

# INTRODUCTION

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you stating that you might also like this.

## THEORY BEHIND RECOMMENDATION SYSTEM

Here is the theory behind the Product Recommendation system, and the process of how we are going to implement it. If we know properly then implementation in python would be easy. Recommendation system can be implemented using two methods.

### 1. Content based filtering

- a. In content-based filtering, the similarity between different products is calculated on the basis of the attributes of the products. For example, in movie recommendation system, movie type, star cast, movie name, director and producer of that movie etc.
- b. Content-based filtering suggests products to customers by using the characteristics of an item they have selected in the past in order to recommend additional items with similar properties.

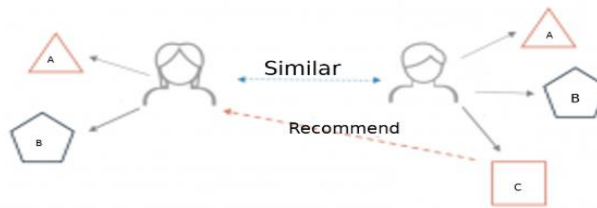
### 2. Collaborative Filtering

- a. Collaborative filtering is a method of recommending products to customers by using their past behaviors or product ratings, as well as similar decisions by other customers to predict which items might be appealing to the original customers.

We will dive into the collaborative filtering method as it is much effective because that considers the impact of different users on the product selectivity.

## What is collaborative filtering?

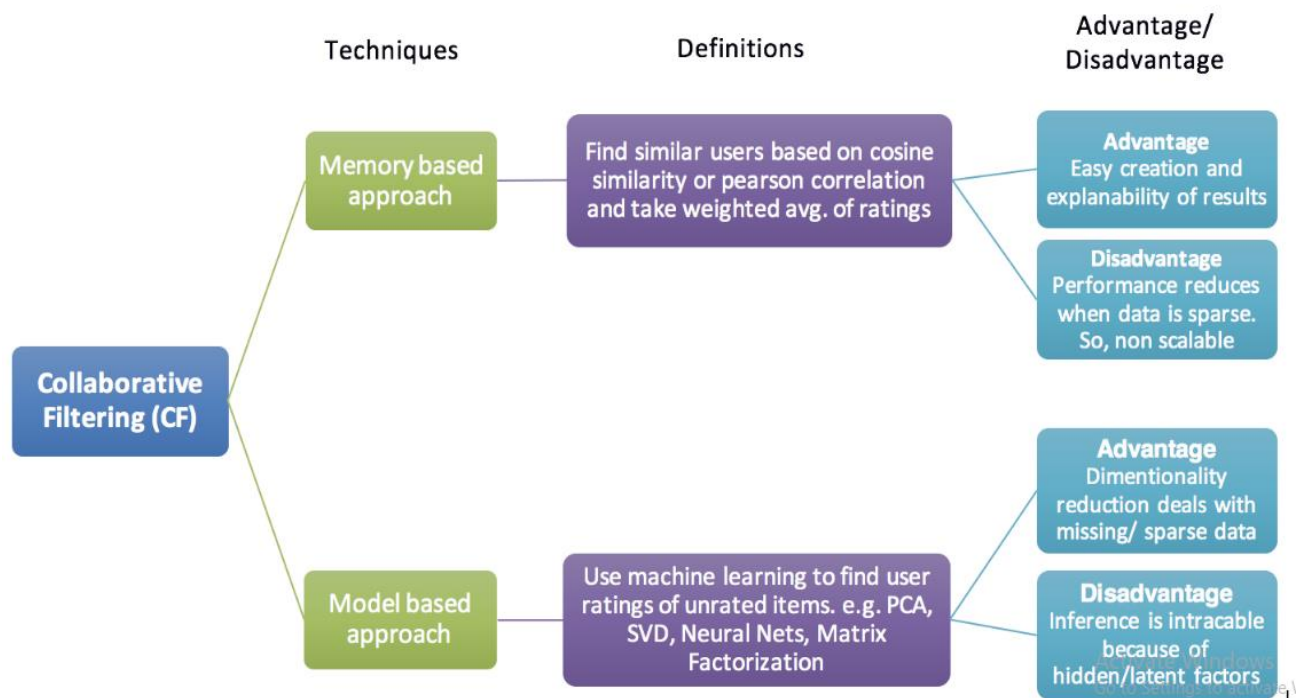
Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users. It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user. It looks at the items they like and combines them to create a ranked list of suggestions.



This type of intelligent recommender systems that can learn to give better recommendations as more information about users is collected.

