Santander Product Recommendation

Proposal

Client/Problem

The client for this project is Santander, a bank that has asked Kagglers to help increase the effectiveness of their current recommendation capabilities. Santander is looking to offer personalized product recommendations to their customers, but they have a fairly big problem: Under the current system, a small number of their customers are seeing lot of recommendations, but a large portion of their customers rarely see any recommendations at all. This has resulted in a lopsided customer experience and something that Santander hopes to have resolved. I found this idea very intriguing considering the state of business right now. Recommendation has become one of the biggest assets to a company and can make or break its customer satisfaction. Amazon, Netflix, Spotify, Walmart, and many others are pushing the art of recommendation forward an effort to set themselves apart and engaged customers in a way in which they have not previously been engaged. I hope to tackle this problem in a way that is efficient and effective; hopefully leading to a better recommendation system than existed at Santander previously.

Data

Quoting from the Kaggle competition page:

> *"In this competition, you are provided with 1.5 years of customers behavior data from Santander bank to predict what new products customers will purchase. The data starts at 2015-01-28 and has monthly records of products a customer has, such as "credit card", "savings account", etc. You will predict what additional products a customer will get in the last month, 2016-06-28, in addition to what they already have at 2016-05-28. These products are the columns named: ind_(xyz)_ult1, which are the columns #25 - #48 in the training data. You will predict what a customer will buy in addition to what they already had at 2016-05-28."*

As mentioned, the data contains whether a product is already owned (Such as a credit card or savings account), but it also includes valuable customer demographic information.

Approach

For the approach, I will implement a few different types of recommendation engines and see which ones are the most effective. Specifically, I will be using a collaborative filtering approach. I am actually going to refine my approach for the sake of time as well. My objective for this project is not to predict which new products a customer will get in the month after the date range of the dataset, but instead, my goal is to create a recommendation engine that will be able to recommend products for customers who have similar purchasing behaviors.

Deliverables

The deliverables for this project will include my code along with annotated explanations of each step in Jupyter Notebooks. I will also likely include a slide deck with visualizations and and consolidate my process and findings in easy-to-understand terms.

Data Wrangling/Cleaning

Data wrangling for this project was fairly straightforward. The project came from a Kaggle competition, so the data was made available to me immediately. The bulk of the work during this section of the project was dedicated to cleaning the data. The first thing that I had to do prior to cleaning was getting the dataset to a manageable size. During the first import, I decided to first limit the rows to 7 million, then from there I took a random sample of 1 million to work with.

Initial Data Cleaning

From there, I decided to translate the column names from Spanish to English to allow the reader to better follow along and understand my work. Next, I noticed that there were a lot of columns that needed to be numeric data types, but were not. Those were then translated with the pd.to_numeric() function. I then noticed that I had a lot of columns that contain NaN values, which needed to be taken care of. The date and customer_code columns each contained a small number of NaN values (Less than 10). For those, I decided it would be best to just simply drop the rows that contained the NaNs.

Next, I found two columns (last_date_primary_customer and spouse_index) of which more than 50% of the rows contained NaN values. Since the columns contained such a high number of NaNs, I decided it would be best to just drop those entire columns from the dataset altogether.

For first_contract_date, I noticed there were quite a few NaNs present (Nearly 7,000), but instead of dropping the rows altogether, I decided to replace them with the median first_contract_date value.

For the NaNs in new_customer_index, I decided to check to see how “new” these customers were by grouping those rows by customer_code and seeing how long their accounts had been active (The result was 6 months). Since we can consider those customers to be new, I decided to change all NaN values to 1 to identify them as such.

I then noticed that customer_seniority contained nearly an identical number of NaN values as new_customer_index. What this told me, is that those customers were probably the same customers as the NaN rows in new_customer_index. Therefore, I decided to change the null values in customer_seniority to 0 to reflect the fact that these customers were new. From there, I dropped both the address_type and province_code columns because I deemed the information in those columns to be irrelevant for the purposes of this project.

At this point, I began to notice a pattern. A lot of the columns contained somewhere around 6800 NaN values. This told me that these were probably just bad rows across most of the dataset. What I decided to do was drop those NaN values from each of the columns that contained them in one fell swoop.

I see the data for this project in two parts: the customer demographic data and the product data. At this point in the process, all but two of the demographic columns’ NaN values were taken care of.

The household_gross_income column contained a lot of NaN values as well. Instead of just putting in the median income for the entire amount in the for the NaNs, I decided to group all non-NaN household_gross_income rows by province_name and take the median of that groupby object and input the median income for each province_name in for the NaN values.

