Section 1: Python Implementation

The installed Python packages are: NumPy, Pandas, Matplotlib, Jupyter, Scikit-learn, Seaborn, Mlxtend, Networkx, adjustText.

Task 1: Data Preprocessing

Modify the directory appropriately if using another testing environment. For example, the directories used in VSCode are **"./data/ecommerce_user_data.csv"** and **"./data/product_details.csv".**

Printing the head of the DataFrame (user_data) tied to **ecommerce_user_data.csv** gives this:

	UserID	ProductID	Rating	Timestamp	Category
0	U000	P0009	5	2024-09-08	Books
1	U000	P0020	1	2024-09-02	Home
2	U000	P0012	4	2024-10-18	Books
3	U000	P0013	1	2024-09-18	Clothing
4	U000	P0070	4	2024-09-16	Toys

Printing the head of the DataFrame (product_data) tied to **product_details.csv** gives this:

	ProductID	ProductName	Category
0	P0000	Toys Item 0	Clothing
1	P0001	Clothing Item 1	Electronics
2	P0002	Books Item 2	Electronics
3	P0003	Clothing Item 3	Electronics
4	P0004	Clothing Item 4	Electronics

We must prepare a DataFrame that has indexes UserID and columns ProductID. Each (index, column) would indicate a rating that the user gave to a unique product. Therefore, a pivot table with the appropriate parameters is created.

user_item_matrix = user_data.pivot_table(index='UserID', columns='ProductID', values='Rating')

However, we did not handle missing ratings in **user_item_matrix**. To make it simple for this assignment, we will treat missing ratings to products as unrated. The missing ratings will be assigned a value of 0.

user_item_matrix_filled = user_item_matrix.fillna(0)

To group and aggregate purchase behaviours per user and category, we can choose to group UserID and Category columns, and within a group, count the number of transactions and take the mean ratings of the transactions.

user_category_agg = user_data.groupby(['UserID', 'Category']).agg({'Rating': ['count', 'mean']}).reset_index()
user_category_agg.columns = ['UserID', 'Category', 'TotalInteractions', 'AverageRating']

user_category_agg shows us that users buy products in many different categories. Some users buy more or less from a certain category than other users.

Task 2: User-Based Collaborative Filtering (Cosine Similarity)

The first part of this task consists of implementing cosine similarity to measure user similarity. In this submission, we have subtracted the row mean from each rating in **user_item_matrix_filled**, to make a DataFrame named **user_item_matrix_mean_subtracted**. This transformation is done to normalize ratings by accounting for user biases.

Then, we use **user_item_matrix_mean_substracted** to compute a user-user cosine similarity as such:

similarity_matrix = cosine_similarity(user_item_matrix_mean_subtracted)
similarity_df = pd.DataFrame(similarity_matrix,
index=user_item_matrix_mean_subtracted.index,
columns=user_item_matrix_mean_subtracted.index)

A recommender system should find the top N similar users, then recommend products based on their ratings. The top N similar users for a certain user is found by taking the highest similarity matrix values in the certain user's row (excluding the certain user's column).

To recommend products to a certain user, we must:
- Get the ratings of the top N users
- Exclude the products already rated by the certain user
- Calculate the weighted sum of the ratings for each item
- Sort the recommended products in DESC order

What we mean by the weighted sum of the ratings is the rating prediction formula.

$$r_{xi} = \frac{\sum_{j \in N(i;x)} S_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} S_{ij}}$$

where:
S_{ij} = similarity between items i and j
r_{xj} = rating of user x on item j
N(i;x) = set of items rated by x that are similar to i

Recommending products requires a rating prediction, because we need some way to quantify how the user would rate the unrated item.

Given that we are able to recommend products, we are able to evaluate the accuracy of the recommendation system. In this submission, we calculate the mean of these metrics over ALL users:

- Precision@K
- Recall@K
- Coverage
- Diversity

Assuming that for each user, we recommend products based on the top 10 most similar users. To calculate Precision@K and Recall@K, we took the top 10 recommended products from the recommendations, and set a rating threshold of 4.5 (so, K=10). This ensures that only the best items are present in the home page of the e-commerce website.

