CSE4077- Recommender Systems

J Component – Project Report

Recommendation based on Amazon food Review

By

19MIA1065	B. Phanindra Sai
19MIA1071	O. Naga Sai Kumar
19MIA1086	T. Siva Nikhil
19MIA1032	M. Jay Kumar Patel

MTech CSE with Specialization Business Analytics

Submitted to

Dr. A. Bhuvaneswari,
Assistant Professor Senior,
SCOPE, VIT, Chennai

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report entitled "Face Generation with Deep Convolutional GANs" is a Bonafide work of B. Phanindra Sai (19MIA1065), O. Naga Sai Kumar (19MIA1071), M. Jay Kumar Patel (19MIA1032), T. Siva Nikhil (19MIA1086) who carried out the J-component under my supervision and guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. A. Bhuvaneswari,

Assistant Professor Senior,

SCOPE, VIT, Chennai

ABSTRACT

Amazon sells lots of products worldwide and it plays a vital role in our life. now we are analysing more on their food products. Considering that everyone has different purchase profile, a recommendation system is required to help and give a personalized suggestion products based on the user's preferences. In recent years, consumer interest in shopping online is increased globally with a focus on home delivery. We have data filled with reviews and the ingredients of food. We are trying out content based, popularity based, collaborative based filtering and SVM methods and we are finding out the best and their performances. We will be making the model using the reviews of the people who purchased past and their reviews.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. A. Bhuvaneswari Assistant** Professor, School of Computer Science Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We express our thanks to our HOD ___________________ for her support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

B. Phanindra Sai – 19MIA1065

O. Naga Sai Kumar – 19MIA1071

M. Jay Kumar Patel – 19MIA1032

T. Siva Nikhil – 19MIA1086

August 2022

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computing Science and Engineering

VIT Chennai

Vandalur - Kelambakkam Road, Chennai - 600 127

FALL SEM 22-23

Worklet details

Programme	M.Tech with Specialization Business Analytics	
Course Name / Code	Recommender system/CSE4077	
Slot	E1+TE1	
Faculty Name	Dr. A. BHUVANESWARI	
Component	J – Component	
J Component Title	Recommendation based on Amazon Food review	
Team Members Name | Reg. No	19MIA1065	B. Phanindra Sai
	19MIA1071	O. Naga Sai Kumar
	19MIA1086	T. Siva Nikhil
	19MIA1032	M. Jay Kumar Patel

Team Members(s) Contributions – Tentatively planned for implementation:

Worklet Tasks	*Contributor's Names*
Data collection & literature survey	Sai Kumar, Jay Kumar Patel, Siva Nikhil, Phanindra Sai
Pre processing	Jay Kumar Patel, Phanindra Sai
Model building	Phanindra Sai, jay Kumar Patel, Siva Nikhil, Sai Kumar
Visualization	Siva Nikhil, Sai Kumar
Technical Report writing	Sai Kumar, Siva Nikhil
Presentation preparation	Phanindra Sai, Jay Kumar Patel

TABLE OF CONTENTS

1. Introduction

Almost all e-commerce websites allow users to rate the products or services which they received when shopping. These feedbacks serve as suggestions to other users and are influential to people’s decisions on whether to buy the product.

Therefore, exploring and understanding the ratings has become an important way to understand the need of users. Moreover, applying the data to build a better recommendation system is an integral part of the success of a company.

Recommendation systems of Amazon brings more than 30% of revenues, and Netflix, where 75% of what people watch is from some sort of recommendation.

Based on the Amazon Data, we built a recommendation system for Amazon users. We implemented Matrix Factorization, SVD, Deep Learning, content based, popularity based, collaborative based filtering. We compared different methods and made a combination of some methods to provide a better recommendation. The problem we are going to solve is how to help users select products which they may like and to make recommendation to stimulate sales and increase profits.

Firstly, we decided to choose the Amazon Fine Food Reviews dataset which consists of 568,454 food reviews Amazon users left up to October 2012 as our dataset.

Secondly, our recommendation system is based on users rating prediction. We assume that users tend to like the products that have a score of greater than 4 and we will consider the highest 5 scores product as our recommendation candidates.

Thirdly, we implemented several algorithms to predict the scores of each product for each user. Distance Based Model Here we use the cosine-distance to give the similarity between vectors. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The similarity ranges from -1 to 1 where -1 means exactly opposite, 1 means exactly the same and in-between values indicating intermediate similarity or dissimilarity.

2. Literature Survey

Sl no	Title	Author / Journal name / Year	Technique	Result
1	Diet-Right: A Smart Food Recommendation System	Faisal Rehman Journal of researchgate, 2017	ACO, Cloud Computing	The result shows that the highest accuracy is achieved with 110 ants. It is quite evident that when we increase the number of ants, the accuracy is also increased. Moreover, it is observed that the accuracy remains constant between 80 to 100 ants.
2	Food recommender systems for diabetic patients: a narrative review	Somaye Norouzi, Mohsen Nematy. Journal of researchgate, 2017	CFRS, KBRS and CARS	Rule- based reasoning and semantic web such as food ontology and the combination of both were the most popular techniques applied to develop food recommender systems
3	A Personalized Food Recommender System for Zomato	Mansi Goel, Ayush Agarwal. Journal of arvix, 2019		Best performance (0.90 F-score) is obtained on manually-annotated ground-truth dataset.
4	Recommendation System for Grocery Store Considering Data Sparsity	NatsukiSanoa, NatsumiMachino Journal of sciencedirect, 2015	SVD-type recommendation based on real POS data	The F-value of the best recommendation method for product category recommendation is increased 5.24 times compared to the product item method.
5	Online Grocery Recommendation System	Suja Panicker	Slope-one and min hash algorithms	Total number of common elements=7 Total number of elements=12 So, 7/12=0.58

		Journal of researchgate, 2016		That means, similarity between User1 and User2 is 58%.
6	Amazon.com Recommendations Item-to-Item Collaborative Filtering	Greg Linden, Brent Smith, and Jeremy York Journal of UMD, 2003	Item-to-Item Collaborative Filtering, search-based model	A good recommendation algorithm is scalable over very large customer bases and product catalog, requires only subsecond processing time to generate online recommendations, is able to react immediately to changes in a user's data, and makes compelling recommendations for all users regardless of the number of purchases and ratings. Unlike other algorithms, item-to-item collaborative filtering is able to meet this challenge.
7	Amazon Food Review Classification Using Deep Learning and Recommender System	Z Zhou, L Xu Journal of stanford Systems, 2009	Feed-forward Neural Network, LSTM.	Model RMSE Popular(baseline) 1.7372 Collaborative Filtering 1.4538 Matrix Factorization 1.1198
8	Exploring hidden factors behind online food shopping from Amazon reviews: A topic mining approach	Yan Heng, Zhifeng Gao, Yuan Jiang, Xuqi Chen Journal of Retailing and	Latent Dirichlet Allocation, Regression analysis,	The online customer review system provides a platform for consumers to share their opinions and experiences, and for potential consumers to obtain information about products that interest them. Customer reviews, especially those with high helpfulness votes, would not only affect

		Consumer Services, 2018		purchase decisions of other customers, but also influence their perceptions and knowledge about food products. The category of food products is a unique experience good, with perceived quality being highly subjective and heterogeneous. Therefore, how customer review readers obtain desirable information on food products becomes intriguing important.
9	Sentimental Analysis on Amazon Fine Food Reviews	Kartikay Thakkar, Sidharth Sharma, Ujjwal Chhabra International Journal of Scientific Research & Engineering Trends, 2020	Random Forest, KNN, Logistic Regression, Navie bayes, SVM Decision Tree	Supervised approach helps to obtain the results. Logistic Regression, naïve Bayes gave the best accuracies in machine learning models. whereas LSTM ran the battle since it’s a deep learning model and gave an accuracy of 92.1%. There were sentiments of all sorts but the overwhelming majority had a positive sentiment.
10	Food recommender system on Amazon	Tao Huang, Huan Zhou, Kai Zhou San Diego: University of California San Diego., 2017	Linear Regression, Latent Factor Model, Bias-SVD	Latent Factor Model is very suitable to predict ratings for experienced users. The reason lies in that experienced users give us more information about their preferences as well as their implicit feedback.

3. Dataset and Tools

https://www.kaggle.com/datasets/snap/amazonfine-food-reviews

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience.

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems.

4. Methodology

1.Dataset and Pre-processing

We will be building popularity-based recommendation system and collaborative based recommendation system and

2.we show that almost all users bought less than 60 times and almost all products have been purchased less than 200 times

4.Distance Based Model In this part, we use the cosine-distance based model, and it is a little different from the traditional collaborative filtering. The traditional CF use either user similarity matrix or product similarity matrix to predict the whole user-product matrix. However, we found it was not applicable in this data set -- the MSE is way bigger than other methods. Then we tried to use a quasi-SVD approach. That is, the output matrix is the dot product of three matrix:

U is user similarity matrix and P is product similarity. And we use this algorithm to update the target matrix, the user-product matrix. Our train set MSE is 2.0860 and our test set MSE is 2.1480. The advantage is that this method is quite interpretable. However, the drawbacks are also apparent: this algorithm is naive with a deficient performance. For a large data set, this algorithm need m2+n2 times to calculate the similarity matrix.

5.Matrix Factorization Latent factor models are an alternative approach that tries to explain the ratings by characterizing both items and users on, say, 20 to 100 factors inferred from the ratings patterns. For products, the discovered factors might measure obvious dimensions such as candy vs drinks, or adult food vs children’s; For users, each factor measures how much the user likes the product that score high on the corresponding movie factor. Using latent factor model, we transform the way to calculate the similarity of users and products. The features become more stable and condense. we first create two new metrics, user-latent_factor and product-latent_factor. The size are qf and pf, p, q are the total number of users and products, f is the number of latent factors. So, every element in the target matrix can be calculate as: And the target matrix is shown as . Next, we need to create the objective function based on least-square method to minimize the loss:

The system adjusts the model by fitting the previously observed ratings. However, the goal is to generalize those previous ratings in a way that predicts the unknown ratings. Thus, the system should avoid overfitting the observed data by regularizing the learned parameters by adding L2 term. The constant λ controls the extent of regularization and is usually determined by cross-validation. Next, we use stochastic gradient descent to optimize the objective function. The processing is:

Where gamma is the stochastic learning rate and e is the error term. In the iteration, when the change in loss is larger than 0, the learning rate increases by 5%; if delta-loss is smaller than 0, it means the new loss is becoming larger. The learning rate decreases by 50% so that the loss can converge. The MF gives a very good prediction: the train set MSE is 0.03542 and the test set MSE is 0.03385. While both MSEs are equally small, thus no overfitting problem occurs. In addition, as Fig 3 shows that the number of latent factors significantly affects the results.

Latent factors and MSE As the number of latent factors increase, the MSE decreases. And MSE converges around n=20.

6.Probabilistic Matrix Factorization PMF is similar to MF. It is nothing more than a MF assuming the distribution of user and production are Gaussian. That is:

And the joint distribution of user and product is the distribution of the scores.

The problem comes to maximum the probability of U, V based on R and variance. This transformation is based on Bayesian formula. Rewrite the function above and applying log to both sides:

When optimizing this function, the standard errors are fixed, so the objective function is:

Gradient descent process is the same with basic MF. PMF does better than MF for sparse matrices. The assumption of Gaussian makes it more accurate to predict. But for our dataset, the chosen data is not sparse, so the performance of PMF is almost the same with MF. the in sample MSE is 0.036697 and the out sample MSE is 0.030156. 4.2.3 SVD In our recommendation system, we have such a matrix which has many scores from the users to the items. We hope to predict the targeted users' score to other unevaluated items and then recommend the items with the highest five scores. The advantage of SVD is that: users' score matrix is a sparse matrix, so we can map the original data into a Low-dimensional space and then calculate the similarity of different items. This can help us reduce calculation complexity.

7.Time Series After our exploratory data analysis, we found out that the ratings are related to the time: the latter, the better.

Score distribution based on Time Besides, the popularity of a product is naturally linked to different time periods. iPhone 4 was a 4.8/5 product in 2010 but it will be inappropriate to recommend the same product in 2013. We decided to convert different ProductId and UserId into dummy variables, and ran RandomForestRegressor on ProductId, UserId and Time. We consider time series forecasting score as a threshold: if it is below 3, we will drop the candidate.

8.Recommendation After comparing different methods, we decided to use Matrix Factorization as our basic algorithm to predict for a certain product for a certain customer. We select the top 5 scores as our recommendation candidates. Then we introduced the time series method to determine if the score is below 3 or not. If yes, we will drop this candidate. Otherwise, we keep the original five. Figure 7

Recommendation Pipeline 5 ELVALUATION After using the different model, we will use the MSE and the confusion matrix to evaluate them. The performance result can be summarized as follows:

Then we compared different models on Confusion Matrix:

Confusion Matrix Here we define articles of score > 4 as our recommendation candidates.

5. Algorithms/Techniques Description

1.Distance Based Model

Here we use the cosine-distance to give the similarity between vectors. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The similarity ranges from -1 to 1 where -1 means exactly opposite, 1 means exactly the same and inbetween values indicating intermediate similarity or dissimilarity.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

Since the MSE of traditional Collaborative Filtering model is quite high, we decided to use quasi-SVD approach instead. In this method, the output matrix is the dot product of three matrix. Here U is user similarity matrix and P is product similarity.

$$\Sigma_{m*n} = U_{m*m} \cdot \Sigma_{m*n} \cdot P_{n*n}$$

2.Matrix Factorization Matrix

Factorization is another way to decompose the user-product matrix. MF models map both users and products to a joint latent factor space of dimensionality f (f refers to the number of the latent factors), such that user-product interactions are modelled as inner products in that space. In this report, we create two new matrices “user-latent factor” (size q*f, q refers to number of users) and “product-latent factor” (size p*f, p refers to the number of products). Element in target matrix can be calculated as $r_{ui} = q_i^T p_u$. We choose Least-square method to minimize the loss. Constant λ controls the extent of regularization.

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

3. Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

This process is sometimes called "feature bagging". These are models built from a training set for new points x' by looking at the "neighbourhood" of the point, formalized by a weight function W:

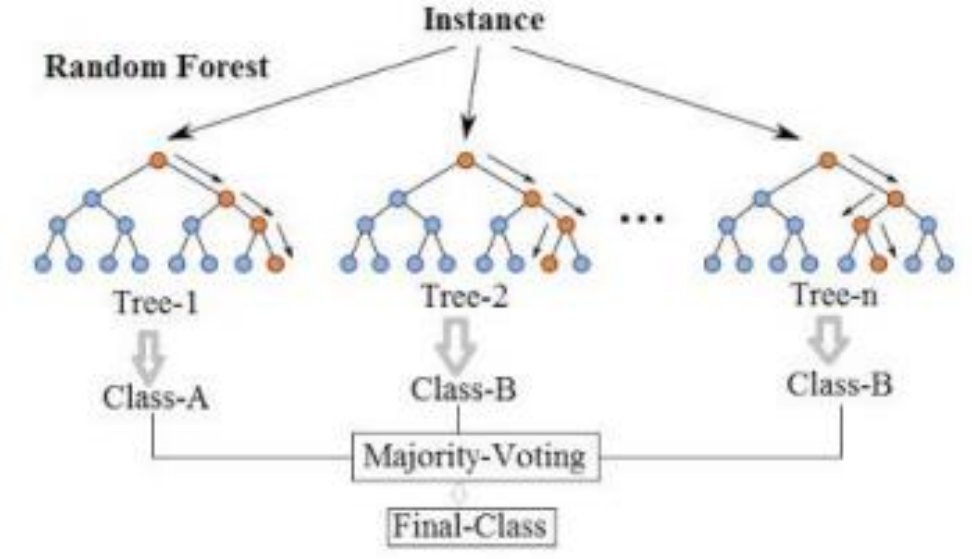
Since a forest averages the predictions of a set of m trees with individual weight functions Wj, its predictions are

$$\hat{y} = \sum_{i=1}^{n} W(x_i, x')\, y_i .$$

One of the advantages of random forests method is that it can avoid overfitting because of the law of large numbers.

$$\hat{y} = \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{n} W_j(x_i, x')\, y_i = \sum_{i=1}^{n}\left(\frac{1}{m}\sum_{j=1}^{m} W_j(x_i, x')\right)\, y_i .$$

Random Forest Simplified



As it is currently the state-of-art method for classification problem,we decided to implement it as a reference.Here, we chose “ProductId”, “UserId”, “Time”, “#Products” and “#Users” as features and “Score” as the classification labels and use API form Scikit-learn to fit a random forest models.

4.Probabilistic Matrix Factorization

In PMF, we assume that the appearance of user and product are compiled to Gaussian distribution as following.

$$p(U|\sigma_U^2) = \prod_{i=1}^{N} \mathcal{N}(U_i|0, \sigma_U^2\mathbf{I}), \qquad p(V|\sigma_V^2) = \prod_{j=1}^{M} \mathcal{N}(V_j|0, \sigma_V^2\mathbf{I}).$$

The distribution of Scores is the joint distribution of the former two variables.

$$p(R|U, V, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[\mathcal{N}(R_{ij}|U_i^T V_j, \sigma^2)\right]^{I_{ij}}$$

According to Bayesian formula, we can get the final objective function as following:

$$\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N} \parallel U_i \parallel^2 \quad + \frac{\lambda_V}{2}\sum_{j=1}^{M} \parallel V_j \parallel^2$$

5.SVD

We apply Singular value decomposition (SVD) algorithms to mapping the user-product matrix into low-dimensional space.Then we can get the predicted scores based on the similarity between unscored items and the others.

$$A_{m*n} = U_{m*m}\Sigma_{m*n}V_{n*n}^T \approx U_{m*k}\Sigma_{k*k}V_{k*n}^T$$

As for evaluation, we choose MSE (including train set and test set MSE) and Confusion Matrix (score >4 as recommend, score <4 as not recommend) to evaluate all the models above. Additionally, we introduce a time parameter with vote to the final model. Which means if the score predicted by the time parameter is below 3, we will not recommend it in the end.

We found out that matrix factorization models are superior to classic nearest-neighbour techniques for producing product recommendations and this are also proved by our research. Generally speaking, recommender systems are based on one of the two strategies: Content filtering and Collaborative filtering.

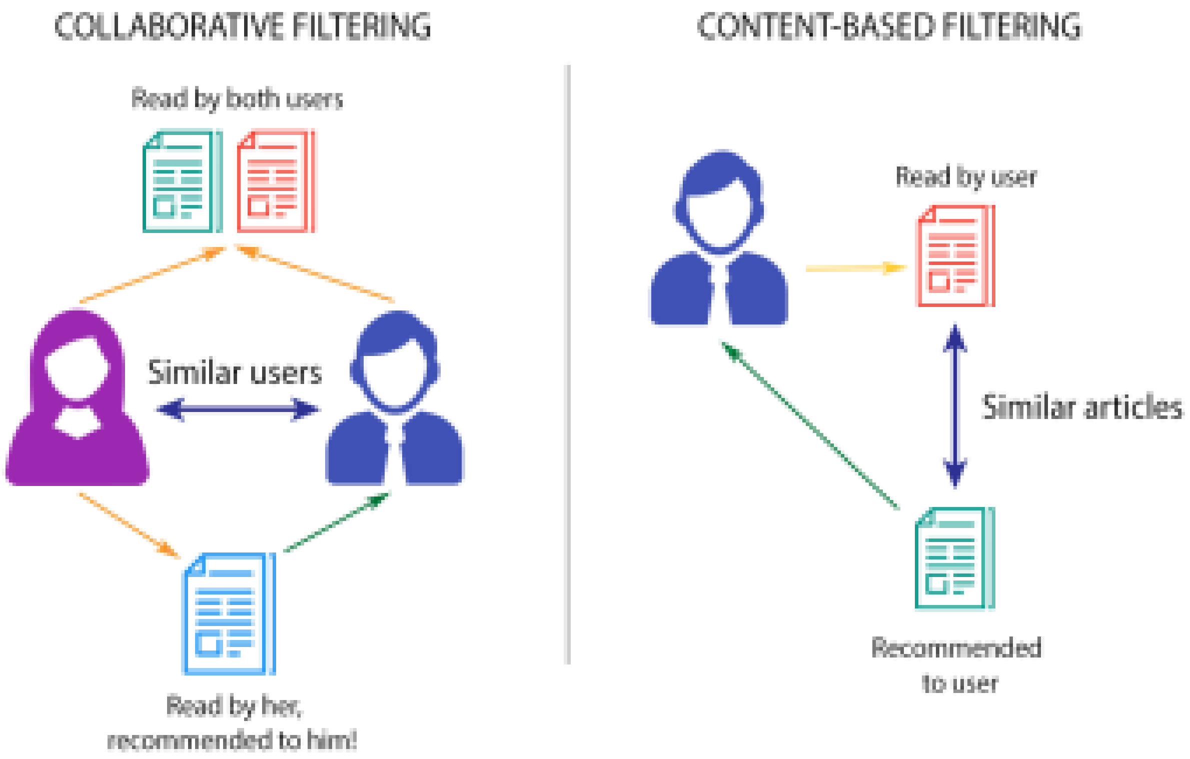


Fig 1 Content vs Collaborative Filtering

6.Content filtering

This approach creates a profile for each user or product to characterize its nature. For example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, etc. The profiles allow programs to associate users to products. The paper mentioned that a known successful realization of content filtering is the Music Genome Project, which is used for the internet radio service Pandora.com. A trained music analyst scores each song in the Music Genome Project based on hundreds of distinct musical characteristics. These attributes will contribute to recommend listeners' songs. However, based on the Amazon Fine Food Database, we don't have the profile information on products nor on users. Thus, we will focus on the second method.

7.Collaborative Filtering

The other algorithm relies on past user behaviour – for example, previous transactions or product ratings – without requiring the creation of explicit profiles. This is more suitable for our project. Collaborative filtering analyses the relationships between users and interdependencies among products to identify new user-item associations.

The two primary areas of collaborative filtering are the neighbourhood methods and latent factor models.

Neighbourhood methods are centred on computing and relationships between items or, alternatively, between users. This method can be categorized into two genres:

user-oriented neighbourhood method and product-oriented neighbourhood method.

User-oriented neighbourhood method: Joe likes the three movies A, B, and C.

To make a prediction for him, the system finds some similar users who also like those movies, and then determines which other movies they liked. Item-oriented collaborative filtering: this in essence just chain recommendations off a particular item.

The main advantage is that it can recommend items to users who we know nothing about except for what item they may be looking at currently.

Latent factor models are an alternative approach that tries to explain the rating by characterizing both items and users on, say, 20 to 100 factors inferred from the rating patterns.