## Types of Collaborative filtering

1. Memory Based
2. Model Based

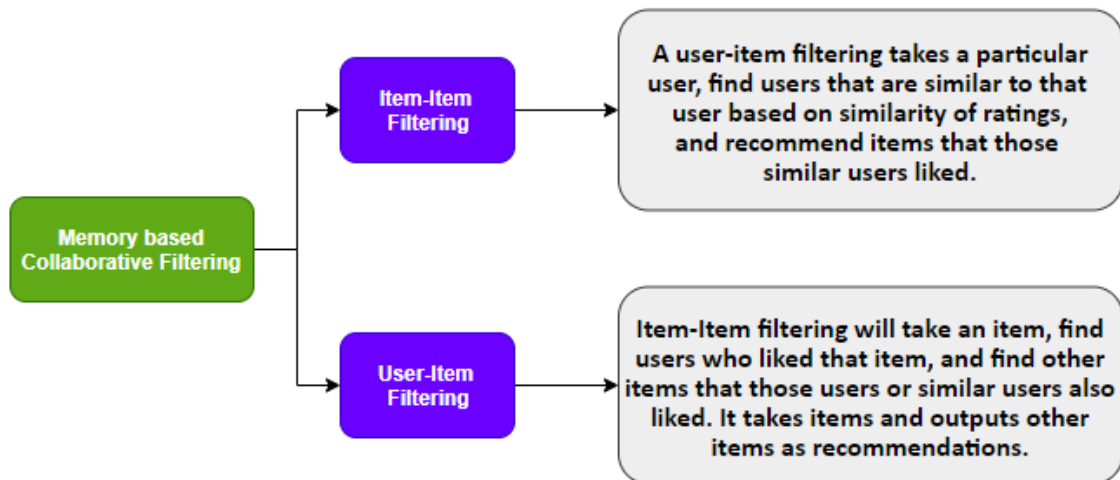


## Memory Based Collaborative Filtering

Memory-Based Collaborative Filtering approaches can be divided into two main sections

1. User-item filtering
2. Item-item filtering ( Developed by Amazon )

A user-item filtering takes a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked. In contrast, item-item filtering will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations.



Item-Item Collaborative Filtering → Users who liked this item also liked

User-Item Collaborative Filtering → Users who are similar to you also liked

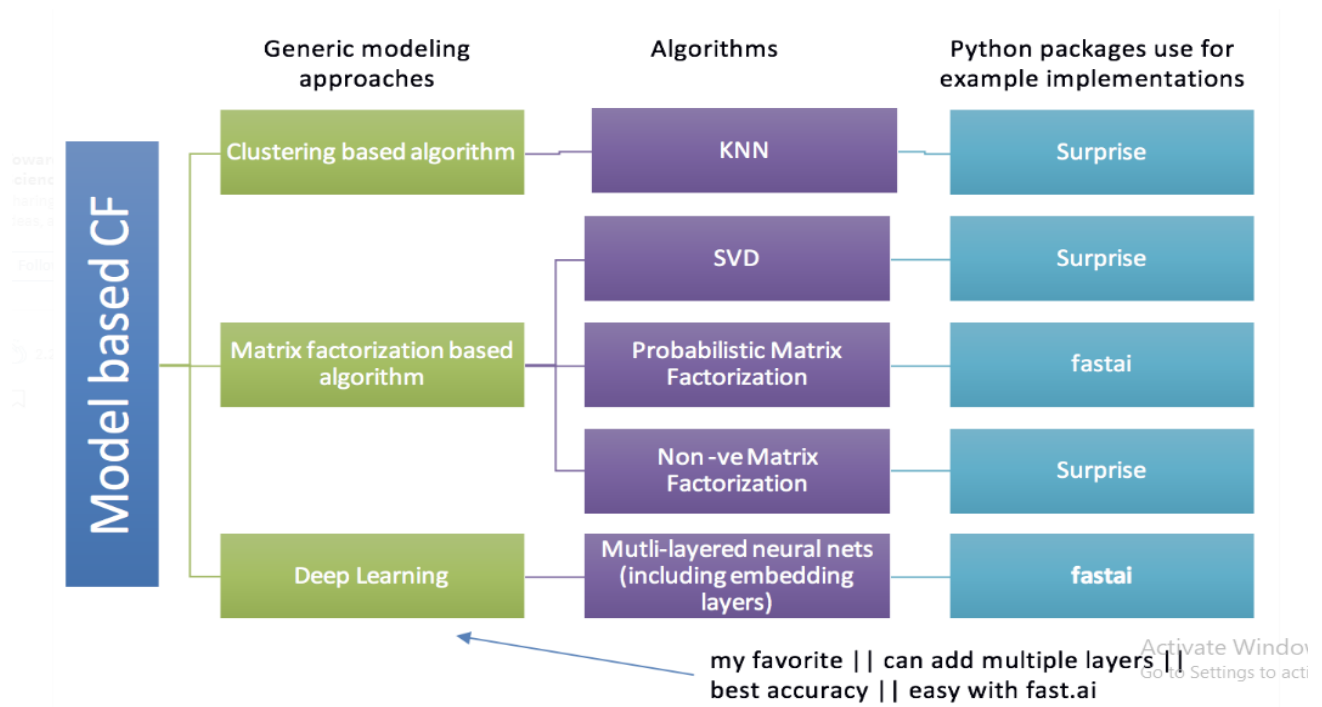
Here, the closest user or items are calculated only by using **Cosine similarity or Pearson correlation coefficients**, which are only based on arithmetic operations. Even, non-parametric ML approaches like KNN (clustering) will also come under Memory based approach.