Then, for coverage and diversity, we set a minimum rating of 3 because we consider anything **below it to be insufficient or problematic**. A rating of 3/5 is often interpreted as **"adequate"** or **"satisfactory"**. It's not perfect, but it's not failing either. By setting that minimum rating, we filter out options that may skew results, introduce bias, or fail to meet user expectations.

For **coverage**, this ensures that the content addresses a sufficient breadth of topics or user needs. For **diversity**, this ensures that there's a minimum level of representation across different groups, categories, or perspectives.

The results we got were:
Mean Precision: 0.7680
Mean Recall: 0.9619
Mean Coverage: 0.3442
Mean Diversity: 0.8727

Precision@K is calculated as:

$$Precision@K = \frac{Number\ of\ relevant\ items\ in\ K}{Total\ number\ of\ items\ in\ K}$$

where we consider items equal or above the rating threshold of 4.5 as relevant. A mean Precision@10 of 0.7680 signifies that 76.8% of the top 10 recommended products are relevant to the users. This is good, since the recommender system is able to recommend products that the user is likely to find relevant. However, there is still room for improvement to get to 100%.

Recall@K is calculated as:

$$Recall@K = \frac{Number\ of\ relevant\ items\ in\ K}{Total\ number\ of\ relevant\ items}$$

where we consider items equal or above the rating threshold of 4.5 as relevant. A mean Recall@10 of 0.9619 indicates that 96.19% of the relevant items in the user's recommended products (NOT ONLY TOP 10 RECOMMENDED, IT'S ALL THE RECOMMENDED PRODUCTS INSTEAD) are recommended by the system. This is a very high recall score, because it shows that the system is doing a good job at covering most of a user's relevant items. However, this high recall might come at the cost of lower precision, as the higher K parameter might lead to the system to recommend less relevant items.

Coverage is calculated as:

$$Coverage = \frac{Number\ of\ satisfactory\ recommended\ products}{Total\ number\ of\ products}$$

where satisfactory recommended products are defined as recommended products that have a predicted rating of at least 3. This metric measures how much of the product catalog is being recommended (above a rating threshold). A coverage of 0.3442 signifies that 34.42% of the items in the product catalog are recommended to users. This suggests that the recommendations are somewhat limited in terms of the product space covered. Increasing coverage may increase user satisfaction, as they would have more variety to choose from.

Diversity is calculated as:

$$Diversity = 1 - \frac{1}{N^2 - N} \sum_{i,j=1}^{N} S_{ij},\ (i \neq j)$$

where satisfactory recommended products are defined as recommended products that have a predicted rating of at least 3. The N^2 - N part signifies the number of recommended product pairs (excluding pairs with itself). S_{ij} is an item-item cosine similarity matrix of recommended products. We take the total sum of its values, excluding the cases where product i is the same as product j.

A diversity of 0.8727 means that the recommended items are fairly diverse. The system is not recommending very similar items to the user, which leads to the user being exposed to new and varied products. As a result, this will lead to increased engagement and user satisfaction.

Another metric that evaluates the accuracy of the recommendations is the **Mean Average Precision at K (MAP@K)**. It is defined as:

$$MAP@K = \frac{1}{U} \sum_{u=1}^{U} AP@K_u$$

where U is the total number of users in the dataset.

AP@K$_u$ is the average precision at K for a given user, which is defined as:

$$AP@K_u = \frac{\sum_{i=1}^{K} Precision@i}{K}$$

By checking for the top 10 products (K = 10), with a rating threshold of 4.5, we got this result:
Mean Average Precision (MAP): 0.9276

This MAP is very high and indicates that on average, the top recommendations are very good and relevant to users. The system is effectively ranking items that are relevant to the user at the top of the recommendation list.

Task 3: Association Rule Mining (Apriori)

We convert the user-product purchase data into transactions, because the **Apriori algorithm** needs a one-hot encoded transaction DataFrame.

Our code converts the **user_data** DataFrame into a **transactions** DataFrame, where each column is tied to a user, and contains the products that the user rated. We can treat each column as a transaction. We use one-hot encoding on **transactions** to get an array where each entry is a one-hot encoded transaction. Finally, convert that array into a **df_trans** DataFrame, where the columns are the productIDs.

We use the Apriori Algorithm to find frequent itemsets, with a minimum **support** of 0.10. Using the frequent itemsets, we create association rules and exclude any rules that have a **lift** under 1.2 and **confidence** under 0.5. This filtering ensures that only **strong and meaningful relationships** between products are considered.

