NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

School of Electrical Engineering and Computer Sciences

CS-404 Big Data Analytics BSCS-11 2K21 (A) GP-1

Semester Project

Submitted To:	**Dr. Syed Imran Ali**
Date of Submission:	**20th May; 2024**
Class:	**BSCS-11A (GP-1)**

Group Members	CMS ID
Haram Nasir	383596
Muhammad Zain Jee	375803

Google drive link to dataset and notebook: 🗀 CS404_BDA_PROJECT

Title

E-commerce Recommendation System

Abstract

In the realm of e-commerce, the abundance of available products poses a challenge for users to navigate and make informed purchasing decisions. Recommender systems play a pivotal role in addressing this challenge by suggesting personalized items tailored to individual preferences. This project focuses on the development of an e-commerce recommender system leveraging the Amazon Food Reviews dataset. The dataset comprises a vast collection of user-generated reviews and ratings for food products, offering rich insights into consumer preferences and product sentiments. By harnessing the power of machine learning algorithms, the system aims to analyze user behavior and preferences encoded within the textual reviews.

Techniques such as collaborative filtering, association rules and matrix factorization will be employed to generate recommendations that align with user tastes and preferences. The recommender system will be designed to provide seamless integration within e-commerce platforms, enhancing user experience and facilitating informed purchasing decisions. Through experimentation and evaluation, the efficacy and performance of the system will be assessed, with the goal of delivering personalized recommendations that enrich the shopping experience for users in the digital marketplace.

Introduction

The need for personalized experiences has become paramount for both businesses and consumers. An e-commerce recommendation system fueled by big data analytics stands as a cornerstone in addressing this necessity. By applying big data techniques, e-commerce websites can derive valuable insights, improve customer experience, and optimize business strategies for their business.

Use case

Product Recommendation:

A recommendation system for E-commerce platforms that suggests products to users based on their past purchases, ratings, reviews, and product attributes.

Market Basket Analysis:

Analyze patterns of co-occurrence of products in users' baskets to uncover associations and dependencies between different items.

Trend Analysis:

Identify trends in product popularity and customer preferences over time.

Dataset

We have chosen the Amazon Fine Food reviews dataset from Kaggle. 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.

Kaggle Dataset: https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

Data includes:
Reviews from Oct 1999 - Oct 2012
568,454 reviews
256,059 users
74,258 products
260 users with > 50 reviews

Why this dataset?

- The dataset contains various attributes such as ProductId, UserId, Score, Summary, and Text.
- These attributes provide rich information which can be leveraged to build a robust recommendation system as it captures a wider range of user preferences and item characteristics.
- The dataset is very large and has a diverse range of products. It also considers the relevance and reliability of reviews by providing helpfulness metrics.

Data import and pre-processing

```
[2] import pandas as pd

    food_reviews_data = pd.read_csv("/content/drive/MyDrive/CS404_BDA_PROJECT/Reviews.csv")
```

```
[3] food_reviews_data.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...

Rename ‘Score’ column to ‘Rating’.

```
food_reviews_data.rename(columns={'Score':'Rating'}, inplace=True )
```

Remove rows with missing values.

```
# dropping rows with missing values
food_reviews_data.dropna(inplace=True)
```

Check for missing values.

```
[ ]  # check for missing values again
     print('Number of missing values across columns: \n', food_reviews_data.isnull().sum())
```

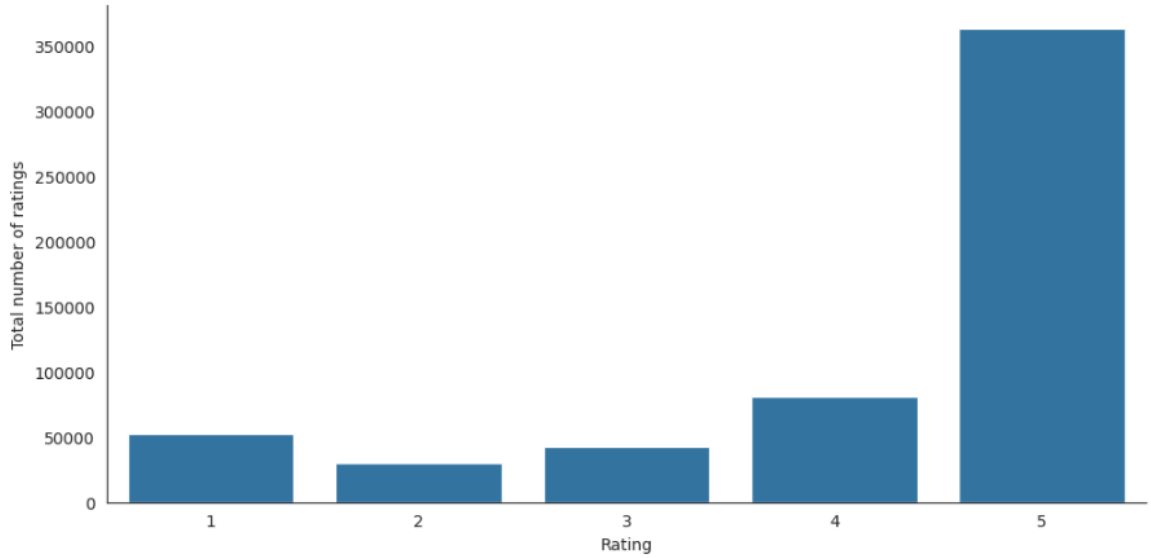
```
Number of missing values across columns:
 Id                      0
ProductId                0
UserId                   0
ProfileName              0
HelpfulnessNumerator     0
HelpfulnessDenominator   0
Rating                   0
Time                     0
Summary                  0
Text                     0
dtype: int64
```

Convert time to date time format.