**As no training or optimization is involved, it is an easy to use approach. But its performance decreases when we have sparse data which hinders scalability of this approach.**

## Model Based Collaborative Filtering

In this type of content filtering technique, models are developed using machine learning algorithms to predict user's rating of unrated items. As per my understanding, the algorithms in this approach can further be broken down into 3 sub-types.

1. Clustering based Algorithms
2. Deep Learning
3. Matrix Factorization



### Clustering based algorithm (KNN)

Here, in this approach similarities are calculated based on an unsupervised learning model, rather than Pearson correlation or cosine similarity. In this approach, we also limit the number of similar users as  $k$ , which makes system more scalable.

### Neural Nets/ Deep Learning

There is lack on online material to learn how to use deep learning models for collaborative filtering. This is something that I learnt in which can be implemented using fast.ai library in python.

### Matrix Factorization Based Algorithms

So, in this collaborative filtering Method we do not need to know much about the products or the customers. We just need to know need to know how many unique products and customers there as well as how the customers rated the products. This information is stored in a user-item table, where the ratings are the entries in the table. The representation is the matrix which is user as row and items as column where each user and item have been assigned a unique integer value.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$
$u_1$	5		4	1	
$u_2$		3		3	
$u_3$		2	4	4	1
$u_4$	4	4	5		
$u_5$	2	4		5	2

Rating Matrix

The general rating matrix  $R$  is in  $R\{ \#(\text{users}) \times \#(\text{items}) \}$  where  $\#(\text{users})$  is the number of users and  $\#(\text{items})$  is the number of items. Let  $R\{u, i\}$  entry correspond to the  $u$  row and  $i$  column of the matrix  $R$ . Then  $R\{u, i\}$  then the rating by user  $u$  on item  $i$ . The entries that have missing values in the matrix will be filled in with 0's.

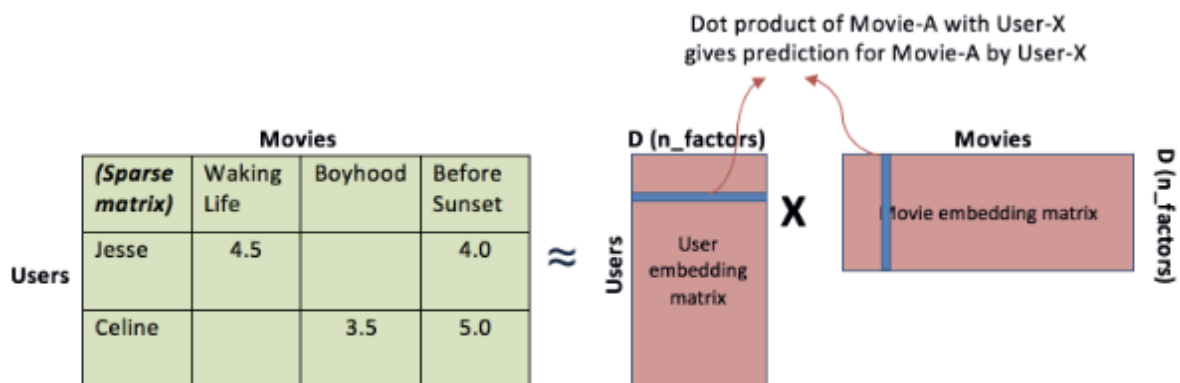
	Feature 1	Feature 2
User 1	?	?
User 2	?	?
User 3	?	?
User 4	?	?
User 5	?	?

$\times$

	Item 1	Item 2	Item 3	Item 4	Item 5
Feature 1	?	?	?	?	?
Feature 2	?	?	?	?	?

$=$

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0?	3	0?	3	0?
User 2	4	0?	0?	2	0?
User 3	0?	0?	3	0?	0?
User 4	3	0?	4	0?	3
User 5	4	3	0?	4	0?



From above two diagrams, we can understand embeddings as low dimensional hidden factors for items and users. One can consider the embeddings as the features to be considered while making the predictions. Let say Users has  $D(n\_factors)$  which has different likings or choices or taste for movies and for movies matrix has also  $D(n\_factors)$  regarding the movies like how recent movie is or is it action movie or drama, likewise. In above figure, a higher number from dot product of user-X and movie-A matrix means that movie-A is a good recommendation for user-X.

## Algorithms for Matrix Factorization

1. **Singular Value Decomposition (SVD/PCA)**
2. **Probabilistic factorization (PMF)**
3. **Non-negative factorization (NMF)**

For SVD or PCA, we decompose our original sparse matrix into product of 2 low rank orthogonal matrices. The idea behind such models is that attitudes or preferences of a user can be determined by a small number of hidden factors. It is called as Embeddings.

## How to find Accuracy?

There are two Measures.

