

Capstone Project # 2

Recommendation System Using Credit Card Transactional Data

Milestone 1 Report - [Recommendation_System_EDA_Milestone1](#)

PowerPoint Slides - [Recommendation_System_Using_CC_Transactional_Data](#)

Code - [EDA.ipynb](#), [Recommendation_System_Goal1](#), [Recommendation_System_Goal2](#)

Problem Statement

Create a recommendation system using credit card transactions records. The data contains transactions from multiple people and contains GPS records. The aim is to provide the customer following

- a. The recommendation system should recommend similar merchants to the customer based on similarities between the merchants
- b. Based on the location of a merchant, other merchants in the vicinity are recommended to a user

Exploratory Data Analysis

1. Data Mining/Data Wrangling

The data for this project came from [FinGoal](#). FinGoal analyzes consumer spending to give radically insightful recommendations built on the experience of other savvy consumers (source: FinGoal.com). The data was anonymized and converted to json format by the company. The file size was 48.2MB. Panda's `pd.read_json` method was used to read the file into a data frame. The file had 29 columns and 90086 rows of transactions.

Dropping unnecessary rows and columns

There were a lot of Null values and hidden duplicates in the data. For starters following columns were deleted as they have less significance compared to other features specifically for creating a recommendation system. The columns deleted/dropped were

'detail_category', 'high_level_category', 'transactionid', 'client_id', 'currency_code', 'country', 'pending', 'original_description', 'category0', 'finsight_api_uid', 'finsight_image', 'insight_ctaurl', 'insight_text', 'original_uid', 'tenant_id', 'accountid'

The merchant information was in the text of the column "simple_description". There were multiple rows that had exactly the same data

(including the date, uid etc) except for transactionid. By deleting transactionid column those duplicate rows were dropped. Other columns were dropped because those were very thinly populated. Column "accountid" is a duplicate of "accountid" hence it was deleted.

Dealing with NaN(Not a Number)

From df.info() output it was observed that there are several columns that could be used to fill NAN values of their "like" or "buddy" columns. For example: zip_code and city columns can be used to fill each other's NaN. The process of this function is (1) Find a row with NaN in zip_code (NaN) and record this row's city (city1). Find another row where the city is set to city1 with corresponding zip_code(zip1). (2) Replace NaN of the row with zip_code(zip1) as they both have the same city. There is an assumption here that each city has only one zip_code.

The column 'accountid' and 'uid' could be considered as "like/buddy" columns but there were several rows that had 'uid' but no 'accountid'. Therefore the logic presented above couldn't completely fill the NaNs of 'accountid'. The logic to fill NaN of accountid was to generate random numbers between 10449313 and 10478014 and apply to the rows with unique 'uid' that had no 'accountid'. After this process column 'accountid' had no NaN in any of the rows.

Category column

In order to go further in the EDA the category column needed to be understood. There were 66 unique values in the category column. The values dropped from the category column were:

'Other Bills', 'Other Expenses', 'Paychecks/Salary', 'Tax'
'Refunds/Adjustments', 'Transfer', 'Transfers', 'Interest', 'Credit Card Payments',
'Uncategorized', 'Business Miscellaneous', 'Paychecks/Salary', 'Insurance',
'Savings', 'Other Income', 'Securities Trades', 'Deposits', 'Services', 'Tax', 'Taxes',
'Rent', 'Mortgages', 'Loans', 'Investment Income', 'ATM/Cash
Withdrawals', 'Investment Income', 'Loans', 'Retirement Contributions', 'Expense
Reimbursement', 'Retirement Income', 'Charitable Giving', 'Bank Fees', 'Wages
Paid', 'Checks', 'Rewards', 'Printing', 'Payment'

Majority of the categories in the above list were related to incoming money(salaries, refunds, retirement income etc) or categories that cannot be used to predict better merchants for a user(loans, withdrawals, transfer, interest, charitable giving etc).

Amountnum Column

The columns with less than or equal to zero amount were deleted. 75% of the transactions were under \$52 and the number of rows with amounts of \$6000 or more looked like outliers. Therefore those rows were dropped. There were several transactions that had \$6000 charges however upon investigations it was found that those were duplicates with different dates and accountid.

2. Data Visualization

Number of Transactions per State

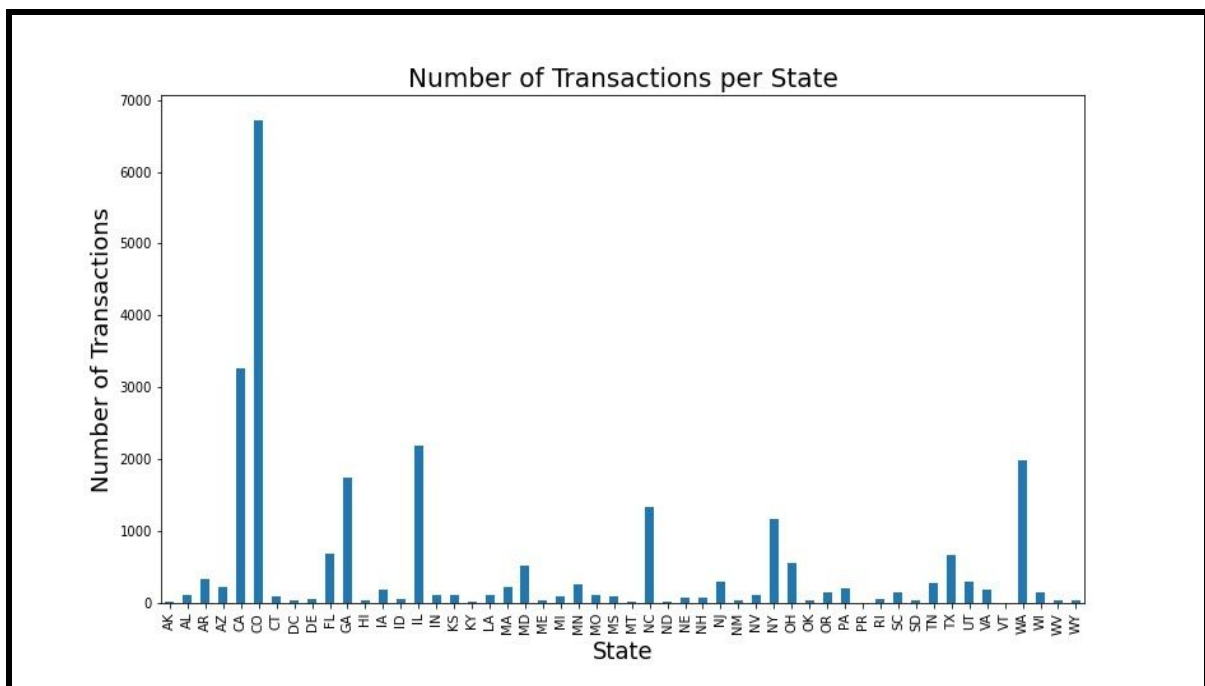


Figure 1: Number of Transactions per State

Figure 1 shows that most of the transactions are from Colorado, followed by California and Illinois. Colorado has around 7K transactions in the data which is almost 50% more than California. Since the FinGoal is in Colorado, the highest transactions in that state makes sense.

