



# **Recommendation System Using Credit Card Transactions**

## **Capstone 2**

Fariha Baloch

Springboard - Data Science Career Track



# Motivation & Purpose

- **FinGoal** is a start-up company in Denver, CO. They build tools for financial institutions to help their customers save money by finding patterns in their spending habits
- They wanted a recommendation system that recommends cheaper items to users based on the item , current location of the user and the price they paid
- **Motivation & Purpose**
  - Get a real world experience building two recommendation systems that will actually be utilized in a real product
  - Working with data scientists who have done these kind of projects & gain knowledge from them
- **Goal**
  - By building the models for recommendation system , FinGoal will be able to speed up their implementation timeline



# Data

- The data for this project came from [FinGoal](#).
- The data was anonymized and converted to json format by the company.
  - The file size was 48.2MB.
- Used `pd.read_json` function to read the file into a data frame.
  - The file had 29 columns and 90086 rows of transactions
  - Information about the Item bought or price of that item were not in those 29 columns.
  - AmountNum variable is total amount paid by the customer
  - Based on that the goal was modified to recommend similar merchants rather than items



# Problem Statement

Create a recommendation system using credit card transactions records. The data contains transactions from multiple users and contains GPS records. The aim is to provide following two type of recommendation system

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

# Data Wrangling - Dropped Columns

- Criteria to drop columns:
  - Multiple occurrences
  - Columns that have NaN values -
  - Columns that can be represented by other columns
  - Deleted Client\_id → source of the transaction
  - Deleted country and currency → Dealing with USA data only

\*13 columns left

```
In [7]: df.columns
```

```
Out[7]: Index(['accountid', 'address', 'amountnum', 'category', 'categoryid', 'city',  
              'client_id', 'country', 'currency_code', 'date', 'detail_category',  
              'high_level_category', 'latitude', 'longitude', 'original_description',  
              'pending', 'simple_description', 'state', 'transactionid', 'uid',  
              'zip_code', 'accountId', 'category0', 'finsight_api_uid',  
              'finsight_image', 'insight_ctaurl', 'insight_text', 'original_uid',  
              'tenant_id'],  
             dtype='object')
```



# Data Wrangling

- Used columns **zip\_code** and **city** to fill NaNs of each other
- Multiple **uid**'s with missing **accountid** were filled with new unique numbers
- **Category** columns had 67 unique categories
  - Deleted incoming monies to a customer, for example, PayChecks/Salary
  - Categories left → 33
- In the **amountnum** column 75% of dollar amount spent by consumers was less than \$52. A few of the consumer purchases were > \$6000 and were considered outliers therefore those rows (44) were deleted
- Duplicates were dropped

---

# Exploratory Data Analysis - EDA

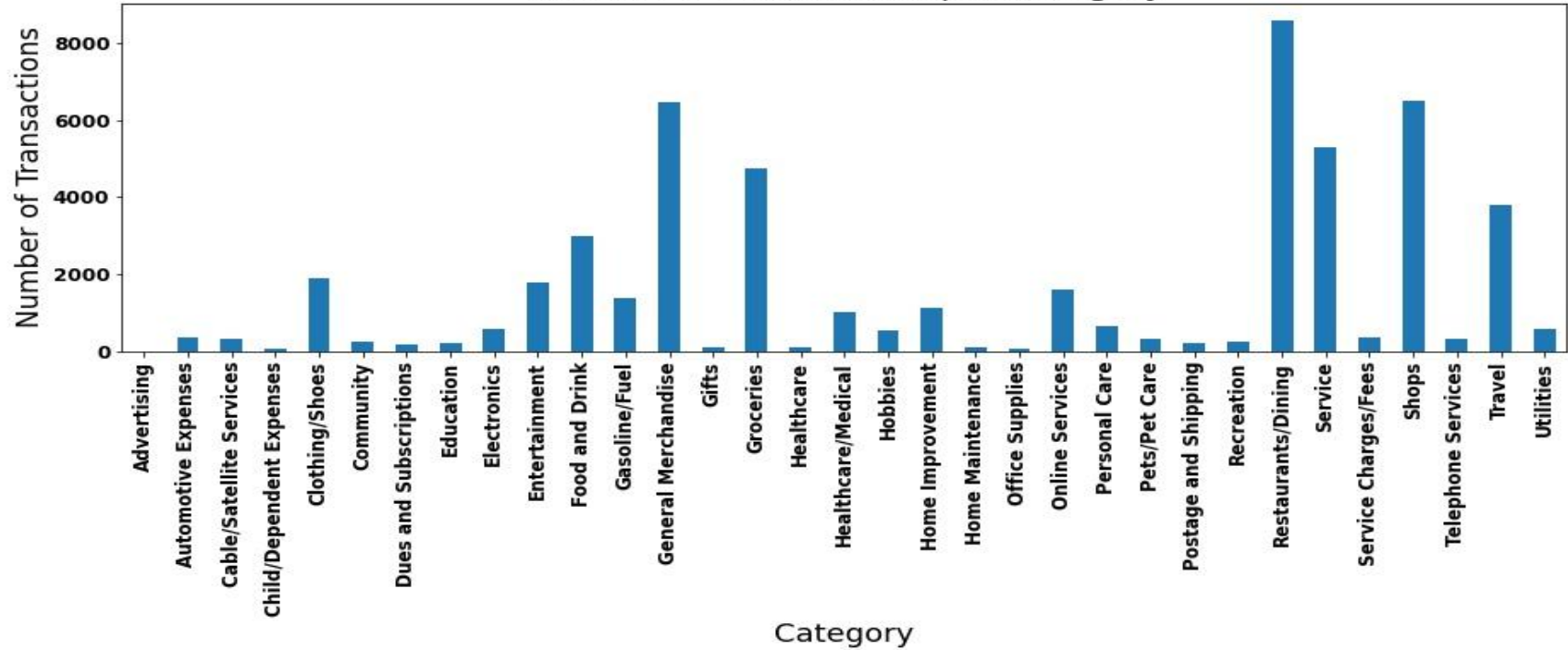


# Exploratory Data Analysis

- How are the transactions distributed per category?
- How are the transactions distributed per state?
- How are the users spending money in different categories?
- Which category gets the most amount of dollars? And how are those distributed among top merchants in that category?
- What is the trend for the month to month average amount spent by users?



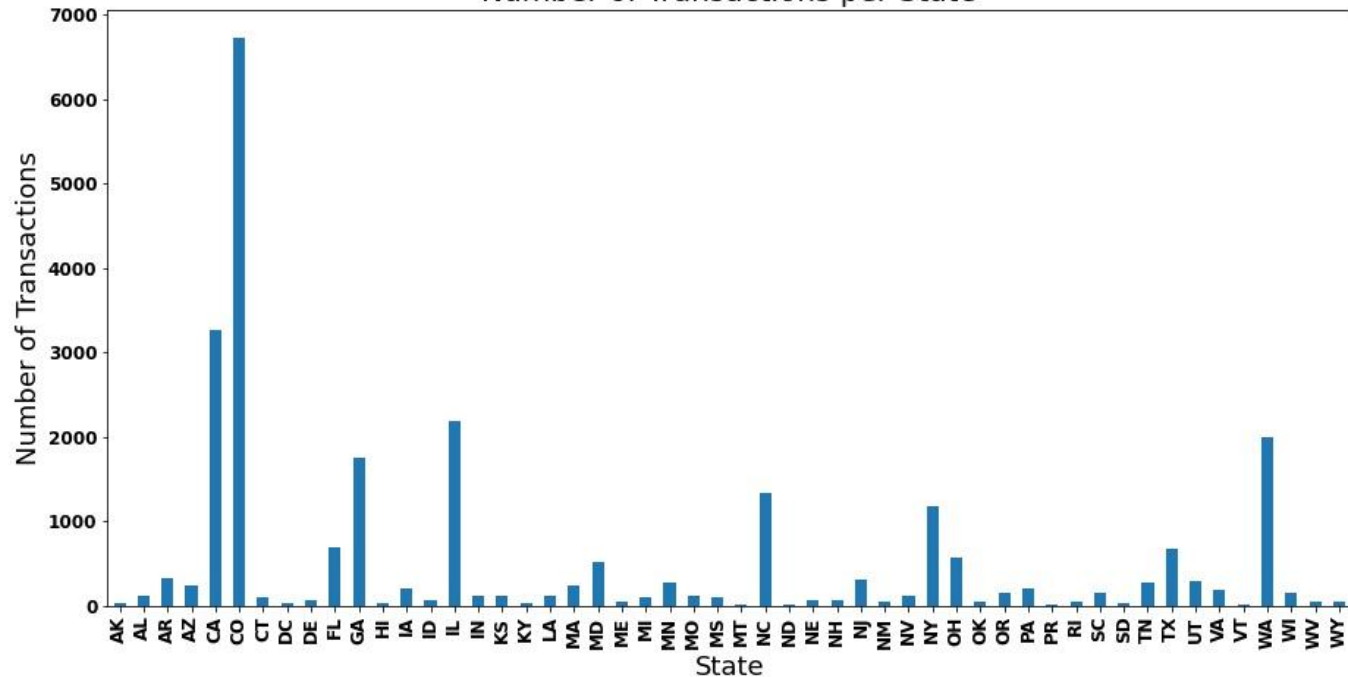
Number of Transactions per Category



Out of 33 categories,  
Restaurants/Dining have most  
transactions (>8k)