1. **Root Mean Square Error (RMSE)**
  - Predict the ratings for a test dataset of user-item pairs, where actual rating value is already known.
  - Find the difference between known value and the predicted value for the test set and then find the Mean
  - Take the Square root of that average to get RMSE
2. **Mean Absolute Error (MAE)**
  - Find the magnitude of error by its absolute value
  - Take the mean of all the error values.

## Dataset Selection

We have selected a dataset from the UCSD site as shown below out of more than 20 product categories for this project.

<http://jmcauley.ucsd.edu/data/amazon/links.html>

## Files

### "Small" subsets for experimentation

If you're using this data for a class project (or similar) **please** consider using one of these smaller datasets below before requesting the larger files. To obtain the larger files you will need to [contact me](#) to obtain access.

**K-cores** (i.e., dense subsets): These data have been reduced to extract the **k-core**, such that each of the remaining users and items have k reviews each.

**Ratings only:** These datasets include no metadata or reviews, but only (user,item,rating,timestamp) tuples. Thus they are suitable for use with [mymedialite](#) (or similar) packages.

Books	5-core (8,898,041 reviews)	ratings only (22,507,155 ratings)
Electronics	5-core (1,689,188 reviews)	ratings only (7,824,482 ratings)
Movies and TV	5-core (1,697,533 reviews)	ratings only (4,607,047 ratings)
CDs and Vinyl	5-core (1,097,592 reviews)	ratings only (3,749,004 ratings)
Clothing, Shoes and Jewelry	5-core (278,677 reviews)	ratings only (5,748,920 ratings)
Home and Kitchen	5-core (551,682 reviews)	ratings only (4,253,926 ratings)
Kindle Store	5-core (982,619 reviews)	ratings only (3,205,467 ratings)
Sports and Outdoors	5-core (296,337 reviews)	ratings only (3,268,695 ratings)
Cell Phones and Accessories	5-core (194,439 reviews)	ratings only (3,447,249 ratings)
Health and Personal Care	5-core (346,355 reviews)	ratings only (2,982,326 ratings)
Toys and Games	5-core (167,597 reviews)	ratings only (2,252,771 ratings)
Video Games	5-core (231,780 reviews)	ratings only (1,324,753 ratings)
Tools and Home Improvement	5-core (134,476 reviews)	ratings only (1,928,047 ratings)
Beauty	5-core (198,502 reviews)	ratings only (2,023,070 ratings)
Apps for Android	5-core (752,937 reviews)	ratings only (2,638,172 ratings)
Office Products	5-core (53,258 reviews)	ratings only (1,243,186 ratings)
Pet Supplies	5-core (157,836 reviews)	ratings only (1,235,316 ratings)
Automotive	5-core (20,473 reviews)	ratings only (1,373,768 ratings)
Grocery and Gourmet Food	5-core (151,254 reviews)	ratings only (1,297,156 ratings)
Patio, Lawn and Garden	5-core (13,272 reviews)	ratings only (993,490 ratings)
Baby	5-core (160,792 reviews)	ratings only (915,446 ratings)
Digital Music	5-core (64,706 reviews)	ratings only (836,006 ratings)
Musical Instruments	5-core (10,261 reviews)	ratings only (500,176 ratings)
Amazon Instant Video	5-core (37,126 reviews)	ratings only (583,933 ratings)

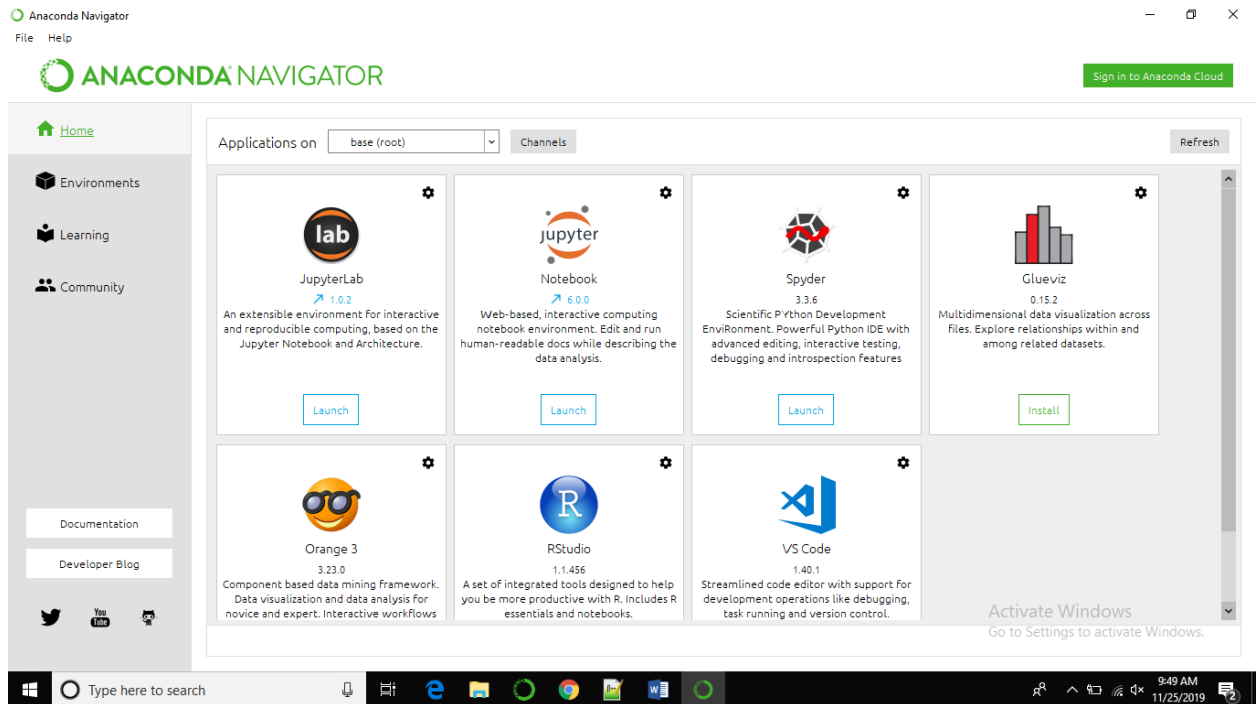
On that site there are two types of data available in "Small Subset for experiment" section.