Number of Transactions per Category

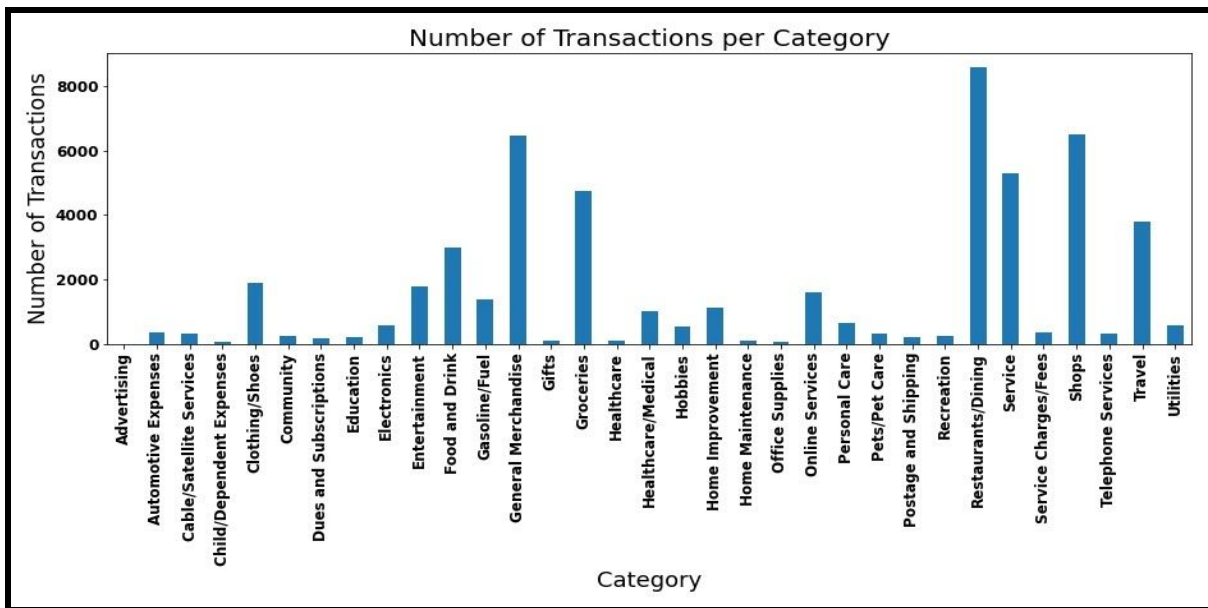


Figure 2: Number of Transactions per Category

From Figure 2 it can be seen that the majority of records are coming from the "Restaurants/Dining" category. Followed by "General Merchandise", "Shops", "Groceries", "Service", and "Travel".

Total Amount(\$) Spent Per Category

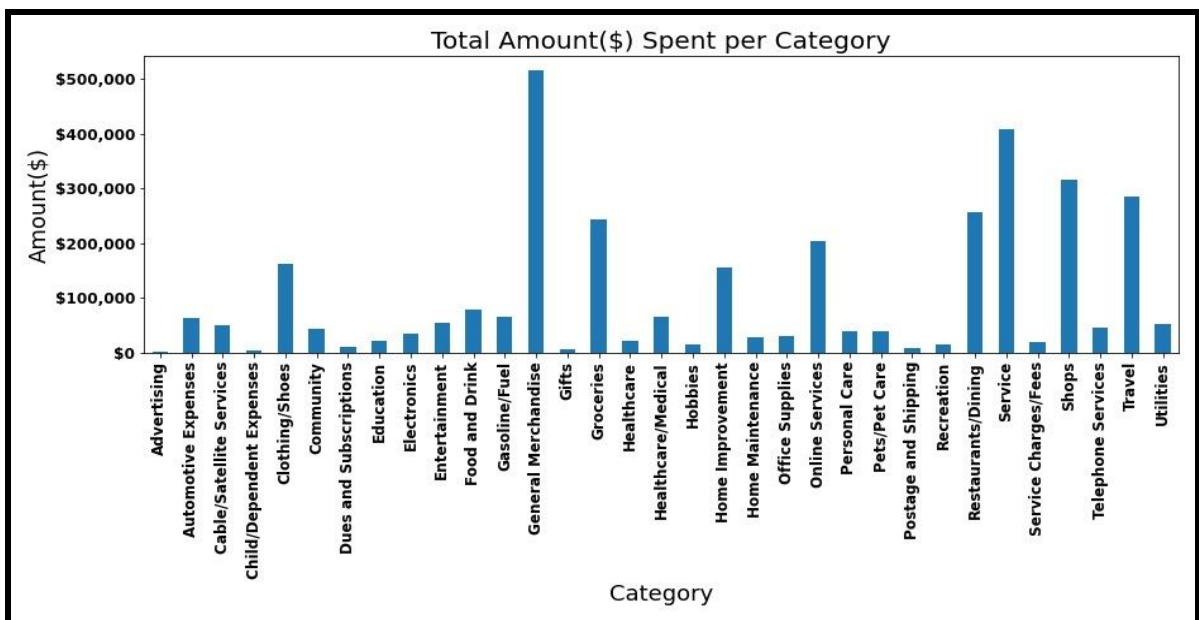


Figure 3: Total Amount(\$) Spent per Category

The category where most dollars were spent is "General Merchandise". "Service", "Travel", "Restaurants/Dining", "Groceries" come next. Since "General Merchandise" brought in the total of around \$500K, looking further into the category and finding the highest earning merchants.

Zooming into General Merchandise Category

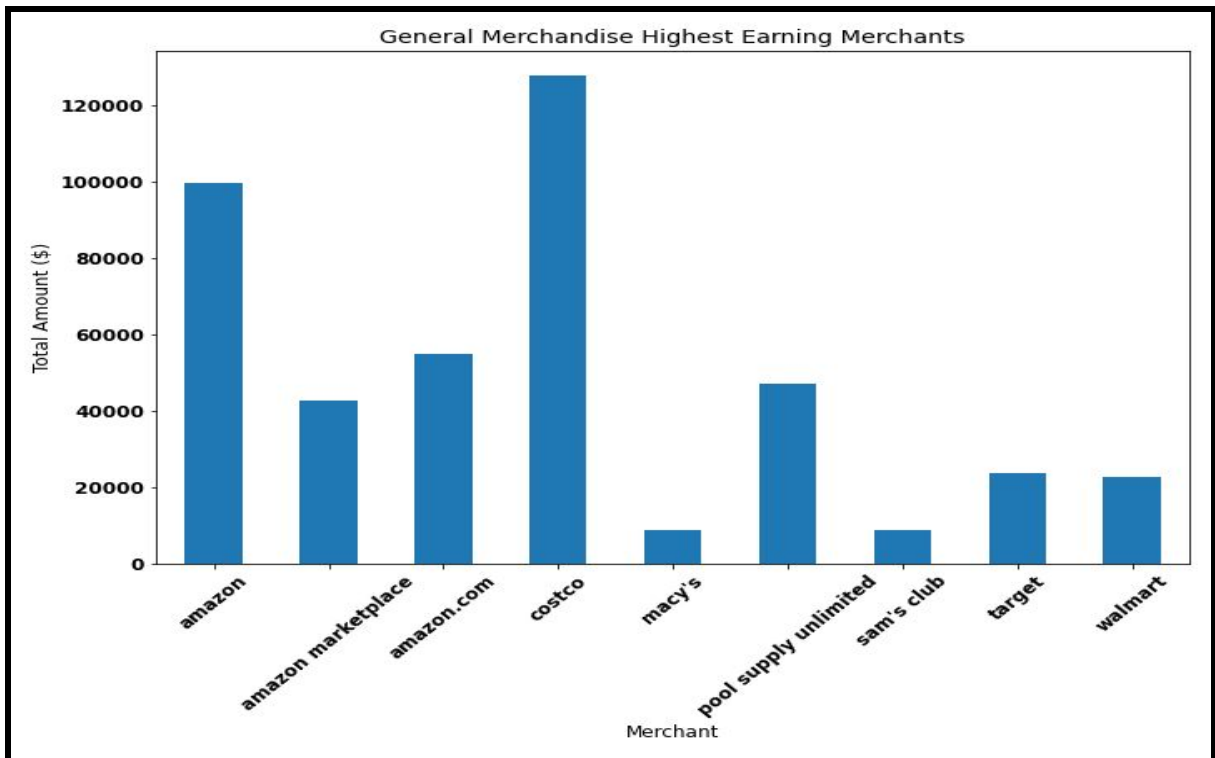


Figure 4: General Merchandise - Highest Earning Merchants

Users spent \$128K at Costco which is the highest amount in the General Merchandise category. That's more than a quarter of the amount spent elsewhere. All the high earning merchants are well known names except for "pool supply unlimited". Upon investigation, it was revealed that the transactions associated with this merchant were added to the data while FinGoal was testing their software. Multiple developers added the same transaction with different uids. Only one transaction per uid was kept in the data for this specific transaction. Others were dropped.