Support measures how frequently an itemset appears in the dataset. A support of 0.10 means the itemset occurs in at least 10% of all transactions. This threshold ignores rare combinations that are statistically insignificant.

Confidence measures how often items in the consequent appear in transactions that contain the antecedent. A confidence of 0.5 means that in at least 50% of the cases where the

antecedent appears, the consequent does too. Low-confidence rules are less reliable for predicting associations.

Lift measures how much more likely the consequent is to appear given the antecedent, compared to it appearing by chance. A lift greater than 1 suggests a positive correlation between items. We use a minimum lift of 1.2 to focus on rules that show a **meaningful increase in likelihood**.

To get the product bundles, we create a graph of nodes and edges using the association rules. The **rules** DataFrame contains every association rule. See **Appendix 1** for the network graph of product bundles. See **Section 2 Q2** for a possible interpretation of them.

Task 4: Analysis & Visualization

See **Appendix 2** for the user similarity heatmaps. In this heatmap, we can see a few dark blue squares, which indicate a strong user to user similarity between a pair of users. Some squares are light blue (average user to user similarity), but most squares are in the green to yellow color range (weak user to user similarity).

See **Appendix 3** for the most frequent itemsets. Support is used, because it is the metric that defines how frequently an itemset appears in the dataset.

Next, we want to segment users based on their similarity scores using K-Means Clustering. We need to determine the optimal number of clusters before applying this clustering method.

The similarity matrix was scaled using **MinMaxScaler** to normalize values between 0 to 1. **K-Means Clustering** was run using a K range between 2 to 6, so that we could plot the inertia (distortion) against K. We noticed that the **elbow point** in the distortion plot was not clearly defined (see **Appendix 4**).

So, we decided to use another metric called the **Silhouette Score**. This score measures how similar an object is to its own cluster compared to other clusters, with a value between -1 to 1. A higher value indicates a better-defined cluster. In the silhouette score plot (**Appendix 5**), we determined the optimal number of clusters to be K = 2.
After applying K-Means Clustering for K = 2, we found these users and clusters:
Cluster 0:
['U003', 'U004', 'U005', 'U007', 'U008', 'U012', 'U013', 'U014', 'U015', 'U019', 'U020', 'U021', 'U023', 'U025', 'U026', 'U030', 'U032', 'U033', 'U036', 'U040', 'U041', 'U044', 'U046', 'U047', 'U048']

Cluster 1:
['U000', 'U001', 'U002', 'U006', 'U009', 'U010', 'U011', 'U016', 'U017', 'U018', 'U022', 'U024', 'U027', 'U028', 'U029', 'U031', 'U034', 'U035', 'U037', 'U038', 'U039', 'U042', 'U043', 'U045', 'U049']

To find the top-5 product recommendations within a user group (cluster), we iterated through a cluster's users to run the recommendation algorithm. The recommendation algorithm took into account the top 10 most similar users to the current iteration's user. Then, we appended the top 5 ratings into a dictionary, defined as such:

```
{
        <productID> : {'predicted_ratings': [ ] },
        …
}
```

Appending the predicted ratings across multiple users for a certain productID will be useful soon. The cluster users iteration is now done. Then, we want to iterate through a cluster's product dictionary of key <productID> to calculate the average of the predicted ratings array. Doing so will result in a dictionary defined as such:

```
{
        <productID> : <avg_rating>,
        …
}
```

Finally, append this cluster dictionary into an array of dictionaries where an entry is defined as such:

```
{
        <cluster#> : { <productID> : <avg_rating>, … },
        …
}
```

We can use this array of dictionaries to print top-5 product recommendations for each user group (cluster). Here are the results:
Cluster 0:
Product ID: P0040 -- Average rating: 5.0
Product ID: P0008 -- Average rating: 5.0
Product ID: P0016 -- Average rating: 5.0
Product ID: P0095 -- Average rating: 5.0
Product ID: P0048 -- Average rating: 5.0

Cluster 1:
Product ID: P0036 -- Average rating: 5.0
Product ID: P0049 -- Average rating: 5.0
Product ID: P0024 -- Average rating: 5.0
Product ID: P0078 -- Average rating: 5.0
Product ID: P0016 -- Average rating: 5.0

The presence of P0016 in Cluster 0's and Cluster 1's top-5 may suggest it's a universally appealing product, cutting across user types. Products like P0036 and P0049 seem more tailored to the tastes or behaviors specific to Cluster 1. The separation of top products by cluster indicates that this clustering method effectively differentiates user preferences, making the recommendations more personalized.