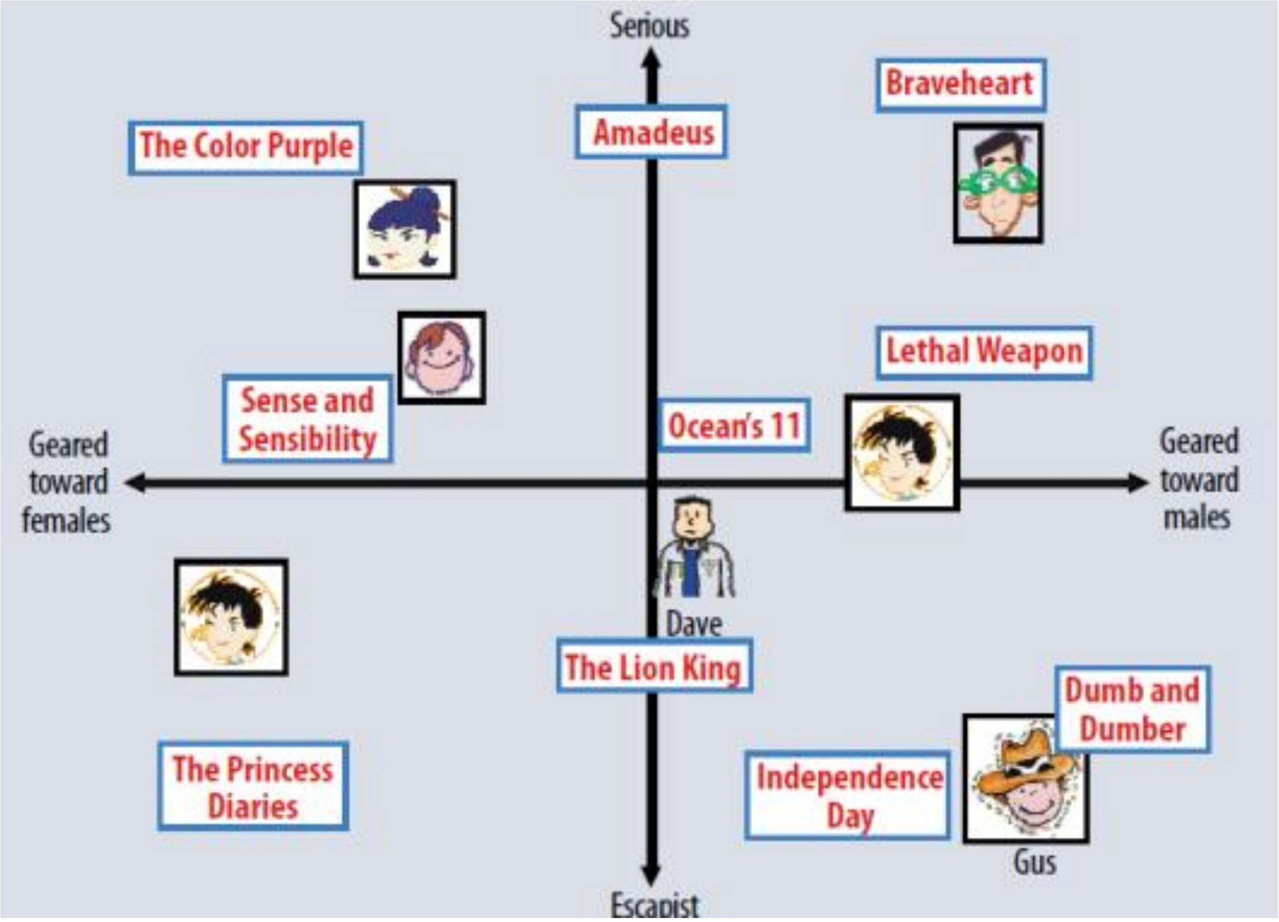


Fig 2 Latent factor approach

For our amazon fine food case, the discovered factors might measure obvious dimensions such as soft drinks, alcohol and food but also, less well-defined dimensions such as how tasty is a food; or completely uninterpretable dimensions. For users, each factor measures how much the user likes foods that score high on a certain type. Figure 2 (also from the paper) illustrates the idea for a simplified example in two dimensions.

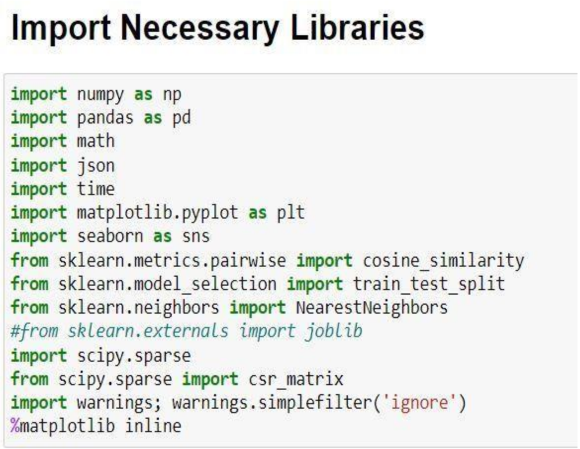
8.Matrix Factorization Methods

Some of the most successful realizations of latent factor models are based on matrix factorization. In its basic form, the matrix factorization characterizes both items and users by vectors of factors inferred from the rating patterns. We will discuss more about the Matrix Factorization Model in Part 4 Methodology.

9.Comments and Improvement

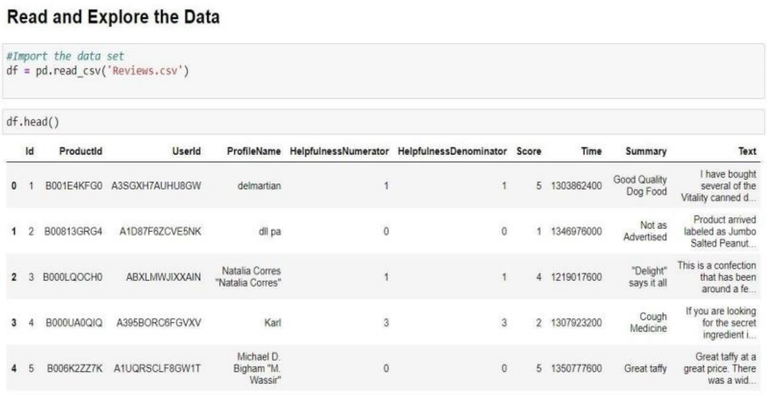
we referred to “Matrix Factorization Techniques for Recommender Systems” as our theoretical framework. We would like to verify if Collaborative Filtering based on Matrix Factorization methods is the state-of-art method for recommendation systems. Besides, we believe that it is not appropriate to omit the time factor. Whether a product is outdated is a key factor for a client to decide to purchase or not.

6. Implementation & Results



Import Necessary Libraries

```
import numpy as np
import pandas as pd
import math
import json
import time
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors
#from sklearn.externals import joblib
import scipy.sparse
from scipy.sparse import csr_matrix
import warnings; warnings.simplefilter('ignore')
%matplotlib inline
```

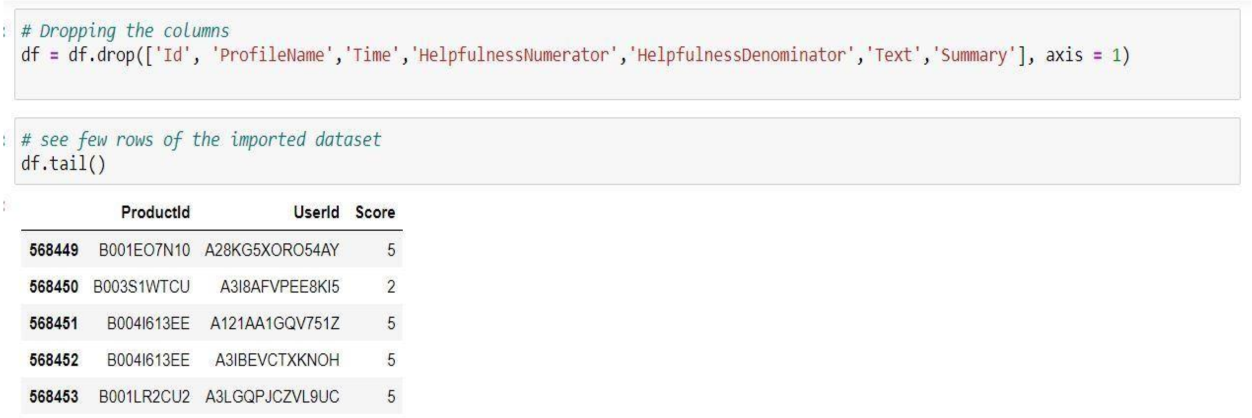


Read and Explore the Data

```
#Import the data set
df = pd.read_csv('Reviews.csv')
```

```
df.head()
```

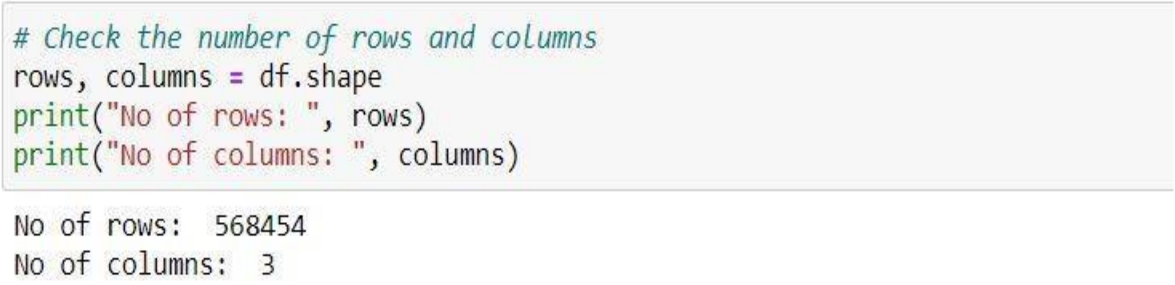
	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...



```
# Dropping the columns
df = df.drop(['Id', 'ProfileName','Time','HelpfulnessNumerator','HelpfulnessDenominator','Text','Summary'], axis = 1)
```

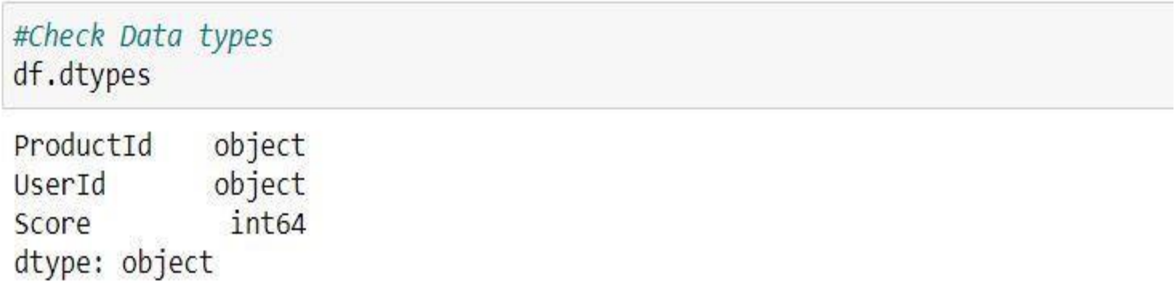
```
# see few rows of the imported dataset
df.tail()
```

	ProductId	UserId	Score
568449	B001EO7N10	A28KG5XORO54AY	5
568450	B003S1WTCU	A3I8AFVPEE8KI5	2
568451	B004I613EE	A121AA1GQV751Z	5
568452	B004I613EE	A3IBEVCTXKNOH	5
568453	B001LR2CU2	A3LGQPJCZVL9UC	5



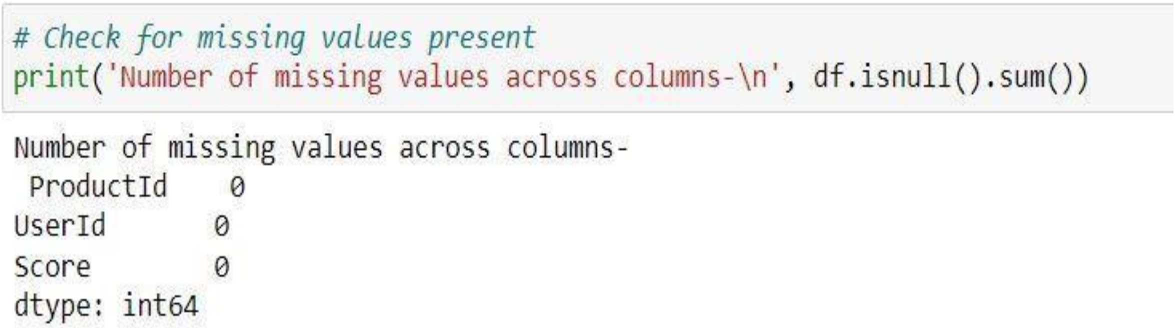
```
# Check the number of rows and columns
rows, columns = df.shape
print("No of rows: ", rows)
print("No of columns: ", columns)
```

```
No of rows:  568454
No of columns:  3
```



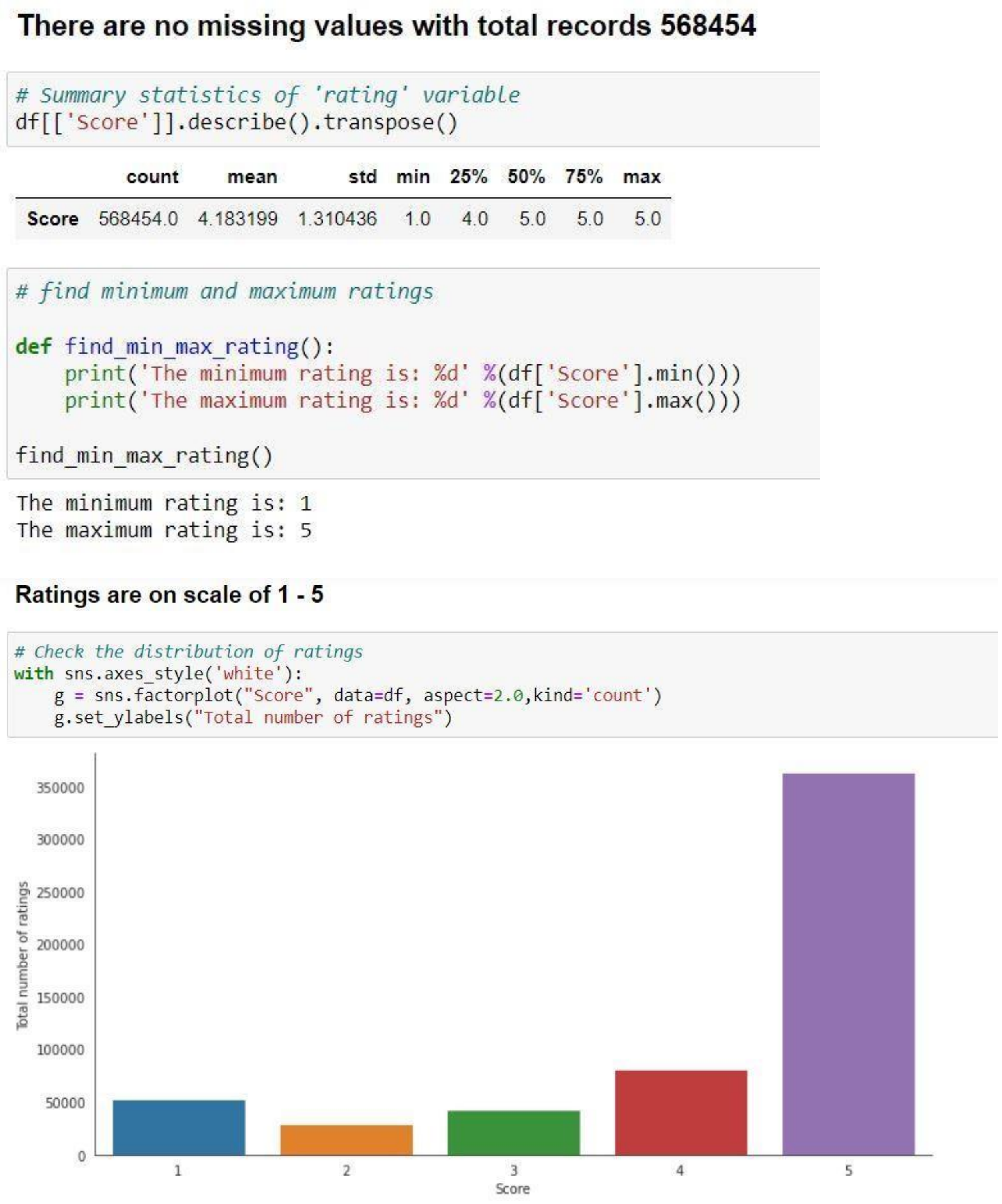
```
#Check Data types
df.dtypes
```

```
ProductId    object
UserId       object
Score         int64
dtype: object
```



```
# Check for missing values present
print('Number of missing values across columns-\n', df.isnull().sum())
```

```
Number of missing values across columns-
 ProductId    0
UserId       0
Score        0
dtype: int64
```



There are no missing values with total records 568454

```python
# Summary statistics of 'rating' variable
df[['Score']].describe().transpose()
```

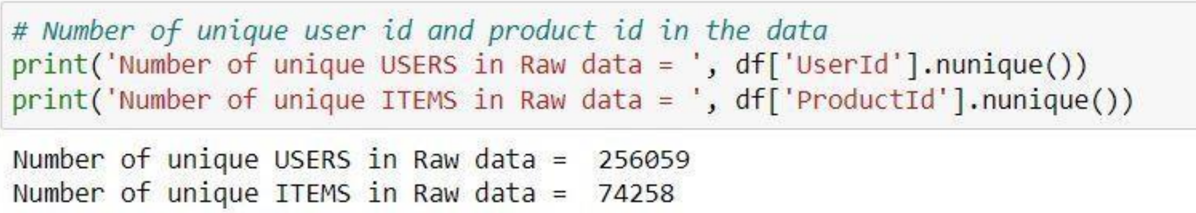
	count	mean	std	min	25%	50%	75%	max
Score	568454.0	4.183199	1.310436	1.0	4.0	5.0	5.0	5.0

```python
# find minimum and maximum ratings

def find_min_max_rating():
    print('The minimum rating is: %d' %(df['Score'].min()))
    print('The maximum rating is: %d' %(df['Score'].max()))

find_min_max_rating()
```

```
The minimum rating is: 1
The maximum rating is: 5
```

Ratings are on scale of 1 - 5

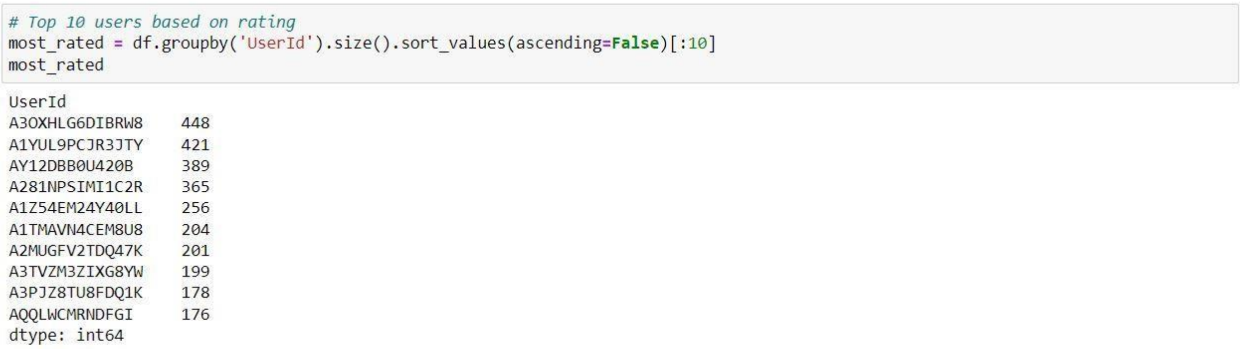
```python
# Check the distribution of ratings
with sns.axes_style('white'):
    g = sns.factorplot("Score", data=df, aspect=2.0,kind='count')
    g.set_ylabels("Total number of ratings")
```



```
# Number of unique user id and product id in the data
print('Number of unique USERS in Raw data = ', df['UserId'].nunique())
print('Number of unique ITEMS in Raw data = ', df['ProductId'].nunique())
```

```
Number of unique USERS in Raw data =  256059
Number of unique ITEMS in Raw data =  74258
```

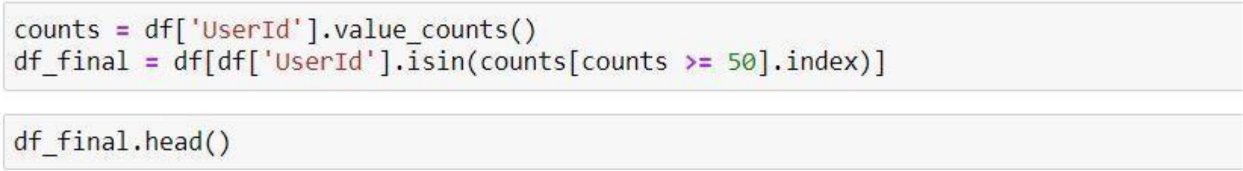
Take subset of dataset to make it less sparse/more dense. (For example, keep the users only who has given 50 or more number of ratings)



```
# Top 10 users based on rating
most_rated = df.groupby('UserId').size().sort_values(ascending=False)[:10]
most_rated
```

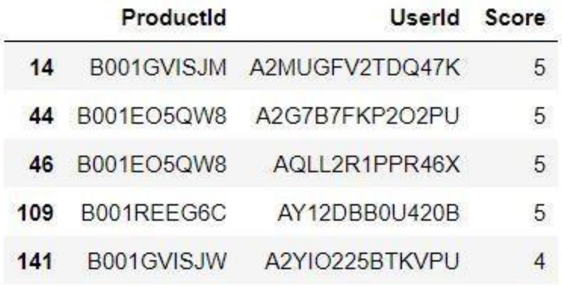
```
UserId
A3OXHLG6DIBRW8    448
A1YUL9PCJR3JTY    421
AY12DBB0U420B     389
A281NPSIMI1C2R    365
A1Z54EM24Y40LL    256
A1TMAVN4CEM8U8    204
A2MUGFV2TDQ47K    201
A3TVZM3ZIXG8YW    199
A3PJZ8TU8FDQ1K    178
AQQLWCMRNDFGI     176
dtype: int64
```

Data model preparation as per requirement on number of minimum ratings

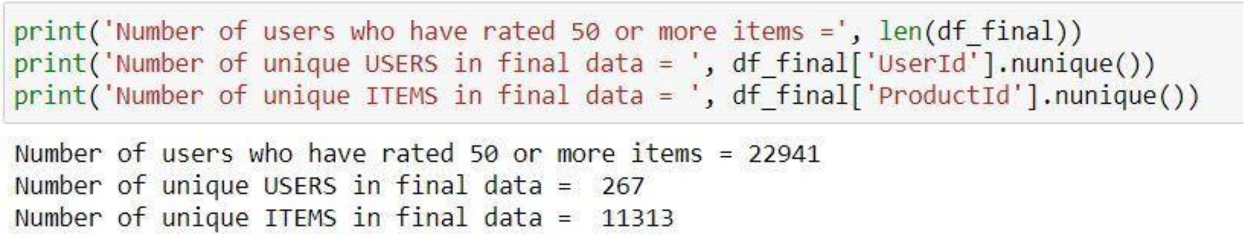


```
counts = df['UserId'].value_counts()
df_final = df[df['UserId'].isin(counts[counts >= 50].index)]
```

```
df_final.head()
```