3. DateTime Analysis

Mean Amount Spent Per Month

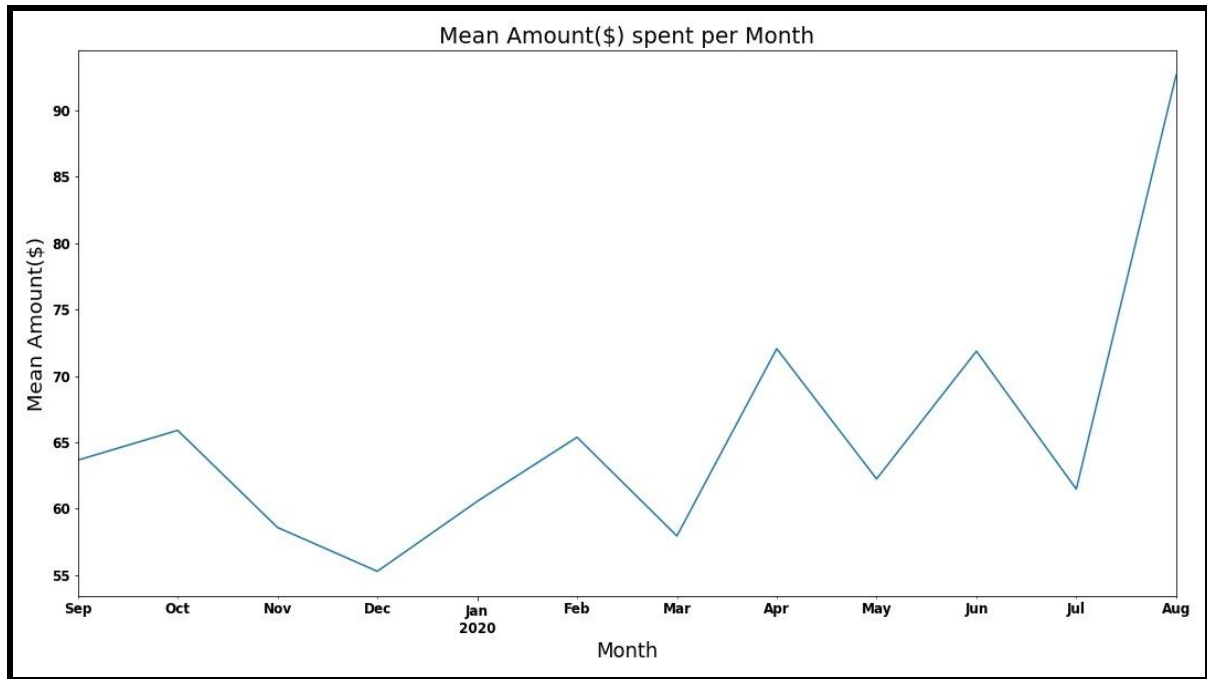


Figure 5: Mean Amount (\$) Spent Per Month

The data was collected from Sep 2019 to August of 2020. Clearly there are spikes in the data where the average amount went up in a few months and sharply went down in other months. In April the mean amount went up to 73K which could be a result of all the schools and workplaces being shutdown due to CoronaVirus. People were forced to spend money on office and school supplies that they normally didn't. The weirdest spike is in August 2020 where the mean amount went above \$90 which is 38% jump from July mean amount.

Average Amount Per Month Per Category

In Figure 6 below, the average amount per month per category is shown for the year 2020. The data in the monthly graph below explain Figure 5 very well. For example before CoronaVirus hit, in February, people spent money on everything and when the shutdown started to occur in March majority of spending was limited to "office supplies". The same trend continued in April and May. In June the spending increased only to be followed by a huge decrease in July. In August, even though the mean maximum amount was around \$200 in category General Merchandise, the amount of transactions

increased and the spending was up in all categories therefore the mean spiked compared to other months. Figure 7 shows August's graph in detail