By now, I have finally taken care of all NaN values for the customer demographic columns. What I needed to do at this point is take care of the NaN values for the product columns. Luckily, for those columns, there were not a lot of NaN values at all (Between 1 or 44 for each column), so what I decided to do for the purposes of this project, is just change all of those NaN values to 0. And with that, I was able to get rid of all remaining NaN values in the entire dataset. The next step is doing some additional data cleaning that will help us get rid of any bad data present in the dataset.

Eliminating Bad Data

What I decided to do from here was go through, column by column, and see what unique values were present for each. I found that nearly half of the columns in the entire dataset contained either bad data or incorrectly formatted data. For example, the first one I looked at was primary_customer_index which should contain a value of either 1 or 99 and nothing else. Instead, this is what I found:

```
pd.Series([i for i in df.primary_customer_index]).value_counts()
```

```
1.0                  880975
 1                   110833
99.0                   1152
99                      109
0.0                       9
0                         1
 LENCIA"                  1
                          1
02 - PARTICULARES         1
dtype: int64
```

Clearly, this needed to be cleaned up. I began by first converting the entire column to numeric values. This left me with only 1, 99 and 0. But again, we only need 1 and 99, so I changed every instance of 0 (Which only ended up being 10 rows) to the most popular option (1).

My next task was taking care of the bad data in province_name. In this column, there should only be names of cities that customers live in. Instead, we had a lot of bad values that made no sense included. I took care of that by first changing the data type to a string. Then, I dropped all NaN values. After that, I created a subset that contained only characters in each unique value that were alphabetic. I filtered the data frame by this subset, and thus eliminated all bad data in the province_name column.

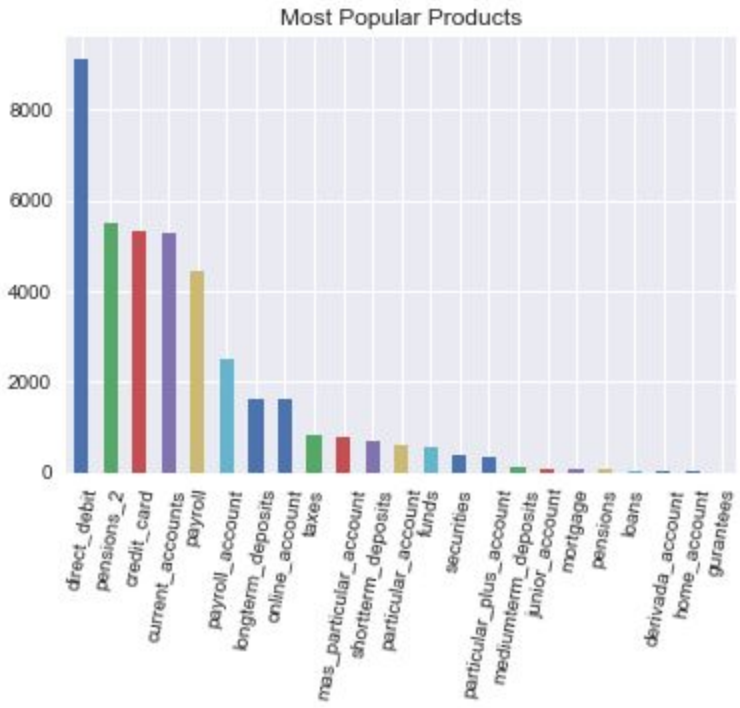
I repeated this process for the rest of the columns with bad data. Most of them were straightforward and only required the data type to be convert to numeric values.

Data Cleaning: Final Steps

For the final step of the data cleaning process, I needed to create a new column that would essentially let me know whether a customer added, dropped or maintained a product/service in each unique billing cycle. The first thing I had to do was convert all of the feature columns (Products offered by the bank) to numeric values. I then had to create an object, unique_months that included all unique months in the date column. Then, I wrote a function that would label whether a customer added, dropped or maintained a product in a given month. Then, I applied the function to the data frame and took out every instance in which a customer maintained a product. I did that because I am only interested in seeing whether a customer added or dropped. Finally, I saved the newly cleaned dataset and converted it to a csv file to allow myself to easily upload it for future notebooks.

