Final Report- Product Recommendation Engine for Olist

Overview

The initial goal of this Capstone was to utilize a dataset of purchases from Olist, the largest department store in Brazilian marketplaces, and provide insights via customer segmentation. Olist provides a service that allows merchants to sell their products via the Olist store, and aids in shipping products to customers as well, providing two customer sets for segmentation in the data, one for buyers and another for sellers. However, after producing RFM scores for both customer sets, no patterns or insight could be derived from the remaining parameters in the data, as the dataset lacks demographic data past geographic location.

So, the direction of the Capstone shifted to producing product recommendations for the customer of the data set using a variety of similarity scores. Category groups were constructed linking Buyers to purchases made in certain Product Categories, Sellers to sales made in certain Product Categories, and Buyer to purchases of certain Products, and similarity scores were constructed comparing a member of a Category group to every other member of that group. Using these Category groups, a Product Recommendation engine was made that, given a specific buyer, can produce product recommendations based on this buyer's purchasing history. This engine could be deployed onto a server for Olist, and code has been provided for updating the Category Groups based on new sales.

Data Sets

Kaggle - Brazilian E-Commerce Public Dataset by Olist
https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

Data Preparation ('2 - EDA.ipynb')

RFM Score Generation:

In order to quantify, rank, and group the quality of customers (buyers and sellers in case of Olist) some metric must be constructed that rates customers against one another. A common metric, and the one explored in this capstone is the RFM score, which ranks customers based on the recency, frequency, and monetary total of their purchases in the data set. Construction of each is briefly explained below.

Recency

A measure of whether the customer has made purchases/sales in the recent past, ranking orders made further in the past as worse than those made closer to the present. With orders in the dataset sorted chronologically, the time of the last order is taken as the zero point. Then each customer is given a score based on the number of days between their last order in the set and the zero point. As such, a lower Recency score is considered better than a higher one.

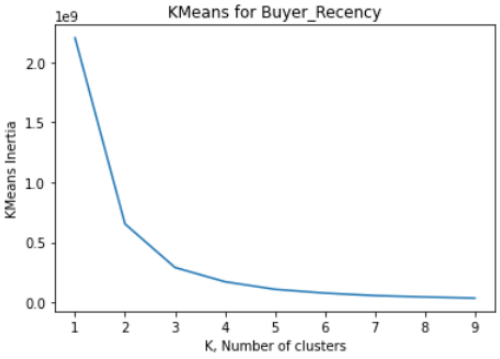
Frequency

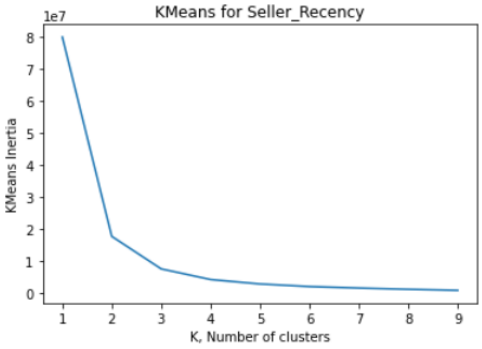
A simple measure of quality based on how many orders a customer has made in the dataset. A higher Frequency is considered better than a lower one.

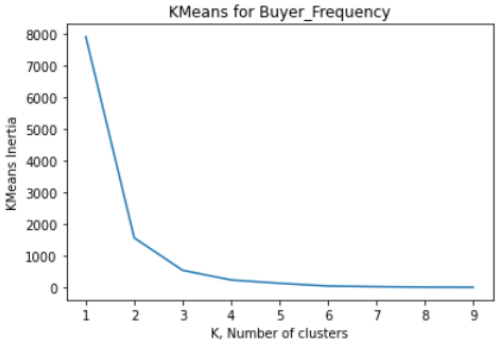
Monetary

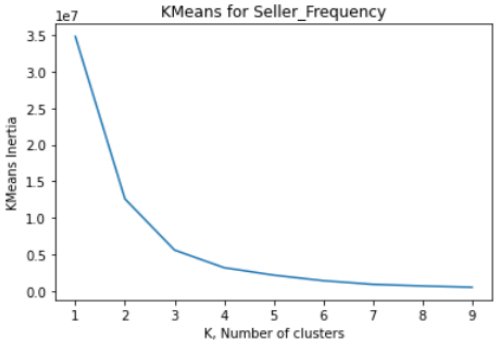
A measure of how much money a customer has spent/made based on the total price of each order. A higher Monetary value is considered better than a lower one.