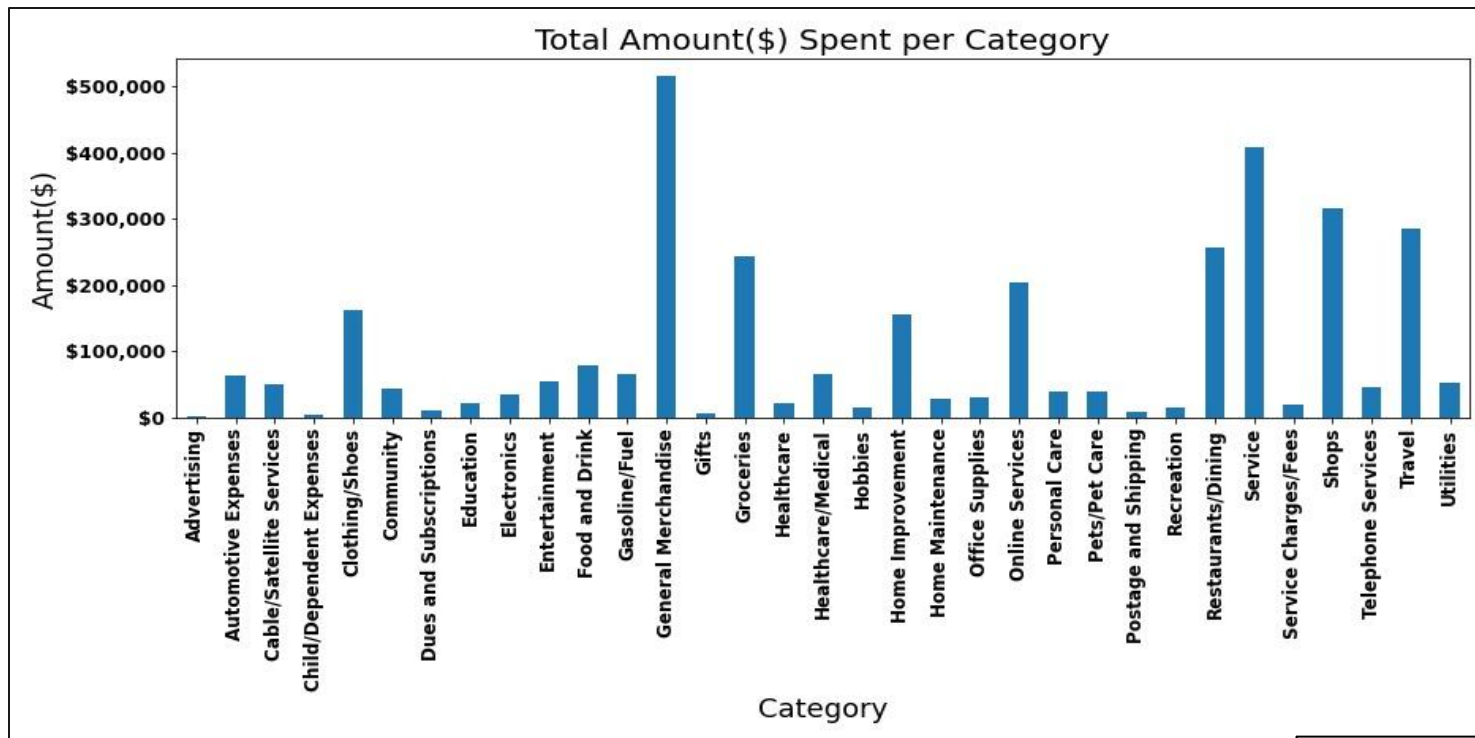
General Merchandise, Shops, Service  
and Groceries follow next

Number of Transactions per State



CO has the most transactions(~7k)

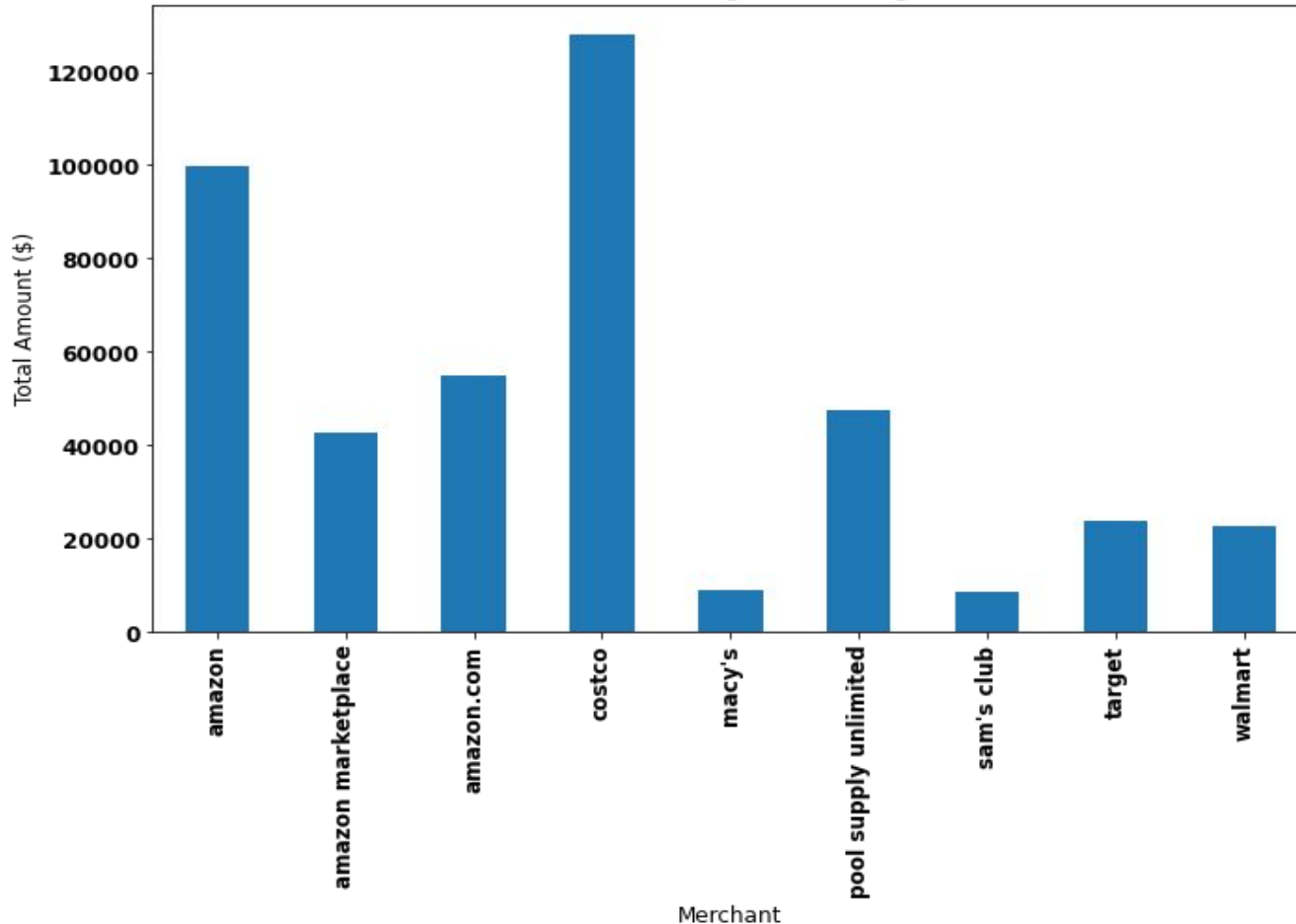
Majority of the states have less than 500 transactions



General Merchandise brought in close to \$500k

Service, Shops, Travel, Restaurants/Dining and Groceries are some categories where customer spent between \$250k - \$400k

