movie Ratings Data" from www.movielens.org web site, we tried different ways of data preprocessing and recommendation systems to understand the data and how different machine learning algorithm works on them. During data exploration, we found out the data was very sparse, and we filled in none values with two different ways: Filling none values with 0, and filling none values with 0 and then normalizing the data.

With the two different preprocessed data, we created train and test data and attempted simple prediction of test data based on cosine similarity and Pearson coefficient matrix of train data. Then we tried recommender system using top-k collaborative filtering also using cosine and Pearson. We used another calculation method, Jaccard distance for the simple prediction and collaborative filtering. For Jaccard distance, we tried another data type and created boolean data from our original data.

For each of the different techniques used to build recommendation system, we performed both user-based and item-based and calculated mean absolute error to see which method and calculation produced the smallest mae and to find out the optimal k. At the end, we tried the famous matrix factorization and overall was able to compare all and see which method produced the highest prediction accuracy.

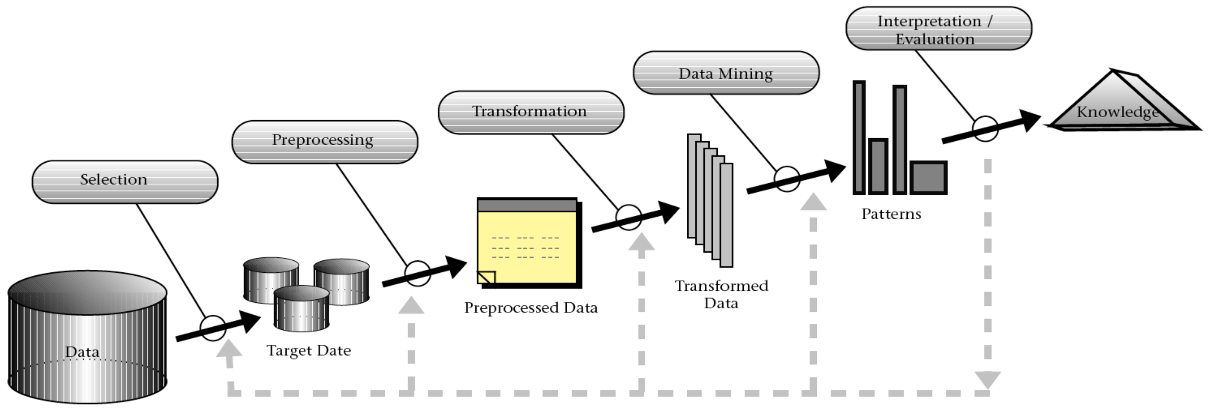
Introduction:

With the technological advances and boom of the internet in the past two decades, consumers today face the problem of information explosion. Due to the availability of so many resources, it is an extremely difficult to look for a good restaurant which pertains to the need of the user and also has high quality. In order to assist the consumers to cope with this explosion of information, businesses have started using recommendation system. In general terminology, a recommendation system looks at the interests and preference of an individual and based on that it can suggest different items and vice versa. Research in the field of recommendation system has been going on for a long time, but even then the level of interest seems pretty high because of the numerous applications and new potential opportunities.

The goal of this project is to explore different types of techniques which can be used to create recommendation system using different similarity metrics. In addition to this, we will also explore different techniques to preprocess the data and see if it will affect the recommendation system.

Methodology:

For the purpose of this analysis, we will follow the traditional Knowledge Discovery Process. The first step is the overview about the data where we will explain the different sources of data that have been used. The next step is to pre-processing data, which is considered to be one of the crucial aspect of data analysis. After the preprocessing has been completed, exploratory data visualization provides the basic insight about the data and provides background information which is essential for the successful implementation of the data analysis process. Under the umbrella of the Data mining we will be using various data analysis techniques. Finally, our results will be summarized in conclusion also specifying future direction of work.



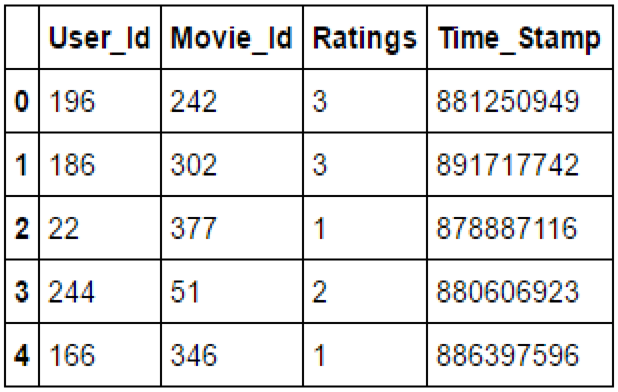
Data Description:

The movie rating data used for our project has been collected from www.movielens.org. The whole dataset comprises a combination of files. One file includes demographic information about the user and other details. There is another file which also contains the names of the movie, their genre, their date of release. The information from these files has been used for the process of exploratory data analysis. Finally, the last file contains user ratings for each movie.

However, since our objective here was to explore different similarity metrics and their application on different versions of pre-processed data, we just take into account the file which contains information about the user, movies and their respective ratings. This file has been used over the course of the report for further analysis.

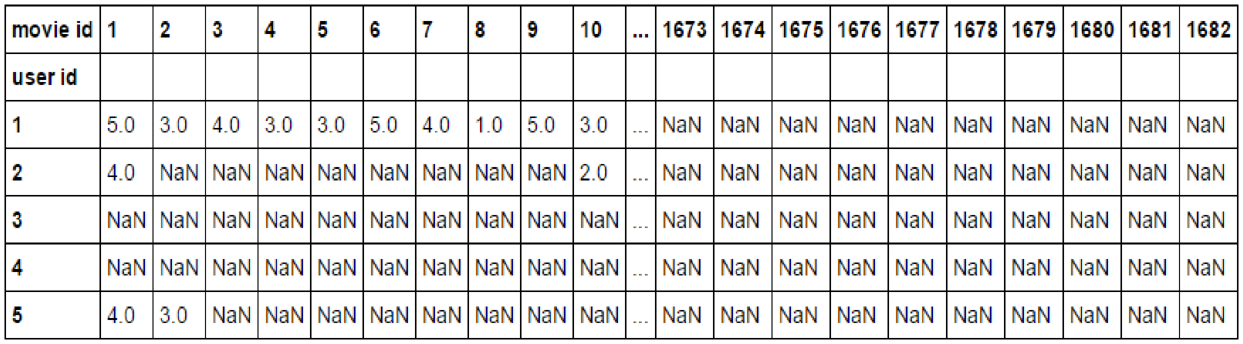
Data Preprocessing:

The data file with user rating contains 943 users and 1682 movies. The total number of ratings provided by all users within the data is 100,000. In addition to this, the data file also contains timestamp, which provides information as to how recent the rating was given by a particular user to a particular movie. A sample of the data is displayed below



	User_Id	Movie_Id	Ratings	Time_Stamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