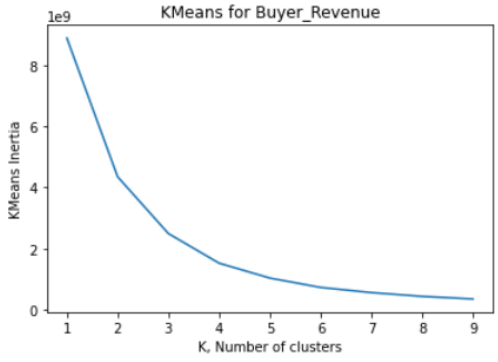
After calculating each score, each individual RFM component is clustered using KMeans with several values of K to determine the appropriate number of levels for the component, determined by the elbow point of cluster inertia. Clusters are then sorted so that a higher cluster number indicates a better score, and cluster numbers are assigned back to the customer in the data set. Using this procedure, the following inertia graphs were produced:











Based on these results the following cluster sizes were chosen for the customers' RFM Scores:

Customer	Recency	Frequency	Monetary
Buyer	4	4	5
Seller	4	4	5

RFM Total Summary:

Using these clusters, a total RFM Score for each customer can be constructed by adding the cluster numbers for each RFM component. These values can then be grouped further into RFM ratings based on the count of customers in each Score category.

Average Buyer RFM scores

Buyer_OverallRFMScore	Buyer_Recency	Buyer_Frequency	Buyer_Revenue
0	486.761675	1.000000	96.817523
1	336.774760	1.011934	123.618145
2	212.951759	1.029242	145.506507
3	98.333307	1.046431	170.908309
4	112.441015	1.278782	469.445107
5	111.177611	1.663540	875.757711
6	104.621469	2.180791	1434.112863
7	102.297468	2.778481	2405.420506
8	97.207547	4.150943	3225.849434
9	69.083333	4.166667	4316.088333

Average Seller RFM scores

Seller_OverallRFMScore	Seller_Recency	Seller_Frequency	Seller_Revenue
0	525.327273	3.523636	739.880145
1	318.578804	7.076087	1200.217310
2	161.557613	8.923868	1680.105700
3	33.487252	13.410057	2236.358215
4	34.827027	59.443243	13024.881946
5	23.024691	126.543210	19654.099012
6	22.560976	232.658537	46439.367073
7	14.851852	425.074074	53798.847037
8	22.625000	638.875000	128986.017500
9	7.375000	1187.125000	175086.316250
10	10.600000	1252.600000	254983.110000

Count of buyers by Overall RFM Score

Buyer_OverallRFMScore	
0	13490
1	21870
2	26264
3	25565
4	4925
5	1599
6	531
7	158
8	53
9	12

Count of sellers by Overall RFM Score

Seller_OverallRFMScore	
0	275
1	368
2	486
3	1412
4	185
5	162
6	41
7	27
8	8
9	8
10	5

Buyers RFM Ranking

Buyer_RFM_Ranking	
0) Lowest - Overall = 0	13490
1) Low - Overall > 0	48134
2) Medium - Overall > 2	32089
3) High - Overall > 5	689
4) Highest - Overall > 7	65

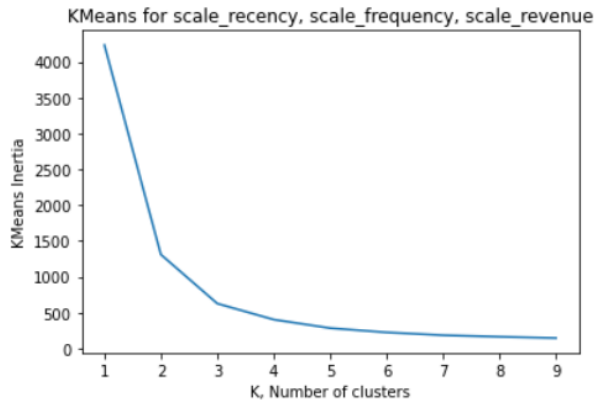
Sellers RFM Ranking

Seller_RFM_Ranking	
0) Lowest - Overall = 0	275
1) Low - Overall > 0	2266
2) Medium - Overall > 4	347
3) High - Overall > 6	89