General Merchandise Highest Earning Merchants

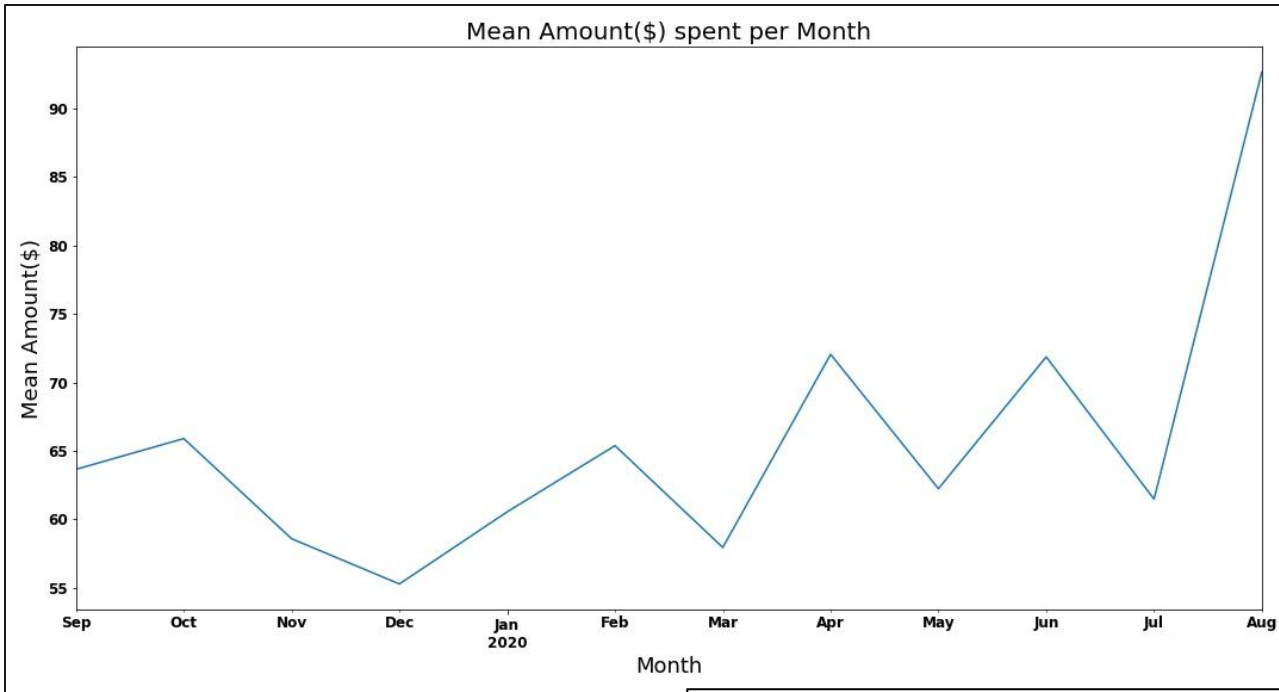


Users spent \$128K at Costco (highest amount in the General Merchandise category)

All the top names in this category are well known except for 'pool supply unlimited'.

Upon investigations it was found that some high dollar amount in 'pool supply unlimited' transactions were duplicates.

## DateTime Analysis

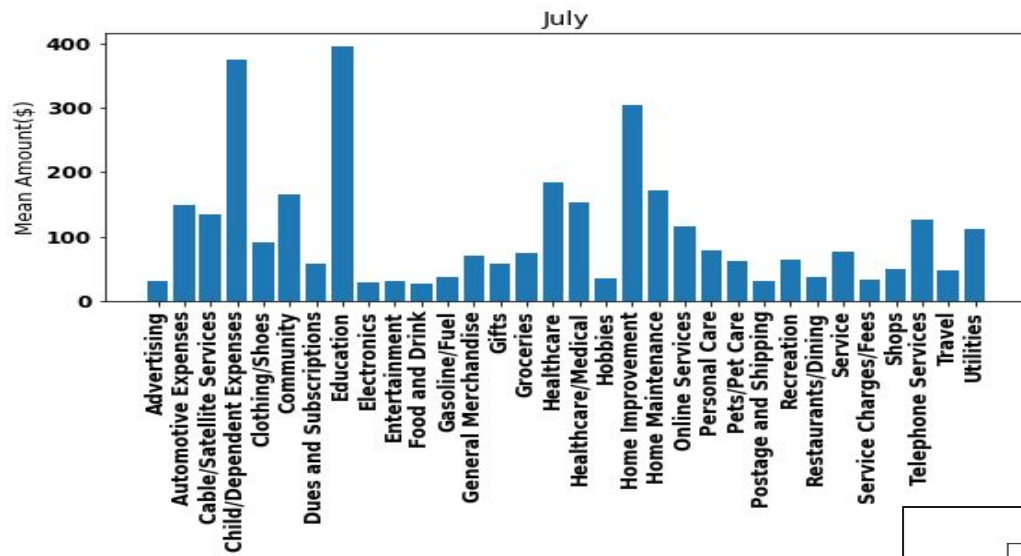


Clear spikes in the data can be noticed in different months

In March a huge dip occurs - Due to Corona sales went down

In April the data shows a high mean amount, however most of the transactions are in "Office Supply" category. Corona did affect this month as well but not in all categories

In August there is a huge mean amount increment from July - Users spent in all categories



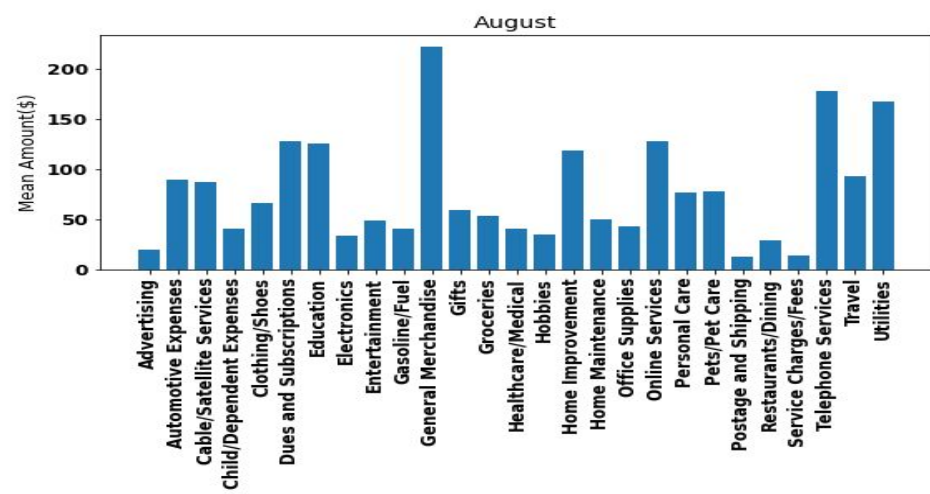
Total Transactions July - 19666  
Total Transactions Aug -2090

Total Transactions Sum July - \$120,893  
Total Transactions Sum Aug - \$193,644

General Merchandise - mean amount increased in August compared to July (~\$100 to ~\$200) - reason might be the start of school year

Child Dev Expenses decreased in Aug (~\$400 to ~\$50) - reason might be the start of school year

Overall there was increase in spending in Aug from amount(\$) perspective. However the number of transaction were lesser than July ....Therefore a spike in Aug Mean Amount



---

## Goal 1: Collaborative Filtering

Based on a user's preference, recommend similar merchants



# Recommendation System V1

- Goal 1:
  - The recommendation system should recommend similar merchants to the customer based on similarities between the merchants





## Recommendation System - Collaborative Filtering

Collaborative Filtering: The process of collaborative filtering for a recommendation system relies on information (ratings, count etc) collected by users in the system

For example, to recommend a movie to a user who has not watched it before, the recommendation system collects ratings from the users that have watched it and are similar to the specific user and recommends the movie (or not) to that specific user

For this project, the value between rows (uid) and columns(merchant) is the total number of times the user visited a merchant. Not a rating given to a merchant



## Some more Data Wrangling

- Went through 'simple\_description' column and extracted the names of the merchant as much as possible.
  - Deleted words like 'pending', 'visa' and all other forms of these before/after the merchant names
  - This helped with consolidating same merchants together
- There are 33 categories and there are 13 categories with > 900 transactions
- Only those categories were used to build the recommendation system
- Each category has several number of merchants therefore a data frame with top 15 merchants is created to train the models

category	
Restaurants/Dining	8261
Shops	6021
General Merchandise	5498
Groceries	4707
Service	4064
Travel	3662
Food and Drink	2836
Clothing/Shoes	1749
Entertainment	1576
Online Services	1522
Gasoline/Fuel	1347
Home Improvement	1091
Healthcare/Medical	970



# Steps

- A merchant with which the similarity is supposed to match is selected (merchant\_selected)
- Create filter(s) apart from the similarity score on which the final recommendation should be posted
  - Popularity of the merchant among other users i.e. the count of the times users went to this merchant (Count\_per\_merchant)
  - Average amount spent by users on this merchant (Av\_Amount\_per\_Merchant)
- A sparse matrix is created with rows as uid and columns as merchants. The values in cells are the number of times the specific merchant has been visited by a user (df\_sparse)
- A series or dataframe is created that only shows popularity (number of visits) of the merchant in question with all the users (df\_num\_visit\_at\_merchant\_selected)



## Models

- To evaluate similarity between merchants two methods were used
  - **Pearson Correlation** and **Cosine Similarity Score**

# Models - Pearson Correlation Coefficient

- **Pearson Correlation** - In statistic 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](#)