Section 2: Conceptual Questions

Q1. How does the sparsity of the data affect your recommender system's performance?

The sparsity of the data affects our recommender system's performance in many ways. Fundamentally, collaborative filtering depends on a user-item matrix. If most entries are missing, it becomes difficult to identify meaningful patterns.

First, sparse data reduces the accuracy of the recommender system's predictions, since it leads to unreliable similarity calculations between users. We say between users, because user-user collaborative filtering was used for this assignment.

Second, we must remember that recommender systems are affected by the cold start problem. Without enough historical data, they struggle to make relevant suggestions.

Third, there is lower diversity in recommendations, because sparse data can cause recommender systems to favor popular items. For example, if users rate a set of 10-20 products, the recommender system will tend to recommend that set of products. It would be difficult for it to recommend products outside of that set.

Q2. What kinds of product bundles were discovered in the association rules?

In the network graph (see **Appendix 1**), we can see that there are clusters of nodes. A cluster of nodes is potentially a product bundle.

We list the clusters below:

1. {P0043, P0089}
2. {P0065, P0076}
3. {P0013, P0079}
4. {P0004, P0011, P0077}
5. {P0003, P0015, P0024, P0033, P0039, P0042, P0051, P0070}

We can describe the first 4 cluster's categories as such:

1. {Clothing, Beauty}
2. {Clothing, Toys}
3. {Clothing, Electronics}
4. {Electronics, Electronics, Electronics}

{P0043, P0089} → Clothing & Beauty products, which could indicate fashion-related purchases where customers buy apparel along with beauty products.

{P0065, P0076} → Clothing & Toys, suggesting that children's clothing might be bundled with toys, or fashion-conscious consumers also shop for gifts.

{P0013, P0079} → Clothing & Electronics, possibly indicating wearable tech or electronic accessories related to fashion.

{P0004, P0011, P0077} → Exclusively Electronics, suggesting a bundle of tech-related products, such as accessories, gadgets, or complementary devices.

The final cluster is more complex and may require further analysis to determine its underlying theme. Given its size, this cluster might represent a diverse shopping cart pattern or a category with many interrelated products.

Q3. What improvements would you recommend for a real e-commerce system using your approach?

In a real e-commerce system, there would be a lot more users which would reduce the dataset sparsity. However, we could reduce sparsity even more by encouraging more interactions, such as:

- Offering discounts or incentives for reviewing/rating products
- Implement features like “Did you find this useful?” to gather implicit feedback (clicks, views, time spent, cart additions, etc.)