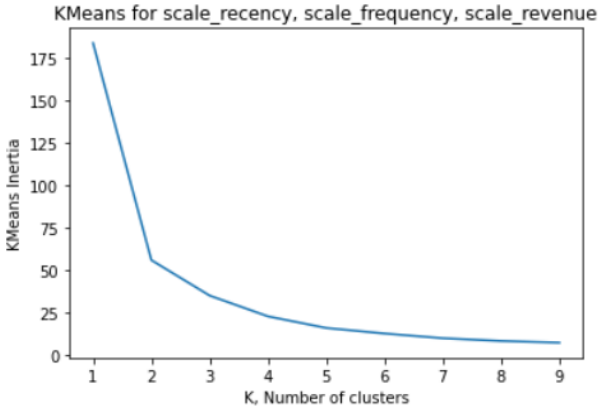
RFM Total Summary:

An alternative way to cluster the RFM scores is to first normalize each component, and use KMeans clustering using each component as a dimension, choosing K based on inertia as before.

Buyer Scaled RFM



Seller Scaled RFM



Buyer_RFM_cluster	
0	10219
1	13831
2	17360
3	17840
4	17624
5	17593

Seller_RFM_cluster	
0	278
1	393
2	530
3	1753
4	23

Tableau Lines Generation:

As another metric for visual exploratory data analysis, lines connecting Buyers to Sellers for each order were generated using the format specific to Tableau. To do this, two rows need to be created for each order, one tagged as ‘Origin’ with the Seller’s Longitude and Latitude data

and the other as 'Destination' with the Buyer's Longitude and Latitude data. Both rows also need a unique identifier linking them to each other.

Exploratory Data Analysis with Tableau

After generating the RFM Scores and lines from Buyer to Seller were created, an attempt was made to find patterns within the RFM clusters using Tableau's graphing software. The intent was to find general trends within the data, and then quantify them formally. This process could have worked with a dataset containing different features, but the Olist order data contains no demographic data past the geographic data of buyers and sellers. With additional features describing the age, gender, race, income, education, marital status, number of children or employment status for the Buyers or describing years of operation, number of employees, industry, net worth of CEO, or stock evaluation for Sellers, this method may have proved fruitful, but as it is the Olist dataset could produce interesting graphics, but not useful customer segmentation insights.

Below are the links to the Tableau workbooks created from this data for this Capstone, but the graphics themselves are not included in this report due to their lack of usefulness.

Tableau Workbooks:
RFM Score Investigation
Mapping of Olist Orders

Generation of Similarity Scores ('3 - Similarity Score Generation.ipynb')

With customer segmentation not feasible with the given features of the dataset, the direction of the Capstone shifted to making product recommendations for the buyers of the dataset. This is a common feature of sites comparable to Olist, such as Amazon, providing customers with lists of products they may be interested in purchasing next based upon their previous purchases.

In order to achieve this, some measure of similarity has to be generated, i.e. for a given customer, the recommendation engine needs to determine which other customers are similar to them and how similar they are so it can rank recommendations. This capstone considered three measures of similarity, described below.

Euclidean Distance

Distance between two rows of a matrix is calculated by subtracting one row from another, squaring the difference between each column, summing the result over the columns, and taking the square root of the result. The smaller the distance between one row and another, the more similar those rows are. An example calculation is shown below.

	1	2	3	4	5	6
A	100	0	0	40	60	35
B	0	50	33	10	0	30
C	0	50	33	50	40	30
D	0	0	33	0	0	0
E	0	0	0	0	0	5

=>

	A	B	C	D	E
A	0	134.59	118.8	132.34	126.89
B	134.59	0	56.57	59.16	65.68
C	118.8	56.57	0	86.6	91.18
D	132.34	59.16	86.6	0	33.38
E	126.89	65.68	91.18	33.38	0

Modified Euclidean Distance

As seen in the example below, the two rows that are the most similar (i.e. the closest) are D and E. However, these two rows have no non-zero features in common, and are considered close because both of them happen to be close to the origin. Euclidean distance will overestimate the similarity of mostly sparse rows, which will largely be the case for the recommendation engine. In order to counteract this, a modified Euclidean Distance was considered, defined below.

When considering the distance between two rows, find the non-zero columns of row 1 and subtract these from the corresponding columns from row 2 if those entries are zero valued. Then follow the same procedure for row 2, taking the non zero-columns and subtracting these from row 1 if the values in those columns are 0. With both rows modified, take the standard Euclidean Distance between them.