- Using Pandas method '`corrwith()`' pairwise correlation between rows of two dataframes is calculated (df\_sparse & (df\_num\_visit\_at\_merchant\_selected)).
- This produces a series of numbers between -1 and 1, showing correlation between all the merchants with the specific merchant. A number close to 1 or -1 shows high similarity



# Models - Cosine Similarity

- **Cosine Similarity Score** - The similarity between two nonzero vectors is determined by measuring cosine of the angle between them. The formula 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)

- Using sklearn's function '**cosine\_similarity**', compute the similarity score between the merchants. The result is a nxn sparse matrix, where n is total number of merchants in category data. A number closer to 1 shows high similarity and vice versa for the number closer to 0



## Final Step

Using the defined threshold for acceptable similarity score and for the defined filters ,show the results that match the criteria

The filter looks like this:

```
Output_list=  
  ( merchant_similarity_score >= Threshold_Sim) &  
(Num_Times_Merch_Was_Visited > Threshold_Num_Visit) &  
(Mean_Amt_Per_Merch > Threshold_Num_MeanAmt) &  
(merchant_recommended != merchant_in_question)
```



# Tests and Results

- Top 3 merchants are displayed
  - Based on similarity scores
  - Filters thresholds
- One was not better than the other - misses were made by both similarity scores
  - Data was unbalanced for the merchants within categories
  - One merchant having multiple names also didn't help with the scores
- FinGoal asked to rebuild a recommendation engine based on selected method of sim\_score and displays all the merchants with their respective scores
  - This took care of empty suggestions
  - Merchants with lowest scores (showing no correlations) were being recommended



---

## Goal 2: Collaborative Filtering

# Location Based Recommendation System



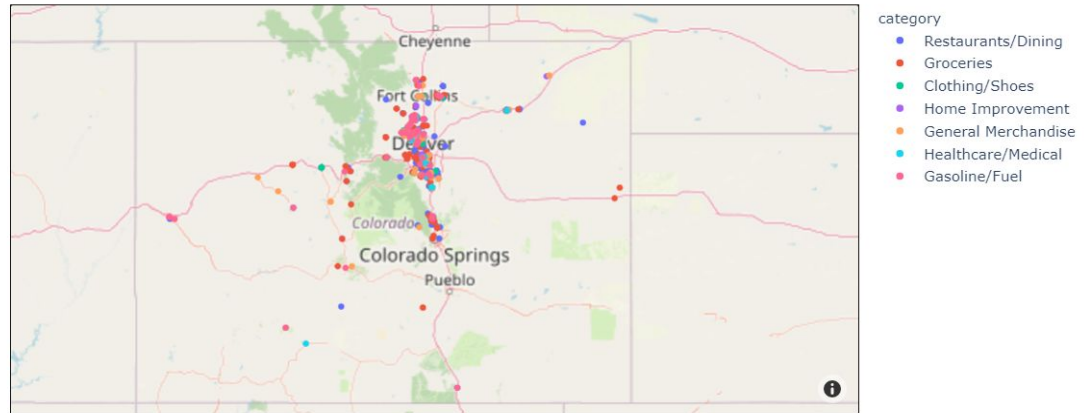
# Location Based Recommendation System

- Goal 2:
  - Based on the location of a merchant, other merchants in the vicinity are recommended to a user

# Data

- For this part of the project only records in Colorado are selected
- Since not all categories are equally represented in the data, only categories with greater than 150 records were selected
- Corrected a few coordinates that were tagged in CO but on plotly showed in Pacific Ocean
- Lots more wrangling was done for missing cities, missing long/lat and wrong state for coords in other states etc.