- K-Cores : These are the reduced data such that each of the users and items have K reviews each.
- Rating Only: These dataset contains only 4 columns (user, item, rating, timestamp) per tuples.

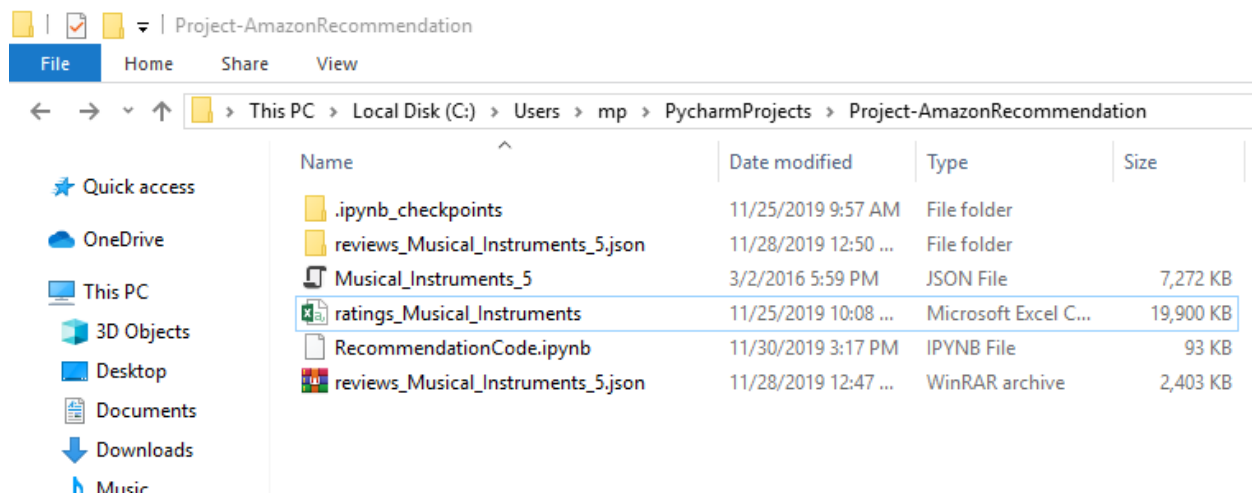
As our Personal laptop doesn't have high configurations, we chose smaller dataset of Musical Instruments.

# Software Setup

We have used Anaconda environment where code is written in the Jupyter Notebook.



Screenshot of local directory where all files stored.





## Tools and Libraries Used

While recommender systems can be built in many different languages, I elected to use Python to build this one. I use a few different popular tools and libraries to complete this project

- Jupyter Notebook 5.3.1 with Python 3.6.2
- **pandas**
- **scikit-surprise**
- numpy
- matplotlib
- collections

## Code and intermediate Results

Below are the libraries and methods which we used to implement the recommendations system.

```
In [130]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from surprise import accuracy
from surprise import Dataset
from surprise import Reader

from surprise.model_selection import train_test_split
from surprise.prediction_algorithms.baseline_only import BaselineOnly
from surprise import SVD, evaluate
from surprise.model_selection import cross_validate
from collections import defaultdict
```

Below code is responsible for reading data from JSON file which we downloaded from the UCSD site. Sample data Set given on that site. I have downloaded that JSON file and get it read from my laptop. This code also prints the shape of the data as well as the Description of the data with column names.