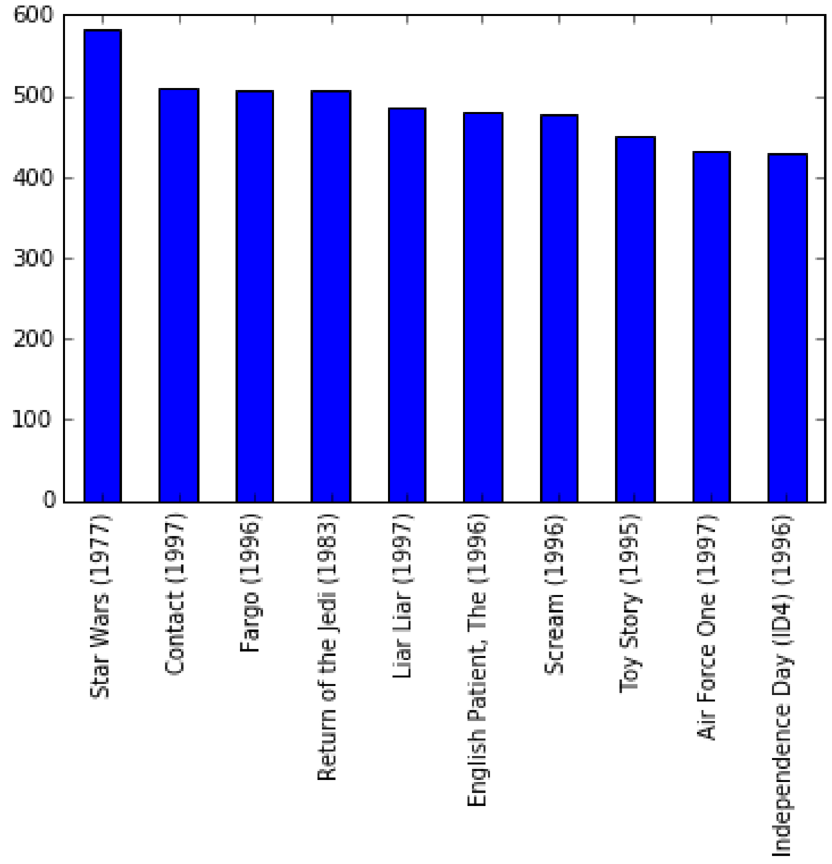
Since we were using a dataset which has been used previously, there was not much pre-processing to be performed. However, from the context of our data analysis, we had to transform the data. The data had 4 columns (user id | item id | rating | timestamp). We dropped timestamp column and transformed the data such that each row represents each user, each column represents each item(movie), and values are ratings. A snapshot of the data is displayed below:



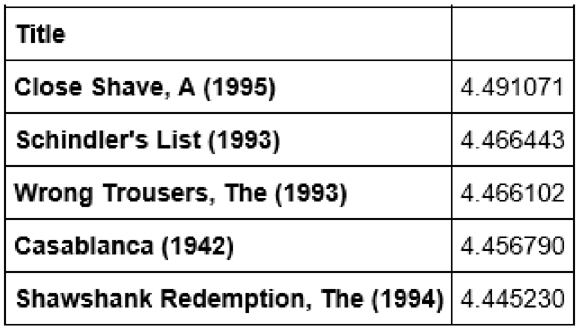
movie id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
user id																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Exploratory Data Analysis:

For the purpose of exploratory analysis, we were intrigued by identifying the movies which were rated the most by the User. The graph below visualizes top 10 movies that were rated the most number of times, with Star Wars leading the list having been rated 583 times.

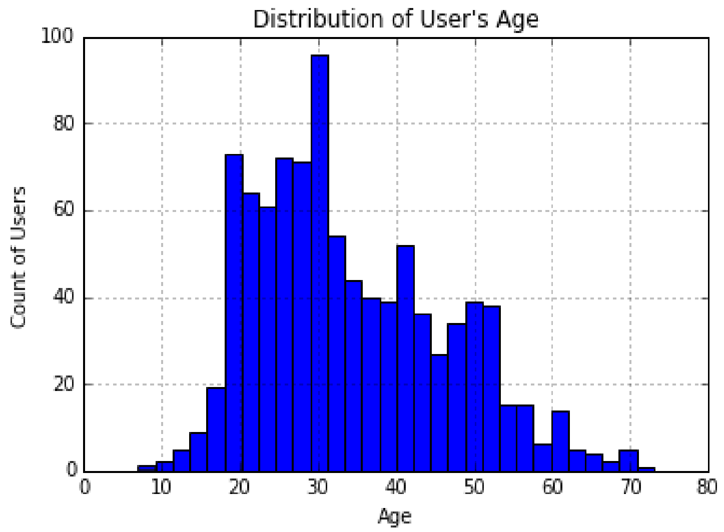


The list of Top 5 Movies that have been rated by more than 100 users and have the highest average rating are represented below in the table:

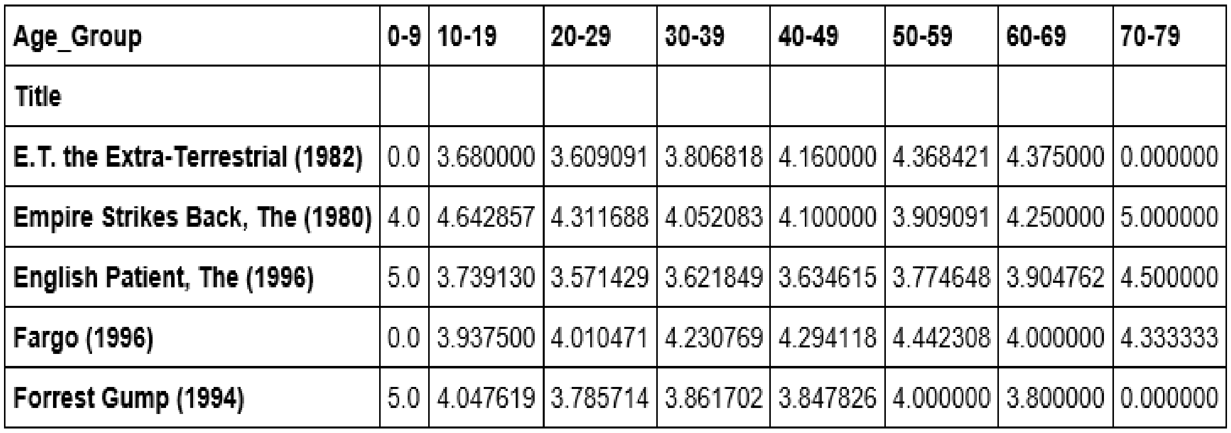


Title	
Close Shave, A (1995)	4.491071
Schindler's List (1993)	4.466443
Wrong Trousers, The (1993)	4.466102
Casablanca (1942)	4.456790
Shawshank Redemption, The (1994)	4.445230

In terms of the users, the age was an important factor as to know the customer base of Movie Lens and the individuals who rate movies. The visualization show below displays the distribution of age. Individuals of ages from 7 years till 73 years have rated the movies. In addition to this, the average age of user is 34 years.

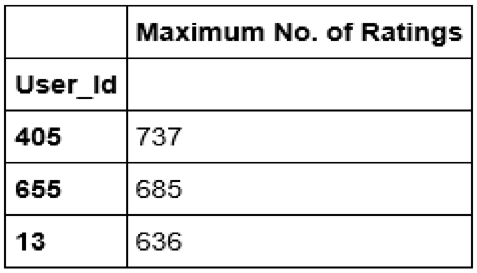


Further expanding the scope of exploratory analysis, we also analyzed the popularity of the top 50 most rated movies amongst users belonging to different age groups were performed. The overall pattern indicated that the users who were young were more critical of the ratings for the movie in comparison to the other users. The table below displays the average ratings for 5 movies.

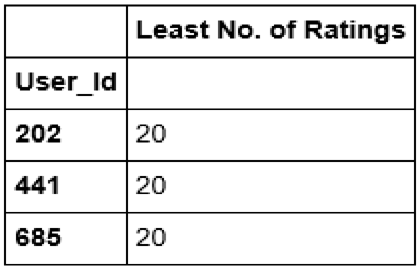


Age_Group	0-9	10-19	20-29	30-39	40-49	50-59	60-69	70-79
Title								
E.T. the Extra-Terrestrial (1982)	0.0	3.680000	3.609091	3.806818	4.160000	4.368421	4.375000	0.000000
Empire Strikes Back, The (1980)	4.0	4.642857	4.311688	4.052083	4.100000	3.909091	4.250000	5.000000
English Patient, The (1996)	5.0	3.739130	3.571429	3.621849	3.634615	3.774648	3.904762	4.500000
Fargo (1996)	0.0	3.937500	4.010471	4.230769	4.294118	4.442308	4.000000	4.333333
Forrest Gump (1994)	5.0	4.047619	3.785714	3.861702	3.847826	4.000000	3.800000	0.000000

For users perspective, the top three users rated a total of 737, 685 and 636 movies, whereas the least number of movies rated by a user is 20.