```
# Convert 'Time' column to datetime
food_reviews_data['Time'] = pd.to_datetime(food_reviews_data['Time'], unit='s', errors='coerce')
```

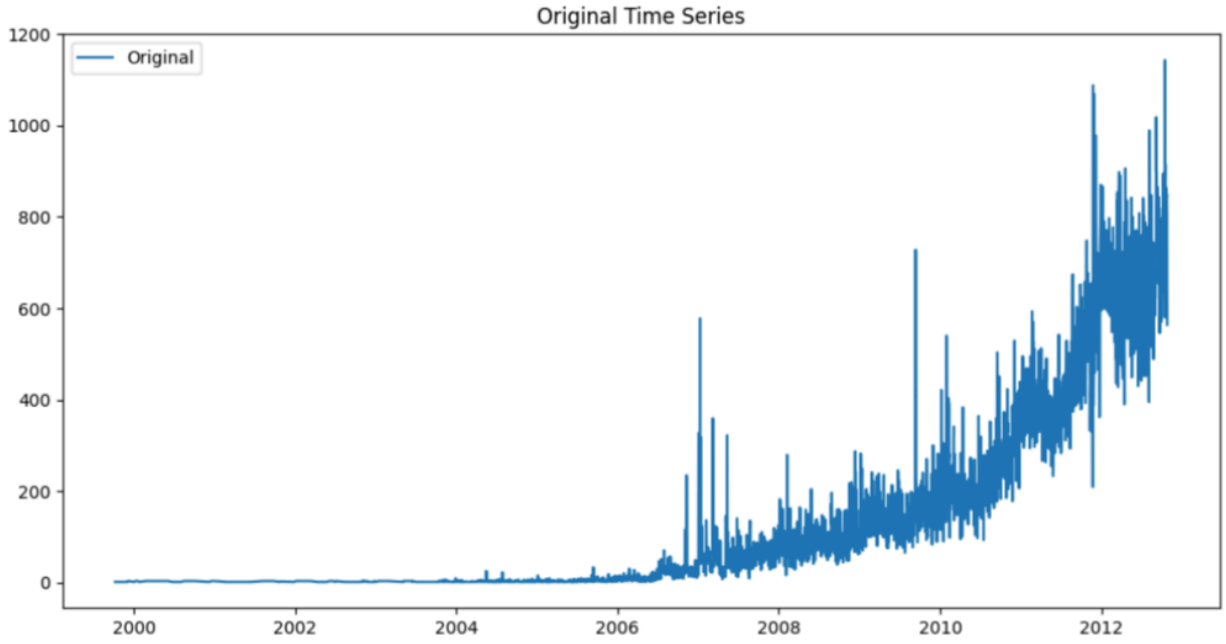
Data Analysis and Insights (EDA)

Ratings Distribution Graph

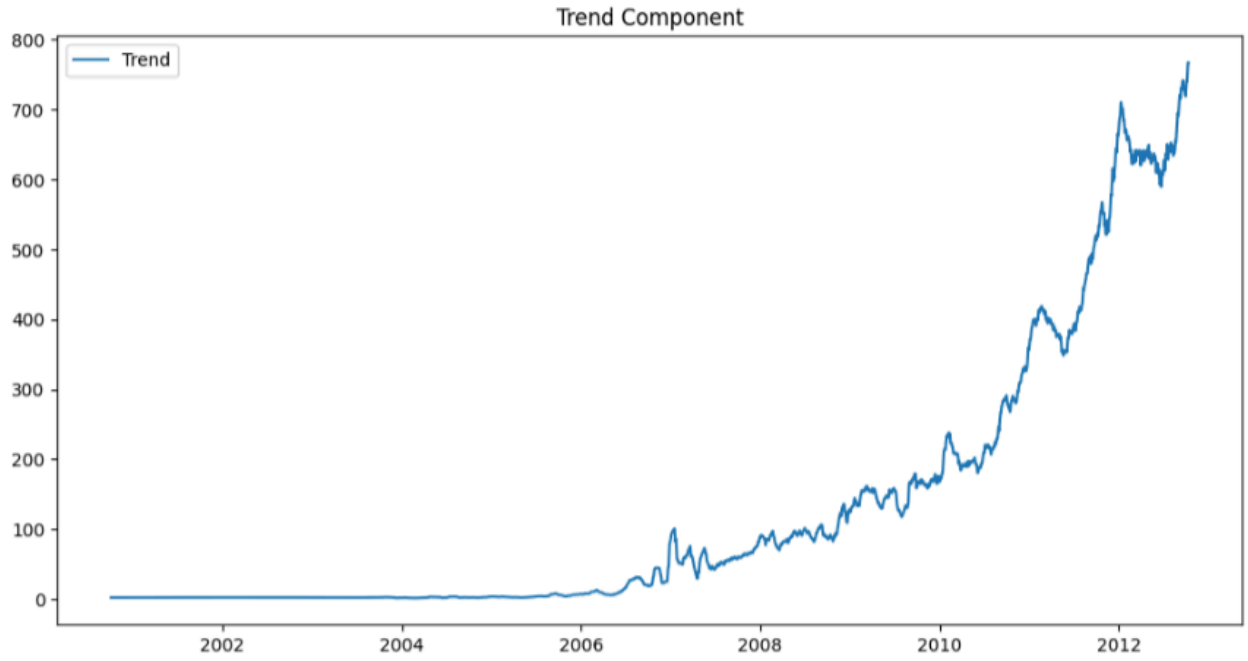


This bar graph provides insights into the distribution of rating numbers across the dataset.

Trend Analysis



This plot shows the number of user-generated reviews over the years.




```
Minimum rating is: 1
Maximum rating is: 5
```

Number of unique users and products.

```
Total no of ratings : 568401
Total No of Users   : 256042
Total No of products  : 74257
```

Size of the dataset.

```
food_reviews_data.shape
```

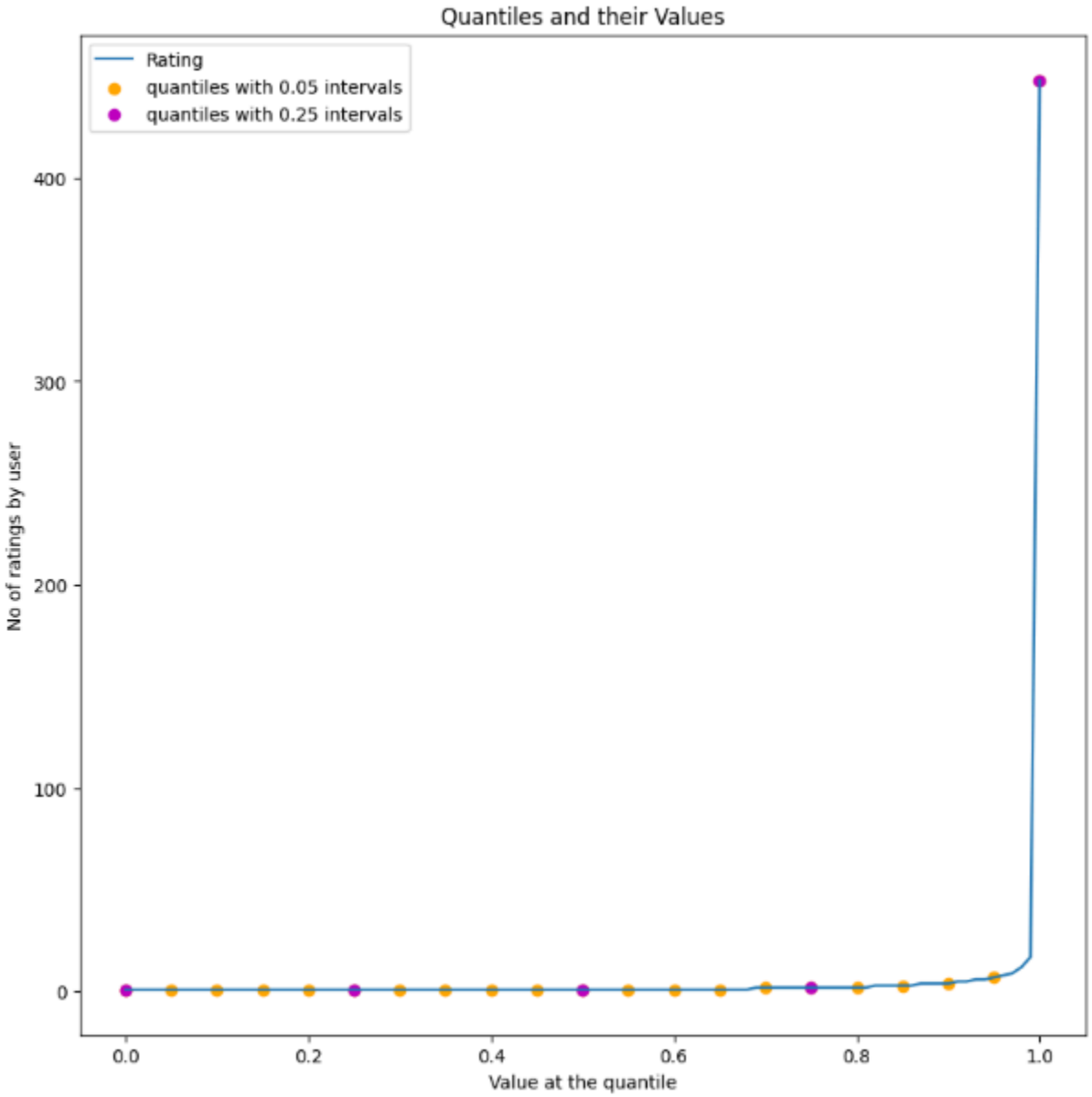
```
(568454, 10)
```

Number of ratings given by each user.