(Next Page)

```

def importdatafromJSON():
    #file_handler = open("ratings_Musical_Instruments.csv", "r")
    #data = pd.read_csv(file_handler, sep = ",")
    #file_handler.close()
    df = pd.read_json("Musical_Instruments_5.json", lines=True)
    print("-----Columns Name-----")
    print(" ")
    print(df.columns)
    print(" ")
    print("-----Columns Info-----")
    print(" ")
    print(df.info())
    print(" ")
    print("-----Shape Of Data-----")
    print(" ")
    print(df.shape)
    print(" ")
    #print(df['overall'].min())
    #print(df['overall'].max())
    return df

```

Once Data get imported, we need to preprocess the data. Following code checks whether one customer has given more than one review to a specific product or not.

```

def checkForDuplicateReviews(data):
    #checking for the IF a user has given tow reviews to a perticular product
    purchase_ids = ['asin', 'reviewerID']
    duplicates = data[data.duplicated(subset=purchase_ids, keep=False)].sort_values(purchase_ids)
    print("-----Checking duplicated reviews-----")
    print(" ")
    duplicates.head(10)
    print("-----END-----")
    print(" ")

```

Following code gives the information about the Dataset. Which products are reviewed Maximum time as well as the minimum time?

```

def MinMaxTimeReviewedProducts(data):
    print("-----Single product reviewed MAX times-----")
    print(data.asin.value_counts().to_frame().head(5))
    print(" ")
    print("-----Single product reviewed MIN times-----")
    print(data.asin.value_counts().to_frame().tail(5))
    print(" ")

```

Below function gives Review Counts and its respective Frequencies.

```
def ReviewCount_VS_Freq(data):
    print("-----Matrix of Review Count VS Frequency-----")
    print(" ")
    print((data.reviewerID.value_counts().rename_axis('id').reset_index(name='frequency').frequency.value_counts(
    print(" ")
    print("-----")
    print(" ")
```

Now we need to split the data into Training and Testing set. I have done many trial and error and find the accuracy for those partition and found that 80% - 20% partition gives better results.

```
def split(data):
    # Setting the rating scale
    read = Reader(rating_scale=(1, 5))
    # Load data from the review data frame and create train and test sets
    dataset = Dataset.load_from_df(data[['reviewerID', 'asin', 'overall']], read)
    #splitting the data with 80% and 20% ratio
    trainset, testset = train_test_split(dataset, test_size=.20)
    return dataset, trainset, testset
```

Here we have counted measured accuracy for this dataset using Baseline Estimates. These are the basic algorithms that do not do much work but that are still useful for comparing accuracies.

Here  $R(u \times i)$  is generated based on the following formulas.

The prediction  $\hat{r}_{ui}$  is generated from a normal distribution  $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$  where  $\hat{\mu}$  and  $\hat{\sigma}$  are estimated from the training data using Maximum Likelihood Estimation:

$$\hat{\mu} = \frac{1}{|R_{train}|} \sum_{r_{ui} \in R_{train}} r_{ui}$$
$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{train}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{train}|}}$$

```
def Recommendation_baseline(trainset, testset):
    print("-----ACCURACY using Baseline-----")
    bsl_options = {'method': 'als'} #with default parameters 'n_epochs': 10, 'reg_u': 15, 'reg_i': 10
    algo_baseline = BaselineOnly(bsl_options=bsl_options)
    fit_baseline = algo_baseline.fit(trainset)
    predictions_baseline = fit_baseline.test(testset)
    print("Baseline Prediction Accuracy : " , accuracy.rmse(predictions_baseline, verbose=False))
```

For our actual prediction method, we have followed Model based Matrix Factorization Method, where we have used SVD as the algorithms. Means for doing matrix factorization Singular Value decomposition method will be used. We have explained the theory in the first few pages of the Report.

```
def Recommendation_MatrixFact_SVD(trainset, testset, dataset):
    algo_svd = SVD() #with default parameters 'n_epochs=20, lr_all=0.005, reg_all=0.02
    fit_svd = algo_svd.fit(trainset)
    predictions = fit_svd.test(testset)
    print("Matrix Factorization SVD Prediction Accuracy : " , accuracy.rmse(predictions, verbose=False))
    print(" ")
    print("-----ACCURACY using SVD-----")
    print(" ")
    #print("Evaluate: ", evaluate(algo_svd, dataset, measures=['RMSE']))
    print(cross_validate(algo_svd, dataset, measures=['RMSE', 'MAE'], cv=5, verbose=True))
    print(" ")
    return predictions
```

Now it's time to get the recommendation for the customers who reviewed a product or products. Following code/function gets the test set data and predicts on the bases of predicted test data. And prints the results and also returns the result to the main function.

But sometimes due to sparse data, or may be lack of data like... a product have only one review, and a customer has reviewed only one product then it becomes difficult for system to product the other product. Here the output from the above method might have less than 5 recommendation due to this problem.