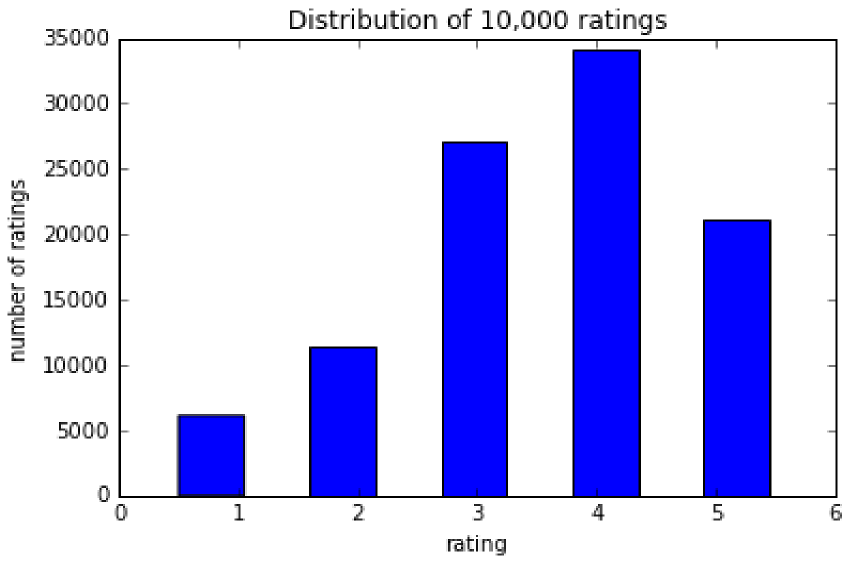


	Maximum No. of Ratings
User_Id	
405	737
655	685
13	636



	Least No. of Ratings
User_Id	
202	20
441	20
685	20

This section covers exploratory analysis done on the movie ratings. The distribution of the rating is visualized below. It can be observed that most number of the ratings given by user to different movies as 4. In addition to this, the average rating was 3.53



.

We will primarily be focusing on Collaborative Filtering which will be used for User as well as Item based recommendation.

Collaborative Filtering:

Collaborative Filtering is a technique that is widely used in building the recommendation systems. It is a process of filtering for patterns/information using techniques involving collaboration from different sources. In the context of recommendation system, it is a method which makes automatic prediction about the interest of a user by collecting his taste or preferences from other users.

User-Based Collaborative Filtering:

User-based collaborative filtering predicts a test user's interest in a test item based on rating information from similar user profiles. Each user is presented as a vector having his/her preferences or taste about different items which build up the profile of that user. Each user profile is sorted by its dis-similarity towards the test user's profile. If the profile of two users are similar to one another, they will contribute more to predicting the test item rating. The set of similar users can be identified by employing a threshold or by selecting top-k nearest users. In the top-k case, a set of top-k similar users towards user are used to generate the prediction. Pearson's correlation, cosine similarity and jaccard are three widely used similarity measure in collaborative filtering.

Item-Based Collaborative Filtering:

Item Based Collaborative filtering is another variation of collaborative filtering, however instead of the user this is a collaborative filtering algorithm which uses an item-oriented approach. Similarity between various items are computed and based on that the prediction is computed on an item for a particular user using the weighted sum of the rating given by the user to similar items. As in the case of user based collaborative filtering, we can also consider the top K most similar items to the given item. We observe that setting K to its maximum possible value, which is the number of items other than a particular item rated by user u, gives the best results in terms of the accuracy error.

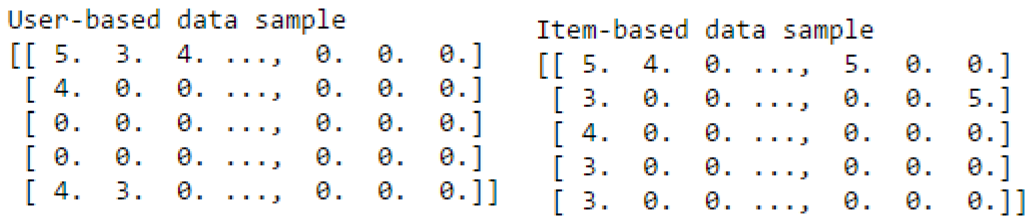
We will explore 4 different avenues of data analysis for the scope of this project. The first three techniques will use a different similarity metric for collaborative filtering while the last one will be Matrix Factorization, the scalable and effective technique that has been used in the field of recommendation system.

Data Preprocessing:

Although data preprocessing has been discussed previously, we will briefly discuss the different ways with which the data is transformed before it can be used for the application of collaborative filtering algorithm.

As seen is the snapshot of the data, there are a lot missing values, which needs to be filled. For the perspective of cosine similarity, Pearson correlation and Matrix Factorization, we rely on the following two techniques.

The first technique is just to fill all the missing values with 0. This approach is widely used and is the building block that we have used for our analysis. After filling the missing values with 0, our data set looks something like this:

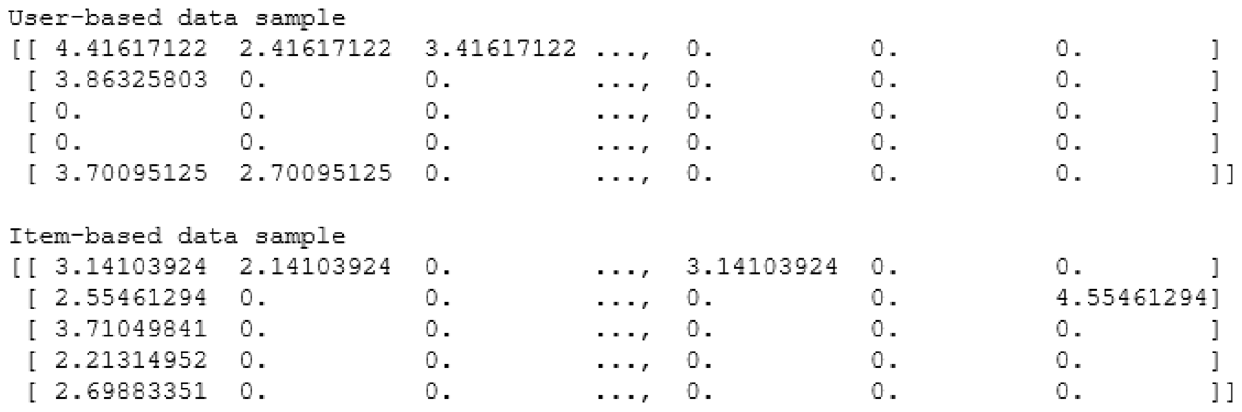


```
User-based data sample
[[ 5.  3.  4. ...,  0.  0.  0.]
 [ 4.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 4.  3.  0. ...,  0.  0.  0.]]
```

```
Item-based data sample
[[ 5.  4.  0. ...,  5.  0.  0.]
 [ 3.  0.  0. ...,  0.  0.  5.]
 [ 4.  0.  0. ...,  0.  0.  0.]
 [ 3.  0.  0. ...,  0.  0.  0.]
 [ 3.  0.  0. ...,  0.  0.  0.]]
```

In addition to this, we also explore another technique which has been previously been used with cosine similarity metrics, however, we will also explore the impact of this technique using other similarity metric. For Matrix Factorization, however we just used the matrix where the missing values were filled with 0.