Example between two rows:

Row 1	100	0	0	40	60	35	**Euclidean Dist**
							118.8
Row 2	0	50	33	50	40	30	
Mod Row 1	100	-50	-33	40	60	35	**Modified Euclidean Dist**
							234.27
Mod Row 2	-100	50	33	50	40	30	

Example using matrix from before:

	1	2	3	4	5	6
A	100	0	0	40	60	35
B	0	50	33	10	0	30
C	0	50	33	50	40	30
D	0	0	33	0	0	0
E	0	0	0	0	0	5

=>

	A	B	C	D	E
A	0	263.97	234.27	264.68	248.39
B	263.97	0	89.44	118.32	124.02
C	234.27	89.44	0	173.21	177.15
D	264.68	118.32	173.21	0	66.75
E	248.39	124.02	177.15	66.75	0

As before, the closest rows are D and E, however, this is now closer to the distance between B and C than it was before, (D-E: 33, B-C: 56 for Euc; D-E: 66, B-C: 89 for Mod Euc). Another interesting aspect here is the distance from B to C and from B to D are nearly the same for Euclidean Distance, but the distance from B to C is noticeably smaller than the distance from B to D for the Modified Euclidean.

Cosine Similarity

One final similarity metric considered in this Capstone was cosine similarity, a measure of the cosine of the angle between two rows when considering them as vectors. Unlike the distance metrics considered before, this metric is normalized to values between -1 and 1, with 1 being the most similar, i.e. the values of one row are a positive multiple of the other, and -1 being the

least similar, i.e. the values of one row are a negative multiple of the other. In this Capstone, the values of the rows will all be positive or zero, so the cosine similarity will return results between 0 and 1. An example of the similarity scores are shown below, using the same matrix as before.

	1	2	3	4	5	6
A	100	0	0	40	60	35
B	0	50	33	10	0	30
C	0	50	33	50	40	30
D	0	0	33	0	0	0
E	0	0	0	0	0	5

=>

	A	B	C	D	E
A	1	0.17	0.46	0	0.27
B	0.17	1	0.79	0.49	0.44
C	0.46	0.79	1	0.36	0.32
D	0	0.49	0.36	1	0
E	0.27	0.44	0.32	0	1

This similarity score shows that the most similar rows are B and C, which is what one might expect when looking at the matrix.

Generation of Category Groups *('3 - Similarity Score Generation.ipynb')*

With the measures of how to compare one user to another defined, the user matrix then had to be constructed. Initially, a simple pivot table of customers on the rows, product categories on the columns, and number of orders in the cell values was constructed. However this led to a non trivial problem of computation time when trying to generate similarity scores. With 2918 unique sellers and 93161 unique buyers in the data set, comparing every customer to every other customer requires a large amount of computation time and computer memory.

To reduce the amount of resources required for comparing the customers to one another, the number of customers had to be reduced. The method this Capstone used to achieve this is described below.

1. First the customer group to be segmented, and the Categories to compare had to be chosen, (e.g. group - customer ids, category - product category groups)
2. Construct a pivot table using the groups as rows, and the category as columns, with the values of the cells containing the number of unique order ids matching that group and category
3. Normalize the rows to a specified percentage, i.e. divide each cell of a row by the sum of the row, multiply by 100 to get a percentage, then round down to the nearest 5%
4. Remove duplicate rows. This effectively groups together all rows that would have a cosine similarity of 1 together, drastically reducing the number of rows in the set.
5. Add Column for Size of each group
6. Since not every member of the selected group (e.g. customer id) is present in the set any longer, rename the group to a new identifier (e.g. CustomerCategoryGroup) and reset the groups values to the index in the array (e.g. 0 to len(table)-1)
7. Remove duplicate columns, this effectively removes any category that is always bought with another

This procedure was used for the following group-category combinations:

Group	Category	Original Size	New Group Id	New Size
Customer Id	Product Category	(93161, 31)	*CustomerGroupId*	(536, 31)
Seller Id	Product Category	(2918, 31)	*SellerGroupId*	(1031, 31)
Customer Id	Product Id	(2868, 4520)	*CustomerProductGroupId*	(2810, 3349)
(only customers with more than one product_ids used)				

Similarity Score Comparison *('3 - Similarity Score Generation.ipynb')*

With the three Category Groups constructed, the next step was to compare the Similarity metrics described above to determine which is best suited for clustering this dataSet.

