E-COMMERCE PRODUCT RECOMMENDATION SYSTEM

A PROJECT REPORT

Submitted by

JUMANA BEGUM J (2019103022)

SAMREEN AYESHA S (2019103055)

JESIMA A (2019103529)

for the course

CS 6026 – SOCIAL NETWORK ANALYSIS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY

CHENNAI 600 025

DECEMBER 2022

ABSTRACT

In our everyday life we use e-commerce websites to shop for many items. This leads to an increasing diversity in consumers' demand, turning into a challenge for retail stores to provide the right products according to customer preferences. These e-commerce websites use a variety of recommendation systems to cope with this challenge. Recommender systems are changing from novelties used by a few E-commerce sites, to serious business tools that are reshaping the world of E-commerce. A recommender system learns from a customer and recommends products that he/she will find most valuable from among the available products. Through product recommendation it is possible to fulfill customers' needs and expectations, thereby helping maintain loyal customers while attracting new customers.

INTRODUCTION

E-commerce recommendations have two parts. First one is the initial phase when the user joins the e-commerce website with no purchasing history. Another is when the user has a purchasing history. When the user has no purchasing history the e-commerce recommender faces the cold-start problem. In such cases the recommender resolves the issue by recommending the most frequently purchased items or items with high ratings. And once the user starts purchasing, we aim to understand how the recommender recommends highly rated similar products or highly rated products bought by similar customers. In this project we are using the ratings given to a product by other users to make recommendations to the current user. The KNN algorithm is used with Pearson's Correlation Coefficient to make recommendations to the user by predicting the ratings of that user for all products and then recommending the top 5 products. Both item based and user based KNN recommendations are done and their performances are compared.

DATASET

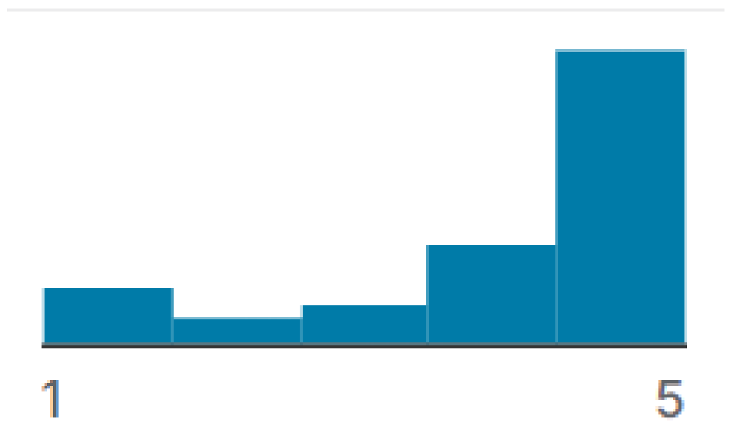
The project is implemented based on the AMAZON electronic products rating dataset. It has 4 columns:

1. UserID: Every user identified with a unique ID
2. ProductID: Every product identified with a unique ID
3. Rating: Rating of corresponding product by corresponding user
4. Timestamp: Time of the rating

By dataset exploration we have identified that the dataset has

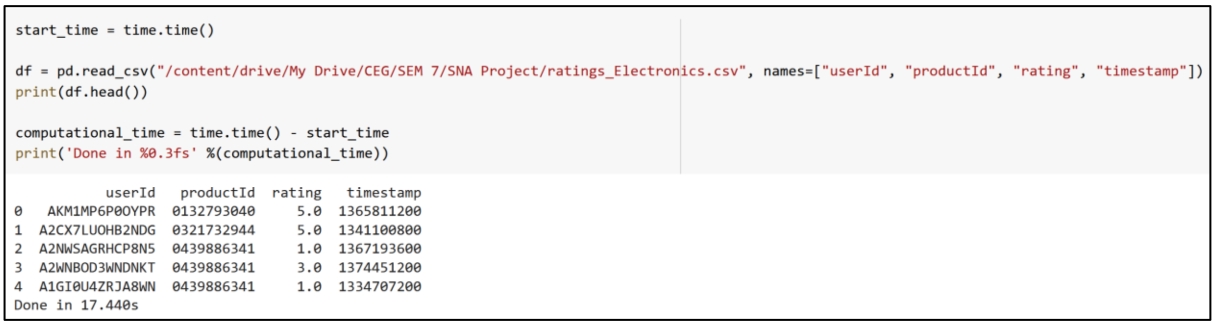
1. 78 lakh records
2. 42 lakh unique user records
3. 4 lakh and 76 thousand unique product records

When analysing the rating, the following distribution is found.