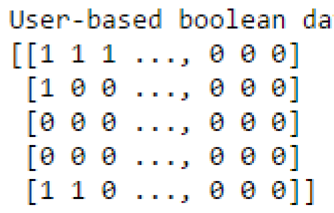
The second technique uses normalized ratings which are computed by subtracted the average rating of that user from each of their ratings. This converts a low rating into negative number, whereas a high number into positive number. Similar process is applied normalize item based ratings. A snapshot of the transformed data set after the application of this technique is visualized below:



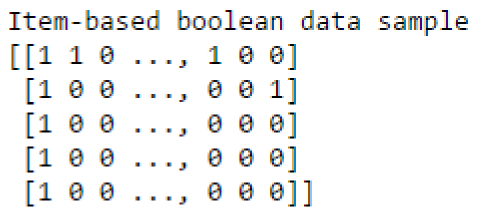
```
User-based data sample
[[ 4.41617122  2.41617122  3.41617122 ...,  0.          0.          0.        ]
 [ 3.86325803  0.          0.         ...,  0.          0.          0.        ]
 [ 0.          0.          0.         ...,  0.          0.          0.        ]
 [ 0.          0.          0.         ...,  0.          0.          0.        ]
 [ 3.70095125  2.70095125  0.         ...,  0.          0.          0.        ]]

Item-based data sample
[[ 3.14103924  2.14103924  0.         ...,  3.14103924  0.          0.        ]
 [ 2.55461294  0.          0.         ...,  0.          0.          4.55461294]
 [ 3.71049841  0.          0.         ...,  0.          0.          0.        ]
 [ 2.21314952  0.          0.         ...,  0.          0.          0.        ]
 [ 2.69883351  0.          0.         ...,  0.          0.          0.        ]]
```

For using Jaccard, as it deals with binary values, we ignored empty values in the matrix (consider them as 0) and focus only on the sets of items rated (consider them as 1). This method works the best if the utility matrix only reflected purchases. A snapshot of the item based and user based data that will be used for collaborative filtering is displayed below:



```
User-based boolean data sample
[[1 1 1 ..., 0 0 0]
 [1 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [1 1 0 ..., 0 0 0]]
```



```
Item-based boolean data sample
[[1 1 0 ..., 1 0 0]
 [1 0 0 ..., 0 0 1]
 [1 0 0 ..., 0 0 0]
 [1 0 0 ..., 0 0 0]
 [1 0 0 ..., 0 0 0]]
```

Data Validation:

For the purpose of validation, we will be using the complete data set. The training set will be a copy of the complete data. However, ‘n’ number of non-zero ratings from the training set will be removed and these data points will be used to create a test set. So, our training set will have all the values except for the 10 non-zero values, whereas the testing set will just have 10 values that are absent from the training set. From validation perspective, we will just validate our predictions for these 10 ratings for each user and vice versa.

As seen in the exploratory analysis, that the least amount of movies that a user has rated is 20, so we have used 10 values for the purpose of validation. This will enable us to have at least half of ratings for users which can be used for training the model.

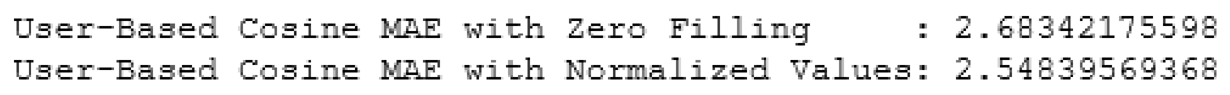
Technique No 1 – Recommendation System Using Cosine Similarity:

Following is the list of function that have been used for applying the collaborative filtering algorithm using cosine similarity:

- **train_test**: creates train and test data from original data. It randomly selects “n” number of non-zero ratings from the original data and creates test data. At the same time, the selected non-empty ratings become 0 in train data. As users gave at least 20 ratings, the biggest value of n can be 20. We used the half value, 10
- **sim_mat_cos**: computes and creates a whole cosine similarity matrix for user or item-based recommendation system
- **recommend_mat**: computes and creates a whole prediction matrix for user or item-based recommendation system
- **compute_error_simple_cos**: computes mean absolute error (mae) or mean squared error (mse) between train and test data. We primarily used mae
- **k_collaborative**: creates a whole prediction matrix depending on k for collaborative filtering
- **compute_error_coll_cos**: computes mean absolute error (mae) or mean squared error (mse) between train and test data based on k for collaborative filtering
- **optimal_k**: using different values of k in a list, it creates a table of mae of user and item-based recommender systems
- **opt_error_graph**: plots two line graphs of user and item-based recommender systems’ mae based on k-values

- **User Based Collaborative Filtering:**

Once the data has been filled out, we use both of the transformed data types and run User Based Collaborative Filtering. The Collaborative Filtering is done on the basis of the whole matrix using cosine similarity. We used 10 as n-value to create train and test data from the original data. So in the test data, each row gets 10 randomly selected non-empty ratings. We used two different preprocessed data for empty values: zero filling and normalized values. Based on cosine similarity matrix from train data, we predicted the 10 selected values in the test data and calculated mean absolute errors.



```
User-Based Cosine MAE with Zero Filling      : 2.68342175598
User-Based Cosine MAE with Normalized Values: 2.54839569368
```

- **Item Based Collaborative Filtering:**

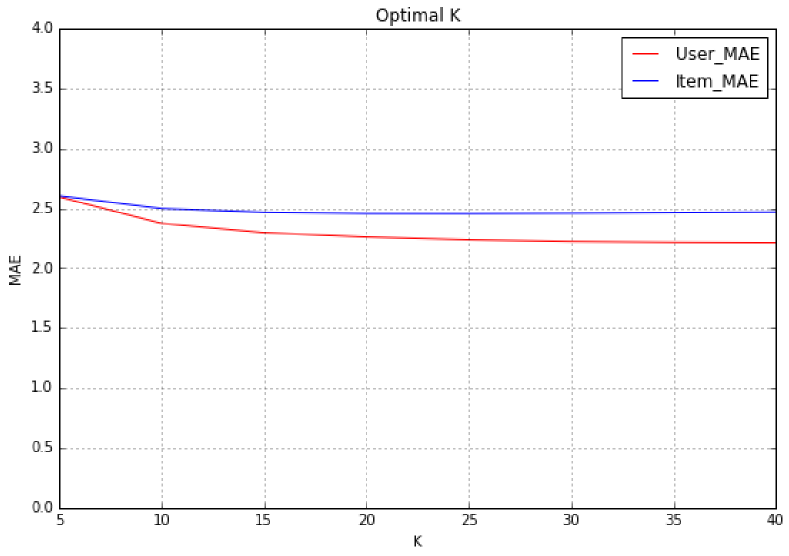
The Item Based Collaborative Filtering algorithm will be applied on both sets of transformed data; one with zero filling and other with normalized values. The Collaborative Filtering is done on the basis of the whole matrix using cosine similarity. We used 10 as n-value to create train and test data from the original data. So in the test data, each row gets 10 randomly selected non-empty ratings. Based on cosine similarity matrix from train data, we predicted the 10 selected values in the test data and calculated mean absolute errors.

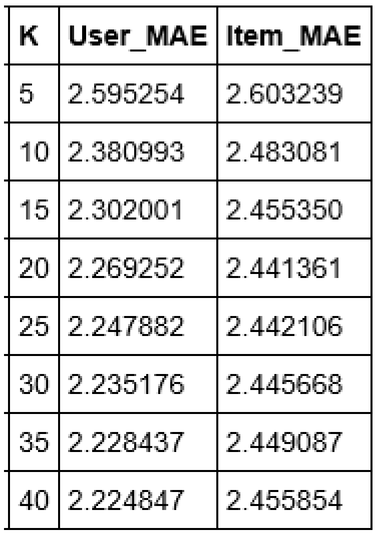
```
Item-Based Cosine MAE with Zero Filling      : 2.82448881747
Item-Based Cosine MAE with Normalized Values: 2.59667606752
```

Selecting the Optimal k Using Collaborative Filtering

In this step, we will apply K-Collaborative Filtering on both User & Item Based. Using "optimal_k" and "opt_error_graph" function, we selected optimal k-values to minimizing mean absolute values for user and item-based recommendation systems for two different data respectively.