```
UserId
A3OXHLG6DIBRW8    448
A1YUL9PCJR3JTY    421
AY12DBB0U420B     389
A281NPSIMI1C2R    365
A1Z54EM24Y40LL    256
Name: Rating, dtype: int64
```

Popular products.

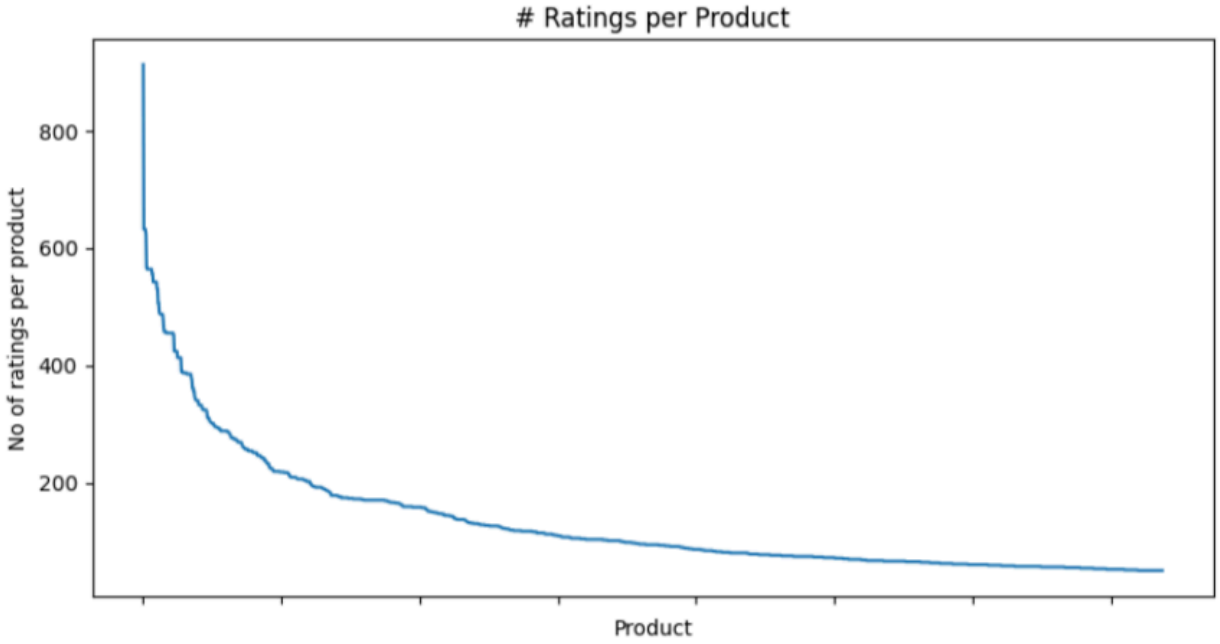
```
No of products rated more than 50 per user : 267
```

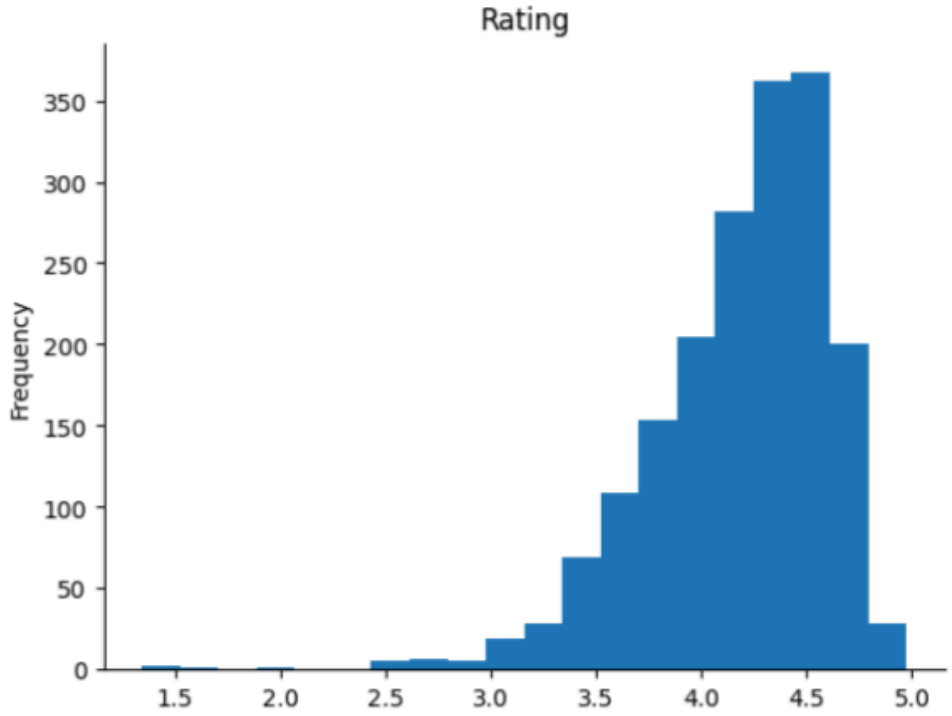


This plot shows the distribution of the number of rated products per user, along with specific quantiles highlighted at intervals of 0.05 and 0.25. The line plot represents the trend of the distribution, while the scatter points emphasize particular quantiles for better interpretation.

Popularity based recommendations to new users

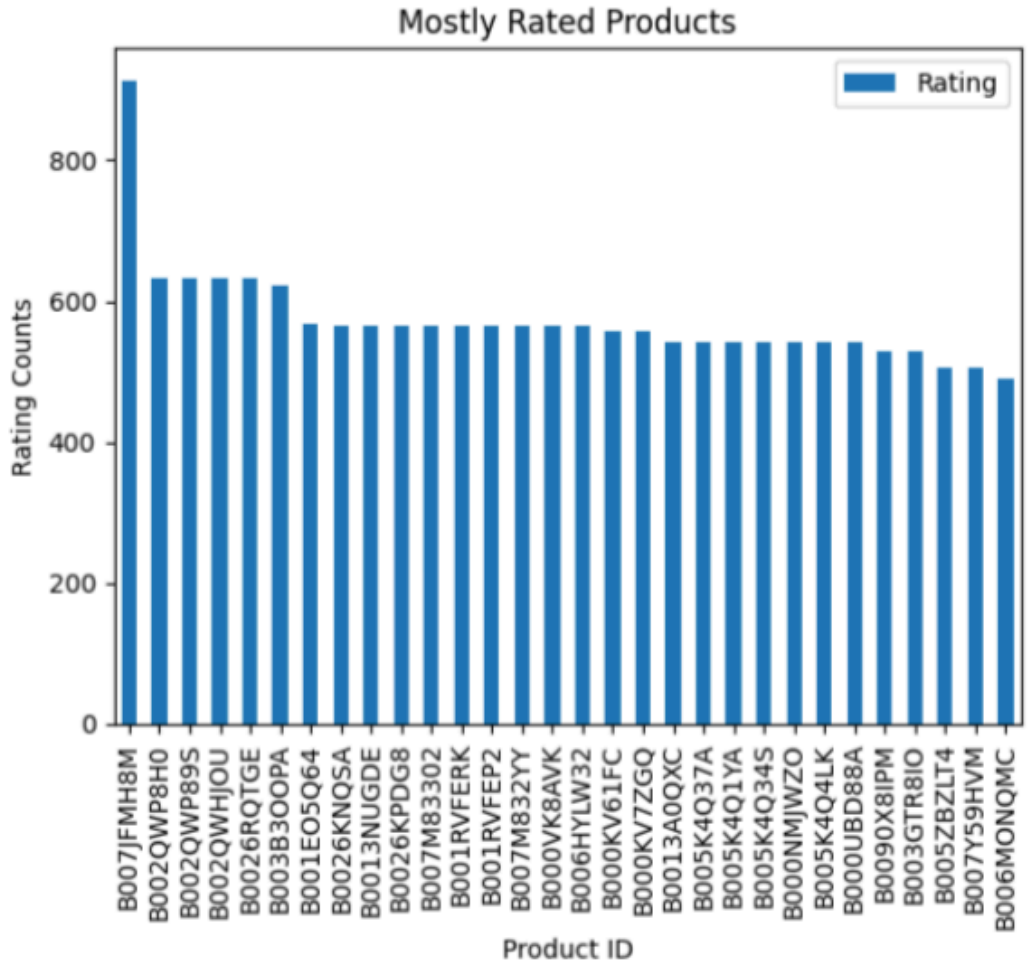
(products which have more than 50 ratings are popular products)





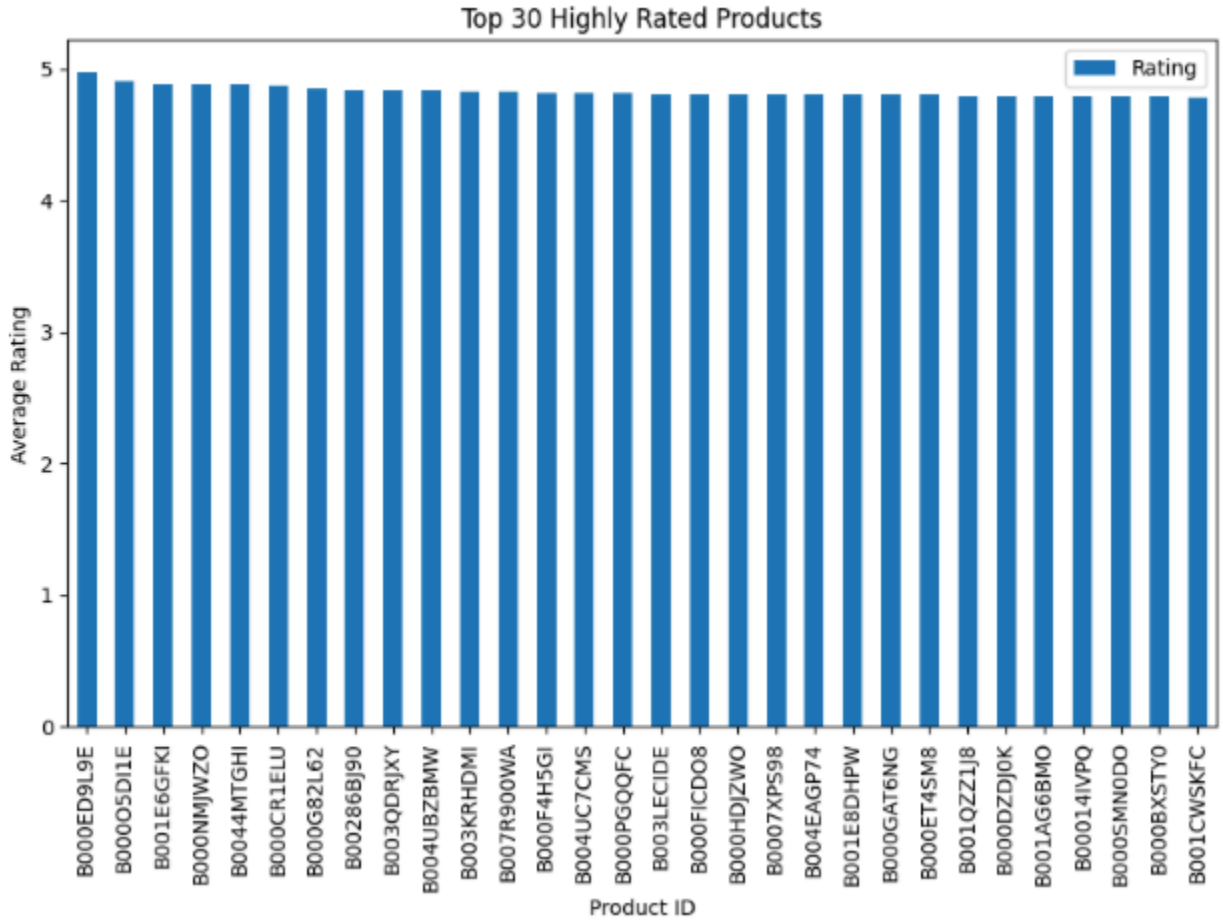
Visualization of popular products

(in terms of count of ratings)



Visualization of Top Rated Products

(in terms of rating number)



Products with the largest count of ratings and which are also highly rated.

ProductId	Rating	rating_counts
7310172001	4.751445	173
7310172101	4.751445	173
B00004CI84	4.486772	189
B00004CXX9	4.405128	195
B00004RAMY	4.104651	172

Big Data Techniques

Item based Collaborative Filtering using KNNWithMeans Algorithm (based on previous user-item interactions)

Collaborative filtering works by generating personalized recommendations for users based on the similarity between items. Techniques like Cosine similarity, Pearson Correlation Coefficient and Matrix Factorization can be used for this.

KNNWithMeans is a basic collaborative filtering algorithm, taking into account the mean ratings of each product. Pearson Correlation Coefficient (PCC) is one of the most popular similarity measures for Collaborative filtering recommender system, to evaluate how much two products are correlated.