(Next page)

```

def get_recommendations(predictions,n):
    # First map the predictions to each user.
    top_n = defaultdict(list)
    for uid, iid, r_ui , 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]
    print(" ")
    print("-----FOR SPECIFIC USER-----")
    print(" ")
    print(top_n['A1L7M2JXN4EZCR'])
    print(" ")
    print("-----FOR ALL USER-----")
    print(" ")
    for uid, user_ratings in top_n.items():
        print(uid, "\t", [ iid for (iid, est) in user_ratings])
    print(" ")
    return top_n

```

To prevent the issue which I mentioned, first I found the most reviewed data, created the list and which user has less than 5 reviews, I appended the most reviewed data at the end.

```

def recommendation_list(user_list, user_predictions, item_list):
    recommendations = {}
    for i in range(100):
        user = user_list[i]
        if user in user_predictions:
            user_recs = [user_predictions[user][i][0] for i in range(len(user_pre
            if user_recs:
                num_items = len(user_recs)
            else:
                num_items = 0

            idx = 0
            while num_items < 10:
                product = item_list[idx]
                if product not in user_recs:
                    user_recs.append(product)
                    num_items = len(user_recs)
                idx += 1
            recommendations.update({user: user_recs})
    return recommendations

```

Below code find the most reviewed products and sort it in descending order.

```
review_count = data.asin.value_counts()
review_count_ten = review_count[review_count >= 60]
hundred_reviews = data[data.asin.isin(review_count_ten.index)]
items = (hundred_reviews[['asin', 'overall']].groupby('asin').agg('mean').sort_values('overall', ascending=False))
t = data.reviewerID.unique().tolist()
#print("Number of Unique Customer/Reviewer In Test Set : ", len(t))
```

Here is the main method, which call all the other functions one by one and gives the result in the sequence.

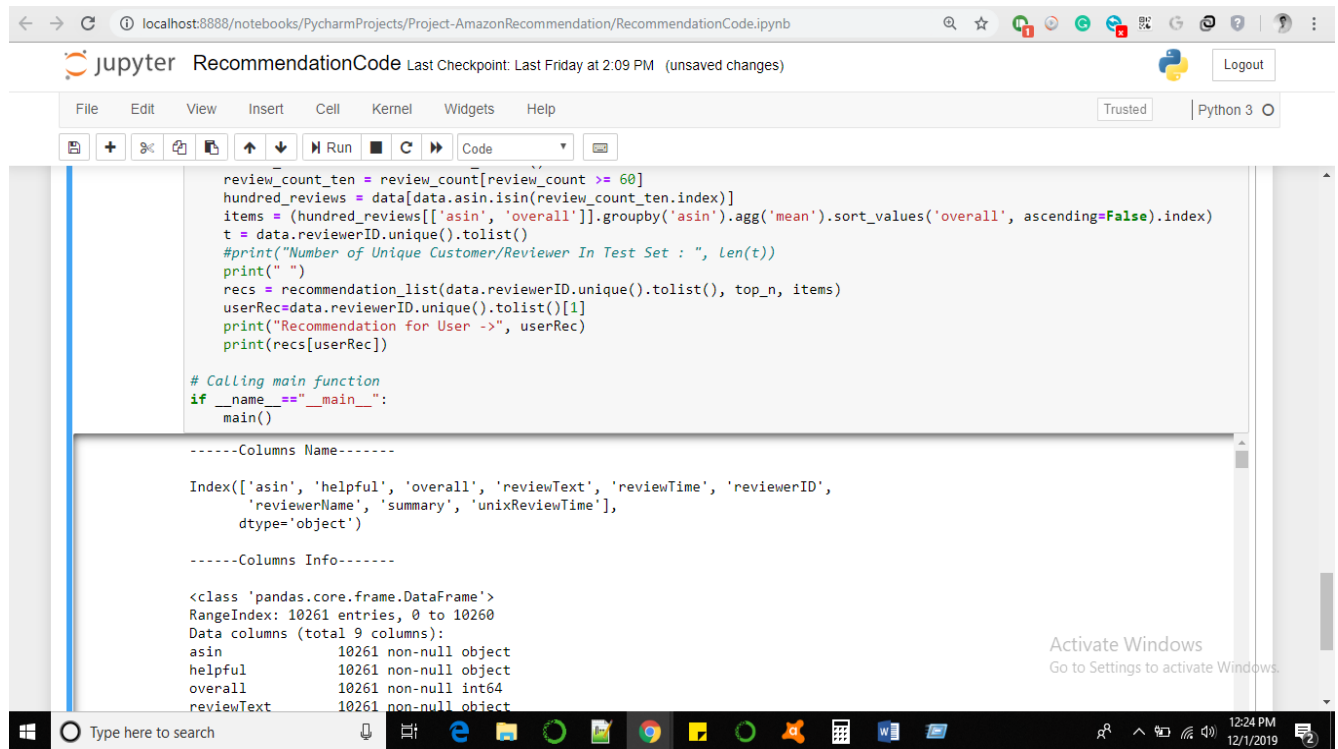
```
def main():
    data = importdatafromJSON()

    checkForDuplicateReviews(data)
    MinMaxTimeReviewedProducts(data)
    ReviewCount_VS_Freq(data)
    dataset, trainset, testset = split(data)
    Recommendation_baseline(trainset, testset)
    predictions = Recommendation_MatrixFact_SVD(trainset, testset, dataset)
    top_n = get_recommendations(predictions, n=200)

    review_count = data.asin.value_counts()
    review_count_ten = review_count[review_count >= 60]
    hundred_reviews = data[data.asin.isin(review_count_ten.index)]
    items = (hundred_reviews[['asin', 'overall']].groupby('asin').agg('mean').sort_values('overall', ascending=False))
    t = data.reviewerID.unique().tolist()
    #print("Number of Unique Customer/Reviewer In Test Set : ", len(t))
    print(" ")
    recs = recommendation_list(data.reviewerID.unique().tolist(), top_n, items)
    userRec = data.reviewerID.unique().tolist()[1]
    print("Recommendation for User ->", userRec)
    print(recs[userRec])
```