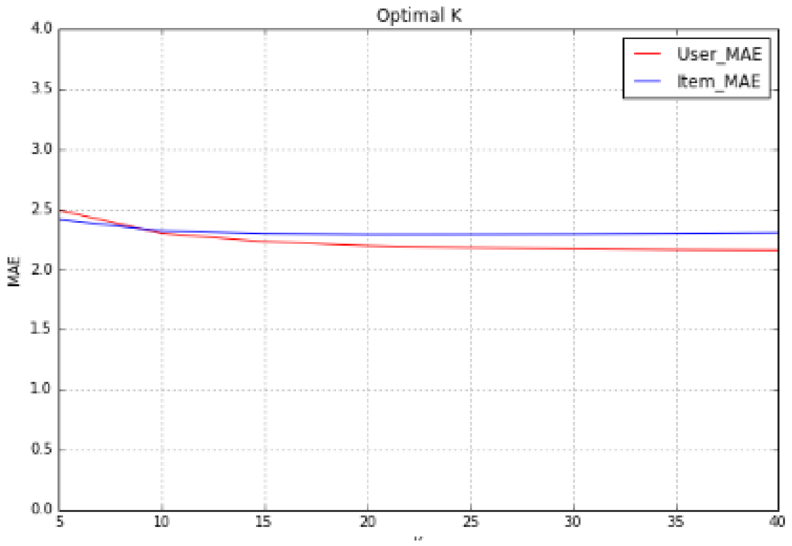
< Zero filling data >

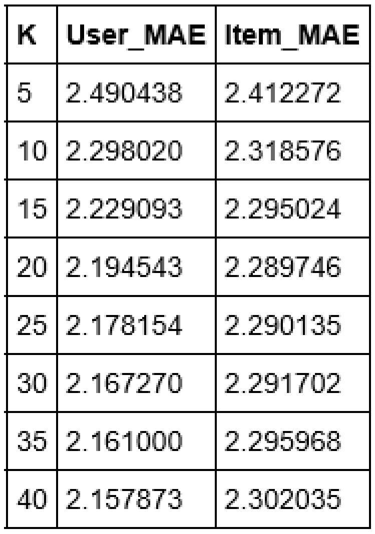




K	User_MAE	Item_MAE
5	2.595254	2.603239
10	2.380993	2.483081
15	2.302001	2.455350
20	2.269252	2.441361
25	2.247882	2.442106
30	2.235176	2.445668
35	2.228437	2.449087
40	2.224847	2.455854

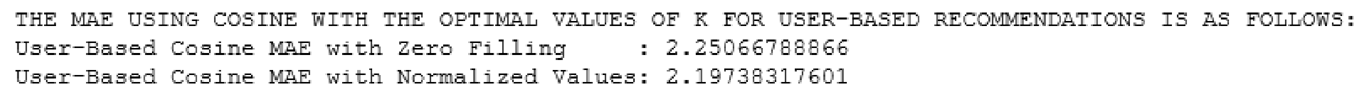
< Normalized Rating Data >



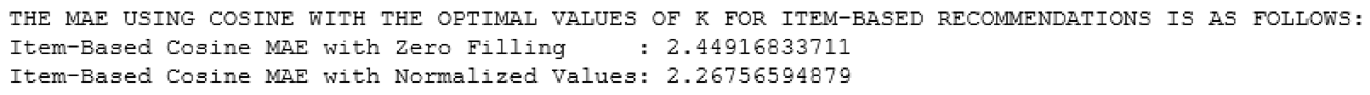


K	User_MAE	Item_MAE
5	2.490438	2.412272
10	2.298020	2.318576
15	2.229093	2.295024
20	2.194543	2.289746
25	2.178154	2.290135
30	2.167270	2.291702
35	2.161000	2.295968
40	2.157873	2.302035

Based on zero filling data, the optimal k for user based is 25, whereas for item-based it is 20. Similarly, for normalized value data, the optimal k for user based is 20, whereas for item based is 15.



```
THE MAE USING COSINE WITH THE OPTIMAL VALUES OF K FOR USER-BASED RECOMMENDATIONS IS AS FOLLOWS:
User-Based Cosine MAE with Zero Filling     : 2.25066788866
User-Based Cosine MAE with Normalized Values: 2.19738317601
```



```
THE MAE USING COSINE WITH THE OPTIMAL VALUES OF K FOR ITEM-BASED RECOMMENDATIONS IS AS FOLLOWS:
Item-Based Cosine MAE with Zero Filling     : 2.44916833711
Item-Based Cosine MAE with Normalized Values: 2.26756594879
```

Technique No 2 – Recommendation System Using Pearson Correlation:

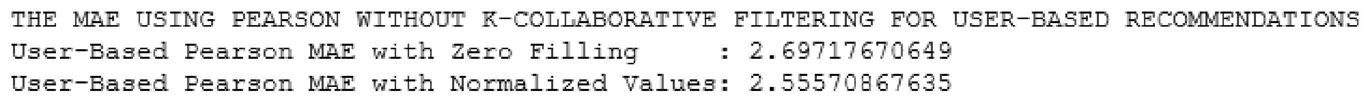
Following is the list of function that have been used for applying the collaborative filtering algorithm using Pearson Correlation:

- **sim_mat_pear:** computes and creates a whole Pearson coefficient matrix for user or item-based recommendation system
- **compute_error_simple_pear:** computes mae or mse between predicted and test data. The prediction gets created from Pearson coefficient matrix
- **compute_error_coll_pear:** computes mae or mse between predicted and test data based on k for collaborative filtering and using Pearson coefficients

- **optimal_k_pear:** creates a table of mae of user and item-based recommender systems based on k-values. The calculation method is Pearson coefficient and this helps you to find the optimal k for small mae
- **opt_error_graph_p:** visualizes the output of "optimal_k_pear: function" as line graphs

User Based Collaborative Filtering:

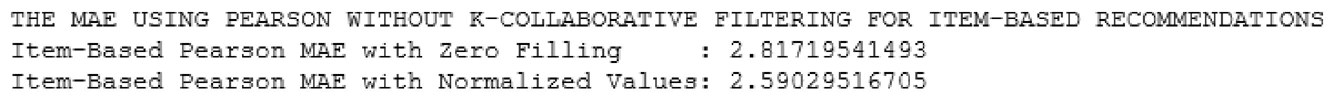
The Collaborative Filtering is done on the basis of the whole matrix using Pearson similarity The process is very similar to Data analysis 1 (using cosine similarity). So we made train and test data with n value 10 and using Pearson coefficient matrix from the train data, we predicted the 10 selected values in the test data and calculated mean absolute errors.



```
THE MAE USING PEARSON WITHOUT K-COLLABORATIVE FILTERING FOR USER-BASED RECOMMENDATIONS
User-Based Pearson MAE with Zero Filling     : 2.69717670649
User-Based Pearson MAE with Normalized Values: 2.55570867635
```

Item Based Collaborative Filtering:

The Item Based Collaborative Filtering algorithm will be applied on both sets of transformed data; one with zero filling and other with normalized values using Pearson similarity. The process is very similar to Data analysis 1 (using cosine similarity). So we made train and test data with n value 10 and using Pearson coefficient matrix from the train data, we predicted the 10 selected values in the test data and calculated mean absolute errors.