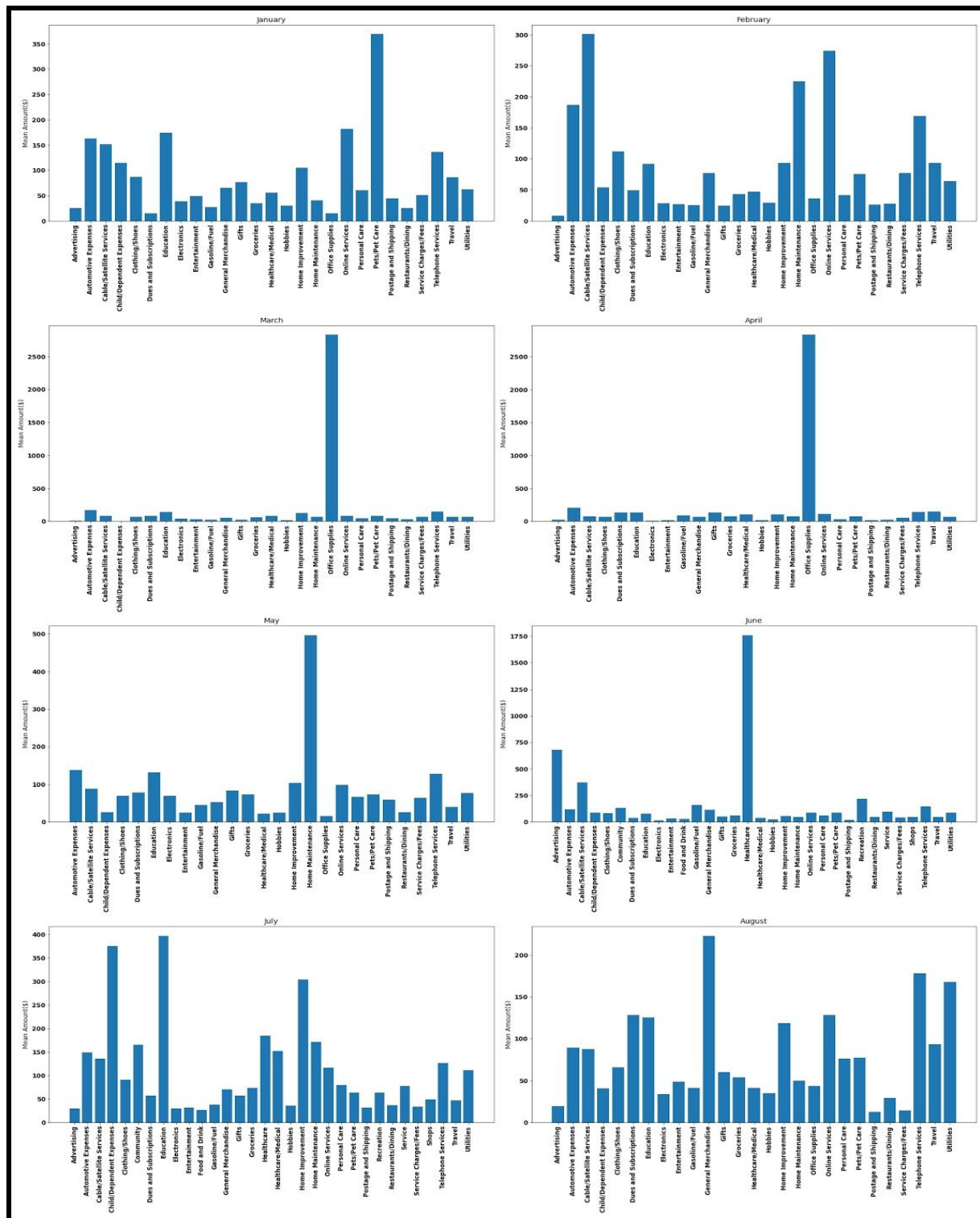


Figure 6: Mean Amount Per Category Per Month

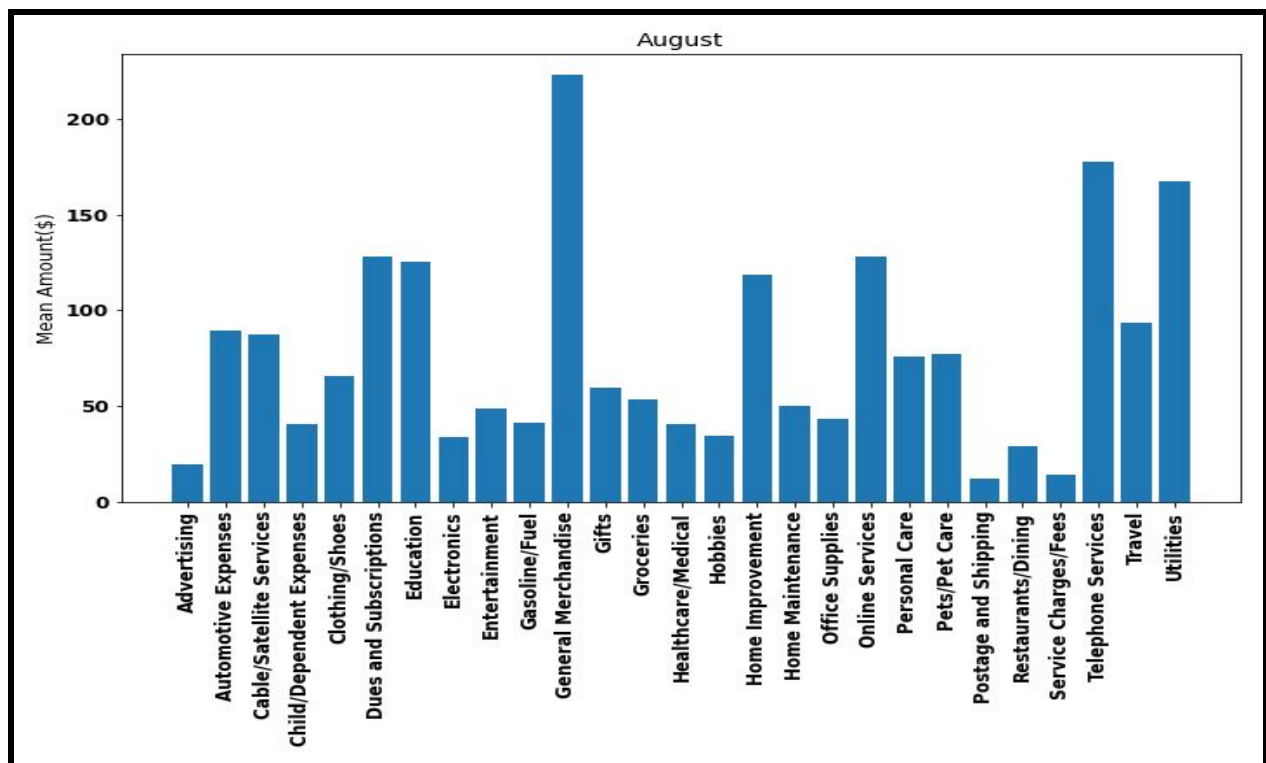


Figure 7: August 2020 - Mean Amount Per Category

Recommendation Systems

Goal 1: Collaborative Filtering Based Recommendation System

Statement: The recommendation system should recommend similar merchants to the customer based on similarities between the merchants

Strategy:

To clean and wrangle the data was quite a bit of a challenge. The most time consuming task was to make sure a single merchant is represented with a single identifier. For example, ziggy's coffee and bar is a shop in Lafayette, CO. There were plenty of records for this merchant but with five different versions of the name, ziggy, ziggy's, ziggy's coffee and ziggi's coffee and bar. Combing through the data and finding merchants like that was cumbersome however it had to be done in order to make sure the machine learning models see those records from one merchant and train properly. Not all the records were cleaned as it was not the goal of this project therefore records like ziggy's coffee and bar still exist in the data given by FinGoal.

To accomplish the first goal, categories with the most records were chosen. Within each of those categories top 15 merchants were used for this project. The categories for which the code was refined and tested were Restaurants/Dining, Shops, General Merchandise, Service, Groceries, Travel, Food and Drink, Clothing/Shoes, Entertainment, Gasoline/Fuel, Home Improvement and Healthcare/Medical. Each of these categories have greater than 900 records.

To find similarities between the merchants Pearson correlation coefficient 'r' and Cosine scores were selected for this goal. Apart from calculating scores with Pearson correlation and Cosine, a logic is introduced that combines the scores in order to get the best recommendations to the users. Following gives details on the steps taken to accomplish the tasks

Models:

Pearson Correlation:

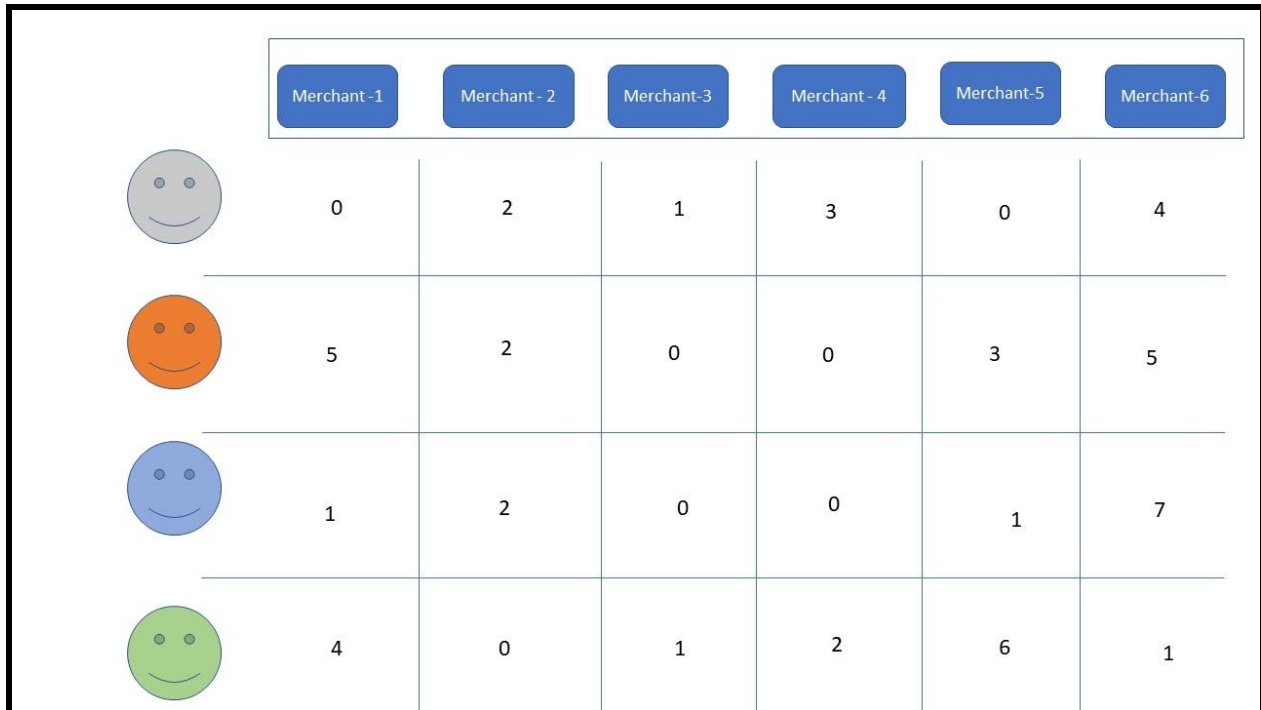
In statistics, Pearson correlation coefficient is a measure of the strength of a linear relationship between two variables. It is denoted by 'r'. A best fit line is drawn between the two variables and coefficient 'r' indicates how far is each point away from the best fit line. Pearson correlation is calculated as:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