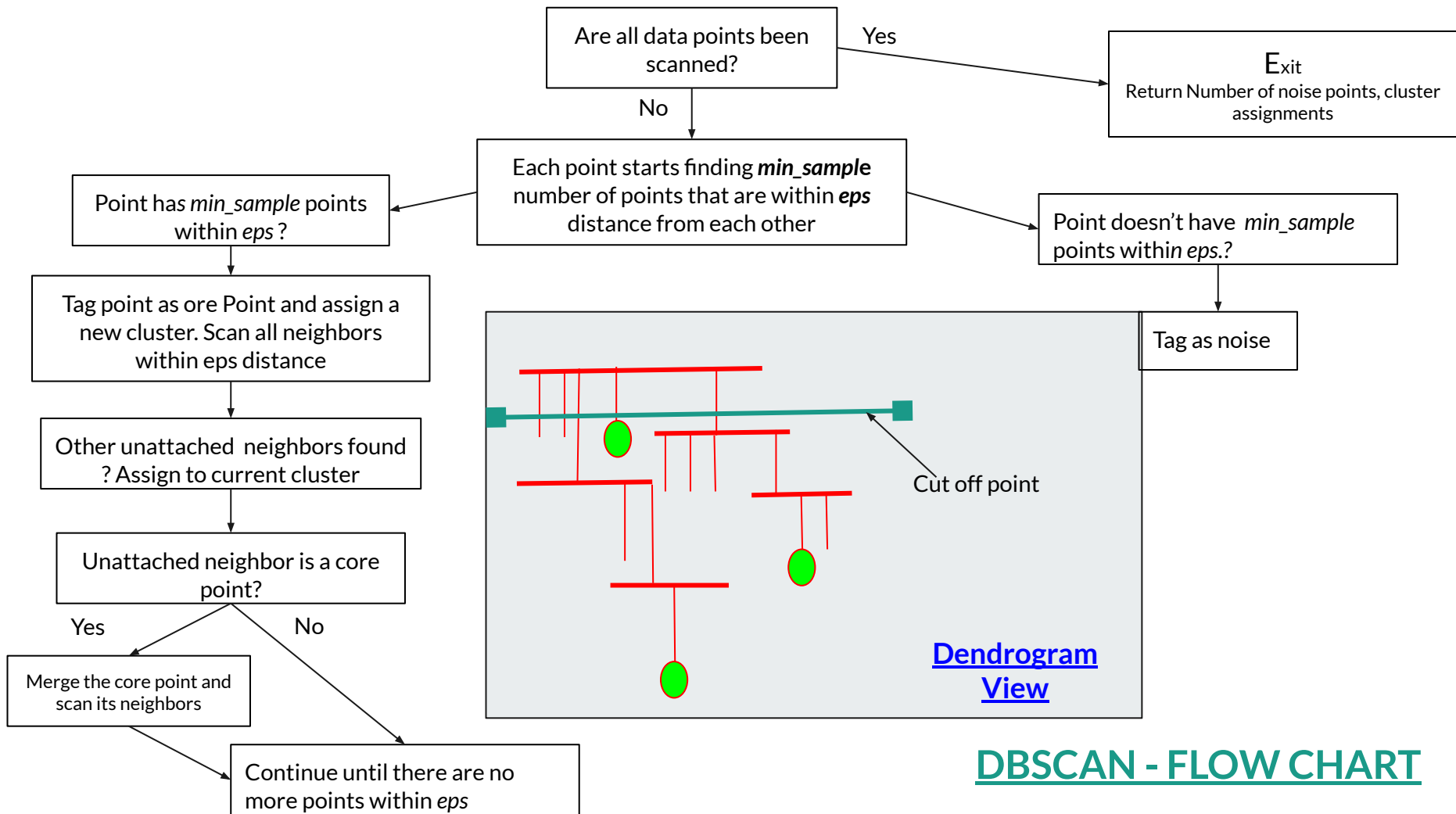
Current\_Transactions\_Map\_Over\_CO





# Models

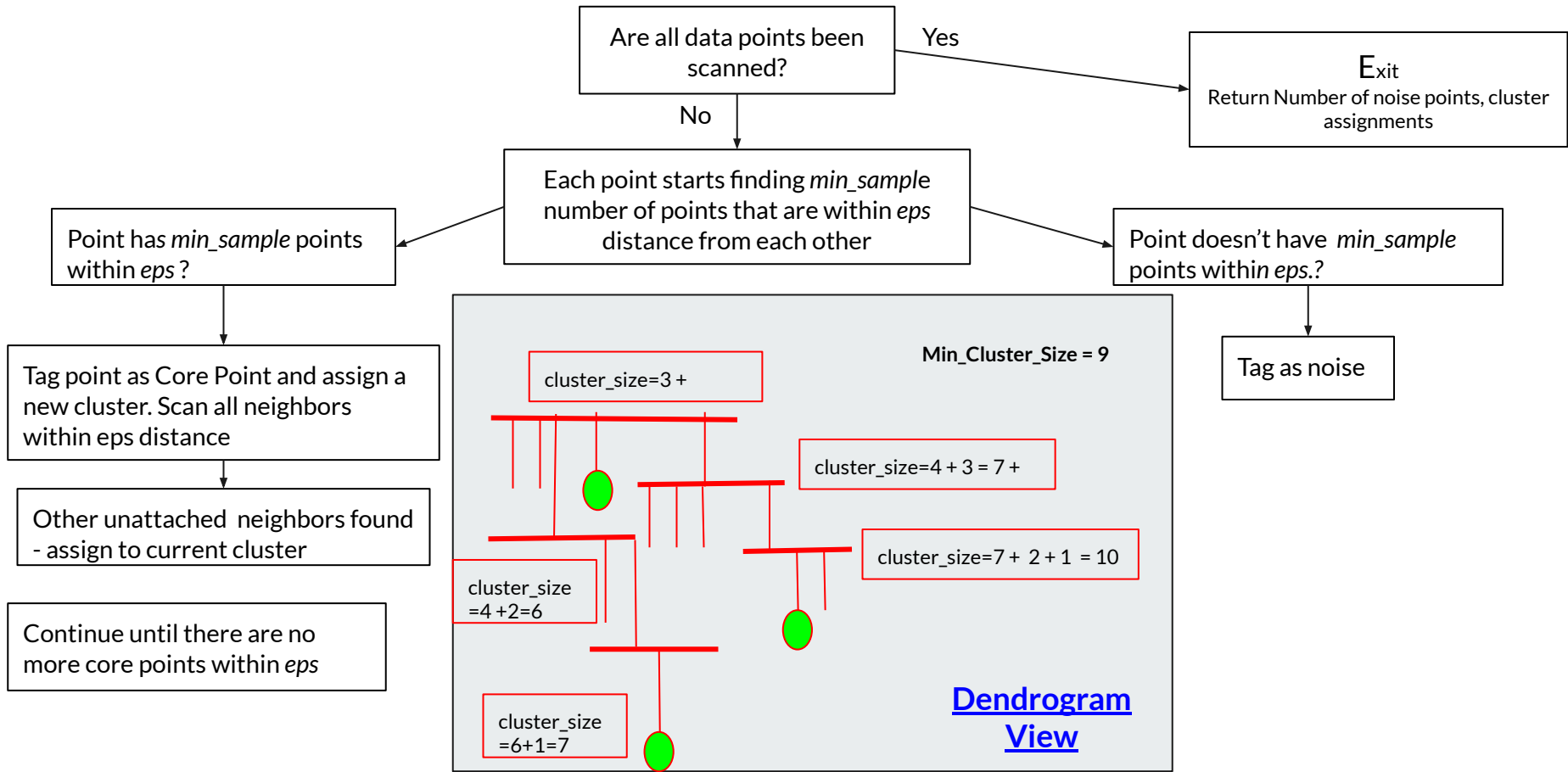
- Three clustering methods chosen were
  - KMeans, DBSCAN and HDBSCAN were used for this project
- **KMeans**
  - Very simple algorithm for clustering but do require number of clusters to be created as as input
  - Doesn't cluster well the data is dense on some locations and sparse on other location - KMeans works best when data is spherical in nature
  - Didn't produce optimal clusters for this project and therefore wrong recommendations were generated
  - Not recommended to the client
- **DBSCAN (Density Based Spatial Clustering of Application with Noise)**
  - Is perfect for a data with varying levels of density
  - Requires epsilon(eps) and min\_samples as inputs
  - The sklearn library does not contain a predict function.
  - Could have chosen classification route but data was highly imbalanced so it wouldn't have worked [5 vs 100 points]
  - Not recommended to the client





# Models - HDBSCAN

- **HDBSCAN (Hierarchical DBSCAN)**
  - Unlike DBSCAN HDBSCAN finds clusters of varying hierarchical densities
  - Performs DBSCAN until the point where core points find other core point
  - Instead of merging other core points, it creates a tree and keeps track of the tree's growth
  - Once the cycle is ended, it goes through, a core point starts finding min\_cluster\_size nodes to create a cluster
  - If cluster cannot be created then nodes are tagged as noise and the process restarts until there are no more points left to be scanned



## HDBSCAN - FLOW CHART



# Model - HDBSCAN

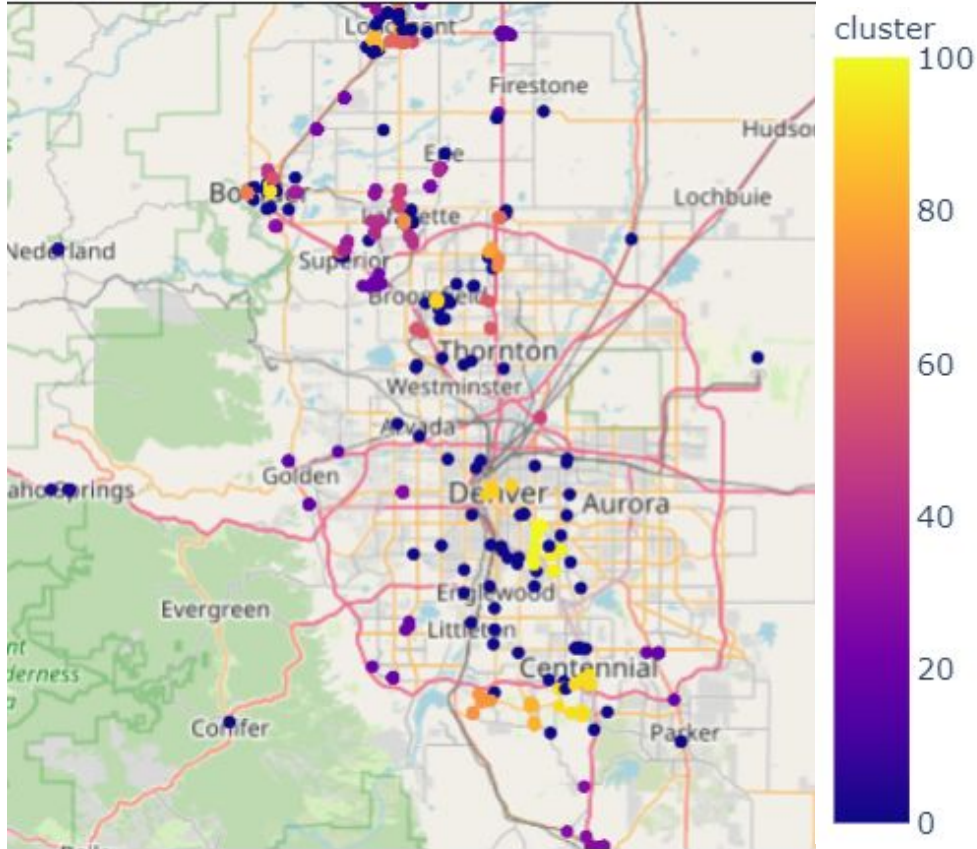
## Steps

- Hdbscan library was imported to apply it on the data
- Only required input parameter is min\_cluster\_size
- Identify the best min\_cluster\_size and apply hdbscan to create clusters (min\_cluster\_size=5)
  - Values (5-10) ~ clusters (214-234) ~ minimum points in noise
  - Values (12-20) ~ cluster (286-376 ~ maximum point in noise
- Using min\_cluster\_size, execute hdbscan and find clusters
- Predict approx cluster of test point
- It uses method approximate\_predict to identify a cluster from a trained data for any new data point
- Verify that the results make sense - i.e. a data point in Lafayette, CO is not matched with point in Vail, CO

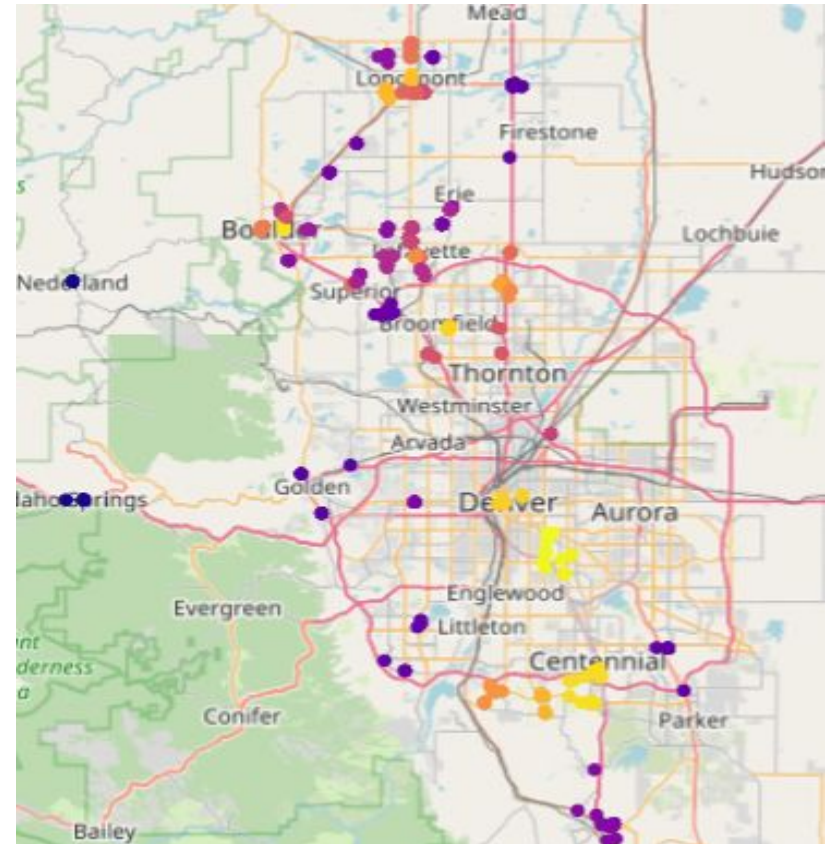


# Clustering with HDBSCAN - Zoomed In Map

With noise



Without noise





# Clustering with HDBSCAN

- **Observations:**

- The parameter `min_sample_size = 5` generated minimum number of noise data points
- Even though overall the noise points were lower, there are many still existent within high density area of Denver and surrounding area
- From the zoomed in versions of the maps 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 made better clusters or added more points around that area into existing clusters
- IT IS NOT PERFECT BUT BETTER THAN KMEANS