For this comparison, Agglomerative Clustering was used to cluster the groups of each Category Group set, as it allows for specifying custom distances between groups (needed for testing Modified Euclidean metric) and for specifying using cosine similarity. To determine which metric performs best, the average Silhouette Score scores were computed for various values of cluster count. For reference, Silhouette Scores are computed as follows for each sample:

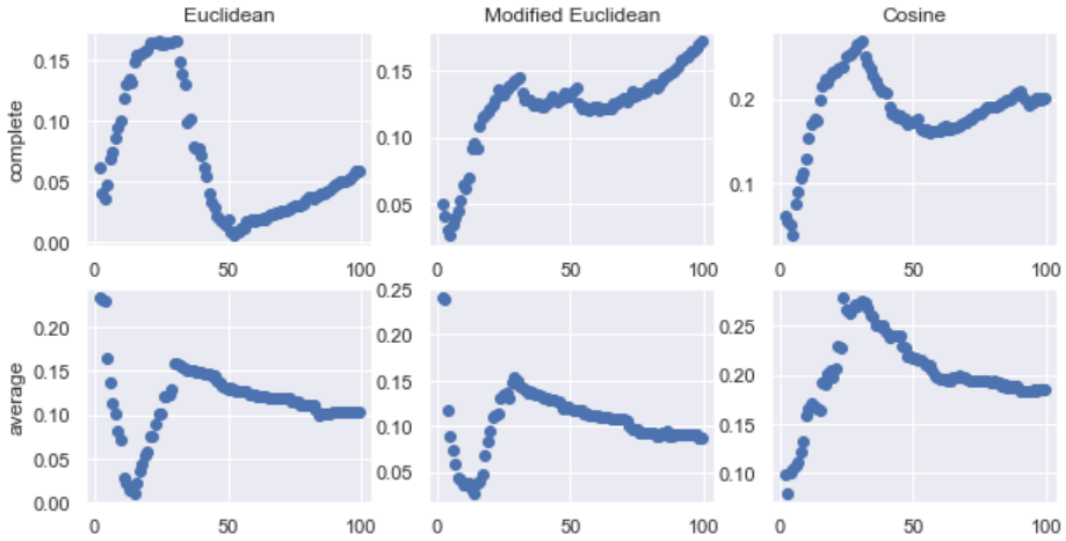
$$a = mean\ distance\ between\ the\ samples\ of\ the\ cluster\ this\ sample\ is\ a\ part\ of$$
$$b = distance\ between\ the\ sample\ and\ the\ nearest\ cluster\ that\ the\ sample\ is\ not\ a\ part\ of$$

$$Silhouette\ Score = \frac{(b-a)}{max(a,b)}$$

Results of clustering:

Below are the silhouette scores for clustering the three Category groups, using values of k form 0-100, different linkage specifications [Average, Complete], and the three similarity metrics described before. Following the graph for each Category group is a summary of the max Silhouette Score for each graph.

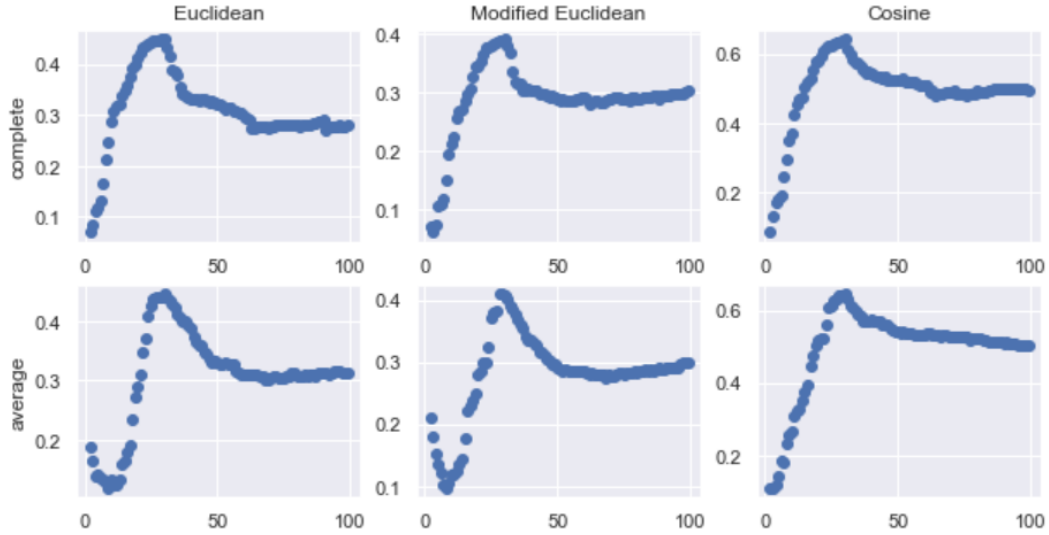
Customer Id - Product Category Groups



Summary

Distance Measurement	Linkage	Max_Silhouette_Score	Number_Of_Clusters
Euclidean	complete	0.166001	24
Modified Euclidean	complete	0.172348	99
Cosine	complete	0.269330	31
Euclidean	average	0.233287	2
Modified Euclidean	average	0.239850	2
Cosine	average	0.278595	24