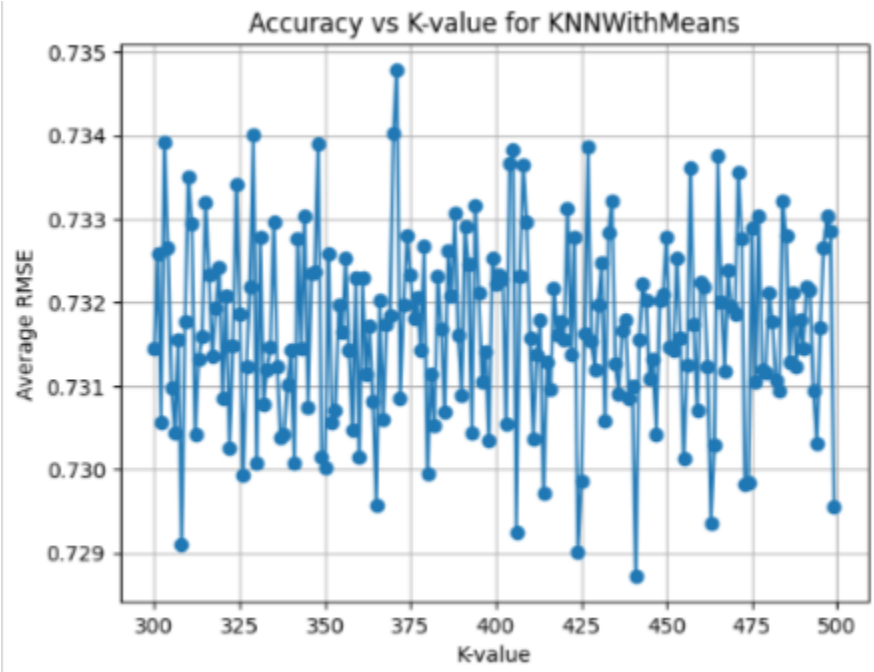
This is how **KNNWithMeans** algorithm works:

- **Item Similarity Calculation:** The first step is to calculate the similarity between pairs of items in the dataset. Various similarity measures can be used for this purpose, such as cosine similarity, Pearson correlation, or Jaccard similarity. The similarity between items is typically calculated based on the ratings or preferences of users who have interacted with both items.

- **Neighborhood Selection:** Once the similarities between items are calculated, a neighborhood of similar items is selected for each item. This neighborhood typically consists of the k most similar items to the target item.

- **Recommendation Generation:** To generate recommendations for a user, the system identifies the items that the user has already interacted with or rated. It then looks at the neighborhood of each of these items and aggregates the ratings or preferences of the user for those items. The items with the highest aggregated scores are recommended to the user.

RMSE vs K-value plot to select optimal value of K



We used **Azure Compute Instance** to train our KNNWIthMeans model.

	Name ↑	Category	Workload types	Available quota ⓘ	Cost ⓘ
○	Standard_DS11_v2 2 cores, 14GB RAM, 28GB storage	Memory optimized	Development on Notebooks (or other IDE) and light weight testing	6 cores	$0.23/hr
○	Standard_DS3_v2 4 cores, 14GB RAM, 28GB storage	General purpose	Classical ML model training on small datasets	6 cores	$0.43/hr
●	Standard_E4ds_v4 4 cores, 32GB RAM, 150GB storage	Memory optimized	Data manipulation and training on medium-sized datasets (1-10GB)	4 cores	$0.35/hr
○	Standard_F4s_v2 4 cores, 8GB RAM, 32GB storage	Compute optimiz...	Data manipulation and training on large datasets (>10 GB)	16 cores	$0.22/hr