*Source: [Source](#)

Steps:

1. For each category with its top 15 merchants a sparse matrix was created between user ids as rows and merchants as columns. The values were the count of each user's visit to a specific merchant. Figure 8 shows a sparse matrix of a toy data as an example.







	Merchant - 1	Merchant - 2	Merchant-3	Merchant - 4	Merchant-5	Merchant-6
	0	2	1	3	0	4
	5	2	0	0	3	5
	1	2	0	0	1	7
	4	0	1	2	6	1

Figure 8: Sparse matrix of users and merchants

2. Another series is then created that contains the number of visits users made to a specific merchant. This specific merchant is a merchant to whom we are comparing other merchants and recommending to a specific user.
3. Compute Pearson correlation between the matrix and the series created in the above steps by using panda's method 'corrwith()'. This will produce a series with numbers between -1 and 1, showing correlation between all the merchants and the merchant_user_visited. Closer the score is to 1 or -1, greater is the correlation between the two entities. Figure 9 below shows an example where merchant-3 is being compared with all the other merchants using the number of counts the users visited those merchants. It shows that based on the Pearson correlation score, Merchant-4, Merchant-6 and Merchant-2, are closest to Merchant-3 in comparison. The formula to compute Pearson correlation is

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

*Source: [Wikipedia](https://en.wikipedia.org/wiki/Cosine_similarity)

Pearson Correlation Scores						
	Merchant-1	Merchant-2	Merchant-3	Merchant-4	Merchant-5	Merchant-6
Merchant-3	-0.2425	-0.5774	1	0.9623	0.2182	-0.8083
Strength	Very Weak	Moderate	Strongest	Strong	Very Weak	Moderate

Figure 9: Pearson Correlation Scores - Merchant-3 compared to other merchants

- In order to make the recommendation more relevant, two filters were created. Number of visits to a specific merchant and mean amount of dollars spent on that specific merchant. If a merchant is highly similar to the merchant in question but the average amount spent there is less than the second best match then the recommendation system will favor the second best merchant and show that to the user. These filters can be controlled by FinGoal in order to create recommendations to their liking and if they are set to zero then only the correlation score is used for recommendations.
- Top 3 filtered merchants are shown to the user.

Test:

To test if the model shows the similar merchants, the code asks for two inputs, the category and the merchant within the category. All the above steps are then performed on the data relevant to the category and the merchant within the category. For example, for the category 'Restaurants/Dining', McDonald's was chosen as the merchant to whom other merchants will be compared. The Pearson correlation score along with the two filters set to their thresholds (Threshold total amount of visits >6, and Threshold_Num_MeanAmt > 10) following restaurants are recommended. The message shown is

"Other merchants you may like ['wendy's', 'chick-fil-a', 'taco bell']"

Several other tests were performed using other merchants and categories. Not all resulted in 3 recommendations with the filters in place. However, if filters are tweaked or ignored top 3 merchants are shown in all the categories.

One thing to note here is that since the data is quite sparse and there are many merchants that are similar to the merchant in question (in theory), the score for those merchants are closer to 0 rather than 1 (or -1). One Starbucks is being compared to four different names for 'ziggy's coffee and bars' therefore even though those two are similar the correlation will not be able to find it unless extensive data cleaning is done on close to 90k records.

Cosine Score:

Cosine similarity is the cosine of the angle between two non-zero vectors. The cosine of 0° is 1, showing the highest similarity between the two vectors. The cosine of 90° is 0, showing no similarity between the two vectors. The results are between [0,1]. The formula to calculate cosine similarity is as follows

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

*Source: [Wikipedia](https://en.wikipedia.org/wiki/Cosine_similarity)

Steps:

1. With the same toy data shown in Figure 8, Cosine similarity is calculated using scikit-learn's cosine_similarity method. The sparse matrix created before (see step 1 of Pearson correlation method above) is given as an argument to the function and function's output is the similarity scores between all the merchants.

2. To recommend merchants similar to the merchant a user likes or has just visited, the information is extracted from the resultant output of the cosine_similarity function.
3. Figure 10 below shows cosine scores between Merchant-3 and other merchants. Toy data shown in Figure 8 is used for this similarity comparison. Figure 10 shows that based on cosine scores, Merchant-4 and Merchant-5 can be shown to the user who is interested in Merchant-3 as additional places he can visit. It can be noticed that even though Merchant-4 got the highest score using both Pearson correlation and Cosine, the other merchants didn't get a similar comparison in scores. For example, Merchant-5 got the weakest score in Pearson correlation however with Cosine similarity it actually made it to top 2.
4. Similar filters are applied to the result of cosine similarity output, i.e. Number of visits to a specific merchant and mean amount of dollars spent on that specific merchant. This step remains the same as Pearson correlation written above.
5. Top 3 filtered merchants are shown to the user.

Cosine Similarity Scores						
	Merchant-1	Merchant-2	Merchant-3	Merchant-4	Merchant-5	Merchant-6
Merchant-3	0.436	0.408	1	0.981	0.626	0.371
Strength	Weak	Week	Strongest	Strong	Moderate	Very Week

Figure 10: Cosine Similarity Scores - Merchant-3 compared to other merchants

Test:

Similar to the Pearson model above, the code asks for two inputs, the category and the merchant within the category. For the same example used in Pearson correlation above, i.e. the category of 'Restaurants/Dining', McDonald's as the merchant to whom other merchants will be compared and using same filter thresholds, following restaurants are recommend to the user:

"Other merchants you may like ['starbucks', 'subway', 'taco bell']"

Several other tests were performed using different combinations of categories and merchants, and due to the nature of the data, sometimes there were no recommendations given if the filters are set. This was the same for Pearson correlation scores too.

