Introduction

Product recommender systems, or recommendation systems, as they're also known, are ubiquitous on e-commerce websites these days. They're relatively simple to create, and even pretty basic ones can give striking results. In this project, we will find how to knock up a simple product recommender system in Python solely using Pandas, NumPy, and Cosine Similarity.

Product recommendation systems can be created using various data science techniques. They can make use of user data to provide personalised recommendations, make content-based recommendations using Natural Language Processing (NLP), utilise data on the user's browsing history, adopt hybrid approaches, or make generalised recommendations based on anonymised user data via a technique known as collaborative filtering.

Collaborative filtering is probably the most widely used algorithm for creating product recommender systems in online retailing. This algorithm is comparatively simple to implement and generates relevant and accurate recommendations. This improves the user experience by presenting products popular with other customers, which can help boost basket sizes, average order value, sales, and revenue.

Cosine similarity is a widely used technique in recommendation systems to identify items that are similar to each other based on user purchase history or item attributes. A product recommendation system calculates the similarity between the products by measuring the cosine of the angle between their feature vectors. The feature vectors can be created using attributes such as product category, price, popularity, etc.

When a user interacts with the system, the system can identify the products that the user has purchased or viewed before and use cosine similarity to identify other products that are similar to those the user has shown interest in. The system then recommends those similar products to the user.

Cosine similarity can also be used in collaborative filtering techniques widely used in recommendation systems. Collaborative filtering involves predicting user preferences based on the likes of other similar users. Cosine similarity can be used to identify the similarity between users based on their purchase history or item ratings and then recommend items to a user based on what other similar users have liked or purchased.

Importing Dependencies and Loading

They are importing dependencies like Pandas, NumPy and Cosine Similarity from sk-learn. Pandas and NumPy are popular Python libraries used for data manipulation and analysis. Cosine Similarity is a function from the scikit-learn (sk-learn) library that measures the similarity between two non-zero vectors in a high-dimensional space. These dependencies are imported into our Jupyter Notebook to facilitate data manipulation and analysis and compute the similarity between vectors to build recommendation systems.

```
In [2]: df = pd.read_excel('Online Retail.xlsx')
```

```
In [3]: df.head()
```

Out[3]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

The Dataset is a very popular dataset called ‘Online Retail’ found in the UCI Machine Learning Repository. Now, we use basic Python commands to read the dataset and see what it looks like.

Preparing the Dataset

After completing fundamental exploratory data analysis (EDA), we discover that there are 1,33,361 null values in *‘CustomerID’*. We have chosen to drop these as they won’t affect the original dataset much. We can see that the dataset has eight columns and more than 500000 rows. Most of the columns have integer data types.

Creating the Customer-Item Matrix

Now, we used pandas’ pivot_table() function to create a pivot table consisting of *CustomerID*, *StockCode* and Quantity. Using this pivot table, we created a customer-item matrix with a shape of 4339 rows and 3665 columns.

```
In [13]: customer_item_matrix.loc[12481:].head()
```

Out[13]:

StockCode	10002	10080	10120	10125	10133	10135	11001	15030	15034	15036	...	90214V	90214W	90214Y	90214Z	BANK CHARGES	C2	DOT	M	PADS
CustomerID																				
12481.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	36.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12483.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12484.0	NaN	NaN	NaN	NaN	NaN	NaN	16.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12488.0	NaN	NaN	NaN	NaN	NaN	10.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12489.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 3665 columns

Further, we apply One Hot Encoding, replacing anything numerical above zero to one and below zero to zero. After this operation, our customer-item matrix looks as follows.

```
In [19]: 1 customer_item_matrix.loc[12481:].head()
```

Out[19]:

StockCode	10002	10080	10120	10125	10133	10135	11001	15030	15034	15036	...	90214V	90214W	90214Y	90214Z	BANK CHARGES	C2	DOT	M	PADS	P
CustomerID																					
12481.0	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	
12483.0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
12484.0	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	
12488.0	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
12489.0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	