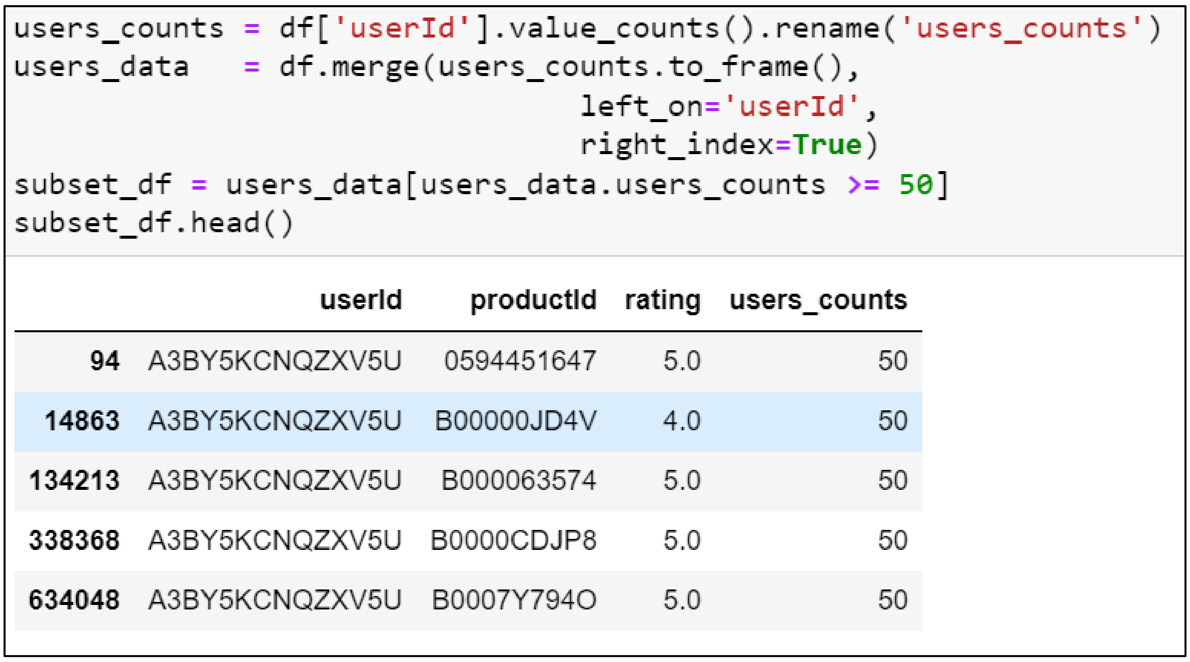
IMPLEMENTATION SCREENSHOTS

We reduced the dataset to 65k records by selecting users who provided more than fifty ratings and products with more than five ratings.



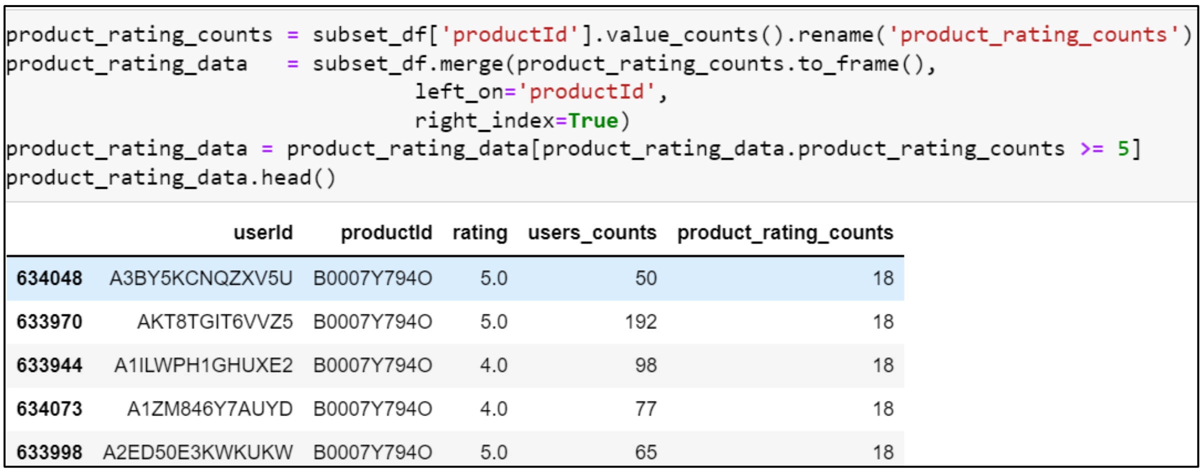
```python
start_time = time.time()

df = pd.read_csv("/content/drive/My Drive/CEG/SEM 7/SNA Project/ratings_Electronics.csv", names=["userId", "productId", "rating", "timestamp"])
print(df.head())

computational_time = time.time() - start_time
print('Done in %0.3fs' %(computational_time))
```

```
          userId   productId  rating   timestamp
0   AKM1MP6P0OYPR  0132793040     5.0  1365811200
1  A2CX7LUOHB2NDG  0321732944     5.0  1341100800
2  A2NWSAGRHCP8N5  0439886341     1.0  1367193600
3  A2WNBOD3WNDNKT  0439886341     3.0  1374451200
4  A1GI0U4ZRJA8WN  0439886341     1.0  1334707200
Done in 17.440s
```



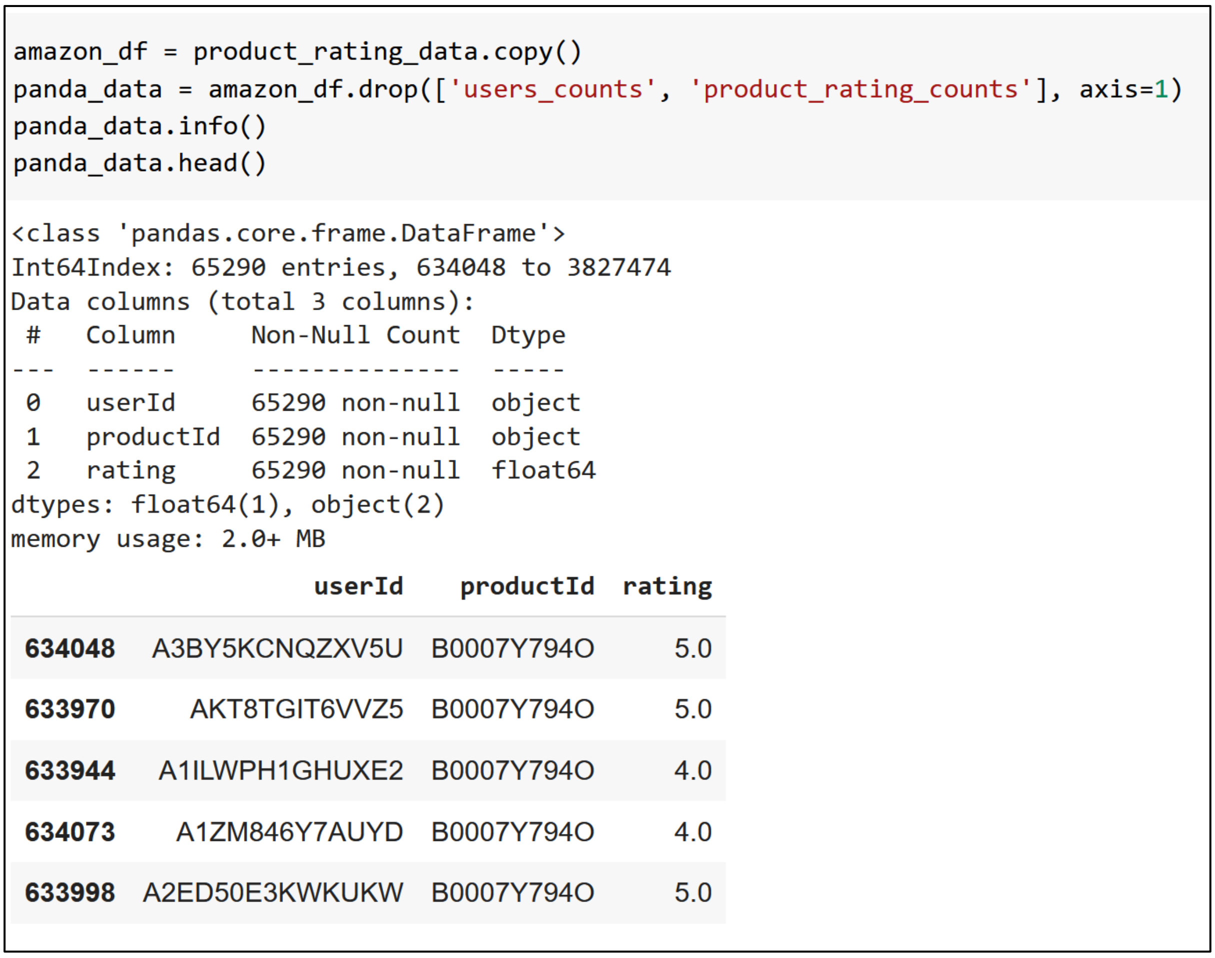
```python
users_counts = df['userId'].value_counts().rename('users_counts')
users_data   = df.merge(users_counts.to_frame(),
                                left_on='userId',
                                right_index=True)
subset_df = users_data[users_data.users_counts >= 50]
subset_df.head()
```