Combined Cosine and Pearson Correlation Score

In order to consolidate the efforts put into cosine and pearson score separately and to also give a consistent none zero recommendation to the user, a logic was introduced where after calculating both scores following steps were performed:

- a. If cosine similarity resulted in 3 or more merchants → Show recommendations from cosine similarity results only
- b. If cosine similarity resulted in < 3 and pearson similarity resulted in < 3 merchants → Show recommendations by joining the results
- c. If cosine similarity resulted in 0 merchants → Show recommendations from pearson similarity results

The emphasis here is clearly given to cosine similarity because of the fact that during the test, Cosine scores often found correct merchants whereas the Pearson correlations often ended up empty. Cosine did sometimes show wrong results or had empty arrays for recommendations, but more than often it did work. In order to remedy for the times when it doesn't show top 3 results within the threshold of the filter, above logic can be performed and top 1 or greater merchants can still be recommended.

Test

For the same category and merchant used in the Pearson correlation and Cosine similarity above, the combined logic showed the following result.

“Other merchants you may like ['starbucks', 'subway', 'taco bell']”

Goal 2: Location Based Recommendation System

Statement: Based on the location of a merchant, other merchants in the vicinity are recommended to a user.

Strategy:

The data frame that was wrangled and prepared in EDA ipynb was imported to create a location based recommendation system. For this project, only the records in the state of Colorado (CO) were considered. Furthermore only categories with greater than 150 records were selected and their records were part of the data frame. The top categories were Restaurants/Dining, Groceries, Clothing/Shoes, Home Improvement, General Merchandise, Healthcare/Medical, and Gasoline/Fuel. To show distributions of the categories, plotly's scatter_mapbox method was used. Figure 11 below shows the distribution of transactional records in CO.

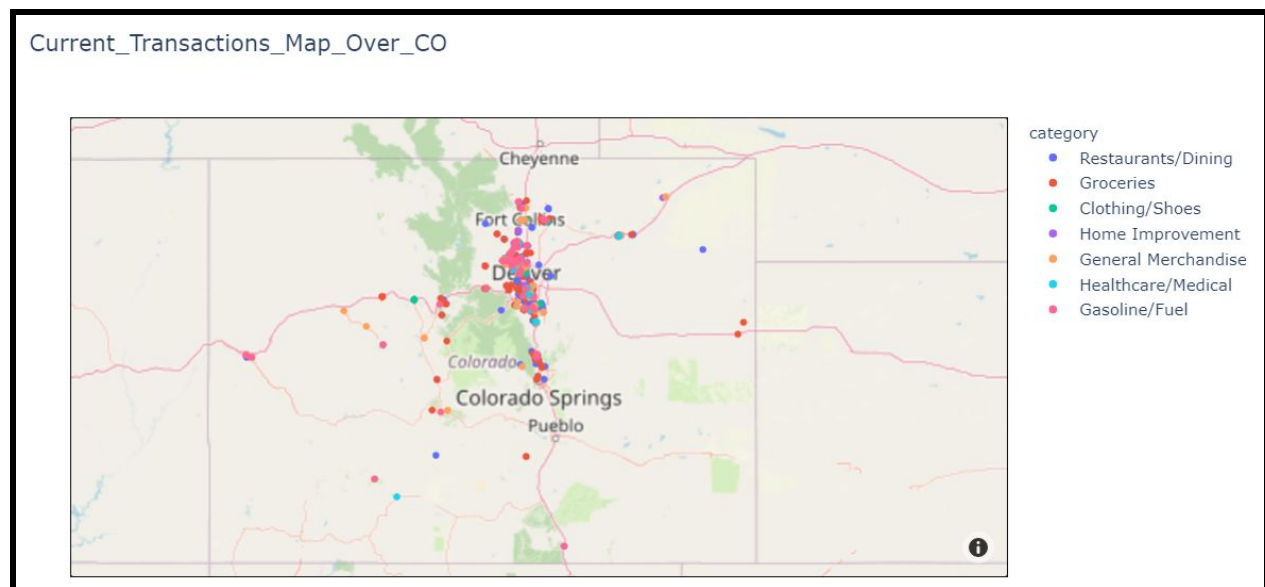


Figure 11: Distribution of transactions from top categories in Colorado

From the above figure it can be noticed that the majority of the transactions are in the middle of the state that is made up of cities like Denver, Boulder, Lafayette, Longmont etc. The West, NE and South CO have very few transactions.

Models:

To recommend similar merchants based on location the data needs to be clustered such that similar locations are within the same cluster. Clustering is grouping of unlabeled data points in a way that the points belonging to a cluster are similar to each other. The intra cluster similarity compared to inter cluster similarity should be high. The clustering methods like

KMeans, DBSCAN and HDBSCAN were used to organize the data points into defined clusters in this part of the project.

KMeans:

KMeans create clusters around the centroids and the number of centroids(K) is an input from the user. KMeans is a distance based clustering technique, where K number of random centroids coordinates are initialized and then Euclidean distance is calculated from each data point to the centroids. The data point is assigned to the nearest centroid. The process is repeated until all the data points are assigned to a centroid. Once the cluster is formed a new mean of the cluster is calculated. Mean of the cluster is the mean of the distances present in the cluster w.r.t centroid. The process is repeated until there is no change in cluster i.e. no data point leaves the cluster or is added to the cluster. This is called achieving the convergence.

Some of the advantages of KMeans are that it is a very simple algorithm to use and it can handle large datasets. KMeans is however very sensitive to outliers as those can impact the location of centroids making wrong predictions for the rest of the data point.

DBSCAN: (Density Based Spatial Clustering of Application with Noise)

DBSCAN is a density based clustering algorithm that creates clusters of dense areas of the data points and tagging the low density areas as noise. DBSCAN requires Epsilon(eps) value as input along with min_sample points. Epsilon(eps) is the maximum distance between the two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. The number of samples (min_samples) in a neighborhood for a point to be considered as a core point.

Some of the advantages of DBSCAN are that it can identify outliers easily and is not controlled by calculating different distances (means) in every iteration. It can identify noise i.e. points that cannot be considered as core points or boundary points. From an implementation perspective, sklearn doesn't implement a predict function for DBSCAN. This is because if a new point is introduced in the clustered data, chances are that the same clusters won't exist anymore because the new point will trigger the process of creating the clusters again and depending on where this point lands there could be an extra cluster created or tagged as noise. Therefore predict function is not a realistic function for DBSCAN or any density based algorithms.

HDBSCAN: Hierarchical Density Based Spatial Clustering of Applications and Noise

Unlike DBSCAN, HDBSCAN doesn't require eps input for it to form clusters. HDBSCAN is basically DBSCAN but with varying epsilon (eps) values. It requires a minimum cluster size (min_cluster_size) parameter which is the number of data points at which a cluster is to be formed. There are multiple steps in finding the flattened clusters in the data. These steps are,

- (1) Transform the space according to the density/sparsity. This is accomplished by identifying islands of high density data using distances between nodes and focusing on points that are closer together

- (2) Build a minimum spanning tree of the distance weight graph. Once the distance between the nodes is established, a spanning tree graph is created by using nodes as vertices and edges are the distances between the nodes
- (3) Construct a cluster hierarchy of connected components. This step can be visualized as a dendrogram. Each node is connected to similar nodes and each of those similar node groups are connected to similar groups.
- (4) Condense the cluster using minimum cluster size (min_cluster_size) parameter. The tree built in last step is broken down to smaller trees based on the minimum nodes required to be in a cluster
- (5) Extract the stable clusters. From the condensed tree evaluate which clusters are stable by looking at their lifespans. Larger the lifespan of a cluster is more stable.

Advantage of HDBSCAN is that it can extract clusters in a data with varying density sizes and it can mark points as Noise if they are not significant to a dense cluster and have to be pulled out from cluster formation.