Exploratory Data Analysis

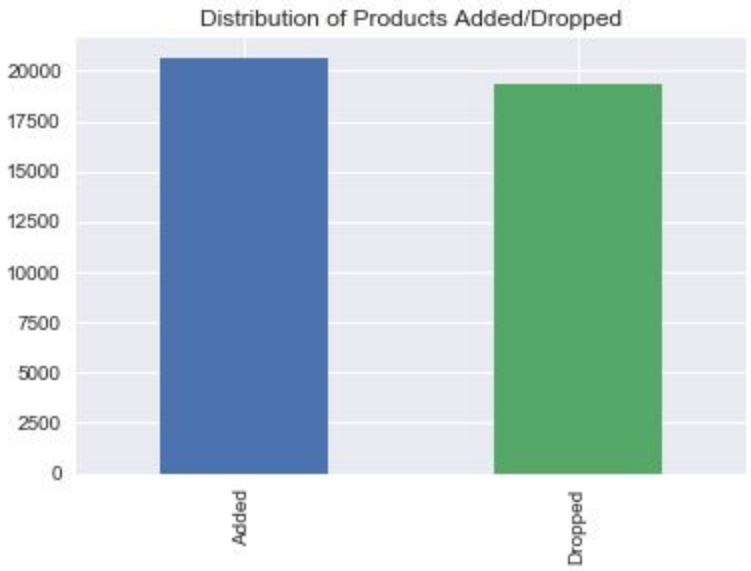
The next step in the process is to explore the data in order to see if I could identify any interesting trends or insights prior to running the recommendation engine. I started by running the '.describe()' method on the data frame to see if anything stood out. Nothing seemed out of the ordinary to me, so I moved forward with plotting some bar plots and histograms on all of the columns in the data frame. Here, I will outline some of the insights that interested me initially.



As you can see, the most popular products are the following:

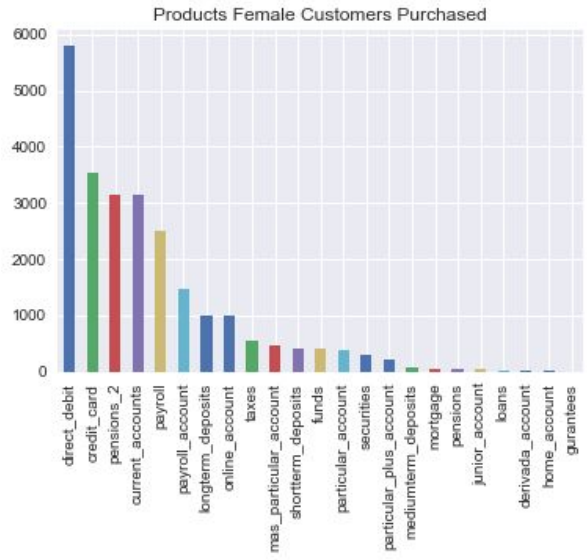
1. Direct Debit
2. Pensions
3. Credit Card
4. Current Accounts
5. Payroll

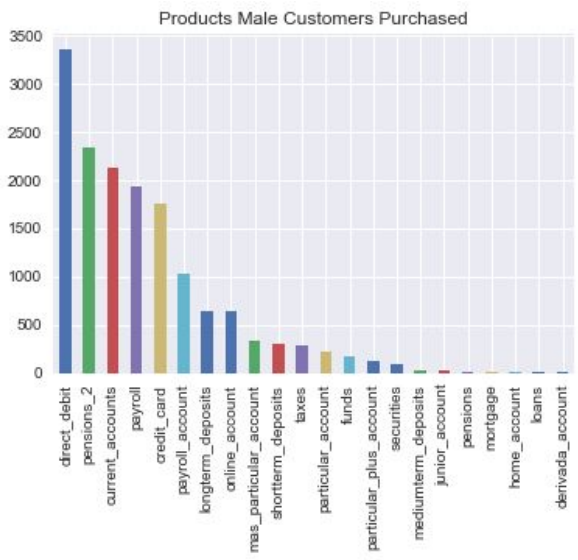
Overall, this is not exactly surprising - I mostly just wanted to show which products the bank can depend on as its most reliably popular ones.



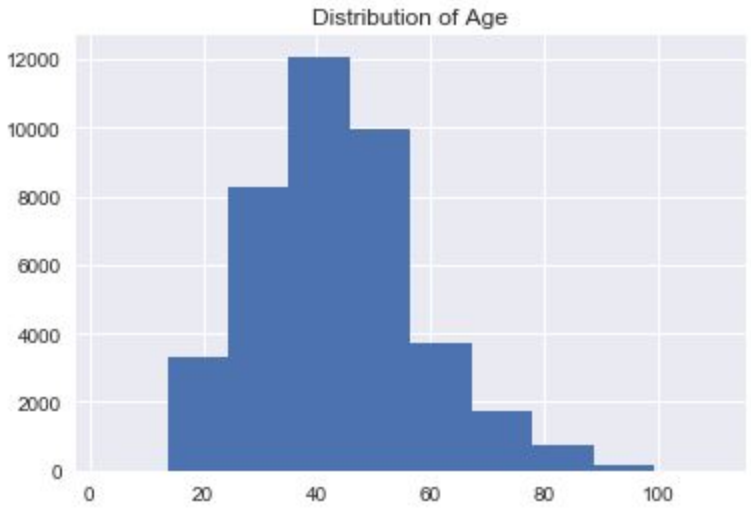
Another thing that stood out to me is that almost as many products have been dropped as they have been added. This could be a problem for the bank and definitely something that I would want to look into further if time permitted.