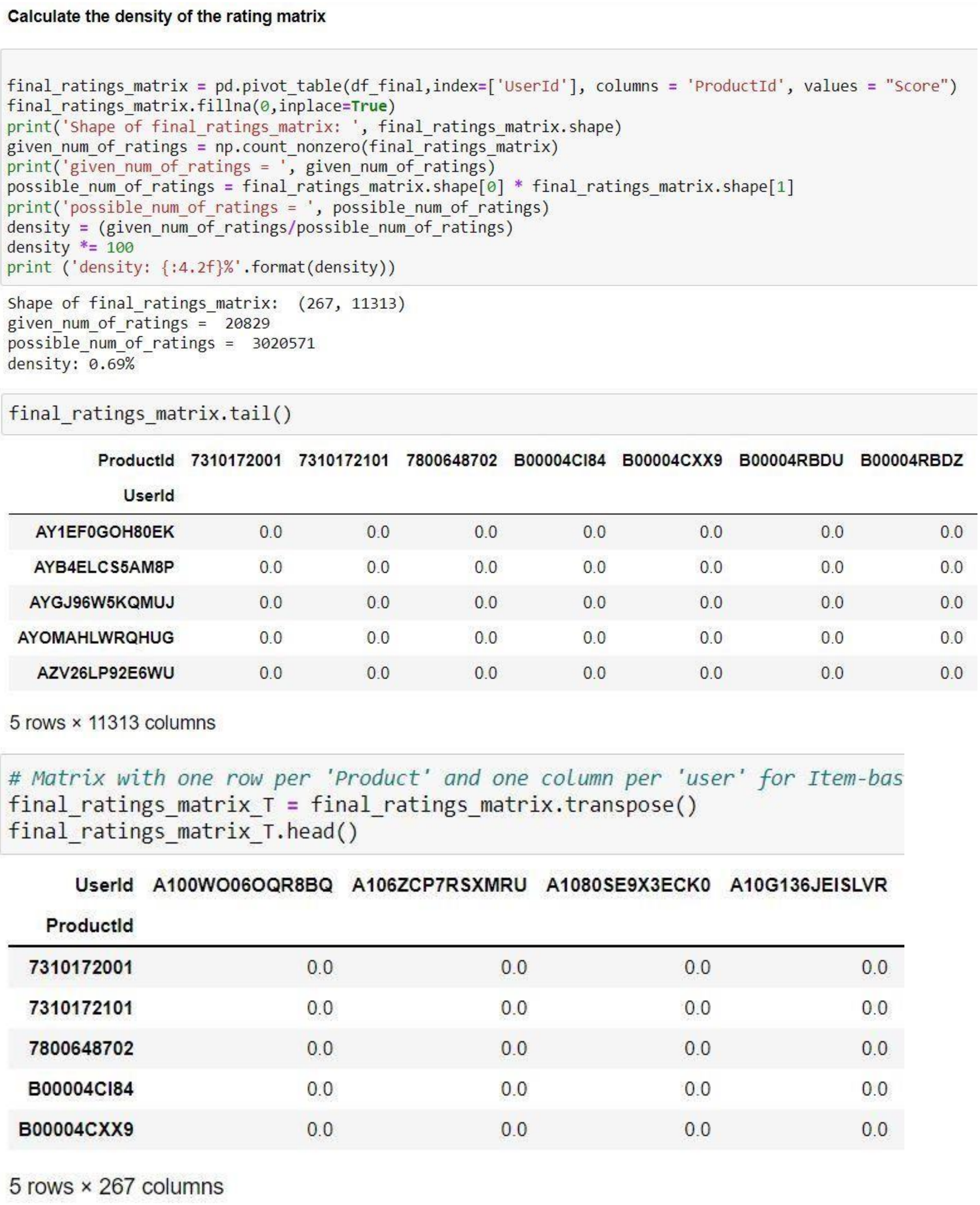
	ProductId	UserId	Score
14	B001GVISJM	A2MUGFV2TDQ47K	5
44	B001EO5QW8	A2G7B7FKP2O2PU	5
46	B001EO5QW8	AQLL2R1PPR46X	5
109	B001REEG6C	AY12DBB0U420B	5
141	B001GVISJW	A2YIO225BTKVPU	4



```
print('Number of users who have rated 50 or more items =', len(df_final))
print('Number of unique USERS in final data = ', df_final['UserId'].nunique())
print('Number of unique ITEMS in final data = ', df_final['ProductId'].nunique())
```

```
Number of users who have rated 50 or more items = 22941
Number of unique USERS in final data =  267
Number of unique ITEMS in final data =  11313
```

df_final has users who have rated 50 or more items



Calculate the density of the rating matrix

```
final_ratings_matrix = pd.pivot_table(df_final,index=['UserId'], columns = 'ProductId', values = "Score")
final_ratings_matrix.fillna(0,inplace=True)
print('Shape of final_ratings_matrix: ', final_ratings_matrix.shape)
given_num_of_ratings = np.count_nonzero(final_ratings_matrix)
print('given_num_of_ratings = ', given_num_of_ratings)
possible_num_of_ratings = final_ratings_matrix.shape[0] * final_ratings_matrix.shape[1]
print('possible_num_of_ratings = ', possible_num_of_ratings)
density = (given_num_of_ratings/possible_num_of_ratings)
density *= 100
print ('density: {:4.2f}%'.format(density))
```

```
Shape of final_ratings_matrix:  (267, 11313)
given_num_of_ratings =  20829
possible_num_of_ratings =  3020571
density: 0.69%
```

```
final_ratings_matrix.tail()
```

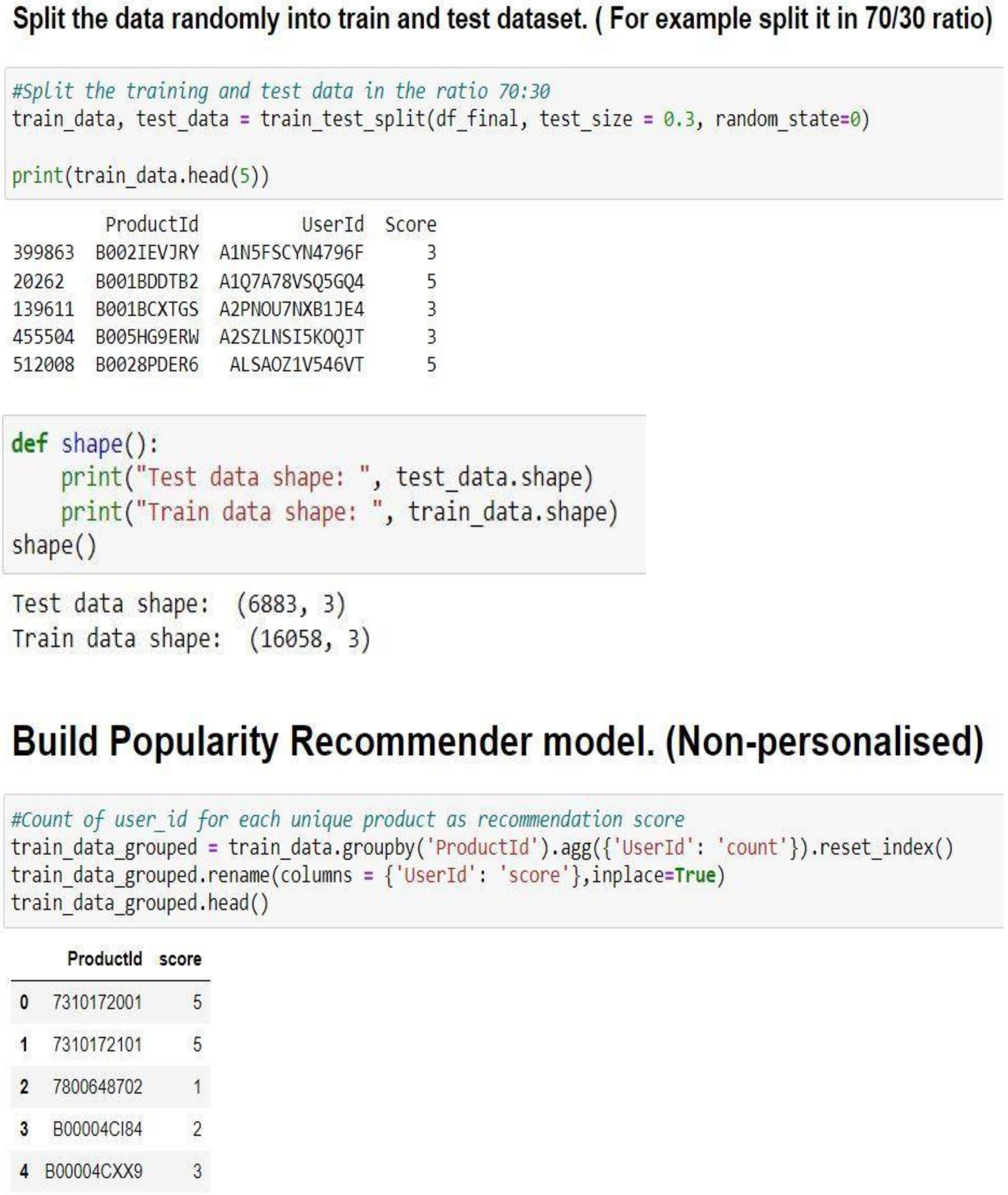
ProductId	7310172001	7310172101	7800648702	B00004CI84	B00004CXX9	B00004RBDU	B00004RBDZ
UserId							
AY1EF0GOH80EK	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AYB4ELCS5AM8P	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AYGJ96W5KQMUJ	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AYOMAHLWRQHUG	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AZV26LP92E6WU	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 11313 columns

```
# Matrix with one row per 'Product' and one column per 'user' for Item-bas
final_ratings_matrix_T = final_ratings_matrix.transpose()
final_ratings_matrix_T.head()
```

UserId	A100WO06OQR8BQ	A106ZCP7RSXMRU	A1080SE9X3ECK0	A10G136JEISLVR
ProductId				
7310172001	0.0	0.0	0.0	0.0
7310172101	0.0	0.0	0.0	0.0
7800648702	0.0	0.0	0.0	0.0
B00004CI84	0.0	0.0	0.0	0.0
B00004CXX9	0.0	0.0	0.0	0.0

5 rows × 267 columns



Split the data randomly into train and test dataset. (For example split it in 70/30 ratio)

```
#Split the training and test data in the ratio 70:30
train_data, test_data = train_test_split(df_final, test_size = 0.3, random_state=0)

print(train_data.head(5))
```

```
         ProductId          UserId  Score
399863  B002IEVJRY  A1N5FSCYN4796F      3
20262   B001BDDTB2  A1Q7A78VSQ5GQ4      5
139611  B001BCXTGS  A2PNOU7NXB1JE4      3
455504  B005HG9ERW  A2SZLNSI5KOQJT      3
512008  B0028PDER6   ALSAOZ1V546VT      5
```

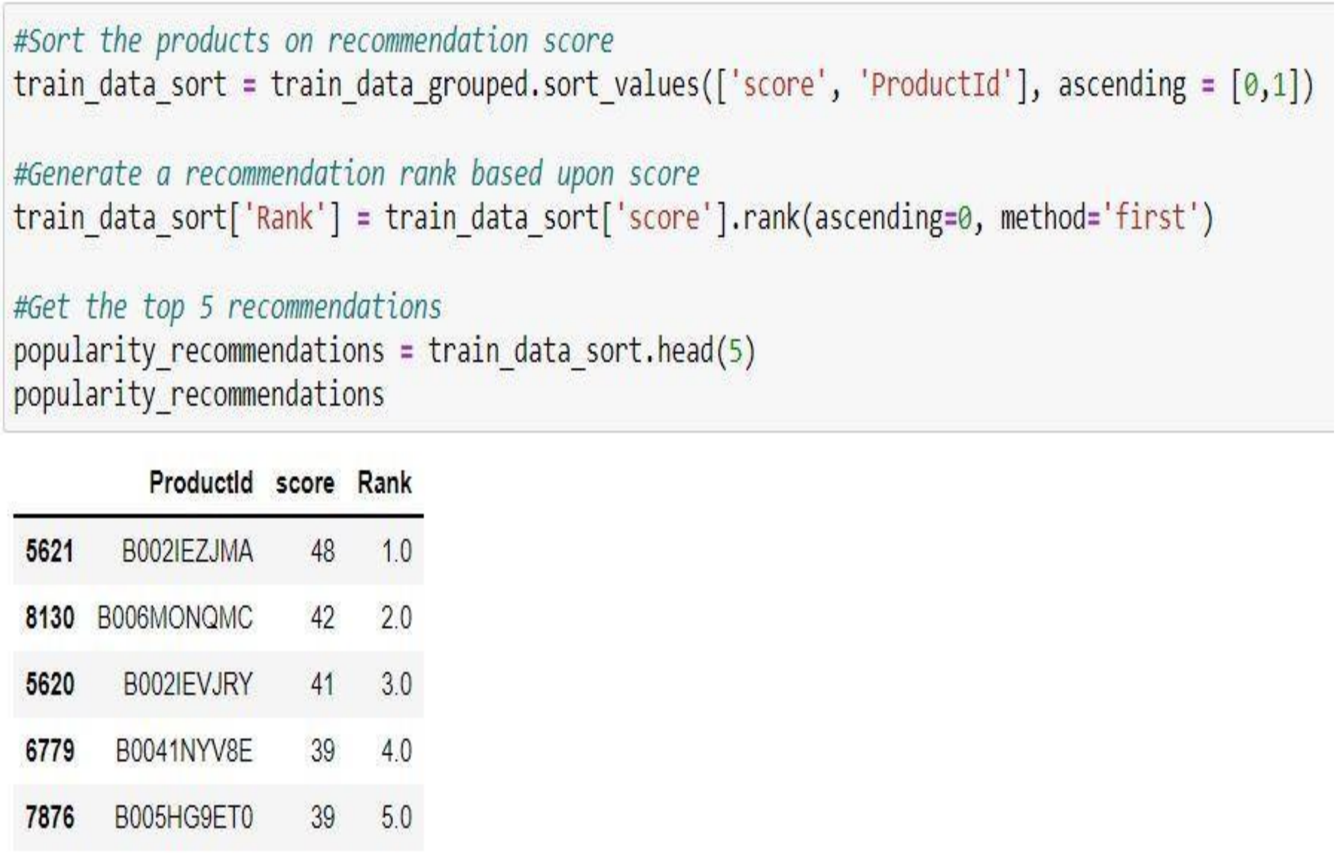
```
def shape():
    print("Test data shape: ", test_data.shape)
    print("Train data shape: ", train_data.shape)
shape()
```

```
Test data shape:  (6883, 3)
Train data shape:  (16058, 3)
```

Build Popularity Recommender model. (Non-personalised)

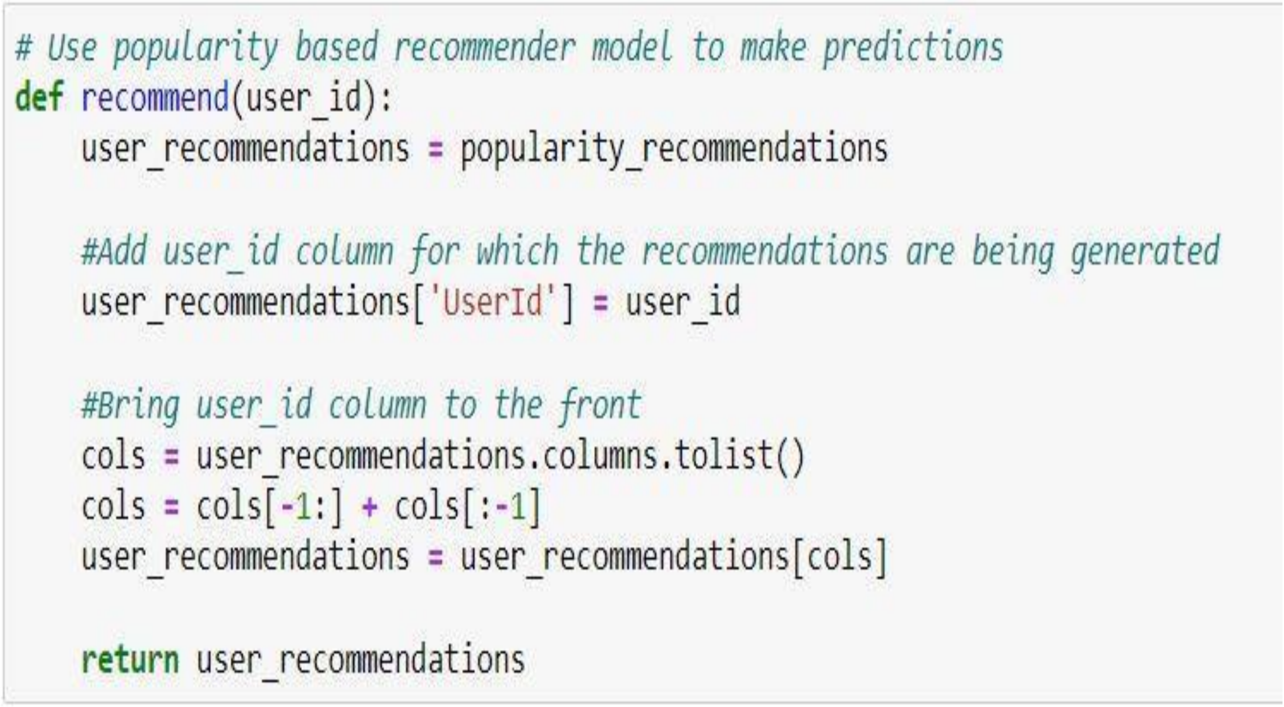
```
#Count of user_id for each unique product as recommendation score
train_data_grouped = train_data.groupby('ProductId').agg({'UserId': 'count'}).reset_index()
train_data_grouped.rename(columns = {'UserId': 'score'},inplace=True)
train_data_grouped.head()
```

	ProductId	score
0	7310172001	5
1	7310172101	5
2	7800648702	1
3	B00004CI84	2
4	B00004CXX9	3

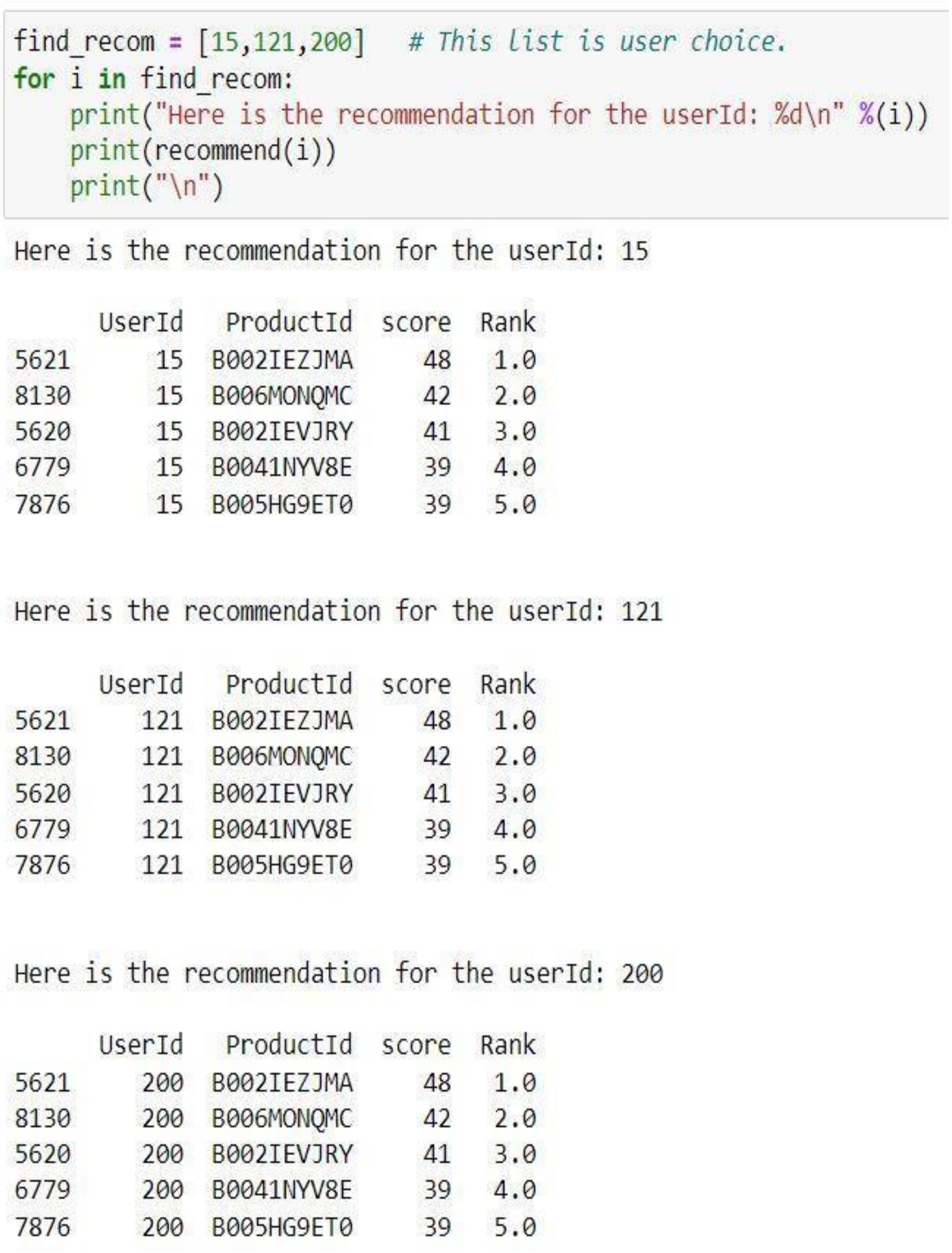


```
#Sort the products on recommendation score
train_data_sort = train_data_grouped.sort_values(['score', 'ProductId'], ascending = [0,1])

#Generate a recommendation rank based upon score
train_data_sort['Rank'] = train_data_sort['score'].rank(ascending=0, method='first')

#Get the top 5 recommendations
popularity_recommendations = train_data_sort.head(5)
popularity_recommendations
```

	ProductId	score	Rank
5621	B002IEZJMA	48	1.0
8130	B006MONQMC	42	2.0
5620	B002IEVJRY	41	3.0
6779	B0041NYV8E	39	4.0
7876	B005HG9ET0	39	5.0

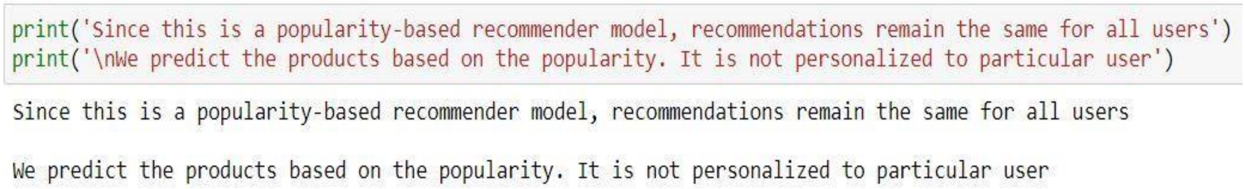


```
# Use popularity based recommender model to make predictions
def recommend(user_id):
    user_recommendations = popularity_recommendations

    #Add user_id column for which the recommendations are being generated
    user_recommendations['UserId'] = user_id

    #Bring user_id column to the front
    cols = user_recommendations.columns.tolist()
    cols = cols[-1:] + cols[:-1]
    user_recommendations = user_recommendations[cols]

    return user_recommendations
```