Location Based Recommendation System using KMeans/DBSCAN:

1. Using KMeans with different numbers of features, clusters were created and a few points were tested. However, it didn't seem to work very well. For example since the aim here was to present near-by merchants to the user, the KMeans predicted merchants that weren't near-by to the current merchant user was visiting. This was tested visually only. If a test point was in Lafayette, CO, KMeans suggested merchants in Boulder, CO which is not close to Lafayette, CO. The code of this effort is present in the github repo.
2. Second attempt was made with DBSCAN. It did identify clusters in the data with points close to each other. However, since DBSCAN doesn't provide any prediction method once the clusters were formed there was no way to assess the accuracy of the suggested recommendations for a given test point. Therefore this effort was not fruitful and not submitted for evaluation.

Location Based Recommendation System using HDBSCAN:

The sklearn library does not have HDBSCAN currently implemented. The code was imported from [hdbscan library](#) and used in this project.

As mentioned before, the value of min_cluster_size is the key in building the location based recommendation engine using HDBSCAN. After experimenting with small values (5-10), it was noticed that there were a significant number of clusters formed but the noise (i.e. points that are not part of any clusters) was lower between 214-234. With larger numbers for min_cluster_size (12-20) there are fewer clusters but the noise was also very high (286-376).

In order to make the majority of the transaction data be part of the non-noise clusters the parameter of min_cluster_size was set to 5. By any means it wasn't a perfect labeling of the data points but this value generated the lowest number in the noise cluster therefore it was selected for this project. With the min_cluster_size equal to 5, noise data points were 214. This was an intuitive selection based on the experimentations performed with several values of min_cluster_size.

To use HDBSCAN, first an object is spun with min_cluster_size of 5, and with the argument prediction_data=True. The data frame, df_cluster with latitude and longitude information only, was used to train the model object. With the new model object trained the label information was extracted and appended to the rows of the original data frame. To visualize the clusters, the following plotly express graph was created. The first one is with the noise cluster added and second one is without noise cluster

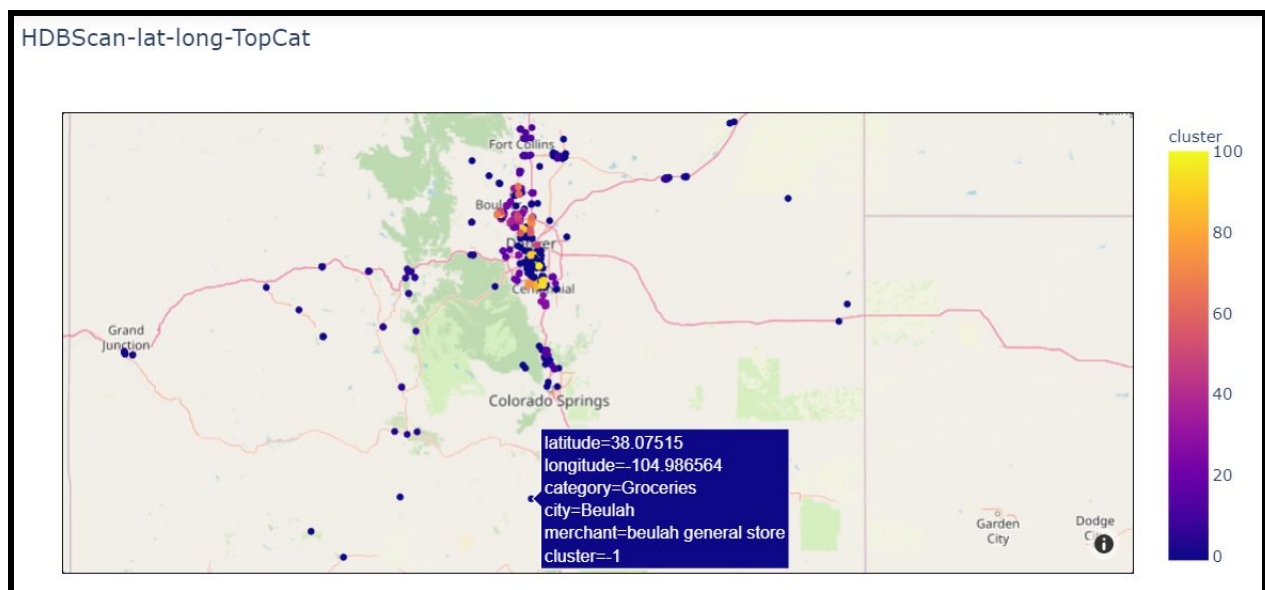


Figure 12: Visualization of clustered data with noise

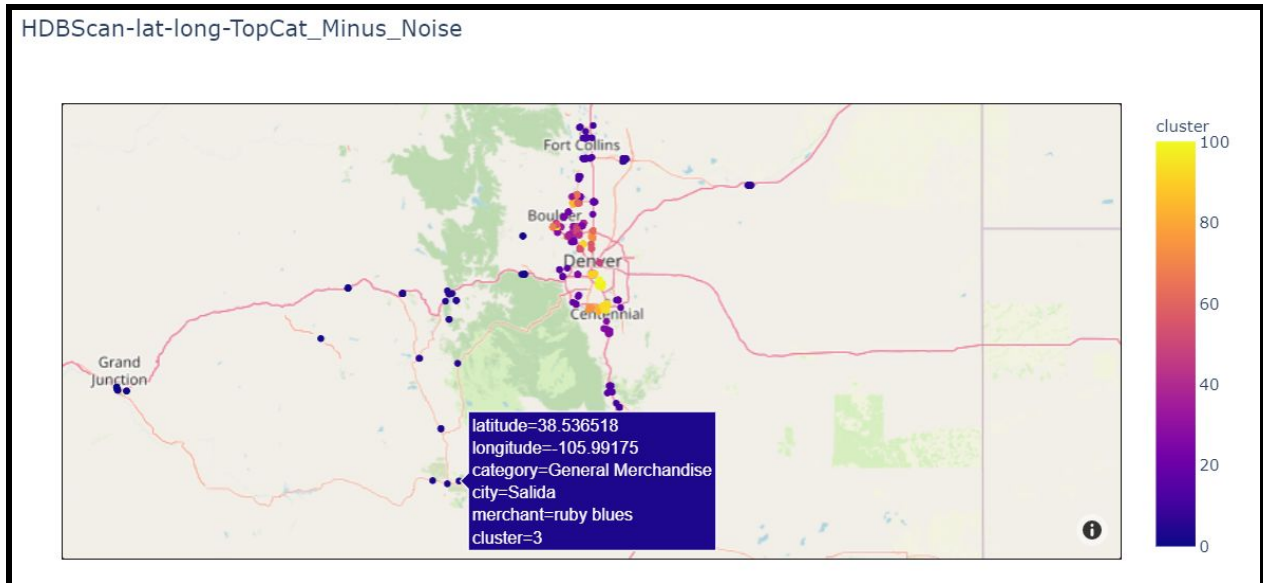


Figure 13: Visualization of clustered data without noise

Comparing Figure 12 and Figure 13, it could be observed that HDBSCAN was able to capture a lot of data points within clusters however there are significant misses as well. This can be observed from zooming into the Denver metro area map where the majority of the records were located. Figure 14 and Figure 15 are the zoomed in versions of the above two maps respectively