```python
# (similarity between items based on user interactions)

# new_df is data frame that contains products who got more than 50 ratings (popular products)

from surprise.model_selection import train_test_split
from surprise import KNNWithMeans
from surprise import Dataset
from surprise import accuracy
from surprise import Reader

# Reading the dataset
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(new_df[['UserId', 'ProductId', 'Rating']], reader)

# Splitting the dataset
trainset, testset = train_test_split(data, test_size=0.3, random_state=10)


# try using square root of number of transactions for optimal value of k
# Use user_based true/false to switch between user-based or item-based collaborative filtering

algo = KNNWithMeans(k=440, sim_options={'name': 'pearson_baseline', 'user_based': False})
algo.fit(trainset)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNWithMeans at 0x789a270c5840>
```

We trained a collaborative filtering model using the KNNWithMeans algorithm with similarity based on Pearson baseline similarity. The model is then trained on the training set and ready to make predictions on the test set.

k=440 indicates the number of nearest neighbors to consider.

sim_options={'name': 'pearson_baseline', 'user_based': False} specifies the similarity metric to use (Pearson baseline similarity) and sets the algorithm to use item-based collaborative filtering.

```python
# run the trained model against the testset
test_pred = algo.test(testset)

# Display the first few predictions
for pred in test_pred[:20]:
    print(pred)
```

```
user: A1EC6RGTJN9I8K item: B005DFL4PM r_ui = 5.00   est = 4.99   {'actual_k': 4, 'was_impossible': False}
user: A2RME4353Y911C item: B0037QGKSK r_ui = 3.00   est = 3.02   {'actual_k': 4, 'was_impossible': False}
user: A1TAARTDC411Q0 item: B004779XHI r_ui = 5.00   est = 4.93   {'actual_k': 5, 'was_impossible': False}
user: A36HQGRJWI5LSG item: B000H1217M r_ui = 5.00   est = 5.00   {'actual_k': 4, 'was_impossible': False}
user: A2CA0DTV2JFL0F item: B003CK7MME r_ui = 5.00   est = 4.98   {'actual_k': 8, 'was_impossible': False}
user: A1WOO2G7ECIP10 item: B002ZX1U9A r_ui = 5.00   est = 5.00   {'actual_k': 6, 'was_impossible': False}
user: A1GEUPI7SIJQ4Z item: B001EYUE5M r_ui = 5.00   est = 4.99   {'actual_k': 3, 'was_impossible': False}
user: A1S2U7YG3K11HZ item: B000EVOSE4 r_ui = 5.00   est = 4.97   {'actual_k': 2, 'was_impossible': False}
user: AEYYRO0ERDP3A item: B0013A0QXC r_ui = 2.00   est = 1.99   {'actual_k': 2, 'was_impossible': False}
user: A3TIACCF9FM8PB item: B004E4EBMG r_ui = 5.00   est = 5.00   {'actual_k': 2, 'was_impossible': False}
user: AZRJH4JFB59VC item: B000YSTIL0 r_ui = 5.00   est = 5.00   {'actual_k': 8, 'was_impossible': False}
user: A3O197O0BP0EA3 item: B002QWP89S r_ui = 5.00   est = 5.00   {'actual_k': 2, 'was_impossible': False}
user: A1IWCCJJ10TRUC item: B001E530J8 r_ui = 5.00   est = 4.17   {'was_impossible': True, 'reason': 'User and/or item is unknown.'}
user: A3NST76OWSYSKK item: B008J1HO4C r_ui = 5.00   est = 5.00   {'actual_k': 1, 'was_impossible': False}
user: ANYRHPZLNJDFX item: B002GKEK7G r_ui = 5.00   est = 4.17   {'was_impossible': True, 'reason': 'User and/or item is unknown.'}
user: A2FRPGKIJJQI3E item: B00764BRS2 r_ui = 5.00   est = 5.00   {'actual_k': 1, 'was_impossible': False}
user: A3K185BXWHVS6X item: B002QWHJOU r_ui = 1.00   est = 1.03   {'actual_k': 7, 'was_impossible': False}
user: A1Y3QKMCHRT73Y item: B003QNLUTI r_ui = 4.00   est = 4.39   {'actual_k': 10, 'was_impossible': False}
user: AKX0GK73F2XVE item: B007PA33NY r_ui = 5.00   est = 4.96   {'actual_k': 8, 'was_impossible': False}
user: A253N5LJL94DLF item: B002ANCDRI r_ui = 5.00   est = 5.00   {'actual_k': 9, 'was_impossible': False}
```

The model makes rating predictions.

Accuracy of model

```
Item-based Model : Test Set
RMSE: 0.3425

0.34252282424166985
```

This means that the predicted ratings made by model deviate from the actual ratings by 34%.

Testing the model (example)

```
def recommend_items_to_user(user_id, num_items_to_recommend=5):
```

```
Recommended items for user A3OXHLG6DIBRW8 :
Item ID: B000EVOSE4 , Estimated Rating: 5
Item ID: B0027UYT4O , Estimated Rating: 5
Item ID: B00474VPY0 , Estimated Rating: 5
Item ID: B00474VPLI , Estimated Rating: 5
Item ID: B000KEPBBY , Estimated Rating: 5
```

Item based Collaborative Filtering using Matrix Decomposition (TruncatedSVD)

Matrix factorization is a way to generate latent (aspects of the data that are not directly measurable or observed) features when multiplying two different kinds of entities. Hence, from the matrix factorization, we are able to discover these latent features to give a prediction on a rating with respect to the similarity in user's preferences and interactions.

Matrix factorization is a collaborative filtering method to find the relationship between items' and users' entities. **Latent features**, the association between users and products matrices, are determined to find similarity and make a prediction based on both item and user entities.

The dot product of the user and item matrix generates the rating matrix. The matrix **|U|*|P|** includes all the ratings given by users. It is the utility matrix. The matrix is sparse (has missing values) because not every user gives ratings to all the products.

Given a scenario, some user didn't give a rating to a particular product. We'd like to know if the user would like that product. The method is to discover other users with similar preferences of the given user, by taking the ratings given by users of similar preferences and predict whether the user would like the product or not.

```
new_df1= new_df.head(10000)
ratings_matrix = new_df1.pivot_table(values='Rating', index='UserId', columns='ProductId', fill_value=0)
ratings_matrix.head()
```

ProductId	B000084ETV	B00008JOL0	B000255OIG	B0007A0AP8	B000DZKKK2	B000EM6PC6	B000EVIDUY	B000FBM3YK	B000FNEX50	B000G6MBX2	..
UserId											
A1004703RC79J9	0	0	0.0	0	0	0	0	0	0	0	
A1008DPSP6KC9J	0	0	0.0	0	0	0	0	0	0	0	
A100WO06OQR8BQ	0	0	0.0	0	0	0	0	0	0	0	
A101C99CG8EFUH	0	0	0.0	0	0	0	0	0	0	0	
A102UXGLDF76G1	0	0	0.0	0	0	0	0	0	0	0	

5 rows × 69 columns

As shown, the utility matrix is pretty sparse.

```
# Decomposing the Matrix

from sklearn.decomposition import TruncatedSVD

# n is the number of dimensions
SVD = TruncatedSVD(n_components=10)
decomposed_matrix = SVD.fit_transform(X)
decomposed_matrix.shape
```

```
(69, 10)
```

n_components=10: This parameter specifies the desired number of components (features) after dimensionality reduction. In this case, it's set to 10, meaning that the algorithm will reduce the dimensionality of the data to 10 features.

random_state=42: This parameter ensures reproducibility of results by fixing the random seed.

```
# Correlation Matrix (using pearson correlation coefficient)
import numpy as np

correlation_matrix = np.corrcoef(decomposed_matrix)
correlation_matrix.shape
```

```
(69, 69)
```

The correlation matrix is a symmetric matrix where each element (i, j) represents the Pearson correlation coefficient between the ith and jth columns (features) of the decomposed_matrix.

Recommending top 25 highly correlated products in sequence.

```
Recommend = list(X.index[correlation_product_ID > 0.65])