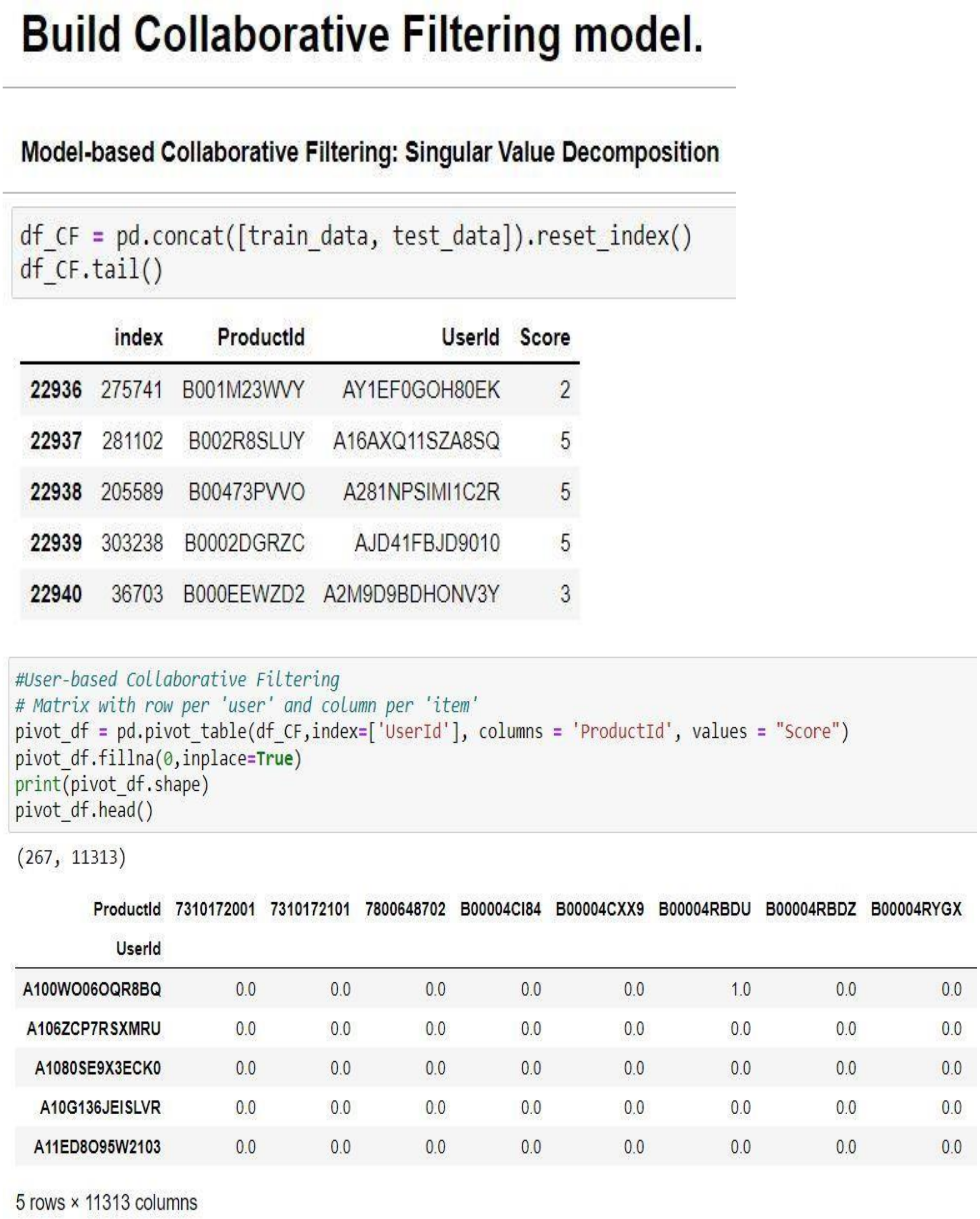
```
find_recom = [15,121,200]   # This list is user choice.
for i in find_recom:
    print("Here is the recommendation for the userId: %d\n" %(i))
    print(recommend(i))
    print("\n")
```

```
Here is the recommendation for the userId: 15

      UserId   ProductId  score  Rank
5621      15  B002IEZJMA     48   1.0
8130      15  B006MONQMC     42   2.0
5620      15  B002IEVJRY     41   3.0
6779      15  B0041NYV8E     39   4.0
7876      15  B005HG9ET0     39   5.0



Here is the recommendation for the userId: 121

      UserId   ProductId  score  Rank
5621     121  B002IEZJMA     48   1.0
8130     121  B006MONQMC     42   2.0
5620     121  B002IEVJRY     41   3.0
6779     121  B0041NYV8E     39   4.0
7876     121  B005HG9ET0     39   5.0



Here is the recommendation for the userId: 200

      UserId   ProductId  score  Rank
5621     200  B002IEZJMA     48   1.0
8130     200  B006MONQMC     42   2.0
5620     200  B002IEVJRY     41   3.0
6779     200  B0041NYV8E     39   4.0
7876     200  B005HG9ET0     39   5.0
```



```
print('Since this is a popularity-based recommender model, recommendations remain the same for all users')
print('\nWe predict the products based on the popularity. It is not personalized to particular user')
```

```
Since this is a popularity-based recommender model, recommendations remain the same for all users

We predict the products based on the popularity. It is not personalized to particular user
```



Build Collaborative Filtering model.

Model-based Collaborative Filtering: Singular Value Decomposition

```
df_CF = pd.concat([train_data, test_data]).reset_index()
df_CF.tail()
```

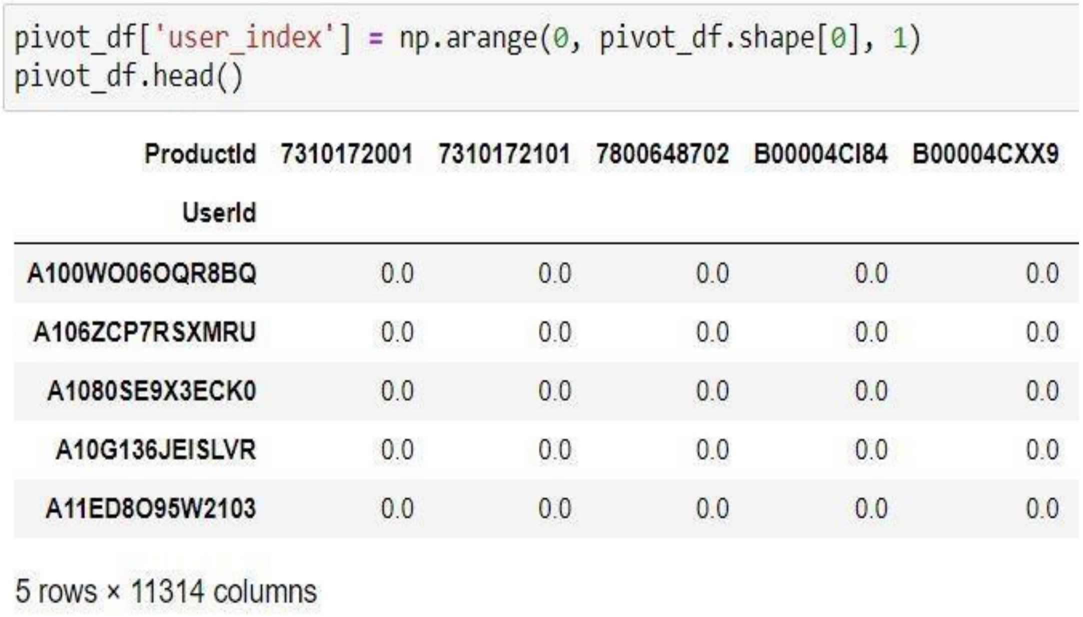
	index	ProductId	UserId	Score
22936	275741	B001M23WVY	AY1EF0GOH80EK	2
22937	281102	B002R8SLUY	A16AXQ11SZA8SQ	5
22938	205589	B00473PVVO	A281NPSIMI1C2R	5
22939	303238	B0002DGRZC	AJD41FBJD9010	5
22940	36703	B000EEWZD2	A2M9D9BDHONV3Y	3

```
#User-based Collaborative Filtering
# Matrix with row per 'user' and column per 'item'
pivot_df = pd.pivot_table(df_CF,index=['UserId'], columns = 'ProductId', values = "Score")
pivot_df.fillna(0,inplace=True)
print(pivot_df.shape)
pivot_df.head()
```

```
(267, 11313)
```

ProductId	7310172001	7310172101	7800648702	B00004CI84	B00004CXX9	B00004RBDU	B00004RBDZ	B00004RYGX
UserId								
A100WO06OQR8BQ	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
A106ZCP7RSXMRU	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A1080SE9X3ECK0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A10G136JEISLVR	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A11ED8O95W2103	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

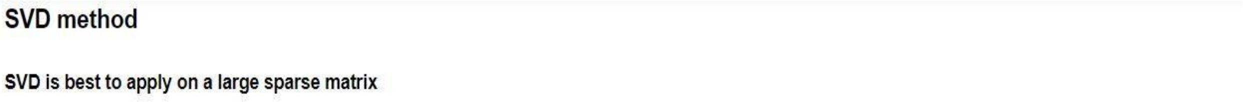
5 rows × 11313 columns



```
pivot_df['user_index'] = np.arange(0, pivot_df.shape[0], 1)
pivot_df.head()
```

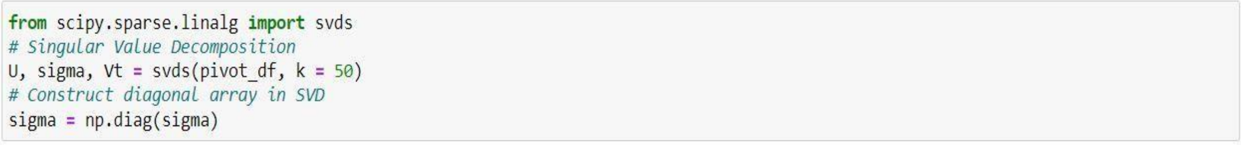
ProductId UserId	7310172001	7310172101	7800648702	B00004CI84	B00004CXX9
A100WO06OQR8BQ	0.0	0.0	0.0	0.0	0.0
A106ZCP7RSXMRU	0.0	0.0	0.0	0.0	0.0
A1080SE9X3ECK0	0.0	0.0	0.0	0.0	0.0
A10G136JEISLVR	0.0	0.0	0.0	0.0	0.0
A11ED8O95W2103	0.0	0.0	0.0	0.0	0.0

5 rows × 11314 columns

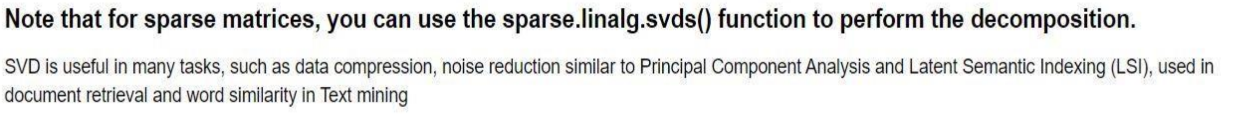


SVD method

SVD is best to apply on a large sparse matrix

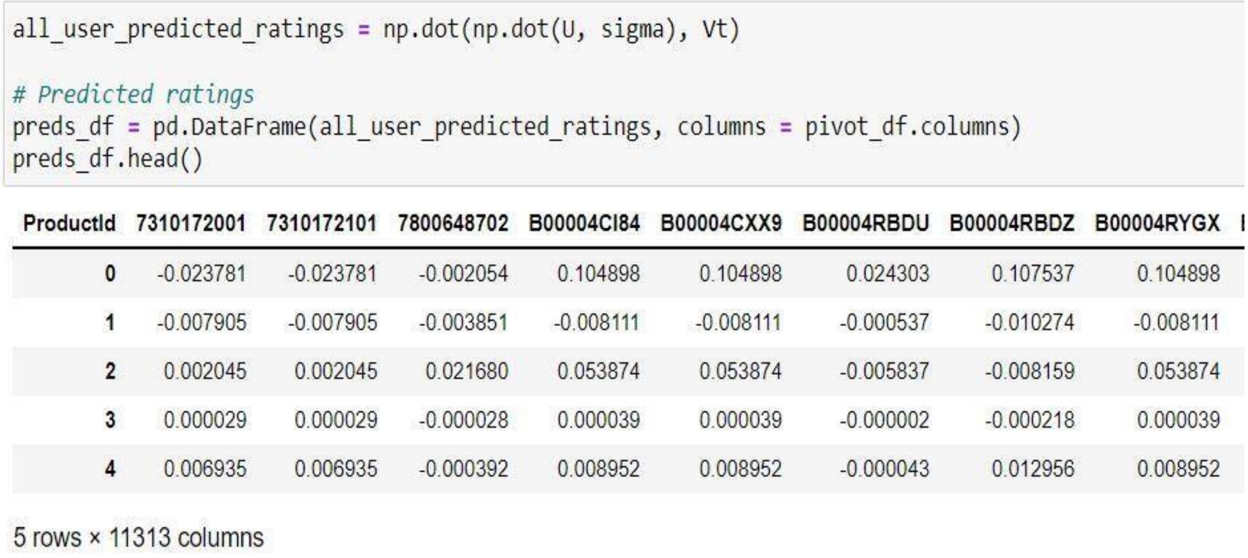


```
from scipy.sparse.linalg import svds
# Singular Value Decomposition
U, sigma, Vt = svds(pivot_df, k = 50)
# Construct diagonal array in SVD
sigma = np.diag(sigma)
```



Note that for sparse matrices, you can use the sparse.linalg.svds() function to perform the decomposition.

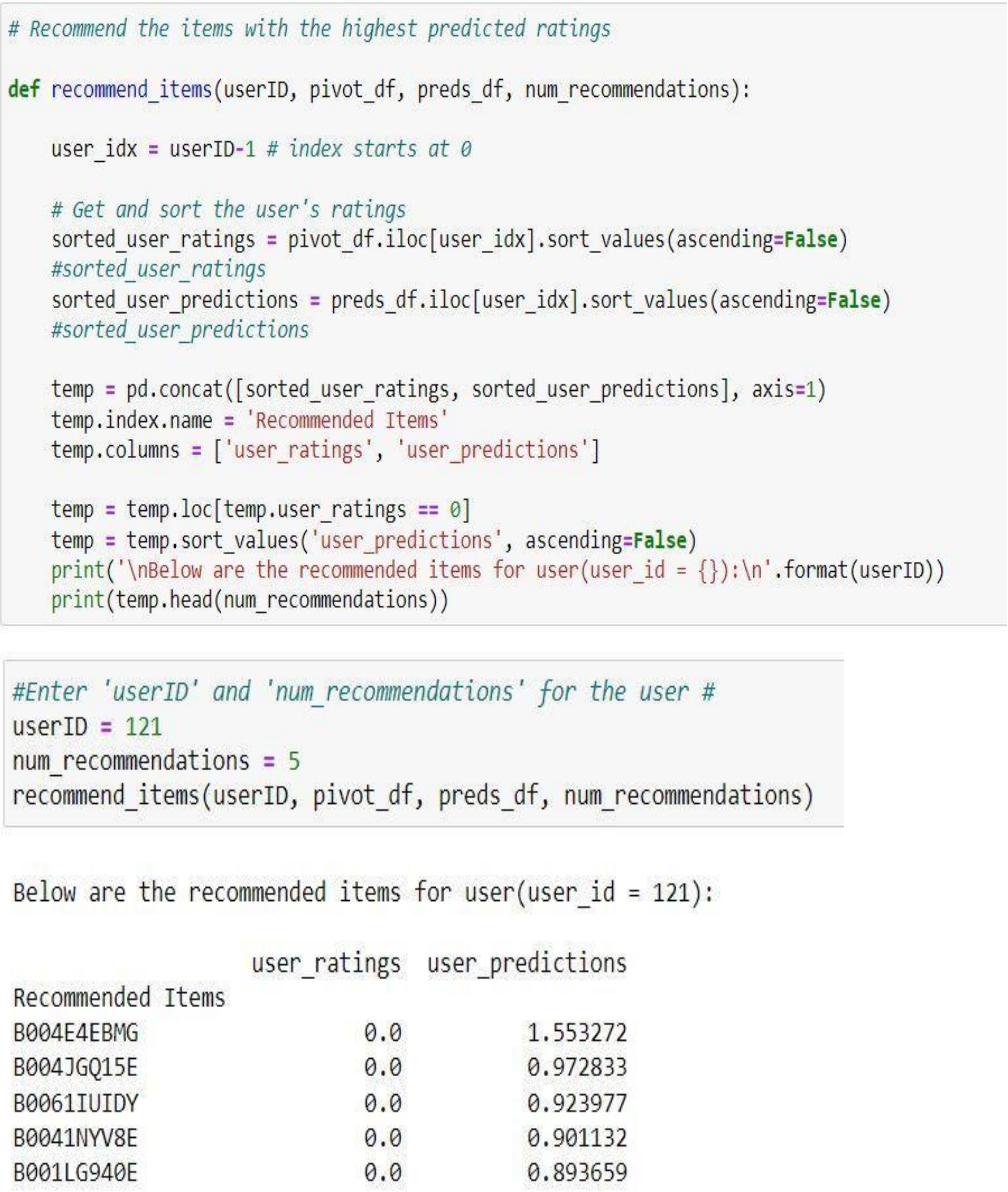
SVD is useful in many tasks, such as data compression, noise reduction similar to Principal Component Analysis and Latent Semantic Indexing (LSI), used in document retrieval and word similarity in Text mining



```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt)

# Predicted ratings
preds_df = pd.DataFrame(all_user_predicted_ratings, columns = pivot_df.columns)
preds_df.head()
```

ProductId	7310172001	7310172101	7800648702	B00004CI84	B00004CXX9	B00004RBDU	B00004RBDZ	B00004RYGX
0	-0.023781	-0.023781	-0.002054	0.104898	0.104898	0.024303	0.107537	0.104898
1	-0.007905	-0.007905	-0.003851	-0.008111	-0.008111	-0.000537	-0.010274	-0.008111
2	0.002045	0.002045	0.021680	0.053874	0.053874	-0.005837	-0.008159	0.053874
3	0.000029	0.000029	-0.000028	0.000039	0.000039	-0.000002	-0.000218	0.000039
4	0.006935	0.006935	-0.000392	0.008952	0.008952	-0.000043	0.012956	0.008952

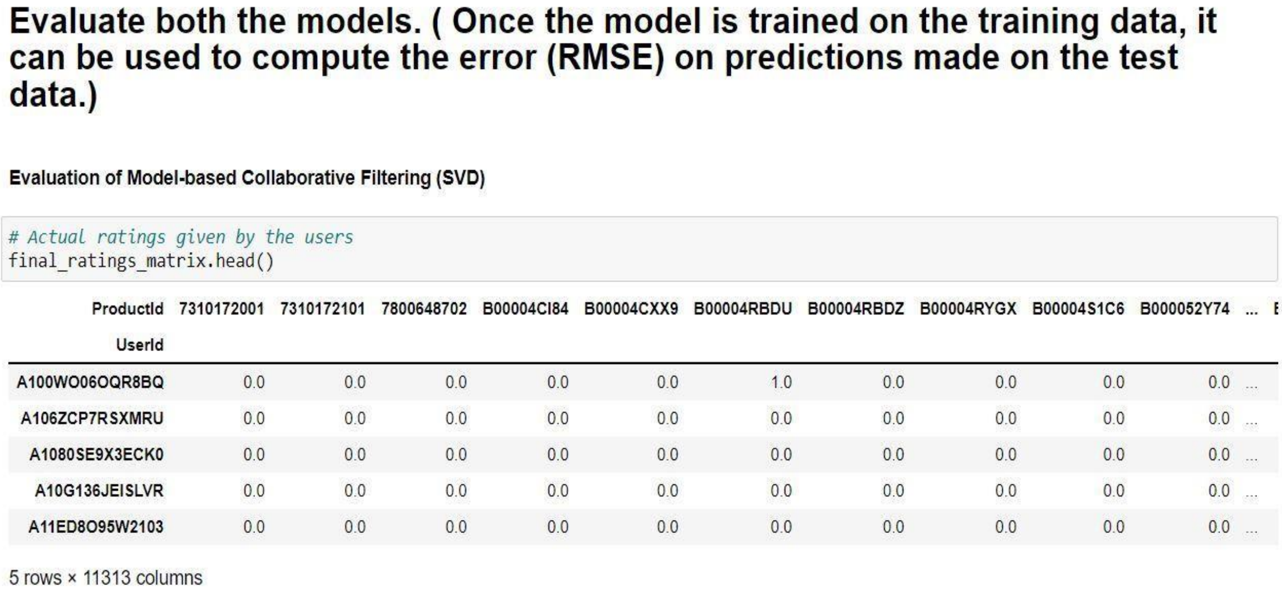
5 rows × 11313 columns



```
# Recommend the items with the highest predicted ratings

def recommend_items(userID, pivot_df, preds_df, num_recommendations):

    user_idx = userID-1 # index starts at 0

    # Get and sort the user's ratings
    sorted_user_ratings = pivot_df.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_ratings
    sorted_user_predictions = preds_df.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_predictions

    temp = pd.concat([sorted_user_ratings, sorted_user_predictions], axis=1)
    temp.index.name = 'Recommended Items'
    temp.columns = ['user_ratings', 'user_predictions']

    temp = temp.loc[temp.user_ratings == 0]
    temp = temp.sort_values('user_predictions', ascending=False)
    print('\nBelow are the recommended items for user(user_id = {}):\n'.format(userID))
    print(temp.head(num_recommendations))
```

```
#Enter 'userID' and 'num_recommendations' for the user #
userID = 121
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