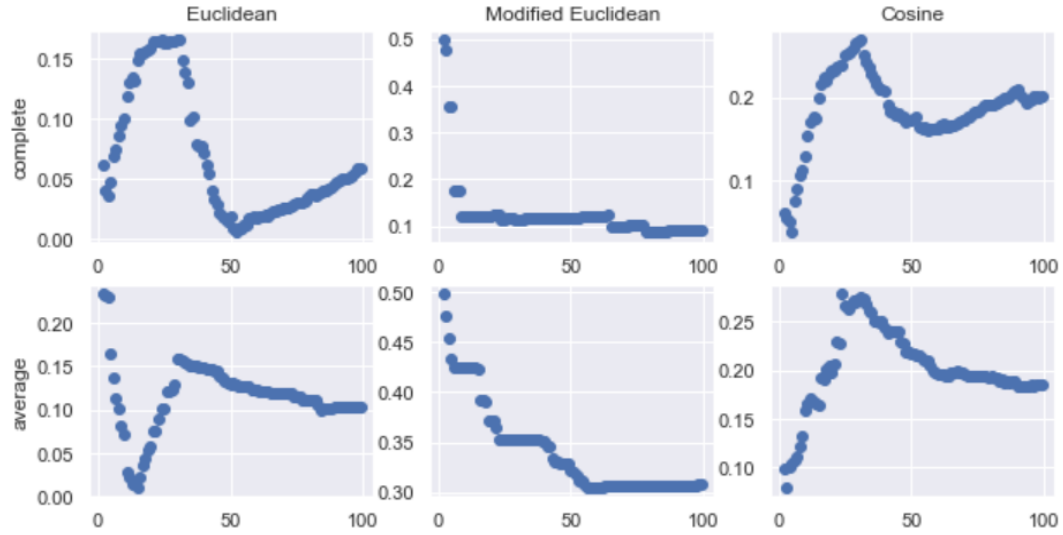
Seller Id - Product Category Groups



Summary

Distance Measurement	Linkage	Max_Silhouette_Score	Number_Of_Clusters
Euclidean	complete	0.448936	30
Modified Euclidean	complete	0.392026	30
Cosine	complete	0.642732	30
Euclidean	average	0.445347	30
Modified Euclidean	average	0.408833	29
Cosine	average	0.643737	30

Customer Id - Product Id Groups



Summary

Distance Measurement	Linkage	Max_Silhouette_Score	Number_Of_Clusters
Euclidean	complete	0.166001	24
Modified Euclidean	complete	0.497897	2
Cosine	complete	0.269330	31
Euclidean	average	0.233287	2
Modified Euclidean	average	0.497897	2
Cosine	average	0.278595	24

For all three Category Groups, cosine similarity performs the best when considering a reasonable number of clusters. For instance, the Modified Euclidean metric performs the best on the Customer Id - Product Id group, but achieves this score using only two clusters, which is unrealistic. Also, the number of clusters that produce the best average Silhouette Score for a metric of cosine similarity and a linkage of 'average' is at or around 30. This cluster size will be used for generating recommendations.

Construction of Recommendation Engine *('4 - Product Recommendation Engine.ipynb')*

With category groups constructed and similarity score metric decided on, most of the heavy lifting of the product recommendation engine is complete. All that is left is to connect these components together. The output of the Similarity Score Jupyter notebook is a csv defining each group in a Category Group, (complete with their percentage distribution among the categories, the Group Identifier, and the number of customers in each group), and a similarity csv with Group Identifiers as the columns, and for each row as well, with each cell defining the similarity between the Groups defined by the cell's row and column.