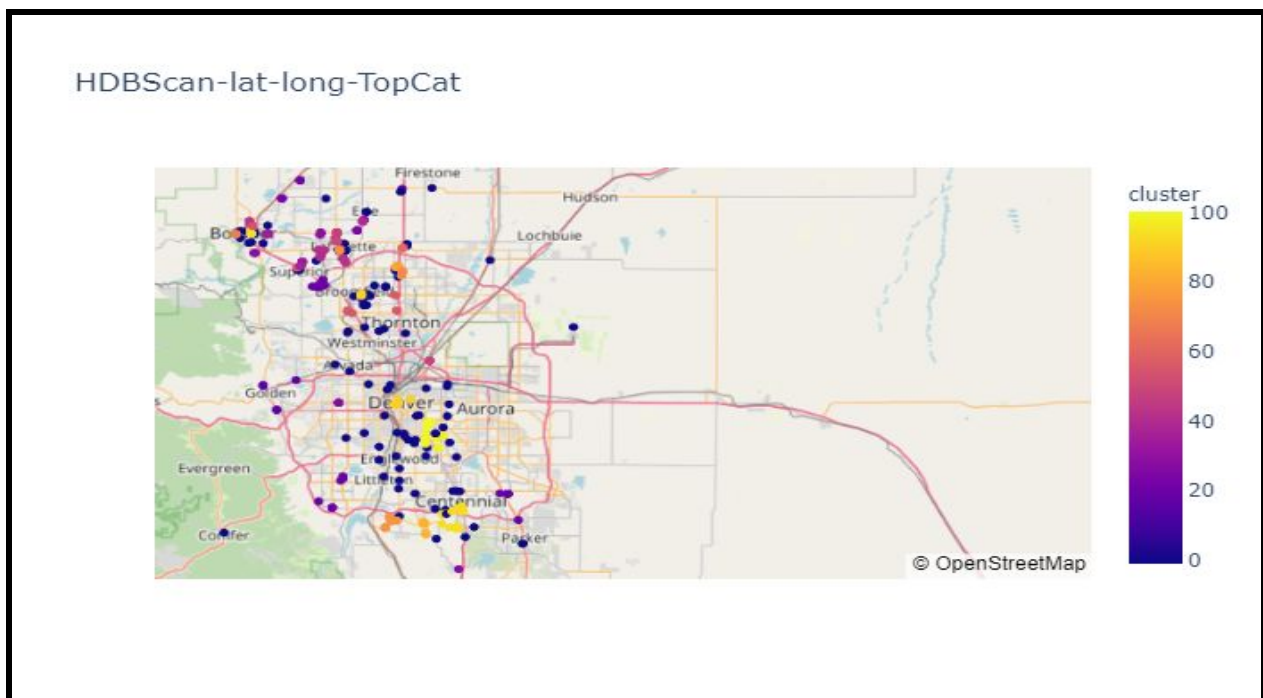


Figure 14: Zoomed in map to view clusters(with noise) within Denver and surrounding areas

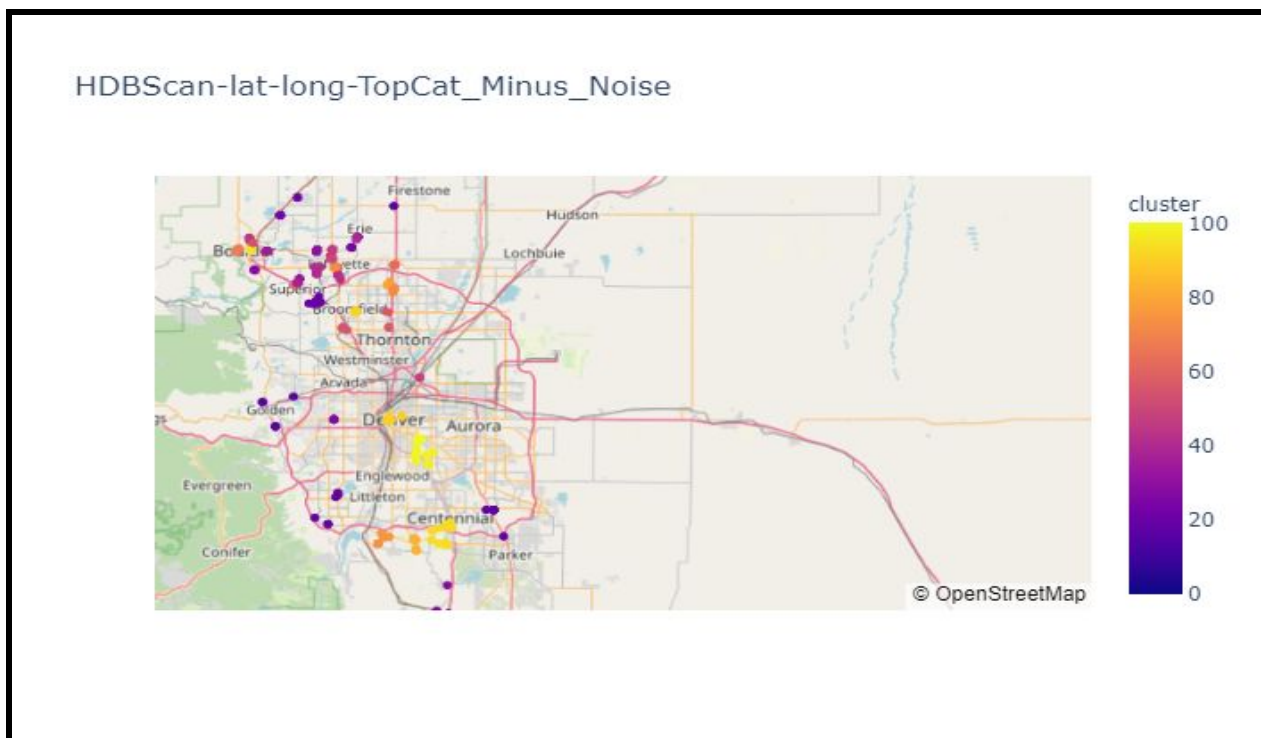


Figure 15: Zoomed in map to view clusters(without noise) within Denver and surrounding areas

From the zoomed in versions of the maps above, it can be observed that while closer data points are clustered correctly, a lot of points around those clusters were tagged as noise as well. The cause of concern here is the data points between the Denver and Aurora area where it seems like that HDBSCAN should have created additional clusters or added more points around that area into the existing clusters.

Predictions on the test data points:

Once all the data points are tagged and hdbscan object has been trained, the method `approximate_predict` is used to predict which cluster the test data point will belong to. For density based clustering algorithms like DBSCAN and HDBSCAN it is nearly impossible to add a new point into the data set and not go through the process of creating the clusters again. The new data points can and should be able to alter the underlying clustering. Given the new data point, it might become necessary to spin a new cluster or the point lies between the two clusters that they now become one. If the actual clusters change, the labels collected after training the model would change too and therefore prediction accuracy would become null. HDBSCAN library has an alternative way of predicting cluster assignment and it is rightfully called `approximate_predict`. When a new data point comes into picture, the approximate cluster is predicted by determining where in the condensed tree the new data point would fall.

Once the data point is assigned to a cluster, the top 5 merchants that belong to that cluster are presented to the user with their location, name and city information.

Test:

Using a random data point from the data set, the `approximate_predict` method was used to predict the cluster of this test point. If the point is tagged to cluster (-1) i.e. noise, the code refrains from recommending any merchant. If a data point does belong to a defined cluster top 5 merchants are displayed. For example is the customer who was at McDonald's located at location (40.15403, -105.10333, 'Longmont'), following top 5 merchants close to his location are displayed

	merchant	city	latitude	longitude
0	first watch	Longmont	40.15203	-105.09626
1	modern market	Longmont	40.15203	-105.09626
2	la mariposa	Longmont	40.15203	-105.09626
4	safeway fuel	Longmont	40.15203	-105.09626

Figure 16: Output of location based recommendation system