```
Below are the recommended items for user(user_id = 121):

                   user_ratings  user_predictions
Recommended Items
B004E4EBMG                  0.0          1.553272
B004JGQ15E                  0.0          0.972833
B0061IUIDY                  0.0          0.923977
B0041NYV8E                  0.0          0.901132
B001LG940E                  0.0          0.893659
```



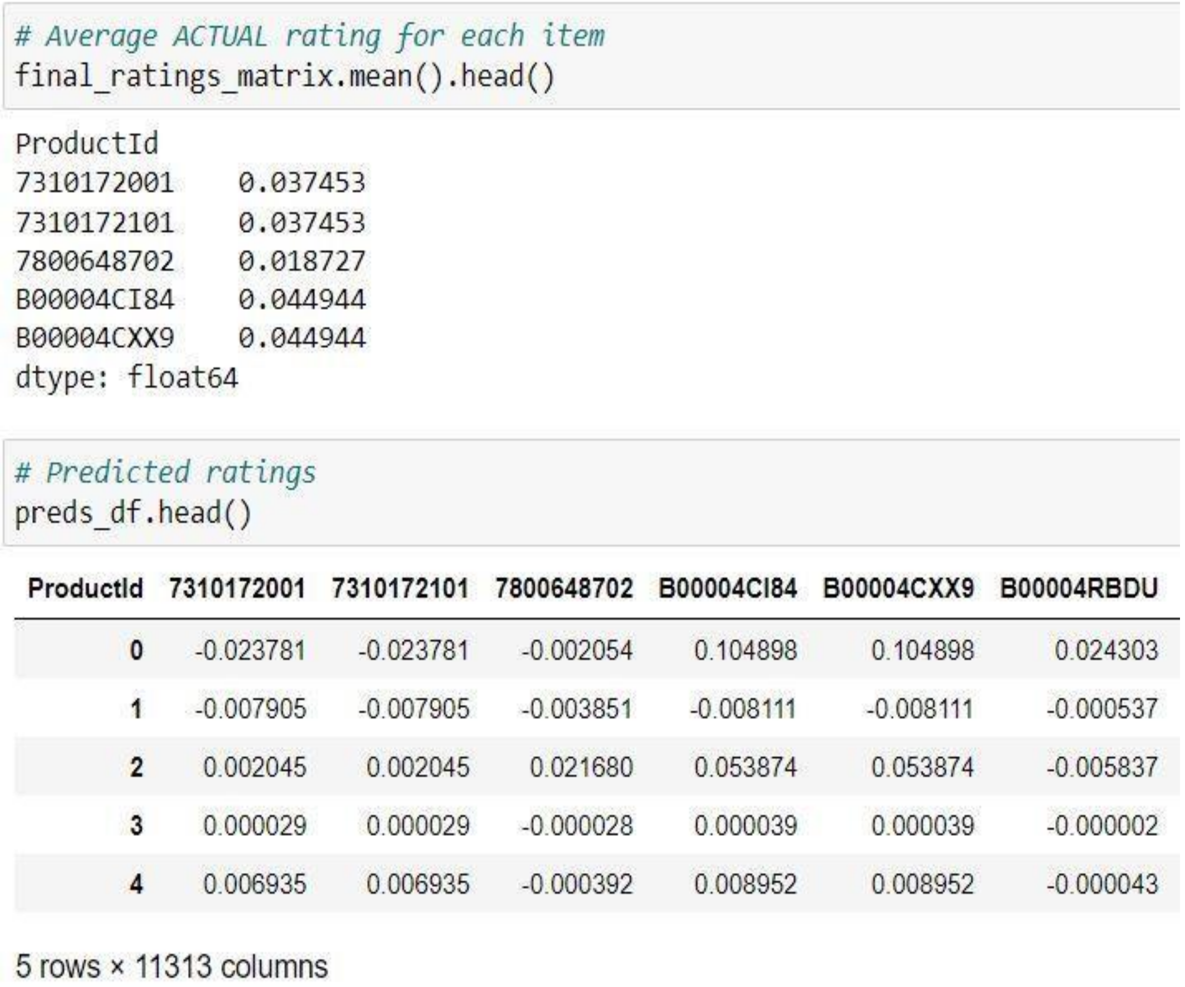
Evaluate both the models. (Once the model is trained on the training data, it can be used to compute the error (RMSE) on predictions made on the test data.)

Evaluation of Model-based Collaborative Filtering (SVD)

```
# Actual ratings given by the users
final_ratings_matrix.head()
```

ProductId	7310172001	7310172101	7800648702	B00004CI84	B00004CXX9	B00004RBDU	B00004RBDZ	B00004RYGX	B00004S1C6	B000052Y74	...	E
UserId												
A100WO06OQR8BQ	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	
A106ZCP7RSXMRU	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
A1080SE9X3ECK0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
A10G136JEISLVR	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
A11ED8O95W2103	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

5 rows × 11313 columns



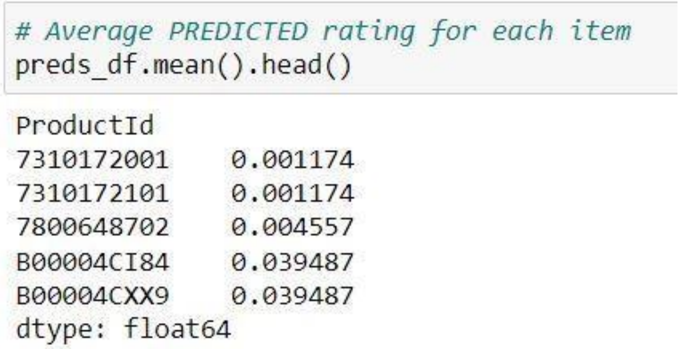
```
# Average ACTUAL rating for each item
final_ratings_matrix.mean().head()
```

```
ProductId
7310172001    0.037453
7310172101    0.037453
7800648702    0.018727
B00004CI84    0.044944
B00004CXX9    0.044944
dtype: float64
```

```
# Predicted ratings
preds_df.head()
```

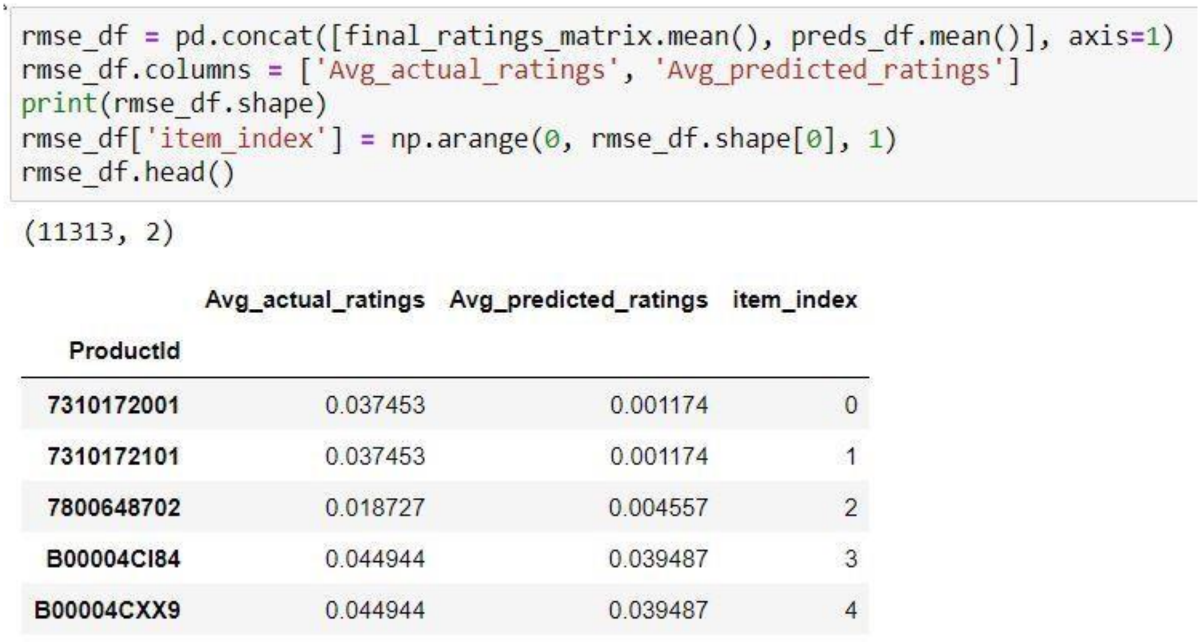
ProductId	7310172001	7310172101	7800648702	B00004CI84	B00004CXX9	B00004RBDU
0	-0.023781	-0.023781	-0.002054	0.104898	0.104898	0.024303
1	-0.007905	-0.007905	-0.003851	-0.008111	-0.008111	-0.000537
2	0.002045	0.002045	0.021680	0.053874	0.053874	-0.005837
3	0.000029	0.000029	-0.000028	0.000039	0.000039	-0.000002
4	0.006935	0.006935	-0.000392	0.008952	0.008952	-0.000043

5 rows × 11313 columns



```
# Average PREDICTED rating for each item
preds_df.mean().head()
```

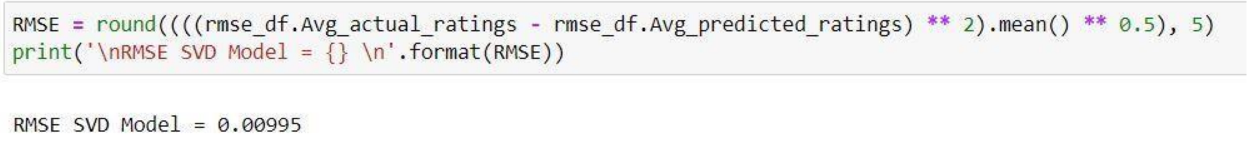
```
ProductId
7310172001    0.001174
7310172101    0.001174
7800648702    0.004557
B00004CI84    0.039487
B00004CXX9    0.039487
dtype: float64
```



```
rmse_df = pd.concat([final_ratings_matrix.mean(), preds_df.mean()], axis=1)
rmse_df.columns = ['Avg_actual_ratings', 'Avg_predicted_ratings']
print(rmse_df.shape)
rmse_df['item_index'] = np.arange(0, rmse_df.shape[0], 1)
rmse_df.head()
```

```
(11313, 2)
```

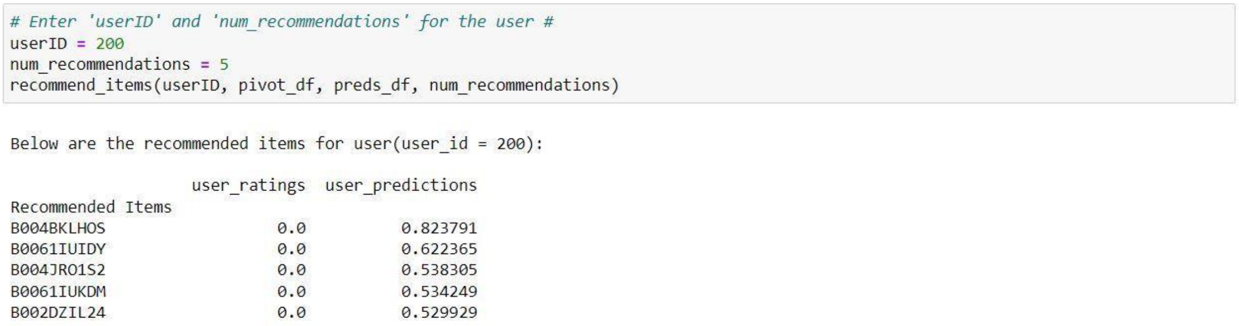
ProductId	Avg_actual_ratings	Avg_predicted_ratings	item_index
7310172001	0.037453	0.001174	0
7310172101	0.037453	0.001174	1
7800648702	0.018727	0.004557	2
B00004CI84	0.044944	0.039487	3
B00004CXX9	0.044944	0.039487	4



```
RMSE = round((((rmse_df.Avg_actual_ratings - rmse_df.Avg_predicted_ratings) ** 2).mean() ** 0.5), 5)
print('\nRMSE SVD Model = {} \n'.format(RMSE))
```

```
RMSE SVD Model = 0.00995
```

Get top - K (K = 5) recommendations. Since our goal is to recommend new products to each user based on his/her habits, we will recommend 5 new products.



```
# Enter 'userID' and 'num_recommendations' for the user #
userID = 200
num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
```

```
Below are the recommended items for user(user_id = 200):

                   user_ratings  user_predictions
Recommended Items
B004BKLHOS                  0.0          0.823791
B0061IUIDY                  0.0          0.622365
B004JRO1S2                  0.0          0.538305
B0061IUKDM                  0.0          0.534249
B002DZIL24                  0.0          0.529929
```

PROBLEM DEFINITION

The problem we are going to solve is how to help users select products which they may like and to make recommendation to stimulate sales and increase profits.

Firstly, we decided to choose the Amazon Fine Food Reviews dataset which consists of 568,454 food reviews Amazon users left up to October 2012 as our dataset.

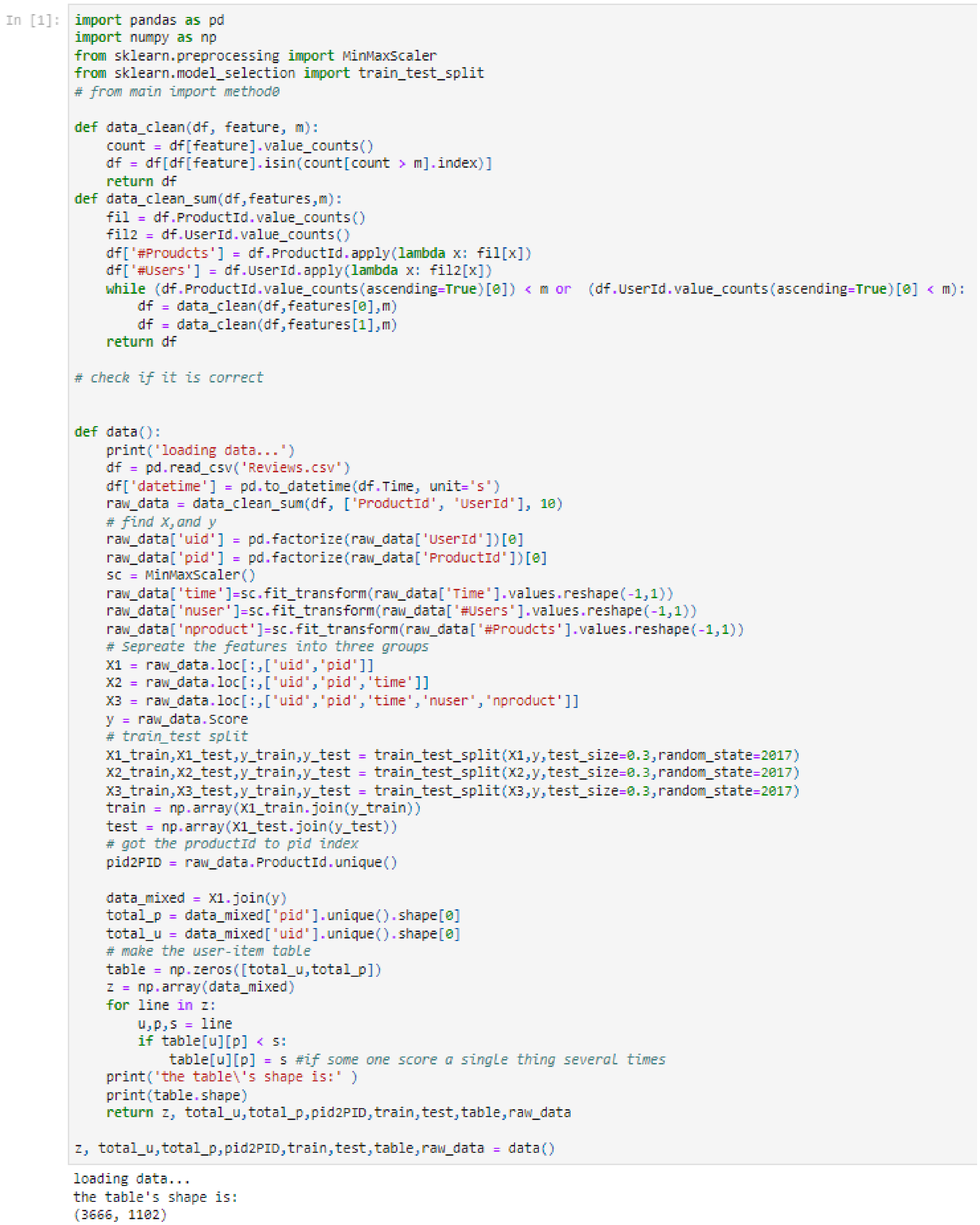
Secondly, our recommendation system is based on users rating prediction. We assume that users tend to like the products that have a score of greater than 4 and we will consider the highest 5 scores product as our recommendation candidates.

Thirdly, we implemented several algorithms to predict the scores of each product for each user.

Data Pre processing

Here, Take the data which the user and item appear more than 10 times in order to reduce the calculation because we just want to take a small-scale test.

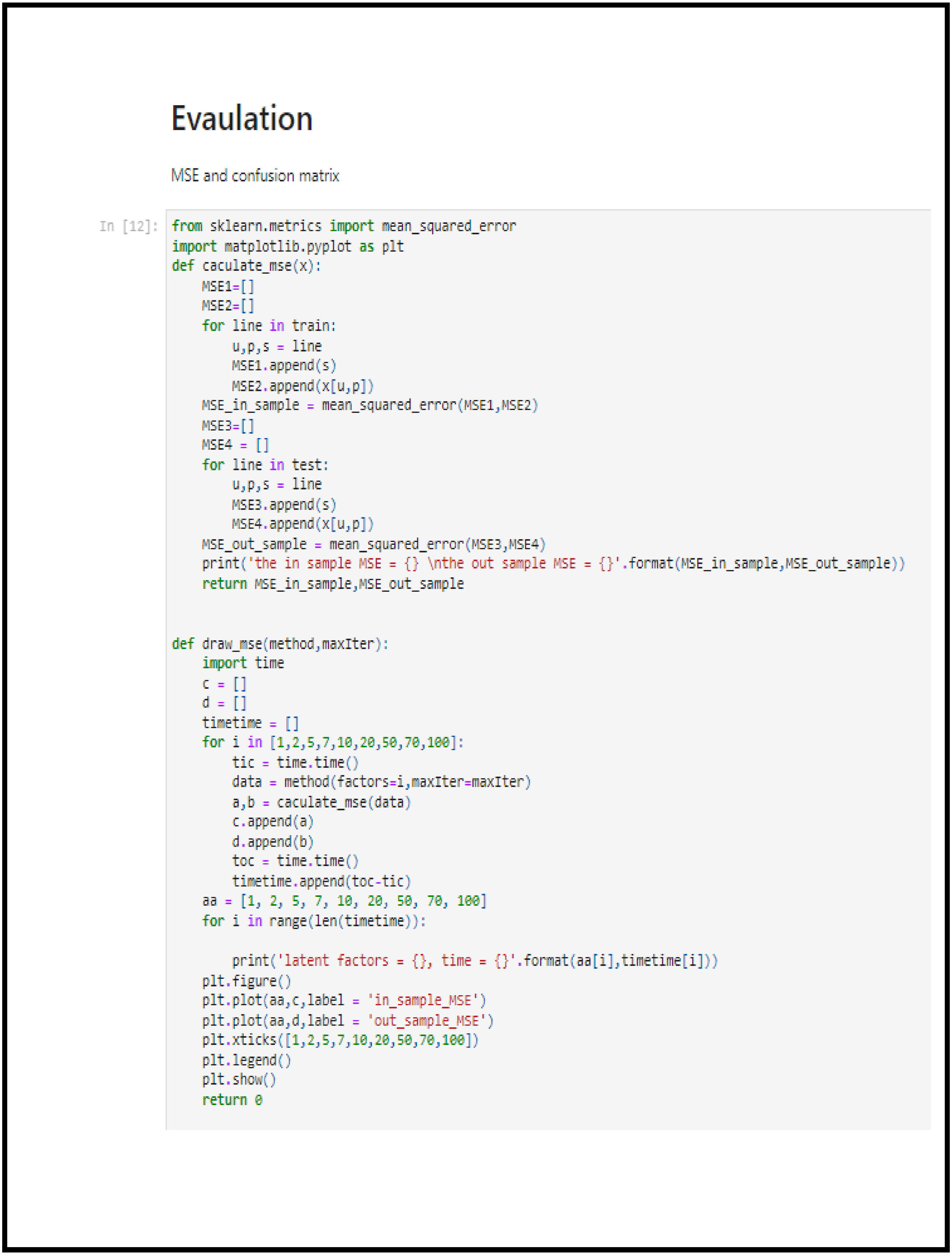
The data () function returns the total number of users and products, the user-item table and also the train & test data set.



In [1]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
# from main import method0

def data_clean(df, feature, m):
    count = df[feature].value_counts()
    df = df[df[feature].isin(count[count > m].index)]
    return df
def data_clean_sum(df,features,m):
    fil = df.ProductId.value_counts()
    fil2 = df.UserId.value_counts()
    df['#Proudcts'] = df.ProductId.apply(lambda x: fil[x])
    df['#Users'] = df.UserId.apply(lambda x: fil2[x])
    while (df.ProductId.value_counts(ascending=True)[0]) < m or  (df.UserId.value_counts(ascending=True)[0] < m):
        df = data_clean(df,features[0],m)
        df = data_clean(df,features[1],m)
    return df

# check if it is correct


def data():
    print('loading data...')
    df = pd.read_csv('Reviews.csv')
    df['datetime'] = pd.to_datetime(df.Time, unit='s')
    raw_data = data_clean_sum(df, ['ProductId', 'UserId'], 10)
    # find X,and y
    raw_data['uid'] = pd.factorize(raw_data['UserId'])[0]
    raw_data['pid'] = pd.factorize(raw_data['ProductId'])[0]
    sc = MinMaxScaler()
    raw_data['time']=sc.fit_transform(raw_data['Time'].values.reshape(-1,1))
    raw_data['nuser']=sc.fit_transform(raw_data['#Users'].values.reshape(-1,1))
    raw_data['nproduct']=sc.fit_transform(raw_data['#Proudcts'].values.reshape(-1,1))
    # Seperate the features into three groups
    X1 = raw_data.loc[:,['uid','pid']]
    X2 = raw_data.loc[:,['uid','pid','time']]
    X3 = raw_data.loc[:,['uid','pid','time','nuser','nproduct']]
    y = raw_data.Score
    # train_test split
    X1_train,X1_test,y_train,y_test = train_test_split(X1,y,test_size=0.3,random_state=2017)
    X2_train,X2_test,y_train,y_test = train_test_split(X2,y,test_size=0.3,random_state=2017)
    X3_train,X3_test,y_train,y_test = train_test_split(X3,y,test_size=0.3,random_state=2017)
    train = np.array(X1_train.join(y_train))
    test = np.array(X1_test.join(y_test))
    # got the productId to pid index
    pid2PID = raw_data.ProductId.unique()

    data_mixed = X1.join(y)
    total_p = data_mixed['pid'].unique().shape[0]
    total_u = data_mixed['uid'].unique().shape[0]
    # make the user-item table
    table = np.zeros([total_u,total_p])
    z = np.array(data_mixed)
    for line in z:
        u,p,s = line
        if table[u][p] < s:
            table[u][p] = s #if some one score a single thing several times
    print('the table\'s shape is:' )
    print(table.shape)
    return z, total_u,total_p,pid2PID,train,test,table,raw_data

z, total_u,total_p,pid2PID,train,test,table,raw_data = data()
```