# Removes the item already bought by the customer
Recommend.remove(i)

Recommend[0:24]
```

```
['B0007A0AP8',
 'B000FBM3YK',
 'B000G6RYNE',
 'B0016FY6H6',
 'B001BDDTB2',
 'B001CD1VI4',
 'B001CGTN1I',
 'B001E5E20O',
 'B001EO653M',
 'B001HTJ49G',
 'B001LG945O',
 'B001LGGH40',
 'B001PO7FIU',
 'B00213ERI0',
 'B002TMV34E',
 'B002TMV3E4',
 'B0034KP00S',
 'B00503DP0O']
```

Truncated SVD is a popular technique in machine learning for reducing the dimensions of high-dimensional data while retaining most of the original information.

This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with sparse matrices efficiently.

Why Truncated SVD?

- Truncated SVD gracefully navigates through the ocean of data, offering a blend of **computational efficiency and information retention.**

- Truncated SVD enhances interpretability by transforming the data into a lower-dimensional space where the relationships between variables can be more readily discerned.

Item-based collaborative filtering has several advantages:

Scalability: It tends to be more scalable than user-based collaborative filtering, especially in scenarios where the number of users is large.
Stability: Item-based recommendations are often more stable over time than user-based recommendations because items tend to change less frequently than users' preferences.

Limitations:

Cold Start: It suffers from the cold start problem for new items, as there may not be enough data available to calculate accurate similarities.
Sparsity: Similar to user-based collaborative filtering, item-based collaborative filtering can suffer from sparsity in the user-item interaction matrix, where most users have only interacted with a small fraction of items.

Why item-based collaborative filtering instead of user-based collaborative filtering?

- The number of users weighs more than the number of products, which makes item-based collaborative filtering more computationally efficient.
- Items (food products) will not change frequently in this project, while users' preferences may change frequently.
- Recommending products based on users' historical data makes more sense intuitively.

Association Rules Mining

(using Hadoop in Cloudera VM)

Cloudera VM hadoop commands

1. sudo -u hdfs hadoop fs -mkdir /frequent_itemset

2. hdfs dfs -ls /

3. sudo -u hdfs hadoop fs -put
   /home/cloudera/workspace/SharedFolder/Transactions.txt /frequent_itemset

4. hdfs dfs -ls /frequent_itemset

5. chmod 777
   /home/cloudera/workspace/SharedFolder/frequent_itemset_mapper.py
   /home/cloudera/workspace/SharedFolder/frequent_itemset_reducer.py

6. sudo -u hdfs hadoop fs -chown cloudera /frequent_itemset

7. hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.4.2.jar \
   -input /frequent_itemset/Transactions.txt \
   -output /frequent_itemset/output2 \
   -mapper /home/cloudera/workspace/SharedFolder/frequent_itemset_mapper.py \
   -reducer /home/cloudera/workspace/SharedFolder/frequent_itemset_reducer.py

8. hdfs dfs -cat /frequent_itemset/output2/part-00000

cloudera@quickstart:~ (Screenshots from cloudera terminal)

packageJobJar: [] [/usr/jars/hadoop-streaming-2.6.0-cdh5.4.2.jar]
/tmp/streamjob664109486487075741.jar tmpDir=null
24/05/10 23:48:48 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
24/05/10 23:48:54 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
24/05/10 23:49:01 INFO mapred.FileInputFormat: Total input paths to process : 1
24/05/10 23:49:02 INFO mapreduce.JobSubmitter: number of splits:2
24/05/10 23:49:04 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1714989208189_0003

24/05/10 23:49:08 INFO impl.YarnClientImpl: Submitted application application_1714989208189_0003
24/05/10 23:49:09 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1714989208189_0003/
24/05/10 23:49:09 INFO mapreduce.Job: Running job: job_1714989208189_0003
24/05/10 23:50:25 INFO mapreduce.Job: Job job_1714989208189_0003 running in uber mode : false

Map Job

24/05/10 23:50:25 INFO mapreduce.Job: map 0% reduce 0%
24/05/10 23:53:04 INFO mapreduce.Job: map 2% reduce 0%
24/05/10 23:53:08 INFO mapreduce.Job: map 3% reduce 0%
24/05/10 23:53:11 INFO mapreduce.Job: map 4% reduce 0%
24/05/10 23:53:27 INFO mapreduce.Job: map 5% reduce 0%
24/05/10 23:53:38 INFO mapreduce.Job: map 6% reduce 0%
24/05/10 23:53:42 INFO mapreduce.Job: map 7% reduce 0%
24/05/10 23:53:57 INFO mapreduce.Job: map 8% reduce 0%