# OUTPUT

## Screenshot:



The screenshot shows a Jupyter Notebook titled 'RecommendationCode' running on a local host. The code in the notebook defines a function to filter reviews based on a count and then calls it. The console output displays the columns of the resulting DataFrame and its information.

```
review_count_ten = review_count[review_count >= 60]
hundred_reviews = data[data.asin.isin(review_count_ten.index)]
items = (hundred_reviews[['asin', 'overall']].groupby('asin').agg('mean').sort_values('overall', ascending=False).index)
t = data.reviewerID.unique().tolist()
#print("Number of Unique Customer/Reviewer In Test Set : ", Len(t))
print(" ")
recs = recommendation_list(data.reviewerID.unique().tolist(), top_n, items)
userRec=data.reviewerID.unique().tolist()[1]
print("Recommendation for User ->", userRec)
print(recs[userRec])

# Calling main function
if __name__=="__main__":
    main()
```

```
-----Columns Name-----

Index(['asin', 'helpful', 'overall', 'reviewText', 'reviewTime', 'reviewerID',
      'reviewerName', 'summary', 'unixReviewTime'],
      dtype='object')

-----Columns Info-----

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10261 entries, 0 to 10260
Data columns (total 9 columns):
asin                10261 non-null object
helpful             10261 non-null object
overall             10261 non-null int64
reviewText          10261 non-null object
```

Here is the copied output from the Console.

```
-----Columns Name-----

Index(['asin', 'helpful', 'overall', 'reviewText', 'reviewTime', 'reviewerID',
      'reviewerName', 'summary', 'unixReviewTime'],
      dtype='object')

-----Columns Info-----

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10261 entries, 0 to 10260
Data columns (total 9 columns):
asin                10261 non-null object
helpful             10261 non-null object
overall             10261 non-null int64
reviewText          10261 non-null object
```

```
reviewTime      10261 non-null object
reviewerID      10261 non-null object
reviewerName    10234 non-null object
summary         10261 non-null object
unixReviewTime  10261 non-null int64
dtypes: int64(2), object(7)
memory usage: 721.6+ KB
None
```

-----Shape Of Data-----

```
(10261, 9)
```

-----Checking duplicated reviews-----

-----END-----

-----Single product reviewed MAX times-----

	asin
B003VWJ2K8	163
B0002E1G5C	143
B0002F7K7Y	116
B003VWKPHC	114
B0002H0A3S	93

-----Single product reviewed MIN times-----

	asin
B0018PZR86	5
B001L8IJ0I	5
B001E43SK0	5
B0002D0MFI	5
B004MNTIL8	5

-----Matrix of Review Count and Frequency-----

	frequency
reviews	
5	558
6	296
7	188
8	128
9	71
10	43
11	38



14                    17  
12                    16  
13                    16

-----  
-----ACCURACY using Baseline-----

Estimating biases using als...

Baseline Prediction Accuracy : 0.8641864631930741

Matrix Factorization SVD Prediction Accuracy : 0.869462231758722

-----ACCURACY using SVD-----

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.9027	0.8724	0.8517	0.8909	0.8496	0.8735	0.0210
MAE (testset)	0.6505	0.6295	0.6228	0.6479	0.6249	0.6351	0.0117
Fit time	0.75	0.74	0.81	0.94	1.16	0.88	0.16
Test time	0.02	0.02	0.02	0.02	0.02	0.02	0.00

-----FOR SPECIFIC USER-----

[('B0002D01IG', 4.741960611991302), ('B00063678K', 4.6696610404405), ('B0009IEB0I', 4.640383315493166), ('B0002GZM00', 4.611787122915465), ('B000CD1R7K', 4.603202867515152), ('B0002D0096', 4.576012861411762), ('B00006LVEU', 4.432763718461853), ('B0002CZW0Y', 4.320583130302947), ('B000EE8YPK', 4.111775790892914)]

**Recommendation for User ->**

**A14VAT5EAX3D9S**

**['B000EEJJ5Y', 'B004XNK7AI', 'B001LJUVO4', 'B003VWJ2K8', 'B0002H0A3S', 'B003VWKPHC', 'B0002F7K7Y', 'B00646MZHk', 'B0002E1G5C', 'B0002CZVXM']**