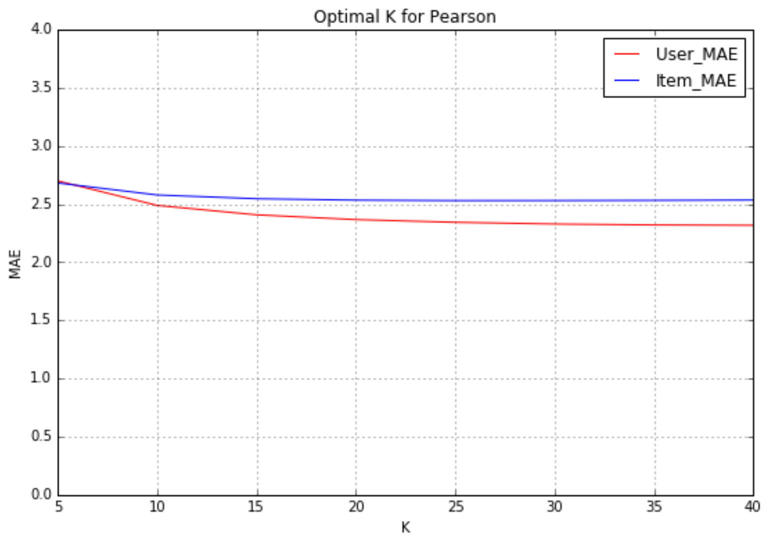


```
THE MAE USING PEARSON WITHOUT K-COLLABORATIVE FILTERING FOR ITEM-BASED RECOMMENDATIONS
Item-Based Pearson MAE with Zero Filling     : 2.81719541493
Item-Based Pearson MAE with Normalized Values: 2.59029516705
```

Selecting the Optimal k Using Collaborative Filtering

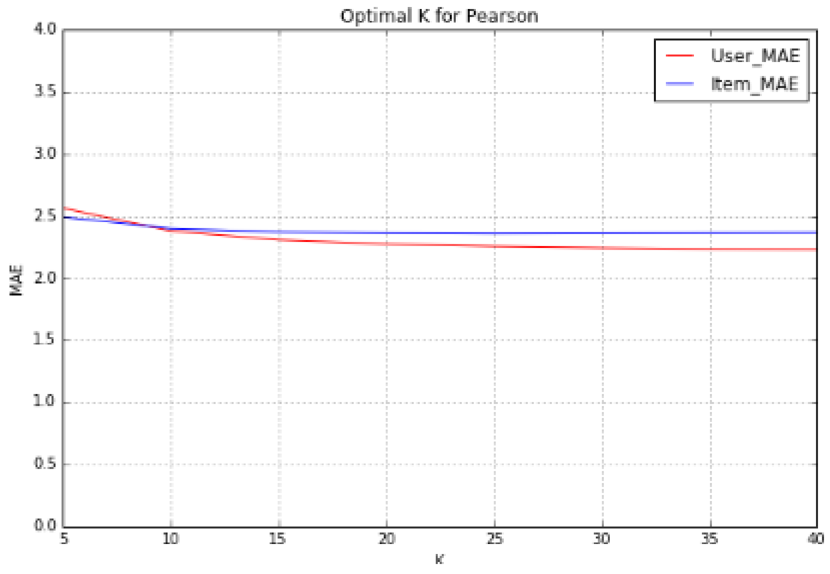
In this step, we will apply K-Collaborative Filtering on both User & Item Based. Using "optimal_k" and "opt_error_graph" function, we selected optimal k-values to minimizing mean absolute values for user and item-based recommendation systems for two different data respectively.

< Zero filling data >



K	User_MAE	Item_MAE
5	2.700455	2.681314
10	2.492151	2.568922
15	2.415820	2.533421
20	2.372629	2.520913
25	2.352886	2.517836
30	2.340097	2.516073
35	2.330892	2.517796
40	2.325157	2.521365

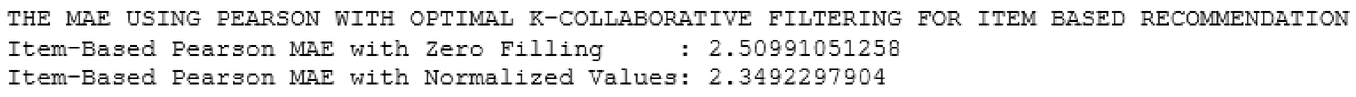
< Normalized Value data >



K	User_MAE	Item_MAE
5	2.563810	2.491721
10	2.377274	2.396084
15	2.306496	2.368747
20	2.272385	2.359350
25	2.255308	2.355782
30	2.242453	2.357780
35	2.234211	2.360398
40	2.230489	2.363117

Based on zero filling data, the optimal k for user based is 25, whereas for item-based it is 25. Similarly, for normalized value data, the optimal k for user based is 15, whereas for item based is 15.

```
THE MAE USING PEARSON WITH OPTIMAL K-COLLABORATIVE FILTERING FOR USER-BASED RECOMMENDATIONS
User-Based Pearson MAE with Zero Filling     : 2.35934323987
User-Based Pearson MAE with Normalized Values: 2.26459624471
```



```
THE MAE USING PEARSON WITH OPTIMAL K-COLLABORATIVE FILTERING FOR ITEM BASED RECOMMENDATION
Item-Based Pearson MAE with Zero Filling     : 2.50991051258
Item-Based Pearson MAE with Normalized Values: 2.3492297904
```

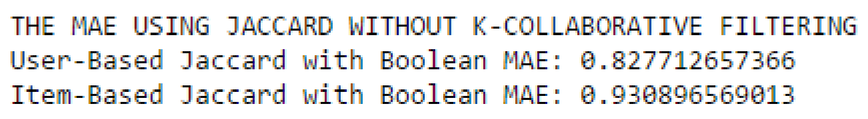
Technique No 3 – Recommendation System Using Jaccard:

Following is the list of function that have been used for applying the collaborative filtering algorithm using Jaccard:

- **jac_data:** transforms the data to boolean type so empty values becomes 0 and all 1 through 5 ratings become 1
- **sim_mat_jac:** computes and creates a whole Jaccard distance matrix for user or item-based recommendation system
- **compute_error_simple_jac:** computes mean absolute error (mae) or mean squared error (mse) between predicted and test data. (We primarily used mae) The prediction gets created from Jaccard distance matrix
- **compute_error_coll_jac:** computes mean absolute error (mae) or mean squared error (mse) between predicted and test data based on k for collaborative filtering and using Jaccard distance
- **optimal_k_jac:** creates a table of mae of user and item-based recommender systems based on k-values. The calculation method is Jaccard distance and this helps you to find the optimal k for small mae
- **opt_error_graph_j:** visualizes the output of "optimal_k_jac: function" as line graphs

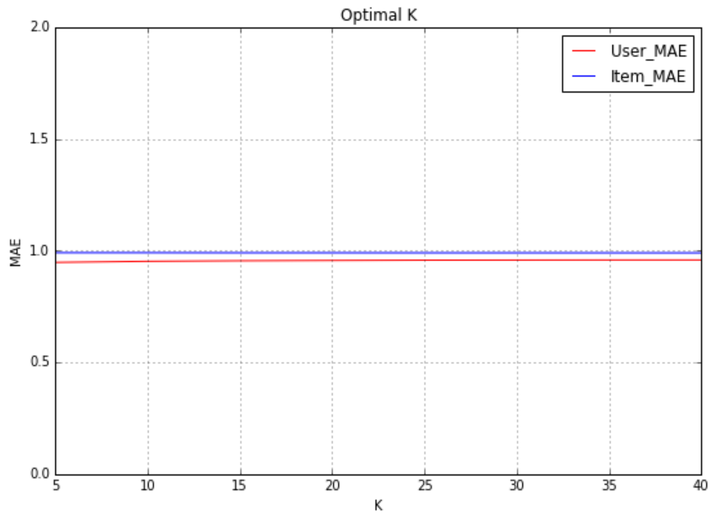
User Based & Item Based Collaborative Filtering:

For Jaccard distance, we used one boolean data. After making the boolean data, we created train and test data and performed prediction and calculated mean absolute error. The mae is much lower. However, we shouldn't compare them as Jaccard distance used boolean data.



```
THE MAE USING JACCARD WITHOUT K-COLLABORATIVE FILTERING
User-Based Jaccard with Boolean MAE: 0.827712657366
Item-Based Jaccard with Boolean MAE: 0.930896569013
```

Selecting the Optimal k Using Collaborative Filtering



K	User_MAE	Item_MAE
5	2.700455	2.681314
10	2.492151	2.568922
15	2.415820	2.533421
20	2.372629	2.520913
25	2.352886	2.517836
30	2.340097	2.516073
35	2.330892	2.517796
40	2.325157	2.521365

For Jaccard, we can see that the mae is very consistent with different k-values:

```
THE MAE USING JACCARD WITH THE OPTIMAL VALUE OF K IS AS FOLLOWS:
User-Based Jaccard with Boolean MAE: 0.951172758912
Item-Based Jaccard with Boolean MAE: 0.990510590531
```

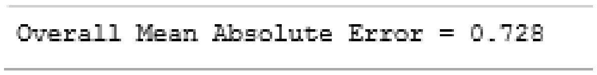
Technique No 4 – Recommendation System Using Matrix Factorization:

Matrix Factorization has become one of the most successful application of latent factor model. Matrix Factorization basically characterizes both the items and users by vectors of factors which are inferred from item rating patterns. High correspondence between item and user factors leads to a recommendation. These methods have become popular in recent years by combining good scalability with predictive accuracy. In addition, they offer much flexibility for modeling various real-life situations. Following functions are used for creating a recommendation system using matrix factorization.