	userId	productId	rating	users_counts
94	A3BY5KCNQZXV5U	0594451647	5.0	50
14863	A3BY5KCNQZXV5U	B00000JD4V	4.0	50
134213	A3BY5KCNQZXV5U	B000063574	5.0	50
338368	A3BY5KCNQZXV5U	B0000CDJP8	5.0	50
634048	A3BY5KCNQZXV5U	B0007Y794O	5.0	50



```python
product_rating_counts = subset_df['productId'].value_counts().rename('product_rating_counts')
product_rating_data   = subset_df.merge(product_rating_counts.to_frame(),
                                left_on='productId',
                                right_index=True)
product_rating_data = product_rating_data[product_rating_data.product_rating_counts >= 5]
product_rating_data.head()
```

	userId	productId	rating	users_counts	product_rating_counts
634048	A3BY5KCNQZXV5U	B0007Y794O	5.0	50	18
633970	AKT8TGIT6VVZ5	B0007Y794O	5.0	192	18
633944	A1ILWPH1GHUXE2	B0007Y794O	4.0	98	18
634073	A1ZM846Y7AUYD	B0007Y794O	4.0	77	18
633998	A2ED50E3KWKUKW	B0007Y794O	5.0	65	18

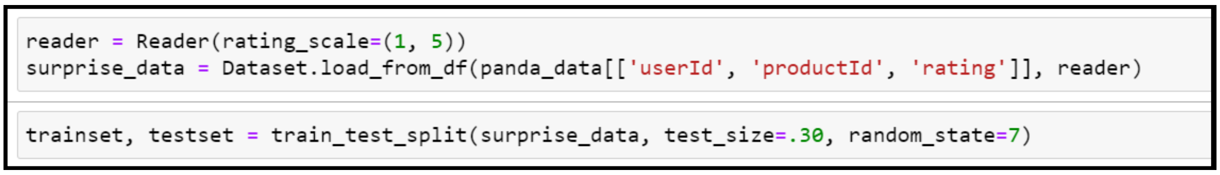


```
amazon_df = product_rating_data.copy()
panda_data = amazon_df.drop(['users_counts', 'product_rating_counts'], axis=1)
panda_data.info()
panda_data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 65290 entries, 634048 to 3827474
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   userId     65290 non-null  object 
 1   productId  65290 non-null  object 
 2   rating     65290 non-null  float64
dtypes: float64(1), object(2)
memory usage: 2.0+ MB
```

	userId	productId	rating
634048	A3BY5KCNQZXV5U	B0007Y794O	5.0
633970	AKT8TGIT6VVZ5	B0007Y794O	5.0
633944	A1ILWPH1GHUXE2	B0007Y794O	4.0
634073	A1ZM846Y7AUYD	B0007Y794O	4.0
633998	A2ED50E3KWKUKW	B0007Y794O	5.0

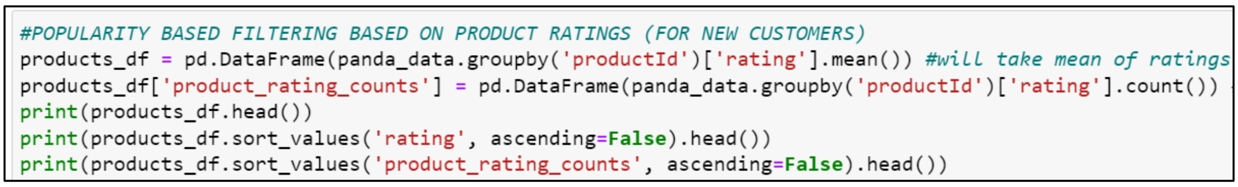
We split the dataset into train and test sets.