For this data set (Since it was originally compiled in Spanish), ‘V’ represents female customers and ‘H’ represents male customers. It was interesting to me that there is a fairly large female majority. At this point, I thought it would be relevant to explore the most popular products by men and women separately to see if I could identify any significant differences in their preferences.

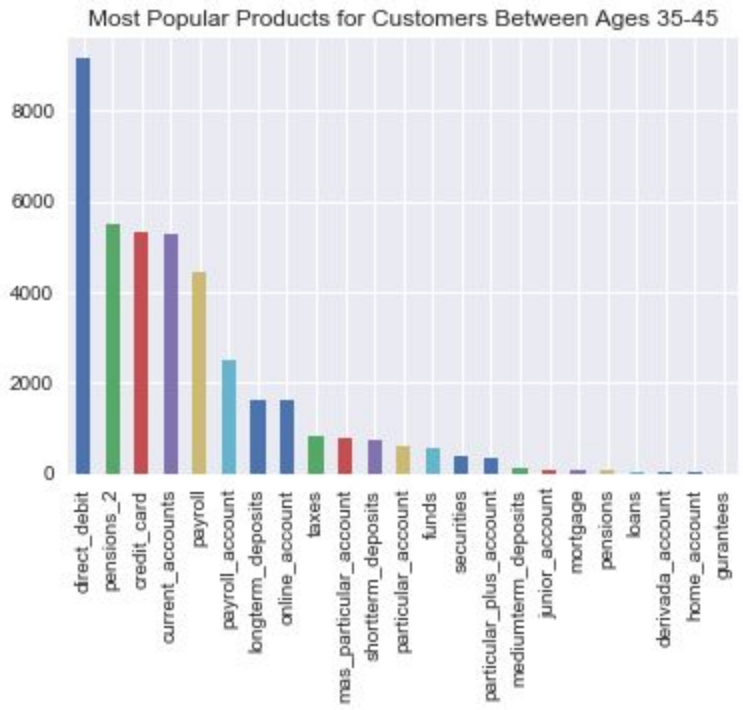




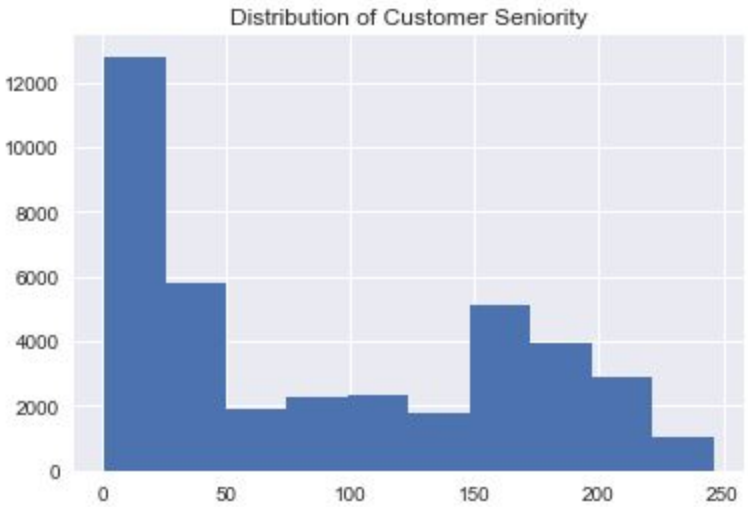
The most interesting insight here was the fact that female customers prefer credit cards significantly more than male customers.