24/05/10 23:54:00 INFO mapreduce.Job: map 9% reduce 0%
24/05/10 23:54:18 INFO mapreduce.Job: map 10% reduce 0%
24/05/10 23:54:26 INFO mapreduce.Job: map 11% reduce 0%
24/05/10 23:54:38 INFO mapreduce.Job: map 12% reduce 0%
24/05/10 23:54:58 INFO mapreduce.Job: map 13% reduce 0%
24/05/10 23:54:59 INFO mapreduce.Job: map 14% reduce 0%
24/05/10 23:55:14 INFO mapreduce.Job: map 15% reduce 0%
24/05/10 23:55:16 INFO mapreduce.Job: map 16% reduce 0%
24/05/10 23:55:30 INFO mapreduce.Job: map 17% reduce 0%
24/05/10 23:55:37 INFO mapreduce.Job: map 18% reduce 0%
24/05/10 23:55:54 INFO mapreduce.Job: map 19% reduce 0%
24/05/10 23:55:55 INFO mapreduce.Job: map 20% reduce 0%
24/05/10 23:56:13 INFO mapreduce.Job: map 21% reduce 0%
24/05/10 23:56:17 INFO mapreduce.Job: map 22% reduce 0%
24/05/10 23:56:27 INFO mapreduce.Job: map 23% reduce 0%
24/05/10 23:56:36 INFO mapreduce.Job: map 24% reduce 0%
24/05/10 23:56:52 INFO mapreduce.Job: map 25% reduce 0%
24/05/10 23:57:06 INFO mapreduce.Job: map 26% reduce 0%
24/05/10 23:57:12 INFO mapreduce.Job: map 27% reduce 0%
24/05/10 23:57:36 INFO mapreduce.Job: map 28% reduce 0%
24/05/10 23:57:40 INFO mapreduce.Job: map 29% reduce 0%
24/05/10 23:57:50 INFO mapreduce.Job: map 30% reduce 0%
24/05/10 23:57:54 INFO mapreduce.Job: map 31% reduce 0%
24/05/10 23:58:07 INFO mapreduce.Job: map 32% reduce 0%
24/05/10 23:58:10 INFO mapreduce.Job: map 33% reduce 0%
24/05/10 23:58:34 INFO mapreduce.Job: map 35% reduce 0%

24/05/10 23:58:48 INFO mapreduce.Job: map 36% reduce 0%
24/05/10 23:58:51 INFO mapreduce.Job: map 37% reduce 0%
24/05/10 23:59:02 INFO mapreduce.Job: map 38% reduce 0%
24/05/10 23:59:18 INFO mapreduce.Job: map 39% reduce 0%
24/05/10 23:59:32 INFO mapreduce.Job: map 41% reduce 0%
24/05/10 23:59:42 INFO mapreduce.Job: map 43% reduce 0%
24/05/10 23:59:54 INFO mapreduce.Job: map 44% reduce 0%
24/05/11 00:00:18 INFO mapreduce.Job: map 45% reduce 0%
24/05/11 00:00:28 INFO mapreduce.Job: map 46% reduce 0%
24/05/11 00:00:35 INFO mapreduce.Job: map 47% reduce 0%
24/05/11 00:00:44 INFO mapreduce.Job: map 48% reduce 0%
24/05/11 00:00:45 INFO mapreduce.Job: map 49% reduce 0%
24/05/11 00:00:59 INFO mapreduce.Job: map 50% reduce 0%
24/05/11 00:01:09 INFO mapreduce.Job: map 51% reduce 0%
24/05/11 00:01:19 INFO mapreduce.Job: map 52% reduce 0%
24/05/11 00:01:25 INFO mapreduce.Job: map 53% reduce 0%
24/05/11 00:01:33 INFO mapreduce.Job: map 54% reduce 0%
24/05/11 00:01:38 INFO mapreduce.Job: map 55% reduce 0%
24/05/11 00:01:54 INFO mapreduce.Job: map 56% reduce 0%
24/05/11 00:01:55 INFO mapreduce.Job: map 57% reduce 0%
24/05/11 00:02:00 INFO mapreduce.Job: map 58% reduce 0%
24/05/11 00:02:06 INFO mapreduce.Job: map 59% reduce 0%
24/05/11 00:02:15 INFO mapreduce.Job: map 60% reduce 0%
24/05/11 00:02:20 INFO mapreduce.Job: map 61% reduce 0%
24/05/11 00:02:28 INFO mapreduce.Job: map 62% reduce 0%
24/05/11 00:02:42 INFO mapreduce.Job: map 63% reduce 0%
24/05/11 00:02:50 INFO mapreduce.Job: map 64% reduce 0%
24/05/11 00:02:55 INFO mapreduce.Job: map 65% reduce 0%
24/05/11 00:02:59 INFO mapreduce.Job: map 66% reduce 0%
24/05/11 00:03:10 INFO mapreduce.Job: map 67% reduce 0%
24/05/11 00:03:30 INFO mapreduce.Job: map 83% reduce 0%
24/05/11 00:03:55 INFO mapreduce.Job: map 100% reduce 0%

Reduce Job

24/05/11 00:52:30 INFO mapreduce.Job: map 100% reduce 17%
24/05/11 00:52:36 INFO mapreduce.Job: map 100% reduce 34%
24/05/11 00:52:40 INFO mapreduce.Job: map 100% reduce 41%
24/05/11 00:52:43 INFO mapreduce.Job: map 100% reduce 49%
24/05/11 00:52:48 INFO mapreduce.Job: map 100% reduce 58%
24/05/11 00:52:52 INFO mapreduce.Job: map 100% reduce 67%
24/05/11 00:54:00 INFO mapreduce.Job: map 100% reduce 100%
24/05/11 00:54:04 INFO mapreduce.Job: Job job_1714989208189_0004 **completed successfully**

24/05/11 00:54:12 INFO mapreduce.Job: Counters: 49

Association Rules (output)

B0095KATS4	→	B005BFJGJU
B0095KATS4	→	B0057ISDW2
B0095KATS4	-->	B006Y02OZO
B0095KATS4	-->	B006ILRBWU
B0095WTUUA	-->	B002O3SAJE
B0095WTUUA	-->	B001AHL6CI
B0096BR5O8	-->	B00012OHZ6
B0096E5196	-->	B0077HS6F0
B0095KATS4	-->	B005BFJGJUS
B0095KATS4	-->	B0057ISDW2
B0096BR5O8	-->	B00012OHZ6
B0096E5196	-->	B0077HS6F0
B00975HC9G	-->	B0030VBQGS
B0098C68SO	-->	B006AV2T3E
B0098C68SO	-->	B000Q56OXQ
B0098C6AKK	-->	B0098C6B36
B0098C6AKK	-->	B0012W1TQE
B0098C6B36	-->	B000H7F5IS
B0098C6B36	-->	B0098C6AKK
B00993CW3M	-->	B002MO765O
B0099AUJBW	-->	B001EQ5KNQ
B009AFH6Y4	-->	B002RU3GH0