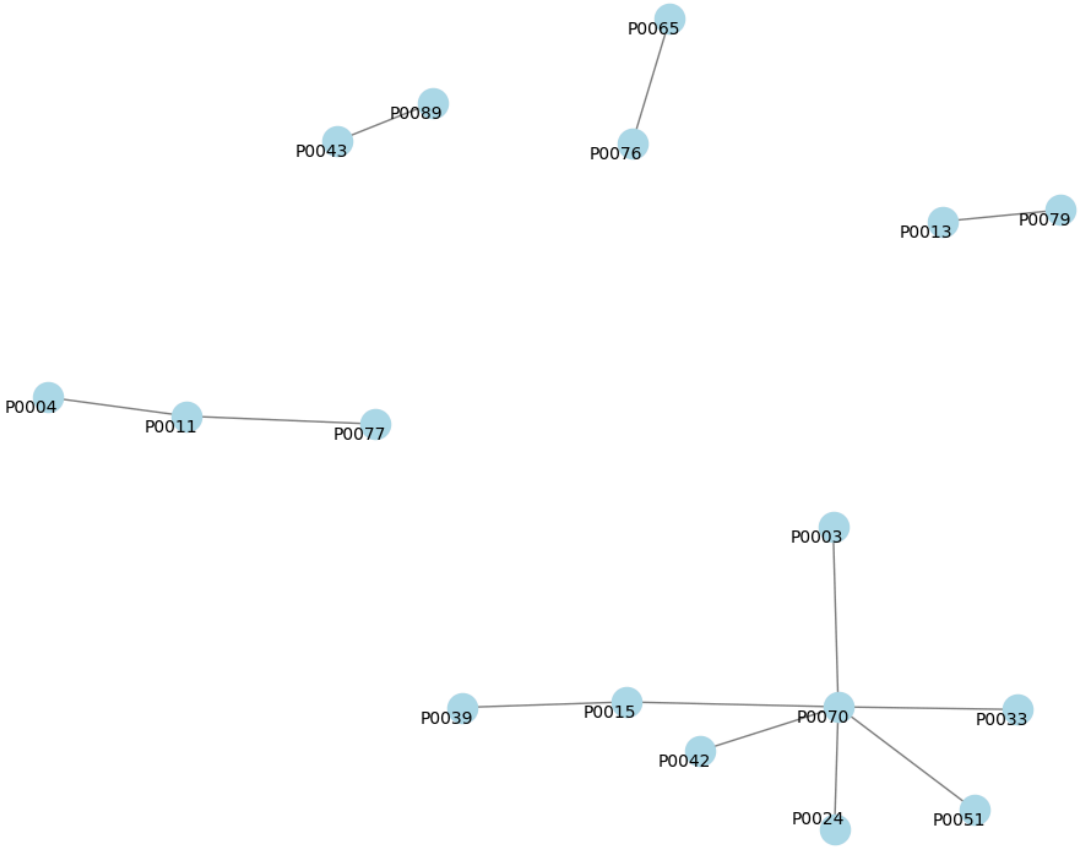
We should compute a cosine similarity matrix per category instead of a global one. Computing a single similarity matrix for all products can dilute the relevance of the recommendations. By computing category-specific matrices, we can improve accuracy by only comparing items within a certain category.

We should also store the similarity matrices for each category in a database, in order to optimize resource