## Test and Result

- Arbitrarily chosen test points from the data set were provided to the model's `approximate_predict`. Following shows the result from one of those tests
- The data point was
  - (40.15403, -105.10333, 'Longmont', "mcdonald's")

	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

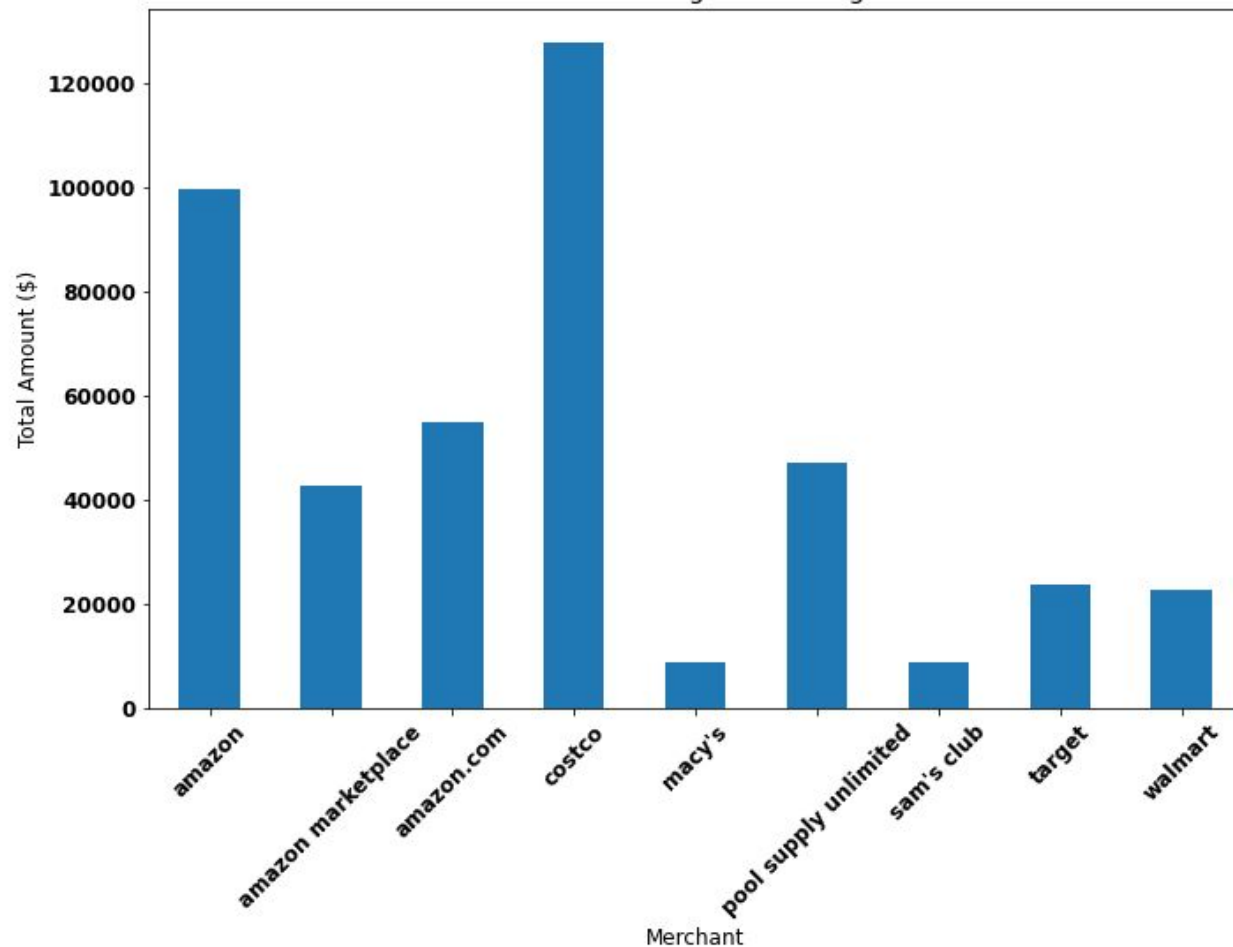
---

# Backup Slides

## Filling one column using another column's values

```
#functions
def fill_NaN_between_two_columns(df, col1, col2):
    df[col1] = df[col1].fillna(df[col1].groupby(df[col2]).transform('first'))
    df[col2] = df[col2].fillna(df[col2].groupby(df[col1]).transform('first'))
```

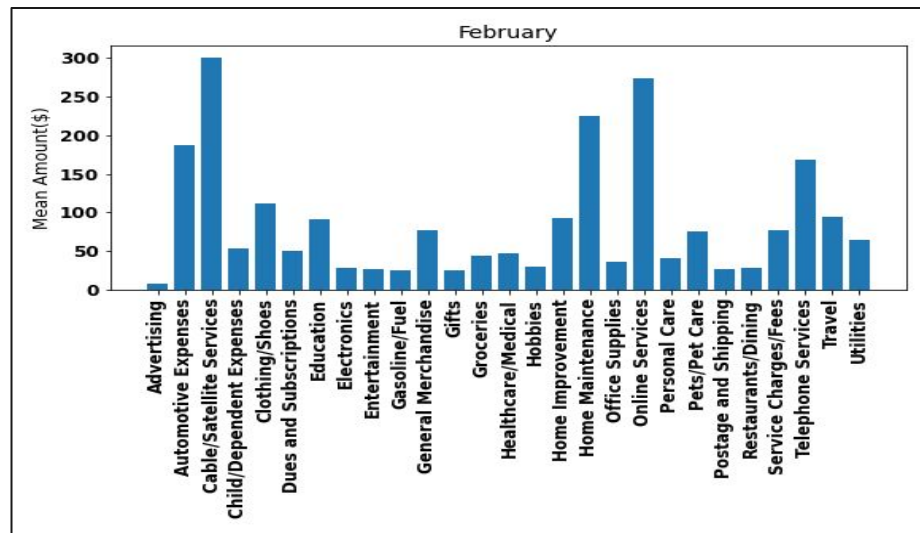
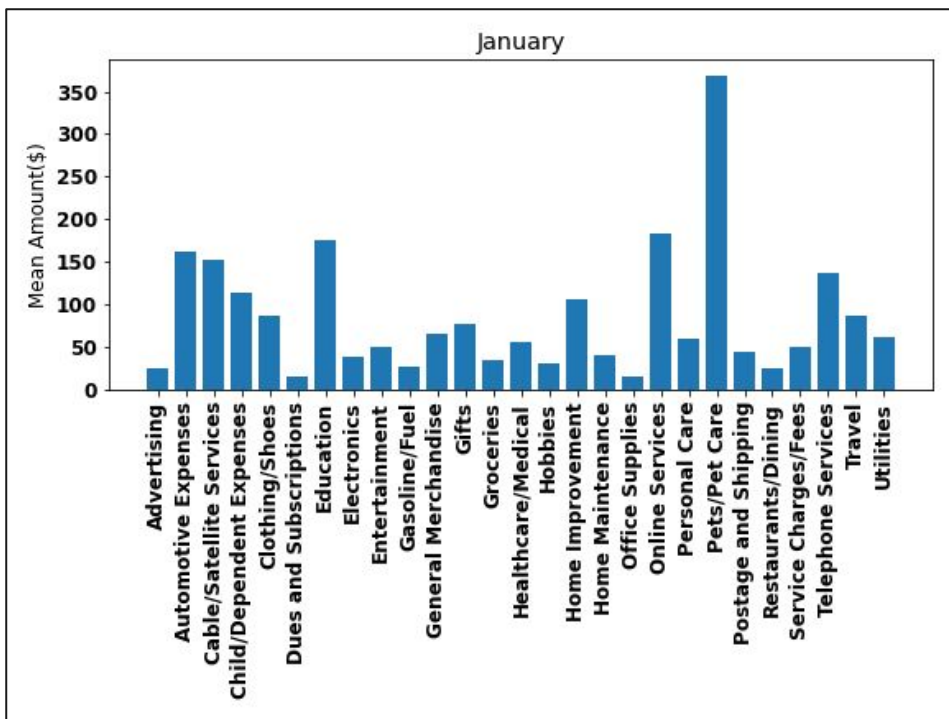
General Merchandise Highest Earning Merchants