5 rows × 3665 columns

User-based Collaborative Filtering

We pass the customer-item matrix to the Cosine Similarity function and further convert it to a data frame to make a user-to-user similarity matrix.

```
In [21]: 1 user_to_user_sim_matrix.head()
```

Out[21]:

	0	1	2	3	4	5	6	7	8	9	...	4329	4330	4331	4332	4333	4334	4335	
0	1.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.00
1	0.0	1.000000	0.063022	0.046130	0.047795	0.038484	0.0	0.025876	0.136641	0.094742	...	0.0	0.029709	0.052668	0.0	0.032844	0.062318	0.0	0.11
2	0.0	0.063022	1.000000	0.024953	0.051709	0.027756	0.0	0.027995	0.118262	0.146427	...	0.0	0.064282	0.113961	0.0	0.000000	0.000000	0.0	0.00
3	0.0	0.046130	0.024953	1.000000	0.056773	0.137137	0.0	0.030737	0.032461	0.144692	...	0.0	0.105868	0.000000	0.0	0.039014	0.000000	0.0	0.06
4	0.0	0.047795	0.051709	0.056773	1.000000	0.031575	0.0	0.000000	0.000000	0.033315	...	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.00

5 rows × 4339 columns

Consequently, we replace the column names and row names with *CustomerID*.

```
In [25]: 1 user_to_user_sim_matrix.head()
```

Out[25]:

CustomerID	12346.0	12347.0	12348.0	12349.0	12350.0	12352.0	12353.0	12354.0	12355.0	12356.0	...	18273.0	18274.0	18276.0	18277.0	18278
CustomerID																
12346.0	1.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	...	0.0	0.000000	0.000000	0.0	0.00000
12347.0	0.0	1.000000	0.063022	0.046130	0.047795	0.038484	0.0	0.025876	0.136641	0.094742	...	0.0	0.029709	0.052668	0.0	0.0328
12348.0	0.0	0.063022	1.000000	0.024953	0.051709	0.027756	0.0	0.027995	0.118262	0.146427	...	0.0	0.064282	0.113961	0.0	0.00000
12349.0	0.0	0.046130	0.024953	1.000000	0.056773	0.137137	0.0	0.030737	0.032461	0.144692	...	0.0	0.105868	0.000000	0.0	0.0390
12350.0	0.0	0.047795	0.051709	0.056773	1.000000	0.031575	0.0	0.000000	0.000000	0.033315	...	0.0	0.000000	0.000000	0.0	0.00000

5 rows × 4339 columns

Using the user-to-user similarity matrix, we can make recommendations. We pick a random *CustomerID* and feed it to generate a set with the *StockCode* that has been bought by the randomly picked customer and name the set item_bought_by_A.

Next, we pick another random *CustomerID*, do the same as above, and name the set items_bought_by_B. We now have two sets with *StockCode* of items bought by two customers. We subtract both of them (item_bought_by_A - item_bought_by_B) and find the items recommended to customer B.

```
In [31]: items_to_recommend_User_B = items_bought_by_A - items_bought_by_B

In [32]: items_to_recommend_User_B
Out[32]: {20615,
          20652,
          21171,
          21832,
          21864,
          21908,
          21915,
          22348,
          22412,
          22620,
          '79066K',
          '79191C',
          '84086C'}
```

Using the *StockCode*, we can learn more about the item, and we have displayed it.

```
In [33]: df1.loc[
             df['StockCode'].isin(items_to_recommend_User_B),
             ['StockCode','Description']
         ].drop_duplicates().set_index('StockCode')
```

Out[33]:

StockCode	Description
21832	CHOCOLATE CALCULATOR
21915	RED HARMONICA IN BOX
22620	4 TRADITIONAL SPINNING TOPS
79066K	RETRO MOD TRAY
21864	UNION JACK FLAG PASSPORT COVER
79191C	RETRO PLASTIC ELEPHANT TRAY
21908	CHOCOLATE THIS WAY METAL SIGN
20615	BLUE POLKADOT PASSPORT COVER
20652	BLUE POLKADOT LUGGAGE TAG
22348	TEA BAG PLATE RED RETROSPOT
22412	METAL SIGN NEIGHBOURHOOD WITCH
21171	BATHROOM METAL SIGN
84086C	PINK/PURPLE RETRO RADIO

Item-based Collaborative Filtering

We use the customer-item matrix to process Item-based collaborative filtering and follow the same steps as above, but instead, we use the transpose of the customer-item matrix. And then, pass it to the Cosine Similarity function and convert it to a data frame. After that, we can change the column and row names to *StockCode* instead of *CustomerID*.

```
In [36]: item_item_sim_matrix.head()
```

Out[36]:

StockCode	10002	10080	10120	10125	10133	10135	11001	15030	15034	15036	...	90214V	90214W	90214Y	90214Z	BANK CHARGE
StockCode																
10002	1.000000	0.000000	0.094868	0.090351	0.062932	0.098907	0.095346	0.047673	0.075593	0.090815	...	0.0	0.0	0.0	0.0	0
10080	0.000000	1.000000	0.000000	0.032774	0.045655	0.047836	0.000000	0.000000	0.082261	0.049413	...	0.0	0.0	0.0	0.0	0
10120	0.094868	0.000000	1.000000	0.057143	0.059702	0.041703	0.060302	0.060302	0.095618	0.028718	...	0.0	0.0	0.0	0.0	0
10125	0.090351	0.032774	0.057143	1.000000	0.042644	0.044682	0.043073	0.000000	0.051224	0.030770	...	0.0	0.0	0.0	0.0	0
10133	0.062932	0.045655	0.059702	0.042644	1.000000	0.280097	0.045002	0.060003	0.071358	0.057152	...	0.0	0.0	0.0	0.0	0

5 rows × 3665 columns

This generated data frame is called the item-to-item similarity matrix. Like User-based Collaborative filtering, we pick a random *StockCode*, and feed it to the system to find out about other top 10 similar items. So, this will return the *StockCode* of the items.

```
In [37]: top_10_similar_items = list(item_item_sim_matrix.loc[23166].sort_values(ascending=False).iloc[:10].index)

In [38]: top_10_similar_items

Out[38]: [23166, 23165, 23167, 22993, 23307, 22722, 22720, 22666, 23243, 22961]
```

We can use those *StockCode* to learn more about the description of the items from the dataset.

```
In [39]: df.loc[
             df['StockCode'].isin(top_10_similar_items),
             ['StockCode', 'Description']
         ].drop_duplicates().set_index('StockCode').loc[top_10_similar_items]
```

Out[39]:

StockCode	Description
23166	MEDIUM CERAMIC TOP STORAGE JAR
23165	LARGE CERAMIC TOP STORAGE JAR
23167	SMALL CERAMIC TOP STORAGE JAR
22993	SET OF 4 PANTRY JELLY MOULDS
23307	SET OF 60 PANTRY DESIGN CAKE CASES
22722	SET OF 6 SPICE TINS PANTRY DESIGN
22722	NaN
22720	SET OF 3 CAKE TINS PANTRY DESIGN
22720	NaN
22666	RECIPE BOX PANTRY YELLOW DESIGN
23243	SET OF TEA COFFEE SUGAR TINS PANTRY
22961	JAM MAKING SET PRINTED

Conclusion

It can be concluded that a simple recommendation system can be made just using Pandas, NumPy and Cosine Similarity. We have learnt how to do data analysis for recommendation systems and make a recommendation system using very few libraries. Based on the need of the client or the business, we can use TensorFlow and Deep Learning to create an improved version of the present recommendation system.

References

https://archive.ics.uci.edu/ml/datasets/online+retail