usage. This would increase scalability. Whenever the recommender system generates recommendations, the appropriate matrix would be selected.

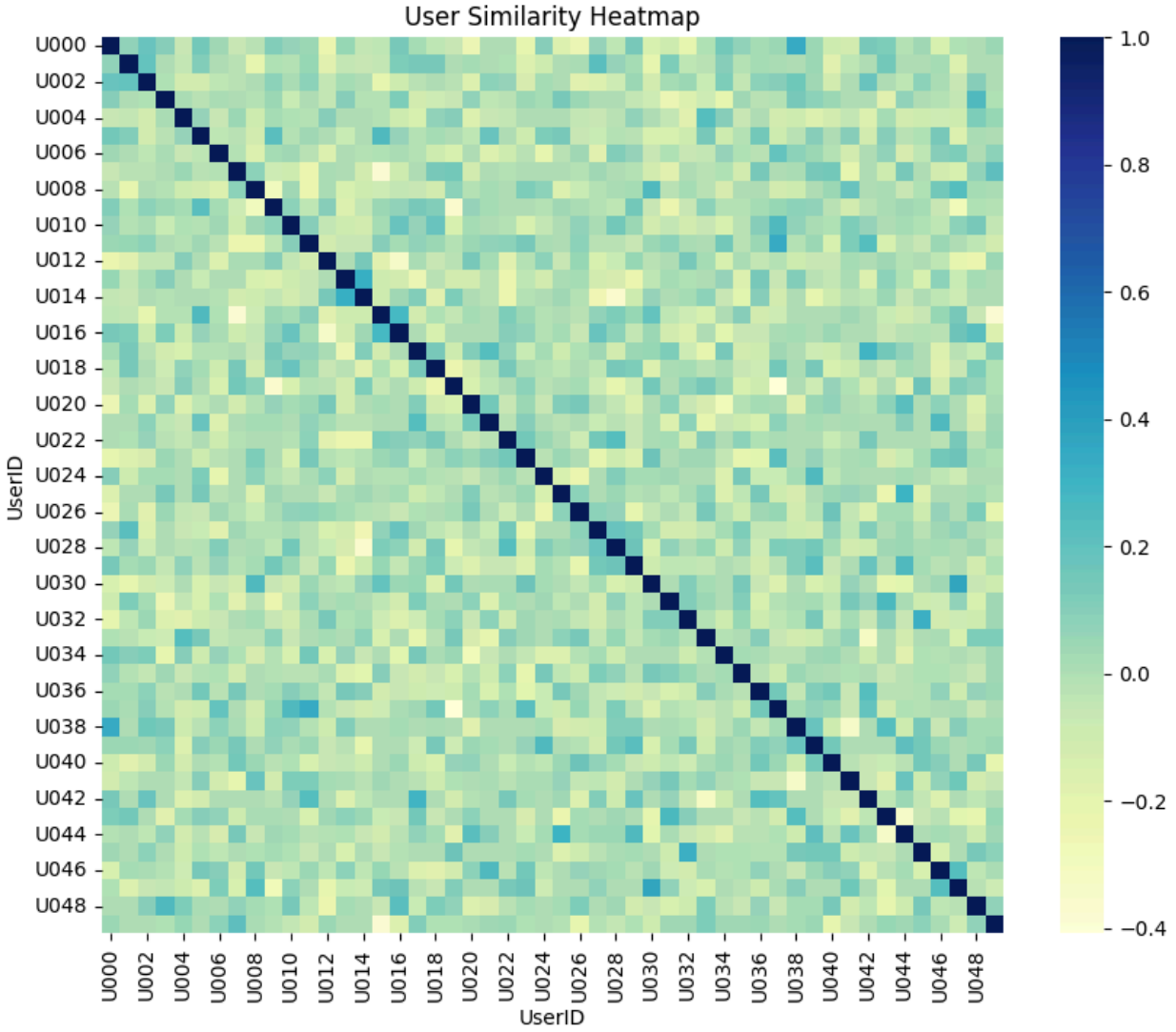
Finally, we recommend that item-item collaborative filtering should be used instead of user-user, because it reduces computational complexity. User preferences change frequently, but item characteristics remain stable, so the recommendations would be more stable.