B009AFJ3I6	-->	B009AFJ548
B009AFJ3I6	-->	B003DVKBK2
B009AFJ548	-->	B003WKJW5Y
B00975HC9G	-->	B0030VBQGS
B0098C68SO	-->	B006AV2T3E
B0098C68SO	-->	B000Q56OXQ
B0098C6AKK	-->	B0098C6B36
B0098C6AKK	-->	B0012W1TQE
B0098C6B36	-->	B0098C6AKK
B0098C6B36	-->	B000H7F5IS
B0098C6B36	→	B0098C6AKK
B00993CW3M	→	B002MO765O
B0099AUJBW	-->	B001EQ5KNQ
B009AFH6Y4	-->	B002RU3GH0
B009AFJ3I6	-->	B009AFJ548
B009AFJ3I6	-->	B003DVKBK2

B009AFJ548	-->	B003WKJW5Y
B009C9BPDG	-->	B003ZNXCFO
B009C9BPDG	-->	B003ZNRDRM
B009CW5VUQ	-->	B00142IBJ0
B009CW5VUQ	-->	B00142IBJ0
B009D53U6Y	-->	B000FTS86Y
B009D53U6Y	-->	B000FTU5MO
B009D53U6Y	-->	B000GG0BLQ
B009D53U6Y	-->	B000FTS86Y
B009D53XKC	-->	B000FTR540
B009D53XKC	-->	B000FTU5MY
B009DQAWJG	→	B009DQLVMI
B009DQAWJG	-->	B009DQIH0W
B009DQIH0W	-->	B009DQLVMI
B009DR3GYI	-->	B009DQIH0W
B009DR3GYI	-->	B009DQAWJG
B009DR3GYI	-->	B009DQLVMI
B009E7YC54	-->	B004P4POZ8
B009E7YC54	-->	B008BY7NSE

Contd ..

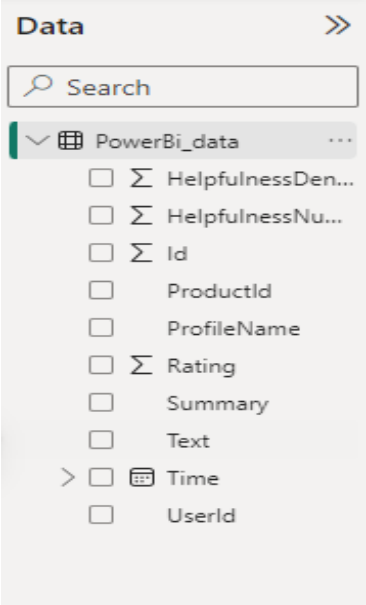
Content-based filtering

We tried performing content-based filtering using **TF-IDF (term frequency-inverse document frequency)** in hadoop. We used the text reviews for the products as a basis for TF-IDF. We preprocessed the text column in our dataset and used the natural language toolkit **(nltk)** to remove stop words, tokenize words and perform lemmatization. But since our dataset was very intensive, the mapreduce job failed. Here's a snapshot of the job that we ran in the mapreduce framework in hadoop.

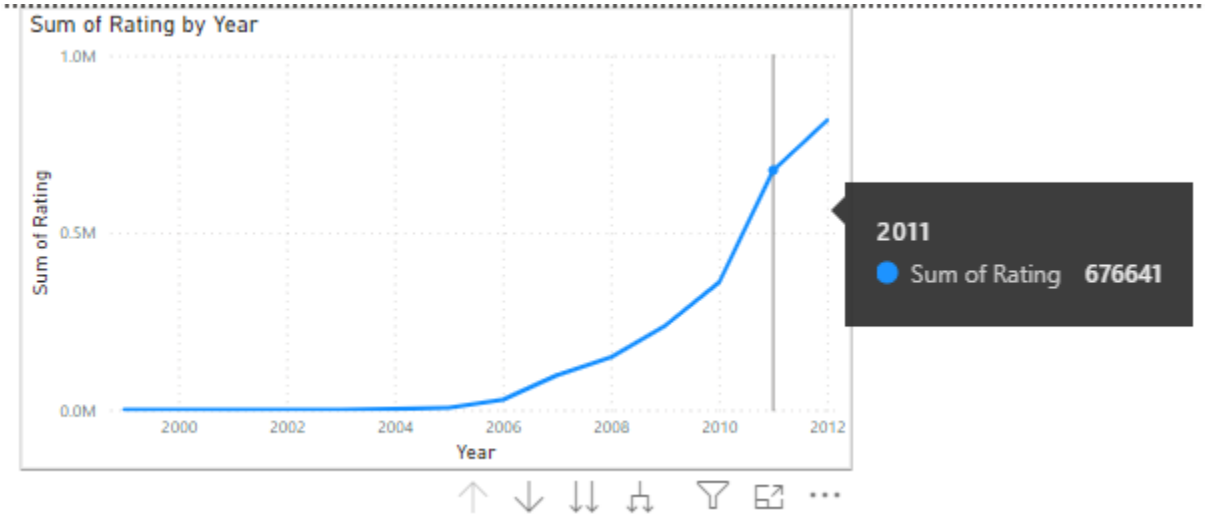
```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming-2.6.0-cdh5.4.2.jar -input /TF_IDF/Reviews_text.txt -
output /TF_IDF/TF_IDF_SCORES -mapper /home/cloudera/workspace/SharedFolder/cbf_mapper.py -reducer
/home/cloudera/workspace/SharedFolder/cbf_reducer.py
packageJobJar: [] [/usr/jars/hadoop-streaming-2.6.0-cdh5.4.2.jar] /tmp/streamjob2438498563757996855.jar tmpDir=null
24/05/20 03:39:48 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
24/05/20 03:39:52 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
24/05/20 03:39:59 INFO mapred.FileInputFormat: Total input paths to process : 1
24/05/20 03:40:00 INFO net.NetworkTopology: Adding a new node: /default-rack/10.0.2.15:50010
24/05/20 03:40:00 INFO mapreduce.JobSubmitter: number of splits:2
24/05/20 03:40:03 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1714989208189_0008
24/05/20 03:40:08 INFO impl.YarnClientImpl: Submitted application application_1714989208189_0008
24/05/20 03:40:12 INFO mapreduce.Job: The url to track the job:
http://quickstart.cloudera:8088/proxy/application_1714989208189_0008/
24/05/20 03:40:12 INFO mapreduce.Job: Running job: job_1714989208189_0008
24/05/20 03:41:20 INFO mapreduce.Job: Job job_1714989208189_0008 running in uber mode : false
24/05/20 03:41:20 INFO mapreduce.Job:  map 0% reduce 0%
```

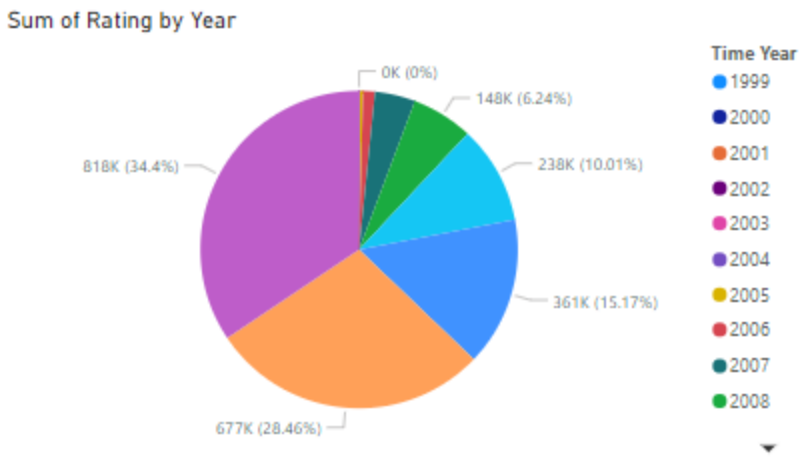
```
24/05/20 03:43:29 INFO mapreduce.Job: Task Id : attempt_1714989208189_0008_m_000001_0, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
```

Power BI Interface

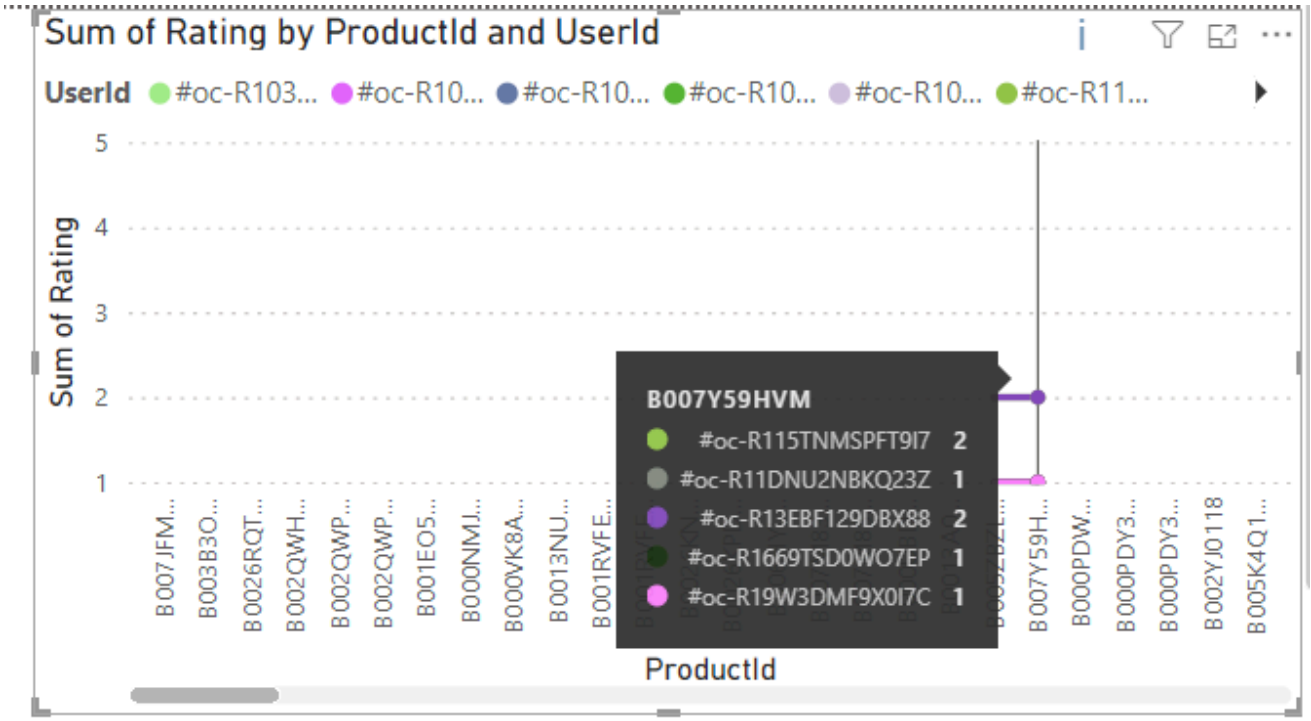


Data visualizations

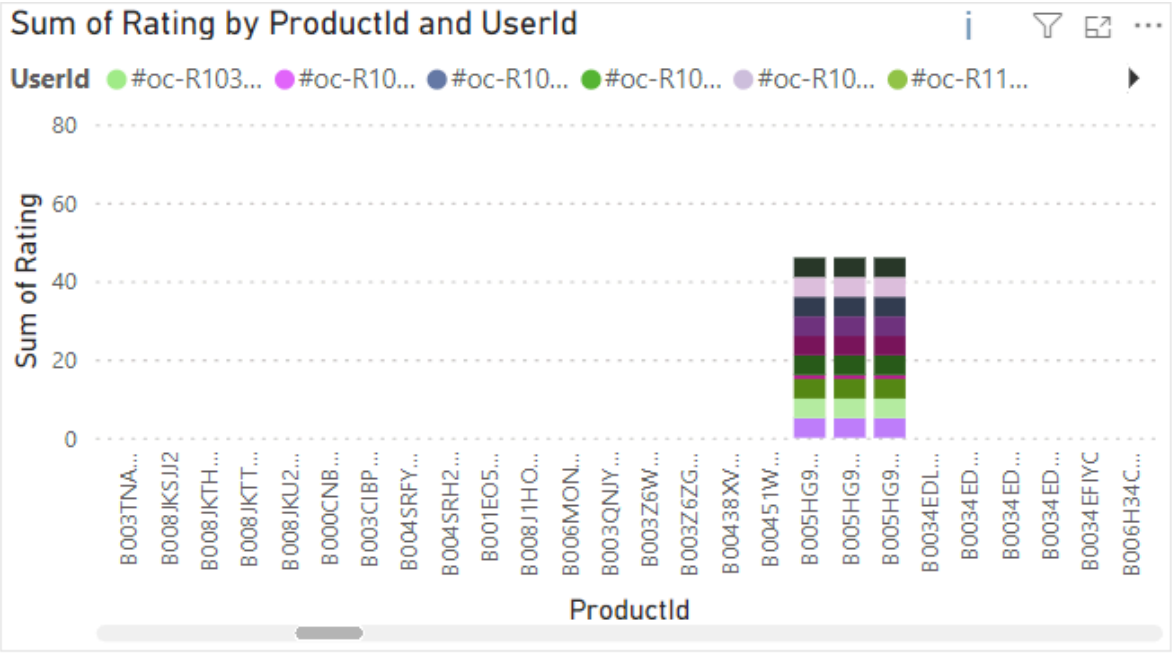




These plots show the distribution of ratings over years.



This plot shows the ratings (y-axis) given by users (in colored) to products (x-axis).



This column chart visualizes the distribution of ratings given by users for different products. This can help identify popular products or patterns in user ratings.

UserId	0006641040	141278509X	2734888454	2841233731	7310172001	7310
#oc-R103C0QSV1DF5E						
#oc-R109MU5OBBZ59U						
#oc-R10LFEMQEW6QGZ						
#oc-R10LT57ZGIB140						
#oc-R10UA029WVWIUI						
#oc-R115TNMSPFT9I7						
#oc-R119LM8D59ZW8Y						
#oc-R11D9D7SHXIJB9						
#oc-R11D9LKDAN5NQJ						
#oc-R11DNU2NBKQ23Z						
#oc-R11O5J5ZVQE25C						
Total	**4.35**	**5.00**	**3.50**	**5.00**	**4.75**	

This matrix represents average ratings for each product.

This word cloud visualizes the most common words or phrases in the summary of the reviews, providing insights into what users are saying about the products.

This KPI displays the average rating across all products.

Conclusion

By leveraging these big data techniques on the dataset and the analysis of user preferences and trends, the system empowers Amazon to deliver personalized recommendations, thereby fostering greater customer satisfaction and optimizing the efficacy of marketing efforts.