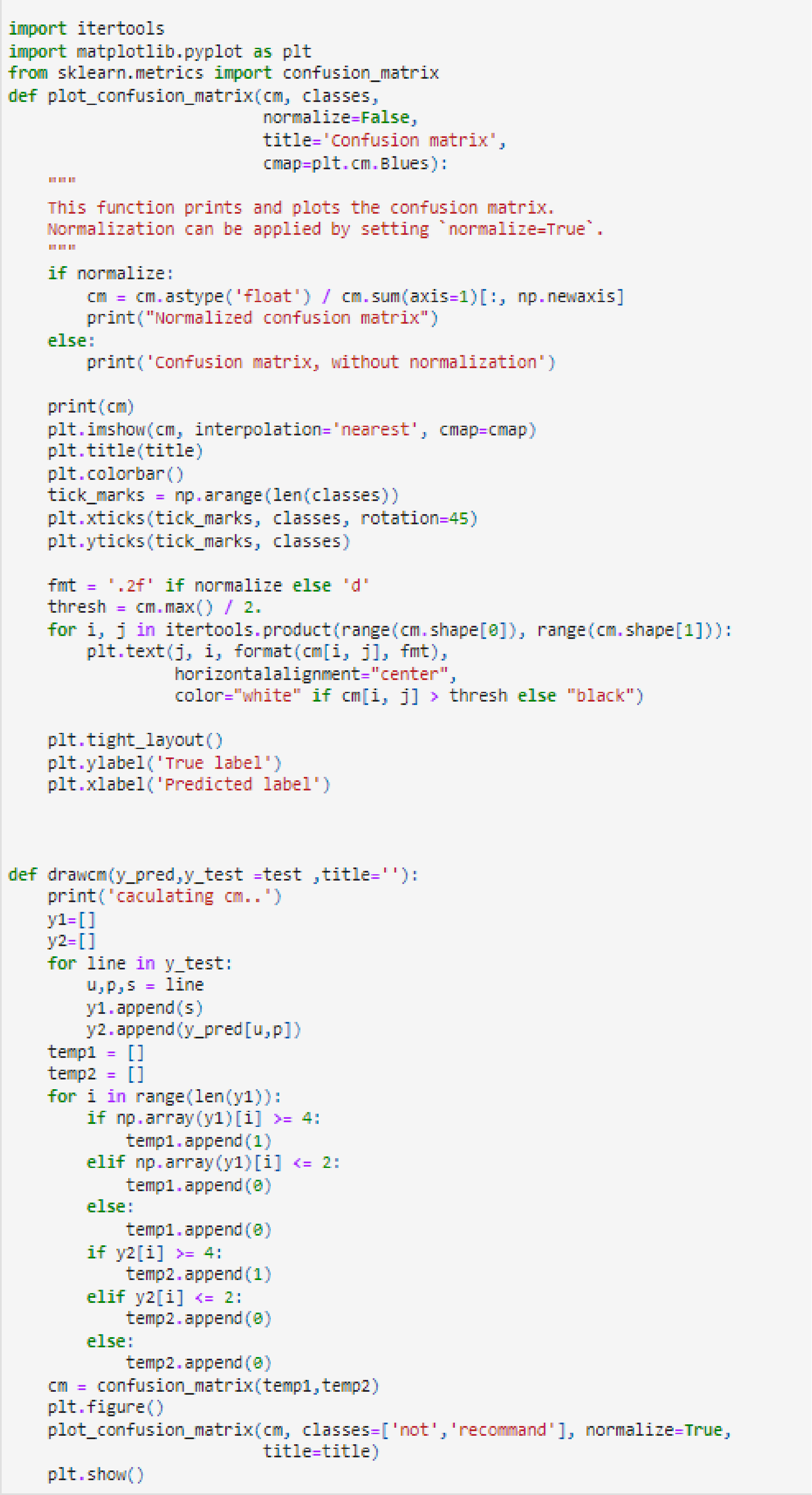
```
loading data...
the table's shape is:
(3666, 1102)
```



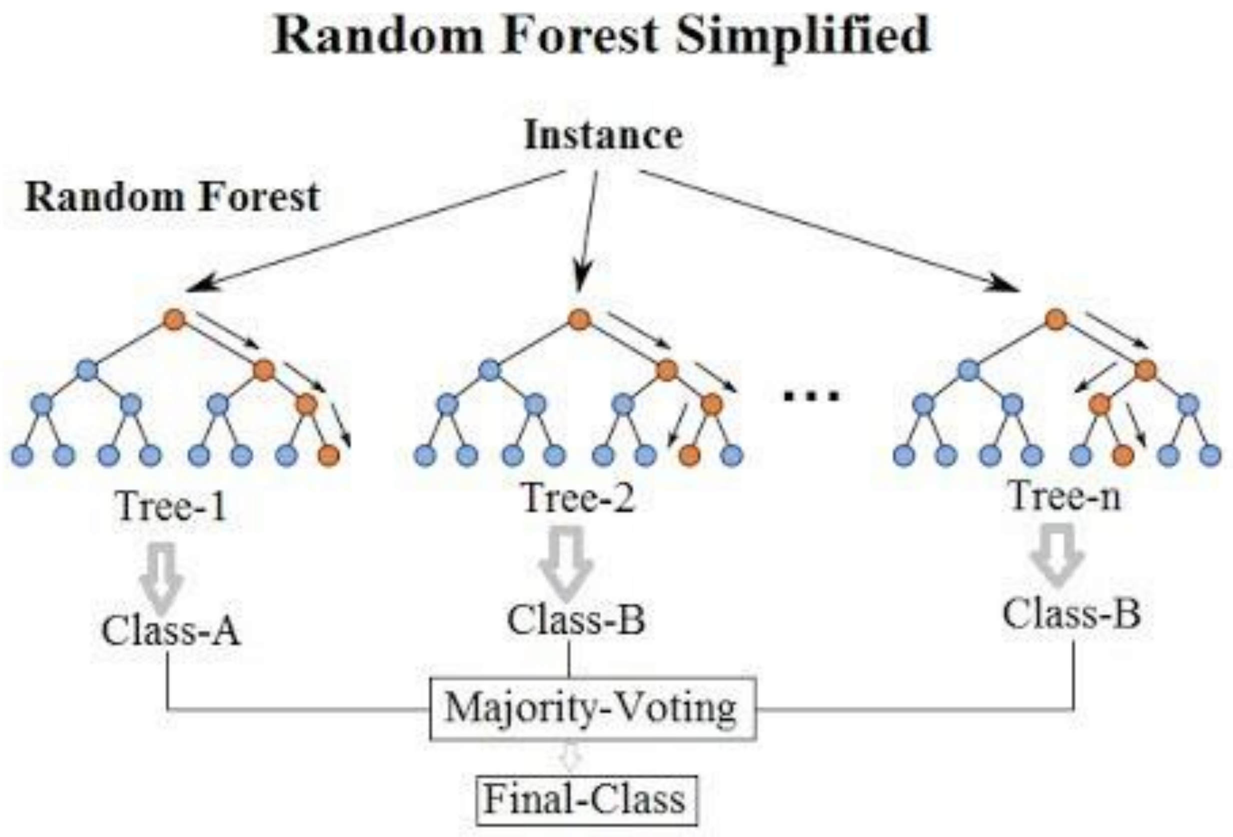
Evaulation

MSE and confusion matrix

```
In [12]: from sklearn.metrics import mean_squared_error
         import matplotlib.pyplot as plt
         def caculate_mse(x):
             MSE1=[]
             MSE2=[]
             for line in train:
                 u,p,s = line
                 MSE1.append(s)
                 MSE2.append(x[u,p])
             MSE_in_sample = mean_squared_error(MSE1,MSE2)
             MSE3=[]
             MSE4 = []
             for line in test:
                 u,p,s = line
                 MSE3.append(s)
                 MSE4.append(x[u,p])
             MSE_out_sample = mean_squared_error(MSE3,MSE4)
             print('the in sample MSE = {} \nthe out sample MSE = {}'.format(MSE_in_sample,MSE_out_sample))
             return MSE_in_sample,MSE_out_sample


         def draw_mse(method,maxIter):
             import time
             c = []
             d = []
             timetime = []
             for i in [1,2,5,7,10,20,50,70,100]:
                 tic = time.time()
                 data = method(factors=i,maxIter=maxIter)
                 a,b = caculate_mse(data)
                 c.append(a)
                 d.append(b)
                 toc = time.time()
                 timetime.append(toc-tic)
             aa = [1, 2, 5, 7, 10, 20, 50, 70, 100]
             for i in range(len(timetime)):

                 print('latent factors = {}, time = {}'.format(aa[i],timetime[i]))
             plt.figure()
             plt.plot(aa,c,label = 'in_sample_MSE')
             plt.plot(aa,d,label = 'out_sample_MSE')
             plt.xticks([1,2,5,7,10,20,50,70,100])
             plt.legend()
             plt.show()
             return 0
```



```python
import itertools
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')



def drawcm(y_pred,y_test =test ,title=''):
    print('caculating cm..')
    y1=[]
    y2=[]
    for line in y_test:
        u,p,s = line
        y1.append(s)
        y2.append(y_pred[u,p])
    temp1 = []
    temp2 = []
    for i in range(len(y1)):
        if np.array(y1)[i] >= 4:
            temp1.append(1)
        elif np.array(y1)[i] <= 2:
            temp1.append(0)
        else:
            temp1.append(0)
        if y2[i] >= 4:
            temp2.append(1)
        elif y2[i] <= 2:
            temp2.append(0)
        else:
            temp2.append(0)
    cm = confusion_matrix(temp1,temp2)
    plt.figure()
    plot_confusion_matrix(cm, classes=['not','recommand'], normalize=True,
                          title=title)
    plt.show()
```

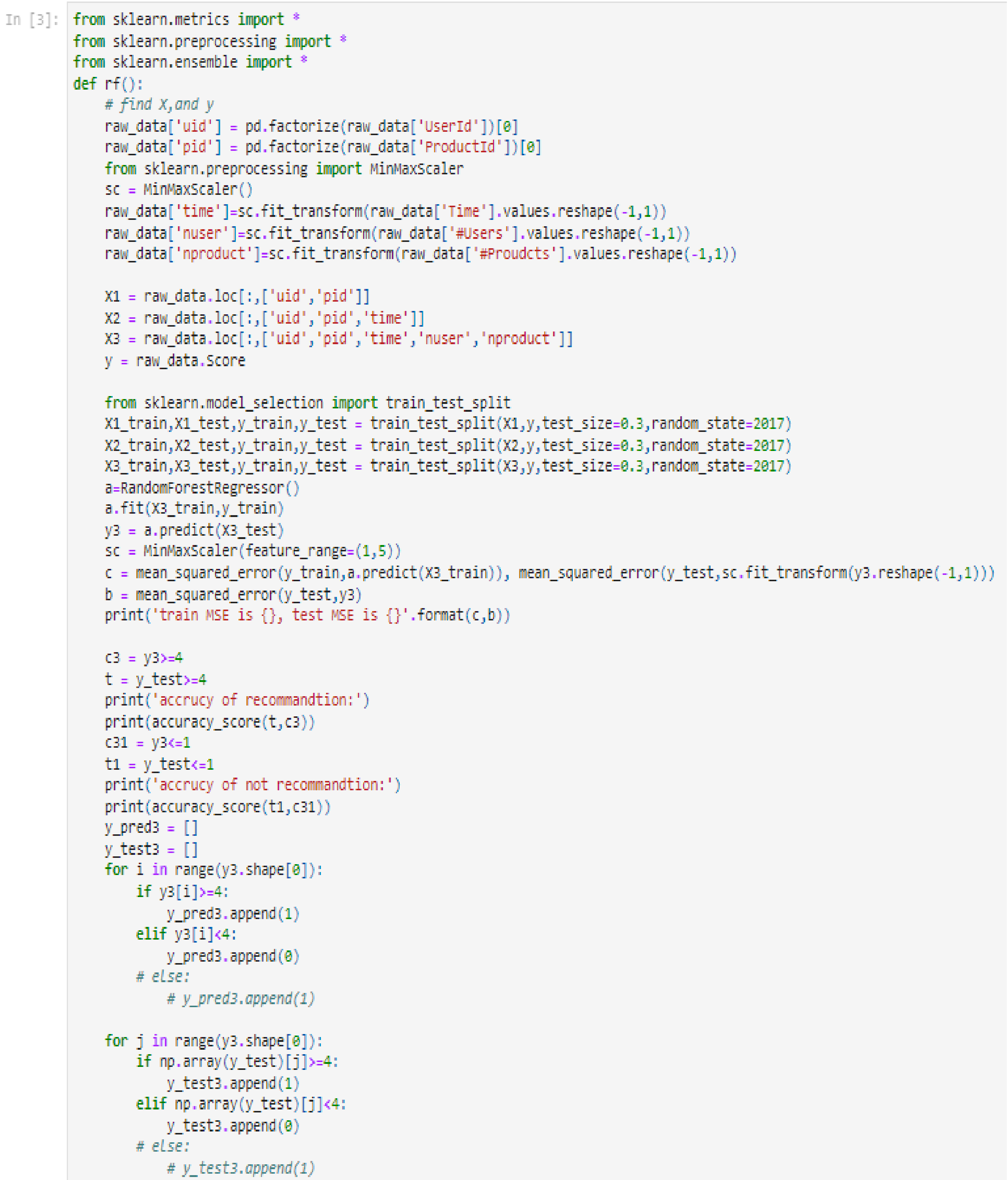


Random Forest Regressor

At the very first, we try to use rfr to build a simple model. The idea is very easy, just use the algorithm find the latent score of one pair of user and item. We can see the MSE is not very high, and the confusion matrix is acceptable.

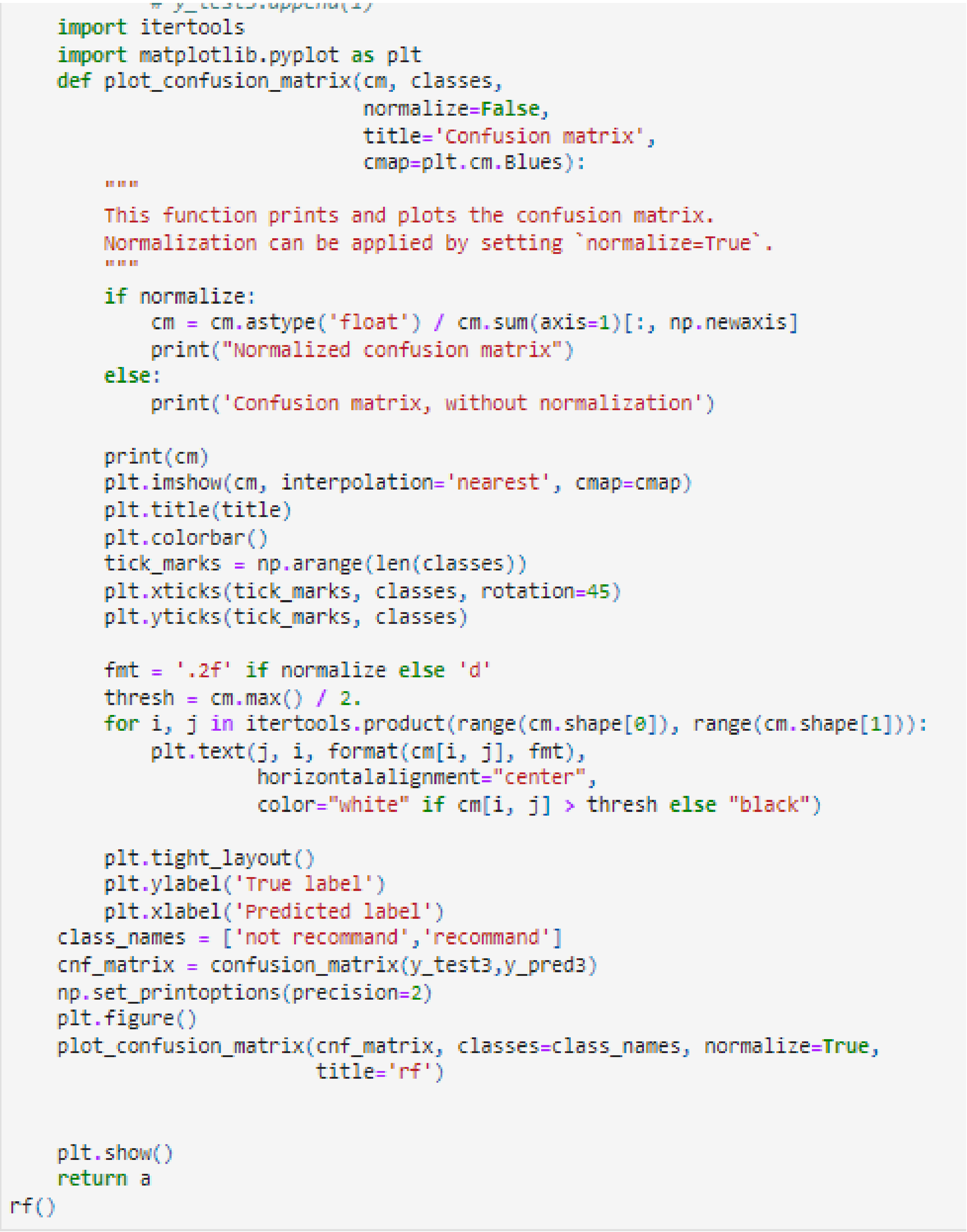
This part, the X are uid, pid,time, number of item and user. y is the score.

But the drawback is very obvious: the test MSE is 0.34 and and train MSE is 0.07 -- overfitting is quite a deal.

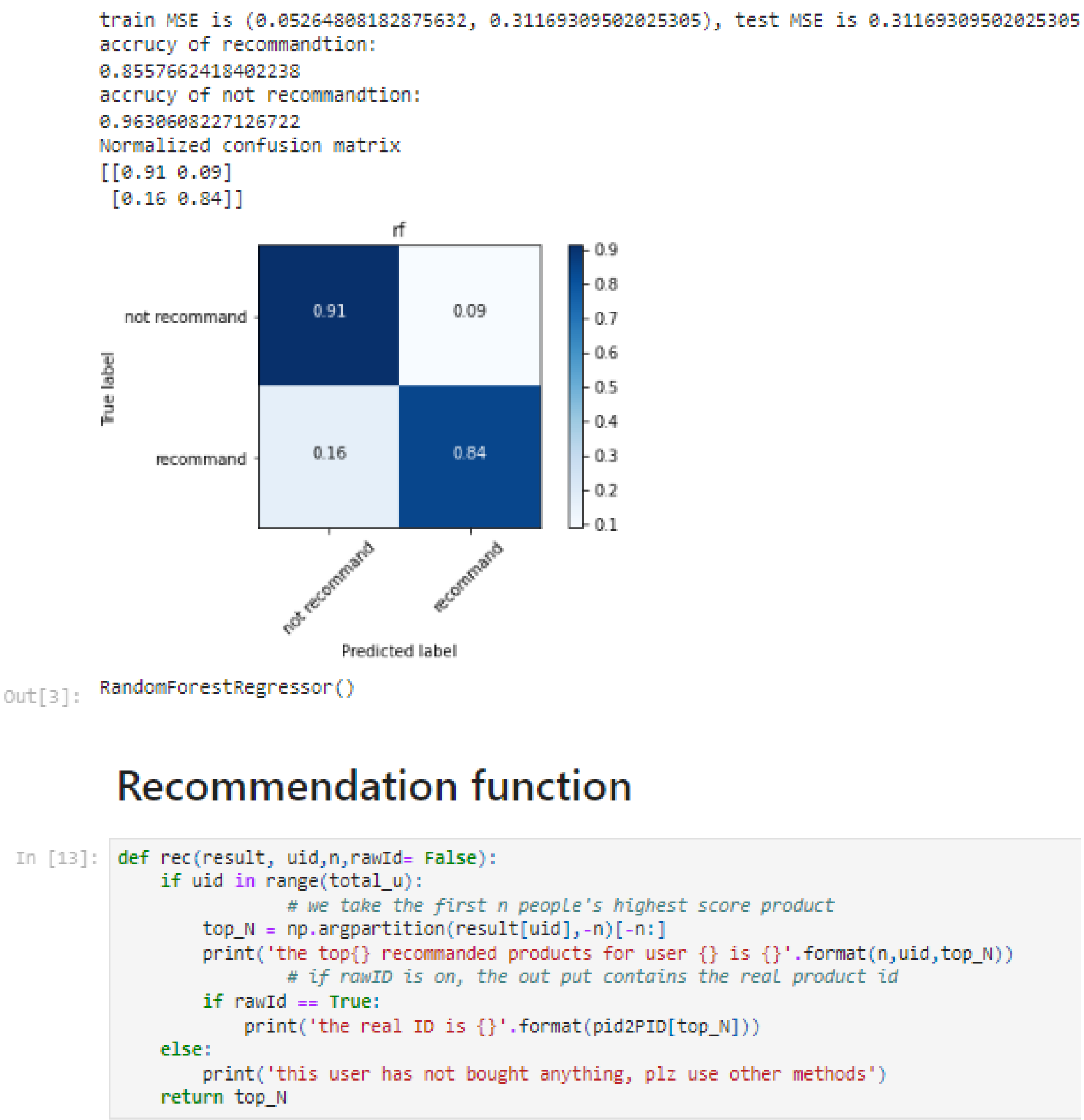


In [3]:

```
from sklearn.metrics import *
from sklearn.preprocessing import *
from sklearn.ensemble import *
def rf():
    # find X,and y
    raw_data['uid'] = pd.factorize(raw_data['UserId'])[0]
    raw_data['pid'] = pd.factorize(raw_data['ProductId'])[0]
    from sklearn.preprocessing import MinMaxScaler
    sc = MinMaxScaler()
    raw_data['time']=sc.fit_transform(raw_data['Time'].values.reshape(-1,1))
    raw_data['nuser']=sc.fit_transform(raw_data['#Users'].values.reshape(-1,1))
    raw_data['nproduct']=sc.fit_transform(raw_data['#Proudcts'].values.reshape(-1,1))

    X1 = raw_data.loc[:,['uid','pid']]
    X2 = raw_data.loc[:,['uid','pid','time']]
    X3 = raw_data.loc[:,['uid','pid','time','nuser','nproduct']]
    y = raw_data.Score

    from sklearn.model_selection import train_test_split
    X1_train,X1_test,y_train,y_test = train_test_split(X1,y,test_size=0.3,random_state=2017)
    X2_train,X2_test,y_train,y_test = train_test_split(X2,y,test_size=0.3,random_state=2017)
    X3_train,X3_test,y_train,y_test = train_test_split(X3,y,test_size=0.3,random_state=2017)
    a=RandomForestRegressor()
    a.fit(X3_train,y_train)
    y3 = a.predict(X3_test)
    sc = MinMaxScaler(feature_range=(1,5))
    c = mean_squared_error(y_train,a.predict(X3_train)), mean_squared_error(y_test,sc.fit_transform(y3.reshape(-1,1)))
    b = mean_squared_error(y_test,y3)
    print('train MSE is {}, test MSE is {}'.format(c,b))

    c3 = y3>=4
    t = y_test>=4
    print('accrucy of recommandtion:')
    print(accuracy_score(t,c3))
    c31 = y3<=1
    t1 = y_test<=1
    print('accrucy of not recommandtion:')
    print(accuracy_score(t1,c31))
    y_pred3 = []
    y_test3 = []
    for i in range(y3.shape[0]):
        if y3[i]>=4:
            y_pred3.append(1)
        elif y3[i]<4:
            y_pred3.append(0)
        # else:
            # y_pred3.append(1)

    for j in range(y3.shape[0]):
        if np.array(y_test)[j]>=4:
            y_test3.append(1)
        elif np.array(y_test)[j]<4:
            y_test3.append(0)
        # else:
            # y_test3.append(1)
```



```
            # y_test3.append(1)
    import itertools
    import matplotlib.pyplot as plt
    def plot_confusion_matrix(cm, classes,
                              normalize=False,
                              title='Confusion matrix',
                              cmap=plt.cm.Blues):
        """
        This function prints and plots the confusion matrix.
        Normalization can be applied by setting `normalize=True`.
        """
        if normalize:
            cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
            print("Normalized confusion matrix")
        else:
            print('Confusion matrix, without normalization')

        print(cm)
        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=45)
        plt.yticks(tick_marks, classes)

        fmt = '.2f' if normalize else 'd'
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
    class_names = ['not recommand','recommand']
    cnf_matrix = confusion_matrix(y_test3,y_pred3)
    np.set_printoptions(precision=2)
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                          title='rf')


    plt.show()
    return a
rf()
```