To make product recommendations for a specific customer, the Engine found in the Jupyter notebook can do the following:

For customers already Identified, (i.e. Group Id assigned to the customer previously)

1. Use customer's group Id to pull the column of similarity scores of this group from the cosine similarity csv
2. Rank the Group Ids (including the users) by their similarity to the user's Group
3. For each Group Id, identify orders in the data set belonging to this group, and pull the product ids of this order, excluding all the user's previously purchased products
4. Add product id to recommendation dictionary as a key, with the similarity score as the value, skipping over product ids added previously (highest similarity kept for duplicates)

For customers without an assigned Group Id

1. Make a dataFrame of the count of all the user's orders for the columns of the Category Group, normalizing the count to percentage of users total orders
2. Use cosine similarity to compare the user's DataFrame to the rows of the Category Group's dataframe
3. Rank the Group Ids (including the users) by their similarity to the user's DataFrame
4. For each Group Id, identify orders in the data set belonging to this group, and pull the product ids of this order, excluding all the user's previously purchased products
5. Add product id to recommendation dictionary as a key, with the similarity score as the value, skipping over product ids added previously (highest similarity kept for duplicates)

Using Agglomerative clustering:

1. Make a dataFrame of the count of all the user's orders for the columns of the Category Group, normalizing the count to percentage of users total orders
2. Add the User's Row to the end of the Category Group DataFrame, assigning it a new Group Id
3. Use the combined DataFrame to fit an Agglomerative Cluster model, using 'average' linking, a cluster count of 30 (determined above), and cosine similarity as the metric
4. Get the cluster number assignment for the user's and find all the other Group Ids assigned to this cluster
5. Rank the other Group Ids by their silhouette score
6. For each Group Id, identify orders in the data set belonging to this group, and pull the product ids of this order, excluding all the user's previously purchased products
7. Add product id to recommendation dictionary as a key, with the silhouette score as the value, skipping over product ids added previously (highest silhouette score kept for duplicates)

Using any or a combination of these methods, the Recommendation Engine can produce product recommendations based on any of the three Category Groups defined above, recommending products for a given user based on:

- Products purchased by other Buyers who buy from specific product categories in a similar proportion to the user

- Products sold by Sellers who sell items from specific product categories in a similar proportion to the product categories the user buys
- Products purchased by other Buyers who have bought specific products in similar proportion to the user

Recommendations considering RFM Scores

As an alternative to the similarity scores used for ranking the recommendations described above, the code also allows for considering RFM Score. This is a simple alteration to the values of the recommendation dictionary, simply adding in the normalized RFM Score for either Buyers or Sellers (depending on the Category Group used) to the original similarity scores.

Recommendation Example

Customer ID - 3e43e6105506432c953e165fb2acf44c

Customer's Product Group Buying Habits

Furniture_Bedroom	7
Furniture_Home	3
Construction	1
Technology	1
HomeAppliance	1

Recommendation Product Groups

Top 200

Furniture_Bedroom	159
Furniture_Home	26
Construction	7
Gardening	5
HomeAppliance	1
Home	1

Top 200 (Considering RFM Score)

Furniture_Bedroom	185
Furniture_Home	9
HomeAppliance	1
Beauty	1
Office	1
Home	1
Construction	1

Recommendation Table:

product_id	SellerGroupId_CosSim	SellerGroupId_Aglom	CustomerGroupId_CosSim	Total	product_category_name_english
0b814a3c8fa6dbb849df7c28c1bd6831	0.0	0.802525	0.916698	1.719223	Furniture_Bedroom
1b0fb6d6a05121d4b76a49f44c2c427b	0.0	0.802525	0.916698	1.719223	Furniture_Bedroom
odee5506e0699fa985e30a00b701b17a	0.0	0.802525	0.916698	1.719223	Furniture_Bedroom
aff39c649de8c4e36d325880de8f4338	0.0	0.802525	0.916698	1.719223	Furniture_Bedroom
428f4b96ba9f630e761b333673039ae6	0.0	0.802525	0.916698	1.719223	Furniture_Bedroom
...	...	...	...	...	...
2eefd8ed7a9782380fe14a1efdec7418	0.0	0.000000	0.943983	0.943983	Furniture_Home
ebad0ed16ecd450b97d2be843d3da86	0.0	0.000000	0.943983	0.943983	NaN
3512f777c335f8297b5ad43416b69cf8	0.0	0.000000	0.935601	0.935601	Furniture_Bedroom
7f9f228320c43765cfe30cdc090e0be4	0.0	0.000000	0.935601	0.935601	Furniture_Bedroom
a02d0123079f4ae96001ba2010d1a2df	0.0	0.000000	0.935601	0.935601	Construction

Updating the Engine

Also included in the notebook is a function to update the input to the recommendation engine, i.e. the Category Groups, and Similarity Matrix using the following procedure for every unique user in a list of new orders.