- **matrix_factorization:** This function takes in the matrix and decomposes into two matrix which allow us to identify the latent features underlying the interaction between users and movies (items).
- **compute_mae_matrix:** computes mae between predicted and test data using matrix factorization

Validation

Based on the results from Matrix Factorization, the Mae is:



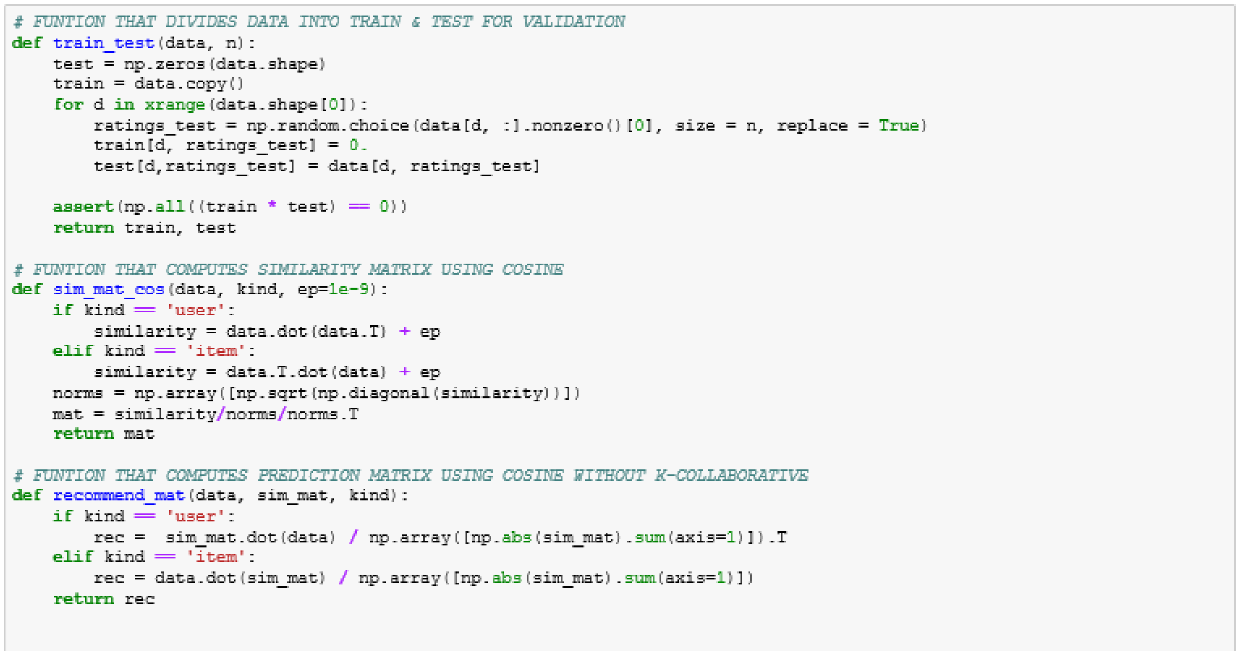
```
Overall Mean Absolute Error = 0.728
```

Conclusion & Future Work:

In conclusion, it is always debatable how to fill in none values, but in this case based on the different analysis techniques that we have used, filling in the data and normalizing the data proved to be much effective in comparison to just filling in the missing values with zero. Between cosine similarity and Pearson coefficient, for user-based recommendation system as well as the item based recommendation systems, the normalized data seems to be doing better in terms of the accuracy relative to the zero filling values, as the Mean Absolute Error was lower. For the case of Jaccard, as the data was converted into binary data, it wasn't a feasible option to compare the results with other technique. Overall, matrix factorization recommender system has far better accuracy.

For a future direction of work, we can look at clustering the users and items and try to build a recommendation system based on the clustering results.