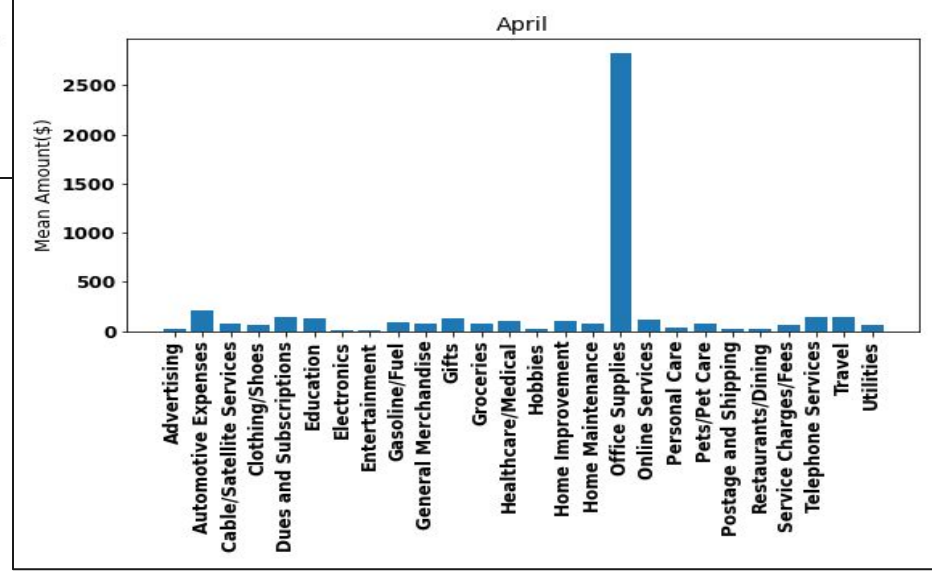
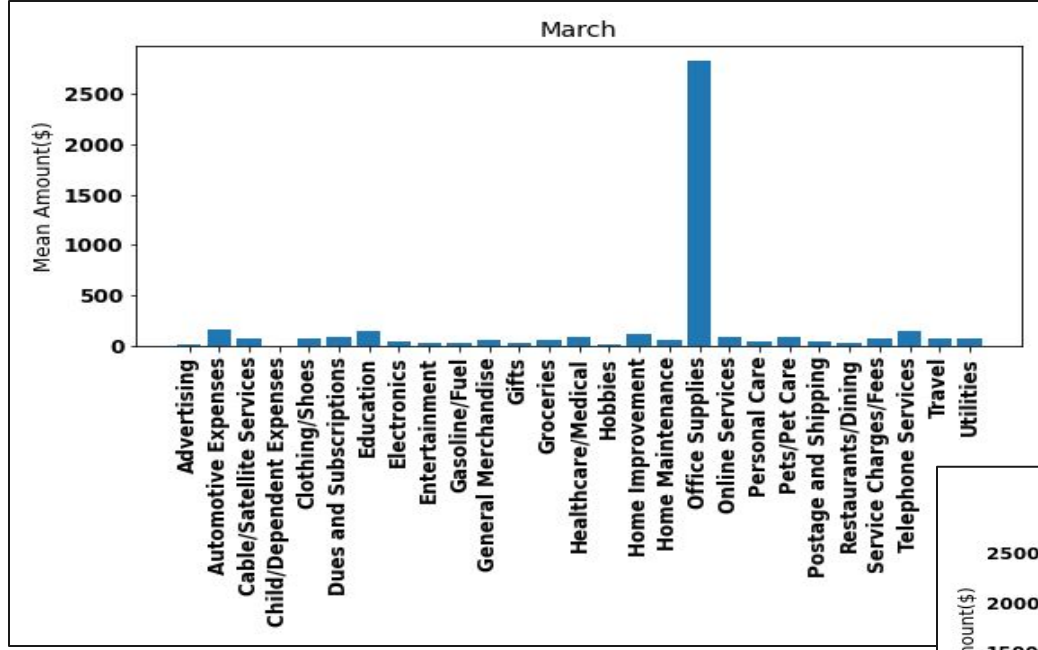


Users spent \$128K at Costco (highest amount in the General Merchandise category)

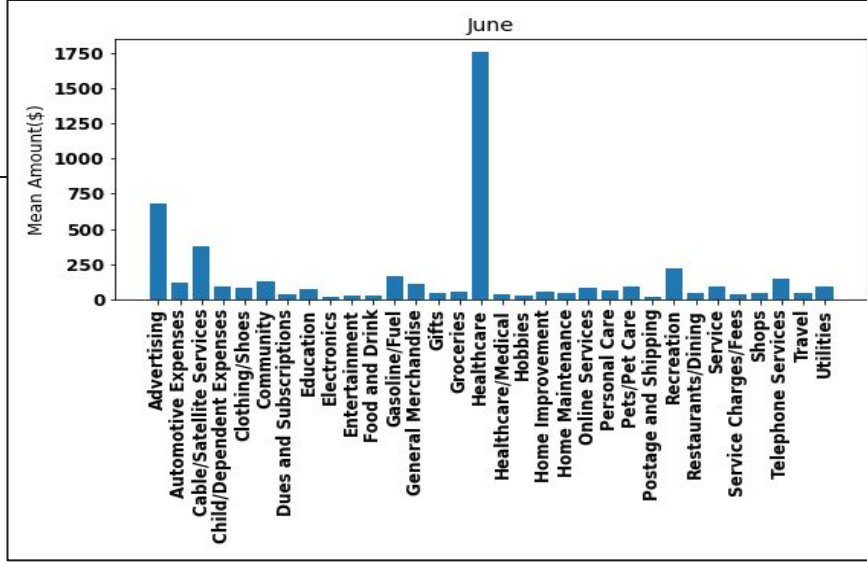
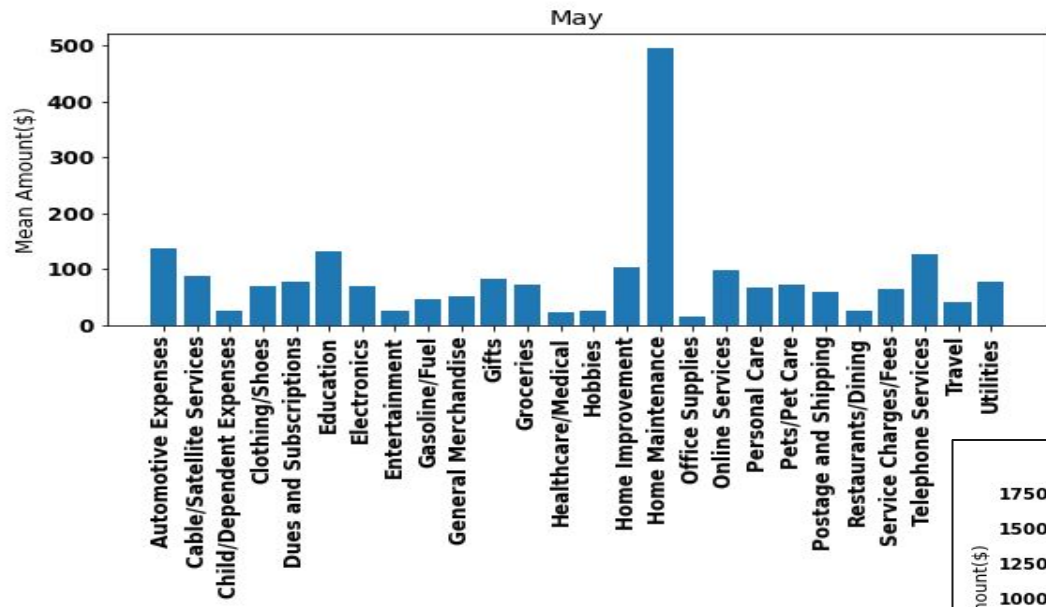
All the top names in this category are well known except for 'pool supply unlimited'.