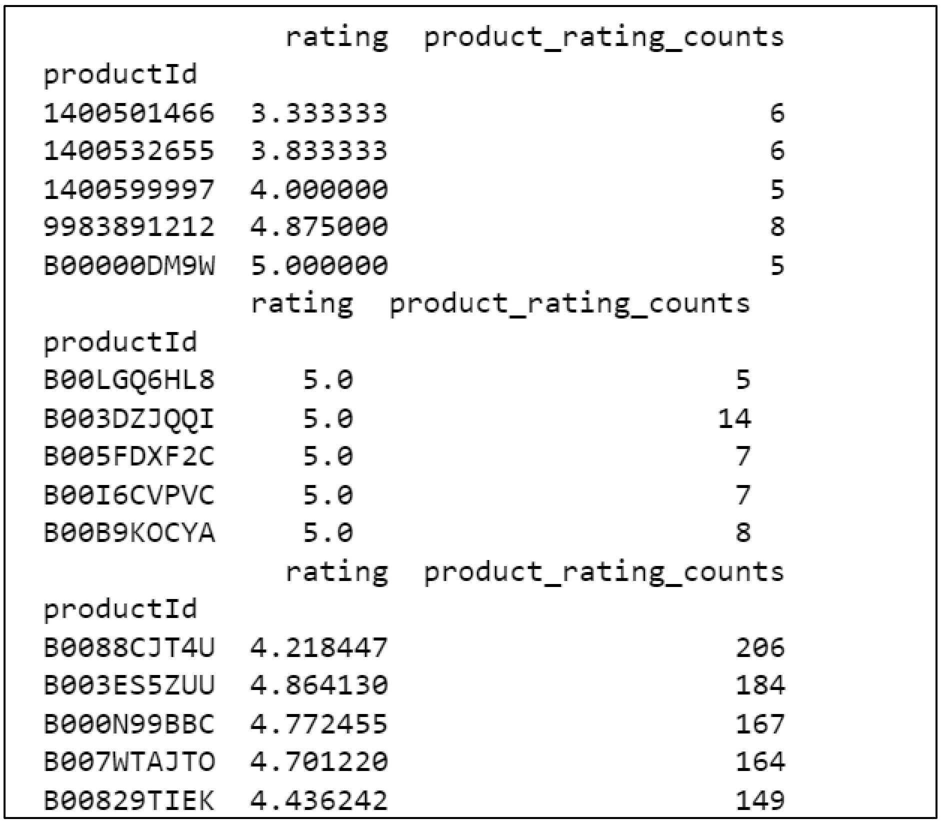
```
reader = Reader(rating_scale=(1, 5))
surprise_data = Dataset.load_from_df(panda_data[['userId', 'productId', 'rating']], reader)
```

```
trainset, testset = train_test_split(surprise_data, test_size=.30, random_state=7)
```

In order to overcome cold-start problem, we developed a popularity-based filtering system that recommends the most frequently purchased items or items with high ratings.

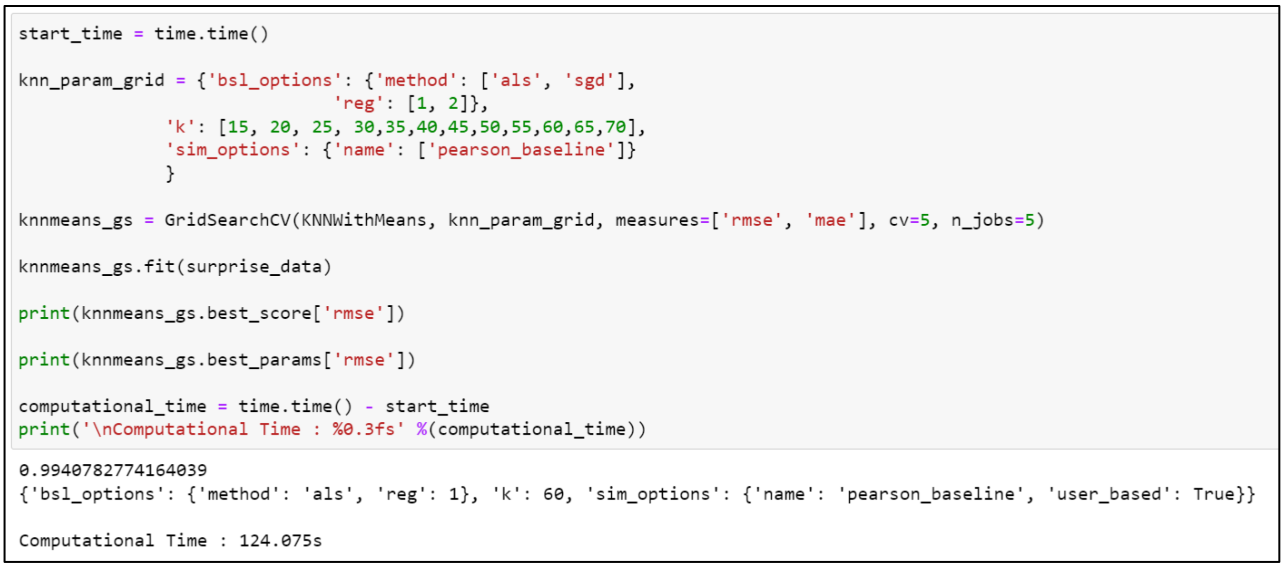


```
#POPULARITY BASED FILTERING BASED ON PRODUCT RATINGS (FOR NEW CUSTOMERS)
products_df = pd.DataFrame(panda_data.groupby('productId')['rating'].mean()) #will take mean of ratings
products_df['product_rating_counts'] = pd.DataFrame(panda_data.groupby('productId')['rating'].count())
print(products_df.head())
print(products_df.sort_values('rating', ascending=False).head())
print(products_df.sort_values('product_rating_counts', ascending=False).head())
```



```
                rating  product_rating_counts
productId
1400501466  3.333333                      6
1400532655  3.833333                      6
1400599997  4.000000                      5
9983891212  4.875000                      8
B00000DM9W  5.000000                      5
            rating  product_rating_counts
productId
B00LGQ6HL8     5.0                      5
B003DZJQQI     5.0                     14
B005FDXF2C     5.0                      7
B00I6CVPVC     5.0                      7
B00B9KOCYA     5.0                      8
                rating  product_rating_counts
productId
B0088CJT4U  4.218447                    206
B003ES5ZUU  4.864130                    184
B000N99BBC  4.772455                    167
B007WTAJTO  4.701220                    164
B00829TIEK  4.436242                    149
```