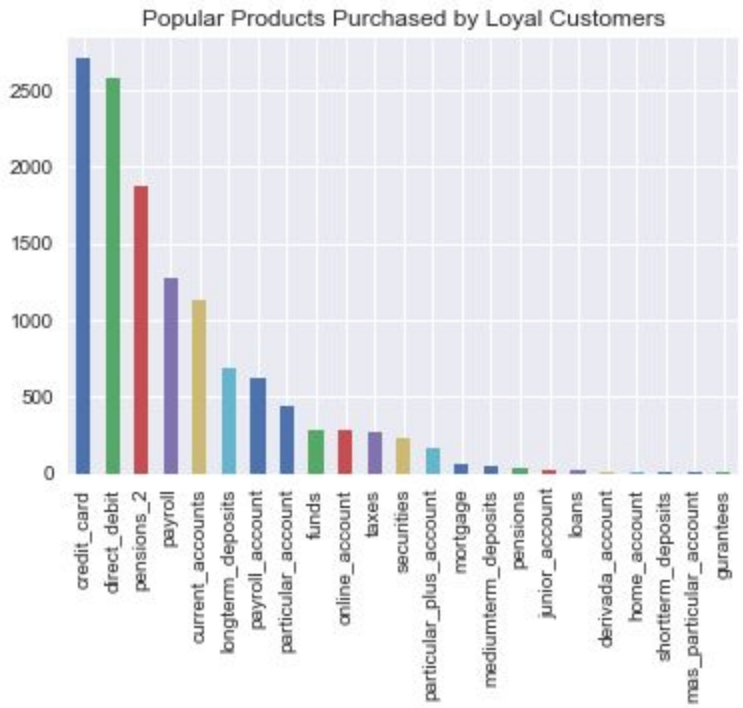
After looking at the distribution of the ages of customers, I noticed that the vast majority are around 40 years old. This is interesting because if we can take a look at what sorts of products those customers have, then it can help inform us on what we should be recommending to customers around the same age range that do not already have those products or for prospective customers.



Interestingly, the customers in the 35-45 age range seem to prefer Direct Debit accounts much more than an average customer. The rest of the top 5 is about the same as the average customer.



The customers in this data set are mostly new, however, we can see a substantial amount of them that have been with Santander for 150 months or more! Let’s take a look to see what these loyal customers have purchased.



Credit card and Direct Debit accounts are the clear winners here by a wide margin. What's striking to me is the fact that while all of the previous filters we looked at only had one clear most popular product, the gap between the most popular and second most popular is very slim.

Machine Learning

The final step in the process was performing some recommendation engines on the data and evaluating which one was the best.

To begin, I had to do a bit of preprocessing on the data. This included converting some of the categorical variables to integers. Specifically, I had to take all of the products that the bank offers and assign them their own product code or ID.

I then had to create 3 new dataframes, one that grouped the original dataframe by customer code, the product that they purchased and the sum of the products each user purchased. The next dataset I had to create was a dummy dataset marking whether a customer bought a specific item or not. The final one was a dataframe that normalized the item values across users. I had to begin by normalizing purchase frequency for each item across users by creating a user-item matrix. This gave us the scaled purchase frequency of each user on a scale of 0 to 1. 0 represented no purchases and 1 represented the most purchases for a customer.

Next, I had to split the data for each new dataframe. The ultimate goal was to be able to recommend products for a specific user based on what similar users previously purchased, which we will be doing with a collaborative filtering technique. However, it was important for me to create a simple popularity recommender to have a baseline to evaluate the collaborative filtering recommenders against.

The results for the popularity models were as expected - They each had the most popular products for each respective dataframe and repeated the results in identical order for every user.

The next step was to create collaborative filtering models in order for me to create recommendation engines that could recommend products for users who are similar to one another. How do we measure similarity? There are a few different ways, but for this project, I focused on cosine and pearson similarity. For these methods the key is to create an item-to-item similarity matrix. For calculating the similarity between two products, I looked at all of the customers who purchased both items. Then I created two-item vectors and found the cosine or pearson angle/distance between the vectors. Total similarity is equivalent to a cosine value of 1 and an angle of 90 degrees represents a cosine of 0 or no similarity.

Here are the results for the cosine and pearson similarity models:

Cosine similarity model using purchase counts:

customer_code	product_code	score	rank
118972	21	0.4760633276694447	1
118972	1	0.3921568627450981	2
118972	19	0.2981129755977626	3
118972	20	0.2921568627450981	4
118972	16	0.28104575163398693	5
118972	18	0.2626402038994857	6
118972	17	0.25262197902416783	7
118972	6	0.2062193627450981	8
118972	13	0.1741793346552104	9
118972	12	0.17360340362560123	10
118972	7	0.17244433707774698	11
118972	4	0.17032411362141253	12
118972	3	0.16138763197586736	13
118972	11	0.12253759312582846	14
118972	10	0.11010558069381604	15
116132	21	0.4760633276694447	1
116132	1	0.3921568627450981	2
116132	19	0.2981129755977626	3
116132	20	0.2921568627450981	4
116132	16	0.28104575163398693	5
116132	18	0.2626402038994857	6
116132	17	0.25262197902416783	7
116132	6	0.2062193627450981	8
116132	13	0.1741793346552104	9
116132	12	0.17360340362560123	10
116132	7	0.17244433707774698	11
116132	4	0.17032411362141253	12
116132	3	0.16138763197586736	13
116132	11	0.12253759312582846	14
116132	10	0.11010558069381604	15