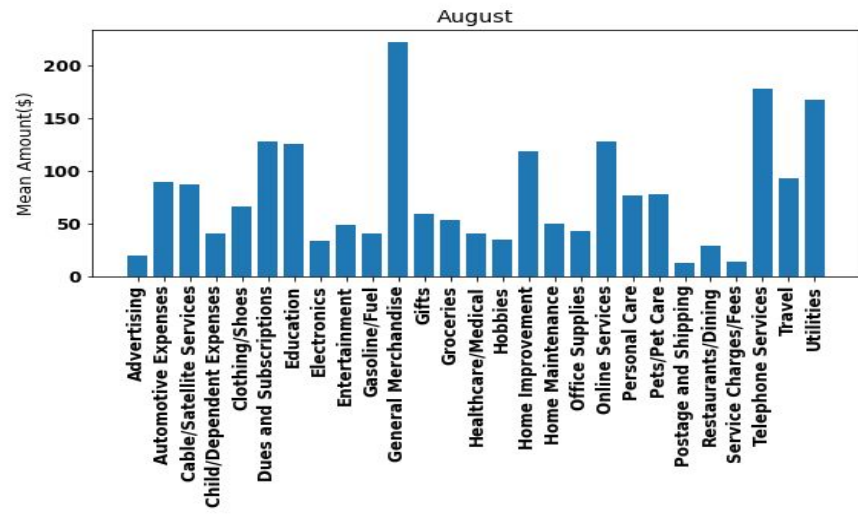
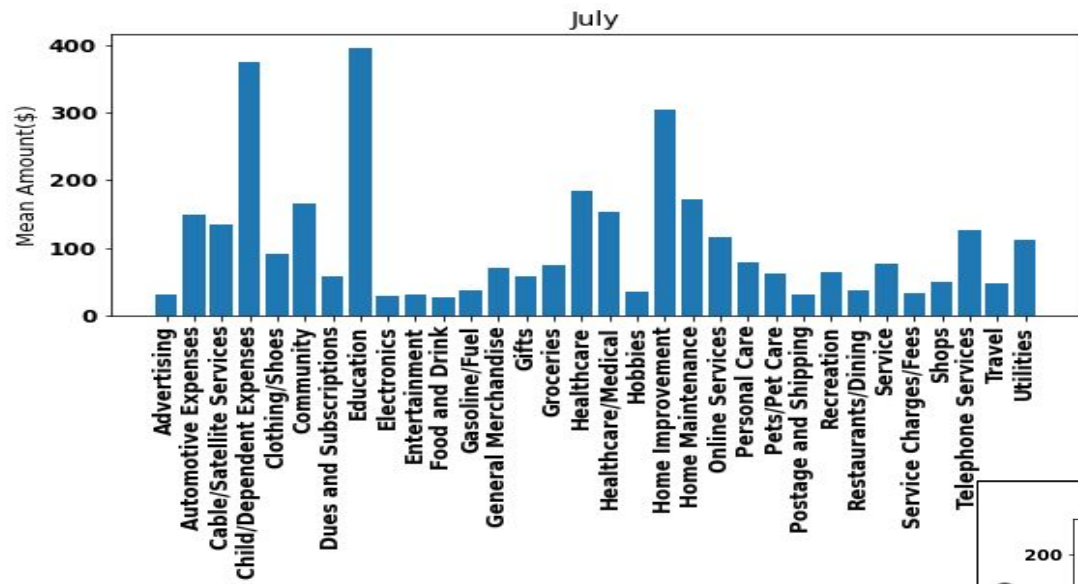
Upon investigations it was found that some high dollar amount in 'pool supply unlimited' transactions were duplicates.











# Pearson Correlation - Toy Data



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

## 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

# Cosine Similarity Score - Toy Data

	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

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

# Location Based Recomm\_System - Code for building the recomm\_df

```
In [34]: recomm_df = df2_co_topcat.copy()
recomm_df = (recomm_df.groupby(['cluster', 'merchant']).agg({'latitude' : 'first',
                                                             'longitude' : 'first',
                                                             'city' : 'first',
                                                             'category' : 'first',
                                                             'cluster' : 'count'}))
            .rename({'cluster' : 'cluster_count'},axis=1).reset_index()
            .sort_values(['cluster', 'cluster_count'], ascending = [True, False])
            .drop('cluster_count', axis=1))

recomm_df.head()
```

Out[34]:

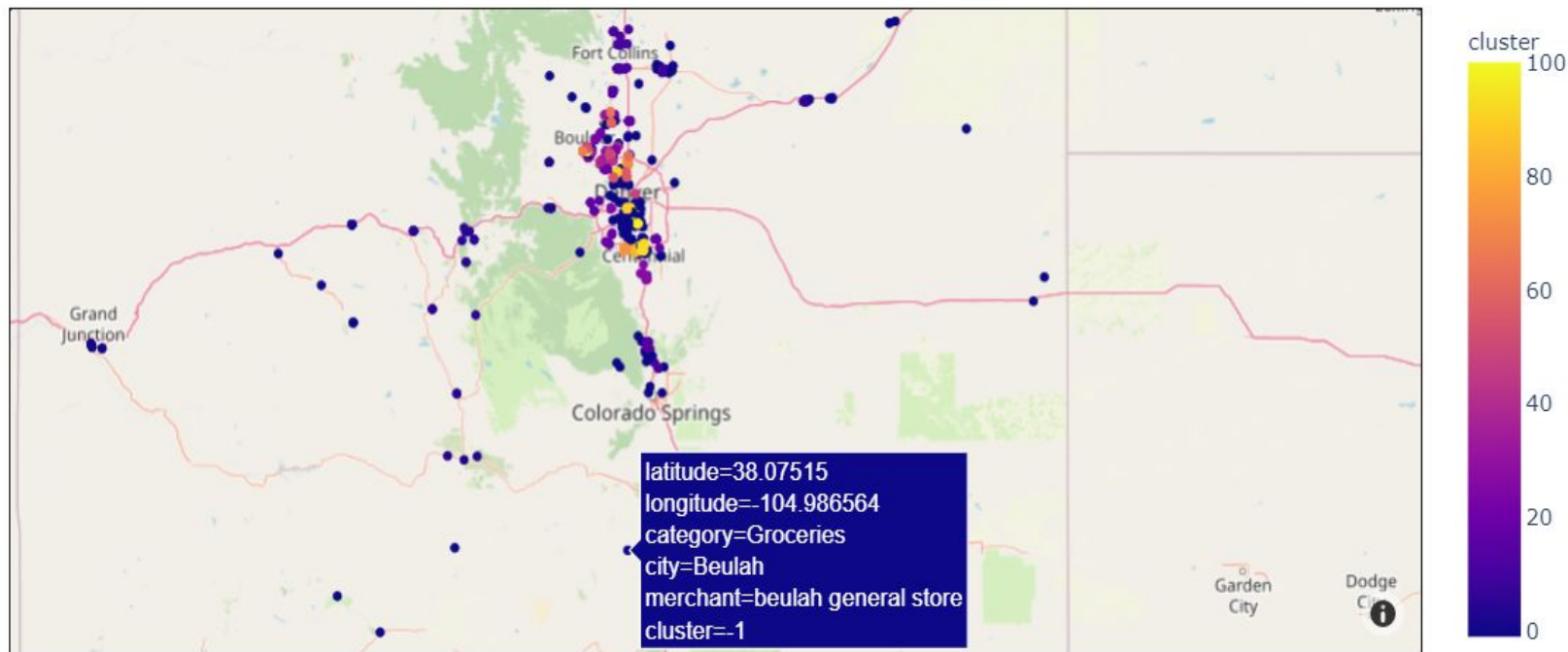
	cluster	merchant	latitude	longitude	city	category
2	-1	ace hardware	39.768741	-105.00495	Denver	Home Improvement
3	-1	altar'd state	40.017740	-105.25541	Boulder	Clothing/Shoes
4	-1	andy's frozen custard	39.646900	-105.02530	Sheridan	Restaurants/Dining
5	-1	arby's	39.596080	-104.88609	Greenwood Village	Restaurants/Dining
6	-1	barking dog cafe	40.224548	-105.27076	Lyons	Restaurants/Dining

# Location Based Recomm\_System - Code for Recommendation Engine

```
def recommend_co_merchants_hdb(df,lat,long,city,merchant,cluster_object):  
    # Predict the cluster for longitude and latitude provided  
    test_labels, strengths = hdbscan.approximate_predict(cluster_object, [[lat,long]])  
    predicted_cluster=test_labels[0]  
    print('Predicted cluster for this lat/long combination is: ' + str(predicted_cluster))  
    print("_____")  
  
    if predicted_cluster==-1:  
        return ('No merchants close by')  
    # Get the best merchant in this cluster  
    else:  
        pop_merch_recomm_df=(df[df['cluster']==predicted_cluster].iloc[0:5][['merchant','city','latitude','longitude']])  
        pop_merch_recomm_df=pop_merch_recomm_df.reset_index(drop=True)  
        mask = (pop_merch_recomm_df.merchant==merchant) & (pop_merch_recomm_df.latitude==lat) & \  
            (pop_merch_recomm_df.longitude==long)  
        print ('Since you are currently in ' + city.capitalize() + ' ' + 'at ' + \  
            merchant.capitalize() + ', how about you visit these merchants around this area? ')  
    return pop_merch_recomm_df[~mask]
```

## Visualization of data with noise present

HDBScan-lat-long-TopCat





## Visualization of data without noise

HDBScan-lat-long-TopCat\_Minus\_Noise