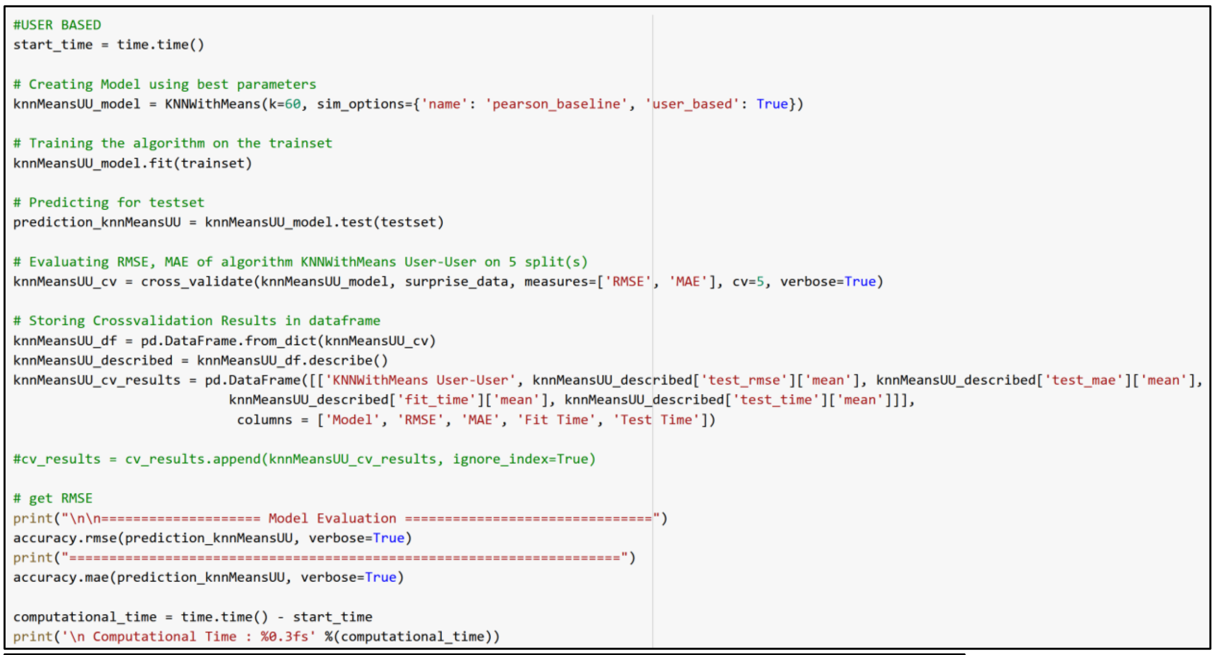
We selected the best values for hyperparameters such as number of clusters, similarity measures and baseline options.



```
start_time = time.time()

knn_param_grid = {'bsl_options': {'method': ['als', 'sgd'],
                              'reg': [1, 2]},
              'k': [15, 20, 25, 30,35,40,45,50,55,60,65,70],
              'sim_options': {'name': ['pearson_baseline']}
              }

knnmeans_gs = GridSearchCV(KNNWithMeans, knn_param_grid, measures=['rmse', 'mae'], cv=5, n_jobs=5)

knnmeans_gs.fit(surprise_data)

print(knnmeans_gs.best_score['rmse'])

print(knnmeans_gs.best_params['rmse'])

computational_time = time.time() - start_time
print('\nComputational Time : %0.3fs' %(computational_time))
```

```
0.9940782774164039
{'bsl_options': {'method': 'als', 'reg': 1}, 'k': 60, 'sim_options': {'name': 'pearson_baseline', 'user_based': True}}

Computational Time : 124.075s
```