Cosine similarity model using purchase dummies:

customer_code	product_code	score	rank
118972	4	0.0	1
118972	18	0.0	2
118972	21	0.0	3
118972	11	0.0	4
118972	13	0.0	5
118972	9	0.0	6
118972	12	0.0	7
118972	6	0.0	8
118972	10	0.0	9
118972	19	0.0	10
118972	7	0.0	11
118972	3	0.0	12
118972	17	0.0	13
118972	8	0.0	14
118972	14	0.0	15
116132	4	0.0	1
116132	18	0.0	2
116132	21	0.0	3
116132	11	0.0	4
116132	13	0.0	5
116132	9	0.0	6
116132	12	0.0	7
116132	6	0.0	8
116132	10	0.0	9
116132	19	0.0	10
116132	7	0.0	11
116132	3	0.0	12
116132	17	0.0	13
116132	8	0.0	14
116132	14	0.0	15

Cosine similarity model using scaled purchase frequencies:

customer_code	product_code	score	rank
118972	20	0.27182361733931265	1
118972	3	0.22896647448216978	2
118972	16	0.1384902840059793	3

118972	15	0.11095405212192133	4
118972	19	0.10349028400597929	5
118972	5	0.10170867481057697	6
118972	10	0.09693184244753772	7
118972	17	0.0800347610343273	8
118972	6	0.07401797470608379	9
118972	14	0.052379172894868126	10
118972	8	0.04829420557460676	11
118972	9	0.035018061783757076	12
118972	7	0.03398577950147544	13
118972	4	0.02859392827210827	14
118972	21	0.026933993688631863	15
116132	20	0.27182361733931265	1
116132	3	0.22896647448216978	2
116132	16	0.1384902840059793	3
116132	15	0.11095405212192133	4
116132	19	0.10349028400597929	5
116132	5	0.10170867481057697	6
116132	10	0.09693184244753772	7
116132	17	0.0800347610343273	8
116132	6	0.07401797470608379	9
116132	14	0.052379172894868126	10
116132	8	0.04829420557460676	11
116132	9	0.035018061783757076	12
116132	7	0.03398577950147544	13
116132	4	0.02859392827210827	14
116132	21	0.026933993688631863	15

Pearson similarity model using purchase counts:

customer_code	product_code	score	rank
118972	21	0.4760633276694447	1
118972	1	0.3921568627450981	2
118972	19	0.2981129755977626	3
118972	20	0.2921568627450981	4
118972	16	0.28104575163398693	5
118972	18	0.2626402038994857	6
118972	17	0.25262197902416783	7
118972	6	0.2062193627450981	8
118972	13	0.1741793346552104	9
118972	12	0.17360340362560123	10
118972	7	0.17244433707774698	11

118972	4	0.17032411362141253	12
118972	3	0.16138763197586736	13
118972	11	0.12253759312582846	14
118972	10	0.11010558069381604	15
116132	21	0.4760633276694447	1
116132	1	0.3921568627450981	2
116132	19	0.2981129755977626	3
116132	20	0.2921568627450981	4
116132	16	0.28104575163398693	5
116132	18	0.2626402038994857	6
116132	17	0.25262197902416783	7
116132	6	0.2062193627450981	8
116132	13	0.1741793346552104	9
116132	12	0.17360340362560123	10
116132	7	0.17244433707774698	11
116132	4	0.17032411362141253	12
116132	3	0.16138763197586736	13
116132	11	0.12253759312582846	14
116132	10	0.11010558069381604	15

Pearson similarity model using purchase dummies:

customer_code	product_code	score	rank
118972	4	0.0	1
118972	18	0.0	2
118972	21	0.0	3
118972	11	0.0	4
118972	13	0.0	5
118972	9	0.0	6
118972	12	0.0	7
118972	6	0.0	8
118972	10	0.0	9
118972	19	0.0	10
118972	7	0.0	11
118972	3	0.0	12
118972	17	0.0	13
118972	8	0.0	14
118972	14	0.0	15
116132	4	0.0	1
116132	18	0.0	2
116132	21	0.0	3
116132	11	0.0	4