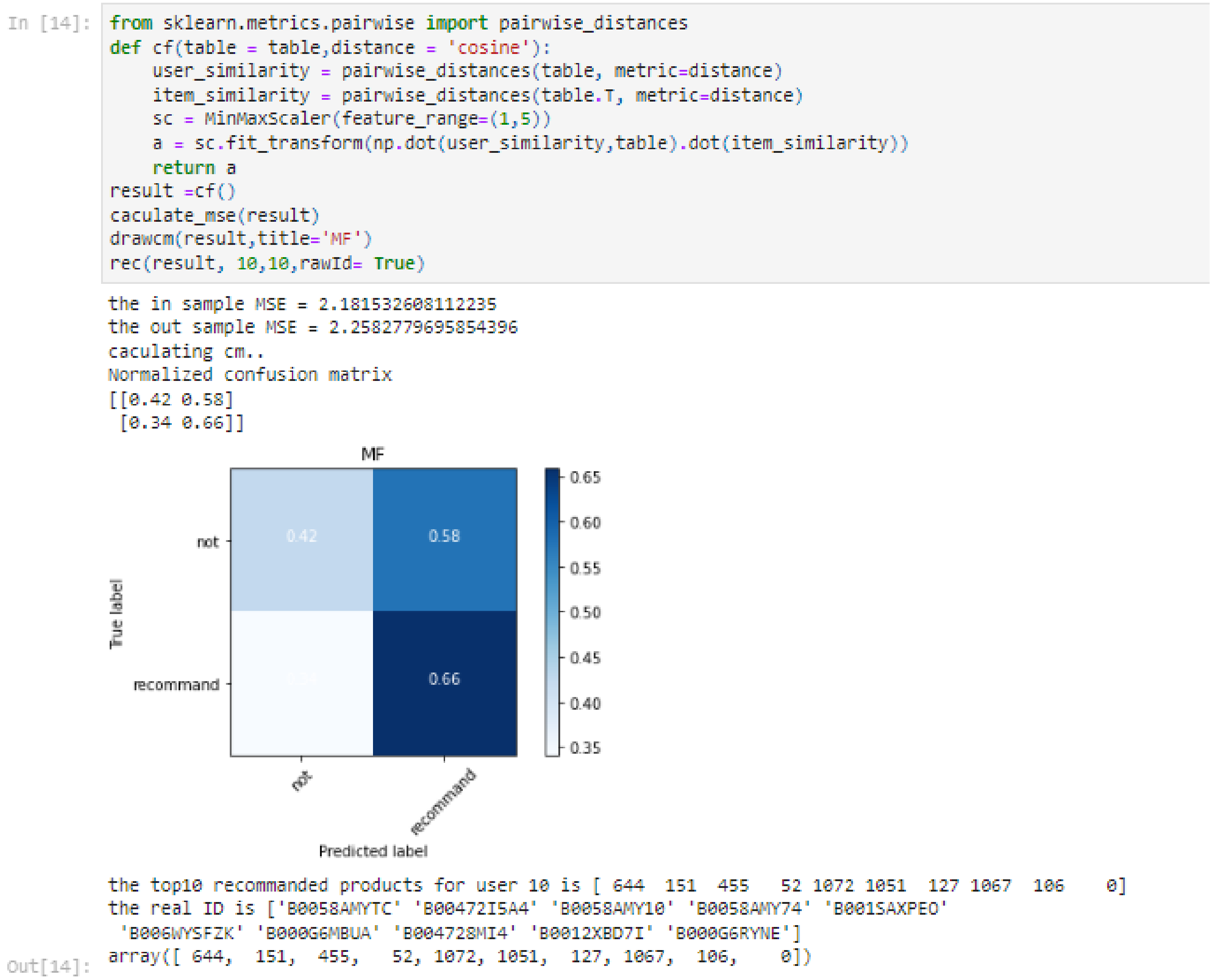
```
train MSE is (0.05264808182875632, 0.31169309502025305), test MSE is 0.31169309502025305
accrucy of recommandtion:
0.8557662418402238
accrucy of not recommandtion:
0.9630608227126722
Normalized confusion matrix
[[0.91 0.09]
 [0.16 0.84]]
```

Out[3]:
```
RandomForestRegressor()
```

Recommendation function

In [13]:
```python
def rec(result, uid,n,rawId= False):
    if uid in range(total_u):
                # we take the first n people's highest score product
        top_N = np.argpartition(result[uid],-n)[-n:]
        print('the top{} recommanded products for user {} is {}'.format(n,uid,top_N))
                # if rawID is on, the out put contains the real product id
        if rawId == True:
            print('the real ID is {}'.format(pid2PID[top_N]))
    else:
        print('this user has not bought anything, plz use other methods')
    return top_N
```

Distance Based Model



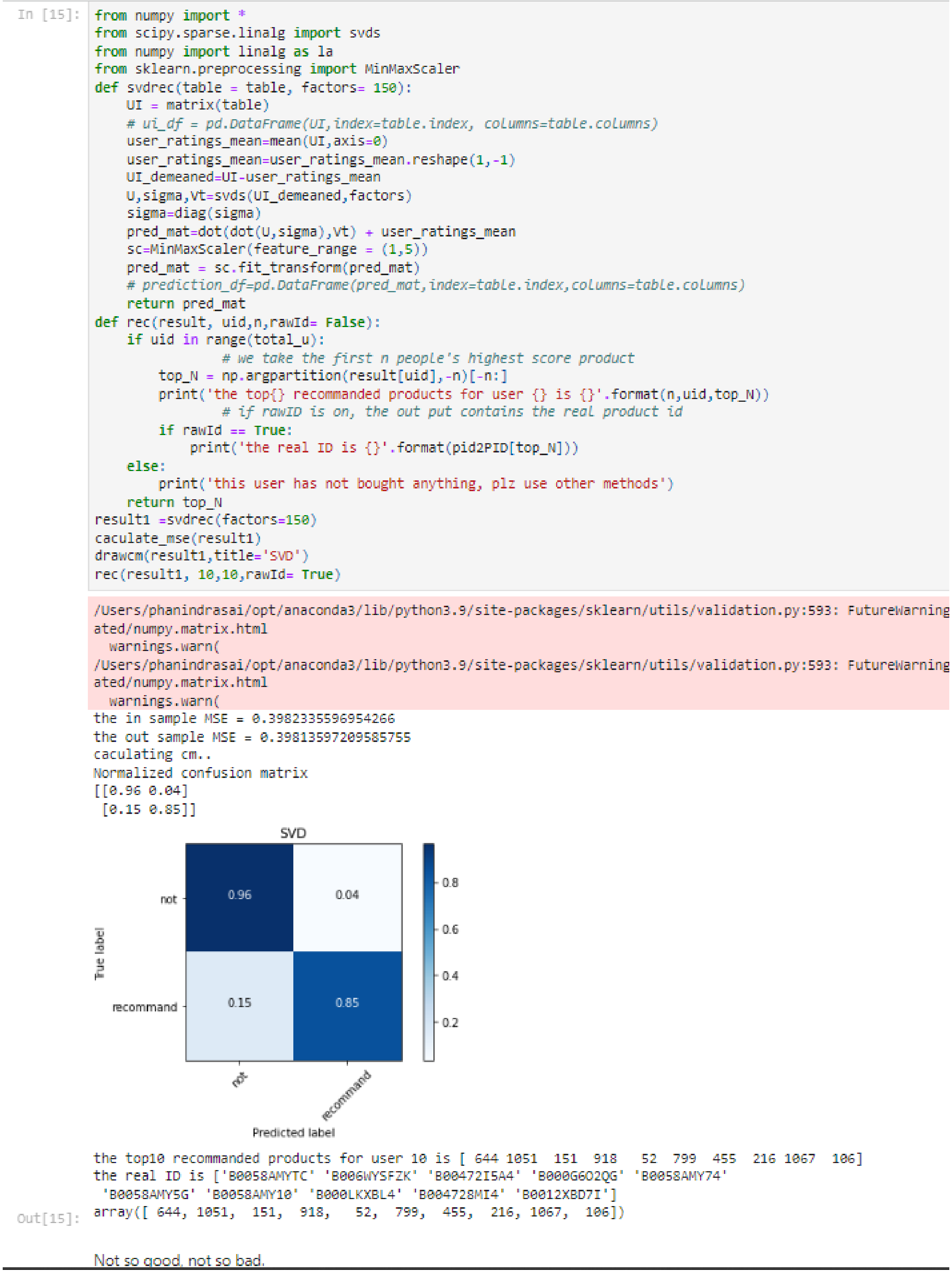
```
In [14]: from sklearn.metrics.pairwise import pairwise_distances
         def cf(table = table,distance = 'cosine'):
             user_similarity = pairwise_distances(table, metric=distance)
             item_similarity = pairwise_distances(table.T, metric=distance)
             sc = MinMaxScaler(feature_range=(1,5))
             a = sc.fit_transform(np.dot(user_similarity,table).dot(item_similarity))
             return a
         result =cf()
         caculate_mse(result)
         drawcm(result,title='MF')
         rec(result, 10,10,rawId= True)
```

```
the in sample MSE = 2.181532608112235
the out sample MSE = 2.2582779695854396
caculating cm..
Normalized confusion matrix
[[0.42 0.58]
 [0.34 0.66]]
```

```
the top10 recommanded products for user 10 is [ 644  151  455   52 1072 1051  127 1067  106    0]
the real ID is ['B0058AMYTC' 'B00472I5A4' 'B0058AMY10' 'B0058AMY74' 'B001SAXPEO'
 'B006WYSFZK' 'B000G6MBUA' 'B004728MI4' 'B0012XBD7I' 'B000G6RYNE']
Out[14]: array([ 644,  151,  455,   52, 1072, 1051,  127, 1067,  106,    0])
```

The advantage is that this method is quite interpretable. However, the drawbacks are also apparent: this algorithm is naive with a deficient performance. For a large data set, this algorithm needs m2+n2 times to calculate the similarity matrix.

SVD



In [15]:

```
from numpy import *
from scipy.sparse.linalg import svds
from numpy import linalg as la
from sklearn.preprocessing import MinMaxScaler
def svdrec(table = table, factors= 150):
    UI = matrix(table)
    # ui_df = pd.DataFrame(UI,index=table.index, columns=table.columns)
    user_ratings_mean=mean(UI,axis=0)
    user_ratings_mean=user_ratings_mean.reshape(1,-1)
    UI_demeaned=UI-user_ratings_mean
    U,sigma,Vt=svds(UI_demeaned,factors)
    sigma=diag(sigma)
    pred_mat=dot(dot(U,sigma),Vt) + user_ratings_mean
    sc=MinMaxScaler(feature_range = (1,5))
    pred_mat = sc.fit_transform(pred_mat)
    # prediction_df=pd.DataFrame(pred_mat,index=table.index,columns=table.columns)
    return pred_mat
def rec(result, uid,n,rawId= False):
    if uid in range(total_u):
                # we take the first n people's highest score product
        top_N = np.argpartition(result[uid],-n)[-n:]
        print('the top{} recommanded products for user {} is {}'.format(n,uid,top_N))
                # if rawID is on, the out put contains the real product id
        if rawId == True:
            print('the real ID is {}'.format(pid2PID[top_N]))
    else:
        print('this user has not bought anything, plz use other methods')
    return top_N
result1 =svdrec(factors=150)
caculate_mse(result1)
drawcm(result1,title='SVD')
rec(result1, 10,10,rawId= True)
```

```
/Users/phanindrasai/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/validation.py:593: FutureWarning
ated/numpy.matrix.html
  warnings.warn(
/Users/phanindrasai/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/validation.py:593: FutureWarning
ated/numpy.matrix.html
  warnings.warn(
the in sample MSE = 0.3982335596954266
the out sample MSE = 0.39813597209585755
caculating cm..
Normalized confusion matrix
[[0.96 0.04]
 [0.15 0.85]]
```

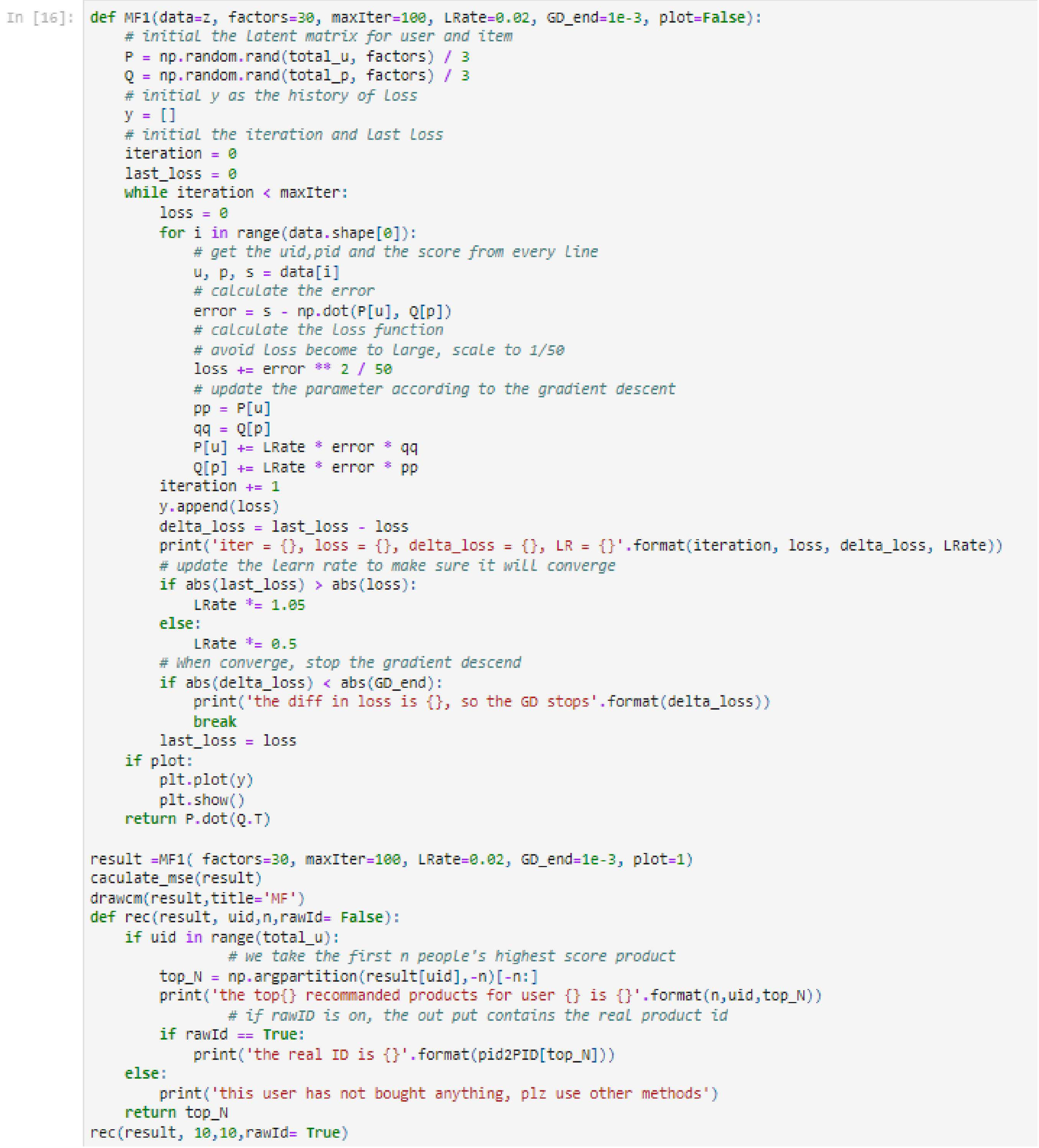
```
the top10 recommanded products for user 10 is [ 644 1051  151  918   52  799  455  216 1067  106]
the real ID is ['B0058AMYTC' 'B006WYSFZK' 'B00472I5A4' 'B000G6O2QG' 'B0058AMY74'
 'B0058AMY5G' 'B0058AMY10' 'B000LKXBL4' 'B004728MI4' 'B0012XBD7I']
```

Out[15]:

```
array([ 644, 1051,  151,  918,   52,  799,  455,  216, 1067,  106])
```

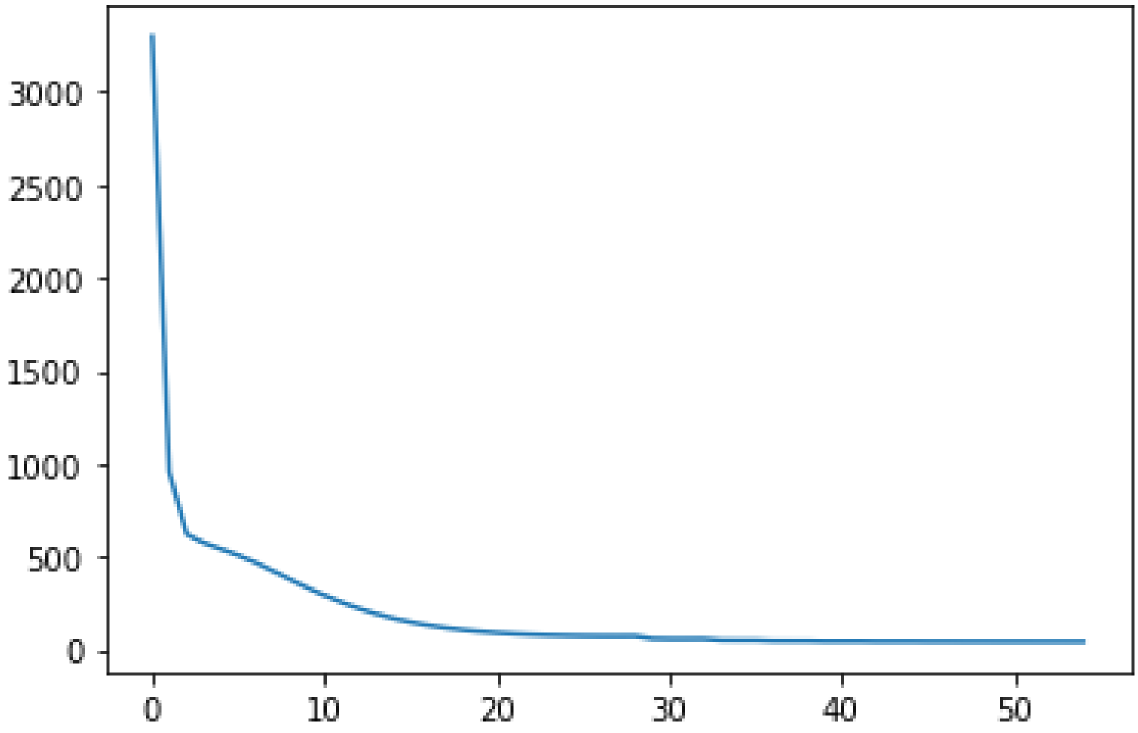
Not so good, not so bad.

Matrix Factorisation

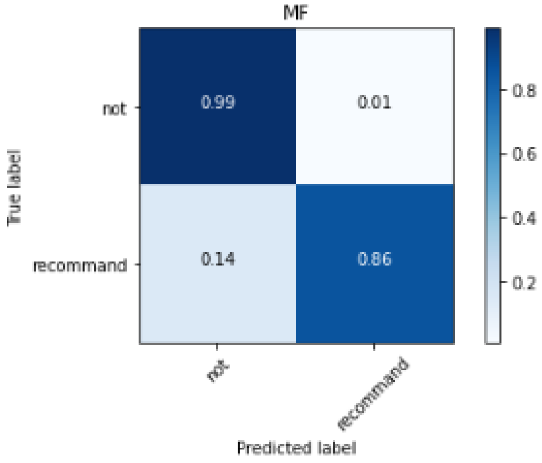


In [16]:

```
def MF1(data=z, factors=30, maxIter=100, LRate=0.02, GD_end=1e-3, plot=False):
    # initial the latent matrix for user and item
    P = np.random.rand(total_u, factors) / 3
    Q = np.random.rand(total_p, factors) / 3
    # initial y as the history of loss
    y = []
    # initial the iteration and last loss
    iteration = 0
    last_loss = 0
    while iteration < maxIter:
        loss = 0
        for i in range(data.shape[0]):
            # get the uid,pid and the score from every line
            u, p, s = data[i]
            # calculate the error
            error = s - np.dot(P[u], Q[p])
            # calculate the loss function
            # avoid loss become to large, scale to 1/50
            loss += error ** 2 / 50
            # update the parameter according to the gradient descent
            pp = P[u]
            qq = Q[p]
            P[u] += LRate * error * qq
            Q[p] += LRate * error * pp
        iteration += 1
        y.append(loss)
        delta_loss = last_loss - loss
        print('iter = {}, loss = {}, delta_loss = {}, LR = {}'.format(iteration, loss, delta_loss, LRate))
        # update the learn rate to make sure it will converge
        if abs(last_loss) > abs(loss):
            LRate *= 1.05
        else:
            LRate *= 0.5
        # When converge, stop the gradient descend
        if abs(delta_loss) < abs(GD_end):
            print('the diff in loss is {}, so the GD stops'.format(delta_loss))
            break
        last_loss = loss
    if plot:
        plt.plot(y)
        plt.show()
    return P.dot(Q.T)

result =MF1( factors=30, maxIter=100, LRate=0.02, GD_end=1e-3, plot=1)
caculate_mse(result)
drawcm(result,title='MF')
def rec(result, uid,n,rawId= False):
    if uid in range(total_u):
                # we take the first n people's highest score product
        top_N = np.argpartition(result[uid],-n)[-n:]
        print('the top{} recommanded products for user {} is {}'.format(n,uid,top_N))
                # if rawID is on, the out put contains the real product id
        if rawId == True:
            print('the real ID is {}'.format(pid2PID[top_N]))
    else:
        print('this user has not bought anything, plz use other methods')
    return top_N
rec(result, 10,10,rawId= True)
```