Appendix

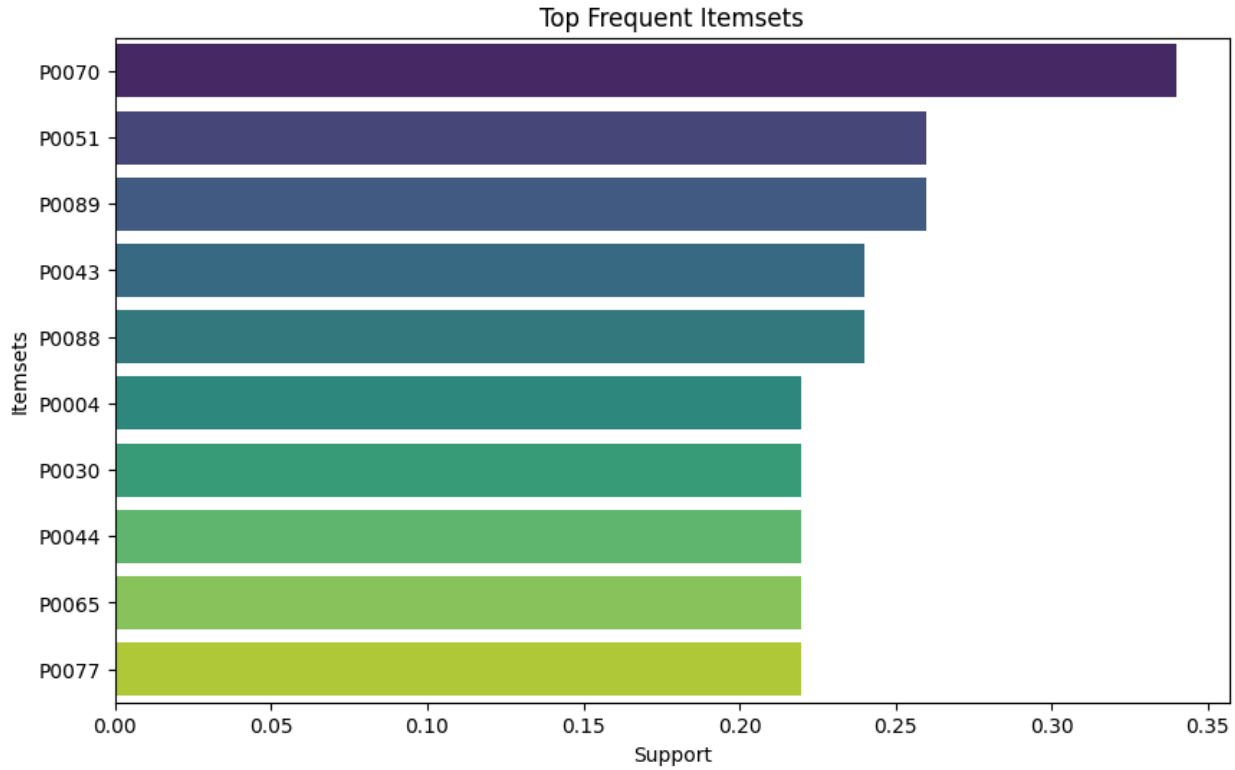
Appendix 1: Product Bundles



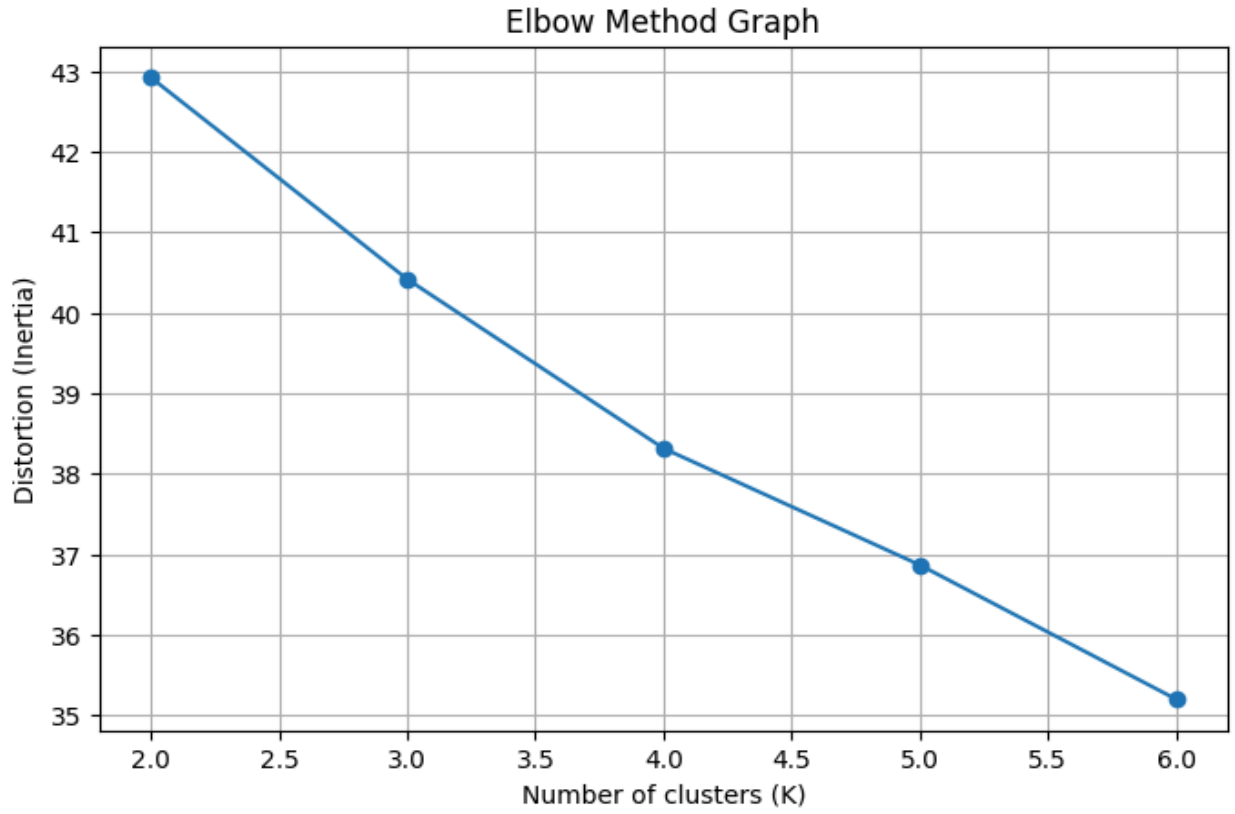
Appendix 2: User Similarity Heatmap



Appendix 3: Top Frequent Itemsets



Appendix 4: Elbow Method Graph



Appendix 5: Silhouette Score Graph