116132	13	0.0	5
116132	9	0.0	6
116132	12	0.0	7
116132	6	0.0	8
116132	10	0.0	9
116132	19	0.0	10
116132	7	0.0	11
116132	3	0.0	12
116132	17	0.0	13
116132	8	0.0	14
116132	14	0.0	15

Pearson similarity model using scaled purchase frequencies:

customer_code	product_code	score	rank
118972	20	0.2718236173393127	1
118972	3	0.2289664744821698	2
118972	16	0.13849028400597932	3
118972	15	0.11095405212192144	4
118972	19	0.10349028400597932	5
118972	5	0.10170867481057702	6
118972	10	0.0969318424475378	7
118972	17	0.08003476103432733	8
118972	6	0.07401797470608382	9
118972	14	0.05237917289486815	10
118972	8	0.048294205574606786	11
118972	9	0.035018061783757104	12
118972	7	0.033985779501475466	13
118972	4	0.028593928272108382	14
118972	21	0.02693399368863192	15
116132	20	0.2718236173393127	1
116132	3	0.2289664744821698	2
116132	16	0.13849028400597932	3
116132	15	0.11095405212192144	4
116132	19	0.10349028400597932	5
116132	5	0.10170867481057702	6
116132	10	0.0969318424475378	7
116132	17	0.08003476103432733	8
116132	6	0.07401797470608382	9
116132	14	0.05237917289486815	10
116132	8	0.048294205574606786	11
116132	9	0.035018061783757104	12

116132	7	0.033985779501475466	13
116132	4	0.028593928272108382	14
116132	21	0.02693399368863192	15

Now that I was able to get the results for each model type, it was time to evaluate the models and decide which one should be the final choice. I decided to evaluate my models using three methods:

1. RMSE - The smaller the RMSE value, the better our model performs
2. Precision - What percentage of products that a user buys that are actually recommended
3. Recall - How many of the recommended items did the user actually like?

Results based on RMSE:

1. Popularity on purchase counts: 0.8286392012124091
2. Cosine similarity on purchase counts: 1.0191326912034409
3. Pearson similarity on purchase counts: 1.0191326912034409

1. Popularity model on purchase dummy: 0.0
2. Cosine similarity on purchase dummy: 1.0
3. Pearson similarity on purchase dummy: 1.0

1. Popularity model on scaled purchase counts: 0.1558247832874474
2. Cosine similarity on scaled purchase counts: 0.2034441843851805
3. Pearson similarity on scaled purchase counts: 0.20344418438518055

Results based on precision/recall:

Popularity:

Purchase counts

cutoff	mean_precision	mean_recall
1	0.15768056968463867	0.15768056968463867
2	0.07884028484231934	0.15768056968463867
3	0.05710410308579187	0.17131230925737503
4	0.0430315361139369	0.1721261444557476
5	0.03458799593082409	0.17293997965412025
6	0.06425907087148194	0.38555442522889094
7	0.059439035023979174	0.4160732451678536
8	0.05592573753814852	0.4474059003051882
9	0.052311518028710285	0.4708036622583921

| 10 | 0.05407934893184134 | 0.5407934893184136 |
+--------+---------------------+---------------------+

Purchase dummies

| cutoff | mean_precision | mean_recall |
+--------+---------------------+---------------------+
1	0.21810783316378465	0.21810783316378465
2	0.15951169888097683	0.31902339776195365
3	0.1774160732451681	0.5322482197355034
4	0.16983723296032516	0.6793489318413006
5	0.14929806714140353	0.7464903357070194
6	0.12858596134282838	0.7715157680569669
7	0.11434384537131241	0.800406917599186
8	0.10818921668362176	0.8655137334689741
9	0.09969481180061054	0.897253306205494
10	0.09011190233977616	0.9011190233977616
+--------+---------------------+---------------------+

Scaled purchase frequencies

| cutoff | mean_precision | mean_recall |
+--------+---------------------+---------------------+
1	0.0008138351983723276	0.0008138351983723276
2	0.0008138351983723274	0.0016276703967446548
3	0.0006781959986436082	0.002034587995930827
4	0.0009155645981688714	0.0036622583926754857
5	0.004231943031536118	0.021159715157680586
6	0.003763987792472031	0.022583926754832142
7	0.0038075861066705374	0.02665310274669379
8	0.007299084435401813	0.0583926754832145
9	0.01001469424663722	0.09013224821973521
10	0.009298067141403877	0.09298067141403837
+--------+---------------------+---------------------+

Cosine:

Purchase counts

| cutoff | mean_precision | mean_recall |
+--------+---------------------+---------------------+
1	0.15768056968463903	0.15768056968463903
2	0.07884028484231954	0.1576805696846391
3	0.05710410308579176	0.17131230925737556
4	0.043031536113936834	0.17212614445574734
5	0.034587995930824005	0.1729399796541202
6	0.06425907087148196	0.38555442522889105
7	0.059439035023979125	0.4160732451678534

8	0.05592573753814843	0.44740590030518745
9	0.0523115180287103	0.4708036622583927
10	0.05407934893184147	0.5407934893184138
+--------+---------------------+---------------------+

Purchase dummies

| cutoff | mean_precision | mean_recall |
+--------+---------------------+---------------------+
1	0.2181078331637853	0.2181078331637853
2	0.15951169888097746	0.31902339776195493
3	0.17741607324516848	0.5322482197355026
4	0.1698372329603261	0.6793489318413044
5	0.14929806714140342	0.7464903357070202
6	0.12858596134282835	0.7715157680569679
7	0.11434384537131237	0.8004069175991858
8	0.10818921668362148	0.8655137334689719
9	0.09969481180061064	0.8972533062054924
10	0.09011190233977631	0.9011190233977603
+--------+---------------------+---------------------+

Scaled purchase frequencies

| cutoff | mean_precision | mean_recall |
+--------+---------------------+---------------------+
1	0.000813835198372328	0.000813835198372328
2	0.0008138351983723276	0.0016276703967446553
3	0.0006781959986436081	0.0020345879959308274
4	0.0009155645981688728	0.0036622583926754913
5	0.004231943031536114	0.021159715157680555
6	0.003763987792472031	0.022583926754832173
7	0.003807586106670539	0.02665310274669382
8	0.007299084435401827	0.05839267548321462
9	0.01001469424663725	0.09013224821973519
10	0.009298067141403884	0.09298067141403844
+--------+---------------------+---------------------+

Pearson:

Purchase counts

| cutoff | mean_precision | mean_recall |
+--------+---------------------+---------------------+
1	0.15768056968463892	0.15768056968463892
2	0.07884028484231947	0.15768056968463895
3	0.05710410308579192	0.1713123092573756
4	0.04303153611393696	0.17212614445574784
5	0.034587995930824074	0.17293997965411972

6	0.06425907087148197	0.3855544252288904
7	0.05943903502397914	0.41607324516785316
8	0.0559257375381485	0.447405900305188
9	0.05231151802871027	0.4708036622583925
10	0.054079348931841194	0.5407934893184132
+--------+---------------------+---------------------+

Purchase dummies

| cutoff | mean_precision | mean_recall |
+--------+---------------------+---------------------+
1	0.2181078331637844	0.2181078331637844
2	0.1595116988809769	0.3190233977619538
3	0.1774160732451676	0.5322482197355042
4	0.16983723296032618	0.6793489318413047
5	0.14929806714140362	0.7464903357070178
6	0.12858596134282843	0.7715157680569668
7	0.11434384537131242	0.8004069175991851
8	0.10818921668362165	0.8655137334689732
9	0.09969481180061053	0.8972533062054924
10	0.09011190233977617	0.9011190233977616
+--------+---------------------+---------------------+

Scaled purchase frequencies

| cutoff | mean_precision | mean_recall |
+--------+----------------------+----------------------+
1	0.0008138351983723276	0.0008138351983723276
2	0.0008138351983723274	0.0016276703967446548
3	0.0006781959986436083	0.002034587995930824
4	0.0009155645981688709	0.0036622583926754835
5	0.004231943031536124	0.021159715157680618
6	0.0037639877924720345	0.02258392675483218
7	0.0038075861066705365	0.0266531027466938
8	0.007299084435401806	0.058392675483214446
9	0.010014694246637246	0.09013224821973527
10	0.00929806714140388	0.09298067141403854
+--------+----------------------+----------------------+

Based on the evaluation metrics, I decided to go with either the pearson similarity model or cosine similarity model for purchase dummy as the final choice. The reason I'm going with either/or, is because they basically had identical grades with the evaluation metrics and I conclude that you cannot go wrong with either one. The RMSE scores for both models are not as great as the other ones shown here, but the biggest difference is the precision/recall metrics. I would rather go with a model that is more precise than one that has a better RMSE score.

Conclusion

The ultimate goal here was to find a model that utilized the collaborative filtering technique and find some recommendations for similar users. I accomplished that with this project. However, if I had more time to work on it, I would research more on the Turi Create package and find ways to tune this model to make it even better. I would also strive to complete the objective of the original Kaggle competition that this data came from which was to predict what additional products customers will purchase in the month after the date range of the dataset.