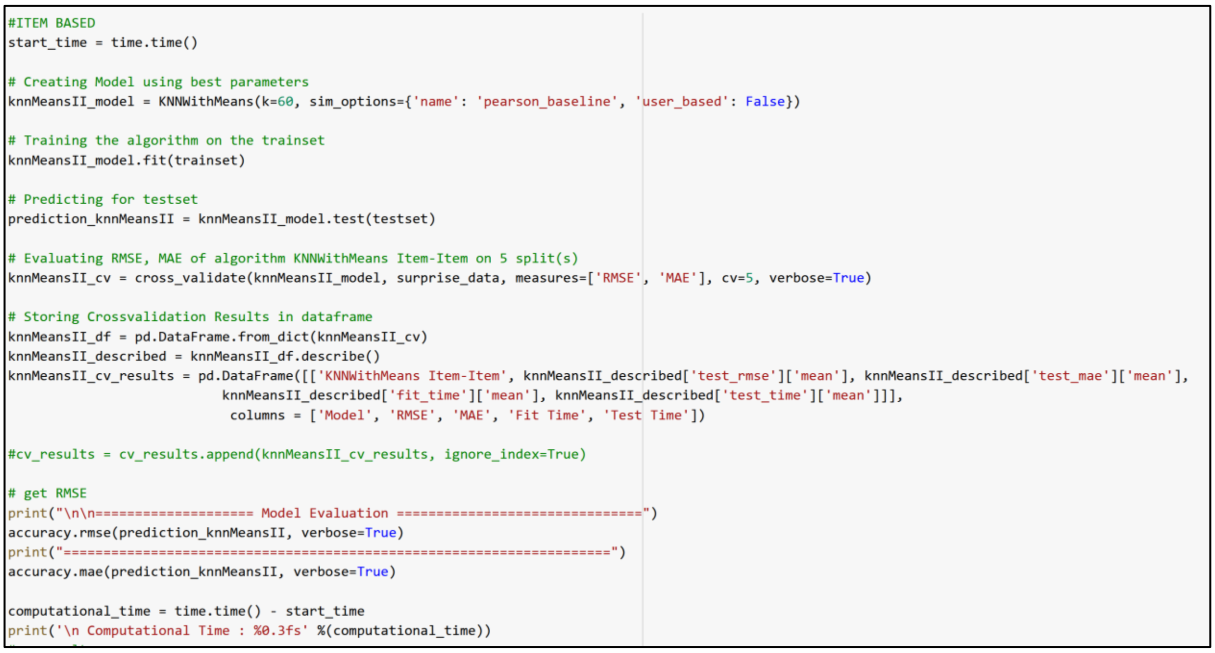
User-Based Collaborative Filtering



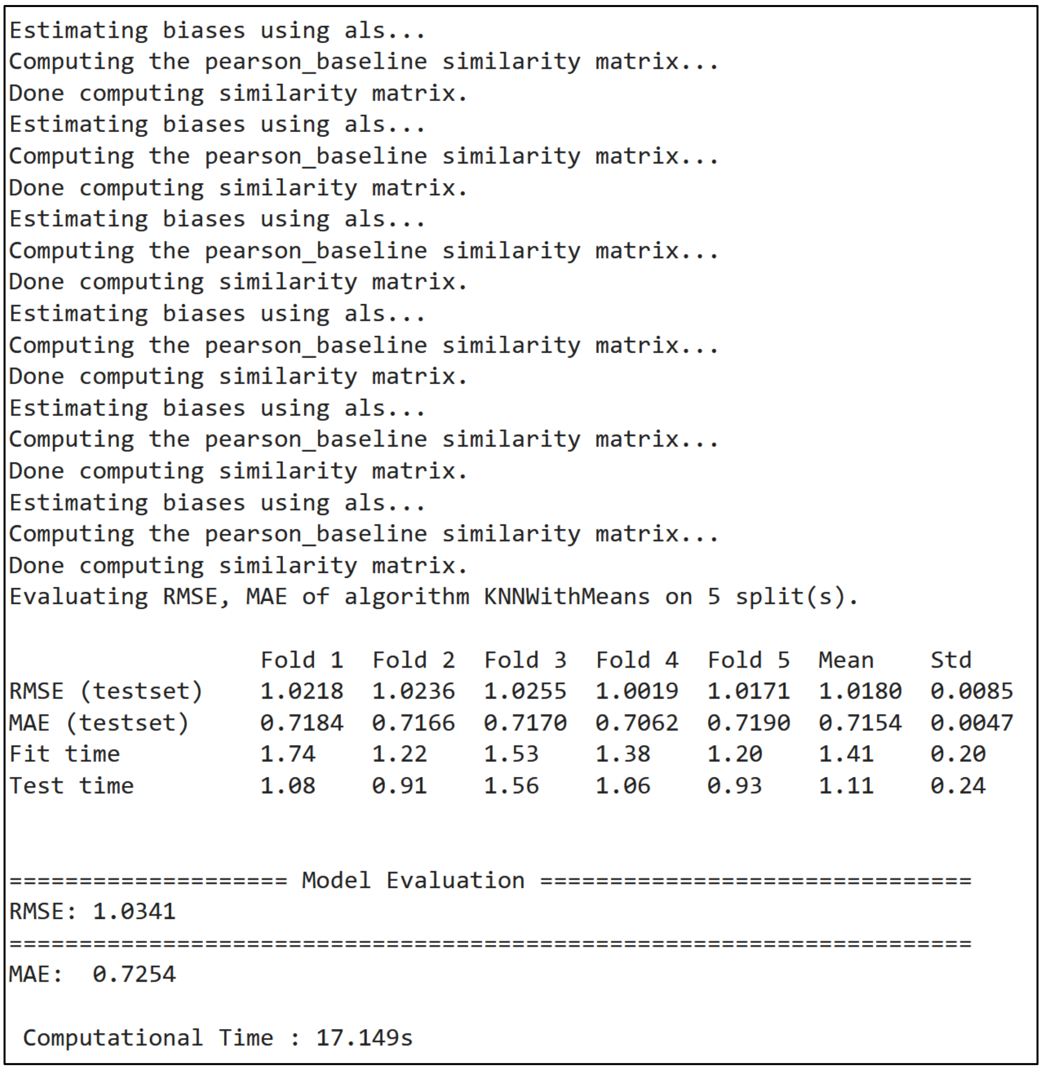
```python
#USER BASED
start_time = time.time()

# Creating Model using best parameters
knnMeansUU_model = KNNWithMeans(k=60, sim_options={'name': 'pearson_baseline', 'user_based': True})

# Training the algorithm on the trainset
knnMeansUU_model.fit(trainset)

# Predicting for testset
prediction_knnMeansUU = knnMeansUU_model.test(testset)

# Evaluating RMSE, MAE of algorithm KNNWithMeans User-User on 5 split(s)
knnMeansUU_cv = cross_validate(knnMeansUU_model, surprise_data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Storing Crossvalidation Results in dataframe
knnMeansUU_df = pd.DataFrame.from_dict(knnMeansUU_cv)
knnMeansUU_described = knnMeansUU_df.describe()
knnMeansUU_cv_results = pd.DataFrame([['KNNWithMeans User-User', knnMeansUU_described['test_rmse']['mean'], knnMeansUU_described['test_mae']['mean'],
                            knnMeansUU_described['fit_time']['mean'], knnMeansUU_described['test_time']['mean']]],
                             columns = ['Model', 'RMSE', 'MAE', 'Fit Time', 'Test Time'])

#cv_results = cv_results.append(knnMeansUU_cv_results, ignore_index=True)

# get RMSE
print("\n\n==================== Model Evaluation ===============================")
accuracy.rmse(prediction_knnMeansUU, verbose=True)
print("=====================================================================")
accuracy.mae(prediction_knnMeansUU, verbose=True)

computational_time = time.time() - start_time
print('\n Computational Time : %0.3fs' %(computational_time))
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9912  0.9947  1.0037  0.9940  0.9851  0.9937  0.0060
MAE (testset)     0.6965  0.6987  0.7099  0.7046  0.6939  0.7007  0.0058
Fit time          0.34    0.37    0.38    0.40    0.37    0.37    0.02
Test time         0.40    0.39    0.39    0.42    0.38    0.40    0.01


==================== Model Evaluation ===============================
RMSE: 1.0046
=====================================================================
MAE:  0.7117

 Computational Time : 5.823s
```