1. If a user is already in the data set, i.e. they are assigned a group number, reduce the count of that group in the Category Group DF by 1
2. Gather all of the users orders (old and new) and produce a data frame for the user using the columns of the Category DF, with the value of each column being the number this user's orders in the category
3. Normalize the user's DataFrame percentages, rounding down to the same precision as the Category DataFrame (nearest 5% in this Capstone)
4. If the User DF matches an existing one in the Category Group DF (using cosine similarity), increase that row's count by 1, otherwise create a new row with a count of 1 and with a new Group Id number
5. If a new group was created, add a corresponding row and column for the new Group Id number in the Similarity Matrix, using cosine similarity scores of the User's DF and the older Groups
6. Repeat steps 1-5 until all users in new orders have been considered
7. If a group in the Category Group DF is now empty, remove that group's row from the Category Group DF, and the corresponding row and column from the Similarity Matrix
8. Order the Category Group DF and Similarity Matrix by Group ID, and reset the IDs to a continuous range of numbers. Update the Group ID assignments for the customers in the orders list

Evaluating the Recommendation Engine

One thing this Capstone lacks is an evaluation of the recommendations made. This is because recommendations are evaluated by user habit changes after encountering recommendations, and this Capstone has no real interaction with the users in the dataset. Ideally, Olist could use the Recommendation Engine and evaluate its performance with A/B Tests, finding the right mixture of similarity scores that produce the desired effect.

In that regard, determining the desired effect is an interesting problem in and of itself. Below is a summary of some effects and the possible downsides of choosing them.

Effect	Could Over- recommend	Possible Issue in Implementation
Increase Sales	Cheap Products	Product selection seen as low quality
Increase Profit	Expensive Products	Product selection seen as too expensive
Increase Customers	Popular Products	Seller's Newest Products not highlighted
Increase Time on Site	Product Variety	Site fun to browse, but not buy

Putting the Product Recommendation into practice, and modifying it based on user interaction is outside the scope of this Capstone.

Improvements/ Future Work

Alternative Clustering

Agglomerative Clustering was chosen for evaluating the various similarity scores, as it allowed for specifying custom distances. This was needed to test the effectiveness of the Modified Euclidean metric, but this metric was out performed by cosine similarity. Other clustering algorithms should allow using cosine similarity as well, and will overcome the major downside of Agglomerative Clustering, which is that all samples must be present for modeling, and new samples cannot be 'predicted' after model creation.

In the present implementation of the Recommendation Engine, the Agglomerative Clustering model is created and fitted at the time the recommendation is made. With other clustering models, this model could be created and stored beforehand and the user's cluster predicted at time of recommendation, drastically increasing the speed of recommendation generation.

Limiting Recommendations

In the present implementation, all products with similarity above a given threshold are passed back. This leads to some products, those from Seller similarity for instance, dominating the results. Likely, recommending every product from a given with the same similarity score is unreasonable, and this procedure should be revisited. Products could be cross referenced with other scores to provide weighted recommendations from a given seller, and they could be weighted in popularity as well. This would provide an efficient way of weighting the returned recommendations, which could then be limited to the top 30 from each Seller instead of returning them all.

Increasing Variety in Recommendations

The product recommendations are currently heavily weighted toward the customer's highest product group percentage. For instance, in the example above, where over half of the order's were for 'Furniture Home', the top 150 recommendations were all 'Furniture Home', and only become more diverse when getting the top 175 recommendations. This is likely related to not limiting the recommendations discussed previously, but the Recommendation Engine could also be modified to favor more variety in recommendations. This could be achieved by only returning the top recommendations in a given product group, with size weighted by the user's product category percentages. For example, for the customer discussed above, limit the recommendations from 'Furniture Home' so that they are at most 50% of all the returned Recommendations.

Conclusion

This Capstone started with an attempt to provide insight into the customer groups (Buyers and Sellers) of Olist, based on a large dataset of orders. This proved to not be feasible given the features in the dataset, which is a valuable lesson that not every data science problem can be solved for every dataset. Another lesson learned here is the need to be adaptable, as although one problem can't be solved for given data, it may be suited to other purposes. This led to an interesting investigation into the nature of similarity between samples, and providing value to Olist not by clustering its customers and finding demographic statistics, but by generating new product recommendations for their customer's next purchase. The process followed in completing this Capstone has been arduous and complex, but rewarding in overcoming the challenges it presented.