APPENDIX

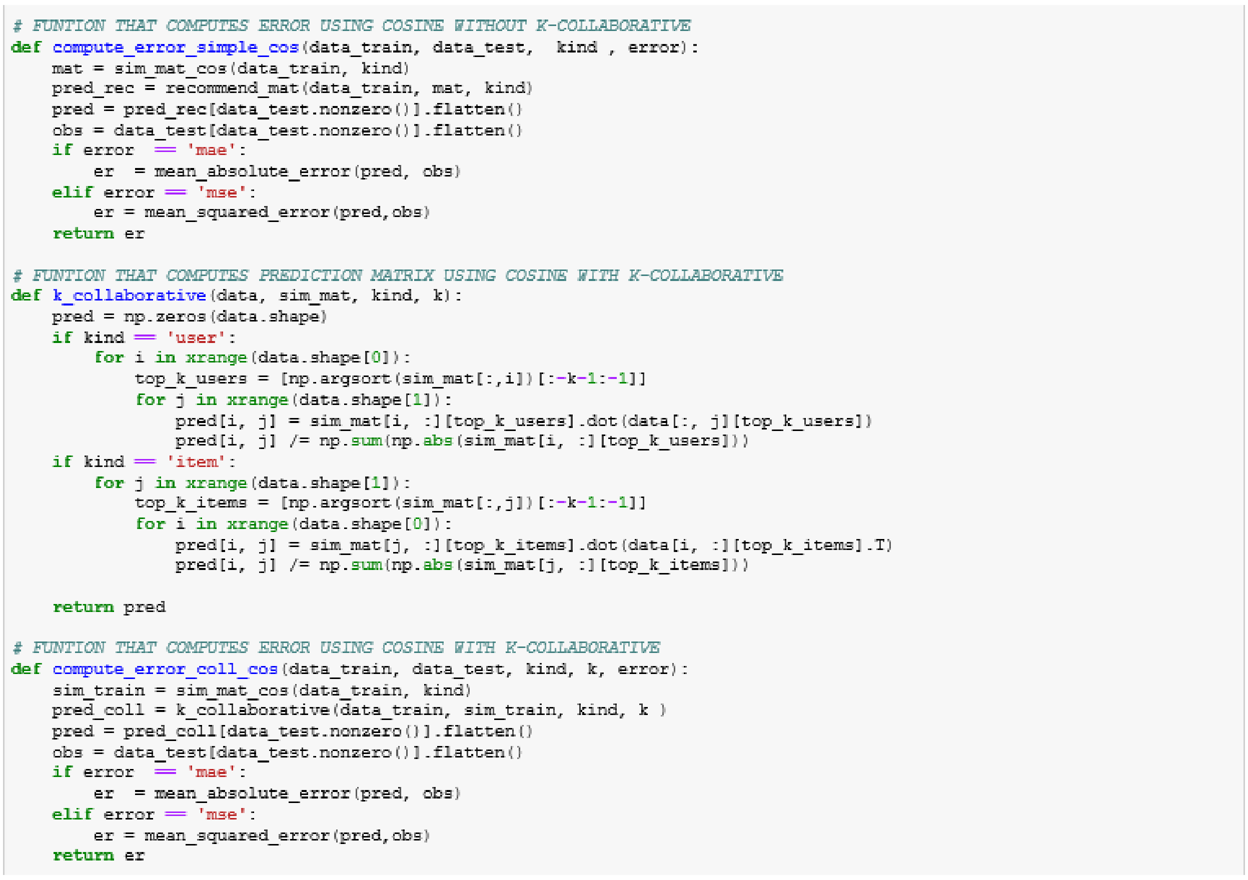


```
# FUNTION THAT DIVIDES DATA INTO TRAIN & TEST FOR VALIDATION
def train_test(data, n):
    test = np.zeros(data.shape)
    train = data.copy()
    for d in xrange(data.shape[0]):
        ratings_test = np.random.choice(data[d, :].nonzero()[0], size = n, replace = True)
        train[d, ratings_test] = 0.
        test[d,ratings_test] = data[d, ratings_test]

    assert(np.all((train * test) == 0))
    return train, test

# FUNTION THAT COMPUTES SIMILARITY MATRIX USING COSINE
def sim_mat_cos(data, kind, ep=1e-9):
    if kind == 'user':
        similarity = data.dot(data.T) + ep
    elif kind == 'item':
        similarity = data.T.dot(data) + ep
    norms = np.array([np.sqrt(np.diagonal(similarity))])
    mat = similarity/norms/norms.T
    return mat

# FUNTION THAT COMPUTES PREDICTION MATRIX USING COSINE WITHOUT K-COLLABORATIVE
def recommend_mat(data, sim_mat, kind):
    if kind == 'user':
        rec =  sim_mat.dot(data) / np.array([np.abs(sim_mat).sum(axis=1)]).T
    elif kind == 'item':
        rec = data.dot(sim_mat) / np.array([np.abs(sim_mat).sum(axis=1)])
    return rec
```



```
# FUNTION THAT COMPUTES ERROR USING COSINE WITHOUT K-COLLABORATIVE
def compute_error_simple_cos(data_train, data_test,  kind , error):
    mat = sim_mat_cos(data_train, kind)
    pred_rec = recommend_mat(data_train, mat, kind)
    pred = pred_rec[data_test.nonzero()].flatten()
    obs = data_test[data_test.nonzero()].flatten()
    if error  == 'mae':
        er  = mean_absolute_error(pred, obs)
    elif error == 'mse':
        er = mean_squared_error(pred,obs)
    return er

# FUNTION THAT COMPUTES PREDICTION MATRIX USING COSINE WITH K-COLLABORATIVE
def k_collaborative(data, sim_mat, kind, k):
    pred = np.zeros(data.shape)
    if kind == 'user':
        for i in xrange(data.shape[0]):
            top_k_users = [np.argsort(sim_mat[:,i])[:-k-1:-1]]
            for j in xrange(data.shape[1]):
                pred[i, j] = sim_mat[i, :][top_k_users].dot(data[:, j][top_k_users])
                pred[i, j] /= np.sum(np.abs(sim_mat[i, :][top_k_users]))
    if kind == 'item':
        for j in xrange(data.shape[1]):
            top_k_items = [np.argsort(sim_mat[:,j])[:-k-1:-1]]
            for i in xrange(data.shape[0]):
                pred[i, j] = sim_mat[j, :][top_k_items].dot(data[i, :][top_k_items].T)
                pred[i, j] /= np.sum(np.abs(sim_mat[j, :][top_k_items]))

    return pred

# FUNTION THAT COMPUTES ERROR USING COSINE WITH K-COLLABORATIVE
def compute_error_coll_cos(data_train, data_test, kind, k, error):
    sim_train = sim_mat_cos(data_train, kind)
    pred_coll = k_collaborative(data_train, sim_train, kind, k )
    pred = pred_coll[data_test.nonzero()].flatten()
    obs = data_test[data_test.nonzero()].flatten()
    if error  == 'mae':
        er  = mean_absolute_error(pred, obs)
    elif error == 'mse':
        er = mean_squared_error(pred,obs)
    return er
```

```
# FUNTION THAT DISPLAYS ERROR AT DIFFERENT K
def optimal_k(data_train, data_test,  kind, list_of_k, error):
    user_mae = []
    item_mae = []
    df = pd.DataFrame(index=list_of_k)
    for k in list_of_k:
        if kind == 'user':
            mae = compute_error_coll_cos(data_train, data_test,  kind, k , error)
            user_mae.append(mae)
        else:
            mae = compute_error_coll_cos(data_train, data_test,  kind, k , error)
            item_mae.append(mae)
    if kind  == 'user':
        df['User_MAE'] = user_mae
    else:
        df['Item_MAE'] = item_mae
    return df

# FUNTION THAT VISUALIZES ERROR AT DIFFERENT K
def opt_error_graph(train_u, test_u, train_i, test_i, list_k, error):
    df_user = optimal_k(train_u, test_u, 'user', list_k, error)
    df_item = optimal_k(train_i, test_i, 'item', list_k, error)
    df_tot = df_user.copy()
    df_tot['K'] = list_k
    df_tot['Item_MAE'] = df_item.Item_MAE
    df_tot = df_tot[['K', 'User_MAE', 'Item_MAE']]
    plt.plot(df_tot.K, df_tot.User_MAE, 'r')
    plt.plot(df_tot.K, df_tot.Item_MAE, 'b')
    params = plt.gcf()
    plSize = params.get_size_inches()
    params.set_size_inches( (plSize[0]*1.5, plSize[1]*1.5) )
    plt.grid()
    plt.ylim(0,4)
    plt.xticks(df_tot.K)
    plt.legend(['User_MAE', 'Item_MAE'])
    plt.title('Optimal K')
    plt.xlabel("K")
    plt.ylabel("MAE")
    plt.show()
    return df_tot
```

Jaccard Distance Functions

```
# CREATE DATA FOR JACCARD
def jac_data(data):
    X = data.copy()
    X = X.astype(bool).astype(int)
    return X

# DISTANCE MATRIX
def sim_mat_jac(data, kind):
    ep = 1e-9
    if kind =='user':
        intersect = data.dot(data.T) + ep
        intersect = np.array(intersect)
        sum_row = intersect.diagonal()
        union = sum_row[:,None] + sum_row - intersect
        mat = 1.0 - intersect / union
    elif kind == 'item':
        intersect = (data.T).dot(data) + ep
        intersect = np.array(intersect)
        sum_row = intersect.diagonal()
        union = sum_row[:,None] + sum_row - intersect
        mat = 1.0 - intersect / union
    return mat

# FUNCTION COMPUTES ERROR USING JACCARD WITHOUT K_COLLABORATIVE
def compute_error_simple_jac(data_train, data_test, kind , error):
    mat = sim_mat_jac(data_train, kind)
    pred_rec = recommend_mat(data_train, mat,  kind)
    pred = pred_rec[data_test.nonzero()].flatten()
    obs = data_test[data_test.nonzero()].flatten()
    if error  == 'mae':
        er  = mean_absolute_error(pred, obs)
    elif error == 'mse':
        er = mean_squared_error(pred,obs)
    return er
```

```
# FUNCTION COMPUTES ERROR USING JACCARD WITH K_COLLABORATIVE
def compute_error_coll_jac(data_train, data_test,kind, k , error):
    sim_train = sim_mat_jac(data_train, kind)
    pred_coll = k_collaborative(data_train, sim_train, kind, k )
    pred = pred_coll[data_test.nonzero()].flatten()
    obs = data_test[data_test.nonzero()].flatten()
    if error  == 'mae':
        er  = mean_absolute_error(pred, obs)
    elif error == 'mse':
        er = mean_squared_error(pred,obs)
    return er

# FUNCTION COMPUTES OPTIMAL K USING JACCARD
def optimal_k_jac(data_train, data_test,  kind, list_of_k, error):
    user_mae = []
    item_mae = []
    df = pd.DataFrame(index=list_of_k)
    for k in list_of_k:
        if kind == 'user':
            mae = compute_error_coll_jac(data_train, data_test, kind, k, error)
            user_mae.append(mae)
        else:
            mae = compute_error_coll_jac(data_train, data_test, kind, k, error)
            item_mae.append(mae)
    if kind  == 'user':
        df['User_MAE'] = user_mae
    else:
        df['Item_MAE'] = item_mae
    return df
```