Item-Based Collaborative Filtering

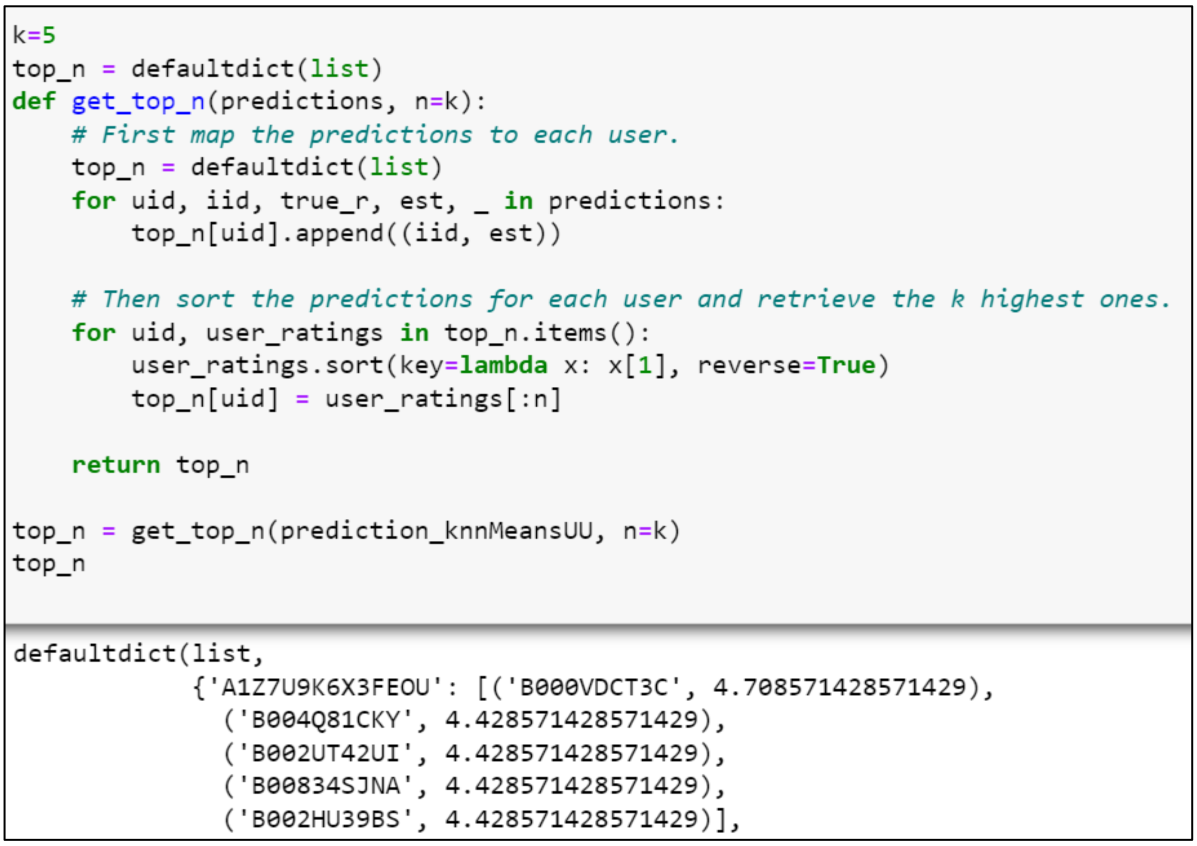


```
#ITEM BASED
start_time = time.time()

# Creating Model using best parameters
knnMeansII_model = KNNWithMeans(k=60, sim_options={'name': 'pearson_baseline', 'user_based': False})

# Training the algorithm on the trainset
knnMeansII_model.fit(trainset)

# Predicting for testset
prediction_knnMeansII = knnMeansII_model.test(testset)

# Evaluating RMSE, MAE of algorithm KNNWithMeans Item-Item on 5 split(s)
knnMeansII_cv = cross_validate(knnMeansII_model, surprise_data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Storing Crossvalidation Results in dataframe
knnMeansII_df = pd.DataFrame.from_dict(knnMeansII_cv)
knnMeansII_described = knnMeansII_df.describe()
knnMeansII_cv_results = pd.DataFrame([['KNNWithMeans Item-Item', knnMeansII_described['test_rmse']['mean'], knnMeansII_described['test_mae']['mean'],
                           knnMeansII_described['fit_time']['mean'], knnMeansII_described['test_time']['mean']]],
                            columns = ['Model', 'RMSE', 'MAE', 'Fit Time', 'Test Time'])

#cv_results = cv_results.append(knnMeansII_cv_results, ignore_index=True)

# get RMSE
print("\n\n=================== Model Evaluation ==============================")
accuracy.rmse(prediction_knnMeansII, verbose=True)
print("====================================================================")
accuracy.mae(prediction_knnMeansII, verbose=True)

computational_time = time.time() - start_time
print('\n Computational Time : %0.3fs' %(computational_time))
```



```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNWithMeans on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std     
RMSE (testset)    1.0218  1.0236  1.0255  1.0019  1.0171  1.0180  0.0085  
MAE (testset)     0.7184  0.7166  0.7170  0.7062  0.7190  0.7154  0.0047  
Fit time          1.74    1.22    1.53    1.38    1.20    1.41    0.20    
Test time         1.08    0.91    1.56    1.06    0.93    1.11    0.24    


=================== Model Evaluation ==============================
RMSE: 1.0341
====================================================================
MAE:  0.7254

 Computational Time : 17.149s
```