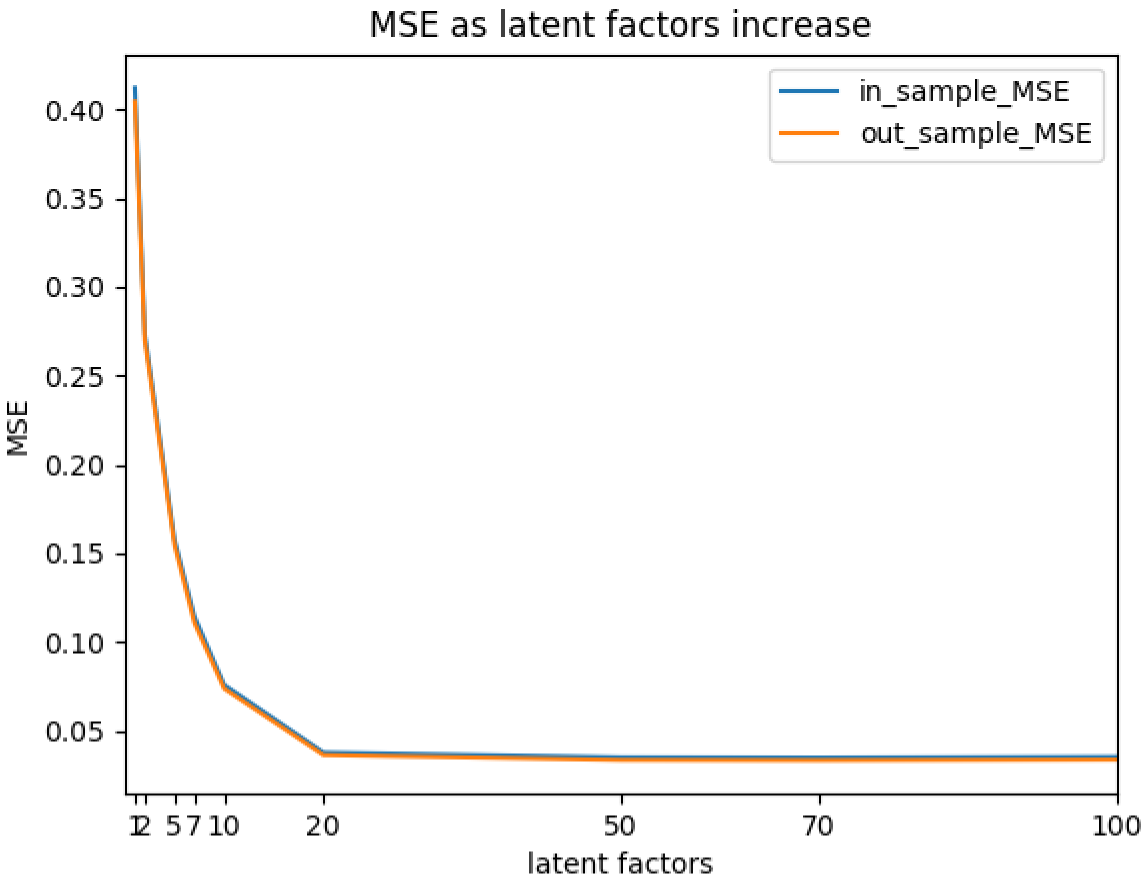
```
the in sample MSE = 0.035382086526041856
the out sample MSE = 0.03382716262595594
caculating cm..
Normalized confusion matrix
[[0.99 0.01]
 [0.14 0.86]]
```



```
the top10 recommanded products for user 10 is [ 256  996  243  729  155  201  182  595   80 1093]
the real ID is ['B004HOQE64' 'B004HOLD60' 'B001KVPBS4' 'B000ZSZ5S4' 'B001EQ4RBM'
 'B001EQ4QJK' 'B004HOLD4W' 'B004HOOZEW' 'B001EQ4P2I' 'B004HOSGWE']
Out[16]: array([ 256,  996,  243,  729,  155,  201,  182,  595,   80, 1093])
```

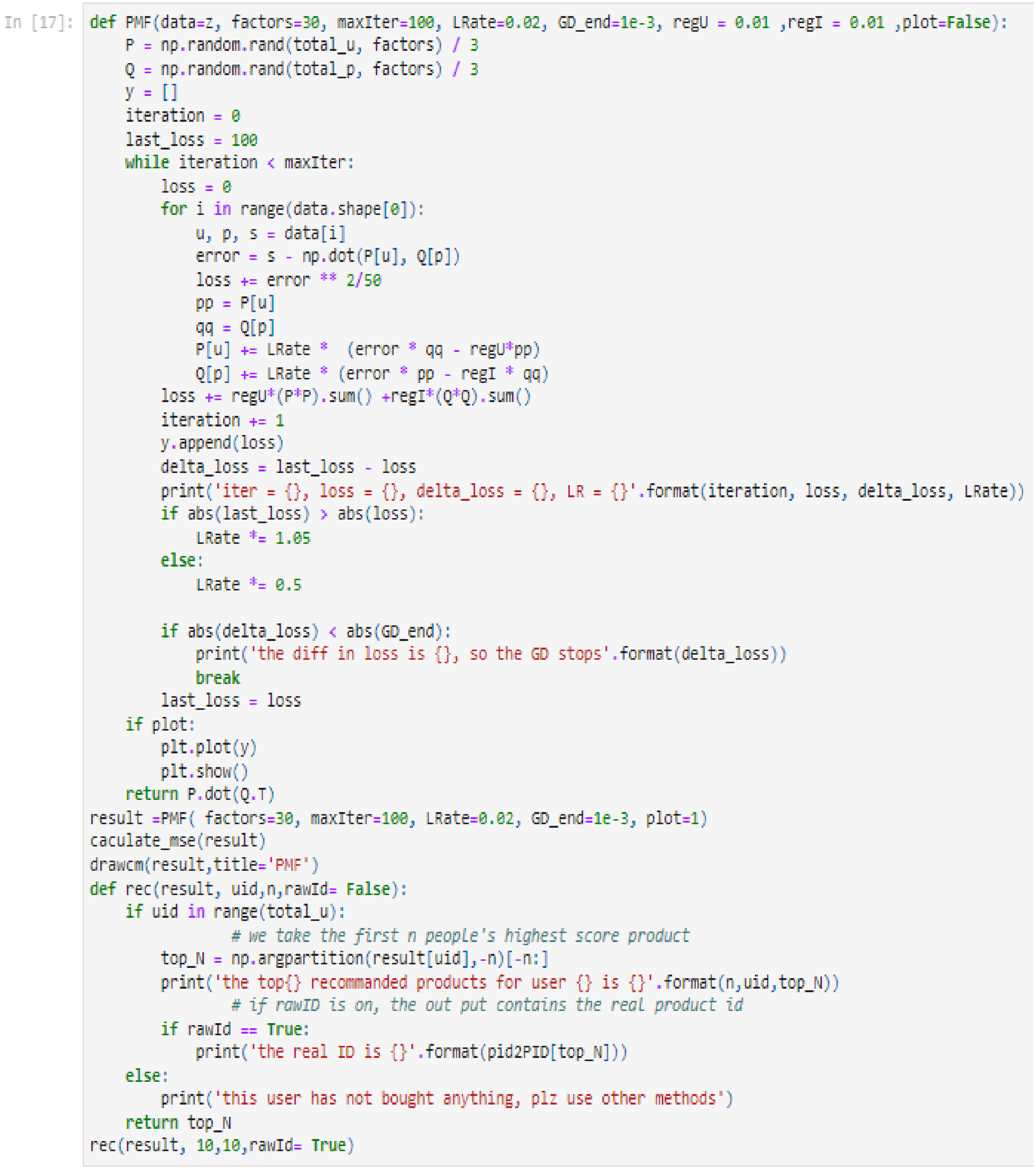
The MF gives a very good prediction: the train set MSE is 0.03542 and the test set MSE is 0.03385. While both MSEs are equally small, thus no overfitting problem occurs.

In addition, as image shows that the number of latent factors significantly affects the results.



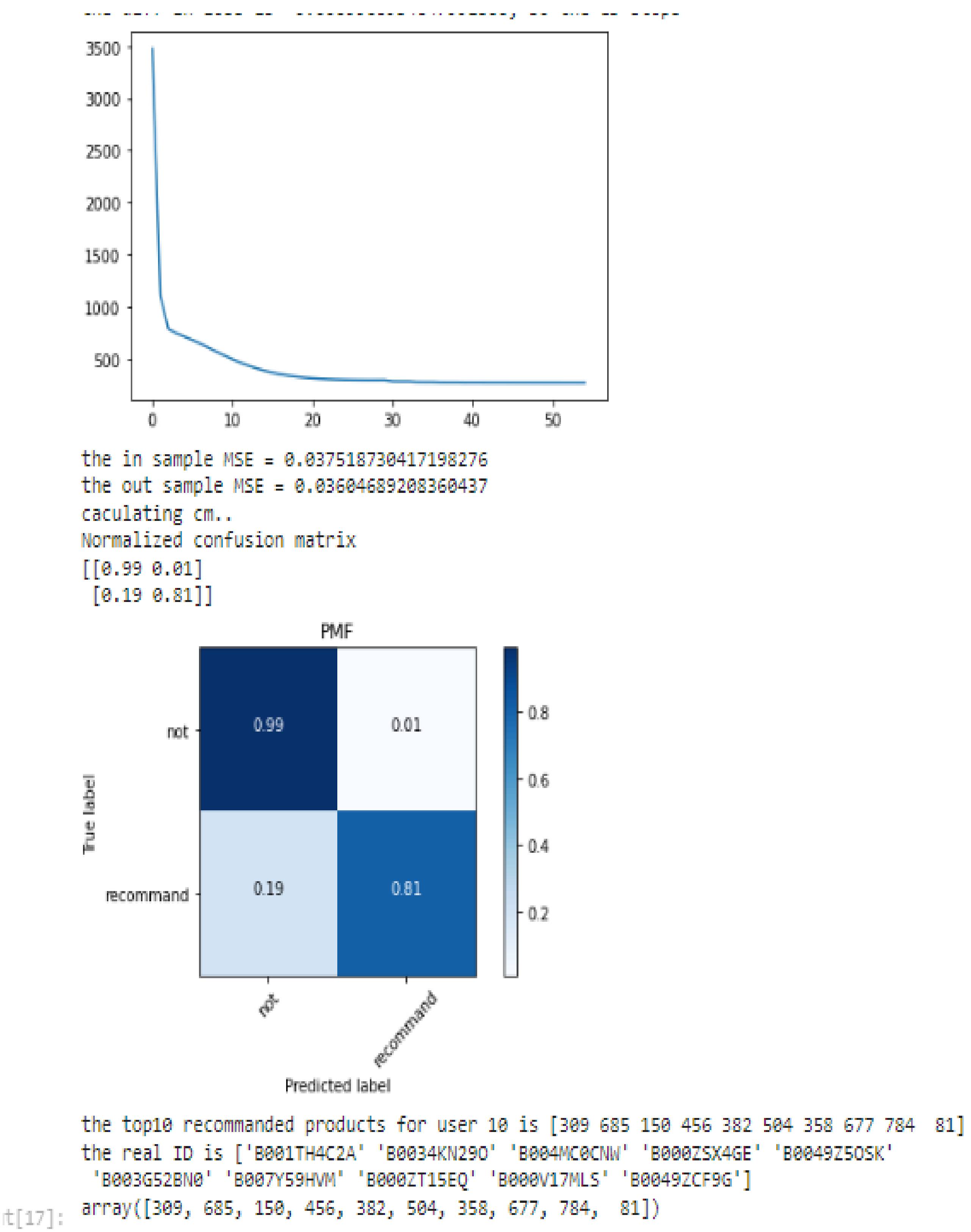
As the number of latent factors increase, the MSE decreases. And MSE converges around n=20.

Probabilistic Matrix Factorization



In [17]:

```
def PMF(data=z, factors=30, maxIter=100, LRate=0.02, GD_end=1e-3, regU = 0.01 ,regI = 0.01 ,plot=False):
    P = np.random.rand(total_u, factors) / 3
    Q = np.random.rand(total_p, factors) / 3
    y = []
    iteration = 0
    last_loss = 100
    while iteration < maxIter:
        loss = 0
        for i in range(data.shape[0]):
            u, p, s = data[i]
            error = s - np.dot(P[u], Q[p])
            loss += error ** 2/50
            pp = P[u]
            qq = Q[p]
            P[u] += LRate *  (error * qq - regU*pp)
            Q[p] += LRate * (error * pp - regI * qq)
        loss += regU*(P*P).sum() +regI*(Q*Q).sum()
        iteration += 1
        y.append(loss)
        delta_loss = last_loss - loss
        print('iter = {}, loss = {}, delta_loss = {}, LR = {}'.format(iteration, loss, delta_loss, LRate))
        if abs(last_loss) > abs(loss):
            LRate *= 1.05
        else:
            LRate *= 0.5

        if abs(delta_loss) < abs(GD_end):
            print('the diff in loss is {}, so the GD stops'.format(delta_loss))
            break
        last_loss = loss
    if plot:
        plt.plot(y)
        plt.show()
    return P.dot(Q.T)
result =PMF( factors=30, maxIter=100, LRate=0.02, GD_end=1e-3, plot=1)
caculate_mse(result)
drawcm(result,title='PMF')
def rec(result, uid,n,rawId= False):
    if uid in range(total_u):
                # we take the first n people's highest score product
        top_N = np.argpartition(result[uid],-n)[-n:]
        print('the top{} recommanded products for user {} is {}'.format(n,uid,top_N))
                # if rawID is on, the out put contains the real product id
        if rawId == True:
            print('the real ID is {}'.format(pid2PID[top_N]))
    else:
        print('this user has not bought anything, plz use other methods')
    return top_N
rec(result, 10,10,rawId= True)
```

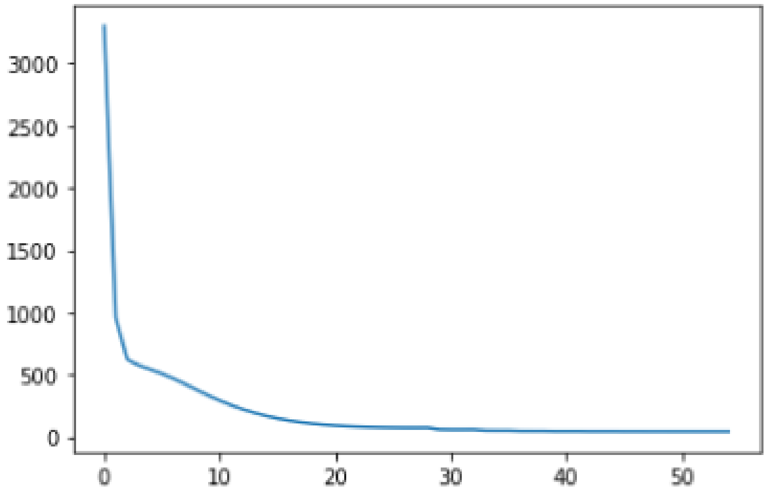


```
[illegible]
the in sample MSE = 0.037518730417198276
the out sample MSE = 0.03604689208360437
caculating cm..
Normalized confusion matrix
[[0.99 0.01]
 [0.19 0.81]]
```

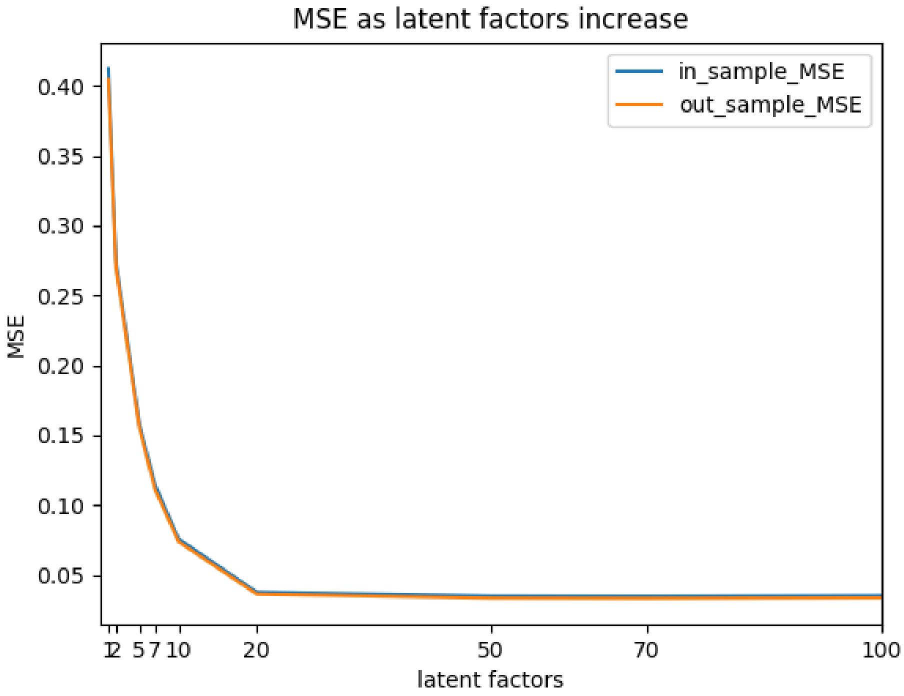
```
the top10 recommanded products for user 10 is [309 685 150 456 382 504 358 677 784  81]
the real ID is ['B001TH4C2A' 'B0034KN29O' 'B004MC0CNW' 'B000ZSX4GE' 'B0049Z5OSK'
 'B003G52BN0' 'B007Y59HVM' 'B000ZT15EQ' 'B000V17MLS' 'B0049ZCF9G']
t[17]: array([309, 685, 150, 456, 382, 504, 358, 677, 784,  81])
```

7. Model Evolution graphs

Matrix Factorisation



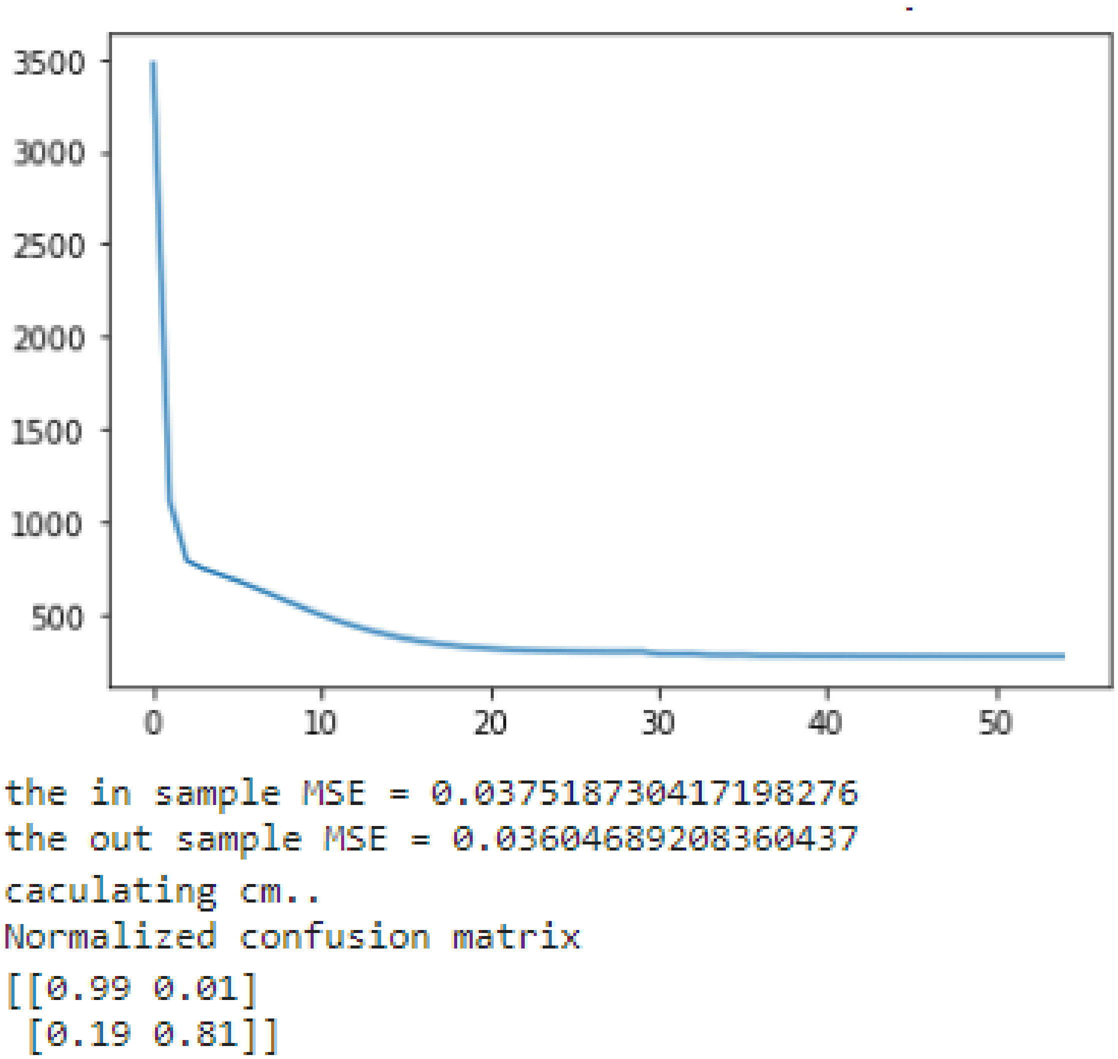
```
the in sample MSE = 0.035382086526041856
the out sample MSE = 0.03382716262595594
caculating cm..
Normalized confusion matrix
[[0.99 0.01]
 [0.14 0.86]]
```



The MF gives a very good prediction: the train set MSE is 0.03542 and the test set MSE is 0.03385. While both MSEs are equally small, thus no overfitting problem occurs.

In addition, as image shows that the number of latent factors significantly affects the results.

Probabilistic Matrix Factorization



```
the in sample MSE = 0.037518730417198276
the out sample MSE = 0.03604689208360437
caculating cm..
Normalized confusion matrix
[[0.99 0.01]
 [0.19 0.81]]
```

8.Conclusion

Model-based Collaborative Filtering is a personalised recommender system, the recommendations are based on the past behavior of the user and it is not dependent on any additional information.

The Popularity-based recommender system is non-personalised and the recommendations are based on frequency counts, which may be not suitable to the user. You can see the difference above for the user id 121 & 200, The Popularity based model has recommended the same set of 5 products to both but Collaborative Filtering based model has recommended entirely different list based on the user past purchase history

We can find that the MF and PMF did a good job. And the recommendation they make are also very reasonable.

For example, the 10th user called 'G Little Value Seeker'. Based on what he bought, the chips, chocolate and tea we recommend him Starbuck coffee and some dark chocolate -- Mr. Value Seeker has not bought these, but I am sure he will like them!

9.GitHub Repository Link

https://github.com/phanindrasai27/Amazon-food-recommendation-system

10.REFERENCES

[1] Pang, Bo, and Lillian Lee.” Opinion mining and sentiment analysis.” Foundations and trends in information retrieval 2.1-2 (2008): 1-135.

[2] Lam, Savio LY, and Dik Lun Lee.” Feature reduction for neural network-based text categorization.” Database Systems for Advanced Applications, 1999. Proceedings., 6th International Conference on. IEEE, 1999.

[3] Sundermeyer, Martin, Ralf Schlter, and Hermann Ney.” LSTM Neural Networks for Language Modeling.” INTERSPEECH. 2012.

[4] Schafer, J. Ben, et al.” Collaborative filtering recommender systems.” The adaptive web. Springer Berlin Heidelberg, 2007. 291-324.

[5] Lops, Pasquale, Marco De Gemmis, and Giovanni Semeraro. ” Content-based recommender systems: State of the art and trends.” Recommender systems handbook. Springer US, 2011. 73-105.

[6] Koren, Yehuda, Robert Bell, and Chris Volinsky. ” Matrix factorization techniques for recommender systems.” Computer 8 (2009): 30-37.

[7] Van den Oord, Aaron, Sander Dieleman, and Benjamin Schrauwen. ” Deep contentbased music recommendation.” Advances in Neural Information Processing Systems. 2013.

[8] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global Vectors for Word Representation. In EMNLP (Vol. 14, pp. 1532-1543).

[9] Christopher Olah, Understanding LSTM Networks, retrieved from http://colah.github.io/posts/2015-08- Understanding-LSTMs/

[10] Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based Recommendations with Recurrent Neural Networks. arXiv preprint arXiv:1511.06939.

[11] Chellapilla, K., Puri, S., & Simard, P. (2006, October). High performance convolutional neural networks for document processing. In Tenth International Workshop on Frontiers in Handwriting Recognition. Suvisoft.

[12] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167. Chicago

[13] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. The Journal of Machine Learning Research, 13(1), 281-305. Chicago

[14] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[15] Derks, E. P. P. A., Pastor, M. S., & Buydens, L. M. C. (1995). Robustness analysis of radial base function and multi-layered feed-forward neural network models. Chemometrics and Intelligent Laboratory Systems, 28(1), 49-60.