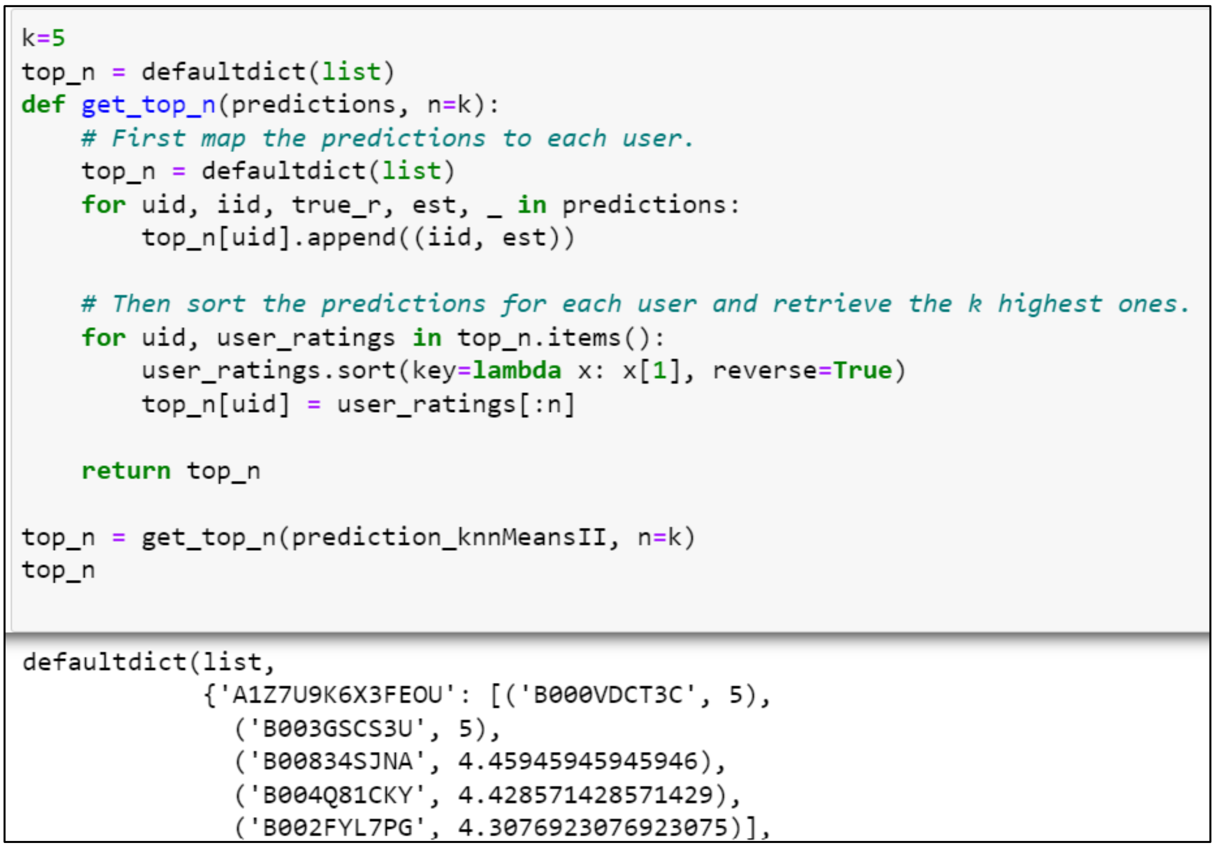
Recommending top five products in user-based collaborative filtering



```
k=5
top_n = defaultdict(list)
def get_top_n(predictions, n=k):
    # First map the predictions to each user.
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the k highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n

top_n = get_top_n(prediction_knnMeansUU, n=k)
top_n
```

```
defaultdict(list,
            {'A1Z7U9K6X3FEOU': [('B000VDCT3C', 4.708571428571429),
              ('B004Q81CKY', 4.428571428571429),
              ('B002UT42UI', 4.428571428571429),
              ('B00834SJNA', 4.428571428571429),
              ('B002HU39BS', 4.428571428571429)],
```

Recommending top five products in item-based collaborative filtering



```
k=5
top_n = defaultdict(list)
def get_top_n(predictions, n=k):
    # First map the predictions to each user.
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the k highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n

top_n = get_top_n(prediction_knnMeansII, n=k)
top_n
```

```
defaultdict(list,
            {'A1Z7U9K6X3FEOU': [('B000VDCT3C', 5),
              ('B003GSCS3U', 5),
              ('B00834SJNA', 4.45945945945946),
              ('B004Q81CKY', 4.428571428571429),
              ('B002FYL7PG', 4.3076923076923075)],
```

PERFORMANCE MEASURES

The performance measures we used to evaluate the recommender systems are the Root Mean Square Error (RMSE) and Mean Absolute Error (MAE).

Root Mean Square Error is one of the standard ways to measure the error of a model in predicting quantitative data. Formally, it is defined as follows:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

$$\begin{aligned} \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n &\text{ are predicted values} \\ y_1, y_2, \ldots, y_n &\text{ are observed values} \\ n &\text{ is the number of observations} \end{aligned}$$

Mean Absolute Error is the difference between the actual value and the predicted value.

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

where y_i - a target value, $\hat{y}_i$ - a predicted value.

The difference between RMSE and MAE is that RMSE penalizes the term when the error is high, and RMSE is always greater than MAE.

Method	RMSE	MAE
User Based	1.0046	0.7117
Item Based	1.0341	0.7254

From the results, it can be concluded that the User-based recommender system yields more accurate results than the Item-based recommender system.

CONCLUSION

In this project, we have implemented an e-commerce product recommendation system based on user-based and item-based techniques. The recommender systems are trained using the Amazon Electronic Products Rating dataset. The KNN algorithm is used with Pearson's Correlation Coefficient to make recommendations to the user by predicting the ratings of that user for all products and then recommending the top five products. The cold-start problem is resolved by recommending the most frequently purchased items or items with high ratings. The performance of both the systems is compared using RMSE and MAE metrics, and it is found that the user-based recommender produces better results than the item-based recommender.

REFERENCES

https://scholarworks.calstate.edu/downloads/1n79h8686